Cached Geometry Manager for View-dependent LOD rendering

Roberto Lario
Universidad Complutense
Madrid, Spain
rlario@dacya.ucm.es

Renato Pajarola
University of California Irvine
USA
pajarola@acm.org

Francisco Tirado
Universidad Complutense
Madrid, Spain
ptirado@dacya.ucm.es

ABSTRACT

The new generation of commodity graphics cards with significant on-board video memory has become widely popular and provides high-performance rendering and flexibility. One of the features to be exploited with this hardware is the use of the on-board video memory to store geometry information. This strategy significantly reduces the data transfer overhead from sending geometry data over the (AGP) bus interface from main memory to the graphics card. However, taking advantage of cached geometry is not a trivial task because the data models often exceed the memory size of the graphics card. In this paper we present a dynamic Cached Geometry Manager (CGM) to address this issue. We show how this technique improves the performance of real-time view-dependent level-of-detail (LOD) selection and rendering algorithms of large data sets. Alternative caching approaches have been analyzed over two different view-dependent progressive mesh (VDPM) frameworks: one for rendering of arbitrary manifold 3D meshes, and one for terrain visualization.

1. INTRODUCTION

The functionality and speed of graphics hardware has increased significantly in last few years, making the GPU a programmable stream processor with sufficient power and flexibility to perform intensive calculations. Despite advances in the graphics hardware, the data transfer from main memory to the graphics card remains the major bottleneck [HCH03]. This restriction prevents the full exploitation of the potential computational horsepower of the GPU and introduces significant overhead in short data transfers [THO02].

View-dependent level-of-detail (LOD) algorithms can significantly reduce the amount of data transfer as the geometric scene complexity is adaptively minimized using a view-dependent error metric [LRC03]. The adaptive nature of such methods introduces constant but infrequent and small geometric changes between consecutive frames. Our goal is to take advantage of this fact using the video memory of modern consumer graphics hardware as geometry cache. The rendering performance can greatly be improved if the geometric data of a given scene is stored in video memory. However, the limited size of available video memory restricts the complete caching of big data models. The use of view-dependent LOD algorithms can provide a solution to this problem because the geometric information required for rendering a scene at a certain LOD is in general only a small fraction of the full resolution model. This visible portion of geometry information can be cached on the graphics card using video memory (see Figure 1) and is updated every frame when the viewpoint location of the camera or the resolution is changing. In order to efficiently handle the constantly occuring video memory updates, a *Cached Geometry Manager* (CGM) is needed. The continuous adaptive LOD changes guarantee that only a small amount of the cached geometry in the video memory has to be updated between consecutive frames.

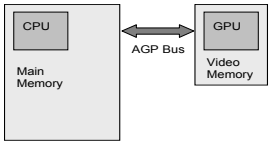


Figure 1: CPU/GPU communication diagram.

In this paper we describe several strategies to implement an efficient geometry-cache manager. Two *view-dependent progressive mesh* (VDPM) frameworks are used to test the proposed techniques and to show the speed-up in rendering performance when applied to a general view-dependent LOD algorithm. The first framework is *FastMesh* [Paj01], it uses an efficient view-dependent and adaptive LOD method for rendering arbitrary 3D meshes in real-time. The general concepts of this framework are common to most similar VDPMs, e.g. such as [XV96], [Hop97], [LE97], [DMP97] or [KL01]. The second framework is *QuadTIN* [PAL02], an efficient quadtree-based triangulation approach for irregular terrain height-fields that provides fast quadtree-based adaptive triangulation, view-dependent LOD-selection and real-time rendering. Many interactive terrain

WSCG 2005 conference proceedings, ISBN 80-903100-7-9
WSCG 2005, January 31-February 4, 2005
Plzen, Czech Republic.

visualization systems, e.g. such as [SS92], [LKR96], [DMP96], [Pup96], [Paj98], [BAV98] or [EKT01], exhibit a similar top-down LOD triangulation and rendering approach.

The remainder of the paper is organized as follows: Section 2 presents a very brief overview of related work. Section 3 describes the Cached Geometry Manager. In Section 4 the two VDPM frameworks are presented to test the CGM approach. Experimental results are presented in Section 5 and Section 6 ends the paper with some conclusions.

2. RELATED WORK

Despite the extensive work on level-of-detail (LOD) techniques [LRC03], only very few methods that use cached geometry have been proposed recently. One possible reason for this lack is that only recent generations of graphics cards allow the application program to manage large amounts of video memory systematically and dynamically for storing geometry. In [Lev02] a terrain rendering algorithm is presented that operates on clusters of cached geometry called aggregate triangles. The dynamically generated aggregate triangles are kept in the geometry cache for several frames to improve rendering performance. A similar concept is followed in [CGG03a], [CGG03b] where a LOD hierarchy of simplified height-field triangle patches is generated in a pre-process. At run-time the appropriate LOD triangle patches are selected for rendering and a LRU strategy is used for caching. In [LPT03] square patches of a quadtree-based hierarchical terrain triangulation are used for fast rendering and caching in video memory. A common limitation of the above methods is that they are restricted special-purpose solutions for terrain rendering and not applicable in general to other VDPM frameworks. In contrast, the concepts presented in this paper are directly applicable to a wide range of VDPM frameworks. A remarkable approach to provide seamless geometric LODs is provided by GLOD [CLD03]. It allows advanced users to define discrete LOD objects as well as specify the use of video memory for patches of the geometry. In contrast to GLOD, the proposed Cached Geometry Manager interacts directly with VDPM frameworks that dynamically generate continuously adaptive LOD meshes and provides transparent use of video memory.

3. CACHE GEOMETRY MANAGER

Most view-dependent simplification frameworks represent the geometry in a hierarchical data structure called vertex hierarchy (see Figure 2). The nodes located near the root correspond to low-resolution vertices while those located farther away represent high-resolution detail vertices. The vertex hierarchy is dynamically queried to perform a view-dependent LOD simplification for each frame. A front of active nodes divides the current nodes used to generate the simplified scene from the rest. This frontier can continuously and incrementally by updated between rendered frames.

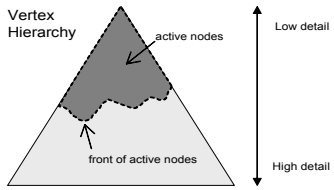


Figure 2: Vertex hierarchy diagram.

One can observe that by far most of the vertices of a scene remain active between consecutive frames and just a small fraction changes its state. Hence most vertices can be stored and kept continually in video memory in order to improve rendering performance. Each frame only few vertices require a read operation from main memory, to transfer to video memory, when they change their state from inactive to active. Note that a remove operation is needed when the video memory is full and new vertices have to be added. Inactive but cached vertices are the prime candidates to be deleted from video memory in this case. A video memory manager is required to carry out these operations.

Vertex Arrays

Indexed vertex buffers or indexed vertex arrays (IVA) in OpenGL, are the best way to take advantage of modern graphics accelerator (see section 11.4.5 in [MH02]). The application puts the data into specific buffers and gives the pointers to the driver, which accesses the data directly. Hence vertex arrays need much fewer OpenGL function calls for rendering than the classic immediate mode vertex submission (using glBegin()/...glVertex().../glEnd() blocks). In [Mar00], several methods to optimize submission of vertex data in OpenGL are described. Our CGM takes advantage of vertex arrays in combination with the OpenGL extension NV_vertex_array_range [Kill99]. The order and positions of vertices is different in the cached IVA from the main memory IVA. Thus the vertex indices of an indexed triangle mesh must be remapped accordingly for rendering. However, the rendering speedup will compensate for this extra re-indexing required by a dynamic CGM.

CGM Strategies

In this and the following section we describe three basic caching strategies to implement the video memory manager. These three strategies are going to be discussed for two variants of VDPM frameworks in order to cover the range of applications: those which calculate an explicit front of active nodes by incremental updates between consecutive frames, and those which implicitly define the active front by selecting the active nodes top-down for each frame (see Figure 2). In this section we first discuss the more

general case of implicit active front VDPM frameworks. Note that a non-explicit front does not mean it does not exist, in fact the front always implicitly exists in any view-dependent LOD framework. The implicitly-defined refers to the behavior of the VDPM framework that has no other information than if a vertex is selected or not for each rendered frame. From here on we will refer to both video memory and geometry cache as equivalent concepts. The basic two tasks of the cache manager are: (1) to determine that a vertex is already resident (cached) in the video memory, and (2) to find and use an open slot in the cache to store a new vertex. Task (1) can efficiently be determined by a cross indexing: each vertex in main memory has a field that indicates the cache index where it was last stored, and each cache slot has an index field indicating which vertex it stores. Hence if both indices coherently cross-link the same vertex then it is already cached and ready for use. More complicated is task (2) for which we describe viable strategies below.

First-Available Strategy (FA): This simple strategy uses the video memory as a linear list of slots with flags. This list is incrementally traversed from the beginning to the first non-used slot (First Available) every time a new vertex must be cached. Then this slot is marked as used. The process continues while there are vertices to cache, and a pointer is moved from the head to the end to search for the next available open slot. Owing to the fact that the list of slots is sequentially traversed this strategy can be implemented using a simple array, as illustrated in Figure 4a). Each slot is considered used when it stores a vertex used in the current or last frame. This policy considers the fact that it is very likely that a vertex used in frame i will also be required in frame i+1. Hence each slot flag is an integer counter which stores the last frame in which that cached vertex was used. This strategy is simple to implement, but has one potential drawback: unused slots near the beginning of the list will immediately be overwritten when a new vertex has to be cached while unused slots at the end may cache an unused vertex for a long time. This bias of reusing cache slots based on their position is not necessarily the best solution. At the expense of more complexity, the next strategy addresses this problem.

LRU Strategy: As mentioned above, the FA strategy considers any empty slot in the cache as equally good. If a slot has not been used in the current or last frame it is considered available. However, there is an intuitive reason that more recently used vertices are more likely to be used again than vertices that have not been used for a long time. Hence a more refined policy is to take into account the age of the unused slots and use a last-recently-used (LRU) strategy. The LRU parameter is directly obtained from the frame counter associated with each slot. One possible data structure to make use of this strategy is a doubly-linked-list. Two pointers (head and tail) are needed for the proposed implementation as shown in Figure 4b). The head points to the youngest slot, and the tail points to the oldest slot. New vertices are cached in the slot pointed to by tail which is then moved to the head. Reused slots of rendered vertices already in cache are simply moved from their current position in the linked list to the head. Consequentially, unused slots automatically move towards the tail which always points to the oldest slot entry. Note that these operations do not imply a displacement of the actual vertex data in video memory, it is just a mechanism for the cache manager to maintain access to the last-recently-used open slot. Each slot in this linked list corresponds to a fixed memory location in the cache.

LRU + Error-PriorityQueue Strategy: Figure 2 shows clearly that the vertices near the top of the hierarchy are more significant as they correspond to coarser LOD information. Consequently, these vertices are included in the mesh representation before any vertices of finer LODs. Therefore, for a new vertex it is more suitable to choose among the empty slots the one that corresponds to an old vertex which represents a fine level-of-detail. In order to add this new feature to the CGM we propose to categorize the age of the unused slots and introduce a priority-queue for the oldest-category vertices. The oldest category vertices are naturally and compactly stored at the end of the LRU list as described above. Hence as shown in Figure 4c) we only manage this last section of the LRU list in a priority-queue with the LOD error-metric parameter as key. Note that it is not advisable to choose a big priority queue size since this data structure is more costly than the doubly-linked-list of the simple LRU approach.

As with the LRU approach, reused slots are moved from the current location to the head and unused slots slowly sink towards the tail. The tail marker also indicates the bounds of the oldest-category. Thus elements at the tail are moved to the priority-queue as soon as their age has reached a certain limit and the priority-queue is not at maximal capacity. When a new vertex has to be inserted into the cache, the top slot of the priority-queue is used and moved to the head.

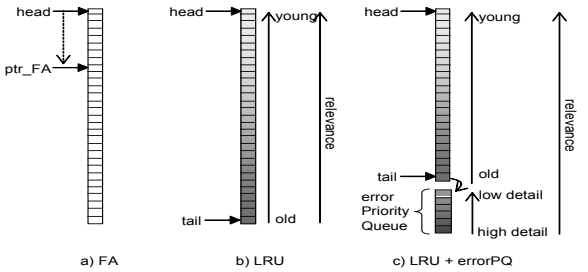


Figure 3: Data structures for the CGM strategies.

CGM Strategies for Front-Frameworks

The strategies described in the previous section can be refined if the VDPM framework has explicit knowledge of which vertices have been removed from and which vertices have been added to the current LOD triangle mesh. Thus if the change from active to inactive, and vice-versa in Figure 2, is explicitly observable by the application. This feature is typical in LOD systems that maintain an explicit active front for the current frame and update this front incrementally as illustrated in Figure 5. For a new frame, the newly activated vertices are called added (+) vertices, and those deactivated are called removed vertices (-). For each frame the added vertices have to be inserted into video memory, if not already cached from previous frames, while the removed vertices (may) remain cached but change their slot flag to be unused. Note that the removed vertices have always just been active in the previous frame.

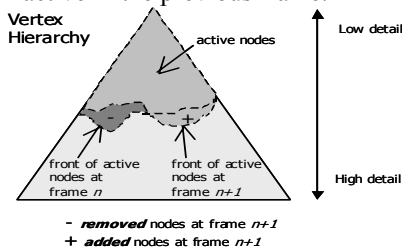


Figure 4: Vertex hierarchy of a front-framework.

FA Strategy for front-framework: This strategy, while obviously suboptimal when information about both added and removed vertices is explicitly provided by the LOD system, applies without changes to explicit-front frameworks.

LRU Strategy for front-framework: The LRU policy described previously can be improved using a third pointer, called frontier in Figure 6 that divides the active slots from the inactive ones.

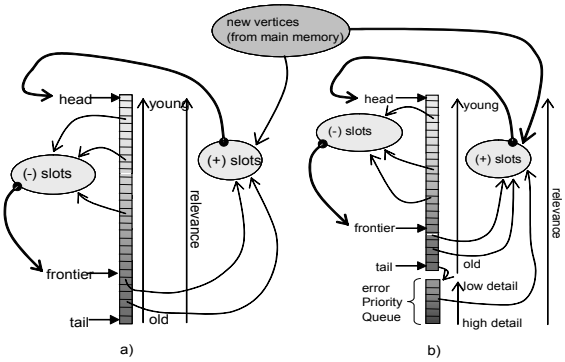


Figure 5: a) LRU for Front-Frameworks. b) LRU + errorPQ for Front-Frameworks. (-) slots of removed vertices. (+) slots of added vertices.

The slots of removed vertices are moved to just below the frontier while slots of added vertices change their position from the tail to the head. Advantage can be taken for vertices that were already active and cached in the previous frame because their corresponding slot in the LRU list is not affected by any move operation in the linked list. Note that these reused vertices are by far the largest fraction of active vertices. Therefore, compared to the basic LRU cache algorithm, linked-list operations are limited to the few removed and added vertices in front-frameworks.

LRU + Error-PriorityQueue Strategy for front-framework: Following the same idea expressed above, a priority queue may consider the LOD error-metric to choose the least relevant available slot. As with the LRU strategy also the LRU + priority-queue strategy can be refined if the sets of removed and added vertices are explicitly known and a fontier pointer separates the active from the inactive slots. As illustrated in Figure 7, the slots of removed (-) vertices are moved to behind the frontier pointer, while the slots of added (+) vertices are moved from the priority queue to the head. The slots between the frontier and the tail pointer are candidates to be transferred to the priority-queue if it is not at maximum capacity. Again, advantage is taken by not touching any of the slots of vertices that remain active between consecutive frames.

4. TARGET FRAMEWORKS

Two different view-dependent LOD frameworks have been analyzed for testing the proposed Cached Geometry Manager: FastMesh [Paj01] and QuadTIN [PAL02]. As mentioned in the introduction, the former is a VDPM system for rendering arbitrary manifold 3D meshes, and the latter is for rendering terrain height-fields. Both frameworks are briefly explained in the following sections before we provide experimental results.

Arbitrary Mesh Render System: FastMesh

FastMesh [Paj01] is an efficient hierarchical multiresolution triangulation framework based on a half-edge triangle mesh data structure and edge-collapse operations. Optimized computation of view-dependent error metrics within the framework provide conservative LOD error bounds. FastMesh is efficient both in space and time cost, and it spends only a fraction of the time required for rendering to perform the view-dependent LOD error calculations and dynamic triangle mesh updates. Conceptually it follows exactly the diagram shown in Figure 5 as many other similar approaches such as [XV96], [Hop97], [LE97], [DMP97], [KL01] do.

One of the main features of FastMesh is the explicit calculation of the front of active nodes, which is obtained for any frame by incremental changes to the previous frame. Hence FastMesh directly provides information about vertices added or removed from the front for every frame and falls into the category of front-frameworks described above in Section 3.3. The initial implementation of FastMesh rendered the mesh

as a list of active triangles in immediate mode vertex submission which has been changed for this project to an indexed vertex array (IVA) rendering mode. Experimental results using the front-framework CGM strategies described in Section 3.3 are given for this VDPM framework in Section 5.1.

Terrain Rendering System: QuadTIN

QuadTIN [PAL02] is an efficient quadtree-based terrain triangulation approach. It provides fast quadtree-based adaptive triangulation, view-dependent LOD-selection and real-time rendering. Its fundamental quadtree-based triangulation method and top-down vertex selection and rendering approach is similar to many other terrain visualization systems such as [SS92], [LKR96], [Paj98], [BAV98], [EKT01]. In contrast to other approaches, however, QuadTIN presents an efficient quadtree-based triangulation approach over irregular input point sets with feature adaptive sampling resolution while preserving a regular quadtree multiresolution hierarchy over the irregular input data set. Although the resulting quadtree hierarchy is not balanced, it conforms to the restricted quadtree constraints [SS92]. Additional information such as geometric approximation error, bounding spheres and normal cones are used for view-dependent LOD-triangulation and rendering. Like most other terrain visualization systems, QuadTIN selects the active vertices for a LOD of a particular viewpoint in a recursive top-down traversal for each frame. Hence QuadTIN belongs to the category of VDPMs with implicitly defined active front and does not provide explicitly the removed or added vertices between two consecutive frames. Experimental results using the basic CGM strategies given in Section 3.2 are reported in Section 5.2 for this VDPM framework.

5. EXPERIMENTAL RESULTS

Experimental results were performed on a 3.0 GHz Pentium 4 with 1GB RAM using an NVIDIA GeForceFX 5200 graphics card. For all scenes a 45º vertical field-of-view camera followed several test trajectories as described below.

FastMesh Results

The models used with the FastMesh rendering system are given in Table 1. The rendering experiments were averaged over 1000 frames in a window of 800 x 800 pixels using an error tolerance equal to one and a half pixels (projective tolerance of geometric error projected on screen).

			
model	**hand**	**dragon**	**happy**
vertices	327323	437645	543652
faces	654666	871414	1087716

Table 1: 3D models used with FastMesh.

Three different camera trajectories have been analyzed to examine the impact of the Cached Geometry Manager within the FastMesh framework as illustrated in Figure 8:

- Circular camera trajectory.
- Small camera rotations.
- Straight line camera trajectory.

These camera movements are very common as they are typical movements observers normally execute to explore a 3D object. The camera is pointing to the center of the model in all the trajectories.

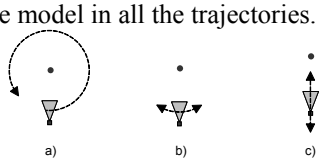


Figure 6: 3D object rendering camera trajectories: a) circular trajectory. b) small rotations. c) straight line trajectory (zoom in/out).

The chosen CGM size (slot-list length) was 2^{16} = 65536 because all models required at least 2^{15} = 35768 vertices to render from every tested viewpoint. Of course, in this context no LOD mesh can have more vertices than available in the geometry cache, and the application program must make sure that the best LOD for a limited number of vertices is selected. If this mesh exceeds the cache size then the application should disable the CGM and render the mesh in normal mode, or possibly render the mesh using the CGM in multiple passes. We are only studying the effect of the CGM in this paper and do not address the latter issues in this work. In the experiments, the size of each vertex element is 36 bytes, consisting of: 3 floats (position) + 3 floats (normal) + 3 floats (color).

We have focused the numerical results on the biggest model (happy) to avoid excessive and repeated information. Statistical data for the other models is given in Table 3. Figure 9 shows the per-frame timing results (in miliseconds) of happy model for the three different camera trajectories. The total time per frame has been divided into three parts: the rendering time, the time needed to construct the vertex array (build IVA), and the time required to perform the error metrics and updating the LOD mesh (others). Note that a brute-force rendering of this model using a standard immediate mode vertex submission achieved less than 4 frames per second, or equivalently required more than 250ms per frame. Our CGM techniques only affect the vertex array construction and rendering time but not any other tasks of the VDPM framework. In particular, the LOD-computation and vertex selection is not affected by the CGM and thus limits the overall speed-up with respect to the observable frame-rate. Hence we focus on the speed-up achieved only within the rendering part of a VDPM framework which is the sole target of the CGM.

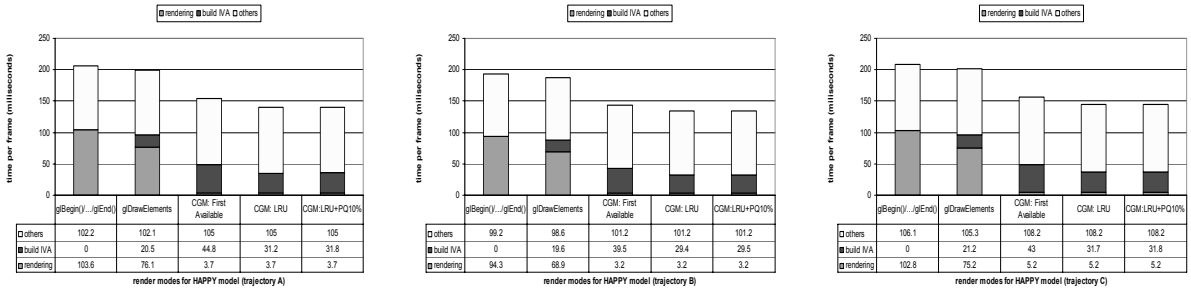


Figure 7: Per frame timing results (in miliseconds) for the Happy model for: a) circular camera trajectory, b) small camera rotations and c) straight line camera trajectory. The build IVA time contains the time consumed by the cache memory manager in modes where the CGM is enabled.

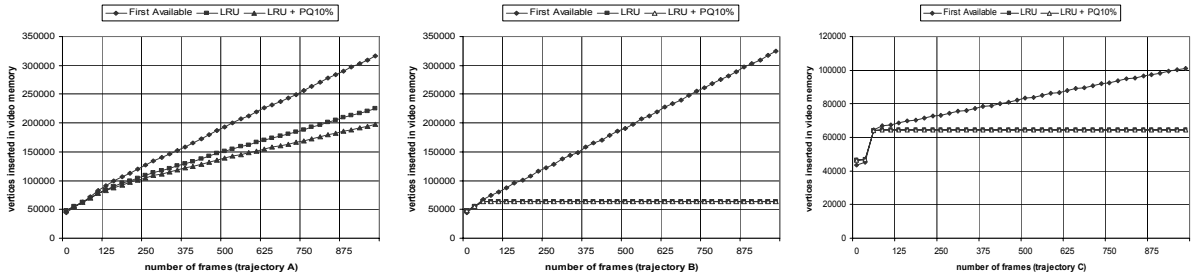


Figure 8: Vertices inserted into video memory for the Happy model for: a) circular camera trajectory, b) small camera rotations and c) straight line camera trajectory.

As we mentioned in section 4.1, the initial version of FashMesh traverses a linked-list of triangles to render the model in immediate mode vertex submission. The standard IVA mode is still faster than the previous despite the transformation from a linked-list of triangles to an indexed triangle array. The LRU+errorPQ%10 strategy employs an error priority queue size equal to 10 percent of the total cache size. As expected, the rendering improvement is significant. The build-IVA time for these modes increases the workload of the CGM. However, the global speedup obtained for the three strategies easily compensates the extra CGM cost. In fact, the actual rendering cost, which is the only cost affected by the CGM, is improved by a factor of up to 3 (including the IVA build time) as shown in Table 3.

Despite three different camera trajectories, all show almost the same behavior. More information may be obtained taking into account the number of vertices inserted in video memory. It allows to undestand which strategy makes better use of the cached geometry since more inserted vertices involves more data transfer from main to video memory. This information is given in Table 2.

CGM TYPE	Trajectory A	Trajectory B	Trajectory C
First Available	322353	329214	102097
LRU	228694	63465	64688
LRU+PQ10%	200126	63465	64690

Table 2: Vertices inserted into video memory for the three CGM modes over 1000 frames (Happy model).

The First Available strategy clearly makes the worst use of cached geometry, especially for small camera rotations. In contrary to our initial expectations, in most cases the simple LRU strategy outperforms the LRU + Error-PriorityQueue strategy. The latter gives the best result only for the circular camera trajectory, and even in this case, the lower data transfer rate does not compensate for the more expensive priority queue operations. Figure 10 b) and c) show the inefficiency of the First Available strategy for the last two camera trajectories, where the insertion of new vertices is unnecessary after a certain number of frames. Recall the most expensive frame is always the first because the cache stores no vertices at that time.

Table 3 lists the rendering speed-ups achieved by the different CGM strategies with respect to the original immediate mode vertex submission FastMesh version. The first column corresponds to the variant with a standard main-memory IVA but no CGM. The last three columns indicate the speedup factors for the three implemented CGM strategies. The individual speed-ups for the rendering stage reach factors up to 3 which shows the real impact of the Cached Geometry Manager on the rendering phase of a VDPM framework.

Model	camera	CGM render modes			
	trajectory	std IVA	First Available	LRU	LRU + PQ10%
happy	A	1.07	2.14	2.97	2.92
	B	1.07	2.21	2.89	2.88
	C	1.07	2.13	2.79	2.78
dragon	A	1.06	1.97	2.69	2.66
	B	1.05	2.12	2.83	2.8
	C	1.05	2.12	2.76	2.76
hand	A	1.07	2.25	3.07	3.06
	B	1.05	2.29	2.92	2.91
	C	1.05	2.31	2.95	2.95

Table 3: FastMesh speed-ups (just rendering stage) for different CGM strategies.

QuadTIN Results

The height-field model used for the QuadTIN rendering experiment is the well known Puget Sound data set (2563548 vertices, after QuadTIN-preprocess error tolerance = 6 meters, [PAL02]). The results were averaged over 3000 frames in a window of 1024 x 768 pixels using an error tolerance equal to one pixel (projective tolerance of geometric error projected on screen).

The chosen CGM size was $2^{16} = 65536$ following the same criteria applied as for the experiments with FastMesh. The size of each vertex element in this case is 32 bytes: 3 floats (position) + 3 floats (normal) + 2 floats (texture coordinate). The camera trajectories tested to perform the CGM analysis with the QuadTIN rendering system are the following (see Figure 11):

- Circular camera trajectory.
- Camera rotation with fixed eye.
- Straight line camera trajectory.

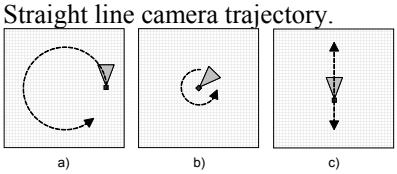
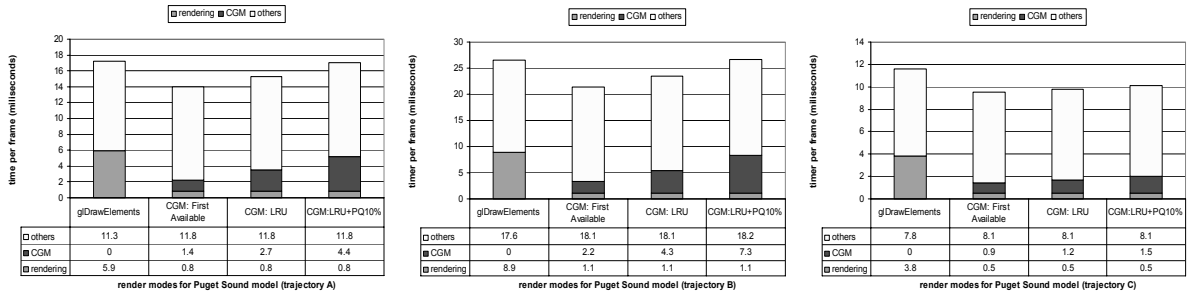


Figure 9: Terrain rendering camera trajectories: a) circular trajectory. b) camera rotation with fixed eye. c) straight line trajectory.

The CGM strategies applied to QuadTIN are the ones described in detail in Section 3.2. The QuadTIN system constructs an indexed triangle strip as required for a standard IVA approach. Note that due to the implicit definition of the active front, no information about incrementally added or removed vertices between consecutive frames is provided. QuadTIN only reports which vertices are selected for a particular frame and LOD. Note again that the CGM extension only affects the rendering time but not the LOD-selection and meshing parts of the VDPM framework, and hence we focus on the achievable rendering speed-up which is the sole target of the CGM.

Figure 12 shows four columns, one for each rendering strategy, for each camera trajectory: the first column for the standard IVA mode, and the three others for the different CGM strategies. In this case, the best result is achieved by the First Available (FA) strategy. Despite the fact that the FA strategy still makes the worst use of the geometry cache (see Table 4), its simple and fast data structures are still advantageous over the doubly-linked list of slots in the two different LRU CGM strategies. This result is not completely surprising as the minor data transfer overhead of FA is amortized by the simple and fast array data structure for slots.

The straight line camera trajectory deserves a special discussion (see Figure 13c)) since the FA strategy remains the fastest despite its bad reuse of cached geometry. The LRU and LRU+PQ10% strategies require more computation time but much less data transfer. Depending on the CPU speed in relation to the AGP bus bandwidth this result may slightly change, and the LRU strategies may win over the FA strategy for certain configurations. The relation between CPU speed and AGP bus bandwith of the system will decide which is the best strategy.



	glDrawElements	CGM: First Available	CGM: LRU	CGM:LRU+PQ10%
others	11.3	11.8	11.8	11.8
CGM	0	1.4	2.7	4.4
rendering	5.9	0.8	0.8	0.8

	glDrawElements	CGM: First Available	CGM: LRU	CGM:LRU+PQ10%
others	17.6	18.1	18.1	18.2
CGM	0	2.2	4.3	7.3
rendering	8.9	1.1	1.1	1.1

	glDrawElements	CGM: First Available	CGM: LRU	CGM:LRU+PQ10%
others	7.8	8.1	8.1	8.1
CGM	0	0.9	1.2	1.5
rendering	3.8	0.5	0.5	0.5

Figure 10: Per frame timing results (in miliseconds) for the Puget Sound data set for: a) circular camera trajectory, b) camera rotation with fixed eye and c) straight line camera trajectory.

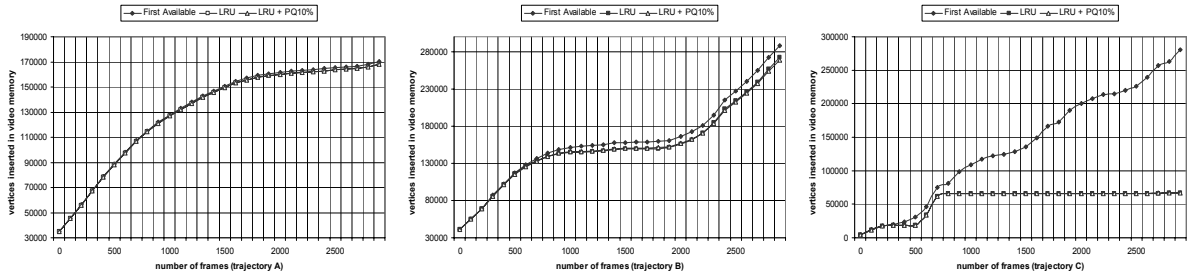


Figure 11: Vertices inserted into video memory for the Puget Sound data set for: a) circular camera trajectory, b) camera rotation with fixed eye and c) straight line camera trajectory.

CGM TYPE	Trajectory A	Trajectory B	Trajectory C
First Available	171873	302288	290747
LRU	169928	286331	66453
LRU+PQ10%	169972	281707	66436

Table 4: Vertices inserted in video memory for the CGM modes over 3000 frames (Puget Sound model).

The QuadTIN rendering speed-up factors are shown in Table 5. The speedup of the rendering stage itself, which is the only stage affected by the CGM, reaches factors up to 2.7. This dramatically shows the potential of using a CGM in a view-dependent LOD rendering system.

camera	CGM render modes		
trajectory	First Available	LRU	LRU + PQ10%
A	2.68	1.69	1.13
B	2.7	1.65	1.06
C	2.71	2.24	1.9

Table 5: QuadTIN speedups (just rendering stage) for different CGM modes.

6. CONCLUSION

This paper presents several strategies to implement an efficient Cached Geometry Manager that takes advantage of on-board video card memory for caching vertex data. It provides effective solutions to manage the video memory as a geometry cache in order to dramatically reduce the vertex data transfer rate from main to video memory for each rendered frame. The proposed techniques can be applied to a wide range of view-dependent LOD rendering frameworks, and allow the efficient reuse of cached geometry information stored on the video graphics card.

The presented approaches significantly improve the rendering performance of view-dependent LOD rendering applications with little extra implementation effort. Experimental results on two different VDPM frameworks have confirmed the suitability and the effectiveness of our approach to dramatically accelerate the rendering stage. Overall performance speed-up of observable frame rates heavily depends on the application-side view-dependent LOD-selection and meshing frameword. More recent and improved VDPM frameworks – compared to the tested FastMesh and QuadTIN systems – with significantly lower LOD-selection and meshing cost will exhibit significanly higher overall frame rate performance if combined with a dynamic CGM as demonstrated in this paper.

7. ACKNOWLEDGMENTS

This research was supported by the Spanish research grant TIC 2002-750 and the New Del Amo award UCDM-33657.

8. REFERENCES

[BAV98] BALMELLI L., AYER S., VETTERLI M.: Efficient algorithms for embedded rendering of terrain models. IEEE Inter. Conference on Image Processing ICIP 98 (1998), pp. 914-918.

[CGG03a] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: BDAM - Batched Dynamic Adaptive Meshes for High Performance Terrain Visualization. EG/IEEE TCVG Symp. on Visualization 2003.

[CGG03b] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: Planet-Sized Batched Dynamic Adaptive Meshes (P-BDAM). IEEE Visualization 2003, pp. 147–154.

[CLD03] COHEN J., LUBKE D., DUCA N., SCHUBERT B.: GLOD: Level of Detail for the Masses (2003). URL http://www.cs.jhu.edu/~graphics/ TR/TR03-4.pdf.

[DMP96] DE FLORIANI L., MARZANO P., PUPPP E.: Multiresolution models for topographic surface description. The Visual Computer (Aug. 96), pp. 317-345.

[DMP97] DE FLORIANI L., MAGILLO P., PUPPO E.: Building and traversing a surface at variable resolution. IEEE Visualization 97 (1997), pp. 103-110.

[EKT01] EVANS W., KIRKPATRICK D., TOWNSEND G.: Right-triangulated irregular networks. Algorithmica (March 2001), pp. 264-286.

[HCH03] HALL J. D., CARR N. A., HART J. C.: Cache and Bandwidth Aware Matrix Multiplication on the GPU. Technical Report UIUCDCS-R-2003-2328. University of Illinois at Urbana-Champaign Computer Science Department. April 2003.

[Hop97] HOPPE H.: View-dependent refinement of progressive meshes. SIGGRAPH 97 (1997), pp. 189-198.

[Kil99] KILGARD M. J.: NVIDIA OpenGL Extension NV_vertex_array_range. URL: http://www.nvidia.com /dev_content/nvopenglspecs/GL_NV_vertex_array_range.txt.

[KL01] KIM J., LEE S.: Truly selective refinement of progressive meshes. Graphics Interface 2001, pp. 101-110.

[LE97] LUEBKE D., ERIKSON C.: View-dependent simplification of arbitrary polygonal environments. SIGGRAPH 97 (1997) pp. 199-208.

[Lev02] LEVENBERG J.: Fast view-dependent LOD rendering using cached memory. IEEE Visualization 2002, pp. 259–265.

[LKR96] LINDSTROM P., KOLLER D., RIBARSKY W., HODGES L. F., FAUST N., TURNER G. A.: Real-time, continuous level of detail rendering of height fields. SIGGRAPH 96 (1996), pp. 109-118.

[LPT03] LARIO R., PAJAROLA R., TIRADO F.: Hyperblock-QuadTIN: Hyper-block quadtree based triangulated irregular networks. IASTED International Conference on Visualization, Imaging and Image Processing (VIIP 2003), pp. 733-738.

[LRC03] LUEBKE D., REDDY M., COHEN J., VARSHNEY A, WATSON B., HUEBNER R.: Level of detail for 3D graphics. Morgan Kaufman. 2003.

[Mar00] MARSELAS H.: Optimizing Vertex Submission for OpenGL. Game Programming Gems, pp. 353-360. Charles River Media. 2000.

[MH02] MÖLER T., HAINES E,: Real-time rendering. 2nd edition. A K Peters. 2002.

[Paj98] PAJAROLA R.: Large scale terrain visualization using the restricted quadtree triangulation. IEEE Visualization 98 (1998), pp. 19-26 and 515.

[Paj01] PAJAROLA R.: FastMesh: Efficient View-dependent Meshing. Pacific Graphics 2001, pp. 22-30.

[PAL02] PAJAROLA R., ANTONIJUAN M, LARIO R.: QuadTIN: quadtree based triangulated irregular networks. IEEE Visualization 2002, pp. 395–402.

[Pup96] PUPPO E.: Variable resolution terrain surfaces. 8th Canadian Conference on Comput. Geometry (1996), pp. 202-210.

[SS92] SIVAN R, SAMET H.: Algorithms for constructing quadtree surface maps. 5th International Symposium on Spatial Data Handling (August 1992), pp. 361-370.

[THO02] THOMPSON C. J., HAHN S., OSKIN M.: Using modern graphics architectures for general-purpose computing: a framework and analysis. 35th annual ACM/IEEE international symposium on Microarchitecture (2002), pp. 306-317.

[XV96] XIA J. C., VARSHNEY A.: Dynamic view-dependent simplification for polygonal models. IEEE Visualization 96 (1996), pp. 327-3.