3D Isocontours – Real-time Generation and Visualization of 3D Stepped Terrain Models

T. Glander, M. Trapp and J. Döllner

Hasso-Plattner-Institut, University of Potsdam, Germany

Abstract
Isocontours (also isopleths, isolines, level sets) are commonly used to visualize real-valued data defined over a 2D plane according to a set of given isovalues. To support the 3D landscape metaphor for information visualization, a 3D stepped terrain can be derived by lifting and extruding isolines to their particular isovalue, but typically requires triangulation of the resulting surface representation in a preprocessing step. We present a concept and rendering technique for triangle-based terrain models that provide interactive, adaptive generation and visualization of such stepped terrains without preprocessing. Our fully hardware-accelerated rendering technique creates additional step geometry for each triangle intersecting an iso-plane on-the-fly. Further, an additional interpolation schema facilitates smooth transition between established 3D terrain visualization and its stepped variant.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems I.3.7 [Computer Graphics]: Picture/Three-Dimensional Graphics and Realism—Animation

1. Introduction

Isolines or isopleths are closed contours that are traditionally used on maps to visualize terrain height, originally especially in mountain cartography [Imh07]. In addition to height of a terrain model, arbitrary real-valued data defined over a 2D domain is commonly presented using isoline visualization. 2D isolines apply quantization to the displayed variable and map it directly to color, 3D visualization using the terrain metaphor applies quantized colors to continuous surface. We suggest to complement these traditional alternatives with stepped terrain visualization, in which quantization is applied not only to color, but also to geometry (Fig. 1).

We present a technique for triangle-based terrain models that provides real-time rendering for a 3D stepped terrain visualization. Using GPU programs, vertex positions are adapted to the nearest particular isovalue. Further, each triangle intersecting an isoplane is tesselated into geometry consisting of vertical or horizontal triangles to form a sharp stair structure. The resulting values can be blended according to a given factor with the original values, allowing intermediate states.

Our technique makes the following main contributions:

- Stepped terrain geometry is created dynamically, supporting interactive changes, e.g., through time-variant data or changing intervals of the contours.
- The proposed interpolation schema enables smooth blending of the original data, allowing for focus+context visualization with soft-edged lenses or camera dependent visualization.

2. Previous work

Visualization of terrains was studied extensively in cartography and presented, for example, by [Imh07]. Regarding layered relief depictions, Tanaka studied their construction and the effect of lighting, therefore these maps often are called Tanaka contour maps though being used earlier in similar form [Tan50]. Today, most researchers dealing with terrain visualization concentrate on photorealistic rendering of massive data models, e.g. [DKW09, PG07]. Methods to apply cartographic visualization styles on terrains were presented in [D07] based on image space operations.

In information visualization, 2D data is often presented using a terrain metaphor [SML06], either by mapping the variable of interest to color (2D) or height (3D). Here, isocontours reveal classes in the data. A typical application is

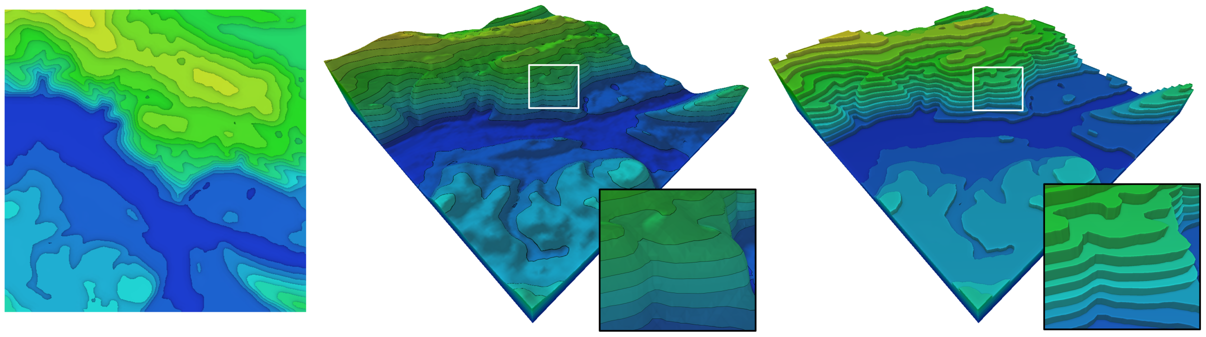


Figure 1: *Traditional isoline visualization (left), continuous 3D terrain with draped isolines (center), our stepped 3D terrain visualization (right).*

dot spatialization, e.g., in visualization of document collections: to show clusters in the data, a density field is computed and presented as a terrain surface. The appropriateness of 3D vs 2D visualization methods depends on the use case and is subject to experiments [TSD09, CM01].

Existing approaches for geometric contour extraction require precomputed index structures, and identify seed sets of cells containing or points lying on a contour [CSA03]. The focus was also put on I/O-efficient processing of massive data sets, e.g., [SCESL02].

A similar problem is approached by the marching cubes algorithm [LC87]: by classifying cells of a 3D scalar field, fitting geometry from a lookup table is placed in the cell. This idea was also transferred to graphics hardware to perform isosurface extraction in realtime, e.g., by [TSD07]. Even before the advent of GPUs capable of geometry generation, they where used to accelerate visualization of isosurfaces, as surveyed in [SCESL02].

3. Rendering

3.1. Preliminaries

Input data commonly used in 2.5D terrain visualization can be defined as a function $f(x,y) = z$ that maps continuous coordinates $(x,y) \in \mathbb{R}^2$ to a continuous value $z \in \mathbb{R}$. In practice, this typically amounts to a grid or a set of triangles that approximates the surface through piecewise planar primitives.

Isovalues are a discrete, uniformly distributed and typically small set of real values $\alpha \in \mathcal{I} \subset \mathbb{R}$. They implicitly define *isolines* as $g(f(x,y),\mathcal{I}) = \{(x,y) : f(x,y) \in \mathcal{I}\}$. 2D plots of $f(x,y)$ use a mapping for values of z to a color, and display isolines in a distictable color. For presentation, the color between isolines is often also quantized according to the isolines.

A 3D visualization can be naturally obtained using $f(x,y) = z$ directly for height. The 3D surface is then described in $\mathbb{R}^3$ by all points $p = (x,y,z)$ that satisfy f. Accordingly, isolines are given by $h(f(x,y),\mathcal{I}) = \{(x,y,z) \in \mathbb{R}^3 : f(x,y) \in \mathcal{I}\}$. For 3D stepped terrain visualization, we have to subdivide triangles crossing isolines and create new geometry representing steps. A threshold t_i halfway between two subsequent isovalues α_i, α_{i+1} describes the split height.

3.2. Step tessellation

We use GPU programs at vertex, primitive, and fragment level to create 3D stepped terrain visualization.

At vertex level, we quantize each vertex' height, adapting it to its nearest isovalue α_i (Fig. 2, left). The major part is done at primitive, i.e., triangle, level, where we generate vertical walls approximating the isoline avoiding alignment to the original structure.

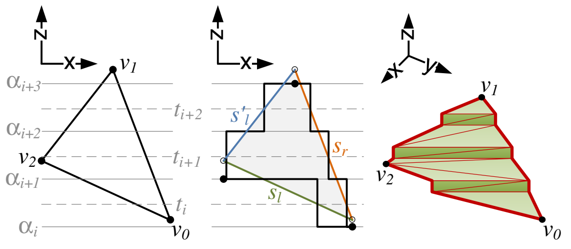


Figure 2: *Left: input triangle spanning several quantization levels, Center: segments s_l and s_r describe the triangle stepwise. Right: tesselation forming step geometry.*

As additional geometry is only needed for triangles crossing one or more isolines, we are only concerned with triangles, where the quantized height values $h_q(v_i)$ of their particular vertices v_i are different: $\bigvee_{i=0,1,2} h_q(v_i) \neq h_q(v_{i^+})$, with v_{i^+}, v_{i^-} defined cyclical. If this criterion is met by a triangle, we calculate its minimum height. Starting with the minimum height, the algorithm increases height in steps of the quantization interval until the maximum height is reached. To guide the geometry construction, we calculate and adapt a left and a right segment $s_l = [v_i, v_{i^-}]$, $s_r = [v_j, v_{j^+}]$ that describe the current part of the triangle (Fig. 2).

At each step, we do the following:

Identify configuration: All vertices' quantized heights are compared with the current height in counter-clockwise order starting with v_0. Zero to two vertices will be found and assigned to s_l, s_r. At the initial iteration, two vertices a_l, a_r are set to the lower boundaries of s_l, s_r. The segments remain unchanged if no new vertex was found.

Calculate threshold positions: At points p_l^t and p_r^t the segments cross the next threshold t. Their positions can be obtained by linearly interpolating each segment's boundaries using the threshold height t:

$$p_l^t = (1-\beta_l)v_i + \beta_l v_{i^-} \text{ with } \beta_l = \frac{t - h(v_i)}{h(v_{i^-}) - h(v_i)}$$

$$p_r^t = (1-\beta_r)v_j + \beta_r v_{j^+} \text{ with } \beta_r = \frac{t - h(v_j)}{h(v_{j^+}) - h(v_j)}$$

Create geometry By lifting p_l^t, p_r^t to the next isovalue α_{i+1} we obtain b_l, b_r, which complete the definition of the current step. The step consists of a horizontal quad with vertices $s_0 = a_l$, $s_1 = a_r$, s_3, s_2 and a vertical quad s_2, s_3, $s_5 = b_r$, $s_4 = b_l$. The concave vertices s_2, s_3 correspond to b_l, b_r with their height set to the current isolevel: $s_2 = \text{lift}(b_r, \alpha_i)$, $s_3 = \text{lift}(b_l, \alpha_i)$.
The algorithm generates these triangles (Fig. 3(a)) and sets color, normal, texture attributes as appropriate, using β_l, β_r to interpolate given values from the original vertices. Correct orientation is guaranteed by the differentiation of left and right segments. Finally, a_l is set to b_l and a_r to b_r for the next iteration.

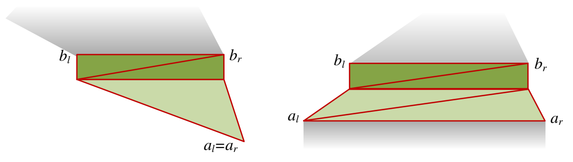
Special handling is needed for the case, that a single vertex v was identified in between (Fig. 3(c)). A new triangle (a_l, a_r, v) must be created and either a_l or a_r replaced by v, then, depending on the segment v belonged to.

The loop stops, after maximum height has been reached. To complete the step geometry, one finishing triangle – both segments' ends share the same vertex – or two finishing triangles – the segments have different ends – are generated (Fig. 3(b)).

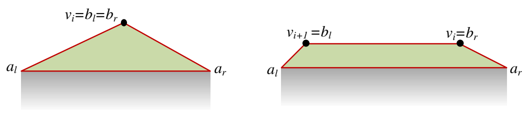
The tesselation creates step geometry based on input triangles and a desired quantization, only. If, additionally, a height field is given, the geometry can be smoothed by applying multi-sampling. Also newly calculated normals at the steps can be smoothed by specifying the same normals at vertices of neighboring triangles, using local gradients in the height field.

3.3. Interpolation schema

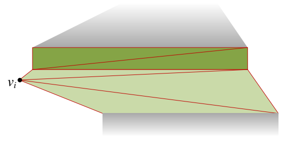
Some applications require the ability to generate intermediate states of the stepped terrain visualization, e.g., for smooth activation of the effect or a soft lens application. Therefore,



(a) Step triangles



(b) Closing triangles



(c) Complex configuration

Figure 3: *All configurations including complex cases (c) can be expressed by four basic tesselations (a,b).*

it is desirable to have an interpolation schema that allows for specifying the degree of the effect in an interval $\gamma \in [0,1]$.

To implement this, we have to linearly blend the quantized heights with the original height $v.z = (1-\gamma) \cdot v.z + \gamma \cdot h_q(v)$. For the step geometry, heights of the vertices on the isoline s_2, s_3, s_4, s_5 have to be blended with the threshold height t_i.

This enables to specify γ globally, apply it depending on camera distance, or individually according to a function defined on the same domain as the height field, i.e., a grayscale texture.

4. Application examples

Dynamic data, which changes according to an additional dimension, e.g., weather forecast, measured audio frequencies, 3D functions, is a common use case for isoline visualization. To demonstrate our technique on dynamic data, we generated a series of images from 3D noise function. At runtime, our application uses these images as height maps of an artificial terrain model, that are iterated at regular time intervals. In the rendering, emerging and collapsing peaks can be observed (see the accompanying video) in quantized visualization.

Lenses are used to show different data representations in the same place using a lens metaphor [BSP*93]. Through the presented interpolation schema, we are able to apply a smooth lens to the visualization, that shows the original continuous terrain model inside the lens, and show the abstracted, quantized model outside (Fig. 4). Our imple-

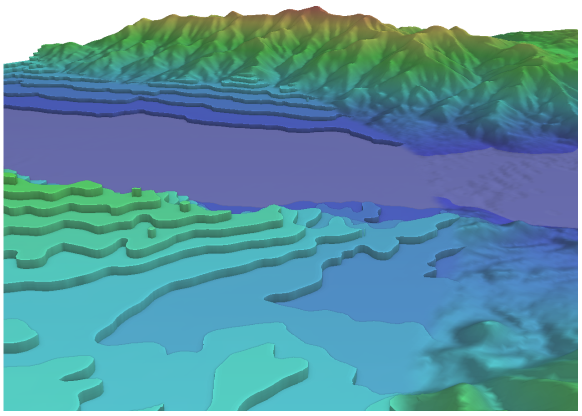


Figure 4: *The lens reveals the ridge on the backround while presenting layered steps on the left side and foreground.*

mentation is based on an additional grayscale texture to represent the lens by values in $[0,1]$.

5. Discussion

Method As our technique creates new geometry in the shader stage, it can be integrated easily into existing terrain rendering systems. While a simple shading approach would suffice for static view parameters, interactive exploration requires extra geometry to yield distinct silhouettes. An alternative would be to preprocess the terrain, which would also allow merging triangles on plateaus. For dynamic data, however, preprocessing *and* upload to the graphics card would occur each frame.
The bottleneck of our current approach is the complex decision tree run for each step of a triangle, the worst case being a triangle crossing all quantization levels.

Performance We tested our technique on an Intel Core2 Duo (3GHz) with 4 GB memory and nVidia Geforce 9800 GT graphics card having 512 MB VRAM. Our performance measurements show interactive rates for small data sets, dropping below 6 FPS for 256x256 sized terrain models (Table 1). It shows that the technique is limited by the size of the input model and the number of quantization steps. In our tests, screen resolution had no significant impact on the performance.

#steps	64x64	128x128	256x256
5	47.42	22.25	5.08
10	27.26	14.04	2.98
15	24.65	11.51	2.32
20	21.25	11.07	1.87

Table 1: *Frames-per-seconds measured at a screen resolution 1024x768 are given for different configurations.*

6. Conclusion

We present a novel concept and technique for the rendering of 3D terrain models using a stepped terrain visualization. Our approach creates the necessary geometry on-the-fly and is therefore suited for dynamic applications such as dynamic magic lenses and dynamic, e.g., time-variant data models.

Achnowledgements

This work has been funded by the German Federal Ministry of Education and Research (BMBF) as part of the InnoProfile research group "3D Geoinformation" (www.3dgi.de). We are grateful for helpful comments from the anonymous reviewers as well as from our colleagues.

References

[BSP*93] BIER E., STONE M., PIER K., BUXTON W., DEROSE T.: Toolglass and magic lenses: the see-through interface. In *Proceedings of the SIGGRAPH conference* (New York, New York, USA, 1993), ACM Press, pp. 73–80.

[CM01] COCKBURN A., MCKENZIE B.: 3D or not 3D? Evaluating the effect of the third dimension in a document management system. In *Proceedings of the SIGCHI conference* (New York, NY, USA, 2001), ACM Press, pp. 434–441.

[CSA03] CARR H., SNOEYINK J., AXEN U.: Computing contour trees in all dimensions. *Computational Geometry: Theory and Applications 24*, 2 (2003).

[DÖ7] DÖLLNER J.: *Non-Photorealistic 3D Geovisualization*, 2nd ed. Springer, Berlin, Heidelberg, 2007, pp. 229–240.

[DKW09] DICK C., KRUGER J., WESTERMANN R.: GPU Ray-Casting for Scalable Terrain Rendering. In *Proceedings of Eurographics 2009* (2009), pp. 43–50.

[Imh07] IMHOF E.: *Cartographic Relief Presentation*. ESRI Press, 2007.

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of the SIGGRAPH conference* (New York, NY, USA, 1987), ACM Press.

[PG07] PAJAROLA R., GOBBETTI E.: Survey of semi-regular multiresolution models for interactive terrain rendering. *The Visual Computer 23*, 8 (2007), 583–605.

[SCESL02] SILVA C., CHIANG Y., EL-SANA J., LINDSTROM P.: Out-of-core algorithms for scientific visualization and computer graphics. In *Visualization '02* (Los Alamitos, CA, USA, 2002), IEEE Computer Society Press.

[SML06] SCHROEDER W., MARTIN K., LORENSEN B.: *Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 4th Edition*. Kitware, 2006.

[Tan50] TANAKA K.: The Relief Contour Method of Representing Topography on Maps. *Geographical Review 40*, 3 (Januar 1950), 444–456.

[TSD07] TATARCHUK N., SHOPF J., DECORO C.: Real-Time Isosurface Extraction Using the GPU Programmable Geometry Pipeline. In *SIGGRAPH course notes* (2007).

[TSD09] TORY M., SWINDELLS C., DREEZER R.: Comparing dot and landscape spatializations for visual memory differences. *IEEE transactions on visualization and computer graphics 15*, 6 (2009), 1033–9.