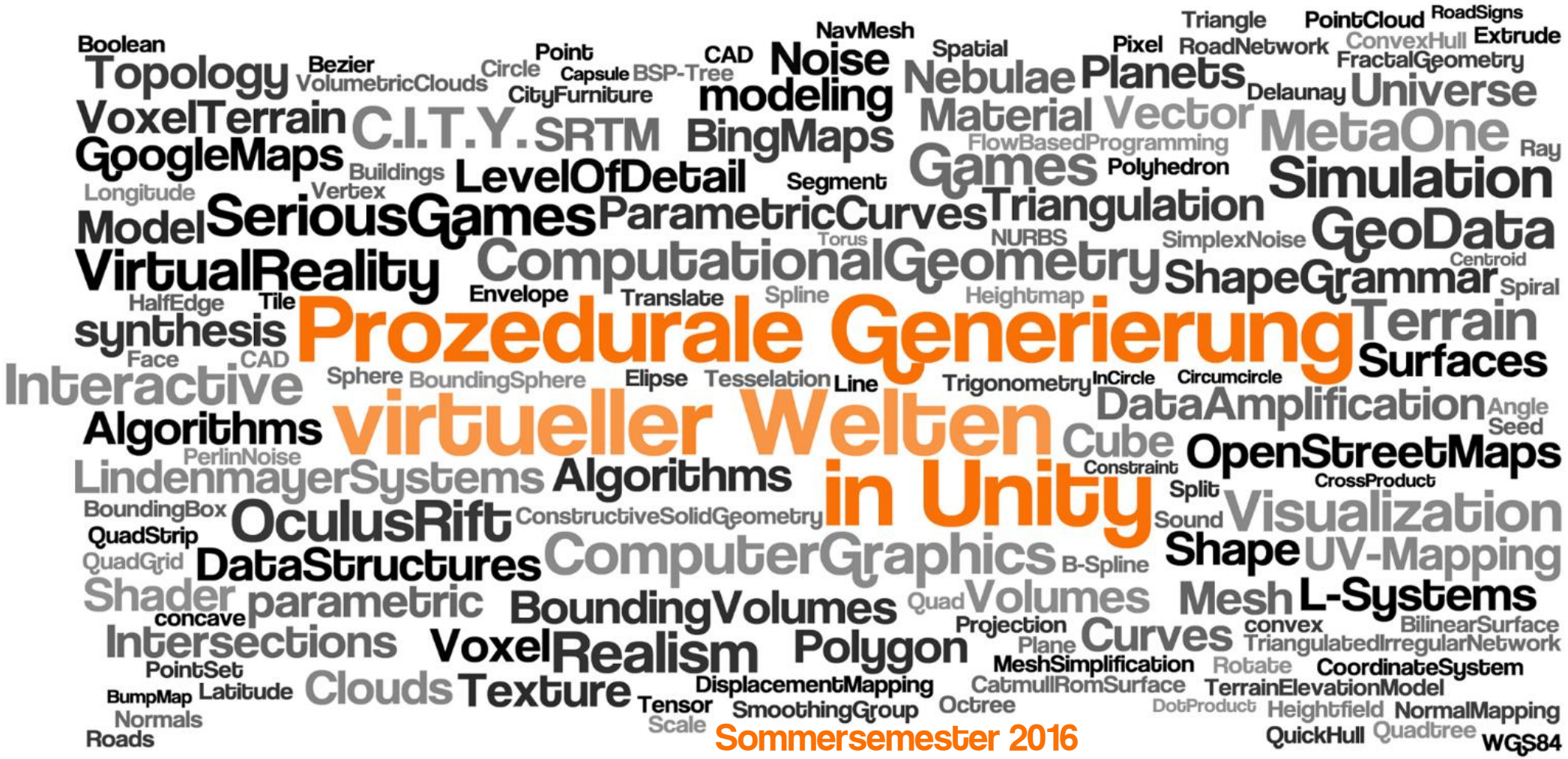
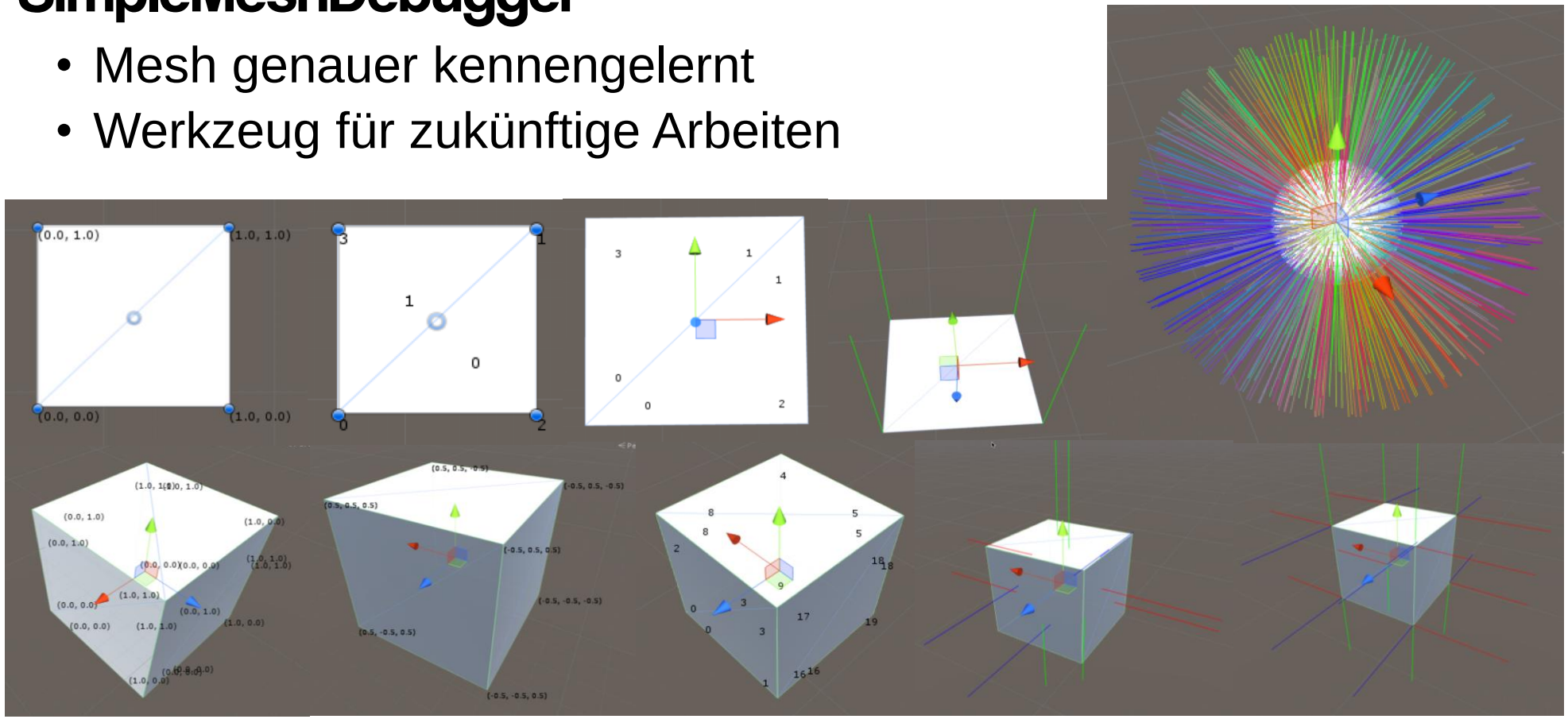

Prozedurale Generierung
virtueller Welten
in Unity
Sommersemester 2016
Boolean
Topology
Bezier
VolumetricClouds
Circle
Point
Capsule
BSP-Tree
CityFurniture
CAD
Noise
modeling
NavMesh
Spatial
Nebulae
Pixel
Planets
Triangle
PointCloud
RoadSigns
RoadNetwork
ConvexHull
Extrude
FractalGeometry
Delaunay
Universe
VoxelTerrain
C.I.T.Y.
SRTM
BingMaps
Material
Vector
MetaOne
FlowBasedProgramming
Ray
GoogleMaps
Buildings
LevelOfDetail
Segment
Games
Polyhedron
Simulation
Longitude
Vertex
Model
SeriousGames
ParametricCurves
Triangulation
GeoData
Torus
NURBS
SimplexNoise
VirtualReality
ComputationalGeometry
Centroid
ShapeGrammar
Spiral
HalfEdge
Tile
Envelope
Translate
Spline
Heightmap
synthesis
Terrain
Face
CAD
Surfaces
Interactive
Sphere
BoundingSphere
Elipse
Tesselation
Line
Trigonometry
InCircle
Circumcircle
DataAmplification
Angle
Seed
Algorithms
Cube
OpenStreetMaps
PerlinNoise
LindenmayerSystems
Algorithms
Constraint
CrossProduct
Split
BoundingBox
OculusRift
ConstructiveSolidGeometry
Sound
Visualization
QuadStrip
QuadGrid
DataStructures
ComputerGraphics
B-Spline
Shape
UV-Mapping
Shader
parametric
BoundingVolumes
Quad
Volumes
Mesh
L-Systems
concave
Projection
convex
BilinearSurface
Intersections
VoxelRealism
Polygon
Plane
Curves
TriangulatedIrregularNetwork
PointSet
MeshSimplification
Rotate
CoordinateSystem
BumpMap
Latitude
Clouds
Texture
DisplacementMapping
CatmullRomSurface
TerrainElevationModel
Tensor
SmoothingGroup
Octree
DotProduct
Heightfield
NormalMapping
Normals
Scale
QuickHull
Quadtree
WGS84
Roads

Rückblick...

- **SimpleMeshDebugger**
  - Mesh genauer kennengelernt
  - Werkzeug für zukünftige Arbeiten



Tagesziele

- **Verwendung von Submeshes**
- Zur Wiederholung: **Materialien aus Datei laden**

- Projekt 5 – **Cube2House** → **Übung**
  - **Materialien**
  - **UV-Mapping-Varianten**
  - **SubMeshes** kennenlernen
  - **Parametrisierung** von Objekten → Modell und UVs

Submeshes - Wozu?

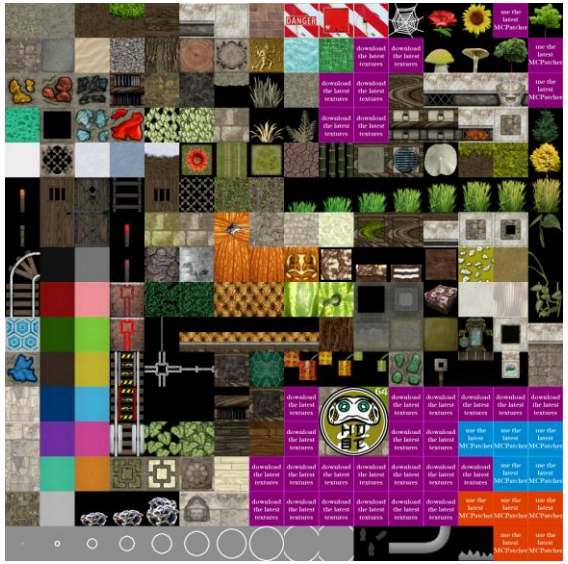
- **Aufgabe:**

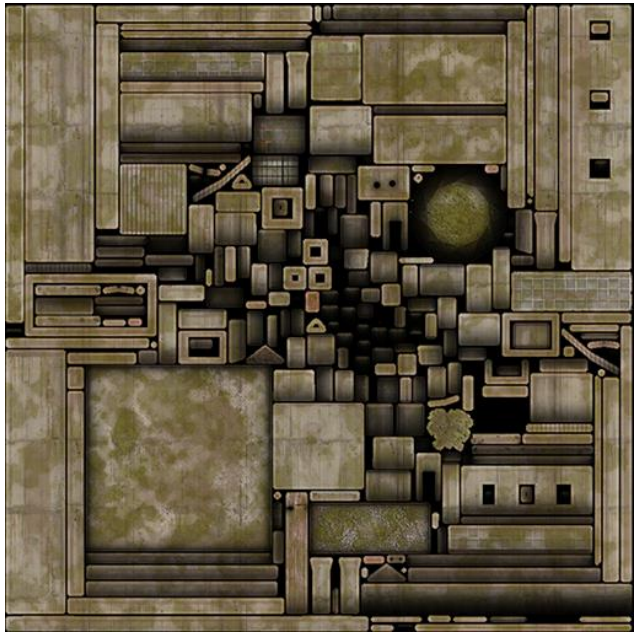
  `Mesh mit mehreren Materialien texturieren…`

- Varianten:
  - **Textur-Atlas verwenden**
  - **Submeshes verwenden**

Textur-Atlas

- Wenn alle UV-Koordinaten in eine Textur eindeutig zuweisbar sind (zwischen 0..1)
- Nicht geeignet, wenn Mesh parametrisiert sein sollte

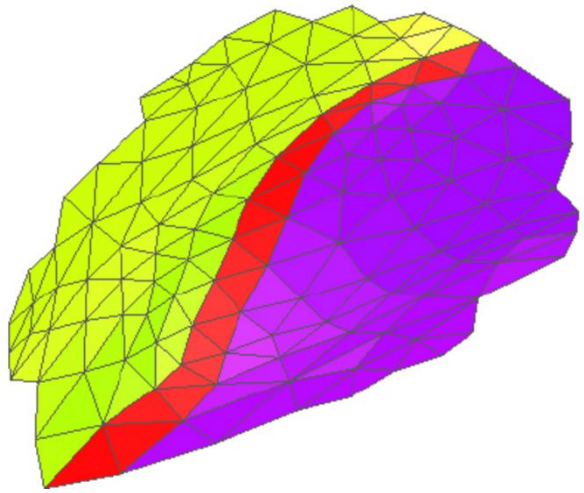




Submeshes

- Wir sammeln jeweils die **Triangles** in einem Submesh, die das **gleiche Material** bekommen sollen
- Durch den Wrap-Mode „repeat“ gut verwendbar bei Parametrisierung
  - UV-Koordianten können auch Werte außerhalb Einheitsfläche haben

- Heute dazu: **Projekt 5 Cube2House**



Verwenden von Submeshes

- **Für jedes Submesh ein eigenes Triangle-Array (int[])**
  - Trotzdem nur ein Vertex- und UV-Array
  - Positionen die mehrfach verwendet werden,
    auch weiterhin als eigenständige Vertices im Vertex-Array anlegen

- **Erst: Setzen des SubMeshCounts beim Mesh**
  - `myMesh.subMeshCount = 3;`
  - !!!MUSS!!! manuell gesetzt werden.

- **Dann: setzen der Triangles für jedes SubMesh**
  - subMesh-Index angeben
  - `mesh.SetTriangles(triangles, index);`
  - Dieser Index wird dazu verwendet das Material zuzuweisen (synonyme Verwendung der Indices im Material-Array des MeshRenderers)

Material aus Texturdatei

- **Material aus Texturdatei anlegen in 3 Schritten**

  1. **Anlegen eines Materials** über `new Material(…);`
     - Benötigt einen Shader, `Shader.Find(„shadername“);`
     - StandardShader ist `„Diffuse“`
     - Also:

```
Material myNewMaterial = new Material(Shader.Find("Diffuse"));
```

  2. **Laden der Datei** über `Resources.Load(„name“)` **als** `Texture`
     - Textur-Dateien müssen in einem Ordner `„Resources“` liegen
     - Liegt die Datei in einem weiteren Unterordner? → „“

```
Texture myLoadedTexture = Resources.Load(„myImageFile“)
```

  3. Dem Material die **geladene Textur zuweisen**
     - `myNewMaterial.maintexture = myLoadedTexture;`

- Für alle Materialien Wiederholen und diese in einem `Texture`-array speichern
- Das neue **Material-Array** dem **MeshRenderer übergeben**

Texturen aus Datei Laden

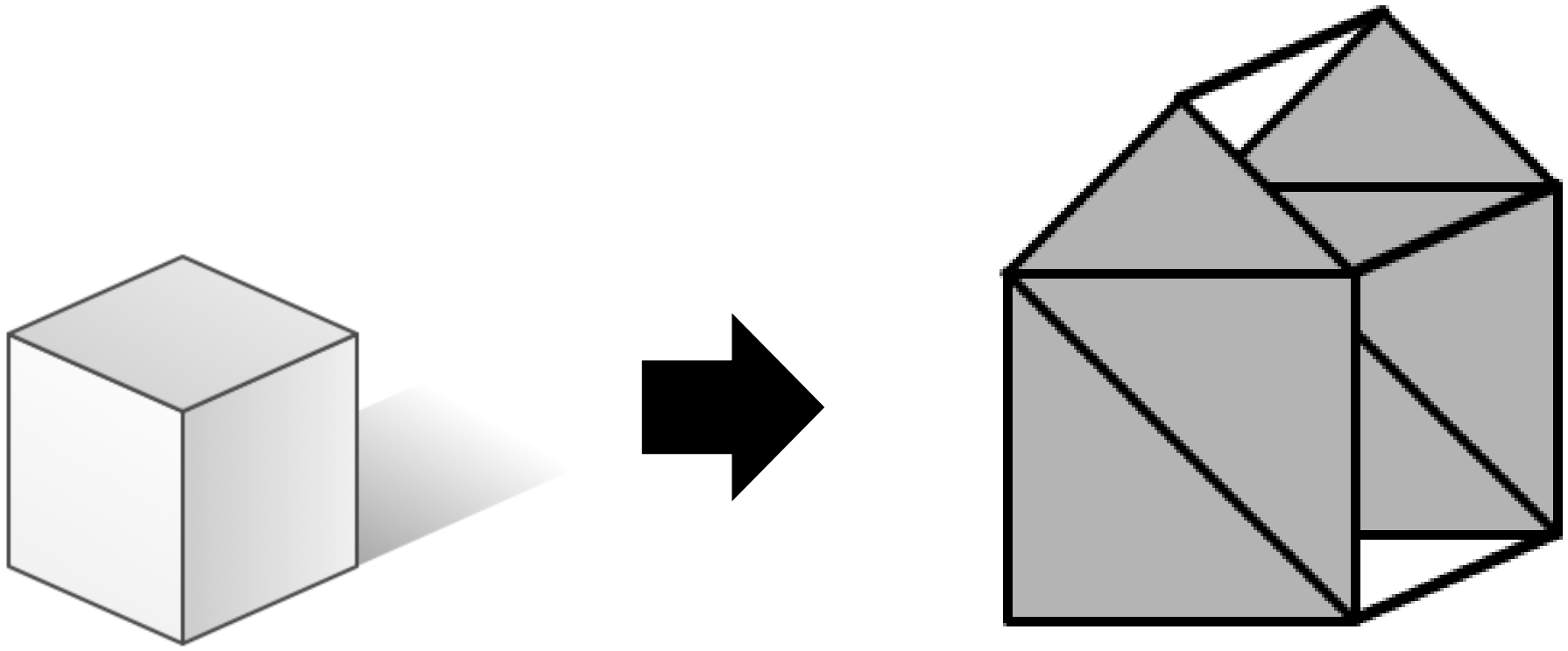
```
Material[] materials = new Material[3];
materials[0] = new Material(Shader.Find("Diffuse"));
materials[0].mainTexture = Resources.Load("brick") as Texture;

materials[1] = new Material(Shader.Find("Diffuse"));
materials[1].mainTexture = Resources.Load("brickWhite") as Texture;

materials[2] = new Material(Shader.Find("Diffuse"));
materials[2].mainTexture = Resources.Load("roof") as Texture;

meshRenderer.materials = materials;
```

Cube → Haus



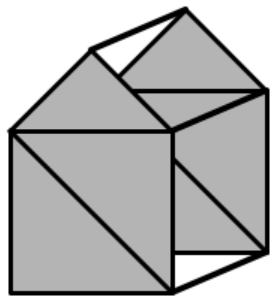
Was wollen wir machen?

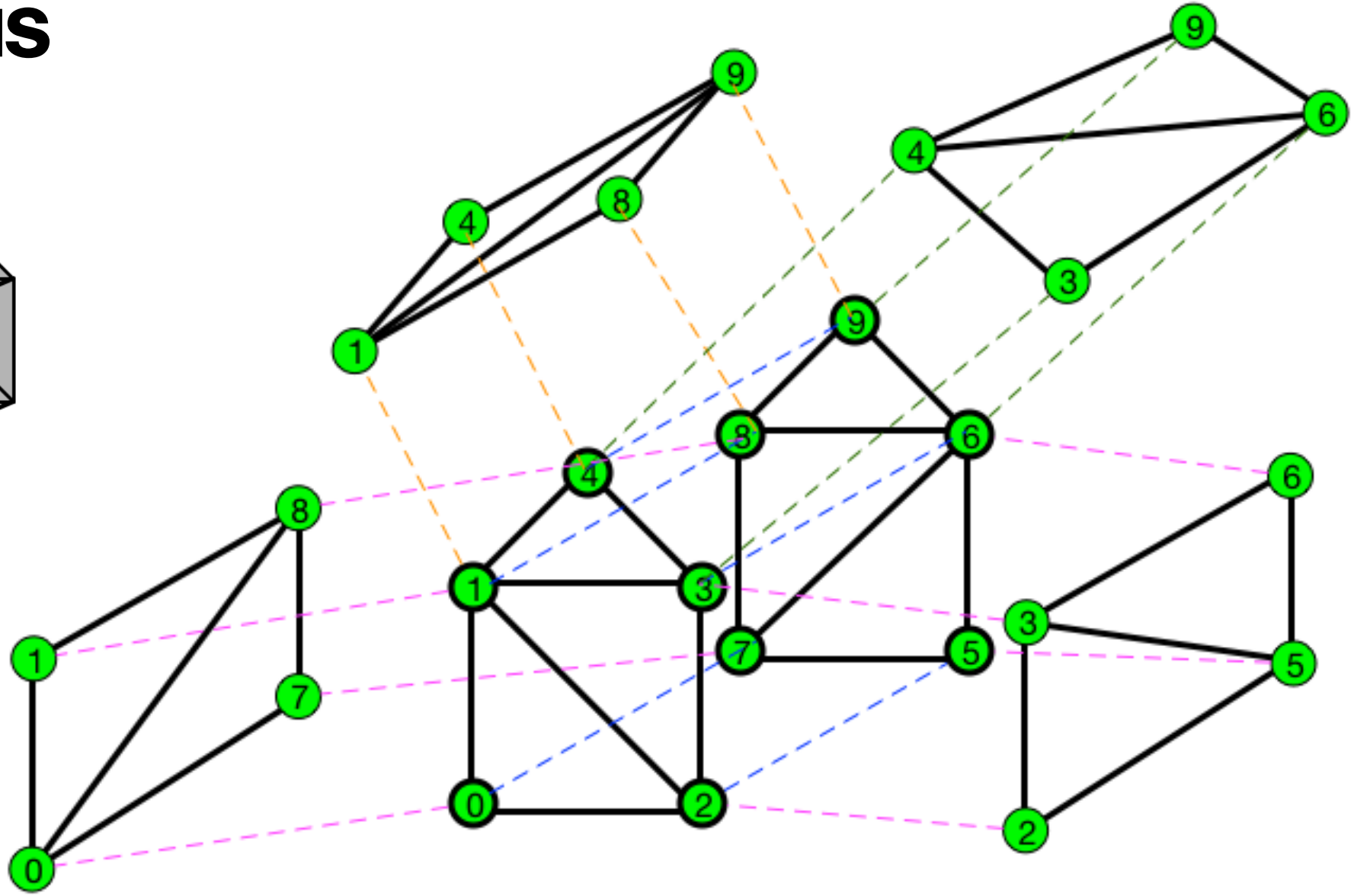
- **Parametrisieren** eines Modells
  - Parameter einführen
  - Editor für Parameter
- **UV-Mapping parametrisiert**
  - Anhängig von eingestellten Größenparametern
- **Submeshes**
  - Unterteilung der Triangles nach Zugehörigkeit/Material
  - Texturen aus Datei laden
- Zufälliges **Erstellen** und **Platzieren** von Häusern

Ausgangssituation

- UnityCube – 8 Positionen
- Faces:
  - Front 2 Triangles 4 Vertices
  - Back 2 Triangles 4 Vertices
  - Left 2 Triangles 4 Vertices
  - Right 2 Triangles 4 Vertices
  - Top 2 Triangles 4 Vertices
  - Bottom 2 Triangles 4 Vertices
  - **Gesamt 12 Triangles 24 Vertices**
- 1 Mesh
  - Keine submeshes
  - Standard Diffuse-Material

Haus



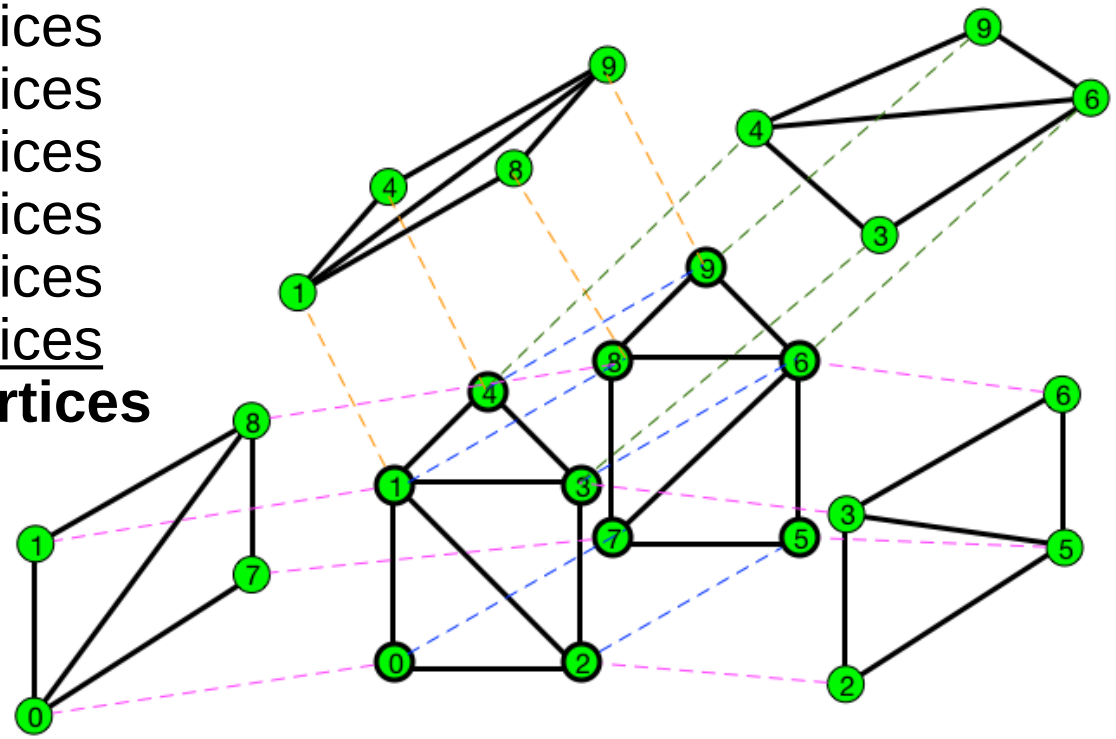


Zielzustand

- Haus – 10 Positionen
  - Über Parameter berechnet
- Faces:

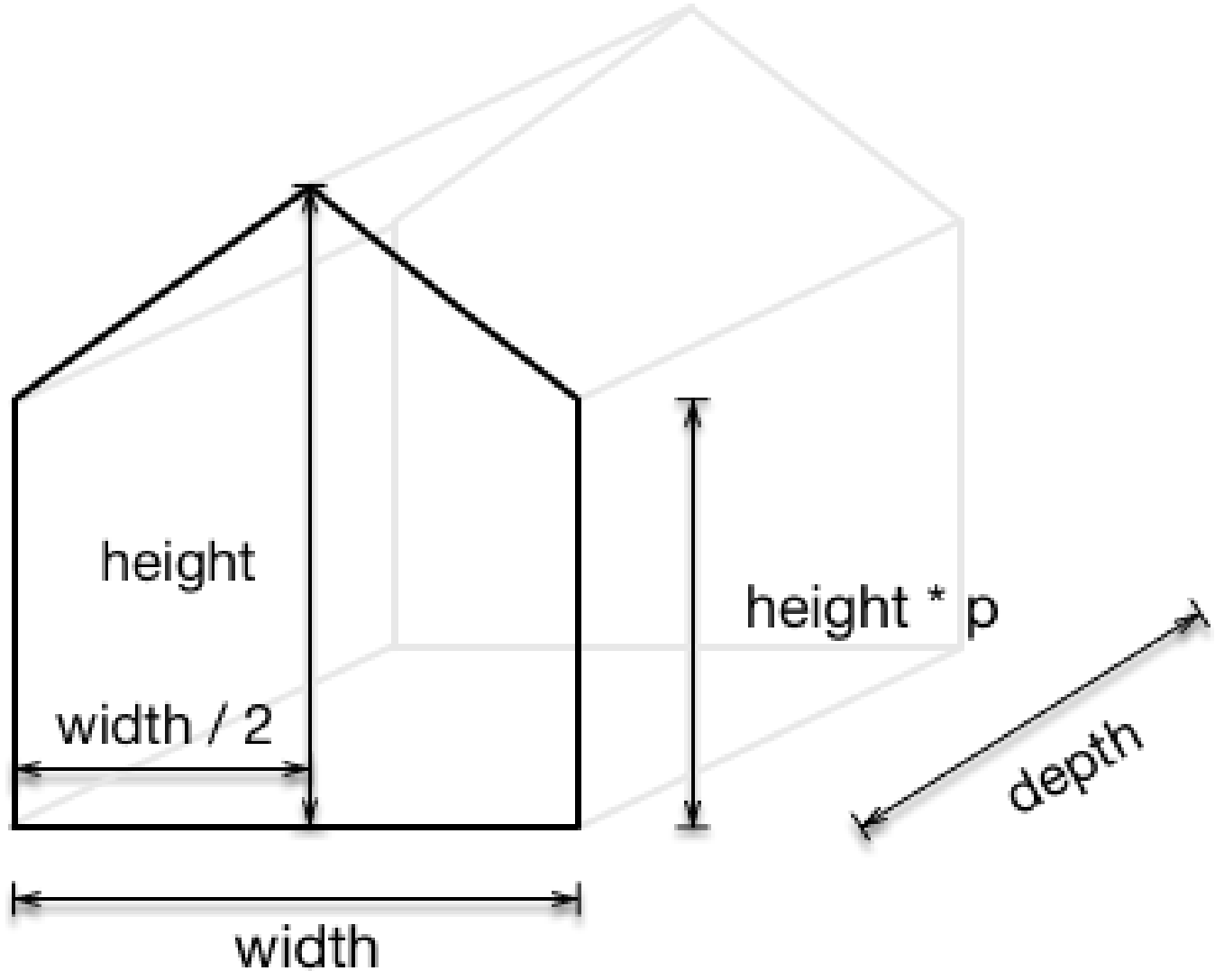
Front	3 Triangles	5 Vertices
Back	3 Triangles	5 Vertices
Left	2 Triangles	4 Vertices
Right	2 Triangles	4 Vertices
RoofLeft	2 Triangles	4 Vertices
RoofRight	2 Triangles	4 Vertices
Gesamt	**14 Triangles**	**26 Vertices**

- 1 Mesh → **3 SubMeshes**
  - FrontBack, Sides, Roof
  - 3 per Script geladene Materialien

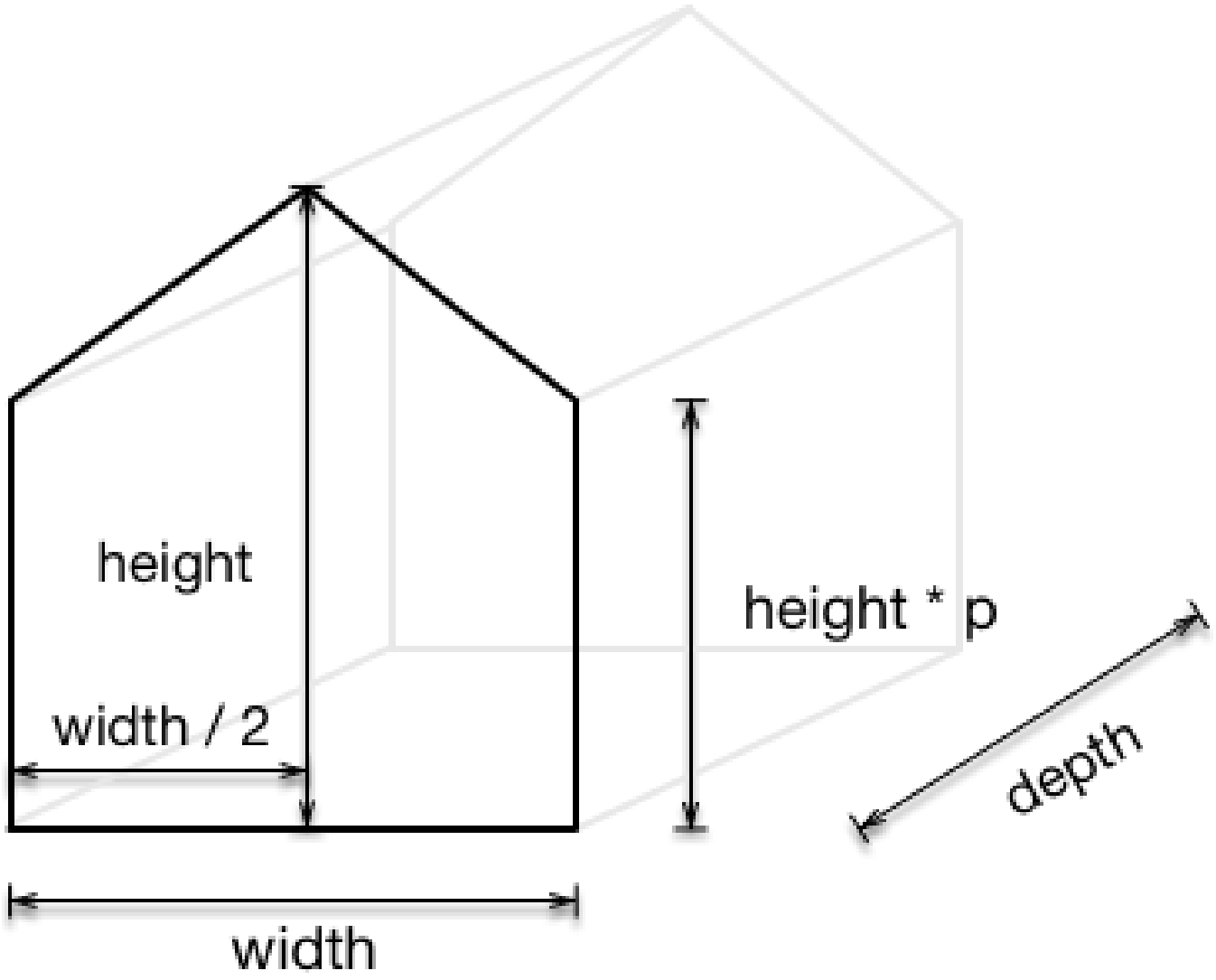


Parameter

- `float Height`
- `float Width`
- `float Depth`
- `float WallheigtParameter (p)`
  - Werte: 0…1
  - Anteil der Wandhöhe zur Gesamthöhe
  - Im Beispiel: p = 0.7



Parameter



UV-Mapping