Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

FAKULTÄT
FÜR MATHEMATIK, INFORMATIK
UND NATURWISSENSCHAFTEN

Objektrepräsentation

Vortrag für das Seminar „Inside a Physics Engnine“ bei Dr. Norman Hendrich

Referent: Lovis König

29.04.2026

Bild:
Yue Li, PARADISE Lab, CALTECH

https://www.paradise.caltech.edu/~yli/software/pceditor.html

Motivation

- Im 2D-Raum:
  - Pixel, Vektor, Punkte
- Im 3D-Raum:
  - Voxel, Mesh, Point Cloud
- Unterschiedliche Repräsentationen haben unterschiedliche geometrische Struktur

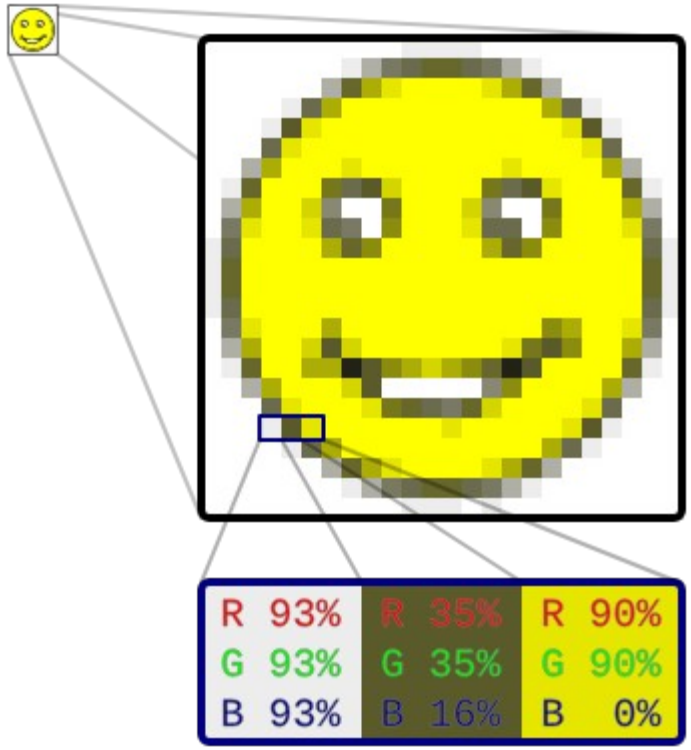


Bild: Wikipedia

https://commons.wikimedia.org/wiki/File:Rgb-raster-image.svg

Wie repräsentieren wir 3D-Objekte?

- Explizit / Implizit: Wie wird die Geometrie gespeichert?
  - Explizit: Point Cloud, Meshes
  - Implizit: SDF
- Boundary / Volume: Speichern wir Oberfläche oder Volumen?
  - Boundary: Meshes
  - Volume: Voxels
- Strukturiert / Unstrukturiert: Bilden wir die geometrische Struktur ab?
  - Strukturiert: Voxel Grids, Meshes
  - Unstrukturiert: Point Clouds

Wie repräsentieren wir 3D Objekte?

- Unterschiedliche Ansätze
- Unterschiedliche Einsatzzwecke
- Was für Repräsentationen gibt es?
- Wofür kann man sie verwenden?
- Was kann man mit ihnen machen?

Themen

1 Motivation ✓

2 Voxel, Tiefenbilder, Point Clouds

3 Polygon Meshes

4 Live-Demo mit Python

Voxel

- Pixel = Picture Element; Voxel = Volume Element
- 3D-Raum in Zellen aufteilen
- Reguläres Grid
- Einfache Repräsentation
- Volumen, Kollision, Nachbarschaft
- Hoher Speicheraufwand, Eckig, Suboptimal bei hohen Detailgraden

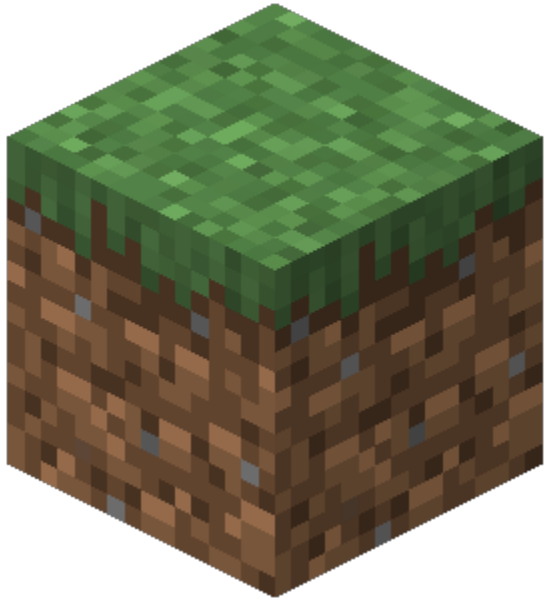


Bild: Mojang A/B

Tiefenbilder

Depth Images

- Speichert pro Pixel die Distanz zur sichtbaren Oberfläche
- „2.5D“: Nur ein Tiefenwert pro Koordinate
- Beschreibt sichtbare Oberfläche von bestimmtem Punkt, nicht die gesamte Objektgeometrie

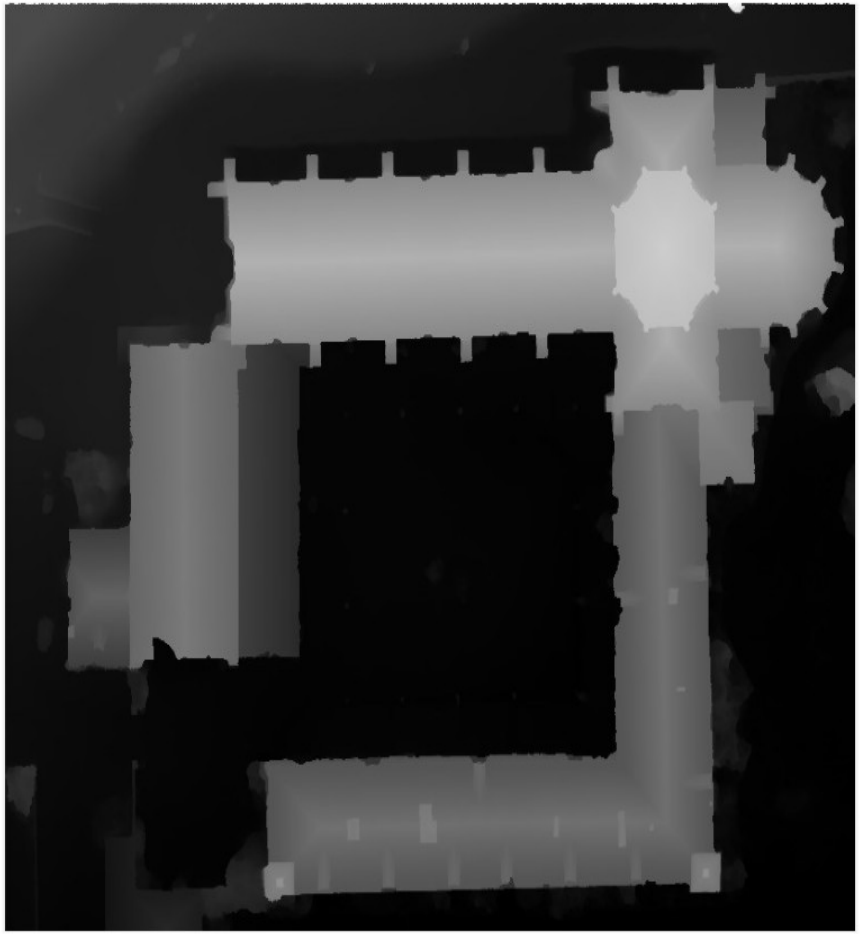


Bild: Poux, F. (2020). How to represent 3D Data?. Towards Data Science. p. 18.

Tiefenbilder

Depth Images

- Sensornahe Repräsentation
- Strukturiert
- Hohe Verfügbarkeit an Datensätzen
  - Deep Learning
- Liefern Entfernungsinformation direkt
- Mehrere Depth Images lassen sich zu anderen Strukturen kombinieren

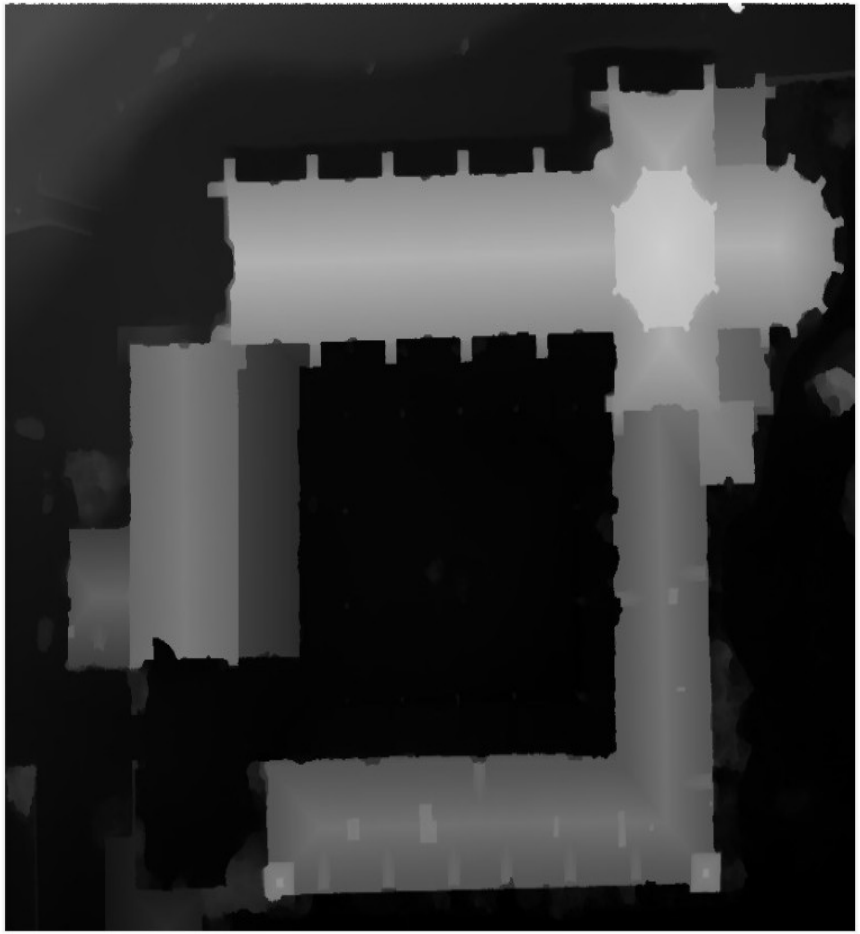
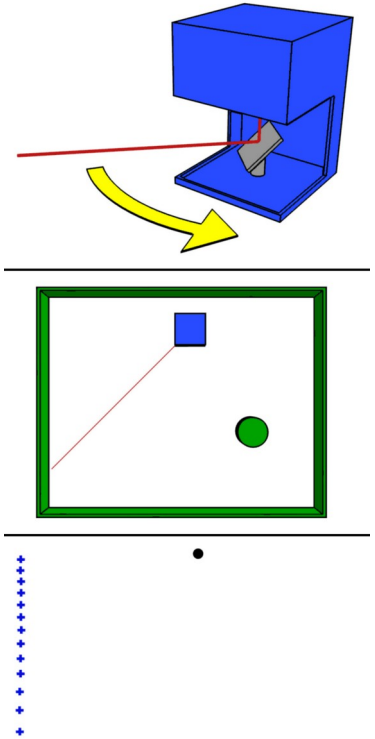


Bild: Poux, F. (2020). How to represent 3D Data?. Towards Data Science. p. 18.

Scannen von 3D-Objekten



Beispiel: einfaches LiDAR System

LiDAR = **Li**ght **D**etection **a**nd **R**anging oder **L**aser **I**maging, **D**etection **a**nd **R**anging

Bild: Wikipedia

https://commons.wikimedia.org/wiki/File:LIDAR-scanned-SICK-LMS-animation.gif

Point Clouds

- Ungeordnete Punktmenge in $P \subseteq \mathbb{R}^3$
- Explizite, unstrukturierte Repräsentation
- Sensordaten (LiDAR, RGB-D)
- Einfaches Konzept
- Schnelles Rendering
- Hohe Genauigkeit
- Transformationen einfach (Lineare Algebra)
- Hoher Speicheraufwand

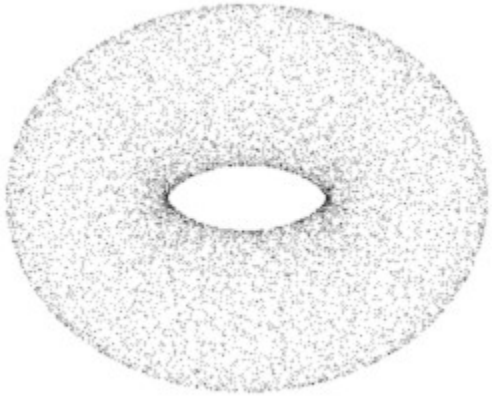


Bild: Wikipedia

https://commons.wikimedia.org/wiki/File:Point_cloud_torus.gif

Point Clouds: Anwendungen

- Nachbarschaftssuche
  - *k*-D-Bäume
- Normal Estimation
- Voxelization
- Surface Reconstruction

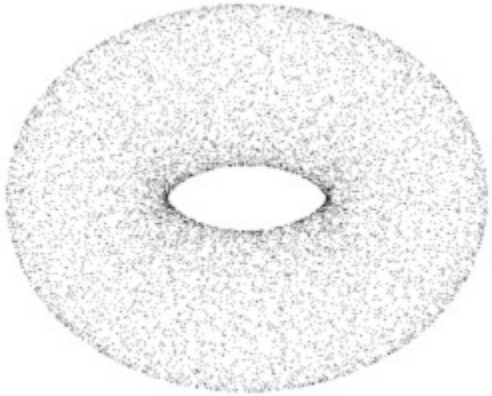


Bild: Wikipedia

https://commons.wikimedia.org/wiki/File:Point_cloud_torus.gif

Octrees

- Octree: Baum mit $\leq$ 8 Kindern pro Knoten
- Sparse Voxel Octree
  - Speichern belegte Voxelzellen
  - Adaptive Auflösung
- Point Cloud Octree
  - Speichert Punkte nach Region
  - Schnelleres Indexing

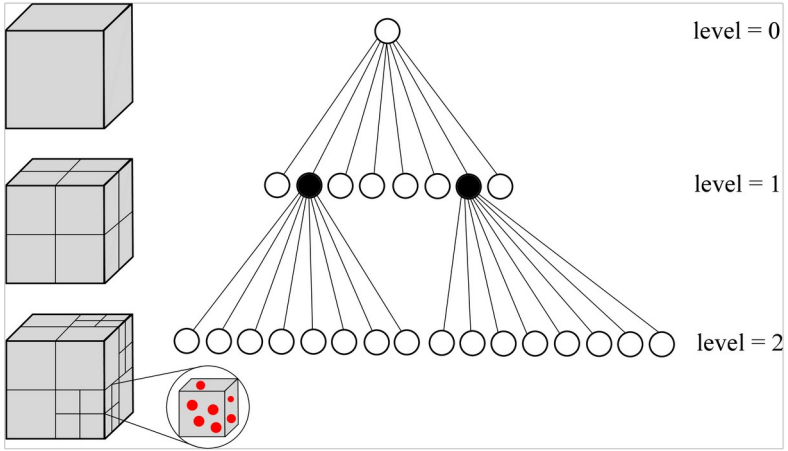


Bild: Mahesh R.G Prasad

https://mrgprasad.github.io/kanapy/builds/html/method.html

Kurzer Vergleich

	Stärken	Schwächen	Bspw. Anwendung
Voxel	• Volumen • Belegung	• Speicher	• Medizin
Depth Image	• Strukturiert • Sensornah	• Abhängig vom Blickwinkel	• Robotik
Point Cloud	• Sensornah • Präzise	• Keine Topologie	• Scannen

Surface Reconstruction

Von Point Cloud zu Mesh

1) Eingabe: Point Cloud (z.B. LiDAR-Scan)

2) Preprocessing
   Outlier Removal, Downsampling

3) Normal Estimation

4) Nachbarschaftsstruktur

5) Oberflächen-Rekonstruktion
   bspw. Poisson Reconstruction
   => Triangle Mesh

6) Cleanup
   z.B. Smoothing, Simplification,...

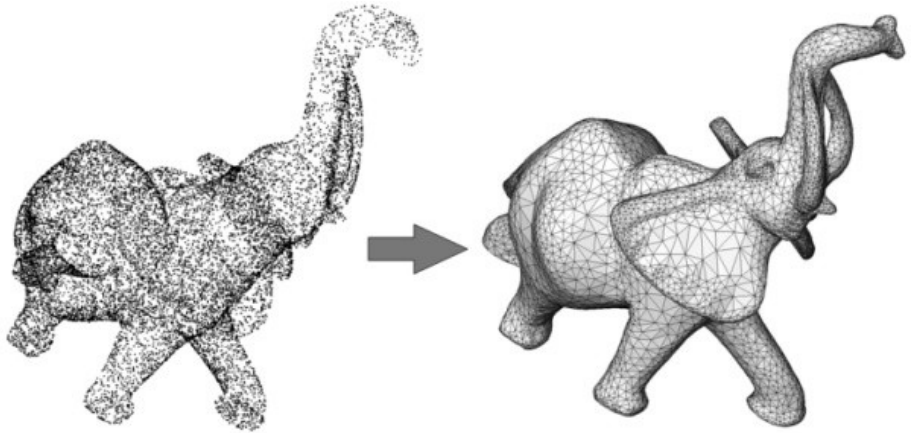


Bild: Pierre Alliez, Laurent Saboret, Gaël Guennebaud

https://doc.cgal.org/latest/Poisson_surface_reconstruction_3/index.html#fig__Poisson_surface_reconstruction_3figintroduction

Polygon Meshes: Grundlagen

- Knoten, Kanten und Oberflächen, die Polyeder bestimmen
- Gespeichert wird also die Oberfläche (B-Rep)
- (Ausnahme: Volumetric Meshes)
- Geometrie + Topologie
  - Wo liegen die Punkte?
  - Wie sind sie verbunden?

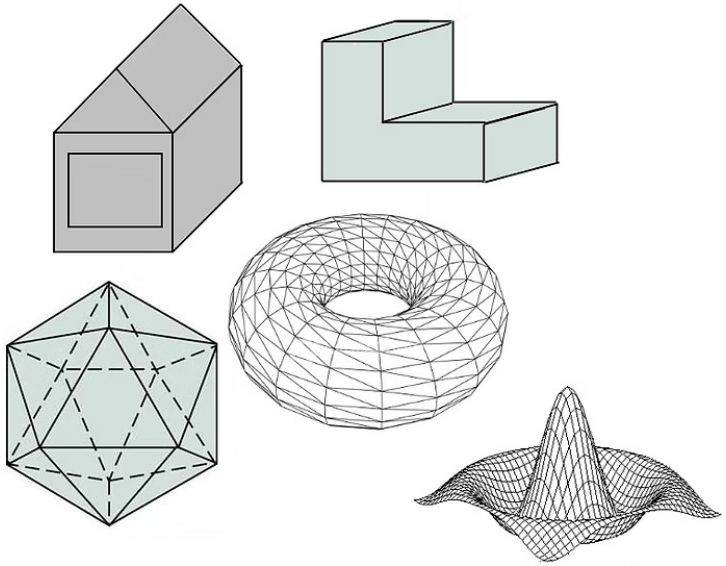


Bild: Crookshanks Academy

https://www.crookshanksacademy.com/post/what-is-a-polygon-mesh-and-its-representations

Polygon Meshes: Grundlagen

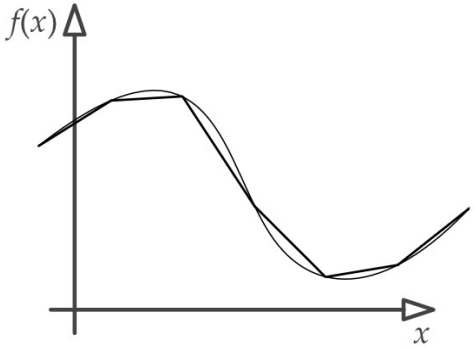
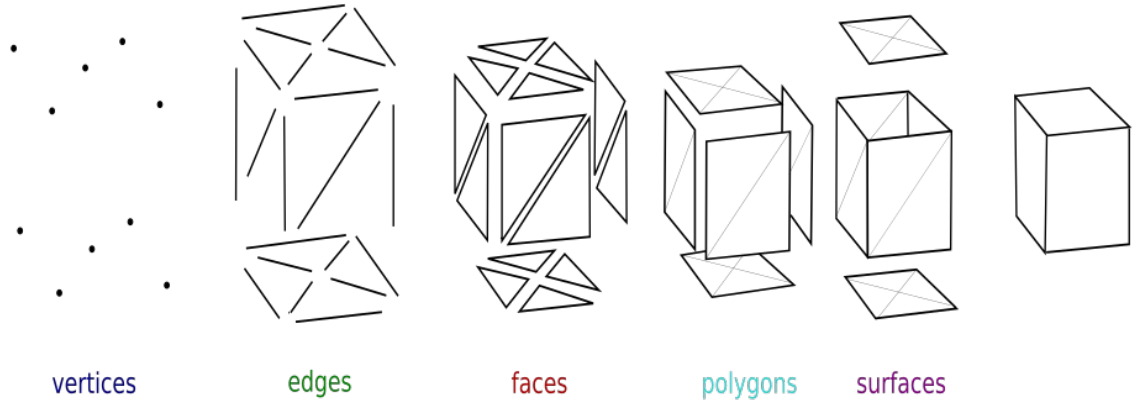


Figure 10.2. A mesh of triangles is a linear approximation to a surface, just as a series of connected line segments can serve as a linear approximation to a function or curve.

Bild:
Gregory, J. (2015). Game Engine Architecture. Philadelphia, PA: CRC Press, p. 447.

Polygon Meshes: Grundlagen



Ein paar wichtige Grundbegriffe

Bild:
Wikipedia

https://commons.wikimedia.org/wiki/File:Mesh_overview.svg

Polygon Meshes: Grundlagen

- **Vertex** (dt. **Knoten**): Position im Raum
- **Edge** (dt. **Kante**): Verbindung zweier Knoten
- **Face** (dt. **Fläche**): Geschlossene Menge an Kanten
- **Polygon**: Coplanare Menge an Flächen; äquivalent zu diesen, falls mehrseitige Flächen unterstützt (bzw. nur Dreiecke verwendet) werden

Meshes: Warum Dreiecke?

- Dreiecke sind das einfachste Polygon
- Dreiecke bestimmen immer eine ebene Fläche
- Bleiben auch unter Transformation (fast) immer Dreiecke
- Moderne GPUs sind für Dreiecke optimiert
- Können so jede Fläche approximieren
- Viele Renderer unterstützen höhere Polygone oder können diese zur Laufzeit in Dreiecke umrechnen

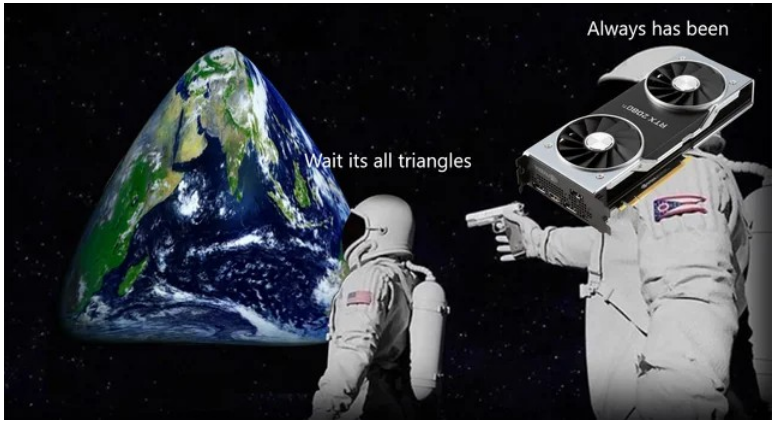


Bild: Reddit (u/UnnervingS auf r/ProgrammerHumor)

https://www.reddit.com/r/ProgrammerHumor/comments/jmix78/an_accurate_depiction_of_me_learning_to_utilise/

Meshes: Warum Dreiecke?

- Dreiecke sind das einfachste Polygon
- Dreiecke bestimmen immer eine ebene Fläche
- Bleiben auch unter Transformation (fast) immer Dreiecke
- Moderne GPUs sind für Dreiecke optimiert
- Können so jede Fläche approximieren
- Viele Renderer unterstützen höhere Polygone oder können diese zur Laufzeit in Dreiecke umrechnen

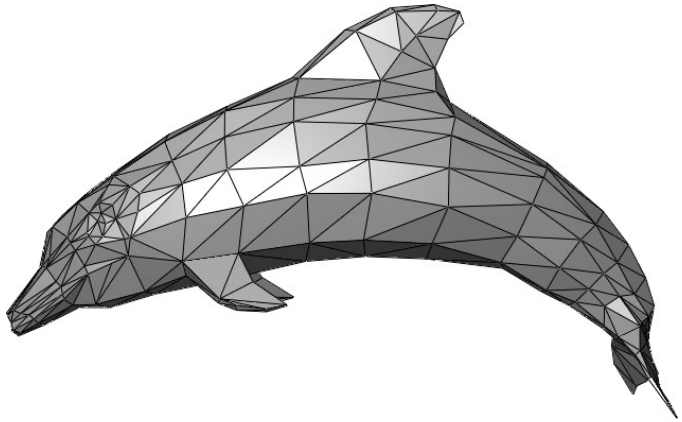


Bild: Wikipedia

https://commons.wikimedia.org/wiki/File:Dolphin_triangle_mesh.png

Meshes: Vertex-Vertex

- Einfache Repräsentation: Knoten und ihre Nachbarn
- Flächeninformation nur implizit
- Effiziente Speicherung
- Effizient für komplexe Änderungen in der Geometrie

Vertex-Vertex Meshes (VV)

	Vertex List	
v0	0,0,0	v1 v5 v4 v3 v9
v1	1,0,0	v2 v6 v5 v0 v9
v2	1,1,0	v3 v7 v6 v1 v9
v3	0,1,0	v2 v6 v7 v4 v9
v4	0,0,1	v5 v0 v3 v7 v8
v5	1,0,1	v6 v1 v0 v4 v8
v6	1,1,1	v7 v2 v1 v5 v8
v7	0,1,1	v4 v3 v2 v6 v8
v8	.5,.5,1	v4 v5 v6 v7
v9	.5,.5,0	v0 v1 v2 v3

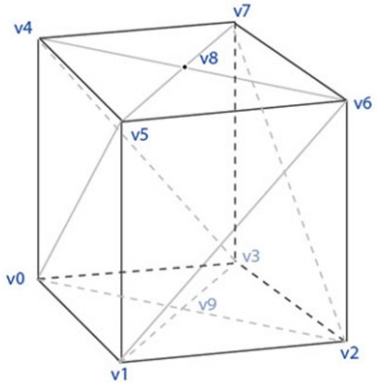


Bild: Wikipedia

https://commons.wikimedia.org/wiki/File:Vertex-Vertex_Meshes_(VV).png

Meshes: Face-Vertex

- Speichert für jede Fläche angrenzende Knoten
- (und/oder) für jeden Knoten angrenzende Fläche
- Explizite Flächen, implizite Kanten
- Indexed Face-Set
  - Vertexes: [(0,0,0),(0,0,1),(0,1,0),...]
  - Faces: [(0,1,2),...]

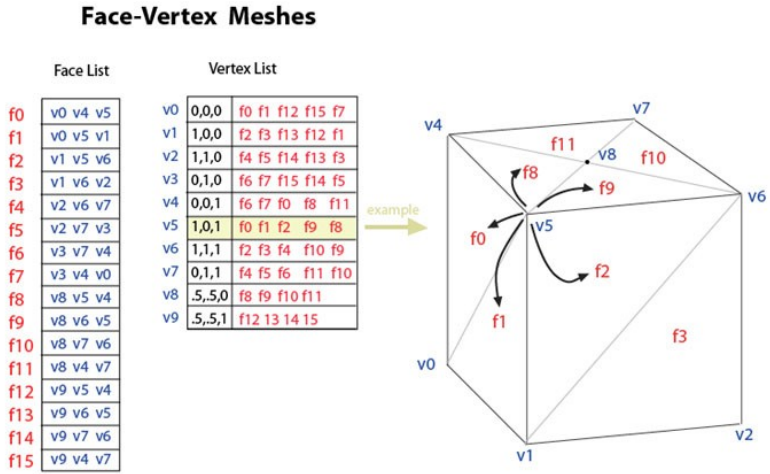


Bild: Wikipedia

https://commons.wikimedia.org/wiki/File:Mesh_fv.jpg

Meshes: Winged Edge

- Speichert Knoten und Flächen
- und Kanten: aber für jede Kante nur vier!
- Flexibel für Bearbeitung der Geometrie
- Schnelles Split, Merge, Traversieren (Kollisionserkennung), Subdivision
- Höherer Speicherbedarf, höhere Komplexität

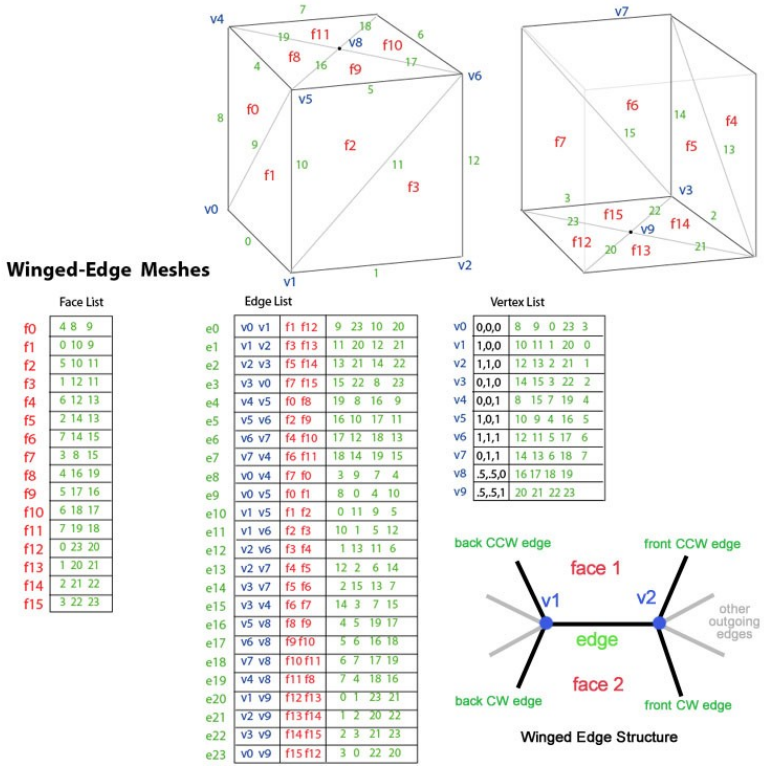


Bild: Wikipedia

https://commons.wikimedia.org/wiki/File:Mesh_we2.jpg

Meshes: Weitere

- Render dynamic: für GPU Rendering optimiertes Winged Edge Mesh
- Streaming: Flächen geordnet aber unabhängig speichern
- Progressive: Erhöhe das Level of Detail progressiv

Operation		Vertex-vertex	Face-vertex	Winged-edge	Render dynamic
V-V	All vertices around vertex	Explicit	V → f1, f2, f3, ... → v1, v2, v3, ...	V → e1, e2, e3, ... → v1, v2, v3, ...	V → e1, e2, e3, ... → v1, v2, v3, ...
E-F	All edges of a face	F(a,b,c) → {a,b}, {b,c}, {a,c}	F → {a,b}, {b,c}, {a,c}	Explicit	Explicit
V-F	All vertices of a face	F(a,b,c) → {a,b,c}	Explicit	F → e1, e2, e3 → a, b, c	Explicit
F-V	All faces around a vertex	Pair search	Explicit	V → e1, e2, e3 → f1, f2, f3, ...	Explicit
E-V	All edges around a vertex	V → {v,v1}, {v,v2}, {v,v3}, ...	V → f1, f2, f3, ... → v1, v2, v3, ...	Explicit	Explicit
F-E	Both faces of an edge	List compare	List compare	Explicit	Explicit
V-E	Both vertices of an edge	E(a,b) → {a,b}	E(a,b) → {a,b}	Explicit	Explicit
Flook	Find face with given vertices	F(a,b,c) → {a,b,c}	Set intersection of v1,v2,v3	Set intersection of v1,v2,v3	Set intersection of v1,v2,v3
Storage size		V*avg(V,V)	3F + V*avg(F,V)	3F + 8E + V*avg(E,V)	6F + 4E + V*avg(E,V)
		Example with 10 vertices, 16 faces, 24 edges:			
		10 * 5 = 50	3*16 + 10*5 = 98	3*16 + 8*24 + 10*5 = 290	6*16 + 4*24 + 10*5 = 242

Figure 6: summary of mesh representation operations

Bild: Wikipedia

https://en.wikipedia.org/wiki/Polygon_mesh

Meshes: Anwendungen

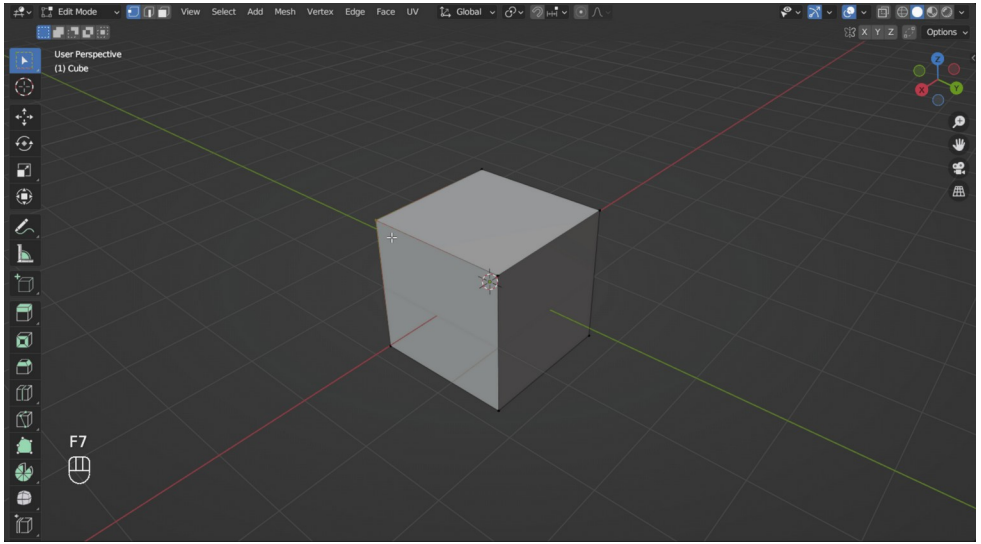
- Vielseitig:
- Games / Echtzeitgrafik
- CAD / Fertigung
- Simulation

Bild: Wikipedia

https://en.wikipedia.org/wiki/Polygon_mesh

Meshes: Generation

- Rekonstruktion aus Sensordaten
- Prozedual
- Konvertierung aus anderen Repräsentationen
- Manuelle Modellierung



Aufnahme: Universität Tôhoku

https://gp-ds.tohoku.ac.jp/media/curriculum/course/visual-computing/markdown/modeling/mesh.html

Mesh-Generation: Ausblick

- Machine Learning kann Mesh-Erstellung unterstützen
- Zumeist Point Cloud als Input und Triangle Mesh als Output
- z.B.: MeshAnything, Nvidia Meshtron

Bild: Nvidia

https://developer.nvidia.com/blog/high-fidelity-3d-mesh-generation-at-scale-with-meshtron/

Transformationen

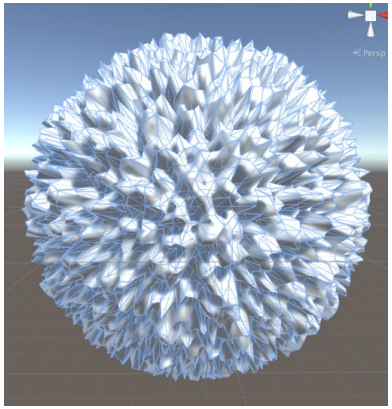
- Rigid Transforms
  - Translation
  - Rotation
- Smoothing
- Simplification
- Subdivision

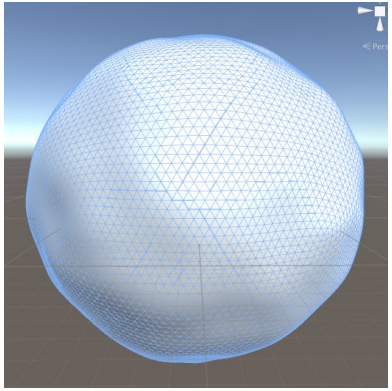
Bild: Wikipedia

https://en.wikipedia.org/wiki/Polygon_mesh

Transformationen: Smoothing

- Knoten wird anhand seiner Nachbarn angepasst
- Laplacian Smoothing
  - Neue Position = alte Position + Schritt richtung Nachbarschafts-Mittelwert
- Zu starkes Smoothing führt zu Schrumpfung oder Detailverlust





Bilder: GitHub (mattatz)

https://github.com/mattatz/unity-mesh-smoothing

Transformationen: Simplification

- Anzahl der Knoten und Flächen reduzieren
- Edge Collapse: Kante zu einem Punkt zusammenziehen
- Wahl der Kante: Quadratic error metrics
- Wenn zu stark vereinfacht verschwinden Features, Artefakte, Löcher

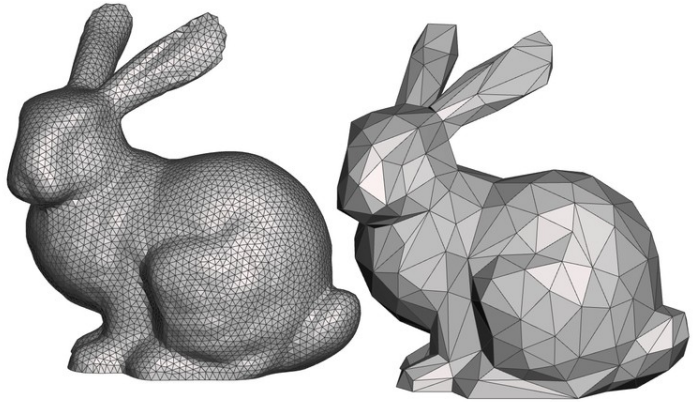


Bild: GradientSpace

https://www.gradientspace.com/tutorials/2017/8/30/mesh-simplification

Transformationen: Subdivision

- Aus groberem Mesh feineres erstellen
- Catmull-Clark
  - Unterteile jede Fläche iterativ in feinere Vierecke
  - Knotenpositionen werden verändert um eine glattere Fläche zu erhalten
- Modeling, Animation

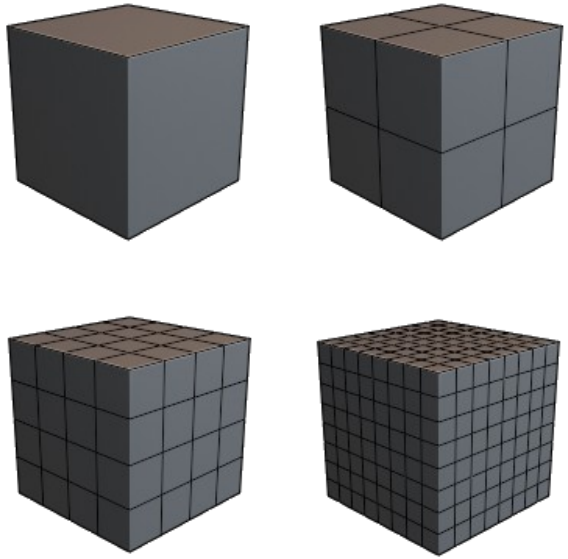


Bild: Wikipedia

https://commons.wikimedia.org/wiki/File:Cube_simple_subdivisions_(0-3).png

Rendering

- Normale
- UV-Mapping
- Tesselation

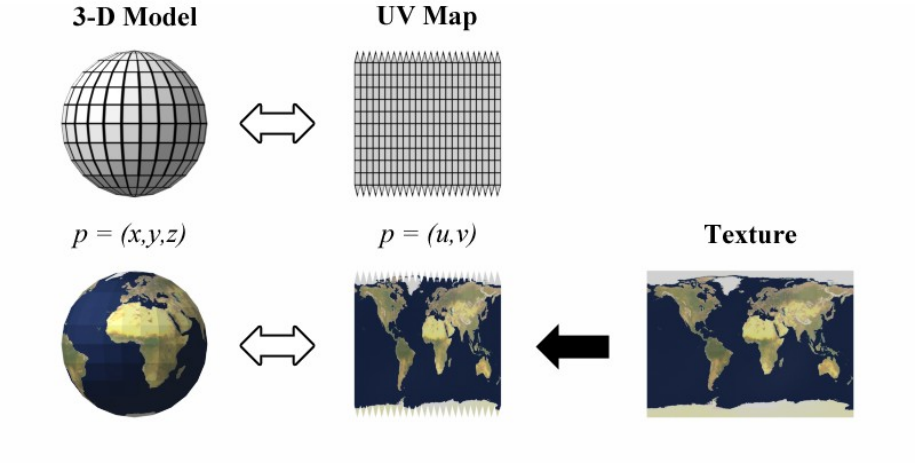


Bild: Wikipedia

https://commons.wikimedia.org/wiki/File:UVMapping.png

Rendering: Tesselation

- Zerlegung einer Oberfläche in kleinere Primitive
- Erhöht die Auflösung der Oberfläche
- Detailgrade können dynamisch hinzugefügt / entfernt werden (Camera Distance)
- Grobes Mesh → Detailliertes Mesh

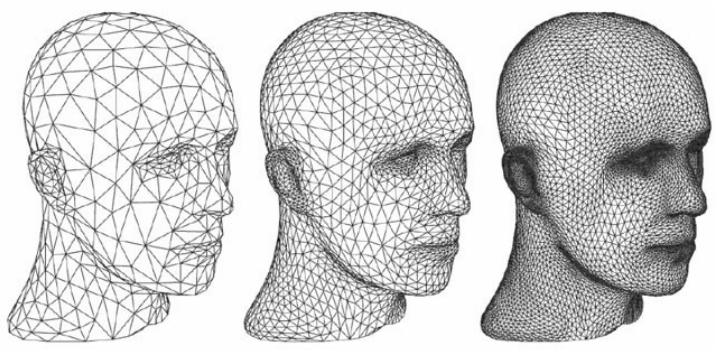


Bild: StackExchange

https://computergraphics.stackexchange.com/a/2020

Analyse und Simulation

- Nachbarschaften
  - Benachbarte Vertices, Faces
  - Smoothing, Shortest Path, ...
- Boundary Detection
  - Löcher erkennen (und reparieren)
- Shortest Path
  - Dijkstra
- Collision Detection

Bild: Wikipedia

https://en.wikipedia.org/wiki/Polygon_mesh