Institut für Computergraphik und Algorithmen

Technische Universität Wien

Karlsplatz 13/186/2
A-1040 Wien
AUSTRIA

Tel: +43 (1) 58801-18601
Fax: +43 (1) 58801-18698

Institute of Computer Graphics and Algorithms

Vienna University of Technology

email:
technical-report@cg.tuwien.ac.at

other services:
http://www.cg.tuwien.ac.at/
ftp://ftp.cg.tuwien.ac.at/

TECHNICAL REPORT

Illustrative rendering of seismic data

Daniel Patel, Christopher Giertsen, John Thurmond, Meister Eduard Gröller

TR-186-2-07-08
May 2007

Illustrative rendering of seismic data

Daniel Patel[*], Christopher Giertsen[†], John Thurmond[‡], Meister Eduard Gröller[§]

May 15, 2007

Abstract

We present multi attribute texture transfer functions for the generation of seismic illustrations. We render seismic data in the style of geological textbook illustrations by combining illustratively rendered axis aligned slices with volume rendering. We have extended the transfer function concept to map volume attributes to 2D textures that flow according to a deformation volume describing the buckling and discontinuites of the layers of the seismic data. Faults in the seismic layers are represented by texture disruptions.

1 Introduction

Illustrations are being used when there are certain aspects of a complex image that needs to be communicated in a simple, condensed and pedagogical way. Making a good illustration for scientific purposes is very time consuming. An interactive system capable of rendering illustrations is valuable both for quickly creating them and for interactive data exploration. Interactive illustrative rendering is particularly well suited when there is underlying information of how the data can be simplified to emphasize the elements of interest. In this paper we address the unexplored domain of illustrative rendering of geological data. We present algorithms that mimic techniques seen in geological illustrations and we apply these techniques on seismic data. Being able to render geological illustrations is advantageous both for illustrating geological books and in the domain of oil exploration when interpreted survey data needs to be communicated as part of decision making.

[*]e-mail: daniel@cmr.no affiliation:Christian Michelsen Research, Bergen, Norway

[†]e-mail: chrisgie@cmr.no affiliation:Christian Michelsen Research, Bergen, Norway

[‡]e-mail: john.thurmond@hydro.com affiliation:Norsk Hydro, Bergen, Norway

[§]e-mail: groeller@cg.tuwien.ac.at affiliation:Institute of Computer Graphics and Algorithms, Vienna University of Technology

Fault and horizon structures are central in defining the visualization of geological illustrations. The horizons are the surfaces that separate one layer from another in the layer-like structure of the earth. A geologic layer is delimited by two horizons, a top horizon and a bottom horizon. Tension in the crust of the earth deforms the layers over time and creates cracks. These cracks, called faults, are more or less vertical discontinuities of the layers. When faults divide an area in several distinct pieces, the pieces are called fault blocks. The faults, horizons, and other subsurface structures are discovered by geoscientists interpreting volumetric descriptions of the subsurface. These volumetric descriptions are typically obtained in geophysical surveys by processing the reflections of waves sent into the surface. The volume storing the reflection data is often referred to as the reflection volume. It is from the reflection volume the faults and horizons are interpreted out. Several other attributes are measured during the survey. Acoustic impedance (Ai) and cross wave/shear wave ratio (Vp/Vs) are other examples. Top half of Figure 2 gives an overview of the data collection process just described.

Geological data have certain typical characteristics, one being the layered structure. This structure is emphasized in geological visualizations by using textures flowing with the layers and with discontinuities over faults. The textures are drawn on the sides of a cube which defines the geological area. The earth ground is typically illustrated with trees, mountains and water. Axis-aligned cut outs are used, sometimes with extruding features in the cut out. See Figure 1 for examples. As opposed to medical data, geological data is dense with no or few cavities and with few familiar structures, this could be the reason illustrations rarely contain semitransparent regions.

Our method renders textures on arbitrary axis-aligned cut planes for a selected area. The textures are deformed to reflect the layer deformations and exhibit discontinuities across faults. For each layer one can assign a texture and the texture's horizontal and vertical frequency. In addition one can choose a volume data set for an attribute that exists for the seismic data, such as the re-

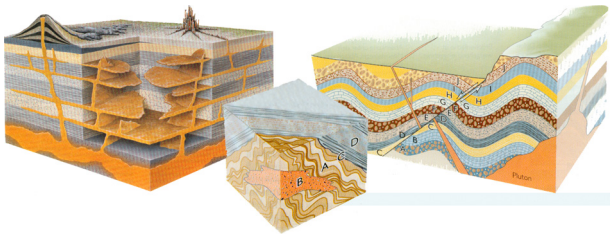

Figure 1: Geological illustrations. Left: A cut out with extruding features. Middle: Texture flow can be seen on all surfaces of the cube. Right: Textured layers with a fault discontinuity in the middle. Pictures are taken from [1]

flectance volume and define a texture transfer function. This texture transfer function assigns different textures with opacities to attribute intervals. These textures are composited with the layer textures. We represent the cut outs by volume rendering inside a movable box with a basic $RGB\alpha$ transfer function. For transparent cut outs a transparent transfer function is used. Furthermore the transparency for each layer in the volume rendered cut out can be set to restrict volume rendering to certain layers or to certain depths within a layer. We render the ground surface opaquely using a color from the texture of the layer just below the ground surface. This gives a consistent coloring of the ground surface according to the textured layers.

Alternatively the ground surface color and the cut-plane textures can be rendered according to the $RGB\alpha$ transfer function used for the volume rendering. This would represent the raw, uninterpreted data. The user can smoothly change between the raw data view and the abstract textured-view with a slider.

We have structured the paper according to the characteristics seen in geological visualization as noted above. After related work we describe the calculation of the texture flow in section 3.1, in section 4 we use the calculated flow in combination with texture transfer functions to texture the cut planes. In section 5 we describe volume rendering for simultaneously displaying a cut out and the surrounding region. We specify how this is integrated with the rendering of textures during raycasting. In section 6 there is a short discussion on illustrative rendering in oil exploration. We end with future work and conclusions in section 7. The bottom half of Figure 2 shows a high level overview of the paper. The whole figure presents how the data flows from acqusition to visualization.

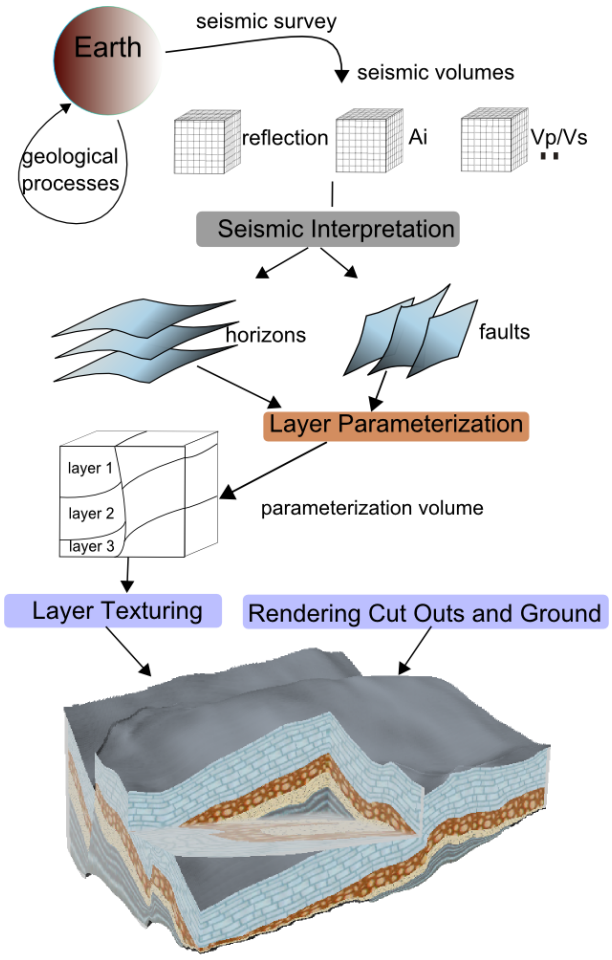

Figure 2: Overview of the process from data collection to visualization. The paper covers the lower three colored rectangles in section 3, 4 and 5.

2 Related work

Illustrative rendering is a term coined some years ago and belongs to the group of non-photorealistic rendering techniques tailored to capture the advantages of conveying information through illustations. We first review work dealing with illustrative techniques and then review work in the field of seismic visualization.

In recent years several illustrative rendering techniques, mainly in the domain of anatomical visualization, have been proposed. Some of these techniques deal with applying textures from reference images as in our work. Whereas texture synthesis [2, 3, 7] is often used in these works, we use a straightforward 2D tileable texture approach. Perhaps the approach that is in this respect most similar to ours is the work by Owada et al. [14] where an interactive system for texturizing arbitrary cuts through

polygonial objects is presented. The user defines the texture flow by specifying a flow field and a distance field on the cut. This information is used in the texture synthesis to create a texture on the cut that follows the flow. Their work does not deal with the integration of textures with 3D volume data visualization as ours does. Their method is general and therefore requires user interaction to specify the deformation and the texture. We calculate a parameterization up front so the texturizing process can be performed quickly and without the need for texture synthesis. Many of the parameters defining the visualization are known prior to rendering, therefore less user specification is required in our approach.

There are also several papers dealing with textures in medical volume rendering. Lu and Ebert [11] generate illustrative renderings of color and scalar 3D volumes by applying textures sampled from illustrations and photographs. 3D textures are created by combining color information from the illustrations with 3D volume data of a corresponding area. Finally the 3D textures are made tileable with Wang Cubes. With segmented volume data they apply the corresponding 3D textures on each segment. With unsegmented scalar data they use a transfer function to map scalar voxel values to the 3D textures in a similar way to what we propose. They do not however deal with multi-attribute texture transfer functions and with deforming the textures to follow the underlying flow of the data. In addition their method of calculating the textures is tailored to handle 3D textures whereas we use 2D textures.

Dong and Clapworthy [6] present a technique that performs 3D texture synthesis following the texture orientation of 3D muscle data. Their algorithm has two steps. First they determine the texture orientation by looking at the gradient data of the volume and by using a direction limited Hough transform. Second they perform a volumetric texture synthesis based on the orientation data. In our work, instead of considering the volume for evaluating texture flow, we consider the geometric layers. In addition the texture synthesis of Dong and Clapworthy has the drawback of not working on textures with large interior variation as textures in geologic illustrations commonly have.

Wang and Mueller [18] use 3D texture synthesis to achieve sub resolution zooming to volumetric data. By having 2D images of several zoom levels of a tissue, they synthesize 3D volume textures for each level and use constrained texture synthesis to blend smoothly between the levels when zooming. They address the issue of sub-resolution details as we briefly do, but do not consider texture flow.

In the domain of seismic processing GeoChron [12, 13] is a formal model for parameterizing the layers defined by faults and horizons. Our parameterization considers only the fault and the horizon data. The GeoChron model allows for additional inputs which act as constraints to the parameterization such that it considers the physical processes behind the deformation. Since our visualization algorithm is decoupled from the parameterization, it would also accept a GeoChron parameterization. We believe that for illustration purposes only, a less accurate but also computationally less intensive algorithm requiring a minimal amount of input and expertise is preferrable. In the domain of oil exploration, it would be an advantage to use a more precise parameterization such as the GeoChron model.

Cut outs on seismic data and interaction in VR was presented in the work by Ropinski et al. [16] where they use volume rendering with two transfer functions. One for the volume inside a user defined box and one for the volume outside. We use a similar method to implement our cut out technique.

We are not aware of any work on illustrative techniques for seismic data. Work has been done on cartographical illustrations [17, 4] but we have found none dealing with volumetric data.

Several papers on visualizing seismic data exist, some deal with automatic horizon extraction [5] or fault extraction [9, 5], others deal with handling large volumes [15, 5], but none deal with illustrative rendering. Somewhat related is the PhD of Frank [8] where the GeoChron parameterization [12, 13] is used as a lookup to unfault and flatten an attribute volume. They present a slice visualization that smoothly blends from grayscale raw data to showing the color coded distance to the closest faults and from grayscale raw data to a visualization showing each layer with a unique color. Apart from this they do not apply any other illustrative techniques.

The major contributions of this paper are:

- Introducing illustrative rendering of geological data
- Using multi-attribute texture transfer functions with underlying flow information
- Combining 2D textured cut outs with volume rendering.

3 Layer parameterization

In this section we will describe how layers are parameterized and how this parameterization is used to look up

a 2D texture for texturing cut planes. We first present our model for parameterizing the layers. We then talk about how faults, being discontinuities in the layers, are handled followed by how we store segmentation information in the parameterization describing each layer. Finally we discuss a problem that arises when accessing the parameterization volume using linear interpolation.

3.1 Layers

We calculate the texture deformations in the layers by precalculating a 3D parameterization of the layers which is later used as a lookup into the texture. The result is stored in a 3D 3-component volume. Our model for calculating the texture deformation of a layer is based on the assumption that the original and undeformed 3D textures describing the layer appearance are defined in a space where both the horizons of a layer are parallel to the xy-plane and connected over faults, i.e. in a space where horizons are flattened and faults are aligned, see Figure 3. We call this space the flat space F and we call the space over a deformed layer the curved space C. For 3D texturing of a deformed layer, the color of a point p is found by finding the corresponding point in the flattened layer and sampling the texture there. We therefore need to define a function mapping from the curved space C to the flat space F. This mapping captures the deformation information and constitutes the parameterization of a layer. Calculating the texture deformation for a layer is the problem of parameterizing an unclosed area between two surfaces. There is no one single parameterization of this area, the problem is underdetermined. We need to define some constraints of the parameterization. The first constraint we define is that the curved horizons in C are mapped to the flat horizons in F. Secondly we wish that the surface in the middle of the layer in C is mapped to a surface in the middle of the layer in F. The intuitive notion of a surface being in the middle of two surfaces can be formulated more precisely as a surface whose points each have the same distance to the closest point in the surface above it as to the closest point in the surface below. We also want the parameterization to be conformal or angle preserving so that the angles of a texture such as a brick texture are preserved through the mapping. These four requirements are represented by the four mappings with red arrows in Figure 3. The vertical line going from the top horizon to the bottom horizon represents the angle preserving requirement and is normal on the horizons and the middle surface both in C and in F.

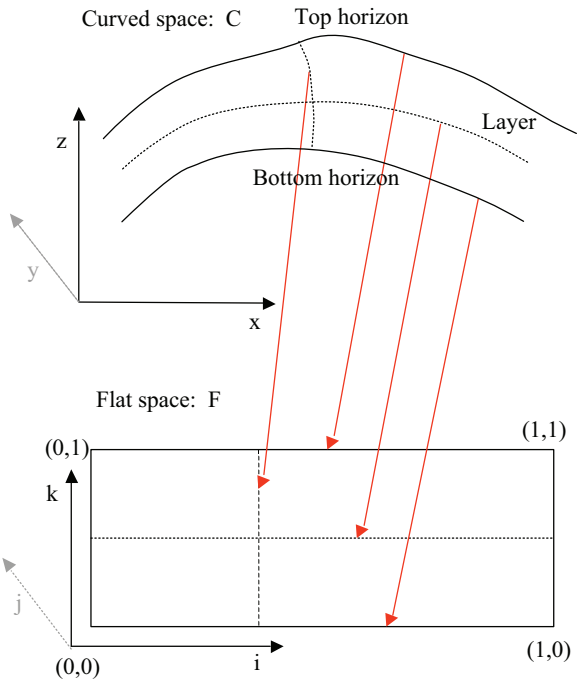

Figure 3: Layer Parameterization. Mapping from curved space C to flat space F. The mapping maps the top horizon to k=1, the bottom horizon to k=0, the middle of the layer to k=0.5. A curved line normal to the horizons, normal to the middle line, and everywhere parallel to ∇f_z is mapped to a vertical line.

Let $p_{curved} = \{x, y, z\}$ be a point in C, and $p_{flat} = \{i, j, k\}$ be the corresponding point in F, we represent the mapping from C to F as $f : C \rightarrow F$:

$$p_{flat} = f(p_{curved}) = \{f_x(p_{curved}), f_y(p_{curved}), f_z(p_{curved})\}$$

For simplicity we assume that both the domain and codomain of f are in the unit cube.

We calculate the mapping f in two steps. We first determine f_z and use f_z to calculate f_x and f_y based on the requirement of angle preservation. We start with f_z since the horizons define two clear correspondences between the flat space and curved space for z values. We let the layer be parameterized in a unit cube such that:

1. for any point p_{top} in the top horizon $f_z(p_{top}) = 1$
2. for any point p_{bottom} in the bottom horizon $f_z(p_{bottom}) = 0$
3. f_z is continuous and changes according to relative euclidean distance between layers such that points in the middle of the layer in C are mapped to the middle in F where k = 0.5.

To find f_z for a point p_{curved} we calculate the point's distance d_{top} to the closest point on the horizon above, and the distance d_{bottom} to the closest point on the horizon below. We let

$$f_z(p_{curved}) = \frac{d_{bottom}}{(d_{bottom} + d_{top})}$$

This expression satisifies the three critieria made above. The closest point calculation is done by discretizing each horizon into a sufficiently dense point cloud and storing the points in a kd-tree for efficient searching. We now have an expression for f_z.

We observe that the lines in F parallel to the k axis, hereby called isolines, each have the same i and j parameter. They are naturally normal on the planes k=c where c is a constant. We call isocurves the curves in C that are mapped to the isolines in F . If we can identify an isocurve in C and the i and j parameter for one point on the curve, we know that all points on the whole curve have the same i and j parameter. We have already found f_z so the k parameters for the points in the curve can be calculated. So if we identify the isocurves in C and the i and j parameter for any point in the curve we have a parameterization of the whole curve. An isocurve can be traced from a point by moving in the normal direction of f_z. Therefore the i and j parameters for a point p_{curved} are found by starting a trace at p and tracing towards the middle in the direction ∇f_z, i.e. normal on isosurfaces of f_z. When the middle of the layer is reached the location of point p in the parameterization volume is assigned an i and j value equal to the x,y coordinate of the middle surface point in C. The three parameters for each point are stored in a three component volume covering the area which contains the horizons and faults.

3.2 Faults

The faults are described as geometries cutting across the horizons where discontinuities in the geological data is found. A fault crossing a layer is dividing the layer in two. This subdivision means we need to perform a separate parameterization for each side of the layers and we need special handling when parameterizing the side borders of the layers. One can observe in Figure 4 that tracing along the ∇f_z gradient to calculate f_x and f_y can result in moving out of the layer where ∇f_z is not 'well defined' due to a lacking neighborhood of points to calculate a distance to. To avoid this we need to extend the layer by linearly extending the horizon above and below it slightly further and calculate f_z in the extended area. Now ∇f_z is well defined here too and a tracing to the middle to get i and j can be performed.

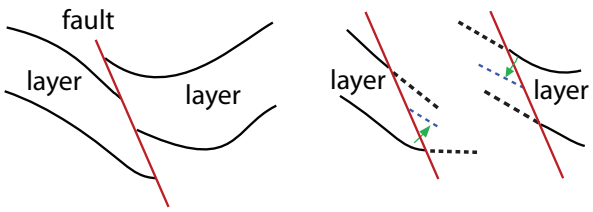

Figure 4: Fault handling. Top: A layer divided by a fault. Bottom: The border ends of the divided layer. Tracing (green arrow) may result in moving out of the layer. Horizons must be extended (dashed black line) so f_z can be calculated outside the ends and tracing can be performed to the middle (dashed blue) to get values for i and j.

3.3 Adding segmentation information in the parameterization

The parameterization volume now contains all layers stacked on top of each so that for each layer the k values go from 0 at its bottom horizon to 1 at its top horizon. Similarly all layers have their own i and j parameterization in the interval $[0,1]$. We perform scaling and shifting of the parameterization values to embed layer segmentation information in the parameterization so that each layer can be assigned an unique texture. Let $NumLayers$ be the number of layers and L be a layer id in the interval $[1, NumLayers]$. To be able to identify which layer a parameterization lookup value comes from we recompute the k values in a layer L from $[0,1]$ to $[\frac{L}{NumLayers}, \frac{L+1}{NumLayers}]$ i.e. $k_{new} = \frac{k+L}{NumLayers}$. For instance if $NumLayers$=4 the lowest layer l_1 would have k values in $[0, \frac{1}{4}]$, l_2 in $[\frac{1}{4}, \frac{1}{2}]$, l_3 in $[\frac{1}{2}, \frac{3}{4}]$ and l_4 in $[\frac{3}{4}, 1]$. This procedure embeds the layer id at the expense of reduced numerical accuracy. We use the same approach to give id's to the fault blocks which are the pieces the faults divide the volume into independent of the horizons. Here we shift and scale the k values in the parameterization so each of the regions have their k values in a unique interval. These new values are the actual values that are stored in the parameterization volume (see Figure 5).To find a unique layer id at a position $p_{curved} = \{x,y,z\}$, we look up in the parameterization volume to find $p_{flat} = \{i,j,k\}$. The id of the layer is calculated by the function $layerId(i,j,k) = \lfloor k * NumLayers \rfloor$. Similarly the id of a fault block is $faultBlockId(i,j,k) = \lfloor j * NumFaultsBlocks \rfloor$

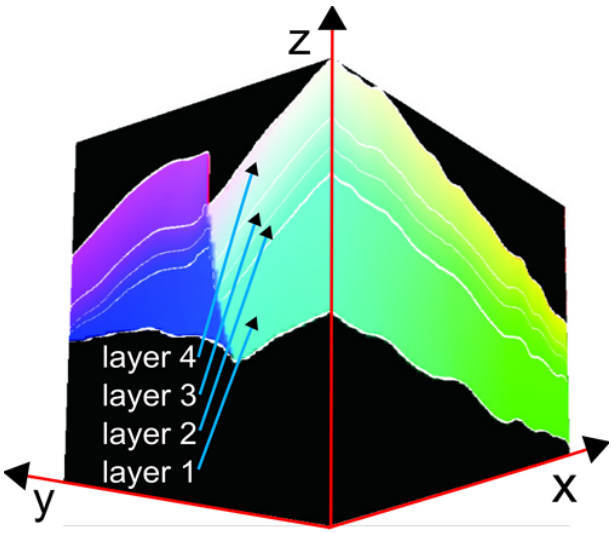

Figure 5: Parameterization volume. The ijk parameterization is stored in an *RGB* volume. The red component represents the k value, green represents j and blue represents i. White lines have been added to the figure and correspond to k=0, 0.25, 0.5, 0.75 and 1 to show the horizons. There is a clear color change across the fault, because the green component storing the j value has been shifted and scaled for each side of the fault as discussed in 3.3

3.4 Interpolation problem around horizons and faults

The parameterization volume is a discrete definition of our parameterization function. To achieve a smoother function we sample the volume trilinearly. However this leads to invalid interpolation in cells on layer boundaries where the 8 cell corners are in different layers or fault-blocks. When we look up in the volume we calculate the layer id of the lookup position and the layer id of the 8 corners of the cell and we label the corners as invalid when they have a different id than the look up position. We calculate new values for the invalid corners by extrapolating from valid neighbor values and then perform the trilinear interpolation for the new corner values on the GPU. The extrapolation will try to assign a new value for invalid corners by considering the corner's two neigbors in positive x direction. If both are valid then their values are linearly extrapolated and assigned to the corner. If not, then the search continues in negative x direction and then similarly in y and z direction. When this procedure fails to extrapolate all the invalid corner values an erroneus interpolation is performed. This happens rarely in practice and the artifacts will be noticeably only at the layer borders and will be of the same size as a voxel in the parameterization volume. We realize that we are not solving the interpolation problem in the best possible way. It does however improve the quality of the renderings as can be seen in Figure 6.

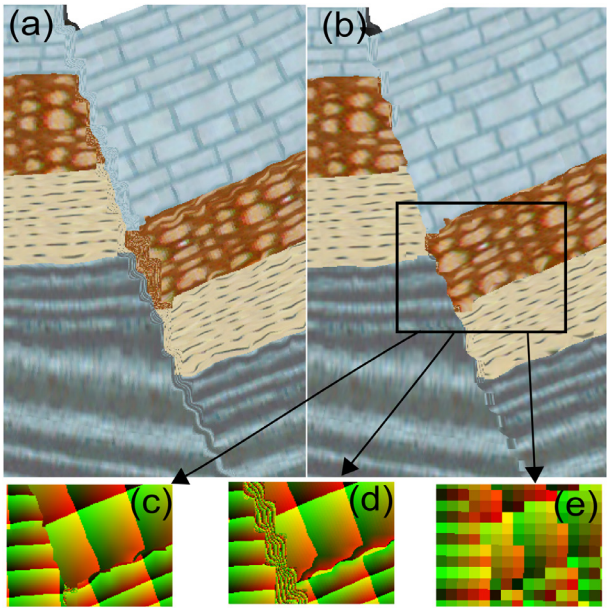

Figure 6: A fault and interpolation problems. In (a) linear interpolation is used. In (b) extrapolation as described in 3.4 improves the quality. In (c) we see the parameterization of the square with extrapolation as opposed to without in (d). In (e) we see the parameterization with nearest neighbor interpolation revealing the resolution of the parameterization volume in the square area.

3.5 2D Texture mapping on axis-aligned cut planes

Our parameterization volume now enables us to apply an undeformed 3D texture stored in flat space and deform it into curved space and texture voxels in our layers. This would require us to generate a 3D texture which is a research topic in its own [11]. Since we limit ourselves to texturizing axis-aligned cut planes as done in geological illustrations we can reduce the problem to a 2D texturing problem. This has several advantages. 2D tileable textures are easy to generate, take little space and can be sampled from illustrations directly. However we need to define a transformation from a 3D sample in flat space to a 2D sample for our 2D textures. We can observe in geological illustrations that there are three types of cut planes according to which axis their normals is parallel to. For cut planes on sides of a cube

we map the parameter k to the height of the 2D texture and either i or j is mapped to the length of the texture depending on the normal of the cut plane. To get a connected texture on cut planes at the top or bottom of a cube we also here map k to the height of the 2D texture and we map i to the length.

4 Layer texturing

This chapter presents three transfer functions that can be used together to visualize data on cut planes in the seismic volume. First we present a layer texture transfer function (abbreviated as layer TTF) that controls the textures for each layer, then we present the scalar texture transfer function (abbreviated as scalar TTF) which maps textures and opacities to intervals of a scalar volume from the seismic data acqusition. We combine the texturing results for these two transfer functions and finally present the concept of smoothly moving from illustrative rendering of cut planes to color rendering with an ordinary color transfer function. See Figure 7 for an overview.

Textures can give additional information of the compression and orientation of a layer on a local scale. Much can be read from a small texture patch, one knows which layer one is looking at by recognizing the type of the texture. The texture also conveys information about the orientation of the layer in terms of the angles in the texture. Finally, the horizontal repeats of a texture in a layer is constant throughout the layer, therefore it would be high where the layer is thin, and lower where the layer is thick which means that information about the relative thickness of the layer also can be seen from the texture. Reading all this information from the texture alone can be useful for instance in the case when the user has zoomed in so much that no horizons are visible anymore. Even when horizons are visible the texture reinforces and repeats the layer, orientation and compression information. Textures are also well suited when zooming to a subresolution scale. Rendering the raw data would in this case give low frequent, low information and aestetically displeasing results. Rendering a texture instead has the advantages as mentioned above in addition to being aestetically pleasing, the last point being important in illustrative rendering.

4.1 Layer texture transfer function

To texture the cut planes we use 2D textures sampled from geological illustrations and map them on the layers. We do this by using a layer texture transfer function (layer TTF) that maps from a scalar to a triplet, the

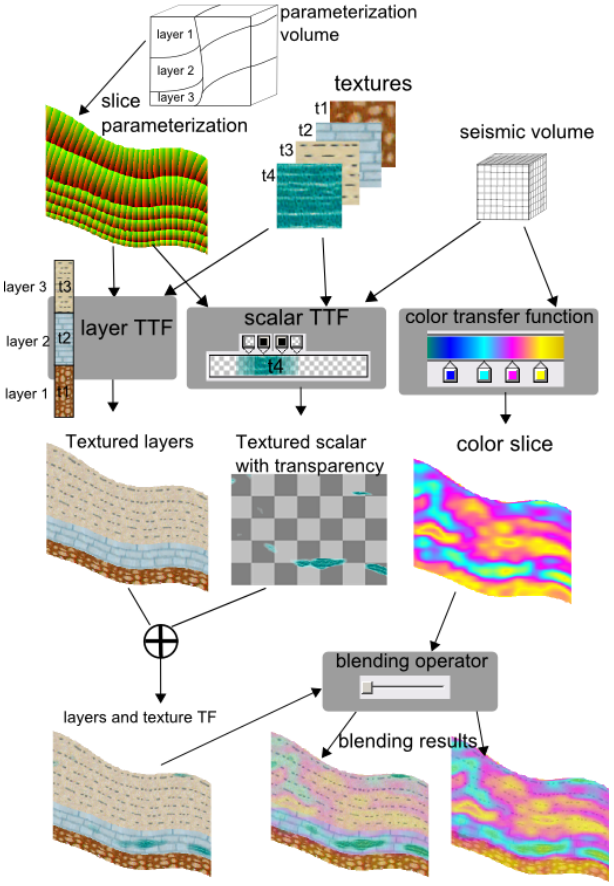

Figure 7: Overview of how textures are combined. Cut plane parameterization, layer TTF, scalar TTF and color transfer function are explained in 3.5, 4.1, 4.2 and 4.3 respectively.

triplet contains the texture id and a horizontal and vertical texture frequency. For a point in the volume with parameter value $\{i, j, k\}$ we use k as the scalar input to the layer TTF. Since k varies in distinct intervals for each layer we can map each interval in the transfer function to a unique texture id and two frequency components. We can even apply different textures to different depths within the same layer, depths being defined in flat space. One could for instance apply one texture on the top half of a layer and another on the bottom half.

4.2 Scalar texture transfer function

The visualization so far only considers the parameterization volume defined by the layers and faults. We will now also consider volumetric scalar data obtained in the seismic survey such as the reflectance data. We will use a scalar texture transfer function (scalar TTF) mapping

from a scalar value in the volume to a layer id, horizontal frequency, vertical frequency, and an opacity. One can then map intervals of scalar values from the volumetric data to textures. These textures will then be drawn on top of the layer textures with the same deformation as the layers. The TTF opacities decide how the texture should be blended with the underlying layer texture. If opacity is 0 only the underlying layer texture will be seen. If it is 0.5 both will be blended, and if it is 1 the underlying texture will be overdrawn. This blending operation is represented as the circle with plus in Figure 7. As default the horizontal and vertical frequency of a scalar texture is set to the frequency of the layer it is drawn on top of, but the user can add multiplicative factors for horizontal and vertical frequencies in the scalar TTF to change this. The reasoning behind using the layer frequencies for the scalar texture is that if two neighboring layers have different thickness, for instance one being twice as thick as the other, then the vertical frequency of a texture will be twice as large on the thin layer as the thick. The user can normalize this by changing the vertical frequency for the thin layer to half of what it is for the thick layer. Now textures will have the same vertical frequency for both layers and a scalar texture crossing from the one layer to the other will have consistent frequency since it uses the frequency information in the layer it is on. See Figure 8 for an example.

The assignment of frequencies for the textures is highly dependent on the degree of zoom, or the distance the perceiver has to the texture. When zooming out after the assignment the textures will be perceived as to high and when zooming in they will be perceived as being too low. For this reason we multiply all the frequencies with a scalar which is editable using a slider. The user can use the slider to adjust the global frequency of all textures after zooming.

4.3 From texturing to color coding of cut planes

For data browsing purposes it is useful to inspect the raw seismic data directly. For this we apply an ordinary *RGB* transfer function on the scalar values of the attribute volume. In addition we introduce the concept of continuous transition from illustrative rendering to raw data rendering by smoothly blending from visualizing textured cut planes to visualizing cut planes colored by the *RGB* transfer function. This can be controlled by a slider. This not only gives a pleasant transition from one mode to the other but also introduces an intermediate rendering mode between illustrative and raw rendering.

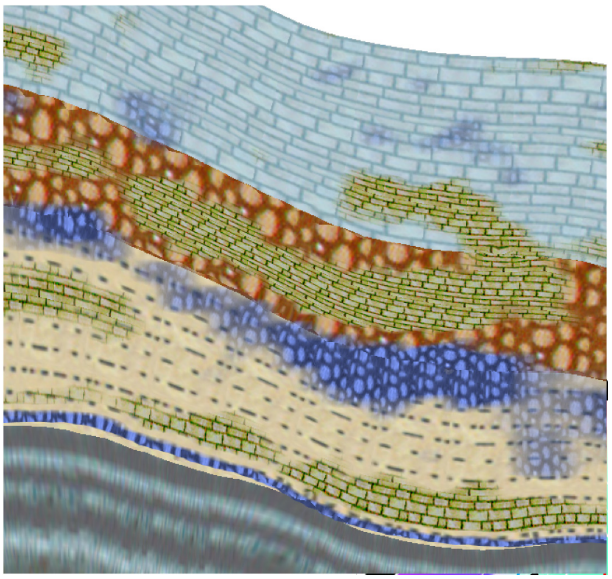

Figure 8: Combination of layer TTF and scalar TTF. There are four textures for the four layers and two textures representing the scalar data. The brown brick texture shows areas of high scalar values, the violet texture shows areas of low scalar values. The scalar values are from the reflectance volume.

This intermediate mode allows for both seeing the underlying data values in terms of the color coding and at the same time seeing the illustrative textures. The balance between the two data sources can be adjusted to get what the user perceives as an optimal balance between the rendering techniques. See Figure 9 for an example.

5 Rendering cut outs and ground surface

We are realizing cut outs as a focus + context visualization with one transfer function for the cut out as focus object, and one for the surrounding. For both areas we perform volume rendering using $RGB\alpha$ transfer functions. The transfer function for the cut out is the same as is being used in section 4.3 for raw data rendering except that now we also consider the α channel. For visualizing the ground surface outside the cut out the transfer function is set to opaque so the ray stops at the first hit of the ground surface. For consistent coloring of the ground surface with respect to the texture pattern just below ground we set the ground surface color to the average color of the texture. This average color is precalculated for each texture. Although we terminate the ray at the ground surface we will in the rest of the paper

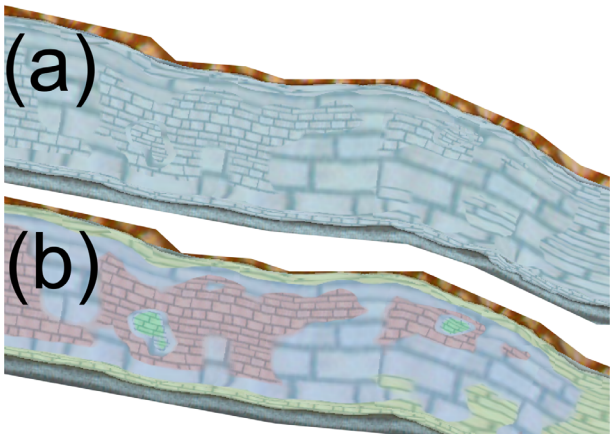

Figure 9: Layer TTF, scalar TTF and color combined. Instead of using different textures on intervals of the scalar values we use the same texture but with different horizontal and vertical frequencies. It is difficult to discern the textures in (a). In (b) we blend in colors from the *RGB* transfer function and can see that three textures are laid on top of the layer texture in the green, red and yellow area.

assume a general transfer function for the surrounding which does not necessarily terminate at the ground surface intersection. If we were to limit ourselves to rendering the ground surface opaque the rendering could have been accelerated by performing geometry visualization of the ground surface instead of raytracing to the first hit, we chose raycasting for flexibility reasons. With ray tracing we can reconstruct any isosurface in the scalar volume or in the parameterization volume without having the explicit geometric definition. In the case of volume rendering for ground surface visualization the performance hit is not too big since we use empty space skipping.

5.1 Combining Volume rendering with textures

From the large seismic data that fits in the memory of the graphics card the user can decide which portion to be visualized by defining the position and extent of a box. Only the data within this box, commonly called a roaming box, is visualized. Within the area of the roaming box another box can be defined which cuts into the roaming box. This box defines the area of the cut out. The roaming box and the cut out box and their intersections with the parameterization volume defines where to perform raycasting, where to perform texturing and where to do both. See Figure 10 for a 2D schematization of the roaming box, the cut out and their relation with the volume.

We perform raycasting as suggested by Krüger and Westermann [10] with empty space skipping. We have subdivided the volume into equally sized blocks and created an octree data structure containing the blocks. Based on an octree traversal and geometric rendering of the nonempty blocks into two color buffers, one describing the rays entry point and the other describing the rays exit point, we implement empty space skipping. We perform the volume rendering and the texturing in a fragment shader in a single pass. For each fragment we find the start, the end and the direction of the ray by looking in the entry and exit buffer. In addition the shader is given the cut out box in terms of the two corners defining it. Then intersections with the cut out box are calculated. If there are intersections then the ray will be divided into three intervals, before the cut out, in the cut out and after the cut out. Before and after the cut out is rendered with the transfer function for the surroundings. In the cut out the transfer function including alpha channel for the color coded cut plane visualization is used. We also make use of the opacity channel of the layer TTF, we multiply the opacity from the $RGB\alpha$ transfer function used in the volume rendering of the cut out with the opacity found at the layer TTF at parameter k for the ray. This enables us to choose the transparency for the different depths in the layers or only do volume rendering on selected layers by making the others transparent. By only performing volume rendering on selected layers we can easily achieve the effect seen in geological illustrations of extruding objects in the cut outs. For exploration of the seismic data this is useful when one wants to consider only one layer at a time. For instance the oil reserves are typically trapped between horizons in so called reservoirs. If one wants to perform volume rendering to explore such a reservoir it would be natural to confine the volume rendering to the layer the reservoir is in. See Figure 11 for examples of volume rendering in a cut out limited to a layer.

Textures are applied at places where either the front faces of the roaming box or the back faces of the cut out box intersect the volume (seen as black thick lines in Figure 10). When the ray is at an exit point of the cut out inside the volume then the texture value at this point is calculated and composited with the accumulated ray color and the ray is terminated. In any case a ray is terminated if its opacity is close to 1. See Figure 11 and 12 for examples.

With a Geforce 8800 GTX graphics card and a image size of 800x800 we have 5 frames per second. Without the extrapolation as described in 3.4 the framerate is

doubled. The 3 component parameterization volume is of size 128x128x128 and the reflectance volume of size 240x271x500. The Ai volume is of size 96x96x500 and covers a smaller area than the reflectance volume.

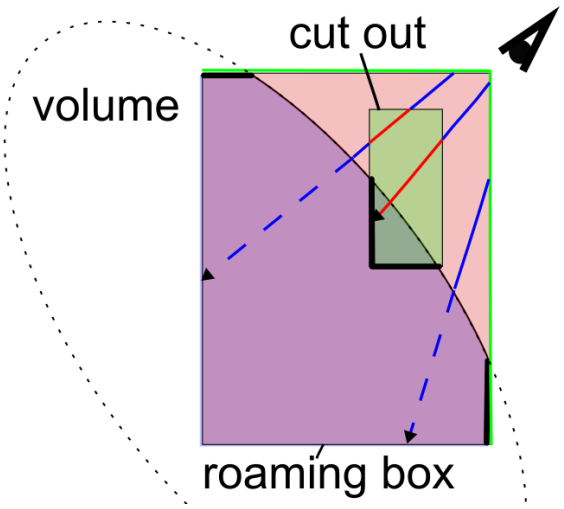

Figure 10: Combining volume rendering with texture rendering. The green line depicts the entry buffer. The thick black lines show where texturing is performed. For explanatory reasons we do not draw the green line according to empty space skipping here.

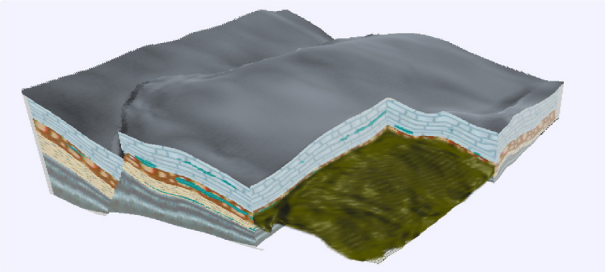

Figure 11: Combined texture and volume rendering with extruding layer. Volume rendering is performed only for layer 3 with brown color to mimic a geological illustration. The layer discontinuity is due to a fault. Turqoise patches show areas with high reflection values.

5.2 Illustration rendering to raw data transition

The slider mentioned in 4.3 for blending from illustrative to raw data visualization is also used to blend from illustrative rendering of the ground surface with a color according to the texture below the surface to a raw data visualization of the ground surface according to scalar values of the attribute volume colored by the *RGB* transfer function. See Figure 12 for a blending time series.

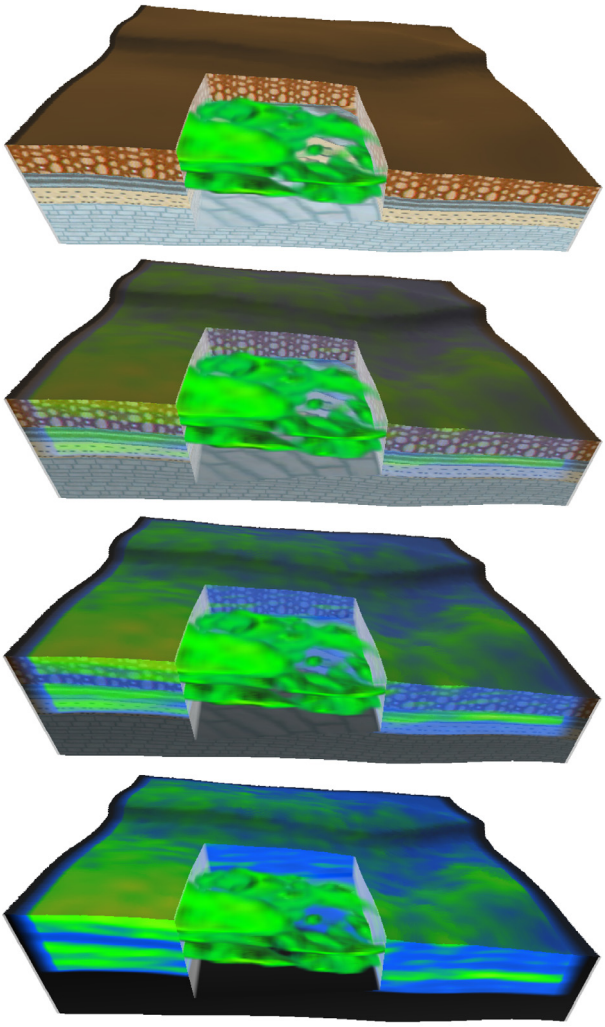

Figure 12: Blending from illustrative rendering to raw data rendering of the Ai attribute. To the right on the images with blended rendering one can see how the green area having high Ai values correspond to layer 3 counted from the bottom. The black areas are areas where there is no Ai data.

6 Illustrative rendering in oil exploration

We have presented the illustrative rendering techniques developed in this paper to people using volume render-

ing software for oil exploration. Their reactions were positive and the feedback was that these technique can be a powerful way for communication of interpreted data to other experts in other areas within an oil company. Usually this is done now by showing the seismic data such as the reflectance volume directly which can be noisy and difficult to understand for non experts. For concepts that are difficult to convey using the raw seismic data they make their own illustrations by hand drawings or using drawing software. For higher quality illustrations used in reports or publications they sketch out what they want and let the drafting department create a better illustration. They found interest in the scalar TTF and pointed out that this can be a good way of seeing how individual attributes relate to layers, i.e. if intervals of the attribute are confined within certain layers or change significantly (or subtly) between layers. In Figure 12 the Ai attribute is visualized, this attribute has as one can see a high correspondance with layers. They liked the way our texture transfer functions visualize layer data and attribute data in a unified way through textures, this is an advantage as compared to the standard approach in seismic visualization software where attribute data is presented with volume rendering and horizons and faults are presented as geometric surfaces. They pointed out that our method only works on data that has already been interpreted. Being able to do illustrative rendering on uninterpreted data would be very interesting for them. They also noted that texture flow based on the horizons is good for educational visualizations but care has to be taken not to mis-interpret the textural flow with the actual arrangement of the sediments. The sediments might have a high correspondence at the horizons, but not necessarily between the horizons.

7 Conclusions and Future work

We have presented a technique for illustrative rendering of geological data through various texture transfer functions. This technique can be the start of a new direction of visualization research targeting the oil&gas sector. On a longer term, we believe that the proposed technique, and subsequent refinements and extensions, may have impact in two important areas. Firstly, illustrative techniques giving good overviews can make it faster and easier to evaluate large prospects. This is a mission critical task in any oil company, and thus of great importance for oil-producing nations. Secondly, illustrative techniques can be used to improve communication between different stakeholders, not only within the oil&gas industry, but also towards media, public sector and politicians. This is definitely a great challenge with current methodology. The texture transfer function concept was well regarded of when we showed it to experts from an oil company. Although we employ some heuristics in our approach, we believe the renderings do not suffer too much from this. In the future we will find another way of solving the extrapolation problem and we will look into methods making it possible to do illustrative rendering on uninterpreted data.

References

[1] *Understanding Earth*. W. H. Freeman and Company, New York, NY, USA, 1994.

[2] M. Ashikhmin. Synthesizing natural textures. In *SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 217–226, New York, NY, USA, 2001. ACM Press.

[3] M. Ashikhmin. Example-based volume illustrations. *Computer Graphics and Applications, IEEE*, 23:38– 43, 2003.

[4] K. Buchin, M. C. Sousa, J. Dollner, F. Samavati, and M. Walther. Illustrating terrains using direction of slope and lighting. In *Proc. of the 4th ICA Mountain Cartography Workshop,*, pages 259–269, Vall de Nuria, Catalunya, 2004. Institut Cartographic de Catalunya.

[5] L. Castanie, B. Levy, and F. Bosquet. Volumeexplorer: roaming large volumes to couple visualization and data processing for oil and gas exploration. *Visualization, 2005. VIS 05. IEEE*, pages 247–254, 2005.

[6] F. Dong and G. Clapworthy. Volumetric texture synthesis for non-photorealistic volume rendering of medical data. *The Visual Computer*, 21(7):463–473, 2005.

[7] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346, New York, NY, USA, 2001. ACM Press.

[8] T. Frank. *Advanced Visualisation and Modeling of Tetrahedral Meshes*. PhD in geosciences, Institut National Polytechnique de Lorraine, F-54501 Vandoeuvre, France, April 2006.

[9] W.-K. Jeong, R. Whitaker, and M. Dobin. Interactive 3D seismic fault detection on the Graphics Hardware. In *Volume Graphics 2001*, pages 111–118, 2006.

[10] J. Krüger and R. Westermann. Acceleration techniques for gpu-based volume rendering. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 38, Washington, DC, USA, 2003. IEEE Computer Society.

[11] A. Lu and D. S. Ebert. Example-based volume illustrations. In *VIS '05: Proceedings of the conference on Visualization '05*, page 83, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[12] J.-L. Mallet. Space-time mathematical framework for sedimentary geology. *Mathematical Geology*, 36:1–32, 2004.

[13] R. Moyen. *Parametrisation 3d de lespace en geologie sedimentaire: le modele geochron*. PhD in geosciences, Institut National Polytechnique de Lorraine, F-54501 Vandoeuvre, France, june 2005.

[14] S. Owada, F. Nielsen, M. Okabe, and T. Igarashi. Volumetric illustration: designing 3d models with internal textures. In *SIGGRAPH '04: Proceedings of the 31th annual conference on Computer graphics and interactive techniques*, pages 322–328, New York, NY, USA, 2004. ACM Press.

[15] J. Plate, M. Tirtasana, R. Carmona, and B. Fröhlich. Octreemizer: a hierarchical approach for interactive roaming through very large volumes. In *VISSYM '02: Proceedings of the symposium on Data Visualisation 2002*, pages 53–ff, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[16] T. Ropinski, F. Steinicke, and K. H. Hinrichs. Visual exploration of seismic volume datasets. *Journal Proceedings of the 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG06)*, 14:73–80, 2006.

[17] M. Visvalingam and K. Dowson. Algorithms for sketching surfaces. *Computers & Graphics*, 22:269–280, 1998.

[18] L. Wang and K. Mueller. Generating subresolution detail in images and volumes using constrained texture synthesis. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 75–82, Washington, DC, USA, 2004. IEEE Computer Society.