Visualization of Large Terrains Made Easy

Peter Lindstrom and Valerio Pascucci
Lawrence Livermore National Laboratory
Hierarchical error + static data layout = large terrain visualization made easy.

• Hierarchical error computation:
  — Independent of error metric
  — Combined view culling
  — Near optimality
  — Asynchronous updates

• Hierarchical indexing:
  — Static data layout
  — Generic paging system

• Simple:
  — No explicit hierarchy
  — No priority queue
  — No specialized I/O system
  — No mesh data-structure (implicit stripping)
Uniform refinement of the rectilinear grid using longest edge bisection.

Level 0 (base mesh)
Uniform refinement of the rectilinear grid using longest edge bisection.

Level 1
Uniform refinement of the rectilinear grid using longest edge bisection.
Uniform refinement of the rectilinear grid using longest edge bisection.

Level 3
Uniform refinement of the rectilinear grid using longest edge bisection.
Uniform refinement of the rectilinear grid using longest edge bisection.
The stripping of any uniform refinement is the Sierpinski space filling curve.
Adaptive refinement of the rectilinear edge bisection.

Level 0 (base mesh)
Adaptive refinement of the rectilinear edge bisection.
Adaptive refinement of the rectilinear edge bisection.
Adaptive refinement of the rectilinear edge bisection.
Adaptive refinement of the rectilinear edge bisection.
Adaptive refinement of the rectilinear edge bisection.
Adaptive refinement of the rectilinear edge bisection.
The dynamic update of an adaptive mesh may be very easy and efficient.

Do not rebuild. Just refine a single triangle.
The dynamic update of an adaptive mesh may be NOT so easy and efficient.

One refinement may trigger ripple effect.
The dynamic update of an adaptive mesh may be NOT so easy and efficient.

One refinement may trigger ripple effect.
The dynamic update of an adaptive mesh may be NOT so easy and efficient.

One refinement may trigger ripple effect.
The dynamic update of an adaptive mesh may be NOT so easy and efficient.

One refinement may trigger ripple effect.
The dynamic update of an adaptive mesh may be NOT so easy and efficient.

One refinement may trigger ripple effect.
The dynamic update of an adaptive mesh may be NOT so easy and efficient.

One refinement may trigger ripple effect.
The dynamic update of an adaptive mesh may be NOT so easy and efficient.

One refinement may trigger ripple effect.
The dynamic update of an adaptive mesh may be NOT so easy and efficient.

One refinement may trigger ripple effect
The dynamic update of an adaptive mesh may be NOT so easy and efficient.

One refinement may trigger ripple effect.
The dynamic update of an adaptive mesh may be NOT so easy and efficient.

Must maintain:
- mesh topology
- priority queue

One refinement may trigger ripple effect.
The rectilinear edge bisection can be defined in terms of a vertex hierarchy.
The rectilinear edge bisection can be defined in terms of a vertex hierarchy.
The rectilinear edge bisection can be defined in terms of a vertex hierarchy.
The rectilinear edge bisection can be defined in terms of a vertex hierarchy.
The rectilinear edge bisection can be defined in terms of a vertex hierarchy.
Selecting nodes on the basis of the error alone does not guarantee a valid mesh $M$. 

$\delta \geq \epsilon$
Selecting nodes on the basis of the error alone does not guarantee a valid mesh $M$. 

$M$ is valid iff:

$\forall v \in M \Rightarrow \text{parent}(v) \in M$
A hierarchical error metric overcomes the dichotomy of error vs. consistency.

We inflate the geometric error from $\delta$ to $\delta^*$

$$\delta^* = \max\{\delta, \delta_1^*, \ldots, \delta_4^*\}$$
A hierarchical error metric simplifies the mesh construction and stripping.

\[
\text{mesh-refine}(VB,v1,v2,l) \\
\text{if } l > 0 \text{ and } \delta^*(v1) \geq \varepsilon \text{ then} \\
\text{mesh-refine } (VB,v2,Cl,l-1) \\
\text{strip-append}(VB,v1,l \mod 2) \\
\text{mesh-refine } (VB,v2,Cr,l-1)
\]
The error projection can destroy the nested structure of the metric.

\[ \rho = \lambda \frac{\delta^*}{\|p - v\|} \]
We inflate the projected error by replacing points with spheres.

\[ \rho^* = \lambda \frac{\hat{\delta}^*}{\|p - v\| - r} \]
Nested spheres yield a view dependent hierarchical error metric.

\[ \rho^* \geq \tau \iff \left( \frac{\lambda}{\tau} \delta^* + r \right)^2 \geq \| p - v \|^2 \]

6 additions
5 multiplications
The view dependent hierarchical error metric is not far from the optimal.
Nested spheres allow fast and simple integrated view culling.

- The culling test is performed only if the sphere of the parent intersects the boundary of the view frustum.

\[ \leq 6 \text{ times dot product and comparison} \]

No test on descendants
A hierarchical error metric simplifies the mesh construction and stripping.

\[
\text{mesh-refine}(\text{VB}, v_1, v_2, l)
\]

\[
\text{if } l > 0 \text{ and } \rho^*(v_1) \geq \tau \text{ then }
\]

\[
\begin{align*}
\text{mesh-refine} & (\text{VB}, v_2, C_1, l-1) \\
\text{strip-append} & (\text{VB}, v_1, l \mod 2) \\
\text{mesh-refine} & (\text{VB}, v_2, C_r, l-1)
\end{align*}
\]
We develop a simple data layout based on quad-trees.

The vertices inserted at the even levels of refinement are the centers of each square and form a (white) quad-tree.
We develop a simple data layout based on quad-trees.

The black vertices inserted at the odd levels are the corners of each square and form a quad-tree if gray vertices are added.
We store the white nodes in place of the gray nodes.
We simply layout the data element level by level starting from

The index of $C_i$ is computed from the index of the parent $P$

$$C_i = 4 \times P + i$$

$$C_1 = 4 \times v_1 - 11 + \left( (2 \times v_1 + v_2 + 2) \mod 4 \right)$$

$$C_r = 4 \times v_1 - 11 + \left( (2 \times v_1 + v_2 + 3) \mod 4 \right)$$
A hierarchical error metric simplifies the mesh construction and stripping.

$$\text{mesh-refine}(VB, v1, v2, l)$$

if $$l > 0$$ and $$\rho^*(v1) \geq \tau$$ then

$$\text{mesh-refine} \ (VB, v2, C1, l-1)$$

$$\text{strip-append} \ (VB, v1, l \mod 2)$$

$$\text{mesh-refine} \ (VB, v2, Cr, l-1)$$
We implemented the scheme with four alternative data layout schemes.

Practical comparison:

- Linear
- Block
- ZH-order
- Quad-tree
Performance Tests
(5GB dataset on a 800MB SGI)

- Linear
- Block
- ZH
- Quad-tree

total number of page faults vs. screen space error tolerance (pixels)
Performance Tests
(5GB dataset on a 800MB SGI)
Performance Tests
(1.25GB dataset on a 64MB PC)

cumulative number of page faults
frame number (log scale)
Comparison of in core performance with respect to threading and culling.
Comparison of in core performance with respect to threading and culling.

- Single-threaded w/o culling
- Single-threaded w/ culling
- Multi-threaded w/ culling

refinement speed (million triangles/s)

rendered frame number
Conclusions and future directions.

- Incore speedup 2X
- Sustained 40k per frame – 30 HZ
- Good multithreading
- 1.5 Millions triangles per second

- Texture
- Mem efficiency
- Compression
- Explore more general 4-k meshes
- Geomorphing
- Add more sophisticated paging and prefetching
UCRL-PRES-154167

- This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.