Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids

Frank Losasso
Stanford University & Microsoft Research

Hugues Hoppe
Microsoft Research

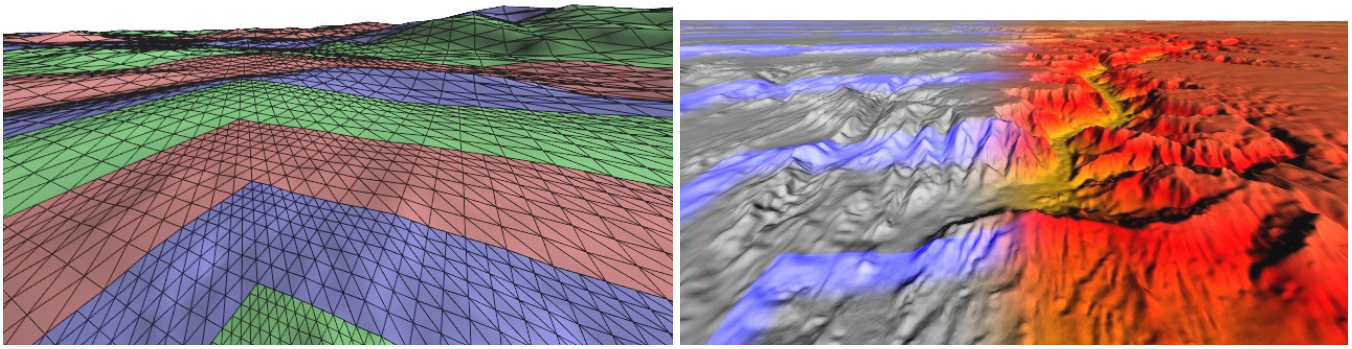


Illustration using a coarse geometry clipmap (size n=31)

View of the 216,000×93,600 U.S. dataset near Grand Canyon (n=255)

Figure 1:Terrains rendered using geometry clipmaps, showing clipmap levels (size $n \times n$) and transition regions (in blue on right).

Abstract

Rendering throughput has reached a level that enables a novel approach to level-of-detail (LOD) control in terrain rendering. We introduce the geometry clipmap, which caches the terrain in a set of nested regular grids centered about the viewer. The grids are stored as vertex buffers in fast video memory, and are incrementally refilled as the viewpoint moves. This simple framework provides visual continuity, uniform frame rate, complexity throttling, and graceful degradation. Moreover it allows two new exciting real-time functionalities: decompression and synthesis. Our main dataset is a 40GB height map of the United States. A compressed image pyramid reduces the size by a remarkable factor of 100, so that it fits entirely in memory. This compressed data also contributes normal maps for shading. As the viewer approaches the surface, we synthesize grid levels finer than the stored terrain using fractal noise displacement. Decompression, synthesis, and normal-map computations are incremental, thereby allowing interactive flight at 60 frames/sec.

Keywords: level-of-detail control, terrain compression and synthesis.

1. Introduction

Terrain geometry is an important component of outdoor graphics environments, as used for instance in movies, virtual environments, cartography, and games. In particular, real-time outdoor games include flight simulators, driving simulators, and massively multiplayer games. In this paper, we focus on the real-time rendering of terrain height-fields.

Large terrain height maps may contain billions of samples, still far too many to render interactively by brute force. Moreover, rendering a uniformly dense triangulation can lead to aliasing artifacts, caused by an unfiltered many-to-one map from samples to pixels, just as in texturing without mipmaps [Williams 1983]. Thus level-of-detail (LOD) control is necessary to adjust the terrain tessellation as a function of the view parameters.

There has been extensive research on terrain LOD, as reviewed in Section 2. Previous schemes adapt refinement based not only on the view but also on the local terrain geometry. Intuitively, planar regions are assigned larger triangles, resulting in irregular meshes with fewer triangles to render. However, these frameworks have several shortcomings. The refinement criteria and/or refinement operations must be pre-computed and consume additional memory. The data structures involve random-access traversals with poor cache-coherence. Changing the tessellation requires slower immediate-mode rendering, while caching static regions hinders temporal continuity. To maintain a constant frame rate, refinement thresholds must change with the bumpiness of the viewed terrain. Finally, surface shading requires texture images, which are stored separately and use an entirely different LOD structure.

Rendering throughput in present GPUs has surpassed 100MΔ/sec, enough to fully cover a framebuffer with *pixel-sized* triangles at video rates. Moreover, vertex processing rates should continue to increase as they catch up with pixel rates. Thus, our premise is that fine LOD adaptivity is no longer essential, and we instead seek a screen-uniform tessellation of the terrain where all triangles are nearly pixel-sized. The key is to develop an LOD framework that can optimally feed the graphics pipeline.

Our contribution is the *geometry clipmap*, which caches the terrain in a set of nested regular grids centered about the viewer (Figure 1). These grids represent filtered versions of the terrain at power-of-two resolutions, and are stored as vertex buffers in video memory. As the viewpoint moves, the clipmap levels shift and are incrementally refilled with data.

The approach has parallels with the LOD treatment of images in texture mapping. To prevent spatial aliasing, an image is prefiltered into a *mipmap* pyramid of power-of-two grids [Williams 1983]. The mipmap level rendered at a pixel is a function of screen-space parametric derivatives, which depend on view parameters, and *not* on the content of the image. A *texture clipmap* caches a view-dependent subset of the mipmap pyramid [Tanner et al 1998]. Fast incremental update of a texture clipmap allows exploration of huge images.

While geometry clipmaps are inspired by texture clipmaps, there are some key differences. Texture clipmaps compute LOD *per-pixel* based on screen-space projected geometry. With terrains however, the screen-space geometry does not exist until the terrain LOD is selected — a circular dependency. More impor-

tantly, per-pixel LOD selection would make it difficult to keep the mesh watertight and temporally smooth.

Instead, we select LOD in world space based on viewer distance, using a set of nested rectangular regions about the viewpoint. We create transition regions to smoothly blend between levels, and avoid T-junctions by stitching the level boundaries using zero-area triangles. The LOD transition scheme allows independent translation of the clipmap levels, and lets levels be cropped rather than invalidated atomically as in [Tanner et al 1998]. Also, we apply the same scheme to texture images, obtaining a unified LOD framework for geometry and images. And unlike texture clipmaps, it does not require special hardware.

Geometry clipmaps provide a number of advantages over previous terrain LOD schemes:

- **Simplicity**. There is no irregular traversal of pointer/index-based structures, and no tracking of refinement dependencies.
- **Optimal rendering throughput**. The clipmap vertices reside in video memory, and their grid structure allows indexed triangle-strip rendering with optimal vertex-cache reuse.
- **Visual continuity**. Inter-level transition regions provide spatial and temporal continuity for both geometry and texture, using a few instructions in the vertex and pixel programs respectively.
- **Steady rendering**. The rendering rate is nearly constant since the tessellation complexity is independent of local terrain roughness. There are no parameters to dynamically adjust.
- **Immediate complexity throttling**. Even with a fixed clipmap size, we can shrink the rendered regions to reduce rendering load. Tanner et al [1998] use a similar idea to regulate the update bandwidth for texture clipmaps.
- **Graceful degradation**. When the viewer is moving quickly, the update bandwidth (to refill the clipmap) can become the bottleneck. As in texture clipmaps, we update as many levels as possible within a prescribed budget. The effect is that fast-moving terrain loses its high-frequency detail.
- **Surface shading**. Normal maps are computed on-the-fly from the geometry, and use the same LOD structure as the geometry.

Geometry clipmaps also enable two new runtime functionalities:

- **Compression**. Since only the clipmap needs to be expanded into vertex data, the remainder of the terrain pyramid can be stored in compressed form. We compress residuals between pyramid levels using a 2D image coder. The high data coherence allows for compression factors around 60-100. Storing terrains entirely in memory avoids disk paging hiccups.
- **Synthesis**. The simple grid structure permits on-the-fly terrain synthesis, so that coarsely specified geometry can be amplified by procedurally generated detail. We demonstrate simple fractal noise, which will soon be possible on the GPU itself.

Limitations. The rendered mesh is more complex than in prior LOD schemes. Essentially, we are always assuming a worst-case terrain, which has uniform detail (everywhere and at all frequencies) and thus does not benefit from local adaptivity. On the other hand, the mesh is regular and resides in video memory, and thus delivers optimal rendering performance for this worst case.

Another limitation is that the terrain is assumed to have bounded spectral density as discussed in Section 9. For example, a tall needle-like feature would morph into view a little too late. Fortunately, practical terrains are well behaved and do not present such problems. Note that buildings, vegetation, and other objects that populate the environment are rendered separately using other LOD techniques.

2. Previous terrain LOD techniques

Terrain LOD algorithms use a hierarchy of mesh refinement operations to adapt the surface tessellation. Algorithms can be categorized by the structure of these hierarchies.

- **Irregular meshes** (a.k.a. triangulated irregular networks) provide the best approximation for a given number of faces, but require the tracking of mesh adjacencies and refinement dependencies. Some hierarchies use Delaunay triangulations [e.g. Cohen-Or and Levanoni 1996; Cignoni et al 1997; Rabinovich and Gotsman 1997] while others allow arbitrary connectivities [e.g. De Floriani et al 1997; Hoppe 1998; El-Sana and Varshney 1999].
- **Bin-tree hierarchies** (a.k.a. longest-edge bisection, restricted quadtree, hierarchies of right triangles) use the recursive bisection of right triangles to greatly simplify memory layout and traversal algorithms. However, these semi-regular meshes still involve random-access memory references and immediate-mode rendering [e.g. Lindstrom et al 1996; Duchaineau 1997; Pajarola 1998; Röttger et al 1998; Blow 2000; Lindstrom and Pascucci 2002].
- **Bin-tree regions** define coarser-grain refinement operations on regions associated with a bin-tree structure. Precomputed triangulated regions are uploaded to buffers cached in video memory, thereby boosting rendering throughput. One drawback is that the caching hinders use of geomorphs for temporal coherence [e.g. Levenberg 2002; Cignoni et al 2003a,2003b].
- **Tiled blocks** partition the terrain into square patches that are tessellated at different resolutions. The main challenge is to stitch the block boundaries seamlessly [e.g. Hitchner and McGreevy 1993; Bishop et al 1998; Wagner 2004]. Rather than defining a world-space quadtree, geometry clipmaps define a hierarchy centered about the viewer, and this greatly simplifies inter-level continuity in both space and time.

Ideally, view-dependent LOD algorithms adaptively refine and coarsen the mesh based on *screen-space geometric error*, the deviation in pixels between the mesh and the original terrain. Screen-space error combines the effects of (1) viewer distance, (2) surface orientation, and (3) surface geometry. Since surface orientation seldom provides significant LOD gain, many schemes choose to ignore it. One common refinement criterion [Blow 2000] stores at each vertex a radius defining an enclosing sphere. The pre-computed radius encodes the local surface approximation error, such that the neighborhood of the vertex is refined if and only if the viewpoint enters the sphere.

Geometry clipmaps are quite different from these prior works. The refinement hierarchy is based on viewer-centric grids, with geomorphs providing inter-level continuity. The refinement criterion still considers viewer distance, but it ignores local surface geometry, i.e. all vertices share the same "sphere radius".

View-dependent displacement mapping. A terrain can be thought of as a displacement map over trivial planar geometry. Some recent papers have proposed hardware schemes for adaptive tessellation of displacement maps [Gumhold and Hüttner 1999; Doggett and Hirche 2000; Moule and McCool 2002]. So far these schemes have only been simulated on relatively simple grids, and they assume that the entire grid is memory-resident.

Textures. There has been less work on handling the huge texture maps that often accompany terrains. Aside from texture clipmaps [Tanner et al 1998], the standard approach is texture tiling. More general texture hierarchies are introduced by Döllner et al [2000].

To our knowledge, none of the prior terrain LOD techniques are able to achieve significant compression or terrain synthesis.

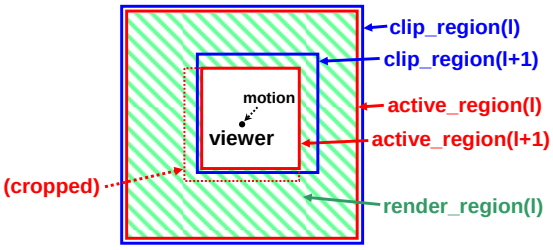


Figure 2: Regions defined within levels of a geometry clipmap.

3. Geometry clipmap overview

The geometry clipmap caches a terrain pyramid using a set of m levels, representing nested extents at successive power-of-two resolutions (Figure 1). Each level contains an $n \times n$ array of vertices, stored as a vertex buffer in video memory. To permit efficient incremental updates, the array is accessed toroidally, i.e. with 2D wraparound addressing using mod operations on x and y. Each vertex contains (x, y, z, z_c) coordinates, where z_c is the height value at (x, y) in the next-coarser level, and is used for transition morphing (Section 6.2).

Clipmap regions. For each level l of the clipmap, we define a set of rectangular regions (see Figure 2). The *clip region* is the world extent of the $n \times n$ grid of data stored at that level. The *active region* is the extent we wish to render, specifically a square of size $n \times n$ centered at the viewer. During viewer motion, we update the clipmap by shifting the clip region of each level such that it matches the desired active region. However, if this update is too costly during fast motion, we let the clip region fall behind the viewer, and then crop the active region to the available data as illustrated in Figure 2. Finally, the *render region* itself is the hollowed frame (shaded in green) whose outer perimeter is active_region(l) and whose inner perimeter is active_region(l+1).

For the finest level m, active_region(m+1) is always defined to be empty. The active and clip regions are updated as the view parameters changed, as described in Sections 4 and 5 respectively.

Texture. Each clipmap level also contains associated texture image(s). We store an 8-bit-per-channel normal map for surface shading, since this is more efficient than storing per-vertex normals. For faithful shading, the normal map has twice the resolution of the geometry, since one normal per vertex is too blurry [Vlachos et al 2001]. The normal map is computed from the geometry whenever the clipmap is updated. Additional images, such as color fields or terrain attributes could also be stored, possibly at different resolutions. Like the vertex arrays, textures are accessed toroidally for efficient update.

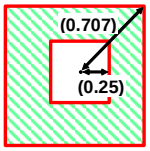
Per-frame algorithm. The following steps are performed each frame, and discussed in the following sections:

- Determine the desired active regions (§4).
- Update the geometry clipmap (§5).
- Crop the active regions to the clip regions, and render (§6).

4. Computation of desired active regions

View-dependent refinement is determined by the selection of active regions for each level of the clipmap. We use a simple strategy. For each level l, with grid spacing $g_l = 2^{-l}$ in world space, we let the desired active region be the square of size $ng_l \times ng_l$ centered at the (x,y) location of the viewpoint. In other words, the desired clipmap is re-centered at the viewer, and we hope to render the full extent of each level.

Let us consider how screen-space triangle size varies with the choice of clipmap size n. For now we assume that the terrain has small slope, so that each triangle is approximately a right triangle of size g_l. (We provide a more general error analysis in Section 9.)



For any visible world point, screen-space size is inversely proportional to depth in screen space. If the view direction is horizontal, screen-space depth is measured in the XY plane. The viewer lies at the center of render_region(l), which has outer edge size ng_l and an inner edge size $ng_l/2$. For a field-of-view $\varphi = 90°$, the average screen-space depth (over all view directions) is about $(0.4)ng_l$. Thus, the approximate screen-space triangle size s in pixels is given by

$$s = \frac{g_l}{(0.4)ng_l}\frac{W}{2\tan\frac{\varphi}{2}} = (1.25)\frac{W}{n\tan\frac{\varphi}{2}},$$

where W is the window size and φ is the field of view. For our default W=640 pixels and $\varphi = 90°$, we obtain good results with a clipmap size n=255. This corresponds to a screen-space triangle size s of 3 pixels. Since the normal maps are stored at twice the resolution, this gives approximately 1.5 pixels per texture sample, which is a reasonable setting for texture sampling.

When the view direction is not horizontal, the screen-space depth of render_region(l) is larger than the $(0.4)ng_l$ derived above, and therefore the screen-space triangle size becomes smaller than s. If the view looks straight down from high above the terrain, triangle size is tiny and aliasing becomes evident. The solution is to disable the rendering of unnecessarily fine levels. Specifically, we compute the height of the viewer over the terrain by accessing the finest valid level of the clipmap. For each level l, the active region is set to be empty if viewer height is greater than $(0.4)ng_l$.

One drawback of the simple viewer-centered regions is that the clipmap size n must grow as the field of view φ narrows. A remedy would be to adapt the location and size of clipmap regions to the view frustum. We instead chose viewer-centered regions because they let the view instantly rotate about the current viewpoint. This is a requirement for many applications such as flight simulators that let the user look in all directions using a joystick "hat switch". We rely on view frustum culling to avoid rendering terrain that is outside the viewport (Section 6.4).

5. Geometry clipmap update

As the desired active regions shift with the viewer's motion, the clip regions should also shift accordingly. Note that with toroidal access, we do not need to copy the old data when shifting a level. Instead, we simply fill the newly exposed "L-shaped" region. The data comes from one of two sources: decompression of explicit terrain or synthesis of procedural terrain (see Sections 7 and 8). Usually, coarser levels are filled from decompressed terrain, and finer levels are synthesized.

When updating the clipmap using either compression or synthesis, we predict the finer level geometry from the coarser one using an interpolatory subdivision scheme. We have chosen the tensor-product version [Kobbelt 1996] of the well-known four-point subdivision curve interpolant, which has mask weights (–1/16, 9/16, 9/16, –1/16) [Dyn et al 1987]. This upsampling filter U has the desirable property of being C^1 smooth.

An alternative update scheme would be to anticipate future viewer motion when translating the clip regions, to reduce the frequency of updates. Because we are able to perform both decompression and synthesis efficiently on small regions, the granularity of updates is not currently an important factor.

When the viewer is moving fast, the processing needed to update all levels can become excessive. As in texture clipmaps, we update levels in coarse-to-fine order, stopping upon reaching a

given processing budget. We have chosen to stop if the total number of updated samples exceeds n^2. Because the clip regions in the non-updated (finer) levels fall behind, they gradually crop the associated active regions, until these become empty. The effect is that the fast-moving (near-viewer) terrain loses its high-frequency detail. An interesting consequence is that rendering load actually decreases as the viewer moves faster.

We enforce the following constraints on the clipmap regions:

(1) clip_region(l+1) $\subseteq$ clip_region(l) $\ominus$ 1, where $\ominus$ denotes erosion by a scalar distance. We need the clip regions to be nested for coarse-to-fine geometry prediction. The prediction requires maintaining one grid unit on all sides.

(2) active_region(l) $\subseteq$ clip_region(l), since the rendered data must be a subset of the data present in the clipmap.

(3) the perimeter of active_region(l) must lie on "even" vertices, to enable a watertight boundary with coarser level l–1.

(4) active_region(l+1) $\subseteq$ active_region(l) $\ominus$ 2, since the render region must be at least two grid units wide to allow a continuous transition between levels.

6. Geometry clipmap rendering

6.1 Basic rendering algorithm

Given the desired active regions, we render the terrain using the following algorithm:

```
// Crop the active regions.
foreach level l∈[1,m] in coarse-to-fine order:
   Crop active_region(l) to clip_region(l)
   Crop active_region(l) to active_region(l-1) ⊖ 2
// Render all levels
foreach level l∈[1,m] in fine-to-coarse order:
   render_region(l) := active_region(l) – active_region(l+1)
   Render render_region(l)
```

The active regions are cropped to the clip regions and coarser active regions to satisfy constraints (2–4) from Section 5. Note that if active_region(k) is empty, then by construction all finer active_region(l), $l > k$ are also empty. It is quite common for the finer levels to have empty active regions, either because their clip regions have not been updated in time (i.e. the viewer is moving fast), or because finer tessellations are unwarranted (i.e. the viewer is sufficiently high above the terrain).

Since finer levels are closer to the viewer, we render levels in fine-to-coarse order to exploit any hardware occlusion culling. The render_region(l) is partitioned into 4 rectangular regions which are rendered using triangle strips as illustrated in Figure 3. The maximum strip length is selected for optimal vertex caching [Hoppe 1999], and strips are grouped together to form large batched primitives. The grid-sequential memory access behaves well at all levels of the video memory hierarchy. Currently, the 2D toroidal access requires CPU recomputation of vertex indices every frame, but this is a small overhead and will go away shortly.

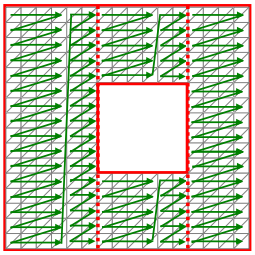


Figure 3: Illustration of triangle strip generation within a render region. (In reality, the strips are up to ~20 triangles in length.)

6.2 Transition regions for visual continuity

The simple algorithm described so far suffers from gaps between the render regions at different levels, due to the power-of-two mismatch at the boundaries. To both eliminate the gaps and provide temporal continuity, we morph the geometry near the outer boundary of each render_region(l) such that it transitions to the geometry of the coarser level l-1. The morph is a function of the spatial (x,y) grid coordinates of the terrain vertices relative to those of the viewpoint (v_x,v_y). Thus the transition is not time-based but instead tracks the continuous viewer position.

Through experimentation, we have found that a transition width w of $n/10$ grid units works well. If w is much smaller, the level boundaries become apparent. And if w is much larger, fine detail is lost unnecessarily. If the finer active_region(l+1) is too close, we compute $w = \min(n/10, \text{min\_width}(l))$, where min_width($l$) is known to be at least 2 (see Figure 4).

Recall that each vertex stores (x, y, z, z_c), where z_c is the terrain height in the next-coarser level l-1. We obtain the morphed elevation as

$$z' = (1-\alpha)z + \alpha z_c$$

where blend parameter α is computed as $\alpha = \max(\alpha_x, \alpha_y)$ with

$$\alpha_x = \min\left(\max\left(\left(\left|x - v_x^l\right| - \left(\frac{x_{\max} - x_{\min}}{2} - w - 1\right)\right)\Big/ w, 0\right), 1\right)$$

and similarly for α_y. Here (v_x^l, v_y^l) denote the continuous coordinates of the viewpoint in the grid of clip_region(l), and x_{min} and x_{max} the integer extents of active_region(l). The desired property is that α evaluates to 0 except in the transition region where it ramps up linearly to reach 1 at the outer perimeter. These evaluations are performed in the GPU vertex shader using about 10 instructions, so they add little to the rendering cost.

T-junction removal. Although the geometry transitions eliminate gaps, the T-junctions along the boundaries still result in dropped pixels during rasterization. To stitch adjacent levels into a watertight mesh, we use the simple solution of rendering zero-area triangles along the render region boundaries.

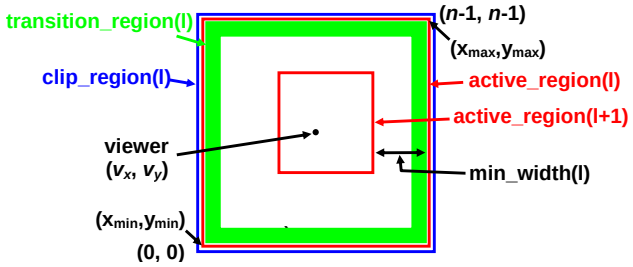


Figure 4: A transition region near the outer boundary lets level l smoothly blend with the next coarser level l-1.

6.3 Texture mapping

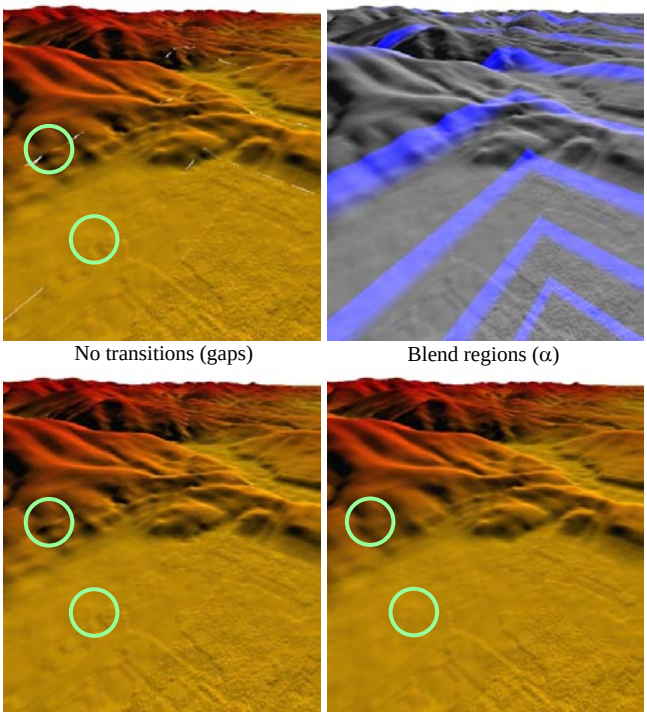
Recall that each clipmap level also stores texture images for use in rasterization (e.g. a normal map in our system).

One option would be to let hardware mipmapping control texture LOD. The texture at each clipmap level would have its own mipmap pyramid, thus requiring 33% more memory. Note that the coarser mipmap levels in this pyramid correspond exactly with sub-regions in coarser clipmap levels, so ideally they should be shared as in texture clipmaps [Tanner et al 1998], but we lack this hardware capability. More significantly, there is a practical problem with letting the hardware control the mipmap level. If

the resolution of the stored texture is not sufficiently high, a sharp transition in texture resolution becomes evident at the render region boundaries, because the mipmap level has not reached the next coarser level. These sharp transitions are visible during viewer motion as "advancing fronts" over the terrain surface.

Instead, we propose an alternate solution. We disable mipmapping altogether, and perform LOD on the texture using the same spatial transition regions applied to the geometry. Thus texture LOD is based on viewer distance rather than on screen-space derivatives as in hardware mipmapping. The main element lost in this approximation is the dependence of texture LOD on surface orientation. When a surface is oriented obliquely, one can no longer access a coarser mipmap level to prevent aliasing. However, graphics hardware commonly supports anisotropic filtering, whereby more samples are combined from the original finest level. Consequently, the absence of mipmaps is not a practical issue for surface orientation dependence.

The spatially based texture LOD scheme is easily implemented in the GPU pixel shader. When rendering level l, we provide the shader with the textures from levels l and l-1, and blend these using the same α parameter already computed in the vertex shader for geometry transitions. Figure 5 shows an example.



No transitions (gaps) | Blend regions (α)

Geometry transitions | Geometry + texture transitions

Figure 5: Visual continuity achieved with transition morphs (demonstrated with a low-resolution clipmap of size n=127).

6.4 View-frustum culling

We apply view frustum culling as follows. For each level of the clipmap, we maintain z_{min}, z_{max} bounds for the local terrain. Recall that each render region is partitioned into 4 rectangular regions. Each 2D rectangular extent is extruded by the terrain bounds [z_{min} , z_{max}] to form an axis-aligned bounding box. We intersect this box with the 4-sided pyramid of the view frustum, and project the resulting convex set into the XY plane. The axis-aligned rectangle bounding this set is used to crop the given rectangular region (Figure 6). View frustum culling reduces rendering load by a factor of about 3 for a 90° field of view.

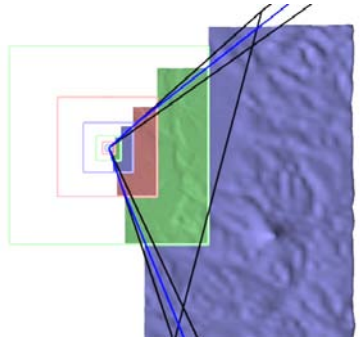


Figure 6: Result of view frustum culling (viewed from above).

7. Terrain compression

Height maps are remarkably coherent in practice, significantly more so than typical color images, and thus offer a huge opportunity for compression. To interact efficiently with the geometry clipmap structure, the decompression algorithm must support "region-of-interest" (ROI) queries at any power-of-two resolution.

We have adopted a simple pyramid compression scheme. We first create a terrain pyramid $T_1..T_m$ by successively downsampling the fine terrain T_m into coarser levels using a linear filter $T_{l-1} = D(T_l)$. Then, each pyramid level T_l is predicted from its next coarser level T_{l-1} through interpolatory subdivision $U(T_{l-1})$ (Section 5), and the residual $R_l = T_l - U(T_{l-1})$ is compressed using an image coder.[1] Since the compression is lossy, R_l is approximated by $\tilde{R}_l$. Therefore, we reconstruct the levels in coarse-to-fine order as $\tilde{T}_l = U(\tilde{T}_{l-1}) + \tilde{R}_l$, and compress the residuals redefined as $R_l = T_l - U(\tilde{T}_{l-1})$, so that the errors do not accumulate.

Since coarser levels are viewed from afar, our first approach was to give their approximations $\tilde{T}_l$ less absolute accuracy. Specifically, we would scale the residuals R_l by 2^{l-m} prior to quantization. However, while this is a correct argument for geometric fidelity, we discovered that this results in poor visual fidelity, because both the normal map and z-based coloring then present quantization artifacts (since they are inferred from the decompressed geometry). The solution is to compress all level residuals with the same absolute accuracy.

The quantized residuals are compressed using the PTC image coder of Malvar [2000], which has several nice properties for our purpose. It avoids blocking artifacts by defining overlapping basis functions, yet the bases are spatially localized to permit efficient regional decompression. Also, the coder supports images of arbitrary size (if the encoding fits within 2 GB). Decompression takes place only during the incremental uploads to video memory, and is thus sufficiently fast (Table 1).

We are able to implement the compression preprocess within a 32-bit address space by performing all steps as streaming computations. For the 40GB U.S. data, the complete procedure from original terrain T_m to compressed residuals $\tilde{R}_l$ takes about 5 hours, much of which is disk I/O. Section 9 reports the rms of the compression error $\tilde{T}_m - T_m$. In our experience, the compressed terrain is visually indistinguishable from the original, except at the sharp color transition associated with the coastline.

As future work, it would be interesting to compare with a compression scheme like Normal Meshes [Guskov et al 2000] in which the downsampling filter D is an impulse function.

[1] We precompute the optimal filter D (of size 11×11) such that $U(D(T_l))$ gives the best L^2 approximation of T_l, by solving a linear system on a subset of the given terrain.

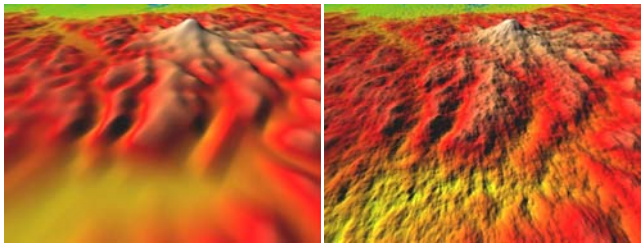
8. Terrain synthesis

The geometry clipmap provides a natural structure to generate detail using either stochastic subdivision [e.g. Fournier et al 1982; Lewis 1987] or multiresolution texture synthesis [e.g. Wei and Levoy 2000]. One constraint is that the synthesis process must be spatially deterministic, so that the same terrain is always created.

We have implemented fractal noise displacement, by adding uncorrelated Gaussian noise to the upsampled coarser terrain. The noise variance is scaled at each level l to equal that in actual terrain, i.e. the variance of the residuals R_l computed in the previous section. The C^1 smoothness of the interpolatory subdivision is key to avoiding surface crease artifacts [Miller 1986]. For efficient evaluation, we store precomputed Gaussian noise values within a table, and index it with a modulo operation on the vertex coordinates. A table size of 50×50 is sufficient to remove any repetitive patterns or recognizable banding (see Figure 7).

We had hoped to implement the synthesis process using GPU pixel shaders, so that the geometry data would remain entirely in video memory. Although some GPUs already have the required "render-to-vertex" capability, it is unfortunately not yet exposed, so for now we resort to CPU computation. Even so, the runtime computation is very fast (Table 1).

Procedural synthesis allows the generation of terrains with infinite extent and resolution, and therefore offers tremendous potential. In our experience, simple fractal noise is less interesting visually than measured elevation data, but we are hopeful that more sophisticated synthesis techniques can lead to realistic landscapes. The challenge will be to make these techniques fast, spatially deterministic, and perhaps parallelizable on the GPU.



Coarse geometry + zero detail Coarse geometry + fractal noise

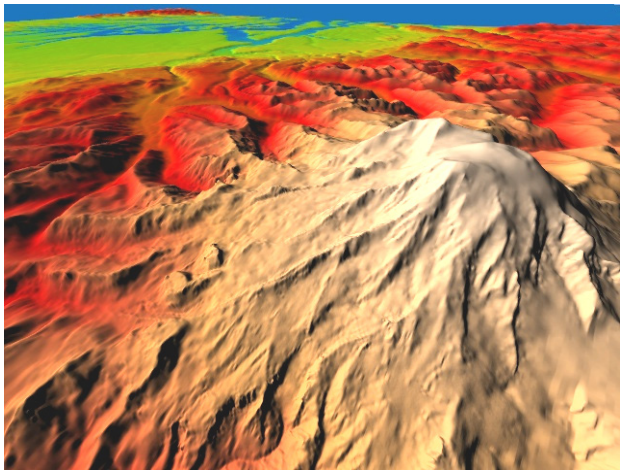
Figure 7: Example of terrain synthesis in finer levels. Of the 11 levels in the clipmap, only the coarsest 3 are stored geometry.

9. Results and discussion

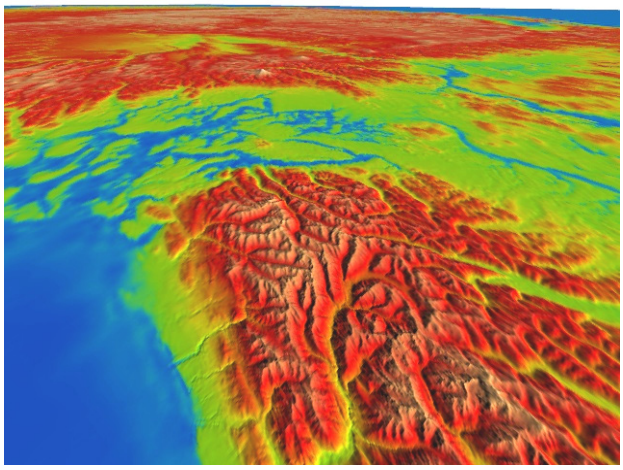
We have experimented with two USGS datasets. The smaller one is a $16{,}385^2$ grid of the Puget Sound area at 10m spacing, with 16-bit height values at 0.1m vertical resolution. The full-resolution grid is assigned to level l=9, such that it has 65^2 extent at the coarsest level l=1.

The larger dataset is a 216,000×93,600 height map of the conterminous United States at 30m spacing and 1.0m vertical resolution. (More precisely, spacing is 1 arc-sec in both longitude and latitude, with extents [126°W,66°W]x[24°N,50°N].) In a clipmap with m=11 levels, it occupies a 212×93 rectangle at the coarsest level. (We render it flat even though it is parametrized by spherical coordinates.)

Figure 8 shows these terrains rendered into a window of size 640×480 pixels, with a field-of-view of 90°. We used a PC with a 3.0 GHz Pentium4 CPU, 1 GB system memory, and an ATI Radeon 9800XT GPU with 256MB video memory. In addition to shading the terrain with a normal map, we also apply color with a simple 1D texture based on the terrain z coordinate.



Portion of 16,385×16,385 grid of Puget Sound



Portion of 216,000×93,600 grid of U.S.

Figure 8: The two datasets rendered using geometry clipmaps.

Rendering rate. For m=11 levels of size 255^2, we obtain 120 frames/sec with frustum culling, at a rendering rate of 59 MΔ/sec. (With 4x framebuffer multisampling, it drops to 95 frames/sec.) By comparison, Lindstrom and Pascucci [2002] report 3 MΔ/sec, and Cignoni et al [2003b] achieve 16 MΔ/sec. On present, comparable hardware (GeForceFX 5800/5900), these authors now obtain rates of 21 MΔ/sec and 65 MΔ/sec respectively.

Update rate. Our threshold processing budget for updating the clipmap is a full n×n level. Table 1 shows the execution times of the update steps for this worst case. It is likely that these times overlap with GPU processing. During smooth viewer motions, the update times are generally less because only fractions of levels need be updated. In practice, our system maintains a nearly uniform rate of 60 frames/sec. Note that it will soon be possible to perform all steps (except for decompression) using the GPU, thanks to the regular-grid data structure.

Update step	Time (msec)
Computation of z_c	2
Interpolatory subdivision U	3
Decompression or Synthesis	8 or 3
Upload to video memory	2
Normal map computation	11
Total	21 or 26

Table 1: Times for updating a full n×n level (n=255).

Error analysis. There are two sources of error, compression error and LOD error, and we analyze these separately.

Compression error. The Puget Sound dataset is compressed from 537 MB to 8.5 MB, with an rms error of 1.0m (PSNR=$20\log_{10}(z_{max}/\text{rms})$=72.6dB). The U.S. dataset is compressed from 40.4 GB to 355 MB, with an rms error of 1.8m (PSNR=67.7dB). These rms errors are quite small – only about 10% and 6% of the inter-sample spacing, respectively.

Screen-space LOD error. In Section 4, we estimated the screen-space triangle size s for a given clipmap size n. The analysis relied on the fact that terrain triangles have compact shape if the terrain slope is assumed small. If instead the terrain has steep slope, triangles can become arbitrarily elongated and their screen-space extent is no longer bounded, which is unsatisfactory.

However, the more relevant measure is the screen-space geometric error, i.e. the screen-projected difference between the rendered mesh and the original terrain (Section 2). And, we can analyze this error if provided knowledge of the spectral properties of the terrain geometry.

For each terrain level T_l, we are interested in the error function $e_l = PL(T_l) - PL(T_m)$ where PL denotes the piecewise linear mesh interpolant over the (x,y) domain. This function is related to the (continuous) spectral density of the terrain signal. Since the grid spacing g_l in level l projects to s pixels in screen space, the screen-space projection of error $e_l(x,y)$ at location (x,y) is at most

$$\hat{e}_l(x,y) = \frac{e_l(x,y)}{g_l} s .$$

(The error is smaller if the view direction is not horizontal.) Thus, given a terrain dataset, we compute norms of $\hat{e}_l$ to estimate the screen-space error for each rendered level, as shown in Table 2.

The results reveal that the rms screen-space error is smaller than one pixel. This is not unexpected, since the triangle size s is only 3 pixels and the difference between those planar triangles and the finer detail is generally smaller yet. We find the larger $\max(\hat{e}_l)$ error values to be misleading, because the acquired terrain data contains mosaic misregistration artifacts that create artificial cliffs, and it only takes one erroneous height value to skew the statistic. Instead, we prefer to examine the 99.9th percentile error, and we see that it too is still smaller than a pixel. (See also the accompanying video.)

In comparison, Cignoni et al [2003b] use the same window size and a tolerance of 3 pixels. Lindstrom and Pascucci [2002] also use a 640x480 window, and mention that geomorphs allow the tolerance to reach 6 pixels without noticeable visual artifacts. The authors of both these papers report that on present hardware their schemes can now maintain a screen-space tolerance of 1 pixel.

The error analysis suggests that we could afford to reduce the clipmap size while still maintaining acceptable geometric fidelity. However, the true limiting factor is *visual* fidelity, which in turn strongly depends on surface shading — this is the basic premise of normal mapping. Therefore, even if we used coarser geometry, we would still have to maintain high-resolution normal maps. In our system, these normal maps are generated from the geometry clipmap itself. Indeed, the compressed mipmap pyramid can be seen as an effective scheme for encoding the normal map, with a secondary benefit of providing carrier geometry.

The non-uniformity of screen-space error $\hat{e}_l$ across levels could be exploited by adapting the sizes of individual clipmap levels. For instance, smooth hills would require a sparser tessellation (in screen space) on the nearby smooth terrain than on the farther hill silhouettes. As just discussed, one would have to verify that the surface shading is not adversely affected. Both the Puget Sound and U.S. terrain datasets appear to have rather uniform spectral densities. In the U.S. data, the error begins to diminish at coarse levels, reflecting the fact that the Earth is smooth at coarse scale.

Level l	Puget Sound			U.S.		
	rms($\hat{e}_l$)	$P_{.999}(\hat{e}_l)$	max($\hat{e}_l$)	rms($\hat{e}_l$)	$P_{.999}(\hat{e}_l)$	max($\hat{e}_l$)
1	0.12	0.58	1.27	0.02	0.12	0.30
2	0.14	0.75	1.39	0.04	0.20	0.43
3	0.15	0.86	2.08	0.06	0.32	0.62
4	0.15	0.93	2.50	0.09	0.51	0.96
5	0.14	0.96	3.38	0.12	0.68	1.37
6	0.13	0.94	5.55	0.13	0.78	2.03
7	0.11	0.83	8.03	0.14	0.84	2.59
8	0.11	0.75	14.25	0.13	0.86	4.16
9	0.00	0.00	0.00	0.12	0.90	8.18
10				0.11	0.90	11.70
11				0.00	0.00	0.00

Table 2: Analysis of screen-space geometric error, in pixels. Columns show rms, 99.9th percentile, and maximum errors. (n=255, W=640, φ=90°, i.e. s=3).

Space requirement. For the U.S. dataset, the number of levels is m=11, and the compressed terrain occupies 355 MB in system memory. For our default clipmap size n=255, the geometry clipmap needs $16mn^2$=11 MB in video memory for the vertex geometry. (Since we cannot yet do level prediction on the GPU, we also replicate the z height data in system memory, requiring $4mn^2$=3 MB.) The normal maps have twice the resolution, but only 2 bytes/sample, so need an additional $8mn^2$=6 MB. Thus, overall memory use is about 375 MB, or only 0.02 bytes/sample. As shown in Table 3, our space requirement is significantly less than in previously reported results. Since the data fits within the memory of a standard PC, we avoid runtime disk accesses.

LOD scheme	Grid size	Num. of samples	Runtime space	Bytes/ sample
Hoppe [1998]	4K×2K	8M	50 MB	6.0
Lindstrom [2002]	16K×16K	256M	5.0 GB	19.5
Cignoni et al [2003a]	8K×8K	64M	115 MB	1.8
Cignoni et al [2003b]	6×13313^2	1G	4.5 GB	4.5
Geometry clipmaps	16K×16K	256M	25 MB	**0.10**
	216K×94K	20G	375 MB	**0.02**

Table 3: Comparison of runtime space requirements. Prior methods also require storage of a normal map for surface shading (which is not included here), whereas ours is computed on-the-fly from the decompressed geometry.

Precision. For m=11 levels, floating-point precision is not yet an issue. To allow an arbitrary number of levels, a simple solution is to transform the viewpoint and view matrix into the local coordinate system of each clipmap level (using double precision as in [Cignoni et al 2003b]).

Networked viewer. The compressed terrain pyramid residuals $\tilde{R}_l$ could be stored on a server and streamed incrementally (based on user motion) to a lightweight client. The necessary bandwidth is small since the compression factor is on the order of 60-100.

10. Summary and future work

A pre-filtered mipmap pyramid is a natural representation for terrain data. We present geometry clipmaps, which cache nested rectangular extents of this pyramid to create view-dependent approximations. A unique aspect of the framework is that LOD is independent of the data content. Therefore the terrain data does not require any precomputation of refinement criteria. Together with the simple grid structure, this allows the terrain to be created lazily on-the-fly, or stored in a highly compressed format. Neither of these capabilities has previously been available.

We demonstrate interactive flight over a 20-billion sample grid of the U.S., stored in just 355 MB of memory and incrementally decompressed at 60 frames/sec. The decompressed data has an rms error of 1.8 meters over the U.S. The view-dependent LOD has a screen-space error whose 99.9th percentile is smaller than one pixel, and the rendering is temporally smooth.

The representation of geometry using regular grids should become even more attractive as vertex and image buffers become unified. This unification will enable the highly parallel GPU rasterizer to process geometry in addition to images. An earlier solution will be to use vertex textures (e.g. as in DirectX9 Vertex Shader 3.0) to toroidally access geometry images [Gu et al 2002], thereby greatly simplifying implementation of geometry clipmaps.

Geometry clipmaps unify the LOD management of the terrain geometry and its associated texture signals. The spatially based LOD structure lets low-resolution textures be applied without visual discontinuities at level boundaries. Beyond our runtime creation of normal maps, we envision that non-local functions such as shadow maps can be similarly computed in a lazy fashion.

Geometry clipmaps present many more avenues for future work:

- Improved terrain synthesis, e.g. using machine learning.
- Geometry synthesis on the GPU, e.g. [Losasso et al 2003].
- Procedural terrain overlays.
- Runtime terrain modification.
- Terrain collision detection within the GPU.
- GPU-based decompression of geometry images.
- Extension to a spherical domain, e.g. [Cignoni et al 2003b].

Acknowledgments

We thank Rico Malvar and Erin Renshaw for the PTC image compression library, the Flight Simulator Group for obtaining the U.S. elevation data, and Peter Lindstrom for preparing the Puget Sound dataset. Thanks also to Cignoni, Gobbetti, and Lindstrom for testing their terrain LOD schemes on comparable hardware.

References

BISHOP, L., EBERLY, D., WHITTED, T., FINCH, M., AND SHANTZ, M. 1998. Designing a PC game engine. *IEEE CG&A* 18(1), 46-53.

BLOW, J. 2000. Terrain rendering at high levels of detail. *Proc. 2000 Game Developers Conference*.

CIGNONI, P., PUPPO, E., AND SCOPIGNO, R. 1997. Representation and visualization of terrain surfaces at variable resolution. *The Visual Computer* 13(5), 199-217.

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2003a. BDAM – Batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum* 22(3).

CIGNONI, P., GANOVELLI, F., GOBBETTI, E., MARTON, F., PONCHIO, F., AND SCOPIGNO, R. 2003b. Planet-sized batched dynamic adaptive meshes (P-BDAM). *IEEE Visualization 2003*.

COHEN-OR, D., AND LEVANONI, Y. 1996. Temporal continuity of levels of detail in Delaunay triangulated terrain. *IEEE Visualization*. 37-42.

DE FLORIANI, L, MAGILLO, P. AND PUPPO, E. 1997. Building and traversing a surface at variable resolution. *IEEE Visualization 1997*, 103-110.

DOGGETT, M, AND HIRCHE, J. 2000. Adaptive view-dependent tessellation of displacement maps. *Graphics Hardware Workshop*, 59-66.

DÖLLNER, J., BAUMANN, K., AND HINRICHS, K. 2000. Texturing techniques for terrain visualization. *IEEE Visualization 2000*, 227-234.

DYN, N., GREGORY, J., AND LEVIN, D. 1987. A 4-point interpolatory subdivision scheme for curve design, *CAGD* 4, 257-268.

DUCHAINEAU, M., WOLINSKY, M., SIGETI, D., MILLER, M., ALDRICH, C., AND MINEEV-WEINSTEIN, M. 1997. ROAMing terrain: Real-time optimally adapting meshes. *IEEE Visualization 1997*, 81-88.

EL-SANA, J., AND VARSHNEY, A. 1999. Generalized view-dependent simplification. *Proceedings of Eurographics 1999*, 83-94.

FOURNIER, A., FUSSELL, D., AND CARPENTER, L. 1982. Computer rendering of stochastic models. *Comm. of the ACM* 25(6), 371-384.

GU, X., GORTLER, S., AND HOPPE, H. Geometry images. *ACM SIGGRAPH 2002*, 355-361.

GUSKOV, I., VIDIMČE, K., SWELDENS, W., AND SCHRÖDER, P. Normal meshes. *SIGGRAPH 2000*, 95-102.

GUMHOLD, S., AND HÜTTNER, T. 1999. Multiresolution rendering with displacement mapping. *Graphics Hardware Workshop 1999*.

HITCHNER, L., AND MCGREEVY, M. 1993. Methods for user-based reduction of model complexity for Virtual Planetary Exploration. *Proc. SPIE 1913*, 622-636.

HOPPE, H. 1998. Smooth view-dependent level-of-detail control and its application to terrain rendering. *IEEE Visualization 1998*, 35-42.

HOPPE, H. 1999. Optimization of mesh locality for transparent vertex caching. *ACM SIGGRAPH 1999*, 269-276.

KOBBELT, L. 1996. Interpolatory subdivision on open quadrilateral nets with arbitrary topology. *Eurographics 1996*, 409-420.

LEVENBERG, J. 2002. Fast view-dependent level-of-detail rendering using cached geometry. *IEEE Visualization 2002*, 259-266.

LEWIS, J. 1987. Generalized stochastic subdivision. *ACM Transactions on Graphics* 6(3), 167-190.

LINDSTROM, P., KOLLER, D., RIBARSKY, W., HODGES, L., FAUST, N., AND TURNER, G. 1996. Real-time, continuous level of detail rendering of height fields. *ACM SIGGRAPH 1996*, 109-118.

LINDSTROM, P., AND PASCUCCI, V. 2002. Terrain simplification simplified: A general framework for view-dependent out-of-core visualization. *IEEE TVCG* 8(3), 239-254.

LOSASSO, F., HOPPE, H, SCHAEFER, S., AND WARREN, J. 2003. Smooth geometry images. *Symposium on Geometry Processing 2003*, 138-145.

MALVAR, H. 2000. Fast Progressive Image Coding without Wavelets. *Data Compression Conference (DCC '00)*, 243-252.

MILLER, G. 1986. The definition and rendering of terrain maps. *ACM SIGGRAPH 1986*, 39-48.

MOULE, K., AND MCCOOL, M. 2002. Efficient bounded adaptive tessellation of displacement maps. *Graphics Interface 2002*.

PAJAROLA, R. 1998. Large scale terrain visualization using the restricted quadtree triangulation. *IEEE Visualization 1998*, 19-26.

RABINOVICH, B., AND GOTSMAN, C. 1997. Visualization of large terrains in resource-limited computing environments. *IEEE Visualization*.

RÖTTGER, S., HEIDRICH, W., SLUSALLEK, P., AND SEIDEL, H.-P. 1998. Real-time generation of continuous levels of detail for height fields. *Central Europe Conf. on Computer Graphics and Vis.*, 315-322.

TANNER, C., MIGDAL, C., AND JONES, M. 1998. The clipmap: A virtual mipmap. *ACM SIGGRAPH 1998*, 151-158.

VLACHOS, A., PETERS, J., BOYD, C., AND MITCHELL, J. 2001. Curved PN triangles. *Symposium on Interactive 3D Graphics*, 159-166.

WEI, L, AND LEVOY, M. Fast texture synthesis using tree-structured vector quantization. *ACM SIGGRAPH 2000*, 479-488.

WAGNER, D. 2004. Terrain geomorphing in the vertex shader. In *ShaderX^2: Shader Programming Tips & Tricks with DirectX 9*. Wordware Publishing.

WILLIAMS, L. 1983. Pyramidal parametrics. *ACM SIGGRAPH*. 1-11.