

Prozedurale Generierung

Einführung in die Computergrafik (für Augmented Reality)

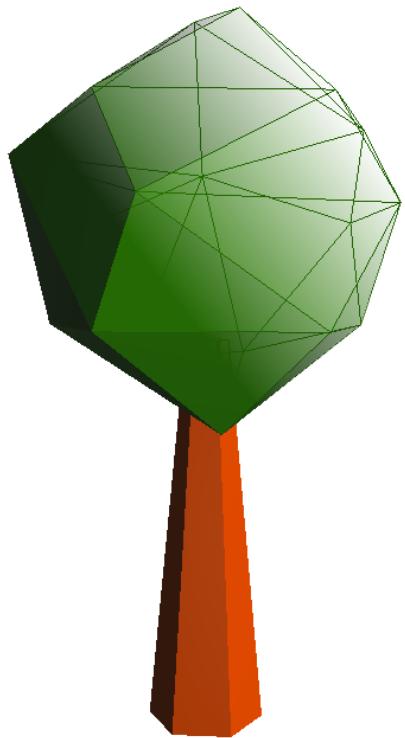
Folien basieren teilweise aus dem Material von <http://pcgbook.com/>

Wiederholung

- Partikel
- Starrkörper

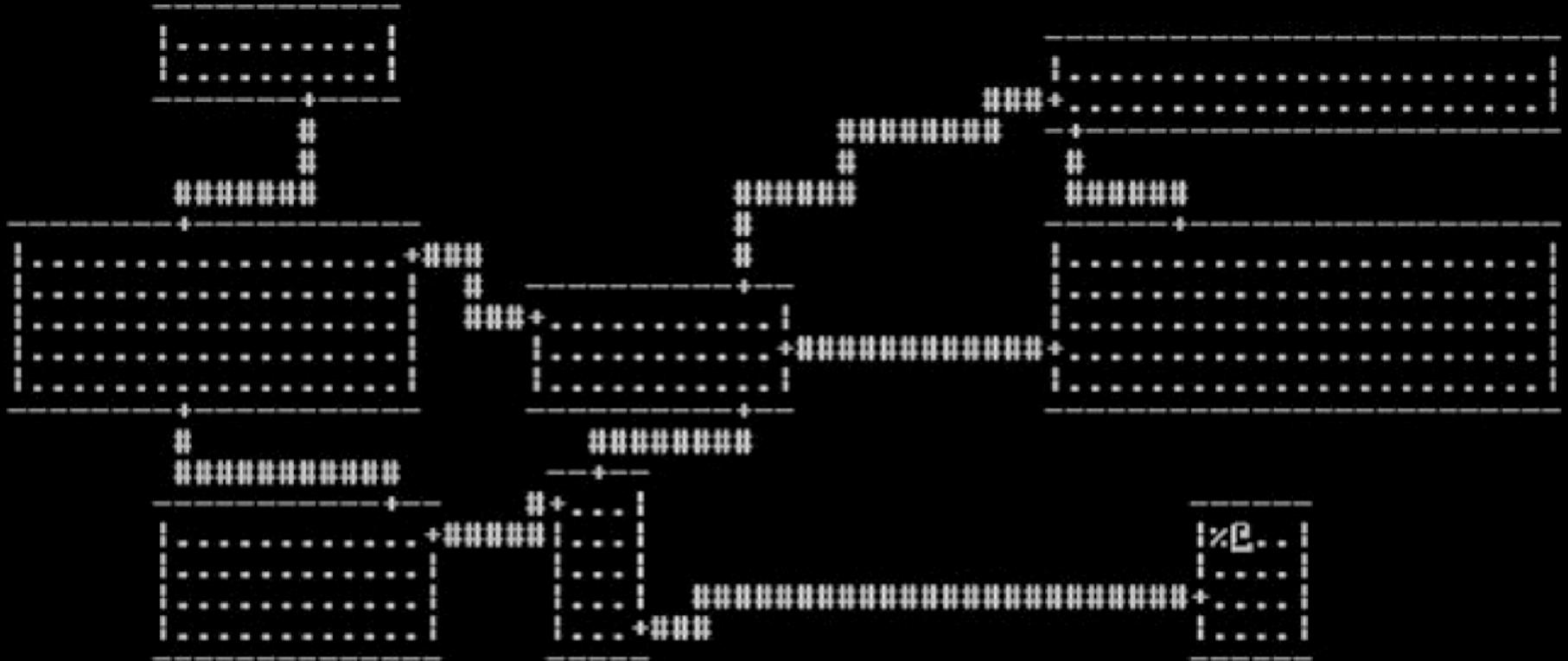
Agenda

- Einführung
- Terrain
- Dungeons
 - BSP-Bäume
 - Zelluläre Automaten
- Objekte
 - Lindenmayer System
 - Shape Grammar



Einführung

Rogue



Level: 1 Gold: 264 Hp: 19(25) Str: 16 Ac: 6 Exp: 3/30

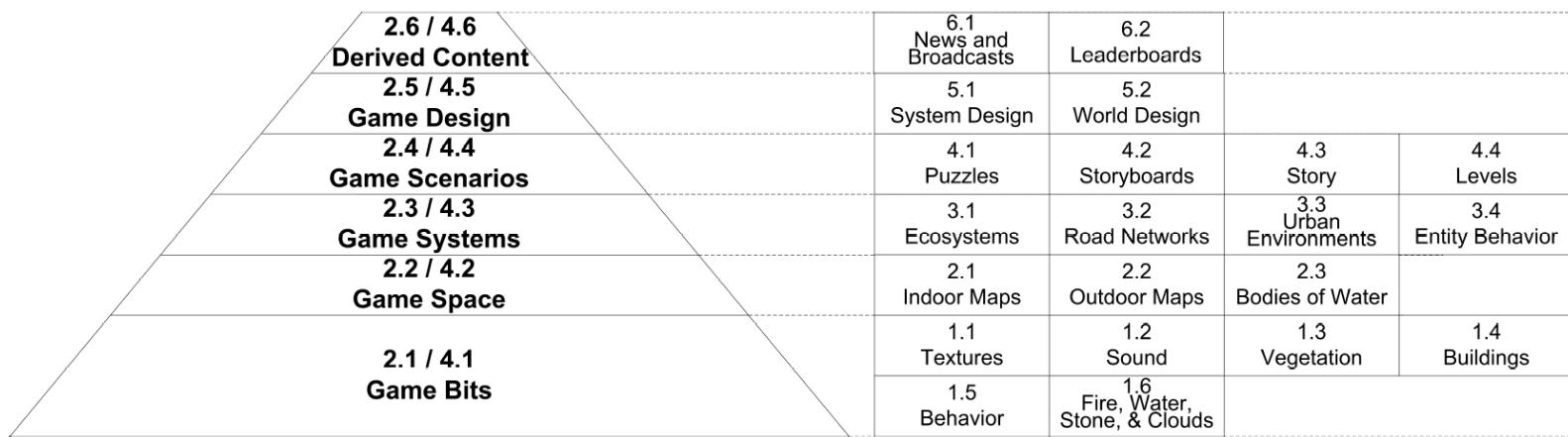
Rogue (Universität Berkeley, 1980): Level-Generierung

Diablo



Diablo (Blizzard, 1996): Generierung von Levels und Gegenständen

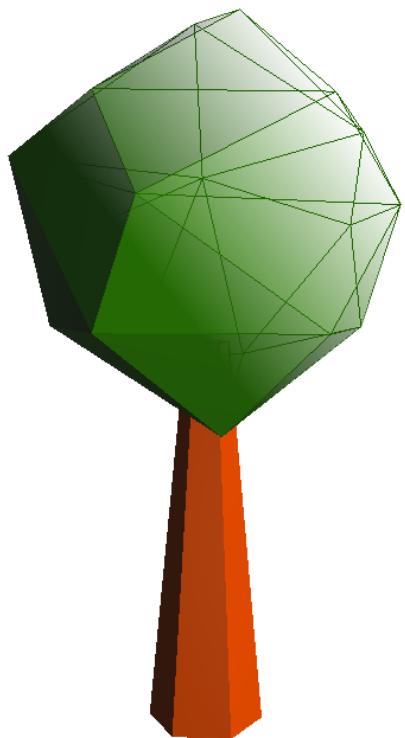
Taxonomie



Quelle: [2]

Was lässt sich prozedural generieren?

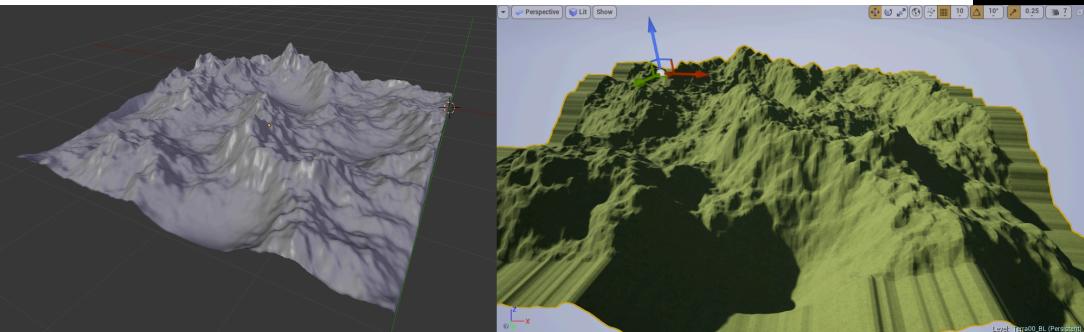
- Spielelemente
 - Texturen
 - Sounds
 - Vegetation
 - Verhalten (z.B: Zerbrechen)
 - Feuer, Wasser, Rauch, ...
- Raum
 - Innenräume (z.B. Dungeons)
 - Außenräume (z.B. Höhenfelder)
- Systeme
 - Ökosysteme
 - Straßen
 - Urbane Umgebungen
 - Agentenverhalten
- Systeme
 - Systeme
 - Straßen
 - Urbane Umgebungen
 - Agentenverhalten
- Szenarien
 - Puzzle
 - Storyboards
 - Story
 - Level
- Game Design
 - System (z.B. Regeln)
 - Welt (z.B. Setting)
- Weitere
 - News, Highscores



Terrain

Modellierung von Höhenfeldern

- meist 2½D Datenstruktur
 - xz-Ebene zur Parametrisierung
 - y-Wert als Höhe
 - Nachteil: keine Überhänge, Höhlen
- Alternative: Voxel-System
 - 3D Gitter, jede Zelle = Voxel



Unreal Engine: Höhenfeld

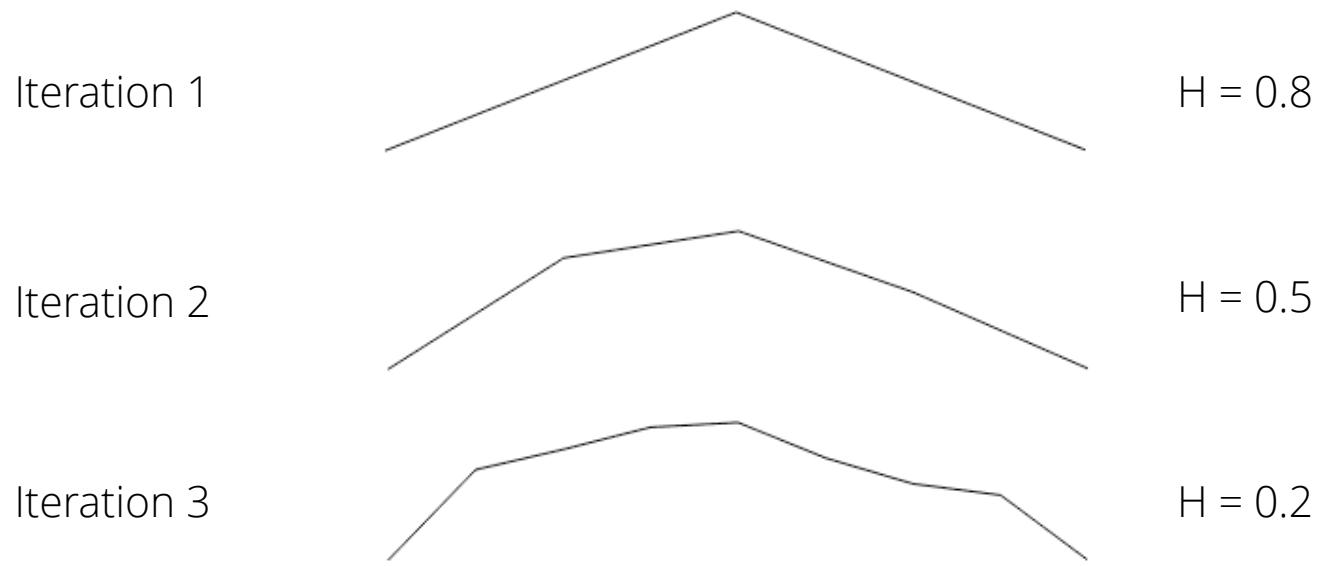


Voxel-Engine (YouTube, Nutzer RileyStarcraft)

Mittelpunkt-Verschiebung

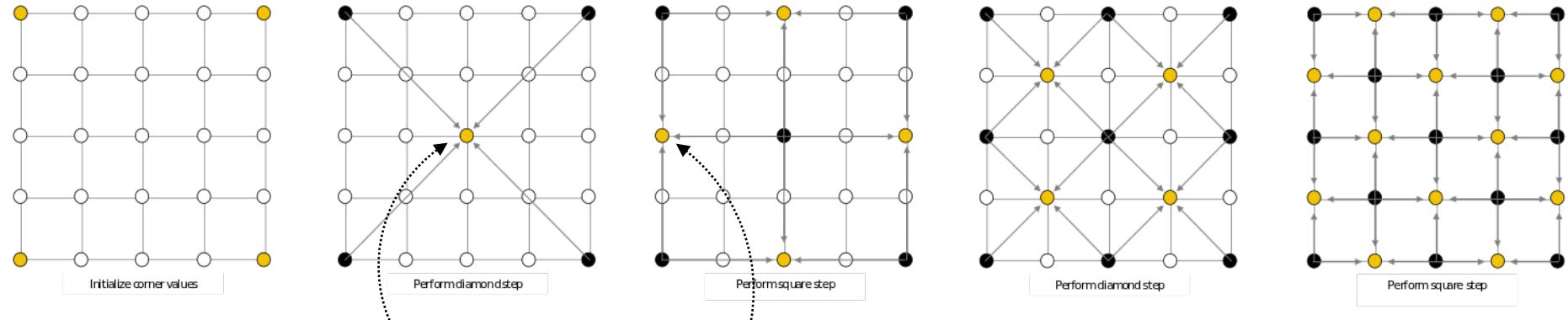
- Herleitung in 2D
- Algorithmus generiert Silhouetten durch Verschiebung entlang eines Polygons
- REPEAT bis Auflösung ausreichend
 - finde Mittelpunkt zwischen benachbarten Punkten
 - verändere Position um Zufallswert in y-Richtung
 - verwende dazu Parameter H:
 - Faktor zur Skalierung des Zufallswertes
 - H wird in jeder Iteration kleiner

Mittelpunkt-Verschiebung



Diamond-Square

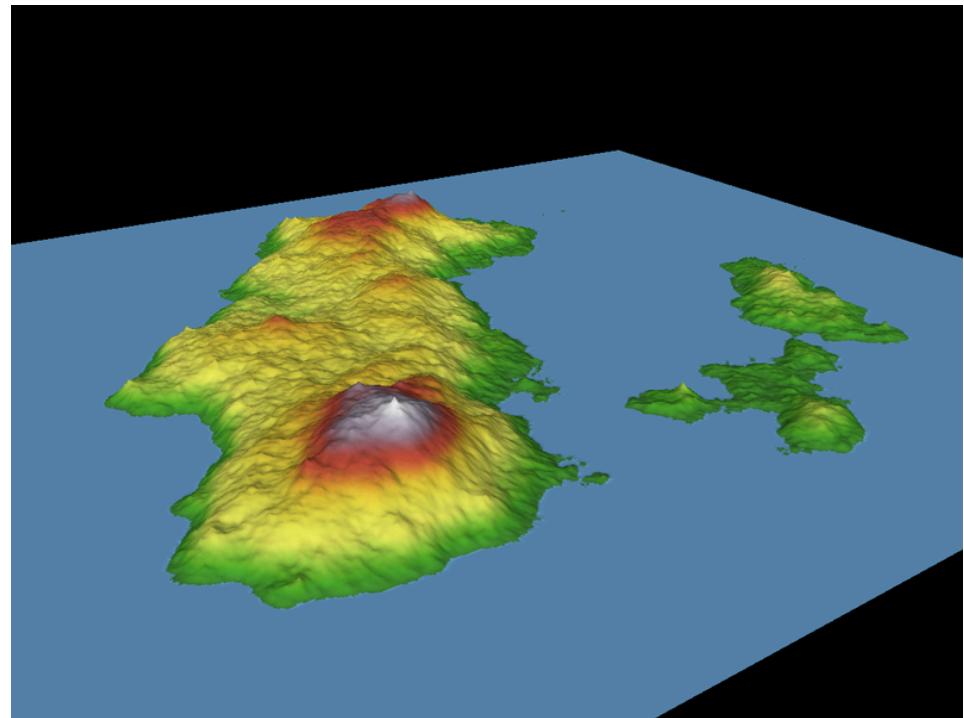
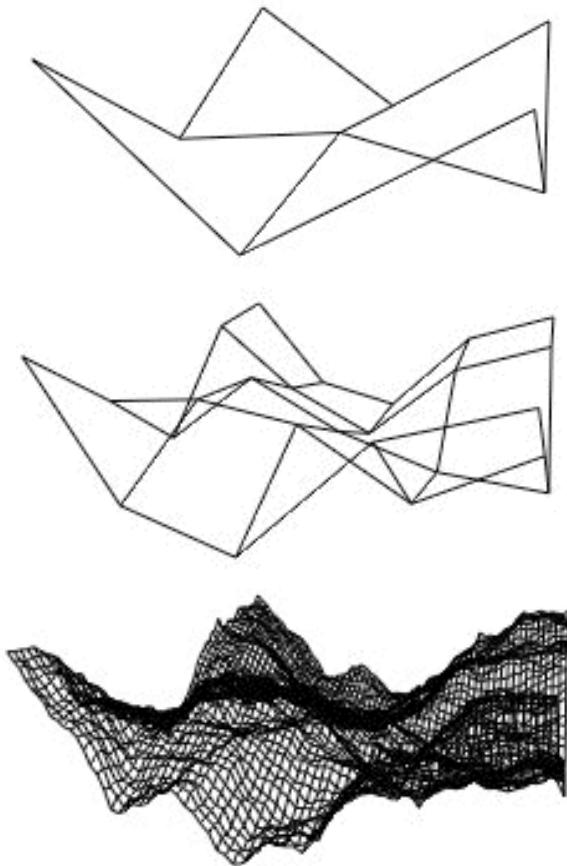
- Erweiterung der Mittelpunkt-Verschiebung auf 3D [6]
- Verschiebung beim Mittelpunkt von Quadraten
 - anstelle von Polygonsegmenten
- In jeder Iteration
 - Diamond-Schritt: squares horizontal
 - Square-Schritt: squares turned 45 degrees



Mittelwert aus 4 umliegenden
Punkten + Zufallswert

Bildquelle: By Christopher Ewin - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=42510593>

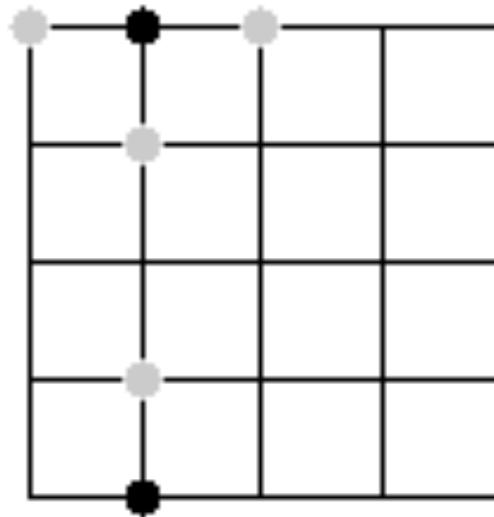
Beispiele: Diamond-Square



Bildquelle: Matěj Zábský

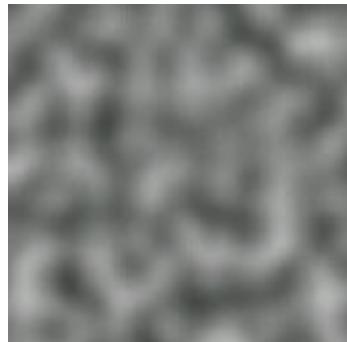
Wraparound?

- Möglichkeit zum Generieren aneinanderfügbarer Höhenfelder
- Überlauf durch Modulo auflösen

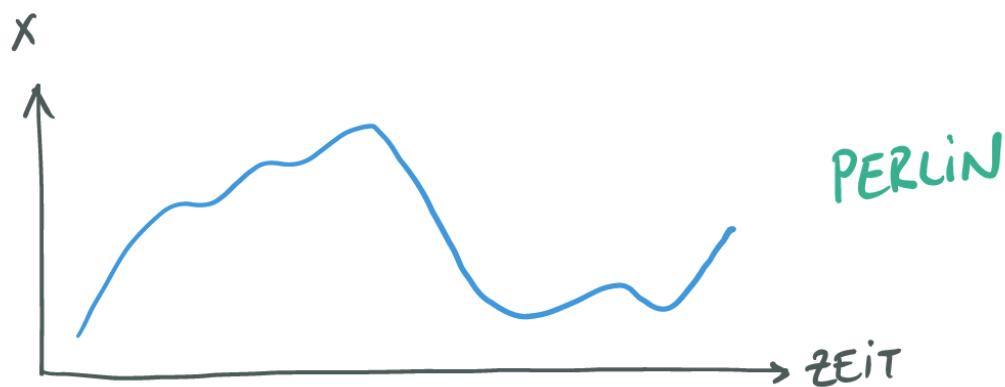
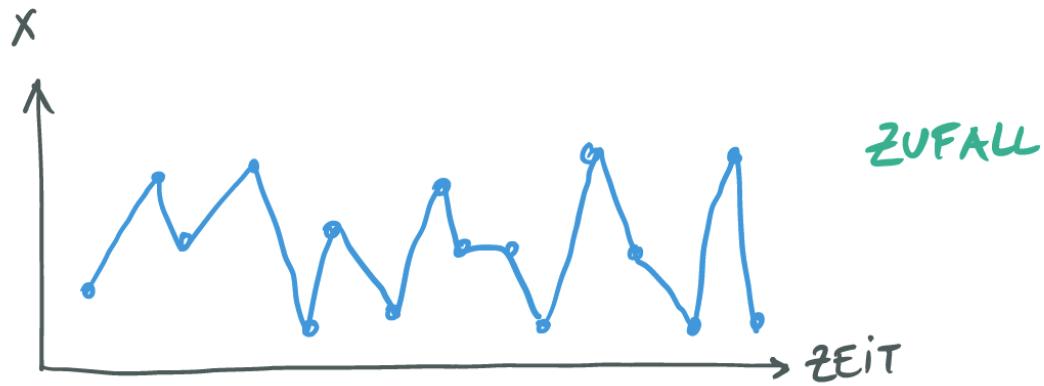


Perlin Noise

- Methode zum Generieren von Rauschen in n Dimensionen
 - skalierungs invariant
- weit verbreitet in Computergrafik, z.B. in Videospielen
- Ken Perlin entwickelte Idee für den Film Tron → Oscar gewonnen
- Hintergrund: real anmutende Texturen für natürliche Phänomene (z.B. Wolken, Marmor, ..., Höhenfelder!)



Motivation



Perlin Noise

- Idee: Modellierung der Gradienten anstelle des Rauschens direkt

Algorithmus

starte mit Gitter

FOREACH Gitterzelle

 Gradient festlegen (Zufallszahlen)

 (Höhenwert folgt aus Gradient)

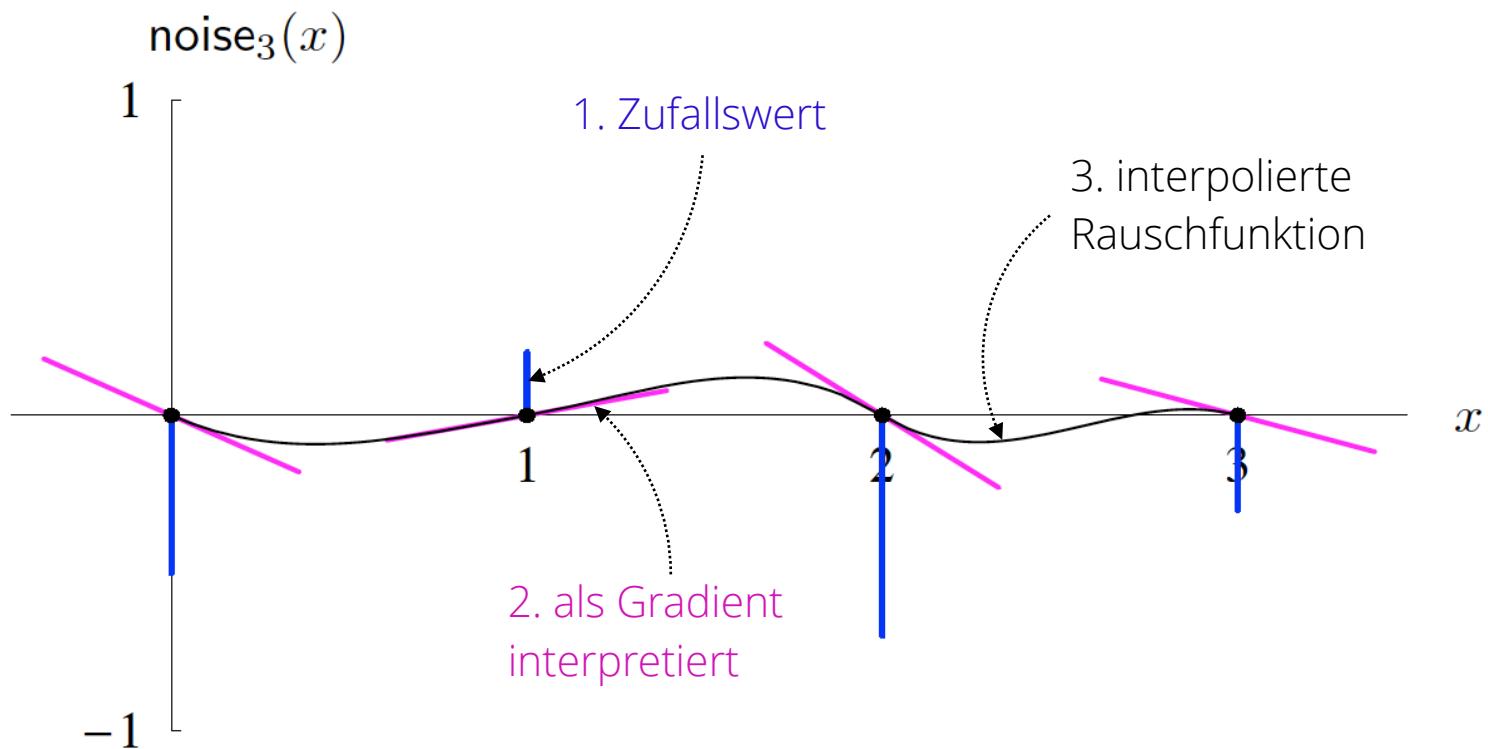
Werte zwischen den Zellen: Interpolation

wiederholte Anwendung in höherer Auflösung mit
geringerer Amplitude (vergleiche Diamond-Square)

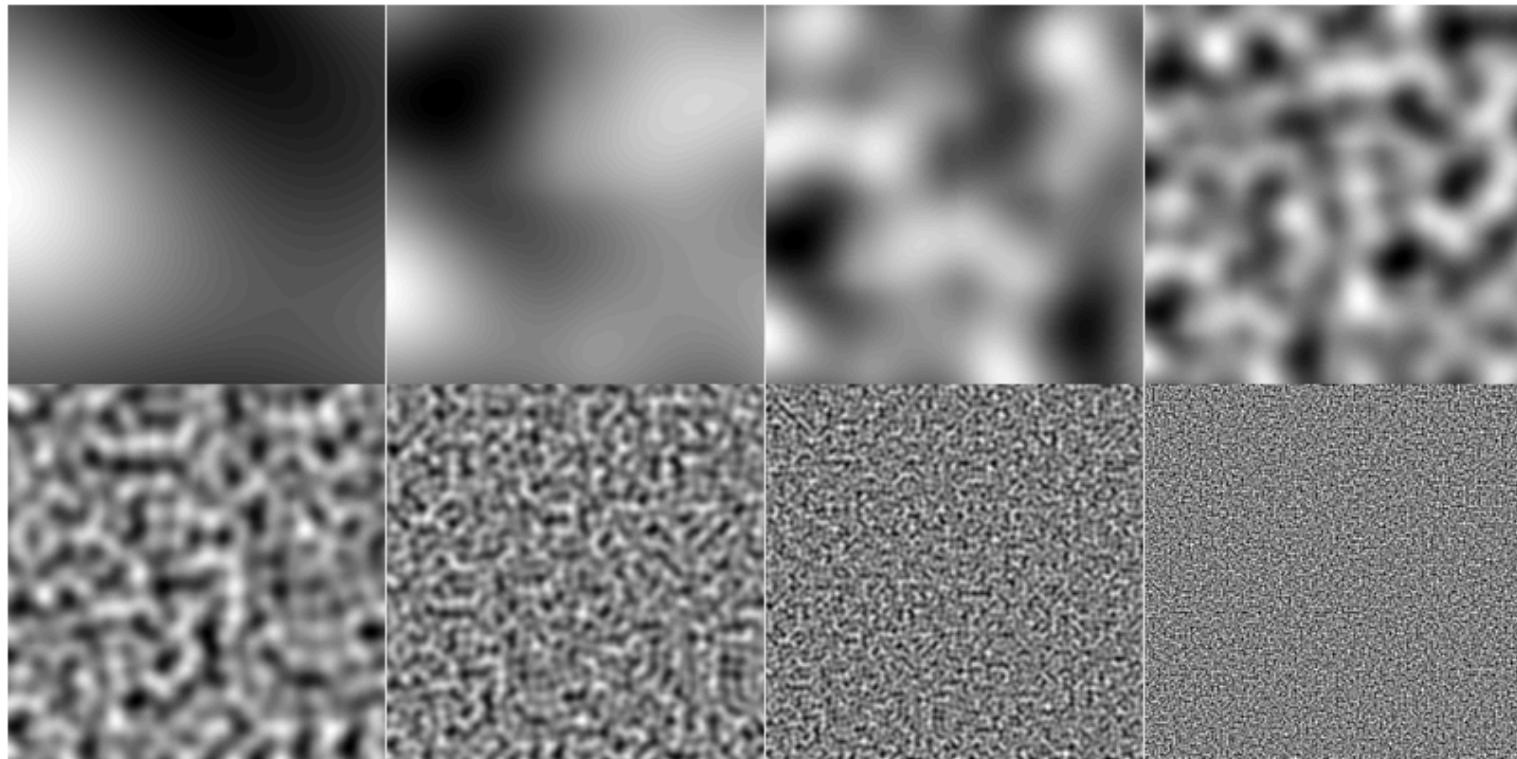
 (= nächste Oktave)

siehe: <https://www.youtube.com/watch?v=8ZEMLCnn8v0>

Beispiel: 1D Perlin Noise



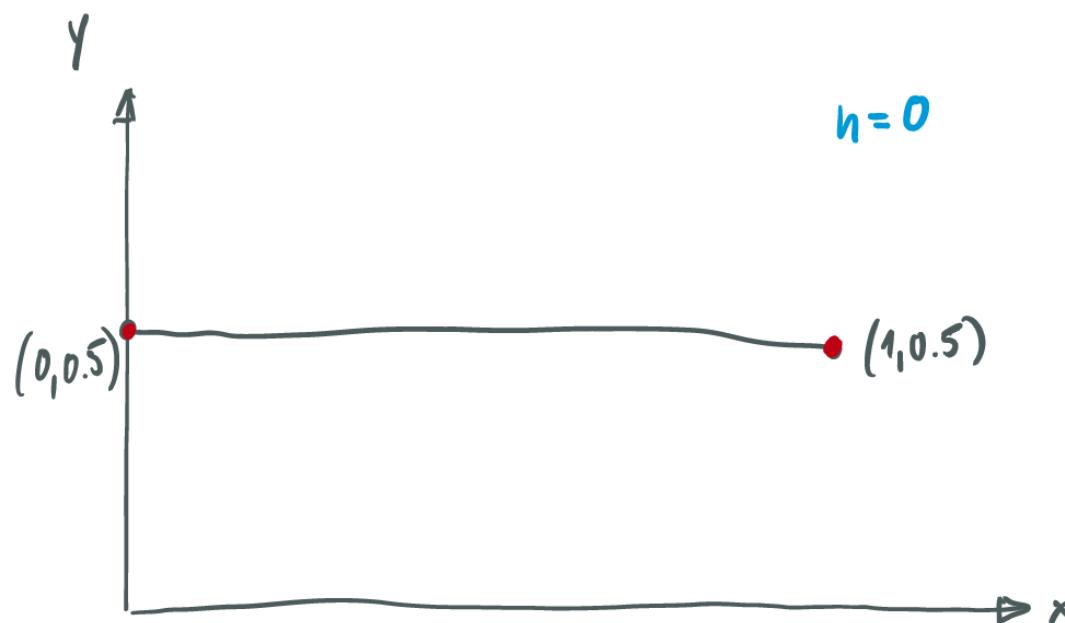
Beispiel: 2D Perlin Noise

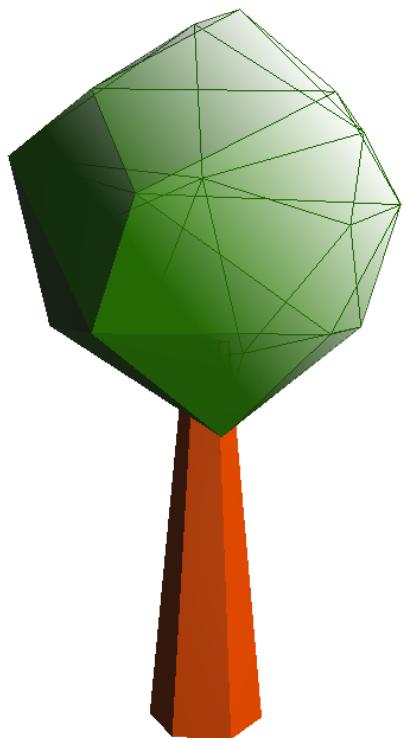


8 Oktaven (Auflösungsstufen) für Perlin Noise in 2D

Übung: Mittelpunkt-Verschiebung

- Führen Sie zwei Iterationen der Mittelpunktverschiebung mit folgenden Parametern durch
 - $H_i = 1 / (2^*i)$
 - Der Zufallszahlengenerator liefert Zahlen aus 0 ... 9
 - 7, 2, 5, 9, 2, 0, ...





Dungeons: BSP-Bäume

Dungeon

- Dungeon = geschlossenes System
 - Räume
 - durch Korridore verbunden

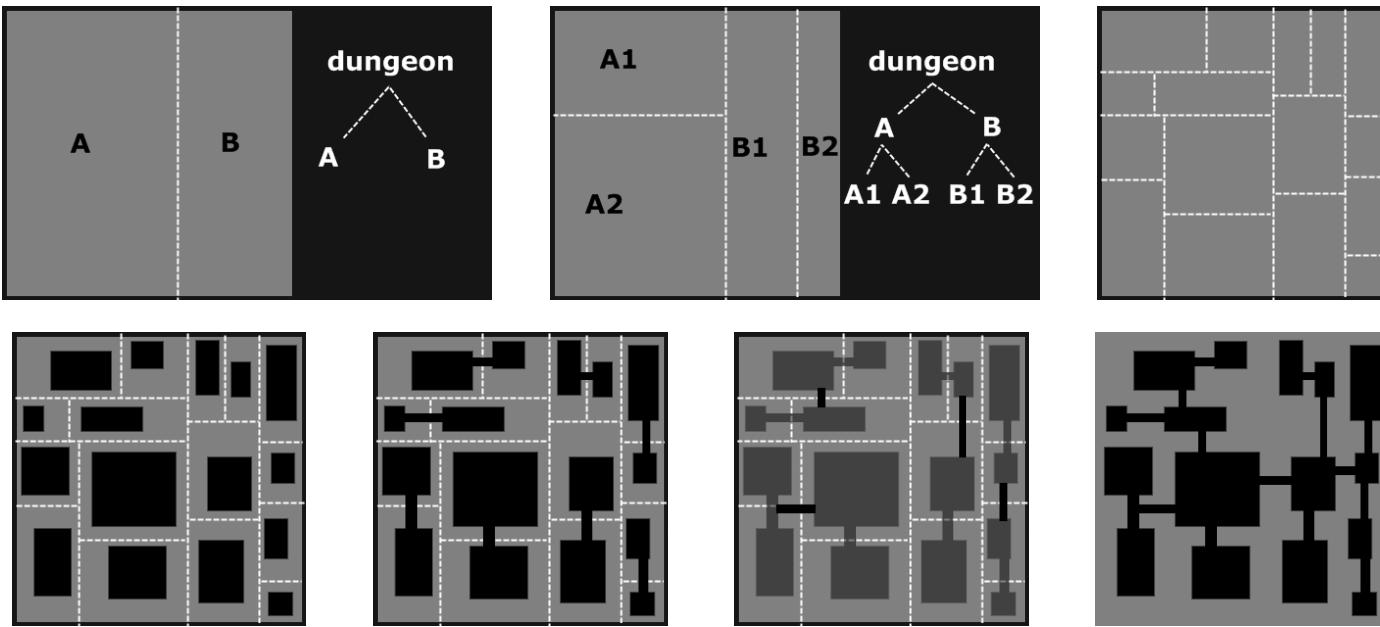


Dungeon in RuneScape (Jagex Games Studio)

Algorithmus: BSP-Dungeon

- Raumunterteilung
 - wähle zufällig Richtung (horizontal, vertikal)
 - wähle zufällige Position (x für vertikal, y sonst)
 - teile Dungeon entlang H in zwei Unterdungeons
 - rekursives abtauchen
- Räume
 - erzeuge Zufallsraum in jedem Blattknoten-Unterdungeon
 - (muss nicht rechteckig sein)
- Korridore
 - erzeuge Korridore, um Räume zu verbinden
 - jeweils zwischen Geschwistern im BSP-Baum

Binary space partitioning



Dungeon-Generierung mittels BSP-Baum

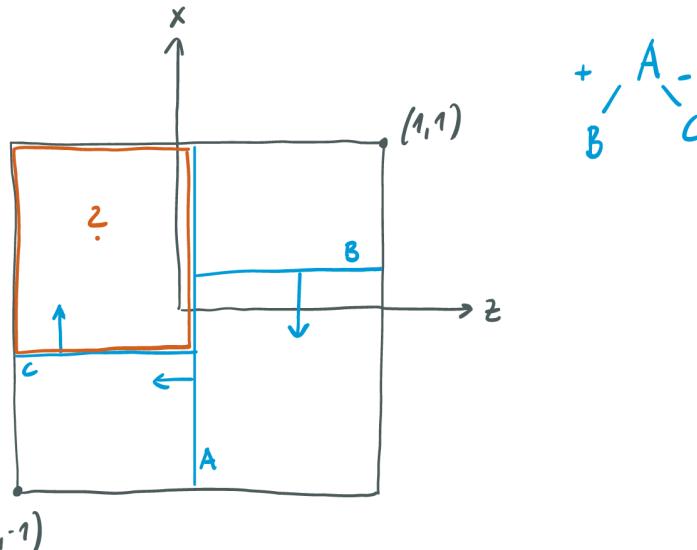
Bildquelle: [http://www.roguebasin.com/index.php?
title=Basic_BSP_Dungeon_generation](http://www.roguebasin.com/index.php?title=Basic_BSP_Dungeon_generation)

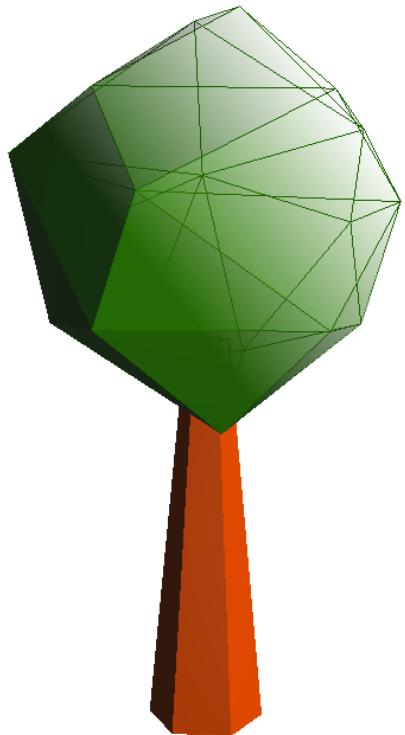
Weiteres

- BSP-Dungeon zum Ausprobieren: <https://eskerda.com/demos/dungeon/>
- Weitere Ansätze [12], u.a.
 - Drunkard's Walk
 - Graph-basierte Ansätze
- Weitere Quellen
 - <http://donjon.bin.sh/code/dungeon/>
 - <http://pcg.wikidot.com/pcg-algorithm:dungeon-generation>
 - https://www.reddit.com/r/gamedev/comments/1dlwc4/procedural_dungeon_generation_algorithm_explained/
 - http://www.roguebasin.roguelikedevelopment.org/index.php?title=Dungeon-Building_Algorithm
 - <http://donjon.bin.sh/code/dungeon/>

Übung: BSP-Dungeons

- Gegeben ist eine Karte mit den Eckpunkten $(-1, -1)$ und $(1, 1)$ und ein BSP-Baum auf der Karte.
- jeder Knoten des BSP-Baum beinhaltet einen Ebene in Normalform $E: nx - np = 0$ (Ebenennormale n und Ebenenpunkt p)
- wir betrachten einen Blattknoten im BSP-Baum und insbesondere dessen positiven Unterraum
- Aufgabe: Geben Sie einen Algorithmus in Pseudocode für die maximale Größe eines Raumes in dem Unterraum an.



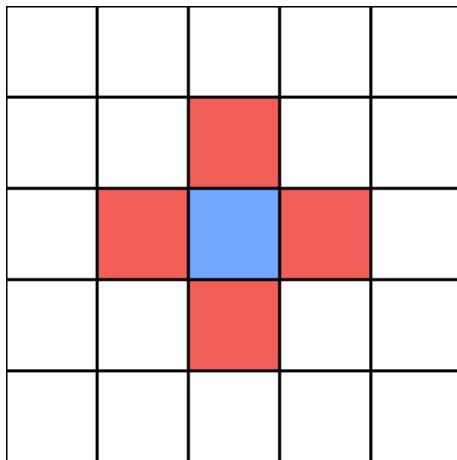


Dungeons: Zelluläre Automaten

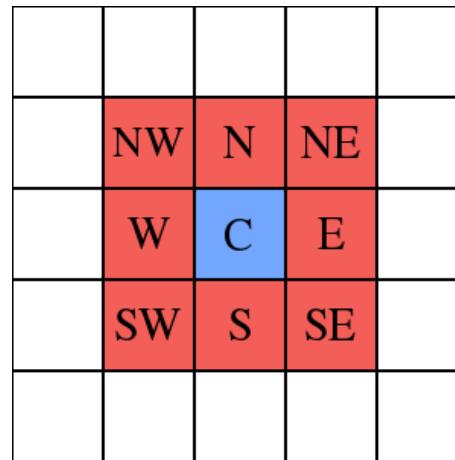
Zelluläre Automaten

- Modellierung räumlich diskreter (Gitter) dynamischer Systeme (= über die Zeit veränderlich)
- Entwicklung einzelner Zellen hängt ab von
 - Nachbarschaft
 - eigenem aktuellen Zustand

Nachbarschaft



von Neumann-Nachbarschaft



1-Moore-Nachbarschaft

Bildquelle: Von MorningLemon - Eigenes Werk, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=38746431>

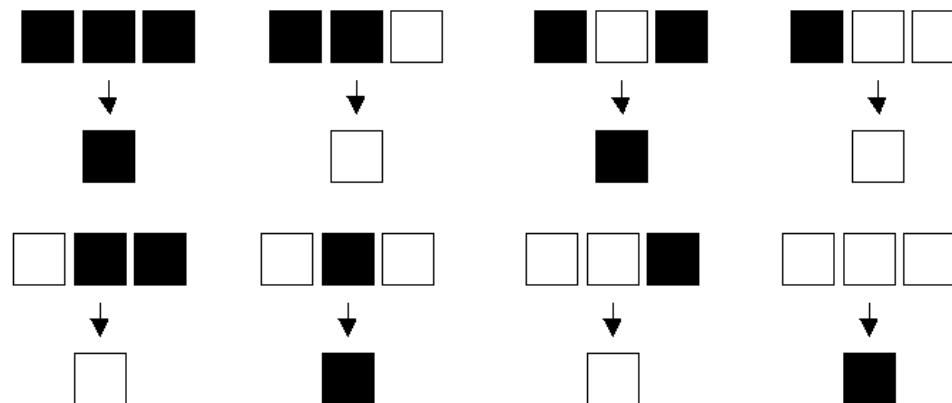
Von MorningLemon - Eigenes Werk, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=38746075>

Zelluläre Automaten

- Bausteine
 - ein Raum R (Zellularraum)
 - eine endliche Nachbarschaft N
 - eine Zustandsmenge Q
 - eine lokale Überführungsfunktion $\delta: Q^n \rightarrow Q$

Beispiel

- Berechnungsschema basierend auf lokaler Interaktion
- Verwendung: künstliches Leben, Komplexitätsuntersuchungen
- Beispiel: Regelsystem in 1D



Beispiel: Conveys Spiel des Lebens

- jede Zelle: entweder tot oder lebendig
- Regeln
 - tote Zelle, genau drei lebenden Nachbarn → neu geboren
 - lebende Zelle, weniger als zwei lebenden Nachbarn → stirbt (Einsamkeit)
 - lebende Zelle, zwei oder drei lebenden Nachbarn → bleibt am Leben
 - lebende Zellen, mehr als drei lebenden Nachbarn → stirbt (Überbevölkerung)

Strukturen

- statische Objekte
- Raumschiffe, z.B.
- pulsierender Objekte, z.B.



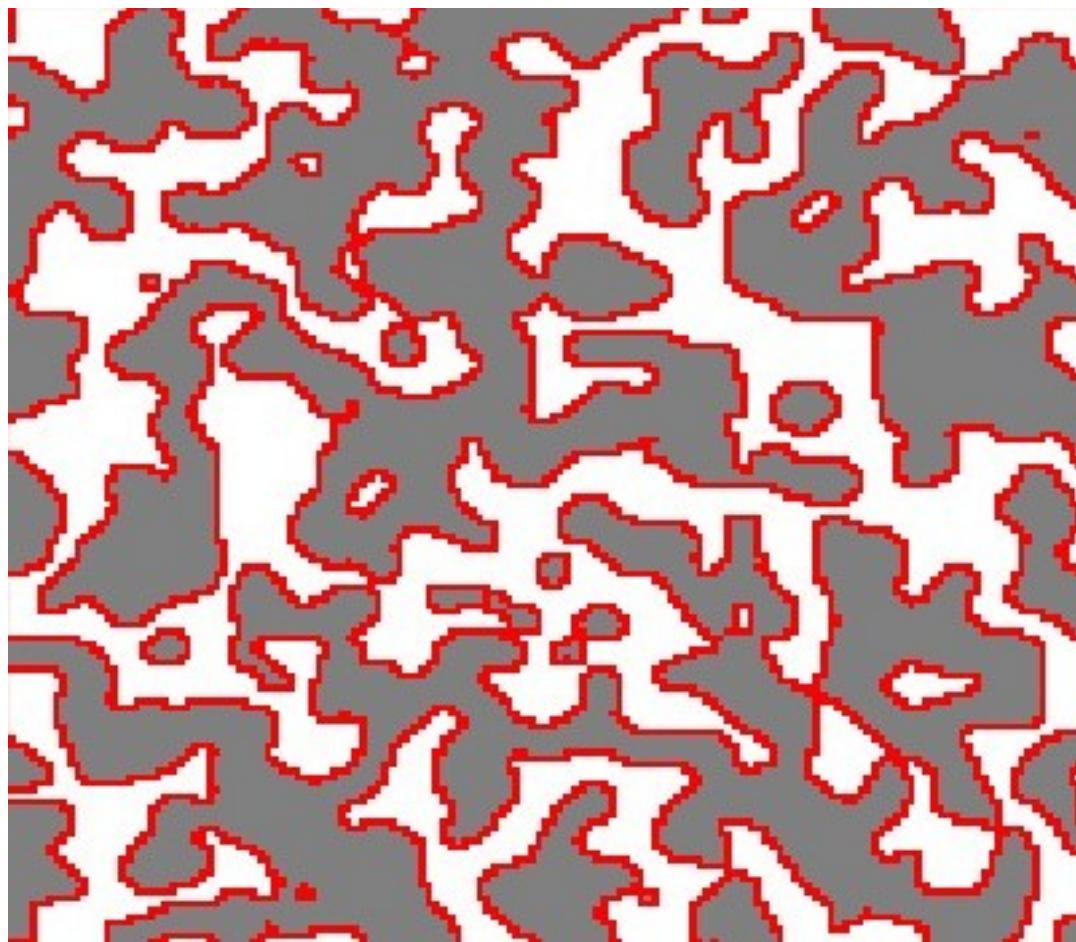
in der Prozeduralen Generierung

- Anwendungsfelder (Auswahl)
 - Ausbreitung von Feuer [7]
 - Olsen: Terrain generation mit Erosion [8]
 - Rogue-likes (verschiedene)

Höhlengenerierung

- Datenstruktur
 - Gitter z.B.. 50 x 50 Zellen
 - jede Zelle: leer oder Stein
 - Initialisierung:
 - alle Zellen: leer
 - zufällig r-Anteil Zellen zu Stein
 - n Iterationen mit Zellulärem Automaten
 - Stein: falls mindestens T Nachbarn in M Stein sind
 - leer: sonst
 - Fülle die Innenräume von Stein-Bereichen
- Parameter**
 - r: initialer Anteil an Stein-Zellen (z.B. 0.5)
 - n: ZA-Iterationen (z.B. 4)
 - T: Nachbarschaftsschwellwert (z.B. 5)
 - M: Moore - Nachbarschaftsgröße (z.B. 1)

Beispiel

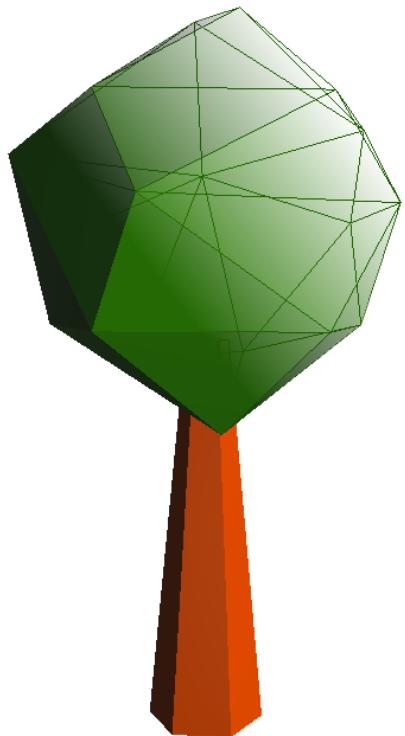


*mit einem Zellulären Automaten generiertes Höhlensystem
[9] ($M=2$, $T=13$, $n=4$, $r=0.5$)*

Übung: Zellulärer Automat

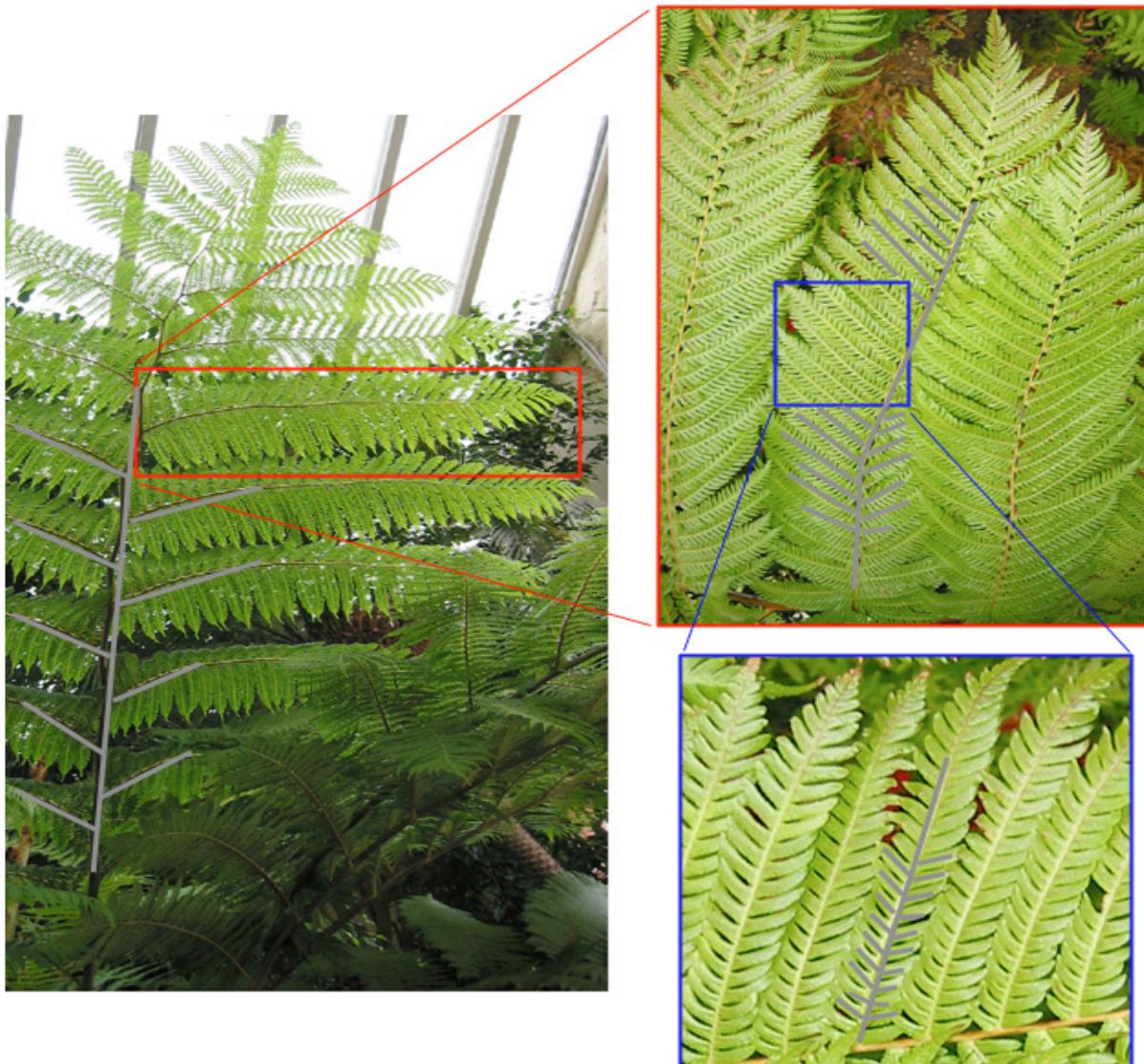
- Gegeben ist ein Gitter der Größe 3x3. In jeder Zelle kann eine Zahl zwischen aus [0 ... 3] stehen. Führen Sie eine Iteration eines Zellulären Automaten mit den folgenden Regeln durch (von Neumann Nachbarschaft, n = Nachbar):
 - falls $\sum n < 3$ dann 3
 - falls $\sum n < 6$ dann 2
 - sonst 1

0	1	2
3	0	1
2	3	0



Objekte: Lindenmayer- Systeme

Selbstähnlichkeit bei Pflanzen



Selbstähnlichkeit bei Pflanzen



Lindenmayer (L)-Systeme

- vorgeschlagen von Aristid Lindenmeyer, 1968 [10]
- Ziel: Modellierung von Pflanzen
- Idee:
 - Wörter aus einem **Alphabet** generieren
 - Grundlage: **Grammatik** und **Axiom(e)**
- enge Verbindung zu Chomsky Grammatiken (Linguistik)
- Unterschied: Produktionen werden parallel ausgeführt, nicht sequenziell

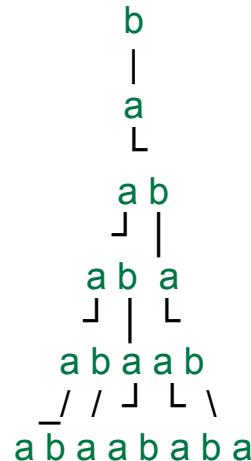
Beispiel

- Alphabet: {a, b}
- Grammatik (Produktionsregeln):

$a \rightarrow ab$

$b \rightarrow a$

- Axiom: b



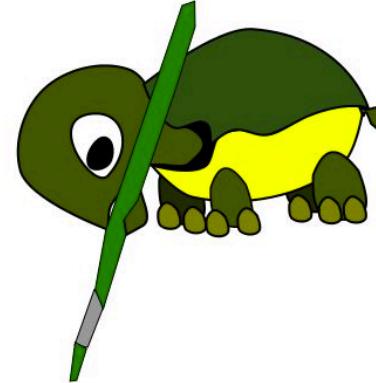
*Beispiel für die Ableitung eines
Lindenmayer-Systems*

Typen von L-Systemen

- kontextfrei: Produktionsregeln basieren nur auf einzelнем Symbol
- kontextsensitiv: Produktionsregeln können von benachbarten Symbolen abhängig sein
- deterministic: es gibt genau eine Produktionsregel für jedes Symbol
- nicht-deterministisch: mehrere verschiedene Produktionsregeln für ein Symbol
 - z.B. randomisierte und/oder wahrscheinlichkeitsbasierte Auswahl

Graphische Interpretation von L-Systemen

- Vorgeschlagen von Prusinkiewicz, 1986 [11]
- Idee:
 - Wörter als Anweisungen für eine Turtle-Grafik interpretieren
 - Wort von links nach rechts lesen
 - Status der Turtle anpassen
 - Position
 - Ausrichtung



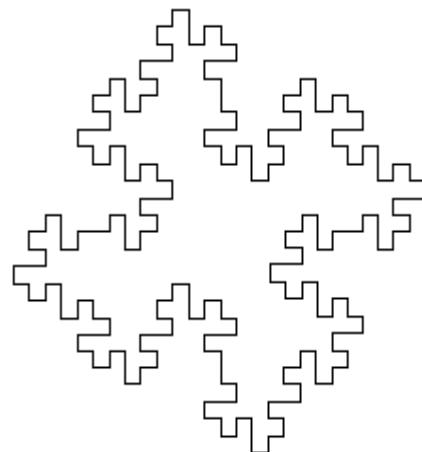
THINK OF A TURTLE HOLDING A PEN

Grammatik

- Alphabet: $\{F, f, +, -\}$
 - F: Länge d Schritt vorwärts, dabei Linie zeichnen
 - f: Länge d Schritt vorwärts, keine Line zeichnen
 - +: um den Winkel δ gegen Uhrzeigersinn drehen
 - -: um den Winkel δ im Uhrzeigersinn drehen
- für Abzweigungen zusätzlich
 - [: aktuellen Zustand (Position + Orientierung) auf einem Stack sichern
 -]: Zustand vom Stack pop'en

Beispiel (ohne Stack)

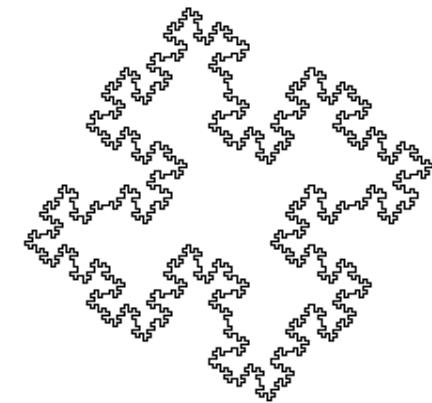
- Axiom: $F+F+F+F$
- Grammatik:
 - $F \rightarrow F+F-F-FF+F+F-F-F$
- $\delta: 90^\circ$



$n = 0$

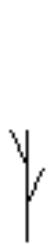
$n = 1$

$n = 2$



Beispiel (mit Stack)

- Axiom: F
- Grammatik:
 - $F \rightarrow F[-F]F[+F][F]$
- $\delta: 30^\circ$



$n = 1$



$n = 2$



$n = 3$



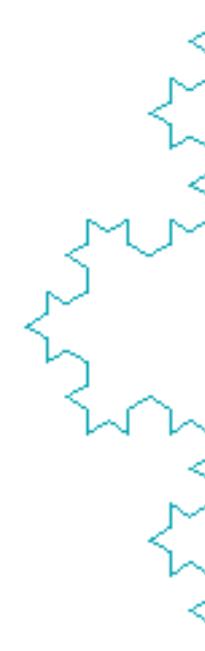
$n = 4$

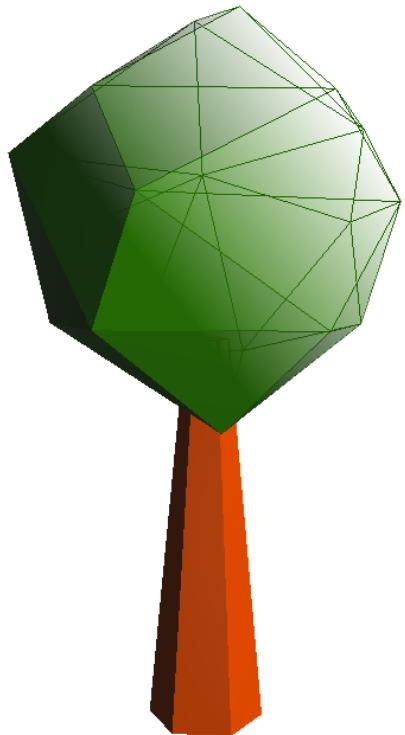


$n = 5$

Übung: L-System

- Gegeben ist das folgenden Alphabet:
 - Symbole: {A, F, +, -}
 - Grammatik:
 $A \rightarrow F$
 $F \rightarrow F+F--F+F$
 - Axiom: A
- Führen Sie zwei Ableitungsschritte durch.

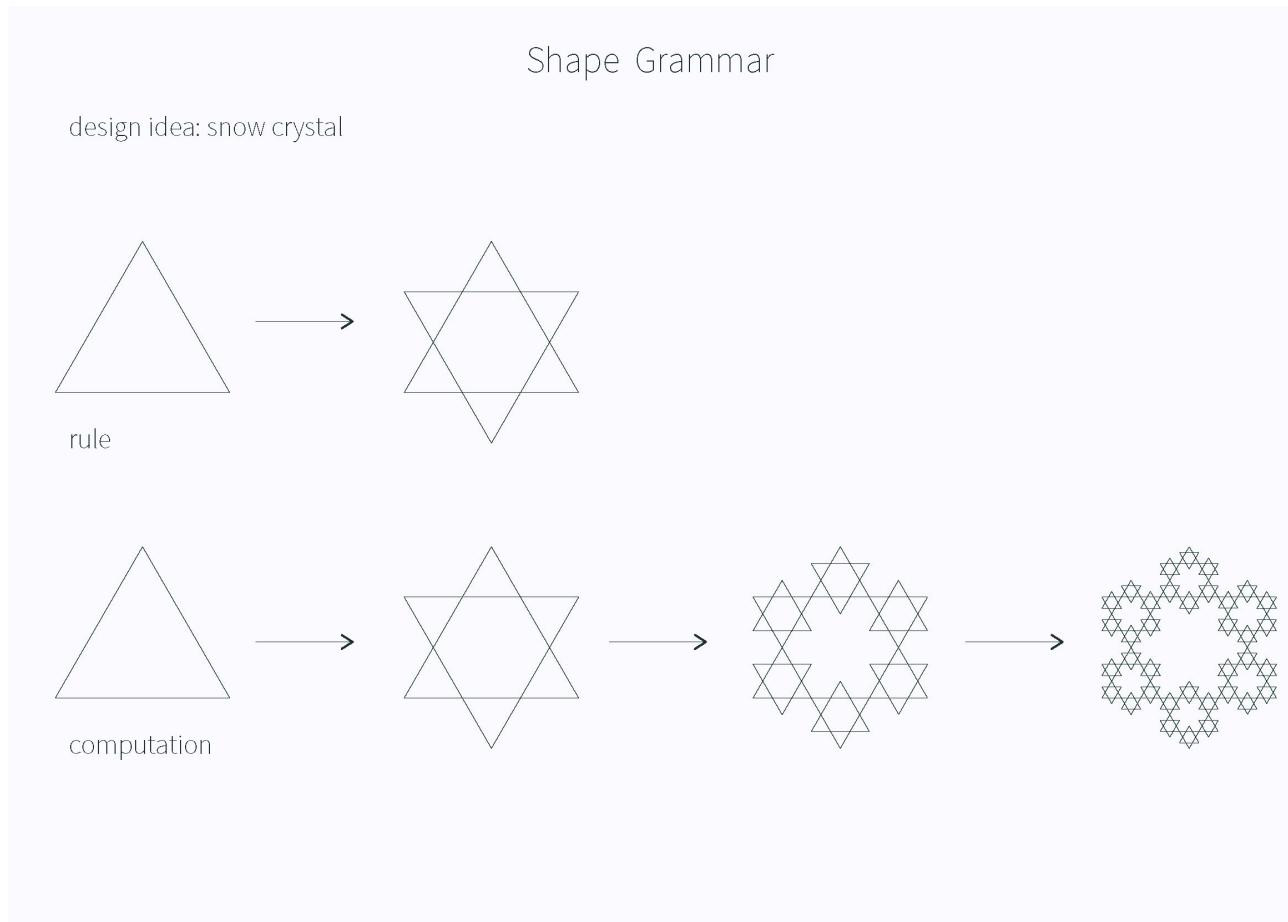




Objekte: Shape Grammar

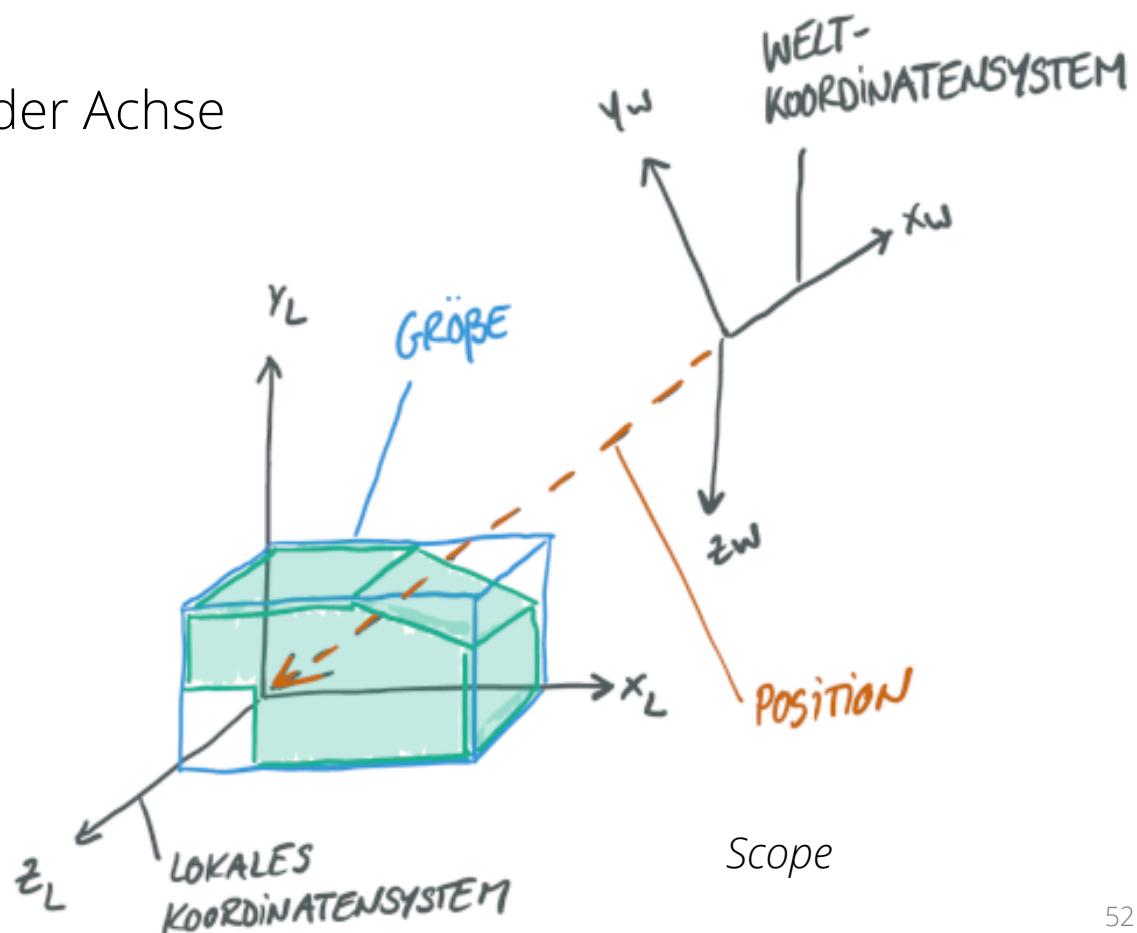
Idee

- Grammatik über Formen (Shapes), siehe [3,4,5]



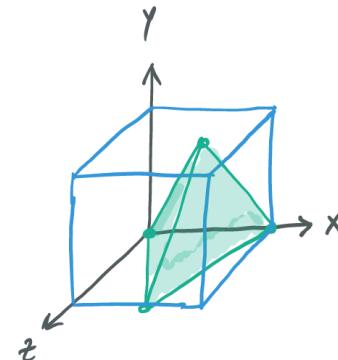
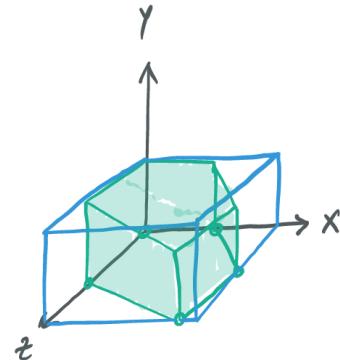
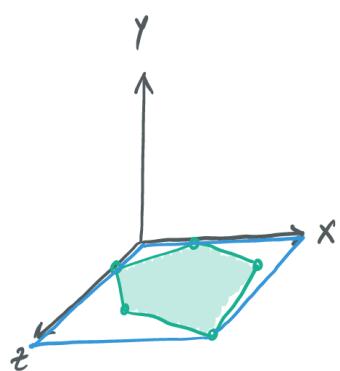
Scope

- Koordinatensystem
 - Achsen: x, y, z
 - Ursprung
 - Größe entlang jeder Achse
- Modifikationen
 - Translation
 - Rotation
 - Skalierung



Form

- Repräsentieren eine geometrische Form
- jeweils eingebettet in einen Scope

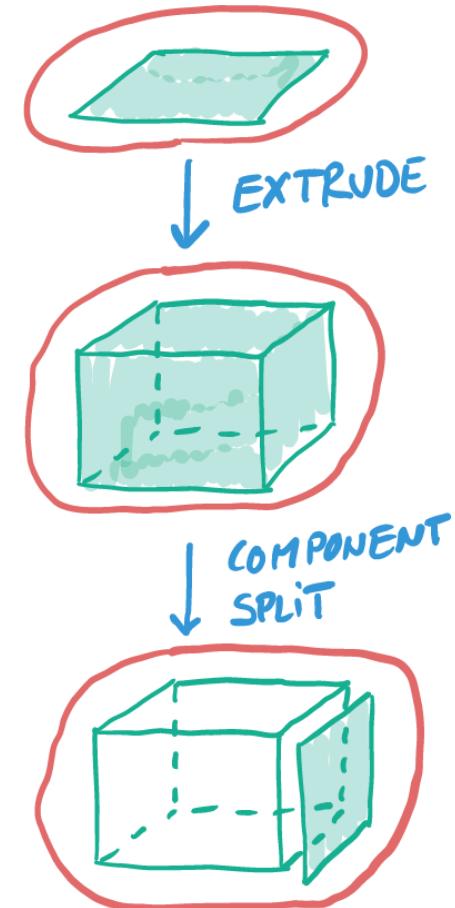


Verschiedene Formen: Polygon, Prisma, Spitzdach

- Dimension:
 - 3D (z.B. Prisma, Dach)
 - 2D (z.B. Polygon)
 - 1D (Kante)
 - 0D (Punkt)

Hierarchische Repräsentation

- jede Form kann Kindformen haben
- Kindformen sind aus Elternform entstanden
 - Anwenden einer Regel → Grammatik



Grammatik

- Grammatik $G = (N, T, R, I)$
- Nicht-Terminalsymbole $N \subseteq U$
- Terminalsymbole $T \subseteq U$
- Startobjekte/Axiome $I \subseteq N$
- Produktionsregeln $R \subseteq U^* \times U^*$
 - $a \rightarrow b$

Produktionsregeln

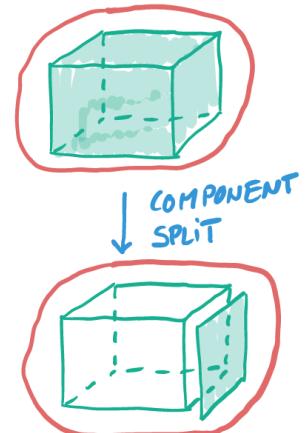
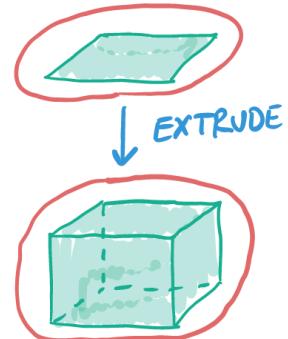
- Regel in der Shape-Grammar

$$\textit{pred} : \textit{cond} \rightarrow \textit{succ} : \textit{prob}$$

- **pred**: Symbol das abgeleitet wird (Vorgänger)
- cond: optionale Bedingung für die Ableitung
- **succ**: Symbole die Ergebnis der Ableitung sind
- prob: optionale Wahrscheinlichkeit für das Anwenden der Regel
- Beispiel
 - *Lot* → *extrude(0.7) Building*
 - Vorgänger: Symbol *Lot*
 - Nachfolger: Symbole *extrude(0.7)* und *Building*

Symbole

- zwei Ausprägungen
 - **Operation:** verändern der aktuellen Form
 - Beispiel: extrude - macht aus dem aktuellen Polygon ein Prisma
 - **Generator:** Erzeugen einer oder mehrere neuen Formen
 - Beispiel: component_split - extrahiert aus dem aktuellen Polygon eine oder mehrere Facetten



Symbole in einer Regel

<Bezeichner>(<Parameterliste>){ID-Liste}

- Beispiel

GroundFloor --> split("Z",0.3,1r){DoorSegment,Floor}

Symbole

- Operationen
 - Extrude
 - Affine Transformationen: Rotation, Translation, Skalierung
- Generatoren
 - Id
 - Split
 - Repeated-Split
 - Component-Split
 - Roof

Extrude

- Syntax

extrude(<offset>)

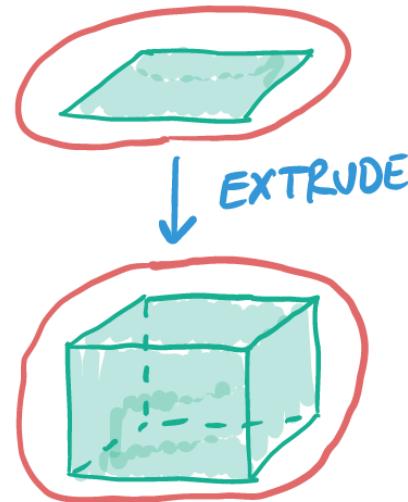
- Beispiel:

extrude(0.7)

- Extrudieren eines Polygons entlang der y-Achse

- Länge: <offset>

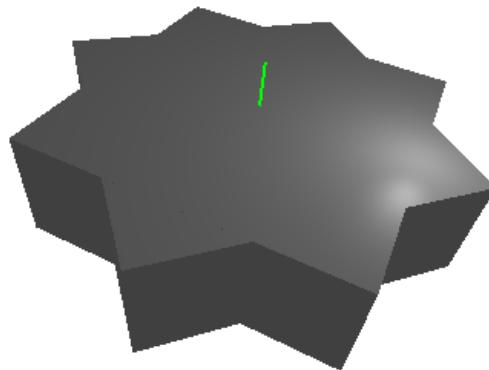
- Polygon → Prisma



Affine Transformation

- Syntax
 - $Rx(<\text{winkel}>)$, $Ry(<\text{winkel}>)$, $Rz(<\text{winkel}>)$, $S(sx,sy,sz)$, $T(tx,ty,tz)$
- Ausprägungen
 - Rotation um x-, y- oder z-Achse
 - Skalierung
 - Translation

Lot --> extrude(0.7) Block S(1,1,1) Ry(45) Block



ID

- Binden einer Id an das die aktuelle Form
- Beispiel
 - *Lot* → *extrude(0.7) Building*
- Vorgehen
 - *Lot* ist Symbol eines Polygons
 - Extrude generiert aus der Form Polygon eine Form Prisma
 - Das Prisma bekommt die ID *Building*

Split

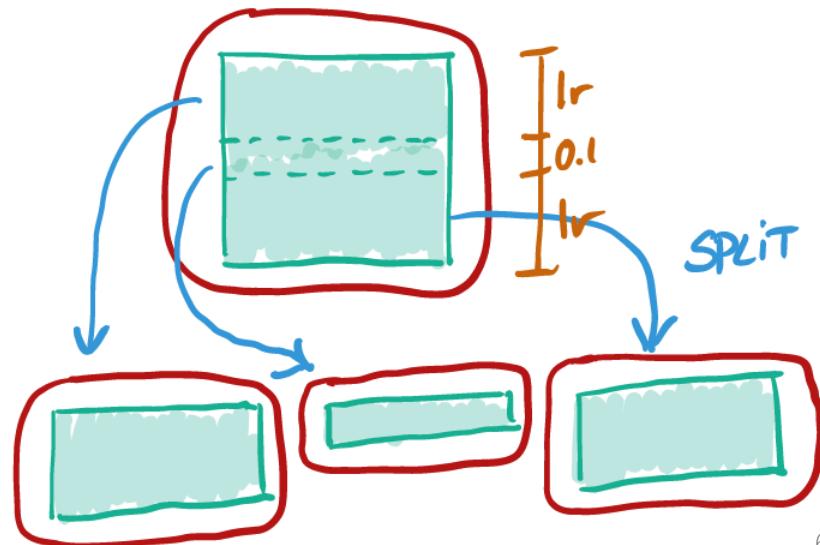
- Syntax

split(<Achse>, <Split-Breiten>){<ID-Liste>}

- Beispiel

Facade --> split("X", 1r, 0.1, 1r){GroundFloor, Ledge, Floor}

- Split-Breiten können absolut (z.B. 0.1) oder relativ (z.B. 1r) sein
 - r wird dynamisch so berechnet, dass gesamte Breite der Komponente unterteilt wird
- Polygon → Polygon*
- Prisma → Prisma*



Repeated-Split

- Syntax

repeated_split(<Achse>, <Split-Breite>){<ID-Liste>}

- Beispiel

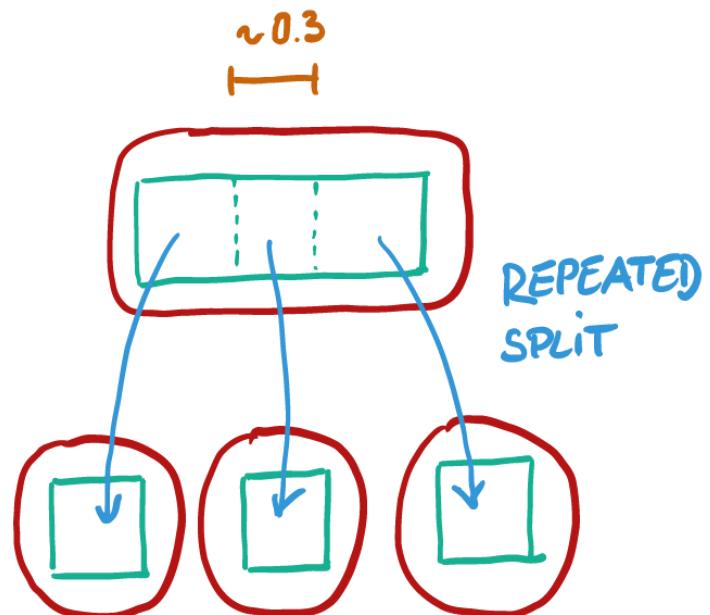
Floor --> repeated_split("Z", 0.3){FloorSegment}

- Split-Breite ist ein Absolutwert, der angenähert wird

- Breite der Komponente muss am Ende ganzzahliges Vielfaches von tatsächlicher Breite sein

- Polygon → Polygon*

- Prisma → Prisma*



Component-Split

- Syntax

component_split(<Typ>){<ID>}

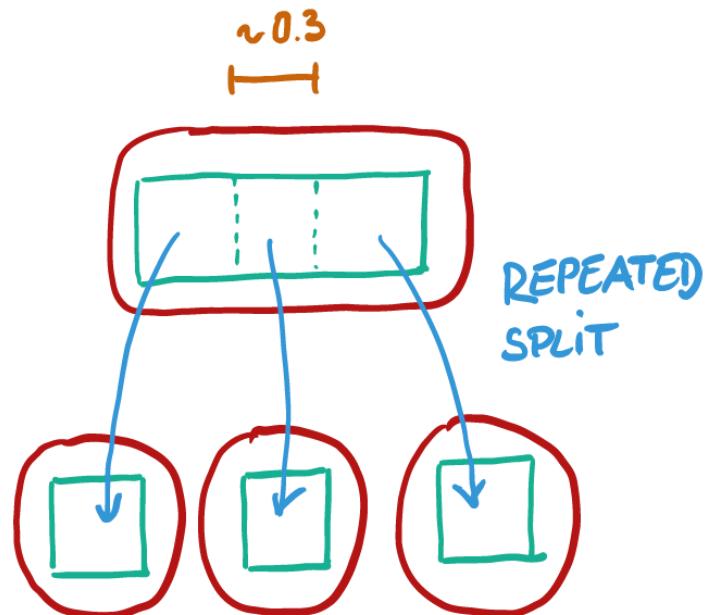
- Beispiel

component_split("side_faces"){Facade}

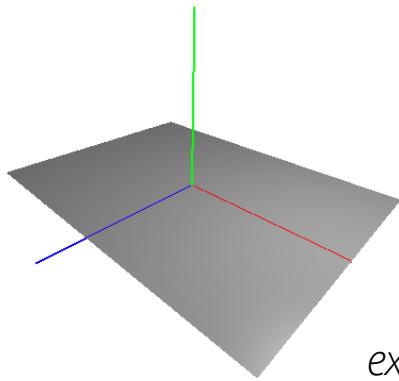
- Typ: Seitenflächen, Oben, Unten, Alle

- ID: ID für Komponente

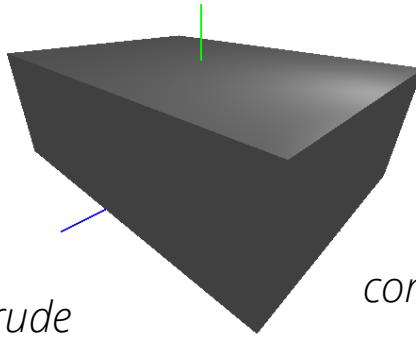
- Prisma → Polygon*



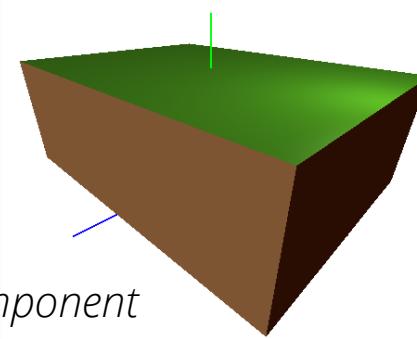
Beispiel



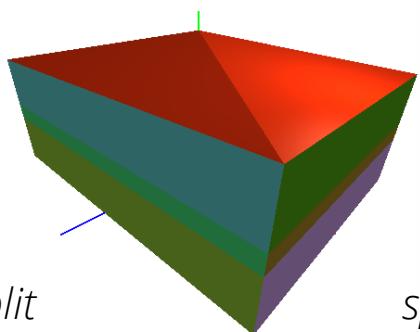
extrude



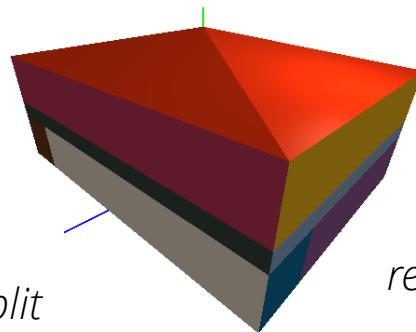
component split



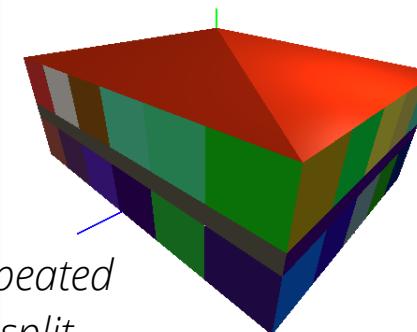
roof



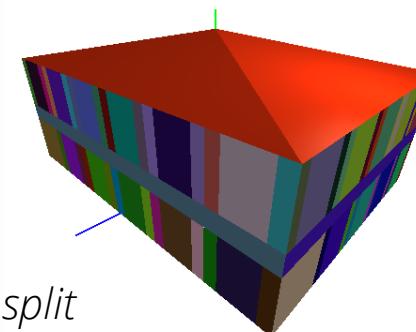
split



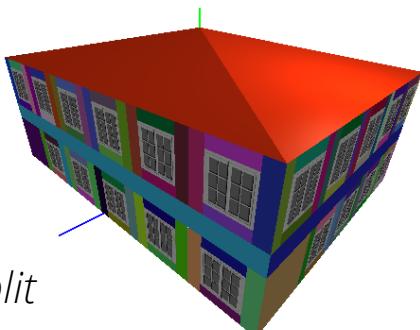
split



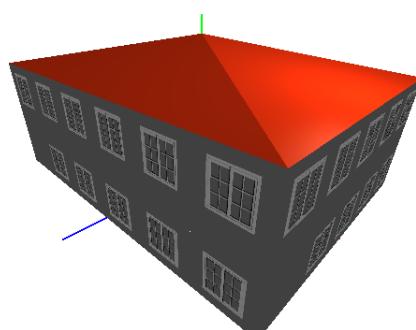
repeated split



split



split



Beispiel



Bildquelle: Müller et al. [5]

Übung: Shape Grammar

- Leiten Sie das Axiom mit der folgenden Shape-Grammar ab:

Axiom :



Regeln :



Zusammenfassung

- Einführung
- Terrain
- Dungeons
 - BSP-Bäume
 - Zelluläre Automaten
- Objekte
 - Lindenmayer System
 - Shape Grammar

Quellen

- [1] Noor Shaker, Julian Togelius, and Mark J. Nelson (2016). Procedural Content Generation in Games: A Textbook and an Overview of Current Research. Springer. ISBN 978-3-319-42714-0.
- [2] Hendrikx, M., Meijer, S., van der Velden, J., and Iosup, A. 2011. Procedural Game Content Generation: A Survey. ACM Trans. Multimedia Comput. Commun. Appl., Article 1 (February 2011),
- [3] George Stiny and James Gips. "Shape Grammars and the Generative Specification of Painting and Sculpture." *IFIP Congress* (1971)
- [4] Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. 2003. Instant architecture. In ACM SIGGRAPH 2003 Papers (SIGGRAPH '03)
- [5] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. 2006. Procedural modeling of buildings. In ACM SIGGRAPH 2006
- [6] A. Fournier,D. Fussell und L. Carpenter: *Computer rendering of stochastic models* In: Communications of the ACM, Band 25, Nr. 6, 1982, S. 371–384
- [7] Penelope Sweetser, An Emergent Approach to Game Design – Development and Play, PhD thesis, 2006
- [8] Olsen, Jacob. (2004). Realtime Procedural Terrain Generation.
- [9] Johnson, Lawrence David et al. "Cellular automata for real-time generation of infinite cave levels." (2010).
- [10] Aristid Lindenmayer: Mathematical models for cellular interaction in development. In: J. Theoret. Biology. 18. 1968, 280–315.
- [11] Prusinkiewicz, Przemyslaw & Lindenmayer, Aristid. (1996). The Algorithmic Beauty of Plants. 10.1007/978-1-4613-8476-2.
- [12] Linden, Roland & Lopes, Ricardo & Bidarra, Rafael. (2014). Procedural Generation of Dungeons. Computational Intelligence and AI in Games, IEEE Transactions on. 6. 78-89. 10.1109/TCIAIG.2013.2290371.