NEUPANE, BABITA, MS. DECEMBER 2020 ARCHITECTURE

EXPLORING FORMS OF MASONRY VAULTS BUILT WITHOUT CENTERING

Thesis Advisor: Dr. Rui Liu

Inspired by the historical construction technique and contemporary form-finding methods, as well as practices in designing and constructing masonry vaults, the thesis explores different forms that can be built without centering during construction. This thesis examines the literature associated with the geometries, construction techniques and methods used to generate masonry arches, vaults, and domes without centering or with minimum supports. The main idea lies on the self-supporting courses where each masonry unit has its own equilibrium condition. Based on the principles of equilibrium, this research develops algorithms for two cases: when bricks are not bonded with mortar and are bonded with mortar. The algorithms to generate an arch when subjected to additional vertical loads for both cases have also been investigated.

Moreover, using the Python component in Grasshopper, a tool is developed using the proposed algorithms in this thesis. The tool is used for parametric designs to create new forms of masonry vaults which could be built without centering.

EXPLORING FORMS OF MASONRY VAULTS BUILT WITHOUT CENTERING

A thesis submitted
to Kent State University in partial
fulfillment of the requirements for the Degree of
Master of Science in
Architecture and Environmental Design

by

Babita Neupane

December 2020

Thesis written by

Babita Neupane

B.Arch., Tribhuvan University, Nepal, 2015

M.S. in Architecture and Environmental Design, December 2020

Approved by

Rui Liu, Ph.D., PE , Advisor

Reid Coffman, MLA, Ph.D. , Coordinator, Master of Science in Architecture and Environmental Design

Mark Mistur, AIA , Dean, College of Architecture and Environmental Design

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Dr. Rui Liu, for guiding my research. I am extremely grateful for his continuous support and encouragement throughout the journey. I want to thank him for pushing me forward during the tough time and his help getting the best out of me. It was a great privilege and honor to work and study under his guidance.

Furthermore, I would like to express my gratitude to my thesis committee, Dr. Elwin C. Robison of CAED at Kent State University, and Prof. Dr. Qindan Huang from the Marquette University. I am thankful for their valuable comments and suggestions on my thesis. Also, I would also like to acknowledge Professor Bill Lucak for helping me learn Python scripting.

I would like to give my special thanks to Ms. Connie Simms and the Writing Commons at the Kent State University. I really appreciate their time and efforts in reviewing my thesis and improving my writing.

Finally, I am thankful to my parents for their love, care, prayers, and sacrifices for teaching me and preparing for my future. I am also grateful to my friends for their encouragement and support, without which this accomplishment would not be possible.

CHAPTER 1: INTRODUCTION AND BACKGROUND

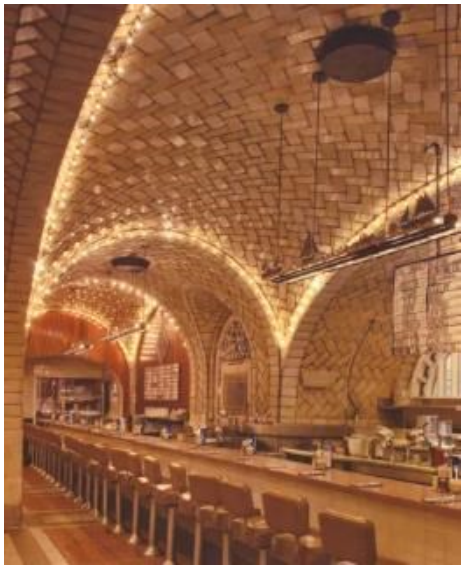
The first chapter provides the overview of the thesis discussing the history and background of masonry structures (arches, vaults, and shells) and present practices of construction. Finally, it outlines the research objectives and assumptions made for the study.

1.1 Introduction

Arches, vaults, and shells have been used historically around the world as early as the 3rd millennium BC in Egypt and countries of the Middle East (Joffroy et al., 1994). The popular masonry materials at that time for building vaults and shell structures were usually stones, bricks, or thin clay tiles. These structures were popular in the history of architecture with large spanning spaces to provide roofing to buildings or as a support for roof or ceiling.

Masonry shell structures are elegant and beautiful. Examples include the magnificent Gothic architectural buildings such as the ribbed vaults of Reims Cathedral in France (Figure 1.1a), and the corbel spans of Spean Praptos of Cambodia. More recent examples include Guastavino vault built in North American in the late 19th and early 20th centuries (Figure 1.1b). Masonry vaults have been used for many years, but are still employed because of the stability gained through their geometrical configuration and durability of the masonry materials. Modern examples include the free-form masonry (brick and stone) shells built by Block Research Group at ETH, Zurich; the Drone port prototype at the Venice Biennale of Architecture 2016 (Foster + Partners); and the roofing of Maya Somaiya Library (Figure 1.2).

(a)



(b)

Figure 1.1: Examples of historical masonry structures (a) Interior of Reims Cathedral, France (image by Iwan Baan www.dezeen.com) (b) Interior of The Oyster bar, New York (Image by Michael Freeman https://www.traditionalbuilding.com/features/book-review-guastavino)

(a)

(b)

(c)

(d)

Figure 1.2: Modern examples of free-form masonry shells (a) Stone vault by Block Research Group at ETH (image by Iwan Baan www.dezeen.com) (b) Drone port prototype at Venice Biennale of Architecture 2016 (image by Jessica Mairs https://www.dezeen.com/2016/05/27/norman-foster-partners-vaulted-drone-port-prototype-medical-supplies-remote-africa-venice-architecture-biennale/ (c) and (d)Maya Somaiya library Kopargaon, India (image by Edmund Sumner https://www.edmundsumner.co.uk/menu.php?catNo=3)

1.2 Background

Historically, some vaults have been constructed without formwork. However, enough information for the actual erection of the masonry vaults is rarely available because there are no early records. Wendland (2007) undertook research which examined the historic documents found so far and discovered that the basic principle for constructing the masonry vaults without the temporary support relies on the self-supporting courses which are stable on themselves by forming the arch shape (Wendland, 2007; Fitchen,1981). There are different techniques for constructing masonry vaults which do not even require a minimum guideline during construction, like the leaning bricks technique and the corbelled vault construction technique. The leaning brick technique was first introduced in Egypt and northern Sudan when brick architecture was flourishing. The technique was revived by an Egyptian architect Hassan Fathy and implemented in his works; a Craft School and Agriculture Cooperative Center at New Baris Egypt, and Cultural Center of Garagus (Serageldin, I. 2007; Miles, M., 2006). In this technique, the first course is laid leaning against the wall or a previously constructed arch using mortar. Other courses are laid one after another leaning against the previous one. The vaults are built in the shape of catenary to reduce the risk of buckling due to the longer courses. This type of vaulting does not require any temporary support during assembly (Gargiulo et al., 2006; Ponce et al., 2004; Van Beek, G., 1987).

In the corbelled vault technique, two sets of overhanging bricks are projected from both sides which can meet at the apex to form a corbel arch. The corbel arch is also called false arch because all the stresses of the structure due to weight of the superstructures are not transformed into compressive stresses. A corbel vault is formed by the extrusion of corbel arches. This type of technique has been used since early architecture history to span entrance gates and vaults. For

instance, vaults in the Maya civilization; buildings of Islamic architecture in Persia and central Asia; and vaults in chamber of Ancient Egyptian Pyramids used the corbel technique shown in Figure 1.3 (Marr et al.,1986; DeLaine, J, 1990).

(a)

(b)

(c)

Figure 1.3: Examples of corbel arches (a)Tomb of Sultan Ghari 1231 AD, India (Image by Waren Apel https://commons.wikimedia.org/wiki/File:Trabeate_Arch_in_New_Delhi_India.jpg) (b) Great Gate at Labna, southern Yucatán (Image source: https://www.historymuseum.ca/cmc/ exhibitions/civil/ maya/mmc02eng.html) (c) Corbel arch at Cahal Pech (Image source: https://www.wikiwand.com/en/Corbel_arch#/Maya_civilization)

Both techniques allowed the early builders to construct arches/vaults without temporary support. The main principle behind that is the self-supporting course in which each unit is in its own equilibrium condition. Therefore, this research aims to explore the possibilities of generating new forms using the idea of self-supporting courses.

1.3 Research Need

Brick is a traditional construction material available in standard sizes as well as in customized sizes, and it does not require any dressing like stone before the construction. Brick masonry structures can be beautiful and strong and have advantages over concrete structures in terms of cost, complexity in construction, and in terms of carbon gas emissions (Danjuma, 2013).

With the recent development in the field of building technology, it has become possible to design free-form shapes for the masonry structures. Design tools like RhinoVAULT help to analyze the force pattern and determine the structural geometry of the shell. The Block Research Group at ETH Zurich has done a tremendous job regarding the form-finding of free-form masonry shells (Block et al., 2007; Rippmann et al., 2013; Adriaenssens et al., 2014). On their projects, centering was extensively used to support the structure during construction, because the force distribution is not uniform in free-form shells and temporary support is required during construction. The installation of centering requires skilled masons. Furthermore, the decentering after the completion is a complicated task which requires a careful consideration. Thus, the construction of the structure becomes uneconomical in terms of the high costs caused by the labor-intensive and dense temporary supports required (Wendland, 2007; Danjuma, 2013).

Researchers have sought to find the alternatives to centering like the method proposed in Davis et al. (2011), using formwork made of cardboard to support the structure until it is fully completed. Deuss et al. (2014) proposed the alternative of using the cables to assemble the units during construction but has a limitation for large scale projects as it becomes uneconomical.

Why build without centering?

Skilled laborers are required during construction of vaults and the installation and decentering of supports. So, building vaults without any temporary supports helps to reduce the number of masons required.

Building without centering saves the total time and cost of a project. Structures can be built faster because the complicated installation and removal of centering is not needed.

Brick masonry shells have less greenhouse gas emission than concrete shells. The CO_2 emission during the production of cement and concrete accounts for 6%-7% of global emissions. In the United States, the percentage of the waste from concrete products is 50% of the total construction waste produced annually (Dahmen et al., 2012). However, brick has an advantage of being reused after careful dismantling from the structure.

Therefore, this research aims on exploring possibilities of finding new forms of brick masonry vaults by utilizing the historical technique (obtaining equilibrium condition for each brick) of construction which requires no centering. This research also discusses the possibility of automation in construction of the masonry vaults by providing the algorithm for robot arms to construct brick vaults.

1.4 Research Objectives

The objectives of the thesis include:

- To understand the construction techniques of the different vaults which require no or minimal centering during the construction
- To reveal the techniques that have made it possible to construct without centering
- To implement those techniques using digital tools like Rhinoceros 3D software and Grasshopper/ Python to generate additional new forms of vault
- To create algorithms for parametric designs to generate new vault forms that require no centering to build.

1.5 Assumptions

The thesis focusses on the form-finding of vaults based on the principle of equilibrium. There are several assumptions made to limit the boundary of the research and are listed below.

- The research defines masonry units as a brick of size: 92mm × 57mm × 203mm (3-$\frac{5}{8}$" ×2-$\frac{1}{4}$" × 8").
- It considers unreinforced masonry vaults only.
- It considers loads in the vertical direction only.

1.6 Structure of the Thesis

This thesis comprises of six chapters which are discussed here briefly.

Chapter 1 provides insight on the focus of the thesis research. It presents introduction and the historical background of masonry arches, vaults, and shells. It then describes the need for this research, the objectives, and assumptions made for the research.

Chapter 2 is a review of the literature required to be studied to gain the knowledge about the masonry materials and the construction. It also discusses the geometries of masonry vaults and shell construction methods in historical context and present-day practices. Furthermore, it also describes the past and present methods applied to generate the forms.

Chapter 3 discusses the two-dimensional equilibrium approach for arches built without centering. It discusses the mathematical derivation of overhanging lengths of each brick stacked one above the other for both cases. Two cases discussed in this chapter are bricks with no bonding and bricks with bonding. This chapter also explores the mathematical approaches to the situation in which the vertical load is induced on the top of the arch.

Chapter 4 describes how Python is used to generate arches following the algorithm developed in Chapter 3. It provides discussion on how the Python code works. It also analyzes arches of both

cases described in Chapter 3. Finally, it also discusses the stability of structures during construction and the safety factor.

Chapter 5 presents the vaults designed from the arch generated in Chapter 4. It describes the Python code for each of the forms generated. This chapter also discusses the possibility of using arches developed from this research as a guideline to create masonry vaults.

Chapter 6 summarizes the research and provides recommendations for future research.

CHAPTER 2: LITERATURE REVIEW

This chapter provides a comprehensive literature review regarding masonry construction and materials, construction techniques of masonry vaults/shells, geometries of masonry vaults, and ancient and modern design tools for generating forms.

2.1 Masonry materials and construction

Masonry is strong in compression and weak in tension. Masonry structures are comprised of the small units which are laid in courses and bound together with mortar so that the system acts as a single unit. Commonly used masonry units are stone blocks, clay bricks, concrete blocks, clay tiles and so on. The units can be solid, hollow, or perforated. The masonry unit used in this research is clay brick.

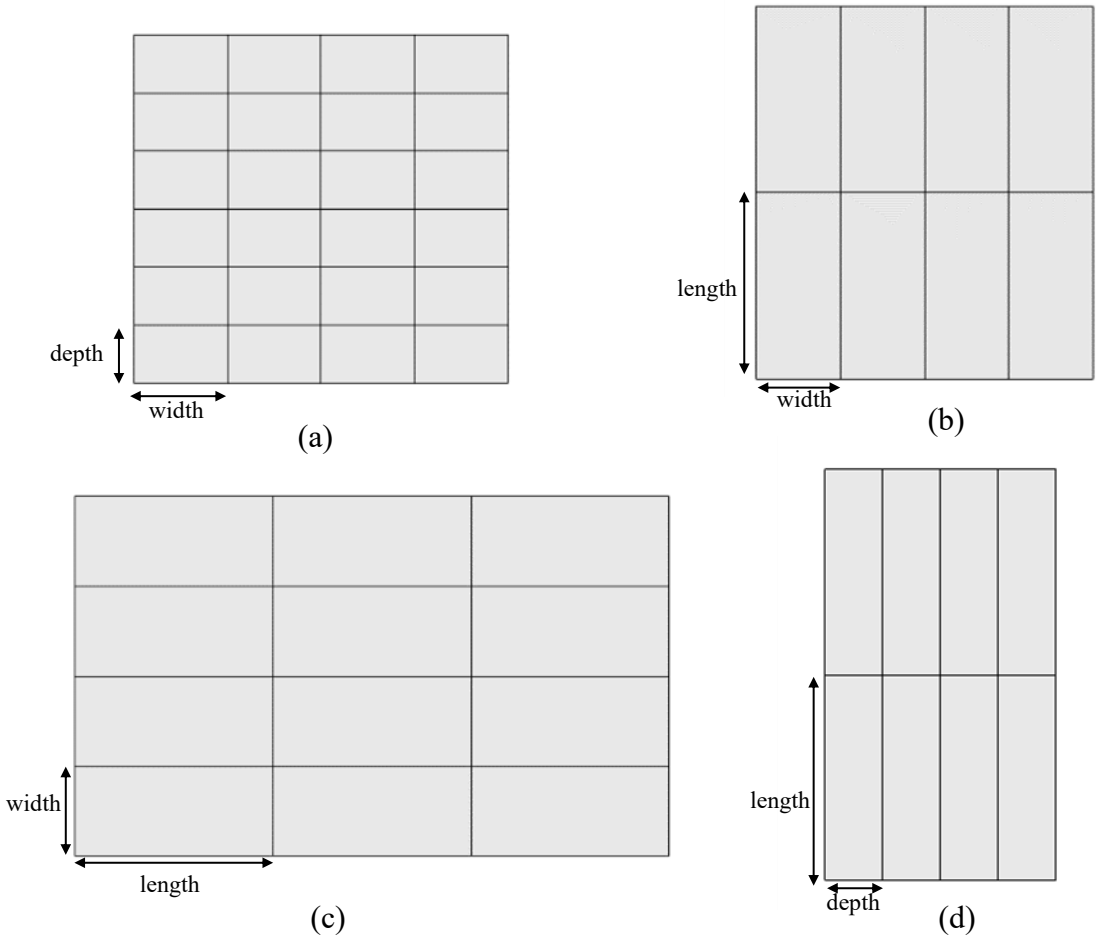
2.1.1 Clay bricks

Brick has been used as a building material since 7500BC and became the principle unit of construction for building vaults and shells. Brick gained its popularity as a building material because it could be produced using locally available material with minimal skills.

The strength of brick depends on the method of manufacture, raw materials used, and the degree of firing. The compressive strength of a fire-burnt clay brick is 19.6 to 24.5 MPa but it depends on how well the bricks have been fired. The compressive strength of poorly fired clay brick can be as lower as 4.9 MPa. For the compressive strength test, a brick is placed in a testing machine and the compression load is applied. After performing at least three tests, an average

compressive strength of the brick is noted. The tensile strength is usually $1/10^{th}$ of the compressive strength (Como et al., 2013).

Masonry units can be arranged in different orientations. Figure 2.1 below shows the different orientations of the masonry units. Among them, the stretcher orientation is the most common.



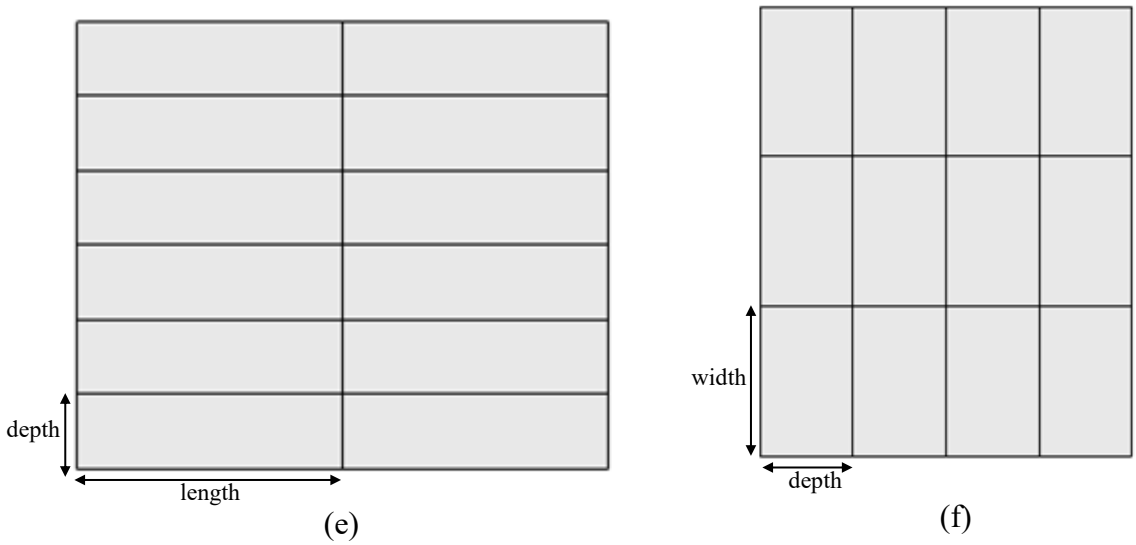
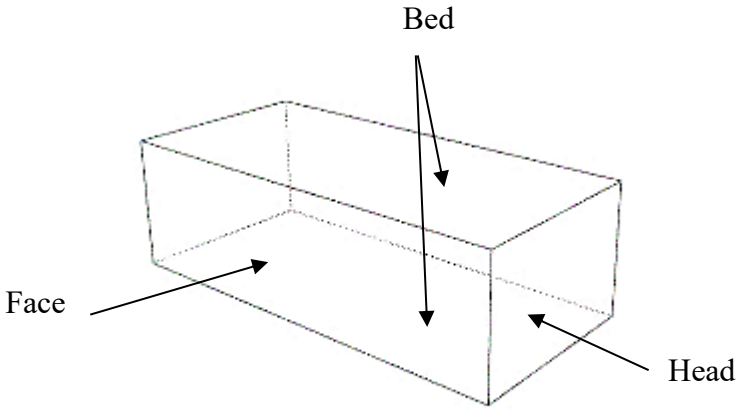
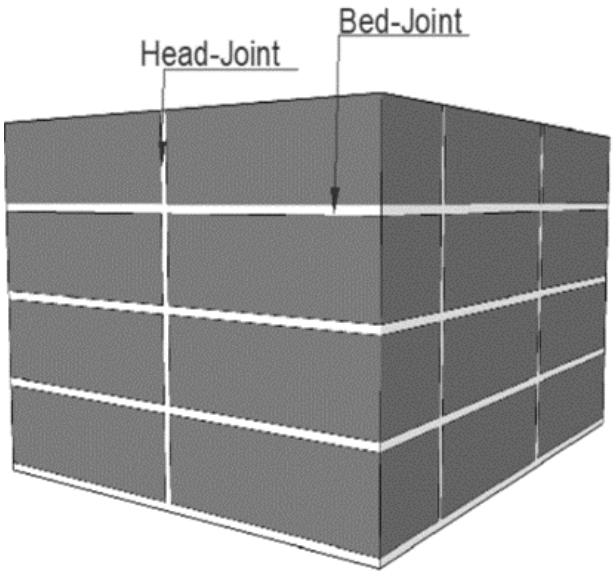


Figure 2.1: Different orientations of masonry units (Hendry,2001). (a)Header (b)Sailor (c) Shiner (d)Soldier (e) Stretcher (f) Rowlock

There are different terminologies used in masonry structures. Figure 2.2 shows some of the terminologies used.



(a)

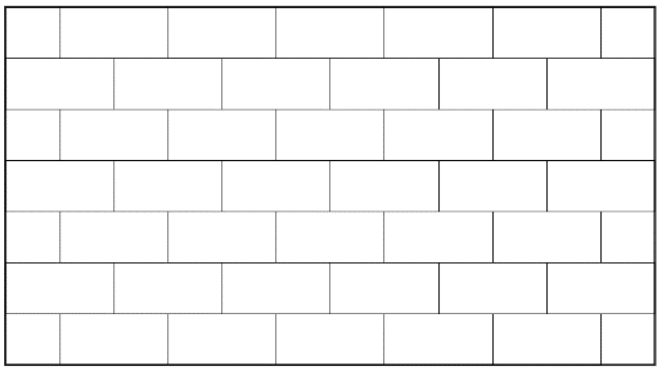


(b)

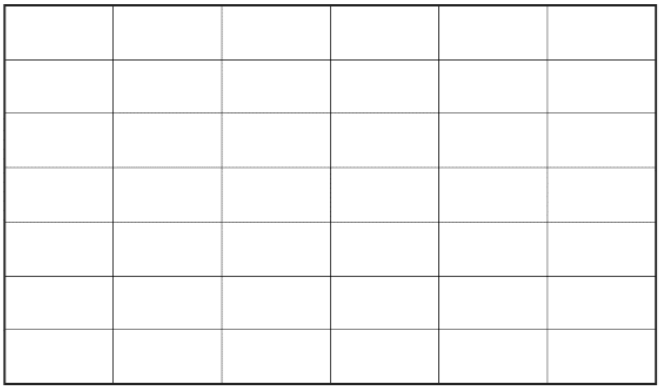
Figure 2.2: Terminologies (a) Terminologies for sides of a masonry unit (b) Terminologies for joints within masonry units in masonry construction (Hendry,2001)

2.1.2 Bond patterns

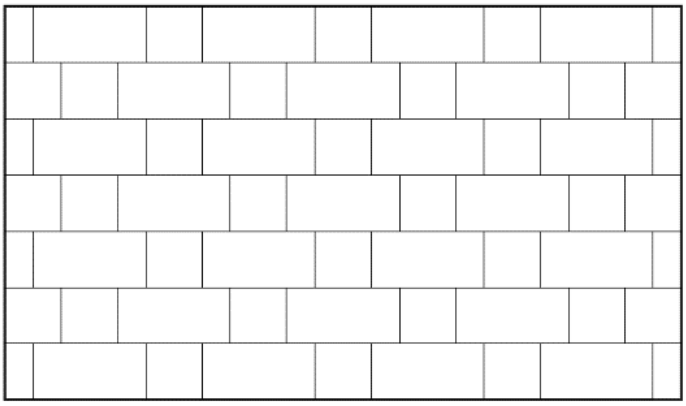
For the construction of a masonry structure, the masonry units can be arranged into different bond patterns. Rajabzadeh et al. (2017), provides an overview on the patterns of masonry units in historical masonry building. The commonly used brick bond-patterns are running bond, stack bond, herringbone bond, Flemish bond, and basket-weave bond which are shown in Figure 2.3 below.



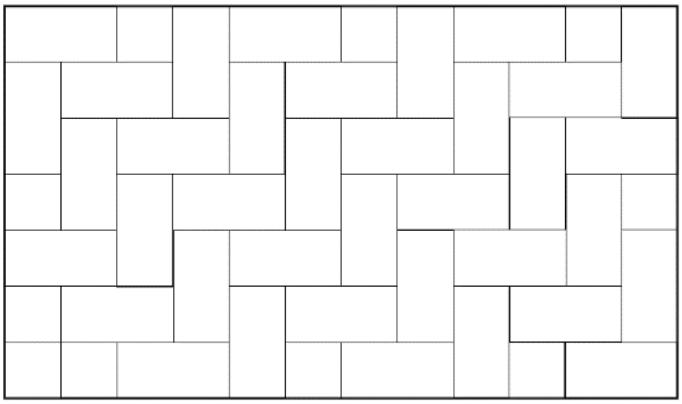
(a) Running bond



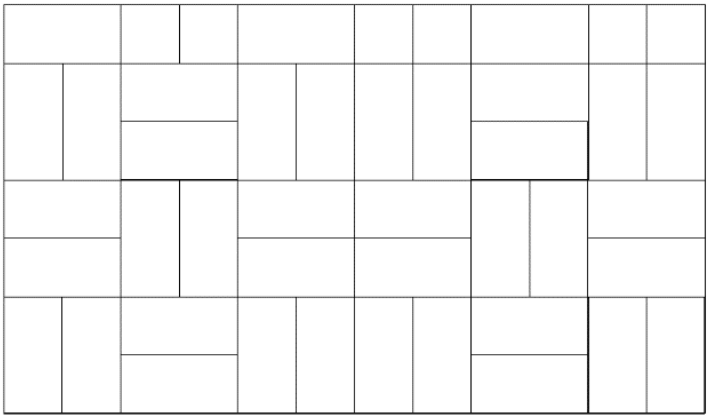
(b) Stack bond



(c) Flemish bond



(d) Herringbone bond



(e) Basket-weave bond

Figure 2.3: Different bond patterns of masonry units (Hendry, 2001)

2.1.3 Mortar

Mortar is the material used to bond the masonry units together so that the system acts as a homogenous unit. The role of mortar in masonry construction is to provide the strength to the bond and to fill the joints between the masonry units. The mortar mixture to be used in masonry construction should follow the specific requirements of ASTM C 270. Moreover, all ingredients in mortar mixture must meet the requirements of respective ASTM specifications.

ASTM C 270 has defined three types of cementitious systems shown below:

- cement-lime mortar

- masonry-cement mortar
- mortar-cement mortar

Cement-lime mortar is the mortar made by mixing cement, lime, and sand in proportion with water. The cement in this mixture provides strength and increases setting time whereas lime and sand provide workability.

Masonry-cement mortar is produced by mixing sand and masonry-cement. Masonry-cements are comprised of different ingredients like Portland cement which increases the setting time, plasticizers which increase workability, and air entraining additives which increase durability. This type of mortar provides good compressive and bonding strength.

Mortar-cement mortar was first developed in 1991 (International Masonry Institute's 'mortar for masonry', 1997). The purpose of developing mortar-cement mortar is to meet the requirement of high flexural bond strength which is higher than that of masonry-cement mortar. The ingredients of this mortar type are the same as that of masonry-cement mortar. The only difference is that the ingredients are optimized so that lower air contents are produced than masonry-cement mortar.

Table 2.1: Mortar type and air contents (ASTM C 270)

	Mortar type		Maximum air content (%)
1	Cement-Lime mortar	M	12
		S	12
		N	14
		O	14
2	Masonry cement mortar	M	18
		S	18
		N	20
		O	20
3	Mortar cement mortar	M	12
		S	12
		N	14
		O	14

The strength and durability of the mortar typically depend on the proportions of the mixture. Based on the proportion of the cement, mortar is characterized as different types:

- Type M: high tensile and compressive strength
- Type S: moderate tensile and compressive strength
- Type N: low tensile and compressive strength

- Type O: very low tensile and compressive strength

The compressive strength of each mortar type is listed in the Table 2.2.

Table 2.2: The compressive strength according to ASTM 270 for each type

	Mortar type	Compressive strength (psi)	Compressive strength (MPa)
1	M	2500	17.23
2	S	1800	12.4
3	N	750	5.17
4	O	350	2.41

Among the types discussed above, Type S is the all-purpose mortar (Tanner et al., 2017; Hendry, 2001). The detailed discussions about bond strength of mortar and factors affecting the strength can be found in (Hogberg, 1967; Palmer et al., 1931; Palmer et al., 1934). The tensile bonding strengths of different mortars investigated by Hetherington (2015) are summarized in Table 2.3.

Table 2.3: The tensile bonding strengths of different mortars (Hetherington, 2015)

Type of mortar	Average bond strength (MPa)	Average tensile bond strength(psi)
1:8 Lime /Sand	0.045	6.52
1:8 Cement /Sand	0.081	11.75
1:6 Lime /Sand	0.051	7.40
1:6 Cement /Sand	0.095	13.78
1:3 Lime /Sand	0.068	9.86
1:3 Cement /Sand	0.117	16.96

Different types of mortars with varying mix proportions were made by the researchers at Sheffield Hallam University. The focus of the research was to compare between their properties and characteristics. The bonding strength tests were carried by using a SHU apparatus developed at Sheffield Hallam University. The research was concluded with the finding that the cement mortar is better than hydraulic lime mortar while taking bond strength in consideration.

2.2 Geometries of masonry arches, vaults, domes, freeform shells

This section of describes different geometries of masonry structures. The geometries of vaults/shells in historical buildings are simple but elegant. They are either based on revolution of the surfaces or formed by the intersection between those surfaces. Some of the geometries are discussed below.

2.2.1 Arches

An arch (Figure 2.4) is defined by Merriam Webster as *"a typically curved structural member spanning an opening and serving as a support (as for the wall or other weight above the opening)."*

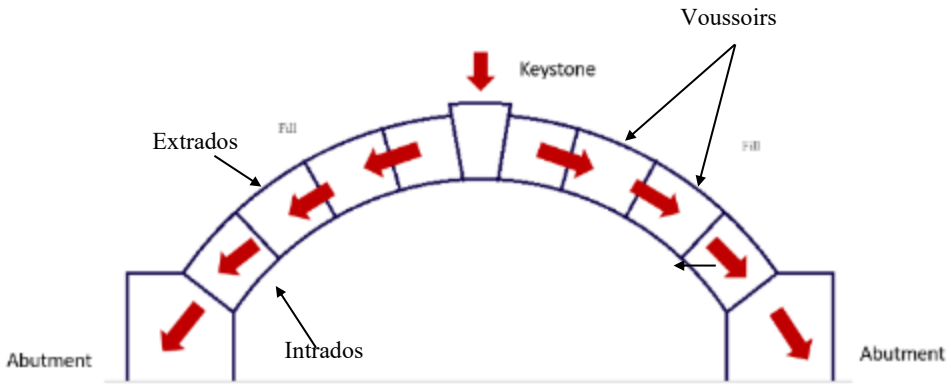


Figure 2.4: Terminologies for an arch

The inner surface of an arch is intrados and outer surface is extrados. The keystone (a final block while building an arch) is a structural part that holds the system in its stable position. The stability of an arch by itself does not depend on the strength of the mortar but on the assemblage of masonry units. The masonry units are placed together so that the internal forces acting on the overall arch system flows to the abutment or the support. Figure 2.4 shows the terminologies for the components of an arch and the flow of internal forces (Heyman, J. 1982). Figure 2.5 presents different types of arches commonly found. The shape of the arch is defined by the shape of the intrados (Joffroy et al., 1994).

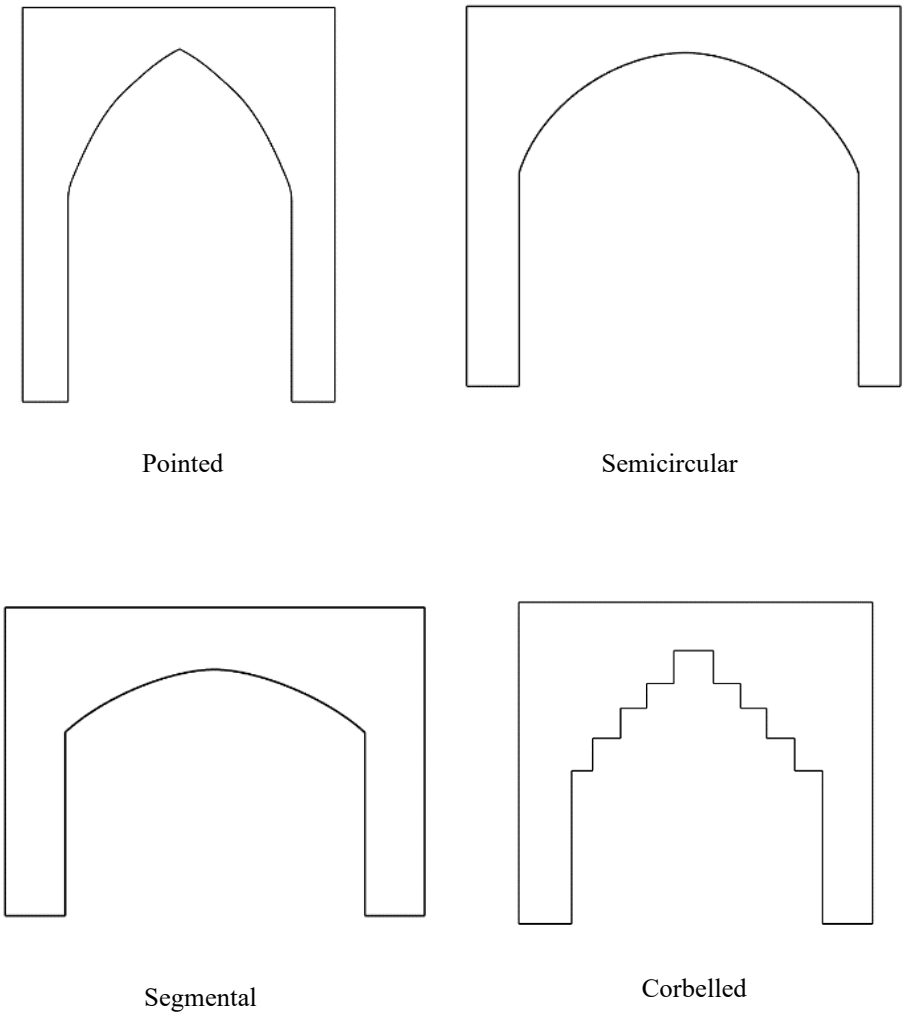


Figure 2.5: Different types of arches (Introduction to Arches, vaults and dome http://www.earth-auroville.com/)

2.2.2 Vaults

Vaults are formed when an arch is extruded along the axis (generally right-angled to the span of the arch). Figure 2.6 above shows some examples of vaults formed by the extrusion of pointed, semicircular, catenary, segmental arches.

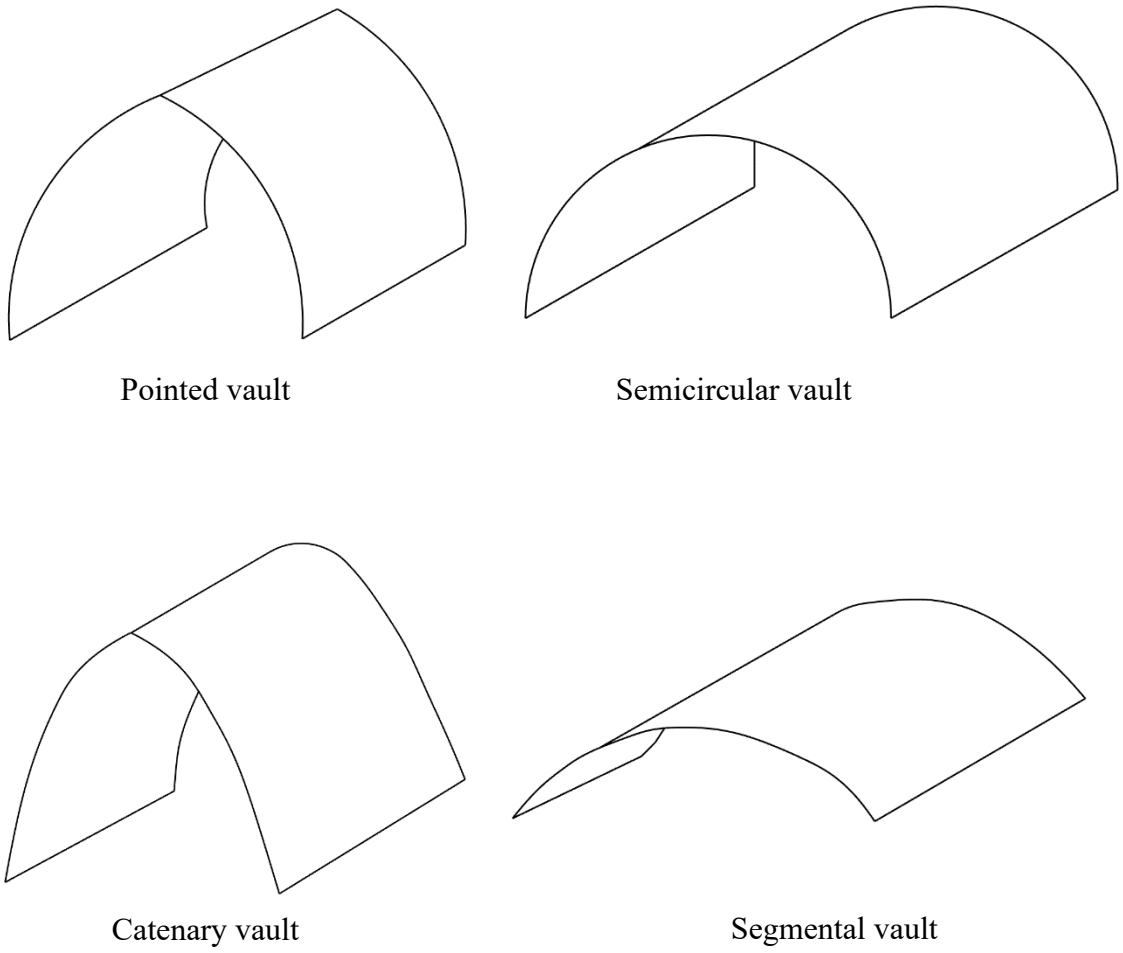


Figure 2.6: Different types of extruded vaults (Introduction to Arches, vaults and dome http://www.earth-auroville.com/)

In Figure 2.7, the arrows show the transfer of load to the ground/support in a vault. The load is transferred from the vaulted thickness to the abutment or any support on the base of the vault. The abutment must be thick enough to resist the thrust coming from the vaulted surface. Typically, the vault requires temporary support during construction until each arch is completed. Vaults/arches, however, can be built by a technique that requires no centering which will be discussed later in the Section 2.3. A groin vault is formed by the intersection of two or more vaults as in example in Figure 2.8.

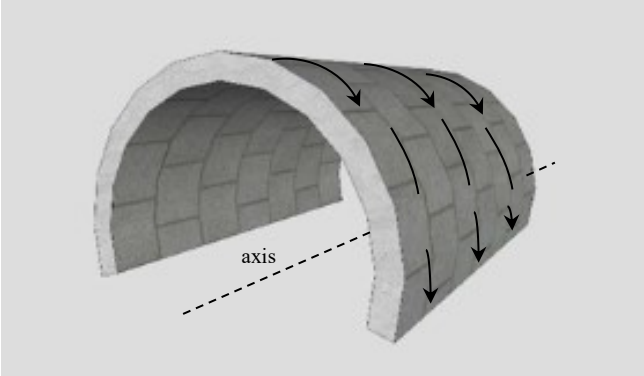


Figure 2.7: Barrel vault

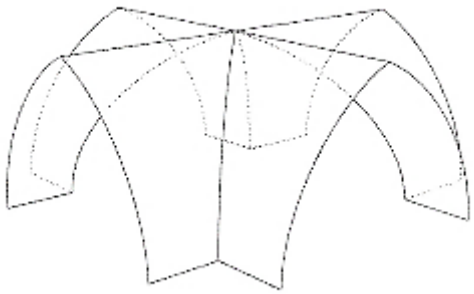


Figure 2.8: Groined vault

A corbelled vault is formed when a corbelled arch is extruded. Corbel arch, when back- weighted with masonry acts as a true arch because leaning action exerts horizontal thrust against the component parts (DeLaine, J. 1990; Van Beek, G. 1987).

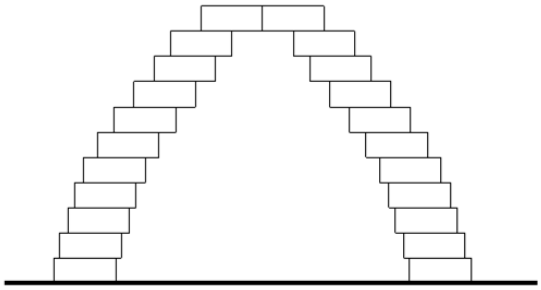
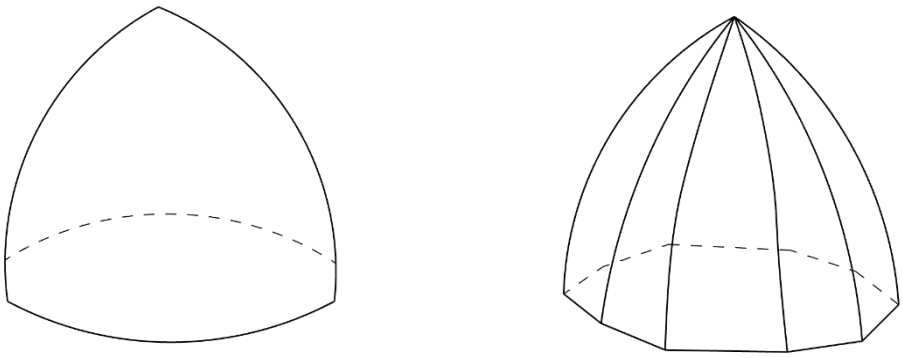


Figure 2.9: Corbelled arch

2.2.3 Domes

Meridian curves rotating around a fixed vertical axis generate domes. Domes can be of different types. Basically, the shape of the dome depends on the curve that is to be rotated. The curve can be a circular arch, a half-ellipse, or a parabola. Semi-circular curve when rotated generates hemispherical dome; ellipse generates ellipsoidal domes; parabolic curve generates paraboloid and so on. Domes can also be conical based on the type of curve. Figure 2.10 below presents some of the types of dome commonly found.



Pointed

Faceted

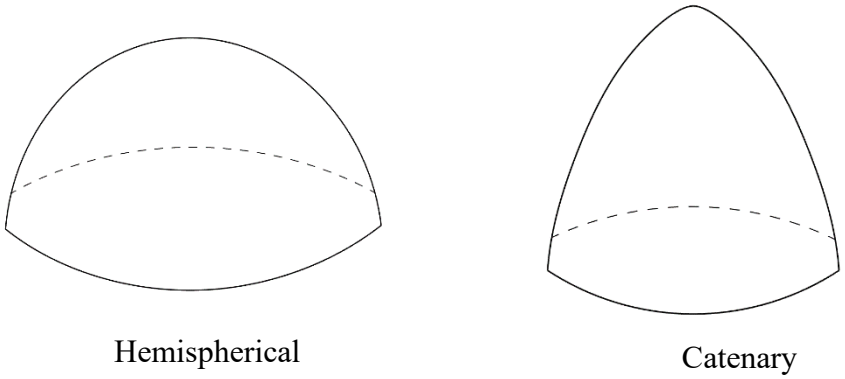


Figure 2.10: Different types of domes (image: Arun G. 2006)

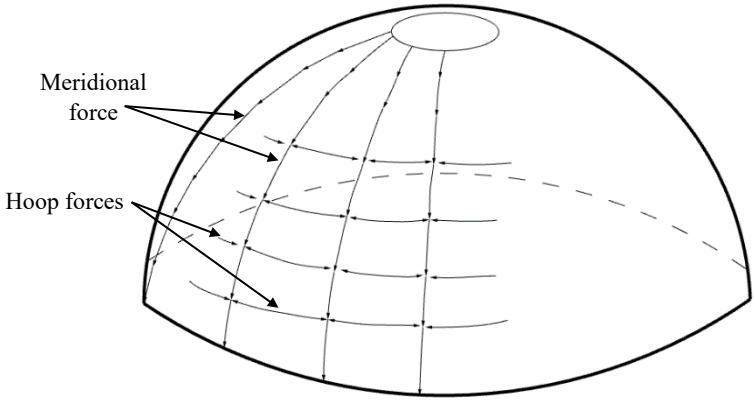


Figure 2.11: Forces acting on a dome

Figure 2.11 above shows different forces acting on a dome. The stresses in the dome are meridional and hoop. Meridional stresses act along the meridian and hoop stress acts along the parallels. The transfer of loads in dome occurs along the meridian to the support structure at the base. The dome structure is in compression and the compression force increases from the crown to the base (Lau, W. W, 2006).

In addition, dome offers hoop forces which helps to overcome the out-of-plane failure. It is possible for builders to construct domes ring-by-ring using hoop forces that can be put into play

without centering. A dome is stable even when it has an oculus on top because of the integrity of hoop force.

Generally, a dome has a circular base, but it can also be used to span square or rectangular spaces. For spanning square or rectangular spaces, the elements like pendentives, squinches or masonry corbels make the base of the dome circular (Figure 2.12) (Arun G. 2006).

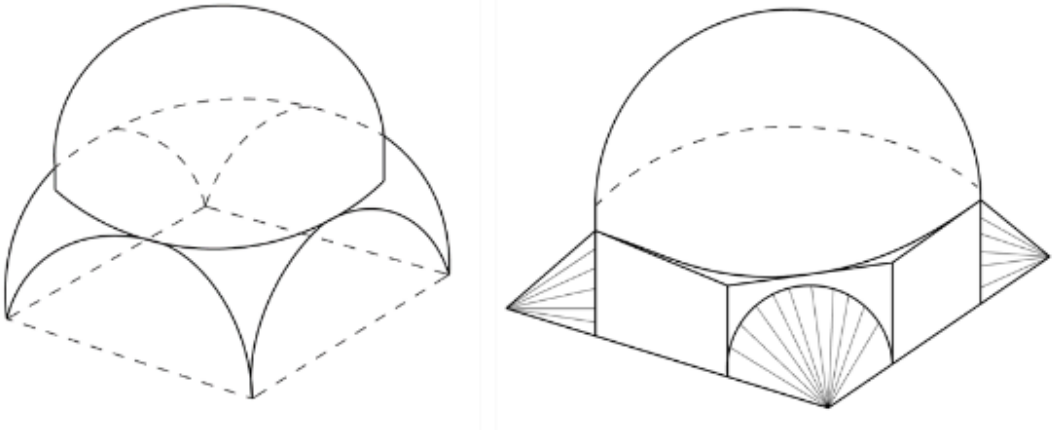


Figure 2.12: Domes on pendentives and squinches

Figure 2.13: Inside panorama of the Treasury of Atreus on the Panagitsa Hill at Mycenae, Greece (image: Wikimedia commons)

A tholos is a conical dome. It was used before 2000 B.C. to build tombs (Hood, 1960). The first known dome, 'the Treasury of Atreus, Mycenae (1325 B.C)'; is one of the largest tholoi (Figure 2.13). The base of the dome is 14.6 m (48 ft.) in diameter (Melaragno, M. 2012). The

construction is same as a corbel dome, but a tholos differs from corbel because it does not have a counterweight. Since it is not back-weighted, it exerts thrust in outward direction. Each inclined successive course exerts inward thrust. The resultant of the forces acting on both sides of each masonry unit (which forms a compressive ring) is equal to the inward force, so the structure does not fail (Allen et al., 2009).

2.2.4 Free-form masonry shells

Unlike the architectural forms discussed above, the free-form structures are irregular and lack symmetry. The free-form masonry shells have an uneven distribution of forces. Due to the nature of force distribution the construction of free-form masonry shells requires extensive centering during their assemblage (Lopez et al., 2014). For example, the free-form brick structure by ADAPt, Iran (Figure 2.14 a); clay brick tile vaults built by Block Research Group (BRG) at the Swiss Federal Institute of Technology in Zurich (Figure 2.14 b); and the Drone port prototype at Venice Biennale of Architecture 2016, Foster + Partners; required centering.

(a)

(b)

Figure 2.14: Examples of free-form masonry shells (a) Free-form brick structure by ADAPt, Iran (image: Mohammad Soroosh Jooshesh www.archdaily.com) (b) Free-form tile vault, Zurich, Switzerland, 2010 (image: Klemen Breitfuss)

2.3 Construction methods

2.3.1 Traditional techniques

Wendland (2007) describes the construction technique of half-stone vaults that can be built without using centering/formwork. Wendland developed theories of construction based on a review of historical literature related to construction techniques and implementing them by creating small-scale models and prototypes. The basic principle for constructing masonry structures without formwork depends on the self-supporting courses which further depends on the direction of the courses. The technique of constructing half stone-vaults without formwork was first introduced in 1829 by German architect J.C. von Lassaulx (Wendland, 2007). But it was also applied earlier in the construction of 'Nubian' barrel vaults in Egypt and Mesopotamia

when brick architecture was flourishing. Different types of vaults, domes, and their traditional construction techniques are discussed here.

2.3.1.1 Nubian vault

The Nubian vault construction technique is a traditional technique which was first introduced in Egypt and northern Sudan. It was revived by an Egyptian Architect Hassan Fathy. For the construction in this technique, the first course is laid roughly in a parabolic shape, leaning against a wall or any existing structure. The first course serves as the stable base for the next course to be laid on the top using mortar. While adding the course, the brick inclination starts approximately from 20° to 90° to the longitudinal axis of the vault. This construction technique requires no formwork because each course is self-supported with the inclination of units in varying angles (Figure 2.15) (Dahmen et al., 2012). Figure 2.16 shows the free-body diagram of a masonry unit and a closed force triangle indicating the equilibrium condition of the unit.

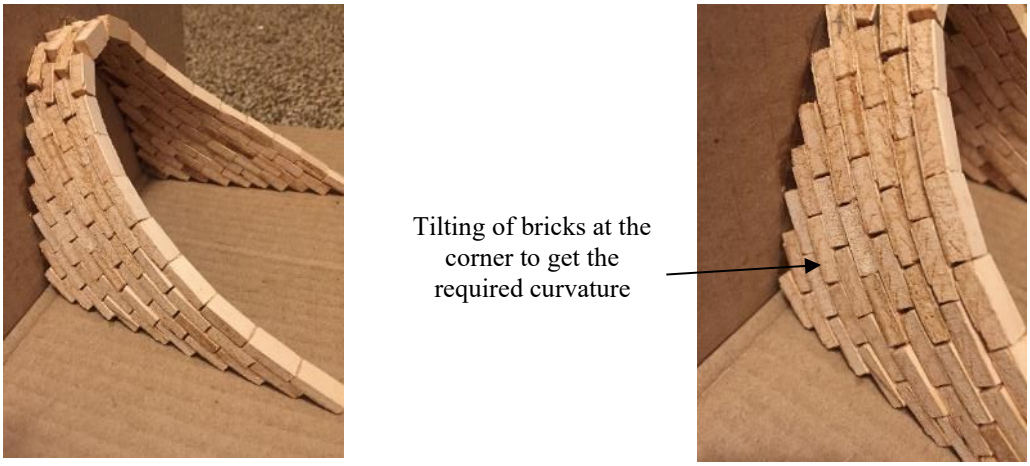


Figure 2.15: Small-scale model of Nubian Vault construction technique

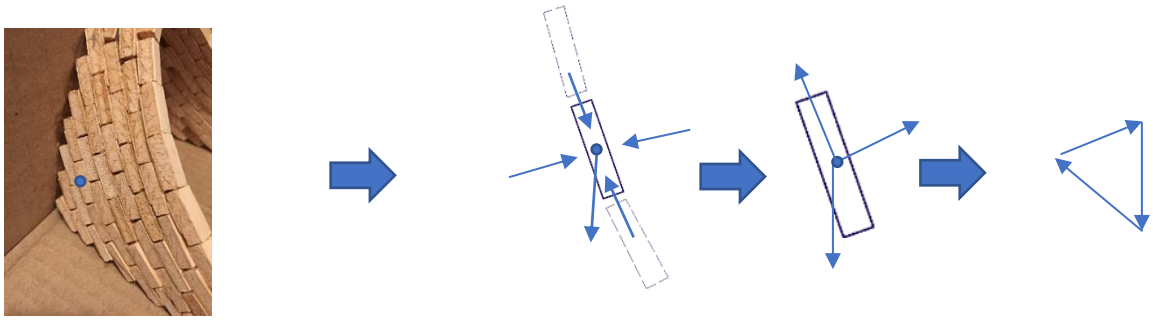


Figure 2.16: Free-body diagram of a unit showing all forces acting on it

2.3.1.2 Barrel vault

During the early medieval period, barrel vault was in practice for constructing churches. Among the different types of vaults, the barrel vault is simple and easy to construct. The barrel vault differs from the Nubian vault as the former has a base/ abutment to support the courses. Generally, the barrel vault requires centering during construction because it is the extrusion of true arch which needs a support until the keystone is placed. The principle of self-supporting courses however, can also be implemented for the construction of the barrel vault as well. The construction technique is like that of the Nubian vault. The first course is laid leaning against a wall or a previously constructed arch. Other courses are laid one after another, leaning against the previous one. The vaults are built in the shape of a catenary to reduce the risk of buckling caused due to the longer courses. The barrel vaults with horizontal courses can also be constructed without centering / formwork. But the upper courses can be stable only over a very short span working as a flat arch.

2.3.1.3 Cross vaults and Sail vaults

A cross vault is formed by the intersection of two or more barrel vaults. It consists of a series of several arches. The construction can also be based on the principle of self-supporting courses whose stability is determined by the geometry.

The cross vault has the diagonal groins or ribs which are constructed using support by centering arches. The webs between the ribs are laid in courses with gradually increasing curvature, and supported by the ribs which run along a diagonal. Masonry units are laid in a bed of mortar. The successive courses are built with the support of the previous layer. Builders avoid direct intersection of arched courses in the groins by laying bricks in horizontal planes and forming corbels stabilized by the weight of the arched courses which are placed on top of them. Laying of bricks is repeated until the arched courses intersect more easily. Dove-tail pattern is common while building cross-vaults. In this pattern, the courses are diagonally tilted perpendicular to groins. The caps for the vaults can be built separately and can be connected over the diagonal ribs with mortar joints (Wendland, 2005).

Figure 2.17 shows a small-scale model of a sail vault with a constructed guideline to enable the shape of the vault. In the sail vault construction, the courses are laid perpendicular to the edge of the temporary support. Each course is inclined increasingly inward in order to get the required curvature. Care must be taken to give curvature to the lowest course. Courses from the four squinches meet forming the groin. A sail-vault with courses parallel to the edge of guideline can also be constructed based on the same principle.

Guideline for the shape

Figure 2.17: Small-scale model of a sail-vault

2.3.1.4 Dome

In a dome, it is easier to stabilize the courses than in other vaults. The form of a dome can be controlled by using a rotating template fixed to the vertical axis. For the construction of the dome, all units are inclined in a radial direction to get the required shape. The bed surface in each course is conical. The masonry courses are self-stabilized in a dome with horizontal compression rings. At the top where the curvature flattens out, a different pattern of courses is applied because the construction is difficult as the beds are inclined. The inclination inward of each course must increase uniformly from springing to crown to get the required curvature (Wendland, 2003; Wendland, 2004).

The model of a dome in Figure 2.18 demonstrates the technique of construction without using formwork. At first the base course is laid. Then, the subsequent course is laid by rotating each course towards the center. In real practice, the inclination of the courses is achieved by the deviation of the mortar bed. Each course must be shifted inwards as well to get the required curvature. Figure 2.19 shows an example of historical dome structure.

Figure 2.18: Small-scale model of a dome made to understand the construction technique

Figure 2.19: Example of historical dome structure - Santa Maria del Fiore, Italy

(Source: media.architecturaldigest.com)

2.3.1.5 Square vault

A square vault is basically used to span and provide roofing to square spaces. The small-scale model of a square vault shown in Figure 2.20 was made to understand the construction process in the real practice.

The construction of this vault type is based on the leaning brick technique. The construction starts from the corners of the space. Bricks are laid leaning against each other as shown in Figure 2.20. The courses from one corner meet courses from the adjacent corner forming a ridge. The vault is completed by adding courses following the same technique. The stability of this type of vault depends upon the arch shape each course forms (Ponce et al., 2004).

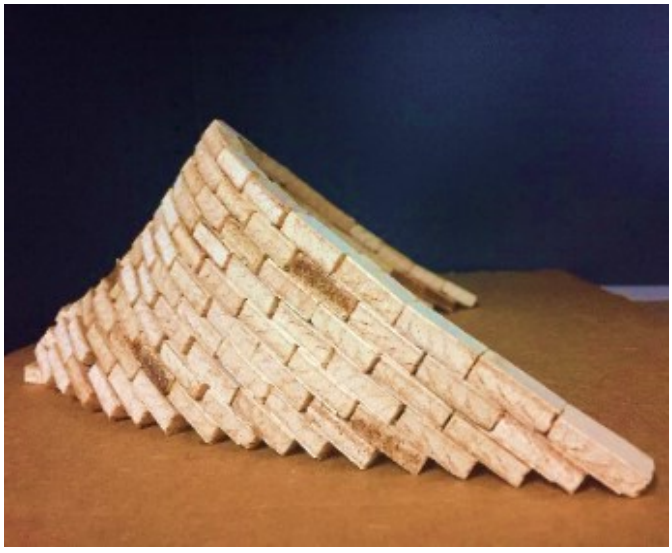


Figure 2.20: Small-scale models of a square vaulting

2.3.2 Guastavino construction technique

A Spanish builder, Rafael Guastavino, used the technique of thin tile structural masonry, also named timbrel vaulting or Catalan vaulting, for building large span factories in Catalonia in the 1860s. He, along with his son after immigrating to the United States, established a company which built many public and private spaces. The technique of timbrel vaulting (Guastavino masonry vault) gained its popularity because of its advantageous properties like the ability to span large spaces and fire resistance. One of the most significant Guastavino's works in the United States is the dome of the Cathedral of St. John the Divine in New York, which spans 40 meters (131.2 ft.) (Ramage, 2007). The construction of timbrel vaulting however, is limited in practice nowadays because of the advent of steel and concrete in the construction industry. Very few contemporary projects utilize this technique.

In timbrel vaulting, tiles are usually of standard size i.e., 25 mm thick and approximately 300 mm long and 150 mm wide. The common pattern used for this construction is herringbone-pattern. For the construction, gypsum mortar is used for the first two tile layers due to its fast

setting properties. The gypsum mortar can create a bond sufficient to hold two tiles together. Succeeding layers are laid by applying cement-based mortar which is stronger and more moisture resistant. The way in which the layers are laid and the shape of the timbrel vault makes it possible to build large and thin masonry structures without external formwork. Formwork guides are provided at intervals to guide the shape of domes or vaults. The number of guides depends on the skill of the mason to place each tile. In this technique, double curvature is provided to get the line of thrust within the thickness (Figures 2.21 and 2.22) (Ramage, M. H. 2004; Ochsendorf, J. 2014; Angelillo, 2015). The description about the line of thrust and its location in the shell will be covered later in this chapter.

Figure 2.21: Tile dome, Pines Calyx, England (Ochsendorf, J. 2014)

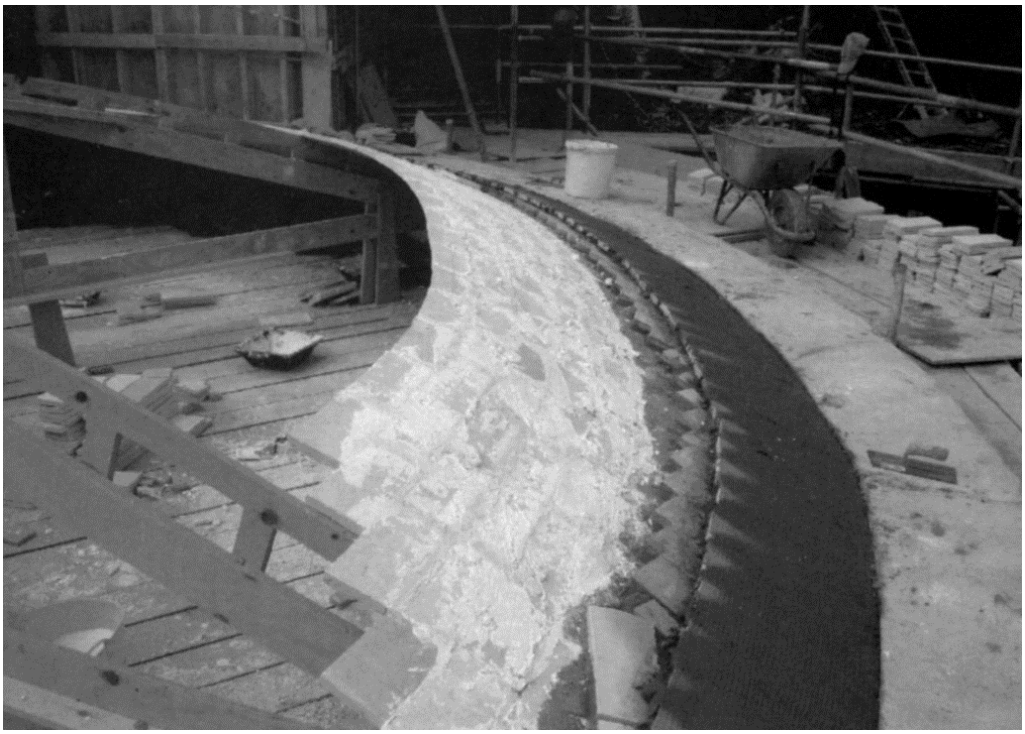


Figure 2.22: Tile Layering technique in Guastavino construction (Ramage, 2007)

2.3.3 Construction method for free-form masonry shells

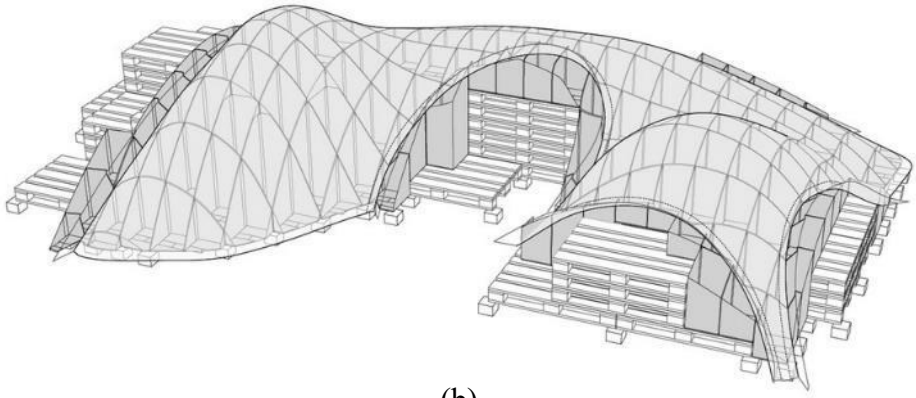
Designing a free-form masonry shell is not an easy task. It requires complex calculations. The computational tool developed by the research group at ETH Zurich has made it possible to design compression-only free-form masonry shells. This section discusses examples of free-form masonry shells and their methods of construction.

The first free-form masonry shell under discussion was built by BRG in ETH Zurich, Switzerland. It was built to check the computational method developed to design compression-only masonry structures. Davis et al. (2011) describes the project of thin tile vaulting whose shape was designed and analyzed using the Thrust Network Analysis originally proposed by Professor Phillipe Block.

BRG employed methodology which can be categorized into three parts:

1. Form finding: The complex form of the vault was formed by enabling the Thrust Network Analysis (TNA) in RhinoVAULT. The plan shape, shape of open edge arches, internal distribution of force, and location of footing which impact the shape of the surface were provided in RhinoVAULT to design the form of the vault.
2. Cardboard formwork system: The centering used during the construction was made from recycled cardboard boxes. The force on the formwork was calculated to evaluate the stability of the cardboard boxes. The cardboard boxes were cut into required shapes using a CNC cutting machine. The cardboard boxes were supported by shipping palettes which reduced the quantity of the boxes to be used. The cardboard boxes were used because they were easier to cut. Cuts were made using a CNC cutting machine and easy to remove. The stacking of the boxes was defined using Rhino-scripts.
3. Decentering: To allow a safe removal of the formwork, sealed plastic tubes were used in the base of the entire formwork. The formwork contained a stack of cardboard spacers. After completion of the construction, the tubes were filled with water to saturate the cardboard. The saturated cardboard compresses under the load of the palettes and lowers the formwork.

(a)



(b)

(c)

Figure 2.23: Construction method of thin-tile vaulting(a) and (b)shows details of centering used during construction (c) completed free-form tile vault, Zurich, Switzerland, 2010 (images: Klemen Breitfuss)

Limitations and findings:

- Controlling the structural stability of the vault in site was challenging as there was small bending in the shell because the tiles did not touch the formwork in some places and caused hairline cracks.
- As the construction was carried out in the open air, the condensed water present in low-quality cardboard was enough to compress the spacer before the construction was completed.

At the Eme3 (the 2013 International Festival of Architecture) in Barcelona, the first ever human scale free-form tile vault was built. It was built by Map13 Barcelona (Lopez et al., 2016). RhinoVAULT, which was used previously in 2011 by BRG of ETH, was employed to design the free-form tile vault named *Bricktopia Pavilion.* The construction process is shown in Figure 2.24.

(a)

(b)

(c)

Figure 2.24: Bricktopia pavilion Eme3 festival 2013, Barcelona (image: Manuel de Lozar + Paula López Barba)

The structure was an unreinforced masonry vault. The pavilion had a maximum height of 4m (13.1 ft.) and had spans ranging between 5m (16.4 ft.) and 7m (22.9 ft.). The materials used in its construction were traditional handmade bricks of size 140mm × 15mm × 280mm (5.5in. × 0.6in. × 11in.) for the first layer. For the second layer, hollow brick of size 140mm × 40mm × 280mm (5.5in. × 1.6in. × 11in.) was used and the final third layer was of solid bricks of size 140mm × 43mm × 280mm (5.5in. × 1.7in. × 11in.) and it was used only over the biggest vault. The bricks were laid in a natural rapid cement binder.

The RhinoVAULT was used to design the shell structure. After several design iterations, a final shape was generated. Since this project had time and budget constraints, the centering materials were selected so that they were cost efficient and required less time to install. Because of the uneven distribution of forces in the free-form shell, centering was required during construction. The weight of thin tile is much less than standard size brick. So, a centering of cardboard was used (making the construction material efficient and less expensive). A 2m × 2m (6.6ft. ×6.6ft.) cardboard formwork and scaffoldings were used. Additional steel rods were used over the cardboard as a guide to shape the shell and carry the load of the vault. Depending on the load distribution in the vault some locations were provided with denser steel rod grids. After building the grids, cardboards were removed so that the worker could stand on the scaffoldings (Lopez et al., 2014).

Other examples include the drone port prototype built at the Venice Architecture Biennale 2016 by Foster + Partners; stone-tile vaultings by BRG ETH; and a marble pavilion at Portugal shown in Figure 2.25 (Howe, 2016).

(a)

(b)

Figure 2.25: Marble pavilion in Portugal designed by Freehaus and Cultural Geometries (a) use of wooden centering for construction (b) completed pavilion (images: Cultural Geometries Group)

Deuss et al. (2014) discusses the alternative of wooden/cardboard formwork for the construction of masonry shells. Deuss's research proposes chains to hold the masonry blocks temporarily

during the construction. Like in medieval vault structures where tensioned ropes were used, chains hold the masonry blocks while constructing.

The masonry structure was modelled as a set of rigid blocks represented by closed triangle meshes. The site had the anchor points which hold the blocks by chains. On the surface of the block, a hook was provided which helps in attachment to the anchors. The researchers used an algorithm which can process the models. A proper sequencing in construction was achieved which reduced the demand of centering and eventually reduces the cost of construction and the materials required. The strategy of sequencing helped to build the masonry structure without centering, as construction can be done gradually placing the masonry blocks one by one and with support of each other. The masonry structure and the construction sites were given as the input and the construction sequence was optimized. Among the several construction sequences, the one which used the minimal sets of chains was chosen. To check whether the blocks were in static equilibrium or not, algorithms were used. However, this research was limited to the small scale and 3D printed models.

2.4 Mechanics of masonry arches

The stresses developed due to the vertical loads on the arches/vaults pass through their curved path. All curved surfaces of vaulted and domed structures are derived from arches. So, it is important to understand the arch and its behavior. Robert Hooke in 1675 found the funicular shape of an arch by observing the shape of a hanging chain- "*as hangs the flexible line, so but inverted will stand the rigid arch*" which describes the behavior of an arch. When a chain is acting under its self-weight, the shape formed is a catenary curve, which when inverted, forms a rigid arch in pure compression.

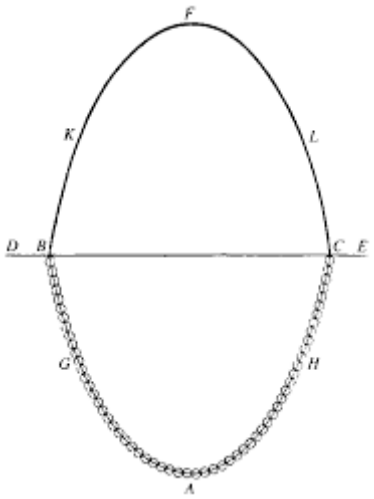


Figure 2.26: Poleni's drawing of Hooke's analogy between arch and hanging chain (Heyman,1982)

The main concept for any masonry shell to be stable is the arch shape it forms. The stability of the structure does not depend on the strength of material but on the geometry. The well-shaped geometry of an arch or vault contains the internal compressive forces within their masonry. As long as the path of compressive forces (line of thrust) lies within the masonry, the structure is stable. By using graphical statics, the behavior of masonry arches /vaults under the application of the external loading can be analyzed.

Figure 2.27 (a) is a system AB which is acted on by parallel loads W_1, W_2, W_3 in the vertical direction. R_1 and R_2 are the reactions at A and B, respectively. If the system is assumed to be a weightless string, it will be in tension as shown in Figure 2.27 (b). The reactions to the tension at A and B in Figure 2.27 (b) are inclined to have horizontal component which tries to pull A and B towards each other. The form diagram AEDCB when inverted represents the line of thrust of an arch system carrying load W_1, W_2, W_3. Figure 2.27 (c) is a force diagram with line lengths scaled to represent the magnitude of forces. The force diagram gives the magnitude of force acting on each element of Figure 2.27 (b).

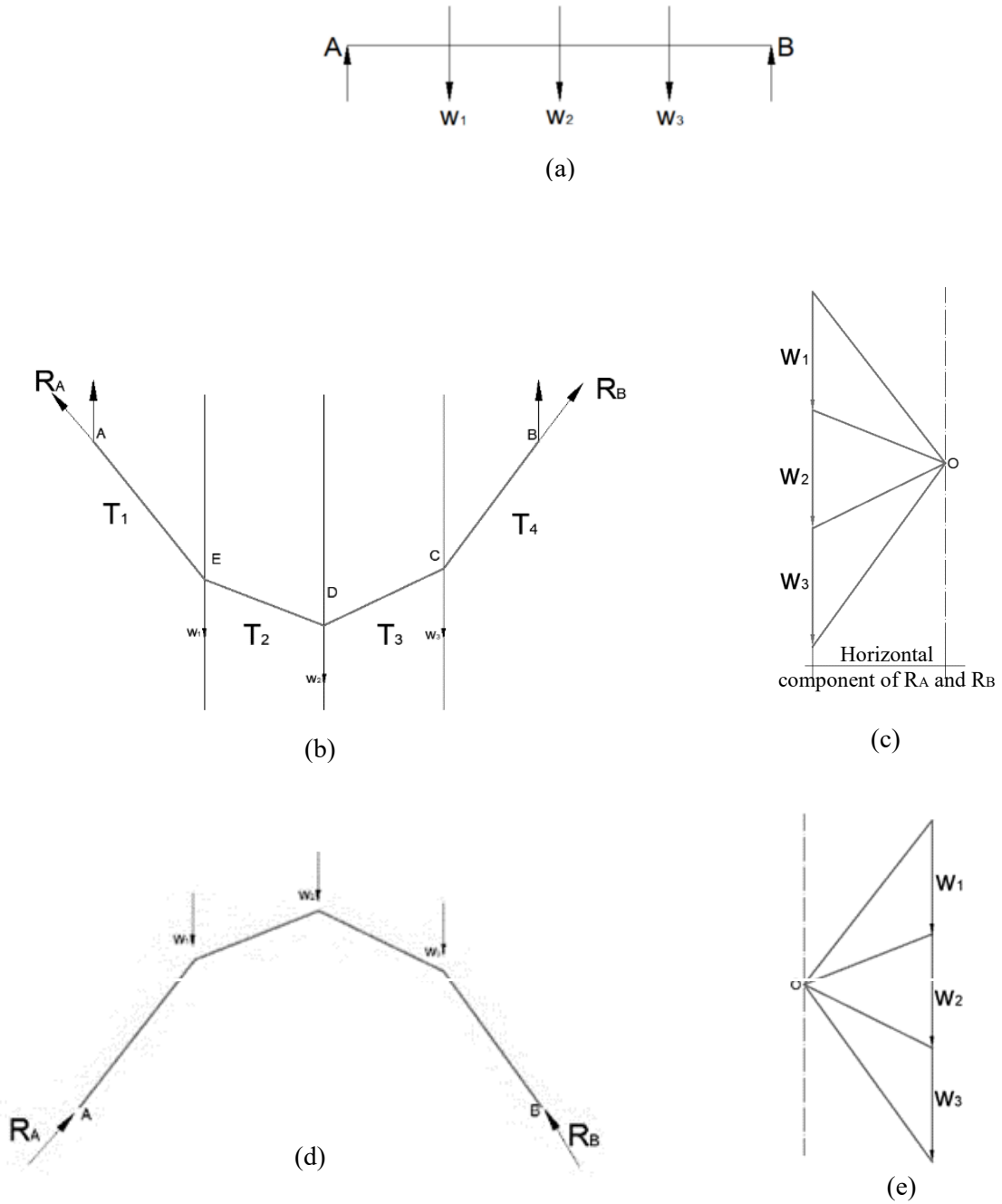


Figure 2.27: Funicular analysis (a) System AB with external loads (b) form diagram (c) force diagram. Shape of the form diagram (d) due to change in the position of the pole in force diagram (e).

The thickness of the voussoirs surrounding the line of thrust influences the stability of the arch. The magnitude and the position of loads across the span determine the line of thrust. Moreover, the magnitude of thrust at the abutment varies according to the rise of the arch. The rise of thrust line of arch in Figure 2.28 (a) is twice than that of arch of Figure 2.28 (c) (Beckmann et al., 2012; Heyman 1982; Allen et al., 2009).

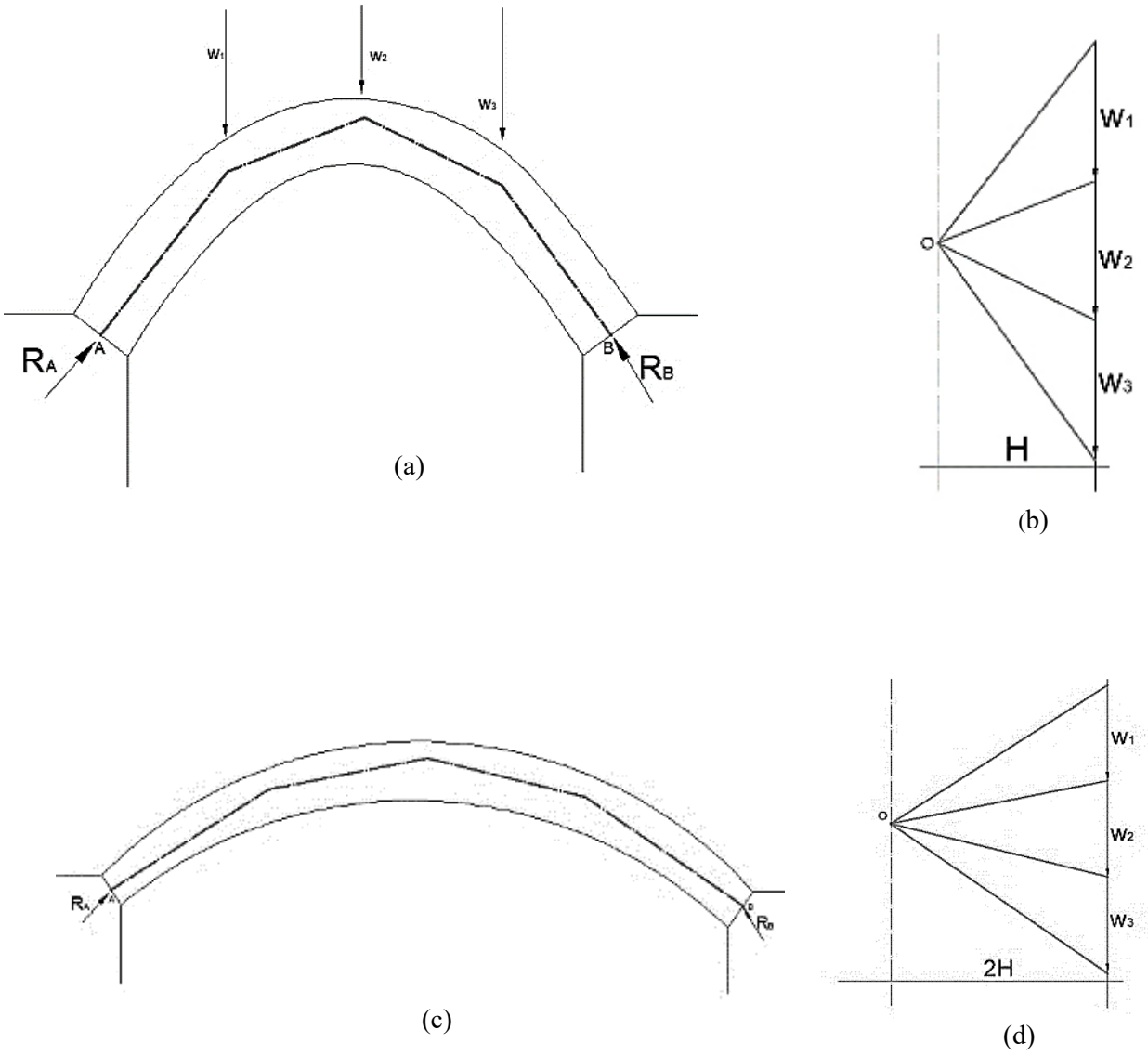


Figure 2.28: Effects in shape of thrust line (a), (c) by changing the position of the pole in the force diagram (b) and (d)

The middle-third criteria can also be applied to the masonry arches to analyze their stability. If the line of thrust lies within the middle third portion of the thickness of arch, then it is stable. Figure 2.29 below explains the middle-third rule for the masonry structures.

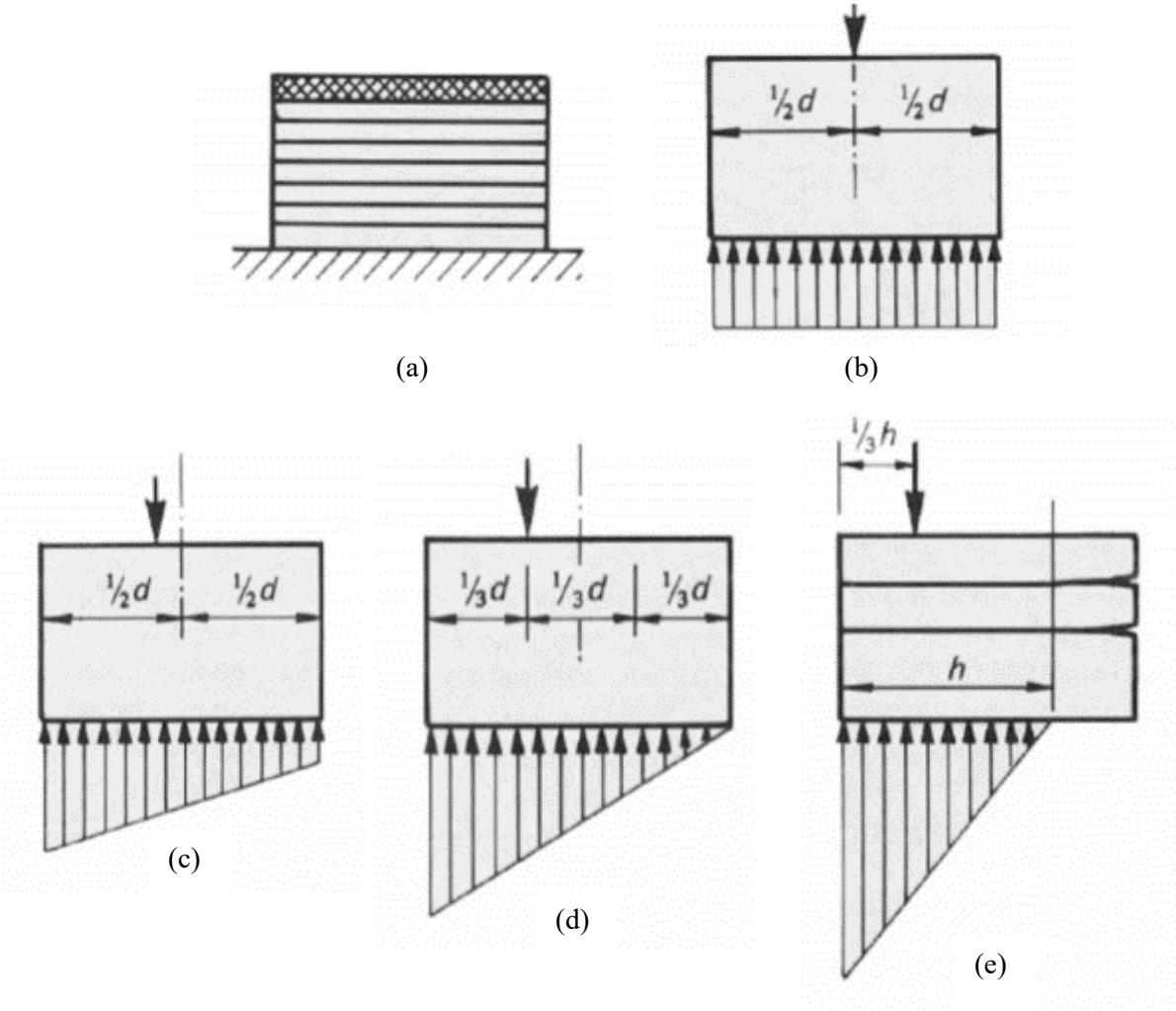
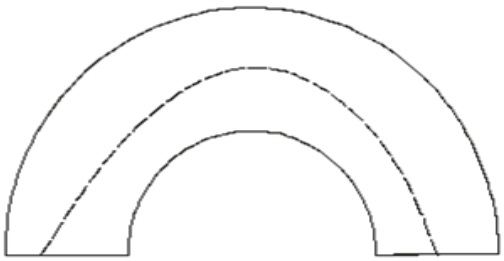


Figure 2.29: Position of point load and compressive stress distribution (Heyman 1982)

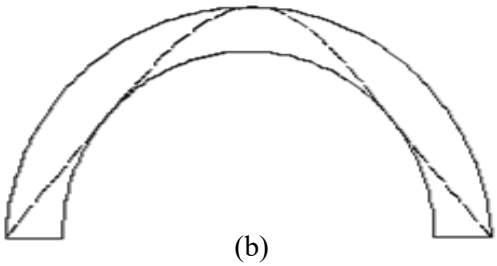
Figure 2.29 (a) has a pile of stone slabs and is surrounded by a rigid foundation. Figure 2.29 (b) shows point load acting on the middle of the system. In Figure 2.29 (c), the load is shifted slightly away from the center. In Figure 2.29 (d) point load is at the one-third of section of the stone slab system. At this point, the stress at one end is zero. Further moving the point load away

from the one-third part as in Figure 2.29 (e), the other edge of the system is in tension. But the pile of stone slabs cannot take tensile forces, which increases the compressing stresses exponentially (if mortar is not applied). The slabs start to separate due to the action of bending moment. Therefore, from the example mentioned above, when the line of thrust lies within the middle-third of the structure no tension is developed, and the entire structure is stable because of the pure compression.

In addition, to derive a numerical factor of safety, it is necessary to understand the condition in which an arch fails. A masonry arch forms hinges when there is a rotation at the abutment. Masonry arches fail due to the mechanism of forming hinges as shown in Figure 2.30 (c).



(a)



(b)

(c)

Figure 2.30: Geometrical factor of safety

Figure 2.30 (a) above shows the semi-circular arch containing a line of thrust within it, so it is considered safe. However, the thinner arch in Figure 2.30 (b) can also carry the same load as it has shrunk just to accommodate the line of thrust. Limiting the value up to which an arch could be shrunk so that it is safe, gives the geometrical factor of safety. So, the geometrical factor of safety is the ratio of actual thickness of the arch and the minimum thickness of arch obtained by scaling just to enclose the line of thrust (Heyman,1982; O'Dwyer, 1999).

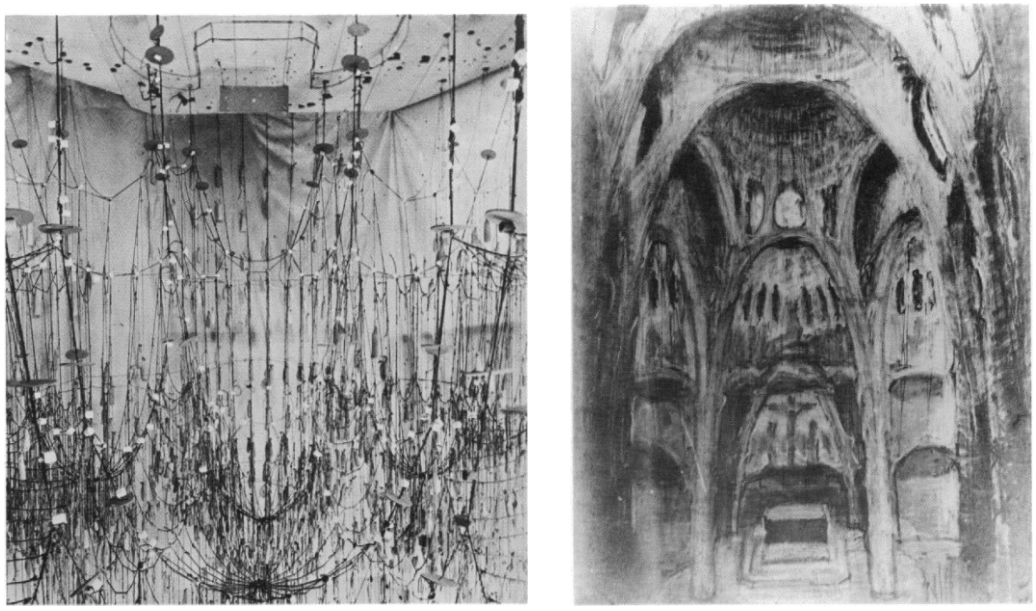
2.5 Design methods

2.5.1 Physical form-finding methods

Form-finding is a process in which new forms of a statically equilibrated structure are generated by controlling the parameters like site conditions, height, shape of the plan, etc. Before the invention of computer-based computational tools, designers used analog models of hanging chains and hanging cloth to study forms of arches and shells.

Robert Hooke devised a method of form-finding for an arch. In this method, a chain is hung on its own weight and is in tension. The inversion of the chain gives the shape of an arch in pure compression. Adding different weights to the chain provides possibilities of generating different new funicular shapes (Adriaenssens et al., 2014).

Antoni Gaudi used the method of hanging chains to generate vault forms. He was the first architect to use the hanging model method to design the whole building. Some of Antoni Gaudi's projects include the Colonia Guell Church and the Sagrada Familia Cathedral shown in Figure 2.31 (Collins, 1963; Wendland 2000).



(a) (b)

Figure 2.31: Gaudi's method of form-finding (a) Interior view of hanging model of the Colonia Guell Church, Barcelona, Spain (1915) (b) Gaudi's design sketch on the inverted photo of the interior of the hanging model. (images: Collins 1963)

Heinz Isler, an innovative Swiss structural engineer of 20th century, is known for his contribution in the field of concrete shells. His methods include form finding using physical models and determining structural strength by load testing. No computer-based tools to find the form or to check the structural stability were used in his methods. Yet, the structures designed are of great elegance. Several projects shown below in Figure 2.32 include Sicli SA factory in Geneva (Figure 2.32 b); roof of the N1 autobahn service station at Deitingen Sud (Figure 2.32 c); concrete shell roof of the garden center Wyss in Zuchwil (Figure 2.32 d), etc.

(a)

(b)

(c)

(d)

Figure 2.32: Form-finding by Heinz Isler (a)Latex rubber physical model for form-finding of Sicli building, Geneva (image by: John Chilton (b) Sicli building, Geneva (c) Service station at Deitingen (d) Wyss Garden Center, Switzerland (Image by: Yoshito Isono, HYPERLINK "http://en.structurae.de/persons/data/index.cfm?id=d000017" \t "_blank" \o "Structurae link" Structurae)

To generate the form, Isler used freely shaped hill, inverted hanging cloth, and membrane under pressure to generate shapes. In freely shaped hill method, the material (i.e., concrete) can be poured over the freely standing hill and allowed to set. Then, the shell can be taken out or hill can be excavated. This is probably the cheapest way of building shell structure. Moreover, in the inverted hanging cloth method, a cloth is supported on the corners and hung under the gravitational pull. The shape formed is then inverted which gives a shape for a shell structure. In the method of membrane under pressure, soap film under pressure generates shapes for the shell structures. Although, this method is not suitable for large scale structures as membrane under pressure is acted upon by forces that are normal to the surface, and not in the direction of gravity (Chilton, J. 2009; 2010; 2012).

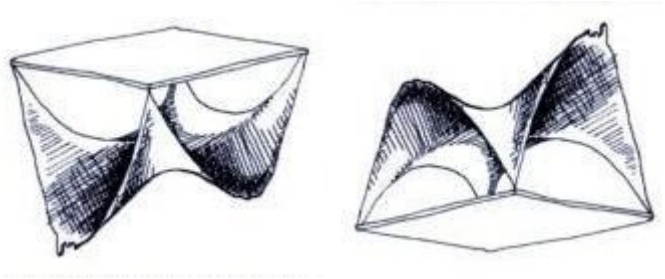


Figure 2.33: Isler's hanging cloth model sketch by Larsen and Tyas (Structurae.net)

2.5.2 Computational methods for designing shells

2.5.2.1 Force density method

Force density method has been commonly used to generate the forms for prestressed and inverted structures. This method enables the form generation through linear systems of equilibrium equations and has been applied to design of roof membranes in tension and timber shell roofs. This method is not dependent on the properties of materials. So, after the satisfactory shape is generated, building components and sizes can be designed after the shape is determined

(Adriaenssens et al., 2014). The algorithm is shown in Figure 2.34 and detailed in Adriaenssens et al. (2014).

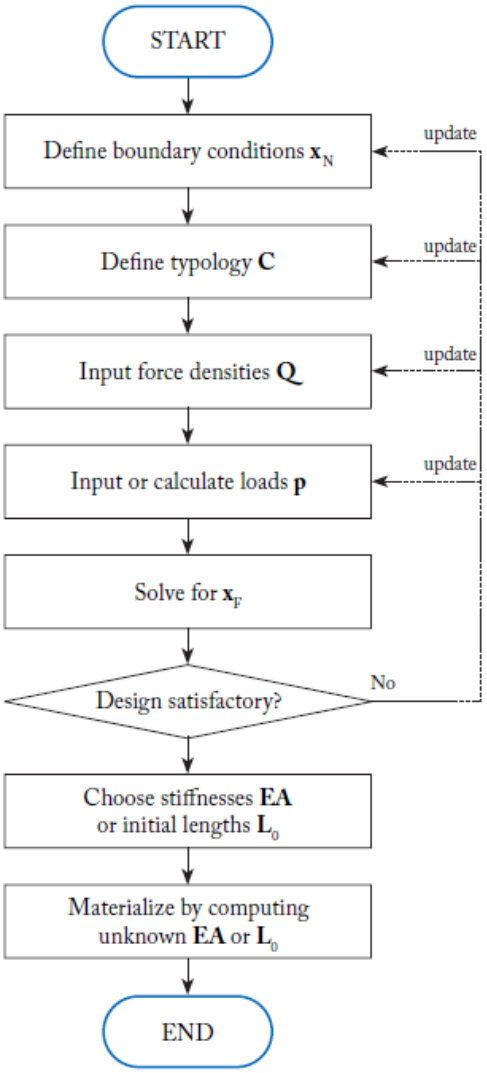


Figure 2.34: Flow chart for force density method (Adriaenssens et al., 2014)

2.5.2.2 Dynamic relaxation

Dynamic relaxation is a numerical method of form finding which was developed by Alistair Day in 1965. It works based on Newton's Second Law. In the engineering practice, it is mostly used to find equilibrium forms of cable and tensile structures. It generates equilibrium forms under

gravity loading. This numerical procedure with the algorithm shown in Figure 2.35 traces the motion of a structure through time under applied loading (Adriaenssens et al., 2014).

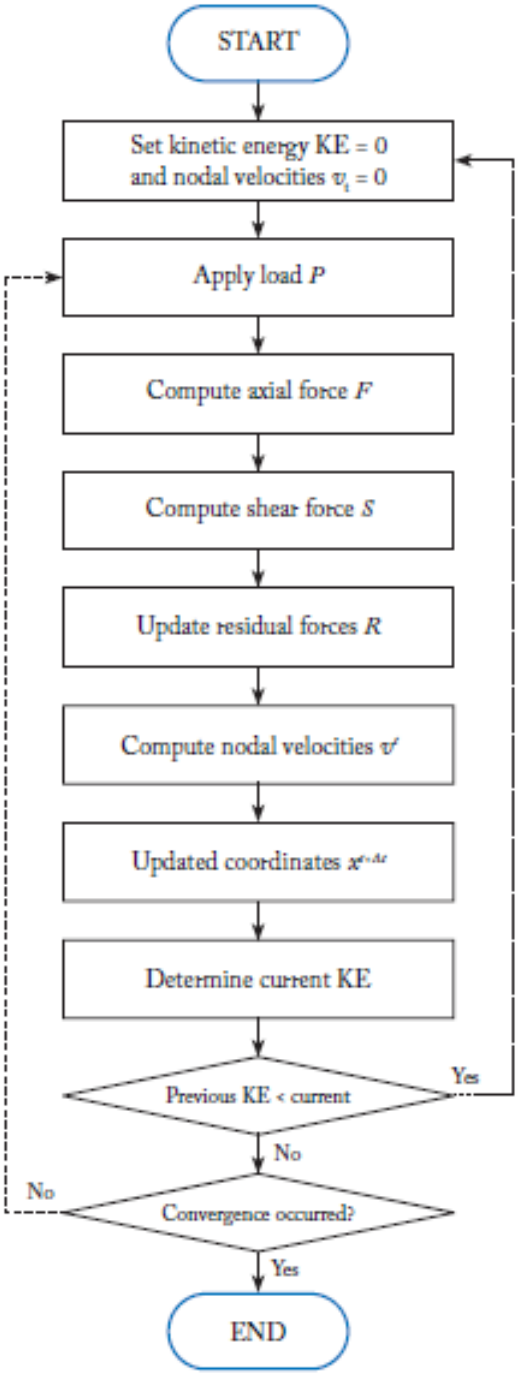


Figure 2.35: Flow chart for dynamic relaxation method (Adriaenssens et al., 2014)

2.5.2.3 Particle-spring systems

This method is based on simulation which generates the equilibrium states by defining mesh topology. Initially a low-poly mesh (i.e., polygon mesh having low number of polygons) is created which provides control over the boundary conditions. So, designers can go through different iterations. Using the subdivision algorithm, the low-poly mesh is converted to a high-resolution mesh. The force calculation on this system depends on the law of gravitation, viscous drag (friction), and Hooke's law of elasticity (Adriaenssens et al., 2014). The algorithm is shown in Figure 2.36.

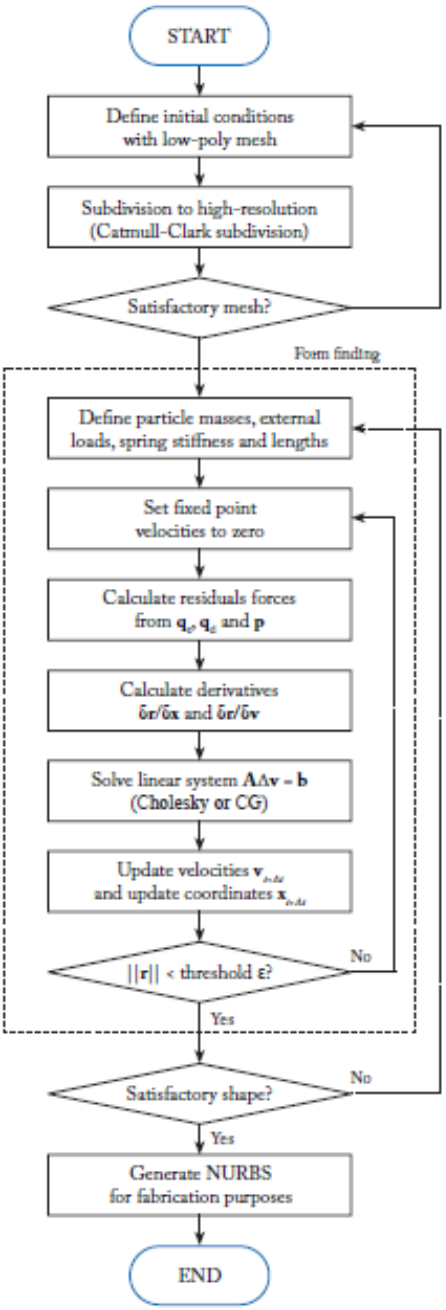


Figure 2.36: Flow chart for particle spring method (Adriaenssens, et al., 2014)

2.5.2.4 Thrust network analysis

Different design tools are available to explore new forms by analyzing in terms of force polygons and form diagrams. Thrust Network Analysis (TNA) is one of the design tools for exploring three-dimensional funicular forms. It was developed in Philippe Block's Ph.D. dissertation advised by Professor John Ochsendorf at MIT. This tool helps to design and analyze the stability of complex compression only vaulted structures.

The force distribution can be controlled within the structure with the application of this design tool. The TNA tool enables the generation of complex and aesthetically pleasing forms. It can be implemented by using *RhinoVAULT* which was developed by Matthias Rippmann based on the theory of TNA proposed by Philippe Block. RhinoVAULT is a plug-in of the Rhinoceros software. Using this plug-in, designers can generate several complex forms of compression-only shell structures. Figure 2.37 shows the user interface of the RhinoVAULT. The algorithm of thrust network analysis is shown in Figure 2.38.

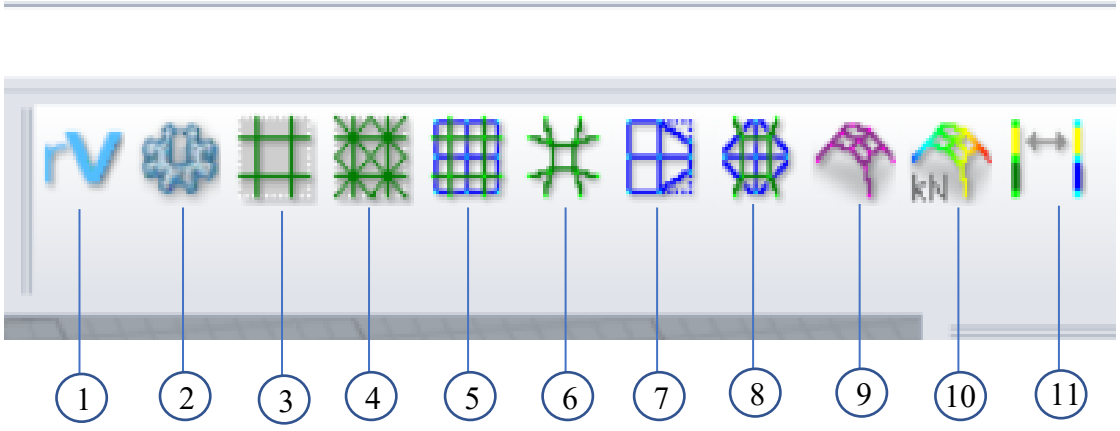


Figure 2.37: RhinoVAULT Plugin

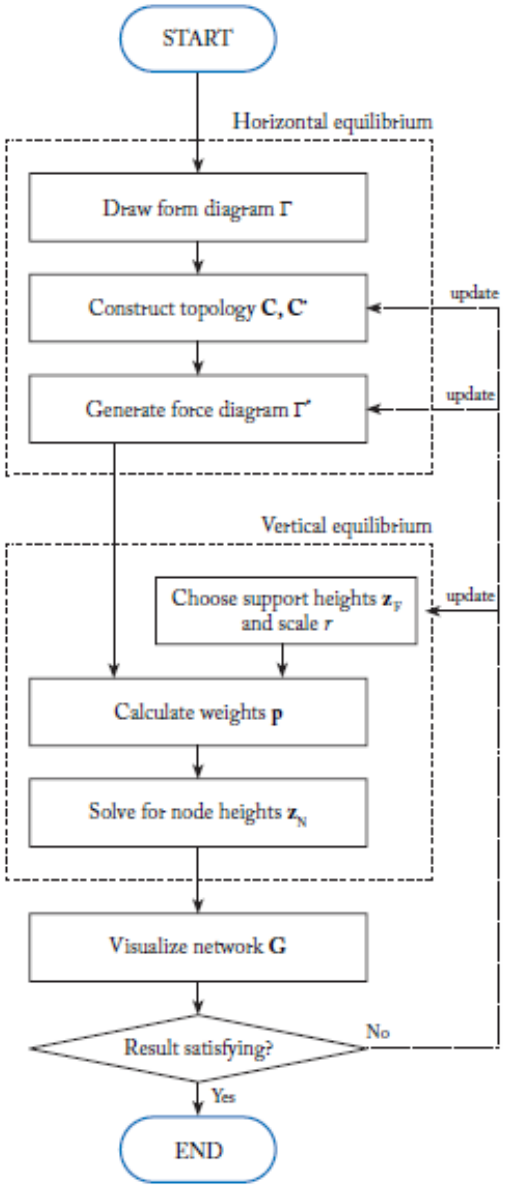


Figure 2.38: Flow chart for thrust network analysis method (Adriaenssens et al., 2014)

To create the shape of a masonry shell, complex boundary conditions can be set while taking the force patterns into consideration. TNA method allows control over the force distribution on the structure. Therefore, by implementing TNA free forms can be generated like the ones created by (Block et al., 2007; Davis et al., 2011; Deuss et al., 2014; and Rippmann et al., 2013).

2.6 Summary

This chapter briefly discussed the masonry materials; various geometries of masonry arches/vaults; construction methods using half-stone and thin clay tiles; mechanics of masonry arches; and different types of form-finding methods used currently. Historically, masonry shells and vaults have been constructed without any centering. This was possible because each unit was self-supported under its own equilibrium condition.

Historically, form finding methods like hanging chains and hanging cloth membranes were widely used. But, when it started to be used for more complex geometries, the design process became tedious and time consuming. Today, computational form finding methods have replaced analog methods using strings and weights. Different digital design tools are available, as discussed in the literature for form finding of shell structures. Complex shell structures have been constructed using these tools. In current practices, however, centering is typically used to construct the vaults.

Based on the literature review presented here, research approaches have been adopted to meet the objective of the thesis. The thesis focuses on achieving equilibrium conditions for each masonry unit, essential to avoid the use of centering during construction. A computational tool is developed to generate 3D models in Rhino 6.0.

CHAPTER 3: EQUILIBRIUM OF A TWO-DIMENSIONAL MASONRY ARCH WITHOUT CENTERING

This chapter of the thesis presents a two-dimensional equilibrium approach for two cases. The first case is where bricks are not bonded with mortar and the second case is where bricks are bonded with mortar. Furthermore, mathematical derivations have also been completed by inducing the uniformly distributed loads in vertical direction for both cases.

3.1 Two-dimensional equilibrium approach for arch without loading

The stability of a masonry structure does not depend on strength of material for all but on the geometry. The correct proportions and the geometry of structures are the reasons behind the existence of the historical masonry structures. The geometry should be such that the resulting stresses are adequately accommodated (Heyman, 1982; Huerta, 2006).

This section discusses the equilibrium approach to achieve a stable structural geometry. The approach has been discussed below for two cases: overhanging bricks when not bonded with mortar and overhanging bricks when bonded with mortar. The discussion assumes the following:

1. Each brick is rectangular, of same size and, have same density
2. The base is horizontal and rigid.

3.1.1 Overhanging bricks without bonding

The main concept on projecting a brick, yet maintaining its stability is to ensure that the 'Center of Mass' is supported by the firm base. Let '*w*' be the weight and '*L*' be the length of each brick.

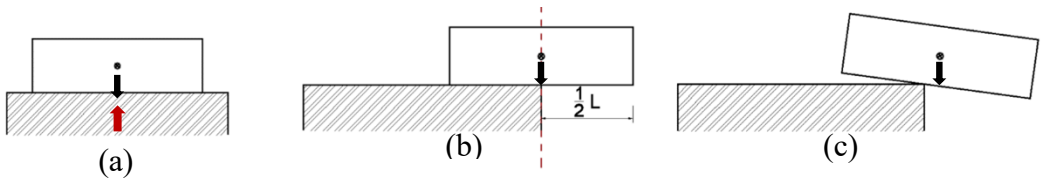


Figure 3.1: Center of mass and stability of a single brick above the base

Figure 3.1(a) shows a brick lying on the top of a firm base. Per Newton's first law of equilibrium, the reaction of weight of the brick is counterbalanced by the opposite reaction from the base. The brick is stable as long as the centroid of the brick is supported by the base.

However, when it is pushed to the edge of the base, it remains stable until it is projected half of its length. When it is projected beyond half of its length it loses its stability and starts to topple. This action is well illustrated in Figure 3.1. It is safe to project a brick if its center of mass does not go beyond the edge of the base. The main idea on projecting the brick lies on balancing the center of mass at the edge of the base support. This concept is used below to calculate the overhanging length of each unit when one brick is added above the other.

3.1.1.1 Overhanging two bricks

Figure 3.2 shows a structure of two bricks without any bonding and resting on the top of the firm base. The maximum projection for the upper brick is half of its length from the discussion above. A calculation can be done on how far the lower brick can be projected. Let l_0 be distance from the mass center of the lower brick to its right edge (which is half of its length), l_1 be the

overhanging length of topmost brick (which is also half of its length), and l_2 be the distance from the combined mass center to the right edge of the lower brick.

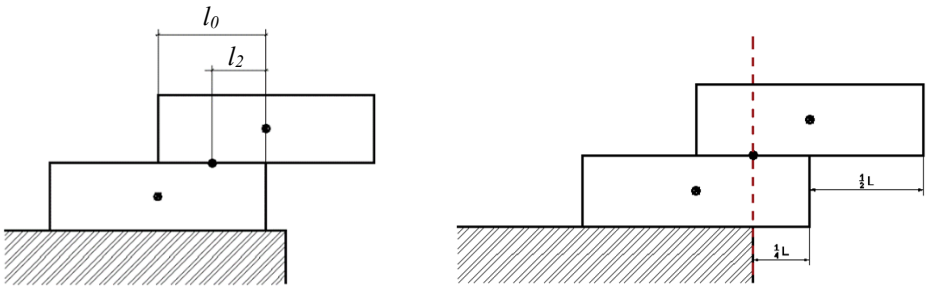


Figure 3.2: Equilibrium of two bricks when not bonded with mortar

The combined moment at the right edge of the lower brick equals the sum of the moment due to the weight of each brick.

$$\text{i.e., } (2w) \times l_2 = w \times 0 + w \times l_0 \tag{3-1a}$$

$$w \times \frac{1}{2} L = 2w \times l_2 \tag{3-1b}$$

$$l_2 = \frac{1}{4} L \tag{3-1c}$$

The l_2 gives the overhanging length of the lower brick. Hence, the combined maximum overhang becomes $\frac{3}{4}$ of the total length of a brick.

3.1.1.2 Overhanging three bricks

In Figure 3.3, let l_1 be the distance from the mass center of the lower brick to its right edge (which is half of its length) and l_3 be the distance between the combined center of mass and the right edge of the lower brick.

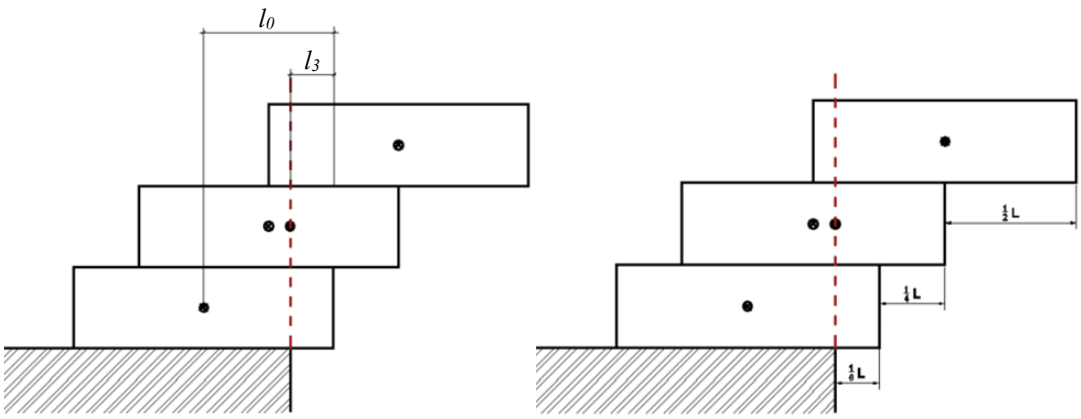


Figure 3.3: Equilibrium of three bricks when not bonded with mortar

The moment of the combined center of mass of the entire structure at the right edge of the lower brick is the sum of the moments due to each brick.

$$\text{Therefore, } (3w) \times l_3 = 2w \times 0 + w \times l_0 \tag{3-2a}$$

$$(3w) \times l_3 = w \times \frac{L}{2} \tag{3-2b}$$

$$l_3 = \frac{1}{6}L \tag{3-2c}$$

Here, l_3 gives overhang of the lower brick and the combined overhang becomes $\frac{11}{12}$ of the total length of the brick.

3.1.1.3 Overhanging four bricks

Figure 3.4 shows the similar situation as discussed in the previous sections for four bricks. Here, l_4 is the distance between the combined mass center and the right edge of the lower brick.

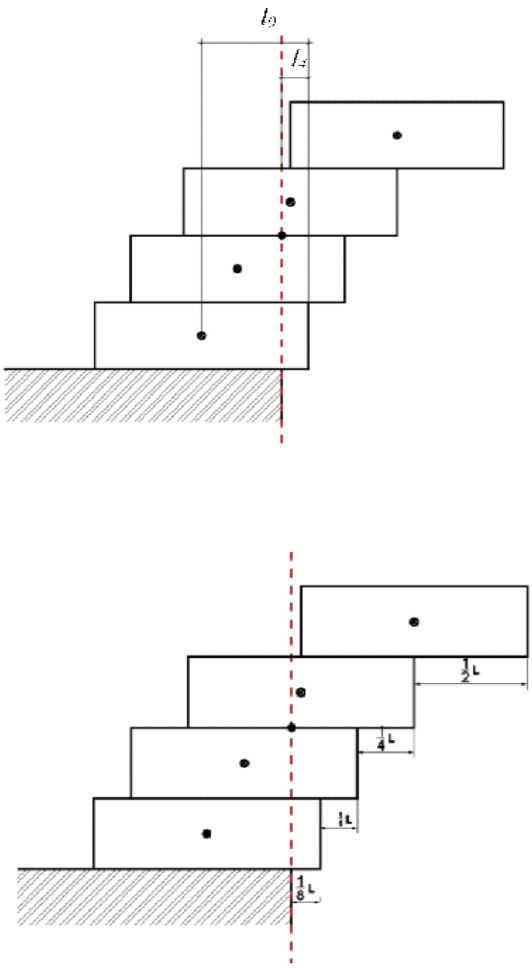


Figure 3.4: Equilibrium of four bricks when not bonded with mortar

Here, the combined moment at the right edge of the lower brick equals the sum of the moment due to each brick.

$$\text{Therefore, } (4w) \times l_4 = 3w \times 0 + w \times l_0 \tag{3-3a}$$

$$w \times \frac{1}{2} L = 4w \times l_4 \tag{3-3b}$$

$$l_4 = \frac{1}{8} L \tag{3-3c}$$

This gives the overhang of the lower brick and the combined overhang becomes $\frac{25}{24}$ of the total length of the brick.

3.1.1.4 Overhanging 'n' bricks

Similarly, mathematical derivation for the *'n'* number of bricks can be done. For the structure shown in Figure 3.5, the equilibrium condition becomes,

$$(nw) \times l_n = (n-1)w \times 0 + w \times l_0 \tag{3-4a}$$

$$(nw) \times l_n = w \times \frac{1}{2}L \tag{3-4b}$$

$$l_n = \frac{1}{2n}L \tag{3-4c}$$

So, when we go on adding *'n'* number of bricks one above the other, the combined overhang becomes $\left(\frac{1}{2}+\frac{1}{4}+\frac{1}{6}+\frac{1}{8}+\cdots+\frac{1}{2n}\right)$ times the length of the brick.

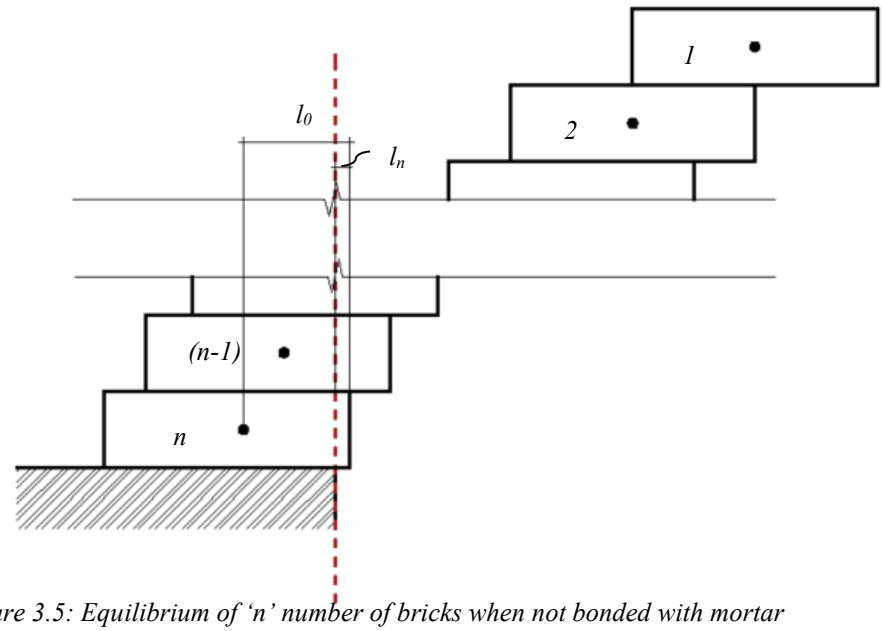
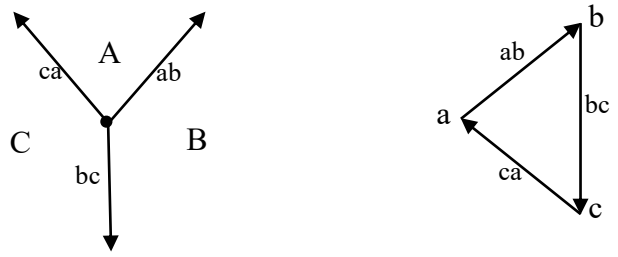


Figure 3.5: Equilibrium of 'n' number of bricks when not bonded with mortar

3.1.2 Validation of the mathematical derivation using graphical analysis

3.1.2.1 Graphical statics

The static forces in the system can be analyzed graphically. The graphical statics method of analysis requires the construction of a form diagram and a force diagram. The form diagram shows all the forces acting on the system. The force diagram is a diagram of the vectors of the forces which are drawn to scale.



(a) Form diagram (b) Force diagram

Figure 3.6: Graphical Statics

The form diagram is labelled by using Bow's Notation. In Bow's Notation, uppercase letters are placed in the space between the line of forces (as in Figure 3.6(a)) in a clockwise direction. The force between two letters is represented by lowercase letters. For instance, in Figure 3.6 (a) the force between the space A and B is represented by 'ab', the force between B and C is 'bc' and the force between C and A is 'ca'. Since the force diagram is drawn to scale, length of 'ab'; 'bc'; and 'ca' gives the magnitude of the respective forces in the force diagram Figure 3.6(b).

3.1.2.2 Validation

Figure 3.7 shows the analysis of the courses of bricks stacked following the algorithm discussed earlier in this chapter. The graphical analysis is done to locate the combined center of mass which lies within the length of the bottom brick. Figure 3.7 (a) shows a stack of 8 courses of bricks. Vertical lines are drawn from the mass center of each brick representing the weights.

Figure 3.7 (b) is the force diagram where '*w*' is the vertical force (i.e., weight of each brick) acting on the structure shown in Figure 3.7 (a). An imaginary pole 'O' is taken and each ends of '*w*' are connected to the pole 'O'. Figure 3.7 (c) is the form diagram. The lines in Figure 3.7 (b) are transferred in parallel fashion to Figure 3.7 (c). The first and last lines are extended until they intersect. The vertical line drawn from the intersecting point to Figure 3.7 (a) represents the combined mass center of the structure. This analysis validates the algorithm discussed in section 3.1.1.

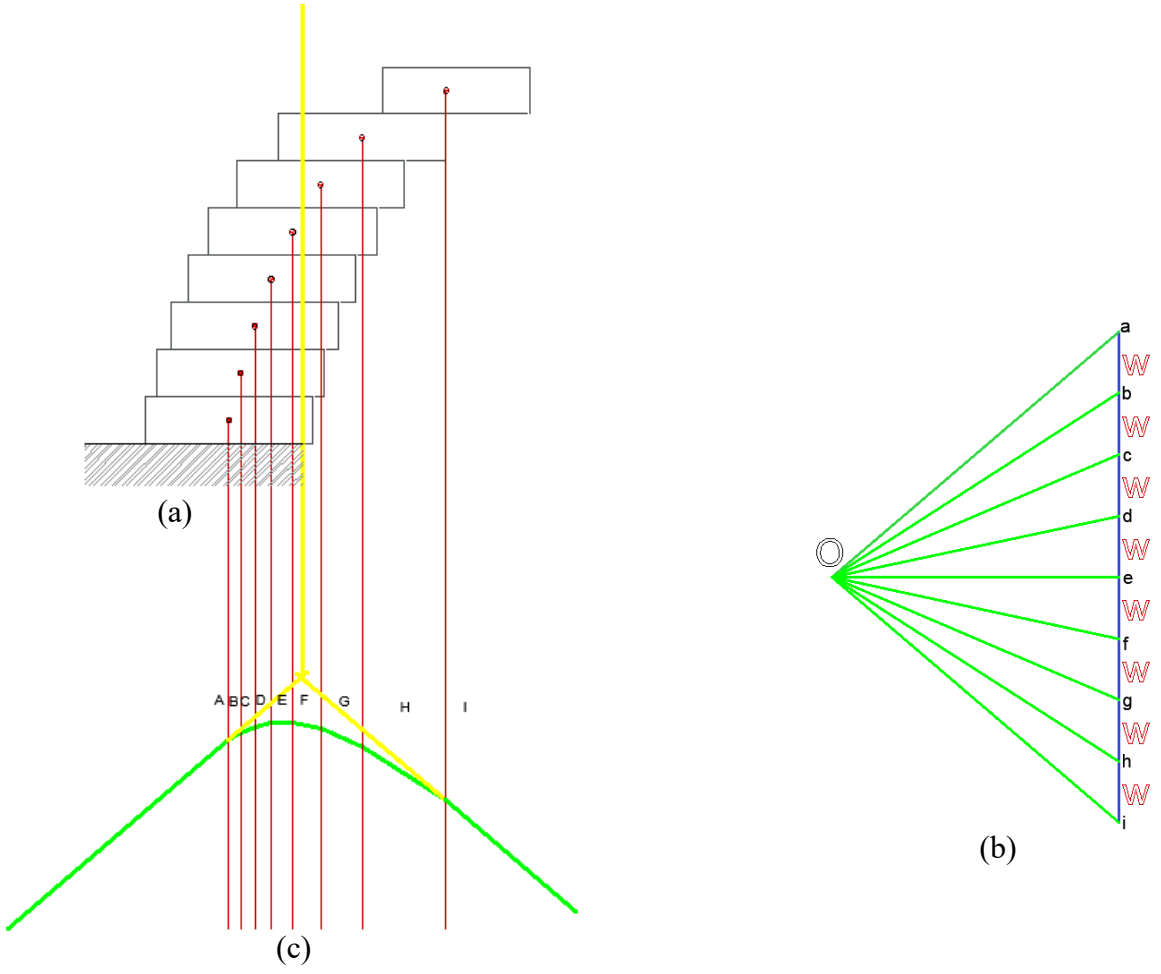


Figure 3. 7: Graphical statics to check the position of the combined center of mass of the structure

3.1.3 Overhanging bricks bonded with mortar

This section discusses the case where bricks are bonded with mortar. The mortar bond between bricks determines how far the brick can be projected from the edge. The bond strength depends on the type of the mortar used.

In Figure 3.8, '*F*' represents the bonding force due to the mortar and '*w*' represents the weight of the brick. Mathematically, F = bonding strength of the mortar(f') × contact surface area.

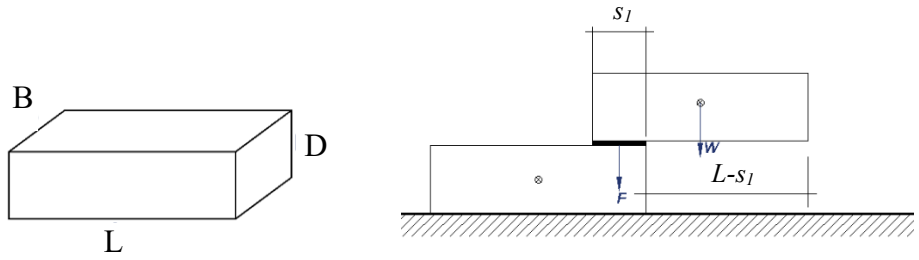


Figure 3.8: Overhanging one brick above another when bonded with mortar

3.1.3.1 Overhanging one brick

Assuming a perfect bonding between the lower brick and the support surface, let s_1 be the part of the brick length which is bonded with mortar (i.e. overlapping length). The overhanging length here becomes $L\text{-}s_1$.

Taking moment about the right edge of the base,

$$w\left(\frac{L-s_1}{L}\right)\left(\frac{L-s_1}{2}\right) = w\frac{s_1}{L}\left(\frac{s_1}{2}\right) + f's_1 B\left(\frac{s_1}{2}\right) \quad (3\text{-}5a)$$

$$w\frac{(L-s_1)^2}{2L} = w\frac{{s_1}^2}{2L} + f'B\frac{{s_1}^2}{2} \quad (3\text{-}5b)$$

$$f'B\frac{{s_1}^2}{2} + ws_1 - w\frac{L}{2} = 0 \quad (3\text{-}5c)$$

$$s_1 = \frac{-2w \pm \sqrt{(2w)^2 - 4f'B(-wL)}}{2f'B} \quad (3\text{-}5d)$$

for the equation (3-5 d) to be valid, the value $\sqrt{(2w)^2 - 4f'B(-wL)}$ must be positive. Therefore, the equation s_1 becomes,

$$s_1 = \frac{-w + \sqrt{w^2 + f'BwL}}{f'B} \tag{3-5e}$$

3.1.3.2 Overhanging two bricks

Figure 3.9 shows the situation with three bricks stacked where two bricks are overhanging above bottom the brick. Similar to Figure 3.8, the equilibrium equation is established in Eq. 3-6 (a-e).

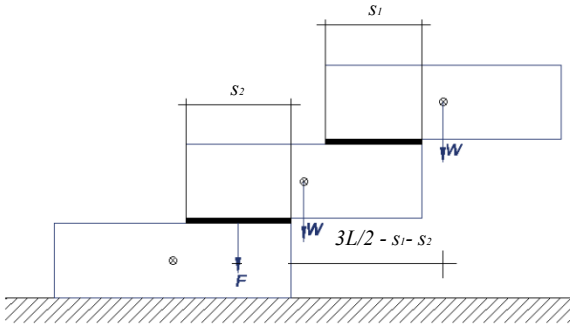


Figure 3.9: Overhanging two bricks when bonded with mortar

$$w\left[\frac{3L}{2} - s_1 - s_2\right] + w\left[\frac{(L-s_2)}{L}\right]\left[\frac{(L-s_2)}{2}\right] = w\frac{s_2}{L} \cdot \frac{s_2}{2} + f's_2\, B\left(\frac{s_2}{2}\right) \tag{3-6a}$$

$$\frac{3wL}{2} - ws_1 - ws_2 + w\left[\frac{(L-s_2)^2}{2L}\right] = w\frac{{s_2}^2}{2L} + f'B\frac{{s_2}^2}{2} \tag{3-6b}$$

$$f'B\,{s_2}^2 + 4ws_2 + 2ws_1 - 4wL = 0 \tag{3-6c}$$

Taking ($2ws_1$ - $4wL$) as a constant and ignoring negative value,

$$s_2 = \frac{-4w + \sqrt{(4w)^2 - 4f'B(2ws_1 - 4wL)}}{2f'B} \tag{3-6d}$$

$$s_2 = \frac{-2w + \sqrt{(2w)^2 + f'Bw(4L - 2s_1)}}{f'B} \tag{3-6e}$$

3.1.3.3 Overhanging three bricks

The equations 3-7a is an equilibrium condition for the structure shown in Figure 3.10. Derivation is done to solve the bonding length s_3.

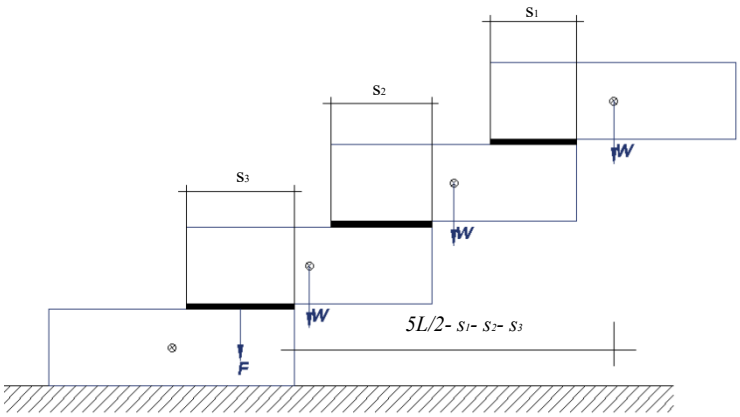


Figure 3.10: Overhanging three bricks when bonded with mortar

Equilibrium condition,

$$w\left[\frac{5L}{2} - s_1 - s_2 - s_3\right] + w\left[\frac{3L}{2} - s_2 - s_3\right] + w\left[\frac{(L-s_3)}{L}\right]\left[\frac{(L-s_3)}{2}\right] = w\frac{s_3}{L} \cdot \frac{s_3}{2} + f' s_3 B \left(\frac{s_3}{2}\right) \quad (3\text{-}7a)$$

$$\frac{5wL}{2} - ws_1 - ws_2 - ws_3 + - ws_2 - \frac{3wL}{2} ws_3 + w\left[\frac{(L-s_3)^2}{2L}\right] = w\frac{{s_3}^2}{2L} + f'B\ \frac{{s_3}^2}{2} \quad (3\text{-}7b)$$

$$f'B\ {s_3}^2 + 6ws_3 + 4ws_2 + 2ws_1 - 9wL = 0 \quad (3\text{-}7c)$$

Taking $(4ws_2 + 2ws_1 - 9wL)$ as a constant and ignoring negative value,

$$s_3 = \frac{-6w+\sqrt{(6w)^2-4f'B(4ws_2 + 2ws_1 - 9wL)}}{2f'B} \quad (3\text{-}7d)$$

$$s_3 = \frac{-3w+\sqrt{(3w)^2+f'Bw(9L-2s_1-4s_2)}}{f'B} \quad (3\text{-}7e)$$

3.1.3.4 Overhanging four bricks

Figure 3.11 shows a structure with four bricks overhanging above the lower brick. An equilibrium equation is established in Eq. 3-8a and solved for the bonding length s_4.

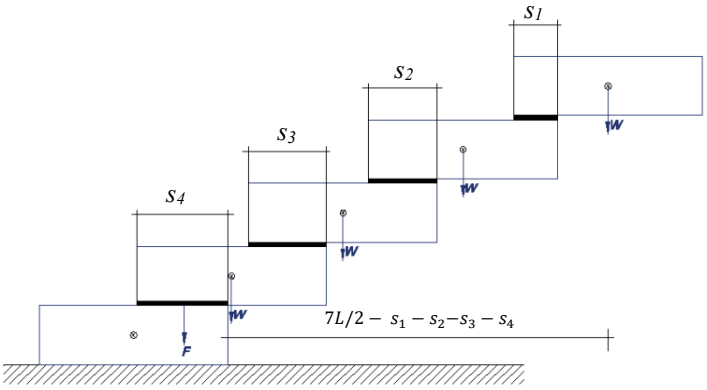


Figure 3.11: Overhanging four bricks when bonded with mortar

Equilibrium condition,

$$w\left[\frac{7L}{2} - s_1 - s_2 - s_3 - s_4\right] + w\left[\frac{5L}{2} - s_2 - s_3 - s_4\right] + w\left[\frac{3L}{2} - s_3 - s_4\right] + w\left[\frac{(L-s_4)}{L}\right]\left[\frac{(L-s_4)}{2}\right] = w\,\frac{s_4}{L}\,\left(\frac{s_4}{2}\right) + f'\,s_4\,B\,\left(\frac{s_4}{2}\right) \quad (3\text{-}8a)$$

$$\left[\frac{7wL}{2} - ws_1 - ws_2 - ws_3 - ws_4\right] + \left[\frac{5wL}{2} - ws_2 - ws_3 - ws_4\right] + \left[\frac{3wL}{2} - ws_3 - ws_4\right] + w\left[\frac{(L-s_4)^2}{2L}\right] = w\,\frac{{s_4}^2}{2L} + f'\,B\,\left(\frac{{s_4}^2}{2}\right) \quad (3\text{-}8b)$$

$$f'B\,{s_4}^2 + 8ws_4 + 2ws_1 + 4ws_2 + 6ws_3 - 16wL = 0 \quad (3\text{-}8c)$$

Taking $(2ws_1 + 4ws_2 + 6ws_3 - 16wL)$ as a constant and considering only positive value,

$$s_4 = \frac{-8w + \sqrt{(8w)^2 - 4f'B(2ws_1 + 4ws_2 + 6ws_3 - 16wL)}}{2f'B} \quad (3\text{-}8d)$$

$$s_4 = \frac{-4w + \sqrt{(4w)^2 + f'Bw(16L - 2s_1 - 4s_2 - 6s_3)}}{f'B} \quad (3\text{-}8e)$$

3.1.3.5 Overhanging five bricks

The equilibrium condition for Figure 3.12 is established in equation 3-9a. From the equation the length bonded with mortar for the fifth brick from the top is derived in the equation 3-9e.

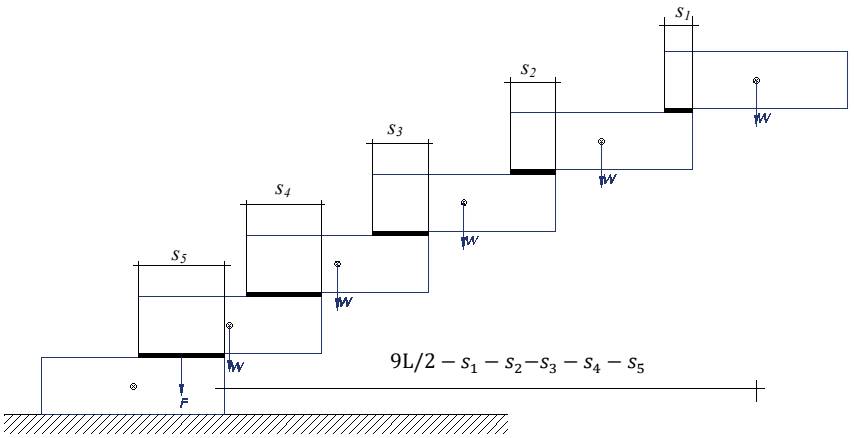


Figure 3.12: Overhanging five bricks when bonded with mortar

$$w\left[\frac{9L}{2} - s_1 - s_2 - s_3 - s_4 - s_5\right] + w\left[\frac{7L}{2} - s_2 - s_3 - s_4 - s_5\right] + w\left[\frac{5L}{2} - s_3 - s_4 - s_5\right] +$$

$$w\left[\frac{3L}{2} - s_4 - s_5\right] + w\left[\frac{(L-s_5)}{L}\right]\left[\frac{(L-s_5)}{2}\right] = w\ \frac{s_5}{L}\ \left(\frac{s_5}{2}\right)\ +\ f'\ s_5\ B\ \left(\frac{s_5}{2}\right) \quad \text{(3-9a)}$$

$$\left[\frac{9wL}{2} - ws_1 - ws_2 - ws_3 - ws_4 - ws_5\right] + \left[\frac{7wL}{2} - ws_2 - ws_3 - ws_4 - ws_5\right] + \left[\frac{5wL}{2} -\right.$$

$$\left. ws_3 - ws_4 - ws_5\right] + \left[\frac{3wL}{2} - ws_4 - ws_5\right] + w\left[\frac{(L-s_5)^2}{2L}\right] = w\ \frac{{s_5}^2}{2L} + f'\ B\left(\frac{{s_5}^2}{2}\right) \quad \text{(3-9b)}$$

$$f'B\ {s_5}^2\ +\ 10ws_5\ +\ 8ws_4\ +\ 2ws_1\ +\ 4ws_2\ +\ 6ws_3 - \ 25wL\ =\ 0 \quad \text{(3-9c)}$$

Taking $(8ws_4 + \ 2ws_1\ +\ 4ws_2\ +\ 6ws_3 - \ 25wL)$ as a constant and considering positive value only,

$$s_5 = \frac{-10w + \sqrt{(10w)^2 - 4f'B(8ws_4 + 2ws_1 + 4ws_2 + 6ws_3 - 25wL)}}{2f'B} \quad \text{(3-9d)}$$

$$s_5 = \frac{-5w + \sqrt{(5w - f'Bw(25L - 2s_1 - 4s_2 - 6s_3 - 8s_4)}}{2f'B} \quad \text{(3-9e)}$$

3.1.3.6 Overhanging 'n' bricks

Figure 3.13 shows the structure of *'n+1'* number of bricks with '*n'* bricks overhanging as shown. From the derivations above, equations can be generalized for the overlapping length of nth brick as,

$$s_n = \frac{-nw + \sqrt{(nw)^2 + f'Bw[n^2L - \sum_{i=1}^{n-1} 2is_i]}}{f'B} \tag{3-10}$$

Therefore, the overhanging length for *'nth'* brick becomes $L - s_n$.

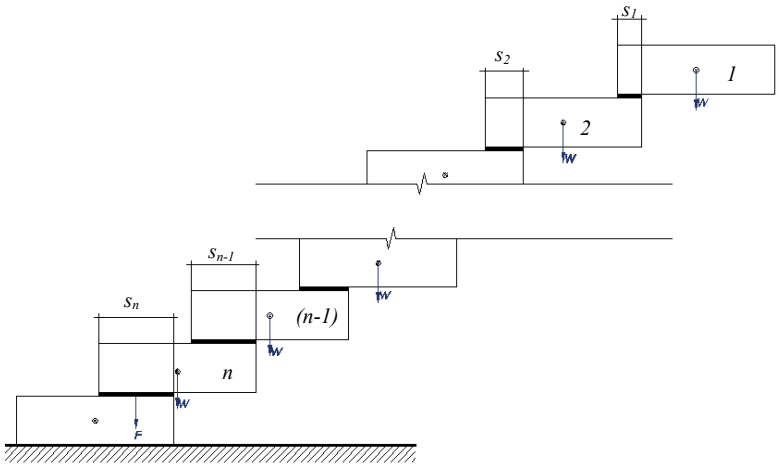


Figure 3.13: Overhanging 'n' bricks when bonded with mortar

3.2. Two-dimensional arch subjected to a uniformly distributed load (UDL) in the vertical direction

3.2.1 Overhanging bricks without bonding subjected to UDL on the top of the arch

A uniformly distributed vertical load (*w' lb. per inch*) is applied on the top of the bricks. The range of UDL starts from the left edge of the bottom brick to the right edge of the brick on the top. The equilibrium condition for this case is achieved by assuming that the resultant of the loads lies within the length of the bottom brick.

3.2.1.1 Overhanging two bricks

Figure 3.14 is a structure with two bricks without any bonding. The UDL is applied on the top of the structure as shown in Figure 3.14. The lowest brick is projected with the length l_2 from the edge of the firm base. The top brick is projected from the lowest brick with the length l_1. An equilibrium equation is established in Eq. 3.11a and solved for the overhanging length of the lower brick.

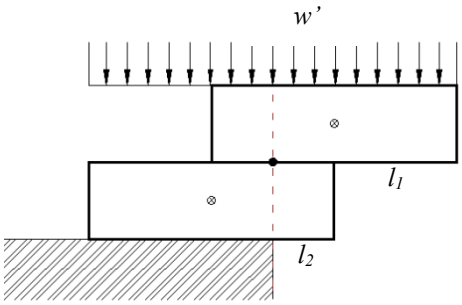


Figure 3.14: Overhanging two bricks when not bonded with mortar for additional load case

Equilibrium equation for two bricks,

$$\{2w + w' \times (L + l_1)\} \times l_2 = \{w + w' \times (L + l_1)\} \times 0 + w \times \frac{L}{2} \quad \text{(3-11a)}$$

$$(2w + w'(L + l_1)\,)\, l_2 = \frac{wL}{2} \quad \text{(3-11b)}$$

$$l_2 = \frac{wL}{2(2w+w'(L+l_1))} \tag{3-11c}$$

3.2.1.2 Overhanging three bricks

Similar to Figure 3.14, Figure 3.15 shows three bricks overhanging on the top of the firm base. The bricks are not bonded with mortar and are acted upon by the UDL. The derivation to achieve the overhanging length l_3 is shown in the equations 3-12(a-c).

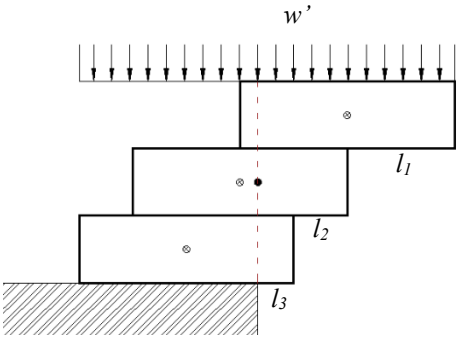


Figure 3.15: Overhanging three bricks when not bonded with mortar for additional load case

Equilibrium equation for three bricks,

$$\{3w + w' \times (L + l_1 + l_2)\} \times l_3 = \{2w + w' \times (L + l_1 + l_2)\} \times 0 + w \times \frac{L}{2} \tag{3-12a}$$

$$(3w + w'(L + l_1 + l_2)\ )\ l_3 = \frac{wL}{2} \tag{3-12b}$$

$$l_3 = \frac{wL}{2(3w+w'(L+l_1+l_2))} \tag{3-12c}$$

3.2.1.3 Overhanging four bricks

Figure 3.16 has four bricks and are not bonded with mortar. The mathematical derivation for the overhanging length of the lowest brick is shown in the equations 3-13 (a-c).

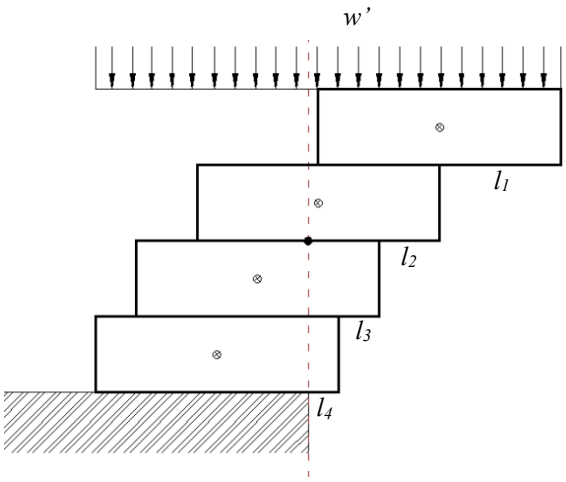


Figure 3.16: Overhanging four bricks when not bonded with mortar for additional load case

Equilibrium equation for four bricks,

$$\{(4w + w' \times (L + l_1 + l_2 + l_3)\} \times l_4 = \{3w + w' \times (L + l_1 + l_2 + l_3)\} \times 0 + w \times \frac{L}{2}$$

(3-13a)

$$(4w + w'(L + l_1 + l_2 + l_3)\ )\ l_4 = \frac{wL}{2} \quad \text{(3-13b)}$$

$$l_4 = \frac{wL}{2(4w + w'(L + l_1 + l_2 + l_3))} \quad \text{(3-13c)}$$

3.2.1.4 Overhanging n bricks

The *'n'* number of bricks without bonding are subjected to the additional loading as shown in Figure 3.17. The overhanging length for the n^{th} bricks is derived through equations 3-14 (a-c).

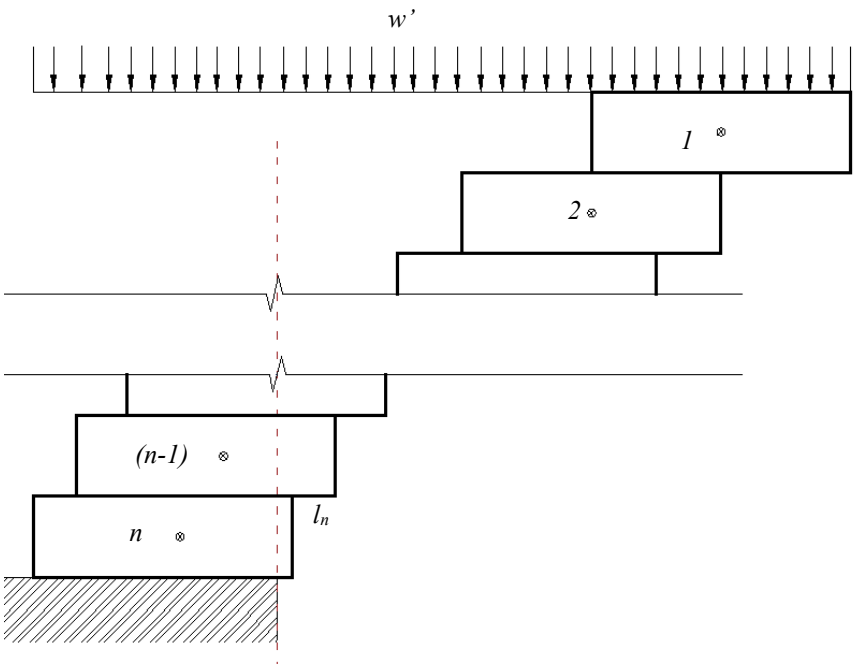


Figure 3.17: Equilibrium of 'n' number of bricks when not bonded with mortar with additional load case

The equilibrium equation is,

$$\{nw + w' \times (L + l_1 + l_2 + l_3 + \ldots + l_{n-1})\} \times l_n = \{(n-1)w + w' \times (L + l_1 + l_2 + l_3 + \ldots + l_n)\} \times 0 + w \times \frac{L}{2} \quad (3\text{-}14a)$$

$$(nw + w'(L + l_1 + l_2 + l_3 + \ldots + l_{n-1})) \, l_n = \frac{wL}{2} \quad (3\text{-}14b)$$

$$l_n = \frac{wL}{2[nw + w'\{L + (l_1 + l_2 + l_3 + \ldots + l_{n-1})\}]} \quad (3\text{-}14c)$$

3.2.2 Overhanging bricks with bonding subjected to UDL on the top of the arch

3.2.2.1 Overhanging one brick

Figure 3.18 shows the situation with two bricks bonded with mortar. Let s_1 be the length of the top brick bonded with mortar with the bottom brick. The additional load *w'* is acted on the top of the structure.

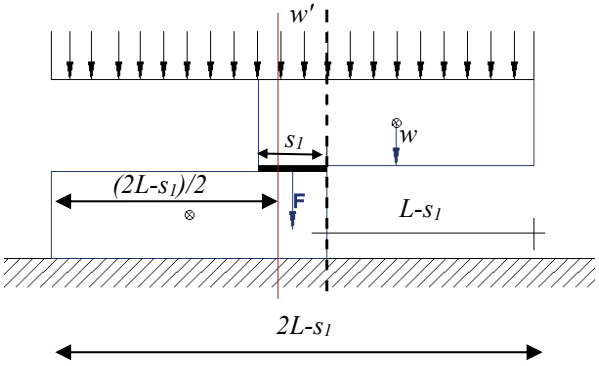


Figure 3.18: Overhanging one brick when bonded with mortar with additional load case

The equilibrium equation for this situation is,

$$w\left(\frac{L-s_1}{L}\right)\left(\frac{L-s_1}{2}\right) = w'(2L-s_1)\left(L-\frac{2L-s_1}{2}\right) + w \times \frac{{s_1}^2}{2L} + f' s_1\, B\, \left(\frac{s_1}{2}\right) \quad \text{(3-15a)}$$

$$\frac{wL}{2} - ws_1 + \frac{w{s_1}^2}{2L} = \frac{w{s_1}^2}{2L} + f'\, B\, \left(\frac{{s_1}^2}{2}\right) + w'Ls_1 - \frac{w'{s_1}^2}{2} \quad \text{(3-15b)}$$

$$\frac{wL}{2} - ws_1 = f'\, B\, \left(\frac{{s_1}^2}{2}\right) + w'Ls_1 - \frac{w'{s_1}^2}{2} \quad \text{(3-15c)}$$

$$(f'\, B - w'){s_1}^2 + (2w'L + 2w)s_1 - wL = 0 \quad \text{(3-15d)}$$

Taking only positive value,

$$s_1 = \frac{-(2Lw'+2w) + \sqrt{(2Lw'+2w)^2 - 4(f'B-w')\ (-wL)}}{2(f'B-w')} \quad \text{(3-15e)}$$

$$s_1 = \frac{-(w'L+w) + \sqrt{(w'L+w)^2 - (f'B-w')(-wL)}}{(f'B-w')} \quad \text{(3-15f)}$$

Overhanging length = L-s_1

3.2.2.2 Overhanging two bricks

Similarly, Figure 3.19 has two bricks above the bottom brick and are bonded with mortar. The overlapping length of the second brick with the bottom brick is derived through the equations 3-16 (a-d).

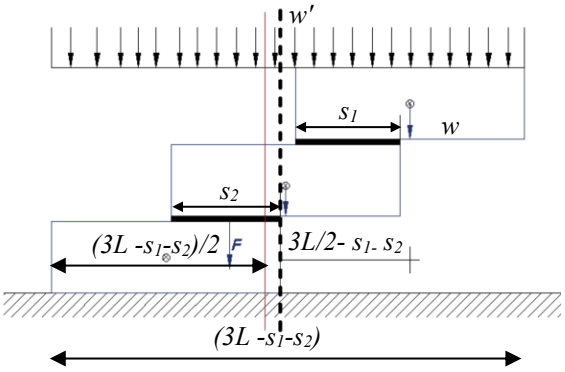


Figure 3.19: Overhanging two bricks when bonded with mortar with additional load case

Equilibrium equation,

$$w\left(L - s_2 + L - s_1 - \frac{L}{2}\right) + w\left(\frac{L-s_2}{L}\right)\left(\frac{L-s_2}{2}\right) = w'(3L - s_1 - s_2)\left(L - \frac{3L-s_1-s_2}{2}\right) + w \times \frac{{s_2}^2}{2L} + f's_2\, B\, \left(\frac{s_2}{2}\right) \quad \text{(3-16a)}$$

$$(f'B - w'){s_2}^2 + [2(2w'L - w's_1 + 2w)]s_2 + [w'(-3L^2 + 4Ls_1 - {s_1}^2) + 2w(s_1 - 2L)] = 0 \quad \text{(3-16b)}$$

Considering positive value only,

$$s_2 = \frac{-2(2w'L - w's_1 + 2w) + \sqrt{[2(2w'L - w's_1 + 2w)]^2 - 4(f'B - w')\,[w'(-3L^2 + 4Ls_1 - {s_1}^2) + 2w(s_1 - 2L)]}}{2(f'B - w')} \quad \text{(3-16c)}$$

$$s_2 = \frac{-[2(w'L + w) - w's_1] + \sqrt{[2(w'L + w) - w's_1]^2 - 4(f'B - w')\,[w'(-3L^2 + 4Ls_1 - {s_1}^2) + 2ws_1 - 4wL]}}{(f'B - w')} \quad \text{(3-16d)}$$

3.2.2.3 Overhanging three brick

In Figure 3.20, *w'* is acted on the top of the structure with four bricks where three bricks are projected above the bottom brick. Let the bonded length of third brick to the bottom be s_3. The s_3 is derived through the equations 3-17(a-e).

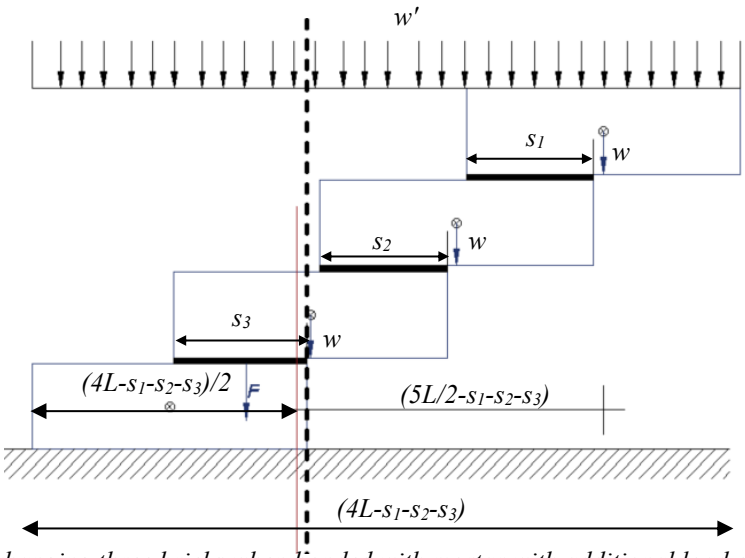


Figure 3.20: Overhanging three bricks when bonded with mortar with additional load case

Equilibrium equation,

$$w\left[\frac{5L}{2}-s_1-s_2-s_3\right]+w\left[\frac{3L}{2}-s_2-s_3\right]+w\left[\frac{(L-s_3)}{L}\right]\left[\frac{(L-s_3)}{2}\right]=w\frac{s_3}{L}\cdot\frac{s_3}{2}+f's_3B\left(\frac{s_3}{2}\right)+w'(4L-s_1-s_2-s_3)\left(L-\frac{4L-s_1-s_2-s_3}{2}\right) \quad (3\text{-}17a)$$

$$w\left[4L-s_1-2s_2-2s_3\right]+w\frac{(L^2-2Ls_3+{s_3}^2)}{2L}=\frac{w{s_3}^2}{2L}+f'B\left(\frac{{s_3}^2}{2}\right)+\frac{w'}{2}(-8L^2+6Ls_1+6Ls_2+6Ls_3-{s_1}^2-{s_2}^2-{s_3}^2-2s_1s_2-2s_1s_3-2s_2s_3) \quad (3\text{-}17b)$$

$$(f'B-w'){s_3}^2+[2(3w'L-w's_1-w's_2+3w)]s_3+[w'(-8L^2+6L(s_1+s_2)-(s_1+s_2)^2)+2w(s_1+2s_2)-9wL]=0 \quad (3\text{-}17c)$$

Considering positive value only the equation 3-17c becomes,

$$s_3=\frac{-[2(3w'L+3w)-2w'(s_1+s_2)]+\sqrt{[2(3w'L+3w)-2w'(s_1+s_2)]^2-4(f'B-w')\left[\begin{matrix}w'(-8L^2+6L(s_1+s_2)-(s_1+s_2)^2)\\+2w(s_1+2s_2)-9wL\end{matrix}\right]}}{2(f'B-w')} \quad (3\text{-}17d)$$

$$s_3=\frac{-[3(w'L+w)-w'(s_1+s_2)]+\sqrt{\begin{matrix}[3(w'L+w)-w'(s_1+s_2)]^2-(f'B-w')\ [w'(-8L^2+6L(s_1+s_2)-(s_1+s_2)^2)\\+\{2w(s_1+2s_2)-9wL\}]\end{matrix}}}{(f'B-w')} \quad (3\text{-}17e)$$

3.2.2.4 Overhanging 'n' bricks

The overlapping length of the n^{th} bricks is derived in the equation 3-18.

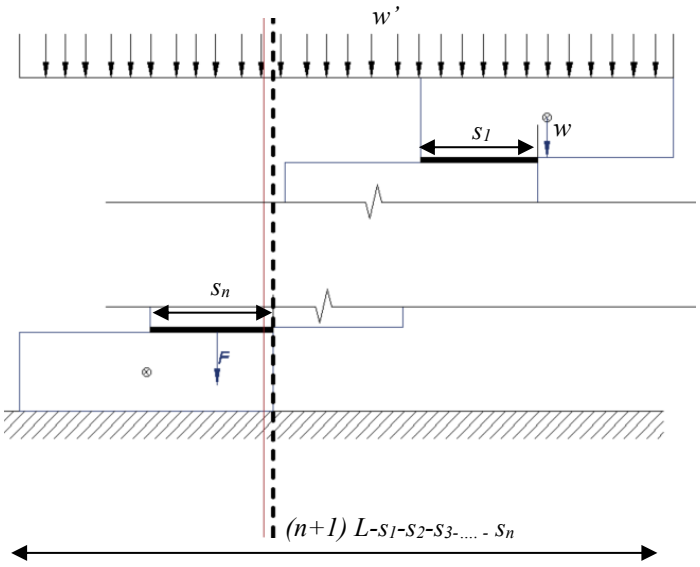


Figure 3.21: Overhanging 'n' bricks when bonded with mortar with additional load case

$$s_n = \frac{-[n(w'L+w)-w'\sum_{i=1}^{n-1} s_i]+\sqrt{[n(w'L+w)-w'\sum_{i=1}^{n-1} s_i]^2-(f'B-w')\ [w'(-(n^2-1)L^2+2nL\sum_{i=1}^{n-1} s_i-(\sum_{i=1}^{n-1} s_i)^2)+2w\sum_{i=1}^{n-1} s_i-n^2wL]}}{(f'B-w')} \quad (3\text{-}18)$$

3.3 Summary

In this chapter, the main idea to build masonry vaults without centering has been achieved through equilibrium analyses on various cases. The overhanging lengths of the bricks one above the other are derived so that curved structure can be formed for a half arch. A full arch is formed when the structure is mirrored. Using the arch developed by this techniques, different forms of vaults could be explored.

Considering the two-dimensional equilibrium approach, the overhanging length of each brick has been derived for the cases with and without bonding (each subjected or not subjected to the additional loading). The validation of the mathematical derivation has also been done using graphical method for the case without bonding and not subjected to the additional loading condition. The idea revealed in this chapter has been coded using Python script in Rhino for the arch construction which is discussed in Chapter 4.

CHAPTER 4: PYTHON SCRIPTING IN RHINOCEROS 3D AND ANALYSIS FOR TWO-DIMENSIONAL MASONRY ARCH

This chapter discusses the development and implementation of a computational design tool for form-finding of a masonry arch that can be built without centering. The algorithm developed in Chapter 3 is coded in the Python environment of Grasshopper in the Rhinoceros 3D modeling software. It also describes the code for generating arches for the case where bricks are stacked with and without bonding.

4.1 Rhinoceros and Python

Rhinoceros (Rhino) is a computer-aided design application software developed by David Rutten at Robert McNeel and Associates (McNeel, 2009). Python is one of the scripting languages supported by the Rhino (Tibbits et al., 2011). It is a component in the Grasshopper plugin which is a visual programming tool (Bachman, 2017). Rhino 6.0 and the Python have been used in this research to code the algorithm. Arches generated by coding the algorithm using the Python component are described in Section 4.2.

4.2 Python scripting for a two-dimensional arch

4.2.1 Two-dimensional arch with no bonding

The default dimension of the brick used in this research is 92mm × 57mm × 203mm (3-5/8” × 2-1/4” × 8”) and the default weight is 20.22 N (4.5 pounds). The dimension and weight could be adjusted to generate new arches in the code. The inputs given for coding in the Python component include the start point as ‘inPts’, length of brick, width of brick, depth of brick, and number of bricks in z-direction ‘N’.

The user interface of the design tool for a masonry arch with no mortar is shown in Figure 4.1. It shows the inputs and parameters given to the Python while coding.

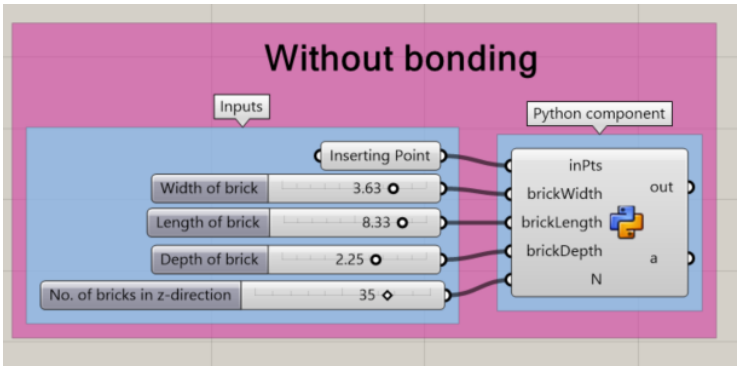


Figure 4.1: Image from grasshopper showing inputs parameters and python script component used when generating models

The coding process starts by creating a brick as shown in Figure 4.2. To create a brick, a definition is established which takes a start point location of the first brick and dimensions of the brick in the x, y, and z axes. A point **‘inPts’** is taken as a start point at which the first brick is placed.

```
#definition for a single brick
def createBrick (insertPt, dimX, dimY, dimZ):
    pt1 = [insertPt[0], insertPt[1] - dimY, insertPt[2]+dimZ]
    pt2 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]+dimZ]
    pt3 = [insertPt[0]+dimX, insertPt[1], insertPt[2]+dimZ]
    pt4 = [insertPt[0], insertPt[1], insertPt[2]+dimZ]
    pt5 = [insertPt[0], insertPt[1] - dimY, insertPt[2]]
    pt6 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]]
    pt7 = [insertPt[0]+dimX, insertPt[1], insertPt[2]]
    pt8 = [insertPt[0], insertPt[1], insertPt[2]]
    newbrick = rs.AddBox([pt1, pt2, pt3, pt4, pt5, pt6, pt7, pt8])
    return(newbrick)
```

Figure 4.2: Image of code showing definition to create a single brick

A list of bricks is created named 'brickList' where the bricks created are stored. By taking 'inPts' as the start point and 'brickLength', 'brickWidth', and 'brickDepth' as three dimensions, a brick is created and appended to the 'brickList' as shown in Figure 4.4. An 'offsetList' is created in the code where lists of overhanging lengths are stored. The overhanging length is denoted by 'x' and is defined by '[x = (1/(2×i) × brickLength]'. The 'i' here represents the range of bricks in Z-direction.

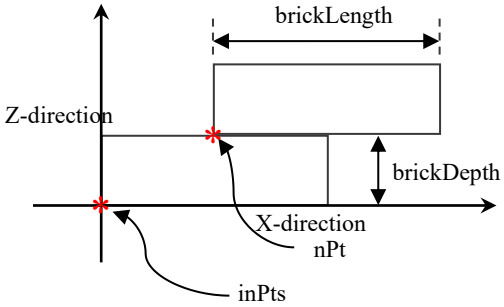


Figure 4.3: Image showing terminologies used while coding

```
#Section-1: method to create the arch on the left side of the structure
#creating first brick
#taking start point as inPts
startPts = inPts
newBrick = createBrick (startPts, brickLength, brickWidth, brickDepth)
brickList.append(newBrick)
#Method to stack bricks
for i in range(N-1,0,-1):
    x=(1/(2*i)*brickLength)
    offsetList.append(x)

for i in range(1,N):
    x = offsetList[i-1]
    print x
    nPt = rs.VectorAdd(inPts, [x,0,brickDepth])
    nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
    brickList.append(nBrick)
    inPts = nPt
```

Figure 4.4: Image of code showing method to generate left side of arch

A new point 'nPt' is created where the upper brick is to be located. To get 'nPt', vector addition is done between 'inPts and [x, 0, brickDepth]. By taking 'nPt', bricks are created and appended to the 'brickList'. The point 'inPts' is updated to 'nPt' which gives new positions for each brick to be stacked. The steps until now generate the left half section of the arch. To generate the right half section of the arch, 'nextStart' is taken as a new starting point to place a brick. Figure 4.5

explains how the coordinates for the 'nextStart' are defined. It is created by vector addition between 'startPts' and $[(inPts.X - startPts.X) \times 2 + brickLength, 0, 0]$. The 'inPts.X' gives the x-coordinate value of the point 'inPts' and startPts.X gives the x-coordinate value of the point 'startPts'.

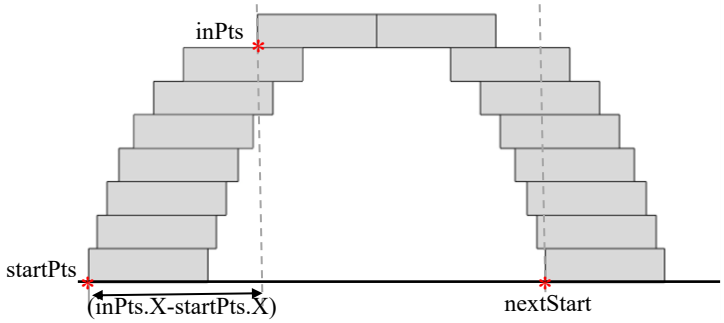


Figure 4.5: Determining a starting point for the right half of the arch

After getting the 'nextStart' point, brick is created and appended to the 'brickList'**.** Again, a new point 'nPt' is created where the upper brick is to be located. For getting 'nPt', vector addition is done between 'nextStart' and [-x, 0, brickDepth] which is shown in Figure 4.6. Taking 'nPt' as new insertion point, bricks are created and appended to the 'brickList' again. The 'nextStart' point is updated to 'nPt' so that the inserting point changes as we add courses of bricks above. Finally, the 'brickList' is printed.

```
#Section-2: method to create an arch on the right side of the structure
#nextStart point as a new starting point
nextStart = rs.VectorAdd(startPts, [(inPts.X-startPts.X)*2+brickLength,0,0])
newBrick = createBrick (nextStart, brickLength, brickWidth, brickDepth)
brickList.append(newBrick)

for i in range(1,N):
    x = offsetList[i-1]
    nPt = rs.VectorAdd(nextStart, [-x,0,brickDepth])
    nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
    brickList.append(nBrick)
    nextStart = nPt
```

Figure 4.6: Image of code showing a method to generate right side of arch

The procedure discussed above for coding the algorithm is summarized in the flow chart shown in Figure 4.7. An example of the arch with 35 courses of bricks (clear height = 1.93m (6.34 ft.) and span = 0.84m (2.75 ft.)) has been created using the algorithm which is included in Figure 4.8.

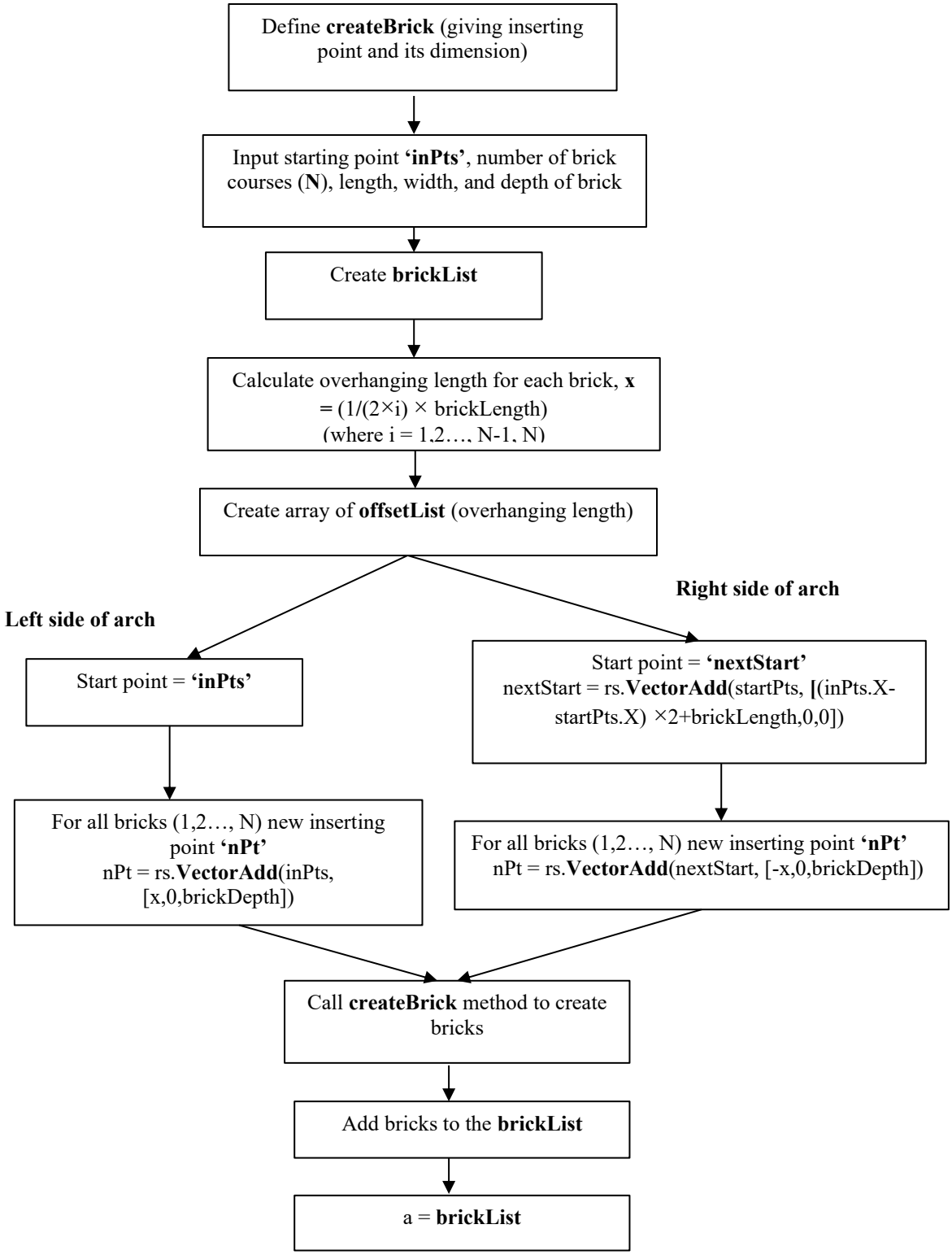


Figure 4.7: Flow chart of the code to generate an arch without bonding

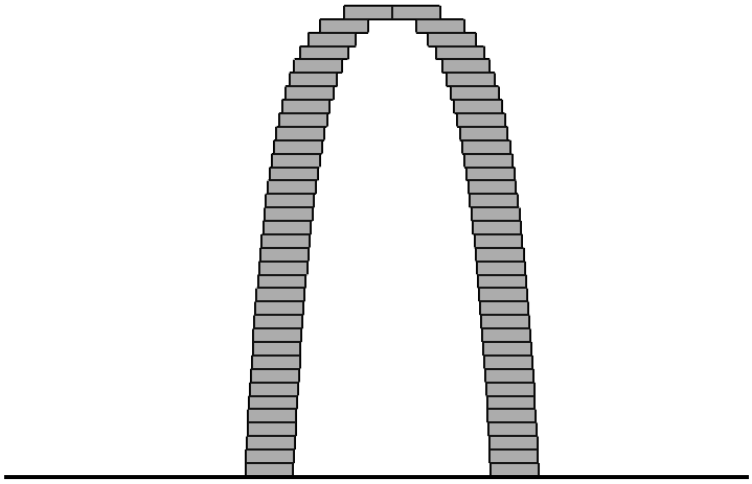


Figure 4.8: Example of an arch generated without bonding

4.2.2 Two-dimensional arch without bonding subjected to a uniformly distributed load

To generate an arch with no bonding and subjected to a uniformly distributed load same procedure taken to generate arch without load is followed. For this case, the UDL denoted by 'additWt' is added as an input shown in Figure 4.9. The definition for the overhanging length is updated according to the mathematical derivation in Section 3.2.1.

Figure 4.9 shows several inputs and parameters given to the python component. The values can be adjusted by sliding the number slider connected to the parameters. The inputs to provide the value of the UDL on the top of arch and the weight of the bricks have been added compared to that for the case without any additional load. The code in Figure 4.10 shows the definition of overhanging length where the mathematical derivation for this case has been calculated.

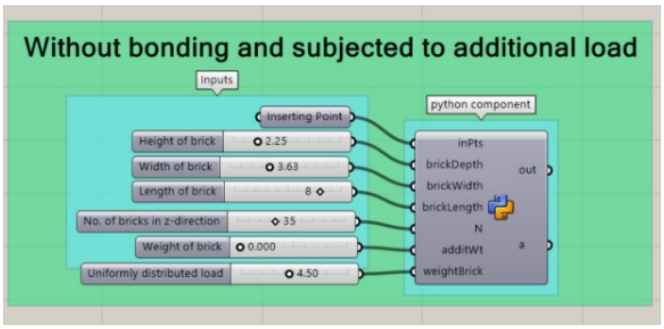


Figure 4.9: Image from grasshopper showing inputs used when generating models for the case with no bonding subjected to the additional load

```
#method to get bonding length of the bricks based on the numbricks using nth terms
#refer: thesis document for the equation
#definition for the overhanging length
def overhangingLength(N,weightBrick, additWt,brickLength):
    sum1=0
    for i in range(1,N-1):
        sum1 =sum1 +listofLength[i]
    sum2 = N*weightBrick+additWt*(brickLength+sum1)
    numerator = weightBrick*brickLength
    denominator = 2*sum2
    overlen= numerator/denominator
    listofLength.append(overlen)
    return overlen
```

Figure 4.10: Image of code showing definition for getting overhanging length

4.2.3 Two-dimensional arch with bonding

Figure 4.11 shows input parameters to design a two-dimensional arch with bonding including dimensions of brick, number of bricks, weight of brick, strength and thickness of mortar using for bonding.

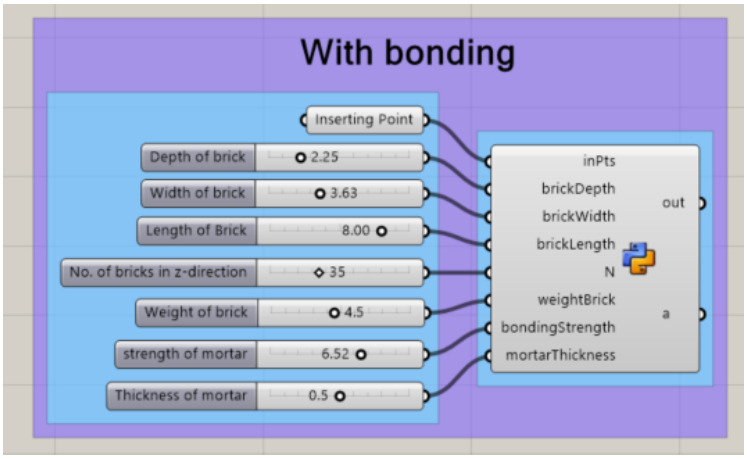


Figure 4.11: Image from grasshopper showing inputs parameters and python script component used when generating arch bonded with mortar

The coding steps for both cases are the same with additional inputs: bonding strength of mortar and mortar thickness. The strength of mortar for this research is taken from Table 2.4.

A definition to create brick is established which takes a start point location for the first brick and dimensions of brick in x, y, and z axes as shown in Figure 4.12. A definition for the bonding length 'BondingLength' of brick is established where the algorithm developed in Section 3.1.3 is coded (Figure 4.13). 'BondedList' is created where lists of lengths generated from the algorithm are stored. It is denoted by 'y' and is calculated using 'BondingLength' definition.

```
import rhinoscriptsyntax as rs
import math

#definition for a single brick
#creates the bricks at the position insertPt with dimension (dimX, dimY, dimZ)
#and returns the newBrick
def createBrick (insertPt, dimX, dimY, dimZ):
    pt1 = [insertPt[0], insertPt[1] - dimY, insertPt[2]+dimZ]
    pt2 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]+dimZ]
    pt3 = [insertPt[0]+dimX, insertPt[1], insertPt[2]+dimZ]
    pt4 = [insertPt[0], insertPt[1], insertPt[2]+dimZ]
    pt5 = [insertPt[0], insertPt[1] - dimY, insertPt[2]]
    pt6 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]]
    pt7 = [insertPt[0]+dimX, insertPt[1], insertPt[2]]
    pt8 = [insertPt[0], insertPt[1], insertPt[2]]
    newbrick = rs.AddBox([pt1, pt2, pt3, pt4, pt5, pt6, pt7, pt8])
    return(newbrick)
```

Figure 4.12: Image of code showing definition to create a single brick

```
listofLength = []
#definition of bonding length
def BondingLength(numbrick,weightBrick,bondingStrength,brickWidth,brickLength):
    sum=0
    for i in range(1,numbrick):
        sum = sum+2*i*listofLength[i-1]
    numerator=-numbrick*weightBrick+math.sqrt(math.pow(numbrick*weightBrick,2)
+bondingStrength*brickWidth*weightBrick*(numbrick*numbrick*brickLength- sum))
    denominator=bondingStrength*brickWidth
    bondingLen= numerator/denominator
    listofLength.append(bondingLen)
    return bondingLen
```

Figure 4.13: Image of code showing definition to get the bonded length

A list of bricks is created as 'brickList' where the bricks created are stored. A point 'inPts' is taken as a start point which is denoted as 'startPts' for the first brick. By taking 'inPts' as start point and 'brickLength', 'brickWidth', and 'brickDepth' as three dimensions, a brick is created and appended to the 'brickList' as shown in Figure 4.14.

```
#create a list of bonded length
bondedList=[]
for i in range(1,N):
    y = BondingLength(i,weightBrick,bondingStrength,brickWidth,brickLength)
    bondedList.append(y)
#Section-1: method to create the arch on the left side of the structure
#Method to stack bricks
for i in range(1,N):
    x = bondedList[(N-1)-i]
    nPt = rs.VectorAdd(inPts, [brickLength-x,0,brickDepth + mortarThickness])
    nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
    brickList.append(nBrick)
    inPts = nPt
```

Figure 4.14: Image of code showing method to generate left side of arch

A new point 'nPt' is created where the upper brick is located (Figure 4.15). For getting 'nPt', vector addition is done between 'inPts' and [brickLength-x, 0, brickDepth + mortarThickness]. Taking 'nPt' as a new insertion point bricks are created and appended to the 'brickList' again. Figure 4.15 shows the terminologies used while coding in Python.

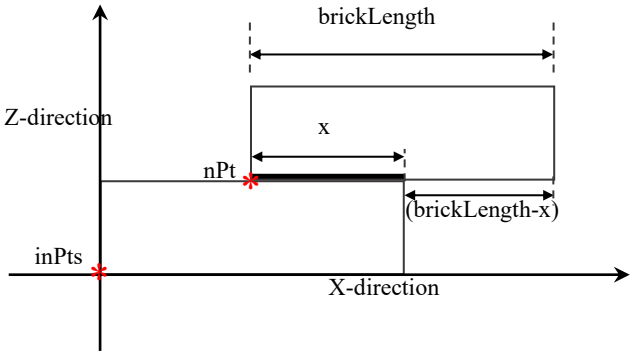


Figure 4.15: Image showing terminologies used while coding

The steps until now generates the left half section of the arch. To generate right half section of the arch, procedure is the same as for the one in Section 4.2.1. A point 'nextStart' is taken as a new starting point to insert a brick. It is created by vector addition between 'startPts' and $[(inPts.X - startPts.X) \times 2 + brickLength, 0, 0]$. The code to generate right side of the arch is shown in Figure 4.16.

```
#Section-2: method to create an arch on the right side of the structure
#nextStart point as a new starting point
nextStart = rs.VectorAdd(startPts, [(inPts.X-startPts.X)*2+brickLength,0,0])
newBrick = createBrick (nextStart, brickLength, brickWidth, brickDepth)
brickList.append(newBrick)
#Method to stack bricks
for i in range(1,N):
    x = bondedList[(N-1)-i]
    nPt = rs.VectorAdd(nextStart, [-(brickLength-x),0,brickDepth+mortarThickness])
    nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
    brickList.append(nBrick)
    nextStart = nPt
```

Figure 4.16: Image of code showing method to generate right side of arch

After getting 'nextStart' bricks are created using the definition established previously. The bricks created are then appended to the 'brickList'. Again, a new point 'nPt' is created where the upper brick is to be located. For getting 'nPt', vector addition is done between 'nextStart' and [-(brickLength-x), 0, brickDepth + mortarThickness]. Taking 'nPt' bricks are created and appended to the 'brickList' again. The 'nextStart' point is updated to 'nPt' so that the inserting point changes as we add courses of bricks above.

The procedure of coding to get an arch when bricks are bonded with mortar has been discussed above. The flowchart in Figure 4.18 also explains the procedure clearly. Figure 4.17 is an example of an arch generated for this case by coding the mathematical derivation. The arch shown in the figure is not subjected to an additional load. It also shows the terminologies used while coding. The arch in Figure 4.17 has 20 brick courses. It has a span of 2.16 m (7.1 ft). and height of 1.34 m (4.4 ft).

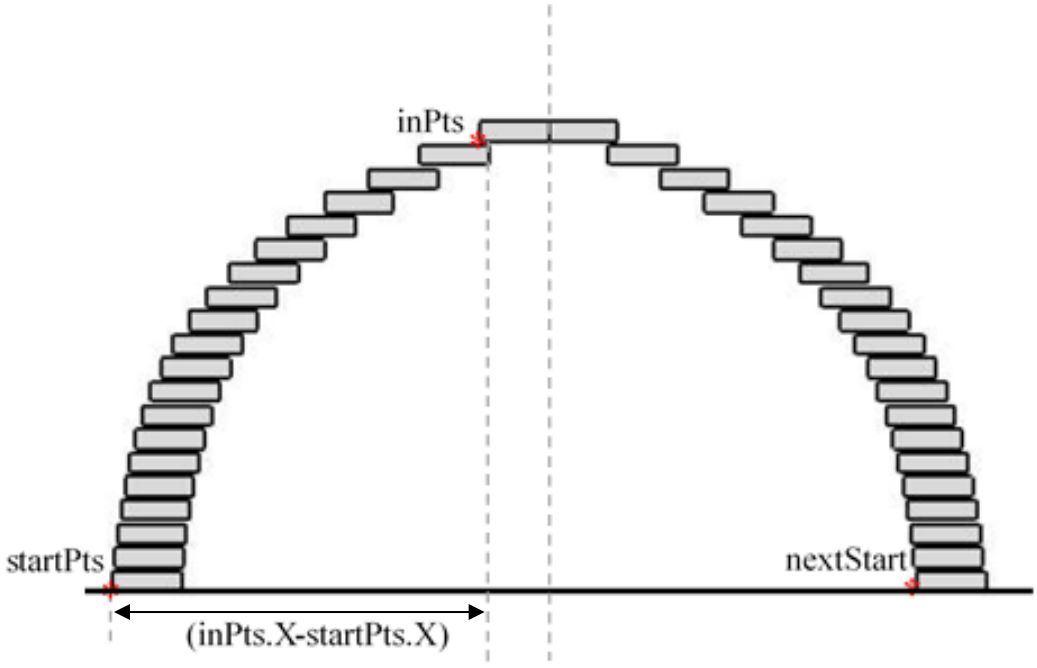


Figure 4.17: Example of an arch with 20 courses of bricks formed by coding the algorithm generated for the case where bricks are bonded with mortar

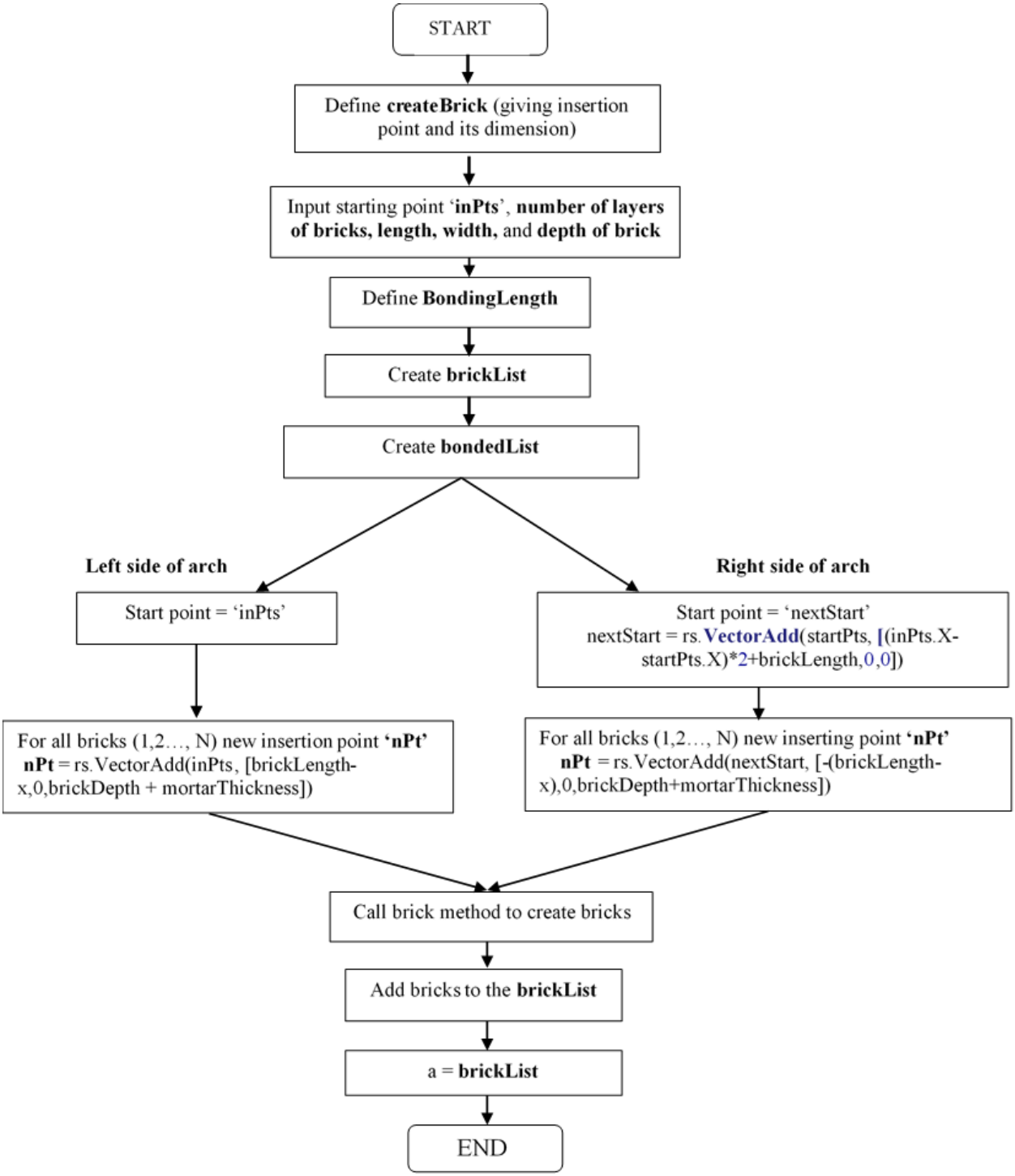


Figure 4.18: Flow chart of the code to generate an arch with bonding

4.2.4 Two-dimensional arch with bonding subjected to a UDL

To generate the arch with bonding and subjected to a UDL, a same procedure taken to generate the arch without load is followed. For this case, the definition for the bonding length is updated according to the mathematical derivation in Section 3.2.2 and is shown in Figure 4.20.

Figure 4.19 has additional inputs for the UDL acting on the top of the arch named 'additWt'. The number slider can be adjusted as required to give a value for the load.

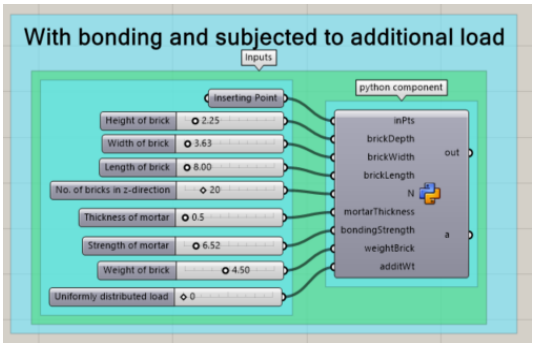


Figure 4.19: Image from grasshopper showing input parameters and python script used when generating models for bonded case with additional load case

```
#definition of bonding length
#method to get bonding length of the bricks based on the numbricks using nth terms
#refer: thesis document for the equation
def BondingLength(numbrick,weightBrick,bondingStrength,brickWidth,brickLength):
    print numbrick
    sum=0
    sum1a = additWt*brickLength+weightBrick
    sum1b = 0
    for i in range(1,numbrick):
        sum1b =sum1b+listofLength[i-1]
    sum1c = numbrick*numbrick-1
    sum1d = 0
    for i in range(1,numbrick):
        sum1d = sum1d+i*listofLength[i-1]
    sum1e =bondingStrength*brickWidth-additWt
    a = sum1e
    b = numbrick*sum1a-additWt*sum1b
    c = (additWt*((-1*sum1c*brickLength*brickLength)+2*brickLength*numbrick*sum1b-sum1b*sum1b)
    +2*weightBrick*sum1d-numbrick*numbrick*weightBrick*brickLength)
    numerator=-b+math.sqrt(b*b- a*c)
    denominator = a
    bondingLen= numerator/denominator
    print bondingLen
    listofLength.append(bondingLen)
    return bondingLen
```

Figure 4.20: Image of code showing definition to create bonding length

4.3 Analysis

4.3.1 Comparison of the arches with catenary and parabolic curves

The arches formed as a result by coding the algorithm developed in Chapter 3 are analyzed in this section. A curve is generated by connecting the centroid of each brick in the arch using straight line. Figure 4.21(a-c) shows the comparisons of the curves, for the case where bricks are not bonded, with catenary curve and the parabolic curve. Similarly, Figure 4.22 (a-c) shows the comparisons of the curves formed for the case with bonding. Both cases discussed here are not subjected to the additional vertical load.

The light blue dashed line represents the catenary curve. It is achieved using the *'catenary curve'* command in Rhino. The dark blue dashed-dot line represents the parabolic curve which is achieved by using the '*parabola form 3-points'* command in Rhino.

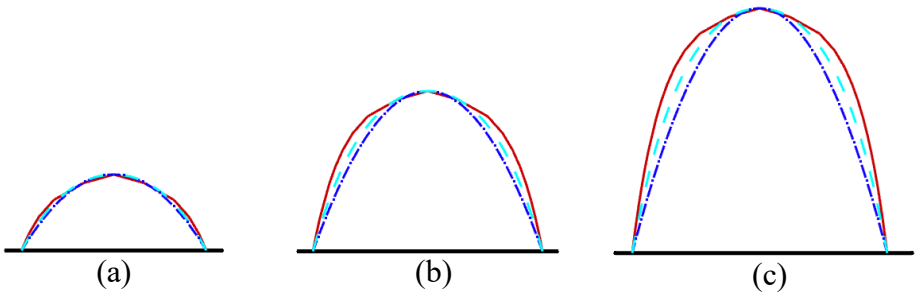


Figure 4.21: Comparison of the arch formed without bonding with catenary curve and parabolic curve

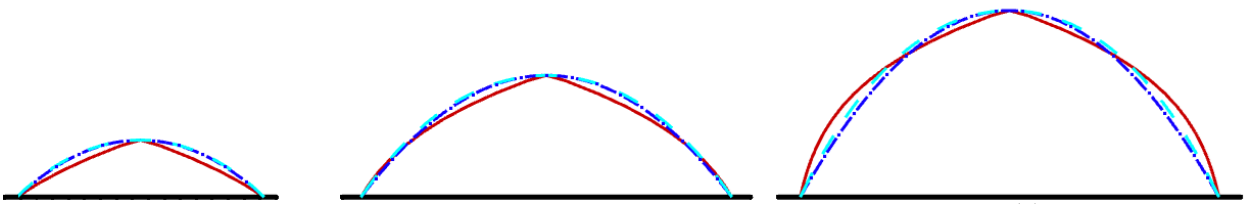


Figure 4.22: Comparison of the arch formed with bonding with catenary curve and parabolic curve

Figures 4.21(a-c) and 4.22(a-c) have different number of brick courses. The figure (a) has 5 courses, figure (b) has 10 courses and figure (c) has 15 courses for both Figures 4.21 and 4.22. The height to span ratio for each figure has been listed in Table 4.1.

Table 4.1: Height to span ratios of the curves in Figures 4.21(a-c) and 4.22(a-c)

	5 courses	10 courses	15 courses
Figure 4.21	0.41	0.70	0.96
Figure 4.22	0.23	0.33	0.45

From the figures, it is seen that with the increase in height to span ratios the curves gradually deviate from the catenary and parabolic curves.

4.3.2 Spans of arches

4.3.2.1 Analyzing spans of arches having mortar with different bonding strengths

Arches are analyzed by using mortar with different bonding strengths. In Figure 4.23, the dark blue arch is bonded with lime/sand mortar in 1:8 proportion. The white arch is bonded with cement/sand mortar in 1:8 proportion. The red arch is bonded with lime/ sand mortar in 1:3

proportion. The green arch is bonded with cement/ sand mortar in 1:3 proportion. From the figure, it is seen that the span of arch increases with increase in strength of bonding. The arch generated using mortar with low bonding strength has a shorter span. The spans for these arches are presented in Table 4.2.

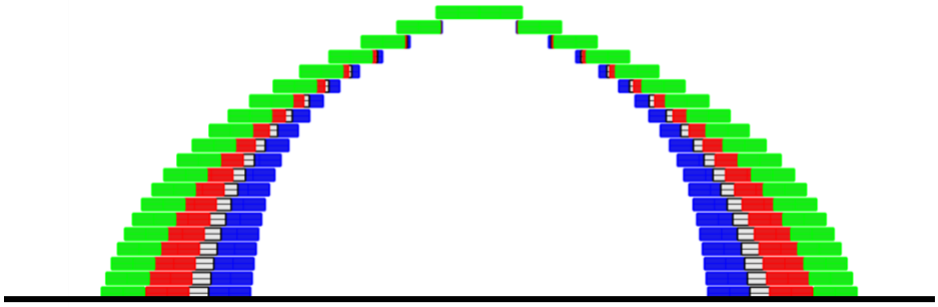


Figure 4.23: Arches with same number of brick courses but different bonding strength

Table 4.2: Influence of strength of mortar in the span of arches

Type of mortar	Average bond strength(psi)	Span (inches)
1:8 Lime /Sand	6.52	85.19
1:8 Cement /Sand	11.75	100.85
1:3 Lime /Sand	9.86	108.23
1:3 Cement /Sand	16.96	124.76

4.3.2.2 Analyzing spans of arches with different loading conditions

Figure 4.24 below shows how the addition of external vertical load impacts the span of the arches. Figure 4.24 (a) is the case without bonding and Figure 4.24 (b) is the case where bricks are bonded with mortar. The three arches shown in the figure have same number of brick courses (i.e. 20) and are subjected to three different external loadings acting on the top i.e., 0 N/m, 875.6

N/m, and 1751.3 N/m (0 lbs/in, 5 lbs/in and 10 lbs/in), respectively. When the external load is increased the span of the arch decreases. In Figure 4.24, the arch with longest span is subjected to 0 lbs/in load and the one with shortest span is subjected to the load of 10 lbs/in. This information is presented well in Table 4.3.

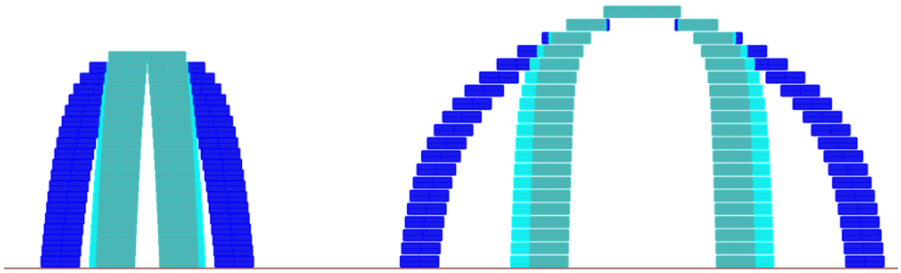


Figure 4.24: Arches with same courses of bricks but different loading case (a) arches with no bonding (b) arches with bonding

Table 4.3: Influence of additional vertical load in the span of arches

Additional Load (lbs/in)	Without bonding	With bonding
	Span (inches)	Span(inches)
0	28.38	85.19
5	8.09	38.99
10	5.26	31.12

4.3.3 Heights of arches

4.3.3.1 Height determined from the compressive strength of bottom brick

The capacity of the bottom brick to hold the load of all the bricks of an arch depends on its compressive strength. Mathematically, the compressive strength of a brick is given by the maximum load at failure(N) divided by the average area of bed face (mm^2). Considering the

brick as first-class bricks and taking compressive strength to be 19.6 N/mm^2 (Section 2.1.1), the maximum capacity of the bottom brick is calculated and height for the arch is derived.

Therefore, the maximum load at failure(N) = Compressive strength of a brick × average area of bed face (mm^2) = $19.6N/mm^2 \times 203mm \times 92mm = 366{,}049.6\ N = 82291\ lbs.$

To get the number of bricks that the bottom brick can hold, the maximum load at failure is divided by the weight of a single brick in the case considering self-weight only,

i.e., $82{,}291 / 4.5 \approx 18{,}286\ courses\ of\ bricks$

So, theoretically the maximum height of the arch should be 18286 × height of a single brick. i.e., 18,286 × 2.25 = 41,143.5 inches.

Therefore, clear height of the arch = 41,143.5 – height of the topmost brick

= 41143.5 – 2.25 = 41141.25 inches = 3,428.4 ft.

Apparently, the height is not realistic even considering the factor of safety, and the height should not be determined from the compressive strength of the bricks.

4.3.3.2 Height determined from the lateral stability for the cases without loading

For lateral stability, the minimum thickness of a masonry structure is determined by the ratio of height of the structure to the thickness. (International Building Code. International Code Council, 2006.) For the non-load bearing external wall the ratio of height to the thickness of structure is 18. When the arch is not subjected to the external loading, this ratio can be taken to calculate the height of the arch generated in this research.

$$i.e., \quad Height\ of\ arch/\ thickness\ of\ structure\ = 18$$

Therefore, $height\ of\ arch\ = (18 \times 8)\ inches = 12\ ft.$

(Here, the length of the brick is taken as 8 inches)

So, for the arch with a height of 12 ft.,

$$number\ of\ brick\ courses\ (for\ the\ \ case\ without\ bonding) = \frac{(12 \times 12)}{2.25}$$

$$= 64$$

For the case bonded with mortar of half inch thickness,

Number of brick courses = $\frac{(12 \times 12)}{(2.25+0.5)} = 52$

For the case with external loading

For load bearing external wall, the ratio of height to the thickness of the wall is 20 when the solid units are used.

Therefore for this research study, $Height\ of\ arch/\ thickness\ of\ arch = 20$ as solid bricks are used.

Mathematically, $Height\ of\ arch = (20 \times 8) inches = 13.33\ ft.$

(Thickness of the arch equals the length of the bricks = 8 inches)

So, for the case without bonding, the number of brick courses = $\frac{13.33 \times 12}{2.25}$

= 71

So, for the case with bonding, the number of brick courses = $\frac{13.33 \times 12}{(2.25+0.5)}$

= 58

4.3.3.3 Height to Span ratios

In this section, the arches are analyzed based on the height to span ratios. For both cases, arches with different brick courses (10, 20, 30, 40, 50) are generated. The span and their respective

heights are measured. From the data, the height/span ratio is calculated and shown in Table 4.4. Figure 4.25 shows a graph of the height/span ratio for both cases. Table 4.5 shows the height/span ratio for the arches when subjected to different loading condition for the case without bonding and Table 4.6 is for the case with bonding and subjected to different loading conditions. Respectively, graphs are plotted for the data in both tables in Figures 4.26 and 4.27.

Table 4.4: Height/Span ratio for both cases without additional loading

Number of brick courses	10	20	30	40	50
With no bonding	0.89	1.51	2.06	2.58	3.08
with bonding	0.35	0.61	0.90	1.17	1.44

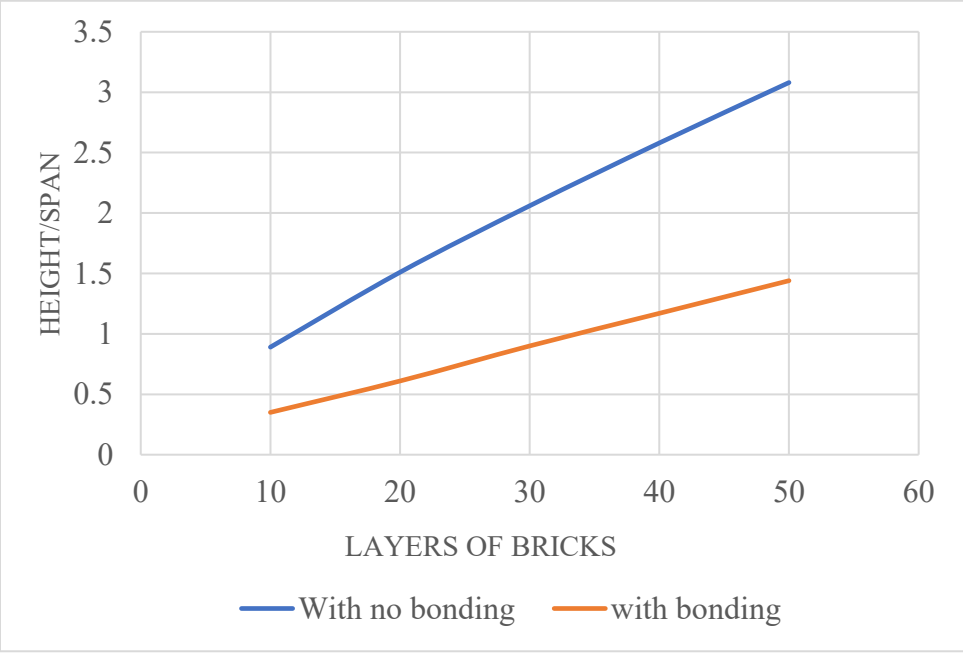


Figure 4.25: Graph showing height/span ratio for the case bonded with mortar and not bonded case without loading

Table 4.5: Height/Span ratio for the case without bonding and subjected to UDL

Additional loading(lbs/in) \ No. of courses	10	20	30	40	50
0	0.89	1.51	2.06	2.58	3.08
5	4.00	5.28	6.40	7.42	8.38
10	6.62	8.13	9.44	10.65	11.79

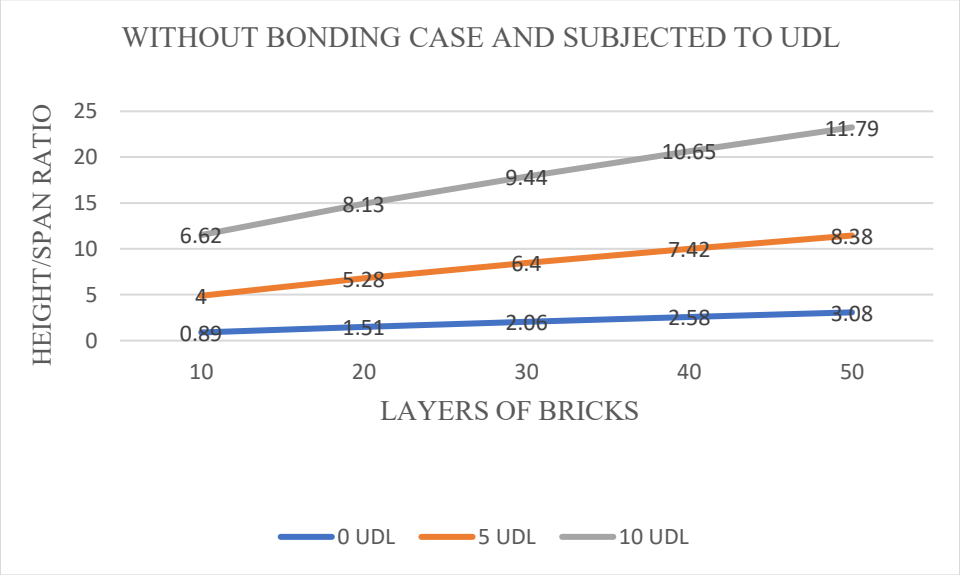


Figure 4.26: Graph showing height/span ratio for the case without bonding and subjected to different loading condition

Table 4.6: Height/Span ratio for bonding case when subjected to UDL

Additional loading (lbs/in) \ No. of courses	10	20	30	40	50
0	0.35	0.61	0.90	1.17	1.44
5	0.68	1.34	1.96	2.54	3.11
10	0.84	1.68	2.46	3.19	3.90

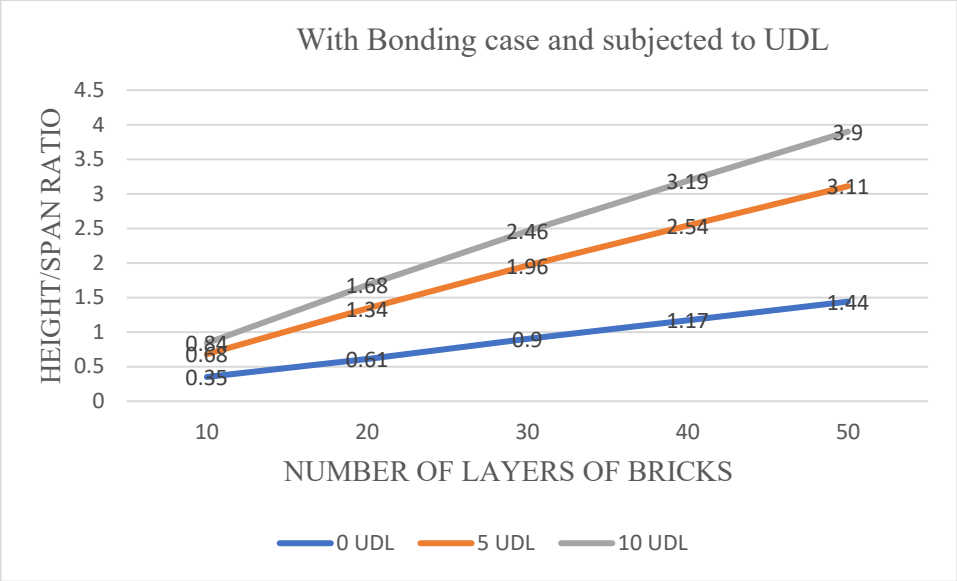


Figure 4.27: Graph showing height/span ratio for the case with bonding and subjected to different loading conditions

4.4 Discussions

4.4.1 Course by course stability

In this section, the stability of the overhanging bricks when bonded with mortar is discussed. The stability check is done on two examples with two bricks and three bricks, respectively.

Two bricks bonded with mortar

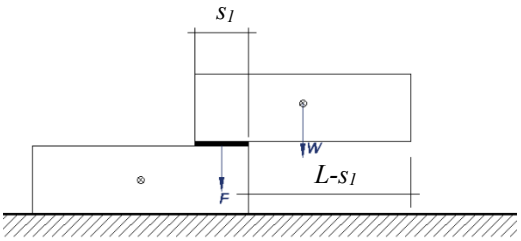


Figure 4.28: Stability check for overhanging one brick above other & bonded with mortar

The theoretical length of a brick bonded with mortar is given by the equation,

$$s_1 = \frac{-w + \sqrt{w^2 + f'BwL}}{f'B} \text{ (from Equation 3-5e)}$$

Taking the values of w = 4.5 pounds, f'= 6.52 psi, B= 3.63 inches, L = 8 inches,

$$s_1 = \frac{-4.5 + \sqrt{4.5^2 + 6.52 \times 3.63 \times 4.5 \times 8}}{6.52 \times 3.63}$$

$$= 1.06 \text{ inches}$$

For accuracy in real field construction,

Let $s_1 = 1\ 1/8\ inches$ = 1.125 inches

$$\text{Overturning moment} = w\left(\frac{L - s_1}{L}\right)\left(\frac{L - s_1}{2}\right) \quad (4\text{-}1a)$$

$$= 4.5\left(\frac{8 - 1.125}{8}\right)\left(\frac{8 - 1.125}{2}\right)$$

$$= 13.29\text{-pound inch}$$

$$\text{Resisting moment} = F \times \frac{s_1}{2} + w.\frac{{s_1}^2}{2L} \quad (4\text{-}1b)$$

$$= f' \times B \times \frac{s_1}{2} + w \times \frac{{s_1}^2}{2L}$$

$$= 6.52 \times 3.63 \times \frac{1.125}{2} + 4.5 \times \frac{1.125^2}{2 \times 8}$$

$$= 13.67\text{-pound inch}$$

The resisting moment is larger than the overturning moment.

Three bricks bonded with mortar

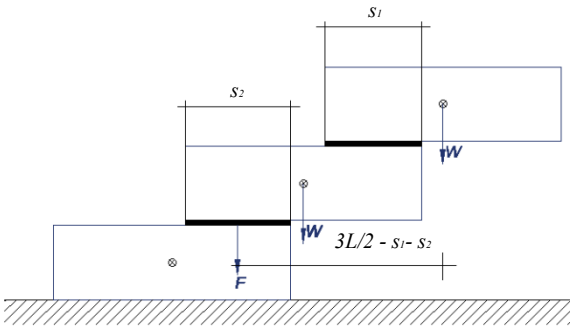


Figure 4.29: Stability check for overhanging two bricks bonded with mortar

The theoretical length of a brick bonded with mortar is given by the equation,

$$s_2 = \frac{-2w + \sqrt{(2w)^2 + f'Bw(4L - 2s_1)}}{f'B} \text{ (from Equation 3-6e)}$$

Taking the values of w = 4.5 pounds, f'= 6.52 psi, B= 3.63 inches, L = 8 inches,

$$s_2 = \frac{-2\times4.5 + \sqrt{(2\times4.5)^2 + 6.52\times3.63\times4.5(4\times8-2\times1.125)}}{6.52\times3.63} = 2.03 \text{ inches}$$

Let $s_2 = 2\ 1/8\ inches = 2.125$ inches

$$\text{Overturning moment} = w\left[\frac{3L}{2} - s_1 - s_2\right] + w\left[\frac{(L-s_2)}{L}\right]\left[\frac{(L-s_2)}{2}\right] \quad (4\text{-}2a)$$

$$= 4.5\left[\frac{3\times8}{2} - 1.125 - 2.125\right] + 4.5\left[\frac{(8-2.125)}{8}\right]\left[\frac{(8-2.125)}{2}\right]$$

$$= 4.5\times8.75 + 4.5 \times 2.157$$

$$= 49.0825\text{-pound inch}$$

$$\text{Resisting moment} = w \times \frac{s_2}{L} \times \frac{s_2}{2} + f' \times s_2 \times B \times \left(\frac{s_2}{2}\right) \quad (4\text{-}2b)$$

$$= 4.5\frac{2.125}{8} \times \frac{2.125}{2} + 6.52 \times 2.125 \times 3.63 \times \left(\frac{2.125}{2}\right)$$

$$= 1.27 + 53.437$$

$$= 54.707\text{-pound inch}$$

The resisting moment is larger than the overturning moment. Theoretical length of the bonding length for each course will be adjusted and can be scripted in the program for the real-world applications. In both examples discussed here, the resisting moment is greater than overturning moment. This proves that the structure is stable during construction.

4.4.2 Safety factor

This research determines the theoretical distance of overhanging bricks to its maximum limit from the equilibrium conditions. A factor of safety should be considered during design and construction for the real-world applications. A reduced length of overhang from the theoretical values or a reduced input bonding strength of mortar in the program could increase the factor of safety for the design.

4.4.3 Comparison with the back-weighted construction

The discussion below is to show the back-weighted construction generates a much higher compressive strength at the bottom which may not be a good option compared to the proposed stacking strategies proposed in the thesis.

Mathematically, compressive stress for the brick at the toe of the cantilever for a typical arch is,

$$\frac{weight\ of\ single\ brick\ \times\ No.of\ bricks\ above}{contact\ surface\ area\ of\ bottom\ brick}$$

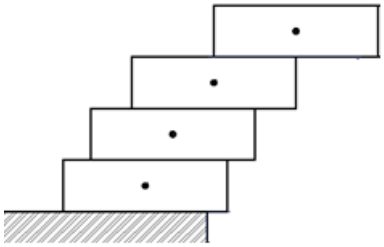


Figure 4.30: Stack of bricks with no backweight

For the structure shown in Figure 4.30, compressive stress on the bottom brick = $\frac{w \times 3}{L \times B}$

Similarly, in the back-weighted construction, the overall weight on the bottom brick is much heavier. Figure 4.31 shows the compressive stress at the bottom brick is three times as the previous example for a four-course arch construction.

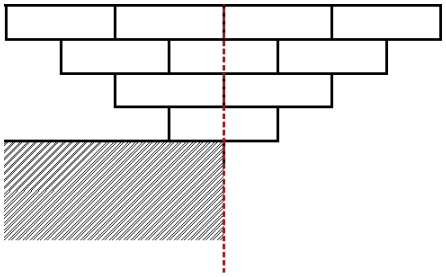


Figure 4.31: Back weighted bricks

The quantity of bricks needed to build an arch in the back-weighted condition is far more than the arch generated from this thesis. It is clearly explained from Figure 4.32. It was not the intention of the research to propose an inefficient strategy in the masonry arch and vault design and constructions. The spanning limit of the arch could be determined considering the overall stability of the arch and strength capacity of the masonry materials, but it is out of the scope of the thesis.

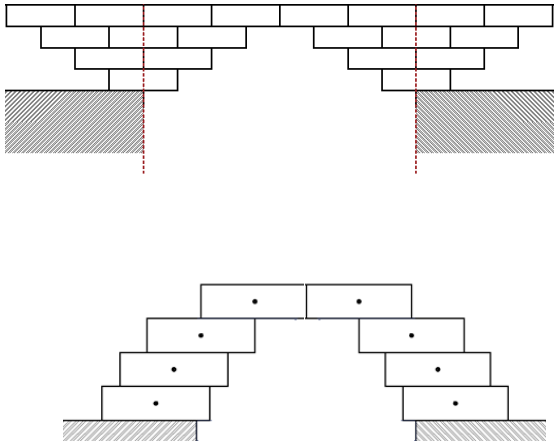


Figure 4.32: Example showing number of bricks required to build arch with/without back weight

4.5 Summary

In this chapter, the algorithms developed in Chapter 3 are coded to generate arches using Python component of Grasshopper plugin in the Rhino 3D software. The arches are generated for the case with and without bonding. In addition, arches subjected to the UDL on the top are also discussed.

Furthermore, the arches generated are analyzed in the later section of this Chapter. Curves for both cases are generated which pass through the centroid of each unit of the arch. And then, the curves are compared with parabolic and catenary curves with a same span. The comparison shows that as the height to span ratio increases the curves of the arches generated from this research deviate from catenary and parabolic curve.

The height to span ratios of the arches with different courses of bricks for both cases are calculated and plotted in the graphs. Analysis is also done to show the influence of the bonding strength of the mortar and addition of external vertical loads on the span of the arches. It is found that with increase in the strength of mortar, span of the arch increases. With increase in external load on the top of the arch, the span decreases.

The analysis regarding the stability of the structure formed with bonding is also done. Safety factor which needs to be considered while building in real field has also been discussed. The comparison with an arch having backweights shows that the arch formed through this research is material efficient.

CHAPTER 5: FROM ARCHES TO VAULTS

This chapter explores different forms of vaults generated employing the algorithms developed in Chapter 3 which is coded in the Python environment.

5.1 Extrusion of two-dimensional arch with no bonding

5.1.1 Example 1

The vault in Example 1 shown in Figure 5.2 is made of 35 courses of bricks. Bricks are stacked and have no bonding. The vault has a span of 2.75 ft. and height of 6.34 ft. To generate the form shown in Figure 5.2, the procedure discussed in Section 4.2.1 for generating an arch is followed. To create a vault, arches have to be generated along the y-direction as well. The number of brick courses in y-direction is controlled by the input 'M'. So, the additional steps for generating this form include getting a new insertion point for the bricks in the Y-direction (i.e, 'yCoordPt'). The point 'yCoordPt' is achieved by the vector addition between 'inPts' and the point [0, brickWidth×j, 0] for the left side of the arch. For the right side of the arch, 'yCoordPt' is achieved by the vector addition between 'nextStart' and the point [0, brickWidth×j, 0]. Taking 'yCoordPt' as new start point bricks are created in the range of 0 to M. The code is shown in Figure 5.1. Figure 5.3 is the flowchart explaining the procedure discussed in this section.

```
  startPts = inPts
  # creating first brick
  for j in range(0,M):
  ····yCoordPt= rs.VectorAdd(inPts, [0,brickWidth*j,0])
  ····nBrick = createBrick(yCoordPt,brickLength ,brickWidth,brickDepth)
  ····brickList.append(nBrick)


#nextStart point as a new starting point
nextStart = rs.VectorAdd(startPts, [(inPts.X-startPts.X)*2+brickLength,0,0])
# creating first brick
for j in range(0,M):
····yCoordPt= rs.VectorAdd(nextStart, [0,brickWidth*j,0])
····nBrick = createBrick(yCoordPt,brickLength ,brickWidth,brickDepth)
····brickList.append(nBrick)
```

Figure 5.1: Additional steps in the script to generate bricks in the y-direction

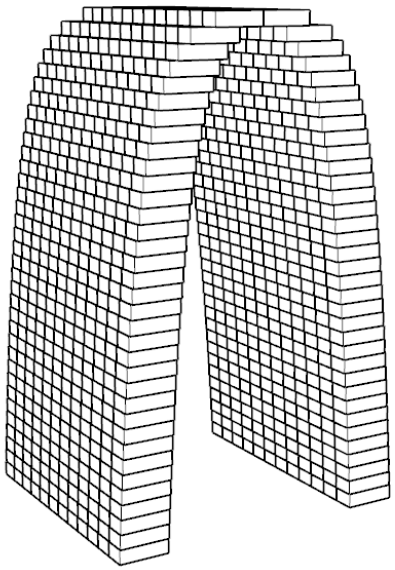


Figure 5.2: Example 1, a vault formed by extruding the arch for the case without mortar

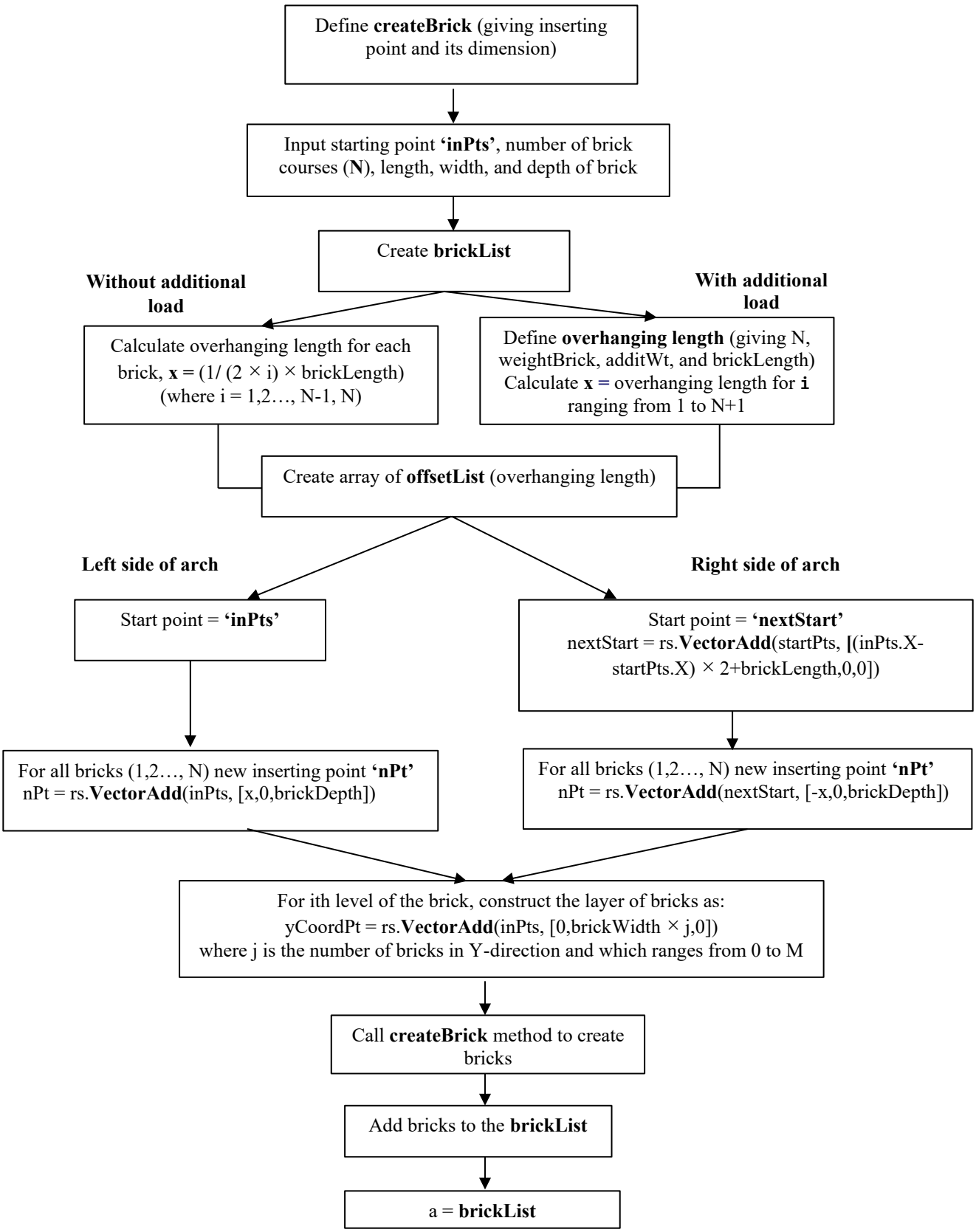
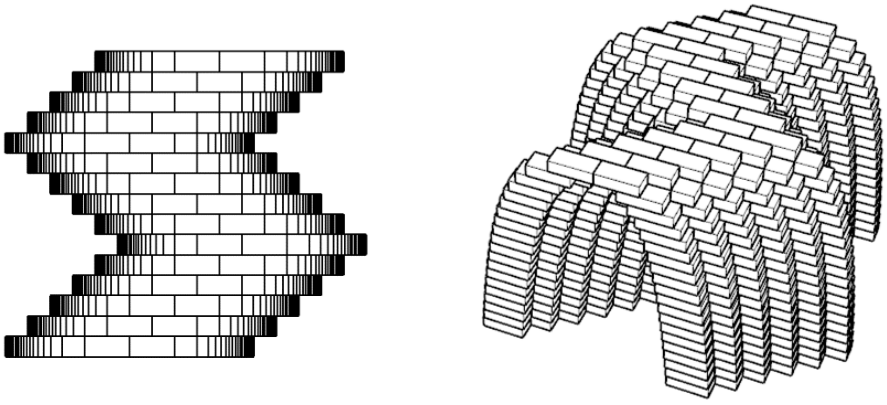


Figure 5.3: Flow chart of the code to generate a vault without mortar

5.1.2 Example 2

To generate the form shown in Figure 5.4, the procedure discussed in Section 5.1.1 is followed. In this example, each brick along the y-direction is shifted half the length of the previous brick towards the x-direction. To do this, a special condition is applied. The 'yCoordPt' for this case is defined by the vector addition of 'yCoordPt' and the point [0.5×brickLength, brickWidth, 0], if j is less or equal to 5 or j is greater than10. If not, 'yCoordPt' is created by vector addition of 'yCoordPt' and point [-(0.5×brickLength), brickWidth, 0]. The process of coding algorithm for this form is well described in the following flowchart in Figure 5.5.



Top view 3D view

Figure 5.4: Example 2, a Zig-zag vault formed by extruding the arch for the case without mortar

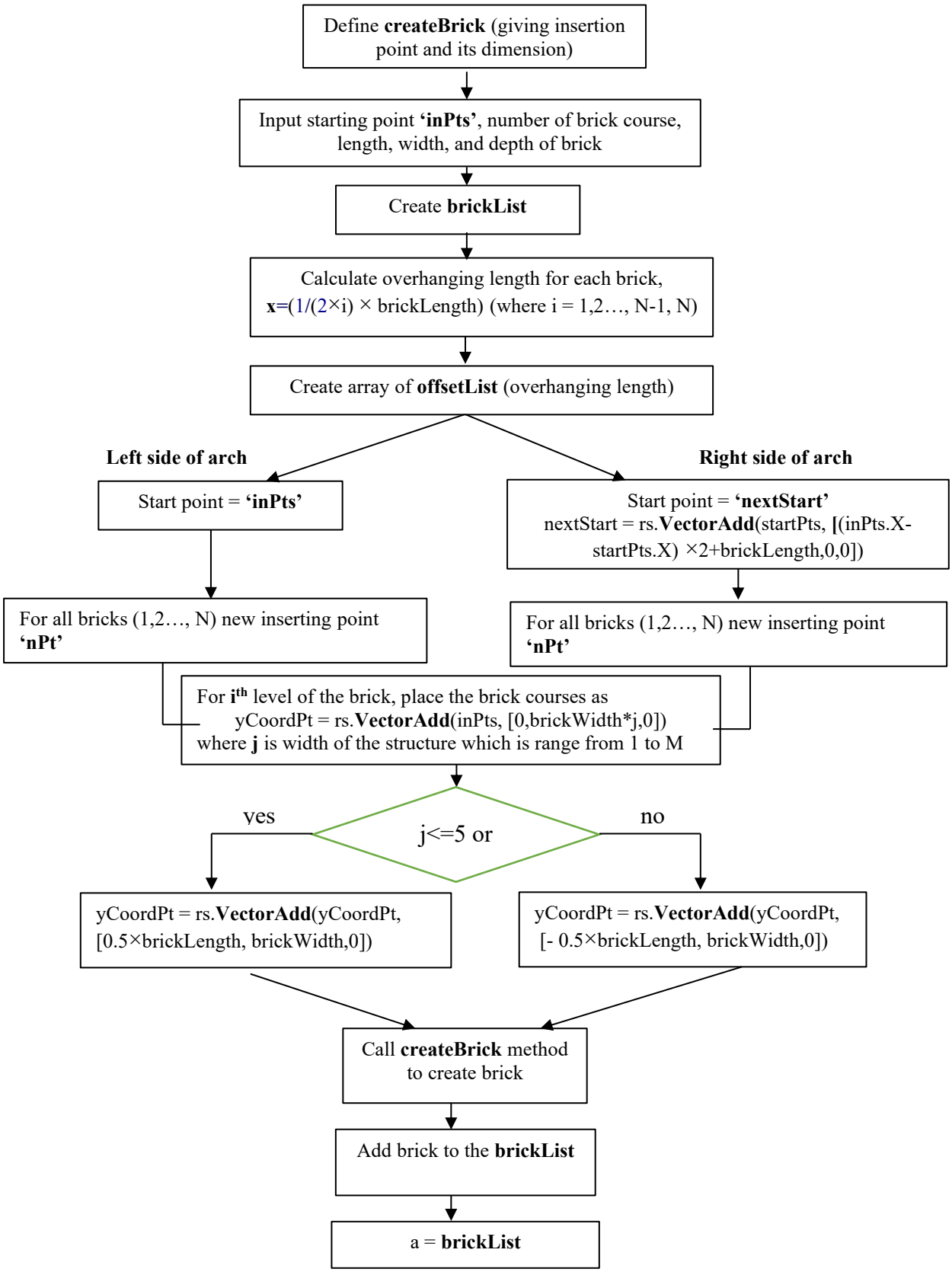


Figure 5.5: Flow chart of the code to generate a zig-zag vault for the case without mortar

5.2 Extrusion of two-dimensional arch with bonding

5.2.1 Example 1

Figure 5.6 is a vault formed when bricks are bonded with mortar. It has a span of 7.5 ft. and height of 7.8 ft. To generate this form, the steps discussed in Section 4.2.2 for generating a single arch is followed. For this case, such arches have to be generated along the y-direction as well. Then, the procedure followed is same as that for generating vault when bricks are not bonded with mortar.

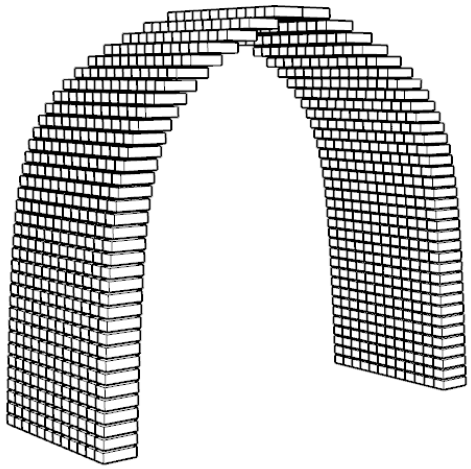


Figure 5.6: Example 1, a vault formed by extruding arch for the case with mortar

```
#to create first brick
newBrick = createBrick (inPts, brickLength, brickWidth, brickDepth)
brickList.append(newBrick)
for j in range(0,M):
    yCoordPt= rs.VectorAdd(inPts, [0,brickWidth*j,0])
    nBrick = createBrick(yCoordPt,brickLength ,brickWidth,brickDepth)
    brickList.append(nBrick)

#Section-2: method to create an arch on the right side of the structure
#nextStart point as a new starting point
nextStart = rs.VectorAdd(startPts, [(inPts.X-startPts.X)*2+brickLength,0,0])
for j in range(0,M):
    yCoordPt= rs.VectorAdd(nextStart, [0,brickWidth*j,0])
    nBrick = createBrick(yCoordPt,brickLength ,brickWidth,brickDepth)
    brickList.append(nBrick)
```

Figure 5.7: Additional step in the script to generate bricks in y-direction

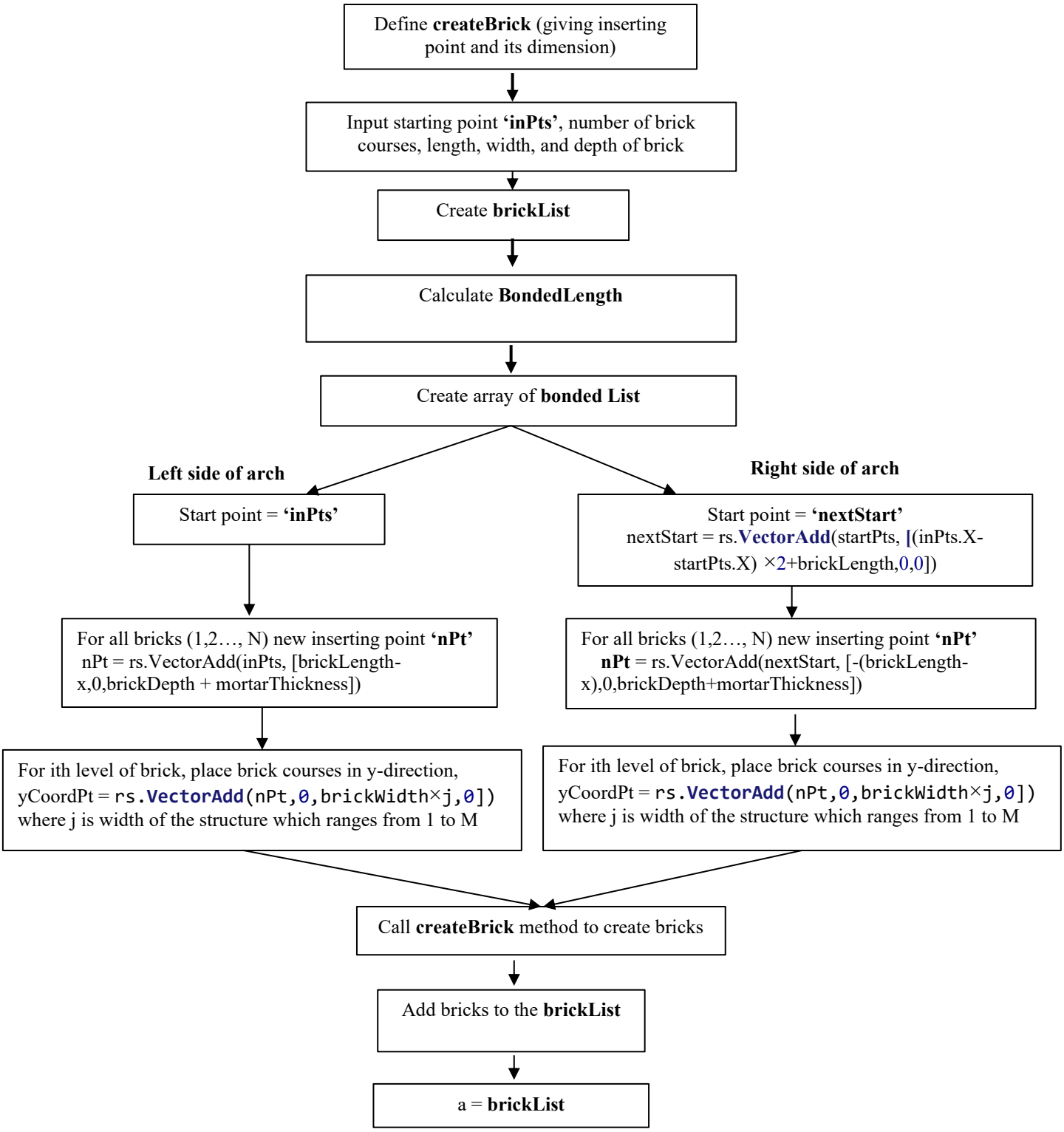
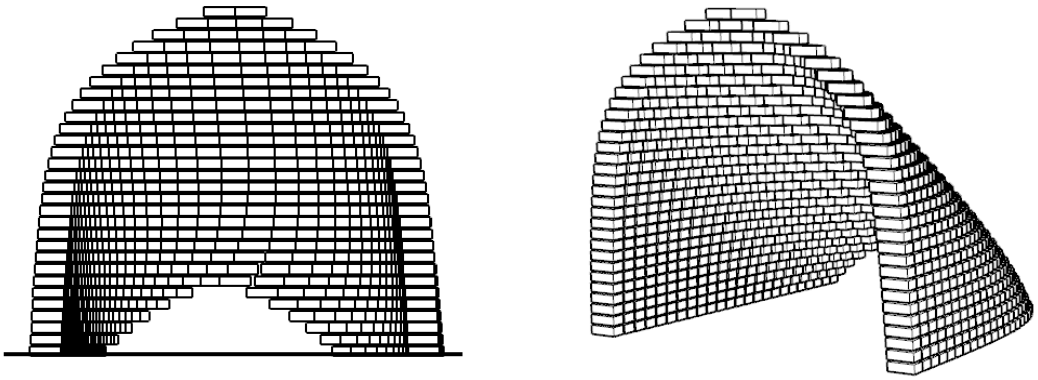


Figure 5.8: Flow chart of the code to generate a vault for the case with mortar

5.2.2 Example 2

A new definition ‘createArch’ for creating an arch is established which takes inputs ‘inPts’ and ‘N’. These inputs permit the generation of an arch at any point and with ‘N’ number of brick courses in the z-direction. Within the definition ‘createArch’ the arch is created using the same code discussed in Section 4.2.2. Procedures up to now create an arch. Taking the range of ‘i’ to be (N, M, -1) arches are created by using the ‘createArch’ definition which uses inputs (‘inPts, i). The ‘inPts’ is generated by vector addition between ‘inPts’ and [0.5, brickWidth, 0]. Here, a negative value ‘-1’ means that the range is decreasing (i.e., number of brick courses in z-direction are in a decreasing pattern). This step gives the arches ranging from N to M.



Front view 3D view

Figure 5.9: Example 2, one-way sloped vault formed for the case with mortar

5.2.3 Example 3

To generate this example, the script used to generate Example 2 in Section 5.2.2 is followed. In addition, the arches in the opposite direction from the same starting point are created. To create arches, ‘newinPts’ is achieved by the vector addition between the ‘newinPts’ and [0.5, -brickWidth, 0]. Taking the ‘newinPts’, arches are created for ‘i’ range. The python script for generating this form is attached in Appendix I of this thesis.

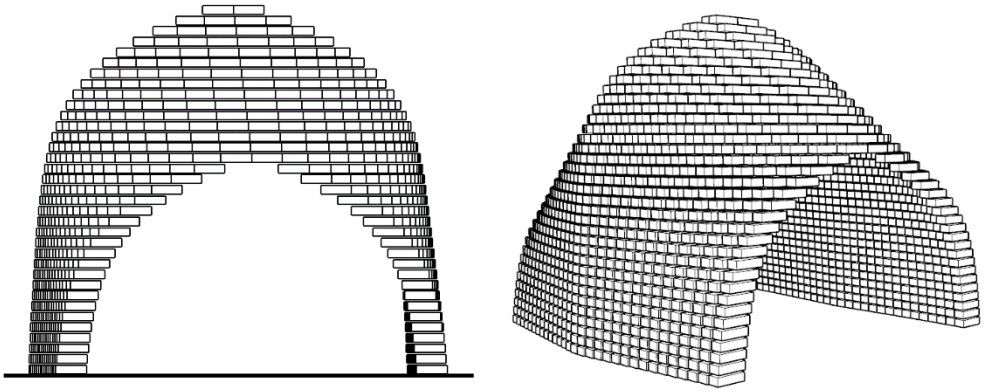


Figure 5.10: Example 3, a two-way sloped vault formed for the case with mortar

5.2.4 Example 4

To generate the form shown in Figure 5.11, the procedures to generate the Example 2 for the case with mortar are followed. In addition, the same script but with increasing number of brick courses on z-direction is used. The form generated can be repeated to generate the wavy vault shown in Figure 5.12. The script to generate all the forms are attached in Appendix of this thesis.

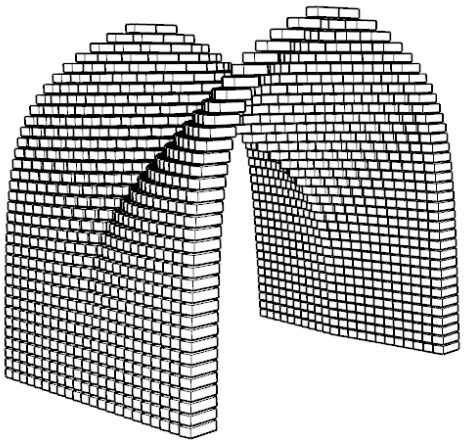


Figure 5.11: Example 4, a wavy vault formed for the case with mortar

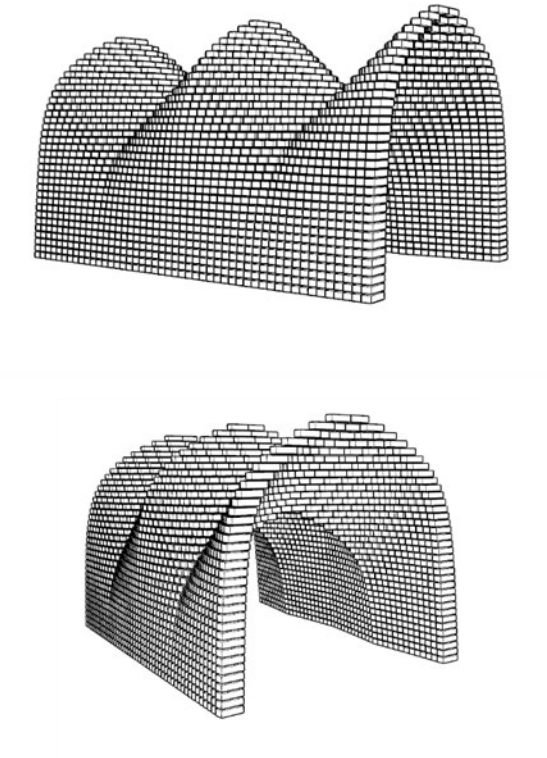
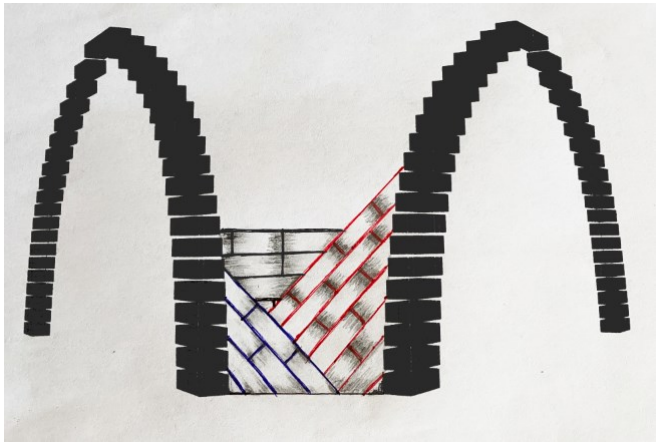


Figure 5.12: Repetition of the form in the Example 4

5.3 Using arches as guidelines for the vault construction

The forms generated until now are only the extrusion of arch. However, the arches can be used as guidelines to generate more exciting three-dimensional forms as the arches could carry additional loads. The tile vaulting technique could be used to vault between the arches. The process of building vaults using thin tile with guidelines has been discussed in Section 2.3.2. Figure 5.13(a) shows the 3D vault construction using thin tile. Figure 5.13 clearly explains one of the construction techniques with the three layers of tiles. The first layer of tile is laid horizontally as shown in Figure 5.13 (b) and is bonded with fast setting gypsum mortar. The second layer of tile is laid at 45° to the base as shown in Figure 5.13(c). The final layer is laid at 45° but in opposite direction of the second layer as shown in 5.13(d) and can also be laid on a different pattern. The second and final layers of tiles are bonded using cement mortar.



(a)

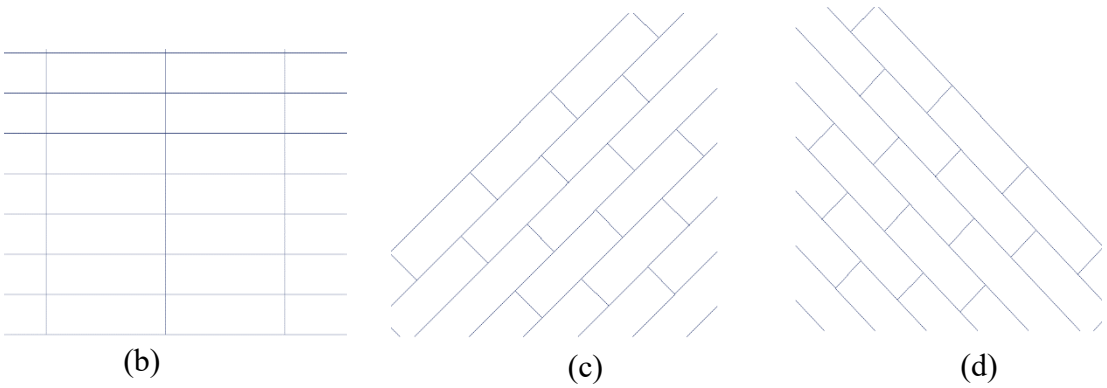


Figure 5.13: Construction of 3D vaults using thin-tile technique (a) an example of 3D vault (b) first layer of tile (c) second layer (d) third/final layer of tile

Several 3D vaults with different base geometries can be generated using this technique. Figure 5.14 shows different geometrical figures that can be used as a base to generate forms for the vaults. Figure 5.15 shows some examples of using the arches as guidelines to generate interesting forms.

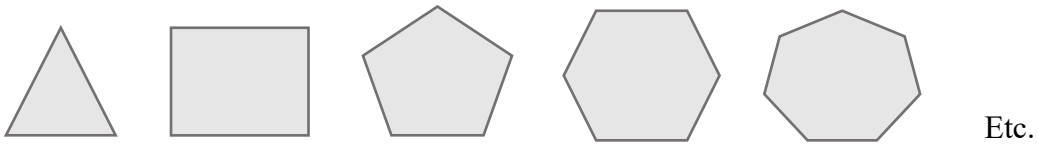
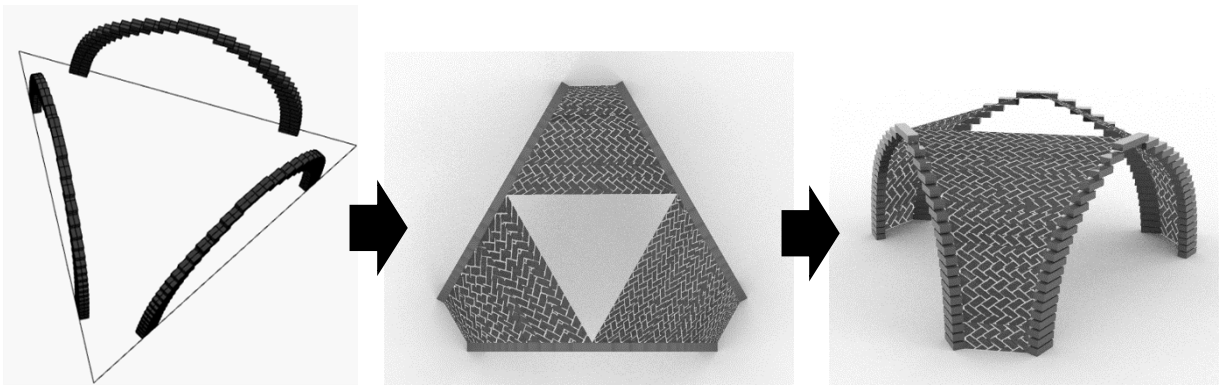
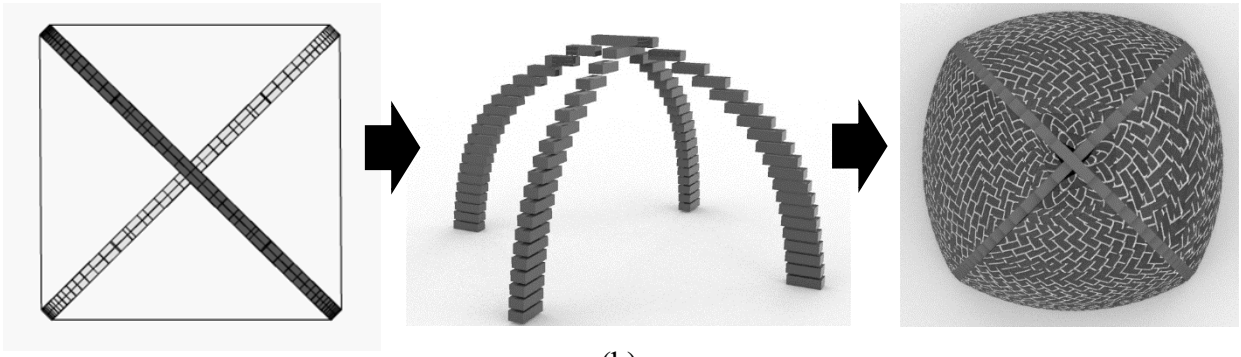


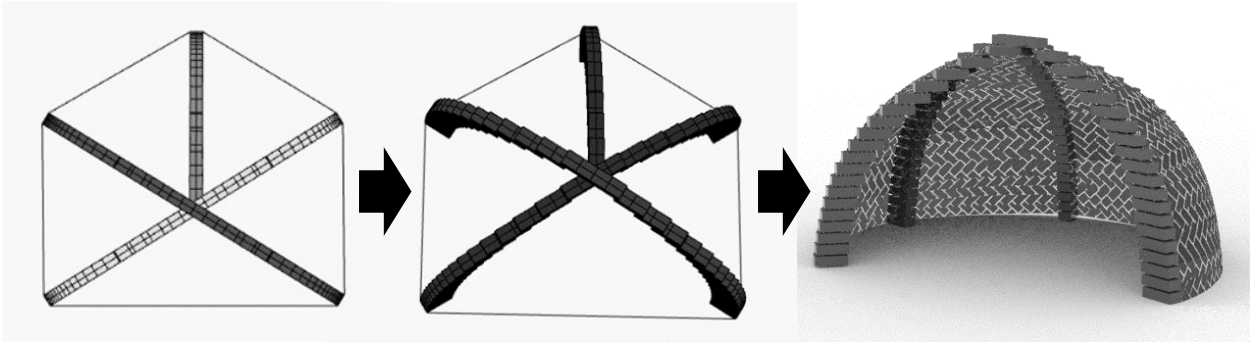
Figure 5.14: Different geometries which can be used as bases while generating 3D forms



(a)



(b)



(c)

Figure 5.15: Use of arch as guidelines to construct 3D vaults (a) triangular base (b) square base (c) pentagon base

5.4 Summary

This chapter discussed about the generation of exciting forms by using arches generated by coding the algorithm derived in Chapter 3. The forms are explored for the case with mortar and without mortar. By extruding the arch, various forms have been explored. The Python codes to generate each form have been explained explicitly using the screenshot images of codes and flowcharts.

It has also provided insight on how the arches can be used as guidelines to build three-dimensional vaults. Section 5.3 has described the technique using the arches as guidelines clearly. Different interesting forms can be generated using tile vaulting and the arches generated from this research as the arches can carry additional vertical loads.

CHAPTER 6: CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH

6.1 Conclusions

The research work presented in this thesis explores the possibility of finding forms of brick masonry vaults which do not need centering during the construction. Two cases are discussed, including bricks not bonded with mortar and bricks bonded with mortar. For each case algorithm is generated which gives the overhanging length of bricks over the lower course bricks. Bricks are then stacked one above the other following the overhanging length for both cases, creating the interesting forms which are discussed in Chapter 4 and 5.

A design tool is developed by coding the derived algorithms in Python environment of the Grasshopper. The tool developed through this research provides opportunities for parametric designs and further gives the possibilities in generating various forms. Several interesting forms can be explored by changing the values of inputs like dimensions of the brick, number of brick courses in y-direction (input M) and z-direction (input N), weight of the brick, strength of the mortar and additional loading condition. The course by course stability check, for the structures formed through this thesis, is also discussed. Reduced value of overhanging lengths can be used in real field to ensure safe construction. The discussion shows that the resisting moment is greater than overturning moment which determines the stability of courses during construction in real field. In addition, comparison with the arch having backweights shows that the forms generated from this research are material efficient.

6.2 Recommendations

- Three-dimensional equilibrium approach for Masonry vaults design

  The research reveals that various forms can be generated by considering the equilibrium condition. For this research study, only two-dimensional equilibrium condition was considered. Similarly, different forms can be explored by considering three-dimensional equilibrium conditions. Therefore, this research could provide a base for the future research for exploring new forms by considering the three-dimensional equilibrium approach.

- Integration of robotic construction for the designed vaults

  The result of this research can be verified using robotic construction. The algorithms developed through this research can be modified and coded to operate the robot arms, which stacks bricks as required to form the vault structures. The incorporation of the robotics in the vault construction has several advantages as it makes the construction process easier, cheaper, quicker, and material efficient. If the technology is not used during construction of the vaults, then workers have to measure the length of each individual brick while placing it. This practice increases the construction time. The incorporation of robotics ensures accuracy in construction and reduces the risk of masons being injured during construction. On the top of that, vaults can be built using robotics technology in the space (e.g. Mars) where human presence is not easy.

- Structural analysis to make the structure resilient

  The lateral loads i.e. wind and seismic forces carried by the arches and vaults were not considered in current study, but they may be discussed as future research topics. This thesis research has considered the vertical loading only. In the future, lateral loads can be designed for the reinforcing in the masonry vaults. Structural analysis can be done to make the

structure resilient to earthquake which will provide the possibility of constructing the vaults generated through this research in earthquake prone zones as well.

REFERENCES

Adriaenssens, S., Block, P., Veenendaal, D., & Williams, C. (Eds.). (2014). *Shell structures for architecture: form finding and optimization*. Routledge.

Allen, E., & Zalewski, W. (2009). *Form and forces: designing efficient, expressive structures*. John Wiley & Sons.

Allen, E., & Zalewski, W. (2009). *Form and forces: designing efficient, expressive structures*. John Wiley & Sons.

Angelillo, M. (2015). Static analysis of a Guastavino helical stair as a layered masonry shell. *Composite Structures*, *119*, 298-304.

Arun, G. (2006). Behaviour of Masonry Vaults and Domes: Geometrical Considerations. *Structural Analysis of Historical Constructions, New Dehli*, 299-306.

Bachman, D. (2017). *Grasshopper: Visual Scripting for Rhinoceros 3D*. Industrial Press, Inc.

Beckmann, P., & Bowles, R. (2012). *Structural aspects of building conservation*. Routledge.

Block, P., & Ochsendorf, J. (2007). Thrust network analysis: A new methodology for three-dimensional equilibrium. *Journal of the International Association for shell and spatial structures*, *48*(3), 167-173.

Chilton, J. (2009, November). 39 etc...: Heinz Isler's infinite spectrum of new shapes for shells. In Symposium of the International Association for Shell and Spatial Structures (50th. 2009.

Valencia). Evolution and Trends in Design, Analysis and Construction of Shell and Spatial Structures: Proceedings. Editorial Universitat Politècnica de València.

Chilton, J. (2010). Heinz Isler's Infinite Spectrum: Form-Finding in Design. *Architectural* Design, 80(4), 64-71.

Chilton, J. (2012). Form-finding and fabric forming in the work of Heinz Isler. In *Proceedings of Second International Conference on Flexible Formwork, eds. John Orr, Mark Evernden, Antony Darby and Tim Ibell* (pp. 84-91).).

Collins, G. (1963). Antonio Gaudi: Structure and Form. *Perspecta,8*, 63-90.doi:10.2307/1566905

Como, M. (2013). *Statics of historic masonry constructions* (Vol. 1). Berlin: Springer

Council, I. C., (2006). International Building Code.

Dahmen, J. F. D., & Ochsendorfs, J. A. (2012). Earth masonry structures: arches, vaults, and domes. In *Modern Earth Buildings* (pp. 427-460). Woodhead Publishing.

Davis, L., Rippmann, M., Pawlofsky, T., & Block, P. (2011). Efficient and Expressive Thin-tile Vaulting using Cardboard Formwork. IABSE—IASS 2011 London Symposium.

DeLaine, J. (1990). Structural Experimentation: The Lintel Arch, Corbel, and Tie in Western Roman Architecture. *World Archaeology, 21*(3), 407-424. Retrieved from http://www.jstor.org/stable/124838

Deuss, M., Panozzo, D., Whiting, E., Liu, Y., Block, P., Sorkine-Hornung, O., & Pauly, M. (2014). Assembling self-supporting structures. ACM Transactions on Graphics, 33(6), 1–10. https://doi.org/10.1145/2661229.2661266

Fitchen, J. (1981). *The construction of Gothic cathedrals: a study of medieval vault erection*. University of Chicago Press.

Gargiulo, M. R., & Bergamasco, I. (2006). The Use of Earth in the Architecture of Hassan Fathy and Fabrizio Carola: Typological and Building Innovations, Building Technology and Static Behaviour. In *Second International Congress on Construction History. Proceedings of the Second International Congress on Construction History. Cambridge* (Vol. 2, pp. 1209-1220).

Golesh Abel Danjuma (2013). Corbelling in Residential Buildings.

Hendry, A. W. (2001). Masonry walls: materials and construction. *Construction and Building materials*, *15*(8), 323-330.

Hendry, E. A. (2001). Masonry walls: materials and construction. *Construction and Building materials*, *15*(8), 323-330.

Hetherington, S. (2015). A comparative study into the tensile bond strength of the brick mortar interface of Naturally Hydraulic lime and Portland cement mortars. *Masonry International*, *28*(2), 67-64.

Heyman, J. (1982). *The Masonry Arch*.

Hogberg, E. (1967). *Mortar bond*. National Swedish Institute for Building Research.

Högberg, E. (1967). *Mortar bond*. National Swedish Institute for Building Research.

Hood, M. (1960). Tholos Tombs of the Aegean. Antiquity, 34(135), 166-176. doi:10.1017/S0003598X00029331

Howe, M. (Author/Creator). (2016). Marble pavilion: Vila Vicosa Portugal. Artefact

Joffroy, T., & Guillaud, H. (1994). The basics of building with arches, vaults and cupolas.

Lau, W. W. (2006). *Equilibrium analysis of masonry domes* (Doctoral dissertation, Massachusetts Institute of Technology).

López, D. L., Rodríguez, M. D., & Fernández, M. P. (2014). "Brick-topia", the thin-tile vaulted pavilion. *Case Studies in Structural Engineering*, *2*, 33-40.

Marr, D., & MILNER, A. (Eds.). (1986). Southeast Asia in the 9th to 14th Centuries. ISEAS–Yusof Ishak Institute.

McNeel, R. (2009). Rhinoceros 3D. *Retrieved Jan 15*.

Melaragno, M. (2012). *An introduction to shell structures: The art and science of vaulting*. Springer Science & Business Media.

Miles, M. (2006). Utopias of Mud? Hassan Fathy and Alternative Modernisms. *Space and Culture*, *9*(2), 115–139.

Mortar for masonry, 1997. *International Masonry Institute*

O'Dwyer, D. (1999). Funicular analysis of masonry vaults. *Computers & Structures*, *73*(1-5), 187-197.

Ochsendorf, J. (2014). Guastavino Masonry shells. *STRUCTURE*, *26*.

Palmer, L. A., & Hall, J. V. (1931). Durability and strength of bond between mortar and brick. *United States Bureau of Standards Journal of Research*, *6*(3), 28.

Palmer, L. A., & Hall, J. V. (1931). Durability and strength of bond between mortar and brick. *United States Bureau of Standards Journal of Research*, *6*(3), 28.

Ponce, A. R., & Meléndez, R. R. (2004). Curves of Clay: Bóvedas del Bajío. *Nexus V, Architecture and Mathematics., Florence: Kim Williams Books*, 143-154.

Rajabzadeh, S., & Sassone, M. (2017). Brick Patterning on Free-Form Surfaces. *Nexus Network Journal*, *19*(1), 5-25.

Ramage, M. H. (2004). Building a Catalan vault.

Ramage, Michael. "Guastavino's vault construction revisited." Construction History 22 (2007): 47-60

Rippmann, M., & Block, P. (2013). Rethinking structural masonry: unreinforced, stone-cut shells. *Proceedings of the Institution of Civil Engineers-Construction Materials*, *166*(6), 378-389

Serageldin, I. (2007). *Hassan Fathy*.

Tanner, J. E., & Klingner, R. E. (2017). *Masonry structural design*. McGraw-Hill Education.

Tanner, J. E., & Klingner, R. E. (2017). *Masonry structural design*. McGraw Hill Professional.

Tibbits, S., A. vd Harten, and S. Baer. "Python for Rhinoceros 5." *URL: http://python. rhino3d. com/content/130-RhinoPython-primer* (2011).

United States. Bureau of Standards, Palmer, L. A., & Parsons, D. A. (1934). *A study of the properties of mortars and bricks and their relation to bond*. US Government Printing Office.

United States. Bureau of Standards, Palmer, L. A., & Parsons, D. A. (1934). *A study of the properties of mortars and bricks and their relation to bond*. US Government Printing Office.

Van Beek, G. (1987). Arches and Vaults in the Ancient Near East. *Scientific American, 257*(1), 96-103. Retrieved from http://www.jstor.org/stable/24979426

Wendland, D. (2000). Model-based formfinding processes: Free forms in structural and architectural design.

Wendland, D. (2003). A case of Recovery of a Medieval Vaulting Technique in the 19th Century: Lassaulx's Vaults in the Church of Treis.

Wendland, D. (2004, November). Some considerations on the shape of the caps of vaults. In *Proc. IV Int. Seminar on Structural Analysis of Historical Constructions, Padova* (pp. 111-120).

Wendland, D. (2005). Vaults built without formwork: comparison of the description of a traditional technique in building manuals with the results of practical observations and experimental studies.

Wendland, D. (2007). Traditional vault construction without formwork: Masonry pattern and vault shape in the historical technical literature and in experimental studies. *International Journal of Architectural Heritage*, *1*(4), 311-365.

Websites:

https://www.rhino3d.com/

arch. 2020. In *Merriam-Webster.com*. Retrieved May 8, 2020, from https://www.merriam-webster.com/dictionary/hacker

Online Lecture:

Ochsendorf, J (2011) Form and Forces, http://www.youtube.com/watch?v=r-tG68WvNDM

APPENDICES

Appendix A- Python code to generate arch with no bonding and with no additional load

```
__author__ = "Babita Neupane, bneupane@kent.edu"
__copyright__ = "Copyright (C) 2020 Babita Neupane"
__version__ = "1.0"
__date__ = "2020.05.30"
"""
This code generates arch for the case where bricks are not bonded with mortar
It includes following steps:
1. Create the list of the overhanging length using provided equation
2. Generate the list of the bricks for the left side of the arch
3. Generate the list of the bricks for the right side of the arch
4. Print all the bricks
"""

import rhinoscriptsyntax as rs

#definition for a single brick
def createBrick (insertPt, dimX, dimY, dimZ):
    pt1 = [insertPt[0], insertPt[1] - dimY, insertPt[2]+dimZ]
    pt2 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]+dimZ]
    pt3 = [insertPt[0]+dimX, insertPt[1], insertPt[2]+dimZ]
    pt4 = [insertPt[0], insertPt[1], insertPt[2]+dimZ]
    pt5 = [insertPt[0], insertPt[1] - dimY, insertPt[2]]
    pt6 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]]
    pt7 = [insertPt[0]+dimX, insertPt[1], insertPt[2]]
    pt8 = [insertPt[0], insertPt[1], insertPt[2]]
    newbrick = rs.AddBox([pt1, pt2, pt3, pt4, pt5, pt6, pt7, pt8])
    return(newbrick)

#List of bricks
brickList=[]
#List of overhanging lengths of each brick
offsetList=[]

#Section-1: method to create the arch on the left side of the structure
#creating first brick
#taking start point as inPts
startPts = inPts
newBrick = createBrick(startPts, brickLength, brickWidth, brickDepth)
brickList.append(newBrick)
#Method to stack bricks
for i in range(N-1,0,-1):
    x=(1/(2*i)*brickLength)
    offsetList.append(x)

for i in range(1,N):
    x = offsetList[i-1]
    print x
    nPt = rs.VectorAdd(inPts, [x,0,brickDepth])
    nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
    brickList.append(nBrick)
    inPts = nPt
```

```
#Section-2: method to create an arch on the right side of the structure
#nextStart point as a new starting point
nextStart = rs.VectorAdd(startPts, [(inPts.X-startPts.X)*2+brickLength,0,0])
newBrick = createBrick (nextStart, brickLength, brickWidth, brickDepth)
brickList.append(newBrick)

for i in range(1,N):
    x = offsetList[i-1]
    nPt = rs.VectorAdd(nextStart, [-x,0,brickDepth])
    nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
    brickList.append(nBrick)
    nextStart = nPt

#to print all the bricks
a = brickList
```

Appendix B- Python code to generate arch with bonding and with no additional load

```
__author__ = "Babita Neupane, bneupane@kent.edu"
__copyright__ = "Copyright (C) 2020 Babita Neupane"
__version__ = "1.0"
__date__ = "2020.05.30"
"""
This code generates arch for the case where bricks are bonded with mortar
It includes following steps:
1. Create the list of bonded length using provided equation
   Using BondingLength method to create the overhanging length for ith brick, i.e. numbrick
2. Generate the list of the bricks for the left side of the arch
3. Generate the list of the bricks for the right side of the arch
4. Print all the bricks
"""

import rhinoscriptsyntax as rs
import math

#definition for a single brick
#creates the bricks at the position insertPt with dimension (dimX, dimY, dimZ)
#and returns the newBrick
def createBrick (insertPt, dimX, dimY, dimZ):
    pt1 = [insertPt[0], insertPt[1] - dimY, insertPt[2]+dimZ]
    pt2 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]+dimZ]
    pt3 = [insertPt[0]+dimX, insertPt[1], insertPt[2]+dimZ]
    pt4 = [insertPt[0], insertPt[1], insertPt[2]+dimZ]
    pt5 = [insertPt[0], insertPt[1] - dimY, insertPt[2]]
    pt6 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]]
    pt7 = [insertPt[0]+dimX, insertPt[1], insertPt[2]]
    pt8 = [insertPt[0], insertPt[1], insertPt[2]]
    newbrick = rs.AddBox([pt1, pt2, pt3, pt4, pt5, pt6, pt7, pt8])
    return(newbrick)

listOfLength = []

#method to get bonding length of the bricks based on the numbricks using nth terms
#refer thesis document for the equation
def BondingLength(numbrick,weightBrick,bondingStrength,bWidth,bDepth):
    sum=0
    for i in range(1,numbrick):
        sum = sum+2*i*listOfLength[i-1]
    numerator=-
numbrick*weightBrick+math.sqrt(math.pow(numbrick*weightBrick,2)+bondingStrength*brickWidth*
weightBrick*(numbrick*numbrick*brickLength - sum))
    denominator=bondingStrength*bWidth
    bondingLen= numerator/denominator
    listOfLength.append(bondingLen)
    return bondingLen


startPts = inPts
brickList = []
#print the first brick
```

```
newBrick = createBrick (inPts, brickLength, brickWidth, brickDepth)
brickList.append(newBrick)


#create a list of bonded length
bondedList=[]
for i in range(1,N):
    y = BondingLength(i,weightBrick,bondingStrength,brickWidth,brickLength)
    bondedList.append(y)
#Section-1: method to create the arch on the left side of the structure
#Method to stack bricks
for i in range(1,N):
    x = bondedList[(N-1)-i]
    nPt = rs.VectorAdd(inPts, [brickLength-x,0,brickDepth + mortarThickness])
    nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
    brickList.append(nBrick)
    inPts = nPt

#Section-2: method to create an arch on the right side of the structure
#nextStart point as a new starting point
nextStart = rs.VectorAdd(startPts, [(inPts.X-startPts.X)*2+brickLength,0,0])
newBrick = createBrick (nextStart, brickLength, brickWidth, brickDepth)
brickList.append(newBrick)
#Method to stack bricks
for i in range(1,N):
    x = bondedList[(N-1)-i]
    nPt = rs.VectorAdd(nextStart, [-(brickLength-x),0,brickDepth+mortarThickness])
    nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
    brickList.append(nBrick)
    nextStart = nPt

#to print all the bricks
a = brickList
```

Appendix C- Python code to generate arch with no bonding and subjected to the additional load

```
__author__ = "Babita Neupane, bneupane@kent.edu"
__copyright__ = "Copyright (C) 2020 Babita Neupane"
__version__ = "1.0"
__date__ = "2020.05.30"
"""
This code generates arch for the case where bricks are not bonded with mortar
It includes following steps:
1. Create the list of the overhanging length using provided equation
2. Generate the list of the bricks for the left side of the arch
3. Generate the list of the bricks for the right side of the arch
4. Print all the bricks
"""


import rhinoscriptsyntax as rs
import math

#definition for a single brick
#creates the bricks at the position insertPt with dimension (dimX, dimY, dimZ)
#and returns the newBrick
def createBrick (insertPt, dimX, dimY, dimZ):
    pt1 = [insertPt[0], insertPt[1] - dimY, insertPt[2]+dimZ]
    pt2 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]+dimZ]
    pt3 = [insertPt[0]+dimX, insertPt[1], insertPt[2]+dimZ]
    pt4 = [insertPt[0], insertPt[1], insertPt[2]+dimZ]
    pt5 = [insertPt[0], insertPt[1] - dimY, insertPt[2]]
    pt6 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]]
    pt7 = [insertPt[0]+dimX, insertPt[1], insertPt[2]]
    pt8 = [insertPt[0], insertPt[1], insertPt[2]]
    newbrick = rs.AddBox([pt1, pt2, pt3, pt4, pt5, pt6, pt7, pt8])
    return(newbrick)


listofLength =[]
listofLength.append(0)

#method to get bonding length of the bricks based on the numbricks using nth terms
#refer: thesis document for the equation
#definition for the overhanging length
def overhangingLength(N,weightBrick, additWt,brickLength):
    sum1=0
    for i in range(1,N-1):
        sum1 =sum1 +listofLength[i]
    sum2 = N*weightBrick+additWt*(brickLength+sum1)
    numerator = weightBrick*brickLength
    denominator = 2*sum2
    overlen= numerator/denominator
    listofLength.append(overlen)
    return overlen


#List of bricks
```

```
brickList=[]



#List of overhanging lengths of each brick
offsetList=[]
for i in range(1,N):
    x=overhangingLength(i,weightBrick, additWt,brickLength)
    offsetList.append(x)
    print i   , x

#Section-1: method to create the arch on the left side of the structure
#creating first brick
#taking start point as inPts
startPts = inPts
newBrick = createBrick (inPts, brickLength, brickWidth, brickDepth)
brickList.append(newBrick)
#Method to stack bricks
for i in range(1,N):
    x = offsetList[N-i-1]

    nPt = rs.VectorAdd(inPts, [x,0,brickDepth])
    nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
    brickList.append(nBrick)
    inPts = nPt

#Section-2: method to create an arch on the right side of the structure
#nextStart point as a new starting point
nextStart = rs.VectorAdd(startPts, [(inPts.X-startPts.X)*2+brickLength,0,0])
newBrick = createBrick (nextStart, brickLength, brickWidth, brickDepth)
brickList.append(newBrick)
#Method to stack bricks
for i in range(1,N):
    x = offsetList[N-i-1]
    nPt = rs.VectorAdd(nextStart, [-x,0,brickDepth])
    nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
    brickList.append(nBrick)
    nextStart = nPt

#to print all the bricks
a = brickList
```

Appendix D- Python code to generate arch with bonding and subjected to the additional load

```
__author__ = "Babita Neupane, bneupane@kent.edu"
__copyright__ = "Copyright (C) 2020 Babita Neupane"
__version__ = "1.0"
__date__ = "2020.05.30"
"""
This code generates arch for the case where bricks are bonded with mortar
It includes following steps:
1. Create the list of bonded length using provided equation
   Using BondingLength method to create the overhanging length for ith brick, i.e. numbrick
2. Generate the list of the bricks for the left side of the arch
3. Generate the list of the bricks for the right side of the arch
4. Print all the bricks
"""

import rhinoscriptsyntax as rs
import math

#definition for a single brick
#creates the bricks at the position insertPt with dimension (dimX, dimY, dimZ)
#and returns the newBrick
def createBrick (insertPt, dimX, dimY, dimZ):
    pt1 = [insertPt[0], insertPt[1] - dimY, insertPt[2]+dimZ]
    pt2 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]+dimZ]
    pt3 = [insertPt[0]+dimX, insertPt[1], insertPt[2]+dimZ]
    pt4 = [insertPt[0], insertPt[1], insertPt[2]+dimZ]
    pt5 = [insertPt[0], insertPt[1] - dimY, insertPt[2]]
    pt6 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]]
    pt7 = [insertPt[0]+dimX, insertPt[1], insertPt[2]]
    pt8 = [insertPt[0], insertPt[1], insertPt[2]]
    newbrick = rs.AddBox([pt1, pt2, pt3, pt4, pt5, pt6, pt7, pt8])
    return(newbrick)

listofLength = []

#definition of bonding length
#method to get bonding length of the bricks based on the numbricks using nth terms
#refer: thesis document for the equation
def BondingLength(numbrick,weightBrick,bondingStrength,brickWidth,brickLength):
    print numbrick
    sum=0
    sum1a = additWt*brickLength+weightBrick
    sum1b = 0
    for i in range(1,numbrick):
        sum1b =sum1b+listofLength[i-1]
    sum1c = numbrick*numbrick-1
    sum1d = 0
    for i in range(1,numbrick):
        sum1d = sum1d+i*listofLength[i-1]
    sum1e =bondingStrength*brickWidth-additWt
    a = sum1e
    b = numbrick*sum1a-additWt*sum1b
```

```
    c = (additWt*((-1*sum1c*brickLength*brickLength)+2*brickLength*numbrick*sum1b-
sum1b*sum1b)+2*weightBrick*sum1d-numbrick*numbrick*weightBrick*brickLength)
    numerator=-b+math.sqrt(b*b- a*c)
    denominator = a
    bondingLen= numerator/denominator
    print bondingLen
    listofLength.append(bondingLen)
    return bondingLen


#Starting point
startPts = inPts
#Creating list of bricks
brickList = []

#print the first brick
newBrick = createBrick (inPts, brickLength, brickWidth, brickDepth)
brickList.append(newBrick)

#create a list of bonded length
bondedList=[]
for i in range(1,N):
    y = BondingLength(i,weightBrick,bondingStrength,brickWidth,brickLength)
    bondedList.append(y)

#Section-1: method to create the arch on the left side of the structure
#Method to stack bricks
for i in range(1,N):
    x = bondedList[(N-1)-i]
    nPt = rs.VectorAdd(inPts, [brickLength-x,0,brickDepth + mortarThickness])
    nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
    brickList.append(nBrick)
    inPts = nPt

#Section-2: method to create an arch on the right side of the structure
#nextStart point as a new starting point
nextStart = rs.VectorAdd(startPts, [(inPts.X-startPts.X)*2+brickLength,0,0])
newBrick = createBrick (nextStart, brickLength, brickWidth, brickDepth)
brickList.append(newBrick)

#Method to stack bricks
for i in range(1,N):
    x = bondedList[(N-1)-i]
    nPt = rs.VectorAdd(nextStart, [-(brickLength-x),0,brickDepth+mortarThickness])
    nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
    brickList.append(nBrick)
    nextStart = nPt

#to print all the bricks
a = brickList
```

Appendix E- Python code to generate vault (i.e., example 1 with no bonding and no additional load)

```
__author__ = "Babita Neupane, bneupane@kent.edu"
__copyright__ = "Copyright (C) 2020 Babita Neupane"
__version__ = "1.0"
__date__ = "2020.05.30"
"""
This code generates vault for the case where bricks are not bonded with mortar
It includes following steps:
1. Create the list of the overhanging length using provided equation
2. Generate the list of the bricks for the left side of the arch
3. Generate the list of the bricks for the right side of the arch
4. Generate the list of the bricks in the y-direction
5. Print all the bricks
"""

import rhinoscriptsyntax as rs

#definition for a single brick
#creates the bricks at the position insertPt with dimension (dimX, dimY, dimZ)
#and returns the newBrick
def createBrick (insertPt, dimX, dimY, dimZ):
    pt1 = [insertPt[0], insertPt[1] - dimY, insertPt[2]+dimZ]
    pt2 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]+dimZ]
    pt3 = [insertPt[0]+dimX, insertPt[1], insertPt[2]+dimZ]
    pt4 = [insertPt[0], insertPt[1], insertPt[2]+dimZ]
    pt5 = [insertPt[0], insertPt[1] - dimY, insertPt[2]]
    pt6 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]]
    pt7 = [insertPt[0]+dimX, insertPt[1], insertPt[2]]
    pt8 = [insertPt[0], insertPt[1], insertPt[2]]
    newbrick = rs.AddBox([pt1, pt2, pt3, pt4, pt5, pt6, pt7, pt8])
    return(newbrick)



#List of bricks
brickList=[]
#List of overhanging lengths of each brick
offsetList=[]

#Section-1: method to create the arch on the left side of the structure
#creating first brick
#taking start point as inPts
startPts = inPts
# creating first brick
for j in range(0,M):
    yCoordPt= rs.VectorAdd(inPts, [0,brickWidth*j,0])
    nBrick = createBrick(yCoordPt,brickLength ,brickWidth,brickDepth)
    brickList.append(nBrick)
#Method to stack bricks
for i in range(N-1,0,-1):
    x=(1/(2*i)*brickLength)
    offsetList.append(x)
for i in range(1,N):
```

```
    x = offsetList[i-1]
    nPt = rs.VectorAdd(inPts, [x,0,brickDepth])
#Method to create bricks in the y-direction
    for j in range(0,M):
        yCoordPt= rs.VectorAdd(nPt, [0,brickWidth*j,0])
        nBrick = createBrick(yCoordPt,brickLength ,brickWidth,brickDepth)
        brickList.append(nBrick)
    inPts = nPt

#Section-2: method to create an arch on the right side of the structure
#nextStart point as a new starting point
nextStart = rs.VectorAdd(startPts, [(inPts.X-startPts.X)*2+brickLength,0,0])
# creating first brick
for j in range(0,M):
    yCoordPt= rs.VectorAdd(nextStart, [0,brickWidth*j,0])
    nBrick = createBrick(yCoordPt,brickLength ,brickWidth,brickDepth)
    brickList.append(nBrick)
#Method to stack bricks
for i in range(1,N):
    x = offsetList[i-1]
    nPt = rs.VectorAdd(nextStart, [-x,0,brickDepth])
#Method to create bricks in the y-direction
    for j in range(0,M):
        yCoordPt= rs.VectorAdd(nPt, [0,brickWidth*j,0])
        nBrick = createBrick(yCoordPt,brickLength ,brickWidth,brickDepth)
        brickList.append(nBrick)
    nextStart = nPt

#to print all the bricks
a = brickList
```

Appendix F Python code to generate zig-zag vault (i.e., example 2 with no bonding and no additional load)

```
import rhinoscriptsyntax as rs

# definition for a single brick
def createBrick (insertPt, dimX, dimY, dimZ):
    pt1 = [insertPt[0], insertPt[1] - dimY, insertPt[2]+dimZ]
    pt2 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]+dimZ]
    pt3 = [insertPt[0]+dimX, insertPt[1], insertPt[2]+dimZ]
    pt4 = [insertPt[0], insertPt[1], insertPt[2]+dimZ]
    pt5 = [insertPt[0], insertPt[1] - dimY, insertPt[2]]
    pt6 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]]
    pt7 = [insertPt[0]+dimX, insertPt[1], insertPt[2]]
    pt8 = [insertPt[0], insertPt[1], insertPt[2]]
    newbrick = rs.AddBox([pt1, pt2, pt3, pt4, pt5, pt6, pt7, pt8])
    return(newbrick)

#M = number of bricks in y-direction
#N= number of layer of bricks stacked

#List of bricks
brickList=[]

#taking start point as inPts
startPts = inPts
#creating first half of the arch
for j in range(0,M):
    if (j<=5 or j>10):
        yCoordPt= rs.VectorAdd(inPts, [(0.5*brickLength),brickWidth,0])
        nBrick = createBrick(yCoordPt,brickLength ,brickWidth,brickDepth)
        brickList.append(nBrick)
    else:
        yCoordPt= rs.VectorAdd(inPts, [-(0.5*brickLength),brickWidth,0])
        nBrick = createBrick(yCoordPt,brickLength, brickWidth,brickDepth)
        brickList.append(nBrick)
    inPts= yCoordPt

#List of overhanging lengths of each brick
offsetList=[]
inPts = startPts
for i in range(N-1,0,-1):
    x=(1/(2*i)*brickLength)
    offsetList.append(x)
for i in range(1,N):
    x = offsetList[i-1]
    nPt = rs.VectorAdd(inPts, [x,0,brickDepth])
    yCoordPt = nPt
    for j in range(0,M):
        if (j<=5 or j>10):
            yCoordPt= rs.VectorAdd(yCoordPt, [0.5*brickLength,brickWidth,0])
            nBrick = createBrick(yCoordPt,brickLength,brickWidth,brickDepth)
            brickList.append(nBrick)
        else:
            yCoordPt= rs.VectorAdd(yCoordPt, [-(0.5*brickLength),brickWidth,0])
```

```
            nBrick = createBrick(yCoordPt,brickLength ,brickWidth,brickDepth)
            brickList.append(nBrick)
    inPts = nPt

#creating arch
#nextStart point as starting point for another half of the arch
nextStart = rs.VectorAdd(startPts, [(inPts.X-startPts.X)*2+brickLength,0,0])
yCoordPt = nextStart
for j in range(0,M):
    if (j<=5 or j>10):
        yCoordPt= rs.VectorAdd(yCoordPt, [0.5*brickLength,brickWidth,0])
        nBrick = createBrick(yCoordPt,brickLength,brickWidth,brickDepth)
        brickList.append(nBrick)
    else:
         yCoordPt= rs.VectorAdd(yCoordPt, [-(0.5*brickLength),brickWidth,0])
         nBrick = createBrick(yCoordPt,brickLength,brickWidth,brickDepth)
         brickList.append(nBrick)

for i in range(1,N):
    x = offsetList[i-1]
    nPt = rs.VectorAdd(nextStart, [-x,0,brickDepth])
    yCoordPt = nPt
    for j in range(0,M):
        if (j<=5 or j>10):
            yCoordPt= rs.VectorAdd(yCoordPt, [0.5*brickLength,brickWidth,0])
            nBrick = createBrick(yCoordPt,brickLength,brickWidth,brickDepth)
            brickList.append(nBrick)
        else:
            yCoordPt= rs.VectorAdd(yCoordPt, [-(0.5*brickLength),brickWidth,0])
            nBrick = createBrick(yCoordPt,brickLength,brickWidth,brickDepth)
            brickList.append(nBrick)
    nextStart = nPt

a = brickList
```

Appendix G- Python code to generate vault (i.e., example 1 with bonding and no additional load)

```
__author__ = "Babita Neupane, bneupane@kent.edu"
__copyright__ = "Copyright (C) 2020 Babita Neupane"
__version__ = "1.0"
__date__ = "2020.05.30"
"""
This code generates Vault for the case where bricks are bonded with motar
It includes following steps:
1. Create the list of bonded length using provided equation
   Using BondingLength method to create the overhanging length for ith brick, i.e. numbrick
2. Generate the list of the bricks for the left side of the arch
3. Generate the list of the bricks for the right side of the arch
4. Generate the list of the bricks in the y-direction
5. Print all the bricks
"""

import rhinoscriptsyntax as rs
import math

#definition for a single brick
#creates the bricks at the position insertPt with dimension (dimX, dimY, dimZ)
#and returns the newBrick
def createBrick (insertPt, dimX, dimY, dimZ):
    pt1 = [insertPt[0], insertPt[1] - dimY, insertPt[2]+dimZ]
    pt2 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]+dimZ]
    pt3 = [insertPt[0]+dimX, insertPt[1], insertPt[2]+dimZ]
    pt4 = [insertPt[0], insertPt[1], insertPt[2]+dimZ]
    pt5 = [insertPt[0], insertPt[1] - dimY, insertPt[2]]
    pt6 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]]
    pt7 = [insertPt[0]+dimX, insertPt[1], insertPt[2]]
    pt8 = [insertPt[0], insertPt[1], insertPt[2]]
    newbrick = rs.AddBox([pt1, pt2, pt3, pt4, pt5, pt6, pt7, pt8])
    return(newbrick)

listofLength = []

#method to get bonding length of the bricks based on the numbricks using nth terms
#refer: thesis document for the equation
def BondingLength(numbrick,weightBrick,bondingStrength,brickWidth,brickLength):
    sum=0
    for i in range(1,numbrick):
        sum = sum+2*i*listofLength[i-1]
    numerator=-numbrick*weightBrick+math.sqrt(math.pow(numbrick*weightBrick,2)
+bondingStrength*brickWidth*weightBrick*(numbrick*numbrick*brickLength- sum))
    denominator=bondingStrength*brickWidth
    bondingLen= numerator/denominator
    listofLength.append(bondingLen)
    return bondingLen


#Starting point
startPts = inPts
#Creating list of bricks
```

```
brickList = []
#to create first brick
newBrick = createBrick (inPts, brickLength, brickWidth, brickDepth)
brickList.append(newBrick)
for j in range(0,M):
    yCoordPt= rs.VectorAdd(inPts, [0,brickWidth*j,0])
    nBrick = createBrick(yCoordPt,brickLength ,brickWidth,brickDepth)
    brickList.append(nBrick)

#create a list of bonded length
bondedList=[]
for i in range(1,N):
    y = BondingLength(i,weightBrick,bondingStrength,brickWidth,brickLength)
    bondedList.append(y)

#Section-1: method to create the arch on the left side of the structure
#Method to stack bricks in the z-direction
for i in range(1,N):
    x = bondedList[(N-1)-i]
    nPt = rs.VectorAdd(inPts, [brickLength-x,0,brickDepth + mortarThickness])
    nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
    brickList.append(nBrick)
    inPts = nPt
#Method to create number of bricks in the y-direction
    for j in range(0,M):
        yCoordPt= rs.VectorAdd(nPt, [0,brickWidth*j,0])
        nBrick = createBrick(yCoordPt,brickLength ,brickWidth,brickDepth)
        brickList.append(nBrick)


#Section-2: method to create an arch on the right side of the structure
#nextStart point as a new starting point
nextStart = rs.VectorAdd(startPts, [(inPts.X-startPts.X)*2+brickLength,0,0])
for j in range(0,M):
    yCoordPt= rs.VectorAdd(nextStart, [0,brickWidth*j,0])
    nBrick = createBrick(yCoordPt,brickLength ,brickWidth,brickDepth)
    brickList.append(nBrick)
#Method to stack bricks in the z-direction
for i in range(1,N):
    x = bondedList[(N-1)-i]
    nPt = rs.VectorAdd(nextStart, [-(brickLength-x),0,brickDepth+mortarThickness])
    nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
    brickList.append(nBrick)
    nextStart = nPt
#Method to create number of bricks in the y-direction
    for j in range(0,M):
        yCoordPt= rs.VectorAdd(nPt, [0,brickWidth*j,0])
        nBrick = createBrick(yCoordPt,brickLength ,brickWidth,brickDepth)
        brickList.append(nBrick)

#to print all the bricks
a = brickList
```

Appendix H- Python code to generate one-way sloped vault (i.e., example 2 with bonding and no additional load)

```
import rhinoscriptsyntax as rs
import math

#definition for a single brick
#creates the bricks at the position insertPt with dimension (dimX, dimY, dimZ)
#and returns the newBrick
def createBrick (insertPt, dimX, dimY, dimZ):
    pt1 = [insertPt[0], insertPt[1] - dimY, insertPt[2]+dimZ]
    pt2 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]+dimZ]
    pt3 = [insertPt[0]+dimX, insertPt[1], insertPt[2]+dimZ]
    pt4 = [insertPt[0], insertPt[1], insertPt[2]+dimZ]
    pt5 = [insertPt[0], insertPt[1] - dimY, insertPt[2]]
    pt6 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]]
    pt7 = [insertPt[0]+dimX, insertPt[1], insertPt[2]]
    pt8 = [insertPt[0], insertPt[1], insertPt[2]]
    newbrick = rs.AddBox([pt1, pt2, pt3, pt4, pt5, pt6, pt7, pt8])
    return(newbrick)

listofLength = []
#method to get bonding length of the bricks based on the numbricks using nth terms
#refer: thesis document for the equation
def BondingLength(numbrick,weightBrick,bondingStrength,brickWidth,brickLength):
    sum=0
    for i in range(1,numbrick):
        sum = sum+2*i*listofLength[i-1]
    numerator=-
numbrick*weightBrick+math.sqrt(math.pow(numbrick*weightBrick,2)+bondingStrength*brickWidth*
weightBrick*(numbrick*numbrick*brickLength - sum))
    denominator=bondingStrength*brickWidth
    bondingLen= numerator/denominator
    listofLength.append(bondingLen)
    return bondingLen

#create a list of bonded length
bondedList=[]
for i in range(1,N):
    y = BondingLength(i,weightBrick,bondingStrength,brickWidth,brickLength)
    bondedList.append(y)

brickList = []

#Method to create an arch
def createArch (inPts, N):
    startPts = inPts
#for creating first brick of left side of the arch at 'inPts'
    newBrick = createBrick (inPts, brickLength, brickWidth, brickDepth)
    brickList.append(newBrick)
#Method to stack bricks
    for i in range(1,N):
        x = bondedList[(N-1)-i]
        nPt = rs.VectorAdd(inPts, [brickLength-x,0,brickDepth + mortarThickness])
        nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
```

```
        brickList.append(nBrick)
        inPts = nPt

#for creating first brick of right side of the arch at 'inPts'
    nextStart = rs.VectorAdd(startPts, [(inPts.X-startPts.X)*2+brickLength,0,0])
    nBrick=createBrick(nextStart,brickLength,brickWidth,brickDepth)
    brickList.append(nBrick)
#Method to stack bricks
    for i in range(1,N):
        x = bondedList[(N-1)-i]
        nPt = rs.VectorAdd(nextStart, [-(brickLength-x),0,brickDepth+mortarThickness])
        nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
        brickList.append(nBrick)
        nextStart = nPt
    return startPts

#Creating arches with decreasing number of brick courses
for i in range(N,M,-1):
    inPts=  rs.VectorAdd(inPts, [0,brickWidth,0])
    createArch(inPts,i)

#to print all the bricks
a = brickList
```

Appendix I- Python code to generate two-way sloped vault (i.e., example 3 with bonding and no additional load)

```
import rhinoscriptsyntax as rs
import math

#definition for a single brick
#creates the bricks at the position insertPt with dimension (dimX, dimY, dimZ)
#and returns the newBrick
def createBrick (insertPt, dimX, dimY, dimZ):
    pt1 = [insertPt[0], insertPt[1] - dimY, insertPt[2]+dimZ]
    pt2 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]+dimZ]
    pt3 = [insertPt[0]+dimX, insertPt[1], insertPt[2]+dimZ]
    pt4 = [insertPt[0], insertPt[1], insertPt[2]+dimZ]
    pt5 = [insertPt[0], insertPt[1] - dimY, insertPt[2]]
    pt6 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]]
    pt7 = [insertPt[0]+dimX, insertPt[1], insertPt[2]]
    pt8 = [insertPt[0], insertPt[1], insertPt[2]]
    newbrick = rs.AddBox([pt1, pt2, pt3, pt4, pt5, pt6, pt7, pt8])
    return(newbrick)

listofLength = []
#method to get bonding length of the bricks based on the numbricks using nth terms
#refer: thesis document for the equation
def BondingLength(numbrick,weightBrick,bondingStrength,brickWidth,brickLength):
    sum=0
    for i in range(1,numbrick):
        sum = sum+2*i*listofLength[i-1]
    numerator=-
numbrick*weightBrick+math.sqrt(math.pow(numbrick*weightBrick,2)+bondingStrength*brickWidth*
weightBrick*(numbrick*numbrick*brickLength - sum))
    denominator=bondingStrength*brickWidth
    bondingLen= numerator/denominator
    listofLength.append(bondingLen)
    return bondingLen

#create a list of bonded length
bondedList=[]
for i in range(1,N):
    y = BondingLength(i,weightBrick,bondingStrength,brickWidth,brickLength)
    bondedList.append(y)

brickList = []

#Method to create an arch
def createArch (inPts, N):
    startPts = inPts
#for creating first brick of left side of the arch at 'inPts'
    newBrick = createBrick (inPts, brickLength, brickWidth, brickDepth)
    brickList.append(newBrick)
#Method to stack bricks
    for i in range(1,N):
        x = bondedList[(N-1)-i]
        nPt = rs.VectorAdd(inPts, [brickLength-x,0,brickDepth + mortarThickness])
        nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
```

```
        brickList.append(nBrick)
        inPts = nPt

#for creating first brick of right side of the arch at 'inPts'
    nextStart = rs.VectorAdd(startPts, [(inPts.X-startPts.X)*2+brickLength,0,0])
    nBrick=createBrick(nextStart,brickLength,brickWidth,brickDepth)
    brickList.append(nBrick)
#Method to stack bricks
    for i in range(1,N):
        x = bondedList[(N-1)-i]
        nPt = rs.VectorAdd(nextStart, [-(brickLength-x),0,brickDepth+mortarThickness])
        nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
        brickList.append(nBrick)
        nextStart = nPt
    return startPts

#Method to create arches in one-direction
newinPts= inPts
#Creating arches with decreasing number of brick courses
for i in range(N,M,-1):
    inPts=  rs.VectorAdd(inPts, [0.5,brickWidth,0])
    createArch(inPts,i)
#for arches in opposite direction from the start point
newinPts=  rs.VectorAdd(newinPts, [0,brickWidth,0])
#Creating arches with decreasing number of brick courses
for i in range(N,M,-1):
    newinPts=  rs.VectorAdd(newinPts, [0.5,-brickWidth,0])
    createArch(newinPts,i)

#to print all the bricks
a = brickList
```

Appendix J- Python code to generate wavy vault (i.e., example 4 with bonding and no additional load)

```
import rhinoscriptsyntax as rs
import math

#definition for creating a brick
def createBrick (insertPt, dimX, dimY, dimZ):
    pt1 = [insertPt[0], insertPt[1] - dimY, insertPt[2]+dimZ]
    pt2 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]+dimZ]
    pt3 = [insertPt[0]+dimX, insertPt[1], insertPt[2]+dimZ]
    pt4 = [insertPt[0], insertPt[1], insertPt[2]+dimZ]
    pt5 = [insertPt[0], insertPt[1] - dimY, insertPt[2]]
    pt6 = [insertPt[0]+dimX, insertPt[1] - dimY, insertPt[2]]
    pt7 = [insertPt[0]+dimX, insertPt[1], insertPt[2]]
    pt8 = [insertPt[0], insertPt[1], insertPt[2]]
    newbrick = rs.AddBox([pt1, pt2, pt3, pt4, pt5, pt6, pt7, pt8])
    return(newbrick)

#list of bonding lengths
listofLength = []
#definition for bonding length
def BondingLength(numbrick,weightBrick,bondingStrength,brickWidth,brickLength):
    sum=0
    for i in range(1,numbrick):
        sum = sum+2*i*listofLength[i-1]
    numerator=-
numbrick*weightBrick+math.sqrt(math.pow(numbrick*weightBrick,2)+bondingStrength*brickWidth*
weightBrick*(numbrick*numbrick*brickLength - sum))
    denominator=bondingStrength*brickWidth
    bondingLen= numerator/denominator
    listofLength.append(bondingLen)
    return bondingLen


#creating a list of overhanging lengths
bondedList=[]
for i in range(1,N):
    y = BondingLength(i,weightBrick,bondingStrength,brickWidth,brickLength)
    bondedList.append(y)

brickList = []
#definition to create an arch
def createArch (inPts, N):
    startPts = inPts
    #for first brick of first half of the arch
    newBrick = createBrick (inPts, brickLength, brickWidth, brickDepth)
    brickList.append(newBrick)

    for i in range(1,N):
        x = bondedList[(N-1)-i]
        nPt = rs.VectorAdd(inPts, [brickLength-x,0,brickDepth + mortarThickness])
        nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
        brickList.append(nBrick)
        inPts = nPt
```

```
    #for second half of the arch
    #to create new start point
    nextStart = rs.VectorAdd(startPts, [(inPts.X-startPts.X)*2+brickLength,0,0])
    newBrick = createBrick (nextStart, brickLength, brickWidth, brickDepth)
    brickList.append(newBrick)
    for i in range(1,N):
        x = bondedList[(N-1)-i]
        nPt = rs.VectorAdd(nextStart, [-(brickLength-x),0,brickDepth+mortarThickness])
        nBrick = createBrick(nPt,brickLength,brickWidth,brickDepth)
        brickList.append(nBrick)
        nextStart = nPt
    return startPts

for i in range(N,M,-1):
    inPts =  rs.VectorAdd(inPts, [0.5,brickWidth,0])
    createArch(inPts,i)
for i in range(M,N):
    inPts =  rs.VectorAdd(inPts, [0.5,brickWidth,0])
    createArch(inPts,i)

a = brickList
```