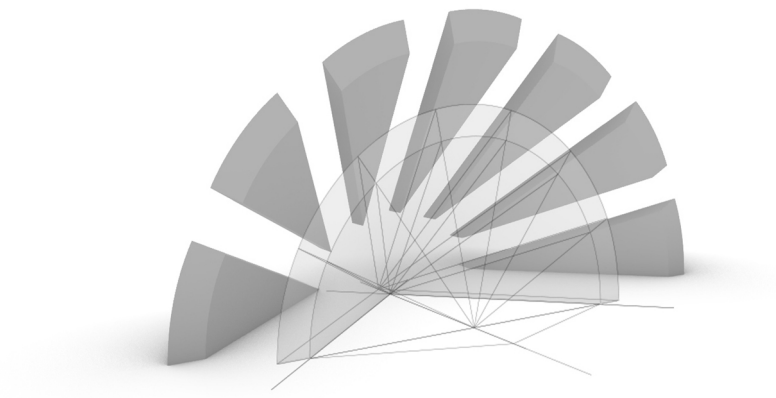
Testo, modello, diagramma: continuità e aggiornamento dei linguaggi per la rappresentazione

Paolo Borin
Devid Campagnolo
Alberto Longhin

Abstract

La creazione di script e algoritmi per la costruzione di forme architettoniche e la gestione informativa dei modelli BIM è diventata ormai pratica comune. La rappresentazione si deve pertanto confrontare con il necessario aggiornamento verso logiche computazionali attraverso linguaggi differenti, che sappiano costruire e operare i modelli tridimensionali. Il presente contributo verifica innanzitutto l'applicazione delle regole di script alle volte coniche trompe, come descritte nei trattati stereotomici. Così facendo, il modello genera molteplici rappresentazioni, che possono abilitare, attraverso opportune connessioni, utilizzi specifici, come quello della verifica strutturale delle volte. D'altra parte, la programmazione visuale genera spesso *script* di difficile comprensione, in mancanza di pratiche e standard per la definizione e organizzazione dei nodi. Ne deriva la necessità di documentare le operazioni e la logica utilizzata, al fine della modifica dello *script* e della comprensione da parte di lettori e interlocutori esterni. Un secondo esempio dimostra così l'applicazione di linguaggi di produzione automatizzata di diagrammi di processo UML, a partire da codice opportunamente scritto e integrabile nelle piattaforme VPL. La rappresentazione parametrica del primo caso viene astratta in un ulteriore diagramma, attraverso l'utilizzo di codice testuale. I due casi, posti a sistema, indagano la relazione tra linguaggi differenti nel contesto della rappresentazione.

Parole chiave
scripting, trattatistica, visual programming, language, diagrammi.

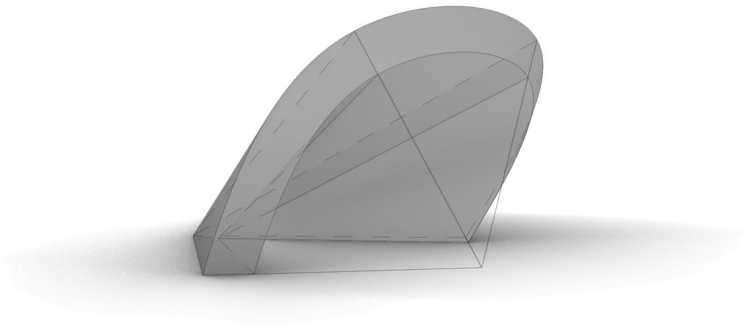


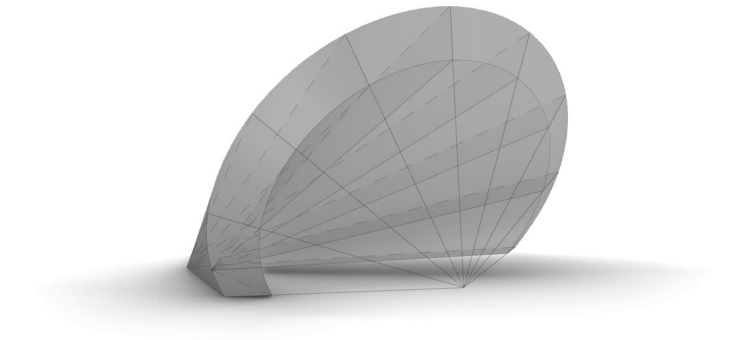
Rappresentazione digitale dell'apparecchiatura teorica di una volta trompe.

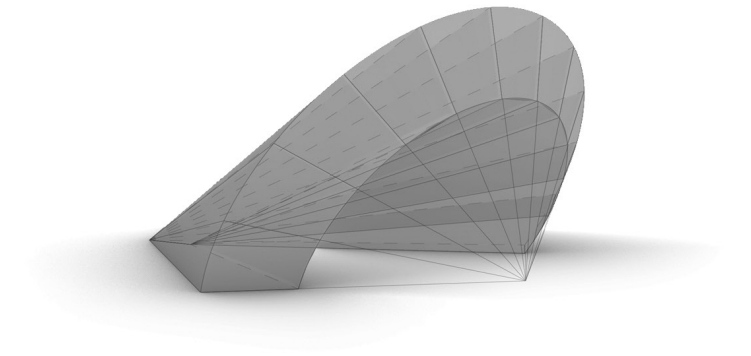
doi.org/10.3280/oa-693.14

Intoduzione

A partire dagli anni 60, la nascita dei codici di progettazione attraverso disegno e progetto automatico (CAD) ha aperto a una nuova era per la rappresentazione architettonica. Nell'ultimo decennio è il termine generale *computational*, a garantire invece un'area di ricerca e sviluppo. Per computazionale si intende la generazione di segni – nell'accezione più ampia di testi, modelli, processi – ottenuti grazie a procedure di scripting e algoritmi. Si tratta di sequenze strutturate di operazioni, spesso di natura matematica, che si basano su parametri generalizzati aventi lo scopo di risolvere una certa classe di problemi attraverso un approccio sistematico e universale.
Nonostante questo oggi venga spesso descritto come un metodo rivoluzionario, il suo impiego come approccio sequenziale e generalizzato, deriva da esempi piuttosto noti nella storia dell'architettura e della rappresentazione. Leon Battista Alberti, nel settimo libro del *De Re Aedificatoria*, propone una serie di istruzioni standardizzate per il calcolo delle proporzioni della base dorica di una colonna [Carpo 2003]. Si ravvisa un'altra testimonianza di un antesignano approccio algoritmico nelle applicazioni per la realizzazione delle apparecchiature stereotomiche di archi e volte. Attraverso rappresentazioni di cantiere (*épure*s), o su carta (*trait*s) e trattati, gli architetti e gli ingegneri per tre secoli hanno dettato istruzioni precise ai *maçon*s per la realizzazione in cantiere di strutture potenzialmente replicabili. Entrambi questi esempi dimostrano la necessaria continuità tra notazione testuale e grafica, che produce, o genera, applicazioni costruite.
A oggi, gli aspetti computazionali sono veicolati attraverso la produzione di *script*, sia nella ricerca formale sia nella gestione informativa dei progetti. La metodologia più comune per questa fase è offerta dagli strumenti di programmazione visuale (VPL, Visual Programming Language), atti a produrre delle funzioni adatte a scopi specifici, in via grafica attraverso una rappresentazione bidimensionale di nodi-funzioni e linee-relazioni. Gli utilizzi variano dalla creazione della forma alla gestione informativa dei progetti BIM.
La situazione attuale porta a riflettere sulla condizione del linguaggio, rapportato alla rappresentazione, e di come questo possa veicolare geometrie e applicazioni [Cocchiarella 2015]. Al celebre linguaggio grafico della rappresentazione, va aggiunta la forma – il codice in linguaggio macchina – con cui si esprime un algoritmo. Nonostante l'attuale disponibilità degli strumenti permetta di produrre *script* attraverso un linguaggio visuale (Visual Programming Language), al loro interno sono presenti, comunque, istruzioni in linguaggi di programmazione monodimensionali, quali Python, C#, ecc. La comprensione di essi implica due conseguenze fondamentali. La prima è la trasformazione della rappresentazione in un risultato conseguente all'esecuzione di procedure definite al di fuori della rappresentazione stessa. Viene effettuata un'astrazione dei processi – logici e geometrici – del progetto o di parti di esso, riducendoli a un solo linguaggio che genera modelli. In secondo luogo, l'informatizzazione dei modelli li abilita a diversi scopi, che vanno anche oltre all'espressione dell'idea progettuale-formale, divenendo oggetti di analisi e contenitori di metadati. Tramite l'uso razionale e un'adeguata strutturazione del linguaggio con cui vengono create le relazioni, si può ampliare il campo di conoscenza trasmissibile del modello definendone proprietà fisiche, meccaniche, informative o relazioni nuove tra diverse entità del progetto che possono mutualmente interagire tra loro. Si assiste così alla possibilità del superamento del limite "statico" consentito solamente dalla rappresentazione della 'forma', abilitando la geometria ad assumere informazioni 'dinamiche' e prestazionali, strumentali alle fasi successive del processo creativo di verifica e realizzazione del progetto architettonico.
Il presente contributo dimostra infine l'invarianza della rappresentazione, in cui essa rimane il risultato di un altro linguaggio testuale, in forma di codice, di cui si deve fornire una necessaria documentazione delle procedure sviluppate all'interno di esso. Il processo di documentazione va strutturato, nuovamente, con opportune rappresentazioni, questa volta diagrammatiche. I diagrammi così creati, come le illustrazioni dei trattati architettonici, guidano il lettore alla comprensione della logica di costruzione del progetto stesso.







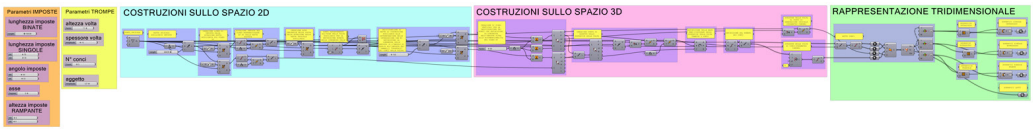


Fig. 1. Grasshopper3d script per la realizzazione della trompe ed esempi di configurazioni risultanti.

Dal testo allo *scripting*: connettere geometria, disegno e simulazione

Nel presente saggio si è improntato uno studio di una volta tipica della pratica stereotomica: la trompe (figura di copertina). Essa è una volta conica, la cui genesi è data dalla semi-rivoluzione di una retta avente un estremo fisso. Il semicono così ottenuto può essere sezionato da diverse superfici in base alla tipologia della configurazione.
Questa soluzione poteva essere impiegata tra due muri intersecanti, sia all'esterno di un edificio al fine di espandere lo spazio di una sua stanza interna, sia all'interno dell'architettura ecclesiastica in presenza di tamburi ottagonali da impostare nello spazio solitamente quadrato che si genera nell'intersezione tra navata principale e transetto [Calvo-López 2020]. È una tipologia molto indagata dai trattatisti europei, grazie alla sua particolarità topologica e azzardo strutturale, tanto da essere definita da De L'Orme "*voûte suspendue en l'air*" [Trevisan 2011, Calvo-Lopez 2020].
Lo studio qui presentato ha affrontato la generalizzazione delle diverse configurazioni di questa volta attraverso operazioni di *scripting* in ambiente Grasshopper (fig. 1).
La genesi della forma parte da una serie di parametri che definiscono la posizione delle generatrici delle imposte che nascono dal vertice del cono. Due parametri regolano la lunghezza delle linee di imposta; un ulteriore governa la posizione della direttrice della trompe, rispetto all'asse del cono. Un secondo gruppo di proprietà è rappresentato dagli angoli: l'angolo di apertura delle imposte è composto dalla somma di due angoli tali che possano definire un cono obliquo. Un terzo insieme di parametri governa la rotazione delle due generatrici sul piano verticale consentendo la generazione di configurazioni rampanti. Infine, nel processo di scripting si definisce l'altezza massima e lo spessore della volta, il numero di conci (*voussoirs*), e l'inclinazione della curva di intersezione frontale. Per la creazione dei singoli conci, si procede generando delle superfici planari dalle linee di costruzione precedentemente descritte.
La procedura utilizzata è capace di creare una molteplicità di configurazioni diverse, tutte geometricamente ammissibili. Esse, tuttavia, non rappresentano soluzioni strutturalmente accettabili. Si è pertanto affrontato lo studio dell'invariante statico di una particolare configurazione, la cosiddetta "trompe fondamentale", tramite lo strumento di calcolo strutturale grafico RhinoVAULT [Rippmann et al. 2012], abilitato per lo stesso ambiente di modellazione di Rhinoceros3D.
Dopo aver ricreato la configurazione attraverso la manipolazione dei parametri, si è applicata la teoria della Thrust Network Analysis e il suo approccio inverso [Block 2009; Block et al. 2014], al fine di valutare gli sforzi agenti sulla struttura, la distribuzione delle forze e i moduli delle risultanti alle imposte sotto l'azione di un carico reale (fig. 2).
Al termine del processo è possibile formulare delle regole di ottimizzazione strutturale andando a ricercare lo spessore minimo tale da assicurare l'equilibrio della struttura. In accordo con le teorie relative alla statica degli archi applicata alle volte [Becchi, Foce 2002; Benvenuto 1991; Heyman 1998; Huerta 2008; Kurrer, Kühn 2009; Ochsendorf 2002], si è quindi proceduto ricercando la rete di spinta che si collocherebbe all'interno del terzo medio della struttura, garantendone l'equilibrio a sola compressione.

Dallo *scripting* al testo: rappresentare per documentare la conoscenza

La documentazione dei processi assume, in ottica di uso intensivo e abituale di programmazione visuale nella progettazione, un ruolo di primaria importanza. Tra le varie, una delle potenzialità della programmazione visuale nella progettazione architettonica risiede nella capacità di generare un risultato coerente con un insieme di input, in un tempo decisamente inferiore a una costruzione manuale.
Il rischio che tuttavia si corre in tale pratica risiede nel produrre uno strumento che, a causa di una scarsa flessibilità data al codice in fase di creazione e della difficoltà di apportarne modifiche, risulti meno performante in termini di tempo rispetto a una risoluzione di programmazione tradizionale monodimensionale [Davis 2016]. Se una previsione a mon-

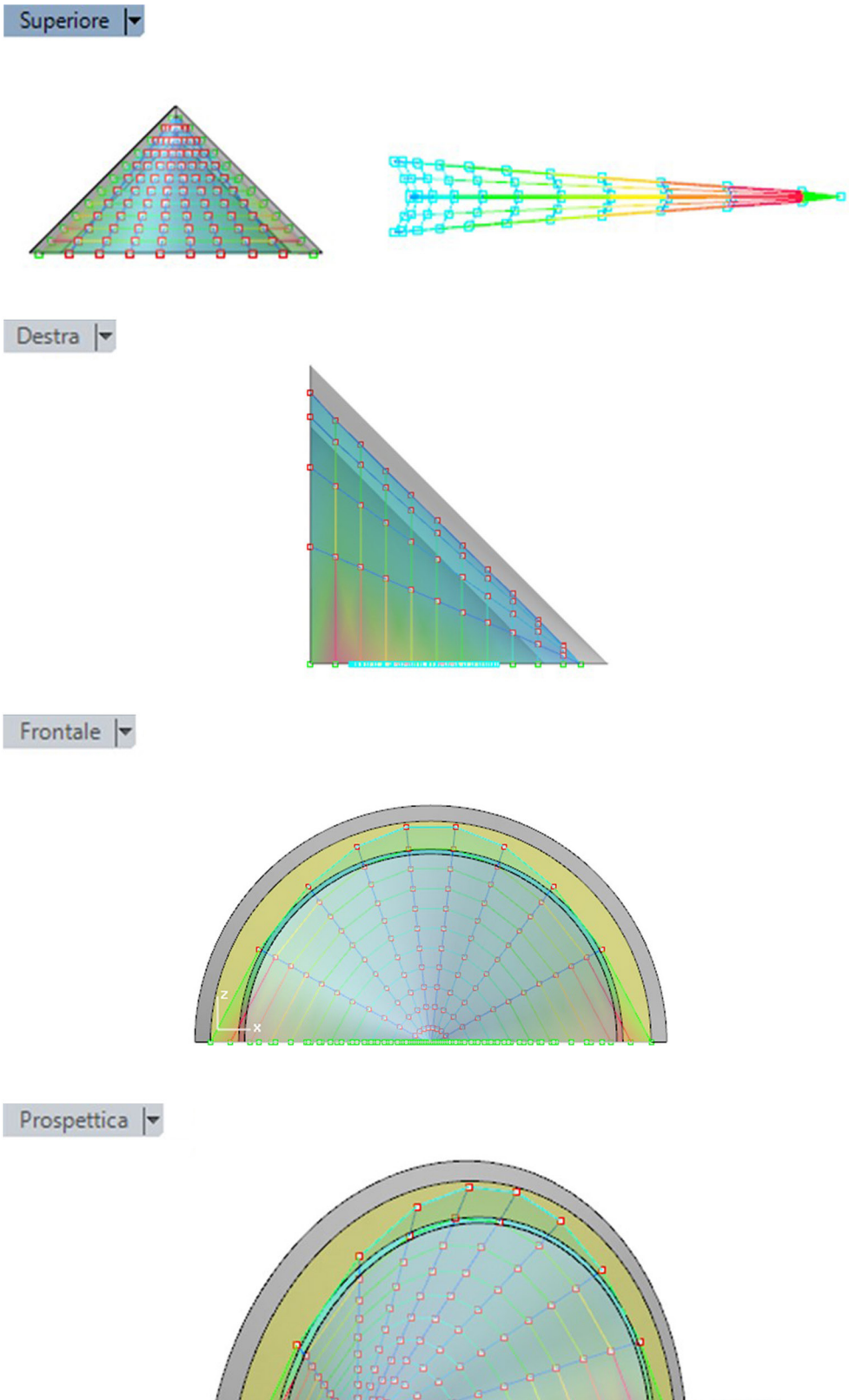


Fig. 2. Configurazione finale della superficie di spinta G all'interno della volta iniziale e individuazione dello spessore minimo risultante dal processo di ottimizzazione.

te delle possibili problematiche che il codice dovrà gestire può influire sulla flessibilità, la documentazione del processo per il funzionamento dello *script* risulta determinante per l'esecuzione delle sue modifiche.
Di conseguenza, documentare un processo in tal senso non acquisisce solo valenza di metodo di trasmissione di una procedura, ma si configura anche come metodo di analisi della stessa: tramite la documentazione, il progettista ha la possibilità di ripercorrere la sequenza logica e funzionale alla base di uno *script* e di individuarne le possibili lacune e criticità. Trasmettere una procedura è inoltre di primaria importanza nei casi in cui l'algoritmo debba essere utilizzato e quindi modificato da differenti attori nel tempo. L'uso da parte di diversi attori – nonché dell'ideatore stesso, ma per differenti progetti – richiede che l'algoritmo venga corredato da documenti che ne garantiscano una facile interpretazione, al fine di operare le necessarie modifiche, qualora ve ne siano, per adattare lo strumento al contesto.
Il primo e principale metodo di documentazione in ambito di VPL è la strutturazione del modello nell'ambiente di programmazione visuale stesso. Tale processo contribuisce in modo notevole alla comprensione del codice in esso contenuto [Davis, Peters 2013].
Tuttavia, in un'ottica di condivisione di algoritmi è necessario completare questa parte con un'ulteriore documentazione in formati aperti e secondo una struttura standardizzata al fine di garantire universalità d'accesso all'informazione. Utile a questo scopo è l'impiego di grafici UML che ne sintetizzino le operazioni peculiari fornendo una visualizzazione d'insieme del processo. Nel caso studio proposto, al fine di garantire una trasmissione di conoscenza che prescinda dall'uso particolare di una piattaforma o formato proprietario è stata realizzata una documentazione del processo tramite Markdown [1].
Il presente lavoro cerca di documentare il caso studio precedente attraverso criteri di universalità, in modo da astrarre il risultato dalle forme di rappresentazione dello strumento VPL. Si tratta di visualizzare la procedura seguita dagli autori per la creazione della volta conica trompe, in modo da offrire uno strumento di visualizzazione della conoscenza acquisita e codificata nello *script* già descritto.

```mermaid

sequenceDiagram
autonumber

participant I as Input
participant 2D as 2D Construction
participant 3D as 3D Construction
participant RT as 3D Representation
participant O as Output

activate I
activate 2D
2D->>2D: Start Point
2D->>2D: Auxiliary line
I->>2D: Impost angle
2D->>2D: Impost axes
I->>2D: Impost height
2D->>2D: Impost rotation
I->>2D: axis
2D->>2D: Vault axis
I->>2D: combined impost length
I->>2D: single impost length
2D->>2D: Get impost length

2D->>3D: Impost segments

deactivate 2D
activate 3D
3D->>3D: Vertical planes
I->>3D: overhang
I->>3D: vault height
3D->>3D: Overhang point creation
3D->>3D: Definition of vault base
I->>3D: ashlars number
3D->>3D: ashlars number definition
I->>3D: Vault thickness
3D->>3D: Vault thickness
deactivate I
3D->>RT: Geometry

deactivate 3D
activate RT

RT->>RT: ashlars planes

RT->>O: intrados single surfaces
activate O
RT->>O: extrados single surfaces
RT->>O: superfici singole fronte
RT->>O: ashlar plane surfaces
deactivate O
deactivate RT

```

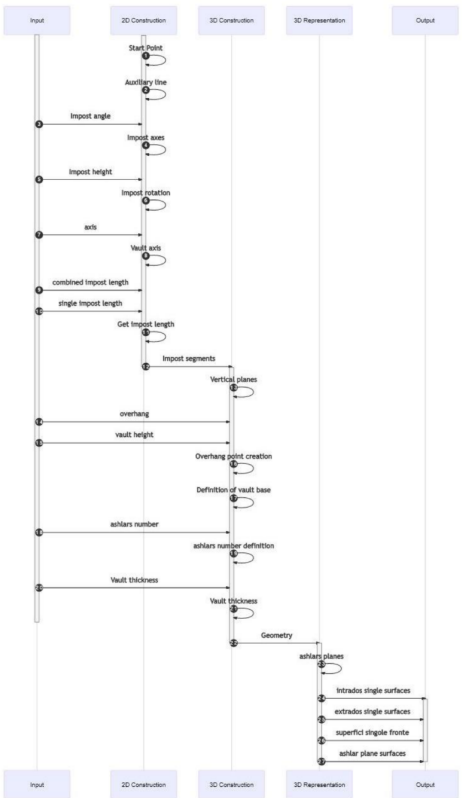


Fig. 3. Documentazione dello *script* della volta a trompe tramite linguaggio Markdown generato attraverso lo strumento Marmaid e rappresentazione tramite diagramma UML dell'algoritmo.

La documentazione presentata è stata realizzata utilizzando due differenti implementazioni di Markdown, Mermaid e *js-sequence-diagrams* (figg. 3, 4). È possibile notare l'universalità di linguaggio che accomuna quasi totalmente i due strumenti: gli elementi simbolici indicanti le relazioni tra i differenti attori, ad esempio, restano costanti nei due *plug-in*. L'informazione risulta facilmente trasmissibile non solo tra piattaforme diverse, tramite l'utilizzo del testo, ma anche tra *editor* differenti mediante una minima variazione di lessico.
I diagrammi così realizzati possono essere facilmente compresi da utenti esterni, anche se privi di conoscenze di programmazione e di costruzione di script. Essi possono intervenire nel processo, individuandone le criticità e proponendo delle modifiche a esso. I diagrammi presentati riflettono l'oggetto della programmazione: nelle colonne si individuano i 'soggetti tematici' che, assemblati, creano il risultato finale (Input, 2D construction, 3D construction ecc.). L'utente può pertanto seguire il flusso di dati per leggere la procedura svolta.

```sequence {theme="simple"}
participant Input as I
participant 2D Construction as 2D
participant 3D Construction as 3D
participant 3D Representation as RT
participant Output as O

2D->>2D: Start Point
2D->>2D: Auxiliary line
I->>2D: Impost angle
2D->>2D: Impost axes
I->>2D: Impost height
2D->>2D: Impost rotation
I->>2D: axis
2D->>2D: Vault axis
I->>2D: combined impost length
I->>2D: single impost length
2D->>2D: Get impost length

2D->>3D: Impost segments

3D->>3D: Vertical planes
I->>3D: overhang
I->>3D: vault height
3D->>3D: Overhang point creation
3D->>3D: Definition of vault base
I->>3D: ashlars number
3D->>3D: ashlars number definition
I->>3D: Vault thickness
3D->>3D: Vault thickness

3D->>RT: Geometry

RT->>RT: ashlars planes

RT->>O: intrados single surfaces
RT->>O: extrados single surfaces
RT->>O: superfici singole fronte
RT->>O: ashlar plane surfaces
```

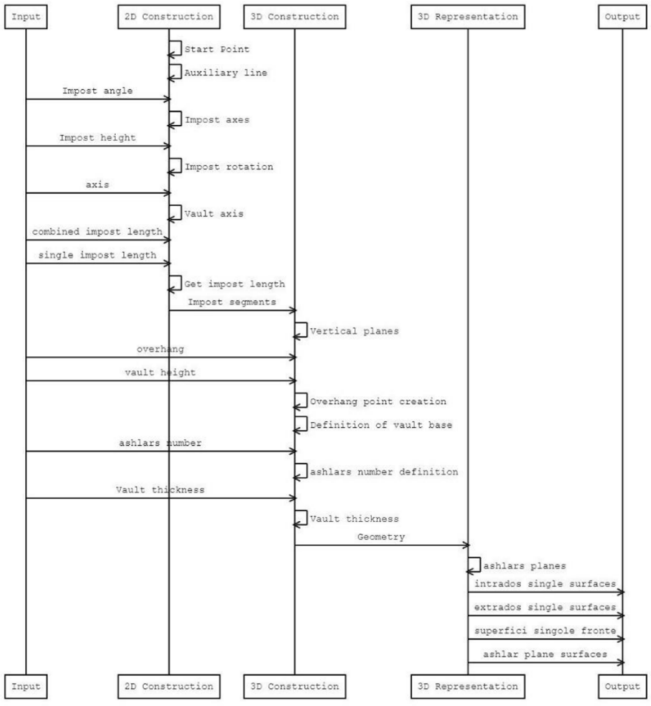


Fig. 4. Documentazione dello script della volta trompe tramite linguaggio Markdown generato attraverso lo strumento js-sequence-diagrams e rappresentazione tramite diagramma UML dell'algoritmo.

Conclusioni

La scienza della rappresentazione è da sempre legata al concetto di regola, intesa come l'istruzione di una procedura predefinita per raggiungere uno scopo. Allo stesso modo, ogni codice di programmazione rappresenta un insieme di regole atte a definire una funzione specifica in modo universale e astratto.
Gli strumenti informatici di disegno e modellazione 3D hanno messo a disposizione degli utenti ambienti di sviluppo per la scrittura di regole atte ad automatizzare il disegno secondo specifici linguaggi. È noto come l'introduzione del Visual Programming Language ha reso la programmazione, in questo senso, una pratica molto diffusa. Grazie a questi strumenti, la rappresentazione, veicolata attraverso *scripting*, diventa il progetto di un modello tridimensionale o di un processo, grazie a specifici attributi iniziali e prodotti finali.
Il principio di generalità e astrazione di pratiche geometriche è presente nei trattati di geometria e rappresentazione per utilizzi specifici, quali ad esempio i trattati di stereotomia. Il presente studio applica i linguaggi di programmazione in modo da garantire la produzione di modelli stereotomici tridimensionali, in modo da abilitare utilizzi del modello che vada-

no oltre la rappresentazione, come nel caso della verifica strutturale. Lo *script*, costruito attraverso una piattaforma VPL, che per definizione utilizza la rappresentazione per essere comprensibile a un pubblico più ampio, genera quindi una molteplicità di rappresentazioni, non solo nel risultato finale ma anche in tutti i risultati intermedi. Si definisce così il linguaggio come mezzo di comunicazione tra usi differenti della rappresentazione.
In un sistema di programmazione visuale intervengono molteplici linguaggi, tra cui quelli testuali atti a generare le singole funzioni. Ogni ambiente di questo tipo si configura quindi come un terreno comune della rappresentazione. Ricavare le regole di organizzazione per aumentare la leggibilità e la modificabilità degli *script* è una necessità e un'occasione di ricerca. Il presente contributo mostra la costruzione di diagrammi standard UML, specifici per questi scopi. Si tratta di generare una seconda forma di rappresentazione del modello 3D, secondo una sequenza logica di operazioni.

Note

[1] Markdown è un linguaggio di *markup* ampiamente adottato per la trasmissione di informazioni mediante la conversione di testo in molteplici formati, tra cui l'HTML per la documentazione Web. Markdown è stato attualmente implementato aggiungendo numerose funzioni, tra le quali la produzione di grafici UML da testo.

Riferimenti bibliografici

Becchi A., Foce F. (2002). *Degli archi e delle volte*. Venezia: Marsilio.

Benvenuto E. (1991). *An introduction to the history of structural mechanics : vaulted structures and elastic systems*. Berlino: Springer.

Block P. (2009). *Thrust Network Analysis: Exploring Three-dimensional Equilibrium*. Doctoral Thesis. Boston: MIT

Block P., Lachauer L., Rippmann M. (2014). Thrust network analysis: Design of a cut-stone masonry vault. In *Shell Structures for Architecture: Form Finding and Optimization* (vol. 9781315849), pp. 71-88.

Calvo-López J. (2020). *Stereotomy*. Berlin: Springer.

Carpo M. (2003). Drawing with numbers: Geometry and numeracy in early modern architectural design. In *Journal of the Society of Architectural Historians*, vol. 62, Issue 4, pp. 448-469. Los Angeles: University of California Press.

Cocchiarella L. (a cura di). (2015). *The Visual Language of Technique Volume 1 - History and Epistemology*. Berlino: Springer.

Davis D. (2016). Evaluating buildings with computation and machine learning. In K. Velikov et al. (a cura di). *ACADIA 2016: Posthuman Frontiers: Data, Designers, and Cognitive Machines*. Proceedings of the 36th Annual Conference of the Association for Computer Aided Design in Architecture, University of Michigan Taubman College of Architecture and Urban Planning, pp. 116-123.

Davis D., Peters B. (2013). Design ecosystems: Customising the architectural design environment with software plug-ins. In *Architectural Design*, 83(2), pp. 124-131.

Heyman J. (1998). *Structural analysis. A historical approach*. Cambridge: Cambridge University Press.

Huerta S. (2008). The analysis of masonry architecture: A historical approach: To the memory of professor Henry J. Cowan. In *Architectural Science Review*, 51 (4), pp. 297-328.

Kurrer K. E., Kühn B. (2009). The history of the theory of structures. from arch analysis to computational mechanics. In *European Journal of Environmental and Civil Engineering*, 13 (3), pp. 369-370.

Ochsendorf J. (2002). *Collapse of Masonry Structures*. Cambrige (MA): University of Cambridge.

Rippmann M., Lachauer L., Block P. (2012). Interactive vault design. In *International Journal of Space Structures*, 27 (4), pp. 219-230.

Trevisan C. (2011). *Per la storia della stereotomia. geometrie, metodi e costruzioni*. Roma: Aracne.

Autori

Paolo Borin, Università di Padova, paolo.borin@unipd.it
Devid Campagnolo, Università di Padova, david.campagnolo@unipd.it
Alberto Longhin, Università di Padova, alberto.longhin@unipd.it

Per citare questo capitolo: Borin Paolo, Campagnolo Devid, Longhin Alberto (2021). Testo, modello, diagramma: continuità e aggiornamento dei linguaggi per la rappresentazione/Text, model, diagram: representation as a changing language. In Arena A., Arena M., Mediati D., Raffa P. (a cura di). *Connettere. Un disegno per annodare e tessere. Linguaggi Distanze Tecnologie. Atti del 42° Convegno Internazionale dei Docenti delle Discipline della Rappresentazione/Connecting. Drawing for weaving relationship. Languages Distances Technologies. Proceedings of the 42th International Conference of Representation Disciplines Teachers*. Milano: FrancoAngeli, pp. 245-260.

Isbn 9788835125891

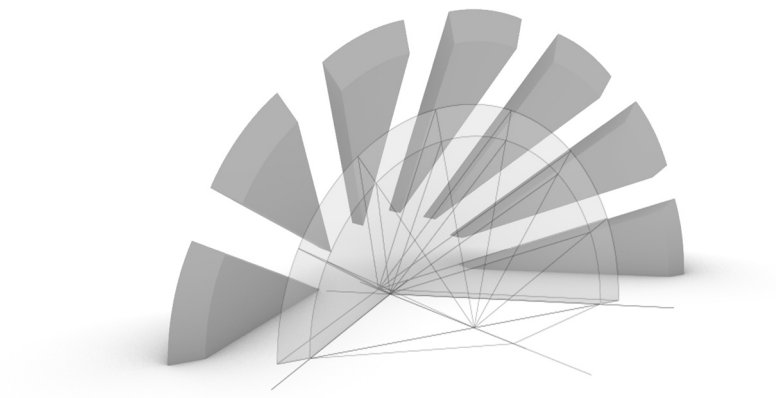
Text, Model, Diagram: Representation as a Changing Language

Paolo Borin
Devid Campagnolo
Alberto Longhin

Abstract

The production of scripts and algorithms has now become common practice to form the architectural shape and the information management of BIM models. The representation has to be compared with the necessary updating towards computational logics through different languages, which know how to build and operate three-dimensional models. This work first verifies the application of scripting rules to conical trompe vaults, as described in the stereotomic treatises. By doing so, the resulting model generates multiple representations, which can enable specific uses, such as the structural verification of elements. Moreover, visual programming often generates scripts that are difficult to understand, in the absence of practices and standards for the definition and organization of nodes. Hence it derives the need to document the logic used, in order to modify the script and to be understood by readers and external interlocutors. A second example thus demonstrates the application of automated production languages for UML process diagrams, starting from suitably written code that can be integrated into VPL platforms. The parametric representation of the first case is abstracted in a further diagram, using textual code. The two cases, placed as a system, investigate the relationship between different languages in the context of representation.

Keywords
scripting, tresatises, visual programming language, diagrams, UML.



Digital representation of the *trompe* vault.

doi.org/10.3280/oa-693.14

Introduction

Since the 1960s, the birth of design codes through computer-aided design (CAD) has opened a new era for architectural representation. However, in the last decade the specification of term computational has guaranteed a vast interest in the research field. By computational we mean the generation of signs –in the broadest sense of texts, models, processes– obtained thanks to scripting procedures and algorithms. These are structured sequences of operations, within a mathematical type, which are based on parameters with the aim of solving a certain class of problems through a systematic and universal approach.
Although this practice is today often described as a revolutionary method, its use as a sequential and generalized approach derives from well-known examples in the history of architecture and representation. Leon Battista Alberti, in the seventh book of *De Re Aedificatoria*, proposes a series of standardized instructions to calculate the proportions of the Doric base of a column [Carpo 2003]. Another evidence of an antecedent algorithmic approach are the applications for the realization of the stereotomic blocks of arches and vaults. Through technical representations drawn on sites (*épures*), or on paper (*traits*) and treatises, architects and engineers have exchanged specific instructions to maçons for the construction of potentially replicable structures *in loco* for three centuries. Both examples demonstrate the necessary connection between textual and graphical notation, which produces, or generates, built applications.
Nowadays the computational aspects are conveyed through scripting procedures, both in formal research and in the information management of projects. The most common methodology for this phase is offered by visual programming tools (VPL, Visual Programming Language), designed to produce functions suitable for specific tasks, graphically through a two-dimensional representation of nodes (functions) and lines (relations). The uses vary from the generation of the form to the information management of BIM projects.
The current status quo leads to reflect on the language, concerning the representation, and how this can convey geometries and uses [Cocchiarella, 2015]. To the famed graphic language of representation, we must add the form –the code in machine language– in which an algorithm is expressed. Although tools allow the production of scripts using a visual language, they still contain instructions in one-dimensional programming languages, such as Python, C #, etc. Understanding them has two fundamental consequences. The first is the transformation of the representation into a result subsequent from procedures defined outside the representation itself. An abstraction of the processes –logical and geometric– of the project or parts of it, reducing them leads to a single language that generates models and drawings. Secondly, the information as a key-point of the models enables them for different purposes, which also overcome the design idea, becoming objects of analysis and metadata containers. Through the rational use and adequate structuring of the language with which the relationships are created, the knowledge transmission of the model can be extended by defining physical, mechanical, informative properties or new relationships between different project entities that can mutually interact. In this way it is pursued the overcome of the 'static' limit allowed by the representation of the 'form', enabling the geometry to take on 'dynamic' and performance connotation, instrumental to the subsequent phases of the creative process but also of verification and realization of the project.

From the treatise to the script: connecting geometry, design and simulation

In the present essay, a study of a typical stereotomic vault is addressed: the *trompe* vault (cover). This structure is a conical vault, whose genesis is given by the semi-revolution of a straight line having a fixed endpoint. The half-cone obtained can be sectioned from different surfaces according to the type of configuration.

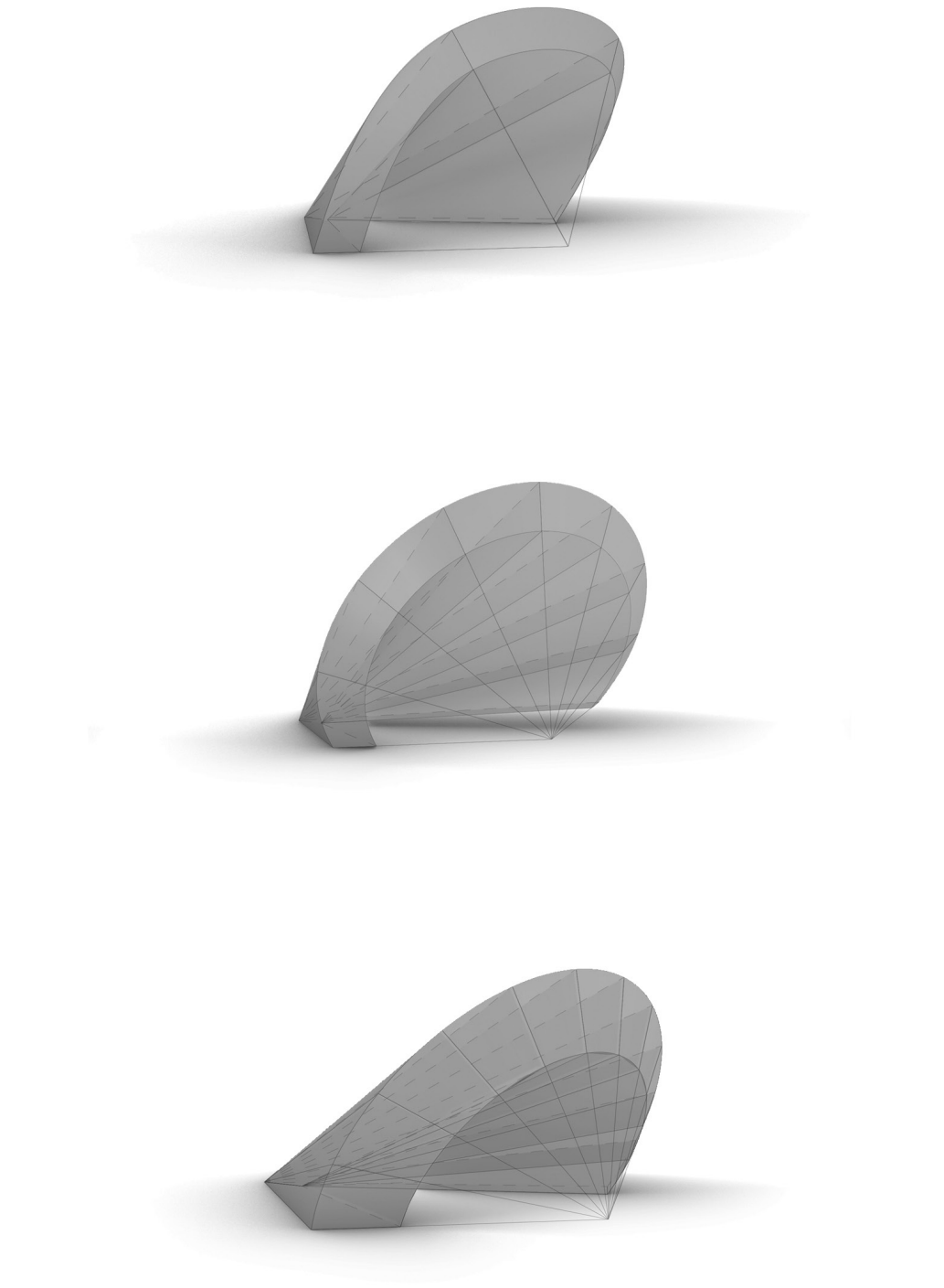


Fig. 1. Grasshopper3d script for building the trompe and examples of resulting configurations.

This solution could be used between two intersecting walls, both outside a building, to expand the space of an internal room, and inside the ecclesiastical architecture in the presence of octagonal drums to be set in the usually square space that is generated at the intersection of the main nave and transept [Calvo-López 2020]. It is a much-investigated typology by European treatises, thanks to its topological peculiarity and structural hazard, so much so that it is defined by De L'Orme *voûte suspendue en l'air* [Trevisan 2011; Calvo-Lopez 2020].
The study dealt with the generalization of the different configurations of the vault through scripting practice using the VPL Grasshopper3d, a plugin of Rhinoceros3D modelling software (fig. 1).
The genesis of the shape starts from a series of parameters that define the position of the generators of the imposts that arise from the vertex of the cone. Two parameters regulate the length of the impost's lines; another one governs the orientation of the directrix line of the *trompe*, with respect to the axis of the cone. A second set of properties is represented by angles: the opening angle of the shutters is structured as sum of two different angles such that they can define an oblique cone. A third set of parameters governs the rotation of the two generatrixes on the vertical plane, allowing the rampant configurations. Finally, in the scripting process is available the definition of the vault's height and thickness, the number of segments (*voussoirs*), and the inclination of the frontal intersection curve. To create the individual segments, we proceed by generating planar surfaces from the construction lines previously described.
The script can create a range of different configurations, all geometrically admissible. However, they do not represent structurally acceptable solutions. The study of the static invariant of a particular configuration, the so-called "*tromp fondamentale*", was therefore tackled using the RhinoVAULT graphical structural calculation tool [Rippmann et al. 2012], enabled for Rhinoceros3D modelling software.

From scripting to text: represent to document knowledge

The documentation of the processes assumes a primary role from the point of view of intensive and routine use of visual programming in design. Among all, one of the potentialities of visual programming in architectural design lies in the ability to generate a coherent result with a set of inputs, in a much shorter time than a manual construction. However, the risk in this practice lies in producing a tool which, due to the lack of flexibility given to the code being created and the difficulty of making changes, is less efficient in terms of time than a traditional one-dimensional programming practice [Davis 2016]. If an upstream forecast of the potential problems that the code will have to manage can affect flexibility, the functionality documentation of the script is crucial for the execution of its changes.
Consequently, documenting a process not only acquires the value of a method of transmission of a procedure, but is also configured as an analysis process: through the documentation, the designer retraces the logical and functional sequence of a script and can identify possible issues and criticalities. Transmitting a procedure is crucial also in cases where the algorithm must be utilized and then modified by different actors over time. The use by different actors –as well as by the creator himself, but for different projects– requires documents that guarantee an easy interpretation, to make the necessary changes to adapt the tool to any potential context.
The main method of VPL documentation is the structuring of the model in the visual programming environment itself. This process contributes significantly to the understanding of the code contained therein [Davis, Peters 2013].
However, in a sharing view it is necessary to supplement the process with documentation in open formats and according to a standardized structure to guarantee universality of access to information. Useful for this purpose is the use of UML graphics that summarize the peculiar operations providing an overall view of the process. In the proposed

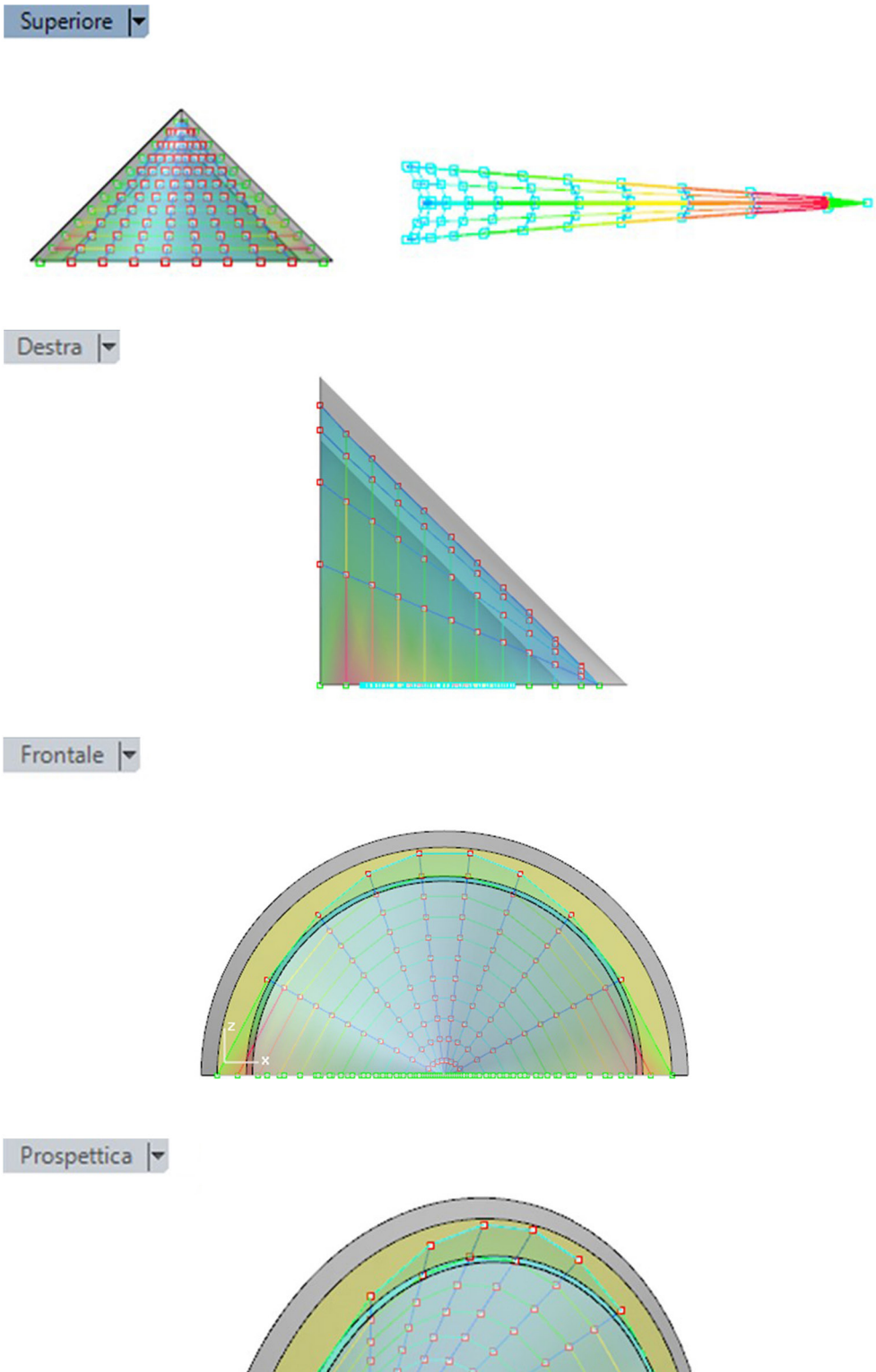


Fig. 2. Final configuration of the G thrust surface within the initial vault and identification of the minimum thickness resulting from the optimization process.

case study, to certify the transmission of knowledge that is independent from the particular use of a proprietary platform or format, a procedural documentation was created using Markdown [1].
The present work tries to document the previous case study through universality criteria, to abstract the result from the forms of representation of the VPL tool. The aim is to provide a visualization of the authors' procedure to create the *trompe vault*, to offer a tool to visualize the knowledge acquired and encoded in the described script.
The documentation presented was created using two different Markdown implementations, Mermaid and js-sequence-diagrams (figs. 3, 4). It is possible to notice the language universality that unites the two tools: for example, the symbolic relationships notation between the different actors remains constant in the two plug-ins. The information is easily transmitted not only between different platforms, via text, but also between different editors by a minimum variation of the vocabulary.
In this way the diagrams can be easily understood by external users, even if they have no programming and scripting background. They can intervene in the process, identifying the critical issues and proposing changes to it. The diagrams reflect the object of the programming: the columns identify the 'thematic subjects' which create the result (Input, 2D construction, 3D construction etc.). Hence the user can follow the data flow to understand the structured process.

Conclusions

The science of representation has always been linked to the concept of rule, assumed as the instruction of a predefined procedure to achieve a specific goal. Likewise, each programming code represents a set of rules designed to define a specific task universally and abstractly.

````
```mermaid

sequenceDiagram
autonumber

participant I as Input
participant 2D as 2D Construction
participant 3D as 3D Construction
participant RT as 3D Representation
participant O as Output

activate I
activate 2D
2D->>2D: Start Point
2D->>2D: Auxiliary line
I->>2D: Impost angle
2D->>2D: Impost axes
I->>2D: Impost height
2D->>2D: Impost rotation
I->>2D: axis
2D->>2D: Vault axis
I->>2D: combined impost length
I->>2D: single impost length
2D->>2D: Get impost length

2D->>3D: Impost segments

deactivate 2D
activate 3D
3D->>3D: Vertical planes
I->>3D: overhang
I->>3D: vault height
3D->>3D: Overhang point creation
3D->>3D: Definition of vault base
I->>3D: ashlars number
3D->>3D: ashlars number definition
I->>3D: Vault thickness
3D->>3D: Vault thickness
deactivate I
3D->>RT: Geometry

deactivate 3D
activate RT

RT->>RT: ashlars planes

RT->>O: intrados single surfaces
activate O
RT->>O: extrados single surfaces
RT->>O: superfici singole fronte
RT->>O: ashlar plane surfaces
deactivate O
deactivate RT

```
````

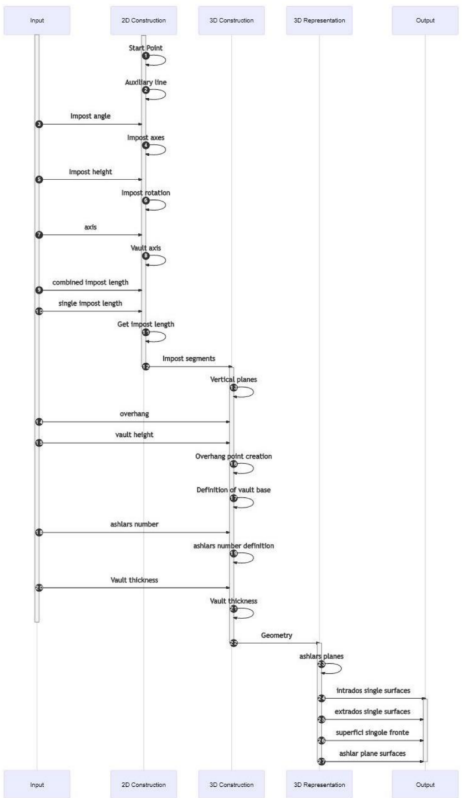


Fig. 3. Documentation of the script for the trompe vault by Markdown language generated through Marmaid tool and UML diagram representation of the algorithm.

The computer 3D design and modeling tools have made accessible environments to users enabling the prospect to write rules to automate the design according to specific languages. It is known how the introduction of the Visual Programming Language has made programming, in this sense, a very widespread practice. Thanks to these tools, the representation becomes the project of a three-dimensional model or a process, thanks to specific initial attributes and final products.
The principle of generality and abstraction of geometric practices is present in geometry and representation treatises for specific purposes, such as stereotomy treatises. This study applies programming languages to ensure the production of three-dimensional stereotomic models, enabling these to uses that go beyond the representation, as in the case of the structural verification. The script, created through a VPL platform, which uses representation to be comprehensible to a wider audience, generates a collection of representations, not only in the result but also in all intermediate results. Thus, the language is defined as a means of communication between different uses of representation.
Multiple languages are involved in a visual programming system, including textual ones that generate the individual functions. Each environment of this type is therefore configured as a common ground of representation. Obtaining the organizational rules to increase the readability and modifiability of scripts is a necessity and a research opportunity. This contribution shows the construction of standard UML diagrams, specific for these purposes. It involves generating a second form of representation of the 3D model, according to a logical sequence of procedures.

```
```sequence {theme="simple"}
participant Input as I
participant 2D Construction as 2D
participant 3D Construction as 3D
participant 3D Representation as RT
participant Output as O


2D->>2D: Start Point
2D->>2D: Auxiliary line
I->>2D: Impost angle
2D->>2D: Impost axes
I->>2D: Impost height
2D->>2D: Impost rotation
I->>2D: axis
2D->>2D: Vault axis
I->>2D: combined impost length
I->>2D: single impost length
2D->>2D: Get impost length

2D->>3D: Impost segments

3D->>3D: Vertical planes
I->>3D: overhang
I->>3D: vault height
3D->>3D: Overhang point creation
3D->>3D: Definition of vault base
I->>3D: ashlars number
3D->>3D: ashlars number definition
I->>3D: Vault thickness
3D->>3D: Vault thickness

3D->>RT: Geometry

RT->>RT: ashlars planes

RT->>O: intrados single surfaces
RT->>O: extrados single surfaces
RT->>O: superfici singole fronte
RT->>O: ashlar plane surfaces

```
```

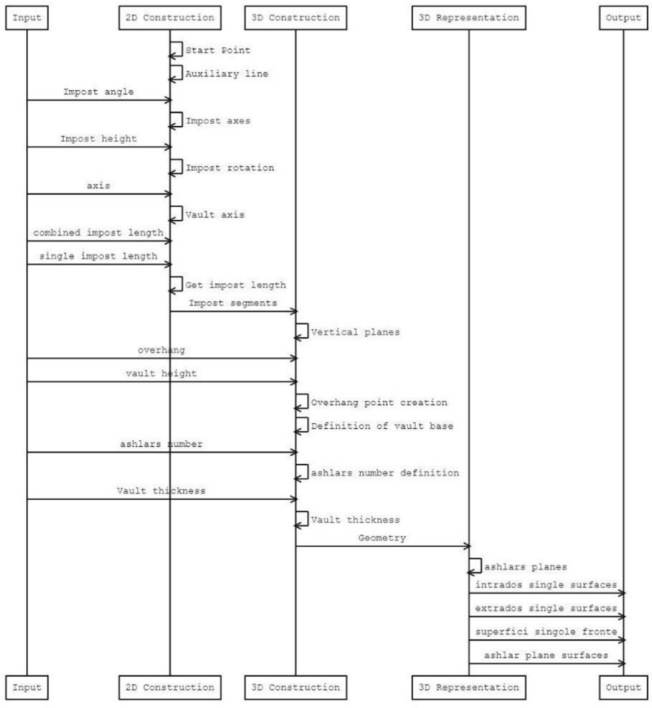


Fig. 4. Documentation of the script for the trompe vault by Markdown language generated through the js-sequence-diagrams tool and UML diagram representation of the algorithm.

Notes

[1] Markdown is a language widely adopted for the information exchange through the conversion in multiple formats (i.e. HTML for Web visualization). Markdown is continuously updated with multiple function, for example the ability to interpret UML diagrams from text.

References

Becchi A., Foce F. (2002). *Degli archi e delle volte*. Venezia: Marsilio.

Benvenuto E. (1991). *An introduction to the history of structural mechanics : vaulted structures and elastic systems*. Berlino: Springer.

Block P. (2009). *Thrust Network Analysis: Exploring Three-dimensional Equilibrium*. Doctoral Thesis. Boston: MIT

Block P., Lachauer L., Rippmann M. (2014). Thrust network analysis: Design of a cut-stone masonry vault. In *Shell Structures for Architecture: Form Finding and Optimization* (vol. 9781315849), pp. 71-78.

Calvo-López J. (2020). *Stereotomy*. Berlin: Springer.

Carpo M. (2003). Drawing with numbers: Geometry and numeracy in early modern architectural design. In *Journal of the Society of Architectural Historians*, vol. 62, Issue 4, pp. 448-469. Los Angeles: University of California Press.

Cocchiarella L. (a cura di). (2015). *The Visual Language of Technique Volume 1 - History and Epistemology*. Berlino: Springer.

Davis D. (2016). Evaluating buildings with computation and machine learning. In K. Velikov et al. (a cura di). *ACADIA 2016: Posthuman Frontiers: Data, Designers, and Cognitive Machines*. Proceedings of the 36th Annual Conference of the Association for Computer Aided Design in Architecture, University of Michigan Taubman College of Architecture and Urban Planning, pp. 116-123.

Davis D., Peters B. (2013). Design ecosystems: Customising the architectural design environment with software plug-ins. In *Architectural Design*, 83(2), pp. 124-131.

Heyman J. (1998). *Structural analysis. A historical approach*. Cambridge: Cambridge University Press.

Huerta S. (2008). The analysis of masonry architecture: A historical approach: To the memory of professor Henry J. Cowan. In *Architectural Science Review*, 51 (4), pp. 297-328.

Kurrer K. E., Kühn B. (2009). The history of the theory of structures. from arch analysis to computational mechanics. In *European Journal of Environmental and Civil Engineering*, 13 (3), pp. 369-370.

Ochsendorf J. (2002). *Collapse of Masonry Structures*. Cambrige (MA): University of Cambridge.

Rippmann M., Lachauer L., Block P. (2012). Interactive vault design. In *International Journal of Space Structures*, 27 (4), pp. 219-230.

Trevisan C. (2011). *Per la storia della stereotomia. geometrie, metodi e costruzioni*. Roma: Aracne.

Authors
Paolo Borin, Sapienza Università di Padova, paolo.borin@unipd.it
Devid Campagnolo, Università di Padova, devid.campagnolo@unipd.it
Alberto Longhin, Università di Padova, alberto.longhin@unipd.it

To cite this chapter: Borin Paolo, Campagnolo Devid, Longhin Alberto (2021). Testo, modello, diagramma: continuità e aggiornamento dei linguaggi per la rappresentazione/Text, model, diagram: representation as a changing language. In In Arena A., Arena M., Mediati D., Raffa P. (a cura di). *Connettere. Un disegno per annodare e tessere. Linguaggi Distanze Tecnologie. Atti del 42° Convegno Internazionale dei Docenti delle Discipline della Rappresentazione/Connecting. Drawing for weaving relationship. Languages Distances Technologies. Proceedings of the 42th International Conference of Representation Disciplines Teachers*. Milano: FrancoAngeli, pp. 245-260.

Isbn 9788835125891