Geometrie

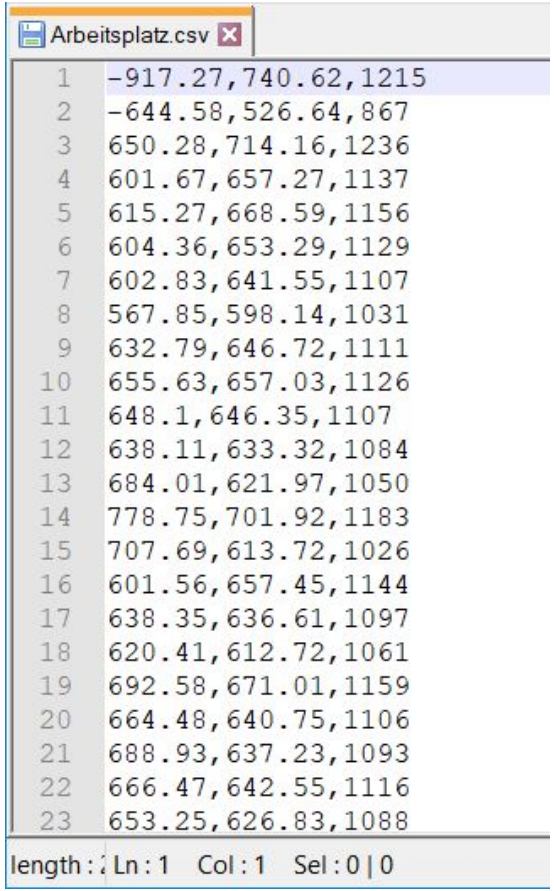
Grundformen, 3D Modelle, Transformationen

3D Modelle

Ausgangsdaten für die Modellierung

- ➔ Punktwolken (x/y/z … /r/g/b/a)
- ➔ Mesh, Gitter
  - ◆ Triangles
  - ◆ Quads (Vierecke)
- ➔ “Vermaschung” von Punktwolken
  - ◆ Von Hand: Voronoi Triangulation
  - ◆ Punkte zu Triangles/Quads machen
  - ◆ Tools: MeshLab ...

https://de.wikipedia.org/wiki/Liste_von_Programmen_zur_Punktwolkenverarbeitung



3D Modelle

Beispiel Kinect Streams:

- ➔ Berechnung der z-Koordinate aus Tiefenbild (0,5m - 8m mit 1mm Auflösung)
- ➔ Berechnung der x/y Koordinaten aus Apertur (Öffnungswinkel)

- ➔ Werkskalibrierung:
  Berücksichtigen von Verzeichnungsfehlern

(Quelle: Wikipedia)

3D Modelle

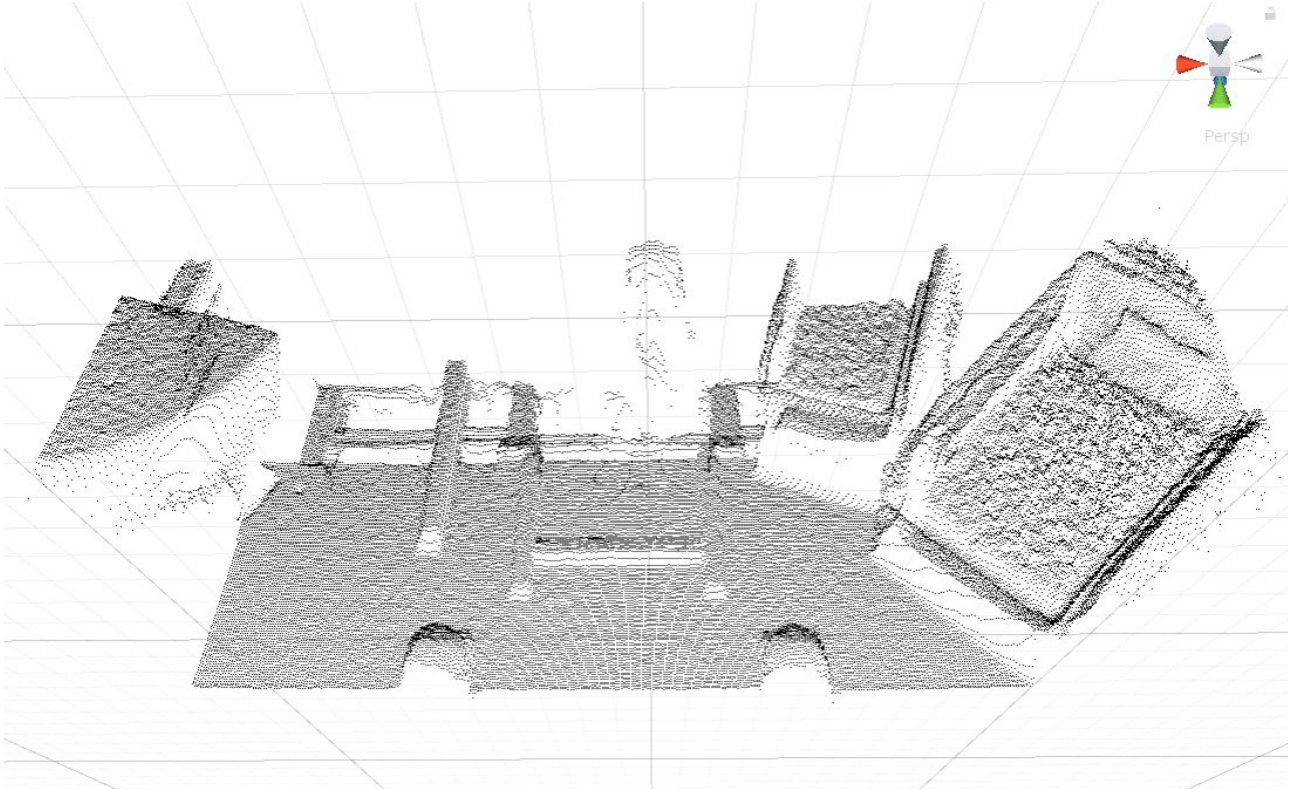
Beispiel Kinect: Matching von Tiefenbild mit Full-HD Bild

- ➔ Bereits berechnet: Winkel(Pixel; Tiefenbild) für horizontal und vertikal
- ➔ Gleiche Rechnung für Winkel(Pixel; Full-HD Bild); andere Apertur!
- ➔ Berücksichtigung Werkskalibrierung
  - ◆ Abstand der beiden Sensoren zueinander
  - ◆ Ausrichtung der beiden Sensoren zueinander

- ➔ Dann: Pixel(Winkel(Pixel; Tiefenbild); Full-HD Bild)

- ➔ Damit: Farbinformation pro Vertex (Vertex-Shader vs. Pixel-Shader)

Verarbeitung mit Unity



C# Skript

Anforderung: Informationen werden zur Laufzeit eingelesen und verarbeitet.

Streamreader

Header:

```
StreamReader sr = new StreamReader(Application.dataPath + dPath);

sr.ReadLine();
string[] buffer = sr.ReadLine().Split();
var numPoints = int.Parse(buffer[0]);

Vector3[] vertices = new Vector3[numPoints];
int[] indecies = new int[numPoints];
Color[] vertexColors = new Color[numPoints];
```

C# Skript

Zeilenweise verarbeiten

```
buffer = sr.ReadLine().Split();

var pos = new Vector3(
    float.Parse(buffer[0]) * ScaleDimensions,
    float.Parse(buffer[1]) * ScaleDimensions,
    float.Parse(buffer[2]) * ScaleDimensions);

var col = Color.black;
if (buffer.Length >= 5)
    col = new Color(
        int.Parse(buffer[3]) / 255.0f,
        int.Parse(buffer[4]) / 255.0f,
        int.Parse(buffer[5]) / 255.0f);

//createSphere(pos, col, parent);

vertices[i] = pos;
indecies[i] = i;
vertexColors[i] = col;
```

C# Skript

Mesh dynamisch erzeugen

Objekt erzeugen:

```
GameObject pointGroup = new GameObject("PointCloudMesh");
pointGroup.AddComponent<MeshFilter>();
pointGroup.AddComponent<MeshRenderer>();
pointGroup.GetComponent<Renderer>().material = VertexMaterial;

Mesh mesh = new Mesh();
mesh.vertices = vertices;
mesh.colors = vertexColors;
mesh.SetIndices(indecies, MeshTopology.Points, 0);
mesh.uv = new Vector2[numPoints];
mesh.normals = new Vector3[numPoints];

pointGroup.GetComponent<MeshFilter>().mesh = mesh;
```

UnityEngine.Mesh

- ➔ void Start() { … Mesh erzeugen … }
- ➔ void Update() { … Vertices modifizieren … }

https://docs.unity3d.com/ScriptReference/Mesh.html

Mesh-Objekt:

- ➔ Gute Performance
- ➔ Zugriff auf alle Relevanten Größen

Procedural Examples: Extrusion

```
var mesh : Mesh = GetComponent(MeshFilter).mesh;

if (baseVertices == null)
    baseVertices = mesh.vertices;

var vertices = new Vector3[baseVertices.Length];

var timex = Time.time * speed + 0.1365143;
var timey = Time.time * speed + 1.21688;
var timez = Time.time * speed + 2.5564;
for (var i=0;i<vertices.Length;i++)
{
    var vertex = baseVertices[i];

    vertex.x += noise.Noise(timex + vertex.x, timex + vertex.y, timex + vertex.z) * scale;
    vertex.y += noise.Noise(timey + vertex.x, timey + vertex.y, timey + vertex.z) * scale;
    vertex.z += noise.Noise(timez + vertex.x, timez + vertex.y, timez + vertex.z) * scale;

    vertices[i] = vertex;
}

mesh.vertices = vertices;
```

Procedural Examples

Per Vertext Operationen:

• Fractal Texture	Besser: Texture Offset im Shader programmieren.
• Heightmap Generator	Bei kleinen Strukturen: Bumpmapping
• Sculpt Vertices	+ Update Normalmap Echtzeit-Anforderungen?
• Sinus Curve Modifier	Nichtlineare Berechnungsfunktionen
• Smooth Random Position	Zufallszahlen
• Twist	Globale Mesh-Deformationen

Grundbegriffe

Vertex

Raumpunkt (dimensionslos) mit x/y/z Koordinate; Plural: Vertices

Weltkoordinaten

Allgemeines Koordinatensystem, auf dessen Ursprung sich alle Objekte referenzieren.

Objektkoordinaten

Objektspezifisches Koordinatensystem, dessen Ursprung meist im Schwerpunkt liegt. Der Ursprung wird auch Pivot-Punkt genannt.

Grundbegriffe

Gittermodell (engl. Mesh)

Benachbarte Vertices, die auf einer gemeinsamen Objektfläche liegen, werden über Kanten verbunden.

Dadurch werden i.d.R. Dreiecke (Triangles) oder Vierecke (Quads) gebildet.
Wichtig: beide Elemente bilden Planflächen (Rechenzeit!)

Die Berechnung eines Mesh aus einer Anzahl Vertices wird Vermaschung genannt.

Facette

Einzelnes Triangle oder Quad im Mesh.

Grundbegriffe

Normalenvektor

Vektor senkrecht zur Fläche. Wichtig für Beleuchtung und Strukturierung.

- pro Facette, vgl. Flat-Shading
- pro Vertex

Die Gesamtheit der Normalenvektoren heißt Normalenvektorfeld. Das Normalenvektorfeld beschreibt die Topologie des Objekts.

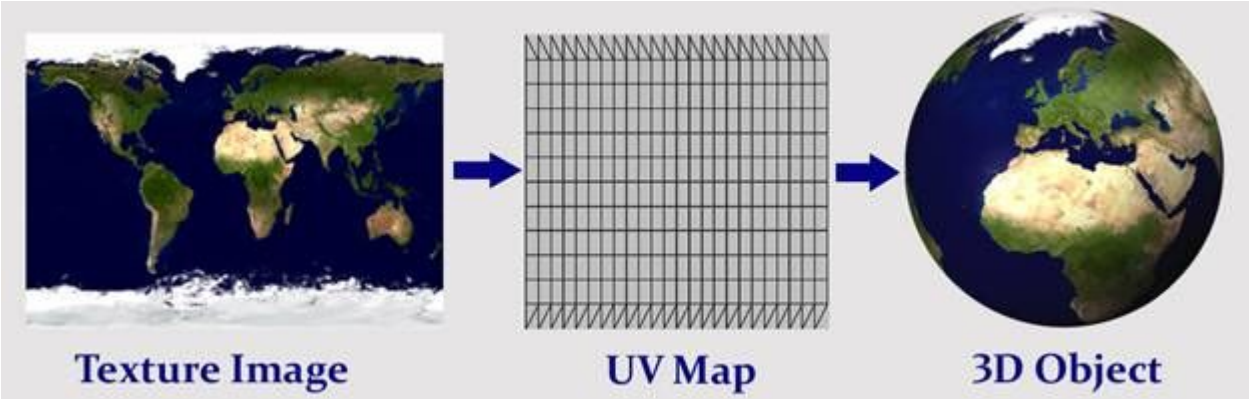
Die Außenseite zeigt in Richtung der Normalenvektoren.

Grundbegriffe

UV Map

Transformation von 3D Objektkoordinaten auf 2D “Textur”-Koordinaten.

Gleiches Prinzip wie bei Projektion Fisheye-Aufnahme nach Merkator-Darstellung.
Zur Erinnerung: Die UV Map einer Sphäre in Unity ist die Merkator-Darstellung.



(Quelle: blenderartist.org)

Grundbegriffe

Bildschirmkoordinaten

2D Koordinaten im gerenderten Bild.

Rendering

Berechnung des 2D Bildes auf Basis der 3D Szene.

Koordinatentransformationen

Homogene Koordinaten:

- Drehung **m**
- Translation **T**

“Hilfsgröße” **w**: Streckungsfaktor

(det <> 0 → inverse Transformation mgl.)

- Translation
- Rotation
- Skalierung
- ...

$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & T_x \\ m_{21} & m_{22} & m_{23} & T_y \\ m_{31} & m_{32} & m_{33} & T_z \\ 0 & 0 & 0 & m_{44} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

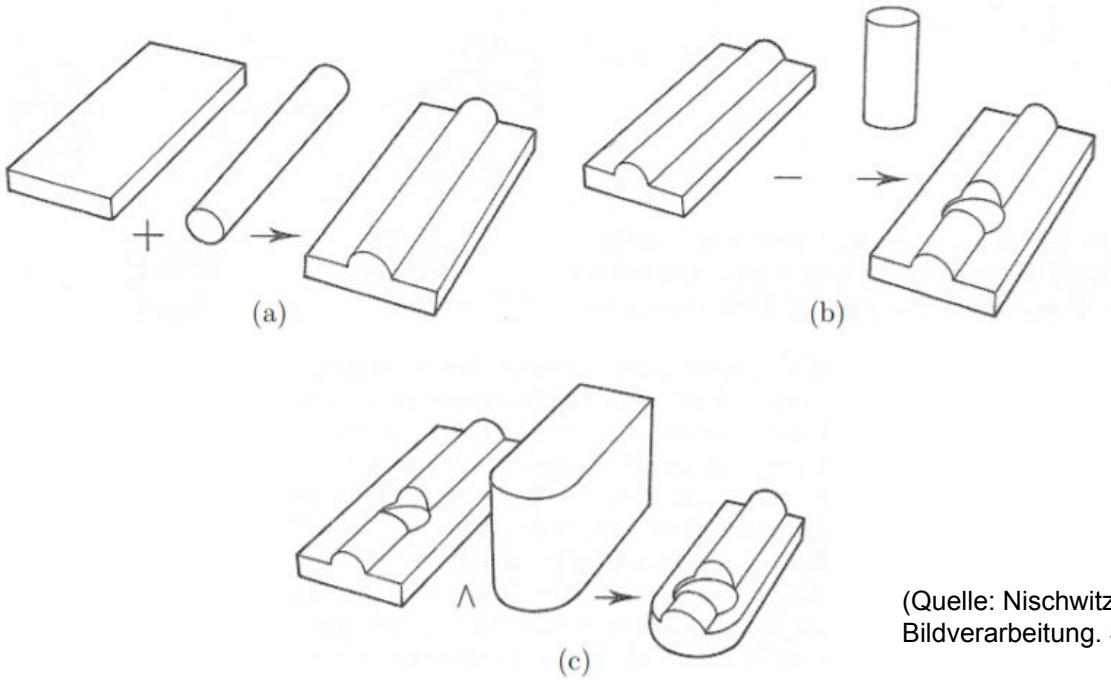
wobei

$$\begin{aligned} x' &= m_{11} \cdot x + m_{12} \cdot y + m_{13} \cdot z + T_x \cdot w \\ y' &= m_{21} \cdot x + m_{22} \cdot y + m_{23} \cdot z + T_y \cdot w \\ z' &= m_{31} \cdot x + m_{32} \cdot y + m_{33} \cdot z + T_z \cdot w \\ w' &= m_{44} \cdot w \end{aligned}$$

(Quelle: Nischwitz, A: Computergrafik und Bildverarbeitung. S. 125, Springer, 2011)

Modellieren

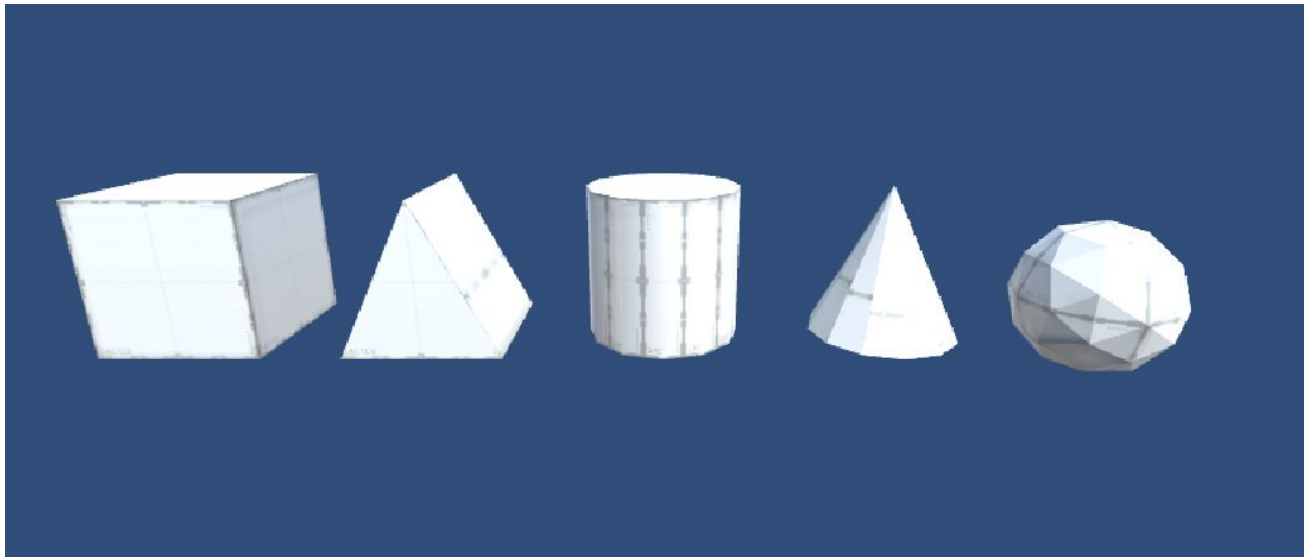
Generatives Modellieren, Bool’sche Operationen



(Quelle: Nischwitz, A: Computergrafik und Bildverarbeitung. S. 76, Springer, 2011)

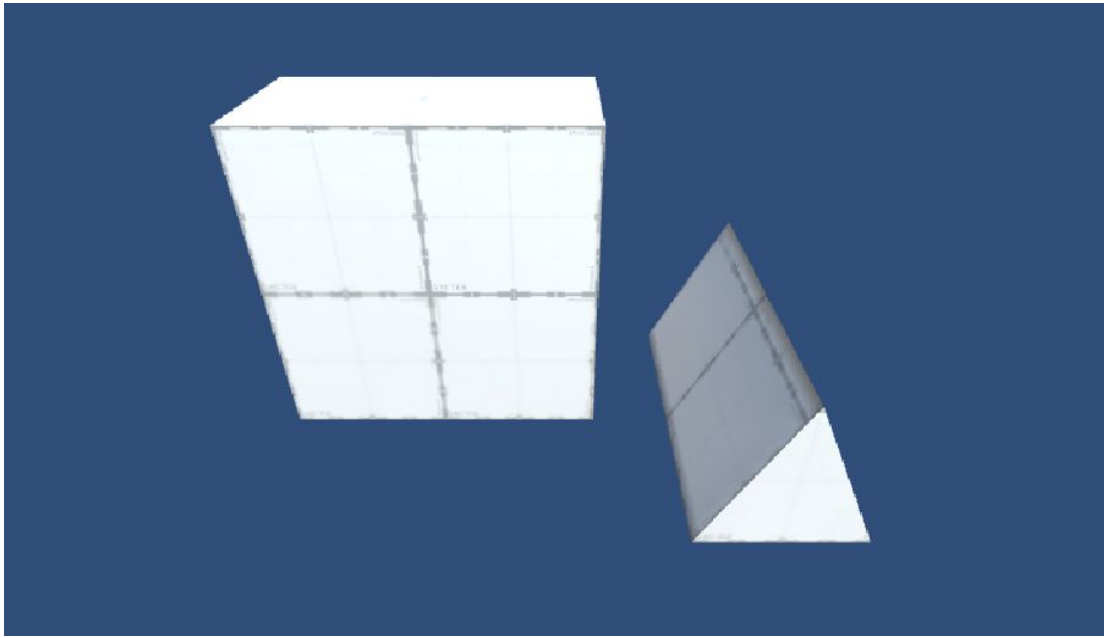
Modellieren

Grundformen in Unity (ProBuilder)

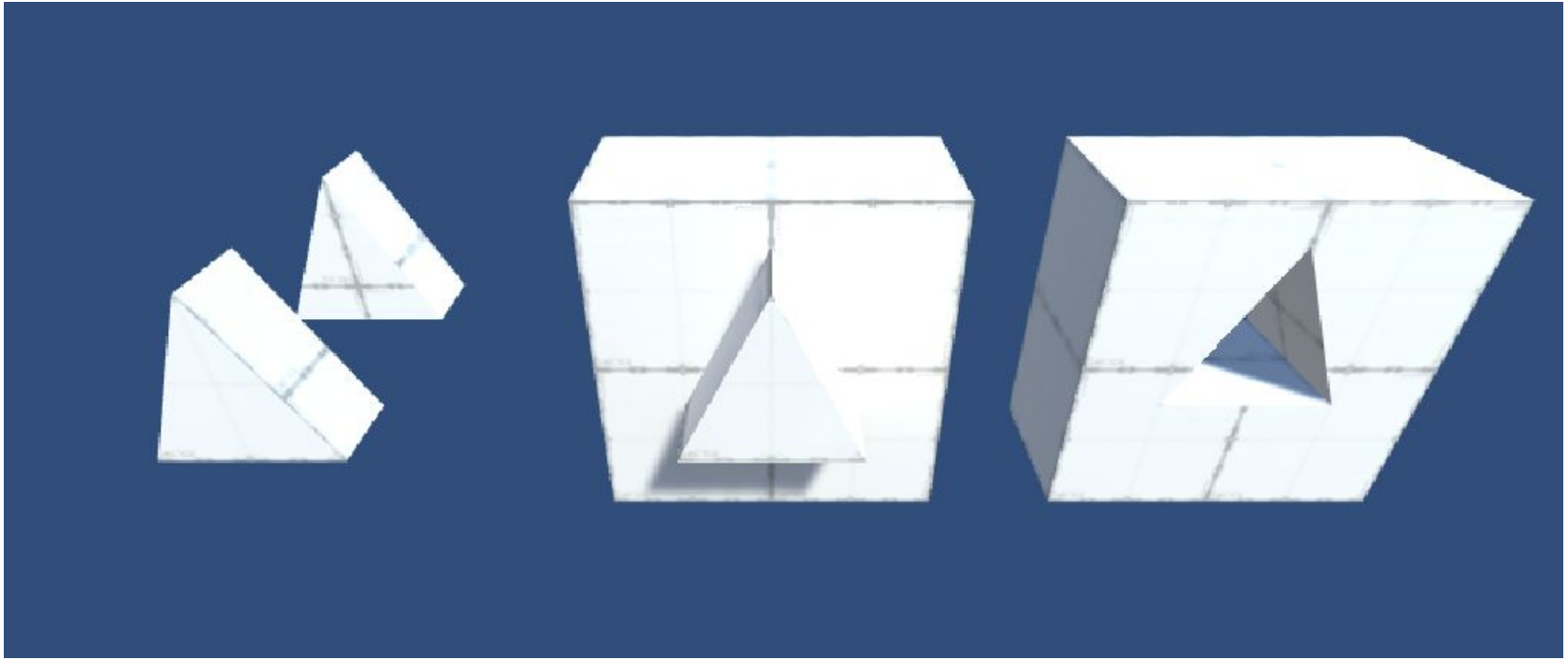


ProBuilder: Sculpting

Face Mode: Shift + Manipulatoren

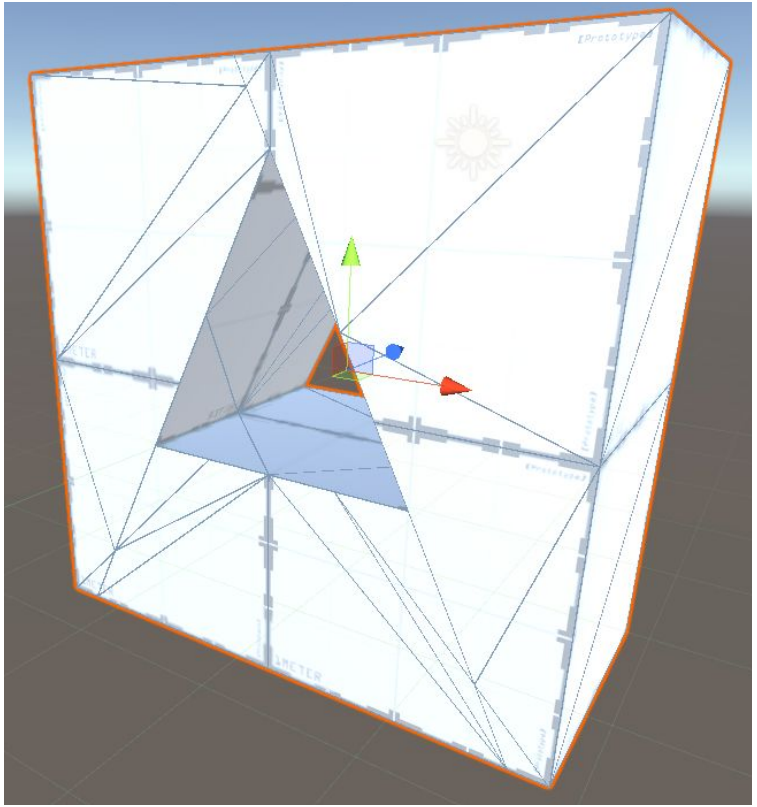


ProBuilder: Boolean CSG Tool

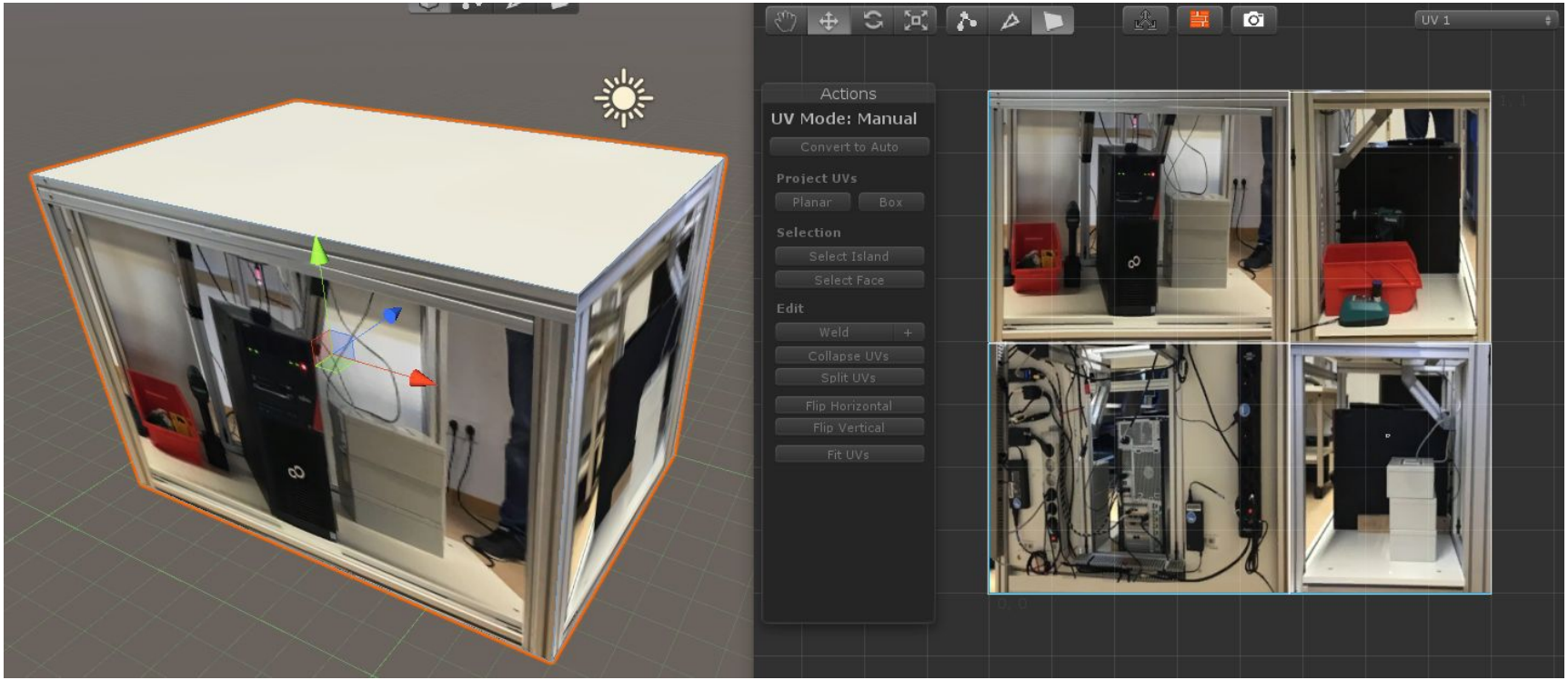


ProBuilder: Boolean CSG Tool

- Erzeugt neues Game Object
- Mesh i.d.R. nicht optimal



ProBuilder: UV Map Tool



Erstellung einer Texturvorlage

- Einfaches Beispiel: Unterbau GePRO Arbeitsplatz
- Aufnahmen von allen Seiten
- Korrektur der Perspektive
- Zusammensetzen zu einem Bild

Nächste Schritte

Wie werden Positionen, Flächen und Farbinformationen zu einer Computergrafik?

- Beleuchtungsmodelle
  - Was ist “Licht” in Unity
  - Cornell Box
  - Grundlagen der Lichtführung (3-Punkt-Beleuchtung)
- Szene → Bild
  - Shader
  - Post Processing