# A New Focus+Context Visualization Technique for Inspecting Black Oil Reservoir Models

Luiz Felipe Netto, Waldemar Celes

Tecgraf/PUC-Rio Institute
Computer Science Department
Pontifical Catholic University of Rio de Janeiro, Brazil
{netto,celes}@tecgraf.puc-rio.br

***Abstract*—In this paper, we propose a new visualization technique to inspect simulated black oil reservoir models. Numerical reservoir simulation is widely used in the oil & gas industry to predict and plan the exploration of petroleum fields. Of particular interest, it is crucial to understand the physical phenomena around injector and producer wells: how is the distribution of pressure, how oil/gas/water saturation varies over time, and others. Traditional visualization techniques only provide local insights, being hard to understand the three-dimension physical behavior. Another challenge is to handle today's massive models. We propose an efficient and effective focus+context visualization technique employing a cutaway approach, having the wells as objects of interest. We explore cone-shaped and box-shaped view-dependent cutting surfaces. We also allow the user to control the cutting surface aperture freely and to freeze the cut to gain motion parallax depth cues during model exploration. The proposed rendering algorithm runs on GPU, delivering real-time frame rate even for large reservoir models.**

## I. Introduction

Numerical black oil reservoir simulation is primarily employed in the oil & gas industry to plan and predict field exploration. The reservoir domain is discretized in a set of irregular hexahedral cells. These cells are organized in a topological grid with discontinuities, modeling geological faults. Initial properties such as permeability and porosity are set to these cells to characterize the reservoir. Injector and producer wells are also placed, and reservoir simulation reports the expected production and the future state of the domain regarding saturation and pressure distributions. Typically, a reservoir engineer tries different well arrangements and exploration strategies to maximize oil & gas recovery.

A simulated reservoir model inspection is crucial to understand the physical phenomena and better plan the exploration. Therefore, appropriate and innovative graphics tools to better analyze simulation results are still a necessity. The simulation outputs expected well productions and predicts how domain properties, such as pressure and oil, gas, and water saturation, evolve. Of particular interest, it is crucial to understand the reservoir behavior in the vicinity of wells.

Traditionally, cutting views are used to analyze the well regions. Figure 1a illustrates the classical fence diagram view. The user traces the fence near the wells of interest. Reservoir scalar fields are mapped onto the fence, allowing the engineer to visualize the well trajectories with the cutting fence. Another useful tool is illustrated in Figure 1b. Here, the cylindrical geometrical well representations are used as cutting surfaces. One can then observe scalar field distributions along well trajectories. Still, another helpful tool allows plotting dynamic profiles along well trajectories together with predicted production rate, as illustrated in Figure 1c. The user can observe how to scalar field evolves along well trajectories, helping to understate production variations.

However, all these techniques provide a limited understating of the physical behavior in the vicinity of wells; all these techniques only partially show the three-dimensional scalar field variation. To improve the interpretation of physical phenomena around wells, we propose a new focus+context visualization technique, cutting away the cells to reveal the well trajectories while preserving the scalar field mapping in their vicinities. Figure 1d illustrates the achieved results. The scalar field variation around the well is revealed from different points of view. Note how informative the visualization is; it immediately discloses the water reaching the frontmost well.

The main contributions of this work are:

- A cutaway visualization technique to inspect the vicinity of wells in simulated black oil reservoir models.
- A new efficient algorithm to render cone-shaped cutting surfaces based on a point location procedure [1], capable of handling objects of interest of any shape.
- A new box-shaped cutting surface construction algorithm using a reservoir layer as the bottom wall.
- An efficient view-independent cutting surface construction based on the relief mapping algorithm [2].

The rest of the paper is organized as follows. Section II reviews some techniques related to this work. The following three sections describe the proposed visualization technique and detail the rendering algorithm to accomplish it. In Section III, we present an overview of the proposed algorithm. Section IV describes the way view-dependent cutaways are achieved, including box-shaped cutaways. Section V explains how we built view-independent cutaways. Section VI presents an analysis of performance and a discussion of achieved visual results. In Section VII, we draw conclusions about the proposed technique.

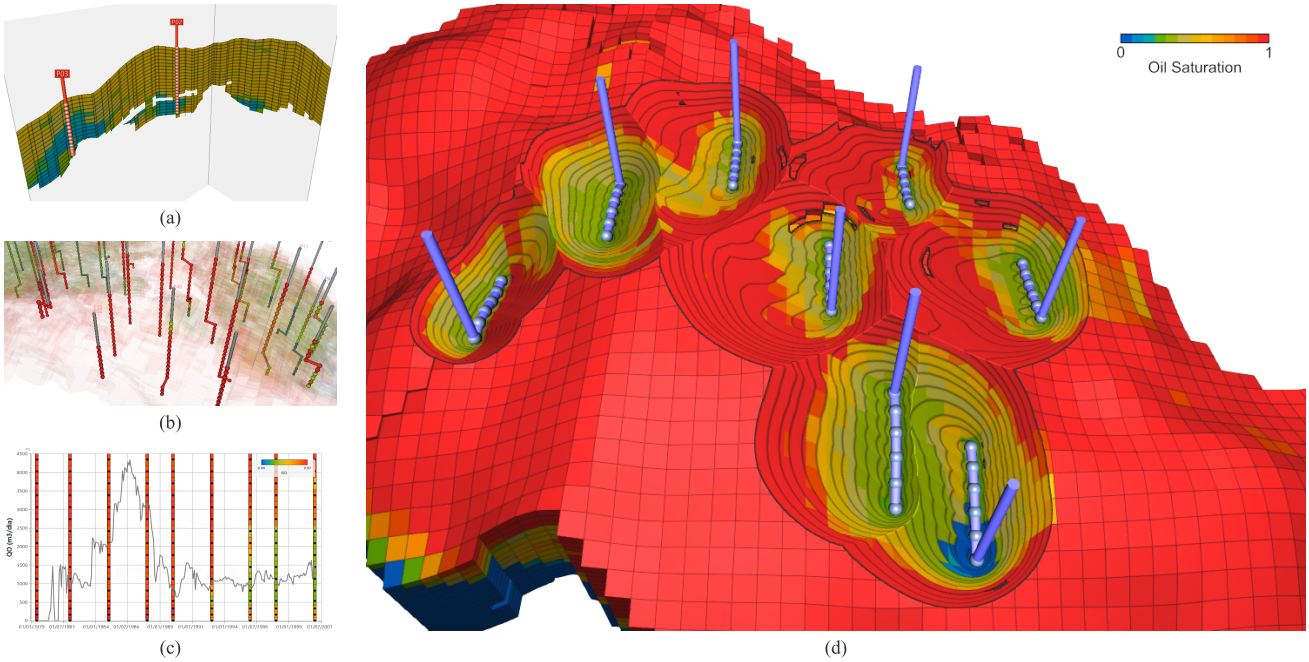



Fig. 1. A comparison among traditional visualization techniques and the proposed one to inspect black oil reservoir model in the vicinity of wells: (a) fence diagram in the proximity of wells; (b) reservoir scalar field mapped along well trajectories; (c) numerical profile along well trajectories over simulation time; (d) proposed focus+context technique.

## II. Related Works

Different focus+context visualization techniques have been proposed. These visualizations are often inspired by scientific and technical illustrations, where some classic examples include Lenses [3], Exploded Views [4], and Cutaways [5], [6]. We have drawn our attention to cutaway visualization, which provides a way to select parts of the model that are of greater interest to the user, and reveals it by removing other parts that obstruct its view while preserving the context.

Feiner and Seligmann [5] introduced cutaway and ghosting algorithms on a z-buffer based system to cut models and reveal the objects of interest. Diepstraten et al. [7] developed an interactive rendering pipeline to generate cutaway illustrations automatically while maintaining interactivity. The cutaways were classified as *break-away*, a narrow cut that reveals the object of interest, and *cutout*, a broader cut in front of the object of interest that reveals more context. In both cases, a set of rules is derived for placing the cuts automatically. Automatic view-dependent cutaway for volume data was addressed by Viola et al. [8], using two different approaches: Maximum Importance Projection and Average Importance Compositing, requiring that the volume data to be previously segmented with importance associated with it. Bruckner and Groller [9] made an interactive direct volume rendering application to generate static illustrations, with the combination of cutaway and ghosting, which resemble those found in technical books. Coffin and Hollerer [10] advocated the use of cutaway as an x-ray tool to see through 3D geometric objects. The user defines the cutaway shape, and the rendering is performed cutting through the object with CSG operations using the stencil buffer.

Li et al. [6] reviewed and analyzed the shape and position of cutaways in traditional illustration literature. They observed the use of simple cutting shape geometries to obtain the cutaway, like boxes or tubes, and developed an automated system to explore and create cutaway illustrations from complex geometric models, where the user choose the cutting shape and define the object of interest. Sigg et al. [11] generated view-dependent cutaway with a box or sphere shape by only requiring the specification of importance to the geometry or volume data, arguing that less interaction is better when placing an object to cut the model.

View-dependent cutaway of complex and dynamic polygonal scenes was performed in interactive frame rates by Burns and Finkelstein [12]. They represented the cutting surface as a depth map computed in a modified distance transform in GPU using the Jump Flooding Algorithm [13], where the depth footprint from objects of interest are taken as seeds. The final model is rendered while discarding fragments that are between the camera and the cutting surface. This approach has been applied to compute the cutting surface from blood-flow vessels [14] and to clip objects from molecular models [15]. A 2D distance field has also been used to obtain a terrain cutaway to visualize sewer networks [16].

Lidal et al. [17] derived a set of design principles applied to geological models to obtain cutaway visualizations. Their geological models consist of layers represented by surfaces, where it is possible to find sand-bodies, water, oil, gas, and structural bodies. By following these principles, they produced

a frustum-shaped cutaway, revealing the feature in focus in a view-dependent manner. Furthermore, their proposal allows to freeze the cutaway view and observe it from a different position, gaining motion parallax depth cues in the process. Finally, they limited the cutaway shape state to a fixed camera threshold, allowing the users to see the objects always in focus.

To our knowledge, cutaway applied to reservoir models was first reported by Martins et al. [18], where the objects of interest, reservoir cells, are chosen by the user. To reveal the selected cells, they did a simple ray cast procedure. If the rays intersect non-chosen cells, the cells are removed from the view. In a more elaborated work, de Carvalho et al. [19] took the design choices of previous works [6], [11], [17], and extended them to reservoir models. The union of all frustums formed by the bounding boxes of selected cells forms the cutting surface. Their proposal also allows the user to freeze the cutaway generation to visualize it from other positions and applies screen-space effects, such as ambient occlusion, to enhance the context. The main drawback of their work is the requirement to render all the reservoir cells to apply the cut. For large models with millions of cells, the rendering of all cells may severely impact performance. Also, the union of frustums may not suit the focus on wells, especially those with curved trajectories.

Our proposed visualization technique builds upon previous works to provide a new efficient cutaway algorithm for reservoir model visualization. We explore the GPU capabilities, which allow the proposed algorithm to render large reservoir models while keeping in mind that a critical component in the reservoir analysis is the interaction, exploration, and inspection of the model by experts.

We choose to primarily compute the cutaway surface as done by Burns et al. [12]; it fits tightly the objects of interest, mainly producer and injector wells. We also explored the use of box-shaped cutting surfaces. Similar to Lidal et al. [17], we also allow the user to freeze the cutaway generation, although we do not change the cutaway even if the objects of interest are occluded. In our case, we want to emphasize the liberty to explore the model since the context surrounding the objects of interest is most important. To avoid the rendering of all reservoir cells, we use the point location algorithm as proposed by Franceschin et al. [1]. Besides the cutting surface, we draw the reservoir hull, composed of external cell faces.

## III. Algorithm Overview

Today's reservoir models reach millions of active cells; therefore, an efficient reservoir model rendering algorithm is crucial. Rendering all the active cells of a reservoir model can be unnecessarily expensive. For instance, for an external point of view, it suffices to render the model hull represented by the external faces of the model. External faces are present at the model's boundary and at the adjacency of inactive cells. Of particular interest is the analysis of the model around injector and producer wells.

There are two main types of reservoir well geometries: vertical and horizontal. Vertical wells present straight vertical trajectories while horizontal ones start vertical and then bend approximately $90^\circ$ to increase reservoir-well contact, thereby improving reservoir productivity. The proposed visualization technique aims to inspect the scalar field along well trajectories, despite their shapes. The user can interactively change the selected wells (objects of interest), control the cutaway aperture, and manipulate the model to analyze the property (scalar field) variation.

To render the cutting surface, we employ the point location algorithm proposed by Franceschin et al. [1], which requires the storage of the reservoir model in the GPU. Such representation is sent to the GPU in a pre-processing stage. In the rendering stage, the proposed visualization algorithm performs the following steps:

1) The cutting surface is computed as a depth map based on the depth footprint of the objects of interest with the Jump Flooding Algorithm [12].
2) The intersection map is computed employing the efficient point location algorithm presented in [1].
3) The rendering is performed by two steps: the external reservoir faces are rendered, being cut away by the cutting surface; the intersection map is used to render the cutaway surface.
4) Finally, the objects of interest are rendered, being entirely revealed.

Figure 2 illustrates the algorithm overview. In the following sections, we describe the algorithm in detail.

## IV. View-Dependent Cutaway

We first describe the view-dependent cutaway. In this scenario, the cutting surface always faces the observer. At each frame, a cutting surface is computed and rendered.

### A. Cutting surface computation

To compute a cone-shaped cutting surface, we follow the work of Burns and Finkelstein [12]. Given a set of objects of interest, we draw their back-faces storing their depth values in a separate texture memory (Figure 2b). Using this texture as input, we compute the cutting surface applying the Jump Flooding Algorithm (JFA) [13]. We proceed with JFA in an interleaved read and write manner with two texture buffers, where the writing is done through render-to-texture. The distance function used in the JFA is defined by:

$$d(p) = \max_{q \in I} \left( z(p) - m(p) \left\| q - p \right\| \right) \tag{1}$$

where $p$ is the position of the current fragment, $z(p)$ is its associated depth, and $q$ is the position of the neighbor fragment being evaluated. The parameter $m(p)$ is a perspective compensation factor [12] to maintain the cutaway surface aperture $\theta$ constant under perspective projection, which is defined by:

$$m(p) = \frac{-(PM_{sz} + z(p))}{\tan(\theta)} \tag{2}$$

with $PM_{sz}$ being the z-scaling factor of the projection matrix. The JFA is finished after $\log n$ passes, where $n$ is the greater

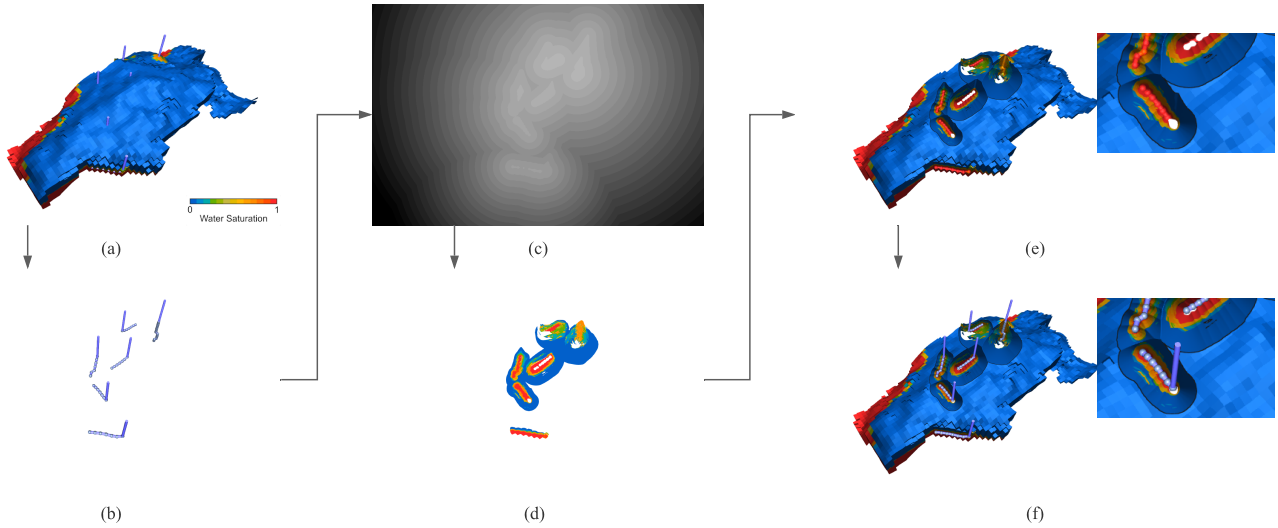



Fig. 2. Overview of proposed algorithm: (a) conventional external view of reservoir model; (b) back-face rendering of objects of interest on the depth buffer; (c) depth map built using the JFA algorithm; (d) colored intersection map according to the scalar field of intersected reservoir cells; (e) drawing of reservoir hull cutaway by the cutting surface; (f) drawing of objects of interest (reservoir wells).

dimension of the framebuffer. As a result, we obtain a cone-shaped view-dependent cutting surface represented as a *depth map* with the size of the assigned framebuffer, as illustrated in Figure 2c.

### B. Cutting surface intersection

Given the computed depth map, the surface that intersects the reservoir volume can be obtained. A quadrilateral with the size of the framebuffer is drawn writing to textures. For each $p_c = (x, y, z_c)$, where $p_c$ is the position of the cutting surface's fragment in screen space ($x$ and $y$ are screen coordinates, and $z_c$ is the associated depth), we transform it to the object space to obtain $p_o$. We evaluate the intersection of $p_o$ with the reservoir model using the point location algorithm as proposed in [1]. If the intersection occurs, i.e., $p_o$ is inside a cell of the reservoir, we output the position $p_o$ and the associated property color of the cell. The result is called the *intersection map*, as illustrated in Figure 2d.

### C. Cutaway reservoir rendering

The cutaway reservoir model is rendered with the following approach in a fragment shader. First, for each fragment of the model hull, its depth is compared against the depth of the computed cutting surface. If the fragment depth is greater than the cutting surface depth (i.e., is behind), the model's position, normal, and color are assigned to the fragment. Otherwise, we access the intersection map generated in the previous pass to check if the fragment is part of the cutting surface. In this case, the computed cutaway surface intersection position and color from the previous pass are assigned to the fragment. The fragment is discarded if it is not part of the cutting surface (the cutting surface may present holes due to inactive cells inside the model). The result is illustrated in Figure 2e.

### D. Shading

Two light sources are used to enhance the visual perception of the cutaway surface. We set one at the camera position and one high above the model. The fragments that lay on the cutaway surface have normals derived by looking up the neighbor's position on the depth map. Fragments along the intersection of the reservoir surface and the cutting surface are rendered black to enhance perception. The wells (objects of interest) are drawn in the last phase, ultimately revealed in the final image, as illustrated in Figure 2f.

### E. Point location threshold

The cells of a reservoir model are non-regular hexahedra, being quite common to present non-planar faces. The point location algorithm [1] correctly considers such non-planarities; however, when drawing the reservoir hull, the surface is represented by a triangle mesh, leading to differences between the clipped cell face and the cutting surface border.

To solve this issue, we employ a threshold. We modified the point location algorithm to return the closest cell for a point outside the model. We then compute the distance from the point to this cell; if the distance is less than a pre-defined threshold $\alpha$, the point is considered inside the closest cell. It works as we had inflated the cutting surface.

### F. Wireframe drawing

For inspecting the reservoir model, it is sometimes essential to render the mesh wireframe. The engineers need to evaluate the mesh quality for a better understanding of the simulation results. However, because the cutting surfaces are curved and oblique, drawing the cell's borders results in a polluted image. Therefore, we have opted to modify the original wireframe drawing, as described in [1], to only preserve the drawing of lines representing borders of geological layers; that is, only

the lines corresponding to the top and bottom faces of each cell are drawn. The achieved result is illustrated in Figure 1d.

### G. Discussion

The texture used to store the cutting surface depth map (and the corresponding cutting surface intersection map) can have dimensions different from the visualization screen. The use of maps with smaller resolutions may speed up the time needed to render each frame, slightly impacting the image quality at the surface ridges.

Also, for the view-dependent cutting surface, explicitly computing the intersection map is unnecessary. The surface color could be computed while drawing the reservoir hull: for each fragment with depth less than the map depth, we could project it on the cutting surface and then check its intersection with the reservoir cells, assigning the appropriate scalar field color. We have opted for explicitly computing the intersection map because this speeds up the computation for view-independent cutaway, as we shall discuss.

### H. Box-shaped cutaway

Lidal et al. [17] suggested the use of box-shaped cutaways for geological models. The reason is to maintain the natural alignment of such models, where the horizontal surfaces represent natural geological layers.

To achieve this goal, we substitute the depth map built by the Jump Flooding Algorithm (JFA) with a depth map obtained by drawing a box-shaped cutting surface. Given the object of interest, we compute a view-dependent, aligned bounding box in the following manner:

- The vertical back wall is always facing the viewer.
- The horizontal reservoir layer, immediately below the object of interest, defines the bottom wall.
- A user-defined aperture angle controls the side walls.

As we want the bottom wall to be defined by a model layer, we incorporate the faces of such a layer as part of the external model hull. The result is illustrated in Figure 3a.

## V. View-independent cutaway

Lidal et al. [17] advise using an oblique viewpoint to observe the cutaway region. Although a cutaway region facing the viewer better reveals the objects of interest, it does not enhance depth perception. To overcome such a drawback, we allow the user to freeze the cutting surface; a strategy also observed in [19]. Thus, we call it a view-independent cutaway.

Anytime, the user can freeze the cutting surface (despite it is a cone or box-shaped cut) and move the camera. Subsequent frames are rendered with the following modifications in the initial algorithm:

1) The depth and intersection maps are no longer computed at each frame. Instead, the maps from the last view-dependent cutaway frame are used.
2) The model-view and projection matrices used to compute the last view-dependent cutaway are stored; we call it the *reference camera*.
3) When the reservoir is rendered, every fragment is transformed to the reference camera to compare its depth with the one stored in the depth map.
4) We render the cutaway surface approximating its position by ray casting into the depth map.

Steps 3 and 4 are described here in details. Let $p_w$ be the position of a reservoir fragment in the current screen space during the rendering of the reservoir model hull. Before writing the fragment to the framebuffer, we need to know if it is discarded by the cutting surface. The position $p_w$ is transformed to the cutting surface system with the following procedures:

- $p_w$ is mapped to the current clip space:

$$p_w \rightarrow p_{clip} \tag{3}$$

- $p_{clip}$ is transformed to the reference clip space:

$$p_{rclip} = \big(M_{rp} \times M_{rv}\big)\big(M_v^{-1} \times M_p^{-1}\big)p_{clip} \tag{4}$$

  where $M_v$, $M_p$, $M_{rv}$, and $M_{rp}$, are the model-view and projection matrices from the current camera and the reference camera, respectively.
- If $p_{rclip}$ components are enclosed in the interval $[-1, 1]$, we map it to the reference screen space:

$$p_{rclip} \rightarrow p_{rw} = (x, y, z) \tag{5}$$

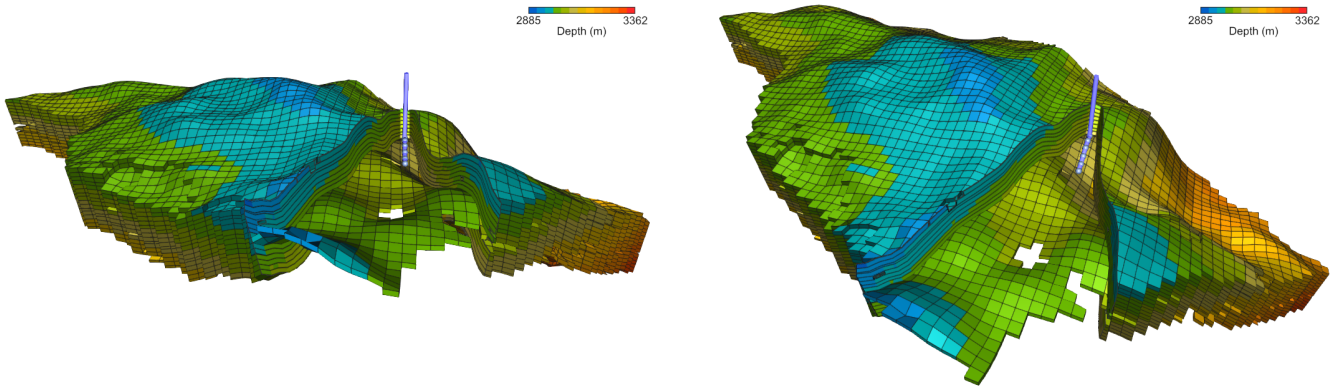
- With the computed $(x, y)$, the cutting surface depth map is sampled to obtain $z_c$. If $z > z_c$, the reservoir fragment is drawn; otherwise, we compute the intersection of the viewing ray with the cutting surface to obtain its color, accessing the intersection map generated with the last view-dependent cutaway.

To compute the intersection of the viewing ray with the cutting surface, the straight projection of the fragment position on the surface, $p_{rw}$, is used as the entry point to the employed relief map algorithm [2], considering the reference depth map. The computation of the ray-surface intersection is challenging because our proposal handles an arbitrary number of objects of interest with arbitrary shapes; therefore, the resulting cutting surface is concave and complex. If a valid point of intersection is found, the intersection map is sampled to retrieve the fragment color. Finally, the normal vector associated with the fragment is computed in the reference camera system and transformed back to the viewing camera. The achieved result is illustrated in Figure 3b.

## VI. Results

We run a set of computational experiments to evaluate the proposed technique. All the experiments were run on a computer with a 2.90GHz Intel i5-9400F processor, 16GB RAM, and the NVIDIA GeForce RTX 2060 graphic card with 6GB of dedicated memory. The proposed visualization was implemented in C++, OpenGL, and GLSL. We use five actual reservoir models, identified here from $A$ to $E$[1]. Table I shows the characteristics of these models.

[1]Model E is a refined version of model A




(a) (b)

Fig. 3. Box-shaped cutting surface using the reservoir layer as bottom wall: (a) view-dependent; (b) view-independent.

TABLE I
ACTUAL RESERVOIR MODELS USED IN THE COMPUTATIONAL EXPERIMENTS.

| Model | # active cells | # triangles of hull |
|---|---|---|
| A | 31,498 | 36,240 |
| B | 36,425 | 47,216 |
| C | 338,453 | 711,644 |
| D | 6,245,219 | 635,012 |
| E | 16,126,976 | 1,400,832 |

TABLE II
PERFORMANCE, EXPRESSED IN FRAMES PER SECOND (FPS), FOR RENDERING DIFFERENT FLAVORS OF CUTTING SURFACE FOR DIFFERENT RESERVOIR MODELS.

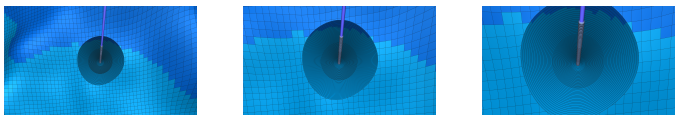
| Model | View-dependent | | View-independent | |
|---|---|---|---|---|
| | cone-shaped | box-shaped | cone-shaped | box-shaped |
| A | 148 | 541 | 952 | 947 |
| B | 142 | 387 | 790 | 671 |
| C | 100 | 229 | 341 | 386 |
| D | 108 | 153 | 252 | 156 |
| E | 81 | 87 | 215 | 200 |

We first conduct an experiment to evaluate the general performance of the proposed rendering algorithm. We specify wells of interest for each model and applied the different flavors of cuts supported by our approach. Table II reveals the achieved frame rate. In all the experiments, the performance was measured with a fullscreen resolution of $1920 \times 1080$.

Note that the construction of the depth map is expensive, even though the JFA delivers real-time rendering. With the box-shaped cutting surface, the performance is increased. This performance improvement also increases for the view-independent cutaway because we re-use the depth and intersecting maps. The ray intersection procedure based on the relief mapping algorithm is fast and does not compromise performance.

A critical aspect of our proposal is the memory requirement and performance related to the point location algorithm employed [1]. In this technique, the entire reservoir model is compactly stored in the GPU. In our experiments, the largest model (model E) takes a total of $584.5$ MB of GPU memory. A regular grid accelerates the point location query in the fragment shader; however, it runs slower when the model has more inactive cells and discontinuities. That is the case with model C, which has a lower performance than model D in the view-dependent mode, as shown in Table II, despite the significant difference in cell count (Table I). As expected, model C performs better when the computation of the intersection map is not required. We kindly suggest the reader check out the work from Franceschin et al. [1] for a more detailed discussion.

We then examined model D, currently a typical large model employed in the industry, and run a detailed performance analysis, measuring the time take by each phase of the algorithm. We tried different sizes of the cutting surface because its rendering cost is directly proportional to the number of pixels it occupies on the screen. Table III shows the achieved result and Figure 4 shows examples of cutaway surfaces that cover $8\%, 22\%$, and $51\%$ of the screen, respectively.



(a) (b) (c)

Fig. 4. Examples of cutting surfaces used to evaluate performance in Table III with different ratio (screen size by pixel count): (a) $8\%$, (b) $22\%$, and (c) $51\%$.

We observe that the construction of the depth map is the most expensive phase. Note also that its construction is not affected by the actual size of the cutting surface because the map is always built for the entire screen since we do not know in advance its actual intersection with the reservoir model. On the other hand, as expected, the construction of the intersection map does vary with the actual cutting surface size.

TABLE III
PERFORMANCE FOR VIEW-DEPENDENT CUTAWAYS, MEASURED IN MILLISECONDS (MS), FOR EACH PHASE OF THE ALGORITHM, CONSIDERING DIFFERENT SIZES OF THE CUTTING SURFACE.

| Cutting Surface Pixel Count | 95105 | 174497 | 452588 | 666834 | 1062888 | 1407543 | 1965969 |
|---|---|---|---|---|---|---|---|
| Ratio (Screensize/PixelCount) | 5% | 8% | 22% | 32% | 51% | 68% | 95% |
| Depth Map (ms) | 5.24 | 5.24 | 5.25 | 5.28 | 5.31 | 5.40 | 5.43 |
| Intersection Map (ms) | 1.00 | 1.38 | 2.52 | 3.13 | 4.00 | 4.47 | 4.83 |
| Reservoir Drawing (ms) | 1.61 | 1.60 | 1.61 | 1.63 | 1.71 | 1.72 | 1.75 |
| **Total (ms)** | 7.86 | 8.21 | 9.38 | 10.05 | 11.01 | 11.59 | 12.02 |

Although all fragments of the covering quadrilateral are tested for intersection with the reservoir model, distant fragments are quickly discarded from the intersection by the efficient point location algorithm proposed by Franceschin et al. [1], while close fragments require the complete intersection computation. The reservoir drawing is only slightly affected by the size of the cutting surface.

To summarize the features of the proposed technique, Figure 5 illustrates different configurations to inspect a selected well. Note the flexibility to choose between cone and box-shaped cutting surface and the ability to freeze the cut to gain motion parallax depth insights. Furthermore, as illustrated in Figure 1d, we are not limited to select only one well.

Although our initial goal was to present a visualization technique to inspect the vicinity of wells, due to the varying shape of the wells, especially the horizontal ones, we ended up with a more general visualization technique. Our proposal handle objects of interest with any shape. The use of JFA gives this flexibility. We only need to input the JFA with the back-face depth-values of the desired objects of interest to compute the depth map. We explore this flexibility to define a block of cells as the object of interest, as in [19]. Figure 6 shows a block of cells as the object of interest in a view-independent cutaway. The cells were selected by defining a sub-region with topological range coordinates $[i_0 : i_1, j_0 : j_1, k_0 : k_1]$.

### A. Limitations

Because the depth and intersection maps are built in image space, the proposed procedure presents the following limitations:

- The objects of interest must be inside the viewing frustum.
- The view-independent cutting surface quality depends on the view-dependent map resolution. Activating the view-independent mode using a camera far away from the objects of interest, and getting close to it afterward, will result in a blurred cut.
- If view-independent mode is activated when a view-dependent cutting surface exceeds the screen limits, it will result in a cutting surface that ends abruptly.

## VII. CONCLUSION

In this paper, we propose a new visualization technique to inspect black oil reservoir simulation results. Of particular interest, we aim to achieve an adequate technique to review the physical phenomena around injector and producer wells over the simulation. Traditional visualization techniques only provide local information, being hard to understand the three-dimensional physical behavior. Our proposal employs the cutaway approach allowing the user to observe the well trajectories while viewing the scalar field variation in its vicinity. We propose a general method capable of handling any shape of objects of interest, using the Jump Flood Algorithm (JFA). We also explored an engaging box-shaped cutting surface, using the underneath reservoir layer as the bottom wall. Our technique allows the user to freeze the cut to take advantage of motion parallax while interacting with the model.

As future work, we plan to control the reference camera from the view-independent cutaway mode, modifying the cutting surface without leaving the current view. We also plan to explore other objects of interest, such as a set of streamlines drawn inside the model according to the fluid flow velocity.

## ACKNOWLEDGMENT

Tecgraf/PUC-Rio is a research institute mainly funded by Petrobras, the Brazilian oil company.

## REFERENCES

[1] B. Franceschin, F. Abraham, L. F. Netto, and W. Celes, "Gpu-based rendering of arbitrarily complex cutting surfaces for black oil reservoir models," in *2019 32nd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, 2019, pp. 131–138.

[2] F. Policarpo, M. M. Oliveira, and J. a. L. D. Comba, "Real-time relief mapping on arbitrary polygonal surfaces," in *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, ser. I3D '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 155–162. [Online]. Available: https://doi.org/10.1145/1053427.1053453

[3] C. Tominski, S. Gladisch, U. Kister, R. Dachselt, and H. Schumann, "Interactive lenses for visualization: An extended survey," *Computer Graphics Forum*, vol. 36, no. 6, pp. 173–200, 2017.

[4] S. Bruckner and M. E. Groller, "Exploded views for volume data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1077–1084, Sep. 2006.

[5] S. K. Feiner and D. D. Seligmann, "Cutaways and ghosting: satisfying visibility constraints in dynamic 3d illustrations," *The Visual Computer*, vol. 8, no. 5, pp. 292–302, Sep 1992.

[6] W. Li, L. Ritter, M. Agrawala, B. Curless, and D. Salesin, "Interactive cutaway illustrations of complex 3d models," *ACM Trans. Graph.*, vol. 26, no. 3, Jul. 2007.

[7] J. Diepstraten, D. Weiskopf, and T. Ertl, "Interactive cutaway illustrations," *Computer Graphics Forum*, vol. 22, no. 3, pp. 523–532, 2003.

[8] I. Viola, A. Kanitsar, and M. E. Groller, "Importance-driven volume rendering," in *IEEE Visualization 2004*, Oct 2004, pp. 139–145.

[9] S. Bruckner and M. E. Groller, "Volumeshop: an interactive system for direct volume illustration," in *VIS 05. IEEE Visualization, 2005.*, Oct 2005, pp. 671–678.

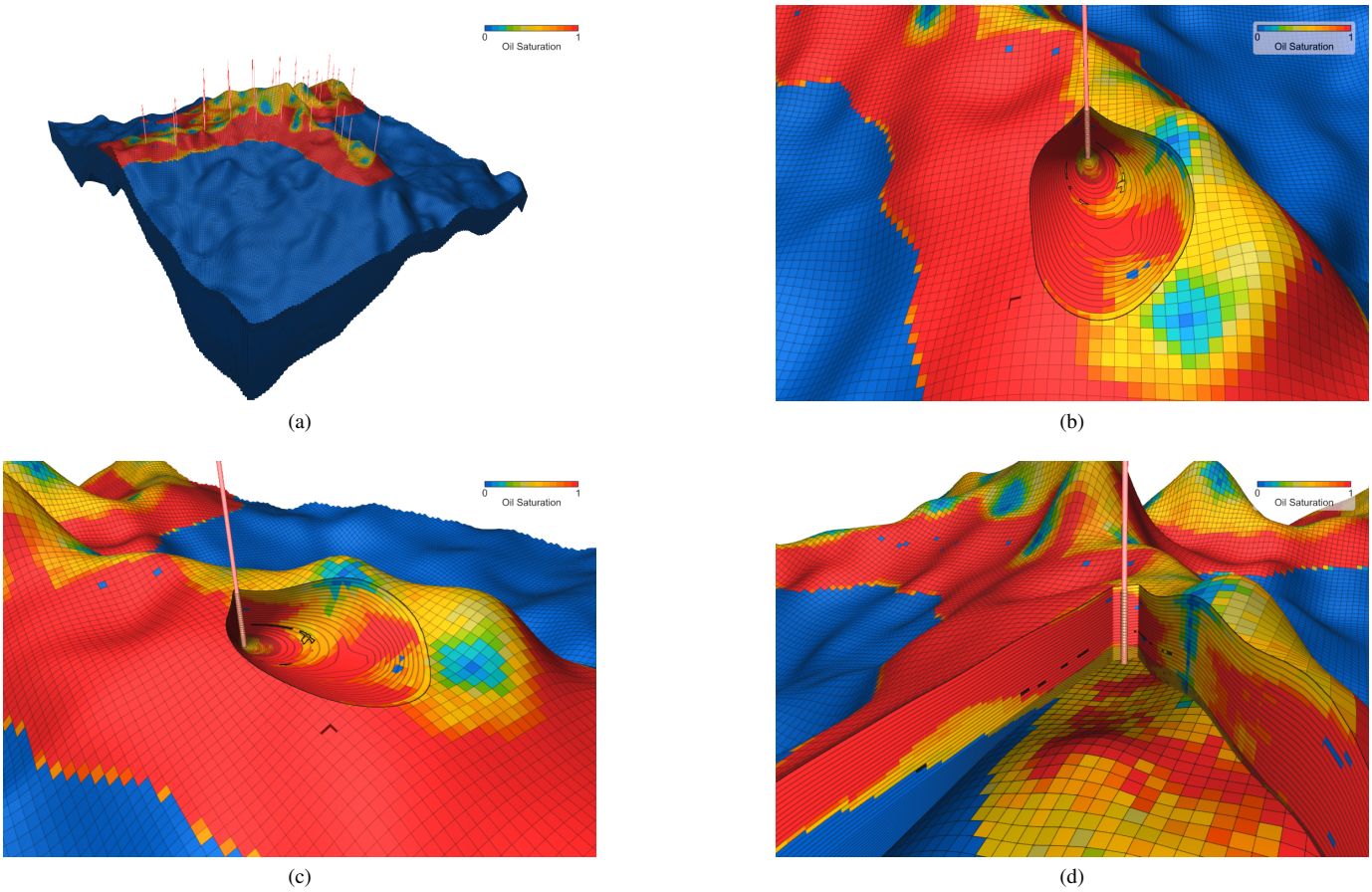



Fig. 5. The reservoir model D with 6 million active cells (a). View-dependent cutaway (b) applied to a selected well, and the same cutaway from a different camera position rendered in view-independent mode (c). A box-shaped cutaway (d) applied to the selected well. Reservoir wireframe is enabled with low intensity due the high density of cells, and horizontal line drawing is enabled in the cutting surface.

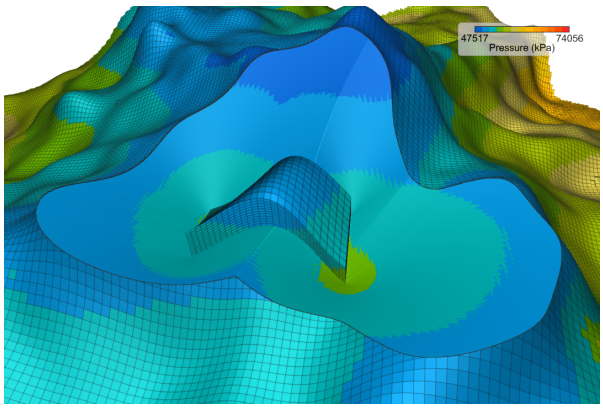



Fig. 6. A view-independent cutaway applied to the reservoir model D using a block of cells as the object of interest.


[10] C. Coffin and T. Hollerer, "Interactive perspective cut-away views for general 3d scenes," in *3D User Interfaces (3DUI'06)*, March 2006, pp. 25–28.

[11] S. Sigg, R. Fuchs, R. Carnecky, and R. Peikert, "Intelligent cutaway illustrations," in *2012 IEEE Pacific Visualization Symposium*, Feb 2012, pp. 185–192.

[12] M. Burns and A. Finkelstein, "Adaptive cutaways for comprehensible rendering of polygonal scenes," *ACM Trans. Graph.*, vol. 27, no. 5, pp. 154:1–154:7, Dec. 2008.

[13] G. Rong and T.-S. Tan, "Jump flooding in gpu with applications to voronoi diagram and distance transform," in *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, ser. I3D '06. New York, NY, USA: ACM, 2006, pp. 109–116.

[14] K. Lawonn, S. Glaßer, A. Vilanova, B. Preim, and T. Isenberg, "Occlusion-free blood flow animation with wall thickness visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 728–737, Jan 2016.

[15] M. Le Muzic, P. Mindek, J. Sorger, L. Autin, D. S. Goodsell, and I. Viola, "Visibility equalizer cutaway visualization of mesoscopic biological models," *Computer Graphics Forum*, vol. 35, no. 3, pp. 161–170, 2016.

[16] A. Konev, M. Matusich, I. Viola, H. Schulze, D. Cornel, and J. Waser, "Fast cutaway visualization of sub-terrain tubular networks," *Computers & Graphics*, vol. 75, pp. 25 – 35, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0097849318301031

[17] E. M. Lidal, H. Hauser, and I. Viola, "Design principles for cutaway visualization of geological models," in *Proceedings of the 28th Spring Conference on Computer Graphics*, ser. SCCG '12. New York, NY, USA: ACM, 2012, pp. 47–54.

[18] Z. Martins Filho, E. V. Brazil, M. C. Sousa, F. D. Carvalho, and R. Marroquim, "Cutaway Applied to Corner Point Models," in *Workshop on Industry Applications (WGARI) in SIBGRAPI 2012 (XXV Conference on Graphics, Patterns and Images)*, S. J. F. G. A. V. Saúde, Ed., Ouro Preto, MG, Brazil, august 2012. [Online]. Available: http://www.decom.ufop.br/sibgrapi2012/index.php/call/wgari

[19] F. M. de Carvalho, E. V. Brazil, R. G. Marroquim, M. C. Sousa, and A. Oliveira, "Interactive cutaways of oil reservoirs," *Graphical Models*, vol. 84, pp. 1 – 14, 2016.