Parallel computation of locally optimal triangulations on GPU

Michal Červeňanský, Zsolt Tóth and Juraj Starinský

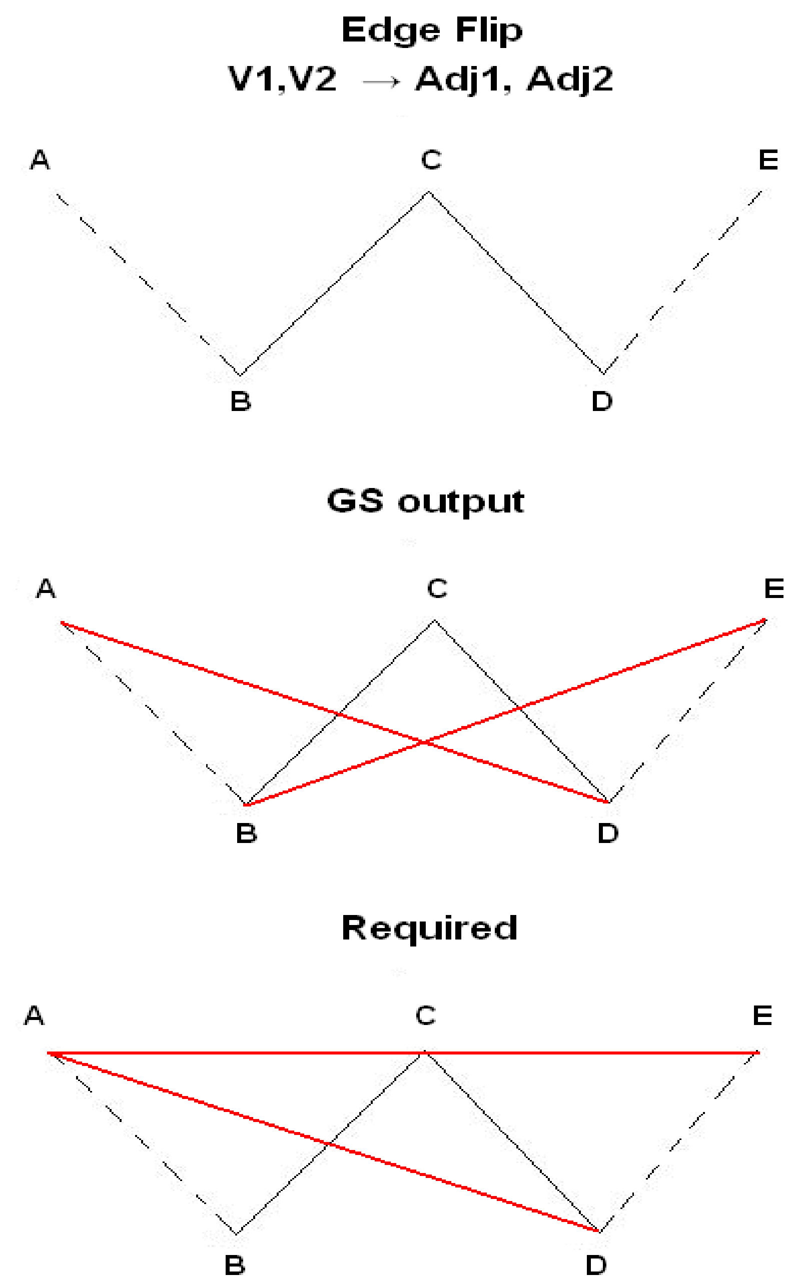
Overview

- Geometry shader
- Design
- Implementation

Geometry Shader

- New output generation
- No adjacency output
- 128 vertices (drivers)

- No information about edge flipping



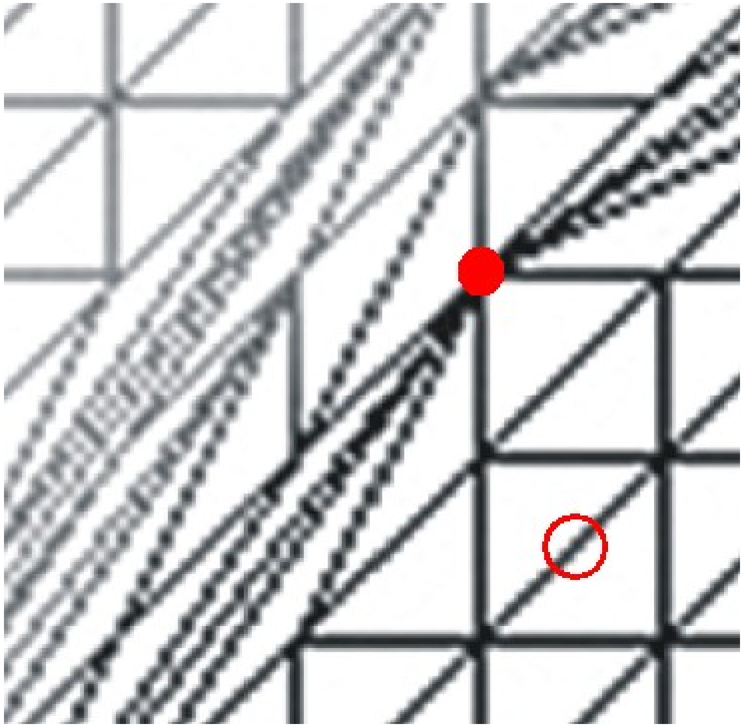
Overview - design

- Geometry shader
- Design
  - Main idea
  - Data structures
  - Parallel edge flipping
- Implementation

Main idea

- Fragment – vertex
  - 2 to N edges
  - Add/remove edges

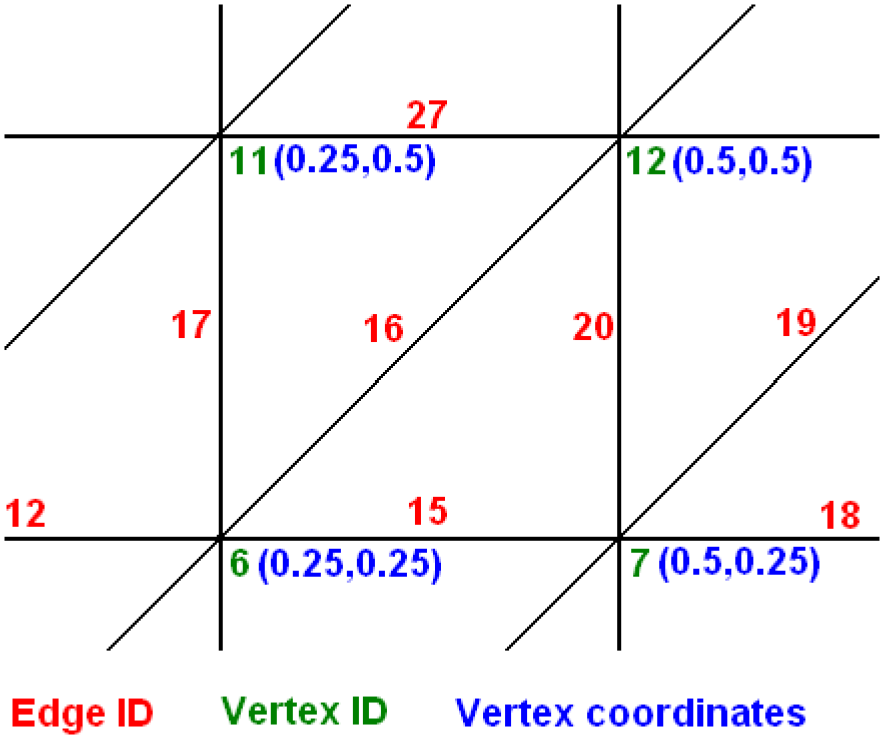
- Fragment – edge
  - 2 vertices
  - Changed coordinates



Data structures

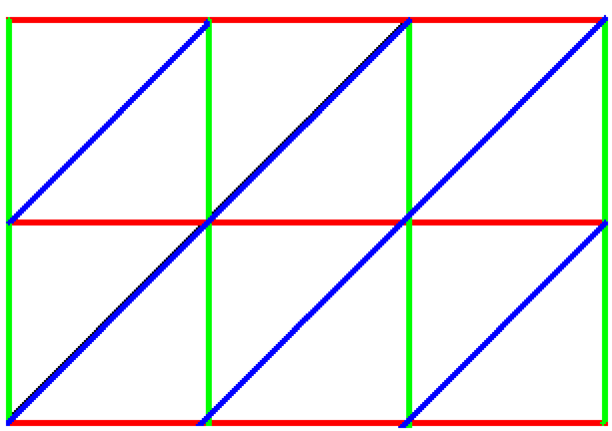
- Edge ID
- Vertices ID, Adjacency ID
- Vertices real coordinates
- Edge neighbours ID

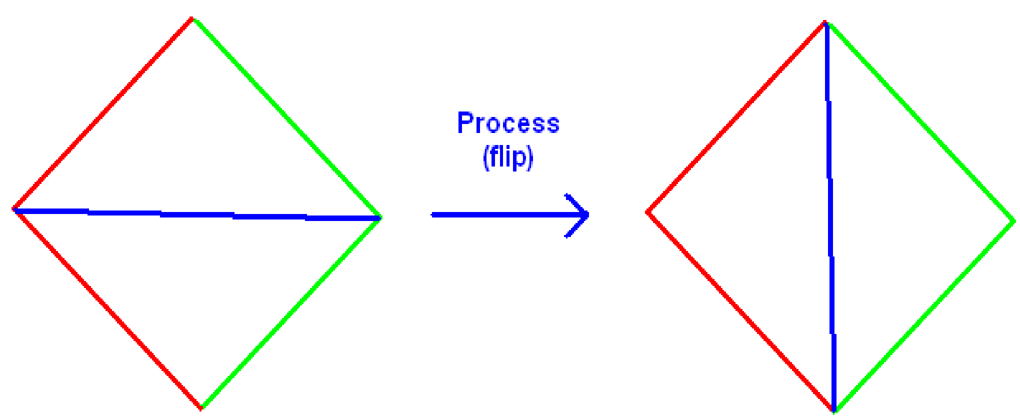
- Preserve orientation



Parallel edge flipping

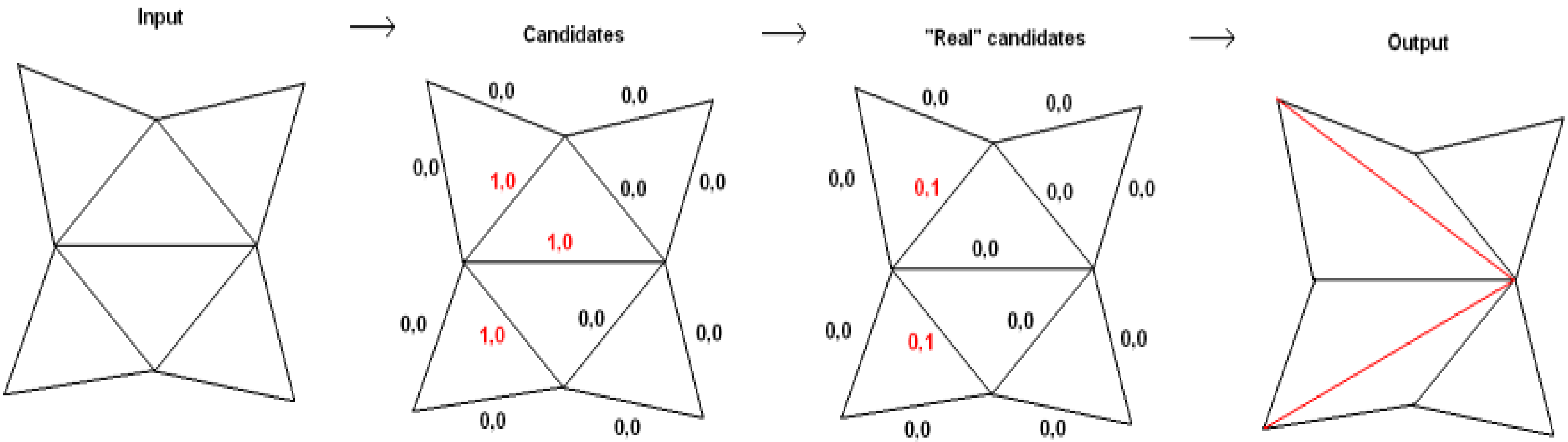
- Fragment shaders
  - Isolated (no communication)
  - Non deterministic (fragment position)
- 1 triangle → 0/1 flipped edge → update data
- 3 packages (colors) of edges (3x processing)
- 1 triangle → 2 colors → collision
- Edge repainting → recursion





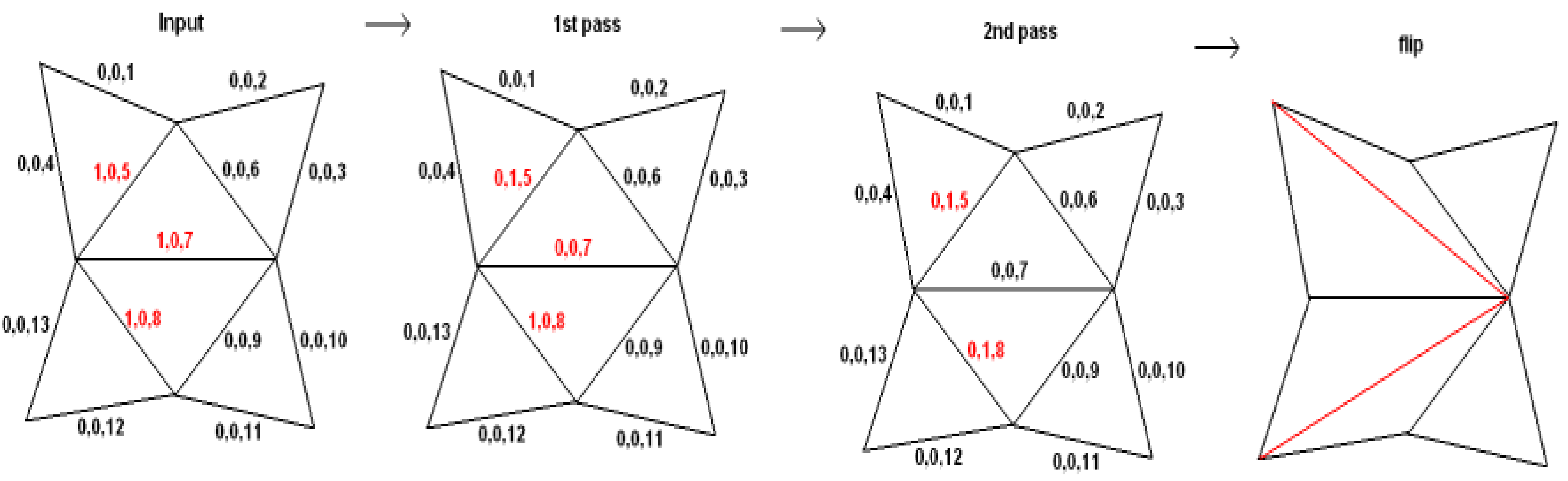
Parallel edge flipping our solution

- Select edges (cost function) → candidates
- Candidates → select „real“ candidates
- 1 triangle → 1 „real“ candidate
- Flip all „real“ candidates parallel
- Iterate again (if necessary)



Parallel edge flipping selection of „real“ candidates

```
while( exists candidates )     //iteration step
{
    do(for each candidate edge)
    {
        obtain ID from neighbourhod edges    //candidate
        if( edge.ID < max(neighbourhoods.ID)
        {
            OUT.color = float4(0, 1, edge.ID, 0);
            OUT.color.neighboor1/2/3/4 = float4(0,0,neigh.ID1/2/3/4,0);
        }
    }
}
```



Overview - implementation

- Geometry Shader
- Design
- Implementation
  - Initial settings
  - Candidate selection
  - Edge selection
  - Edge flip

Initial settings

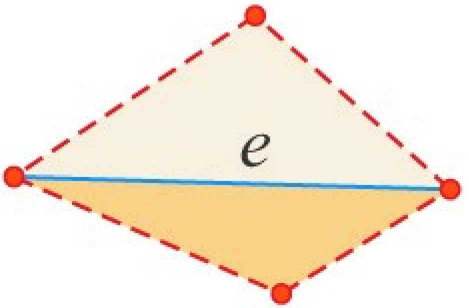
- 2D textures (IDs)
- Image: (m,n) → Edges: 3(m-1)(n-1)+(m-1)+(n-1)
- 32bit precision (256*256 → 195586)

- TexImg – input image
- TexEdge (V1,V2,Adj1,Adj2)
- TexNeigh (N1,N2,N3,N4)
- TexVert (x,y)
- TexCand (A,F,ID) – 2x

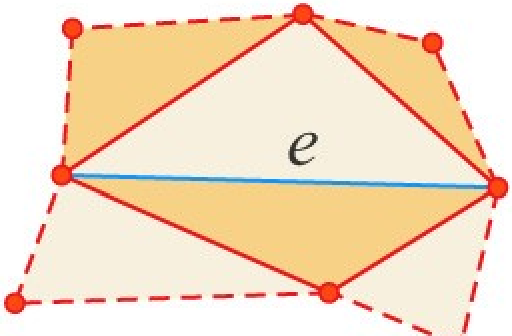
- FBO (framebuffer object) – render to texture

Candidate selection

- Fragment shader
- Output in texCand (A*vailable*,F*lippable*,ID)
- Discard non candidate fragments
- Occlusion query - #candidates (depth test)
- Cost function (4 / 8 neighbours)
- Geometry test – convexity

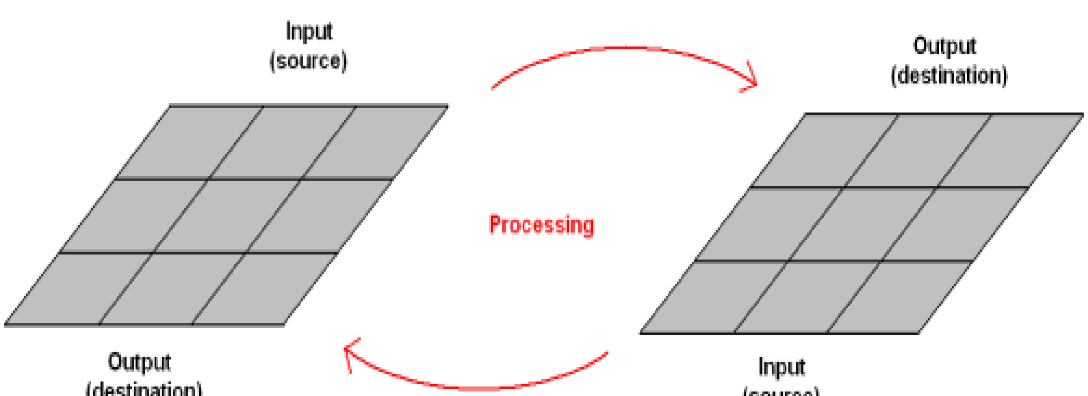


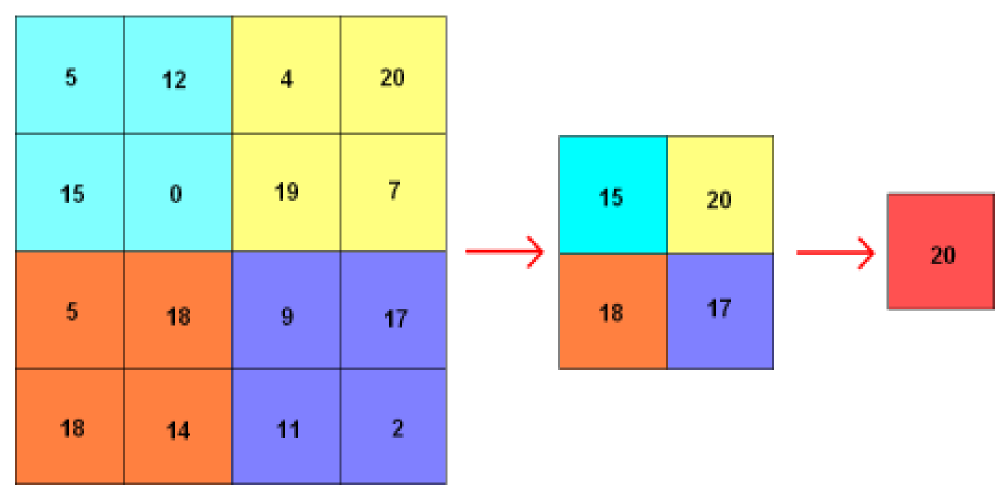

```
if((colV1+colV2) >= (colA1+colA2))  //do not flipp
{
   //OUT.color = float4(0.0, 0.0, ID, 0.0);
   discard;
}
else  //flipp
{
   OUT.color = float4(1.0, 0.0, ID, 0.0);
}
```



Edge selection

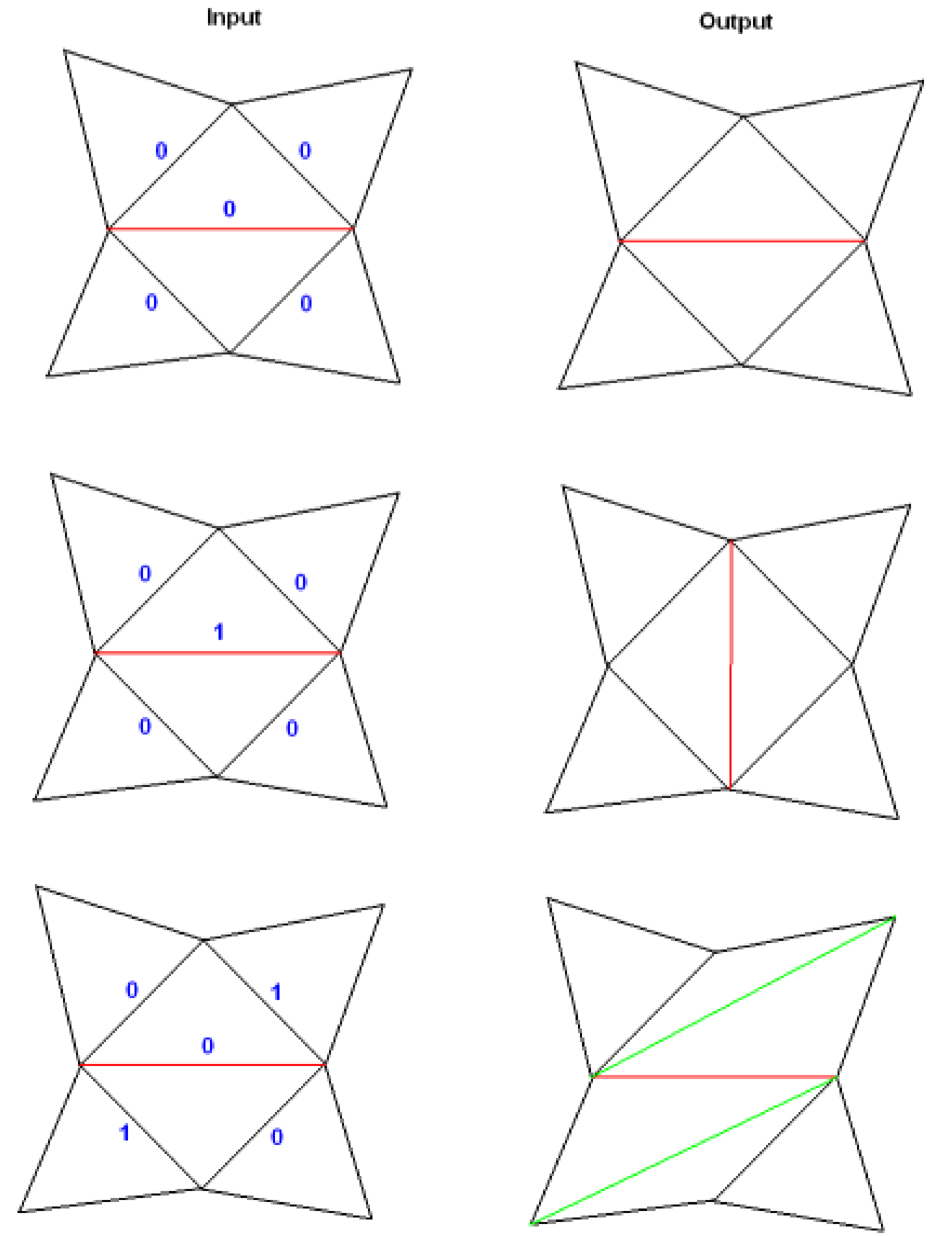
- Fragment shader
- Comparison neighbours, min(IDs)
- Iterative process
- TexCand1/2 – read / write (ping - pong)
- Occlusion query (reduction scheme)
- Process / discard fragment
- If($\sum(\#occQ_i)$ == #can) $\rightarrow$ stop iteration





Edge flip

- Based on F flag
- 3 types of edge flips
- MRT Multiple render targets
  - texEdge
  - texNeigh
- 2 pairs of texEdge, texNeigh
- Preserve orientation



Iteration

- Candidate selection process (1st step)
- If (#can) → do main iteration
- Else → next processing, draw mesh

- Draw mesh
  - GS
  - download to CPU

```
for(int i=start; i<end; i++)
{
  float2 edgeTC = float2(i%resx, floor(i/resx));
  float4 edgeInfo = tex2D(texEdge, edgeTC);

  float v1ID = edgeInfo.r;
  float v2ID = edgeInfo.g;

  float2 v1TC = float2(v1ID%resx, floor(v1ID/resx))
  float2 v2TC = float2(v2ID%resx, floor(v2ID/resx))

  float2 v1 = tex2d(texVert, v1TC);
  float2 v2 = tex2d(texVert, v2TC);

  emitVertex(mul(matrixModelProj, v1): POSITION);
  emitVertex(mul(matrixModelProj, v2): POSITION);
  EndPrimitive();
}
```

Thank you.