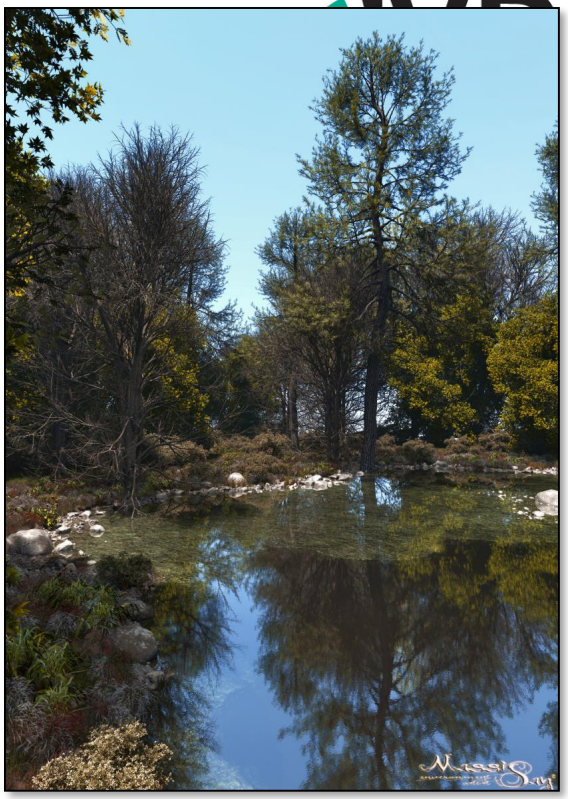
Computergrafik

Vorlesung im Wintersemester 2024/25
Kapitel 8: Prozedurale Modelle, Texturen & Geometrie

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

Überblick



Motivation: prozedurale Modellierung

- Erzeugung von großen Szenen/viel Content
- kompakte(re) Beschreibung statt Speicherung
- kein Übertragen/Streaming großer Daten:
  Berechnung „am Ort“, ggf. zur Laufzeit
  (z.B. auch auf der GPU)

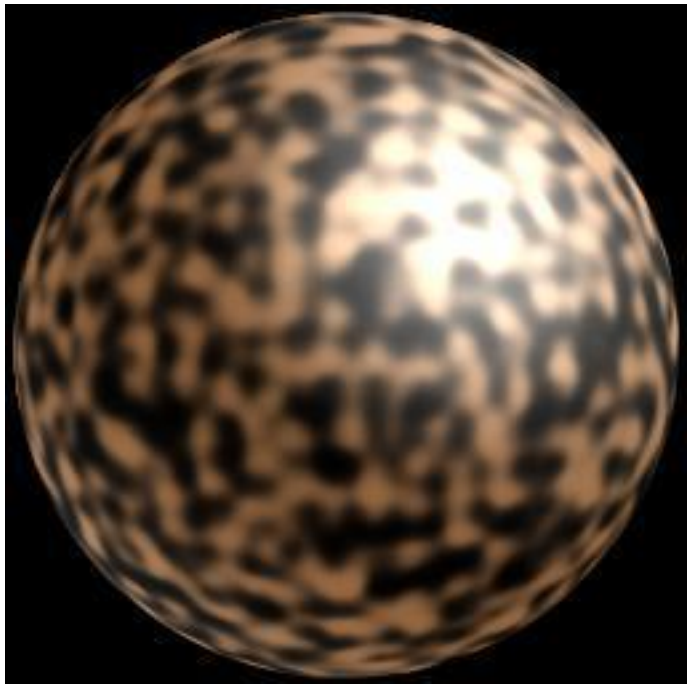
Überblick

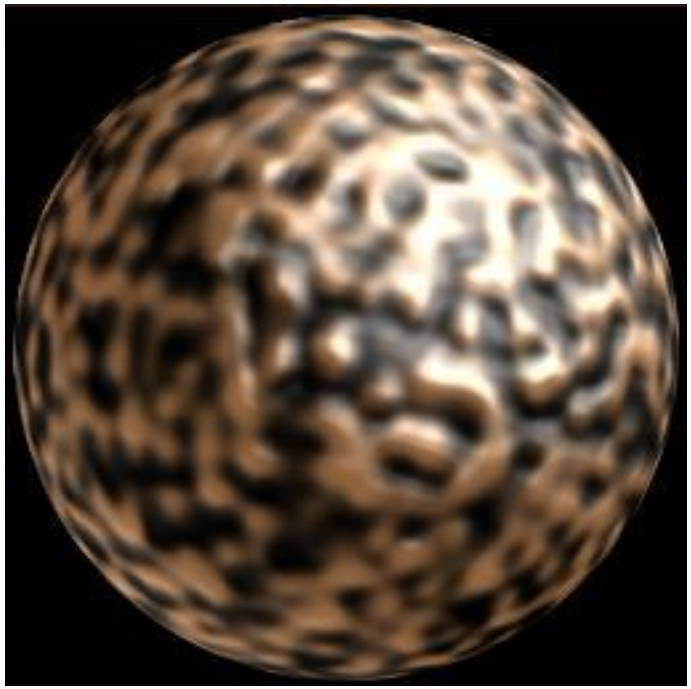
- Rauschen, Turbulenz und prozedurale Texturen
  - nachbilden von natürlichen Phänomenen, aber Zufall („Rauschen“) statt phys. Simulation, z.B. Wolken, Wasseroberfläche, Terrain, …
  - Rausch-/Noise Texturen: Rauschen ist ein Signal mit bestimmter Energie in einem Frequenzbereich → „spektrale Synthese“
- Hypertextures und Distanzfelder
- Textursynthese
- L-Systeme

Rauschen und Rauschtexturen

Noise-Funktionen nach Ken Perlin (1985)

- …*die* Grundlage für stoch. Modellierung/proz. Texturen
- die Basis bilden **Pseudo-Zufallszahlen** aus denen wir Bilder mit den gewünschten 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

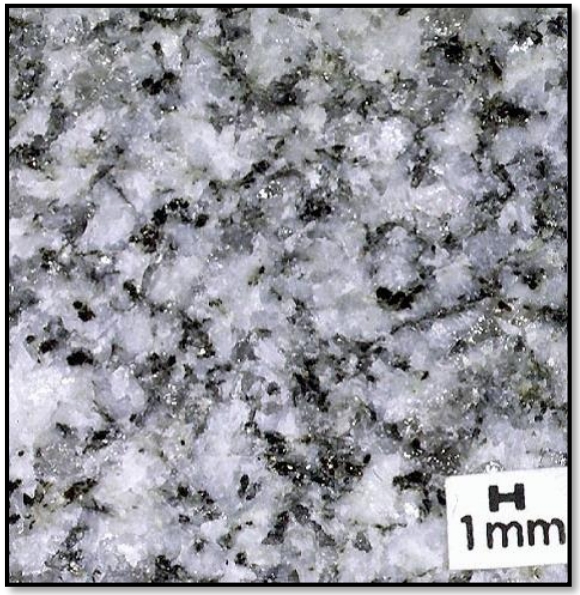




Rauschtexturen und Strukturen

Motivation

- natürliche Objekte zeigen stochastische Variationen – auf unterschiedlichen Längenskalen – bei räumlicher Kohärenz
- versuche solche zufälligen Strukturen aus Rauschtexturen/-funktionen mit unterschiedlichen Frequenzspektren zu berechnen (Turbulenz)

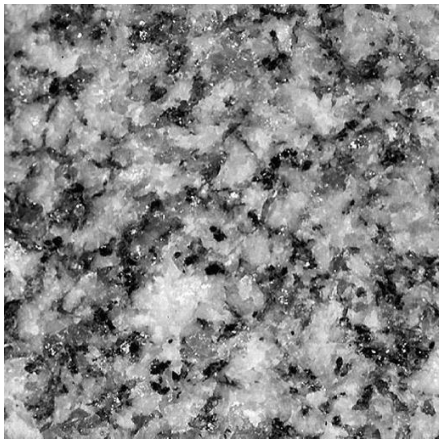


Granit

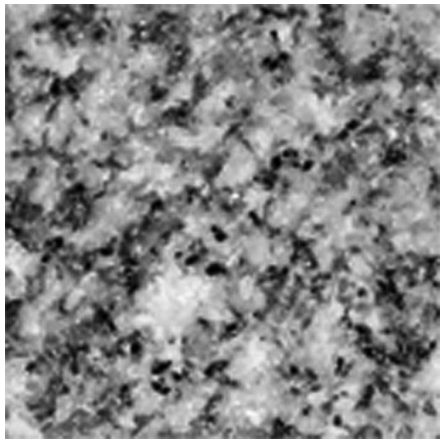
Marmor

Rauschtexturen und Strukturen

Motivation: Auflösungspyramide Granittextur

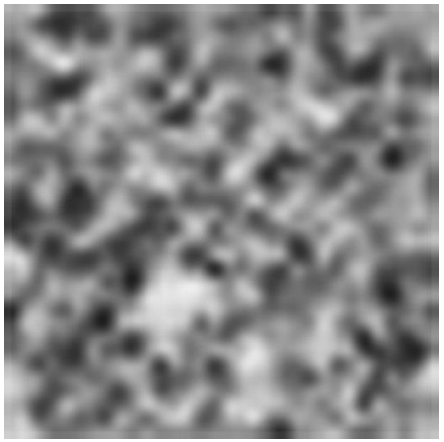


512^2

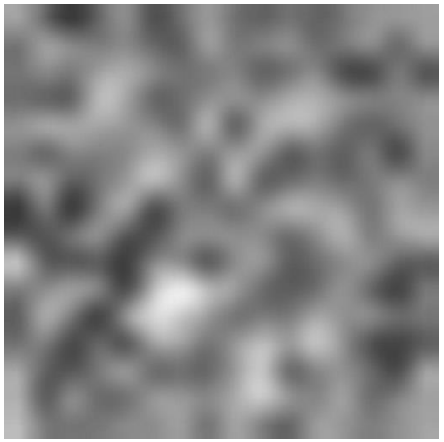


256^2

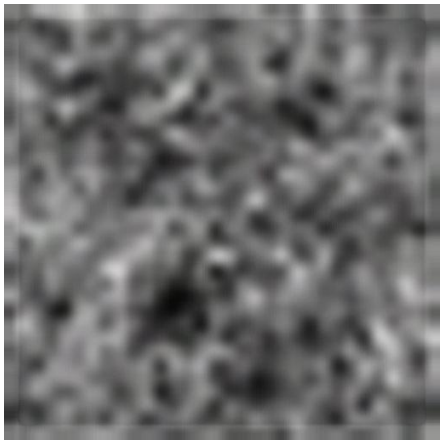
...



32^2



16^2

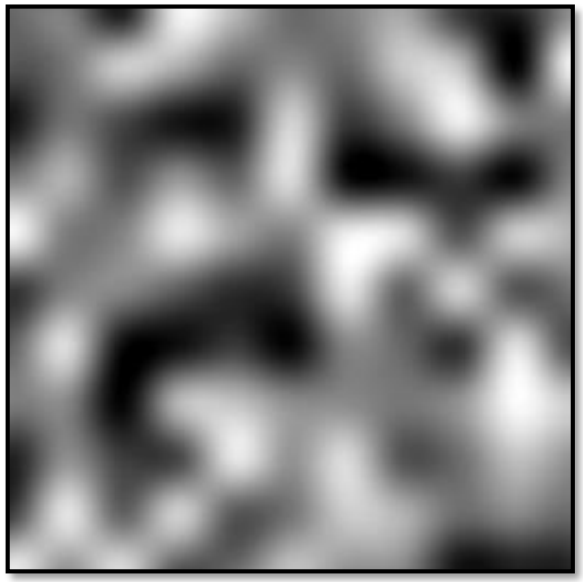


Differenz 32^2 und 16^2

Rauschen und Rauschtexturen

Eigenschaften/Anforderungen

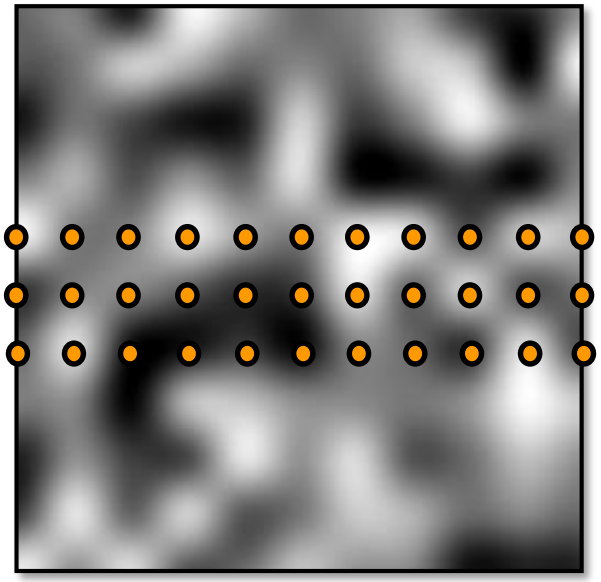
- Rausch-/Noise-Funktion $n(\mathbf{x}) = c$ mit $\mathbf{x} \in \mathbb{R}^n$, $c \in \mathbb{R}$
  - reproduzierbar: $n(\mathbf{x})$ liefert für jede Auswertung bei gleichem $\mathbf{x}$ dasselbe Resultat
  - keine Periodizität (zumindest nicht sichtbar)
  - begrenzter Wertebereich (Abb. auf Farben etc.)
  - definierte Frequenzverteilung, bandlimitiert (Aliasing reduzieren)
  - Stetigkeit, räumliche Korrelation: $n(\mathbf{x}) \approx n(\mathbf{x} + \boldsymbol{\epsilon})$ (hängt mit Bandbegrenzung zusammen)

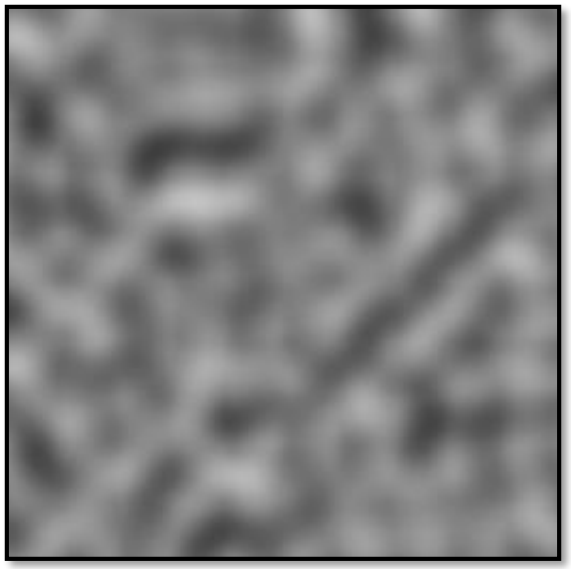


Rauschen und Rauschtexturen

Noise-Funktionen nach Ken Perlin (1985)

- …*die* Grundlage für stochastische Modellierung/prozedurale Texturen
- die Basis bilden **Pseudo-Zufallszahlen** aus denen wir Bilder mit den gewünschten Eigenschaften erzeugen
- es gibt zwei häufig verwendete Klassen von Noise-Funktionen:
  - Lattice Value Noise (lattice value = Werte auf einem 1D/2D/3D-Gitter)
  - Lattice Gradient Noise (Grad. statt Absolutwerte, nicht i.d. Vorlesung)

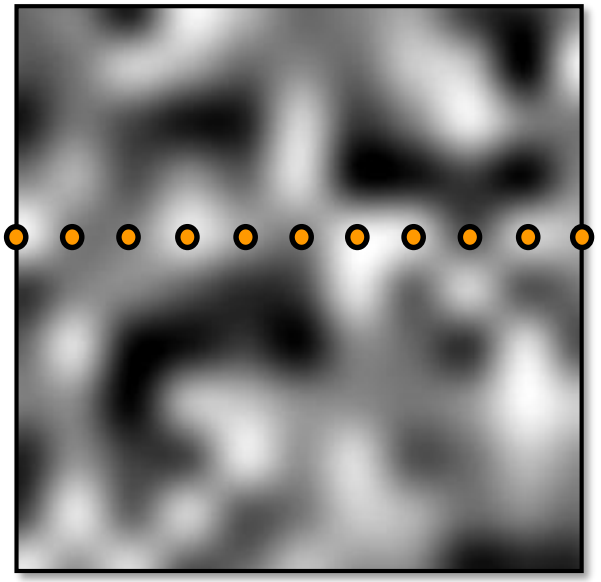


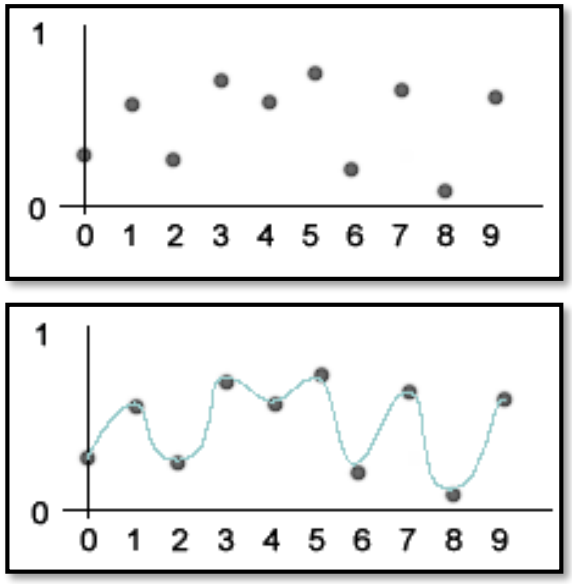


Rauschen und Rauschtexturen

Noise-Funktionen nach Ken Perlin (1985)

- …*die* Grundlage für stochastische Modellierung/prozedurale Texturen
- die Basis bilden **Pseudo-Zufallszahlen** aus denen wir Bilder mit den gewünschten Eigenschaften erzeugen
- es gibt zwei häufig verwendete Klassen von Noise-Funktionen:
  - Lattice Value Noise (lattice value = Werte auf einem 1D/2D/3D-Gitter)
  - Lattice Gradient Noise (Grad. statt Absolutwerte, nicht i.d. Vorlesung)



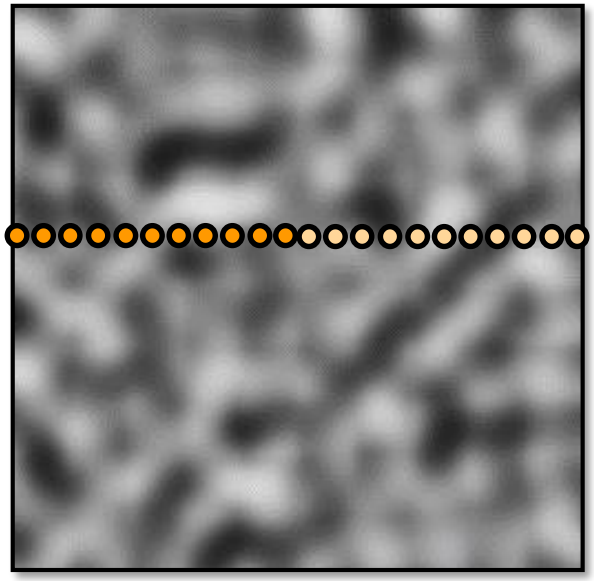


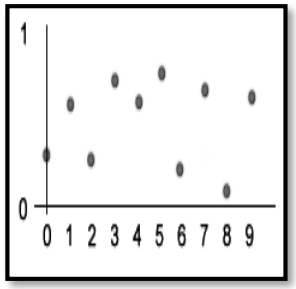
$n(\mathrm{x}) \in [0;1]$

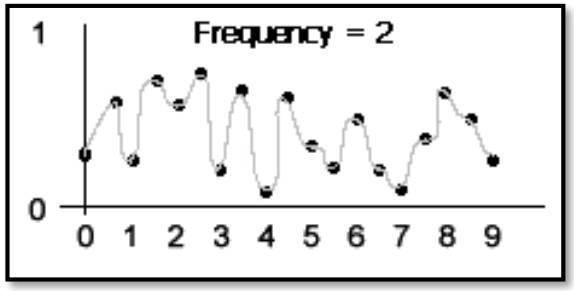
Rauschen und Rauschtexturen

Noise-Funktionen nach Ken Perlin (1985)

- …*die* Grundlage für stochastische Modellierung/prozedurale Texturen
- die Basis bilden **Pseudo-Zufallszahlen** aus denen wir Bilder mit den gewünschten Eigenschaften erzeugen
- es gibt zwei häufig verwendete Klassen von Noise-Funktionen:
  - Lattice Value Noise (lattice value = Werte auf einem 1D/2D/3D-Gitter)
  - Lattice Gradient Noise (Grad. statt Absolutwerte, nicht i.d. Vorlesung)





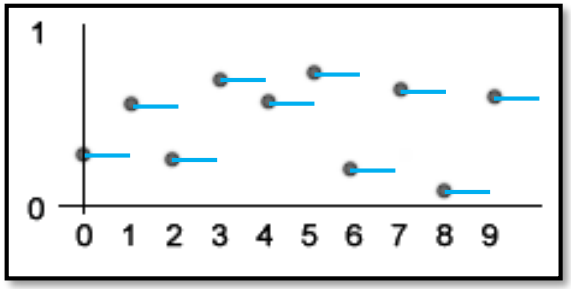


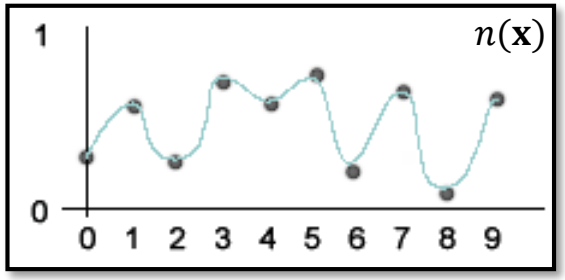
$n(2x) \in [0;1]$

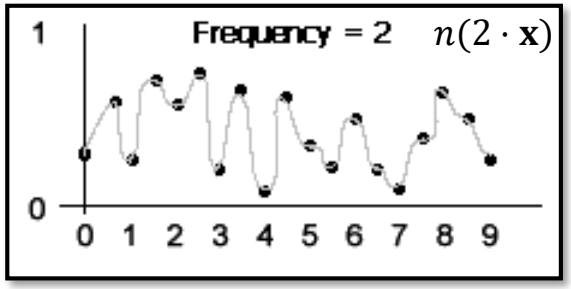
Lattice Value Noise

(Pseudo-)Zufallszahlen auf einem Gitter

- Abbildung $\mathbf{x} = (x, y, z) \in \mathbb{R}^3 \longmapsto Z \in [0; 1]$ mit
  $Z(\mathbf{x}) \coloneqq random(\lfloor x \rfloor, \lfloor y \rfloor, \lfloor z \rfloor)$, 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 $f_x = x - \lfloor x \rfloor$
    - in 2D/3D: bi-/trilineare Interpolation aus $Z(\mathbf{x})$, **besser bi-/trikubisch**
- unterschiedlicher Abstand liefert anderes Frequenzspektrum
  - z.B. $n(2 \cdot \mathbf{x})$ hat doppelt so hohe Frequenzen wie $n(\mathbf{x})$



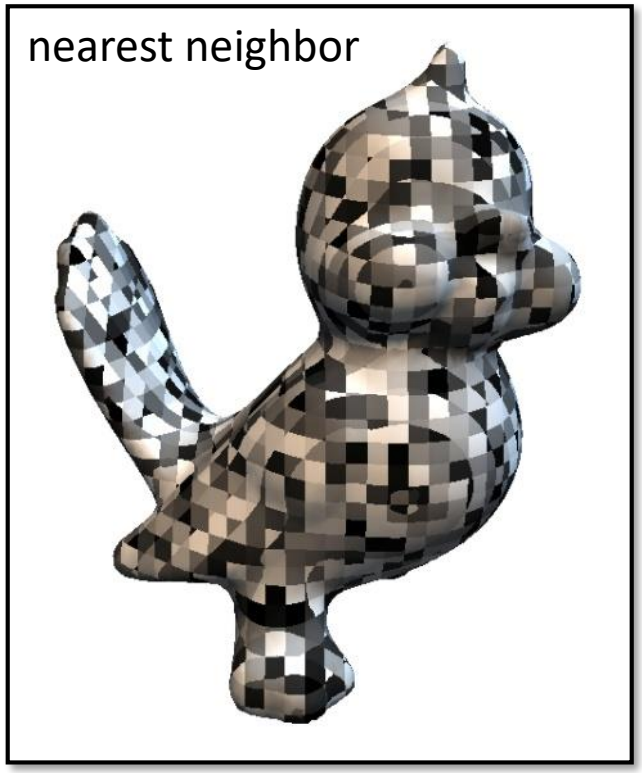


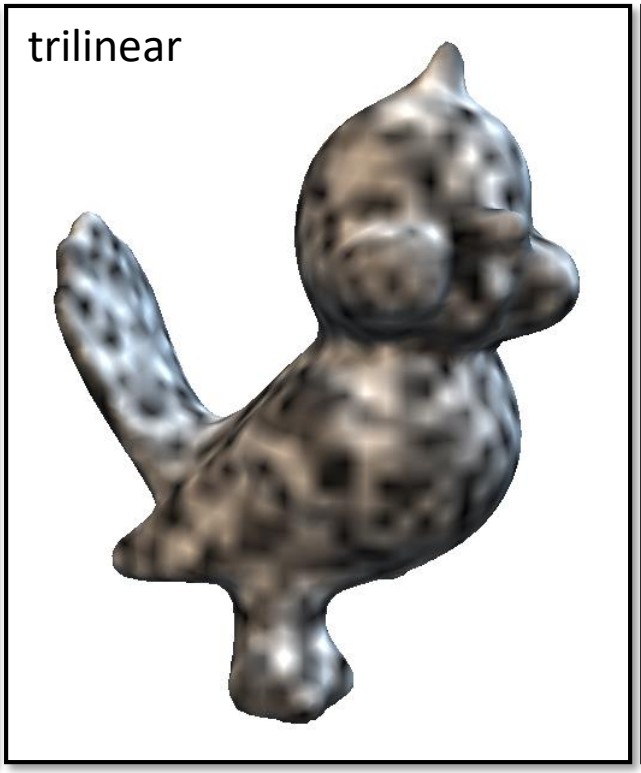


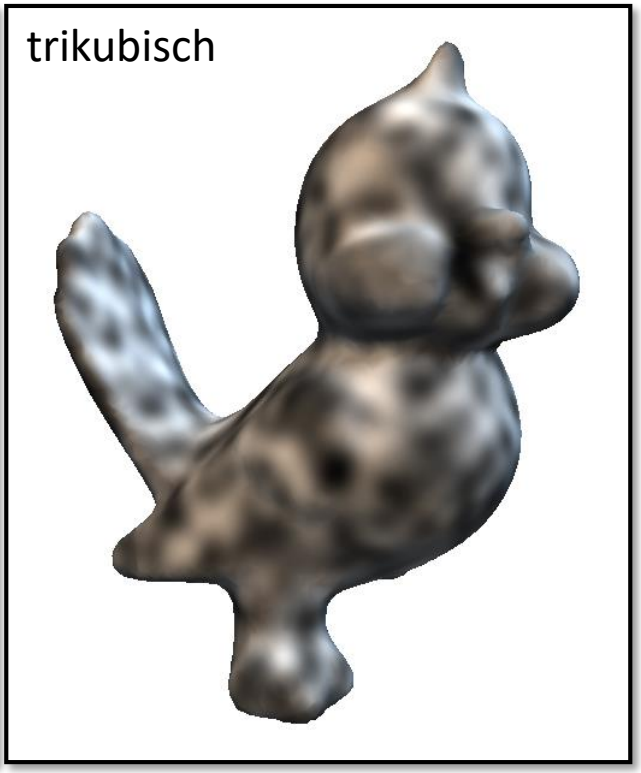
Lattice Value Noise

Beispiel: Interpolation einer 3D-Noise-Funktion

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







Lattice Value Noise

(Pseudo-)Zufallszahlen auf einem Gitter in der Praxis

- **naiver Ansatz**: erzeuge großes 2D/3D Gitter gefüllt mit Zufallszahlen
  - $Z(\mathbf{x})$ gibt Werte aus dem Gitter zurück
  - ist natürlich unpraktisch wg. großer Datenmenge und Periodizität

- **eleganter**: erzeuge einmalig ein 1D-Array mit Zufallszahlen aus $[0; 1]$
  - z.B. $\text{float } rtab[256];$
  - verwende Hash-Funktion um darauf zuzugreifen:

    $Index(\lfloor x \rfloor, \lfloor y \rfloor, \lfloor z \rfloor) := P\left(\lfloor x \rfloor + P\left(\lfloor y \rfloor + P(\lfloor z \rfloor)\right)\right)$
  - $P(.)$ ist eine Permutation (z.B. bijektive Abbildung von i auf j mit $i, j \in [0{,}255]$, d.h. $\lfloor . \rfloor$ wird skaliert), vermeidet wiederholende Muster

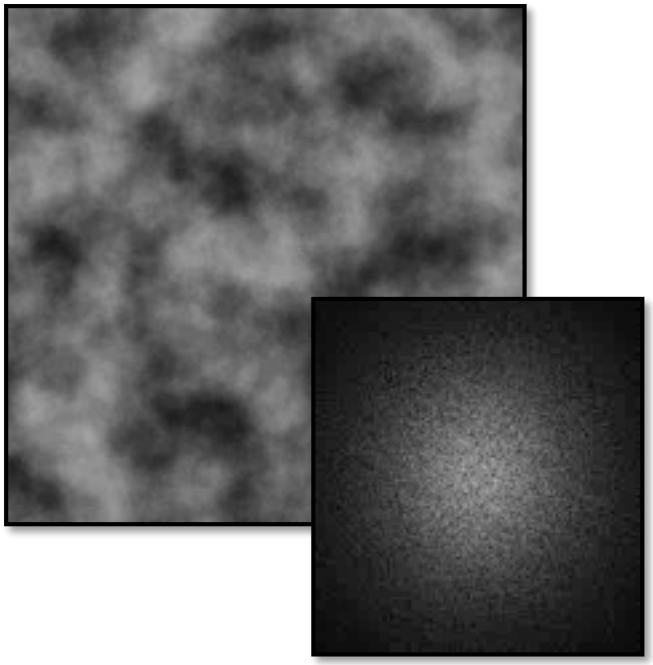
  - **ergibt die Zufallszahlen für die Noise-Funktion**:

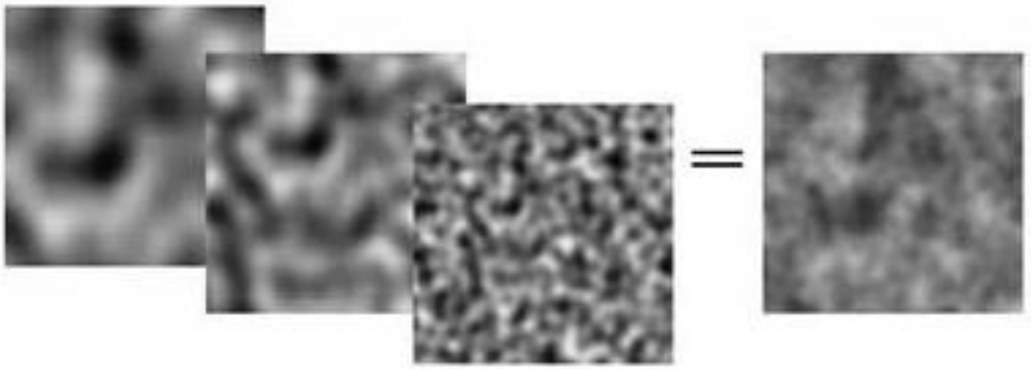
    $Z(\mathbf{x}) := rtab[\, Index(\lfloor x \rfloor, \lfloor y \rfloor, \lfloor z \rfloor)\,]$ mit $\mathbf{x} = (x, y, z)$

Rauschen und Turbulenz

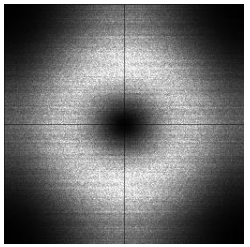
Spektrale Synthese: Kombination unterschiedlicher Frequenzbereiche

- $\text{turbulence}(\mathbf{x}) = \sum_k \left({}^1/_f\right)^k n\left(f^k \cdot \mathbf{x}\right)$
  - Aufsummieren von k Oktaven, mit Amplitudenskalierung $\left({}^1/_f\right)^k$
  - höhere Oktaven erzeugen höherfrequente Anteile (= feinere Strukturen), deren Amplitude wird aber geringer
  - hier: Frequenzspektrum ${}^1/_f$ (mit $f = 2$ erzeugt man „rosa Rauschen“)



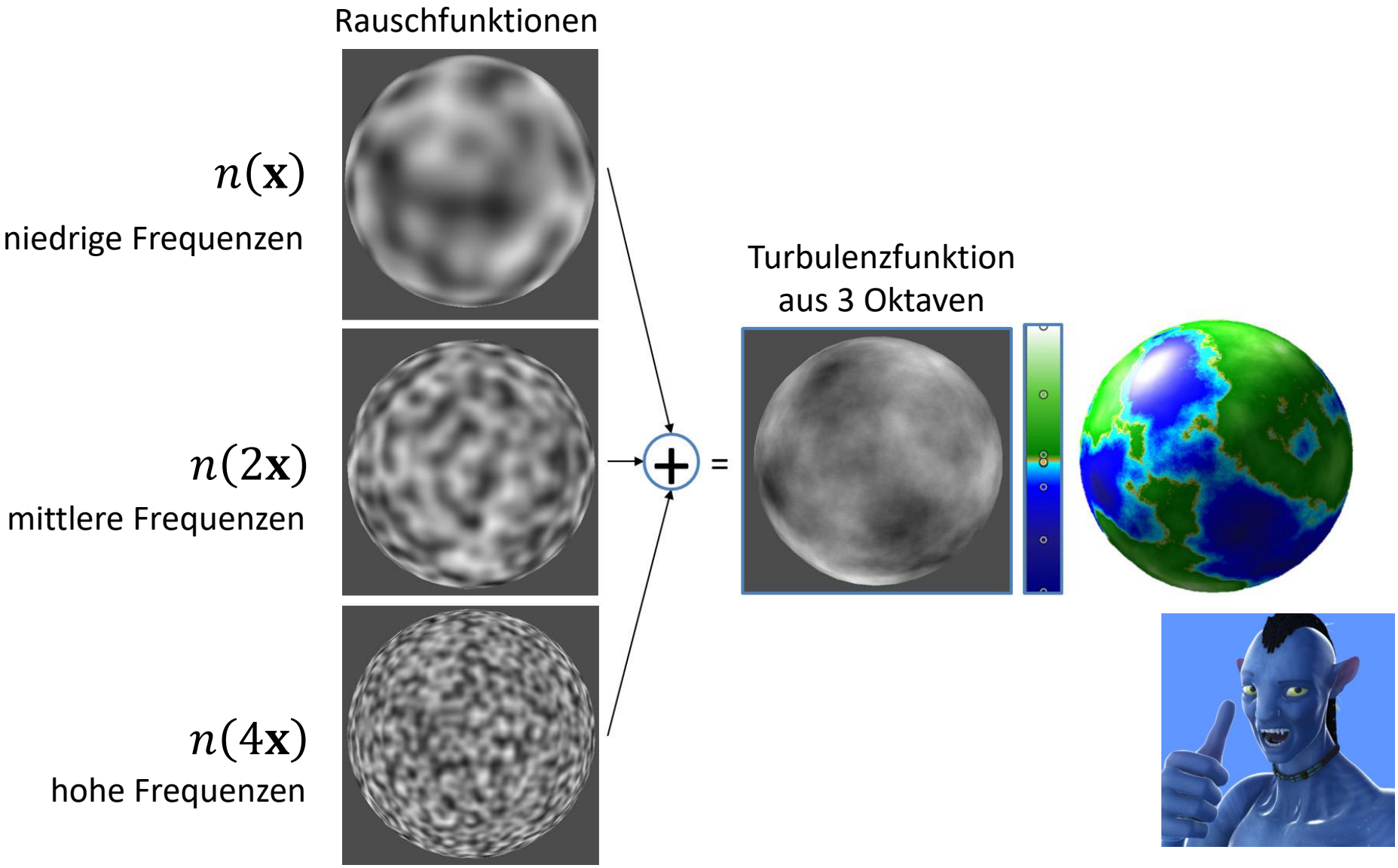


Frequenzspektrum einer Oktave (idealisiert):



Rauschen und Turbulenz

- Abbildung der Rausch-/Turbulenzfkt. häufig mittels einer Farbtabelle



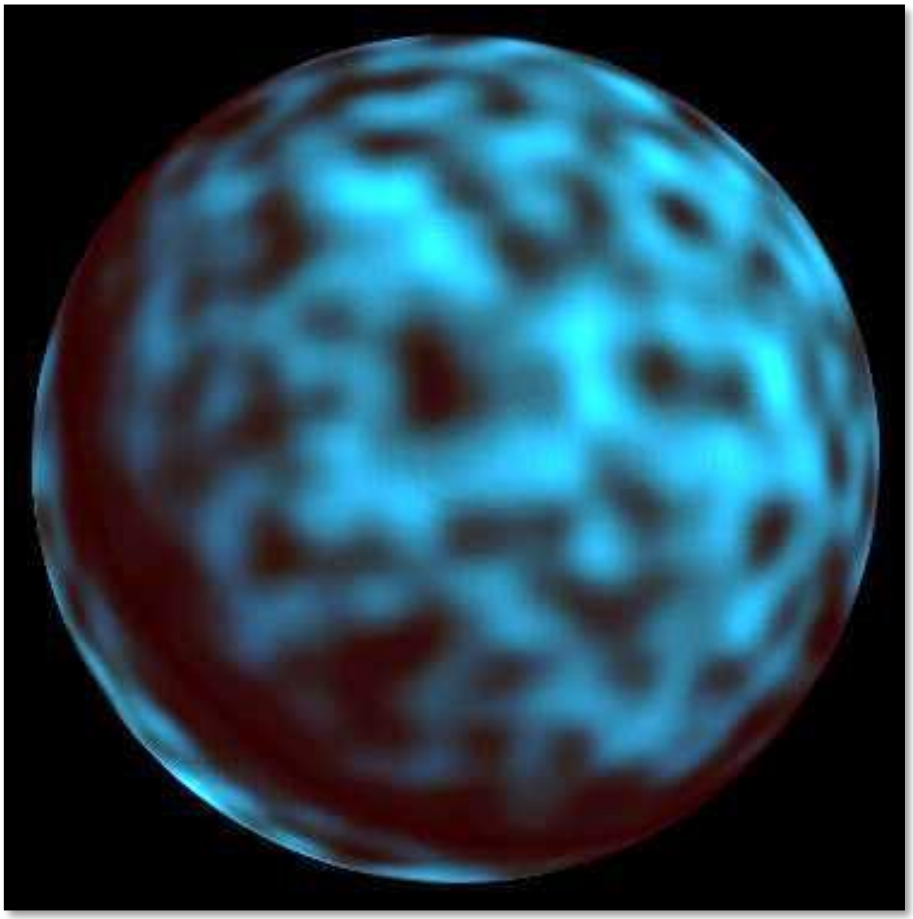
Rauschen und Turbulenz

Beispiel

- einzelne Noise-Funktion (links) und Turbulenz mit k Oktaven (rechts)

$n(\mathbf{x})$

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



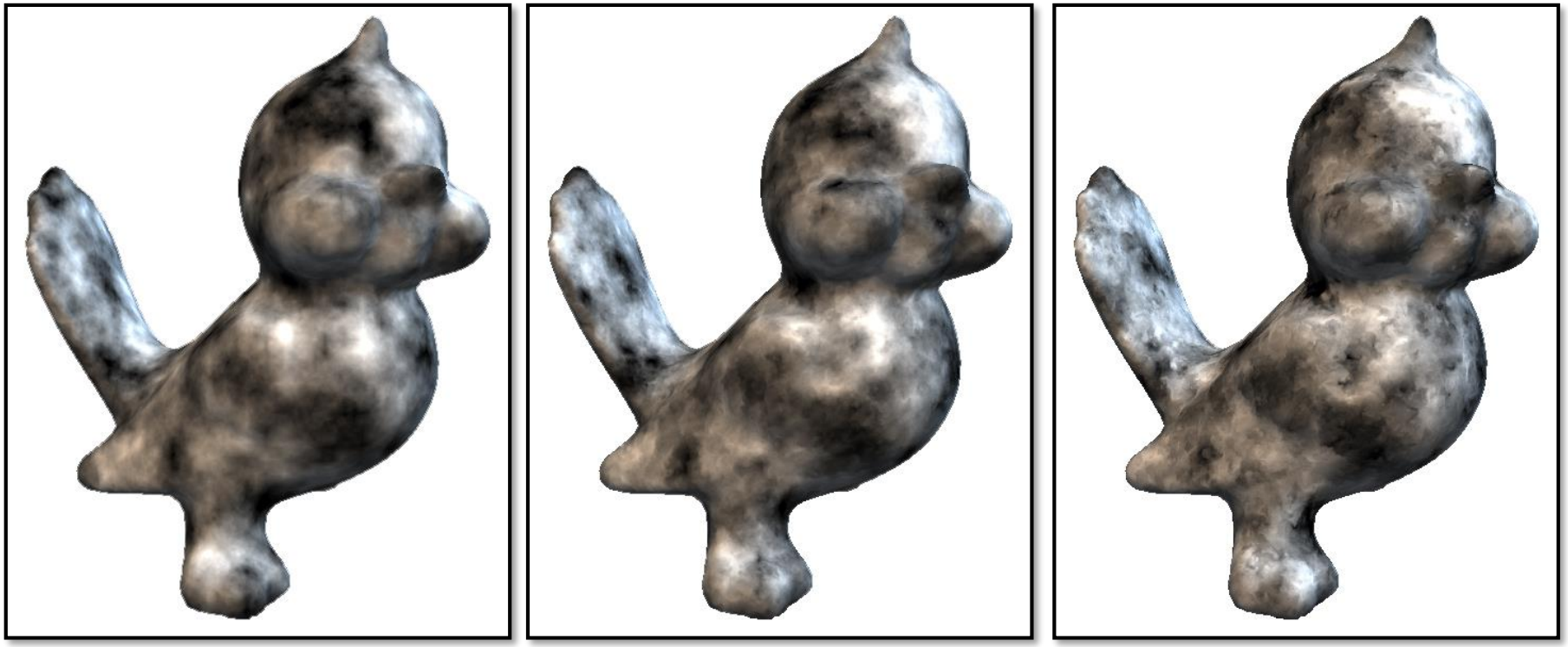
Turbulenz mit Strukturen (und Abb. auf Farbe)

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

$$\sum_k \left({}^1/_f\right)^k \left|2n(f^k\mathbf{x}) - 1\right| \qquad \sin\left(x + \sum_k \left({}^1/_f\right)^k n(f^k\mathbf{x})\right)$$

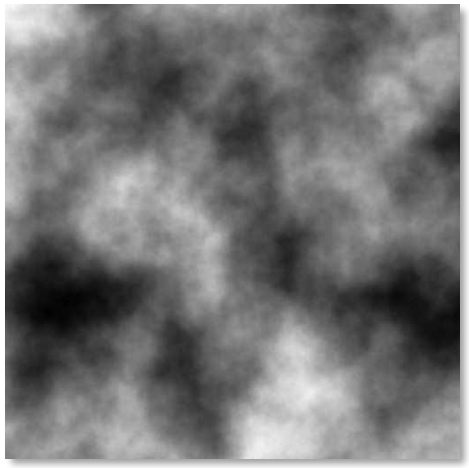
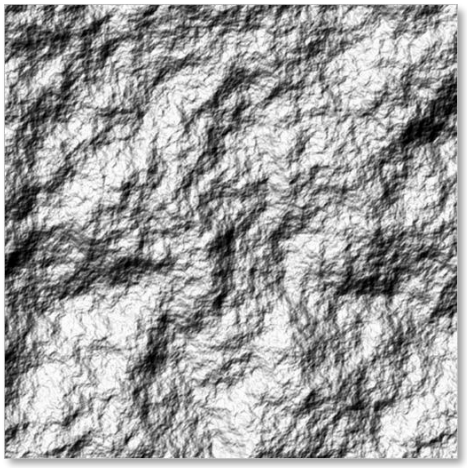
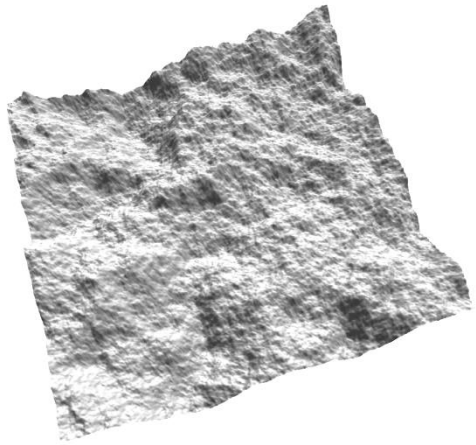
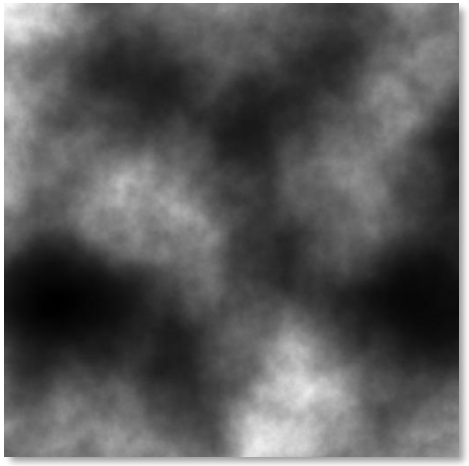
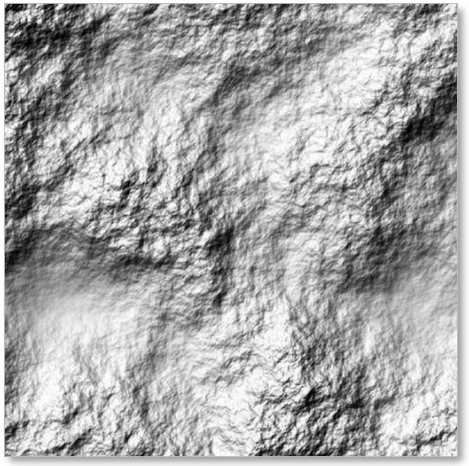
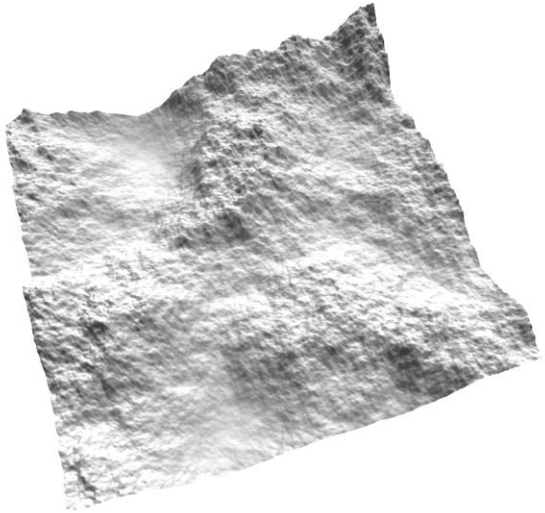
Kombination mehrerer Turbulenz-Texturen

- Bsp. ineinander verschachtelte Turbulenzfunktionen („Marmor“)
  - links: $\mathrm{turbulence}(\mathbf{x}) = \sum_k \left({}^1/_2\right)^k \left| n\left(2^k \mathbf{x}\right)\right|$
  - mitte: $\mathrm{turbulence}(\mathbf{x} + {}^1/_2\, \mathrm{turbulence}(\mathbf{x}))$
  - rechts: $\mathrm{turbulence}(\mathbf{x} + \mathrm{turbulence}(\mathbf{x}))$



Beispiele: Höhenfelder aus Noise-Funktionen

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

Prozedurale Landschaften

- Höhenfelder und Texturen generiert aus Noise-Funktionen

Bilder: http://www.decarpentier.nl/scape-render

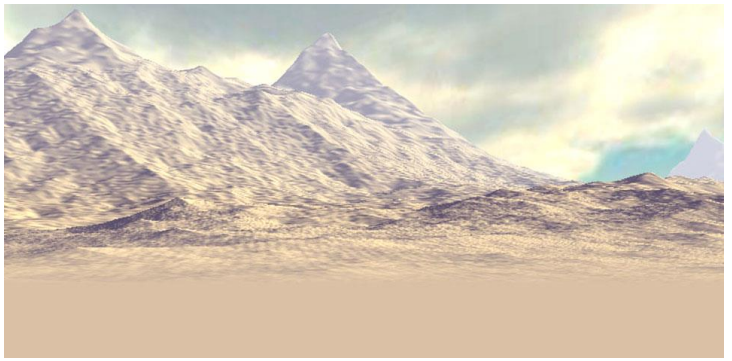
Prozedurale Landschaften

- Höhenfelder und Texturen generiert aus Noise-Funktionen
  … nicht in diesem Beispiel ;-)

Northern Territory, Australien (Foto: www.spiegel.de)

Prozedurale Landschaften

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



- keine Textur

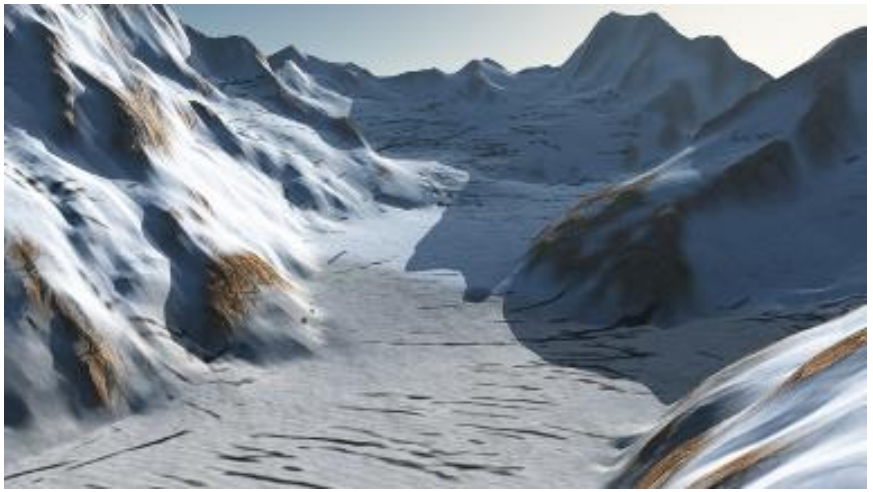
- Texturierung abhängig von Höhe $h(\mathbf{x})$
  $if\ (h(\mathbf{x}) > 1000m)\ color = white;$

- Textur mit Steigung $s(\mathbf{x})$ und Noise
  $$if \begin{pmatrix} \big(h(\mathbf{x}) + n_1(\mathbf{x})\big)\ in\ [h_{min}; h_{max}] \wedge \\ \big(s(\mathbf{x}) + n_2(\mathbf{x})\big)\ in\ [s_{min}; s_{max}] \end{pmatrix}$$
  $$color = green;$$

Bilder: http://www.decarpentier.nl/scape-render

Aktuelle Forschung

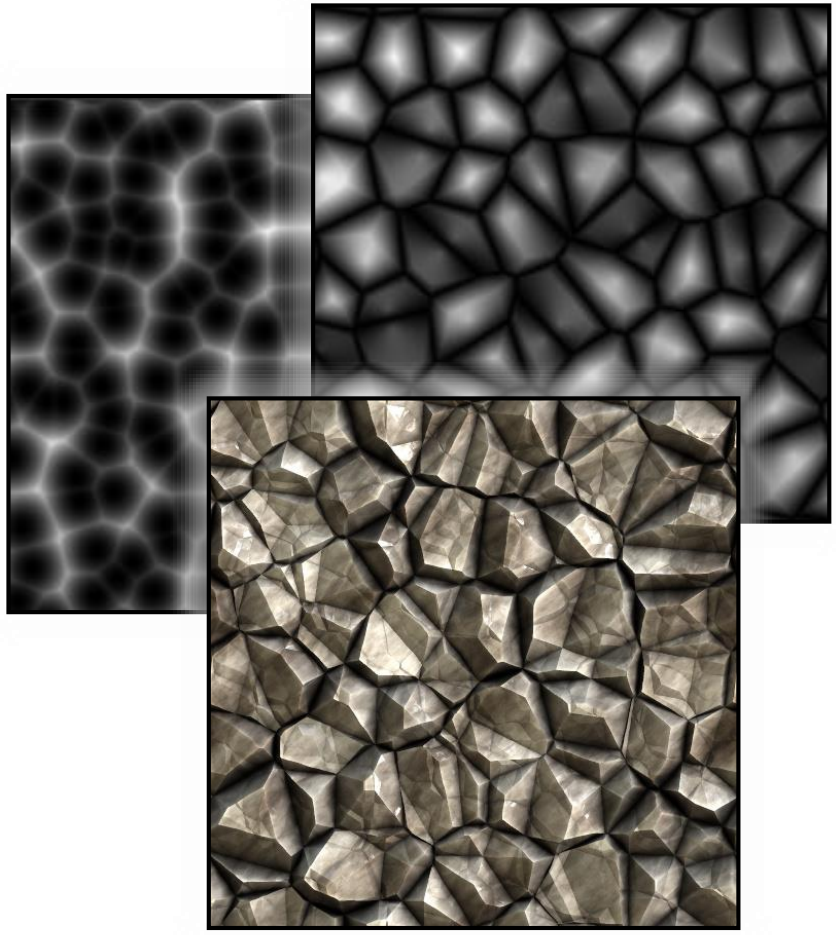
- Höhlen, Gletscher, Tektonik, Wolken, Flussläufe, …
  Bilder: Eric Galin (https://perso.liris.cnrs.fr/eric.galin)



State of the Art Report: https://hal.archives-ouvertes.fr/hal-02097510/file/A%20Review%20of%20Digital%20Terrain%20Modeling.pdf

Beispiele

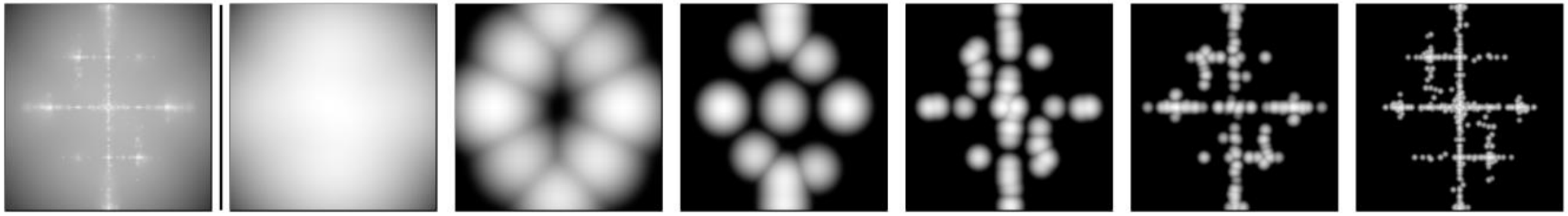
- viele weitere Beispiele und mehr Infos:
  „Texturing and Modeling, A Procedural Approach” (687 Seiten!)
  David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, Steve Worley

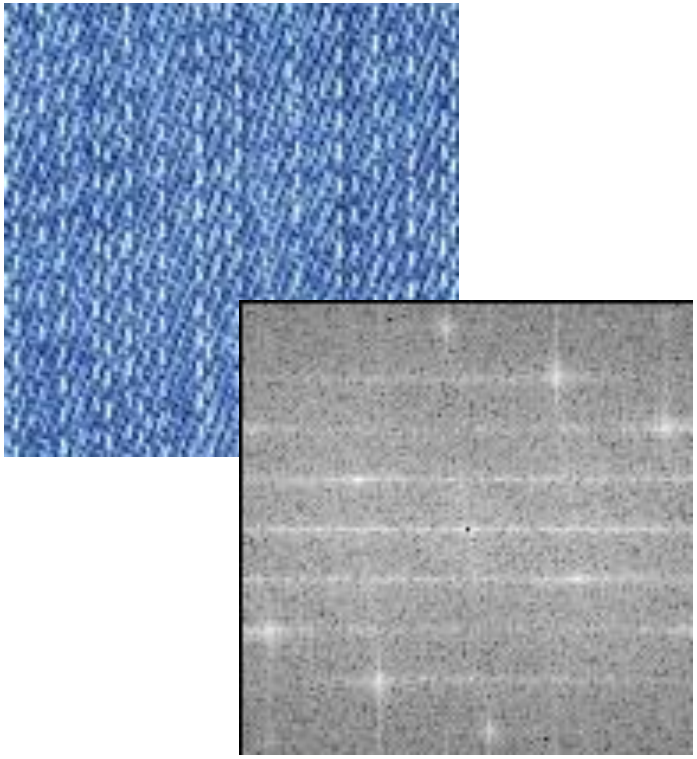


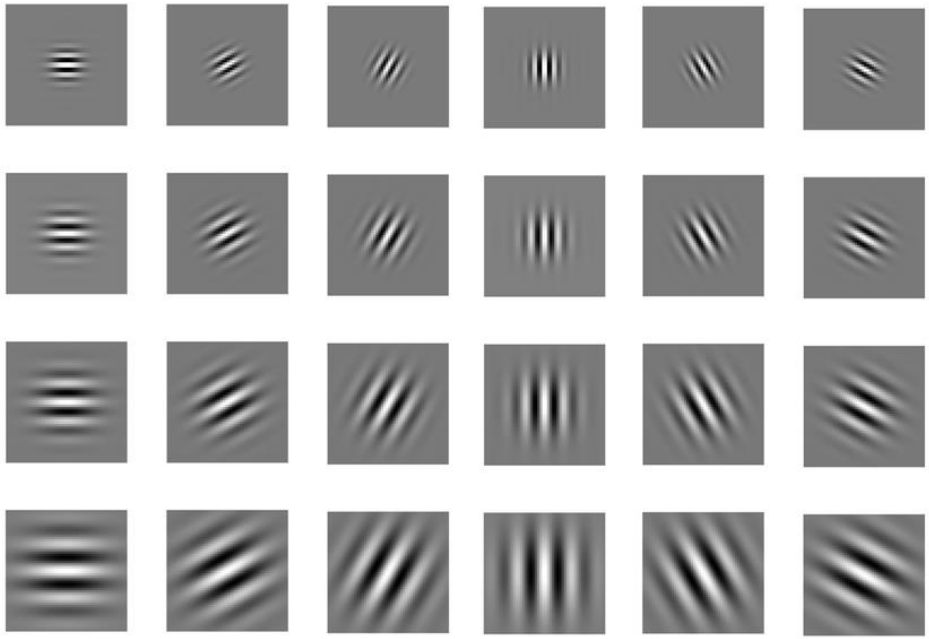
Worley‘s Cellular Textures

Gabor Noise by Example

- gezielte Reproduktion eines Frequenzspektrums mit Gabor-Noise
  https://graphics.cs.kuleuven.be/publications/GLLD12GNBE/2



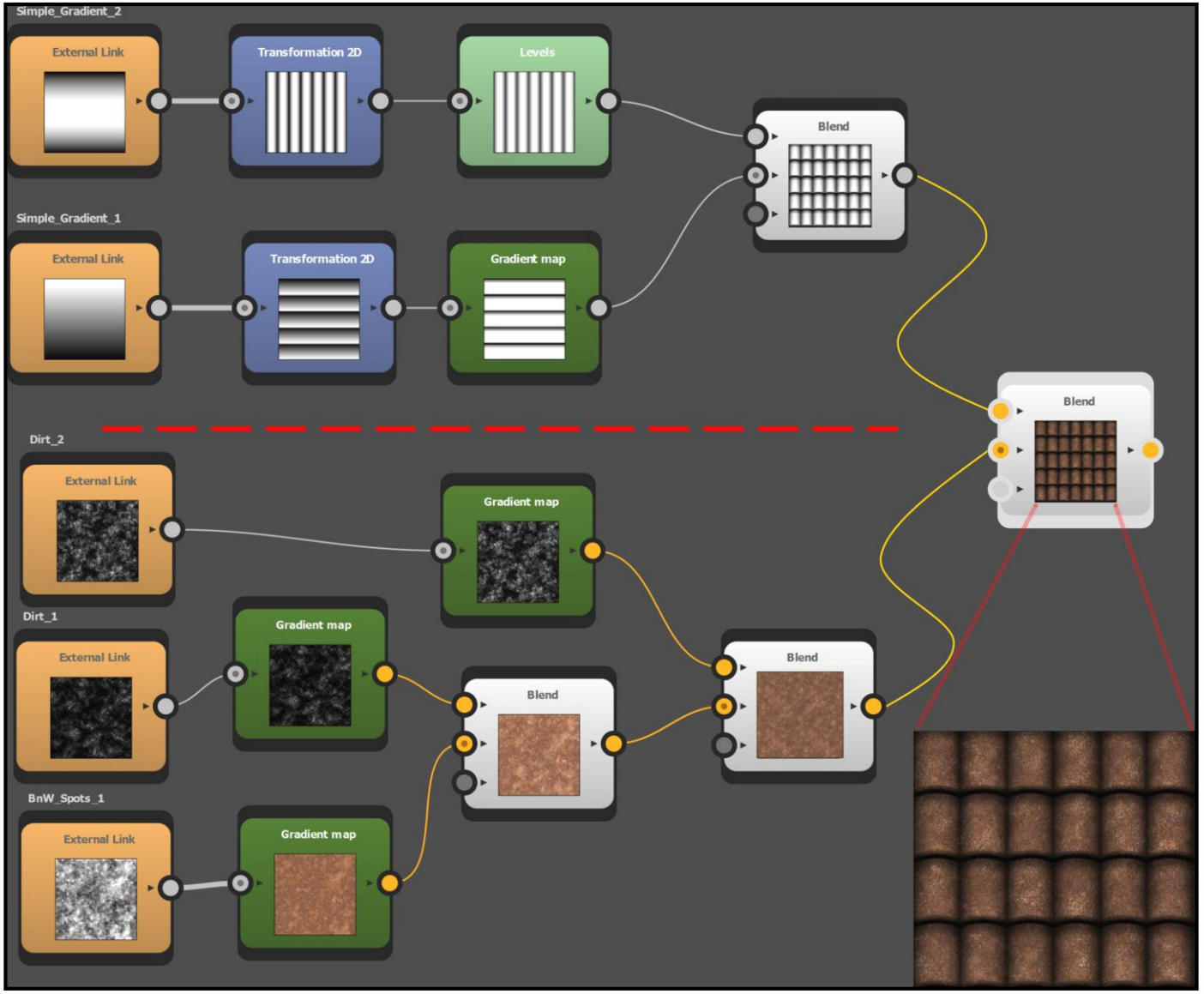




https://www.researchgate.net/figure/Gabor-Kernel-using-4-Scales-and-6-orientations_fig1_281995750

Ausblick: prozedurale Texturen in der Praxis

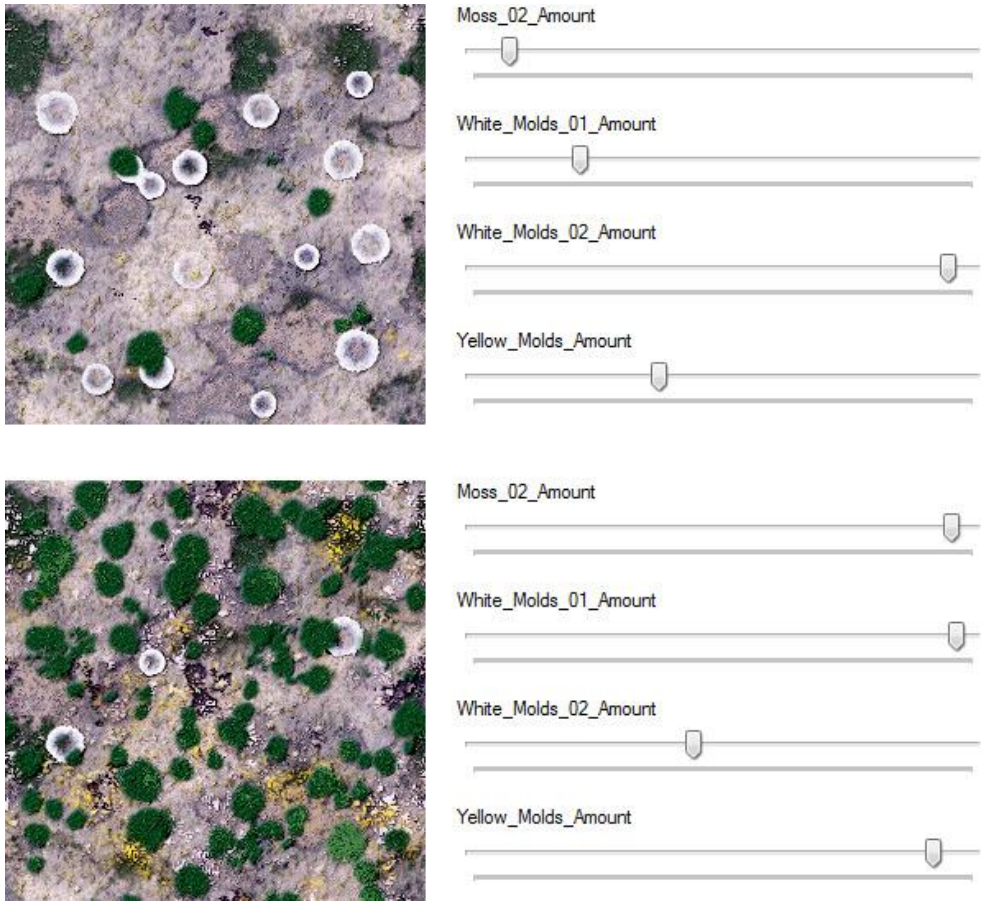
- Kombination mit regelbasierten Systemen



Ausblick: aktuelle Forschungsthemen

- Modelle mit semantischen Parametern
  (Software: Substance/Allegorithmic)

- Forschungsprojekte,
  z.B. ReProcTex
  https://www.shadertoy.com/view/wstcRH



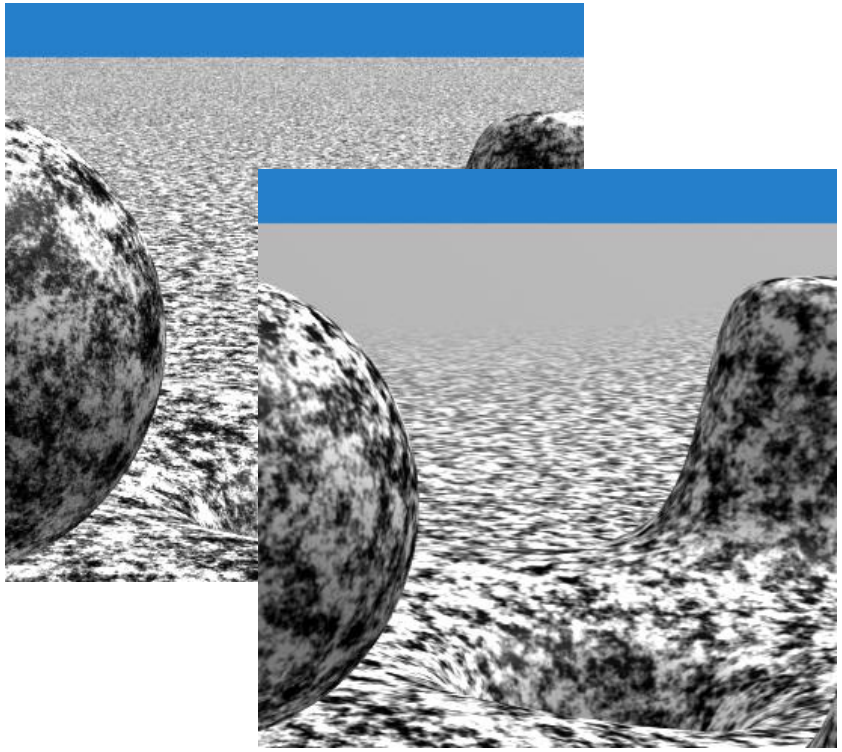
Zwischenfazit: Prozedurale Texturen

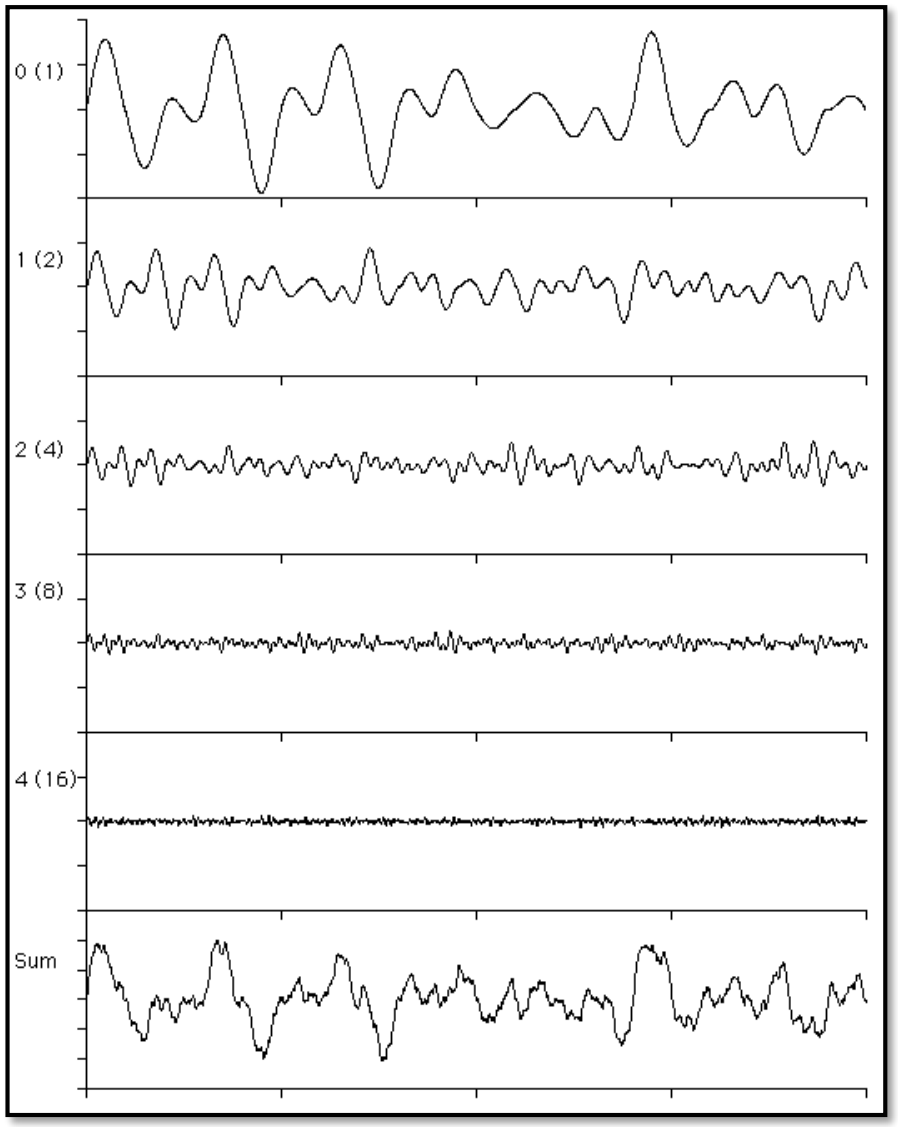
- algorithmische Beschreibung von Texturen
  - kompakte Repräsentation: wenige Zeilen Code
  - auflösungsunabhängig, beliebig große Flächen ohne sichtbare Wdh.
  - parametrisierbar (z.B. mehrere Sorten Holz, Terrains, …)
- **implizite Methode**: Aufruf des Shaders für jeden Pixel/Schnittpunkt
  - benötigt evtl. Zugriff auf benachbarte Pixel
    (z.B. Ableitungen für prozedurale Normalmaps)
  - Filterung ist i.A. nicht trivial (nur bei einfacher
    Turbulenz u.ä., siehe nächste Folie)
- **explizite Methode**: generiere 2D/3D-Textur vor der Verwendung
  - verwende Texture-Mapping wie bisher
  - Vorteil: Texture-Filtering ist einfach, Nachteil: Speicherverbrauch
- **Vorteile beider Welten**: on-demand Synthese und Caching

Filterung bei Turbulenz

Spektrale Synthese: Kombination unterschiedlicher Frequenzbereiche

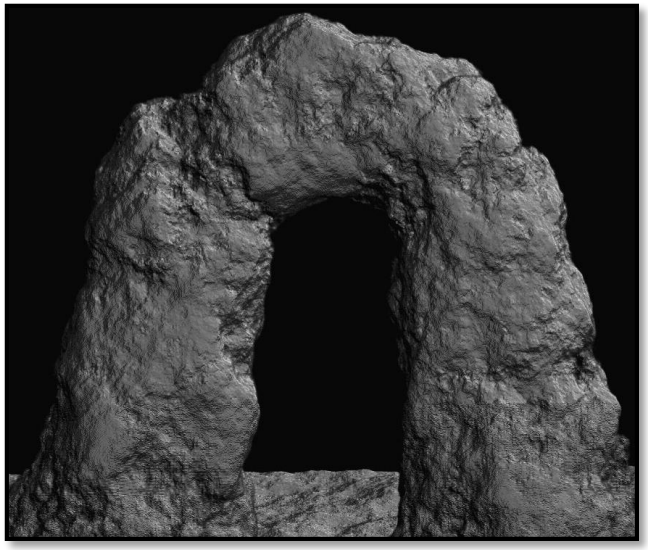
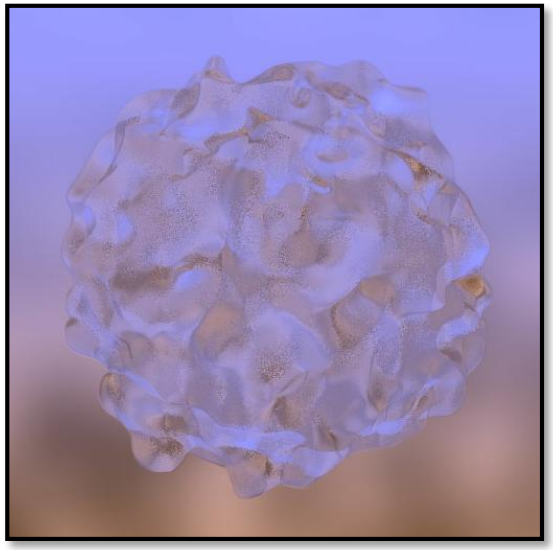
- $\text{turbulence}(\mathbf{x}) = \sum_k \left({}^1/_f\right)^k n\left(f^k \mathbf{x}\right)$
- wie kann man hohe Frequenzen entfernen? (z.B. um beim Texturieren Aliasing zu vermeiden)
  - höhere Oktaven weglassen!





Hypertextures (Perlin 1989)

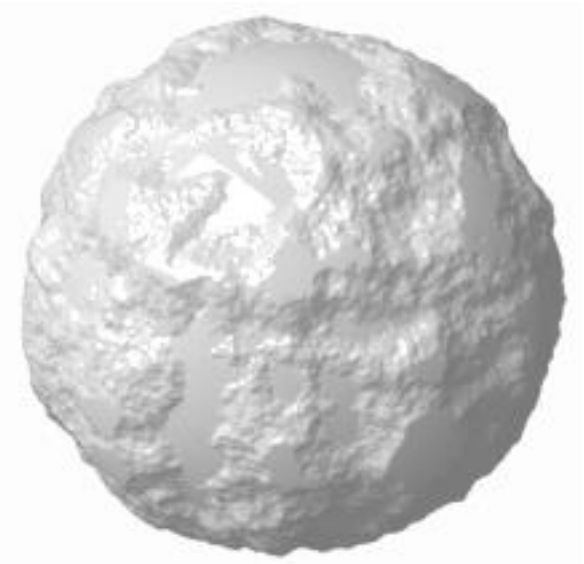
- Ziel: Erzeugung von komplexen 3D-Volumenmodellen (z.B. Feuer, Wolken)
  - Analogie zu Displacement-Mapping von Flächen:
    verändern von volumetrischen Gebilden (vgl. animiertes Feuer gleich)
- Beschreibung eines Hypertexture-Objekts durch eine Dichtefunktionen (DF) $D(\mathbf{x})$ mit $\mathbf{x} \in \mathbb{R}^3$
  - opak $D(\mathbf{x}) = 1$, transparent $D(\mathbf{x}) = 0$, Übergangsbereich $0 < D(\mathbf{x}) < 1$
  - modifizierte eine einfache Grundform, um komplexe Strukturen durch Modulation mit Noise-Funktionen zu erhalten


Hypertextures

- Objekte werden beschrieben durch eine Dichtefunktionen (DF) im $\mathbb{R}^3$
  - innen $D(\mathbf{x}) = 1$, außen $D(\mathbf{x}) = 0$, Übergangsbereich $0 < D(\mathbf{x}) < 1$

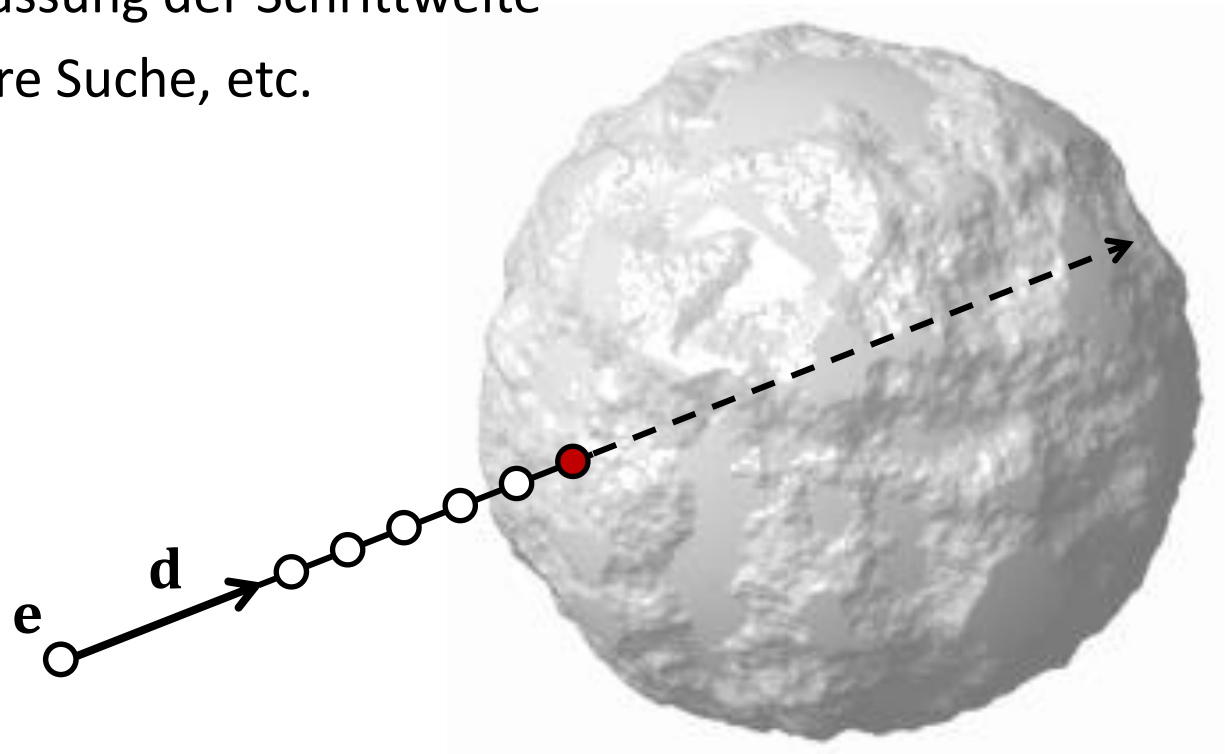
- Bsp. Dichtefunktion ohne Transparenz
  - $D(\mathbf{x}) = sphere\big(\mathbf{x} \cdot (1 + a \cdot \text{turbulence}(\mathbf{x}))\big)$
  - mit $sphere(\mathbf{x}) = \begin{cases} 1 & |\mathbf{x}| < 1 \\ 0 & sonst \end{cases}$



Hypertextures

Ray Marching (opake Objekte)

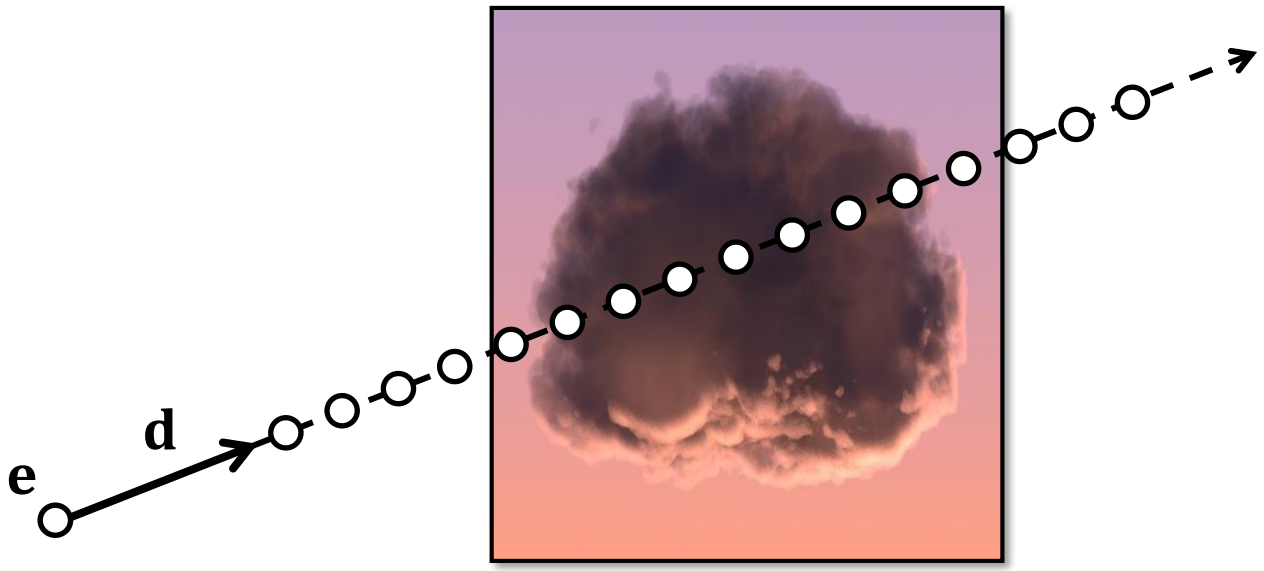
- direkte Darstellung, Alternative zur Triangulierung der Oberfläche
- gehe in kleinen Schritten entlang des Strahls
- teste: wenn $D(\mathbf{x}) = 1$, dann Schnittpunkt gefunden
- viele verschiedene Optimierungen für Ray Marching
  - Anpassung der Schrittweite
  - binäre Suche, etc.



Hypertextures

Ray Marching (semitransparente Objekte)

- akkumuliere Opazität entlang des Strahls
- $D(\mathbf{x}) = sphere\left(\mathbf{x} + \mathrm{a}\begin{pmatrix}\mathrm{turbulence}(\ldots)\\ \mathrm{turbulence}(\ldots)\\ \mathrm{turbulence}(\ldots)\end{pmatrix}\right)$
- mit $sphere(\mathbf{x}) = \begin{cases} 1 & |\mathbf{x}| < 1 \\ \max(0,\ 2 - |\mathbf{x}|) & sonst \end{cases}$



Feuer mit Hypertextures

Beispiel: Feueranimation

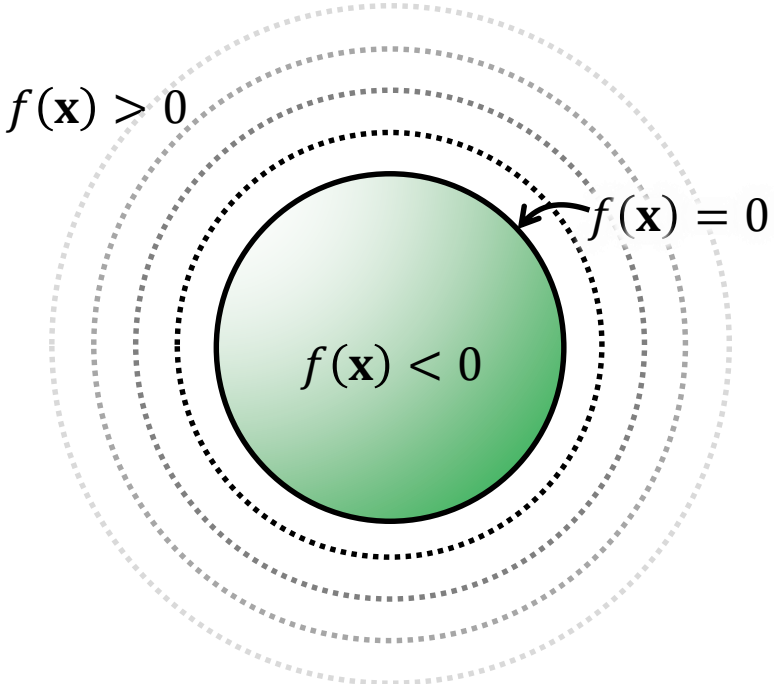
- Feuer mit Hypertexture (Bild rechts): Form einer undeformierten Flamme als Basis, $D(\mathbf{x})$ durch Verformung mit Turbulenzfunktion
  http://blog.char95.com/post/fire/
  http://developer.download.nvidia↷
  .com/SDK/10.5/direct3d/↷
  samples.html#PerlinFire

- Alternative: 4D-Turbulenztextur
  - Zugriff mit Position (3D) und Zeit (1D): 3D-Schnitte aus 4D-Turbulenz-Textur, räumliche Kohärenz wird zu zeitlicher Kohärenz
  - Turbulenz wird auf Farbwerte und Semitransparenz abgebildet
  - zusätzlich zu Turbulenz: nach oben hin transparenter

Distanzfelder, Distanzfunktionen

Implizite Darstellung von (soliden) Objekten

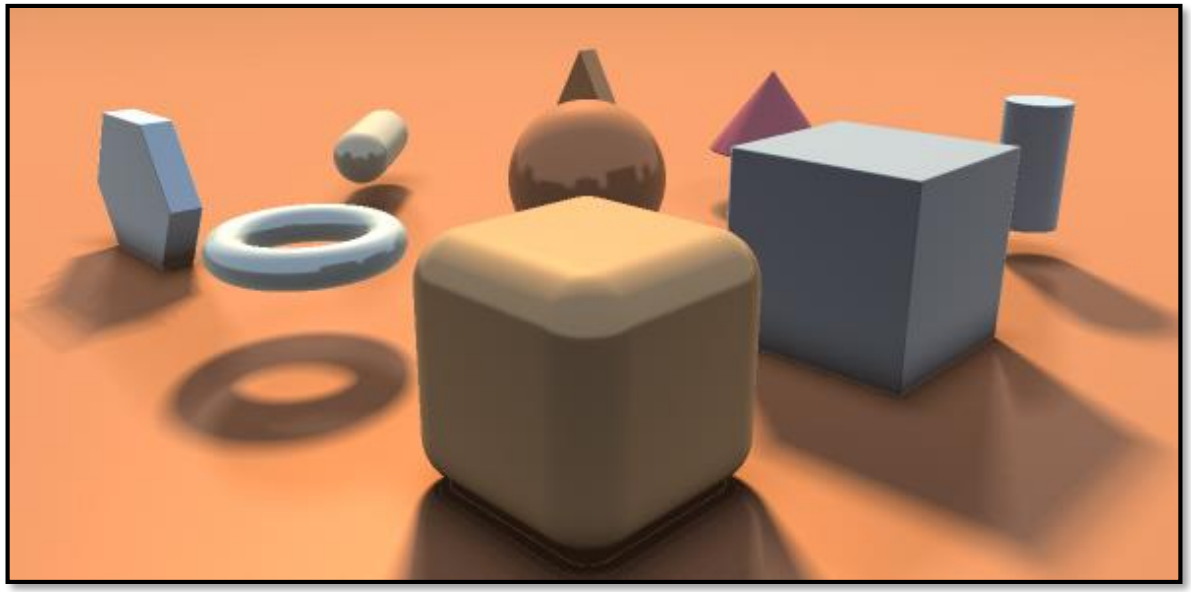
- **Distanzfunktion** $f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^3$, die für jeden Punkt $\mathbf{x}$ die kürzeste Entfernung zur Oberfläche $f(\mathbf{x}) = 0$ liefert
  - man spricht auch von der **Isofläche** für den Wert 0
  - idealerweise ist Abstand vorzeichenbehaftet, z.B. $f(\mathbf{x}) < 0$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



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
  - idealerweise ist Abstand vorzeichenbehaftet, z.B. $f(\mathbf{x}) < 0$ im Inneren
- einfache Objekte durch Distanzfunktionen beschreibbar

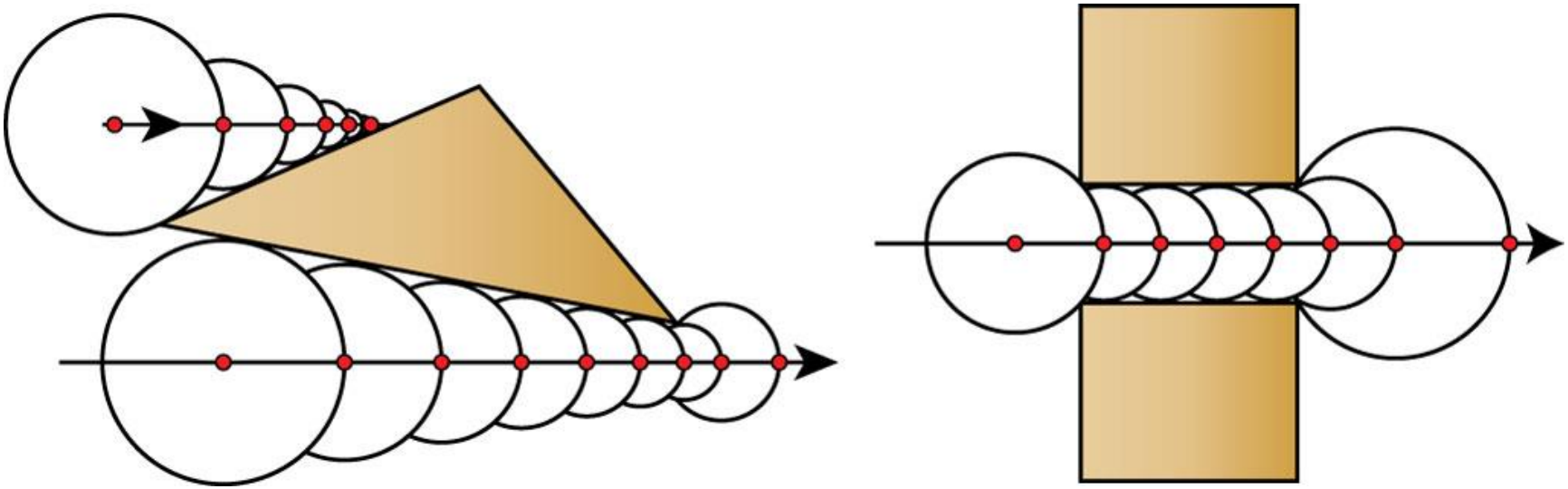


- wird $f(\mathbf{x})$ diskret gespeichert (z.B. in einer 3D Textur) dann spricht man von einem **Distanzfeld**

Distanzfelder, Distanzfunktionen

Schnittpunktberechnung mit Sphere Tracing

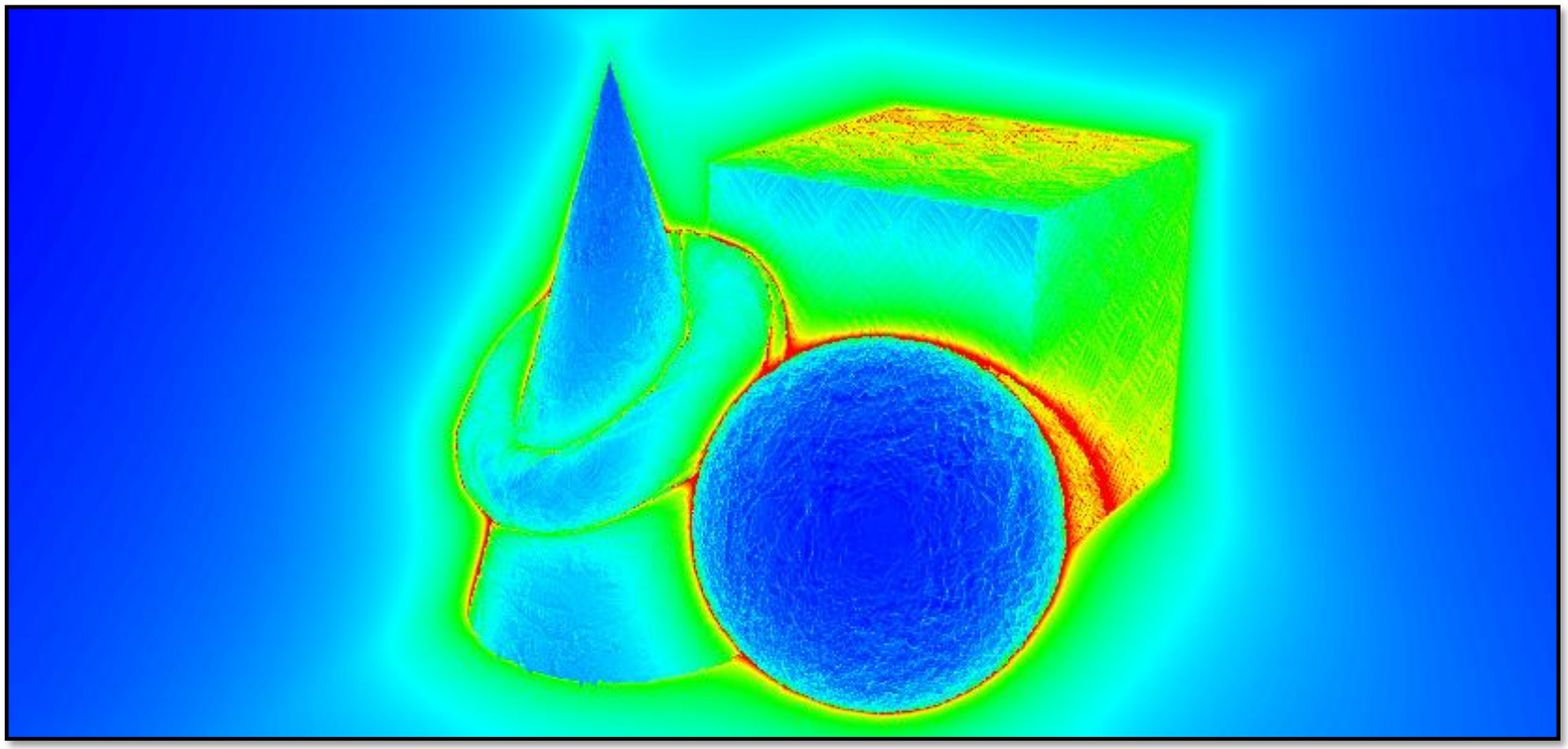
- Distanzfkt./-felder müssen für die Darstellung nicht trianguliert werden
- **Sphere Tracing**: Finden des Schnittpunkts durch Ray Marching *und* Ausnutzen der Distanzfunktion/-felds für die Schrittweite
  - $f(\mathbf{x})$ liefert die kürzeste Entfernung zur nächsten Oberfläche
  - man kann immer eine Strecke $f(\mathbf{x})$ zurücklegen (also auch entlang des Strahls), 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

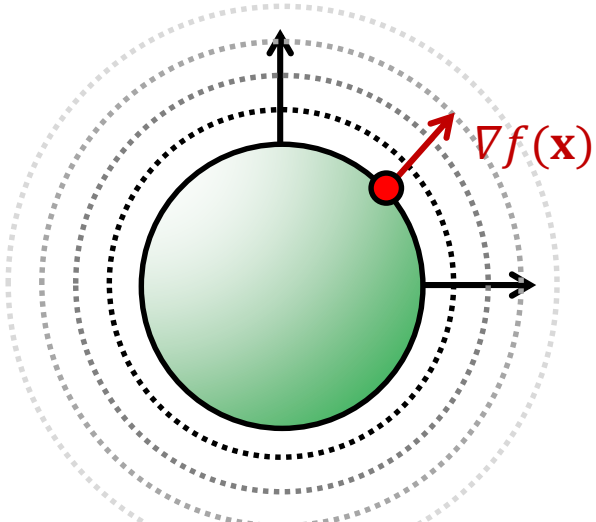


Distanzfelder, Distanzfunktionen

Normale und Beleuchtungsberechnung (auch für Hypertexture-Flächen)

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

$$\nabla f(\mathbf{x}) = \begin{bmatrix} {}^{d}/_{dx} \\ {}^{d}/_{dy} \\ {}^{d}/_{dz} \end{bmatrix} f(\mathbf{x}) \approx \begin{bmatrix} f\left(x + {}^{1}/_{2}\, h, y, z\right) - f\left(x - {}^{1}/_{2}\, h, y, z\right) \\ f\left(x, y + {}^{1}/_{2}\, h, z\right) - f\left(x, y - {}^{1}/_{2}\, h, z\right) \\ f\left(x, y, z + {}^{1}/_{2}\, h\right) - f\left(x, y, z - {}^{1}/_{2}\, h\right) \end{bmatrix}$$



Bsp. Unreal Engine

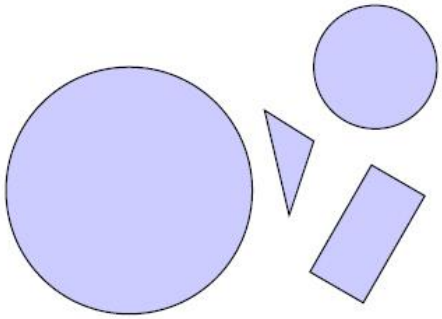
- Distanzfelder zur Approximation von
  - Strahl-Geometrie-Schnitten,
  - Ambient Occlusion,
  - Kollissionsberechnung, …

Bild: https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LightingAndShadows/MeshDistanceFields/

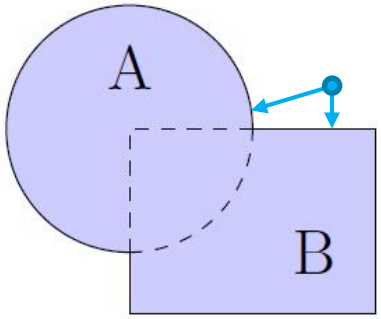
Verknüpfung von Distanzfunktionen

Constructive Solid Geometry (CSG)

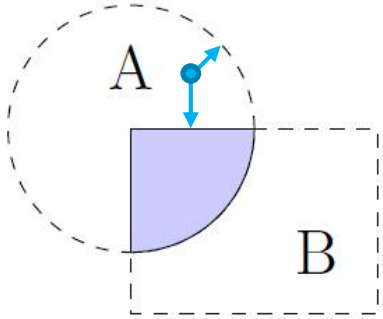
- Boolsche Operatoren für Objekte (aus vorzeichenbehafteten DF)
  - Vereinigung: $f_{A\cup B}(\mathbf{x}) = \min(f_A(\mathbf{x}), f_B(\mathbf{x}))$
  - Schnitt: $f_{A\cap B}(\mathbf{x}) = \max(f_A(\mathbf{x}), f_B(\mathbf{x}))$
  - Differenz: $f_{A\setminus B}(\mathbf{x}) = f_{A\cap -B}(\mathbf{x}) = \max(f_A(\mathbf{x}), -f_B(\mathbf{x}))$



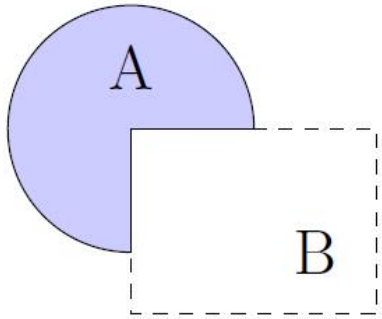
Vereinigung



$A \cup B$



$A \cap B$



$A\setminus B$

- Bemerkung:
  - 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!

Ray Marching von DFs

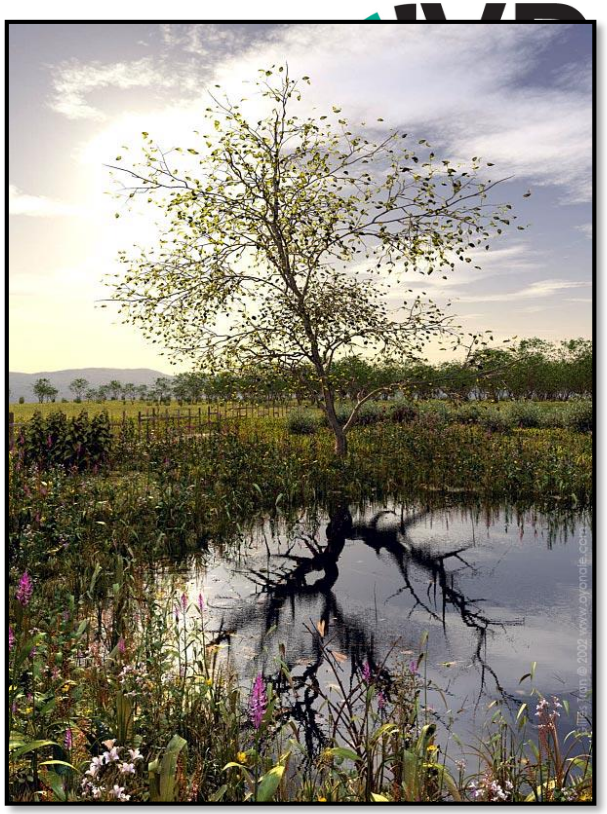
- Distanzfkt. können wie $D(\mathbf{x})$ bei Hypertextures modifiziert werden
- http://iquilezles.org/www/articles/raymarchingdf/raymarchingdf.htm

Ray Marching von DFs

- www.shadertoy.com/view/ld3Gz2

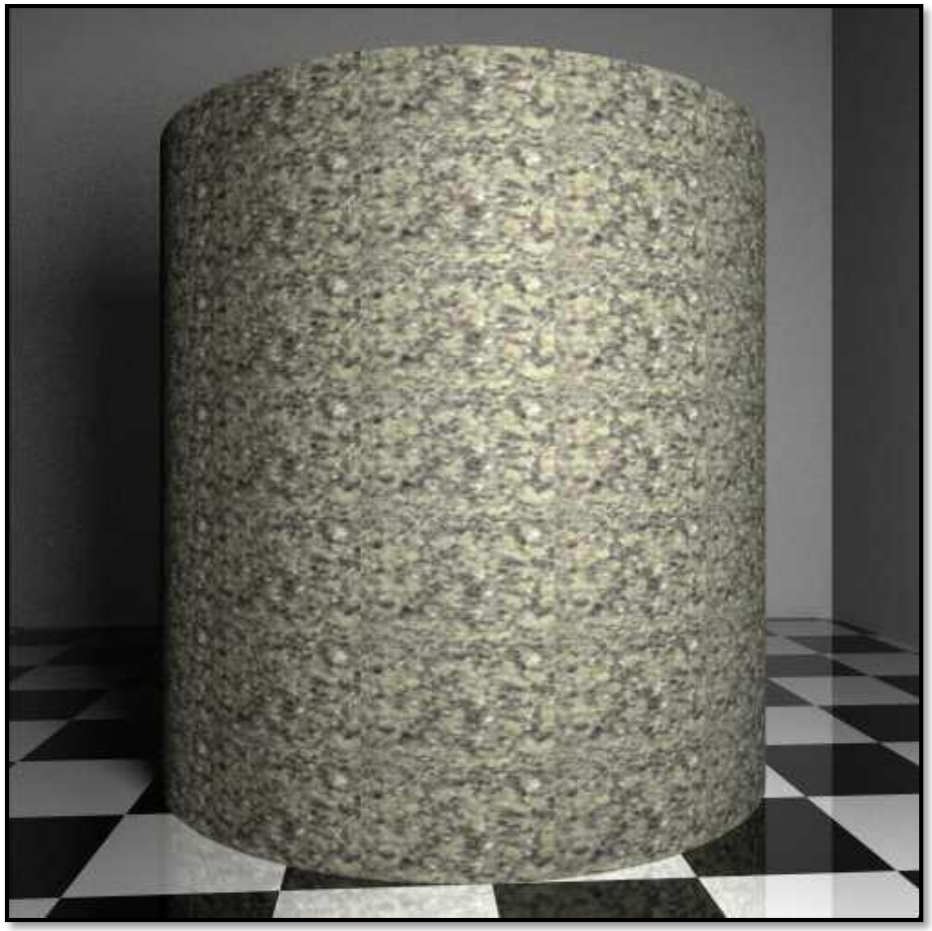
Überblick

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



Textursynthese

- oft kleine Textur („Exemplar“) vorgegeben, z.B. extrahiert aus Fotografie
  - Tiling (Kachelung) führt zu sichtbaren Wiederholungen (Bild links)
- 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“ und Muster – Texturen mit semantischen Strukturen lassen sich so nicht erzeugen

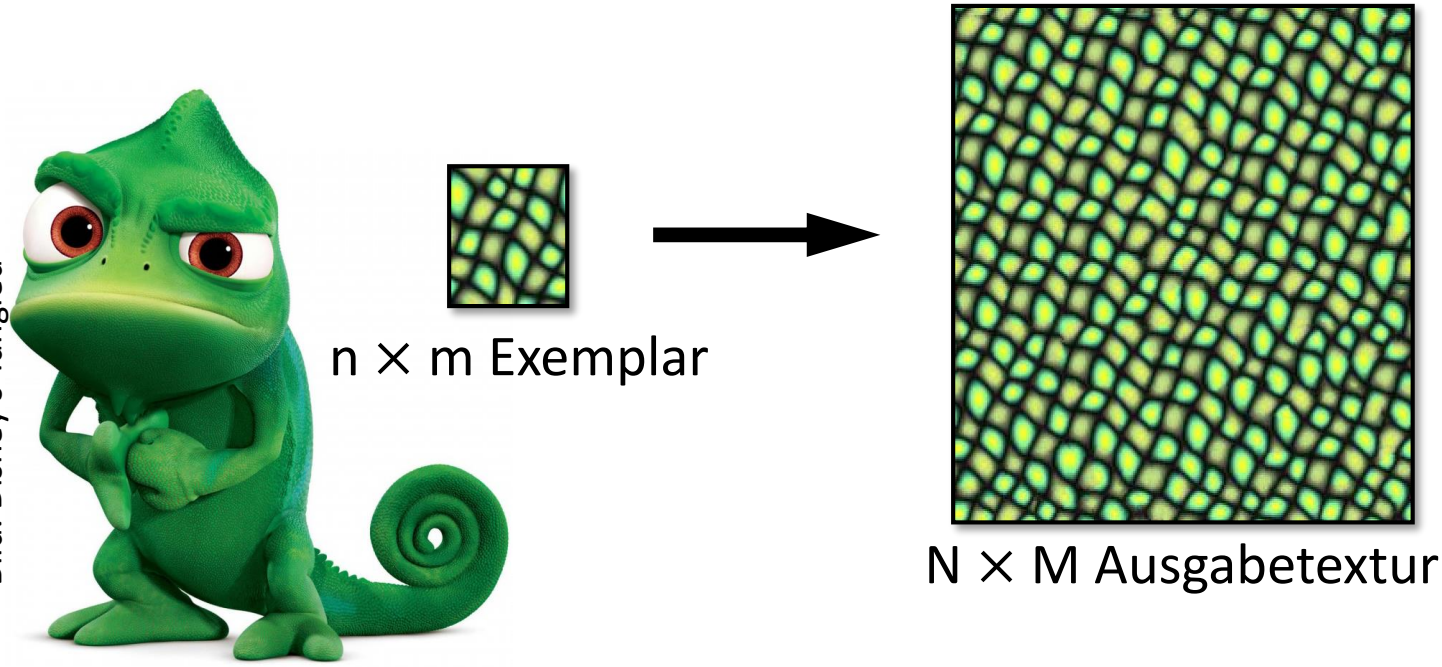


Bild: Disney's Tangled

Bilder: Andrew Nealen (www.nealen.com)

State of the Art in Example-based Texture Synthesis, Wei et al., http://research.microsoft.com/en-us/projects/texturesynthesiseg09star/

Textursynthese (Photoshop)

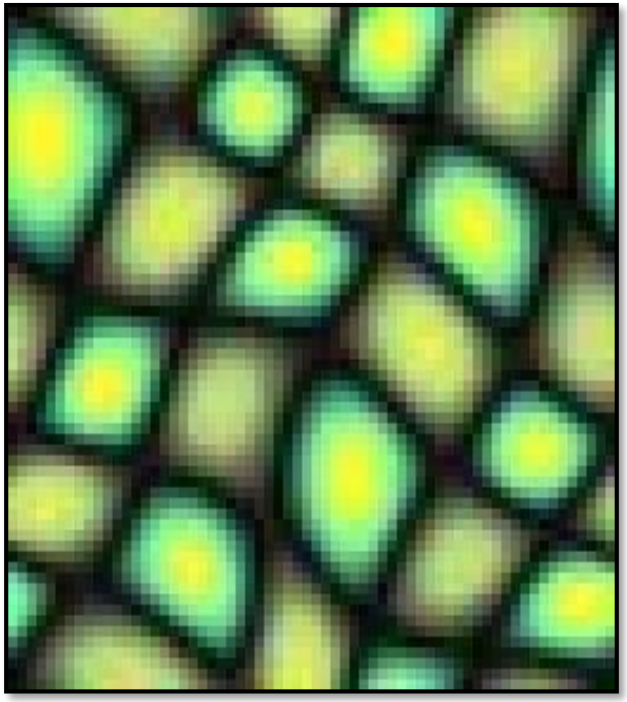
Bilder: James Hays und Alexei A. Efros, Carnegie Mellon University

Textursynthese (Photoshop)

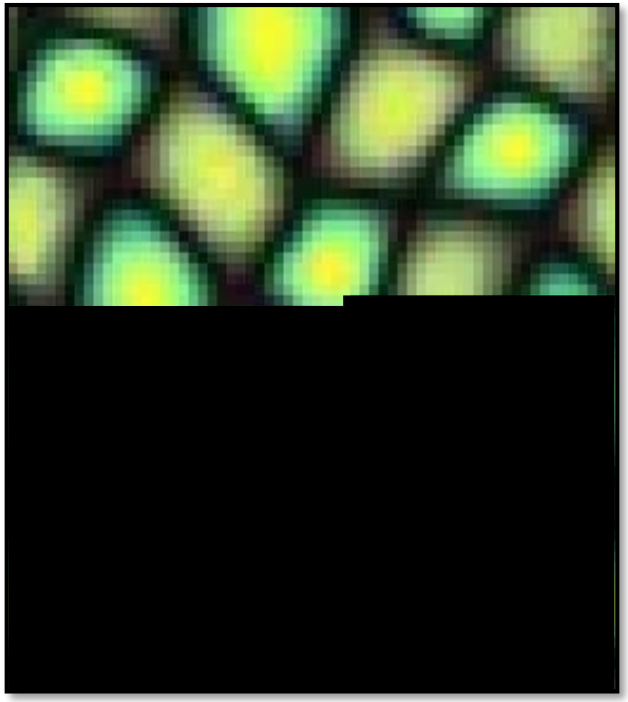
Bilder: James Hays und Alexei A. Efros, Carnegie Mellon University

Pixelbasierte Textursynthese

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



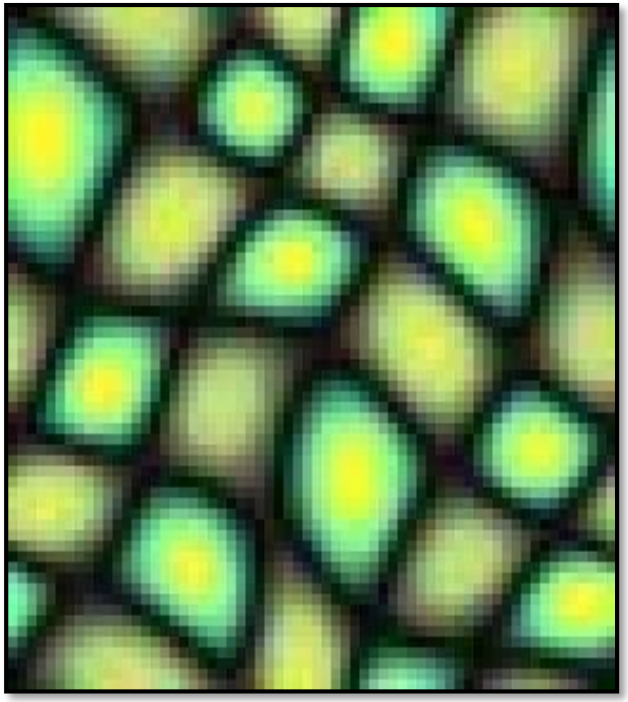
Exemplar



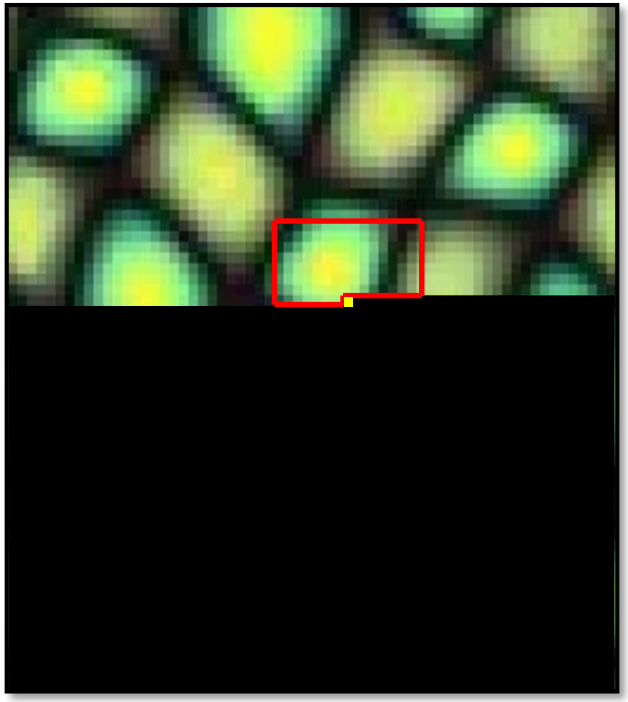
Resultat

Pixelbasierte Textursynthese

- Ausgabetextur wird Pixel für Pixel erzeugt
  - z.B. von links nach rechts und oben nach unten
- betrachte Nachbarschaft des nächsten zu erzeugenden Pixels
  - i.d.R. kleine Nachbarschaften ca. 30-100 Pixel
  - aber groß genug, um Strukturen in der Textur zu erfassen



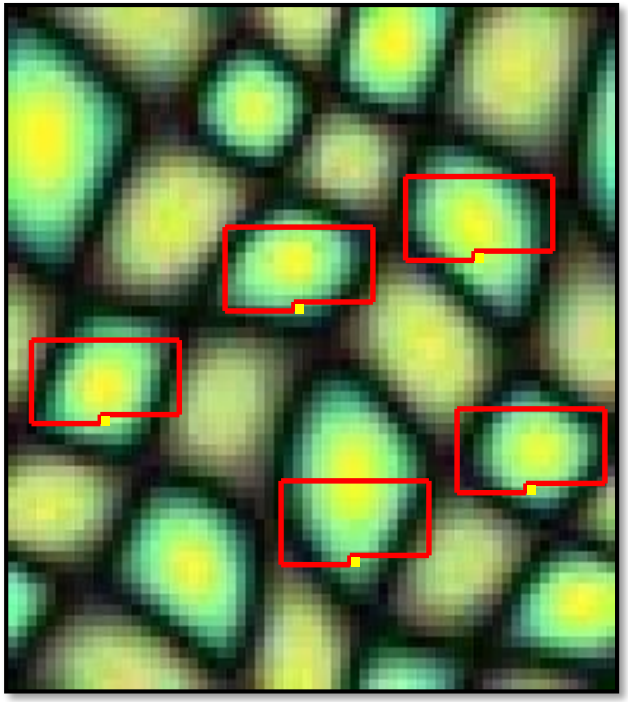
Exemplar



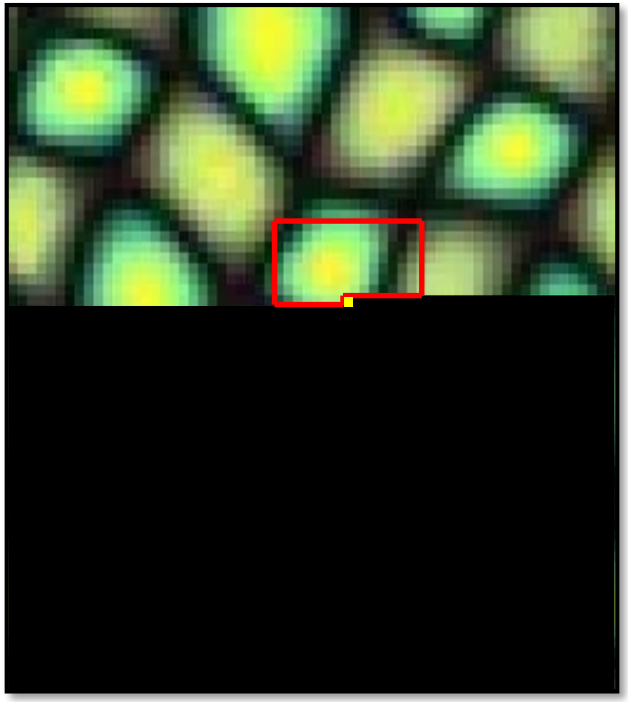
Resultat

Pixelbasierte Textursynthese

- suche ähnliche Nachbarschaften in der Eingabetextur
  - ähnlich = Summe über alle Pixelfarbdifferenzen klein



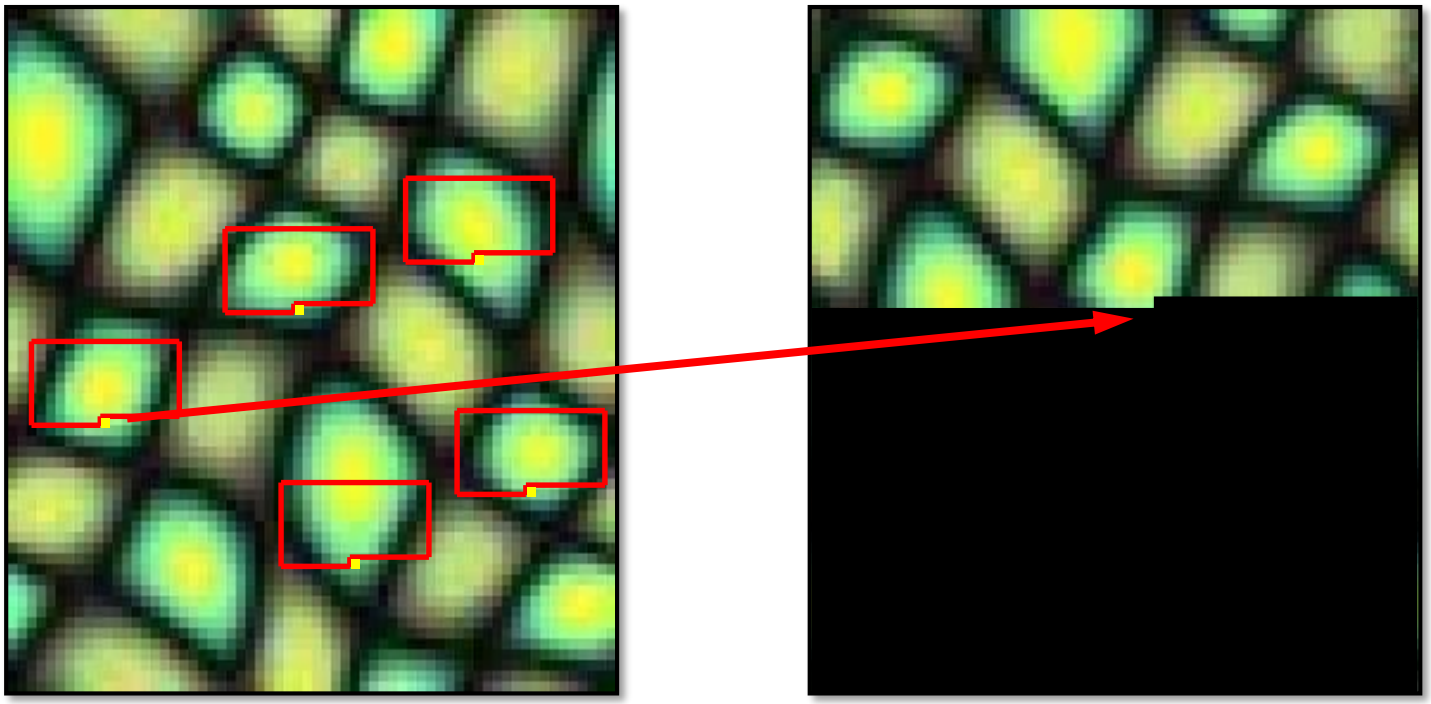
Exemplar



Resultat

Pixelbasierte Textursynthese

- suche ähnliche Nachbarschaften in der Eingabetextur
  - ähnlich = Summe über alle Pixelfarbdifferenzen klein
- kopiere den Pixel **einer der ähnlichsten** Nachbarschaften
  - zufällige Auswahl garantiert die stochastische Variation

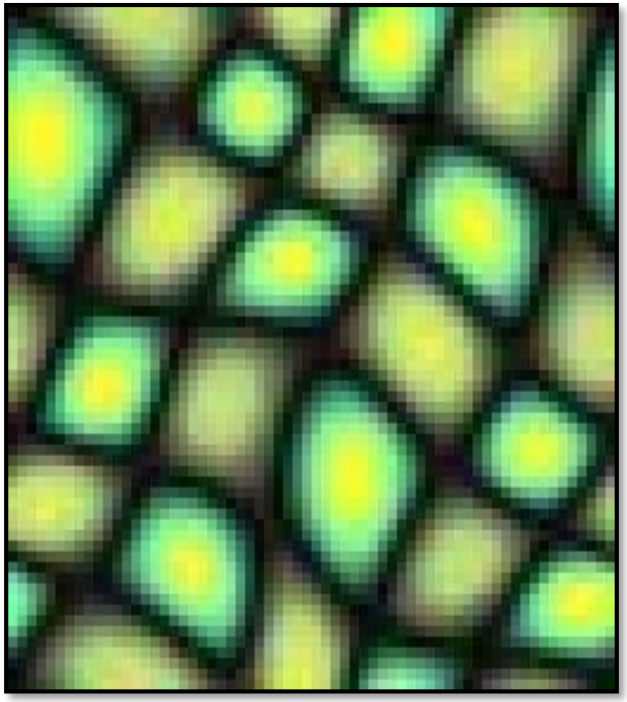


Exemplar

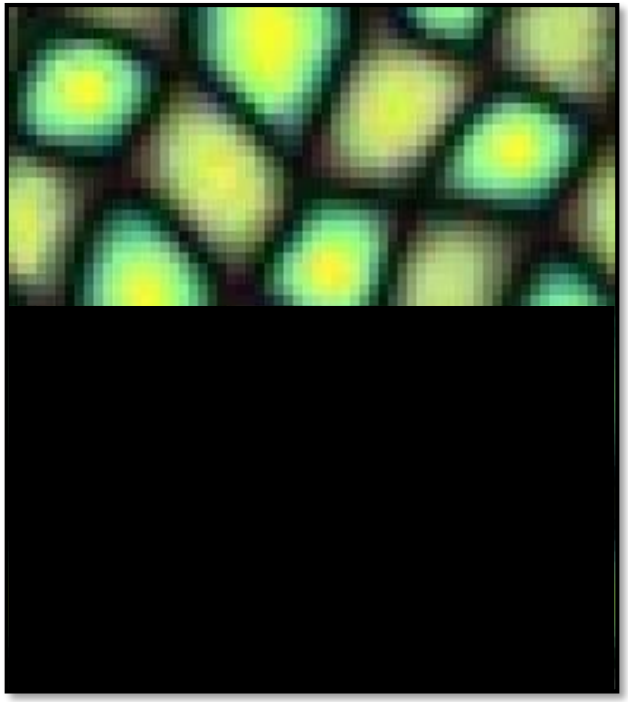
Resultat

Pixelbasierte Textursynthese

- suche ähnliche Nachbarschaften in der Eingabetextur
  - ähnlich = Summe über alle Pixelfarbdifferenzen klein
- kopiere den Pixel **einer der ähnlichsten** Nachbarschaften
  - zufällige Auswahl garantiert die stochastische Variation
- verfahre so, bis alle Ausgabe-Pixel berechnet



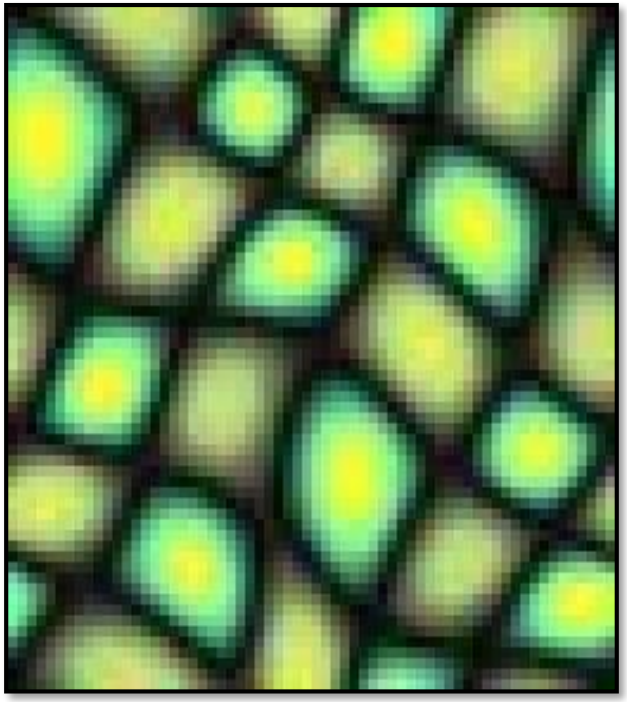
Exemplar



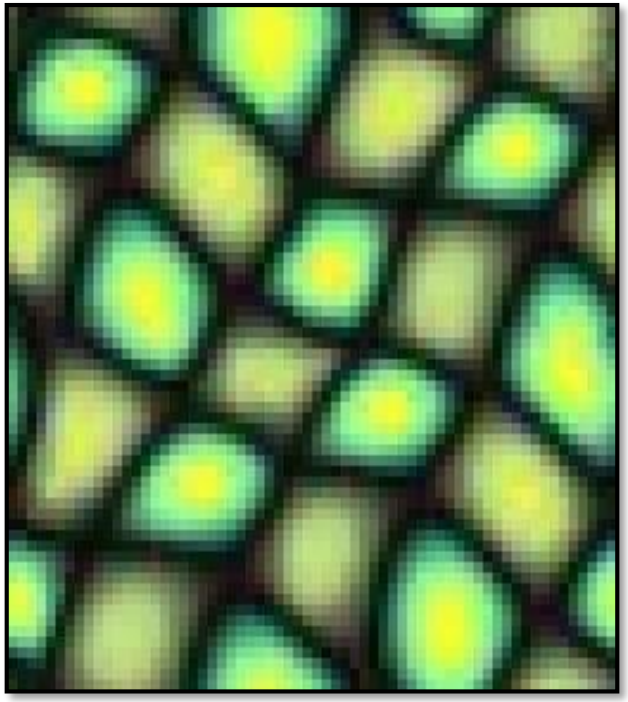
Resultat

Pixelbasierte Textursynthese

- das Prinzip / der Algorithmus ist sehr einfach
- sehr sensitiv gegenüber der Nachbarschaftsgröße
- **sehr langsam** durch die Suche nach passenden Nachbarschaften (Optimierungen: Hauptachsentransformation, Auflösungspyramide, …)

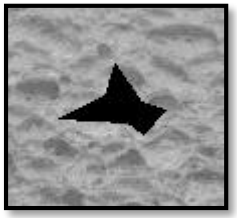


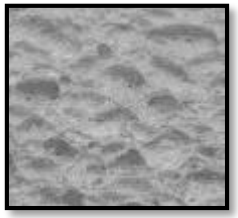
Exemplar

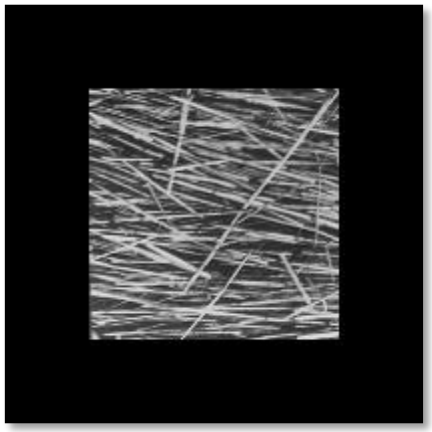


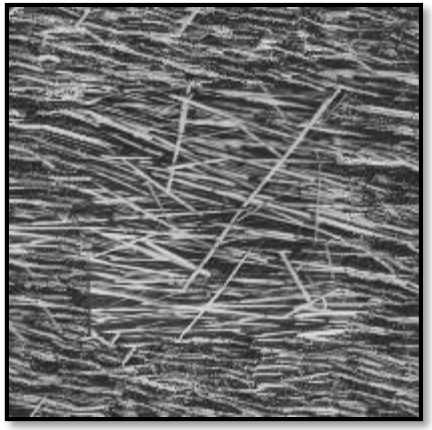
Resultat

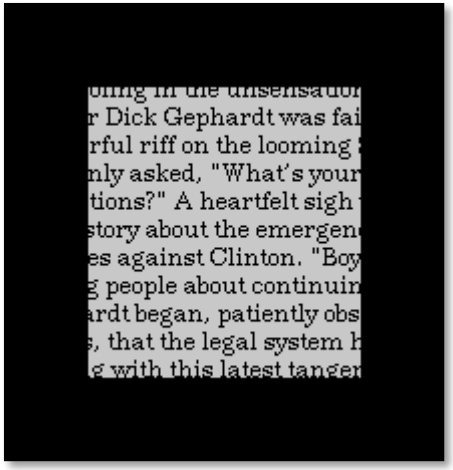
Pixelbasierte Textursynthese [Efros99]

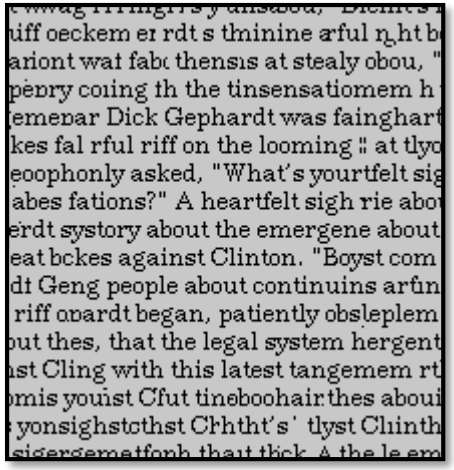






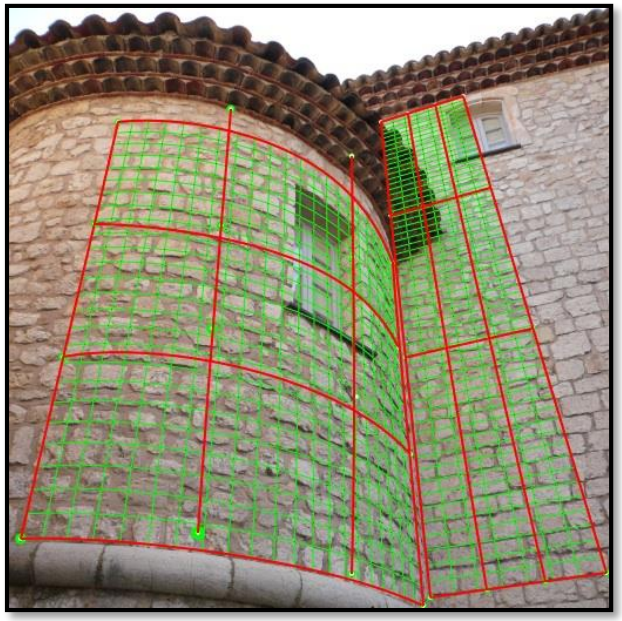






Exkurs: Pixelbasierte Textursynthese

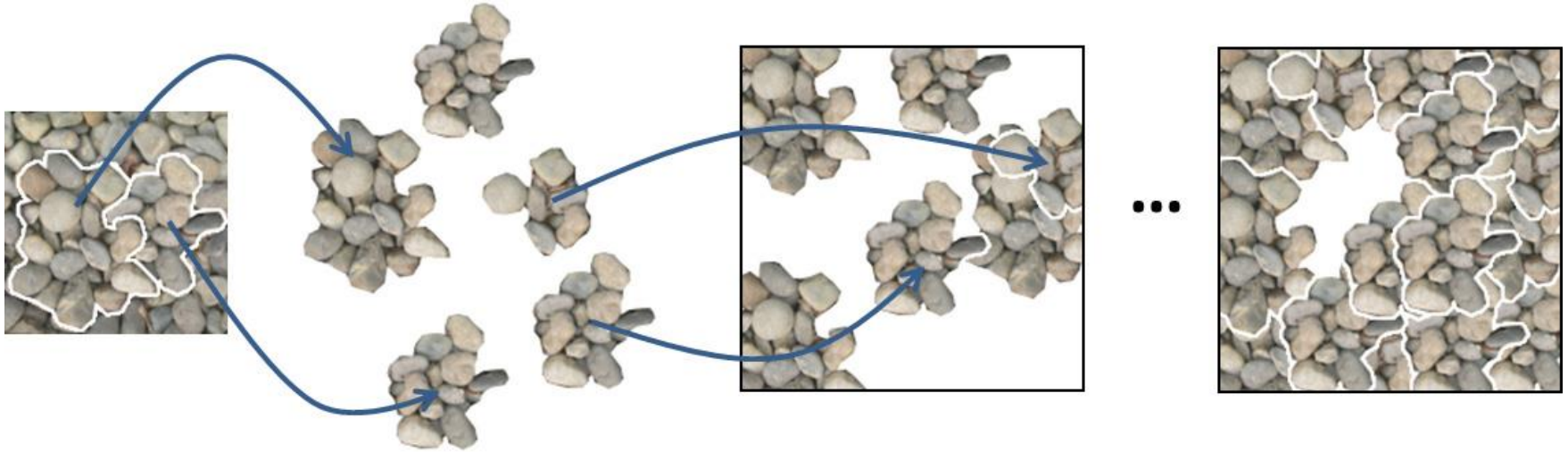
- Berücksichtigung der Perspektive der Eingabe- und Ausgabetextur ist ebenfalls direkt möglich [Eisenacher et al. 08]



- heute: diverse Deep Learning Ansätze zur Bildvervollständigung

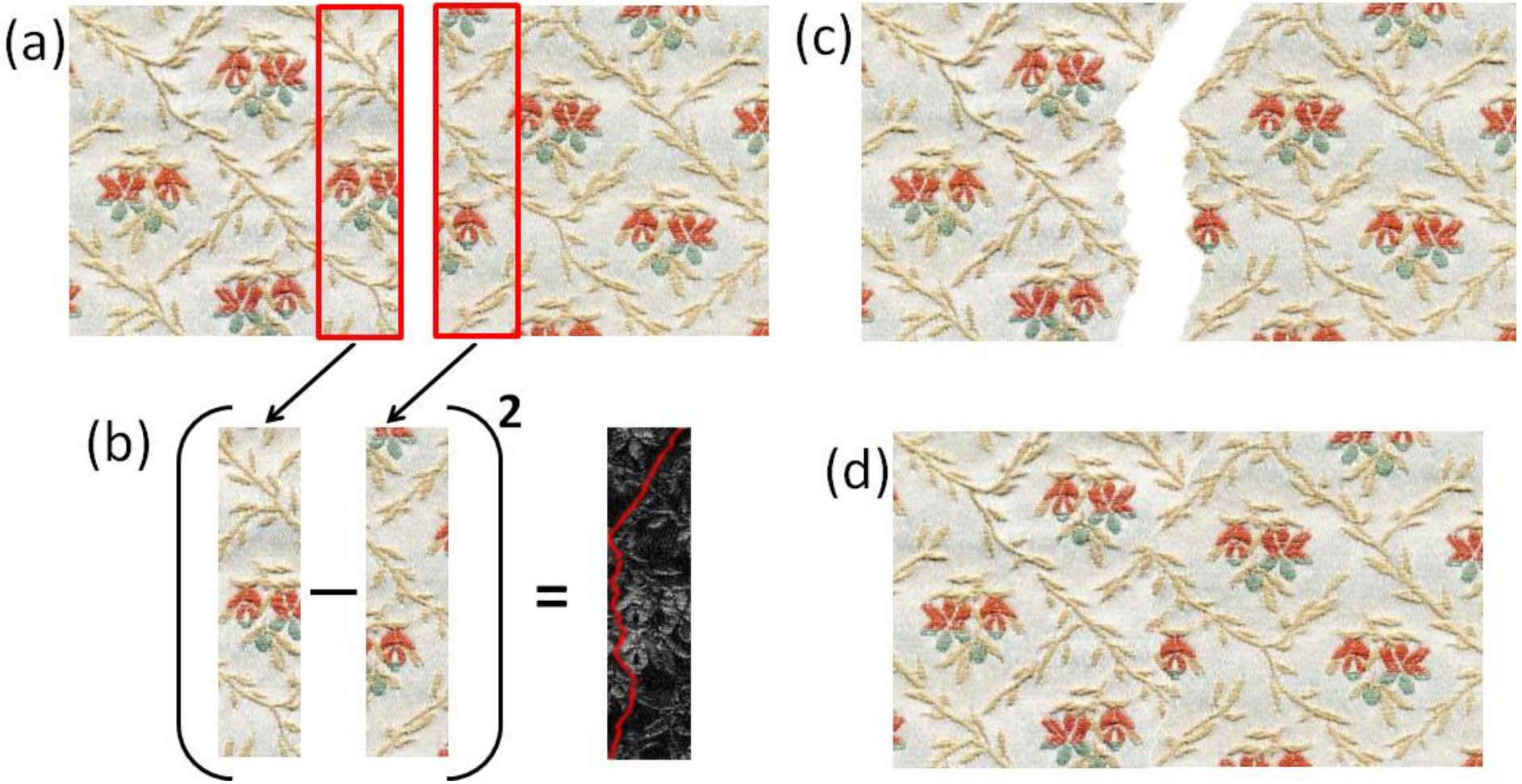
Exkurs: Patchbasierte Textursynthese

- Grundidee: verwende größere Bereiche des Exemplars
- … aber nicht bei jeder Textur ist es einfach gute Patches (Texturstücke) zu extrahieren



Exkurs: Patchbasierte Textursynthese

- Grundidee: verwende überlappende Bereiche des Exemplars und…
- …was nicht passt, wird passend gemacht (Graph Cut/Partitionierung)



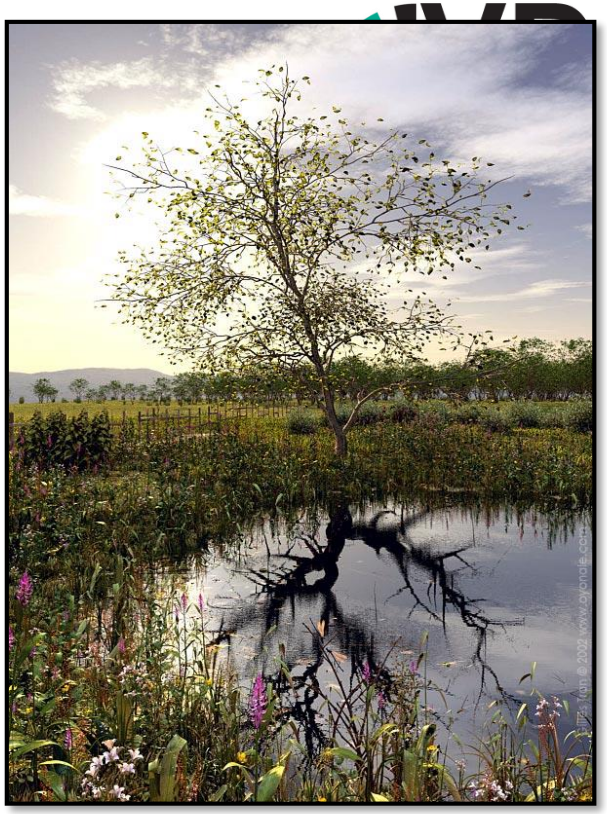
Bilder: Graphcut Textures: Image and Video Synthesis Using Graph Cuts, Kwatra et al.
A simple min-cut algorithm, Journal of the ACM, 1997, Stoer et al.

Exkurs: Image Quilting (patchbasierte Synthese)

- http://graphics.cs.cmu.edu/people/efros/research/quilting.html

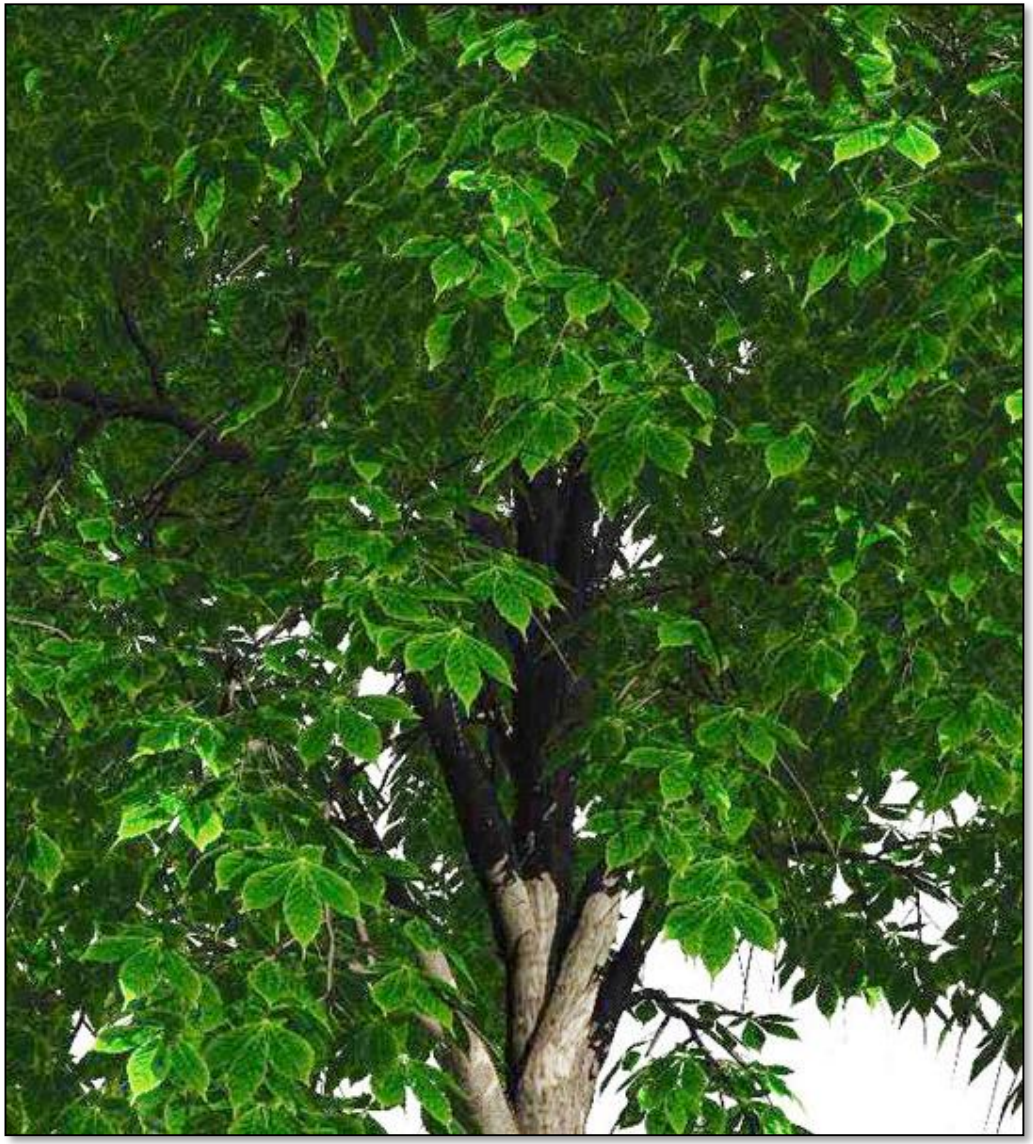
Überblick

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



Vegetation

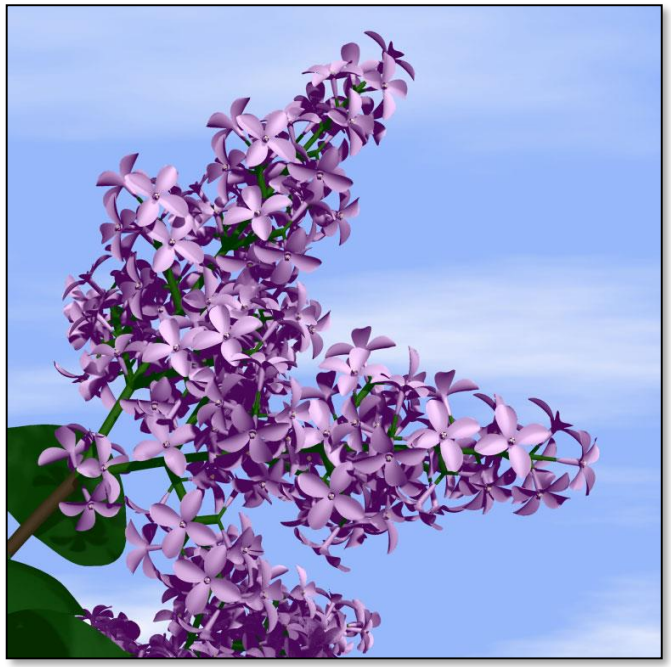
- manuelle Erstellung der Vegetation? 10^8 Dreiecke!



Bilder: www.xfrog.com

Lindenmayer-Systeme, L-Systeme

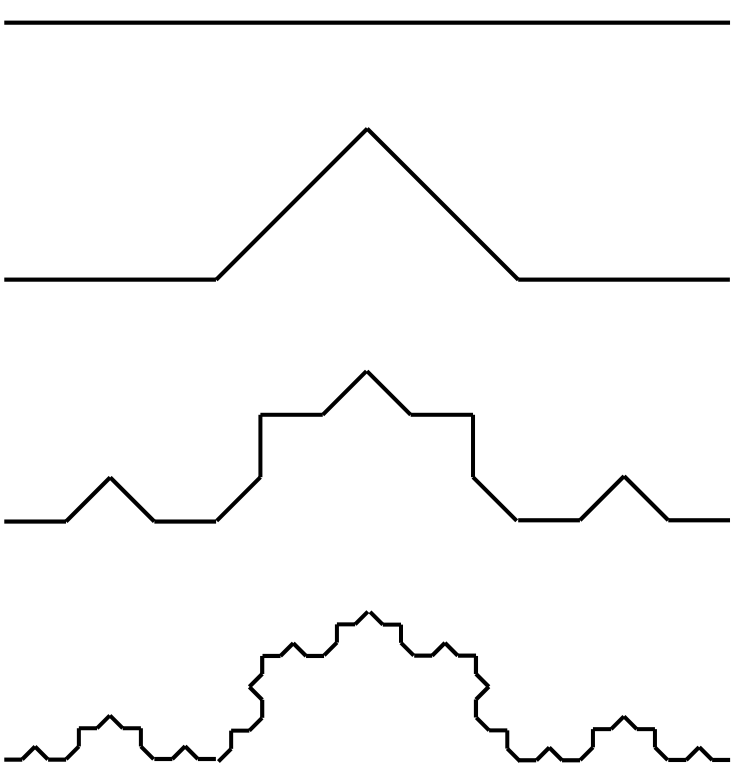
- theoretisches Modell für biologische Entwicklung und Morphogenese (= u.a. Entwicklung von Organismen)
  - nach Aristid Lindenmayer, 1968
  - basiert auf formalen Grammatiken
  - ursprünglich entworfen für die Beschreibung der Entwicklung von mehrzelligen Organismen
  - später: Erweiterung auf Pflanzen und Strukturen mit Verzweigungen



- Grundidee:
  definiere ein komplexes Objekt durch sukzessives Ersetzen von Teilen eines einfacheren Objekts
  (vgl. Wachstumsprozesse)

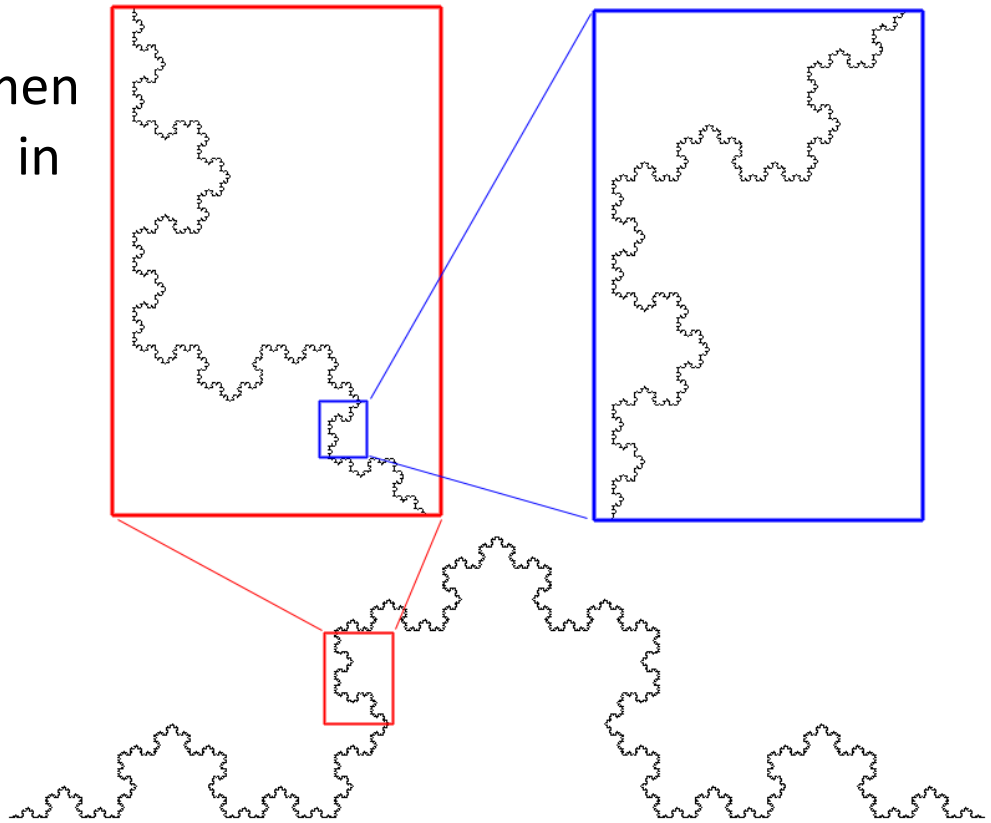
Selbstähnlichkeit durch sukzessives Ersetzen

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



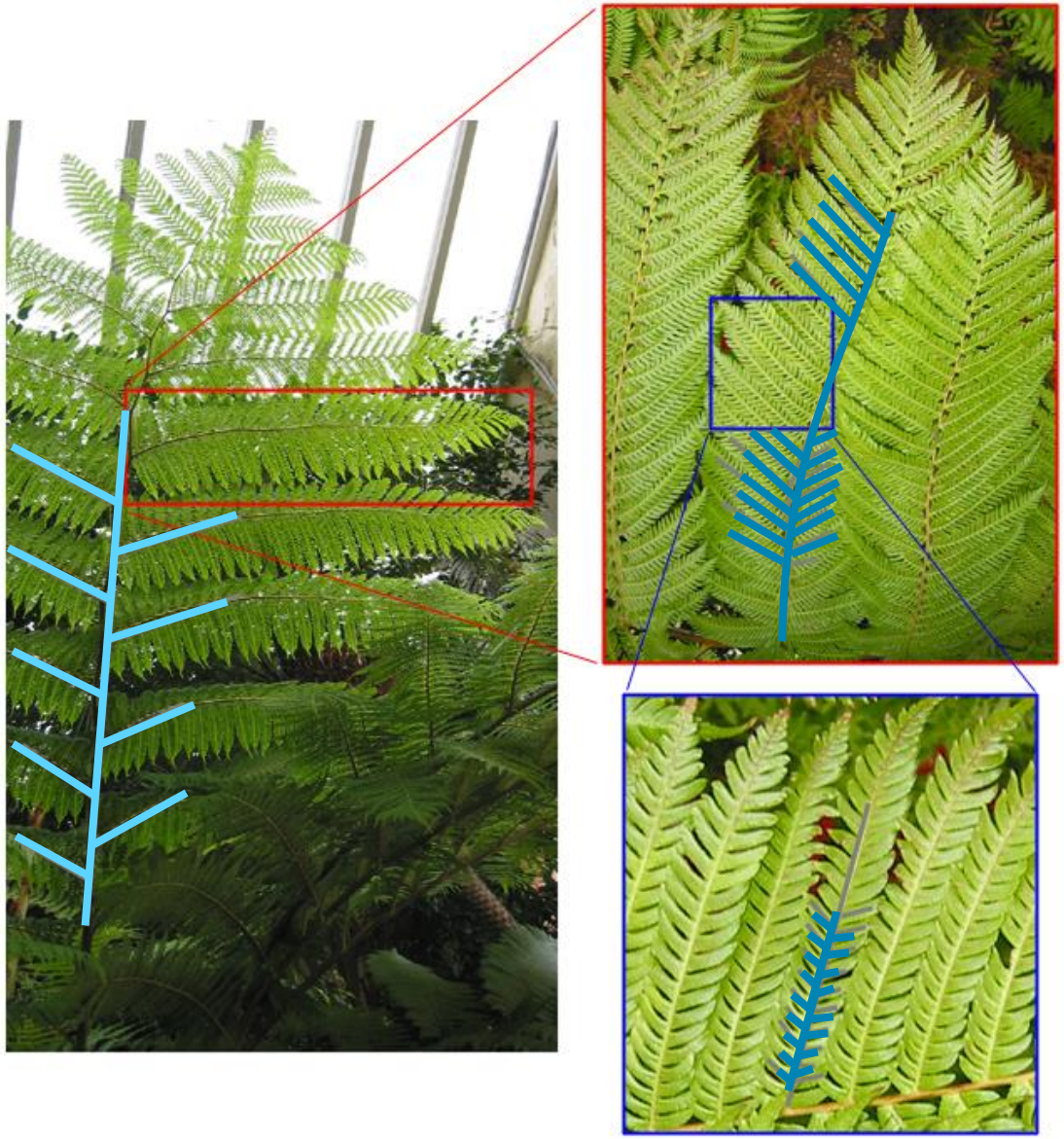
Selbstähnlichkeit in Fraktalen

- “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, Skaleninvarianz, oft nicht-ganzzahlige Dimension
- natürlicher Fraktale sind z.B. Formen von Bergen, Flüssen, Sternhaufen in Galaxien, Blitze
- Bsp. Kochsche Kurve
  - Selbstähnlichkeit ist hier exakt
  - die Form findet sich bei Vergrößerung immer wieder



Selbstähnlichkeit in der Natur

- … ist nicht exakt und 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 (fast) 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 $P \subset (V^* \backslash \Sigma^*) \times V^*$
  - **wichtiger Unterschied**: Produktionsregeln werden parallel angewendet, um biologische 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 $P = \{a \mapsto ab, b \mapsto a\}$
  - Terminalsymbole $\Sigma = \{\}$

- Ableitung (5 Ersetzungsschritte, vorher festgelegt)
  - b
  - a
  - ab (jetzt beide Produktionsregeln gleichzeitig anwenden…)
  - aba
  - $abaab$
  - $abaababa$

Grafische Interpretation: Turtle-Grafik

- „Bildbeschreibungssprache“, wie bei einem Plotter:
  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

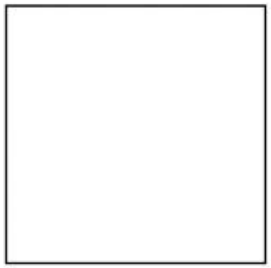
- F Vorwärtsbewegung mit Länge d
  - neue Position: (x', y') mit $x' = x + d \cos \alpha$ und $y' = y + d \sin \alpha$
  - gezeichnetes Liniensegment von (x, y) nach (x', y')
- f Vorwärtsbewegung mit Länge d ohne eine Linie zu zeichnen

- $+$ Rechtsdrehung um den Winkel δ
  - neue Orientierung: $\alpha' = \alpha + \delta$
- $-$ Linksdrehung um den Winkel δ
  - neue Orientierung: $\alpha' = \alpha - \delta$

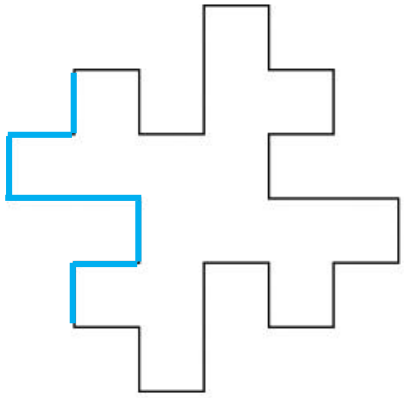
Turtle-Grafik

Beispiel mit Turtle-Kommandos als Zeichen des Alphabets

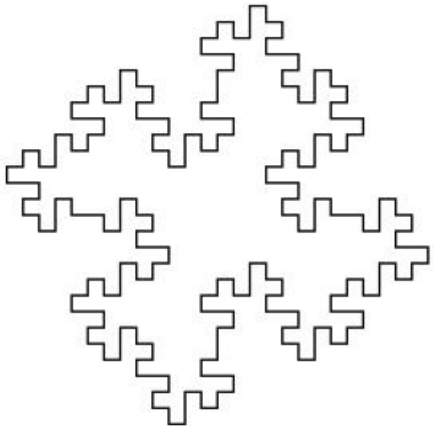
- $G = (V, \Sigma, S, P)$ mit
  - Alphabet $V = \{F, +, -\}$
  - Startwort $S = F + F + F + F$
  - Produktionsregel $P = \{F \mapsto F + F - F - FF + F + F - F\}$
  - Terminalsymbole $\Sigma = \{\}$
  - $\alpha = 90°$



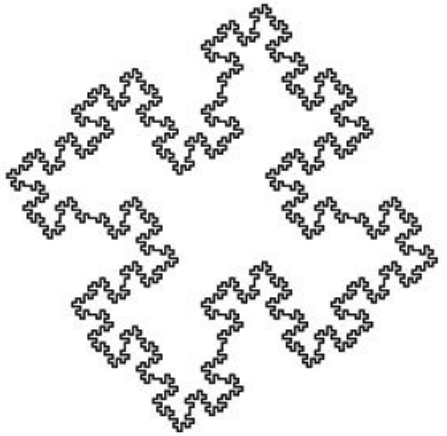
n = 0



n = 1



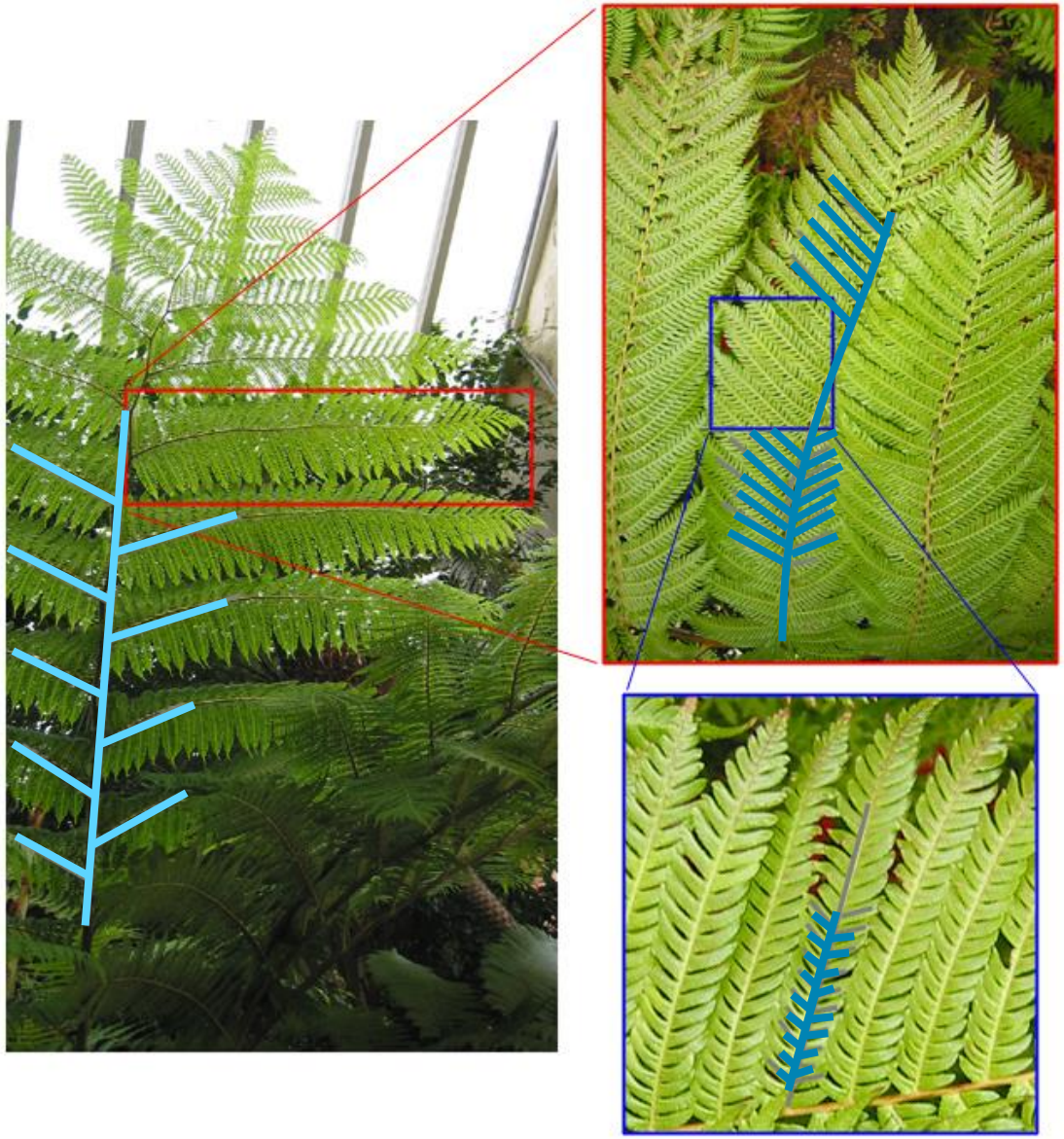
n = 2



n = 3

Wdh. Selbstähnlichkeit in der Natur

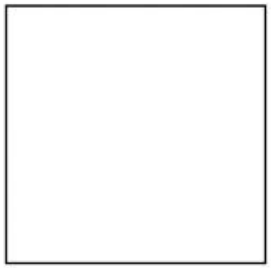
- … ist nicht exakt und tritt nur auf wenigen diskreten Skalen auf (hier 3)
- Selbstähnlichkeit bei Pflanzen ist ein Resultat der Wachstumsprozesse (Mandelbrot, 1982)



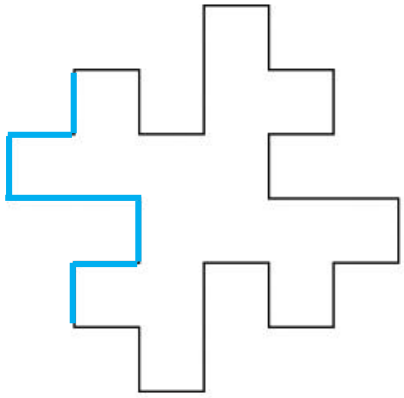
Wdh. Turtle-Grafik

Beispiel mit Turtle-Kommandos als Zeichen des Alphabets

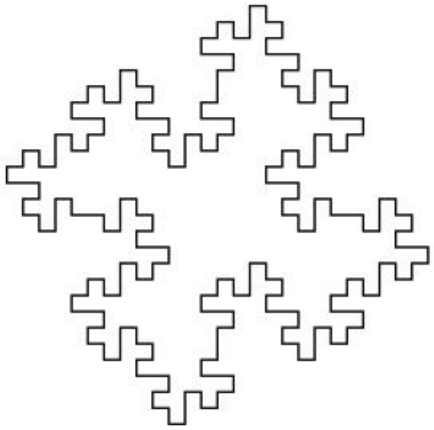
- $G = (V, \Sigma, S, P)$ mit
  - Alphabet $V = \{F, +, -\}$
  - Startwort $S = F + F + F + F$
  - Produktionsregel $P = \{F \mapsto F + F - F - FF + F + F - F\}$
  - Terminalsymbole $\Sigma = \{\}$
  - $\alpha = 90°$



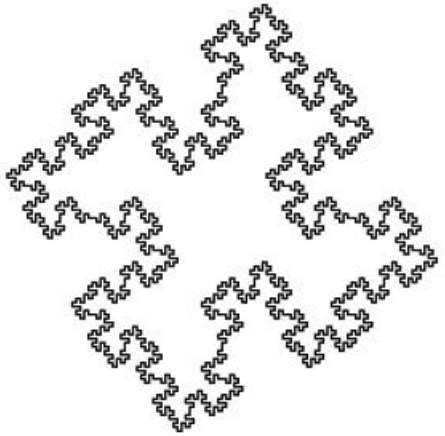
n = 0



n = 1



n = 2

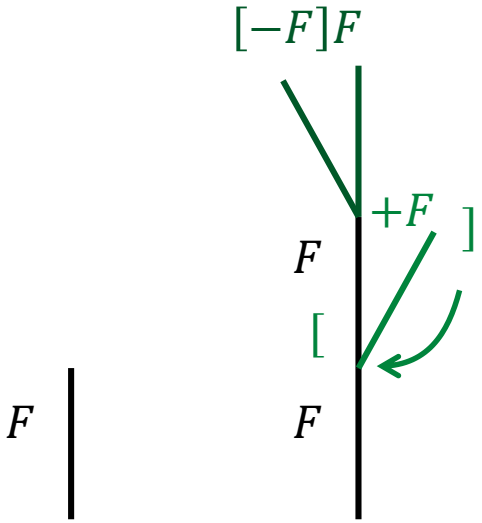


n = 3

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)

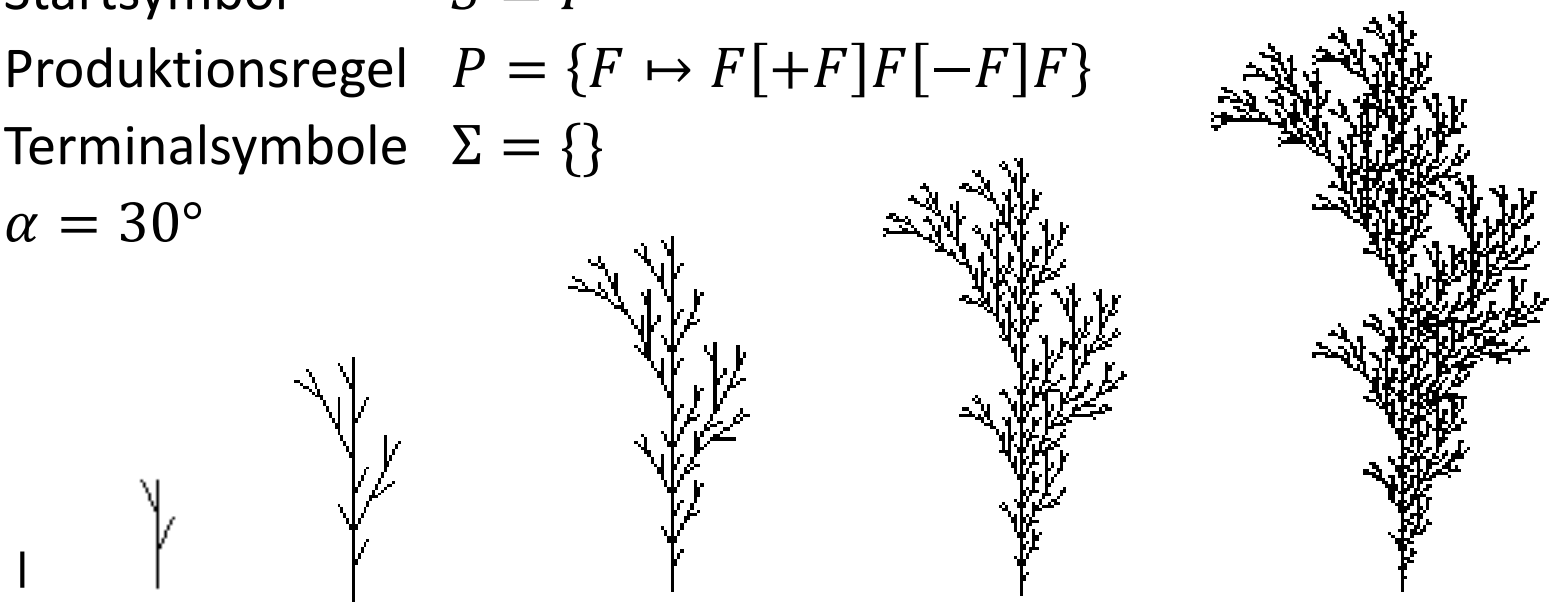
- Beispiel:
  - Alphabet $V = \{F, +, -, [,]\}$
  - Startsymbol $S = F$
  - Produktionsregel $P = \{F \mapsto F[+F]F[-F]F\}$
  - Terminalsymbole $\Sigma = \{\}$
  - $\alpha = 30°$



L-Systeme mit Verzweigungen

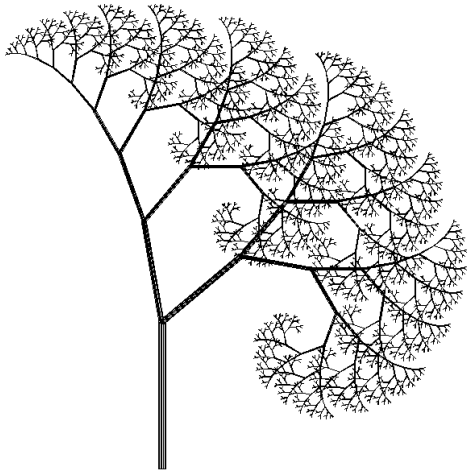
- um Verzweigungen darzustellen wird das Alphabet um zwei Symbole erweitert: „[“ und „]“
  - „[“ push: sichert den aktuellen Zustand (x, y, α) auf einem Stack
  - „]“ pop: holt den letzten gesicherten Zustand vom Stack (dabei wird keine Linie gezeichnet)

- Beispiel:
  - Alphabet $V = \{F, +, -, [,]\}$
  - Startsymbol $S = F$
  - Produktionsregel $P = \{F \mapsto F[+F]F[-F]F\}$
  - Terminalsymbole $\Sigma = \{\}$
  - $\alpha = 30°$



L-Systeme: Erweiterungen (siehe Bonus Material)

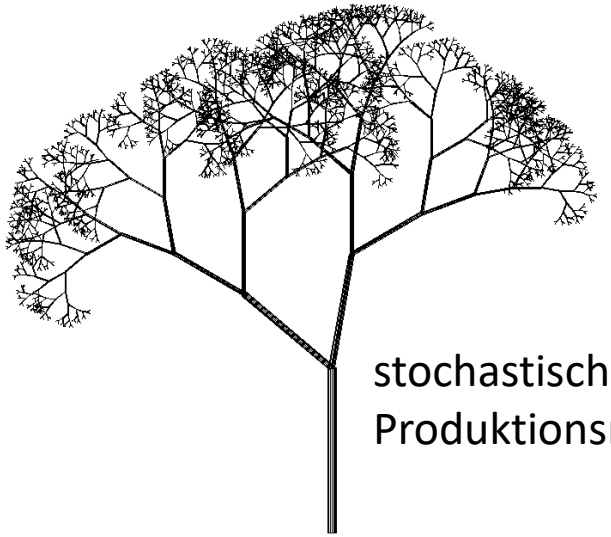
- parametrisierte Symbole (links-oben), stochastische Produktionsregeln (links-unten) und Mutationen (rechts), Orientierung in 3D



parametrisierte Symbole (z.B. Winkel)

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

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



stochastische Produktionsregeln

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

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

Ausblick: 3D L-Systeme

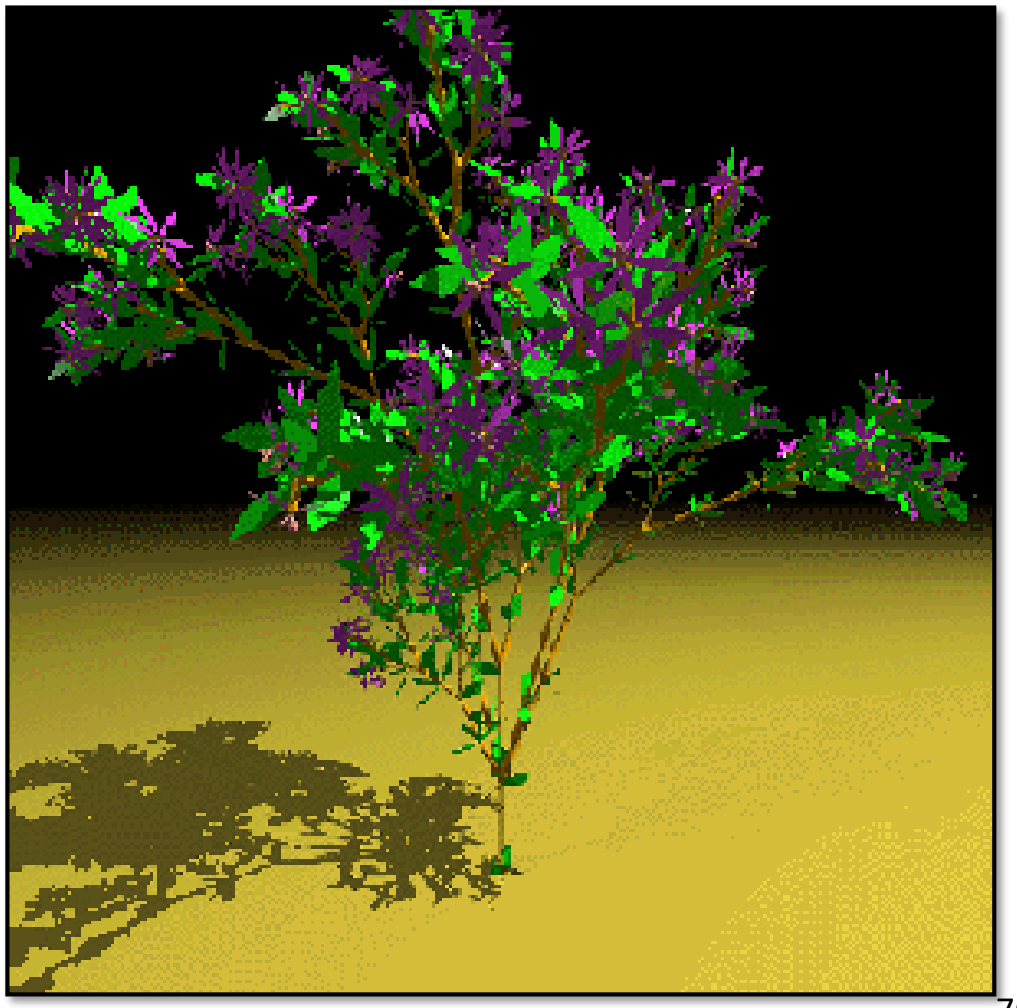
Bild: https://en.wikipedia.org/wiki/L-system

Ausblick: 3D L-Systeme

- The Algorithmic Beauty of Plants von Przemyslaw Prusinkiewicz und Aristid Lindenmayer, Buch zum Download: http://algorithmicbotany.org/papers/#abop
- grafische Interpretation in 3D aufwändiger, z.B. Kantenzug, der ein Polygon definiert

n=7, δ=22.5°

ω : A
p_1 : A → [&FL!A]/////'[&FL!A]////////'[&FL!A]
p_2 : F → S ///// F
p_3 : S → F L
p_4 : L → [''']^^{-f+f+f-|-f+f+f}]



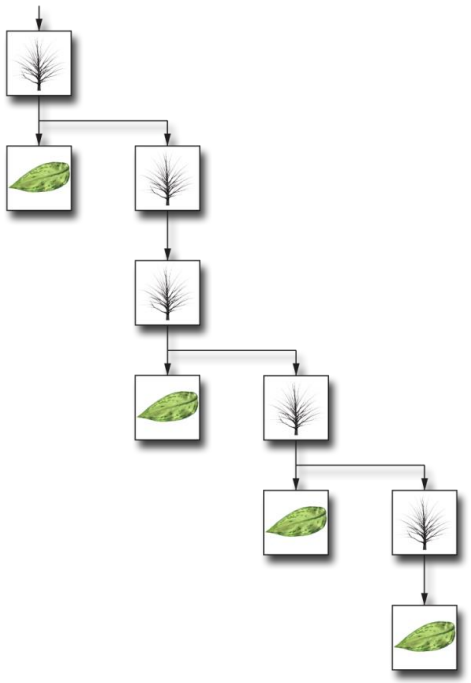
L-Systeme in der Praxis

- 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

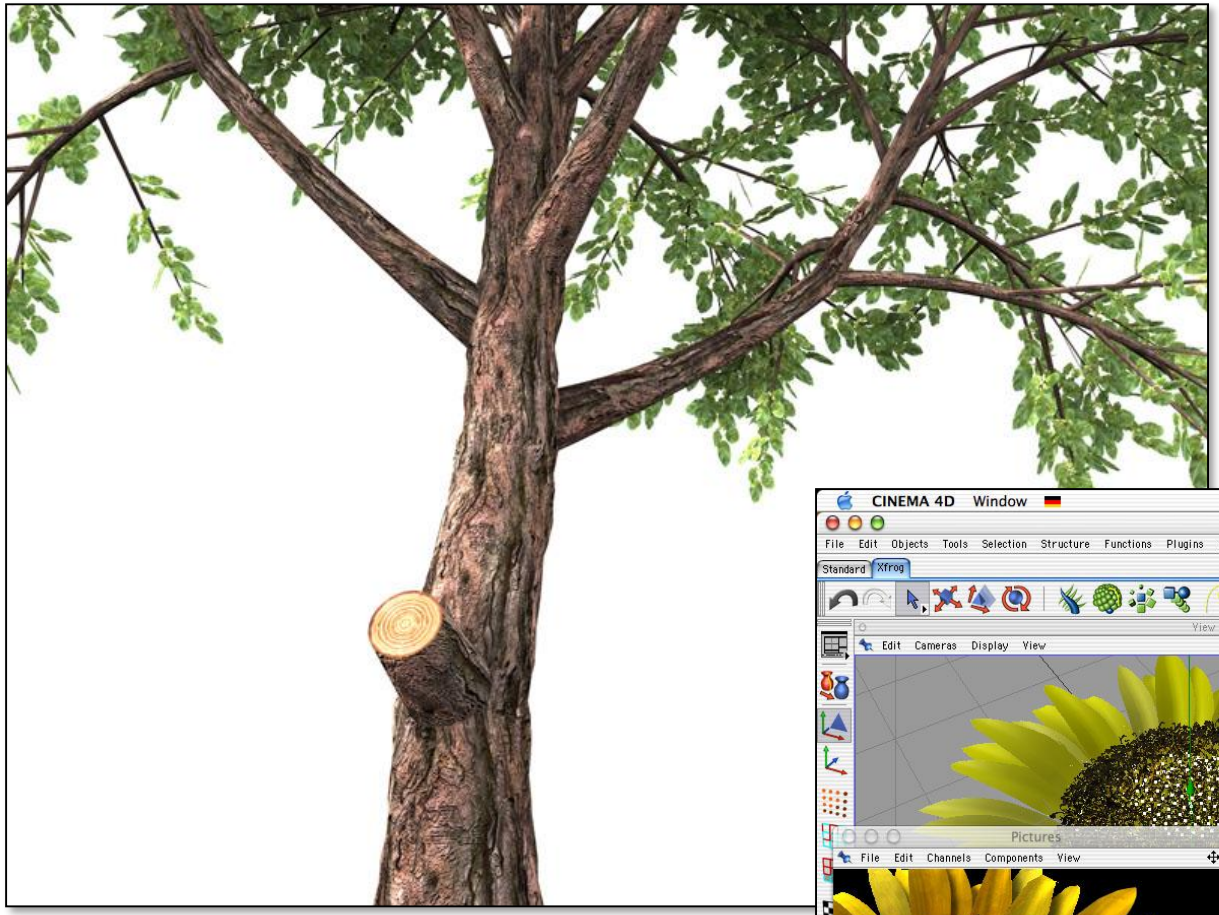
```
n=10, δ = 60 ◦
#include K /* flower shape specification */
ω : A~K
p 1 : A : * → [-/~K][+/~K]I(0)/(90)A
p 2 : I(t) : !(t=2) → FI(t+1)
p 3 : I(t) : t=2 → I(t+1)[-FFA][+FFA]
```

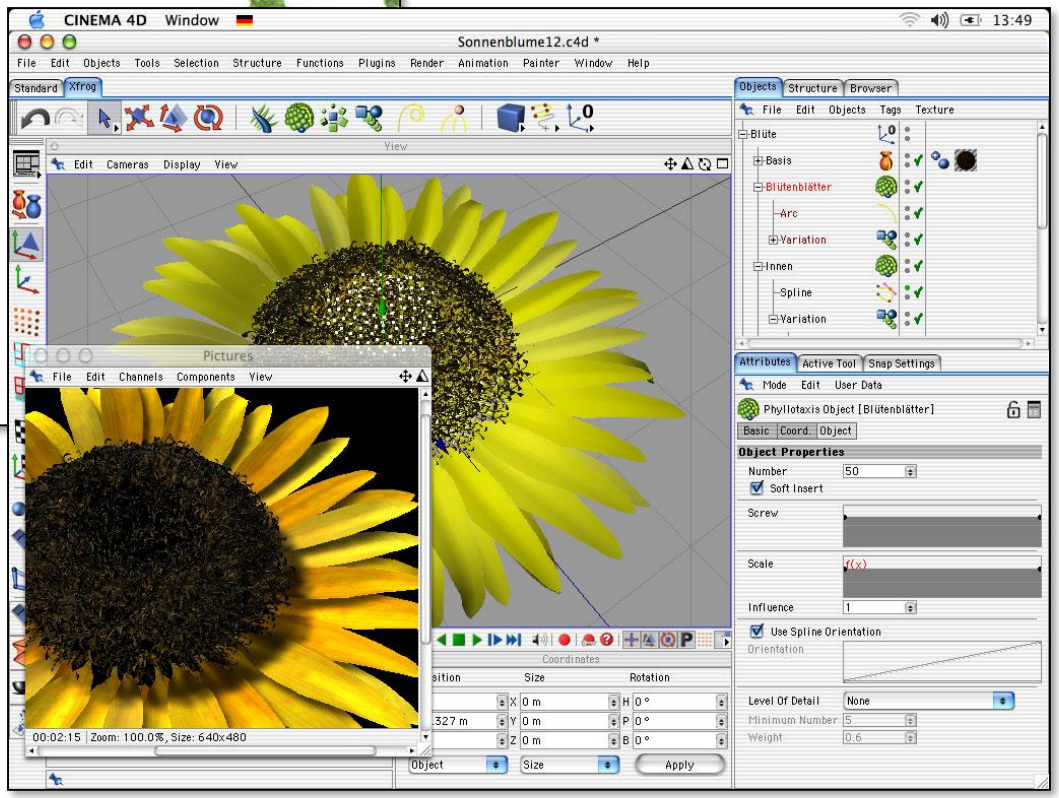
L-Systeme in der Praxis

- 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




CINEMA 4D Window
13:49
Sonnenblume12.c4d *
File Edit Objects Tools Selection Structure Functions Plugins Render Animation Painter Window Help
Standard Xfrog
Edit Cameras Display View
Pictures
File Edit Channels Components View
00:02:15 | Zoom: 100.0%, Size: 640x480
Objects Structure Browser
File Edit Objects Tags Texture
Blüte
Basis
Blütenblätter
Arc
Variation
Innen
Spline
Variation
Attributes Active Tool Snap Settings
Mode Edit User Data
Phyllotaxis Object [Blütenblätter]
Basic Coord. Object
Object Properties
Number 50
Soft Insert
Screw
Scale
Influence 1
Use Spline Orientation
Orientation
Level Of Detail None
Minimum Number 5
Weight 0.6
Coordinates
Position Size Rotation
Object Size Apply

Fazit

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

n=7, δ=22.5°

ω : A
p_1 : A → [&FL!A]/////'[&FL!A]///////'[&FL!A]
p_2 : F → S ///// F
p_3 : S → F L
p_4 : L → [’’’∧∧{-f+f+f-|-f+f+f}]

Rendering komplexer Geometrie?

Ausblick: aktuelle Forschungsthemen

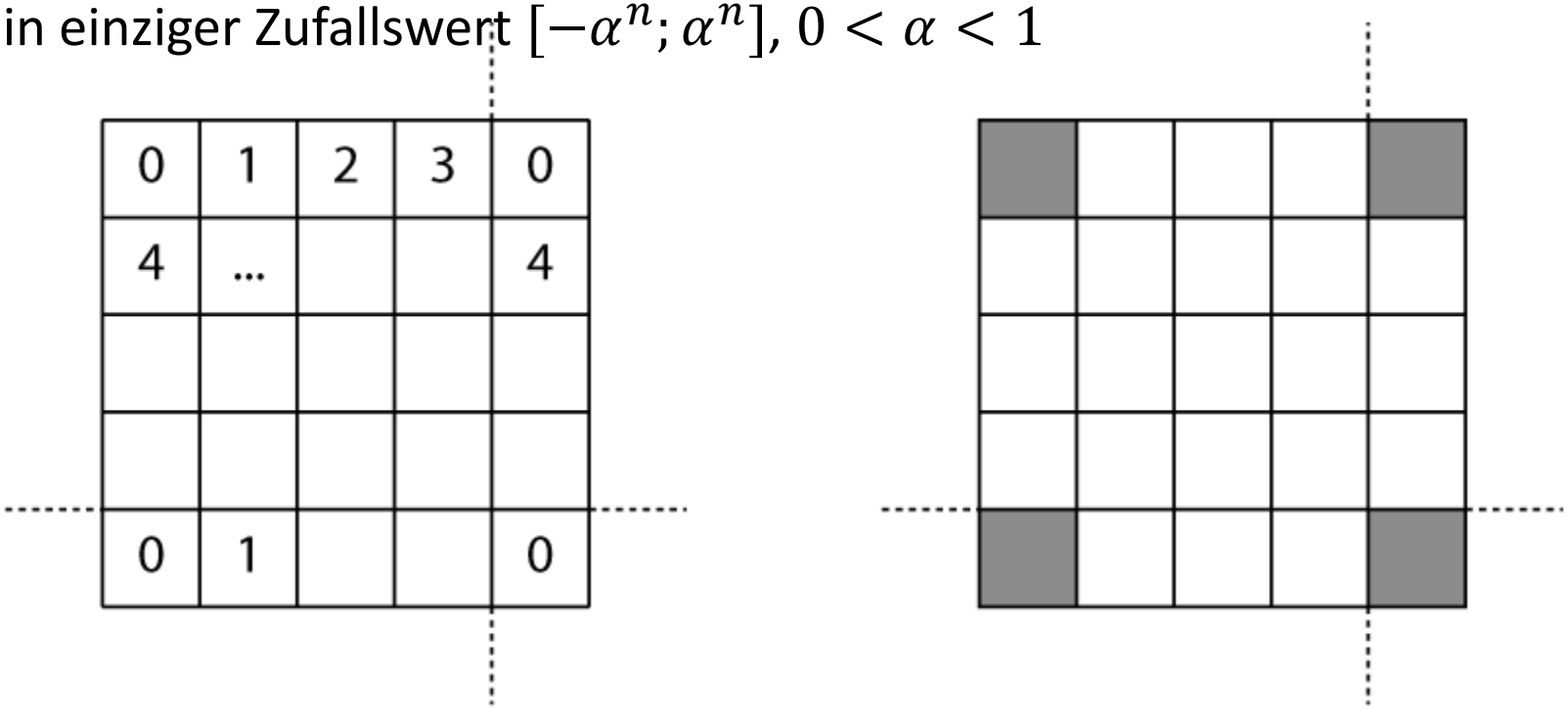



Bonus Material

Turbulenz Texturen (… schnell berechnet)

Diamond Square/Midpoint Displacement Algorithmus

- erzeuge zunächst tiefe Frequenzen durch weit voneinander entfernte Zufallswerte, höhere Frequenzen durch Interpolation und Offset
- hier: Zugriff auf 2D Textur durch Wrapping
- Initialisierung: Schritt $n = 0$,
  ein einziger Zufallswert $[-\alpha^n; \alpha^n]$, $0 < \alpha < 1$

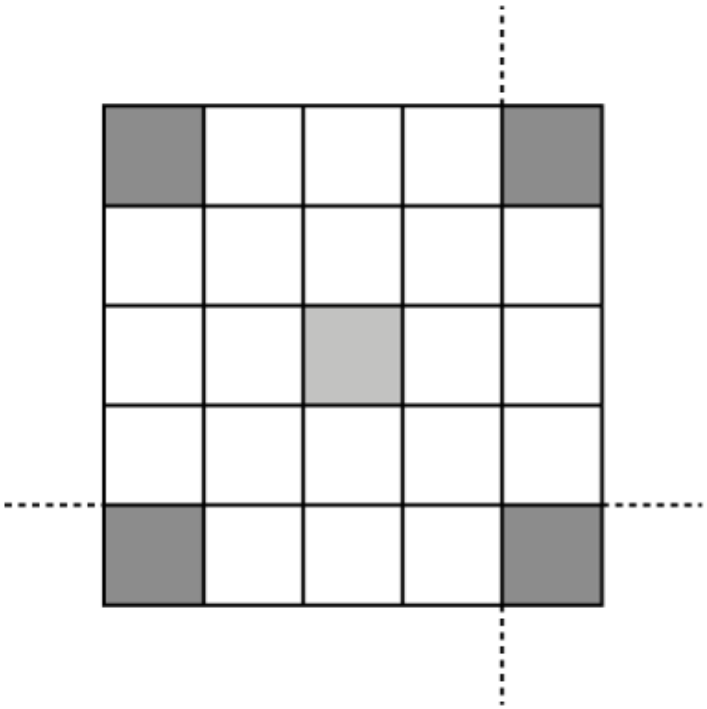


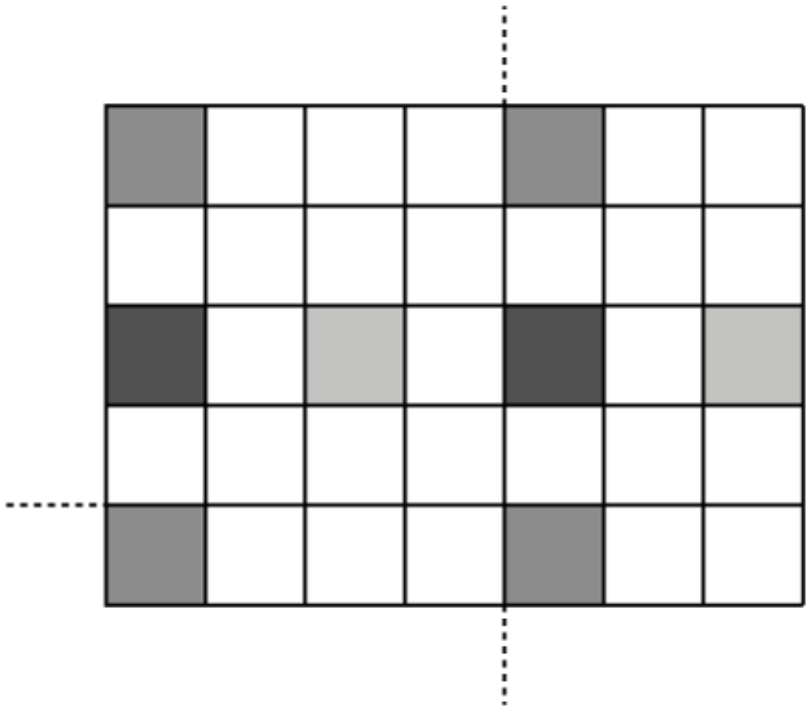
„The Definition and Rendering of Terrain Maps“ Gavin S.P. Miller, SIGGRAPH 1986

Turbulenz Texturen

Diamond Square/Midpoint Displacement Algorithmus

- Verfeinerungsschritt $n > 0$
  - mittle die 4 Werte an den Ecken und addiere Zufallszahl aus $[-\alpha^n; \alpha^n]$
  - mittle je 2 Werte (horizontal bzw. vertikal), addiere Zufallszahl aus $[-\alpha^n; \alpha^n]$
- Zugriff mit Wrapping ergibt automatisch eine Textur, die ohne sichtbaren Rand kachelbar ist (engl. tileable)

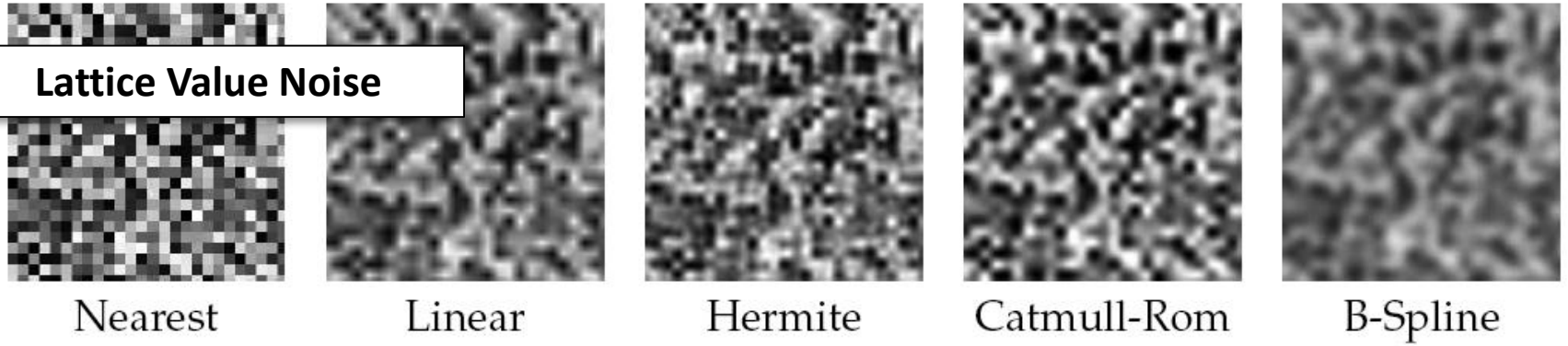




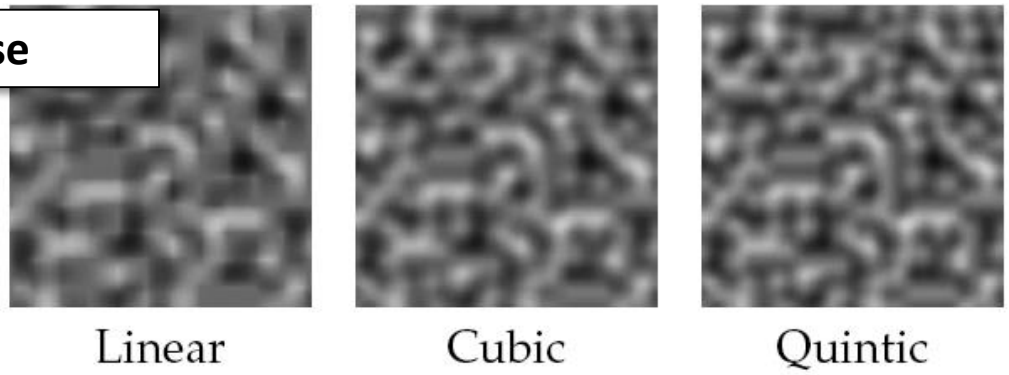
Exkurs: Noise-Funktionen und -Interpolation

kubische Interpolation

Lattice Value Noise

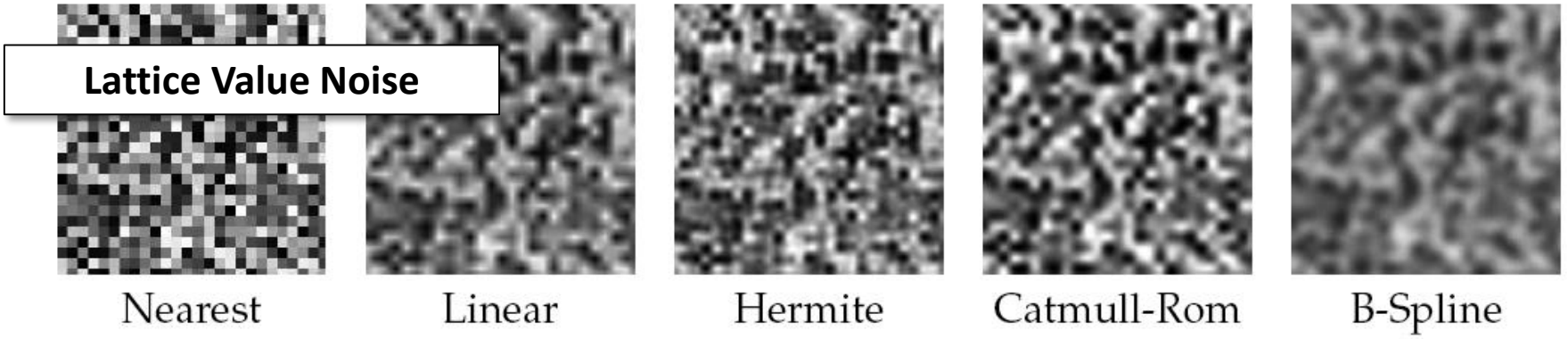


Gradient Noise

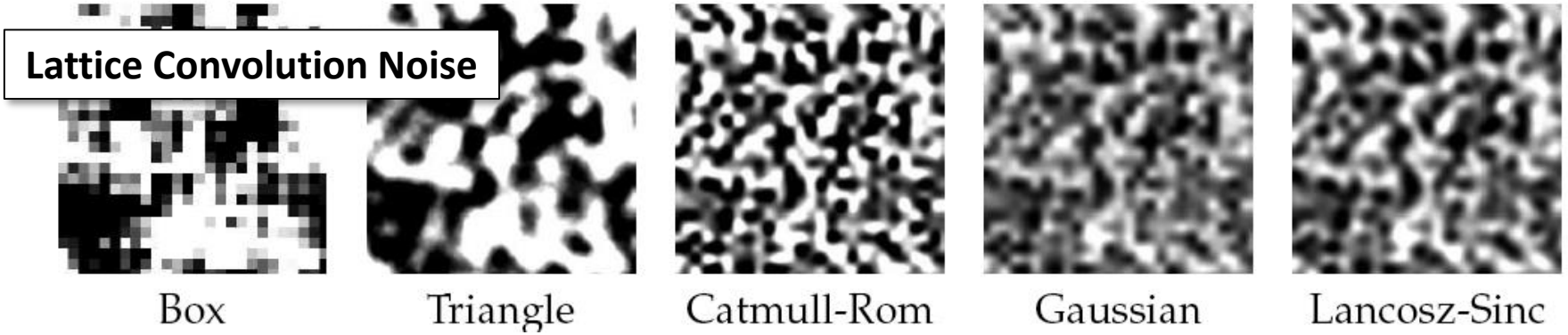


Exkurs: Noise-Funktionen und -Interpolation

- Lattice Value Noise:
  Gitterstruktur oft durch die Anisotropie der Filter sichtbar

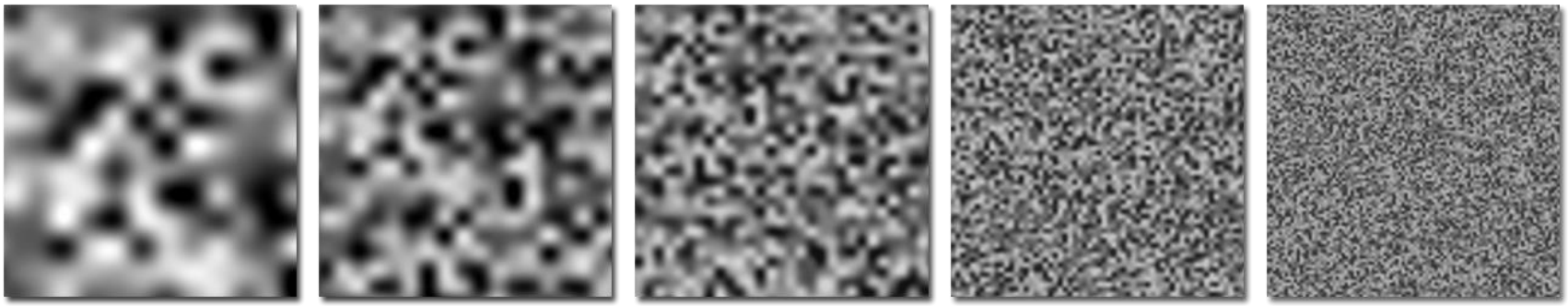


- Lattice Convolution Noise:
  Zufallswerte sind Impulse, die mit einem radialsymmetrischen Filter gefaltet werden



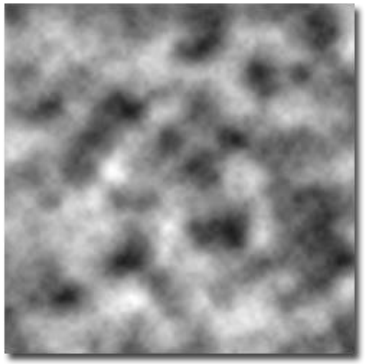
Kombination von Noise-Texturen

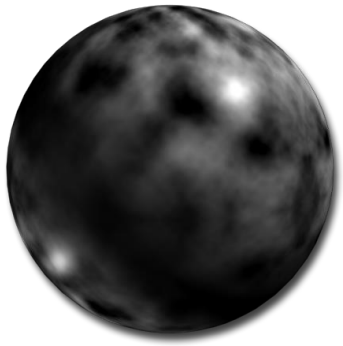
- Kombination kann additiv, multiplikativ, im Prinzip beliebig erfolgen
- Bsp. Turbulenz in 2D (also additiv)



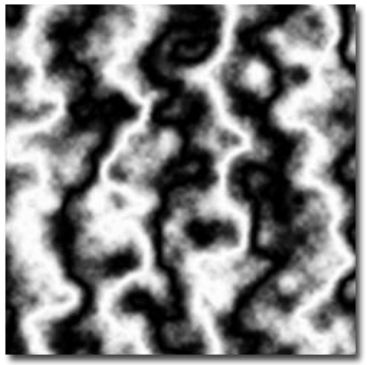
- Bsp. Turbulenz normal und mit anschließender trigonometrischer Funkt.

$$\sum_k \left({}^1/_2\right)^k n(2^k \mathbf{x})$$





$$\cos\left(\mathrm{x} + \sum_k \left({}^1/_2\right)^k n(2^k \mathbf{x})\right)$$



Prozedurale Shader

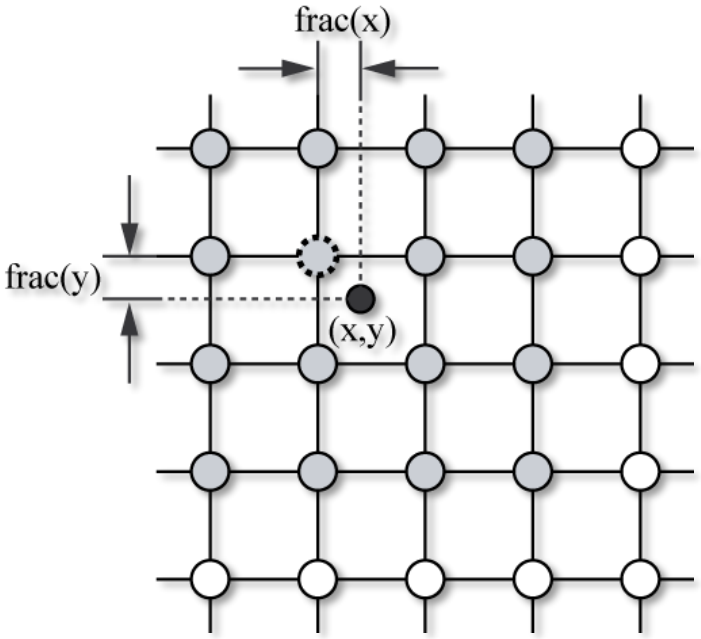
- trikubische Interpolation in 3D: 64 Textur Look-Ups

$$n(x,y,z) = \sum_{i=-1}^{2} \sum_{j=-1}^{2} \sum_{k=-1}^{2} l(\lfloor x \rfloor + i, \lfloor y \rfloor + j, \lfloor z \rfloor + k) a_{ijk}(x,y,z)$$

$$a_{ijk}(x,y,z) = R(i - \mathrm{frac}(x)) R(j - \mathrm{frac}(y)) R(k - \mathrm{frac}(z))$$

$$R(x) = \frac{1}{6}[\max(0, x+2)^3 - 4\max(0, x+1)^3 + 6\max(x, 0)^3 - 4\max(0, x-1)^3]$$

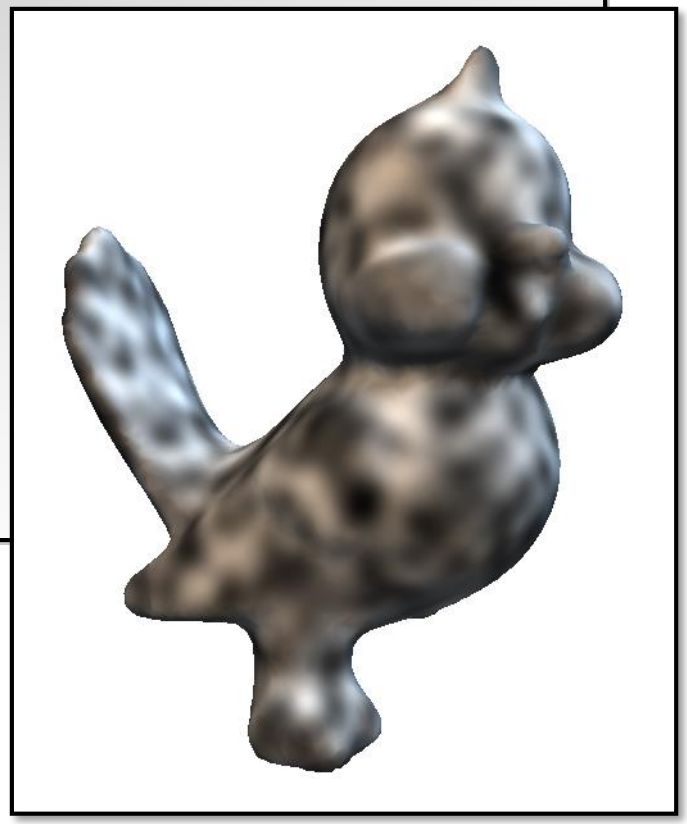
- Prinzip in 2D:



Prozedurale Shader

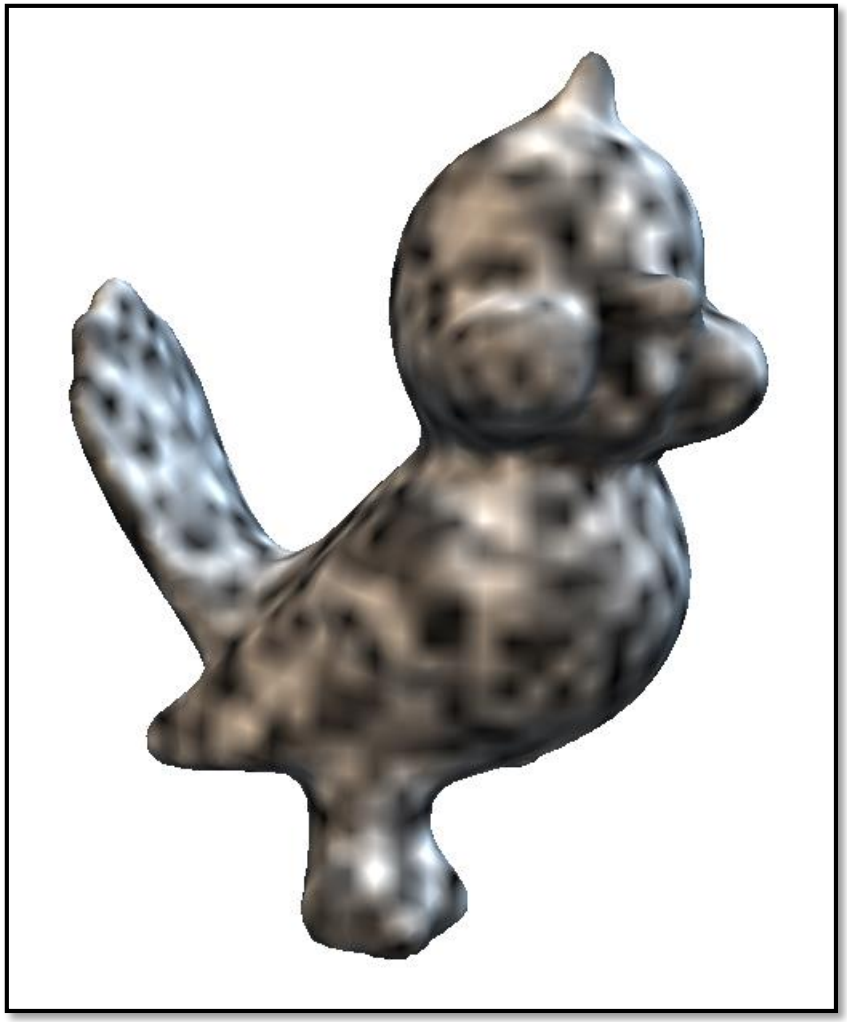
- trikubische Interpolation: 64 Textur Look-Ups

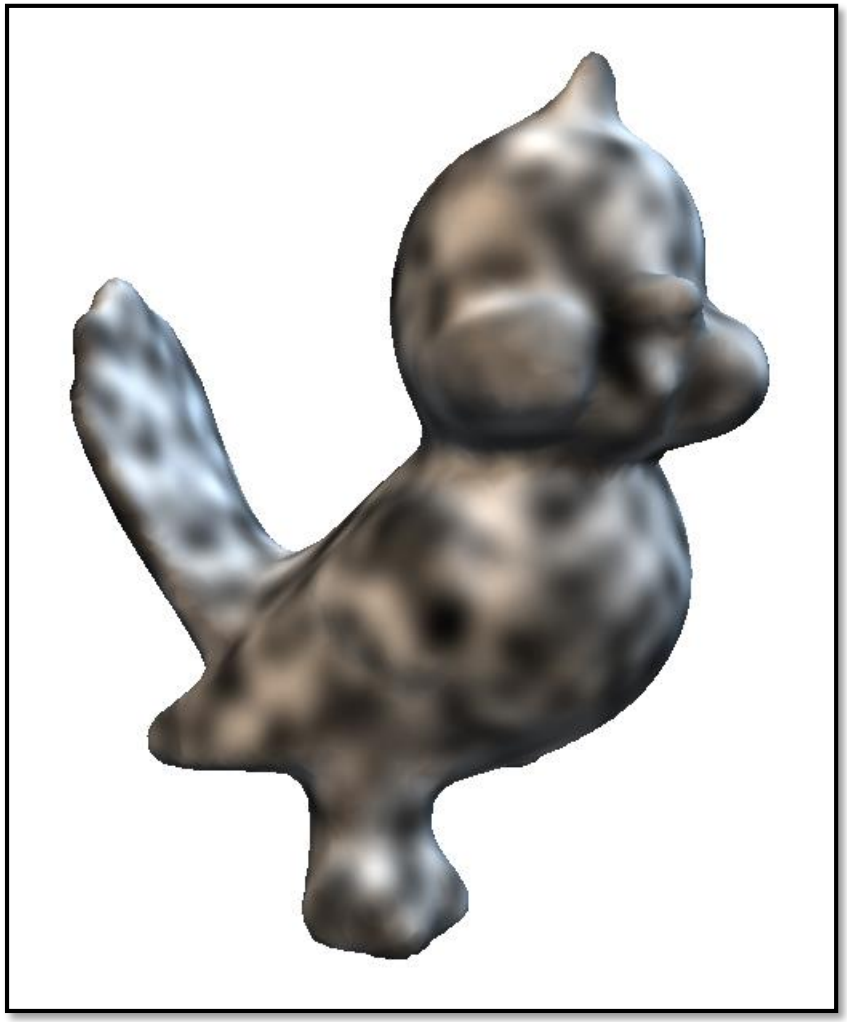
```
float tricubic_noise( vec3 v ) {
 int3 iv = ivec3( v );          // Ganzzahlige Koordinaten
 vec3 fr = v - iv;              // Nachkommastellen
 ...
 float n = 0.0;
 for ( i = -1; i <= 2; i++ )
  for ( j = -1; j <= 2; j++ )
    for ( k = -1; k <= 2; k++ ) {
     n += noise( iv + vec3(i,j,k) ) *
          R(i-fr.x)*R(j-fr.y)*R(k-fr.z);
}
 return n;
}
```



Prozedurale Shader

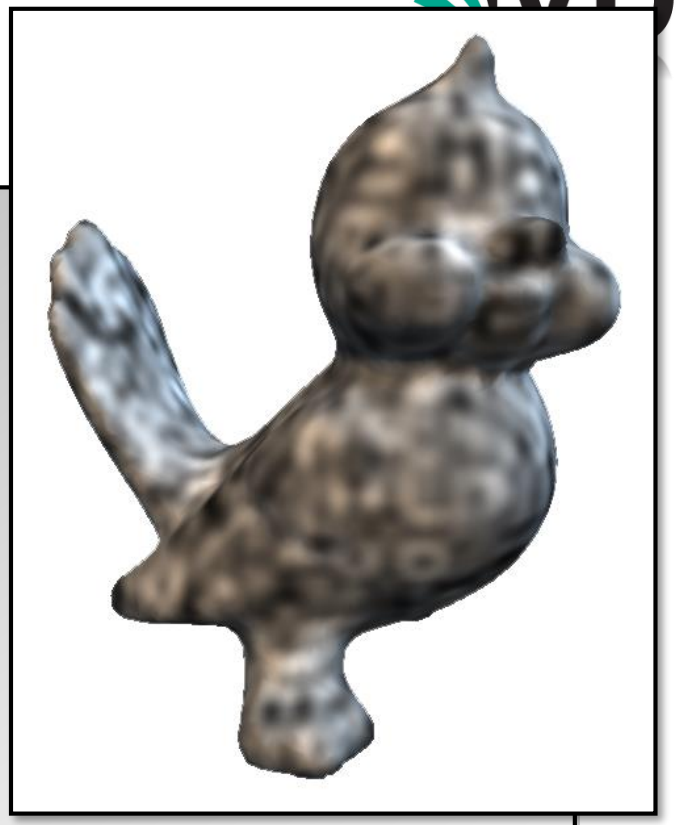
- trilineare vs. trikubische Interpolation





Prozedurale Shader

- 3D Gradient Noise



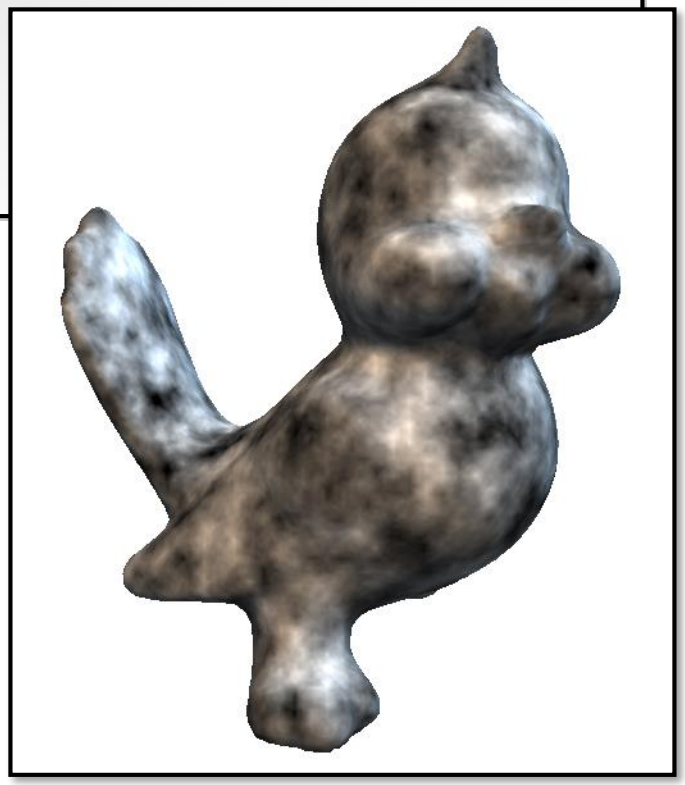
```
vec3 glattice( vec3 v, vec3 f ) {
  int x = floor( v.x ), y = ..., z = ...;
  return
    dot( -f, g[ ( z*64 + y ) * 64 + x ] );
}

float tricubic_gnoise( vec3 v ) {
  // Integer-Koordinaten & Nachkommateil
  vec3 iv = floor( v );
  vec3 fr = v - iv;
  ...
  a = glattice(iv+vec3(0,0,0), vec3(fr.x,fr.y,fr.z));
  b = glattice(iv+vec3(1,0,0), vec3(1-fr.x,fr.y,fr.z ));
  ...
  h = glattice(iv+vec3(1,1,1), vec3(1-fr.x,1-fr.y,1-fr.z));
  ...
  // anschließend trilineare Interpolation der Werte a bis h,
  // gemäß fr
  ...
}
```

Prozedurale Shader

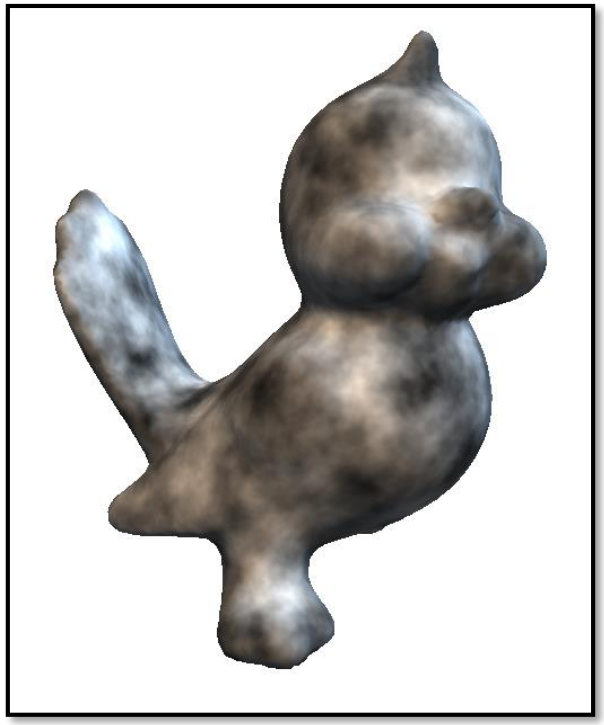
- Turbulence (hier mit trilinear interpolierter Noise)

```
// o = Anzahl Oktaven, L Lacunarity, H Fractal Increment
float turbulence( vec3 v, int o, float L, float H )
{
  float n = 0.0, w = 0.0;
  for ( int i = 0; i < o; i++ ) {
    n += trilinear_noise( v * pow( L, i ) ) / pow( L, i * H );
    w += 1.0 / pow( L, i * H );
  }
  return x / w;
}
```



Weiter Shader

- Noise als Basis für weitere Texturen



```
float turbulence(v,…)
```

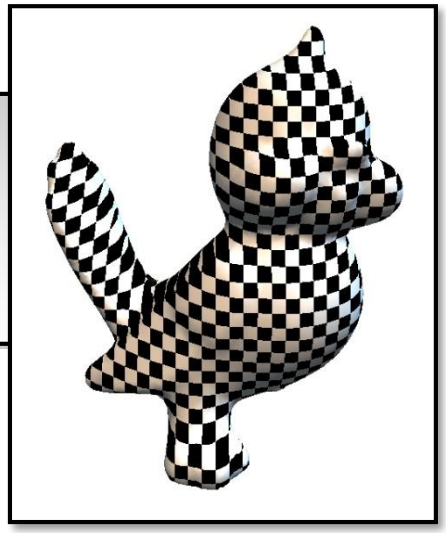
```
frac( scale *
  turbulence(v,…) )
```

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

Prozedurale Shader

- prozedurale Texturierung kann man direkt in einem Fragment Shader programmieren (Texturfilterung ist allerdings i.d.R. nicht trivial!)
- Schachbrett

```
// Eingabe: Oberflächenkoordinate x, y
int a = int( x * 30 );
int b = int( y * 30 );
vec3 color = ( a + b ) % 2;
```



- Hilfsfunktionen für Skalare und Vektoren
  - (komponentenweise) lineare Interpolation:
    $\texttt{lerp}(x, y, a) = x \cdot (1 - a) + y \cdot a$
  - Abrunden:
    $\texttt{floor}(x) = \lfloor x \rfloor$
  - Nachkommastelle:
    $\texttt{frac}(x) = x - \lfloor x \rfloor$

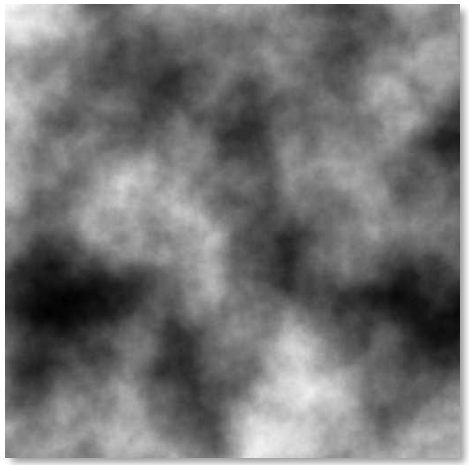
Exkurs: Höhenfelder aus Noise Funktionen

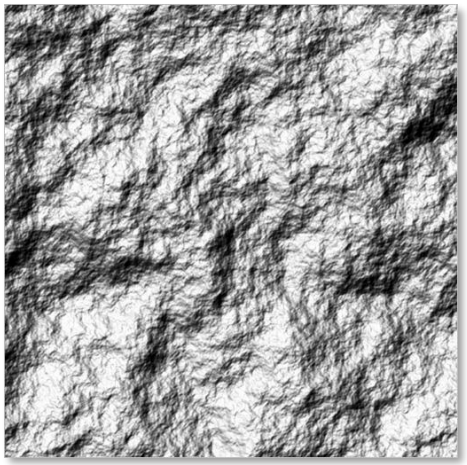
Allgemeine fractional Brownian Motion (fBM)

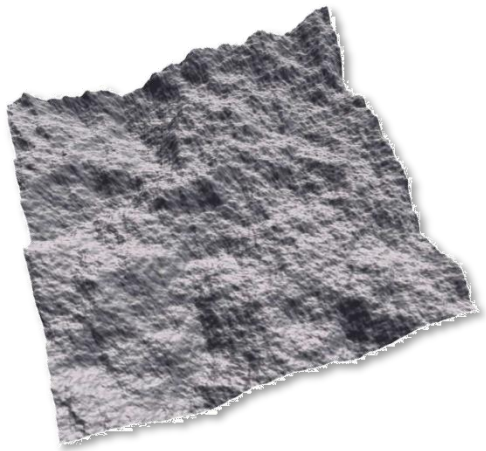
(nach F. Kenton Musgrave)

$$fBM(\mathbf{x}) = \sum_{i=0}^{k-1} n\left(\mathbf{x} \cdot L^i\right) L^{-iH}$$

- k Anzahl der Oktaven
- L Lacunarity (Frequenzabstand zwischen Oktaven), $L = {}^{1}/_{f}$
- H fraktales Rauschen (Gewichtung der Oktaven)
- gerade gesehener Spezialfall: Perlins Turbulence mit $H = 1, L = 2$





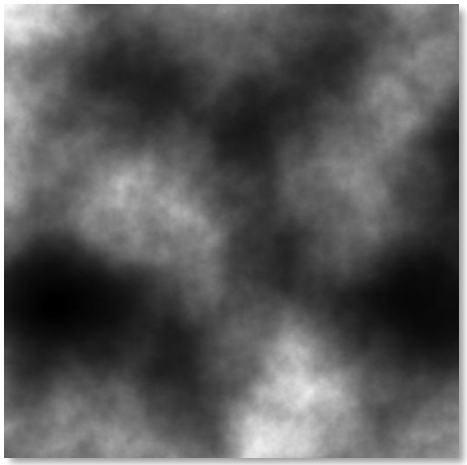


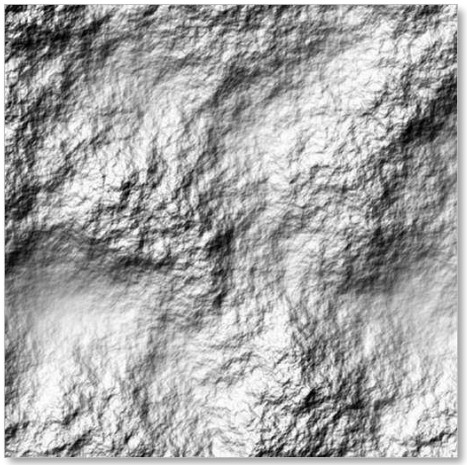
Exkurs: Höhenfelder aus Noise Funktionen

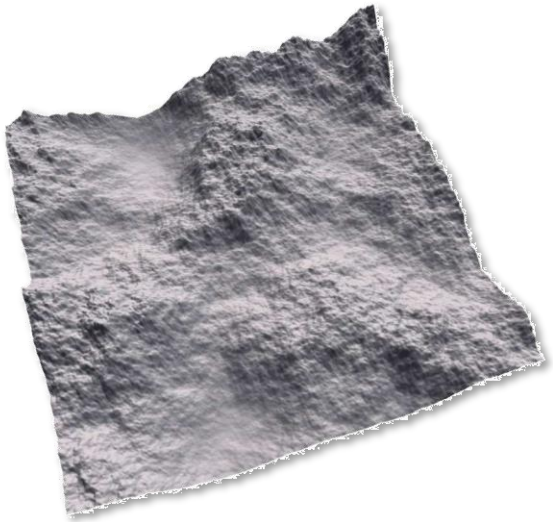
Multifractals: Multiplikative Verknüpfung

$$M(\mathbf{x}) = \prod_{i=0}^{k-1} \left(n\left(\mathbf{x} \cdot L^i + \mathbf{o}\right) L^{-iH} \right)$$

- Gewichtung von Oktaven abhängig vom Wert der vorherigen Oktave(n)
- schwache hohe Frequenzen, wo Niedrige geringere Werte annehmen
  → Nachempfinden von Erosion und Sedimentbildung







Exkurs: Höhenfelder aus Noise Funktionen

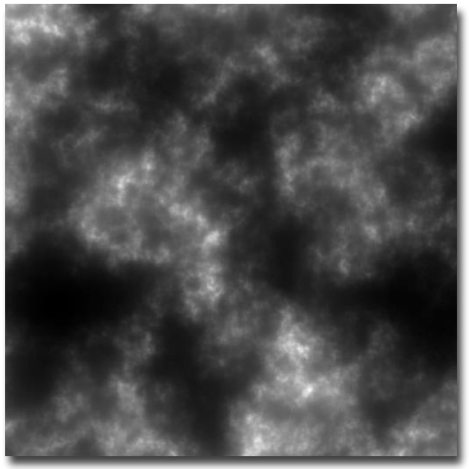
Heterogene Modelle

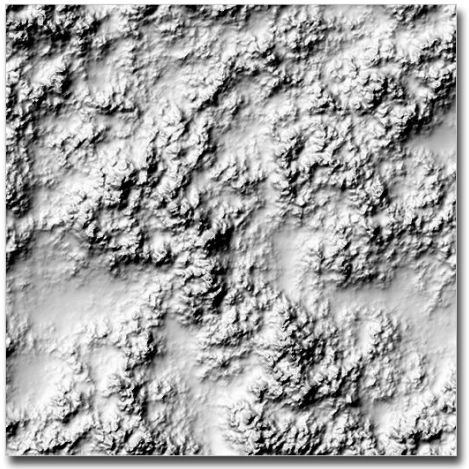
$$f_0(\mathbf{x}) = n(\mathbf{x} + \mathbf{o})$$

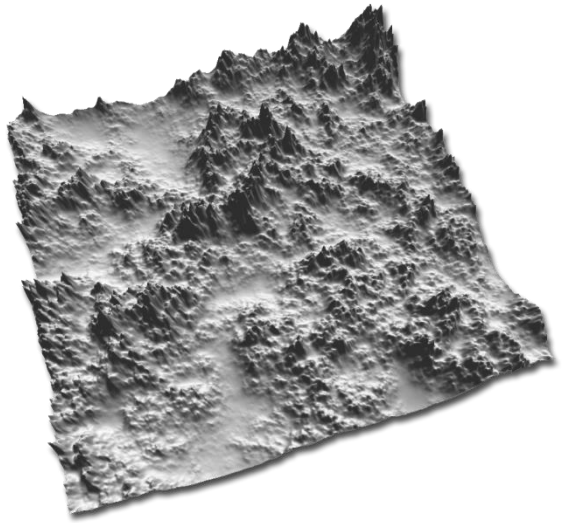
$$f_i(\mathbf{x}) = \left(n\left(\mathbf{x} \cdot L^i + \mathbf{o}\right)\right) L^{-iH} f_{i-1}(\mathbf{x})$$

$$H(\mathbf{x}) = \sum_{i=0}^{k-1} f_i(\mathbf{x})$$

- gegenseitige Beeinflussung und additive Verknüpfung der Oktaven







Exkurs: Höhenfelder aus Noise Funktionen

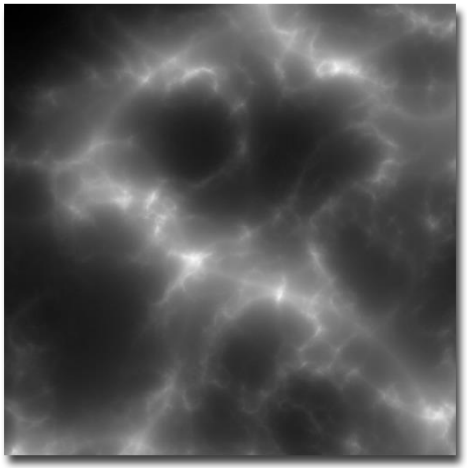
Heterogene Modelle mit Betragsfunktion

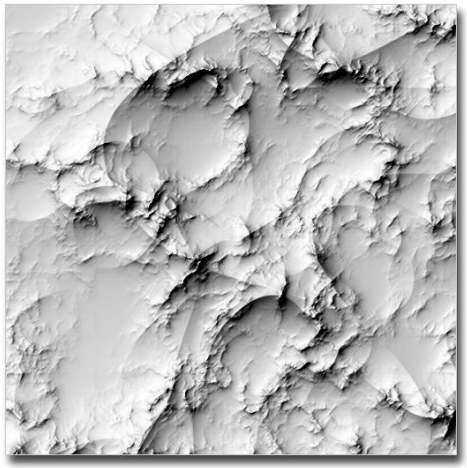
$$f_0(\mathbf{x}) = n(\mathbf{x} + \mathbf{o})$$

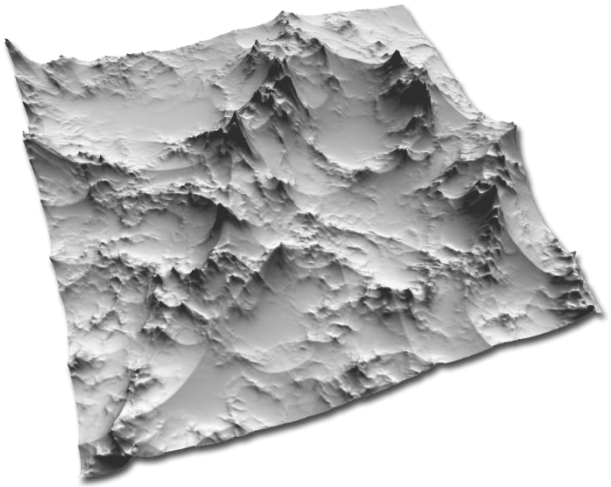
$$f_i(\mathbf{x}) = \left(n\left(\mathbf{x} \cdot L^i + \mathbf{o}\right)\right) L^{-iH} f_{i-1}(\mathbf{x})$$

$$H(\mathbf{x}) = \sum_{i=0}^{k-1} |f_i(\mathbf{x})| \qquad n(.) \in [-1; 1]$$

- es bilden sich Bergkämme (Ridges)







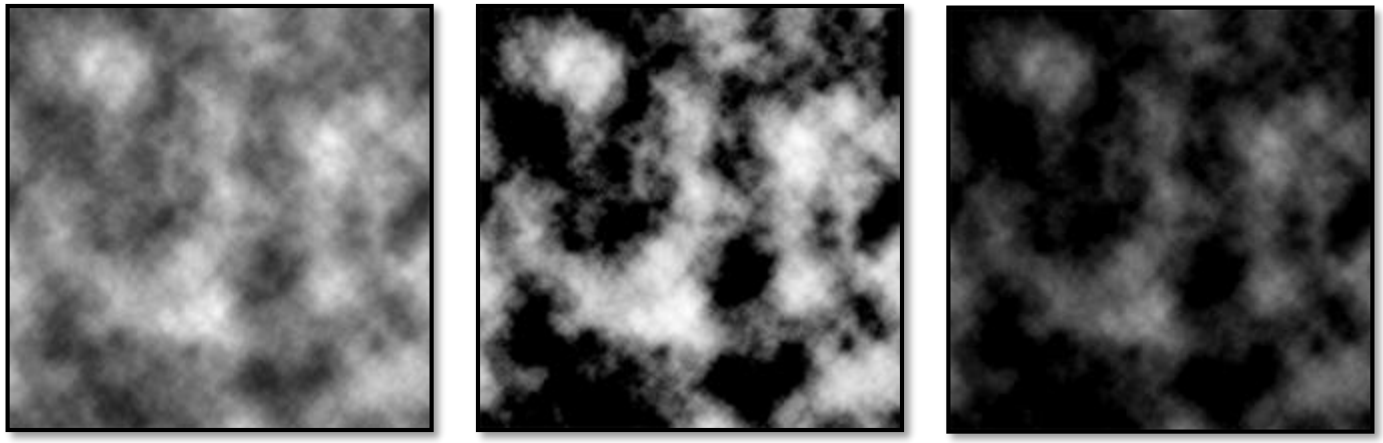
Exkurs: Prozedurale Texturierung von Landschaften

- effizient und realistisch, angepasst an Landschaftscharakteristika
- Grundidee:
  - Oberfläche der Erde besteht aus Schichten: Fels, Sand/Erde, Grass…
  - u.U. Abhängigkeit (Grass wächst nur auf Erde etc.)

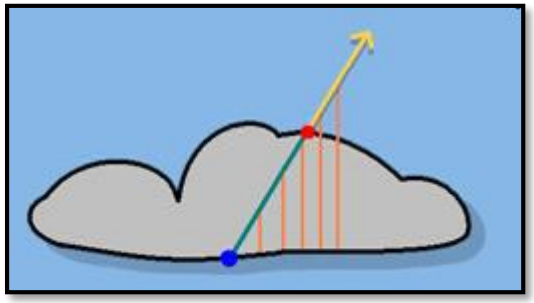
Exkurs: 2.5D Wolken

Wolkendicke durch animierte 2D Turbulenz Textur

- Animation über Offsets
- Höhe der Wolken: $h(\mathbf{x}) = \max(0, n(\mathbf{x}+\mathbf{o}) - c), c = const$
- Transparenz: $\alpha(\mathbf{x}) = 1 - s^{n(\mathbf{x}+\mathbf{o})}$, Schärfe der Wolkenränder s



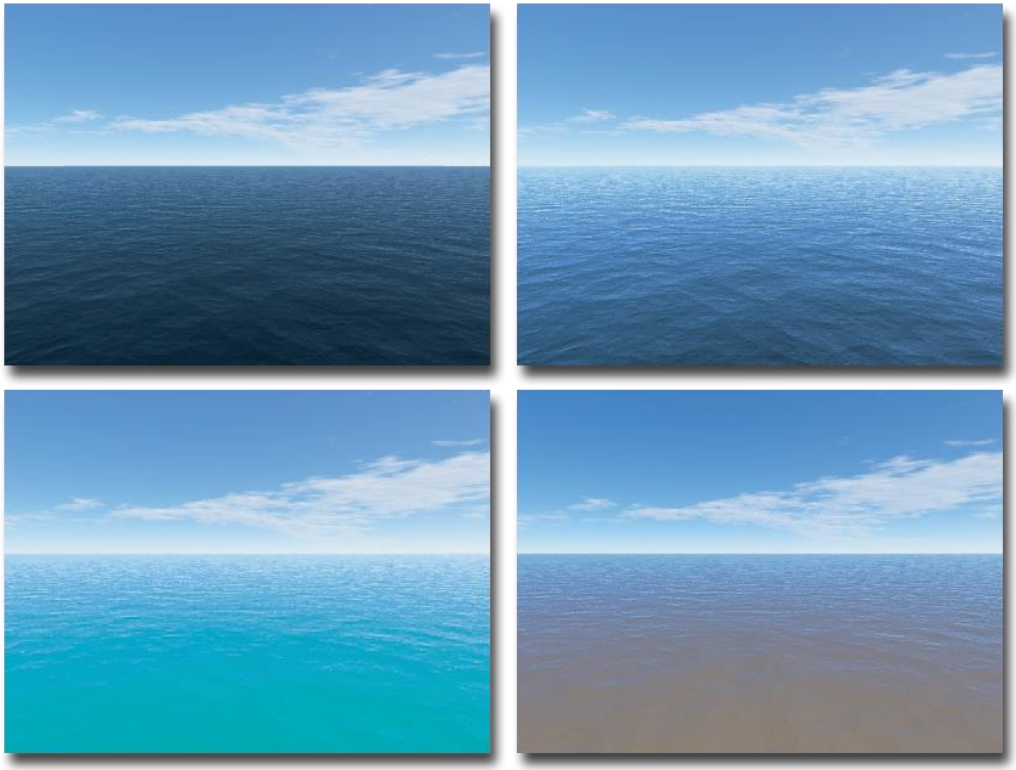
- Strahlverfolgung: je weiter der Weg durch die Wolke, desto weniger Licht

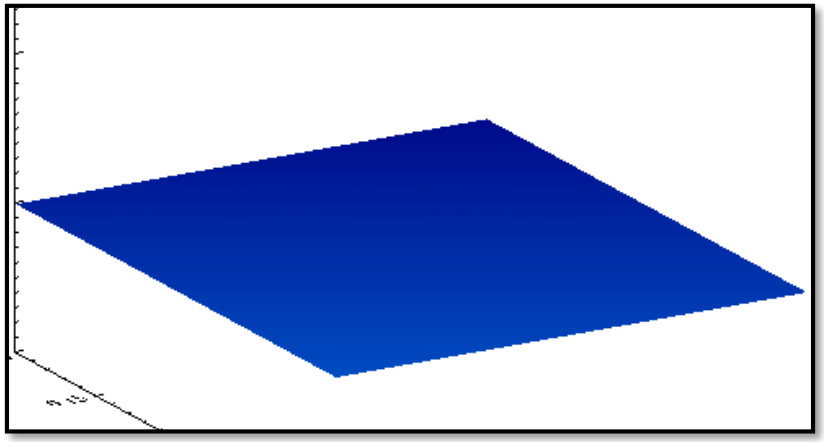


Exkurs: 2.5D Wolken

Exkurs: Darstellung von Gewässern

- Modelle für Lichtstreuung in Gewässern
- Modelle für Wasserbewegung:
  - empirische Modelle für spektrale Wellenverteilung
  - Flüssigkeitssimulation, Shallow-Water-Equations (2D)



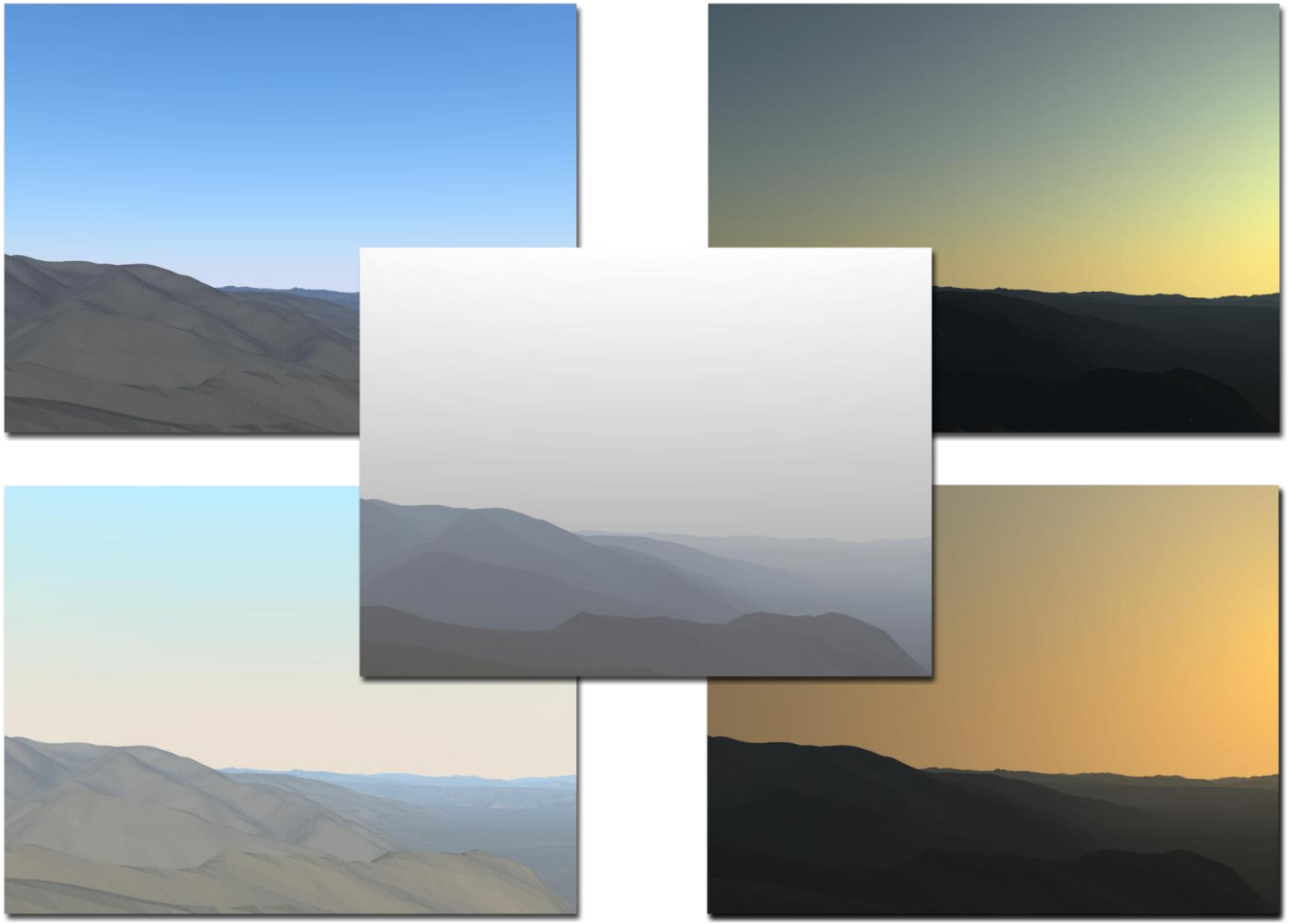


Exkurs: Darstellung von Gewässern

Exkurs: Darstellung von Gewässern

Exkurs: Sonne und Erdatmosphäre

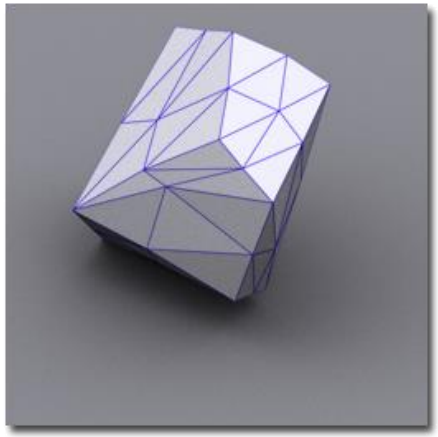
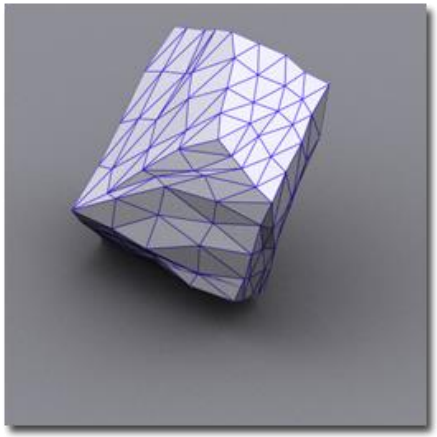
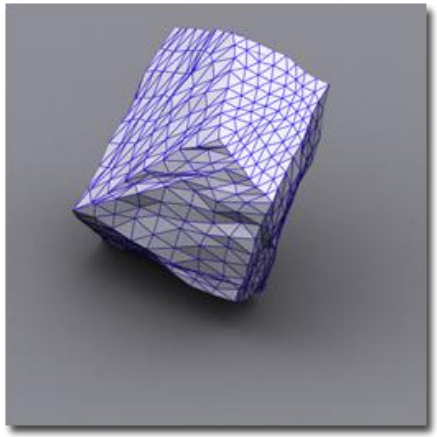
- Modelle für Lichtstreuung (Arcot Preetham)



Exkurs: Prozedurale Landschaften

Ausblick: Felsen erzeugen

- konvexe Hülle einiger Zufallspunkte
- Unterteilen und zufällige Verschiebung neuer Vertices


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 mit
  - Zeichen (Symbolen)
  - Produktionsregeln
  - wichtiger Unterschied: Produktionsregeln werden parallel angewendet, um die biologischen Prozesse nachzubilden

- 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 $P \subset (V^* \backslash \Sigma^*) \times V^*$

- ein L-System alleine macht noch kein geometrisches Objekt!
  - dazu wird noch eine Interpretationsvorschrift benötigt

L-Systeme

Unterscheidung nach folgenden Kriterien

- **kontext-frei**: Produktionsregeln $P \subset (V \backslash \Sigma) \times V^*$
- **kontext-sensitiv**: enthält Produktionsregeln die ein Symbol nur ersetzen, wenn es bestimmte Nachbarsymbole hat

- **deterministisch**: genau eine Produktionsregel für jedes Symbol
- **stochastisch**: gibt es mehrere anwendbare Produktionsregeln, so wird eine Regel mit einer bestimmten Wahrscheinlichkeit angewendet

- **D0L-Systeme** sind kontext-frei und deterministisch
  - es wird eine vorher festgelegte Anzahl an Ersetzungsschritten durchgeführt

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
  - aba
  - $abaab$
  - $abaababa$

Grafische Interpretation

- L-Systeme waren als formale Theorie für biologische Entwicklung gedacht – geometrische Aspekte wurden zunächst nicht berücksichtigt

- grafische Interpretation der Zeichenketten basierend auf Turtle-Grafik
  - damit war fraktales Modellieren möglich

- Turtle-Grafik (Programmiersprache Logo):
  - die Schildkröte ist ein Cursor mit
    - Position (x, y)
    - 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

Turtle-Grafik

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

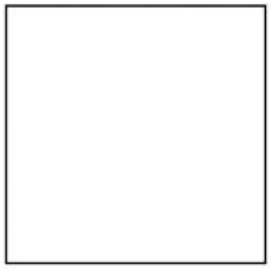
- $+$ Rechtsdrehung um den Winkel δ
  - neue Orientierung: $\alpha' = \alpha + \delta$

- $-$ Linksdrehung um den Winkel δ
  - neue Orientierung: $\alpha' = \alpha - \delta$

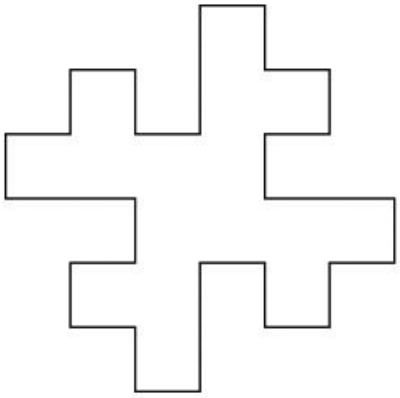
Turtle-Grafik

Beispiel

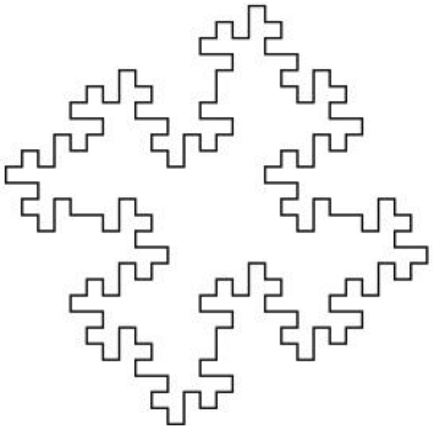
- $G = (V, \Sigma, S, P)$ mit
  - Alphabet $V = \{F, +, -\}$
  - Startwort $S = F + F + F + F$
  - Produktionsregel $P = \{F \mapsto F + F - F - FF + F + F - F\}$
  - $\alpha = 90°$



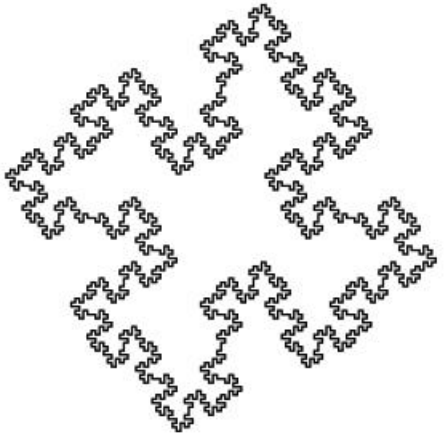
n = 0



n = 1



n = 2

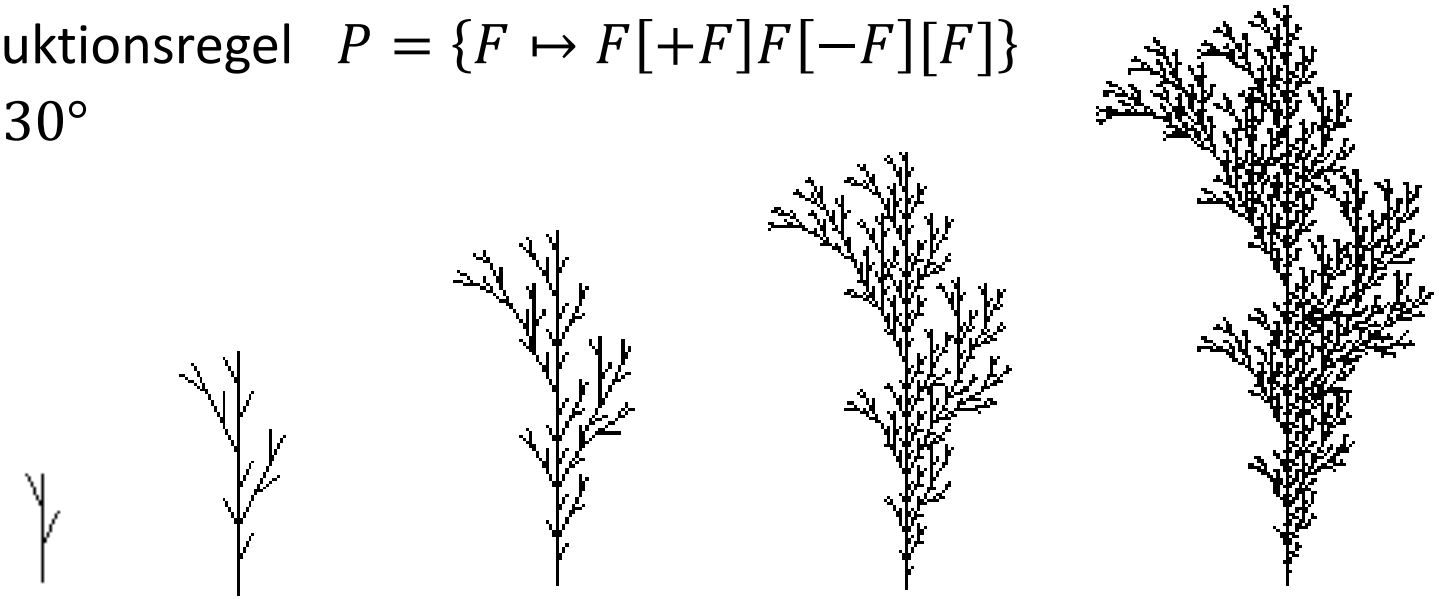


n = 3

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 $P = \{F \mapsto F[+F]F[-F][F]\}$
  - $\alpha = 30°$



Turtle-Grafik in 3D

Erweiterung auf 3D

- aktuelle Orientierung repräsentiert durch 3 Vektoren:
  Heading $\mathbf{H}$, Left $\mathbf{L}$, Up $\mathbf{U}$
- 3 Rotationsmatrizen $R_{\mathbf{H}}$, $R_{\mathbf{L}}$, $R_{\mathbf{U}}$ für Drehung um den Winkel δ um die jeweilige Achse
- Symbole zur Kontrolle der Orientierung (nur eine mögliche Konvention)
  - $+$, $-$ Links-/Rechtsdrehung über Matrix $R_{\mathbf{U}}(\delta)$
  - &, ^ Drehung nach oben bzw. unten über Matrix $R_{\mathbf{L}}(\delta)$
  - \, / Links-/Rechtsdrehung um die eigene Achse über $R_{\mathbf{H}}(\delta)$
  - | Umkehren mit $R_{\mathbf{U}}(180°)$

L-Systeme zur Entwicklung von Pflanzen

Simulation der Evolution von künstlichen Pflanzen

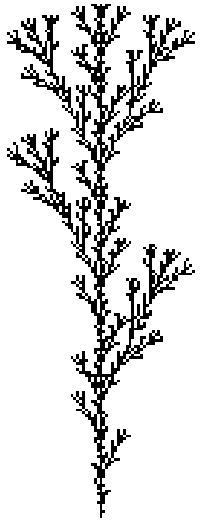
- “Erbgut” (Genotyp): einfaches D0L-System
  - $S = F,\ P = \{F \mapsto F[-F]F[+F][F]\}$
  - „Chromosom“, „Erbinformation“: $F \mapsto F[-F]F[+F][F]$
- „Phänotyp“: 2D verzweigte Strukturen resultierend aus der Ersetzung und grafischen Interpretation des L-Systems
- „genetische Operatoren“: Rekombination und Mutationsoperatoren, die die syntaktischen Regeln einhalten

Rekombination

- Austausch der Bausteine $[-F-F]$ und $[+F]$

Eltern

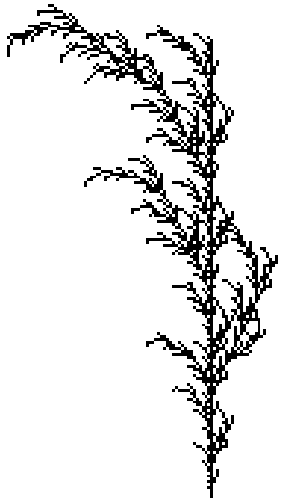
F[-FF]+[FFF]-FF**[-F-F]**



F[+F]+[-F-F]-FF**[+F]**[-F][F]

Nachkommen

F[-FF]+[FFF]-FF**[+F]**



F[+F]+[-F-F]-FF**[-F-F]**[-F][F]

Mutation

- Mutation des Symbols F bzw. Blocks $[FFF]$

Symbol Mutation

F[+F]+[+F-F-F]-F**F**[-F-F]

F[+F]+[+F-F-F]-F**[-F]**[-F-F]

FF[+FF][-F+F]**[FFF]**F

FF[+FF][-F+F]**[-F]**F

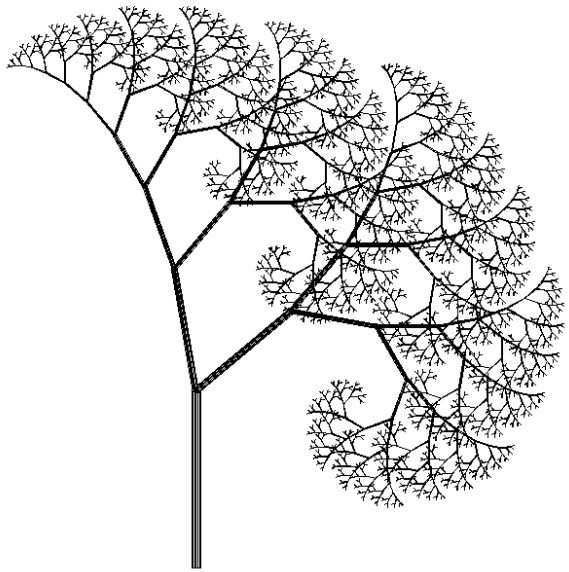
Block Mutation

Parametrisierte L-Systeme

Verwende weitere Symbole um die grafische Interpretation zu steuern

- Zustand $\mathbf{H}$, $\mathbf{L}$, $\mathbf{U}$, Länge einer Linie s (zu Beginn $s = 1$)
- Alphabet $V = \{F, A, L\}$
- Startwort $S = L(1.0)F(1.0)A$
  - $L(x)$ Änderung der Länge einer Linie mit $s' = s \cdot x$
  - $F(d)$ zeichnet Linie der Länge $d \cdot s$

- Produktionsregel $P = \{A \mapsto F(1.0)L(0.7)[R_{\mathbf{U}}(50°)A][R_{\mathbf{U}}(-10°)A]\}$



Stochastische L-Systeme

Verwende weitere Symbole um die grafische Interpretation zu steuern

- Zustand $\mathbf{H}$, $\mathbf{L}$, $\mathbf{U}$, Länge einer Linie s (zu Beginn $s = 1$)
- Alphabet $V = \{F, A, L\}$
- Startwort $S = L(1.0)F(1.0)A$
  - $L(x)$ Änderung der Länge einer Linie mit $s' = s \cdot x$
  - $F(d)$ zeichnet Linie der Länge $d \cdot s$
- Produktionsregel mit einer Zufallszahl $0 \leq \psi < 1$

$$P = \left\{ A \mapsto \begin{cases} FL(0.7)[R_{\mathbf{U}}(50°)A][R_{\mathbf{U}}(-10°)A], & \text{wenn } \psi < 0.5 \\ FL(0.7)[R_{\mathbf{U}}(-50°)A][R_{\mathbf{U}}(10°)A], & \text{wenn } \psi \geq 0.5 \end{cases} \right\}$$