Animating Raindrops on a Surface

Alina Chin

Abstract

Rain falling on a window is difficult to animate because of the number of droplets and the complexity of their behaviour. We propose a method of animating raindrops that is suitable for real-time rendering, using a combined particle system and height map to model the behaviour of water on a flat surface.

1 Introduction

Weather phenomena can make a rendered scene more effective. Rain falling on a windshield or window is a fairly common phenomenon. In an animation or video game, we would like to animate the droplets in a believable way. However, we do not want to restrict ourselves to a precise physical simulation because of the complexity involved. We would like our method to be relatively time-efficient, even for a large number of droplets, which is particularly important for video game applications.

2 Literature Review

Much of the existing literature is based on Kaneda et al's discrete method for water drops on hydrophilic surfaces [1993]. It is notable for modelling the interaction between the water and the surface. Water drops are represented by particles that move from cell to cell on a grid covering the surface. Each cell holds water mass left behind by moving drops. The authors later expanded the method to include curved surfaces, moving obstacles (e.g. windshield wipers), and depth-of-field effects [Kaneda et al. 1999]. Rendering is done using cube-mapping. Neither of these methods (as implemented on the CPU) gave results in real-time, making them unsuitable for use in video games.

Another paper augments Kaneda et al's method with a multi-texture mapping technique to improve rendering speed [Sato et al. 2002]. Although an improvement on the 1.25min rendering time in [Kaneda et al. 1999], the rendering speed with depth-of-field effects enabled still falls short of real-time (12-14 fps).

Real-time rendering methods for diverse rain phenomena are covered in the ATI ToyShop paper [Tatarchuk and Isidoro 2006]. Notably, they implement [Kaneda et al. 1999]'s method on the GPU, which bumps the speed up to real-time. They use the grid to generate a normal map which is used to produce shadows, reflection and refraction, and pseudo-caustics.

Similarly, Chen et al use a height map combined with Kaneda et al's particle system to simulate more realistic drop shapes and residual droplets from a stream [Chen et al. 2013]. They apply various convolutions to the height map in order to represent the thinning over time of water trails left by running droplets. The height map is also used for collision detection, since the shapes of droplets can be irregular. Their results rendered at 15-18 ms per frame, which satisfies the real-time requirement.

Nakata et al addressed a similar problem using rigid-body physics to model droplets on curved, hydrophobic windshields [Nakata et al. 2012]. To reduce rendering costs, large and small droplets are rendered using different data structures: small static droplets are represented by a normal map, and large droplets by pill shapes. This technique incorporates many physically-based effects, such as wind drag, contact angle hysteresis drag, Laplace pressure, and the Lotus effect. It is implemented on the Unity game engine and achieves impressive rendering speeds of about 100fps.

3 Methodology

We initially decided to implement the method from [Kaneda et al. 1993] but changed course. It was unclear whether the cell size should be smaller than the average droplet or much larger. Handling the timing would also be a more complicated task, since each droplet's direction of movement must be recalculated whenever it enters a new grid cell.

We chose to implement Chen et al's height map method because the paper demonstrated good-looking results even with simplified particle behaviour [2013]. The timing is simpler and more well-defined, and collision detection is much simpler because of the construction of an ID map.

However, the Kaneda method has more accurate droplet physics that we chose not to implement on top of the height map method. It takes into account the different friction coefficients of wet and dry surfaces, as well as letting particles decelerate on a dry surface before stopping. The method we use calculates droplet acceleration as a property of mass, without considering the surface.

Even so, our method successfully models several water-droplet behaviours, which are both artistically interesting and physically plausible.

Static droplets are droplets whose mass is too small to overcome friction to slide down the surface. **Water trails** are left on the surface by droplets big enough to move. They are partly an optical effect, and partly due to adhesion forces between water and surface. Due to depositing some of their mass on the surface, moving droplets get smaller and stop when they are small enough. We assume that the surface is somewhat hydrophobic as well, causing the trails to bead up into small **residual droplets** over time. Due to gravity, the mass within a water trail moves downwards, causing the water trail to lose depth and width (**flattening and thinning**). Instead of drawing straight lines downwards, moving droplets exhibit **following behaviour** where they seek out and follow the same paths as existing water trails. They also **meander** due to small variations in the surface, and **merge** into larger droplets when they collide with each other.

- Generate
- Update moving particles
    - Generate residual particle
    - Get new position
    - Place particle
- Merge
- Place static particles
- Smooth
- Erode

Figure 1: *Main loop in simulation*

The main loop of our simulation is described in figure 1. We did not implement the rendering part of the pipeline described in [Chen et al. 2013], but we produce the same type of output, the height map, that Chen et al's simulation pipeline produces. The height map would be used to generate a normal map, which would then be used in cube-mapping.

Symbol	Value/Distribution	Meaning
b	4	subdivisions of timestep
f	$N(0.5, 1.5)$	affinity to water
$\mathbf{g}$	$-9.8 \times 10^3 \hat{\jmath}$ mm/s^2	gravitational acceleration
ℓ	user-defined	cell length
m_c	”	critical mass
m_{min}	$2\ell^3\rho$ mg	min initial mass
m_{max}	user-defined	max initial mass
M	”	grid columns
N	”	grid rows
n	”	maximum particles
Δt	”	timestep
T_e	0.01 mm	*erode* high-pass value
T_{id}	0.5 mm	ID threshold value
α	$U(0.1, 0.3)$	residual particle mass (relative to parent)
β	3	coefficient in residual probability function
θ	$U(-0.3, 0.3)$	meander angle
ρ	1 mg/mm^3	density of water
τ_{max}	0.1 s	max timestep to generate residual particle

Table 1: *Symbols used in this paper*

4 Data Structures and Algorithms

Two types of models are used: water droplets are particles, and the surface is a 2D grid with an angle of inclination, as in [Kaneda et al. 1993]. Because we render the results in a preliminary, non-realistic form, both the particle and grid data are stored in contiguous memory blocks.

4.1 Particles

Particles are responsible for storing the position, velocity, and mass $\{\mathbf{p}, \mathbf{v}, m\}$ of droplets. Position and velocity are stored as 2D Cartesian coordinates.

The mapping from particle coordinates $\langle x, y\rangle$ to grid cell is

$$i = \lfloor \frac{N\ell - y}{\ell} \rfloor \quad j = \lfloor \frac{x}{\ell} \rfloor \tag{1}$$

The friction between water and the surface is a constant. We set it indirectly by specifying a critical mass m_c, which is the maximum mass of a static particle. $\mathbf{F} = m_c\mathbf{g}$

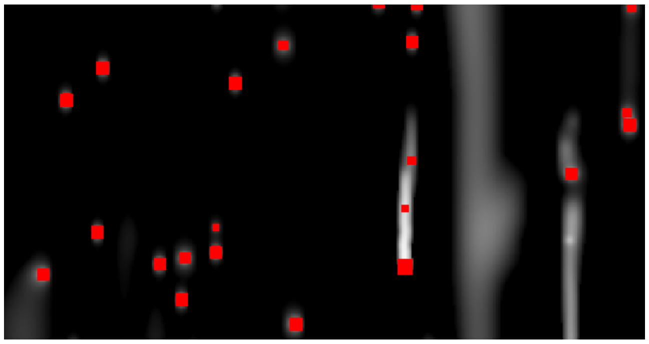


Figure 2: *Particles displayed with the height map*

With the parameters we chose, most particles generated are small, static particles. We flag them to skip calculating their movement.

Each cell of the grid is randomly assigned an affinity to water f_{ij}. Over the entire grid, this represents variations in the surface that lead to meandering. Each cell also has a height H_{ij}, which roughly approximates mass (we will discuss this height map later).

A *region* here is a 3×3 block of cells. We take the average height and affinity over regions centered on $(i + 3, j)$, $(i + 3, j - 2)$, and $(i + 3, j + 2)$.

The velocity direction $\hat{\mathbf{w}}$ is chosen from 3 discrete unit vectors in the directions of the 3 region centres listed above. We first try to choose the direction from the largest height of the three regions. (This creates the following behaviour (figure 3), e.g. if there is a preexisting water trail in one of the regions, the particle will head towards it and follow roughly the same path.)

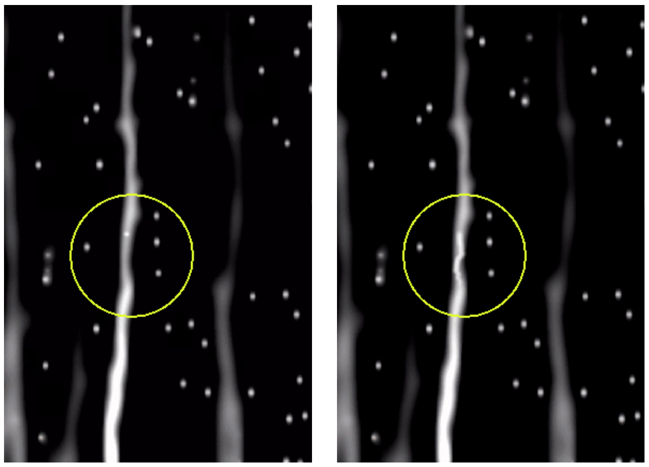


Figure 3: *Following behaviour*

If the heights are all 0, we compare the affinities. We prefer the south direction (i.e. aligned with gravity) if some of the comparisons are equal. To create the meandering behaviour, when the velocity direction is south, it is rotated by a random amount θ to the left or right.

The particle's new position and velocity are calculated (see equations 4, 5).

$$\mathbf{a} = \mathbf{g} - \frac{\mathbf{F}}{m} = \mathbf{g} - \frac{m_c\mathbf{g}}{m} \tag{2}$$
$$s = \|\mathbf{v} + \mathbf{a}\Delta t\| \tag{3}$$
$$\mathbf{v} = s\hat{\mathbf{w}} \tag{4}$$
$$\mathbf{p} = \mathbf{v}\Delta t \tag{5}$$

Then we check the new position to see if it has gone off the bounds of the surface. If it has, we mark the particle for deletion.

Because we chose to check for merges immediately after processing all the moving particles (figure 1), we realized we had to update the grid immediately after updating each particle instead of batching the static and moving particles together. We will discuss how the grid is updated in the next section, but in short, the particle's new position and radius are used to draw a hemisphere on the grid.

Unless the timestep Δt is very small, particles can appear to jump from position to position instead of leaving a continuous trail. To fix this, Chen et al. use a *line rasterization* algorithm to linearly interpolate the old and new positions, but we went with the cruder solution of placing hemispheres at intervals of 1.5ℓ in the direction of the new position.

With this fix, the size of the timestep presents another problem: visible zigzags. To mitigate this, we subdivide the timestep by b and recalculate the position and velocity for each subdivision.

Residual droplets are added according to a probability function of the time since the parent particle last spawned a residual particle [Chen et al. 2013].

$$\mathbf{pr}(\tau, \Delta t) = \min(1, \beta \frac{\Delta t}{\tau_{max}} \min(1, \frac{\tau}{\tau_{max}})) \tag{6}$$

The residual particle's mass is set to $\min(m_c, \alpha m)$ and its position is the same as the parent's. The parent loses mass in order for mass to be conserved.

4.2 Height Map

We have used the term *grid* to refer to the aggregate data structure that represents the surface that the particles interact with.

$\mathbf{c}_{ij}$	position of centre
f_{ij}	affinity
H_{ij}	height
ID_{ij}	particle ID

The grid represents a planar surface. We fixed the angle of inclination at 90 degrees because of a lack of experimental data for other values.

The grid size is $N \times M$ with square cells of side-length ℓ.

The height map discretizes the volumes of water on the surface.

Placing particles on the height map: each particle adds a hemisphere with centre $\mathbf{p}$ to the height map. We select a square block of cells of radius $\lceil \frac{r}{\ell} \rceil$ and use the Pythagorean theorem to get the height of the hemisphere at the centre $\mathbf{c}_{ij}$ of each cell.

$$r = \sqrt[3]{\frac{3m}{2\pi\rho}} \tag{7}$$

$$\mathbf{c}_{ij} = \langle \ell j + \frac{\ell}{2}, N\ell - (\ell i + \frac{\ell}{2}) \rangle \tag{8}$$

$$(H_{ij})^2 = r^2 - \|\mathbf{c}_{ij} - \mathbf{p}\|^2 \tag{9}$$

After all of the particle operations, we perform **smoothing**, which represents the flattening of water over time due to adhesion. The smoothing operation is implemented as a 3×3 *box blur* over the entire height map, followed by a *high-pass filter* with threshold T_e. Figure 4 shows a water trail being reduced in depth and width after several iterations.

The **erosion** operation represents thinning of a water trail as its mass moves downwards. We implemented this using a simplified form of horizontal erosion.

Unlike the particle system, the height map does not conserve mass. Placing additional hemispheres along the path of a moving particle results in the net mass of the height map being greater than that of the particle system. However, over time, the smoothing and erosion operations reduce the mass of the height map. But also a hemisphere for each particle's current position is added back to the height map on each iteration, so the height map converges over time. We can think of each particle as "anchoring" water mass to the grid. If the particle data are deleted, the smoothed regions of water shrink and eventually vanish.

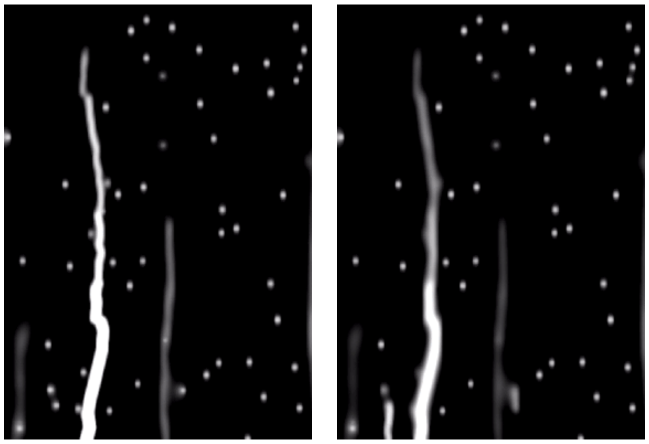


Figure 4: *Result of smoothing and erosion after >20 iterations*

4.3 ID Map

In order to detect when particles need to merge, we add an ID map with the same dimensions and resolution as the height map. Each cell on the ID map is coloured with the ID of the particle that "owns" the water on that cell. If there is no water, the value 0 is assigned.

The ID map is derived from the height map (see figure 5). Whenever a hemisphere is placed on the height map, the corresponding cells of the ID map are updated with the particle ID. However, because the smooth and erode operations on the height map only take the height map as input, some experimentation is needed to make the ID map approximately match the height map. That is, if there is a water region expands on the height map, the ID region needs to expand too. Our collision detection primarily depends on the ID map, so the merging behaviour is only as good as the ID map.

To smooth the ID map, we tried to find an operation that met the above requirement yet was not too expensive. We switched from calculating the local maximum of a 3×3 region to a simplified 2D *dilation* followed by a threshold function with $T = 0.5\text{mm}$. Ero-

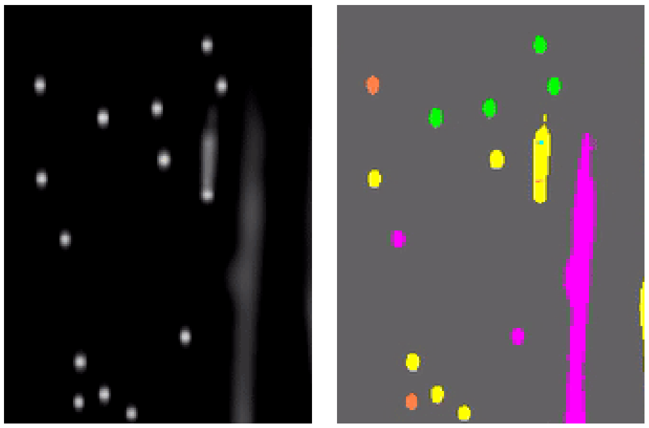


Figure 5: *Height map and corresponding ID map*

sion is simply performed at the same time as the height map.

We detect collisions by scanning the ID map and comparing each cell with its 4-way neighbours. If two adjacent cells have different, non-null IDs, a merge takes place.

If both IDs belong to particles that still exist, we have enough information to construct a new particle to replace the merging particles. The new position is set to the position of the lower particle, and the new mass and velocity is calculated from the combined mass and momentum of the merging particles. We construct a particle lookup table, e.g. `map<ID, Particle>`, for this purpose.

We also construct a parent-child table, e.g. `multimap<ID, ID>`, to check that we are not trying to merge a residual particle back into the larger parent particle.

There are other cases: only one of the IDs is still an active particle; neither ID belongs to an active particle. For example, a static particle was added to the grid, overlapping a water trail whose source particle has already travelled off the surface. We tried different approaches for each case. Our aim was to eliminate the static "clusters" of particles on the height map that were smoothed into unnatural-looking static regions of water. In the end, we decided to count both cases as a merge, and recolour the ID map with the same ID without changing the particle properties (if existing). This resulted in large regions of water moving towards the bottom of the grid and being eroded at the top, which looks more plausible than the static regions (compare the results without merging and with merging, figures 7d, 7e).

Apart from updating the particle data structures and lookup tables, we do a 4-way *flood fill* on the ID map to replace the merging IDs. For example, in the 2-particle case, we replace both component IDs with the new ID in a 4-way contiguous area (figure 6).

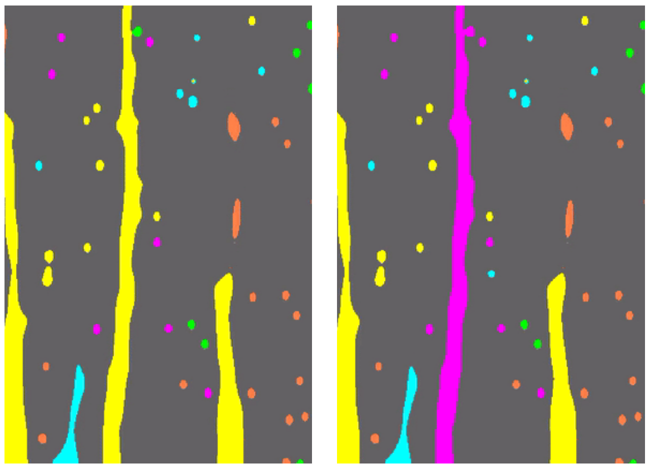


Figure 6: *ID map, before and after merging*

5 Results

For educational purposes, we render the three main data structures (particles, height map, and ID map) to the screen. We also add keybindings to the program to enable or disable various parts of the simulation (Table 2).

Our implementation runs on the CPU. We developed and tested on an Intel Core i5-5200U at 2.20GHz with 6GB RAM. The simulation runs well at 30fps (33.3ms/frame). When we stopped constraining the FPS, we observed average timings of 14.8-31.8ms per frame.

Table 2: *Keybindings*

E	erosion on/off
M	merging on/off
R	residual particles on/off
S	smoothing on/off
I	switch between height map and ID map
P	show/hide particles
X	delete all particles

Table 3: *Example parameters*

N	M	ℓ	n
1000	600	0.5mm	700

erode	smooth	residual	merge
on	on	on	on

5.1 Limitations

With the current method of placing hemispheres, the size and appearance of the droplets depends on the grid resolution. If the cells are too big, any particles smaller than a cell will be poorly represented on the grid. This increases the grid's memory requirements, as well as the computational expense (several operations are done over the entire grid) of each iteration.

One behaviour we've observed with the current model is that merged particles appear to instantaneously change position and size. Sometimes it is apparent that the droplet has merged with a region and contributed mass to the particle at the bottom of it, but sometimes it is not clear and the droplet seems to have jumped from one position to the other. Perhaps we could interpolate the positions, but because of the irregularity of regions on the ID map, the path is not guaranteed to be linear.

The merging algorithm also sometimes causes residual particles to "disappear" (e.g. by merging into its parent water trail after the parent particle has exited the map). We need to change (1) the ID map smoothing to give more useful regions and (2) the merging algorithm, perhaps to take the mass of a contiguous region into account.

6 Conclusion and Future Work

Our method for simulating the movement of droplets on a flat surface combines a particle system with a height map and an ID map. It is the height map rather than the particle system that will be used to render the droplets. After the particles are updated, we apply simple image-processing operations to the height map and ID map, allowing them to continuously change over time. For example, raindrops running down the surface leave water trails that shrink and fade over time.

To be useful for video games where driving is part of the gameplay, we need to expand the model from a flat 90-degree surface to angled and curved surfaces. One way of modelling a curved surface with the grid is described in [Kaneda et al. 1999]. We would also need more data on the forces acting on the water drops (friction, surface tension, drag, etc.)

We chose not to implement the irregular droplet shapes described in [Chen et al. 2013], focusing on the behaviour instead. We could add this feature and also look into smoothing methods to achieve physically-accurate shapes.

We have achieved real-time performance on the CPU, but implementing our method on the GPU would allow us to decrease the timestep size and add more sophisticated behaviour. For example, we could use OpenGL4 compute shaders. The grid (height map and ID map) is already packed rectangular data and the smooth and erode operations are easy to implement. The particle data structures would need more rewriting because we currently rely on hashmaps and pointers. If we can guarantee that IDs of new particles are monotonically increasing, we could replace the lookup tables with a binary search.

References

CHEN, K.-C., CHEN, P.-S., AND WONG, S.-K. 2013. A heuristic approach to the simulation of water drops and flows on glass panes. *Computers & Graphics 37*, 8, 963–973.

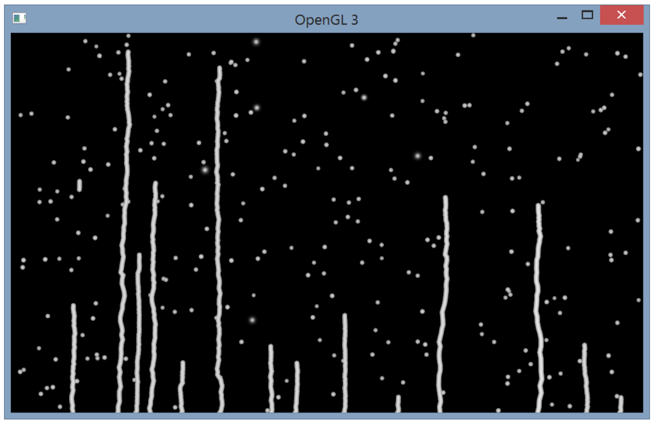
KANEDA, K., KAGAWA, T., AND YAMASHITA, H. 1993. Animation of water droplets on a glass plate. In *Models and Techniques in Computer Animation*. Springer, 177–189.

KANEDA, K., IKEDA, S., AND YAMASHITA, H. 1999. Animation of water droplets moving down a surface. *The Journal of Visualization and Computer Animation 10*, 1, 15–26.

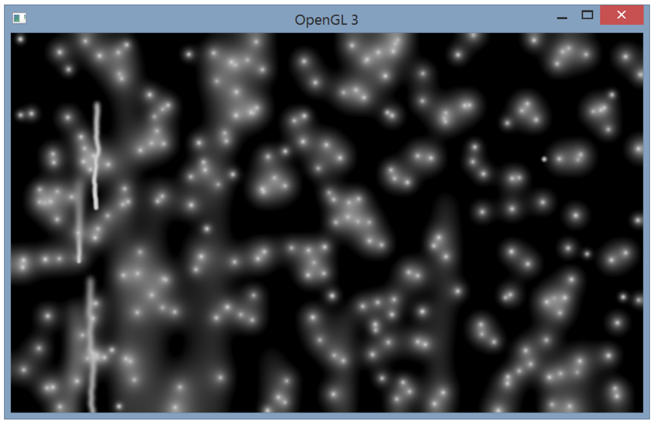
NAKATA, N., KAKIMOTO, M., AND NISHITA, T. 2012. Animation of water droplets on a hydrophobic windshield. In *WSCG 2012 Conference Proceedings*, WSCG, 95–103.

SATO, T., DOBASHI, Y., AND YAMAMOTO, T. 2002. A method for real-time rendering of water droplets taking into account interactive depth of field effects. In *Entertainment Computing: Technologies and Applications, IFIP First International Workshop on Entertainment Computing (IWEC 2002)*, vol. 240, IFIP.

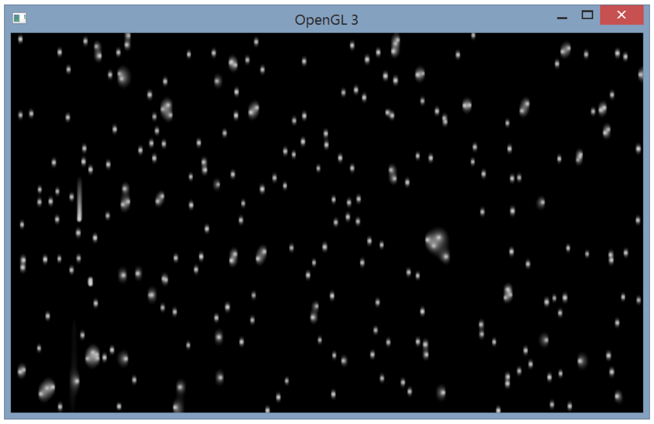
TATARCHUK, N., AND ISIDORO, J. 2006. Artist-directable real-time rain rendering in city environments. In *Proceedings of the Second Eurographics conference on Natural Phenomena*, Eurographics Association, 61–73.



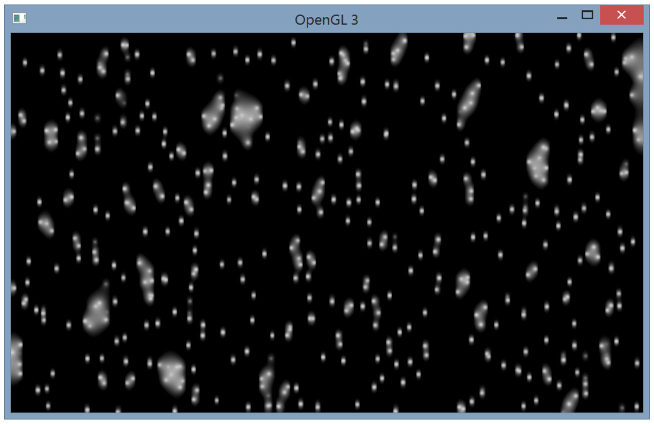
(a) *All operations disabled*



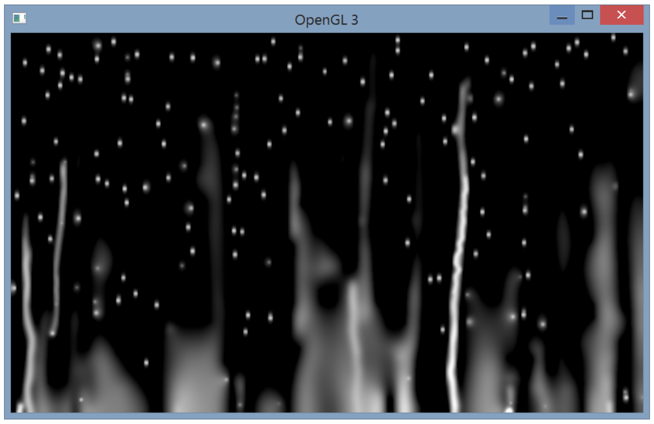
(b) *Smoothing enabled*



(c) *Erosion added*



(d) *Residual droplets added*



(e) *Merging added*

Figure 7: *Effects of running with various operations enabled*