Fire Safety Journal 64 (2014) 48–60

Contents lists available at ScienceDirect

Fire Safety Journal

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

Interactive fire spread simulations with extinguishment support for Virtual Reality training tools

Aitor Moreno [a,*], Jorge Posada [a], Álvaro Segura [a], Ander Arbelaiz [a], Alejandro García-Alonso [b]

[a] Vicomtech-IK4, Mikeletegi Pasealekua 57, 20009 San Sebastián, Spain
[b] University of the Basque Country, Paseo Manuel de Lardizabal 1, 20018 San Sebastián, Spain

ARTICLE INFO

Article history:
Received 27 March 2013
Received in revised form
1 October 2013
Accepted 21 January 2014

Keywords:
Fire spread simulation
Wildfires
Urban fires
Virtual Reality
Extinguishment support
Spotting fires

ABSTRACT

Virtual Reality training for fire fighters and managers has two main advantages. On one hand, it supports the simulation of complex scenarios like big cities, where a fire cannot be simulated in the real world. On the other hand, fire fighting VR simulators allow trainees to experience situations as similar as possible to real fire, reducing the probability of accidents when they are practising exercises with real fire.

The success of the Virtual Reality training tools also depends on how close to reality the simulation process is. This work provides fire spread algorithms for forest and urban environments, which can be used at interactive rates. Due to the interactive nature of the algorithms, the users are able to fight the fire by throwing extinguishing agents.

Although the algorithms assume many simplifications of the problem, their behaviour is satisfactory. This is due to the efficient management of the cell states in a 3 m × 3 m cell grid. Also the variables that have more influence on fire propagation constitute the core of the algorithms. The overall system deals with user extinguishment actions, natural and artificial firebreaks, variable wind conditions (even at a cell level) and non-contiguous fire propagation (embers and spotting fires). The unified forest/urban model leads to an object oriented architecture which supports the fire propagation algorithms. This also allows the system to compute efficiently mixed forest–urban environments.

1. Introduction

The impact of forest and urban fires on the society is high. Although the consequences might vary from an event to another, the impact normally resides on the potential economical losses due to the fire, the damage to the environment in short and long terms, the damages to urban structures and the eventual human casualties.

Despite the preventive measures that avoid the development or limit the impact of the forest and urban fires, such fire events will happen eventually. At that point, the fire fighters will be called into the field to limit the devastation of the fires and to protect the people.

The fire fighters and managers must be fully trained professionals to do their job with the best possible outcome and to reduce unnecessary risks. The training is composed of theoretical content and some supervised practical exercises with controlled real fire. Essentially, these practical exercises are oriented to teach the trainees how to wear the equipment in warm environments under stress situations. Due to security reasons, the practical exercises are constrained to just a few physical simulators with induced fuel gas fire. In these situations, the fire does not behave realistically, so the fire behaviour knowledge is left at a theoretical level till the first time the trainees go to the field in a real fire event.

Virtual Reality techniques have been used as a key technology in driving simulators, machinery handling simulators, etc. The application of VR techniques to the fire fighting scenario helps in the training process by increasing the possible scenarios and modifying the conditions of the training sessions. Furthermore, the absence of real fire increases dramatically the security and therefore, the safety measures can be reduced significantly.

From the point of view of such a VR simulation system for training fire fighters, there is a main algorithmic element: the simulation of how the fire spreads as simulation time advances. If a very unrealistic behaviour of the fire is shown to the trainees, the *immersiveness* and the credibility of the simulation will be reduced dramatically and the training objectives would be compromised. The introduction of complex mathematical models to simulate the fire behaviour makes difficult a realtime implementation, which is a mandatory requirement in VR setups. Thus, to meet the interactivity requirements, a simplification of the algorithms involved in the fire spread simulation is needed.

* Corresponding author.
E-mail address: amoreno@vicomtech.org (

http://dx.doi.org/10.1016/j.firesaf.2014.01.005
0379-7112

The targeted scenarios for the VR training tool are the forest and urban areas. The different types of vegetation and buildings modify how the fire spreads. Additionally, the apparition of spotting fires is common in both forest and urban areas, being one of the main fire spreading mechanisms.

The suppression of the fire is supported in two ways: by self-extinguishment (fuel combustion) and by the action of the fire fighters. The interactive nature of the VR simulation is required for the trainees, since they interact with the fire behaviour by throwing an extinguishing agent.

In this work, we present fire spread algorithms that can be used in real time within interactive virtual simulations. The algorithms are intended to produce approximated but fast results that could be used in the training of fire fighters and managers.

In the next section, some of the related work for fire spread will be reviewed. Next, the proposed algorithms for the forest and urban environments will be described, followed by the validation. Finally, the conclusions and future work will be addressed.

2. Related work

There are two major models of fire simulation: empirical models and physical models.

The empirical models follow the experiences gathered with real fires. These models use statistical relationships found between the fire evolution and different parameters tested on the field [1]. Within this group, we can mention FARSITE [2], which uses the Huygens principle of wave propagation.

The physical models use convection and heat transfer mechanisms and/or Computational Fluid Dynamics methods. The main mathematical tools they use are partial differential equations and reaction diffusion systems. Fire Dynamic Simulator (FDS, National Institute of Standards and Technology - NIST) or FIRETEC [3] follow this approach.

Seron et al. [4] and Ferragut et al. [5,6] provided physical models with some empiric variables, which make their solutions hybrid. The advantage of these models is their accuracy in the fire prediction. Morvan et al. [7] used them to study the interactions between fire fronts, but the computational effort is very high. The mathematical models are too complex and computers can only provide approximate solutions [8]. Another consequence is that increasing the spatial resolution causes too long computational times.

Unlike the two previous models, other research works have taken a different direction from the complex mathematical models. Their objective is to reduce computation time and to implement a real time simulation. Achtemeier [9] presented the *Rabbit Model*, a collection of basic rules of fire evolution, which are implemented as autonomous agents (the rabbits). The scope of the *Rabbit Model* is limited to the evolution of wildland fires.

Lee et al. [10] proposed a physical model for urban fires. The authors use equations to describe the heat transfer between buildings (radiation and convection), the temperature modification, and flame shape (direction and length) coming out through the windows.

Weise and Biging [11] proposed a physics based model to simulate the fire spread in non-homogeneous cities with high resolution. Cheng and Hadjisophocleous [12] modelled the fire spread in buildings taking into account the connectivity between rooms and stories. Stern-Gottfried and Rein [13] introduced the *travelling fires* to support the fire dynamics in buildings.

Iwami et al. [14] introduced a very descriptive physical model for urban fires, providing different stages for each considered building type. Ohgai et al. [15] presented a physical model using cellular automata over a grid of 9 m^2.

The algorithms proposed in this work present an urban and forest fire spreading simulation, whose main characteristics are:

- In forest areas, the fire evolution is based on the terrain topology, material and wind conditions. In urban areas, in order to obtain more accurate results, the different building characteristics are used.
- The fire may cross rivers, firebreaks or other barriers by throwing firebrands, producing spotting fires on the other side of the barrier. This mechanism is also used to spread the fire between buildings. In a similar way, urban fire can spread to forest areas and vice versa.
- Very low complexity, allowing real time simulation even with standard computing power.
- Fire suppression support. Throwing the extinguishing agent affects how the fire spreads.

Sections 3 and 4 describe the proposed algorithms. Following sections present results and performance analyses.

3. Fire spread algorithms for wildland areas

The fire fighting simulator has to get rapid results for the fire spread process in forest and urban areas. The system (see Fig. 1) must provide a fast response to support the user interactions and dynamic changes of the wind conditions.

The main objective of this work is to provide novel algorithms to reduce the algorithmic complexity and the processing time. They follow and extend the works of Achtemeier [9], Iwami et al. [14] and some of our previous work [16]. The most relevant variables are taken into account: wind and slope [11,17].

3.1. Field definition

The simulation algorithms utilise a regularly divided field (grid). Each cell of the field has its geometrical information (position and altitude) and its state. Fig. 2 shows in a graphical way the terminology used to define the relationships of the cells in the field. For a given cell, we define its neighbours as the 8 closer cells. The surroundings of a cell include a set of cells enclosed in an elliptical or circular region.

Each cell has a type, which determines the nature of the cell (dry grass, tall trees, water bodies, roads, etc.) and its behaviour. Also, each cell has variables which are updated by the algorithms in each simulation step: *State*, *FirePower*, *MaxFirePower*, *Fuel*, *AmountAgent*, etc. These variables will be defined in the algorithms.

3.2. Cell states

All the cells in the field have an internal state which describes the state of the fire that exists in such cell. The different states are `Safe`, `Activated`, `Burnt`, `Survive`,

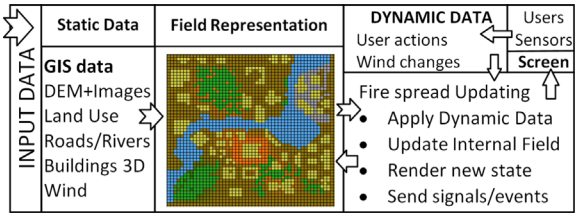


Fig. 1. The simulation architecture builds the scenario from static information. During the simulation, the fire evolves according to the user actions, the fire spread algorithms and the wind changes [16].

`BarrierCrossing` and `FireStopped` (see Fig. 3). An extended description of the states can be found in [16].

When a cell is in the `Safe` state, there is no fire in it. The `Activated` state indicates that there is an active fire in the cell.

`Burnt` is the final state of a cell. All its fuel is burned and it has cooled down. The `Survive` state is a pseudo-final state, where all its fuel is burnt, but still has residual heat (*FirePower*). Even when its fuel is completely consumed by the fire, the cell can irradiate some heat to other cells. Eventually, the cell will pass to the `Burnt` state.

`BarrierCrossing` is a state that controls whether the fire spreads through cells that represent a river or firebreak (Spotting Fires).

`FireStopped` means that the fire in the cell has been stopped by an external extinguishing agent.

3.3. Simulation step

Algorithm 1 shows the *SimulationStep* procedure, which is run in each simulation step and defines the main simulation loop. Firstly, the cells that were activated in the previous step (stored in the temporary *ActivatedCellList*) are moved into *ActiveCellList*.

Afterwards, the *UpdateState* method of each active cell is called. If the state of the cell is `Burnt`, the cell is removed from the list.

3.4. Update step method

Algorithm 1. The *SimulationStep* procedure defines the main simulation loop.

```
1: procedure SimulationStep()
2:   // Update list ActiveCellList
3:   for each cell cc in ActivatedCellList do
4:     ActiveCellList.Add(cc)
5:     ActivatedCellList.Remove(cc)
6:   end for
7:   // Compute new states
8:   for each cell cc in ActiveCellList do
9:     cc.UpdateState()
10:    if cc is Burnt then
11:      ActiveCellList.Remove (cc)
12:    end if
13:  end for
14: end procedure
```

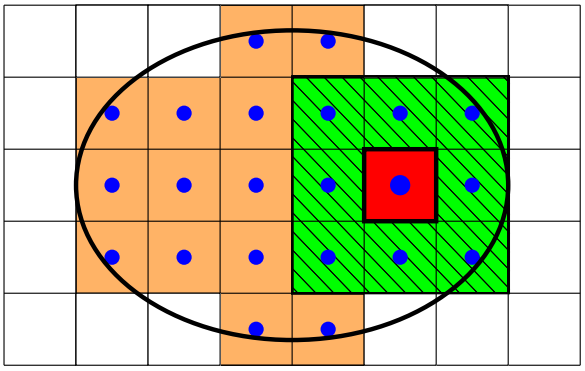


Fig. 2. Terminology: the neighbours of the red cell (in striped green) and its surrounding cells (cells inside an ellipse or a circle). (For interpretation of the references to colour in this figure caption, the reader is referred to the web version of this paper.)

The *UpdateState* method is presented in Algorithm 2. This is a generic version of the method, since every subclass of *Cell* will provide its own implementation. The method contains three main actions: (i) consume fuel, (ii) evaporate water in the surroundings and (iii) spread to other cells (ignite cells).

The *ConsumeFuel* function will reduce a certain amount of fuel, calculated as a function of the existing *FirePower*. As the fuel is consumed, the *FirePower* will be increased. As an example, a grass cell will consume a small amount of fuel at the beginning, but it will increase the *FirePower* very fast, increasing the amount of consumed fuel in each step.

The *FirePower* cannot grow infinitely, so a maximum *FirePower* (*MaxFirePower*) is defined for each cell type. When a cell is in the `Survive` state, the *FirePower* decreases in each step. Having no fuel, the cell will pass to the `Burnt` state when the residual *FirePower* becomes zero.

The *Evaporation* method reduces a certain amount of extinguishing agents in the surrounding cells (the method is addressed in Section 3.9).

The *SpreadSlopeWind* and *SpreadSpotting* methods evaluate if the cell triggers the ignition of new cells. The methods are described in their corresponding sections. The activation of the ignited cells is performed in the *IgniteFire* method.

The *UpdateState* method finishes checking if the remaining fuel is zero. In this case, the cell state is set to `Burnt` or `Survive`.

3.5. IgniteFire method

The *IgniteFire* method is presented in Algorithm 2 and it is run when a cell activates another cell. The state of the target cell is set to `Activated` and an initial *FirePower* is calculated. This value takes into account the *FirePower* in the cell which is igniting this cell, and the *FirePower* of the neighbour cells.

The newly ignited cell is added to *ActivatedCellList*.

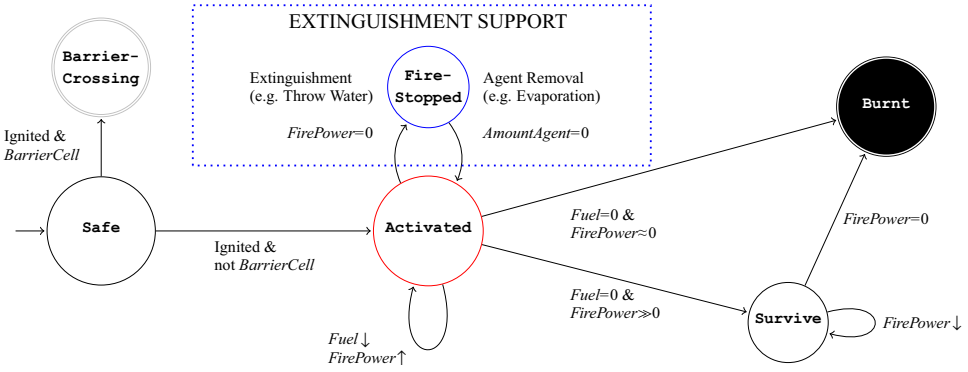


Fig. 3. Cell states and transitions.

Algorithm 2. The methods *UpdateState* and *IgniteFire* are the basic actions when any active cell is updated or ignites another cell.

```
1: method Cell::UpdateState()
2:    if state==FireStopped then return
3:    ConsumeFuel ()
4:    Evaporation ()
5:    // Spread to other Cells
6:    SpreadList.Add (SpreadSlopeWind())
7:    SpreadList.Add (SpreadSpotting())
8:    for each cell cc in SpreadList do
9:       IgniteFire(cc)
10:   end for
11:   if Fuel==0 then
12:      SetState (Burnt or Survive)
13:   end if
14: end method
```

```
1: method Cell::IgniteFire(Cell c)
2:    calculate intensity (neighbour cells)
3:    c.SetState (Activated)
4:    c.SetFirePower (intensity)
5:    // In next step, this cell c will be active
6:    ActivatedCellList.Add(c)
7: end method
```

3.6. Spread fire: slope and wind

The *SpreadSlopeWind* method is presented in Algorithm 3. The `Safe` neighbours of a given cell are visited. For each one, the slope is calculated. A probability (p) is computed considering slope, wind and *FirePower*. *PassTest* generates a random number; if p is greater, then a function returns *true*, and the cell is selected for ignition.

Algorithm 3. The method *SpreadSlopeWind* checks neighbours for ignition.

```
1: method Cell::SpreadSlopeWind()
2:    create new Cell list IgnitedList
3:    for each neighbour cell vc do
4:       calculate slope
5:       // p ∈ (0,1) is a function of
6:       // slope, wind and FirePower
7:       p=CalculateProbability()
8:       if PassTest (p) then
9:          IgnitedList.Add (vc)
10:      end if
11:   end for
12:   return IgnitedList
13: end method
```

3.7. Barriers: rivers, firebreaks, roads

Barrier cells model non-combustible barriers such as rivers or roads. A *BarrierCell* class has its own implementation of *UpdateState* and *IgniteFire* methods. They are presented in Algorithm 4 and they have significant differences from the generic implementation, explained in Sections 3.4 and 3.5.

Only the *SpreadSpotting* method is used in *BarrierCell::UpdateState* (see Section 3.8) and the ignition probabilities are heavily reduced (75%).

When a *BarrierCell* is activated, its state changes to `BarrierCrossing` instead of `Activated`. As there is no real fuel to burn, there is no *FirePower*. We define *CrossIntensity* as the intensity of the radiation of the fire through a *BarrierCell*. The initial value is the 20% of the sum of its neighbours' *FirePower*. If the cell is surrounded only by *BarrierCell*'s, the *CrossingIntensity* will be set as the 50% of the maximum *CrossingIntensity* value found in its neighbours.

Algorithm 4. The methods *UpdateState* and *IgniteFire* for barrier cells.

```
1: method BarrierCell::UpdateState()
2:    SpreadList.Add (SpreadSpotting())
3:    for each cell cc in SpreadList do
4:       IgniteFire(cc)
5:    end for
6: end method
```

```
1: method BarrierCell::IgniteFire(Cell c)
2:    calculate crossIntensity (neighbours)
3:    if crossIntensity > 1 then
4:       c.SetState(BarrierCrossing)
5:       // In next steps, c will be active
6:       ActivatedCellList.Add(c)
7:    end if
8: end method
```

Algorithm 5. The method *SpreadSpotting* checks the surrounding cells for ignition (*Spotting Fires*).

```
1: method Cell::SpreadSpotting()
2:    create new cell list IgnitedList
3:    create new cell list Candidates
4:    get ellipse parameters (wind)
5:    Candidates=SelectCells (ellipse)
6:    for each cell vc in Candidates do
7:       calculate slope
8:       // p ∈ (0,1) is a function of
9:       // slope, wind and FirePower
10:      p=CalculateProbability()
11:      if PassTest (p) then
12:         IgnitedList.Add (vc)
13:      end if
14:   end for
15:   return IgnitedList
16: end method
```

3.8. Spread by spotting fires

Spotting fires provide a mechanism to spread fires further than neighbour cells. This mechanism also allows the fire to propagate over natural or artificial barriers (roads, rivers, etc.) and to the surrounding cells (see Fig. 2).

The *SpreadSpotting* method is presented in Algorithm 5. Its behaviour, although similar to *SpreadSlopeWind*, takes into account the non-combustible surrounding cells.

In this method, an active cell may ignite other cells within an elliptical shape. The ellipse is a function of the wind direction and the velocity. For each surrounding cell, an ignition probability is calculated. If the *PassTest* function returns *true*, the cell is selected for ignition.

The vegetation type is very important in the ignition of spotting fires. Tall trees are more prone to propagate embers through the air (reaching distant areas), than grass or bushes.

Fig. 4 shows in vertical bars the *CrossingIntensity* in the cells which represent the river. In the figure, the fire spreads to the other bank.

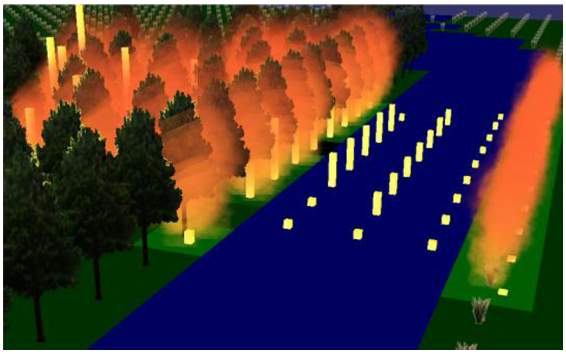


Fig. 4. Visual representation of the *CrossingIntensity* value when the fire crosses a river [16].

3.9. Extinguishment

The calls to *ThrowWater* are triggered by user actions. This is done on the main simulation loop (*SimulationStep*). This method is presented in Algorithm 6. A given quantity of extinguishing agents reduces instantaneously some *FirePower* which depends on the type of the agent.

When a cell has no *FirePower*, the state changes to `FireStopped` and the remaining agent is accumulated in the cell. In this state, any additional thrown agent increases the cell *AmountAgent*. This accumulated agent can be evaporated by nearby active cells.

Algorithm 6. The method *ThrowWater* is triggered by the user actions and supports the suppression of the fire with extinguishing agents.

```
1: method FloorCell::ThrowWater(factor)
2:    calculate reduction (factor, Type)
3:    if state==Activated then
4:       FirePower -=reduction
5:       if FirePower <=0 then
6:          SetState (FireStopped)
7:          AmountAgent=−1×FirePower
8:          FirePower=0
9:       end if
10:   els if state==FireStopped then
11:      AmountAgent +=reduction
12:   end if
13: end method
```

The *Evaporation* and *EvaporateWater* methods are presented in Algorithm 7. *Evaporation* is called within the *UpdateState* performed in each simulation step (see Algorithm 2). An active cell selects surrounding cells within a circular area, in which radius is proportional to the *FirePower*. Then, the *EvaporateWater* method of the selected cells is called with a reduction factor as a parameter. This factor takes into account the distance between the cells and the *FirePower*.

The *EvaporateWater* method uses the calculated factor and the extinguishing agent type to calculate the amount of agent to reduce. If *AmountAgent* goes below zero, the cell's state changes to `Activated`. In the next simulation step, the fire will be effectively rekindled.

Algorithm 7. The method *Evaporation* reduces the accumulated extinguishing agent in the surroundings cells by means of the supporting *EvaporateWater* method.

```
1: method Cell::Evaporation()
2:    create new cell list Affected
3:    // this active cell evaporates cells
4:    // within a radius (range)
5:    Affected=SelectCells(FirePower,range)
6:    for each cell cc in Affected do
7:       if cc has extinguishing agent then
8:          set factor (FirePower, distance)
9:          cc.EvaporateWater (factor)
10:      end if
11:   end for
12: end method
1: method Cell::EvaporateWater(factor)
2:    calculate reduction (factor, Type)
3:    AmountAgent -=reduction
4:    if AmountAgent <=0 then
5:       AmountAgent=0
6:       if state==FireStopped then
7:          SetState (Activated)
8:       end if
9:    end if
10: end method
```

4. Fire spread algorithm for urban areas

To support the fire spread in urban areas, the *BuildingCell* and *FloorCell* classes are introduced. A *BuildingCell* is composed of a vertical stack of *FloorCell* cells. Urban cells require the definition of new types of materials. Iwami et al. [14] classify buildings according to their structure and define some basic parameters like the initial fuel. We follow a similar classification that considers three types: *ShantyUnit*, *WoodenUnit* and *SecureUnit* [16].

Algorithm 8 presents the adaptation of the Iwami formulation to the cell representation of the field. The method *initialFuel* is specific for each building type. Although generic for all the *FloorUnit*'s, the method *calculateMaxFirePower* uses the previously calculated value of *initFuel*. The area of the cell (A) is the square of the cell side. The height of a floor (HF) has been chosen to be a constant for all the buildings in the field (3 m). The number of floors (N), the window size (WS) and the quantity of fuel per m^2 ($FUELM2$) are parameters which characterise each of the buildings.

Algorithm 8. The methods *initialFuel* and *calculateMaxFirePower* implement the Iwami formulation for the initial fuel in a *BuildingCell* and its *MaxFirePower*.

```
1: method WoodenUnit::initialFuel()
2:    initFuel=150×FUELM2×A
3: end method
4: method SecureUnit::initialFuel()
5:    initFuel=0.5×1.5×HF+FUELM2×A
6: end method
7: method FloorUnit::calculateMaxFirePower()
8:    Af=0.54×A×pow (FUELM2, 1/3)
9:    Af +=2×N×A
10:   Af+=8×HF×sqrt (A)
11:   Af +=0.09×A×pow (initFuel/A, 2/3)
12:   m=(1/3)×25×WS
13:   fc=FACTOR (BuildingType)
14:   Qmax=min (fc×A, fc×m)
15: end method
```

The *UpdateState* method for *FloorCell*'s is similar to Algorithm 2. The consumption function depends on the building type. Fig. 5

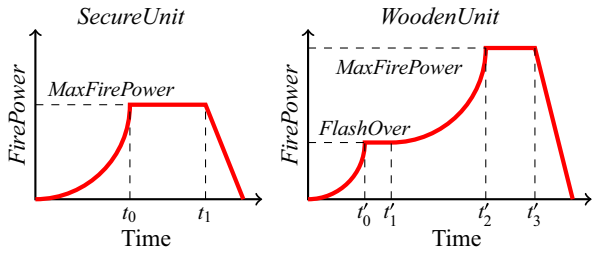


Fig. 5. Evolution of *FirePower* for *SecureUnit* and *WoodenUnit* building types. Each type has a different value for the *MaxFirePower*. The *WoodenUnit* presents a *FlashOver* event, where the *FirePower* rises very quickly consuming the building structure [14].

shows the free fire evolution for the *SecureUnit* and the *WoodenUnit*.

The *SecureUnit* consumes fuel following a quadratic function till the calculated *MaxFirePower* is reached (t_0), limiting the consumed fuel in each simulation step. When the fuel is totally consumed (t_1), the *FirePower* begins to decrease (`Survive` state). The cell passes to the `Burnt` state when the *FirePower* reaches the zero value.

The *WoodenUnit* fire evolution is similar to the presented for the *SecureUnit*, but including a flashover event. After the t'_1 mark, the cell starts to burn it own wooden structure, producing a high increment in the *FirePower* till the *MaxFirePower* is reached (t'_2). When the structure is totally burnt (t'_3), the cell passes to the `Survive` state.

The *Evaporation* method reduces some of the accumulated extinguishing agent in the neighbour cells. In buildings, neighbour cells are contiguous *FloorCell*'s in the same floor, plus two other cells: the cell above and the cell below the *FloorCell* (if they exist).

After checking if the cell can still be active, the fire spreading methods are run. In the buildings, they are *SpreadHorizontal* and *SpreadVertical*. If the cell is part of the facade, the fire can spread to the nearby buildings or to the existing vegetation using *SpreadSpotting* and *SpreadVegetation* methods.

4.1. Fire spread algorithms

The *SpreadHorizontal* and *SpreadVertical* methods are presented in Algorithm 9. Some of the neighbours in the same floor are selected as candidates for ignition. The selection takes into account the *FirePower* and the building type of the cells. For each candidate, a probability is calculated and the cell is ignited if a function *PassTest* returns *true*.

The fire can spread vertically to the cell above and, sometimes, to the cell below. In the method *SpreadVertical*, a probability test is performed for the cell above. Another test is run for the cell below, using a much lower ignition probability.

Algorithm 9. The methods *SpreadHorizontal* and *SpreadVertical* provide the essential mechanism for fire spread in buildings.

```
1: method FloorCell::SpreadHorizontal()
2:    Neighbours=GetNeighboursCells ()
3:    Candidates (Neighbours, FirePower)
4:    for each cell vc in Candidates do
5:       // p ∈ (0,1) is a function of
6:       // slope, wind and FirePower
7:       p=CalculateProbability()
8:       if PassTest (p) then
9:          IgniteFire (vc)
10:      end if
11:   end for
12: end method
1: method FloorCell::SpreadVertical()
2:    state=FloorAbove.getState()
3:    // p ∈ (0,1) is a function of
4:    // slope, wind and FirePower
5:    p=CalculateProbability()
6:    if PassTest (p) then
7:       IgniteFire (FloorAbove)
8:    end if
9:    // Repeat for FloorBelow
10:   // with smaller probability
11: end method
```

The *SpreadSpotting* method is presented in Algorithm 10. It is only run for cells in the building facades.

This method calculates an elliptical shape around a burning *FloorCell*, whose parameters depend on the wind direction and the velocity. All facade *BuildingCell*'s inside this 2D ellipse are considered for ignition. The ignition probability for each *FloorCell* is computed as a function of the wind condition, the distance and the 3D angular deviation from the candidate *FloorCell* to the already burning *FloorCell*.

Algorithm 10. The method *SpreadSpotting* for *FloorCell*'s provides the *Spotting Fires* mechanism for buildings.

```
1: method FloorCell::SpreadSpotting()
2:    if isNotFacade() then return
3:    get ellipse parameters (wind)
4:    List=SelectBuildingCells (ellipse)
5:    for each BuildingCell b in List do
6:       for each FloorCell fc do
7:          dist=distanceTo (fc)
8:          α=angleDeviation (fc)
9:          // p ∈ (0,1) is a function of
10:         // wind, distance and α
11:         p=CalculateProbability()
12:         if PassTest (p) then
13:            IgniteFire (fc)
14:         end if
15:      end for
16:   end for
17: end method
```

The *SpreadVegetation* method is presented in Algorithm 11. As the previous one, it only can be run for cells in the building facades and only the vegetation cells in the neighbourhood are taken into account. For each selected cell, an ignition probability is calculated as a function of the *FirePower*, the vegetation type and the wind direction and velocity.

Algorithm 11. The method *SpreadVegetation* provides the *Spotting Fires* mechanism between buildings and vegetation cells.

```
1: method FloorCell::SpreadVegetation()
2:    if isNotFacade() then return
3:    List=SelectNeighbourCells ()
4:    for each cell cc in List do
5:       // p ∈ (0,1) is a function of
6:       // wind and FirePower
7:       p=CalculateProbability()
8:       if PassTest (p) then
9:          IgniteFire (cc)
10:      end if
11:   end for
12: end method
```

4.2. Extinguishment support

The mechanism to support the suppression of the fire is the same in the buildings and in vegetation cells. The method *ThrowWater* is presented in Algorithm 6.

The *ThrowWater* method is not used directly in the buildings. We have introduced the *WaterJet* concept, which emulates the actions of the fire fighters with high pressure hoses in the facade of the buildings.

A *WaterJet* is conceptualised as a water jet thrown from the exterior of a building to a target *FloorCell*. It has to be in the facade of the building. The *WaterJet* is parameterised by its *WaterJetLevel*, which determines how many *FloorCell*'s are reachable in a straight line. If the value is 1, only the target cell receives the extinguishing agent. If the value is 2, the agent reaches to an extra cell in the water jet direction. Both of them receive half the amount of the agent.

5. Analysis of the algorithms

This section measures and analyses the main parameters involved in the fire spread algorithms. Results show the coherent behaviour of the algorithms regarding their expected behaviour.

5.1. Forest environments

Firstly, we analyse the behaviour of the algorithms in forest environments, which is composed of the following key factors: the terrain (slope and composition), wind conditions and spotting fires. Also, we present the behaviour of the extinguishment support.

5.1.1. Terrain slope and composition

The slope of the terrain has a direct influence on the fire spread direction and the speed. In horizontal fields, the algorithm tends to create circular shapes. On slopes, the fire tends to go up, following the slope direction. The tests confirm that the speed of the fire propagation is the function of the slope (see Fig. 6).

Different cell types also modify the fire spread results. Changing the type or the quantity of the existing fuel in a cell impacts the simulation result. Fig. 7 shows three vegetation cell types (grass, bush and tall trees) in an horizontal terrain with no wind. Each vegetation strip is separated from each other with water cells. The simulation shows the different spread speed of each cell type.

The vegetation cell types have been chosen as a generalisation of the fuel types defined by Rothermel [1] and extended by Albini [18]. In our preliminary implementations, we have selected the fuel type *Short Grass* to represent the grass zones; *Chaparral* and *Brush* to represent bushes; and *Timber* to represent trees in the field (see Table 1). The United States Department of Agriculture provides some basic fire spread parameters for each of the fuel types [19, p. 18].

5.1.2. Wind direction

Wind is one of the most influential variables in fire spread. The wind direction and the speed modify how the fire spreads. Fig. 8 shows how the wind spreads the fire in the corresponding direction.

The fire algorithms use specific wind information in each cell and each time step. In this work, all the cells have the same wind information and it is constant in time. This architecture allows the simulation to consider more realistic wind effects without increasing the simulation time in the fire spread simulation module. Therefore, each cell could be updated with new wind values, which should be computed by another process. This means that a feedback between both modules might appear: the fire spread simulation would modify the fire states in the field, which could be used by the external wind model to update the simulated wind field. This wind field could be used by the fire spread algorithms in the next simulation step.

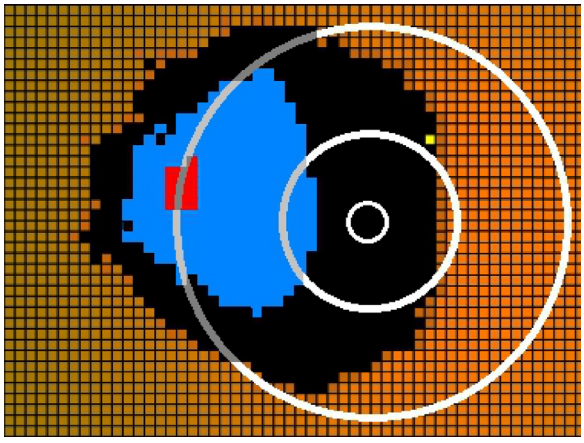


Fig. 6. The fire spreads in a terrain with a hill (represented with white circles) and no wind. The fire started in the red zone and then it spread faster (blue zone) towards the top of the hill. Finally, the fire spread sideways and went down the other side of the hill. (For interpretation of the references to colour in this figure caption, the reader is referred to the web version of this paper.)

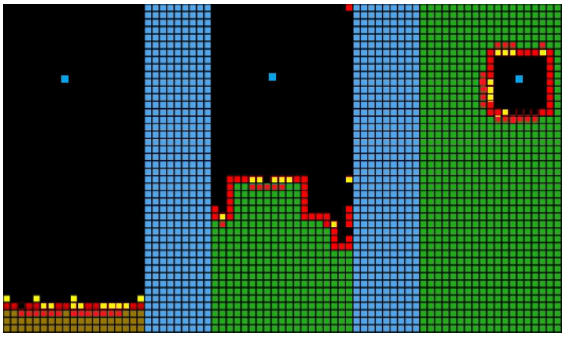


Fig. 7. Scenario with different vegetation types, horizontal terrain and no wind. The fire spreads at different speeds. Each zone (grass, bush, trees) is separated with water bodies (rivers). The ignition points are in similar positions (in blue). (For interpretation of the references to colour in this figure caption, the reader is referred to the web version of this paper.)

Table 1
Mapping from the Fuel Model (FM) [18], and the fuel types defined by the United States Department of Agriculture [19]. The rows in bold correspond to values used in this work.

FM	Description	Fuel model codes
1	**Short grass**	**GR1, GR2, GR4, GR7**
2	Timber grass and understory	GR2, GR4, GR7, GS1, GS2
3	Tall grass	GR3, GR5, GR6, GR7, GR8, GR9
4	**Chaparral**	**SH5, SH7**
5	**Brush**	**SH1, SH2, SH5, SH7, TU5, GS2**
6	Dormant brush	SH1, SH2, SH4, SH6
7	Southern rough	SH3, SH4, SH6, SH8, SH9
8	Compact timber litter	TL1, TL3, TL4, TL5, TL7, TU1
9	Hardwood litter	TL2, TL6, TL8, TL9
10	**Timber (understory)**	**TU1, TU2, TU3, TU4, TU5, SH2**
11	Light logging slash	TL5, SB1, SB2
12	Medium logging slash	SB1, SB2, SB3
13	Heavy logging slash	SB2, SB3, SB4

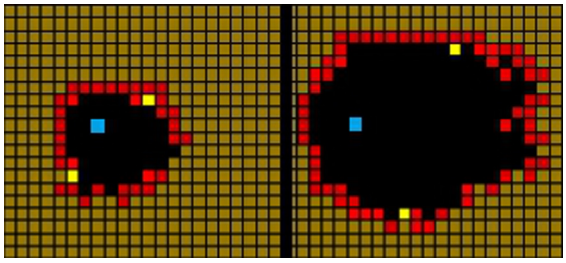


Fig. 8. Two time steps at a horizontal scenario with uniform vegetation and constant wind (from the West). The fire spreads faster in the wind direction.

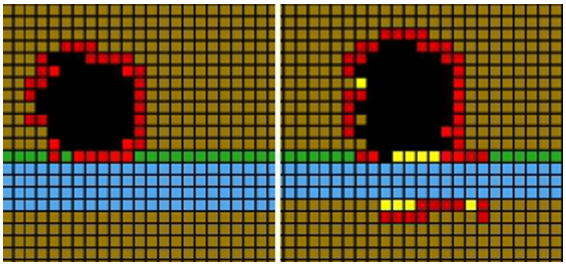


Fig. 9. Two scenarios with a river, horizontal terrain and constant wind (coming from the North). The fire only spreads to the other bank in the scenario with the narrower river.

5.1.3. *Spotting fires*

The algorithms simulate how the fire spreads by throwing embers (*Spotting Fires*). When there is a barrier (river, road, etc.), the spread probability depends on the width of the barrier and on the vegetation on both sides. Fig. 9 shows two examples with the wind coming from the North. The only difference is the width of the barrier. The first barrier is wide enough to stop the fire. The second one does not stop the fire spread.

5.1.4. *Extinguishment*

The extinguishment support was tested with different experiments. Fig. 10 shows the evolution of the *FirePower* in a given cell with extinguishing action from Step 1020 to Step 1260.

Initially, the fire evolves freely until the extinguishing action starts (Step 1020). The extinguishing agent manages to control the fire and starts to accumulate in the cell (Step 1150).

After stopping the extinguishing action in Step 1260, the surrounding active cells evaporate gradually the accumulated extinguishing agent. Once the remaining extinguishing agent is evaporated (Step 1320), the cell rekindles and its *FirePower* starts to increase till the fuel is fully consumed (Step 1540).

5.2. *Analysis of urban environments*

In the urban environments, the fire spreads inside the buildings or by spotting fires from one building to another. In this section some case studies are presented.

Fig. 11 shows the evolution of the *FirePower* in a *WoodenUnit FloorCell*. As the fire evolves freely, eventually it spreads to a neighbour cell.

The main extinguishing action is to throw the extinguishing agent to a facade of a *FloorCell*, trying to reduce its *FirePower*. Fig. 12 presents two possible *FirePower* evolutions in a *ShantyUnit FloorCell*. The extinguishing agent thrown by the *WaterJet* is not enough to put out the fire and the *FirePower* rises again when the extinguishing action stops.

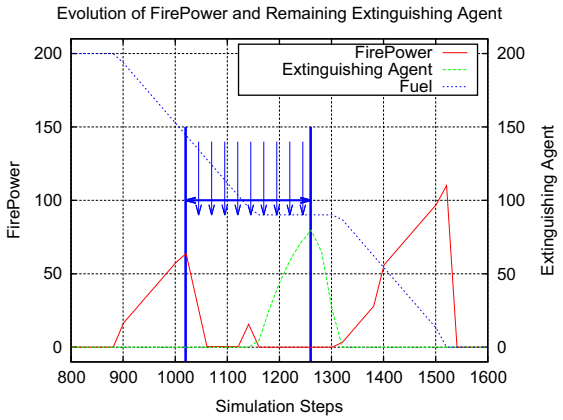


Fig. 10. The evolution of the cell is explained in Section 5.1.4.

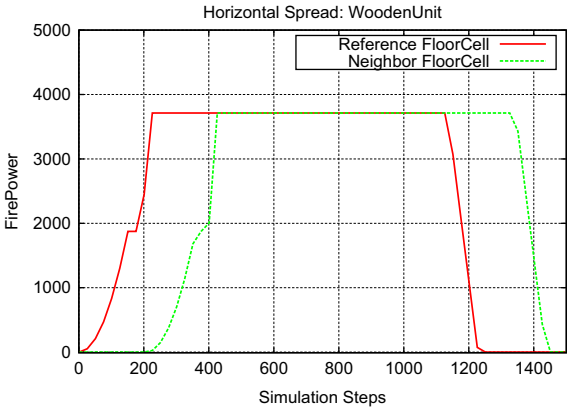


Fig. 11. An urban fire spreads from a *WoodenUnit FloorCell* to another.

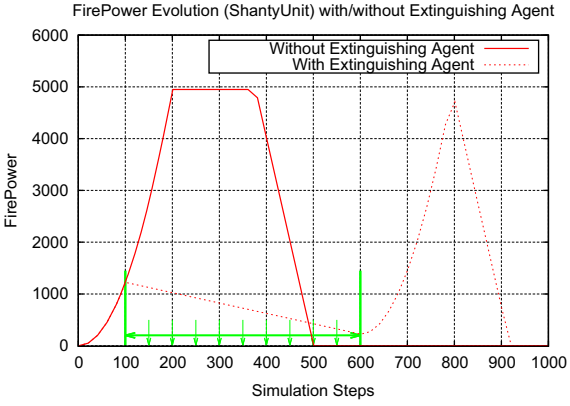


Fig. 12. Free fire evolution in a *ShantyUnit FloorCell* (solid red line). Throwing the extinguishing agent reduces *FirePower* (dotted green line). In the simulation step 600, no more agent is thrown and the fire rekindles. (For interpretation of the references to colour in this figure caption, the reader is referred to the web version of this paper.)

5.3. *Stochastic behaviour analysis*

In Algorithms 3, 5, 9, 10 and 11, a probability is calculated using an abstract *CalculateProbability* function. It combines the base probability for each event (see Table 2) with the cell information to get the final probability. The Algorithm 12 shows how we calculate the spread probability in a vegetation cell.

Algorithm 12. The function *Cell::SpreadProbability* calculates the fire spread probability from a vegetation cell to another. It

Table 2
Base probabilities for some of the most relevant stochastic events in the fire spread algorithms presented in this work.

Base probability of an event	Percentage
Spread in favour of the slope	80
Spread against slope	15
Spread in favour of the wind	80
Spread against wind	10
Spotting Fires spread	10
Horizontal spread in buildings	15
Upwards spread in buildings	30
Downwards spread in buildings	5
Spotting Fires spread in buildings	10

considers the distance between them (*pD*), the relative percentage of *FirePower* (*pFP*), the relative percentage of remaining fuel (*pF*) and the slope (*pS*). For slopes greater than 45°, the *pS* value is clamped to 1.0. For negative slopes, the base probability shown in Table 2 is used.

```
1: method Cell::SpreadProbability(Cell candidate)
2:    pD=cellSize/getDistanceTo (candidate)
3:    pF=fuel / initialFuel
4:    pFP=FirePower / MaxFirePower
5:    angle=getSlopeAngleTo (candidate)
6:    if angle ≥ 0 then
7:       pS=0.50 + angle / (PI/4)
8:       prob=baseProb × pD × pF × pS × pFP
9:    else
10:      prob=baseProbAgainst × pD × pF × pFP
11:   end if
12: end method
```

The *PassTest* function generates a random number and compares it with the calculated probability to check if the event occurs or not.

The values shown in Table 2 were chosen heuristically.

6. Performance analysis

The analysis of the computation performance of the algorithms has been made through multiples scenarios under different conditions. All the measures were obtained using an Intel Quad Core Q9400 processor, 4 GB of RAM and a GeForce GTX 285,Windows 7 64 Bit (Service Pack 1) with the latest stable graphics drivers. In all simulations, the number of active cells per step and the time needed by the algorithms are registered.

Fig. 13 shows the behaviour in a scenario with a constant 45° slope, composed of one single vegetation type and no wind. The number of active cells increases gradually, as the fire spreads. The figure shows a linear evolution in the number of active cells. This is consistent with the algorithms because burnt cells are removed from the active cell list as the new ones are added. The expected quadratic behaviour is converted into linear by the algorithm: the active cells represent approximately the fire front.

Fig. 14 shows performance in an urban area with constant wind. The number of active cells is more irregular, depending on the evolution of the fire across the buildings. Buildings have a different number of storeys. In the statistics, a *BuildingCell* is counted as one active cell, even when it is composed of multiple floors.

In both examples, the simulation time per active cell is around 2 ms. We find 300 ms as the maximum computation time for one simulation step. As the simulation time per step is below 1 s (the simulation step), the interactivity capabilities of the simulations are guaranteed.

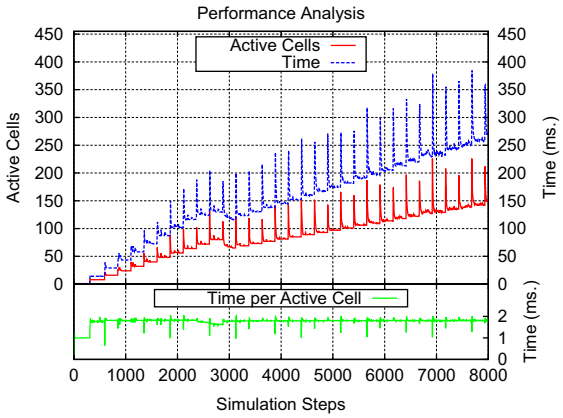


Fig. 13. Performance in a forest scenario.

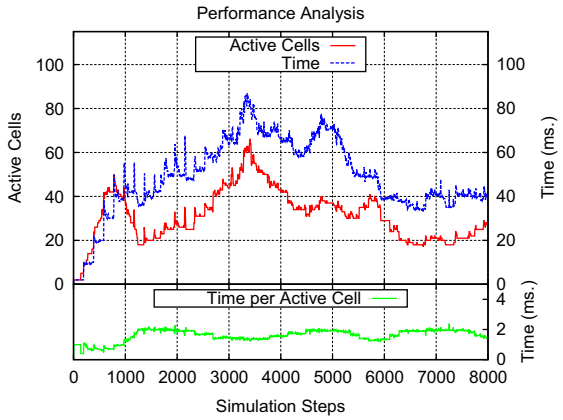


Fig. 14. Performance in a mixed scenario (forest and urban).

7. Validation

The validation of the presented algorithms has been performed in three steps. The first one addresses the utilisation of FARSITE software to validate the forest fires simulation. The second step compares with the simulation output obtained by Zhao [20] in an urban scenario. For the last step, we have contacted fire fighting experts to get their opinion about our simulation achievements.

7.1. Forest areas

FARSITE [2] is a well-known fire simulation application oriented to forest environments and its simulation results are usually utilised in scientific comparisons [21,22]. It is based on the Huygens principle of wave propagation to imitate the fire propagation in a highly realistic way by using complex calculations.

The field spatial resolution can be configured to match the field resolution used in this work, i.e., 3 × 3 m cells. However, FARSITE cannot be configured to set a simulation step of 1 s. The minimum selectable value is 1 min.

FARSITE calculates the fire spread as a collection of fire fronts, represented as contours. The humidity, the terrain aspect, the dead fuel models and other variables are used in the FARSITE to calculate the simulation.

We have used a section of the tutorial field included in the FARSITE installation for a qualitative comparison with the proposed algorithms. The size of this test area is 81 ha and it is sampled by 300×300 cells.

The simulation results after 1 h are similar on both cases (see Fig. 15). The real difference is in the simulation time. Our algorithms are much faster than the FARSITE simulation, even disabling some features in FARSITE (varying wind and dead fuel models).

7.2. Urban areas

Zhao [20] has researched the fires after earthquakes in urban environments. In his work, a real urban fire scenario is presented and we have used it to validate the behaviour of our algorithms. We have constructed a cell representation of one urban scenario analysed by Zhao. As the number of stories is not described, we have modelled the buildings with 3 stories. The roads and streets are considered non-combustibles, so the fire spreads by jumping over streets (Spotting Fires).

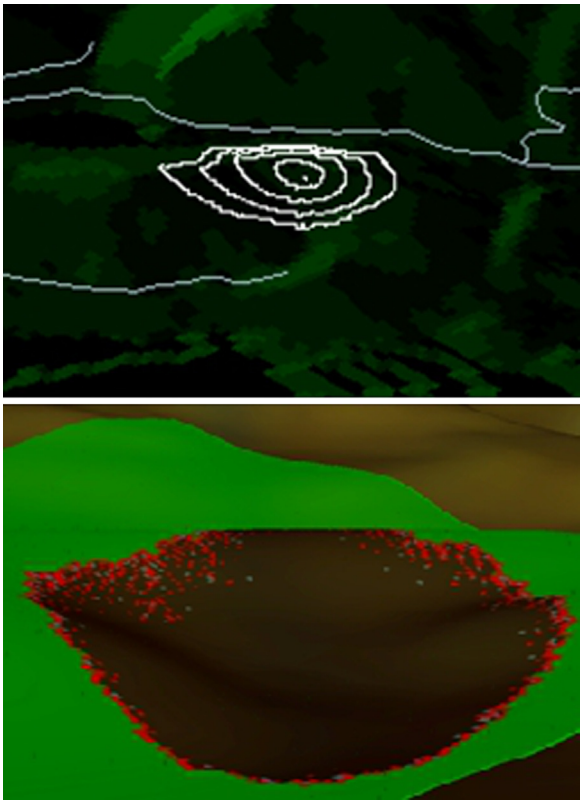


Fig. 15. Visual comparison between the results obtained in FARSITE (top) and the simulation results using the proposed algorithms (bottom).

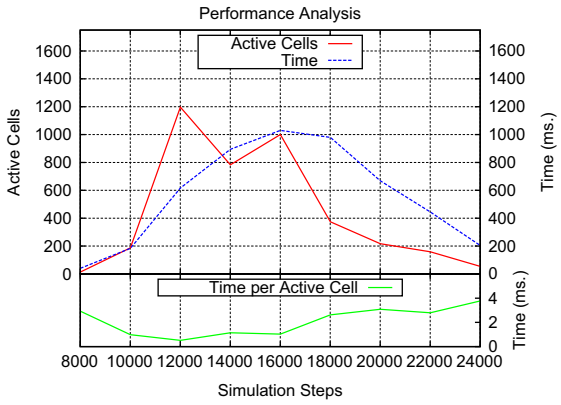


Fig. 17. The time spent per active cell is around 4 ms. The shown scenario is an urban scenario with buildings with 3 stories. The number of active cells varies depending on the fire spread in the buildings and between them.

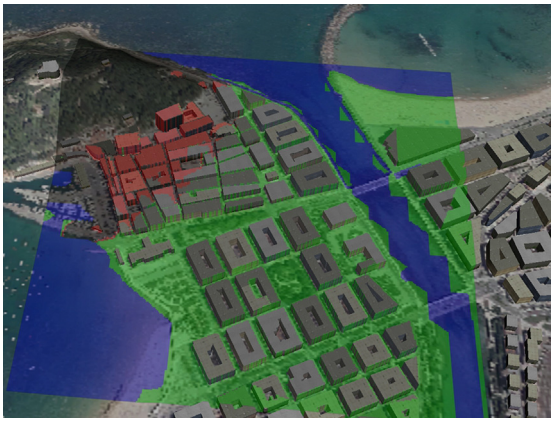


Fig. 18. Simulation of a theoretical mixed fire in San Sebastian (Spain). The terrain, the land use and the building information have been constructed from open data sources.

Fig. 16. Validation of urban algorithms. From left to right and top to bottom: (a) real status of the fire after 7 h; (b) Zhao's simulation after 7 h and (c) simulation output with our algorithms.

Fig. 16 shows the real fire (reported by Zhao), the Zhao's simulation [20] and our results. They correspond to 7 h after fire starts (our step is 25,200). The figure shows that simulation results are similar (spatially and temporally).

From the performance point of view, Fig. 17 shows that the simulation time per active cell is about 4 ms. The simulation time is below 1 s threshold, except in a brief period of time. That precise moment corresponds to an extreme situation, where almost 70% of the buildings are burning at the same time.

Fig. 18 shows another use case study. The sampled area is part of San Sebastian (Spain). It is a coast zone with a small hill near the old-town of the city.

The scenario contains an urban zone, composed of wooden buildings, surrounded by vegetation (including a small forest):

- The Digital Elevation Model (DEM) was constructed from open data sources.
- The land use (water, forest, vegetation, roads and buildings) was loaded as Shapefiles (SHP). As building footprints came from different providers, an intelligent combination of the available data sources was performed to create an improved scenario.
- The number of storeys of the buildings was not available. Thus, we provided a method to calculate the height of the buildings by using the difference between the DEM and the Digital Surface Model (DSM).

With the composed scenario, different fire situations can be simulated. Fig. 18 shows how a fire started in the vegetation zone and due to the wind conditions, it managed to spread to the urban area.

7.3. Validation with experts

We have collaborated with the *Centro de Jovellanos* [23] in the validation phase of the algorithms. *Centro de Jovellanos* provides training courses for different professionals such as fire fighters, brigades or civil protection. A selection of experts has tested the algorithms through a simulation tool, specifically designed for that purpose (see Fig. 19).

7.3.1. Validation of forest scenarios

The experts confirmed that the fire behaviour on the forest environment met their experience. They noticed how the wind and the slope modified the fire spread behaviour. They appreciated how the fire could jump over a river or road if the wind conditions were favourable.

In a second run of testing, the experts were provided with a virtual hose, so they could try to suppress a fire. Their experiences were satisfactory. However, in their opinion, the behaviour of the extinguishing agent was not correctly balanced. However, the *factors* of the extinguishing agents can be easily modified to match the expected behaviour.

7.3.2. Validation of urban scenarios

The experts explained that it is quite difficult to determine the correct behaviour of a big scale urban fire because it does not happen very often and there are few records. One of their main concerns was related to the modelling of the interior of the buildings, as it is important to determine how the fire would spread.

The experts verified that the suppression techniques with virtual hoses (the *WaterJet* method described in the algorithms) fulfill a great variety of suppression tactics from the exterior of the buildings.

8. Conclusions and future work

This work presents fire spread and extinguishment algorithms for forest and urban environments which can be used at interactive rates. This is a requirement that must be met by virtual training systems: fire fighting trainees have to throw extinguishing agents and get an immediate feedback of their actions. Although

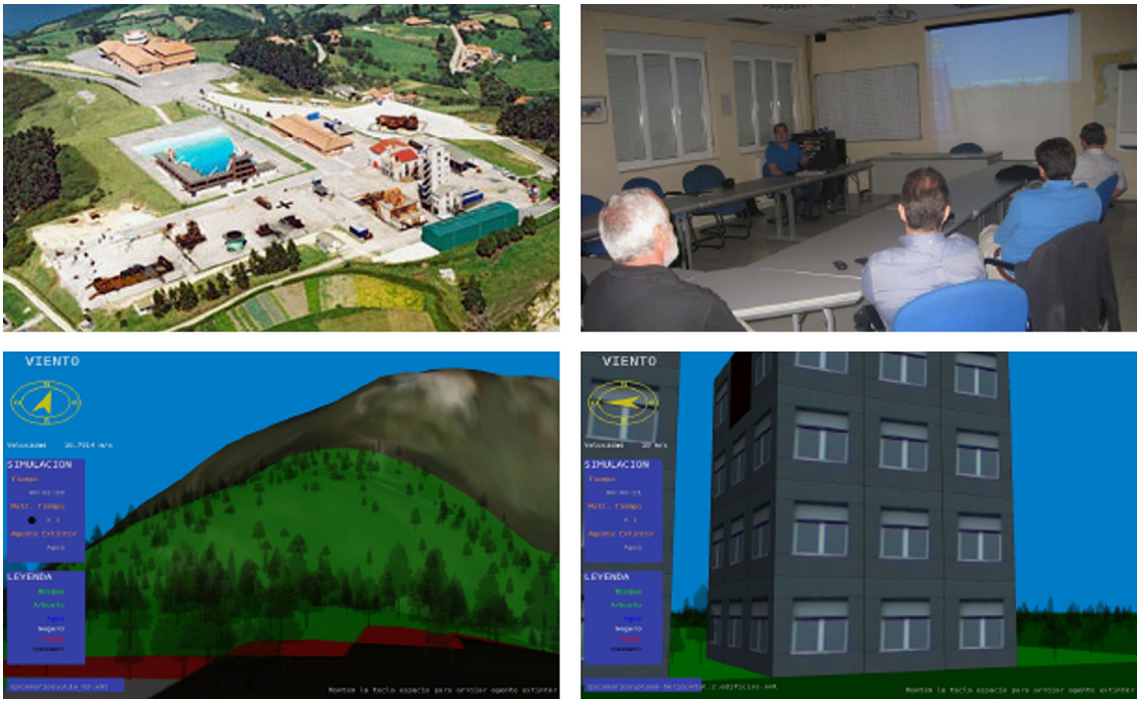
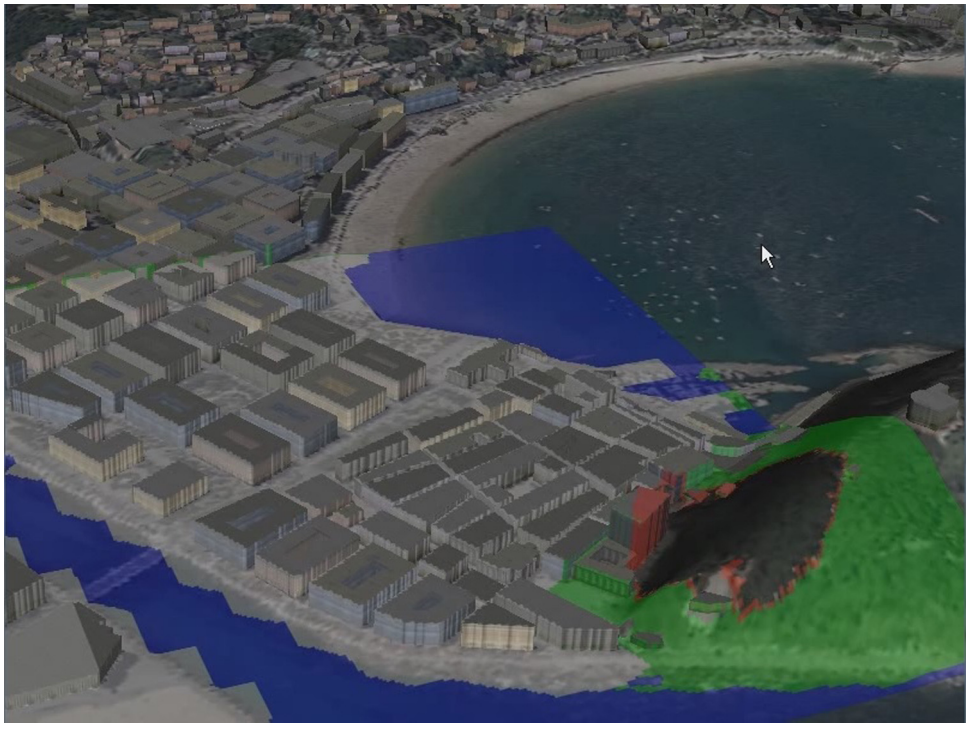


Fig. 19. Fire training facilities at *Centro Jovellanos* [23], validation session and two screenshots of the simulation tool.



Video S1. Main features of the presented fire spread algorithms. A video clip is available online. Supplementary material related to this article can be found online at doi:http://dx.doi.org/10.1016/10.1016/j.firesaf.2014.01.005

the algorithms have been simplified to match this requirement, they support variables and models that have a great influence on fire evolution. The analyses and validations show that the simplifications proposed keep proper fire spread behaviour. The paper shows numerical and graphical results. It also summarises assessments from experts, who state the consistency of the system behaviour with their experiences.

The algorithms support different types of vegetation and buildings. Barriers like roads, rivers or firebreaks can be jumped by the fire creating spotting fires (given the proper wind and slope conditions). Wind can be set at a local level (each 3×3 m cell) without any performance decay. Wind and slope have a significant influence on the fire spread behaviour.

Urban fires are seamlessly integrated in the field definition. The specific fire behaviour for buildings includes horizontal and vertical spread mechanisms. Furthermore, the fire can spread from building to building or from building to vegetation areas (spotting fires). *WaterJet* conceptualises the attack of urban fires with hoses from the exterior of the buildings.

The algorithms allow developers the design of elegant object oriented architectures which have as a central core a convenient definition of cell states and state transitions. These models, as previously published, allowed the integration of geoinformation and semantic architectures [24].

The addition of other variables not covered in this work (e.g. weather, season, temperature and relative humidity) or the integration of an external wind simulation module would enrich simulations, but the impact on the performance has to be evaluated. It is also important to use more detailed information, including the typology of buildings, land use and extinguishing agents to achieve more accurate simulations. Using GPU or GPGPU implementation techniques will bump up the performance as the algorithms are highly parallelizable.

The utilisation of the algorithms in the early stages of a real emergency can help in the decision making process. We have been working in some preliminary analyses [25].

Acknowledgements

This work was supported by COST Action TU0801 "*Semantic Enrichment of 3D City Models for Sustainable Urban Development*", and it was carried out in the context of project SIGEM, funded by the Spanish Industry Ministry through its *Avanza I + D Programme*. Dr. García-Alonso was supported by the Spanish MEC TIN2009-14380 and the Basque Government IT421-10.

Authors thanks Basque Country government for the *Open Data* initiative, which provided the geographic information needed to construct the scenarios.

Appendix A. Supplementary material

The following are the supplementary data to this article:
Video S1

References

[1] R.C. Rothermel, A mathematical model for predicting fire spread in wildland fuel, Technical Report, U.S. Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station, 1972.
[2] M.A. Finney, FARSITE: fire area simulator-model development and evaluation, Research Paper RMRS-RP-4, Ogden, UT: USDA Forest Service, Rocky Mountain Research Station, 1998, p. 47.
[3] R. Linn, J. Reisner, J.J. Colman, J. Winterkamp, Studying wildfire behavior using FIRETEC, Int. J. Wildland Fire 11 (2002) 233–246.
[4] F.J. Serón, D. Gutiérrez, J. Magallón, L. Ferragut, M.A. Asensio, The evolution of a wildland forest fire front, Vis. Comput. 21 (2005) 1–18.
[5] L. Ferragut, S. Monedero, M.I. Asensio, J. Ramírez, Scientific advances in fire modelling and its integration in a forest fire decision system, in: W. Press (Ed.), Modelling, Monitoring and Management of Forest Fire I. WIT Transactions on Ecology and the Environment, vol. 119, 2008, pp. 31–38.
[6] L. Ferragut, M.I. Asensio, S. Monedero, Modelling radiation and moisture content in fire spread, Commun. Numer. Methods Eng. 23 (2007) 819–833.
[7] D. Morvan, C. Hoffman, F. Rego, W. Mell, Numerical simulation of the interaction between two fire fronts in grassland and shrubland, Fire Saf. J. 46 (8) (2011) 469–479.

[8] Y. Dumond, Forest fire growth modelling with geographical information fusion, 11th International Conference on Information Fusion, 2008, pp. 1–6.
[9] G.L. Achtemeier, Rabbit rules. An application of Stephen Wolfram new kind of science to fire spread modeling, Technical Program of the Joint 2nd International Wildland Fire Ecology and Fire Management Congress and 5th Symposium on Fire and Forest Meteorology.
[10] S. Lee, R. Davidson, N. Ohnishi, C. Scawthorn, Fire following earthquake-reviewing the state-of-the-art of modeling, Earthq. Spectra 24 (4) (2008) 933–967.
[11] D.R. Weise, G.S. Biging, Effects of wind velocity and slope on flame properties, Can. J. For. Res 26 (1996) 1849–1858.
[12] H. Cheng, G.V. Hadjisophocleous, Dynamic modeling of fire spread in building, Fire Saf. J. 46 (4) (2011) 211–224.
[13] J. Stern-Gottfried, G. Rein, Travelling fires for structural design. Part II: design methodology, Fire Saf. J. 54 (0) (2012) 96–112.
[14] T. Iwami, Y. Ohmiya, Y. Hayashi, K. Kagiya, W. Takahashi, T. Naruse, Simulation of city fire, Fire Sci. Technol. 23 (2) (2004) 132–140.
[15] A. Ohgai, Y. Gohnai, S. Ikaruga, M. Murakami, K. Watanabe, Cellular automata modeling for fire spreading as a tool to aid community-based planning for disaster mitigation, in: J.P. Leeuwen, H.J.P. Timmermans (Eds.), Recent Advances in Design and Decision Support Systems in Architecture and Urban Planning, Springer, Netherlands, 2005, pp. 193–209.
[16] A. Moreno, A. Segura, A. Korchi, J. Posada, O. Otaegui, Interactive urban and forest fire simulation with extinguishment support, in: Advances in 3D Geo-Information Sciences, Lecture Notes in Geoinformation and Cartography, Springer, Berlin, Heidelberg, 2011, pp. 131–148.
[17] F. Morandini, X. Silvani, L. Rossi, P.A. Santoni, A. Simeoni, J.H. Balbi, J.L. Rossi, T. Marcelli, Fire spread experiment across mediterranean shrub: influence of wind on flame front properties, Fire Saf. J. 41 (3) (2006) 229–235.
[18] F.A. Albini, Estimating wildfire behavior and effects, General Technical Report INT-30. Ogden, Utah, Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station, 1976.
[19] J.H. Scott, R.E. Burgan, Standard fire behavior fuel models, a comprehensive set for use with Rothermel's surface fire spread model, General Technical Report rmrs-gtr-153, United States Department of Agriculture, June 2005.
[20] S. Zhao, GisFFE, an integrated software system for the dynamic simulation of fires following an earthquake based on GIS, Fire Saf. J. 45 (2010) 83–97.
[21] X. Yan, F. Gu, X. Hu, S. Guo, A dynamic data driven application system for wildfire spread simulation, in: Proceedings of the 2009 Winter Simulation Conference (WSC 2009), vol. 1–4, Winter Simulation Conference Proceedings, IEEE, 2009, pp. 2972–2979.
[22] G.A. Trunfio, D. D'Ambrosio, R. Rongo, W. Spataro, S. Di Gregorio, A new algorithm for simulating wildfire spread through cellular automata, ACM Trans. Model. Comput. Simul. 22 (1) (2011) 61–626.
[23] Centro de Seguridad Marítima Integral Jovellanos, ⟨http://www.centrojovellanos.com⟩, September 2013.
[24] A. Moreno, A. Segura, S. Zlatanova, J. Posada, A. García-Alonso, Benefit of the integration of semantic 3D models in a fire-fighting VR simulator, Appl. Geomat. 4 (3) (2012) 143–153, http://dx.doi.org/10.1007/s12518-012-0093-1.
[25] A. Moreno, A. Segura, S. Zlatanova, J. Posada, A. García-alonso, Introducing GIS-based simulation tools to support rapid response in wildland fire fighting, in: C. Brebbia, G. Perona (Eds.), Modelling, Monitoring and Management of Forest Fires III, WIT Transactions on Ecology and the Environment, WIT Press, 2012, pp. 163–174, http://dx.doi.org/10.2495/FIVA120141.