Traditional ‘Level of Detail’

Entire meshes are uniformly simplified.
Problems with LoD

- Appearance
- Silhouettes
- Efficiency
- Non-visible faces
- Back-facing
- Off-screen
View-Dependent LoD

distant

distant

outside

back-facing

Eye
View-Dependent LoD
Progressive Meshes

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

$$M^0 \xrightarrow{\text{vsplit}_0} M^1 \xrightarrow{\text{vsplit}_1} \ldots \xrightarrow{\text{vsplit}_{n-1}} (M^n = \hat{M})$$

$$(\hat{M} = M^n) \xrightarrow{\text{ecol}_{n-1}} \ldots \xrightarrow{\text{ecol}_1} M^1 \xrightarrow{\text{ecol}_0} M^0$$
Progressive Meshes

vsplt\( (v_s,v_t,v_u,f_l,f_r,f_{n0},f_{n1},f_{n2},f_{n3}) \)
\* 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
\* vsplt
  \* \( v_s \) 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

\[ \text{vsplit}(v_s,v_t,v_u,f_l,f_r,f_{n0},f_{n1},f_{n2},f_{n3}) \]

- replaces ‘parent’ \( v_s \) with ‘children’ \( v_t \) and \( v_u \).
- creates faces \( f_l \) and \( f_r \).

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

* Performed only once (i.e., 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.
PM is stored as a binary dependency tree where parent nodes 'vsplit' into their children:
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'
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

<table>
<thead>
<tr>
<th>Model</th>
<th>Fully detailed M</th>
<th>Disk (MB)</th>
<th>Mem. (MB)</th>
<th>V hier. height</th>
<th>Constr. (mins)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>V</td>
<td>F</td>
<td>PM</td>
<td>{\mu, \delta}</td>
<td></td>
</tr>
<tr>
<td>canyon200</td>
<td>40,000</td>
<td>79,202</td>
<td>1.3</td>
<td>0.3</td>
<td>8.9</td>
</tr>
<tr>
<td>canyon400</td>
<td>160,000</td>
<td>318,402</td>
<td>5.0</td>
<td>1.1</td>
<td>35.8</td>
</tr>
<tr>
<td>canyon600</td>
<td>360,000</td>
<td>717,602</td>
<td>11.0</td>
<td>2.6</td>
<td>80.6</td>
</tr>
<tr>
<td>” trunc.</td>
<td>200,600</td>
<td>400,000</td>
<td>6.6</td>
<td>1.5</td>
<td>44.9</td>
</tr>
<tr>
<td>sphere</td>
<td>9,902</td>
<td>19,800</td>
<td>0.3</td>
<td>0.1</td>
<td>2.2</td>
</tr>
<tr>
<td>teapot trunc.</td>
<td>5,090</td>
<td>10,000</td>
<td>0.2</td>
<td>0.0</td>
<td>1.1</td>
</tr>
<tr>
<td>gameguy</td>
<td>21,412</td>
<td>42,712</td>
<td>0.8</td>
<td>0.2</td>
<td>4.8</td>
</tr>
<tr>
<td>bunny</td>
<td>34,835</td>
<td>69,473</td>
<td>1.2</td>
<td>0.2</td>
<td>7.8</td>
</tr>
</tbody>
</table>

10 hours to compute PM

Time Complexity

<table>
<thead>
<tr>
<th>procedure</th>
<th>% of frame time</th>
<th>cycles/call</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td></td>
<td></td>
</tr>
<tr>
<td>adapt.refinement</td>
<td>14 %</td>
<td>-</td>
</tr>
<tr>
<td>(vsplit)</td>
<td>(0 %)</td>
<td>2200</td>
</tr>
<tr>
<td>(ecol)</td>
<td>(1 %)</td>
<td>4000</td>
</tr>
<tr>
<td>(qrefine)</td>
<td>(4 %)</td>
<td>230</td>
</tr>
<tr>
<td>render (tstrip/face)</td>
<td>26 %</td>
<td>600</td>
</tr>
<tr>
<td>GL library</td>
<td>19 %</td>
<td>-</td>
</tr>
<tr>
<td>System</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS + graphics</td>
<td>21 %</td>
<td>-</td>
</tr>
<tr>
<td>CPU idle</td>
<td>20 %</td>
<td>-</td>
</tr>
</tbody>
</table>