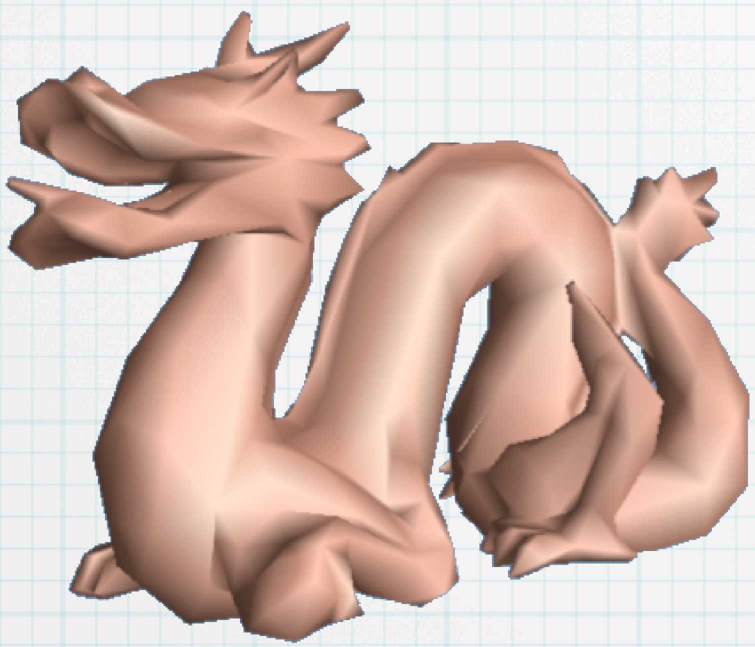
View-Dependent Refinement of Progressive Meshes

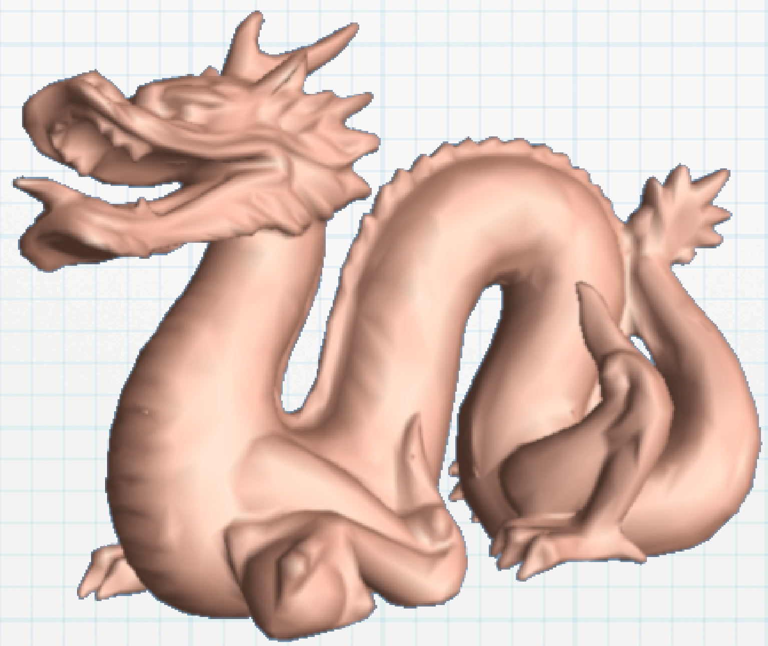
Hugues Hoppe
Microsoft Research
SIGGRAPH 1997

presented by Peter Bogatsky

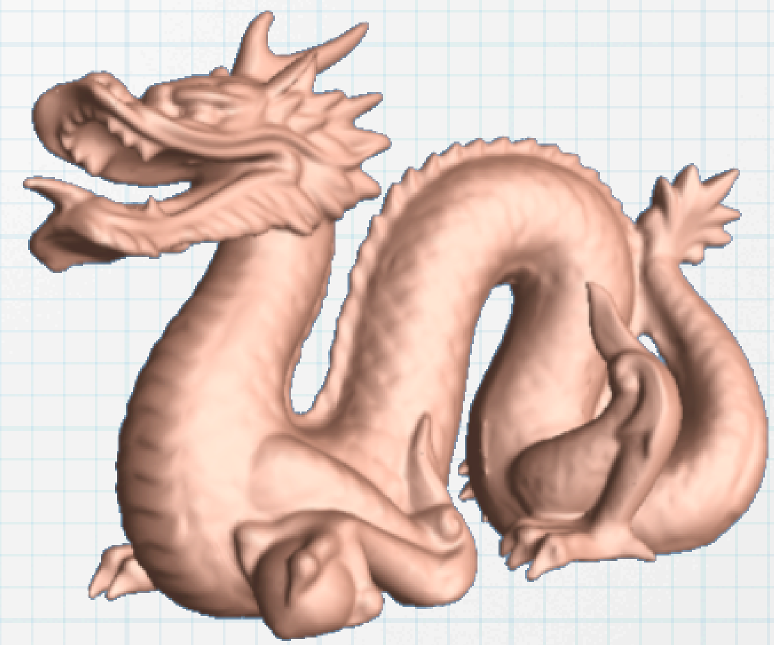
Traditional 'Level of Detail'



2,000 faces



10,000 faces

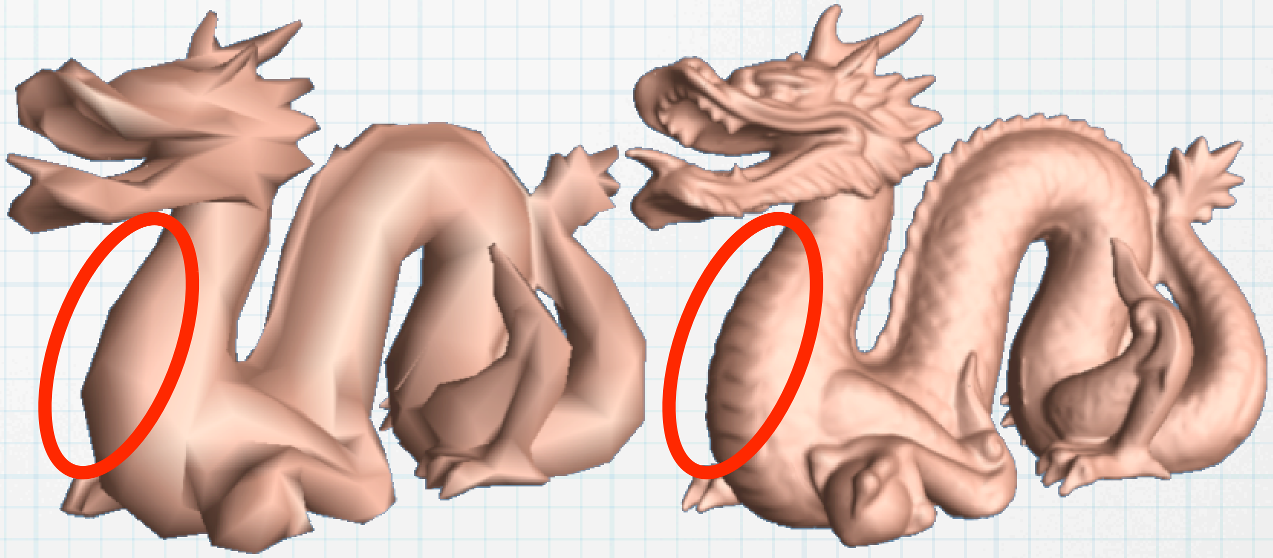


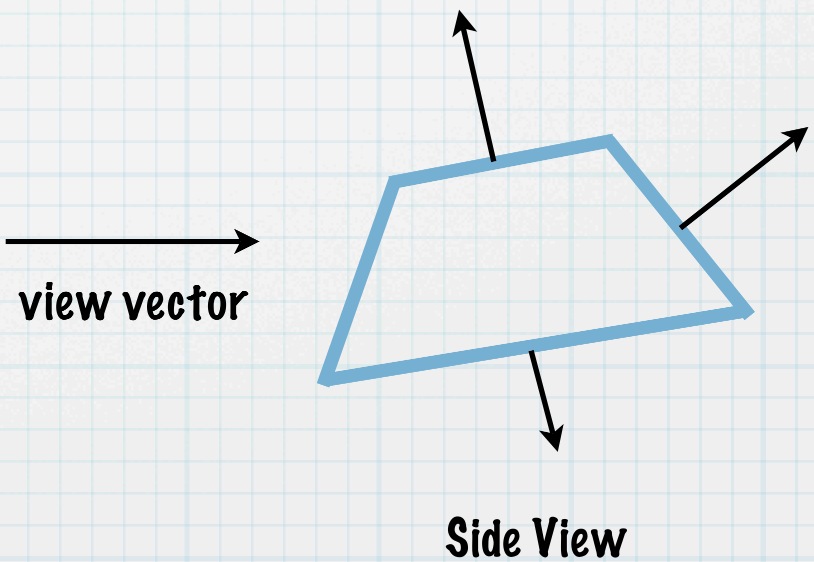
50,000 faces

Entire meshes are uniformly simplified.

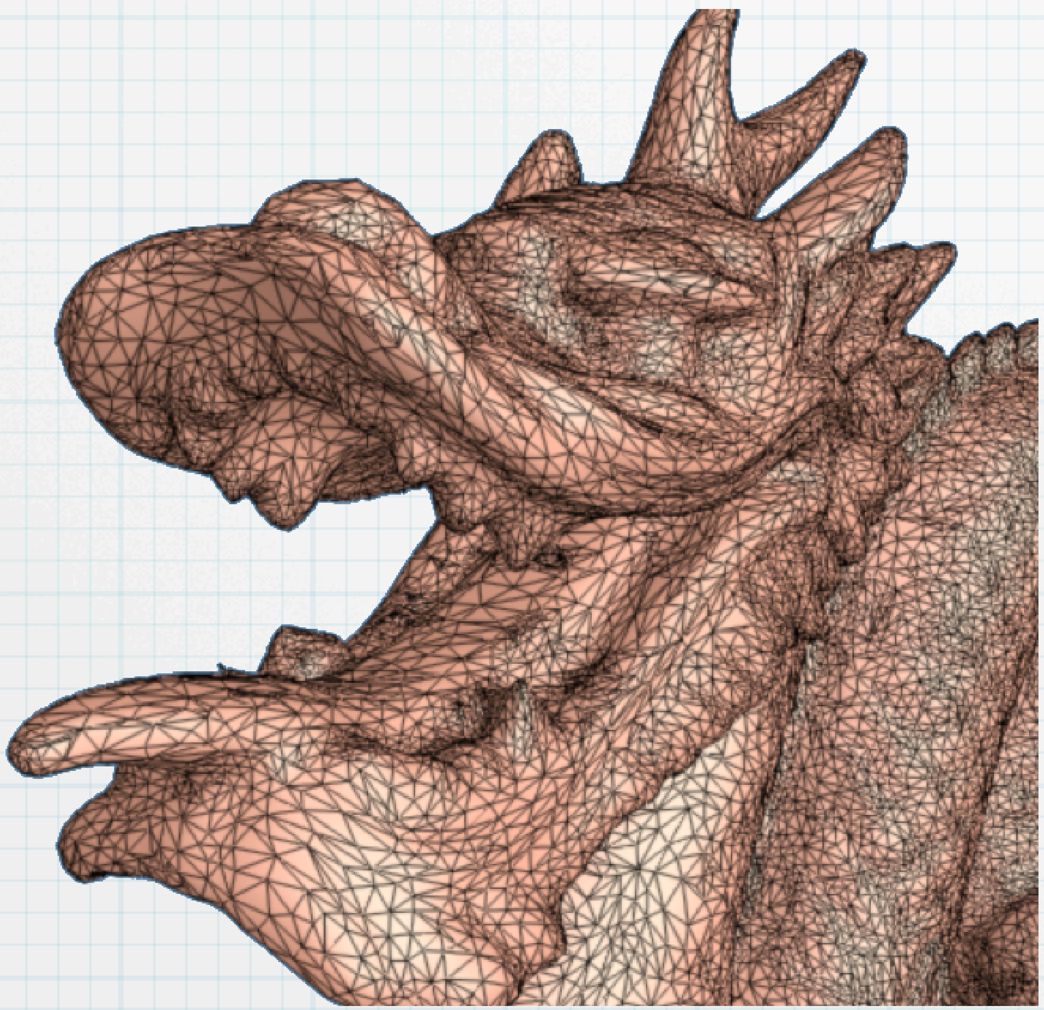
Problems with LoD

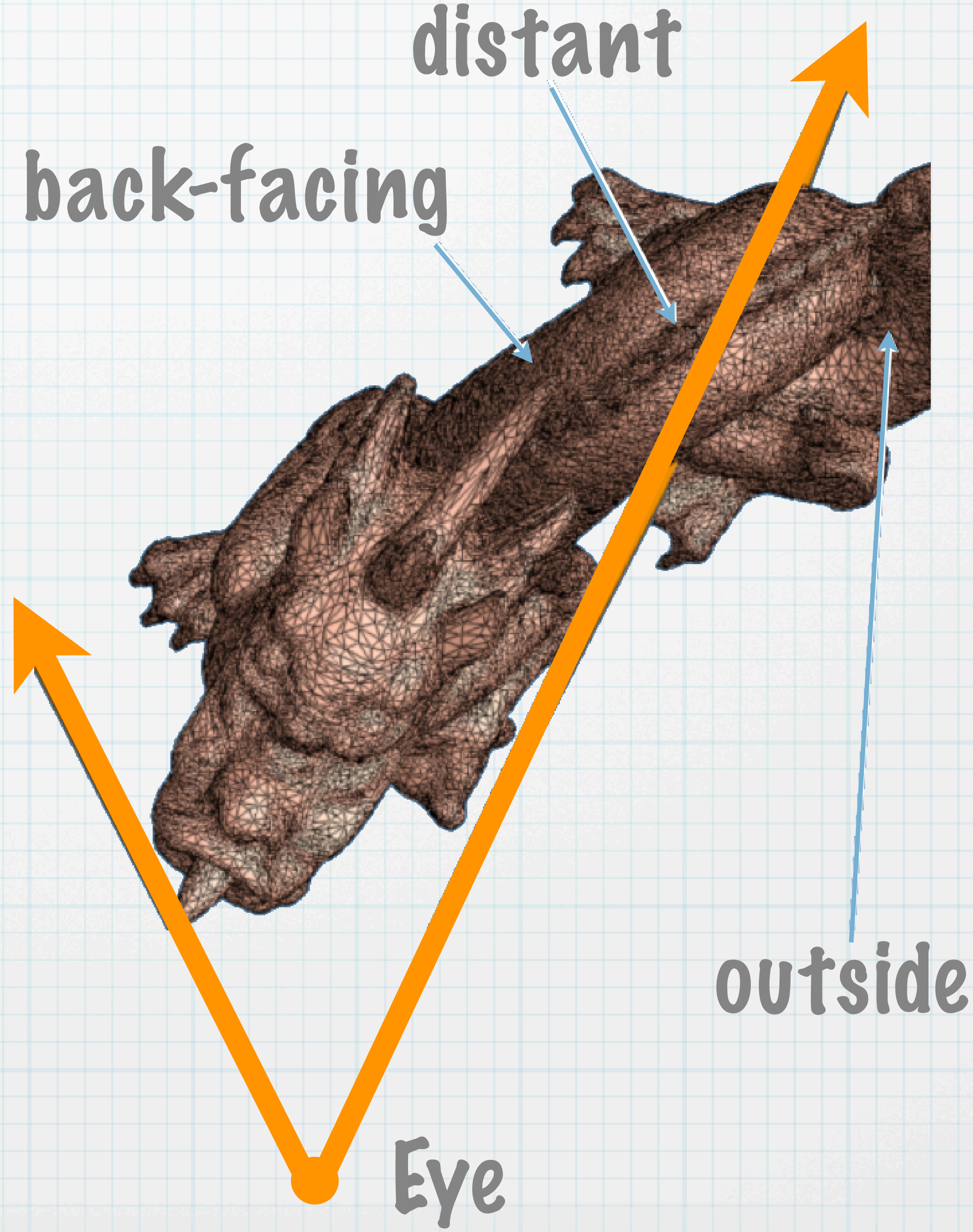
- Appearance
  - Silhouettes
- Efficiency
  - Non-visible faces
    - Back-facing
    - Off-screen



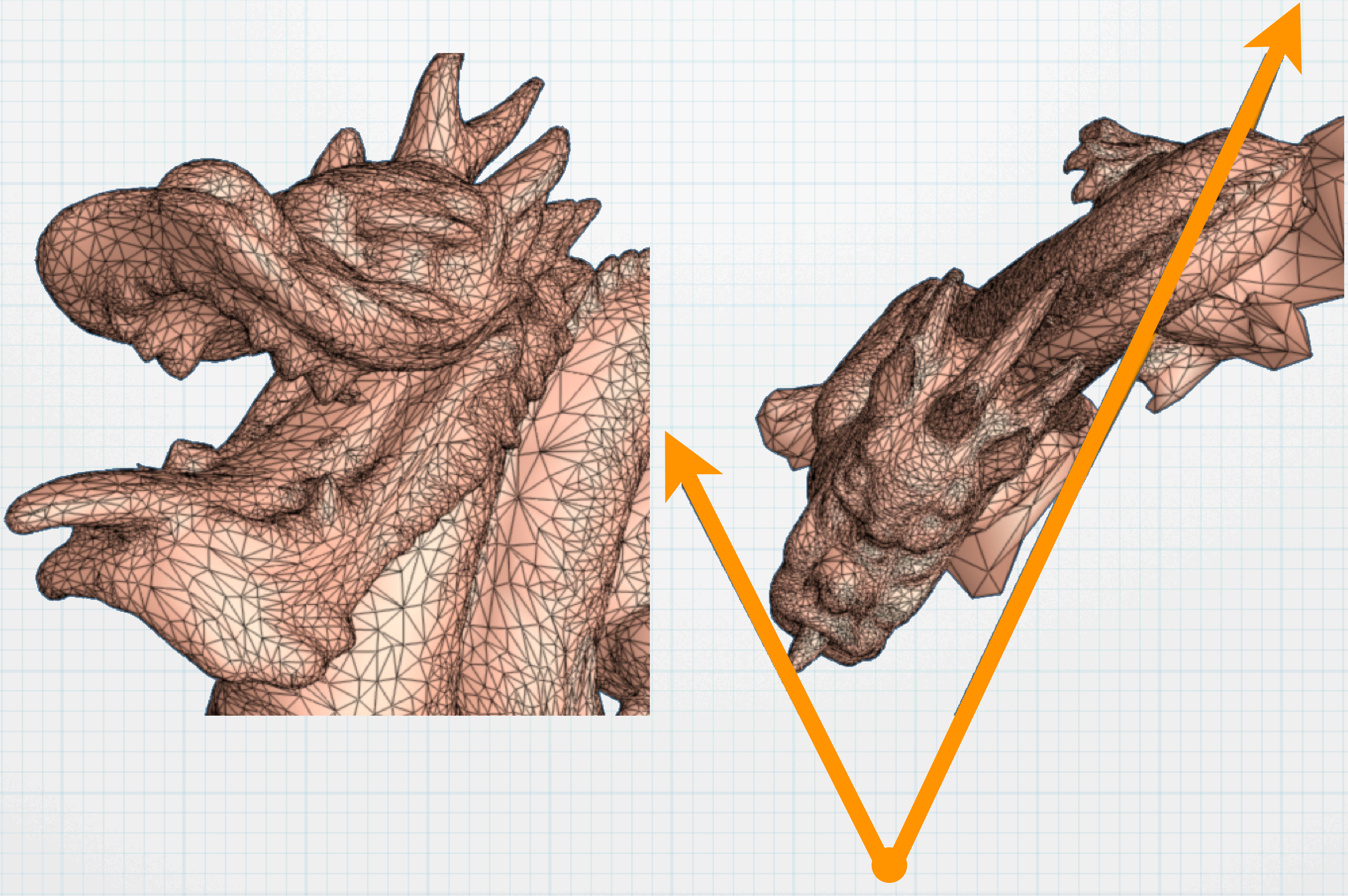


View-Dependent LoD



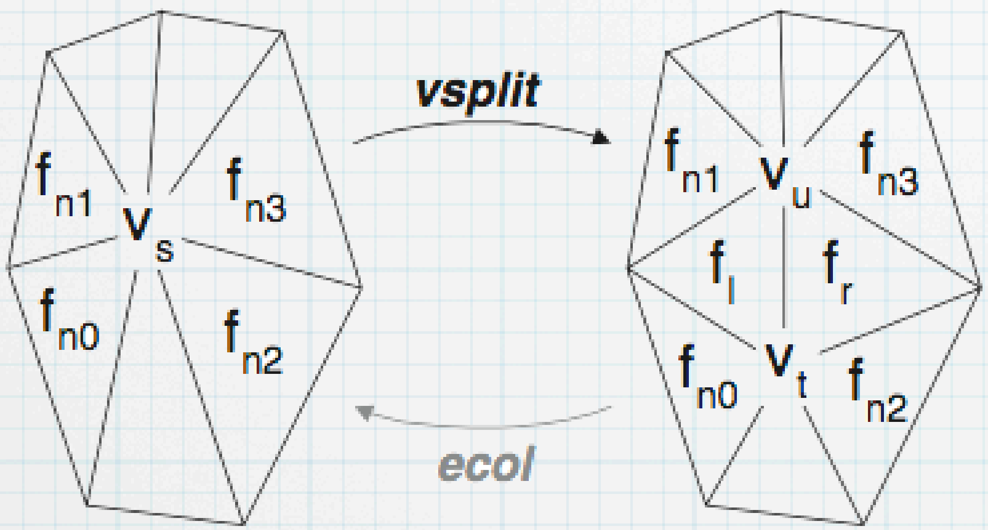


View-Dependent LoD



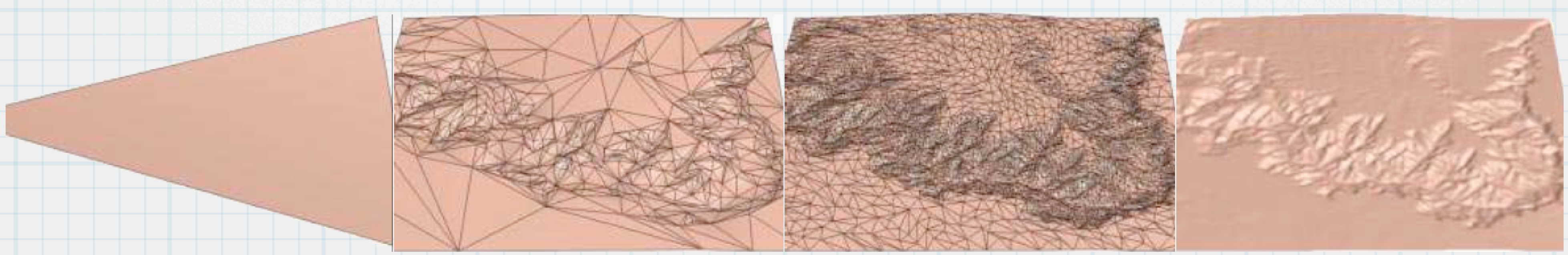
Progressive Meshes

* Mesh M is coarsened by sequence of edge collapses
* Edge collapses are reversible using vertex splits



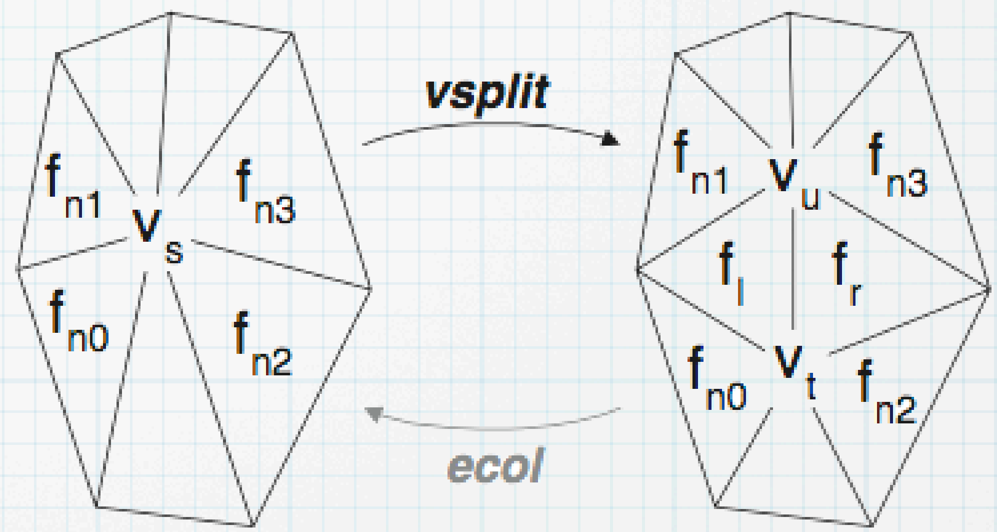
$$M^0 \xrightarrow{vsplit_0} M^1 \xrightarrow{vsplit_1} \cdots \xrightarrow{vsplit_{n-1}} (M^n = \hat{M})$$

$$(\hat{M} = M^n) \xrightarrow{ecol_{n-1}} \cdots \xrightarrow{ecol_1} M^1 \xrightarrow{ecol_0} M^0$$



M^0 M^{514} M^{5066} ... M^n

Progressive Meshes

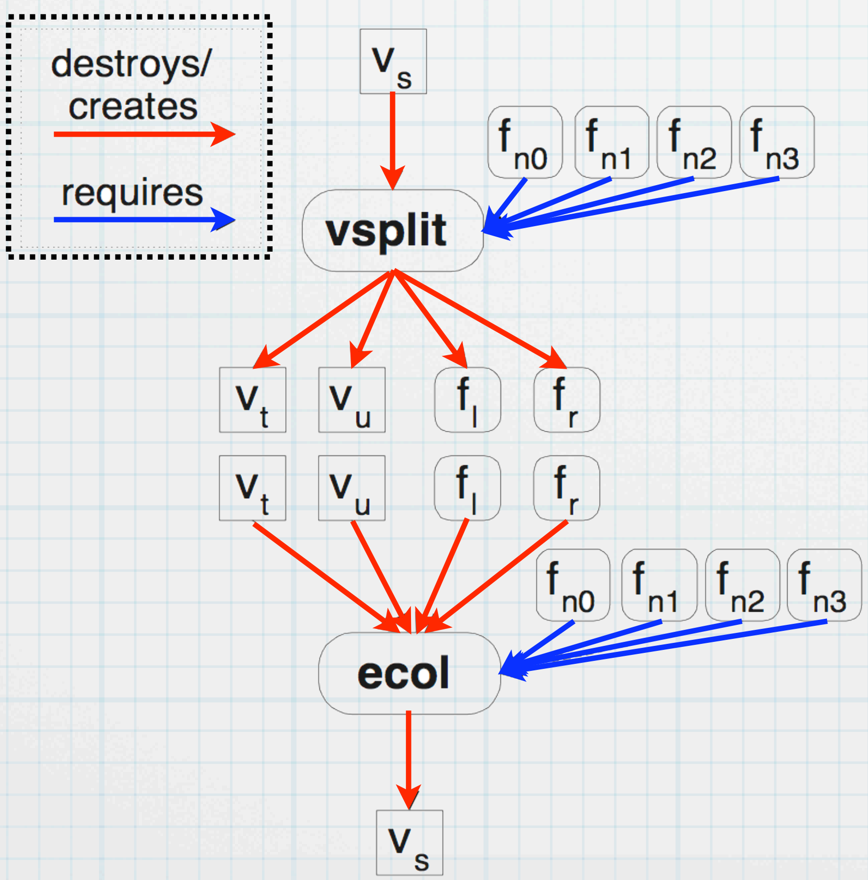


vsplit($v_s, v_t, v_u, f_l, f_r, f_{n0}, f_{n1}, f_{n2}, f_{n3}$)

* replaces 'parent' v_s with 'children' v_t and v_u.
* creates faces f_l and f_r.

ecol($v_s, v_t, v_u, f_l, f_r, f_{n0}, f_{n1}, f_{n2}, f_{n3}$)

* replaces 'children' v_t and v_u with 'parent' v_s.
* removes faces f_l and f_r.



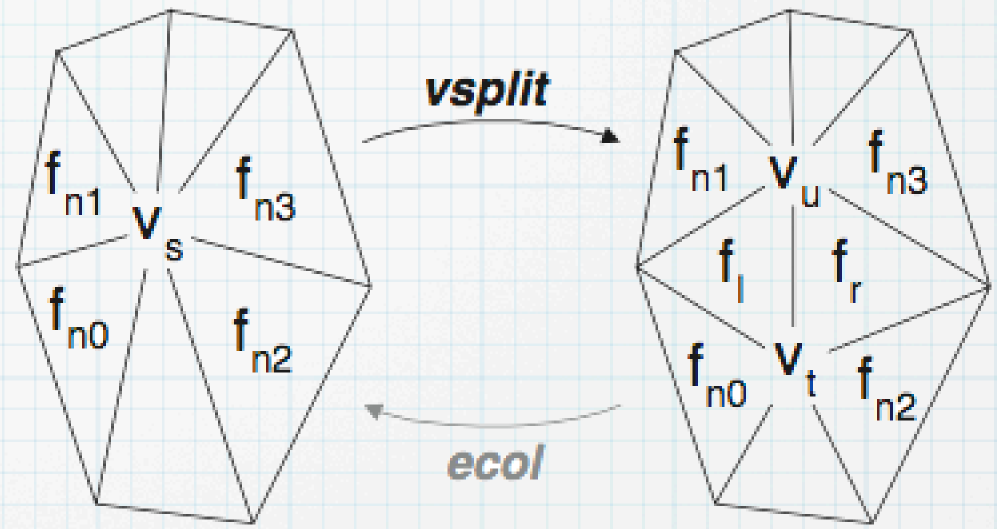
Pre-conditions

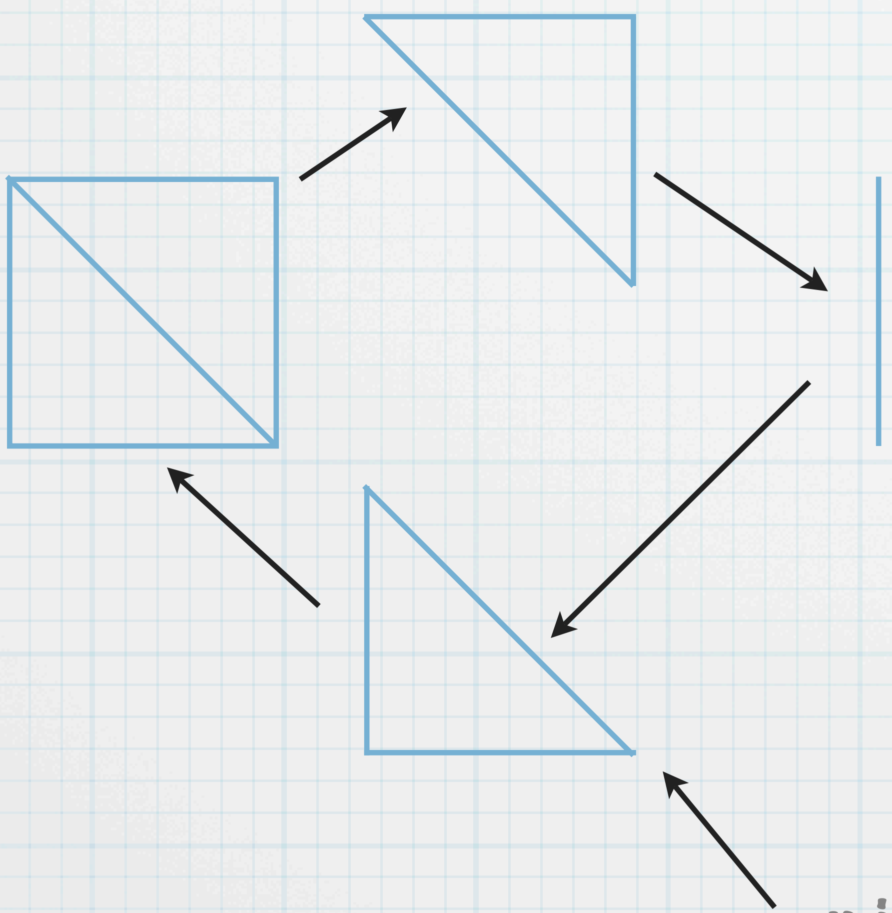
* vsplit
  * v_s is an active vertex.
  * f_{n0}, f_{n1}, f_{n2}, and f_{n3} are active faces
* ecol
  * v_t and v_u are active vertices
  * the adjacent faces to f_l and f_r are {f_{n0}, f_{n1}} and {f_{n2}, and f_{n3}}, respectively.

Progressive Meshes

$vsplit(v_s, v_t, v_u, f_l, f_r, f_{n0}, f_{n1}, f_{n2}, f_{n3})$

- replaces 'parent' v_s with 'children' v_t and v_u.
- creates faces f_l and f_r.



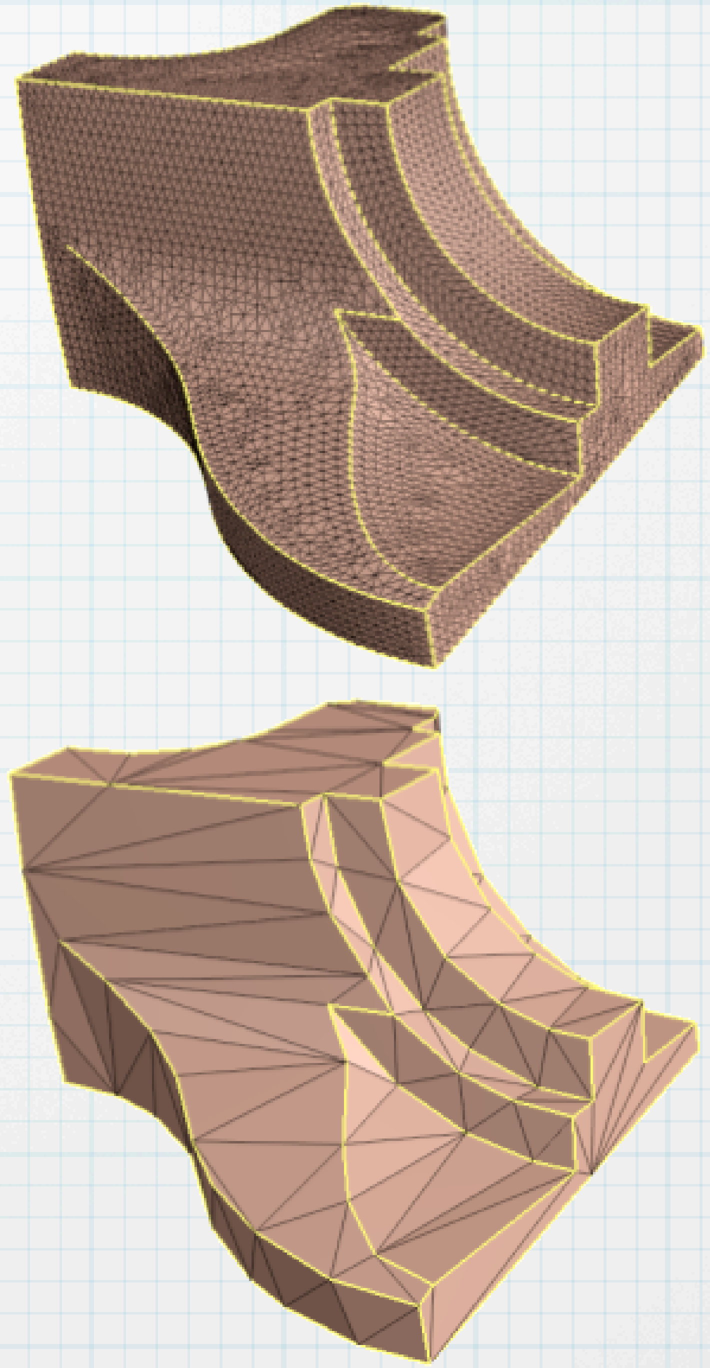


might not have
existed in original PM

Using triangle numbers versus vertex numbers removes some dependency constraints. This allows more freedom in intermediate meshes.

PM Construction

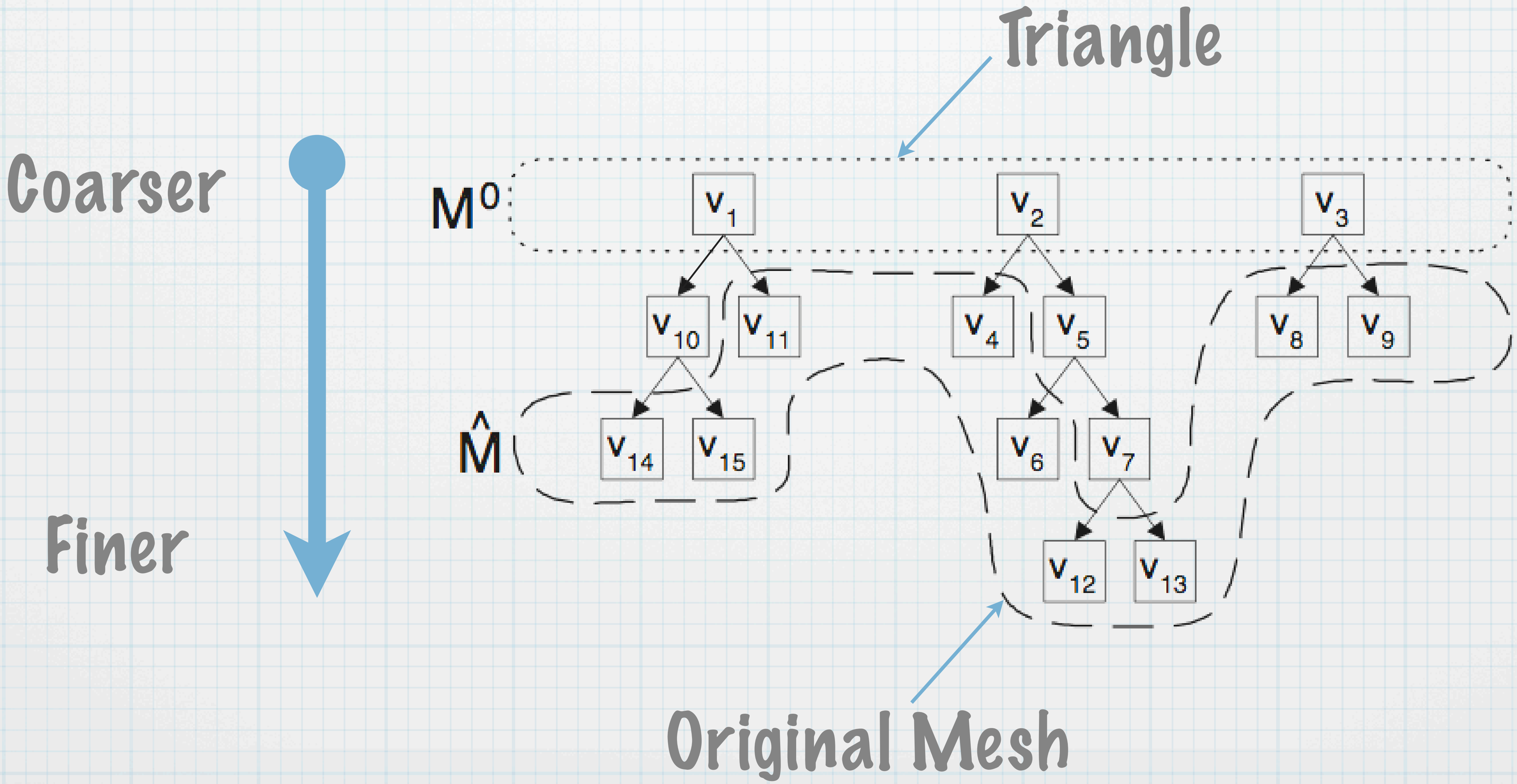
- Performed only once (ie. pre-process)
- Quality of intermediate meshes depends directly on the order of chosen edge collapses. Methods include:
    - Choosing edges randomly
    - Choosing edges according to per-step mesh optimization.
    - Or some way in between these two extremes (such as Garland and Heckbert's error quadrics)



Sharp edges are preserved using the optimization method

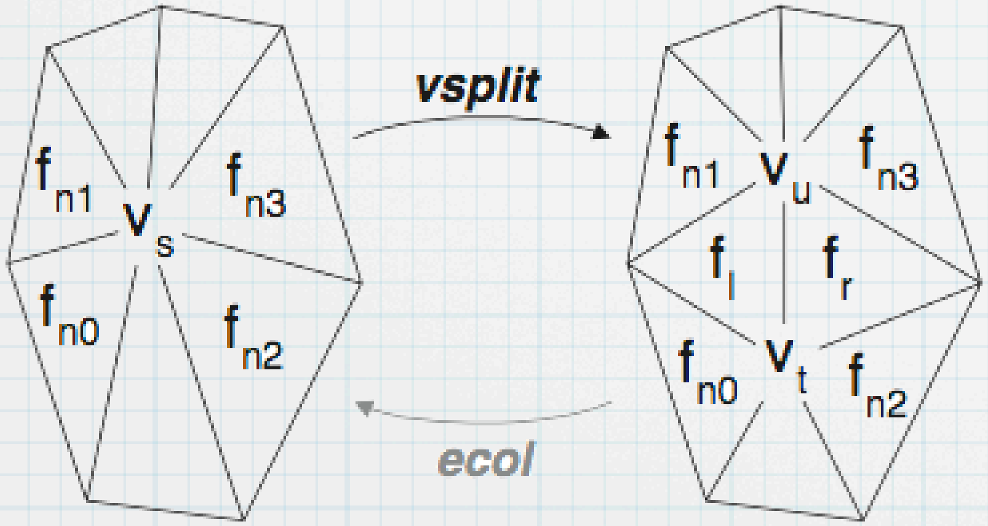
Vertex Hierarchy

PM is stored as a binary dependency tree where parent nodes 'vsplit' into their children:



Selective Refinement

Define a callback function 'refine(v)' which returns 'true' if a vertex should be refined, and 'false' otherwise. Example: return true iff 'v' and its neighbors lie inside view frustum

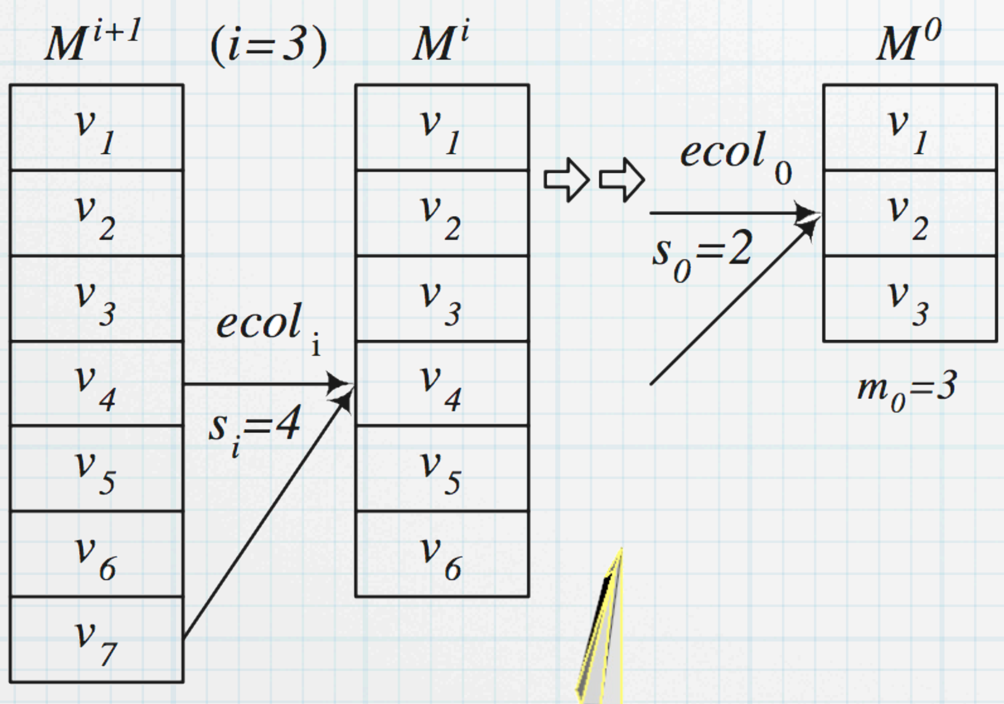


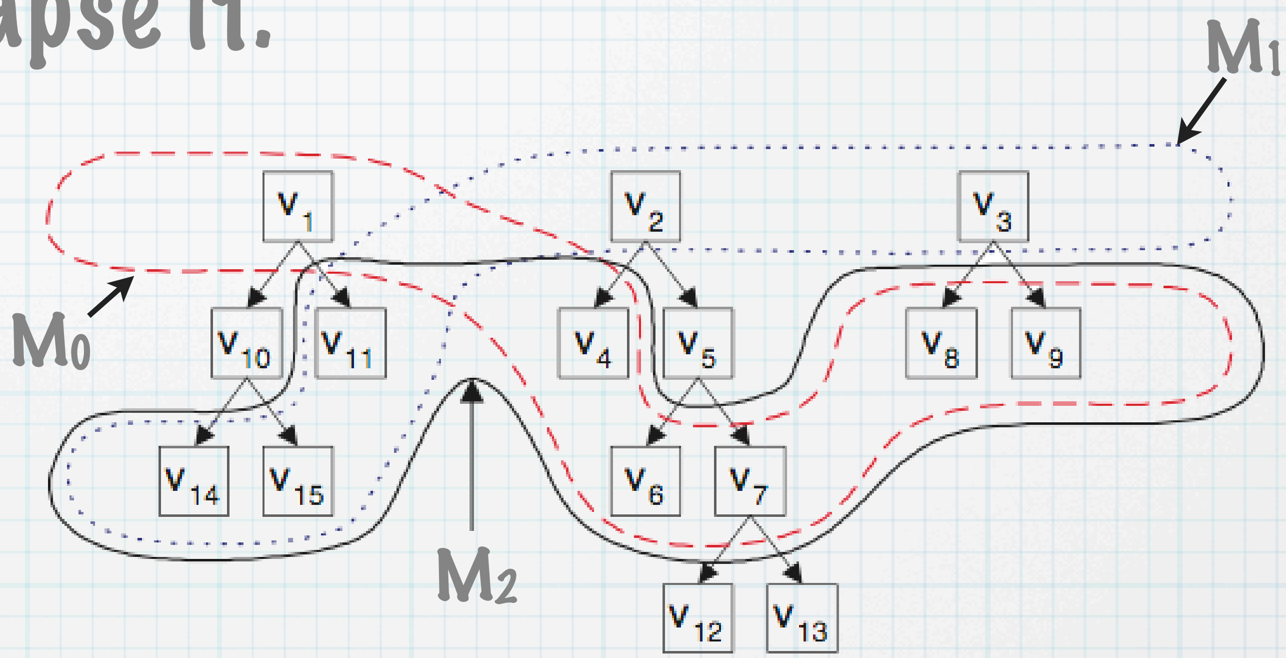
An intermediate mesh M^i is then refined by iterating through the list of 'vsplit' operations, but only 'vsplitting' if:

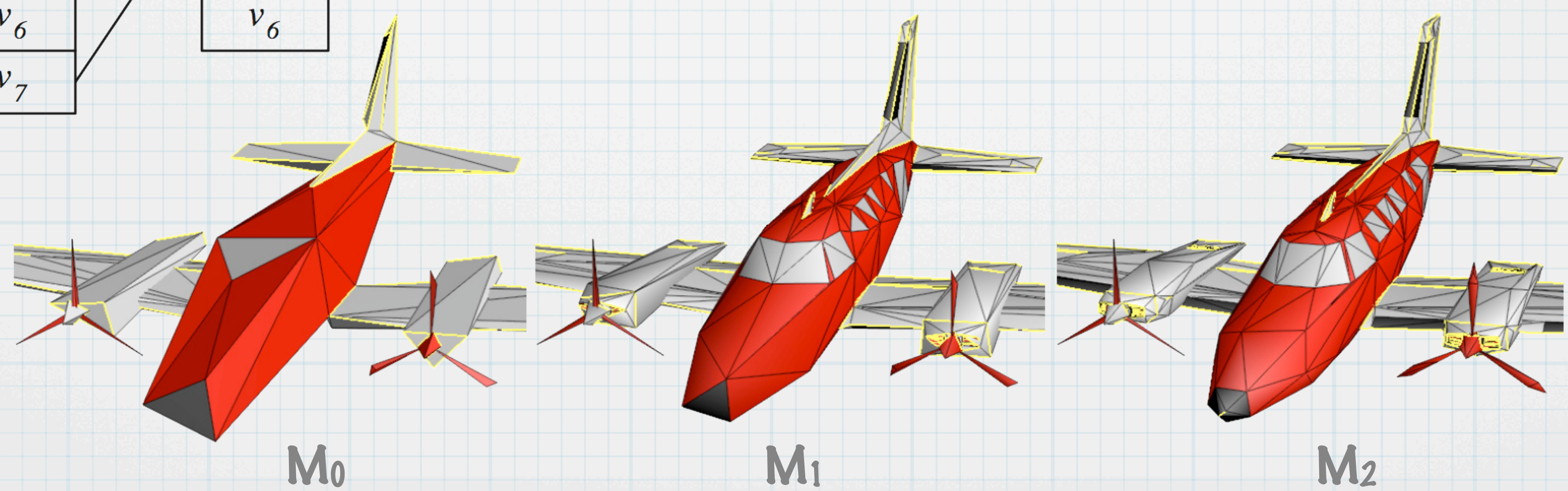
- the vsplit is valid
- refine(V_s) is 'true'

Algorithm

Idea: maintain list of active vertices. Traverse each V in this list and either leave as-is, split it, or collapse it.







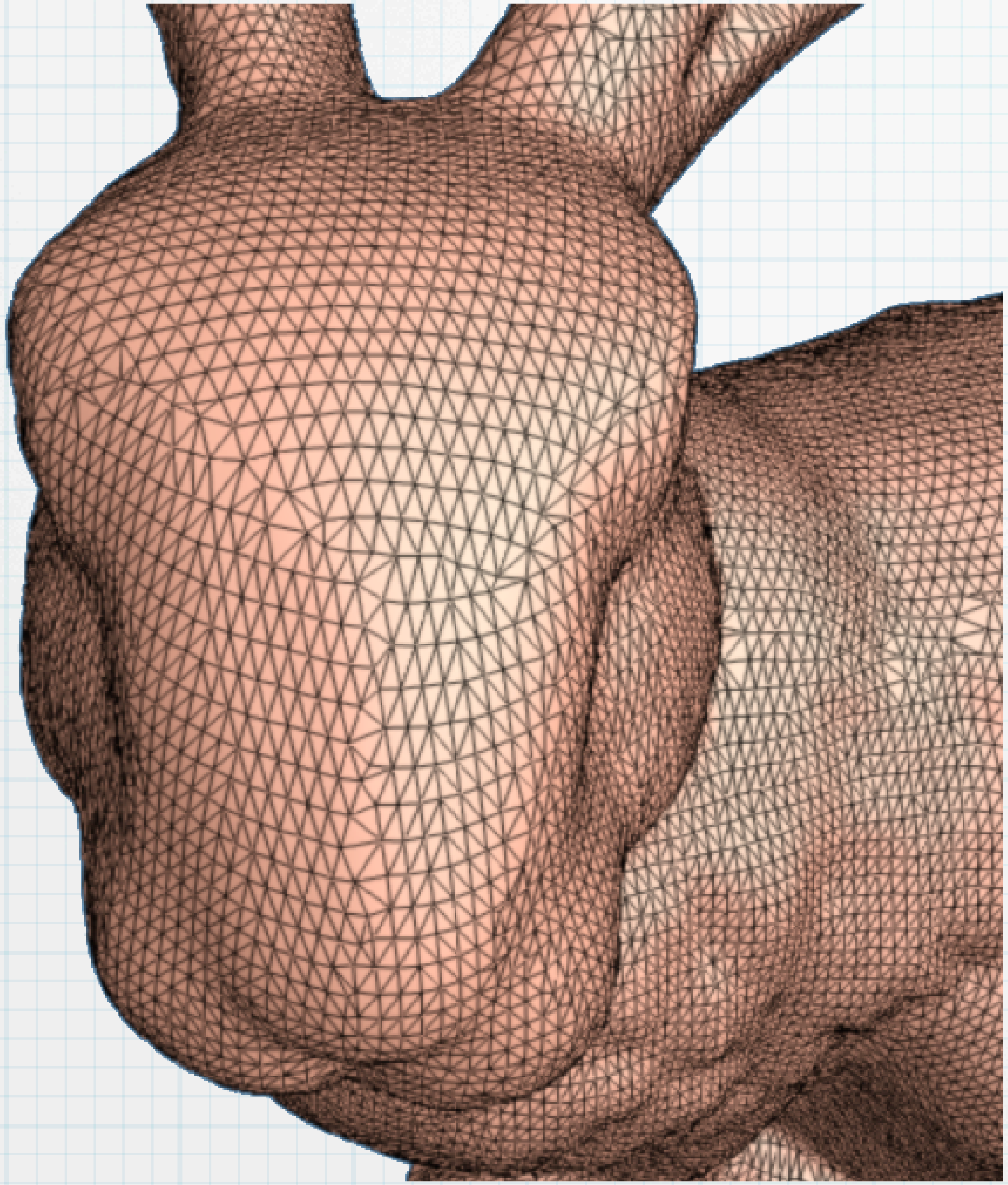
Algorithm

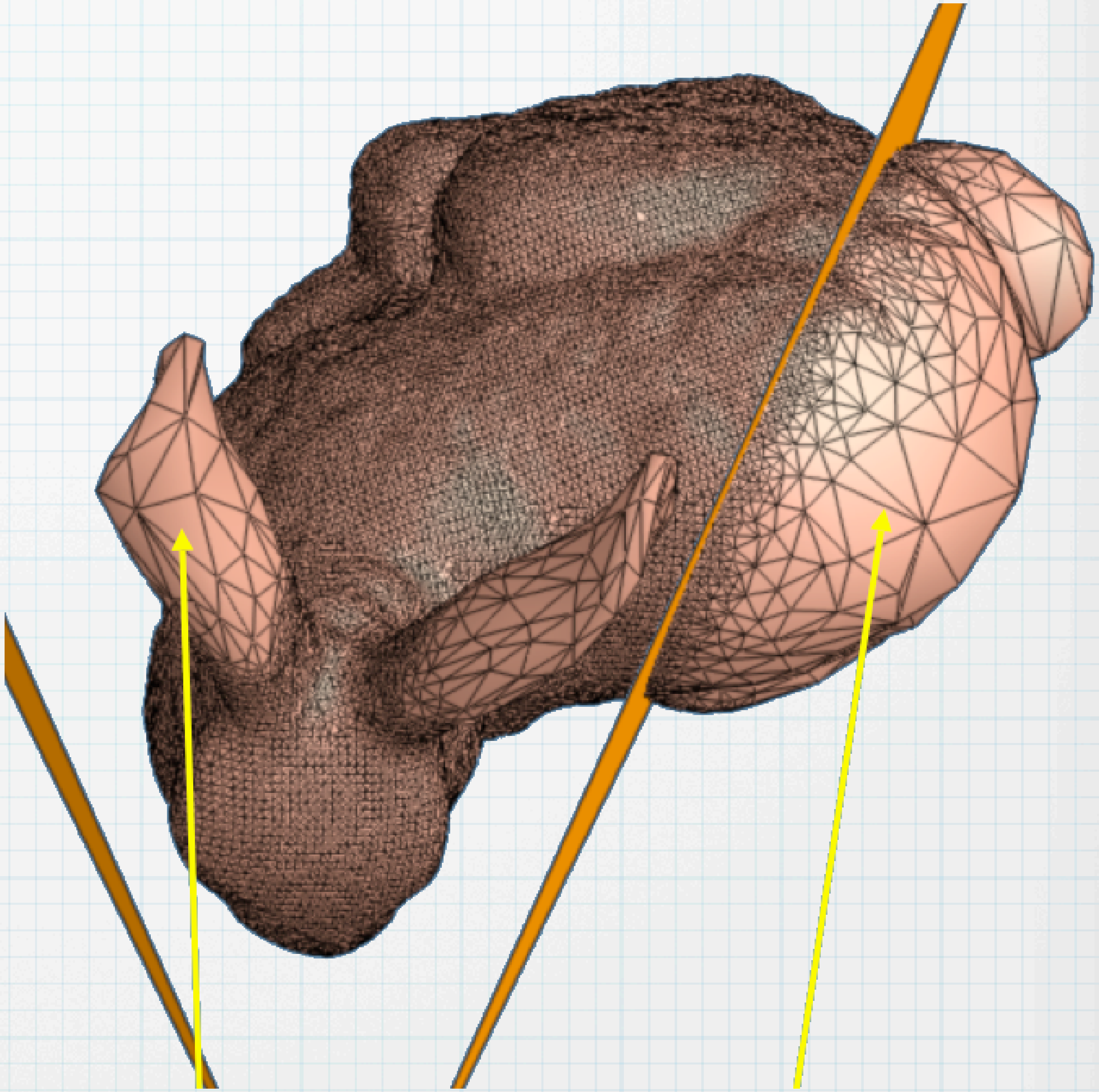
```
function adaptive_refinement()
        for each vertex V:
                if V is active and refine(V) is true:
                        force_vsplit(V)
                else if V has a parent P and refine(P) is false:
                        ecol(P) // (and reconsider some vertices)

function force_vsplit(v')
    stack = new stack; stack.push(v')
    while( v = stack.pop() ):
        if v active and v.left_face is active:
            stack.pop() // v was split earlier in the loop
        else if v is not active:
            stack.push(v's parent)
        else if vsplit_legal(v):
            stack.pop()
            vsplit(v)
        else
            for i in {0,1,2,3}: // force split that creates face i
                if(v.face[i] is not active):
                    stack.push( v.face[i]'s parent )
```

split other vertices to make the split legal

Results

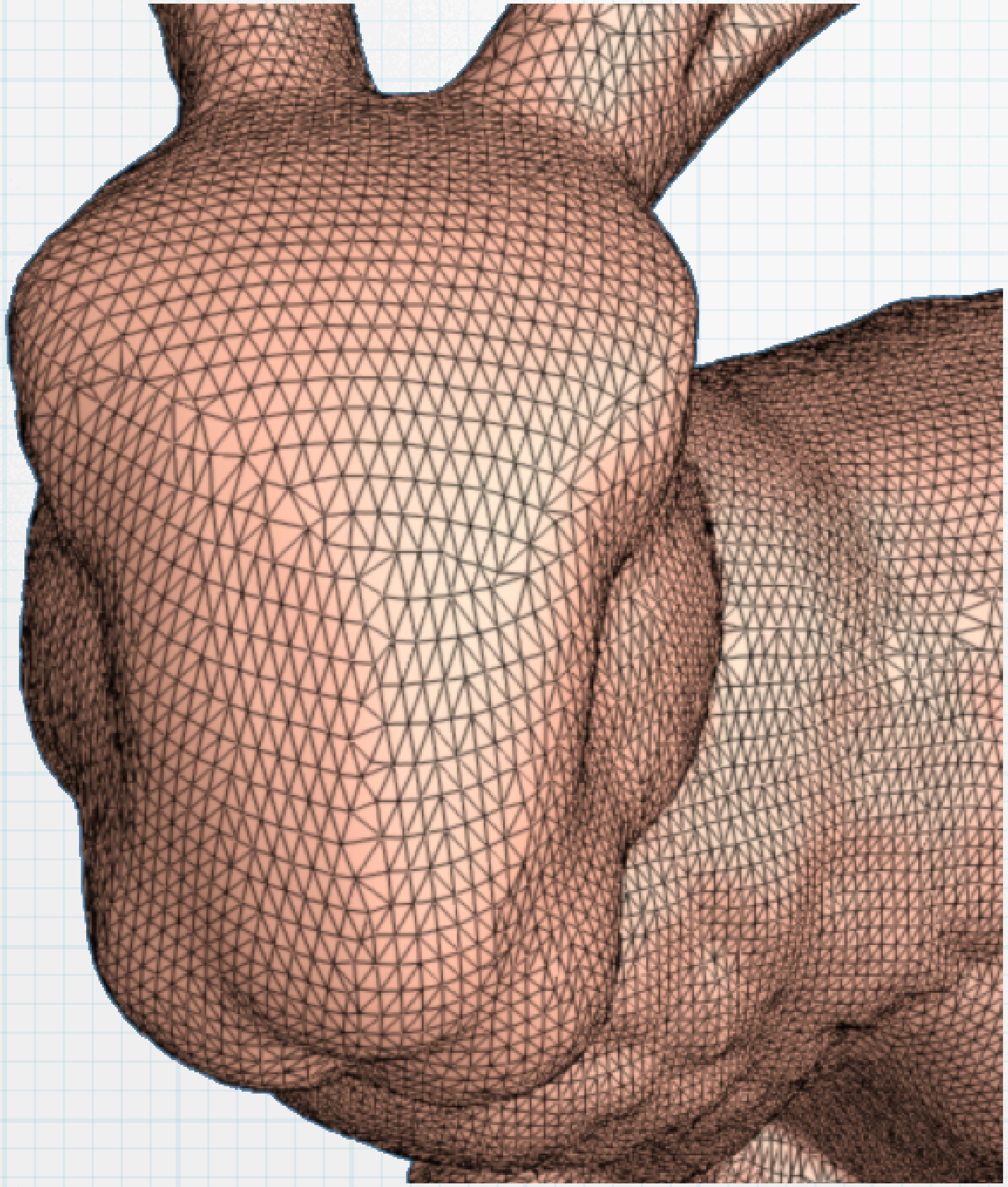


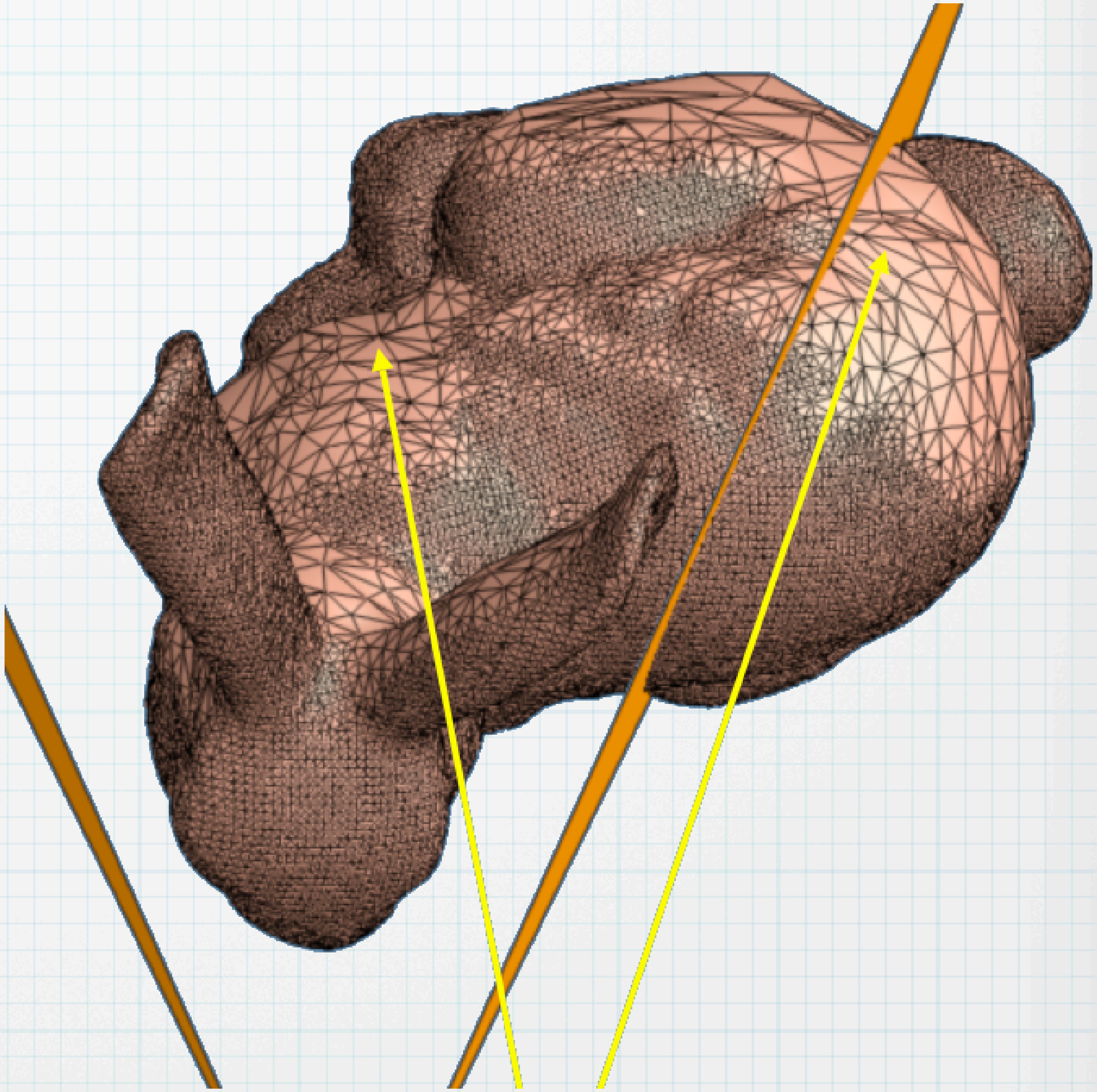


Too high

Too far right

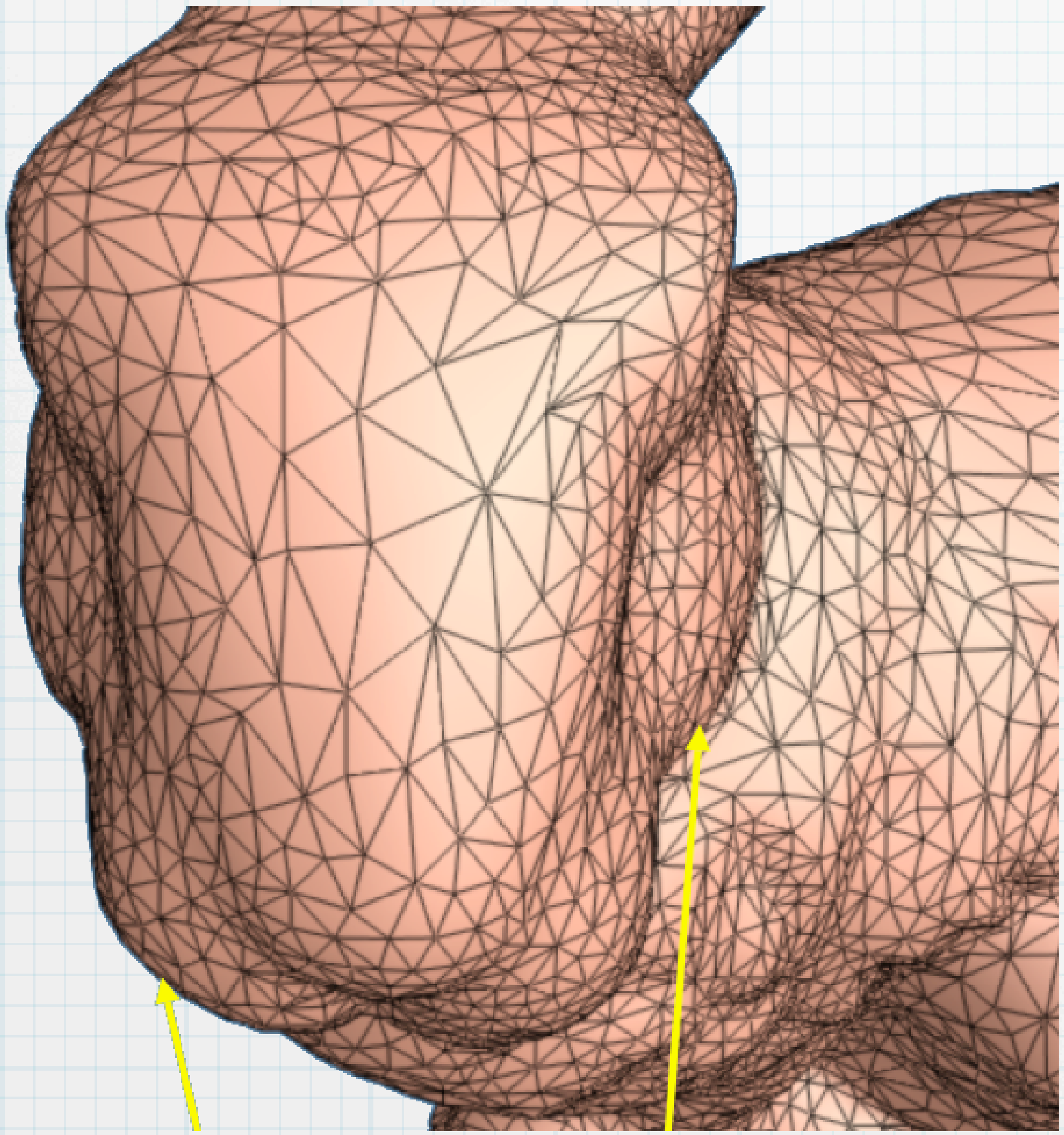
Results



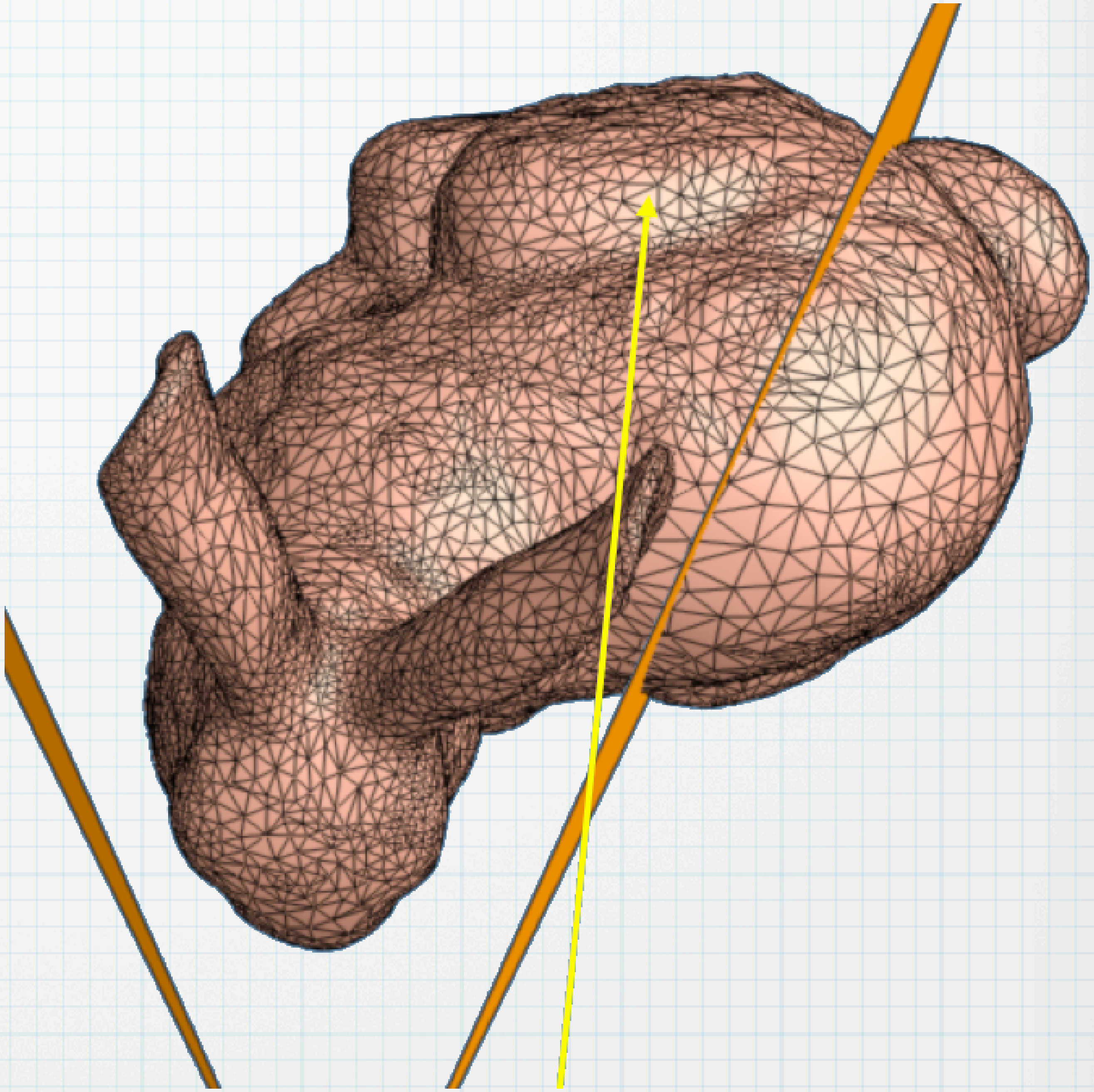


Oriented away

Results

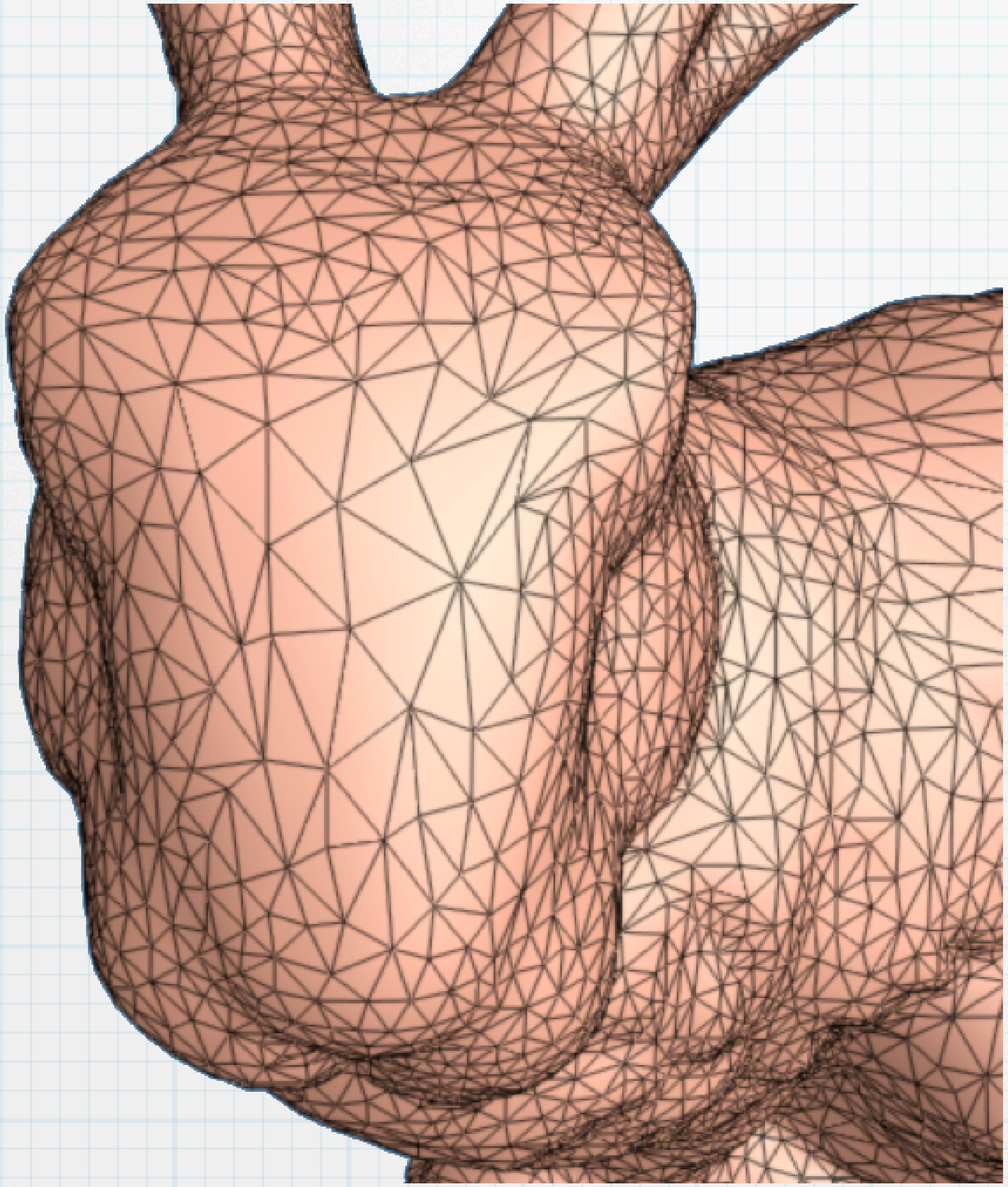


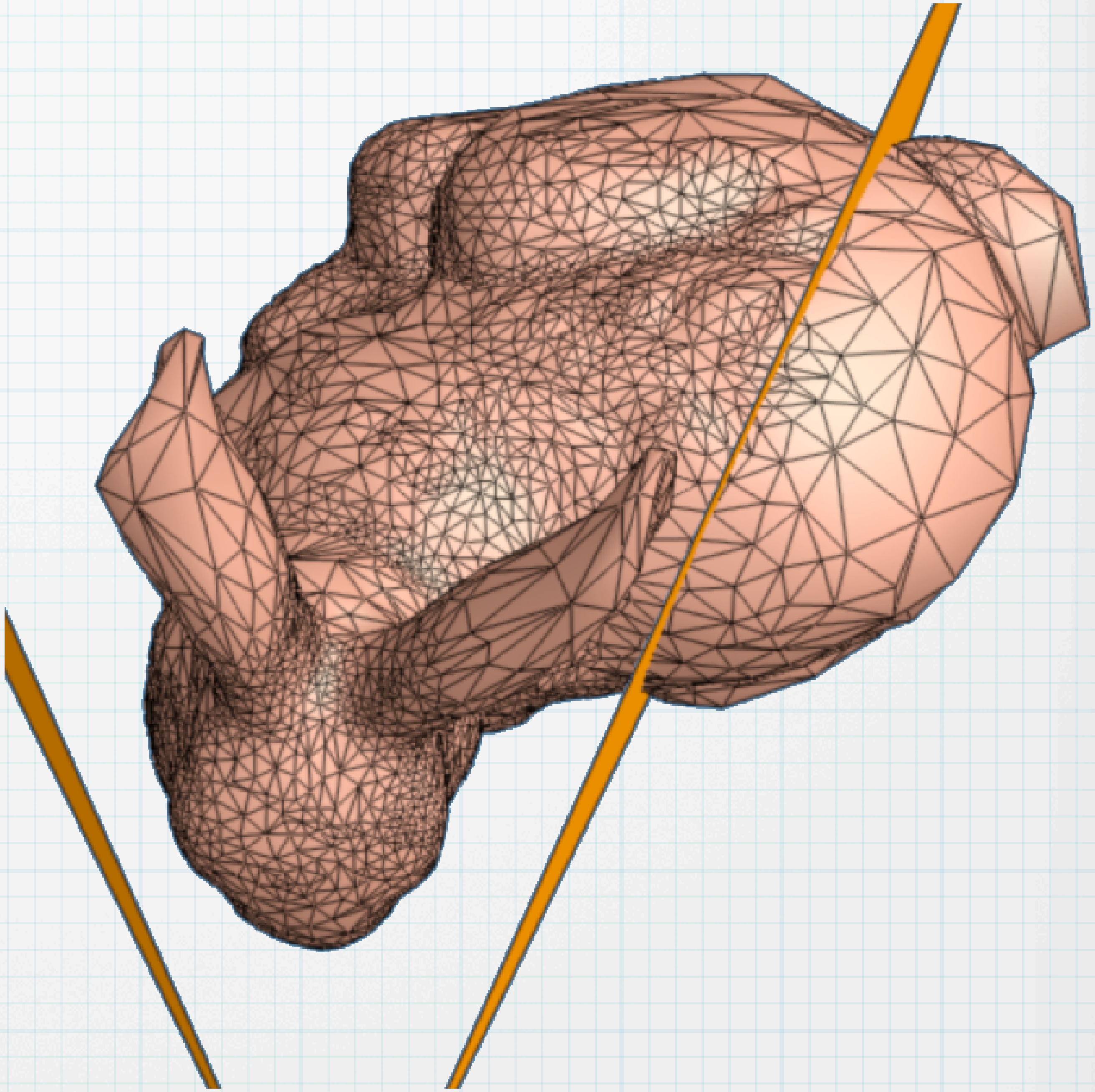
Refined silhouette



Coarse in distance

Results





Final Result

69k faces -> 10k faces   1.9 fps -> 6.7 fps

Results

Space Requirements

Model	Fully detailed $\hat{M}$		Disk (MB)		Mem.	V hier.	Constr.
	$\lvert\hat{V}\rvert$	$\lvert\hat{F}\rvert$	PM	$\{\mu, \delta\}$	(MB)	height	(mins)
$canyon_{200}$	40,000	79,202	1.3	0.3	8.9	29	47
$canyon_{400}$	160,000	318,402	5.0	1.1	35.8	32	244
$canyon_{600}$	360,000	717,602	11.0	2.6	80.6	36	627
” trunc.	200,600	400,000	6.6	1.5	44.9	35	627
sphere	9,902	19,800	0.3	0.1	2.2	19	11
teapot trunc.	5,090	10,000	0.2	0.0	1.1	20	12
gameguy	21,412	42,712	0.8	0.2	4.8	26	30
bunny	34,835	69,473	1.2	0.2	7.8	24	51

10 hours to compute PM

Time Complexity

	procedure	% of frame time	cycles/call
User	adapt_refinement	14 %	-
	(vsplit)	(0 %)	2200
	(ecol)	(1 %)	4000
	(qrefine)	(4 %)	230
	render (tstrip/face)	26 %	600
	GL library	19 %	-
System	OS + graphics	21 %	-
	CPU idle	20 %	-

refinement about twice as fast as rendering