A Framework for Local Terrain Deformation Based on Diffusion Theory

by

Marco Gilardi

A thesis submitted in fulfillment of
the requirements for the degree of
Doctor of Philosophy
at University of Sussex

Interactive Systems Research Group
School of Engineering and Informatics
Univeristy of Sussex
Brighton
BN1 9QT

April 2015

Declaration

I hereby declare that this thesis has not been and will not be submitted in whole or in part to another University for the award of any other degree.

Signed: Marco Gilardi

Interactive Systems Research Group
School of Engineering and Informatics
University of Sussex
Brighton
BN1 9QT

April 2015

Acknowledgements

This thesis would not have been possible without the help, support and guidance of my supervisors: Dr. Paul Newbury and Dr. Phil Watten.
I am also grateful to the group of Differential Geometry of University of Cagliari, namely Dr. Stefano Montaldo, Prof. Renzo Caddeo, Dr. Paola Piu and Dr. Gianluca Bande. Without them it would have never been possible for me to start my PhD, they helped and supported me at the start of it and made me love research during my undergraduate years under them.
Finally I would like to thank Dr. Giorgia Tranquilli and Miss Marcella Mascia for their friendship which kept me (almost) sane during these past years.

University of Sussex
Doctor of Philosophy Degree

Marco Gilardi

A Framework for Local Terrain Deformation Based on Diffusion Theory

Abstract

Terrains have a key role in making outdoor virtual scenes believable and immersive as they form the support for every other natural element in the scene. Although important, terrains are often given limited interactivity in real-time applications. However, in nature, terrains are dynamic and interact with the rest of the environment changing shape on different levels, from tracks left by a person running on a gravel soil (micro-scale), to avalanches on the side of a mountain (macro-scale).

The challenge in representing dynamic terrains correctly is that the soil that forms them is vastly heterogeneous and behaves differently depending on its composition. This heterogeneity introduces difficulties at different levels in dynamic terrains simulations, from modelling the large amount of different elements that compose the soil to simulating their dynamic behaviour.

This work presents a novel framework to simulate multi-material dynamic terrains by taking into account the soil composition and its heterogeneity. In the proposed framework soil information is obtained from a material description map applied to the terrain mesh. This information is used to compute deformations in the area of interaction using a novel mathematical model based on diffusion theory. The deformations are applied to the terrain mesh in different ways depending on the distance of the area of interaction from the camera and the soil material. Deformations away from the camera are simulated by dynamically displacing normals. While deformations in a neighbourhood of the camera are represented by displacing the terrain mesh, which is locally tessellated to better fit the displacement. For gravel based soils the terrain details are added near the camera by reconstructing the meshes of the small rocks from the texture image, thus simulating both micro and macro-structure of the terrain.
The outcome of the framework is a realistic interactive dynamic terrain animation in real-time.

List of Publications

Gilardi M., Watten P. L., and Newbury P., 'Unsupervised three-dimensional reconstruction of small rocks from a single two-dimensional image', in *Proceedings of Eurographics 2014 - Short Papers*. Strasbourg, France, April 2014, pp 29-32

Table of Contents

List of Figures

List of Listings

List of Tables

List of Symbols

Mathematical Symbols

a	Scalar value
$\mathbf{a}$	Vector value or vector field, depending on the context
$\mathbf{a}_i$	i-th component of vector (field) $\mathbf{a}$
$\mathbf{a} \cdot \mathbf{b}$	Euclidean scalar product between vectors $\mathbf{a}$ and $\mathbf{b}$
$\|\mathbf{a}\|$	Euclidean norm of vector $\mathbf{a}$
$\mathbf{a}^T$	Transpose operator applied to vector $\mathbf{a}$
∂D	Boundary of the closed domain $D \subset \mathbb{R}^n$

Differential Operators

∇f	Gradient of the scalar function f
$\nabla \cdot \mathbf{f}$	Divergence of the vector field $\mathbf{f}$
$\nabla^2 f$ or Δf	Laplace operator of the scalar function f

Physics Notation

ζ^{-1}	Mobility
D	Diffusion constant
$\bar{F}(x, y)$	Force vector field

Numerical Notation

$q_{i,j}^n$	Value of the quantity q held in the gird node (i, j) at time step n

Terrain Modelling

h or $h(x, y)$	Scalar height field

List of Acronyms

AABB: Axis Aligned Bounding Box
ALU: Arithmetic and Logic Unit
APU: Accelerated Processing Unit

BCRE: Bouchaud, Cates, Ravi Prakash, Edwards

CG: Computer Graphics
CPU: Central Processing Unit

DDH: Dynamically-Displaced Height-map
DDR: Dynamic Divisible Regions
DEM: Digital Elevation Model
DEXTER: Dynamic Extension of Resolution
DT: Dynamic Terrain
DTM: Digital Terrain Model

FDM: Finite Difference Method
FLIP: Fluid-Implicit-Particles
FPU: Floating Point Unit

GPU: Graphic Processing Unit

HS: Height Span
HSM: Height Span Map

LOD: Level Of Detail

MD: Molecular Dynamics

PCISPH: Predictor-Corrector Implicit Smoothed Particles Hydrodynamics
PDE: Partial Differential Equation

RGN: Regular Grid Network
ROAM: Real-time Optimally Adapting Mesh

SFU: Special Function Unit
SM: Streaming Multiprocessor

SPH: Smoothed Particles Hydrodynamics
SPMD: Single Program Multiple Data

TIN: Triangulated Irregular Network

CHAPTER 1

Introduction

1.1. Motivation

Natural environments are extremely complex systems which have captured the interest of the scientific community for decades. In particular in Computer Graphics (CG) natural environments have an important role as they are key in making outdoor virtual scenes believable and immersive.

A fundamental role in a natural environment is played by the terrain as it forms the background for every other natural element in the scene. Reconstruction and visualization of a terrain from digital elevation information spans many areas of scientific interest from geology and civil engineering to CG and entertainment. In particular, the production of tools to model and visualize terrains has been largely studied in recent years [**1, 2, 3**], an overview of these methods will be given in section 2.1. However, the focus of such studies neglects to take into account soil composition and aim for static visualisations of the terrain.

In CG applications terrains are often limited to a background role and they are often displayed as static elements. Terrains are often represented using height maps [**4**], which are generally associated with data structures that help generating the terrain mesh, as explained in section 2.1.1. Important examples of such data structures are the Regular Grid Network (RGN) [**5**] and the Triangulated Irregular Network (TIN) [**6**]. For interactive applications TIN is inefficient when compared with the memory friendly layout of RGN, which makes RGN a more popular choice. However, as RGN representation is based on a regular grid too many details are used in areas in which they are not needed. For this reason LOD [**7**] techniques, section 2.1.2, have been introduced in order to localize the amount of detail in areas of interest.

LODs are generally classified on the basis of how the hierarchy of meshes is built, *bottom-up* or *top-down*, on how the choice of which level of detail is done, *distance based* or *view based*, and on how the hierarchy is computed, *discrete* (generally precomputed) or *continuous* (generally in real time) [**1, 8**]. Recent hardware advances allow simulating continuous LOD cheaply on a Graphic Processing Unit (GPU) using the tessellation pipeline, section B.1.3, making it a viable technique to represent high resolution terrain meshes.

Although LOD allows large elevation databases to be displayed as a terrain mesh, the mesh representation of a terrain is limited in the number of dynamic natural effects that can be simulated. In nature terrains are not static as they interact with the rest of the environment changing shape on different levels, see figure 1.1, from micro-scale, for instance tracks left by a person running on a gravel soil, to macro-scale, such as avalanches.

Figure 1.1. Left: Static virtual terrain; Right: Natural terrain with tracks [**9**]; Terrain deformations tell the story of what happened in the area in the recent past. From the tracks on the natural terrain the viewer can deduce that a machine travelled on the terrain in the recent past. By contrast, on the static terrain no recent history can be deduced.

This dynamism in a terrain is best known in CG as *Dynamic Terrain (DT)* [**10**]. DT are mesh representations of a terrain capable of changing its morphology as a consequence of its interaction with the user or other agents in the virtual environment.

Dynamic terrains can be classified in three categories: *physics based*, *appearance based*, or *hybrid*. Physics based dynamic terrains [**10, 11, 12, 13**] use physics simulations to compute the soil slippage, which is then used to deform the terrain mesh. Although these methods produce realistic deformations they are computationally expensive and, due to the nature of soil physics, they are often difficult to implement for generic soils. By contrast, appearance based methods [**14, 15, 16**] focus on producing a believable representation of the dynamic terrain rather that a physically correct one, often through the use of physics inspired rules. The use of appearance based methods can improve performances of the simulation to the detriment of accuracy. Recently developed hybrid methods [**17, 18**] try to balance the correctness of physics based methods and performance of appearance based methods in order to obtain visual and physical realism in the simulation. The research presented in this thesis belongs to the last category.

The challenge in representing dynamic terrains correctly is that terrains and soil are vastly heterogeneous, and behave differently depending on their composition. This heterogeneity implies that many parameters need to be taken into consideration to simulate each element of the terrain at different levels. Moreover no unified physics model exists capable to describe every single material at once, further increasing the difficulty of reproducing local deformations during simulation. This work aims to address this problem and to give a solution for it.

1.2. How many particles

The methods mentioned in section 1.1 represent soils as a continuum. These representations, although efficient, are limited in considering only fine grained soils such as clay, silt, or sand, and are incapable of simulating the class of gravel soils mainly composed of pebbles and cobbles. In order to correctly represent these kind of soils a particle based representation is required.

From this observation a classification can be given for DTs as: *continuum based*, *particles based*, or *hybrid*. The continuum based methods model the terrain as a continuum and use meshes to represents it. Particles based methods, on the other hand, model the terrain as a granular material and represent it using particles. Hybrid methods simulate the terrain in one of the two models and visualize it using the other.

Granular material simulations are a well studied topic. Methods proposed in literature make use of discrete elements [**19**], spatial decomposition [**20**], Smoothed Particles Hydrodynamics (SPH) [**21, 22**], continuum based methods [**23**] or a combination of various techniques, as in *The birth of sandman* [**24**]. Although results obtained with each of the methods mentioned are impressive, simulating fine grained soils requires a large amount particles. To have an idea of how many particles are needed let's give an estimate of the number N of grains of sand in a volume V. Under the assumption that particles are approximatively spherical and have the same size, N can be estimated as:

$$N \approx \frac{V}{\frac{4}{3}\pi r^3}\varrho \tag{1.1}$$

where ϱ is the sphere packing density, and the radius r of a grain is expressed in mm. Let's consider a volume of a cubic meter, $V = 1000^3$ mm^3. Let's also assume a face centred cubic packing, see figure 1.2 on the next page, $\varrho = \frac{\pi}{\sqrt{18}}$, [**25**], and the maximum possible size for grains of sand, $\sim$ 2mm diameter, [**26**]. Under these assumptions equation (1.1) yields:

$$N \approx 1.77 \times 10^8 \text{ grains/m}^3 \tag{1.2}$$

This estimate represents an upper limit for grains of the maximum size for sand as the face centred cubic packing is a maximal packing for spheres [**25**]. As the value

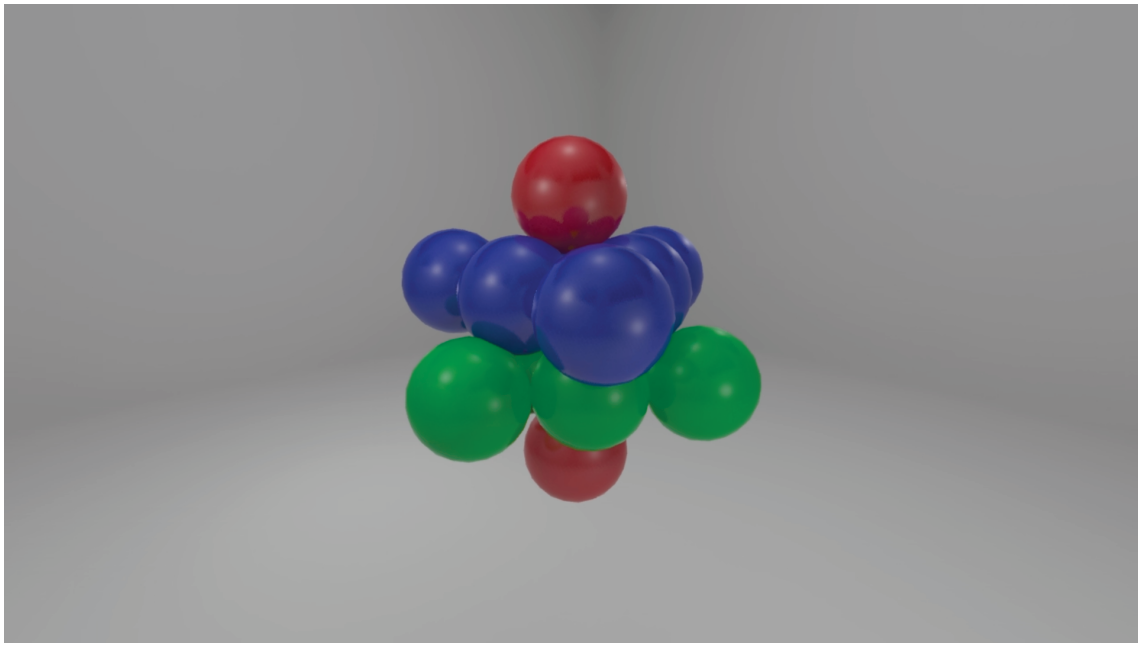


Figure 1.2. Visualisation of a face centred cubic packing of particles

of N is estimated for the biggest grains that, according with Wentworth [**26**], can still be classified as sand it is reasonable to consider it as a minimum for the class of fine grained soils. Moreover, as N in (1.2) is estimated for a volume of one cubic meter, it has to be multiplied by the number of cubic meters of sand to simulate, thus producing an even larger number of grains.

In order to cope with this large number of particles Alduán [**20**] proposes to subdivide the simulation in two steps. A first step uses low resolution particles to compute internal forces, while the second step transfers the internal forces to high resolution particles and computes external forces on them. This split is effective in simulating realistic sand, but is still computational expensive for real time simulations.

In contrast with sand, and other fine grained soils, gravel soils require fewer particles, see bottom of figure 1.3 on the following page. In fact for those soils equation (1.1) computed for different radii yields:

$$N \in [85,\ 4 \times 10^4]\ \text{grains/m}^3 \tag{1.3}$$

N varies depending whether the maximum sized cobbles, ~ 250 mm diameter [**26**], or mid sized pebbles, ~ 33 mm diameter [**26**], are considered.
These numbers are more manageable and well within hardware capabilities making possible to simulate granular materials in real time. Moreover, the visualization of

Figure 1.3. Four examples of terrain grains sizes as classified by Wentworth [**26**]. Top left: silt; Top right: sand [**27**]; Bottom left: gravel soil - granule; Bottom right: gravel soil - pebbles;

the simulation does not need to use all particles as many of them will be hidden underneath the topmost layers, thus further reducing the number of particle at rendering time. This observation can be used to further reduce the number N of grains during simulation through culling.

Assuming we wish to simulate the top thirty centimetres of soil, i.e. a volume of one meter by one meter by thirty centimetres, see figure 1.4 on the next page, the number of particles for mid sized pebbles is approximately:

$$N \approx 984 \text{ grains/m}^3 \tag{1.4}$$

The idea of culling invisible particles during simulation is developed by Zhu and Yang [**28**] which propose to replace the lower static layers in granular material simulations using height-maps, while Holladay and Egbert [**29**] suggests to cull particles which are not visible and replace the internal volume using a mesh. The advantage of Holladay and Egbert [**29**] method over Zhu and Yang [**28**] method is that it is more generic and can be applied to closed volumes, while Zhu and Yang is limited to height-maps. Either reduction method makes the simulation and visualization of granular materials more efficient as only the potentially visible, or moving, grains will be considered. Another approach proposed by Holladay and Egbert [**30**] is to

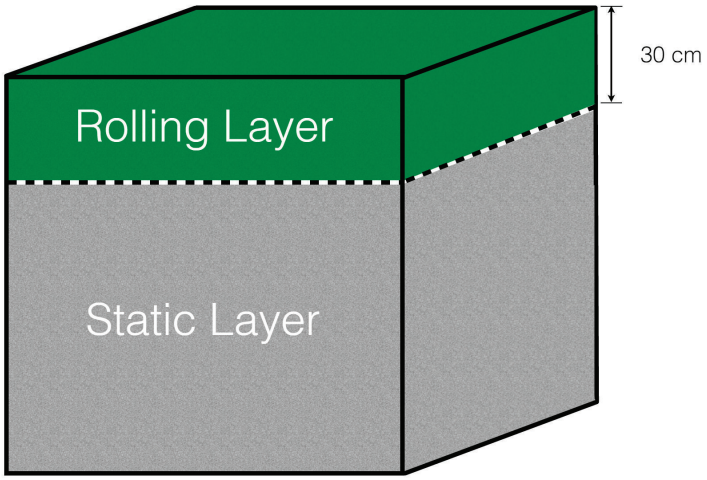


Figure 1.4. Subdivision of a soil volume in static and rolling layer

cull the particles accordingly to the camera frustum so that only particles visible to the camera are actually loaded in the system and simulated.

1.3. Framework Overview

As observed in section 1.2 it is theoretically possible to simulate pebbles and cobbles as granular material in real time in a neighbourhood of the camera, while granular simulations of sand are still too expensive for real time simulations. This observation lead to the novel idea of simulating the terrain taking into account its composition. Granular material simulations can be used in a neighbour of the camera for pebbles and cobbles terrains, while plain mesh deformation can be used for both fine grained terrains and for the parts of the terrain too far away from the camera.

A further improvement can be obtained by dividing the terrain in three ranges according with the distance from the camera: near, mid and far. If the terrain deformation is limited to near and mid ranges and the deformation of the terrain for far ranges is simulated through the use of dynamic normals displacement the amount of mesh tessellation required is reduced, saving computation.

The framework described in this thesis develops this idea by taking inspiration from the concept of DDH [**31**]. A DDH describes the intersection of an object with an hight-map based terrain via the use of a depth map rendered from underneath the terrain with the camera pointing along the up direction, this intersection information is then used to deform the terrain mesh by displacing its vertices. In this work the DDH will be obtained through a mathematical simulation of the soil slippage instead of from a depth map rendering.

The framework presented in this thesis deforms a height-map based terrain as well, but instead of relying on the sole intersection information for the displacement, a

hybrid dynamic terrain approach is used to compute the terrain deformation, see section 2.3.3. The hybrid dynamic terrain approach makes use of soil information to compute discrete solutions of a modified drift-diffusion equation for fine grained soils, while, for gravels soils, it reconstructs the meshes of the small rocks, which will be then used to render a particle simulation of the gravel soil.

The framework designed to achieve this goal is composed of five components, as illustrated in figure 1.5:

1) Regions classification;
2) Unsupervised small rocks reconstruction;
3) Terrain mesh deformation and dynamic normals;
4) Interaction area manager;
5) Physics and collision detection;

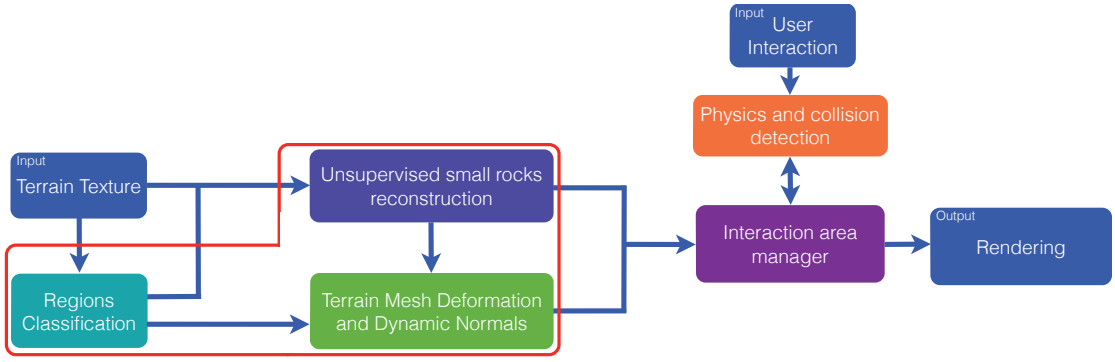


Figure 1.5. Overview of the framework. The region classification, unsupervised small rocks reconstruction and the terrain mesh deformation and dynamic normals, highlighted in the red box, are the three fundamental components that are developed in this thesis. See colour plate C.1 on page 117 for a larger version.

Soil information is obtained from the textures applied to the terrain mesh through regions classification implemented by the mean of a *description map*, see chapter 4.6. This information is then used as input to two components: the *unsupervised small rocks reconstruction* [**32**] component and the *terrain mesh deformation and dynamic normals* component.

The unsupervised small rocks deformation component obtains from the description map the textures that are composed of gravel soil and reconstructs the small rocks meshes from them. The terrain mesh deformation and dynamic normal component computes the deformations due to user interaction and stores them into a DDH. The DDH is then used as a displacement map in the domain shader during tessellation. The two components just described are directed by the interaction area manager which decides whether to load the small rocks meshes to simulate high detailed soil, to use the deformed tessellated mesh or to just use the dynamic normals obtained

from the DDH to fake the terrain deformation. If high detailed soil is being used the physics and collision detection component computes a granular simulation to drive the small rocks animation.

This thesis focuses on the design and development of three fundamental components of the framework, highlighted in red in figure 1.5 on the previous page, namely the *unsupervised small rocks reconstruction*[**32**], described in chapter 3, the *terrain mesh deformation and dynamic normals* and the *regions classification* components, described in chapter 4.
In the following a more detailed explanation of each component is given. A more detailed diagram of the framework is also given in figure 1.6.

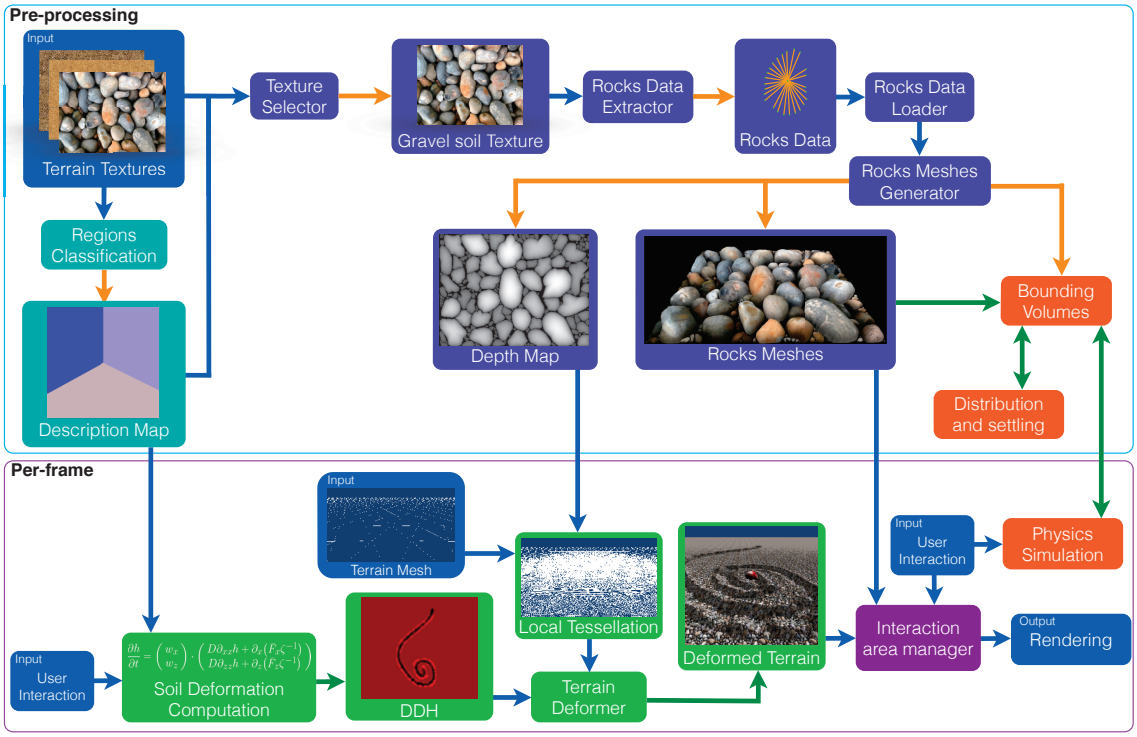


Figure 1.6. More detailed diagram of the framework. Blue arrows represents inputs, orange arrows represents outputs, green arrows represents updates. See colour plate C.2 on page 118 for a larger version.

1.3.1. Regions Classification and the Description Map. In order to take into account soil heterogeneity the terrain is subdivided in regions which are then classified depending whether they contain mainly grass, mainly pebbles, mainly generic soil, mainly mud or other kinds of terrain materials. This classification is then encoded into a description map that identify areas of the terrain with different materials. The map is then used as input for the soil slippage computations. The description map contains informations about the physics parameters of the terrain for each area, it is also linked with the textures that will be applied to each area

though the ID numbers used to identify the textures in a database.

In this thesis region classification and subsequent encoding has been performed by hand through the use of an application specifically built for this purpose, see chapter 4.6. However, this step should be automated. This automation will depend on the kind of texturing used for the terrain. If a single texture is used to texture the terrain via virtual texturing [**33, 34, 35**], a possible algorithm for automation could be the following: identify the materials using a good texture segmentation algorithm, like [**36**], use a good texture classification algorithm on each region identified by the segmentation, like [**37**], and then encode the classified regions in a lower resolution description map, which associate each texture area to each material. Alternatively, if texturing is performed using the terrain masking [**38**] the description maps could be automatically generated by mapping each single texture to a material description texture containing the physics parameters for the simulation using the slope of the terrain and the detail masks. This component runs as a pre-process stage in the framework and will be explained in chapter 4.6.

1.3.2. Unsupervised small rock meshes generation. The description map generated by the previous components is used to identify whether the area under consideration is composed by gravel soil and, if so, to obtain the texture used in the terrain area. The texture is used as input for the rocks data extraction module, see figure 1.7, which estimates a polar representation of the rock, computing the centre and 360 radii for each rock. The polar representation for each rock is stored in an array of Rocks Data data structures.

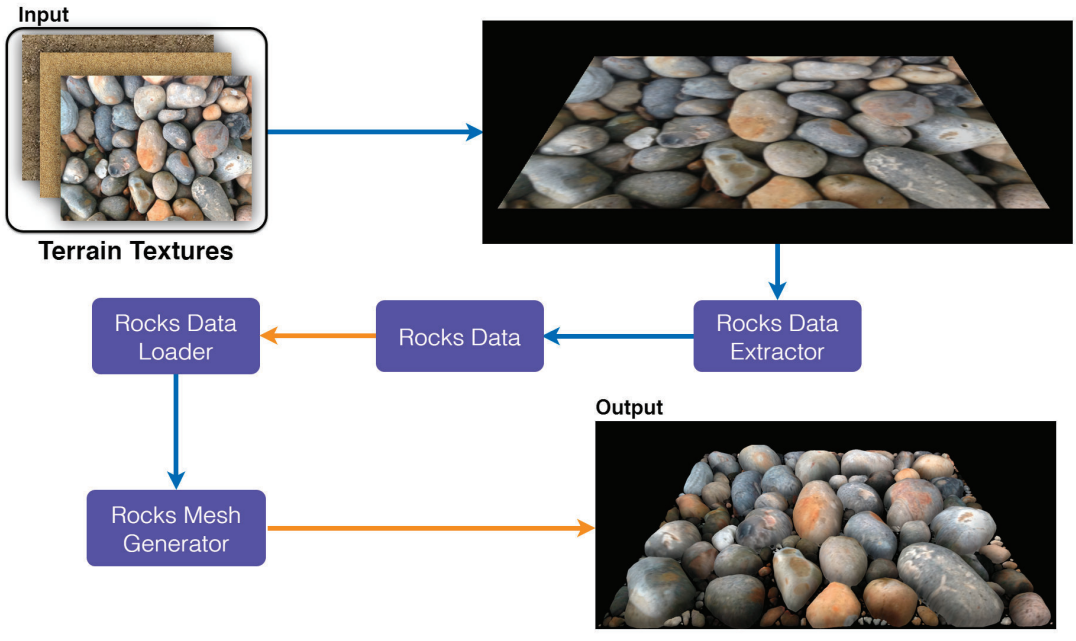


Figure 1.7. Pipeline for small rocks mesh generation from a single image

A Rock Data data structure contains the centre of the rock, the set of radii on the (x, z) plane and an estimate of the depth of the rock, which is obtained from the minimum radius computed for that rock. The Rock Data structure is used as an input for the Rocks Data Loader module. This module takes as input from the user the maximum number of segments with which the largest rock should be displayed, it then uses a simple formula to compute the optimal number of segments for all the remaining rocks depending on their size. Once the Rocks Data have been loaded in memory they are used as input for the Rock Meshes Generator module. This module give three outputs. A set of rocks meshes, used for increasing visual details near the camera, a depth map of the rocks, which is used as a displacement map in the next component, and a set of bounding volumes for the rocks, which are used in the physics component for collision detection.

This concludes the generation of small-rocks meshes from the texture, see figure 1.8 for an example of the results that can be obtained from this component. This component runs as a pre-process stage in the framework and will be detailed in chapter 3.

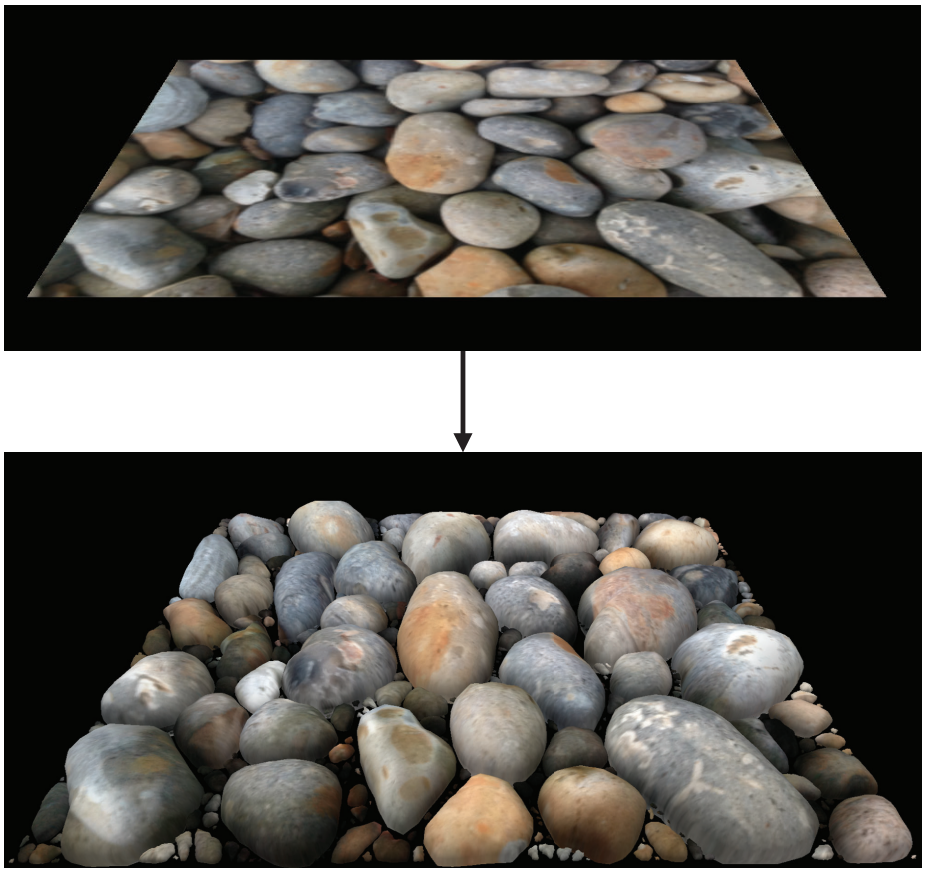


Figure 1.8. Small rocks reconstruction from a single texture as detailed in chapter 3

If the terrain is entirely composed by gravel soil, for instance pebbles, it will be computationally too expensive to simulate its dynamic by covering it with small rocks meshes. In this works it is proposed to use a three levels of details approach to simulate the terrain dynamic. The three levels of detail approach works as follows: it covers the area closest to the camera with rocks meshes and represents the areas further away from the camera using the plain original texture, while it tessellates the terrain mesh in between the nearest and the furthest areas to the camera using a displacement map to gradually increase the details. This approach switches from a discrete representation of the soil to a continuous one thus optimising the simulation.

1.3.3. Terrain deformation. The areas of terrain that are not composed by gravel soil or that are too far from the camera to be covered by actual rocks meshes need to be deformed when the user interacts with them. The deformation is computed by the soil slippage computation module, implemented on a GPU using a compute shader, on a per-frame basis as it is driven by the user interaction, The compute shader solves numerically a discretisation of a modified drift-diffusion equation, which unifies in a single model multiple material behaviours. The solutions to the drift-diffusion equation are used to update a DDH that is then used as input to the terrain deformer module, see figure 1.9.

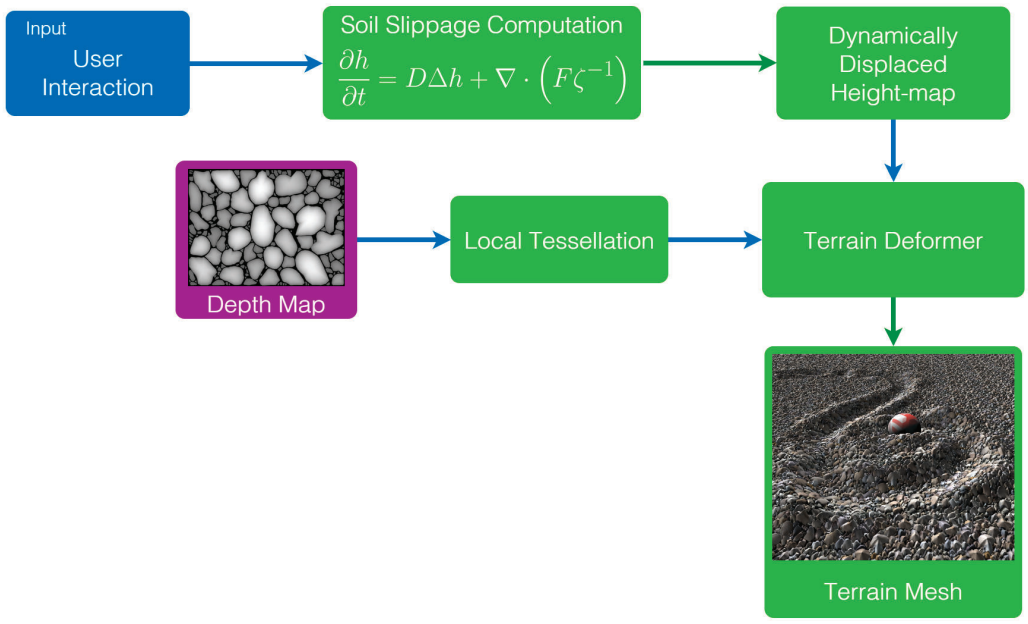


Figure 1.9. Terrain deformation pipeline

The terrain deformer module, implemented on a GPU in the tessellation stages, takes as inputs the local tessellation and the DDH, and displaces the vertices of the tessellated terrain mesh to reproduce the deformation caused by the user interaction with the terrain, for instance by dragging an object on the terrain, see figure 1.10 on the next page. The local tessellation module takes as input the depth map

generated by the unsupervised reconstruction of small rocks component and uses it as a displacement map in the regions of the terrain composed by gravel soil, the module tessellates the terrain mesh in a neighbour of the camera only.

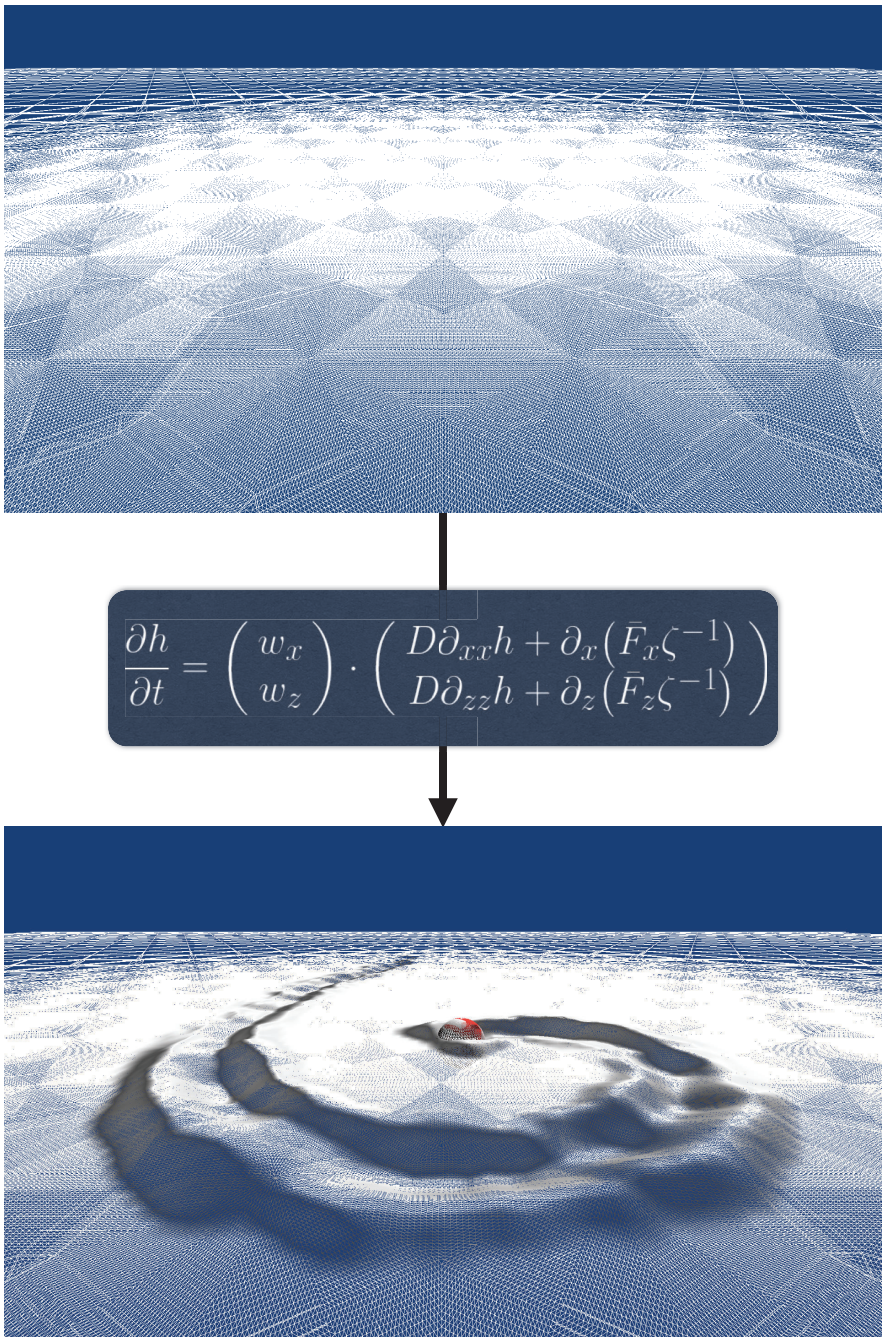


Figure 1.10. Mesh deformation using a drift-diffusion based mathematical model as detailed in chapter 4

The DDH is also used to dynamically compute the normals of the terrain where the deformation happens. The dynamic normals are then blended together with the static normals of the mesh. This approach has two benefits. Firstly, it hides the artefacts which are generated by the low resolution of the DDH, secondly it allows for multiple levels of details during the deformation. If the deformation happens

in an area away from the camera where the mesh is not tessellated there is not enough detail to displace the mesh. However, the use of dynamic normals allows simulating the displacement of the mesh by modifying the direction of the normals of the terrain. This component of the framework will be detailed in chapter 4.

1.3.4. Interaction areas. As observed above, if the terrain is completely composed by gravel soil it will be computationally too expensive to recreate every single small rock. However, if only the top layer of the soil closest to the camera is reproduced the number of particles, and small rock meshes, which needs to be handled is manageable. In order to achieve this, the terrain is covered by a grid, called the interaction grid, see figure 1.11.

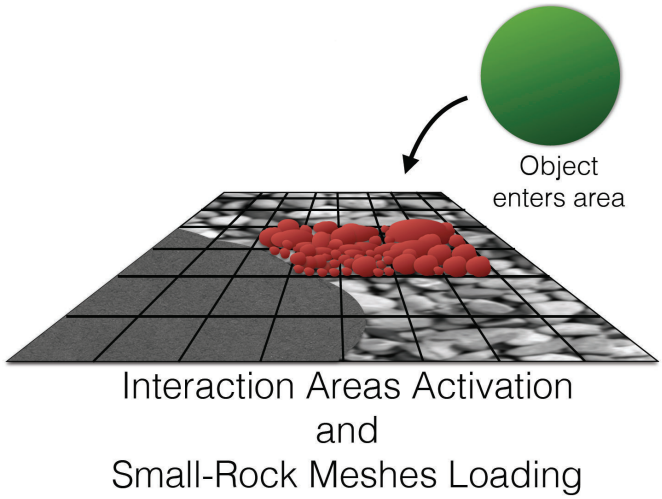


Figure 1.11. Visualisation of how the interaction areas module works

Each area of the grid becomes active whenever there is an interaction by the user in that area, the area is close to the camera and the area is partially or completely composed by gravel soil. When the interaction area become active, rocks meshes and the computation particles, which guide their movement, are loaded and used to simulate the terrain. The area become inactive when the user gets far away from it. When this happens, particles are unloaded and the underlying terrain mesh is deformed using the DDH. For areas that are not set as active the terrain deformation component is used to simulate the terrain dynamic. The interaction areas component is not further developed in this work.

1.3.5. Physics and collision detection. The final components of the framework are the physics and the collision detection. This component is in charge of computing the collision areas between objects and the terrain and of simulating granular materials, driving the animation of the areas of terrain composed by gravel soil. Although the development of this concept is out of the scope of this thesis, it is of interest to observe that in the literature there are many techniques that allow the simulation in real time of a small number of particles for granular materials.

One of the techniques which is of particular interest is the one developed by Narain, Golas and Lin in [**23**]. In [**23**] the simulation of an object falling on a box of sand is reported to run at 6.1s per frame. Although the reported simulation and rendering time is nowhere near to real time frame rate it has to be noted that Narain, Golas and Lin report using 403k simulation particles and 5.2M rendering particles on a $50 \times 40 \times 50$ grid for the simulation.

The number of simulation particles reported is one order of magnitude higher than the number of particles needed to simulate a cubic meter of mid sized pebbles, as computed in equation (1.3), and several orders of magnitude higher for the rendering of particles. Moreover, if the culling approach described in section 1.2 is used, the difference in terms of order of magnitudes increases considerably, making this method a possible candidate for simulating granular materials for real time applications. For collision detection, techniques such as those described by Aquilio [**31**] and Schaffer [**39**] can be used to accurately capture the shapes of the objects colliding with the terrain.

1.4. Summary and Contributions

The aim of this work is to present a framework to simulate dynamic terrains taking into account their composition, discuss the design and development of three of its fundamental components, and suggest possible methods for implementing the other components.

Contributions of this work are:

a) A framework for local simulation of dynamic terrains;
b) A method for unsupervised generation of rock meshes from a two-dimensional image [**32**];
c) A unifying mathematical model for real time deformation of multi material terrain mesh representations based on diffusion theory;
d) A technique for interactive simulation of dynamic terrains taking in account soil composition and its heterogeneity on GPU;

The main idea behind the framework is to use three levels of details to model terrain deformations. The lowest level of details, used for terrain areas far away from the camera, represents the soil via a texture and terrain deformations through dynamical normals displacement. The mid level of detail makes use of the tessellation pipeline and of a DDH to simulate terrain deformations by displacing the terrain mesh. Finally, the finest level of detail simulates the terrain as a granular material in the areas where the terrain is composed by gravel soil, i.e. pebbles and small rocks.

This work is organized as follows. Different techniques used to simulate dynamic terrains are considered and analysed in a state of the art review given in chapter 2. The design and development of three of the components of the framework here presented are reported in chapter 3 and chapter 4. Chapter 3 focuses on the *Unsupervised small rocks reconstruction* component of the framework and describes a method for unsupervised reconstruction of small rocks from a single image. While chapter 4 mainly focuses on the *Terrain Mesh Deformation and Dynamic Normals* component describing a mathematical model for multi-material terrain deformation based on diffusion theory and also introduces the *Regions Classification* component as a mean to edit material description maps for the terrain. Finally in chapter 5 future work is discussed and conclusions are presented.

CHAPTER 2

Literature Review

This chapter will review the current state of the art in Dynamic Terrains (DT). In section 2.1 an overview of terrain visualization is given, introducing the most common terrain representation techniques and describing Level Of Detail (LOD) methods. Section 2.2 overviews texturing techniques for terrains while section 2.3 discusses the state of the art on dynamic terrains describing techniques used for terrain interaction with users and other agents. Finally, section 2.4 gives a short overview of mesh reconstruction from a single image.

2.1. Terrain visualisation

Terrains are a fundamental part of a natural environment as they form the basis on which plants and animals can grow and live and human activities can take place. However, terrains tend to be a background element in the interaction between the user and a virtual world, as they are usually static and not modifiable by the user or other agents present in the virtual environment. In contrast, in a natural environment terrains are often modified by other agents, like human activities, by the weather or by water flowing in rivers. The discrepancy of behaviour between the natural world and virtual environments has been addressed by the introduction of the concept of Dynamic Terrain (DT) by Musgrave in 1989 [**10**], i.e. a terrain representation capable of changing its morphology due to the interaction with the user or other agents present in the virtual environment. This terrain representation will be discussed in section 2.3, in this section a brief review on terrain visualization techniques is given.

2.1.1. Data Structures for Terrain Visualization. In visualization and games a terrain is generally represented using a regular bi-dimensional grid. This representation was introduced for the first time by Miller [**4**] in 1958 for civil engineering purposes, and to the CG community by Strat [**5**] in 1978. Miller called this grid representation of elevation data *Digital Terrain Model (DTM)*, but in modern literature it is known as *Digital Elevation Model (DEM)*. Miller defined DEM as *"a statistical representation of the terrain with a system of discrete points with xyz values"* ([**4**], page 23), and consists of a set of spatial data collocated on a two dimensional array containing elevation information of the terrain. The elevation data are either generated from elevation sampling of the terrain, obtained for instance using photogrammetry techniques [**40**], for scientific or engineering purposes or, as in the

case of the entertainment industry, are created by artists or procedurally generated using fractal [**10**] or erosion [**41**] algorithms. For scientific or engineering applications the elevation data are usually stored in databases or in specific file formats, which couple elevations with georeferencing informations and other informations useful for the simulation. For entertainment purposes data are usually stored using a gray scale texture, known as *height map*, or encoding the elevation information in the alpha channel of the *normal map* of the terrain. Areas in the height map close to white (or full opacity, in the case of alpha encoding) are generally interpreted as peaks or areas with high elevation while areas close to black (or full transparency) are generally interpreted as low level ground, although the opposite interpretation is possible. Storing data in height maps is advantageous as it is easily modifiable and interpretable by artists and allow for indexed access as data are stored sequentially in memory.

Data stored in height maps are used to build the terrain mesh through the use of a bijective map, $\phi : T \to V$, from a bi-dimensional $n \times m$ texture space T to a regular grid of $n \times m$ vertices V. The map ϕ associate to each texel $t_{ij} \in T$ a vertex $v_{i,j} \in V$ with the same indexes displaced by the value h_{ij}, sometimes scaled by a certain amount, stored in t_{ij} along the height direction of the coordinate system in which the grid has been built. Assuming that the grid V as been laid down on the xz plane and that $c \in \mathbb{R}^+$ is a positive constant scaling factor, a common choice for ϕ is the following:

$$\phi_{ij} = \phi(t_{ij}) = (v^x_{i,j}, v^y_{ij} + ch_{ij}, v^z_{ij})$$

where v^k_{ij} represents the k-th component of the vertex v_{ij} located at the entry (i, j) in the grid, see figure 2.1.

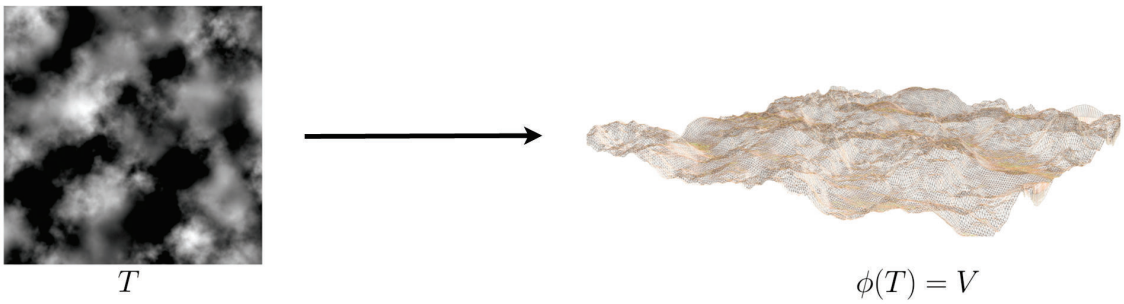


Figure 2.1. Example of bijection from an height map T and a terrain mesh V

The bijection between T and V imply that the resolution of the terrain mesh, generated with the method described, is the same as the resolution of the height map. This restriction is in general excessive and can be relaxed by refining the vertex grid after the bijection as been applied, as in the case of tessellation algorithms, section B.1.3. This generates new vertices inside each rectangle described by the vertices of the grid V, the height of these new vertices can be obtained through barycentric

interpolation. Due to the way the mesh is built this kind of construction and storage of the terrain is either known as *regular grid network* (RGN), if one wants to emphasize the data structure, or as *height map*, if one wants to emphasize the way in which the elevation data are stored on disk.

Although advantageous in terms of memory storage and information editing, the RGN representation implies that the same amount of information is used regardless its actual contribution to the final image and, due to the existence of a bijective map between the height map and the mesh, only one vertex can be associated to each height in the height map, making it impossible to generate caves or overhangs with this method, see figure 2.2.

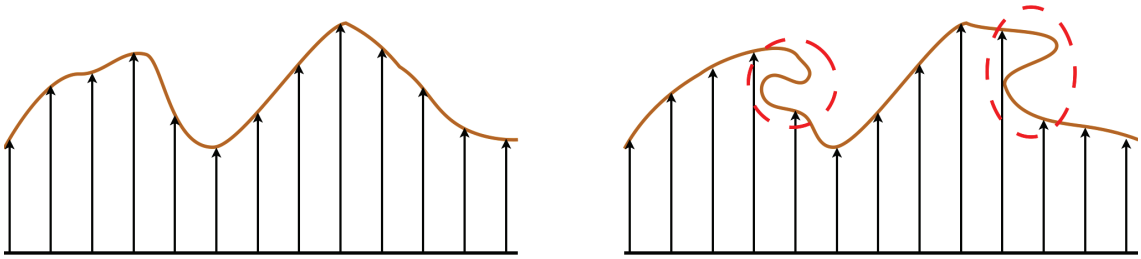


Figure 2.2. Limit of the RGN format, overhangs and caves can not be reproduced with this data structure.

In contrast with RGN, TIN [**6**] is a terrain data representation model which allows for caves and overhang representations focusing vertices in salient areas of the terrain. TIN is devised to solve the excessive resolution problem generated by the RGN in areas with low relevance for the visualization or with no elevation change from one location to the next. In fact, TIN is an adaptive representation of the terrain that concentrates information where major changes in morphology happen, allowing for an optimal usage of memory. However, in contrast with the RGN representation, TIN suffers from memory access issues. Each node in the TIN stores, together with its spatial positions and attributes, a pointer to its neighbours, making memory allocation non-sequential. The non-sequentiality of the data can result in slow memory access affecting the application performances.

Due to the fast access and the easiness of producing height maps as textures, RGN models are a popular choice for games and interactive applications. However, the constant resolution typical of this model represents an issue in terms of rendering. In fact, keeping the resolution of a mesh constant independently of the actual contribution to the final images means that algorithms for clipping and hidden surface removal have to process a large amount of vertices even when no details are needed. This observation, made by Clark [**7**] in 1976 in his seminal paper for generic objects

in a virtual scene, developed in time into the concepts of *LOD* [**7**], see figure 2.3 and *multi-resolution* meshes [**42**]. Recent hardware advances allow to mitigate the RGN problem in terms of constant resolution. In fact, the tessellation pipeline, see section B.1.3, allows to implement algorithms that focus the vertices distribution only in salient areas.

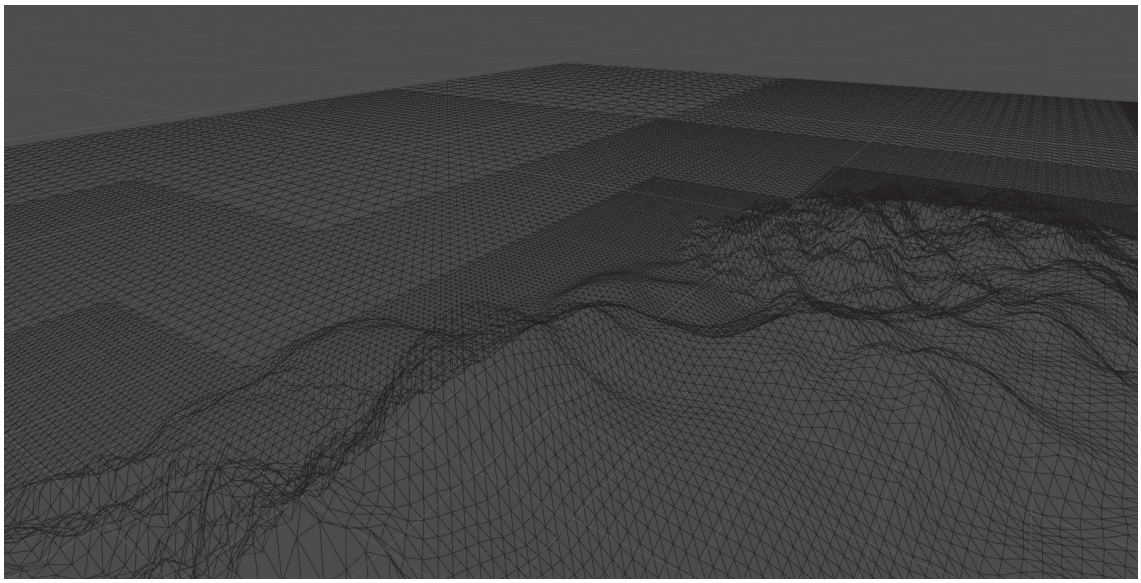


Figure 2.3. Example of Level of Details on a terrain, notice how the mesh density reduces as the distance from the camera increases

Another way to approach dynamic terrains and in general terrain representations is through implicitly describing the terrain either using iso-surfaces [**43**] or the use of voxels [**44, 45, 46**]. Voxels based terrains are volumetric representation of terrains, these terrains representations are more flexible than height map representations as they allow for the formation of overhangs and caves. However, they are computationally more expensive as the surface of the terrain needs to be recomputed after each deformation, usually using a marching cubes algorithm [**47**]. Cheaper versions of voxels representations just render the terrain surface as the set of cubes faces visible to the camera. Although cheap and fast to render these representations do not look realistic.

2.1.2. Levels of detail. The concept behind LOD is simple, as an object moves away from the point of view fewer and fewer pixels are involved in the final image, this means that less details are required to render the object and hence fewer polygons can be utilized in modelling a distant object than in modelling one in a closer neighbourhood of the camera [**1, 8**]. In order to take into account Clark's [**7**] observations it is necessary to have a way to reduce the complexity of an object preserving its silhouette and a way to determine how many polygons are needed for the object at rendering time.

Two main approaches are used to reduce the complexity of a mesh, *bottom-up* or *top-down*. In a bottom-up approach the object's mesh at full resolution is given and operations like edge collapse and vertex clustering are used to reduce the amount of polygons required to represent the object, while in a top-down approach the starting point is generally a low resolution representation of the object and operations like vertex split are used to increase the detail of the object [**1, 8**].

When generating LODs the resulting coarse model can be of a different shape than the original one, so one of the goals during the polygon reduction is to minimize such difference. In order to do so an *error metric* is introduced, and an optimization problem on the error metric is solved. Various error metrics have been proposed, based on distance between geometrical features of the model, like vertex to vertex distances or surface to surface distances, or based on human perception, however the main point of the reduction, whichever metrics is chosen, is to solve an optimization problem on the error, i.e. to find the sequence of reduction operations that reduce the number of polygons of the object and keep the error to the minimum [**1, 8**].

The bottom-up approach is generally used to produce off-line level of details as it requires to take in account the entire model during the reduction, this pre-processing generate a *discrete* number of meshes with different numbers of polygons, *discrete LODs*, that are then used in the interactive application using a top-down approach, i.e. loading the low level of detail model first and then increasing the amount of detail as needed. This approach is particularly useful for on-line 3D applications as it guarantee that even if the connection between server and client slow down the client is capable to visualize something which resembles the intended model [**1, 8**]. Moreover, this approach sits well with the tessellation pipeline allowing full use of displacement mapping to increase, or reduce, details as the camera approaches, or moves away from, an object or a region of the environment.

To determine which level of detail to use one can choose between two main approaches, a *distance based* approach or a *view dependent* approach. In a distance based approach the level of detail is chosen based on a distance metric between the point of view and the object itself. This approach does not take in account the actual frustum of the camera nor the size in pixels of the object once visualized, moreover particular care have to be taken in determining which point to use on the model for distance computation, as in certain situations certain choices generate oscillations between LODs when the object is on the boundary between two levels. To avoid these oscillations some solutions have been proposed, for example *alpha-blending* between objects, which generates an hysteresis zone around the boundaries between level of details [**8**]. In view dependent approaches the view direction is taken into

account and generally a *screen space error metric* is used to determine which level of detail has to be used [**8**]. Discrete LODs can suffer of so called *popping effects* when switching from one LOD to another, alpha-blending mitigates this issue by loading two LODs of the model and linearly interpolating their opacity in order to obtain a smooth transition between the two LODs. This approach however, can not be used when applying LOD to a terrain because in this case the object is large enough to span several LOD levels.

In contrast with the discrete approach the *continuous LOD* approach computes a transition between a coarse and a fine resolution version of an object. This computation is generally done at run time and, before tessellation was implemented on hardware, it used to be computationally heavy in some applications. This approach allows to solve problems like popping and cracking in a unified framework. In fact, in the continuous LOD operations like mesh subdivision and mesh reduction are done in such way that boundaries between different LOD levels always match, avoiding cracking of the mesh, in this way the transition between LOD levels can be done continuously. This idea is the core of techniques like *geomorphing*, whereby when a transition between a coarse LOD level and a finer LOD level is needed the vertices of the higher resolution version of the mesh are loaded on top of the vertices of the coarse resolution and then gradually moved towards their final position [**1, 8**].

For terrains LOD generation and management is generally different than for a generic object in the scene. As previously observed a terrain mesh spans different LOD levels on each frame, this poses the problem of how to generate the various LOD and how to manage them to avoid popping when the camera moves. Ideally the best approach to use for LODs on a terrain is a continuous and view dependent one as it allows to focus computations on the area in which the camera is pointing and to avoid popping and cracking. However, for an intensive interactive application like a game this approach can require excessive computation as the LOD levels have to be computed at run time [**1, 8**]. However, the recent introduction of the tessellation pipeline, see section B.1.3, mitigated the problems due to the excessive computational cost of continuous LOD techniques rendering them more viable as a solution for terrain LOD.

In order to manage LODs in terrains data structures such as quad-trees [**48**], binary triangle trees [**48**], and right triangulated irregular networks [**49**] have been proposed to smoothly increase level details. These data structures allow LOD algorithms to increase the number of triangles or quads in the mesh taking into account the connectivity between areas of low LODs and of high LODs. The use of such data structure also allow to generate subdivisions in such a way that no T-edges are formed

in the mesh between a LOD level and another. This approach avoids cracking, but still presents popping problems which have to be handled using geomorphing.

In order to produce different levels of details on the terrain various techniques have been developed in the literature. The Real-time Optimally Adapting Mesh (ROAM) algorithm was firstly introduced by Duchaineau et al [**50**] is a continuous LOD technique based on binary trees and right triangles. The idea behind the algorithm is to split an initial right triangle by its height, i.e. along the line connecting the vertex on the right angle to the mid point of the hypotenuse, and to add the new triangles as child of the initial triangle, thus generating a binary tree, see figure 2.4. The

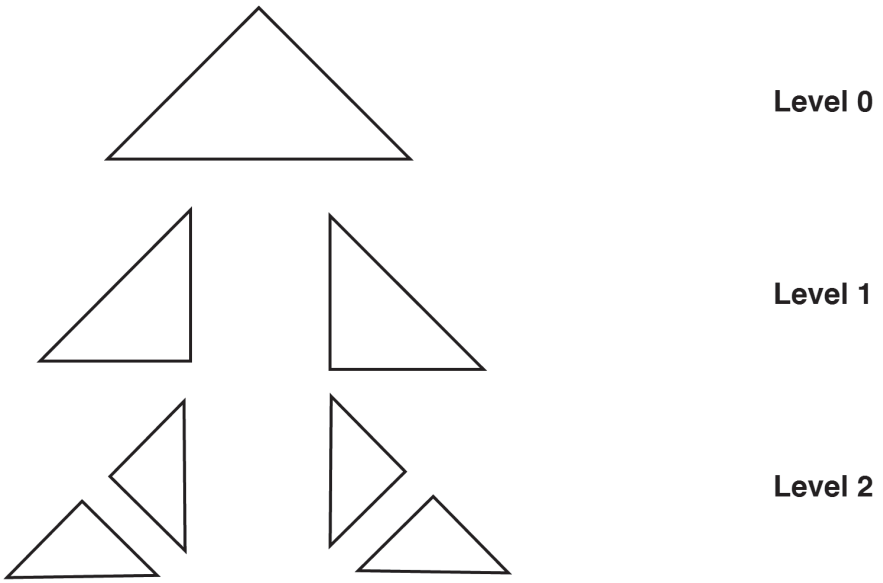


Figure 2.4. First three levels of the binary tree obtained applying ROAM to a right triangle

splitting operation is iterated on each triangle for each level until a specified initial maximum level of detail is reached. In order to describe a single quad two of such binary trees are used. In the application the pre-computed binary tree is used to split and merge triangles in real time by culling or traversing the tree to a specified level of detail, computed based on a view dependent metric, depending whether a specific area need to reduce its mesh resolution or increase it. Mesh cracking is naturally avoided by the use of two binary trees at the time to represent a single quad, for more details refer to [**50**]. The main limitation of ROAM is that for local dynamic terrains the whole terrain needs to be processed to generate the highest level of resolution, making it memory inefficient as many areas will never use that level of detail.

The Dynamic Extension of Resolution (DEXTER) view dependent algorithm developed by He [**51, 52**] is built upon ROAM adapting it to support local terrain

deformations. DEXTER is based on a regular grid and increases its resolution as follows. The initial grid is initially subdivided in axis aligned patches, called cells, which contain information about their location, size and resolution. The resolution of a specified region is increased by adding new cells in the region, thus generating a hierarchy of cells. The new cells are generated by interpolating the initial grid posts, see figure 2.5. This approach allows to extend ROAM by using the grid posts

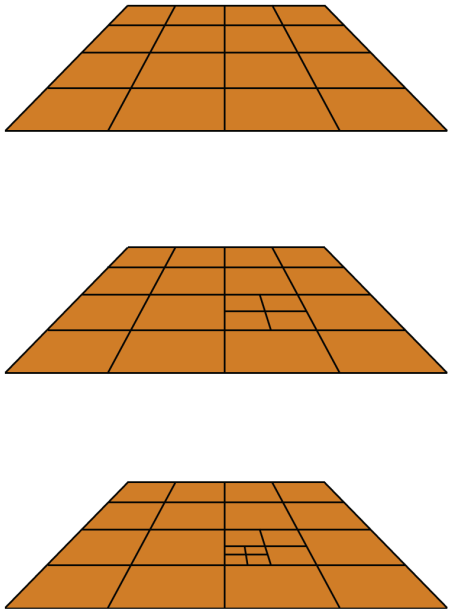


Figure 2.5. First three levels of localised grid refinement obtained applying DEXTER to a terrain patch

as new vertices for the mesh triangles thus locally extending the triangles binary trees. As the technique is based on ROAM the final mesh is naturally crack free. A drawback of DEXTER is that it is intimately coupled with the ROAM algorithm, making it hard to extend to non right triangles.

Aquilio further elaborate on the concept of local increase of resolution by developing the concept of Dynamic Divisible Regions (DDR) [**31**], which is based on DEXTER and decouples it from the specifics of the ROAM algorithm. The main advantage of using DDR is that it is a highly parallelizable technique that is well suited for GPU implementation, its main drawback is that it can be memory intensive and can produce undesired excesses of resolution in some areas. Aquilio in [**31**] describes the algorithm for implementation of DDR on CPU for: TINs, by finding each triangle centroid and splitting the triangle from it, see figure 2.6; Right triangles, by spitting the triangle connecting the hypotenuse midpoint to the vertex on the right angle, exacly as in ROAM and DEXTER; Rectangles, by splitting the rectangle using the lines passing for the midpoints of its edges, as in DEXTER. Moreover, its implementation on GPU for terrain tiles is particularly elegant. The GPU implementation of DDR given by Aquilio [**31**] is based on mapping a terrain area formed

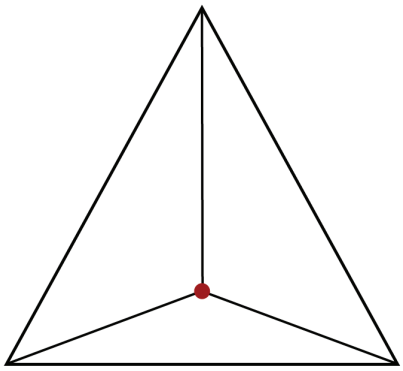


Figure 2.6. Split of a triangle from its centroid

by a $m \times n$ regular grid on a $m \times n$ texture by associating to each vertex a texel in the texture, load the texture on GPU memory and then rescale the texture to the new resolution. Finally the new texture is remapped to the original area of the terrain generating a vertex for each texel of the new texture effectively increasing the resolution of the mesh in the area.

Recently Yusov [**3**] propsed a framework for level of details on GPU which supports terrain deformation. Yusov framework takes advantage of the tessellation pipeline for the parts of the terrain close to the camera. The framework is based on quad-trees to determine the level of details for each area of the terrain and on an ad-hoc compression/decompression algorithm to store them. Although Yusov method is efficient in simulating large terrains it does not model the deformation of the terrain simply giving support for it.

2.2. Terrain texturing techniques

An important element of the terrain is the material which composes it. Terrain materials are represented through the use of textures. In order to efficiently apply textures to terrain meshes various texturing technique have been developed. In this section a short overview of the three main texturing techniques for terrains is presented.

2.2.1. Texture blending. Texture blending, also known as texture splatting, was firstly introduced by Bloom [**53**] and recently improved by Ferraris and Gatzidis [**54**]. Texture splatting is a terrain texturing technique which blend tileable textures together on GPU. The technique consists in evaluating the alpha value of different textures by using a weight function and then compose them together to generate a blended texture. Hardy and Mc Robrerts [**55**] improved the technique by using a mask in the alpha channel so that details could be preserved during blending. For instance Hardy and Mc Roberts technique allows to blend a pebbles textures and

a grass texture so that the grass only appear in between pebbles and not on the pebbles themselves, this is not possible just using standard blending techniques.

2.2.2. Masks Textures. Masks textures [**38**] are an alternative to texture blending which is more efficient in terms of memory, as it allows to use different shaders for different parts of the terrain depending on the need for high frequency details thus dynamically compositing textures for each terrain material. This technique is a procedural texturing technique which depends on three parameters: the height of the terrain, its slope and its normals expressed in world-space coordinates. These parameters are used to determine how the material looks and where it appears. For example, using the slope of the terrain to determine whether the side of a mountain should be grassy or rocky. Details to low resolution diffuse maps are added by reusing single channels of the tileable diffuse maps already in use in the shader sampled with different texture coordinates and then remapping the values to change the contrasts. This method saves memory and computation power with respect to details maps or noise functions, as it avoids loading new textures in texture memory or the computation of noise functions on GPU. To add details which can not be produced procedurally, like crop fields on a landscape or roads, sparse mask textures are used. These grey-scale textures contain information about where visual details generated by artists should appear on the terrain. As these details only cover a small amount of the terrain the mask textures are stored in a sparse quad-tree texture, thus avoiding to store a large texture which will be empty for the most part. A further saving is made by packing four sparse quad-trees together in a single texture by using a single colour channel for each quad-tree. Finally mask texturing allows to use pre-computed static destruction textures to destroy the terrain interactively simply by displacing the terrain geometry and locally changing the terrain texture.

2.2.3. Virtual Textures. Although the techniques presented so fare are very effective in texturing terrains efficiently they suffer by the fact that only a limited amount of textures and texture samplers can be used on GPU at the same time. This problem does not exists with virtual textures [**33, 34, 35**]. The virtual texture technique, as the name suggests, make use of virtual memory and consists of generating a single high-resolution texture for the whole terrain and then in loading the texture on the GPU in pages. Once loaded, these pages are stored in texture memory in a single texture which contains multiple mip-levels. The pages are loaded and unloaded depending on what part of the scene is visible and a page table is used to determine for each fragment both which textures coordinates and which mip-level have to be used. Virtual texturing is very effective in producing high-quality terrains with an high level of small details. However, depending on the hardware, when pages are being loaded for a new portion of the terrain low resolution mip-level textures could be rendered before the high resolution texture is loaded. This causes the

scene to be first rendered with low resolution textures, making it blurry, and then, as the page finish loading, to be re-rendered with the high resolution texture, thus generating texture popping artefacts as the sharp details of the texture suddenly appear in the scene.

2.3. Dynamic Terrains

Real terrains are composed of countless grains of different shapes and sizes, unfortunately this heterogeneity is not taken in account on a mesh representation. Limitations in memory availability and processing power render it impossible to simulate each single grain of terrain, to mitigate the homogeneity of meshes in representing terrains details are added using textures blended with parallax and bump maps. Although texturing is visually acceptable for static terrains, when the terrain become dynamic a large variety of phenomena are impossible to animate. Moreover, the illusion of three-dimensional details added using parallax mapping [**56**] fails, as the terrain deformation changes the mesh normals as shown in figure 2.7. With the sole use of meshes and texturing and displacement techniques it is not possible to properly reproduce detailed avalanches, terrain crumbling or explosions on a terrain.

Figure 2.7. Example of parallax mapping failing, notice that the edges of the mesh are straight instead of following the curves of the stones and the ripples caused by the low camera angle with respect to the terrain

Various approaches have been developed to simulate dynamic terrains, and can be mainly classified as *physically-based*, as *appearance-based*, or as *hybrid-based* approaches. A physically-based approach aims to use a mathematical model to describe the dynamic of the terrain, while an appearance-based approach is focused on the visual result. An hybrid-based approach is generally based on physically

inspired algorithms, but its aim is to produce visually realistic deformations rather than physically accurate simulations. Those categories can be further subdivided on the basis of the point of view they assume for the simulation of the terrain. In the *continuum* point of view, the terrain is seen as a continuous entity and modelled as a mesh, in the *discrete* based point of view terrain is modelled as a set of particles that interact with each other. Due to the high computational costs of discrete approaches continuum appearance-based approaches are most popular for real-time interactive applications, while discrete physically based approaches are generally more suitable for off-line applications. Also in this case hybrid models have been proposed as they attempt to take the best from both worlds. Thanks to the improvement of the GPU capabilities some real-time implementation of soil based on particles have been recently proposed [**57, 58, 59**], which are discussed in section 2.3.2.

2.3.1. Continuum-based systems. A first model of continuum physically-based dynamic terrains was proposed by ***Li and Moshel*** in [**11**] by introducing a physical model for soil slippage capable of interacting with a virtual bulldozer blade and the scoop of a virtual scoop-loader. In the Li and Moshel model [**11**], soil stability is based on an height map and the computation of the *failure angle* between two adjacent cells of the map, i.e. the angle over which the soil stop piling and avalanches start to happen. This computation is based on the calculation of the *factor of safety*, i.e. the ratio between the *strength force* and the *stress force*, which regulate the potential failure of a soil and hence its stability. The strength and stress forces are, respectively, the soil resistance to deformation per unit area and the force per unit area experienced by the soil, both multiplied by the total area [**11**]. In order to move the soil between adjacent cells Li and Morsel derive a system of differential equations that aim to satisfy the soil dynamics laws and the principle of volume conservation. An Eulerian approach is then taken to solve the equations that describe the dynamic of the soil, finally the soil is moved accordingly with the results of the numerical simulation from a cell to the adjacent. In order to simulate the action of a blade on the soil Li and Moshel observe that the soil under the action of the blade move up along the blade, it broken up into individual lumps and then move downwards again. To simulate this phenomena the area under the action of the virtual blade is subdivided in chunks with sizes determined by the volume of moving soil and a constant that controls the forward movement of the soil chunk in a time step. Although Li and Moshel's model is capable to describe a dynamic terrain, the effect of soil movement does not achieve an high level of realism, in fact, tyres marks are not simulated on the terrain, the soil in the model is assumed to be homogeneous, overhangs can not be created within the model and physical properties of the soil are not taken in account. However, the model proposed by Li and Moshel is the first attempt to create a physics-based solid dynamics simulation

in CG.

A different physically-based approach is taken by ***Chanclou et al*** for dynamic soil simulation. The Chanclou et al model [**12**], called *Cordis-Anima*, is based on a particle physics paradigm, i.e. each object can be represented by a system of mass points and interaction forces. The Cordis-Anima model act on two levels, 1) on a large scale level, composed by a control network of mass points and interaction forces called *interaction elements*, and 2) it describes large scale phenomena like tracks formations by mean of a *plastic bed*, i.e. a mesh deformed under the action of the control network. The plastic bed model is a simplification of the particle physics paradigm and translates a particle setting into a continuum setting. On small scale level a refinement network, linked to the control network, handle local phenomena like avalanches and soil slippage. For large scale phenomena interaction forces are modelled based on the ideal elasto-plastic model, and they can efficiently simulate non linear phenomena, while for small scale phenomena the interaction forces are linearised and modelled like a linear spring-damper force. The Cordis-Anima model successfully in simulates tyres tracks on plastic soils, but is incapable of handling water interaction and overhang formations. Due to the continuum approach effects like dust cloud formations, in cases of high velocity objects moving on the surface, are not possible, and as in [**11**] physical properties of the soil material are not taken in consideration.

Physically-based simulations of soils are generally computationally demanding, for this reason appearance-based models have been proposed. In 1999 ***Sumner et al*** [**14**] propose a continuum appearance-based model based on height maps to simulate foot or tire tracks on sand, mud or snow. Similarly to Li and Moshel [**11**], Sumner et al base their approach on checking the failure angle between neighbouring soil columns in the height map. However, in contrast with Li and Moshel, Sumner et al do not base the angle computation on a physically-based model but rather on empirical rules. If the angle α between the centre of two neighbouring columns is bigger of a certain angle α_{max} then material is transferred between the two cells until $\alpha < \alpha_{max}$. The resolution of the height map determine the minimum size of the phenomena which can be simulated with Sumner model. In order to capture the shape of the object that interacts with the terrain and be able to leave the track of the object on it Sumner et al propose the use of a contour map. A contour map is formed by a grid aligned with the height-map grid and filled with a number that measures the discrete distance of a cell from the contour of the object [**15, 16**].

The algorithm adopted by Sumner for interaction is the following:

1) Intersect objects with the ground;

2) Modify the penetrating columns reducing them to the height of the object intersection;
3) Move the material removed from the affected columns to the non penetrating columns;
4) Erode the columns outside the contour map;

Objects in motion are capable to drag material on the air, for instance by generating clouds of dust, and this phenomena is taken in account in [**14**] by introducing a particle generator. In order to optimize the algorithm, Sumner et al, detect the influence of a moving object over the terrain projecting an enlarged bounding box on the ground plane and then generating the grid nodes needed for the algorithm on the fly. This localization of the computation allows to use computational resources and memory only where interactions between the soil and the objects occurs. Moreover, to further optimize the scene management the interaction with the soil due to different objects is implemented in parallel, reducing in this way the overall computation time of the simulation. Despite this efficiency, the Sumner et al algorithm is limited in terms of what can be simulated with it. In fact, it aims to simulate tracks on a terrain and not a more generic interaction with the user, the algorithm does not run in real-time and fluid and terrain interaction is not considered. Finally, the terrain surface generated by the algorithm is unrealistically smooth and object dynamic is not influenced by the ground interaction.

The Sumner et al algorithm [**14**] assumes that the terrain is homogeneous. However, natural terrains are generally formed by a number of different materials mixed or layered on top of each other. To simulate material soil stratification voxels could be used, but this approach is computationally more expensive than height maps. To take advantage from both height maps and voxels ***Beneš and Forsbach*** [**60**] proposes a data structure for simulating layered materials on a terrain. The data structure is composed by a two-dimensional array of structures containing a list of materials and the height of the terrain in that entry. Beneš and Forsbach use this data structure to simulate *thermal erosion*, i.e. erosion due to changes in temperature. This structure allows for modelling overhangs and caves as well as water in the terrain. Computationally Beneš and Forsbach observe that the algorithms using this structure are equivalent to those using voxels, $O(n^3)$, but in practice the cost depend on the number k of layers simulated, which reduce the computational cost from n^3 to kn^2 [**60**].

The Sumner et al algorithm in [**14**] has been improved by ***Onue and Nishita*** in 2003 [**15, 16**] by the introduction of a new data structure called *Height Span Map (HSM)*, which is a modification of the layered structure proposed by Beneš and Forsbach in [**60**]. A HSM is a data structure built from a modified z-buffer placed

on the lower plane of the Axis Aligned Bounding Box (AABB) of the object. The resolution of the HSM is set to the resolution of the height map. Similarly to Beneš and Forsbach in [**60**] in each pixel of the HSM is stored a list of pairs (d, or) where d is the depth of the polygon and or is the orientation (downward or upward) of the normal of that polygon. Each list is sorted by d values and Height Spans (HSs) are generated by coupling successive downward-upward values of or. Main difference between HSM and the layered data structure of Beneš and Frosbach is that with HSM is possible to capture also objects on the terrain, while in Beneš and Forsbach layered data structure only terrain layers are stored.

The HSM representation of an object allows to describe concave objects and to represent granular material on top of them. The simulation algorithm follow Sumner et al [**14**] with some modifications, for each agent in the scene Onoue and Nishta perform the following steps, see [**15, 16**] for the details:

1) Initialize the simulation time t to zero;
2) Compute a rough collision detection between the AABB and the terrain, if the collision occur go to 3, otherwise end the simulation for the object;
3) Update the HSM;
4) Compute the collision detection between the object and the area of terrain covered by the AABB of of the object, if it is successful continue to 5 otherwise go to the end;
5) Displace the HSM accordingly with the interpenetration of the object and the terrain moving material form cells with high contour value to cell wit lower contour value;
6) Compute the collision detection again, if successful go to 7 otherwise go to 8;
7) Displace the HSM accordingly with the interpenetration of the soil and the object, from cells of high contour value to cells with low contour value, this is done as in step 5 some material could have accumulated on the boundary cells intersecting the object;
8) Erode the columns of material outside the boundary, as now all the material that was inside is accumulated there. Erosion follows the *response angle* rule, i.e. if the angle between two columns is bigger of the failure angle an avalanche should happen, hence material is moved from a columns to its neighbour;
9) Render;
10) Check whether the animation has stopped, if yes end, otherwise go to 2;

In order to simulate sand falling from an object, such as the blade of a bulldozer, a particle system similar to Sumner's is implemented, in which particles are determined by the volume of material which is falling.

Although the Onoue and Nishita algorithm moves around material and modifies the terrain, the visual results are not realistic. In fact, terrain is moved as a continuum, often stretching the texture when movement of material happens. This problem is partially solved by the introduction of *texture sliding*. Texture sliding applies to the terrain mesh a tiled texture depicting a fine grained soil and moves subregions of it accordingly with the amount of terrain moved in the simulation. This method takes advantage of the fact that in a texture that depicts fine grained soil it is hard to identify single grains when looked from a certain distance, allowing to copy and move regions of the texture around, keeping sewing borders are not recognizable. Another visual artefact produced in the Onoue and Nishita system is aliasing, as terrain is moved on a square grid by finite discrete amounts the borders of sand laying on an object are jagged, to correct this Onoue and Nishita propose to introduce a virtual span inside the object or in the surface of the terrain and use it to compute the final soil mesh. As the Onoue and Nishta algorithm is strongly based on Sumner's algorithm [**14**] it inherit the same limitations and problems. It assumes the terrain is homogeneous, disregarding the fact that real soils are composed by a number of different materials, no overhangs are possible as the underlying granular paradigm used in [**15, 16**] implements dry sand, this excludes also water interaction with the soil. As in Sumner [**14**] soil is only *one way coupled* with the objects used to modify it. Although the particle system implemented allows to simulate falling sand no dust clouds or spattering happens when an object move on the surface at high speed.

The idea of generating a HSM from the object was used and modified in 2006 by ***Aquilio et al*** [**31**] which introduces the DDH, a data structure which generate dynamic terrains on a GPU. The idea behind DDH is to project the object in a z-buffer like in [**15, 16**], but instead of computing the HS the intersection between the terrain and the object is checked. If the object depth is lower than the terrain depth the object depth is used to displace the terrain height map. More in detail, a buffer with the same size of the height map is stored in video memory in order to allocate the DDH, the algorithm performs the following steps:

1) Transform the height map in DDH;
2) Compute the terrain elevation offsets of the active terrain area rendering on a depth buffer the terrain and the objects and storing the values of depth of the objects only if they are lower of the value of depth of the terrain;
3) Use the vertex shader and the results from steps 1 and 2 to displace the vertices of the terrain mesh;
4) Repeat 2 and 3 for each frame in the simulation;

Although Aquilio et al [**31**] method is efficient and runs in real time it presents the same limitations of Sumner et al [**14**] and Onoue and Nishita [**15, 16**] algorithms,

i.e. it is not suitable to render terrain deformations other than tracks on the terrain. Moreover, Aquilio et al system is not based on the granular material paradigm, but rather on a continuum model, this make impossible for the system to simulate material on top of other objects, which the Onoue and Nishita algorithm can do, and does not allow for dust clouds formation even though the system is designed for off track cars simulations. No topological changes can occur during the terrain deformation and no overhangs can be generated with the algorithm.

Wang and Wang [**61**] apply Aquilio DDH technique to simulate craters taking into account the angle of impact with the terrain. However, the method used by Wang and Wang can not generate arbitrary craters as the offset map used for the simulation is pre-computed. ***Wang, Zhu and Xia*** [**62**] improve on Wang and Wang method by computing the offset map in real time. Although the methods proposed by Wang and Wang and by Wang, Zhu and Xia produce realistic craters they are specialized methods that do not simulate general tracks nor take into account different soil materials.

Zeng et al [**17**] modify Onoue and Nishita algorithm [**15, 16**] using a physically-based approach to determine the dynamic of soil and implement two way coupling between solids and soil. Unfortunately the approach used by Zeng et al lose the capability of simulating granular materials on top of other object and the capability of simulating concave object of Onoue and Nishita algorithm [**15, 16**]. Zeng et al [**17**] algorithm is at the high level the same as Onoue and Nishita, but modifies the collision and displacement steps in order to implement ray casting collision detection for better accuracy in the collision and the Mohr-Coulomb yield condition, which regulate the yielding of a material based on the strength stress and the shear stress of the material, for terrain displacement. In order to accelerate the ray casting process Zeng et al uses a texture based approach similar to Aquilio et al [**31**]. As in Onoue and Nishita [**15, 16**] indented objects can not leave traces on the soil and spattering is not possible in case of impact of high velocity object.

An important continuum model, which is often used in conjunction with particles models has been proposed by ***Zhu and Bridson*** [**63**]. In their model Zhu and Bridson propose to modify the standard algorithm for fluid simulation in order to integrate in it the Mohr-Coulomb yield condition. Zhu and Bridson [**63**] identify two stress tensors in their algorithm, the *frictional stress*, which is responsible for the pressure distribution in a granular material, and the *rigid stress*, which is responsible for the clustering of grains. The Zhu and Bridson algorithm makes use of particles and a computational grid in order to simulate granular materials, it works as follows.

For each time step:

1) Advect particles;
2) Apply fluid solver;
3) Evaluate strains in each grid cell and compute stresses for the cells;
4) If the rigid stress satisfy Mohre-Coulomb condition mark the cell as rigid and store the rigid stress on it otherwise store the frictional stress;
5) Identify groups of rigid cells and compute for them a velocity based on rigid motion;
6) For non rigid cells update velocity using the frictional stress;
7) Repeat from 1 until the simulation end;

Although the algorithm makes active use of particles to simulate the granular material it is classified as continuum-based as the underlying physic model is a continuum model that ignores inter-particle interactions, moreover the final result is a continuous mesh advected by the computational particles. The algorithm proposed is capable of shifting from fluid to granular material simulations depending on the initial parameter fed to the system, however interaction between the two states is not considered. The Zhu and Bridson algorithm [**63**] is successful in simulating the bulk of sliding granular materials, but as with every continuum-based approach, it lacks the capability of simulating small scale phenomena.

Zhu, Chen and Owen [**18**] propose an method based on terramechanics, which is capable of simulating different soil materials, but not in the same simulation. Zhu, Chen and Owen technique processes deformations in the vertex and fragment shaders by using three textures one for track deformation one for lateral displacement and one for tire marks. The displacements read form textures are blended with sinkage and lateral settlement factors to simulated the materials as functions of tire width, vehicle load. Although the technique produces convincing results it is limited to tire tracks as it rely on the pre computed tires displacement textures to simulate the track shape.

A different approach is followed by ***Pla-Castells et al*** [**64, 65, 66**] which develop a method for sand simulation based on cellular automata. The foundation of their method lies on modelling mathematically the static layer $s(x, y, t)$ and the rolling layer $r(x, y, t)$. They then discretise the mathematical model on a 3D cellular automata system as a series of rules each automata must follow. The final result is a convincing sand simulation capable of simulating sand depositing on other objects and being carried and deposited in different locations. However, the model presented by Pla-Castells et al is limited to sand only and is not capable of simulating multi-material terrains. Although Pla-Castells et al method is highly parallelisable a parallel approach is not developed in their work as they run their simulation on a

single thread on Central Processing Unit (CPU).

Recently ***Schäfer et al*** [**39**] proposed a general technique to deform objects by voxelising objects in the collision area to compute mesh displacements and using a Catmull-Clark subdivision surfaces to obtain high frequency deformations on a mesh. Although when applied to terrains this method is capable to capture the high frequency shapes of the object colliding with the terrain it does not take into account the material of the terrain nor its behaviour during interaction, deforming the terrain simply by direct displacement.

Although continuum models for terrain simulation are efficient, they lack realism when detailed phenomena need to be simulated or they are composed by different materials. These limitations motivate research into particle based models for terrain simulation.

2.3.2. Discrete-based systems. ***Bell et al*** in [**19**] develop a physical model based on Molecular Dynamics (MD) [**67**] to simulate interaction forces between non spherical particles and interaction between rigid objects. The MD model has been chosen by Bell et al as it efficiently simulates heaps of grains and collisions. In the Bell et al model [**19**] a grain is approximated by a collection of spherical particles "glued" together, a grain result then becomes a non-spherical particle. The choice of generating grains as non-spherical particles is due to the fact that those kinds of particles are capable of better simulating response angles, stick-slip behaviours and frictions between grains. Interactions between particles are obtained by modelling contact, normal and shear forces between single grains, while friction forces are implicitly modelled by contact forces thanks to the non-spherical particles. Collision between particles are optimized using hashing methods and updating the table only when and where changes happen between frames. Interactions between rigid bodies are handled building a layer of particles on the actual mesh and using it to handle the granular-solid interactions. Integration of the system is performed using an adaptive Runge-Kutta-Fehlburg method [**68**], in accordance with the following passes:

1) Update the hash structure in accordance with the positions of the particle at the current frame;
2) Compute forces;
3) Obtain derivatives and feed them to the integrator;
4) Update particles with the results of step 3 and return to step 1;

Bell et al [**19**] successfully implement a system capable of simulating *two way coupling* between granular materials and rigid bodies. Their particle treatment of rigid bodies allows them also to simulate collisions between rigid bodies at different resolutions. However, their system ignore cohesion forces between particles, this exclude water interaction with granular material. Their method also ignores the issues of

texturing the particles in order to generate more fine scale details. The simulation performances are coupled with the number of particles of the system and due to the physic treatment of the problem the simulation is not in real time.

Particle dynamics for fluid and rigid bodies has been implemented on the GPU by ***Harada*** in [**57**] and [**69**], using the discrete element method for simulating rigid bodies and SPH for simulating fluids. However, Harada does not couple fluids or rigid bodies with granular materials.

Building upon Harada [**57, 69**] works, ***Rungjiratananon et al*** [**58**] implement two way coupled granular materials with fluids, simulating for the first time water absorption in a particle based context on a GPU. As in Harada [**57, 69**] the Rungjiratananon et al simulation method is based on discrete element methods for sand simulation and SPH for fluid simulation. The interaction between the granular material domain and the fluid domain is achieved introducing for each particle a wet parameter and a wet threshold, which allows for a distinction between dry, wet and over-wet sand particles. If the wet parameter for a sand grain is under the threshold the water neighbouring particles are shrank proportionally to the wet parameter. In addition also the sand grain radius is reduced in order to simulate depressions formed by water poured on the granular material. If the wet parameter for a sand grain is over the threshold the excess of wetness is transferred to the neighbouring sand particles in order to simulate capillary action. For water-sand interaction the sand particles are treated as fluid particles with constant pressure and density, while for sand-sand interaction pressure and friction forces are computed in the discrete element methods domain as well as bridge forces for wet particles, to simulate the rigid behaviour of the wet sand. Bridge forces reduce as the wet parameter increase. When the particles are too wet the bridge forces disappear and particles behave as fluid.

In the Rungjiratananon et al model [**58**] for wet granular materials particles are treated singularly allowing them to detach themselves form the bulk of granular material, which is not possible to do using a continuum model. However, this approach to granular materials has a disadvantage in terms of scalability, as treating particles singularly means that memory have to be allocated and computations have to be performed for each one of them, limiting in this way the number of reproducible particles and the performances of the system. Moreover, the Rungjiratananon et al model [**58**] does not consider water spilling for over-wet materials or material drying as their model does not take in account temperature, pressure or any form of interaction with air over time. The GPU implementation of Rungjiratananon et al model make possible to simulate in real-time a small number of particles interacting

with water.

Narain et al in [**23**] proposed a continuum-based model for granular materials simulations, which make use of particles for rendering. Narain et al in [**23**] extend the Fluid-Implicit-Particles (FLIP) method used by Zhue and Bridson [**63**] allowing to simulate loose granular material and effects like splashing of sand when high velocity object collide with it. The Narain et al [**23**] method is physically-based and makes use of a combination of computational grid and computational particles in order to simulate a physically correct behaviour of dry lowly cohesive granular materials. In order to keep the material sampling uniform computational particles can be split or merged depending on their size. Narain et al algorithm [**23**] is the following. For each time step:

1) Accumulate values of density and velocity on the grid;
2) Correct the density;
3) Repeat until convergence or maximum iterations:
    3a) Compute friction for a fixed pressure;
    3b) Compute pressure for a fixed friction;
4) Find an intermediate value for the velocity;
5) Apply an impulse force to solid bodies;
6) Update particles:
    6a) Update velocities using FLIP;
    6b) Move particles using the intermediate velocity field found in step 4;
    6c) Update computational particles shape;
    6d) Split and merge particles as needed;

The Narain et al algorithm allows to simulate multiple materials with different densities and frictions. However, the algorithm is designed for dry materials and cohesion between particles is not considered in the model, in contrast with Zhu and Bridson [**63**] and Lenaerts and Dutré [**70**] that models cohesive materials but not loose materials. Rendering is performed using particles placed accordingly with the computational particles. As the main computations are done using a computational grid, which improve the efficiency of the method in respect of computing the forces on the particles directly, some visual artefacts appears causing aliasing during the particles motion. Narain et al [**23**] does not propose a solution to this problem. Inter-grain interaction is ignored in Narain et al [**23**] model and the material is supposed to be homogeneous.

Aldúan and Otaduy [**22**], modify Narain et al [**23**] model adapting it to a Predictor-Corrector Implicit Smoothed Particles Hydrodynamics (PCISPH) setting and solving in this way the aliasing problems observable in Narain et al [**23**].

Moreover, they introduce cohesion and friction forces in the model allowing to simulate rigid structures like sand castles. The particles paradigm used to compute the physics of the system allows to simulate free-scattering flow when particles are hit by an high speed object. Visually the model proposed by Aldúan and Otaduy [**22**] shows some regular patterns when sand is in movement, but this is probably due to the starting configuration of the simulation. Moreover, in Aldúan and Otaduy model [**22**] cohesion can't be kept for long as a rest configuration is absent, this causes the particles to slightly drift over time. Fracture as well presents some problems as they differ from the fractures presents in real sand, this is due to the lack of a brittle fracture model. Aldúan and Otaduy [**22**] does not explicitly couple water and granular materials, and porosity and permeability are not considered in the physical model, although the presence of a cohesion parameter could allow to tune the particle simulation to behave like a wet material. As in all the model presented the material under simulation is assumed to be homogeneous, both in composition and in physical properties. By the results reported in [**22**] it appears that the simulation performances depend on the number of particles used.

A first attempt to simulate heterogeneity can be found in [**59**]. ***Harada*** [**59**] has started to tackle the issue of implementing heterogeneous granular materials on an Accelerated Processing Unit (APU) architecture [**71**], i.e. an heterogeneous architecture in which the CPU and the GPU share the same die and have coupled shared memory, eliminating the bottleneck generated by sending data from CPU to GPU through a PCIExpress® bus.

One of the major problems of pure particles simulations is that the system has to take in account all the particles in the simulation even though they are not contributing to the current evolution of the system. This observation suggests that particles that are not visible and are not in movement could be removed from the simulation reducing the total number of particles. Hybrid models are based on this observation.

2.3.3. Hybrid Models. In order to take advantage of the computational efficiency of continuum-based models and keep the richness of detail, discrete-based models that are capable of producing hybrid models have been proposed. These combine continuum and discrete-based models using the continuum approach to simulate soil underlying the top layer of the terrain, which are modelled using a discrete approach.

One of the first hybrid approaches to soil simulation is the ***Lenaerts and Dutré*** approach in [**70**] to two way coupled granular material and water. Their algorithm is based on a continuum water absorption model proposed first in [**72**] and adapted

in [**70**] to fit in a unified SPH model for water and granular material simulation. In Lenaerts and Durté [**70**] sand is modelled using particle volumes, each particle has porosity and permeability attributes, which allow them to simulate water absorption. Fluid advection through the granular material is achieved using the sand-as-a-fluid model of Zhu and Bridson [**63**]. As a continuum model is used together with a particle model, a method for transferring properties from the two models is given. Wet granular materials tend to change state from granular to solid to viscous fluid depending on the level of water present in the mixture. The behaviour is regulated by stresses, classified in shearing and rigid stresses, which are used in [**70**] to determine cohesion forces between neighbouring particles. A saturation parameter controls the changes of state as follow. A null value for the saturation parameter means that the granular material is completely dry, i.e. dry sand, on the other hand a high value means that the granular material behave like a viscous fluid, e.g mud. In between lays the moist sand state, which behaves like a rigid body or a soft body depending on the level of saturation. Saturation is used to compute cohesion, viscosity and friction for the granular material. In order to visualize the sand grains rendering particles are added to the system, they are generated as *pseudo-random particles* and combined with the volume particles used to compute the simulation, for rendering purposes each volume is represented as a cluster with a fixed number of pseudo-random particles. As saturation increase the particles are substituted with a mesh to simulate wet sand in accordance with Zhu and Bridson's model [**63**], the transition from the particle visualization to the mesh visualization is achieved through alpha-blending and using the saturation parameter as the alpha value. The Lenaerts and Dutré algorithm is capable of simulating interaction between particles and water in a realistic way, and is scalable as the use of Zhu and Bridson's [**63**] model help decouple the simulation from the number of particles involved in it guaranteeing good scalability for the system. However, as the wet material model is based on Zhu and Bridson's [**63**] it inherits the same inaccuracy problems as continuum models fail to generate fine scale phenomena like splashing and spattering and it is limited to sand. Although the Lenaerts and Dutré algorithm uses particles to simulate granular materials, they cannot simulate material dispersion under water and an air film appears around the wet material when rendered underwater. Although having adopted Zhu and Bridson's model [**63**] for wet materials help them to decouple the simulation from the number of particles, this also slows down the computation process rendering their algorithm suitable for off-line rendering.

Other hybrid approaches have been recently proposed to simulate dry granular materials. ***Zhu and Yang*** in [**28**] build on Onoue and Nishita's [**15, 16**] work and on Bell et al's [**19**] granular model to propose a system based on an underlying height map structure and a particle system. Zhu in [**28**] subdivides the height map in two

layers and an exchange interface, the bottom layer is the static field simulated using an height map, the top layer is the dynamic field in which granular material motion is simulated using particles, finally the exchange interface is the interface between the static and dynamic field and in this layer particles are generated or removed as needed. In order to simulate the particle flow in the top layer the Bouchaud, Cates, Ravi Prakash, Edwards (BCRE) model [**73**] is used incorporated in a discrete element method solver, exchanges of materials between layers is regulated by an erosion and deposition model simulated maintaining a constant thickness layer of evolving interface particles. During simulation the position of all particles underneath the interface layer is stored using a hash map and then the particles are deleted. The Zhu and Yang [**28**] method allows efficient simulations of piles of sand, but fails to simulate more complex aggregations, for example as the method is based on height maps overhangs can not be simulated, moreover porosity and permeability are not taken into account in the simulation and hence water interaction is not considered. The material is considered homogeneous in both composition and stratification, ignoring moisture variation between layers. The optimization introduced by Zhu and Yang [**28**] works only in the case of particle piling as shallow layer of particles do not trigger the height map generation.

The ***Holladay and Egbert*** [**29**] model build upon [**28**] eliminating the 2D constrain caused by the height map by generating surfaces in 3D, thus allowing to create overhangs and independent sand clusters. Like in Bell et al [**19**] Holladay and Egbert use non-spherical grains obtained by gluing together spherical particles. This allows for an easier treatment of static friction between particles. Particles are modelled through discrete element methods while the solid state is modelled by means of a mesh surface. Once the grains settle down and stabilize in a pile they are classified as interior, exterior or interface. Interior grains are those that have low difference in speed between each other and are deleted from the simulation. Exterior grains are those which simulate the top layers of the material and are more subject to changes in speed due to agents interaction or avalanches, finally interface grains are those that lie on the surface of the interface mesh between exterior and interior grains; they are used to simulate frictions between the exterior layers of material and the interior one. The algorithm allows re-population of interior areas with particles when exterior particles are likely to move enough to expose interface particles. The algorithm proposed in [**29**] is the following:

1) Run the DEM simulation on exterior grains;
2) Delete interior grains settled into solid state;
3) Create interface mesh and constrain interface particles to it;
4) When interface grain are likely to be exposed locally repopulate interior layer with grains;

Although efficient Holladay and Egbert [**29**] method suffers of volume loss when passing from the DEM representation to the solid state representation, moreover the method considers only homogeneous dry materials ignoring stratifications of composition and moisture of real sand or terrains. Moreover Holladay and Egbert [**29**] model does not run in real time.

Hybrid methods are the key for linking continuum-based models with particle-based models. As continuum models are more efficient in simulating large scale phenomena they can be used for simulating terrain in the mid and far distance, while particles with an underlying continuum model, as in [**28**] and [**29**], could be used to simulate high resolution phenomena in the vicinity of the point of view.

2.4. Mesh reconstruction from a single image

As shown in section 1.2 of chapter 1 natural soil is composed of a large number of moving particles. When simulating dynamic terrains with particle based methods it is not computationally possible, in general, to generate the same amount of particles as in nature. However, in special cases, like for gravel soils, it is reasonable to generate some of the small rocks that reside in a neighbour of the point of view inside the view frustum, as discussed in section 1.2. In order to keep the visual appearance of these small rock particles realistic and consistent with the terrain mesh appearance a method for generating the small rocks meshes from the terrain texture is needed. In this section a short overview of existing methods for objects reconstruction from images and videos is given.

The basic idea behind image-based methods for the three-dimensional reconstruction of an object is to obtain a *point cloud* by retrieving the coordinates of points on the object surface through triangulation using information about how each image was taken, i.e. taking into account the motion, the field of view, the focus and the exposure of the camera, which took the image [**74**]. The point cloud is then fed to a surface reconstruction algorithm to generate the a three-dimensional mesh representation of the object under consideration, a popular algorithm for this reconstruction task is the Poisson surface reconstruction, introduced by Kazhdan et al [**75**]. The Poisson surface reconstruction algorithm transforms the problem of reconstructing of a surface from a point cloud to the Poisson problem: $\Delta\tilde{\chi} = \nabla \cdot \bar{\mathbf{V}}$, thus approximating the indicator function of the model $\tilde{\chi}$, which tells whether a point belong to the interior or exterior of the model, from the vector field $\bar{\mathbf{V}}$ obtained by smoothing the normal field of the surface. The indicator function is used to obtain the model's isosurface. This approach is capable of reconstructing a detailed mesh from the point clouds and is robust against noise.

Recently interest has grown in developing techniques for environmental reconstruction from images and videos, for example Bradley, Noworouzezhari and Beardsley [**76**] reconstruct foliage from a set of still images, Li et al focuses on the reconstruction of trees from videos [**77**] and also propose a technique to reconstruct moving bodies of water from videos [**78**]. Reconstructing a three-dimensional model of an object from a single two dimensional image is challenging due to the lack of information along one of the spatial axis. In literature the problem has been often studied for architectural purposes [**79, 80, 81, 82**], or allowing a user to guide the process [**83, 84**]. However, in the literature there is no reference to the use of an image to guide the unsupervised generation of small rocks meshes. Kita and Miyata [**85**] use a reference image to describe the general shape of the mosaic in their pebbles mosaic generator but not to describe each rock shape and texture. Liu and Xing [**86**] model a rock starting from an image of its material interfaces. However, they focus on reproducing a single rock with an high level of detail. By contrast, procedural generation of rocks has been addressed by Peytavie, Galin et al. [**87, 88**] which describe a method to aperiodically tile surfaces with rocks; on the same line of work Sakurai and Miyata [**89**] use Voronoi cells to generate and pile different kinds of rocks, while Dart, de Rossi and Togelius [**90**] use three-dimensional L-systems to procedurally generate rocks. Although these approaches are very effective in producing random or regular distributions of small rocks, none of them use the texture of the terrain where the small rocks should lie to generate the meshes and the texture or the small rocks. Chapter 3 focuses on describing a new method to reproduce a large number of small rocks that have appearance, position and shape defined by a given two-dimensional single image.

2.5. Summary

In this chapter the main algorithms for DT simulations have been reviewed together with a short overview on methods for three-dimensional mesh reconstruction from a single image. Continuum-based algorithms are efficient for large scale simulations of terrains but lack the capability to simulate fine scale details and free-scattering flow. On the other hand discrete-based model are capable to simulate small scale phenomena but at the cost of an increased request of computational power and memory. As shown hybrid models that combine particles and continuum models have been proposed, however they tend to focus mainly on dry sand or homogeneous materials. However, natural terrains are generally heterogeneous and they are rarely formed by a single material as terrains are in general layered both in composition and on the level of moisture. These observations are often ignored during terrains modelling or granular material simulations as terrains are generally considered static and only the top layer is considered important for the simulation. Another aspect often overlooked is the reconstruction of soil particles from the texture. Although

it is not possible to reconstruct soil particles for fine grained soils it is reasonable to do so for gravel soils. A general real-time terrain deformation model that takes into account the terrain composition to simulate multiple materials at the same time and that is capable to model small scale details for gravel soils where needed is still missing as the review presented in this chapter showed. The research presented in this work addresses this problem and proposes a framework for its solution.

CHAPTER 3

Unsupervised small rocks reconstruction from a single image

This chapter describes the *Unsupervised small rocks reconstruction* [**32**] component of the framework proposed in this thesis, the purple component highlighted in figure 3.1 and is part of the pre-processing stage of the framework.

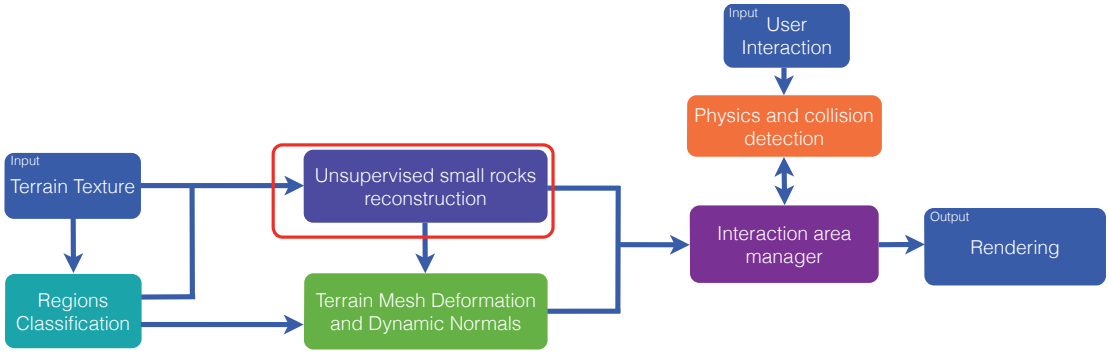


Figure 3.1. Overview of the system. The Unsupervised small rocks reconstruction component is highlighted in red.

The basic mathematical concepts and definition needed for this chapter can be found in appendix A. In this chapter *generalized ellipsoids* [**32**] are introduced as a novel way to generate organic small rock meshes, section 3.1. Moreover, a new way to identify outer edges of small rocks reducing texture noise, and a method to describe the equatorial slices of a small rock given a binary image using the signed turning angle of a polygonal curve are introduced [**32**], section 3.2.1. The mesh reconstruction from the data obtained from the binary mask is described in subsection 3.2.3 and a novel method for determining the LOD based on the size of each small rock is given in section 3.3. The software engineering is briefly discussed in section 3.4 while results are presented in section 3.5. Finally, a discussion on how to use geometry instancing to reproduce a large number of small rocks from a single reconstruction batch is given in section 3.6.

3.1. Generalised Ellipsoids and a small rock's parametrisation

The general shape of small rocks and pebbles can be roughly approximated by an ellipsoid. However, in nature small rocks are not always convex, in fact weathering processes generate concave shapes, which can not be approximated by a simple ellipsoid.

In order to better approximate the shape of a small rocks parametrization (A.1) has to be modified. To make these modifications a definition will be needed:

DEFINITION 3.1 (Star shaped set). Let S be a subset of $\mathbb{R}^n$ and let $C \in S$ be a point in S. The set S is said to be *star shaped with respect to* C if for each point $P \in S$ the segment $\overline{CP}$ is entirely contained in S, see figure 3.2. If S is star shaped with respect to each of its points then S is said to be a *star shaped set*.

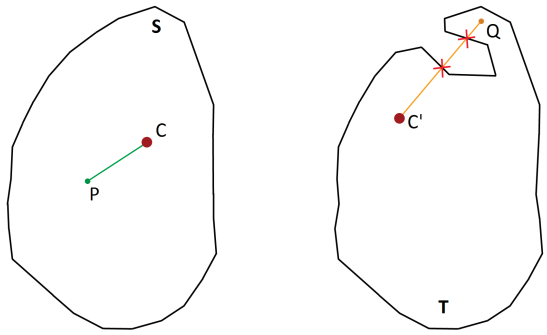


Figure 3.2. Left: Example of star shaped set with respect to C, for each P there exists a segment $\overline{CP}$ entirely contained in **S**. Right: Example of star shaped set with respect to C', the segment $\overline{C'Q}$ is not entirely contained in **T**

Let's recall parametrization (A.1) on page 103 of appendix A:

$$\mathbf{X_E}(\varphi,\vartheta) = \begin{cases} x(\varphi,\vartheta) &= a\cos(\varphi)\cos(\vartheta) \\ y(\varphi,\theta) &= b\sin(\vartheta) \\ z(\varphi,\vartheta) &= c\sin(\varphi)\cos(\vartheta) \end{cases} \tag{3.1}$$

where $\mathbf{X_E} : U \to \mathbb{R}^3$, $U = [0, 2\pi) \times (-\frac{\pi}{2}, \frac{\pi}{2})$ is an open subset of $\mathbb{R}^2$, and a, b, and c are the semi-principal axes of the ellipsoid . These three numbers determine the shape of the ellipsoid. This observation is the key for modifying parametrization (3.1). In fact, if these constants are substituted with continuous functions of the angles φ and ϑ the shape of the ellipsoid can be modified to fit non-convex objects. The limitation of this representation is that the object being described by it must be star shaped with respect its centre and it must have genus zero, see section A.2 of appendix A.

Let $f(\varphi,\vartheta)$, $g(\varphi,\vartheta)$ and $h(\varphi,\vartheta)$ be three continuous functions from $\mathbb{R}^2$ into $\mathbb{R}$. Parametrization (3.1) can be rewritten as:

$$\mathbf{X_{G_E}}(\varphi,\vartheta) = \begin{cases} x(\varphi,\vartheta) &= f(\varphi,\vartheta)\cos(\varphi)\cos(\vartheta) \\ y(\varphi,\theta) &= h(\varphi,\vartheta)\sin(\vartheta) \\ z(\varphi,\vartheta) &= g(\varphi,\vartheta)\sin(\varphi)\cos(\vartheta) \end{cases} \tag{3.2}$$

as choosing the three functions as constants yields back the parametrization of an ellipsoid. In the following parametrization (3.2) will be referred to as the parametrization of a *generalised ellipsoid*.

To obtain a regular generalised ellipsoid the functions $f(\varphi, \vartheta)$, $g(\varphi, \vartheta)$ and $h(\varphi, \vartheta)$ have to respect the following conditions:

1) $f(\varphi, \vartheta) > 0$ and $g(\varphi, \vartheta) > 0$ for each $(\varphi, \vartheta) \in U$

2) $\nabla h(\varphi, \vartheta) \mid_{(\varphi, -\frac{\pi}{2})} = \mathbf{0}$ and $\nabla h(\varphi, \vartheta) \mid_{(\varphi, \frac{\pi}{2})} = \mathbf{0}$ for each $\varphi \in [0, 2\pi)$

Condition 1 requires that the functions f and g are positive and do not intersect the y axis, while condition 2 requires that at the poles the generalized ellipsoid stays smooth.

In the following generalised ellipsoids are used to reconstruct small rocks from a single image. A key concept for the reconstruction is the concept of *equatorial slice*.

Definition 3.2 (Equatorial Slice). Let Ro be the set of points of $\mathbb{R}^3$ belonging to a rock resting on a plane π. A *shadow* Sh of a small rock is the set of points of π obtained by projecting orthogonally Ro on π along a given direction $\mathbf{v}$. A shadow Sh of Ro will be said to be an *equatorial slice* of Ro and noted E_s if the direction of projection $\mathbf{v}$ coincide with the normal $\mathbf{n}$ of π.

As the small rock can rest on a plane in different positions the equatorial slice for a small rock is not uniquely defined. Another concept that will be useful in the following is the concept of a neighbour of a point.

Definition 3.3 (Neighbour of a point). Let $\mathbf{p}$ be a point of $\mathbb{R}^n$, a *neighbour* $N_\mathbf{p}$ of $\mathbf{p}$ is the set of points $\mathbf{q}$ of $\mathbb{R}^n$ such that the Euclidean distance from each point $\mathbf{p}$ and $\mathbf{q}$ is less or equal to a positive small value ϵ. In symbols:

$$N_\mathbf{p} = \{\mathbf{q} \in \mathbb{R}^n \mid d(\mathbf{p}, \mathbf{q}) \leq \epsilon\}$$

where $d(\mathbf{q}, \mathbf{p})$ is the Euclidean distance. If the equality condition is dropped the neighbour $N_\mathbf{p}$ of $\mathbf{p}$ will be said an *open neighbour*.

Given an equatorial slice E_s, a point $\mathbf{p}_i = (x_i, y_i) \in \mathbb{R}^2$ is said to be an *interior* point of E_s if there exists a neighbour N_i of $\mathbf{p}_i$ entirely enclosed in E_s. $\mathbf{p}_i$ is said to be on the border ∂E_s of the equatorial slice if there exists a neighbour N_i of $\mathbf{p}_i$ that has a non-empty intersection with E_s, but it is not entirely enclosed in it. Finally, a point is said to be *exterior* to E_s if N_i has an empty intersection with E_s. The set of interior points of an equatorial slice will be denoted by $\mathring{E}_s$. It is easy to see that $E_s = \mathring{E}_s \cup \partial E_s$.

In the following the *centroid* of E_s is defined as a point $\mathbf{C} \in \mathring{E}_s$ such that it has minimum distance from each point p of the border of E_s. Given an equatorial slice E_s of a small rock and its centroid $\mathbf{C}$ the *set of radii* for the equatorial slice can

be defined as the set R of vectors $\mathbf{r} = \mathbf{p} - \mathbf{C}$, which connect $\mathbf{C}$ to each point $\mathbf{p} \in \partial E_s$. Under the condition that E_s is a star shaped set, given the centroid $\mathbf{C}$ of the equatorial slice E_s and its set of radii R it is possible to describe the border of E_s through the parametrization:

$$\mathbf{X}_{\partial E_s}(\varphi) = \begin{cases} x(\varphi) &= f(\varphi)\cos(\varphi) \\ z(\varphi) &= g(\varphi)\sin(\varphi) \end{cases} \tag{3.3}$$

where $\varphi \in [0, 2\pi)$, and $f(\varphi)$ and $g(\varphi)$ are two real functions such that for each $\varphi \in [0, 2\pi)$, $\mathbf{r}(\varphi) = (f(\varphi), g(\varphi)) \in R$. Parametrization (3.3), represent the border of the equatorial slice on the plane. To reconstruct the surface of the rock, parametrization (3.3) can be extended using parametrization (3.2) on page 44, all that is required is to identify a third function $h(\varphi)$ which describes the height of the rock. Having such function the rock surface can be described as:

$$\mathbf{X}_{Rock}(\varphi, \vartheta) = \begin{cases} x(\varphi, \vartheta) &= f(\varphi)\cos(\varphi)\cos(\vartheta) + \mathbf{C}_x \\ y(\varphi, \vartheta) &= h(\varphi)\sin(\vartheta) + \mathbf{C}_y \\ z(\varphi, \vartheta) &= g(\varphi)\sin(\varphi)\cos(\vartheta) + \mathbf{C}_z \end{cases} \tag{3.4}$$

where $\mathbf{X_{Rock}} : U \to \mathbb{R}^3$, $U = [0, 2\pi) \times (-\frac{\pi}{2}, \frac{\pi}{2})$ and $\mathbf{C}$ is the position of the centre of the small rock in space.

An example of application of parametrization (3.4) is the procedural generation of rocks using noise functions on a sphere, for example using spherical Worley noise [**91**] one can obtain results similar to what shown in figure 3.3.

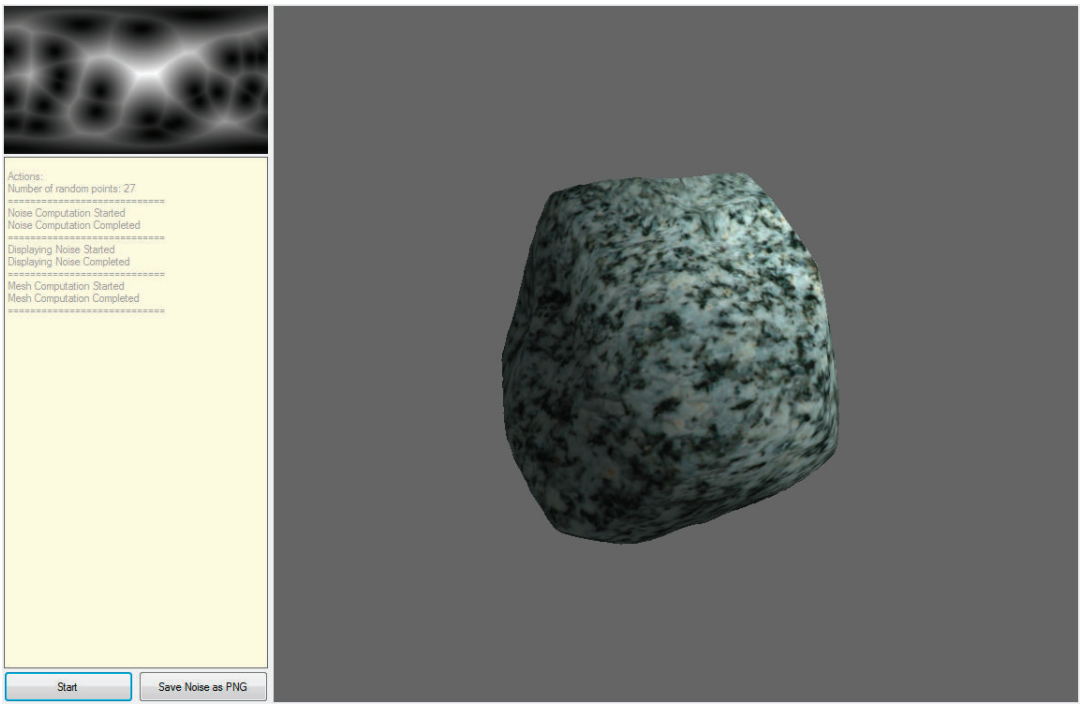


Figure 3.3. Application which generates procedural rocks using the components of spherical Worley noise as functions in parametrization (3.4)

For the purposes of reconstructing small rocks from a single image the functions $f(\varphi)$ and $g(\varphi)$ are condensed in a single function $r(\varphi) = ||\mathbf{r}(\varphi)||$. Also choosing $h(\varphi)$ as constant in φ, specifically $h = \min_{\mathbf{r}\in R}(||\mathbf{r}||)$, results in reasonably smooth small rock shapes. With these choices the parametrization of a small rock becomes:

$$\mathbf{X}_{Rock}(\varphi,\vartheta) = \begin{cases} x(\varphi,\vartheta) &= r(\varphi)\cos(\varphi)\cos(\vartheta) + \mathbf{C}_x \\ y(\varphi,\vartheta) &= h\sin(\vartheta) + \mathbf{C}_y \\ z(\varphi,\vartheta) &= r(\varphi)\sin(\varphi)\cos(\vartheta) + \mathbf{C}_z \end{cases} \tag{3.5}$$

which will be used in the following to generate the mesh of the reconstructed small rocks.

In this section a mathematical abstraction for a small rock has been given. The abstraction identifies a specific small rock once the centroid $\mathbf{C}$ of the equatorial slice of the rock and the sets of its radii are given. The problem now consists in retrieving those two parameters from a single image.

3.2. Rocks data extraction from a single image

Let's consider the photo in figure 3.4. The goal is to extract for each of the N small rocks depicted on it a centroid $\mathbf{C}_i$ and a set of radii R_i, $i = 0, \ldots, N-1$, which can then be used in parametrization (3.5) to obtain a mesh representation of each rock. The method described in this section assumes that the picture of the small

Figure 3.4. Picture of a terrain covered by small rocks. The picture has been taken so that the view direction is orthogonal to the plane where the rocks rest. Each rock image is the equatorial slice of the rock photographed.

rocks is taken so that the view direction of the camera is orthogonal to the plane

on which the small rocks rest, as in figure 3.4. Under this condition perspective can be ignored during the reconstruction process. This orthogonality assumption also ensures that each small rock depicted in the image is actually the equatorial slice of the real small rock. The method for obtaining the centre and radii of the small rocks is depicted in figure 3.5. The method can be divided into the following steps:

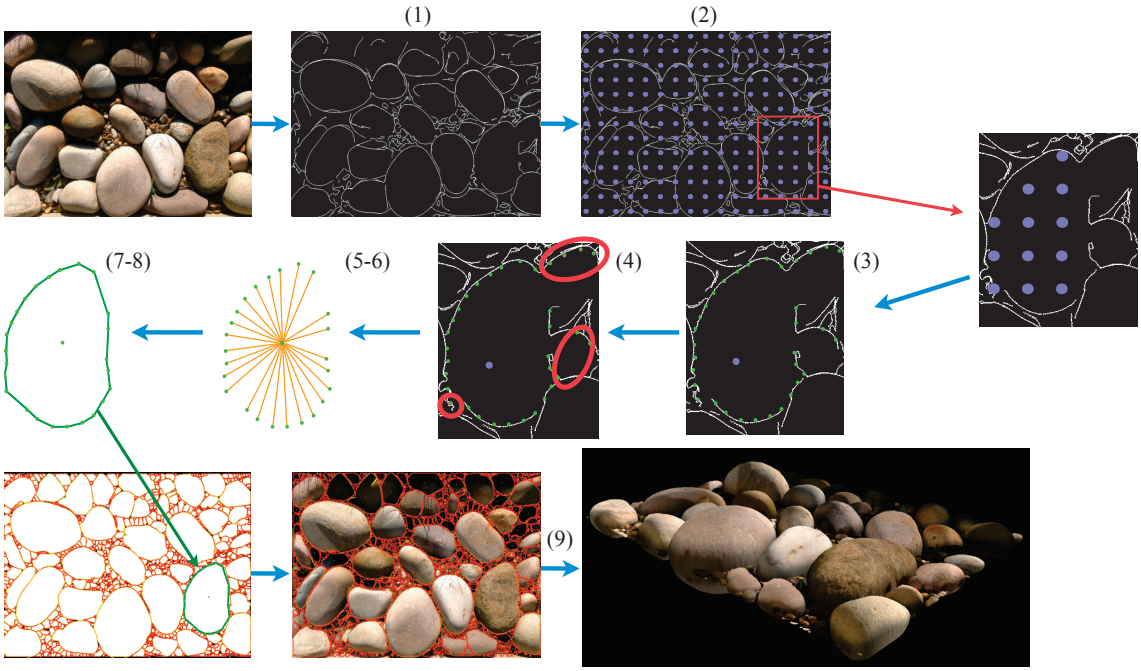
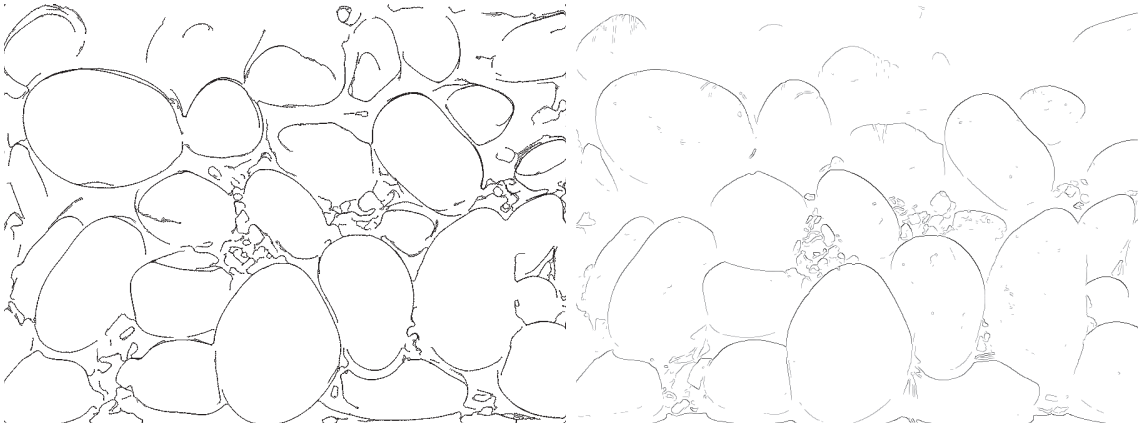


Figure 3.5. Rock models extraction from a single image. See colour plate C.3 on page 119 for a larger version

1) Obtain a binary mask from the initial image which highlight the (incomplete) edges of each equatorial slice of each small rock;
2) Sample the mask to obtain an initial position in the interior of the each equatorial slice;
3) For each position shoot a set of rays in n directions and find the intersection of each ray with the edge of the equatorial slice;
4) Filter each set of intersections to remove outliers;
5) Compute the centroid of the equatorial slice from the set of intersections;
6) Repeat steps 3 to 5 until the centroid stabilises;
7) For each set of filtered intersections fill the gaps due to removed outliers generating points from a Bezier curve;
8) Compute the set of radii for the small rock from the resulting set of intersections and the centre;
9) Use the radii and the centre to generate a mesh for the rock using parametrization (3.5);

In the following each step will be detailed.

3.2.1. Step 1: Mask generation. The first step of the algorithm aims to obtain the boundaries ∂E_s of each equatorial slice. The ideal output for this step would be a mask that contains all complete edges belonging to the boundaries of each equatorial slice. However, existing algorithms do not fulfil this expectation. Edges in an image can be found through standard edge detection such as Canny [**92**], but these methods often detect edges caused by the changes in the texture of the object that do not necessarily belong to the edges of the equatorial slice, see figure 3.6b. Moreover, these edges are often incomplete.



(a) **(b)**

Figure 3.6. Comparison of the Canny algorithm (b) with the method developed in this work (a). Notice that in (a) edges due to textures are ignored and that more edges belonging to the equatorial slices' boundaries are detected.

Another approach to obtain the boundaries of each equatorial slice is to segment the image using the watershed algorithm [**93, 94, 95**]. The watershed algorithm identifies closed contours of objects in an image by interpreting the grey-scale version of the image as a height map and conceptually flooding it from its local minimum. This approach separates areas of low elevation by generating watersheds when two body of water meet. These watersheds constitute the contours of the objects in the image. Applying the watershed algorithm to an image straight away produces over-segmentation and an extra merging pass is necessary to obtain the correct contours of the objects. To eliminate this problem a foreground, i.e. the objects whose contour are of interest, and a background, i.e. the rest of the image, have to be identified. This can be done by labelling the objects either manually or automatically. The result is an image split in background and foreground objects, see figure 3.7 on the next page. However, in the case of the images under analysis there is no such division as all small rocks in the image are of interest. The watershed algorithm could be used to complete edges obtained with a Canny edge detection algorithm as shown at the top of figure 3.8 on page 51. However, as figure 3.8 shows, the edges obtained in such a way do not follow the edges of the rocks closely, and generates false edges due to the rock texture.

Figure 3.7. Output from the watershed algorithm with automatic labelling applied to figure 3.4. The red area is identified as background by the algorithm, coloured areas are identified as foreground.

The bottom of figure 3.8 shows the segmentation obtained using the algorithm that will be described in the following sections. Although the algorithm defined in the following sections over-segments some areas of the image the small rocks boundaries obtained are a better fit than those obtained with the techniques described described in 2.4. Moreover, the over-segmentation happens mostly in areas where very small rocks should be, helping to reconstruct them as well.

In order to obtain a mask which captures as much complete edges as possible the initial image I is passed through a median filter, which has the property of preserving sharp edges and reducing noise produced by the texture of the small rocks. The following coarse definition of edge will be used in what follows.

Definition 3.4. Let $f : I \subset \mathbb{Z}^2 \to [0, 1]$ be a function that maps each pixel of the $w \times h$ image I to an intensity value $f(x, y)$. An *edge* in the image I is a significant local change in the intensity f of I.

The problem with definition 3.4 above is that the meaning of "significant" is not specified. Local changes on a continuous function can be obtained by the mean of the gradient operator, pixels whose gradient is higher of an arbitrarily fixed threshold are labelled as edges. This gradient approach is the foundation of Canny algorithms. However, the arbitrary nature of the threshold makes hard to find a single value that works well with a large number of images. To overcome this the approach to determine what is a "significant local change" is revisited in the following.

Figure 3.8. Top: output from the watershed algorithm applied to edges detected using Canny. Bottom: Segmentation obtained using the method described in this section superimposed to the original image.

A local change happens in a pixel p whenever:

$$\begin{aligned} f(x,y) &< \tau f(x+\varepsilon, y) \\ f(x,y) &< \tau f(x-\varepsilon, y) \\ f(x,y) &< \tau f(x, y+\varepsilon) \\ f(x,y) &< \tau f(x, y-\varepsilon) \end{aligned} \tag{3.6}$$

where $\tau \in [0,1]$ is a threshold value described in the following, and ε is a small positive value. The value τ is used to slice the image intensity in a set of L levels L_i, $i \in [0, L]$, if a significant change in the image at level L_i is detected a value of $\frac{1}{L}$ is added in the same location to an image I_L initially initialized to black. After all the intensity levels have been processed the image I_L contains all the intensity

changes in the initial image I. Significant changes in I corresponds to brighter pixels in I_L, to isolate these pixels an hysteresis thresholding process is performed using two threshold values t_H and t_L. The resulting image I_T contains all the significant changes in I. However at this stage the edges of the small rocks are still disconnected, this is partially corrected by analysing the connectivity of each pixel and linking the edges whenever possible. Although this process yields an image I_E containing edges that are true edges for the small rocks it still produces a set of edges that is not complete, this is addressed in step 7 in the following section.

3.2.2. Steps 2 to 8: Equatorial Slices description. In order to describe each equatorial slice E_{Si} their centroid $\mathbf{C}_i$ and set of radii R_i, $i = 0, \ldots, N-1$, where N is the number of small rocks in the image, have to be identified. In order to find these data the image I_E is covered by a regular grid, each of the M nodes in the grid corresponds to a positional sample $\mathbf{S}_k$, $k = 0, \ldots, M-1$, of the image. A sample $\mathbf{S}_k$ is selected from the grid and rays are shot from it in H different directions $\mathbf{d}_\ell$: $\mathbf{s}_{k\ell} = \mathbf{S}_k + t\mathbf{d}_\ell$, where $\ell \in 0, \ldots, H-1$ and $t \in \mathbb{R}^+$, see figure 3.9. For

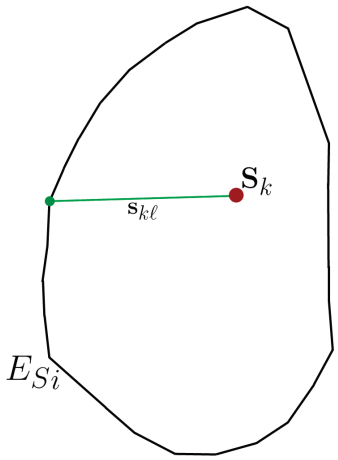


Figure 3.9. Segment $\mathbf{s}_{k\ell}$ starting from the sample $\mathbf{S}_k$ and intersecting the edge of the equatorial slice E_{Si}

each segment $\mathbf{s}_{k\ell}$ its intersection with an edge in I_E is found using ray marching. The set R_{ki} of H intersections from $\mathbf{S}_k$ will contain points that belong to the true edge of E_{Si} and, if the edge is incomplete, points that belong to the edge of a neighbouring equatorial slice. In the following, points that do not belong to the true edge of the small rock will be called *outliers*. To obtain a good representation of each E_{Si} the outliers have to be removed from each R_{ki}. To identify the outliers the following definitions will be useful.

Definition 3.5 (Degree of a point). Let V be a finite set of points $\mathbf{v}_\ell$ of $\mathbb{R}^n$, $\ell = 0, \ldots, H-1$, and E be the set of segments defined by $e_\ell = \overline{\mathbf{v}_\ell \mathbf{v}_{\ell+1}}$ for $\mathbf{v}_\ell, \mathbf{v}_{\ell+1} \in V$, see figure 3.10 on the next page. The number of segments intersecting on $\mathbf{v}_\ell$ is called the *degree* of the point and it is noted as $deg(\mathbf{v}_\ell)$, the degree is a natural number.

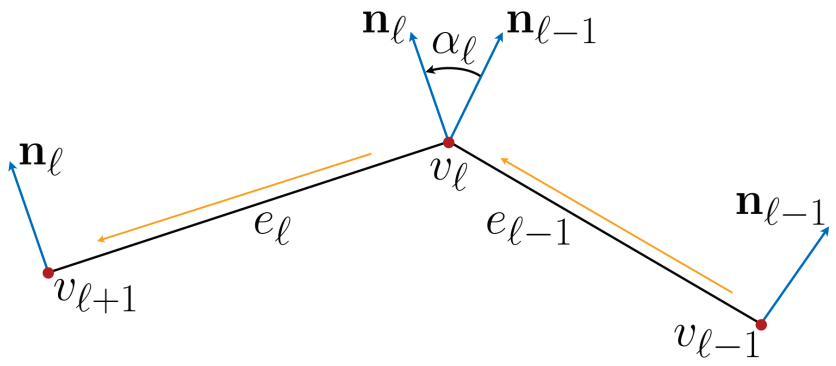


Figure 3.10. Graphical representation of the concepts in definitions 3.5, 3.6 and 3.7

DEFINITION 3.6 (Polygonal chain and Polygon). A *polygonal chain* P in $\mathbb{R}^n$ is defined by the pair (V, E) where V is called the set of *vertices* of P, and E is called the set of *edges* of P with the condition that $0 < deg(\mathbf{v}_\ell) \leq 2$ for each vertex $\mathbf{v}_\ell \in V$, $\ell = 0, \ldots, H-1$, see figure 3.10. A *closed* polygonal chain, i.e. a polygonal chain for which $\mathbf{v}_0 = \mathbf{v}_{H-1}$, is called a *polygon*.

DEFINITION 3.7 (Signed Turning Angle [**96**]). Let P be a polygonal chain, $P = (V, E)$. The *signed turning angle* α_ℓ at vertex $\mathbf{v}_\ell \in V$, $\ell = 0, \ldots, H-1$, is the signed angle between the normals, $\mathbf{n}_\ell$ and $\mathbf{n}_{\ell+1}$, to the edges intersecting in $\mathbf{v}_\ell$, see figure 3.10. For $deg(\mathbf{v}_\ell) < 2$, α_ℓ is defined as 0.

The sine and cosine of the signed turning angle α_ℓ can be computed by:

$$\begin{aligned} \cos(\alpha_\ell) &= \mathbf{n}_\ell \cdot \mathbf{n}_h \\ \sin(\alpha_\ell) &= ||\mathbf{n}_\ell \times \mathbf{n}_h|| \end{aligned} \tag{3.7}$$

The problem of identifying the outliers in R_{ki} is solved using the following geometric approach. A polygon P_{ki} is constructed using R_{ki} as the set of vertices of P_{ki} and defining the edges as the segment $e_\ell = \mathbf{v}_\ell \mathbf{v}_h$ for $\ell = 0, \ldots, H-1$ and $h = (\ell + 1) \bmod H$. For each vertex $\mathbf{v}_\ell$ of P_{ki} the singed turning angle α_ℓ is computed, a vertex $\mathbf{v}_\ell$ is marked as an outlier and removed from R_{ki} if the following condition is satisfied:

$$\begin{aligned} \cos(\alpha_\ell) &< a\alpha_T \\ \sin(\alpha_\ell) &< b\alpha_T \end{aligned} \tag{3.8}$$

where a and b are two real numbers and α_T is a threshold angle. Conditions (3.8) are repeatedly checked until no new outliers in R_{ki} are identified. The remaining points in R_{ki} are then used to compute an approximation of the centroid $\bar{\mathbf{C}}_0$ of E_{Si}, from this new point the process is repeated generating a new point $\bar{\mathbf{C}}_1$. From this new point the process is iterated again generating a new point $\bar{\mathbf{C}}_2$. This iteration process is carried on until the sequence of approximated centroids $\bar{\mathbf{C}}_s$ stabilizes, i.e. $||\bar{\mathbf{C}}_s - \bar{\mathbf{C}}_{s-1}|| < \epsilon$ where ϵ is a positive small number. The stabilization point is considered as the centroid $\mathbf{C}_i$ of the equatorial slice. From this point $\mathbf{C}_i$ the process

is repeated one last time, outliers are found and removed but this time the gap is filled by points generated from a Bezièr curve generated using the vertex tangents to the polygonal on the vertices immediately before the gap and immediately after it. This final set of intersections E_C is then used to produce the set of radii R_i for the equatorial slice E_{Si}. Before selecting a new sample $\mathbf{S}_k$ the samples that are inside the equatorial slice just processed are removed from the grid. The whole process is repeated until no samples remain on the grid.

3.2.3. Setp 9: Mesh generation. This section gives the details to generate the vertex and index buffers and the vertex attributes for the small rock mesh.

3.2.3.1. *Vertex and Index buffers.* Once the set of radii R_i and the centroid $\mathbf{C}_i$ are identified for each equatorial slice E_{Si} the set of vertices for the small rock's mesh can be produced using parametrization (3.5) on page 47. Values obtained for the radii and the centre depends on the resolution of the image passed as input. In order to reconstruct the rock in world space the values have to be scaled. The scaling matrix is obtained by considering the dimensions or the rectangular area where the small rocks should lie and normalizing them respect the image width, height and diagonal length. Given a rectangular region of a plane π with width π_w, height π_h and diagonal π_d and a rectangular image I with width w, height h and diagonal d the scaling matrix is given by:

$$\mathbf{W} = \begin{pmatrix} \frac{\pi_w}{w} & 0 & 0 & T_x \\ 0 & \frac{\pi_d}{d} & 0 & T_y \\ 0 & 0 & \frac{\pi_h}{h} & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.9}$$

where $\mathbf{T} = (T_x, T_y, T_z)^T$ is the translation vector that moves the rocks form a rectangular area around the origin of the world reference system to the specified rectangular region π.

Vertices in world space are then obtained as:

$$\mathbf{V}(\varphi, \theta) = \mathbf{W} \begin{pmatrix} \mathbf{X}_{Rock}(\varphi, \theta) \\ 1 \end{pmatrix} \tag{3.10}$$

Parametrization (3.5) on page 47 generates vertices that are spatially organized along *rings*, which are generated by the parameter φ, and *segments*, which are generated by the parameter θ, as shown in figure 3.11 on the next page. The total number of rings N_R and segments N_S have to be set before the mesh generation, to reduce the number of parameters to set N_S and N_R could be linked one to the other. A possible choice for their relation is: $N_R = \lfloor \frac{N_S}{2} \rfloor + 1$, where $\lfloor \cdot \rfloor$ is the floor operator. In addition to the vertices a set of indices is generated to describe the vertices'

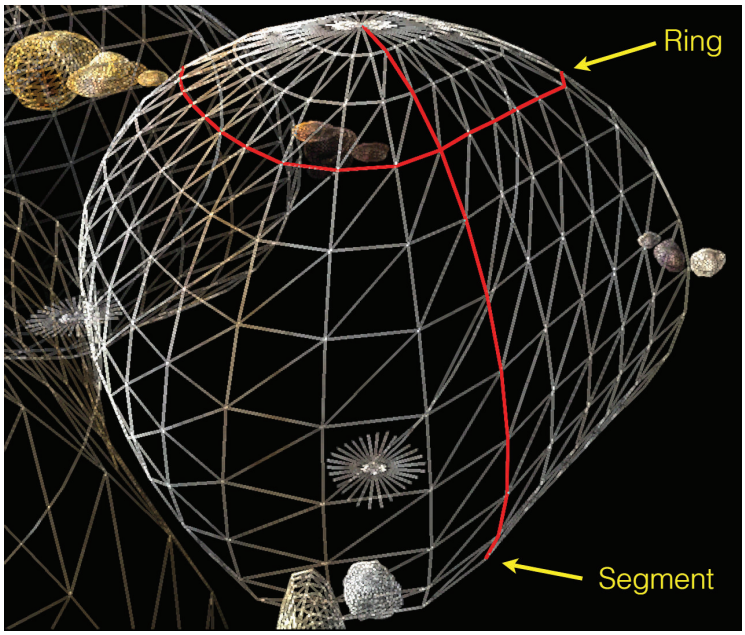


Figure 3.11. Spatial organization of vertices obtained from parametrization (3.5) on page 47

connectivity using algorithm 3.1, which is represented graphically in figure 3.12 on the next page.

Algorithm 3.1 Indexing of the small rock vertices. Each iteration defines a quad on the vertices of the small rock mesh.

$currentIndex \leftarrow 0$
$i \leftarrow 0$
for $ring = 0 \longrightarrow numOfRings$ **do**
  for $segment = 0 \longrightarrow numOfSegs$ **do**
    if $ring \neq numberOfRings$ **then**
      $index[i] \leftarrow currentIndex$
      $i \leftarrow$Increment(i)
      $index[i] \leftarrow currentIntex + numOfSegs$
      $i \leftarrow$Increment(i)
      $index[i] \leftarrow currentIntex + numOfSegs + 1$
      $i \leftarrow$Increment(i)
      $index[i] \leftarrow currentIntex + numOfSegs + 1$
      $i \leftarrow$Increment(i)
      $index[i] \leftarrow currentIntex + 1$
      $i \leftarrow$Increment(i)
      $index[i] \leftarrow currentIntex$
      $i \leftarrow$Increment(i)
      $currentIndex \leftarrow$Increment$(currentIndex)$
    end if
  end for
end for

3.2.3.2. *Texturing and Normals.* For each vertex the texture coordinates are generated by projecting the vertex coordinates in local space on the plane where the

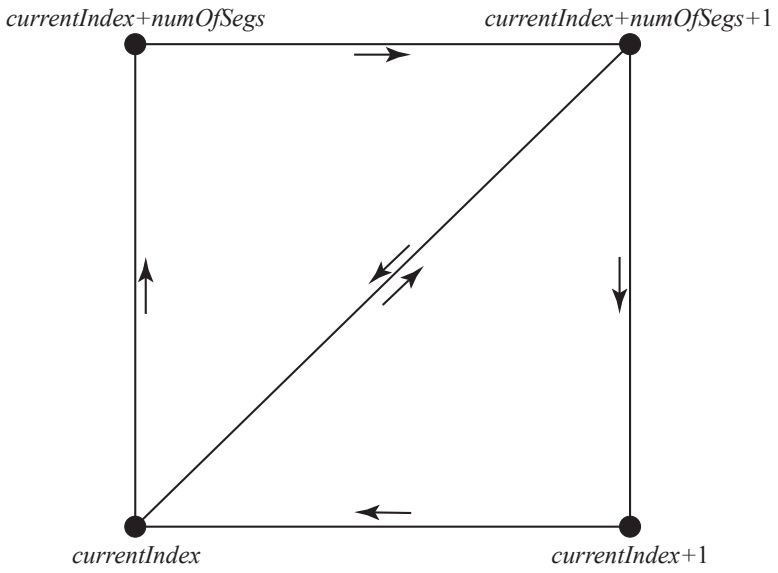


Figure 3.12. Graphic representation of the indexing scheme described by algorithm 3.1

rocks lie and then normalizing the coordinates of projected point to [0,1].
In formulae:

$$\mathbf{TC}(\varphi, \vartheta) = \left(\frac{x(\varphi, \vartheta)}{w}, \frac{z(\varphi, \vartheta)}{h} \right) \tag{3.11}$$

Exact normals for the reconstructed small rock can be computed from parametrization (3.5). The non normalized form of the normals obtained from parametrization (3.5) is:

$$\mathbf{N}(\varphi, \theta) = - \begin{pmatrix} h(x(\varphi, \theta) - \frac{r_\varphi}{r} z(\varphi, \theta)) \\ \frac{r_\varphi}{r} y(\varphi, \theta) \\ h(z(\varphi, \theta) + \frac{r_\varphi}{r} x(\varphi, \theta)) \end{pmatrix} \tag{3.12}$$

Although accurate, the normals computed with equation (3.12) are computationally expensive. An approximation, which in practice gives good looking results is to use as a normal vector the position vector in local space of the small rock translated on the origin of the local reference system.

3.3. Size based level of details

Small rocks meshes generated with the method described in this chapter have all the same number of polygons independently of whether they are large or very small. This is not efficient as very small small rocks meshes end up having many triangles that are in size less than a pixel. In order to correct this problem equation (3.13) is used to determine the number of segments a small rock should have based on its size and on the size of the largest small rock in the batch being currently reconstructed.

$$N_{Si}(x, p) = \lfloor (x - 3)\, p \rfloor + 3 \tag{3.13}$$

where x is the maximum number of segments desired for the largest small rock in the batch and $\lfloor \cdot \rfloor$ is the floor operator. The parameter p will be called the *size ratio*, and it is defined as the ratio between the maximum radii for the small rock and the maximum radii of the largest rock in the batch:

$$p = \frac{\max_{r \in R_i}(r)}{\max_{R_i}(\max_{r \in R_i}(r))} \tag{3.14}$$

Equation (3.13) makes sure that at least three segments are used for a small rock mesh, ensuring that no degenerate small rocks, i.e. small rocks with non zero radii but with zero segments, are generated in the reduction process. Figure 3.13 shows results obtained using equation (3.13) on the preceding page. The following formula

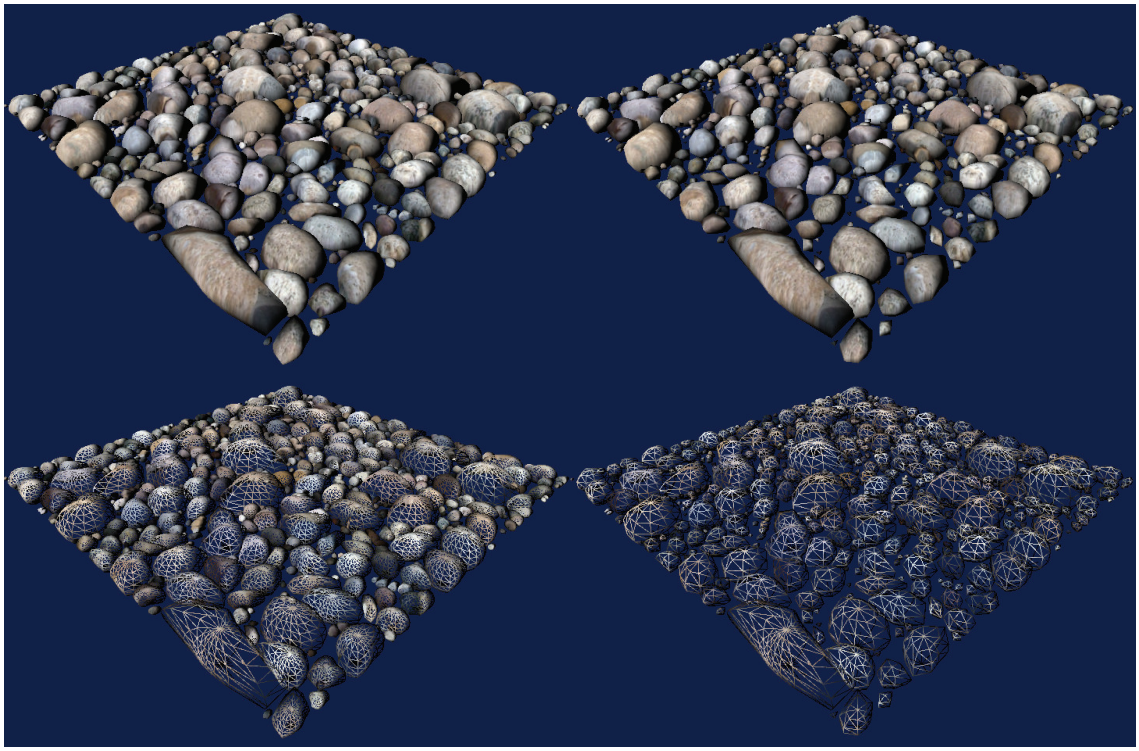


Figure 3.13. Rock models without (left) and with (right) size based LOD segmentation.

is used to understand how many segments equation (3.13) produces for a fixed value x_0 of x when p changes :

$$S(p) = \frac{N_{Si}(x_0, p)}{x_0} \tag{3.15}$$

equation (3.15) normalizes the number of segments produced against the maximum number of segments desired for the largest small rock, this gives the percentage of segments generated with respect to the desired maximum. Figure 3.14 shows a plot of equation (3.15) for $x_0 = 30$. From figure 3.14 it can be seen that when the rock is very small, i.e. choosing a theoretical radius of the small rock in a neighbour of zero, only 10% of the desired segments are generated by equation (3.13), for a small rock that is in size 50% of the largest rock the number of segments generated is 53.33%.

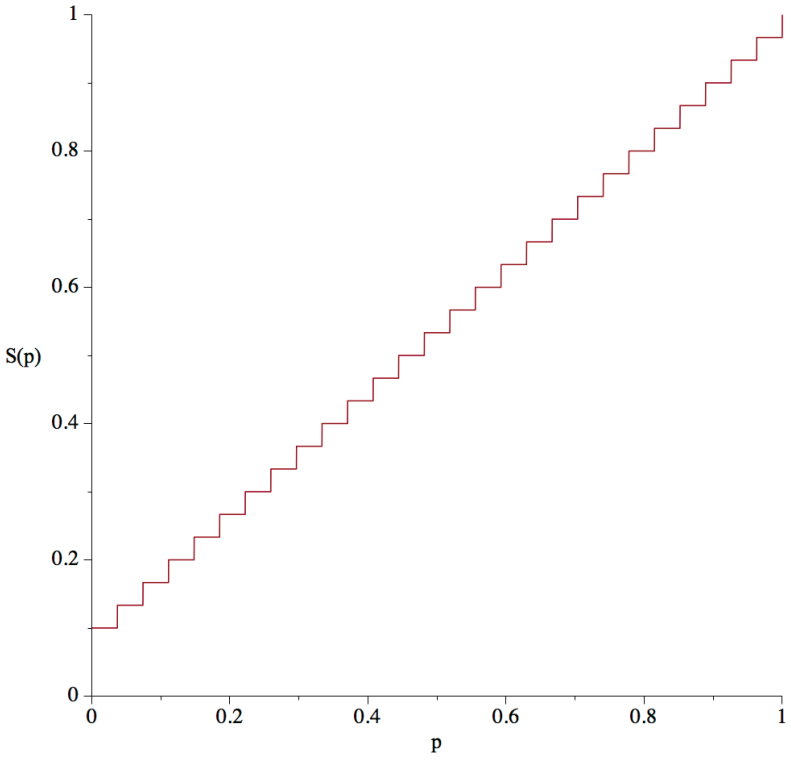


Figure 3.14. Plot of equation (3.15) for $x_0 = 30$

Changing the value x_0 the percent of polygon generated changes accordingly, in general the minimum percent of segments generated will be $\frac{3}{x_0}$ and the maximum 1.0. A drawback of reducing the number of segments using equation (3.13) is that the final reconstructed patch of small rocks loses density as the small rocks occupy less volume after segments' reduction. This loss of volume happens because the reduced small rock is a less accurate approximation of the original small rock. However, this is compensated by a speed up in rendering by a factor of 2.8.

3.4. Software Engineering

Figure 3.15 on the next page shows the class diagram for the small rocks reconstruction software, named *RocksCreator*, that implements the method explained in sections 3.1 and 3.2. RockCreator is subdivided in two modules. The top module in figure 3.15 is named *Rocker* and contains the classes in charge of the user interface, the 3D rendering, the saving of the data and the description of the small rocks data structure. The second module, named *Masker*, is in charge of generating the binary mask from which the data to describe the small rock are extracted and of extracting those data from it. This division effectively decouple the parameter extraction from the mesh representation of the small rock allowing for reuse and easy modification of the software to accommodate different rendering engines.

The Masker module can be reused for purposes different from small rocks mesh generation. Masker is composed by four classes. The *EdgeDetector* class extracts the

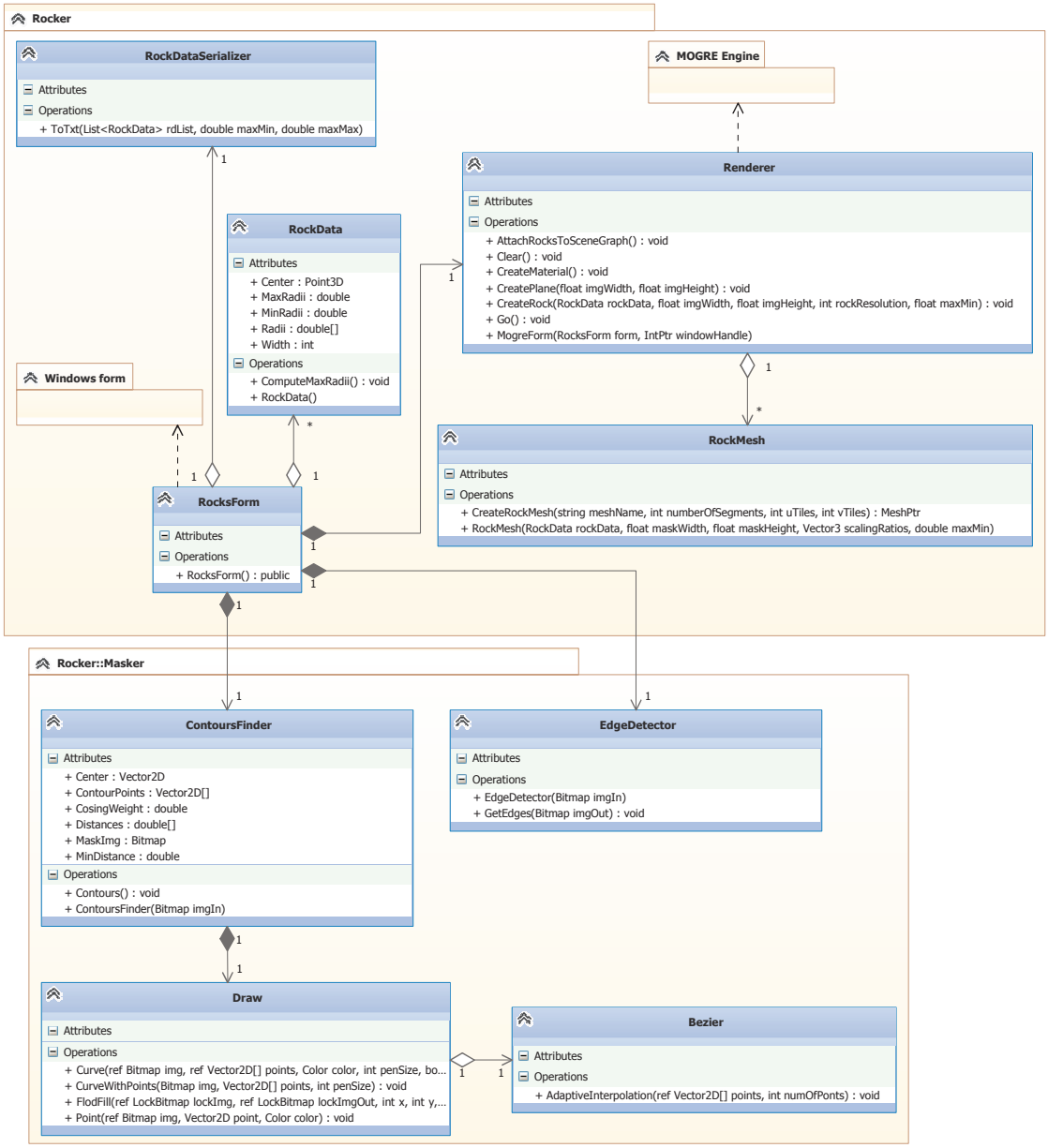


Figure 3.15. UML Diagram fro the small rocks reconstruction software. The top package of the diagram contains the main program, the 3D module and the data structure used to describe a small rock. The bottom package contains the classes used for edge detection and contouring of the small rocks

edges from the image as described in subsection 3.2.1, the *ContourFinder* is used to remove the outliers from the mask obtained from the EdgeDetector and uses the *Draw* class to complete the contours of the small rocks using the data obtained from the *Bezier* class. Separating the Edge detection from the contour finding allows to change the algorithm for edge detection without having to modify the algorithm for the outliers recognition and contours completion and the two classes are independent. The Rocker module integrates the user interface and the rendering engine. The Rocker module is composed of five classes. The *RockForm* class is in charge of starting the rendering engine and manages the data transfer between different

objects in the application. The *Renderer* is in charge of 3D rendering of the meshes, generated by the *RockMesh* class using the data output from the Masker module, which are stored in a list of *RockData* data structures. The list of RockData data structures are saved in a human readable text file to allow reuse of the meshes in other applications.

The software was implemented in C# and uses Windows Forms to generate the user interface and the MOGRE rendering engine [**97**] to render the 3D scene. Figure 3.16 shows two screen-shots of the application, one immediately after the input image has been loaded, the other after the meshes have been reconstructed.

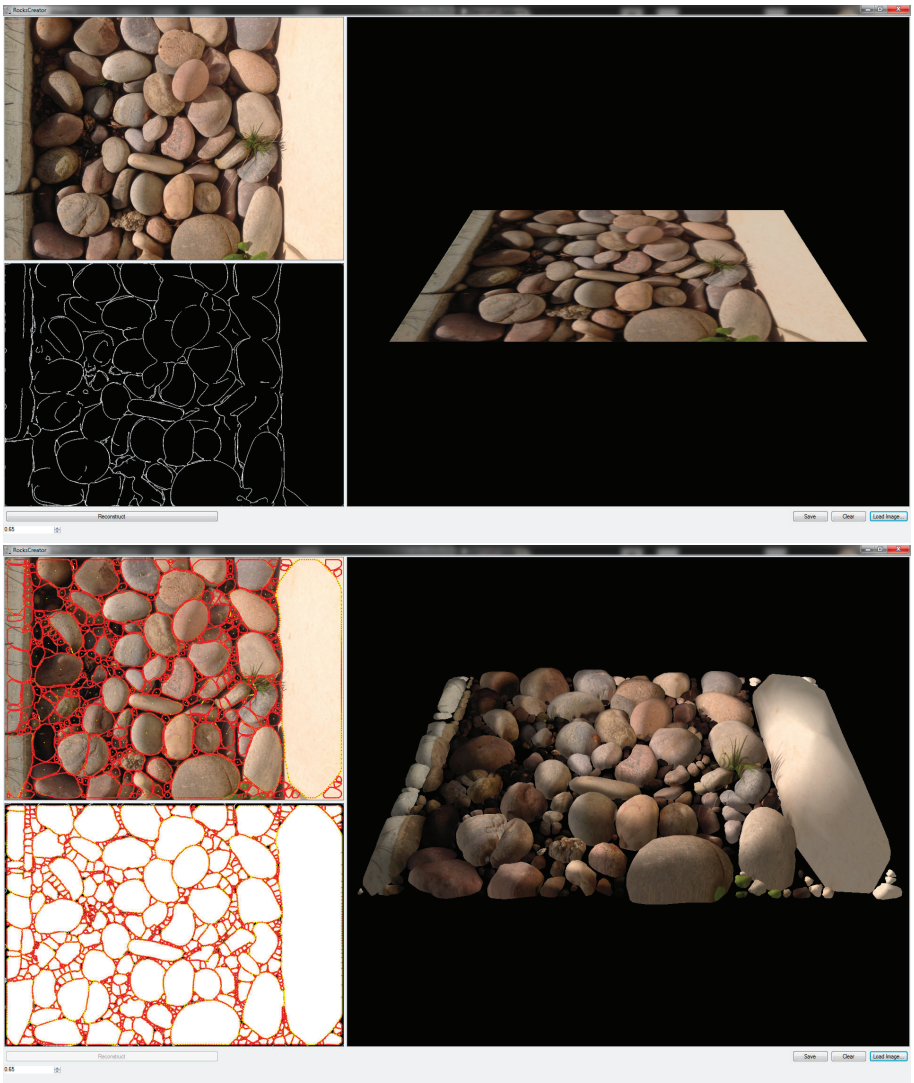


Figure 3.16. Screen-shots of the RocksCreator application. Top: the input image is loaded and binary mask has been extracted. Bottom: the equatorial slices data have been extracted from the binary mask and the meshes reconstructed from them.

3.5. Results

Figures 3.17 and 3.18 on the following page show that the approach described in sections 3.1 and 3.2 is capable of producing realistic small rocks models and distributions that match a reference image. These results have been generated using a 64 bit DELL Precision T1650, with a 4 cores Intel(R) Xeon(R) CPU at 3.4 GHz, 8 GB RAM, and an NVIDIA Quadro 2000 graphics card.

Figure 3.17. Results obtained applying the method described in sections 3.1 and 3.2 to different images of heterogeneous rocks. For larger versions see colour plates C.4 on page 120, C.5 on page 121, C.6 on page 122

The average time for reconstructing a single image is 105 seconds, split as follows: 14.37% of the total time is spent for the edges extraction phase, 85.08% for the equatorial slices reconstruction phase and, 0.56% for the mesh generation phase. The bottleneck is clearly the equatorial slices reconstruction phase, which takes the majority of the time. Parameters described in section 3.2 have been estimated as follows. All test images have been rescaled to have width equal to 900 pixels and height determined by the image aspect ratio. For mask generation, $L = 256$, t_H has

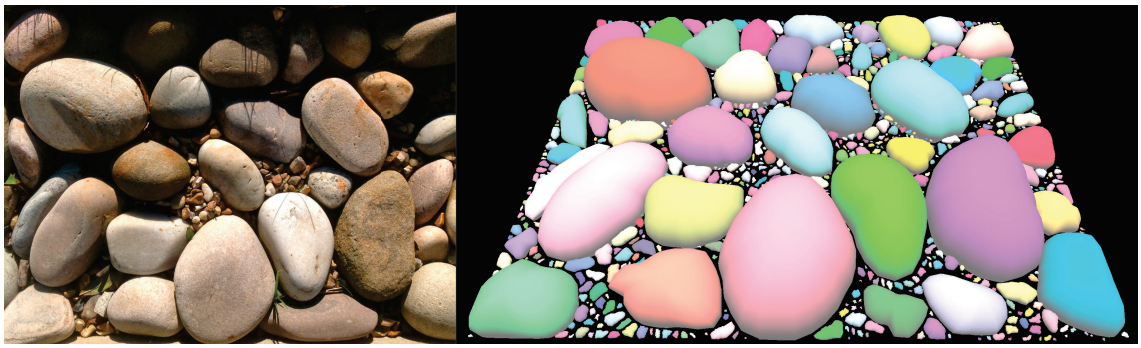


Figure 3.18. Reconstructed small rocks rendered without texturing to show the reconstructed shapes. For larger versions see colour plate C.7 on page 123

been set to the first quartile for the histogram of I_T and then scaled to stay in $[0, 1]$, while, following Canny [**92**], $t_L = \frac{t_H}{3}$. For equatorial slices descriptions the initial sampling grid was set to $\frac{w}{5} \times \frac{h}{5}$, and the number of initial sampling points to 90. Experiments on a set of 80 images shown that $a = 0.65$ and $b = 0.5$ give acceptable results for most of them, and that in general $a \in [0.5, 0.75]$ is a good choice. The angle threshold α_T has been computed iteratively reducing its value from 1 toward zero by 0.01 steps, stopping whenever the minimum number of vertices of the initial polygonal curve P, which was set to 10, is reached. For the images in figure 3.17 the number of vertices interpolated by the Bezièr curve to close gaps has been set to 5.

The method proposed to describes equatorial slices fails to reproduce areas where edges meet with acute angles, resulting in an over-segmentation of the equatorial slice of the small rock near those areas, "crumbling" the mesh as shown by the large white block on the right of the third image in figure 3.17. Although the method is capable of producing realistic models and distributions of small rocks, it relies on a series of parameters that have been estimated heuristically to work on as many test images as possible. A possible approach to automatically estimate those parameters could involve machine learning techniques. Finally, speed and efficiency of the method can be improved, ideally parallelizing it for GPU implementation.

3.6. Patches instancing

Small rocks obtained with the method described in this paper can be used to generate a layer of rocks on a dynamic terrain in the area near the interaction between the terrain and an agent in the virtual environment. However, loading a large amount of meshes by brute force affects the frame rate, because the GPU has to process each mesh singularly to display it. Hardware geometry instancing [**98**] allows this problem to be easily solved. Hardware geometry instancing improves the transfer of data between CPU and GPU by storing one copy of the geometry in the graphics card memory and then sending at each frame only the transformation matrices that move the geometry in space. The geometry is duplicated to match the number of matrices sent on that frame and then transformed by the matrices. Figure 3.19 on the next page shows a grid of 10×10 batches of 492 small rocks (for a total of 49200 rock meshes) reconstructed from a texture applied to a terrain mesh with the maximum segmentation set to 30. Despite the large amount of triangles required to load the meshes the application still run with an average of 50fps (19.5ms per frame). This would not have been possible without using hardware geometry instancing. A drawback of using geometry instancing for the small rocks meshes is that there is a distinctive repeated pattern which detracts from the scene realism. This could be attenuated by rearranging the position of the single small rocks from patch to patch, a possible algorithm for this could be composed of two steps:

1. identify a position using a Poisson-disk sampling algorithm [**99, 100, 101**] where the radii of the dart changes accordingly with the maximum radii of the small rock;
2. move the rocks close to each other to remove possible gaps left by the dart throwing algorithm;

However, this solution has not been implemented in this work.

3.7. Summary

In this chapter a new method for generating small rocks' meshes from a single image has been presented. The method makes use of the parametrization of a generalized ellipsoid to mathematically describe a small rock. This parametrization depends on two parameters, the centre of the equatorial slice of the small rock and the set of radii from the centre. In order to identity these two parameters a method for identify small rocks edges that reduces the influence of the small rocks' texture and save them in a binary mask has been introduced. Although the method for edge detection capture true edges, for the small rocks only it produces incomplete edges. These incomplete edges are processed when the parameters for the generalized ellipsoid parametrization are being extracted from the binary mask. Ray marching and the signed turning angle of a polygon are used to identify outliers to the small rock edge, the outliers are then removes them and complete the edge using a Bèzier

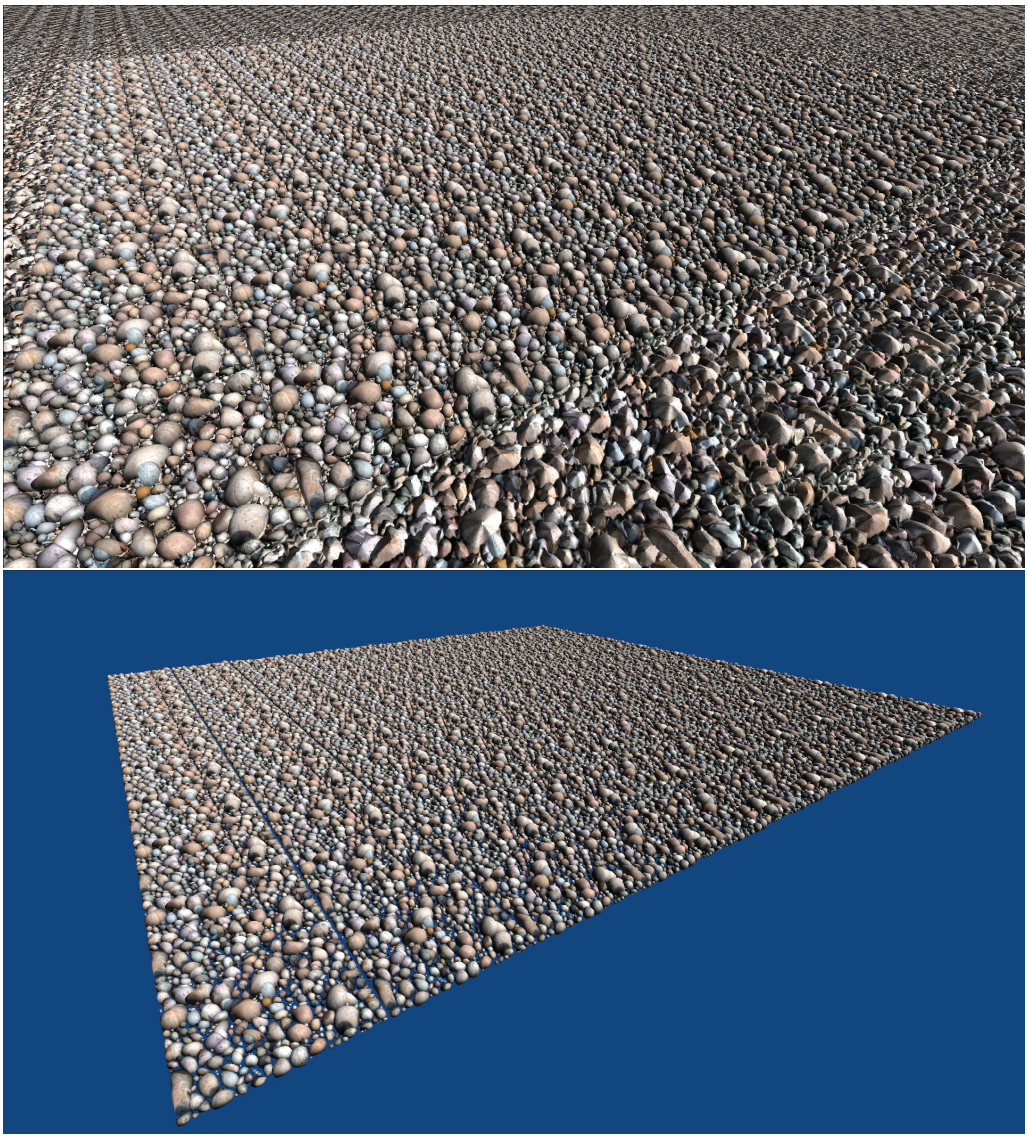


Figure 3.19. A grid of 10×10 patches of reconstructed small rocks loaded in the scene using geometry instancing rendered with (top) and without (bottom) a ground plane.

curve. The completed small rock contour is then used to approximate the centre and the set of radii of the equatorial slice. The parameters thus identified are then used to generate a set of vertices using the parametrization of a generalized ellipsoid. A set of indices, texture coordinates and vertex normals are also produced. Different level of details for the small rock are generated depending on the small rock size. Finally, geometry instancing is used to generate multiple copies of the small rocks efficiently. The method described in this chapter fits in the pre-processing stage of the framework discussed in chapter 1, and it is discussed in [**32**]. In the framework the set of centres and radii are used as inputs to the physics engine for the simulation process. As pointed out in section 2.4 and in section 1.2 it is not possible to reproduce each single rock on the terrain. Moreover, the simulated terrain may be composed of different kind of soils. The approximation that the framework proposes

requires the use of different representations of the terrain depending on the distance of the user from the area where the interaction happens and the kind of soil to simulate. The full reconstruction of rocks can be applied for a small area of gravel soil around the user avatar in an interactive application, for areas away from the agent in the virtual environment, or areas not composed by gravel soil, the simulation has to change from particles based to continuum based.

In chapter 4 a mathematical model based on diffusion theory for deformation of continuum based terrains representations is presented. The model can be used in areas of the terrain away from the user or areas where the terrain is not composed by gravel soil to simulate terrain deformation in real time.

CHAPTER 4

Multi-material terrain deformation

Tracks on a natural terrain are a recording of the terrain interaction history, they tell the story of what happened on the area and what has passed through it. To generate realistic and rich virtual worlds this recording aspect of natural terrains has to be taken into account and reproduced on the virtual environment terrain. To achieve this the composition of the terrain has to be taken into account as terrains are often composed of different materials that behave differently when interacting with other agents of the environment. In chapter 3 a method to reconstruct small rocks from a single image has been described. Results from the method presented in chapter 3 can be used to simulate small rocks for gravel soils terrain in a neighbour of the user avatar. However, for terrains that are not gravel soil based and for areas of gravel soil based terrains far from the user the use of a particle based representation is not efficient. For these a continuum based representation of the terrain is more suitable.

This chapter presents a novel real-time 2D method for simulations of continuum dynamic terrains capable of representing different kinds of soils in real time. The method is based on a modified form of the drift-diffusion equation [**102**] that is used to update a Dynamically-Displaced Height-map DDH. In turn the DDH is used both to update the height map of the terrain and for shading purposes, as explained in section 4.4. The method is implemented on GPU though the use of a compute shader. Although the method is not physically accurate results show that tracks generated on virtual terrains with the method here proposed compare well with tracks left by objects on real natural terrains. Moreover, the mathematical model on which the method is based unifies in a single model for behaviours of multiple materials.
This chapter presents the *Terrain Mesh Deformation and Dynamic Normals* and the *Regions Classification* components of the framework introduced in chapter 1, the components are highlighted in red in figure 4.1 on the following page.

4.1. Terrain deformation based on diffusion theory

When applying a force on an area of a terrain the grains that constitute the terrain in that area are compressed together. This compression phenomena continues as long as there is space for the grains to move close to each other. As soon as the grains run out of space they start to push each other. This pushing makes them

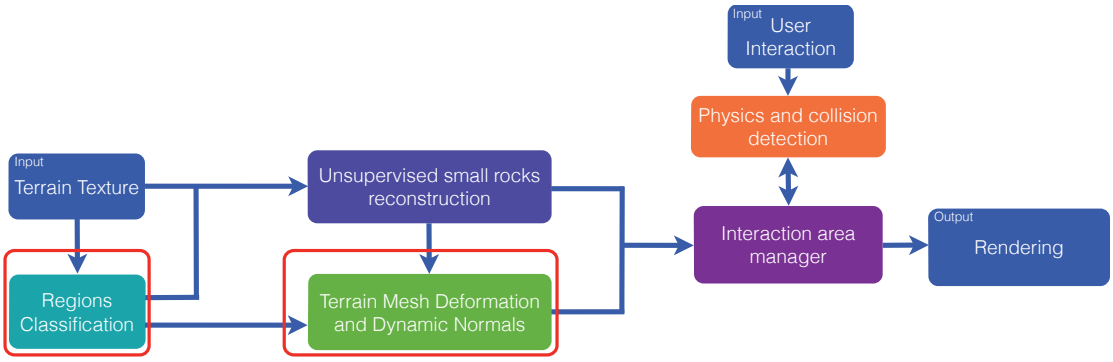


Figure 4.1. Overview of the system. The Terrain Mesh Deformation and Dynamic Normals component is highlighted in red.

escape the action of the force by moving them to areas of the terrain where they are less concentrated. This observation makes terrain deformation akin to, although not exactly the same as, a diffusion process where particles move over a period of time from areas of high concentration to areas of lower the concentration. This similarity forms the foundation of the method described in this chapter.
While it is easy to simulate compaction of the terrain by displacing the shape of the object on the terrain, and this technique is at the basis of Aquilio [**31**] and Schäfer et al. [**39**] methods, simulation of soil diffusion on local deformations is often overlooked in recent literature. Hnaidi et al. [**103**] use diffusion theory for modelling a terrain on a large scale. However, at present there is no model that allows us to simulate small scale local terrain deformations based on diffusion theory. This new model will be developed in the remainder of this chapter.

4.1.1. Mathematical model. The mathematical model for local terrain deformation is inspired by the drift-diffusion equation, also known as the Smoluchowsky equation [**102, 104**], applied to the scalar height field $h(x, z, t)$, see section A.2 of appendix A. In the following formulae the explicit dependency on space and time will be dropped to simplify the notation.

Under the assumption that the diffusion coefficient D is constant, the Smoluchowsky equation is [**102**]:

$$\frac{\partial h}{\partial t} = D\Delta h + \nabla \cdot \left(\bar{F}\zeta^{-1}h\right) \tag{4.1}$$

where Δh is the Laplacian operator applied to the scalar field h, ζ^{-1} is the inverse of the friction constant of the diffusion medium, also known as the *mobility*, $\nabla\cdot$ is the divergence operator, and $\bar{F}$ is an external force field generating an average velocity in the medium. However, in this section $\bar{F}$ will be interpreted as the external force field exerted by an object interacting with the terrain.

In the following equation (4.1) is modified removing from the force term the dependency from h, thus obtaining:

$$\frac{\partial h}{\partial t} = D\Delta h + \nabla \cdot \left(\bar{F}\zeta^{-1}\right) \tag{4.2}$$

The justification for this modification is that the action of the force over the terrain does not depend on the height of the terrain, but only on the mobility and the force itself.

Every granular material when piled up rests with a specific angle α with respect to the ground. This angle is called the *angle of repose* and it is a property of the material, see figure 4.2. However, equation (4.2) does not take the angle of repose of the material into account and diffuses piles of material until the ground is levelled. To include the angle of repose in equation (4.2) the slope of the pile of material at

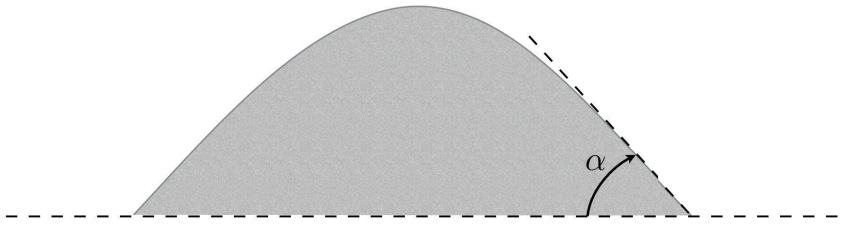


Figure 4.2. Angle of repose α of a pile of granular material

time t is split into two angles, one measured on the xy plane the other measured on the zy plane, where y is the up direction.

$$\alpha_k = \arctan\left(\sqrt{|d_k|}\right) \tag{4.3}$$

where $k \in \{x, z\}$ and the d_ks will be obtained in the following. The values obtained from equation (4.3) are mapped in $[0, 1]$ with the following mapping:

$$w_k = \frac{\alpha_k - \alpha}{\frac{\pi}{2} - \alpha} \tag{4.4}$$

where α is the angle of repose characteristic of the material and $k \in \{x, z\}$. Observing that for a twice differentiable scalar field f and a differentiable vector field $\bar{G}$:

$$\begin{aligned} \Delta f + \nabla \cdot \bar{G} &= (\partial_{xx} f + \partial_x G_x) + (\partial_{zz} f + \partial_z G_z) \\ &= \begin{pmatrix} 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} \partial_{xx} f + \partial_x G_x \\ \partial_{zz} f + \partial_z G_z \end{pmatrix} \end{aligned} \tag{4.5}$$

equation (4.2) can be re-written as:

$$\frac{\partial h}{\partial t} = \begin{pmatrix} w_x \\ w_z \end{pmatrix} \cdot \begin{pmatrix} D\partial_{xx} h + \partial_x\left(\bar{F}_x \zeta^{-1}\right) \\ D\partial_{zz} h + \partial_z\left(\bar{F}_z \zeta^{-1}\right) \end{pmatrix} = \bar{w} \cdot \begin{pmatrix} d_x \\ d_z \end{pmatrix} = \bar{w} \cdot \bar{d} \tag{4.6}$$

where $\bar{w}$ is the vector that has as component the angle's weights obtained in equation (4.4). The Dirichlet boundary conditions for equation (4.6) for $\bar{\mathbf{x}} = (x, y) \in [a, b] \times [c, d]$ with $a, b, c, d \in \mathbb{R}$ and $t > 0$ are:

$$\begin{aligned} h(x, c, t) = 0, \quad h(x, d, t) = 0, \quad x \in [a, b] \\ h(a, y, t) = 0, \quad h(b, y, t) = 0, \quad y \in [c, d] \end{aligned} \tag{4.7}$$

and $\bar{\mathbf{F}}(x, y) = 0$ for $(x, y) \notin [a, b] \times [c, d]$ while initial condition is:

$$h(x, y, 0) = 0 \tag{4.8}$$

where $(x, y) \in [a, b] \times [c, d]$.

4.1.2. Discretization. Due to their simplicity of implementation explicit finite differences methods have been chosen to obtain a discrete form of equation (4.6) on the preceding page, see section A.3 of appendix A. The discrete form is obtained using a square $n \times n$ grid with cells of side Δx equal to one. The scheme used is forward in time, centred in space for the first and second order terms on a von Neumann (4-cells) neighbourhood, see figure 4.3. Using the aforementioned scheme

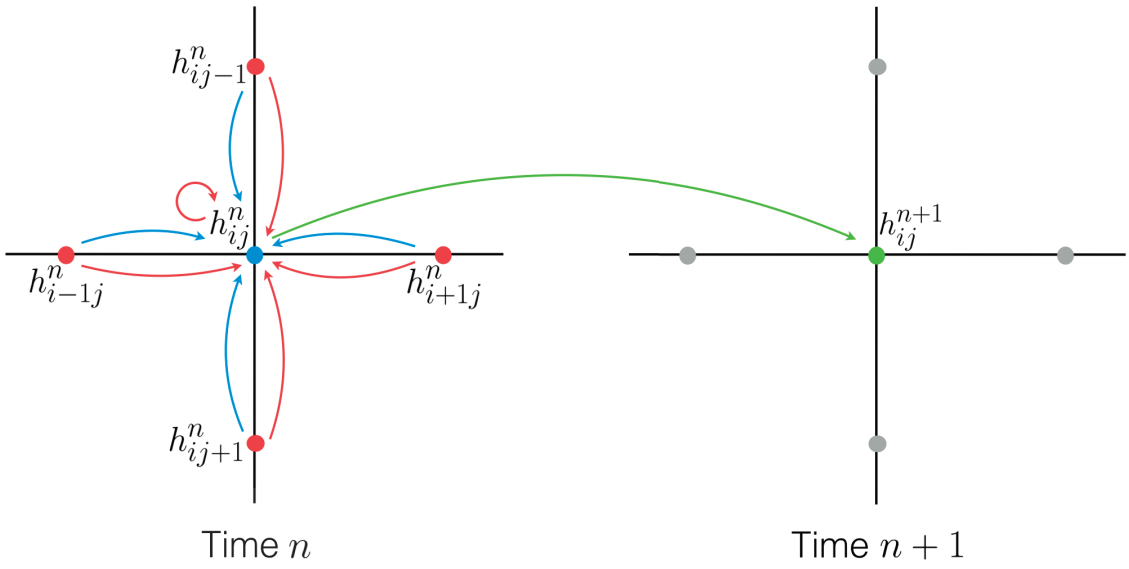


Figure 4.3. Graphical representation of the numeric scheme used to discretise equation (4.6). Cyan lines represent the discretisation of the first order derivatives, red arrows are the discretisation of second order derivatives, the green arrow is the time derivative

and under the assumption that the terrain mobility is constant for terrains composed of a single material the first component of the vector $\bar{d}$ in equation (4.6) becomes:

$$(d_x)^n_{ij} = D(h^n_{i-1j} + h^n_{i+1j} - 2h^n_{ij}) + 0.5\zeta^{-1}((F_x)^n_{i+1j} - (F_x)^n_{i-1j}) \tag{4.9}$$

similarly for the second component.
The discrete form of equation (4.6) is then:

$$h^{n+1}_{ij} = h^n_{ij} + \Delta t \left(w^n_{ij} \cdot d^n_{ij}\right) \tag{4.10}$$

where w_{ij}^n is the discrete form of the weight vector in equation (4.4) obtained from:

$$(\alpha_k)_{ij}^n = \arctan\left(\sqrt{|(d_k)_{ij}^n|}\right) \tag{4.11}$$

where $k \in \{x, z\}$.

4.2. Handling multi-material terrains

Discretization given in equation (4.9) on the previous page is obtained assuming that the diffuse constant D and the mobility coefficient ζ^{-1} are constant across the whole terrain. For multi-material terrains this is not true as each material will have different values for the diffusion constant and mobility coefficient, transforming them in two functions of the position $D(x, z)$ and $\zeta^{-1}(x, z)$. Equation (4.6) on page 68 has then to be modified taking the diffusion function $D(x, z)$ inside the derivative symbol, applying the chain rule one obtains:

$$\frac{\partial h}{\partial t} = \bar{w} \cdot \begin{pmatrix} D\partial_{xx}h + \partial_x h \partial_x D + \zeta^{-1}\partial_x \bar{F}_x + \bar{F}_x \partial_x \zeta^{-1} \\ D\partial_{zz}h + \partial_z h \partial_z D + \zeta^{-1}\partial_z \bar{F}_z + \bar{F}_z \partial_z \zeta^{-1} \end{pmatrix} = \bar{w} \cdot \bar{d} \tag{4.12}$$

where explicit spatial dependencies have been dropped to simplify the notation.

A discretization with finite differences for the first component of d returns:

$$\begin{aligned} (d_x)_{ij}^n &= D_{ij}^n (h_{i-1j}^n + h_{i+1j}^n - 2h_{ij}^n) \\ &+ (h_{i+1j}^n - h_{ij}^n)(D_{i+1j}^n - D_{ij}^n) \\ &+ 0.5\ (\zeta^{-1})_{ij}^n \left((F_x)_{i+1j}^n - (F_x)_{i-1j}^n\right) \\ &+ 0.5\ (F_x)_{ij}^n \left((\zeta^{-1})_{i+1j}^n - (\zeta^{-1})_{i-1j}^n\right) \end{aligned} \tag{4.13}$$

A similar discretization can be obtained for the second component of d.
Although this formulation is mathematically correct, deformations obtained using equation (4.13) do not differ significantly under the visual point of view compared with deformations obtained using equation (4.9) during the simulation. For this reason in the following multi-materials terrains are treated mathematically in the same way as single material terrains.

4.3. Shaders Pipeline

Terrain mesh data were transferred from the application to the Input Assembler in the form of a vertex buffer and an index buffer with topology set to *6 control point patch-list*. As the terrain is modelled as an isometric grid this setting for the mesh topology avoids geometry cracking when tessellating the main triangular patch, formed by three control points, as it takes into account its three neighbouring triangles, each described by adding an additional control point to the two control points describing the edge of the main patch, see section 4.4.1.1 for more details.
Figure 4.4 shows the pipeline for the terrain generation, see appendix B for more

details on the rendering and compute pipelines.
The programmable shaders used for the terrain mesh are the following:

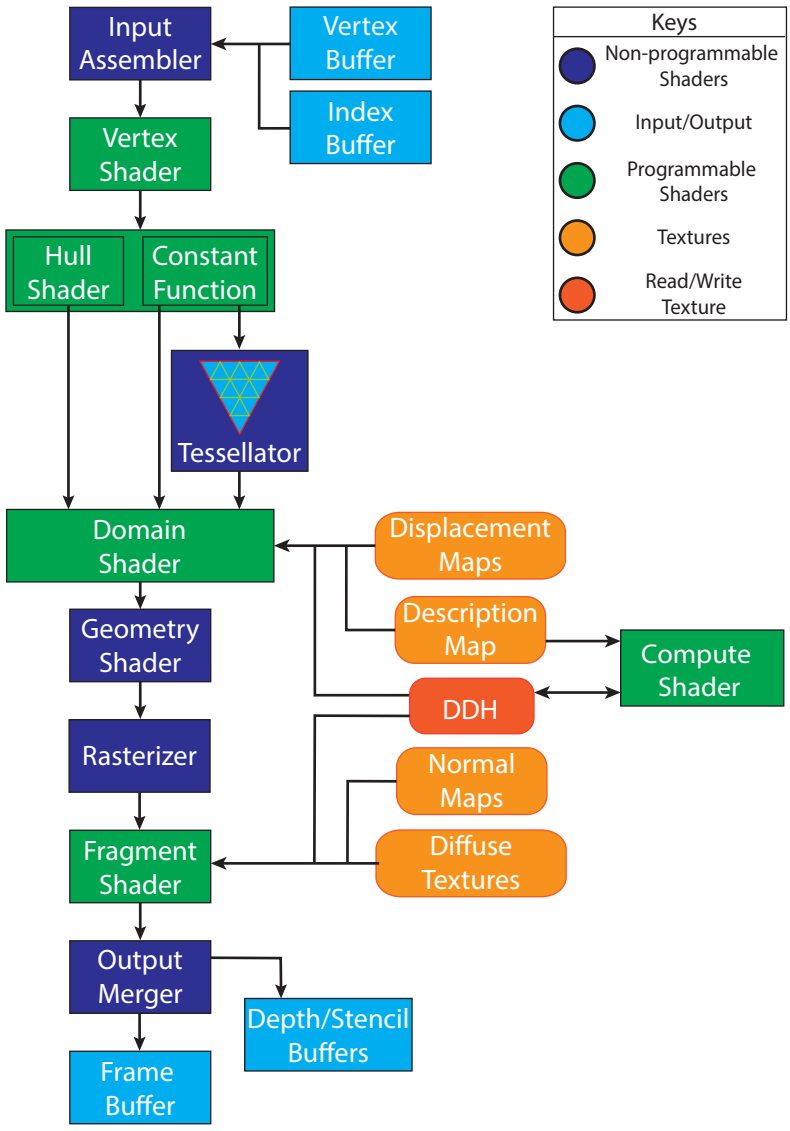


Figure 4.4. Rendering pipeline for the terrain mesh, its resources and its interaction with the compute shader through a texture used as DDH.

- Vertex Shader: receives the vertex data from the input assembler and just passes them through to the hull shader;
- Constant function: Determines the amount of tessellation for both the patch and its edges. The tessellation factor is computed on run time and decrease exponentially with the distance from the camera, thus ensuring that the maximum level of detail for the mesh is the closest to the point of view;
- Hull Shader: set the tessellator stage as a triangular domain with fractional partitioning to return counter-clockwise oriented triangles with three

control points. It also passes through the data regarding texture coordinates, the area of tessellation and the maximum height of the terrain to the domain shader;

- Domain Shader: interpolates vertices attributes using the baricentric coordinates received from the tessellator. It also interpolates the displacement maps accordingly with the ID information received from the description map for the terrain. Finally, this stage reads the DDH and displaces the terrain mesh.
- Fragment Shader: colours the fragments accordingly with the material id passed through by the domain shader. It also computes the dynamic normals for the DDH and blends them with the normal maps for the soil materials. The final colour is computed using Phong shading [**105**].
- Compute shader: implements the discretization of equation (4.6) on page 68 and updates the DDH with the numeric solutions obtained from the discretization.

More details about the implementation are given in section 4.4.

4.4. Implementation

The applicaiton was implemented in C++ using DirectX11 [**106**] while shaders were implemented in High Level Shading Language (HLSL). The numerical scheme in section 4.1.2 was implemented on GPU using a DirectX11 compute shader with 16×16 threads per group, which was tested for different grid sizes as explained in section 4.5.

A structured buffer containing objects interacting with the terrain has been passed to the compute shader, listing 4.1 on the next page shows the structure for the object data. To specify the material parameters for the simulation a *description map* is used. A description map consists of a two dimensional texture storing the material parameters identified in section 4.1.1, description maps are detailed in section 4.6. To speed up the data access during computation a padded buffer of 18×18 Cell elements in shared memory was used in addition to the buffer in the device global memory. The shared memory buffer was used to temporary store the data used by the thread group during the evaluation of equation (4.10) on page 69.

The Cell structure contains the height of the terrain, the fraction of the force distribution exerted on the area of the terrain covered by the Cell and all the parameters necessary for the simulation. The force distribution was obtained by spreading the force exerted by the object on the terrain over the area of interaction, the height was read from from the terrain height-map and summed with the value read from DDH, and the other parameters are read from the description map. The use of a shader buffer reduces the texture data fetching operations for each group of threads because the data needed for the computation of equation (4.10) by a group of threads resides

```
struct Object{
 float3 position;
 float radii;
 float3 force;
 float mass;
};

struct Cell
{
  float height;
  float2 force;
  float reposeAngle;
  float mobility;
  float diffusion;
};

RWTexture2D<float> ddh;                 // Holds the DDH map
StructuredBuffer<Object> Objects;       // Holds the objects data

groupshared Cell sharedData[PADDING_X * PADDING_Y];

Texture2D<float3> desc:register(t1);
```

Listing 4.1. Data strucutres and resources used in the implementation of equation (4.10) in a compute shader.

now on local shared memory, which is a faster access memory than texture memory.

After data fetching from global and texture memory to shared memory, equation (4.10) was evaluated and the results stored in the DDH as a 2D texture as shown in figure 4.5. Listing 4.2 on the next page shows the core loop of the compute shader where equation (4.6) is computed.

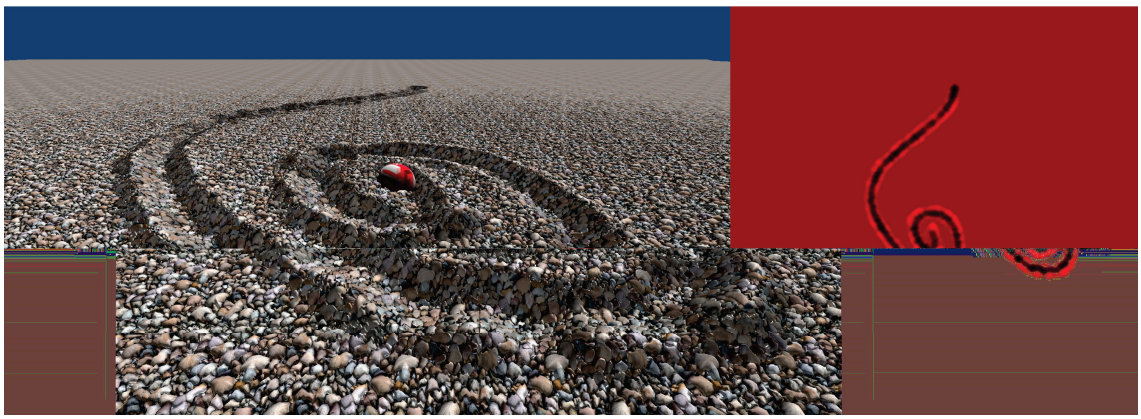


Figure 4.5. Left: Terrain interactively deformed by a user dragging an object on it. Right: DDH updated by the the solution to the modified drift diffusion equation.

To speed up the movement of material during the simulation the computation of equation (4.10) was iterated a number of times. Experiments show that three to five iterations produce realistic movements of material on the terrain. Materials simulated with less than three iterations move slowly giving the impression of being highly viscous. Although materials simulated using more than five iterations look

```
[unroll] for (int i = 0; i < numIterations; i++)
{
  //Computes the Laplacian of the height
  heightLaplacian = sharedData[gIndex].diffusion * float2(
      sharedData[gIndexXm1].height + sharedData[gIndexXp1].height
                                                    - 2 * heightUpdate,
      sharedData[gIndexYm1].height + sharedData[gIndexYp1].height
                                                    - 2 * heightUpdate
    );
  //Computes the divergence of the force
  forceDivergence = -sharedData[gIndex].mobility * float2(
      sharedData[gIndexXp1].force.x - sharedData[gIndexXm1].force.x,
      sharedData[gIndexYp1].force.y - sharedData[gIndexYm1].force.y
    );

  sum = heightLaplacian + forceDivergence;

  angles  = atan(sqrt(steps * steps * abs(sum)));
  weights = anglesWeights(angles, reposeAngle, PI_2);

  heightUpdate += timestep * dot(sum, weights);

  sharedData[gIndex].height = heightUpdate;
}
ddh[dTId.xy] = sharedData[gIndex].height; //DDH Update
```

Listing 4.2. Implementation of discretization (4.6)

less viscous they are less stable, this can be fixed by tweaking the value of the mobility constant. However, too many iterations affect the frame rate of the simulation making the application run slower.

The description map used to identify the materials parameters in the compute shader was used by the domain and pixel shader for displacement and texturing purposes. Moreover, the DDH obtained from the compute shader was used in the domain shader to displace the vertices of the mesh, and in the fragment shader to compute the normals to the surface dynamically.

The dynamic computation of normals from the DDH allows the use of dynamic levels of details for the terrain mesh as the resulting shading approximate the mesh deformation in a visually convincing way when the deformed area is far from the camera, see figure 4.6 on the following page. The dynamic normals are blended linearly with the normals read from the normal maps thus preserving small details when the mesh is deformed.

4.4.1. Terrain mesh generation. In order to implement the terrain mesh different kind of topologies have been considered. Terrains are classically represented using either diamond patterns [**107, 108**] or regular square grids [**1**]. Although both these patterns are designed to easily generate multiple levels of details for the terrain without cracks they introduce an error when the terrain is deformed [**109**]. By contrast *isometric grids* [**109**] have the advantage of reducing the deformation

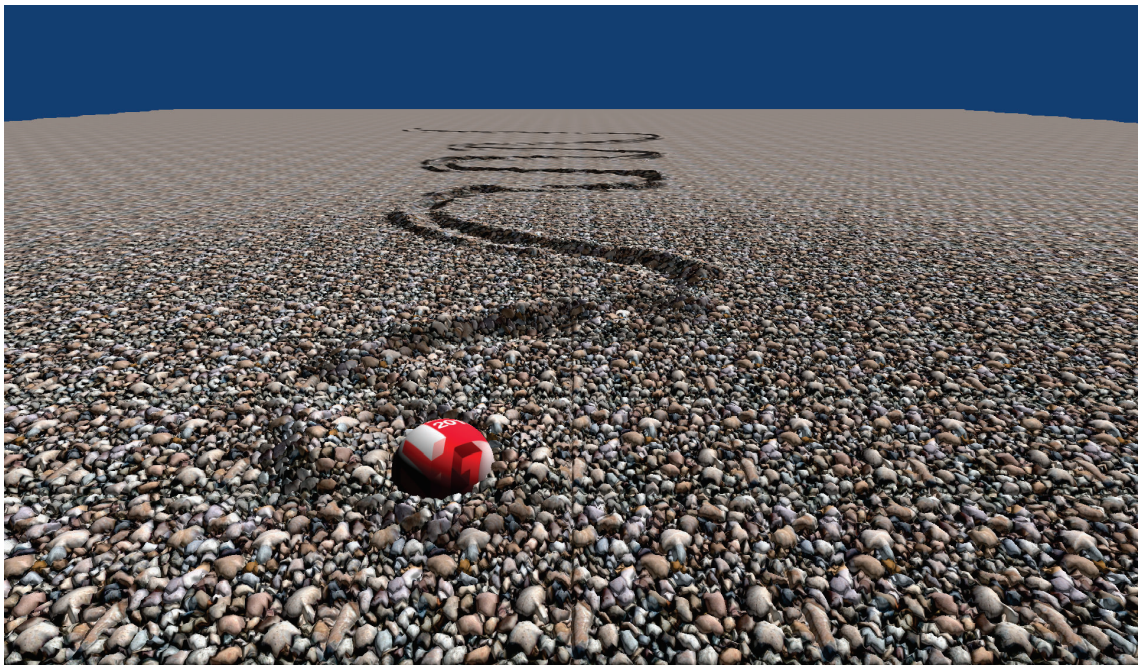
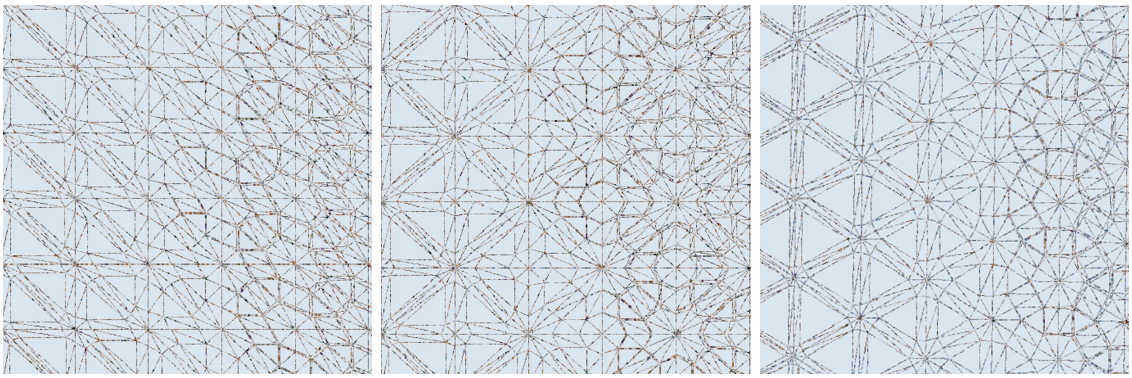


Figure 4.6. Deformation level of details on a terrain. Far deformation simulated by dynamic normals, near terrain mesh displaced using DDH.

error in comparison with the other methods. Moreover, isometric grids have the advantage of producing isotropic tessellations as shown in figure 4.7. While square grids and diamonds respectively produces ellipsoidal and complex patterns isometric grids form almost perfect circles around each vertex meaning that the tessellation will be the same on each direction per each vertex. For this reasons isometric grids have been used for during implementation.



(a) **(b)** **(c)**

Figure 4.7. Local tessellation for a square grid (a), diamond grid (b) and isometric grid (c). Notice that the square grid and the diamond grid form anisotropic patterns, while the isometric grid forms almost perfect circles.

4.4.1.1. *Tessellation.* To deform the terrain mesh a tessellation shader was used to tessellate a neighbourhood of the point where the camera is looking at. In order to tessellate the terrain mesh a patch constant function must be defined. This function determines the level of tessellation for each patch and for the edges of the patch. To

avoid cracking the level of tessellation of an edge shared by two adjacent patches must be set to be the same for each patch. This can be achieved by using the topology scheme depicted in figure 4.8. The red triangle in figure 4.8 is the current patch being processed while the green are the adjacent patches, bold indices are relative vertex indices while the regular indices are edges indices. Coordinates are relative to a reference system centred on index 1.

Listing 4.3 on the next page shows the constant function for the terrain. In order

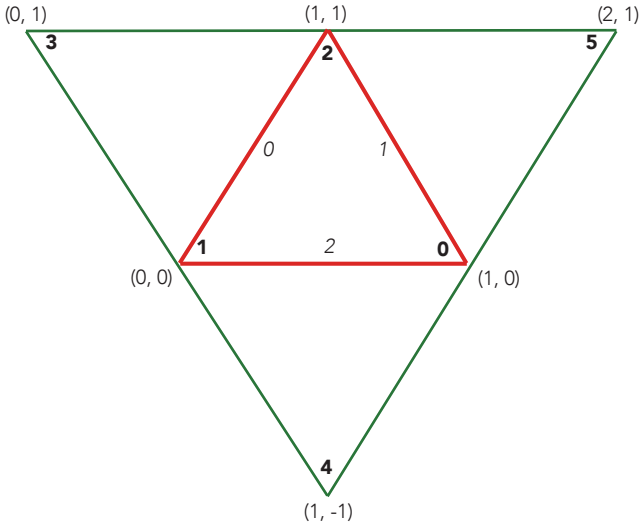


Figure 4.8. Topology scheme specified in the constant function part of the tessellation pipeline

to localize tessellation in a neighbourhood of the camera the distance d between the camera and the midpoint of each triangle in the 6 points patch is computed. The tessellation is then weighted based on the following equation:

$$w_i = e^{-\left(\frac{d}{k}\right)^2} \tag{4.14}$$

where k is the radius of the neighbour that has to be tessellated and $i \in \{0, 1, 2, 3\}$.

The tessellation amount for an edge is weighted using the minimum between the weight w_0 computed for the central patch, in red in figure 4.8, and the weight w_i computed for neighbouring patch sharing the edge under consideration, in green in figure 4.8. Finally, the tessellation amount for the centre patch is weighted using w_0 and adding 1 to make sure that the the tessellation factor does not to zero, which is an inadmissible value. The for loops in listing 4.3 on the next page are unrolled to optimize their execution on GPU.

4.5. Results

Results shown in figure 4.9 on page 78 have been produced using a 64 bit DELL Precision T1650, with a 4 cores Intel(R) Xeon(R) CPU at 3.4 GHz, 8 GB RAM, and an NVIDIA Quadro 2000 graphics card with a rendered frame size of 1280×720.

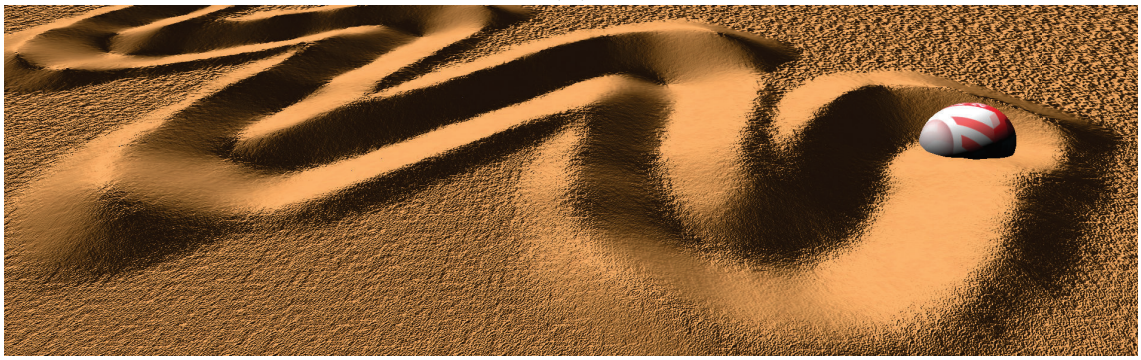
```
struct CFOut
{
  float edges[3]  : SV_TESSFACTOR;
  float inside[1]   : SV_INSIDETESSFACTOR;
  float influenceRadius : TEXCOORD0;
};

// ------------- Helper function -------------
//    Computes the midpoint of a triangle
inline float3 Midpoint(float3 p1, float3 p2, float3 p3)
{
  return (p1 + p2 + p3 ) * 0.3333333333f;
}

// ------------- Patch Constant function -------------
CFOut ConstantFunction(InputPatch<VSOut, 6> inputPatch)
{
  CFOut cOut = (CFOut)0;

  float influence = influenceRadius*edgeLength;

  float3 mid[] =                //Patches midpoints
  {
    Midpoint(inputPatch[0].position.xyz, inputPatch[1].position.xyz,
    inputPatch[2].position.xyz), //patch 012
    Midpoint(inputPatch[1].position.xyz, inputPatch[2].position.xyz,
    inputPatch[3].position.xyz), //patch 123 -> edge 0
    Midpoint(inputPatch[0].position.xyz, inputPatch[2].position.xyz,
    inputPatch[5].position.xyz), //patch 025 -> edge 1
    Midpoint(inputPatch[0].position.xyz, inputPatch[1].position.xyz,
    inputPatch[4].position.xyz)  //patch 014 -> edge 2
  };

  float dist[4];
  float weights[4];
  [unroll] for (int i = 0; i < 4; i++)
  {
    dist[i] = distance(mid[i], objPos);
    weights[i] = exp(-pow(dist[i] / influence, 2));
  }

  [unroll] for (int i = 1; i < 4; i++)
  {
    cOut.edges[i-1] = min(weights[0], weights[i]) * tessAmount + 1;
  }

  cOut.inside[0] = weights[0] * tessAmount + 1;
  cOut.tessWeight = weights[0];
  return cOut;
}
```

Listing 4.3. Implementation of the patch constant function

Simulation and rendering times for different grid sizes and tessellation factors are reported in table 4.1 on page 79. Times do not depend on the kind of soil simulated and represent the combination of simulation and rendering time for a single frame. In the last row the running time of the compute shader has been isolated. Notice that when the grid grows in size the method proposed scales linearly and that even

(a)

(b)



(c)

Figure 4.9. Results obtained with the method presented in this chapter: (a) comparison between tracks on simulated pebbles and on a real pebbles terrain, for a larger version see colour plate C.8 on page 124; (b) comparison between tracks on simulated mud and on a real muddy terrain, for a larger version see colour plate C.9 on page 125; (c) simulation of sand;

with the largest grid and highest tessellation factor experimented with the simulation still runs over 60 fps.

Although grid sizes smaller than 512×512 run with lower computation time they produce non visually appealing deformations as the DDHs thus obtained are too low in resolution. Moreover, values of the tessellation factor lower than eight does not produce any significant time improvement and produce mesh displacements, which are of lower quality.

Pla-Castells et al in [**13**] report that they achieve 0.13 ms per iteration with their method using 60 grid points, with a grid size of 60×60 points the method proposed in this paper achieves 0.15 ms per iteration, which is comparable with the timings

Table 4.1. Average rendering times expressed in milliseconds for different grid sizes and tessellation factors. On the top of the table times refers to the rendering of a single frame comprising tessellation, a single iteration of the compute shader, dynamic normals computation, texturing, lighting and rasterisation. The last row isolates the running time of a single iteration the compute shader.

Grid sizes Tessellation	512×512	1024×1024	2048×2048
8	2.12 *ms*	3.88 *ms*	10.56 *ms*
16	2.32 *ms*	4.08 *ms*	10.78 *ms*
32	2.80 *ms*	4.55 *ms*	11.28 *ms*
64	4.32 *ms*	6.08 *ms*	12.85 *ms*
Compute Shader	0.58 *ms*	2.09 *ms*	7.97 *ms*

obtained by Pla-Castells et al, but uses twice the number of grid nodes, guaranteeing more details. Pla-Castells et al method is limited on the number of computational nodes that they can use for their simulation, maximum 60 grid points, and it is capable to only simulate sand, while the method proposed in this work is capable of simulating multiple materials at the same time with a minimum grid resolution for visually appealing results 8.5 times higher than Pla-Castells et al method, meaning that finer details and smaller objects can be captured during the simulation. Figure 4.9 on the preceding page shows that different kind of terrains can be simulated realistically by changing the parameters ζ^{-1}, D and α in equation (4.6). Moreover, figure 4.10 shows that the method presented is capable of handling multiple materials at the same time. The material distribution is described using a description map, detailed in section 4.6. All figures have been obtained on a grid of size 512×512, for the sand and mud scenes the tessellation factor has been set to 8, while for the pebbles scene the tessellation factor has been set to 64 to better simulate the pebbles using a displacement map.

The displacement map for this scene has been obtained as follows. The small rocks meshes were reconstructed from the texture using the method described in chapter 3, an orthographic camera was then set so that its view direction was parallel to the normal of the resting plane of the small rocks and its distance from the resting plane was set to be equal to the maximum height of the small rocks. A depth buffer was rendered from this camera and saved as a displacement map. Figure 4.11 on page 81 shows a comparison between results obtained using a grey-scale version of the pebbles texture as displacement map and using the depth buffer of the reconstructed pebbles models. Pebbles displaced using the depth map appear more smooth and of different size from one another. Moreover they are better defined as the displacement happens where the pebble actually is instead of when a there is change in intensity on the texture.

Objects can dynamically change sizes during the simulation. As long as the size

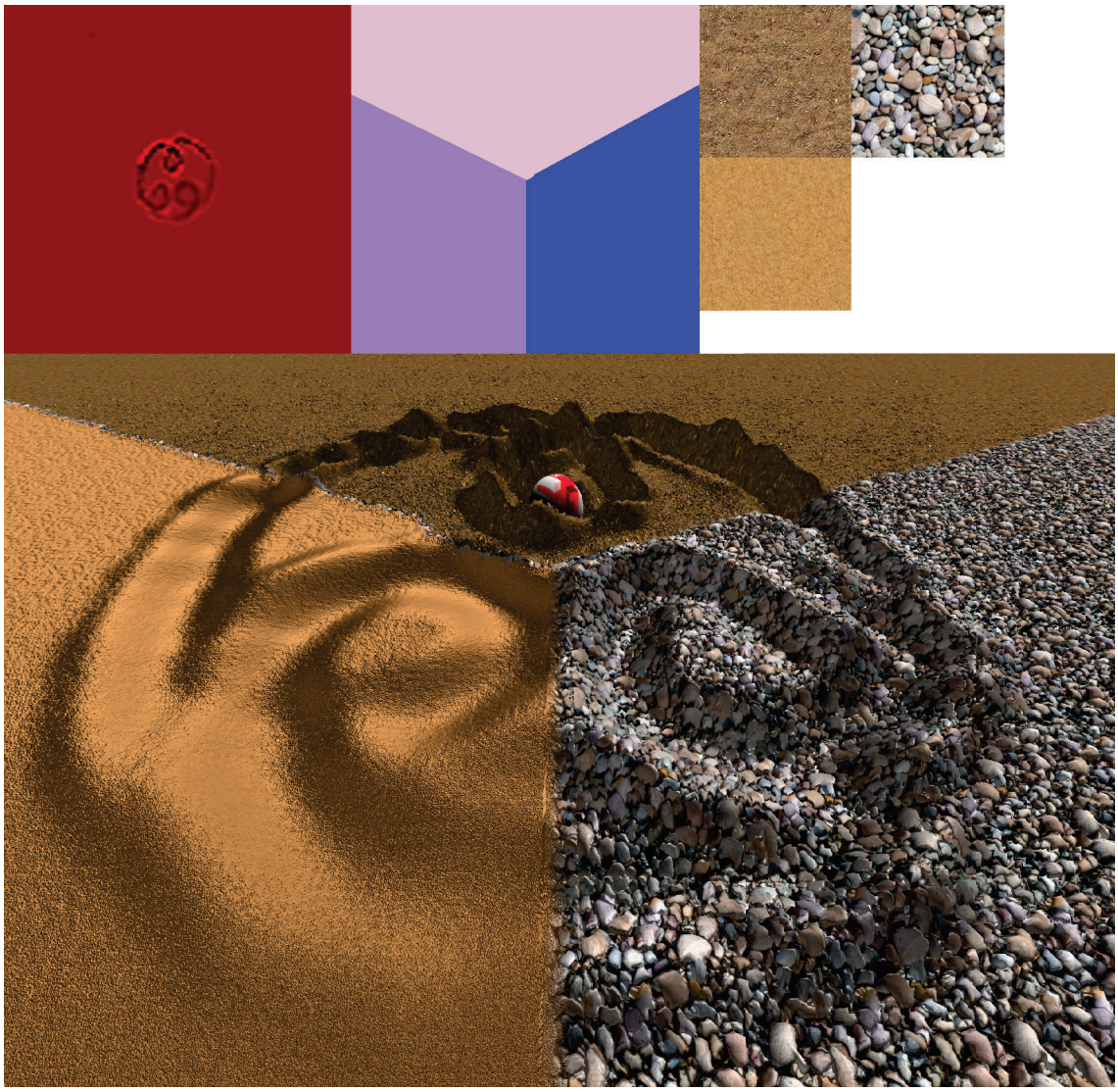
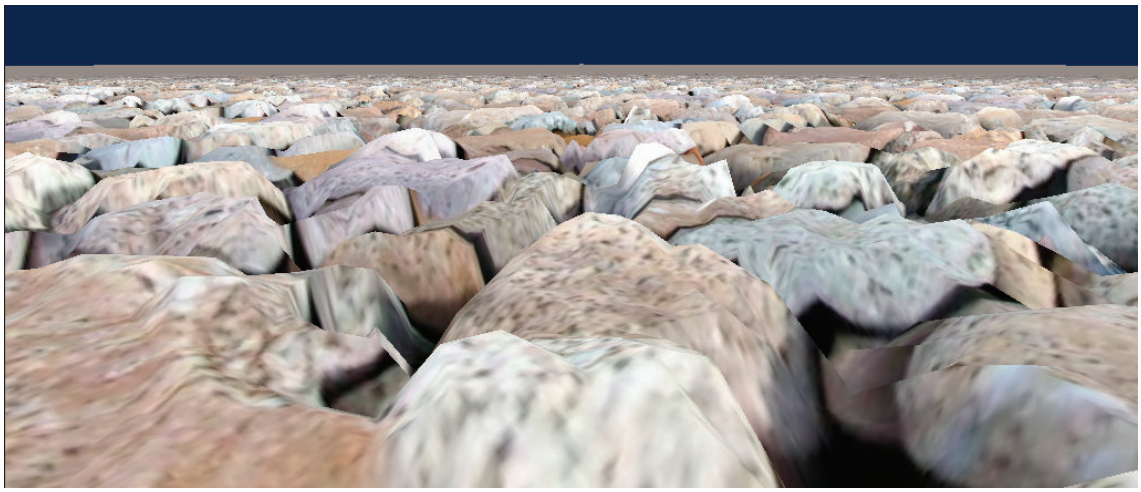


Figure 4.10. Example of multi-material terrain composed by sand pebbles and mud, all three materials are simulated in real time responding differently to the interaction between the object and the terrain.
Top to bottom, left to right: DDH, description map (discussed in section 4.6), diffuse textures [**110, 111, 112**], screen-shot from the simulation.

of the object stays bigger than the size of one cell the method proposed is capable of capturing the object track on the terrain. However, in this work general collision detection between objects and the terrain has not been formally addressed. General collision detection methods, as those proposed by Schäffer [**39**] and Aquilio [**31**], can be easily incorporated in the method by adding a function in the compute shader that computes the collision area and the force distribution on the area, either by using the voxels from Shäffer [**39**] method or the pixels flagged as collisions in the projected DDH used by Aquilio [**31**]. Moreover the method here presented can be incorporated in larger frameworks for terrain level of details as the one proposed by Yusov [**3**].



(a)

(b)

Figure 4.11. (a) detail displacement obtained using grey scale version of the texture; (b) displacement obtained using the depth map from the reconstructed pebbles.

During simulation a loss of volume has been noticed during pile of sand simulations while a slight increase of volume has been noticed during the interaction of objects with the terrain. These changes in volume are due to the fact that no mass conservation constraint have been applied to the simulation. The simulation produces good looking results with just one iteration, but the material looks slightly viscous, increasing the number of iteration reduces this problem. Finally, the interaction of an object with a pile of material produces limited landslides effect, this is due to the modification made in equation (4.6) for taking into account the angles of repose, which is perhaps too restrictive.

4.6. Regions Classification

This section describes the *Regions Classification* component of the framework described in chapter 1. In order to easily edit the materials distribution for the terrain

a description map editor was built, shown in figure 4.12.

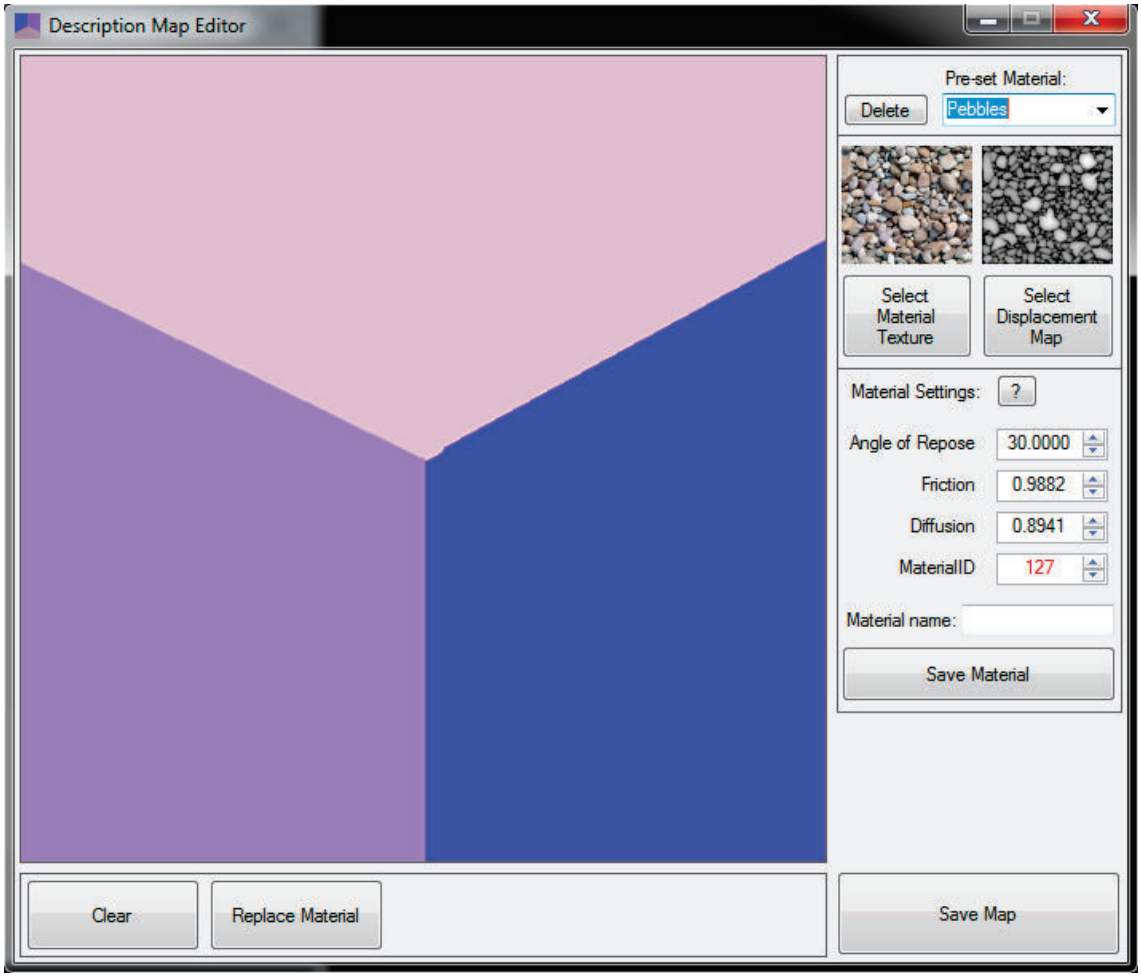


Figure 4.12. Screen-shot of the application developed to draw the description maps used to simulate different materials for the terrain

The editor allows to set the material parameters and link them to a diffuse and a displacement map by generating a description map. A description map is a two dimensional 32-bit image whose colour channels are used to store the angle of repose α (red channel), the mobility constant ζ^{-1} (green channel), the constant of diffusion D (blue channel), and the material ID (alpha channel), which is used to map a material to its displacement and diffuse textures.

The mapping between IDs and textures is saved in a textual file and used by the application at loading time to allocate texture memory on the GPU and generate the shader resource views to bind to the shaders. In order to use those textures the fragment and displacement shaders must have the appropriate number of texture variables set. To ensure that the shaders are set to use the textures the editor automatically generates a fragment shader and domain shader upon saving the description map. These shaders can be either used as they are or further edited to adapt them to the application, in this work they are used as they are. Each shader

contains an array of two dimensional textures, set to receive the number of textures used by the description map, and a function which performs the texture blending. The function is used for colouring the fragment in the fragment shader and to add details to the terrain mesh by using displacement maps in the domain shader.

The blending of the textures is performed by using modified hat functions similar to the following:

$$\psi(a,b,c,t) = \begin{cases} \frac{t-a}{b-a} & if\ a \leq t \leq b \\ \frac{c-t}{c-b} & if\ b < t \leq c \\ 0 & otherwise \end{cases} \tag{4.15}$$

where a, b and c are real numbers between zero and one and represent respectively the minimum, an internal and the maximum point of the interval where the hat function is different from zero, as shown in figure 4.13, while t is the texture ID normalized to lie between zero and one. Function (4.15) is not a proper hat function as it is not symmetric nor its maximum is on zero. Although proper hat functions could be used for the blending, the formulation given in here is more flexible as it allow to asymmetrically control how fast each texture blend with the other.

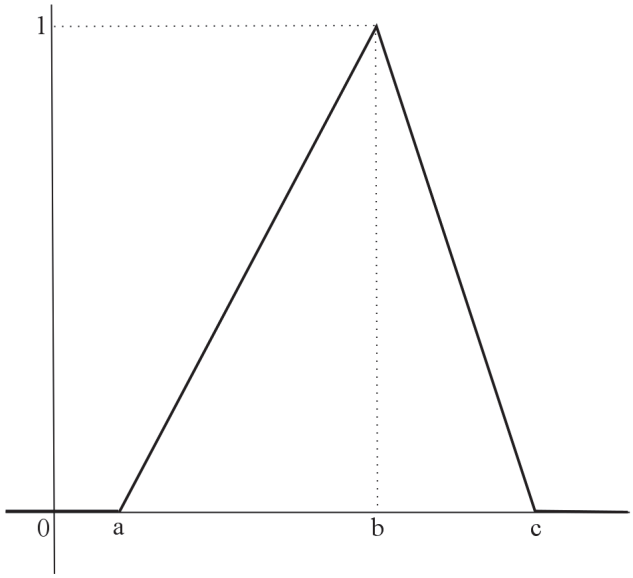


Figure 4.13. Graph of the modified hat function for blending textures together using the information in the description map.

A limitation of the multi-material description map presented is that the materials can not mix. This is due to the use of the alpha channel as an identifier for the material, if the materials id mix due to alpha blending the mapping between the material

parameters and diffuse and displacement textures is lost. A possible solution for this is to use three-dimensional textures, containing a material description per slice. The texture could be traversed along its height at each texture location (u, v) and the parameters blended and diffused to neighbouring location as the materials which compose the soil mixes, the alpha channel could be used to alpha blend the textures using blend maps [**55**] computed at run time. Although this potentially solves the problem of mixing different materials it is both memory and computationally more expensive than the solution used in this work.

4.7. Summary

In this chapter a method for local deformation of multi-material terrains based on diffusion theory has been presented. The mathematical model for the simulation, based on a modification of the drift diffuse equation, has been explained and a discretisation for it has been given. The model was implemented on a compute shader and it is capable to simulate multiple materials at the same time, however material mixing is not possible. Regions of the terrain composed by multiple materials have been identified using a description map, which encodes the material parameters in the red, green and blue channels of a texture, the alpha channel is used as ID for the mapping between the region and the diffuse and displacement textures. In order to create and edit description map an editor has been built that outputs the description map itself and a Domain and a Pixels shaders, which perform the mappings between the regions IDs and the diffuse and displacement maps. The editor addresses the *Regions Classificaiton* component of the framework presented in this thesis, while the mathematical model addresses the *Terrain Defromation and Dynamic Normals* component. Results obtained with the method presented in this chapter show that local deformations for different soil materials in the simulation are visually comparable to tracks left on natural soils composed by the same material.

CHAPTER 5

Conclusions and Further Work

In this chapter contributions of this work will be summarized, further work needed to complete and expand the framework will be outlined and conclusions for this work given.

In nature terrains are not static as they interact with the rest of the environment changing shape on different levels.

As pointed out in chapter 1 this dynamism in a terrain is known in CG as Dynamic Terrain (DT). DT are mesh representations of a terrain capable of changing their morphology as a consequence of their interaction with the user of other agents in the virtual environment. The challenge in locally representing DTs correctly is that terrain soils are vastly heterogeneous and behave differently depending on their composition. This heterogeneity implies that many parameters need to be taken into consideration to simulate each element of the terrain at different levels. Moreover, no unified physics model exists that is capable of describing every single material at once, further increasing the difficulty of reproducing local deformations during simulation. This work contributed to the solution of the problem of simulating local terrain deformation by introducing:

a) a framework for local terrain deformations, which takes into account the terrain composition, chapter 1;
b) a method for unsupervised reconstruction of small rocks from a single image, chapter 3;
c) a unifying mathematical model, based on diffusion theory, capable of simultaneous multi-material simulations, chapter 4;

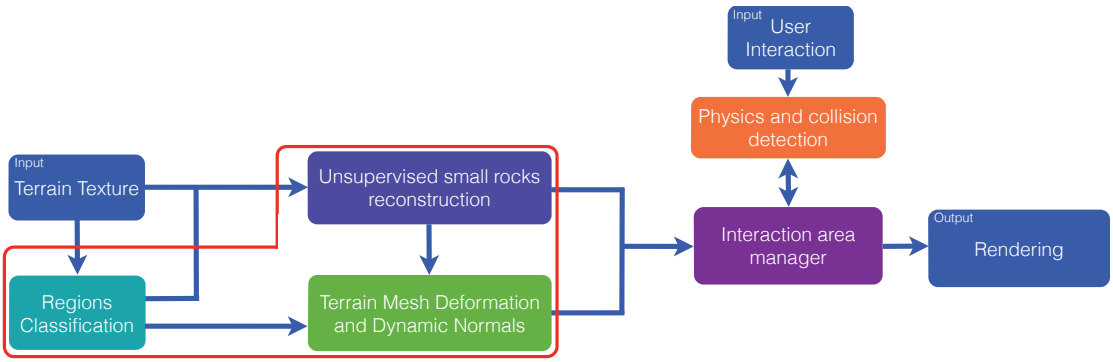


Figure 5.1. Overview of the framework. The Region Classification, Unsupervised small rocks reconstruction and the Terrain Mesh Deformation and Dynamic Normals, highlighted in the red box, are the three fundamental components developed in this thesis.

5.0.1. The framework. The framework for local simulation of local terrain deformations, illustrated in figure 5.1, takes into account the terrain composition and aims to simulate multi-material terrains.

The framework relies on a three level of details paradigm. To understand this paradigm two ideal circles of radii r_1 and r_2, with $r_1 < r_2$, centred on the camera are considered, see figure 5.2. Deformations happening further than r_2 are considered

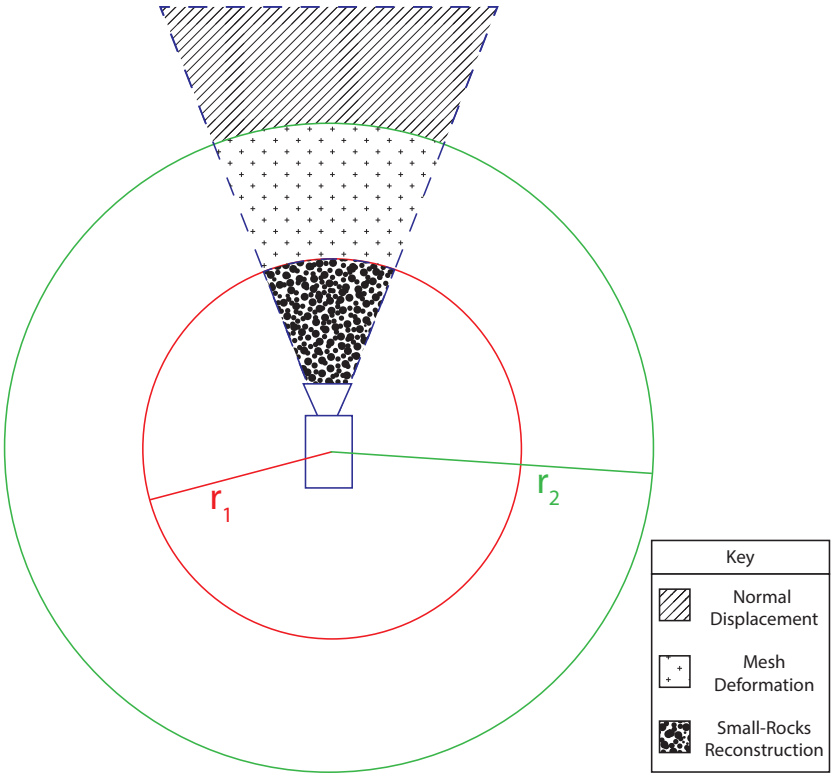


Figure 5.2. Diagram of the three levels of details upon which the framework is designed

away from the point of view and are simulated by dynamically displacing the terrain mesh normals, deformation between r_1 and $r2$ are simulated by dynamically displacing the terrain mesh while deformations happening within r_1 are considered in a neighbourhood of the camera and if happening on gravel soil are simulated by particles otherwise by mesh displacement.

For gravel soil within r_1 the shapes of the grains are reconstructed from the terrain texture. To understand whether it is possible to locally simulate terrains as particle systems in real time the number of particle necessary to reconstruct a cube meter of gravel soil was estimated in chapter 1. From this estimate we see that between 85 to 4×10^4 small-rocks are needed to simulate a cube meter of gravel soil, such a number is within capabilities of modern techniques for particles simulations. Moreover, a series of techniques for particles culling were taken into consideration and discussed. These techniques further facilitate the use of particles for local terrain representations.

The framework is organized in five main components:

1. Region Classification: identifies areas with different materials;
2. Unsupervised small rocks reconstruction: reconstructs the shape of particles from the terrain texture for gravel based soils;
3. Terrain Mesh Deformation and Dynamic Normals: computes the terrain mesh deformations accordingly with a mathematical model based on diffusion theory;
4. Interaction area manager: determines where particles need to be loaded;
5. Physics and collision detection: simulates the particles for the terrain and detects where interactions with the terrain happen;

The components of the framework interact with each other as in the high level pseudo-code presented in algorithm 5.1.

Algorithm 5.1 High level pseudo-code for the framework proposed in this work

```
1: Generate description map
2: for all Gravel Textures do
3:     Pre-compute small-rock meshes from terrain texture and save them
4: end for
5: Divide the terrain in M areas, A_i, i = 0, ..., M − 1
6: for all A_i, i = 0, ..., M − 1 do
7:     Label each A_i accordingly with the soil it contains using the description map
8: end for
9: while the application runs do
10:    Tessellate terrain mesh inside the area of radius r_2
11:    for all Gravel areas A_g, g = 0, ..., K − 1 do
12:        if A_g intersects r_1 radius from camera and the view volume then
13:            if Particles are not loaded yet then
14:                Temporarily displace mesh down by a value h_g for the whole area
15:                Obtain the dimensions of the volume V formed by A_g and h_g
16:                Load physics particles and small-rock meshes in V
17:            end if
18:        else if A_g outside the r_1 radius from the camera then
19:            Bake meshes back into terrain texture
20:        end if
21:    end for
22:    Identify collision areas with the soil, A_i, i = 0, ..., N − 1
23:    Set areas active
24:    for all A_i, i = 0, ..., N − 1 do
25:        if A_i is a gravel area then
26:            Run particle simulation
27:            Update position of small-rock meshes based on the particle simulation
28:        else
29:            Run drift-diffusion simulation
30:            Update DDH
31:            Displace mesh and normals using DDH
32:        end if
33:    end for
34: end while
```

In this work three main components of the framework were developed: the *Unsupervised small rocks reconstruction* component (line 3 in algorithm 5.1 on the preceding page), the *Terrain Deformation and Dynamic Normals* component (lines 10, 29, 30 and 31) and the *Regions Classification* component (line 1, which is used in lines 7 and 29). Figure 5.3 represent the component developed in this work graphically.

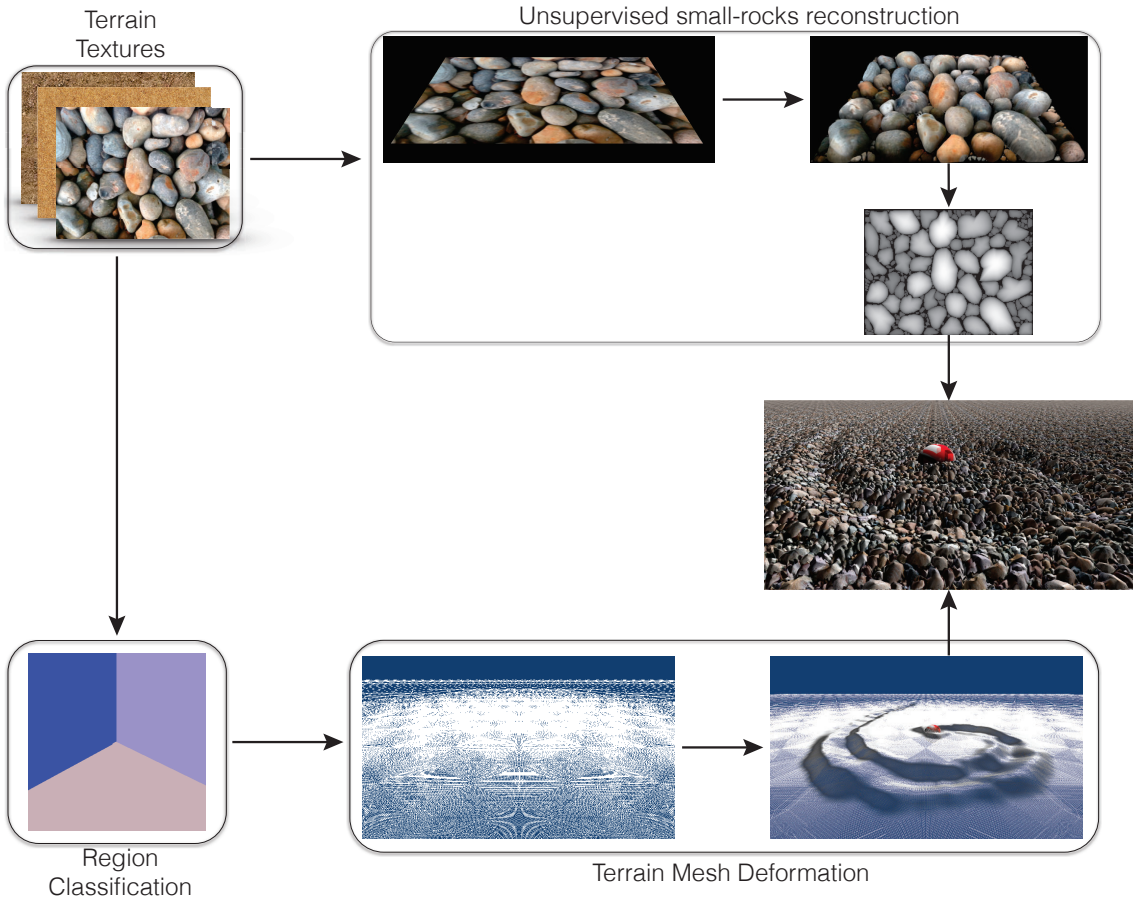


Figure 5.3. Diagram of the framework components developed in this thesis

The framework presented focuses on local deformation of the terrain, and does not consider terrain level of details on a large scale. However, its modular nature allows it to easily be integrated into larger frameworks such as those recently proposed by Yusov [**3**] and Schäfer [**39**], further work can be done towards this integration.

5.0.2. Unsupervised small rocks reconstruction. In order to keep visual coherence between levels of details particles are reconstructed from the terrain texture. To achieve this a parametric mathematical description has been given by introducing generalized ellipsoids. In order to reconstruct the mesh of a small rock the mathematical description requires a centre and a set of radii of the equatorial slice of the rock, these have been extracted from the two dimensional texture of the terrain by identifying the equatorial slice of each small rock and sampling it in various directions. When the equatorial slice representation was incomplete identification of outliers in the sampling was required. This identification was achieved

by exploiting the concept of signed turning angle for a polygon. Results from the reconstruction method show that it is capable of automatically reconstructing small rocks form a single two dimensional image under the condition that the equatorial slices of the small rocks are star shaped sets of genus zero.
Limitation of the method are that it is specialized in reconstructing small rocks only and only from images taken so that the view vector is orthogonal to the plane where the small rocks lies. The method also over-segments the image where sharp corners are present thus crumbling the final small rock mesh. Moreover, the method relies on a series of parameters, which have been estimated heuristically.

5.0.3. Terrain Mesh Deformation and Dynamic Normals and Region Classification. In order to deform the terrain a mathematical model based on a modified form of the Smoluchowsky equation has been proposed. This approach unify multiple materials simulation in a single model. The method computes on a GPU the displacements for the deformation and applies them to a locally tessellated terrain mesh. To compute the displacements the method relies on a description map which classifies regions of the terrain composed of different materials and stores the parameters used for the simulation, i.e. the angle of repose, the mobility, the diffusion coefficient and an ID that identifies the type of material. Results show that the method is capable of realistically deforming the terrain and it is capable of simulating different materials at the same time. However, a limitation for the method is that in the current form is not capable of mixing materials together. This is mainly due to the data structure chosen to describe the regions of the terrain with different materials. When mixing of material is achieved the physics parameters, such as the angle of repose, should be extrapolated for aggregate materials that are formed during simulation, for instance when mixing of pebbles and sand. Further work should be done by understanding whether an interpolation technique will suffice in determining them or whether more complex approaches should be taken.
In this work the discretisation of mathematical model presented is obtained through explicit methods, which are unstable for large time-steps. Further work should be done to obtain an implicit discretisation of the mathematical model as these are unconditionally stable disecretisation methods. Finally, it is important to observe that the method proposed is a 2D method, therefore it is not possible to generate overhangs when digging into the terrain nor deposit material on top of other objects.

5.1. Further Work

Further work is necessary to fully develop the framework proposed. Two components need to be completed the *Interaction Manager* and the *Physics and collision detection*, as highlighted in red in figure 5.4. In this section these components will be discussed and directions for their development will be discussed.

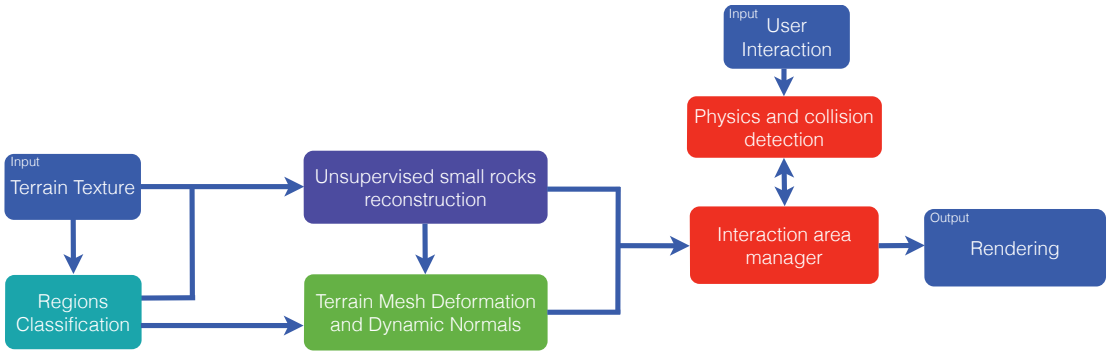


Figure 5.4. Diagram of the system with components in need of development highlighted in red.

5.1.1. Interaction areas. Further work will be necessary to develop this component. This component manages the loading and unloading of the small rocks particles. If the terrain is completely covered by small rocks it will be computationally too expensive to recreate every single one of them. However, as observed in chapter 1, if only the top layers of rocks closest to the camera are reproduced the number of particles, and rock meshes, to handle is manageable.
In order to achieve this, the terrain is divided in equal areas forming a grid, called an interaction grid, see figure 5.5 on the following page. Each area becomes active whenever there is an interaction by the user in that area, and the area is close to the camera and the area is partially or completely covered by rocks. When the interaction area becomes active, rocks meshes and the computation particles which guide their movement are loaded and used to simulate the terrain. Deformations already present on the mesh should reflect on the particles loaded. This can be achieved by displacing the particles using the DDH information.

The area becomes inactive when the user is far away from it. When this happens, the particles and meshes are backed into the terrain texture, unloaded and replaced by the terrain mesh. To maintain deformation coherence the displacement of particles should also be transferred to the terrain mesh. This could be achieved by getting the displacement in y-position of the top layer particles and saving it on the DDH. Top layer particles can be identified by labelling them as they move, while the displacement can be obtained by confronting their y-position with the value contained in the DDH of the area. The particles that update a specific pixel in the DDH can be identified by checking the x and z coordinates and clustering them to an integer by using the floor operator. As many particles may correspond

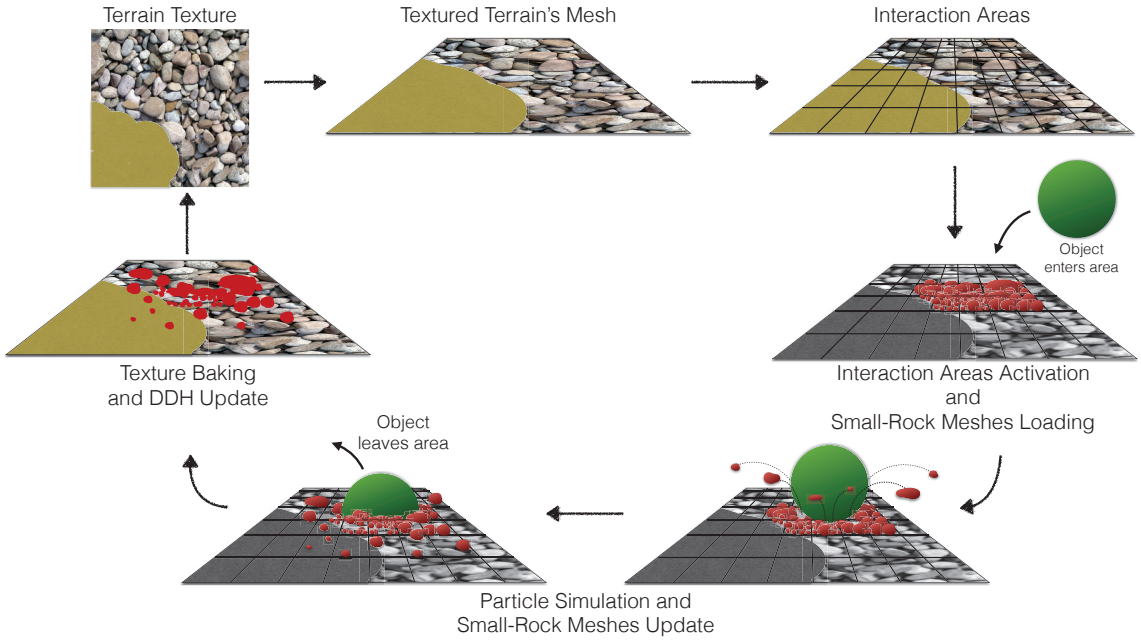


Figure 5.5. Diagram showing the design of the interaction manager component. Rocks meshes are highlighted in red to better distinguish them from the terrain texture. See colour plate C.10 on page 126 for a larger version

to a single pixel of the DDH the average y-displacement can be computed to be used as an update value for the pixel in the DDH.

5.1.2. Physics and collision detection. Further work will be necessary to develop this component. This component is in charge of computing the collisions areas between objects and the terrain and of simulating granular materials, driving the animation of terrains covered by small rocks. Although this work did not develop this component it is of interest to note that in the literature there are many techniques that allow simulating in real time a small amount of particles for granular materials. As observed in chapter 1 the work of Narain, Golas and Lin [**23**] can be used as basis to further develop the component.
Collision detection was not formally addressed in this work, techniques such as those described by Aquilio [**31**] and Schaffer [**39**] can be used to accurately capture the shapes of the objects colliding with the terrain. In particular, as Aquilio [**31**] version of the DDH, ADDH, is the inspiration of this work it can be easily integrated into it. Aquilio's method renders an orthographic depth map, ADDH, of the objects interacting with the terrain from underneath the terrain's surface. The depth map contains the penetration of the object in the terrain. Both ADDH and the simulation DDH presented in this work, SDDH, contains a single value representing the displacement of the terrain due to the object interaction. The difference between the techniques lays in the way the displacement occurs, while Aquilio applies the penetration directly, in this work the displacement is obtained from a drift-diffusion

like simulation, see equation (4.10). When integrating the techniques the information from the ADDH can be used to compute the force on the terrain area needed for the drift-diffusion simulation.
A possible algorithm for integrating the ADDH with the SDDH is the following:

1. Render the ADDH, with resolution equal to the SDDH;
2. Set the ADDH to be a reading resource for the compute shader;
3. Set SDDH to be a read/write resource for the compute shader;
4. Set the compute shader to have a computation thread t_{ij} for each texel in the DDHs
5. Render ADDH;
6. Update ADDH resource in the compute shader;
7. In each compute shader thread t_{ij}:
   7a. Set $p_{ij} = \text{ADDH}[i, j]$;
   7b. Set $q_{ij} = \text{SDDH}[i, j]$;
   7c. If $p_{ij} \leq q_{ij}$
   7d. Re-scale p_{ij} to world sizes;
   7e. Set the force for the thread t_{ij} to $F_{t_{ij}} = p_{ij}\zeta^{-1}F_{obj}$;
   7f. Else set the force to $F_{t_{ij}} = 0$;
   7g. Evaluate equation (4.10);
   7h. Update SDDH;
8. In the domain shader, displace the terrain mesh accordingly with the SDDH;
9. Repeat from 5 until the application ends;

This integration will allow any shape to be detected on the terrain and to simulate material displacement accordingly with the method proposed in this work.

5.1.3. Region Classification. Although a solution for this component has been developed in this work it can be further improved by automatically generating description maps based on the terrain morphology and the texture, for example taking in account the slope of the terrain. A way to achieve automatic classification in the case of virtual texturing is to pre-process the mega-texture by segmenting it and subsequently classifying its segments based on their textures, the classification can then be used to automatically generate a description map in an automated manner. Another improvement necessary is to devise a data structure which better maps the terrain material to the texture thus allowing for material mixing during the simulation. A possible solution for this is to use three-dimensional textures, containing a material description per slice. The texture could be traversed along its height at each texture location (u, v) and the parameters blended and diffused to neighbouring location as the materials that compose the soil mixes, the alpha channel could be used to alpha blend the textures using blend maps [**55**] computed at run time.

5.2. Conclusions

In conclusion, the terrain is an important element of the natural environment and reproducing its dynamical behaviour in a virtual environment is one of the key factors for reproducing immersive environments. Tracks left by objects on the terrain tell the history of what happened in the area in a recent past and the capability of reproducing them in an interactive application adds elements of realism to the whole scene. Although in interactive applications the user can usually modify the morphology of the terrain on a large scale he can rarely leave permanent tracks of his passage. This omission is often done for efficiency reasons, but it limits the interaction of the terrain with the rest of the environment detracting from the realism of the virtual scene as, in nature, terrains are highly deformable and behave differently on a local scale depending on their composition.

This problem has been addressed in the literature multiple times, as discussed in chapter 2, however no approach capable so far of unifying different terrains under a single mathematical model and simulate different material behaviours at the same time has been developed so far. In this work a mathematical approach for local terrain deformations that unifies multi-materials terrain simulations in a single model was successfully developed. Although not physically accurate, results obtained in this work show that the model, based on diffusion theory, produces terrain deformations comparable to tracks left on a natural terrain. In addition to the mathematical model this work also introduced a method for unsupervised small rocks reconstruction from a single image demonstrating that it produces realistic models of small rocks. Finally, these results have been framed in a larger framework for local terrain deformation, which is based on a three level of details and it is capable of capturing different terrains behaviours depending on their composition and distance from the viewer. Although further work is necessary to perfect them, the framework and the methods proposed in this work are a significant step toward generating realistic interactive multi-material terrains in real-time, which mimics all behaviours of a natural terrain, bringing natural environments simulations a step closer to reality.

References

[1] Renato Pajarola and Enrico Gobbetti. Survey of semi-regular multiresolution models for interactive terrain rendering. *Vis. Comput.*, 23(8):583–605, June 2007. (Cited on pages 1, 19, 20, 21, and 74.)

[2] Marc Treib, Florian Reichl, Stefan Auer, and Rüdiger Westermann. Interactive Editing of GigaSample Terrain Fields. *Comput. Graph. Forum*, 31(2), 2012. (Cited on page 1.)

[3] Egor Yusov. Real-Time Deformable Terrain Rendering with DirectX 11. In *GPU Pro 3*, chapter 2, pages 13–39. 2012. (Cited on pages 1, 24, 80, and 88.)

[4] Charles L. Miller. *The theory and application of the digital terrain model*. PhD thesis, MIT, 1958. (Cited on pages 1 and 16.)

[5] Thomas M. Strat and Thomas M Start. Shaded Perspective Images Of Terrain. *M.I.T. Art. Int. Memo.*, 463:38, 1978. (Cited on pages 1 and 16.)

[6] Thomas K Peucker, Robert J. Fowler, James J. Little, and David M. Mark. The triangulated irregular network. In *Proc. Auto Cart. III, Am. Congr. Surv. Mapping, Fallschurch, Virginia*, pages 516–540, 1978. (Cited on pages 1 and 18.)

[7] James H Clark. Hierarchical geometric models for visible surface algorithms. *Commun. ACM*, 19(10):547–554, October 1976. (Cited on pages 1, 18, and 19.)

[8] David Luebke, Martin Reddy, Jonathan D. Choen, Amitabh Varshney, Benjamin Watson, Robert Huebner, Jonathan D. Cohen, and David Luebeke. *Level of Detail for 3D Graphics*. Morgan Kufmann Publishers, fifth edit edition, 2003. (Cited on pages 1, 19, 20, and 21.)

[9] Paolo Neo. Field close-up. `http://www.public-domain-image.com/full-image/nature-landscapes-public-domain-images-pictures/field-public-domain-images-pictures/field-closeup.jpg-free-photograph.html`. On-line; last accessed: 04 April 2015. (Cited on page 2.)

[10] F Kenton Musgrave, Craig E Kolb, and Robert S Mace. The Synthesis and Rendering of Eroded Fractal Terrains. *Comput. Graph. (ACM).*, 23(3):41–50, 1989. (Cited on pages 2, 16, and 17.)

[11] Xin Li and J Michael Moshell. Modeling Soil : Realtime Dynamic Models for Soil Slippage and Manipulation. In *Proc. 20th Annu. Conf. Comput. Graph. Interact. Tech. (ACM SIGGRAPH)*, 1993. (Cited on pages 2, 27, and 28.)

[12] Benoit Chanclou, Annie Luciani, and Arash Habib. Physical Models of Loose Soils Dynamically Marked by a Moving Object. In *Proceedengs 9th IEEE Comput. Animat. Conf.*, pages 27–35, 1996. (Cited on pages 2 and 28.)

[13] Marta Pla-Castells, Ignacio García-Fernández, and Rafael J. Martínez-Durá. Physically-Based Interactive Sand Simulation. In *Eurographics 2008*, 2008. (Cited on pages 2 and 78.)

[14] Robert W. Sumner, James F. O'Brien, and Jessica K. Hodgins. Animating Sand, Mud, and Snow. *Comput. Graph. Forum*, 18(1):17–26, March 1999. (Cited on pages 2, 28, 29, 30, and 31.)

[15] Koichi Onoue. Virtual Sandbox. *Proceedings. 11th Pacific Conf. Comput. Graph. Appl. 2003.*, pages 252–259, 2003. (Cited on pages 2, 28, 29, 30, 31, 32, and 38.)

[16] Koichi Onoue and Tomoyuki Nishita. An Interactive Deformation System for Granular Material. *Comput. Graph. Forum*, 24(1):51–60, March 2005. (Cited on pages 2, 28, 29, 30, 31, 32, and 38.)

[17] Ya-Lun Zeng, Charlie Irawan Tan, Wen-kai Tai, Mau-tsuen Yang, Cheng-chin Chiang, and Chin-chen Chang. A momentum-based deformation system for granular material. *Comput. Animat. Virtual Worlds*, 18(4-5):289–300, 2007. (Cited on pages 2 and 32.)

[18] Ying Zhu, Xiao Chen, and Scott G Owen. Terramechanics Based Terrain Deformation for Real-Time Off-Road Vehicle Simulation. *Adv. Vis. Comput. - Lect. Notes Comput. Sci.*, 6938:431–440, 2011. (Cited on pages 2 and 33.)

[19] Nathan Bell, Yizhou Yu, and Peter J Mucha. Particle-Based Simulation of Granular Materials. In *SCA '05 Proc. 2005 ACM SIGGRAPH/Eurographics Symp. Comput. Animat.*, pages 29–31, 2005. (Cited on pages 3, 34, 38, and 39.)

[20] Iván Alduán, Angel Tena, and Miguel A Otaduy. Simulation of High-Resolution Granular Media. In *CEIG '09*, volume 11, pages 1–8, 2009. (Cited on pages 3 and 4.)

[21] Toon Lenaerts. *Unified particle simulations and interactions in computer animation*. PhD thesis, Katholieke Universiteit Leuven, 2009. (Cited on page 3.)

[22] Iván Alduán and Miguel A Otaduy. SPH Granular Flow with Friction and Cohesion. In *Proc. 2011 ACM SIGGRAPH/Eurographics Symp. Comput. Animat.*, pages 25–32, 2011. (Cited on pages 3, 36, and 37.)

[23] Rahul Narain, Abhinav Golas, and Ming C Lin. Free-flowing granular materials with two-way solid coupling. In *ACM SIGGRAPH Asia 2010 Pap. SIGGRAPH ASIA '10*, volume 29, page 1, New York, New York, USA, 2010. ACM Press. (Cited on pages 3, 14, 36, and 91.)

[24] Christoph Ammann, Doug Bloom, Jonathan M Cohen, John Courte, Lucio Flores, and Sho Hasegawa. The Birth of Sandman. In *SIGGRAPH 2007 - Sketches*, 2007. (Cited on page 3.)

[25] Thomas C. Hales, Samuel P. Ferguson, and Jeffrey C. Lagarias. *The Kepler Conjecture*. Springer New York, 2011. (Cited on pages 3 and 4.)

[26] Chester K Wentworth. A Scale of Grade and Class Terms for Clastic Sediments. *J. Geol.*, 30(5):377–392, 1922. (Cited on pages 3, 4, and 5.)

[27] Titus Tscharntke. White sand. `http://www.public-domain-image.com/full-image/nature-landscapes-public-domain-images-pictures/sand-dunes-public-domain-images-pictures/white-sand.jpg-royalty-free-stock-photograph.html`. On-line; last accessed: 31 March 2015. (Cited on page 5.)

[28] Bo Zhu and Xubo Yang. Animating Sand as a Surface Flow. In *EUROGRAPHICS 2010 (Short Pap.*, pages 9–12, 2010. (Cited on pages 5, 38, 39, and 40.)

[29] Seth R. Holladay and Parris K. Egbert. Solid-state Culled Discrete Element Granular Systems. In *Eurographics 2012 Short Pap.*, pages 65–68, 2012. (Cited on pages 5, 39, and 40.)

[30] Seth Holladay and Parris K. Egbert. Granular material deposition for simulation and texturing. In Francisco J. Perales Lpez, Robert B. Fisher, and Thomas B. Moeslund, editors, *AMDO*, volume 7378 of *Lecture Notes in Computer Science*, pages 163–172. Springer, 2012. (Cited on page 5.)

[31] Anthony Aquilio, Jeremy Brooks, Ying Zhu, and G. Owen. Real-time gpu-based simulation of dynamic terrain. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Paolo Remagnino, Ara Nefian, Gopi Meenakshisundaram, Valerio Pascucci, Jiri Zara, Jose Molineros, Holger Theisel, and Tom Malzbender, editors, *Advances in Visual Computing*, volume 4291 of *Lecture Notes in Computer Science*, pages 891–900. Springer Berlin / Heidelberg, 2006. 10.1007/11919476_89. (Cited on pages 6, 14, 23, 31, 32, 67, 80, and 91.)

[32] Marco Gilardi, Phil L. Watten, and Paul Newbury. Unsupervised three-dimensional reconstruction of small rocks from a single two-dimensional image. In *Eurographics 2014 Short Pap.*, pages 29–32, 2014. (Cited on pages 7, 8, 14, 43, and 64.)

[33] Martin Mittring and Crytek GmbH. Advanced virtual texture topics. *ACM SIGGRAPH 2008 classes - SIGGRAPH '08*, page 23, 2008. (Cited on pages 9 and 25.)

[34] Javier Taibo, Antonio Seoane, and Luis Hernández. Dynamic Virtual Textures. In *17th Int. Conf. Cent. Eur. Comput. Graph. Vis. Comput. Vision, WSCG'2009*, pages 25–32, 2009. (Cited on pages 9 and 25.)

[35] Graham Sellers, Juraj Obert, Patrick Cozzi, Kevin Ring, Emil Persson, Joel de Vahl, and J. M. P. van Waveren. Rendering massive virtual worlds. In *ACM SIGGRAPH 2013 Courses*, pages 1–88, New York, New York, USA, 2013. ACM Press. (Cited on pages 9 and 25.)

[36] Luc Vincent and Edward R Dougherty. Morphological Segmentation for Textures and Particles. *Digit. Image Process. Fundam. Appl.*, pages 43–102, 1994. (Cited on page 9.)

[37] Manik Varma and Andrew Zisserman. A Statistical Approach to Texture Classification from Single Images. *Int. J. Comput. Vis. - Spec. Issue Texture Anal. Synth.*, 62(1-2):61 – 81, 2004. (Cited on page 9.)

[38] Johan Andersson. Advanced Real-Time Rendering in 3D Graphics and Games Course: Terrain Rendering in Frostbite Using Procedural Shader Splatting. In *ACM SIGGRAPH 2007 courses - SIGGRAPH '07*, 2007. (Cited on pages 9 and 25.)

[39] Henry Schäfer, Benjamin Keinert, Matthias Nieß ner, Christoph Buchenau, Michael Guthe, and Marc Stamminger. Real-Time Deformation of Subdivision Surfaces from Object Collisions. In *Proc. 6th High-Performance Graph. Conf.*, pages 1–8, 2014. (Cited on pages 14, 34, 67, 80, 88, and 91.)

[40] Paul R. Wolf, Bon A. Dewit, and Wilkinson Benjamin E. *Elements of Photogrammetry with applications in GIS*. Morgan Kaufmann, fourth ed edition, 2014. (Cited on page 16.)

[41] Ondrej St'ava, Bedich Beneš, Matthew Brisbin, and Jaroslav Kriv´anek. Interactive Terrain Modeling Using Hydraulic Erosion. In *Proc. 2008 ACM SIGGRAPH/Eurographics Symp. Comput. Animat.*, 2008. (Cited on page 17.)

[42] Hugues Hoppe. Progressive Meshes. In *Proc. 23rd Annu. Conf. Comput. Graph. Interact. Tech. - SIGGRAPH '96*, pages 99–108, New York, New York, USA, 1996. ACM Press. (Cited on page 19.)

[43] Falko Löffler and Heidrun Schumann. Generating Smooth High-Quality Isosurfaces for Interactive Modeling and Visualization of Complex Terrains. In *Vision, Modeling, and Visualization*, pages 79–86, 2012. (Cited on page 19.)

[44] J. Kaba, J. Matey, G. Stoll, H. Taylor, and P. Hanrahan. Interactive terrain rendering and volume visualization on the Princeton Engine. In *Proc. Vis. '92*, 1992. (Cited on page 19.)

[45] Rüdiger Westermann and Thomas Ertl. Efficiently using graphics hardware in volume rendering applications. *Proc. 25th Annu. Conf. Comput. Graph. Interact. Tech. - SIGGRAPH '98*, pages 169–177, 1998. (Cited on page 19.)

[46] Mark Miller, Andrew Cumming, Kevin Chalmers, Benjamin Kenwright, and Kenny Mitchell. Poxels : Polygonal Voxel Environment Rendering. In *Proc. 20th ACM Symp. Virtual Real. Softw. Technol.*, pages 235–236, 2014. (Cited on page 19.)

[47] William E Lorensen and Harvey E Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Comput. Graph. (ACM).*, 21(4):163–169, 1987. (Cited on page 19.)

[48] Hanan Samet. *Foundations of Multidimensional And Metric Data Structures*. Morgan Kaufmann, 2006. (Cited on page 21.)

[49] W. Evans, D. Kirkpatrick, and G. Townsend. Right-Triangulated Irregular Networks. *Algorithmica*, 30(2):264–286, 2001. (Cited on page 21.)

[50] M. Duchaineau, M. Wolinsky, D.E. Sigeti, M.C. Miller, C. Aldrich, and M.B. Mineev-Weinstein. ROAMing terrain: Real-time Optimally Adapting Meshes. *Proceedings. Vis. '97 (Cat. No. 97CB36155)*, pages 81–88, 1997. (Cited on page 22.)

[51] Yefei He. *Real-time visualization of dynamic terrain for ground vehicle simulation*. PhD thesis, University of Iowa, 2000. (Cited on page 22.)

[52] Yefei He, James Cremer, and Yiannis Papelis. Real-time extendible-resolution display of on-line dynamic terrain. In *Proceedings of the 2002 Conference on Graphics Interface*, 2002. (Cited on page 22.)

[53] C Bloom. Terrain texture compositing by blending in the frame-buffer (aka "Splatting" Textures). `http://www.cbloom.com/3d/techdocs/splatting.txt`, 2000. On-line; last accessed: 25 March 2015. (Cited on page 24.)

[54] Jonathan Ferraris and Christos Gatzidis. A Rule-based Approach to 3D Terrain Generation via Texture Splatting. In *Proc. 16th ACM Symp. Virtual Real. Softw. Technol.*, pages 407–408, 2009. (Cited on page 24.)

[55] Alexandre Hardy and Duncan Andrew Keith Mc Robrerts. Blend Maps : Enhanced Terrain Texturing. In *Proc. SAICSIT 2006*, pages 61–70, 2006. (Cited on pages 24, 84, and 92.)

[56] Morgan McGuire and Max McGuire. Steep parallax mapping. *I3D 2005 Poster*. (Cited on page 26.)

[57] Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi. Smoothed Particle Hydrodynamics on GPUs. In *Proc. Comput. Graph. Int.*, pages 63–70, 2007. (Cited on pages 27 and 35.)

[58] Witawat Rungjiratananon, Zoltan Szego, Yoshihiro Kanamori, and Tomoyuki Nishita. Real-time Animation of Sand-Water Interaction. *Pacific Graph.*, 27(7), 2008. (Cited on pages 27 and 35.)

[59] Takahiro Harada. Heterogeneous Particle-based Simulation. In *SIGGRAPH Asia 2011 Sketches*, volume 23, 2011. (Cited on pages 27 and 37.)

[60] Bedich Beneš and R. Forsbach. Layered data representation for visual simulation of terrain erosion. In *Proc. Spring Conf. Comput. Graph.*, pages 80–86. IEEE Comput. Soc, 2001. (Cited on pages 29 and 30.)

[61] Da Wang and Cheng Wang. Real-time GPU-based Simulation of Dynamic Terrain in Virtual Battlefield. *J. Comput. Inf. Syst.*, 7(6):1924–1933, 2011. (Cited on page 32.)

[62] Dong Wang, Qing-sheng Zhu, and Yi Xia. Real-time Multiresolution Rendering for Dynamic Terrain. *J. Softw.*, 9(4):889–894, 2014. (Cited on page 32.)

[63] Yongning Zhu and Robert Bridson. Animating Sand as a Fluid. *ACM Trans. Graph.*, pages 965–972, 2005. (Cited on pages 32, 33, 36, and 38.)

[64] Marta Pla-Castells, Ignacio García, and Rafael J Martínez. Aproximation of continuous media models for granular systems using cellular automata. *Lect. Notes Comput. Sci.*, pages 230–237, 2004. (Cited on page 33.)

[65] Marta Pla-Castells, Ignacio Garc, Ignacio García-Fernández, and Rafael J. Martínez-Durá. Interactive terrain simulation and force distribution models in sand piles. *Cell. Autom. - Lect. Notes Comput. Sci.*, 4173:392–401, 2006. (Cited on page 33.)

[66] Marta Pla-castells, Ignacio García-Fernández, and Rafael J. Martínez-Durá. Modelling And Simulation Of Several Interacting Cellular Automata. In *Ind. Simul. Conf. ISC 2010*, number 2006, 2010. (Cited on page 33.)

[67] Poschel Thorsten and Thomas Schwager. *Computational Granular Dynamics*. Springer-Verlag, 2005. (Cited on page 34.)

[68] Erwin Fehlberg. Low-order classical Runge-Kutta formulas with step-size control and their application to some heat transfer problems. *NASA Tech. Rep.*, (July):R–315, 1969. (Cited on page 34.)

[69] Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi. Real-time Fluid Simulation Coupled with Cloth. In *Proceending Theory Pract. Comput. Graph.*, 2007. (Cited on page 35.)

[70] Toon Lenaerts and Philip Dutré. Mixing Fluids and Granular Materials. *Comput. Graph. Forum*, 28(2), 2009. (Cited on pages 36, 37, and 38.)

[71] AMD. Apu 101: All about AMD Fusion Accellerated Processing Unit. Technical report, 2011. (Cited on page 37.)

[72] Toon Lenaerts, Bart Adams, and Philip Dutré. Porous flow in particle-based fluid simulations. *ACM Trans. Graph.*, 27(3):1, August 2008. (Cited on page 37.)

[73] J.-P. Bouchaud, M. E. Cates, J. Ravi Prakash, and S. F. Edwards. A model for the dynamics of sandpile surfaces. *J. Phys. I*, 4(10):1383–1410, 1994. (Cited on page 39.)

[74] Marc Pollefeys and Luc Van Gool. From Images to 3D Models. *Commun. ACM*, 45(7):50–55, July 2002. (Cited on page 40.)

[75] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Eurographics Symp. Geom. Process.*, pages 61–70, 2006. (Cited on page 40.)

[76] Derek Bradley, Derek Nowrouzezahrai, and Paul Beardsley. Image-based Reconstruction and Synthesis of Dense Foliage. *ACM Trans. Graph.*, 32(4):74:1–74:10, July 2013. (Cited on page 41.)

[77] Chuan Li, Oliver Deussen, Yi-Zhe Song, Phil Willis, and Peter Hall. Modeling and Generating Moving Trees from Video. *ACM Trans. Graph.*, 30(6):127:1–127:12, December 2011. (Cited on page 41.)

[78] Chuan Li, D. Pickup, T. Saunders, D. Cosker, D. Marshall, P. Hall, and P. Willis. Water Surface Modeling from a Single Viewpoint Video. *Visualization and Computer Graphics, IEEE Transactions on*, 19(7):1242–1251, July 2013. (Cited on page 41.)

[79] Paul E Debevec and Camillo J Taylor. Modeling and Rendering Architecture from Photographs : A hybrid geometry- and image-based approach. In *Proceeding SIGGRAPH '96 Proc. 23rd Annu. Conf. Comput. Graph. Interact. Tech.*, pages 11–20, 1996. (Cited on page 41.)

[80] Frank a. van den Heuvel. 3D reconstruction from a single image using geometric constraints. *ISPRS J. Photogramm. Remote Sens.*, 53(6):354–368, December 1998. (Cited on page 41.)

[81] Peter F Sturm and Stephen J Maybank. A Method for Interactive 3D Reconstruction of Piecewise Planar Objects from Single Images. In *10th Br. Mach. Vis. Conf. (BMVC '99)*, volume 89221, pages 265–274, 1999. (Cited on page 41.)

[82] Nianjuan Jiang, Ping Tan, and Loong-Fah Cheong. Symmetric Architecture Modeling with a Single Image. *ACM Trans. Graph. - Proc. ACM SIGGRAPH Asia 2009*, 28(5):1–8, 2009. (Cited on page 41.)

[83] Sujin Liu and Zhiyong Huang. Interactive 3D Modeling Using Only One Image. In *VRST '00 Proc. ACM Symp. Virtual Real. Softw. Technol.*, pages 49–54, 2000. (Cited on page 41.)

[84] Tao Chen, Zhe Zhu, Ariel Shamir, Shi-min Hu, and Daniel Cohen-Or. 3-Sweep : Extracting Editable Objects from a Single Photo. In *Proc. ACM SIGGRAPH Asia 2013*, 2013. (Cited on page 41.)

[85] Naoki Kita and Kazunori Miyata. Interactive Procedural Modeling of Pebble Mosaics. In *SA '11 SIGGRAPH Asia 2011 Sketches*, pages 12–13, 2011. (Cited on page 41.)

[86] Yan Liu and H L Xing. Surface mesh generation of large-scale digital rock images in 3D. *Procedia Comput. Sci.*, 18:1208–1216, 2013. (Cited on page 41.)

[87] Adrien Peytavie, Éric Galin, J Grosjean, and S Merillou. Arches : a Framework for Modeling Complex Terrains. *Comput. Graph. Forum*, 28(2):457–467, April 2009. (Cited on page 41.)

[88] Adrien Peytavie, Éric Galin, J Grosjean, and S Merillou. Procedural Generation of Rock Piles using Aperiodic Tiling. *Pacific Graph.*, 28(7), 2009. (Cited on page 41.)

[89] Kaisei Sakurai and Kazunori Miyata. Procedural Modeling of Multiple Rocks Piled on Flat Ground. In *SIGGRAPH Asia 2010 Posters*, pages 1–2, 2010. (Cited on page 41.)

[90] Isaac M Dart, Gabriele De Rossi, and Julian Togelius. SpeedRock : procedural rocks through grammars and evolution. In *Proc. 2nd Int. Work. Proced. Content Gener. Games*, pages 7–10, 2011. (Cited on page 41.)

[91] Steven Worley. A cellular texture basis function. *Proc. 23rd Annu. Conf. Comput. Graph. Interact. Tech. - SIGGRAPH '96*, pages 291–294, 1996. (Cited on page 46.)

[92] John Canny. A Computational Approach to Edge Detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-8(6):679–698, 1986. (Cited on pages 49 and 62.)

[93] Serge Beucher and Christian Lantuèj. Use of watersheds in contour detection. In *International workshop on image processing, real-time edge and motion detection/estimation*, 1979. (Cited on page 49.)

[94] Jean Serra and Luc Vincent. Lecture notes on morphological filtering. *Cah. du Cent. Morphol. Mathématique*, (8), 1989. (Cited on page 49.)

[95] Jean Serra, Luc Vincent, Centre De Morphologie Math, and Ecole Nationale Sup. An overview of morphological filtering. *Circuits, Syst. Signal Process.*, 11(1):47–108, 1992. (Cited on page 49.)

[96] Eitan Grinspun and Adrian Secord. Introduction to Discrete Differential Geometry : The Geometry of Plane Curves. In *Proceeding SIGGRAPH Asia '08 ACM SIGGRAPH ASIA 2008 courses Artic. No. 5*, pages 1–4, 2008. (Cited on page 53.)

[97] Torus Knot Software Ltd. Managed OGRE (MOGRE) - Rendering Engine. `http://www.ogre3d.org/tikiwiki/MOGRE`, February 2009. On-line; last accessed: 23 March 2015. (Cited on page 60.)

[98] Francesco Carucci. Inside geometry instancing. In Matt Pharr, editor, *GPU Gems 2*, pages 47–67. Addison-Wesley, 2005. (Cited on page 63.)

[99] Mohamed S Ebeida, Scott A Mitchell, Anjul Patney, Andrew A Davidson, and John D Owens. A Simple Algorithm for Maximal Poisson-Disk Sampling in High Dimensions. *Comput. Graph. Forum*, 31(2):2–11, 2012. (Cited on page 63.)

[100] Manuel N. Gamito and Steve C. Maddock. Accurate multidimensional Poisson-disk sampling. *ACM Trans. Graph.*, 29(1):1–19, December 2009. (Cited on page 63.)

[101] Daniel Dunbar and Greg Humphreys. A spatial data structure for fast Poisson-disk sample generation. *ACM SIGGRAPH 2006 Pap. - SIGGRAPH '06*, page 503, 2006. (Cited on page 63.)

[102] M. Doi and S. F. Edwards. The Smoluchowsky equation. In *Theory Polym. Dyn.*, chapter 3.2, pages 46–50. Oxford Science Publications, 1986. (Cited on pages 66, 67, 104, and 105.)

[103] Houssam Hnaidi, Éric Guérin, Samir Akkouche, Adrien Peytavie, and Éric Galin. Feature based terrain generation using diffusion equation. *Pacific Graph.*, 29(7), 2010. (Cited on page 67.)

[104] M. V. Smoluchowsky. Über Brownsche Molekularbewegung unter Einwirkung äußerer Kräfte und deren Zusammenhang mit der verallgemeinerten Diffusionsgleichung. *Ann. Phys.*, 353(24):1103–1112, 1916. (Cited on page 67.)

[105] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, 1975. (Cited on page 72.)

[106] Microsoft Corporation. DirectX11 SDK - Graphics Library. `http://www.microsoft.com/en-gb/download/details.aspx?id=6812`, June 2010. On-line; last accessed: 23 March 2015. (Cited on page 72.)

[107] Kenneth Weiss, College Park, and Leila De Floriani. Sparse Terrain Pyramids. In *GIS '08 Proc. 16th ACM SIGSPATIAL Int. Conf. Adv. Geogr. Inf. Syst.*, page Article No. 15, 2008. (Cited on page 74.)

[108] K Weiss and Leila De Floriani. Simplex and Diamond Hierarchies: Models and Applications. *Comput. Graph. Forum*, 30(8):2127–2155, December 2011. (Cited on page 74.)

[109] Morgan McGuire and Peter G Sibley. A Heightfield on an Isometric Grid. In *SIGGRAPH 2004 - Sketch*, 2004. (Cited on page 74.)

[110] Giles Hodges. Mud. `http://4.bp.blogspot.com/-Mz94fzjf9DM/UmpLfICutiI/AAAAAAAAEk8/8Uid3yVbuzc/s1600/Dirt+00+seamless.jpg`. On-line; last accessed: 09 April 2015. (Cited on page 80.)

[111] Nomer Adona. Pebbles. `https://nomeradona.files.wordpress.com/2009/05/pebbles.jpg`. On-line; last accessed: 09 April 2015. (Cited on page 80.)

[112] Anonymous. Sand. `http://stunningdesignz.com/wp-content/uploads/2013/11/Free-Sand-Textures.jpg`. On-line; last accessed: 09 April 2015. (Cited on page 80.)

[113] Manfredo Perdigao Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice Hall, 1976. (Cited on pages 102 and 103.)

[114] JungHyun Han. *3D Graphics for Game Programming*. Chapman & Hall/CRC, 1st edition, 2011. (Cited on page 110.)

[115] Jason Zink, Matt Pettineo, and Jack Hoxley. *Practical rendering and computation with Direct3D 11*. CRC Press, 2011. An A.K. Peters book. (Cited on page 110.)

[116] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008. (Cited on page 110.)

[117] NVIDIA. Whitepaper - NVIDIAs Next Generation CUDA Compute Architecture: Fermi. Technical report, 2009. (Cited on pages 114 and 115.)

Appendices

APPENDIX A

Mathematical Background

In this chapter an overview of the background concepts needed to develop the main methods presented in this theses is presented.
This appendix covers the mathematical concepts used in chapters 3 and 4 to automatically generate rocks from a single image and to deform terrains in response to user interaction. The concepts necessary for those chapters are presented in sections A.1, A.2 and A.3.

A.1. Surfaces parametrization: a short overview

An elegant way to describe objects in a virtual scene is to give, whenever possible, a mathematical description for them. This form of representation allows for procedural generation of elements of the scene given a specific set of parameters. To explain how do so, some terminology needs first to be specified. This section gives a set of basic definitions that are mainly needed in chapter 3 and also used in other parts of this theses. For further details on the following concepts please refer to [**113**].

Definition A.1 (Regular mapping, [**113**]). A mapping $\mathbf{F} : U \subset \mathbb{R}^2 \to V \subset \mathbb{R}^3$ is said to be a *regular mapping* if for each $p \in U$

$$\left(\frac{\partial \mathbf{F}}{\partial u} \times \frac{\partial \mathbf{F}}{\partial v}\right)(p) \neq 0$$

where $\times$ is the cross product of $\mathbb{R}^3$.

Due to its dependence on the cross product, this definition of regularity can not be extended to dimensions other than 3, but for the purposes of this work it will suffice.

Definition A.2 (Differentiable mapping, [**113**]). A mapping $\mathbf{F} : U \subset \mathbb{R}^2 \to V \subset \mathbb{R}^3$, where U is and open set of $\mathbb{R}^2$, is said to be *differentiable* if all of its components $\mathbf{F}(u, v) = (f_1(u, v), f_2(u, v), f_3(u, v))$ have continuous partial derivatives in U.

Definition A.3 (Homeomorphism, [**113**]). A differentiable mapping $\mathbf{F} : U \to V$, where U is an open set of $\mathbb{R}^2$ and $V \subset \mathbb{R}^3$ is said to be an *homeomorphism* if it is invertible and its inverse $\mathbf{F}^{-1}$ is differentiable.

Definition A.4 (Regular Surface, [**113**]). A subset $S \subset \mathbb{R}^3$ is a *regular surface* if for each $p \in S$ there exists a neighbourhood $V \subset \mathbb{R}^3$ and a mapping $\mathbf{X}$ from an open set $U \subset \mathbb{R}^2$ in $V \cap S \subset \mathbb{R}^3$, $\mathbf{X} : U \to V \cap S$, that is regular, differentiable and an homeomorphism. The mapping $\mathbf{X}$ is called a *local parametrization of S in p.*

DEFINITION A.5 (Monge parametrization, [**113**]). A local parametrization of a surface S in $p \in S$ is called a Monge parametrization if it is of the following form: $\mathbf{X}(u, v) = (u, v, f(u, v))$, where f is a differentiable function from an open set $U \subset \mathbb{R}^2$ in $\mathbb{R}$

It can be proved that the parametrization in definition A.5 is indeed the parmterization of the graph of f [**113**]. It can also be proved that every surface locally admit a Monge parametrization [**113**]. In section 3.1 the following parametrization is generalized to describe pebbles and small rocks. Let $U = (0, 2\pi) \times (-\frac{\pi}{2}, \frac{\pi}{2})$ an open subset of $\mathbb{R}^2$, and let a, b, and c be three positive real numbers. The parametrization $\mathbf{X} : U \to \mathbb{R}^3$ defined by:

$$\mathbf{X}(\varphi, \vartheta) = \begin{cases} x(\varphi, \vartheta) &= a\cos(\varphi)\cos(\vartheta) \\ y(\varphi, \theta) &= b\sin(\vartheta) \\ z(\varphi, \vartheta) &= c\sin(\varphi)\cos(\vartheta) \end{cases} \tag{A.1}$$

describes the surface of an ellipsoid embedded in $\mathbb{R}^3$.

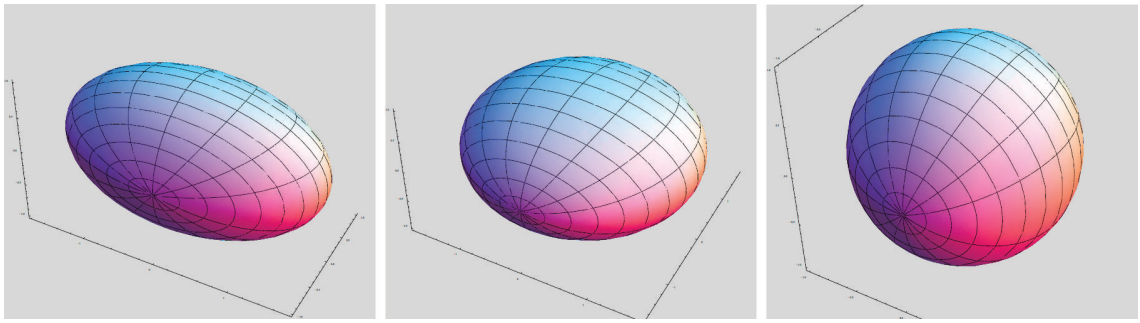


Figure A.1. Visualizations of the ellipsoids obtained using parametrization (A.1) with different values of a, b and c.

Figure A.1 shows some visualizations of parametrization (A.1) with different values of the parameters a, b and c, which represent the lengths of the *semi-principal axes* of the ellipsoid. Notice that if $a = b = c = r_0$, whrere $r_0 \in \mathbb{R}^+$, parametrization (A.1) coincide with a parametrization of a sphere of radius r_0.

A final definition will be useful in the following:

DEFINITION A.6 (Genus of a surface). The *genus* of a surface S is the number of simple closed curves that can be drawn on S without separating it, i.e. it is still possible to reach from a point every other point of the surface through a curve segment on the surface.

The genus of a surface counts the number of holes the surface has. For instance a sphere has genus zero, if a simple closed curve is drawn on the surface of a sphere it separate a piece of the surface from it, as shown on the left of figure A.2. The

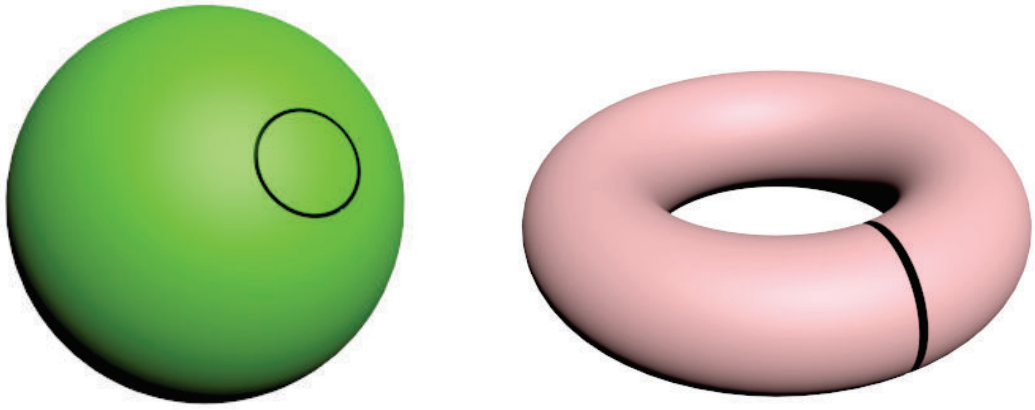


Figure A.2. Examples of simple closed curves drawn on a sphere and on a torus. On a sphere it is not possible to find a closed curve which does not separate the surface. However, on a torus, such curve exists.

number of simple closed curves which do not separate the surface of a sphere is zero. On the other hand, the torus has genus one as it is possible to draw a simple closed curve on its surface which does not separate a piece of its surface from it, as shown on the right of figure A.2.

A.2. The drift-diffusion equation

In chapter 4 terrain deformations are modelled as a drift-diffusion phenomena. In this section the derivation of the drift-diffusion equation is given. This equation is used as the starting point for the description of terrain deformations.

Fick's laws describe diffusion phenomena mathematically. The first Fick's law describe the flux of particles concentration, while the second Fick's law describe the variation of particles concentration over time. Fick's laws can be used to derive a mathematical description of the diffusion of the concentration of a substance into another, the equation which describes this dynamic is known as the drift-diffusion equation, or as the Smoluchowsky equation [**102**].
In this section an overview of the drift-diffusion equation is given and the meaning of its terms explained.

Fick's s law states that the *concentration flux*, $\mathbf{j}(\mathbf{x}, t)$, in an non-uniform ideal mixture is proportional to the spatial gradient of the concentration, $c(\mathbf{x}, t)$, [**102**]:

$$\mathbf{j}(\mathbf{x}, t) = -D\nabla c(\mathbf{x}, t) \tag{A.2}$$

where $\mathbf{x} \in \mathbb{R}^n$, $t \in \mathbb{R}^+$, D is the diffusivity of the substance and $\nabla = \left(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}\right)$ is the gradient operator in $\mathbb{R}^n$.
Fick's second law states that the time variation of the concentration is proportional

to the Laplacian of the concentration, [**102**]:

$$\frac{\partial c(\mathbf{x},t)}{\partial t} = D\Delta c(\mathbf{x},t) \tag{A.3}$$

where $\Delta = \Sigma_{k=1}^{n} \frac{\partial^2}{\partial x_k^2}$ is the Laplacian operator in $\mathbb{R}^n$.

Under the hypothesis that D is constant in space, equation (A.3) can be expressed in terms of the concentration flux:

$$\frac{\partial c(\mathbf{x},t)}{\partial t} = D\Delta c(\mathbf{x},t) = \nabla \cdot (D\nabla c(\mathbf{x},t)) = -\nabla \cdot \mathbf{j}(\mathbf{x},t) \tag{A.4}$$

Let now the substance be under the action of a potential $U(\mathbf{x},t)$. This potential causes a force $\mathbf{F}(\mathbf{x},t) = -\nabla U(\mathbf{x},t)$, which act on the substance adding a velocity $\mathbf{v} = \frac{1}{\zeta}\mathbf{F}(\mathbf{x},t)$, where ζ is the friction constant, to its concentration flux, [**102**]:

$$\mathbf{j}(\mathbf{x},t) = -D\nabla c(\mathbf{x},t) + c(\mathbf{x},t)\frac{1}{\zeta}\mathbf{F}(\mathbf{x},t) \tag{A.5}$$

Assuming that the diffusion coefficient is constant in space and plugging equation (A.5) into equation (A.4) one obtains:

$$\frac{\partial c(\mathbf{x},t)}{\partial t} = D\Delta c(\mathbf{x},t) - \nabla \cdot \left(\frac{c(\mathbf{x},t)}{\zeta}\mathbf{F}(\mathbf{x},\mathbf{t})\right) \tag{A.6}$$

which is kown as the drift-diffusion, or Smoluchowsky, equation [**102**].
Equation (A.6) states that the variation in concentration of a substance over time is due to the diffusion of the concentration in the mixture (first term on the r.h.s.) and to the drift due to the velocity caused by the potential field in which the substance is immersed (second term on the r.h.s.).

In chapter 4 the drift diffusion equation will be modified based on a set of heuristic rules to better describe the behaviour of soil.

A.3. Finite differences methods

In this section a brief overview of the Finite Difference Methods (FDMs) is given as well as the discretisation of two important operators, the gradient and the Laplacian. FDMs aim to find the numerical solutions of a Partial Differential Equation (PDE) by discretising space and time on a grid. Each node in the grid is used as computational point to approximate the value of a solution of the equation in a neighbour of that point.
For simplicity this section describes FDMs for the one dimensional case. Extension to higher dimensions is straightforward and will be given for the gradient and Laplacian operators in subsections A.3.1 and A.3.2.

Let $f : I \subset \mathbb{R} \to \mathbb{R}$ be a real valued function on an interval I of $\mathbb{R}$ with extremes a and b. Let the interval I be subdivided by a monotone succession of $n \in \mathbb{N}$ points,

$a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b$ or $\{x_i\}_{i=0}^n$ for short. The subdivision $\{x_i\}_{i=0}^n$ is called a *grid* on I. If $||x_i - x_{i+1}|| = h$, where $h \in \mathbb{R}$ is constant and $i = 0, 1, \ldots, n$, the grid is said to be a *regular grid*. Computing now f on the regular grid identified by $\{x_i\}_{i=0}^n$ one obtains a succession $\{f(x_i)\}_{i=0}^n$ that is said a *discretisation* of f.
In order to obtain a discretisation of the first derivative of f on a regular grid $\{x_i\}_{i=0}^n$ let's recall the definition of derivative of a one dimensional function:

Definition A.7. Let $f : I \to \mathbb{R}$ be a function from an open interval $I \in \mathbb{R}$ into the set of real numbers. Let also x_0 be in I and h a small real number such that $h \neq 0$ and $x_0 + h \in I$. The function f is said to be derivable in x_0 if the limit:

$$\frac{d}{dx} f\Big|_{x_0} = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h} \tag{A.7}$$

exists and it is finite. The function $\frac{d}{dx} f : I \to \mathbb{R}$ is called the derivative of f.

If the function f is derivable for each $x \in I$ than f is said to be derivable in I.

Informally speaking, on a fixed regular grid the limit in (A.7) can not be computed as h is constant. If the limiting process is dropped one obtains:

$$\frac{d}{dx} f\Big|_{x_0} = \frac{f(x_0 + h) - f(x_0)}{h} + O(h) \tag{A.8}$$

where $O(h)$ is the truncation error term introduced by dropping the limit.
Equation (A.8) can be also obtained formally by developing the function f to the first order around a point $x_0 \in I$ using Taylor's polynomial:

$$f(x_0 + h) = f(x_0) + \frac{d}{dx} f\Big|_{x_0} h + O(h) \tag{A.9}$$

and then rearranging the terms to isolate the first order derivative:

$$\frac{d}{dx} f\Big|_{x_0} = \frac{f(x_0 + h) - f(x_0)}{h} + O(h) \tag{A.10}$$

Observing that on a regular grid $x_{i+1} = x_i + h$:

$$\frac{d}{dx} f\Big|_{x_i} = \frac{f(x_{i+1}) - f(x_i)}{h} + O(h) \tag{A.11}$$

The succession $\left\{ \frac{d}{dx} f\big|_{x_i} \right\}_{i=0}^n$ is a discretisation of $\frac{d}{dx} f$ on a regular grid $\{x_i\}_{i=0}^n$. Equation (A.11) is called the *forward difference* approximation of $\frac{d}{dx} f$ in space.

To simplify the notation f_i will be used in place of $f(x_i)$, and if the function depends on time as well as on space the notation f_i^ℓ will be used in place of $f(x_i, t_\ell)$.

If the function f is developed using Taylor's polynomial for $x - h$ and the same process used to obtain equation (A.10) is used the discretisation becomes:

$$\frac{d}{dx} f \Big|_i = \frac{f_i - f_{i-1}}{h} + O(h) \tag{A.12}$$

which is the *backward difference* approximation of $\frac{d}{dx} f$ in space.

Subtracting the Taylor's polynomial of f around $x+h$ from the Taylor's polynomial around $x - h$ one obtains the *central difference* approximation:

$$\frac{d}{dx} f \Big|_i = \frac{f_{i+1} - f_{i-1}}{2h} + O(h) \tag{A.13}$$

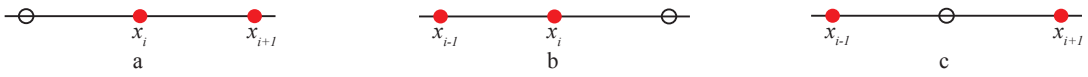


Figure A.3. a. Forward difference stencil; b. Backward difference stencil; c. Central difference stencil;

A visualisation of the part of the grid that only involves the generic nodes used to obtain the numerical solution of a PDE is called a *stencil*. Figure A.3 shows samples for forward, backward and central differences in one dimension.

Desirable properties for a numeric method in general, and for a FDM in particular, are *consistency*, *stability* and *convergence*.
Consistency refers to the fact that the truncation error approaches zero as h tends to 0. Stability means that the numerical solution is not affected by rounding-off or fluctuation in the initial data, if the round-off errors are present at some stage or the computation the method is stable if they shrink or stay the same in the following steps. Convergence means that at the limit the numeric solution converges to the real solution of the PDE.
It can be proved that if a method is consistent and stable then it is also convergent.

If the PDE depends on the time derivative the process of discretisation can result in two different methods depending on which FDM for the time derivatives is used. When the time derivative is considered it is usually the case that the numerical solution at the time step $\ell + 1$ has to be approximated using the known its values at the time steps ℓ and $\ell - 1$.
Forward differences are straightforward to apply in this situation and result in an *explicit method*, were the value of the numerical solution at time $\ell + 1$ can be computed explicitly.
Applying backward or central differences instead poses the problem of not knowing the values of the numerical solution at successive time steps. This results in an *implicit method* constituted by a system of linear equations that can be solved to

obtain the value of the numerical solution at time $\ell + 1$.

Explicit and implicit methods presents both advantages and disadvantages. Explicit methods are easy to implement but the time steps have to be kept small in order to maintain numerical stability. On the other hand, implicit methods are unconditionally stable, but they are more demanding in terms of computations and are more complex to implement.
Second order derivatives can be obtained again using Taylor's polynomial and using a backward difference to discretise the first order derivative:

$$\left.\frac{d^2}{dx^2} f\right|_i = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} \tag{A.14}$$

A.3.1. Discretization of the gradient operator. The gradient operator ∇ is the vector composed by the partial derivatives along each spatial axis. In $\mathbb{R}^2$ it is defined as $\nabla = (\partial_x, \partial_y)$. To discretise this operator in two dimension let's consider a two dimensional regular grid and a stencil such as the one presented in figure A.4.

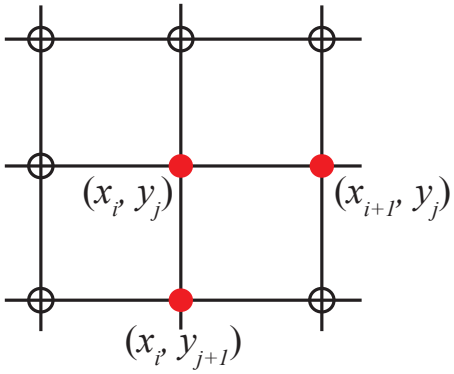


Figure A.4. Stencil for partial derivatives discretisation using forward differences in two dimensions

In this setting the partial derivatives along each axis can be discretised using one of the methods seen in the previous section. Using forward differences along the x axis will yield:

$$\partial_x f|_{ij} = \frac{f_{i+1j} - f_{ij}}{h} + O(h) \tag{A.15}$$

where the notation f_{ij} is a shorthand for $f(x_i, y_j)$.
While along the y axis:

$$\partial_y f|_{ij} = \frac{f_{ij+1} - f_{ij}}{h} + O(h) \tag{A.16}$$

The forward differences discrete gradient, denoted by $\bar{\nabla}$, is then:

$$\bar{\nabla} = \left(\frac{f_{ij+1} - f_{ij}}{h} + O(h), \frac{f_{ij+1} - f_{ij}}{h} + O(h) \right) \tag{A.17}$$

Similar formulations can be obtained using either backward or central differences.

A.3.2. Discretization of the Laplacian operator. The Laplacian operator Δ, or ∇^2, is defined as the sum of the second order partial derivatives operators, in two dimensions is written as $\Delta = \partial_x^2 + \partial_y^2$.
Replacing the second partial derivatives with their discrete form one obtains:

$$\bar{\Delta} = \frac{f_{i+1j} + f_{ij+1} + f_{i-1j} + f_{ij-1} - 4f_{ij}}{h^2} \tag{A.18}$$

this discretisation is called a *five points* scheme for the discretisation of the Laplacian

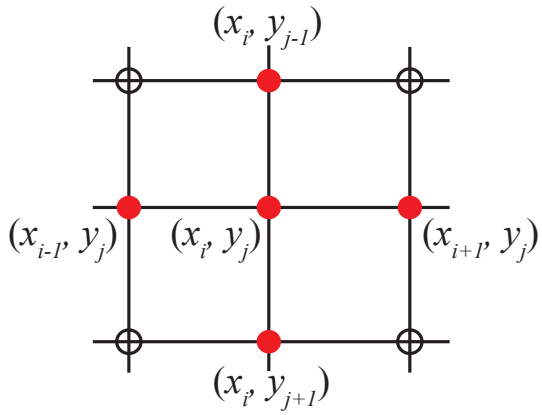


Figure A.5. Five points stencil for the discretisation of the Laplacian operator in two dimensions

operator and the stencil for it is presented in figure A.5. Other schemes are possible but in this thesis the five point scheme will be used.

APPENDIX B

Rendering and GPU computation

To render an image on screen the data describing the image have to be processed by the GPU and transformed in pixels. This process is described in section B.1 where the rendering pipeline is described stage by stage.
The GPU is a massively parallel processor capable of billion of computations per clock. This property makes the GPU attractive for high speed general purpose computations which are not necessarily related to the rendering process. To take advantage of this computation power an overview of the way a GPU works is given in this chapter. Specifically the NVIDIA® Fermi GPU architecture and its threading model is explained in section B.2.

B.1. The rendering pipeline

In this section an overview of the Microsoft's DirectX 11 rendering pipeline abstraction (Shader Model 5.0) is given with a detailed explanation of each of its stages and how they work. For more details on the rendering pipeline see [**114, 115, 116**].

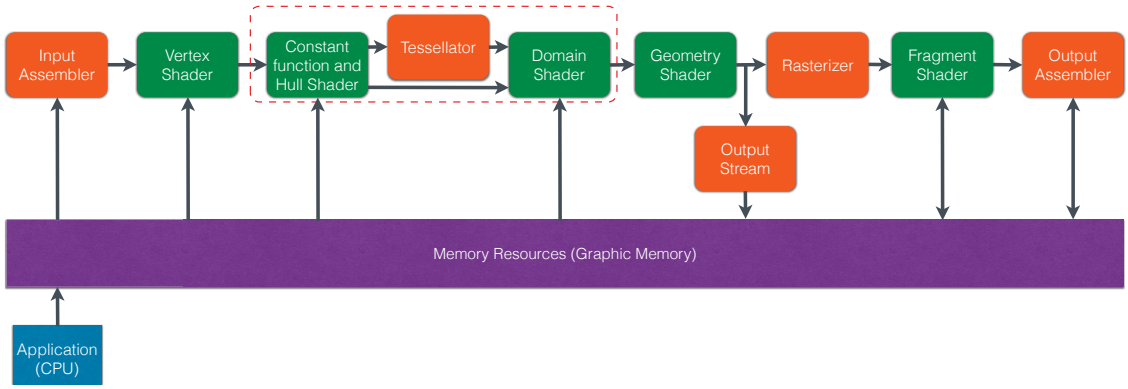


Figure B.1. The rendering pipeline abstraction (Shader Model 5.0)

Data to be rendered are usually organized in data structures called *vertices*. A vertex possesses different attributes which determine the way it will generate geometry and appear on screen. Examples of these vertex attributes, which are commonly used, are its position in local space, the texture coordinates, the normal, the skinning weights and a colour information. As shown in figure B.1 vertices forming the mesh to be rendered are transferred from main memory to the graphics card memory. These data are then picked up by the *Input Assembler* which start their processing though the pipeline.

B.1.1. Input Assembler. This mandatory configurable (but not programmable) stage receives vertices from the application. The input assembler organizes the vertices into geometric primitives so that subsequent stages can work with them. In order to transform vertices into geometry primitives such as triangles, points or lines, the input assembler receives from the application two pieces of information: a data structure which contains the vertex layout, i.e. a description of the vertex data structure, and the topology of the data, which describes how the input assembler is meant to connect the data geometrically. The Input Assembler uses the input layout to attach to each piece of data in the vertex data structure a label, called *semantic*, to it. The semantic is used in subsequent stages to identify what each piece of data is supposed to represent.

At this point the input assembler transformed the input vertices into geometric primitives. However, these geometric primitives live still in the object reference system, also known as local space. Although this space is convenient for defining the geometry it is not the best choice to build the whole scene to render nor to perform the rendering itself. The transformations necessary to move the geometry from local space to a more convenient reference system for rendering are usually performed by the *Vertex Shader*, unless some of the optional stages are active.

B.1.2. Vertex Shader. This programmable stage is mandatory for the rendering pipeline. The Vertex Shader works on one vertex at the time, applying geometric transformations to it whenever necessary. For example transforming the vertex from local space to clip space through the use of the world, view and projection matrices. The vertex shader is also used to perform the weighting of the vertices for skinning operations.

Geometric transformations are not always applied during the Vertex Shader stage of the pipeline. If the following stage is one of the optional stages, tessellation or geometry shaders, the geometric transformation may be delayed and applied by these stages. In this case the Vertex Shader simply passes the data trough. To the next stage that can be either the *Tessellation sub-pipeline*, the *Geometry Shader* or the *Rasteriser*. If the data move to the Tessellation sub-pipeline they can not be sent to directly to the Geometry Shader, as the Tessellation sub-pipeline make use of it to increase the geometry resolution. Vice-versa if they are sent to the Geometry Shader they can not be used in the Tessellation sub-pipeline as it sits upstream with respect to this stage. However it is possible to make use of the *Output Streams* to reinsert the geometry generated by the Geometry Shader into the pipeline after the Input Assembler stage, allowing the geometry to be tessellated in a second pass. However this process affect the rendering performance. The Tessellation sub-pipeline and the Geometry Shader are explained in the following.

B.1.3. Tessellation sub-pipeline. The tessellation sub pipeline is optional and composed of three stages: the Constant function and Hull Shader, both programmable, the Tessellator, configurable but not programmable, and, finally, the Domain Shader, programmable. The outputs from the tessellation sub-pipeline are fed to the Geometry Shader for creating the extra geometry defined by the tessellation.
Main purpose of the tessellation sub-pipeline is to increase the number of vertices on demand for displacement operations. Thus, greatly increasing the detail of meshes without having to use a large amount of bandwidth to pass data from host memory (CPU side) to device memory (GPU side). The stages of the tessellation sub-pipeline are described in the following.

B.1.3.1. *Constant function and Hull Shader.* This stage of the sub-pipeline requires two user defined functions, the Constant function and the Hull Shader. The Constant function is executed once for each patch and determines how the tessellated patches will fit together defining the amount of tessellation for the patch edges and the internal tessellation of the patch.
The Hull Shader, in addition to passing through the values output of the Vertex Shader, determines the kind of tessellation, integer or fractional, the topology of the patch, and how many control points will be generated per patch.

B.1.3.2. *Tessellator.* This stage is fixed and configured through the constant function. In fact, the Tessellator determine the amount of tessellation for the patch based on the tessellation factors for the edges and the interior of the patch set in the Constant function. This stage output to the Domain Shader a set of weights based on the topology defined in the hull shader.

B.1.3.3. *Domain Shader.* The Domain Shader put together the controls points defined in the Hull Shader and the weights output of the Tessellator to create a set of new vertices for each patch. These new vertices are then transformed to clip space before they are passed to the next stage of the rendering pipeline.

B.1.4. Geometry Shader and Output Streams. These stages are optional the Geometry Shader is programmable and is capable to generate geometry from positions. The Output Stream is configurable and can be used to send back the geometry generated by the Geometry Shader to the Input Assembler for a second pass through the rendering pipeline. Main use of the Geometry Shader is for particle systems.
The next step in the pipeline is the Rasteriser which takes the geometry in output from either the Vertex Shader, the Tessellation sub-pipeline or the Geometry Shader and transforms it in *fragments*.

B.1.5. Rasteriser. The Rasteriser is mandatory and configurable (but not programmable). It is in charge of: transforming the geometry into fragments, which may or may not end up as pixel on the screen, vertex attributes are interpolated

and degenerate geometry is removed. This stage also performs: clipping against the frustum, perspective correction and primitive mapping to view-port.

Once the geometry has been fragmented and the vertices attributes interpolated and assigned to each fragment the colour of the fragments are determined by the *Fragment Shader*.

B.1.6. Fragment Shader. The Fragment (or Pixel) Shader is a mandatory stage of the rendering pipeline in charge of determining the colour of each fragment. This stage is programmable and common uses for it are texturing and lighting operations and other visual effects. This stage can be used to perform general purposes computations if needed. However, this use is out of date as the recent introduction of the computation pipeline make easier and more natural to perform highly parallel computations on a GPU.

With all the fragments colored and lit all that remain to do is to determine which are visible and organize them in the final image to display on screen. This job belongs to the *Output Merger*.

B.1.7. Output Merger. This stage is again mandatory and configurable (but not programmable) it is charge of fill the depth (or Z) buffer, which is used to determine which fragment is visible in the final image. The output merger is also in charge of filling the frame buffer (final image) with the fragments survived from the Z-tests.

B.2. The computation pipeline

In this section the compute pipeline is presented. This section will be used in section 4.1 Terrain Deformations.

B.2.1. Threading Model. The computation pipeline is based on the Single Program Multiple Data (SPMD) paradigm, by which multiple copies of the same program, also known as *kernel*, run in parallel on different *threads* using different input data. This model is designed to fit the *divide et conquer* paradigm, allowing the programmer to focus on generating a data model which reduce the problem under study into a series instances of the same smaller problem.

As shown in figure B.2 threads are organised on a three dimensional grid and clustered in *groups of threads*. These groups are in turn organised on a three-dimensional grid, called a *dispatch*. To each tread is assigned a three-dimensional index that uniquely identify its position in the group of threads, and each group of threads is in turn uniquely identified in the dispatch by a set of three indices. With these two set of indices it is easy to compute the global indices for a thread in the

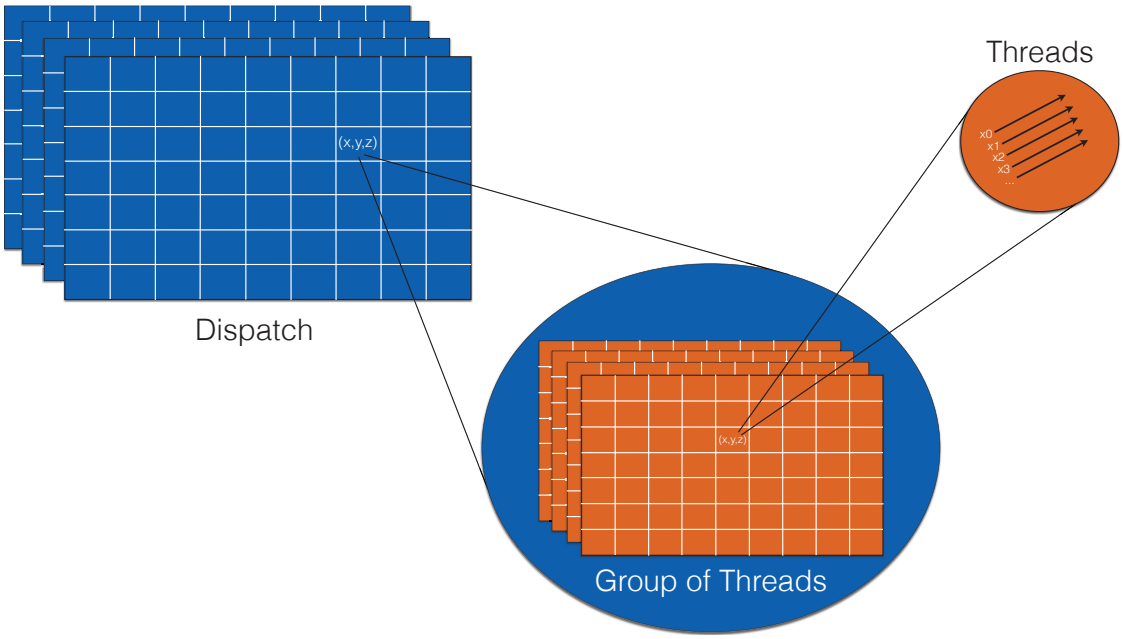


Figure B.2. Diagram showing the GPU threading model. Threads are clustered in groups of thread organized on a three-dimensional grid. Each of these groups of threads grids are in turn organized on a three-dimensional grid called a dispatch

dispatch. This structure makes easy to associate threads to data as to do so the only requirement is to match the indices of a thread and its input data.

B.2.2. Memory hierarchy. In the computation pipeline three memory levels are available. The fastest access memory, but limited in size (on the order of KB), is the on chip memory. This memory can be spilt in L1 cache and in shared memory depending on the program requirement. Shared memory is a resource pool created for each group of threads, only threads within the same group can access it. The next kind of memory available is the unified L2 memory cache, which is used for load, store and texture operations. Finally, global memory is the slowest access memory although it is big in size (on the order of GB).

The amount of memory and the number of cores available depends on the specific architecture of the GPU and on the graphics card model.

B.2.3. GPU Architecture. The software presented in this thesis is developed on a NVIDIA® Fermi architecture, namely on a NVIDIA® Quadro® 2000, and a short review of this architecture will be presented in this section.
The NVIDIA® Fermi GPU [**117**] is organized in 16 Streaming Multiprocessors (SMs) arranged around a common L2 memory cache. A SM is composed by 32 CUDA cores organized in two groups of 16 cores, 16 load/storage units and 4 Special Function Unit (SFU), capable of computing transcendental functions.
A CUDA core, composed by an Arithmetic and Logic Unit (ALU) and a Floating

Point Unit (FPU), is capable to execute up to two integer or floating points operations per clock per thread or a single double operation per clock per thread.
The GPU communicates with the CPU through an Host Interface (PCI-Express). A global scheduler, called GigaThread scheduler, distributes groups of 32 threads, called *warps*, to the SM *Dual Warp* schedulers. *Dual Warp* schedulers allow to dispatch instructions from one or two independent warps to two of the four components of the SM simultaneously.
The GPU is interfaced with six 64 bits memory partitions which support up to 6 GB GDDR5 DRAM.
The NVIDIA® Fermi architecture feature 64KB RAM of configurable on chip memory, compliance with IEEE754-2008 standard for 32-bit floating point or 64-bit double precision [**117**].

APPENDIX C

Colour Plates

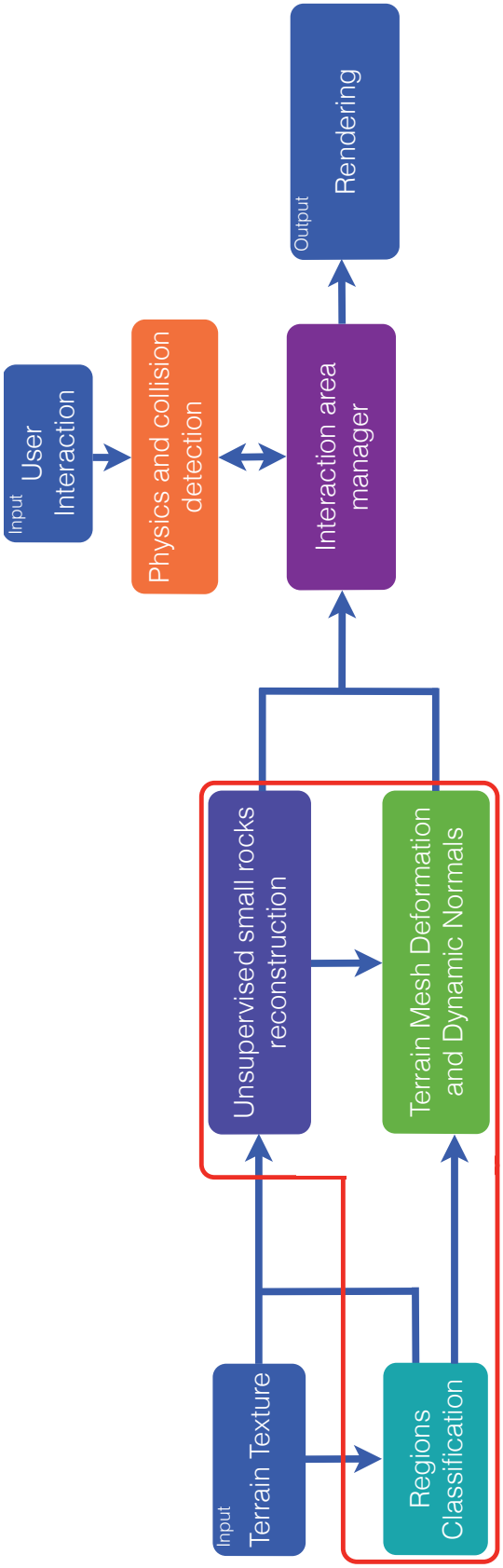


Figure C.1. Larger version of figure 1.5. Overview of the framework. The region classification, unsupervised small rocks reconstruction and the terrain mesh deformation and dynamic normals, highlighted in the red box, are the three fundamental components that are developed in this thesis.

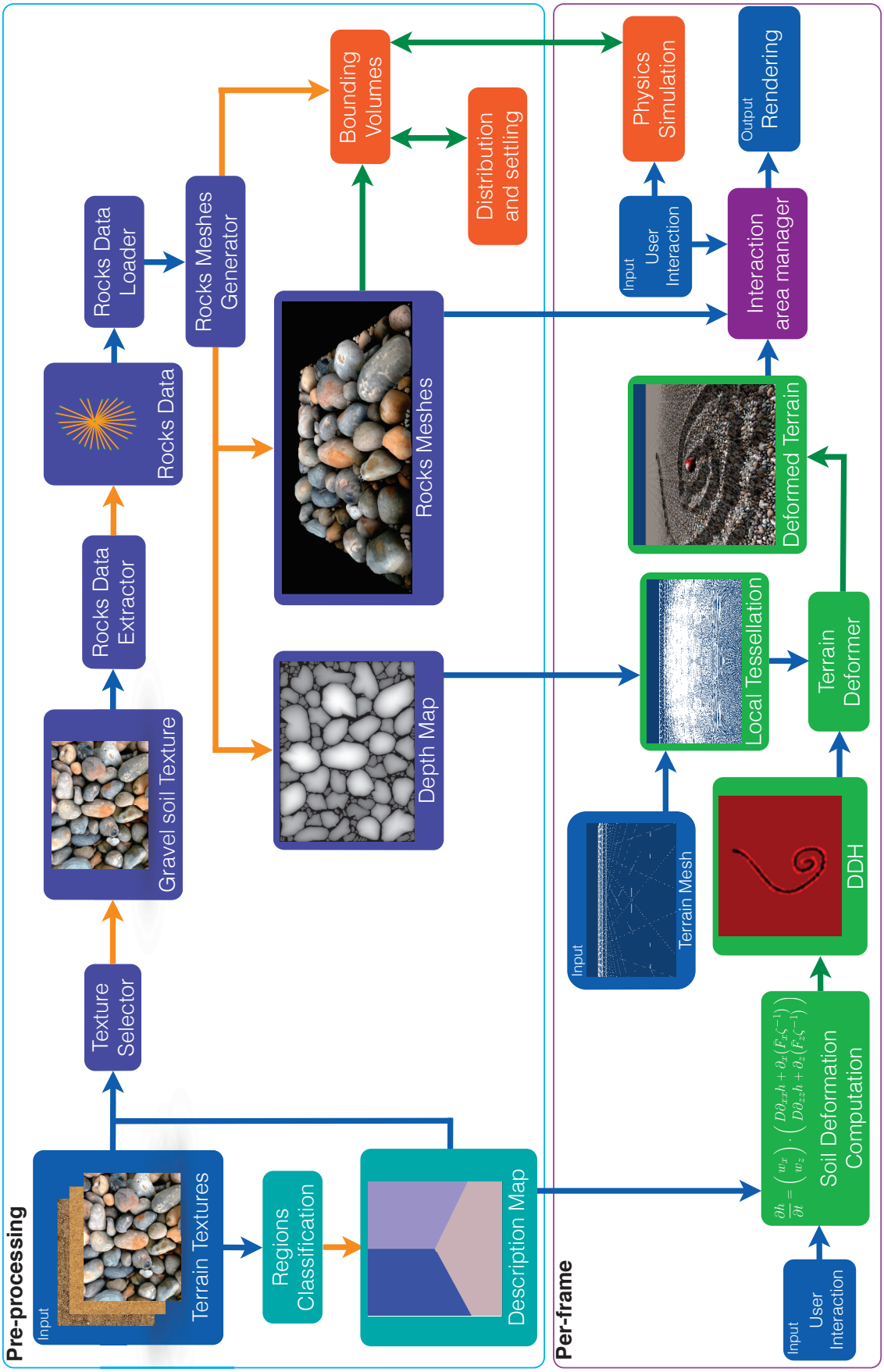


Figure C.2. Larger version of figure 1.6. More detailed diagram of the framework. Blue arrows represents inputs, orange arrows represents outputs, green arrows represents updates.

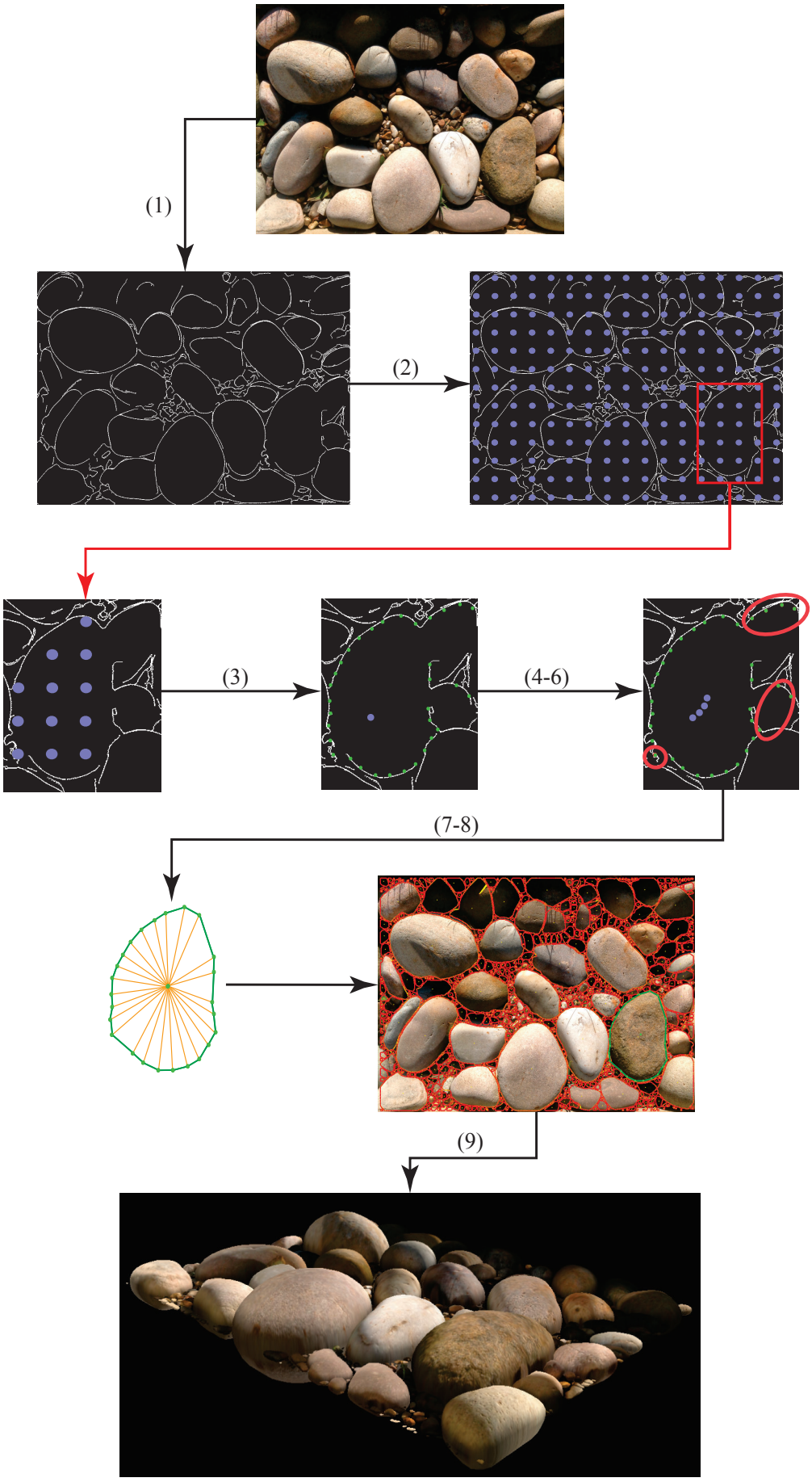


Figure C.3. Larger version of figure 3.5. Rock models extraction from a single image.

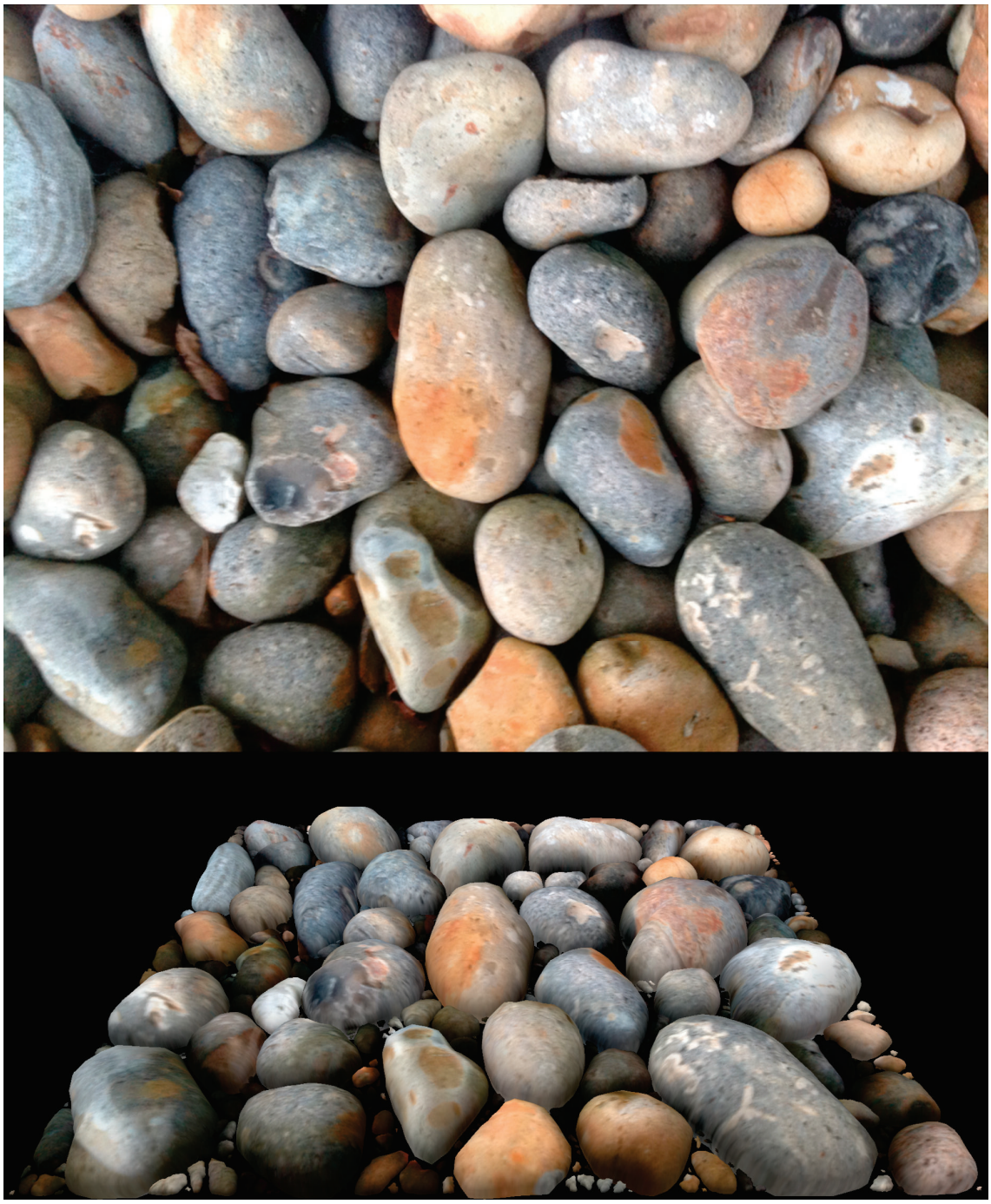


Figure C.4. Larger version of the top image in figure 3.17. Results obtained applying the method described in sections 3.1 and 3.2 to different images of heterogeneous rocks (in sizes and textures) under different lighting conditions.

Figure C.5. Larger version of the middle image in figure 3.17. Results obtained applying the method described in sections 3.1 and 3.2 to different images of heterogeneous rocks (in sizes and textures) under different lighting conditions.

Figure C.6. Larger version of the bottom image in figure 3.17. Results obtained applying the method described in sections 3.1 and 3.2 to different images of heterogeneous rocks (in sizes and textures) under different lighting conditions.

Figure C.7. Larger version of figure 3.18. Reconstructed small rocks rendered without texturing to show the reconstructed shapes.

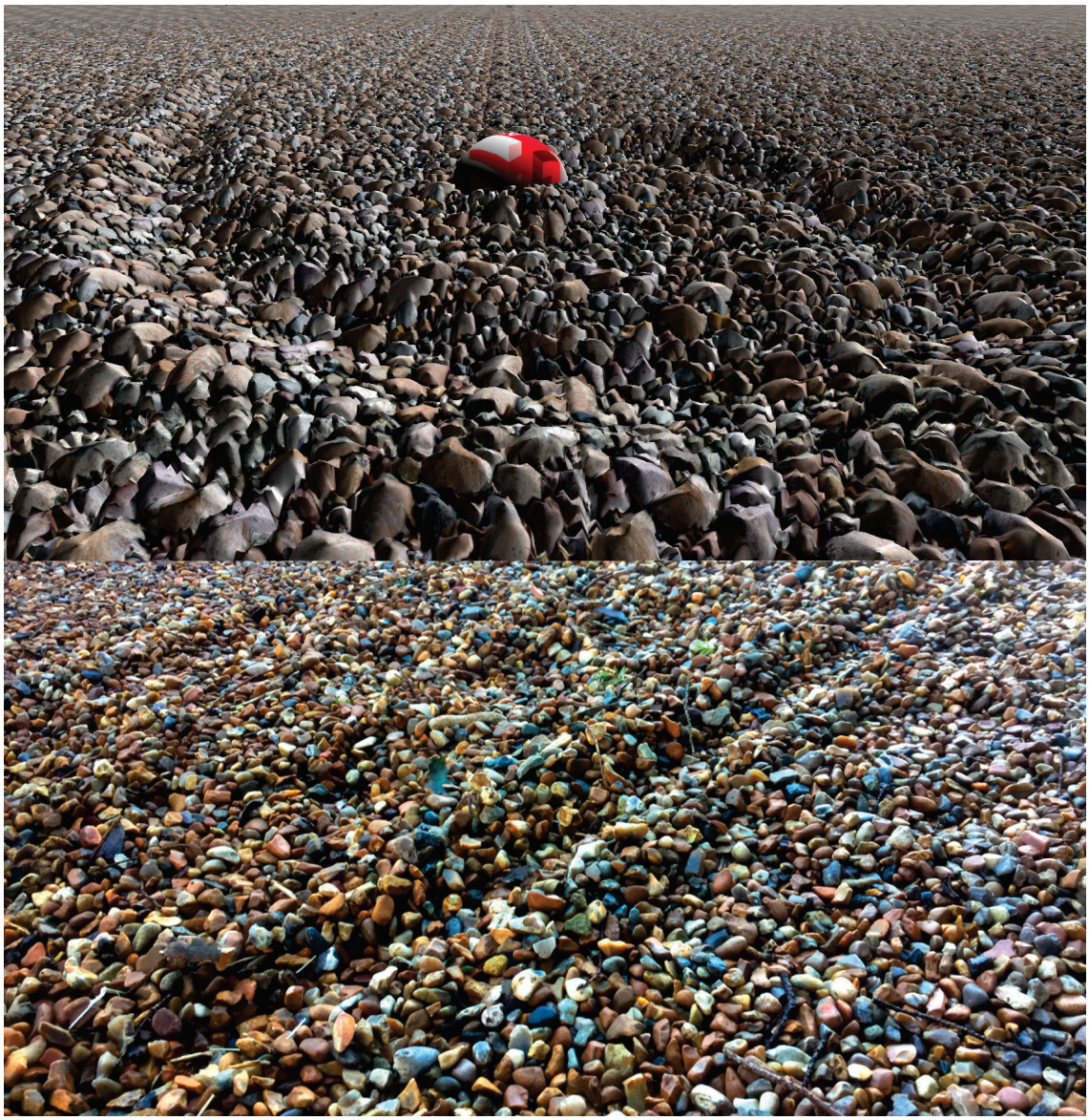


Figure C.8. Larger version of figure 4.9a. Results obtained with the method presented in chapter 4: comparison between tracks on simulated pebbles and tracks on a real pebbles terrain

Figure C.9. Larger version of figure 4.9b. Results obtained with the method presented in chapter 4 comparison between tracks on simulated mud and tracks on a real muddy terrain

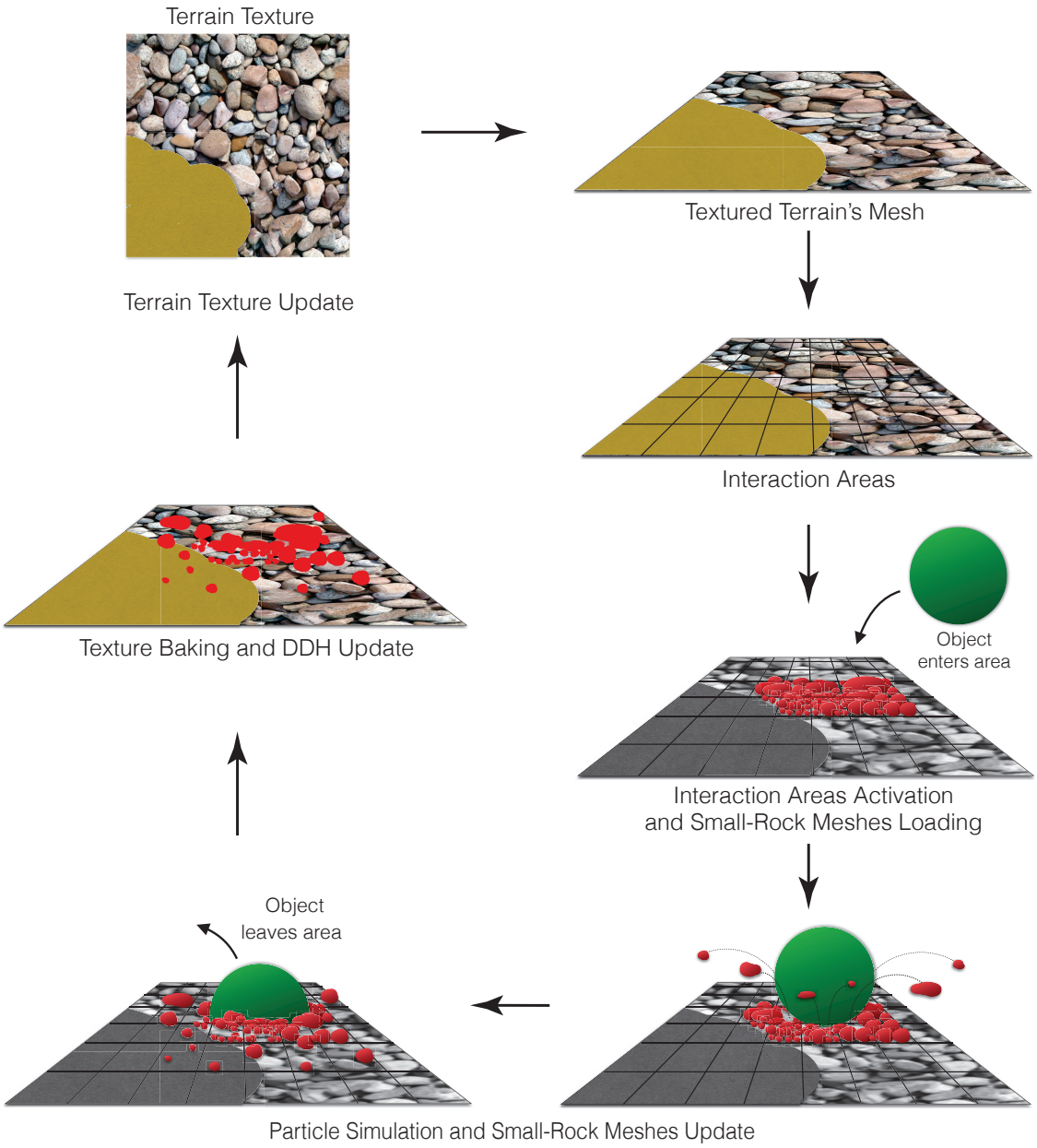


Figure C.10. Larger version of figure 5.5. Diagram showing the design of the interaction manager component. Rocks meshes are highlighted in red to better distinguish them from the terrain texture.