Visually appealing water flow over a terrain

Eddie Lau
CSCI-6530 Advanced Computer Graphics Final Project 2010
Rensselaer Polytechnic Institute

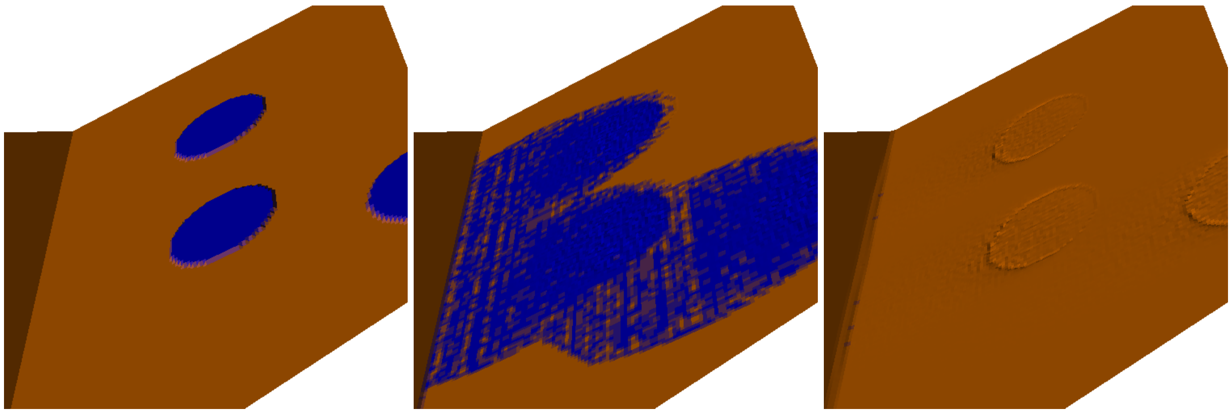

Figure 1: *Water flow simulation of a scene with three water pools on a slope. Initial (left), In-between (middle), Final (right)*

Abstract

This project implements a program to simulate water flow over terrains. As in many other computer graphics applications, we look for reasonably fast computation and rendering. So the prediction of exact quantities as stated in physical laws is not our goal. However to ensure our results look visually appealing, we still need to apply the physical laws to a certain extent so as to account for several characteristic features of the phenomenon, including water flow, soil erosion and deposition, and evaporation. Which physical laws to use and how much they should be simplified are challenges facing us. The framework introduced by Benes et. al. [Benes and Forsbach 2002], which is later extended by Anh et. al [Anh et al. 2007], is our final choice. It is which we find to-date balancing the two conflicting requirements well. The model adopts a layered data structure to store the elevation, water volume, dissolved material volume and water velocity of every grid point of the terrain. Every simulation step is discretized into a few operations to decide how much soil gets eroded, which direction(s) the water should flow, how much soil should be deposited, and how much water to be evaporated.

Keywords: simulation, water flow, terrain

1 Introduction

My project aims at creating visually appealing water flow animations over a terrain. By appealing we expect more than deriving the river locations, assigning a single water flow direction for each of them and finally animating water flow along the streams. We wish it to show some characteristics phenomena happened in the real world, including water spreading to all deeper neighbors (instead of just one particular direction), and terrain changed due to soil erosion and sedimentation as water passes.

The paper by [Benes and Forsbach 2002] introduces a simple framework capable of simulating all the phenomena mentioned above. Its use of a layered data structure to store the essential terrain attribute values like elevation, water and dissolved sediment amounts, together with the way it takes to handle water movement and soil erosion and deposition on the course of simulation, are still the foundation of many successors like the one presented by [Anh et al. 2007], which features a few improvements over the original Benes' model, in particular the incorporation of water speed into consideration of sediment capacity.

This paper summarizes my final model mostly referenced from Anh et, al., with side notes on a few issues not explicitly addressed in both Benes' and Anh's articles. It is structured as follows: In section 2, I present the details of the data structure. In section 3, I will go through the dynamics of the simulation system, including water flow, sediment dissolution and deposition, and water evaporation. In section 4, I will describe the overall simulation process. This is followed by a result and discussion in section 5, before coming to a conclusion and further works in section 6.

2 Data model

If we assume the terrain to have no concave structures like caves and overhangs, what we need to know ultimately in order to render the terrain are the single elevation and the amount of water at each point over a regular 2D spatial grid. While numerous more compact models have been developed, a layered data representation in which we keep a single value for each attribute at every spatial location, is the choice of Benes et al and their successors. Data can be readily updated and rendered without much additional processing, which simplifies coding and speeds up rendering for interactive display.

To model sediment transport, Benes introduces an additional attribute called *dissolved sediment* to store the corresponding amount. To allow water speed to affect the sedimentation process as in real-

Attribute	Symbol
Elevation	H
Water Amount	W
Dissolved Sediment	S
Velocity (3D)	$\vec{v}$

Table 1: *Attributes for each spatial location*

ity (in which a faster current is more likely to carry soil with it than the lower current), Anh et. al. adds a 3D vector called *velocity* for each spatial location. Table 1 below summarizes the layers in the spatial data grids of our program.

3 Operations

3.1 Water Flow

To simulate water flow over a terrain, we need to decide in what direction and the amount that the water at each location is going to flow. Velocity is updated as well, as it speeds up on travelling down a slope and gets retarded due to sliding friction.

3.1.1 Direction

[Hengl and Reuter 2008] gives an excellent overview of conventional algorithms for finding the direction of water flow in a terrain.

D-4 and D-8 [O'Callaghan and Mark 1984] are a few *single-neighbor algorithms* commonly used to derive a river network from a given elevation grid. Both assume water flows towards a single direction with deepest descent, and therefore direct water to the deepest location from among the four or eight immediate neighbors. Either one is reported to work reasonably fine on finding river locations: the deepest direction is likely to obtain most water to survive under evaporation. But the total lack of water flow to all the other directions means erosion or sedimentation which should have been there (as in reality, some water flows to these directions anyway) are missed. Another problem is that the candidate flow directions are discretized to multiples of 90/45 degree. If the actual deepest slope is somewhere between some two adjacent discrete directions, the algorithm has to pick one of them. Huge errors can be accumulated as the water flow across more and more locations.

D-infinity algorithm [Tarboton 1997] virtually allows flow to any directions. The trick is to distribute the water over the two neighboring cells proportionate to how close the deepest descent direction is relative to the two approximate discrete directions. However, this variant still cannot model possible flows in opposite directions, as at real mountain peaks or ridges.

In contrast, a *multiple-neighbor algorithm* distributes water to all 8 discrete neighboring directions at the same time, as long as they are deeper than the current center location. Water that flows away is partitioned to the immediate neighbors proportional to how much lower neighboring water levels are relative to that of the current location. Suppose ΔW is the total amount of water flowing away from the current cell (whose value is discussed in the next subsection), and Δhw_i is the water level difference between the current location and the neighborhood location indexed i (for 8-neighborhood, $i = 1, 2, ..., 8$). The amount of water that flows to a particular neighboring spatial location i, ΔW_i, is evaluated as

$$\Delta W_i = \Delta W \frac{\Delta hw_i}{\sum_{n=1}^{8} \Delta hw_n} \tag{1}$$

Because of the above water partitioning scheme, the flow direction is virtually not confined to a single direction as in the D-infinity algorithm. Also, we now can distribute water to multiple directions even if they are not adjacent. Because I see these two features are essential for visual-appealing modeling of the reality, I adopt this multiple-neighbor, just like Benes and Anh do.

3.1.2 Amount

There is an outflow as long as there is at least one neighbor with lower water level. But for the total amount, [Benes and Forsbach 2002] highlights that it may, or may not, be the entire amount present in the location. They use a simplified case with just two spatial locations, which is reproduced in Figure 2, to explain the issue. The point is that if the water level of the highest deepest neighbor is higher than the elevation level of the current location (the top case), blindly moving all water to the neighbors leads to the neighboring locations now having water level higher than that of the current location. This is impossible with real world large-scale water flow. In such a case, **the total amount is set such that after flowing the appropriate portion to the highest deeper neighbor (according to Equation 1), the water levels of the current location and that neighbor location are the same (Figure 2 top)**.

If it is not the case, removing all water from the current cell is fine. (Figure 2 bottom)

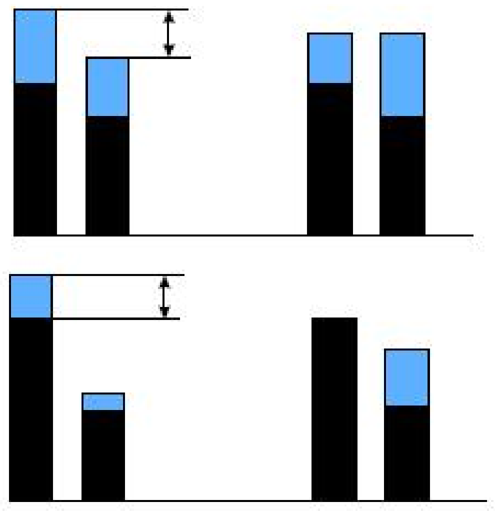

Figure 2: *The two different cases for total water transport amount calculation (top and bottom), reproduced from Benes' paper. In both cases, water is to be flown from the left location to the right. Initial conditions are shown on the left figures and the expected final conditions are described by the right figures.*

3.1.3 Velocity

Ahn et. al. suggests the concept of water velocities on top of the Benes' original model. It is indeed essential to model how water speed affects sediment transportation over the terrain, as discussed in the next subsection.

Water at a spatial location is accelerated when the location receives water flowing downwards from some neighbor. The acceleration for that portion of moved water, $\vec{a}$, can be calculated according to the geometric model suggested by Anh et. al. (Figure 3).

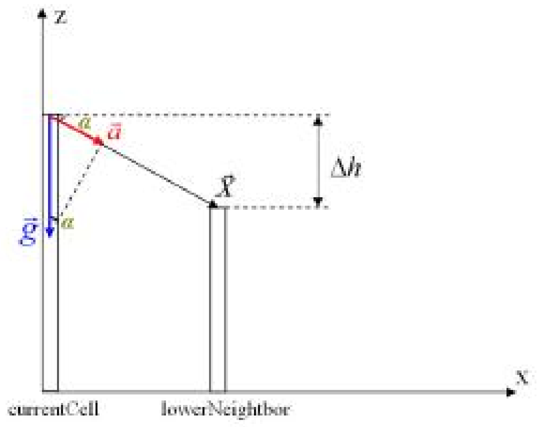

Figure 3: *The geometric model for acceleration calculation, reproduced from Anh's paper*

$$\vec{a} = \vec{g} sin\alpha = \vec{g}\frac{\Delta h}{|\vec{X}|} \quad (2)$$

However, as the water moves across spatial locations, or simply as time goes on, it is also facing sliding friction. This is modeled by applying a retardation factor $K_a \in [0...1]$ to the water velocity of every spatial locations at every simulation step.

The following formula computes the velocity of a volume of water moving down to the neighbor $v_{t+\Delta t}$, based on its velocity at the previous time step v_t, acceleration $\vec{a}$ and the duration of a simulation step Δt (which represents how long it takes for water to travel from one location to an adjacent one). This quantity is used when this amount of water is mixing with water in the destination location, for computing the new water velocity there (our next subsection will have a discussion on this issue).

$$\vec{v}_{t+\Delta t} = \vec{v_t}(1 - K_a) + \vec{a}\Delta t \quad (3)$$

It should be noted that this velocity attribute does not affect how much water is carried out from one spatial location to another in each simulation step. That amount is determined solely by the water level difference between adjacent spatial locations. Within one simulation step, all such water level inbalances are settled.

When a lower neighbor cell receives the water ΔW with velocity $\vec{v}_{\Delta W}$, its water amount and velocity are updated as follows.

$$W_{new} = W + \Delta W \quad (4)$$

$$\vec{v}_{new} = \frac{W\vec{v} + \Delta W \vec{v}_{\Delta W}}{W_{new}} \quad (5)$$

The following simple formula slows down water in the current location in every simulation step, which occurs as time goes on.

$$\vec{v}_{t+\Delta t} = \vec{v_t}(1 - K_a) \quad (6)$$

3.2 Sediment dissolution and deposition

This is another interesting phenomenon that one would expect for a visually realistic water flow simulation. Soil gets eroded at the upstream and finally deposited at the downstream.

I adopt the model by Ahn et. al., which assumes the sediment transportation capacity of water S_{cap}, is proportional to both the total amount of water W and speed $|v|$.

$$S_{cap} = K_c W|v| \quad (7)$$

So whenever W or $|v|$ changes (due to water flow towards the location, natural water slowdown, or evaporation, described below), we compute the new S_{cap} and then compare it with the actual sediment amount S.

If $S > S_{cap}$, then deposit

$$H_{new} = H + K_d(S - S_{cap}) \quad (8)$$
$$S_{new} = S - K_d(S - S_{cap}) \quad (9)$$

Otherwise dissolve

$$H_{new} = H - K_s(S_{cap} - S) \quad (10)$$
$$S_{new} = S + K_s(S_{cap} - S) \quad (11)$$

where $K_d \in [0...1]$ and $K_s \in [0...1]$ control the rate at which sediment is deposited and dissolved.

In contrast, in Benes' original model, S_{cap} is assumed to be related to the total water amount only. Velocity does not play any role. While it is fine for sediments that is dissolvable even in static water (like salt), it may not be adequate for sediments which will eventually get settled down once the water is stopped (like sand).

One point to note is that on dissolution, we need to make a height boundary check: make sure $K_s(S - S_{cap})$ is not larger than H before doing the update. Otherwise we may end up with negative heights, as shown in Figure 4 below.

3.3 Evaporation

This accounts for water loss from time to time as the water molecules gain enough energy to escape from the liquid surface. As described by Benes et. al., in reality the amounts of water evaporated depends on the temperature and the area of the water surface. With our regular grid representation, the surface area is more or less the same for all the grid locations. Further suppose that the temperature is constant over the same area. The evaporation of the water can be described as

$$\frac{dW}{dt} = -K_e W \quad (12)$$

where K_e is the evaporation coefficient talking about the speed of water evaporation per unit volume of water, and W is the total amount of water in the location. This implies the amount of water decreases exponentially as time goes on.

Meanwhile, I adopt the following simplified version suggested by Anh et. al, which has the advantage of simpler computation. It says the amount of water survived from evaporation at the next simulation step, $W_{t+\Delta t}$, is reduced by a factor K_e when compared with that at the current simulation step W_t.

$$W_{t+\Delta t} = W_t K_e \quad (13)$$

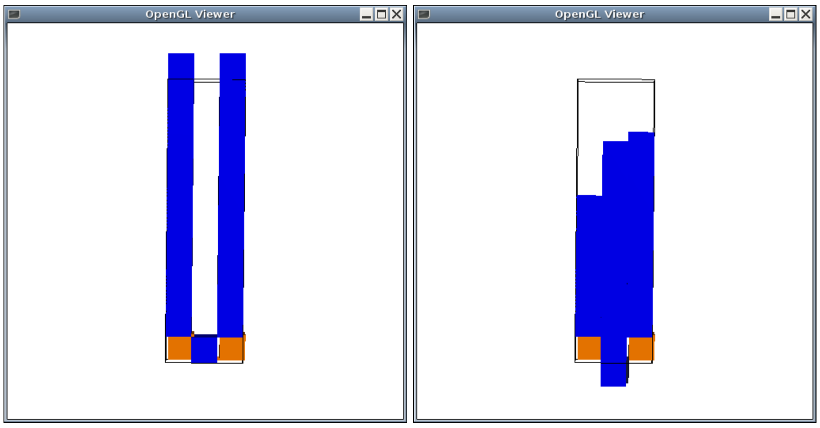

Figure 4: *Buggy result for forgeting height boundary check on dissolution. Before (top), After (Down)*

In either case, the amount of water will never reach zero, which may cause problems to system expecting complete evaporation at the end of the simulation. So Benes suggests if the water amount drops below a certain threshold T, the value is simply set to zero.

4 Simulation flow

4.1 Algorithm overview

The operations are combined together in the manner specified in Algorithm 1 to proceed the *forward erosion simulation*, in which every spatial location is checked one by one to deliver water to lower neighbors.

4.2 Grid process randomization

In my earlier implementation, I processed the spatial location grid from top to bottom, from left to right all the time. But the results was consistently unsymmetrical even when I worked with symmetric terrains, like *staircase2d* presented in Figure 5. To remedy, I randomize the processing sequence for each simulation step. The result looks much better now (See Figure 6 for the results of the randomized version).

5 Results

While existing ray-tracing software package like POV-Ray [Persistence of Vision Raytracker 2008] can readily render height fields with water, finally I decide to modify the raster renderer we have for assignment 2 for immediate result display. This feature allows me to visualize the simulation result for every simulation step as soon as it is available, which is useful for debugging purpose.

To verify correct implementation, first I start with a 2D scene *staircase2d* (Figure 6). Going down the slope from either the left hand side or the right hand, at the beginning more and more soil gets eroded, as water travelling down the slope are getting more and more kinetic energy to carry soil with them. But after some point, the amount of soil eroded is decreasing as the water has already taken considerable amount of soil from the upstream, leaving behind less and less capacity for new soil there. Even water speed increase cannot help much. Finally when reaching the bottom, water with opposite velocities mixes together, velocities vanish. At the same time, it has nowhere to go and stay there, waiting for evaporation. Both factors favor massive soil deposition there.

Scene	Grid Resolution	Frame per second
staircase2d	255×1	160
noise	128×128	44
pool	256×256	11

Table 2: *Frame rate for the three test scenes using a Ubuntu 10.04 Desktop machine with 4 Intel Xeon CPU of speed 2.4GHz, 4GB memory and ATI Radeon R300 ND*

I have also attempted to reproduce a few scenes presented in Benes' paper with my system. In the *noise* scene (Figure 7), a hole on an artificial white noise surface is filled with water that evaporates. Similar to Benes' result, we get the bottom of the pool turn flat. Water moves towards the center deep region with some soil. Water gets evaporated there at the center, leaving behind sediment contributing to the flat area.

Another scene named *pools* (Figure 8) involves three water pools on one non-steep slope. This implies the deposition process may occur right on the slope, well before the water gets at the valley. As expected, we obtain bumpy results as Benes on the part of the slope where the water passes. As the soil is repeatedly deposited and dissolved, characteristic ridges are formed on the slope and down at the bottom of the hills.

I use a Ubuntu 10.04 Desktop machine with 4 Intel Xeon CPU of speed 2.4GHz, 4GB memory and ATI Radeon R300 ND to perform the simulation. Table 2 shows the frame rates of the above three test scenes. Even the most demanding *pool* can be rendered interactively with a reasonable frame rate.

6 Conclusion and future works

In this paper I summarize what I have learnt from Benes' and Anh's works and implemented for my terrain water flow simulation program. A layered data structure recording the height, water amount, sediment amount of each spatial location is adopted to capture necessary data compactly for fast interactive rendering. Multiple-neighbor flow is allowed so as to enable flow divergence which occurs in the real world. The total amount of flow is carefully decided to ensure no overflow. Water speed is updated on the course of simulation so as to model how this factor affects sediment dissolution and deposition. Removing water away from the system is possible with water evaporation.

Currently my system does not allow flow beyond the terrain bound-

Algorithm 1 Forward erosion simulation

```
while step < stepNumber do
  for all locations in the Height Map do
    Determine the total amount of water to be removed from current location (Figure 2)
    for all lower neighbors do
      Calculate acceleration of water to be moved to that lower neighbor (Equation 2)
      Calculate velocity of water to be moved (Equation 3)
      Move the appropriate part of water from current location to this neighbor (Equation 5)
      Calculate sediment capacity of the moved water (Equation 7)
      Deposit or dissolve sediment carried by the moved water to destination cell if needed (Equations 9 and 11)
    end for
    Slow down water in the current cell (Equation 6)
    Calculate sediment capacity of water in the current cell (Equation 7)
    Deposit some sediment of current cell if needed (Equation 9)
  end for
  Water evaporation and distribution (Equation 13)
end while
```

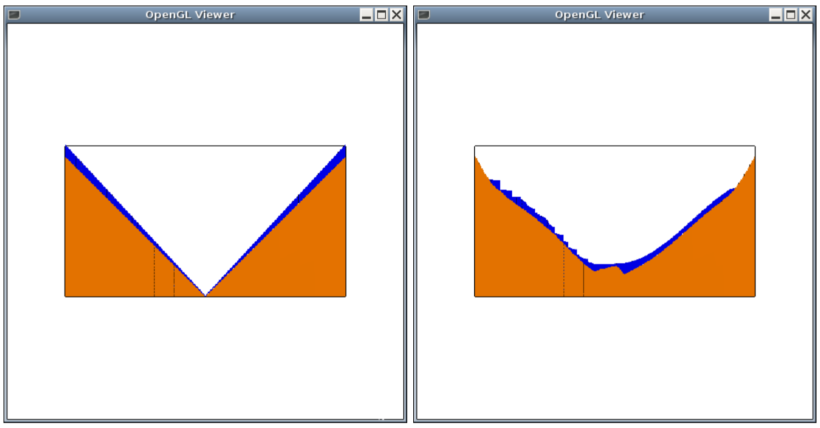

Figure 5: *Buggy result for not randomizing the processing sequence of spatial locations. Initial (left), Final (right)*

ary. A possible future work is to allow this to happen. Also, at one simulation step the water levels, water flow to lower neighbors have to be completed. There may be room for some techniques to allow frames in-between to be interpolated. For the interactive display, the use of interpolated tetrahedral mesh as described in [Stuetzle et al. 2009] looks to provide an even better visualization with a low cost and is worth a try.

Additional Information

This is an individual project. It takes me around 50 hours to finish this work.

References

ANH, N. H., SOURIN, A., AND ASWANI, P. 2007. Physically based hydraulic erosion simulation on graphics processing unit. In *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, ACM, New York, NY, USA, ACM, 257–264.

BENES, B., AND FORSBACH, R. 2002. Visual simulation of hydraulic erosion. *Journal of WSCG 10*, 79–86.

HENGL, T., AND REUTER, H. I. 2008. *Geomorphometry: concepts, software, applications*. Elsevier Science.

O'CALLAGHAN, J. F., AND MARK, D. M. 1984. The extraction of drainage networks from digital elevation data. *Computer Vision, Graphics, and Image Processing 28*, 323–344.

PERSISTENCE OF VISION RAYTRACKER, 2008. POV-Ray. `http://www.povray.org`, (retrieved Apr 7, 2010).

STUETZLE, C. S., CHEN, Z., PEREZ, K., GROSS, J., CUTLER, B., FRANKLIN, O., AND ZIMMIE, T., 2009. Segmented height field and smoothed particle hydrodynamics in erosion simulation.

TARBOTON, D. G. 1997. A new method for the determination of flow directions and upsloe areas in grid digital elevation models. *Water Resources Research 33*, 2 (February), 309–319.

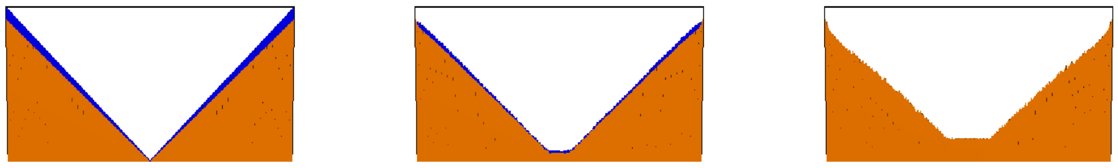

Figure 6: *Simulation result of the scene staircase2d. Initial (left), In-between (middle), Final (right)*

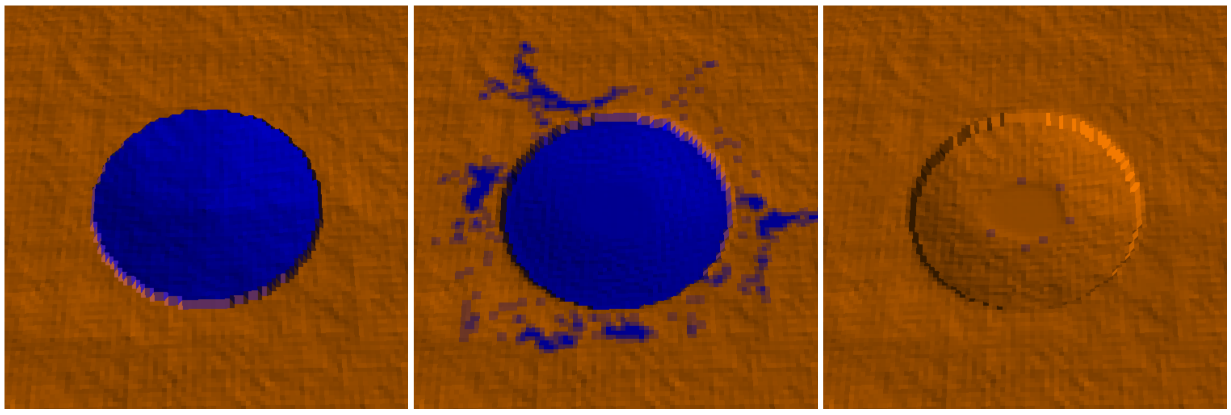

Figure 7: *Simulation result of the scene noise. Initial (left), In-between (middle), Final (right)*

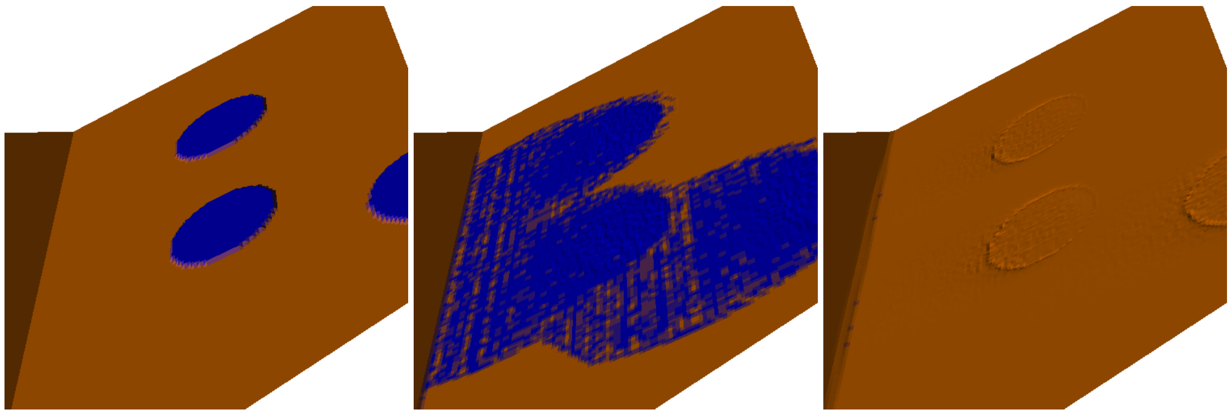

Figure 8: *Simulation result of the scene pools. Initial (left), In-between (middle), Final (right)*