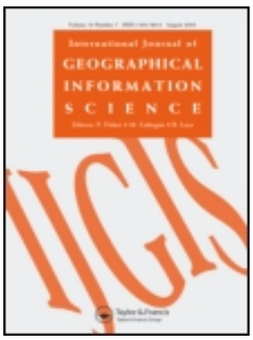

International Journal of Geographical Information Science

ISSN: 1365-8816 (Print) 1362-3087 (Online) Journal homepage: https://www.tandfonline.com/loi/tgis20

Modelling landscape dynamics with Python

D. Karssenberg , K. de Jong & J. van der Kwast

To cite this article: D. Karssenberg , K. de Jong & J. van der Kwast (2007) Modelling landscape dynamics with Python, International Journal of Geographical Information Science, 21:5, 483-495, DOI: 10.1080/13658810601063936

To link to this article: https://doi.org/10.1080/13658810601063936

International Journal of Geographical Information Science
Vol. 21, No. 5, May 2007, 483–495

Taylor & Francis
Taylor & Francis Group

Research Article

Modelling landscape dynamics with Python

D. KARSSENBERG*, K. DE JONG and J. VAN DER KWAST
Department of Physical Geography, Faculty of Geosciences, Utrecht University, PO Box 80115, 3508 TC Utrecht, The Netherlands

(*Received 1 November 2005; in final form 23 May 2006*)

A new tool for construction of models is presented that allows earth scientists without specialist knowledge in programming to convert theories to numerical computer models simulating landscape change through time. This tool, referred to as the PCRaster Python library, consists of: (1) the standard Python programming language, which is a generic, interpreted scripting language, supporting object oriented programming; (2) a large set of spatial and temporal functions on raster maps that are embedded in the Python language as an extension; (3) a framework provided as a Python class to construct and run iterative temporal models and to calculate error propagation with Monte Carlo simulation; and (4) visualization routines to display spatio-temporal data read and written by this framework. Python is a high-level programming language, and users of the tool do not have to be specialist computer programmers. Users of the PCRaster Python library can take advantage of several other Python libraries, such as extensions for matrix algebra and for modelling in three spatial dimensions.

Keywords: Dynamic modelling; Monte Carlo simulation; Error propagation modelling; Python; PCRaster; Environmental modelling

1. Introduction

Numerical environmental models simulating landscape change through time with equations representing physical, chemical, or biological landscape processes are one of the cornerstones of current research in physical geography and other earth sciences (Wainwright and Mulligan 2004). An environmental model considers the landscape as a system whereby each system component is described using process equations derived from scientific theory related to that component (Beck *et al.* 1993, Jørgensen and Bendoricchio 2001). As such, an environmental model carries and integrates a large body of scientific knowledge which can be tested against field data by running the model for a specific study area, comparing modelled and measured system variables at the landscape scale. Thus, models can be used to test theory and field data, in a way which would be impossible without them, since theories embedded in a system description are tested. In applied research, environmental models fed with digital data from earth observation satellites or automatic data loggers are relevant to understand and predict future changes under different scenarios of human-induced change (Giupponi *et al.* 2007).

*Corresponding author. Email: d.karssenberg@geo.uu.nl

International Journal of Geographical Information Science
ISSN 1365-8816 print/ISSN 1362-3087 online © 2007 Taylor & Francis
http://www.tandf.co.uk/journals
DOI: 10.1080/13658810601063936

Since progress in science involves adjustment of theories of environmental processes and testing these theories against increasing amounts of empirical data, models need to evolve continuously. Thus, earth scientists need programming tools that make construction of new models as easy as possible, while these tools should allow for modification of models without the need to reprogram everything from scratch (Karssenberg 2002).

Just like state-of-the-art equipment in a laboratory is essential for research, efficient model construction tools can be considered very important in the earth sciences. The main requirement of such a tool is its capability to construct all types of models used in the discipline where it is applied, which means that for most applications in earth sciences, the tool needs to support two- and three-dimensional spatial modelling through time, using model equations derived from spatio-temporal landscape processes. Since earth scientists, and not software engineers, have the knowledge of environmental processes that need to be embedded in the models, it is required that the tool can be used by earth scientists who have in most cases no specialist knowledge in programming. In addition, Karssenberg (2002) claims that tools for model construction in the earth sciences should support reuse of program code, should provide a generic approach to model construction, need to hide technical details from the user, should result in short development time, need to minimize programming errors, and should be easy to learn, while computer run times should be minimal.

The range of existing tools for environmental model construction is large, and most tools are very useful, although none of these fulfil all requirements given above (Karssenberg 2002). For example, technical computer languages such as MATLAB (2005) are useful thanks to their large range of generic functions that can be used for model construction in many disciplines of the earth sciences, but these languages do not provide a generic approach to temporal modelling while embedded coupling with Geographical Information Systems (GIS) providing data and visualization is not provided. Non-spatial modelling tools with graphical interfaces (MODELMAKER 2005, STELLA 2005) can be used without any programming knowledge, but do not come with functions for modelling spatial interaction. Environmental modelling languages embedded in GIS (PCRaster 2006) are tailored to spatio-temporal modelling in the earth sciences, although their application domain is limited to two-dimensional modelling with a fixed set of spatial functions which is not easy to extend. Most models can be programmed from scratch with system programming languages (C++, Fortran), but these languages are difficult for non-programmers to use, while a generic approach to modelling is not provided (Karssenberg *et al.* 2001, Karssenberg 2002).

In this paper, a new tool for construction of environmental models is presented. This tool is based upon a generic programming language, but additional functionality has been added specifically meant for environmental modelling. In section 2, we describe the different components of the tool providing some application examples. Section 3 explains how this tool can be extended if additional functionality is needed which is not embedded in the standard tool. In section 4, we evaluate the value of this tool for constructing models simulating landscape dynamics.

2. PCRaster Python library

2.1 *Adding spatial and temporal functions to the Python language*

Python is a generic, interpreted scripting language, supporting object-oriented programming (Python 2005). First released in 1991, it is a relatively young

programming language that is continuously being improved and extended, and is distributed for free. In recent years, the number of users has grown significantly, and several textbooks have been written on the language (e.g. Downey *et al.* 2002, Langtangen 2004). Compared with other scripting languages like Perl and system programming languages such as C++ or Fortran, Python has a very simple syntax that reads almost like pseudo-code. As a result, it is relatively easy to learn, as illustrated by the use of the language to teach programming at schools for elementary education.

The simple syntax of Python is certainly appealing to researchers who want to construct environmental models, because it results in a language that is easy to learn for people without specialist knowledge of programming, while the Python language completely hides technical details with which users of system programming languages need to concern themselves. But a simple syntax alone does not make a programming language an ideal tool for constructing environmental models. In addition, data structures and language constructs specifically meant for spatio-temporal modelling are needed, and visualization tools for spatio-temporal data are required. So, the Python interface is a very useful language for generic programming by non-specialists, but it is not tailored to construction of environmental models. The approach we follow in this project is to add components to the Python language that make it more specifically useful for the construction of spatio-temporal models. The result is a tool consisting of the standard Python language bundled with these extra components that are available to the model builder. Since many components are taken from the PCRaster software (PCRaster 2006), we refer to this tool as the PCRaster Python library. We added the following components to standard Python:

- data structures containing spatial and temporal data and functions useful for environmental modelling operating on these data structures;
- a script framework that can be used for temporal modelling and error propagation modelling through Monte Carlo simulation;
- an interactive visualization module that invokes spatio-temporal stochastic data read or written by models created with this PCRaster Python library.

These components are added as Python and C++ code, where C++ code is embedded in Python as a Python extension module. To provide functionality for field-based modelling on raster maps, a data structure representing raster maps with a constant cell size and standard functions operating on these maps were added. The C++ code required for this was largely taken from the existing PCRaster PCRcalc software (van Deursen 1995, PCRaster 2006), which means that all functions available in the PCRcalc software are also available in the PCRaster Python library. This set of functions includes a large range required for environmental modelling: point functions, direct neighbourhood functions, entire neighbourhood functions, functions with a neighbourhood defined by a local drain direction map with a single flow direction per cell (Burrough and McDonnell 1998) and functions calculating descriptive statistics over areas (Karssenberg and de Jong 2005a, b). Since PCRaster uses strict data typing using six different data types for map data (van Deursen 1995), this is also applied for map data in the PCRcalc Python extension. In addition to the functions from the PCRaster code, functions on maps programmed in Python were added, calculating statistics of Monte Carlo samples (Karssenberg and de Jong 2005b).

The PCRcalc Python extension, developed with Boost.Python (Boost, 2005), makes the PCRcalc functions callable from Python. Our goals are to keep the

amount of code for the interface between Python and the C++ code small, to prevent code duplication of C++ code in Python and to leave room for optimization of models. As a result, the interface code does nothing more than 'tell' the PCRcalc library which function should be called and what arguments to pass, and return the result. So, for example, when a PCRcalc function is called from Python, all error checks on the arguments are done by the same C++ code as used in the stand-alone PCRcalc version. Furthermore, PCRcalc might decide to delay executing point operations until a neighbourhood operation is called for optimization purposes.

With these functions on raster maps embedded, Python becomes a powerful tool for Map Algebra and static modelling. The syntax is similar to mathematical notation, as shown in the example script of a static hydrological model in table 1. The first line makes everything from the PCRaster Python library available to the script. The first argument of the function **accuflux** is a local drain direction map that is directly read from disk, just like `rain.map`, which is a map containing the amount of rainfall. The **accuflux** function calculates a map with the discharge resulting from the amount of rainfall and assigns it to `discharge`. The third line uses the operator 'greater than' assigning a Boolean TRUE to cells with a value on `discharge` greater than ten and FALSE otherwise. To write maps in memory to the hard disk, the **report** function is used, which is done for the map `highDischarge`, stored under the filename `highdis.map`, in the last statement. All other functions and operators on maps have a syntax similar to these in the example.

The example in table 1 is a static script that is interpreted by Python and run from beginning to end. Python can also be used in interactive mode, by executing individual commands at the Python command prompt, which is very useful for teaching, since spatial analyses can be done step by step, displaying and checking results after each step.

2.2 *Providing a framework for stochastic dynamic modelling in Python*

In addition to standard functions on maps, a tool for the construction of environmental models needs to provide a framework for modelling changes through time, iterating spatial functions over time steps (van Deursen 1995, Wesseling *et al.* 1996, Karssenberg and de Jong 2005a). For modelling error propagation in these models, an additional iteration over Monte Carlo samples is required (Pebesma *et al.* 2000, Karssenberg and de Jong 2005b). Although developers of models could program these iterations from scratch with standard Python language constructs, it is expected that this will be difficult for them, since nested loops are required, and multidimensional data (space, time, Monte Carlo loops) need to be read from and written to hard disk. To solve this problem, the PCRaster Python library provides a framework within which model builders can construct their dynamic models, calculating error propagation if required. This framework has the additional

Table 1. Example of a static hydrological model[a].

```
from pcraster import *
discharge=accuflux('ldd.map', 'rain.map')
highDischarge=discharge > 10.0
report(highDischarge, 'highdis.map')
```

[a]Keywords are shown in bold.

advantage that it provides a generic approach to the common problem of iterations through time and through Monte Carlo loops, which improves the readability of programs exchanged between scientists, while data visualization tools can be tailored to data read and written by this framework, as illustrated below.

The framework provided in the PCRaster Python library contains a class named **ModelScript**. This class consists of components written by the authors in Python, and it is included in the standard distribution of the PCRaster Python library. In the first line of the example script in table 2, the class definition is imported to the Python script, together with all other components of the PCRaster Python library, just like in the script in table 1. The **ModelScript** class allows the model builder to define four different functions (**premcloop**, **initial**, **dynamic**, and **postmcloop**) in the program of their model, building the model by providing a list of map operations in each of these functions. The functions on maps in each of these **ModelScript** functions are executed in the order:

```
run functions in premcloop
for m in Monte Carlo sample:
  run functions in initial
  for t in timesteps:
    run functions in dynamic
run functions in postmcloop
```

As described in more detail by Karssenberg and de Jong (2005b), functions in the **premcloop** represent variables with a value that is exactly known and fixed

Table 2. Example of a stochastic dynamic model[a].

```
from pcraster import *

nrSamples=1000
nrTimeSteps=36

class DischargeModel(ModelScript):
  def __init__(self, argv):
    ModelScript.__init__(self,argv,nrSamples,nrTimeSteps)

  def premcloop(self):
    self.ldd='ldd.map'
    self.report(self.Ldd, 'ldd')

  def initial(self):
    self.infCapacity=max(8+mapnormal() * 4.0,0.0)
    self.report(self.infCapacity, 'ic')

  def dynamic(self):
    rain=timeinputscalar('rain.tss',boolean(1))
    discharge=accuthresholdflux(self.ldd, rain,
    self.infCapacity)
    self.report(discharge, 'dis')

  def postmcloop(self):
    reportpercentiles('dis', nrSamples, [0.25,0.50, 0.75],
                                        timeSteps=nrTimeSteps)

sys.exit(Model(sys.argv).run())
```

[a]Keywords are shown in bold. Note that Python uses indentation instead of braces used in some other languages to group statements.

through time. Functions in the **initial** represent the initial state of the model at $t=0$, given as realizations of stochastic variables. Functions in the **dynamic** represent the change in state of the variables for each time step. The **postmcloop** is used to calculate descriptive statistics of variables over the time steps or over the Monte Carlo loops.

The upper part of the script built by the model developer needs to define the number of Monte Carlo samples and the number of time steps (table 2). Below this, a class **DischargeModel** is defined, with a given initialization. This part of the script does not need to be changed by the model builder, since it is always the same. Below this fixed part, the model builder has given a statement that reads a local drain direction map from disk assigning it to the variable `self.Ldd`. Since the local drain direction is fixed through time and over the Monte Carlo loops, this is done in the **premcloop**. The **initial** defines an infiltration capacity map as a stochastic variable. The function **mapnormal** generates a realization of a random value from a normal distribution with unit variance. This is a very simple error model. Realizations of more advanced, and more realistic, error models could be created with a call to a geostatistical software package, like Gstat (2005). The **timeinputscalar** function in the **dynamic** reads an amount of rainfall for each time step from the time series file `rain.tss`. The **accuthresholdflux** function calculates for each time step the amount of runoff generated by rainfall, using an infiltration threshold given by a realization of `self.infCapacity`. The function in the **postmcloop** reads the discharge maps stored in the dynamic and calculates percentiles of the discharge, for each cell.

In addition to providing a standard script structure for iterative scripts, the class **ModelScript** organizes storage of files done by the report function. Storing a file with report in the premcloop stores the file in the run directory, while files reported in the **initial** and the **dynamic** are stored in a separate subdirectory for each Monte Carlo loop, using time step numbers embodied in the file name when the report is done in the **dynamic**. Spatio-temporal data stored in this way can be displayed with the PCRaster Python extension visualization software without explicitly providing the location of the variables on disk. For instance, an animation of the first, second, and third quartiles of discharge can be shown by typing:

```
aguila --percentiles=[0.25,0.50,0.75] dis
```

The visualization software will show an animated map with a quartile and a timeseries of the three quartiles (figure 1). The use of the Aguila software included in the PCRaster Python distribution to visualize stochastic data is further explained in (Pebesma and de Jong this issue).

A large example application of the PCRaster Python library is a spatio-temporal soil moisture model which integrates a soil-water balance model and a remote-sensing-based energy flux model based on SEBS (Su 2002) by means of data assimilation. This model is currently being developed for a semi-arid region near Rabat (Morocco), where an accurate prediction of soil moisture patterns in space and time is important for water-management practices. A three-layer soil water balance model is constructed, based on gravitational flow. Using a Kalman-filter data-assimilation algorithm, the soil moisture is corrected by comparing the evapotranspiration flux of the soil-water model and of SEBS. SEBS is an analytical physically based model, which calculates the turbulent heat fluxes from

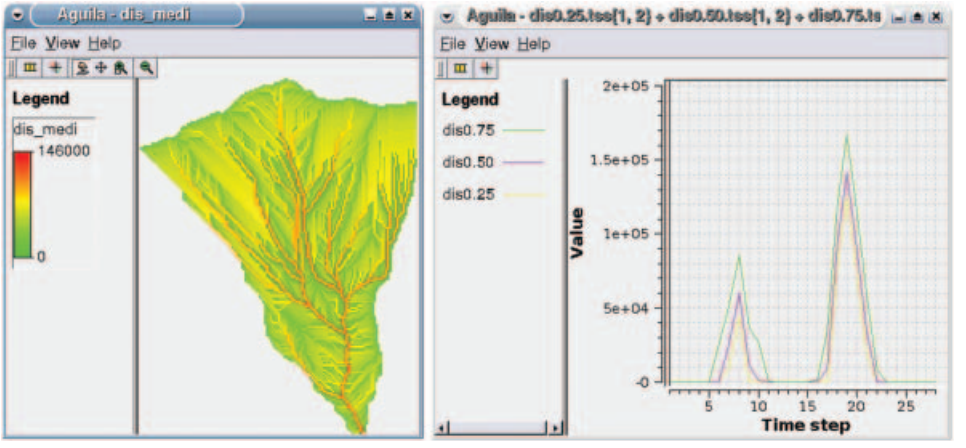

Figure 1. Visualization of stochastic variable `dis` with the Aguila visualization software. Left: map showing median of the discharge; right: time series showing first quartile, median, and second quartile.

remote-sensing imagery and meteorological data. The PCRaster Python version of SEBS has been applied to an area in Morocco (van der Kwast and de Jong 2004) and Spain (Timmermans *et al.* 2005), and validated with field measurements of energy fluxes. Figure 2 shows an output.

2.3 *Python language features to improve program quality*

Obvious quality aspects of model scripts (and software in general) are correctness, completeness, and documentation. Another quality aspect is the readability of the source code. A readable model is relatively easy to understand, maintain, and share. Some aspects influencing readability are the style used to write it, whether or not logically related statements are grouped together and whether duplicate code is

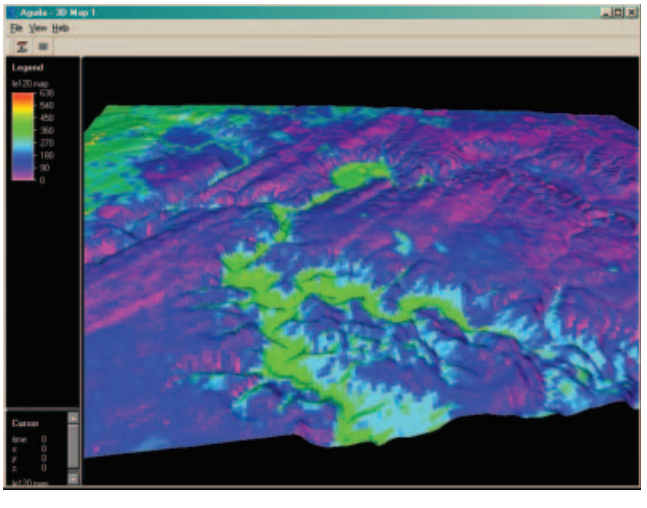

Figure 2. Aguila visualization of latent heat flux derived from a Landsat TM satellite image of Morocco draped over a digitial elevation model from the Shuttle Radar Topography Mission (Rabus *et al.* 2003).

Table 3. Script shown in table 1 with code encapsulated in the function `floodedAreas` defined by the model builder.

```
from pcraster import *

def floodedAreas(rainfall,ldd,floodingThreshold):
  discharge=accuflux(ldd, rainfall)
  highDischarge=discharge>floodingThreshold
  return highDischarge

highDischarge=floodedAreas('rain.map','ldd.map',10.0)
report(highDischarge,'highdis.map')
```

avoided. Pythons language features like control flow constructs, functions, built-in types, and classes can be used to turn a hard-to-read and long list of model statements, requiring a lot of documentation to be able to understand the code, into a readable model which needs minimal documentation.

Table 3 shows how the script in table 1 can be rewritten by defining a new function `floodedAreas` containing two statements calculating discharge and areas above a threshold discharge. Functions are defined in Python using the `def` keyword, as shown in table 3. At the bottom of the script, this function is called, using three input arguments as defined in the function definition. The advantage of embedding code in a function is that it improves the readability of the model script, while the code inside the function can be reused several times in the script, by calling the same function at several places in the script.

Another example of using Python language features to improve program quality is given in the scripts in Tables 4–6. Table 4 is a piece of a simplified model to simulate water uptake by roots from the soil represented by four superimposed soil layers. The code has no shareable parts, and it does not scale well to larger model scripts: a simulation with more than four soil layers would require the script to be modified by adding model code for each layer separately. By adding control flow statements, this code fragment can be refactored to the code in table 5. Here, an ordered list is created where each element of the list is a map. By iterating over the list elements, which is done in the `for. .` statements, each of the maps is assigned

Table 4. Script for simulating evaporation from four soil layers.

```
# Initialise moisture content
moisture1='moisture1.map'
moisture2='moisture2.map'
moisture3='moisture3.map'
moisture4='moisture4.map'

# Initialise evaporation
evaporation1='evaporation1.map'
evaporation2='evaporation2.map'
evaporation3='evaporation3.map'
evaporation4='evaporation4.map'

# Calculate new moisture content
moisture1=moisture1 - evaporation1 * moisture1
moisture2=moisture2 - evaporation2 * moisture2
moisture3=moisture3 - evaporation3 * moisture3
moisture4=moisture4 - evaporation4 * moisture4
```

Table 5. Script in table 4 rewritten by storing moisture content and evaporation in Python lists[a].

```
moisture=[ ]
evaporation=[ ]
nrLayers=4

# Initialise water content and evaporation
for i in range(1, nrLayers+1):
  moisture.append('moisture%d.map' % (i))
  moisture.append('evaporation%d.map' % (i))

# Calculate new moisture content
for i in range(1, nrLayers+1):
  moisture[i]=moisture[i] - evaporation[i] * moisture[i]
```

[a]Each list element consists of a PCRaster map. The code in the first **for** loop reads files from harddisk appending each file as a map to the list. The second **for** loop updates soil moisture by subtracting evaporation for each layer.

an initial soil moisture content and an updated soil water content, respectively. Changing the discretization of the soil from four layers to nine is a matter of replacing the 4 with a 9. In the previous version in table 4, this would require making the model script three times longer. The next step would be to rewrite the code using functions, as is done in table 6. Now, we also have achieved shareability, because another person is now able to call the implementation of the infiltration function. Furthermore, we have achieved the readability while we have actually removed two lines with comments. Choosing apt names for functions and creating lists of high level function calls often renders some documentation to be redundant. Since global variables are no longer needed, the reader can examine the implementation of an individual function by looking only at that function.

Another quality aspect of a model is its testability. It can be useful to be able to automatically test various parts of a model after it has been changed. Python contains a module named unittest that can be used for this purpose.

Table 6. Script in table 5 rewritten using three functions, defined with **def**.

```
def initialiseSoilLayers(nrLayers):
  layers=[ ]

  for i in range(1, nrLayers+1):
    moisture.append('moisture%d.map' % (i))
    evaporation.append('evaporation%d' % (i))

  return moisture, evaporation

def evaporate(moisture, evaporation):
  for i in range(0, len(moisture)):
    moisture[i] -=evaporation[i] * moisture[i]

  return soilMoist

def runModel( ):
  moisture, evaporation=initialiseSoilLayers(4)
  moisture=evaporate(moisture, evaporation)

runModel( )
```

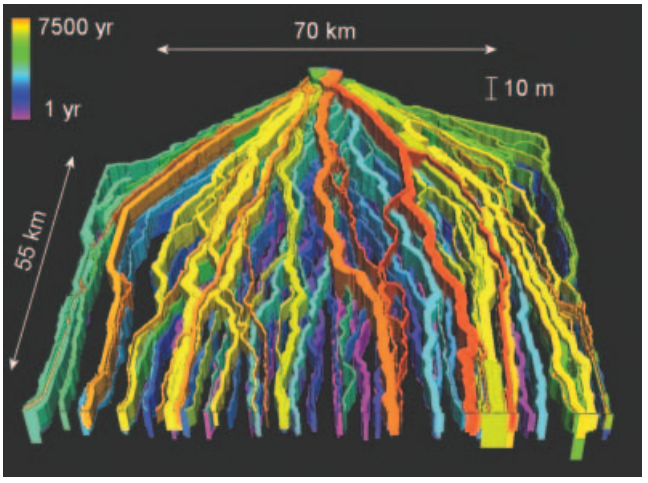

Figure 3. Deposits of channel belts (colours) in an alluvial fan generated by a process-based sedimentation and erosion model. Colours represent the model time at the moment of deposition. Flow was from top to bottom with inflow point at the top-centre.

3. Extendibility beyond the PCRaster Python library

3.1 *Creating other Python extensions or using existing extensions*

Other Python extensions are available that are very useful when applied in combination with the PCRaster Python library. Although still under development, the Python extension for modelling in three spatial dimensions developed according to the concepts described in Karssenberg and de Jong (2005a) can be used to construct static and dynamic models of landscape processes in three dimensions, particularly in the field of hydrology, sedimentology, ecology, and soil science. Using three-dimensional voxels with variable thickness and a constant discretization of the lateral dimensions, it can be used in combination with the functions on maps in the PCRaster Python library, since data from maps can be written to three-dimensional blocks and vice versa without resampling. This approach was followed in a project constructing a process-based model of sedimentation and erosion in a fluvial environment, storing the resulting stratigraphy in a three-dimensional block of data (Karssenberg and Bridge 2005) (figure 3).

The Numarray extension (Numarray 2005) provides a large set of operations on multidimensional matrices. Since the PCRaster Python library comes with conversion functions between maps used in the library and the matrices used in the Numarray module, the functions available in Numarray can be used in combination with the functionality provided by the PCRaster Python library. An example is given in table 7 of a script that uses the Numarray function **correlate** to apply a horizontal edge enhancement filter on a land-use map with continuous cell values, which could be reflection values retrieved by remote sensing. The second argument of the **correlate** function defines a 3×3 window with weights that need to be applied in the filter, given as a nested list. Using this approach, many filters used in image processing or cellular automata modelling can be defined by the model builder.

Reading and writing maps from a script written with the PCRaster Python library is done with the support of the GDAL Open Source library (GDAL 2005). GDAL can read many proprietary and generic raster formats.

Table 7. Script using the Numarray extension for edge enhancement on a landuse map[a].

```
from pcraster import *
from numarray.nd_image import correlate

...

landUseArray=pcr2numarray(landUseMap)
edgeArray=correlate(landUseArray,[[-1,-1,-1][1,-2,1][1,1,1]],
mode='constant')
edgeMap=numarray2pcr(edgeArray)
report(edgeMap,'edge.map')
```

[a]At the top of the script, the **correlate** function is imported from Numarray. The functions **pcr2numarray** and **numarray2pcr** convert between the PCRaster and Numarray formats.

In addition to the existing Python extensions, researchers can add their own functions written in Python, for instance when field based models need to be linked to agent-based models. In this case, the agent-based modelling components could be written from scratch in Python.

4. Discussion

From the discussion of model construction tools by Karssenberg (2002), it emerged that the PCRaster environmental modelling language PCRcalc has several advantages over system programming languages for construction of hydrological models. Most of these advantages are due to the fact that the PCRaster environmental modelling language is developed specifically for environmental model construction, while system programming languages are generic programming languages. As a result, the PCRaster environmental modelling language is easier to use for environmental researchers with no knowledge of programming. The PCRaster Python library presented here has similar advantages to the PCRaster environmental modelling language, although it is more difficult to use for environmental researchers. Just like the PCRaster language, it uses building blocks—maps and spatial functions on maps—representing attributes, dimensions and processes that are familiar to environmental researchers. However, unlike the PCRaster language being completely tailored to environmental model construction, the PCRaster Python library is based upon a generic programming language, which means that the model builder needs to have at least some basic knowledge of programming, since more technical details need to be defined in the model script, while error messages might appear that are somewhat harder to understand for non-specialists. As a result, the user of the PCRaster Python library will need to invest some time and energy in learning the basics of the Python language for successful application of the PCRaster Python library in a modelling project. This is in many cases not wasted time, since the Python language is a general-purpose language which can be used in many other applications. The main strengths of the PCRaster Python library are its extendibility and scalability. Python has the possibility to embed additional functions and data structures when a modelling project requires functionality that is not available in the PCRaster Python library. This can be done (1) by using existing extensions, such as the extension for three-dimensional modelling described here, (2) by writing new extensions, or (3) by developing

modules written in Python. Especially the last option will be attractive to environmental researchers, since developing code from scratch in Python is much easier than doing so using a system programming language such as C++. Such code can be replaced with C++ code in a later stage of a project if this results in improvements of the model. In some cases, for instance, rewriting Python code in C++ may result in shorter run times.

As shown in this paper, Python provides language features that are very useful to construct large models while keeping the code readable. Models consisting of different components each with a specific meaning in a model can be split up in different script components that can be invoked in the main script.

Acknowledgements

Our grateful thanks go to Peter Burrough for his enthusiasm and support through the years. Cees Wesseling is acknowledged for developing components of the PCRaster Python library.

References

Beck, M.B., Jakeman, A.J. and McAleer, M.J., 1993, Construction and evaluation of models of environmental systems. In *Modelling Change in Environmental Systems*, M.B. Beck, A.J. Jakeman and M.J. McAleer (Eds), pp. 3–35 (New York: Wiley).

Boost 2005, Boost C++ libraries. Available online at: http://www.boost.org (accessed 15 October 2005).

Burrough, P.A. and McDonnell, R.A., 1998, *Principles of Geographical Information Systems* (Oxford: Oxford University Press).

Downey, A., Elkner, J. and Meyers, C., 2002, *How to Think Like a Computer Scientist: Learning with Python* (Wellesley, MA: Green Tea Press).

GDAL 2005, Geospatial Data Abstraction Library. Available online at: http://gdal.maptools.org (accessed 15 October 2005).

Giupponi, C.,Jakeman, A.J.,Karssenberg, D., and Hare, M.P. (Eds), 2007, *Sustainable Management of Water Resources: an Integrated Approach* (Cheltenham, UK: Elgar).

Gstat 2005, Gstat package. Available online at: http://www.gstat.org (accessed 15 October 2005).

Jørgensen, S.E. and Bendoricchio, 2001, *Fundamentals of Ecological Modelling* (Amsterdam: Elsevier).

Karssenberg, D., 2002, The value of environmental modelling languages for building distributed hydrological models. *Hydrological Processes*, **16**, pp. 2751–2766.

Karssenberg, D. and Bridge, J.S., 2005, A 3D model simulating sediment transport, erosion and deposition within a network of channel belts and an associated floodplain. In *8th International Conference on Fluvial Sedimentology*, 7–12 August, 2006, Delft, the Netherlands.

Karssenberg, D. and de Jong K., 2005a, Dynamic environmental modelling in GIS: 1. Modelling in three spatial dimensions. *International Journal of Geographical Information Science*, **19**, pp. 559–579.

Karssenberg, D. and de Jong K., 2005b, Dynamic environmental modelling in GIS: 2. Modelling error propagation. *International Journal of Geographical Information Science*, **19**, pp. 623–637.

Karssenberg, D., Burrough, P.A., Sluiter, R. and de Jong K., 2001, The PCRaster software and course materials for teaching numerical modelling in the environmental sciences. *Transactions in GIS*, **5**, pp. 99–110.

Langtangen, H.P., 2004, *Python Scripting for Computational Science* (Berlin: Springer).

MATLAB 2005, Available online at: http://www.mathworks.com (accessed 15 October 2005).

MODELMAKER 2005, Available online at: http://www.modelkinetix.com/modelmaker (accessed 15 October 2005).

NUMARRAY 2005, Numarray Python module. Available online at: http://www.stsci.edu/resources/software_hardware/numarray (accessed 15 October 2005).

PCRASTER 2006, PCRaster internet site. Available online at: http://pcraster.geo.uu.nl (accessed 1 January 2006).

PEBESMA, E.J. and DE JONG K., this issue, Interactive visualization of uncertain spatial and spatio-temporal data under different scenarios: an air quality example. *International Journal of Geographical Information Science*, **21**, pp. 515–527.

PEBESMA, E.J., KARSSENBERG, D. and DE JONG, K., 2000, The stochastic dimension in a dynamic GIS. In *Compstat 2000, Proceedings in Computational Statistics. 14th Conference of the International Association for Statistical Computing*, 21–25 August, J.G. Bethlehem and P.G.M. van der Heijden (Eds) (Utrecht: Physica-Verlag).

PYTHON 2005, Python programming language. Available online at: http://www.python.org (accessed 15 October 2005).

RABUS, B., EINEDER, M., ROTH, A. and BAMLER, R., 2003, The shuttle radar topography mission—a new class of digital elevation models acquired by spaceborne radar. *ISPRS Journal of Photogrammetry and Remote Sensing*, **57**, pp. 241–262.

STELLA 2005, Available online at: http://www.hps-inc.com/ (accessed 15 October 2005).

SU, Z., 2002, The Surface Energy Balance System (SEBS) for estimation of turbulent heat fluxes. *Hydrology and Earth System Sciences*, **6**, pp. 85–99.

TIMMERMANS, W.J., VAN DER KWAST, J., GIESKE, A.S.M., SU, Z., OLIOSO, A., JIA, L. and ELBERS, J., 2005, Intercomparison of energy flux models using ASTER imagery at the SPARC 2004 site (Barrax, Spain). In *SPARC Final Workshop. ESA Proceedings WPP-250*, Enschede, the Netherlands: ESA Proceedings WPP-250.

VAN DER KWAST, J. and DE JONG S.M., 2004, Modelling evapotranspiration using the surface energy balance system (SEBS) and Landsat TM data (Rabat region, Morocco). In *3rd Workshop of the EARSeL Special Interest Group on Remote Sensing for Developing Countries*, 26–29 September, Cairo.

VAN DEURSEN, W.P.A., 1995, *Geographical Information Systems and Dynamic Models* (Utrecht: Koninklijk Nederlands Aardrijkskundig Genootschap/Faculteit Ruimtelijke Wetenschappen, Universiteit Utrecht).

WAINWRIGHT, J. and MULLIGAN, M., 2004, *Environmental Modelling* (Chichester, UK: Wiley).

WESSELING, C.G., KARSSENBERG, D., VAN DEURSEN, W.P.A. and BURROUGH, P.A., 1996, Integrating dynamic environmental models in GIS: the development of a dynamic modelling language. *Transactions in GIS*, **1**, pp. 40–48.