Geoscientific
Model Development
Discussions

Open Access

CellLab-CTS 2015: a Python library for continuous-time stochastic cellular automaton modeling using Landlab

G. E. Tucker[1,2], D. E. J. Hobley[1,2], E. Hutton[3], N. M. Gasparini[4], E. Istanbulluoglu[5], J. M. Adams[4], and S. S. Nudurupati[5]

[1]Cooperative Institute for Research in Environmental Sciences (CIRES), University of Colorado, Boulder, USA
[2]Department of Geological Sciences, University of Colorado, Boulder, USA
[3]Community Surface Dynamics Modeling System (CSDMS), University of Colorado, Boulder, USA
[4]Department of Earth and Environmental Sciences, Tulane University, New Orleans, USA
[5]Department of Civil and Environmental Engineering, University of Washington, Seattle, USA

Correspondence to: G. E. Tucker (gtucker@colorado.edu)

Discussion Paper | Discussion Paper | Discussion Paper | Discussion Paper

GMDD

8, 9507–9552, 2015

Landlab cellular automata

G. E. Tucker et al.

Title Page

Abstract	Introduction
Conclusions	References
Tables	Figures

|◄ | ►|

◄ | ►

Back | Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion

Abstract

CellLab-CTS 2015 is a Python-language software library for creating two-dimensional, continuous-time stochastic (CTS) cellular automaton models. The model domain consists of a set of grid nodes, with each node assigned an integer state-code that represents its condition or composition. Adjacent pairs of nodes may undergo transitions to different states, according to a user-defined average transition rate. A model is created by writing a Python code that defines the possible states, the transitions, and the rates of those transitions. The code instantiates, initializes, and runs one of four object classes that represent different types of CTS model. CellLab-CTS provides the option of using either square or hexagonal grid cells. The software provides the ability to treat particular grid-node states as moving particles, and to track their position over time. Grid nodes may also be assigned user-defined properties, which the user can update after each transition through the use of a callback function. As a component of the Landlab modeling framework, CellLab-CTS models take advantage of a suite of Landlab's tools and capabilities, such as support for standardized input and output.

1 Introduction

The discovery of cellular automata in the 1940s (Von Neumann, 1951) laid the groundwork for a type of computational model that distinctly differs from numerical solutions to partial-differential equations (PDEs). For certain types of geoscientific problem, cellular automaton (CA) models and their relatives offer several advantages over numerical solutions to PDEs. In place of the continuum approximation, CA models operate on a discrete lattice, which makes them useful for natural systems that have an identifiable characteristic spatial scale. Unlike a numerical approximation, the solution to a deterministic CA is exact. In some cases, CA models are quite computationally efficient. Partly for this reason, cellular automata techniques are sometimes used to implement "reduced complexity" models, which deliberately omit aspects of the physics

of a system in an attempt to identify the essential underlying principles. Perhaps most important for geoscience applications, CA models are well-suited to systems with complex boundaries and interfaces that involve contact between different types of media. These properties have made CA-based approaches attractive for modeling a range of geophysical systems; examples include the dynamics of the core-mantle boundary (Narteau et al., 2001), eolian dunes (Anderson, 1990; Anderson and Bunas, 1993; Werner, 1995; Narteau et al., 2009; Zhang et al., 2010, 2012), hillslope morphology and evolution (Jyotsna and Haff, 1997; Tucker and Bradley, 2010), river channels (Murray and Paola, 1994; Nicholas, 2005; Coulthard and Van De Wiel, 2006; Jerolmack and Paola, 2007), drainage basins and networks (Chase, 1992; Coulthard et al., 1996, 2002, 2007), ecohydrology (Zhou et al., 2013; Caracciolo et al., 2014), and permafrost features (Kessler et al., 2001; Plug and Werner, 2002).

One interesting variant on CA is the continuous-time stochastic CA, in which cell transitions occur at randomly chosen time intervals rather than in discrete steps. These models are especially attractive for geoscience applications because their parameters represent rates that can be directly related to field and laboratory measurements, and because they avoid the need for a discrete-time approximation (Narteau et al., 2001, 2009; Rozier and Narteau, 2014). Here we describe a new modeling framework called CellLab-CTS, which is written in Python and built on the Landlab platform (Landlab Development Team, 2015). CellLab-CTS allows one to quickly build and explore two-dimensional, continuous-time stochastic CA models. A novel feature is the option of using either square or hexagonal cells. The aim of CellLab-CTS is to greatly simplify the process of creating, configuring, and exploring 2-D CA models. We provide a brief background on the theoretical framework, describe the algorithms and data structures used to implement CellLab-CTS, and present several examples that illustrate its versatility.

2   Background

Wolfram (1983) defines cellular automata as "simple mathematical idealizations of natural systems [that] consist of a lattice of discrete identical sites, each site taking on a finite set of...values." Like other types of mathematical model, CA models describe
5 natural systems in the form of symbolic logic. They are similar to partial differential equations in the sense that both can be written in compact form; in the case of a CA, however, the compact form is a set of algorithms ("rules") that evolve the numerical value of cell states or change cell attributes based on interactions among cells. In both cases, the compact form itself (the equation or the algorithm) often reveals little about
10 the dynamics of the system and its potential spatial outcomes and self-organization. To discover and visualize the system's behavior, one needs to perform numerical calculations. These calculations are often more computationally efficient, involving fewer parameters or degrees of freedom, than in the case of numerical solutions of comparable PDEs. The need for numerical calculations applies especially for complex physical
15 systems that are computationally irreducible, such that the physical state of the system at a certain time can only be predicted by simulating the evolution of states through time.

   In a classical CA, the cell values evolve over a sequence of discrete time steps on the basis of a set of deterministic rules that describe the nature of the system (e.g.,
20 Chopard and Droz, 1998). These rules describe sequential transitions in the state of each cell as a function of the other cells in its immediate neighborhood. Since CAs were first invented in the late 1940s, many variations on this basic concept have been developed and explored. For example, some use continuous (real) numbers instead of discrete lattice states (as in the example of lattice Boltzmann models e.g., Chen
25 and Doolen, 1998). The class of stochastic cellular automata use random rather than deterministic rules for updating, which allows one to explore ensembles of outcomes.

   Of particular interest here are continuous-time stochastic (CTS) cellular automaton models: those in which transitions between discrete cell states occur at random time

intervals. This technique was introduced into the geosciences by Narteau et al. (2001) in an application to the dynamics of the core-mantle boundary. The same approach was later used (in 3-D instead of 2-D) to study the growth of instabilities that leads to the formation of eolian dunes (Narteau et al., 2009). In the next section, we describe
5 the theory behind the CTS approach; the following sections describe how the concept is implemented in CellLab-CTS.

3 Cellular automata with stochastic, pairwise transitions

CellLab-CTS implements a two-dimensional, continuous-time stochastic cellular automaton with pairwise transitions (Narteau et al., 2001). As a simple example, consider
10 a model for the mixing of suspended sediment particles in a turbulent fluid with isotropic turbulence. As in a classical CA, the domain of interest is represented as a lattice of cells, each of which belongs to one of N discrete states. For suspended sediment, we have two states: a cell may be occupied by fluid, or by a sediment grain. The width of each cell is taken to be the characteristic diameter of a sediment grain. The lattice
15 represents a vertical cross-section through the fluid. Such a model might be initialized with a bed of sediment particles below a body of (initially) still fluid (Fig. 1).

The essence of the procedure is that for each pair (or "doublet"; Narteau et al., 2001) of adjacent cells, there is a certain probability that during a small interval of time dt, the states of one or both cells will change. For example, in a suspended-
20 particle model, when a sediment cell lies adjacent to a fluid cell, there is a certain probability that a turbulent eddy will cause the grain to move into the fluid cell, while its previous location is replaced by fluid. In other words, the grain and fluid switch places. Using Poisson process theory, we can describe transitions in terms of the probability distribution of time to the next transition:

25 $$p(\tau) = R_T \exp\left(-R_T \tau\right) \qquad (1)$$

|◀ | ▶|

◀ | ▶

Back | Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion

where τ is the time between transitions at a particular pair, and R_T is the average transition rate for a particular transition type (with dimensions of $1/T$). The reciprocal of the transition rate, $\bar{\tau} = 1/R_T$, is the average waiting time between transitions of that type. Once the cell size, δ, is specified, one obtains a mean transition velocity: $V_T = \delta/\bar{\tau}$.

The transition probability or rate depends on the states of the two cells. We will refer to a particular pairing of cell types as the pair state. The number of pair states depends on the number of cell states, and on whether spatial orientation matters. For example, if our turbulence is isotropic and the particles are neutrally buoyant, the transition probability for a fluid-plus-particle pair would be independent of orientation. In that case, there are $N(N+1)/2 = 3$ unique pair states: fluid-fluid, fluid-grain, and grain-grain (Fig. 1). On the other hand, if the particles are denser than the fluid, orientation matters because the downward transition rate will be greater than the upward or lateral rates (Fig. 2). When direction matters, we have MN^2 cell-pair states, where M is the number of orientations: two for a raster grid (vertical and horizontal), and three for a triagonal lattice with hexagonal cells. In the suspended-sediment example, $N = 2$; a square grid implies $M = 2$, so that there are eight cell-pair states (the six shown in Fig. 2 plus fluid-fluid and grain-grain pairs).

Note that it is possible, and often likely, to have a particular node (representing, say, a grain) scheduled to undergo more than one transition. For example, consider a "grain" node in the suspended-sediment model. That node belongs to four different pairs. If the grain is surrounded by fluid nodes, then each of those pairs will have the potential to undergo a transition in which the grain and fluid switch places. When such a transition occurs at one of the four pairs, it invalidates the other three pair transitions. Below, we explain how this common situation is handled.

The suspended-sediment examples also illustrate how it is possible to scale a CTS model. Here, we have chosen the cell size as the diameter of the particles (1 mm), and the time scale as 1 s. For the neutrally buoyant case, the transition rate is set to equal a characteristic velocity perturbation of $1\,\text{cm}\,\text{s}^{-1} = 10\,\text{cells}\,\text{s}^{-1}$. For the denser-than-

fluid case, the transition probabilities for upward and downward motion are decreased or increased, respectively, by a factor $\epsilon = w/2\delta$, where $w = 0.0011\,\mathrm{m\,s^{-1}}$ is settling velocity and $\delta = 0.001\,\mathrm{m}$ is cell width. Thus, the scales and transition rates are not arbitrary but have a direct physical meaning. For more on this point, see Narteau et al. (2009) and Rozier and Narteau (2014).

4 Algorithms, implementation, and capabilities

CellLab-CTS is built on Landlab, a Python-language library for constructing and exploring two-dimensional grid-based models (Landlab Development Team, 2015), which imparts several unique features and capabilities. Because of Landlab's support for multiple grid types, models built in CellLab-CTS can use either of the two grid types that are commonly used in cellular automata: a raster grid with square cells, or a triagonal grid with hexagonal cells (Fig. 3). Landlab's grid design lends itself naturally to pairwise cellular automata because the data structures include links: directed line segments that represent the connections between adjacent cell pairs.

4.1 Landlab's grid design

One of Landlab's unique features is the ability to create any of a variety of grid types, including raster (square cells), rectilinear (rectangular cells), triagonal (hexagonal cells), Delaunay-Voronoi (Voronoi polygon cells), and radial (a special class of Delaunay-Voronoi in which nodes are arranged in concentric circles). Each grid type uses the same "flat" data structure, in which grid elements are listed sequentially in one-dimensional arrays. For CellLab-CTS, this means that one can implement CA models using either a raster or hexagonal grid.

To understand how CellLab-CTS works, it is helpful to know a bit about Landlab's grid composition and data structures. Each grid contains a set of nodes, which are points in (x, y) space (Fig. 3). Each adjacent pair of nodes is connected by a link, which is an

Full Screen / Esc

Printer-friendly Version

Interactive Discussion

oriented line segment. Every link connects a tail node to a head node, with the implied direction being from the tail to the head. Each node in the grid interior – that is, every node except those along the perimeter of the grid – sits inside a polygon known as a cell. In the case of CellLab-CTS, cells are either squares or hexagons. Every cell face is crossed by a link. (Note that CellLab-CTS actually operates on arrays of nodes rather than cells, so that the outer perimeter of "cell-less" nodes may be included as a boundary condition; for this reason, the internal documentation refers to nodes and node pairs rather than cells and cell pairs).

To facilitate boundary-condition handling, nodes come in two flavors: core nodes and boundary nodes (Fig. 4). Core nodes are those that constitute the computational domain. When a Landlab grid is created, the default configuration has all interior nodes flagged as core nodes, and all perimeter nodes flagged as boundary nodes. For modeling irregular domains (such as a watershed within a rectangular DEM), one can set up a grid to have boundary nodes in the interior as well as along the perimeter.

In configuring a two-dimensional model, one often needs to specify boundary conditions that are open to flow (such as the downstream end of a stream channel), and those that are closed to flow (such as the wall of a closed vessel). To facilitate such boundary-condition handling, boundary nodes in a Landlab grid are flagged as either open or closed (Fig. 4). Once boundary types have been assigned, the grid's links are then flagged as active or inactive according to the following criteria: (1) a link is active if it connects two core nodes, or if it connects a core node and an open boundary node; (2) a link is inactive if either or both of its nodes is a closed boundary, or if both are open boundaries (Fig. 4). This classification allows a model developer to perform calculations only on active links, thus effectively treating the inactive links (and the corresponding cell faces) as walls across which there is no flow of mass, momentum, or energy.

4.2 Node and node-pair (link) states

In order to implement a cellular automaton model, CellLab-CTS assigns an integer node state to each node in the grid. These values are encoded in a one-dimensional array of integers called the node-state grid. The number of possible node states and

5 their transition behaviors are determined by the model developer, as explained below. For example, in the example of turbulent suspension (Figs. 1 and 2), there are two possible node states: 0 and 1, representing fluid and a solid particle, respectively.

As discussed above, a CellLab-CTS model is based on transitions from one pair of node states to another pair. Each unique pair is referred to as a pair state or a link

10 state. To implement pairwise transitions, CellLab-CTS takes advantage of the fact that a Landlab grid includes a set of links connecting pairs of neighboring nodes. We can take advantage of this fact by creating a data structure in which each active link in the grid is assigned a pair code. The pair code is a single integer value that represents the states of the two adjacent nodes, and possibly also the spatial orientation of the pair.

15 To understand how pair coding works, we need to look more closely at orientation. In some applications, spatial orientation does not matter. For example, in the isotropic suspension model (Fig. 1), the transition rate for a solid-grain pair is the same regardless of whether the pair in question is horizontal (aligned with the x axis) or vertical (aligned with the y axis). In other applications, orientation does matter. For example,

20 when gravitational settling is added to the turbulent suspension model, the transition rates differ for horizontal and vertical pairs (Fig. 2). Moreover, the transition rate for a vertical pair also depends on whether the solid particle lies above the fluid particle, or vice versa.

To accommodate these difference, CellLab-CTS allows users to create either an

25 oriented or a non-oriented model. In a non-oriented model, the sequence of node-state codes associated with the nodes of a given link, say 0 at the link's tail node and 1 at its head node, is treated the same regardless of whether the link is vertical, horizontal, or (in the case of a hex grid) at an angle of +30° or −30°. For a non-oriented

model, whether raster or hex, there are N^2 unique link states, where N is the number of possible node states. For example, the isotropic turbulent suspension model (Fig. 1), which is an example of a non-oriented (raster) model, there are just four link states: (1) two adjacent fluid cells, (2) fluid cell at the link's tail and a solid cell at the link's head,

5 (3) a solid cell at the link's tail and a fluid cell at its head, (4) two adjacent solid particles. Note that the order from tail node to head node still matters; the pair $0 \rightarrow 1$ is different from the pair $1 \rightarrow 0$.

    By contrast, an oriented model treats cell pairs in different orientations as different pair-states. In a raster grid, there are two possible orientations: horizontal and vertical.

10 In a hex grid, there are three. In our examples, these three hex-grid orientations are vertical, angling up ($+30°$), and angling down ($-30°$) (Fig. 5) (note that one can rotate this so that one of the axes is horizontal rather than vertical, but in either case there are still three orientations). Each of these is given a separate pair-state code, indicating that the transition type and rate may be different depending on orientation. For exam-

15 ple, in an oriented raster, the pair $0 \rightarrow 1$ has different codes for horizontal and vertical orientation. An oriented raster has $2N^2$ link states, whereas an oriented hex has $3N^2$ link states.

4.3 Transitions and the event queue

Unlike a traditional discrete-time CA, CellLab-CTS does not use time steps. Instead, we

20 iterate through a sequence of pair transitions, or events. As noted earlier, the time intervals between successive transition events at a particular location are stochastic, with an exponential probability distribution (Eq. 1). At the start of a run, every node pair (i.e., every link) is assigned a transition time. These transition-event times are generated at random from an exponential probability distribution, the mean of which is $\bar{\tau}_i = 1/R_{\text{T}i}$,

25 where i indicates the particular type of transition involved. If a particular node-pair is of a type that has more than one possible transition, event times are generated for each transition type, and the soonest is selected and assigned. If a given node pair is of

a type that has no transitions (such as the fluid-fluid and grain-grain pairs in Fig. 1), it is assigned an arbitrarily large transition time that is longer than the duration of the run.

Once the initial event times are assigned, we need to iterate through them in chronological order. In order to place events in the correct time sequence, we adopt an approach that is commonly used in other types of discrete-event simulation (e.g., Karimabadi et al., 2005; Omelchenko and Karimabadi, 2006, 2007), in which future events are recorded in a queue that sorts them according to time of occurrence. When a new event needs to be scheduled, we create an event object, which stores the location of the transition, the time at which it is scheduled to occur, and the new node states. This event object is then placed in the event queue: a data structure that contains all scheduled future transitions (Fig. 6). The event queue is implemented as a heap, using the Python `heapq` library. Events in the event queue are automatically and efficiently sorted such that the event with the smallest value of transition time – that is, the next one to occur – is always at the top. At the same time, we also record the transition time in a separate array that contains the transition times for every link (Fig. 6). Recording the transition times in two different locations allows us to handle the common case in which a scheduled transition becomes invalid because the state of one or both cells has changed. The easiest way to understand how this works is to examine the algorithm for implementing pair transitions, which we turn to next.

4.4 Algorithm for pair transitions

At the beginning of a CellLab-CTS simulation, we set up the initial array of node states. We then loop over all active links, assigning to each one the corresponding pair code. The pair code, L_i, for a given link i is calculated from the state of each node and the link's orientation:

$$L_i = O_i N^2 + T_i N + H_i, \tag{2}$$

where O_i is the link's orientation code (Fig. 5), T_i is the state of the tail node, H_i state of the head node, and N is the number of potential node states in the model. Recall

that a particular pair state may have zero, one, or more than one possible transition. For those pair states that have only possible transition, the transition time is selected at random from an exponential distribution with the appropriate rate parameter, and an event object is created and pushed to the event queue. When a given pair state is associated with two or more possible transitions, transition times are drawn at random for each of the potential transitions. The soonest of these is then entered into an event object and pushed to the event queue.

A scheduled event at one pair can become obsolete if one or both nodes changes state as a result of a transition in another pair to which the node is connected. For instance, consider a particle (state 1) surrounded by four fluid nodes (state 0) in the suspended-sediment example. The particle belongs to four different pairs: it is connected to the nodes above, below, right, and left. Each of these pairs will have a transition scheduled in which the solid and fluid states switch places, simulating motion of the grain. When a transition occurs at one of the four pairs, the other three scheduled transitions immediately become invalid, and their scheduled transitions should be ignored when they are popped from the event queue. To handle this situation, whenever an event is scheduled, its transition time is also recorded separately in the transition-time array (as is done in the discrete-event algorithms of Karimabadi et al., 2005; Omelchenko and Karimabadi, 2006, and Omelchenko and Karimabadi, 2007). Then, each time an event is popped from the event queue, it is executed only if its transition time matches the entry in the transition-time array.

The event-loop algorithm is illustrated in Algorithm 1. Note that each event object E includes a time of occurrence (E.time), the ID number of the link at which the event occurs (E.link), and the new link state to which the pair transitions (E.xn_to).

The algorithm for a transition event is illustrated in Algorithm 2. When a transition occurs, the state of one or both nodes will change. This will invalidate any scheduled transitions at the affected node(s). Therefore, the pair-state codes for each pair attached to the transitioning nodes must be updated, and new events generated and scheduled.

Full Screen / Esc

Printer-friendly Version

Interactive Discussion

Algorithm 1 Event loop

$T_r \leftarrow$ duration for run
$t \leftarrow 0$
while $t < T_r$ and event queue is not empty **do**
　$E \leftarrow$ pop next event from event queue
　if E.time matches the event time for this link recorded in the transition-time array **then**
　　Process the transition event, schedule the next event, and update surrounding pairs
　end if
　$t \leftarrow E$.time
end while

Algorithm 2 Processing a transition event

Update the states of the two nodes attached to the link
Update the link's pair code
Generate the next event for this link (if any) and push it to the event queue
for N = each of the link's two nodes (tail and head) **do**
　if the state of N has changed as a result of this transition **then**
　　for all other active links L connected to N **do**
　　　Update the state-code for L
　　　Generate the next event for L and push it to the event queue
　　　Update the entry for L in the transition-time array
　　end for
　end if
end for

Full Screen / Esc

Printer-friendly Version

Interactive Discussion

4.5 Class hierarchy and data structures

CellLab-CTS uses Python classes to implement four sub-types of CA model. The class inheritance structure is quite simple (Fig. 7). The base class, CellLabCTSModel, handles most of the primary data structures. The simulation grid is represented us-
ing a Landlab ModelGrid object (either a RasterModelGrid or HexModelGrid). The `node_state` array contains the state-code for every node (recall that Landlab nodes represent cells in the model; Landlab reserves the term "cell" for polygons that surround interior nodes).

Information about the relationship between link states and node states is contained
in the the `node_pair` list. When a transition occurs, this list is used to look up the new states of the two nodes given the new state of the link. The list is indexed by the pair-state codes, and each entry is a three-element tuple: (T, H, O), where T is the state of the tail node, H is the state of the head node, and O is the orientation of the link. For example, the configuration represented by pair-state number three would be found at
`node_pair[3]`.

As discussed previously, transition events are recorded in a heap known as the event queue. The time of transition for each pair (link) is also recorded in the `next_update` array (Fig. 6).

For models in which pair orientation matters, we need to keep track of the orienta-
tion code for each pair. This is done using `link_orientation`, an array of integer orientation codes (as defined in Fig. 5) whose length is equal to the number of links in the grid. The state of each pair is encoded in the `link_state` array. Finally, information about pair-state transitions is encoded in three arrays. We encode the number of potential transitions for each pair state in the array `n_xn`. For every transition type,
we need to record the new pair state and the rate; these two pieces of information are recorded in `xn_to` and `xn_rate`, respectively. Both are two-dimensional arrays with dimensions equal to the number of unique pair states and the maximum number of transitions for any given pair state.

4.6 Tracking properties associated with moving particles

The suspended-particle simulations (Figs. 1 and 2) are examples of models in which certain node states represent moving particles. For such models, it is often desirable to keep track of various properties associated with these particles. For example, one might wish to keep track of the original position of a particle, or calculate a position-dependent accumulation or loss of a property such as cosmogenic nuclide concentration, luminescence, or chemical composition. CellLab-CTS provides the ability to define and track user-defined data that are associated with certain node states, which are treated as mobile particles.

To implement property tracking, the user defines an array or list of properties that are assigned to nodes in their initial locations. The properties themselves may be of any data type; for example, for a simple scalar property, one might use floats, whereas for a collection of properties, user-defined objects containing multiple data items could be assigned. CellLab-CTS then creates an array that contains, for each node, the index in the user-defined property array/list that corresponds to that node. To handle movement of particles, each transition includes a flag indicating whether the transition in question involves an exchange of properties between the two nodes in the pair. For example, the transitions in the turbulent suspension model (Figs. 1, 2) represent particle motion, and would therefore be flagged as involving an exchange of properties between the node pairs. For each transition involving such an exchange, the simulation keeps track of the location of the properties in question.

Updating of properties is handled by the CellLab-CTS user, and it involves the use of a callback function. If a particular transition type involves particle motion, and one or more user-defined properties of particles evolves in time, the user would write a function to update these properties. The function arguments are: a CellLabCTSModel object (i.e., the instance of one of the four subclasses listed in Fig. 7), the IDs of the two nodes involved in the transition, and the time at which the transition occurs. This function is then passed as an optional argument when the transition in question is set up at

the beginning of a run. Then, whenever a transition of that type takes place, CellLab-CTS automatically calls the user's function. This use of a callback function gives the user flexibility to implement any kind of updating of properties during each transition event.

5　　As an example of user-defined property updating, imagine a version of the suspended-sediment model (Fig. 2) in which the grains are quartz sand (instead of tea leaves), and possess the property of luminescence. When quartz grains are exposed to background ionizing radiation in the soil, electrons gradually become displaced from their rest states, and become "trapped" within defects in the crystal lattice. When the
10　grain is exposed to light, the trapped electrons are released, returning to their rest states and giving off a faint glow in the process. This phenomenon, known as optically stimulated luminescence (OSL), is commonly used in geologic dating applications (Rhodes, 2011).

　　Imagine then that our grains begin with a certain luminescence signal L, which we
15　will assume is initially uniform for all grains. Imagine also that a light is positioned above the container, but that the fluid is partly opaque, so that the light intensity attenuates with depth below the surface. For quartz OSL, the rate of signal loss (bleaching) can be approximated by

$$\frac{\mathrm{d}L}{\mathrm{d}t} = -\frac{L}{T(z)},\tag{3}$$

20　where t is time and $T(z)$ is an effective timescale for bleaching that depends on the incoming photon flux and a material-dependent bleachability parameter (both integrated over the light spectrum). Because the photon flux attenuates with depth in the fluid column, z, according to the Beer–Lambert law, the bleaching time scale grows with depth below the fluid surface:

25　$$T(z) = T_0 \exp\left(z/z_*\right)\tag{4}$$

where z_* is the attenuation length scale, which depends on the fluid opacity (note that we ignore scattering here). For our example, we will use $T_0 = 2.42\,\mathrm{s}$ (Bailey and Arnold,

Full Screen / Esc

Printer-friendly Version

Interactive Discussion

2006) and $z_* = 0.025$ m. The latter represents quite opaque fluid (think tea with milk), and is used here simply to create a strong (~ 50x) variation in bleaching rate through the depth of our 10 cm container. Grains that diffuse toward the top of the container will therefore experience a much higher bleaching rate than those that settle toward the bottom.

To implement a bleaching model, the user's code would define and initialize an array containing the luminescence signal for each node. It would also define a simple call-back function. The callback function detects whether the state of either or both nodes represents a particle. If so, the corresponding entry in the user's luminescence array is updated by extrapolating equations 3 and 4:

$$L_i \leftarrow L_i \exp\left(-\frac{t - t_l}{T_0 e^{z/z_*}}\right) \tag{5}$$

where t is the current time and t_l is the last time the luminescence at node i was updated (which the callback function also tracks). The turbulent suspension model with bleaching is illustrated in Fig. 8. As one might expect, the rapid attenuation of light creates a strong vertical gradient in the degree of bleaching, with some dispersion that reflects turbulent mixing. Although this particular example is somewhat unrealistic in that light attenuation is treated as independent of sediment concentration (in other words, light passes equally through fluid and grains), the example illustrates the ability to treat certain cell states as moving particles, to track their movement, and to associate each particle with one or more properties.

5 Other examples

Three additional examples serve to illustrate the diversity of applications that can be written using CellLab-CTS. These applications span the fields of geomorphology (chemical weathering of crystalline rock), epidemiology (a Susceptible-Infectious-Recovered model of disease spread), and granular mechanics (a Lattice Grain Model).

5.1 Weathering of fractured rocks

One of the current frontiers in geomorphology and soil science lies in understanding the transformation of rock to soil. One-dimensional reactive transport models have been used to study the time evolution of a weathering front in homogeneous, unfractured rock (e.g., Lebedeva et al., 2007, 2010; Maher, 2010). In crystalline rocks, however, it is often observed that chemical alteration of the original rock takes place primarily along fracture planes, which serve as conduits for water, oxygen, and reactive aqueous elements such as hydrogen ions (Pandey and Rajaram, 2014). The model shown in Fig. 9 implements a simple hypothesis for the transformation of parent rock into saprolite (material that has been chemically altered by not disaggregated). Nodes in the model represent mineral grains; a typical diameter of such grains in nature might be ~ 3 mm. The model begins with a network of fractures, each initially one grain wide. The rules representing hydrology and geochemistry are deliberately simplified for the sake of illustrating a single-transition CellLab-CTS model: any rock-saprolite pair has a fixed probability per unit time of transforming into a saprolite-saprolite pair. This example uses the RasterCTS class.

The model domain forms a set of fracture-bounded blocks of rock, which weather inward over time from their perimeters. The effective probability of weathering of a grain depends on its local geometry: a grain exposed on only one side has a lower probability of weathering within a given time period than one exposed on all four sides, simply because of the combined probabilities in the latter. This simple geometric principle leads to a gradual rounding of the blocks as they weather.

The code to implement this model is very simple, with only two transition rules: the pair *saprolite (1)→rock (0)* transitions to *rock (1)→rock (1)*, and the pair *rock (0)→saprolite (1)* also transitions to *rock (1)→rock (1)*. Although this example uses a raster grid, one could instead use a hex grid (using the class HexCTS), which might provide a more faithful representation of the geometry of packed crystals. One could also of course take a more sophisticated approach to the hydrology and geochem-

istry, for example by introducing chemically saturated and unsaturated fluid states (cf. Narteau et al., 2001).

5.2 Susceptible-infectious-recovered model

The Susceptible-Infectious-Recovered (SIR) concept is a classic mathematical model in epidemiology. The simplest form of the SIR model represents a population as having three compartments: those who are infected, those who have not yet been infected and are therefore susceptible to the disease, and those who have recovered and are now immune (Hethcote, 2000). Figure 10 illustrates an implementation of the SIR model as a continuous-time stochastic cellular automaton, using a hex grid. Each node has one of three states: susceptible (gray), infectious (black), and recovered (white). Infection and recovery are modeled as stochastic processes. An infected node has a user-specified transition rate to recovery (probability per unit time of recovering), which translates into an exponential probability distribution of recovery times with mean $\bar{\tau}_r$ (Eq. 1). When a susceptible node lies adjacent to an infectious node, there is a specified infection rate (probability per unit time that the susceptible node will become infected). Again, this translates into an exponential probability distribution of time to infection, with mean $\bar{\tau}_I$. Thus, for any adjacent susceptible-infectious pair, there is a race against time: will the infected node recover before passing on the infection to its neighbor? The outcome depends on the ratio of infection to recovery rates. When the ratio is modest, an initial disease cluster spreads relatively slowly, and is likely to die out before spreading very far (Fig. 10, top row). When the ratio is higher, disease is likely to spread throughout the population, leaving few individuals untouched (Fig. 10, bottom row).

5.3 Lattice grain model

Granular-flow phenomena are ubiquitous in nature. Examples include landslides, debris flows, talus-pile formation, and pyroclastic flows, among others. A variety of cellular automata approaches have been used to model granular flows (Baxter and Behringer,

1990, 1991; Fitt and Wilmott, 1992; Osinov, 1994; Kozicki and Tejchman, 2005; Jasti and Higgs, 2006; LaMarche et al., 2007; Jasti and Higgs III, 2010). Among these are a family of models known as lattice-grain models (LGrM), which are based on the well-known lattice-gas model in fluid dynamics but with additional rules for gravity and friction (Gutt and Haff, 1990; Peng and Herrmann, 1994; Alonso and Herrmann, 1996; Károlyi and Kertész, 1998, 1999; Károlyi et al., 1998; Martinez and Masson, 1998; Désérable, 2002; Cottenceau and Désérable, 2010; Désérable et al., 2011). Here we describe and illustrate a continuous-time stochastic version of a lattice-grain model.

In the classic lattice-gas model, a fluid is represented as a set of particles on a regular lattice (Frisch et al., 1986; d'Humieres et al., 1986; Rothman and Zaleski, 2004). Each particle is assumed to have unit mass and speed, and each is assigned a direction of motion, which may be any of the lattice's cardinal directions. On a hexagonal lattice (the most common for lattice-gas models), there are therefore six possible directions. In addition, some models also include stationary particles. Each iteration of a lattice-gas model has two steps: a movement step, in which each particle moves one unit in its given direction, and a collision step, in which collisions between particles are resolved by changing particle directions (as a representation of collision and rebound between perfectly elastic particles).

A typical lattice-grain model starts with these basic rules but with modifications. Each cell may be occupied by only one grain (as opposed to one for each possible direction in lattice-gas models). Because grains are not perfectly elastic, collisions may result in a loss of momentum, with a specified probability. Finally, gravity is represented by applying a certain probability for a particle to alter its direction and/or velocity (for those models that allow varying velocity among particles).

The stochastic, pair-wise transition model of CellLab-CTS can be used to construct versions of both a lattice-gas and a lattice-grain model; the latter is simply a version of the former that adds rules for gravity and friction. Here we present examples of both types of model. The examples are implemented on a hex grid with a vertical axis and

two other axes at 60° from vertical. There are nine possible node states, corresponding to the six directions of motion, an empty state, a resting state, and a wall state (Table 1).

The motion and collision rules for a pair-wise CTS lattice gas model are illustrated in Fig. 11. Because the model is stochastic, with binary transitions, the rule set is somewhat different from that of a traditional deterministic lattice-gas model (e.g., Chopard and Droz, 1998). There is no need to deal with three-way collisions, for example, and we allow only one particle to occupy each node. Furthermore, the stochastic nature of transitions means that particles effectively have varying velocity and momentum. This in turn raises the possibility of collisions from the side or behind (relative to a grain's direction of motion), as one particle overtakes another. Such collisions are not possible in a traditional lattice gas, in which particles have the same velocity and cannot overtake one another. Motion is implemented as a simple exchange of states (Fig. 11, top). In some cases, a collision may produce any of two or three different outcomes, as in the example of a head-on collision. In these cases, multiple transitions are encoded, each with a reduced transition rate; this is equivalent to assigning a fractional probability to each of the potential outcomes. Interestingly, indirect collisions tend to be less common than head-on collisions, because they only occur if one of the two particles does not move out of the way first.

The behavior of the CTS lattice-gas model is illustrated in Fig. 12, which shows particles in closed vessel. The number of particles in each motion state remains roughly constant over time, indicating that momentum is conserved. The CTS lattice-gas model lacks the speed advantage of traditional lattice-gas models; we present it here because it forms the basis for a cellular model of granular mechanics, which we turn to next.

To construct a lattice-grain model, we start with the CTS lattice-gas rules (Fig. 11) and add rules that implement gravity and frictional energy dissipation. To represent the effects of friction, we add an extra set of transitions, so that each collision has two possible types of transition: elastic and frictional. The frictional transition rules are illustrated in Fig. 13. The transition rates for elastic collisions (Fig. 11, rules 2–11) are reduced by multiplying each one by a user-specified coefficient of restitution, e. The

corresponding frictional transitions are assigned a rate of $f = 1 - e$ (except for rule 8, in which each of two frictional transitions is assigned a rate of $f/2$; Fig. 13, upper right). This approach to inelastic collisions is similar, for example, to that of Károlyi and Kertész (1998).

5   Gravity is implemented by assigning transitions that have the effect of adding downward momentum (Fig. 14). Thus, upward-moving particles transition to resting ones, while resting ones transition to downward-moving particles, and so on. Each of these transitions is assigned a user-specified rate g. Because of the limitations of the CTS approach, these rules provide only an approximate representation of gravitational accel-

10   eration. For example, in the CTS framework, the average speed (and hence momentum magnitude) of a particle reflects its average transition rate. Because the transition rate parameter is constant for each transition type, it is not possible to represent acceleration (though this limitation could be overcome in a future version by allowing a variable transition rate). For example, we approximate the tendency for velocity vectors to orient

15   downward by applying the transitions shown in the third row of Fig. 14, in which a particle moving both horizontally and downward at a 30° angle for the horizontal transitions to a state of moving purely downward. Obviously, this is somewhat unrealistic: in the real world (and in the absence of fluid drag), such a particle would sustain its horizontal momentum while accelerating downward. We also apply an "angle of repose"

20   rule (bottom column of Fig. 14), which allows particles resting on a slope to undergo down-slope motion. This rule has the effect of imparting a 30° angle of repose.

    Despite its limitations, the CTS lattice-grain model exhibits many characteristics of real granular flows, such as the emptying of a funnel and the resulting formation of a grain pile with angle-of-repose side slopes (Fig. 15). The CTS lattice-grain approach

25   appears to be promising for geomorphic systems, in which the detailed physics of particle acceleration and momentum exchange are likely to be less important than the general characteristics of granular behavior (e.g., Anderson and Bunas, 1993; Werner, 1995; Jyotsna and Haff, 1997; Tucker and Bradley, 2010).

Full Screen / Esc

Printer-friendly Version

Interactive Discussion

6 Discussion

CellLab-CTS provides a simple, easy-to-use framework for creating pairwise continuous-time stochastic cellular automata of the form pioneered by Narteau et al. (2001). As a modeling technique, the pairwise CTS method offers several advantages, both in comparison to more traditional differential-equation models, and in comparison to other forms of cellular automaton. The granularity of the approach can bring one closer to the "micro-physics" of a particular system; rather than adopting continuum equations that are assumed to capture the average behavior of a large ensemble of particles (e.g., Furbish and Haff, 2010; Furbish et al., 2012), one can instead directly address the statistics of interactions among discrete entities, such as sediment grains. This approach can shed light on the relationship between micro-scale and macro-scale behavior, rather than simply having to assume a particular relationship.

Pairwise CTS models are not appropriate for every problem. Their limitations include the use of a single cell size, which makes it difficult to address granular systems with a large range of particle sizes. The stochastic framework partly negates the speed advantage of deterministic models such as lattice gas automata. Nonetheless, there remains a wide variety of problems that can usefully be addressed with a pairwise CTS approach.

As a software implementation of the pairwise CTS concept, CellLab-CTS offers several practical capabilities. Its overall design makes the process of building a CTS application quite simple. A user needs only to write a relatively short Python script that contains (1) a definition of cell states, (2) definitions of the transitions involved, (3) a function that initializes, runs, and plots (and/or saves) output from the resulting model, and optionally (4) a callback function that updates any user-defined data upon each transition. CellLab-CTS users can choose between raster and hex grids, and between oriented and non-oriented models. The fact that CellLab-CTS is built on Landlab means that a user can take advantage of Landlab's various tools, such as reading of input parameters from a formatted text file using the ModelParameterDictionary tool,

and writing of output to standard file formats such as netCDF and VTK. These capabilities speed the development process, while also allowing users to take advantage of Python's extensive visualization and analysis libraries (such as matplotlib, mayavi, pandas, and others).

5   Another novel feature of CellLab-CTS is the ability to assign properties (including continuum values) to the grid nodes, and to update these properties dynamically using a callback-function approach. The property-tracking capability is complemented with the ability to treat certain node states as moving particles and to track their trajectories. Any properties associated with such particles automatically move with them. These

10 capabilities are especially useful in modeling assemblages of grains: a common use for cellular automata in geomorphology and granular mechanics (e.g., Furbish and Haff, 2010).

CellLab-CTS has some important limitations that could be addressed in future versions. Unlike the ReSCAL software of Rozier and Narteau (2014), the 2015 version of

15 CellLab-CTS is restricted to two-dimensional applications (this is actually a limitation of the current version of Landlab; once Landlab itself provides for 3-D grids, adaptation of CellLab-CTS to 3-D will be essentially automatic). CellLab-CTS 2015 was written completely in Python, and lacks the speed advantage of a compiled language. Although CellLab-CTS, like Landlab, makes use of the NumPy library for speed and efficiency,

20 the nature of the discrete-event simulation algorithms do not lend themselves to array operations; by definition, the CTS concept is an event-by-event approach. The speed limitation could be improved by translating CellLab-CTS' core routines into a compiled language such as Cython or C++, while preserving the flexibility of Python interfaces and libraries.

25 One limitation of CellLab-CTS that applies to granular-flow and sediment-transport problems is the present lack of a "binary" transition rule, in which a particular transition has a certain probability of not occurring at all, even if the states of the neighboring cells remain unchanged. Imagine closing your eyes and placing a pebble on a steep hillslope. Depending on the microtopography, the pebble might end up in a stable location,

Back | Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion

or it might end up rolling partway down the hill. If placed in a stable location, the pebble will not move until and unless something in its immediate neighborhood changes. This type of "either-or" situation could be implemented by modifying the transition rules to include a conditional probability of occurrence. A particular transition would be scheduled only with probability p_T. Such an approach would, for example, allow for a more realistic angle of repose in the lattice-grain model.

7   Conclusions

CellLab-CTS 2015 is a Landlab component that implements pairwise, continuous-time stochastic (CTS) cellular automata in two dimensions. CellLab-CTS enables researchers to efficiently create and explore CTS models by writing a short Python script that encodes the states, transition rules, and rates. The choice of square or hexagonal cells gives users control over grid symmetry. CellLab-CTS also provides the capability to represent moving particles, to assign user-defined properties to these particles, and to update these properties after each transition with a user-defined callback function. Integration with Landlab means that CellLab-CTS users can take advantage of a suite of capabilities, including input and output in standardized formats, and coupling with other Landlab components.

Code availiability, license, and requirements

CellLab-CTS 2015 is a component of Landlab version 0.1.28. The source code for version 0.1.28, which was released in September 2015, is provided in a git repository hosted on GitHub at https://github.com/landlab/landlab/tree/v0.1.28 (the latest version of Landlab is always available at http://github.com/landlab/landlab). Documentation and installation instructions for Landlab, which includes CellLab-CTS, is provided at http://landlab.readthedocs.org. Codes for the examples presented in this paper can be found at https://github.com/landlab/pub_tucker_etal_gmd. Software dependencies

Back | Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion

are listed at http://landlab.readthedocs.org/en/latest/install.html#dependencies. To the best of our knowledge, Landlab and CellLab-CTS will operate on any system that meets these software requirements; as of this writing, Landlab is known to work on recent-generation Mac, Linux, and Windows platforms. Landlab and its components, including

5 CellLab-CTS, are distributed under an MIT open-source license.

References

Alonso, J. and Herrmann, H.: Shape of the tail of a two-dimensional sandpile, Phys. Rev. Lett., 76, 4911, doi:10.1103/PhysRevLett.76.4911, 1996. 9526

Anderson, R. S.: Eolian ripples as examples of self-organization in geomorphological systems, Earth-Sci. Rev., 29, 77–96, 1990. 9509

15 Anderson, R. S. and Bunas, K. L.: Grain size segregation and stratigraphy in aeolian ripples modelled with a cellular automaton, Nature, 365, 740–743, 1993. 9509, 9528

Bailey, R. and Arnold, L.: Statistical modelling of single grain quartz D e distributions and an assessment of procedures for estimating burial dose, Quaternary Sci. Rev., 25, 2475–2502, 2006. 9522

20 Baxter, G. W. and Behringer, R.: Cellular automata models of granular flow, Phys. Rev. A, 42, 1017, doi:10.1103/PhysRevA.42.1017, 1990. 9525

Baxter, G. W. and Behringer, R.: Cellular automata models for the flow of granular materials, Physica D, 51, 465–471, 1991. 9526

Caracciolo, D., Noto, L. V., Istanbulluoglu, E., Fatichi, S., and Zhou, X.: Climate change and

25 Ecotone boundaries: insights from a cellular automata ecohydrology model in a Mediterranean catchment with topography controlled vegetation patterns, Adv. Water Resour., 73, 159–175, 2014. 9509

Chase, C. G.: Fluvial landsculpting and the fractal dimension of topography, Geomorphology, 5, 39–57, 1992. 9509

Chen, S. and Doolen, G. D.: Lattice Boltzmann method for fluid flows, Annu. Rev. Fluid Mech., 30, 329–364, 1998. 9510

Chopard, B. and Droz, M.: Cellular automata, Springer, 1998. 9510, 9527

Cottenceau, G. and Désérable, D.: Open environment for 2d lattice-grain CA, in: Cellular Automata, Springer, 12–23, 2010. 9526

Coulthard, T. and Van De Wiel, M.: A cellular model of river meandering, Earth Surf. Proc. Land., 31, 123–132, 2006. 9509

Coulthard, T., Kirkby, M., and Macklin, M.: A cellular automaton landscape evolution model, in: Proceedings of the First International Conference on GeoComputation, vol. 1, 248–281, 1996. 9509

Coulthard, T., Macklin, M., and Kirkby, M.: A cellular model of Holocene upland river basin and alluvial fan evolution, Earth Surf. Proc. Land., 27, 269–288, 2002. 9509

Coulthard, T., Hicks, D., and Van De Wiel, M.: Cellular modelling of river catchments and reaches: advantages, limitations and prospects, Geomorphology, 90, 192–207, 2007. 9509

Désérable, D.: A versatile two-dimensional cellular automata network for granular flow, SIAM J. Appl. Math., 4, 1414–1436, 2002. 9526

Désérable, D., Dupont, P., Hellou, M., and Kamali-Bernard, S.: Cellular automata in complex matter, Complex Systems, 20, 67–91, 2011. 9526

d'Humieres, D., Lallemand, P., and Frisch, U.: Lattice gas models for 3D hydrodynamics, Europhys. Lett., 2, 291–297, 1986. 9526

Fitt, A. and Wilmott, P.: Cellular-automaton model for segregation of a two-species granular flow, Phys. Rev. A, 45, 2383, doi:10.1103/PhysRevA.45.2383, 1992. 9526

Frisch, U., Hasslacher, B., and Pomeau, Y.: Lattice-gas automata for the Navier-Stokes equation, Phys. Rev. Lett., 56, 1505, doi:10.1103/PhysRevLett.56.1505,1986. 9526

Furbish, D. and Haff, P.: From divots to swales: Hillslope sediment transport across divers length scales, J. Geophys. Res., 115, F03001, doi:10.1029/2009JF001576, 2010. 9529, 9530

Furbish, D. J., Haff, P. K., Roseberry, J. C., and Schmeeckle, M. W.: A probabilistic description of the bed load sediment flux: 1. Theory, J. Geophys. Res.-Earth, 117, F03031, doi:10.1029/2012JF002352, 2012. 9529

Gutt, G. and Haff, P.: An automata model of granular materials, in: Proceedings of the fifth distributed memory computing conference, 629 pp., 4–6 April 1990. 9526

Hethcote, H. W.: The mathematics of infectious diseases, SIAM Rev., 42, 599–653, 2000. 9525

Title Page

Abstract | Introduction

Conclusions | References

Tables | Figures

◄ | ►►

◄ | ►

Back | Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion

Jasti, V. K. and Higgs, C. F.: A lattice-based cellular automata modeling approach for granular flow lubrication, J. Tribol.-T.-ASME, 128, 358–364, 2006. 9526

Jasti, V. K. and Higgs III, C. F.: A fast first order model of a rough annular shear cell using cellular automata, Granul. Matter, 12, 97–106, 2010. 9526

Jerolmack, D. and Paola, C.: Complexity in a cellular model of river avulsion, Geomorphology, 91, 259–270, 2007. 9509

Jyotsna, R. and Haff, P.: Microtopography as an indicator of modern hillslope diffusivity in arid terrain, Geology, 25, 695–698, 1997. 9509, 9528

Karimabadi, H., Driscoll, J., Omelchenko, Y. A., and Omidi, N.: A new asynchronous methodology for modeling of physical systems: breaking the curse of courant condition, J. Comput. Phys., 205, 755–775, 2005. 9517, 9518

Károlyi, A. and Kertész, J.: Granular medium lattice gas model: the algorithm, Comput. Phys. Commun., 121, 290–293, 1999. 9526

Károlyi, A. and Kertész, J.: Lattice-gas model of avalanches in a granular pile, Phys. Rev. E, 57, 852, doi:10.1103/PhysRevE.57.852, 1998. 9526, 9528

Károlyi, A., Kertész, J., Havlin, S., Makse, H. A., and Stanley, H. E.: Filling a silo with a mixture of grains: friction-induced segregation, Europhys. Lett., 44, 386, 1998. 9526

Kessler, M., Murray, A., Werner, B., and Hallet, B.: A model for sorted circles as self-organized patterns, J. Geophys. Res.-Sol. Ea., 106, 13287–13306, 2001. 9509

Kozicki, J. and Tejchman, J.: Application of a cellular automaton to simulations of granular flow in silos, Granul. Matter, 7, 45–54, 2005. 9526

LaMarche, K. R., Conway, S. L., Glasser, B. J., and Shinbrot, T.: Cellular automata model of gravity-driven granular flows, Granul. Matter, 9, 219–229, 2007. 9526

Landlab Development Team: Landlab Documentation, available at: http://landlab.readthedocs.org, last access: 11 September 2015. 9509, 9513

Lebedeva, M., Fletcher, R., Balashov, V., and Brantley, S.: A reactive diffusion model describing transformation of bedrock to saprolite, Chem. Geol., 244, 624–645, 2007. 9524

Lebedeva, M., Fletcher, R., and Brantley, S.: A mathematical model for steady-state regolith production at constant erosion rate, Earth Surf. Proc. Land., 35, 508–524, 2010. 9524

Maher, K.: The dependence of chemical weathering rates on fluid residence time, Earth Planet. Sc. Lett., 294, 101–110, 2010. 9524

Martinez, J. and Masson, S.: Lattice Grain Models, E & FN Spon, 556–563, 1998. 9526

Murray, A. and Paola, C.: A cellular model of braided rivers, Nature, 371, 54–57, 1994. 9509

Narteau, C., Le Mouël, J., Poirier, J., Sepúlveda, E., and Shnirman, M.: On a small-scale roughness of the core–mantle boundary, Earth Planet. Sc. Lett., 191, 49–60, 2001. 9509, 9511, 9525, 9529

Narteau, C., Zhang, D., Rozier, O., and Claudin, P.: Setting the length and time scales of a cellular automaton dune model from the analysis of superimposed bed forms, J. Geophys. Res., 114, F03006, doi:10.1029/2008JF001127, 2009. 9509, 9511, 9513

Nicholas, A. P.: Cellular modelling in fluvial geomorphology, Earth Surf. Proc. Land., 30, 645–649, 2005. 9509

Omelchenko, Y. and Karimabadi, H.: Self-adaptive time integration of flux-conservative equations with sources, J. Comput. Phys., 216, 179–194, 2006. 9517, 9518

Omelchenko, Y. and Karimabadi, H.: A time-accurate explicit multi-scale technique for gas dynamics, J. Comput. Phys., 226, 282–300, 2007. 9517, 9518

Osinov, V.: A model of a discrete stochastic medium for the problems of loose material flow, Continuum Mech. Therm., 6, 51–60, 1994. 9526

Pandey, S. and Rajaram, H.: Investigating the influence of subsurface heterogeneity on chemical weathering in the critical zone using high resolution reactive transport models, in: AGU Fall MeetingAbstracts, vol. 1, p. 3599, 2014. 9524

Peng, G. and Herrmann, H. J.: Density waves of granular flow in a pipe using lattice-gas automata, Phys. Rev. E, 49, R1796, doi:10.1103/PhysRevE.49.R1796, 1994. 9526

Plug, L. J. and Werner, B.: Nonlinear dynamics of ice-wedge networks and resulting sensitivity to severe cooling events, Nature, 417, 929–933, 2002. 9509

Rhodes, E. J.: Optically stimulated luminescence dating of sediments over the past 200,000 years, Annu. Rev. Earth Pl. Sc., 39, 461–488, 2011. 9522

Rothman, D. H. and Zaleski, S.: Lattice-gas cellular automata: simple models of complex hydrodynamics, vol. 5, Cambridge University Press, 2004. 9526

Rozier, O. and Narteau, C.: A real-space cellular automaton laboratory, Earth Surf. Proc. Land., 39, 98–109, 2014. 9509, 9513, 9530

Tucker, G. and Bradley, D.: Trouble with diffusion: reassessing hillslope erosion laws with a particle-based model, J. Geophys. Res, 115, F00A10, doi:10.1029/2009JF001264, 2010. 9509, 9528

Von Neumann, J.: The general and logical theory of automata, Cerebral Mechanisms in Behavior, 1–41, 1951. 9508

Werner, B.: Eolian dunes: computer simulations and attractor interpretation, Geology, 23, 1107–1110, 1995. 9509, 9528

Wolfram, S.: Cellular automata, Los Alamos Science, 9, 2–27, 1983. 9510

Zhang, D., Narteau, C., and Rozier, O.: Morphodynamics of barchan and transverse dunes using a cellular automaton model, J. Geophys. Res.-Earth, 115, F03041, doi:10.1029/2009JF001620, 2010. 9509

Zhang, D., Narteau, C., Rozier, O., and du Pont, S. C.: Morphology and dynamics of star dunes from numerical modelling, Nat. Geosci., 5, 463–467, 2012. 9509

Zhou, X., Istanbulluoglu, E., and Vivoni, E. R.: Modeling the ecohydrological role of aspect-controlled radiation on tree-grass-shrub coexistence in a semiarid climate, Water Resour. Res., 49, 2872–2895, 2013. 9509

Title Page

Abstract | Introduction

Conclusions | References

Tables | Figures

|◄ | ►|

◄ | ►

Back | Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion

Table 1. States in the CTS Lattice Gas and Lattice Grain models

State code	Description
0	empty or fluid
1	moving upward
2	moving right and upward
3	moving right and downward
4	moving downward
5	moving left and downward
6	moving left and upward
7	resting
8	wall
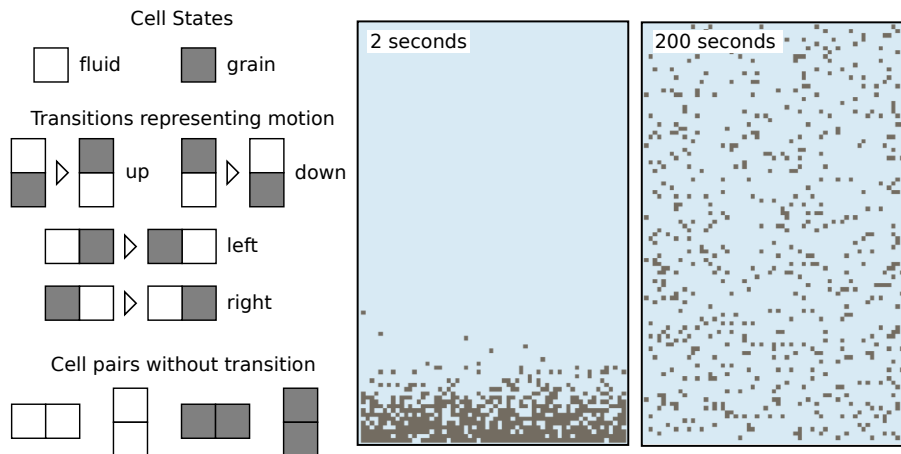
Figure 1. CellLab-CTS model of grains suspended in a stirred (turbulent) fluid. Left: illustration of cell states, cell-pair states, and cell-pair transitions. Grains are assumed to be neutrally buoyant, and turbulence is isotropic, so that there is an equal probability of grain motion in each direction. Grain motion is modeled as a transition that switches the position of grain and fluid at a user-specified rate. In this example, we assume that the grains are 1 mm diameter tea leaves and the characteristic turbulent velocity fluctuation is $0.01\,\mathrm{m\,s^{-1}}$, so that cell size $\delta = 0.001\,\mathrm{m}$ and the mean transition rate is $10\,\mathrm{cells\,s^{-1}}$.
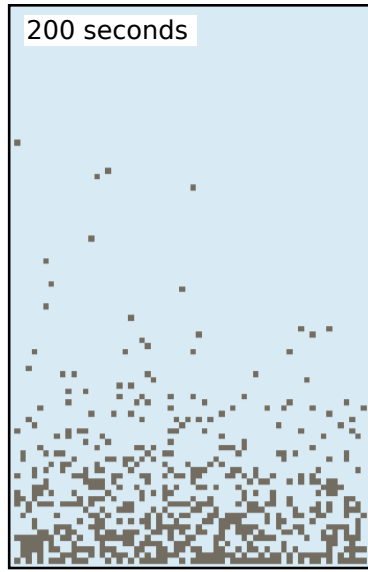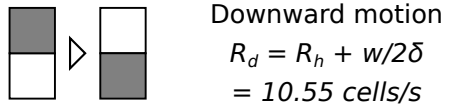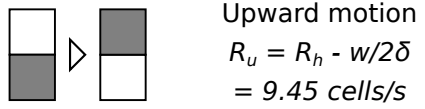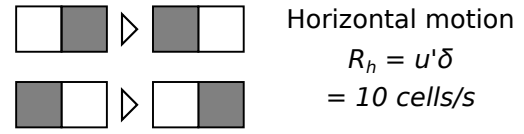
Figure 2. Turbulent suspension model with grains (1 mm tea leaves) that are 0.2 % denser than the surrounding fluid. Left: illustration of cell-pair transitions representing motion in the four directions. Grain settling velocity of 0.0011 m s^{-1} imparts asymmetry to the transition probabilities.

(a) Raster grid

(b) Hex grid

○ Node   ⟋ Link   ▦ Cell
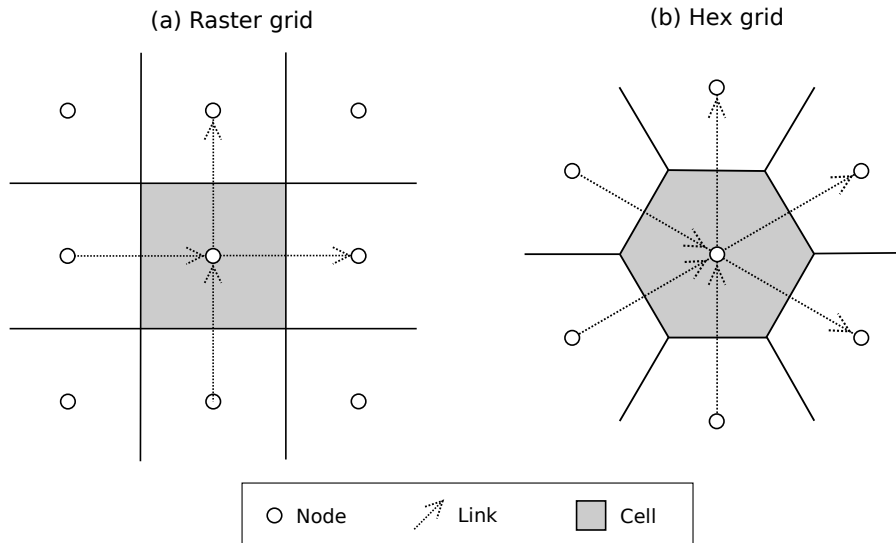
Figure 3. Schematic illustration of CellLab-CTS grid types and geometric primitives. **(a)** Regular (raster). **(b)** Triagonal lattice with hexagonal cells. Note that standard link orientation is within the $x \geq y$ half-plane; in other words, the angle of a link, θ, with respect to the positive x axis is always $-45° \leq \theta \leq 135°$.
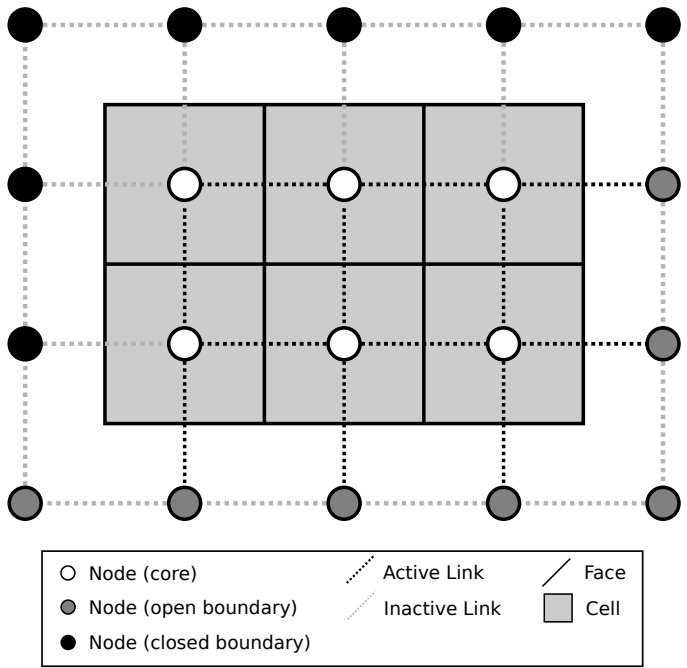
Figure 4. Example of a simple Landlab raster grid, illustrating open and closed boundary nodes, and active and inactive links.

Raster grid
Vertical pair
Orientation code 0

Raster grid
Horizontal pair
Orientation code 1

Hex grid
Vertical pair
Orientation code 0

Hex grid
Angling-up pair
Orientation code 1

Hex grid
Angling-down pair
Orientation code 2
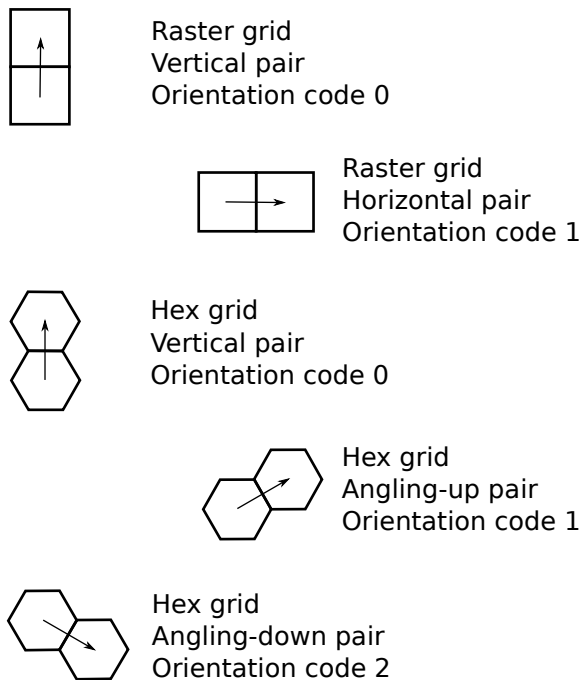
Figure 5. Illustration of cell-pair orientations and corresponding orientation codes. Arrows represent links, each of which connects a tail node to a head node.

Full Screen / Esc

Printer-friendly Version

Interactive Discussion

Event queue

Array of transition times

LINK ID	2
TRANS. TIME	0.5612
TRANS. TO	1

LINK ID	17
TRANS. TIME	0.6188
TRANS. TO	0

LINK ID	5
TRANS. TIME	1.1972
TRANS. TO	2

0 | 4.9462
1 | 7.3115
2 | 0.5612
...

Figure 6. Illustration of the event queue and the transition-time array. Each event object contains the ID number of the link, the time at which the next transition is scheduled to occur, and the link-state code for the transition (that is, the code for the new cell pair after transition occurs). Left: events are stored in a heap that is sorted by transition time, soonest on top. Right: the transition-time array, which is simply an ordered array, indexed by link ID, containing the time of the next scheduled transition at the corresponding link. If the transition time for an event object does not match the corresponding entry in the transition-time array, it means that the originally scheduled transition has been nullified by other transitions and is no longer valid (see text).

CellLabCTSModel		
Variable name	**Type and size**	**Description**
`grid`	*Landlab grid object*	simulation grid
`current_time`	*float*	current simulation time
`node_state`	*1xN array of int*	state of each node
`node_pair`	*list (xNL) of tuple*	(tail, head, orientation) for each pair state
`event_queue`	*heap of Event objects*	transition events, soonest on top
`next_update`	*1xL array of float*	time of next transition at each link
`link_orientation`	*1xL array of int*	orientation code for each link
`link_state`	*1xL array of int*	pair state of each link
`n_xn`	*1xNL array of int*	number of transitions for each pair state
`xn_to`	*NLxNT array of int*	new pair state for each transition
`xn_rate`	*NLxNT array of float*	rate parameter for each transition

`RasterCTS`	`OrientedRasterCTS`	`HexCTS`	`OrientedHexCTS`
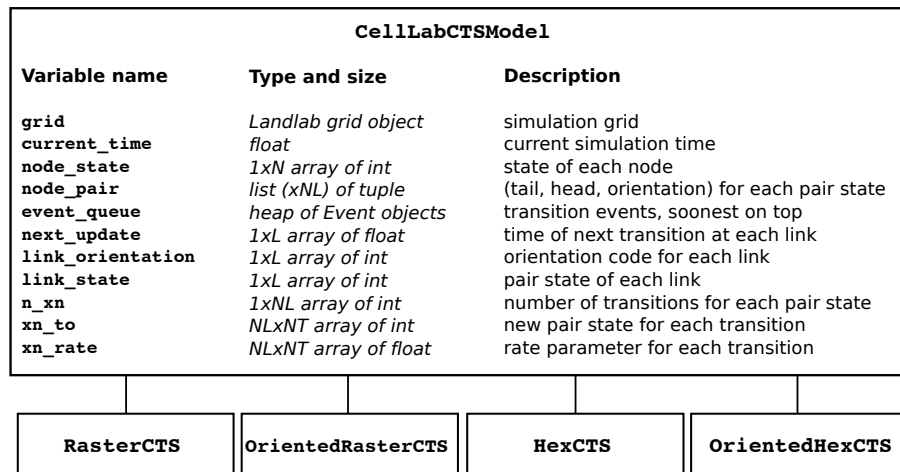
Figure 7. CellLab-CTS class hierarchy and main data structures. The base class, Cell-LabCTSModel, has four subclasses: RasterCTS, OrientedRasterCTS, HexCTS, and Oriented-HexCTS. A user selects one of these four subclasses based on whether the model to be built has a hex or raster grid, and on whether pair orientation matters. N = number of grid nodes, L = number of grid links, NL = number of possible link (node pair) states, NT = maximum number of transitions for any link state.
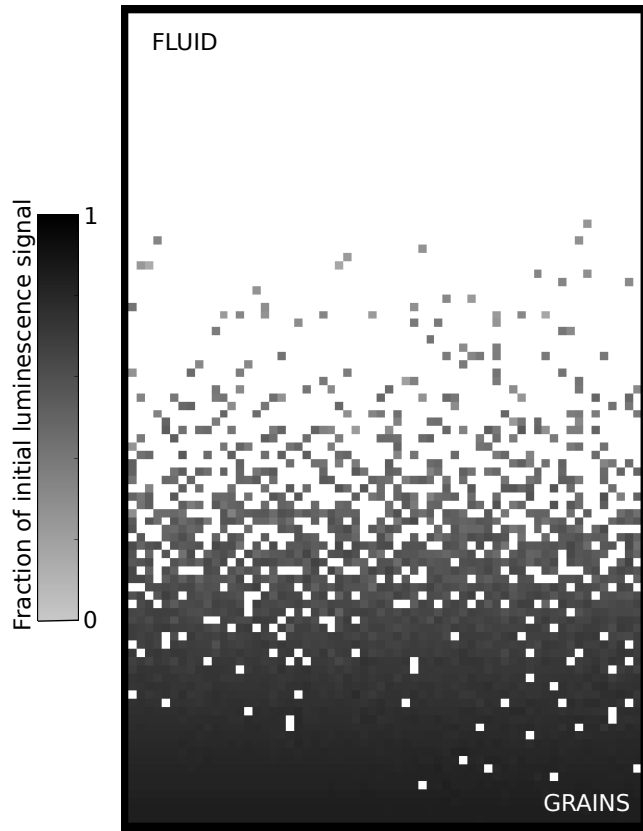
Figure 8. Model of turbulent sediment suspension, showing the computed luminescence signal after 20 s of stirring and bleaching. The original luminescence signal is removed (bleached) by exposure to light. Because light intensity declines exponentially from top to bottom, grains near the bottom are less fully bleached than those near the top. Turbulent mixing disperses the partially bleached grains.

Title Page

Abstract | Introduction

Conclusions | References

Tables | Figures

⏮ | ⏭

◀ | ▶

Back | Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion
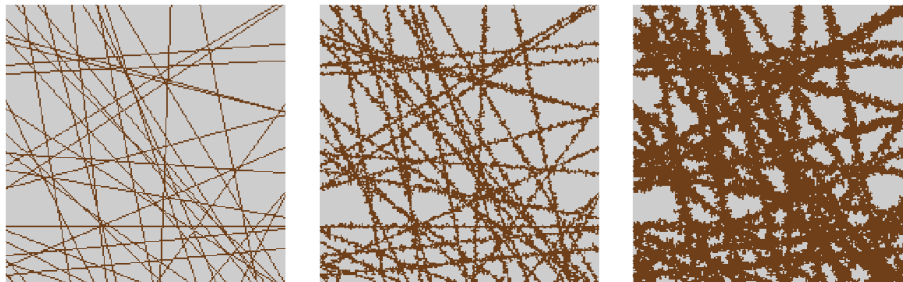
Figure 9. Three time-slices from a CellLab-CTS model of bedrock weathering along fracture planes. Light colored cells are unweathered mineral grains; dark-colored cells are grains that have been chemically altered to form saprolite (rock material that has been weathered but not displaced). Left to right: time 0, time 10, and time 30. Here the time is normalized by initial fracture width and weathering rate; one time unit represents the average time to weather fresh rock to a depth of one initial fracture width. For example, if fracture width (cell size) were 1 cm and weathering rate 10^{-5} m yr^{-1}, then one time unit = 1000 years.
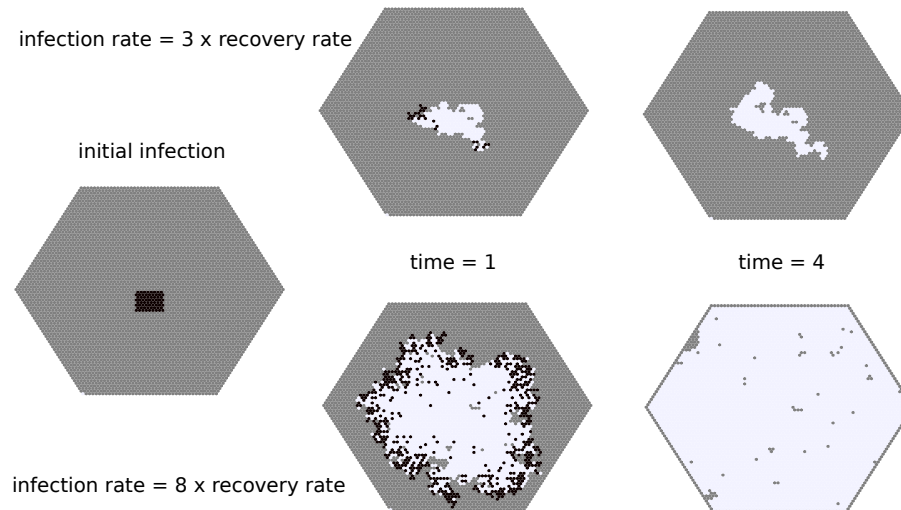
Figure 10. A Susceptible-Infectious-Recovered (SIR) model built with CellLab-CTS. This example compares runs with two different infection rates (top and bottom rows, respectively). Gray = susceptible; black = infectious; white = recovered. One time unit = one average recovery time. This example model uses the HexCTS subclass (non-oriented model with hexagonal cells).
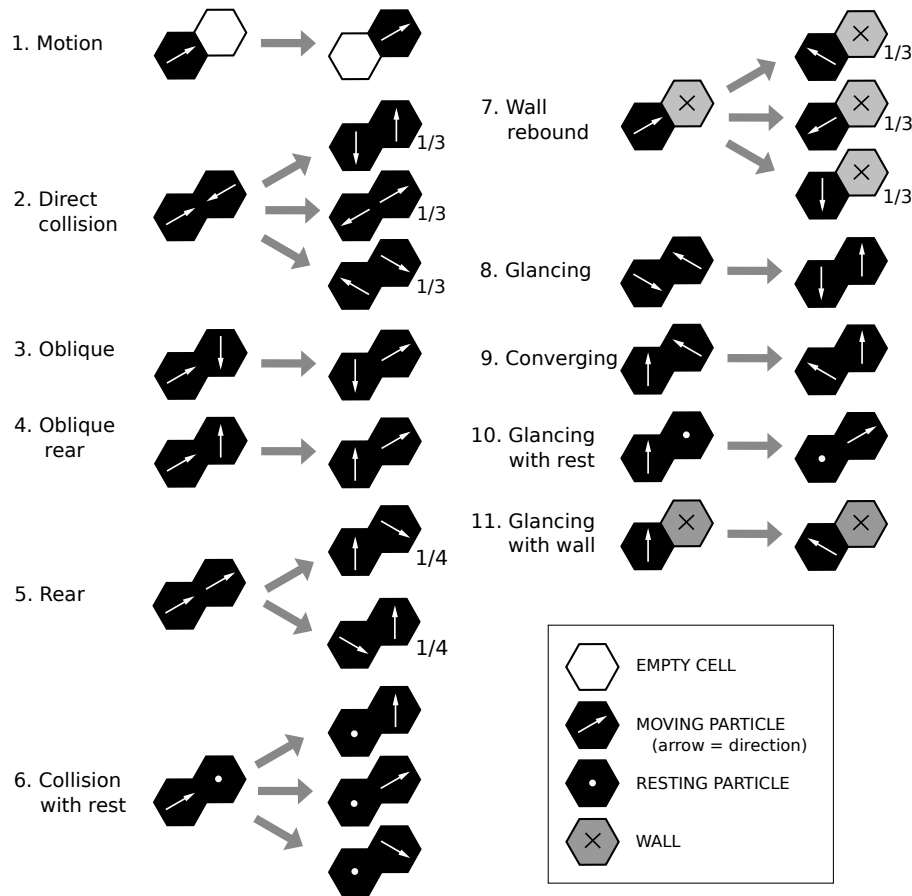
Figure 11. Motion and collision rules for a pairwise CTS lattice gas model.

Title Page

Abstract | Introduction

Conclusions | References

Tables | Figures

◄ | ►►

◄ | ►

Back | Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion

Full Screen / Esc

Printer-friendly Version

Interactive Discussion
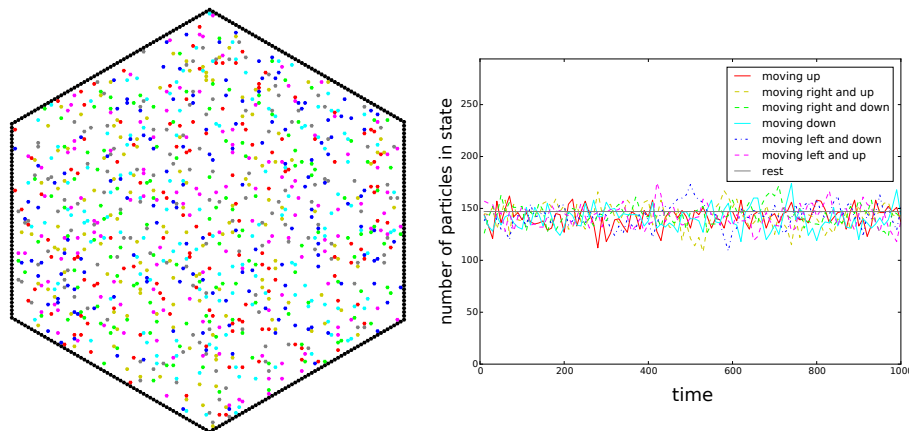
Figure 12. Example of a CTS lattice gas model. Left: particles undergoing random (brownian) motion in a vessel. Color codes: white = empty, black = wall, gray = resting; others moving in the directions indicated in inset legend. Right: number of particles in each movement direction vs. time, illustrating the conservation of average momentum.
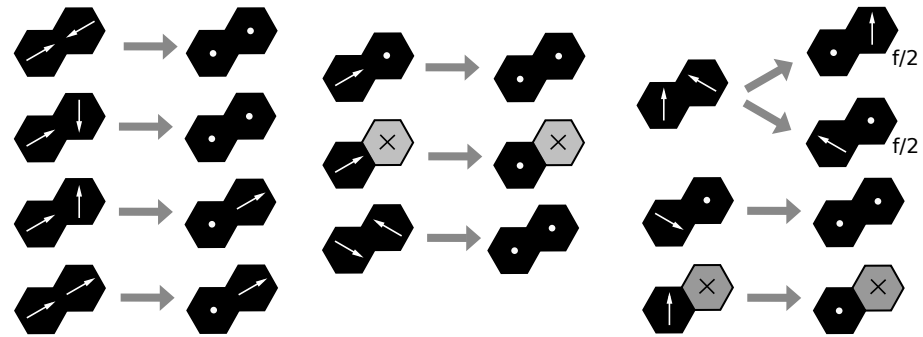
Figure 13. Friction rules for lattice-grain model. Unless otherwise noted, each transition has a rate f, whereas the corresponding transition rates for elastic behavior (Fig. 11) have rate $e = 1 - f$.
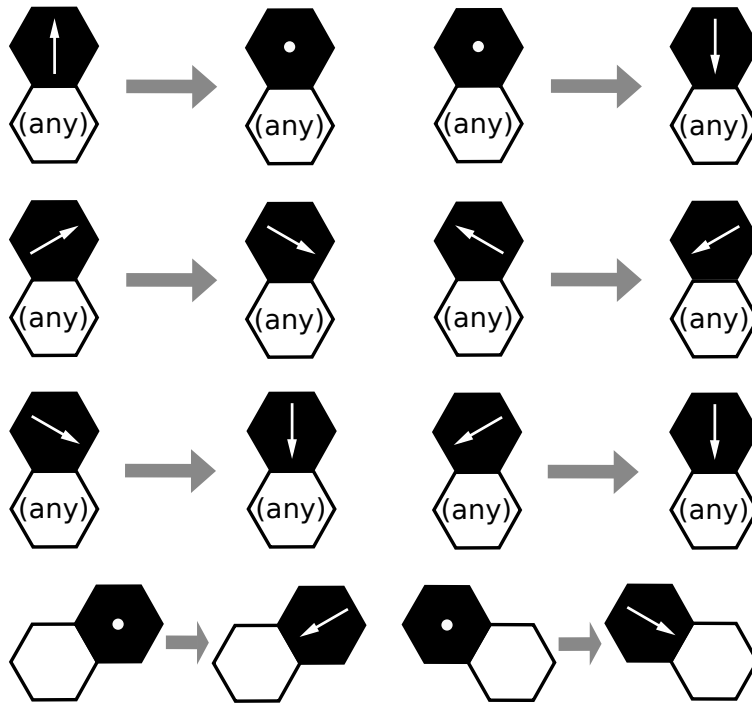
Figure 14. Gravity rules for lattice-grain model. Each transition has a rate g. The bottom two transitions represent angle-of-repose behavior. This example uses the OrientedHexCTS subclass.
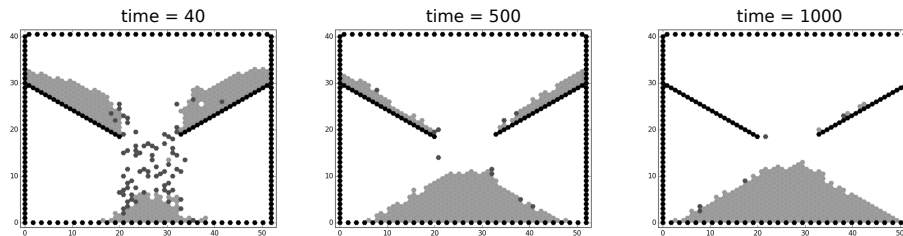
Figure 15. Three snapshots from a lattice-grain simulation showing the emptying of a silo. Color codes: white = empty, black = wall, light gray = resting grain, and dark gray = moving grain.

Back | Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion