Texturing Techniques for Terrain Visualization

Jürgen Döllner Konstantin Baumann Klaus Hinrichs*
Department for Computer Science, University of Münster

Abstract

We present a new rendering technique for processing multiple multiresolution textures of LOD terrain models and describe its application to interactive, animated terrain content design. The approach is based on a multiresolution model for terrain texture which cooperates with a multiresolution model for terrain geometry. For each texture layer, an image pyramid and a texture tree are constructed. Multiple texture layers can be associated with one terrain model and can be combined in different ways, e.g., by blending and masking. The rendering algorithm traverses simultaneously the geometry multiresolution model and the texture multiresolution model, and takes into account geometric and texture approximation errors. It uses multi-pass rendering and exploits multitexturing to achieve real-time performance. Applications include interactive texture lenses, texture animation, and topographic textures. These techniques offer an enormous potential for developing new visualization applications for presenting, exploring and manipulating spatio-temporal data.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation – Display algorithms; I.3.6. [Computer Graphics]: Methodology and Techniques.

Additional Keywords: Terrain Rendering, Texture Mapping, Multiresolution, Level of Detail, 3D Maps.

1 INTRODUCTION

For a long time research in terrain visualization has been concerned with developing techniques to cope with complex terrain geometries. While LOD terrain models as well as image-based modeling and rendering techniques form the basis of any kind of large-scale, real-time terrain visualization, texturing techniques for terrain visualization have not been studied as much. If textures have been considered at all, the texture management refers to one single large-scale texture such as satellite images.

Texture mapping, however, is a fundamental graphics primitive [14] with various applications and implications for terrain visualization: luminance textures, alpha textures, multitexturing, and real-time texture generation offer an enormous potential for developing new terrain visualization techniques for presenting, exploring and manipulating spatio-temporal data.

Terrain texturing has to deal with the following problems: Handling of large-scale multiresolution textures and multiple texture layers; developing a rendering algorithm using multitexturing and multiple passes; and investigating applications of texture mapping to terrain visualization. We understand *texture mapping as a fundamental tool for the visual design of terrain contents.*

We present a multiresolution model for terrain textures as an extension of a generic multiresolution model for terrain geometry. Consequently, it is possible to integrate the texturing techniques in any kind of geometry multiresolution model, for example, grid-based models or TIN-based models.

For each texture layer, the multiresolution model preprocesses the original texture and derives an image pyramid as well as a texture tree (Figure 1). The texture tree represents texture patches at different levels of detail; each texture patch is associated with a geometry patch provided by the geometry multiresolution model. The original texture may have arbitrary size and may cover the whole terrain or only parts of it. Texture layers can be combined (in screen space) by imaging operations such as addition, blending, and masking.

The rendering algorithm takes into account both geometric and texture approximation errors. User-defined thresholds for these errors control the visual quality as well as rendering performance. Both multiresolution models have to cooperate closely because the rendering algorithm traverses them simultaneously. Multi-pass rendering is required to cope with multiple textures. To improve real-time performance, the number of rendering passes can be reduced using multitexturing [29]; multitexturing is now available even on low cost graphics hardware. The algorithm has been efficiently implemented on top of OpenGL.

The texturing techniques allow for implementing interactive, animated visualization tools. Multiple texture layers representing different thematic data sets can be visually combined in the image. A texture layer can be used to modify other texture layers, e.g., their appearance or luminance. Furthermore, multiple texture layers can be used for animating spatio-temporal data. Visual tools for guiding the user by highlighting and animating information can be implemented in a straightforward manner. In addition, a so-called topographic texture layer can substitute geometry-based shading which drastically improves the perception of morphology and rendering performance.

Due to their effective and efficient visual design capabilities, the presented texturing techniques permit to implement new tools for presenting, exploring, and manipulating terrain data as required by a growing number of applications [19] such as spatial decision support systems, virtual reality applications, real-time GIS, and interactive cartographic environments [21]. We have implemented the presented texturing techniques as part of a real-time terrain visualization system [1] focusing on information visualization for cartography [2] and navigation systems.

The remainder of this paper is structured as follows: Section 2 briefly summarizes related work. Section 3 describes the texture

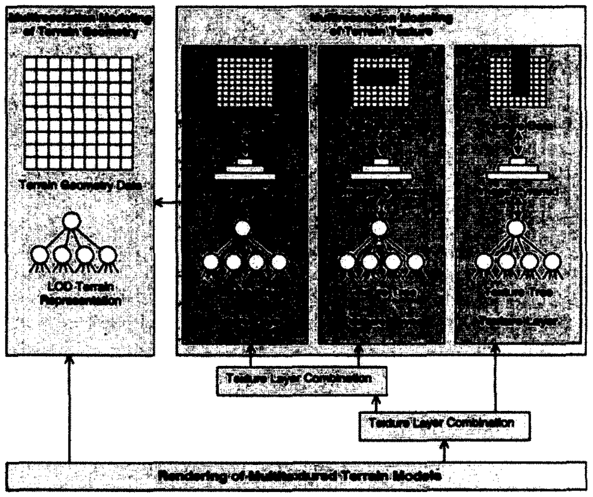
Multiresolution Modeling of Terrain Geometry
Multiresolution Modeling of Terrain Texture
Terrain Geometry Data
LOD Terrain Representation
Texture Tree
Texture Layer
Texture Layer Combination
Texture Layer Combination
Rendering of Multitextured Terrain Models

Figure 1: Conceptual model of multiple texture layers.

* Email: {dollner, kostab, khh}@uni-muenster.de
Department of Computer Science, University of Münster
Einsteinstraße 62, 48149 Münster, Germany

multiresolution model. Section 4 explains the rendering algorithm. Sections 5, 6, and 7 describe applications of the presented texturing techniques. Section 8 reports results, and Section 9 gives some conclusions.

2 RELATED WORK

Image-based modeling and rendering (IBMR) and many LOD techniques for terrain models have been developed for terrain visualization. Hierarchical triangulations for terrain geometry based on TINs are applied to generate multiresolution models which can be used by LOD algorithms (de Floriani et al. [8], Gross et al.[13], Voigtmann et al. [27], and Xia et al. [30]). Hoppe [15] introduces a general view-dependent LOD triangulation, called progressive mesh, and the geomorph, a technique to minimize popping effects [16]. Recent work takes into account both geometric and visual quality of a LOD representation (e.g., Hoppe [17]). Regular grids have been used for multiresolution modeling (Falby et al. [12]) and for real-time, continuous LOD rendering (Duchaineau et al. [11], Lindstrom et al. [20], and Pajarola [23]). As a common characteristic of these approaches, they do not provide an explicit model for multiple, multiresolution textures.

Several methods for real-time terrain rendering use IBMR techniques. Cohen et al. [4] as well as Soucy et al. [24] model a scene by combining 3D geometry and 2D image sprites; they create a texture map from an object represented at a high-resolution which is used to texture the same object represented at a lower resolution – re-introducing detail contained in the texture. Chen et al. [3] computes these textures at run-time and takes advantage of the frame-to-frame coherence in screen-space. These approaches use texturing basically to substitute geometric modeling with IBMR in order to reduce geometric complexity of terrain models; they do not focus on texturing as a mechanism for the visual design of terrain contents.

The FlyAway system (Hüttner [18]) uses a mipmap-approach for a single, memory-resident, texture which must satisfy the constraints imposed by the graphics system. Lindstrom et al. [20] and Hüttner [18] proposed a method which handles a single large-scale texture related to a LOD terrain geometry. Cline and Egbert [6] discuss how one large-scale texture, understood as bandwidth-limited resource, can be efficiently treated for terrain rendering using a cache hierarchy. The clipmap [25] provides a hardware-supported mechanism to handle a single extremely large texture such as needed by earth-wide visualizations whereas our data structure simulates a clipmap-like behavior without specialized hardware. A texture paging mechanism for large-scale external texture data has been investigated by Davis et al. [7]. Most other terrain visualization approaches are limited with respect to the management of large-scale texture data: In contrast to the LOD mechanism for geometry data, no similar LOD management is provided for texture data.

Most terrain visualization techniques consider textures to be images superimposed on geometry. The applications of textures, however, go far beyond as recent developments in real-time rendering show (e.g. elevation maps [9] and per-pixel lighting [10]).

3 MULTIRESOLUTION MODELS

Conceptually, the multiresolution model for texture data is independent from the concrete multiresolution model used for terrain geometry. We introduce briefly a generic multiresolution model for geometry used to explain the links between both types of multiresolution models.

3.1 Geometry Multiresolution Model

Multiresolution models for terrain geometry are essential to visualize large-scale geometry terrain data. We summarize briefly the key properties of a generic multiresolution model for terrain geometry which hold for other known LOD terrain models as well.

Let G be a terrain surface and $H(G)$ be a corresponding hierarchical terrain model represented by a tree. The tree nodes are called *geometry patches*. Each geometry patch N represents a rectangular region within the terrain domain, i.e., $D(N) \subseteq D(G)$, and approximates the terrain surface G in that region by an approximating terrain surface $G(N)$. The way the node calculates $G(N)$ depends on the *type* of the given terrain surface and the *approximation strategy* adopted by the LOD algorithm. For example, a grid-based node could select evenly spaced points from a grid data set, whereas a TIN-based node could select points from an arbitrary data set based on an error criterion. The *geometric approximation error* $\varepsilon(N)$ of a geometry patch N is defined as the maximal vertical distance between the terrain surface G and the approximating surface $G(N)$.

The hierarchical representation defines how many child nodes a geometry patch N can have. In general, the child nodes are constructed as follows: If the geometric approximation error $\varepsilon(N)$ exceeds a certain threshold $\varepsilon \geq 0$, the domain $D(N)$ is decomposed into a set of at most d rectangular, disjoint sub-domains $D(N_i)$. The strategy for decomposing a patch depends on the type of LOD technique: a grid-based technique could apply a quadtree-like subdivision, whereas a TIN-based technique could subdivide by a line parallel to the x- or y-axis such that the number of points on each side of the line are nearly equal. For each sub-domain $D(N_i)$, a child node N_i of N is constructed which approximates the terrain surface in that sub-domain. The domain of the root node of $H(G)$ is $D(G)$ covering the whole domain of the terrain surface G. A sample implementation can be found in [1].

3.2 Texture Multiresolution Model

Multiresolution modeling for texture data is as important as for geometric data because large-scale texture data is unlikely to fit into texture memory or even into main memory. Furthermore, texturing techniques must go beyond the concept of a single texture covering the whole terrain: It must be possible to texture a terrain partially, to texture a terrain surface with multiple, possibly overlapping textures, to combine and mask textures, and to generate textures dynamically. Our multiresolution model for terrain textures provides the foundations for this functionality. It preprocesses texture data by an internal *image pyramid* and organizes the texturing of a terrain hierarchically by a *texture tree*. Both constructs represent a *texture layer* of a terrain model.

3.2.1 Image Pyramid

The *image pyramid* [28] is derived from the original image data, which might be contained, for example, in a 2D image file. The image pyramid of a texture layer consists of a sequence of images with decreasing resolution. Each image is created by scaling down the predecessor image by a factor of ½. The first image of the sequence is identical to the original image data, the last image consists of 1 x 1 pixels. The initial image can have an arbitrary size: Cartographic textures and satellite images, for example, are likely to have a width and height of several ten thousands pixels. In addition, the pyramid can store further images at each level which are anisotropically scaled versions of the main image of the corresponding level, used to reduce blurring effects [29]. In contrast to [18] which uses several image pyramids, called *MPGrids*, we use only one image pyramid for a terrain texture, and there is no restriction to the sizes of the images used as terrain textures, such as that the sizes must be powers of 2.

The implementation is facilitated by memory-mapped files provided by the operating system which permit an application to map the contents of a file directly to its virtual address space. Memory-mapped files are useful for managing extremely large images since their usage consumes few physical resources – only small portions of the file are mapped into the physical address space. Without memory-mapping, image files not fitting into main memory could not be used as initial images of image pyramids.

3.2.2 Texture Tree

A *texture tree* is specified by a tree of nodes, which are called *texture patches*. The texture tree is associated with the hierarchical model $H(G)$ of the terrain geometry. Each texture patch M is related to exactly one geometry patch N_M, covering (or at least overlapping) the domain of the geometry patch N_M with respect to its geo-referenced coordinates. Therefore, the texture tree has a similar structure as the corresponding geometry tree $H(G)$. Compared to the hierarchical model of the terrain geometry, however, the texture tree may prune its sub-trees for those texture tree nodes which reference already a sub-image of the image pyramid that has full resolution. If such a node would have child nodes, they would require additional textures, but these textures would have the same texture resolution as the large texture of their parent node, i.e., there would be no quality improvement.

A texture patch M specifies that sub-image of one of the pyramid's images that covers the domain of the associated geometry patch, i.e., $D(M) \supseteq D(N_M)$, and has the highest resolution while still satisfying the texture constraints imposed by the rendering system. For example, each OpenGL implementation defines a maximal texture size (e.g., 1024 x 1024) and requires that the texture size is a power of 2 [29]. Since a texture patch specifies only the region of the corresponding sub-image for one of the pyramid levels, without storing the actual image data itself, a redundant replication of image data is avoided, even though the domains of the texture patches may overlap each other within a level of the texture tree.

3.2.3 Texture Layers

Technically, a *texture layer* is represented by an image pyramid and a texture tree (Figure 1). Conceptually, we can distinguish between four types of texture layers:

- **Thematic Texture Layer:** A thematic texture layer carries color information used as terrain content, typically imported from image files and implemented by a three channel RGB texture.
- **Luminance Texture Layer:** A luminance texture layer specifies brightness modifications applied to other texture layers. It is implemented as a one channel luminance texture.
- **Topographic Texture Layer:** A topographic texture layer specifies shading information for a terrain model calculated, for example, based on the high-resolution geometric model (see Section 7). It can be implemented as a specialized luminance texture layer.
- **Visibility Texture Layer:** A visibility texture layer specifies the visibility of other texture layers in terms of transparency (see Section 5). Technically, a one channel alpha texture is used for implementation.

Usually, thematic and topographic texture layers are precalculated and stored persistently, while visibility and luminance texture layers could be generated by the application, e.g., in response to user interaction and preferences. Geo-coordinates are needed in order to construct the terrain-aligned texture tree. The depth of the pyramid can be controlled by the application.

If more than one texture layer is used, it must be specified how these layers are combined in image space. The layer operations correspond to the texture blending operations supported by OpenGL and include:

- **Texture Layer Blending:** Two texture layers can be blended based on weights provided by a separate alpha texture, that is, the weights can vary from pixel to pixel. This way, arbitrarily shaped blending regions can be defined (see Section 5).
- **Weighted Texture Layer Addition:** Several texture layers can be added to a terrain surface. The weights are specified as constant factors for each of the texture layers (see Section 6).
- **Texture Layer Modulation:** The contents of two or more texture layers can be multiplied on a per-pixel basis. This technique is usually used for visualizing several independent thematic data layers at once and for adding the shading information provided by a topographic texture (see Section 7).

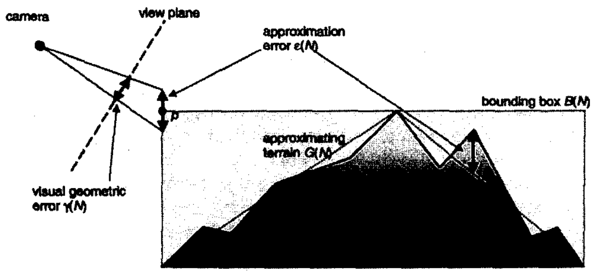

Figure 2: Visual geometric error $\gamma(N)$ and its calculation.

These operations can be applied successively, i.e., the result of one layer operation can be piped as input to another layer operation. Multiple texture layers require multiple rendering passes. If the hardware supports multitexturing, the number of actual rendering passes can be reduced.

Texture layers are independent from each other regardless of the layer operations since texture layers are combined in screen space. Consequently, no preprocessing and costly 2D imaging operations are needed to combine texture layers, which is important to achieve real-time rendering for dynamic texture layers. This technique, based on multi-pass rendering, can be scaled to future hardware designs that support multiple texture units.

4 RENDERING ALGORITHM

The rendering algorithm simultaneously traverses the multiresolution model for terrain geometry and the texture trees for all active texture layers, selecting geometry patches and texture patches according to a user-defined visual geometric error threshold $\gamma_{user} \geq 0$ and a user-defined visual texture error threshold $\tau_{user} \geq 0$.

We start describing the rendering algorithm for a hierarchical terrain model $H(G)$ without any active texture tree, and refine this algorithm step by step to conclude with a rendering algorithm which can manage a hierarchical terrain model together with an arbitrary number of active texture layers.

Let $B(N)$ be the axis-parallel 3D bounding box containing the approximating terrain surface $G(N)$ of the geometry patch $N \in H(G)$. If $B(N)$ intersects the current view frustum, the *visual geometric error* $\gamma(N)$ is defined as follows: Determine the nearest point $p \in B(N)$ to the current camera position inside the view frustum. Construct a line segment of length $\varepsilon(N)$, centered at p, and parallel to the z-axis (the direction of elevation). Project this segment onto the view plane. The visual geometric error $\gamma(N)$ is the length of this projected segment measured in screen pixels (Figure 2). If N has no child patches we define $\gamma(N) := 0$. $\gamma(N)$ describes the maximum texel displacement measured in screen pixels and the quality of the silhouette of the approximating terrain surface $G(N)$ with respect to the current camera settings: the lower the value the smaller the deviation. Using this visual geometric error as a quality criterion we can now specify a recursively defined rendering algorithm:

```
Algorithm render(GeometryPatch N)
    if(B(N) ∩ view frustum = ∅)
        // hierarchical view frustum culling
        return;
    if(γ(N) > γ_user)
        // recursive refinement
        for(each child N_i ∈ N)
            render(N_i);
        return;
    // painting
    paint(N);
```

The recursion is started by calling the rendering algorithm for the root patch $N_{\text{root}} \in H(G)$.

Similarly to the visual geometric error $\gamma(N)$ we can define the *visual texture error* $\tau(N,M)$ for a given geometry-texture-patch pair (N,M) with $D(N) \subseteq D(M)$ as follows: Determine the width w and height h of a texel of M in the coordinate system of the terrain surface. Construct two line segments centered at p: one parallel to the x-axis of length w, the other parallel to the y-axis of length h. The visual texture error $\tau(N,M)$ is the maximum length of both segments projected to the view plane and describes the maximum size of a texel of M in screen pixels. If M has no child patches we define $\tau(N,M) := 0$.

With this additional quality criterion we can modify the rendering algorithm to handle one texture tree as well. The algorithm traverses the geometry tree and the texture tree simultaneously. It exploits the fact that the structure of the texture tree is the same as that of the geometry tree, except that some sub-trees may be missing. Hence, we use an additional test to determine whether M has a corresponding child patch:

```
Algorithm renderWithTexture(GeometryPatch N,
                            TexturePatch M)
    if(B(N) ∩ view frustum = ∅) return;
    if((γ(N) > γ_user) or (τ(N,M) > τ_user))
        // recursive refinement
        for(each child N_i ∈ N)
            if(M has child M_i)
                renderWithTexture(N_i,M_i);
            else
                renderWithTexture(N_i,M);
        return;
    paintWithTexture(N,M);
```

The recursion is started by calling the rendering algorithm for the geometry-texture-root-patch pair $(N_{\text{root}},M_{\text{root}})$.

Note that a texture patch M can be used for texturing all geometry patches N with $D(M) \supseteq D(N)$. Thus, if a geometry-texture-patch pair (N,M) has been chosen for painting by the rendering algorithm, not only M can be used for texturing the geometry patch N, but also the parent texture patch M_{parent} of M can be used, since $D(M_{\text{parent}}) \supset D(M) \supseteq D(N)$ holds:

```
Algorithm renderWithTexture(GeometryPatch N,
                            TexturePatch M)
    if(B(N) ∩ view frustum = ∅) return;
    if((γ(N) > γ_user) or (τ(N,M) > τ_user))
        ...; return;
    while(τ(N,M_parent) ≤ τ_user)
        // reduction of texture data
        M = M_parent;
    while(data of M is not loaded)
        // ensuring interactivity
        M.requestTextureData(); // non-blocking!!!
        M = M_parent;
    paintWithTexture(N,M);
```

We can exploit this fact for two different purposes:

- If we use the parent texture patch M_{parent} instead of M for texturing, a single texture patch can be used for texturing more than one geometry patch. This reduces the amount of texture data to be loaded and processed during the rendering, and thus reduces memory requirements and increases performance of the algorithm. We can recursively choose the parent texture patch as long as this texture patch satisfies the user-defined quality criterion τ_{user} (first `while`-loop of the algorithm).
- Only a part of the image pyramid is actually required to render a single frame. Therefore, a texture patch M loads its texture data only on demand, spanning a separate thread (non-blocking call of `M.requestTextureData()`). While the texture data is being loaded, the texture data of the parent texture patch M_{parent} can be used instead. In this case, texture resolution is not optimal, but *interactivity is ensured* because the application is not blocked and can use at least a reasonable approximation of the required texture. If the user-defined texture capacity of the main memory is exceeded, texture data is unloaded for the least recently used (LRU) texture patches.

For a set $S = \{M_1,\dots,M_n\}$ of texture patches M_i we define $\tau(N,S) := \max_{M \in S} \tau(N,M)$. The algorithm for dealing with a set of active texture trees is essentially the same as the algorithm for one texture tree, but with an additional loop iterating over the set of texture patches, implementing a simultaneous traversal of all trees:

```
Algorithm renderWithTextures(GeometryPatch N,
                             Set<TexturePatch> S)
    if(B(N) ∩ view frustum = ∅) return;
    if((γ(N) > γ_user) or (τ(N,S) > τ_user))
        ...; return;
    Set<TexturePatch> T = ∅;
    for(each M ∈ S)
        while(τ(N,M_parent) ≤ τ_user)
            M = M_parent;
        while(data of M is not loaded)
            M.requestTextureData();
            M = M_parent;
        T.insert(M);
    paintWithTextures(N,T);
```

The algorithm `paintWithTextures(N,T)` gets one geometry patch N and a set T of n texture patches. It renders the approximating terrain surface $G(N)$ n times, each time texturing it with the texture data of another texture patch. The result of each pass (a 2D color image) is combined with the results of the previous passes (already stored in the frame buffer) using 2D image operations like multiplication, addition, blending or interpolation. Thus, the process of combining all texture layers is performed solely in screen-space. If the graphics hardware supports multitexturing with m texture units, the algorithm can take advantage of it and render $G(N)$ applying m textures simultaneously, reducing the number of needed passes to $\lceil n/m \rceil$. Newer graphics hardware commonly supports 2 texture units, but support for 4 texture units are already announced and can be expected in the near future.

Sometimes a texture T does not cover the whole terrain surface. For example, high resolution satellite images may only be available for some parts of the terrain domain. In order to handle this case as well, we have to modify the properties of a texture patch and the rendering algorithm.

A texture patch M can have fewer children than the associated geometry patch N_M. The missing children correspond to the children of N_M which do not overlap with $D(T)$. The property of covering the whole domain of N_M ($D(N_M) \subseteq D(M)$) can be weakened to overlapping ($D(N_M) \cap D(M) \neq \emptyset$) for those geometry patches which are not totally covered by $D(T)$ but overlap with it. The recursive refinement part of the algorithm has to be changed accordingly by one additional test: we check whether the corresponding child texture patch exists:

```
Algorithm renderWithTextures(GeometryPatch N,
                             Set<TexturePatch> S)
    if(B(N) ∩ view frustum = ∅) return;
    if((γ(N) > γ_user) or (τ(N,S) > τ_user))
        for(each child N_i of N)
            Set<TexturePatch> T = ∅;
            for(each M ∈ S)
                if(M has child M_i)
                    T.insert(M_i);
            renderWithTextures(N_i,T);
        return;
```

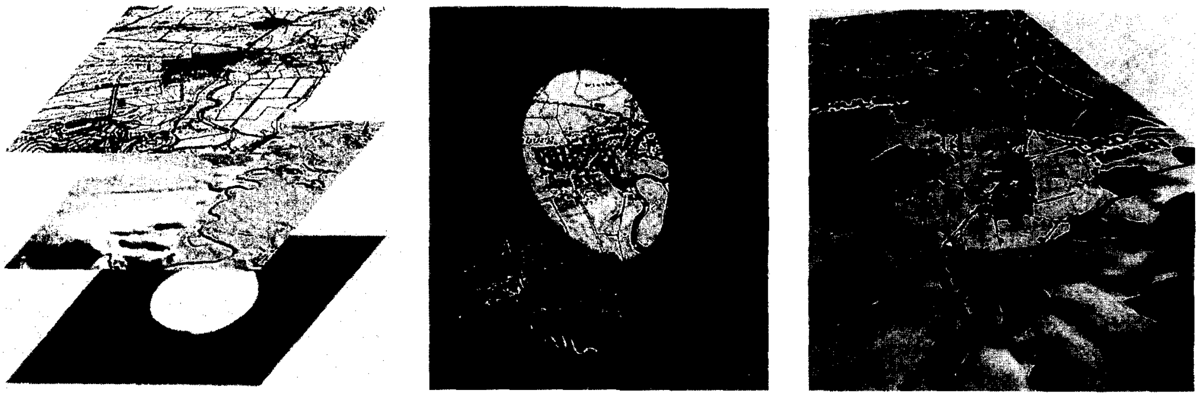
Figure 3: Texture lenses. Conceptual view of the topographic, thematic, and luminance texture layers (*left*) modeling a highlight lens used to focus spatial positions (*center*). Partial combination of two thematic texture layers by a visibility texture layer (*right*).

For textures covering the terrain partially there should be a white border of one pixel around the texture since OpenGL uses this border color for texturing parts of graphics primitives that are not covered by the applied texture.

5 TEXTURE LENSES

As a direct application, multiple thematic data sets can be superimposed on the terrain by multiple thematic texture layers which are added or blended (e.g., land use information and temperature).

For one or more texture layers, we can specify a luminance texture layer to highlight certain regions of the terrain. Such a visual focus control can be used, for example, to guide the attention of the user. In Figure 3, the conceptual model (*left*) and the resulting image (*center*) of a terrain with a topographic and thematic texture layer are illustrated.

For two thematic texture layers, we can specify the visibility regions of both layers by an additional visibility texture layer which contains the weights used to combine both thematic texture layers. For example, to model an interactive thematic texture lens, the visibility texture layer defines a circular region with large weights around a hot spot, controlled by user interaction; only a low texture resolution (e.g., 128x128) is actually needed (Figure 3 *right*). In the case of static lens shapes, it suffices to manipulate the texture matrix of the visibility texture layer, that is, the underlying alpha texture needs not to be recalculated at all.

6 TEXTURE ANIMATIONS

Dynamic terrain data can be represented by a sequence of texture layers. During animation, two consecutive texture layers are active according to their corresponding time coordinates. These layers are rendered with the weights α and $(1-\alpha)$, respectively. No intermediate texture has to be created which allows us to animate even high-resolution texture sequences. Both active texture layers represent a weighted texture layer combination and can be rendered together with other texture layers, e.g., thematic texture layers.

Figure 4 shows a flooding animation. The texture key frames are 1200 x 2400 pixels large and describe the flooding state in a landscape at discrete time stamps.

7 TOPOGRAPHIC TEXTURES

The user perceives and recognizes the morphology of a terrain mainly by the silhouette and shading of the terrain model. We take this into account by so-called topographic textures, provided as a separate texture layer, which improve the perception of a terrain.

7.1 Appearance Preservation

The strategies rely on the following principle: a high-resolution topographic texture containing appearance information such as shades is pre-computed based on the original geometric model and stored separately [5]. Thus, visual detail is re-introduced into the LOD terrain model since topographic textures are applied to terrain patches in screen-space, that is, a pixel-precise shading is obtained even for low-resolution terrain parts. In particular, a wrong visual impression of the terrain's topography which would result for simplified, low-resolution parts of the terrain is avoided.

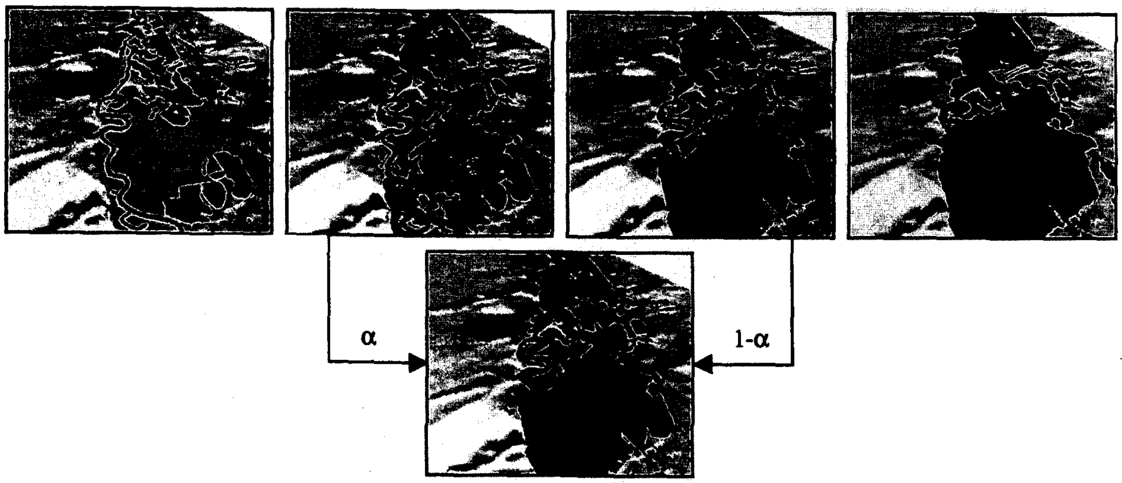

Figure 4: Animation of temporal flooding data by interpolation between two consecutive textures of a texture layer sequence.

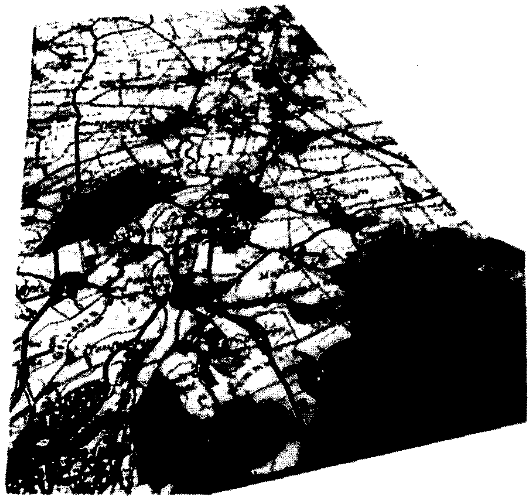

Figure 5: Topographic texture layer with self-shadowing combined with cartographic thematic texture layer.

The shading quality depends only on the resolution of the topographic texture and not on the geometric resolution.

No triangle-based shading is required during rendering, saving per-vertex lighting calculations. Both impressive speed and quality improvements can be achieved using topographic textures on low-cost 3D graphics hardware with accelerated texture mapping because the texture-based approach bypasses the (generally limited) geometric processing capabilities of these platforms.

7.2 Calculating Topographic Textures

The calculation of a topographic texture may depend on surface properties, surface geometry, topographic features, light sources, and shading rules (e.g., cartographic terrain shading). In addition, it can take into account topographic features such as peaks, pits, saddles, valleys, ridges and other landform elements [26].

Topographic textures can be calculated in an automated preprocessing step by an orthogonal projection of the full-resolution, illuminated terrain model. Then, the frame buffer content is used as image data to construct a texture layer. To achieve a resolution higher than the maximal frame buffer size, the topographic texture can be composed of tiles. The full-resolution terrain model can be shaded using the standard OpenGL lighting, an application-specific illumination model (e.g., cartographic hill shading), or it can be based on elevation mapping [9].

In Figure 5, the topographic texture takes into account self shadowing of the terrain model. The self-shadowing is calculated by a ray intersection test between light source and full-resolution terrain model. In the example, the topographic texture layer is modulated with a cartographic texture layer.

7.3 Hybrid Shading

Since the quality of the shading depends on the resolution of the topographic texture, the resolution can become insufficient if the camera gets very close to the terrain surface. For all parts close to the camera, we can apply standard Gouraud shading instead of topographic texturing in order to prevent blurring effects (Figure 6).

We identify these parts of ***insufficient topographic texture resolution*** by evaluating the visual texture error $\gamma(N,M)$. If M is a leaf patch of a topographic texture tree and $\gamma(N,M)$ exceeds the

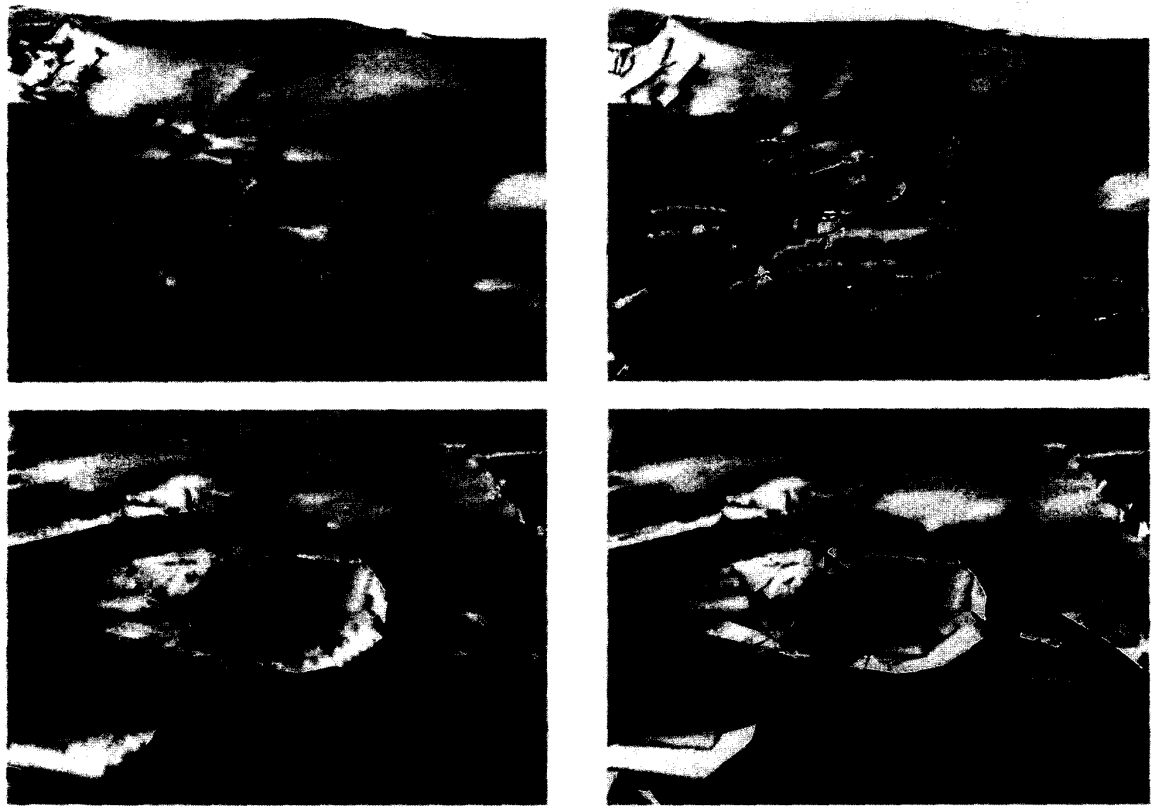

Figure 6: Gouraud-shaded terrain (*upper left*). Same terrain using a topographic texture (*upper right*). Closer view of the terrain with a topographic texture (*lower left*). Same view using a hybrid shading scheme (*lower right*).

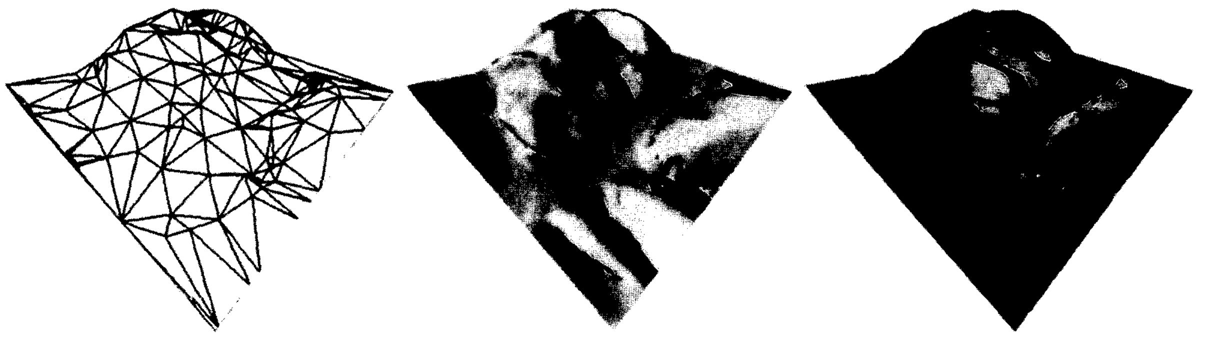

Figure 7: Topographic textures encoding surface normals: wire-frame (*left*), directional point light (*center*), point light source (*right*).

user-defined threshold γ_{user} then the topographic texture is switched off and the standard lighting calculations are used for shading the geometry patch *N*.

This approach combines the advantages of both shading strategies: it provides pixel-precise shading for geometry patches with a coarse approximation and object-precise shading for geometry patches with small approximation errors for which the topographic texture resolution is insufficient.

7.4 Encoding Surface Normals by Topographic Textures

New graphics hardware (e.g., *GeForce* from *NVidia*) support *register combiners* [22] that can perform dot-product calculations based on texel data for each rasterized fragment. Used for topographic texturing, this permits the implementation of dynamic shading effects, such as changing the direction of a directional light source or simulating a point light source moving above the terrain surface, without the need to recalculate the topographic texture.

To encode surface normals, the topographic texture stores normal vectors of a high-resolution terrain surface as an RGB-texture. During the texturing process the dot-product of the encoded surface normal vectors and the direction of the light vector is used as a luminance value simulating the shading of the terrain surface. Note that this shading simulation is based on a per-pixel calculation so the shading information is as precise as the topographic texture is. We can also apply the hybrid shading scheme explained in Section 7.3 for terrain regions where the resolution of the topographic texture is inadequate.

For a directional light source the light vector is constant for the whole texturing process. For the simulation of a point light source we use a second texture which encodes the light vector just like the topographic texture does for the surface normal vectors. By perspective projection of this light vector texture onto the terrain surface as a second texture and moving the center of this projection we can simulate a moving point light source which can be arbitrarily placed above the terrain surface.

Figure 7 *left* shows the triangles used to render the images in the *center* and to the *right*. The *center* image is rendered with a directional light source. The *right* image is rendered with a simulated point light source. Both images use the same topographic texture without the need for recalculation.

8 RESULTS

The time measurements presented in Figure 8 were performed on a standard PC equipped with a 400 MHz Pentium II processor, 256 MB RAM, Riva TNT2 graphics card with 16 MB graphics memory and 2 texture units, and running Windows NT 4.0. The terrain data set consists of about 500.000 triangles and has been rendered in a 640x480 pixels true-color canvas. For each test, 1000 frames have been generated flying over the terrain with an average terrain visibility of 70%.

Figure 8 *left* compares the standard Gouraud shading with the topographic texturing approach presented in Section 7. The usage of a topographic texturing compared to Gouraud shading has no noticeable impact on rendering performance if we take the same number of rendered triangles. But to achieve the visual quality of topographically textured terrains, the number of rendered triangles would have to be drastically increased if standard Gouraud shading is used. Therefore, the topographic texturing approach is actually more efficient.

Figure 8 *right* compares the impact of the number of active texture layers on rendering performance. Visualizing many independent texture layers can be done efficiently using the multitexturing capabilities of modern graphics hardware.

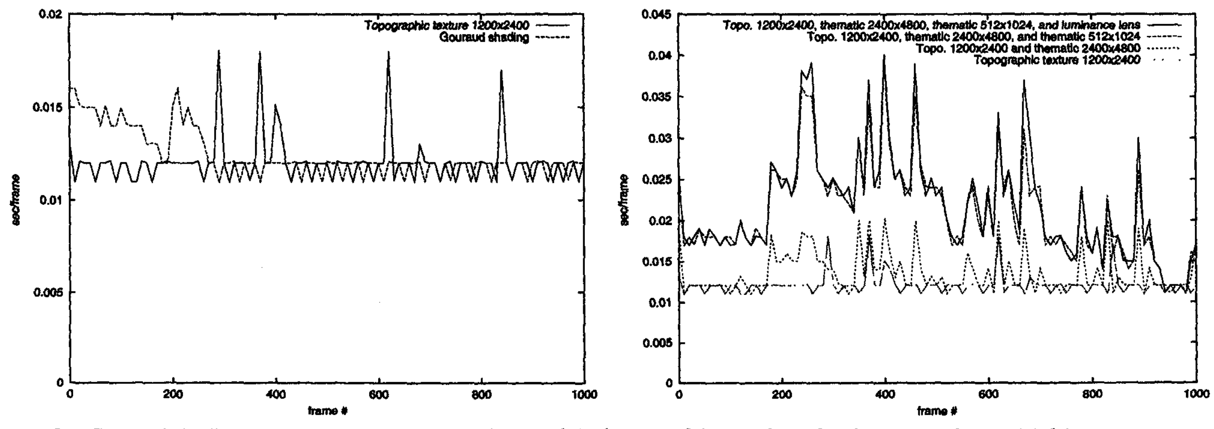

Figure 8: Gouraud shading vs. topographic texturing (*left*), and the impact of the number of active texture layers (*right*).

9 CONCLUSIONS

The visual design of terrain contents becomes more and more important because terrain models form the graphics basis for any kind of spatial data visualization. The presented texturing techniques enable terrain visualization applications to design interactive, animated terrain textures. The multiresolution model for texture data can be integrated with and complements most existing LOD terrain models. Real-time rendering with multiple texture layers relies on multitexturing which can be implemented efficiently using today's graphics hardware. The approach can be extended towards procedural texture definitions which could be used for visualizing motion and flow. Also the application of multiple texture layers as a technical tool for information visualization has to be investigated. Technical details and the prototype implementation are available at the following WWW site:

`http://www.mamvrs.de/geovisualiz.htm`

References

[1] K. Baumann, J. Döllner, K. Hinrichs, O. Kersting: A Hybrid, Hierarchical Data Structure for Real-Time Terrain Visualization. *Proceedings Computer Graphics International '99*, 85-92, 1999.

[2] G. Buziek, J. Döllner: Concept and Implementation of an Interactive, Cartographic Virtual Reality System. *International Cartographic Conference*, Ottawa, 1999.

[3] B. Chen, J. Swan, E. Kuo, A. Kaufman: LOD-Sprite Technique for Accelerated Terrain Rendering. *Proceedings IEEE Visualization '99*, 1999.

[4] J. Cohen, M. Olano, D. Manocha: Appearance-Preserving Simplification. *Proceedings of SIGGRAPH '98*, 115-122, 1998.

[5] P. Cignoni, C. Montani, C. Rocchini, R. Scopigno: A General Method for Preserving Attribute Values on Simplified Meshes. *Proceedings IEEE Visualization '98*, 59-66, 1998.

[6] D. Cline, P. Egbert: Interactive Display of Very Large Textures. *Proceedings IEEE Visualization '98*, 343-350, 1998.

[7] D. Davis, T. Y Jiang, W. Ribarsky, N. Faust: Intent, Perception, and Out-of-Core Visualization Applied to Terrain. *Proceedings IEEE Visualization '98*, 455-458, 1998.

[8] L. De Floriani, P. Magillo, E. Puppo: Efficient Implementation of Multi-Triangulations. *Proceedings IEEE Visualization '98*, 1998.

[9] S. Dietrich: Elevation Maps. NVIDIA Corporation, White Paper, http://www.nvidia.com, 2000.

[10] S. Dietrich: Dot Product Texture Blending and Per-Pixel Lighting. NVIDIA Corporation, White Paper, http://www.nvidia.com, 2000.

[11] M. Duchaineau, M. Wolinsky, D. Sigeti, M. Miller, C. Aldrich, M. Mineev-Weinstein: ROAMing Terrain: Real-time Optimally Adapting Meshes, *Proceedings IEEE Visualization '97*, 81-88, 1997.

[12] J. Falby, M. Zyda, D. Pratt, R. Mackey: NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation. *Computers & Graphics*, 17(1): 65-69, 1993.

[13] M. Gross, R. Gatti, O. Staadt: Fast Multiresolution Surface Meshing. *Proceedings IEEE Visualization '95*, 135-142, 1995.

[14] P. Haeberli, M. Segal: Texture Mapping as a Fundamental Drawing Primitive. *Proceedings of the 4th Eurographics Workshop on Rendering*, M. Cohen, C. Puech, F. Sillion (Eds.), 259-266, 1993.

[15] H. Hoppe: Progressive Meshes. *Proceedings of SIGGRAPH '96*, 99-108, 1996.

[16] H. Hoppe: View-Dependent Refinement of Progressive Meshes. *Proceedings of SIGGRAPH '97*, 189-198, 1997.

[17] H. Hoppe: New quadric metric for simplifying meshes with appearance attributes. *Proceedings IEEE Visualization '99*, 59-66, 1999.

[18] T. Hüttner, W. Strasser: FlyAway: a 3D terrain visualization system using multiresolution principles. *Computers & Graphics*, 23: 479-485, 1999.

[19] M. Kraak: Interactive Modelling Environment for Three-dimensional Maps: Functionality and Interface Issues. In: A. M. MacEachren, D. R. Fraser Taylor (Eds.), *Visualization in Modern Cartography*, Pergamon, 269-285, 1994.

[20] O. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, G. A. Turner: Real-time continuous level of detail rendering of height fields. *Proceedings of SIGGRAPH '96*, 109-118, 1996.

[21] A. M. MacEachren, D. R. Fraser Taylor: Visualization in Modern Cartography. *Modern Cartography*, Vol. 2, Pergamon, 1994.

[22] NVIDIA Corporation. NVIDIA OpenGL Extension Specifications, http://www.nvidia.com, 1999.

[23] R. Pajarola: Large Scale Terrain Visualization Using The Restricted Quadtree Triangulation. *Proceedings IEEE Visualization '98*, 19-26, 1998.

[24] M. Soucy, G. Godin, M. Rioux: A texture-mapping approach for the compression of colored 3D triangulations. *The Visual Computer*, 12(10): 503-514, 1996.

[25] C. C. Tanner, C. J. Migdal, M. T. Jones: The Clipmap: A Virtual Mipmap. *Proceedings of SIGGRAPH '98*, 151-159,1998

[26] M. van Kreveld: Digital Elevation Models and TIN Algorithms. In: M. van Kreveld, J. Nievergelt, T. Roos, P. Widmayer (Eds.), *Algorithmic Foundations of Geographic Information Systems*, Lecture Notes Computer Science, Springer-Verlag, 1340: 37-78, 1997.

[27] A. Voigtmann, L. Becker, K. Hinrichs: A Hierarchical Model for Multiresolution Surface Reconstruction. *Graphical Models and Image Processing*, 59: 333-348, 1997.

[28] L. Williams: Pyramidal Parametrics. *Proceedings of SIGGRAPH '83*, 17(3): 1-11, 1983

[29] M. Woo, J. Neider, T. Davis, D. Shreiner: OpenGL Programming Guide, 3rd Edition, Addison-Wesley, 1999.

[30] J. Xia, J. El-Sana, A. Varshney: Adaptive Real-Time Level-of-detail-based Rendering for Polygonal Models. *IEEE Transactions on Visualization and Computer Graphics '97*, 3(2): 171-183, 1997.