Interactive Adversarial Terrain Generation

by Hunter Borlik
Advisor Zoë Wood

Abstract

Virtual terrain authoring presents a challenging task due to both the fidelity and variety of terrain data. This project attempts to create a terrain authoring tool that simplifies the creation of elevation maps used in interactive applications. The final application is able to generate elevation maps based on a user provided sketch. The application also allows for real-time viewing of terrain and adjustment of erosion parameters.

Introduction

Artificial content generation has been a focus of computer graphics for a long time and terrains have been no exception. Terrain data is required for representing parts of the real world in an interactive form. A terrain is often the central element of a scene making up the majority of visible details. An image with pixel values that reflect the elevation at a point. This project aims to simplify the process of creating these elevation maps.

Authoring terrain data presents a challenge due the the variety of topographic features present in real-world terrains. Existing methods for artificial terrain generation can be classified into procedural, physics-based, and example-based.

The availability of detailed and realistic terrain data is limited. Other widely available terrain data lacks resolution: SRTM data is 30 meter per sample [Farr et al. 2007] . Limiting its usefulness in first person perspectives. The level of skill required for hand sculpting a height map restricts the availability of custom terrains. There is a need for terrain authoring tools that allow high user control through simple inputs.

In this approach, the user starts by providing a rough sketch of their desired terrain to the generator. The generator then returns a terrain corresponding to the input sketch. Then the user can further alter the input sketch to get desired terrain features. This iterative process allows for small changes, which simplify the overall artistic process. After the terrain is generated from a sketch, it is further refined through erosive process.

Related Work

The most popular methods of terrain generation algorithms rely on mixing together different types of coherent noise. Fractal methods [Cristea & Liarokapis 2015] excel at creating natural looking terrain data. However, coherent noise uses a random process to evaluate elevation, and as a consequence it is difficult to build a terrain with expected features. The user must vary the generator seed until they achieve a landscape layout with the desired underlying shape. These procedural methods are computationally efficient because they use fractal noise.

Guérin et al. [2017] use a similar approach to this project and generate terrains based on supplied user sketches. Using a conditional generative adversarial network (GAN) for fast example-based terrain

synthesis. This project uses a similar approach for synthesizing the initial terrain height maps. A different approach is used for erosion, and no terrain amplification is performed.

Zhao,Liu,Borovikov,Beirami,Sanjabi and Zaman [2019] use embedded themes called Generative Adversarial Terrain Amplification. This method takes a low resolution terrain as an input and increases the high resolution detail. The final output is a higher resolution terrain with features that match a given theme.

Algorithm

Terrain Generation

An important feature in this approach is its ability to synthesize terrain from user sketches. The generator used in this project is based on the pix2pix network from (3) using a similar architecture to (4). The GAN in this project is a modified pix2pix network. The pix2pix GAN is intended to be used in image to image translation. It is designed to map from one representation of an image to a different representation of that image. In this process, detail can be added to make the image look like a real terrain. The U-Net architecture tackles the problem of mapping a high resolution input grid to a high resolution output grid [Isola,Zhu,Zhou & Efros 2017].

Data and Preparation

Training data is prepared using an automated pipeline with libraries that index and download digital elevation data. The raw digital elevation tiles are first combined to form a composite terrain image, then re-projected and split back into smaller tiles of a more manageable size. These smaller tiles are used to generate input sketches.

The data used in training is from the publicly available NASA SRTM v1 dataset. This dataset contains terrain elevation data at a resolution of about 30 meters per pixel. Version one contains raw and uncorrected elevation data. It is missing data in some places, which creates holes in the terrain [Farr et al. 2007].

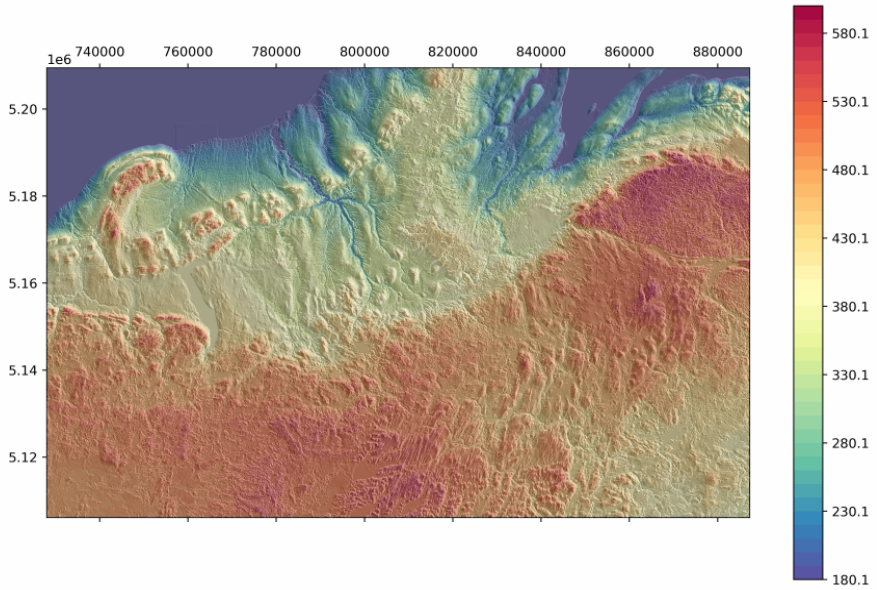


Figure 1: Digital elevation data prior to tiling: measurements in meters

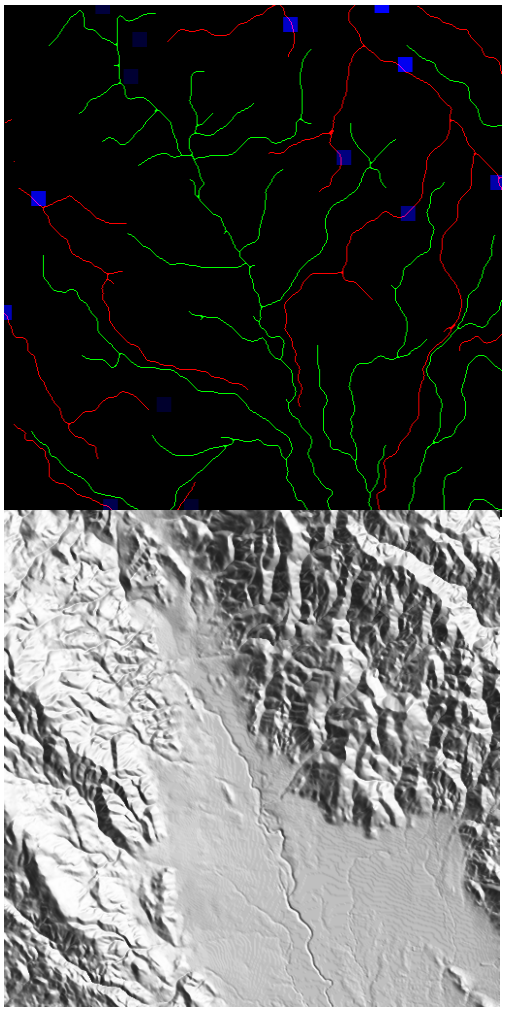


Figure 2: Example terrain tile data used in training: sketch (top) for elevation (bottom)

Tile extraction from large utm tiles. Tiles sized at 1200 by 1200 pixels, covering an area of 900 square kilometers.

Following Guérin et al. [2017], sketches are created by extracting topographic information from tiles: Rivers, ridges, *peaks*, and *basins* from real elevation data.

Rivers are detected by accumulating flow based on computed flow directions. A Python library, *pysheds*, supplies functions for both flow and accumulation. Flow is a vector field of downhill directions. Accumulated flow represents river networks as lines with cell values that reflect the number of uphill cells contributing to flow in the cell. Only the cells with an accumulation value exceeding a lower limit are used to generate sketches. To create a sketch from accumulated flow data, the accumulations are first normalized, then converted to a binary representation using a threshold. The binary river network image is then skeletonized and morphologically closed to create a clean 1-pixel width representation of all rivers.

Ridges are detected by inverting terrain elevations and applying the same approach used to detect *rivers*.

Peaks and *basins* are found using by first convolving the elevation with a max and min filter. If the difference in pixel values between outputs of these two filters is greater than a threshold value, then it is selected as a local maxima and its elevation saved.

These four types of features extracted from elevation data are then combined into a three channel image. The *ridges* and *rivers* are put into red and green channels. The presence of either of these features is indicated by a maximal value in the image data. Then the *peaks* and *basins* are combined to create an elevation target layer. These features are drawn into the sketch as a 15 by 15 pixel patch filled with a value that scales the desired elevation for that location. For these patches, the elevation target value is selected from the patch center.

Training

During training tiles are randomly flipped vertically and horizontally to give four times the original number of tiles. The the tiles are also randomly cropped to a 512 by 512 grid from the 1200 by 1200 input tiles. These cropped tiles, along with their corresponding *rivers*, *ridges*, *peaks* and *basins* make the data set. To further increase the variety of data in training, sketch channels are randomly zeroed. Each layer has a 50% chance of elimination. This has the effect of preventing the trained network from relying heavily on any single input layer. The generator learns to not strictly respect constraints in the sketches. Additionally, the network should then be able to generate outputs for sparser user sketches.

The river detection process yields very precise *river* and *ridge* networks. User supplied sketches will not strictly follow every river twist, rather they will provide course directions [Guérin et al. 2017. To reduce the preision of the generated sketches, the elevation data is blurred and downsampled prior to feature extraction.

The model uses a similar structure as pix2pix [Isola et al. 2017], but has a different loss function and network activation. The loss function for the generator is the Wasserstein distance [Arjovsky,Chintala & Bottou 2017] function combined with mean absolute error [Isola et al. 2017] [Guérin et al. 2017] between generated and target image. This loss function treats discriminator to act

as a critic rather than a classifier, allowing more meaningful gradients to be passed to the generator. Equalized learning rate [Karras,Aila,Laine & Lehtinen 2017] is used to scale all weights in both the generator and discriminator. Additionally, the discriminator has a linear activation on for the final layer.

Terrain Erosion

To improve the high-frequency detail of terrains generated by the GAN, they can be exposed to a process that models hydraulic and thermal erosion. The following erosion algorithm is parallelized to run on GPU.

Hydraulic erosion

Hydraulic erosion models a natural process of mechanical weathering as water flows a surface. Hydraulic erosion models will also model sediment transport so that topsoil can be transported from high to low areas. The hydraulic erosion algorithm uses virtual pipes to connect adjacent cells in the terrain. Water acts as a solvent, lifting and depositing sediment based on slope and other variables such as viscosity [Cristea & Liarokapis 2015] [Jákó & Tóth 2011]. The hydraulic erosion algorithm was modified to dampen velocities based on the amount of dissolved sediment. This erosion process is used to correct elevation features in the terrains created by the GAN. Details of the algorithm are in Appendix B.

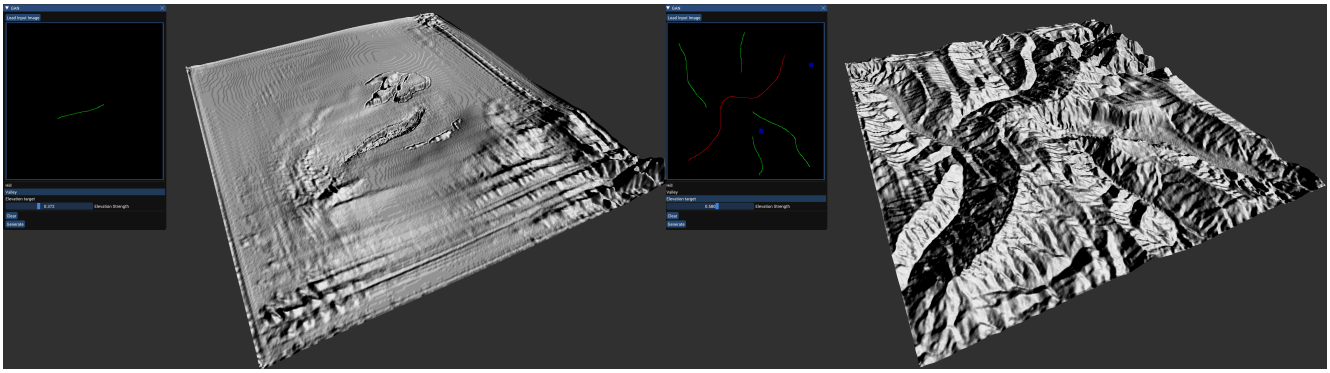
Thermal Erosion

The model used for thermal erosion follows Jákó and Tóth [2011] and Cristea and Liarokapis [2015]. First, in each cell, height differences between it and eight neighbors is found. If the angle formed between a neighbor and the current cell exceeds a limit set by the user, material is moved. This model causes near linear slopes to be enforced; similar to the structure of real sand.

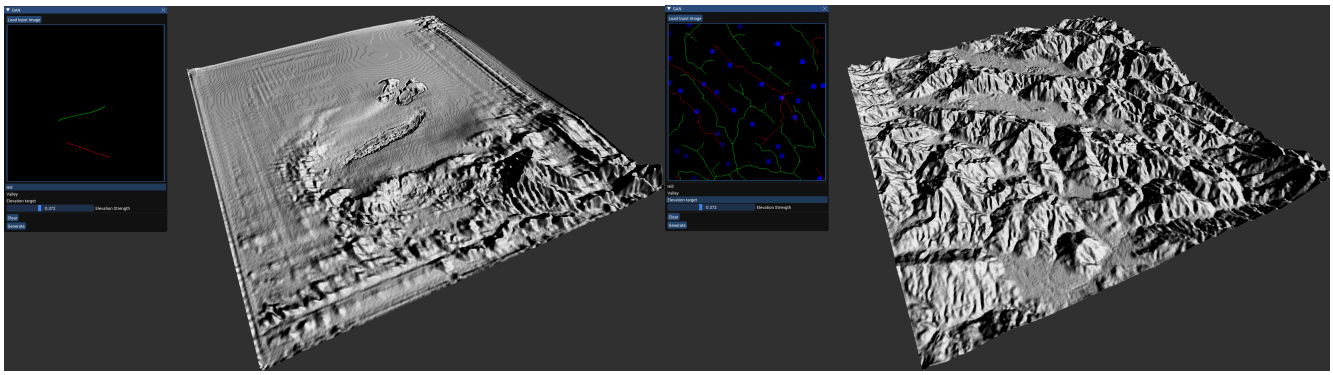
Results

Terrain synthesis

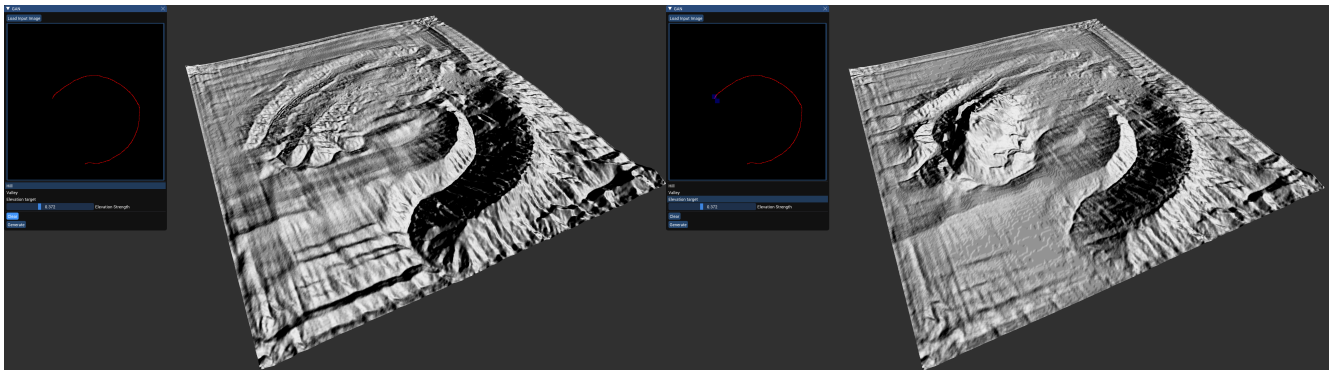
The trained network is able to output reasonable terrains for many input sketches.



The network struggles to generate terrains for sparse input sketches. There are many artifacts until a minimum amount of detain is added to the sketch. The above two images show how additional detail in the input sketch result in less artifacts in the generated terrain.

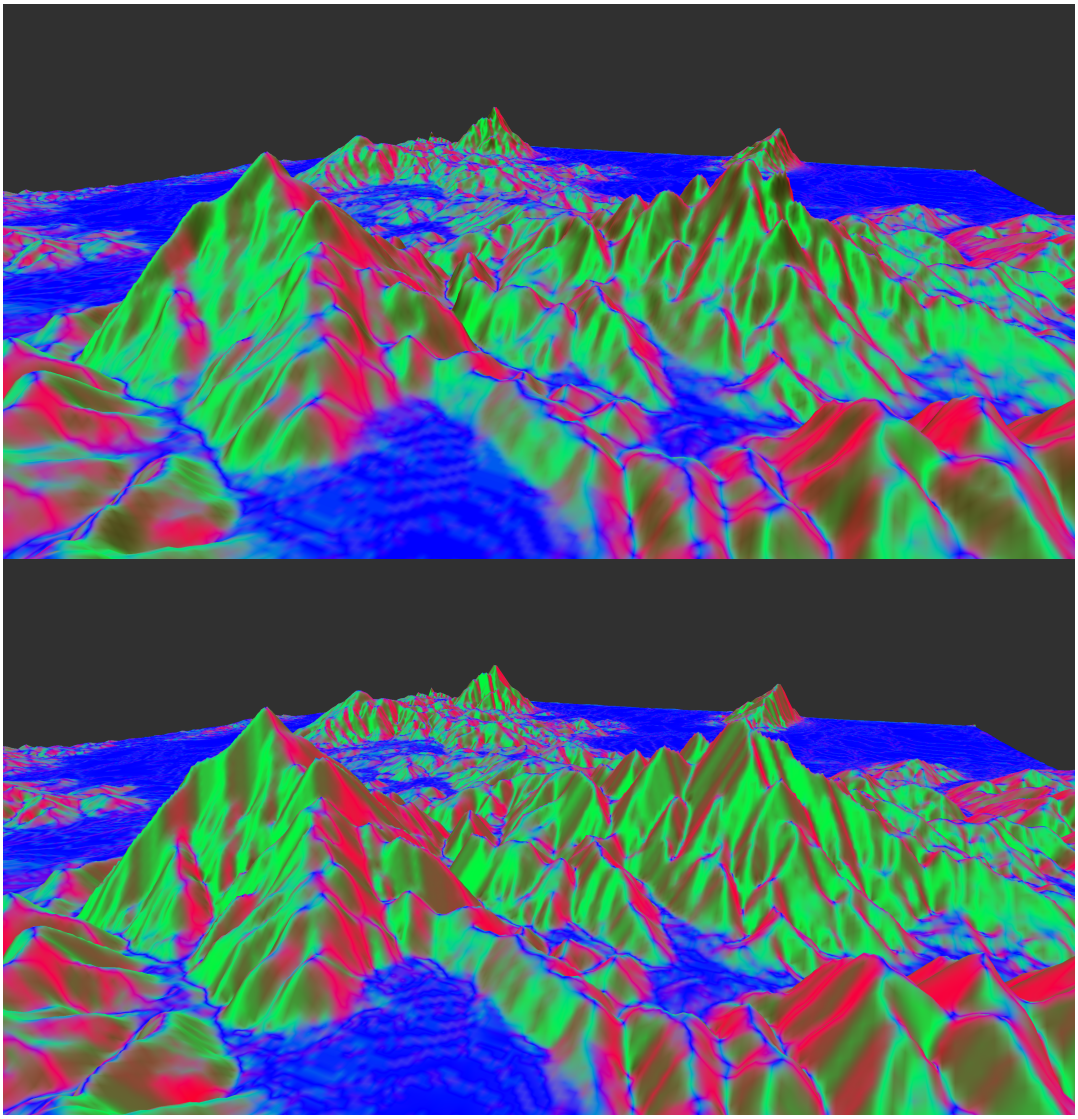


As the density of details in the input sketch increases, the network is less sensitive to changes and the weight of each stroke decreases.



Elevation targets have a much more influence over final elevation than either peak or valley lines. See additional outputs in Appendix A.

Erosion



The erosion parameters are difficult to tune for high-frequency detail. Low resolution terrain prevented much additional detail from being added. This erosion process often added more artifacts that it removed. The final algorithm occasionally introduced nan floats, which quickly propagate and can crash the application. The difficulty of debugging GPU programs, and poor texture initialization both contribute to this bug.

Reflection and Future Work

The most successful part of this project was the GAN. It was able to generate realistic looking elevation maps and worked for a variety of inputs. Prior to this project I had some experience with GANs, but there were many challenges in getting the final network to train properly. The method used by pix-2-pix regularly created networks with extreme layer weights.

Evaluating the network in python required a background webservice while the main application, written in C++, was running. Due to thread limitations in the web server, the model had to be reloaded regularly from disk. This reloading caused a large delay for every generated terrain. It was also not possible to export terrains.

The terrain maps used in this project were fairly low resolution and this limited detail over large areas. In the future, terrains should be upsampled to a higher resolution prior to erosion. The erosion process used here suffered from artifacts related to a large virtual cell size, and poor constraints within the algorithm. There are many other aspects of digital terrains that were not explored in this project. Objects such as rocks, trees, and vegetation could also be created as part of placement density maps. A major part that was also left out is terrain color. It could be possible to generate textures as part of the terrain synthesis process, or as a post-processing step. Additionally, real terrain has different material types throughout the terrain. Generating soil hardness and type at the surface could help improve the results obtained when using terrain erosion.

References

Arjovsky, M.; Chintala, S. and Bottou, L. (**2017**). *Wasserstein GAN*, .

Cristea, A. and Liarokapis, F. (**2015**). *Fractal Nature - Generating Realistic Terrains for Games*, 2015 7th International Conference on Games and Virtual Worlds for Serious Applications (VS-Games) : 1-8.

Farr, T. G.; Rosen, P. A.; Caro, E.; Crippen, R.; Duren, R.; Hensley, S.; Kobrick, M.; Paller, M.; Rodriguez, E.; Roth, L.; Seal, D.; Shaffer, S.; Shimada, J.; Umland, J.; Werner, M.; Oskin, M.; Burbank, D. and Alsdorf, D. (**2007**). *The Shuttle Radar Topography Mission*, Reviews of Geophysics 45.

Guérin, É.; Digne, J.; Galin, É.; Peytavie, A.; Wolf, C.; Benes, B. and Martinez, B. (**2017**). *Interactive example-based terrain authoring with conditional generative adversarial networks*, 36 : 1-13.

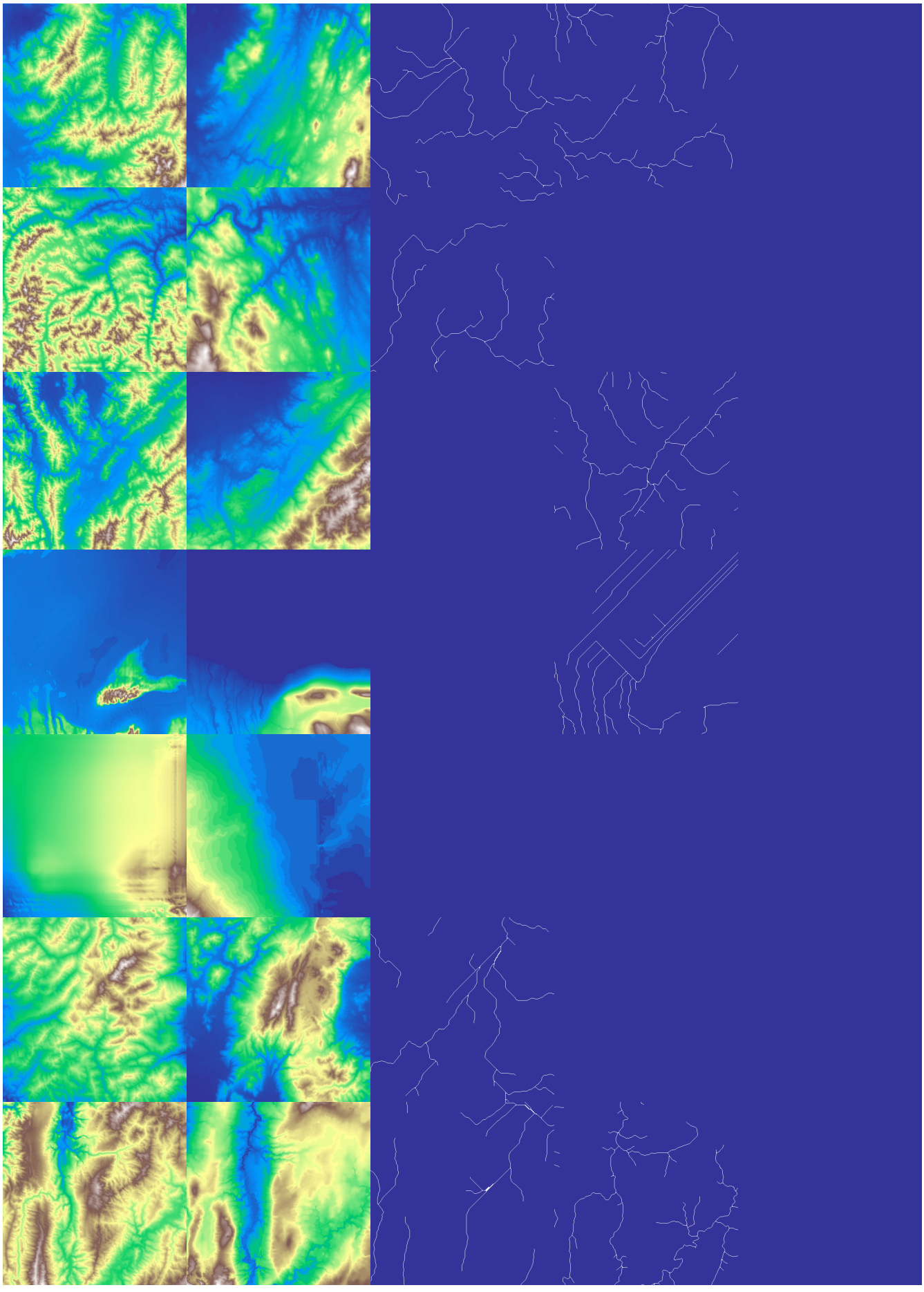
Isola, P.; Zhu, J.; Zhou, T. and Efros, A. A. (**2017**). *Image-to-Image Translation with Conditional Adversarial Networks*, : 5967-5976.

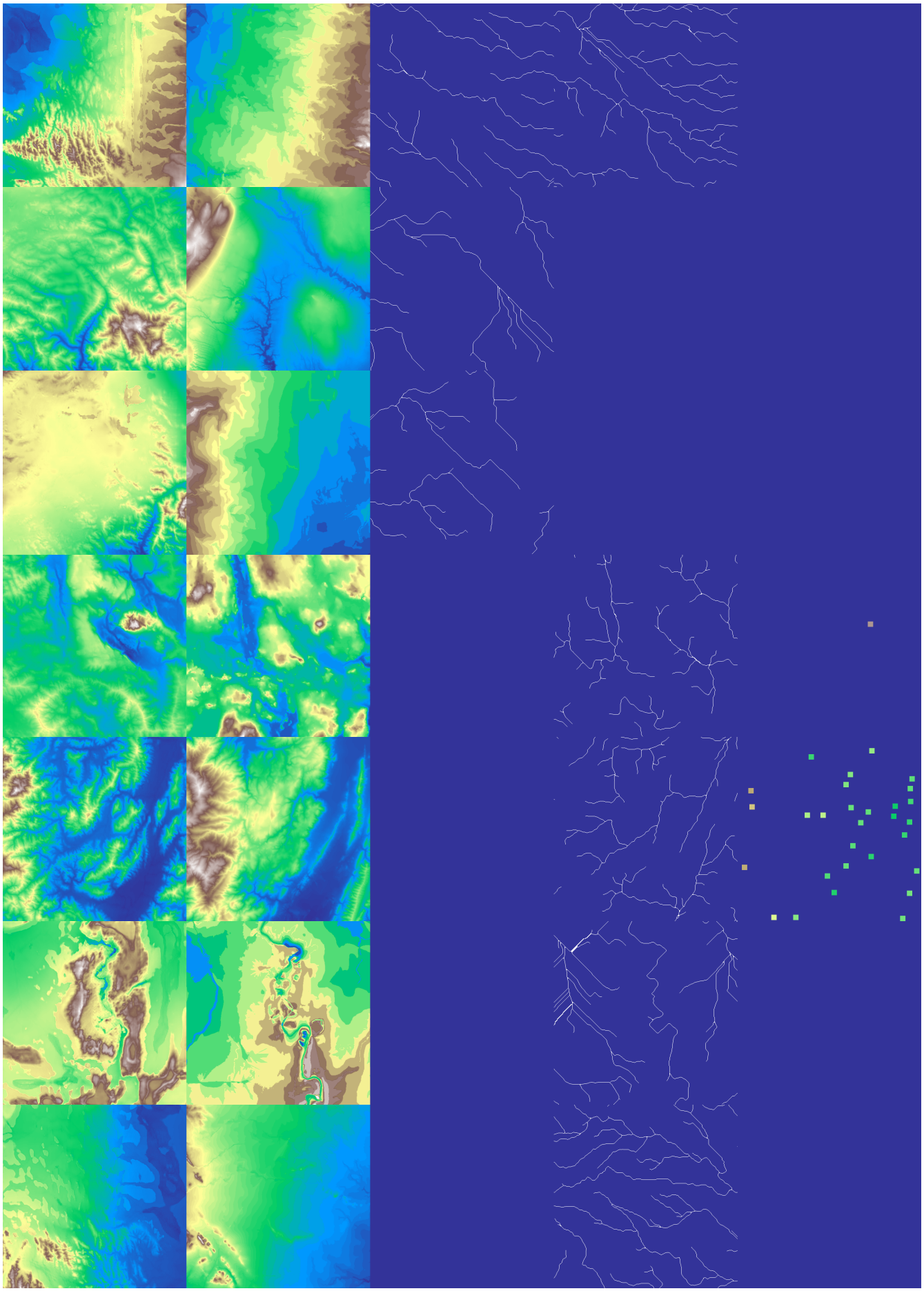
Jákó, B. and Tóth, B. (**2011**). *Fast Hydraulic and Thermal Erosion on GPU*, : 57-60.

Karras, T.; Aila, T.; Laine, S. and Lehtinen, J. (**2017**). *Progressive Growing of GANs for Improved Quality, Stability, and Variation*, CoRR abs/1710.10196.

Zhao, Y.; Liu, H.; Borovikov, I.; Beirami, A.; Sanjabi, M. and Zaman, K. (**2019**). *Multi-theme generative adversarial terrain amplification*, 38 : 1-14.

Appendix A: random selected outputs. output (left), real data, hills, valleys, elevation targets (right)





Appendix B

Hydraulic Erosion Detail

Each step of the hydraulic erosion stores data in different layers at each elevation point (x, y): water height, *w*; water outflow volume, $\mathbf{f} = (f^L, f^R, f^T, f^B)$; water velocity, $\mathbf{v}$ = (x, y); dissolved sediment, *s*; terrain height, t_h; and soil hardness, *h*. Global parameters for hydraulic erosion include: differential rainfall, R_r; differential water evaporation rate, R_e; soil suspension rate, S_r; soil deposition rate, S_d; soil softening rate, K_h; water evaporation coefficient, K_e; cell size, *l*; virtual pipe area, *A*; acceleration due to gravity, *g*; and water sediment capacity, K_C. Each iteration runs with a constant time delta, Δt. The steps of the hydraulic erosion occur in the following order:

First, rainfall increases the amount of water present in each cell. Rather than using a raindrop model like Cristea and Liarokapis [2015], where water is added through the modeling of individual drops, water is added to every cell at the same rate in each time step and scaled by the cell area. R_r, describes the rate rainfall per Δt per area, so the following describes the change in water height:

$$w_{[x,y]1} = w_{[x,y]}(t) + R_r \cdot \Delta t$$

After updating the water level due to rainfall, each cell's outflows to its neighbors are calculated. The outflow map is updated by checking the slopes of the terrain. Each cell has two sets of values describing flow along the four connecting virtual pipes of its neighbors. The amount of water in the cell is determined using the values of these four pipes. To minimize performance time, the inflows can be directly acquired from the cell neighbors [Cristea & Liarokapis 2015]:

$$inFlow^R_{[x,y]} = outFlow^L_{[x+1,y]}$$

Since no water can flow outside the terrain, border conditions must be checked for outflow and inflow lookup. In Jákó and Tóth [2011] and Cristea and Liarokapis [2015] the outflows are calculated with the following equation:

$$f^L_{[x,y]}(t + \Delta t) = max\left(0, f^L_{[x,y]}(t) + \Delta t \cdot A \cdot \frac{g \cdot \Delta h^L_{[x,y]}}{l}\right)$$

Where Δh^L is the difference in height between the current cell and its neighbor to the left.

$$\Delta h^L_{[x,y]} = w_{[x,y]} + t_{h[x,y]} - w_{[x-1,y]} - t_{h[x-1,y]}$$

The calculation of f^R, f^T, *and* f^B is performed similarly. These above equations are based on Darcy's law for viscous fluid flow in a porous medium such as a bed of sand. This law forms the basis of hydro-geology. This equation is stable in some situations, however, it is not dampened and suffers from oscillations due to the tendency to build a flow momentum. If the area of the virtual pipes, *A*, is not matched to the cell size, *l;* the water levels become unstable as volumes move between neighboring cells. By integrating a viscosity damping factor into the volume flow calculation, the system becomes stable over a larger time-step domain. Darcy's formula for head loss is as follows:

$$h_L = f_c \cdot \frac{k\, l\, v^2}{2 D\, g}$$

Where D is the diameter of the virtual pipe; the friction coefficient, f_c; and porosity constant, k. In this equation, the porosity is used to model how easily groundwater flows through its medium. By substituting head loss due to friction and velocity from flow of the previous time step, the following equation is found:

$$f^L_{[x,y]}(t+\Delta t)=max\left(0, f^L_{[x,y]}(t)+\Delta t\cdot\left(A\cdot\frac{g\cdot\Delta h^L_{[x,y]}}{l}-f_c\cdot\frac{k\cdot f^L_{[x,y]}(t)^2}{4\,g\,\Delta t^2\,A\sqrt{\frac{A}{\pi}}}\right)\right)$$

Where k is replaced with a function that increases viscosity as the water becomes more saturated with sediment:

$$k(s_{[x,y]})=\frac{s_{[x,y]}}{A\cdot K_C}+1$$

If the calculated outflows are greater than the amount of water in the cell, then more water will be removed from the cell than is available. Again similar to Cristea and Liarokapis [2015], **f** is scaled down with an factor based on available volume of water. However, the flows are not scaled by the time step constant:

$$K=min\left(1, \frac{A\cdot w_{[x,y]}}{f^L+f^R+f^T+f^B}\right)$$

Then the outflows are multiplied by K:

$$f^i_{[x,y]}=K\cdot f^i_{[x,y]}, i=L, R, T, B$$

At this point, all the water flows for the step have been found. The change in water volume, ΔV, for the cell at (x,y) can then be found by adding inflows and outflows:

$$\Delta V_{[x,y]}=f^L_{[x+1,y]}(t+\Delta t)+f^R_{[x-1,y]}(t+\Delta t)+f^T_{[x,y+1]}(t+\Delta t)+f^B_{[x,y-1]}(t+\Delta t)-\sum_{i=L,R,T,B} f^i_{[x,y]}(t+\Delta t)$$

Then the change in water height, w:

$$w_{[x,y]2}=w_{[x,y]1}+\frac{\Delta V}{A}$$

Using the flow values, the net velocity of water moving through the cell is also found. The velocity field is needed to calculate hydraulic erosion and deposition [Jákó & Tóth 2011]Since the flow values store volumes moving through the cells in each time step, the original equation is modified to account for difference in values. Net flow volume is scaled by cell size, water depth, and the time step constant to get water velocity across the cell surface:

$$\boldsymbol{v}^x_{[x,y]}=\frac{f^R_{[x-1,y]}-f^L_{[x,y]}+f^R_{[x,y]}-f^L_{[x+1,y]}}{\Delta t\cdot A\cdot w_{[x,y]}}$$

The y component of water velocity, $\mathbf{v}^{y}_{[x,y]}$ is calculated similarly. Using the velocities, terrain heights are changed through erosion and deposition. Each cell has a sediment transport capacity, $C_{[x,y]}$, that is calculated in each time step. The transported sediment, $s_{[x,y]}$, is adjusted through soil uptake and deposition to match the target sediment capacity. First, transport capacity is found:

$$C_{[x,y]} = max\left(0, K_C \hat{\boldsymbol{n}} \cdot \hat{\boldsymbol{k}} \left|\boldsymbol{v}_{[x,y]}\right| l_{max}\left(w_{[x,y]}\right)\right)$$

Where $\hat{\boldsymbol{n}}$ is the surface normal, $\hat{\boldsymbol{k}}$ is the z-axis, and $\hat{\boldsymbol{n}} \cdot \hat{\boldsymbol{k}}$ has a minimum value so flat areas can still move some sediment when the normal is close to vertical. Areas with steeper slope will erode faster. l_{max} is a ramp-like function that prevents large amounts of water from eroding terrain too quickly [Jákó & Tóth 2011]. Once the sediment capacity is found, there is a decision made based on current dissolved sediment and the cell's capacity. If there is less transported sediment $s_{[x,y]}$ than capacity, then soil is dissolved in an uptake step:

$$\Delta s_{[x,y]} = min\left(t_{h[x,y]}, \Delta t \cdot h_{[x,y]} \cdot S_r\left(C_{[x,y]} - s_{[x,y]}\right)\right)$$
$$\Delta t_{h[x,y]} = -\Delta s$$

Otherwise, if $s_{[x,y]} < C_{[x,y}$, soil is deposited back to the terrain:

$$\Delta s_{[x,y]} = -\Delta t \cdot S_d\left(s_{[x,y]} - C_{[x,y]}\right)$$
$$\Delta t_{h[x,y]} = \Delta s$$

In nature, moving sediment softens over time [Jákó & Tóth 2011]. To immitate this, the soil hardness parameter is updated when soil is deposited. Here, the soil hardness has an initial value of zero, preventing erosion in the first iteration. The value is then updated to reflect R_{min}.

$$h_{[x,y]} = max\left(R_{min}, h_{[x,y]} + \Delta t \cdot K_h S_r\left(s_{[x,y]} - C_{[x,y]}\right)\right)$$

Where K_h is a global parameter controlling rate of sediment softening; R_{min} is the lower limit of soil hardness. The next step in the model is to transport sediment along water velocities, $\boldsymbol{v}_{[x,y]}$. The sediment is moved from upstream:

$$s_{[x,y]} = s_{[(x,y) - \boldsymbol{v}_{[x,y]} \cdot \Delta t]}$$

If the point $(x - \boldsymbol{v}_x \Delta t, y - \boldsymbol{v}_y \Delta t)$ is not on the grid, linear interpolation between four neighboring points is used. The sediment values for outside of grid points are copied from the nearest valid cells. The last step in hydraulic erosion is to simulate evaporation through exponential decay. It is important to prevent the scene from filling up with with liquid [Jákó & Tóth 2011]:

$$w_{[x,y]}\left(t + \Delta t\right) = w_{[x,y]2}\left(1 - K_e \cdot \Delta t\right)$$