Discrete Event Simulation

Jean-Luc Falcone

May 21, 2021

Introduction to Discrete Events

Simple physics

- A point particle in an infinite 2D space, no forces:

$$x(t) = v_x t + x(0)$$
$$y(t) = v_y t + y(0)$$

- The trajectory can be accurately computed at each time in the future (or even in the past).
- For instance if the particle is at $t = 0$ at position $(4, 5)$ with $\mathbf{v} = (2, 0.9)$. Three minutes later ($t = 180$) the particle will be located at position $(364, 267)$.

Still simple physics ?

- Let's now assume that the same particle is in fact in a square 2D box with sides of 16 meters.
- Let's also assume perfectly elastic collisions:

$$v'_x = \begin{cases} -v_x & \text{if } x = 0 \text{ or } x = 16 \\ v_x & \text{otherwise.} \end{cases}$$

$$v'_y = \begin{cases} -v_y & \text{if } y = 0 \text{ or } y = 16 \\ v_y & \text{otherwise.} \end{cases}$$

- What will be the position of the particle after 3 minutes ?

A naive & brute force approach

- Move the particle successively by small increments of time Δt.
- At every time step, check for collisions and update velocities before position:

$$v_x(t+\Delta t) = \begin{cases} -v_x(t) & \text{if } x(t) \leq 0 \text{ or } x(t) \geq 16 \\ v_x(t) & \text{otherwise.} \end{cases}$$

$$v_y(t+\Delta t) = \begin{cases} -v_y(t) & \text{if } y(t) \leq 0 \text{ or } y(t) \geq 16 \\ v_y(t) & \text{otherwise.} \end{cases}$$

$$x(t+\Delta t) = v_x(t+\Delta t)\Delta t + x(t)$$

$$y(t+\Delta t) = v_y(t+\Delta t)\Delta t + y(t)$$

A smarter approach

- When will the particle collide with north boundary, assuming it is an infinite line:

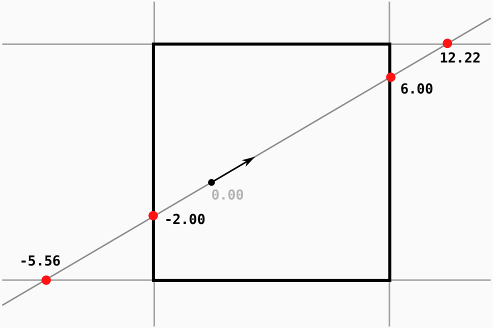
$$v_y t_\mathrm{N} + y(t) = L \quad \Leftrightarrow \quad t_\mathrm{N} = \frac{L - y(t)}{v_y}$$

- We can compute next collision time for all boundaries

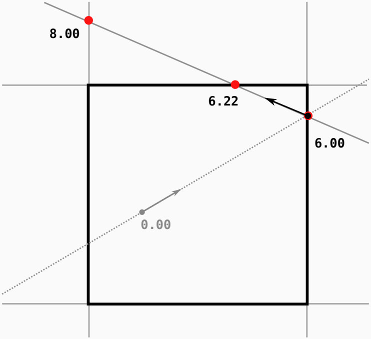
$$t_\mathrm{N} = \frac{L - y(t)}{v_y} = 12.22 \qquad t_\mathrm{E} = \frac{L - x(t)}{v_x} = 6.00$$

$$t_\mathrm{S} = \frac{-y(t)}{v_y} = -5.56 \qquad t_\mathrm{W} = \frac{-x(t)}{v_x} = -2.00$$

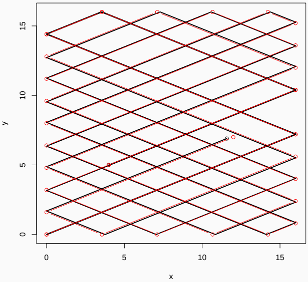
The first collision

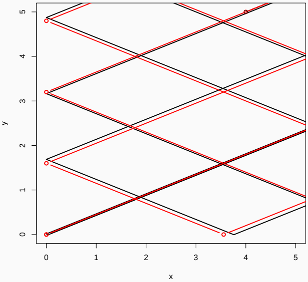


The second collision



And so on





Comparison

- The discrete version is both faster and accurate:

Approach ($t = 180$)	Iterations	Accuracy
Continuous ($\Delta t = 0.01$)	18'000	Error ≤ 4 %
Discrete	34	Exact

Definition of Discrete Events Simulations

Discrete Event Simulations (DES)

- Contrasts with *time-driven* simulations
- There exists timed events causing significant modifications to the system state
- The systems is simulated by jumping from one event to the next in time.

Requirements

- We can compute analytically the state of the system at any time between two events
- Each event takes place at a given time $t \in \mathbb{R}^+$
- Only a single event can occur at a given time

System description

- The system is composed of many entities i
- Each entity is described by its state at time t $(s_i(t))$
- We denote $S(t)$ the set $\{s_i(t)\}$ at time t

Events

- Each event j is associated with an action a_i
- Each action a_j is a function which modifies the state:

$$a_j : S \to S$$

- The event is the cause; the action is the effect

Events kind

- We will distinguish two kind of events:

  Endogeneous Consequences of the system evolution

  Exogeneous Originating from outside the system

- Exogeneous events can be seen as the system boundary or inlet conditions.
- Exogeneous events are often generated using a pseudo-random generator and an appropriate distribution function.

Terminology

Warning

- Discrete events systems time is continuous
- Continuous time model time is discrete

The DES algorithm (simplified)

Initialisation

```
t_current = t0
s_i = s_i(t_current)
```

Evolution

```
while not end_condition(t_current, s_i):
  events = f(s_i)     #compute all next events
  e_next = g(events) #Choose the closest in time
  t_next = e_next.t
  s_i = e_next.action( s_i ) #Execute the action
  t_current = t_next          #Jump to next time
```

End condition

- Depends of the considered problem
- For instance, we could stop the system if:
  - The system cannot generate more events
  - $t_{\text{next}} >= t_{\text{max}}$

Optimisation problems

Description: Windows

- Post office with n windows
- Each window w_i has a state:
  - `CLOSED`
  - `OPEN`
  - `BUSY`
- Each window has a daily schedule, describing when the window *opens* and when the window *closes*. For instance:

	w_1	w_2	w_3
08:30	`OPEN`	`OPEN`	`CLOSED`
09:30	`OPEN`	`CLOSED`	`CLOSED`
10:30	`CLOSED`	`OPEN`	`CLOSED`
11:30	`OPEN`	`OPEN`	`OPEN`
...	...	...	...

Description: Customers

- Each customer j is associated with the duration needed to process his/her request p_j.
- For a given simulation, we know in advance the arrival time and processing duration of each customer. For instance:

Arrival	Duration
08:30:00	5’
08:30:10	2’
08:31:00	10’
08:31:25	3’
...	...

Description: Queue

- When a customer j arrives, it looks for an `OPEN` window.
  - If found: the window is now `BUSY` for the duration p_j.
  - If not found, the customer waits in a FIFO queue.
- When a window becames `OPEN` the first consumer in queue goes to the window. The window will become `BUSY`.

Optimization questions

- How to organize windows schedule such as to minimize waiting line length ?
- How could we reduce the open window before the waiting line length comes above a given threshold ?
- Is it interesting to have a separate queue for light requests ?

State

To describe fully describe a state of this post office model, we need to keep track of:

- A vector W of length n with elements w_i equal to the state of the corresponding window.
- A FIFO queue of waiting customers (C), where each customer is represented by its processing duration p_j.

Events

Windows (endogeneous)

- `Open(t,i)`
- `Process(t,i)`
- `Close(t,i)`

Customers (exogeneous)

- `Arrival(t,p_j)`

Exogeneous events

- The exogeneous events are analoguous to *boundary conditions*
- Since they don't depend on the system state, they can be generated before the simulations starts.
- They are often generated at random using a distribution function obtained empirically.

Optimisation process

- The model parameter to be optimised are identified and fixed for each run
- The DES allows to compute an objective function
- The objective function is measured every time an event occurs.
- If the exogeneous event are random, several run will allow to estimate the objective function: Monte-Carlo method.

Implementation matters

Event queue

- All future events are inserted to a priority queue (Q), sorted by event time.
- Each event action a_j can also read and modify this queue:

$$a_j : (S \times Q) \to (S \times Q)$$

- An action could:
  - Insert one or several new events
  - Remove one or several existing events

Event queue

- For good performances choose an adequate datastructure
- My favorites (fast, fun to implement):
  - Calendar queue
  - Pairing heap

Initialisation

```
t_current = t0
Q = exogeneous()  #Add exogeneous events to the
S = initialState(t0)                    # queue
```

Evolution

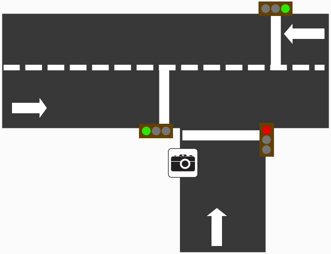
```
while not end_condition(t_current, S, Q):
  e_next = Q.next()        #Next event in Q
  t_next = e_next.time
  (S,Q) = e_next.action( S, Q ) #Execute action
  t_current = t_next       #Jump to next event
```

Parallel discrete events

- Because of causality parallelism is hard
- Three common strategies:
  - Optimistic
  - Pessimistic
  - Careless

Traffic intersection example

General overview



Model description

State

- Secondary street traffic light: $\texttt{f} \in \{\text{RED}, \text{GREEN}\}$
- Cars waiting in secondary stree: $\texttt{C} \in \mathbb{N}$

Event queue

- Priority queue of future events sorted by time of occurence

Events:

- $\texttt{t} \in \mathbb{R}^{+}$
- $\texttt{action} : (E \times Q) \to (E \times Q)$

Parameters

Traffic parameters

- The models needs as an input a list of car arrivals in the secondary street.
- This could be given by empirical measurement on an existing intersection.
- The car arrival could be drawn at random from an estimated distribution.

Traffic light parameters

- Latency to switch the traffic light from green to red: a
- Duration of green light per waiting car: b

Car arrival event

```
event CAR(t):
  def action():
    if C == 0 and f == "RED":
      Q.insert( R2G(t + a) )
    elif f == "GREEN":
      pass
    else:
      C = C + 1
```

Traffic light events

```
event R2G(t):
  def action():
    f = "GREEN"
    Q.insert( G2R( t + C*b ) )
    C = 0

event G2R(t):
  def action():
    f = "RED"
```

Evolution example

- Model Parameters: $a = 30$, $b = 10$

Time	Event	C	f	Queue
0		0	R	`CAR(10) CAR(25) CAR(35) CAR(60) CAR(75)`
10	`CAR`	1	R	`CAR(25) CAR(35) R2G(40) CAR(60) CAR(75)`
25	`CAR`	2	R	`CAR(35) R2G(40) CAR(60) CAR(75)`
35	`CAR`	3	R	`R2G(40) CAR(60) CAR(75)`
40	`R2G`	0	G	`CAR(60) G2R(70) CAR(75)`
60	`CAR`	0	G	`G2R(70) CAR(75)`
70	`G2R`	0	R	`CAR(75)`
75	`CAR`	1	R	`R2G(105)`
105	`R2G`	0	G	`G2R(115)`
115	`G2R`	0	R	ϵ

Complications

- Green light duration limit
- Taking into account a realistic passing tme for cars
- Pedestrian crosswalk
- Better traffic light synchronisation

Volcano ballistics

Eruption (night)

Volcano bombs (1)

Volcano bombs (2)

Base equations

- A bomb is a sphere with position $\mathbf{r}$, velocity $\mathbf{v}$, radius R and mass m:
- It follows an uniform acceleration motion:

$$\mathbf{r}(t) = \frac{1}{2}\mathbf{g}t^2 + \mathbf{v}(0)t + \mathbf{r}(0)$$

- To detect collisions between bombs:

$$\begin{aligned} d^2[B_1, B_2](t) &= \left(\mathbf{r}_1(t) - \mathbf{r}_2(t)\right)^2 \\ &= \left(\Delta\mathbf{v}t + \Delta\mathbf{r}\right)^2 \leq (R_1 + R_2)^2 \end{aligned}$$

Model simulation

- No analytic solution
- Continuous time solution is time consuming
- DES seems a good candidate

State

- For each bomb: $(\mathbf{v}, \mathbf{r}, R, m)$
- Airborne bombs are added to the list L_A
- Deposited bombs are added to the list L_D

Events

- We need three kind of events:
  - `ERUPTION( t, distribution )`
  - `COLLISION( t, b1, b2 )`
  - `GROUND( t, b1 )`
- Each events is related to zero, one or two bombs.

Event: Eruption

```
event ERUPTION(t, distribution):
  def action():
    for b in LA:
      jumpToTime( b, t )
    bs = distribution.generate()
    for b1 in bs:
      tD = depositionTime(b1)
      Q.insert( GROUND( t, b1 ) )
      for b2 in L_A:
         tC = collisionTime(b1,b2)
         Q.insert( COLLISION( tC, b1, b2 )
      La.append(b1)
```

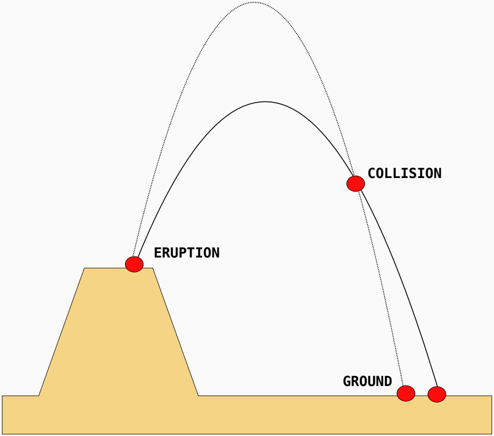
Event: Collision

```
event COLLISION(t, b1, b2):
  def action():
    for b in LA:
      jumpToTime( b, t )
    #remove all events related to b1 or b2
    cleanQueue( Q, b1, b2 )
    for b in [b1, b2]:
      tD = depositionTime(b)
      Q.insert( GROUND( t, b ) )
      for bb in L_A:
         tC = collisionTime( b, bb )
         Q.insert( COLLISION( tC, b, bb )
```

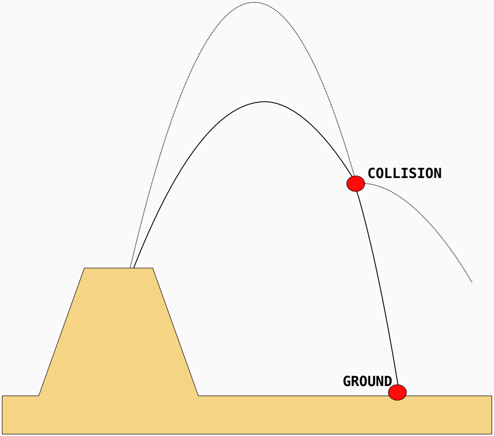
Event: Ground

```
event GROUND(t, b ):
  def action():
    for b in LA:
      jumpToTime( b, t )
    cleanQueue( Q, b )
    La.remove( b )
    Ld.append( b )
```

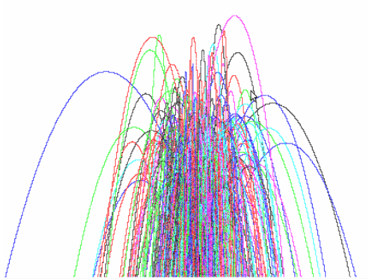
Example of evolution (1)

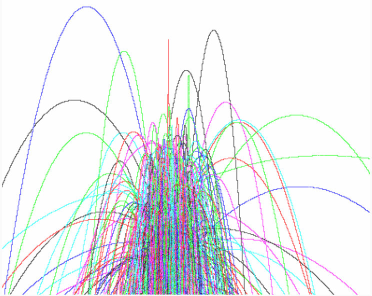


Example of evolution (2)

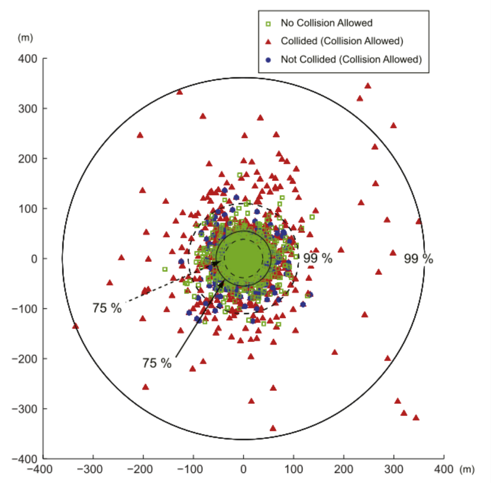


Qualitative results

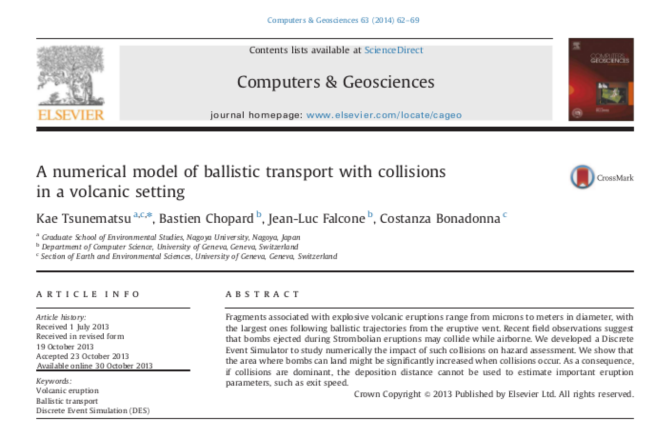




Quantitative results



Further information



Computers & Geosciences 63 (2014) 62–69

Contents lists available at ScienceDirect

Computers & Geosciences

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

ELSEVIER

A numerical model of ballistic transport with collisions in a volcanic setting

CrossMark

Kae Tsunematsu[a,c,*], Bastien Chopard[b], Jean-Luc Falcone[b], Costanza Bonadonna[c]

[a] Graduate School of Environmental Studies, Nagoya University, Nagoya, Japan
[b] Department of Computer Science, University of Geneva, Geneva, Switzerland
[c] Section of Earth and Environmental Sciences, University of Geneva, Geneva, Switzerland

ARTICLE INFO

Article history:
Received 1 July 2013
Received in revised form
19 October 2013
Accepted 23 October 2013
Available online 30 October 2013

Keywords:
Volcanic eruption
Ballistic transport
Discrete Event Simulation (DES)

ABSTRACT

Fragments associated with explosive volcanic eruptions range from microns to meters in diameter, with the largest ones following ballistic trajectories from the eruptive vent. Recent field observations suggest that bombs ejected during Strombolian eruptions may collide while airborne. We developed a Discrete Event Simulator to study numerically the impact of such collisions on hazard assessment. We show that the area where bombs can land might be significantly increased when collisions occur. As a consequence, if collisions are dominant, the deposition distance cannot be used to estimate important eruption parameters, such as exit speed.

Tsunematsu *et al.*, Computers Geosciences 63 (2014) 62–69

Image credits

- Stromboli 2 by “Victor” is licensed under CC BY-NC-ND 2.0
- bombs by “GEO M I” is licensed under CC BY-ND 2.0