Abstract

Live Range Visibility Constraints for Adaptive Terrain Visualization

Xiaohong Bao*
University of California, Irvine

Renato Pajarola†
University of California, Irvine

Michael Shafae‡
University of California, Irvine

1 Introduction

Although there is a remarkable pace in the advance of computational resources and storage for real-time visualization the immensity of the input data continues to outstrip any advances. The task for interactively visualizing such a massive terrain is to render a triangulated mesh using a view-dependent error tolerance, thus intelligently and perceptually managing the scene's geometric complexity. At any particular instance in time (i.e. displayed frame), this *level-of-detail* (LOD) terrain surface consists of a mesh composed of hundreds of thousands of dynamically selected triangles. The triangles are selected using the current time-step's view parameters and the view-dependent error tolerance. Massive terrain data easily exceeds main memory storage capacity such that out-of-core rendering must be performed. This further complicates the triangle selection and terrain rendering owing to tertiary storage's relatively poor performance.

The proposed algorithm, SMART, in this paper dramatically improves the previous state-of-the-art by introducing the paradigm of a vertex *live range*. Each vertex has an associated live range. An a priori judgment can be made of what vertices need to be recalled from tertiary storage using a live range check. Using this live range, per-vertex LOD selection and culling costs are dramatically reduced. Moreover, this reduction in LOD and culling costs directly reduces the amount of out-of-core I/O.

Consider a continuous fly-over of a massive terrain data. Arbitrarily choosing between any two consecutive frames, there exists a significantly large spatial coincidence. By using a vertex live range, our method exploits this fact and generates an updated view-dependent LOD terrain representation by touching only those vertices that have changed between frames. Furthermore, our method does not maintain any sorting data structure, neither accesses the vertex itself, usually it is costly from out-of-core, if the vertex status is constant. SMART minimizes the per-vertex LOD and view culling tests to such a large degree that it is competitive to cluster-based LOD selection, however, offering at the same time fine-grain LOD vertex selection to generate smoother view with minimal number of necessary triangles.

2 Vertex Live Range

Our goal is to avoid the repetitive cost and only perform LOD and view culling computation for the few vertices that are indeed candidates for updating the LOD-mesh for a given viewpoint. Ideally, when we check a vertex's status, we want to know not only its status at the current moment. We would also like to know when its current status will possibly change, no matter how the view-dependent parameters will vary. This time period is called the *live range* of a vertex. To compute a vertex's live range, we define for each vertex a *safe-distance*. No matter how the view point moves, a vertex status keeps constant if the view point is within its safe-distance. Only if the viewpoint invalidates the safe-distance to a vertex does its visibility status possibly change.

*e-mail: xbao@ics.uci.edu
†e-mail: pajarola@acm.org
‡e-mail: mshafae@ics.uci.edu

Among the multiresolution terrain models reviewed in [Pajarola 2002], we use the triangle bin-tree definition [Duchaineau et al. 1997; Lindstrom and Pascucci 2001] here. In this context, mesh refinement is performed by recursively splitting a triangle at the midpoint of its longest edge which defines a binary hierarchy on the triangles. See [Duchaineau et al. 1997; Lindstrom and Pascucci 2001] for more implementation details.

2.1 τ-safe-distance

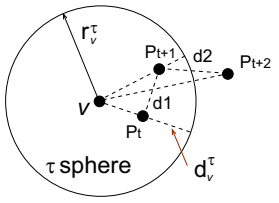

Figure 1: The concept of τ-sphere of a vertex v and its τ-safe-distance $d_v^\tau(t)$, given the view point moves from P_t, through P_{t+1}, to P_{t+2}. At P_{t+1}, $d_v^\tau(t) - d_1 \geq 0$, vertex v remains visible. But at P_{t+2}, $d_v^\tau(t) - d_1 - d_2 < 0$ causes re-evaluation of the τ-safe-distance d_v^τ at time $t+2$.

The τ-safe-distance $d_v^\tau(t)$ measures the degree of freedom of unconstrained movements the viewpoint P can enjoy before the LOD status of a vertex v is potentially affected. The τ-safe-distance $d_v^\tau(t)$ is initialized when a vertex visibility is computed at the first time and is then estimated at $\bar{d}_v^\tau(t+\delta t) = d_v^\tau(t) - \sum_{i=1}^{\delta t} d_i$ with $d_i = |P_{t+i} - P_{t+i-1}|$. Only when the sign of $\bar{d}_v^\tau(t)$ changes, the actual $d_v^\tau(t)$ is re-evaluated. This conservative check guarantees that all high risk vertices are being found. The τ-safe-distance is defined as:

$$d_v^\tau(t) = r_v^\tau - |P_t - v| \tag{1}$$

where $r_v^\tau = \frac{\alpha}{\tau}\delta_v + r_v$ is the radius of v's τ-sphere. Here α is a constant respect to the view volume, δ_v is the saturated object space error of v and r_v is the radius of v's bounding sphere.

2.2 culling-safe-distance

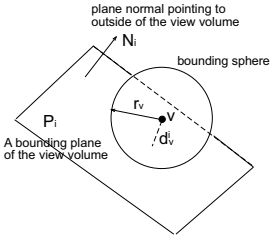

Figure 2: The concept of a vertex' bounding sphere and its culling-safe-distance $d_v^c(t) = \min_i(r_v - d_v^i)$. The view frustum plane P_i can move by $d_v^c(t)$ without changing vertex v's culling status.

Similar to the definition of the τ-safe-distance d_v^τ, culling-safe-distance $d_v^c(t) = \min_i(r_v - d_v^i)$ defines a conservative safe distance the view point can move before the culling status of vertex v is re-evaluated. Vertex τ-safe-distance is similarly computed along the nested bounding sphere hierarchy. For instance, if the bounding sphere of a vertex is in the view frustum, we set the τ-safe-distance

of each of its descendants equal to its r_v without any further accurate computation.

2.3 Dynamic Vertex Selection

We define the *live range* of a vertex v as

$$L_v = f_t + \frac{\min\left(|d_v^\tau(t)|, |d_v^c(t)|\right)}{S_{min}} \quad (2)$$

where f_t is the frame number at time t, and S_{min} is the minimal speed the view point moves (per frame). Assume the positions of the viewpoint at time $t-1$ and t are P_{t-1} and P_t respectively , then f_t is updated as follows:

$$f_t = f_{t-1} + \lfloor \frac{|P_t - P_{t-1}|}{S_{min}} \rfloor. \quad (3)$$

The live range L_v of v defines the period that v's visibility status is constant. L_v is updated only when v's status is updated. f_t is calculated once for each frame. v's visibility status will not change at the moment t if $f_t < L_v$; otherwise L_v needs to be re-evaluated. Here L_v and f_t are both integers.

Even though we define a S_{min}, the low bound of speed here, the user's navigation speed can be less than S_{min} without violating the algorithm correctness. In that case the less the actual speed is than S_{min}, the more conservative the vertex L_v test will be.

The algorithm works as follows: we check each vertex in a depth-first traversal of the triangle bin-tree hierarchy. If the vertex live range is valid, the vertex status is constant and accessing the vertex itself is unnecessary. Otherwise the vertex status is re-evaluated. Given the return vertex status with respect to both LOD and culling, the traversal recursively subdivides triangles if necessary and generates the triangle-strip rendering primitive along the way as described in [Lindstrom and Pascucci 2001].

3 Results

We present the results of our algorithm and compare them with SOAR[1]. The terrain data set is made up of 8193×8193 elevation grid. The data file on disk is organized with the technique of interleaved quadtree indexing used in SOAR, occupying 2GB disk space.

Figure 3 demonstrates the effectiveness of our algorithm. When the screen error tolerance is one pixel, SMART performs less than 10% of the vertex visibility computations that the basic SOAR engine does for most frames. Even in the worst case this reduction ratio is about 20%. This number shows that during the mesh computation, not only the 90% expensive visibility computations are avoided, but possible out-of-core vertex accessing for these vertices is also avoided. This leads to great improvement in the final rendering performance.

Figure 4 presents SMART speedup over SOAR with different frame-to-frame incoherence. The average frame-to-frame incoherence is measured by the average percentage of changed triangles between consecutive frames. When the incoherence is 5.29%, in which the view point moves at around 20 km per second, the overall speedup and pure mesh computation speedup are more than 7 over SOAR. The process of pure mesh computation is the whole visualization process except OpenGL calls, which includes vertex visibility status computation and final triangle strip construction. They reach 8 and more when the frame-to-frame incoherence declines to 1.36%, due to much less floating number computations for updating vertex visibility status. The speedup is reduced to 6

[1]The source code of SOAR and the data set used in this paper are offered by Peter Lindstrom and Valerio Pascucci at http://www.cc.gatech.edu/gvu/people/peter.lindstrom.

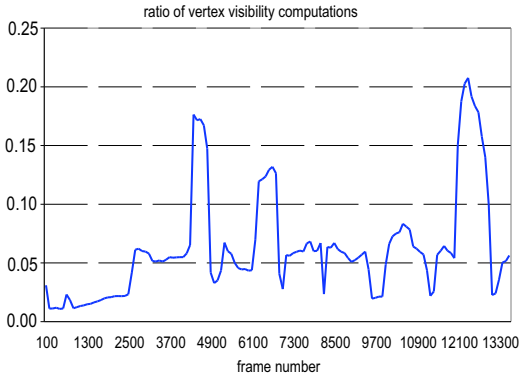

Figure 3: Compared to the technique in SOAR, the ratio of necessary vertex accesses and visibility computations with SMART along the flying route when the screen space error tolerance is 1 pixel and frame-to-frame incoherence is 0.61%.

when the incoherence is 0.61%. In this case good data locality is preserved with the data layout technique in SOAR, reducing the number of page faults, thus improving the out-of-core data access efficiency for SOAR.

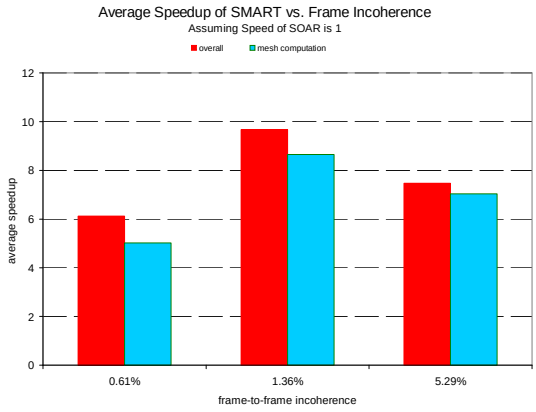

Figure 4: Overall and pure mesh computation speedup with SMART for different frame-to-frame incoherence when the screen space error tolerance is 2 pixels.

References

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2003. BDAM - batched dynamic adaptive meshes for high performance terrain visualization. In *Proceedings EUROGRAPHICS 2003*, 505–514. also in Computer Graphics Forum 22(3).

DUCHAINEAU, M., WOLINSKY, M., SIGETI, D. E., MILLER, M. C., ALDRICH, C., AND MINEEV-WEINSTEIN, M. B. 1997. Roaming terrain: Real-time optimally adapting meshes. In *Proceedings IEEE Visualization 97*, 81–88.

LEVENBERG, J. 2002. Fast view-dependent level-of-detail rendering using cached geometry. In *Proceedings IEEE Visualization 2002*, Computer Society Press, 259–266.

LINDSTROM, P., AND PASCUCCI, V. 2001. Visualization of large terrains made easy. In *Proceedings IEEE Visualization 2001*, Computer Society Press, 363–370.

PAJAROLA, R. 2002. Overview of quadtree-based terrain triangulation and visualization. Tech. Rep. UCI-ICS-02-01, Information & Computer Science, University of California Irvine.