SeiVis: An Interactive Visual Subsurface Modeling Application

Thomas Höllt, *Student Member, IEEE*, Wolfgang Freiler, Fritz M. Gschwantner,
Helmut Doleisch, *Member, IEEE*, Gabor Heinemann, and Markus Hadwiger, *Member, IEEE*

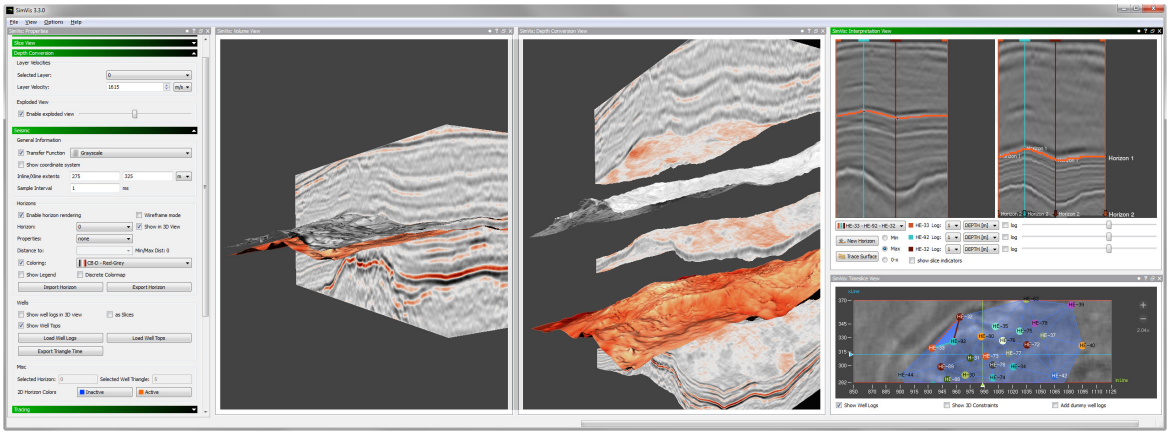

Fig. 1: Screenshot of our application, illustrating joint time/depth domain visualization. A seismic reflection dataset with two interactively interpreted horizons is shown. The two 3D views on the left show volume renderings of the seismic data in time and depth domain, respectively, cut open at the horizons. The two views on the top right show our interpretation views in time and depth as well. The slice view on the bottom right shows the reflection data from the top in combination with well positions.

Abstract— The most important resource to fulfill today's energy demands are fossil fuels, such as oil and natural gas. When exploiting hydrocarbon reservoirs, a detailed and credible model of the subsurface structures is crucial in order to minimize economic and ecological risks. Creating such a model is an inverse problem: reconstructing structures from measured reflection seismics. The major challenge here is twofold: First, the structures in highly ambiguous seismic data are interpreted in the time domain. Second, a velocity model has to be built from this interpretation to match the model to depth measurements from wells. If it is not possible to obtain a match at all positions, the interpretation has to be updated, going back to the first step. This results in a lengthy back and forth between the different steps, or in an unphysical velocity model in many cases. This paper presents a novel, integrated approach to interactively creating subsurface models from reflection seismics. It integrates the interpretation of the seismic data using an interactive horizon extraction technique based on piecewise global optimization with velocity modeling. Computing and visualizing the effects of the changes in the interpretation and velocity model on the depth-converted model on the fly enables an integrated feedback loop that allows a completely new connection of the seismic data in time domain and well data in depth domain. Using a novel joint time/depth visualization, showing side-by-side views of the original and the resulting depth-converted data, domain experts can directly fit their interpretation in time domain to spatial ground truth data. We have conducted a domain expert evaluation, which illustrates that the presented workflow enables the creation of exact subsurface models much more rapidly than previous approaches.

Index Terms—Seismic visualization, volume deformation, exploded views, seismic interpretation.

1 Introduction

Even with the recent trend towards alternative and renewable energy sources more than half of today's energy demand is still fulfilled by fossil fuels and will continue to be in the next couple of years. According to the International Energy Outlook 2010 [22] by the U.S. Energy Information Administration, worldwide marketed energy will rise by 50% until the year 2035.

- *Thomas Höllt and Markus Hadwiger are with King Abdullah University of Science and Technology, Saudi Arabia,*
  E-mail: {thomas.hollt|markus.hadwiger}@kaust.edu.sa.
- *Wolfgang Freiler and Helmut Doleisch are with the SimVis GmbH, Austria,*
  E-mail: {freiler|doleisch}@simvis.at.
- *Fritz-M. Gschwantner is with the VRVis Research Center, Austria,*
  E-mail: gschwantner@vrvis.at.
- *Gabor Heinemann is with the Heinemann Oil GmbH, Austria,*
  E-mail: gheinemann@heinemannoil.com.

Manuscript received 31 March 2012; accepted 1 August 2012; posted online 14 October 2012; mailed on 5 October 2012.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

The oil in place, however, is limited, and thus efficient valorization of existing reservoirs is necessary. For planning production wells to drill into oil and gas reservoirs one has to have an exact model of the subsurface structures. These include the different layers, the boundaries in between, the so called *seismic horizons*, but also *faults*, and other structures. To create such a model, usually a *seismic survey* is acquired which contains seismic reflection data, today often as 3D volumes, called *seismic cubes*, but also other data such as *well logs* and *well tops*. Well logs are 1D datasets that contain the information gathered from drillings in the area of the seismic survey. Well tops contain information on the position of subsurface layer boundaries.

The seismic cube is acquired by sending seismic waves into the ground. At a horizon, part of the waves will be reflected, while others will proceed. The reflected waves are then measured on the ground using a 2D grid of *geophones*. The result is a set of 1D traces, one for each geophone, with z being the two-way travel time of the seismic wave, and $f(z)$ being the amplitude. The seismic traces are then usually *time-* or *depth-migrated*. In this process, the actual lateral positions of the events in the x,y-plane are computed, as in the original data one does not know from which direction the event actually arrived at the geophone. For a 3D survey, the migrated 1D traces are

combined to form a 3D *seismic cube*. The dimensions of the resulting volume are lateral distances for the geophone grid, and time or depth on the z-axis, depending on the migration technique. Even though the depth migration delivers depth on the z-axis, this value does not directly correspond to actual spatial depth in the real world as the *prov-elocities* used for calculating the depth-migration do not account for the horizontal energy in the seismic waves. Thus, time- as well as depth-migrated data still need to be *depth-converted*.

The ground truth data gathered from exploration wells come in three types: (1) measured in spatial depth at the drill holes, (2) measured in spatial depth, and converted to the time- or depth-migration domain, or (3) measured directly in the time domain. Unlike well data available in the time domain, the much more common and also more accurate data only available in the spatial depth domain cannot be used directly for interpreting a time-migrated seismic measurement. In the conventional workflow for seismic interpretation, matching the interpretation to the ground truth data in the spatial depth domain only happens once the interpretation and velocity model are complete and the complete subsurface model has been depth-converted.

Depth conversion is the process of computing actual spatial depths for seismic structures using the extracted horizons and a velocity model. The latter can be simplified to a layer-cake model. Like in a layered cake, each subsurface layer is assumed to consist of a single material only, or an equal distribution of a mixture of materials. This makes it possible to assign an average velocity value to each layer. In the same way subsurface layers do not intersect, neither do their boundaries. We also assume that boundaries do not fold and thus can be defined as a function over the lateral domain, resulting in a height-field. According to our domain expert collaborators, this is a reasonable assumption that subsumes the largest part of seismic interpretation work. These two constraints make it possible to interpret depth conversion as a piecewise linear scaling of layers along the depth axis.

Contributions. We present a novel integrated, interactive application for the creation of subsurface models. We combine three previously separated modules, namely *horizon extraction*, *velocity modeling*, and *depth conversion* (Figure 2a), into a fully integrated update loop (Figure 2b). For the horizon extraction module, we build on the basic idea of previous work [9], but introduce a novel, more precise cost function. We are able to integrate the depth conversion module directly into the rendering pipeline via a novel live volume deformation technique, allowing live evaluation of the results from the horizon extraction and velocity modeling modules. We present a completely new interpretation workflow that shows the original data and the depth-converted data in a novel joint time/depth domain visualization, with live updates in all views. This for the first enables time-domain ground-truth data to be integrated directly into a time domain-based workflow. The former were previously only available in spatial depth. We show the benefits of our application via an evaluation with our domain expert partners, who have started to employ it in practice.

In addition, we present a novel volume-shading approach tailored to highlight horizon structures, as well as a new approach to exploded views using a single-pass ray casting algorithm. Both techniques enhance the visualization of dense seismic reflection data considerably.

2 Related Work

We review related work clustered into three areas that are relevant for our application.

Seismic Interpretation and Depth Conversion. Etris et al. [7] explain the need for depth conversion of the seismic interpretation and why depth migration, even though resulting in a seismic in the depth domain, is not sufficient to get a good subsurface model.

Pepper and Bejarano [16] give a good overview of seismic interpretation techniques in general. There exist a couple of fully automatic horizon extraction approaches. Keskes et al. [10] and Lavest and Chipot [12] present abstract outlines for these kind of approaches. Faraklioti and Petrou [8] as well as Blinov and Petrou [2] later on employ connected component analysis for 3D surface reconstruction of seismic horizons. These fully automatic approaches all require parametrization. Parameters, however, are not necessarily equal for all features and might even vary throughout a feature. Moreover, having to adjust parameters is inconvenient due to lengthy computations.

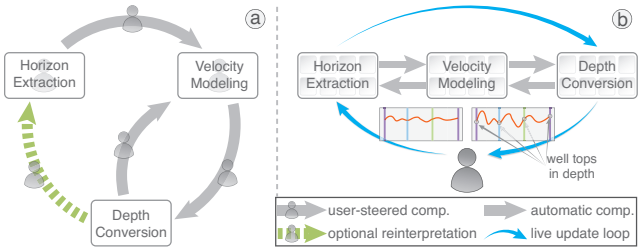

Fig. 2: The conventional workflow (a) with multiple user-steered computations, compared to our novel joint time/depth modeling workflow (b) with a fully integrated, automatic live update loop.

Patel et al. [15] present an interactive workflow for horizon extraction. In a pre-processing step, they compute a hierarchy of possible surface patches, which the user can then interactively put together to create the horizon surfaces.

In previous work [9] we have presented an interactive workflow for the seismic horizon extraction module, which is based on well positions. Rather than tagging horizons on axis-aligned slices, well positions are triangulated, and interaction happens on the sides of the resulting prisms. This makes it possible to easily integrate additional data, like logs and tops acquired at these wells, directly into the interpretation workflow. This workflow constrains most of the user interaction to the unfolded prism sides, allowing easy interaction on a 2D plane, while the actual surface computation exploits global energy minimization to compute a minimal cost path around the prism and a real 3D minimal cost surface on the inside.

Volume Deformation and Exploded Views. Westermann and Rezk-Salama [23] propose a method for free-form volume deformation on graphics hardware. A key element of their approach is to not deform the volume itself, but rather the mapping into the volume. They use slicing for volume rendering, but their approach should also work for ray casting. Rezk-Salama et al. [17] describe an approach for volume deformation by adaptively subdividing the volume into blocks which can be linearly deformed. They reached interactive rendering speeds for small datasets on then-available programmable graphics hardware. Their approach, however, does not work when using advanced memory layouts like bricking. Schulze et al. [21] have presented an approach for non-physically based direct volume deformation. They resample the volume during deformation and render the deformed volume with standard volume rendering. Their technique allows deformation of moderately-sized volumes at voxel resolution at interactive framerates, but the affected area must be limited.

Bruckner et al. [3] present exploded view for volume data. In their approach, the dataset is first split into multiple convex parts which are then transformed using a force-directed layout. They render the parts one by one. Therefore the parts have to be sorted according to their visibility and after rendering blended into a single buffer.

Volume Visualization for Seismic Data. Engel et al. [6] give a comprehensive overview of the basics of volume graphics, including slicing and ray casting-based approaches.

Meaningful visualization of 3D seismic data is a very hard problem. Seismic volume data are very dense and noisy. Gradients cannot be used well in large parts of the volume, and generally have different semantics than for example in typical medical datasets. Where a strong gradient in a computed tomography (CT) scan usually corresponds to a material boundary, in seismic reflection data the subsurface boundaries are represented as local extrema, where the gradients are usually very small. This means that volume illumination with gradient-based approaches like the Blinn-Phong model [1] in combination with classical volume rendering approaches does not work well for these data.

Castanie et al. [4, 5] were the first to use pre-integrated ray casting for seismic visualization. They give an overview of the special demands for visualization of seismic data and demonstrate the advantages of ray casting compared to slicing approaches for direct volume rendering of seismic data.

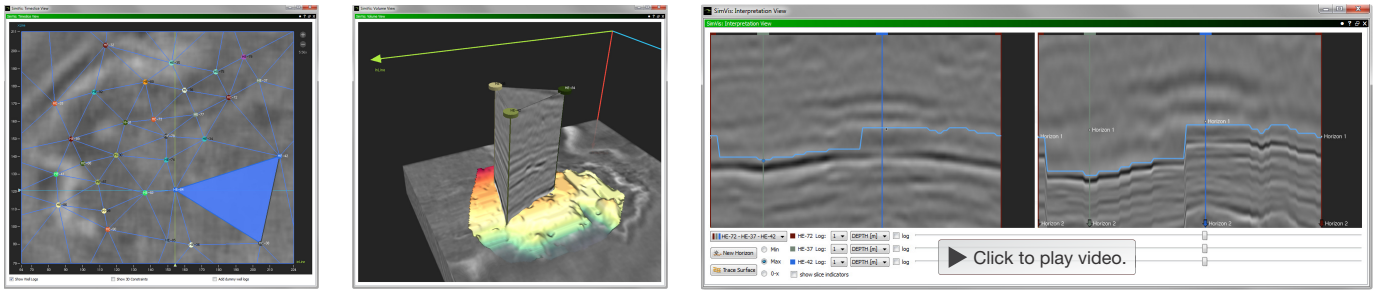

Fig. 3: Fundamental views for our joint time/depth domain workflow. The user can select prisms in the timeslice view (left). The volume view (center) allows to render each prism in context of the volume (in both domains). The main interaction for the horizon extraction happens in the interpretation views (right), showing the unfolded prism sides alongside well data in the time and spatial depth domain.

Patel et al. [15] present a volume rendering technique employing gradient-free shading. They argue that local ambient occlusion as presented by Ropinski et al. [18], a common gradient free shading approach, is not a good fit for seismic data, not only because of the time and memory intensive precomputation, but also because of the high frequency and noisy nature of seismic data. Instead they propose a technique called forward scattering. They implemented a volume rendering approach using view aligned slicing and apply incremental blurring to the accumulated opacity buffer. The buffer is then used to determine the shadowing for the subsequent slices. As a slicing-based approach this technique is not easily adapted to ray casting. Lampe et al. [11] present a technique to deform and render volumes along curves. They show two applications, one of which is the visualization of seismic reflection data along well logs.

3 Workflow

The conventional workflow (Section 3.1) for creating subsurface models and the novel workflow that we propose (Section 3.2) share the same three conceptual modules: Horizon extraction, velocity modeling, and depth conversion. However, as indicated by the illustrations in Figure 2, they differ vastly in how these modules are integrated, as well as implemented. In the conventional workflow, the three modules are connected in a pipelined fashion, where a final result of one module is the input for the next. In contrast, in our workflow the three modules are tightly integrated, and all computations happen on the fly.

3.1 Current Practice

The following description of the current practice is based on the industry standard software Schlumberger Petrel [20]. While implementation details might vary for different software tools, to our knowledge the workflow description reflects the common practice in the industry.

Horizon Extraction. The horizon extraction is usually carried out as a combination of 2D segmentation on the axis-aligned slices of the seismic reflection data. Depending on the variation in the data, up to ten slices are skipped between 2D segmentations, and filled by interpolation or automatic growing. On a single slice, typically the interpreter tags the desired horizons, starting with an automatic trace, which is then manually refined. The auto tracer of the Petrel software uses a local approach, which—unlike our global approach—cannot be forced through constraint points, and stops at ambiguous areas. While there exists a possibility to automatically grow a complete horizon in 3D, the results are in most cases not good enough for production and are usually only used to guide the manual extraction. The horizon extraction process is very lengthy and accounts for the major part of the time in the typical workflow. Tagging several horizons on tens to hundreds of slices takes at least several hours but can easily keep an interpreter busy for multiple days, if the structures are not clearly visible or ambiguous. The output of the horizon extraction module is a set of surfaces describing the boundaries of the subsurface layers.

Velocity Modeling. After the horizon extraction is finished, the resulting horizon surfaces are used as the basis to create a velocity model. The velocity model is defined in a table view, where for every subsurface layer the corresponding top and bottom boundaries (horizons or other surfaces), as well as velocities are set. Once all desired layers are defined, a volume containing per-voxel velocities (the *velocity cube*) can be computed off-line.

Depth Conversion. Finally, based on the velocity cube, the depth conversion can be computed. Using the per-voxel velocities, the original volume and the extracted horizons can be resampled, again off-line, from top to bottom into a new, depth-converted dataset.

Integration. Creating the velocity cube, as well as the final depth conversion, requires the user to manually create a new derived dataset, as well as lengthy computations. In addition, the derived datasets are not coupled, meaning an update in one of the modules does not automatically result in recomputation in the subsequent modules, but rather requires manually creating a new derived dataset and triggering the computation. This results in the linear workflow illustrated in Figure 2a, where each module requires the result of the previous module as input. Due to the lengthy computation in between, usually no intermediate results are pushed to the next module in the pipeline. The drawback of this approach is that mistakes that occur during the horizon extraction, often only become visible after finishing the complete pipeline, when matching the depth converted data to the ground truth data, which is available only in depth. In an optimal workflow, now the interpreter would go back to the interpretation and fix mistakes (indicated by the green arrow in Figure 2a). More commonly, however, a shortcut is taken to save time, by "hot fixing" the velocity cube to match the features, accepting a possibly unphysical velocity model.

3.2 Joint Time/Depth Domain Workflow

Horizon Extraction. Instead of extracting the horizons on axis-aligned slices, we build on the basic idea presented in our previous work [9], and triangulate well positions and tag the horizons on the sides of the resulting prism. For interaction, the sides of the prisms are unfolded into a single 2D image. By using the well positions as the prism corners, we can incorporate the information of three wells into each 2D view. For extracting a horizon, the user just has to place a single seed-point. Using a global minimal path algorithm, we find the optimal path around the prism sides, which is then used to compute the minimal surface for the prism body. If the path or surface do not fit, the user can drag a node in the path to a desired position. A new optimal path is then computed to incorporate the position. The surface can automatically be extended to neighboring prisms using the data on the shared face. Using asynchronous computations of the optimization for multiple prisms, changes by the user on one prism are automatically propagated throughout the complete dataset in the background. Section 4.2 introduces a new cost function that is specifically suited to seismic data, and that enhances the precision of the automatic horizon extraction.

Velocity Modeling. For the velocity modeling, we automatically sort the horizons and allow the user to specify the velocity for the resulting layers. In contrast to the conventional workflow, we do not need to compute a velocity cube explicitly, but rather keep the description only.

Depth Conversion. Instead of resampling the original dataset into the depth domain off-line, we compute the depth conversion live during rendering (Section 5.2). The volume, as well as the unfolded prism sides, are deformed live, using the conceptual velocity model in an indirect texture look-up.

Joint Time/Depth Domain Interaction. By removing the penalties for computing a velocity cube and resampling into the depth domain, we can now compute the depth conversion in real time. All three modules are connected, and updates can be triggered automatically whenever the user modifies the state of any module. This allows for a completely new workflow, in which for the first time, ground truth data that are only available in the depth domain, can be integrated directly into the horizon extraction process in the time domain. We call this technique *Joint Time/Depth Domain Interaction* (Section 5).

To take full advantage of our live depth conversion, creating a horizon segmentation and the velocity for the resulting layer go hand in hand. Usually, the velocity values are defined using data from well logs. In our approach, a very simple method is possible. If well top data are available in time and spatial depth, the user can simply tune the velocity value to match the well tops in both domains, using our joint time/depth domain interaction. Once the velocity is defined, all changes in the horizon segmentation are immediately reflected in the depth-converted model. The user can now use the well top data, available only in the depth domain, to guide the segmentation in the time domain. If the user is unsure about a segment of the horizon, he can drag the segmentation in the time domain and see immediately how the model fits the ground truth data in the spatial depth domain. Figure 3 illustrates this process. Since all computations happen in real time, enabling a live feedback loop at any stage during the interpretation, the interpretation can always be matched perfectly to the ground truth data in the first pass. This eliminates the need for costly post-processing or "hot fixing" the velocity cube.

3.3 Discussion

Most seismic interpretation is done in the time domain. However, additional ground truth data gathered from exploration wells are usually available in spatial depth only. By placing the depth conversion at the end of the interpretation pipeline, as in the conventional workflow, the interpretation is often a guessing game where errors only become apparent once the completed interpretation is depth-converted and matched to the ground truth data. "Hot fixing" these errors in the velocity cube often results in an unphysical velocity model. Having a live depth conversion of the intermediate interpretations, as in our proposed workflow, makes a huge difference for the interpretation workflow. For the first time, features in the time as well as the spatial depth domain can be matched live during interpretation, eliminating the need for a cumbersome back and forth between the different modules, and the need for taking shortcuts that result in incorrect models.

4 Horizon Extraction

Our horizon extraction module is based on a graph-based global optimization with the following properties: A voxel in the volume maps to a node in the graph. Directly neighboring voxels are connected by an edge, and four neighbors in a plane, connected by a cycle of four edges, form a facet. Costs are assigned to edges and facets based on their bounding voxels. The minimal path around each prism is the set of connected edges with the least combined cost bounded by a predefined start and end point. The minimal surface on the inside of the prism is the set of facets with the least combined cost that forms a closed surface that is bounded by the minimal path. While this idea was also used in previous work [9], the cost function used there has several shortcomings, which we describe in Section 4.1 to motivate the development of a new cost function. We propose a new waveform-based cost function (Section 4.2), as well as a new render mode based on this cost function (Section 4.3).

4.1 Motivation for the New Cost Function

The cost function presented in [9] consists of two components. The first one defines the *snappiness* to ridge and valley lines/surfaces. The second component defines the *smoothness* of the resulting interpretation. Both components are based solely on the gray values of the current sample, ignoring the neighborhood. The snappiness term at a node is defined as the distance of the amplitude at the current node $f(x,y,z)$ to a predefined target amplitude t. The cost g_1 for an edge or a facet F is then defined as the sum over all nodes belonging to the edge/facet as:

$$g_1(F) = \sum_F |t - f(x,y,z)|\,. \tag{1}$$

The target value t is predefined globally as the minimum or maximum amplitude in the dataset.

Even though it has proven to be quite effective, this basic cost function has a major problem: Horizons are indicated in the data by local extrema of the 1D waves at each (x,y)-position. As can be seen in Figure 4a, the resulting costs, however, are not necessarily an indicator for extrema. Even though locally g_1 will assign the lowest cost to an extremum, the costs at different positions in the volume or for different horizons are not comparable, as an extremum might be different from the absolute target value.

4.2 The New Waveform-Based Cost Function

To overcome the problems described in Section 4.1, we propose to replace the snappiness term of the cost function with a new term, based on the local waveform of the 1D trace. The obvious approach to such a term would be computing the derivative, e.g., with central differences to find extrema, and then use the result of the central difference computation as the cost. In addition, one could modulate the result to enhance the extrema for example using a smooth bump function such as:

$$\phi(v) = \begin{cases} \exp^{-\frac{1}{1-v^2}} & \text{for } |v| < 1 \\ 0 & \text{else} \end{cases} \tag{2}$$

where v is the derivative $f' = df(x,y,z)/dz$ scaled with a user-defined scale factor α to define the threshold for mapping the result to zero. Figure 4b shows the plot of the cost defined as:

$$g_\phi(F) = \sum_F 1 - \phi(\alpha \cdot f'(x,y,z)). \tag{3}$$

For the plot, minima are canceled out using the second derivative. While this is a much better result compared to the old cost function in most places (compare for example the small local maximum marked by the green box), especially the sharp features are desired, some problems become obvious in the plot. While minima can be canceled out effectively using the second derivative this is not true for saddles, resulting in undesired low costs at the two saddle-points marked by the blue boxes. In addition, the strong maximum marked with the magenta box was assigned a relatively high cost due to the fact that it has two samples on its peak which results in an asymmetry of the samples around the peak.

These shortcomings can be removed by the use of the following non-linear waveform-based term to replace g_1:

$$g_{\text{wave}}(F) = \sum_F 1 - \sum_{k=1}^{n} \varphi_s(x,y,z,k), \tag{4}$$

where n is a predefined 1D neighborhood-size and

$$\varphi_s(x,y,z,k) = \begin{cases} \varphi(x,y,z,k) & \text{if } (k \geq s) \text{ or} \\ & \quad (k < s \text{ and } \varphi(x,y,z,k+1) \neq 0) \\ 0 & \text{else} \end{cases} \tag{5}$$

with

$$\varphi(x,y,z,k) = \begin{cases} \frac{1}{n} & \text{if } f(x,y,z-k) < f(x,y,z) \text{ and} \\ & \quad f(x,y,z+k) < f(x,y,z) \\ 0 & \text{else} \end{cases} \tag{6}$$

for maxima, or

$$\varphi(x,y,z,k) = \begin{cases} \frac{1}{n} & \text{if } f(x,y,z-k) > f(x,y,z) \text{ and} \\ & \quad f(x,y,z+k) > f(x,y,z) \\ 0 & \text{else} \end{cases} \tag{7}$$

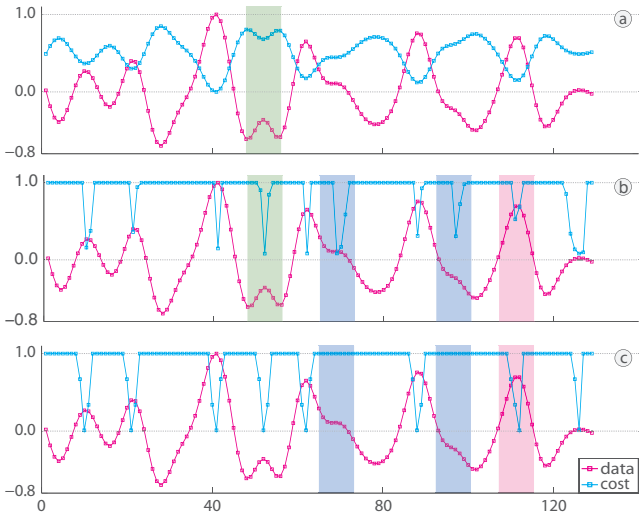

Fig. 4: Comparison of the cost to the maxima for an exemplary 1D seismic trace. a) shows the old snappiness term from [9] (Equation 1), the derivative based term is shown in b) (Equation 3) and our new non-linear term in c) (Equation 4).

for minima, respectively. This cost function is a simple step function which basically returns a smaller value the closer the current sample point is to a local extremum in the predefined neighborhood and 1 if there is no extremum in the neighborhood. The additional constraint in φ_s provides an implicit smoothing, using s as a minimal feature size.

The resulting cost-plot is shown in Figure 4c. As can be seen, the problems for the derivative-based term are canceled out in this plot, while the sharp features for the maxima are retained.

4.3 Cost Function-Based Volume Rendering

Inspired by the gradient magnitude-modulated shading presented by Levoy [13], and the cost function presented in Section 4.2 we have developed a shading approach that focuses on highlighting horizon structures. While in horizon extraction local waveform inspection is common, to our knowledge there are no attempts to exploit the local waveform for enhancing the volume rendering of seismic data.

Prototyping the Cost Function. With the non-linear filter presented in the previous section, we can roughly estimate the distance of the current sample to a local minimum or maximum in the trace. While the cost function term shown in Equation 4 maps extrema to low cost for the rendering, we would like to assign large opacity to the extrema and low opacity to the remaining data. Therefore we can directly use the term from Equation 5 to modulate the opacity during volume rendering. The resulting opacity is a direct indicator of the local cost. This makes it possible to prototype the cost function parameters, such as the neighborhood size, while getting an on-the-fly visualization of the resulting local costs. Areas of low cost, corresponding to the horizon structures, are rendered opaque, while areas of high cost will be rendered transparently. Even though this technique cannot replace the additional global optimization for extracting the horizons, it can give a good idea of how the global optimization will behave.

Horizon Enhancing Shading. In addition to using this technique for prototyping the cost function, it also works well as a general approach to highlighting horizon structures in seismic reflection data. Common shading approaches use the gradient for shading. For visualizing horizons, this is not suitable as the gradient vanishes at the local extrema, indicating the horizons. Hence we compute the inverse of the local cost and use it as opacity modulation in the transfer function, in the same way as the gradient magnitude is used for gradient magnitude-modulated shading [13].

Table 1: Performance for cost function-based rendering compared to Phong shading (base). See Section 5.4 for measurement setup.

Neighborhood Size	Rendering Times		
	cf-based	base (Phong)	% of base
3	28fps	28fps	100%
5	25fps	28fps	89%
7	21fps	28fps	75%

Performance is mostly dependent on the neighborhood size. We show performance for $k = 1, k = 2$ and $k = 3$ in Table 1, compared to a shader using standard Phong shading without modulation. It can be seen that, while there is a performance loss for neighborhoods larger than three voxels, rates are still interactive. For the performance comparison we adjusted the transfer functions such that transparency was similar for the standard and cost-modulated shading methods, as the modulation usually results in much more transparent images. Figures 5a-c show a comparison of the quality with different neighborhood sizes. Figure 5d shows a rendering using the same transfer function with standard Phong shading. The cost-based rendering was set up to highlight local maxima only.

A big advantage of this approach is that horizons can efficiently be highlighted without adjusting the transfer function. While similar results could be achieved with a carefully designed transfer function with amplitude-based shading, using the proposed render mode requires no user interaction at all. In addition, not all features can be highlighted with a transfer function based on gray values. Compare the trace in Figure 4: using an amplitude-based classification, the local maximum highlighted by the green box could not be rendered without also rendering everything else with the same amplitude, which is mostly in the range of local minima.

5 Joint Time/Depth Domain Interaction

Our joint time/depth domain workflow is only possible with live computation of the results of the depth conversion. Therefore, an efficient rendering pipeline (Section 5.1), which builds the foundation of our framework as well as the live volume deformation (Section 5.2) are essential. In addition we present a simple implementation of a single pass exploded views algorithm (Section 5.3) based on the same technique as the live deformation which enables efficient exploration of the dense seismic reflection data.

5.1 Rendering Pipeline

Figure 6 shows our integrated pipeline for rendering seismic volume data and horizon surfaces with the application of deformation and exploded views. While the figure illustrates the pipeline for the volumetric case we use the same pipeline for the unfolded prism views to provide depth conversion of this view during interpretation. The main difference is that the heightfield geometry as well as the boundaries texture are of one dimension less (compare Table 2). The pipeline is divided into two major blocks, the interpretation block on the left and the rendering block on the right.

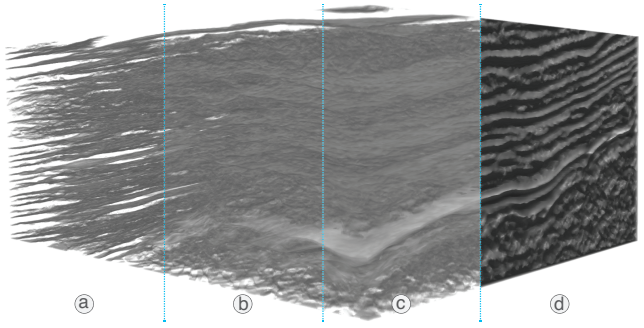

Fig. 5: Different kernel sizes for the cost-modulated shading (a: $k = 1$, b: $k = 2$, c: $k = 3$), and comparison with Phong shading (d).

Table 2: Textures and buffers needed in our pipeline in addition to the existing textures needed for basic volume rendering. #B and #L represent the number of boundaries and number of layers, respectively. Sizes are in number of 32bit floating point entries.

Buffer	Dim. 3D	Dim. 2D	Type	Function	Size 3D	Size 2D
heightfield geometry	2D	1D	vertex buffer	generic vertex buffer for horizons	$x \cdot y \cdot 3$	$x \cdot 2$
boundaries	3D	2D	luminance-α texture	layer boundaries in original (.r) and deformed (.a) space	$x \cdot y \cdot \#B \cdot 2$	$x \cdot \#B \cdot 2$
velocities	1D	1D	luminance texture	layer velocities	#L	#L

Interpretation comprises two modules. The first module combines the interactive horizon extraction and velocity modeling modules described in Section 3.2. For each created horizon the module delivers a 2D heightfield which covers the complete volume domain plus a 1D heightfield covering the boundary of the current prism. Both are constantly updated during the interpretation. Each heightfield, corresponding to a single horizon, is stored as a layer in the first channel of the 3D or 2D boundaries texture on the GPU side, and is also available to the depth conversion module on the CPU side.

The surface deformation module takes the updated heightfields and converts the values from the time to the depth domain using the defined velocity model. Compared to recomputing the deformation for the complete volume, very little data needs to be processed. We also parallelized the CPU-based computation using OpenMP. The complete computation can be done on the fly. Table 3 shows the computation time for a complete boundary update for two to five layers. The small added penalty for more than two layers can be explained by the overhead for setting up the multi-threaded computation. Additionally, in a typical workflow, where the interpretation is done from top to bottom, only the last two boundaries need to be updated when modifying the last layer, allowing more than 100 updates per second.

The resulting depth-converted heightfields are stored in the second channel of the 2D or 3D boundaries textures described above. In addition, the depth conversion module outputs a 1D texture containing the velocity values for each layer and also the maximum scaling factor needed to cover the depth conversion at any x,y-position.

Rendering. The data is shared between all views and steps in the visualization pipeline. Table 2 gives an overview of the shared textures and buffers. Basically all of our views make use of vertex and fragment shaders to exploit the possibilities of the programmable OpenGL pipeline. We used this for example to streamline the horizon surface rendering part of the pipeline. Instead of creating geometry for each horizon, we use a single generic vertex buffer, covering the complete x,y-domain, but without any depth (z) information at the vertices. We render the surfaces one by one and use the boundaries texture in the vertex stage to assign z-values to each vertex and move it to the appropriate position.

Invalid fragments, i.e., those belonging to triangles in the mesh which are not yet covered by the interpretation, are discarded. Additionally, we use the fragment shader to compute several properties

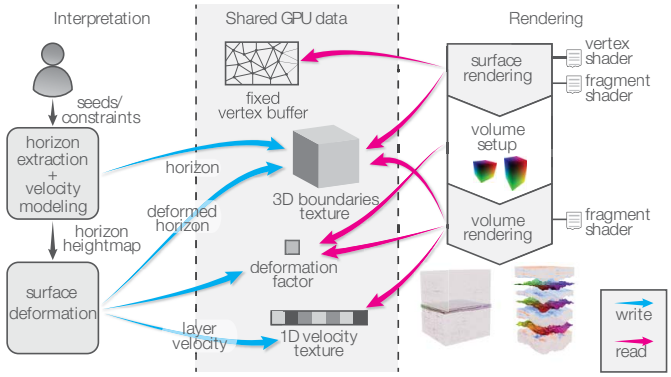

Fig. 6: Rendering pipeline for joint time/depth interaction. Surface extraction and deformation modules constantly update the texture data. The data is shared over all views and the different steps in the pipeline.

of the surfaces on the fly. Using the data which is already available for the other rendering stages on the GPU, several properties can be plotted directly onto the surface without precomputing a separate texture for the surface. The amplitude can be looked up directly from the volume texture. Cost or deviation from the target amplitude can be computed on the fly based on data from the same texture. The distance to other surfaces can be evaluated using the boundaries texture.

5.2 Volume Deformation

The volume deformation required for the depth conversion is highly constrained. Deformation only needs to be applied to the depth axis of the volume, to convert its unit from time to spatial depth in the subsurface. Horizons are represented as heightfields, and we can safely assume that they do not intersect. Thus, every two adjacent horizons enclose one subsurface layer. The velocity for each layer in the volume can be assumed to be constant, using an average value. Thus, the deformation can be simplified to a piecewise linear stretching or compression of the volume between two horizons. Taking these constraints into account, it is possible to implement the deformation in a very simple and highly efficient manner, allowing depth conversion at real-time frame rates during volume and slice rendering without precomputing the deformed volume.

Concept. Our approach is inspired by the work of Westermann and Rezk-Salama [23]. In contrast to other approaches, they do not resample the volume but rather deform the mapping into the volume. Conceptually we never deform the original volume either, but use a virtual deformed volume and convert the look-ups in this volume into the original volume space on the fly in the fragment shader. We do that by converting only the layer boundaries (which are the result of the interpretation in progress) from time to depth. For volume rendering, we use a single-pass ray caster. The ray caster is set up to cast into a virtual volume, resized to fit the deformed volume. During ray casting, the depth-converted boundaries are used to compute the actual look-up in the original volume. The process for rendering the unfolded prism sides is very similar. For each side, we set up a quad that fits the size of the deformed side, and the same indirect texture look-up as described for volume rendering is used for the slice rendering.

Integration into Rendering Pipeline. The result of the interpretation (in progress) is a set of heightfields corresponding to the seismic horizons in the time domain. The heightfields represent the layer boundaries inside the volume. In addition to these boundaries we also need the top and bottom boundary of the volume. Layer 0 is then bounded by the top boundary of the volume on top and horizon 0 on the bottom, layer 1 by horizon 0 on top and horizon 1 on the bottom, and so on. The top boundary functions as our start value and is equal in both, the time and depth domain. Starting with horizon 0 the layer boundaries are iteratively converted from depth to time by multiplying the thickness of the prior layer at each x,y-coordinate with a user-defined velocity value and adding the resulting depth-converted thickness to the computed depth value of the previous boundary. The boundaries have to be recomputed only when a velocity value is changed or a horizon is modified. Computational complexity is $O(n)$, with n the number of vertices which need to be updated. Since the computation is independent for each x,y-position, it can easily be parallelized. For the surfaces in our dataset, sized 240×240, the update time for a couple of horizons is well within real-time performance (see Table 3).

In the setup step of the ray caster, the extents of the bounding geometry have to be adapted to fit the deformed volume. This is done by scaling the z-axis of the bounding geometry with the maximum dis-

Table 3: Performance comparison of the live depth conversion and exploded views. We used standard ray casting without illumination for the comparison. The dataset used for comparison is shown in Figure 9.

Number of layers	Compute layer update	Rendering Times					
		base	depth conversion	% of base	dc + exploded views	% of dc	% of base
2	9ms	82fps	53fps	65%	40fps	75%	49%
3	11ms	82fps	50fps	61%	32fps	64%	39%
4	12ms	82fps	46fps	56%	27fps	58%	33%
5	13ms	82fps	42fps	51%	24fps	57%	29%

tance between the deformed top and bottom boundaries of the volume. A tighter fit of the bounding geometry, which would enable empty space skipping below the bottom boundary could be achieved by scaling each x, y-coordinate with the actual distance between the deformed boundaries at this coordinate. However, this would result in a more complex shader with the need for an additional texture for the scales per x, y-coordinate and an additional look-up in this texture per sample. With our simple setup we can just pass a single uniform as scale factor for all samples. The performance loss due to not skipping empty space below the boundary is most likely canceled out by the simpler shader and fewer texture look-ups.

Fragment Shader. In the ray casting fragment shader the coordinates for the volume texture look-up have to be modified. Pseudo code, similar to GLSL shader code, describing the conversion of the coordinates to the actual sampling position in undeformed volume coordinates can be found in Figure 7. Setting up the bounding geometry as described results in sampling positions in the deformed volume space scaled to $[0, 1]$. First the z-value of the current sample's position is multiplied with the scale factor provided to the shader as a uniform. This gives us the position in the deformed volume space at the same scale as the original volume. To get the layer id, all layers are checked whether the sample lies within their bounds. Once found, the layer is returned. The top boundary of the layer is now looked up in the original and deformed boundaries textures. These values correspond to the same point in different spaces and are used as an offset. By subtracting this offset in the deformed space from the sample z-value we get the position in the layer. As we assume a constant velocity for that layer we can simply divide this position by the layer's velocity resulting in the position inside the layer in undeformed space. By adding the offset received from the un-deformed boundaries texture we get the position in undeformed volume space.

The same principle is used for the main interpretation view containing the unfolded prism sides. To apply the depth conversion to the slice views, we simply scale the side of the quads corresponding to the volume's z-axis and use the same code as shown in Figure 7 in the fragment shader to compute the position for the texture look-ups.

```
uniform sampler1D velocities;
uniform sampler3D boundaries;
uniform float scaleFactor;

function CONVERTSAMPLEPOSTOVOLUMESPACE( x, y, z )
    // scale from [0..1] to deformed volume coordinates
    z *= scaleFactor;
    // get the id of the current layer
    float layerId = GETLAYERID( x, y, z );
    // lookup layer offset in undeformed and deformed volume coordinates
    vec2 layerOffset = texture3D( boundaries, vec3( x, y, layerId ) ).xw;
    // distance to layer boundary in deformed volume coordinates
    float posInLayer = z - layerOffset.y;
    // distance to layer boundary in deformed volume coordinates
    float velocity = texture1D( layerVelocities, layerId );
    posInLayer /= velocity;
    // update z coordinate to sample in volume coordinates
    z = layerOffset.x + posInLayer;
    return vec3( x, y, z );
end function
```

Fig. 7: GLSL-like code for the coordinate conversion used for the live volume deformation.

5.3 Exploded Views

The general approach to exploded views [3, 14, 19] for volume rendering is setting up the bounding geometry around each part of the volume, render each of the parts separately, and then blend the results. These techniques make very flexible exploded views possible, allowing translation and rotation of arbitrary cut planes along/around arbitrary axes. However, the same constraints of our application scenario which allow us to set up the simple volume deformation approach described above also make much of this flexibility unnecessary. We use the seismic horizons as the cut geometry, and then for the explosion itself compute layers of translations along the z-axis. This is sufficient to enable unobstructed views onto the surfaces.

Integration into Rendering Pipeline. The data already available for the deformed rendering makes it possible to deploy exploded views, using the z-axis for translation, and the horizons as cut surfaces. We use a single-pass ray casting approach without any additional setup besides a modified scale factor. Again, we set up a virtual volume with a modified size for the z-axis. To comply with the spacing in between the different layers, here the summed up spacing for all horizons is added to the scale factor for the volume's z-axis. Adding the spacing to the scale factor for the depth conversion allows us to combine the exploded views with the depth conversion.

Fragment Shader. The fragment shader for rendering the exploded view is only a slight modification of the shader described in Section 5.2. We interpret the spacing as empty space preceding each horizon. Thus, the offset to get to the deformed layer is not a direct look-up in the deformed boundaries texture, but the spacing corresponding to all preceding horizons has to be added. Assuming a uniform spacing, this is simply layerId times spacing. To check whether the sample position is in the spacing or in the volume, the deformed boundary value of the bottom boundary of the current layer has to be fetched. If the current sample position is larger than the deformed boundary value minus the size of the spacing, the sample is in the spacing. If that is the case, we can set the current sample's alpha value to zero and proceed with the next sample. The user interaction for this technique is also very simple, as only the spacing needs to be defined, which can be done using a slider in the GUI.

5.4 Performance

For the following performance measurements, we use a volume of $240 \times 240 \times 1509$ voxels, shown in Figure 9, with one to four interpreted horizons with: deformation only; deformation and exploded views; standard ray casting. The horizons cover the complete volume at a resolution of one vertex per voxel resulting in surfaces of 57.600 quads each. Computation of the deformed surfaces was done on the CPU using a dual six-core Xeon X5680 at 3.33Ghz. Rendering was done on a NVIDIA Geforce GTX 580 with 1.5GB of memory, with the volume rendered screen-filling into a 1900×765 viewport with two samples per voxel.

Performance for the ray casting algorithm with live depth conversion, as presented in Section 5.2, is shown in Table 3. In comparison to the standard ray casting algorithm used for rendering the un-deformed volume, this technique requires three additional texture look-ups and four floating point operations per sample to compute the sample position in the original volume. In addition, to get the layer id of the current sample, a number of additional texture look-ups have to be performed. The actual number depends on the search algorithm used. Right now we step through all layers from top to bottom until reaching the current

sample point. This results in the performance loss shown in Table 3, when adding more layers. Performance for the single-pass ray casting with exploded views, described in Section 5.3, can be seen in Table 3. Compared to the depth-conversion shader, there is virtually no difference in computational complexity (the texture fetch, described above, to decide whether the sample is inside the spacing area can be cached when computing the current layer). However, we currently do not employ empty space skipping in the explosion spacing, which results in a performance loss.

6 Evaluation

We have evaluated our system in a real-world scenario with our domain expert partners, in order to compare our workflow with the industry standard as described below.

6.1 Setup

For comparison we asked our partners to provide us with a typical application scenario. To avoid an interpretation session over several days we asked for a rather small dataset. As time and effort should increase linearly with the number of prisms for our proposed workflow and with the number and size of slices in the slice-based approach a small dataset is representative of the relative performance. We were provided with a moderately-sized dataset consisting of 325 inlines and 275 crosslines at 151 samples per trace (Figure 5 was produced with the same dataset). The dataset covers an area of roughly 7.5×6.5km in western Hungary. Time between samples was 4ms. In addition to the seismic cube, we were provided with a total of 39 well positions, with well tops for two horizons each, of which 5 were available in depth and time, and the remaining 34 in depth only. A large part of the dataset was covered by the resulting triangulation. We added 10 more positions so that it is possible to create the interpretation up to the volume's boundaries without falling back to a slice-based approach.

The task for the interpreter was then to interpret the horizon defining the top boundary of a clay layer throughout the volume. Well tops indicating the top and bottom boundaries of this layer were provided. Velocity modeling is usually done using well logs. Using our joint time/depth workflow, the interpreter could simply adjust the velocity values such that the well tops available in time and depth were matching. For comparison we gave the expert one hour with our workflow and one hour with the standard workflow using the Petrel [20] software framework and asked to create an interpretation, as complete as possible, without sacrificing exactness. A comparison of the resulting interpretations is shown in Figure 8. Table 4 shows timing comparisons of the different steps. For quality comparison we were provided with the actual interpretation of the same horizon used for production, which was created manually over several working days.

Before starting the evaluation, we gave our partners an in-depth introduction to our framework and provided guidance during a test drive with a different dataset for several hours. Even though we also supervised the actual evaluation run, the expert was able to work on his own and did not ask for any further assistance.

6.2 Results

Joint Time/Depth Domain Workflow. After the test drive, the domain expert decided to approach the interpretation task in the following way: First, he adjusted the velocity model to match the well tops available in the time as well as depth domain. Then he started with seeding a horizon at one prism and adjusted the 2D curve on the prism side. Using the live depth conversion, he was able to directly match the interpretation to the well tops regardless of the domain they were available in. In the background, our system would use the intermediate interpretation to recursively create interpretations on the neighboring prisms as well as compute the optimal surface on the inside. After being satisfied with the current prism, the expert would then proceed to a neighboring prism and check the precomputed curve and adjust it if necessary. The expert was able to go over all prisms in well under one hour. After finishing the complete horizon, he quickly checked the result by skimming inlines and crosslines and looking at the extracted horizon overlaid with the amplitude. The exploded views were very helpful in this step, as he could get a clear look at the horizon in context of the volume and also look at the volume data itself, which was revealed by cutting at the horizon position. He then proceeded to adjust a few imprecisions resulting from the 3D optimization. After the last change, the expert had to wait only for a few seconds for the final result, as the optimization runs continuously in the background.

Conventional Workflow. For the comparison run, the expert interpreted the same horizon using Petrel, which is the industry standard software for geological modeling. The software offers automated tracing on 2D slices, as well as in 3D. From what we could gather from testing these automated approaches are both local. Starting from a seed in 2D, or an interpreted slice in 3D, the horizon is propagated into as many neighboring pixels/slices as possible. When the data gets ambiguous, the auto-tracer stops. It is also not possible to set constraints besides the initial seed. If the interpretation is wrong, it needs to be fixed manually and 3D traces between two interpreted slices do not necessarily connect these slices. When doing the interpretation, the expert started out with a single slice and fully tagged the horizon on this slice using a combination of piecewise auto-tracing and manual intervention for corrections, as well as to connect the automatically traced parts at areas where the automatic approach stopped. After that

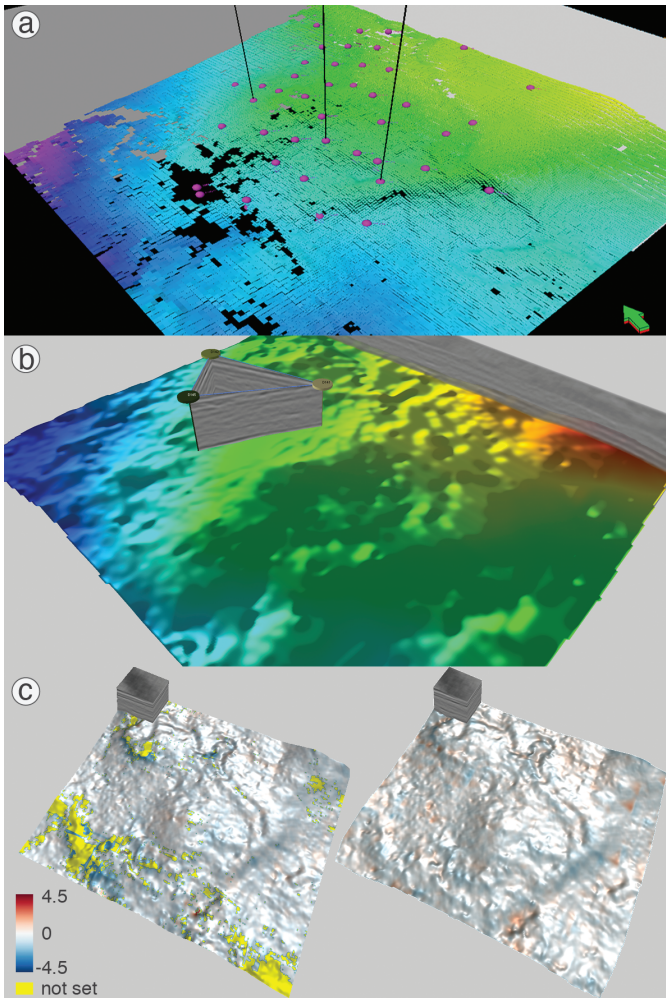

Fig. 8: Comparison of the horizons extracted during the expert evaluation. a) shows a screen capture of Petrel with the extracted surface using the conventional workflow. b) shows the result using our workflow. The relative height is encoded in the surface coloring, but with slightly different colormaps for a) and b). We imported the result from Petrel into our application to compare it to our result. Error plots on the ground truth surface are shown in c). The left image shows Petrel, with gaps in the surface marked in yellow, the right shows our result.

Table 4: Timings for the interpretation process. Initial Interpretation corresponds to a first complete extraction of the horizon. Velocity- and DC Computation correspond to the time needed to compute the velocity cube and the depth conversion, respectively. Refine describes the time used by the expert to refine the automatic interpretation between the manually interpreted slices and inside prisms after the first complete run.

App	Initial Interpretation	Velocity Computation	DC Computation	Refine	#Slices/Prisms	Avg Time per Slice/Prism
Petrel	> 60min*	21s	29s	n/a+	18*	3:20min
Our	≈ 45min	on the fly	on the fly	< 10min	63	43s

*With the Petrel software the expert was able to finish 18 slices (about 60% of the dataset) before the time limit of 60 minutes. +Not reached before the time limit of 60 minutes.

he used the 3D auto-tracer for a first result. Besides the described disadvantages of using a local approach, a big advantage compared to our global approach is that the auto-tracer computes a result nearly instantly. However, according to our expert the resulting surfaces are far from complete and need thorough inspection. As a result they are used mostly for guidance during a manual interpretation rather than for directly producing final results. After this first slice, the expert gradually refined the interpretation by manually adjusting every tenth slice followed by another 3D trace. After 60 minutes, the expert finished 18 slices with a distance of ten, covering roughly 60% of the dataset.

Comparison. Figure 8 shows comparisons of the created surfaces. For the manually interpreted part, it can be seen that the results of both approaches are quite similar. The lower third shows the part that was not reached within the time frame. There, large holes and also a phase jump can be seen, which clearly shows that the automatic tracing alone is not sufficient to reach acceptable results with the conventional workflow. After the 60 minutes, we also measured the time for computing the velocity model and the deformed seismic cube (see Table 4).

After the interpretation, we conducted a brief interview. The expert was impressed by the live depth conversion in combination with the prism-based workflow. Having three well tops on each 2D slice and being able to match the interpretation directly to the tops in the depth domain was really helpful in finding the correct surfaces. A big advantage of the global optimization used for the automatic tracing is the possibility to set constraints and force the auto-tracer through defined points. Also, it will always result in a closed surface even when the data is unclear. A point that we did not expect was made by the expert that the prism-based approach was also very motivating in contrast to the standard approach of going from slice to slice. The data on nearby slices is often very similar, making slice-based interpretation a very tedious process so that subtle, but important, changes are sometimes missed by the interpreter.

Conclusion. We can say that the expert was able to work with our system on his own after a short introduction. He was able to extract surfaces close to manual interpretations in very short time. Being able to match an interpretation in the time domain to ground truth data in the depth domain can not only significantly speed up the interpretation process but also allows very exact results without tedious back and forth between the different domains. By using asynchronous computations in the background, it is also possible to minimize waiting times for the interpretation, even though the global optimization technique is much more computationally demanding than local approaches. Following the evaluation, our partners installed our application on a testing system where it is now used for experimental projects.

7 Conclusions and Future Work

In this paper, we have introduced and evaluated SeiVis, a novel, interactive application for integrated seismic interpretation and the creation of subsurface models. The main contributions are a novel joint time/depth domain workflow for creating subsurface models, merging time and spatial depth domains using on-the-fly volume deformation, live exploded views using seismic horizons as cutting geometry between volume parts, a horizon-enhancing shading mode, and the integration of these techniques into an integrated workflow for seismic interpretation and depth conversion. The expert evaluation that we have conducted has clearly shown the advantages of this new workflow.

In the future, we would like to extend the interpretation capabilities of our system to fault extraction to be able to create more complete subsurface models. We think of precomputing a likelihood of belonging to a fault for each voxel, and using this in the cost function to steer the horizon tracker along faults.

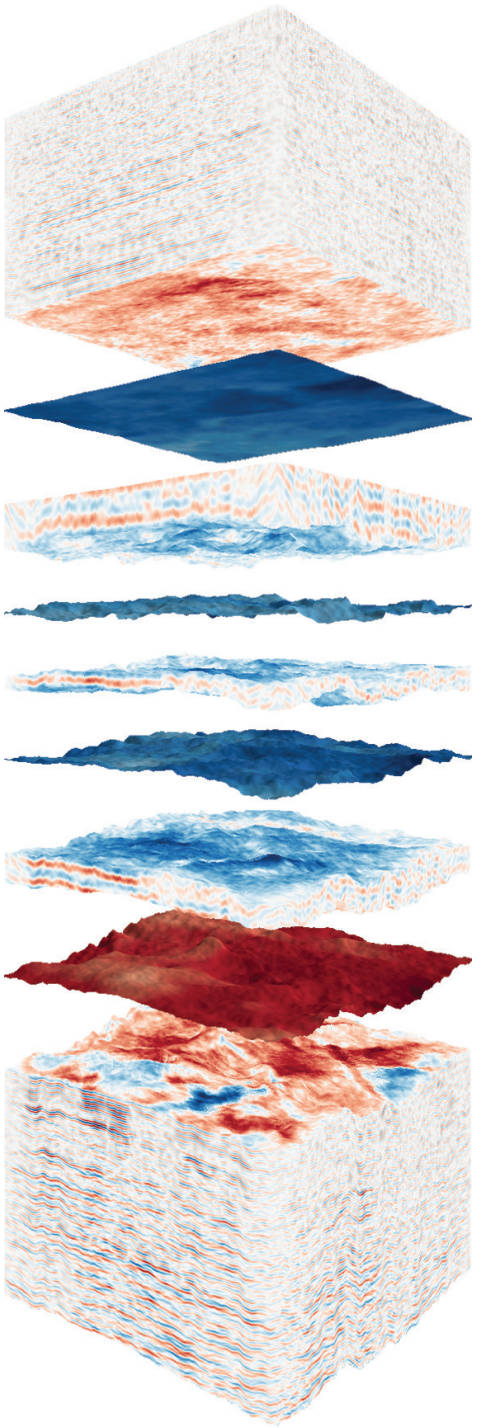

Fig. 9: Example rendering of a dataset of size $240 \times 240 \times 1509$ with heavily compressed top and bottom layers, using exploded views in combination with standard direct volume rendering.

REFERENCES

[1] J. F. Blinn. Models of light reflection for computer synthesized pictures. In *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '77, pages 192–198, 1977.

[2] A. Blinov and M. Petrou. Reconstruction of 3-d horizons from 3-d seismic datasets. *IEEE Transactions on Geoscience and Remote Sensing*, 43(6):1421–1431, 2005.

[3] S. Bruckner and M. E. Gröller. Exploded views for volume data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1077–1084, 2006.

[4] L. Castanie, B. Levy, and F. Bosquet. Advances in seismic interpretation using new volume visualization techniques. *First Break Journal*, pages 69–72, 2005.

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

[6] K. Engel, M. Hadwiger, J. M. Kniss, C. Rezk-Salama, and D. Weiskopf. *Real-Time Volume Graphics*. A K Peters, Ltd, 2006.

[7] E. L. Etris, N. J. Crabtree, J. Dewar, and Pickford. True depth conversion: More than a pretty picture. *CSEG Recorder*, 26:11–22, 2001.

[8] M. Faraklioti and M. Petrou. Horizon picking in 3d seismic data volumes. *Machine Vision and Applications*, 15:216–219, 2004.

[9] T. Höllt., J. Beyer, F. Gschwantner, P. Muigg, H. Doleisch, G. Heinemann, and M. Hadwiger. Interactive seismic interpretation with piecewise global energy minimization. In *Proceedings of the IEEE Pacific Visualization Symposium 2011*, pages 59–66, 2011.

[10] N. Keskes, P. Zaccagnino, D. Rether, and P. Mermey. Automatic extraction of 3-d seismic horizons. *SEG Technical Program Expanded Abstracts*, 2(1):557–559, 1983.

[11] O. D. Lampe, C. Correa, K.-L. Ma, and H. Hauser. Curve-centric volume reformation for comparative visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1235–1242, 2009.

[12] P. Lavest and Y. Chipot. Building complex horizons for 3-d seismic. *SEG Technical Program Expanded Abstracts*, 12(1):159–161, 1993.

[13] M. Levoy. Display of surfaces from volume data. *Computer Graphics and Applications, IEEE*, 8(3):29 –37, may 1988.

[14] M. J. McGuffin, L. Tancau, and R. Balakrishnan. Using deformations for browsing volumetric data. In *Proceedings of IEEE Visualization 2003*, pages 401–408, 2003.

[15] D. Patel, S. Bruckner, I. Viola, and M. E. Gröller. Seismic volume visualization for horizon extraction. In *Proceedings of the IEEE Pacific Visualization Symposium 2010*, pages 73–80, 2010.

[16] R. Pepper and G. Bejarano. Advances in seismic fault interpretation automation. *Search and Discovery Article 40170, Poster presentation at AAPG Annual Convention*, pages 19–22, 2005.

[17] C. Rezk-Salama, M. Scheuering, G. Soza, and G. Greiner. Fast volumetric deformation on general purpose hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, HWWS '01, pages 17–24, 2001.

[18] T. Ropinski, J. Meyer-Spradow, S. Diepenbrock, J. Mensmann, and K. H. Hinrichs. Interactive volume rendering with dynamic ambient occlusion and color bleeding. *Computer Graphics Forum (Eurographics 2008)*, 27(2):567–576, 2008.

[19] M. Ruiz, I. Viola, I. Boada, S. Bruckner, M. Feixas, and M. Sbert. Similarity-based exploded views. In *Proceedings of 8th International Symposium on Smart Graphics*, pages 154–165, 2008.

[20] Schlumberger Information Solutions. Petrel seismic to simulation software. http://www.slb.com/services/software/geo/petrel.aspx.

[21] F. Schulze, K. Bühler, and M. Hadwiger. Direct volume deformation. In *Computer Vision and Computer Graphics. Theory and Applications*, volume 21 of *Communications in Computer and Information Science*, pages 59–72. Springer Berlin Heidelberg, 2009.

[22] U.S. Energy Information Administration. International energy outlook 2010, 2010.

[23] R. Westermann and C. Rezk-Salama. Real-time volume deformations. *Computer Graphics Forum*, 20(3):443–451, 2001.