View-Dependent Refinement of Progressive Meshes

Hughes Hoppe
Microsoft Research

Author

- Hughes Hoppe

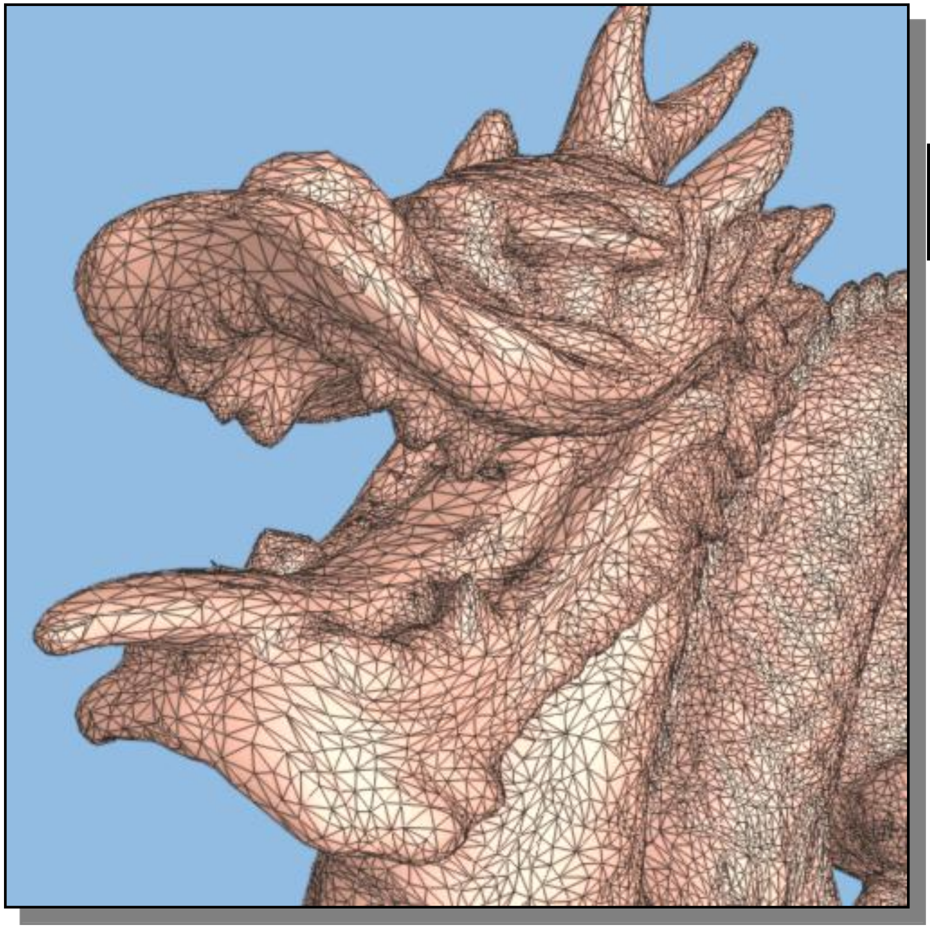
Outline

- Introduction
- Selective Refinement Framework
- Refinement Criteria
- Algorithm
- Rendering
- Result

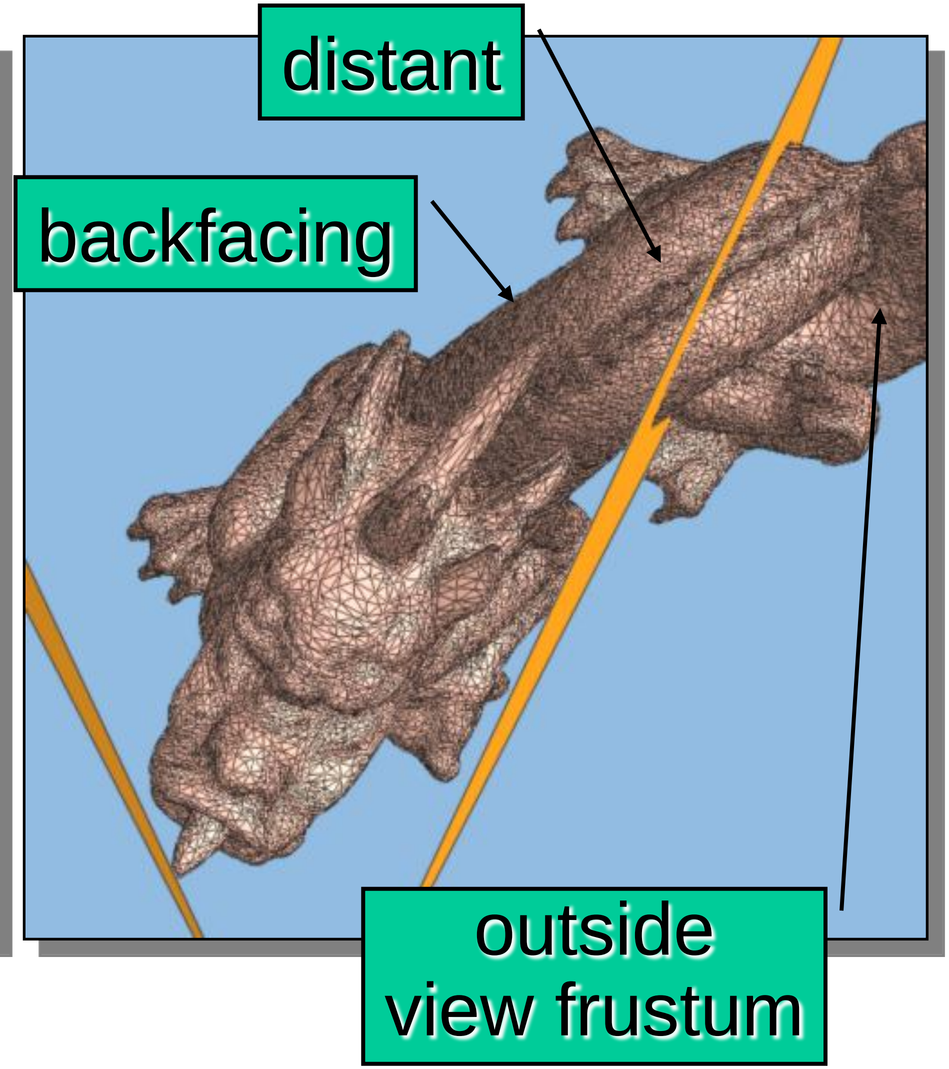
Introduction

- Presenting a framework for real-time selective refinement of arbitrary progressive meshes.

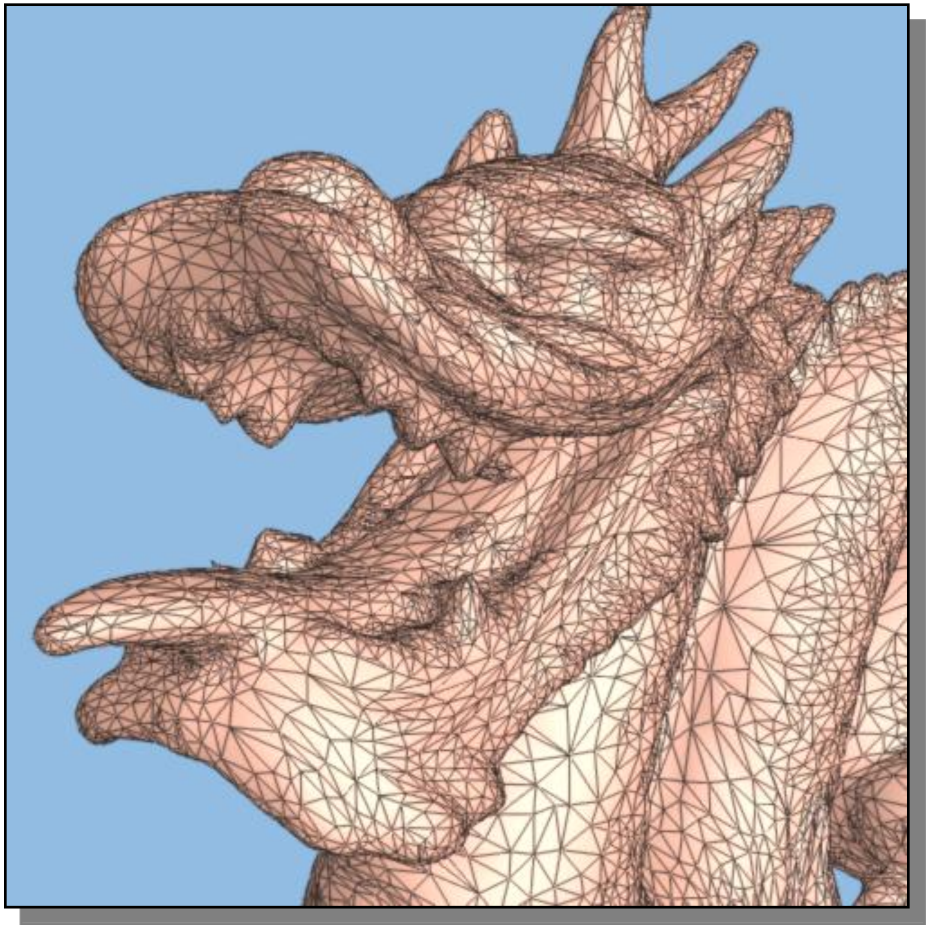
View-Independent LOD



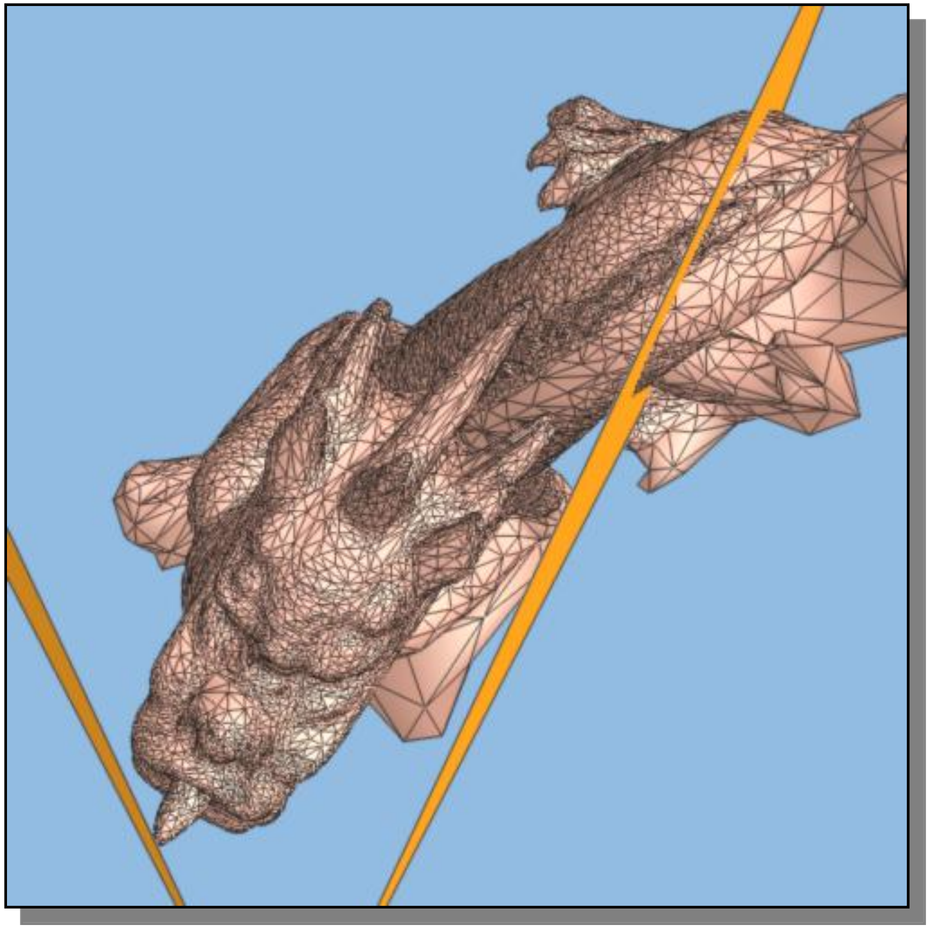
100,000 faces



View-dependent LOD



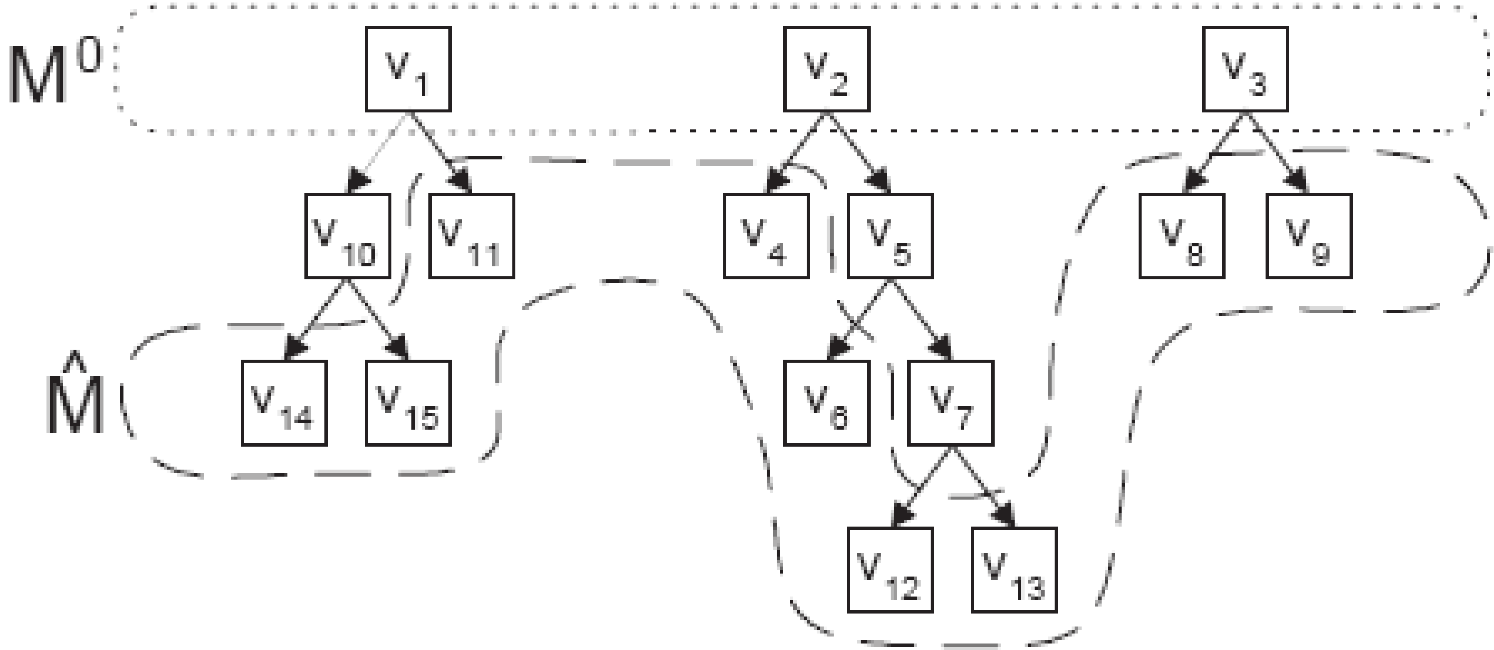
29,400 faces



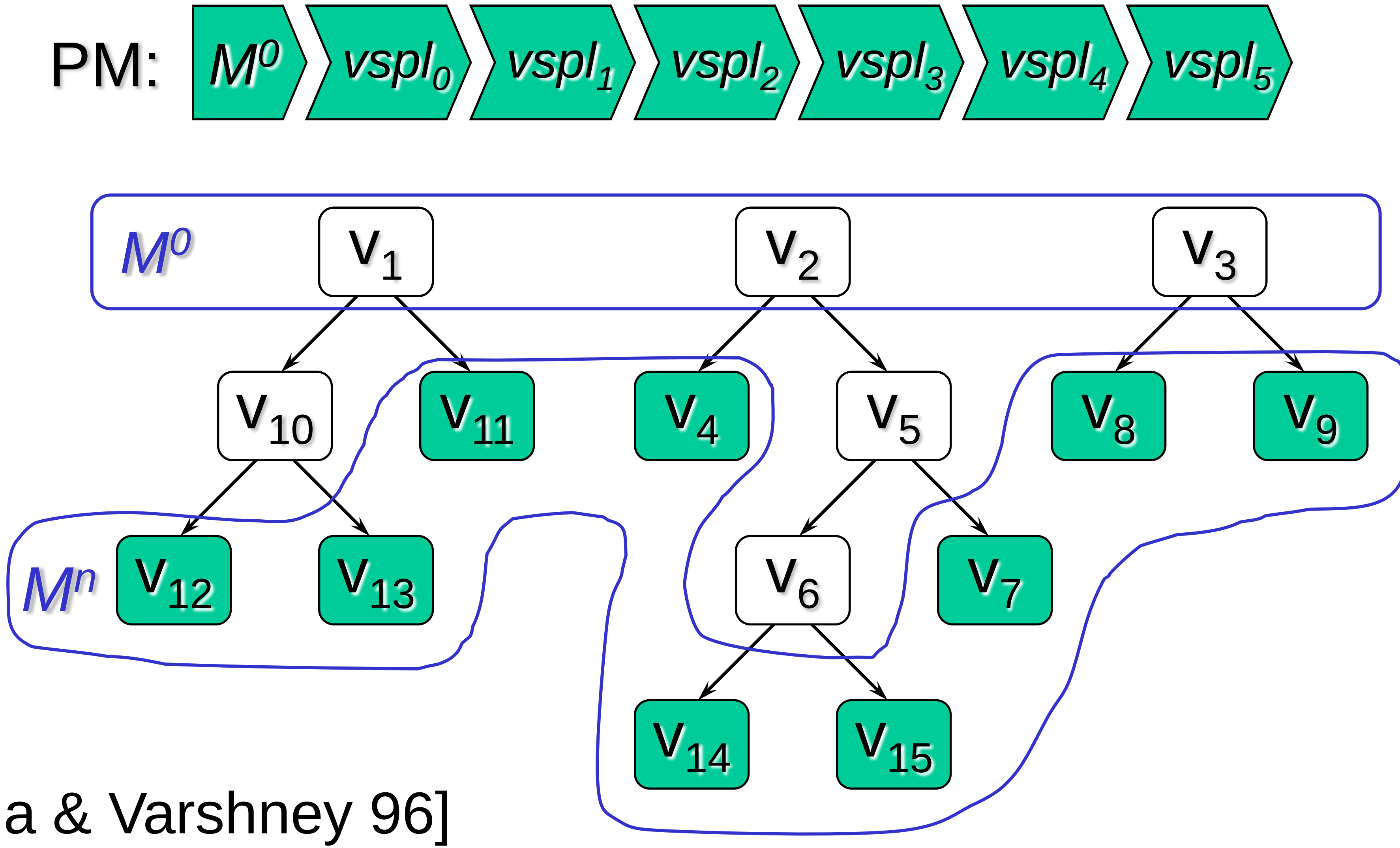
different LOD’s coexist over surface

Selective Refinement Framework

- Vertex hierarchies

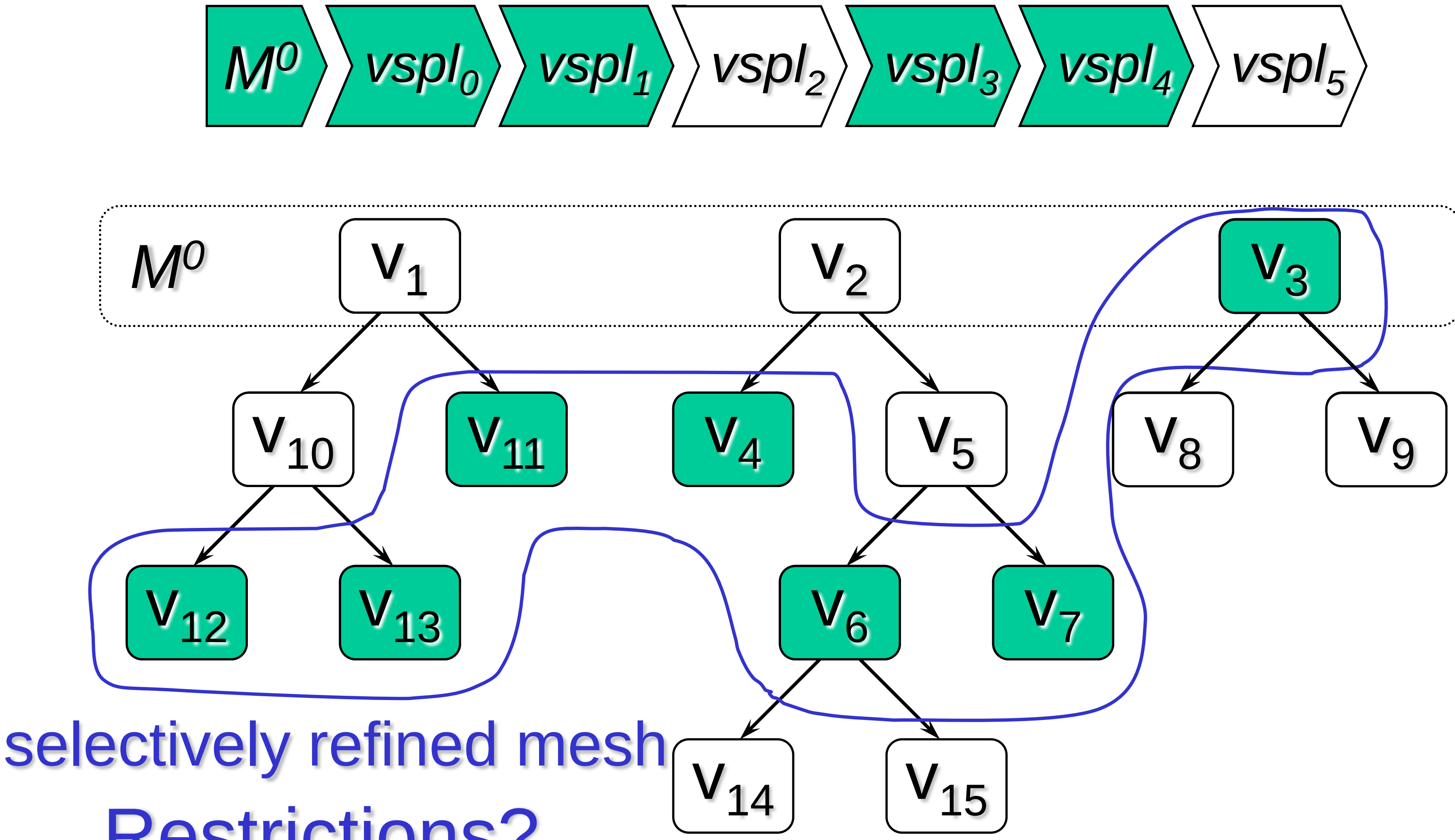


Vertex hierarchy



[Xia & Varshney 96]

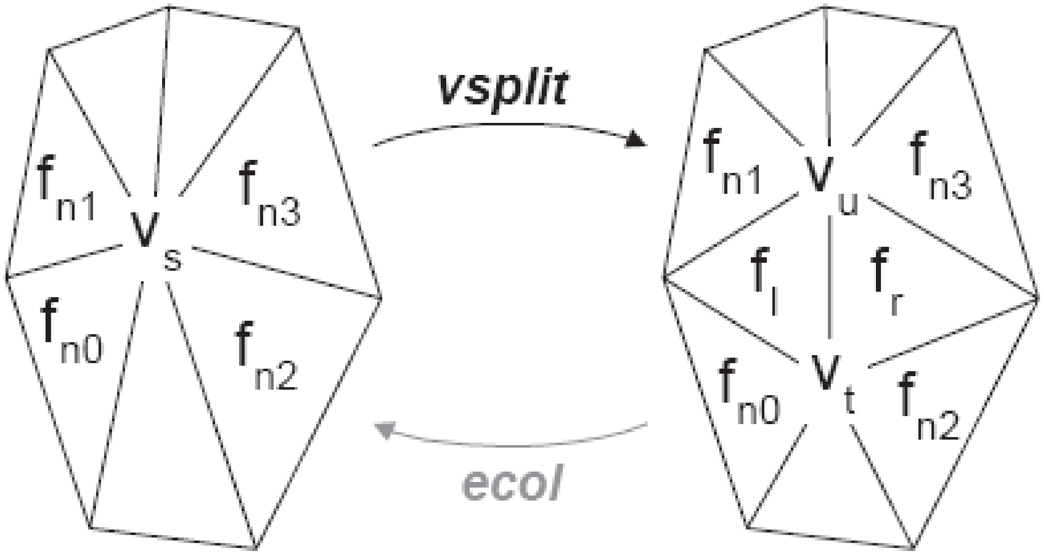
Selective refinement



Restrictions?

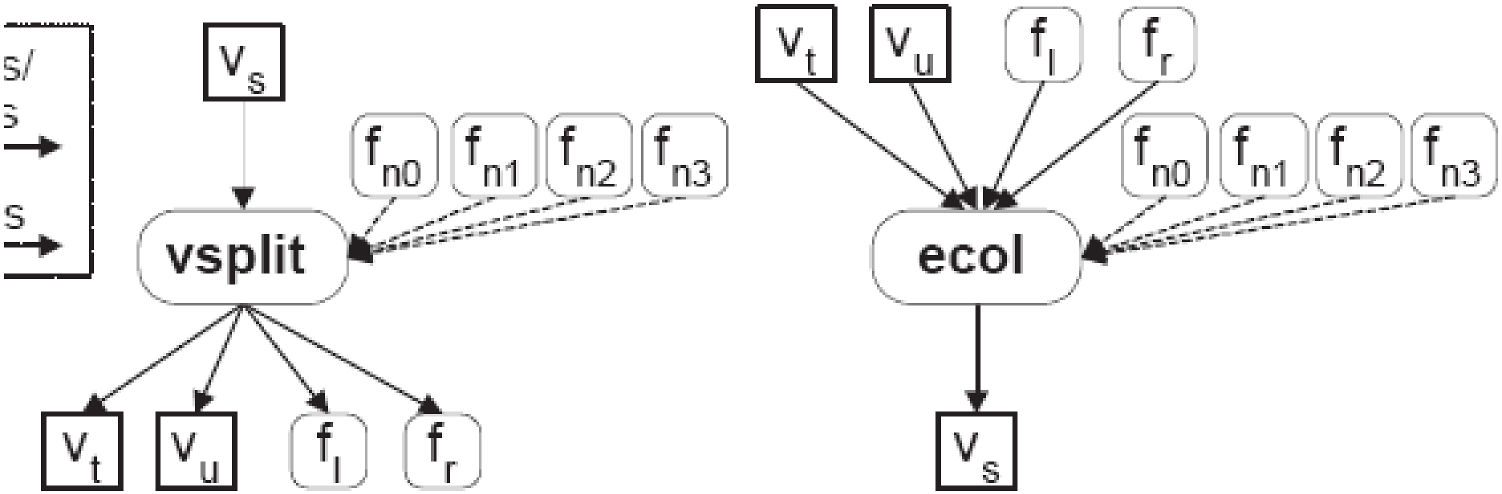
Selective Refinement Framework

$\mathcal{M}$:The set of all meshes M^s produced from M^0 by a subsequence S of legal vsplit transformations.



Selective Refinement Framework

- We define a set of preconditions for *vsplit* and *ecol* to be legal.



Selective Refinement Framework

```
struct ListNode {                // Node possibly on a linked list
    ListNode* next;              // 0 if this node is not on the list
    ListNode* prev;
};
struct Vertex {
    ListNode active;             // list stringing active vertices V
    Point point;
    Vector normal;
    Vertex* parent;              // 0 if this vertex is in M^0
    Vertex* vt;                  // 0 if this vertex is in M̂; (vu=vt+1)
  // Remaining fields encode vsplit information, defined if vt ≠ 0.
    Face* fl;                    // (fr=fl+1)
    Face* fn[4];                 // required neighbors fn0,fn1,fn2,fn3
    RefineInfo refine_info;      // defined in Section 4
};
```

Selective Refinement Framework

```
struct Face {
    ListNode active;                 // list stringing active faces F
    int matid;                       // material identifier
  // Remaining fields are used if the face is active.
    Vertex* vertices[3];             // ordered counter-clockwise
    Face* neighbors[3];              // neighbors[i] across from vertices[i]
};
struct SRMesh {                      // Selectively refinable mesh
    Array<Vertex> vertices;          // set 𝒱 of all vertices
    Array<Face> faces;               // set F̂ of all faces
    ListNode active_vertices;        // head of list V ⊆ 𝒱
    ListNode active_faces;           // head of list F ⊆ F̂
};
```

Refinement criteria

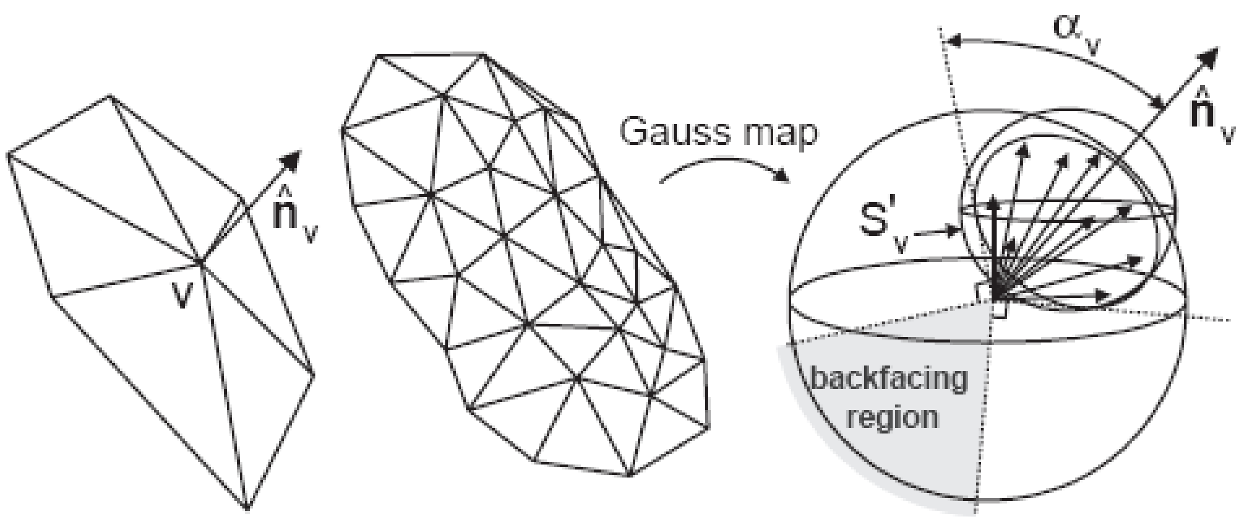
function qrefine(v_s)
  // Refine only if it affects the surface within the view frustum.
  if outside_view_frustum(v_s) **return** *false*
  // Refine only if part of the affected surface faces the viewer.
  if oriented_away(v_s) **return** *false*
  // Refine only if screen-projected error exceeds tolerance τ.
  if screen_space_error(v_s) $\leq \tau$ **return** *false*
  return *true*

Refinement criteria

- View frustum
- Use bounding sphere

Refinement criteria

- Surface orientation
- Consider the space of normals



(a) N_v (b) region of $\hat{M}$ (c) S^2

Refinement criteria

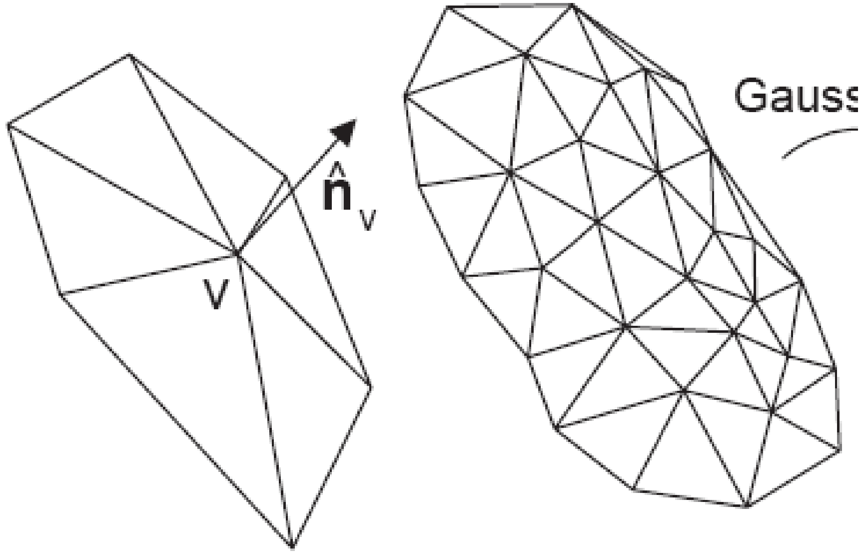
$$\frac{\mathbf{a}_v - \mathbf{e}}{\|\mathbf{a}_v - \mathbf{e}\|} \cdot \hat{\mathbf{n}}_v > \sin \alpha_v$$

$$\frac{a_v - e}{\|a_v - e\|} \cdot \hat{n}_v > \cos(\frac{\pi}{2} - \alpha_v) = \sin \alpha_v$$

$$(\mathbf{v} - \mathbf{e}) \cdot \hat{\mathbf{n}}_v > 0 \quad \textbf{and} \quad ((\mathbf{v} - \mathbf{e}) \cdot \hat{\mathbf{n}}_v)^2 > \|\mathbf{v} - \mathbf{e}\|^2 \sin^2 \alpha_v$$

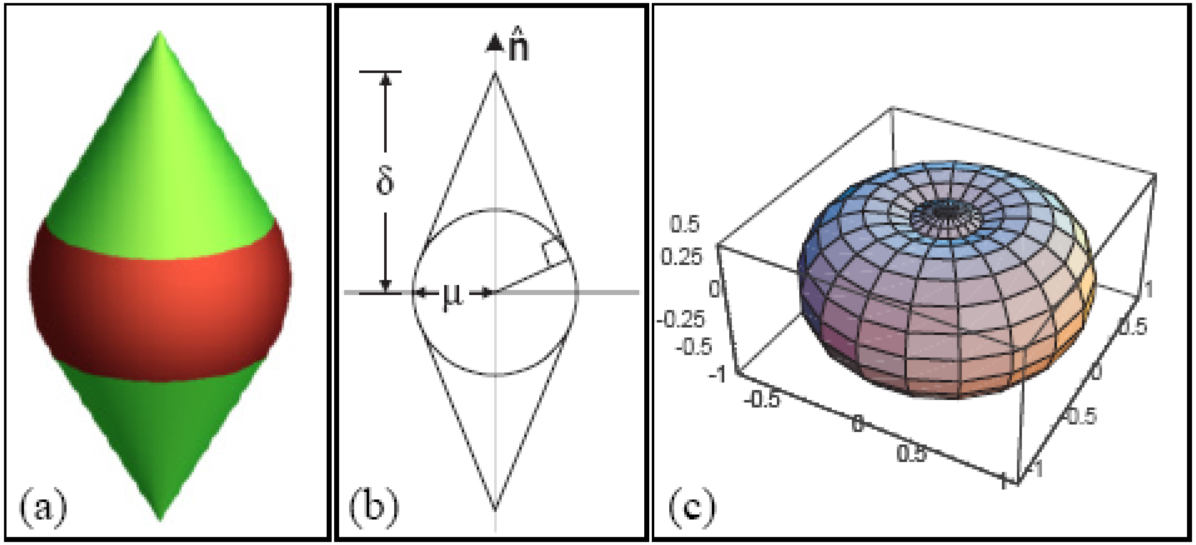
Refinement criteria

- Screen-space geometric error



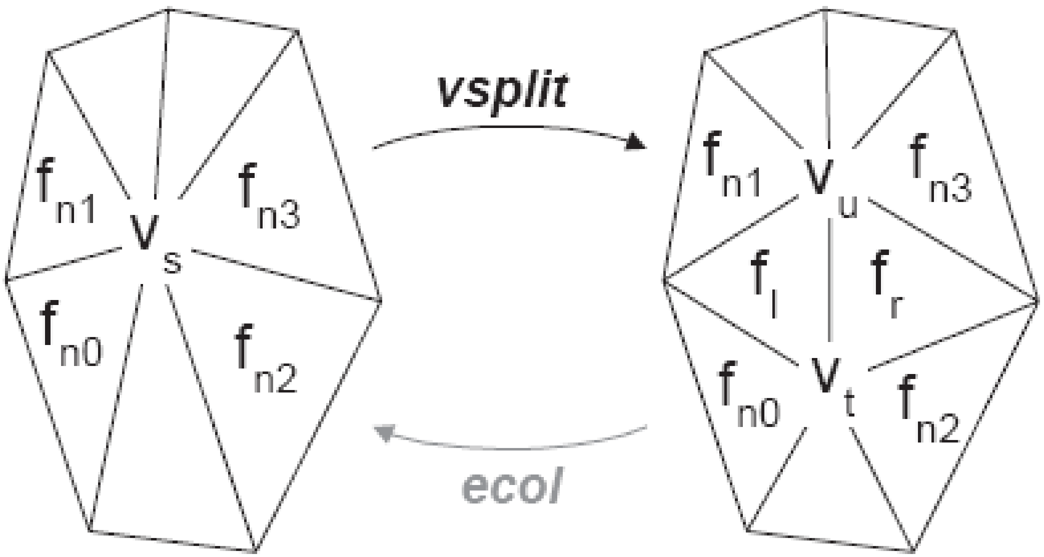
(a) N_v (b) region of $\hat{M}$

Refinement criteria



```
procedure adapt_refinement()
    for each v ∈ V
        if v.vt and qrefine(v)
            force_vsplit(v)
        else if v.parent and ecol_legal(v.parent) and
                not qrefine(v.parent)
            ecol(v.parent)      // (and reconsider some vertices)
procedure force_vsplit(v′) {
    stack ← v′
    while v ← stack.top()
        if v.vt and v.fl ∈ F
            stack.pop()         // v was split earlier in the loop
        else if v ∉ V
            stack.push(v.parent)
        else if vsplit_legal(v)
            stack.pop()
            vsplit(v)           // (placing v.vt and v.vu next in list V)
        else for i ∈ {0 . . . 3}
            if v.fn[i] ∉ F
                // force vsplit that creates face v.fn[i]
                stack.push(v.fn[i].vertices[0].parent)
```
3

Algorithm



Algorithm

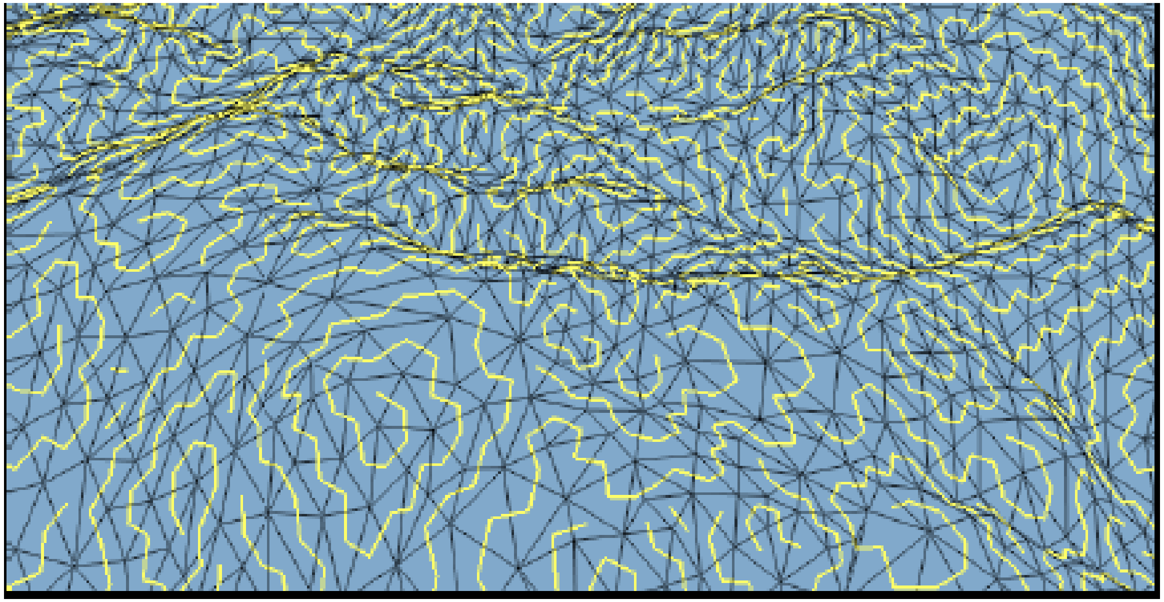
- The number of active faces can vary dramatically depending on the view.
- We can change tolerance τ

$$\tau_t = \tau_{t-1}(|F_{t-1}|/m)$$

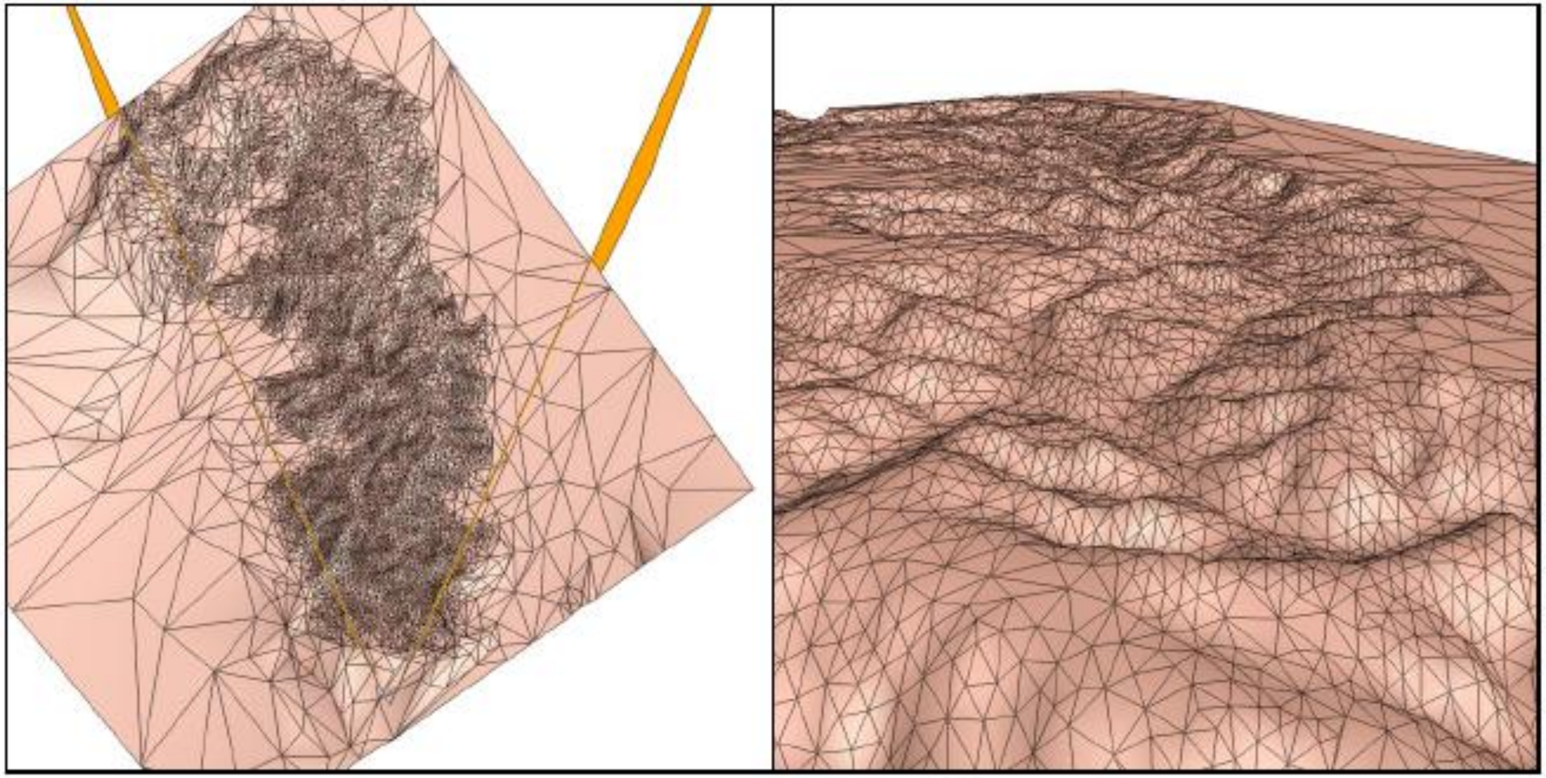
- m: desired number of faces.

Rendering

- Use triangle strip to achieve optimal rendering performance.

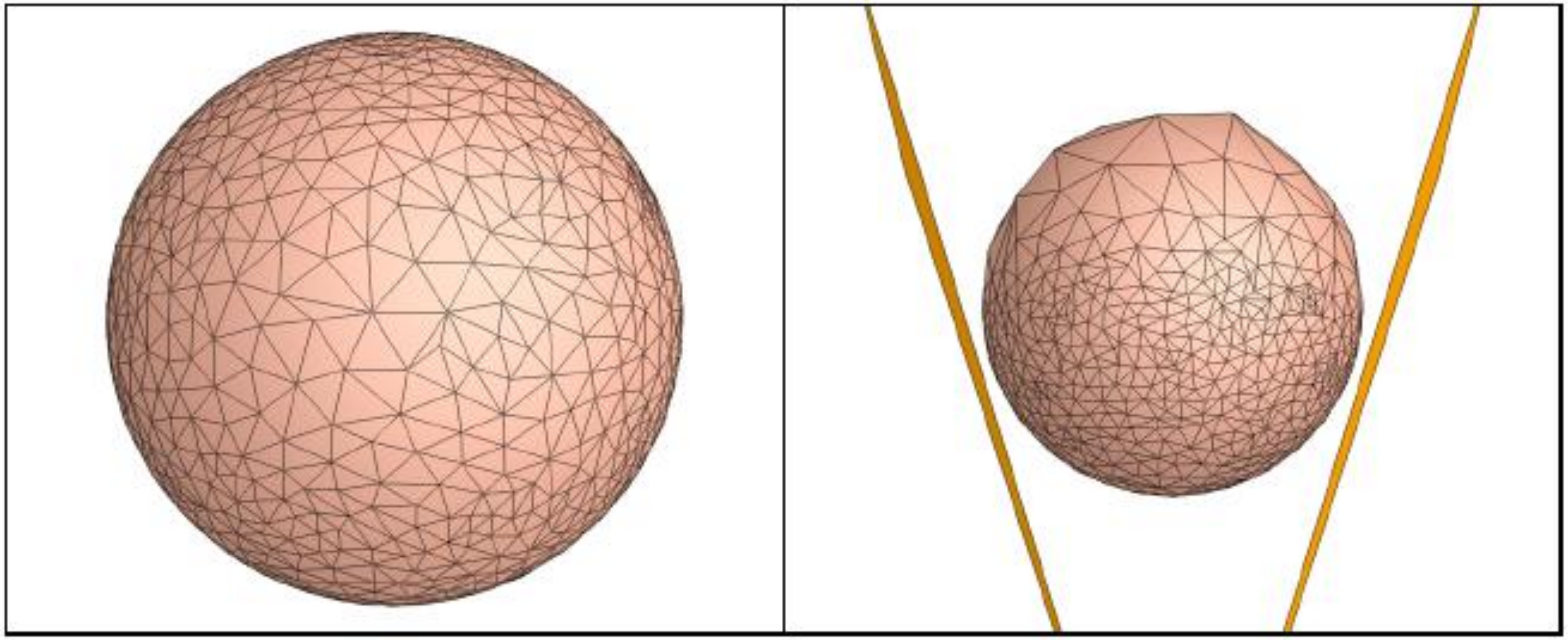


Result



(b) Top and regular views (τ=0.33%; 10,013 faces)

Result



(b) Front view and (c) Top view ($\tau = 0.075\%$; 1,422 faces)