Computers & Graphics 112 (2023) 40–49

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Deep weathering effects☆

Adrien Verhulst [a,b], Jean-Marie Normand [a], Guillaume Moreau [a,c], Gustavo Patow [d,*]

[a] Nantes Université, École Centrale Nantes, CNRS, AAU-CRENAU, UMR 1563, Nantes, France
[b] Sony Computer Science Laboratories, Inc., Japan
[c] IMT Atlantique - Lab-STICC UMR 6285, Brest, France
[d] ViRVIG-UdG, Universitat de Girona, 17003 Girona, Spain

ARTICLE INFO

Article history:
Received 17 December 2022
Received in revised form 10 March 2023
Accepted 25 March 2023
Available online 28 March 2023

Keywords:
Weathering
Rust
Bricks
Iron

ABSTRACT

Weathering phenomena are ubiquitous in urban environments, where it is easy to observe severely degraded old buildings as a result of water penetration. Despite being an important part of any realistic city, this kind of phenomenon has received little attention from the Computer Graphics community compared to stains resulting from biological or flow effects on the building exteriors. In this paper, we present physically-inspired deep weathering effects, where the penetration of humidity (i.e., water particles) and its interaction with a building's internal structural elements result in large, visible degradation effects. Our implementation is based on a particle-based propagation model for humidity propagation, coupled with a spring-based interaction simulation that allows chemical interactions, like the formation of rust, to deform and destroy a building's inner structure. To illustrate our methodology, we show a collection of deep degradation effects applied to urban models involving the creation of rust or of ice within walls.

1. Introduction

Weathering effects have received a lot of attention from the Computer Graphics community [1]. Several key techniques have allowed the production of urban landscape images with very realistic degradation effects, including erosion, pollution, flow, peeling, and cracking of the building's surfaces [2,3].

However, in spite of the many achievements of the past decades, physically-based simulations of deep weathering effects (i.e., the ones involving not only the outer surfaces of buildings but also the inner layers of the walls) have not been extensively tackled by the community. In particular, the simulation of the interplay between rust, plaster, brick, and other masonry structural elements, despite being a major factor in the degradation of older buildings [4], has not been addressed.

In this paper, we present a technique that simulates deep weathering effects on building materials (see Figs. 1 and 2) leading to the exposure of buildings' internal structures such as bricks or rusted surfaces.

Our method approximately simulates the penetration of water particles in a wall volume, their interaction with the inner elements present in the wall, (like concrete, iron, or other metals), and the consequent formation of new compounds, like rust; and similarly for the humidity that remains and is converted later on into ice (e.g., in winter). Since these new compounds have a higher volume than their original counterparts, their expansion leads to the formation of cracks in the wall which result in the exposure of bricks or rusted surfaces. This phenomenon is ubiquitous in any city with old buildings as illustrated in the real pictures from Fig. 4.

Our main contributions are:

1. A physically-inspired simulation of the penetration of water particles (e.g., humidity, rain) in the bulk volume of walls, and their interactions with the inner structures of the wall (e.g., concrete, iron lattices, lead pipes, etc.).
2. An approximate simulation of the expansion of these elements, like water turning into ice, or metal into rust.
3. An approximate simulation of the deformation of concrete, plaster, and bricks in the wall, and of the formation of cracks inside the wall. Cracks can lead to the breaking of concrete, the fall of loose plaster, as well as the acceleration of the two previous steps given the increased exposure of plaster, bricks, and metallic structures to air as well as other natural weathering elements (e.g., rain).

In the remainder of this article we start with an overview of the related work in Section 2 before presenting our proposal (Section 3). Section 4 details our simulation framework while Section 5 illustrates deep weathering effects that can be achieved by our method and discuss its limitations. Finally, we draw conclusions and suggest future work in Section 6.

☆ This article was recommended for publication by J. Dorsey.

* Corresponding author.
E-mail address: gustavo.patow@udg.edu (G. Patow).

https://doi.org/10.1016/j.cag.2023.03.006

Fig. 1. A building with deep weathering effects. Left: the original building. Middle: the same building after the simulation of water particles penetrating the gray area. Right, a closeup view of the resulting degradation is due to weathering effects. Surface noise is part of the wall texture itself.

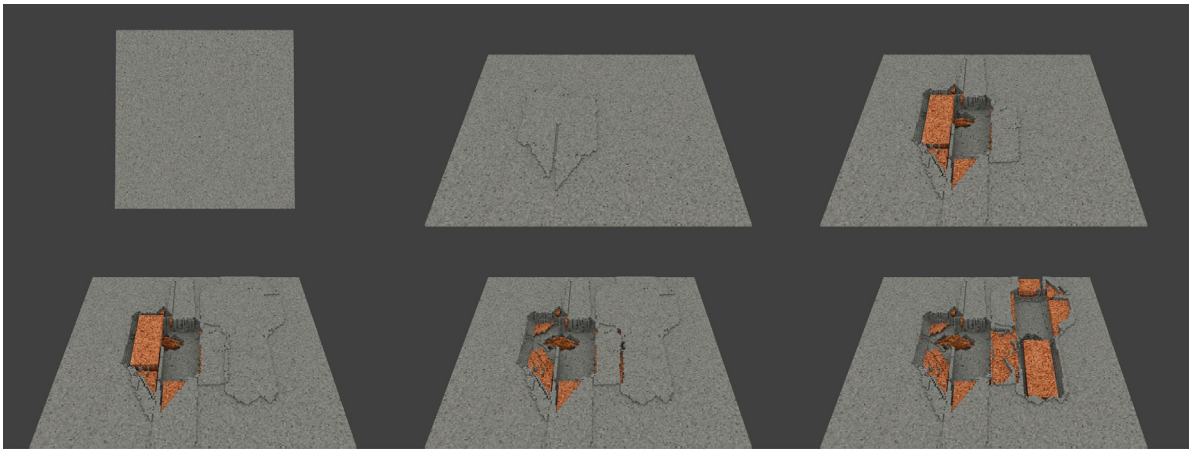

Fig. 2. From left to right, top to bottom, part of a wall being pushed over by rust over time. Surface noise is part of the wall texture itself.

2. Previous work

Aging phenomena have attracted a great deal of interest in the Computer Graphics community. Covered topics include the deposition of dust on objects [5], terrain erosion [6], wrinkles generation on organic materials [7] (which was later improved with specific crack generation algorithms e.g. [3]), material peeling [8], surface erosion [9], tarnishing effects [10], flow and its impact on appearance changes [11], metallic patinas [12,13], or destructive corrosion [14]. More material-oriented studies were also proposed, especially on material dissolution [15], material fluorescence [16], material decay [17], or organic material growth [18]. The main difference between our approach and previous works [15,17] is the objective of our simulations: while previous works focused on *appearance* changes due to weathering, our objective is to model *mechanical* strong deformations caused by weathering processes, such as rust and its deformation effect on plaster and bricks.

In the past, there has been a lot of very interesting research on the generation of cracks on surfaces, starting from the works by Hirota and coworkers [19,20], dynamic animations [21,22], including the work of Gobron and Chiba [23] using cellular automata, the works using finite element analysis [24], the work by Bosch et al. [25] on the generation of scratches and impacts, or the more recent works by Iben and O'Brien [26], and Muller [27], up to some fast approximations for brittle fracture simulation, as in the work by Hann and Wojtan [28]. In all these cases, these works are more concerned with the realistic generation of cracks than with the simulation of their underlying process, as we do in this paper for the building elements interacting with water. We refer the interested reader to recent surveys like the one by Muguercia et al. [3] for an in-depth treatment of the subject. Again, these thorough works do not attempt to model weathering effects, only the result of tears in a model surface.

Particle tracing has also been used: γ-ton is a technique developed by Chen et al. [29] for visually simulating weathering. This was later improved by Kider [30] with a system of particles that allowed to simulate a 3D model's shape and appearance aging by a number of phenomena, including physical, chemical, biological, environmental, and weathering effects. Although our work shares the particle-based approach with these works, our aim is not to simulate decay or degradation, but *mechanical* deformations on the inner elements of an architectural structure.

Dorsey and colleagues [15] studied numerous stone weathering behaviors, employing complex modeling of chemical reactions. Later, Mérillou et al. [17] proposed a simple model for simulating the aging of building materials, being able to handle a variety of damage patterns related to salt decay, as well as locating the phenomena with a physically inspired method that leads to plausible results. Our proposed technique is also related to the work of Cutler et al. [31], where a procedural approach to solid model authoring was presented, based on a volumetric approach. However, our implementation is based on voxels, while theirs was based on tetrahedrons and a distance function. Recently, Ishitobi et al. [32] developed a method for weathering simulations of coated metallic objects, with a particular focus on the processes of cracking and peeling. An introduction to these topics can be found in the seminal book by Dorsey et al. [1] or in the survey by Mérillou and Ghazanfarpour [2]. As mentioned, the main difference with our approach is on the scale and volume of the effects simulated, as we aim to model the macroscopic mechanical deformation weathering may produce on architectural elements.

3. Overview of our proposal

Buildings may suffer from structural defects at different levels (roof, walls, foundations, etc.) and with various causes and degrees of importance. Structural dampness is one cause of structural defects which is due to the penetration of moisture within a building's structure. A high proportion of those damp problems

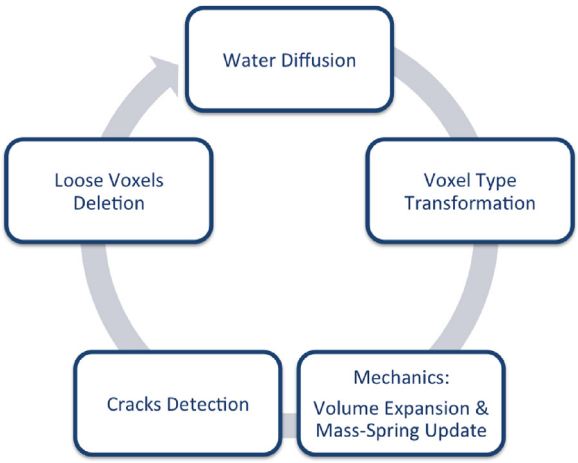

Fig. 3. Overview of the simulation loop.

are caused by rain penetration through porous masonry. In this paper, we focus on rain penetration in old brick walls.

Our system simulates deep weathering effects and is based on two stages: (i) a physical-inspired simulation of the interactions of water particles with buildings' materials, and (ii) computation of the deformation of structural elements due to internal forces arising from weathering effects (computed from a voxelization of the simulation space). Water particles are the main simulation entities, penetrating the wall and interacting with its inner elements (e.g., bricks, concrete, plaster, beams, wood). For instance, corrosion of reinforcing bars is a major cause of reinforced concrete failure because of rust swelling. Here, voxels are merely used as a data structure to store the accumulated water, which in turn will be used to compute the strength of the materials created by chemical reactions between water and the inner structure's materials. To allow for an efficient simulation, a wall (e.g., brick masonry) and its inner structural elements (e.g., metallic structures) are voxelized. We approximate the contents of each voxel as being of a unique type (i.e., material, plaster, iron, etc.), and each voxel is connected to its surrounding voxels via a net of springs.

The simulation loop (see Fig. 3) is as follows:

1. Water particles are instantiated on the wall surfaces, where they diffuse into the wall material. Although currently not implemented, at this point particles could flow on the wall exterior surface, further wearing its appearance and reaching other areas [11,33]. Then, their positions are updated by the particle system, diffusing into materials. When a water particle enters a non-empty voxel, its velocity is updated (depending on the voxel's water permeability) and the voxel's water content is updated (depending on the particle's velocity).
2. Whenever they interact with water and depending on their types, the voxels can turn into another type of voxel (e.g. iron turns into rust due to corrosion or water into ice).
3. Since the transformation may require a larger volume than its original counterpart (e.g. rust takes 4 to 12 times more space [4] than iron), mechanical forces are exerted, leading to the displacement of the neighboring voxels and to the generation of new voxels.
4. The mass–spring system is updated.
5. Each spring having a length above its maximum length is broken (denoted as s_b) and a crack is instantiated.
6. Cracks close enough to each other are merged.

Fig. 4. Two real examples of weathering effects of water and rust on a building facade.

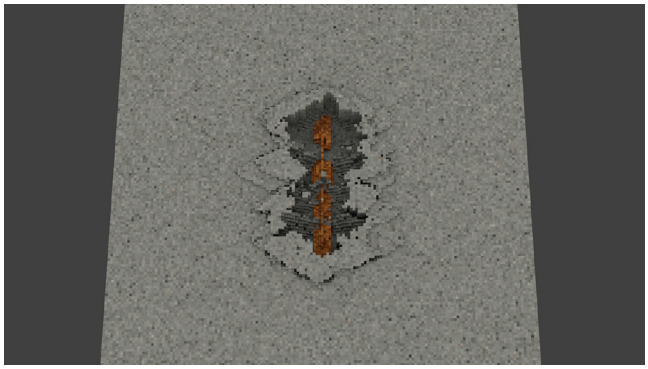

Fig. 5. Simulation results of weathering effects on a building facade. Notice the now rusty iron pipe.

7. Each voxel with a broken spring s_b checks if it still has a connection with some other voxel. If it does, the spring remains; if not, it is flagged as "loose".
8. Each rust voxel (noted v_{r1}) having exerted a force, checks if it has enough space to instantiate a new rust voxel (noted v_{r2}); if yes, v_{r1} loses energy, and v_{r2} is instantiated (with the same energy as v_{r1}) and linked to its surrounding voxels.
9. if a set of voxels, e.g., belonging to a brick or a broken piece of concrete, is found not to be connected to the rest of the wall anymore, and if these voxels have clear access to the exterior of the wall, then they fall. In our current implementation, they are just removed, but it is not difficult to imagine an animated loose brick falling off a damaged wall.

At the end of this process, the inner materials of a wall might be exposed. If the process continues, in the case of rust, the degradation process will accelerate because of the direct interaction of rust with air and rain. These phenomena can clearly be seen in older, poorly maintained buildings, resulting in a characteristic degraded appearance, as seen in Fig. 4.

In this paper, we focus on the rain penetration phenomenon and on two types of representative water–material interaction: (i) the creation of rust from iron and (ii) the creation of ice from water (at low temperatures). Both of them can lead to the creation of cracks and can result in deep weathering effects, see Fig. 5.

4. The approximate simulation model

In this section, we present more details regarding our simulation model. First, we focus on voxels creation and on the mass–spring system we use to model the forces. Then we detail how deep weathering effects are simulated.

4.1. Voxelization

In order to simulate deep weathering effects, we need building models to be segmented into their distinct elements, depending on their materials (plaster, bricks, iron, etc.). This could prove challenging if one wants to apply our technique to existing, artist-created 3D models without internal structure. Nevertheless, the recent development of Building Information Modeling (BIM) as well as the procedural shift within the Architecture, Engineering, Construction, and Operations (AECO) industry will greatly simplify the availability of such semantic models. In our current implementation, the designer selects an area on a building wall, and automatically the system generates a generic multi-layer structure for that area, adding further layers of plaster, bricks, and random vertical or horizontal metallic pipes within the structure.

After the segmented building model is obtained, the next step in our process is the voxelization of the wall volume to be simulated. As usual, there is a trade-off between the voxels' size, the volume of the simulation domain, and the number of voxels.

We settle for enough voxel resolution to capture the important structural details of the model in order to provide visually plausible results. The simulation domain was restricted to specific parts of 3D building models. Those models were not voxelized but directly imported and rendered (cf. Fig. 1), and only a part of their walls were voxelized.

For each voxel, we store the following information: its position, the springs connecting it to its neighbors, the force going through it (cf. Section 4.2), the amount of moisture on each of its faces, and its type. Our voxels can be of the following types: brick, plaster, iron, rust, space (empty voxels that can be filled with ice or rust), ice, and "fixed". In addition to the aforementioned attributes, voxels producing other voxels (e.g. iron, rust, ice) possess an "energy" attribute, to represent whether a voxel has enough *chemical energy* to produce other voxels.

Note that "fixed" voxels are a special type of voxels used to represent the interface between our simulation space (composed of voxels) and the rest of the larger model which is composed of vertices and triangles (cf. Fig. 1). As a consequence, those fixed voxels cannot be displaced and indicate a junction with the main structural elements of the building.

During the simulation, if a voxel is not linked (either directly or indirectly through other voxels) to at least one fixed voxel, then it becomes "loose" (it is part of a "falling" set of voxels), meaning that it does not interact with the other voxels but instead is affected by external forces (i.e., gravity). In this case, the loose element can be animated during its fall, or simply removed if a detailed animation over time is unimportant for the final result.

4.2. Forces

In our system, we model contact forces using a mass–spring system, where the object is approximated by a finite set of masses represented by the aforementioned voxels. Mass–spring models are one of the simplest yet most flexible ways to model a deformable body [3].

Basically, mass–spring systems quantify the simulation volume into a finite set of particles $\{p_i; 1 \leq i \leq n\}$. Each particle p_i (a voxel in our system) has its own mass m_i and position $\mathbf{r}_i$. The particles are pairwise connected with springs, each with its own properties (stiffness, damping factor, and rest length). Each particle is set to a classic equilibrium equation:

$$m_i \frac{\partial^2 \mathbf{r}_i}{\partial t^2} = f_i \tag{1}$$

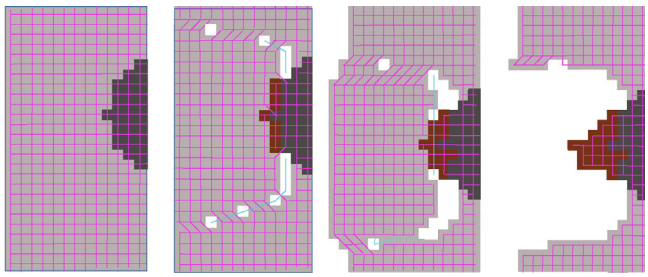

Fig. 6. A schematic 2D example of a rusting pipe inside a wall. Forces are represented as strings (purple lines) that propagate stress among the voxels. Cracks lines are shown (in cyan), as well as iron voxels (in dark gray), wall material voxels (plaster or concrete, in light gray), and rust voxels (in dark red). From left to right: the pipe in its initial state inside the wall material. As rust appears to occupy a larger volume, material voxels are pushed over and fall off.

where f_i is the sum of all the forces acting on particle p_i [3]. We distinguish between external forces, like gravity; and internal ones, which come from the springs attached to the particle p_i.

In general, springs follow Hooke's law [34], which can be stated as:

$$f_i = k \frac{(|\Delta \mathbf{r}_{ij}| - r_{ij}^0)\Delta \mathbf{r}_{ij}}{\|\Delta \mathbf{r}_{ij}\|} \tag{2}$$

with k being the spring constant that characterizes its stiffness, r_{ij}^0 its original or rest length, and $\Delta \mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ its current length, measured as the difference between the positions of voxels i and j. In practice, we use an effective force f_i' which is defined as

$$f_i' = t * f_i \tag{3}$$

where t is a force-damping coefficient that depends on the type of voxel connection. Typically, a connection between two brick voxels will have $t \approx 1$, in order to allow the brick voxels to move together, if possible. See Section 4.3.3 for exact values for t in each use case. This choice is justified from a physical standpoint since materials react differently to pressure. Moreover, forces are only transmitted from voxel to voxel when they are above a small threshold, in order to limit the calculation to the surrounding voxels.

When the limit of a spring exceeds a given threshold, we model breakage by flagging the corresponding spring as broken, and by removing the corresponding connections between the voxels. See Fig. 6 for an example using rust as expanding material. In practice, when many springs are broken in the same region, cracks appear, and eventually, concrete gets broken or plaster gets loose, and can fall, see Section 4.3.4 for implementation details on this effect.

4.3. Deep weathering effects

In order to simulate deep weathering effects, we use particles to represent water and use them to simulate the way water penetrates and interacts with a building's inner materials. However, depending on the exact materials in the building and the length of the whole simulation, the particles can have different behaviors, from a more plastic one similar to plaster (but with somewhat more rigidity, if needed), to a full rigid body. In the first case, the particles are treated exactly as plaster (see below), simply taking into account their extra rigidity. On the other hand, when they act as full rigid bodies, we have to resort to a different approach to guarantee rigidity, see below. Finally, we provide details of how cracks are generated and how they spread within our model.

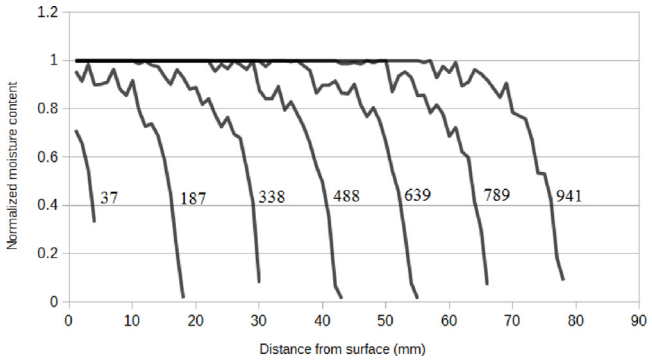

Fig. 7. Curves representing the *Volume Fractional Saturation* (VFS) obtained with our particle-based simulation at different time steps for a wall made of concrete. The moisture advance depends on the material's water saturation.

4.3.1. Water particles

As mentioned, humidity is an important source of structural defects. Most of the inorganic materials used in construction are porous. This means they absorb water particles according to their exposure to environmental factors like rain, humidity, groundwater, condensation, etc., and they release them according to the drying properties of the atmosphere. The flow of water on and within a building's materials depends on the water content of these materials, namely whether they are saturated or not. Saturated flow can be explained through Darcy's law, but for the unsaturated case an extended version of the same equation should be used [35,36] (see Eq. (4)):

$$\frac{\partial \theta}{\partial t} = \nabla D \nabla \theta \tag{4}$$

where θ is the ratio of liquid volume to bulk volume (called Volume Fractional Saturation), and D is the hydraulic diffusivity, which is generally a function of θ. This equation can be analytically solved in 1D for very simple cases, resulting in a *sigmoid*-like the function of $\theta(x)$ (where x is the spatial dimension), which "slope" decreases as water penetrates into the material volume.

To simulate the penetration of water into buildings' structures, we decided to use a Monte Carlo-based approach, where water particles are "sprayed" onto the building's voxels in contact with weathering elements. These particles penetrate the voxels of the wall volume according to their *water permeability*, which represents the easiness for a water particle to go through this particular material (see Fig. 7 for concrete).

For each water particle, we compute its instantaneous water content (p_w) based on three parameters: the surface of our simulation area (a); the rain intensity (i in liters per hour on a unit of surface area), and the number of particles generated per second (nps):

$$p_w(t) = \frac{(a * i)}{(nps(t) * 3600)} \tag{5}$$

Thus, units for p_w are liters per second.

The particle system is responsible for setting a direction and a velocity for each water particle it generates.

According to their directions and velocities, water particles navigate through the simulation space to saturate the voxels of water. Whenever a water particle goes through a voxel, we use the Russian Roulette technique to decide whether the particle is "*killed*" or whether it continues to penetrate the building.

A killed particle, adds its water content to the voxel's water saturation level. As a consequence, the behavior of Eq. (4) can be approximated with an extinction probability having the form of a sigmoid function, see Eq. (6):

$$Pr = 1/(1 + e^{k*-\theta}) \tag{6}$$

Fig. 8. From left to right: rust presses against the voxels above and around it; When the above voxels reach a threshold, they are displaced; A new rust voxel is instantiated at the vacant position.

where k depends on the voxel's type and where θ is the current ratio of water saturation in the voxel. Fig. 7 presents implementation results of Eq. (6) obtained with our particle-based simulation.

Each voxel can store a given amount of water before being saturated. The more water it contains, the easier it is for the next water particle to go through, see Eq. (4). Moreover, only saturated voxels may give rise to chemical reactions favoring deep weathering effects (e.g. the formation of rust from saturated iron voxels), we detail these reactions in the next Section.

4.3.2. Water interaction: material transform

As mentioned above, saturated voxels may give rise to chemical reactions involved in the deep weathering effects we can simulate, namely:

- saturated iron voxels may produce rust voxels;
- saturated space voxels may produce ice voxels;
- saturated rust (resp. ice) voxels may produce more rust (resp. ice) voxels.

Since rust (resp. ice) has a larger volume (see [4]) than the iron (resp. empty space) it replaces, its chemical creation applies a force on the mass–springs of the voxels in its vicinity. Whenever the tension is above a threshold, voxels are moved in order to make space for the newly created "rust" (resp. "ice") voxels, see Fig. 8.

4.3.3. Bricks

As our domain is represented by a grid of voxels (see Section 4.1), we simulate all these effects at the voxel level. In the case of plastic deformation, bricks are simply dealt with by changing the spring parameters to make them stiffer. However, when brick rigidity is above a given user-provided threshold, we switch to a full rigid body behavior to guarantee perfect rigidity. In this scenario, "brick" voxels do not move independently from the other voxels representing the brick.

In order to do so, the springs between brick voxels transmit their forces to each other with very little loss (a transmission of 0.99 cf. Eq. (3)) to guarantee that a similar amount of force is exerted on them. Therefore when one brick voxel has enough force to move, all the other connected brick voxels are more likely to move at the same time. On the other hand, interfaces between materials (e.g. brick/plaster or plaster/brick), transmit force with a heavy loss (a transmission of 0.1), to ensure that the rigid object, when moving, does it independently from the other material.

While our focus remains on bricks, other very rigid objects (as opposed to plaster), could be simulated in a similar fashion.

4.3.4. Crack generation

Cracks, in our system, are the result of broken springs, i.e., springs that stretch beyond their maximum allowed distances. Whenever a spring breaks, we (i) remove the connection between the voxels it used to connect; (ii) instantiate one *crack-point* at each end of the spring (cf. Section 4.2). Observe that, in our

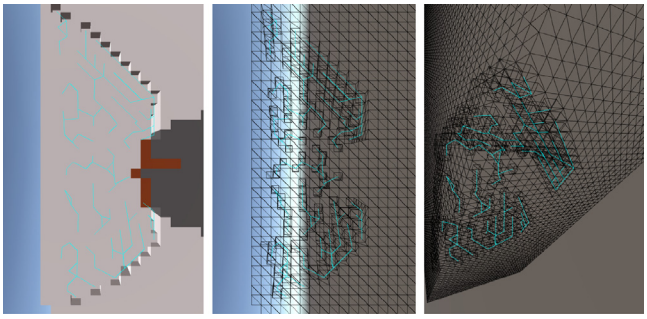

Fig. 9. 3D view of a cracked wall, where the voxels were displaced in a similar way as in Fig. 6. The cracks are represented with cyan lines. Whenever possible, we join and merge the cracks. From left to right: lateral "voxel view" of a wall segment, cracks inside the material with voxel silhouettes, and 3D view of the cracks inside the material where we can see them gathering together.

implementation, crack points have fixed positions in space. It should be noted that crack points and cracks are not rendered in the final result of our simulation.

Cracks are computed at each iteration of the simulation, depending on the crack points they contain. The process of crack generation is the following:

1. Remove all crack points already assigned to any crack.
2. Check each unassigned crack-point whether it is inside a voxel (i.e., voxels may have been previously removed if they belong to fallen material from the wall). If not, the crack point is removed.
3. For each crack-point cp_{cur}:
   - 3.1 cp_{cur} is added to an existing crack $crack_{cur}$. if cp_{cur} is not close enough to an existing crack, create a new crack $crack_{cur}$ at its position.
   - 3.2 All crack-points cp_{other} within a radius r of cp_{cur} are added to $crack_{cur}$.
   - 3.3 Each spring crossing a line between cp_{cur} and any cp_{other} is broken. At this point, all crack-points around cp_{cur} have been added to $crack_{cur}$. Now, we check if one of those crack points could also be assigned to another crack c.
4. For each crack c; if at least one of its crack-points is also part of $crack_{cur}$, then every crack-point of c is added to $crack_{cur}$ and c is removed.

This process of crack generation and merging ensures that we track every crack point and that cracks in the vicinity are merged into larger cracks. Springs that connect cracks are removed and may accelerate the deterioration process as is the case in real life.

Cracks are typically not supposed to be rendered since they represent ruptures between the materials of our walls. For illustration purposes, we use lines to visualize them in Fig. 9. Breaks and fall of materials will eventually happen by themselves along with the progress of the simulation, see Fig. 10.

5. Results and discussion

In this section, we first present some of our deep weathering effects results before discussing limitations.

As mentioned, when the user selects a building part, the system automatically generated the wall volume for that area, randomly adding horizontal or vertical pipelines. The simulation produces a voxelized output, which can have a block-based appearance (3D aliasing). Just as in any other of these cases, the output voxels can be post-processed to be refined and smoothed (e.g., with a marching cube algorithm [37]) to improve the result.

Fig. 10. Even without the rendering of cracks, we can clearly see where the fractures are going to happen. Noise is part of the wall texture.

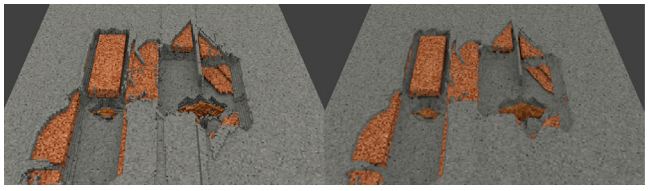

Fig. 11. Left: The original 3D mesh. Right: A heavily smoothed version. This Figure is included for illustrative purposes, as this smoothing operator is not part of the core proposal of this paper.

The results were obtained from a standard consumer-grade laptop (Intel Core I5@2.4 GHz processor; 8 GB of DDR3 RAM; GPU: NVIDIA GeForce 840M).

5.1. Results

Our particle system, voxel engine, and simulation were implemented in Unity3D V5.4.0. Since our simulation produces a voxelized output, to obtain the images in this paper, and only for aesthetic reasons, we exported the simulation meshes at a given interval of time, and imported them into a dedicated modeling software (Blender v2.78). Also, for illustrative purposes only, and to showcase the possibilities, we post-processed the model (with a smoothing algorithm) and rendered it to obtain the final results, see Fig. 11.

Our standard scene, unless stated otherwise, contains 983k voxels of 0.5 cm^3, allowing a simulation volume of 0.9 m $\times$ 0.9 m $\times$ 0.20 m similar to Fig. 2. It was also made of 20 bricks of approximately 30k voxels each (of size 0.3 m $\times$ 0.15 m $\times$ 0.1 m) as well as an iron pipe (of about 15k voxels, with 0.15 m of diameter and 0.9 m of length) going under the bricks; the rest of the wall was made of plaster voxels. The particle system produces 500 water particles per second, and the mass–spring system contains about 2 million springs.

With those parameters, it is only during the displacement of voxels and the calculation of cracks that our simulation slows down. It takes about 1.5 s to displace about 150k voxels (corresponding to the displacement following the update of the mass–spring system) and about 2.0 s to calculate the resulting cracks (for which almost all of the 1 million voxels are checked). Our implementation is running on a single thread and neither the voxel engine nor the mass–spring system was implemented with any kind of optimization techniques. Beyond this worst case, the simulation runs at an average of 5–6 fps; it takes about 3 min in Blender to obtain nicer rendered results of Fig. 12.

Fig. 13 shows the result of our deep weathering technique applied to an iron pipe in a Manhattan-like building. In the figure

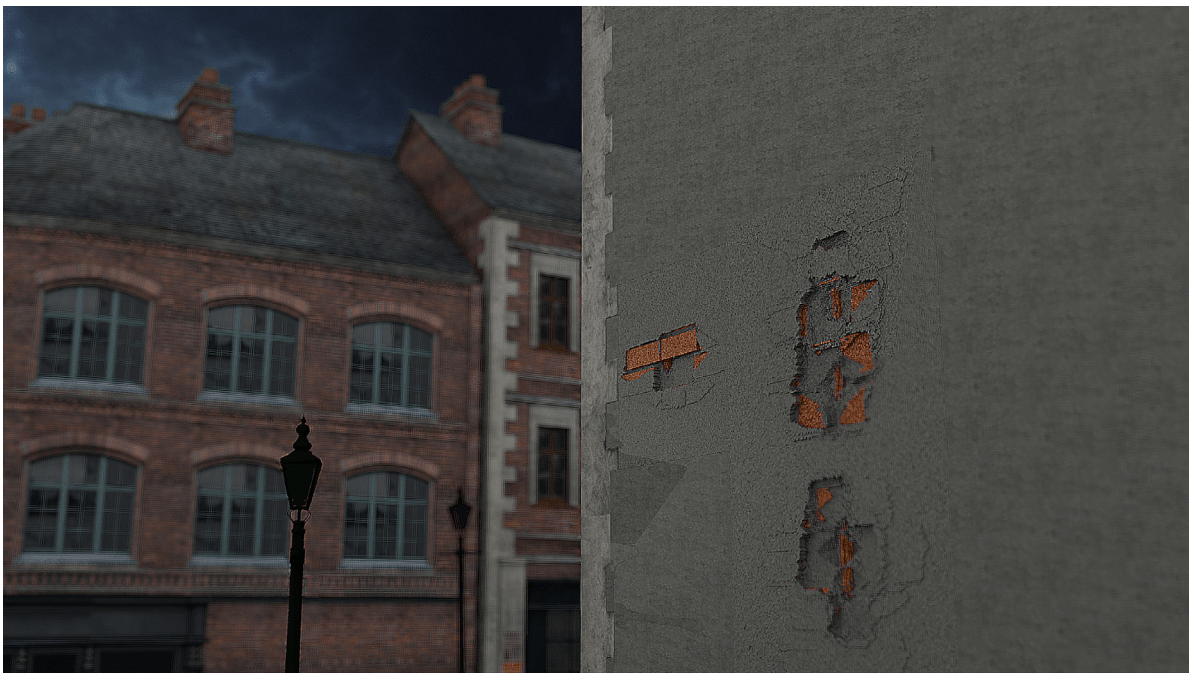

Fig. 12. An older building with plaster and bricks falling off one of its walls. For illustrative purposes, in this render, we have not smoothed the surfaces out, so voxels are visible in some areas, but the original building textures have been kept in non-weathered areas. Also, a sharpening post-processing filter has been applied to enhance voxel visibility.

we can appreciate the effect of water corroding an iron pipe deeply inside the wall, and the chemical and physical interactions that resulted in some plaster being removed, exposing concrete and the rusted iron itself. The simulation results were duplicated (the bottom pipe is a duplicate of the top one), before the final smooth render.

Fig. 12 illustrates the same effect on a Victorian London building where the plaster has fallen to reveal the bricks underneath. To obtain this rendering, we ran two separate simulations: the first one corresponds to the leftmost fallen plaster; and the second to the right area with fallen plaster. This second area was bigger with a volume of 1.4 m $\times$ 1.4 m $\times$ 0.20 m and about 2.6 million voxels.

In Fig. 14 we can see another degradation example of a house with fallen plaster and bricks that are surfacing because of some material expansion (e.g., frozen water) inside the wall. To obtain this rendering, we ran three separate simulations on three distinct simulation areas.

5.2. Discussion

Our deep weathering simulation system combines a particle system with a voxel engine. Each voxel is linked to its neighbors via a mass–spring system that represents relations between the voxels composing the walls.

In this article, we only focused on simulating the creation of rust and cracks in walls that arise from the infiltration of water within the materials of a wall (i.e., concrete, plaster, bricks) to reach the inner part of the wall, and interact with the materials there, like metallic structures. However, our method can easily be applied to other deep weathering effects such as water infiltration and the creation of cracks due to the creation of ice (as seen in Fig. 15 for our standard wall).

On the validation side of this work, although qualitative comparison with real-world images is fairly easy, quantitative validation is incredibly hard to do, as well as comparing to previous results. Actually, there are no concrete previous works to compare with, except for isolated aspects. For instance, we could compare our specific crack-generation module with previous approaches, such as the ones by Bosch et al. [25], Iben and O'Brien [26], Muller [27], Muguercia et al. [3], and Hann and Wojtan [28], to cite just a few. However, in this case, we would only be comparing one subpart of our approach with state-of-the-art specialized approaches, which is far from our current objectives.

Fig. 13. A building with a visible rusting iron pipe exposing concrete and the pipe itself, top: from a distance view; bottom: closeup view.

Fig. 14. Closeup view from the building where plaster and bricks fell off from a wall. Note that the original building textures have been kept in non-weathered areas. A sharpening post-processing filter has been applied to enhance voxel visibility.

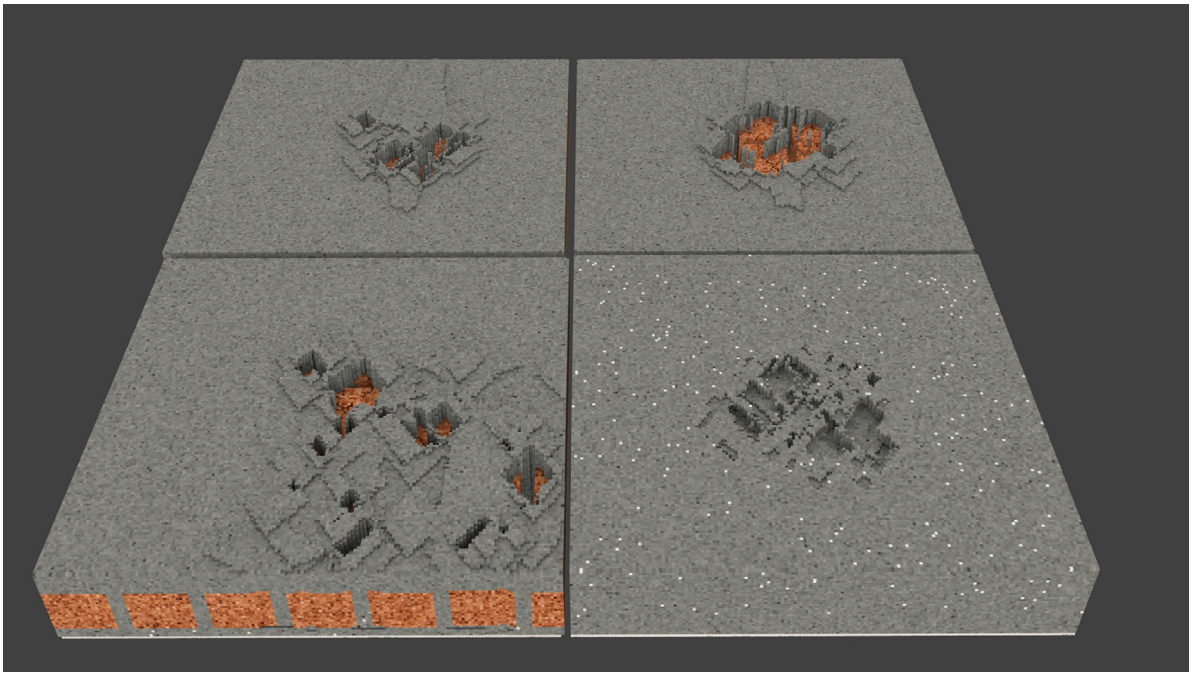

Fig. 15. Ice formation on a layer of plaster lying on top of bricks, from left to right and top to bottom: (1) a small rain area; (2) small rain density area under a longer exposure; (3) same exposure but with a larger area; (4) a wall with plaster only.

Other previous works, such as the ones mentioned in our Previous Work Section, focus on different phenomena than our proposal, so comparisons are nearly impossible.

Although our approach presents promising results on deep weathering effects it nevertheless suffers from a few drawbacks:

- The weathered areas should be manually created by the designer, in a way completely isolated from the rest of the building. It would be interesting but left as future work, to integrate this system into a whole-building degradation simulation, which could retrieve aging information from an independent simulation module such as the one by Munoz-Pandiella et al. [38]. Also, rain density and direction could be used to automatically determine the most weathered areas, as well as puddle formation and similar effects.
- Very rigid bodies (e.g., bricks) were not displaced as realistically as we wanted them to, since our implementation could not exercise any torque on them (and therefore bricks were unable to rotate). Depending on the type of building and the weathering effect, this can be an issue. A possible solution

would be to move and rotate every brick voxel as a single rigid body when a displacement of a brick voxel occurs.
- Mass–springs systems can be troublesome to configure and predict. Even if it is not the focus of our work to allow for easier interaction, other methods to link voxels may be explored.
- Voxels of the simulation area need to be entirely generated from scratch.
- Since the results are exported in Blender for the final rendering, it is always possible to work by "pieces" in the simulation of a large facade, so larger models can be dealt without any problems.
- More effects, such as water flow on the exterior wall surface could also be added [11,33].
- Finally, as of now, our technique only generates discrete moments in time, as shown in Fig. 15. Generating an animation would mean adding synthetic frames that have nothing to do with our technique, but with animating falling particles, and displacing bricks which, eventually, may fall off. All these intermediate frames are quite important and interesting, but we consider them to fall beyond the scope of our paper, which is focused on the simulation itself and not on producing believable animations.

6. Conclusions

In this paper, we presented a method to simulate deep weathering effects applied to architectural buildings. Our method combines a particle system with a voxel engine and allows us to simulate the infiltration of water particles within a building's outer (e.g., plaster, concrete, bricks, etc.) and inner (e.g., iron pipes and other metallic structures) materials, their interaction with those materials, the generation of rust and similar expansive effects, and the consequent displacement of outer material under the pressure created by the formation of rust.

This allowed us to simulate the deterioration of the outer materials of a building (represented by the fall of plaster in our simulations) and the exposure of inner materials to weathering effects, thus accelerating the formation of rust (when water interacts with metallic objects).

While not targeted to real-time applications, our method presents acceptable performance and can manage up to millions of voxels and hundreds of particles, which has been demonstrated by our urban examples showing some complex deep weathering effects.

In the future, we will focus on other deep weathering effects such as the degradation effects of materials due to penetration of moss or other biological compounds, together with other situations that result in important damage to a building wall.

CRediT authorship contribution statement

Adrien Verhulst: Software, Writing – original draft, Formal analysis, Writing – review & editing. **Jean-Marie Normand:** Conceptualization, Writing – review & editing. **Guillaume Moreau:** Conceptualization, Writing – review & editing. **Gustavo Patow:** Conceptualization, Formal analysis, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article

Acknowledgments

Adrien Verhulst was supported by a joint research grant from the École Centrale de Nantes, France, and Audencia Business School, France. This work was also partially funded by Grant PID2021-122136OB-C22 funded by MCIN/AEI/10.13039/501100011033 and by ERDF A way of making Europe.

References

[1] Dorsey J, Rushmeier H, Sillion F. Digital modeling of material appearance. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.; 2008.
[2] Mérillou S, Ghazanfarpour D. A survey of aging and weathering phenomena in computer graphics. Comput Graph (Pergamon) 2008;32(2):159–74.
[3] Muguercia L, Bosch C, Patow G. Fracture modeling in computer graphics. Comput Graph 2014;45:86–100.
[4] Roberge PR, Tullmin M. Handbook of Corrosion Engineering. McGraw-Hill handbooks, New York, NY: McGraw-Hill Professional; 1999.
[5] Hsu S-c, Wong T-t. Simulating Dust Accumulation. IEEE Comput. Graph. Appl. 1995;15(1):18–22.
[6] Galin E, Guérin E, Peytavie A, Cordonnier G, Cani M-P, Benes B, Gain J. A review of digital terrain modeling. Comput Graph Forum 2019;38(2):553–77. http://dx.doi.org/10.1111/cgf.13657.
[7] Blinn JF. Simulation of Wrinkled Surfaces. In: Proceedings of the 5th Annual conference on computer graphics and interactive techniques. SIGGRAPH '78, New York, NY, USA: ACM; 1978, p. 286–92.
[8] Paquette E, Poulin P, Drettakis G. The simulation of paint cracking and peeling. In: In Graphics Interface. 2002, p. 59–68.
[9] Beneš B, Těšínský V, Hornyš J, Bhatia SK. Hydraulic erosion. Comput Anim Virtual Worlds 2006;17(2):99–108.
[10] Miller G. Efficient algorithms for local and global accessibility shading. In: Proceedings of the 21st Annual conference on computer graphics and interactive techniques - SIGGRAPH'94. ACM Press; 1994, p. 319–26. http://dx.doi.org/10.1145/192161.192244.
[11] Dorsey J, Pedersen HK, Hanrahan P. Flow and changes in appearance. In: Proceedings of the 23rd Annual conference on computer graphics and interactive techniques - SIGGRAPH '96. 1996, p. 411–20.
[12] Dorsey J, Hanrahan P. Modeling and Rendering of Metallic Patinas. In: Proceedings of the 23rd Annual conference on computer graphics and interactive techniques. SIGGRAPH '96, New York, NY, USA: ACM; 1996, p. 387–96.
[13] Chang Y-X, Shih Z-C. Physically-Based Patination for Underground Objects. Comput Graph Forum 2000;19(3):109–17.
[14] Mérillou S, Dischler J-M, Ghazanfarpour D. Corrosion: Simulating and Rendering. In: Proceedings of graphics interface 2001. GI '01, Toronto, Ont., Canada, Canada: Canadian Information Processing Society; 2001, p. 167–74.
[15] Dorsey J, Edelman A, Jensen HW, Legakis J, Pedersen HKh. Modeling and Rendering of Weathered Stone. In: Proceedings of the 26th Annual conference on computer graphics and interactive techniques. SIGGRAPH '99, New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.; 1999, p. 225–34.
[16] Shahidi S, Merillou S, Ghazanfarpour D. Phenomenological Simulation of Efflorescence in Brick Constructions. In: Poulin P, Galin E, editors. Eurographics workshop on natural phenomena. The Eurographics Association; 2005, p. 17–23.
[17] Mérillou N, Mérillou S, Galin É, Ghazanfarpour D. Simulating how salt decay ages buildings. IEEE Comput Graph Appl 2012;32(2):44–54.
[18] Desbenoit B, Galin E, Akkouche S. Simulating and modeling lichen growth. Comput Graph Forum 2004;23(3):341–50.
[19] Hirota K, Tanoue Y, Kaneko T. Generation of crack patterns with a physical model. Vis Comput 1998;14(3):126–37.
[20] Hirota K, Tanoue Y, Kaneko T. Simulation of three-dimensional cracks. Vis Comput 2000;16:371–8.
[21] O'Brien J, Hodgins J. Graphical model and animation of brittle fracture. In: In Proceedings of SIGGRAPH'99. 1999, p. 137–46.
[22] O'Brien JF, Bargteil AW, Hodgins JK. Graphical modeling and animation of ductile fracture. ACM Trans Graphics 2002;21(3):291–4.
[23] Gobron S, Chiba N. Crack pattern simulation based on 3D surface cellular automata. Vis Comput 2001;17(5):287–309.
[24] Federl P, Prusinkiewicz P. Finite element model of fracture formation on growing surfaces. In: Proceedings of international conference on computational science. 2004, p. 138—145.
[25] Bosch C, Pueyo X, Mérillou S, Ghazanfarpour D. A Physically-Based Model for Rendering Realistic Scratches. Comput Graph Forum 2004;23(3):361–70.
[26] Iben HN, O'Brien JF. Generating Surface Crack Patterns. In: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. 2006, p. 177–85.

[27] Müller M, Chentanez N, Kim T-Y. Real time dynamic fracture with volumetric approximate convex decompositions. ACM Trans Graph 2013;32(4):115:1–115:10.
[28] Hahn D, Wojtan C. Fast approximations for boundary element based brittle fracture simulation. ACM Trans Graph 2016;35(4):104:1–104:11.
[29] Chen Y, Xia L, Wong T-T, Tong X, Bao H, Guo B, Shum H-y. Visual simulation of weathering by γ-ton tracing. ACM Trans Graph 2005;24(3):1127.
[30] Kider JT. Simulation of 3D Model, Shape, and Appearance Aging by Physical, Chemical, Biological, Environmental, and Weathering Effects (Ph.D. thesis), Department of Computer and Information Science, University of Pennsylvania; 2012, p. 117.
[31] Cutler B, Dorsey J, McMillan L, Müller M, Jagnow R. A procedural approach to authoring solid models. ACM Trans Graph 2002;21(3):302–11.
[32] Ishitobi A, Nakayama M, Fujishiro I. Visual simulation of weathering coated metallic objects. Vis Comput 2020;36(10–12):2383–93. http://dx.doi.org/10.1007/s00371-020-01947-w.
[33] Bosch C, Laffont P-Y, Rushmeier H, Dorsey J, Drettakis G. Image-guided weathering: A new approach applied to flow phenomena. ACM Trans Graph 2011;30:20:1–20:13.
[34] Petroski H. Invention By Design: How Engineers Get from Thought To Thing. Cambridge, MA, USA: Harvard University Press; 1996.
[35] Hall C, Hoff WD. Water transport in brick, stone and concrete. CRC Press; 2011..
[36] Pel L. Moisture transport in porous building materials (Ph.D. thesis), the Netherlands: Eindhoven University of Technology; 1995.
[37] Lorensen WE, Cline HE. Marching cubes: A high resolution 3D surface construction algorithm. In: Proceedings of the 14th Annual conference on computer graphics and interactive techniques - SIGGRAP'87. ACM Press; 1987, p. 163–9. http://dx.doi.org/10.1145/37401.37422.
[38] Munoz-Pandiella I, Bosch C, Merillou N, Patow G, Merillou S, Pueyo X. Urban weathering: Interactive rendering of polluted cities. IEEE Trans Visual Comput Graph 2018;24(12):3239–52. http://dx.doi.org/10.1109/tvcg.2018.2794526.