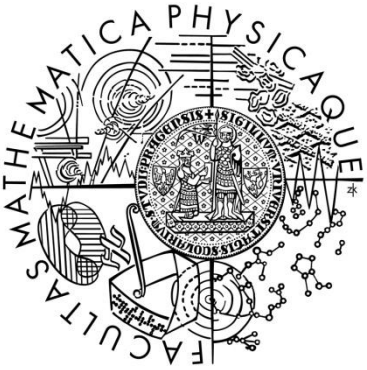
Charles University in Prague

Faculty of Mathematics and Physics

BACHELOR THESIS



Matěj Zábský

GeoGen - Scriptable generator of terrain height maps

Department of Software and Computer Science Education

Supervisor of the bachelor thesis: Mgr. Martin Petříček

Study programme: Computer Science

Specialization: Programming

Prague 2011

I would like to thank my supervisor, Mgr. Martin Petříček, for valuable and helpful advice, and for smooth supervision.

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In…..... date..... signature

Název práce: GeoGen – Skriptovatelný generátor výškových map terénu

Autor: Matěj Zábský

Katedra / Ústav: Kabinet software a výuky informatiky

Vedoucí bakalářské práce: Mgr. Martin Petříček, Kabinet software a výuky informatiky

Abstrakt: Tato práce představuje skriptovatelný generátor výškových map terénu. Skripty pro tento generátor jsou psány v skriptovacím jazyce podobném jazyku C. Nástroje použitelné při generování terénu zahrnují generátory náhodného šumu, filtry, kombinační funkce, maskovací funkce a simulátory vodní a tepelné eroze. Grafické vývojové prostředí integrující tento generátor s moderním editorem kódu, 3D prohlížečem výškových map a dalšími nástroji je také představeno. Existující programy pro vytváření terénu jsou analyzovány. Probrány jsou různé algoritmy použité při generování výškových map. Také jsou prezentovány ukázkové skripty napsané pro tento generátor.

Klíčová slova: Výšková mapa, terén, generátor, eroze

Title: GeoGen - Scriptable generator of terrain height maps

Author: Matěj Zábský

Department / Institute: Department of Software and Computer Science Education

Supervisor of the bachelor thesis: Mgr. Martin Petříček, Department of Software and Computer Science Education

Abstract: This thesis presents a scriptable terrain height map generator. Its scripts are written in a powerful C-like scripting language. Terrain generation tools available include random noise generators, filters, combiners, masking functions and natural erosion simulators. Graphical script development environment which integrates the generator with modern code editor, interactive 3D height map viewer and other tools is presented. Existing terrain creation tools are analyzed. Various algorithms used in terrain generation are discussed. Example scripts written for this generator are presented.

Keywords: Height map, terrain, generator, erosion

Contents

Introduction

Creation and rendering of virtual terrains are some of the most interesting topics in computer graphics. Usage of these terrains ranges from high-budget movies to video games. As varied as uses are requirements of these applications – landscape in a movie has to be beautiful, but believable. On the other hand, terrain in a computer game has to be adapted to the game's rules and has to be challenging and fair to all the players. Features of terrain generation tools have to reflect these varying requirements.

One of the oldest and most widely used representations of a terrain in computer memory is a height map – a bitmap where each pixel represents height of the terrain in corresponding coordinate. This thesis implements a complete solution for fully procedural generation of height maps. Advantages and disadvantages of other representations are discussed.

Terrain generation process in GeoGen is controlled by a text script written in a C-like scripting language. Vast array of terrain generation and modification tools is available to the scripts. However the scripts have great freedom in what they do – low-level pixel data access is also possible.

Unlike existing terrain generation tools, this generator is designed as a library, which can be integrated into other applications and games. Focus is put on simplicity and variability of this integration API.

This thesis also introduces a graphical script development environment, which offers all necessary tools to write and test map scripts, including modern code editor and interactive 3D viewer of generated terrains. Height maps can also be imported from hard drive and explored in this application.

Chapter 1 analyses some of the existing terrain creation applications. Chapter 2 outlines basic parts of GeoGen. Chapter 3 introduces most important ideas of this project's approach to terrain generation. Chapter 4 details its height map generation pipeline. Chapter 5 discusses various terrain generation and modification algorithms.

Chapter 6 presents some map scripts created for GeoGen. Chapter 7 discusses potential improvements of the current implementation.

Appendix A contains detailed user manual of GeoGen. Appendix B is devoted to its programmer documentation. Contents of the attached CD are listed Appendix C.

1. Review of existing projects

There are many applications which for some purpose generate height maps (or terrains in general), but only a limited number of them offer a complete suite of terrain creation tools (none of them are freeware). This is not a full list of all terrain creation programs; these are only representatives of various categories of existing applications.

1.1. World Machine

World Machine (1) is a terrain creation tool written by Stephen Schmitt. It is a commercial application (single license costs $89 - $189), however very limited free version is available.

World Machine's philosophy is from existing applications the closest to GeoGen's. Rules for terrain generation are defined by wiring devices which represent individual generation and filtering functions. Input plugs allow detailed customization of device's behavior. Output plugs then provide device's output. Devices are placed and wired using a drag-and-drop user interface.

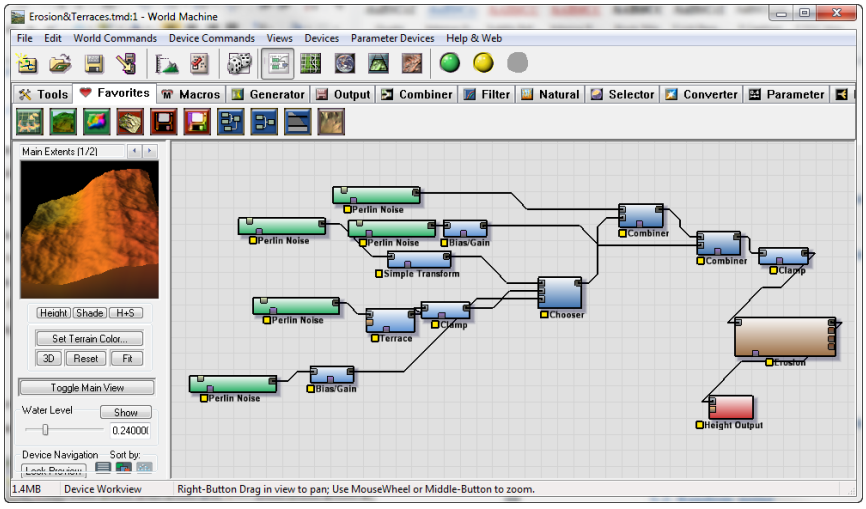


Figure 1: World Machine device network editor.

The device network can then be repeatedly executed to generate terrains. These terrains can be saved to disk or explored using 2D and 3D viewers.

The selection of devices is very wide – it ranges from simple constant height generator or Perlin noise generator to sophisticated natural phenomena simulators. These include hydraulic erosion, thermal weathering, snow cover and coastal erosion simulators.

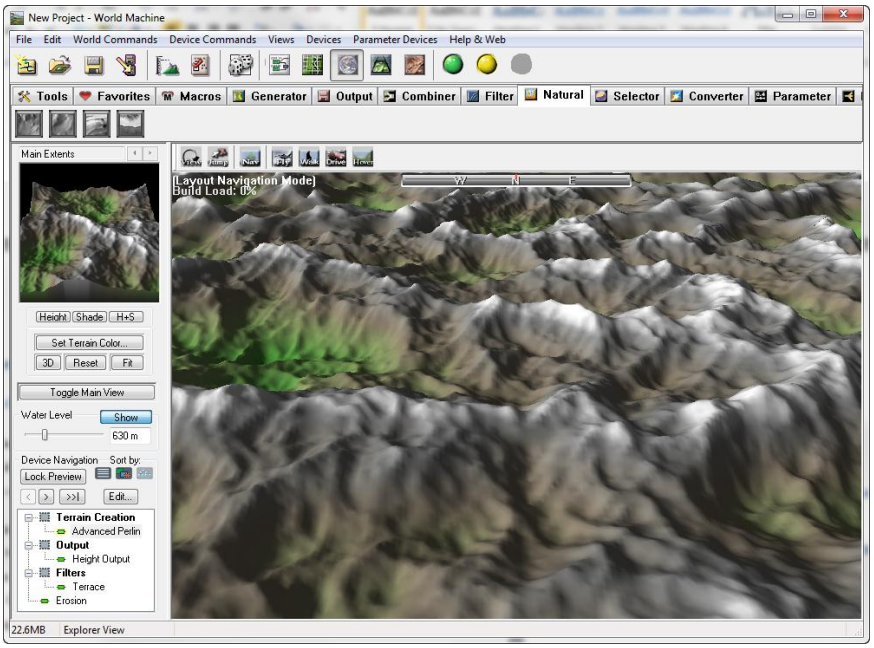


Figure 2: World Machine 3D viewer.

Latest version of World Machine also allows manual definition of major terrain features like mountains or canyons; however focus of the application is still clearly in purely procedural terrain generation.

Notable feature of World Machine is generation of infinite terrains – the rules defined with the device network generate infinite world, from which the user can select a region (or multiple regions) for export.

1.2. GeoControl

GeoControl (2) is an application developed by Johannes Rosenberg. Single license costs $129.

Terrain generation starts with simple fractal noise. Then a fully customizable sequence of filters from a wide selection including multiple erosion simulators is applied. Most filters can have their effect limited based on local terrain properties

(such as height, orientation, slope, roughness etc.) or by a mask produced by another filter.

The finished terrain can be rendered with inbuilt renderer or exported into a variety of image, height data and 3D model formats.

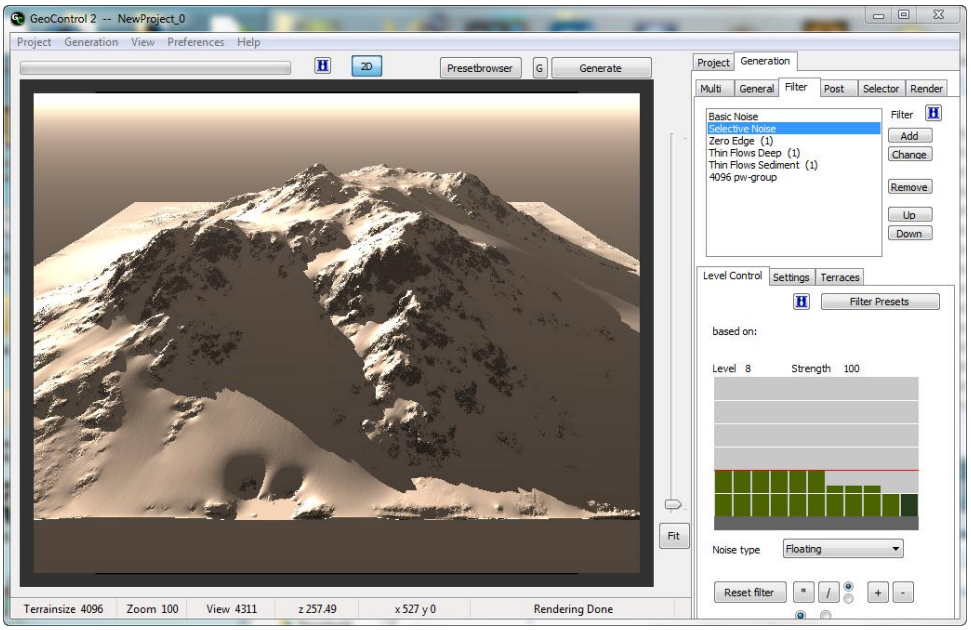


Figure 3: GeoControl 2.

1.3. DAZ 3D Bryce

Bryce (3) is complete world creation and rendering application currently developed by DAZ 3D. Single license costs $99.95 - $199.95; however a Personal Learning Edition is free for non-commercial use.

Its terrain editor features a fractal generator with several filters as well as an erosion tool. Large focus is put on manual painting tools – terrain is typically created by painting approximate terrain shape with brushes and then applying some filters to give it a natural look.

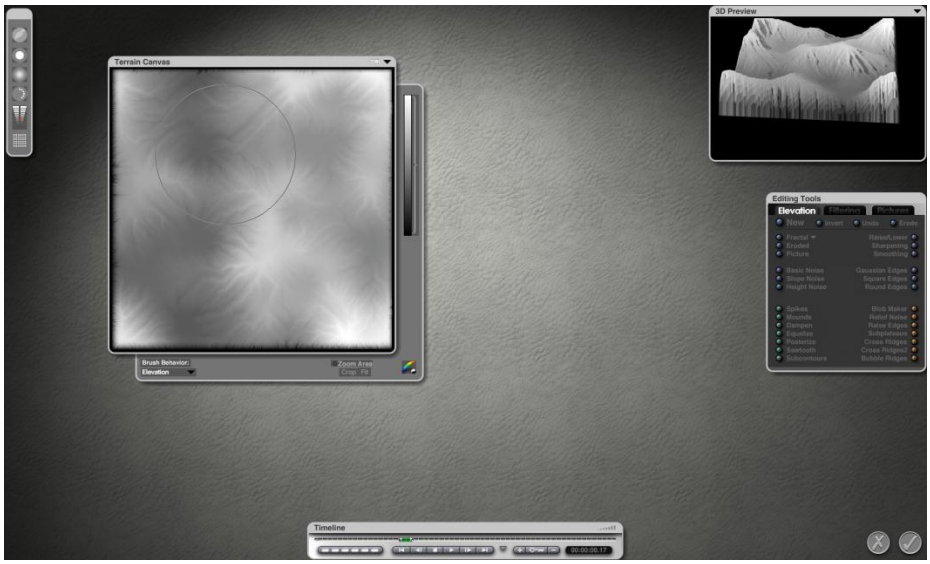


Figure 4: Bryce terrain editor.

Once the terrain is created, main window of the application allows filling it with objects like water surfaces, lights and trees. Modeling and importing tools are included, so the objects can be completely arbitrary. Complex materials can be assigned to terrain and other objects. Animation is supported for all aspects of the world, including shape of the terrain.

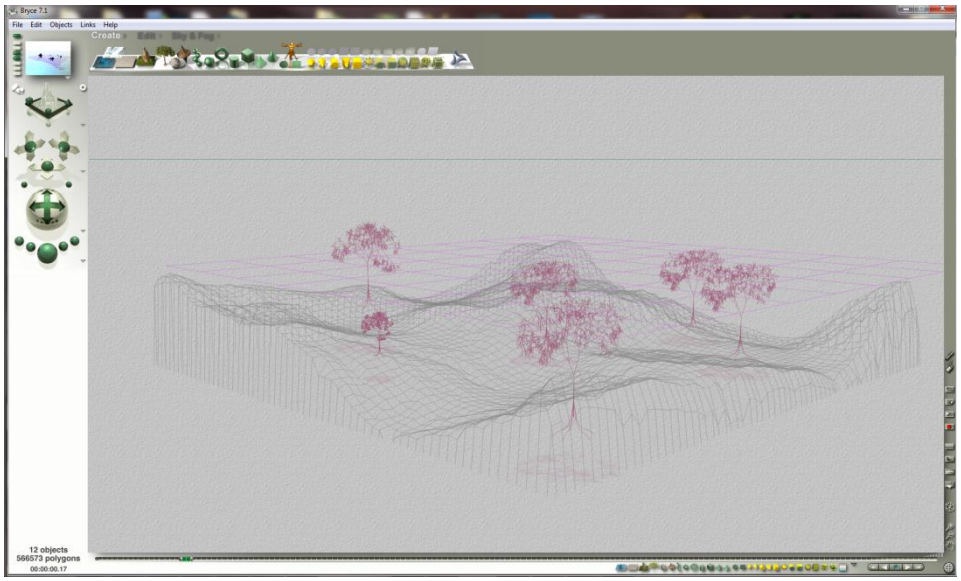


Figure 5: Bryce world editor.

2. Structure of the project

GeoGen is composed of three parts: core library, console interface and Studio – a graphical IDE.

This chapter is only very brief overview of the individual parts. Detailed description of these components can be found in following chapters and appendixes.

2.1. Core library

The generator itself is designed as a library, which can be implemented into other applications. This library contains all terrain generation logic and an encapsulating generator class. The generator allows easy configuration and execution of scripts. Height maps are always returned as raw height data.

The library is written in C++, but a binding class library for .NET is available.

See appendix B.1. for details about the core library.

2.2. Console interface

Console interface is a simple application implementing the core library. All the necessary information is passed into it via command line options, including script parameters.

Generated height maps can be saved on hard drive as BMP (Windows bitmap), PGM (portable grey map) or SHD[1] files. The height maps can be also colorized using overlays (see appendix A.2.).

See appendix A.3. for detailed console interface manual.

[1] SHD (short height data) is a lossless native file format for GeoGen height maps.

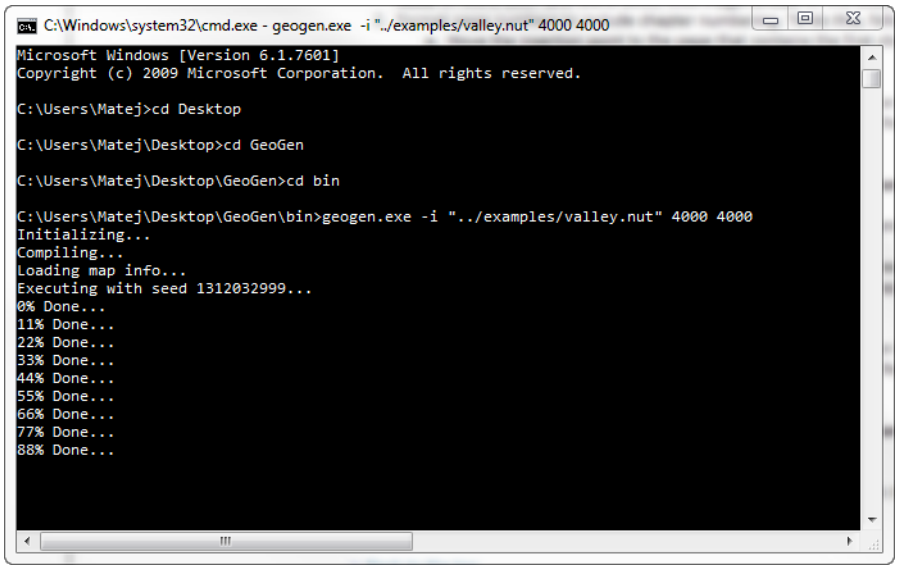


Figure 6: GeoGen console interface.

2.3. Studio

GeoGen Studio is full-featured graphical script development environment. It has modern code editor with syntax highlighting, code completion and real-time script validation. The scripts can be executed directly from the application. Generated maps can be explored in 2D and 3D views.

See appendix A.4. for detailed GeoGen Studio manual.

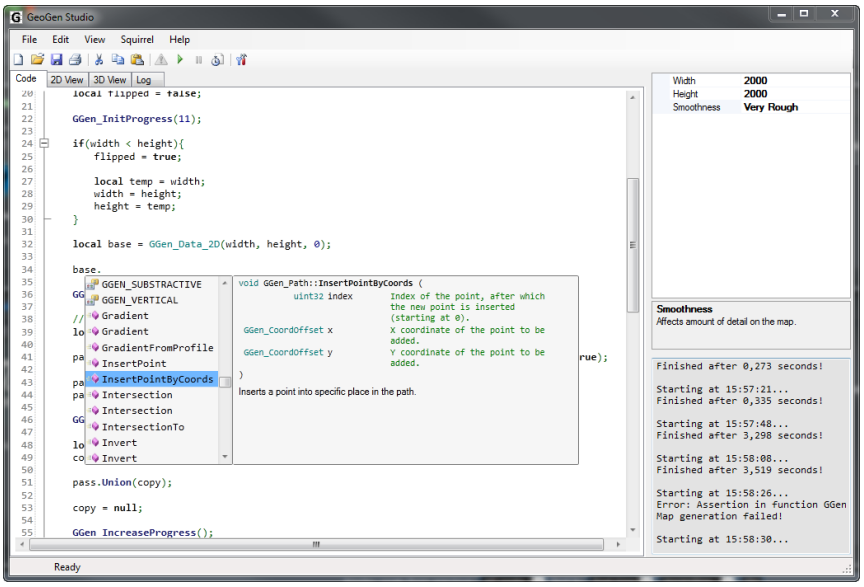


Figure 7: GeoGen Studio code editor.

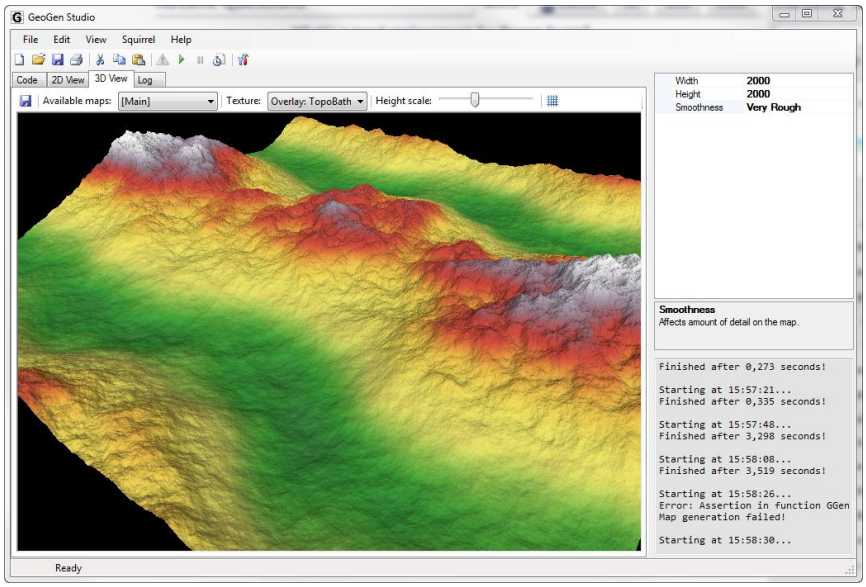


Figure 8: GeoGen Studio 3D view.

3. Terrain generation philosophy

This chapter outlines the most important ideas of GeoGen's approach to terrain generation.

3.1. Terrain representation

There are multiple ways terrain can be represented in computer memory:

- **Height map** – the terrain is represented as 2D array of height values. This representation is easy to convert into any other representation. It can also be stored using common image formats such as BMP or PNG.
- **Layered arrays** – terrain is represented as multiple arrays of height values. Each array represents heights of material in one layer. Total height of the terrain is then the sum of heights in all the layers in given pixel. This representation requires considerably more memory than height map of the same size. However the ability to compose single terrain of multiple materials may allow for more realistic physical simulations (4).
- **Voxel array** – the terrain is stored as 3D array of voxels (3D pixels). Though each voxel can be stored as a single bit, this representation has by orders of magnitude higher memory requirements than height map of the same size and height range. Unlike height maps, this representation can store completely arbitrary terrain features including caves and overhangs.
- **Vector representation** – the terrain is represented as a 3D model composed of vertices, edges and faces. This allows terrain features impossible with height maps including caves and overhangs. Another advantage is that empty areas or very large flat surfaces occupy very little to none memory. The disadvantage is that otherwise very simple algorithms that work with height map may be considerably more complex or even impossible with this representation.

GeoGen uses the height map representation, because it is memory efficient, well supported in 3rd party software and easy to work with. The heights are represented as signed 16bit integers. 16bit integers offer good compromise between value range and memory requirements. Floating point height representation was rejected because of

its inefficiency – because constant precision across whole value range is needed, only a small subset of supported values could be used (such as [-1, 1] range).

3.2. Purely procedural generation

The height map generation process is purely procedural in GeoGen – no existing graphical sources are used during the process. While synthesis of new terrains from existing height maps is possible (5), main objective of this project is to generate random maps. Non-procedural data sources on the other hand tend to remove randomness.

Therefore the only data inputs are scalar script parameters (see section 3.4.).

However all randomness is only psedo-randomness. The generator is fully deterministic – two maps generated with the same generator version, script, parameters and random seed are guaranteed to be the same.

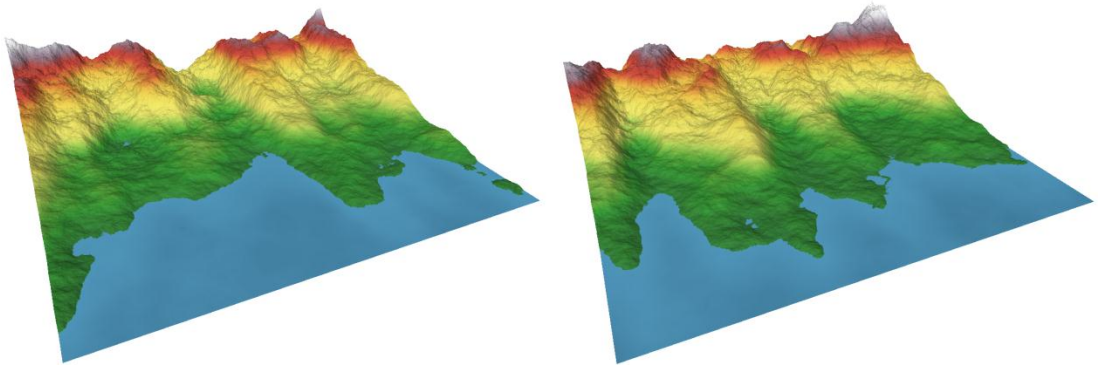


Figure 9: Two maps generated with the same script and parameters, but different random seeds.

3.3. Creating complex terrains

Complex terrains are created using a sequence of terrain generation, modification and composition functions. Along with the 2D data arrays (which represent height maps and any other 2D data) the scripts can work with 1D data arrays, integers, floating point numbers and strings.

This philosophy is illustrated in Figure 10 (which is a diagram of “valley.nut” example found on the attached CD, see Appendix C).

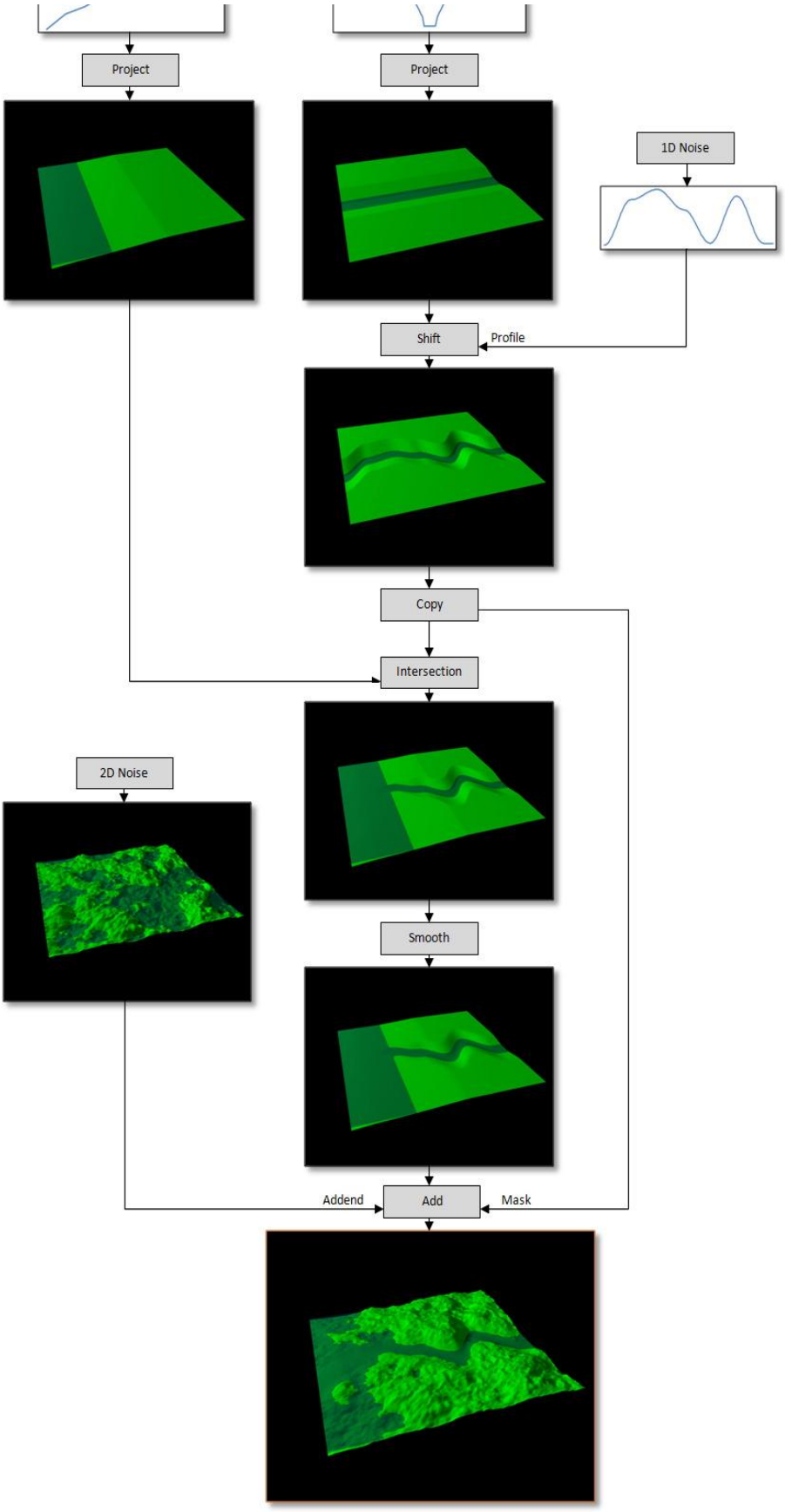


Figure 10: Complex map script diagram.

3.4. Script parameterization

Map script may define any number of parameters. These parameters can be of three types:

- **Integer** – value is an integer in specified range.
- **Boolean** – value is either true or false.
- **Enumeration** – value is one from a specified list of strings.

Each parameter has also its name, title, description and a default value.

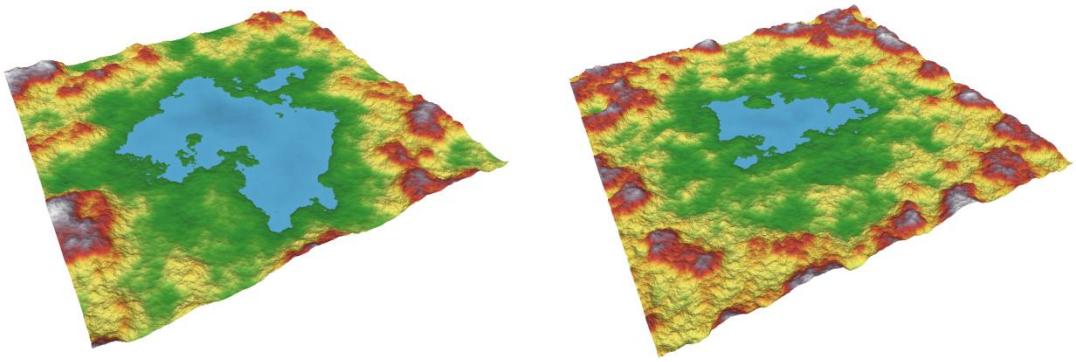


Figure 11: Maps generated by a single script with different parameter values.

All the information is made available to the hosting application, which can then for example use it to build a graphical configuration interface with sliders, checkboxes, selection boxes etc.

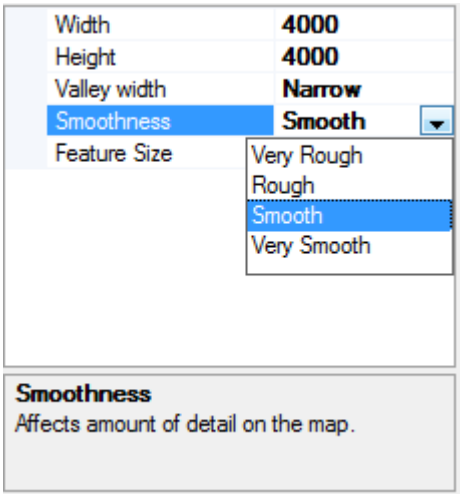


Figure 12: Script configuration UI as found in GeoGen Studio.

3.5. Returning multiple maps

GeoGen script can go further than just generating a terrain height map. A map script can return any number of named secondary maps along with exactly one primary map.

These maps may contain any 2D data; their interpretation depends on the hosting application – for example GeoGen's console interface saves them directly onto hard drive along with the primary map, while Studio offers to use them as a texture for the 3D model.

4. Terrain generation pipeline

This chapter will outline what GeoGen does from the moment it is fed with a map script until the script finishes executing and an output map is successfully returned.

4.1. Script compilation

Once a map script is passed into GeoGen, Squirrel immediately attempts to compile it. If there are any errors, the generation is terminated and an error message is returned.

4.2. Header execution

If the compilation is successful, the hosting application can ask the script for various metadata such as script name, description and author. Important parts of the metadata are map parameters. The metadata are located in a special part of the script called header.

If there were no errors during header execution, the hosting application can configure the script by adjusting its parameters.

4.3. Body execution

Once the parameters are loaded, body of the script can be executed any number of times. Body is a part of the script which contains the terrain generation logic.

The body will (if it succeeds) always return at least one map.

Any error during its phase will cancel the generation, but it won't unload the script – the script can be executed again without compiling it or reloading the metadata (though it will most likely fail again, unless e.g. the parameters are changed).

5. Algorithms used in terrain generation

This chapter will detail some of the more interesting algorithms implemented in terrain generation functions.

5.1. Interpolation

Interpolation algorithms are used to calculate values between other values or for resizing data sets (in case of GeoGen both 1D and 2D).

There are multiple interpolation algorithms used in GeoGen in various contexts.

For downsizing, the simple nearest neighbor algorithm is used in both 1D and 2D cases.

For upsizing in 1D, cubic interpolation is used. Both linear interpolation and cosine interpolation have proven to cause significant artifacts in typical scenarios in which 1D data sets are typically used.

On the other hand, these artifacts don't generally cause issues in 2D, so simple bilinear interpolation is used for upsizing in 2D.

Some algorithms have more specific needs and employ different interpolation algorithms – for example random noise generator uses bicubic interpolation, since linear interpolation in its context caused major artifacts (see chapter 5.2.).

5.2. Random noise

Diamond-square algorithm (6) is used as a basic random terrain generation function.

GeoGen uses diamond-square algorithm instead of common Perlin noise because it is less prone to generate axially-oriented terrain features such as valleys and mountain ranges.

First, an initial grid of points spaced by initial wave length w_0 is generated with random value range $a(0)$ where a is a function which assigns amplitude to each recursion level.

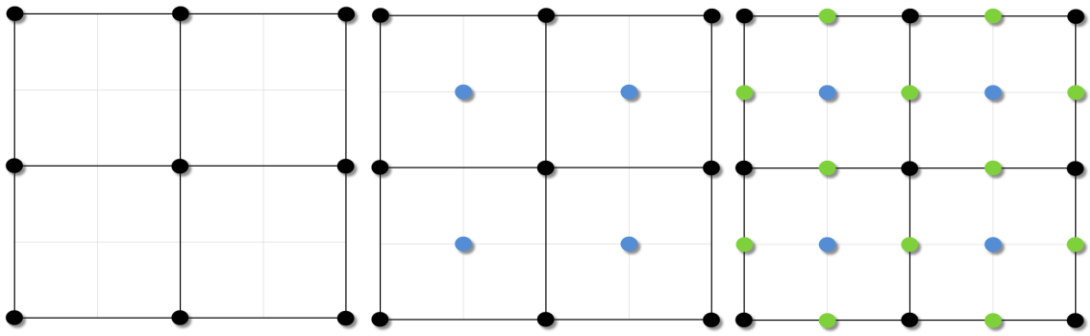


Figure 13: Diamond-square algorithm after initial grid generation (left), after first square step (middle) and after first diamond step (right).

Then a new point is generated in middle of each square formed by 4 previously generated points (this is called the "square step"; blue dots in Figure 13). Then another new point is generated in middle of each square rotated by 45° formed by two points from the initial step and two points from the first square step (this is called the "diamond step"; green dots in Figure 13).

Square step is then repeated on newly created squares, followed by another diamond step. This process is repeated until every pixel in the map is filled.

Important part of any coherent noise algorithm is an interpolation function – interpolation is required to express dependency of a random value on surrounding random values. Unlike Perlin noise, for which is bilinear interpolation generally sufficient (though more complex interpolation algorithms may improve noise quality), diamond-square algorithm with bilinear interpolation generates very significant artifacts in form of spikes and sinkholes in local extremes (see Figure 14).

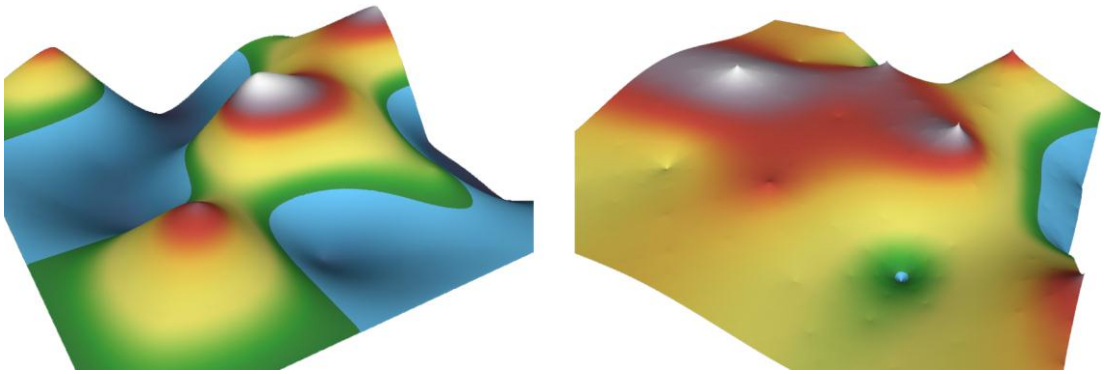


Figure 14: Single-layer noise generated with diamond-square algorithm with bicubic interpolation (left) and with bilinear interpolation (right).

The same applies for most other 4-sample interpolation functions (such as bicosine interpolation), because the interpolated point in diamond-square algorithm is always[2] in exact center of 4 other points and 4-sample interpolation functions in such case tend to return arithmetic mean of the 4 samples (all the samples have the exact same weight). Therefore a higher-degree function has to be used, such as bicubic interpolation, which uses 16 samples to calculate its value.

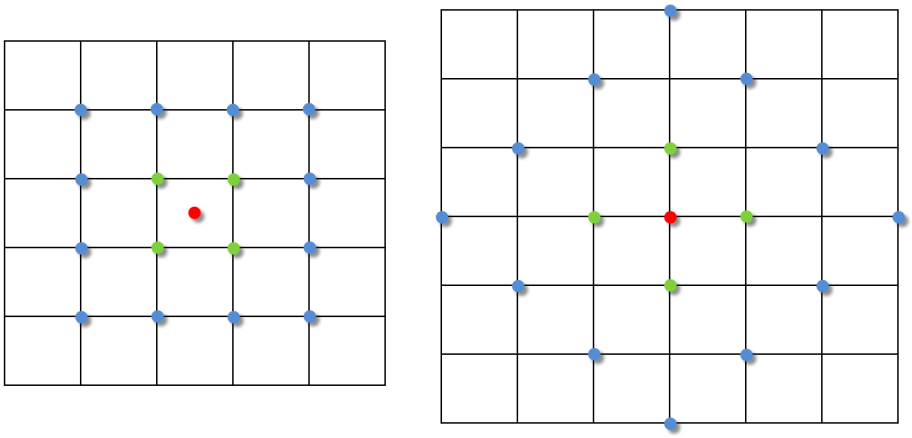


Figure 15: Samples for interpolation in red point with bilinear 4-sample interpolation (green points) and with bicubic 16-sample interpolation (green and blue points). Left grid is sample arrangement during square step, right grid is sample arrangement during diamond step.

GeoGen uses coefficients by (7) to solve the bicubic polynomial.

Major issue with diamond-square algorithm is that it by default handles only maps with dimensions $n + 1 \times n + 1$ where n is some power of 2. GeoGen overcomes this issue by keeping a single vertical and a single horizontal buffer one pixel-wide wide/high respectively and as high/wide respectively as the height map. Then if the interpolation algorithm would have to pick a sample from the missing area (or write into it), it uses the appropriate buffer instead.

5.3. Voronoi noise

Voronoi noise is another random height map generation algorithm. A number of points are randomly distributed into the height map area. Height in each pixel is then calculated as a value of distance mapping function $f(d_1, d_2, \dots)$ where d_1 is distance to the nearest point, d_2 is distance to the second nearest point, etc.

[2] Possibly excluding border cases.

GeoGen distributes the points regularly into cells of a uniform grid to improve regularity of the resulting height map as proposed in (8). However the size of the cells is configurable, so irregular noise can still be achieved by setting the cell size to the height map's size.

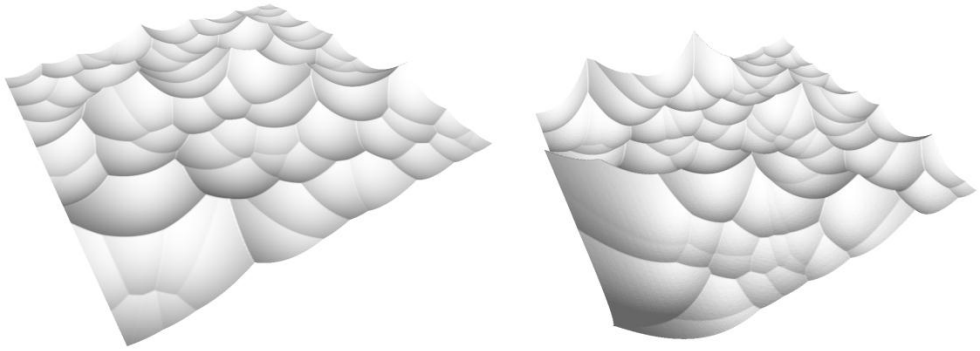


Figure 16: Comparison of regular (left) and irregular (right) Voronoi noise.

For each pixel in the height map, only distances to points in its cell and 8 adjacent cells have to be calculated (since maximum number of parameters of implemented distance mapping functions is 2 and each cell contains at least one point, then at least nine points will be found in these cells, excluding border cases where at least three points will be found).

GeoGen implements only two simple distance mapping functions:

- $f_1(d_1) = d_1$
- $f_2(d_1, d_2) = -d_1 + d_2$

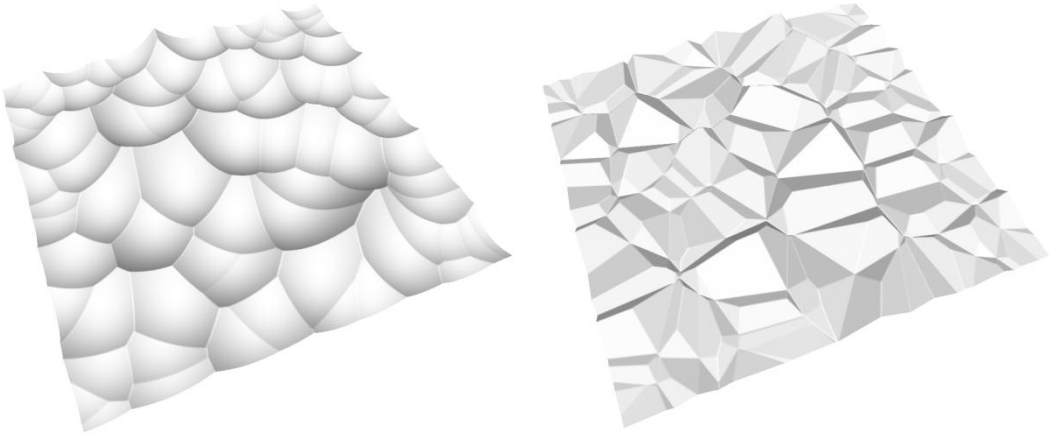


Figure 17: Distance mapping functions 1 (left) and 2 (right).

Optimization proposed by (9) is used: following non-Euclidean distance metric is used instead of regular Euclidean distance, which significantly reduces computational cost of the calculation by removing square root from the metric:

$$d = (x_1 - x_2) + (y_1 - y_2)$$

5.4. Value gradient

GeoGen implements two basic types of value gradients:

- **Linear gradien**t – values progress in a strip between two points.
- **Radial gradient** – values progress in all directions from a single point.

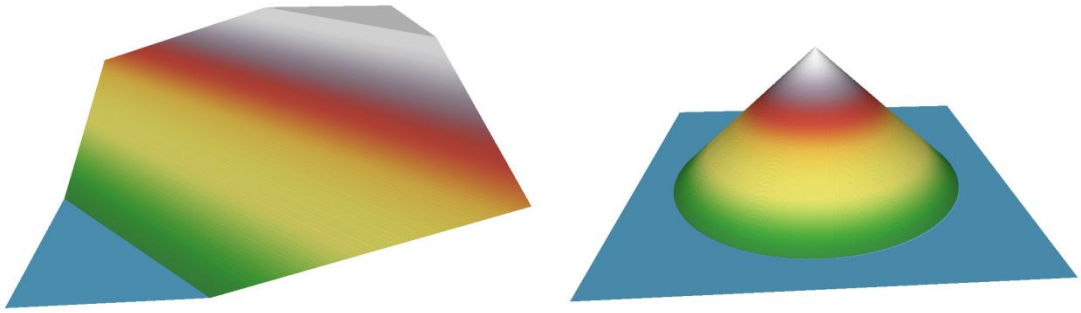


Figure 18: Linear gradient (left) and radial gradient (right).

Linear gradient value in a pixel is calculated as a distance from one of the ending points to intersection point between the line connecting the two ending points and its normal going through the pixel.

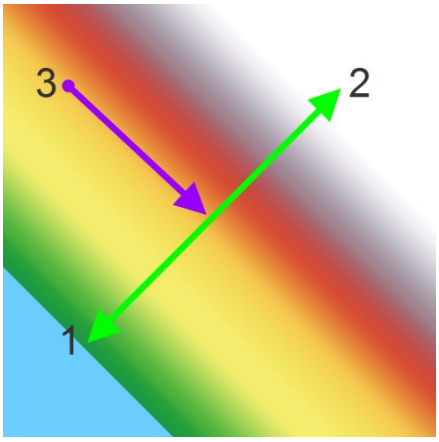


Figure 19: Linear gradient with its starting point (1) and ending point (2). 3 is the current point for which is the value being calculated.

Radial gradient value in a pixel is trivial – it is a distance to the central point.

5.5. Smoothing

Smoothing is often necessary to reduce roughness in the map or to wash away hard edges produced by other functions. Sliding window box blur algorithm (10) is used.

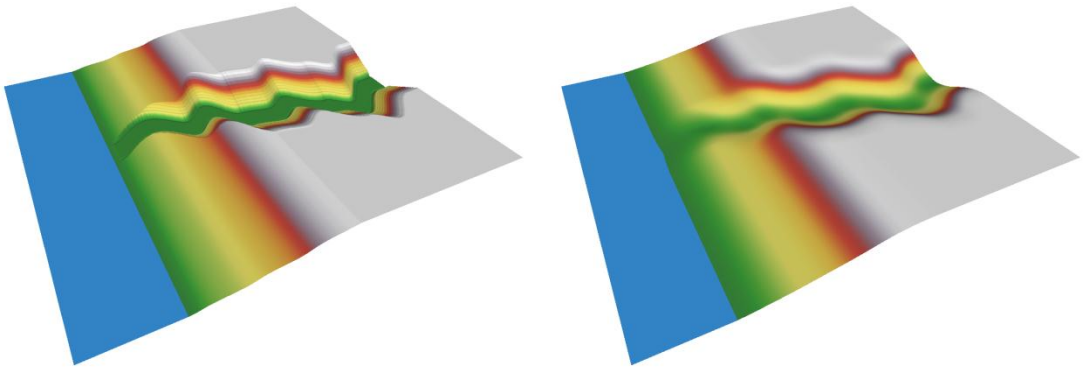


Figure 20: Terrain before (left) and after (right) smoothing.

Box blur in 2D can be separated into two 1D passes – the first pass blurs the image horizontally and the second blurs the horizontally blurred image vertically, resulting in image which is equivalent to an image blurred with a single 2D box blur pass.

The smoothing is done individually for each row/column (depending on direction of the blurring pass) with a 1D sliding window. In each step is value in front of the window added to its accumulated value and the last value in the window is subtracted from its accumulated value.

Value of each pixel is $accumulated\_window\_value/window\_size$ when the window is centered on that pixel. This also correctly handles border cases – the window is smaller if its portion would be outside of the image area.

Thanks to this algorithm, performance of the smoothing function doesn't depend on blur radius at all.

5.6. Linear transformations

Linear matrix transformations are used for rotations and shearing of height maps (scaling and flipping is done with specialized routines).

Straightforward approach would be to simply multiply coordinate of each pixel in the height map with the transformation matrix. However transformed coordinates could be non-integral, resulting in either blank pixels (if the coordinates were rounded to nearest whole number) or having to be split between multiple result pixels.

This is fixed with inverse approach. First are calculated boundaries of the transformed map by multiplying each of the corner points with the transformation matrix. Then each transformed map's pixel's coordinate is multiplied with an inverse transformation matrix resulting in a potentially non-integral source coordinate. Reading from non-integral coordinate can be easily handled by using interpolation algorithm (in this case bilinear interpolation as stated in chapter 5.1.).

Downside to this approach is that it can't be used if the transformation matrix is singular. However very few legitimately useful 2D transformation matrices are singular; therefore this is not a limiting issue.

5.7. Convexity map

Convexity map is a bitmap which contains information whether an area in a height map is convex or concave (and how convex/concave it is).

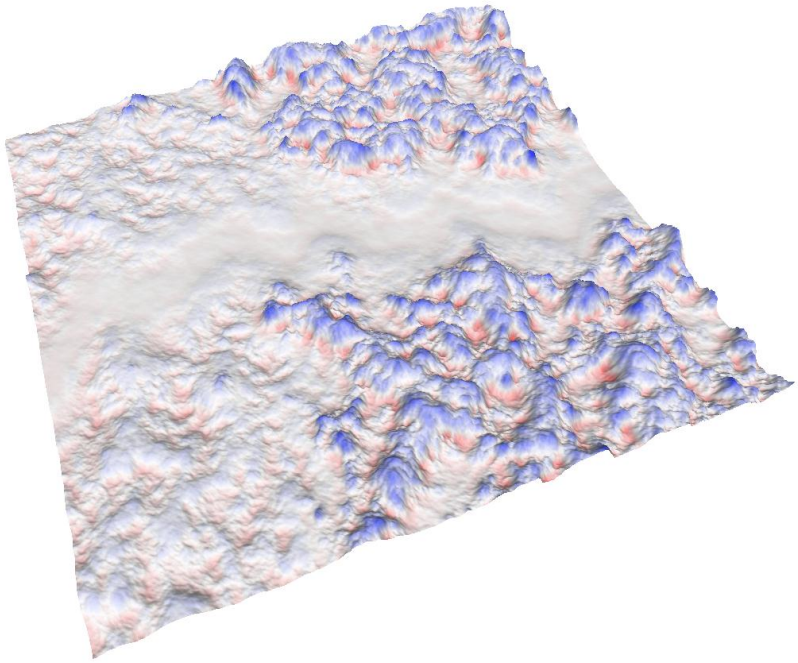


Figure 21: Terrain with its convexity map - convex areas are blue, concave areas are red.

The information in each pixel actually says “how far and in which direction is height of the pixel from other pixels in its area.” This already answers how the convexity map be can constructed: $convexity\_map(map) = smooth(map) - map$ where “-” means per-value subtraction.

5.8. Path stroke

Path stroke algorithm calculates distance from each pixel to a path defined by a sequence of points. A height is then assigned to each pixel based on this distance.

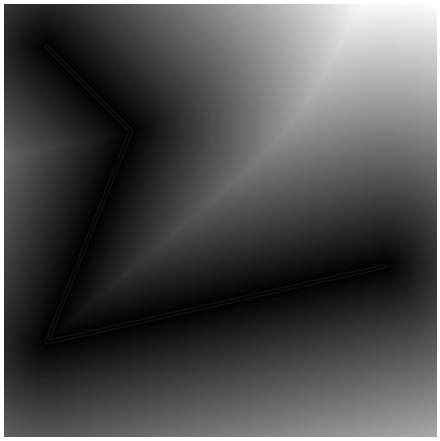


Figure 22: Result of path stroke; the path is highlighted.

Naïve algorithm would loop over all edges in the path in each pixel of the map to find the one to which is the pixel closest and then calculate the distance. This approach however results in extremely poor performance.

A recursive divide and conquer algorithm is used instead. The algorithm starts with whole map: for each of the corners of the map the closest path edge is calculated (by looping through all edges of the path).

If the result is the same for all four corners, the assumption that this applies to all points within the map is made. Then the distance in each point can be calculated with this one edge instead of looping over all the edges.

If at least one of the corners has different result, the area is divided into four parts and the algorithm is recursively executed on the four parts.

When the algorithm is executed on 1×1 pixel area, the recursion stops (distance to all the path edges has to be calculated to find the closest one).

There is an issue which might potentially cause artifacts (see Figure 23): if there are two consecutive path edges with the same distance to a point, angle between the two path segments and the pixel has to be taken into consideration when deciding which edge to use to ensure that the area is divided correctly between the two edges.

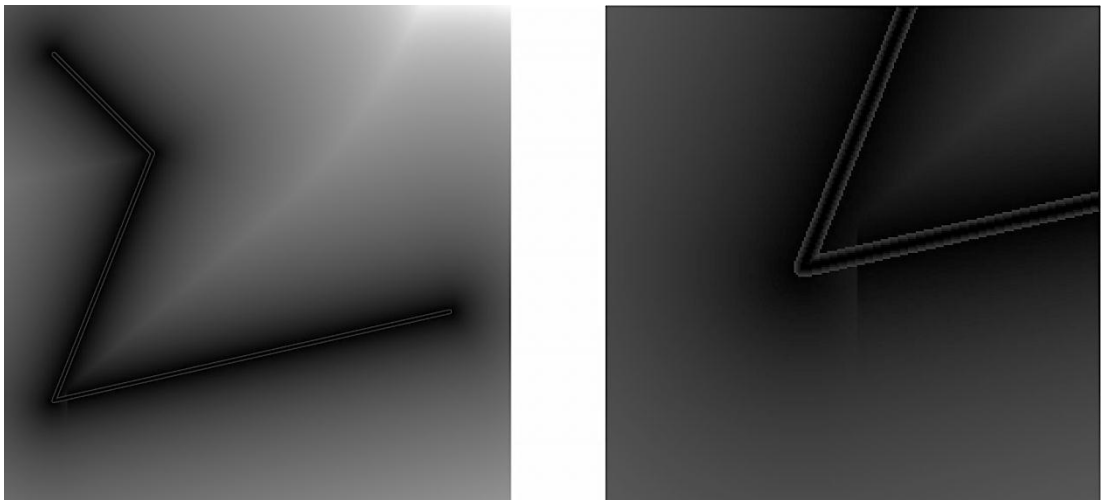


Figure 23: Stroke artifact caused by ambiguity of closest point calculation on the example from Figure 22 (left) and a detail of the artifact (right); brightness of this picture was increased to improve its clarity.

This algorithm may fail when the path crosses itself. This is obviously caused by the fact that the closest path edge is always determined only in edges of sometimes very large areas. However since the stroke function is designed primarily for creation of valleys and mountain ranges, it is considered to be an acceptable drawback.

5.9. Hydraulic erosion

Hydraulic erosion is a process where material from the terrain is being eroded by running water and transported by the stream until it can be deposited somewhere else.

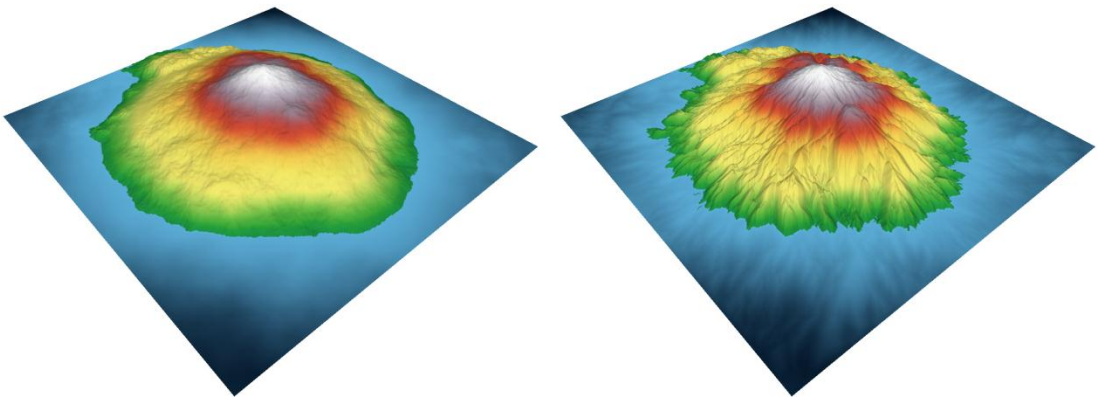


Figure 24: Terrain before (left) and after (right) hydraulic and thermal erosion.

GeoGen's hydraulic erosion model is based on algorithm proposed by (11) and also implements some of the improvements by (12).

Each iteration of the algorithm consists of following steps:

1. Add new water to the system.
2. Simulate flow of water over the terrain.
3. Simulate erosion and deposition of material currently suspended in water.
4. Simulate transportation of suspended material by the flow.
5. Remove some water from the system due to evaporation.

This algorithm uses virtual pipe model. Each pixel in the map is connected with its four neighbors by virtual pipes. The pipes are represented as outflow flux values – each pixel has a set of four outflow values, total flow between two neighboring pixels can be calculated as difference of the two outflow flux values of the pipe that connects them.

These are data layers used by the algorithm:

- **Height map**
- **Water map** – current amount of water in each pixel.
- **Flux map** – current outflow flux value for each pixel (4 directions per pixel).
- **Velocity field** – flux map represented as a velocity vector in each pixel (each vector has X and Y component).
- **Sediment map** – current amount of sediment suspended in water in each pixel.

All these maps except the height map are initialized to zero when the algorithm starts. Zero level is not interpreted as water level by this algorithm (it is completely ignored; therefore erosion and sedimentation work exactly under as above the zero level).

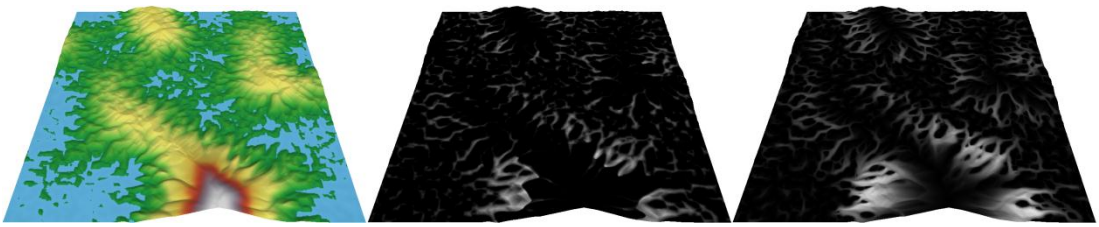


Figure 25: Terrain after 300 iterations of the hydraulic erosion algorithm (left) with its water map (middle) and sediment map (right).

5.9.1. Water increment

During this step a constant amount of water is added to each pixel in the map. This approximates addition of new water due to rain fall over extended period of time.

5.9.2. Flow simulation

This step updates flow values in each pipe based on hydrostatic pressure difference between its two ends.

Outflow flux value change in a pipe in single direction (therefore this calculation is done four times for each pixel) can be expressed by following equation:

$$f_{t+\Delta t} = max(0, f_t + \Delta t \cdot A \cdot \frac{g \cdot \Delta h}{l})$$

f_t is the flux value from previous iteration, Δt is time difference between the two iterations, A is the pipe's cross-sectional area, g is gravitational acceleration, Δh is water level height difference between the ends of the pipe (water level height on each end is terrain height plus water height in the pixel) and l is length of the pipe.

Additionally, each pixel's outflow flux values have to be scaled make sure that total outflow flux from the pixel is less than or equal to water value in the pixel.

After all pipes are updated, water level in the pixels can be updated. Water change in each pixel can be expressed by following equation:

$$w_{t+\Delta t} = w_t + \Delta t \cdot \frac{\sum f_{in} - \sum f_{out}}{l^2}$$

w_t is water level from previous iteration, $\sum f_{in}$ is sum of flows of all pipes in the incoming direction, $\sum f_{out}$ is sum of flows of all pipes in the outgoing direction.

Now the velocity field can be pre-calculated. A single component of velocity vector in a pixel is calculated as sum of total flows in pipes in that direction in that pixel.

5.9.3. Erosion and deposition simulation

Two scenarios can apply when sediment value in a pixel is being updated:

- Water is carrying more sediment that it can hold; some sediment will be deposited in that pixel.
- Water can carry some more sediment; some material in the pixel is dissolved and added to the sediment carried in water.

Which scenario is applied depends or whether current amount of sediment in the pixel is greater than or less than sediment capacity of water in the pixel. Sediment capacity in a pixel is calculated with following equation:

$$c = K_c \cdot sin(max(\alpha_{min}, \alpha)) \cdot |v|$$

K_c is sediment capacity constant, α is terrain tilt angle in the pixel, α_{min} is minimum allowed terrain tilt angle (clamping the angle is necessary to prevent sediment capacity from being zero on flat terrain) and v is velocity vector in that pixel.

If the current sediment amount is less than the capacity, sediment change is calculated with following formula:

$$\Delta s = K_s \cdot (c - s_t)$$

K_c is material dissolution constant, s_t is current sediment amount.

If the current sediment amount is greater than the capacity, sediment change is calculated as follows:

$$\Delta s = K_d \cdot (c - s_t)$$

Here K_d is sediment deposition constant.

Δs is then added to the current pixel's sediment amount and subtracted from current pixel's height. It also has to be added to water amount in current pixel (otherwise

total water level height in the pixel would change, potentially causing oscillations in longer term).

5.9.4. Sediment transportation

In this step, the suspended sediment is carried by the velocity field.

The vectors in the velocity field will most likely end between pixels in the grid, so sediment from a single pixel would have to be distributed among 4 other pixels.

Inverse step is performed instead: inverse vector is used to accumulate the sediment from 4 source pixels (using linear interpolation).

5.9.5. Water evaporation

During this step a small constant percent of water is in each pixel is subtracted. This is necessary to limit excessive accumulation of water in the system.

5.9.6. Adaptive time step

Time step in this algorithm determines strength of effect of individual equations. The lesser the time step, the more iterations are necessary to achieve similar results. Upper bound for the time step is determined by following formula:

$$|v_{max}| \cdot \Delta t \leq l$$

v_{max} is the longest vector in the velocity field, Δt is the time step and l is pipe length. If this rule is broken, powerful oscillations and other artifacts start to emerge.

This formula can be also used to determine optimal time step for the simulation. After every iteration of the algorithm, following formula is used to calculate time step for next iteration:

$$\Delta t = \frac{l}{K_v \cdot |v_{max}|}$$

The velocity vector length is multiplied by a constant K_v, because the velocity vectors are expected to become longer in the next iteration. This modification allows them to become up to K_v-times longer without breaking the rule.

5.10. Thermal erosion

Thermal erosion is process of disintegration of material by thermal shocks and its transportation to lower locations.

Key property of the material is “talus angle” – minimum angle of slope which will be eroded (see Figure 26).

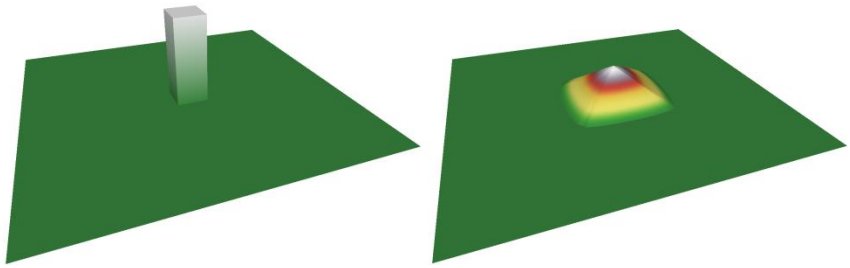


Figure 26: Pillar of material before (left) and after (right) thermal erosion. The material on the right image is fully eroded – all the slope angles are less than or equal to the talus angle.

GeoGen uses combination of algorithms proposed by (12) and (13).

During each iteration of the algorithm, each pixel in the map can be eroded and portion of its height be transported to nearby pixels with lower height as long as height difference between the two pixels is greater than the talus angle. No material can be transported if the height difference between the two pixels is less than the talus angle (or negative).

Total amount of material transported away from the pixel is equal to $H/2$ where H is the greatest height difference with any of the neighbors. This amount is distributed among the valid transportation targets proportionally to height difference of that neighbor.

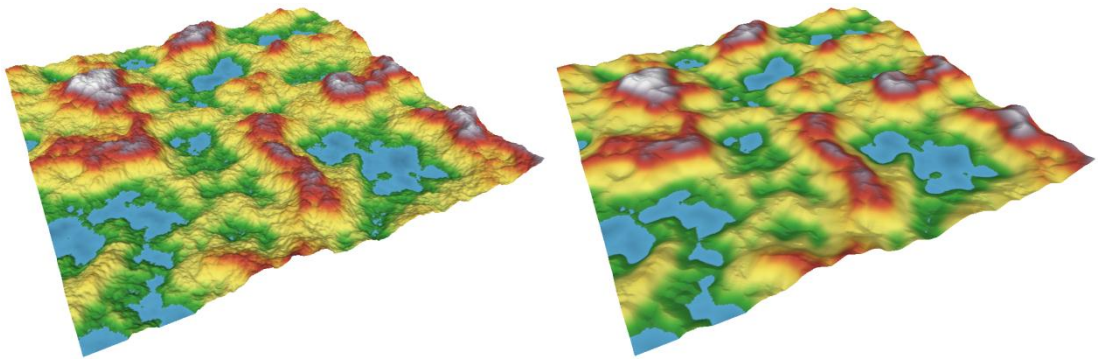


Figure 27: Terrain before (left) and after (right) thermal erosion.

6. Experimental results

This chapter presents some scripts created for GeoGen. All of these scripts can be found among examples on the attached CD.

Figure 28 contains performance statistics about several map scripts. Each data item was acquired as arithmetic mean of three independent executions of the script for each map size.

The measurements were performed on a laptop with Intel Core2Duo P8600 processor running at 2.40 GHz, 4 GB RAM and Windows 7 64-bit.

	256	512	1024	2048	4096	10000
Atoll	0.09 s	0.24 s	0.86 s	3.76 s	15.98 s	86.41 s
Erosion	8.96 s	31.41 s	123.83 s	440.29 s	-	-
Highlands lake	0.07 s	0.22 s	1.06 s	4.43 s	14.10 s	81.18 s
Seaside country	0.11 s	0.37 s	1.50 s	6.10 s	24.30 s	126.91 s
Valley	0.15 s	0.38 s	1.576 s	7.65 s	27.50 s	141.32 s
Voronoi	0.12 s	0.37 s	1.65 s	6.57 s	28.05 s	150.64 s

Figure 28: Example script performance table. On vertical axis are the scripts, on horizontal axis are various map sizes (the maps are square). Dash means the script does not support that map dimension.

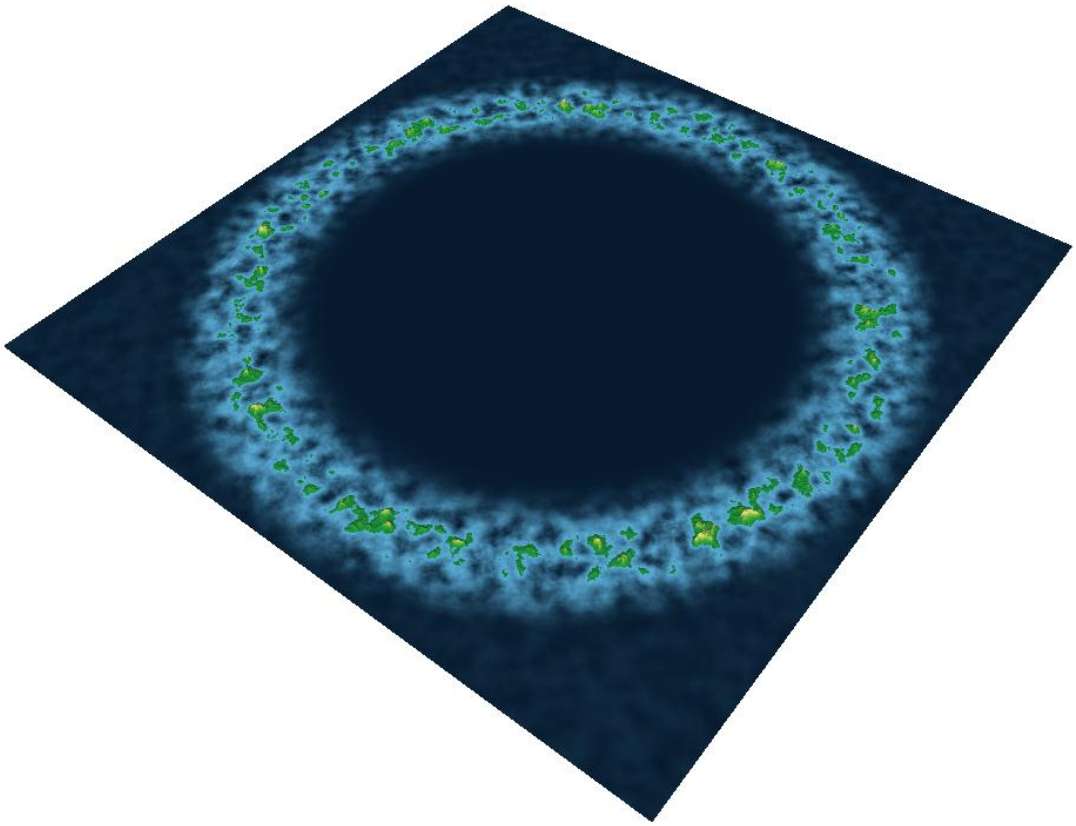


Figure 29: Script "Atoll" with default parameters.

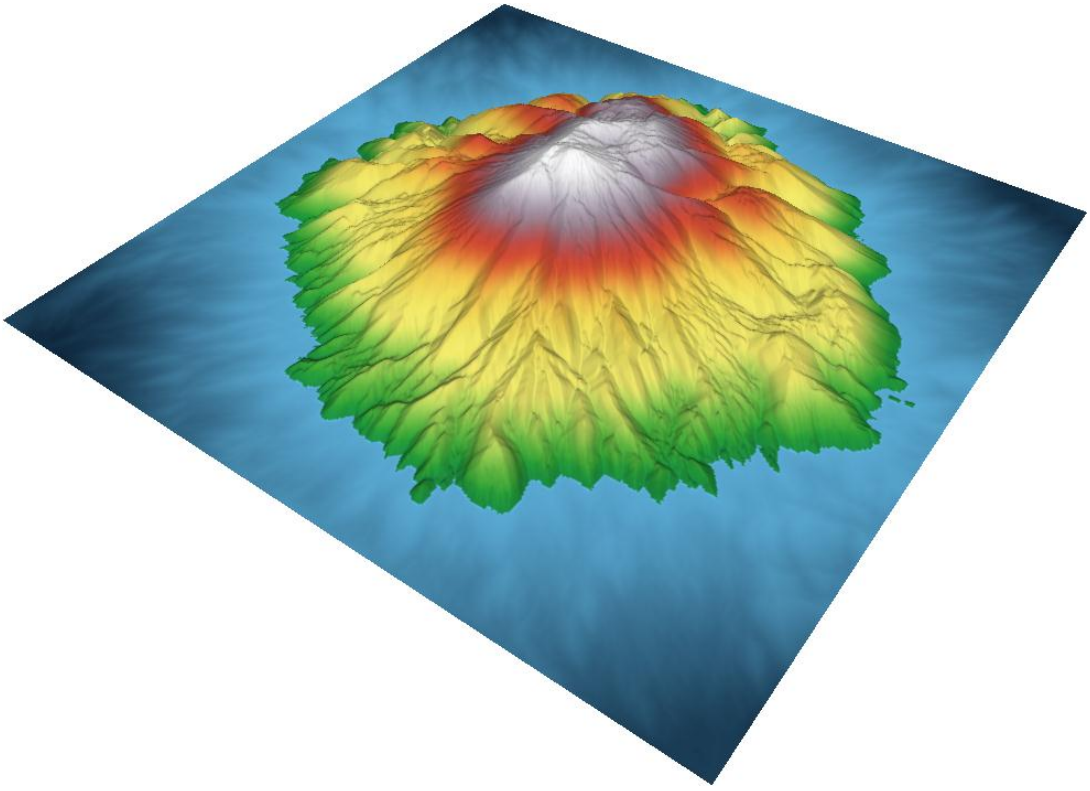


Figure 30: Script "Erosion" with default parameters.

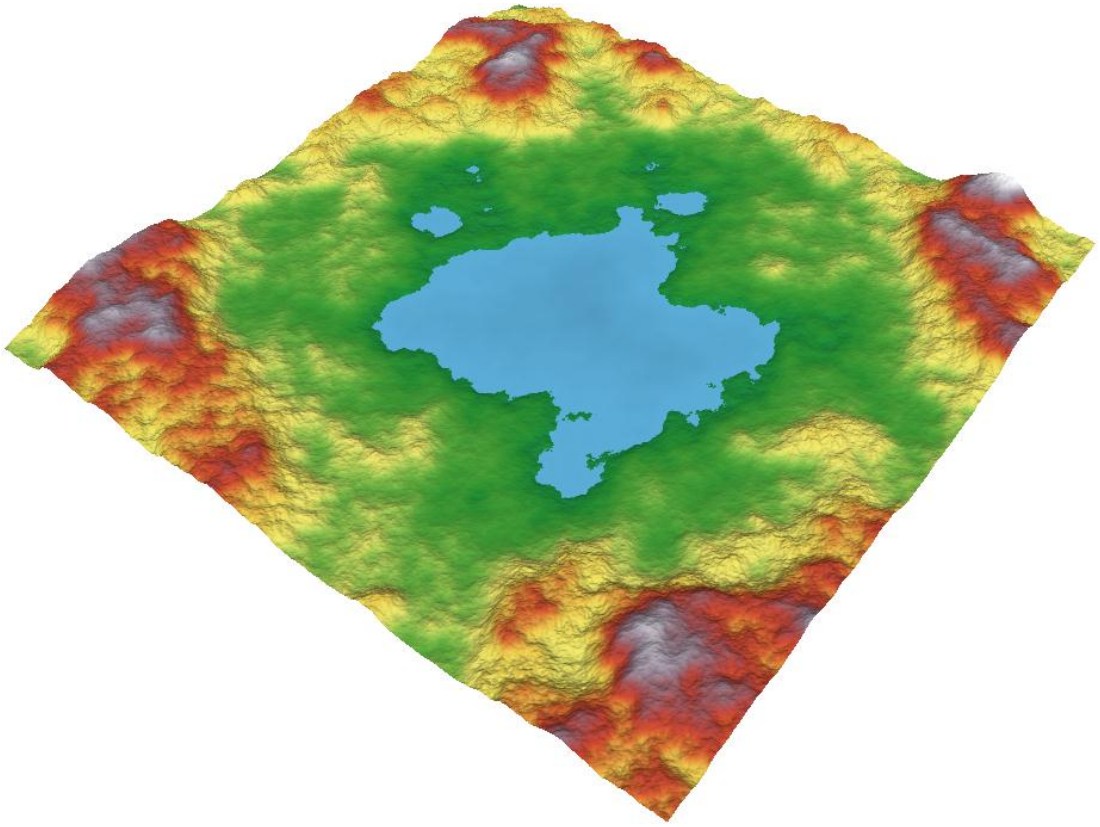


Figure 31: Script "Highlands lake" with default parameters.

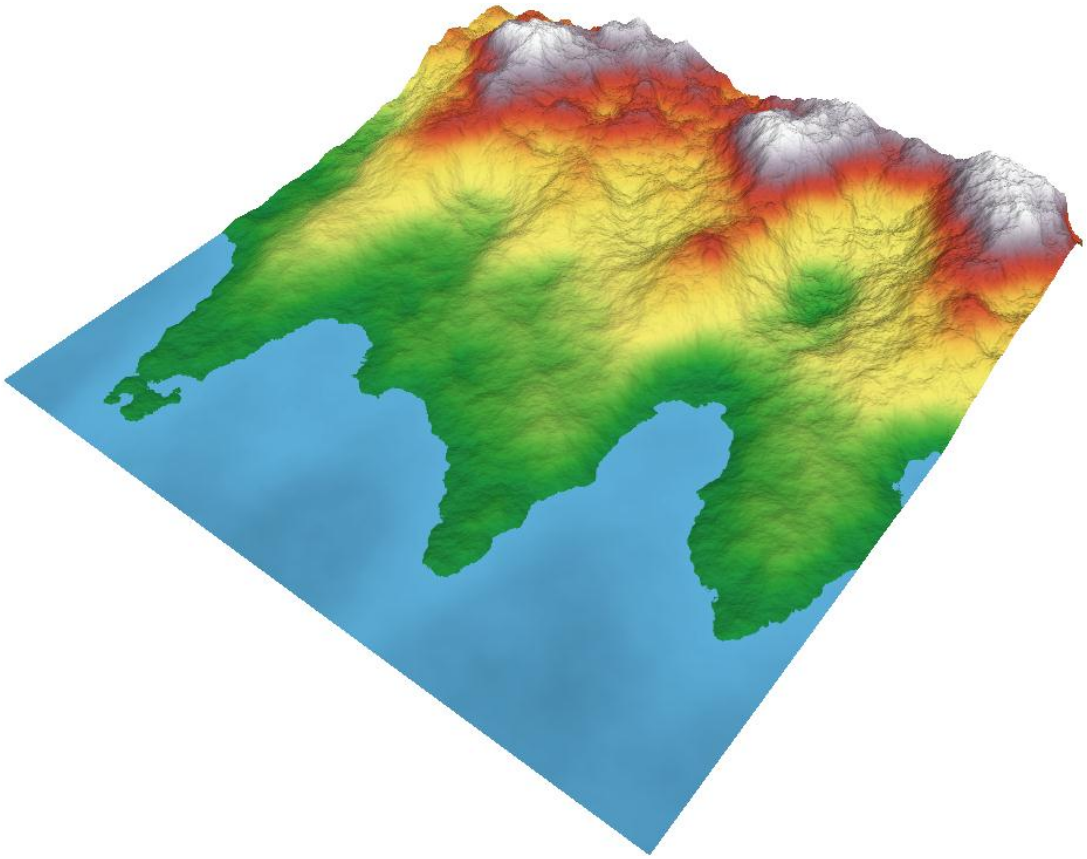


Figure 32: Script "Seaside country" with default parameters.

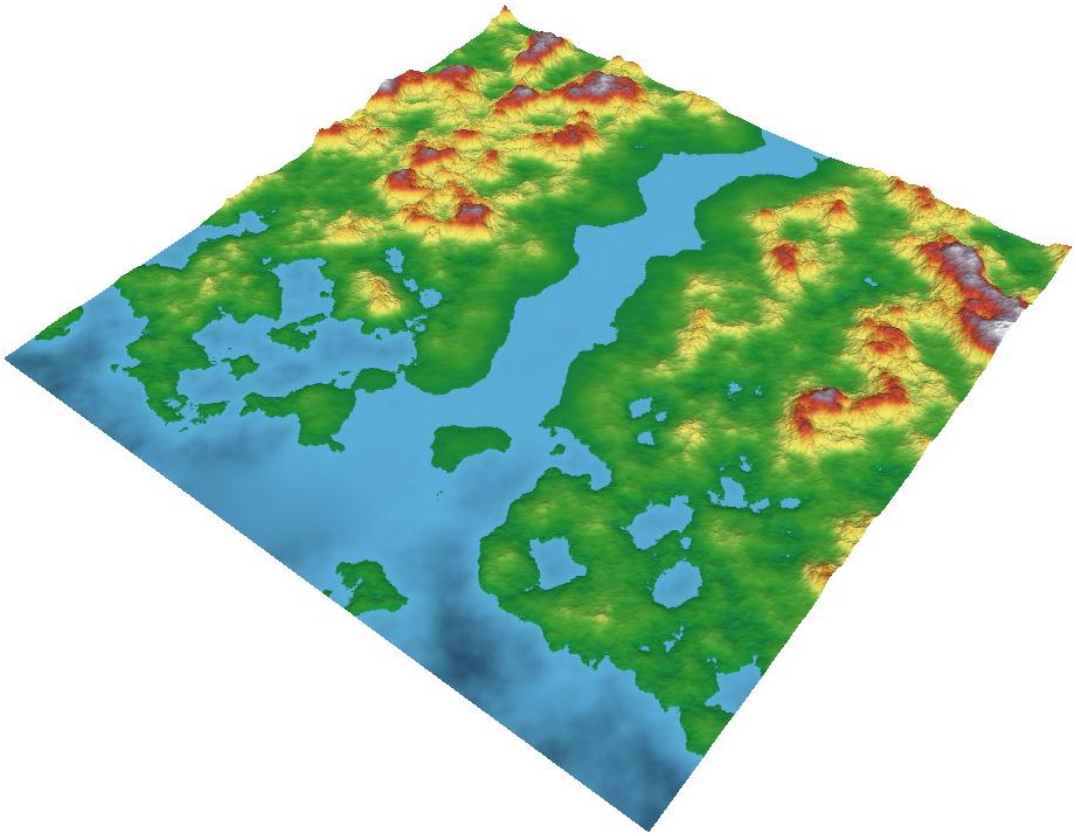


Figure 33: Script "Valley" with default parameters.

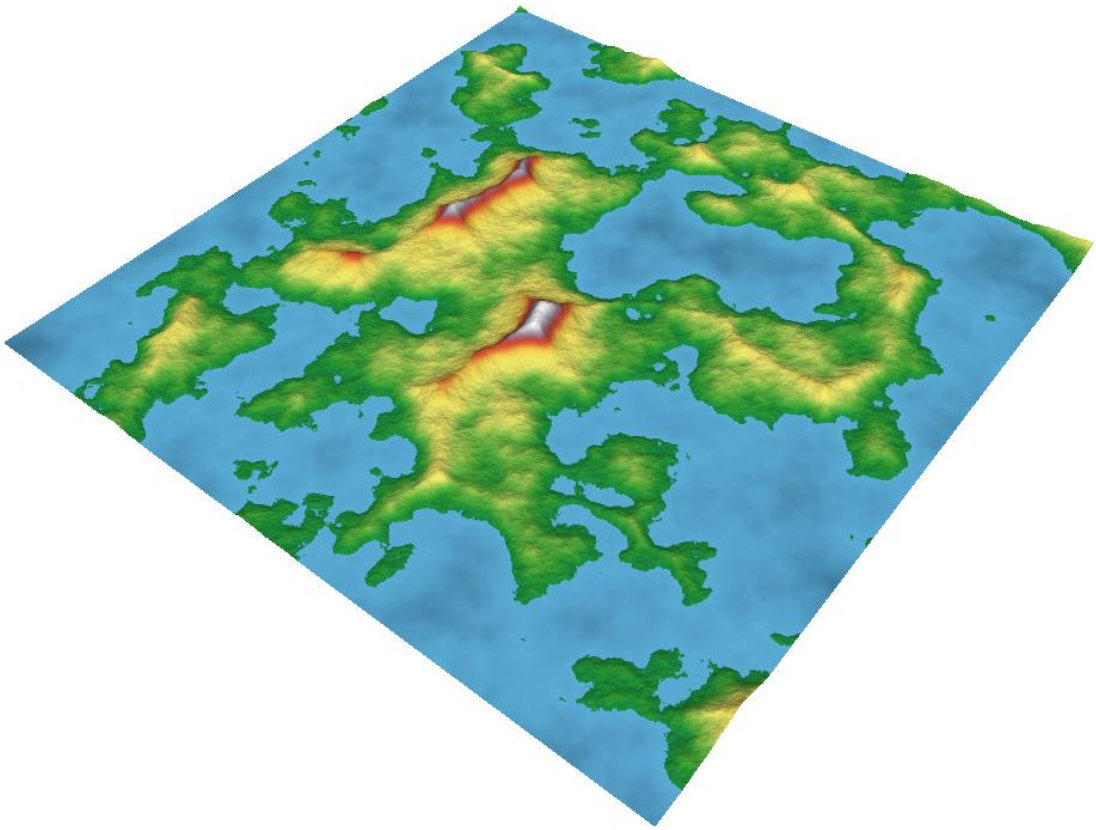


Figure 34: Script "Voronoi" with default parameters.

7. Future work

There are many areas in which this project can be improved and expanded.

One obvious improvement is acceleration of the generation process by GPU. Most algorithms currently used in terrain generation parallelize fairly well, because they are usually performed independently on individual pixels, though this is not always the case (for example flood fill does not parallelize at all). The performance improvement would be most important for the erosion simulator – up to 3000% performance improvement of GPU implementation over CPU implementation was reported by (12). However relying on GPU acceleration could possibly limit support for more exotic platforms and weaker computers, which is also a factor to be taken into consideration.

A feature which would be very helpful for users is ability to quickly generate previews of maps generated by map scripts. A naïve approach is to simply generate the map with significantly decreased requested map size, but it fails because many of the functions behave slightly differently with very small map sizes (aesthetically, of course formally the behavior is the same). Correct solution would be to generate only every n-th pixel in the map, but this would require support in all map generation functions (and it also might not be always 100% reliable).

Similar very useful feature would be ability to generate only certain region in the height map. This would allow for extremely huge maps which don't fit into computer memory (such as 100000×100000 pixels) to be generated and stored in smaller pieces. This function would also require support in all map generation functions as well as detailed information about structure of the script (for example smoothing function would require all previous functions to generate some overlap, so there aren't any broken joints between map pieces). The Squirrel compiler doesn't make this kind of information available, so perhaps a custom compiler and virtual machine would have to be created.

8. Conclusion

Terrain generation is a very interesting field which is attracting attention of many people from various backgrounds – even as this thesis was being written, new articles and papers with exciting new solutions to existing problems were published.

This project demonstrated a novel approach to this problem, which seemed to be completely unexplored – so far there were no ready-to-use height map generator libraries available. The GeoGen library now offers very easy to implement, but powerful terrain generator. Power of this generator was demonstrated by a set of very varied map scripts, which also illustrate how even complex terrain shapes can be created with no more than just a few lines of script code.

A full suite terrain generation and modification tools was implemented, including random noise generators, filters, combiners, masking functions and natural erosion simulators.

The generator library is accompanied by an intuitive script development environment, which integrates the generator with powerful code editor, interactive 3D height map viewer and other tools.

However some issues were not satisfactorily solved yet – the most pressing one being excessive amount of time required by the hydraulic erosion simulator. This prevents hydraulic erosion from becoming a standard post-processing filter used in most map scripts.

This thesis has fulfilled its objective – a powerful height map generator library was created. With some future improvements, integration of this library into visualization applications and video games can become a very attractive option for software developers.

A. User Manual

This chapter provides user documentation for all aspects of GeoGen.

A.1. Script writing

This chapter is an introduction to writing map scripts for GeoGen.

A.1.1. Scripting language

GeoGen scripts are written using Squirrel 2.2 programming language. Documentation of this language and its syntax can be found on its website (14) or in PDF format on the attached CD (see appendix C).

A.1.2. Script layout

Each map script is composed of two main parts – header and body.

Header is represented by a function named `GetInfo` which accepts one parameter – information type (such as script name, script author or parameter list). This function will be repeatedly called by the API requesting various information types. Typical content of this function is a switch statement responding to individual request strings. This function returns either string or integer. It must not contain any map-generation logic.

```
function GetInfo(info_type){
    switch(info_type){
        case "name":
            return "Short name";
        case "description":
            return "Long description of the map.";
        case "args":
            GGen_AddIntArg("width","Width","Width of the map.", 1024, 128, 4096 1);
            GGen_AddIntArg("height","Height","Its height.", 1024, 128, 4096, 1);
            return 0;
    }
}
```

Figure 35: Example script header.

Body is represented by parameter-less function named `Generate`. This function will be called every time a new map is requested. This function must return a single `GGen_Data_2D` object[3].

```
function Generate(){
    // Load parameter values
    local width = GGen_GetArgValue("width");
    local height = GGen_GetArgValue("height");

    // Create a new 2D data array with given width and height filled with value 0
    return GGen_Data_2D(width, height, 0);
}
```

Figure 36: Trivial script body.

A.1.3. Standard library

GeoGen offers extensive array of height map creation and manipulation functions. Full reference of GeoGen scripting library can be found in HTML format on the attached CD (see appendix C).

The script can also use all functions found in the Squirrel Standard Math Library. Full reference of this library can be found on the internet (15) or in PDF format on the attached CD (see appendix C).

A.1.4. Standards for indexing, axes and measures

All indexes in GeoGen are zero-based. Axes use the standard usually used in in computer graphics – X axis is horizontal, Y axis is vertical, origin is in top left corner of the height map (therefore top right corner has coordinate $[width - 1,0]$, bottom left corner has coordinate $[height - 1,0]$ and bottom right corner has coordinate $[width - 1, height - 1]$).

Interpretation of directions as cardinal directions is also sometimes used (top then means north, bottom means south, left means west and right means east).

Minimum terrain height in a pixel is -32767 (also available as constant `GGEN_MIN_HEIGHT`), maximum height is 32767 (also available as constant

[3] Secondary maps are returned using method `ReturnAs` of a `GGen_Data_2D` object.

GGEN_MAX_HEIGHT). Height 0 is widely used as a base level (a default value) and is also interpreted as water level (though this interpretation is not obligatory[4]).

Height -32768 is reserved as invalid height (similar to NaN in float-point arithmetic; available as constant GGEN_INVALID_HEIGHT).

Notation of angles is also not surprising. Angle 0° equals to 360° and means east, 90° means north, 180° means west and 270° means south.

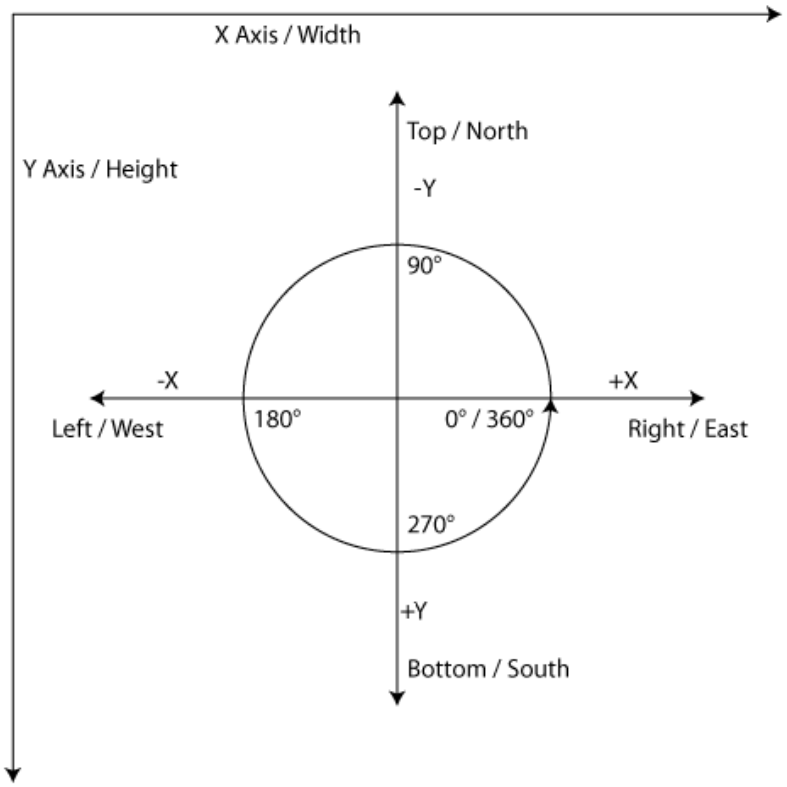


Figure 37: Standards for axes and angles.

However a different angle notation is sometimes used when angle information needs to be stored in height data array (for example as result of function NormalMap of GGen_Data_2D object). Then the angles are linearly mapped onto the supported height range - -32767 and 32767 mean 0°/360° and 0 means 180°.

[4] Height maps by themselves don't carry any information about absolute measures. All dimensions expressed in a height map are always relative, if no further information is provided.

A.1.5. Example scripts

GeoGen installation comes with many commented example scripts which demonstrate various aspects of map script-writing. These scripts can be found in user directory after installing GeoGen using the installation wizard or on the attached CD.

A.2. Overlays

Overlays define a transformation of a height map to a fully colored image suitable for human viewing. Overlay assigns a color to each possible height in the height map.

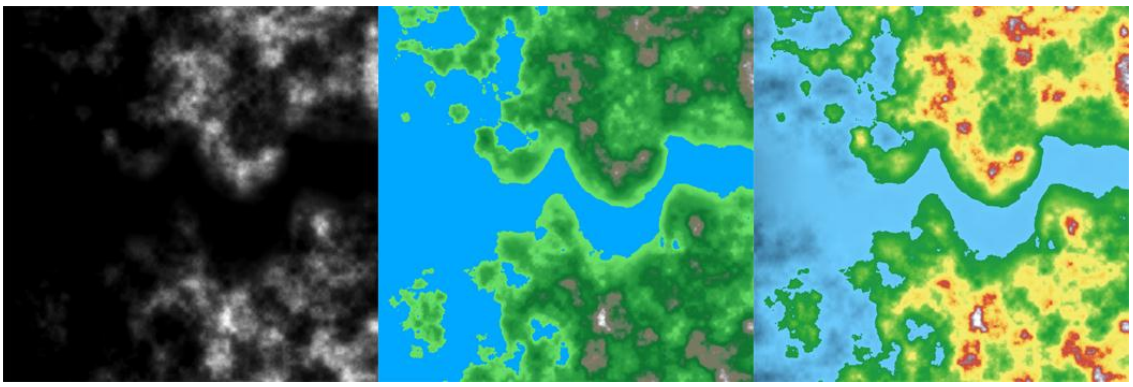


Figure 38: Overlays - height map (left), basic overlay "Alpine" (middle), extended overlay „TopoBathy“ (right).

Overlay is a “Windows Bitmap” (BMP) image. There are two types of overlays:

- **Basic overlay** – image with dimensions 256×1. This type of overlay neglects negative heights (all negative heights are treated as “0”).
- **Extended overlay** – image with dimensions 511×1. This type of overlay supports negative heights.

Figure 39: Basic overlay "Alpine".

Figure 40: Extended overlay "TopoBathy".

Overlays can be used with both the console interface and with GeoGen Studio.

A.3. Console interface

The basic command line syntax for the GeoGen console interface is this:

```
geogen -i script -o output [options] [script_arguments]
```

Neither argument is obligatory – when executed with no options, GeoGen will ask for all necessary arguments to be filled in interactively.

`script` indicates relative path to a Squirrel script file. `output` will be the place where the output bitmap is saved. If this file already exists, it will be overwritten.

`script_arguments` is a space-separated list of arguments in order in which the script requests them. If a script requests an argument value and you don't set it, default value proposed by the script is used. You can use "r" instead of any value to pick it randomly from the range of valid values.

`options` can be used to adjust more advanced aspects of the generator. Complete list of options:

`-i FILE, --input FILE`	Input Squirrel script to be executed.
`-o FILE, --output FILE`	Output file, the extension determines file type of the output (*.bmp for Windows Bitmap, *.pgm for Portable Grey map and *.shd for GeoGen Short Height Data are allowed). Set to "../temp/out.bmp" by default.
`-d DIRECTORY, --output-directory DIRECTORY`	Directory where secondary maps will be saved. Set to "../temp/" by default.
`-v FILE, --overlay FILE`	Overlay file to be mapped on the output. This file must be a Windows Bitmap file one pixel high and either 256 or 511 pixels wide.
`-s SEED, --seed SEED`	Pseudo-random generator seed. Maps generated with same seed, map script, arguments and generator version are always the same.

`-a, --all-random`	All unset script arguments are generated randomly.
`-z, --ignore-zero`	Height data range will be rescaled to fit the output file format including negative values. Zero level will probably not be preserved.
`-n, --no-rescaling`	The height data will not be rescaled at all. Might cause color overflows if the format's value range is lower than <-32787, 32787>.
`-h, --split-range`	Splits the value range of a file format, which doesn't support negative values, so lower half of the range covers negative values and upper half covers positive values. Value "(max + 1) / 2" will be treated as zero.
`-?, --help`	Displays detailed usage help.
`-x, --syntax-check`	Will print OKAY if script is compilable or describe the error found.
`-p, --param-list`	Lists the script's parameters in machine-readable format.
`-e, --simple`	Mode which allows all necessary data to be entered interactively. This mode is automatically activated if no parameters were entered.
`-m, --manual`	Script arguments will be entered interactively.
`-D, --disable-secondary-maps`	All secondary maps will be immediately discarded, ReturnAs calls will be effectively skipped.
`-V, --overlay-as-copy`	Color files with overlays will be saved as copies.

`-g SIZE, --grid SIZE`	A grid with spacing SIZE will be painted onto output file along with chosen color overlay.

A.4. GeoGen Studio

This chapter is a detailed GeoGen Studio user manual.

A.4.1. Prerequisites

GeoGen Studio requires .Net Framework 4.0. Installer can be found on the attached CD (see appendix C).

GeoGen Studio is not compatible with Mono.

Hardware requirements are:

- 1 GB RAM (2 GB recommended).
- For 3D support is required OpenGL 1.4-compatible graphics accelerator with at least 256 MB of dedicated video RAM.
- Dual-core processor is strongly recommended.

A.4.2. Layout of main window

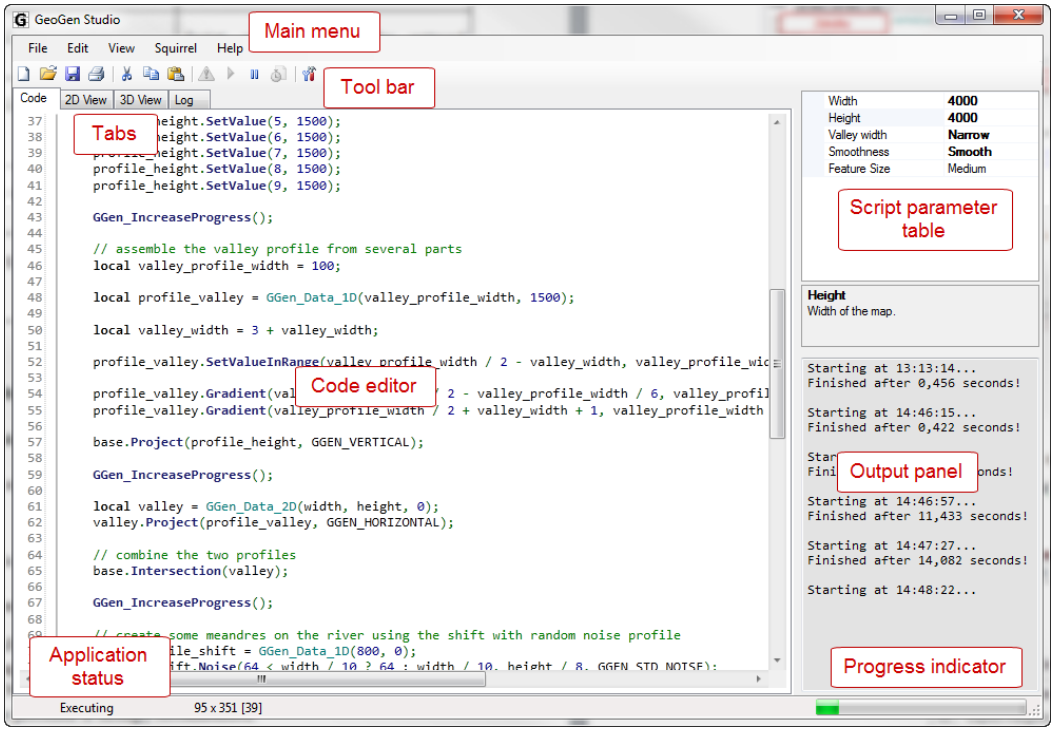


Figure 41: Layout of main window

Main window of GeoGen Studio contains several main feature areas:

- **Main area with tabs –** the tabs contain main tools used while developing GeoGen scripts – code editor, 2D viewer and 3D viewer.
- **Main menu** – menus provide unified access to most features of the application.
- **Tool bar** – contains shortcuts to the most frequently used features. "2D View" and "3D View" tabs have their own tool bars with context-specific features.
- **Script parameter table** – configures currently edited script. These values will be used the next time the script is executed.
- **Output panel –** displays various runtime information about the application's state such as script execution information, error messages etc.
- **Progress indicator** – displays execution progress of current script (if the script supports progress reporting).
- **Application status** – displays what state the application is currently in.

A.4.3. File manipulation

GeoGen Studio supports all file operations common for text editors. All these functions are accessible from the „File“ menu, the most common operations are present on the tool bar as well.

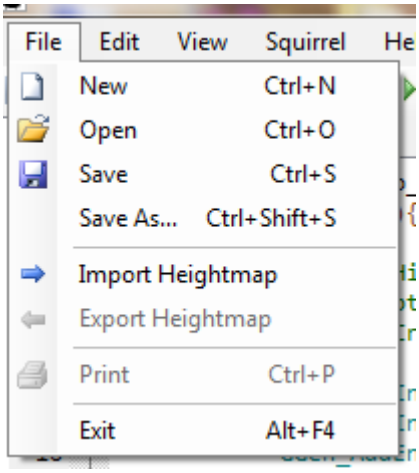


Figure 42: "File" menu

Description of individual operations:

- **"New"** – creates a new script based on a script template. This template contains all the minimum code necessary for an executable GeoGen script.
- **"Open"** – loads a script file from the hard drive.
- **"Save"** – saves current file on the hard drive. If the file was already saved before, it will be overwritten; otherwise a file selection dialog will open.
- **"Save As... "** – saves current file on the hard drive. File selection dialog will open.
- **"Import heightmap"** – loads a height map for viewing in 2D or 3D.
- **"Export heightmap"** – opens window "Export Heightmap" (see next section).
- **"Exit"** – terminates the application. If there are any unsaved changes in the current script, the user will be asked to save it.

Height maps and scripts can be also opened by dragging them onto the main area of the window.

A.4.4. Window "Export Heightmap"

Window "Export Heightmap" saves height map currently opened in 2D view on the hard drive.

Fields "Height" and "Width" define size of the exported height map. Only dimensions smaller or equal to dimensions of the original height map are allowed. Button "Reset" will reset the values to the original dimensions.

Switch "Negative heights" defines how negative values will be treated if the file format doesn't support them (only SHD supports negative values).

Option "Replace negative heights with 0" will discard all negative values and replace them with 0. Option "Rescale all heights into positive range" will rescale all values (including negative heights) into positive value range. Using this option will cause loss of information about level 0.

Button "Export" will start the export; button "Cancel" will close the window without exporting the map.

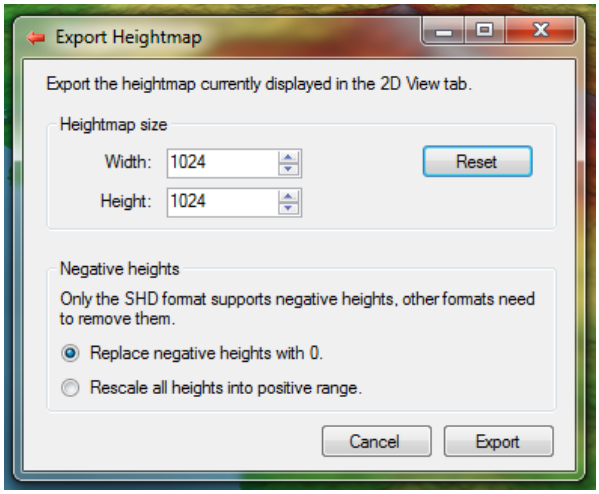


Figure 43: "Export Heightmap" window.

.

A.4.5. Code editor

Code editor is the main tool used when creating and debugging GeoGen scripts. It is on the tab "Code" in main application window.

Figure 44: Code editor.

Code editor highlights syntactical elements in the code:

- **Black (bold)** – Squirrel keywords.
- **Dark red** – string constants.
- **Blue** – numeric constants, function calls.
- **Teal** – GeoGen objects.
- **Green** – special characters (brackets, commas, semicolons, operators etc.), comments.

On the right side between line numbers bar and the editor itself is the code folding bar: clicking "-" hides logical blocks of the code (function blocks, conditions, loops etc.). Only its first line will be visible from a hidden block and the "-" symbol will change to "+". Clicking on this "+" will then unhide the block.

Code is being checked for validity as it is being edited. Whenever a change is done that leads to invalid script code, the output panel will change its background to red. When the code is valid once again, the color will revert to its usual gray. Detailed information about the error can be found by clicking "Syntax Error Details" in menu "Squirrel" (see chapter A.4.8.).

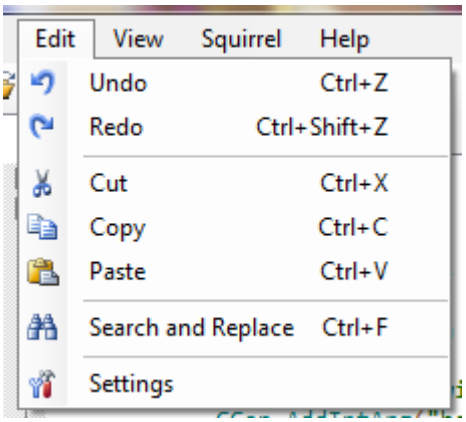


Figure 45: "Edit" menu.

More code editor related features are in the "Edit" menu:

- **"Undo"** – reverts last change done in code editor.
- **"Redo"** – reverts last "Undo" operation.
- **"Copy"** – copies text selected in code editor into system clipboard.
- **"Cut"** - copies text selected in code editor into system clipboard and erases it from the editor.
- **"Paste"** – inserts text from system clipboard into the code editor.
- **"Search and Replace"** – opens „Find and Replace“ window (see next section).
- **"Settings"** – opens "Settings" window with application settings (see chapter A.3.14.)

A.4.6. Code completion

When typing a function or constant name in the code editor, pressing Ctrl + Space will open the Code completion window. The window contains a list of all GeoGen's functions and constants (the list will be filtered as you type). Arrow keys can be used to navigate the list, and Enter or Space keys to accept the suggestion. Escape key will close the Code completion window.

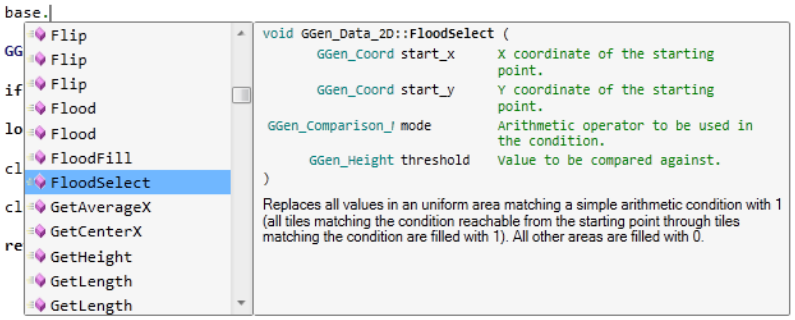


Figure 46: Code completion window.

A.4.7. Window "Find and Replace"

Window "Find and Replace" contains text searching and replacing tools. All these tools work with the code currently present in the code editor.

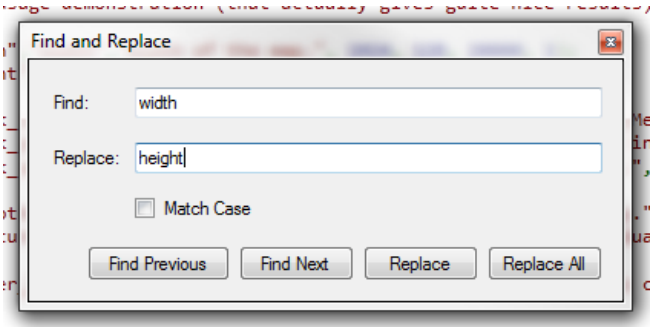


Figure 47: "Find and Replace" window.

Controls in the window:

- **"Find"** – text string to be searched for.
- **"Replace"** – text string to replace the searched string.
- **"Match Case"** – toggles case sensitive search.
- **"Find Previous"** – moves to previous occurrence of the string in the „Find“ field.
- **"Find Next"** - moves to next occurrence of the string in the „Find“ field.
- **"Replace"** – replaces current occurrence of the string in the „Find“ field with the string in the „Replace“ field and moves to next occurrence.
- **"Replace All"** – replaces all occurrences of the string in the „Find“ field with the string in the „Replace“ field.

A.4.8. Executing scripts

GeoGen Studio can execute currently edited file without having to save it to hard drive. All script execution-related features are in the "Squirrel" menu:

- **"Syntax Error Details"** – prints information about last error found by real-time script validation into output panel.
- **"Execute"** – executes currently edited script. If the execution succeeds its output will be loaded into 2D and 3D views. Error details will be printed into ouput panel otherwise.
- **"Terminate"** – cancels currently executed script.
- **"Benchmark"** – opens „Benchmark“ window.

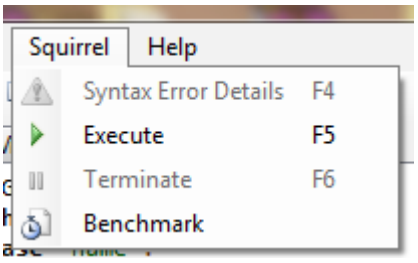


Figure 48: "Squirrel" menu.

A.4.9. Benchmarking

Benchmark is a tool which assigns a score to a script based on its performance.

Score is calculated as ratio of execution speed of the tested script and a reference script to make this score as independent on user's hardware as possible. For best accuracy, both scripts are executed alternatively with various map sizes (specifically 512×512, 1024×1024, 2048×2048 and 4096×4096). Resulting score is for example „2.71×" for a script which is 2.71-times slower than the reference script.

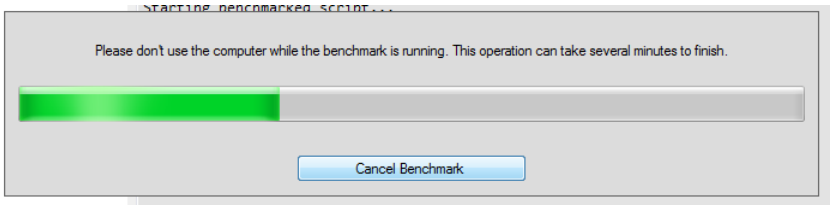


Figure 49: "Benchmark" window.

Benchmark is started by clicking "Start Benchmark" in "Benchmark" window. Once started, the benchmark can take many minutes to finish. It can be canceled at any time by clicking "Cancel Benchmark".

A.4.10. 2D view

2D view displays the height maps in a fashion similar to common image browsers. It is on the tab "2D View" in main window of the application. All the controls on this tab will be inactive if no maps are loaded.

The displayed map can be panned by dragging mouse with left mouse button down. Mouse wheel zooms in and out.

Figure 50: 2D view.

Tools found on the 2D view tool bar:

- **"Save currently displayed image"** – saves currently displayed image on hard drive (including colors provided by currently selected overlay).
- **"Export current heightmap"** – opens window "Export Heightmap" (see chapter A.3.4.). Map saved this way won't be colorized by an overlay.
- **"Available maps"** - selects which one of the returned (or imported) maps will be displayed.
- **"Overlay views"** – selects overlay to colorize the map.
- **"Toggle overlay display"** – toggles between the default grayscale height map and its colorized version.
- **"Reset view"** – resets the view to its original position and zoom level.

A.4.11. 3D view

3D view displays the terrain as a 3D model. It is on the tab "3D View" in main window of the application. All the controls on this tab will be inactive if no maps are loaded.

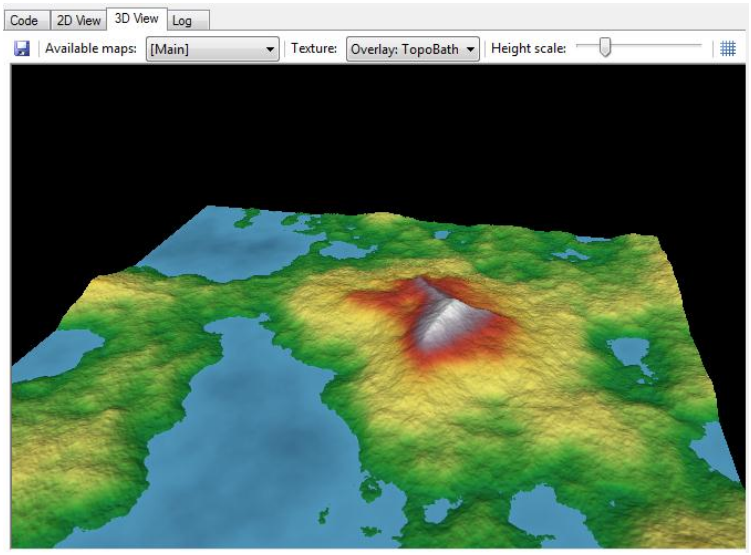


Figure 51: 3D view.

Mouse is used to navigate through the 3D space. Dragging the mouse with left mouse button down rotates the view, dragging the mouse with right mouse button down pans the view. Mouse wheel is used to zoom in and out.

Tools found on 3D view tool bar:

- **"Save screenshot"** – saves view currently displayed in the 3D viewport to an image file.
- **"Available maps"** – selects which one of the returned (or imported) maps will be used as a base for the 3D model.
- **"Texture"** – selects texture for the 3D model. It contains all overlays (marked as „Overlay“), all maps generated by the last executed script (marked as "Map") and an option to import any image file from hard drive (marked as "Import External").
- **"Height Scale"** – vertical scale of the terrain model.
- **"Toggle wireframe"** – toggles between regular shaded mode and wireframe mode.

Performance and quality of 3D view can be configured in the window "Settings".

A.4.12. Application settings

Some customization options can be found in menu "View":

- **"Sidebar"** – toggles visibility of the side bar. Submenu allows to select where will the side bar be displayed and which panels will it contain.
- **"Toolbar"** – toggles visibility of the tool bar.
- **"Statusbar"** – toggles visibility of the status bar.

- **"Word Wrap"** – toggles wrapping of long lines in the code editor.
- **"Increase Font Size"** – increases font size of the code editor.
- **"Decrease Font Size"** - decreases font size of the code editor.
- **"Reset Font Size"** – resets font size of the code editor to its default value.

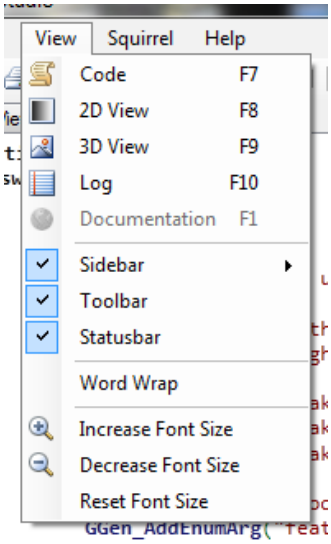


Figure 52: "View" menu.

More advanced settings can be found in "Settings" window accessible from "Edit" menu:

- **"BackgroundColor3d"** – background color of the 3D viewport.
- **"EnableBlackCompensation"** – do not allow completely black areas on textures, so the shading highlighting shape of the model is always visible.
- **"EnableTerrainUnderZero"** – display 3D terrain under level 0 (usually water level). This setting won't affect rendering of extended overlays.
- **"ModelDetailLevel"** – quality of the terrain 3D model (quantity of rendered triangles). This setting determines how much time will calculation of the model from a height map take. Higher settings can cause issues on machines without graphics accelerator with large amount of dedicated video memory.
- **"TextureDetailLevel"** – quality of 3D model texture. Higher settings allow to create illusion of higher display quality even with low "ModelDetailLevel" setting.
- **"TextureScalingAlgorithm"** – algorithm used to scale textures in 3D view. „NearestNeigbor" produces blocky, but sharp textures; „LinearInterpolation" produces smooth, but blurry textures.
- **"AutomaticallyAcceptSuggestions"** – automatically accept code suggestions if there is only one applicable suggestion.
- **"AutomaticallyInsertBrackets"** – insert brackets when appropriate after a code suggestion was accepted.
- **"OpenSuggestionsOnDot"** – open code suggestion list after typing „.".
- **"OverlayDirectory"** – directory from which are overlays loaded.

- **"TemplateFile"** – file opened when a new script is created.
- **"ActionAfterExec"** – determines behavior of the application after map generation is finished: "DoNothing" – does nothing, "GoTo2DOutput" – goes to 2D view a "GoTo3DOutput" – goes to 3D view.
- **"MapDetailLevel"** – application-wide maximum map size setting. All generated or imported maps are immediately reduced to this size. Decrease this setting if the Studio is frequently running out of memory.
- **"RandomSeed"** – pseudo-random number generator seed. Same seed, same script, same parameters and same generator version always guarantees 100% identical output. Value „0“ means a new seed will be generated every time a script is executed (every map will be different from the previous one as long as it contains any randomized functions).
- **"OpenLastFileOnStartup"** – opens last opened file when the program is started. The template will be loaded if this setting is set to false.

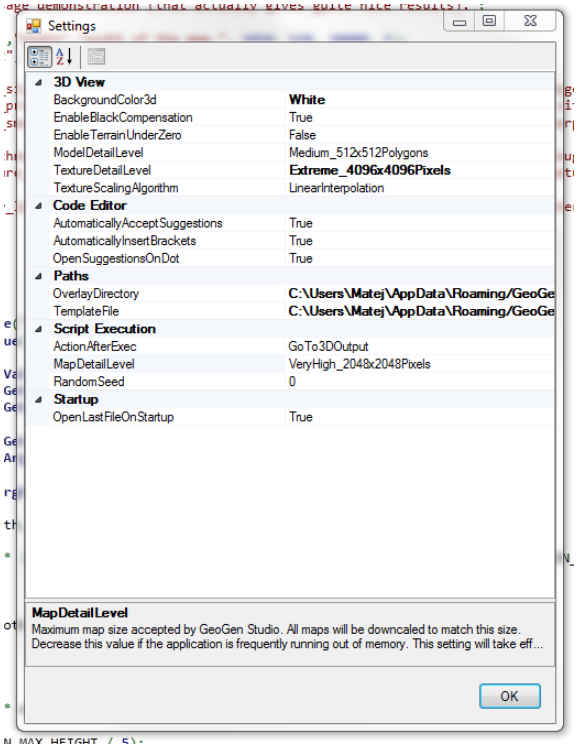


Figure 53: "Settings“ window.

B. Developer manual

This chapter details structure of GeoGen's source code.

Project files for Visual Studio 2010 with all the components are included. Compiling for Windows is only matter loading the VS 2010 solution and building the project.

Makefile is also included, however it only compiles the core library with console interface (Studio is not compatible with any non-Windows platforms).

B.1. Core library

GeoGen core library is written in C++ and can be compiled with Visual Studio 2008 or newer or with GCC 4 or newer. The library source code can be directly included into the hosting application or it can be compiled as either statically linked or dynamically linked library.

The console interface is a simple example implementation of GeoGen in an application.

B.1.1. Third party components

GeoGen uses two third party libraries:

- **Squirrel** (16) – script compiler and virtual machine used to execute the scripts.
- **SqPlus** (17) – binding library used to expose GeoGen scripting functions to scripts (a very complex task to do without binding library).

Additionally, the console interface uses EasyBMP (18) library to read and write BMP files.

All of these libraries are open-source and released under permissive licenses.

B.1.2. Library architecture

GeoGen source code is composed of two distinctive parts:

- **Generator** – this initializes the Squirrel engine, feeds it with data (scripts and parameters) and serves outputs back to the client code. This logic is separated into two classes: "GGen" which contains common logic every language extension would need and "GGen_Squirrel" which contains the Squirrel implementation specific logic. The idea is that in future adapters for more scripting languages like Lua or Python could be relatively easily added.
- **Exposed classes** – these are the classes that are exposed to scripts. These classes represent logical objects that contain the actual height map generation logic: height data arrays, points, paths etc.

B.2. GeoGen Studio

GeoGen Studio is a Windows application written in C# 4.0. User interface is created using the Windows Forms framework (except the code editor, which is created in WPF).

B.2.1. Third party components

Several free third party components are used in this application:

- **AvalonEdit** (19) – this component is the code editor present in the Studio. It comes with support for syntax highlighting, line wrapping etc. It uses WPF, therefore it must be wrapped in ElementHost to interact with rest of the application.
- **OpenTK** (20) – this library is a .NET wrapper for OpenGL. Studio uses it for its 3D viewport.
- **PropertyGridEx** (21) – this control improves the PropertyGrid control provided by Microsoft with Windows Forms. The regular PropertyGrid is unable to display a custom set of properties, only properties of a certain class – which is what PropertyGridEx improves. This control is used as the script parameter table in the Studio.

All of these components are open-source and released under permissive licenses.

B.2.2. Interaction with the native core library

Interaction with the core library is handled by a bridge class library written in C++/CLI. This library wraps every part of the public API exposed by the GeoGen core library and exposes them as CLR-compatible classes.

However the translation is not 1:1 – the exposed .NET API is adapted to fit into the .NET ecosystem and follows as closely as possible Microsoft's standards for class libraries (22). For example methods, which indicate failure by returning NULL in the native API, now throw exceptions with detailed information about the error.

B.2.3. Application architecture

The application logic is divided into four main components:

- **Main window** – handles application initialization and user interface logic.
- **Process manager** – this class is responsible for interaction with the generator core (starting and terminating scripts, script configuration, real-time script validation and benchmarking).
- **Output manager** – responsible for processing height maps after they leave the generator (conversion into Windows Forms compatible bitmaps, resizing, overlays, exporting).
- **Viewport manager** – this class contains all functionality related to the OpenTK viewport (viewport initialization, model construction, scene loop, texture manipulation and camera movement).

B.2.4. Configuration persistence

All persistent settings are stored in a single configuration object. When the application closes, this object is serialized into a file using Microsoft's XML serializer. When the application is started once again, the XML file is deserialized back into the object. If the file is either corrupted or not present, an empty instance of the configuration class is loaded (which contains defaults for all settings).

The configuration file is either located in GeoGen's directory (this one always has higher priority when looking for the file) or in Windows user's directory. This is necessary, because Windows Vista and newer restrict writing access to the Program

Files directory if the User Account Control feature is enabled. If the configuration file was saved into the program directory, the application would require elevated privileges (possibly opening the “Do you really want to do this?” dialog on every launch).

C. Contents of the CD

The attached CD contains following files and directories:

- **source** – source code of GeoGen along with all necessary files to build them with Visual Studio 2010 or GCC 4.
- **documentation** – documentation for writing scripts. Contains following files and subdirectories:
    - **GeoGen** – complete reference of all GeoGen-specific features available to map scripts (includes all classes, methods and constants).
    - **Squirrel language.pdf** – documentation of the Squirrel language.
    - **SqStdLib reference.pdf** – reference of Squirrel standard library (however only the math library is available to GeoGen script).
- **examples** – example map scripts (those are also included in the installation and archive).
- **DotNetFX40** - .NET Framework 4.0 installer (GeoGen setup will install this automatically if necessary).
- **WindowsInstaller3_1** – Windows Installer 3.1 installer (GeoGen setup will install this automatically if necessary).
- **GeoGen.zip** – ZIP archive containing GeoGen for Windows with all necessary files (no further installation other than the .NET Framework is necessary).
- **GeoGen Installer.exe** and **GeoGen Installer.msi** – Windows installer which installs the application onto the system (including dependencies and file type associations).
- **thesis.pdf** – electronic version of this thesis.

Bibliography

1. *World Machine 2.* [Online] http://www.world-machine.com/index.php.

2. *GeoControl 2.* [Online] http://www.geocontrol2.com/e_index.htm.

3. Bryce 7. *Daz 3D.* [Online] http://www.daz3d.com/i.x/software/bryce/-/.

4. *Layered Data Representation for Visual Simulation of Terrain Erosion.* **Beneš, B.** 2001. 17th Spring Conference on Computer Graphics.

5. **Zhou, Howard, et al., et al.** *Terrain Synthesis from Digital Elevation Models.* 2007.

6. *Algorithms for Generating Fractal Landscapes.* **Stanger, Keith.** 2006.

7. **Breeuwsma, Paul.** Bicubic Interpolation. *PaulInternet.* [Online] http://www.paulinternet.nl/?page=bicubic.

8. *A Voronoi-based Distributed Genetic Algorithm.* **Whigham, Peter A. and Dick, Grant.** 2003. SIRC 2003: the 15th Annua Colloquium of the Spatial Information Research Centre.

9. **Olsen, Jacob.** *Realtime Procedural Terrain Generation: Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games.* Department of Mathemathics and Computer Science, University of Soutern Denmark. 2004.

10. *Fast Image Convolutions.* **Jarosz, Wojciech.** 2001. ACM SIGGRAPH.

11. **Mei, Xing, Decaudin, Philippe and Hu, Bao-Gang.** *Fast Hydraulic Erosion Simulation and Visualisation on GPU.* 2007.

12. *Fast Hydraulic and Thermal Erosion on the GPU.* **Jákó, Balász.** 2011. The 15th Central European Seminar on Computer Graphics.

13. **Olsen, Jacob.** *Realtime Procedural Terrain Generation: Realtime Synthesis of Eroded Fractal Terrain for Use in Computer Games.* 2004.

14. Squirrel 2.2 Reference Manual. *Squirrel Programming Language.* [Online] http://www.squirrel-lang.org/doc/squirrel2.html.

15. Squirrel Standard Library 2.2. *Squirrel Programming Language.* [Online] http://squirrel-lang.org/doc/sqstdlib2.html.

16. *Squirrel - The Programming Language.* [Online] http://www.squirrel-lang.org/.

17. Sq Plus. *Squirrel Wiki.* [Online] http://wiki.squirrel-lang.org/default.aspx/SquirrelWiki/SqPlus.html.

18. EasyBMP Cross-Platform Windows Bitmap Library. *SourceForge.* [Online] http://easybmp.sourceforge.net/.

19. AvalonEdit. *SharpDevelop Wiki.* [Online] http://wiki.sharpdevelop.net/AvalonEdit.ashx.

20. *OpenTK.* [Online] http://www.opentk.com/.

21. PropertyGridEx. *The Code Project.* [Online] http://www.codeproject.com/KB/tabs/PropertyGridEx.aspx.

22. **Cwalina, Krzysztof and Abrams, Brad.** *Framwork Design Guidelines: Conventions, Idions and Patterns for Reusable .NET Libraries.* 2nd. s.l. : Addison-Wesley Professional, 2008. 978-0321545619.

23. *Interactive Terrain Rendering: Towards Realism with Procedural Models and Graphics Hardware.* **Dachsbacher, Carsten.** 2006.

24. **Foley, James D.** *Computer Graphics: Design and Principles.* 2nd. s.l. : Addison-Wesley Professional, 1996. 978-0201848403.

25. **Grumet, Matthias.** *Terrain Modeling.* 2004.

26. **Schneider, Jens, Boldte, Tobias and Westermann, Rudiger.** *Real-Time Editing, Synthesis, and Rendering of Infinite Landscapes on GPUs.* 2006.