Real-time Extendible-resolution Display of On-line Dynamic Terrain

Yefei He
National Advanced Driving Simulator
& Simulation Center
The University of Iowa

James Cremer
Computer Science Department
The University of Iowa

Yiannis Papelis
National Advanced Driving Simulator
& Simulation Center
The University of Iowa

Abstract

We present a method for multiresolution view-dependent real-time display of terrain undergoing on-line modification. In other words, the method does not assume static terrain geometry, nor does it assume that the terrain update sequence is known ahead of time. The method is both fast and space efficient. It is fast because it relies on local updates to the multiresolution structure as terrain changes. It is much more space efficient than many previous approaches because the multiresolution structure can be extended on-line, to provide higher resolution terrain only where needed. Our approach is especially well-suited for applications like real-time off-road driving simulation involving large terrain areas with localized high-resolution terrain updates.

Key words: dynamic terrain, triangle bintree, multiresolution representation, view-dependent mesh, level of detail

1 Introduction

Many techniques have been developed for representation and efficient visualization of terrains and other surfaces. In particular, the recent development of view-dependent multiresolution methods has provided a strong advance over distance-based discrete level-of-detail and other simple methods aimed at minimizing rendered polygons.

Existing methods focus either on static terrains or on time-varying geometry where all changes are known prior to any rendering. In this paper, we present methods for representation and real-time visualization of *on-line dynamic* terrain. In on-line dynamic terrain, surface geometry, color, and material properties can change over time and the particular changes are not known *a priori*; this precludes the preprocessing approaches of many techniques, which build a multiresolution structure that is dependent on the initial terrain geometry.

Specifically, our approach to view dependent visualization on-line dynamic terrain is:

- *fast*. The approach extends the ROAM[6] algorithm, and is fast because local terrain updates require only local updates to the multiresolution structure.
- *space efficient*. Through on-line extension of the multiresolution structure only where needed, dynamic terrain applications can save an enormous amount of space over methods that "prepare for the worst" everywhere (even if they never need it).

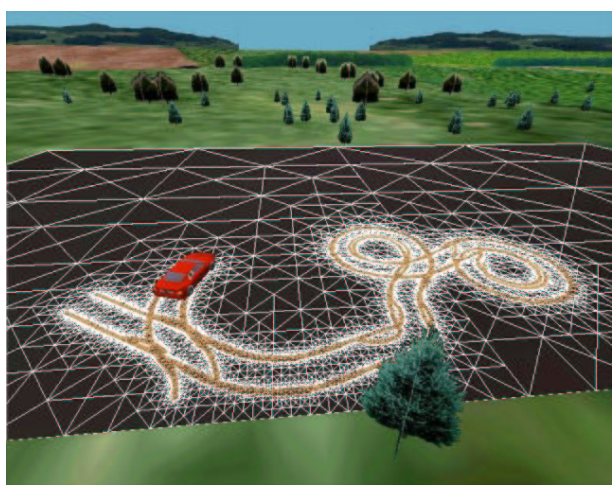

Figure 1: A screenshot from an off-road driving simulation application using our dynamic terrain algorithm.

We do not claim to have developed major new algorithmic results. But, we believe that support for on-line changes to geometry, color, and texture, represents an very important challenge for multiresolution methods, and that we have demonstrated useful practical early results in the area.

Our work was initially motivated by off-road driving simulation applications for The National Advanced Driving Simulator[23] — automobile and agricultural vehicle industry, as well as military, applications requiring real-time on-line simulation and visualization of vehicle-terrain interaction. Real-time determination of the effects of vehicle-terrain interaction, computed via tire-soil dynamics simulation, is a challenging computational problem. Furthermore, integration and correlation between very high resolution (non-visual) terrain databases required for tire-soil dynamics, and visual databases used for rendering is itself a challenging software systems issue (see [1]). In this paper, we address only the visualization of terrain changes determined through real-time tire-soil simulation or other processes.

2 Background and Related Work

Several good surveys on multiresolution surface representations exist, including Garland and Heckbert [7, 11], De Floriani et al. [5], Luebke [17], etc.

A multiresolution, or multiple level-of-detail (LOD), representation of a surface typically contains a sequence of approximations of the input surface, each with a different level of detail. These approximations are organized into a hierarchical structure, such as a DAG, a tree, or a forest, where the nodes represent parts of the approximations of various detail, with nodes closer to the top having lower detail. Edges relate parts from different approximations. In addition to the hierarchical structure, an *error metric* is also given, to use in measuring the deviation, or error, of approximation meshes in the hierarchy from the input surface. In some *view dependent* algorithms, screen-space error is used, which measures the size of the geometric approximation error projected onto the screen. The world space error, from which the screen-space error is derived, is view independent. During each iteration, the hierarchy is "trimmed" to get a single approximation of the surface that satisfies the error criteria.

Multiresolution surface representations can be classified as either vertex hierarchies or face hierarchies.

- Face hierarchy models (FHMs) are constructed to indicate the relation between the faces from approximations of different LODs. Existing algorithms based on face hierarchies include Lindstrom and Pascucci [16], Lindstrom et al. [15], Scarlatos and Pavlidis [20], de Berg and Dobrindt [2], Gross, Gatti and Staadt [9], De Floriani et al. [4], Duchaineau et al. [6], etc. A general face hierarchy model, *multi-triangulation*, was presented in De Floriani [3].

- Vertex hierarchy models (VHMs) are built from the relation between vertices from different approximations. Usually, each node of the hierarchy corresponds to a set of vertices, and the set of vertices denoted by the children of that node are the vertices used to replace them in a more refined approximation of the input surface. Error measures can be associated to each node in the hierarchy. Since vertices alone cannot determine the approximation, additional information about how the surface is polygonalized is required. Vertex hierarchy methods include Rossignac and Borrel [19], Luebke and Erikson [18], Hoppe [12, 13, 14], Xia and Varshney [24], and Garland and Heckbert [8].

Shamir, Pascucci and Bajaj [22] presented an approach for multiresolution dynamic surface visualization. Their method uses a DAG as the hierarchical structure and incrementally modifies the DAG as the surface deforms. The result is a super hierarchy, T-DAG, that combines the DAGs at all time-steps. T-DAG is capable of visualizing dynamic surfaces with arbitrary changes, including changes in topology and connectivity. The approach is not well suited for on-line updates due to the relatively high cost of T-DAG modification.

3 Real-time Visualization of Dynamic Terrain

As mentioned above, many terrain visualization algorithms rely on an assumption of static surface geometry to create, during a preprocessing step, a multiresolution structure from which efficient rendering may be done at run time. The assumption is invalid in on-line dynamic terrain applications,

To support on-line dynamic terrain, one must construct the hierarchical structure from the initial input surface and then modify it to reflect any changes made to the input surface. In some algorithms, such as Schroeder et al. [21] and Hoppe [12], the approximation sequence or multiresolution structure is constructed through an optimization process so that each approximation simplifies the previous mesh in the sequence while increasing the approximation error as little as possible. When applied to height fields, such a process can be called data-dependent simplification because the organization of the precomputed multiresolution depends on the heights of the vertices. In data-dependent approaches, structure updates to account for terrain changes are generally quite costly.

In data-independent approaches, only the (x, y) values of the vertices affect the organization of the multiresolution structure. For example, sub-sampling a regular grid mesh is a data-independent process. Data-independent approaches are thus good for on-line dynamic terrain based on height fields; terrain updates do not necessitate multiresolution structure reorganization. The block-based face quadtree method presented in Lindstrom et

al. [15] and the ROAM algorithm in Duchaineau et al. [6], both use data-independent hierarchies.

In the following section, we present the general idea of extendible resolution terrain representation. Sections 5 and 6 then present the details of our dynamic terrain representation and visualization approach.

4 DEXTER: Dynamic EXTEnsion of Resolution

The methods described in Section 2 pre-construct hierarchies from which meshes of various detail level can be derived at run time. Even in view dependent methods, the hierarchy itself is fixed, and the highest detail available to approximate any part of the surface is pre-determined.

In on-line dynamic terrain applications, greater interest may be put on the deformed regions, requiring higher resolution there than on untouched regions. In some cases, the particular maximum deformed resolution requirement may be known ahead of time. But, terrain deformation is often sparse and the precise location and degree of the deformation is not known until run time. A preconstructed "prepared for the worst" hierarchy that represents, everywhere in the terrain, the high resolution required by potential terrain modifications, can be prohibitively and unnecessarily space inefficient. When only a small portion of terrain will be modified, additional levels of the hierarchy in regions of untouched terrain waste memory that could better be used to represent important areas of dynamic modification in even better detail.

Thus, instead of a fixed-hierarchy, we use a dynamically extendible hierarchy for multiresolution terrain representation. The initial hierarchy is created so that it satisfies the resolution requirement of the initial non-deformed terrain. The finest mesh constructed from the hierarchy may have different detail at different parts of the terrain, depending on the local ruggedness of the initial terrain and other attributes such as the variation in color and texture, etc. The hierarchy is not fixed, however; as terrain deformation takes place, the hierarchy is extended only where necessary. The dynamic extension of resolution provides additional levels of detail at the modified regions without wasting memory space representing untouched terrain at unnecessarily high resolution.

DEXTER is a simple but general idea that can be applied to enhance many multiresolution surface representation methods. For different methods, there are different issues that need to be addressed in order to use DEXTER. In this paper, we demonstrate the necessary modifications for our real-time ROAM-based terrain visualization method.

5 Dynamic Terrain Extension to ROAM

A uniformly spaced, axis-aligned grid of terrain posts is a compact and efficient way of representing terrain surfaces. This grid is the foundation of the hierarchical structure built for the purpose of a multiple level-of-detail representation. Every grid point can be used as a vertex in the mesh that approximates the terrain surface. The finest approximation mesh is obtained when all the grid points are present in the mesh. All the elements that make up that mesh are considered to have zero approximation error.

The ROAM algorithm is well suited for the task of real-time visualization of terrain surfaces represented by a regular grid. In order to use ROAM for dynamic terrain visualization, two extensions of the algorithm were carried out. First, necessary updates of mesh data are added in each iteration to reflect the deformation of the terrain. Second, run-time extension of the hierarchical structure, i.e. the DEXTER augmentation, is incorporated into the algorithm. In this section we present the basic dynamic terrain extension to ROAM. In Section 6 we present the modification to the terrain grid representation and to the ROAM algorithm in order to incorporate DEXTER.

5.1 A Brief Review of ROAM

Real-time Optimally Adapting Meshes (ROAM), first presented by Duchaineau et al. [6], is a terrain visualization algorithm that adaptively generates right isosceles triangle meshes to render the underlying regular terrain grid.

The hierarchical structure of ROAM is a binary tree of right isosceles triangles, which is established during the preprocessing stage. Each non-leaf triangle has two children, obtained by splitting the triangle with an edge that links its apex vertex to the midpoint of its base edge. Figure 2 shows how a bintree of four levels is formed by recursively splitting the triangles of higher levels. The root triangle (v_a, v_0, v_1) has a midpoint v_m at its base edge, and its two children, (v_m, v_a, v_0) and (v_m, v_1, v_a), are called its left child and right child, respectively.

To represent a square region, a pair of triangle bintrees is needed. The two root triangles (a *diamond* in ROAM terminology) should be of the same size, and share the base edge.

As in other hierarchical models, a trimming of the tree is required to obtain a mesh representation of the terrain. A view-dependent error metric is used as the criterion for trimming. During each frame, the trimming result from the previous frame is adjusted by moving the trim line further down toward the leaf nodes at some places and up toward the root at some others. Intuitively, some triangles in the previous mesh are replaced by its descendents in the bintree, and some others are replaced by their an-

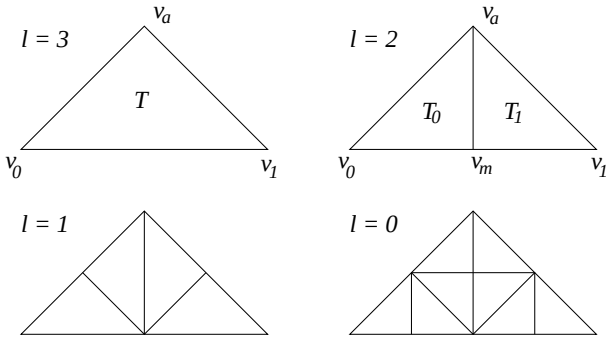

Figure 2: A triangle bintree of four levels.

cestors. Such adjustments are achieved by a sequence of *split* and *merge* operations on triangles. A *split* operation replaces a triangle with its two children in the bintree, while a *merge* operation replaces two siblings in the bintree with their parent. Figure 3 shows the effects of *split* and *merge* in aspects of both triangulation and trimming of the bintree. It is clear that a *split* is the reversal of a *merge*, and vice versa.

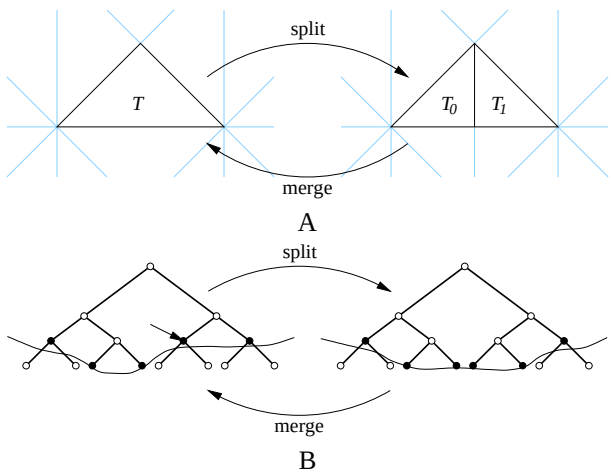

Figure 3: split *and* merge *operations: A. effects on triangle mesh; B. effects on triangle bintree trimming. Solid nodes in B represent the triangles selected for the approximation mesh.*

ROAM-based visualization is free of "crack" problems; mesh continuity is maintained by keeping track of the neighbors of each triangle, executing appropriate recursive *split* and *merge* operations.

5.2 Dynamic Terrain ROAM

The ROAM algorithm uses a pair of triangle bintrees as its data-independent hierarchy. Therefore, the basic structure of the bintrees does not need to be altered when the terrain is modified. However, the world space errors of the mesh elements – the triangles – need to be updated during each iteration. In the original ROAM algorithm, the computation of the world space errors is done in a bottom-up fashion. Therefore their updating needs to be performed bottom-up as well. Although the world space errors are associated to triangles, not vertices, they are derived from the geometry of the vertices. Which triangles are affected can be easily determined by checking which vertices have been modified. Update flags are therefore added to the data structure for the vertices, but not to that for the triangles. During the error updating procedure, the errors of leaf triangles remain zero, but leaf triangles with modified vertices need to notify their parents, whose errors need to be recomputed. This notification is passed on until the root triangles, and the error of every notified triangle is recomputed. Details of world- and screen-space error management for our algorithm are given in [10].

The vertices of the mesh triangles also need to be updated, e.g. the change of elevation, etc. In our implementation, the vertices are posts in the terrain cells (see Section 6.1), and the triangles in the bintrees use pointers to refer to them. Therefore by updating the posts, the vertices of the mesh triangles are automatically updated as well.

6 DEXTER Extension to ROAM

ROAM uses regular terrain grid as the basis of its triangle bintree structure. In order to accommodate DEXTER, the terrain grid representation needs to be modified. We introduce *terrain cells* to enhance the grid representation, and then discuss transition zones required to maintain mesh continuity after local extensions to the mesh hierarchy.

6.1 Terrain Cells

Starting from a uniform-resolution regular terrain grid, we can modify it to accommodate DEXTER. DEXTER allows the resolution to be extended at the deformed region of the terrain, therefore the grid resolution across the terrain may become non-uniform after such extensions. We divide the terrain surface into patches, each allowed to have its own grid resolution. To make the algorithm efficient, we restrict the shape of the patches to axis-aligned rectangles. We call these rectangles *cells*. The data structure for a cell should include its location, size, and grid resolution. The uniform resolution terrain grid can be considered as a special case – a grid with a single cell.

To extend the resolution of the terrain representation at designated regions, new cells that contain a terrain grid with desired resolution are created to cover those regions. Newly created cells overlap existing lower reso-

lution cells, and consistency must be maintained among terrain posts common to multiple cells. The properties of the new grid posts can be obtained from the input surface. If the input surface is simply represented by the initial terrain grid, an interpolation on that initial grid can be performed to get the properties for the new posts.

In our implementation, the terrain is initially represented by a single cell that covers the complete region. As terrain deformation takes place, new cells with higher grid resolution are created dynamically to cover the modified regions. The dynamic cells are all aligned to a cell grid and have uniform sizes. Furthermore, in the implementation demonstrated in this paper, the dynamic cells all have the same grid resolution. The grid resolution of dynamic cells is that of the initial cell times a user-specifiable power of two, so that the hierarchical structure in ROAM – the triangle bintrees – can be extended accordingly, which uses the grid posts of the cells as the vertices of the mesh triangles. Figure 4 shows a dynamic cell that doubles the initial grid resolution partially overlapping the initial terrain cell. The dynamic cells can be organized using a simple two-dimensional array.

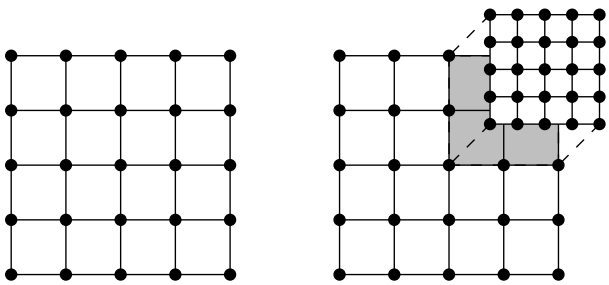

Figure 4: A dynamic cell overlaps the top right quadrant of the initial terrain cell.

The grid resolution may be increased by different amounts at different places. Different kinds of terrain modifiers produce shapes of various details. For example, footprints and tire tracks might require higher resolution representation than bulldozer tracks. This is easily supported - terrain cells form a hierarchy of multiple levels, where a dynamic cell may be partially coveredy by further even-high-resolution dynamic cells - but was not included in the implementation demonstrated on the accompanying video.

6.2 Extending ROAM's Mesh Hierarchy

As mentioned in Section 6.1, the triangle bintrees in ROAM are tightly associated with the terrain cells. As dynamic cells with a high grid resolution are created, the triangle bintrees are extended accordingly. The extension is achieved by recursively subdividing the leaf triangles of the initial bintrees whose areas of coverage are inside the area covered by the newly created cells. The number of levels of triangles that should be added to the bintrees are determined by the extent to which the grid resolution is increased via dynamic terrain cells. Suppose the grid resolution of the dynamic cells is 2^n times the initial resolution, then $2n$ levels of triangles should be added. Figure 5 shows the creation of a dynamic terrain cell matched by the extension of the triangle bintrees.

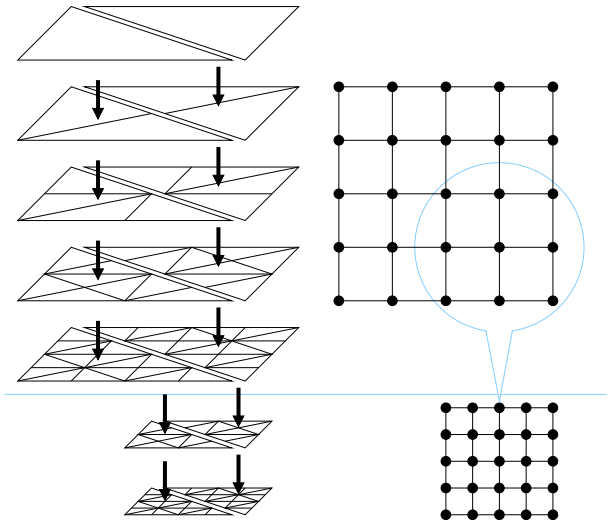

Figure 5: A 5×5 initial terrain grid is matched by a pair of triangle bintrees with 5 levels. The creation of a dynamic cell that doubles the grid resolution leads to the extension of the bintrees by 2 levels.

6.3 Transition Zones

After extending the mesh hierarchy, the triangle bintrees are no longer complete binary trees (except in the special case where the mesh was extended over the whole terrain). There are more levels of approximation available for the regions covered by triangles added to the bintrees at run time than for the rest of the terrain. This could cause problems in mesh continuity, as shown in Figure 6, where the mesh contains cracks around the two circled vertices. Here, the error criteria determine that triangles from newly added levels are needed to approximate the top left quadrant, but no triangles outside that quadrant can match them.

To preclude mesh discontinuities, *transition zones* are introduced. Given terrain region $\mathcal{R}_1$, where the highest grid resolution among all terrain cells that cover it is δ_1, and an adjacent region $\mathcal{R}_2$, whose highest grid resolution is δ_2, a transition zone is defined along the boundary between $\mathcal{R}_1$ and $\mathcal{R}_2$, in $\mathcal{R}_2$, if $\delta_1 > \delta_2$ and the higher

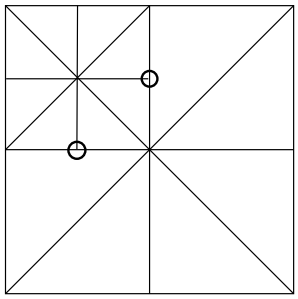

Figure 6: Mesh discontinuity caused by high level-of-detail mesh patches that cannot be matched elsewhere.

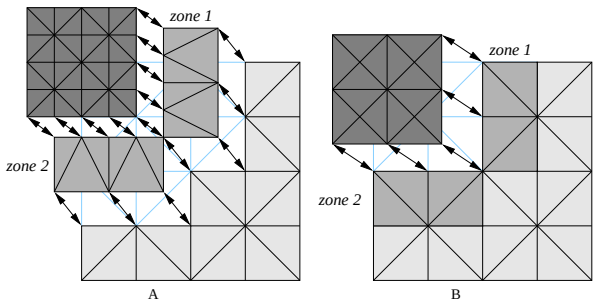

Figure 7: Possible transition zones between regions of different resolutions. Note that although they do preserve continuity, they use special non-right-triangles, and are not *the form of triangulation used in our algorithm.*

resolution $\mathcal{R}_1$ meshes cannot be matched by $\mathcal{R}_2$ meshes. This transition zone offers the basis for constructing special mesh patches that will match the higher resolution meshes for $\mathcal{R}_1$ with the highest resolution mesh available for $\mathcal{R}_2$. The transition zone needs to have the same high resolution as $\mathcal{R}_1$. In essence, the transition zone is the expansion of $\mathcal{R}_1$, and a simple way to do it is to expand the highest resolution cell that covers $\mathcal{R}_1$ to cover the transition zone as well, and at the same time expand the higher resolution meshes to include the special matching patches for the transition zone. In Figure 7, the top left quadrant of the terrain is of higher resolution than the rest, and two more levels of mesh are available there. Two transition zones are assigned outside the top left quadrant. In this particular case, a special mesh is necessary only to match the highest LOD mesh in the top left quadrant with the highest LOD mesh elsewhere, as shown in Figure 7.A; the second highest LOD mesh in the top left quadrant matches the highest LOD mesh outside perfectly, as in Figure 7.B. The result is, the highest LOD mesh for the top left quadrant is expanded to cover the transition zone, while the second highest LOD mesh is not.

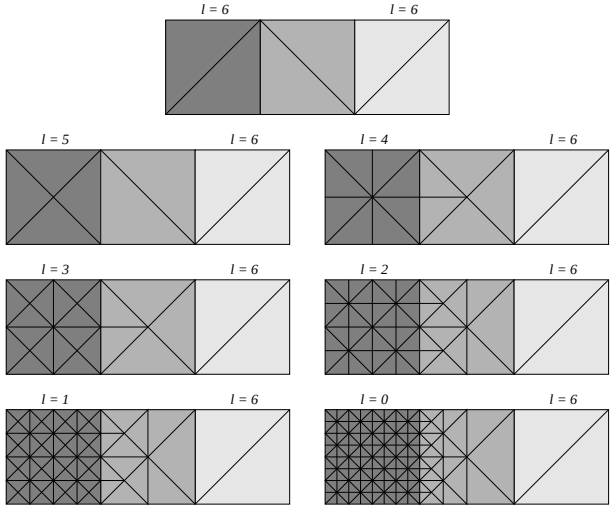

Figure 8: The high resolution meshes of six new levels of detail in the darkest region, from level 0 *to level* 5*, are connected to the mesh in the lightest region with help from the meshes in the transition zone.*

The transition zones shown in Figure 7 contain triangles that are not right isosceles triangles; this is not the way we prefer to extend ROAM. Instead, the special transition zone meshes can be constructed in the same fashion as the newly added high resolution mesh patches.

Consider the example shown in Figure 8. The six triangles in the first mesh of the sequence are from the lowest level mesh of an initial mesh hierarchy. They form three diamonds. The region covered by the leftmost diamond is now deformed, and its grid resolution is extended to be 2^3 times the original, thus introducing 6 new levels of detail in the triangle bintree. These 6 levels of detail are not available at the rightmost diamond. But as we can see, if we assign the region covered by the middle diamond as the transition zone, and extend it too by six levels, mesh continuity can be maintained no matter which of the six newly added levels of detail is used to approximate the region covered by the leftmost diamond. Not all triangles in the six levels of meshes in the transition zone are needed to maintain mesh continuity. However, it is simpler to extend the bintree fully there as well. Besides, it is likely that the transition zone will become a high resolution zone later and require the full expansion.

One may observe that in the above example the transition zone is made of just one pair of leaf triangles from the original bintree. In fact, if the resolution is extended by 2^n times, no matter how large n is, the transition zone always only needs to be as wide as a diamond made of a pair of original leaf triangles. Figure 8 shows the case for $n = 3$. The case for $n = 1$ is just the first three meshes;

$n = 2$, the first five. To prove the claim for any n, use mathematical induction. The claim is true for $n = 1$ as demonstrated in Figure 8. Assume the claim is true for $n = k$. For $n = k + 1$, extend the bintree by two more levels in the transition zone than in the case of $n = k$ to match the high resolution region. Next construct a mesh in the transition zone that connects the level 2 mesh in the high resolution region with the lowest level mesh available in the low resolution region. Here, that level is 2^{k+1}. The mesh detail near the boundary between the high resolution region and the transition zone is shown in Figure 9.A. Now split triangles T_1, T_2 and their base neighbors, and we get the mesh in Figure 9.B, which remains continuous. One can now notice that the highlighted region has the same configuration as the first graph in Figure 8, and edge (a, b) corresponds to the boundary between the transition zone and the low resolution region in Figure 8. Lowering the mesh level in the high resolution region to 1 and 0 are just like the second and third meshes in Figure 8, as shown in Figure 9.C and Figure 9.D. In both cases, in order to maintain mesh continuity, the refinement of the mesh in the transition zone can be limited to the left of (a, b). Therefore, the transition zone for $n = k$ is also sufficient for $n = k + 1$.

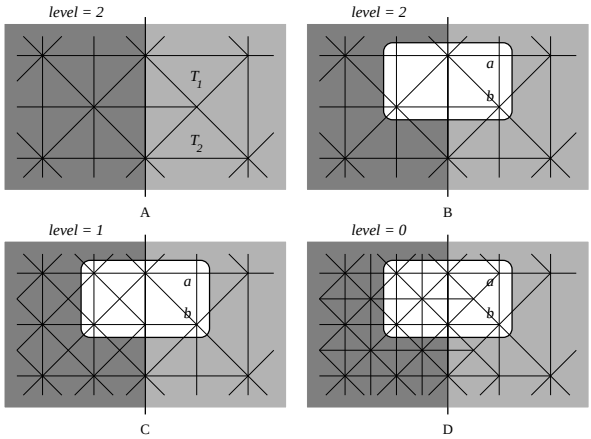

Figure 9: Mesh details near the boundary between the high resolution region on the left and the transition zone on the right.

Since transition zones surround a rectangular region, some occur at corners. These yield different transition trianglations but are still handled easily (see [10] for details).

Note that no special code needs to be implemented for the transition zones. The zones' high resolution combined with error values of zero ensure that the proper transition triangulation will be created as a natural part of the trimming process.

6.4 Updating Triangle Errors

World space errors need to be assigned to triangles added due to bintree extension. The errors of existing triangles can change when the underlying grid resolution is increased and the triangle bintrees extended. Most significantly, some leaf triangles in the initial bintrees now have descendents, so they are no longer part of the finest mesh available, and their errors are generally no longer zero. So, the errors of affected triangles need to be recomputed after hierarchy extension. A bottom-up approach is adopted, similar to the way the world space errors are computed during initialization.

New higher-resolution triangles in transition zones simply have zero error, since the terrain has not been (yet) been modified and the original coarser triangles still have zero error. Some of the high resolution triangles will ultimately be selected into a displayed mesh based not on their error values but because of the component of the algorithm that maintains mesh continuity.

6.5 DEXTER Algorithm Outline

Following is a general multiple LOD dynamic terrain visualization algorithm with DEXTER, applicable to all DEXTER methods that use terrain cells and transition zones[1].

begin
1. Initialize terrain cells
2. Initialize mesh hierarchy
3. **while** no exit signal **do**
   begin
4. deform terrain
5. add high resolution cells to deformed regions if necessary
6. assign transition zones around new cells
7. update mesh hierarchy, recompute errors
8. trim mesh hierarchy to obtain approximation mesh for the complete terrain
9. render the approximation mesh
   end

end

7 Implementation Results

The accompanying video shows DEXTER with ROAM running smoothly in real time on an 866MHz Pentium III with a GeForce2 graphics card and 384MB of RAM. Figures 10, 11, 12, and 13 show screenshots from the same application.

[1]Some multiple LOD terrain visualization methods do not require terrain cells or transition zones when extended with DEXTER, e.g certain methods that do not use regular grids or vertex hierarchy methods that perform triangulation at run time.

The algorithm was implemented as a research prototype with no code tuning or low-level code optimization. Some basic timing results are included in the following paragraphs. The results were recorded from identical 1800 frame portions from the middle of runs using the same vehicle path and scene shown in the video. The error tolerance, τ, was .04 in each case. The terrain for the test runs is perfectly flat at the start. Thus, no loss of detail occurs in representation of the initial unmodified terrain, eliminating differences in the number of triangles that would result from different basic terrain grid sampling resolutions for the regular ROAM and the DEXTER version. Note that the average frame rates include significant non-visualization-related simulation time.

Run 1: ROAM without DEXTER

.

The grid size was 513-by-513, so the finest mesh consists of $512*512*2 = 524,288$ triangles, while the triangle hierarchy has $2+4+8+...+524,288 = 1,048,574$ triangles.

```
Error threshold tau = 0.04
Total time:                    51.47s
 - rendering:                  25.92s
 - DT rep. and error update:   11.43s
 - vehicle/soil simulation:    14.12s
Avg. tri count:           4140
Avg. frame rate:          35.0
```

Run 2: ROAM with DEXTER

.

The initial grid size was 129-by-129, and the maximum grid size, through DEXTER, was 513-by-513, the same as in Run 1 without DEXTER. In this case, however the initial hierarchy has only $2+4+8+...+128*128*2 = 65,534$ triangles, $1/16th$ that of Run 1.

```
Error threshold tau = 0.04
Total time:                    52.18s
 - rendering:                  25.45s
 - DT rep. and error update:   10.61s
 - vehicle/soil simulation:    16.12s
Avg. tri count:           4131
Subtrees created:         2186
Avg. frame rate:          34.5
```

2186 subtrees created means a total of $2186 * 30 = 65,580$ triangles were added to the hierarchy, leaving the final total to be $65,534 + 65,580 = 131,114$. Compare this to $1,048,574$ in ROAM without DEXTER.

The two runs result in nearly identical visual appearances, and the triangle counts are also very close. This is expected because the resolution of the regular ROAM and the maximum resolution of the DEXTER version are the same, and the same error threshold is used. The frame rates are also very similar, which means the extra work on the DEXTER version only accounts for a small portion of the total computation.

Run3: ROAM with DEXTER

.

In this run the initial grid size was again 129-by-129. However, the maximum grid size is increased to 1025-by-1025. The error threshold is kept at 0.04.

```
Error threshold tau = 0.04
Total time:                    86.65s
 - rendering:                  49.47s
 - DT rep. and error update:   22.51s
 - vehicle/soil simulation:    14.67s
Avg. tri count:           9681
subtrees created:         2186
Avg. frame rate:          20.8
```

As expected, the triangle count increases and the frame rate drops, when compared to the previous two runs. But the detail available in the tire tracks is much higher because highest resolution triangles are smaller than before, and are therefore better equipped to represent the curves along the tracks. The number of subtrees created was the same as in Run 2 because the vehicle path is identical. The subtrees are taller than in Run 2, though, so a total of $21,886 * 126 = 275,436$ triangles were added. Still, the final total of $65,534 + 275,436 = 340,970$ is much smaller than Runs 1's 1,048,754. And, the maximum resolution of the grid is four times higher. The test runs clearly exemplify significant reduction in data size without a large speed penalty.

8 Conclusion and Continuing Work

We described methods that enable practical real-time visualization of on-line dynamic terrain. Our approach provides multiresolution view-dependent representation and display of dynamic terrain by extending ROAM with efficient hierarchy updates as terrain deforms, and using DEXTER to provide only-where-needed memory efficient resolution extension.

This paper directly addressed only the geometric aspects of dynamic terrain. Interesting additional research problems remain in dynamic terrain representation and visualization, particularly related to color and texture issues. Error measures in view-dependent multiresolution techniques have largely concentrated on geometric error, but can account for color and texture as well. Like the situation with geometry, however, they are more difficult to

handle (and perhaps even more important) in a dynamic terrain setting. Some possible solutions and additional research directions are discussed in He[10].

It is clear that our approach is quite effective for applications, such as off-road driving simulation, that require only small localized terrain updates. We have not yet carefully assessed the method's practical effectiveness for applications involving larger deformations of more extensive terrain areas.

Acknowledgements

This work was supported in part by Automotive Research Center Contract Number DAAE07-98-3-0022.

References

[1] J. Cremer, Y. He, and Y. Papelis. Dynamic terrain for real-time ground vehicle simulation. In *Proceedings of the Image 2000 Conference*, pages 98–105, July 2000.

[2] M. de Berg and K. T. G. Dobrindt. On levels of detail in terrains. Technical Report UU-CS-1995-12, Department of Computer Science, Utrecht University, 1995.

[3] L. De Floriani. A pyramidal data structure for triangle-based surface description. *IEEE Computer Graphics and Applications*, 9(2):67–78, March 1989.

[4] L. De Floriani, P. Magillo, and E. Puppo. Efficient implementation of Multi-Triangulations. In *Proceedings IEEE Visualization '98*, October 1998.

[5] L. De Floriani, P. Marzano, and E. Puppo. Multiresolution models for topographic surface description. *The Visual Computer*, 12(7):317–345, August 1996.

[6] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich, and M. B. Mineev-Weinstein. ROAMing terrain: real-time optimally adapting meshes. In *Proceedings IEEE Visualization '97*, pages 81–88, 1997.

[7] M. Garland and P. S. Heckbert. Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS 95-181, Department of Computer Science, Carnegie Mellon University, 1995.

[8] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 209–216, 1997.

[9] M. H. Gross, R. Gatti, and O. Staadt. Fast multiresolution surface meshing. In *Proceedings IEEE Visualization '95*, July 1995.

[10] Y. He. *Real-time visualization of dynamic terrain for ground vehicle simulation*. PhD thesis, The University of Iowa, December 2000.

[11] P. S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. In *Multiresolution surface modeling (SIGGRAPH '97 Course notes #25)*. ACM SIGGRAPH, 1997.

[12] H. Hoppe. Progressive meshes. *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 99–108, 1996.

[13] H. Hoppe. View-dependent refinement of progressive meshes. *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 189–198, 1997.

[14] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proceedings IEEE Visualization '98*, October 1998.

[15] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. Real-time, continuous level of detail rendering of height fields. *Computer Graphics (SIGGRAPH '96 Proceedings)*, pages 109–118, 1996.

[16] P. Lindstrom and V. Pascucci. Visualization of large terrains made easy. In *Proceedings IEEE Visualization '01*, pages 363–370, October 2001.

[17] D. Luebke. A survey of polygonal simplification algorithms. Technical Report TR97-045, Department of Computer Science, University of North Carolina at Chapel Hill, 1997.

[18] D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 199–208, 1997.

[19] J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering complex scenes. In B. Falcidieno and T. Kunii, editors, *Modeling in Computer Graphics: Methods and Applications*, pages 455–465. Springer-Verlag, Berlin, 1993.

[20] L. Scarlatos and T. Pavlidis. Hierarchical triangulation using cartographic coherence. *CVGIP: Graphical Models and Image Processing*, 54(2):147–161, March 1992.

[21] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. *Computer Graphics*, 26(2):65–70, July 1992.

[22] A. Shamir, V. Pascucci, and C. Bajaj. Multiresolution dynamic meshes with arbitrary deformations. In *Proceedings IEEE Visualization '00*, pages 423–430, October 2000.

[23] The National Advanced Driving Simulator. URL: http://www.nads-sc.uiowa.edu.

[24] J. Xia and A. Varshney. A dynamic view-dependent simplification for polygonal models. In *Proceedings IEEE Visualization '96*, pages 327–334, 1996.

Figure 10: ROAM without DEXTER. Resolution 513×513*, on* $44m \times 44m$ *square region.*

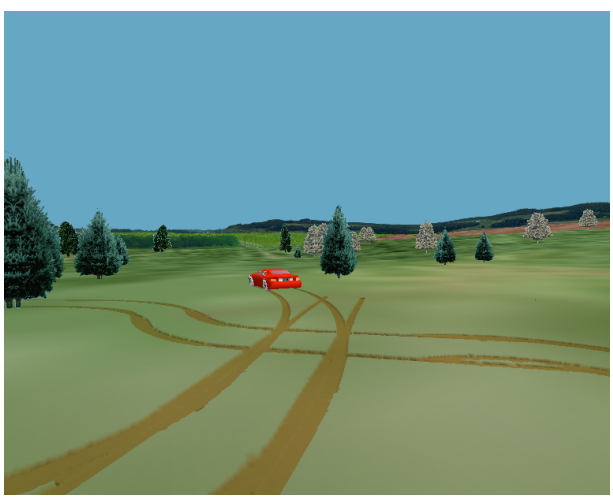

Figure 11: ROAM with DEXTER. Basic resolution 129×129*, extended* 2049×2049*, on* $44m \times 44m$ *square region.*

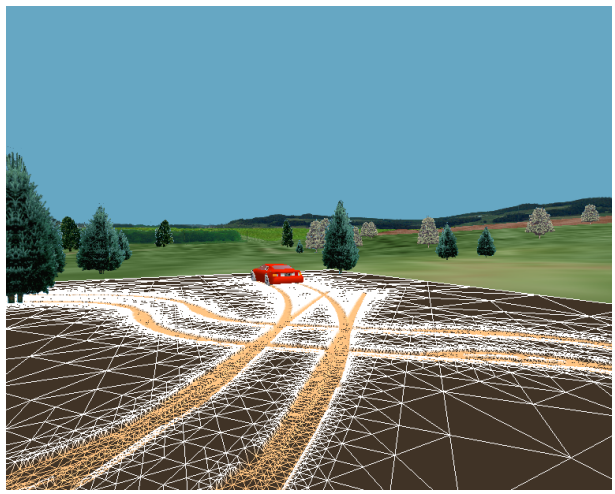

Figure 12: ROAM with DEXTER. Basic resolution 129×129*, extended* 2049×2049*, on* $44m \times 44m$ *square region.*

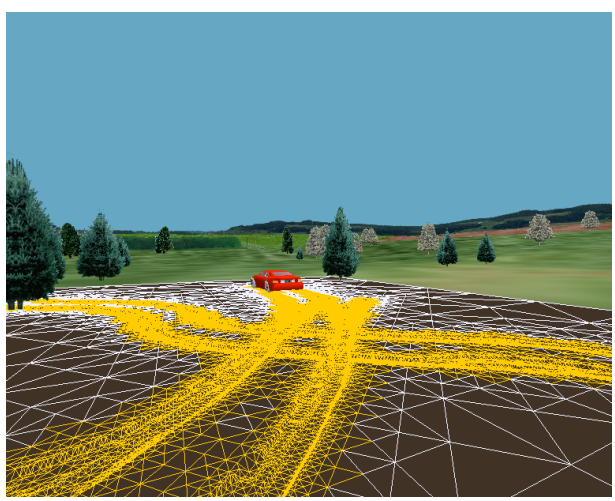

Figure 13: ROAM with DEXTER. The triangles shown in yellow are from cells added to extend the initial hierarchy.