GDC | March 20-24, 2023
San Francisco, CA

Using Vertices over Pixels: Achieving Cartoon Graphics on Standalone VR

Stefan Hell
Lead Programmer

#GDC23

Using Vertices Over Pixels: Achieving Cartoon Graphics on Standalone VR

Using Vertices Over Pixels

- Introduction
- Research
- Implementation
- Conclusion

Introduction

Sweet Surrender

Target Visual Look

- Cartoon Graphics
- Sharp inlines and outline
- Run on mobile hardware

Our Solution

Our Solution

- Stack multiple meshes
- Create gaps that form inlines and outlines
- Offline geometry calculations
  + Runtime vertex shader calculations

Research

Developing for Quest 1

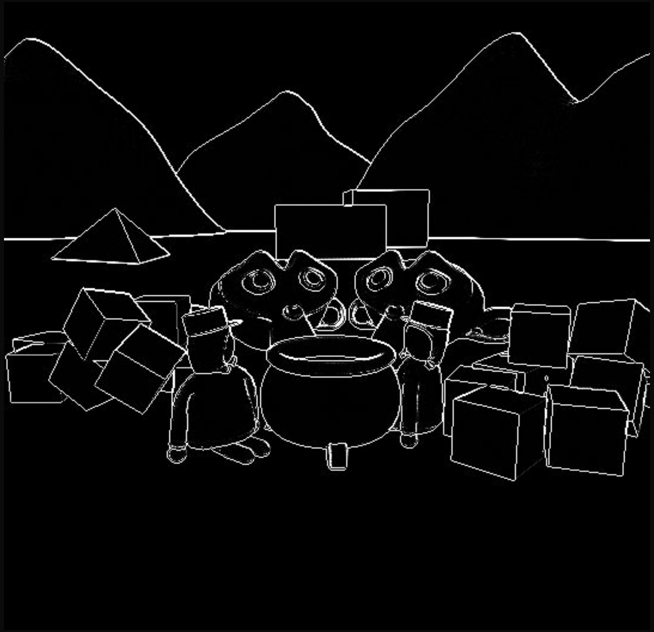
- ~75 draw calls per eye
- ~250k Vertices per eye
- Resolution of 1440 x 1600 = 2,304,000 pixels per eye
- No Post Processing effects
- No Depth Pass

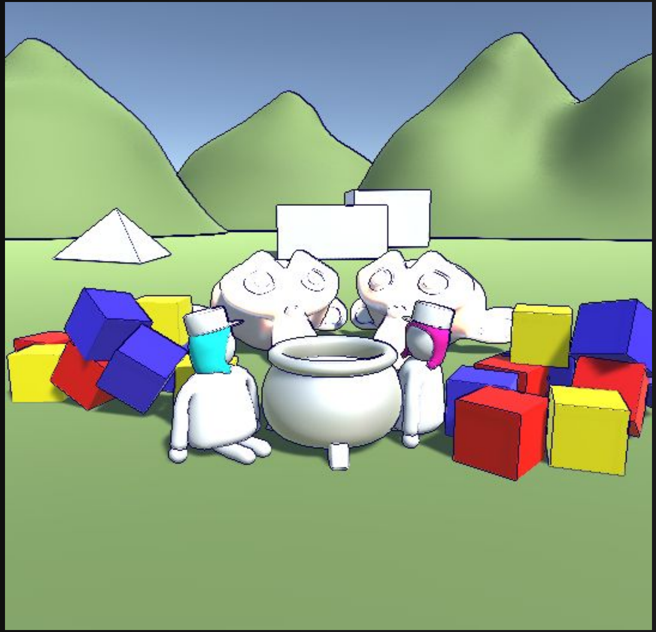
https://developer.oculus.com/blog/pc-rendering-techniques-to-avoid-when-developing-for-mobile-vr/

Existing Solutions

- Post Processing
- High Detail Texture
- Inverted Hull

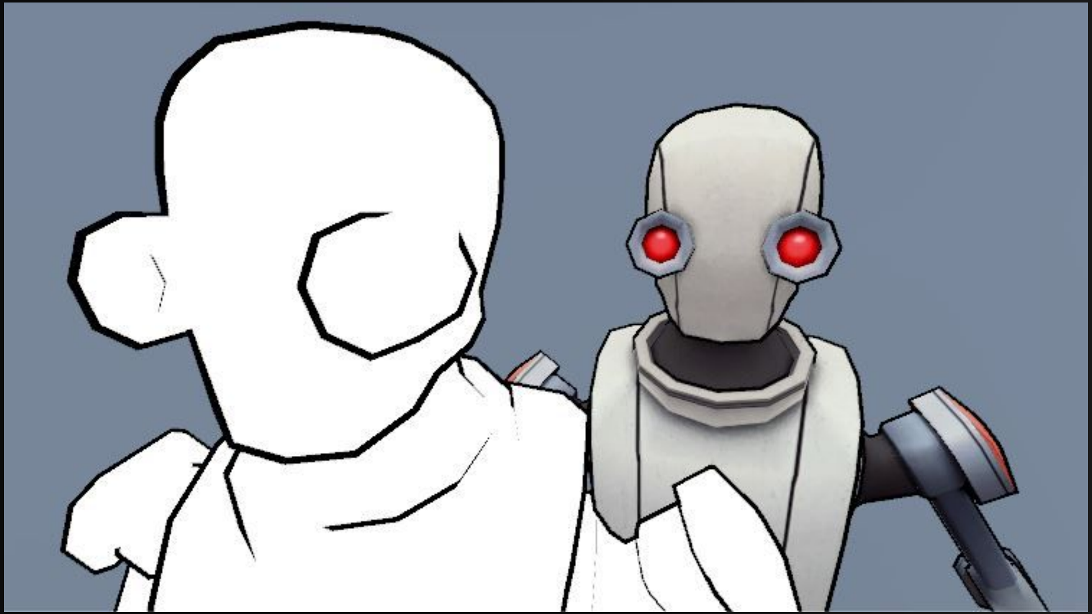
Post-Processing Outlines





High-Detail Textures

Inverted Hull



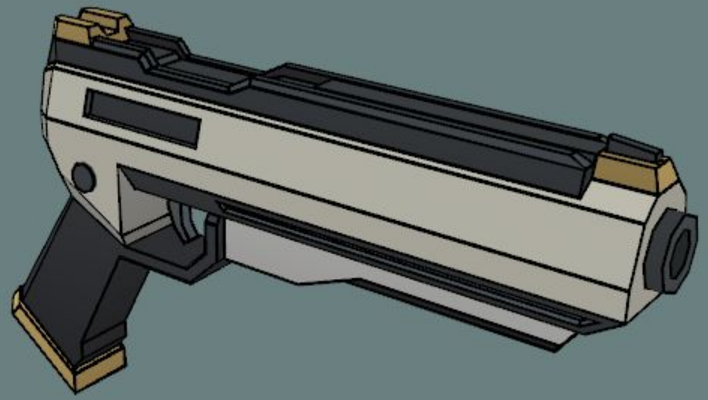
Implementation

Implementation

- Offline Mesh Baking Tool
- Vertex Shader
- Pixel Shader

The pipeline

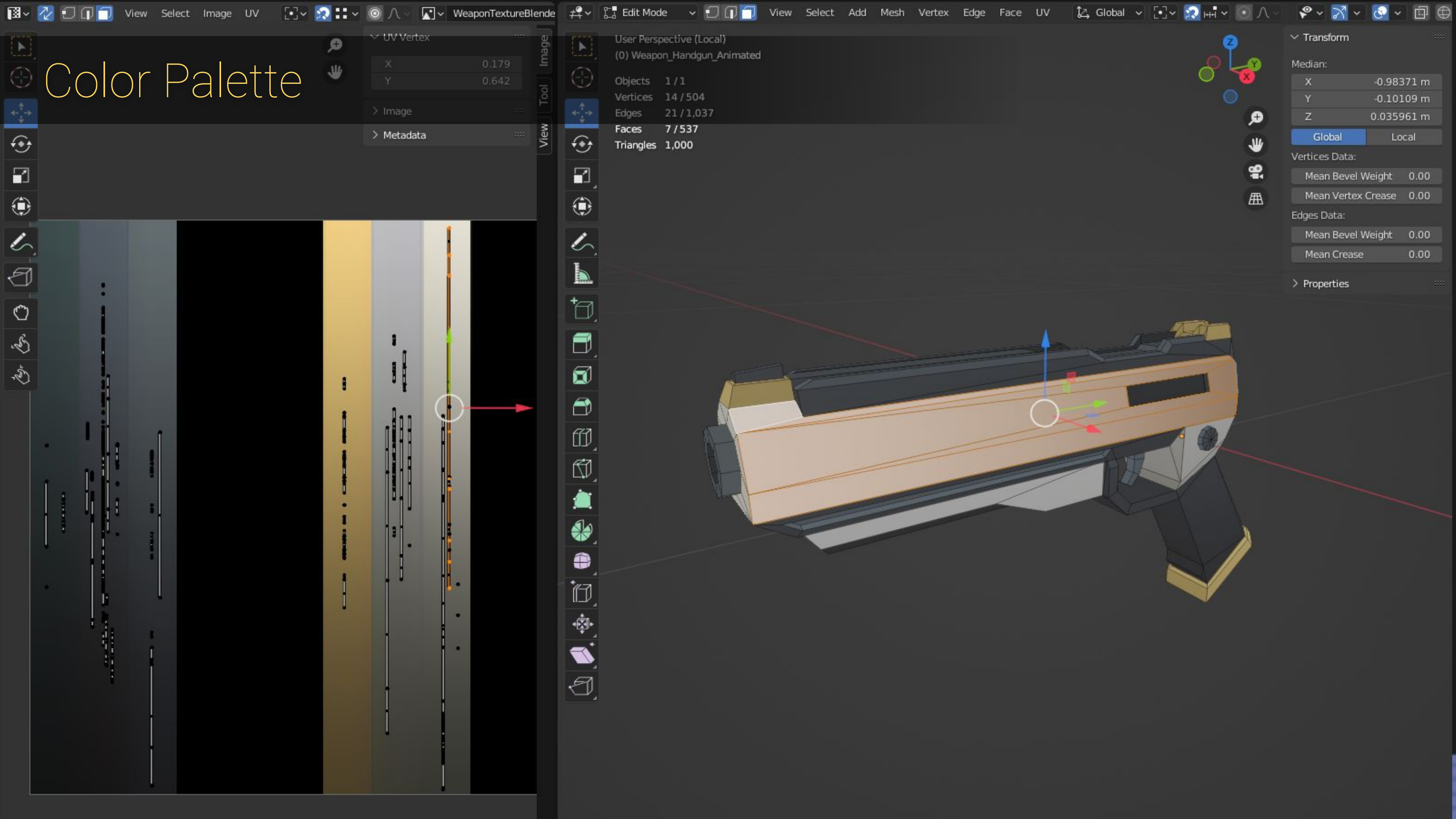
Mesh Preparation - Step by Step



Mesh Structure

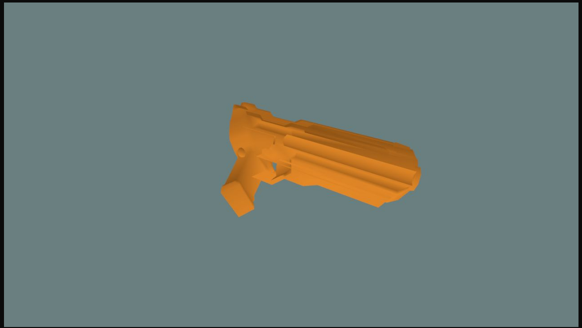
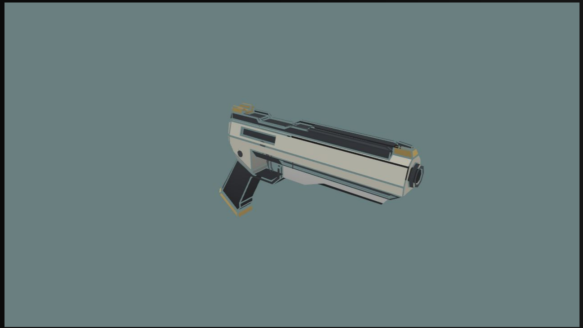
- Vertex List
  - Position (Vector3)
  - Normal (Vector3)
  - UV (Vector2)
  - **Tangent (Vector4)**
- Triangle List

Color Palette

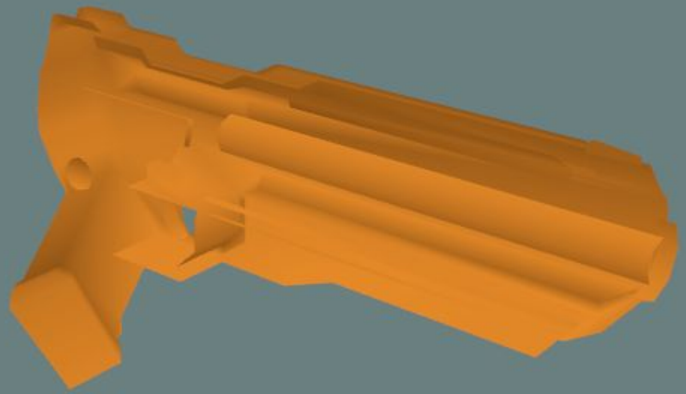


Offline Mesh Baking Tool

- Outline Mesh
- Face Mesh
- Inline Mesh

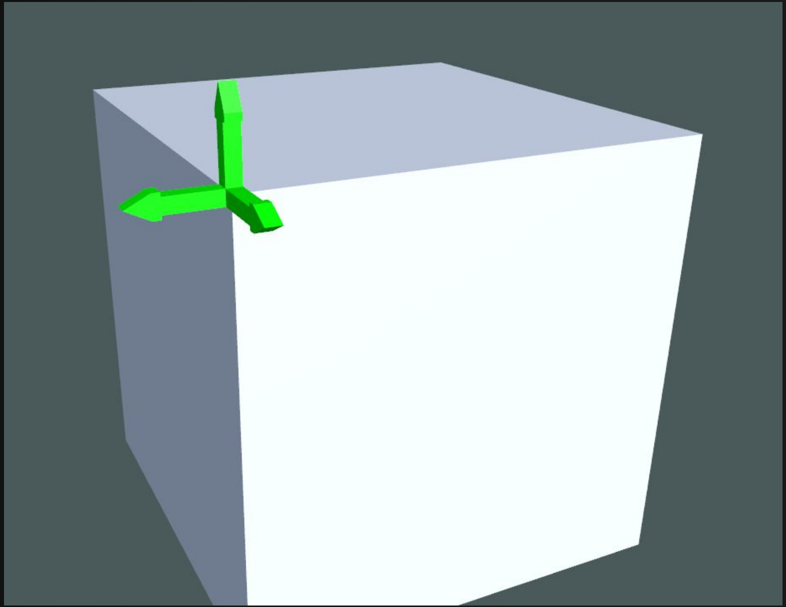




Outline Mesh



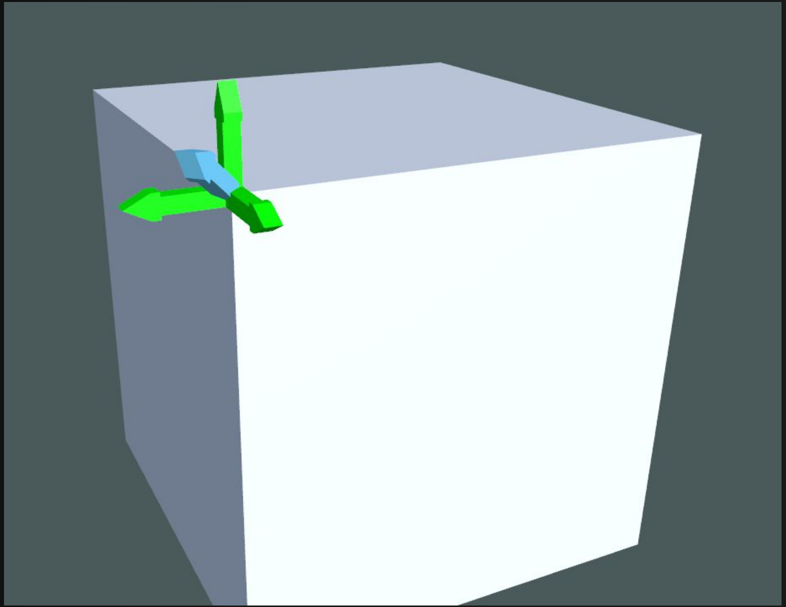
Outline Mesh

- Group all normal vectors by position



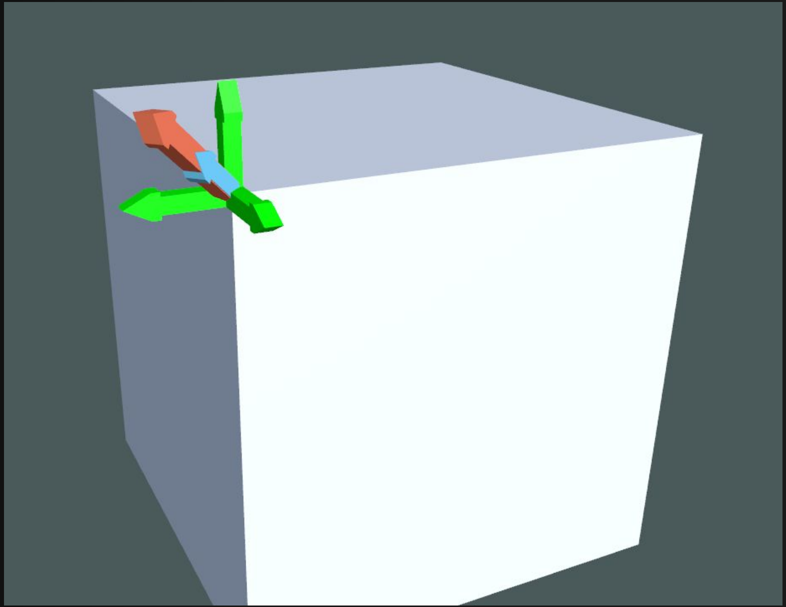
Outline Mesh

- Group all normal vectors by position
- Combine them into one normal



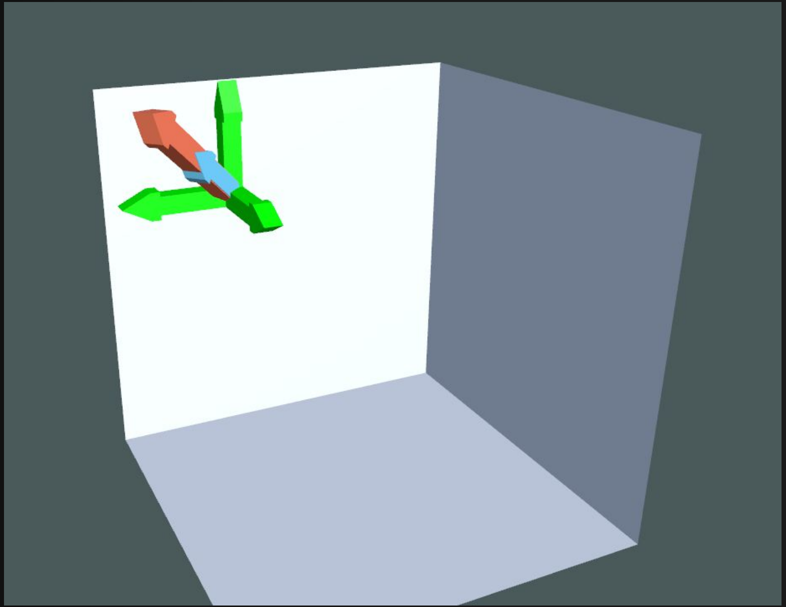
Outline Mesh

- Group all normals by position
- Combine them into one normal
- Fit the length of the combined normal to the length of the green normals



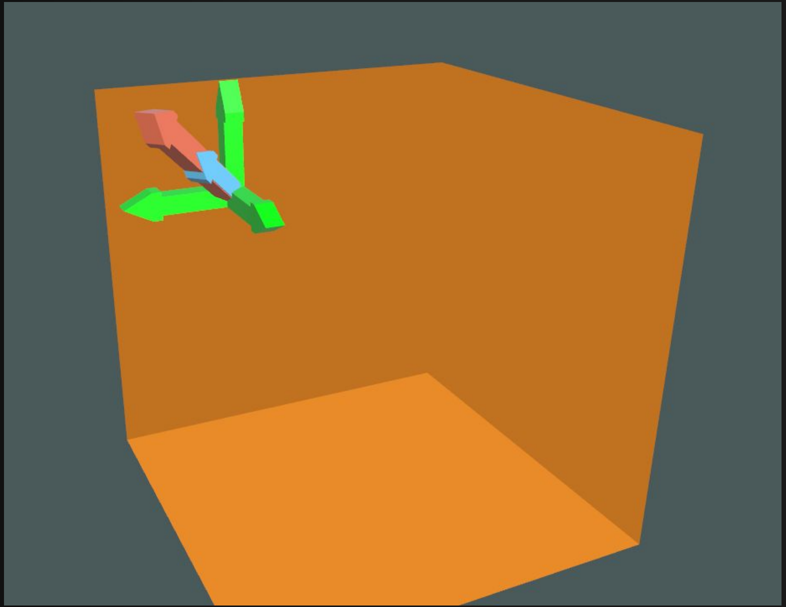
Outline Mesh

- Group all normals by position
- Combine them into one normal
- Fit the length of the combined normal to the length of the green normals
- Flip triangle faces by changing the triangle order, e.g. (0,1,2) -> (0,2,1)



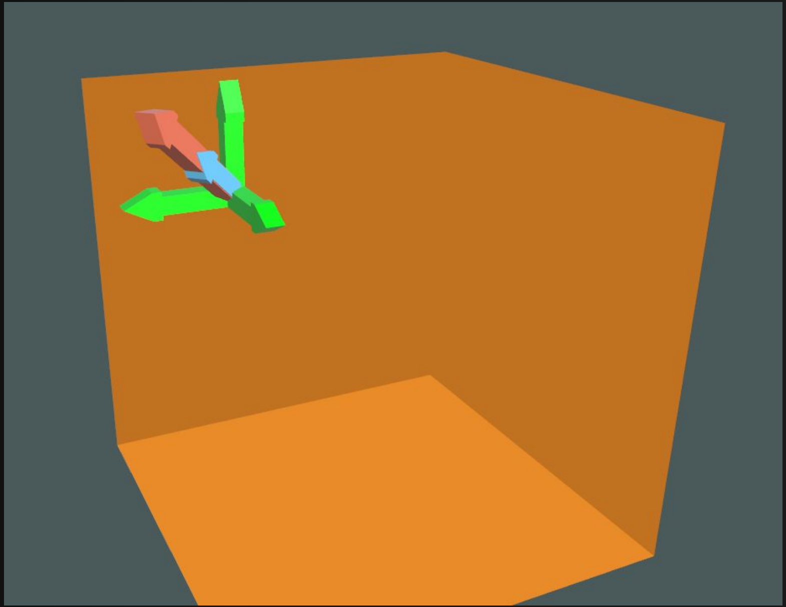
Outline Mesh

- Group all normals by position
- Combine them into one normal
- Fit the length of the combined normal to the length of the green normals
- Flip triangle faces by changing the triangle order, e.g. (0,1,2) -> (0,2,1)



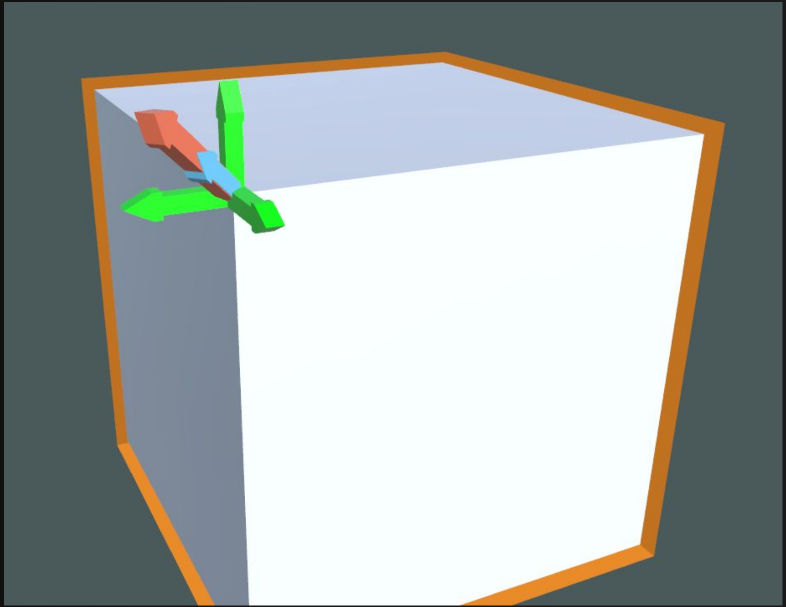
Outline Mesh

- Group all normals by position
- Combine them into one normal
- Fit the length of the combined normal to the length of the green normals
- Flip triangle faces by changing the triangle order, e.g. (0,1,2) -> (0,2,1)

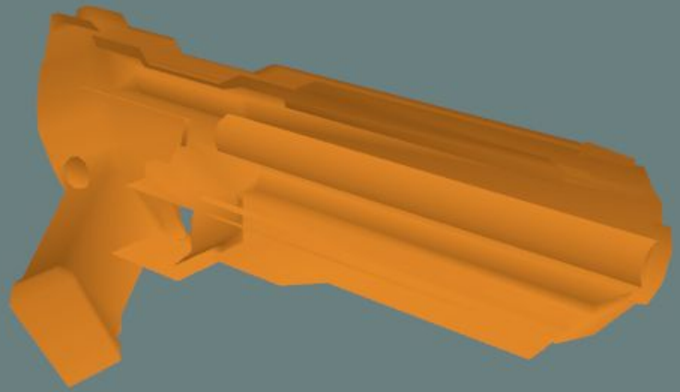


Outline Mesh

- Group all normals by position
- Combine them into one normal
- Fit the length of the combined normal to the length of the green normals
- Flip triangle faces by changing the triangle order, e.g. (0,1,2) -> (0,2,1)



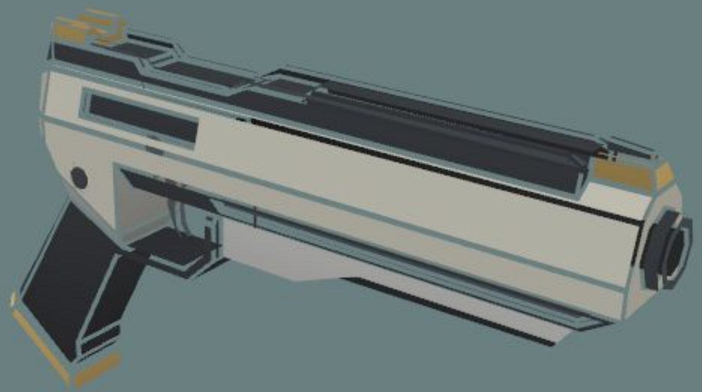
Outline Mesh



Outline Mesh + Default Mesh

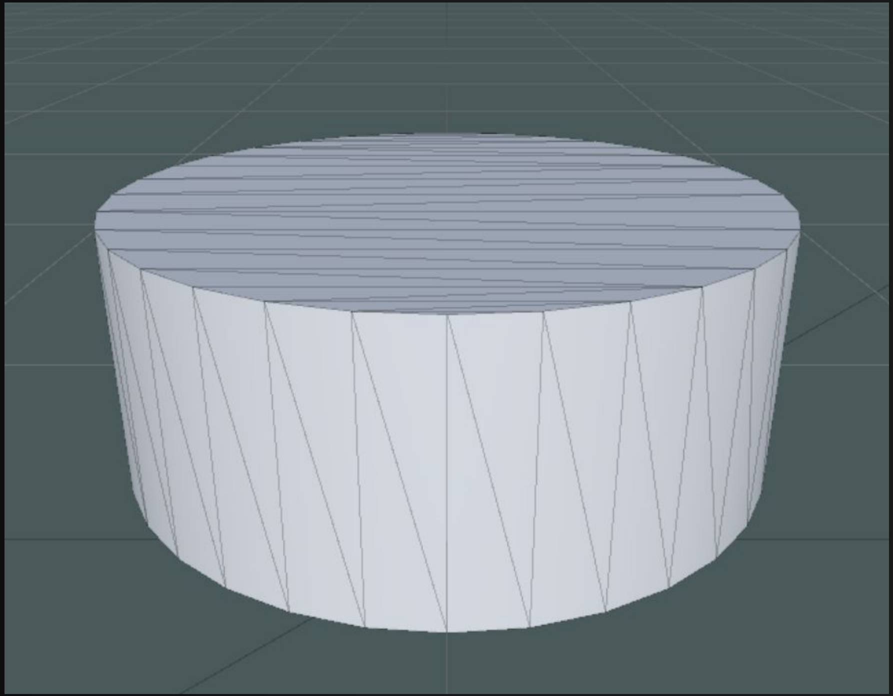
Face Mesh

Face Mesh



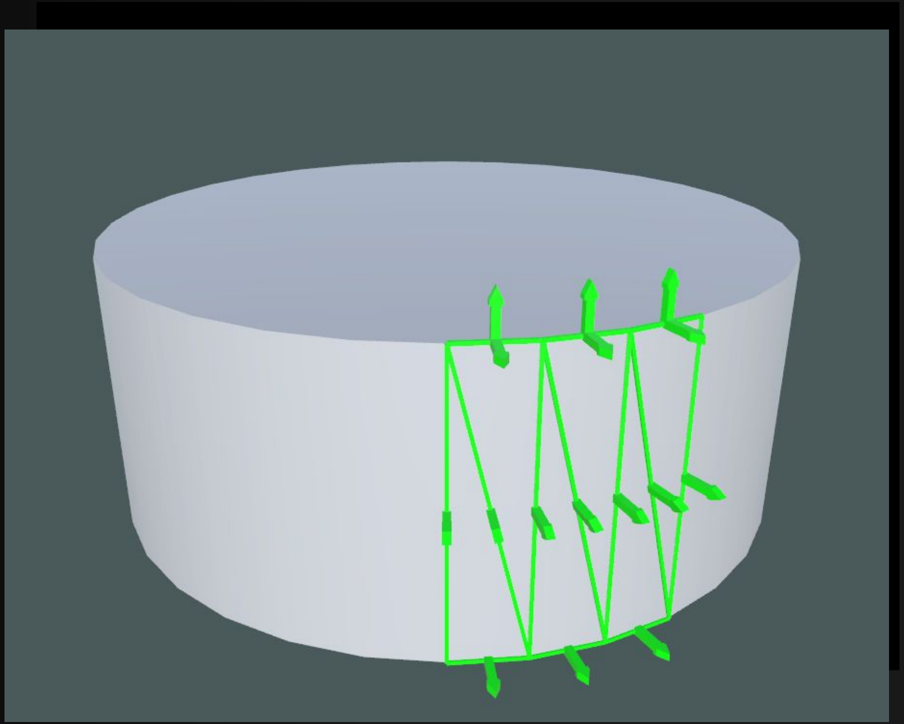
Step 1: Detect All Edges

- Iterate over all triangles



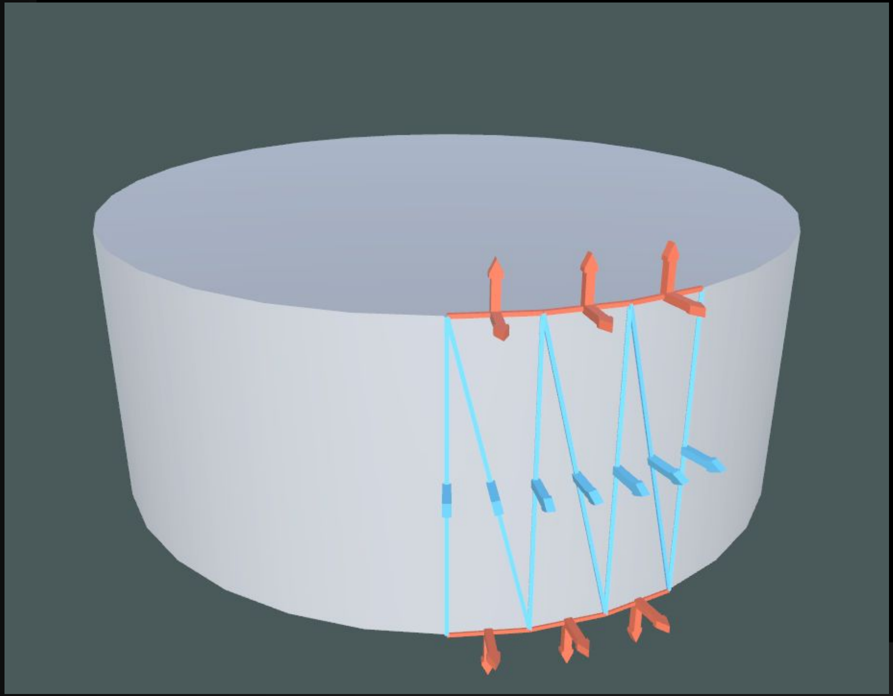
Step 1: Detect All Edges

- Iterate over all triangles
- Create a list of all lines (storing position A, B and normal A)



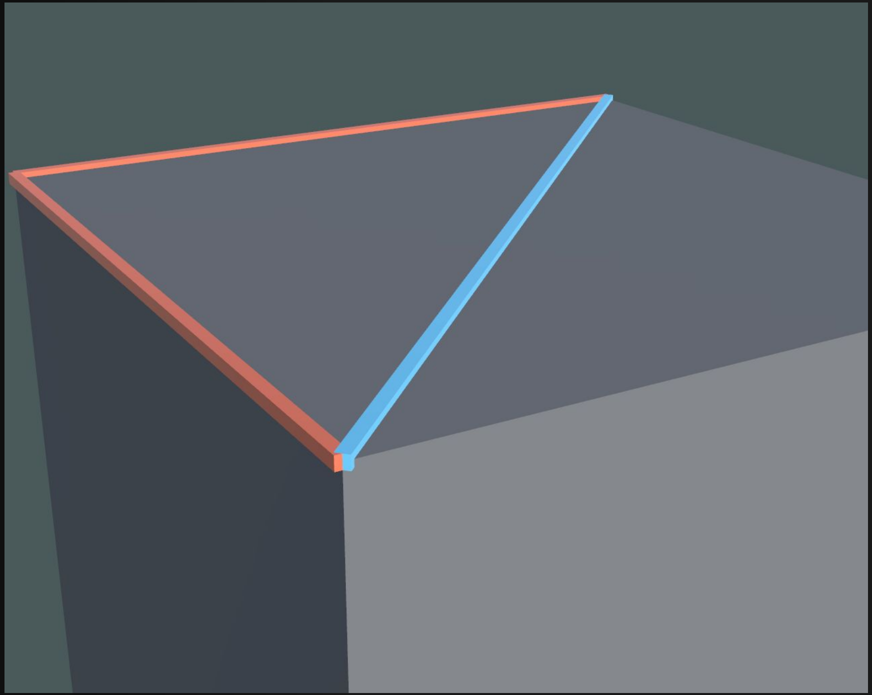
Step 1: Detect All Edges

- Iterate over all triangles
- Create a list of all lines (storing position A, B and normal A)
- Lines sharing position and normal with another line have smooth edges
- All other lines are hard edges



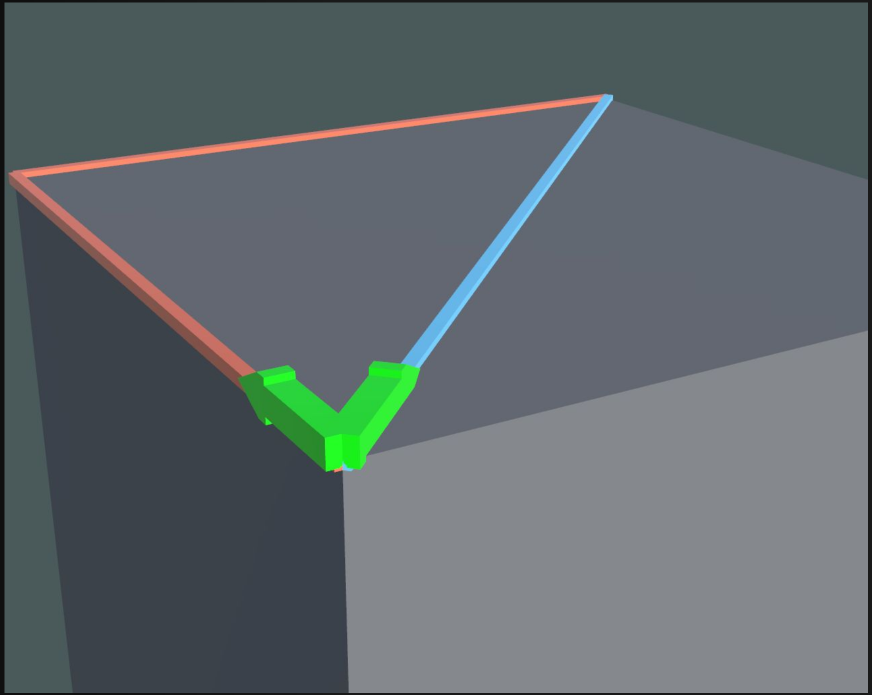
Step 2: Define Inward Vector

❖ Iterate over all hard edges (△ABC)



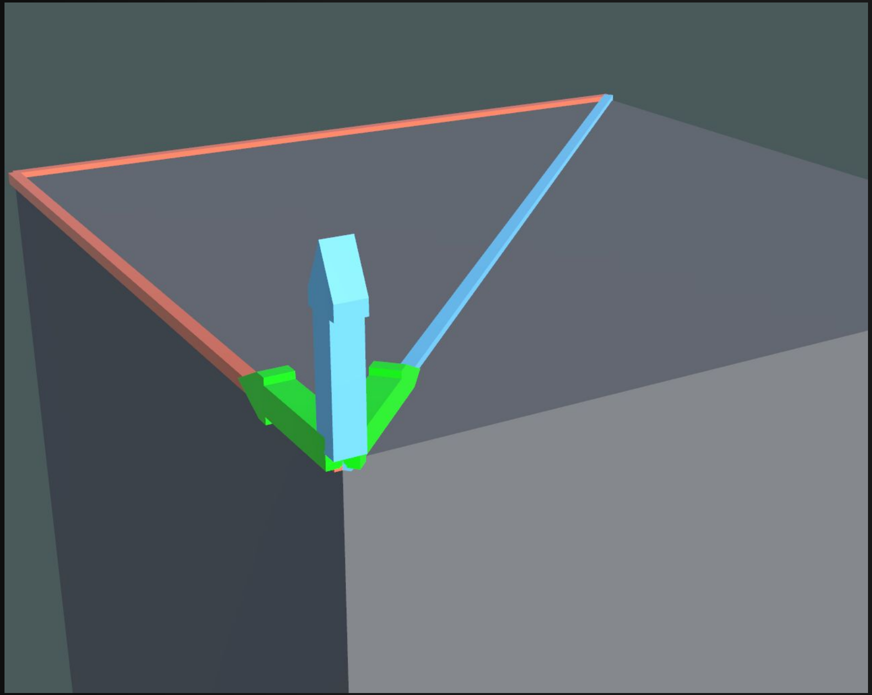
Step 2: Define Inward Vector

- Iterate over all hard edges (△ABC)
- Form vectors AB, AC



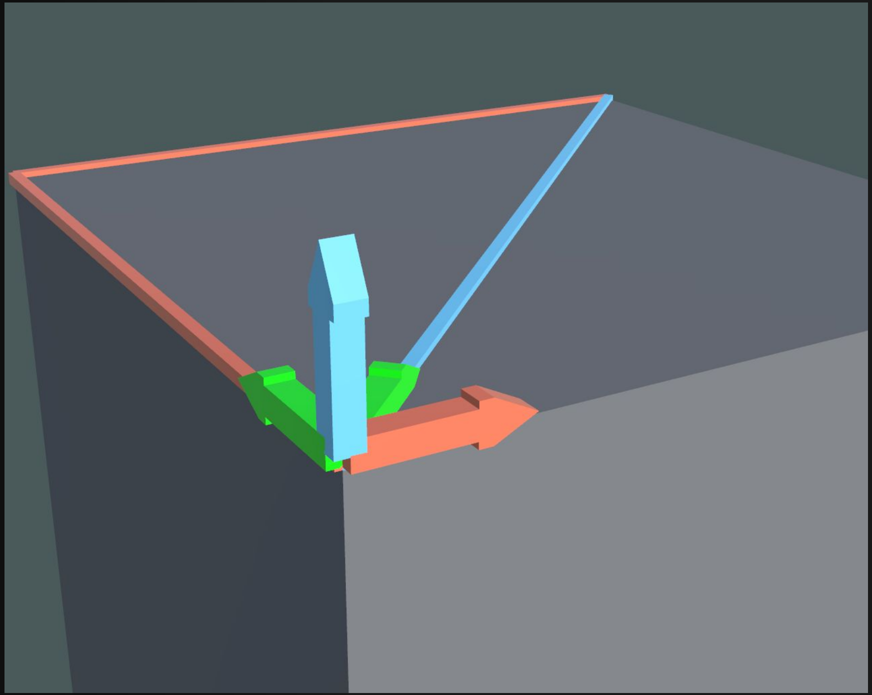
Step 2: Define Inward Vector

- Iterate over all hard edges (△ABC)
- Form vectors AB, AC
- Take cross-product (AB)x(AC)



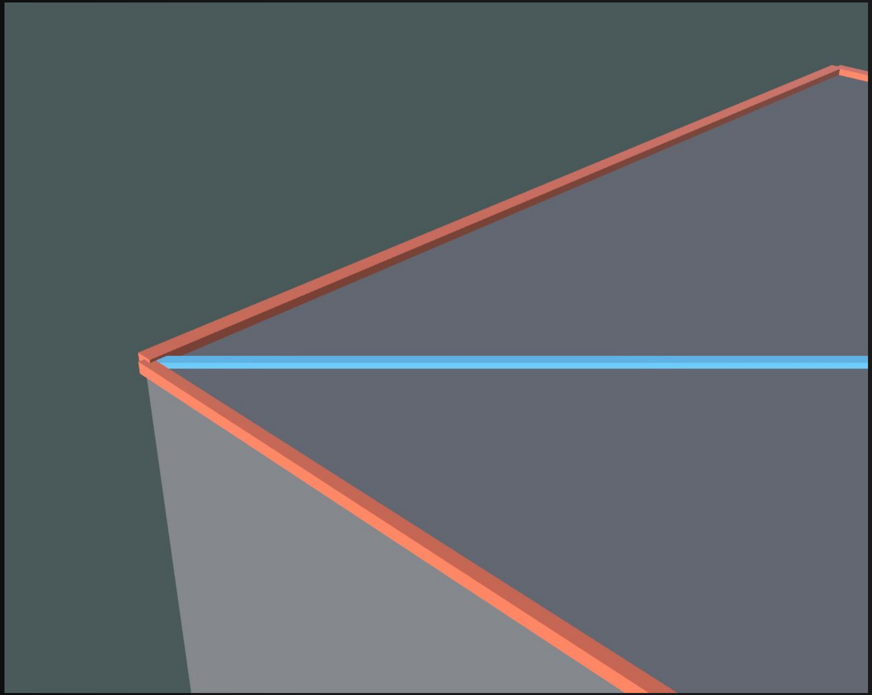
Step 2: Define Inward Vector

- Iterate over all hard edges (△ABC)
- Form vectors AB, AC
- Take cross-product (AB)x(AC) = AD’
- Take cross-product (AD’)x(AB)



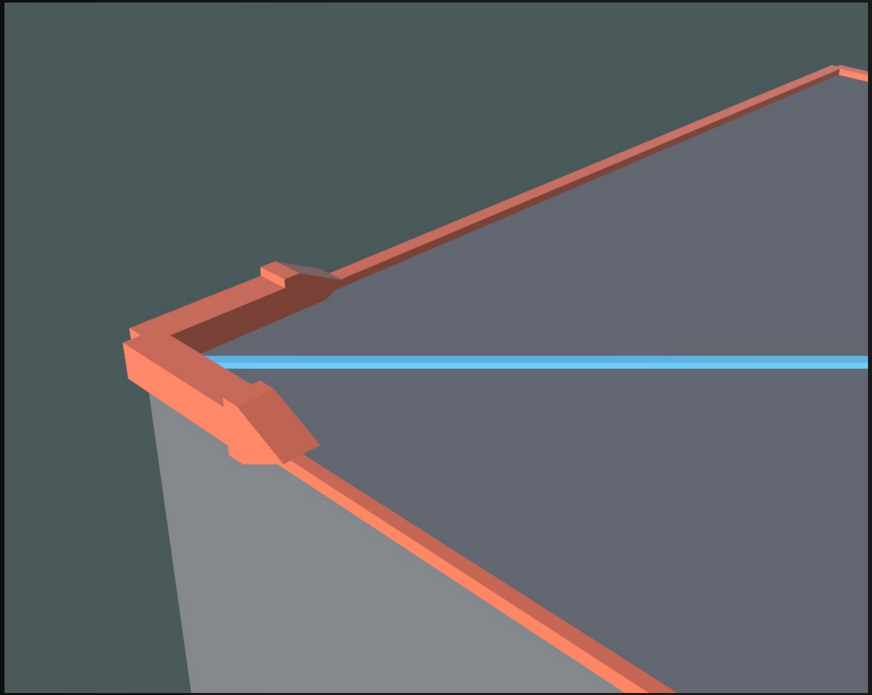
Step 3: Combine Inward Vectors

- Iterate over all vertices



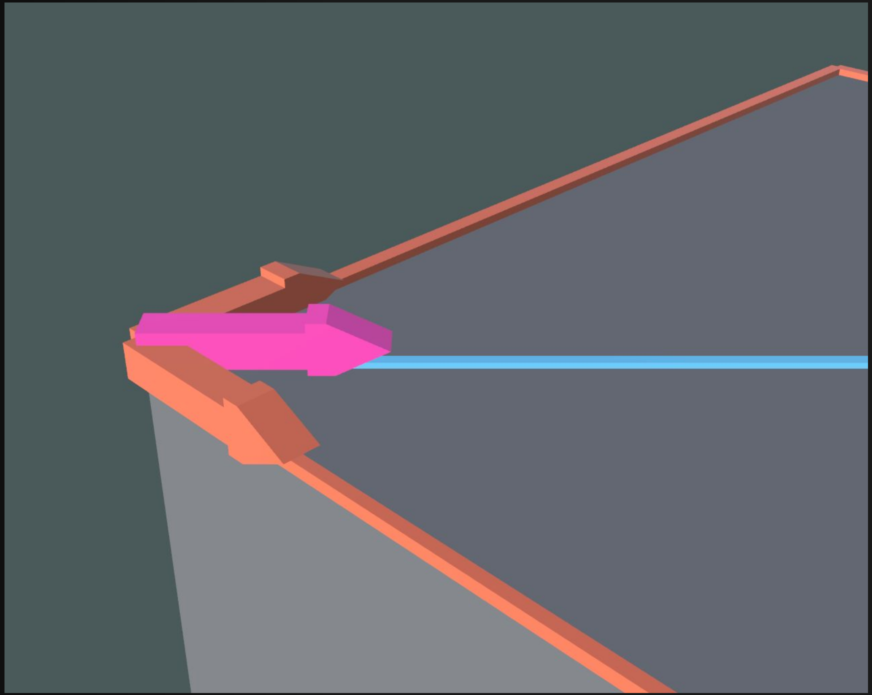
Step 3: Combine Inward Vectors

- ❖ Iterate over all vertices
- ❖ Select all inwards vectors



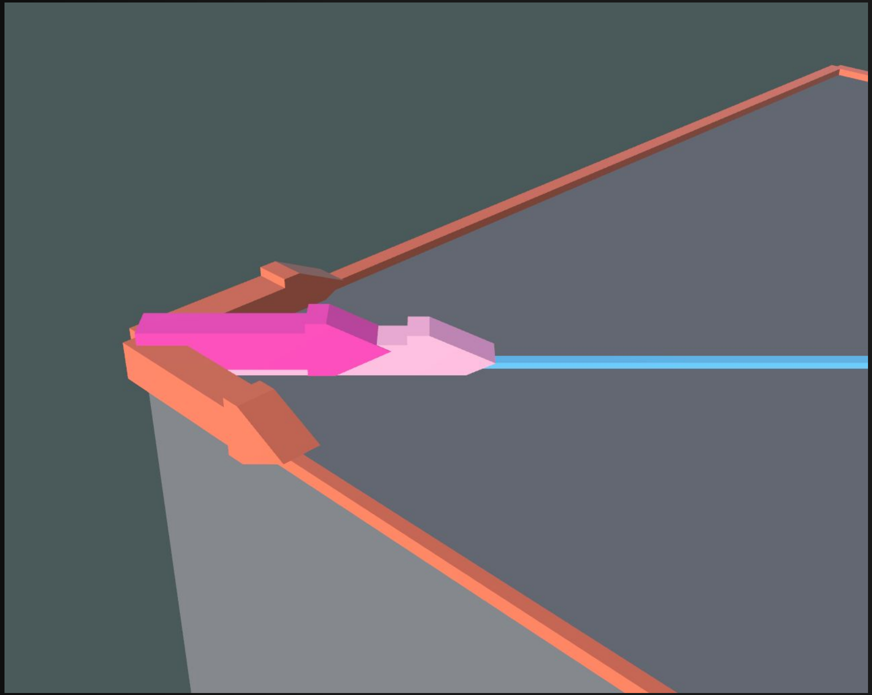
Step 3: Combine Inward Vectors

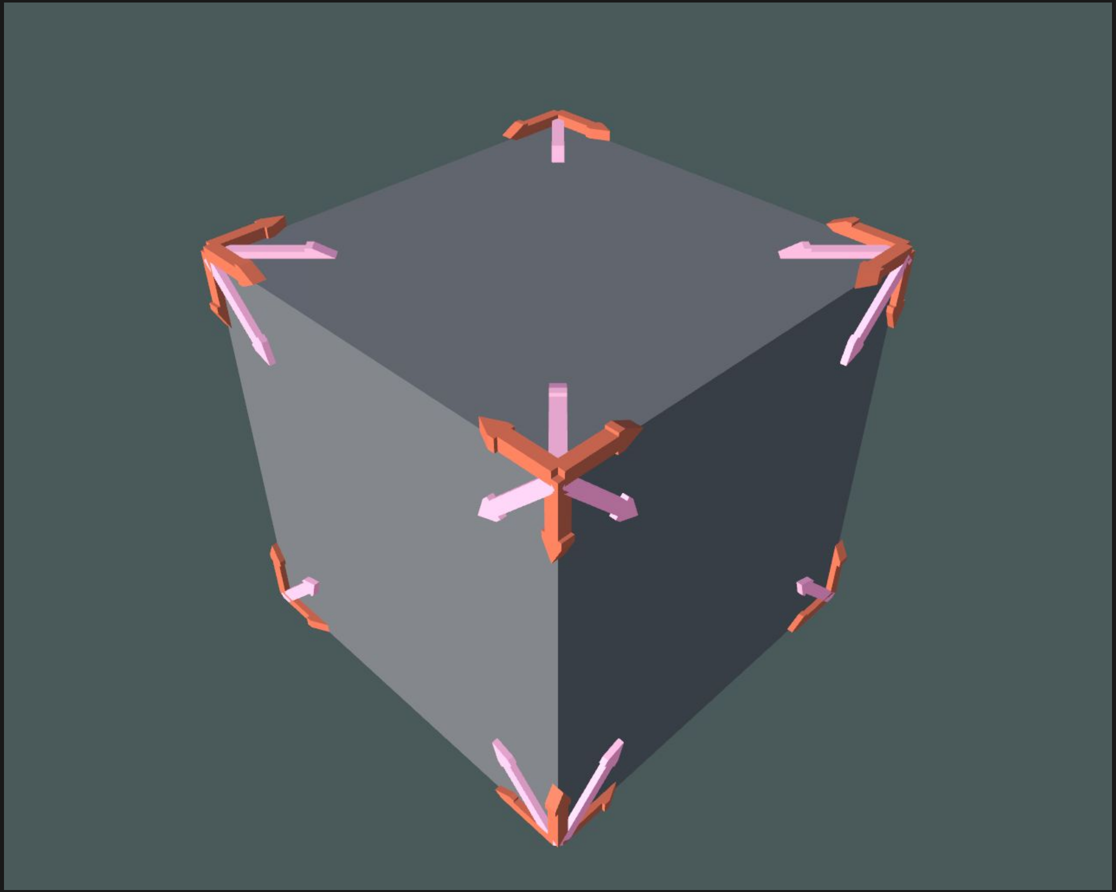
- Iterate over all vertices
- Select all inwards vectors
- Combine vectors

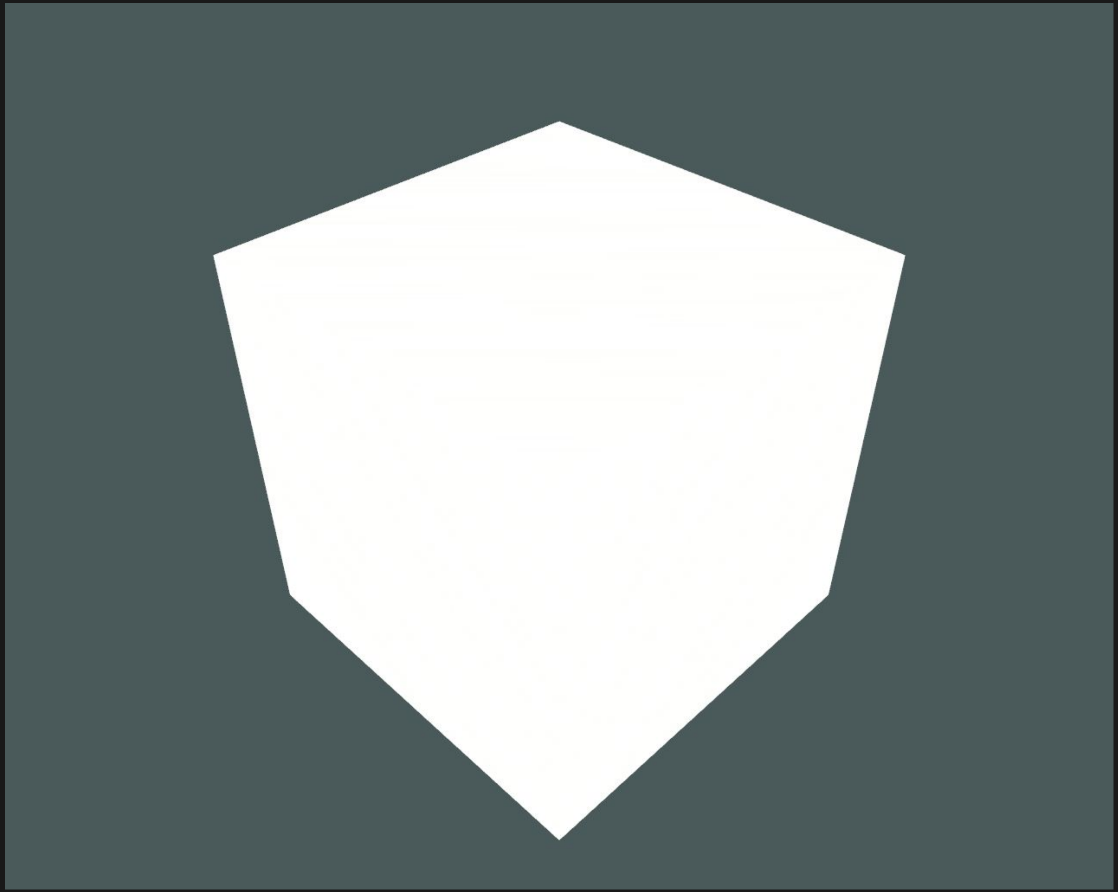


Step 3: Combine Inward Vectors

- Iterate over all vertices
- Select all inwards vectors
- Combine vectors
- Fit length of combined normal to length of the inwards vectors







Face Mesh

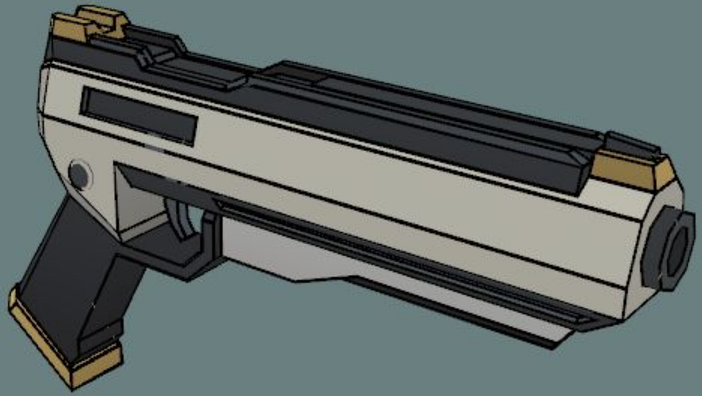
Face Mesh + Outline Mesh

Face Mesh + Outline Mesh

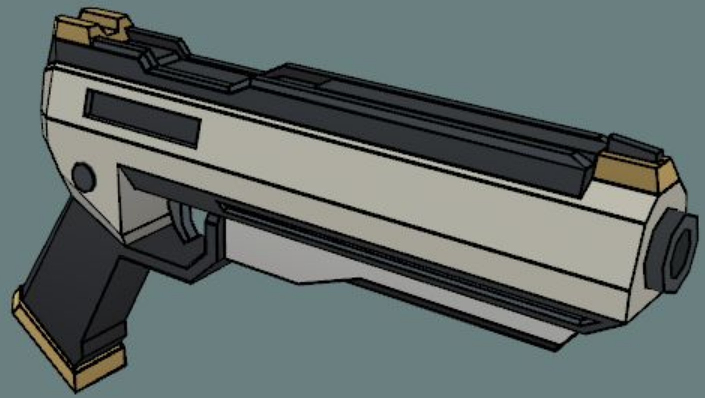
Inline Mesh

- Similar to Outline Mesh but
  - Displacement vector is **inverted**
  - Triangles are **not inverted**

Face + Outline Mesh

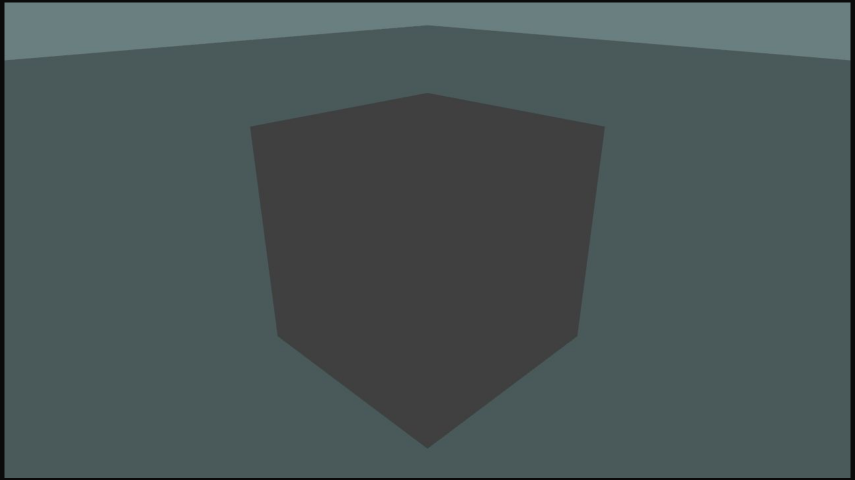


Face + Outline + Inline Mesh



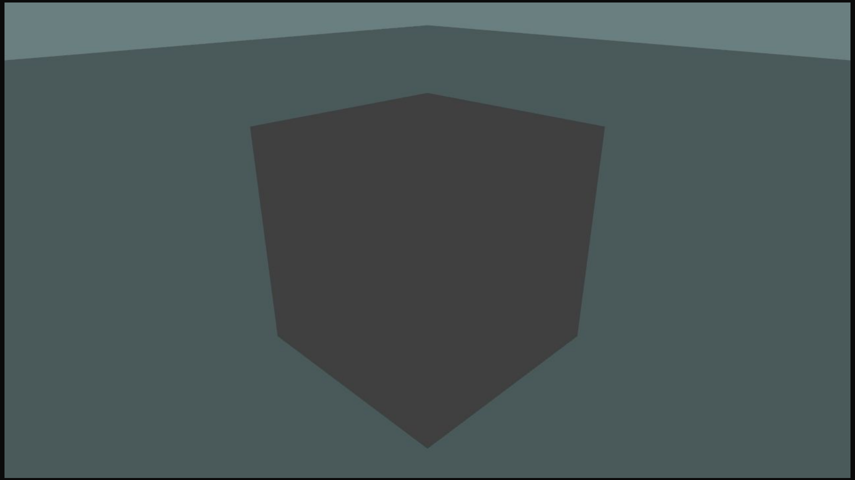
Insetting the hull

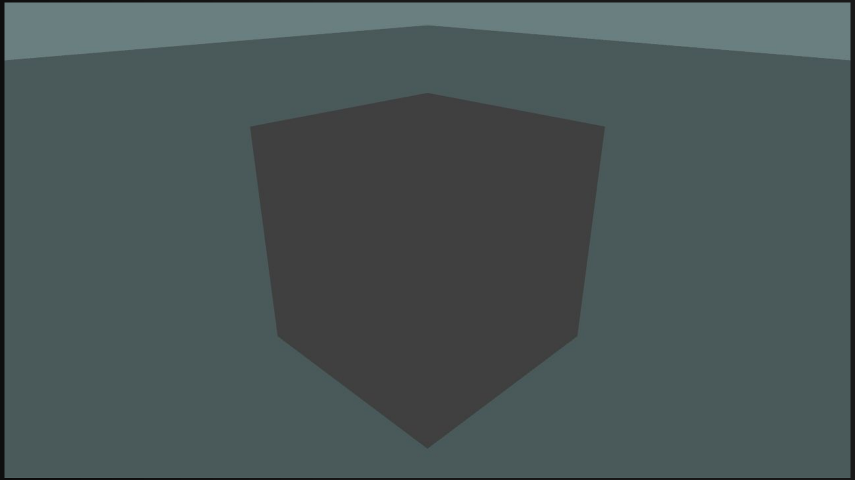
- Outline mesh can clip into floor
- Inset the color mesh and inline mesh instead



Insetting the hull

- Go over all outline vertices
  - Subtract the displacement vectors from all vertices at the same position

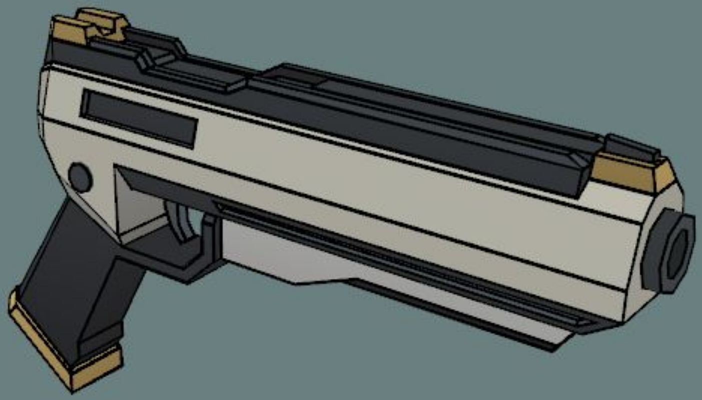




Combine Meshes

- 1 Draw call instead of 3
- Vertex count
  - Original 962

  - Color 962
  - Outline + 326
  - Inline + 326
  - Final = 1614



Recap

Recap - Face Mesh

Recap - Face + Outline Mesh

Recap - Face + Outline + Inline Mesh

Recap - Face + Outline + Inline Mesh (Inset)

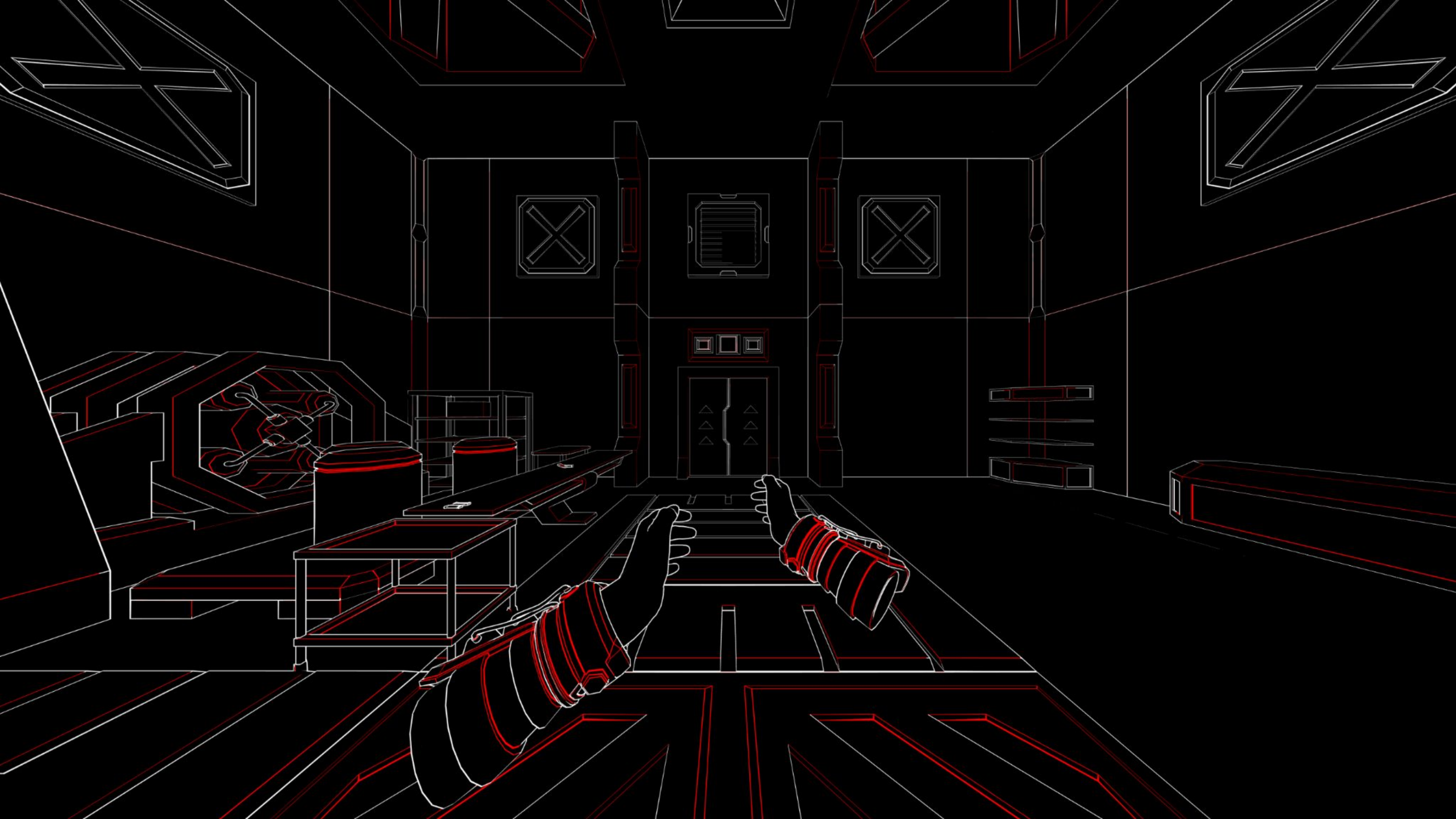
Vertex Shader

Vertex Shader

- Executed once for each vertex
- Calculates the line width
- Moves position along displacement vector
- Applies colors to face mesh and edge meshes

Color Mesh, Inline Mesh or Outline Mesh?

```
half alpha = IN.tangent.w;
half edgeVal = 1 - step(alpha, 0.6); // is 1 if it is an outline or inline
half outlineVal = 1 - step(alpha, 0.9); // is 1 if it is an outline
half inlineVal = edgeVal - outlineVal; // is 1 if it is an inline
```



Vertex Shader - Calculating Color

```
half4 baseColor = SAMPLE_TEXTURE2D_LOD(_BaseMap, sampler_BaseMap, IN.uv, 0);

baseColor = lerp(baseColor, _OutlineColor, edgeVal);
```

Object -> World Space

```
float3 worldPosition = TransformObjectToWorld(IN.pos);

float3 displacementNormal =  TransformObjectToWorldNormalScaled(IN.tangent.xyz);
```

Displacing the vertex

```
half lineWidth = min(distanceToCamera * _LineWidth, maxLineWidth);

worldPosition = worldPosition + lineWidth * displacementNormal;


// translate the world position (after being changed in world space) to clip space

OUT.positionHCS = TransformObjectToHClip(TransformWorldToObject(worldPosition));
```

Z-Fighting

```
float4 zFightingOffset = inlineVal * float4(outlineScreenOffset, 0) * distanceToCamera;
zFightingOffset -= outlineVal * _OutlineOnlyZFightingOffset


OUT.positionHCS += zFightingOffset;
```

Pixel Shader

```
half4 frag(Varyings IN) : SV_Target
{
    return IN.color;
}
```

Conclusion

Advantages

- Sharp lines at any distance from camera
- Leverages MSAA Antialiasing
- Supports batching
- Low artistic effort
- Modify effects on a per-object basis

Disadvantages

- Increases vertex count ~65 %
- Cannot have outlines on intersecting model parts
- Potential issues with very thin geometry parts

Takeaways

- ❖ Timeless visual look within the limits of the hardware
- ❖ No Post Processing
- ❖ Realizable with limited art budget

Questions

Thank you for your attention