University of West Bohemia
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

Modeling and visualization of changes of geometric models caused by small tools and erosion

The State of the Art and Concept of Ph.D. Thesis

Václav Purchart

Technical Report No. DCSE/TR-2013-01
January, 2013

Distribution: public

Technical Report No. DCSE/TR-2013-01
January 2013

Modeling and visualization of changes of geometric models caused by small tools and erosion

Václav Purchart

Abstract

Terrain modeling has been an important area of computer graphics and GIS for more than 20 years and despite this long history many problems still prevail. One of the important challenges is an interactive physics-based authoring and editing of large scale terrains. Most of published work use a regular grid to represent a terrain elevation data. Due to the simplicity of this representation, visualization and processes such as editing, thermal erosion, water erosion, etc. could be easily and efficiently implemented. However, such a regular structure cannot provide an adaptive level of detail. Moreover, as the resolution of the data structure is fixed, a smallest detail of the terrain is essentially defined and the mesh cannot capture any finer features which are clearly visible on close-ups (without enormous memory requirements). This problem could be solved by using irregular (or pseudo-irregular) structures such as a triangulation, quad-tree, etc. But it brings new problems and challenges.

This work describes the state of the art of interactive terrain modeling. We will mostly focus on terrains covered by granular materials such as, e.g., sand. This work also contains description of our method for interactive modeling and editing of sand-covered terrains based on triangulated irregular network (TIN).

First we describe basic data structures used for granular terrain representations in computer graphics, followed by a minimum geometry knowledge about the Delaunay triangulation and triangulations in general. Next, methods for interactive modeling of sand are described in detail. Then we concentrate on the state of the art of the terrain erosion modeling in computer graphics and state of the art of haptic visualization of the terrains. Then our improvements and published algorithms are presented.

This work was supported by the Ministry of Education of the Czech Republic: Project Kontakt No. ME09051 (Modeling of natural phenomena using computational geometry), Project Kontakt No. LH11006 (Interactive geometric models for simulation of natural phenomena and crowds), by the Grant Agency of the Czech Republic: Project No. 201/07/0927 (Triangulated Models for Haptic and Virtual Reality), and the project SGS-2010-028 (Advanced Computing and Information Systems).

Contents

1 Introduction

This work contains mainly state of the art of interactive sand-covered terrain modeling. The reader will be guided to the description of our method – an interactive sand-covered terrain model, which is based on a Delaunay triangulation, allows real-time thermal erosion modeling, and contains set of virtual tools which are used to deform the terrain by the user. These tools can be optionally controlled with the haptic device which provides a force feedback to the user and adds more fidelity to the simulation process.

Terrain modeling and model editing in general is an important part of computer graphics that is used in wide range of disciplines (such as GIS, game development, physical simulations, CAD, etc.). The basis of such methods and applications is a deformable model (DM) which will be described later.

The most often data structure for an interactive terrain modeling is a regular grid. It is mainly used for its implementation simplicity and it is ideal for small, uniformly distributed terrain. When the terrain model becomes too variable, large parts of the grid structure become redundant and for a larger terrain data the terrain model memory consumption is high. This can be solved by compression of such a structure. But these methods lost the simplicity and efficiency of the grid. The task can be also solved by using an irregular data structure (such as, e.g., tree structures, or triangulations). These data structures are more memory efficient (in more cases) and scalable but they have some major drawbacks. For uniform data they have certain geometric overhead, the implementation is not so straightforward as for a grid, and they have performance handicap for global deforming operations.

2 Data Structures for Sand Modeling

The most important part of each method for the terrain modeling is a data structure used to store terrain data. Usability, performance, and efficiency of the method depend on the qualities of this data structure. Therefore, this section contains an overview of existing data structures which are used for the modeling of sand. In the terrain modeling in general the terrain representation is known as the Digital Terrain Model (DTM). An overview of digital terrain modeling could be found in the paper published by Weilber and Heller [66]. As we are choosing a suitable DTM approach, we have to consider these general tasks related to the terrain modeling:

- *DTM generation*: model construction; sampling of the original terrain data,
- *DTM manipulation*: modification and refinement of the terrain model,
- *DTM interpretation*: analysis and information extraction from DDM,
- *DTM visualization*: graphical rendering of DDM and derived information,
- *DTM application*: development of appropriate application for specific discipline. Each application has its own specific requirements.

2D versus 3D Terrains

The first thing that we need to take into account in a terrain (especially sand) modeling is the dimension of samples of a terrain shape. Basically, there are two ways to sample a terrain data – 2D and 3D. The 3D approach is obviously more general and it can capture any shape of the terrain. Unfortunately, the computation and memory costs of such a terrain model could be very high as it is mentioned in [63].

For many applications the 2D model is sufficient even though it cannot describe cavities (e.g., overhangs and tunnels). Such a model considers only the surface of the terrain and is often called *2.5D model* or *Digital Elevation Model* (DEM). For a sand the 2D model is sufficient (as it is mentioned in [6]), because sand does not contain cavities in a steady state. The nature of a sand is to smooth any eventual abrupt terrain features.

2.1 Static Terrains

In the following subsections we will mention terrain models from the view of their time variety. Let us first consider the most simple case – the static terrain model which does not change its shape in time and which is homogeneous. Note that such a representation is often used in GIS-related methods.

Grid-based Terrains

As it has been said, in the interactive terrain modeling techniques the most used data structure for the terrain representation is a regular grid also called *heightmap* or *heightfield* (see Figure 2.1, [5, 6, 7, 4, 17, 64, 39, 42, 41, 68]). Heightmap models are usually represented as a raster image which is used to store a surface elevation data – each cell indicates the height of the terrain in a given location. Such models have uniformly distributed height samples, they are easy to implement, they are fast, and provide reasonably realistic appearance. However, due to their nature, they are not capable of providing an adaptive level of the detail. The memory required for the grid is a quadratic function of the smallest detail that we want to preserve in the terrain. For example, grid with the sample size 1cm and the terrain size 10m requires 1M height samples. Such a resolution could be sufficient in the global scale, but it cannot hold details for close-ups. Another example where grid approaches fail is a terrain which is almost flat and in a few areas, there is a very detailed relief.

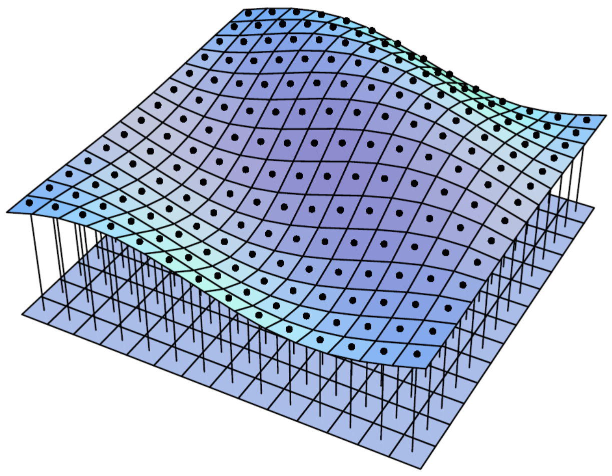


Figure 2.1: An example of the surface mapped by a regular grid [60].

Height-span maps

Because a heightfield is a single-valued function, on itself it cannot represent granular material on objects. Height-span maps (HS maps) are special matrices which overcome this restriction and which contain the height distribution of a given object. HS maps are somewhere between heightfield and voxel representation. The described model is represented by a grid, but each cell of this grid contains a transition list (the list of heights). So each cell stores multiple intervals as it is depicted in Figure 2.2. Therefore, unlike grid, such a representation can describe cavities. HS maps have been proposed by Onoue and Nishita [42] and further used, e.g., by Festenberg and Gumhold [64].

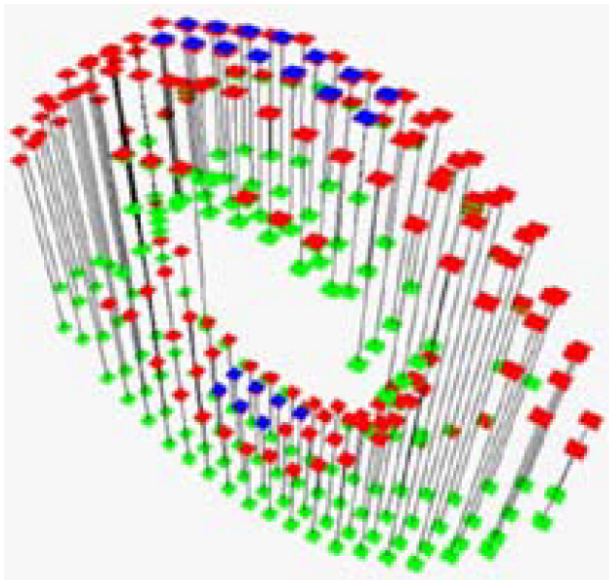


Figure 2.2: HS map of torus. Each red/green rectangle represents the upper/lower side of the height span, and each blue rectangle is for the upper limit of granular material on each height span [42].

Adaptive Terrains

Adaptive terrain representations are used much more rarely (e.g., [3, 8, 45, 46, 50, 59, 60, 63]). This fact could be caused mainly by their more difficult implementation. From irregular data structures, a quad-tree is the primary choice for adaptive graphics applications. GIS applications rather use triangulated terrains due to their adaptability. The use of complicated data structures brings some overhead, which means that we need more computation time (in comparison with the grid) and in certain cases such applications require more memory.

2.2 Dynamic Terrains

To model complex simulations it is necessary to incorporate changing of the model in time. For more sophisticated data structures it is not trivial to keep the model updated. The core of each dynamic terrain is the deformable model (DM). The DM describes properties, topology of the continuum and behavior of entities in the continuum in time. The term "deformable model" has been introduced by Kass, Witkin and Terzopoulos in [29] in 1988 for image segmentation.

In more general case the DM describes behavior of (at least) two entities variable in time with the dimension N. Interfaces of these entities have dimension $N - 1$. The solution of this problem usually cannot be solved exactly. Behavior is often described by a partial differential equation (PDE). Therefore, the PDE is solved numerically. Methods using DM can be subdivided into two groups according to a sampling of the continuum (or its parametrization) to Eulerian methods and Lagrandian methods.

2.2.1 Eulerian Approach

Eulerian methods regularly sample the continuum to the grid. Eulerian approach is also known known as *front capturing method* in computational physics. Two most important Eulerian methods are *level-set method* published by Osher and Sethian [43] and *volume-of-fluid method* published by Hirt and Nichols [27].

Level-set Method

This method samples the continuum in the regular intervals to the grid. Interfaces are described by the function Φ which is defined in hyperplane space of the interfaces. This allows to model complex topology changes, sharp interface corners, etc. Time complexity of the level-set method is $\mathcal{O}(N^2)$ in E^2, where N is the number of samples (grid cells), and $\mathcal{O}(N^3)$ in E^3. Because of huge memory demand, the space is usually restricted to 512^3 samples. More accuracy can be achieved by using the scene decomposition, and other compression schemes. Adalsteinsson and Sethian [2] published an optimized Fast Level-set method, where only some neighborhood of Φ is considered. This method has the time complexity $\mathcal{O}(N \cdot k)$, where k is the width of the neighborhood.

2.2.2 Lagrandian Approach

In contrast with the Eulerian approach, the Langrandian one does not use regular parametrization for PDEs. This results in irregularly placed sample points.

The Langrandian formulation explicitly tracks interfaces between different materials. There are two main classes of Lagrandian techniques: *mesh-based* methods (Brivio and Marini [9]) and *particle-based* methods (Cani-Gascuel and Desbrun [11]).

Mesh-based methods correspond to the ”snake” methodology in computer vision, and it is known as the *front tracking method* in computational physics. Some edges of the generated mesh represents interfaces between entities. This approach is able to avoid distortion while it can handle large deformations. On the other hand, mesh entanglements can be sources of numerical instabilities. Moreover, fully automatic, robust and efficient handling of topology changes in mesh-based methods remains an open issue.

Particle-based approach has gained popularity, because it can handle complex topology changes. However, it has problem with the localization of the interface and computation of interface properties such as the normal curvature is cumbersome (because of missing connectivity). It is usually used in the fluid simulation processes. The speedup can be achieved, e.g., by a subdivision of computational space.

Delaunay Deformable Models

Delaunay Deformable Models (DDM) is an efficient Langrandian approach for modeling of moving surfaces which can handle large deformations and topology changes. The work published by Pons and Boissonnat [46] uses a restricted Delaunay triangulation to model interfaces between materials.

3 Geometry for Sand Terrain Modeling

As mentioned in the Introduction, our method (which is described in detail in Section 7) adopts a 2.5D model model for the terrain representation as the most of the published work, but unlike existing methods our model is not sampled in regular intervals, but rather on-demand. On the set of these samples the Delaunay triangulation is computed and it maintains all the information about the terrain. The user can edit the terrain using a set of tools, which add constrains into the triangulation. This section and following subsections describe geometric state of the art necessary for subsequent explanation of our method.

3.1 Planar Triangulations

At first we formally define a planar triangulation:

Definition 3.1. *A triangulation $T(\mathcal{P})$ of a set $\mathcal{P}$ of n points in the Euclidean plane is a maximum set of edges $\mathcal{E}$ such that no two edges in $\mathcal{E}$ intersect at a point not in $\mathcal{P}$ and the edges in $\mathcal{E}$ divide the convex hull of $\mathcal{P}$ into triangles [13].*

A triangulation allows to model terrain features with the highest level of freedom (in comparison with other terrain representations). In a general planar triangulation triangles with an arbitrary shape can appear which can be a source of numerical instabilities. Such a triangulation allows creation of nearly singular triangles, or triangles which have some edges very long. Therefore, there are many types of triangulations which optimize the shape of triangles. It is proved [22], that *some* triangulation of the set of points $\mathcal{P}$ can be computed in time $\mathcal{O}(N \cdot logN)$, but for some criteria such an algorithm is not known or it is not possible.

There are many types of planar triangulations such as the greedy (GT) triangulation, the Delaunay triangulation (DT), the data dependent triangulation (DDT), the minimum weight triangulation (MWT), and others. In addition, triangulations could contain some constraints, e.g., the edges prescribed into a triangulation. The constraints ignore or modify a triangulation criterion in some way.

The MWT (or the optimal triangulation) is the triangulation which minimizes the total edge length. This is known to be NP-hard problem [38], therefore, it is not suitable for real-time applications.

The DDT takes into account also the third coordinate (the terrain height in the case of terrain modeling). More general criteria [18] can take into account for example curvatures of the resulting surface. DDT can be created by swapping of edges of another triangulation in time $\mathcal{O}(N^2)$ in the worst case and in $\mathcal{O}(N)$ in an average case.

The GT [16] can be computed in $\mathcal{O}(N \cdot logN)$ (for uniformly distributed data). It does not optimize any particular characteristic. It is important mainly due to its connection to MWT (GT is an approximation of MWT). It can be constructed by the following algorithm – we start with the empty set of edges and in each step we add the shortest compatible edge between two vertices. The compatible edge is an edge that does not intersects any previously inserted edges. GT can be also computed using DT or the Voronoi diagram.

The DT [13] maximizes the minimum angle of all the angles of the triangles in the triangulation. More important is that this global optimization is achieved by a simple local criterion. This triangulation tends to avoid skinny triangles if possible, which is also beneficial because it avoids numerical errors during computation on the resulting mesh. Another advantage is that in E^2 it can be computed in $\mathcal{O}(N \cdot logN)$. Now we will focus on DT, and define it more exactly.

3.2 Delaunay Triangulation

Definition 3.2. *The triangulation of a given set $\mathcal{P}$ of points in E^2 is called Delaunay triangulation $DT(\mathcal{P})$ if the circumcircle of any of its triangle does not contain any other point from $\mathcal{P}$. This empty cicumcircle criterion is also called Delaunay criterion [13].*

The DT criterion can be computed according to Equation 3.1.

$$\begin{vmatrix} A_x - D_x & A_y - D_y & (A_x - D_x)^2 + (A_y - D_y)^2 \\ B_x - D_x & B_y - D_y & (B_x - D_x)^2 + (B_y - D_y)^2 \\ C_x - D_x & C_y - D_y & (C_x - D_x)^2 + (C_y - D_y)^2 \end{vmatrix} = 0, \qquad (3.1)$$

where $A, B, C \in \mathcal{P}$ are vertices of some triangle from $DT(\mathcal{P})$.

This triangle meets the Delaunay empty circumcircle criterion if for each vertex D, where $D \in \mathcal{P} \wedge D \neq A, B, C$, the Equation 3.1 is met. The situation is depicted in Figure 3.1.

There are many algorithms to compute the Delaunay triangulation (DT) and we will mention the incremental insertion algorithm [13] that allows an easy insertion of a new point into the mesh with a minimal overhead. All changes of the terrain can be reduced to the adding and removing of vertices and edges into/from the mesh. In following sub-chapters we will describe the geometric operations used for the terrain modeling.

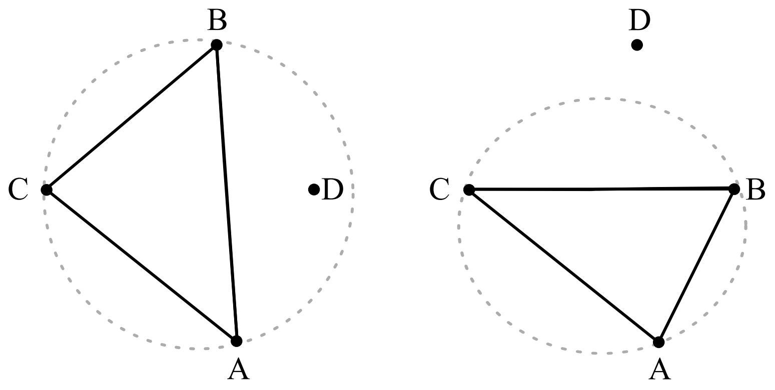


Figure 3.1: Delaunay circumcircle criterion. Left: Criterion is not met – point D lies in the circumcircle of triangle ABC. Right: Criterion is met.

3.3 Triangulation Construction

This section contains description of geometric operations which are used for the triangulation construction. Such operations are insertion of a new point, and insertion of the constrained edge.

3.3.1 Incremental Insertion Algorithm

We would prefer the construction algorithm which can add vertices into the triangulation on the fly. The incremental insertion algorithm can do that. It starts with the Delaunay triangulation of a convex hull or with the super-triangle. Super-triangle is an auxiliary triangle that contains all the vertices of the future triangulation.

If there is a request to add some vertex to the triangulation at coordinates $[x, y]$ we have to find its incident triangle (using triangle walking described in Section 3.4) and split this triangle into three new triangles (or two in a singular case), see Figure 3.2. The triangulation is then adjusted by flipping the edges so that the triangle mesh meets the Delaunay criterion again. This process is called *edge legalization* and is depicted in Figure 3.3.

3.3.2 Constrained Delaunay Triangulation

The Delaunay triangulation creates triangles which are as close to equilateral on the given set of points as possible. This is good for numerical computation, but in some cases we do not want to use some predefined set of edges. If we want to adjust the edges in the way that does not comply with the Delaunay criterion, we have two approaches how to do it.

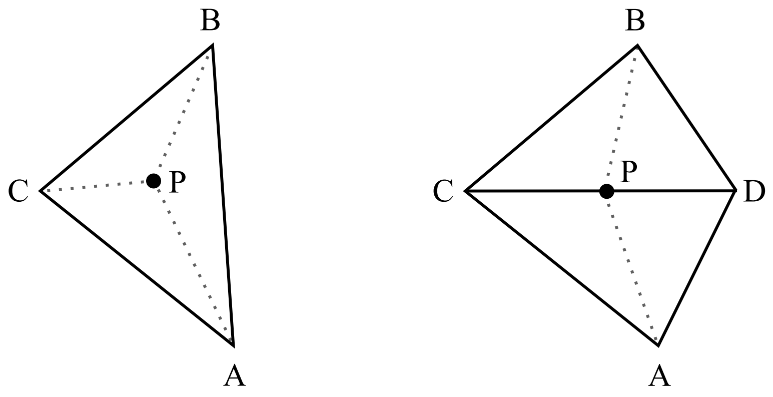


Figure 3.2: An example of triangle subdivision. Left: New point is inserted into the triangle ABC. Three new triangles are created. Right: New point is inserted on the edge CD. Both triangles must be subdivided. Four new triangles are created.

We can change the edges arrangement by adding new vertices sampling the predefined edge to the mesh in the way that the resulting DT will contain the desired edges. Such an approach was described, e.g., by J. R. Shewchuk [54] and is known as a *conforming DT*. This approach can use the existing DT and its construction without any modification, but by adding new vertices we increase the mesh complexity. This in result slows down further manipulation with the mesh.

Another approach is to enforce some edges into the mesh despite that they do not meet the Delaunay criterion. Such edges have a special flag that they have to stay in the triangulation although they do not fill the Delaunay criterion. Such edges are called *constrained edges*. Resulting triangulation is called the constrained Delaunay triangulation (CDT).

There are many algorithms to compute the CDT. We use the algorithm developed by S. W. Sloan [55], because it is fast and allows to add constrained edges to the already computed triangulation which can contain both normal and constrained edges.

Pseudocode of edge insertion is described in the Algorithm 3.1. Such inserted edges stay in the triangulation although they do not meet the Delaunay criterion. The inserted edge e is defined by two points P_1 and P_2.

We use a triangle walking algorithm to get the first incident triangle in which P_1 lies. Then the search continues through the neighbors of this triangle in the triangle-fan around point P_1. If a triangle that contains P_1 is found such that it intersects the edge e, the algorithm walks through all triangles via their neighbors in the direction to P_2. All the edges from these neighbors, which intersects e, are added into the queue Q. When some edge e_i is then dequeued from Q, the incident triangles form a quadrilateral. If this quadrilateral is convex, we can flip

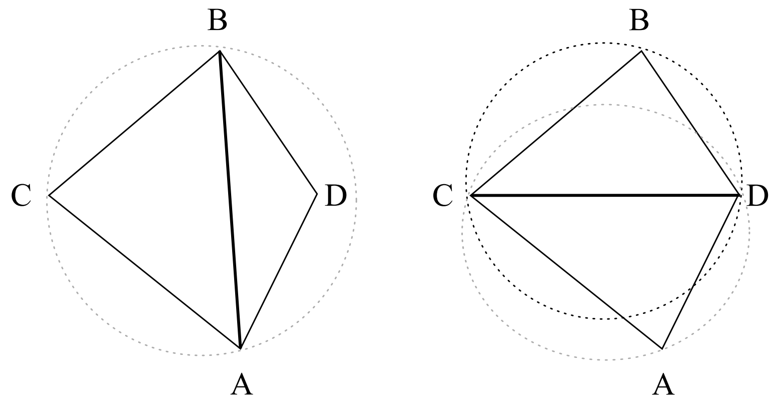


Figure 3.3: An example of the edge legalization for two triangles. Left: The configuration does not satisfy the empty circumcircle criterion, because the point D is inside of the circumcircle of the triangle ABC. Right: After the flipping of the edge shared by both triangles, the criterion is met for these triangles (both circumcircles of new triangles BCD and ADC are empty) and the check continues to the neighboring triangles.

e_i, so that it does not intersects edge e any more. When the queue is empty, it is guaranteed that there is no intersection [55] with e.

It may happen that some existing vertex in the mesh lies exactly on the edge e. In such a case we divide e into two constrained edges. Then all these edge-pieces are inserted into the mesh by the above described algorithm. When all edge-pieces are inserted, the algorithm ends and the mesh contains the required edge(s).

3.4 Point Location

Almost all operations which somehow modify the mesh need to locate the triangle which contains a given point. A triangle look-up is the most often used geometric operation, so it must be efficient. The mesh is constantly changing due to adding and removing vertices from the mesh. Therefore, additional search data structures would not be optimal for point location. A triangle walking algorithm, which only needs an information about neighbors of each triangle, is suitable for this task. The triangle walking algorithm works as it is described in Algorithm 3.2.

The key step in the Algorithm 3.2 is to get a starting triangle for the location (see Mücke at al. [37]). We want to choose the starting triangle as close to the target point as possible, because the walk is short (and fast) in this case. We get a random sample of triangles and compute the quadratic distance from the target point. The closest triangle (with the shortest distance) will be the starting triangle. The optimal size of the random set is $\sqrt[3.5]{n}$, where n is the number of

```
Input: Triangulation $CDT(\mathcal{P})$, desired new constrained edge $e = [P_1, P_2]$
Output: Triangulation $CDT(\mathcal{P})$ which contains $e$
Let $Q$ be queue of edges, and $F$ queue of flipped edges
Enqueue all existing edges which intersect $e$ into the $Q$
while $Q$ is not empty do
    Dequeue edge $e_i$ from $Q$
    if $e_i$ can be flipped then
        Flip $e_i$ Enqueue $e_i$ into $F$
    end
    else
        Enqueue $e_i$ into $Q$
    end
end
Add $e$ into the mesh
Legalize all edges in queue $F$
```

Algorithm 3.1: Adding constrained edge into the mesh.

```
Input: Triangulation $T(\mathcal{P})$, target point = $[x, y]$
Output: Target triangle $t$ in which $[x, y]$ lies
$t \leftarrow$ Get starting triangle
while $t$ does not contain $[x, y]$ do
    foreach Edge $e_i$ of triangle $t$ do
        if $[x, y]$ lies in a half-plane behind $t$ and $e_i$ then
            break
        end
    end
    $t \leftarrow$ Neighbor of $t$ over the edge $e_i$
end
$t$ is the target triangle
```

Algorithm 3.2: Simplified triangle walking pseudocode.

triangles in the triangulation.

3.4.1 Remembering Stochastic Walk

The above algorithm could cycle in an infinite loop in certain cases [57], so the randomized version of the Algorithm 3.2 can be recommended: edges of the current triangle are tested in a random order. With this modification the algorithm reaches the target triangle after a finite number of steps. The walk of the algorithm in an example triangulation is depicted in Figure 3.4.

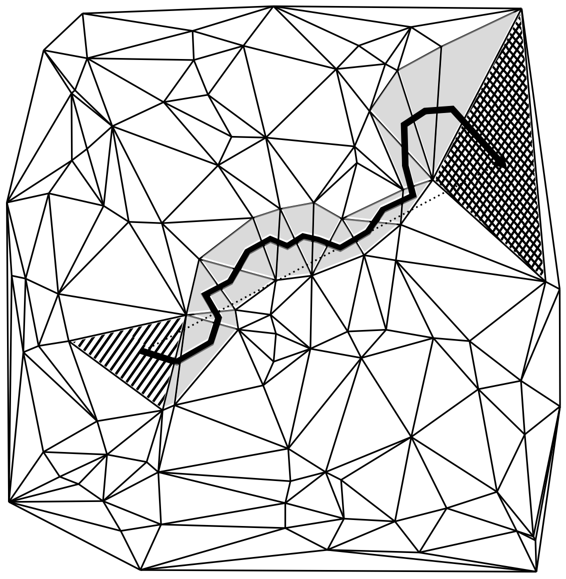


Figure 3.4: An example of the remembering stochastic walk. The search starts at the hatched triangle. Then it goes through all grey triangles in the direction of dotted line. It ends in the cross-hatched triangle where the target point/triangle lies.

3.4.2 Orthogonal Walk

There are more efficient triangle walking algorithms (in the terms of required computational operations). One of them is the orthogonal walk algorithm [15]. This algorithm moves through triangles strictly in the direction of one axis (e.g., X) and then in the second direction (Y). This algorithm visits more triangles than the previous one, but the trick is in the step complexity: expensive sign tests are not needed. Orthogonal walk path in an example triangulation is depicted in Figure 3.5. This algorithm may not stop in the target triangle. Therefore, for the final triangle look-up the previously mentioned remembering stochastic walk is used. The final number of steps is very small (~ 4).

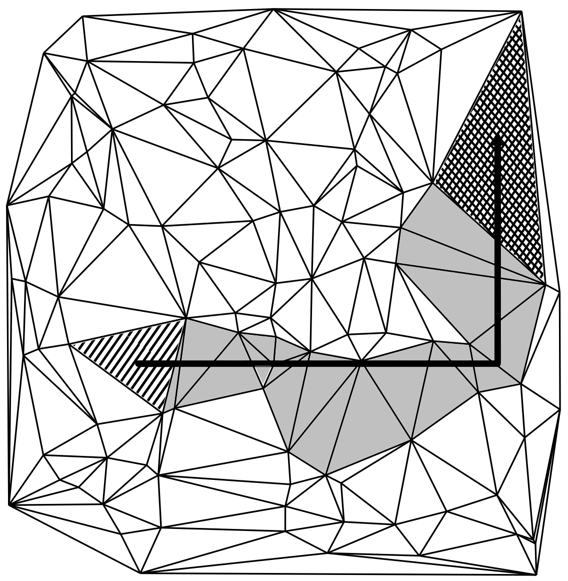


Figure 3.5: An example of the orthogonal walk. The search starts at the hatched triangle. Then it goes through all grey triangles. It ends in the cross-hatched triangle where the target point/triangle lies.

3.4.3 Advanced Search Structures

The optimal triangle look-up in a planar subdivision has computational complexity $\mathcal{O}(logN)$, where N is number of polynomal plane subdivision segments (e.g., triangles) as it is mentioned by D. G. Kirkpatrick in [30]. The memory consumption is $\mathcal{O}(N)$ or higher. This complexity is usually achieved by building an auxiliary hierarchical search structure. An another important fact that has to be taken into account is a required preprocessing time – the time required to construct a search structure from a standard representation. For example the Lipton and Tarjan's algorithm [34] requires $\mathcal{O}(logN)$ search time, $\mathcal{O}(N)$ memory, and $\mathcal{O}(NlogN)$ preprocessing time in triangular subdivisions. Problem of these optimal algorithms for changing meshes is in their required preprocessing time. The search structures usually cannot handle adding or removing vertices on the fly or the implementation of these operations is very difficult.

3.5 Triangulation Destruction

The above described operations increase the mesh complexity (by adding vertices and edges). As some of the mesh vertices could become redundant in time, an inverse operation may be needed: to simplify the mesh by removal some edges and vertices.

There are several algorithms for mesh decimation. We will describe two main approaches in the following sub-chapters.

3.5.1 Edge Collapsing

The first approach to the mesh decimation is based on edge collapsing. Basically, some edge from the triangulation is selected and shrunk to the zero length. The two vertices which defined the edge are identical now and two triangles which are incident with the edge are destroyed. Such approach is described for example by Garland at al. [24]. An example of the mesh collapsing is depicted in Figure 3.6. The modifications of this method usually deal with the displacement of the remaining point according to some application criteria. It is not guaranteed that the resulting triangulation will comply with the Delaunay criterion, nevertheless, we can swap new edges according to the Delaunay criterion.

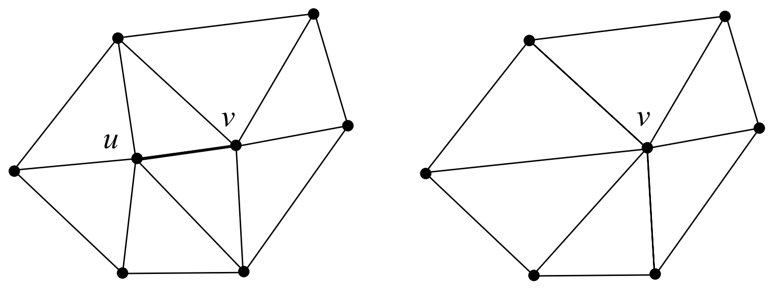


Figure 3.6: An example how an edge-collapsing algorithm works. Left: The edge uv will be collapsed. Right: The same mesh after collapsing uv. Only one inner vertex remains.

3.5.2 Point removal

Another approach how to simplify the mesh is a direct removal of some vertex from the mesh. The hole which appears is retriangulated. The algorithm for deletion on Delaunay triangulation has been published by O. Devillers [14]. It uses the term "ear" – a 3 following vertices on a polygon boundary, forming a triangle. If the vertex is removed from the Delaunay triangulation, the star-shaped polygon appears. By cutting the ears of this polygon in a *correct order* we fix this hole and resulting triangulation is again the DT (see Figure 3.7). The ears order is given by their priority computed as the rate of incircle test result (of the ear and deleted vertex) and the orientation test result (of the ear vertices). See the *power* function in Equation 3.2. If vertices do not form the ear, the priority is $-\infty$. For a detailed information see [14]. In each algorithm step we get the ear with the minimal priority. Deletion of a vertex can be computed in $\mathcal{O}(k \cdot logk)$, where k is the degree of deleted vertex.

$$power(p, circle(q_0, q_1, q_2)) = \frac{\begin{vmatrix} x_{q0} & x_{q1} & x_{q2} & x_p \\ y_{q0} & y_{q1} & y_{q2} & y_p \\ x_{q0}^2 + y_{q0}^2 & x_{q1}^2 + y_{q1}^2 & x_{q2}^2 + y_{q2}^2 & x_p^2 + y_p^2 \\ 1 & 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} x_{q0} & x_{q1} & x_{q2} \\ y_{q0} & y_{q1} & y_{q2} \\ 1 & 1 & 1 \end{vmatrix}}, \quad (3.2)$$

where p is the deleted vertex, q_0, q_1, q_2 are the vertices which form the ear.

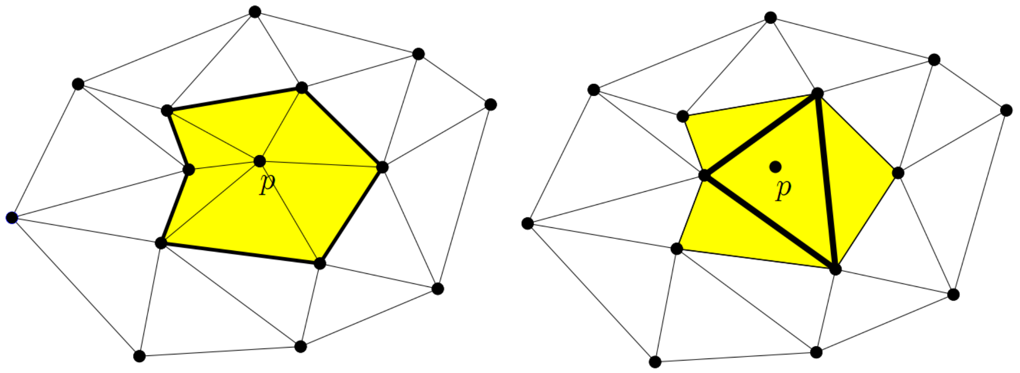


Figure 3.7: An example of vertex removal [14].

Above described essential geometric operations are necessary for geometric modeling in our approach and they will be further used in Section 7. Now we will focus on the modeling of sand in computer graphics and the related papers.

4 Sand Modeling in Computer Graphics

Existing approaches in sand terrain modeling can be subdivided into several groups, e.g., according to their accuracy, to physically accurate and inaccurate methods. Physically accurate methods (nowadays) exclude interactivity, but they provide perfectly looking scenes which is beneficial, e.g., in movies. An example of such an approach has been published by Summer at al. [60]. Inaccurate, interactive methods have been published by Onoue at al. [42], and Beneš at al. [6]. Another physically inaccurate method not running in real-time has been published by Tychonievich at al. [63]. We will describe these methods later in detail.

Another subdivision criterion is the data structure used for terrain representation. Most published work is based on a regular grid (Summer at al. [60], Onoue at al. [42], Beneš at al. [6]). These methods work with a surface model (a heightmap). Surface model is a terrain representation which does not take into account underground. It does not allow to model overhangs and subterranean structures. More formally, the terrain model has to be expressible by a bivariate function. The elevation of the terrain is regularly sampled in grid-based methods and thus, these methods fall into the Eulerian approach.

To the best of our knowledge, the only solution which works with adaptive data structure (the 3D Delaunay triangulation) has been published by Tychonievich at al. [63], such a sampling corresponds with the Lagrandian approach.

Scientific description of granular materials could be found in the paper by Jeager at al. [28]. Physically exact methods usually use particle systems represented as point clouds or springs.

4.1 Modeling of Sand

In this section we will introduce the related work about modeling of sand. For a completeness we will at first shortly discuss all sand-modeling papers in chronological order.

In 1989 Miller and Pearce [36] published a simple particle system with short interaction forces between the particles for animating viscous fluids. This system can simulate a powder-like material. Luciani et al. [35] presented a similar particle system for granular materials using springs. The most important non-particle approaches have been published by Li and Moshell [31] whose developed dynamic heightfield soil simulation is based on Mohr-Coulomb theory. Sumner et al. [60] use a similar heightfield with tool's caused displacement of the terrain and erosion. Onoue and Nishita [42] use height span maps (multi-value heightfields) and particle system to model some 3D effects and they introduce real-time user-driven editing tools. Zhu and Birdson [68] presented application of the Navier-Stokes equations to simulation of sand motion as a continuum. In this paper the sand is considered as a fluid with slightly modified fluid equation solver. Benes at al. [6] add a haptic feedback to control the user tools and they published simplified sand erosion algorithm which works very well for heightfield sand. Tychonievich and Jones [63] use a 3D tetrahedra mesh from point cloud to get surface mesh. On this mesh the spheroidal erosion is applied. Now we will discuss sand-related papers in a detail ([60, 42, 6, 63, 68]).

4.1.1 Animating Sand, Mud, and Snow

Summer at al. [60] describe the principle of surface terrain model that is capable of deformation and allows simulation of different materials, such as sand, mud or snow. This solution uses a heightmap to represent a terrain elevation data (see Figure 2.1).

The terrain deformations are simulated by the computing of an intersection of a solid object (which is used to deform the terrain) and the heightmap. We call these rigid objects *virtual tools* or *user tools*. The height of the cells in a heightmap is continually reduced while the virtual tool does not intersect with the heightmap any more.

Material affected by the tool is either pushed down (according to the compression coefficient) or it is pushed to the near neighbor cells (which do not collide with the tool). This is based on the material properties and shape of the tool. Each deformation results into erosion which reduces the elevation between grid neighbor cells and it smooths the terrain.

In Figure 4.1 a virtual tool collision with the terrain is depicted. The numbers inside the cells determine through how many neighboring cells the material will be moved. In this article sand, mud and snow is modeled. Properties of the modeled material can be defined using six parameters (e.g., the compressibility). This work deals with the physical accurate modeling, but it does not run in real time. Some of time demanding calculations are computed in parallel threads.

The biggest disadvantage of this approach, as mentioned in [60], is the memory consumption of the application. The resolution of the grid must correspond to the smallest details which we want to simulate. Contrarily, the benefit of the grid is an easy access to individual cells of the heightmap. Grid-based terrain could be also textured quite easily.

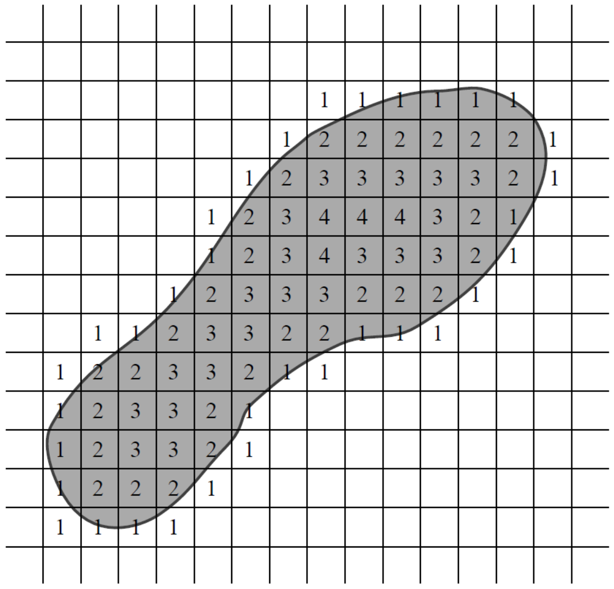


Figure 4.1: An example of virtual tool collision with the grid [60].

4.1.2 Virtual Sandbox

Onoue and Nishita [42] introduce a real-time terrain editing of the terrain covered by powdery material (such as sand). The output of the program [42] is in Figure 4.2. This method also allows to use both virtual tools and erosion. If the virtual tool is in contact with the terrain, the height of surrounding vertices is increased. This simulates a material extrusion. For collision detection bounding boxes and HS maps are used. HS maps are special matrices which contain the height distribution of a given object as it has been mentioned in Section 2. The authors of this paper primarily focus on achieving a reliable visualization of sand. They mention that classic texturing does not look realistic for the sand visualization, so they use a special "sliding textures" technique (which will be described later). As we know where the material will be relocated, we can move the textures in that way. The terrain model is described as a regular grid as in previous case with all its advantages and disadvantages.

In comparison with other methods this model can represent a material on objects, by using HS maps. This is not possible on a simple heightfield because it can hold only one value for the given *xy* position. To enable representation of material on objects, there is a particle system where each particle represents a certain amount of volume of the sand (see Figure 4.3). This volume is excluded from the heightmap. When the particle reaches the terrain surface, the particle is destroyed and vertex elevation in that place is increased. This work is one of the applications which work in real-time.

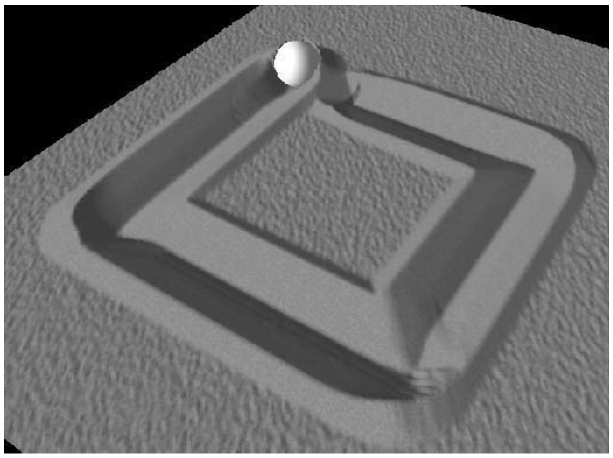


Figure 4.2: Terrain visualization [42].

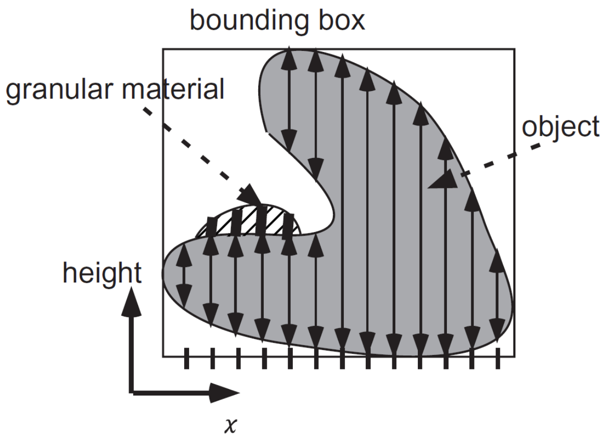


Figure 4.3: HS map example [42]. Each arrow represents a material mass in one span. Spans are stacked for each cell in the matrix. *Granular material* is represented as a particle system and it is independent of HS map.

Texture Sliding

The granular material is visualized using textures in this method. A 256×256 texture image is repeatedly mapped onto the polygonal mesh. This texture is split into 32×32 parts and each part is mapped onto one polygon of the mesh. If the erosion of the granular material occurs, visualization does not look realistic. Therefore, a flowing texture sliding technique is proposed that uses the fact that sand does not need to preserve continuity of the texture pattern at the edges of mesh polygons.

Each vertex of the mesh (x, y, z) has a texture coordinate $T(u, v) = (w_x x, w_y y)$, where w_x and w_y determine the size of the texture. The authors use $w_x = w_y = 0.125\ \ (= 32/256)$. Each polygon has an offset T_{offset} of the texture coordinate.

For the column where material relocation has been computed, T_{offset} is updated by adding ΔT calculated by Equation 4.1.

$$\Delta\vec{T} = -\gamma \cdot Q \cdot \vec{E}, \tag{4.1}$$

where $\vec{E}(e_x, e_y)$ is the material relocation direction (one of eight possible directions in the grid), Q is the quantity of relocated material, and γ is a constant which determines the apparent speed of granular material ($\gamma = 0.05$). After render of each polygon, a texture coordinate is shifted by T_{offset} for each vertex.

4.1.3 Granular Material Interactive Manipulation: Touching Sand with Haptic Feedback

Benes at al. [6] presents a method for the haptic visualization of a sand surface. Interactive haptic editing adds more fidelity to the simulation and it allows the user to really feel the resistance of sand. The force feedback consists of two parts – the penetration force and the thrust force. Sphere is used as a virtual tool, because it is invariant to transformations. Sliding textures are used for a graphics visualization (see Figure 4.4).

The material relocation which is caused by an erosion depends on the humidity of the sand and the *talus angle*.

For details about talus angle see work published by Musgrave at al. [39] or Section 5.1.1.

The authors experimentally set this angle to 30^o. If some part of the terrain reaches this angle (in ascend), the material relocation is stopped until the next terrain deformation caused by a virtual tool. The method belongs to interactive applications. Representation of terrain elevation by a regular grid brings the same advantages and disadvantages as in previous cases.

4.1.4 Delaunay deformable mesh for the weathering and erosion of 3D terrain

Tychonievich and Jones [63] published a mesh-based 3D method using a 3D Delaunay triangulation (DT). Basically this method takes a point cloud and in every frame it computes a 3D DT using CGAL library [1]. Authors use the Delaunay deformable models (DDM) for surface mesh creation and resolving of material differences in the mesh. Details about DDMs can be found in the work published by Pons and Boissonnat [46] and in Section 2.2.2 about data structures. Each material has its own surface mesh. On the resulting surface meshes the spheroidal weathering and hydraulic erosion is computed. Local spheroidal weathering is

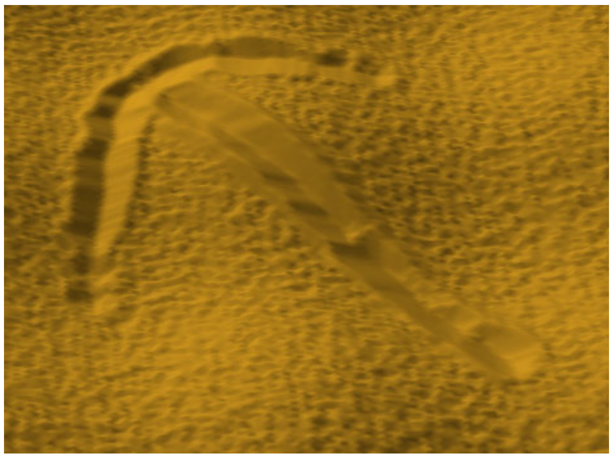


Figure 4.4: A sand terrain example [6].

implemented as a removal of certain small amount of material in the shape of a sphere. Hydraulic erosion uses a dual fluid grid data structure.

By recomputation of all the meshes in every frame the authors avoid topology problems during an object deformations. However, they loose an adjacency of the vertices and materials between frames, because some vertices may move. So it is difficult to estimate boundaries of the materials. This algorithm can neither simulate global weathering effects nor changes to surface appearance. It would be useful to use some kinetic data structure for vertex updates.

This approach is numerically robust, it automatically handles a topology changes, geometric properties such as a normal curvature can be easily computed, and it can, e.g., easily handle viscosity simulations.

It cannot handle higher resolutions. During a strong mass loss, it smears high curvature areas and it cannot resolve very thin parts. The tracking of interface properties (such as color or texture coordinates) is problematic.

4.1.5 Animating Sand as a Fluid

Zhu and Birdson [68] presented application of the simplified Navier-Stokes equations to simulation of sand motion. The sand is considered as continuous fluid. Authors use a cloud of Lagrandian particles for simulation of the sand mass and the grid for environment representation. This allows an easy interaction modeling between individual particles.

Graphics visualization of the result is rather difficult, some surface-tracking method has to be used. This paper contains a superior overview of the previous work published in soil and sand modeling techniques.

4.2 Other Related Papers

This section presents other interesting related papers which usually deal with the terrain modeling in general, such as [10, 62]. Bruneton and Neyret [10] deal with editing of large vector-based terrains. Treib at al. [62] present grid-based, GPU friendly approach for large out-of-core terrain data rendering and editing.

Papers related to a clay modeling are [20, 21]. Ferley at al. [20] developed a sculpture metaphor which uses. Its surface is an iso-surface of a scalar field sampled on an adaptive hierarchical grid. Frisken at al. [21] present concept of adaptively sampled distance fields (ADFs). The ADF is a scalar field that specifies the minimum distance to a shape and this distance is adaptively sampled. ADF can contain both positive or negative values. Authors use ADF as a general representation of shape in CG. ADFs due to their nature could be beneficial to editing operation.

Festenberg and Gumhold [64] present snow modeling framework based on HS maps. The paper also contains an algorithm for HS map creation from 3D model.

Pajarola and Gobbetti [44] published the survey about semi-regular terrain data structures. It contains examples of models based on tiled blocks and nested regular grids, surveys of quadtree and triangle bin-trees triangulations, and cluster-based methods. It contains discussion about system-level aspects of interactive terrain visualization including dynamic scene management, out-of-core data organization and compression, numerical accuracy, and error metrics. There are described structures such as bDAM, P-BDAM, HyperBlock-QuadTIN, and compression.

Next we mention some references to methods which enable efficient adaptive resolution of triangle meshes based on ROAM [25] where the terrain is decomposed to a set of spheres. The authors also deal with compression of the model. Last approach published by Weiss and De Floriani [67] considers sparse terrain pyramids (STP) as a compact multiresolution representation for terrain datasets (a kind of quad-tree structure). Samples of this representation are a subset of samples in regular grid. STP deal with large at areas, where the elevation is uniform.

4.2.1 Real-time rendering and editing of vector-based terrains

Bruneton and Neyret [10] present a GPU friendly algorithm for rendering and editing of large vector-based terrains. This method uses a hybrid terrain representation. The input of method is a regular grid with heightfield which describes the terrain shape together with some vector data which describes features laying onto the terrain (such as roads, trees, and rivers). To achieve interactive framerates, the authors build a view-dependent quad-tree which is used for rendering.

In this approach the terrain and features (objects) are separated. There are basically three approaches how to mix them together:

Overlay-geometry – geometry of features is put over the terrain mesh. There is a problem to keep the objects above the terrain. It can be done with a type of shadow volume technique [53].

Geometry-based – objects geometry is inserted into the terrain mesh itself. There is a problem with the level of detail (LOD).

Texture-based – features are mapped onto the terrain as textures. We need some LOD management to keep the features sharp on close-ups.

The approach [10] is inspired by texture-based mapping of features. Each leaf of above mentioned quad-tree stores an appearance texture (color, material, object meshes). Vector data is stored on the CPU, texture and mesh data is stored on GPU. Rendering is performed as a rendering of series of flat quads for each leaf of the quad-tree. Elevation texture is used for displacement of vertices in the way which connects neighboring quads.

The vector data consists of three types:

nodes – represent point features such as crossings, ends of paths, rivers, etc.; store 2D position and reference to adjacent curves,

curves – represent linear features such as roads, thin rivers; described as a 2D Bezier spline,

areas – represent areal features such as large rivers, fields, and forests, described as a list of curves forming loop.

This method also allows editing of vector data by moving its control points. As the data is stored in a quad-tree, editing requires only local updates. The elevation texture and incident meshes have to be recomputed on scratch.

4.2.2 Interactive Editing of GigaSample Terrain Fields

Treib at al. [62] present grid-based, GPU friendly approach for large out-of-core terrain data rendering and editing. This method is especially designed for applications which require both data decoding and encoding to be faster than disk transfer. For this purpose they use a sparse wavelet representation of terrain data which easily incorporates a compression. The input of method is height-field raster data. The method contains GPU realization of wavelet transforms according to Laan at al. [65].

The kind of hierarchical structure is built on top of the original data. Each 2×2 tile is shrunk to one tile on next level. This quadtree stores on each level the difference between the original tile and the low-pass filtered copy of it. This results in a many close-to-zero coefficients which can be effectively encoded by Huffman encoding. This GPU encoder is a main novelty of the paper.

Authors use a lazy evaluation access. Only a sphere area around the camera is loaded from the disk at the start. Then other tiles are loaded on demand. Editing operations are also delayed – all brush positions and changes are recorded and applied only when some tile is requested. After 250ms changes are propagated to the data. When CPU has not enough memory, changes are written to disk.

4.2.3 A Geometric Algorithm for Snow Distribution in Virtual Scenes

Festenberg and Gumhold [64] have created a simple snow deposition model according to the real world observations. They use the same HS maps as a previously mentioned Onoue and Nishita [42] (square grid of transition lists). Each list is sorted by its transition height along the snow fall direction.

Each snow site transition holds:

- its height,
- its snow height,
- indices of eight neighboring incides,
- its snow patch index,
- its discrete outer and inner edge distances.

After depositing of snow the triangulation of surface is created for rendering and a gentle normal noise is added to achieve more realistic look.

Authors use a grid resolution of a few hundred cells. Computation time is usually a few seconds.

As other grid-based methods there is a problem with grid resolution as mentioned in the paper – each object must be covered by more than single line of snow, this criterion has not to be met for insufficient scene sampling.

As pre-processing step the authors compute a snow cover probability distribution for the whole scene.

This method does not consider topological changes of the scene.

5 Modeling of Erosion on Sand

Scientific description of the behavior of granular materials could be found in the paper of Jeager at al. [28]. In this section we will describe some major types of erosion which affect sand-covered terrains.

5.1 Erosion Modeling

In nature there are many types of terrain erosion, such as an erosion caused by temperature changes, water erosion, wind erosion, etc. A common result of all these processes is that the material is moved from higher places to lower locations to reach a stable state. We will focus on two main types of erosion – erosion caused by temperature changes (thermal weathering), and water erosion. Another important type of erosion is erosion caused by wind.

5.1.1 Thermal Weathering

The moved material settles down by gravity in this case and is stopped by the inner material friction. This process is known as thermal weathering and it is both simple to implement and fast as it is described by Musgrave at al. [39]. Musgarve's work is followed by Benes at al. [3, 5, 6] and others. The terrain is modeled as a regular grid. The thermal weathering process creates talus slopes of uniform angle. This angle in which the material friction force is equal to the gravity is called the *talus angle*. Material transfer is described by the Equation 5.1. The formula moves Δm portion of material from some grid cell to its lower-located neighbors. Other than lower-located neighbors are not taken into account, because the material cannot be relocated uphills.

$$\Delta m_k = \Delta m \, \frac{m_k}{sum}, \tag{5.1}$$

where Δm_k is the amount of material relocated to the k-th neighbor, Δm is a total material amount which can be relocated, m_k is the actual difference of k-th neighbor, sum is the sum of all differences to the lower located vertices.

5.1.2 Water Erosion

Water flow of the Newton's fluid is described by the Navier-Stokes equations. Several papers about the water flow or water erosion have been published. It is usually modeled using a simple 2D/3D grid or by using the Lagrangian particles (it brings more dynamic into the system).

According to Benes and Forsbach [4] the water relocation equation is very similar to sand relocation equation in the grid. The water flows from the higher places to the lower places. Its amount and speed is affected mostly by elevation difference between neighbor cells. This method does not take into account the acceleration of water.

The acceleration of water is used in the method published by Stam [58] or later by Neidhold at al. [40]. This method uses simplified Navier-Stokes equations (see Equation 5.2) which describe the movement of material depending on velocity $\vec{v}$, and acceleration $\vec{a}$ and it can be used for water simulation in discrete grid cells.

$$\begin{aligned} \dot{\vec{v}} &= \vec{a} - K_A \cdot \vec{v} = \tfrac{\vec{F}}{m} - K_A \cdot \vec{v} \\ \dot{\vec{x}} &= \vec{v} \end{aligned} \tag{5.2}$$

$K_A \in < 0; 1 >$ is the coefficient for the friction between the water and the material. $K_A = 0.3$ is a very tough material which slows down the water by about 30% in one simulation step. $\vec{v}$ represents the velocity, $\vec{a}$ acceleration, $\vec{F}$ is an external force, m represents the mass, and x is the position in space.

A time-discrete version of the simplified Navier-Stokes is in Equation 5.3.

$$\begin{aligned} \vec{v}_{t+\Delta t} &= \vec{v}_t + \vec{a}_t \cdot \Delta t - K_A \cdot \vec{v}_t \cdot \Delta t \\ \vec{x}_{t+\Delta t} &= \vec{x}_t + \vec{v}_{t+\Delta_t} \cdot \Delta t \end{aligned} \tag{5.3}$$

The acceleration of the water in xy plane is $|\vec{a}| = \sin(\alpha \cdot g)$. The direction of $\vec{a}$ could be obtained via the gradient I of the heightfield altitude. The situation is depicted in Figure 5.1.

If we incorporate the z axis, the acceleration direction will be according to Equation 5.4.

$$\vec{M} = \left(-\frac{\Delta I}{\Delta x}, -\frac{\Delta I}{\Delta y}, -\frac{\Delta I^2}{\Delta x^2} - -\frac{\Delta I^2}{\Delta y^2} \right)^T \tag{5.4}$$

We can see that the acceleration in xy is only the projection of M. We can substitute $sin(\alpha)$ by $\frac{M_z}{\vec{M}}$.

So the resulting acceleration in the given point will be according to Equation 5.5.

$$\vec{a} = \frac{M_z}{\vec{M}} \cdot g \cdot \frac{\vec{M}}{|\vec{M}|} \tag{5.5}$$

The three dimensional acceleration $\vec{a}$ is the product of the acceleration amount and the normalized acceleration direction vector $\vec{M}$. This result is used in the Equation 5.3 for the fluid simulation.

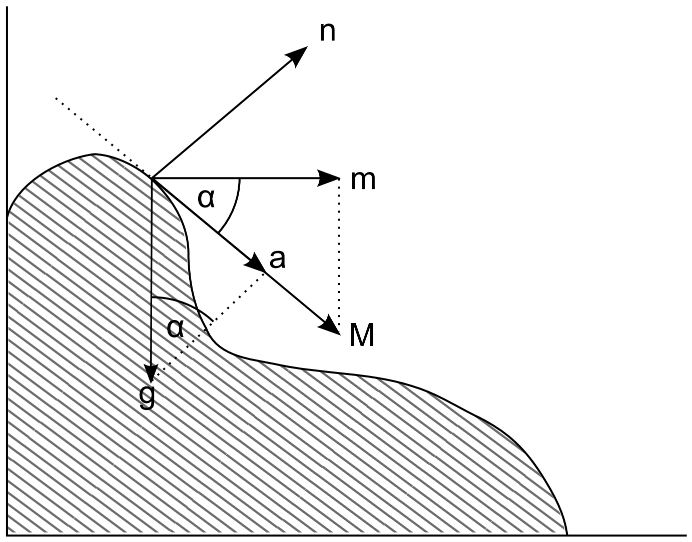


Figure 5.1: Relationship of all the vectors which are used for acceleration computation (a). m is the orthogonal projection of the movement direction $M = [M_x, M_y, M_z]$. n is the normal vector to the surface and g is the standard gravity acceleration.

When the current acceleration is obtained, the new velocity for the next time step has to be be computed at each discrete grid cell. If the destination of the transported material is not exactly in one grid cell, the material is then distributed to the four nearest neighbors with a bilinear interpolation. Multiple values for one grid cell are simply summed up.

To the best of our knowledge there is no published solver for Navier-Stokes equations which works directly with triangulation in 3D space.

6 Haptic Interaction with Terrains

The obvious extension for interactive virtual reality applications (VR) is incorporating of other kinds of VR devices such as a 3D projection, haptic glows, motion detection, or other devices. The terrain manipulation is also a kind of VR application, so it is possible to use VR devices to enhance the user over all impression. In our lab we have the a haptic device Phantom Omni® (see Figure 6.1), so we use this haptic device for the terrain editing and the manipulation with the terrain model.

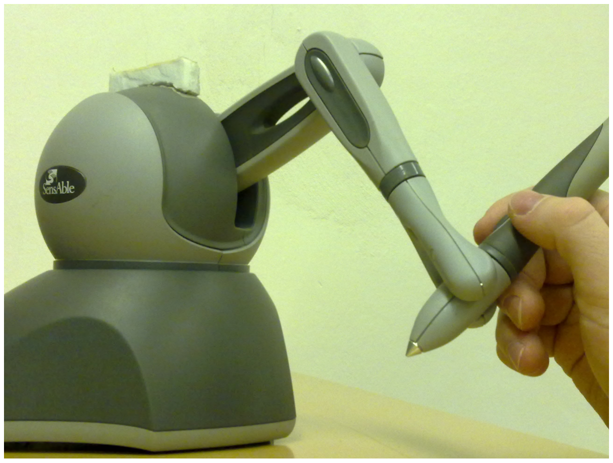


Figure 6.1: The haptic device Phantom Omni® used in our lab.

6.1 Published Methods

In this section we briefly describe the published methods for touching and editing of terrains by haptic devices.

Combining 3-D geovisualization with force feedback driven user interaction

Faeth at al. [19] present a framework for haptic visualization of geospatial data using X3D[1]. Their framework also allows making deformation of the model. They use the ArcGIS program to convert various GIS formats into X3D indexed mesh and grid image. Vertices change their heights after deformation and, additionally, they are rearranged according to Generalized Chainmail deformation algorithm [32]. It results in a cloth-like behavior of the terrain, where the topology

[1]X3D is the ISO standard XML-based file format for representing 3D computer graphics, the successor to the Virtual Reality Modeling Language (VRML)

of the mesh cannot be changed (no vertices are added or removed). "Soft" and "hard" materials are distinguished. For example, moving some mesh vertex up when deforming the soft terrain results in a steep and pointy hill, whereas the identical movement for deforming the hard terrain would result in a rounder hill. Force response can map terrain properties.

Touch-Based Haptics for Interactive Editing on Point Set Surfaces

Guo at al. [26] present a method for interactive haptic manipulation of point sets. User edits are described with implicit functions. The model consists of two representations. The first is a global scalar field which stores a distance from the zero plane. The other representation is a voxel regular grid. A global mass-spring system is created between voxels. This method allows simulation of voxel-based physics processes. Haptic visualization uses the exponential weighted average extrapolation scheme to smooth the resulting force.

Interactive Haptic Rendering of Deformable Surfaces Based on the Medial Axis Transform

Corso at al. [12] presented a haptic visualization method of elastic surface based on B-Splines. They use a medial axis transform where the material is modeled as a set of spheres. Each sphere contains a spring-damper to model elasticity.

Analysis of Haptic Perception of Materials by Multidimensional Scaling and Physical Measurements of Roughness and Compressibility

Tiest and Kappers [61] published an analysis of 124 different material samples of real materials. Each material was cut to a square $10 \times 10cm$. They focus on a measurement of roughness and compressibility of each material. The roughness has been measured by a sampling of a height change during dragging of a pen of the "Universal surface tester" by Innowed GmpH. This pen acts with the constant force, speed, and angle against the surface. The compressibility has been measured by a special device with a rod on which the force has been applied. The displacement of the rod has been measured.

Here the state of the art is concluded. The rest of the report describes our method and our research results in detail.

7 Our Method

This section is devoted to the description of our method for an interactive sand-covered terrain modeling in detail. For an exact explanation we refer reader to our papers [47, 48, 49, 50, 51, 52]. In short, our method is an interactive sand-covered terrain model, based on the Delaunay triangulation, allowing real-time thermal erosion modeling, and containing a set of virtual tools used to deform the terrain by the user. These tools can be optionally controlled with the haptic device which provides a force feedback to the user and adds more fidelity to the simulation process.

Our solution consists of three main parts. The most important part of our system stores the geometric model (a triangulated mesh) that represents the terrain and allows geometric operations, such as searching triangles, adding and deleting vertices, and imposing constraints on certain edges of the mesh. Next part is the physics-based model, which is responsible for the changes of the terrain-geometry and is based on thermal weathering. The last part of the solution is a set of user tools. They are used to change the shape of the terrain according to user requirements and are implemented as constraints of the edges in the triangulation. Both the erosion and the user tools change the terrain shape. While the user tools add features, dig holes, etc., the erosion has an opposite effect as it smoothes the mesh. To implement both operations we need a mechanism that allows adding and erasing edges as well as marking some edges as constrained. The adaptive triangulation is a mechanism that allows for both operations to be implemented efficiently.

We have used a triangulated irregular network (TIN) for representation of the terrain in an interactive physical sand simulation and manipulation, which is the first framework of this type ever used. We have developed a generalized thermal weathering algorithm [51] which could work on both grid and TIN and probably other generic (and optionally irregular) data structures. We have proposed a multiple material modeling on TIN which could be used in the erosion algorithm. We have developed two new TIN haptic visualization techniques: the control point-based method [50] and the triangle normal-based method [52]. We have proposed a method for multiple material haptic visualization. Our method has a potential to be extended to a 3D interactive model which overcomes the heightmap restrictions.

7.1 The Geometry

The core of our method is a terrain representation. Our terrain model is represented by a set of points $\mathcal{P}$ with coordinates x, y, z, where x and y specify the position in the plane and z the height at that location. Triangulation $T(\mathcal{P})$ is defined on a set of points $\mathcal{P}$, consists of a set of triangles with defined edges, definition of the adjacency of the triangles, and adjacency of the points (vertices of the mesh). The set of edges is redundant, but it allows to make certain operations such as erosion computation and edge constraints faster. The terrain can be also described as a function of two variables $f(x, y) = z$. It is important to note that equally as in the heightmap models, the limitation of the triangulated set of vertices is that it does not allow to model overhangs.

A planar triangulation $T(\mathcal{P})$ is in fact also a planar graph $G = \langle \mathcal{P}, \mathcal{E} \rangle$, with $\mathcal{P}$ being the set of vertices and $\mathcal{E}$ the set of edges. For each vertex P_i, where $i = 1 \dots n$ we can define its neighborhood $N_i = \{P_k\}$, $k = 1 \dots n_b$, where N_i contains those vertices $P_k \subset \mathcal{P}$ for which edge $E_{ik} \subset \mathcal{E}$ exists, n_b is the size of N_i.

As the terrain model will be used in the computation of terrain changes and visualization, it is beneficial for the shape of triangles to be nearly equilateral which is automatically provided by the already described Delaunay triangulation.

Standard geometric operations have been already described in Section 3, our algorithms and modifications will be described in the rest of Section 7.1.

Planarity Test for Point Removal

In our model it is beneficial to use a planarity criterion to detect redundant parts of the mesh, because granular materials almost approximate the plane in a steady state. We will discuss several planarity criteria that could be used for the detection.

The first obvious criterion uses the Gaussian curvature (K, see Equation 7.1). The surface for which $K \approx 0$ is near to plane.

$$K = \kappa_1 \cdot \kappa_2, \tag{7.1}$$

where κ_1 and κ_2 are the principal curvatures.

Another planarity criterion could be obtained using the least squares method. As the mesh stores a dual-edge representation (the vertices adjacency), we can test the neighborhood of each point P by computing an approximation plane defined by its neighbors ($N_i = P_k$, where $k = 1 \cdots N_b$, N_b is a number of neighbors) along edges of P. This test will not require any search in the mesh. Let the target plane ρ be $Ax + By + Cz = D$. The plane will certainly pass through the center of the point-mass $C = [X_c, Y_c, Z_c]$. C can be obtained as a mean value of point coordinates $C = \sum_{i=1}^{N_b} P_i / N_b$. We define:

$$a_{00} = \sum_{i=1}^{N_b} (X_i - X_c)^2,$$

$$a_{11} = \sum_{i=1}^{N_b} (Y_i - Y_c)^2,$$

$$a_{22} = \sum_{i=1}^{N_b} (Z_i - Z_c)^2,$$

$$a_{01} = a_{10} = \sum_{i=1}^{N_b} (X_i - X_c) \cdot (Y_i - Y_c),$$

$$a_{02} = a_{20} = \sum_{i=1}^{N_b} (X_i - X_c) \cdot (Z_i - Z_c),$$

$$a_{12} = a_{21} = \sum_{i=1}^{N_b} (Y_i - Y_c) \cdot (Z_i - Z_c).$$

Then the desired plane coefficients A, B, C can be computed solving Equation 7.2.

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} A \\ B \\ C \end{pmatrix} = 0 \tag{7.2}$$

We must add an additional condition (Equation 7.3) to avoid the trivial solution $A = B = C = 0$.

$$A^2 + B^2 + C^2 = 0 \tag{7.3}$$

We do not need an exact planarity test. It is sufficient to detect surfaces which are nearly flat (their maximal distance from an ideal plane is not greater than some user defined constant). Let this distance be d_m. Then we could simplify the above-defined computation. We excluded the Gaussian approach because the user has to enter some "imaginary" plane curvature constant. We only have to find the plane ρ defined by the vertices P, P_1, and P_2 and for the neighbors $(k > 2)$ we perform the following test:

The vertex P is not redundant and it will remain in the mesh if, for a neighbor $P_k, k > 2$, the distance from the plane ρ is greater than a user-defined constant d_m. This constant corresponds to the size of the smallest detail which the terrain model can preserve. We experimentally set d_m on 1‰ of the heightmap width.

7.1.1 Automatic Mesh Simplification

The vertices count can be huge in the whole mesh. So, it is impossible to test all of them at once in each algorithm iteration. We test only the limited area around the changed vertices. The only operation which could change vertices are user actions and the erosion. So, if some vertex has been created, or its height has been changed, we add this vertex to the special queue L_R which accumulates changed vertices. It is incorrect to test each changed vertex immediately after the change, because such a vertex could still be changed by another process in the current algorithm iteration and, moreover, we would perform several extra tests for such a vertex. With the queue L_R with vertices which are suspicious to be redundant one vertex requires only one redundancy test. We perform these tests for all vertices in the queue at the end of algorithm iteration (the mesh is retriangulated and the erosion has been computed). Mesh smoothing and simplification sequence is in Figure 7.1. The initial number of triangles was 1975 and it has dropped to 23% after 507 iterations.

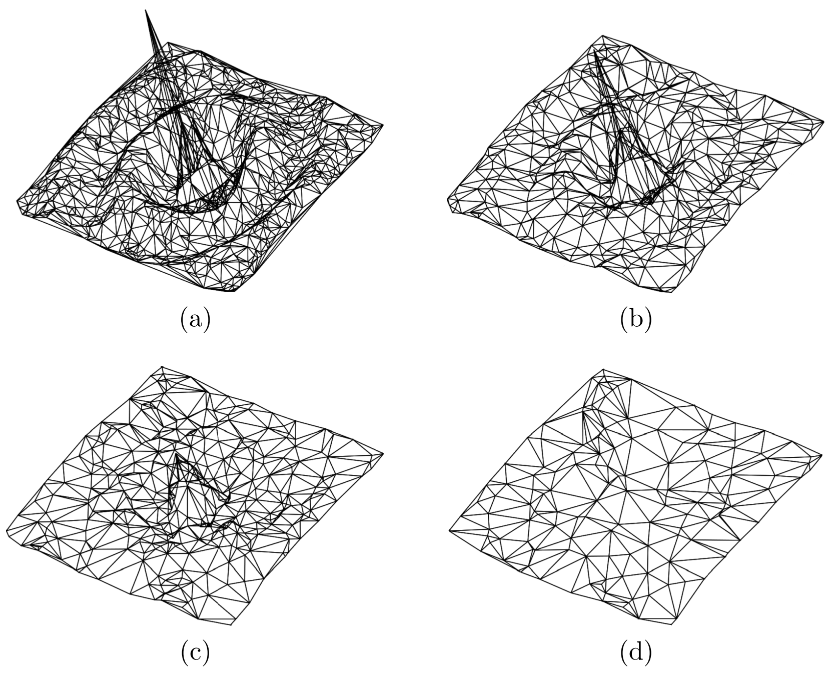


Figure 7.1: Four frames from the mesh simplification sequence, starting with the orginal mesh in (a) and ending with (d). Mesh is continuously being flattened by the erosion and simultaneously decimated.

7.1.2 Point Removal in CDT

In our case the vertex removal is a little bit complicated (more than it has been described in Section 3.5.2), because of constrained edges. So, by cutting ears of the star-shaped hole in an arbitrary order we are unable to restore DT, thus we cut all ears one by one and we store all newly created edges together with incident triangles. When the hole is retriangulated, we check all stored edges, whether they meet the Delaunay criterion. If not, some edges are flipped.

7.2 Erosion for Irregular Meshes

In our method we have focused on the above described thermal weathering. We follow the framework created by Benes at al. [6]. The paper [6] is based on a regular grid which samples the terrain heightfield. It is obvious that distances to the neighbor vertices in the grid are constant and each vertex has exactly 4 (or 8) neighbors, not so in the triangulation. The distances to the neighbors and even neighbors count could vary. For our purposes we need to restate the problem.

For each vertex $P_i \subset \mathcal{P}$ from $CDT(\mathcal{P})$ we find its neighborhood $N_i = P_k$, $k = 1, \cdots, n_b$. Also for each pair of vertices P_i, P_k we compute the slope β. That is the angle formed by a horizontal line which is going through the P_k and the line defined by the edge from P_i to P_k. The material tends to reach a minimal angle, called the *talus angle*, by relocation from the weathering. As it is mentioned in [5], the dry sand has talus angle $\approx 30°$.

Our erosion algorithm uses a queue of vertices to be eroded denoted L_E. Neighborhood of each vertex in the queue is checked for the material transfer. When the initial mesh is loaded, all vertices have to be enqueued in L_E, because there is no guarantee that β is minimal for all vertices. In runtime, the erosion computation takes place only on vertices which has changed their height. These vertices are stored in the queue. The vertex can be removed from L_E, if it did not change its height in a last few erosion iterations.

The erosion computation takes place only locally around the selected vertices. When some vertex P is dequeued from L_E, we check its neighborhood and transfer material to neighbors P_k which lie below the P (if the talus angle criterion is met of course). Material transfer is modeled as a height change of the central vertex P (decrease) and P_k (increase). When the whole queue L_E is processed, the terrain has been eroded in a global scale.

The amount of moved material is given by Equation 7.4:

$$\Delta m = \frac{1}{2} max(m_k), \quad k = 1 \ldots n_b \tag{7.4}$$

where n_b is the neighbor vertices count, m_k is the height difference to the k-th neighbor of P. The constant $\frac{1}{2}$ is used to damp the erosion and prevent the system from oscillation.

From Δm, the material moved to the k-th neighbor Δm_k can be computed as follows:

$$\Delta m_k = \Delta m \; w_k \frac{m_k}{sum}, \tag{7.5}$$

where Δm is given by Equation 7.4, w_k is a weight of the vertex P_k given by its distance from P (explained below) and sum is the sum of all differences of height of P to the lower neighbors. The w_k incorporates the influence of vertex distances in projection, so all distances in the following explanation are computed in xy plane only.

Material transfers along very long edges do not look realistic. We need to split these edges, so that the material will fall down only into some limited distance. Let this maximal material relocation distance be r_S and the distance from P to P_k be r_k.

Let us suppose (for now) that all edges from P are longer than r_S. Each edge has to be subdivided by inserting additional vertex $B_k, \; k = 1, \cdots, n_b$. All vertices B_k lie (in a projection) on the circle with radius r_S and the center in P (the distance to any B_k from P is constant). See vertices $P_2, \cdots, P_5$ and $B_2, \cdots, B_5$ in Figure 7.2. Then we move material to all vertices B_k using Equation 7.5, keeping in mind that n_b differs for each vertex.

Now we will consider neighbors whose r_k is smaller or equal than r_S (see P_1 in Figure 7.2). We will move a certain amount of the material to these vertices. This amount is scaled by a distance from P. No new subdividing vertices are created in this case (the length of the edge to B_k is short enough).

Considering both cases, the weight w_k is given by Equation 7.6.

$$w_k = \frac{min(r_s, r_k)}{r_s}, \quad k = 1, \ldots, n_b \tag{7.6}$$

The value of w_k is in the interval $(0; 1]$, where weight 1 corresponds to the point on the circle (P, r_S), any closer point will received smaller amount of material according to its weight.

Prevention of Long Edges

The edges which are connected to some B_k (and do not contain P) can be arbitrary long. So we also need to subdivide edges of vertices which are subjects to erosion in the similar way as we computed vertices B_k. The heights of newly

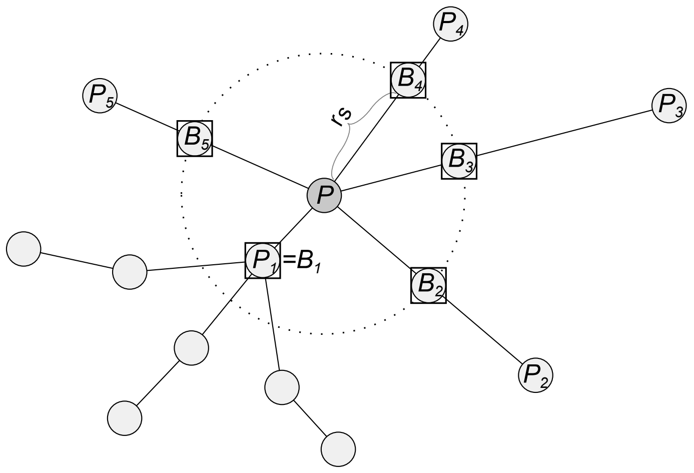


Figure 7.2: Material from the central vertex P will be moved to subdivision vertices B_k. Vertex P has neighbors P_i, $k = 1 \ldots 5$. Vertex P_1 has the distance to P smaller than r_s. So a smaller amount of material will be relocated to it.

created vertices are set according to the terrain height interpolated in the places of the insertion of new vertices.

7.3 User Interaction

A set of tools is used for the interactive user-driven terrain editing. The tool could have any shape allowed by our model (it must be expressible as a bivariate function; it cannot contain vertical edges or overhangs). The tool can leave its footprint in the terrain or it can be dragged on the terrain surface. Tools are modeled using CDT. Basically, each footprint is made by enforcing a set of edges into the triangulation and the terrain height changes as it is described below. Dragging is achieved by sampling tool's footprints in a moving trajectory.

Each tool consists of two main parts (see Figure 7.3). The first one is the inner part of the tool. Second one is the outer part. The purpose of the outer part is to protect the rest of the terrain model from the tool deformations. It divides edges in the place of user action by adapting to the terrain shape and it enforces the tool border into the terrain, whereas the inner part levels area inside its boundaries.

Because the control points are usually mutually near, we can speed up the triangle walking algorithm from Section 3.4, which has to be performed for each point. We store the previously found triangle in cache. This triangle is added to the random sample of triangles for triangle walking algorithm. If it is closer to the target point than other triangles, it is chosen as a starting triangle.

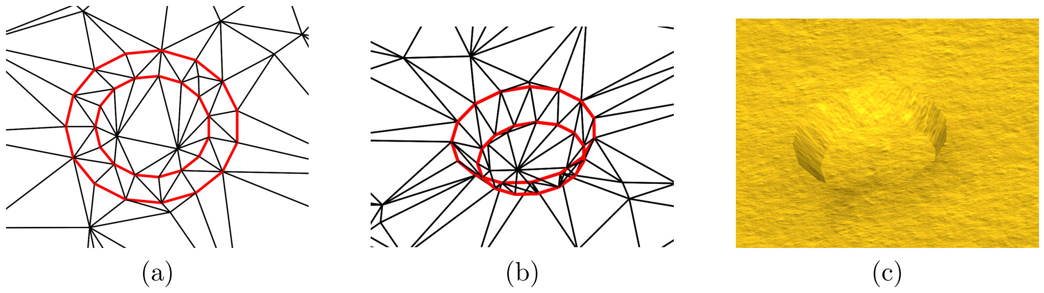


Figure 7.3: User tool footprint which compresses material. (a) Footprint view in a vertical projection. Wireframe model with thick lines indicating constrained edges. The enclosing circle represents the set of outer edges, inner circle the set of inner points. (b) Same as (a) but in isometric view. (c) Textured model. All models are displayed without erosion.

7.3.1 Outer Part of the Tool

The outer part of each tool is defined as a set of constrained edges as it can be seen in Figure 7.3a. If the tool has to be imprinted into the terrain, we compute position of points that form the set of outer edges together with the absolute tool position in the terrain. Then we get incident triangles for these vertices. The first triangle is get by a triangle walking algorithm. Triangles for other control points are got by walking from the first triangle along its neighbors (the last found triangle is used as a starting triangle for a new triangle look-up). As a next step we simply interpolate the height of the terrain in the current position of control vertices. After that we constrain the shape of the outer part of the tool by enforcing one edge after another into the triangulation. The heights of newly inserted vertices are the heights of terrain (in the position of new vertices) we have interpolated before constraining. Now the terrain is geometrically split to the part inside the tool and the rest of the terrain. We want to affect only the part restricted by the outer part of the tool.

At this moment we have tools with the rectangle, circle, lambda shape, and man foot shape of outer part.

7.3.2 Inner Part of the Tool

The inner part can be defined by a set of edges or by a set of control points. When the tool has to be imprinted, we first constrain the outer part (as it is mentioned above). After that we compute the position of the inner control vertices (using the current absolute position of the tool and the tool depth level). And we constrain inner edges into the mesh if there are defined for the inner part. Next, we insert control vertices which are not connected by constrained edges. Inner

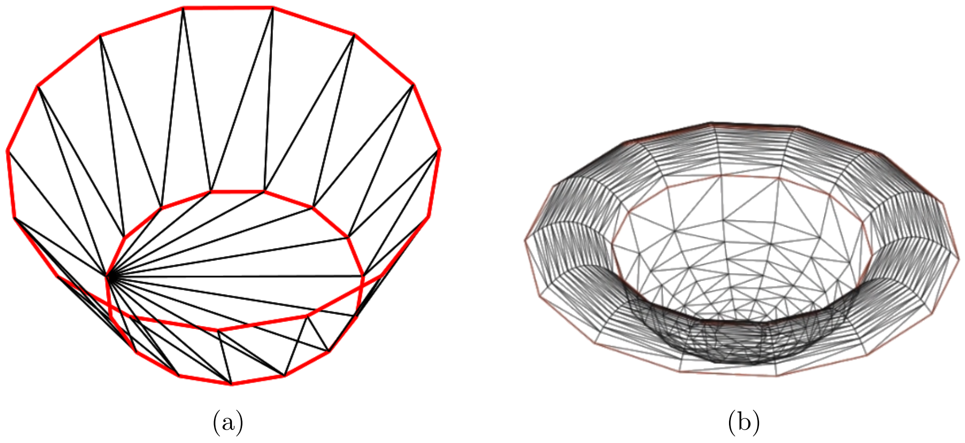


Figure 7.4: Two different inner parts. (a) Flat inner part. (b) Half-sphere ($z = \sqrt{x^2 + y^2 - r^2}$) inner part.

control vertices have the height given by the inner tool shape. Therefore we must set the correct height of existing vertices. To do it, we get some triangle inside the inner part (for most tools we simply compute the center of the tool and use triangle look-up). Then we use breadth-first search to find all inner triangles. Vertices of these triangles are set to the correct height (see Figure 7.4).

We have tools with various inner part shape – it can be flat, or a spheroidal shape. Additionally, we have tools whose inner surface is described by a 3D model or a bitmap heightfield.

The illustrative situation of the tool imprinting is depicted in Figure 7.5.

7.3.3 Material Extrusion

To achieve a more realistic simulation, it is necessary to extrude the material around the tool imprint. The physically accurate method has to take into account all material properties (such as thrust, granularity, humidity, stiffness, etc.) and their distribution around the imprint, moreover, the extrusion depends on the velocity of the tool, the tool's movement force, penetration, terrain shape, etc.

We have developed several approximations of the above processes. The most simple approach appears to be a constant elevation around the outer part of the tool (see Figure 7.6a). But this could cause problems during tool dragging, because the extrusion elevations are gradually increased. The extruded shape has to be separated from the rest of the terrain by another "outer part" called the extrusion part.

A better way is to use a parabolic material extrusion (see Figure 7.6b). We have a tool with the shape of half-sphere and it creates a parabolic extrusion which

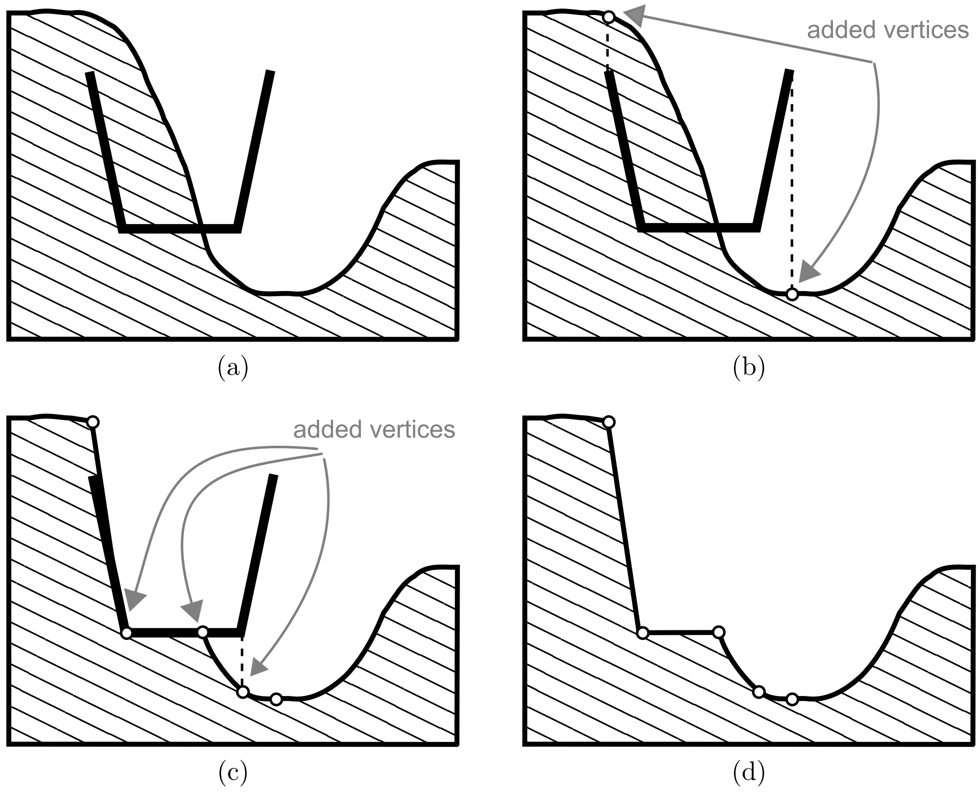


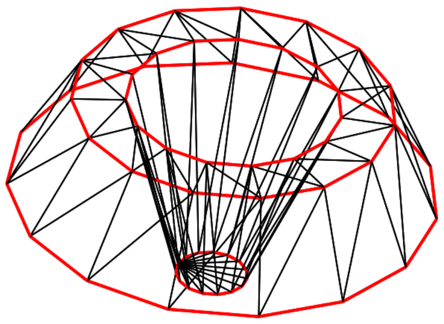
Figure 7.5: A simple tool imprinting sequence. Starting with (a) original terrain, thick edges represent the tool shape, (b) the outer part has been imprinted, (c) the inner part has been imprinted, (d) the terrain after imprinting of tool shape.

is always fitted to the extrusion part (or the outer most part) of the tool. To avoid problem with the gradual height increase, we define weights for each control vertex. Each weight depends on a height difference between each outer control vertex and the tool depth level. If the heights are equal, the weight is zero. With this approach the material is pushed away only in the direction where the terrain has been deformed. While the user drags the tool, it extrudes material only on the sides of the movement course and against the tool movement direction.

7.3.4 More advanced tools

To achieve more interesting effects, we need to create tools with complicated shapes. As an example we present a tool with the shape of lambda letter, see figure 7.7.

Another example is a tool which has been created using a 3D model. Basically, we choose one side of the model which should be imprinted into the terrain and compute the projection of the model points into plane in the imprinting direction.


(a)

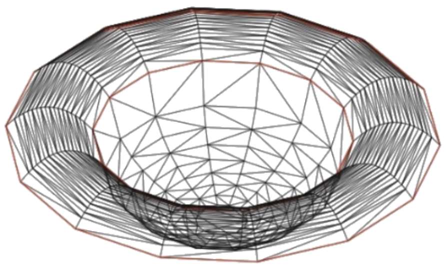
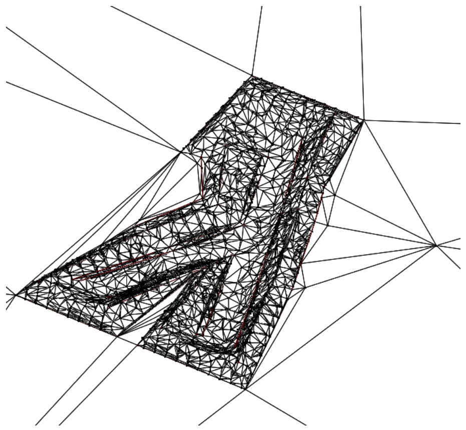

(b)

Figure 7.6: Two types of material extrusion. (a) Simple elevation extrusion. (b) Parabolic extrusion.


(a)

(b)

Figure 7.7: Non-convex tool with the shape of lambda letter. (a) The wireframe model. (b) A textured version.

We choose only those points, which are visible from the plane. It guarantees that we obtain only 1 value in one (x, y) point. Finally we return the lost dimension to the chosen points. Points are added into the separate triangulated auxiliary mesh to create triangle heightfield. This allows us to easily get a height in any place inside the imprint. It is important when the inner part of such a tool has to be imprinted into the real terrain because of height decimation in the inner part. The final step is tracing the tool contour. Traced vertices form the outer part of the tool rest of vertices form the inner part.

When we have inner and outer part, we can use the above described mechanism for the user tools handling. The foot tool example is in Figure 7.8.

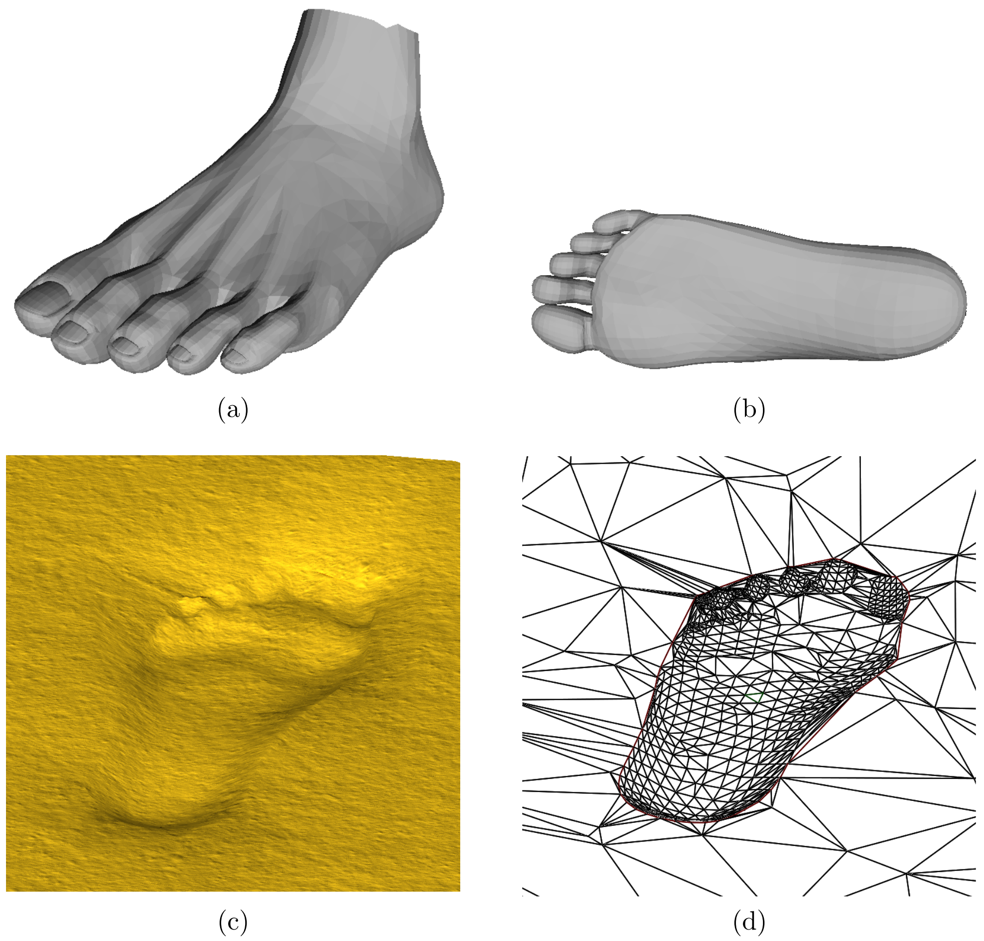

(a) (b) (c) (d)

Figure 7.8: A foot tool. Original model of the foot is in (a) and (b) (generated by the program Poser [56]). (c) Textured version of the tool footprint. (d) Wireframe footprint.

7.4 Haptic Feedback

To add a new dimension to our modeling framework, we have incorporated haptic devices to control user tools. Haptic devices are special input/output devices which provide the force feedback (as an output) and 3D position of haptic cursor (as an input). These devices have a potential to provide additional information to the user (sense of touch) in addition to visual output only. Unlike the mouse or keyboard, the haptic devices provide native 3D manipulation, which is more intuitive. Haptic devices were only used in a high-tech applications in the past (such as space technology and medicine) and hence they were very expensive. But, nowadays, they are becoming affordable to general public. For example new Falcon Novint devices are priced at $250 in March 2012.

The mesh is constantly changing due to erosion and user edits. It is difficult to maintain a correct force feedback during the geometric changes in the mesh. Therefore, we define a stable state of the mesh as the state when all mesh updates have been performed and the mesh has been retriangulated (it is the end of the graphics loop). The graphics (and geometric) visualization thread has the refresh rate about 60Hz which is sufficient for eyes, but that is below the required refresh rate for haptic visualization that is about 1,000Hz. It means that we have to set the resulting force F_{out} to the device 1,000 times per second (as a 3D vector). The device acts in the direction of force vector with the given force amount. The force is set in newtons. Because of such differences in the refresh rates we use parallel rendering threads – one for graphics and one for haptic rendering. We have developed two methods for haptic visualization of TIN surface – *the control point-based method* and *the triangle normal-based method*. The first method is described in subsequent sections, for the description of second method we refer reader to our paper [52].

7.4.1 Force computation – control point-based method

We can update the mesh dependent data for haptics only in a stable state. The data, whose update is fast enough, is the tool position $[x, y, z]$. The tool movement direction and its velocity is acquired as the difference between two consecutive states of the tool.

When we compare refresh rates of both rendering threads we get $1,000Hz/60Hz \approx 17$. So the haptic thread makes about 17 iterations for one graphics render. Such a slow communication with the haptic thread is not optimal, the haptic feedback feels runny for slow framerates. To avoid slow updates we interpolate the force between two consecutive states of the haptic thread. This makes the force feedback more smooth.

Force feedback components
The force feedback F consists of two forces – the penetration force F_P and the friction force F_r.

The F_p depends on a tool depth level. A deeper penetration results in a large force which acts strictly in the vertical direction for the sand and our set of tools. At it is mentioned in [6], sand settles down quickly, so we can expect nearly a horizontal surface. It is impossible to feel difference between the physically accurate penetration force and our simplified version. The real penetration force is handled by triangle normal-based method method.

A basic physical definition of the friction force is in Equation 7.7. It depends on the tool velocity – a higher velocity means that the large force acts again tool movement. F_r also depends on the penetration depth but only in the direction of tool movement.

$$F_r = \mu \cdot F_N, \tag{7.7}$$

where μ is the coefficient of friction, which is an empirical property different for each material, and F_n is the normal force perpendicular to the terrain surface acting against the tool movement. We use just an approximation of this physical law.

As the haptic device we use (Phantom Omni) has only a point cursor and our tools could have a generic shape, we need to map all the forces along the circumfence of the tool into a single point. As it is described in Section 7.3, our virtual tool consists of a set of control points. We can compute the correct force feedback in each individual control point. The force mapping can be roughly computed as a mean value of these forces (see Equation 7.8). Such approach is not physically accurate for non-circular tools, but it gives good feedback for the tools we have tested.

$$F = \frac{\sum_{i=1}^{N} F_{ri} + F_{pi}}{N} + F_G, \tag{7.8}$$

where F_{ri} is the friction force for i-th control point, F_{pi} is the penetration force for the given control point, N is the number of control points, and F_G is the force caused by the "gel effect" (described below).

The force F_G is an additional force which acts in the horizontal direction. It is a haptic effect known as gel effect. It is described e.g. by Lin at al. [33]. The force depends on the velocity of the tool and acts as a tangent viscosity. Higher velocity causes the large force acting against the tool movement direction. This effect very well describes the behavior of real sand and simulates the resistance of the sand against the fast movement of the tool.

Force computation

Let z_{tool} be the current depth level of the tool, c_i the position of i-th tool's control point, P_{act} a current position of the tool, P_{old} position of the tool in previous haptic render, and T the vertical force tolerance. If the tool is close to the terrain surface (the distance of the tool from the terrain surface is smaller than T) the force starts to act.

The height difference for the i-th control point is computed according to Equation 7.9.

$$\Delta h_i = z_{tool} - z_{ci} \tag{7.9}$$

The force starts to act only when the tool is slightly above the terrain surface (the height difference $\Delta h_i > -T$). This tolerance avoids some numerical errors and provides better force feedback. The direction of the tool movement from the i-th control point dir_i is computed according to Equation 7.10.

$$dir_i = \frac{c_i - P_{act}}{|c_i - P_{act}|} \tag{7.10}$$

The partial friction force is computed with Equation 7.11.

$$F_{ri} = -dir_i \cdot \Delta h_i \cdot \frac{F_M}{2}, \tag{7.11}$$

where F_M is the maximal force allowed by haptic device (10N for the Phantom Omni). The partial penetration force is computed with Equation 7.12.

$$F_{pi} = \Delta h_i \cdot \frac{F_M}{2} \tag{7.12}$$

For the computation of F_G we need the velocity of the tool. It is computed with Equation 7.13. We expect a constant time interval between the frames (equal to one).

$$v = P_{old} - P_{act} \tag{7.13}$$

Now the gel effect force can be computed with Equation 7.14. The zero z coordinate restricts F_G to the horizontal direction.

$$F_G = [v_x, v_y, 0] \cdot F_M \tag{7.14}$$

Now we can compute the mean force F from Equation 7.8.

Before we set the final force F_{out} to the device we use the above mentioned force interpolation to smooth any abrupt changes between geometry updates. We have two counters in haptic rendering thread: the total frame count between the last two geometry updates it_p, and the frame count rendered since the last geometry update it. The force F_{out} is computed as Equation 7.15.

$$F_{out} = F_{old} + \frac{F - F_{old}}{it_p} \cdot it, \tag{7.15}$$

where the force F is from Equation 7.8 and F_{out} is the force F computed before the last geometry update.

The force F_{out} is set to the device and acts against the user. When the haptic cursor does not move, the partial forces in control points are mutually balanced in the sum.

Force feedback is smooth and it is almost the same as the real sand. The penetration force is slightly increased with increasing depth of the tool. The user must make a significant effort to push the tool to the sides if its depth is high. If the tool rests, the force components are mutually balanced.

8 Results

We have implemented our solution described in Section 7 in our testing framework using C++ with the help of OpenGL library and GLSL. The application runs on Windows 7 and Linux. All experiments run on a desktop under Windows 7, with 12GB RAM, Intel Core i7 clocked at 3.2GHz, and NVIDIA GeForce GTX 580, with 1.5GB of memory.

8.1 Visual Output

This chapter contains rendered images created using our testing application.

The demonstration of tools changed in runtime is in Figure 8.1. We use two 3D model user tools (a human footprint described in Section 7.3.4), one for the left foot and one for the right foot. We can see that the mesh well adapts to the shape of the footprint.

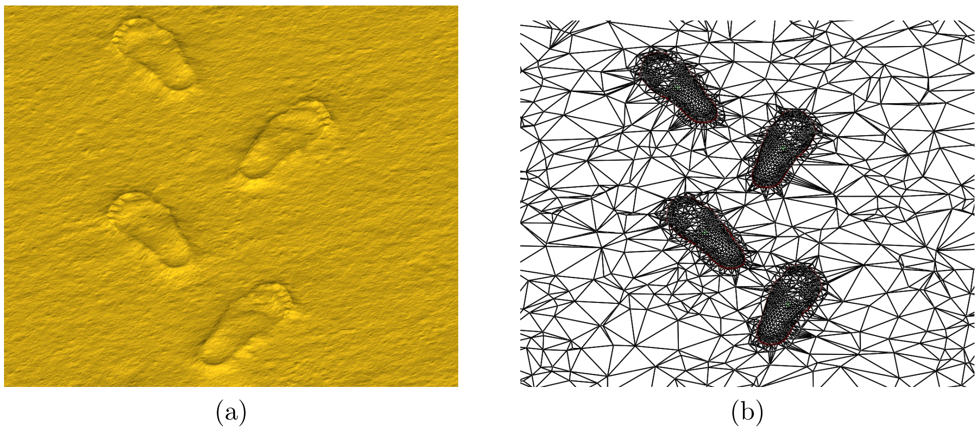


Figure 8.1: Two user tools changed in runtime (footsteps), (a) textured model, (b) wireframe model.

Figure 8.2 contains a half-sphere tool footprints described in Section 7.3.3. This tool extrudes the material and this material is pushed up to a parabolic shape which imitates the behavior of a natural sand.

Figure 8.3 is a model of the Crater Lake (Oregon) edited by a tool with the shape of half-sphere. Crate Lake data has been provided by [23]. This model has about $100k$ vertices. The model is very dense in the locations with complex shapes, whereas the flat areas require only a few triangles.

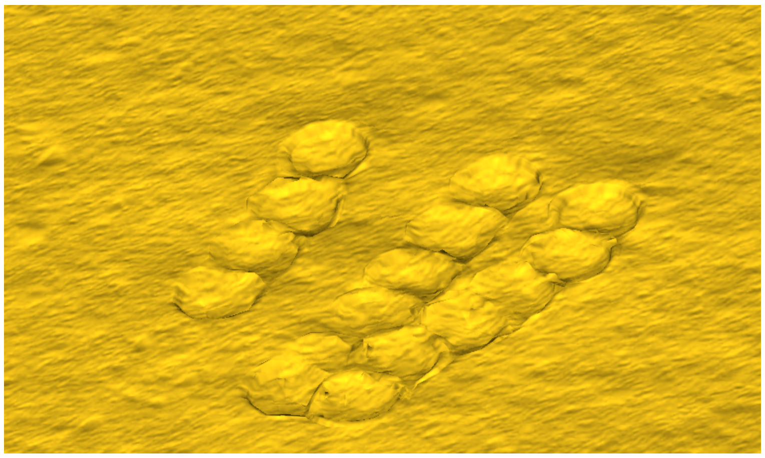


Figure 8.2: Separated eroded footprints of tool of half-sphere shape with parabolic material extrusion.

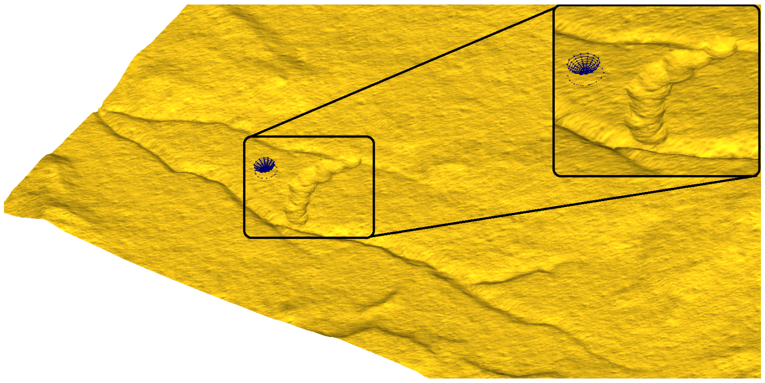


Figure 8.3: A part of Crater Lake model covered by sand. Two canyons in the bottom-left part are now connected by another path crated by user tool dragging. The user tool is shown in blue.

8.2 Measurements

This chapter contains several memory and performance measurements and comparison with the grid-based framework created by B. Benes. We have published the same results in [51]. The graph in Figure 8.4a shows comparison of the framerate of the mesh-based and grid-based methods as a function of mean terrain approximation error. This error has been computed as the absolute volume difference between an exact model (with large difference in detail) and the mesh approximation model. Mesh-based solution is faster when there is a large difference in detail, such as a terrain which is mostly monotonous and in one part, high level of detail is needed. The grid-based approach was not able to achieve as low error as the mesh-based approach with the same framerate.

The framerate of our implementation as a function of mesh size and framerate of grid-based method for the same number of vertices are reported in Figure 8.4b. Both solutions worked here on a model with uniform distribution of vertices and uniform level of detail. Mesh-based solution is slower for the same number of vertices, because terrain features are uniformly distributed along the whole terrain model and the triangle distribution in the mesh is close to the grid.

The memory requirements of the mesh-based method compared to the grid-based approach as the function of the model resolution is shown in Figure 8.4c. Vertices in the triangulation are uniformly distributed. Our solution has higher memory requirements than the grid if the data is uniformly distributed and level of detail is the same at the whole terrain because of the triangulation data structures overhead.

In the next experiment, we show a terrain model with large difference in detail and we compare the number of vertices needed in grid-based and mesh-based approach. Both models describe the same level of detail. The detail is measured using the mean terrain error. Memory consumption comparison can be seen in Figure 8.4d and required vertices count for the given detail as in Figure 8.4e. For models with non-uniformly distributed data the adaptive solution performs better in terms of memory requirements.

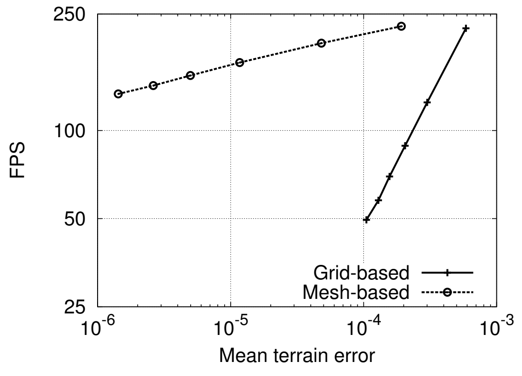
We have also tested our approach in the user experiment to verify fidelity and usability of material simulation and editing [52]. The number of test participants was 24, a suitable number as described by Tiest and Kappers [61]. The characteristics of the age Ag of participants are: $\overline{Ag} = 31.32\ years$, $med(Ag) = 25$, $mod(Ag) = 24$, $\sigma_{Ag} = 14.52$, $min(Ag) = 20\ years$, and $max(Ag) = 72\ years$.

Our experiment consists of four parts (A, B, C, and D) and each part has some number of questions (A and B have eight questions, C two questions and D has one question). In the part A the user is a kind of "blind". He cannot see the material he is touching and he has to choose a correct answer from the list of available materials. Results of this experiment are in Table 8.1 as a percentual correctness (Co). The best result had the glass followed by wool and tiles. It shows that the materials with low friction have better correctness.

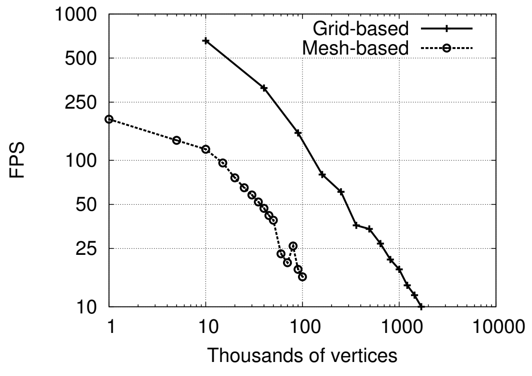
The part B contains both graphics and haptic visualization of materials and the user has to evaluate his satisfaction with the force feedback E from 1 to 5, where $E = 1$ means the best and $E = 5$ the worst sense. There were also eight questions. The results are also in Table 8.1 in the row "*eval.*". Surprisingly, the best marks were collected for sand, wood, and asphalt – materials with higher friction.

In the part C the user has to guess which continent outline, covered by some of available materials, is touched. There were two questions $C1$ and $C2$. The experiment $C1$ has correctness 83.33% (Africa), $C2$ has 54.17% (Australia).

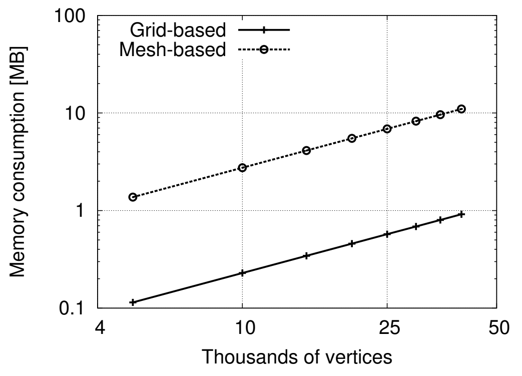
The part D contains a sense evaluation of deformable terrain mesh model. User



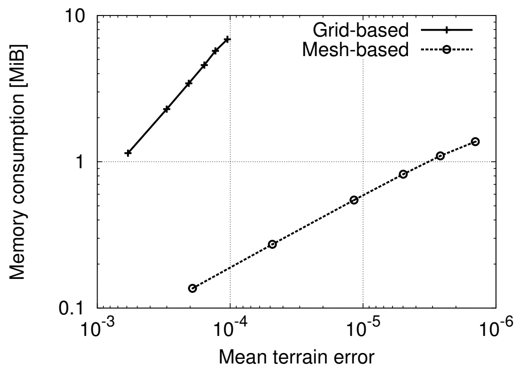
(a) FPS dependency on mean terrain approximation error (logarithmic scale, higher is better). Mesh-based solution is faster when there is a large difference in detail.



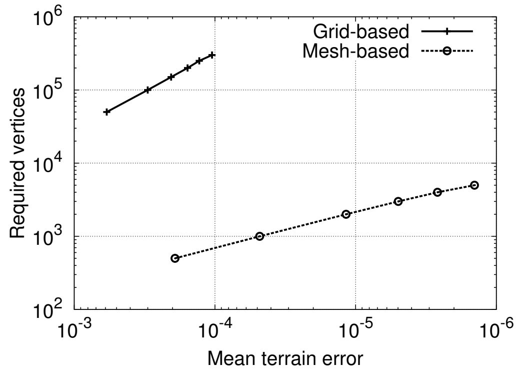
(b) FPS dependency on mesh/grid size (logarithmic scale, higher is better). Grid-based solution is faster when the detail is more or less the same.



(c) Memory consumption comparison of grid-based solution and mesh-based method for uniformly distributed data (logarithmic scale, lower is better).



(d) Memory requirements of both methods for non-uniform terrain (logarithmic scale, lower is better).



(e) Required vertices count for given detail (logarithmic scale, lower is better).

Figure 8.4: Measurement results.

Table 8.1: Relative ratios of recognition of all tested materials. Correct answers are in bold. The row *eval.* contains satisfaction evaluation of the users (1 – means the best, 5 – means the worst). Rows $A1$-$A8$ correspond to part A and contain the percentual correctness of answers Co.

	asphalt	*cloth*	*wood*	*wool*	*glass*	*sand*	*concrete*	*tiles*
eval.	1.83	1.88	1.42	2.92	2.50	1.33	2.38	2.33
A1	**45.83%**	4.17%	8.33%	0.00%	0.00%	0.00%	29.17	0.00%
A2	4.17%	**29.17%**	25.00%	0.00%	0.00%	16.67%	12.50%	0.00%
A3	4.17%	16.67%	**25.00%**	0.00%	0.00%	12.50%	16.67%	12.50%
A4	4.17%	0.00%	0.00%	**66.67%**	16.67%	0.00%	0.00%	0.00%
A5	4.17%	8.33%	0.00%	4.17%	**70.83%**	0.00%	0.00%	0.00%
A6	16.67%	8.33%	4.17%	0.00%	0.00%	**41.67%**	12.50%	4.17%
A7	25.00%	4.17%	16.67%	4.17%	0.00%	8.33%	**20.83%**	8.33
A8	0.00%	8.33%	4.17%	4.17%	8.33%	0.00%	0.00%	**62.50%**

has to edit the model in some way and to evaluate the editing. Available shapes are @, □, and ◯. There was only one question $D1$. The average sense evaluation of the experiment $D1$ was 2.75 ($D1 \in < 1; 5 >$).

We have measured random variables such as the age and the sex of participants and we have found interesting correlations. Let the Ag be the random variable of age, Co the vector of response correctness of all participants, E the mean evaluation vector for all materials, and S the sex of participants. Then the correlation $\rho_{Ag,Co} = -0.33$ means that the participants with higher age performed more wrong answers than younger ones. $\rho_{E,Co} = 0.50$ means that materials which had better evaluation had worse correctness, and finally $\rho_{S,Co} = 0.02$ shows that the correctness of the answers does not depend on the sex of participants.

Subjectively, the most entertaining questions for the participants were those to guess something (parts A, and C). Most of the test participants were students, so they took this quiz as a funny competition. The question which was also favored was $D1$, where participants had to create something.

9 Future Work

Our framework is in a mature state, however, in the near future we want to incorporate simulation of different materials in the terrain. Our erosion algorithm should handle it without major modifications. This could be probably done in such a way that the talus angle will not be constant any more, but it will differ for each vertex. The most challenging part will be modeling of interfaces between materials. For rigid materials the interface is simply some edge in the mesh, but viscous or granular materials do not have fixed boundaries. We would also like to develop a new types of erosion which will work with TIN.

The haptic visualization has to be improved much more. Each material should have a special label which will be stored in the mesh. We are preparing a general purpose haptic library which allows to simulate different materials on our mesh. Next problem could be with graphics visualization of solid materials. On the boundaries of two granular materials we could simply interpolate material properties. But for the boundaries of for example solid rock and a sand we have to choose more sophisticated approach. We propose a weighted interpolation for both graphics and haptic visualization.

Our erosion algorithm is probably able to simulate directional erosion (without an auxiliary particle system and solving Navier-Stokes equations) just by adjusting the erosion radius r_s along the eroded vertex. We want to implement and test this generalization, which could be used to simulate wind erosion in a restricted area (e.g., to model sand dunes).

Apparent extension of our method is a multi-layered surface model in the manner presented by Benes at al. [4]. Unlike the grid layers in triangulated terrain there could be problems in the communication between the terrain layers, because cells in different layers may not correspond with each other. Solving of this problem in real-time could be challenging.

The Holy Grail of terrain modeling is to create an interactive 3D terrain model. There were some attempts to develop such a method, but, nowadays, there is no interactive fully 3D method, which allows to handle both user edits and erosion. We want to develop such a method, but this 3D version will be probably done by another team member.

10 Conclusion

We have presented the current state of the art for sand-covered terrain modeling, some related general terrain modeling papers, necessary geometry know-how, and a few related methods for haptic visualization of terrains.

We have also presented our novel approach for interactive sand-covered terrain modeling. Our model is based on the adaptive Delaunay triangulation. The triangulated model allows to achieve any desired level of detail, and if the details exist only locally we can save a great of amount memory. In the triangulated terrain there are also no shape restrictions for user tools or the terrain features. However, our method has a certain overhead in the erosion computation and mesh updates. According to measurements the grid approach is beneficial for terrains with uniform level of detail and for applications which require global edits, such as a falling rain. Overall, the adaptive nature of our algorithm makes it suitable for interactive editing operations.

The developed solution is general enough to incorporate different materials, advanced physics for material relocation, and more sophisticated user editing tools. Also the proposed solution for visualization of multiple haptic materials is unique.

References

[1] Cgal, Computational Geometry Algorithms Library. `http://www.cgal.org`.

[2] D. Adalsteinsson and J. A. Sethian. A fast level set method for propagating interfaces. *Journal of Computational Physics*, 118:269–277, 1994.

[3] N. Bell, Y. Yu, and P. J. Mucha. Particle-based simulation of granular materials. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 77–86, New York, NY, USA, 2005. ACM Press.

[4] B. Benes and R. Forsbach. Layered data representation for visual simulation of terrain erosion. In *SCCG '01: Proceedings of the 17th Spring conference on Computer graphics*, pages 80–86, Washington, DC, USA, 2001. IEEE Computer Society.

[5] B. Beneš and X. Arriaga. Table mountains by virtual erosion. In *Proceedings of Eurographics Workshop on Natural Phenomena*, volume 1, pages 33–40, 2005.

[6] B. Beneš, E. Dorjgotov, L. Arns, and G. Bertoline. Granular material interactive manipulation: Touching sand with haptic feedback. In *Proceedings of the 14-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 295–304, 2006.

[7] B. Beneš and R. Forsbach. Visual simulation of hydraulic erosion. *Journal of WSCG*, 10(1):79–86, 2002.

[8] M. Bertram, S. E. Konkle, H. Hagen, B. Hamann, and K. I. Joy. Terrain modeling using voronoi hierarchies. In *Hierarchical Approximation and Geometrical Methods for Scientific Visualization*, pages 89–97. Springer, 2001.

[9] J. Bredno, T. M. Lehmann, and K. Spitzer. A general discrete contour model in two, three, and four dimensions for topology-adaptive multichannel segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(5):550–563, May 2003.

[10] E. Bruneton and F. Neyret. Real-time rendering and editing of vector-based terrains. *Computer Graphics Forum*, 27(2):311–320, 2008.

[11] M.-P. Cani-Gascuel and M. Desbrun. Animation of deformable models using implicit surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 3:39–50, 1997.

[12] J. J. Corso, J. Chhugani, and A. M. Okamura. Interactive haptic rendering of deformable surfaces based on the medial axis transform. In *Proc. EUROHAPTICS*, pages 92–98, 2002.

[13] M. de Berg, M. van Kreveld, M. Overmans, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 1997.

[14] O. Devillers. On deletion in Delaunay triangulations. In *SCG '99: Proceedings of the fifteenth annual symposium on Computational geometry*, pages 181–188, New York, NY, USA, 1999. ACM.

[15] O. Devillers, S. Pion, and M. Teillaud. Walking in a triangulation. In *SCG '01: Proceedings of the seventeenth annual symposium on computational geometry*, pages 106–114, New York, NY, USA, 2001. ACM.

[16] M. T. Dickerson, R. L. S. Drysdale, S. A. McElfresh, and E. Welzl. Fast greedy triangulation algorithms. *Computational Geometry*, 8(2):67 – 86, 1997.

[17] J. Dorsey, A. Edelman, H. W. Jensen, and H. K. Pedersen. Modeling and rendering of weathered stone. In *Proceedings of SIGGRAPH '99*, volume 25(4) of *Computer Graphics Proceedings, Annual Conference Series*, pages 225–234. ACM, ACM Press / ACM SIGGRAPH, 1999.

[18] N. Dyn, D. Levin, and S. Rippa. Boundary correction for piecewise linear interpolation defined over data-dependent triangulations. *Journal of Computational and Applied Mathematics*, 39(2):179 – 192, 1992.

[19] A. Faeth, M. Oren, and C. Harding. Combining 3-d geovisualization with force feedback driven user interaction. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, GIS '08, pages 25:1–25:9, New York, NY, USA, 2008. ACM.

[20] E. Ferley, M. Cani, and J. Gascuel. Resolution adaptive volume sculpting. *Graphical Models (GMOD)*, 63(6):459–478, November 2001. Special Issue on Volume Modeling.

[21] S. F. Frisken, R. N. Perry, A. P. Rockwood, and T. R. Jones. Adaptively sampled distance fields: a general representation of shape for computer graphics. In *Proceedings SIGGRAPH*, pages 249–254, New Orleans, LA, July 2000. ACM Press.

[22] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan. Triangulating a simple polygon. *Inform. Process. Lett. 7*, pages 175–179, 1978.

[23] M. Garland. Sample data for terrain simplification. `http://mgarland.org/research/quadrics.html`, 2002.

[24] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[25] T. Gerstner. Multi-resolution visualization and compression of global topographic data. *GeoInformatica*, 7(1):7–32, 2003.

[26] X. Guo, J. Hua, and H. Qin. Touch-based haptics for interactive editing on point set surfaces. *IEEE Computer Graphics and Applications*, pages 31–39, 2004.

[27] C. W. Hirt and B. D. Nichols. Volume of fluid (vof) method for the dynamics of free boundaries. *J. Comput. Phys.*, 39(1):201–225, 1981.

[28] H. M. Jaeger, S. R. Nagel, and R. P. Behringer. Granular solids, liquids, and gases. *Rev. Mod. Phys.*, 68(4):1259–1273, Oct. 1996.

[29] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[30] D. G. Kirkpatrick. Optimal search in planar subdivisions. Technical report, Vancouver, BC, Canada, Canada, 1981.

[31] X. Li and J. M. Moshell. Modeling soil: realtime dynamic models for soil slippage and manipulation. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 361–368, New York, NY, USA, 1993. ACM.

[32] Y. Li and K. Brodlie. Soft object modelling with generalised chainmail – extending the boundaries of web-based graphics. *Computer Graphics Forum*, 22(4):717–727, 2003.

[33] M. C. Lin, M. Otaduy, M. C. Lin, and M. Otaduy. *Haptic Rendering: Foundations, Algorithms and Applications*. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[34] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 162–170, Washington, DC, USA, 1977. IEEE Computer Society.

[35] A. Luciani, A. Habibi, and E. Manzotti. multiscale physical model of granular materials. In *Graphics Interface*, pages 136–146, 1995.

[36] G. Miller and A. Pearce. Globular dynamics: A connected particle system for animating viscous fluids. *Computers & Graphics*, 13(3):305 – 309, Elsevier, 1989.

[37] E. P. Mücke, I. Saias, and B. Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. In *Proceedings of the twelfth annual symposium on Computational geometry*, SCG '96, pages 274–283, New York, NY, USA, 1996. ACM.

[38] W. Mulzer and G. Rote. Minimum-weight triangulation is NP-hard. *J. ACM*, 55(2):11:1–11:29, May 2008.

[39] F. K. Musgrave, C. E. Kolb, and R. S. Mace. The synthesis and rendering of eroded fractal terrains. *SIGGRAPH Comput. Graph.*, 23:41–50, July 1989.

[40] B. Neidhold, M. Wacker, and O. Deussen. Interactive physically based fluid and erosion simulation. In *Proceedings of Eurographics Workshop on Natural Phenomena*, pages 25–32, 2005.

[41] K. Onoue and T. Nishita. A method for modeling and rendering dunes with wind-ripples. In *PG '00: Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, page 427, Washington, DC, USA, 2000. IEEE Computer Society.

[42] K. Onoue and T. Nishita. Virtual sandbox. In *Proceedings of Pacific Graphics'03*, pages 252–260. IEEE Computer Society, 2003.

[43] S. Osher and J. A. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on hamilton-jacobi formulations. *JOURNAL OF COMPUTATIONAL PHYSICS*, 79(1):12–49, 1988.

[44] R. Pajarola and E. Gobbetti. Survey of semi-regular multiresolution models for interactive terrain rendering. *The Visual Computer*, 23(8):583–605, 2007.

[45] H. Pedrini. Multiresolution terrain modeling based on triangulated irregular networks. In *Revista Brasileira de Geocincias*, volume 31, pages 117–122, 2001.

[46] J.-P. Pons and J.-D. Boissonnat. Delaunay deformable models: Topology-adaptive meshes based on the restricted delaunay triangulation. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1 –8, June 2007.

[47] V. Purchart. Modelling of erosion and deformation of the terrain for virtual reality. In *Student Research Conference*, FAV/ZCU, 2008.

[48] V. Purchart. The sandy terrain modeling for virtual reality. In *ACM Student Research Competition*, Prague, 2008.

[49] V. Purchart. Interactive manipulation of sand on a TIN terrain model for virtual reality. In *CECSG 2009*, pages 105–111, Budmerice, Slovakia, 2009.

[50] V. Purchart, I. Kolingerová, and B. Benes. Interactive sand-covered terrain surface model with haptic feedback. In *GIS Ostrava 2012 - Surface models for geosciences*, pages 215–223, 2012.

[51] V. Purchart, I. Kolingerová, and B. Beneš. Interactive editing of sand-covered terrains using adaptive triangular meshes. *Submited to Advances in Engineering Software*, 2012.

[52] V. Purchart, T. Pašek, I. Kolingerová, and P. Vaněček. Haptic visualization of material on TIN-based surfaces. In *Computer Vision and Graphics*, volume 7594 of *Lecture Notes in Computer Science*, pages 220–227. Springer Berlin / Heidelberg, 2012.

[53] M. Schneider. Efficient and accurate rendering of vector data on virtual landscapes. *Journal of WSCG*, 15:1–3, 2007.

[54] J. R. Shewchuk. Lecture notes on Delaunay mesh generation. Technical report, UC Berkeley, 2012.

[55] S. Sloan. A fast algorithm for generating constrained Delaunay triangulations. *Computers & Structures*, 47(3):441 – 450, 1993.

[56] SmithMicro Software. Poser, 2011, Available from `http://poser.smithmicro.com`.

[57] R. Soukal and I. Kolingerová. Star-shaped polyhedron point location with orthogonal walk algorithm. *Procedia Computer Science*, 1:219–228, 2010.

[58] J. Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 121–128, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[59] O. Stava, B. Benes, M. Brisbin, and J. Krivanek. Interactive terrain modeling using hydraulic erosion. *Eurographics/SIGGRAPH Symposium on Computer Animations SCA*, pages 201–210, 2008.

[60] R. W. Sumner, J. F. O'Brien, and J. K. Hodgins. Animating sand, mud, and snow. *Computer Graphics Forum*, 18(1):17–26, 1999.

[61] W. M. B. Tiest and A. M. Kappers. Analysis of haptic perception of materials by multidimensional scaling and physical measurements of roughness and compressibility. *Acta Psychologica*, 121(1):1 – 20, 2006.

[62] M. Treib, F. Reichl, S. Auer, and R. Westermann. Interactive editing of gigasample terrain fields. *Computer Graphics Forum*, 31(2):383–392, 2012.

[63] L. Tychonievich and M. Jones. Delaunay deformable mesh for the weathering and erosion of 3d terrain. *The Visual Computer*, 26:1485–1495, 2010. 10.1007/s00371-010-0506-2.

[64] N. v. Festenberg and S. Gumhold. A geometric algorithm for snow distribution of snow in virtual scenes. *Eurographics Workshop on Natural Phenomena*, pages 17–25, 2009.

[65] W. J. van der Laan, A. C. Jalba, and J. B. T. M. Roerdink. Accelerating wavelet lifting on graphics hardware using cuda. *IEEE Trans. Parallel Distrib. Syst.*, 22(1):132–146, Jan. 2011.

[66] R. Weibel and M. Heller. Digital terrain modelling. In D. Maguire, M. Goodchild, and D. E. Rhind, editors, *Geographical Information Systems: Principles and Applications*, volume 1, Principles, pages 269–297. Longman, Harlow, 1991.

[67] K. Weiss and L. De Floriani. Sparse terrain pyramids. In *Proceedings ACM SIGSPATIAL GIS*, pages 115–124. ACM, 2008.

[68] Y. Zhu and R. Bridson. Animating sand as a fluid. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 965–972, New York, NY, USA, 2005. ACM.

Activities

Reviewed Publications

- V. Purchart, T. Pašek, I. Kolingerová, P. Vaněček: Haptic visualization of material on TIN-based surfaces. In Computer Vision and Graphics, volume 7594 of Lecture Notes in Computer Science, pages 220-227. Springer Berlin/Heidelberg, 2012.
- V. Purchart, I. Kolingerová, and B. Benes: Interactive sand-covered terrain surface model with haptic feedback. In GIS Ostrava 2012 – Surface models for geosciences, 2012.
- V. Purchart, I. Kolingerová, and B. Benes. Interactive editing of sand-covered terrains using adaptive triangular meshes. Submited to Advances in Engineering Software, Elsevier, 2012.

Non-Reviewed Publications

- Purchart, V.: Interactive manipulation of sand on a TIN terrain model for virtual reality. CECSG 2009, pp 105–111. Budmerice, Slovakia, 2009.
- Purchart, V.: The sandy terrain modeling for virtual reality. ACM Student Research Competition. Prague, 2008 (in Czech).
- Purchart, V.: Modelling of erosion and deformation of the terrain for virtual reality. Student Research Conference. FAV/ZCU, 2008 (in Czech).

Related Talks

- 2011 – Interactive sandy terrain model based on the Delaunay triangulation. University of West Bohemia, Czech Republic (in Czech).
- 2010 – Interactive manipulation of sand using a TIN terrain model for VR. University of West Bohemia, Czech Republic (in Czech).
- 2010 – Triangulation-based interactive terrain model. Purdue University, Lafayette, Indiana, U.S.A.
- 2009 – Triangulations for Computer Graphics Applications – terrain modeling. Purdue University, Lafayette, Indiana, U.S.A.

Stays Abroad

- 2010 – Lafayette, Indiana, U.S.A. (1 week)
- 2009 – Lafayette, Indiana, U.S.A. (2 weeks)

Participation in Projects

- Modeling of natural phenomena using computational geometry, Kontakt No. ME09051, Ministry of Education, Youth and Sports of the Czech Republic, 2009 – 2010.
- Interactive geometric models for simulation of natural phenomena and crowds, Kontakt No. LH11006, Ministry of Education, Youth and Sports of the Czech Republic, 20011 – 2013.
- Triangulated Models for Haptic and Virtual Reality, Project No. 201/07/0927, Grant Agency of the Czech Republic, 2009 – 2011.
- Advanced Computing and Information Systems, SGS-2010-028.

Other Activities

- Participation on the coaching of successful student project and bachelor thesis (T. Pašek – Modeling of haptic force feedback of different materials). The student has won 4^{th} place on Student Science Conference 2012 (University of West Bohemia).
- Member of Local Organizing committee of ACM Programming Contest for Czech Republic and Slovakia, 2011.
- Coaching of student team in the Europian finale of the ACM Programming Contest, 2010.
- 4^{th} place in the ACM Student Research Competition for Czech Republic and Slovakia (Václav Purchart, supervised by Doc. Dr. Ing. Ivana Kolingerová), Praha, 2008.