FFI-rapport 2011/01216

Interactive Terrain Editor (ITED)

Martin Ferstad Aasen and Lars Hofsøy Breivik

Norwegian Defence Research Establishment (FFI)

27 June 2011

FFI-rapport 2011/001216

1140

P: ISBN 978-82-464-1937-4
E: ISBN 978-82-464-1938-1

Keywords

Modellering og simulering

Grafisk databehandling

Databehandling – Visuelle metoder

Databehandling – Interaktive metoder

Approved by

Karsten Bråthen	Project Manager
Eli Winjum	Director of Research
Vidar S. Andersen	Director

English summary

In military Modelling and Simulation (M&S) the synthetic natural environment (SNE) is a very important component. The first and most obvious aspect is the visual quality a high detail, high resolution SNE makes achievable. Other, but not less important aspects are concerned with the interaction between simulation components, human actors and the SNE.

There has been a gap in the tool chain for design and modification of elevation data for use in M&S. Most available tools either offer low-level editing functionality that is hard to use for sculpting realistic looking terrains, especially if the terrains need to be of a certain size, or they offer high level design mechanisms that are hard to tune and in general unable to produce realistic terrains. This is more valid for geo-typical than geo-specific terrains, but also the latter kind may need editing and enhancements.

This report discusses the field of elevation data generation and modification and presents the FFI developed tool Interactive Terrain Editor (ITED) that aims to fill the gap in the available tool chain for elevation data processing.

Sammendrag

I militær modellering og simulering (M&S) er det syntetiske, naturlige og menneskeskapte miljøet (SNE) en veldig viktig komponent. Det mest iøyenfallende aspektet er den visuelle kvaliteten et detaljrikt SNE med høy oppløsning gjør mulig. Andre, men ikke mindre viktige aspekter er tilknyttet simulerte komponenters og menneskelige spilleres interaksjoner med miljøet.

Det har hittil vært et hull i det verktøysettet som har vært tilgjengelig for design og modifikasjon av høydedata for bruk i M&S. Det fleste tilgjengelige verktøy tilbyr enten lav-nivå editeringsoperasjoner som er vanskelige å bruke for å lage terreng med realistisk utseende, spesielt hvis terrengene er av en viss størrelse, eller så tilbyr de høy-nivå mekanismer som er vanskelige å konfigurere og generelt sett ikke i stand til å produsere realistiske terrreng. Dette er mer gyldig for geotypiske enn geospesifikke terreng, men også høydedatasett av den sistnevnte typen kan trenge bearbeiding og forbedringer.

I denne rapporten diskuteres eksisterende metoder og verktøy for generering og bearbeiding av høydedata. I tillegg presenteres verktøyet Interactive Terrain Editor (ITED). ITED utvikles ved FFI og bidrar til å tette det nevnte hullet i verktøysettet for bearbeiding av høydedata.

Contents

1 Introduction

In military Modelling and Simulation (M&S) the synthetic natural environment (SNE) is a very important component. The first and most obvious aspect is the visual quality a high detail, high resolution SNE makes achievable. Other, but not less important aspects are concerned with the interaction between simulation components, human actors and the SNE. E.g. a soldier, controlled by an Artificial Intelligence (AI) component or a human in the loop (HITL), can make use of the terrain for cover when advancing through it. In the same way, the quality of the SNE is an important factor when a soldier tries to detect an enemy who is hiding. Simulations of sensors and vehicles are also dependent on the environment. A high fidelity ground vehicle simulation is highly dependent on the quality of the SNE. Two representations of the same real area might give completely different results, because the lower resolution representation does not contain the information that in fact makes the area impassable. A sensor's ability to detect and identify an object positioned in a terrain can also be highly dependent on the terrain representation, especially the encoding of the different material types different parts of the SNE consist of. Because of these strong connections between the simulations and the environment, it is important that the SNE is as realistic as possible.

Elevation data is an important part of an SNE and synthesis of elevation data has long been an active field of research (an SNE does not need to be geo-spesific). Algorithms based on noise functions, stochastic and fractal theory have been designed and implemented to aid in development of geo-typical elevation data. This set of techniques is called procedural techniques. The use of procedural terrains for military M&S and games is not widespread. One reason for this might be that the mathematical nature of the algorithms has not yet been properly separated from the user interface [1], making tools unintuitive and hard to use.

Interactive Terrain Editor (ITED) is an application currently under development by FFI. ITED was initially designed as a tool for interactive editing of elevation data, and as a framework for rapid prototyping of new techniques for elevation data processing. Interactive processing rates are achieved by utilizing the graphics processing unit (GPU) both for visualization and processing of elevation data. Functionality that was foreseen implemented in ITED includes the ability to add high resolution, realistic details to lower resolution elevation data; as well as editing of elevation data to shape a terrain to meet specific needs. One early idea was a cut and paste function that would allow a user to compose a new terrain from parts of existing elevation data. This idea has later evolved into an implementation of an interactive and completely user controlled blending technique for elevation data. This functionality is probably the most useful part of ITED at the time being. This report gives an overview of techniques to generate and edit the elevation data component of SNEs, as well as a thorough introduction to ITED.

2 Motivation

This section explains what motivated the development of ITED. The main factors are the NATO Research and Technology Organization (RTO) Modeling and Simulation Group Task Group MSG-071 Missionland: A Universal, Generic, Multi-Spectral Simulation Environment Database and a general desire to be able to enhance the low resolution data that is most often available to M&S users.

2.1 Enhancement of existing data

Available geo-specific elevation data is often of a resolution of 20 m or more. This can be sufficient for training high or even low-level air operations, but can limit the possibility of tactical use of the terrain in ground based training. In many real areas, for example in Afghanistan or Norway, the terrain has structures which allow soldiers on foot, or even vehicles, to move in cover of the terrain.

One possible approach to obtaining data with higher resolutions is to modulate existing low resolution data with higher frequency information. In [2] this approach is used with real-world data in the high resolution component, in an attempt to: “replace procedural and erosion synthesis parameters with use of an example terrain to make it easier to synthesize realistic, heterogeneous terrain”. ITED can be used as a tool to aid the development of such techniques.

2.2 Fictitious landscapes with specific characteristics

There are several reasons why copies of real world areas can be unsuited for use in military synthetic training. One reason can be that one or more nations involved can find the available areas hard to use due to political relations and/or history involving the areas. It can also be favourable to have a training scenario with a composition of terrain types and characteristics not easily found in real areas of a preferred size. This argument for why the ability to design artificial areas is beneficial is used in [3]. ITED provides good support for development of fictitious landscapes.

2.3 Missionland

MSG-071 - Missionland has an objective to develop a dataset with environment data for a fictitious continent located in the middle of the Atlantic Ocean. The before mentioned political issues, sometimes involved in military synthetic training involving several nations, is one reason why the group was formed. Another is the need for available data to support the diversity of M&S needs in NATO. With Missionland, research groups within NATO no longer need to rely on members of the group to provide data suitable for all involved parties.

The wide scope of possible usage scenarios puts a requirement on the Missionland dataset to cover many different terrain types and characteristics, as well as the combination of huge landmasses and high resolution areas. The size of the continent is set to be roughly 2000 km by 2000 km.

Furthermore, the elevation data in the current version of the Missionland dataset has a full coverage of data with resolution of about 30 m. Selected areas will be enhanced with higher resolution elevation data. These specifications result in roughly 65 000 by 65 000 data points of elevation data for the 30 m set. The size of the dataset puts strain on the tools used for generation and editing the data.

MSG-071 has tested some tools for procedurally generating terrain. The results from the experimentation with these tools align well with the impression obtained from reading academic works in the field of terrain synthesis: procedurally terrains do not yet possess the same heterogeneous characteristics found in natural terrains. There are tools capable of producing quite realistic looking elevation data from a ground perspective and even in a low or medium level flight. However, where all procedural algorithms known to MSG-071 fail is the introduction of larger terrain features like long systems of valleys and river networks, etc. The statistical nature of procedural algorithms result in landscapes where one area is similar to the next and where global trends are missing. This is illustrated in figure 2.1, where a procedurally generated terrain and a real terrain are compared. Both terrains are 100 km by 100 km in size.

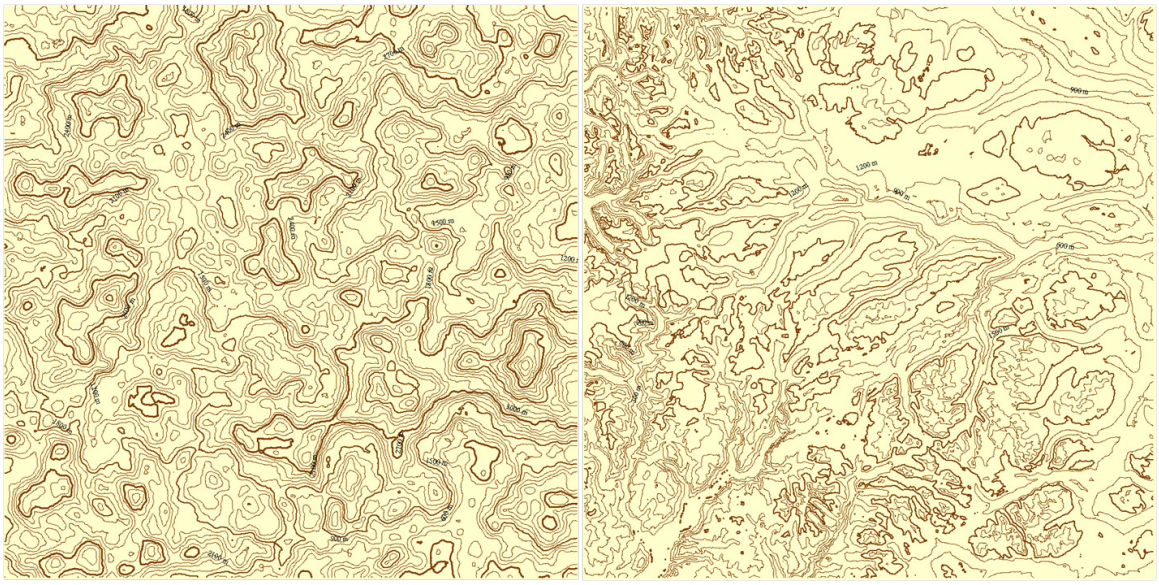

Figure 2.1 Elevation contours of a procedural terrain generated by the use of Large 3D Terrain Generator (L3DT) [4] to the left and a geo-spesific, Norwegian terrain delivered by "Norge digitalt" [5] to the right. The contours and visualizations have been produced by Global Mapper [6].

A terrain intended for use in military M&S have other requirements than a terrain intended for use in a first person shooter (FPS) game. In an average FPS game the playable area is much smaller than what is often needed for many military operations. There are also other requirements for navigation. An FPS player normally does not need to navigate through the terrain using a map and compass or perform a map analysis to establish a navigational route for heavy machinery. Maps of the Missionland continent is an important requirement established by potential users of the dataset

[7]. The height contours of a large area of procedurally generated terrain do not necessarily look like the height contours of a real world area. An example of this is shown in figure 2.1. This is not ideal, as synthetic training should provide as realistic training conditions as possible. However, some real-world areas might actually look like the procedural terrain on the left, but in general that is not the case.

ITED is regarded by MSG-071 as the most promising tool that can support development of the elevation dataset. The blending functionality in ITED is key in this respect. Blending of elevation data will be discussed further in sections 3.3, 6.3 and 6.4.4. The use of ITED in MSG-071 is described in more detail in section 7.2.

3 Prior Work

There are many available tools for the purpose of generating synthetic terrain. However, few or none of these are easy to use and powerful (fast) enough to be used for creation of data for large areas, while providing a high degree of user control. A common characteristic of most tools is that they are controlled through a number of parameters. Because the parameters seldom have intuitive effects, and because the generators runs in less than interactive rates, it is hard for a user to tune the parameters to best fit the terrain that he or she imagined [1].The control mechanisms available to a user more often have a global than local effect. In other words: It is hard to generate a terrain with a specific design and composition of terrain types, etc. Production of elevation data can be conducted in two ways: manual editing or generation. These two approaches are both relevant in the context of ITED.

There is an ongoing work at TNO in cooperation with Delft University in the Netherlands concerned with sketch-based editing. They have developed a prototype tool that gives a user control of procedural terrain generators through intuitive user interfaces [8]. One of the members of the group has written an interesting master thesis [9] which is perhaps more relevant in the context of ITED and the functionality implemented in it than the rest of the group's publications are. The thesis describes a work conducted to improve the tools and techniques currently available to game level designers. Some procedural tools (e.g. TerraGen and World Machine), as well as editors more focused on game level editing (e.g. CryENGINE Sandbox and UnrealEd 3), were reviewed. The thesis points at a gap in the editing techniques between global, high-level design (generation) and very low-level brush-based editing with simple brushes. The author of the thesis has also published a paper based on the thesis [10]. The paper describes the implementation of procedural height field brushes in a tool that in many ways are quite similar to ITED.

3.1 Editing

The authors of this report has not been able to find an editor, free or proprietary, that can provide interactive editing of elevation datasets of some size (larger than 4096×4096). Neither have the

other members of MSG-071.

It may seem that commercial games have quite low capabilities when it comes to elevation data resolution and size. According to the official support cite [11] of the Unreal Engine, 1024 by 1024 data points is the recommended height map size [12]. The source also states that at 2048 by 2048 data points the editor is slow and hard to work with. The CryENGINE Sandbox 2 editor has a capability of 8192 by 8192 elevation data point, which is considerable higher. However, both these editors have only got quite primitive elevation data editing tools.

Bryce, which is a professional 3D landscaping and animation tool from DAZ 3D [13], only supports 4096 by 4096 elevation data points. Bryce has got the ability to generate fractal landscapes and apply erosion simulation to them. Manual editing through brushes is also available, but the brushes do not seem to provide good means of creating realistic looking terrains. In the CryENGINE Sandbox 2 manual [14], in a section that describes Sandbox 2's abilities to generate elevation data, the following is stated: "There are no erosion features so adjusting a texture in photoshop or other packages such as bryce3d is highly recommended". All-in-all it seems that these tools have not got good abilities to create realistic elevation data for use in M&S from scratch. In video tutorials for Sandbox, which is the most impressive tool overall, more focus is put on other parts of the environment. Interactive placement of roads that automatically cut into the terrain, interactive design of buildings and vegetated areas, are however impressive sides of the Sandbox editor.

3.2 Generation

This section provides a short introduction to methods for generation of elevation data for geo-typical terrains. In [10] algorithms for generating elevation data in one single operation is split into two groups: procedural synthesis algorithms and simulation algorithms. The first group use fractals and random distributions to generate geo-typical elevation data. The second group simulates geological phenomena like different types of erosion. This report refers to the first group as procedural methods and the second group as erosion based methods. Each of these methods are described below.

3.2.1 Procedural methods

Procedural terrain algorithms are often based on fractional Brownian motion (fBm) [9]. According to [15], the connection between Brownian Motion and mountainous terrain was first discovered by B. B. Mandelbrot and reported in [16].

Fractals can be divied in two groups: deterministic and random. The von Koch snowflake and Julia sets are deterministic fractals, while Brownian motion in one variable can be regarded as the simplest random fractal [17]. A Brownian motion in one variable is a function $X(t)$ here the values of X are random. The difference between two samples $X(t_1)$ and $X(t_2)$, $X(t_1) - X(t_2)$, have a

Gaussian distribution and is expected to have a square variance, $E[X(t_1) - X(t_2)]$, proportional to the time difference, $t_2 - t_1$.

There are several algorithms that can be used to create fractal surfaces. Poisson faulting, Fourier filtering, midpoint displacement, successive random additions and summing band-limited noises are listed in [15]. Poisson faulting was used by Mandelbrot and results in an fBm. Midpoint displacement however, produces true fractal surfaces, but without the correct statistical properties to be an fBm [15].

The text in rest of this section is strongly based on excerpts from [18]. As explained in the introduction of this section: current procedural terrain elevation methods are hard to configure and control, which makes it hard to produce the intended results [19, 20]. Another drawback with current methods is that the random nature of the algorithms does not model all the structures found in nature accurately enough [15] to make the result indistinguishable from real areas. Structures that are hard to model are typically formed over many years of natural occurring phenomena like erosion. Unfortunately, implementations of erosion based methods run much more slowly than their procedural counterparts [10]. It seems that current procedural methods, and even erosion based methods, are not able to produce elevation data that really looks natural [21].

An example of the difference between real and procedural terrain is displayed in figure 3.1. Sea level is rendered in black. The elevation then goes from blue through green and yellow to red. The highest areas are rendered in white. The colour legend has been chosen individually for the two data sets to best visualize the topologies found in the two, not to compare the elevations. The highest point in the procedural terrain is considerably higher than in the real terrain. Both terrains are however 100 km by 100 km in size. The purpose of this figure is mainly to illustrate the topological differences often found when real and procedural terrains are compared.

One observation made by the MSG-071 task group (see section 2.3) is that most methods for generation of elevation data seem to focus on mountainous scenery. Generally it seems that procedural techniques are best suited to produce mountainous or hilly areas. Another observation is that most screenshots of terrains produced by procedural methods are from a low flight/ground perspective. Methods based on real-world data more often use screenshots of larger areas to show structural differences between procedural terrain and real-world data [21, 2]. This is similar to what is shown in figure 3.1. In other words: publications on procedural terrain generation provide examples of their techniques' usefulness for generating small areas, but not for areas with sizes comparable to the size of the Missionland continent.

3.2.2 Erosion based methods

Erosion simulation is an area of research concerned with the effects the environment has on the terrain, and how floods and other extreme, natural events and phenomena will affect an area. Erosion simulation is also an area of interest in the context of terrain synthesis. One reason for this is the inability of fractal noise algorithms to produce natural terrain formations like river beds,

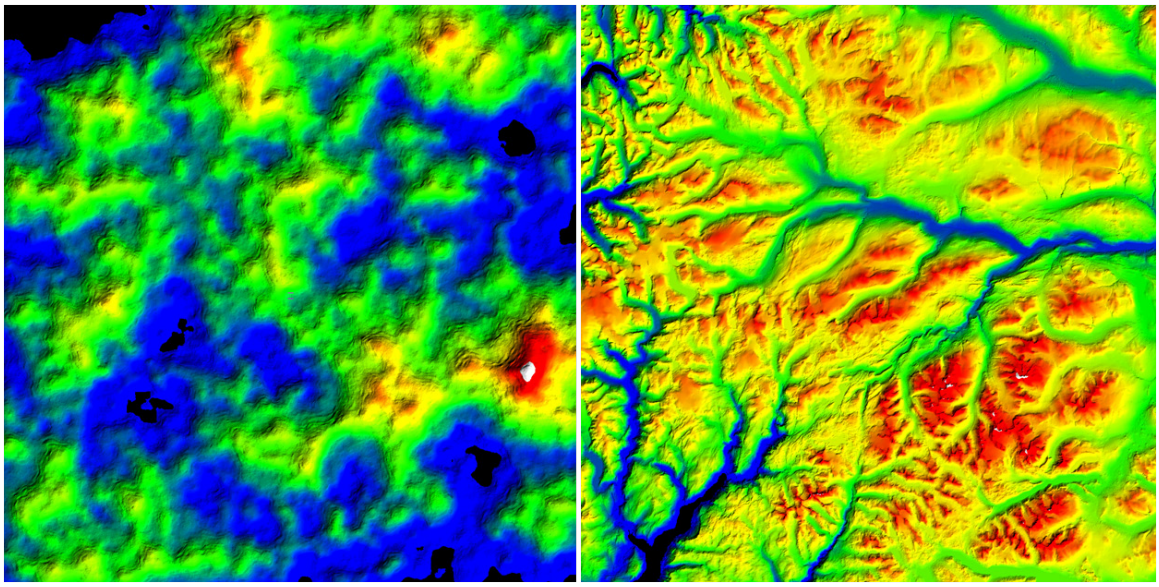

Figure 3.1 *Visualizations of procedural terrain on the left and real terrain on the right. The elevation data is the same as the data that is visualized in figure 2.1, but in this case the images are screen dumps from ITED.*

ridge systems and long connecting valleys. Erosion simulation methods are regarded as being too slow and is therefore not considered to be an effective tool in the context of the work described in [10]. The here referenced work also regards the methods to be oversimplified algorithms based on poorly understood natural processes.

Erosion simulations can be quite sophisticated, for instance using several layers of different soil types [22]. Such methods seem to produce plausible terrains of a certain size. However, erosion simulations are run on a base terrain, which to some degree dictates parts of the result. It may seem that erosion simulations more often produce plausible small features than global features; a kind of feature which seem to be missing in terrains produced by procedural terrains as well. One reason for this might be that u-valleys, the most common type of valley, are signs of the effect of glacial erosion, not hydraulic erosion. Most erosion simulations for SNEs seem to be fluvial/hydraulic simulations.

An example of glacial erosions simulation can be found in [23]. The here cited work is not aiming to produce elevation data for SNEs, but rather understand and predict the geological processes that has shaped and will continue to shape our world. However, the erosion simulation presented in the paper might be usefull if one wants to use erosion simulation in the context of SNEs. That being said, the dataset used in [23] is very small compared to what is normally needed in an SNE.

3.3 Elevation Blending

The possibility of combining different elevation datasets using techniques borrowed from image processing is accounted for in [9]. Hypotheses for how elevation data can be blended using user controlled brushes are presented. One suggestion presented is to use a “mask” dataset to control the properties of the blend in each data point. In this way one can tune the parameters stored in the mask data to find the correct blend, and even reset them to undo the blending.

In [9] the author foresees a problem with the edges around the areas that are blended, even if brushes with strength declining toward the edges of the brush are used. Large differences in the mean elevation in two blended areas are pointed out as a complicating factor. One presented possible approach is to subtract the mean elevation from each dataset, blend the result and then add the original mean elevation.

In [20] a semi-automatic process for generation of elevation data based on a library of real-world data is presented. The process is based on blending elevation data using a genetic algorithm. A user sketches a world by specifying the terrain type in different areas. Samples from the real-world data library are then selected based on the terrain type and pieced together to match the sketch. Blending of elevation data is performed at boundaries between samples, to insure smooth transisions from one sample or region to another.

As [9] represent the work closest to what is done with ITED, the authors of this report feels fairly confident that no others have implemented and tried the blending algorithm implemented in ITED in a similar fashion. The foreseen problem with edges is not a problem in ITED, because the blending is performed incrementally in small steps, allowing the user to move the brush between each of them.

4 Graphics Processing Unit (GPU)

This section provides background information on Graphics Processing Units (GPUs) and how their parallel architecture can be exploited to speed up calculations not directly connected to computer graphics.

4.1 Architecture

This section discusses GPU architecture based on NVIDIA’s new generation GPU architecture, the Fermi architecture. The Fermi architecture is designed with more emphasis on General Purpose GPU Programming (GPGPU) performance than previous NVIDIA architectures. In GPGPU applications the computational power of GPUs is utilized to perform calculations not necessarily related to computer graphics. This will be explained in more detail in section 4.2.

In [24] the real-time rendering pipeline is explained to have three conceptual stages: application, geometry and rasterizing. The border between operations performed by a CPU and a GPU has traditionally run between the application and the geometry stages. Now some of the higher level application stages like collision detection can be performed by the GPU. In the application stage a computer program configures the rendering of three dimensional, geometric objects using a graphics Application Programming Interface (API) like OpenGL [25] or Direct3D [26]. The objects pass through the rest of the pipeline, where different operations are performed on them based on the configuration setup in the application stage. The end of an object's travel is, simplified speaking, a set of pixels on a computer screen.

What makes rendering of complex scenes run in real-time (more than 30 frames per second) is amongst other things the parallel architecture of the GPU. Modern graphics cards have more than 400 processor cores [27], compared to eight for workstation CPUs [28]. Three dimensional objects passing through the pipeline are processed in parallel by these cores.

Both the CPU and the GPU are pipelined, meaning that the operations performed are divided into smaller stages that run in parallel, even if there is only one core. This is analogue to other pipelined systems like a car assembly line or ski lift. Because a GPU has a more narrow work area than a CPU, it can be more specialized, sacrificing flexibility for performance. One way this is done is by dividing the pipeline of the GPU in more and smaller stages. The performance of a pipeline is dictated by its slowest stage. Dividing this stage into several, equally loaded, stages will speed up the pipeline.

In this section the term "CUDA" is used. CUDA is an acronym for Compute Unified Device Architecture, NVIDIA's parallel computing architecture [29]. CUDA gives access to the parallel graphics hardware from applications written in conventional programming languages. ITED is based on shader programs, not CUDA. Shader programs are small programs that a graphics application can set up to run on the GPU. For more information about shader programs, please see [30] or [31].

4.1.1 Steaming Multiprocessor

The Fermi GPU is built around a group of Streaming Multiprocessors (SMs). The SM in Fermi is of second generation and has 32 CUDA cores. This is four times more than the previous generation had. The Fermi cards then have n by 32 CUDA cores available. The workstation flagship card, Quadro 6000, has 448 CUDA cores, which means it has 14 SMs. Each CUDA core has an integer arithmetic logic unit (ALU) and a floating point unit (FPU). New to the Fermi GPU is the implementation of the IEEE 754-2008 floating-point standard and the fused multiply-add (FMA) instruction for both single and double precision arithmetic [32]. The implementation of FMA improves the accuracy of floating point operations where the product of two numbers is added to a third. The improvement in precision is gained by only rounding down to the target precision once [33]. Double precision is important in many GPGPU applications.

4.1.2 Cache Hierarchy

With Fermi, NVIDIA introduced what they call a “true cache hierarchy” in addition to shared memory [32]. The Fermi GPU is equipped with a L1 cache, shared memory and a L2 cache. The L1 cache can be configured as either 48 KB of cache and 16 KB of shared memory (scratchpad memory) or vice versa. The L1 cache is per SM while the L2 is unified across all SMs.

4.1.3 Threads

When using CUDA, threads are organized in groups called blocks. A number of blocks are again organized in grids. The threads in one grid execute the same kernel, the threads in a block execute concurrently with communication, amongst other things, through shared memory. The GPU executes one or more grids and the blocks of the grids area divided amongst the SMs so a block is executed on one SM. For execution on the SM, the threads of a block is split into groups called warps, consisting of 32 threads.

It is unknown to the authors of this report how threads are organized when a graphics application using OpenGL or DirectX executes, but it is assumed that the graphics drivers manage the threads in a similar fashion.

4.2 General Purpose GPU Programming (GPGPU)

A GPGPU application can set up a graphics application using a graphics API to calculate numbers that represent something entirely different than coordinates or colours on a computer screen.

A standard graphics application does the following (simplified):

1. Transform local coordinates of 3D objects into world space coordinates to position them in a scene.
2. Project the world space coordinates of 3D objects onto a 2D grid.
3. For each element in the 2D grid, do some calculations.
4. Store results in 2D buffers with the same size as the 2D grid.

This process is visualized in figure 4.1.

For graphics applications the two most important buffers are the colour buffer and the depth buffer. The colour buffer is a 2D array storing floating point vectors of size four. This is the same data a bitmap image stores: one component for each of the colours red, green and blue, plus the alpha value, the transparency of a pixel. The depth buffer is a 2D array with the same dimensions as the colour buffer, but instead of storing a float vector with four components, it stores single floats in each location. This float represents the distance from a grid point to the corresponding point on a 3D object. A scene is processed as explained in figure 4.2.

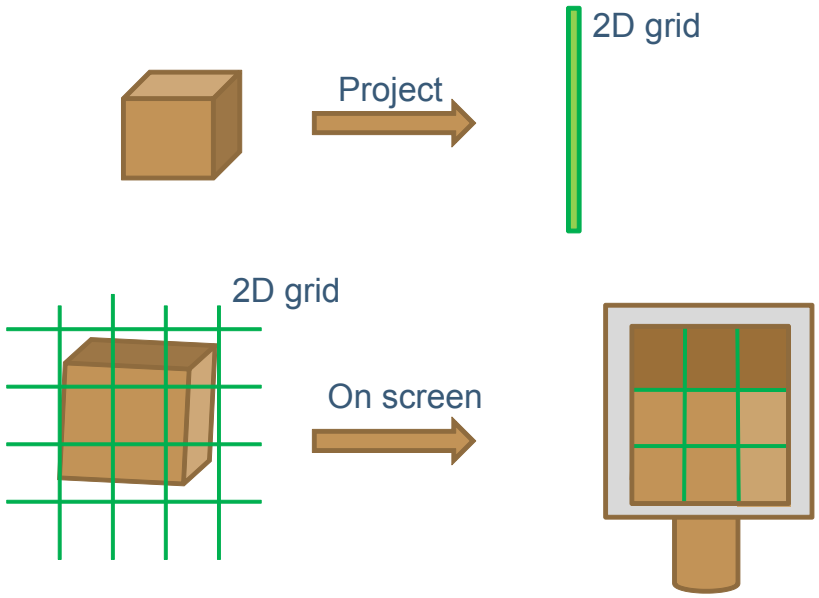

Figure 4.1 A 3D object projected onto a 2D grid. The colour of the projected object in centre of each grid cell is used as the pixel colour on screen.

```
for each object in scene do
    Project onto grid
    for each grid point covered by projected object do
        Calculate colour of this point on the object.
        Calculate initial distance between point on un-projected object and grid.
        if initial distance < depth stored in corresponding location in colour buffer then
            Store colour in corresponding location in colour buffer.
            Store distance in corresponding location in depth buffer.
        end if
    end for
end for
```

Figure 4.2 Pseudo code for simplified 3D graphics.

The resulting colour buffer is displayed on screen. Through a graphics API one can configure the type of buffers an application will write into. A GPGPU application could attach a buffer storing single floats instead of the four-component colour buffer. Then a scene with a single quad can be "rendered", with a projection which perfectly aligns it with the 2D grid. This is done just to set up the GPU processing. By assigning coordinates to the four corners of the quad, interpolated coordinates are later automatically available in each grid point. Instead of calculating the colour for a pixel, the application can calculate the one-component floats and store them in the custom buffer.

When used in GPGPU the calculations for the colour is replaced with a custom shader program, a small program that performs the calculations for a single grid cell. The type of shader discussed

here is a fragment shader, run for each pixel on the screen (really all potential pixels on screen). Shaders for use in graphics applications are explained in more detail in [30]. Good resources on shader programming and graphics programming in general are [31] and [24] respectively.

5 ITED - Architecture

This section describes the architecture of ITED and what programming languages, libraries and toolkits are used.

5.1 Conceptual Structure

This section explains how ITED organizes processing and visualization of elevation data. Conceptually, ITED is split into two parts: visualization and processing. The conceptual part concerned with visualization is from now on referred to as the visualization engine (VE), and the corresponding part for processing, the processing engine (PE).

ITED is based on the concept that the same data structures can be used both by the VE and the PE without modification. In this way the elevation data can both be visualized and processed at the same time, without having to keep separate copies of the data.

As shown in figure 5.1, the VE and PE both access the same data. The VE calculates colours in lines of what is explained in section 4.2 and displays the result on the computer screen. The PE accesses the same data, calculates changes, and updates the data so the changes are reflected in the next frame produced by the VE.

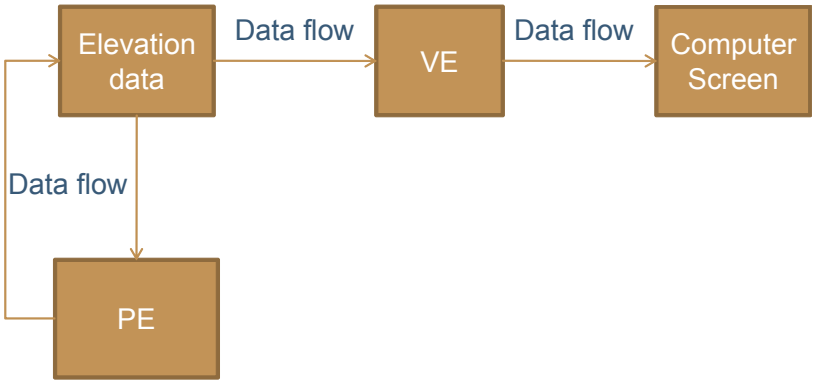

Figure 5.1 The conceptual structure of ITED.

In a typical work session in ITED a user controls a “brush” by moving the mouse around in the application window. The position and size of the brush is accessed by the PE and used for configuring the processing of all elevation data elements that are covered by the brush. A screen shot of an ITED session with four loaded tiles and a mouse controlled brush is displayed in figure 5.2. In this case the PE would calculate which data elements of the two lower tiles are covered by the brush (outlined in pink) and set up processing for only those elements.

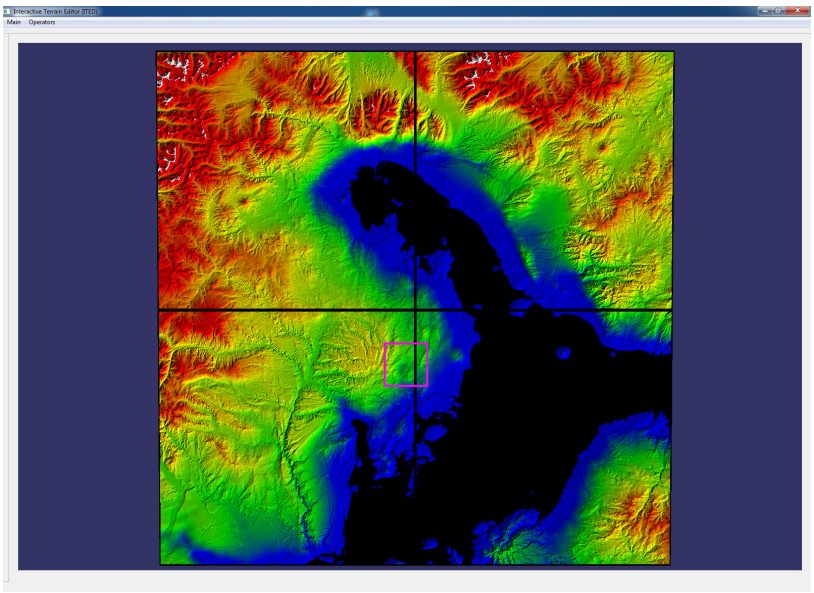

Figure 5.2 Four loaded tiles and a mouse controlled brush in ITED.

5.2 Libraries and Programming Languages

The programming language of choice for ITED is C++. This is mostly because the application is based around OpenSceneGraph (OSG) [34], a visualization toolkit written in C++, but also because C++ is the dominant language for real-time graphics applications today.

OSG is a layer on top of OpenGL. The toolkit consist of a scene graph [30] as well as a rich set of plugins for reading and writing different formats for representation of 3D models and images, etc. ITED uses OSG for visualization and processing of elevation data through shader based GPGPU. ITED also uses the matrix and vector implementations the toolkit includes for linear algebraic operations.

The use of OSG implies that OpenGL is the graphics API utilized in ITED. Most of the OpenGL API is hidden in OSG, unless you look for it, but the configuration of the graphics application exposes enums, constants and functions known from OpenGL.

OpenGL Shading Language (GLSL) [35] is the shading language used in ITED. Because the graphics API is OpenGL, GLSL was a natural choice, but it might just as well have been C for Graphics (Cg) [27]. For the use in ITED, there would probably not have been a big difference in implementation difficulty or performance.

Geospatial Data Abstraction Library (GDAL) [36] is a C++ library for representing geospatial data and performing Input/Output (I/O) on common file formats for raster and vector data. ITED uses GDAL for reading and writing elevation data from and to files, as well as for handling some aspects of geo-referencing and geospatial coordinate systems. ITED also loads vector data for visual reference through GDAL.

5.3 Data Structures

This section explains how the elevation data is structured and stored.

5.3.1 Tiles

Many terrain rendering systems organize elevation data in tiles. In this way the data can be portioned in chunks that can be moved up and down the cache hierarchy (disc, RAM and GPU memory) in a manageable way. When the size of geographical areas is too big and/or the resolution of the data is too high, the amount of data can also be too large for file formats and file systems, but advantages in memory management etc., expose themselves earlier than that. TerraPage [37] and Common Database (CDB) [37] are examples of terrain database formats that organize data in tiles for paging (loading parts from disc). CDB organizes the tiles so that they are equal in size in memory. In this way an application will know beforehand the amount of memory that needs to be allocated, and can optimize its data structures for this size.

ITED organizes the elevation data in tiles for two main reasons. First of all, source files are always organized this way, for example in one by one degree geo-cells. Secondly, GPUs have a limit in the size of a single texture that can be allocated in GPU memory. ITED uses textures for representing elevation data in GPU memory (read more about this in section 5.3.2). The limit has up until the Fermi architecture been 8192 by 8192 elements, but some Fermi cards now have the possibility to use textures with 16384 by 16384 elements. The organization of the data in tiles, where one loaded source file is represented as one tile, seemed like the most straight forward approach implementation wise as well.

Currently there is no caching or paging implemented. The tiles (textures) are loaded into GPU memory once, at full resolution. If the combined size of the tiles is too large to fit in GPU memory, the result is undefined.

5.3.2 Storing Data in Textures

In graphics terms a texture is a one, two or three dimensional data structure that can store up to four components (elements of a given data type) in each slot. As mentioned in section 5.3.1, this is the data structure used for storing elevation data in ITED. Instead of specifying a 2D texture with four components, each of 8 bits, representing the four colour channels red, green, blue and alpha (as for a normal image texture) the 32 bits are used in one component to represent a 32 bits elevation value. This is a perfectly valid way of specifying a texture, and there are configuration flags available in OpenGL which can be used to create such textures.

5.4 Coordinate Systems

Representation of different coordinate systems and the conversion between them are essential in ITED. All tiles of elevation data and mouse controlled objects, etc., have their own coordinate

systems. The coordinate systems are Cartesian and have normalized coordinates. One corner of an object has coordinates $(0, 0)$, while the diagonally opposite corner has coordinates $(1, 1)$. These coordinate systems are called reference frames (RFs).

A bounding frame just encompassing all loaded tiles is used as a global RF and has its own Cartesian coordinate system with normalized coordinates. Mathematically, a point within this RF can be specified using the RF of any of the objects in the workspace. If you know what RF the coordinates are specified in, a conversion to any of the other RFs can be performed through a simple matrix-vector multiplication. The referencing of objects within the global RF is illustrated in figure 5.3.

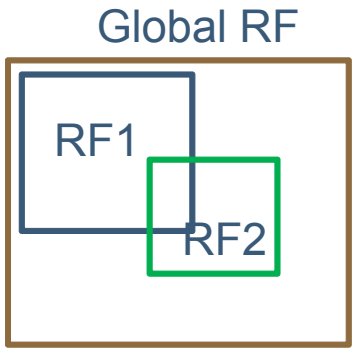

Figure 5.3 Two objects with local RFs termed RF1 and RF2 are referenced within a global RF.

Matrix-vector multiplications are a central part of computer graphics. The positioning of an object in the 3D world space of a scene is done through matrix multiplication. The same is true for the projection of world coordinates into screen coordinates. Due to this, matrix multiplication functionality is available both in OSG and in GLSL.

The most important coordinate systems in use in ITED are: the geographic coordinate system of the source data, the coordinate system of the global RF, the coordinate system of each tile and the coordinate system of a mouse controlled object which can be moved around on the computer screen. All these coordinate systems are 2D, representing a position on earth without any elevation information. The geographic coordinates of the source data are only used in the load and initialization phase, and eventually in the save phase where the modified data is written to disc again. As soon as the data is loaded, the coordinates of a tile is transformed into the normalized coordinates and the tile is referenced within the global RF. ITED does not care about the geographic coordinates from then on and to the point where the data is saved.

Both geographic lat/lon coordinates and UTM coordinates are treated as linear, Cartesian coordinates by ITED. If a dataset spans from $a = -37°$ West to $c = -32°$ East and $b = 50°$ South to $d = 52°$ North, the coordinates (x, y) can be normalized to yield the normalized coordinates (x_n, y_n) using the following equations:

$$x_n = \frac{x-a}{c-a} = \frac{x-(-37)}{-32-(-37)}, \tag{5.1}$$

$$y_n = \frac{y-b}{d-b} = \frac{y-50}{52-50}. \tag{5.2}$$

For each tile of elevation data that is loaded, the global RF is updated to encompass the new geographic extents. The transformation matrices of the loaded tiles are updated so they reflect the new size and shape of the global RF. Each tile has a transformation matrix that converts from coordinates in the local RF to coordinates in the global RF, and the inverse matrix that converts the other way around.

The equations (5.1) and (5.2), with a, b, c and d specifying the geographical extents the global RF represents, can be used to calculate the normalized coordinates of the boundaries of a tile specified in the global RF. The coordinates x_o and y_o of the origin of the tile, specified in the global RF and x_d and y_d the coordinates of the diagonally opposite corner, can be calculated using these functions. Then the coordinate conversion matrix M that converts from local RF to global RF in normalized coordinates can be specified as $M = T * S$ where S first scales the coordinates with $\frac{1}{x_d - x_o}$ and $\frac{1}{y_d - y_o}$ so that the size of the local RF is correct with respect to the global RF, and T translates the coordinates by adding x_o and y_o to the scaled coordinates so that the origin of the local RF ends up in correct position in the global RF. The effect of the matrices S and T are visualized in figures 5.4 and 5.5 respectively.

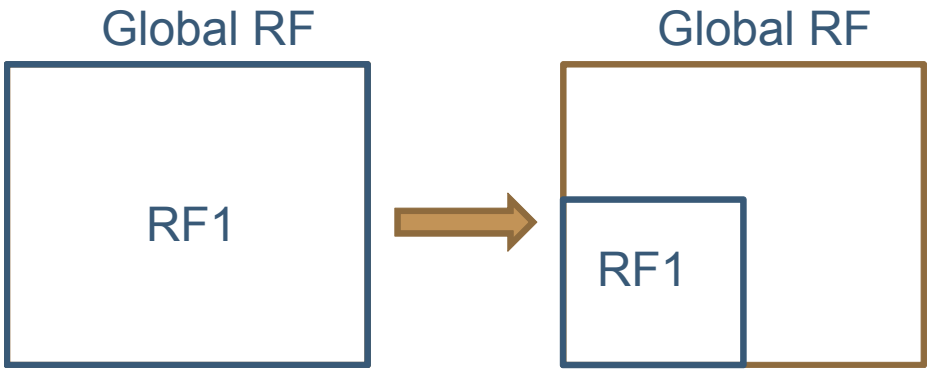

Figure 5.4 The effects of the scaling matrix S.

If a mouse controlled object is used to determine what parts of the loaded data should be subject to modification, the PE can simply convert the coordinates of the current elevation data element specified in the tile RF into the RF of the mouse controlled object and check if both the x and y coordinates are within $[0, 1]$. If not, the element is not covered by the mouse controlled object and should, by the rules set for this type of processing, not be changed. If the coordinates are within $[0, 1]$ the new elevation is calculated. Either way, the new or unchanged elevation value is written back to the dataset (texture).

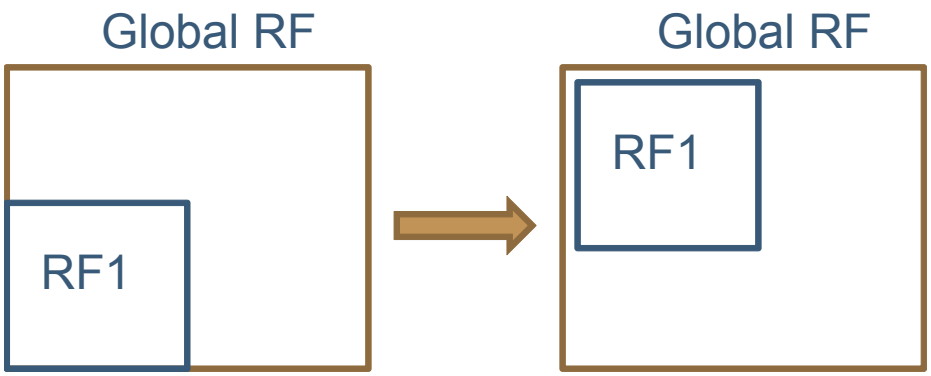

Figure 5.5 The effects of the translation matrix T.

5.5 The Scene Graph

The VE and the PE uses the same OSG scene graph. The scene graph is composed of nodes of different types: transformation nodes, geometry nodes, camera nodes, switch nodes, etc. The theory of scene graphs is not covered in this report, the reader is advised to read [30] for an introduction to OSG and scene graphs.

The transformation nodes are for example used to position objects in the world space, which align with the global RF as discussed in section 5.4. In this case the transformation node is tied to the transformation matrix that converts from the local RF of the object to the global RF. If any geometry associated with the object is specified in its local RF and located below the transformation node in the scene graph, it is rendered at the correct location.

The camera nodes are used both by the VE for positioning a virtual camera for rendering to screen and by the PE to set up processing for tiles of elevation data. The PE uses one camera per tile. The switch nodes are used by the PE to switch the processing cameras on and off according to a user's interaction with the application. If the user presses the left mouse button when in "processing mode", the PE sets the switch nodes to "ON" which causes OSG to use the cameras located under the switch nodes in the scene graph, hence switching on the processing.

The scene graph populated for a single elevation tile is illustrated in figure 5.6. The figure is a bit simplified, for example geometry is omitted, but it shows the concept. The camera "Cam1" renders the scene graph onto the screen. It renders only the "render node", since the single other path to a drawable object is through "Cam2", and a camera node is not rendered by another camera. The elevation texture is attached to "render node" and is used when "Cam1" renders the node. The same texture is used as a write buffer, as explained in section 4.2, when "Cam2" is switched on. The "process node" is used for overriding some of the state information set by "render node"for rendering to screen.

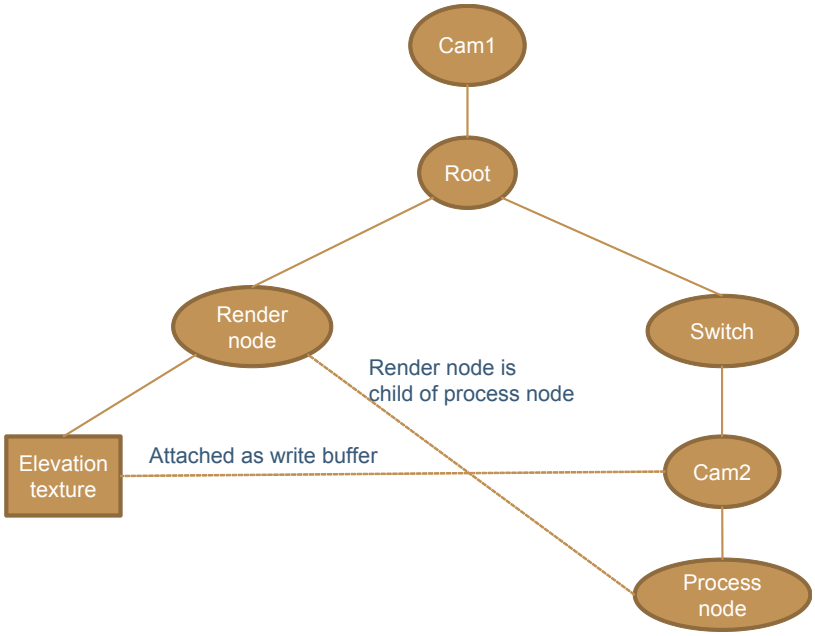

Figure 5.6 A simplified view of the scene graph of ITED populated for a single tile of elevation data.

5.6 Processing

This section explain how ITED configures the PE and briefly describes some of the C++ classes involved.

5.6.1 Processable

Processable is a class that handles the creation and configuration of the processing mechanisms for instances of the *Height Tile* (HT) class. A *Processable* object (PO) is created for each HT and instructed to create a camera which is configured for writing to the texture that holds the elevation data of the HT. The PO also creates the "process node" from figure 5.6.

5.6.2 Operators

An operator is a conceptual part which in ITED is defined to be a type of operation that can be performed on elevation data, relative to some RF. The RF is normally the RF of a mouse controlled object. The simplest operators are the lower and raise operators. These operators raise or lower the terrain that is directly underneath a mouse controlled object. Another operator is the smooth operator which performs a low pass filtering of the terrain.

The operator concept is implemented through the *FilterOperator* class (FO). This class is attached to a movable object implemented through the *Movable* class. An FO object has an attached shader program that implements the operation that is to be performed, and a list of variables that are used

for tuning the processing. These variables are “uniform variables”, a type of variable that gives global control of an instance of a shader program. The value of a uniform variable for a shader program will be the same for all threads that executes it. A schema that shows an operator and its access to different data is presented in figure 5.7.

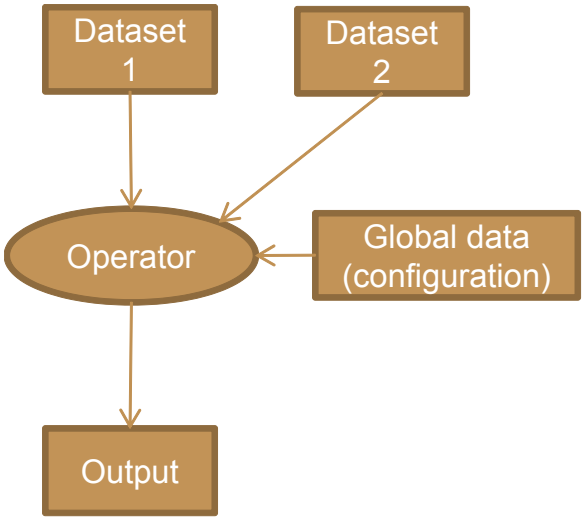

Figure 5.7 An operator can read from one or more datasets and access global parameters used for configuration. An output is written to an output dataset.

5.6.3 Control

A PO will attach the shader program of the FO to the “process node” (see figure 5.6) in the scene graph, the same goes for the uniform variables of the FO. The “process node” is configured to override any conflicting configurations that are found beneath it in the scene graph. As the “render node” of a HT has its own shader program attached for the rendering to screen, this shader program is replaced with the one attached to the “process node” when OSG runs the camera (“Cam2”) that is set up to do the processing. As the texture holding the elevation data is set up as the camera’s write buffer, the output is written to the texture, not to the colour buffer that is displayed on screen. For every frame that is displayed on screen, “Cam2” is first processed so the elevation data is up to date, then “Cam1” is processed to create the actual frame that is displayed on screen.

5.7 Visualization

Visualization is a crucial part of an application for interactive editing of SNEs. Ideally, the visualization should both provide the user with a clear view of the environment, as well as a good overview of the controls. The most important control in ITED is the mouse controlled movable.

The only view currently implemented in ITED is a 2D view. The rendered images are of the terrain viewed directly from above. The terrain is coloured according to the elevation, with user controlled parameters for the min and max elevation that are assigned colours. Elevations below the min

value are rendered as black, while elevations above the max level are rendered as white. The elevations between min and max are assigned colours using a look-up table with interpolation. This results in a smooth transition through blue, green, yellow and red, from min to max elevation. Shading is used to provide the same visual impression as a 3D view would, if it was rendered directly from above. The geometric representation of the terrain is flat in the VE, but lighting is applied as though the geometry match the slope and curves of the terrain. The render shader computes the normal vector of the terrain in positions corresponding to pixels on the screen. As explained in section 4.2, the colour is calculated for each slot in the colour buffer, which corresponds to a pixel on screen. This colour is calculated by inserting the colour retrieved from the look-up table and the normal vector into a standard lighting equation:

$$C = (w_{amb} + w_{dif} \cdot (\mathbf{n} \cdot \mathbf{l})) \cdot C_{LUT}. \tag{5.3}$$

This equation yields the colour C which is displayed on screen. The weights w_{amb} and w_{dif} are controlled through a Graphical User Interface (GUI) and represent the strength of ambient and diffuse light, respectively. The factor $\mathbf{n} \cdot \mathbf{l}$ is the dot product of the normal vector of the terrain and the position vector of a light source, and C_{LUT} is the colour fetched from the look-up table. These calculations are performed per-pixel. See [30] for more information on lighting and light equations.

The result is a view with hillsides shaded according to the position of a defined global light source. A screen shot of such a rendering is included in figure 5.8. A rendering of the same view, but without the hillside shading is presented in figure 5.9.

The 2D view gives a good overview of the terrain and good control of the position, orientation and size of the movable that controls the processing. However, the lack of a more ground orientated perspective and a 3D view can make it hard to make out how steep an area really is. This can for example result in unnaturally steep mountains when operations that “stretch” the terrain are used.

5.8 Framework

Operators are quite easy to implement in ITED. A new operator can be added by implementing a new shader program. This can be done in a few minutes for simple, but still useful operations. The file path of the shader program, the given name of the operator and the names and ranges, etc. of uniform variables that are to be controlled by the user, are added with a few lines of code in the application source code. After compiling the application, the new operator will be available through the GUI menus. When the operator is selected, another GUI window pops up, with sliders for the user controlled uniform variables that were registered. This can be regarded as a framework for development of new operators. This approach only allows for development of mouse controlled operators.

There is also an option to subclass the *Processable* class to automatically update the *Movable* object attached to the FO. This has been done in an experiment aimed at processing elevation data based on road vectors. The experiment is explained in more detail in section 7.4.

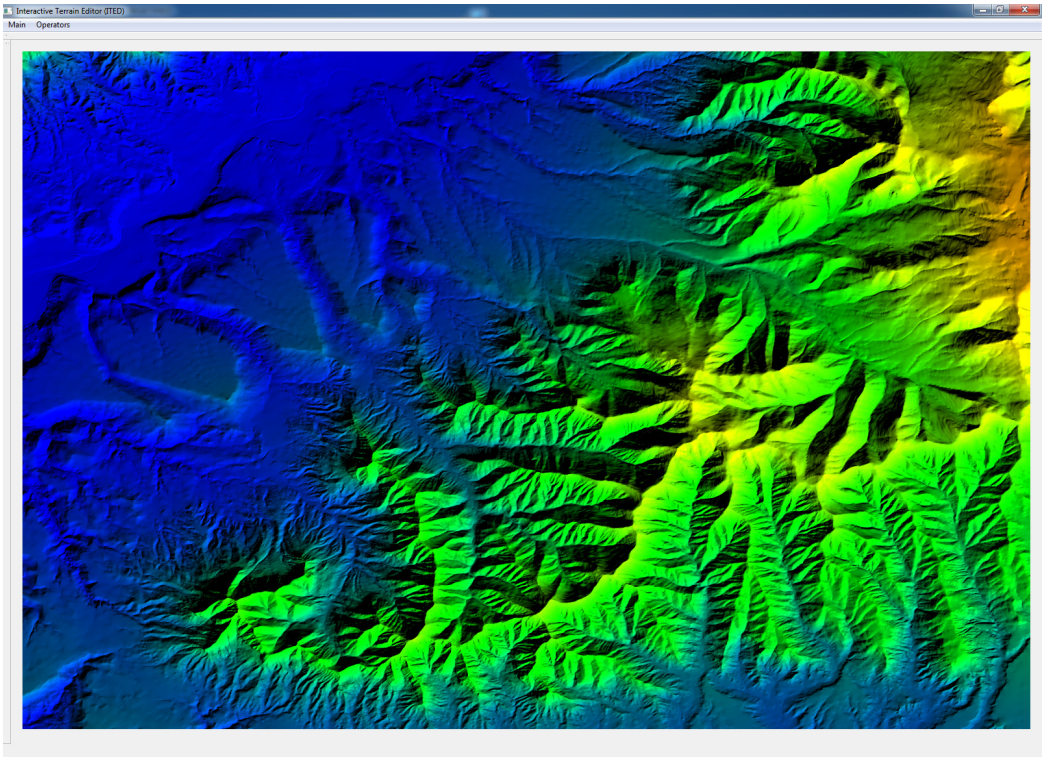

Figure 5.8 Hillside shading in ITED.

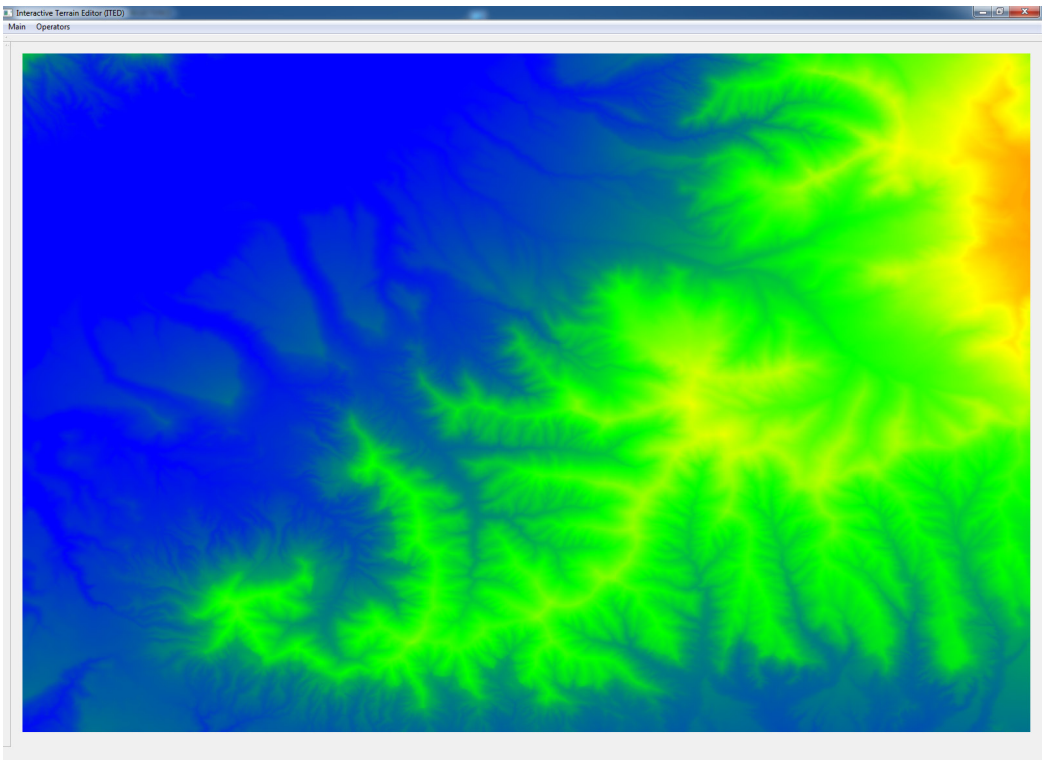

Figure 5.9 Ambient (no hillside shading) rendering in ITED.

6 ITED - Current Functionality

This section describes the functionality and user interface currently available in ITED.

6.1 User Interface

ITED has a quite simple user interface. When the application is started, only a main menu is available to the user. A screen shot of the options available in this menu is displayed in figure 6.1. To be able to do anything, a user needs to select the “Open elevation data” menu item. This causes a file dialog to open. One or more elevation data files can be selected and loaded.

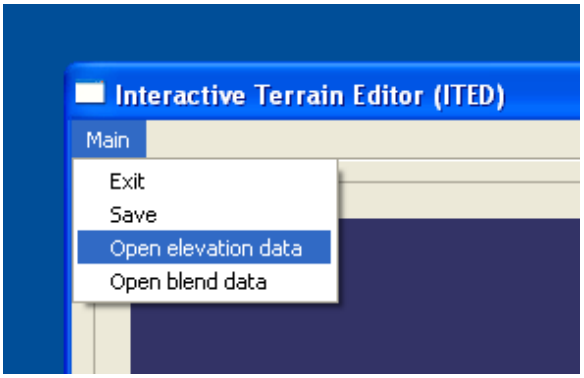

Figure 6.1 The options available in the main menu in ITED.

When one or more elevation data files have been loaded, a second menu becomes available. This menu is called “Operators” and the available menu options are displayed in figure 6.2. These options represent different operators as explained in section 5.6.2. If one is selected, the corresponding FO object is set as the current FO to be used when editing the terrain.

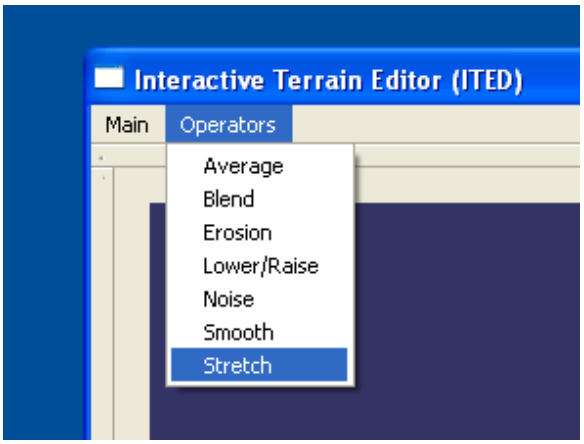

Figure 6.2 The options of the operator menu in ITED.

Another item that appears when one or more elevation data files are loaded is the “Render control” window. This window is displayed in figure 6.3. The render control window contains sliders that

enable a user to adjust the values of parameters that affect the visualization of elevation data. The results displayed in figures 5.9 and 5.8 were obtained by adjusting the “Ambient light” and “Diffuse light” parameters.

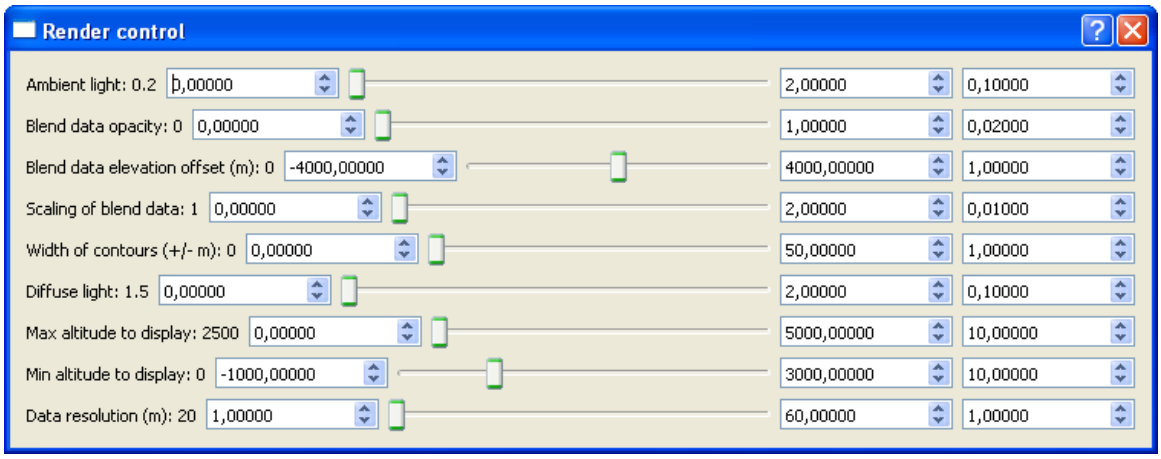

Figure 6.3 The render control parameters in ITED.

When an operator is selected, an additional window pop up, the operator control window. Similar to the render control window, this window contains sliders for control of parameters used in the operator.

In ITED the render camera is controlled through the ’wasd’ gaming keys as well as ’q’ and ’e’. Because the only view available is in 2D and because the mouse is used for control of the operator brush, this works quite well.

6.2 Data IO

ITED allows a user to load one type of files: raster elevation files. In general ITED could read all raster formats supported by GDAL, but only Digital Terrain Elevation Data (DTED), US Geological Survey Digital Elevation Model (USGS DEM) and GeoTIFF, a Tag Image File Format (TIFF) based interchange format for geo-referenced raster imagery have been tested. ITED only supports the use of one type of geographic coordinate system at the time: either lat/lon or a single UTM zone. If lat/lon and UTM are mixed, or tiles with different UTM zones, ITED will fail to reference the data.

6.3 Elevation Blending

The content of this section is strongly based on section 4.5 in [18].

ITED implements one function that is very useful in the context of MSG-071 and the design of large and realistic looking SNEs. This function is the ability to blend pieces of existing elevation data (sample terrain) into a target terrain. The blending process is completely interactive and user

controlled, and no visual borders or edges are left. The concept of blending elevation data is not new, [20, 9], but to the knowledge of the authors of this report, no one has implemented this type of interactive, seamless and user controlled blending before.

Results from elevation blending in ITED are displayed in figure 6.4. The left shows a rendering of the target terrain with a down-scaled, rotated and translated sample terrain on top. The image is just a rendering of two individual data sets, with one positioned over the other. No changes had been written to any of the data sets at the time when the screenshot was taken. To produce the terrain to the right, the sample terrain was scaled to its normal size (equal to the size of the target terrain), rotated and translated into different positions over the target terrain. The user then blended selected parts of the sample terrain into the underlying target terrain using a blend brush. The areas are 100 km by 100 km with 20 m horizontal resolution and the blending process took less than five minutes to complete.

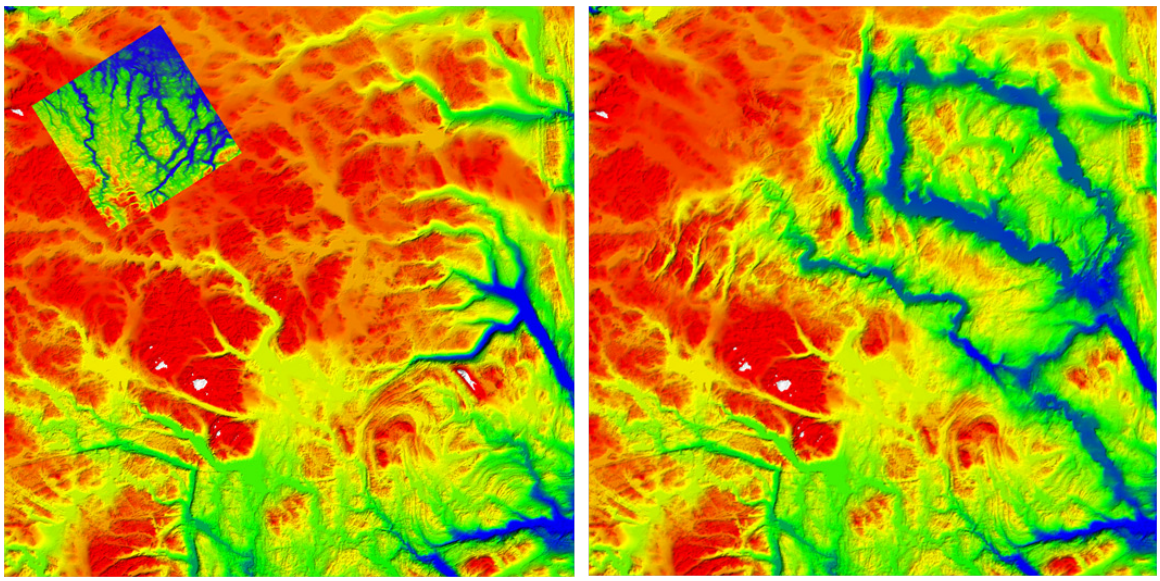

Figure 6.4 Renderings of elevation data from real areas (left) and the result of user controlled blending operations (right).

A short work session is also visualized in figure 6.5. The first image is a rendering of the target terrain and the second shows a sample terrain positioned over the target terrain. The third image is a screenshot from the middle of the blending process. The pink square visualizes the outer boundaries of a mouse controlled blend brush operated by a user. The sample terrain is rendered transparently so the target terrain shines through from below. In the image, the terrain formation (blue structure) being blended into the target terrain is sharper than the surrounding areas. This is because the formation now is part of both data sets. The last image shows the target terrain after the blending has taken place.

Figure 6.6 shows a 3D rendering of a cut from the target terrain after the blending process was completed. This is the same data that is rendered as a whole in 2D in the forth image in figure 6.5,

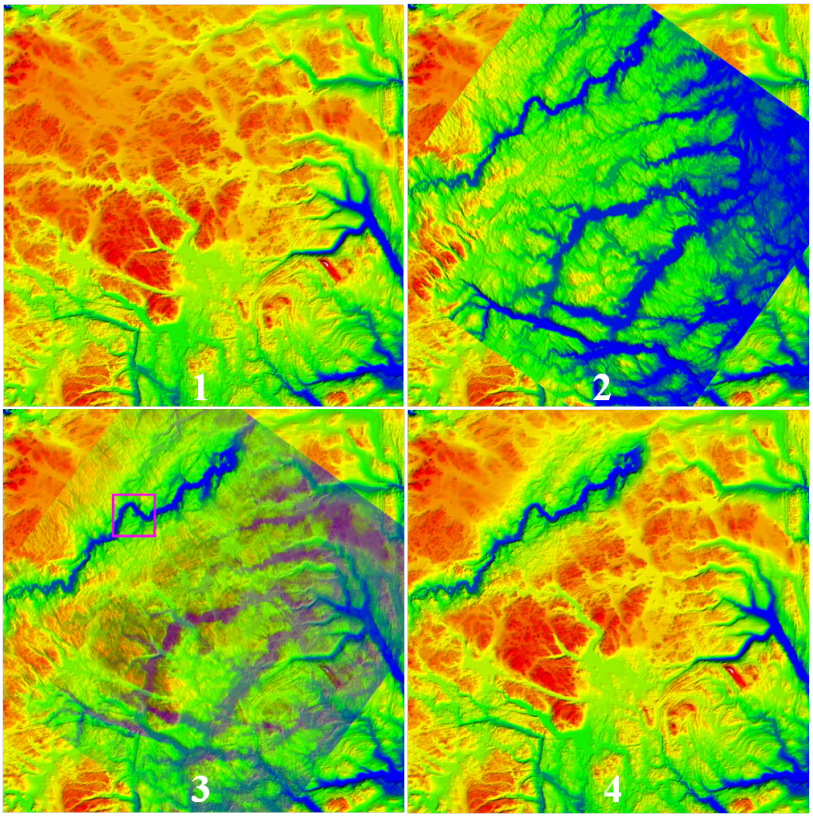

Figure 6.5 A blend sequence.

and the white square approximates the position of the pink square in the third image. Inspection of the terrain shows no unnatural borders or other artefacts created by the blending operations.

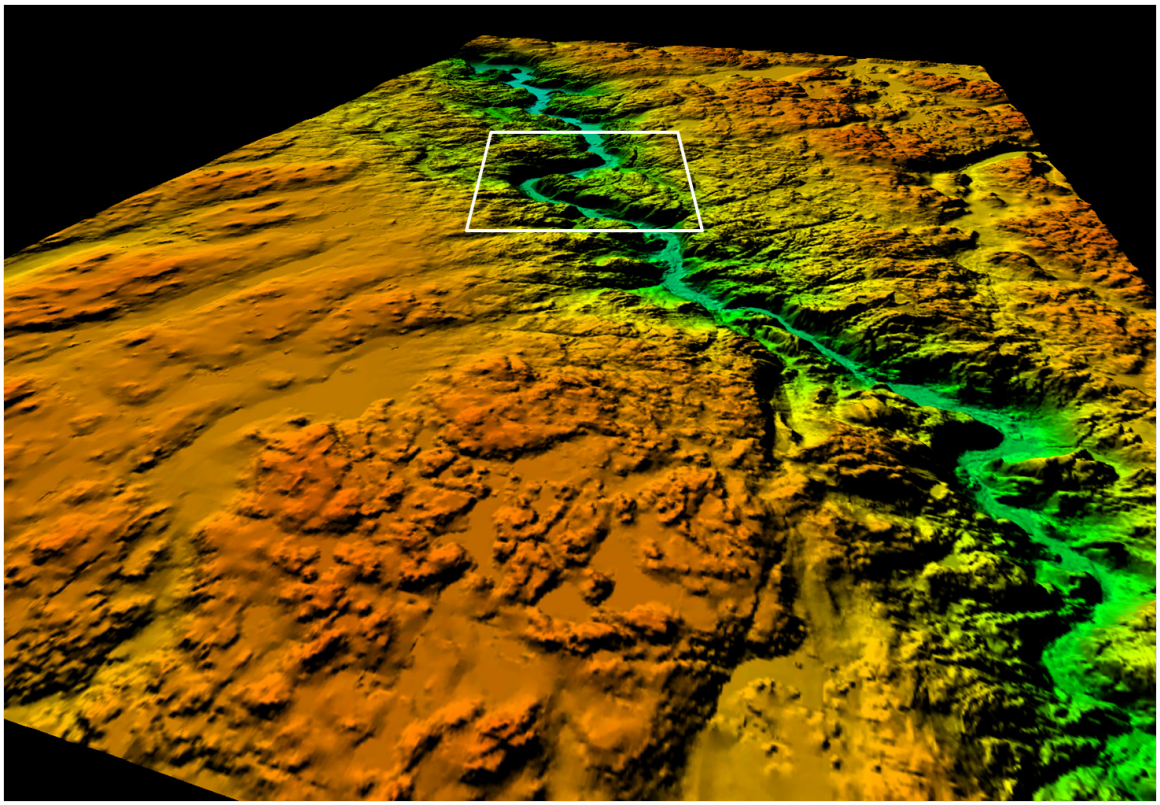

Figure 6.6 A 3D visualization of an area composed by blending in ITED. The rendering was done by Global Mapper.

Interactive blending of elevation data supports an artistic approach to design of new terrains. A user can choose features from a library of existing sample terrains and place them with precise position and orientation in a terrain being developed. Figure 6.7 shows the result of about five minutes of work. This time the target terrain was blank (zero elevation in all data points). The sample terrain was blended into the target terrain as explained earlier. As a last step the coastline and fjords were created by using another brush to lower the terrain.

The blending process has so far proven itself as a promising approach in enhancing areas of the Missionland elevation data. It has also been used in a process aimed at increasing resolution and detail in an originally low resolution area for use in VBS2 (see section 7.1 for more details.) In that setting the large lines in the terrain where to be preserved, while introduction of smaller, more high frequency structures was needed. This is a bit in the lines of what is done in [2]. One important difference, however, is that with ITED the process is interactive and user controlled, while the process in [2] is more automatic. With large datasets of elevation data now available in the public domain [38], such uses of real data can be good alternatives to procedural methods.

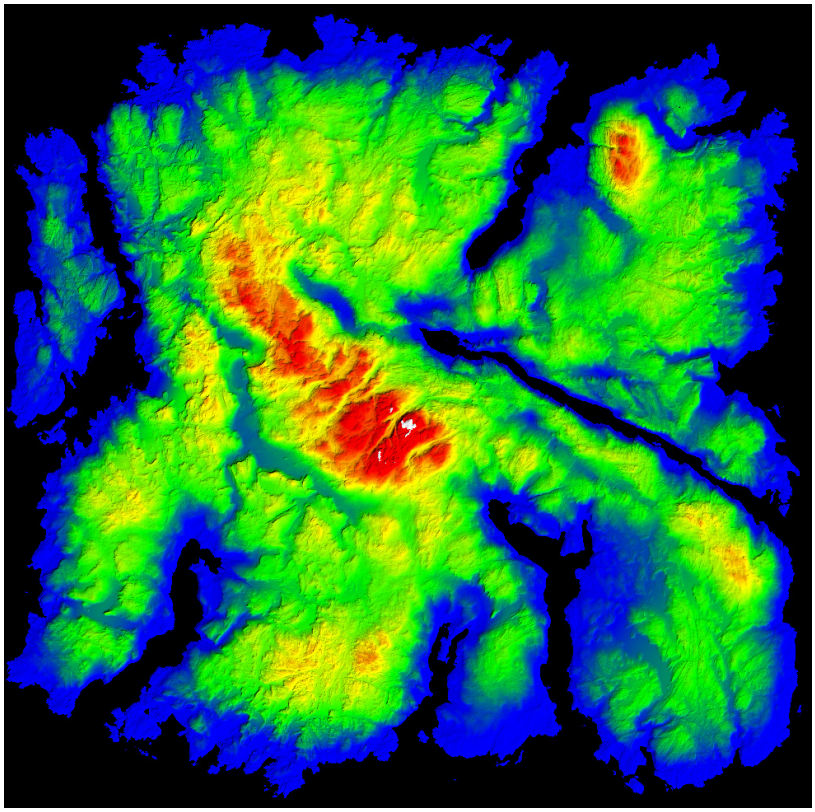

Figure 6.7 A result of blending and lowering operations. Sea level is rendered as black.

6.4 Operators

This section describes the operators currently implemented in ITED. The functionality of the blending operator is also described separately in section 6.3. Mathematical equations will be used as a mean to explain how the different operators process elevation data. The following coordinates will be used: rf_t, the coordinates of an elevation element specified in the RF of the current tile; and rf_o, in the RF of the *Movable* object attached to the FO. All operators also use two float variables: h_o and h_n, the old and new height respectively. The old height, h_o, is obtained by looking up the value in the elevation texture of the current HT using rf_t. The value of h_n is written back to the elevation data texture.

6.4.1 Lower/Raise

The lower operator implements both lowering and raising of the terrain. The sign of a parameter, s (strength), controls if the terrain is lowered or raised. The strength is controlled through the operator control window as described in section 6.1. The new height is calculated using the equation

$$h_n = h_o + s \cdot w, \tag{6.1}$$

where w is a computed weight that causes the operator's strength to fall off toward the edges of the RF of the *Movable* attached to the FO. All operators use the same equations for computing the weight:

$$w = \cos(d \cdot 180), \tag{6.2}$$

where d is the euclidean distance between $(0.5, 0.5)$ and rf_o, and $d < 0.5$. If $d \geq 0.5$ then w is set to 0.

6.4.2 Smoothing

The smoothing operator computes an average elevation, a, of an $n \cdot n$ neighbourhood around the current elevation element. The variable n is controlled through the operator control window as described in section 6.1. Then the new height is calculated using the equation

$$h_n = h_o \cdot (1 - s \cdot w) + a \cdot s \cdot w. \tag{6.3}$$

6.4.3 Stretching and Compressing

The stretch operator computes the same average, a, as the smoothing operator. It can push the value of an elevation element toward the average, in the same way as the smoothing operator, but it can also pull the value away from the average. The new height is calculated using the equation

$$h_n = h_o + (h_o - a) \cdot s \cdot w, \tag{6.4}$$

where s can be both negative and positive. If $s < 0$, a compression toward the average occurs, and if $s > 0$, the elevation is pulled away from the average. When $s > 0$ the terrain is stretched into a steeper terrain, hence the name.

6.4.4 Blending

The blending operator is different from the other operators in the sense that it uses an additional set of coordinates, rf_b. These are the coordinates of the current elevation element, in the RF of the sample elevation tile loaded for blending (blend tile). The new height is calculated using the equation

$$h_n = h_o \cdot (1 - s \cdot w) + h_b \cdot s \cdot w, \tag{6.5}$$

where h_b is the elevation at the coordinates rf_b in the blend tile. Using a low strength and a moving mouse yields an iterative blending and prevents sharp edges from being formed. If the frame rate of the system is high, the process is still fast and interactive.

7 Use cases

This section describes the different settings ITED has been used in.

7.1 Enhancement of Zoran Sea Elevation Data

The Zoran Sea scenario is a NATO scenario with an elevation dataset used by the FFI. The dataset is made available to FFI by the Norwegian Military Geographic Service (FMGT). The elevation data in the dataset contain elevation contours with 100 m resolution from which a raster elevation dataset can be built (the dataset also contain slope information, but this data was not used in the current example). The resulting raster data contains very little detail and some dominant terrace effect due to the nature of the contour representation. A rendering of a part of the Zoran Sea raster elevation data is presented in figure 7.1.

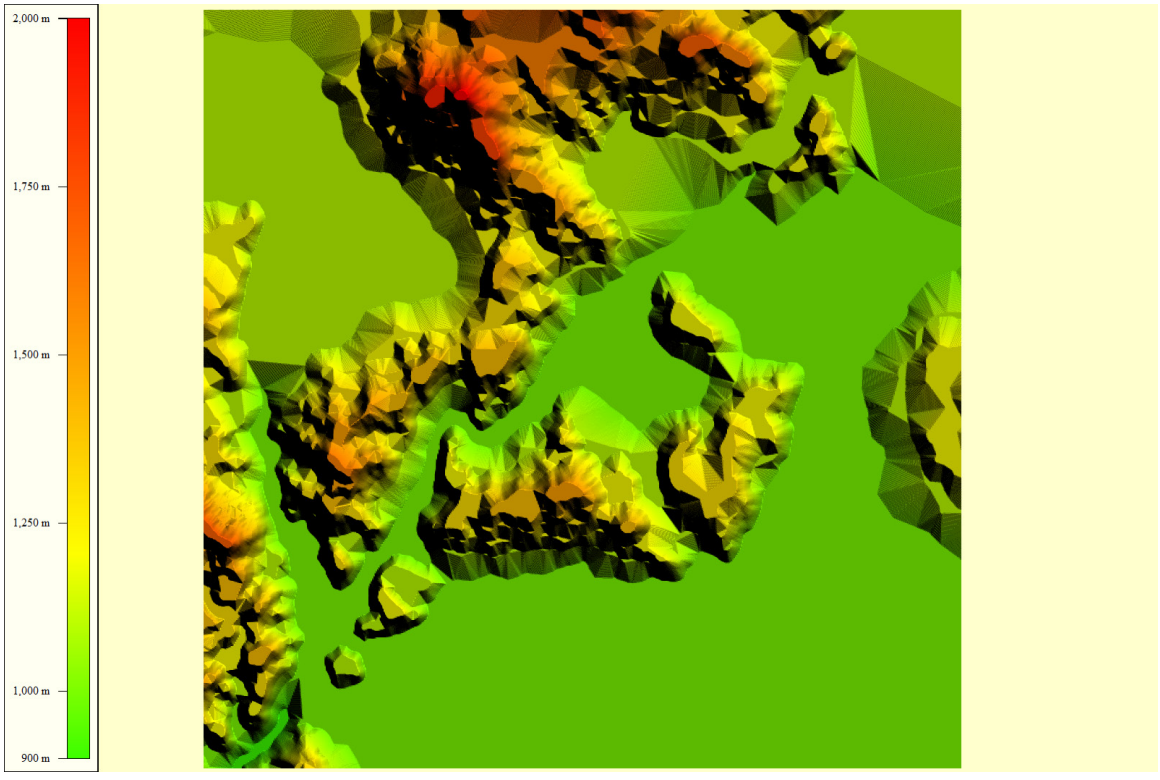

Figure 7.1 Zoran Sea raster elevation data derived from elevation contours.

In figure 7.1 the terraces are clearly visible. The displayed area was to be used in VBS2 and more detail and less terraces were wanted. As a first step the raster data was oversampled to obtain a

10 m resolution dataset. Then the 10 m raster data was interactively smoothed in ITED to reduce the terrace effects. The result from this step can be viewed in figure 7.2.

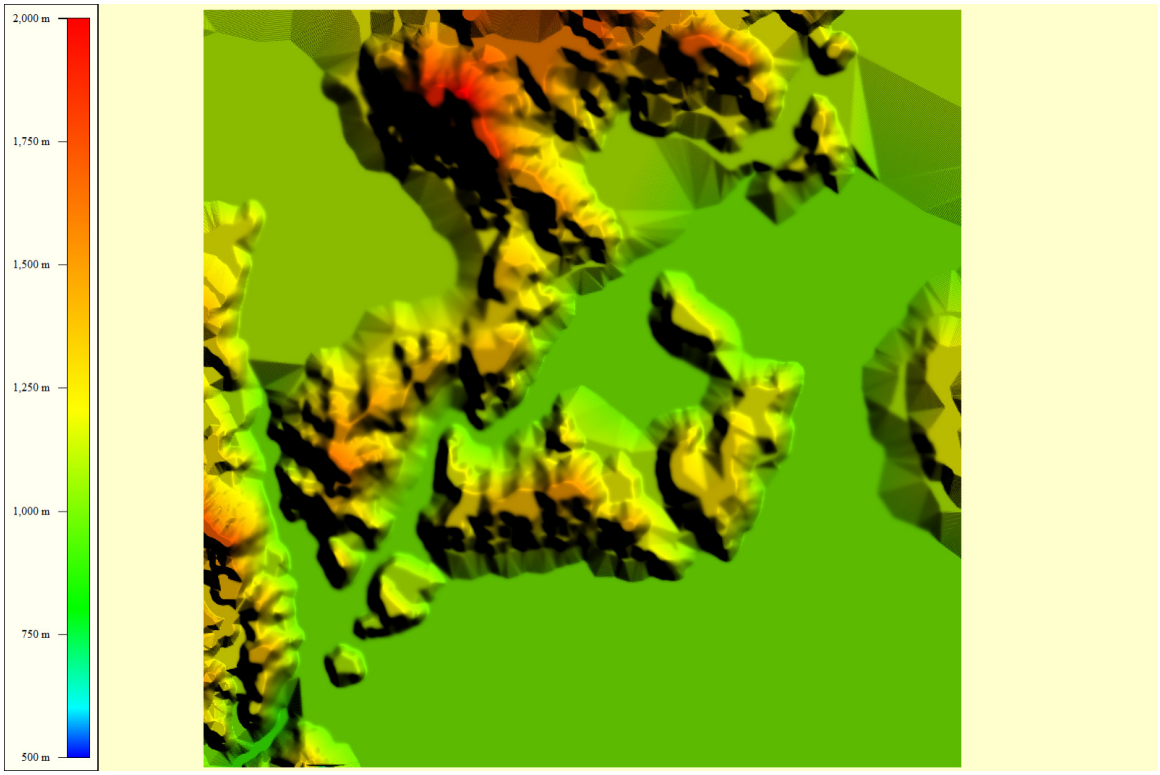

Figure 7.2 Zoran Sea raster elevation data after smoothing in ITED.

The elevation data displayed in figure 7.2 result in a higher visual quality than the original raster data, but terraces are still clearly visible and the resolution of the data (10 m) is much higher than the information contained in it. This is due to the fact that oversampling by linear interpolation, as was done here, does not introduce any new information in the dataset. To be visually compelling, and to support more realistic simulations in VBS2, more detail had to be introduced. In an effort to preserve the large lines of the terrain (so maps would still be accurate) while introducing more detail, real-world data was blended into the terrain using ITED. A rendering of the results of the blending process is displayed in figure 7.3.

After inspection in VBS2, the enhanced version of the dataset was concluded to be much more visually compelling. It was also clear that added detail in the areas that were originally very flat helped provide a more realistic line of sight, and thus better support for realistic simulation and experimentation conditions.

7.2 Missionland

As described earlier in this report, ITED was developed with the needs and challenges of MSG-071 in mind. The task group has an objective to produce an environment dataset for a large fictitious continent in the Atlantic Ocean. The group has identified a detailed and realistic elevation dataset

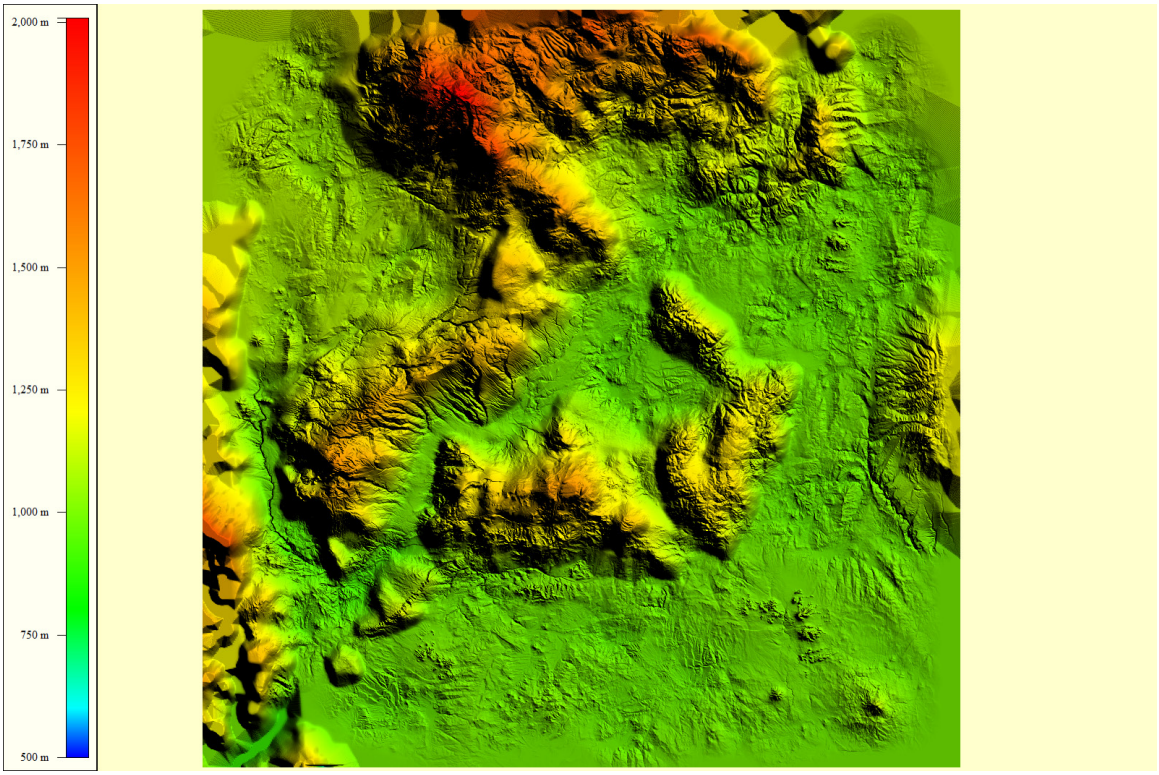

Figure 7.3 Zoran Sea raster elevation data after enhancement by blending in ITED.

as an important fundament. As the group considers it easier to derive imagery and vector data from the elevation data than vice versa, the group started out by generating elevation data.

The procedural terrain generation tools Large 3D Terrain Generator (L3DT) [4] and GenMap have been tested by the group. The latter is developed by the Turkish company SimBT, which has a representative in MSG-071. At the time of writing, an elevation dataset of the whole continent with roughly 30 m resolution serves as a basis for further enhancements. This dataset was produced by the use of L3DT.

The L3DT elevation data was generated from a rough elevation data model called a design map. This design map was edited in ITED, as the internal editor of L3DT could not provide interactive rates while editing the map of 8192 by 8192 data points. In addition, the brushes available in the editor were all solid. This rendered it difficult to edit the map in a smooth fashion without leaving visible artifacts (sudden and abrupt changes in the elevations). ITED however, provided interactive rates while editing with the much smoother lower/raise (see section 6.4.1) operator brush.

As the work of MSG-071 has progressed, it has become more and more clear that procedural terrain in general does not look real when compared to real-world data. Because of this, and because of lack of other input and initiatives, ITED is the key tool in elevation data production in MSG-071. At the time of writing, an area of the continent has been enhanced using ITED. This process involved blending real-world data with 10 m resolution into an oversampled version of the 30 m L3DT data. The results of this effort were presented at the MSG-071 meeting in Amersfoort,

the Netherlands in January 2011. At the same meeting it was agreed to try to completely redo the 30 m basis data produced in L3DT by blending real-world data into the dataset using ITED. When source data is available as large tiles for use as “blend data” in ITED, the process of enhancing the whole continent in 30 m resolution is predicted to only take a couple of days. The process of pre-processing the source data is predicted to take longer.

Even though the blending process is a fast way of creating new elevation data for areas with given specifications (size, shape and terrain characteristics, etc.), the quality of the result is 100 % dependent of the skills of the user. When the first version of the enhanced 30 m dataset of Missionland is ready, it has to be inspected to see if the terrain is plausible. There is no guarantee that transitions between different terrain types are realistic and that drainage networks and long valleys, etc. are contained in the data just because data containing such characteristics and elements are used as input. However, this blending process seems like the best available option if one aims to create realistic looking areas with specific shapes, sizes and with the same characteristics as real world areas.

7.3 Dynamic Terrain

FFI has implemented a prototype simulation of a battlefield with dynamic terrain (elevation data). ITED was used as a basis for the simulation. The VE of ITED was used to build a viewer and the PE was used to integrate the effects of detonating grenades and moving heavy machinery in the environment. The simulation applied crater shaped brushes centered at the impact points of grenades. In the same fashion; moving armoured vehicles triggered trailing brushes shaped as track marks. These dynamic effects only targeted the elevation data and terrain textures, no other 3D models were affected by the grenades or moving vehicles. A screenshot from the viewer is presented in figure 7.4.

The main concern with such an approach is that the elevation data is resident in GPU-memory. A transfer from GPU-memory to CPU-memory is needed for all environment aware operations that are to be performed on the CPU. In this particular case, the elevation in points that were needed to perform ground clamping was read from GPU to CPU memory on demand. A close up of a track encoded in the elevation data is presented in figure 7.5.

The work on dynamic terrain was not continued after the prototype was completed, but it might be something to look into if one were to consider new and alternative ways of representing the SNE in a simulation. However, the combination of GPU resident and dynamic elevation data seems to complicate matters when compared to more traditional approaches. On the other hand, it is not sure that it is more complicated than other ways of representing dynamic terrains.

Figure 7.4 Armored vehicle tracks. Both elevation data and textures encode the tracks.

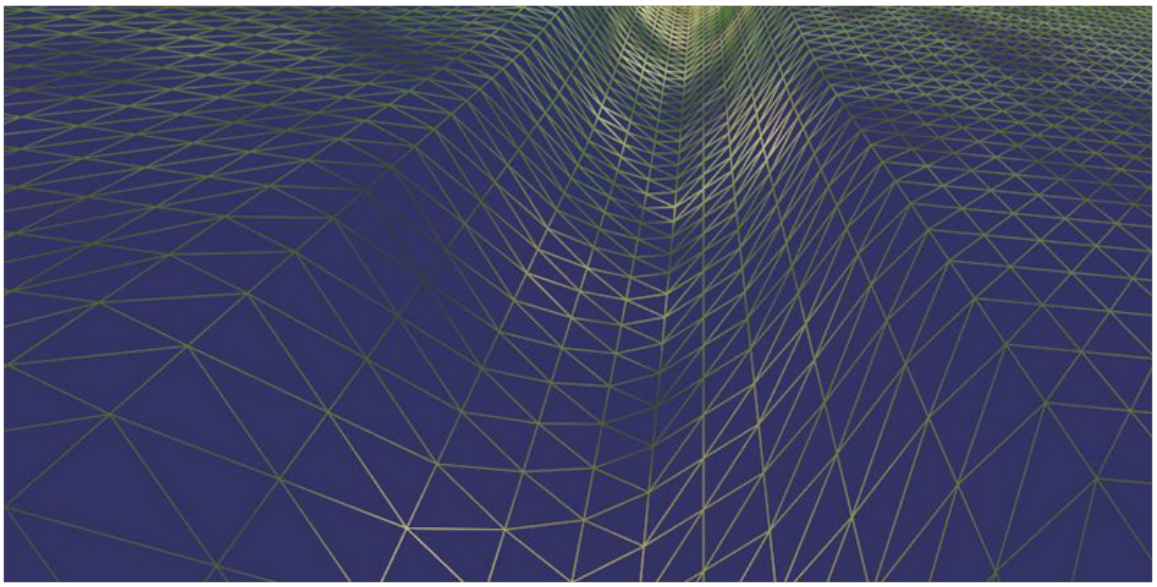

Figure 7.5 A close up of tracks encoded in elevation data.

7.4 Automatic Processing of Roads

In section 7.3 automatic processing of elevation data was described in the context of dynamic terrain. Similar automatic processes can also be used for pre-processing of elevation data for use in M&S databases. For instance, vector data representing roads can be used to control a brush that flattens the surface of the road and encodes ditches in the elevation data. Of course, this requires high resolution elevation data ($\leq$ 20 cm?) for good end results. A subclass of the *Processable* class (see section 5.6.1) has been implemented for this purpose. An object of the class can move a brush along a set of road vectors (line segments) and process the area covered by the road and ditch as explained earlier.

The processable object iterates through the vertices in the vector data. In figure 7.6 each rectangle represents one iteration. The rectangles also represent the shape and location of the *Movable* object used by the brush (see section 5.6.2). The green vertices are the vertices that are sent as parameters to the brush in the iteration of the green rectangle. These vertices are used in the shader program that implements the brush. The access to the coordinates of the vertices enables the shader program to reference all elevation data elements within the rectangle relative to the road. Thus, for each elevation data element, the distance to the closest point on the centre line of the road can be calculated.

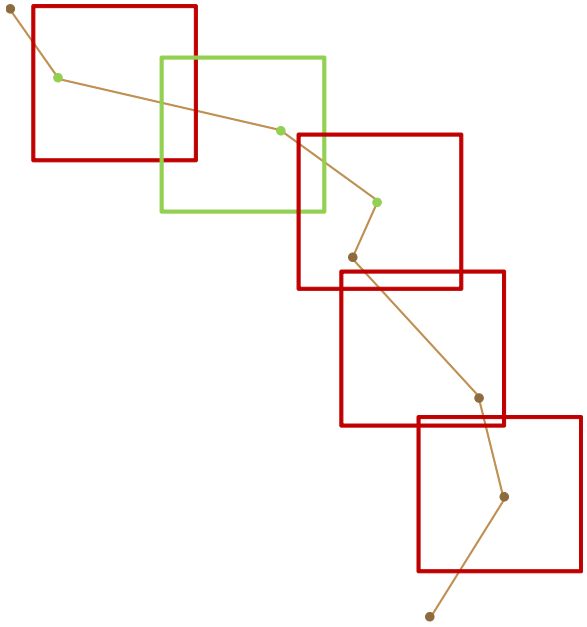

Figure 7.6 Processing elevation data based on road vectors.

This was done as an experiment, as the functionality is not explicitly required in ITED at the time being. The work was left of with an implementation that could process a set of road vectors, but would produce artefacts when vertices in the segmented lines were too close to each other. Intersections were not handled in the implementation.

The authors of this report regard the experiment as a success in the sense that it gave and

impression that such processing of roads could be achieved if a little more time was put into the implementation. Also, even though not tested, implementation of the same type of processing for ditches and other simpler (more randomized) terrain structures can probably be implemented using the vector controlled *Processable* subclass and brushes similar to the road brush.

8 Further Work

This section describes and discusses functionality that is considered to be both useful and feasible to implement in ITED.

8.1 3D View

A 3D view could complement or replace the 2D view currently implemented in ITED. As mentioned in section 5.7, a 3D view would assist a user in getting a good impression of how the terrain will look in a 3D application. This would especially be useful when performing operations that affect the slopes of the terrain.

Unfortunately, a good 3D view is a bit more complicated to implement than a 2D view for this kind of application. The process of determining what object in a scene a user is trying to select with the mouse is called picking. ITED performs picking through functionality in OSG. A ray is cast from the position of the mouse and into the screen. The intersection between this ray and the terrain defines the “picked” point. Because the 2D view uses a completely flat geometry, this is an easy operation that will always yield the result the user expects. When using a static 3D geometry, the process would be more or less the same, only slightly harder to implement. In ITED the geometry would have to reflect the current status of the elevation data. Because of performance reasons, both the geometry and the elevation data would have to reside in GPU memory, and are not readily available for processing by OSG using the CPU. One way to overcome this is to dedicate another write buffer to storing the global RF coordinate of every pixel on screen. In that way the GPU could calculate the coordinates and store them in GPU memory. Then only the slot in the buffer corresponding to the current mouse position on screen would be read back to the CPU each frame. This is currently not implemented in ITED.

Since having a 3D view would involve more geometry for rendering, a system for handling Level of Detail (LOD) might have to be implemented. However, since the shading is performed per pixel with hillside shading as shown in figure 5.8, the geometry can probably have a considerably lower resolution than the elevation data and still yield good results.

8.2 Source based rendering and simulation

Most visual systems that render terrains are built on triangle meshes. A popular type of mesh is a triangulated irregular network (TIN). A TIN is a triangle mesh that has more resolution in the areas

where the most detail is needed. When a TIN is used as a terrain skin, the most detailed areas of the TIN normally correspond to areas where the source elevation data contains the most detail. Flatter areas will have a corresponding area of the mesh with a lower density of triangles.

One motivation for using a TIN is to limit the number of triangles to be drawn on screen at any given frame. Nowadays high end graphics cards have a very high triangle through-put. That is one reason why some efforts are focused on other ways to represent a terrain skin. A TIN is a data structure that takes a long time to generate and is hard to maintain. It also needs to store the x, y and z components of every vertex in the mesh. That is an effect of the irregularity.

If one were to build a visual solution using regular meshes, one could separate the z component of the mesh from the x and y components. The z component could then be stored in a regular data structure, like a texture. The remaining 2D mesh could then be sewed together by copies of one small, regular mesh. A mesh of 1000 by 1000 vertices could be composed of 10 by 10 copies of a mesh of 100 by 100 vertices. In modern graphics applications this can be done using only a single memory representation of the 100 by 100 mesh, a technique called instancing. When rendering a terrain, an application can transform copies of the small mesh to different positions in the terrain, and adjust the z-component of the transformed vertices by looking up the elevation in a texture. Since a complete, full resolution representation is only stored for the z-component, and the small mesh has a low memory foot print, the compression is almost a factor three compared to a complete representation of the terrain using a regular mesh. This kind of separation only makes sense for regular meshes, as instancing cannot be used to implement a TIN and the regular structure of a texture does not map to a the vertices of a TIN.

A 2D illustration of the concept of instancing and separation of the z-coordinate from the x- and y-coordinates is presented in figure 8.1. The first line represents the small line object which longer lines will be composed of (the analogue of a small mesh which are used to compose larger meshes), the second line visualizes how several instances of the small line are assembled to represent a longer line (more line segments). The third line shows the assembled, long line. In the 2D case used here, the y-component is separated from the x-component and stored in a separate array. This array is visualized in row four. The last row visualizes the curve (long line) after the z-components have been applied. In a 3D system this would be done by looking up the elevations from textures and applying them to the separate vertices of the meshes in a vertex shader.

The ITED library contains a data model and functionality that an implementation of a source data based visual solution could be based on. Such a solution would probably improve the performance compared to a TIN based viewer because the use of simpler data structures allow for better paging performance. This is amongst other things due to the fact that the raster data can be split up in any size tiles and easily be down sampled, etc. Even more importantly: it would be possible to achieve large time savings since a lot less time would be needed for building visual databases. An important fact is that the quality of the source data would be just as important regardless which approach is chosen.

Figure 8.1 A 2D illustration of instancing and separation of elevation data and geometry.

8.3 Virtual datasets

As described in section 5.3.1, ITED organizes the elevation data in tiles. The elevation data belonging to each tile is stored in a 2D texture. One implication caused by the tiled approach is that when one tile is processed with the purpose of modifying the elevation data, an instance of the shader program does not necessarily have any information of the neighbouring tiles. This causes algorithms that base the changes in one element on the values found in a neighbourhood around the element to break down. Filtering is an example of this. Filtering within a cell works fine, but when a filter is applied to an area containing a border between two tiles, the neighbouring elements cannot be accessed for elements close to the border. This results in different neighbourhood representations and discontinuity in elevations across the border. Because the processing is performed by the GPU, with somewhat inflexible GPU data structures, this is not straight forward to overcome, though not impossible.

One way to implement seamless, virtual textures of an “infinite” size could be to use an array of 2D textures to store all the data. This array could be made available in the shader program. By calculating the index in the array, as well as the texture coordinates for the texture at the index, from the global RF coordinates of an element, the shader program could access any data element in a loaded dataset.

9 Conclusion

Military M&S is often based on a representation of a natural environment. In many cases the quality of synthetic exercises, especially those that involve single soldiers or vehicles in a natural environment, is highly dependent on the quality of the environment representation.

ITED is a tool that provides means to rapidly design or improve the elevation data component of an SNE. If a scenario description specifies a layout of a terrain, ITED can be used to quickly compose

an elevation dataset from parts of real-world elevation data. If an existing scenario has low resolution elevation data, high resolution structures can be blended into the existing low resolution terrain. When performed carefully, the blending process preserves the large structures of the terrain while adding more detail.

ITED has filled a gap in the tool chain available to MSG-071 for developing elevation data for a large, fictitious continent. The blending process enables composition of elevation data from design specifications.

References

[1] R. Smelik, T. Tutenel, K. de Kraker, and R. Bidarra, “A proposal for a procedural terrain modelling framework,” in *Proceedings of Eurographics Virtual Environments Symposium*. The Eurographics Association, 2008.

[2] J. Brosz, F. F. Samavati, and M. C. Sousa, “Terrain synthesis by-example,” in *Proceedings of the first International Conference on Computer Graphics Theory and Applications*, 2006, pp. 122–133.

[3] R. M. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra, “Declarative terrain modeling for military training games,” *International Journal of Computer Games Technology*, vol. 2010, 2010.

[4] A. Torpy. (2010) L3DT website. [Online]. Available: www.bundysoft.com/L3DT

[5] Statens kartverk. (2011) Norge digitalt website. [Online]. Available: http://www.norgedigitalt.no

[6] Global Mapper Software LLC. (2011) Global mapper website. [Online]. Available: http://www.globalmapper.com

[7] A. Lemmers, A. Gerretsen, and M. Kleijhorst, “Missionland: User needs for a virtual continent,” in *Proceedings of the 2010 European Simulation Interoperability Workshop (SIW)*, 2010, 10E-SIW-005.

[8] R. Smelik, K. J. de Kraker, T. Tutenel, and R. Bidarra, “Integrating procedural generation and manual editing of virtual worlds,” in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games - Fifth International Conference on the Foundation of Digital Games*. ACM, 2010, pp. 1–8.

[9] G. J. de Carpentier, “Effective GPU-based synthesis and editing of realistic heightfields,” Master’s thesis, Delft University of Technology, The Netherlands, 2008.

[10] G. J. de Carpentier and R. Bidarra, “Interactive GPU-based procedural heightfield brushes,” in *Proceedings of the 4th International Conference on Foundations of Digital Games*. ACM, 2009, pp. 55–62.

[11] Epic Games. (2011) Unreal developer network (UDN) website. [Online]. Available: http://udn.epicgames.com

[12] ——. (2011) Unreal developer network (UDN) - terrain design: Guidelines and information. [Online]. Available: http://udn.epicgames.com/Three/TerrainDesign.html

[13] DAZ 3D. (2011) DAZ 3D website. [Online]. Available: http://www.daz3d.com

[14] Crytek. (2011) CryENGINE Sandbox 2 Manual. [Online]. Available: http://doc.crymod.com/SandboxManual

[15] F. Musgrave, C. E. Kolb, and R. S. Mace, "The synthesis and rendering of eroded fractal terrains," in *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*. ACM, 1989, pp. 41–50.

[16] B. B. Mandelbrot, *The Fractal Geometry of Nature*. New York, NY, USA: W. H. Freeman and co., 1982.

[17] H.-O. Peitgen and D. Saupe, *The Science of Fractal Images*. New York, NY, USA: Springer-Verlag, 1988.

[18] M. F. Aasen, A. Gerretsen, and S. Skinner, "Environment data for a high fidelity fictitious continent," in *Proceedings of the 2011 Spring Simulation Interoperability Workshop (SIW)*, 2011, 11S-SIW-046.

[19] R. M. Smelik, K. J. de Kraker, and S. A. Groenewegen, "A survey of procedural methods for terrain modelling," in *Proceedings of the CASA Workshop on 3D Advanced Media in Gaming and Simulation (3AMIGAS)*, 2009.

[20] T. J. Ong, R. Saunders, J. Keyser, and J. J. Leggett, "Terrain generation using genetic algorithms," in *Proceedings of the 2005 Conference on Genetic and Evelutionary Computation (GECCO '05)*. ACM, 2005, pp. 1463–1470.

[21] H. Zhou, J. Sun, G. Turk, and J. M. Rehg, "Terrain synthesis from digital elevation models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 4, pp. 834–848, 2007.

[22] O. Šťava, B. Beneš, M. Brisbin, and J. Křivánek, "Interactive terrain modelling using hydraulic erosion," in *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Dublin, Ireland: Eurographics Association, 2008, pp. 201–210.

[23] J. H. Tomkin, "Numerically simulating alpine landscapes: The geomorphologic consequences of incorporating glacial erosion in surface process models," *Geomorphology*, vol. 103, no. 2, pp. 180–188, 2009.

[24] T. Akenine-Møller, E. Haines, and N. Hoffman, *Real-Time Rendering*, 3rd ed. Wellesley, MA, USA: A K Peters, Ltd., 2008.

[25] Khronos Group. (2011) OpenGL website. [Online]. Available: http://www.opengl.org

[26] Microsoft. (2009) DirectX. [Online]. Available: http://www.microsoft.com/games/en-US/aboutGFW/pages/directx.aspx

[27] NVIDIA Corporation. (2011) Nvidia website. [Online]. Available: http://www.nvidia.com

[28] Intel Corporation. (2011) Intel website. [Online]. Available: http://www.intel.com

[29] NVIDIA Corporation. (2011) What is CUDA? [Online]. Available: http://www.nvidia.com/object/what_is_cuda_new.html

[30] M. F. Aasen, “Bildegeneratorer - visualiseringsverktøy for bruk innen modellering og simulering,” Forsvarets forskningsinstitutt, FFI-notat 2010/00758, 2010.

[31] R. J. Rost and B. Licea-Kane, *OpenGL Shading Language*, 3rd ed. Boston, MA, USA: Pearson Education, Inc., 2010.

[32] NVIDIA Corporation, “Nvidia’s next generation CUDA compute architecture: Fermi,” NVIDIA Corporation., Tech. Rep., 2009.

[33] IEEE, “IEEE standard for floating-point aritmetic,” IEEE, IEEE Std 754-2008, 2008.

[34] OSG Community. (2011) OpenSceneGraph (OSG) website. [Online]. Available: http://www.openscenegraph.org

[35] J. Kessenich. (2009) The OpenGL Shading Language. [Online]. Available: www.opengl.org

[36] Open Source Geospatial Foundation. (2011) Geospatial Data Abstraction Library (GDAL) website. [Online]. Available: http://www.gdal.org

[37] Presagis. (2011) Preasagis website. [Online]. Available: http://www.presagis.com

[38] US Geological Survey (USGS). (2010) USGS home page. [Online]. Available: http://www.usgs.gov

Abbreviations

AI	Artificial Intelligence
ALU	Arithmetic Logic Unit
API	Application Programming Interface
Cg	C for Graphics
CDB	Common Database
COTS	Commercial off the shelf
CUDA	Compute Unified Device Architecture
DEM	Digital Elevation Model
DTED	Digital Terrain Elevation Data
fBm	Fractional Brownian motion
FMA	Fused Multiply-Add
FMGT	Norwegian Military Geographic Service
FO	FilterOperator
FPS	First Person Shooter
FPU	Floating Point Unit
GDAL	Geospatial Data Abstraction Library
GeoTIFF	Geo Tag Image File Format
GLSL	OpenGL Shading Language
GPGPU	General Purpose GPU Programming
GPL	GNU General Public License
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HITL	Human-in-the-loop
HLSL	High Level Shading Language
HT	Height Tile
I/O	Input/Output
ITED	Interactive Terrain Editor
L3DT	Large 3D Terrain Generator
LGPL	GNU Lesser General Public License
LOD	Level of Detail
LOS	Line of Sight
M&S	Modelling and Simulation
OSG	OpenSceneGraph
PE	Processing Engine
PO	Processable Object
RF	Reference Frame
RTO	NATO Research and Technology Organization
SM	Streaming Multiprocessor
SNE	Synthetic Natural Environment

TIN	Triangulated Irregular Network
USGS	US Geological Survey
USGS DEM	USGS Digital Elevation Model
VBS2	Virtual Battle Space 2
VE	Visualization Engine