ISSN 2464-4617 (print)
ISSN 2464-4625 (online)
Computer Science Research Notes - CSRN 3301
http://www.wscg.eu
WSCG 2023 Proceedings

Operational theater generation by a descriptive language

Matis Ghiotto

Aix Marseille Univ, CNRS
LIS, Marseille, France
matis.ghiotto@lis-lab.fr

Brett Desbenoit

Aix Marseille Univ, CNRS
LIS, Marseille, France
brett.desbenoit@univ-amu.fr

Romain Raffin

Université de Bourgogne
LIB EA 7534, Dijon, France
romain.raffin@u-bourgogne.fr

ABSTRACT

3D landscapes generation is an interdisciplinary field that requires expertise in both computer graphics and geographic informations systems (GIS). It is a complex and time-consuming process. In this paper, we present a new approach to simplify 3D environment generation process, by creating a go-between data-model containing a list of available source data and steps to use them. To feed the data-model, we introduce a formal language that describes the process's sequence. We propose an adapted format, designed to be human-readable and machine-readable, allowing for easy creation and modification of the scenery. We demonstrate the utility of our approach by implementing a prototype system to generate 3D landscapes with a use-case fit for multipurpose simulation. Our system takes a description as input and outputs a complete 3D environment, including terrain and feature elements such as buildings created by chosen geometrical process. Experiments show that our approach reduces the time and effort required to generate a 3D environment, making it accessible to a wider range of users without extensive knowledge of GIS. In conclusion, our custom language and implementation provide a simple and effective solution to the complexity of 3D terrain generation, making it a valuable tool for users in the area.

Keywords

Geographics data, operational theater, descriptive approach, multi-modal geometry processing

1 INTRODUCTION

3D landscape generation is an active topic of research in the field of computer graphics and GIS. Operational theater consists in a subset of landscapes used for serious games or military simulations. The goal of operational theater generation is to create realistic representations of natural and urban environments based on real-world data, or GIS data. The integration of GIS data into 3D landscape generation is a challenging task. It requires the efficient processing of large and complex datasets as well as some knowledge of cartography and coordinates system. Environment generation in the industry are made by either procedural or sketching [2] tools. As procedural generation tends to create an artificial (yet realistic) terrain, as opposed to a terrain depicting a real place, it does not fit for operational theater generation. Sketching is more adapted, allowing human intervention to curate the terrain into something close to the existing surface it tries to emulate, but it is a lengthy process. Sketching requires expertise in GIS and use of sketching software. In this paper, we propose a method to reduce time and complexity of operational theater generation by limiting sketching to a formalized description of the terrain to generate. Smelik et al. [11] gave a relevant overview of the different elements used to compose a 3D environment, but also highlighted the lack of methods allowing for the generation of a complete terrain using all the different elements. In recent literature, this segregation between natural terrain and urban terrain is still existing. Eric Galin et al. [3] present a state-of-the-art review for the different methods of natural scene generation, but with no insight on how to integrate them in an urban landscape. In contrast, Hoang Ha et al. [8] only approach road network generation and Tang Ming [12] city generation. Each paper focuses on a specific type of terrain generation and, when they provide a new way to simplify their specific task, they do not provide an easy way to interface their work with others of the same type, in order to generate a complex landscape with natural and urban terrain alike. Our study-case is the generation of an operational theater fit for military simulation. It is a complex task with concrete industrial application. Operational theater includes both urban and natural environment. We address this problem by a flexible way, generating any type of terrain that can be easily enhanced by future field-specifics works. The output at this time is a full 3D scene, merging with geomet-

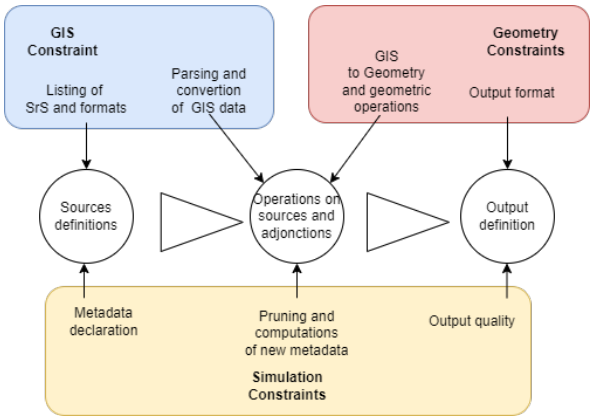


Figure 1: Language principle overview

ric processing of geographic data. The new method we propose rely heavily on the formal description of the scene to generate, so our first approach was to identify the different elements that make an operational theater.

This lead to a three-step process represented in figure 1:

- identify sources, ranges and format
- extract, process and merge data
- adapt outputs

We defined a data model and a specific language to generalize our approach and facilitate user's interactions. This language will be embedded in a file and will be sufficient to create a full operational theater.

1.1 GIS data diversity

The input of our process is mainly GIS data. It involves handling geospatial ranges, coordinates and references, data types and associated metadata. Surface coordinates face approximation due to the curved surface of earth. Geocentric coordinates are inconsistent with altitude, as the sea level is not constant and the earth is not a perfect sphere. This lead to historical disparities between coordinates systems [4]. It has become even more complex with new technologies and the number of digital data formats.

Initiatives have been made to solve this problem. The Open Geospatial Consortium (OGC) is a major actor in the interoperability of geographical data. The consortium has introduced many standards, helping to harmonize the use of GIS data and intercompatibility between them. Webservices such as Web Map Service (WMS) and Web Feature Service (WFS), are OGC standards. They allow users to fetch information (maps and features) from geographic servers. It is an important part of multi-format data process, as it decentralizes data sources. Each Webservice can implement a process for the data type it uses and requires minimal user's expertise. Webservices are a reliable bridge between OGC data formats if the user knows which format they have access to and which format they want to use.

Even with OGC standard, some tasks are traditionally performed in specific formats that are not adapted for a generic operational theater generation. For example, describing a city with CityGML is a documented task [13]. But, if we need to incorporate this city into a larger environment, CityGML is no longer suitable. The transformation of CityGML data into another more suitable format will need processing and expert knowledge. Hopefully, the format is documented and based on norms.

Our first problem will be to use data in different formats that are not trivially intercompatible. To do so, we will need to list the formats and be able to use the right OGC protocol to translate them. This is depicted in figure 1 as GIS constraint in blue.

1.2 Geometry constraints

Rendering geographical data on screen require their conversion into geometrical elements. We will first describe how to construct the operational theater from the geometrical point of view. An operational theater is composed of a base, the ground terrain and surface elements, such as vegetation or buildings. The terrain is a Digital Terrain Model (DTM) [3] with geographic information superposed on it in most cases. This geographic elements, studied in [11], can be grouped in three geometric categories:

- Surface elements, which cover large parts of the DTM and represent a large homogeneous chunk of the environment such as sea, farm, forests...
- Discrete elements which are small meshes and usually represent buildings, individuals trees or particular geological formations.
- Continuous elements which are linear components, such as roads or rivers.

The geometry can cover a wide country-sized area. Even with optimization [5] [15], covering that surface and retaining details is not trivial. Large operational theaters, typically covering over 250,000 hectares (50 km $\times$ 50 km) with 1 meter to 10 meters precision, require tiling and level of detail (LoD) or a patch-based generation method [1]. Tiling refers to the subdivision of the terrain into smaller areas that can be loaded and cleared as needed.

LoD refers to the superposition of meshes of various resolutions to represent the same object, with only one visible at a time, depending on the proximity between objects and the viewpoint origin. The use of both LoD and tiling reduces much of the cost and allows large and more detailed scenery to be made suitable for rendering.

Our second problem is to split the GIS elements into the corresponding geometrical elements, and process them to create tiling and LoD appropriate for rendering. This can be generalized as all optimization operations made on the data, and is displayed in red on figure 1 (Geometry constraint).

1.3 Simulation constraints

Thirdly, there are specific requirements for simulation. Creating a scenery for 3D mobiles deals with different constraints than creating a web application for flood control [10], [6]. These constraints vary from a simulation to another and can be summarized as:

- Geographical extent, referencing the part of our world described by the scenery
- Theater accuracy, whether it is exactly the same as reality or if there is a margin of error on the present elements
- Metadata, such as soil quality, vegetation density, or street names
- Specific information requiring computation, such as an inter-visibility check or ground distance computations

A user may need an operational theater to comply with any number of these previously defined constraints. Modifying any of these constraint parameters calls for recreating the entire operational theater. It is a time-consuming operation without an automated generation process. Therefore, our third problem is to modify parameters or add new data or method to the operational theater without having to rethink the entire creation process. This impacts the generation process at all levels and is displayed in yellow in figure 1 (Simulation constraint).

These problems are difficult to handle. Users must understand simulation constraint as well as geometrical and GIS constraints. Moreover, once an operational theater is generated, it is not possible to modify it with new data or add support to a new simulation constraint, without restarting a new creation from scratch. In the method we proposed, we shift this difficulty into something easier to manage: the created landscape is still static, but the model representing it, written in a human-readable script, is easy to modify.

2 DATA MODEL

In previous sections, we identified difficulties linked to the many constraints of operational theater generation. To address them we present a new approach, based on a descriptive-oriented language, seeking to explicit simulations needs and constraints that can be used to automatically create an operational theater.

2.1 Overview

The identified constraints are linked to three main categories:

- GIS constraints
- Geometry constraints
- Simulation constraints

We consider these constraints as tangled elements rather than independent elements. We unraveled them and exposed that GIS constraints never impact the geometrical output outside construction. With the same logic, outside the specific data format that was already accounted as a GIS constraint, geometry constraints are never used to determine what kind of sources will be used as base material for the operational theater generation. Our approach is to propose a description of these two subsets of constraints (the available data and the expected output) to get parameterized input and output, adapted to computer process. Then, we define a subset of operations allowing to extract and modify the input to obtain the output. As shown in figure 1, this is the global structure of our solution.

Sources used are determined from available GIS data according to a geographical extent and metadata needed by simulations constraints.

Outputs are determined from geometrical constraints, limiting data quantity and format, and by simulation constraints requiring a minimal precision and quality.

2.2 Model specification

Generation methods based on scripting exist in the literature [12] but are traditionally not used in conjunction with wide real-world data. To handle these complex data, we took inspiration from L-System [9] and conceived a theoretical data model that allows data mutation. This abstract model is fed and manipulated by a scripting language. The language we defined is composed by three core-concepts: Data, Sources, and Operations. The main idea of the language is to handle these concepts to create a data model representing an abstract operational theater before processing.

Data represent all the information needed in the 3D scenery representation. They can be assigned as variables to feed operations.

Sources are all the protocols needed to communicate with external data sources. We collect sources to obtain raw data or letting a server do requested modifications on raw data before their acquisition.

Operations encompass all the modifications that can be done to process some data and result to other ones, assigning them to a variable.

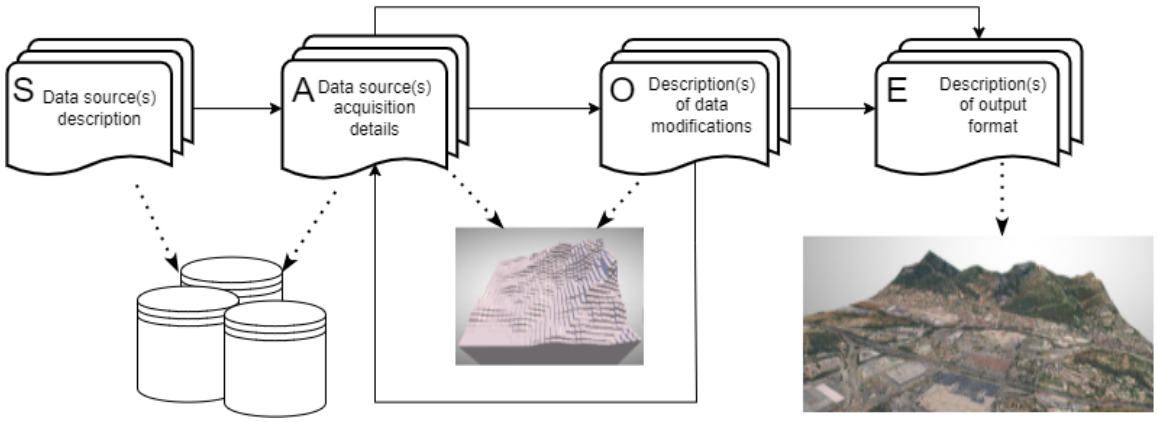


Figure 2: Description language schematic representation

The Data model is the most complex part of our language. This is the part where we perform acquisition and transformation of data sources, so we have usable and inter-operable data. The Data model allows a global and abstract representation of scenery while retracing the origin of all available data, lists all the transformations and applies them. As described before, these data can contain metadata about their actual quality or about their formats, and it's possible to use these as criteria to manipulate them. This model is innovative as it allows a representation of a 3D scenery composed of multiple georeferenced elements and their relative disposition without being a graphical data.

Operations can make use of external software by specifying the executable path and the output location. The commands will be handled as if the executable output is a data from a specified format coming from a local, non-mutable, source. It allows extensibility for the language, making use of state-of-the-art processes without having to re-implement them.

3 METHODS

3.1 Language

We proposed the implementation of the previous concepts in a language including the followings elements:

- One or more Source-declaration blocks (formula 2) that can be either:
  - A local source, indicating path, format and all necessary metadata (formula 3).
  - A geoserver source, indicating the connection address, the protocol (WMS, WFS, WPS) and optional connection parameters (formula 4).
- One or more data Acquisition blocks (formula 5), indicating a predeclared source, an identifier (id) for the newly obtained data and an acquisition parameter if the source allows it.
- Any number of Operation blocks (formula 6) declared with the following information:
  - A declared source data id.
  - A not previously declared new data id.
  - A processing name (already implemented or external command).
  - Arguments needed by the operation.
  - Any number of nested operation blocks.
- One or more Export blocks (formula 9), containing:
  - The format for the output model.
  - Constraints linked to this format (such as number of vertices, texture quality, etc.).
  - The list of data to use for model creation.
  - A fusion heuristic if the format is implemented with more than one fusion processes.

Formally, the language specification is as following:

Let D the description language, S a source block, A an acquisition block, O an operation block, and E an export block, we define:

$$D = S^{+} A \left[A \cup O\right]^{*} E^{+} \tag{1}$$

Figure 2 is a schematic representation of this equation 1.

3.1.1 Sources

Sources blocks noted S are defined as:

$$S = S_1 \cup S_2 \tag{2}$$

$$S_1 = t_1 \, a \, s \, f \, c \, g^{*} \tag{3}$$

$$S_2 = t_2 \, a \, s \, g^{*} \tag{4}$$

With $S1$ and $S2$ respectively a local and distant source. t is a source type (t_1 covers file sources and t_2 covers

geoserver sources), a is an address or a path, f is a file format, c is a CRS, s is a source id. A specific s id must be present in only one S block. g is an unspecified argument, it may be used to convey additionals informations, g is included in the language for the sake of extensibility. It can be any number of g arguments. Block S, which stands for Source, describes the data to obtain and how to obtain them. They are declared in the header of our description file and can show different behaviors depending on the given arguments at the time of data acquisition.

Sources can depict a folder, local or upon a network (t_1), a requestable database, or a geographical data server (t_2). Geographical data servers can alter data before serving them. These alterations are defined by the geoserver standard, and they can be, but are not limited to: georeferencial rebasement, aggregation of multiple data on the same bounding boxes and segmentation of data to obtain only the one inside a defined bounding box.

This differentiation is made via a specific keyword and does not impede future implementations of other data acquisition methods.

S blocks are logically linked to Acquisition block, denoted by A.

3.1.2 Acquisition

$$A = s\,d\,b\,g^* \tag{5}$$

s is a source id and d is a data id and b a geographic bounding box limiting the data to get. A specific source id denoted by s can be used only if it was previously declared in a S block. A specific d argument can't be reused if already used in a A block. A b argument must describe a bounding box included in the one declared in the corresponding s. Block A allows loading the information described in a source into the data model. It makes use of all the information specified in the source and the added arguments g to create a named data d, which will be viable to modify or export later. The same S block can be used by multiple A blocks to create different data by specifying various bounding boxes b or using the arguments to request different server-side data management.

3.1.3 Operations

Operations blocks noted O are defined as follows:

$$O = O_1 \cup O_2 \tag{6}$$

$$O_1 = o\,d_1\,d_2\,g^*\,O^* \tag{7}$$

$$O_2 = p\,d_1\,d_2\,a\,f\,g^+\,O^* \tag{8}$$

Where d_1 represents an already existing data (and thus must be previously declared) that will be used as source for creating a new data d_2. d_2 must not have been previously declared. In O_1, an o operation will be declared to be executed by the interpreter, using an internal process to modify a d_1 data into a d_2 data using only this data and optional arguments depending on specifics o. In O_2, an external process hosted at address p will be executed, its result must be stored in a. Once completed, the file at the address a is read and loaded into d_2, as if it was a local source of format f. O blocks augment the data model with newly made data fitting the user's needs.

3.1.4 Export

Export blocks denoted E are defined as follows:

$$E = f\,d^+\,(c\,g^*)^* \tag{9}$$

c is used as a geometry constraint to create an exportable version of the data model.

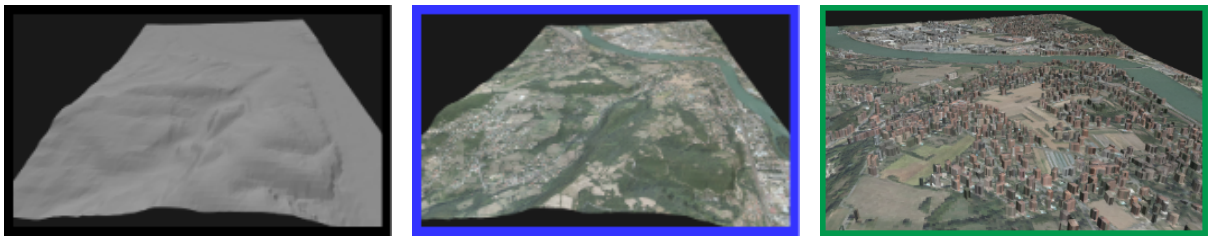
The E blocks, or Export are in charge of producing the 3D scene or other data format specified by the user. This part allows the specification of the non-geographical simulation constraint c. These constraints are deeply linked to a specific format, and thus, do not have a place in the data model. These constraints can be the quality of the 3D meshes created, their sizes, or their formats. It can also be the used metadata, or simulation-only data that must be attached to the scenery elements.

We presented a proof of concept for a new method of landscape generation, incorporating the basic features required to create a landscape. Our approach includes the ability to import GIS data, as well as the ability to use process from external tools, such as GDAL, and maintain relationships with the source data.

3.2 Implementation

The implemented subset of the language contains the following elements:

- Data format
  - Image for texture purpose, PNG and TIFF
  - Raster data, tiff and XYZ
  - Vector data, Shapefile and GML
- Operation
  - HeightMap mesh creation from XYZ data
  - Mapping of terrain mesh and vector features
  - Texturing
  - Land flattening around vector features
- Output formats



(a) texture-less terrain (b) textured terrain (c) textured terrain with buildings

Figure 3: Visual examples

– GLTF as 3D mesh

In our proof of concept, we have selected XYZ, TIFF, GML and Shapefiles as sources data formats. We selected these formats due to their support from GIS standard libraries such as GDAL [14] and widespread availability, notably by local organizations such as the French survey (IGN, from which we get most of the data). For our test outputs, we have chosen the GLTF format, backed by Khronos, as it provides an optimized geometric format for rendering and is straightforward to write. This choice of formats and processes reflects a meaningful range of capabilities within the GIS field, as it covers all steps involved in the creation of a complete terrain. The process starts with data collection and includes the aggregation of vector and raster data, leading to the creation of a simulation-ready 3D operational theater. We have also added some specialized operations to demonstrate the versatility of our model, its ability to integrate various data sources, and its alignment with the needs of simulations.

The executable running the language is implemented with C++. The computer executing it has a AMD Ryzen 5 3400G processor with 4 cores and a 16 GB RAM.

This implementation is a showcase, and the operations can be further optimized with state-of-the-art algorithms. The test case's goal is to determine the difficulty to produce different operational theaters. We evaluate the time used to create a specific theater, and also assess the visual coherence of the results, the skill level required to conduct the test case and the ease of modifying it to meet other requirements.

4 RESULTS

To represent elements constituting our scenery, we chose an indicative and functional-like language, based on the JSON standard. JSON has the advantage to be easily interpreted by computers without being too harsh to read and write by humans.

4.1 Incremental complexity

The figures 6i, 6j and 6k show one of those files. We can see the all the elements of our language represented here.

Each letter under bracket (and text color) shows a different step of the test. The same bracket letter (and corresponding color) is used for figures 3 and 4. The construction of this file is incremental. The text under brackets "a" (black text) alone is enough to create a simple texture-less terrain, and adding step by step, "b" (green) and "c" (blue) will refine it into a textured terrain with buildings. "d" (red) shows an optional terrain modification to demonstrate the possibility of the language.

"Sources" in figure 6i that can contain one or more S Sources as defined previously, here contains a $t1$ "localAccess" named s SourceHeightmap at address a "../GrandLyonData.XYZ" with a data format f "XYZ" and a geographic system c "EPSG:3946". A second local source with similar parameters named "SourceBuildings" and a distant source "SourceOrthophoto" are also present, but we will see them in detail later with an advanced example.

"Populate" in figure 6j is the first part of the data model and can contain one or more A blocks. Here it contains the acquisitions of data described by the previous s, into a new data id d "LoadedHeightMap" (and respective "LoadedBuildings" and "RequestedOrtho"), but limiting the acquisition to the range of the specified bounding box b, expressed in the previously declared geographic system for or in a new one in the case of Webservice sources as in "b" blocks.

"Build", first half of figure 6k, can contain any number of O blocks. Here it contains three operations o. The first one, under bracket "a", named "HeightMapToMesh", builds a mesh from the previous data d_1 "LoadedHeightMap" to the new data d_2 "TerrainMesh". The second, under bracket "b", will similarly create a new data d_2 "TerrainMeshWithBuildings" using the previously created d_1 "TerrainMesh" and the data given by argument g "LoadedBuilding". The third, under bracket "d", will similarly apply a new geometrical process to the generated data.

"Outputs", second half of figure 6k, must contain at least one way to export E for the data. It contains here an f GLTF output from the d "TerrainMesh" (or "TerrainMeshWithBuild" or "TerrainMeshFlattened" and "RequestedOrtho" if we go farther into the exam-

ples) data using arguments *g*, such as the model size, bounding box and the output filename.

Sections "a" of this descriptor file show a short generation script for making an untextured 3D model. This illustrates the language simplicity. The only skill needed to use it is to be able to express the bounding box coordinate of model integration step in the coordinates reference system (CRS) declared in source description step. It is too simple for any real-world use, but it is a base easy to increment.

Sections "c" show the language capacity to communicate with a geoserver and aggregate sources of different origins. Data present on the geoserver referenced in figure 6i are initially under CRS EPSG:3857, but we can request them in another CRS providing the right arguments in figure 6j. If the data in the geoserver and the one used to create the terrain model are correct, they will match and create a textured terrain when declared together at the GLTF generation in figure 6k. This is as simple as the previous step and will require no more skills. The biggest difficulty is to verify beforehand if the data are not faulty. There is no easy automated way to do so embedded in the language, and it falls to the user charge to verify their data sources.

Sections "b" show the language capacity to re-use previously created element and aggregate complex vector sources. We declare a vector file containing buildings in 6i and 6j as done in the first steps. It should be mentioned that the data are this time from the same provider as the ones used for the Heightmap in the black part, so there was no need to verify if they were coherent with the already present ones. The fusion of the data is happening in the build step in 6k. The process "addBuildingsToMesh" takes a Mesh as source data and a vectorial feature list as argument and creates a new data that contains both. The implementation checks if the data used are of an expected format and issues an error if they are not. With this step we get a minimal but simulation ready operational theater we can further supplement by adding roads and rivers delimitations, vegetation or any required element by following the same steps. We see the progression of these three use-cases in figure 3.

Section "d" shows the model capacity to keep track of data hierarchy. The process declared is a simple geometrical operation that flattens the ground under given features. It may seem of no more interest for our example than the "b" process, as it takes the same kind of arguments and produces the same kind of output. But we only use as input for these process the data created with the precedent processes. This highlights that data are keeping a symbolic link to the data that was used for their creation, and that we can use this hierarchy to easily reuse data. The process can use the feature list of id "LoadedBuildings" referenced by "TerrainWithBuildings" to flatten the ground without damaging the buildings, as shown in the wireframe of figure 4.

4.2 External command

The main limitation of this language is the number of implemented processes that cannot possibly match the wideness of GIS. That why the language can also use external processes such as GDAL command line to process data. This is a less flexible way to use data, and it requires knowledge about the external program. The example shown in figure 5 was obtained by using the build step in listing 1.

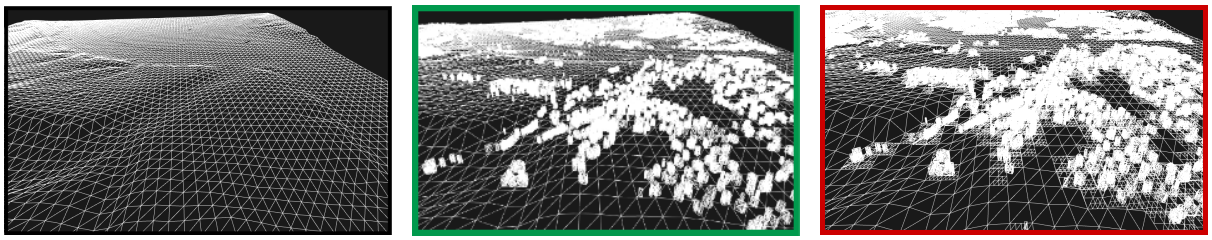
4.3 Gathering results and analysis

These examples demonstrate how to add a complex element in the operational theater, as well as the communication and fetching data from geoserver. They also demonstrate that a small amount of knowledge of GIS or information processing is required to generate a simulation ready simple operational theater. Some may still be needed to understand errors inherent to the sources, such as a delta between different sources created by the conversion of CRS or by inaccurate sources, which are at least not format-specific knowledge. Once the sources are selected successfully, it is particularly easy to manipulate the script to display more or less elements or another geographic area.

Table 1 gathers analyses of the examples. Three additional tests to the incremental construction have been made: "Contouring with GDAL" is for the contouring example (see section 4.2) that displays a smaller area but with the call to an external process; The "Full large zone" is expensive to generate, but it is coherent as it covers a zone 625 times larger than the first example with the same level of detail. The high triangles count makes it difficult to display or load in a visual engine; the "Full zone split into 16 tiles" is, as the name suggest, the same zone but generated with 16 square tiles instead of only one in the previous example. It is obtained by adding multiple delimitation sections in Models and Outputs (respectively, figure 6j and figure 6k). This method enables reduced generation time and easier load/display.

Listing 1: External process example

```
"Build" : [
    {
        "pAdress" :
        "./gdal_contour -p -i 10.0",
        "origData_id" : "HeightMap",
        "pResult" : "./contour.png",
        "format" : "png"
        "newData_id" : "Contouring"
    }, ...
```



(a) featureless (b) with building (c) with geometrical process

Figure 4: Wireframe examples

	Acquisition Time (s)	Triangles count (k)	Generation time	Surface
a (Black) - Base case	4	4,016	1 min 21 s	2 km × 2 km
c (Blue) - Texturing	12	4,016	1 min 31 s	2 km × 2 km
b (Green) - Adding building	10	8,126	2 min 15 s	2 km × 2 km
d (Red) - Geometric operation	10	10,252	4 min 10s	2 km × 2 km
Other - Contouring with GDAL	4	2.048	32 s	500 m × 500 m
Other - Full large zone	24	6,089,800	3 h 15 min	50 km × 50 km
Other - Full zone split into 16 tiles	24	380,600 × 16	1 h 07 min	50 km × 50 km

Table 1: Summary table

5 PERSPECTIVES

As we present this work, we do not implement all the operations that a user may need. It is a first proof of concept and the next version is planned to include a plugin system to allow GIS or simulation communities to add any sources needed from data or operations, and to upgrade it as new cutting edge algorithms, or with new data formats. Our objective is to focus in particular on the integration of 3D tiles formats, a natural extension of the GLTF we use, allowing georeferenced tiling. This is essential, as tiling shows very good performances compared to non-tiled methods for large surfaces. We also consider adding conditional branching structures to the language. Doing so will further reduce the knowledge needed for the user, by letting the implementation taking decisions based on formula or metadata. The conditional system, in addition to data-quality measurements, may alleviate to some extent the difficulty of sources selection, by choosing automatically the best of two sources under an objective criterion, such as approximation error on the CRS or comparison with another data marked as "trustworthy". Another perspective is to add new interfaces. We will add dynamic information provider to our data model to communicate the data from the model directly to a simulation client. This is included in a SaaS (Simulation as a Service) and WebGIS 2.0 [7] approach of the simulation's problem that breaks down complex simulation elements into more understandable services. Doing so improves accessibility for nonspecialized users who will only be confronted to a standard interface and is not required to understand all the underlying complexities.

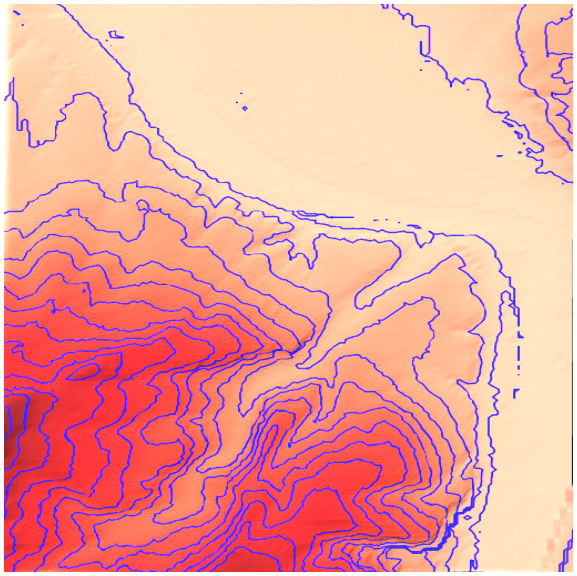


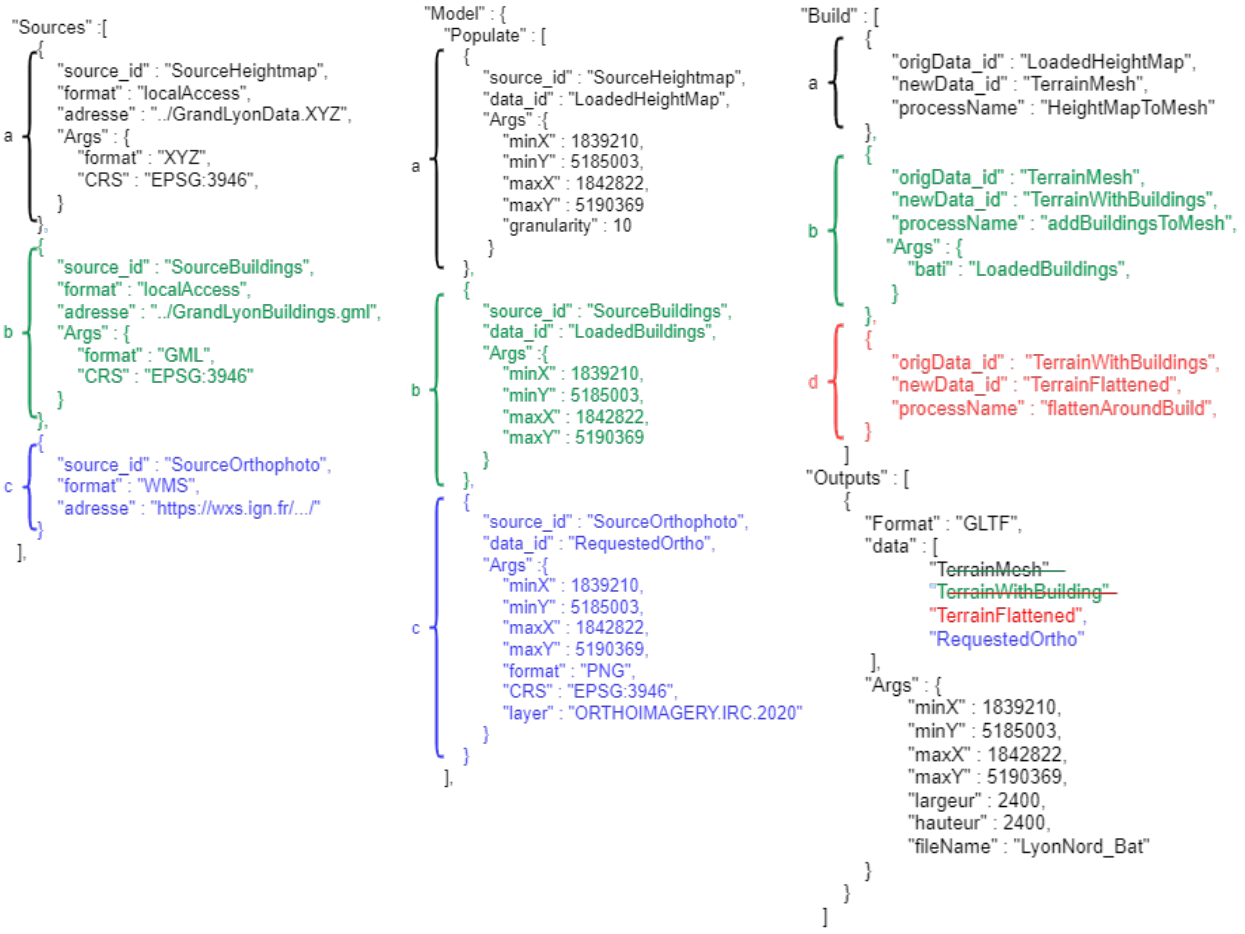
Figure 5: Contouring line by call to an external process

Acknowledgements

The authors acknowledge support by SopraSteria and French ANRT "Association Nationale de Recherche et Technologie" under CIFRE n° 2020/0364. The authors also thank the GrandLyon and IGN open data for their resources.

6 REFERENCES

[1] Leandro Cruz, Luiz Velho, Eric Galin, Adrien Peytavie, and Eric Guérin. Patch-based Terrain Synthesis. In *International Conference on Computer Graphics Theory and Applications*, Proceedings of the 10th International Conference



(i) Sources description (j) Model Integration (k) Build and Output

Figure 6: Descriptor file

on Computer Graphics Theory and Applications, page 6, Berlin, France, March 2015.

[2] James Gain, Patrick Marais, and Wolfgang Straßer. Terrain sketching. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, I3D '09, page 31–38, New York, NY, USA, 2009. Association for Computing Machinery.

[3] Eric Galin, Eric Guérin, Adrien Peytavie, Guillaume Cordonnier, Marie-Paule Cani, Bedrich Benes, and James Gain. A review of digital terrain modeling. *Computer Graphics Forum*, 38(2):553–577, 2019.

[4] Ian N. Gregory and Richard G. Healey. Historical GIS: structuring, mapping and analysing geographies of the past. *Progress in Human Geography*, pages 638–653, 2007.

[5] Pedro Morillo, Juan Manuel Orduna, Miguel Fernandez, and Jose Duato. Improving the performance of distributed virtual environment systems. *IEEE Transactions on Parallel and Distributed Systems 16 (7), 637-649*, 2005.

[6] Rostivlav Nètek and Marek Balun. Webgis solution for crisis management support - case study of olomouc municipality. In *ICCSA 2014, Part II, pp394-403*, 2014.

[7] Rostivlav Nètek, Vit Vozenilek, and Alena Vondrakova. Webgis 2.0 as approach for flexible web-based map application. In *ICCSA 2018, pp1-5*, 2018.

[8] Hoang Ha Nguyen, Brett Desbenoit, and Marc Daniel. Realistic urban road network modelling from GIS data. In *UDMV*, pages 9–15, 2016.

[9] Yoav IH Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308, 2001.

[10] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[11] Ruben Smelik, Klaas Jan De Kraker, Tim Tutenel, Rafael Bidarra, and Saskia A Groenewegen. A

survey of procedural methods for terrain modelling. In *Proceedings of the CASA workshop on 3D advanced media in gaming and simulation (3AMIGAS)*, pages 25–34. sn, 2009.

[12] Ming Tang. City generator: GIS driven genetic evolution in urban simulation. In *SIGGRAPH Posters*, 2009.

[13] Gwenola Thomas and Stéphane Donikian. Modelling virtual cities dedicated to behavioural animation. *Computer Graphics Forum*, 19(3):71–80, 2000.

[14] Frank Warmerdam, Even Rouault, et al. Gdal documentation: Raster drivers. `https://gdal.org/drivers/raster/index.html`, `https://gdal.org/drivers/vector/index.html`, 2022. Accessed: 2022-09-13.

[15] Ye Zhi, Yong Gao, Lun Wu, Liang Liu, and Heng Cai. An improved algorithm for vector data rendering in virtual terrain visualization. In *2013 21st International Conference on Geoinformatics*, pages 1–4. IEEE, 2013.