Tetris: a training image generator for SGeMS

Alexandre Boucher[*], Rahul Gupta, Jef Caers, Addy Satija
Stanford University

Abstract

The recent developments of training image-based geostatistics have allowed the creation of stochastic models with greater realism than before. These algorithms aim at reproducing spatial patterns, such as connectivity, that are depicted in a training image. A training image contains the possible spatial configurations for any given geological object and relationships between objects. However, algorithms for generating training images are still incomplete and unsatisfactory. The proposed design overcomes a major hurdle of rigidity of pre-defined geological objects in the current training image generators. This paper presents a plugin to the SGeMS software that allows modelers to generate geological objects with complex geometries and the relevant interactions between these objects.

First, a geological object is built either from pre-coded shapes, such as ellipsoid, cuboid, kernels, or user defined. These basic shapes are then assembled with geometrical operations (difference, union and intersection) to create new complex shapes. For added flexibility, any shape can be translated, rotated and sheared. A second set of parameters control the interactions between the geological objects. Each parameter used in the training image construction (e.g. size, the rotation angles, number of stacks) can vary in space. This locally varying parametrization allows the representation of trends in geological body geometry, interactions, and locations.

[*]The auhors would like to thanks Marco Pontiggia, Sergio Nardon and Giuseppe Serafini at Eni for their support and valuable suggestions.

1 Introduction

With the greater availability of training-image (TI) based algorithms the need to generate 3D TIs is growing rapidly. At this time, object based algorithms (Lantuejoul, 2002) are the best sources of 3D TIs. Training images are easy to generate since they need not to be conditional to data, they must only correctly represents the geo-objects and the spatial relationship between the geo-objects. The challenge is to design a program with a clean interface that can handle many types of geological morphology without requiring the user to change the source code.

Most current object-based simulation programs are specific to a geological environment (Tetzlaff and Harbaugh, 1989) with most of the developments targeted for fluvial systems such as fluvsim (Deutsch and Wang, 1996; Deutsch and Tran, 2002), lobesim (Deutsch and Tran, 2004) or more recently with event-based fluvial model (Pyrcz et al., 2009) which can reproduce a wide variety of fluvial systems.

Maharaja (2008) took the generalist approach in writing the TiGenerator plugin to SGeMS (TiGenerator is currently part of the SGeMS distribution). The program is an object-based program with four shapes (ellipsoid, half-ellipsoid, cuboid and sinusoid) plus the capability to load user-defined geometries. The program allows to control the level of overlap between geo-objects to constraint the relative positions between geo-objects. The important difference of that generalist approach versus the geology specific approach is that the user must interpret the geological entitities into geometrical shapes prior of creating a training image. The choice of shapes available by TiGenerator plugin is too limited to mimic complex geological objects and environments.

The Tetris plugin can be seen as the next generation of TiGenerator for the SGeMS software. It is general at its roots but offer the possibility to model complex geological objects such as channel-lobe-crevasse complex. By providing a mechanism to combine different shapes the Tetris plugin can create geo-objects with complex geometries. Furthermore, it contains a larger library of shapes and can accomodate trends in location and geometries of the geo-objects.

In Section 2 the design, structure and algorithms are detailed. Section 3 presents the graphical interfaces for the users to parameterize the geo-objects and the simulation parameters. Some examples are shown in Section 4

1.1 Notation

Text written in the format `formatForClass` refers to C++ object of Tetris. The term object refers to a instantiation of a C++ class. The term geo-object refers to the modeling of a geological entity, it is the output of the Tetris program.

1.2 About SGeMS

SGeMS (Remy et al., 2009) is free and open source. Open source software gives the users access to the source code, which can be modified with or without restrictions, depending on the license requirement. SGeMS is distributed under a GNU General Public license (GPL) and must be distributed with its source code [1]. Open source software gives flexibility to the users: the source code can be modified to address complex problems that may not have been easily handled from the traditional interface of geostatistical algorithms. Access to the source code also allows the users to track and correct bugs without waiting for an official release or patches from a vendor. More importantly, the transparency permits the user to validate the algorithms by inspecting the actual code instead of using them as black boxes.

2 Design and algorithm

The design of the Tetris algorithm consists of two main steps: (a) building the geological objects (geo-objects) and (b) positioning them in a grid given a set of constraints. The geo-objects can be build through a meta-programming approach based on recursive composition technique (Gamma et al., 1994),i.e. building increasingly complex elements from simpler ones.

[1] See `http://www.gnu.org/copyleft/gpl.html` for more details on the GPL license.

The construction of the geo-objects is the responsibility of a geo-object manager that store the relevant parameters in order to generate geo-object raster on demand.

The defined geo-objects are then positioned on a grid given a set of constraints through a simulation event that controls the target number of geo-objects (stopping criterion), the interaction between geo-objects (interactions rules) and potential preferential locations (positioning and spatial intensity field).

The algorithm is summarized as follows:

```
1: Initialize the geo-object managers
2: Initialize the simulation events
3: for each simulation event do
4:    while stopping criterion not met do
5:       Get a location in the grid
6:       Create a geo-object
7:       if geo-object violates interaction rules then
8:          Delete geo-object
9:       else
10:         Draw the geo-object
11:         Update the stopping criterion
12:      end if
13:   end while
14: end for
```

2.1 Building Geological Object

A geo-object is the top level representation of a geological entity. It is modeled with the class `GeoObject`. A geo-object is composed of elements, modeled with the class `Element`, which are themselves an assemblage of geometrical shapes (`Shape`). The geo-objects and the elements have geological meaning, for instance a channel complex would be a geo-object and a crevasse of that system an element. The shapes (e.g. sphere, cylinder, cuboid, ...) are just the building blocks without geological significance.

This hierarchy gives flexibility in creating complex elements by assembling any available shapes and then setting up these element together to create a geological object. Each of these concepts are discussed in detail below.

Shape

The Tetris library currently has 10 geometrical shapes and 3 types of operations for these shapes. A shape operation aims at combining the existing parametric shape into more complex shape. Currently available shape types are:

Half ellipsoid Defined by the x, y and z directions radii.

Ellipsoid Defined by the x, y and z directions radii.

Kernel Shape Either a radial (Gaussian), exponential or spherical kernel. Each is defined by the x, y radii and height.

Cuboid Defined by the x, y and z side dimensions.

Sphere Defined by its radius.

Cylinder Defined by its length and radius of the cross-section

Lobe Shape Partial implementation of the lobesim shape of Deutsch and Tran (2004) It is defined by its length, width, thickness and the distances along the length where the width and the thickness are maximum.

Sinusoid A sine function defined by its thickness, width, wavelength and amplitude.

Gaussian Sinusoid A shape generated by the convolution of a radial kernel with a Gaussian process. The user can control the bandwidth of the kernel, the amplitude, the thickness, and the length of the resulting shape. The cross-section is an implementation of Deutsch and Tran (2002).

User Defined A set of relative coordinates (i,j,k) defining a shape.

Each shape can be translated, rotated and sheared. The rotation is done using a series of nine shear operations, see Chen and Kaufman (2000). The shearing is a translation gradient in either the x or y direction along the vertical axis.

The shapes can be combine through the following three shape operations:

Difference Return a shape which is composed of the difference between the modifiable shape and a set of modifier shapes.

Union Return a shape made of the union of a set of shapes.

Intersection Return a shape made of the intersection of a set of shapes.

Each shape and shape operation is controlled by a manager `ShapeManager`. The task of a `ShapeManager` is to create a rasterized shape with specified dimensions. The `ShapeManager` for the shape operation calls the `ShapeManager` objects for each underlying shapes and performs the requested operations on the generated `Shape` objects.

Element

An element is simply the root of a shape construction. The `Element` object is controlled by a manager of abstract type `ElementManager`. Only one implementation of `ElementManager` is currently available `ElementFromShapesManager`, which in turns control the underlying `ShapeManager` objects.

Geo-object

A geo-object is the final product of the design. A manager abstract class, called `ObjectManager` creates and manages each geo-object by creating in turns each one of the elements. The `ObjectManager` can also operate on the element by taking the difference or the intersection between elements.

Currently, three implementations of `ObjectManager` are available:

Using Element `ObjectFromElementTreeManager` object: Create geo-object through shapes and shape operations. It offers the full flexibility of the design.

Simple Carbonate Mound `CarbonateMoundObjectManager` object: Specialized geo-object manager to create carbonate mounds assembled from up to three elements: the core, the shell and the debris. Each of the element can either be a half-ellipsoid or a kernel shape. This geo-object manager automatically performs the required operations on these shapes. Only the shell element is required, the core and the debris are optional. The operations on the elements are:

```
1: Intersect the core with the shell
   core.intersection( shell )
2: Remove the core from the shell
   shell.difference( core )
3: Remove the shell and the core from the debris
   debris.difference( shell)
   debris.difference( core )
```

The result is three elements that are self-consistent to create a geo-object reprenting a carbonate mound.

Channel-Lobe-Crevasse `ChannelLobeCrevasseObjectManager` object: This specialized geo-object manager constructs a channel system made of the channel, the terminal lobe, crevasses and shale drapes around the channel. Only the channel element is required. The manager ensures that (1) the terminal lobe is attached at the end of the channel with the proper initial thickness and width, and (2) that the crevasse splays are located where the channel curvature is high. The steps to create a geo-object are:

```
1: Create the channel element with a Gaussian Sinusoid shape
2: Add the drape element to the channel element
3: Create a terminal lobe element with lobeShape shape
4: Translate the lobe element to the end of the channel
```

```
5:  Extract from the channel the high curvature locations
6:  Randomly select a number of crevasse attachment from the high
    curvature locations
7:  for Each high curvature point selected do
8:      Create a crevasse with proper rotation
9:      Translate the crevasse such that its starting point coincides with
        the selected location of high curvature
10: end for
```

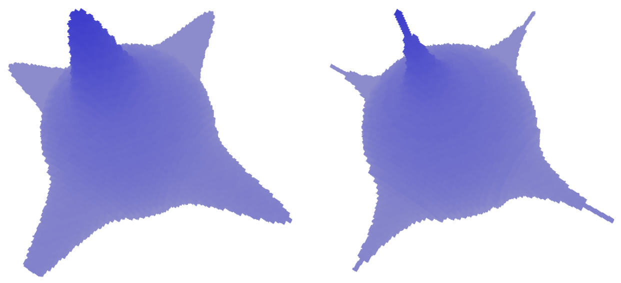
The `CarbonateMoundObjectManager` manager can be fully implemented using the `ObjectFromElementTreeManager` type of manager. This is not the case with the channel-lobe-crevasse manager where special operations are internally performed that could not be done with the available shape or element operations. Additional specialized managers can be written to facilitate the construction of specific training images such as fractures networks.

Geo-object examples

Figure 1 shows the resulting shape of a union between 6 kernels shapes and one sphere. The kernel shapes have been translated and rotated to fit on the surface of the sphere. This shape could mimic pore spaces for instance.

Figure 2 shows a geo-object representing a carbonate mound made of three elements with `ElementFromShapesManager`. The steps to construct that geo-object are:

```
1: Create the shape for the inner element with a half-ellipsoid with radius
   10,25 and 40 units in the x,y and z direction respectively
   inner = HalfEllipsoid( 10, 25, 40 )
2: Create the shape for the outer element with a half-ellipsoid
   outer = HalfEllipsoid( 30, 30, 35 )
3: Perform a N-S shearing operation on both the inner and outer shapes
   inner.shear( 0,0.3 )
   outer.shear( 0,0.5 )
```



(a) Gaussian (radial) kernels (b) Exponential kernels

Figure 1: Complex shapes made from one sphere and 6 kernel shapes.

4: Create the drape element by taking the difference between two sheared half-ellipsoid

```
drapes_top = HalfEllipsoid( 40, 40, 15 )
drapes_bot = HalfEllipsoid( 35, 35, 9 )
drapes.shear(0,0.5)
drapes_bot.shear(0,0.5)
drapes.difference(drapes_bot)
```

5: Take the intersection between the inner and the outer elements to confine the top of the inner elements to the top of the outer elements

```
inner.intersection( outer )
```

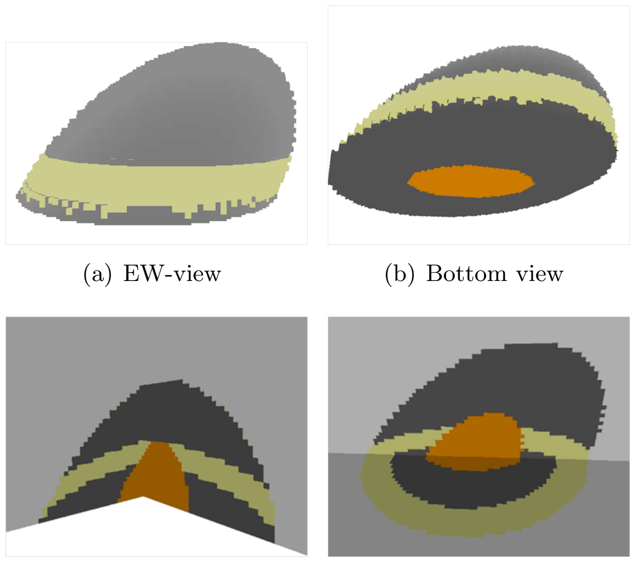
6: Remove the shape of the inner element from the shape of the outer and drape elements

```
outer.difference( inner )
drapes.difference( inner )
```

7: Remove the shape of the drape element from the shape of the outer element

```
outer.difference( drapes )
```

Figure 3 shows a geo-object representing a carbonate mound with a complex internal structure built with eight elements 4 half ellipsoids (the core part) and 4 radial kernels (the debris) and twelve operations (intersections



(a) EW-view (b) Bottom view

(c) EW and NS cross-sections (d) EW and plan cross-section

Figure 2: View of a carbonate mound build with three elements, the inner core (orange), the drape (green) and the outer shell (grey).

and differences). Figure 4 shows a carbonate mound with a displacement due to fault for instance. The steps in building thiscomplex geo-object are:

1: Create the shape for the inner element with a half-ellipsoid
```
inner = HalfEllipsoid( 10, 25, 40 )
```
2: Create the shape for the outer element with a half-ellipsoid
```
outer = HalfEllipsoid( 30, 30, 35 )
```
3: Perform a shearing operation on both the inner and outer shapes
```
inner.shear( -0.2,-0.5 )
outer.shear( -0.2,-0.5 )
```
4: Create two cuboids to cut the inner and outer shapes
```
innerFaulted = Cuboid( 40, 40, 40 )
outerFaulted = Cuboid( 40, 40, 40 )
```
5: Translate the cuboids to overlap with the portion that needs to be split
```
innerFaulted.translate( 25, 0, 20 )
```

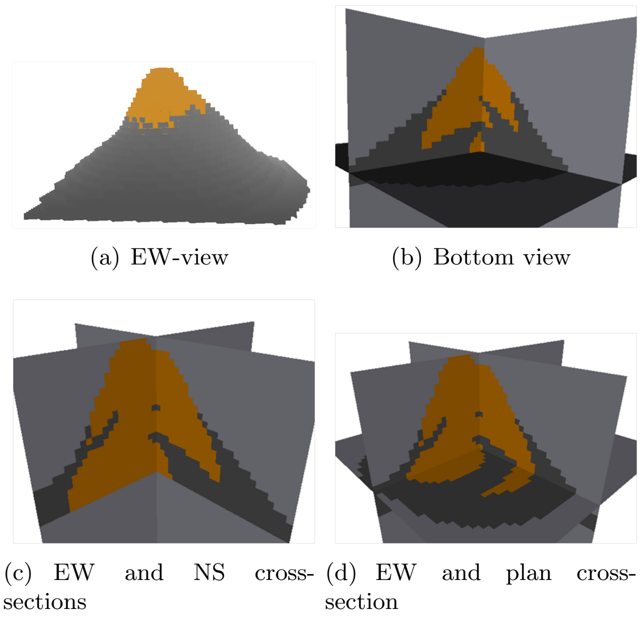
```
    outerFaulted.translate( 25, 0, 20 )
6:  Take the intersection between cuboid and the half-ellipsoids
    innerFaulted.intersection( inner )
    outerFaulted.intersection( outer )
    The result is that innerFaulted and outerFaulted now take the shape
    of a quarter ellipsoid.
7:  Remove the faulted inner and outer from the original hald-ellipsoids
    inner.difference( innerFaulted )
    outer.difference( outerFaulted )
8:  Rotate and translate the faulted section to mimic displacement
    innerFaulted.rotate( 0,20,0 )
    outerFaulted.rotate( 0,20,0 )
    innerFaulted.translate( 0,7,5 )
    outerFaulted.translate( 0,7,5 )
```

Finally, Figure 5 shows the results of the specialized object manager `ChannelLobeCrevasseObjectManager` to represent channels with terminal lobes, crevasses and drapes. It would be impossible to build such a geo-object with the `ObjectFromElementTreeManager` since it would not have been possible to correctly place the lobe at the end of the channel, and each crevasse where the curvature is high.

2.2 Simulating geo-objects with simulation event

Once the geo-objects have been defined, the next step is to position them on the grid. The locations and number of geo-objects depend on how many are needed (stopping criterion), the stacking options, preferential locations (trends) and the intended interactions between geo-objects. A simulation event controls these parameters.

The basic algorithm of a simulation event is to select a voxel in the grid; check if that location is admissible; and if so draw a geo-object at that location.


(a) EW-view (b) Bottom view

(c) EW and NS cross-sections (d) EW and plan cross-section

Figure 3: View of carbonate mound build with eight elements and twelve operations.

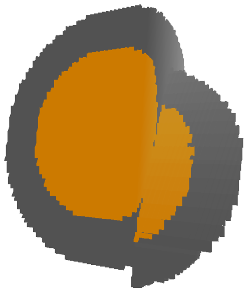
Stopping criterion

There are currently two available criteria to stop the simulation process: (a) the desired number of geo-objects have been reached or (b) the desired proportion of the grid has been filled with the a specific type of geo-object.

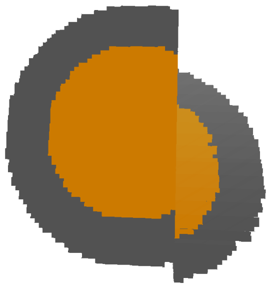
Stacking

A geo-object defined in Section 2.1 can be stacked by defining the number of geo-objects to stack and the distance in the x, y and z directions between each of the geo-object in the stack.

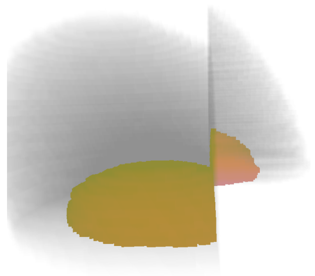
Two types of stacking are available. The first type is the single geo-object stack defined by the manager `SingleObjectStackerManager`, which consist of stacking the same type of geo-object. The second type of stacking of type `ObjectSequenceStackerManager` allows to define a sequence of geo-objects (not necessarily the same type) to be stacked. That sequence can



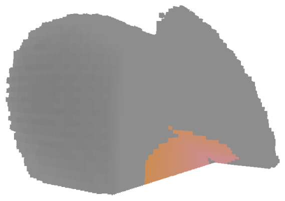
(a) EW-view



(b) Bottom view



(c) EW and NS cross-sections



(d) EW and plan cross-section

Figure 4: View of faulted carbonate mound with displacement.

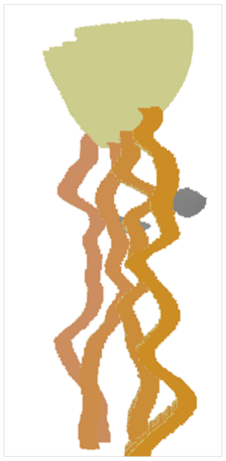
be repeated.

Positioning

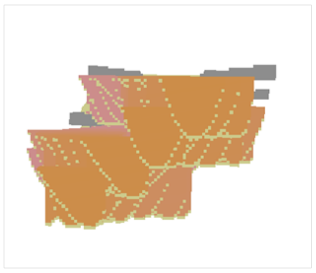
The positioning partly controls where the geo-objects are rasterized on the grid. Currently, three type of positioning options are available and each one can also be used with an intensity field:

`Random` Selection of random locations on the grid

`BottomToTop` Choose a random location in the plane (x,y) and place the object as low as possible until it either touches an already drawn geo-object or reaches the bottom of the grid. This positioning fills the grid with geo-objects from the bottom up. This positioning does not


(a) EW-view

(b) Bottom view


(c) EW and NS cross-sections

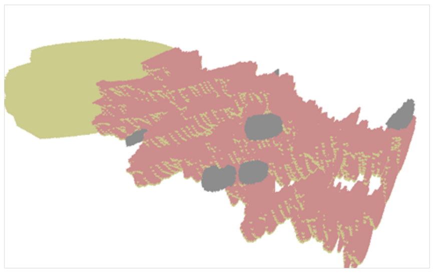

(d) EW and plan cross-section

Figure 5: Geo-objects generated with the manager.

deform the geo-objects. Figure 6 shows an example of three-element geo-objects with the `BottomToTop` positioning option.

`BottomToTopDrape` This object is similar to the `BottomToTop` but the geo-objects are deformed to drape over the existing geo-objects in the grid. This is to mimic a geo-objectfalling from the top to the bottom of the grid and conforming itself to the existing topology.

Each of these positionings can also be coupled with an intensity field which consists of a grid property with values between zero and one. The intensity values correspond to the probability of acceptance if that location is randomly selected. Given an intensity field $I(v_i), i = 1, ..., N$, with N the size of the grid, the probability of a location v_i being selected is

$$Pr\left(v_i \text{ being chosen}\right) = \frac{I\left(v_i\right)}{\sum_{j=1}^{N} I(v_j)} \tag{1}$$

The intensity field for the `BottomToTop` and `BottomToTopDrape` only concerns the (x,y) location in the plane as the z-position of the object is function of the already drawn geo-objects.

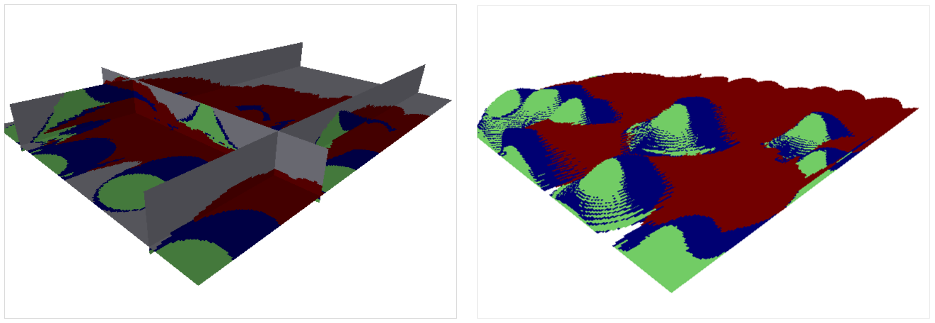


Figure 6: Geo-objects generated with the BottomToTop positioning option.

2.2.1 Interaction rules

Finally the exact location of a geo-object can also be partially controlled by the interaction rules between geo-objects. For instance, geo-object A may never overlap geo-object B. There are currently three interaction rules available:

No overlap Indicates that two geo-object types never overlaps. It is possible for a geo-object to not overlap with its own type.

Attachment Indicates that two geo-object types should always touch one another. Geo-object B is said to be attached to geo-object A if at least one voxel but less than 25% of the volume of geo-object B overlaps with geo-object A. One geo-object may be attached with its own type.

Full overlap Geo-object B is said to fully overlaps with geo-object A if 100% of the volume of geo-object B overlaps with geo-object A. A geo-object cannot fully overlaps with its own type.

Note that the interactions may not be consistent with the `BottomToTop` and `BottomToTopDrape` positioning options.

2.2.2 Algorihm for a simulation event

The algorithm for positioning a geo-object under constraint is:

```
1: Initialize a random path
2: while not found a valid location do
3:     Get a location $v_i$
4:     if using an intensity field then
5:         if location $v_i$ is refused with a probability $1 - I(v_i)$ then
6:             Try next location along the random path
7:         end if
8:     end if
9:     Create the geo-object centered at $v_i$
10:    for each interaction rule do
11:        Check if the current geo-object is valid considering the previously
           simulated geo-objects.
12:    end for
13:    if geo-object violates at least one interaction rule then
14:        Delete geo-object
15:    else
16:        Draw the geo-object
17:        Update the stopping criterion
18:    end if
19:    if Not a found a potential location having visited three times all the
       voxels then
20:        Stop and exit: convergence failed
21:    end if
22: end while
```

Note that this is a greedy algorithm and may generate inconsistencies when many criteria are combined, see Lantuejoul (2002) for more details on boolean models.

2.3 Parameterization

A flexible parameterization of the simulation process (positioning, stacking and interactions) and the geo-object geometry is key in having a program that can build training images for a wide variety of geological environments.

All geo-object parameters such as the dimension of the shapes and the rotation angles can either be constant or be drawn from a triangular, exponential or uniform distribution. Any value, be it a constant or a parameter of a distribution, can either be constant or vary in space. A parameter that can vary in space is termed a locally varying parameter (LVP). A LVP is provided through a grid property. See Section 4 for an example of locally varying properties.

3 Interfaces

The various Tetris user interfaces shield the user from most of the algorithm and design complexities by providing intuitive and easy to use widgets to build the geo-objects. The main interfaces are designed such that each part of the algorithm, be the geo-objects, the elements or the simulation events, can be saved and loaded from a file. Once a complex geo-object or element structure has been build, it can be saved and stored in a database to be used again when needed.

The interface is shown in Figure 7. The parameters associated with the numbered bullets in Figure 7 are:

1. Number of geo-objects to parameterize.

2. Remove the geo-object from the list.

3. Save the geo-object to a file.

4. Load a geo-object from a file.

5. Button to parameterize the geo-object. When pressed a new dialog window (see Figure 8) appears to enter the geo-object parameters.

6. Type of geo-object (see Section 2.1).

7. Number of simulation event to parameterize.

8. Remove the simulation event from the list.

9. Save the simulation event to a file.

10. Load a simulation event from a file.

11. Button to parameterize the simulation event. When pressed a new dialog window (see Figure 9) appears to enter the simulation event parameter.

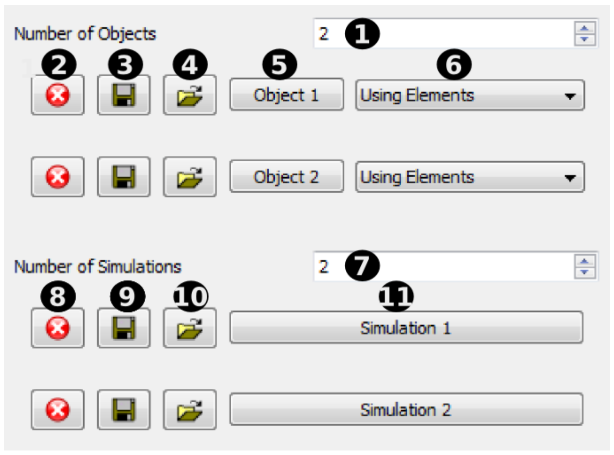


Figure 7: Tetris main interface to parameterize the geo-objects and the simulation events

The interface to build a geo-object from a set of elements is shown in Figure 8. The parameters are:

1. Name of the geo-object.

2. Number of elements to build the geo-object.

3. Remove the element from the list.
4. Save the element to a file.
5. Load an element from a file.
6. Button to parameterize the element. When pressed a new dialog window appears to enter the shape(s) parameters.
7. Number of element operations to parameterize.
8. Remove the element operation from the list.
9. Move the element operation up the execution order.
10. Move the element operation down the execution order.
11. Parameterize the operation (difference or intersection).

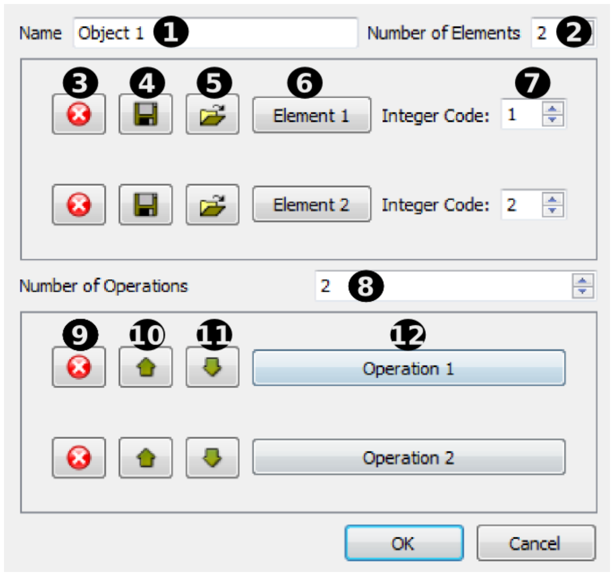


Figure 8: Tetris element interface to build object from the ObjectFromElementTreeManager geo-object manager.

The interface to simulate a geo-object is shown in Figure 9. The parameters are:

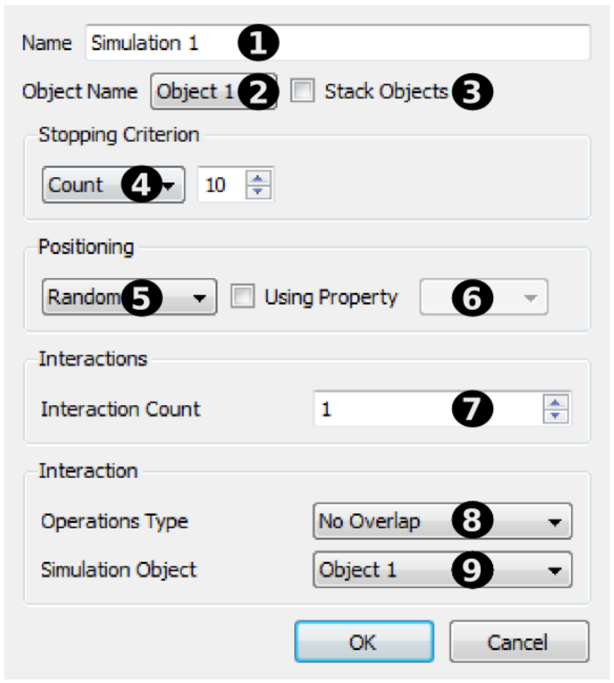
1. Name of the simulation event.
2. Name of the geo-object to be simulated.
3. Stacking option.
4. Stopping criterion (count or proportion).
5. Positioning option
6. Option to use an intensity field for the preferential location of the geo-objects.
7. Number of interaction rules.
8. Type of interaction rule (No Overlap, Attach, Full Overlap).
9. Geo-object type to apply the interaction rule.

4 Examples

4.1 Cross-correlated parameters

By using multiple cross-correlated LVPs it is possible to build training images depicting complex morphological behaviors. Take for instance Figure 10 representing a series of mound geo-objects with dependent size, shear intensity and rotation angles. Whereas steeper angles correlate with increasing shearing, decreasing outer envelope sizes and rapidly decreasing core sizes. First, four LVP properties are created (Figure 10(a) - (d)); the relationships between these geo-object properties are clearly visible. The shear intensity follows the rotation angles trend, while the sizes of both elements are inversely proportional to the absolute value of the rotation angles.

The resulting realization shows larger mounds in the center and smaller mounds toward the extremities. As the mounds are tilted, the length of the tails (created by shearing) increases in the direction of the tilt mimicking the deposition of sediment downslope.


Name Simulation 1 1

Object Name Object 1 2 Stack Objects 3

Stopping Criterion

Count 4 10

Positioning

Random 5 Using Property 6

Interactions

Interaction Count 1 7

Interaction

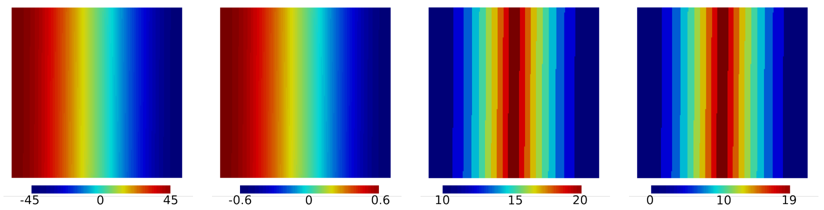
Operations Type No Overlap 8

Simulation Object Object 1 9

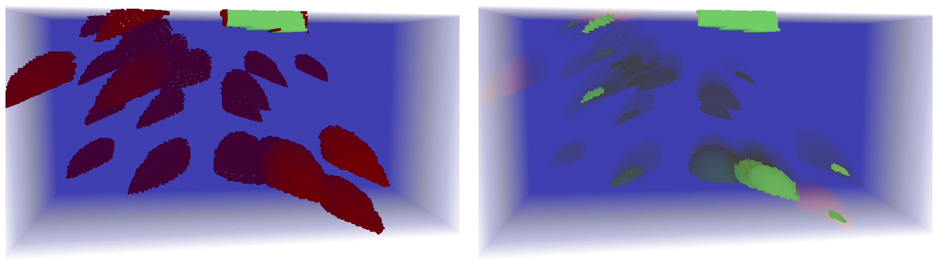
Figure 9: Tetris simulation event interface to simulate a geo-object.

4.2 Geo-objects interactions

Figure 11 shows an example of three geo-objects with specific attachment interactions. It consists of two sets of fractures, North-South (blue) and East-West (yellow) and faults (red). All geo-objects are modeled with the cuboid shape. The planes and the N-S fractures systematicaly intersect the E-W set of fractures. That simulation was performed using three simulation events: (1) simulate the E-W fractures, (2) simulate the N-S fractures with an attachment rule with the E-W fractures and (3) simulate the plane also with an attachment rule with the E-W fractures.



(a) Rotation angles (plan view)
(b) Vertical shear intensity (plan view)
(c) Size of the outer shell (plan view)
(d) Size of the inner core (plan view)

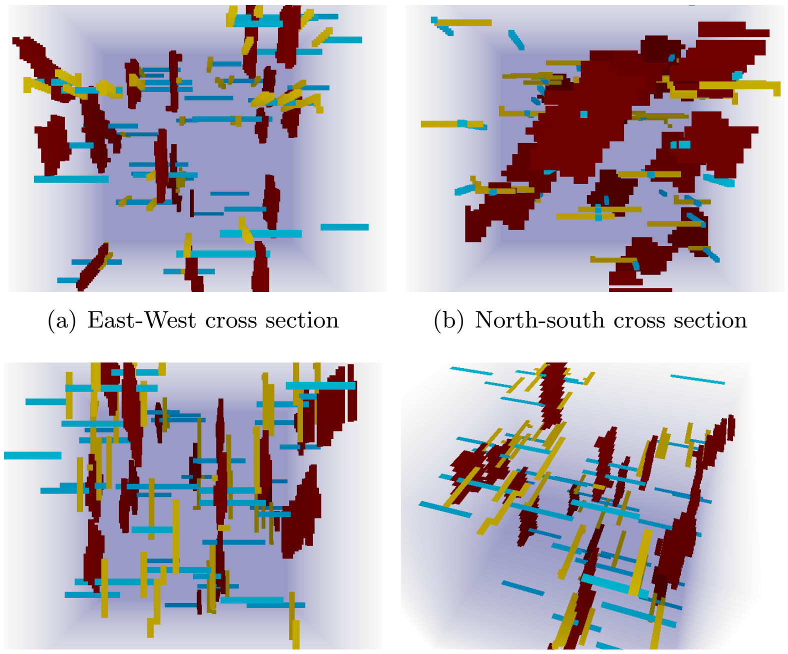


(e) One simulation of the training image (side view)
(f) View of the simulated inner core (side view)

Figure 10: Carbonate mound simulated with cross-correlated spatially varying parameters.

5 Conclusion

The Tetris plugin is a flexible boolean simulation tool to generate training images. It can generate complex geo-objects with complex trends. However, generating an image with the program *does not* guarantee good reproduction when used with a multiple-point simulation algorithm. Not all realizations from the Tetris SGeMS plugin can systematically be used as training images. The readily available version of snesim (Strebelle, 2002), or filtersim (Zhang et al., 2006; Wu et al., 2008) available on SGeMS or Petrel cannot (and should not) be used with training images containing trends. However, the new generation of training image-based algorithms (de Vries et al., 2008;



(a) East-West cross section (b) North-south cross section

(c) Plan view (d) Global view

Figure 11: Fractures and fault representation. The N-S fracures (blue) always intersect the E-W fracures (yellow). The planes (red) also systematicaly intersect the E-W set of fractures (blue).

Boucher, 2009; Chugunova and Hu, 2008; Mariethoz, 2009) can handle very complex training images by considering trends.

The flexible design also allows to extend the plugin by adding new shapes, geo-object managers, distribution types, stopping criteria and positioning options. Adding these features requires writing C++ classes derived from the proper base class and new widgets for the user interface. The library Qt is used for the interface.

Finally, the plugin can be used for boolean simulation when soft data are available and no hard data conditioning is required. Enabling hard data conditioning would require new stopping criterion and positioning classes.

References

Boucher, A., 2009. Considering complex training images with search tree partitioning. Computers & Geosciences 35 (6), 1151 – 1158.

Chen, B., Kaufman, A., 2000. 3d volume rotation using shear transformations. Graphical Models 62, 308322.

Chugunova, T. L., Hu, L. Y., 2008. Multiple-point simulations constrained by continuous auxiliary data. Mathematical Geosciences 40, 133–146.

de Vries, L. M., Carrera, J., Falivene, O., Gratacs, O., Slooten, L. J., 2008. Application of multiple point geostatistics to non-stationary images. Mathematical Geosciences 41 (1), 29–42.

Deutsch, C. V., Tran, T. T., 2002. Fluvsim: a program for object-based stochastic modeling of fluvial depositional system. Computers & Geosciences, 525–535.

Deutsch, C. V., Tran, T. T., 2004. Simulation of deepwater lobe geometries with object based modelling: Lobesim. Tech. rep., University of Alberta. URL `http://www.uofaweb.ualberta.ca/ccg//pdfs/1999%2004-LobeModeling1.pdf`

Deutsch, C. V., Wang, L., 1996. Hierarchical object-based stochastic modeling of fluvial reservoirs. Mathematical Geology 28 (7), 857–880.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. M., 1994. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional.

Lantuejoul, C., 2002. Geostatistical Simulation: Models and Algorithms. Springer-Verlag, Berlin, Germany.

Maharaja, A., 2008. Tigenerator: Object-based training image generator. Computers & Geosciences 34 (12), 1753 – 1761.

Mariethoz, G., 2009. Geological stochastic imaging for aquifer characterization. Ph.d. dissertation, University of Neuchtel, Switzerland.

Pyrcz, M., Boisvert, J., Deutsch, C., 2009. Alluvsim: A program for event-based stochastic modeling of fluvial depositional systems. Computers & Geosciences 35 (8), 1671 – 1685.

Remy, N., Boucher, A., Wulf, J., 2009. Applied Geostatistics with SGeMS: A User's Guide. Cambridge University Press.

Strebelle, S., 2002. Conditional simulation of complex geological structures using multiple-point statistics. Mathematical Geology 34 (1), 1–21.

Tetzlaff, D. M., Harbaugh, J. W., 1989. Simulating Clastic Sedimentation. Van Nostrand Rheinhold, New York.

Wu, J., Boucher, A., Zhan, T., 2008. A sgems code for pattern simulation of continuous and categorical variables: Filtersim. Computers & Geosciences 34 (12), 1863–1876.

Zhang, T., Journel, A. G., Switzer, P., 2006. Filter-based classification of training image patterns for spatial simulation. Mathematical Geology 38 (1), 63–80.