

GPU accelerated Boolean operations for complex 3D GIS queries

Tobias Frank, Pierre Kedzierski

Gocad Research Group, ASGA, ENSG-INPL/CRPG-CRNS, 54501 Vandoeuvre-les-Nancy – France,
tobias.frank@gocad.org

Today's 3D GIS tools work on an explicit definition of the model to be examined. Therefore, triangulated surfaces or structured volumetric grids (e.g. Cartesian or stratigraphic grids) are state-of-the-art. This article introduces a new method to perform Boolean operations for complex 3D GIS queries on the Graphics Processing Unit (GPU) in real-time. Geological attributes are defined by implicit functions on unstructured tetrahedral meshes with adaptive cell size. Discontinuities in the tetrahedral mesh allow to model geological unconformities like faults.

Each value of an attribute defines an iso-value surface. Considering several iso-value surfaces of different attributes, the introduced method computes Boolean operations known from constructive solid geometry by combining clipping functions. The clipping functions are defined by texture maps and their combination is performed by the texture blending capabilities of modern graphics hardware. The results of these operations can be further combined by queries involving Euclidean distance transforms to geological interfaces (e.g. faults and horizons) or artifacts like wells. The complexity of the introduced method is constant for any model size and complexity and depends only on the screen size of the 3D camera.

1 Introduction

Computer Aided Design (CAD) plays an outstanding role for industrial modeling, design and development. Today, three classes of mathematical model representation can be found: parametric, implicit and point sets. Each category has its special field of application. Parametric representation like splines or triangle surfaces is widely used for classical CAD construction applications. Implicit methods enable efficient computations of intersections and penetration of solids and are widely used to interpolate physical measurements on 3D meshes. As the fragment size of modern GPUs is getting smaller than the point size of dense data sets the point set techniques are getting more and more popular. Today, three dimensional representation is state of the art and implies its own problems like creating and modeling the 3D model. In this work geological models are based on implicit representation (BLOOMENTHAL & WYVILL 1997). Hereby geological attributes are defined as implicit functions. COWAN *et. al.* (2003) use Radial Basis Functions to interpolate geological attributes from scattered samples. Radial Basis Functions cannot model discontinuities. Thus, geological interfaces like faults cannot be considered by this type of implicit model. The approach developed in this paper is based on unstructured tetrahedral meshes. The implicit functions are defined on the

nodes of the mesh and geological discontinuities are represented by discontinuities in the tetrahedral mesh (FRANK 2006). Further, this type of implicit representation allows quantitative estimations (ROYER 2005). In this paper we illustrate Boolean operations of Constructive Solid Geometry performed on the Graphics Processing Unit (GPU). In a first section the basic mode of operation of a GPU is explained.

2 A very short trip down the OpenGL graphics pipeline

Figure 1 shows a schematic view of a programmable OpenGL graphics pipeline. The OpenGL commands enter the pipeline from the left side. These commands define both the model and the view. The result of the graphic pipeline is written into the frame buffer.

- **Evaluator:** The Evaluator stage can be used to approximate curves or surfaces by computing polynomial functions over the input data. This feature is not used by our implementation and is not further described.
- **Vertex unit:** The vertex unit operates on geometrical primitives like points, line segments or polygons. In this unit the vertices are transformed, then lighting operations are performed if requested. A user-defined vertex program can be used to add custom func-

tionality to this stage. Finally, the primitives are clipped to the viewing volume.

- **Rasterizer:** The rasterizer generates a two-dimensional, discrete representation of the points, line segments and polygons. The result is a set of fragments and for each fragment a frame buffer address, a color value and texture addresses are computed.
- **Fragment unit:** The fragment unit operates on every single fragment generated by the rasterizer. In this stage the fragment value (color and opacity) can be altered by blending, conditional and logical operations and many more. Again this stage can be customized by a user-defined fragment shader. The result of the fragment unit is stored in the frame buffer.

This work makes use of vertex shaders to generate and transform texture coordinates for multitexturing applications. Adapted blending functions for geological applications can be implemented by fragment shaders. These shader programs are written using the CG shading language. A detailed description of CG can be found in (FERNANDO & KILGARD 2003; MARK *et al.* 2003).

3 Multitexturing principles

Recent graphics hardware supports multiple texture maps that can be combined in a single rendering pass. Multitexturing opens the door to a wide variety of applications for geological modeling. Distance maps are a widely used tool to determine the distances to faults or wells inside a 3d-model (see figure 3a). Also complex problems of CSG (Constructive Solid Geometry) like intersection and penetration of arbitrary bodies or surfaces can be solved by texture-based techniques.

To co-visualize several attributes, the colors defining these attributes have to be blended for each fragment. This operation is performed on the fragment unit of the OpenGL graphics pipeline (see figure 1). The following sections propose two solutions. The first solution uses the built-in OpenGL blending functions together with a multitexture environment. The second solution defines advanced blending functions using fragment shaders.

3.1 Multitexturing using OpenGL texture units

The blending of several textures can be performed by applying the texture operations sequentially for each texture on every fragment. An

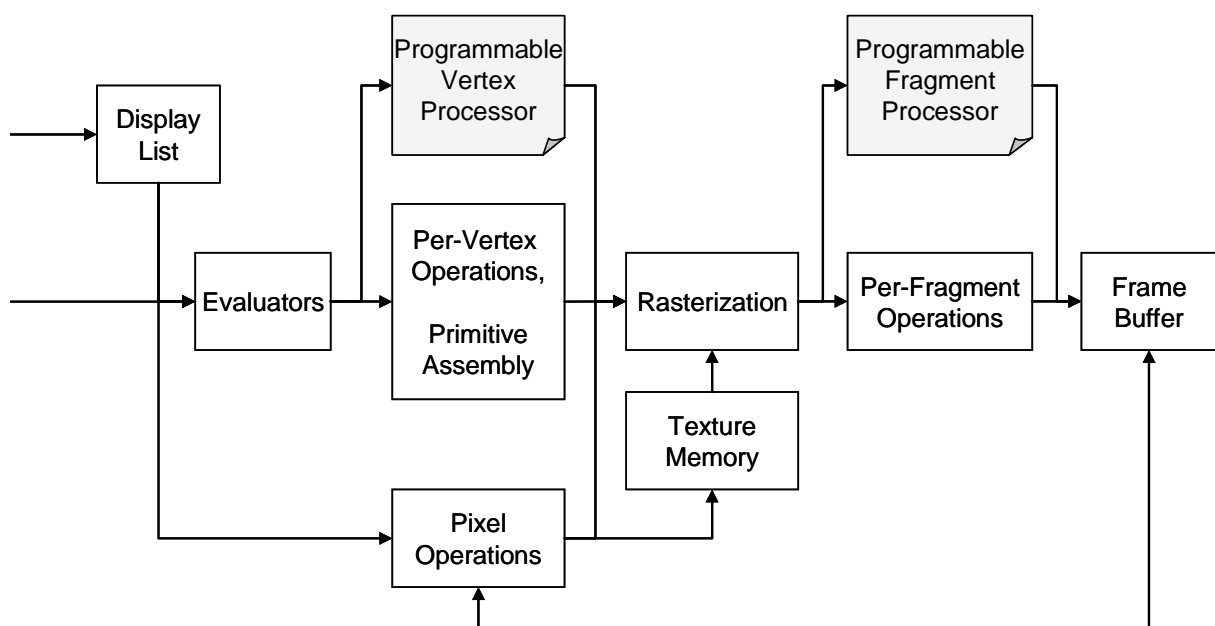


Fig. 1: This figure illustrates a programmable OpenGL graphics pipeline. OpenGL commands enter the pipeline from the left side. These commands describe both the model and the parameters needed to create the view. The vertex and fragment unit can be customized by user-defined programs (vertex and fragment shaders). The result is written into the frame buffer (SEGAL & AKELEY 2003).

OpenGL texture unit (TU_i) takes the current fragment color C_f and texture lookup color CT_i to compute the fragment output color C'_f . The initial fragment color C_f entering the first texture unit TU_0 is rasterized from an opaque white polygon on which lighting operations were performed. This color is blended with the texture lookup color CT_0 of the texture belonging to TU_0 . The result of this blending operation is used as input fragment color for the next texture unit TU_1 . This cascade-like multitexture procedure is repeated for all active textures but is limited by the maximum number of texture stages. As texture blending function we use `GL_MODULATE`. The output argument of a texture unit is the product of its input arguments. If the luminance of the multitextured image is too weak, one can scale the output argument with the extension `GL_RGB_SCALE_EXT`. The result is clamped to $[0..1]$ for alpha and each color channel. The built-in OpenGL texture blending functions are quite limited but useful. For more advanced texture combinations there exists a variety of possibilities. Texture shaders offer a bigger number of

built-in texture blending functions and texture combiners join arithmetical operations to create custom texture blending functions (KILGARD 2004). One crucial argument not to use these extensions is the lack of support of these extensions by a series of GPUs. Nowadays, programmable graphic pipelines as illustrated in figure 1 are common features of recent GPUs. A fragment shader extends the per-fragment operations and allows the definition of customized texture blending functions. This solution is illustrated in the following section.

3.2 Customized blending functions using fragment shaders

A fragment shader operates on each single fragment created by the rasterizer of the graphics pipeline (see figure 1). A fragment shader has access to the data of all active texture maps and the texture coordinates of a fragment, which was interpolated by the rasterizer, is available as parameter. Figure 3 shows a basic fragment shader for customized texture blending. This shader

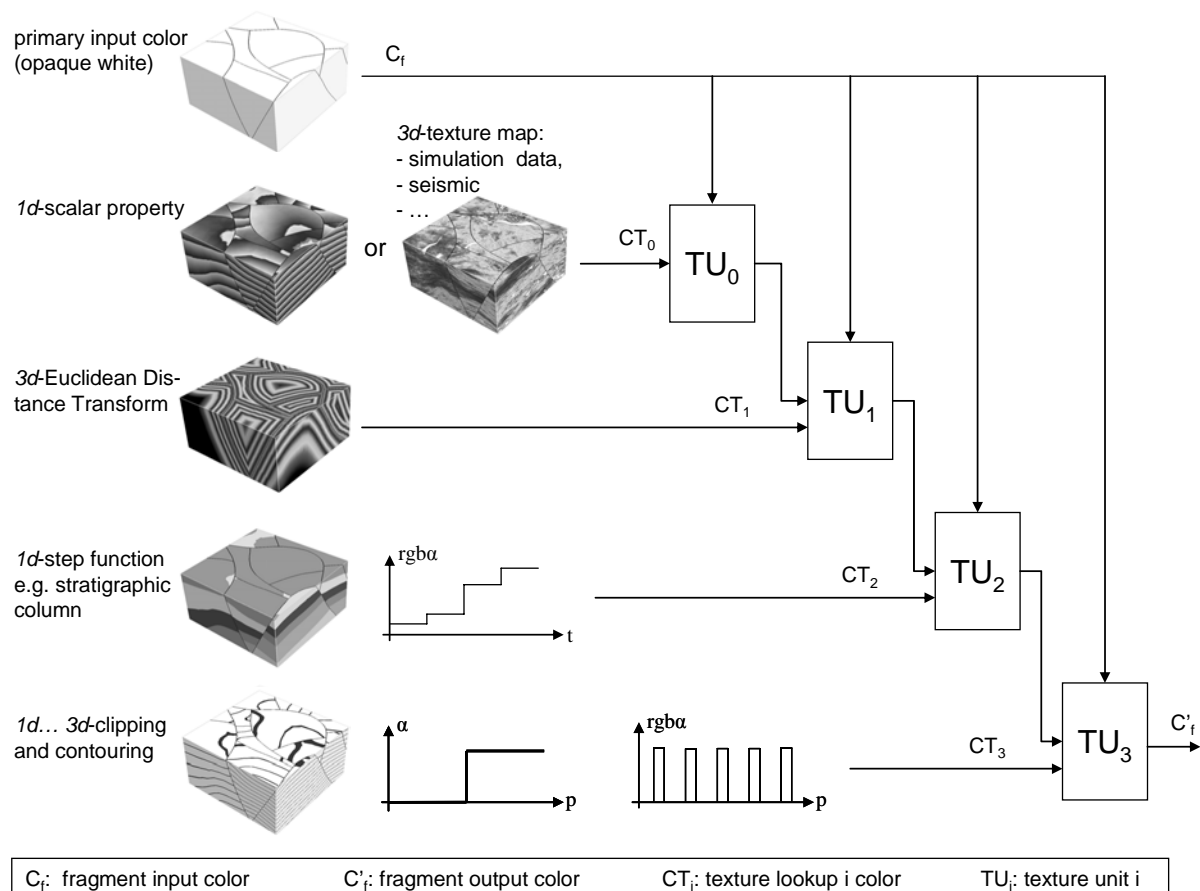


Fig. 2: This figure illustrates an OpenGL multitexture environment. A texture unit TU_i takes the current fragment color C_f and the texture lookup color CT_i to blend the output color C'_f . Several texture units can be combined in a cascade where the output color of one texture TU_i unit is used as input fragment color of the following texture unit TU_{i+1} .

```

void main(   float3 tex_coord_0 : TEXCOORD0,
            float3 tex_coord_1 : TEXCOORD1,
            float4 frag_color_in : COLOR,
            out float4 frag_color_out : COLOR,
            uniform sampler3D tex_0,
            uniform sampler3D tex_1,
            uniform float bias )
{
    float4 tex_color_0 = tex3D( tex_0, tex_coord_0 );
    float4 tex_color_1 = tex3D( tex_1, tex_coord_1 );
    frag_color_out = frag_color_in * lerp( tex_color_0 ,
    tex_color_1 , bias );
}

```

Fig. 3: CG fragment shader for texture blending function.

uses two 3D texture maps and the initial fragment color as input arguments to compute the resulting output fragment color. The texture coordinates `tex_coord_0`, `tex_coord_1` and the fragment color `frag_color_in` are passed as values of function parameters. `frag_color_out` is the alias name for the register where the output fragment color is written, `tex_0` and `tex_1` are the two 3d-texture maps. The CG function `tex3D` is used to perform the texture fragment color lookup. The CG function `lerp` interpolates linearly the two texture lookup colors `tex_color_0` and `tex_color_1`. The `lerp` function performs a linear interpolation between two vectors like: $c = a \cdot (1 - \text{weight}) + b \cdot \text{weight}$. The value of weight is passed as a user-defined variable `bias` from the client space. Modifying this variable by the user application results in interactive fading effects between the two texture maps. Finally the result of the linear interpolation is multiplied with the input fragment color. The input fragment color is interpolated by the rasterizer from white opaque polygons on which lighting computations were performed. Thus, the lighting is honored by the fragment shader. The fragment shader from figure 3 can easily be extended for additional texture maps and blending functions. Of course, color and alpha channels can also be treated independently.

4 Implicit intersection and clipping

The previous section introduced step functions over geological time to visualize the stratigraphic column on a model. This section will generalize the concept of procedural texture maps combined with different texture coordinates. The basic ideas behind this concept will be elaborated in

two steps: (1) the implicit calculation of intersections between tri-variate functions and (2) the application of Boolean CSG operations to compute clipping and penetration of implicitly defined bodies.

4.1 Implicit intersection

The examined problem is the computation of the intersection of an iso-value surface S_A of one function φ_A with an iso-value surface S_B of a second function φ_B . An example is the outline of a geological horizon on a cross-section. The numerical solution is the application of the Marching Tetrahedron (MT) algorithm twice. In the first pass the polygons of the iso-value surface S_A are computed. Applying MT a second time constrained by an iso-value of function φ_B to this set of polygons results in the intersection curve. So the additional cost is the second run of MT. When the extracted iso-value surface S_A consists of n polygons and k of these n are intersected by S_B , the additional effort is at least $O(k + \log(n))$ using an interval tree (BAJAJ 1999) or parametric octree decomposition. This effort increases linearly for each additional intersection to be computed. A graphical algorithm with constant complexity $O(1)$ will be introduced in the following. To visualize an intersection curve of two functions, let us draw the iso-value surface S_A with a step function over the domain of φ_B . This step function takes identity values for the blending function except for the value the intersection is to be computed. For the `GL_MODULATE` blending function we define a 1D texture map with opaque white values. Only the position of the intersection value carries the color information of the intersection curve. The normalized values of φ_B are used as texture coordinates.

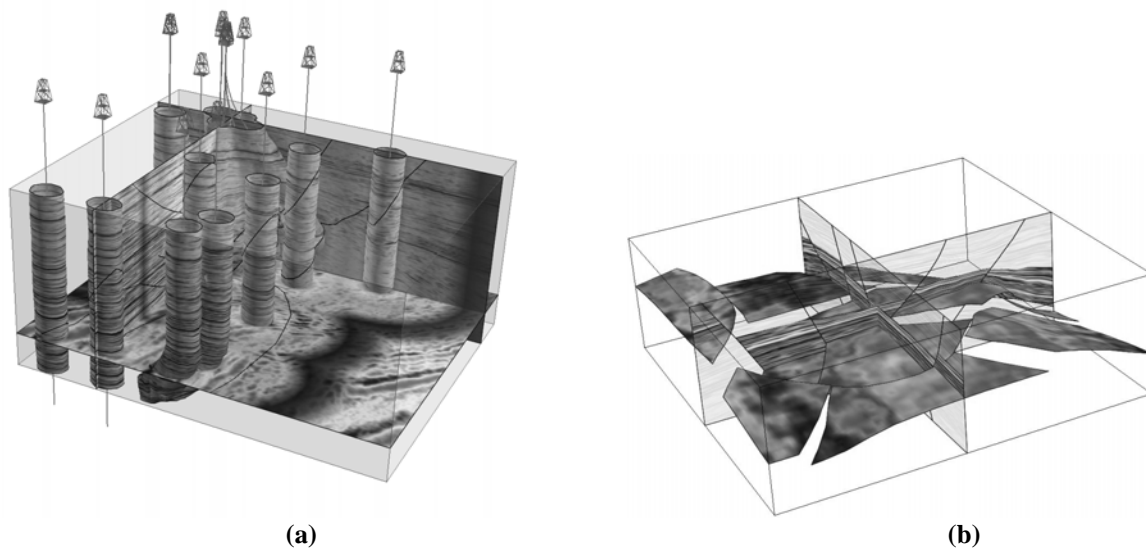


Fig. 4: (a) shows cross-sections and iso-value surfaces extracted from a tetrahedral mesh. The iso-value surfaces are defined by an iso-distance to the wells. The distance function and seismic data are co-visualized. Fault traces are highlighted by black outlines. Data by courtesy of Earth Decision. (b) Shows two cross-sections and an isochronous surface in a tetrahedralized model. The surfaces were texture-mapped with a synthetic seismic cube and co-rendered with a stratigraphic column. One stratigraphic layer is highlighted, the other parts have a higher transparency. Data by courtesy of Total.

4.2 Clipping of surfaces and penetration of bodies - applications to CSG

The above introduced algorithm for visualizing the intersection of two implicit surfaces is developed a little further. Especially for CSG applications, not only the intersection between two bodies but also their clipping and penetration is of great importance. Let us consider a step function over the domain of the function φ_A . This function takes zero values in the interval $[\min(\varphi_A), b]$ and is one in the interval $[b, \max(\varphi_A)]$ where b is an arbitrary value of the range of φ_A and defines the clipping threshold. The values of this step function can be treated as alpha values of a texture map, and rendering an iso-value surface of the function φ_B with this texture map will paint all parts of the iso-value surface that take values less than b of φ_A transparent. This is a graphical solution for a clipping algorithm with constant complexity $O(1)$ for two implicit surfaces of arbitrary size and complexity. Figure 4(b) shows two cross-sections and an isochronous surface with one highlighted stratigraphic layer. The other stratigraphic layers were clipped using the described algorithm together with the GeoChron time parametrization (MALLET 2004).

This method can easily be extended to more surfaces using more or higher-dimensional texture

maps. In CSG applications, bodies of high complexity are often defined as implicit functions. Implicit functions can be easily defined on the nodes of the tetrahedral mesh. In the following, the convention that positive values define the interior and negative values define the exterior of a body is used. The iso-value surface where the implicit function takes the value zero is the boundary surface of the body. Multitexturing techniques can be used to perform Boolean operations like union, difference and intersection on two or more bodies defined by the implicit functions. Figure 5 shows a reconstruction of the Stanford Bunny data set and a sphere. Both bodies are defined as implicit functions with the abovementioned constraints. The images show the results of the Boolean CSG operations. For each Boolean CSG operation a transfer function over the domains of the implicit function φ_A and φ_B defines the transparency of a fragment according to the value tuple of the implicit functions. The transfer functions are illustrated in figure 6. This method can be generalized to n -implicitly defined bodies by nD transfer functions. Again the complexity of this method only depends on the computation of the texture maps from the nD transfer functions and is hereby constant $O(1)$ for any model size and complexity.

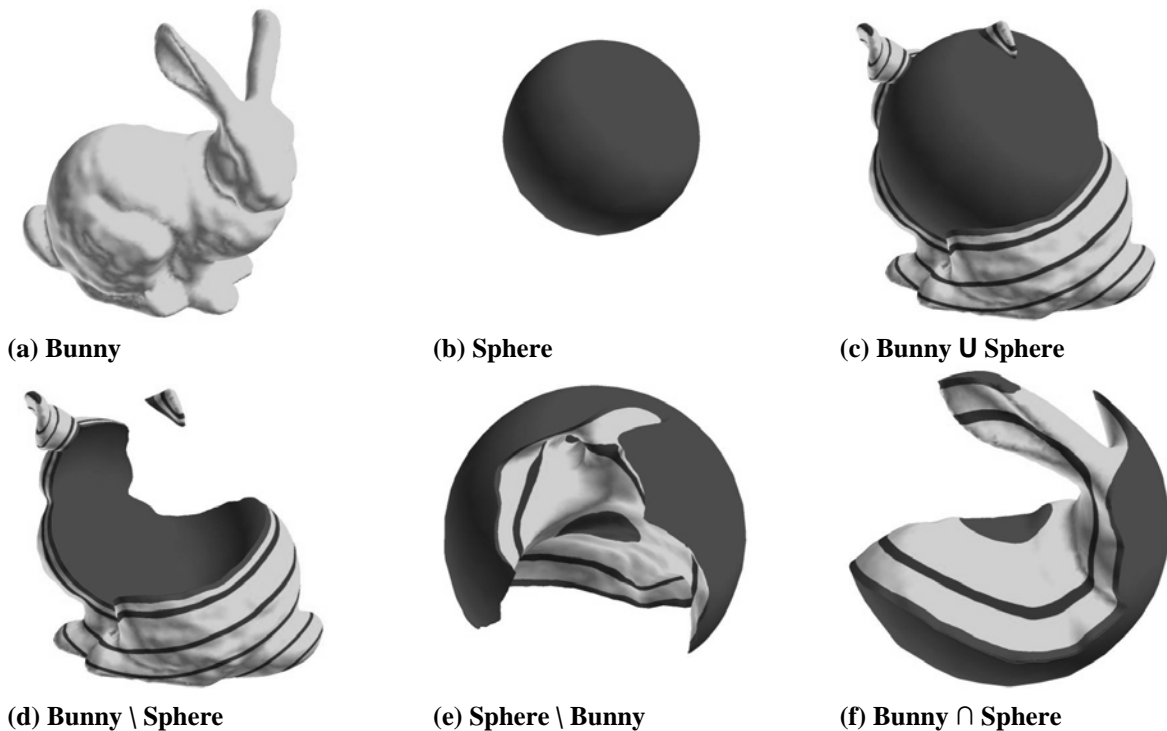


Fig. 5: This series of figures shows the application of Boolean operators to CSG. The Stanford Bunny (a) and a sphere (b) are defined as implicit functions on the nodes of a simplicial 3d-complex. The CSG operations were performed using multitexturing techniques with constant complexity $O(1)$. Data by courtesy of Stanford 3d-Scanning Repository.

5 Application to complex 3D GIS queries

GIS involves simultaneous display of quantitative and qualitative properties to extract space domains favourable to studied phenomenon. The choice of the characteristics to combine lies on the understanding and the knowledge of the phenomenon origins or causes. For instance, uranium deposits are grandly controlled by fluid circulations controlling the redox conditions of the milieu. In such example, potential ore deposition volumes appear by combining distance to fault, lithology and geochemical properties.

Another example in sedimentary geology is the study of stratigraphic sequences and the facies associated with. In addition to the 3D display of stratigraphic column (see figure 7a), the clipping function allows the extraction of specific sequence (figure 7b). On the figure 7b, the 3D GIS query could be summarized as “display the facies property associated to the specified sequence”. Facies maps can also be extracted at the same time (figure 7c).

6 Conclusions

In this paper we introduced a new and innovative method to perform Boolean operations on implicitly defined models. If geological attributes are defined by implicit functions this method can be used to perform complex 3D GIS queries in real time. The implicit model is defined on an unstructured tetrahedral mesh that can take geological discontinuities (e.g. faults) into account and enables to perform quantitative estimations. The 3D queries are performed on the Graphics Processing Unit (GPU). The 3D query is coded into procedural texture maps that are combined on the texture unit of the GPU or by user defined fragment shaders. Applying these textures to iso-value surfaces of the implicitly defined geological attributes, results into the desired 3D query. The complexity of such a query is constant of any model size and complexity and only depends on the screen size. The computation is performed in one rendering path (less than one second). The main drawback of this method is the need for an implicitly defined geological model. But, latest development in geomodeling software shows that implicit modeling becomes more and more important and will be one of the main future trends to explore.

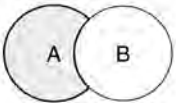
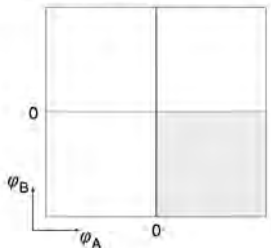

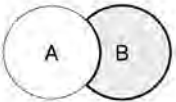
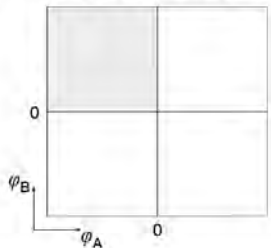

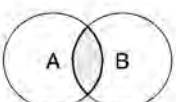
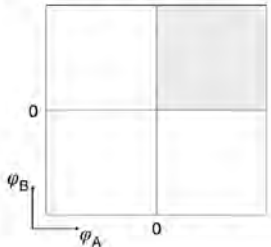

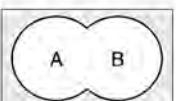
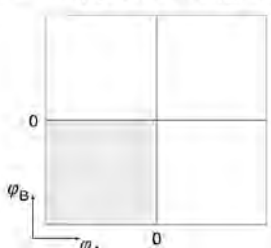

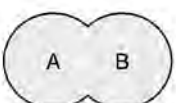
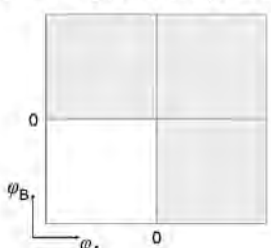

CSG operation	Look-up table for transparency	Cross-section through CSG result
$A \setminus B$ 	$\alpha = 1 \vee \varphi_A \geq 0 \text{ AND } \varphi_B < 0$ 	
$B \setminus A$ 	$\alpha = 1 \vee \varphi_A < 0 \text{ AND } \varphi_B \geq 0$ 	
$A \cap B$ 	$\alpha = 1 \vee \varphi_A \geq 0 \text{ AND } \varphi_B \geq 0$ 	
$\overline{(A \cup B)}$ 	$\alpha = 1 \vee \varphi_A < 0 \text{ AND } \varphi_B < 0$ 	
$A \cup B$ 	$\alpha = 1 \vee \varphi_A \geq 0 \text{ OR } \varphi_B \geq 0$ 	

Fig. 6: The look-up tables define the transparency (white quadrants) over the domains of the implicit functions φ_A and φ_B . φ_A and φ_B describe two bodies A and B with the convention that positive values define the interior and negative the exterior of the bodies. Each texture map is applied to the iso-value surface S_A and S_B . The right column illustrates cross-sections through the result of the Boolean CSG operation with additional texture mapping. φ_A defines a bunny and φ_B a sphere.

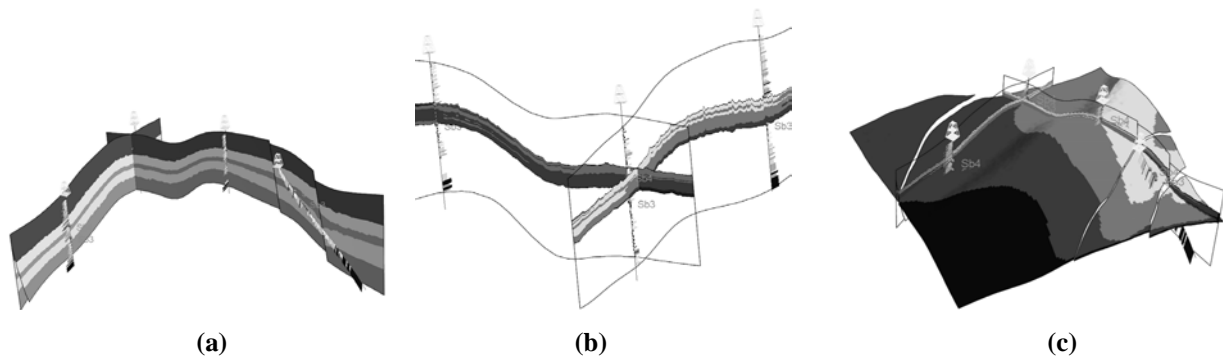


Fig. 7: Applications of 3D GIS queries in sedimentary modeling (Kedzierski, 2006): real-time rendering of the stratigraphic column (a), corendering of specified stratigraphic sequence and simulated facies defined as an implicit function (b), and real-time extraction of cross-sections and isochronous map painted with the facies property (c). Data by courtesy of Total.

7 Acknowledgements

This research work was performed in the frame of the GOCAD research project. The companies and academic members of the GOCAD consortium are hereby acknowledged.

8 References

- BLOOMENTHAL, J. & WYVILL, B. (1997): Introduction to Implicit Surfaces, Morgan Kaufmann Publishers Inc., San Francisco.
- COWAN E. J., BEATSON R. K., ROSS H. J., FRIGHT W. R., MCLENNAN T. J., EVANS T. R., CARR J. C., LANE R. G., BRIGHT D. V., GILLMAN A. J., OSHUST P. A. & TITLEY M. (2003): Practical Implicit Geological Modelling, Fifth International Mining Geology Conference. Bendigo.
- FERNANDO, R. & KILGARD, M. J. (2003): The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- FRANK, T. (2006): Advanced Visualization and Modeling of Tetrahedral Meshes, PhD thesis. INPL Nancy and TU Freiberg.
- KEDZIERSKI, P. & MALLET, J.-L. (2006): The Thalassa Project, In 26th Gocad Meeting Proceedings. Nancy, France.
- KILGARD, M. J. (2004): NVIDIA OpenGL Extension Specifications, NVIDIA Corporation.
- MALLET, J.-L. (2004): Space-time mathematical framework for sedimentary geology, Mathematical Geology 36, 1.
- MARK, W. R., GLANVILLE, R. S., AKELEY, K., & KILGARD, M. J. (2003): CG: a system for programming graphics hardware in a C-like language, ACM Transactions on Graphics 22, 3, 896–907.
- ROYER, J.-J. (2005): Conditional integration of a linear function on a tetrahedron, In 25th Gocad Meeting Proceedings. Nancy, France.
- SEGAL, M. & AKELEY, K. (2003): The OpenGL Graphics System: A specification (Version 1.5), Silicon Graphics.