Modeling Geological Fault Configurations

ARYA BANAEIZADEH, University of Calgary, Canada

We present a framework for the geometric modeling of geological fault configurations. The program will use a sketching interface to handle user input. The surface is then interpolated into a 3D space while maintaining basic surface-related properties derived from their geological features. In addition, some sketching/modeling operations for further modifying the data are provided so that various fault configurations can be modeled using less effort.

Key Words: Computer Graphics, Geometric Modeling, Geology

1 INTRODUCTION AND GEOLOGICAL BACKGROUND

Geological modeling is an approach where geological properties of a certain region of the earth's crust are computerized and represented [12]. There are various types of surfaces that can be represented in geological modeling software. One of these geological phenomena is known as faults. According to Encyclopedia Britannica, a **Geological Fault** is a planar or gently curved fracture in the rocks of Earth's crust, where compressional or tensional forces cause relative displacement of the rocks on the opposite sides of the fracture. Faults are mostly studied for gathering more information about the seismicity of a certain area.

Geological Modeling can give valuable information on why and when a certain phenomenon has occurred. One of the more significant applications of geological modeling arises from hydraulic fracturing operations. Hydraulic fracturing operations extract fossil fuels out of the deep layers of earth by essentially causing cracks in the earth's rocks and extracting out the fuel by injecting a liquid and sucking it back. This whole process may contribute to creating faults. These faults can be the primary reason for causing earthquakes. Studying these faults can tremendously help with better understanding how earthquakes of this kind take shape and how they can be prevented.

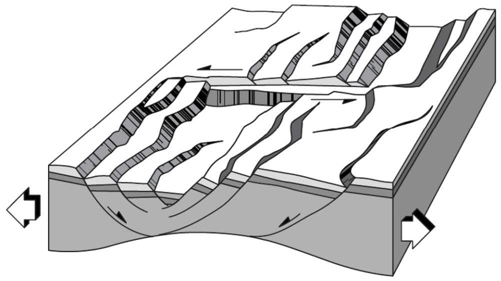


Fig. 1. This figure shows a sketched version of an actual fault configuration, showcasing its complexity. The image is taken from the book Earth Structures [21].

Author's address: Arya Banaeizadeh, arya.banaeizadeh@ucalgary.ca, University of Calgary, 2500 University Drive N.W., Calgary, Alberta, Canada, T2N 1N4.

The difficulty with geological modeling, however, is that oftentimes it is very difficult to be able to accumulate accurate data such that it is directly transmissible to a digitized representation. Luckily, there is a solution to at least part of the problem; geologists speculate over what is more likely possible to have taken place in the real world, then they continue to validate their assumptions by using different methods and techniques such as flow simulation. That is why most of the research in the area has shifted focus to creating tools and programs for geologists so that they can easily model their assumptions. This way hopefully, the uncertainty problem can be addressed.

This is where our motivation comes in; our goal is to facilitate the process for modeling geological fault configurations. We mainly set out to address the following in our program:

- Modeling fault configurations with intersecting fault surfaces.
- Freedom of control and ease in the modeling process
- Modeling operations for handling different scenarios and configurations

Most geologists are not familiar with the techniques of recreating the complex geometries of geological fault configurations, as many of these techniques require substantial knowledge in computer graphics. Unfortunately, there are not that many programs curated for the sole purpose of modeling fault configurations (some of the relevant research and how they compare to our research are discussed in section 2). That is why our goal is to make some progress towards this specific research so that hopefully, more sophisticated tools can be provided to geologists to create better models.

In other words, our main objective here is to deal with the mathematical and modeling challenges that arise from these types of configurations, with the secondary goal of making the process easy for the people who will perform these modeling operations.

2 RELATED WORK

Various modeling and sketch-based programs are used in service of geological modeling and in general, modeling natural phenomena. From modeling salt domes [14] to interpreting seismic data [5]. More and more possibilities and alleyways are explored by the day. In the following, we will introduce some of the more relevant works in more detail.

The first prominent work in modeling geological faults is the work of Wu et al. [23]. They use relevant geological properties of faults to represent a digitized, low-resolution model of fault configurations.

in their paper, Caumon et al. present a series of guidelines and rules for geological modeling; their paper mostly focuses on stratigraphy [4]. Their main purpose was to point out the geometrical properties of geological surfaces and how they should be maintained.

In their paper, Amorim et al. have used a sketch-based modeling approach for creating different types of geological surfaces [1]. Their work allows the user to sketch layers from the top view and indicate the arrangement of layers using sketch gestures. The user can create stratigraphical surfaces and faults using sketches and gestures. In this work, a significant amount of focus is on the gestures and user experience.

In 2015, the project Rapid Reservoir Modeling was introduced [7]. This paper presented a Sketch-based approach toward modeling reservoirs and well trajectories. The project has developed mechanisms for correct mathematical models of stratigraphy. The prototype allows the users to intuitively sketch stratigraphical layers and get geologically correct 3D models. The project is still ongoing with the latest publication by Jacquemyn et al. [9].

3 MODELING FRAMEWORK

In this section, we will explain how the overall framework of the modeling program works. The framework consists of a few main parts which are as follows:

- Overall procedure of the modeling
- Sketching input and filtering
- Surface interpolation and representation
- Modeling operators

In terms of the geometry of faults, there are various properties that one can take into account. One of the more significant properties is whether or not a fault completely divides the rock that it is formed into two separate rocks. For this project, we are not going to focus on parts that do not divide their rocks into two parts, we will refer to these types of faults as **partial faults**. In the future, we plan to incorporate partial faults into the modeling process. Thus, when we talk about modeling geological faults in the context of this project, we are only talking about non-partial faults. In section 7, we are going to explain how this expansion is going to be done. The figreffig:blindfault shows an example of a partial fault.

In section 4, we will discuss the modeling operators in more detail, as well as how surfaces will be updated as these operators are applied. Here, we will describe the core functionalities of the framework.

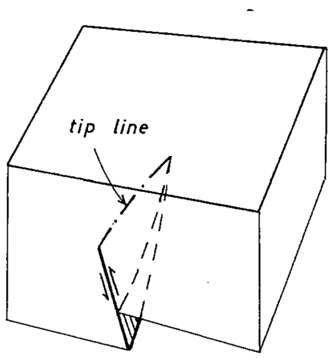


Fig. 2. A blind fault is a good example of a fault that does not divide the rock that they were originated into two separate rocks. This figure is originally demonstrated in the book Modern Structural Geology [17].

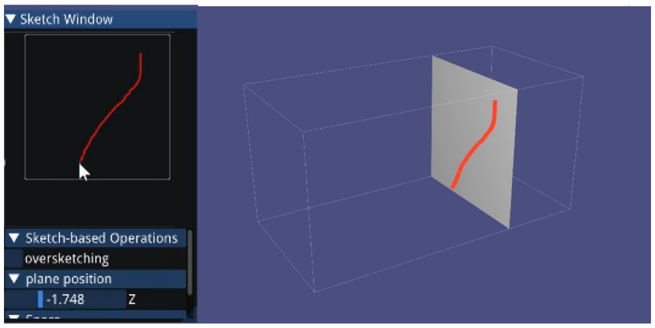


Fig. 3. The sketch window alongside where it will be placed in the 3D space. The user can specify the plane position by changing the value of the corresponding slider.

3.1 Modeling Procedure

In this section, we will explain the overall procedure of how modeling can be done. In sections 3.2 and 3.3, the mathematical and algorithmic sides of the procedure are explained.

3.1.1 Sketch Placement. The user begins the modeling process by sketching. A 3D box is provided and shown, indicating that all the models will exist inside this box. Inside the box exists a plane, called the **sketch plane**. The sketch plane has a corresponding 2D window in which the user can draw a stroke. Whenever the user has finished drawing the stroke, it is projected onto the sketch plane. The user can change the position of the sketch plane (sliding it between the two ends of the bounding box plane) or change the axis of the plane, in the x, y or z-direction. After the user is done with a sketch, a surface is generated. The figure 3 is showing the sketch window and its corresponding place in the 3D space.

3.1.2 Surface Generation. After the sketching process, a surface is generated and shown back to the user. The user can update the surface by placing new sketches on different regions of the bounding box, or over sketching on top of the current sketch, which will update the existing curve. Another option is to create a new fault by drawing a new sketch. Creating or even modifying the surfaces may update other existing surfaces in the model. This part is explained in more detail in the section 4.1.

3.2 Sketching Input

The approach in which the user communicates the surfaces is by sketching. In their survey paper, Olsen et al. provide the general pipeline of sketch-based modeling programs [15]. This pipeline has three major steps; taking the sketch as input, filtering and resampling the sketch, and creating a surface based on the resampled input. Filtering overall is an inevitable part of the sketching pipeline, and when applied, it can resolve many errors with regards to the sketching input, whether it is a jiggle due to user, or sampling issues. In our framework, we have used a combination of supersampling, filtering and weighted averaging. The filtering algorithm used is the *Reduce Resolution* algorithm that was first introduced in the work of Samavati and Bartels [19]. After the filtering is done, we get a consistent stroke that can be processed into a surface.

3.3 Surface Interpolation and Representation

After taking the sketch as input and filtering the sketch it is time to generate the surface based on the input. The surface generation method composes of two steps, surface interpolation and surface representation. For surface interpolation, we are going to use Hermite radial basis functions. Hermite radial basis function implicits(HRBFs) were first presented by Macedo et al. [11]. This approach was first utilized in a sketch-based environment in the work of Brazil et al. [3]. our approach is also based on the latter paper, in which a variation of what was initially proposed is used. For more information on the mathematical aspect of HRBFs refer to these works, as we only briefly explain how this is done.

3.3.1 Theory. Given a set of n-dimensional pairwise distinct points $P = \{p_1, p_2, ..., p_m\}$ and their normals $N = \{n_1, n_2, ..., n_m\}$, HRBFs will provide a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ which will interpolate a surface with these features:

$$f(p_i) = 0 \tag{1}$$

$$\nabla f(p_i) = n_i \tag{2}$$

Equation 1 is similar to any other interpolation scheme. We define the surface whenever the function is equal to zero. In addition, the gradient of the function is equal to the normal that is given as input (equation 2). We define the input normals using the tangent of the two-dimensional curve. More specifically:

$$\vec{N} = (\overrightarrow{P(i) - P(i-1)}) \times \vec{N_A} \tag{3}$$

Where $P(i)$ is the ith point on the curve and $\vec{N_A}$ is the unit vector in the direction of the sketch plane.

There are a list of reasons explained in both sources as to why HRBFs are an appropriate choice for surface interpolation, we'll mention the ones that are specifically useful for our case

- **Implicit surface interpolation:** HRBFs give us a clear notion of the inside and outside of an object, which is useful when we are dealing with multiple fault surfaces. Needless to say, this is not strictly an advantage of HRBFs and can be achieved with other implicit surface generation algorithms.
- **Being defined everywhere:** Perhaps the most important reason for our choice is that HRBFs are defined everywhere no matter the input. This will help us in designing the sketch operators; since there would be scenarios where users decide to update some surfaces which leads to the requirement for updating all the other surfaces that are in interaction with the initially updated surface. In these cases, we can use this property to update the surfaces. This advantage paired with the implicit surface property of HRBFs will help us also define the inside/outside regions for each fault.

In addition, these surfaces are relatively easy to calculate, can be interpolated using a linear system, and work efficiently for sparse inputs.

since Hrbfs produce an implicit surface, we need an additional step to render the surface. For this reason, we have used the marching cubes algorithm first introduced by Lorensen and Cline [10]. The region where marching cubes are applied is the same as the bounding box region.

As mentioned, the HRBF interpolation will define the surfaces everywhere. This causes a problem because when dealing with faults, fault surfaces are usually truncated or cut against other faults; this means that we cannot only rely on the interpolation to model different configurations. That's why in some cases we need to rely on some computational geometry algorithms to compute on the explicit geometry. In section 4, we will get to these algorithms in more detail.

4 MODELING OPERATIONS

This section will explore the two main modeling operations that are part of the framework; starting with the operation for intersecting faults and moving on to sketching and creating surfaces on existing surfaces.

4.1 Fault Intersection

Geographically speaking, when new faults are formed in a region where existing faults lie they change the location of the existing faults that they intersect with, since faults displace the rock in which they are formed [17]. figure 4 shows how this displacement is done.

There is one direct result that we would get when we apply this to our framework; whenever the user decides to create a new fault the old fault's geometry should be updated. More specifically, it should be cut. We cannot solely rely on our implicit interpolation scheme for cutting surfaces as the surfaces are defined everywhere in HRBFs. However, we can treat the geometries as explicit meshes and we can use specific computational geometry algorithms to update the geometries.

Here is how we are going to conceptually proceed with updating the meshes:

(1) detect if there are any conflicts between the newly inserted geometry and any existing geometry.
(2) for all the geometries that have conflicts with the existing mesh, the curve in which they intersect with the new mesh is calculated.
(3) the existing geometries will divide into two geometries wherever there is an intersection curve.

In the following, we will explain how the process for computation of these steps is done.

4.1.1 Method. The first item on the list is to see whether the newly added surface intersects with any old surface. We have to rely on explicit meshes and thus, computational geometry algorithms to detect conflicts. We start by combining the geometry of the two objects and after that, we check whether there are conflicts between the faces of the surfaces. Since we have to check each face of the geometry against all other faces in the other geometry, we cannot achieve a linear runtime with this algorithm. This part is one of the main hurdles against a reliable performance of the framework, and thus is an interesting focus point to have a better algorithm for intersecting meshes in the future. Finding mesh conflicts is a vast research topic on its own. Please refer to the work of Park [16] and Botsch et al. [2] for more information.

At this stage, we have one mesh that is a combination of the two surfaces with vertices and edges at the points of intersecting faces. This mesh is non-manifold, i.e., there are edges in the mesh that connect more than two faces together. We can find these edges and label them as the intersection curve. Here, we might need to change the data structure for the mesh to something like a Half-edge data structure [13], as we can easily infer what are the adjacent faces to a certain edge. If this number is more than two faces, then that edge is part of the intersection curve. based on this we can determine the intersection edges and reorder them to get the full curve. After getting the curve we can separate the two meshes back into two different meshes and proceed.

Finally, we have to separate the geometries. Here, we will take advantage of the HRBF function to determine which parts of the object belong to the same geometry. Algorithm 1 will show a pseudocode of separating geometries. The fault plane that is to be separated will be inputted as an explicit geometry. The interacting fault will be inputted as an implicit surface.

Algorithm 1 Separate Geometry

```
procedure separateGeo(Object o, Implcit i)
    Object o1,o2
    for Face $f$ in o do
        if i.evaluate(f.center()) > 0 then
            o1.add(f)
        else
            o2.add(f)
        end if
    end for
    return o1,o2
end procedure
```

The order of complexity for a mesh with n faces is $O(hn)$, where h is the number of control points for the implicit surface. The *evaulate* function is essentialy the calculated HRBF function and it will take $O(h)$ for a point [11]. Since we are doing this operation for each face we get to the order $O(hn)$. Usually, the number of sample points are quite low, so we can say with a high degree of confidence that the order in most cases is close to a linear runtime. Furthermore, the *evaluation* function can be parallelized to a high degree.

4.2 Sketching on Surface

One of the ways in which the users can communicate new surfaces is by sketching directly on top of existing surfaces. However, this requires an extra step to project the sketched curve onto surfaces. In our framework, we let the user choose a surface that they wish to sketch on and then they can draw a stroke and see the result. project each sampled point on the surface to get a series of points that comprise the drawn stroke.

Commonly, any type of interaction with surfaces in a 3D environment is imbued with the ray casting algorithm. Ray casting was first introduced by Scott as a means of projecting a point from the screen to the 3D scene for Constructive Solid Modeling [18]. We are going to use the same algorithm in our work.

4.2.1 Method. Given a point in the screen space with the normalized coordinates $m = (x, y)$ we wish to calculate where the point intersects with the object a in the 3D scene s with camera c. Algorithm 2 shows the pseudocode for ray casting.

Algorithm 2 Ray Casting

```
procedure RayCast(Vector2D m, Object a, Scene s, Camera c)
    ray_clip = vector3d(m.x,m.y,1)
                                        ▷ make $m$ a point in 3D space
    ray_eye = inverse(c.projection_matrix) * ray_clip
                                        ▷ project to camera space
    ray_eye = vector3d(ray_eye.x,ray_eye.y, 1)
    ray_world = inverse(s.view_matrix) * ray_eye
                                        ▷ poject to world coordinates
    ray_world.normalize()
    for Face $f$ in a do
        if hit(ray_world,f) then
            bc = getBarycentric(ray_world,f)
            if hit_point inside triangle then
                return f, bc;
            end if
        end if
    end for
end procedure
```

The worst case of the ray casting algorithm is $O(n)$ for n number of faces. For each vector and triangle we can calculate whether the vector hits the corresponding plane of the triangle and whether it lies inside the trinagle in a constant time.

After projecting each point to the surface, it is time to interpolate the surface based on these new points. The approach is the same as sketching on the sketch plane; first the points are resampled, using the same resampling algorithm, but this time in 3D (we are not going to have any issues here as the algorithms are generalizable to higher dimensions). Afterwards, the normals are calculated so that they can be inputted to the interpolation algorithm. We generalize equation 3 for calculating the normals:

$$\vec{N} = (\overrightarrow{P(i) - P(i-1)}) \times \vec{N_S}(P(i)) \tag{4}$$

Where $P(i)$ is the ith point on the curve and $\vec{N_S}(P(i))$ is the normal of the surface. The surface normal can be calculated using the interpolated function of HRBFs. $\nabla f(p)$ will give us the gradient of the surface at point p, which is equal to the surface normal at the function level set. As a result, the normals are calculated and given as inputs to the interpolant for generating a surface.

In the future, this feature will be used for turning complete faults into partial faults. The users can draw on existing surfaces from boundary to boundary to segregate part of the goemetry and then delete part of the surface to make the fault partial. This operator will prove much more beneficial when dealing with these types of surfaces.

5 DATA STRUCTURE FOR STORING FAULTS

As we are dealing with multiple surfaces in a geological model, we need to store the fault geometry information in a data structure. We can use simple data structures like lists, but they will not utilize most of the properties that we can infer from our fault surfaces. In this section we will explore the appropiate data structure that is suitable for faults and how they can contribute to having a better

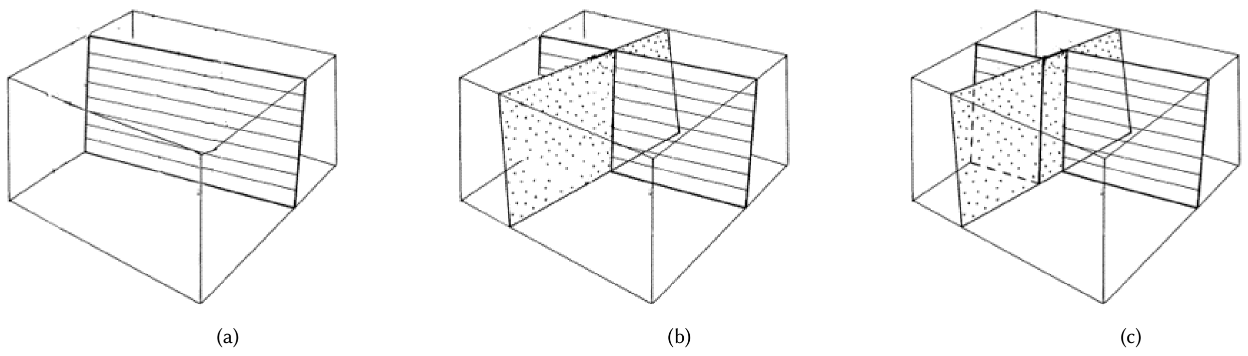


Fig. 4. This figure showcases what happens when a new fault is created in a region where a fault already exists. in figure 4b we see the new fault without displacement, and in figure 4c the displacement is also shown. pictures are a modified version of a picture from the book Modern Structural Geology [17].

modeling framework and optimizing techniques. Please note that this part is subject to change as it is an ongoing part of the research.

5.1 Model Properties

After a more detailed examination of our model while considering data structures, a few key points emerge.

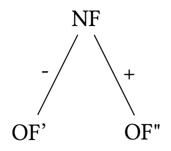
- As faults get added to the model, existing surfaces might be updated.
- Each added fault will divide its existing region into two separate regions. Theoretically, a newly added surface can double the number of regions in a certain model, meaning that for a model with n faults, we might have up to but no more than 2^n regions. notwithstanding, having 2^n regions is geologically impossible for higher n values.
- Based on our interpolation scheme, for a single point we can determine in which region of the space it can lie. if we do this for a system of n faults with r regions, we can determine this with the time complexity of $O(hn)$, where h is the average number of control points per n faults.

Based on these properties, the data structure most appropiate for our case is to use binary trees; we can proceed to create methods for insertion and removing operations, as well as the general structure. This part is discussed in section 5.2

5.2 The Tree

As shown in section 4.1, the newer faults[1] divide the older faults into two separate geometries. This is used as the general rule for our database; if in a model there is a new fault that has divided the older fault into two parts (figure 4), add the older fault as the child nodes of the new fault. The tree below demonstrates this for the new fault NF and old fault OF which will divide into two separate geometries after the new fault is added.

[1]In this context, by new we mean faults that are geologically formed later. We are not talking about the order in which the user creates the models.



Furthermore, each edge of the tree corresponds to the positive or negative value of the interpolant function. This will also result in the leaves of the tree corresponding to the regions that are formed by the faults.

When the user adds surface, based on the location of the curve in our entire modeling region we determine which regions the new surface should be added to. This also depends to the geological order of the faults; the user can choose whether the surface that they want to add is formed earlier or later in the context of geological development. If the surface is formed later in the model, it will divide all the surfaces it intersects to two. On the other hand, if the surface is older, it will be divided by all the surfaces it intersects with.

There are a lot of possibilities with regards to optimizing the algorithms we use by having a proper data structure such as the one introduced in this section. However, since this part is still in development we are going to suspend exploration of this topic in light of avoiding over-optimization and not getting too much in theory without having a practical use. Algorithm 5 will show how the insertion procedure works in the data structure.

6 DISCUSSION

6.1 Results

In this section, we will go through some of the results that were generated in the framework.

6.1.1 A simple case. As mentioned the fault displacements similar to 4happen quite common in the real world. Figure 5 shows an instance where this can be applied in our framework.

6.1.2 Flower Structure. One of the common configurations that can take shape as fault systems are flower structures [21]. As seen in figure 6. We tried to replicate this in our framework which resulted in figure 7.

Algorithm 3 Finding the region

```
procedure FindRegion(ControlPoint[] p, Node n=root)
    if !n then
        return n
    end if
    if all ControlPoint in p is on positive side of n then
        return FindRegion(p , n->rightChild)
    else if s is on negative side of n then
        return FindRegion(p , n->leftChild)
    else
        return n
    end if
end procedure
```

Algorithm 4 Tree Insertion

```
procedure InsertOld(Surface s, Node n=root)
    Node s_node = node(s)
    if !n then
        n = s_node
        return
    end if
    if s is on positive side of n then
        Insert(s_node , n->rightChild)
    else if s is on negative side of n then
        Insert(s_node , n->leftChild)
    else
        Insert(s_node , n->rightChild)
        Insert(s_node , n->leftChild)
    end if
end procedure
```

Algorithm 5 Tree Insertion

```
procedure InsertNew(Surface s, Node n=root)
    Node s_node = node(s)
    if !n then
        n = s_node
        return
    end if
    Node temp = n
    for node t in temp subtree do
        InsertOld(t.s,n)
    end for

end procedure
```

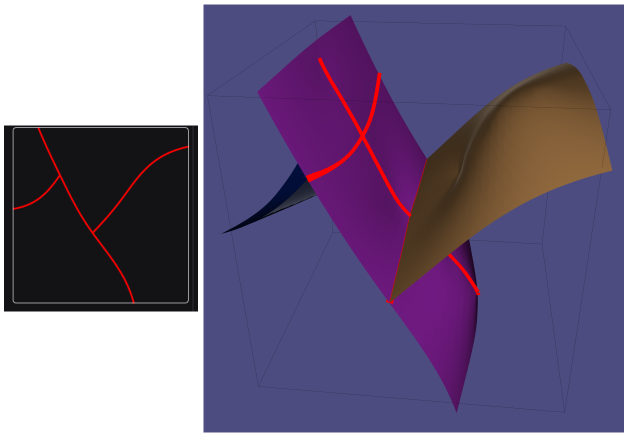


Fig. 5. Modeling a fault displacement in our modeling framework and the corresponding sketch window. The drawn strokes by the user need not complete boundaries as the interpolation scheme will properly define the surface.

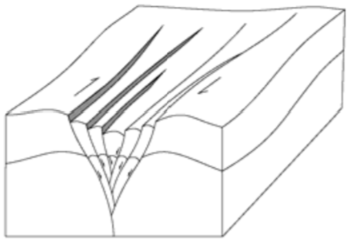


Fig. 6. The flower structure configuration (conformal negative) [24]

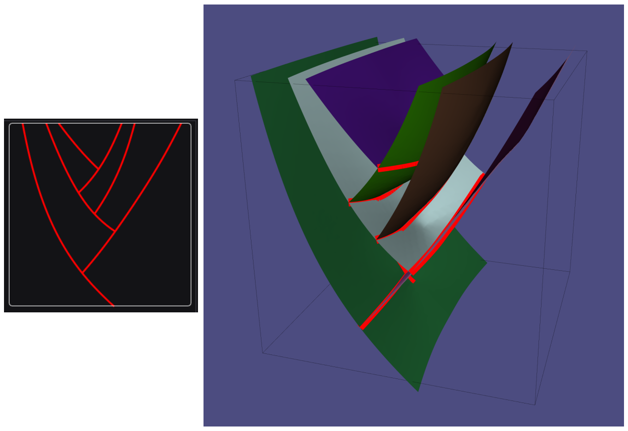


Fig. 7. Modeling a flower structure in our modeling framework and the corresponding sketch window.

6.2 Implementation

We have used **C++** as the main programming language for implementing the project. C++ is usually the go-to place for creating applications that involve heavy computer graphics concepts due to its high performance. **OpenGL** has been used for rendering and visualizing the graphical data. OpenGL is an API for rendering two- and three-dimensional vector graphics. OpenGL utilizes the graphics processing unit to achieve fast results.

For some of the geometry processing operations, we have used **libigl**. libigl is a simple C++ geometry processing library. libigl was developed at the University of Toronto with the main goal of facilitating the workflow of implementing new research ideas for fellow researchers in computer graphics and geometric modeling [8].

For some of the computational geometry algorithms, libraries **CGAL** and **Embree** are used. CGAL is a software project that provides easy access to efficient and reliable geometric algorithms in the form of a C++ library [20]. Embree is a collection of high-performance ray tracing kernels, developed at Intel [22].

For doing matrix calculations, we have used **Eigen**. Eigen is a C++ library for calculating matrices, vectors, and other related algorithms in the field of linear algebra [6]. We will be using Eigen for calculating matrices. libigl also heavily relies on Eigen for its calculations.

Finally, for handling the graphical user interface operations we have used **Dear ImGui**. Dear ImGui is a very powerful library for creating and visualizing graphical data, and it is designed specifically for 3D real-time programs in mind.

7 CONCLUSION AND FUTURE WORK

In this report, we presented several methods for recreating and modeling geological fault configuration. Hopefully, this approach will prove useful for modeling various fault configurations.

This is ongoing research and we are not even close to the end. We plan to improve our program in the following respects:

- **Incorporating Partial Faults:** Since partial faults are caused often in fault configurations, there is a strong need for updating the framework to handle these types of surfaces.
- **Updating the Database to Handle Partial Faults:** Currently, our database works only with complete faults; as we update the modeling framework we have to update the data structure for it as well.
- **More modeling Operations:** Currently, we have the basic modeling framework for modeling faults, but we need more modeling operations for handling more complex configurations. Operations like separating geometry, displacing geometry or extending geometry (these will be discussed in extensive detail in future works).
- **Non-destructive modifiers:** A nice to have feature for any modeling software is to make the modifiers non-destructive so that the users can edit the surfaces more freely without worrying about the modeling operations that they have applied on the models. Adding this to our framework makes the process less tedious for users.

As it is clear, most of these points relate to partial faults. If we could come up with a sustainable set of operations and database for modeling partial faults it would be very helpful to the geological society since currently there are no modeling frameworks that support this feature.

Also, there are some future focus points that will not be studied in this research, but they can help with the progress of this area:

- **User Interface:** We did not focus on the different gestures that could be set for the program for better interactions with the users. This is a crucial part of the whole program and can be studied rigorously.
- **Handling Ambiguities in the user inputs:** Though our interface will create the models using sketches, there is not that much focus on handling the ambiguities that could be caused by the user inputs, or specifying which types of inputs should not be deemed valid. Our main focus was mostly on the modeling part as opposed to dealing with the challenges that arise from having a sketch-based interface. In other words, in our research scope, we can model certain configurations using valid inputs, but cannot specify the invalid inputs.
- **Combining Fault Modeling with Stratigrphical Modeling:** Currently, our framework does not support modeling stratigraphy. However, being able to combine the two would make the modeling framework more complete and useful.

REFERENCES

[1] Ronan Amorim, Emilio Vital Brazil, Daniel Patel, and Mario Costa Sousa. 2012. Sketch modeling of seismic horizons from uncertainty. In *Proceedings of the International Symposium on Sketch-based interfaces and modeling*. 1–10.

[2] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. 2010. *Polygon mesh processing*. CRC press.

[3] E Vital Brazil, Ives Macedo, M Costa Sousa, Luiz Henrique de Figueiredo, and Luiz Velho. 2010. Sketching variational hermite-rbf implicits. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*. 1–8.

[4] Guillaume Caumon, PLCD Collon-Drouaillet, C Le Carlier De Veslud, Sophie Viseur, and J Sausse. 2009. Surface-based 3D modeling of geological structures. *Mathematical Geosciences* 41, 8 (2009), 927–945.

[5] Rodrigo S Ferreira, Julia Noce, Dario AB Oliveira, and Emilio Vital Brazil. 2019. Generating Sketch-Based Synthetic Seismic Images With Generative Adversarial Networks. *IEEE Geoscience and Remote Sensing Letters* 17, 8 (2019), 1460–1464.

[6] Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

[7] MD Jackson, Gary J Hampson, Daisy Rood, Sebastian Geiger, Zhao Zhang, Mario Costa Sousa, R Amorim, E Brazil, FF Samavati, and LN Guimaraes. 2015. Rapid reservoir modeling: prototyping of reservoir models, well trajectories and development options using an intuitive, sketch-based interface. In *SPE Reservoir Simulation Symposium*. OnePetro.

[8] Alec Jacobson and Daniele Panozzo. 2017. Libigl: prototyping geometry processing research in C++. In *SIGGRAPH Asia 2017 courses*. 1–172.

[9] Carl Jacquemyn, Margaret EH Pataki, Gary J Hampson, Matthew D Jackson, Dmytro Petrovskyy, Sebastian Geiger, Clarissa C Marques, Julio D Machado Silva, Sicilia Judice, Fazilatur Rahman, et al. 2021. Sketch-based interface and modelling of stratigraphy and structure in three dimensions. *Journal of the Geological Society* (2021).

[10] William E Lorensen and Harvey E Cline. 1987. Marching cubes: A high resolution 3D surface construction algorithm. *ACM siggraph computer graphics* 21, 4 (1987), 163–169.

[11] Ives Macedo, Joao Paulo Gois, and Luiz Velho. 2011. Hermite radial basis functions implicits. In *Computer Graphics Forum*, Vol. 30. Wiley Online Library, 27–42.

[12] Jean-Laurent Mallet. 2008. Numerical Earth Models (EET 3).

[13] Max McGuire. 2000. The half-edge data structure. *Website: http://www. flipcode. com/articles/article halfedgepf. shtml* (2000).

[14] Suellen Motta, Marcelo Gattass, and Deane Roehl. 2019. Sketch-based modeling of salt domes. In *SEG International Exposition and Annual Meeting*. OnePetro.

[15] Luke Olsen, Faramarz F. Samavati, Mario Costa Sousa, and Joaquim A. Jorge. 2009. Sketch-based modeling: A survey. *Computers Graphics* 33, 1 (2009), 85–103. https://doi.org/10.1016/j.cag.2008.09.013

[16] Sang C Park. 2004. Triangular mesh intersection. *The Visual Computer* 20, 7 (2004), 448–456.

[17] John G Ramsay and MARTIN I Huber. 1987. Modern structural geology. *Folds and Fractures* 2 (1987), 309–700.

[18] Scott D Roth. 1982. Ray casting for modeling solids. *Computer graphics and image processing* 18, 2 (1982), 109–144.

[19] Faramarz F Samavati and Richard H Bartels. 2004. Local filters of b-spline wavelets. In *Proc. of Intl. Workshop on Biometric Technologies*. 105–110.

[20] The CGAL Project. 2021. *CGAL User and Reference Manual* (5.3 ed.). CGAL Editorial Board. https://doc.cgal.org/5.3/Manual/packages.html

[21] Ben A Van der Pluijm and Stephen Marshak. 2004. Earth structure. *New York* (2004).

[22] Ingo Wald, Sven Woop, Carsten Benthin, Gregory S Johnson, and Manfred Ernst. 2014. Embree: a kernel framework for efficient CPU ray tracing. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–8.

[23] Qiang Wu and Hua Xu. 2003. An approach to computer modeling and visualization of geological faults in 3D. *Computers & Geosciences* 29, 4 (2003), 503–509.

[24] HU Zhiwei, XU Changgui, WANG Deying, REN Jian, LIU Yubo, XIAO Shuguang, and ZHOU Xin. 2019. Superimposed characteristics and genetic mechanism of strike-slip faults in the Bohai Sea, China. *Petroleum Exploration and Development* 46, 2 (2019), 265–279.