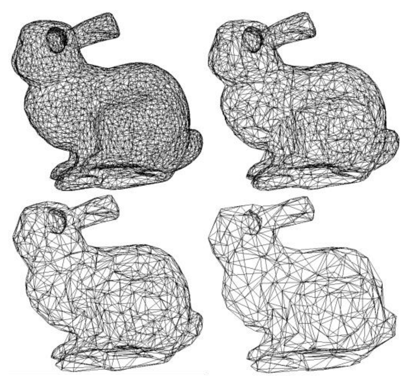
Dynamic Vertex Hierarchies for Parallel View-Dependent Progressive Meshes

Jonathan Merrin, Northeastern University **Mike Shah,** Northeastern University

INTRODUCTION



Triangles are the basic unit of complexity in computer rendering applications. More triangles makes for higher fidelity models and more complex scenes, but longer overall rendering times. As our scenes and models get more detailed, we draw more triangles to the screen per frame.

Due to this trade-off between quality and speed, we need ways to reduce the number of triangles we draw to the screen without sacrificing image quality. Here we explore view-dependent methods for reducing the number of triangles in a mesh and propose a new, easily parallelizable scheme for efficient view-dependent simplification. Our method requires less additional storage than other view-dependent methods and is more flexible in its ability to simplify meshes.

PRIOR WORK

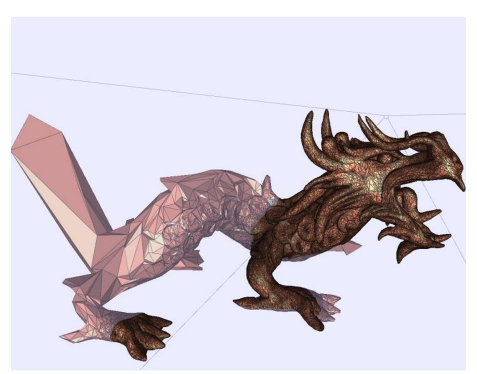


Figure 1: *An example of view dependent simplification*

Progressive Meshes (PM) allow for dynamic LODs through incremental mesh simplification and redetailing using vertex splits (vsplit) and edge collapses (ecol). View-dependent meshes (VDPM) allow for selective refinement of the mesh via check for whether a vsplit introduces error into the geometry. [Hu et al. 2009]'s contribution is an implicit way to check this criteria in parallel. These approaches each rely on preconstructing and storing a vertex hierarchy to simplify computation.

[Odaker et al. 2015] provided a system for parallel ecols using the half-edge data structure rather than a prebuilt vertex hierarchy. This approach does not include a method for performing vsplits, but it does introduce a method for checking whether an edge collapse will cause a mesh fold-over in parallel.

OUR METHOD

We propose a new legality check for vsplits equivalent to that of [Hu et al. 2009] that allows for generating dynamic vertex hierarchies. An example of the legality check is in Figure 2 (right), and works as follows:

- We start by adding a counter to every edge in the mesh
- Every time a pair of faces is removed, each of the incident edges have their counters incremented by 1 and merge each pair that shares a triangle.
- To check if a vsplit is legal, we check if the edges being split have the same values for their counters *that they had after the corresponding ecol*.
- After performing a vsplit, we decrement the counters by 1.

This condition reduces to the legality rules outlined in [Hoppe 1997].

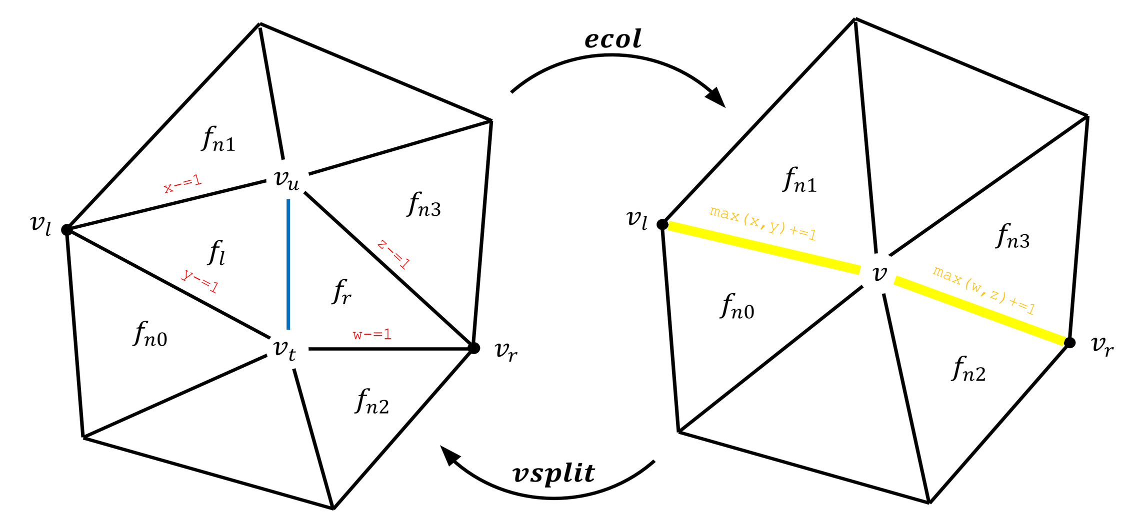


Figure 2: *An update to the vertex split/edge collapse operation that shows the additional effect and requirements of the operation under our new data structure.*

```
1: // Select edges to be collapsed, mark their vertices as being removed.
2: for e ∈ Edges do in parallel
3:     if should_collapse(e) then
4:         e.v1.ecol = e // arbitrarily have v1 consume v2.
5:     end if
6: end for
7:
8: // Check each vertex per face and set up the boundary rules according to
   [Odaker], then check the edges against the boundary.
9: for f ∈ Faces do in parallel
10:    vertices_removed = 0
11:    for v ∈ f.vertices do // 3 vertices
12:        if v.ecol ≠ null then
13:            vertices_removed++
14:        end if
15:    end for
16:    boundaries = corresponding boundary cases
17:    for v ∈ f.vertices do // 3 vertices
18:        if not (v.ecol and test_boundaries(v.ecol)) then
19:            v.ecol = null //cancel the ecol if it fails the boundary test
20:        end if
21:    end for
22: end for
23:
24: // Perform vsplits and ecols
25: for v = v0, v1, ..., vn ∈ Vertices do in parallel
26:    Assert(not splitting and collapsing)
27:    if v.ecol ≠ null then
28:        PerformEcol(v)
29:        continue
30:    end if
31:    if should_split(v) then
32:        PerformVsplit(v)
33:    end if
34: end for
```

Data Structures		
Static Structures		
Name	Members	Size
VertexBuffer	*positions*	$12n$
	normals	$4n$
	texCoords	$4n$
Total		$20n$
Dynamic Structures		
Index Buffer	$\{v_1, v_2, v_3\}$	$24m$
Edges	$\{v_1, v_2\}$	$8m * 3$
	$\{f_1, f_2\}$	$8m * 3$
Edge Metadata	*counter*	$n * 3$
	consumed	$4n * 3$
Vertex Information	$\{count_1, count_2\}$	$2n$
	split/collapse target	5n
	consumedBy	4n
Total		$46n + 72m$

Figure 3: *Our pseudo-code algorithm (left) and our data structures (right). Here n is used to denote the number of vertices in the original mesh and m is used to denote the number of vertices being displayed.*

Figure 3 (left) depicts the pseudocode for a simplification algorithm using our legality check and a list of the relevant data structures.

First we mark edges for removal (lines 1-6), then we test the edges to see if they cause fold-overs [Odaker et al. 2015] (lines 8-22), then we execute the remaining edge collapses and vertex splits (lines 24-34), which includes updating our counters.

Our method uses a total of 46n + 72m bytes to represent, where n is the number of vertices in the mesh and m is the number of vertices being displayed. This compares to [Hu et al. 2009]'s 69n + 56m. Since m is much smaller than n, we expect this to be a significant improvement.

CONCLUSIONS AND FUTURE WORK

We believe our method will improve the flexibility of parallel view-dependent progressive meshes and reduce the amount of space required to use them with minimal performance penalty. Unlike precomputed vertex hierarchies, our dynamic vertex hierarchies can be built in a view dependent way. This minimizes constraints on removing geometry from the mesh and allows us to decide which vertices depend on which at runtime.

One potential problem we foresee is that having a dynamic vertex hierarchy could increase the chance of worst case behavior in the form of long dependency lines. For this we propose an amortized rebalancing operation (Figure 4), which works as follows:

- First, we mark an edge with a dependency line that is more than two greater than any of its neighbors as a candidate for rebalancing. Conveniently, the counter described in Figure 2 also serves to keep track of the length of dependency lines.
- Next, we mark edges with dependency lines more than two greater than their neighbors as candidates for rebalancing.
- Finally, in the rebalancing step, we shift one node to a neighboring dependency line, as seen in Figure 4. This will not affect performance, since the amount of rendered geometry is unchanged.

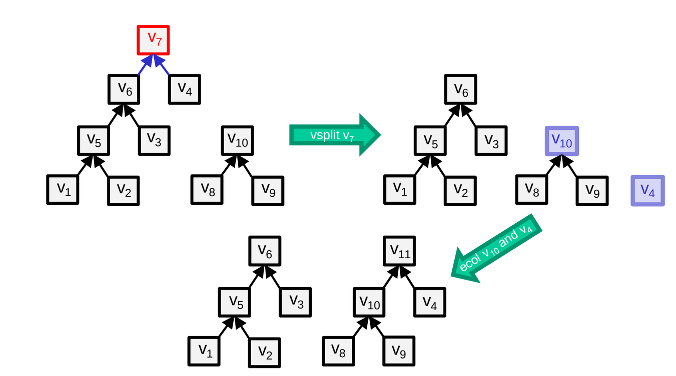


Figure 4: *A visualization of our hierarchy rebalancing operation. This example requires there to be an edge between **v4** and **v10**.*

Upon completion of the implementation and benchmarking of the above work, we will analyze the effect of including the amortized rebalancing operation against our baseline. Afterwards we will explore other applications of our system, including reducing meshes for shadow-casting and collision detection.

LITERATURE CITED

Hoppe, H. 1996. Progressive Meshes *ACM SIGGRAPH 1996 Proceedings*, 99-108.

Hoppe, H. 1997. View-dependent Refinement of Progressive Meshes. *ACM SIGGRAPH 1997 Proceedings*, 189-198.

Hu, L.; Sandler, P.; Hoppe, H. 2009. Parallel view-dependent refinement of progressive meshes. *Symposium on Interactive 3D Graphics and Games (I3D), 169-176.*

Xia, J., Varshney, A. 1996. Dynamic view-dependent simplification for polygonal models. *Visualization '96 proceedings*, IEEE 327-334

Odaker, Thomas, Dieter Kranzlmueller, and Jens Volkert (2015). "View-dependent simplification using parallel half edge collapses". In: SCG 2015 Conference on Computer Graphics, Visualization and Computer Vision

Liang Hu, Pedro V. Sandler and Hugues Hoppe (2009). "Parallel View-Dependent Refinement of Progressive Meshes". In: Symposium on Interactive 3D Graphics and Games (I3D), pp. 169–176.

Northeastern University
Khoury College of Computer Sciences

ACKNOWLEDGEMENTS

I would like to thank:

Mike Shah for advising on this project,

Jackie Allex for her insight and help with literature review,

and Heila Precel for being a sounding board and editing this poster.