

# PROCEDURAL MODELLING FOR RECONSTRUCTION OF HISTORIC MONUMENTS

M. Koehl\*, F. Roussel

TRIO / Photogrammetry and Geomatics Group, ICube Laboratory UMR 7357, INSA de Strasbourg, 24 Boulevard de la Victoire,  
F-67084 Strasbourg, France {mathieu.koehl, felix.roussel}@insa-strasbourg.fr

## Topic C-3: Modelling methods for architecture and archaeology

**KEY WORDS:** 3D modelling; Procedural modelling; Parametric model; Construction rules; Cultural heritage.

### ABSTRACT:

The reconstruction of historic or archaeological monuments bases on the architectural knowledge of the architects and the archaeologists. For the 3D modelling, we can use several technologies as the meshing of point clouds, reconstruction by geometrical primitives and more often the completely manual reconstruction based on a geometry measured on the field. The procedural methods of modelling also allow to build, even to reconstruct historic buildings. They are very effective when several primitives are repeated in regular structures. In this paper, we tested the efficiency of a procedural modelling within the framework of the modelling of the church of Turckheim, Haut-Rhin - France. This church has been built around an older chapel of the XII<sup>th</sup> century of which there exists no more than a bell tower. The procedural modelling allowed to reconstruct the church in the current style borrowed from that of the XII<sup>th</sup> century. The architectural elements built on the basis of rules were then able to be resumed to propose hypotheses of reconstruction of the anterior chapel. Even if the procedural modelling is not the most adapted to this kind of reconstruction, nevertheless it allowed to offer methods of original modelling in an open environment (Esri's *CityEngine 2014.0*) and, what is here the most important, interoperable with other 3D products.

## 1. INTRODUCTION

### 1.1 Project context

Turckheim is a small village situated in the department of the Haut-Rhin in Alsace, France. (Figure 1).

At the heart of the Alsatian vineyard, this city holds numerous historic monuments of which its church. To understand the aim of the project described in this paper, it is important to be interested in the history of this church. Indeed there are numerous historic events which explain its renovation and even its partially reconstruction. The first mention of a Christian sanctuary to Turckheim is dated from 898. This one has been replaced in 12th century by a Romanic building (chapel) of which remains today only the bell tower. Destroyed by a lightning in 1661, the bell tower was then reconstructed and enhanced. In 1736, it was again repaired further to the damages undergone during the passage of a tornado. In 1791, the city endows the church with an organ. Last milestone, the church is the prey of flames in 1978. The organ as all the heart of the church will then has been restored again (1983). From this heavy and rich past it remains unfortunately and as it is often the case only very few documents and available information except for some facts told in diverse papers. But as regards to the geometry, the positioning, the physical characteristics of the initial work, there rest only few elements. Some hypotheses of positions and ground trace were well given by various historians or archaeologists. Thus the project of acquisition and 3D modelling has to allow to supply the current state of the building to be able to be used as base to new investigations and to the generation of new hypotheses about the locations and the historic forms of this church and its original chapel.

To allow the modelling, topographic survey and the inside and outside scans via terrestrial laser-scanner (TLS) has been realized and used.

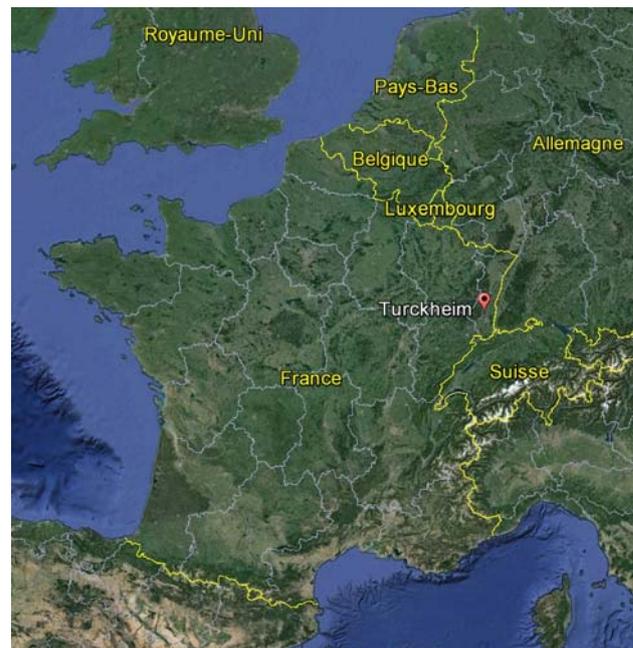


Figure 1: Location of the project  
Turckheim – Alsace – France (GoogleEarth imagery)

A first method described in (Koehl, 2014) was performed in a CAD environment (*Bentley Microstation*) by using the possibilities of partial display of the point clouds and by using especially the familiar tools of CAD engine. This first method required long manual work, but allows a detailed approach by using the principles and the methods known for modelling in a CAD environment.

This project had to define if the use of a software that proposes procedural modelling, in particular the ESRI's *CityEngine 2014.0* package would gain efficiency in this context of modelling of existing object, then to see if the same environment could be used to realize easily simulations within the framework of archaeological restorations applied to the today disappeared chapel.

This paper will begin with the description of the object of the study, then with that of the environment of modelling, to show then the different phases of elaboration and process of modelling.

## 1.2 Modelling environment

*CityEngine* is a three-dimensional modelling software application developed now by Esri. It is specialized in the generation of 3D urban environments with the procedural modelling approach. *City Engine* enables the efficient creation of detailed large-scale 3D city models with merely a few clicks of the mouse instead of the time-exhaustive and work intensive method of object creation and manual placement. The acquisition of *CityEngine* is aiming to push the innovations in 3D GIS and geodesign technology (Sostaric, 2012). In fact, the realism of computer graphics images requires the creation of objects (or scenes) of increasing complexity, which leads to very huge costs. Procedural modelling can help to automate the creation process, to simplify the modification process or to generate multiple variations of an object instance.

*CityEngine* is experienced for mass modelling (Mueller *et al.*, 2012) with more or less random generations. But the examples and applications in the field of modelling of very detailed scenes with modelling patterned after the reality is still rare and limited.

That is why, it could be interesting to use this software application in order to represent places. Experts and laymen study historical structures closely. Digital models of these elements of our cultural heritage are valuable tools for analysis, reconstruction, and virtual display. While the focus of cultural heritage digitization is still on 3D modelling of important major monuments, there are often additional demand for modelling larger settlements. Such settlements might be interesting or might only form the context for a monument.

While archaeologists have detailed architectural knowledge of the monuments and the settlements they study, they have little formal training in CAD or computer graphics modelling packages. Procedural modelling tools such as the *CityEngine* can bridge this gap, providing a user-friendly, high level interface and filling in detail where the archaeological record is incomplete (Piccoli, 2013).

With the objective to reduce the time and facilitate the generation of 3D plans, we created libraries of rules files and textures applied to Turckheim Church.

## 2. MATERIAL AND METHODS

### 2.1 The description of the church

We separate the church into two main areas. The nave and the steeple. The nave is split into four other parts. The choir, the nave, the collateral and the transept (Figure 2). The steeple is split into two parts. The steeple and the porch (Figure 3).

Some assumptions of positions and right-of-way were well issued by different historians or archaeologists. The acquisition and 3D modeling project must therefore provide the current state of the edifice to serve as the basis of new investigations and for the generation of new hypotheses on the locations and

historical shapes of this church and its original chapel (Figure 4).

We initially had at our disposal a lot of data. We had a point cloud from outside and inside of the church obtained from a terrestrial laser scanning survey. We had a photogrammetric survey of the interior and exterior of the church.

We finally also used photographs to create realistic textures for the model.



Figure 2: From left to right: the back of the nave, the steeple, the front of the nave



Figure 3: Red: nave. Green: collateral. Blue: transept. Yellow: choir. Purple: steeple. White: porch.

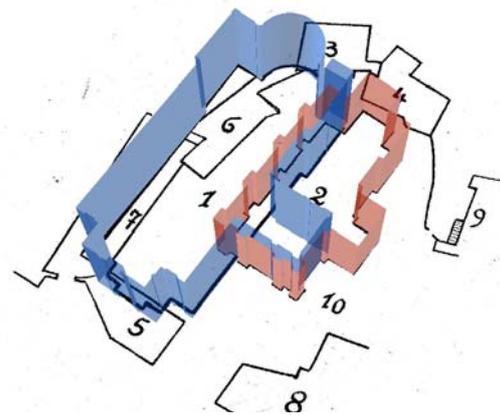


Figure 4: Hypothetic situation of the Sainte-Anne's chapel (2) versus actual church (1)

### 2.2 *CityEngine* as modelling environment

(Mueller, 2007) defined a number of ground breaking techniques for the procedural modelling of 3D architectural contents which make up the foundation of *CityEngine* today. *CityEngine* has been presented for the first time outside of the

research community in 2009 as a commercial product (Procedural inc.). Then it was acquired in 2011 by Esri, one of the most powerful and well-known GIS Company. The main usefulness and highlight of *CityEngine* is to supply 3D contents from GIS data and also generate scenes and 3D models. Once the database loaded in *CityEngine* such as DTM, roads axes, footprints of buildings, vegetation, etc., the application of procedural rules can help to build the 3D entities. To better understand the process of generating 3D models, construction rules of 3D entities have been analysed and assigned in order to model a historic castle, the Engelbourg, Thann, France (Koehl *et al.*, 2014). (Piccoli, 2013) presented also a very complete overview of the potential use of *CityEngine* in archaeological projects.

*CityEngine* is a 'Procedural Modelling Application'. This means that *CityEngine* uses rules to generate the 3D Model according to the process described in Figure 5.

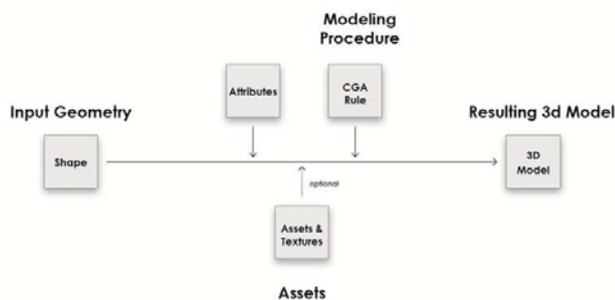


Figure 5: General process of model generation

Beginning from the base form, we have to describe construction rules translated in CGA language allowing to modulate the forms (extrusion, division, repetition, etc.). To write a rule, the user can use a rule editor that works in a direct visual mode or means code writing.

The textual way allows to edit live code in particular to comment it so it will be able to share it and reuse it in other applications, while the visual way will be more intuitive to analyze a complex flow of the same type of rules.

Rules are then applied to reference entities (faces) what allows the complete reconstruction of the 3D object. This application of rules can be made entity by entity or on several entities at the same time if they contain form similarities. In this case the dimensions can afterwards be adapted through individual parameters.

### 2.3 Edition of reconstruction rules applied to facades

There exist also other software applications allowing to edit rules, particularly in the case of facades for which we can use a "wizard" (Mueller *et al.*, 2007). The "facade Wizard" is an adapted tool allowing the user to create complex CGA rule models for facades. One of the advantages is that no CGA codes is needed to be written by the user, it is directly generated by *CityEngine* as background task. So complex architectural structures can be generated in an effective and fast way. This "wizard" uses as first entry an image of the facade. The image can directly be ortho-rectified by using a simple algorithm based on 4 coplanar points. As a result, the ortho-image in .PNG format can then be segmented in a following stage.

At this stage, the facade is going to be divided horizontally into floors or levels, then every level is going to be vertically divided into columns, doors or windows, etc.

Every individual rule or the aggregation of all the rules can then be attributed to every face of the facade.

### 2.4 The CGA language

This part introduces the basics of the CGA shape grammar of *CityEngine*. We will analyze a finished rule file which contains all steps to create a basic building.

#### Building attributes

Building attributes are normally defined on the very beginning of the rule file (although they can be put anywhere in the rule file). These attributes are used through the whole rule set and appear in the CGA "attribute mapping area" of the "Inspector", where their values can be set and modified outside the CGA grammar editor as well.

```

attr groundfloor_height =4
attr floor_height = 3.5
attr tile_width =3
attr height = 11
attr wallColor = "#fefefe"
    
```

#### Window asset

The window asset used for the creation of the simple building is defined here. The current asset is loaded from the project's assets folder, found in the navigator.

```

// geometries
window_asset = "facades/window.obj"
    
```

#### Lot rule

The creation of the building starts now. Our first rule is called "Lot". Remember the assigned start rule in the "Inspector". The mass model is created with the extrude operation:

```

Lot -->
Extrude (heights) Building
    
```

#### Building rule

Usually in the next step, such a mass model can be divided into its facades by applying the component split.

```

Building -->
Comp(f) (front: Frontfacade | side: Sidefacade | top: Roof)
    
```

This rule splits the shape named Building, the mass model, into its faces by applying a component split. This results in a front shape (usually the main front facade with entrance), several side shapes as Facades, and a Roof shape.

In a first step, the facade can be decomposed into Floors. Afterwards, the floors are further broken down into elements we call Tiles (floor subdivisions). A tile consist usually of wall and window elements. This subdivision scheme can be implemented in the CGA shape grammar as follows (Figure 6):

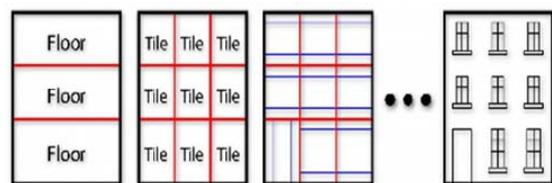


Figure 6: Split organization

### Frontfacade rule

```
Frontfacade -->
Split(y) { groundfloor height : Groundfloor
          | {~floor_height      : Floor}* }
```

The FrontFacade rule splits the front face into a ground floor shape (of predefined 4 m. height) and repeating (using the repeat operator ' \* ') upper floor shapes (of approximate predefined 3.5 m. height). The tilde operator '~' guarantees that no matter how high the building actually is, always a full number of upper floors is created. Especially for front facades, the appearance of the ground floor is often different from the other floors. It differs not only due to the fact that it has an entrance, but often also due to different floor heights, different window appearances, other colors, and so on.

### Sidefacade rule

The Sidefacade rule splits the side facades into floor shapes. Therefore, the subdivision split is performed in the same way to assure that the floor heights are in synch with the front facade.

```
Sidefacade -->
Split(y) { groundfloor height : Floor
          | {~floor_height      : Floor}* }
```

### Floor rule

The Floor rule is a typical example of a subdivision of a floor into tiles (of approximate predefined width 3 m.). To make the floor design slightly more interesting, we also split away a wall element of width 1 m. on each side.

```
Floor -->
Split(x){1           : Wall
        | {~tile width : Tile }*
        | 1           : Wall }
```

### Groundfloor rule

The Groundfloor rule refines the ground floor shape with a similar subdivision split, with the difference that an entrance is placed on the right. The following Figure 7 depicts the extruded mass model on the left side and the described decomposition into floors and tiles on the right.

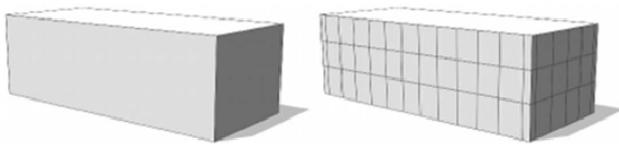


Figure 7: Extruded mass model and decomposition into floors and tiles

### Tile rule

After the initial facade structure has been defined, the tiles can be modeled:

```
Tile -->
Split(x){ ~1: Wall
          | 2: split(y){ 1:Wall | 1.5:Window | ~1:Wall }
          | ~1: Wall }
```

The Tile rule defines the appearance of the tile by splitting along x- and y-axis (with a nested split). We can notice that in this design the wall elements are floating (with tilde) and that the window has a fixed size: 2 m. in width and 1.5 m. in height.

### EntranceTile rule

```
EntranceTile -->
Split(x) { ~1: SolidWall
          | 2: split(y) { 2.5: Door | ~2: SolidWall }
          | ~1: SolidWall }
```

The EntranceTile rule defines the entrance shape in a similar way as the tile shape (but obviously with no wall on the bottom).

### Window, Door and Wall rules

Finally, the last rules replace the geometry of the Window, Door and Wall shapes with the corresponding assets, position them and set the texture:

```
Window -->
s('1, '1, 0.4)
t(0,0,-0.25)
i(window asset)

Door -->
s('1, '1, 0.1)
t(0, 0, -0.5)
i("builtin:cube")

Wall -->
Color(wallColor)

SolidWall ->
color(wallColor)
s('1, '1, 0.4)
t(0, 0, -0.4)
i("builtin:cube:notex")
```

Via the operation  $t(x,y,z)$ , the current shape is translated - 0.25 m. in the z-direction. By this way the windows and textures are set back 0.25 m. into the facade. Afterwards the insert operation  $i(objectname)$  inserts an asset into the current scope. If the dimensions are not set like in the Window or Door rules, the sizes are adapted automatically - otherwise the given dimensions are used. Via the operation  $s(x,y,z)$  the size of the scope can be set in the Wall rule. The width and height of the scope are not affected since relative coordinates are used: the x and y dimensions of the current scope are scaled by one ('1) resulting in no change. The z dimension is set to - 0.4 resulting in a wall with a thickness of 0.4 m. (pointing inwards).

When we put all the above rules together, we get the final un-textured simple building (Figure 8):



Figure 8: Final un-textured model

### Texture Declaration

Like the assets, we define the textures we will use at the beginning of the rule file. The textures are loaded from the assets folder. We have to add the following new lines to our rule file below the `window_asset` declaration.

```
// textures
frontdoor_tex = ".../texture1.tif"
wall_tex = ".../texture2.tif"
dirt_tex = ".../Texture3.tif"
roof_tex = ".../Texture4.tif"
```

Now, we add the two first lines to the Frontfacade and the Sidefacade rules:

```
Frontfacade -->
SetupProjection(0, scope.xy, 1.5, 1, 1)
SetupProjection(2, scope.xy, scope.sx, scope.sy)
Split(y) { groundfloor height : Groundfloor
           | {~floor_height      : Floor}* }
```

```
Sidefacade -->
SetupProjection(0, scope.xy, 1.5, 1, 1)
SetupProjection(2, scope.xy, scope.sx, scope.sy)
Split(y) { groundfloor height : Floor
           | {~floor_height      : Floor}* }
```

The `setupProjection()` command prepares the UV coordinate projections on the facades for color (channel 0) and dirtmap (channel 2), projected onto the `scope.xy` plane, therefore `scope.xy`, is set as second parameter.

The brick texture (channel 0) will be repeated every 1.5 m in X and every 1 m in Y axis, whereas the dirtmap (channel 2) will span over the whole facade and therefore uses `scope.sx` and `scope.sy` as size parameters.

Again, we have to add the lines to the subsequent rules:

```
Door -->
s('1, '1, 0.1)
t(0, 0, -0.5)
texture(frontdoor_tex)
i("builtin:cube")
```

```
Wall -->
Color(wallColor)
Texture(wall_tex)
Set(material.dirtmap, dirt_tex)
projectUV(0 projectUV(2))
```

```
SolidWall -->
color(wallColor)
s('1, '1, 0.4)
t(0, 0, -0.4)
texture(wall_tex)
set(material.dirtmap, dirt_tex)
i(builtin:cube:notex")
projectUV(0) projectUV(2)
```

Wall and SolidWall use the UV's prepared in the Facade rules. Besides choosing the textures for color and dirt channel, we also need to project the UV's on those two channels. Below is the final building model (Figure 9).



Figure 9: Final textured building

## 2.5 CGA rules applied for church reconstruction

For the creation of the 3D model of the church, we use the same process of creation decomposed in the following phases:

- Creation of the project
- Creation of the scene
- Import of the data

We import in this stage, the 2D vector data of the footprint of the church.

- Creation of the rules file
- Writing the rules file

The first part includes the modelling of the main building:

- Attributes: this model rests on 5 different attributes corresponding to the main geometrical dimensions.
- Assets: an asset is an essential entity for the creation of a detailed model. An asset is an entity in the .OBJ format created here with another modeler (like *Trimble Sketchup*).

Our project contains 4 main assets:

- The Lot rule: the actual creation of the building begins at this stage. The first rule is called the Lot. The mass model is created with an extrusion operation. A second rule allows to create the internal walls.
- Building rules: during this stage, the model of mass is divided into facades (FrontFacade, BackFacade, SideFacadeL, SideFacadeR, TopFacade) just like the inside (FrontFacadeInt, BackFacadeInt, SideFacadeIntL, SideFacadeIntR). These facades are then cut in faces.
- FrontFacade rule: this rule cuts the front face in several variable and adjustable levels by a parameter.
- GroundFloorFront: the ground floor is divided horizontally into an entrance surrounded with two wall of FrontFacade.
- WallFrontFacade and EntranceFrontFacade rules: here we divide WallFrontFacade into SoilStrip, WallClassical, HigherStrip, etc.

All this decomposition were made by analyzing the architectural structure of FrontFacade (and farther also of all other facades).

- Assets insertion: we insert then assets after new subdivisions and the preparation of the surfaces.

The main front facade will be used as illustration of this asset insertion and as its resizing fitted to the geometry of the building (Figures 10).

As we can see, the detailed elements cannot be directly modelled without difficulties with *CityEngine*.

Then we use then assets created by means of other modelers: we summarize below the various methods of asset generation.

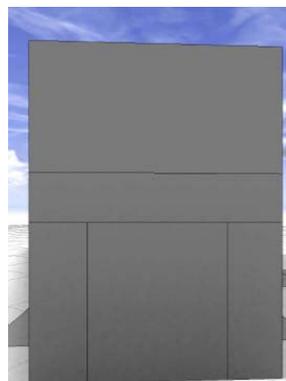


Figure 10-a: Division of the ground floor

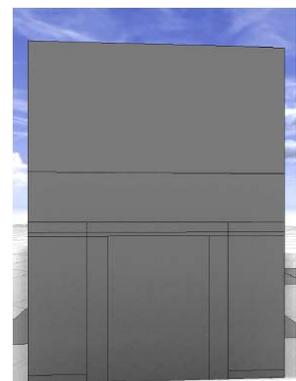


Figure 10-b: Complementary subdivision of the ground floor

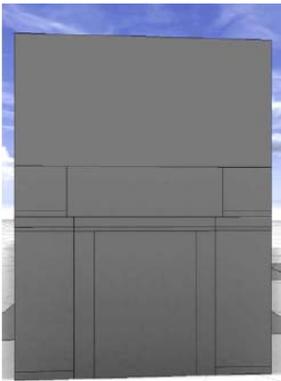


Figure 10-c: Subdivision of first level

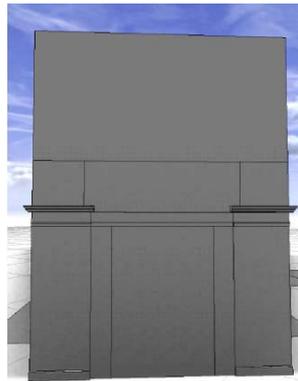


Figure 10-d: Asset insertion

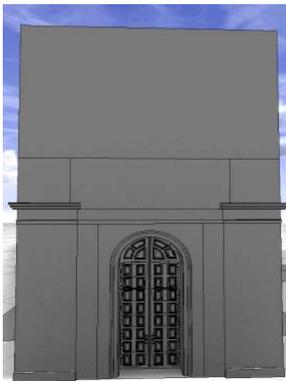


Figure 10-e: Door asset insertion

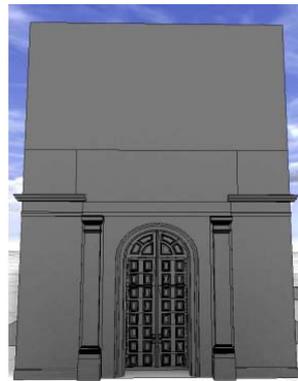


Figure 10-f: Column asset insertion

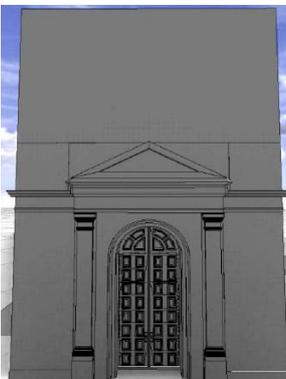


Figure 10-g: Asset insertion for door ornament

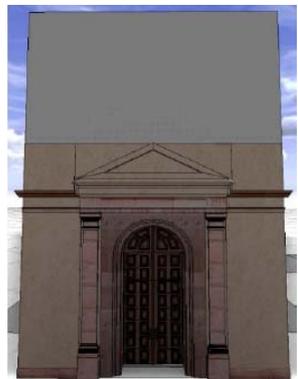


Figure 10-h: Texture

### 3. MODELLING PROCESS

#### 3.1 Use of point cloud measured by TLS

From a point cloud, we can generate meshes by using the mesh software associated to scanners. However, the import of these meshes is not adapted to *CityEngine* who represents all the edges in a visible way. Thus an excessive large number of meshes does not allow an interesting description.

#### 3.2 Creation of objects by means of *CityEngine* modeler

The manual creation of assets is possible in *CityEngine*, but tools and available functions are relatively limited. The only accessible geometries are polylines that will be difficult to size exactly.

#### 3.3 Creation of the object by means of the *Sketchup* modeler

*Sketchup* is an accessible and well spread modeler, offering multiple tools and well adapted to fine modelling. Having drawn the object on *Sketchup*, it's possible to export it in *Wavefront* .OBJ and *Collada* .DAE formats, which can be directly imported in *CityEngine*.

Another advantage of *Sketchup* is the importance of users' community which shares numerous 3D objects in an Internet accessible library called 3D Warehouse. Some simple modifications often allow to use already existing objects and to adapt them to our needs.

The Figure 11 shows an example of window asset created under *Sketchup* and optimized under *Blender*.

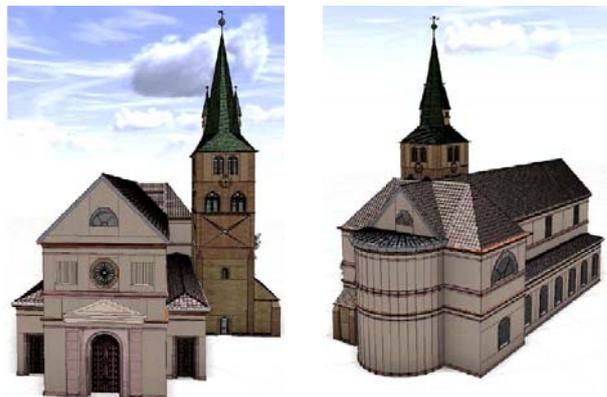


Figure 11: Visualization in *CityEngine* after normals recalculation and the triangulation optimization in *Blender*

## 4. RESULTS

#### 4.1 The model of the church

The Figures 12 show the results obtained for the outside of the church. The bell tower is particularly interesting because of its geometry which is not simply extrudable from a face. The back parts in form of circular shape must be decomposed into polygons with large number of sides because *CityEngine* does not manage directly circular objects.





Figures 12: Modelling results



Figure 14: Modelling of the church and the addition of the chapel of the XII<sup>th</sup> century.

#### 4.2 Exterior of the church and hypotheses for the disappeared chapel

The model of the church includes all the rules of construction and assets of the associated architectural details. The existence of a chapel built in the XII<sup>th</sup> century is proved true, but only hypotheses allow to locate it even today. Archaeologists' works allowed to supply such proposals. We find in the Figure 13 a proposal of the original footprint.

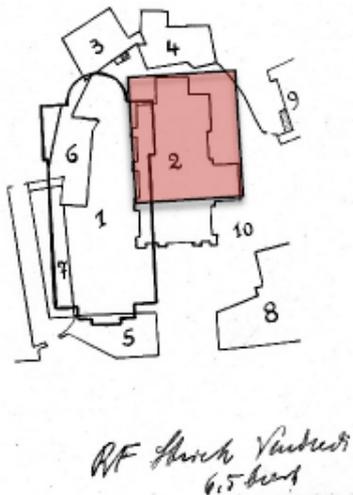


Figure 13: Hypothesis of footprint of the Sainte-Anne's chapel (in red the disappeared part)

We can reuse rules established for the modelling of the church for the hypothetical reconstruction of the chapel (Figure 14).

#### 4.3 Interior part of the church

The inside of the church was also modelled. A Web scene was exported to allow an access there. The Figure 15 illustrates the inside of the church.



Figure 15: Inside of the church.

### 5. DISCUSSION AND CONCLUSION

Through this project we have tried to handle the problem of detailed modelling by testing the possibilities of use of *CityEngine* and procedural modelling.

We can tell at first rather long handling of *CityEngine*, because it is necessary to master the various tools. But this handling is also necessary within the framework of other modelers.

The CGA language is well documented and numerous examples are accessible (CityEngine Blog, 2015; CityEngine Tutorial 2015). However it is not very obvious to manage especially that it is very specific to this environment.

The unique use of *CityEngine* does not allow to reach models with sufficient detailed resolutions. Indeed, it seems efficient to work in parallel with other modelers to be able to develop objects or parts of detailed objects as assets.

Thus, in one hand the implemented method did not proficient within the outline of an isolated modelling. On the other hand, the procedural modelling becomes interesting once we have to model several objects having similar constitutions. This has proved to be very interesting, in particular in the case of the disappeared chapel which has been reconstructed from elements developed for the nave.

Operation	Efficient use of CityEngine	Comments
Learning time	+	Good documentation, adapted to complete virtual cities
GUI	+	Complete GUI, complex for sample operations
Time for modelling setting-up	+	Allows integration of many different formats
DTM	++	Construction and integration from GIS data
Integration of basic outlines	++	Integration from GIS and CAD datasets
Modelling of basic outlines	++	Extrusion tools, including for complex forms
Modelling of facades / faces	++	To decomposition in horizontal and vertical subsections
Modelling from ortho-images	+	Works well in classical cases
Modelling of curved elements	-	To decompose into polygons
Integration of existing models	++	Use of other modelers for complex details
Export Interoperability / Texture	+	Towards GIS or classical formats
Rendering	++	Easy application
Efficiency of unique model	-	Several renderings available
Writing of rules	+	Writing of a rule for a single operation is not effective in this case
Re-use of rules	++	Simple for simple rules, CGI scripts
Creation of rules library	++	Simple but requires a decomposed end structured writing of rules
Export and web visualization	++	Interesting for re-use in case of massive modelling.
		Web service for remote consultation

However, we were able to show on the basis of this project, that a realistic modelling was practicable. Numerous exchange formats offered by *CityEngine* in import and export profit to this promising environment. Indeed, the model can be transmitted and published, exported easily even towards a GIS to enrich it besides semantic and descriptive data.

## REFERENCES

- Chege, P., and Kamau, R. 2012. 3D City models in ArcGIS and CityEngine, Experience Geopower in the east Africa, 3-5 Oct, ESRI, Eastern Africa, 2012.
- CityEngine Tutorial: Essential Skills. Retrieved February, 2015, from: [https://www.youtube.com/watch?v=fLPrAnyyGlg&list=PLWG P11THb9EowkYZ3\\_0L-dZTFcyeqO2zy](https://www.youtube.com/watch?v=fLPrAnyyGlg&list=PLWG P11THb9EowkYZ3_0L-dZTFcyeqO2zy)
- CityEngine Blog: CityEngine Starter Project 2 -Rule file set up, attribute generation and reporting. Retrieved February, 2015, from: <http://cityengine.blogspot.fr/%202013/09/starter-project-2.html>
- Creating Smart 3D City Models with Esri CityEngine. Retrieved February, 2015, from: <https://www.youtube.com/watch?v=zScyrwQLf0k>
- Di Angelo, M., Ferschin, P. and Paskaleva, G., 2012. Shape Grammars for Architectural Heritage, (April), 1<sup>st</sup> International Conference on Architecture & Urban Design, Proceedings 19, Tirana, Albania, pp. 107–116.
- Dylla K., Frischer, B., Mueller, P., Ulmer, A., Haegler, S., 2009. Rome Reborn 2.0: A Case Study of Virtual City Reconstruction Using Procedural Modelling Techniques, Proceedings CAA conference. pp. 62–66.
- Edvardsson, K. N., 2013. 3D GIS modelling using ESRI's CityEngine: A case study from the University Jaume I in Castellón de la plana Spain, MSc (Geospatial Technologies) Thesis, University Jaume I in Castellón de la Plana Spain.
- Fedczyszyn, J., 2013. Intégration de modèles de sites historiques sous forme de modélisation paramétrique dans CityEngine, Projet de Recherche Technologique INSA Strasbourg, 20 pages.
- Geoplanit, 2014. A very quick Rule Wizard for CityEngine tutorial | GeoPlanIT. Retrieved February, 2015, from: [http://www.geoplanit.co.uk/?page\\_id=457](http://www.geoplanit.co.uk/?page_id=457)
- Hohmann, B., Havemann, S., Krispel, U. and Fellner, D. 2010. A GML shape grammar for semantically enriched 3D building models, Computers & Graphics, Elsevier, 34, pp. 322–334.
- Jacquot, K., 2010. Restitution des plans reliefs datant du XIX<sup>ème</sup> siècle, Mémoire de Master Design Global, ENSA de Nancy, Univ. Poincaré Nancy I, INPL, pp.25– 51.
- Koehl, M., 2008. SIG 3D ET 3D dans les SIG : Application aux modèles patrimoniaux. HAL archives-ouvertes.fr, 15 pages.
- Koehl, M., Fedczyszyn, J., 2014. Modelling historic site as parametric model. Application to the Engelbourg castle - Thann, France. 3DGeoInfo proceedings, Dubai, pp. 169-183.
- Mueller, P., 2007. PhD Thesis, ETH Computer Vision Lab. Zürich, Switzerland.
- Mueller, P., Zeng, G., Wonka, P., and Gool, L. V. (2007). Image-based Procedural Modeling of facades. In: SIGGRAPH '07: Proceedings of the 34<sup>th</sup> Annual Conference on Computer Graphics and Interactive Techniques, volume 26, New York, NY, USA. ACM. 10 pages.
- Mueller, P., Zeng, G., Wonka, P. and Van Gool, L. 2012. Mise en œuvre du logiciel CityEngine pour la modélisation 3D d'un territoire urbain.
- Pal Singh, S., Jain, K. and Ravibabu Mandla, V., 2014. Image based Virtual 3D Campus modelling by using CityEngine. American Journal of Engineering, 2(1), pp.1–10.
- Piccoli, Ch., 2013. CityEngine for Archaeology. In: 3D GIS for mapping the Via Appia. VU University Amsterdam, (2013).
- Schirmer, P. and Kawagishi, N., 2011. Using shape grammars as a rule based approach in urban planning -a report on practice, eCAADe 29 City Modelling, pp. 116-124.
- Specht, M. and Van Maren, G., 2014. Creating and sharing Rule Packages with CityEngine. In Creating and sharing Rule Packages. Available at: <http://video.esri.com/watch/3266/creating-and-sharing-rule-packages-with-cityengine>
- Sostaric, Z., 2012. Using CityEngine 2012 for Geodesign, Esri Asia Pacific User Conference, November 5-7, Auckland, New Zealand, 2012.
- Watson, B. et al., 2008. Procedural Urban Modelling in Practice. IEEE Computer, (June), pp. 18–26. Available at: <http://www.peterwonka.net/Publications/pdfs/2008.CGA.Watson.ProceduralModelingTutorial.pdf>