Encryption of Progressive Meshes

Goals of this work

- Medium meshes often require many Megabyte of storage data and big meshes even hundreds of Megabyte
- Especially those high level Meshes are often licensed under very restrictive conditions.
- They are very expensive in production
- So this data should be encrypted to store it in a secure way
- Working with such amount of data is already a very CPU intensive task, if it needs to be encrypted too it gets even worse
- So our goal is to determine if the cost of encryption can be reduced while still keeping a decent level of security

Encryption of Progressive Meshes

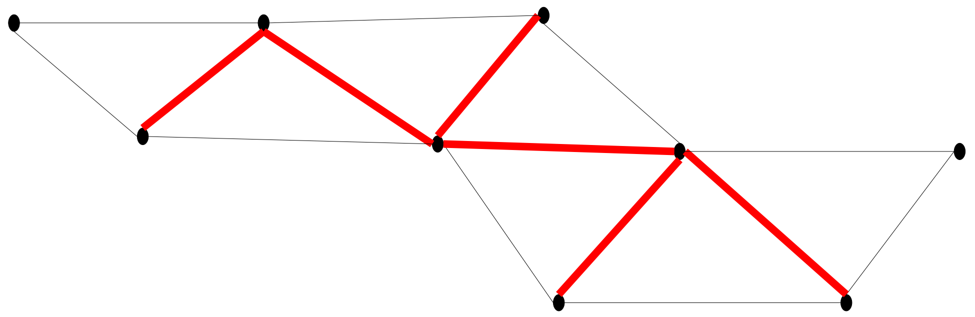
- Geometry Primer
- Mesh Simplification Basics
- Hausdorff Distance
- Progressive Meshes
- Progressive Meshes in MPEG4
- Testimplementation
- Testresults

Geometry Primer

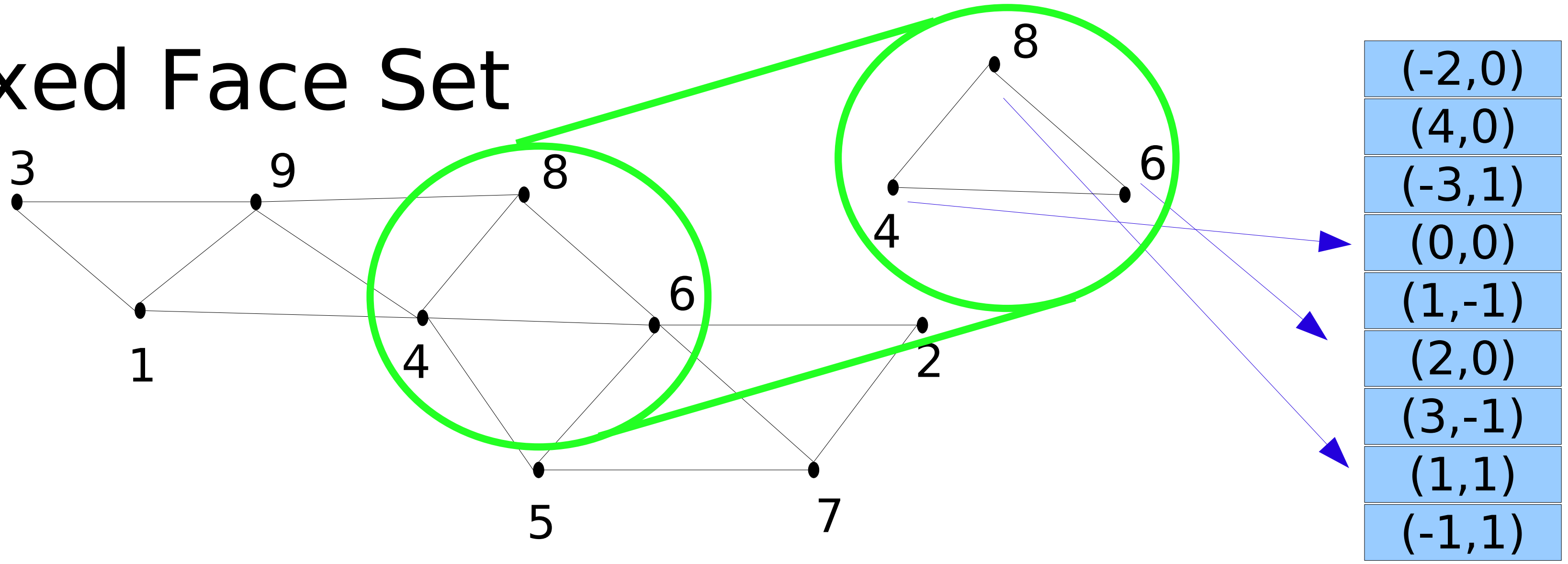
- Vertex
  - a point in 3D
- Triangle
  - A closed polygon consisting of three vertices that are connected through three edges
- Trianglestrip
  - A set of triangles where every triangle shares two edges with other triangles of the strip (except the first and the last ones)
- Indexed Face Set
  - A representation of triangulated polygons that is commonly used in 3D Applications. Every triangle corner is an index pointing in the list of vertices

Geometry Primer

Trianglestrip



Indexed Face Set



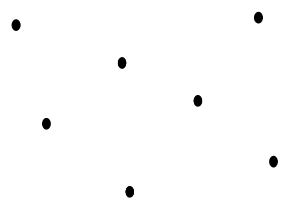
Geometry Primer

- Indexed Face Set in VRML

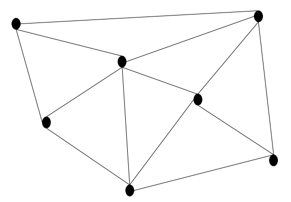
```
<IndexedFaceSet solid="true"
                coordIndex="7 0 3 -1, 4 0 7 -1, 2 6 7 -1, 2 7 3 -1,
                            1 5 2 -1, 5 6 2 -1, 0 4 1 -1, 4 5 1 -1, 4 7 5 -1,
                            7 6 5 -1, 3 0 2 -1, 2 0 1 -1, ">

           <Coordinate DEF="coord_Cube"
                point="-1.000 1.000 -1.000, -1.000 -1.000 -1.000,
                       -1.000 -1.000 1.000, -1.000 1.000 1.000,
                        1.000 1.000 -1.000, 1.000 -1.000 -1.000,
                        1.000 -1.000 1.000, 1.000 1.000 1.000, " />

</IndexedFaceSet>
```

Geometry Primer

- Triangulated Irregular Network
- A set of irregulary distributed points



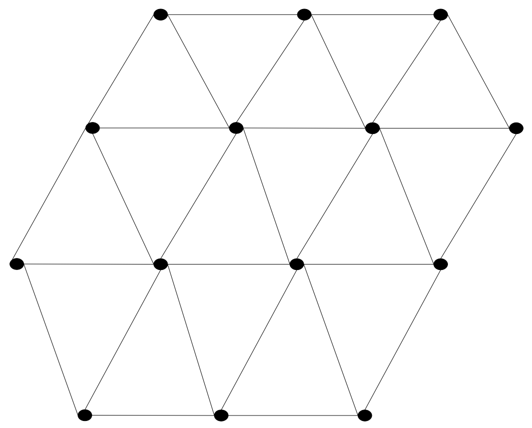
- Connected through triangles



- Example Application: GIS

Geometry Primer

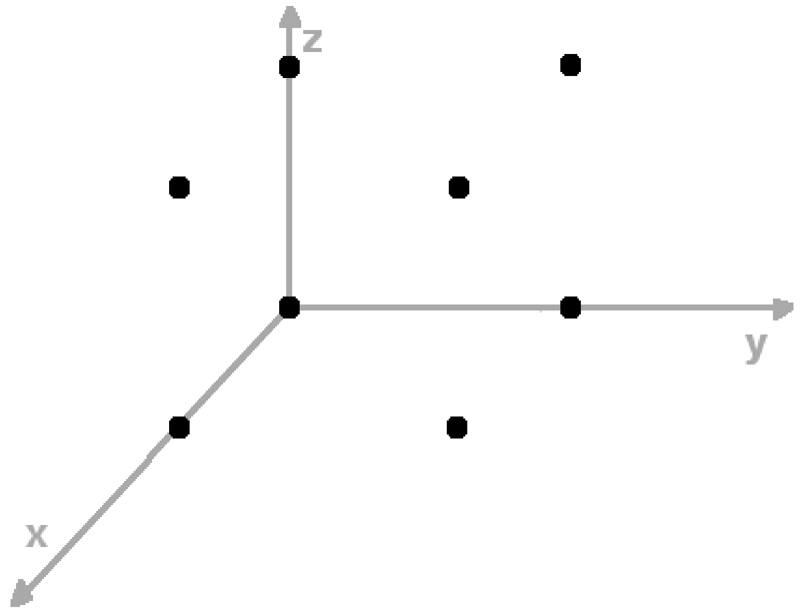
- Semi-Regular Mesh
- a semiregular solid has all its faces regular polygons, and it has the same pattern of polygons around each vertex (most of the vertices)

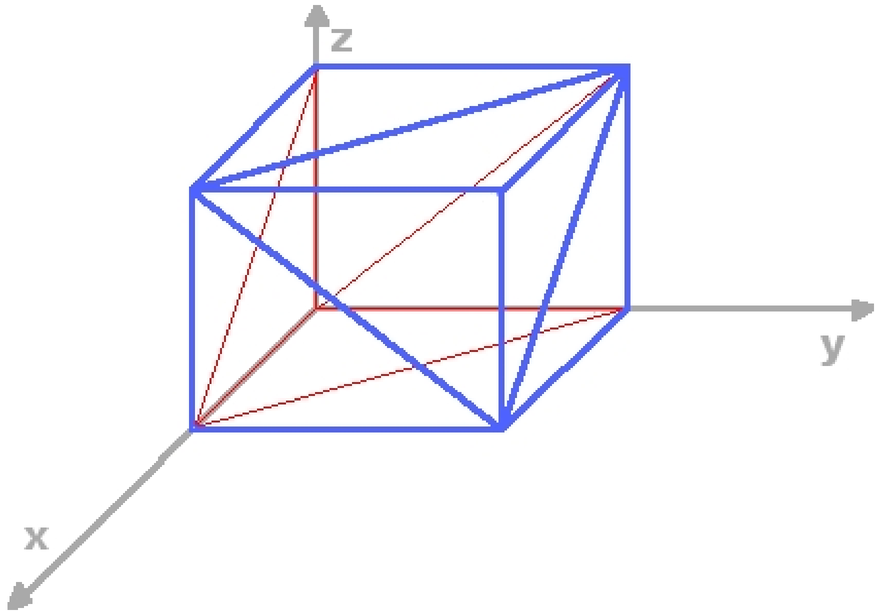


- Semi-Regular meshes can be stored efficiently as the connectivity information is implicit defined

Geometry Primer

- A Triangulation of a 3D Model can be separated into two parts
- Geometry Information (Vertex-Positions)
- Connectivity Information (Vertices that form triangles

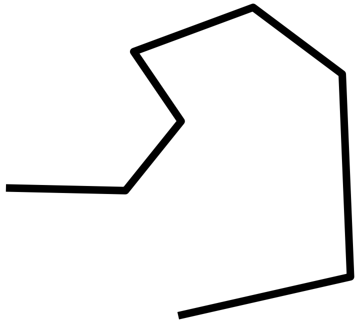




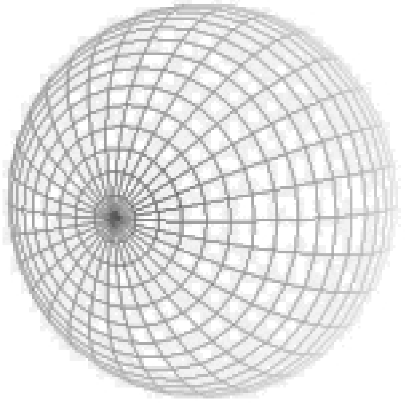
Geometry Primer

- Manifolds

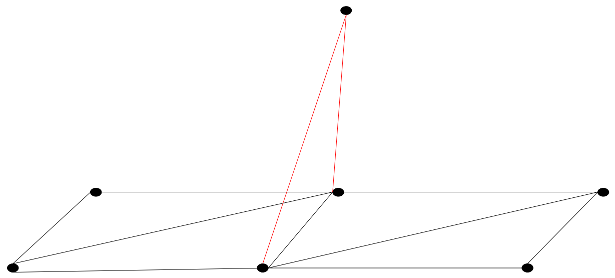
One -Manifold



Two -Manifold



No Manifold



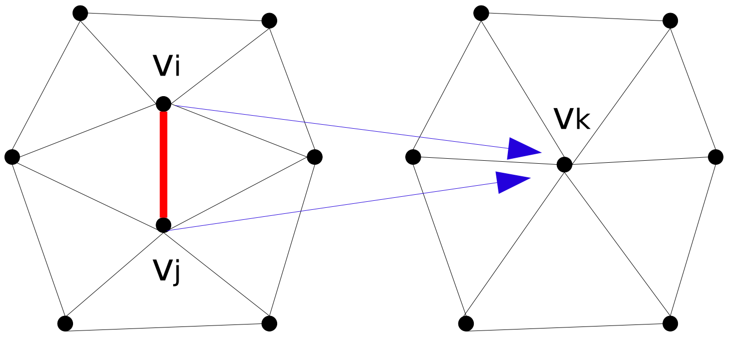
Mesh Simplification Basics

Basic Mesh simplification operations:

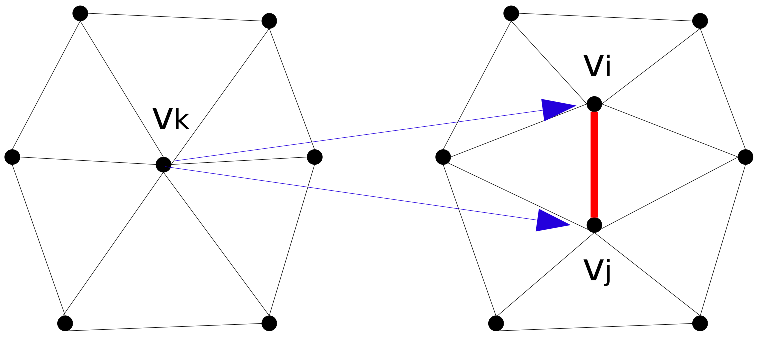
- Edge-Collapse
- Vertex-Pair-Collapse
- Triangle-Collapse
- Cell-Collapse
- Vertex-Removal (Vertex-Decimation)

Edge-Collapse

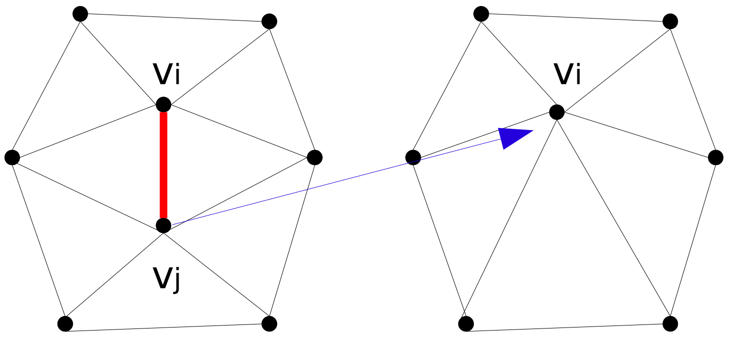
Edge-Collapse



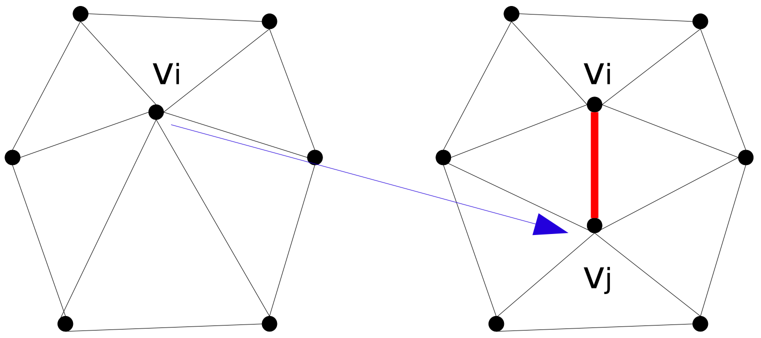
Vertex-Split



Half-Edge-Collapse

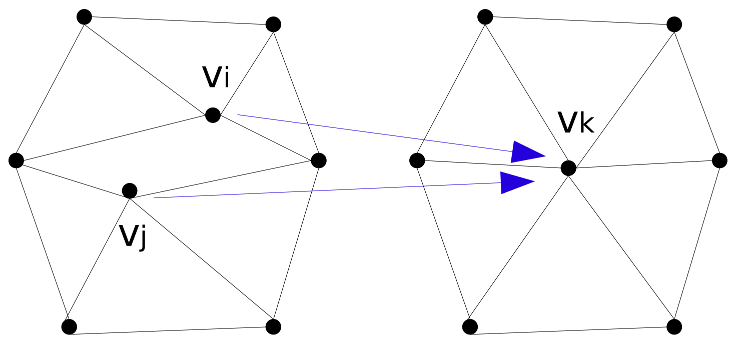


Vertex-Split

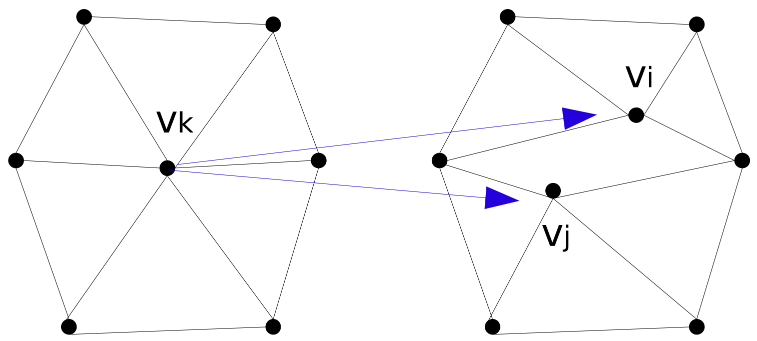


Vertex-Pair-Collapse

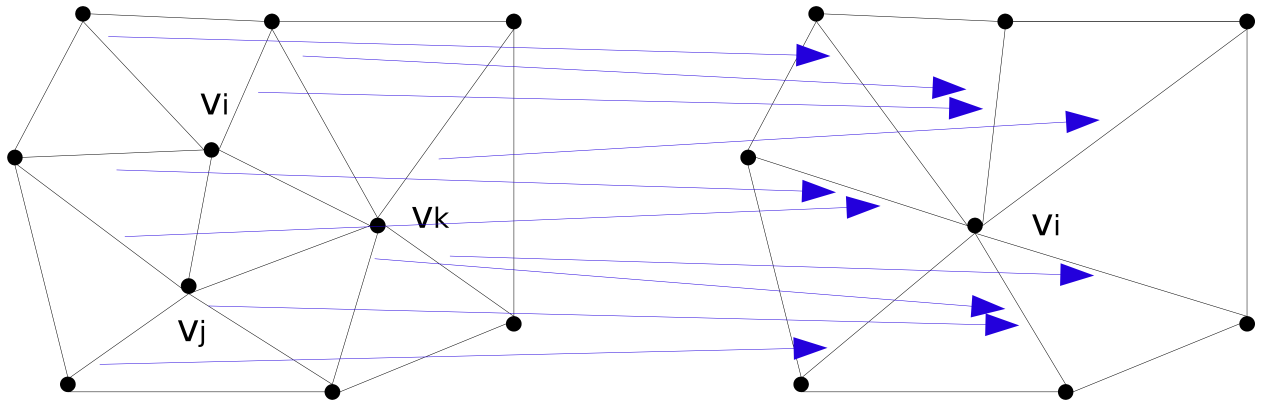
Vertex-Pair-Collapse



Vertex-Split

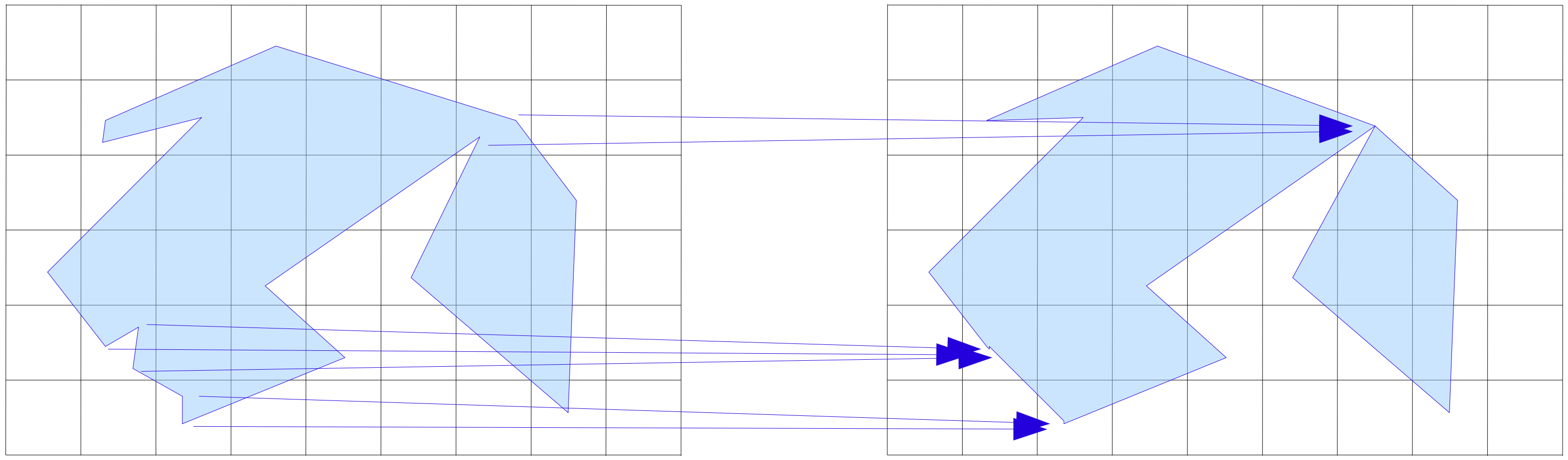


Triangle-Collapse



Cell-Collapse

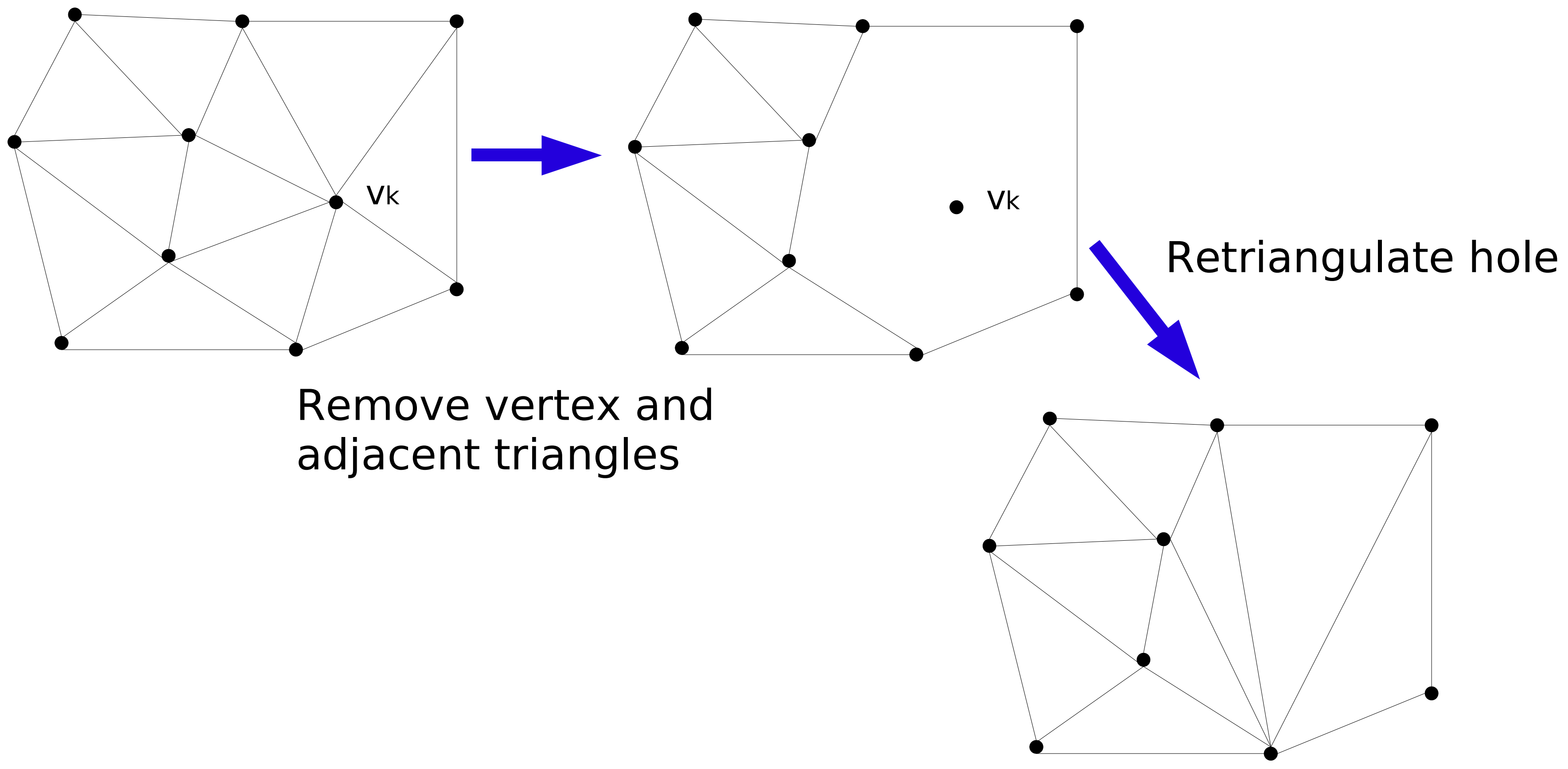
- Fast
- Simple



- Problem: Easily produces degenerated Polygons

Vertex-Removal

- A generalization of the half-edge-collapse



Hausdorff Distance

- Calculating the difference betweeen 2D Images is relatively easy. One can simply subtract one pixel in the second image from the pixel in the first image. This is because the pixel positions in the first image correspond to those in the second image (or the positions can be easily interpolated)
- Such a simple algorithm for 3D Objects is not available and simply calculating the difference between vertex positions may not even be possible as the vertices in the first Object may not have corresponding vertices in the second object.

Hausdorff Distance

- One solution to this Problem that is commonly used is the Hausdorff distance
- Basically it calculates the maximum of all minimum vertex to plane distances
- Vertex to Surface Distance

$$d(p, S') = \min_{p' \in S'} \| p - p' \|_2$$

- One-sided Hausdorff Distance

$$d(S, S') = \max_{p \in S} d(p, S')$$

- Symmetric Hausdorff Distance

$$d_S(S, S') = \max[d(S, S'), d(S', S)]$$

Progressive Meshes

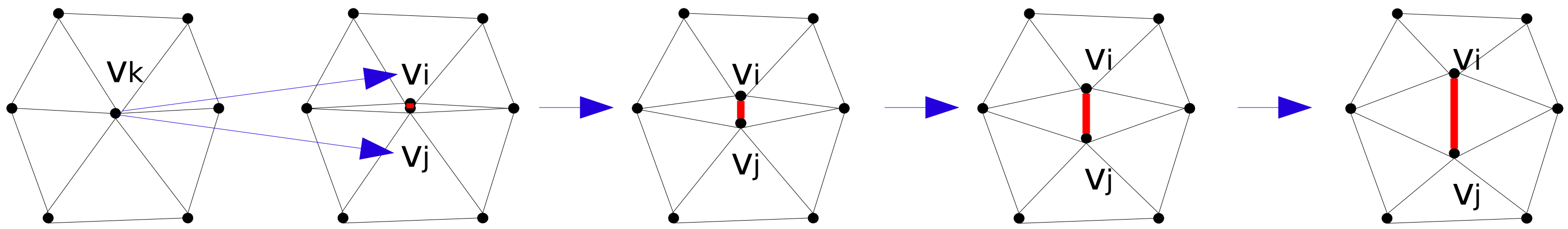
- Progressive Meshes
- Compressed Progressive Meshes
- Progressive Geometry Compression
- Edgebreaker
- Touma-Gotsman
- Quadric Error Metric

Progressive Meshes

- By Hugues Hoppe
- The Mesh M is represented through a base Mesh M_0 and a series of Refinement steps (vertex-split)

$$\begin{tikzcd}
M_0 \arrow[r] & M_1 \arrow[r] & M_2 \arrow[r] & M_3 \arrow[r] & M_4 \arrow[r] & M_5 \arrow[r] & \cdots\cdots\cdots \arrow[r] & \mathbf{M}
\end{tikzcd}$$

- Vertex-Splits with Geomorphs for a smooth LOD change (only for Realtime Apps)



Progressive Meshes (creation)

- The Progressive Mesh Hierachy is created through a series of edge-collapse operations
- The quality of the mesh depends largely on the algorithm selecting the edges for the collapse operator
- All possible edge-collapses are stored in a priority queue according to the “cost” of applying them
- After an edge-collapse the immediate neighborhood is updated and the queue positions are recalculated

Progressive Meshes

- Hoppe uses an Energy Metric for qualifying the mesh fidelity that is derived fom his previous work (Mesh optimization)
- The goal of mesh optimization is to find a mesh that fits a set of points $x_i \in \mathbb{R}^3$ and has a small number of vertices
- The energy metric consists of 4 parts
- $E_{dist}(M) + E_{spring}(M) + E_{scalar}(M) + E_{disc}(M)$
- $E_{dist}(M)$ ... the total squared distance of the points to the Mesh (measures accuracy)

Progressive Meshes

- $E_{spring}(M)$ ... corresponds to a spring of rest-length zero on each edge (measures conciseness). Shorter edges require less energy
- $E_{scalar}(M)$ ... measures accuracy of the scalar attributes
- $E_{disc}(M)$ ... measures geometric accuracy of the discontinuity curves to preserve the overall appearance of the mesh attributes (color,texture,...)

Progressive Meshes

- The cost used for sorting the priority queue is the difference between the energy functions before and after the edge collapse (lower delta means lower cost)

$$\Delta E = E_{K'} - E_K$$

- The energy value is calculated by minimzing this function through varying the vertex position of the collapsed vertex and the attributes

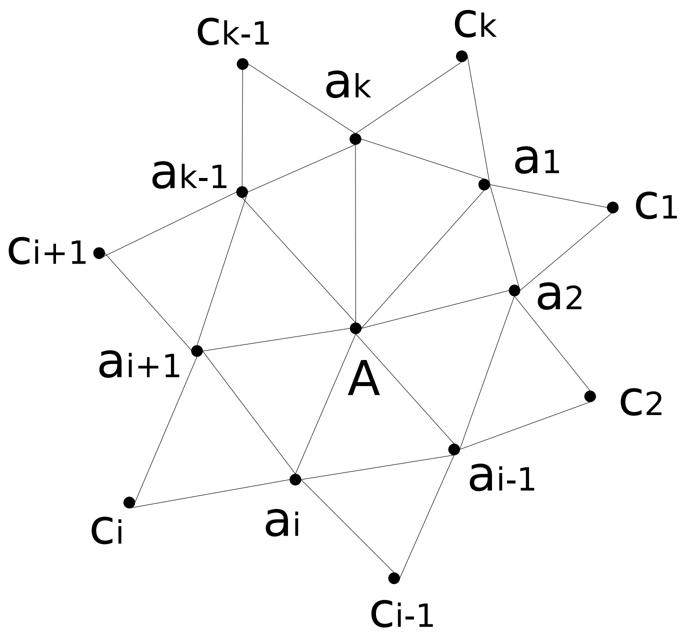
$$E_{K'} = \min_{V,S} E_{dist}(V) + E_{spring}(V) + E_{scalar}(V,S) + E_{disc}(V)$$

Compressed Progressive Meshes

- Pajarola Rossignac
- They group series of refinement steps to batches to get better compression results (granularity vs. compression ratio)
- The lowest level is a coarse mesh at about 5% to 10% the vertex count of the original mesh
- The connectivity information is efficiently encoded using only $\lceil \log_2 \binom{d}{2} \rceil$ bits.
- This is achieved by traversing a vertex spanning tree and indexing the two cut edges per vertex split as a *v out of d* choice for the first edge and *v out of d-1* for the second edge

Compressed Progressive Meshes

- The vertex positions are predicted (and then entropy encoded) through a variant of the Butterfly interpolation scheme.



$$A' = \alpha \cdot \frac{\sum_{i=1}^{k} a_i}{k} + (1-\alpha) \cdot \frac{\sum_{i=1}^{k} c_i}{k}.$$

alpha = 1.15

Compressed Progressive Meshes

- With this interpolation the predicted position of the two original vertices can be expressed as a linear combination of each other

$$\begin{aligned} A' = \alpha \cdot \frac{\sum_{i=1}^{k_a} a_i + v_1 + v_2 + B'}{k_a + 3} \\ + (1 - \alpha) \cdot \frac{\sum_{i=1}^{k_a+1} c_i + b_1 + b_{k_b}}{k_a + 3} \end{aligned}$$

$$\begin{aligned} B' = \alpha \cdot \frac{\sum_{i=1}^{k_b} b_i + v_1 + v_2 + A'}{k_b + 3} \\ + (1 - \alpha) \cdot \frac{\sum_{i=1}^{k_b+1} d_i + a_1 + a_{k_a}}{k_b + 3}. \end{aligned}$$

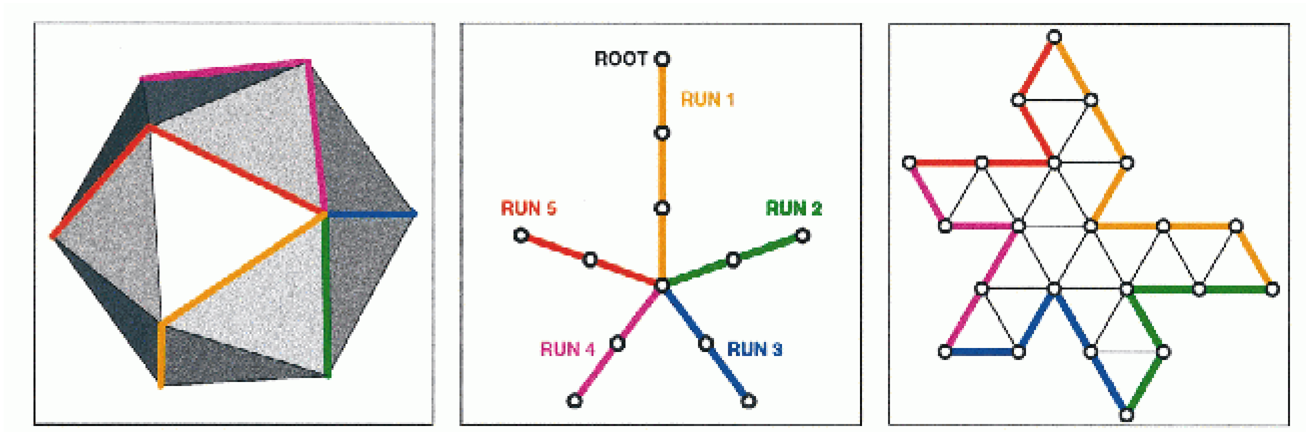
- This equation can be solved with the position of the collapsed vertex leaving only a single predicton error that needs to be stored

Progressive Meshes in MPEG4

- Topological Surgery
- Progressive Forest Split

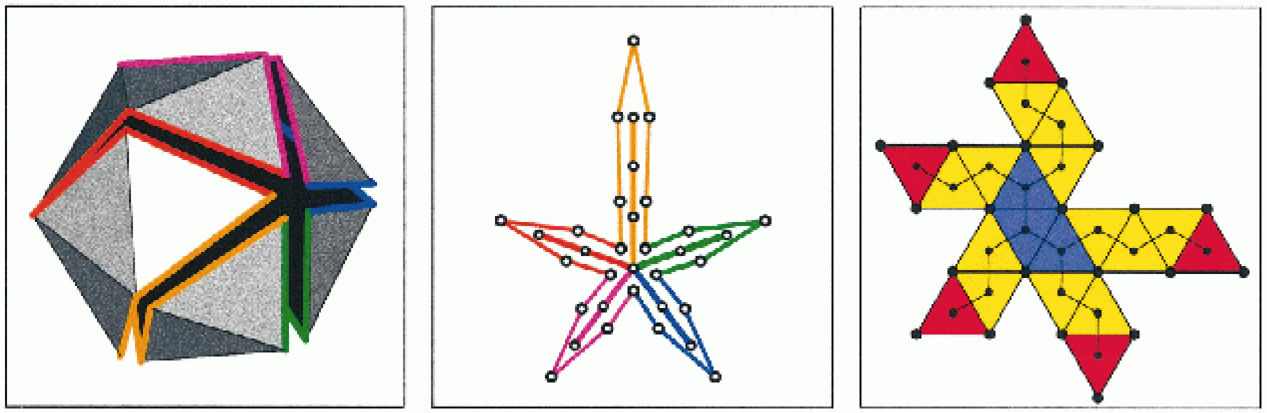
Topological Surgery

- Create a Spanning Tree over the edges and vertices
- The branching nodes are connected through so called vertex runs



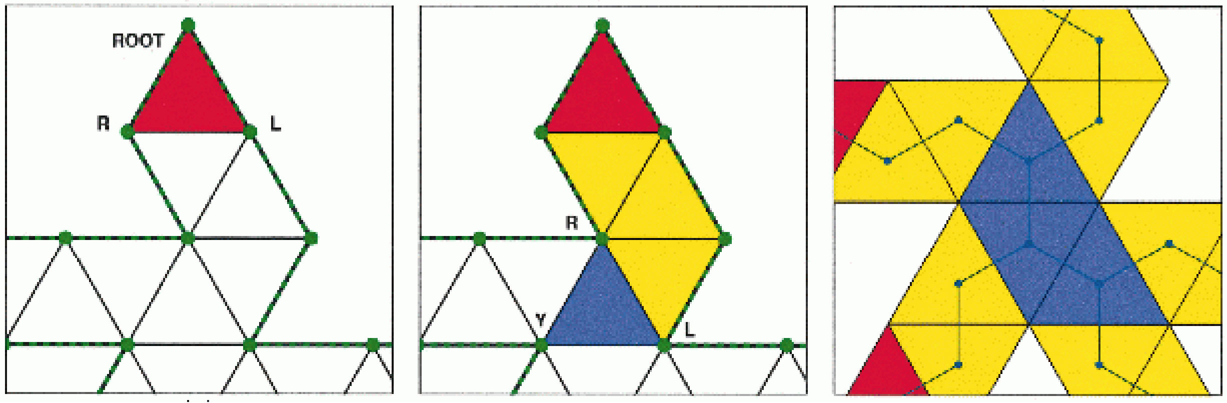
Topological Surgery

- Cut the Mesh along the Tree.
- The resulting Mesh can be flattened to a 2d planar graph
- The branching triangles connect 3 triangle runs
- Create a bounding loop of the polygon



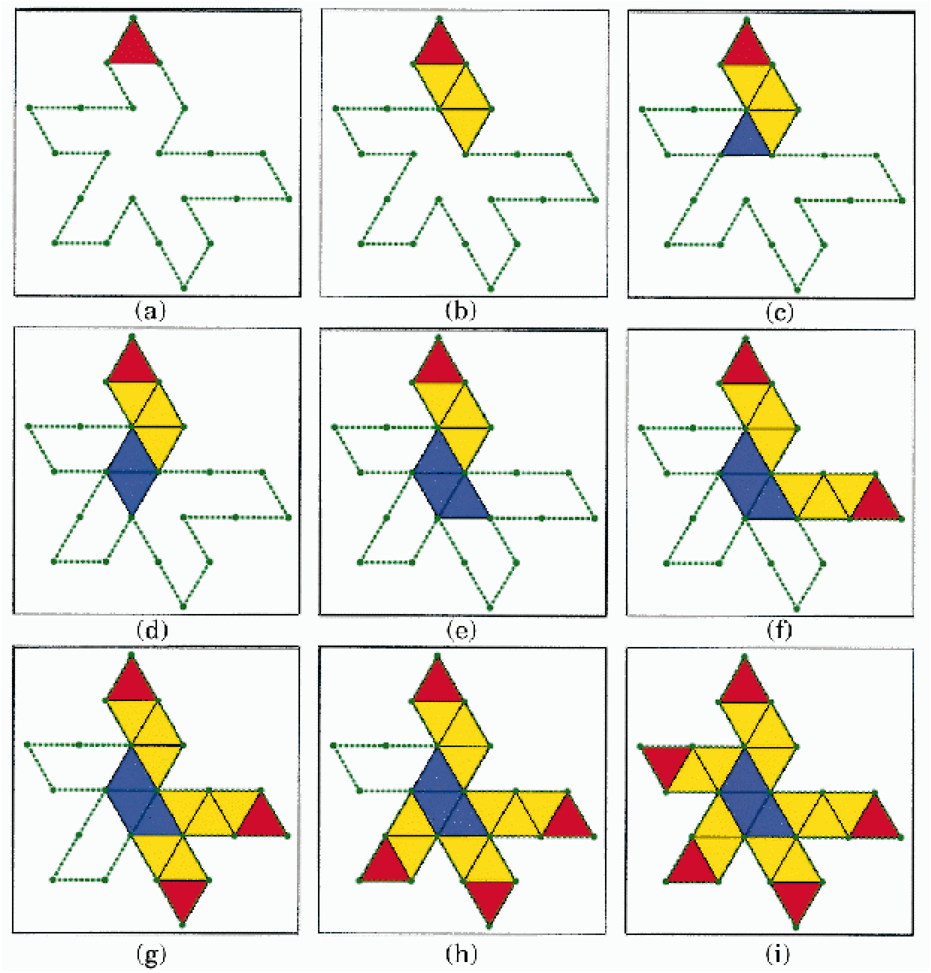
Topological Surgery

- Triangles in a run are connected throguh marching triangles
- The third vertex of a branching triangle is called the Y-vertex
- The two outgoing triangle runs of the branching triangle start with the edges (L,Y) and (R,Y)



Progressive Forest Split

- Encoding the the triangle tree

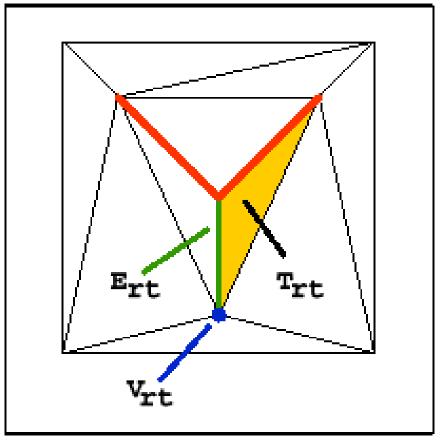


Topological Surgery

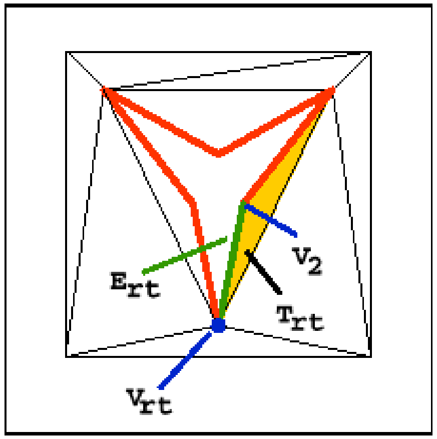
- The Algorithm needs to store:
- Vertex-Tree (triples of *length,branching bit, leaf bit*)
- Vertex Coordinates (the difference to the predictor)
- Triangle Tree (*run length, leaf bit*)
- Marching Pattern (a pattern where each bit describes the position of the next triangle *left/right they*). The bits need to be stored in the order the triangles get visited by the decompression algorithm.

Progressive Forest Split

- Many edges are grouped to a forest along which the mesh is cut open.
- The root vertex of each tree is the vertex with the minimum index
- The root edge of each tree is the edge with the root vertex
- The root triangles is the triangle with the root edge and the minimum triangle index



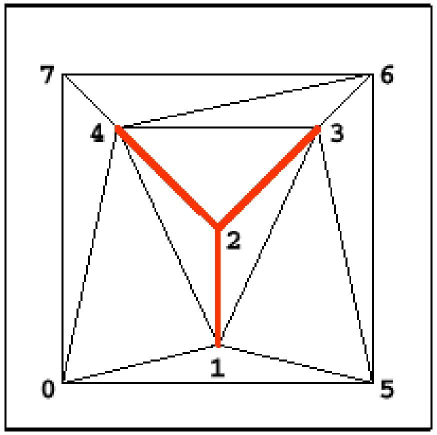
A



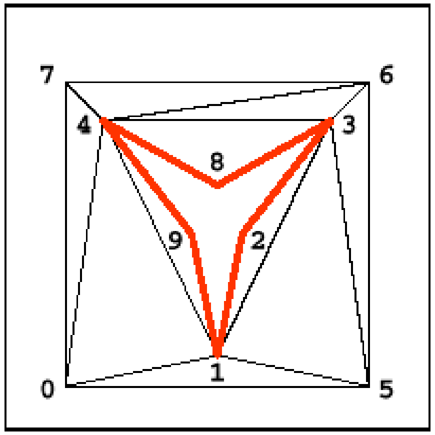
B

Progressive Forest Split

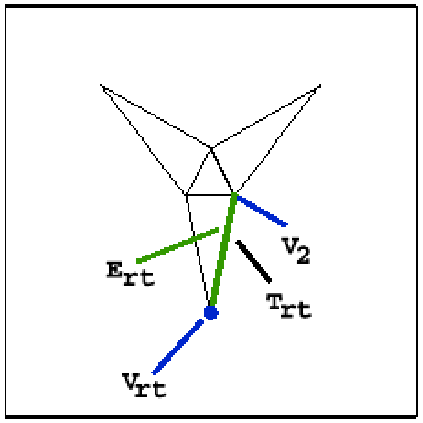
- The root triangle determines the direction of the boundary loop traversal
- The forest of trees can split by splitting on tree at a time. The trees are ordered according to their root vertex index
- The simple polygon that fills the hole is traversed in the same order as the hole itself, which creates a one-to-one mapping of the vertices



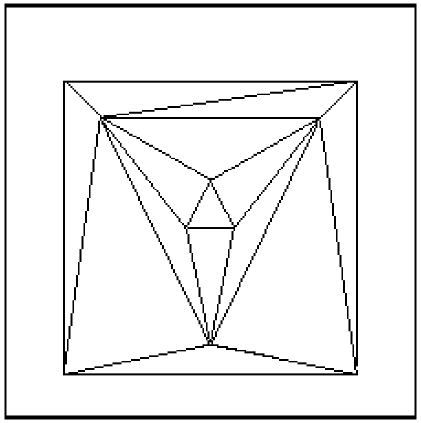
C



D



E


F

Progressive Forest Split

- The boundary loop is traversed by walking from one triangle to next while always keeping contact with the tree.
- This creates a list of vertex runs i.e

```
12333332444444421111
```

- This can be split up into 5 runs (11111; 2; 33333;2; 444444; 2). Every run corresponds to a vertex in the bounding loop(simple polygon) and needs a vertex index. First the algorithm reuses the old indices arising from this hole and then adds new ones at the end of the global list. 11111 => 1; 2 => 2; 33333 => 3; 2 => 8; 444444 => 4; 2 => 9

Progressive Forest Split

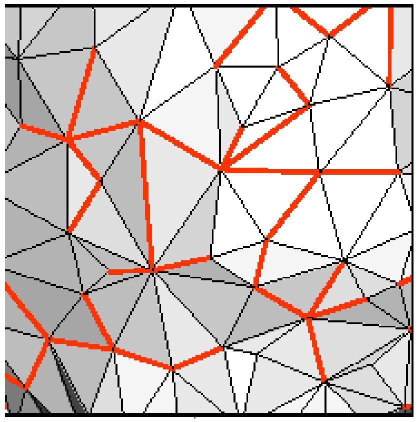
Efficiently encoding the simple polygons

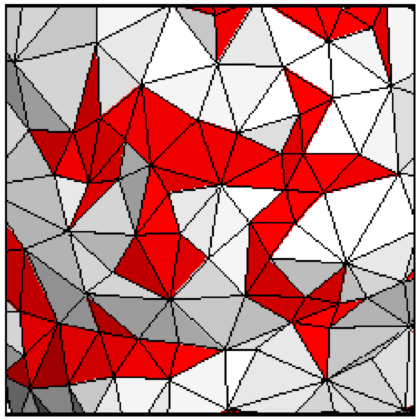
- Especially for small polygons (few triangles) or polygons with short triangle runs, the enconding mentioned earlier is not working properly. Alternative: Fixed length coding of the polygon. 2 Bits per triangle mark the left and right edge as boundary or non-boundary. The triangle tree is traversed in a depth-first manner starting with the left edge first.
- Both encodings are calculated for every polygon and the better one is used
- That way the triangles can be stored with a cost of <= 2 Bits per triangle

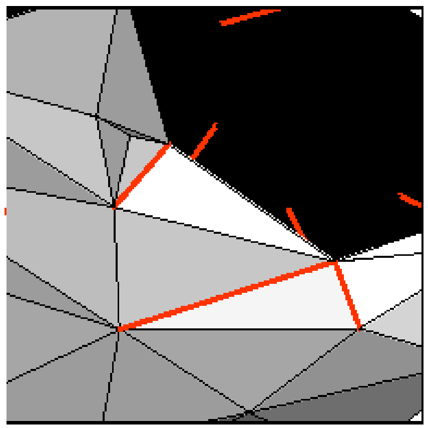
Progressive Forest Split

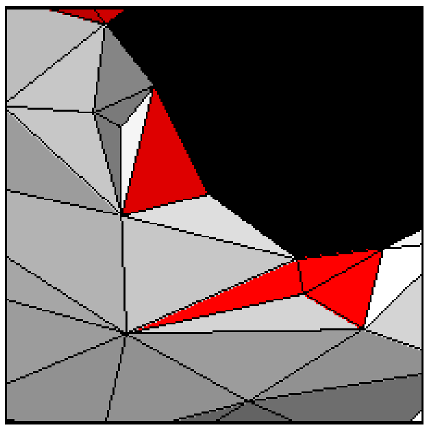
- The forest of edges is stored as a bitmap with one bit per vertex (1 if it belongs to the tree, 0 if it doesn't)
- When reading this bitstream all edges of the forest that are already identified rule out some edges that are yet to be read (because they would produce loops in the tree). These edges may not be stored which further reduces the storage cost
- To create the forests (remove the simple polygons) one can use an edge-collapse algorithm by adding two constraints to the collapsible test:
  - The polygon filling the hole is simple
  - No vertex that is already part of a tree may be added to another one (this includes direct neighbors)

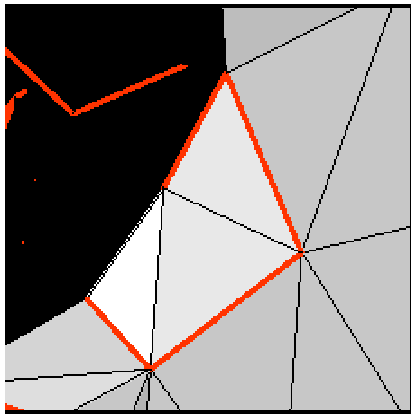
Progressive Forest Split

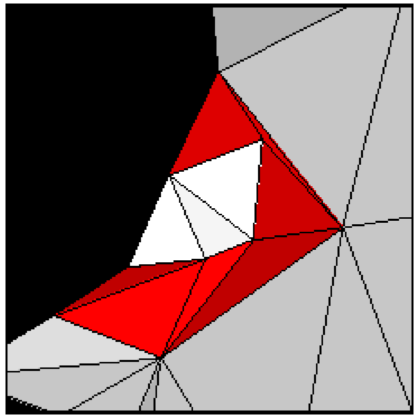

A


B


C


D


C


D

Testimplementation

- Calculate the weight of each vertex. The weight is the average angle between all consecutive triangles
- Remove the vertex with the smallest weight that is non-blocking
- Remove the corresponding triangles from the mesh
- Triangulate the hole arising from the removal
- Mark all vertices of the hole as blocking in this level

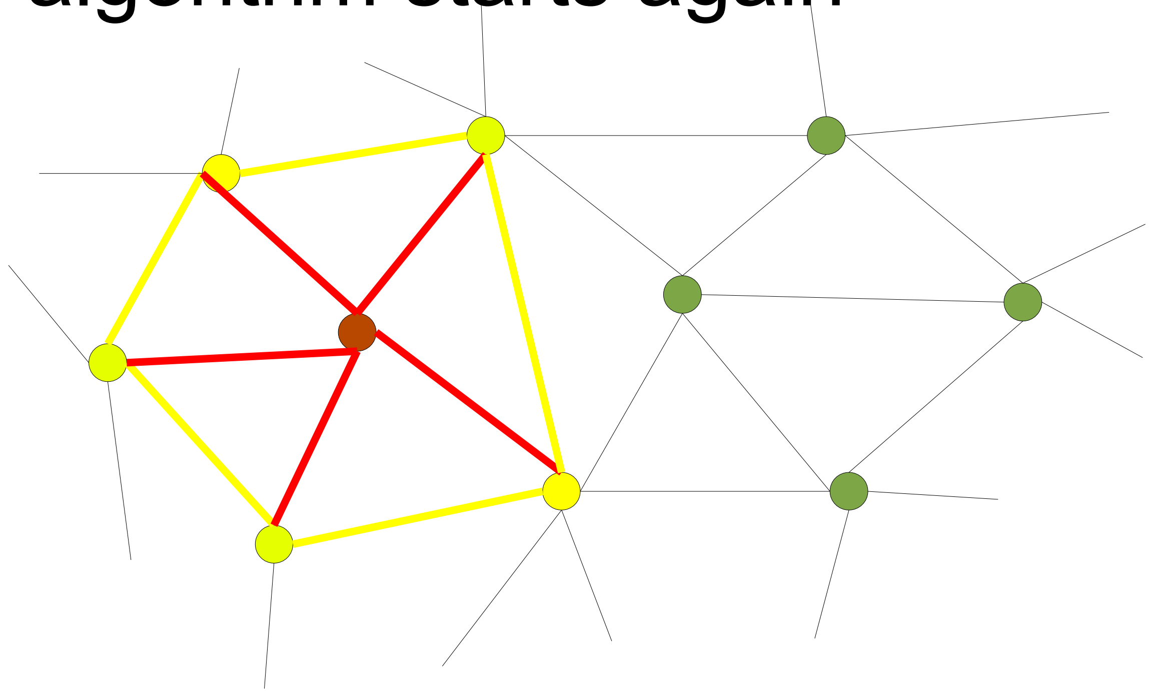
Testimplementation

- Repeat the vertex removal/retriangulation until all remaining vertices are blocking or exceed the max-weight for this level
- In the next level all blocking marks are removed and the algorithm starts again

● Removed

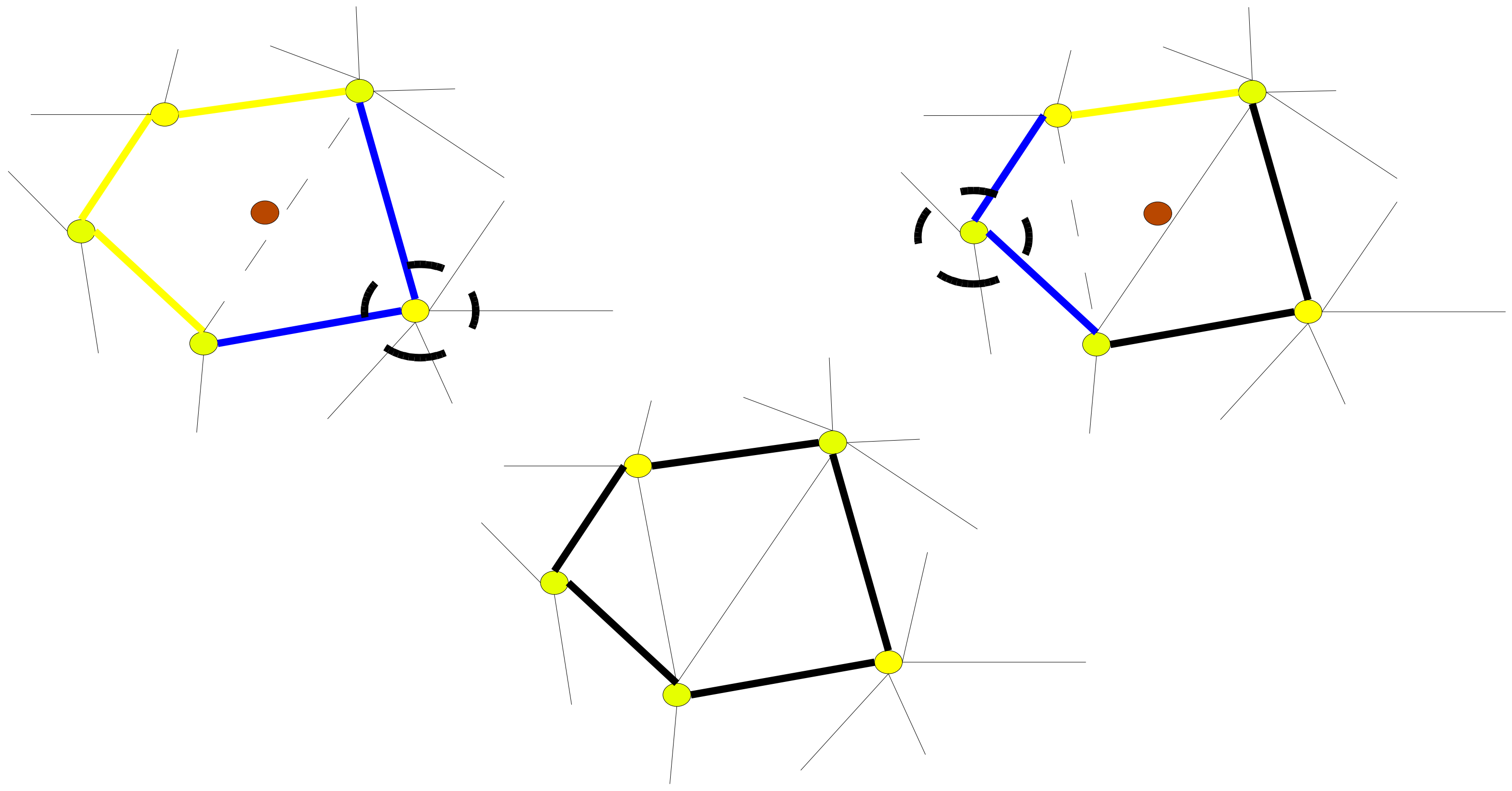
● Blocking

● Removable



Testimplementation

- Triangulate the hole



Testimplementation

- The algorithm produces several levels of detail where higher levels of detail increase the fidelity of the mesh based on the previous levels
- By storing the list of deleted triangles and vertices one can recover every level of detail that was produced during the simplification process
- As vertices are not displaced by some collapse functions the list of vertices can simply be appended

Testimplementation

Base Mesh

```
<Shape>
   <IndexedFaceSet solid="false" coordIndex="1 0 2 -1, 0 2 3 -1, 0 3 1 -1, 3 1 2 -1, ">
     <ProgressiveInformation PointOffset="0" TriangleOffset="0" DeletesTriangles=",,,,"/>
     <Coordinate DEF="coord_level1_Cube" point="-4.266000 -4.266000 4.266000,
                                  -4.266000 4.266000 4.266000, 4.266000 4.266000 -4.266000,
                                   4.266000 -4.266000 4.266000, "/>
   </IndexedFaceSet>
</Shape>
```

Level One

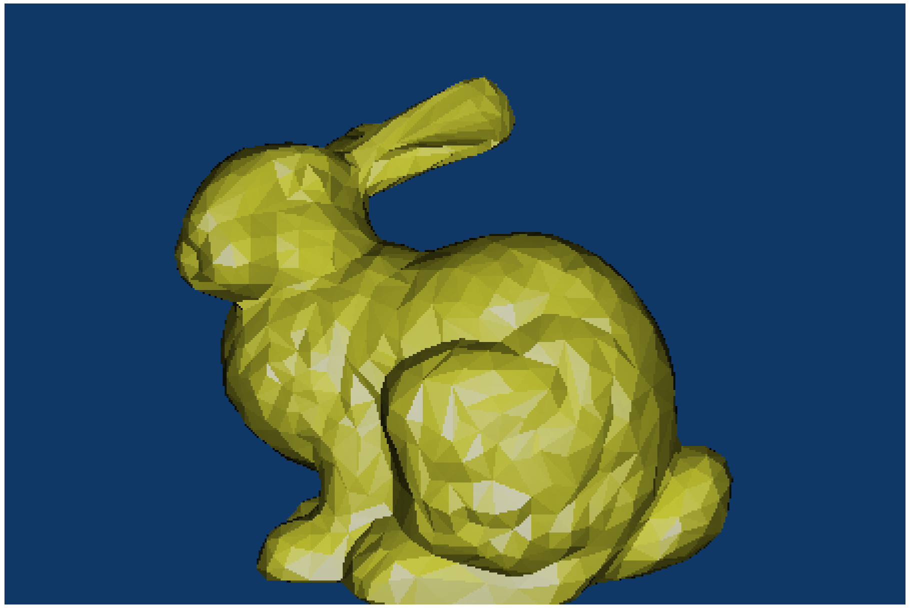
```
<Shape>
   <IndexedFaceSet solid="false" coordIndex="0 1 4 -1, 1 4 2 -1, 0 4 3 -1,
                                              2 3 4 -1, 0 3 5 -1, 0 5 1 -1, 5 1 2 -1, 5 2 3 -1, ">
     <ProgressiveInformation PointOffset="4" TriangleOffset="4" DeletesTriangles=" 0 1, 2 3,"/>
     <Coordinate DEF="coord_level2_Cube" point="-4.266000 -4.266000 -4.266000,
                                                          4.266000 4.266000 4.266000, "/>
  </IndexedFaceSet>
</Shape>
```

Level Two

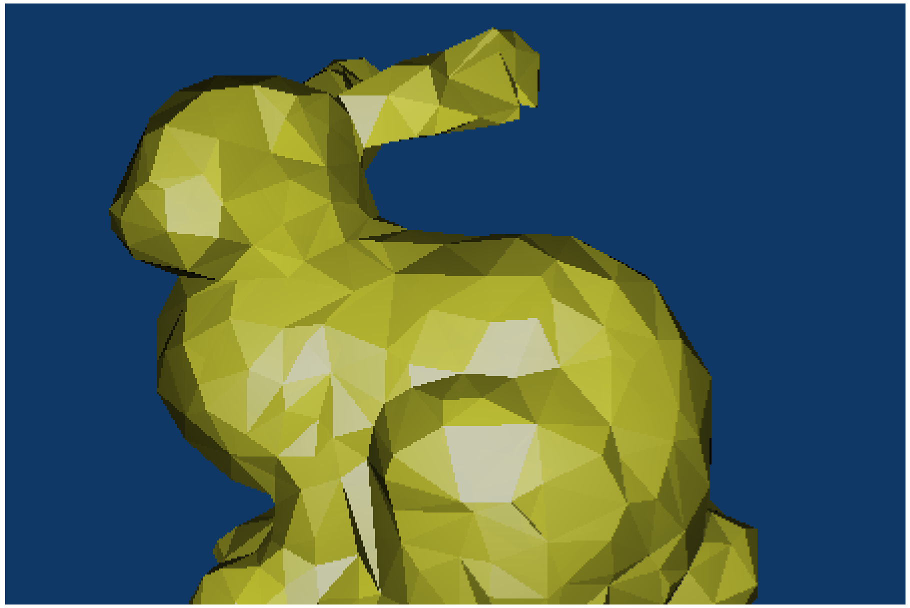
```
<Shape>
  <IndexedFaceSet solid="false" coordIndex="4 6 0 -1, 6 3 0 -1, 2 6 4 -1, 2 5 6 -1,
                                             5 3 6 -1, 5 7 1 -1, 2 7 5 -1, 7 2 4 -1, 1 7 0 -1, 0 7 4 -1, ">
   <ProgressiveInformation PointOffset="6" TriangleOffset="12" DeletesTriangles=" 6 11 7, 10 4 5,"/>
   <Coordinate DEF="coord_level3_Cube" point="4.266000 -4.266000 -4.266000,
                                                        -4.266000 4.266000 -4.266000, "/>
  </IndexedFaceSet>
</Shape>
```

Example

2915 Vertices



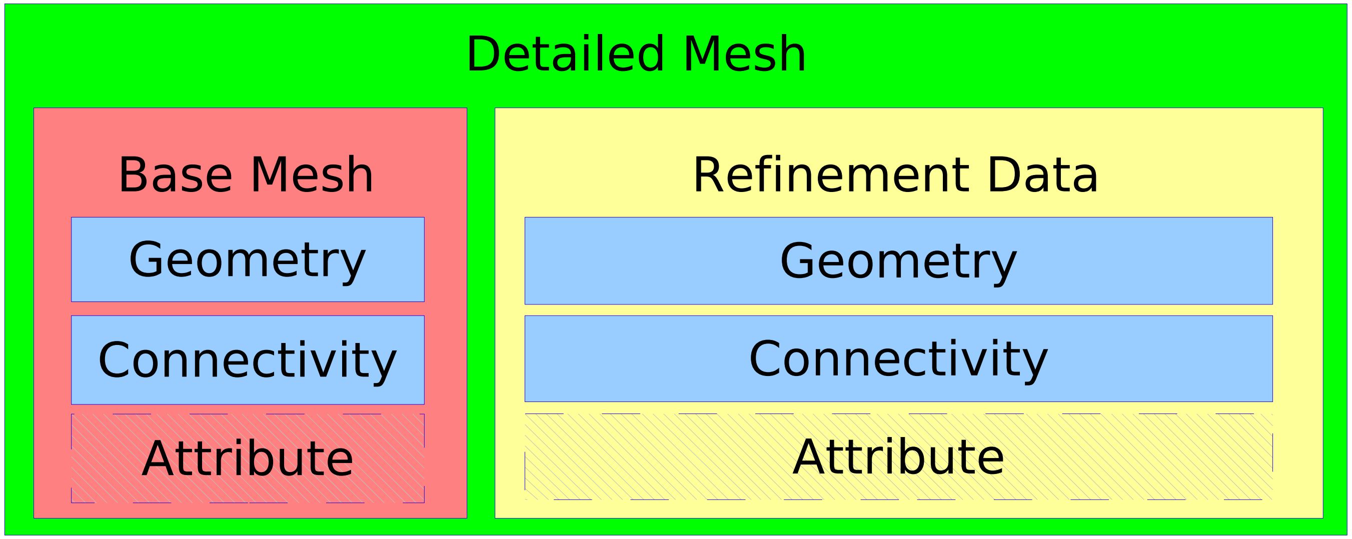
1096 Vertices



Tests

- We heave learned that basically every progressive mesh consists of a base mesh and refinement data and both parts can be split up into geometry data and connectivity Information
- Additional attribute data like texture coordinates, color, ... are not considered here but can basically be handled like vertex positions
- Now we need to take a look at the possible parts that can be encrypted

Basic Progressive Structure



- The Base Mesh may be empty. In that case the whole mesh is represented through the refinement data
- The geometry part of the refinement is normally the geometric error of the predicted positions to the real positions

Possible starting points

One can encrypt:

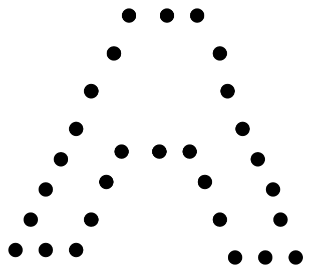
- The geometry information of the Base Mesh
- The connectivity information of the Base Mesh
- The geometry information of the Refimenent Data
- The connectivity information of the Refinement Data
- Any combination or only parts of the above points

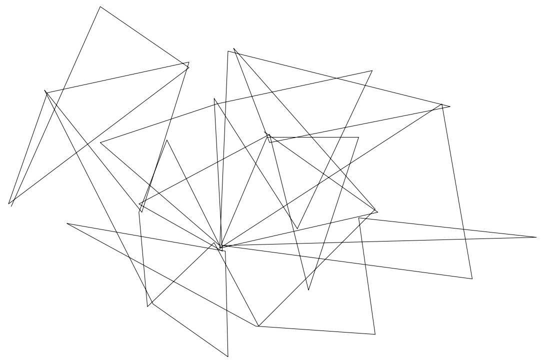
Our work tries to achieve two things:

- Find out which data needs to be encrypted to gain best security while on the other hand keep the costs down
- Use the intact data to predict the encrypted data as good as possible. This is what an attacker would do

Importance of the different parts

- Especially for densely sampled (semi-regular) meshes the connectivity information is very redundant and therefor it should not make much sense to encrypt it.
- This should be true if and only if the geometry information does not depend on the connectivity information i.e. If the vertex positions are stored by saving only the difference to a predicted position the predictions can't be calculated without knowing the immediate neighborhood of the vertex





Calculating Wrong Positions

- Our testimplementation stores the absolute vertex positions, but for simulating advanced progressive formats where only the prediction error is stored we need to simulate this behavior

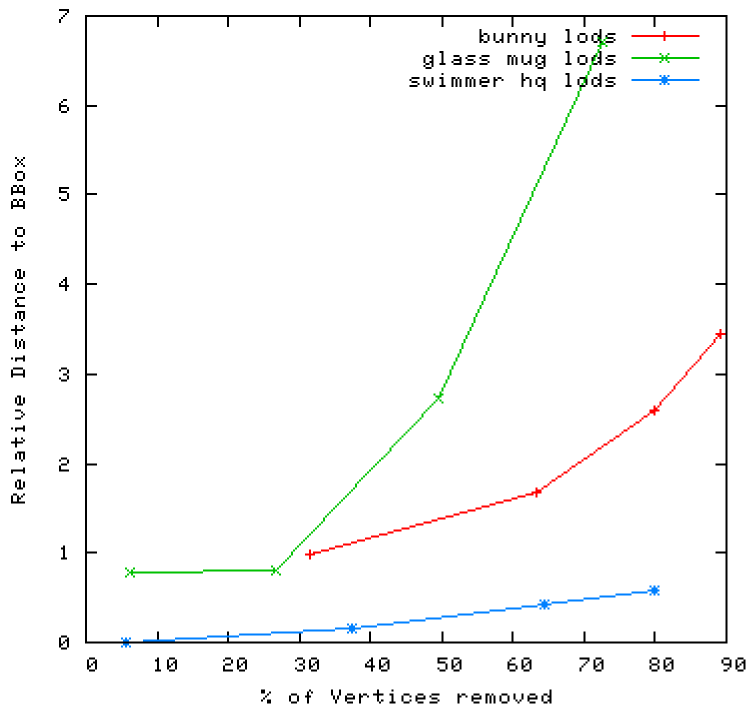
$$v_i = P(v_1, v_2, .., v_{i-1}) + E_{v_i}$$
$$v'_i = P(v'_1, v'_2, .., v'_{i-1}) + E_{v_i}$$

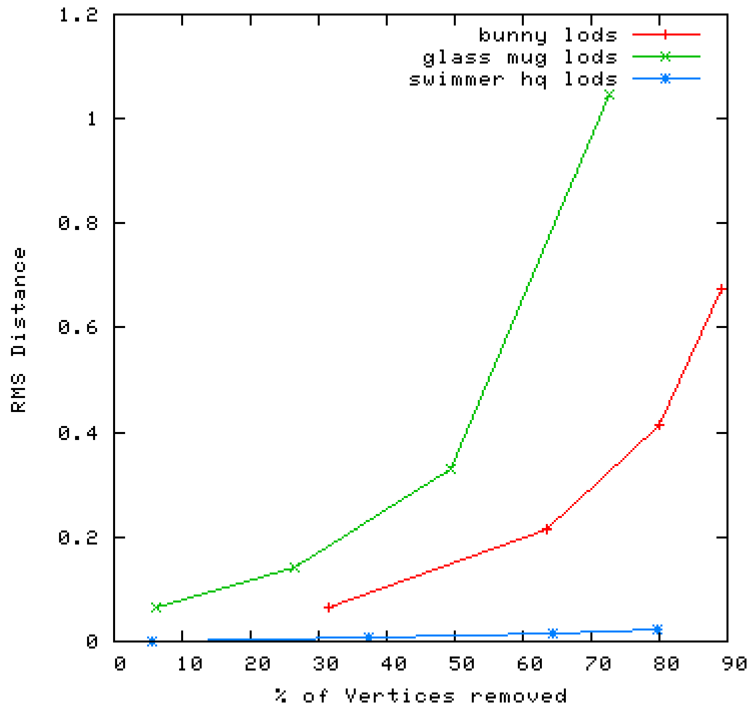
- We first need to calculate the prediction error
- And then add the prediction error to the wrong predicted position

$$E_{v_i} = v_i - P(v_1, v_2, .., v_{i-1})$$
$$v'_i = v_i - P(v_1, v_2, .., v_{i-1}) + P(v'_1, v'_2, .., v'_{i-1})$$

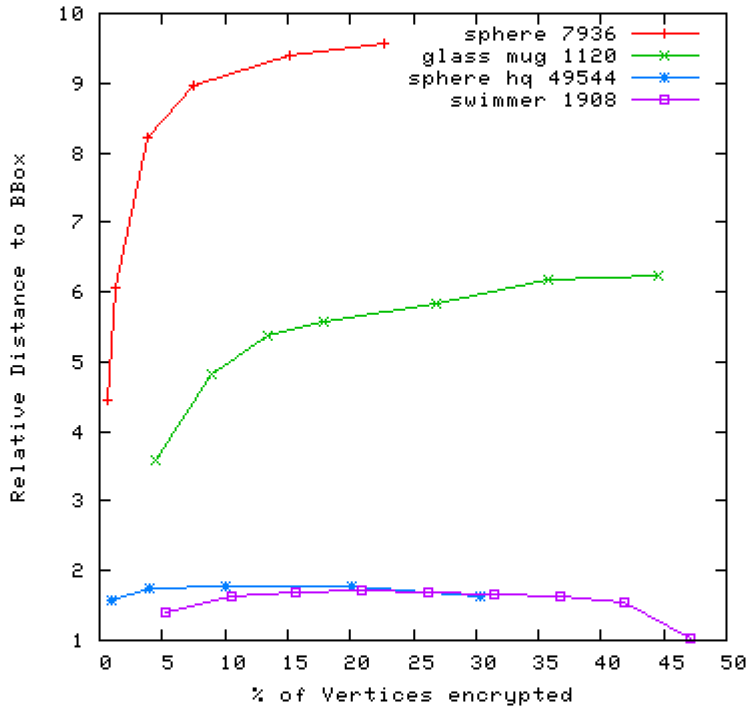
Reference Diagrams

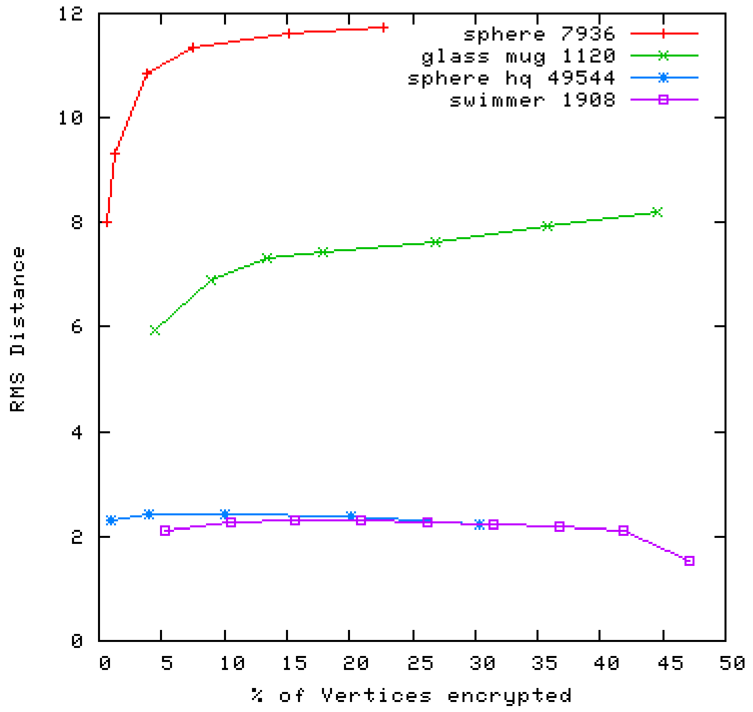
- This is the difference of the normal simplified mesh to the original mesh
- Some lower bound for our further tests





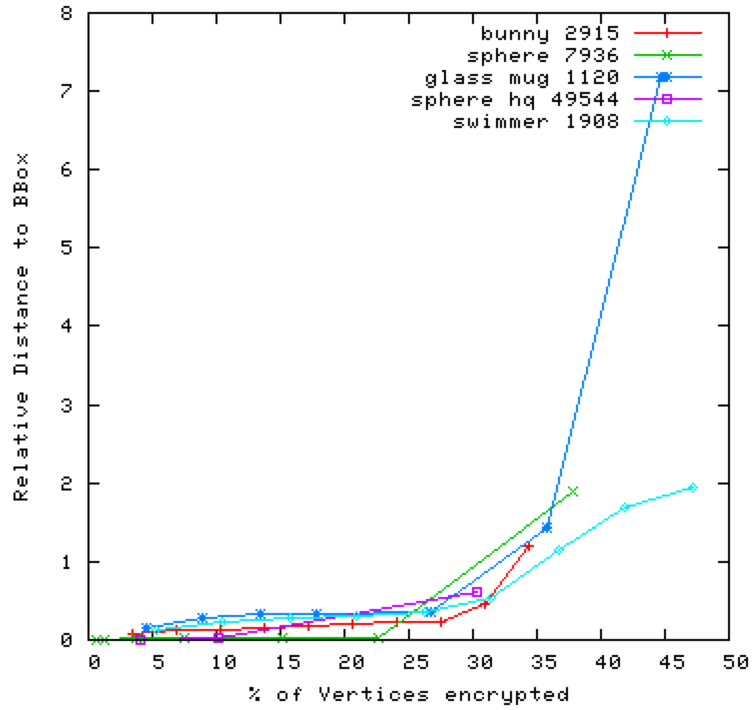
Encrypting absolute Vertex Positions

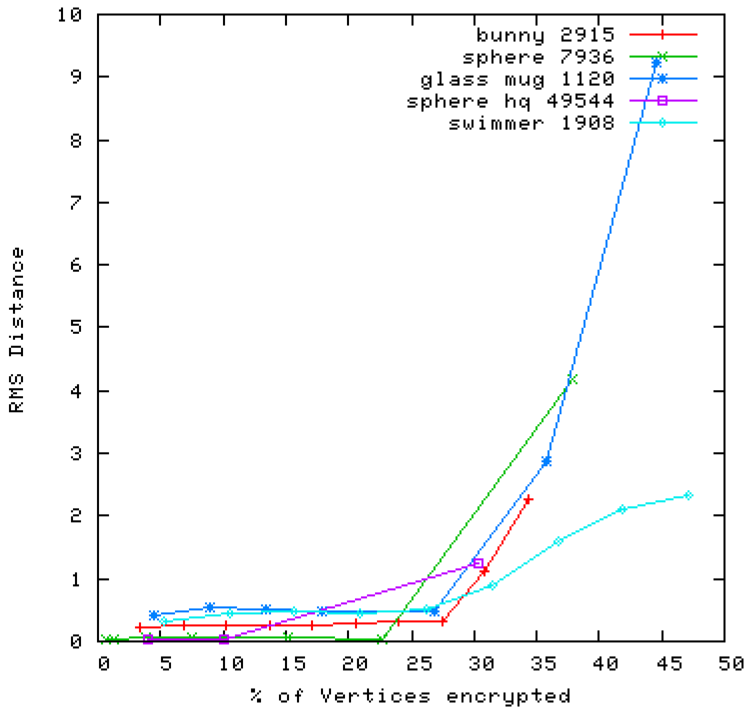




Guessing the absolute Positions

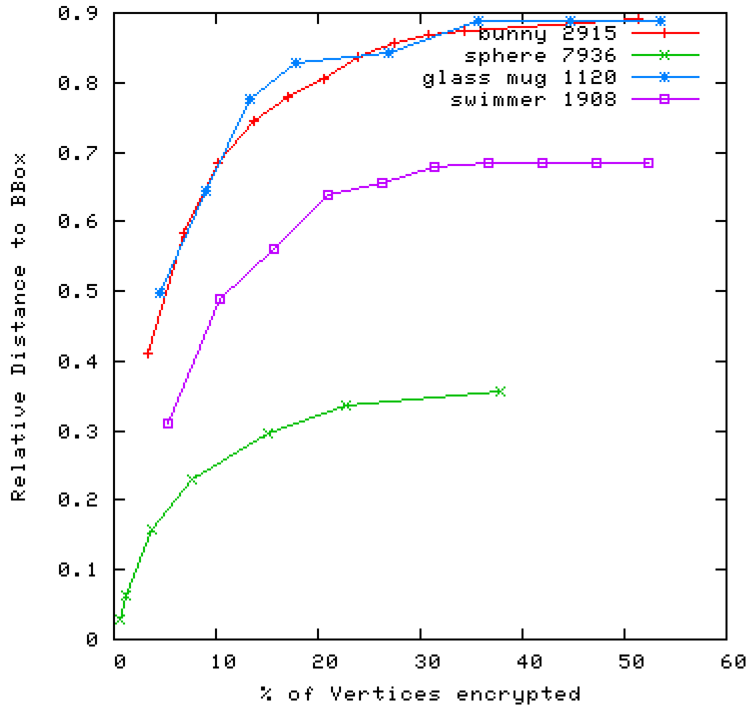
- The algorithm guesses the position based on the direct (one edge away) and indirect (two edges away) neighbours
- This will work for many meshes but not for arbitrary degenerated ones

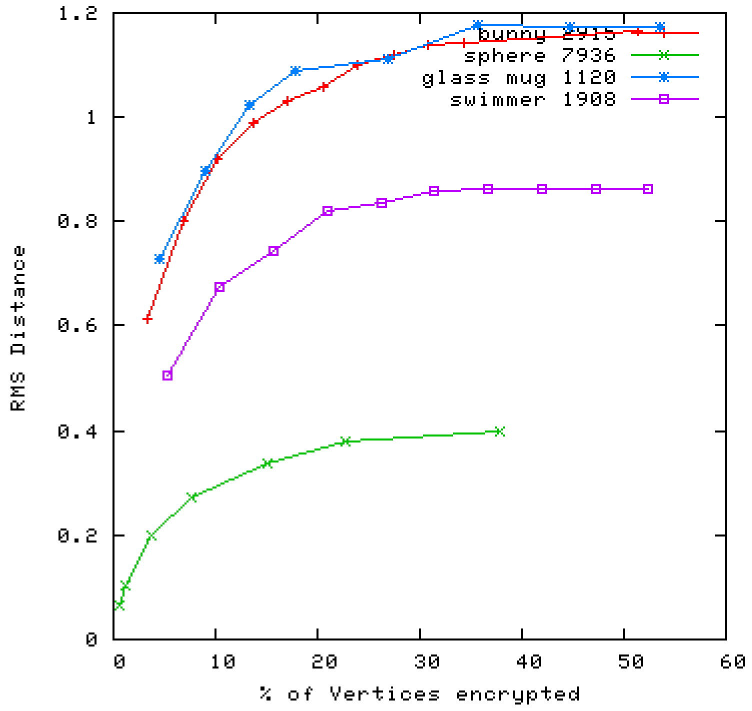




Encrypting the differences to the predictor (simple)

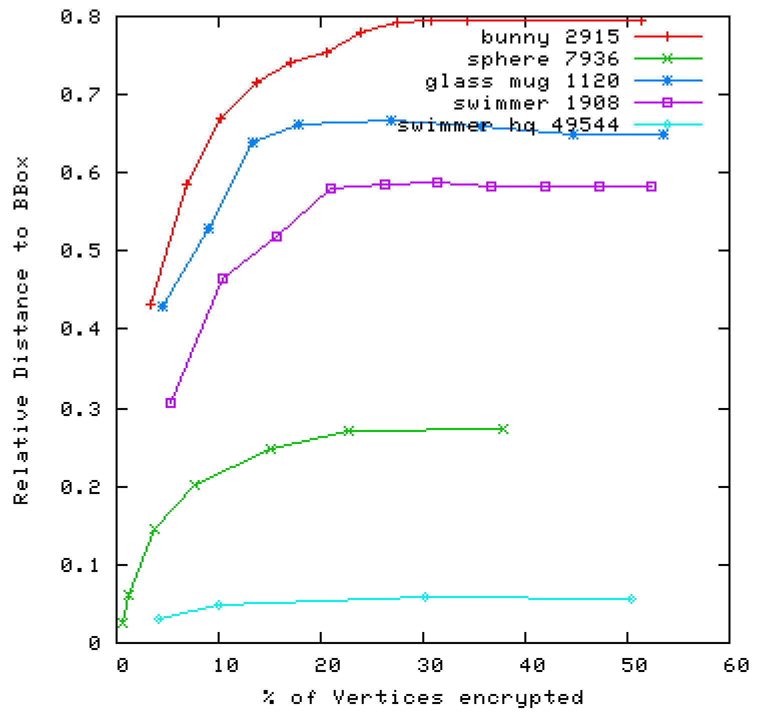
- Averaging the direct neighbors to get the predicted position
- The better the predictor during compression the smaller the error coefficient. This means that we need to encrypt very much coefficients to get “good” results

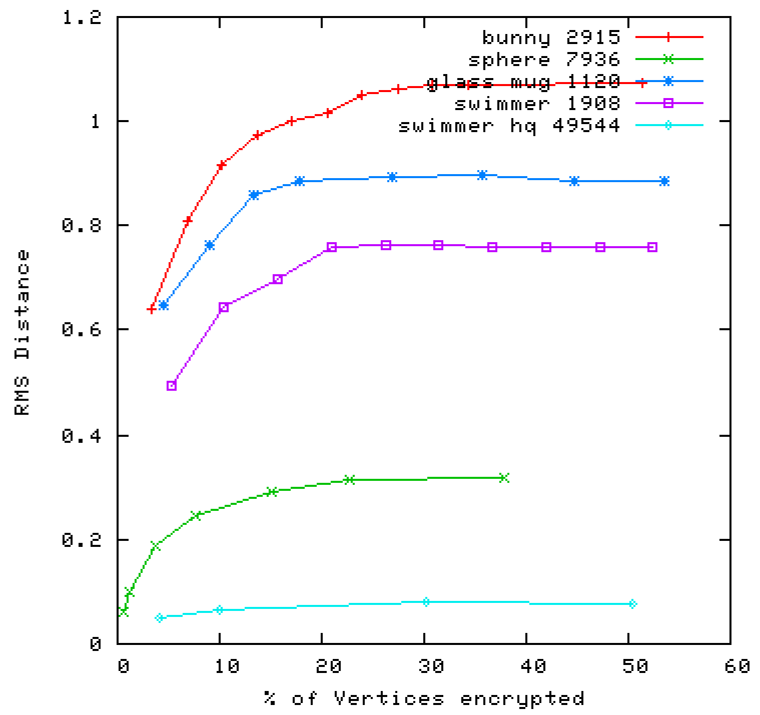




Encrypting the differences to the predictor (butterfly)

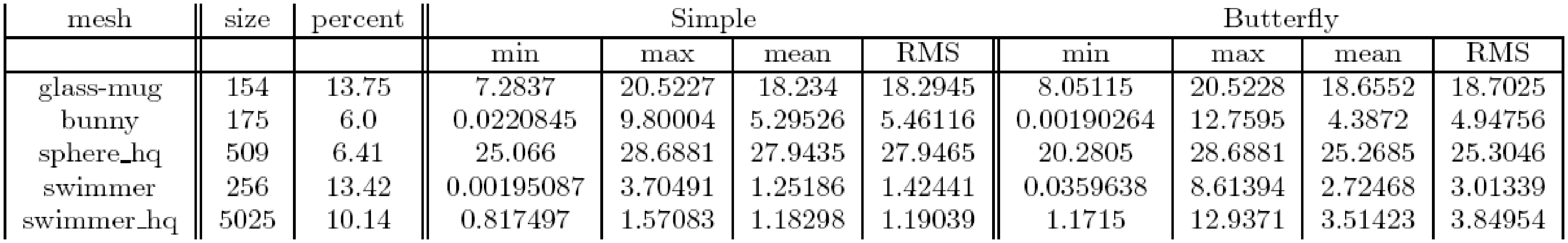
- Slightly better than the simple prediction





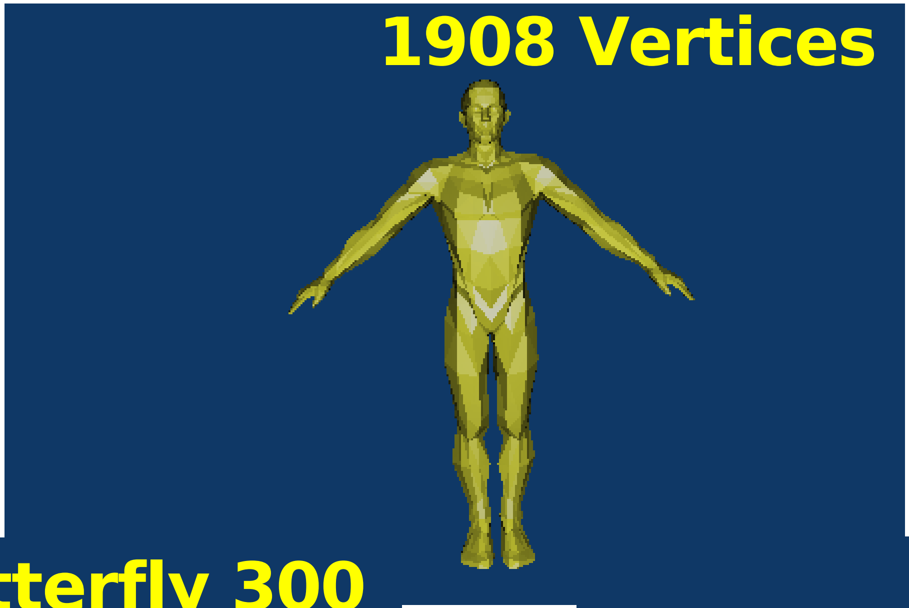
Encrypting the Base Mesh

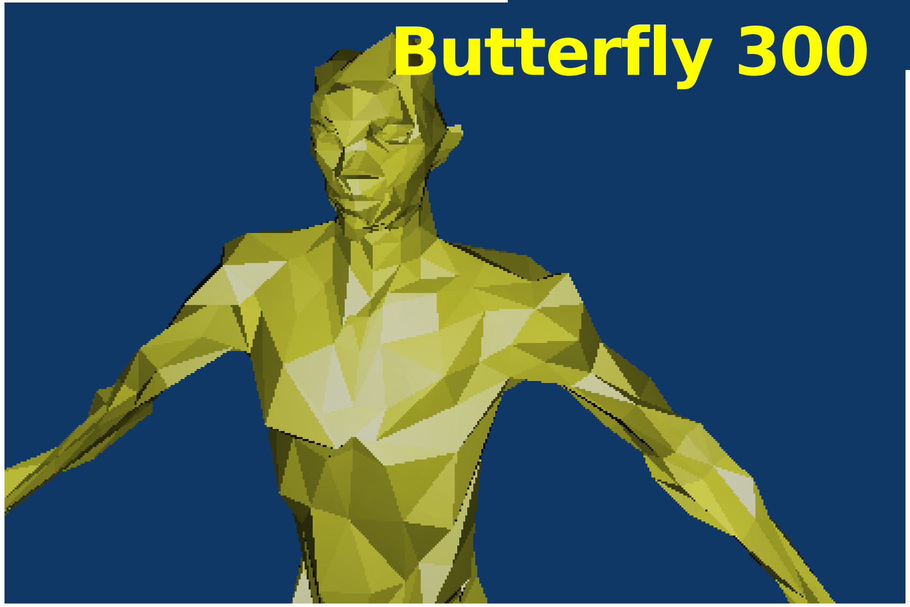
- Very effective as every refinement step is based either directly or indirectly on the base mesh (which must not even be true for very low detail data that is not part of the base mesh)

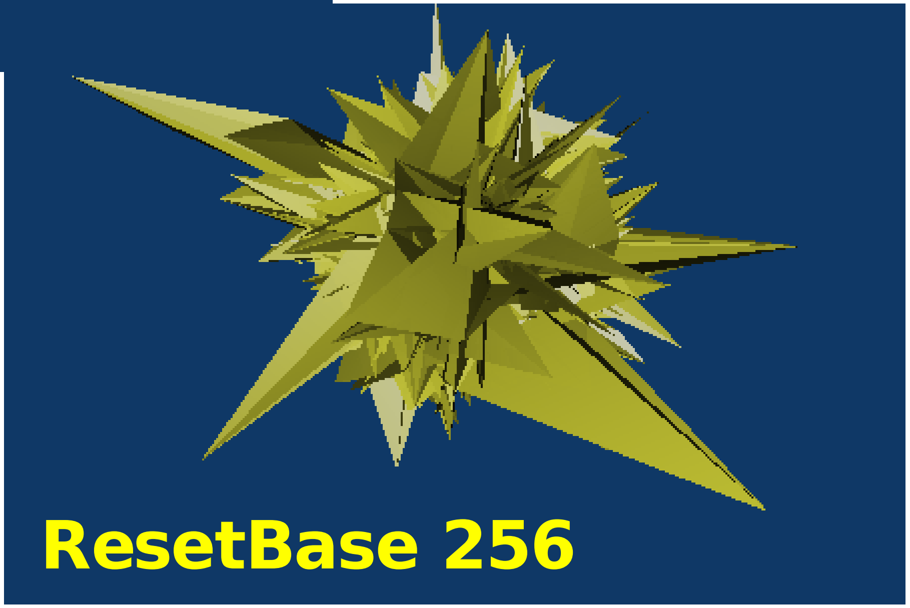


mesh	size	percent	Simple				Butterfly			
			min	max	mean	RMS	min	max	mean	RMS
glass-mug	154	13.75	7.2837	20.5227	18.234	18.2945	8.05115	20.5228	18.6552	18.7025
bunny	175	6.0	0.0220845	9.80004	5.29526	5.46116	0.00190264	12.7595	4.3872	4.94756
sphere_hq	509	6.41	25.066	28.6881	27.9435	27.9465	20.2805	28.6881	25.2685	25.3046
swimmer	256	13.42	0.00195087	3.70491	1.25186	1.42441	0.0359638	8.61394	2.72468	3.01339
swimmer_hq	5025	10.14	0.817497	1.57083	1.18298	1.19039	1.1715	12.9371	3.51423	3.84954

Encrypting the Base Mesh







Conclusion

- The connectivity information without the geometry information is useless
- If the connectivity information is missing it depends on the type of geometry information if and how much data can be recovered
  - If the geometry information is stored as the difference to a predicted value, the data is useless
  - If the geometry information is stored as absolute values (and the points are densely sampled) the mesh can be reconstructed to a certain point

Conclusion

- The refinement data without the base mesh is useless if it is based on some predictor function (entropy coded errors)
- Those base meshes are normally only about 5%-10% of the size of the full mesh
- Thus the amount of data that needs to be encrypted to protect the mesh is reduced to 5%-10%

End

Thank you for your attention

Questions?

Copyrights