Report

D3b decimation of height fields with missing values

Author(s):
Hubeli, Andreas; Hammersley, Richard

Publication Date:
2002

Permalink Link:
https://doi.org/10.3929/ethz-a-006654442

Rights / License:
In Copyright - Non-Commercial Use Permitted
D3b
Decimation of Height Fields With Missing Values

Andreas Hubeli†, Richard Hammersley*

†Computer Science Department
ETH Zurich, Switzerland
e-mail: hubeli@inf.ethz.ch
http://graphics.ethz.ch

*Schlumberger Austin Technology Center
Austin, TX, 78726
USA
D3b: Decimation of Height Fields With Missing Values

Andreas Hubeli
Department of Computer Science
ETH Zurich, Switzerland

Richard Hammersley
Schlumberger ATC
Austin, TX, USA

Abstract

We present a novel framework to build topologically correct continuous level of detail approximations of height field data that contains missing values. The framework is based on an unconstrained Delaunay triangulation algorithm, which enables us to construct approximations efficiently. The extra steps required to handle topologically non-trivial datasets are efficient, keeping the overhead to a minimum. Our algorithm is general in the sense that it only requires the input surface to be connected, but no assumptions are made on the topological type or the size of the input data.

Most of the results that we will present in this paper have been generated from geological datasets. However, the techniques we describe are inherently application independent.

1 Introduction

Geometric models, usually represented by a collection of triangles, are becoming increasingly complex, as acquisition systems improve. In order to make use of this data in interactive environments it has to be reduced in size. This is the objective of surface simplification techniques, which take surfaces as input and return arbitrarily simplified meshes as output.

Initially, research focused on handling regular height field data [8], [7] and later moved to more general topology, such as two-manifold surfaces [10], [6], [14]. However, height field data remains important, as it is still used in many fields, such as geoscience. Furthermore, there are still a few outstanding challenges that must be addressed.

An interesting problem that has not been fully investigated in the literature is how to simplify height fields which contain missing values, the focus of this paper. In particular, we are interested in the construction of an algorithm that generates topologically correct continuous LOD approximations efficiently. A topologically correct simplification algorithm must be able to detect and avoid self-intersections of the boundaries in the parameter space of any approximation [2]. Figure 2 depicts a simplification step that introduces a self-intersection.

The standard simplification operators presented in the literature cannot handle the self-intersection problem well, and consequently they cannot guarantee intersection-free approximations in image space. This problem can be attacked using two strategies:

I. Self-intersections can be detected by explicitly checking the result of every simplification operation [12]. This approach is not efficient, since the operators used to perform these tests have a global support on the mesh.

II. A more efficient approach consists in triangulating the entire domain of a surface, i.e. its interior and its holes. The result of every simplification operation can then be tested for self-intersections using local operators.

Consider the example illustrated in figure 2: the vertex \( x_{ij} \) is removed by collapsing it onto \( x_{kl} \). This operation introduces a self-intersection in the boundary, which is reflected in the change of the orientation of the grey triangle.

In the following we will present D3b, a simplification algorithm that generates arbitrary approximations of height field data that are guaranteed to be topologically correct. The framework is based on the second approach just discussed.

The paper is structured as follows: section 2 will present a brief survey of the previous work in this field. An overview of the D3b framework will be presented in section 3, followed by a more detailed description of the algorithm in section 4 and section 5. In section 6 we will present and discuss some of the results generated by D3b. Finally, section 7 will draw conclusions and propose some future work.
I. Our framework handles height fields with non-trivial topology, techniques can be described as follows: The D3b framework is based on a vertex removal strategy, as described in section 1 has to be implemented.

II. Our technique handles a large class of height fields, making few assumptions on the input data. Currently, we only require the input surface to be connected, but make no assumptions on its topological type nor on the size of the underlying grid. II. Our technique handles a large class of height fields, making few assumptions on the input data. Currently, we only require the input surface to be connected, but make no assumptions on its topological type nor on the size of the underlying grid.

III. Our algorithm is efficient, both in speed and in memory requirements. Being based on a Delaunay algorithm, the framework will generate approximations that possess fair triangulations, provided the input surface is smooth and the parameterization has been chosen carefully.

2 Previous Work

Surface simplification is an important core technology in many applications, and as a consequence many different research groups have focused on creating flexible and efficient frameworks to handle large datasets. We will briefly discuss the approaches most relevant to this paper.

In the pioneering paper [10], the authors introduced the fundamental vertex removal operator. Surfaces are simplified using this operator by removing one vertex at a time and then re-triangulating the resulting hole using a divide-and-conquer strategy. This approach can be used to simplify two-manifold surfaces, but it has also been applied to solve special problems such as the construction of multiresolution representations of height field data. The D3b framework belongs to this class of algorithms.

The second fundamental surface simplification operator is the edge collapse [6], [3], which simplifies meshes by collapsing pair of vertices joined by an edge. This approach has the advantage that no re-triangulation is required after the simplification step. On the downside, meshes generated by these techniques might have poor characteristics, unless the set of allowable edge collapses is chosen carefully.

While the previous operators were generally applicable to two-manifold surfaces, much effort has been invested in the construction of representations for height field data stored in regular grids. In [4], the authors used a wavelets representation of height fields coupled with a look-up table to construct multiresolution approximations of data defined over a regular domain. In [9], a quadtree was used to store surfaces. By restricting the choice of the vertices present in any approximation, the authors could define a look-up table capable of triangulating the chosen set of vertices while generating a single triangle strip. Both approaches have the advantage of being very fast, since they are optimized for height field data, but they have the disadvantage of working best with grids whose size is a power of two. If this is not the case, then it is necessary to extend them to handle many different special cases. For the same reason, it would not be efficient to let these structures handle height fields with missing values.

The surface representations discussed in this section guarantee that a simplification operation will not change the topology of the surface locally. However, the self-intersection problem is not handled explicitly by any representation. If an application requires topologically consistent approximations, one of the approaches described in section 1 has to be implemented.

3 The D3b Framework

The D3b framework is based on a vertex removal strategy, as introduced in [10], combined with an unconstrained Delaunay triangulation algorithm, which computes the connectivity of sets of vertices. The advantages and novelties of D3b versus previous techniques can be described as follows:

1. Our framework handles height fields with non-trivial topology, and it generates topologically correct continuous level of detail approximations. The operations required to guarantee correctness are strictly local, and do not impact the overall performance of the algorithm. Furthermore, the performance loss in the run-time component of the algorithm is negligible: once the simplification information has been constructed, approximations can be generated efficiently.

2. Our technique handles a large class of height fields, making few assumptions on the input data. Currently, we only require the input surface to be connected, but make no assumptions on its topological type nor on the size of the underlying grid.

3. Our algorithm is efficient, both in speed and in memory requirements. Being based on a Delaunay algorithm, the framework will generate approximations that possess fair triangulations, provided the input surface is smooth and the parameterization has been chosen carefully.

The D3b framework is subdivided into two components, described by the pipelines shown in figure 3:

- In the pre-computation step the input data is analyzed, and an importance value is assigned to every vertex in the mesh. These values depend on the underlying error norm, but also on the requirement that we only allow topologically consistent approximations. Furthermore, the strategy used to analyze the input surface enables us to minimize the overhead incurred by the run-time component of the framework. Detailed information on this first step is provided in section 4.

- The run-time component uses the information computed in the previous step to construct topologically correct continuous level of detail approximations of the input data efficiently. Efficiency is the most important requirement to this second component since it will be used in interactive environments. Section 5 will describe this step in detail.

4 The Pre-computation Pipeline

The goal of the pre-computation step is to analyze the input height field and to generate an importance value per vertex. This operation must be implemented efficiently, since we need to handle large grids. However, the efficiency of the run-time pipeline is of paramount importance, since it is used in interactive environments. Consequently, extra effort will be invested in this first pipeline to optimize the second pipeline. In the following subsections we will describe the tasks performed in each stage of the pre-computation.

4.1 Overview

A high-level overview of the pre-computation process is provided by the following pseudo-code fragment:

```c
void D3b_Precompute(heightField hF) {
    // Section 4.3
    prefilterGrid(hF);
    // Section 4.4
    bnd = traceBoundaries(hF);
    // Section 4.5
    neighborhood = triangulateSurface(hF);
    // Section 4.6 & 4.7
    for all vertices x in the surface do
        updateVrtError(x, neighborhood, bnd, errTable);
        while errTable is not empty and minError != \infty do |
```
The information required by the algorithm comprises the input height field, the neighborhood of every vertex and the boundaries of the surface. The neighborhood of a vertex is stored as an array of vertex indices sorted counterclockwise. The boundary structure maps vertices to their boundary index. The vertices in each full-resolution boundary are numbered, starting from any vertex and moving counterclockwise on the boundary.

The goal of the pre-computation is the construction of three structures: the set of vertices required to build a topologically correct approximation, a vector of vertices sorted by their importance and an array that stores the error associated with every vertex in the surface.

### 4.2 Error norms

In D3b we provide three error norms, which were used to generate the results presented in this paper:

- **Distance norm:** the error associated to a vertex \( x_{ij} \) is proportional to the distance between \( x_{ij} \) and the surface obtained by the removal of \( x_{ij} \) \([10]\).
- **Area norm:** the error is proportional to the change in the surface area that the removal of the vertex \( x_{ij} \) produces.
- **Volume norm:** the error is proportional to the change in volume caused by the removal of the vertex \( x_{ij} \).

The use of different error norms leads to different importance arrays being constructed, and therefore to different approximations. Figure 4 displays the approximations of an height field with a single hole computed using the three norms.

![Figure 4: Approximation generated using different error norms:](image)

- a) Distance norm.
- b) Area norm.
- c) Volume norm.

### 4.3 Grid pre-filtering

If the simplification of an height field is interpreted as an irregular subsampling operation, then the grid must be pre-filtered before the simplification step. In our framework, we used Taubin fairing \([13]\) to remove high frequencies from the surface. A vertex is added to the simplification step. In our framework, we used Taubin fairing \([13]\) to remove high frequencies from the surface. A vertex is added to the simplification step.

The first two steps of the pipeline, described in section 4.3 and section 4.4, are applied to the whole grid and use the information provided by the missing values. The remaining pipeline stages will possess complex boundaries. These boundaries are not readily available, and must be extracted from the available data. Although the input data is stored in an height field, we are interested in its full-resolution triangle mesh representation, which will be iteratively simplified in the pipeline. Therefore, the neighborhood information used in the tracing process is computed on a triangulation of the height field, which must be compatible with the convex hull triangulation computed in section 4.5.

The tracing algorithm is subdivided into two components. First, the algorithm scans the grid for a vertex \( x_{ij} \) on a boundary that has not already been traced. A vertex lies on a boundary if one of its neighbors is a missing value. Next, the boundary is traced starting from \( x_{ij} \). To this end, we apply the following rule: consider the neighboring vertices of \( x_{ij} \), sorted counterclockwise. The \( k \)-th neighbor of \( x_{ij} \) is also a boundary vertex if and only if:

- the \((k-1)\)-st and the \(k\)-th neighbors of \( x_{ij} \) are not missing values and the \((k+1)\)-st neighbor is a missing value, or
- the \((k+1)\)-st and the \(k\)-th neighbors of \( x_{ij} \) are not missing values and the \((k-1)\)-st neighbor is a missing value.

These tests are evaluated modulo the number of neighbors of \( x_{ij} \).

Consider the configuration illustrated in figure 5. Our tracing algorithm guarantees to trace the boundary of the grey surface correctly (figure 5.a), while simpler approaches could generate the boundary depicted in figure 5.b.

![Figure 5: Problematic boundary configuration:](image)

- a) Correct boundary traced by our strategy.
- b) Incorrect boundary traced by a naive strategy.

Note that the strategy we discussed assumes that the boundary of the vertex \( x_{ij} \) is closed, i.e. that \( x_{ij} \) does not lie on the boundary of the grid. This assumption does not limit the applicability of the approach, since the boundaries of the surface must be traced only up to the boundaries of the grid. Furthermore, this approach performs correctly only if \( x_{ij} \) is not a pseudo-manifold. A pseudo-manifold singularity can be interpreted as a vertex in the grid that belongs to two or more boundaries of the surface.

### 4.4 Tracing the boundaries

Surfaces stored in height fields with missing values potentially possess complex boundaries. These boundaries are not readily available, and must be extracted from the available data. Although the input data is stored in an height field, we are interested in its full-resolution triangle mesh representation, which will be iteratively simplified in the pipeline. Therefore, the neighborhood information used in the tracing process is computed on a triangulation of the height field, which must be compatible with the convex hull triangulation computed in section 4.5.

The tracing algorithm is subdivided into two components. First, the algorithm scans the grid for a vertex \( x_{ij} \) on a boundary that has not already been traced. A vertex lies on a boundary if one of its neighbors is a missing value. Next, the boundary is traced starting from \( x_{ij} \). To this end, we apply the following rule: consider the neighboring vertices of \( x_{ij} \), sorted counterclockwise. The \( k \)-th neighbor of \( x_{ij} \) is also a boundary vertex if and only if:

- the \((k-1)\)-st and the \(k\)-th neighbors of \( x_{ij} \) are not missing values and the \((k+1)\)-st neighbor is a missing value, or
- the \((k+1)\)-st and the \(k\)-th neighbors of \( x_{ij} \) are not missing values and the \((k-1)\)-st neighbor is a missing value.

These tests are evaluated modulo the number of neighbors of \( x_{ij} \).

Consider the configuration illustrated in figure 5. Our tracing algorithm guarantees to trace the boundary of the grey surface correctly (figure 5.a), while simpler approaches could generate the boundary depicted in figure 5.b.

Note that the strategy we discussed assumes that the boundary of the vertex \( x_{ij} \) is closed, i.e. that \( x_{ij} \) does not lie on the boundary of the grid. This assumption does not limit the applicability of the approach, since the boundaries of the surface must be traced only up to the boundaries of the grid. Furthermore, this approach performs correctly only if \( x_{ij} \) is not a pseudo-manifold. A pseudo-manifold singularity can be interpreted as a vertex in the grid that belongs to two or more boundaries of the surface.

### 4.5 Triangulation of the full resolution surface

The first two steps of the pipeline, described in section 4.3 and section 4.4, are applied to the whole grid and use the information provided by the missing values. The remaining pipeline stages will work on the surface represented in the height field data. To this end, the missing values must be discarded from the grid, and a valid triangulation of the remaining vertices has to be computed.
**Definition:** The Delaunay triangulation of a point set is a triangulation with the property that no point in the point set falls in the interior of the circumcircle of any triangle [11]. The circumcircle of a triangle is defined as the circle that passes through its three points.

We use an unconstrained Delaunay triangulation to compute the convex hull triangulation of the vertices, which is defined as the complete triangulation of the convex hull of the vertices present in the surface. Thus, we also triangulate the holes of the surface.

We decided to use a Delaunay algorithm for several reasons. First, by choosing an appropriate parameterization of the vertices, such as the one presented in the appendix, it is possible to generate a unique triangulation of the vertices. Furthermore, local operations such as the removal of a vertex in the surface result in local changes of the triangulation. This allows us to make use of incremental and decremental update algorithms. Finally, fast and robust implementations of the full Delaunay triangulation algorithm are readily available.

### 4.6 Update of the error table

This pipeline step is responsible to compute or update the error associated with the vertices in the surface. The error of a vertex can be interpreted as a value describing how much the surface would change if the vertex was to be removed from the mesh.

In order to compute the error of a vertex \( x_{ij} \), the error function must be provided with the current approximation of the surface and the approximation that would be generated by the removal of \( x_{ij} \). It is crucial to construct the new approximation efficiently, since this operation has to be performed every time a vertex is removed or the error table is updated.

It can be shown [5] that the removal of a vertex \( x_{ij} \) in a Delaunay triangulation only influences its neighbor simplices. Therefore, we can construct the new surface by removing \( x_{ij} \) and its neighboring triangles, and by re-triangulating the hole using a decremental Delaunay triangulation algorithm. In our framework we chose [1], which is guaranteed to generate a correct Delaunay triangulation efficiently.

Although detailed information on this technique can be found in the reference, it is helpful to understand the basics. Given a vertex \( x_{ij} \) and the sorted list of its neighbor vertices, a weight is first computed for all of the neighbors. A simplex is then constructed with the neighbor that possesses the smallest weight, and the previous and next vertices in the neighborhood of \( x_{ij} \). This operation is repeated until the neighborhood of \( x_{ij} \) has been fully re-triangulated, as illustrated in figure 6.

Before the first vertex removal operation is performed, the error table has to be initialized, and the error associated with every vertex must be computed. In all other instances, this step only updates the error associated with the neighbors of the last vertex removed.

### 4.7 Consistency check for boundary vertices

The topological consistency check has been separated from the error norm used in the previous pipeline step. Thus, application programmers are facilitated in the construction of custom error norms, because they do not need to understand the intricacies of the framework. The consistency of the information computed in the pipeline is guaranteed in a post-processing step. We defined a condition that allows us to build a simple and efficient run-time pipeline that produces topologically correct approximations.

**Condition:** a vertex removal operation is allowed if and only if no interior vertex of the surface before the removal operation becomes a boundary vertex afterwards.

If this condition is satisfied in all the stages of the pre-computation pipeline, the topology of any approximation is guaranteed to be correct. Furthermore, the boundary vertices of any approximation are a subset of the boundary vertices of the full-resolution surface, and every boundary edge is a Delaunay edge. These properties will be used in section 5.3 to construct approximations at run-time effectively.

Consider a boundary vertex \( x_{ij} \) and its two neighbors \( x_{kl} \) and \( x_{mn} \) on the boundary. The removal of \( x_{ij} \) satisfies the condition if two tests are passed:

- After the removal of \( x_{ij} \) an edge connects \( x_{kl} \) and \( x_{mn} \), and
- If the edge connecting \( x_{kl} \) and \( x_{mn} \) was present before the removal operation, then the vertices \( x_{ij} \), \( x_{kl} \), \( x_{mn} \) must have defined a triangle in the surface.

Figure 7 depicts a few admissible and non-admissible vertex removal operations. If a boundary vertex fails either test, then under no circumstance will it be removed from the surface. This is guaranteed by setting its associated error to infinity. If subsequent removal operations change the neighborhood of this vertex, the consistency check will be performed once more.

![Figure 7: Consistency check.](image)

**Figure 7:** a) Valid configurations. b) Invalid configuration.

### 4.8 Vertex removal operation

After the two steps described in section 4.6 and section 4.7 have been performed, the error associated to every vertex in the surface has been updated. In this pipeline stage the vertex \( x_{ij} \) with the smallest error is retrieved, removed from the mesh, and added to the front of the importance array. Refer to the pseudo-code presented in section 4.1 for an overview of this step.

The retrieval operation is performed efficiently by keeping the list of vertices in a priority queue sorted by their associated error. The removal operation is carried out as follows: first, the vertex \( x_{ij} \) and its neighbor triangles are removed from the surface. The
hole generated by this operation is triangulated using the algorithm described in section 4.6. The set of simplices generated by this process are then used to extract the new neighborhood information of the neighboring vertices \( x_{n_i} \) of \( x_{i,j} \). Each simplex is analyzed only once except for the last simplex, which is only required to remove vertices on the boundary of the convex hull triangulation. Figure 8 illustrates an example of this process. The operation is completed by replacing the entry \( x_{i,j} \) with the new neighborhood information of \( x_{n_i} \).

\[
\begin{align*}
  x_1 & N(x_{n_1}) = \{ \emptyset \} \\
  N(x_{n_1}) & = \{ x_{n_1} \} \\
  N(x_{n_2}) & = \{ x_{n_2} \} \\
  x_2 & N(x_{n_2}) = \{ \emptyset \} \\
  N(x_{n_2}) & = \{ x_{n_2} \} \\
  N(x_{n_3}) & = \{ x_{n_3} \} \\
  x_3 & N(x_{n_3}) = \{ \emptyset \} \\
  N(x_{n_3}) & = \{ x_{n_1}, x_{n_2} \} \\
  N(x_{n_4}) & = \{ x_{n_1}, x_{n_3} \} \\
  N(x_{n_4}) & = \{ x_{n_2}, x_{n_3} \}
\end{align*}
\]

**Figure 8:** Update of the neighborhood information.

The index of \( x_{i,j} \) and its associated error are stored at the front of the importance and error arrays. At the end of the pre-computation these arrays will store the information required to construct all the admissible approximations of the input surface.

Once the vertex has been removed from the surface, the error associated with the neighboring vertices \( x_{n_i} \) of \( x_{i,j} \) must be updated, as their neighborhood information changed.

Vertices are removed from the surface one at a time, until the error associated with all the remaining vertices is infinity. This stage represents the coarsest approximation that is still topologically correct. None of the remaining vertices can be removed without changing the topology of the surface and thus they must be added to the set of required vertices. Ideally, the coarsest approximation should not contain any internal vertices, and only three vertices per boundary. Although we cannot guarantee to obtain this optimal configuration, we reached it for all the input surfaces we tested. As an example consider the surface displayed in figure 1.

5.1 Overview

A high-level overview of the run-time component of D3b is provided by the following pseudo-code fragment:

```c
void D3b_Runtime(float userParam) {
    // Section 5.2
    vertices = selectVertices(userParam);
    simplices = convexHullTriangulation(vertices);

    // Section 5.3
    for all triangles t in simplices do
        if (isExtraTriangle(t) == true)
            removeTriangle(t, simplices);
}
```

This component of the framework requires the height field data, the boundary information and the three vectors computed by the pre-computation pipeline to construct approximations. The largest structure used in the pre-computation, the neighborhood information, is not required since the triangles of an approximation are constructed using a full Delaunay triangulation algorithm.

5.2 Construction of the convex hull

The construction of an approximation is initiated by the user, who specifies the error that the approximation must satisfy or the number of vertices \( n \) that the approximation must contain.

The first step in the pipeline consists in the selection of the vertices that will be used in the construction of the approximation. The set of required vertices is selected, since it is needed to construct topologically correct approximations. The remaining vertices are extracted from the importance array, starting from the most important vertices.

The set of vertices is then triangulated using a full Delaunay triangulation algorithm. The triangulation generated by the algorithm must be unique and compatible with one of the triangulations generated in the pre-computation step. To this end, we chose the parameterization presented in the appendix. The use of a consistent and unique triangulation in the pre-computation and in the runtime components of the framework guarantees that any convex hull triangulation generated at run-time was also generated by the pre-computation pipeline.

5.3 Removal of extra triangles

The result of the previous operation is the convex hull triangulation of a set of vertices. From section 4.7 we known that all the boundary edges of the approximation are present in this triangulation. Therefore, the correct approximation can be constructed by removing the extra triangles that cover the holes in the surface.

In order to simplify the description of this step, let us assume that the vertices in each full-resolution boundary are numbered, from \( l \) to \( n \), starting from a vertex and moving counterclockwise on the boundary, i.e. the interior of the surface is defined on the left-hand side of the boundary. Figure 10 depicts an example of how the vertices of a boundary are to be numbered. Based on this assumption, extra triangles can be identified with two efficient tests:

I. The three vertices that define the triangle must lie on the same boundary of the surface. This is only true if the surface is connected, since if disconnected components are present, an extra triangle might exist with vertices on different boundaries.

II. The three vertices that define the triangle have decreasing boundary indices. If the boundary is closed, this check is performed modulo the total number of vertices \( n \) in the boundary.

5 of 8

**Figure 9:** Construction of an approximation:

- a) Select a set of vertices.
- b) Compute the triangulation of the convex hull.
- c) Remove any extra triangles.
Figure 10 shows a simple boundary configuration, and how extra triangles are identified and removed from an approximation.

![Figure 10: Removal of extra triangles.](image)

The efficiency of this step is the direct consequence of the removal policy implemented in the pre-computation. The condition tested in section 4.7 guarantees that all boundary edges are present in the triangulation of any approximation. Therefore, we do not need to use a constrained Delaunay triangulation algorithm. Furthermore, by assuming that the surface is connected, the vertices of any extra triangle must lie on the same boundary. This reduces the identification process to computation-efficient checks of the orientation of the triangle vertices in the boundaries.

6 Results & Discussions

In this section we present and discuss the results computed with D3b on height field data that represents interfaces between geological layers. The missing values represent regions of space where no data could be collected or where other formations punctured through the surface.

The height field shown in figure 11 is not very complex from a topological point of view, but it possesses complex boundaries. By handling these boundaries correctly, we could simplify them without introducing self-intersections and without performing global operations. In figure 11.b-c the complexity of the boundaries has been reduced significantly, but the overall shape of the surface has been preserved.

The next surface, presented in figure 12, is topologically more complex. A salt dome perturbs the acquisition process, which resulted in a surface that contains multiple undefined regions. Our framework can now be applied to the data to generate arbitrarily coarse approximations that are still topologically identical to the input surface, and its boundaries are guaranteed not to self-intersect in parameter space. Furthermore, we optimized the run-time component of our framework by applying extra tests during the pre-computation pipeline. This strategy resulted in the construction of an efficient run-time algorithm capable of handling height field data with missing values with an overhead of less than 1%. The D3b algorithm has been included into the Common Model Builder (CMB), a commercial application of Schlumberger ATC, and it has been tested on many different surfaces. The technique is simple, stable and efficient, and it can be applied on a large collection of height fields. The error norm has been abstracted out of the framework, thus making it application independent.

There are a few research directions and improvements we would like to incorporate into the representation:

I. **Controlled topology simplification**: We constructed D3b with the specific goal in mind to preserve the topology of the input data. However, an operator that allows controlled topology simplification could be useful in many applications.

II. **Flexibility**: We would like to extend the framework to handle arbitrary parameterized surfaces. The main challenge consists in designing a parameterization guaranteed to generate a unique triangulation of any set of vertices.

III. **Robustness**: We would like to handle surfaces with disconnected components or pseudo-manifold singularities. In the current implementation of the algorithm, we remove pseudo-manifolds in a pre-processing step, and create different instances of D3b to process each of the disconnected components of a surface separately.

7 Conclusions and Future Work

In this paper we presented a framework capable of constructing continuous level of detail approximations of height field data that contains missing values. The key result is that any approximation is topologically identical to the input surface, and its boundaries are guaranteed not to self-intersect in parameter space. Furthermore, we optimized the run-time component of our framework by applying extra tests during the pre-computation pipeline. This strategy resulted in the construction of an efficient run-time algorithm capable of handling height field data with missing values with an overhead of less than 1%. The D3b algorithm has been included into the Common Model Builder (CMB), a commercial application of Schlumberger ATC, and it has been tested on many different surfaces. The technique is simple, stable and efficient, and it can be applied on a large collection of height fields. The error norm has been abstracted out of the framework, thus making it application independent.

There are a few research directions and improvements we would like to incorporate into the representation:

I. **Controlled topology simplification**: We constructed D3b with the specific goal in mind to preserve the topology of the input data. However, an operator that allows controlled topology simplification could be useful in many applications.

II. **Flexibility**: We would like to extend the framework to handle arbitrary parameterized surfaces. The main challenge consists in designing a parameterization guaranteed to generate a unique triangulation of any set of vertices.

III. **Robustness**: We would like to handle surfaces with disconnected components or pseudo-manifold singularities. In the current implementation of the algorithm, we remove pseudo-manifolds in a pre-processing step, and create different instances of D3b to process each of the disconnected components of a surface separately.

Acknowledgment

We would like to thank Markus Gross and Karen Lu for their helpful comments on the text.

Literature


Figure 11: Geological surface with complex boundaries simplified using the distance norm:
   a) Input surface. b) Approximations with 10% of the vertices. c) Approximations with 1% of the vertices. d) Wireframe of c).

Figure 12: Geological surface punctured by a salt dome simplified using the distance norm:
   a) Input surface, containing 45410 vertices. b) Filtered surface. c) Approximation with 10% of the vertices. d) Approximation with 1% of the vertices. e) Wireframe of d).

Figure 13: Geological surface simplified using the volume norm:
   a) Input surface, containing 180889 vertices. b) Filtered surface. c) Approximation with 10% of the vertices. d) Approximation with 1% of the vertices. e) Wireframe of d).
Appendix

In this appendix we introduce the parameterization strategy we used in our framework. In our investigation we will consider an height field of size \(s_x \times s_y\), further assuming \(s_x \geq s_y\).

Initially, we analyzed the standard parameterization \(P_1(x_{ij})\) depicted in figure 14.a. The advantage of this approach is that it is simple and it works with integer arithmetic. The main disadvantage is that it is ridden with degenerate point configurations where four or more points lie on a circle. In a Delaunay-based algorithm, this is not acceptable, since in these cases the triangulation is not unique.

The first inequality in (6) can be satisfied by setting the parameter \(\Delta\) to

\[
(0, s_y) (s_x, s_y)
\]

\[
(0, 0) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x, s_y + \Delta s_x)
\]

\[
(0, 0)
\]

\[
(0, s_y) (s_x, 0)
\]

\[
(s_x + \Delta s_x, s_y)
\]