D.8.2 – Second Year Report
WP8 – Synthesis of 3D Artefacts

Version 2.2 - FINAL

27 November 2010

Grant Agreement number:	231809
Project acronym:	3D-COFORM
Project title:	Tools and Expertise for 3D Collection Formation
Funding Scheme:	FP7
Project co-ordinator name, Title and Organisation:	Prof David Arnold, University of Brighton
Tel:	+44 1273 642400
Fax:	+44 1273 642160
E-mail:	D.Arnold@brighton.ac.uk
Project website address:	www.3d-coform.eu

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 231809.

Authors: Luc Van Gool, Frédéric Bosché

Eidgenossiche Technische Hochschule Zurich (ETHZ)

Contributing partner organizations: University of Brighton (UoB)

Consiglio Nazionale Delle Ricerche ISTI (CNR-ISTI)

Fraunhofer Gesellschaft Zur Forderung Der Angewandt Forschung E.V. (Fh-IGD)

Technische Universitaet Graz (TU Graz)

University of East Anglia (UEA)

Table of Content

1 Executive Summary

This deliverable, D8.2, presents the work conducted within Work Package 8 (WP8) during the second year of the 3D-COFORM project. This work mainly consisted of the development of the different software tools included in this WP, namely CityEngine, the Pictorial Data Modelling tool, the Numerobis component (a GML shape compositor based on data flow networks), the Shape Sketching component and the Fragment Re-assembler. Regarding Task 8.1, some work is also reported on the creation of procedural grammar rules for the Brighton Regency architectural style to be used for the digital reconstruction of Brighton at the time of that period.

Overall, this document shows that WP8 has made significant progress towards completion of its deliverables. The framework of the Fragment re-Assembler, Shape Sketching and Numerobis components are now all in place. For the first two, the main functionalities have also been developed, and only a few less critical ones are missing, including the harmonisation of the produced functionalities into integrated software packages (planned for the first part of Year 3). Their overall integration with the RI and the rest of the 3D-COFORM infrastructure are also planned for Year 3. Regarding the Numerobis component, a change of approach resulted from the Year 1 Review (from a fully automated, to a user driven approach). This led to the need for new modelling functionalities that had to be researched, which thus resulted in delays. Progress is nonetheless reported here, and a functioning system is now expected to be delivered and tested by the end of Year 3. The CityEngine 3D-COFORM Edition and the Footprint Extractor are now fully functional including with respect to RI integration. Nonetheless, work on these is not finished. Further improvements are still expected during Year 3 on the Footprint 9 È ú NJ l· Ö ú 2 NJ Ω ă Y l· λ y on the automatic modelling capabilities of CityEngine. l· ă

It must be noted that CityEngine and Footprint Extractor have already been part of the first round of 3D COFORM user testing organized during Year 2, with the testing procedures and datasets prepared by the partners. The testing results are however not reported here, but in D10.2, Second Year Report on Business Strand. Feedback obtained during these tests resulted in adjustments in the functional requirements (reported in D3.2, Second Year Report on Repository Infrastructure) and their implementations (discussed herein).

This document presents work per Task (8.1, 8.2 and 8.3). Each task reviews the work planned for Year 2, the work actually achieved, the deviations, and finally the plans for Year 3. The publications that resulted from the work done during Year 2 are listed and appended to the document.

2 Objectives

The 3D-COFORM framework and its components have been divided into four clusters:

1. Acquiring and Processing (A&P), encompassing the developments in WP4/WP5
2. Integrated Viewer/Browser (IVB), encompassing the developments in WP6/WP7
3. Modelling and Presenting (M&P), encompassing the developments in WP8/WP9
4. Repository Infrastructure (RI), encompassing the developments in WP3

The main objective of WP8 is to create 3D-COFORM components enabling the synthesis of 3D artefacts (objects and scenes). Two sub-goals are identified: (1) the modelling of the 3D artefacts using procedural methods; and (2) the assembly of fragmented artefacts.

Within the first sub-goal, the partners distinguish two types of 3D artefacts: individual objects and large scenes. For individual objects (Task 8.2), the objective is to develop two 3D modelling components (the Numerobis and Shape Sketching components) that enable Cultural Heritage (CH) professionals interested in a particular shape class to produce parametric shape templates for that class. The main goal of the components is to provide means of creating procedural models without programming or scripting. In the case of the Numerobis component, the templates it produces are specifically planned to be used for matching to digitized artefacts, as covered in WP5 Task 5.3, so that a synthesized, but semantically-rich, representation of that artefact can be obtained. With respect to large scenes (Task 8.1), the objective is to develop tools (Footprint Extractor, and extensions to CityEngine) to assist CH experts in rapidly building detailed hypotheses about large-scale sites (e.g. the past of landscapes, villages and cities) for which little or no structure is still standing. These synthesized scenes will be reconstructed by taking as much as possible of the existing evidence into account, i.e. GIS data, maps, contours, sketches and drawings. Since some of this information, in particular maps, is often hand drawn, an important part of this task is the development of tools for their digitization (with the difficult challenge of maintaining the semantic information they contain). In addition, Task 8.1 aims at testing and demonstrating, through a couple of examples, the capacities of CityEngine and Footprint Extractor to aid the CH professional in creating realistic hypotheses and/or reconstructions of heritage sites based on limited evidence.

The second sub-goal, the assembly of fragmented artefacts (Task 8.3), aims to develop intuitive components and explore automated approaches to re-assemble artefacts (which can be either dismantled complex artefacts or fragmented items), of which the individual elements are digitized in 3D and available in the 3D-COFORM Repository Infrastructure (RI). The goal is to have a process driven by expert users (an art historian or a restorer) with semi-automatic procedures to help the user in the fitting and reassembling task.

3 Work Review

3.1 Task 8.1 – Site modelling tools for CH experts

3.1.1 Work planned

For Task 8.1, the goals for Year 2 were to (1) study a first architectural style for creation of a set of rules usable in CityEngine, (2) improve the functionalities delivered by the Footprint Extractor as well as its integration in the 3D-COFORM framework (RI), (3) improve integration of CityEngine in the 3D-COFORM framework, and (4) work on how to model and present uncertainty in a procedural model of a CH site.

3.1.2 Work performed

We present the work performed during Year 2 in three subsections: (1) *CityEngine*, which focuses on development of that software package, the problem of modelling and presenting uncertainty in CityEngine procedural models, and the assessment of its use by an architect through a specific case (Brighton Regency period); (2) *Footprint Extractor*, with focus on the development of that software package; and finally (3) *Integration*, which presents the results achieved by the partners for integrating their work within 3D-COFORM.

CityEngine

Software Development

We have worked on an in-depth analysis of the reconstruction uncertainty in the field of CH and how to represent it with CGA Shape grammars used by CityEngine. The following gives a summary of our research article [T8.1.1]

The Problem of Reconstruction Uncertainty: Virtual 3D models of monuments and sites that have largely disappeared can serve many purposes. They allow experts to visualize and investigate features of a site, which may otherwise be difficult to appreciate. Frischer et al. [1] give an overview of the history of Cultural Virtual Reality (CVR) and discuss the need for aesthetic, scientific and technical standards. One [illegible] sites is that *the more convincing a reconstruction* becomes, the more the general *public may take the correctness of every detail for granted*, even if part of the reconstruction is based on *not much more than a dedicated guess* or a hypothesis.

It should remain possible to visualize the levels of uncertainty for all parts of a model. Rationales behind particular completions and choices should be documented, preferably also as annotations to the model [illegible] its principles (e.g. [2] [illegible] [4] and [illegible]

6). The LC lists a number of fundamental principles for 3D visualization methods in cultural heritage. Similar issues have also been raised by Frischer, a pioneer in 3D modelling of cultural heritage: [3]. ά b ž ű ƛ Ŏ Ŝ ł ō f Ŝ Ǝ ł LJ ă ł NJ Ŝ NJ Ŝ LJ NJ Ŝ ă Ŝ y ˇ ú Ŝ Ř ō è ű K Ŝ Ŧ ł Ŏ ú ű K ł initial information (what were the initial data?) and by the use of peremptory single reconstruction ó ƛ ű K ž dz ú ž Ŧ Ŧ Ŝ NJ Ɔ Anyone of the other hand, 3D computer models should not be judged too harshly either. Quoting [4] Y ά ± w ƛ ă ű K Ŝ Y ž Ř Ŝ NJ y ˇ Ó ł Ŝ Nj ă LJ ž ă y ã ƛ ž ō Ŧ f Ŝ ú ć K Ŝ NJ Ŝ ł Ŏ ž ú using water. Ŏ ž f . ž One need only has to think of the Halicarnassos mausoleum to have a vivid example of how different such hand-drawn reconstructions could be [4]. Omitting uncertain parts would basically only leave one with thin air.

Another danger lies in oversimplification, omission and use of effect placeholder shapes and repetitions. These often used tools may convey a false impression of technological sophistication and do not necessarily let observers infer uncertainty. Moreover, such models will almost certainly fail to generate interest from the public. Especially younger viewers need visually compelling images to retain their interest, given their constant exposure to impressive graphical imagery in games and movies [8].

Alternatively, what we propose in paper [T8.1.1] is to use the increased efficiency of procedural modelling: Rather than using coloration, levels of transparency or non-photorealistic rendering to indicate that one is not quite that certain about a part of a model, one can produce several, realistic models. Each of these models can be truly enjoyed by the public (coloration or transparency invariably ruin the experience). Thinking about uncertainty automatically leads to the idea of probability distributions. One way of representing those is to sample them, with a sufficiently large number of samples. Rather than producing the model, one could therefore in analogy create a representative multitude of models. Each model is a sample for the underlying, more abstract and semantic description, which also contains explicit indications of uncertainty and expert annotations. Presenting such models in sequence gives the public a good impression of how it may have been, while still clearly bringing home the message of not being certain across the board.

Representing Uncertainty with the CGA Shape Grammar: The CGA grammar example in Figure 1 shows how basic uncertainty and sources can be embedded in the CGA source code. Uncertainty about the window dimensions is encoded into shape attributes *winW* and *winH* where the values have certain probabilities assigned. The actual value of the attribute is determined prior to the generation process and stays the same for each call to Tile and Split. In contrast, rule *SubR* uses conditions per successor list to decide on the window height. In this latter case, each window would potentially get a different height.

```
attr randomly = 0
attr winW = 2 // fixed, no uncertainty
attr winH =  60% : 1.5 // in meters, from "archaeological source"
             20% : 1.6 // in meters, from "another archaeological source"
            else : 1.7 // in meters

Tile -->
     case randomly == 0: split (x) { D1: Wall | winW : Sub  | D1: Wall }
     else:               split (x) { D1: Wall | winW : SubR | D1: Wall }

// sub tile alternative 1
Sub -->
     split (y) {D1: Wall | winH : Window | D1: Wall}

// sub tile alternative 2
SubR -->
      60% : split (y) {D1: Wall | 1.5 : Win | D1: Wall}
      20% : split (y) {D1: Wall | 1.6 : Win | D1: Wall}
     else : split (y) {D1: Wall | 1.7 : Win | D1: Wall}
```

Figure1: Examples of CGA grammar rule with embedded uncertainty.

Figure2 shows two examples for multiple models per view. On the left, for each view two hypotheses are created in order to compare roof types, floor layouts and door/window dimensions. Note that in this example the footprints of the buildings and their functions (e.g. atrium) are fixed, that is, there is a high certainty of correctness for the footprints. On the right, the example uses a subset of the rules created Ŧ 2 NJ ű K Ŝ á w 2 Y Ŝ . We constrained the rule attributes such that the lower floor always has the same layout (door-window-door) and always contains a larger door at the left and a smaller door at the right. The window between the doors has a probability of 50% (the first five facades are with, the last 5 without the window). The upper floor uses the original range of layouts designed for á w 2 Y Ŝ w Ŝ ō 2 NJ y ́ é Φ

CityEngine Enhancements for Modeling Uncertainty: CityEngine has received two major enhancements in the last period concerning its use in CH and specifically to model reconstruction uncertainty. First, a á ± ƛ ă dz l · f / D ! é Y 2 Ř Ŝ ƛ ă y ́ 2 ó l · Ó l · ƛ dz ă l · ƛ ō y f ∃ Ŝ ű ó K K Ŝ ƛ Ŏ d K ± ƛ l · ă t dz l 2 t ó ă t N paradigm. Figure3 shows an example of the new VCGA mode. Please note the textual representation is kept in sync in the background and can still be used to add more detailed annotations. Then, CityEngine has received numerous new commands which make the grammar more expressive and ultimately help encoding uncertainty and variation: convexify, rotateUV, deleteUV, normalizeUV, various math functions, assetInfo, assetsSortRatio, assetsSize, count, fileExists, fileSearch, find, imageInfo, ƛ Y l · ∃ Ŝ { 2 NJ ű w l · ű ƛ 2 Σ Y ƛ NJ NJ 2 NJ Σ . Y The NJ NJ l · he { Ŏ 2 y s t ê m 2 Σ provides detailed { Ŏ 2 LJ Ŝ Σ documentation of these: http://www.procedural.com/support/documentation.html

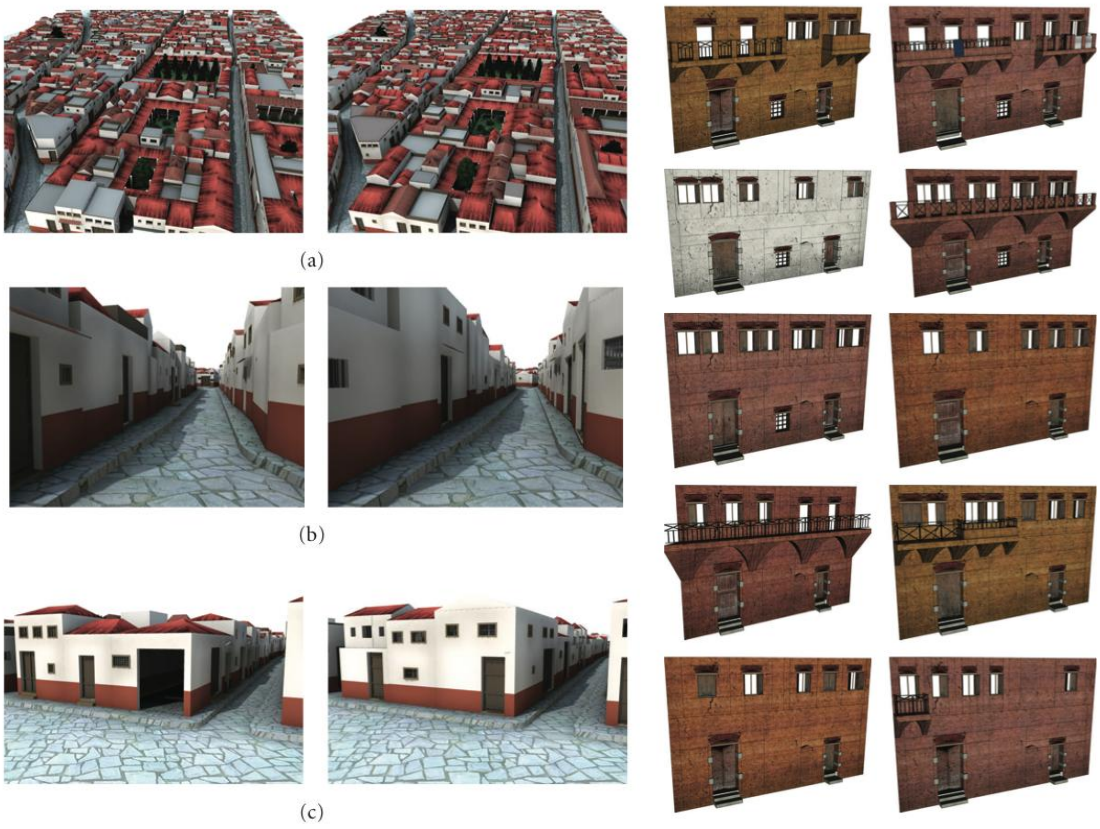

Figure2: Examples for multiple models per view (left) For each view two hypotheses are created in order to compare roof types, floor layouts and door/window dimensions (right) Example using a subset of the NJdz f Ŝ ă ŎNJ Ŝ I· ú Ŝ Ř Ŧ 2 NJ ú KVE constrained the vocabulary such that the lower floor always has the same layout (door-window-door) and always contains a larger door at the left and a smaller door at the right. The window between the doors has a probability of 50% (the first five facades are with, the last 5 without the window). The upper floor uses the original range of layouts designed for á w 2 Y Ŝ w Ŝ ō 2 NJ y ε Φ

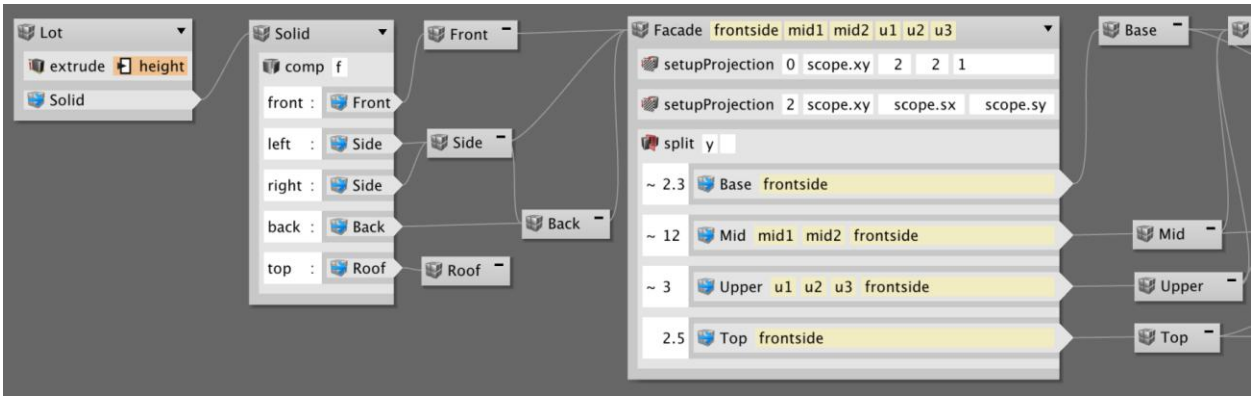

Figure3: Grammar writing in VCGA mode.

Assessing CityEngine in practice:

It must first be noted that this task effectively started on 4th September 2010. In this capacity the first case study, Regency Brighton and Hove, has been selected for the working up of a set of grammar rules to be used in the CityEngine. Before any technical work (i.e. grammar writing) could be performed, an extensive background analysis of the architectural style of the period has been conducted. In addition, GIS data from Ordnance Survey (OS) Mastermap have been obtained (and already successfully imported into CityEngine). Fifteen historic maps of Brighton ranging from 1779 to 1834 have been also scanned and are being geo-registered with the present day GIS information. Furthermore, work has started on understanding historical street network growing patterns, city block subdivision patterns, and naturally building decomposition patterns (volume, roof, façade). These are the patterns that need to be encoded as CityEngine grammars. To this end a multi-pronged approach is planned that encompasses four ways of validating the rules. These are in the first instance to conduct historical research into contemporary texts, theories and design templates in use at the time. Secondly, to gather contemporary maps and plans of buildings that were proposed or built. Thirdly, to create a building stock inventory of buildings ű Ƙ ł ű ă ű ƛ f f ă ű ł y ř ł y ř ű Ƙ ł ú ř ł ú Ŝ Ŧ N J 2 Y ű Ƙ Ŝ L J Ŝ N J ƛ 2 ř ł y in order to correctly characterize the key features of the Regency style. First results of this work are summarized in the as-yet confidential document presented in Appendix 1.

With respect to the actual generation of grammars, an initial area of Regency Brighton, Brunswick Square (Figure 4), has been chosen for reconstruction in the first instance. The person conducting this study has been simultaneously (1) learning the grammar writing process, through multiple tutorials available in the CityEngine, and (2) trying to apply this in the Brunswick Square test-case. Some technical issues have already been faced. For instance, vertex ordering in the OS data is incompatible with the ordering required in CE in order to guarantee the correct orientation of front facades. A workaround has been proposed through a script to re-order the vertices in a CAD package.

We note that, at the time of the writing of this report, the person conducted this modelling work is visiting ETHZ in order to accelerate CityEngine modelling tasks by leveraging the expertise of this partner with this software package. It is therefore expected that first modelling results be presented at the Year 2 review.

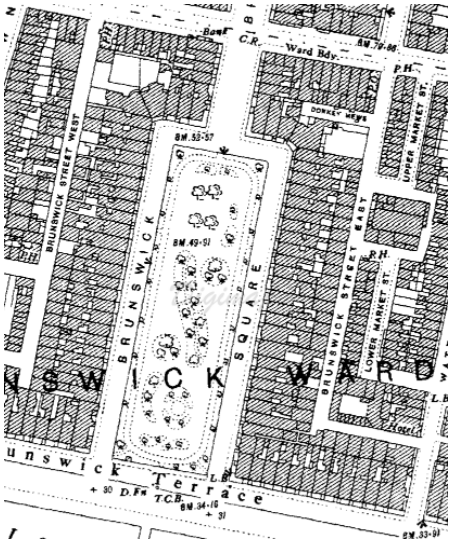

Figure 4: Map of building footprints of Brunswick Square

Footprint Extractor

The *Footprint Extractor* has evolved since the testing workshop in Crete, June 2010. Feedback from this workshop led to the addition of features for editing shapes, manual sketching, and more interactivity with the results. These features allow manual improvement of the results in areas where the automatic process currently provides inadequate results. Figure 5 shows the new feature developed for *manual editing of footprints*. Then, an *alignment and matching* feature has been created, allowing modern vector data to be aligned to historic maps. This modern data can then be compared to historic maps, and in cases where buildings are judged to match, the accurate modern data can be used. This also illustrates changes in an area over time. This feature is shown in Figure 6.

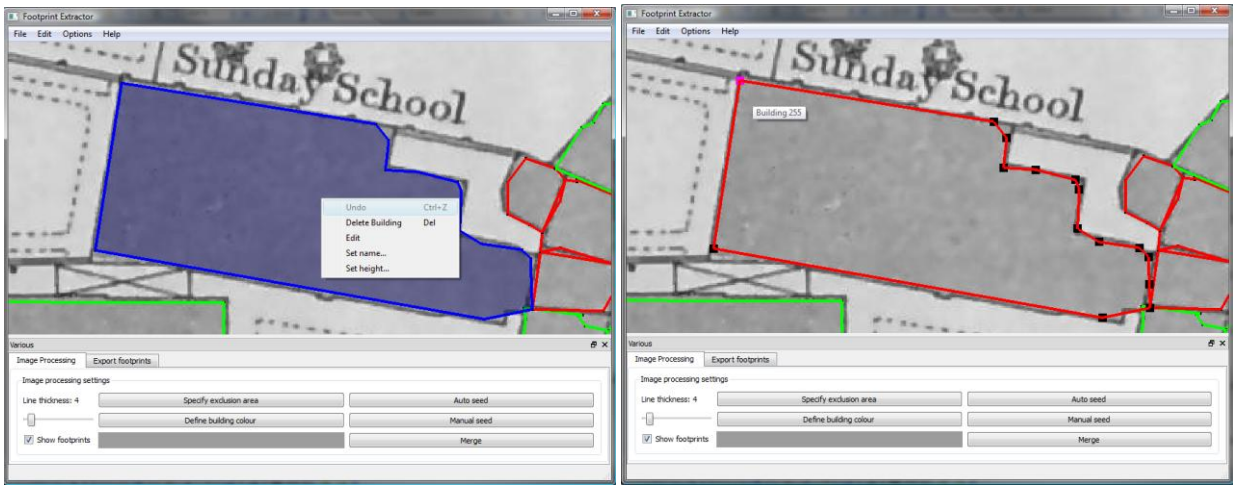

Figure 5: Added feature to the Footprint Extractor for editing footprints (left) A footprint automatically extracted; (right) the same footprint in manual editing mode.

Figure 6: Added feature to the Footprint Extractor for aligning maps and vector data (top-left) Historical map; (top-right) input vector data (bottom-left) Alignment result; (bottom-right) Final result showing the preserved vector data after matching.

Further image processing techniques have been investigated to improve the quality of results and more reliably detect building shapes. First, edge detection is now being used to create an image mask. This mask allows for more accurate and more efficient detection of building shapes. To further enhance this, corner detection algorithms are used on this image mask. The results of this corner detection are compared to the corners of the shapes that are extracted in order to check the positions of the shape corners for accuracy. These can be corrected if necessary to improve the quality of the results. Then, some post-processing of the results now intelligently simplifies the vectorised shapes. This attempts to correct aliasing issues (which can cause a straight line to appear as many small jagged lines) and smooth the lines such that they form a more accurate building shape.

Figure 7 shows screenshots of the Footprint Extractor during and at the end of the processing of an old digitized map.

Integration

The integration of the CityEngine and Footprint Extractor through the 3D-COFORM RI has been one of the first to be implemented and demonstrated at the beginning of Y2. The Footprint extractor is able to ingest the original map image, the extracted vectorised map (as Shapefile) and provenance metadata into the RI. Then, CityEngine includes a new feature for retrieving data from the repository (e.g. vectorised maps) and one for the ingestion of CityEngine projects along with provenance Metadata. Currently, retrieval is performed by manual supply of the UUID of the vectorised map (Figure 8). While

this feature can be sufficient, a more integrated approach for retrieving UUIDs through the use of the IVB can be envisaged.

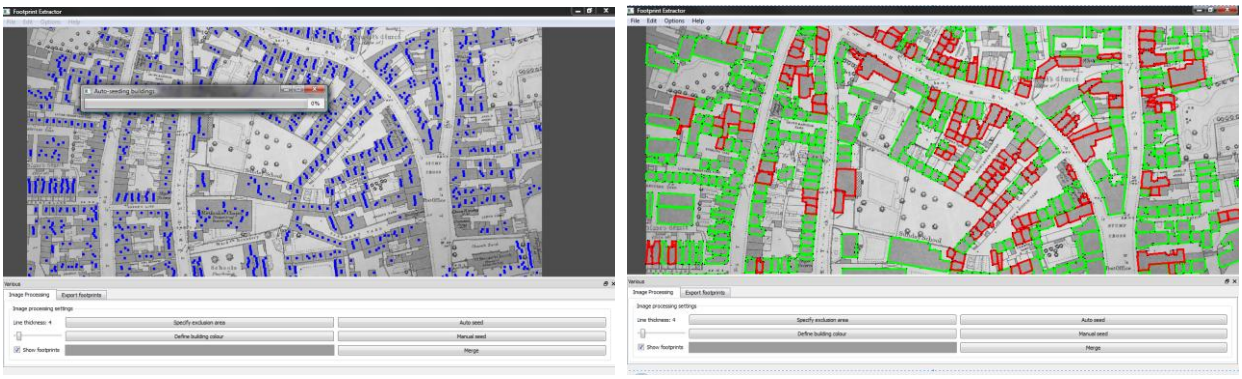

Figure 7: (left) The CH digitized map under processing; (right) The result of the automated footprint extraction: the non-intersecting footprints are shown in green, while the intersecting ones are shown in red, so that the user knows where his/her intervention is mainly required.

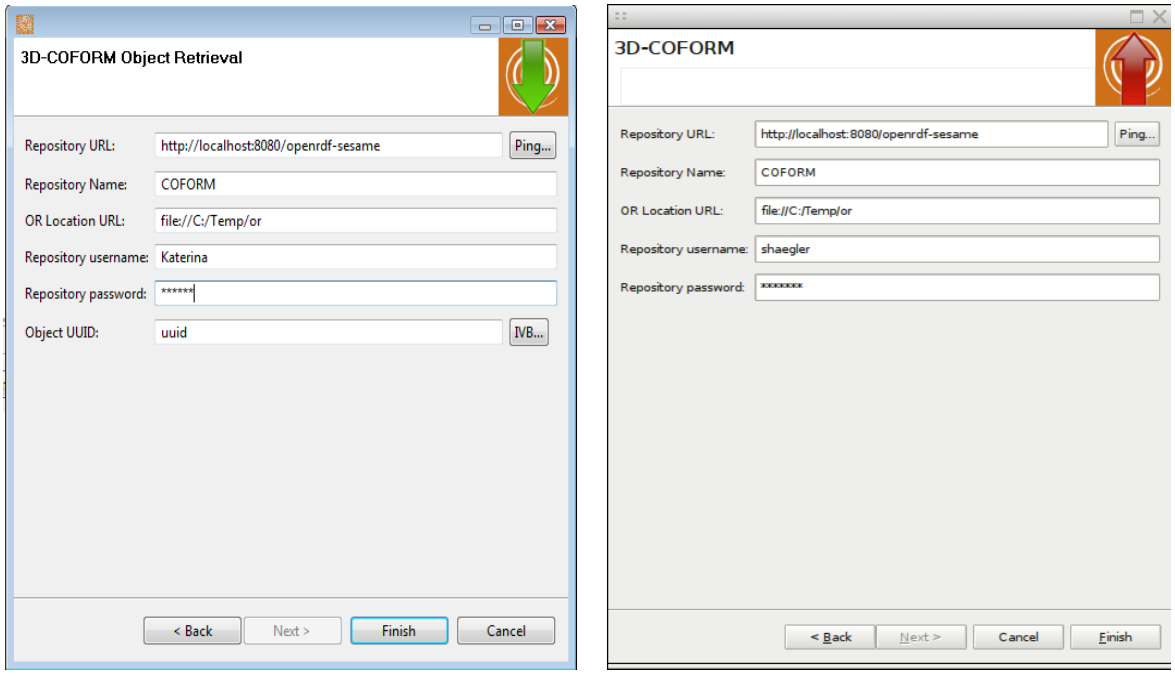

Figure 8: Ingestion and retrieval within CityEngine: (left) the window widget for retrieval of data from the RI. Note that this can be used for any type of data, not only vectorised ones; (right) the window widget for ingestion of CityEngine projects into the RI.

3.1.3 Deviation from work plan

Overall, the Task 8.1 team considers that no serious deviations from plan are to be reported at this point.

Nonetheless, it is to be noted that work on the *modelling* of uncertainty in CityEngine models has not been started yet due to a switch of focus towards WP5, 3D Artefact Processing, during Year 2. The work for WP5 having now been delivered, focus will return to this task.

Furthermore, work on the study of the Brighton Regency style for the writing of procedural grammars within CityEngine has not progressed as much as expected. While this is unfortunate, it is due to the difficulty for the partner to hire an adequate person for the work. A person has now been hired and work is being conducted very actively and thoroughly. First results will be shown shortly. This person has achieved rapid progresses and a visit to ETH is planned at the end of November in order for the user to gain further insights on the functionalities of CityEngine, and collaboratively identify solutions to some technical issues.

3.1.4 Plans for the next period

CityEngine: Some work has been reported in both the *modelling* and *presentation* of uncertainty. Until the end of the project, we now expect to focus further on the issue of *modelling* uncertainty. This is critical, as the current approach may not be sufficiently flexible for CH users. In parallel, the analysis of the feasibility of using CityEngine, and more generally procedural modelling, for CH applications will be continued. Focus will first remain on the Brighton Regency test case, but it is also hoped that work on a second example can be reported at the end of Year 3.

Footprint Extractor: The main aim is to continue improving the quality of the extractions. We intend to simplify and semi-automate the pre-processing that images currently require. We may also investigate pattern detection in order to cope with maps where buildings are shaded with cross-hatching. More intelligent line processing, both for detection and for merging neighbouring footprints, may help to improve results. Finally, automatic adjustment of image processing thresholds, if possible, would help to make the software more adaptable.

Integration: For Year 3, our goal is to maintain the current integration working, since the RI will likely undergo some interface changes. The integrity of data at ingestion time remains an aspect that has not been addressed yet. Finally, if necessary, additional provenance information will be gathered for inclusion at the time of data ingestion.

3.2 Task 8.2 – Toolkit for non-expert users to create procedural models

3.2.1 Work planned

Task 8.2 pursues a twofold strategy, with two approaches for interactive design of procedural models:

- The Numerobis component is a GML extension for data flow networks. The user can interactively issue modelling operations, and internally the dependency between the operations is logged. Each operation has certain parameters, which are either free or bound parameters. The user can group a sequence of operations and make them available as a new modelling tool.
- The Shape Sketching component is for the processing of mouse strokes. The user can activate a ú 2 2 f LJ l· yŽŘ ú á l· LJ l· ú Kan input parameter for different types of sketching functionalities. The sketching is encoded in GML, and consequently so are the modelled objects.

3.2.2 Work performed

Numerobis

For the important task of enabling non-expert users to create procedural models we are currently pursuing a data flow graph approach. The modelling operations issued by a user are logged, the input and outputs of each operation are recorded, and a graph representation is created internally that contains the flow of operations. There is one node for each object (point, line, plane, polyhedron, etc), and one node for each operation (intersect, copy-translate, copy-rotate, extrude, subdivide, etc.). The user primarily works in the 3D environment, and the graph is assembled in the background. Operations on the graph allow grouping certain operations to create higher-level operations. This approach is inspired by (a) Google SketchUp, and (b) Rhino Grasshopper:

Google SketchUp ŐdzNJNJŜ y˝ú f è Ő 2 Y Ŝ ă Ő f 2 ă Ŝ ă ú ú 2 l· á o Ř Y 2 -Ř Ŝ f f ŜNJ professional users all over the world, for its few simple but powerful modelling operations. What we like about it:

- Few but well defined modelling operations (closed, complete for large class of models)
- Use of helper geometry (dotted lines)
- Snaps to reasonable positions (proposes interpretation when dragging the mouse)
- Exact modelling possible (just enter a number after a translation to make it exact)

What we do *not* like about it:

- Uses meshes, which are brittle and difficult to log
- Model has no procedural structure

Instead, we propose to use convex polyhedral, and we also show the data flow of the model.

Rhino is a very popular NURBS modeller, a standard tool for high-quality surfaces in industrial design. Grasshopper is a Rhino plugin providing in fact a data flow graph editor, with which interesting and powerful procedural constructions can be created. Grasshopper defines the current state of the art in, e.g. freeform architecture. What we do *not* like about it:

- First the data flow graph must be edited (curve node),
- The 3D construction window is used to instantiate geometry (draw curve interactively, then attach to curve node).

Instead, we want to let the user work in 3D as usual, and use the data flow just for illustration so that the user sees an explicit representation of the operations she issued, and the dependencies between them. The data flow graph can then be used to define modelling procedures, which can then be made available as new modelling tools. In this way, the data flow graph becomes a tool for structuring the interactive modelling workflow.

An example is shown in Figure 9. Modelling starts with a box that is procedurally created from two points in a plane. The 2D point parameters (0,0) and (10,10) are converted to 3D coordinates by the *pointInPlane* node, and the *makeBox* node creates a box that is 10 units high. The result of *pointInPlane* encodes the plane as well. The split function in (B) takes the box as input as well as a split direction *globalYZ* and a split interval. The third of the four results of (B) is selected to apply the next split (C), the second result of which is discarded. Note that the graph node (D) refers to the GML function *void*, and not to some object called void. All nodes in the graph are considered operations. Input values are considered constant functions that can be evaluated to produce an array of four integer values on the operand stack. *[1 -1 -1]* is an array of four integer values on the operand stack. reduced by identifying all intervals, merging three nodes into one. On the one hand this would allow more powerful model changes, on the other this would mean that all the splits are coupled. Another well known, more serious problem is that selections can become ambiguous when models are changed. When (B) produces less than three results, then (C) cannot be applied to its third result. This is the so-called *persistent naming problem*, and there is no general solution for it. The only remedy is to provide a range of selection tools that the user can apply to specify more explicitly which things in the scene are to serve as input parameters.

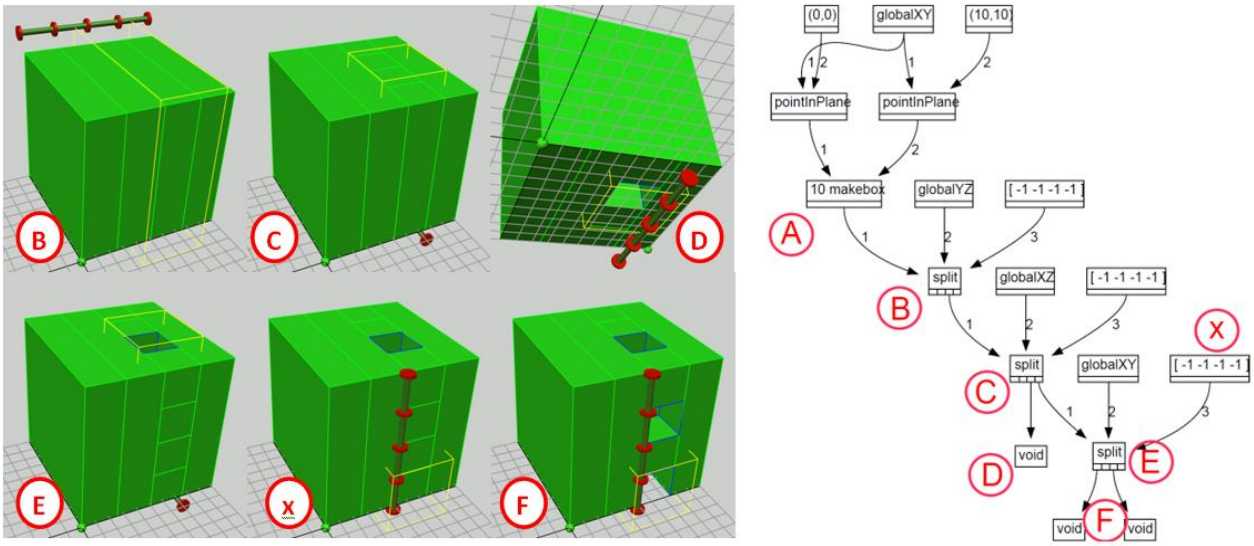

Figure 9: Procedural modelling operations and corresponding dataflow graph, as indicated by the labels.

The current user interface of Numerobis is shown in Figure 10, top left. This is not the end-user interface but the development interface. A procedural modeller with logging functionality requires a very dynamic user interface, both in 2D and in 3D: Each modelling operation has a different set of parameters, which must be shown when browsing through the construction history. The dataflow graph is shown only as a linear list (partial ordering), which is difficult to keep track of (see Figure 10 top left and right). Each operation is also associated with 3D manipulators which are shown automatically whenever the user clicks an object in the scene. By multiple clicking in 3D the user can also navigate through the graph, e.g. to interactively manipulate the operation producing the parent of the parent.

The feature that makes the difference of Numerobis with respect to other procedural modelling tools is shown in Figure 10 (7): The user extracts a new function *A* and adds it to the set of available modelling operations. In this case, *A* comprises only the operations shown in image (6), split in one direction followed by another split in the other applied to the box in the middle. The result is just the central box in the centre. The following images 8-15 show multiple applications of this new modelling tool, and subsequent editing of the tool parameters.

This illustrates that the functionality of the dataflow engine under the hood of Numerobis is operational. The challenge in Year 3 will be to extend it to cover all required grammar operations and, most importantly, to find a method to put it behind an easy-to-use user frontend.

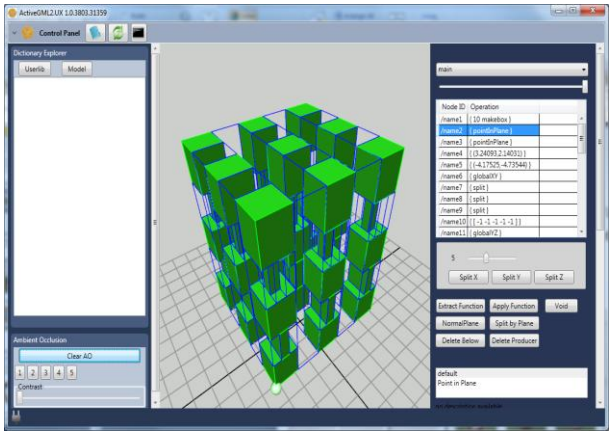

Figure 10: The Numerobis framework. A sequence of parametric construction steps can be extracted as a new function that is then re-applied multiple times. This works even after the basic shape is changed.

Shape Sketching

During Year 2, the Shape Sketching component has been further developed. It offers interactive modelling of shapes with subdivision surfaces by using Euler operators. For the time being, six sketching operations are available:

- Sketch a silhouette of a new object (Figure 11);
- Sketch a silhouette of a rotational extrusion (Figure 12);
- Transfer attributes of the extrusion (Figure 13);
- Sketch the path of a sweep (Figure 14);
- Sketch the silhouette of a lofting operation (Figure 15);
- Split a face with a sketched path (Figure 16).