

Designing Performative Surfaces
Computational Interpretation of Flow Pattern Drawings

by

Masoud Akbarzadeh

M. Arch Massachusetts Institute of Technology 2011
M. Sc. in Earthquake and Dynamics of Structures, Iran University of Science and Technology, 2007
B. Sc. in Civil and Environmental Engineering, Zanzan University 2004

Master of Science in Architecture Studies

at the

Massachusetts Institute of Technology

June 2012

©2012 Masoud Akbarzadeh, All rights reserved.


The author hereby grants to MIT permission to reproduce
and to distribute publicly paper and electronic versions
of this thesis document in whole or in part.

Signature of Author :

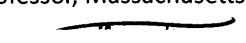


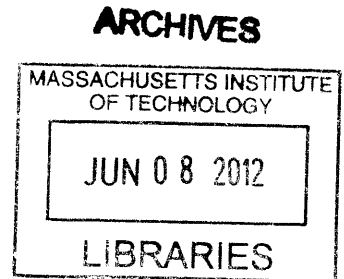
Department of Architecture
May 24, 2012

Certified by:


Takehiko Nagakura
Associate Professor, Massachusetts Institute of Technology
Thesis Advisor

Accepted by:


Takehiko Nagakura
Associate Professor, Massachusetts Institute of Technology
Chair of the Department Committee on Graduate Students



Dennis Shelden
Associate Professor, Massachusetts Institute of Technology
Thesis Reader

Joel Lamere
Lecturer, Massachusetts Institute of Technology
Thesis Reader

Acknowledgment

I am thankful to:

Takehiko Nagakura, for his great comments and support at the beginning and through the whole process of this research, Dennis Shelden, For his generous helps and comments on generalizing the mathematical grammar for the research, Joel Lamere, for his important suggestions to frame the research in the boundaries of design. I am also thankful to his generosity to use the tool developed in this thesis for a design purpose and sharing the final result as a design instance, a representative of a complex geometry generated by the tool, Paul Kassabian, for his great structural consultation throughout the research, Morteza Zadimoghaddam, for his great help clarifying some computer science related algorithms to be used in this research, Onur Gun, for his great comments on simplification of ideas presented in this research, Alan Tai, for his great helps on developing some of the algorithms in this research, William O' Brien Jr, for his great design related suggestions, and Ksenia, for her generosity, support and patience throughout the process of this research.

Table of Contents

1. Introduction	8
1.1 Motivation	
1.2 On geometry and Architecture	
1.3 Surface representation	
1.4 Problem Statement	
2. Methodology	14
2.1 Introduction	
2.2 Surface Data Structure Background	
2.3 Regenerating Surface using Surface Network Graph	
2.4 Properties of Surface Network Module	
2.5 per-formative Surface Generative Algorithms	
3. Surface Data Structures	19
3.1 Introduction	
3.1.1 Surface Definition: History	
3.1.2. Surface Definition: Mathematical Representation	
3.2 Surface Network Extraction Methods	
3.2.1 Bilinear Surface Patches Algorithm	
3.3. Regenerating Surface Using Surface Network Graphs	
3.3.1. Contour extraction	
3.3.2. Contour regeneration	
3.3.3. Topography Resolution	
3.4. Summary	
4. Generating Performative Surfaces	38
4.1. Introduction	
4.2. Performative Aggregation of Critical Graph	
4.3. Surface Generative Algorithm Version 1.	
4.4. Surface Generative Algorithm Version 2.	
4.5. Surface Generative Algorithm Version 3.	
4.5.1. Rationalized drainage direction of a surface	
4.5.2. Algorithm Description	
4.5.3. Global Drainage	
4.5.4. Point Grid Pre-Transformation	
4.5.5. Non-linear Transformation of height	
4.6. Summary	
5- Conclusions	68
6- References	70
7- Appendix	76

Chapter One: Introduction

1.1 Motivation

In spring 2011, while I was working on my thesis in architectural Design degree, I came across with an interesting problem in design: a parametric river. I realized that it is not possible to control the river parameters without understanding the geometry of the surface of terrain. In other words, the shape of the terrain or topography may change the shape of the river down the hills. I started to look up more examples in geoscience and geomorphology to find out more about this topic. I came across drainage patterns which vary based on the shape of the terrain in different parts of the world [Howard 1967].

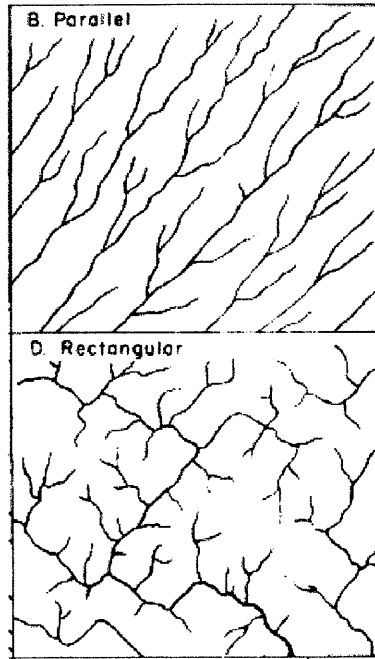


Figure 1.0. Arthur David Howard, Drainage Analysis in Geologic Interpretation A Summation [Howard 1967].

As a designer, the first thought passed through my mind was: “is it possible to design a terrain using drainage patterns?! There must be a way to derive the landscape geometry from the one of the the river!” Later on, through searching related topics in geoscience, I realized that this topic has interested researchers from 1858 and there is a quite enormous body of research on that in geo-computation and geography and computer science.

I made this topic as the main goal of present thesis to explore the design possibility of such representation in architecture and connecting the world of design with hydrological and geological characteristics of the land (Fig. 1.0).

Recently the design proposals tend to become more engaged in sustainability aspects, more recently in energy generation. Therefore, many designers now seek approaches to integrate architectural ideas with interdisciplinary subjects to tackle the different aspects of energy constrains and sustainability issues. There is a recently developed area of research among architects which tries to define the design through the lenses of energy production. This field has received more attention in landscape design and planning strategies. Among all energy generating methods such as wind and solar, there are no many examples of addressing the design through hydropower energy generation which is the main basis of investigation in current study.

In order to explain the goals of the thesis it is important to clarify the objectives of this study in a simple question: Is it possible to construct complex geometry of the surface of the terrain using drainage analysis? Or is it possible to embed required information of 3-dimentional space into 2-dimentional drawing. In that case, designers can design complex geometries using simple plan drawings which might result in more function-oriented design.

1.2 On Geometry and Architecture

“Geometry is one subject, architecture another, but there is geometry in architecture. Its presence is assumed much as the presence of mathematics in physics, or letters in words. Geometry is understood to be a constitutive part of architecture, indispensable to it, but not dependent on it in any way. The elements of geometry are thus conceived as comparable to the bricks that make a house, which are reliably manufactured elsewhere and delivered to site ready for use. Architects do not produce geometry, they consume it. “

Robin Evans, 'The Projective cast' [Evans 1995]

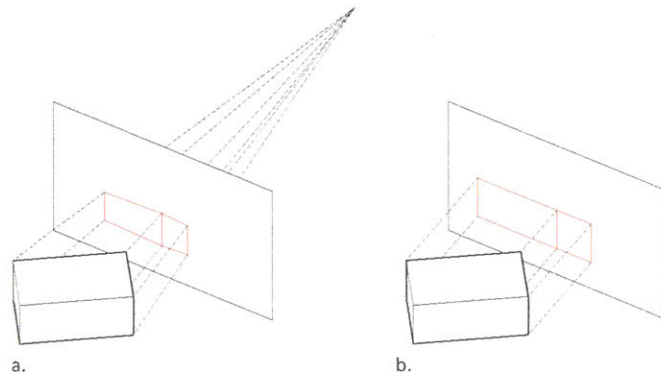
Before any step forward in geometric exploration, it is quintessential to step back and realize the relationship between geometry and architecture through the history of architecture. If I want to define the place of geometry with respect to architecture, I should say geometry is completely independent from architecture. It is a rational science which is manufactured somewhere else and consumed by architects. Geometry is a rational science, whereas architecture is a kind of art which is produced and judged by intuition. In other words, geometry gives architecture a rational foundation to start, but it does not limit it to pure rationality.

The first place anyone searches geometry in architecture is in the shape of the buildings. What establishes the shape of the building in our perception is its projection into our eyes. Projection is the process of producing an image of an object on a planar surface. Projection extensively exists in architectural drawings or representations.

Generally, there are two types of projections in drawings: parallel and central (conical). What is in common for both is that both use lines of sight to project the object onto the projection plane. The difference is that in parallel projection the lines are parallel, whereas in conical projection they converge to the point of view (focal point). (Fig. 1.1.)

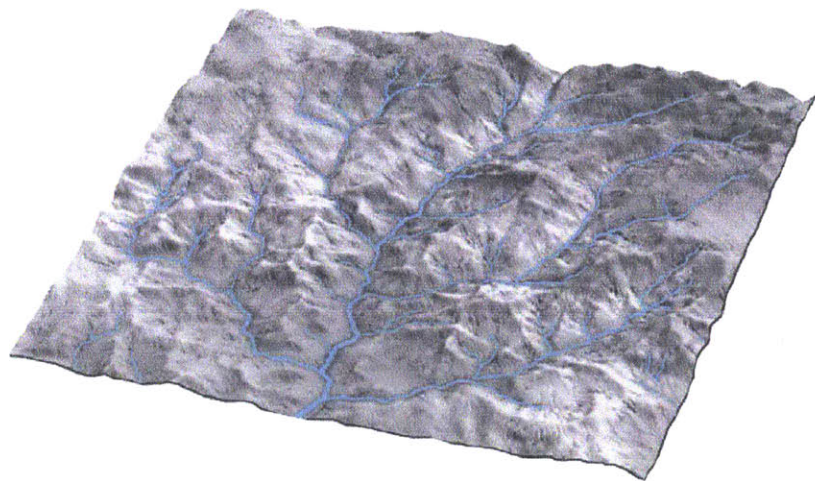
Normal orthographic projection of an object is a branch of parallel projection which we use in everyday design tasks, called plan, section, and elevation. In order to describe the object fully in three dimensions, three projection planes are required, perpendicular on each other. Obviously, this is the reason for having at least three projection planes. Previous to computers, all these representations have been done, in an analogue style, on paper and presented two dimensionally as well. Preserving the same concept, computers revolutionized this approach and presentation styles.

Figure 1.1. a. Conical Projection
b. parallel projection



Advance in computer graphics and visualization allows modelers to design and represent highly complicated geometrical models. Lots of these geometries are regular geometries that can be articulated in computer easily using conventional techniques. But there are many other irregular shapes of practical important to us which cannot be represented easily. For example, topography of the earth is one. How do we represent the properties of such complex geometries? This is a desired subject for researchers in different disciplines including geometers, geographers, computer scientist and even architects! (Fig. 1.2.)

Figure 1.2. Surface geometry of the Earth, dendritic drainage pattern [A].



1.3 Surface Representation

Since current research is intended for designers and architects, Lets have a look at main different methods of surface construction and representation in different disciplines from design point of view. This overview helps us understand the characteristics of each method and its potentials and drawbacks in terms of design.

1.3.1 Interpolation of Sectional Curves

In this technique, surface is constructed using multiple sections in two main directions of u, v or s, t. This is the main principle used in constructing continuous surfaces in NURBS¹ modeling softwares. The main advantage of this technique is generation of precise and manipulatable surfaces in section. The disadvantage of this technique is poor control in plan as well as large number of required sectional curves in constructing complex surfaces. (Fig. 1.3.)

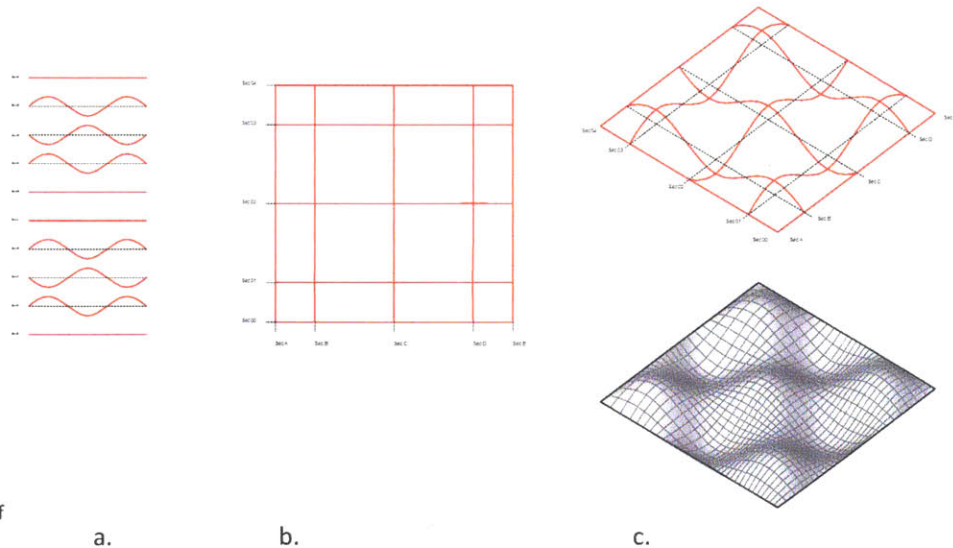
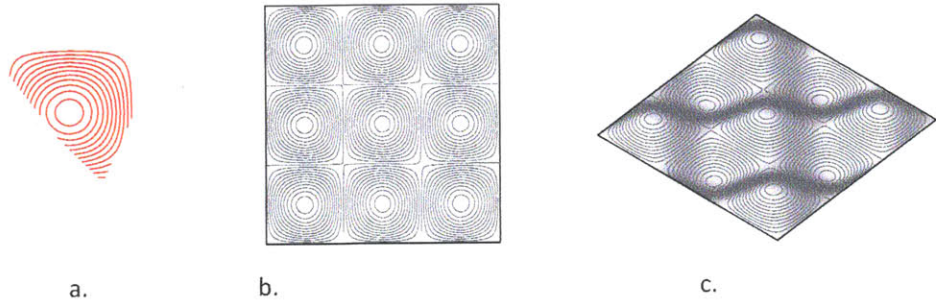


Figure 1.3. Interpolation of Sectional Curves. a. Design curves / Required curves to start. b. Plan location of each sectional curve. c. Interpolated result/ Surface of the curves.

1.3.2. Contour Manipulations

In this technique, designer is obliged to draw the contour representation of the final surface on plan. Contour lines are basically intersection lines between the final surface and parallel planes with specific intervals from defined origin. The advantage of this method is a good control for designers in plan. The disadvantages of this method are: poorly manageable features in section, required time and effort to draw all the contours to represent and construct a complete surface. (Fig. 1.4.)

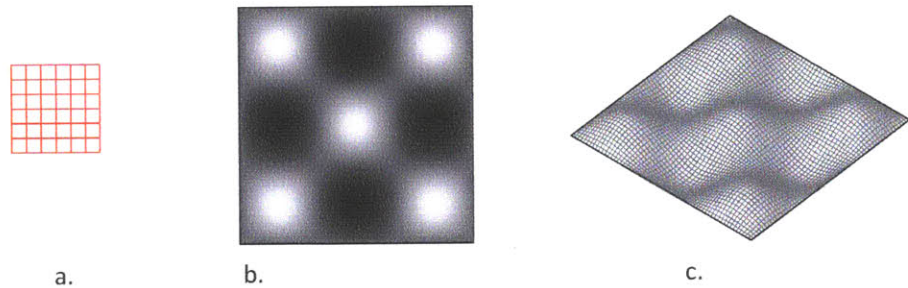
Figure 1.4. Contour manipulation
 a. design requirement elements.
 b. plan representation of con-
 toured based design. c. Surface
 result of the contour drawn plan



1.3.3. Digital Elevation Model

Digital Elevation Model (DEM) is a technique in surface reconstruction which uses pixel properties of an images. This method is highly interested for Geographers and geo-computationalist. The color of each pixel, which changes in a range from black to white, is translated to the height and consequently the surface is generated using height information of each pixel. The main advantage of this method is surface generation of image representation. The main disadvantage of this technique is the difficult control of surface behavior in plan and section (Fig. 1.5.)

Figure 1.5. Digital Elevation
 Model representation of a
 surface. a. Design elements
 as pixels and their colors. b. Image
 representation of surface
 as pixels rangnig from baclk
 to white. c. Surface result of
 elevation data



1.3.4. Triangulation

Computer graphists use triangulationⁱ extensively in design and representation. It is a very powerful technique in constructing complex geometries. This method translate geometry to collection of vertices and creates triangles from them. The advantage of this method is the possibility of representing very complex geometries using simple triangles, whereas, the disadvantages are the enormous amount of time required to design a geometry and poor global manipulation of the complex geometries. (Fig. 1.6.)

ⁱ The triangulation is named after Boris Delaunay for his work on this topic from 1934.

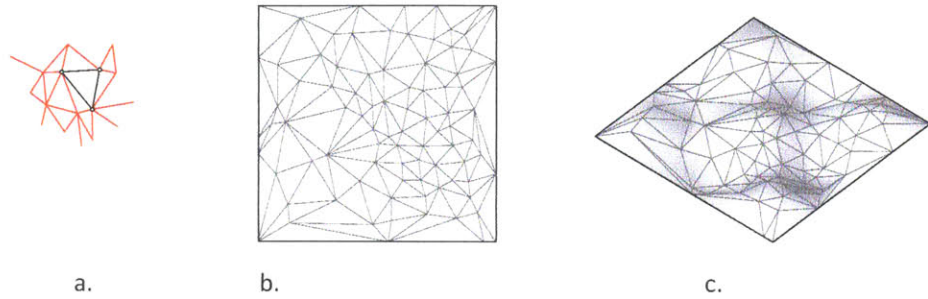


Figure 1.6. Triangulation. a. Vertices and their corresponding triangles. b. Plan representation of triangulated vertices. c. Mesh Surface Representation of triangles.

1.4. Problem Statement

As I referred to geomorphological drawings of the surface of the earth, there is a conventional method of drawings in geomorphology which represents different complex surface geometries of the earth. now that we revisited main surface design and representation methods in different disciplines, it is the time to restate the problem for the current research:

Is it possible to design and construct complex surface geometries using only plan drawings? in Other word, is it possible to use the drainage pattern of a surface to reconstruct its geometry?

The main intention of this research is to answer this question from designers' point of view and provide them a tool for designing such surfaces using only plan drawings. (Fig. 1.7)

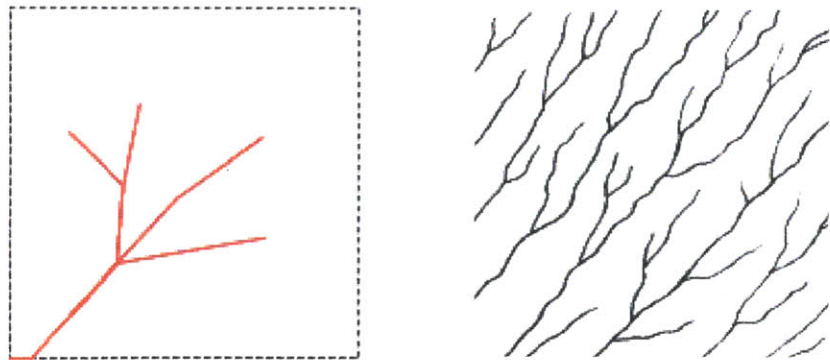


Figure. 1.7. Plan drawings of a drainage pattern

Chapter Two: Methodology

2.1. Introduction

The main outcome of this research is a tool and its relevant algorithm for designers to design and generate complex geometries using only plan drawings.(Fig. 2.1.) The algorithm to generate this tool has been achieved through series of surface generation processes using the idea of simple plan drawings. In order to conduct the research, author takes the following steps.

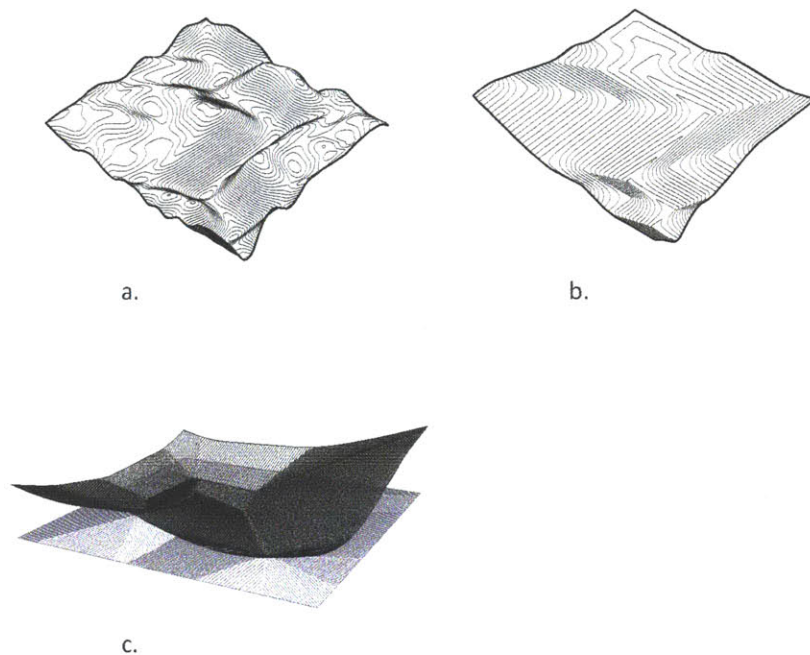


Figure. 2.1. Performative surface algorithms. a. Algorithm 1. b. Algorithm 2. c. Algorithm 3.

2.2. Surface data structure background

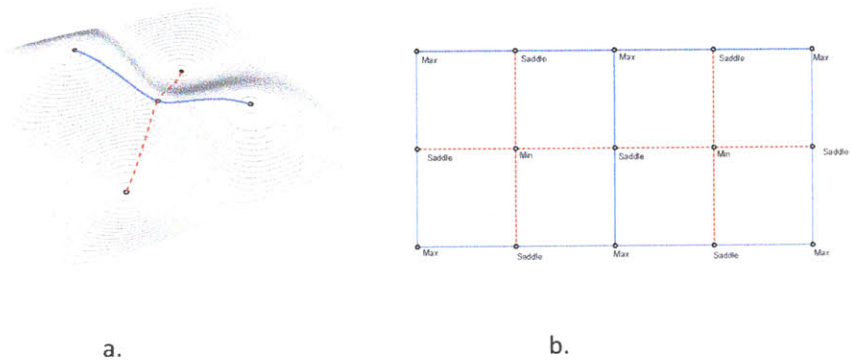
The main idea of this step is to reduce the whole information of a surface into a graph consisting of points and lines. This will simplify the explanation of any complex surface topography into a simple graph representation. (Fig.2.2)

Initially, author starts with the definition of surface data structure, historically [Cayley 1859, Maxwell 1870] and mathematically [Morse 1965]. In Mathematics, Surface data structure is called Critical graph which is a basis for *surface networks* topic in geo-computation. Respectively, the geo-computational methods for extraction of surface network graph from a given geometry or topography is used to understand the behavior of different surface geometries. [Pfaltz 1976, Schneider. B. and Jo wood 2004].

Assumptions

- In surface network extraction, Each surface or topography is a continuous function, $z = f(x,y)$. There is no whole or under cut in the geometry of the assumed surface
 - Extraction of surface network graph is based on bilinear surface patches
- Surface structure extraction*

Figure 2.2. a. Critical Graph consisting of maximum, minimum and saddle points and the lines connecting these points to each other. b. simplified version of critical graph



2.3. Regenerating Surface using Surface Network Graph

The importance of this step is to learn to construction or regeneration of different types of surface geometry using simple surface network graph concept. The general graph of surface networks is used as a basis to regenerate different surface geometries. In this step algorithm to extract contours from graphs is provided.(Fig. 2.3)

Assumptions

- There are different types of surface network graphs in geo-computation. The graph that has been used in this step is based on the simple connection between maximum points to pass points, and pass points to minimum points of a surface.
- In this study contours are generated from the graph. An algorithm to organize and connect the contours provided by the author
- A secondary algorithm also provided to reshape the contours and regenerating multiple surfaces from a single surface network graph

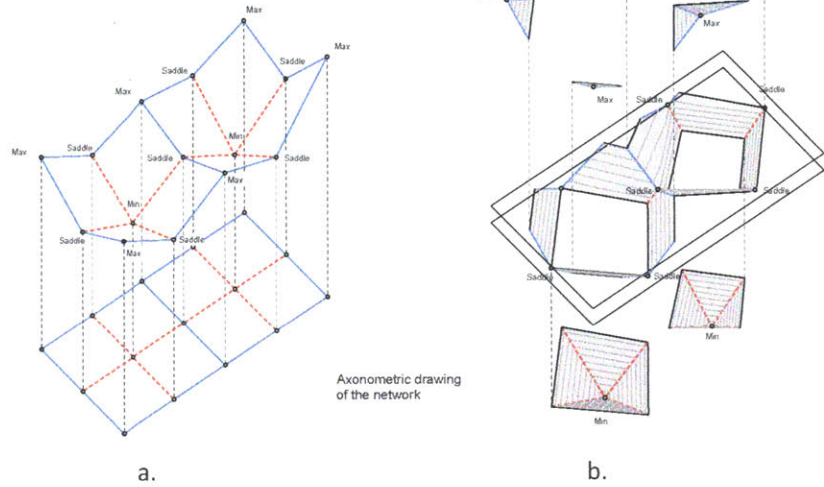


Figure 2.3. a. Surface Network Graph b. Contour Extraction algorithm from Surface Network Graph

2.4. Properties of surface network module

In order to design performative surfaces, it is necessary to understand the properties of a surface generated from different types of network graphs. In this step author starts to change the parameters of the network graph to observe the change in behavior of the module in surface generation process. This step is the basis for the author to arrive to the surface regeneration algorithms using simple plan drawings (Fig. 2.4).

Assumptions

- The properties of each module are investigated with respect to idea of flow of water on the surface. Consequently the network graph with the directionality of flow is chosen as a basis for further generative algorithms.

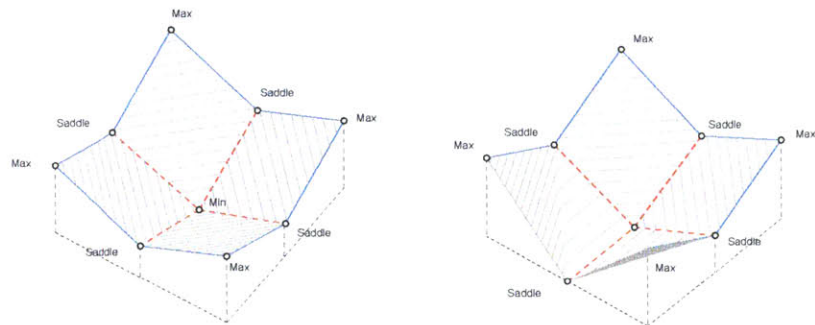


Figure 2.4. a. Surface Network Module b. Surface Network Module Transformed

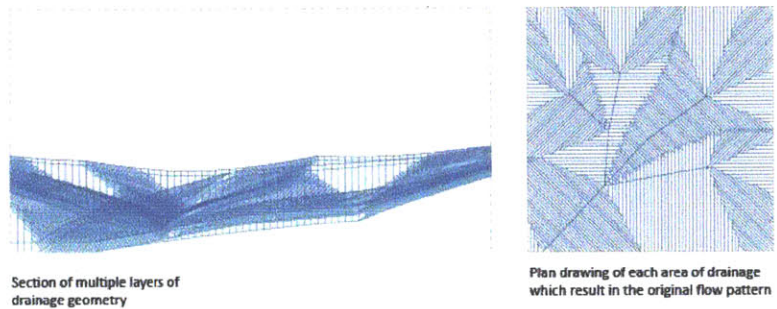
2.5. Performative Surface generative Algorithms

In this step three main algorithms are provided to reconstruct the surface from plan drawings. The algorithms are chronologically related, meaning that the first algorithm is considered as the ancestor for the following ones. The main direction in all algorithms is to use the plan drawing as a basis to transform the rest of the surface (Fig. 2.5)

Assumptions

- In all algorithms the surface is a result of transformation of 2 dimensional point grids into three dimensional point grids
- Surface generation is based on the contour extraction algorithm developed by author in the previous section. The interpolation of contours and generation of continuous field should be achieved by existing tool in geometric modeling software.
- The definition of a surface in all these algorithms is different from a continuous function with all its corresponding points. Instead a surface data structure is provided as a result. Turning this data structures into a continuous surface requires further algorithms and tools which is beyond the scope of this research.

Figure 2.5. Surface Generation Algorithm 3.



Chapter Three: Surface Data Structures

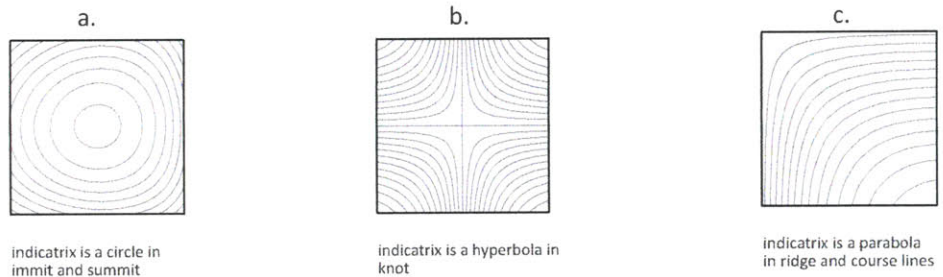
3.1 Introduction

3.1.1 Surface Definition: History

Prior to inventing a new geometrical methodology to generate complex surfaces, it is necessary to obtain a good understanding of a geometry and definition of surface from different points of views. The author of this research is interested in The definition of the surface data structure historically and mathematically to find a way to describe complex surfaces with simple graphs. In this respect, through this chapter, he will explore the idea of surface data structure in mathematics and geo-computation to build a basis for regenerating a surface using only plan drawings.

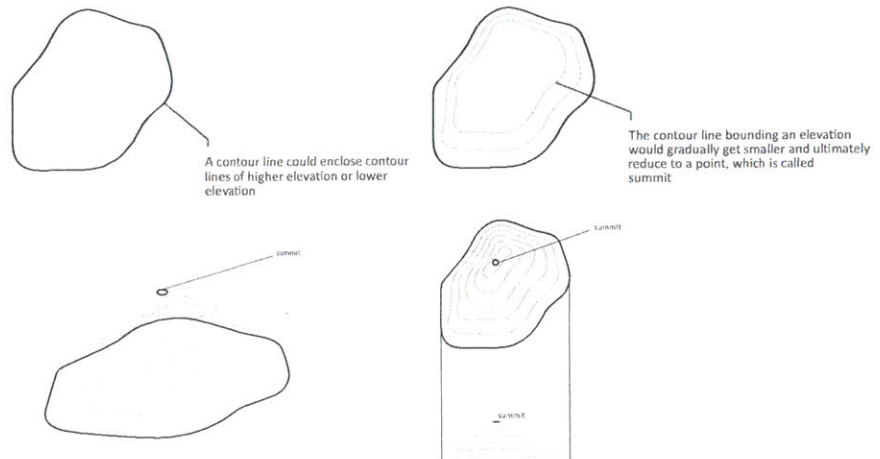
Probably Cayley, in 1859, [Cayley 1859] was the first person to describe the surface data structure. He defines different areas of a surface based on the term indicatrix or contours. He provides a grammar with his explanation and describes the properties of the most important points of a topography. (Fig. 3.1)

Figure 3.1. a. Circular Indicatrix b. Hyperbolic Indicatrix c. Parabolic Indicatrix



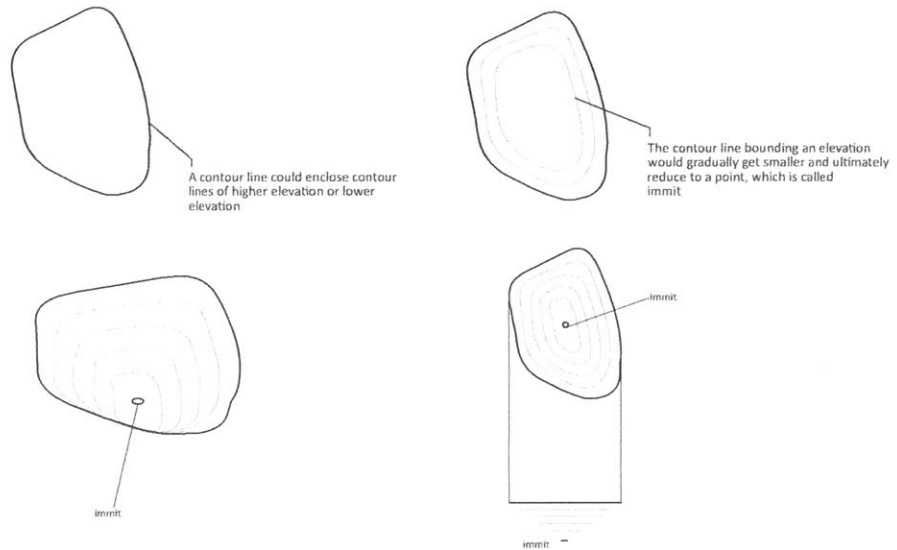
According to Cayley There are three types of indicatrix in the whole area of a continuous topography: circular, Hyperbolic and parabolic. He defines each indicatrix as a closed curve which encompasses other indicatrix inside itself. By the time that we travel up in z direction, the indicatrix becomes smaller and smaller till it turns into a point called Peak or Maximum (Fig. 3.2).

Figure 3.2. a. indicatrix or contour is a closed curve b. It encompasses other smaller indicatrices inside c. By Travelling in z direction the indicatrices become smaller and smaller and finally become a peak point



Based on the same notion, if travelling in the z direction down, the indicatrices become smaller and smaller till become the local pit point or minimum. The indicatrix around the local minimum and maximum is circular. There is a point in topography where three indicatrices meet one indicatrix. This point is called the saddle point (Fig. 3.3).

Fig. 3.3. a. indicatrix or contour is a closed curve b. It encompasses other smaller indicatrices inside c. By Travelling in (- z) direction the indicatrices become smaller and smaller and finally become a minimum point



The indicatrix in this point is hyperbola. If we move from a saddle point upward we will arrive to local maximum and if we move down ward we will arrive to local minimum. The indicatrix along the path which connects the saddle point to maximum and minimum is parabola. (Fig. 3.4)

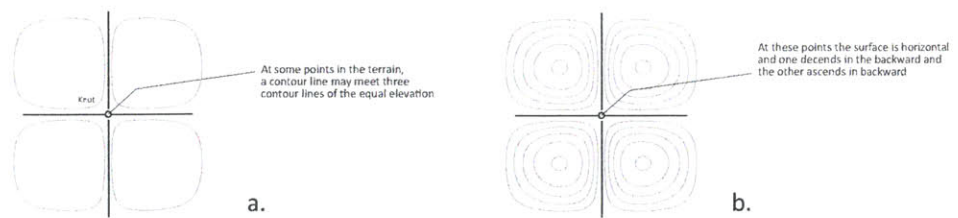


Figure. 3.4. a. There is a point in topography where four indicatrices meet each other. b. the contours in this point look like hyperbola c. Saddle point of topography

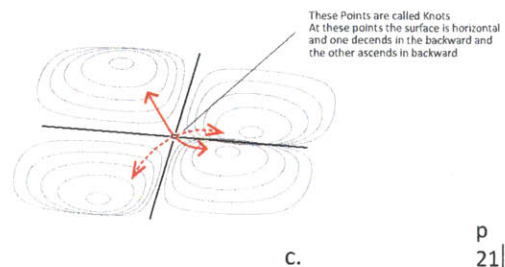
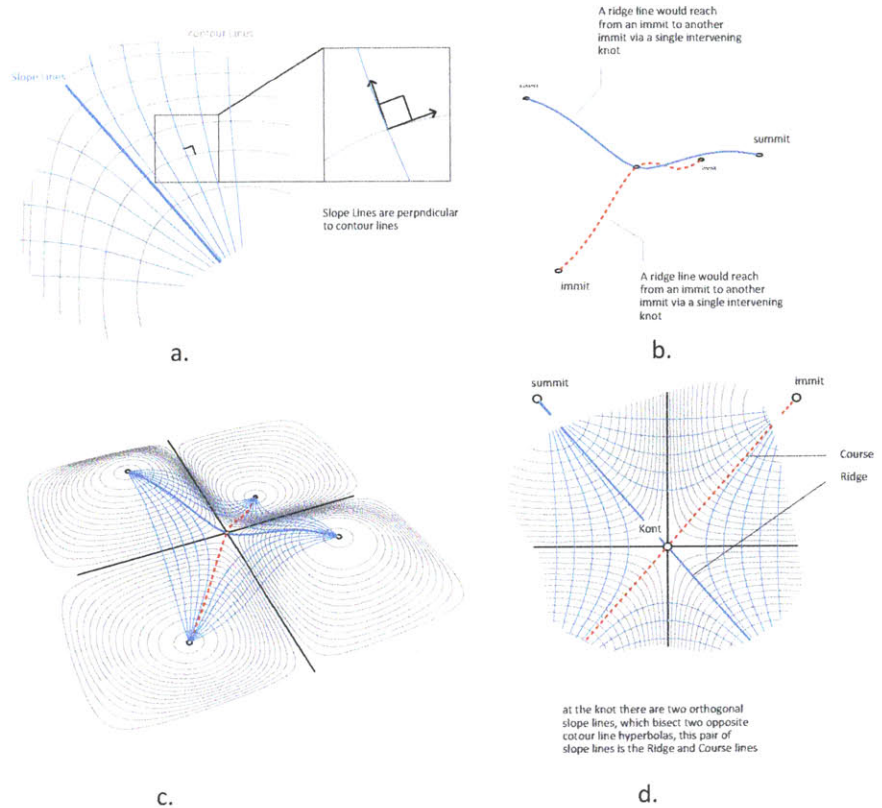


Figure. 3.5. a. The drainage patterns are always perpendicular on contour lines b. There is only one path exist, If we travel from saddle point with steepest paths upward or downward which connect saddle point to maxima and minima. c. series of drainage paths on the topography. d. plan view of slope paths and contour paths



Cayley also mentions that if moving from the saddle point upward with the steepest slope you will reach to local maxima and moving downward will take us to the local minima. This path always perpendicular on contour lines. If we start from the local maxima or local minima try to go up or down in multiple directions and in each direction we travel based on the steepest path, we will have a family of paths which are perpendicular on contours. These paths are called water drainage patterns. (Fig. 3.5.)

Maxwell in 1870 completed the intuitive description of surface data structures [Maxwell 1870]. He divided the whole topography into hills and dales. In describing these notions, he points out the local maxima and minima and saddle points and the lines which connect these points to each other. Then he explains that if he defines dale or valley as an area which is surrounded by three local maxima. As a result, if we travel from one maximum to saddle and from saddle to another maximum and again to saddle and to the third maximum, we will establish a valley. There is a mathematical relationship between the number of maxima and saddle points, according to Maxwell. (Fig. 3.6, 3.7)

The number of peaks minus number of passes equals to one and number of pits (minima) minus number of passes (saddle) equals one:

$$\text{Peak} - \text{Saddle} = 1$$

$$\text{Pits} - \text{Saddle} = 1$$

$$\text{Peaks} + \text{Pits} - \text{Saddle} = 2$$

Figure 3.6. a, b. Local maxima and local minima and saddle points on the topography. c, d dale or valley. e, f. Hills

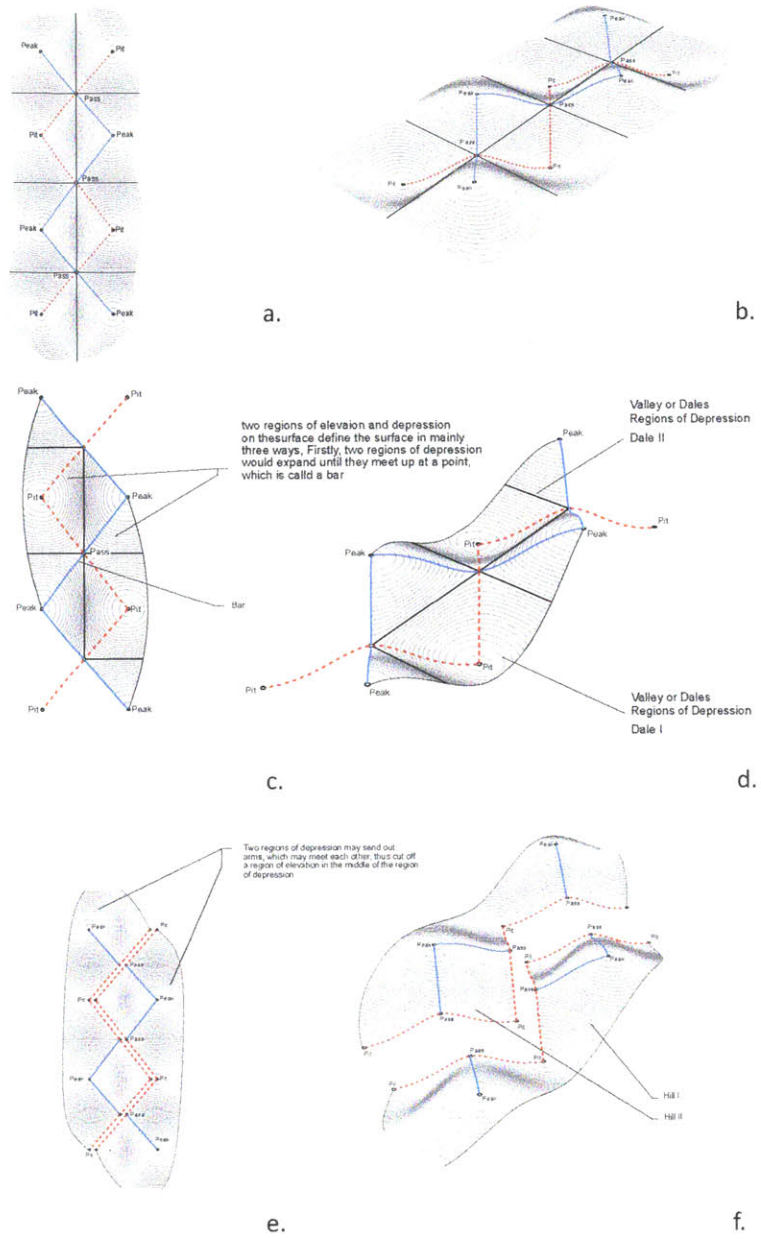
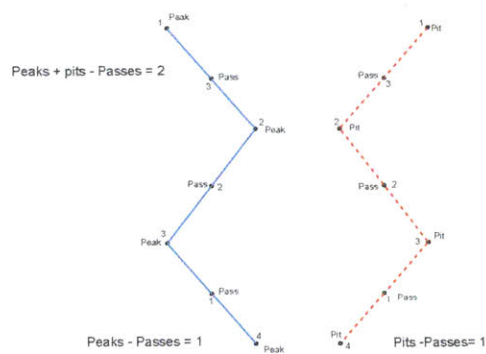


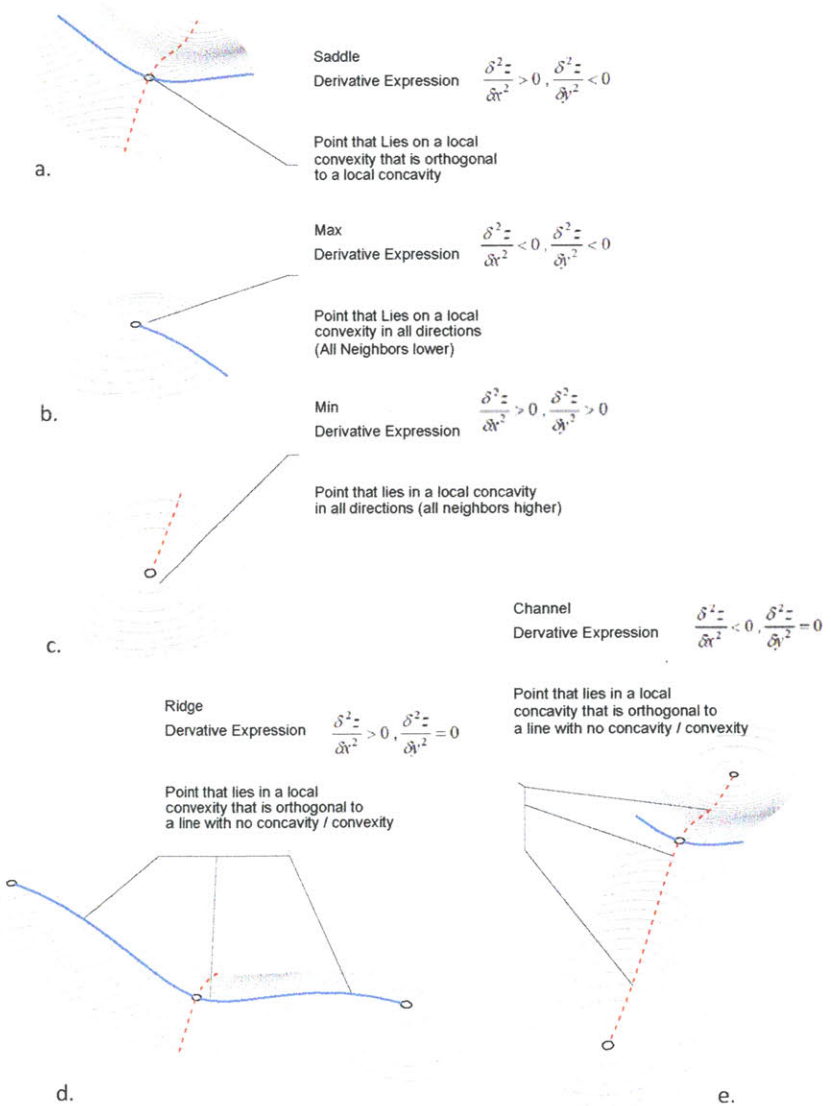
Figure 3.7. Relationship between Peak, Pits and passes (Maxima, minima and Saddle points)



3.1.2 Surface Definition: Mathematical Representation

After Cayley and Maxwell, Morse in 1945 [Morse 1965], presented the mathematical definition for important points on the surface and their connecting graph. According to Morse, the second derivative of the surface in two major directions of its curvature are responsible for establishing the critical points on the surface. If traveling from saddle point to each local extremum points, we should always travel along a path on which always one of the primary curvatures of the surface is zero. (Fig. 3.8)

Figure.3.8. a. Saddle point Mathematical definition. b. Local Maximum. c. Local Minimum d. Coarse line, the line which connects saddle point to maximum points. e. Ridge lines which connects saddle points to minima

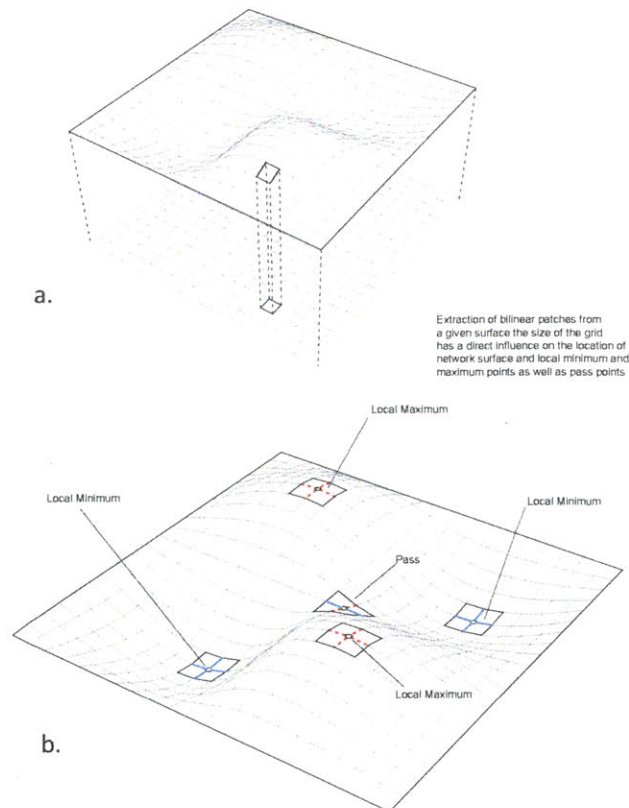


3.2. Surface Network Extraction Methods

Critical graph is the basis for lot of areas of research in GIS and geo-computation. Researchers in these fields use critical graph ideas to extract the most information of the complex surface of the earth and represent it with simple graph ideas. This helps them reduce the amount of space and effort they need to store the information about the geometry of the terrain. In current research, author visited the idea of surface network extraction to earn the different properties of surfaces with respect to network representation of it. Consequently, chosen surface network extraction method is after [Schneider. B. and Jo wood 2004].

This method is called extraction from bilinear surface patches. In this method the whole surface geometry is subdivided into point grids. For this reason a 2 dimensional point grid is projected onto the surface. The reason for this is to have square modules of surface patches with corners sitting on the geometry of the original surface. There is a difference between the method used in this experiment and the method which is used in geo-computation field. In geo-computation there is no surface exists at the begging and the process of extraction is based on digital elevation model of a terrain. (Fig. 3.9)

Figure. 3.9. a. Projecting a point grid onto the geometry of a surface to construct square surfac patches
b. locations of extermum points of the surface as well as the saddle points



The reason for choosing the bilinear surface patches is that the properties of extremum points of a surface can be easily discovered. In order to do so, each surface patch is compared with its surrounding neighbors to define the maximum, minimum or saddle points. For maximum and minimum points the property of the surface patches is quite easy: if the central vertex has a height bigger than its surrounding neighbors, then the point is maximum. If the height is smaller than that of surrounding neighbors, then the point is minimum.

For saddle point there are two possibilities: first, there is a possibility that the saddle point happens on the grid. This means that the neighbors are higher and lower alternatively. Second, the saddle point might happen in the middle of the surface patch. This means that the corners of the surface patch is alternatively higher and lower. (Fig. 3.10)

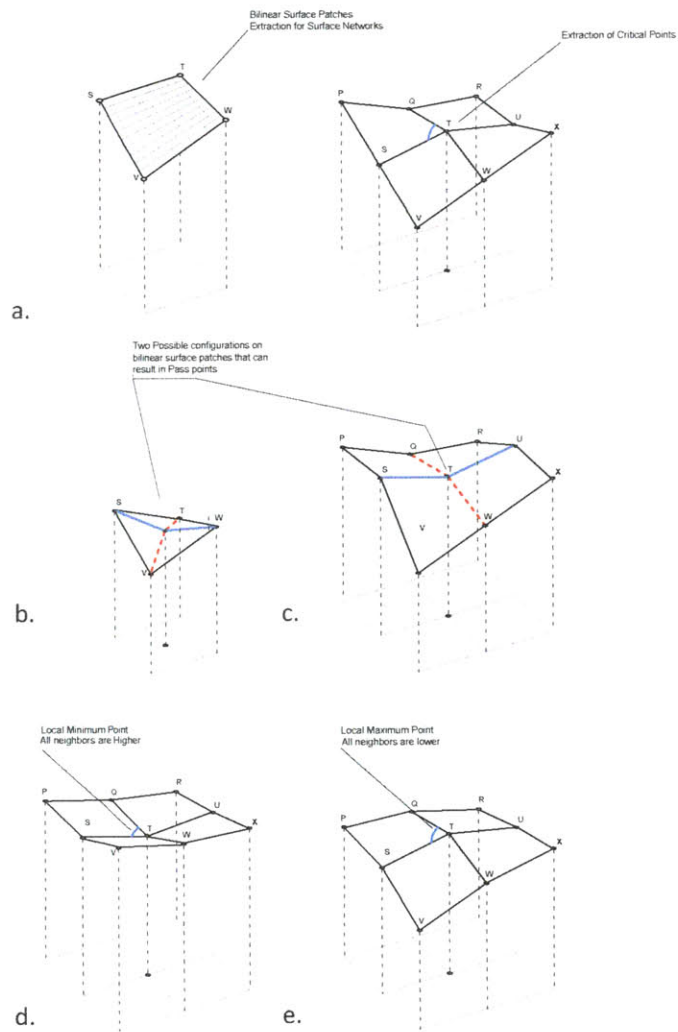


Figure 3.10. a. Surface patch and central vertex and its surrounding. b. Saddle point on the surface. c. Saddle point on the grid. d. Minimum point e. Maximum Point.

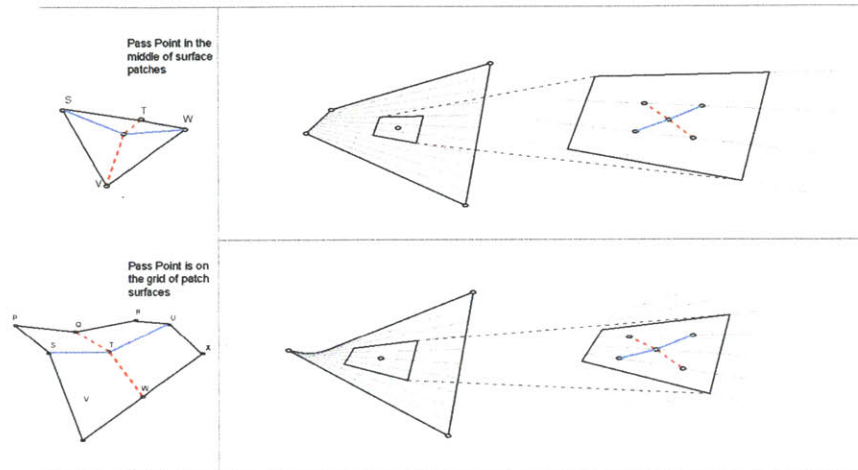
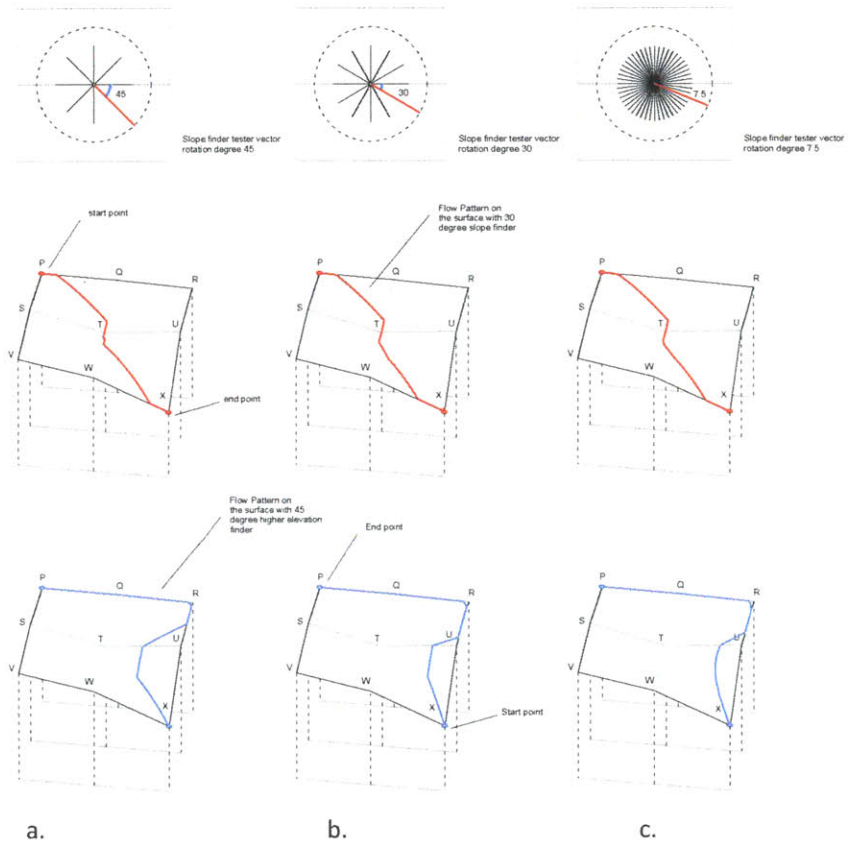


Figure. 3.11. Locating the saddle points on the topography and finding the principal directions to go up or down

The process of the surface network extraction starts right after the process of locating the saddle points. Since the graph is achievable connecting the saddle points to maximum and minimum (Fig. 3.11). for this reason we need to find the steepest path from the saddle points to take us to the maximum and minimum points of the surface. There are two conditions based on the location of the saddle point. First, is the saddle point on the surface patch. In this condition, the starting points for up and down movement is chosen based on their heights. The second condition is when the saddle point is on the grid. In this case, the starting point will be based on the surrounding neighbors.

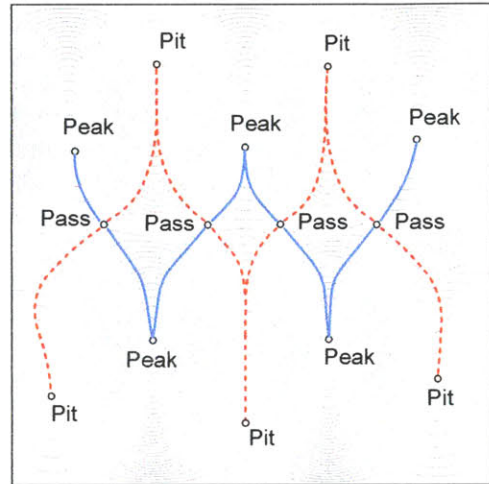
Figure 3.12. Different resolution of path finding. a. 45 Degree Resolution, top path downward, bottom, path upward. b. 30 degree resolution, top path downward, bottom, path upward. c. 7.5 degree resolution, top, path downward, bottom, path upward



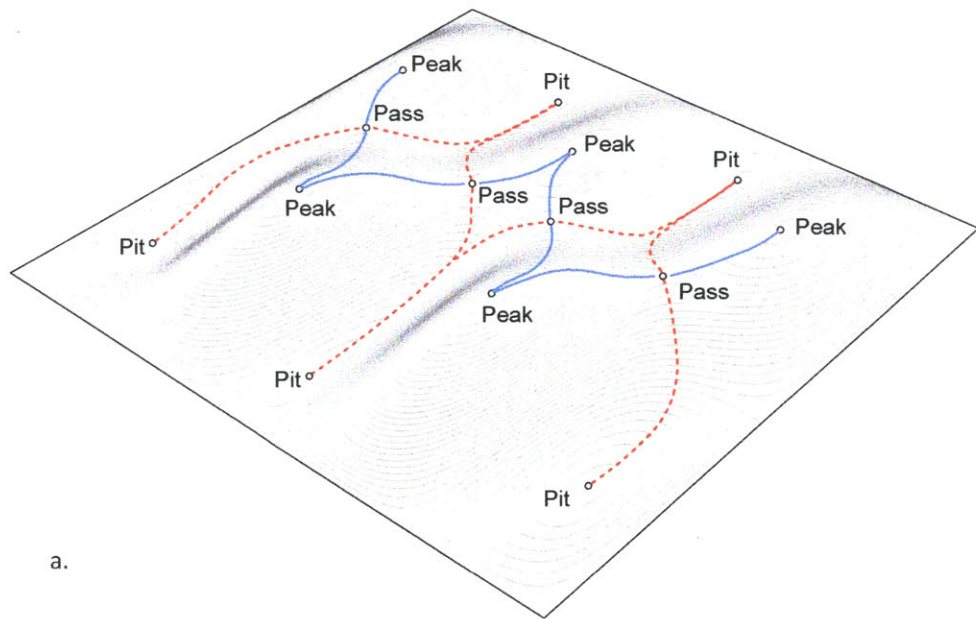
There are also different resolutions to find the steepest paths along the surface. The 45 degree resolution only finds the steepest paths in 8 primary directions, while 30 degrees and 7.5 degree and so on provides finer resolution of paths for the surface network graph (Fig. 3.12)

Continuing the path finder algorithm results in clear surface network patterns. Having this graph helps us understand the behavior of the surface with respect to the shape of the graph and the overall geometry of the topography (Fig. 3.13). This graph would be the basis for this research to reconstruct complex geometry using plan drawings.

Figure 3.13. a. axonometric view of the surface with its surface network graph. b. plan view of the surface with surface network graph.



b.



a.

3.3. Regenerating Surface Using Surface Network Graphs

So far we learned how to extract the surface network graph from the geometry of a given surface. From this step it is important to generalize the topologies of surface network graphs and control them to reconstruct the surface geometry. In (Fig. 3.14) the general topology of surface networks are represented. [Surface network graph]. in the step of the process the most simple network is chosen to extract the contours for the reverse process of surface generation.

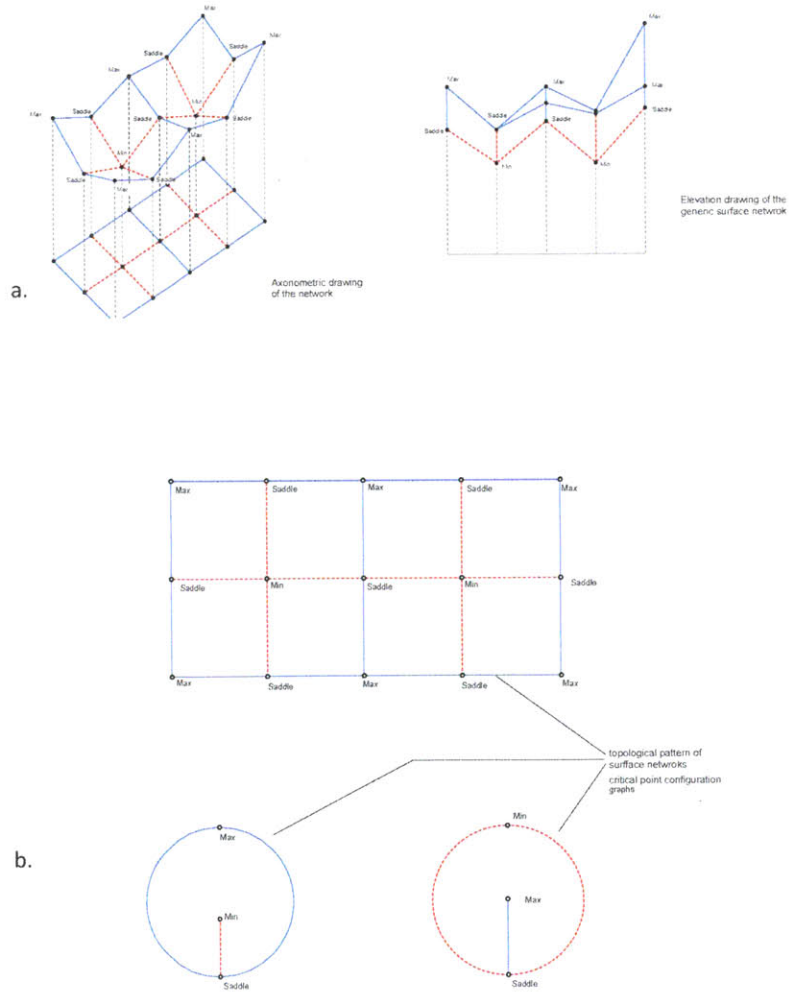
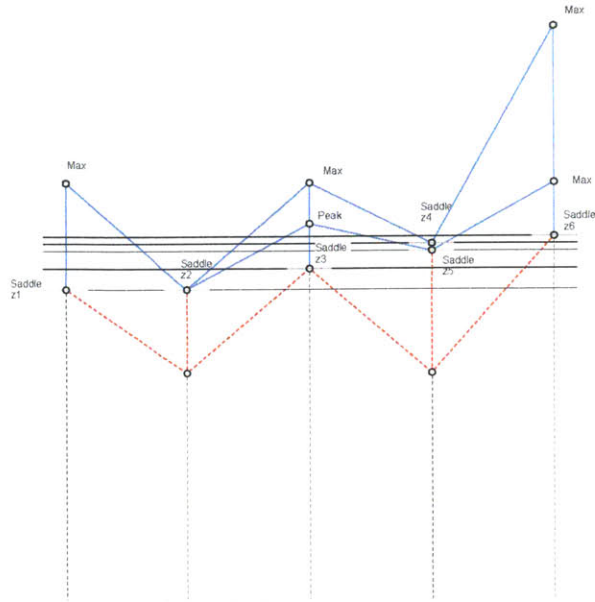
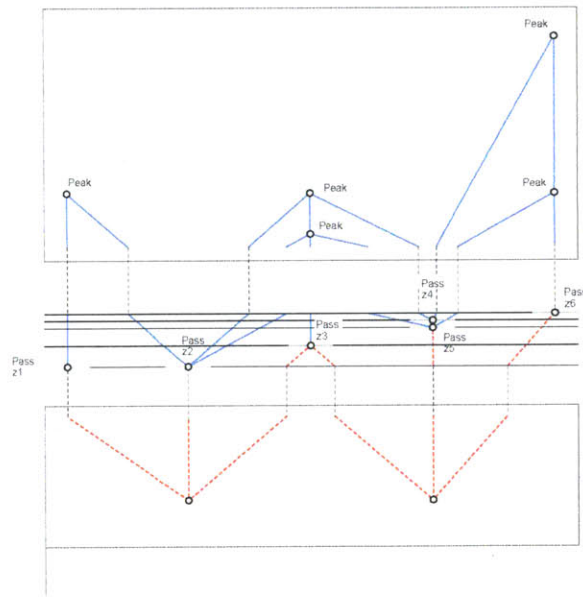


Figure 3.14. a. axonometric and elevation view of general surface network diagram. b. Plan view of different types of surface network after [30a 2002].



In order to find the contours of the graph we need to divide the graph based on the heights of the pass points

Figure.3.15. Subdivision of the graph into three main parts: upper, middle and lower parts.



As we can see the parts of the graph which are above and below the maximum and minimum heights of the pass points contain the local maxima and local minima of the graph and consequently the contours are closed curves around those points.

Based on this notion the author developed an algorithm to extract the contours from the network and reconstruct the surface. In order to extract the contours, we need to divide the whole graph into mainly three parts (Fig. 3.15) upper part, which includes all the peak points. Contour lines in this part of the graph is closed curves. This can be proven using the [Cayley 1859] grammar of a surface. According to him indicatrix around local maximum or minimum points of a surface is a closed curve and circular. The Lower part can be derived respectively. The most important point in this subdivision is the location of the middle part or the lower bound of upper part and the upper bound of lower part.

For the upper part of the lower part, if we search among the saddle points and find the one which has the lowest height among the others, then that can be used as the basis for the boundary between the lower part and the middle part. For the lower bound of the upper part, if we find a saddle point which has the highest height among the others, that would be the upper bound for the middle part (Fig. 3.16)

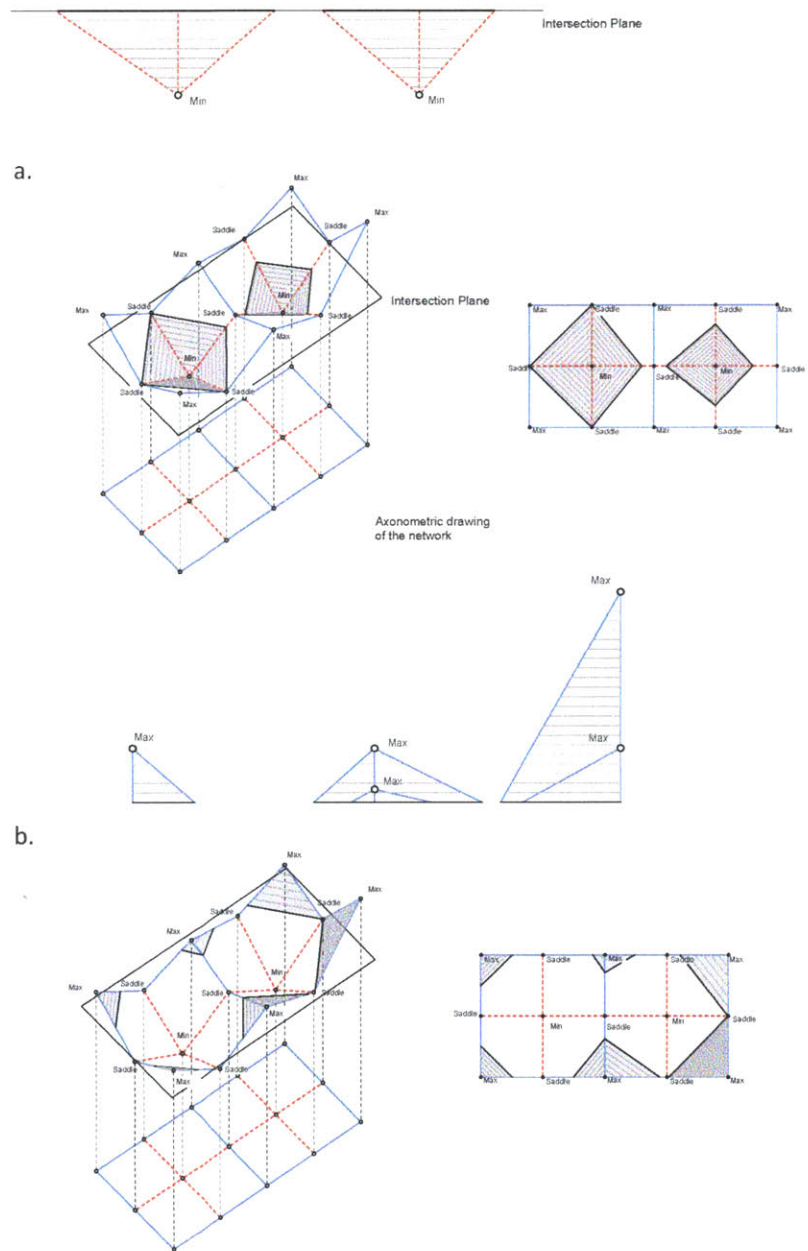


Figure 3.16. a. The lower part of the graph and lower bound of the middle part b. upper part of the graph with the upper bound of the middle part.

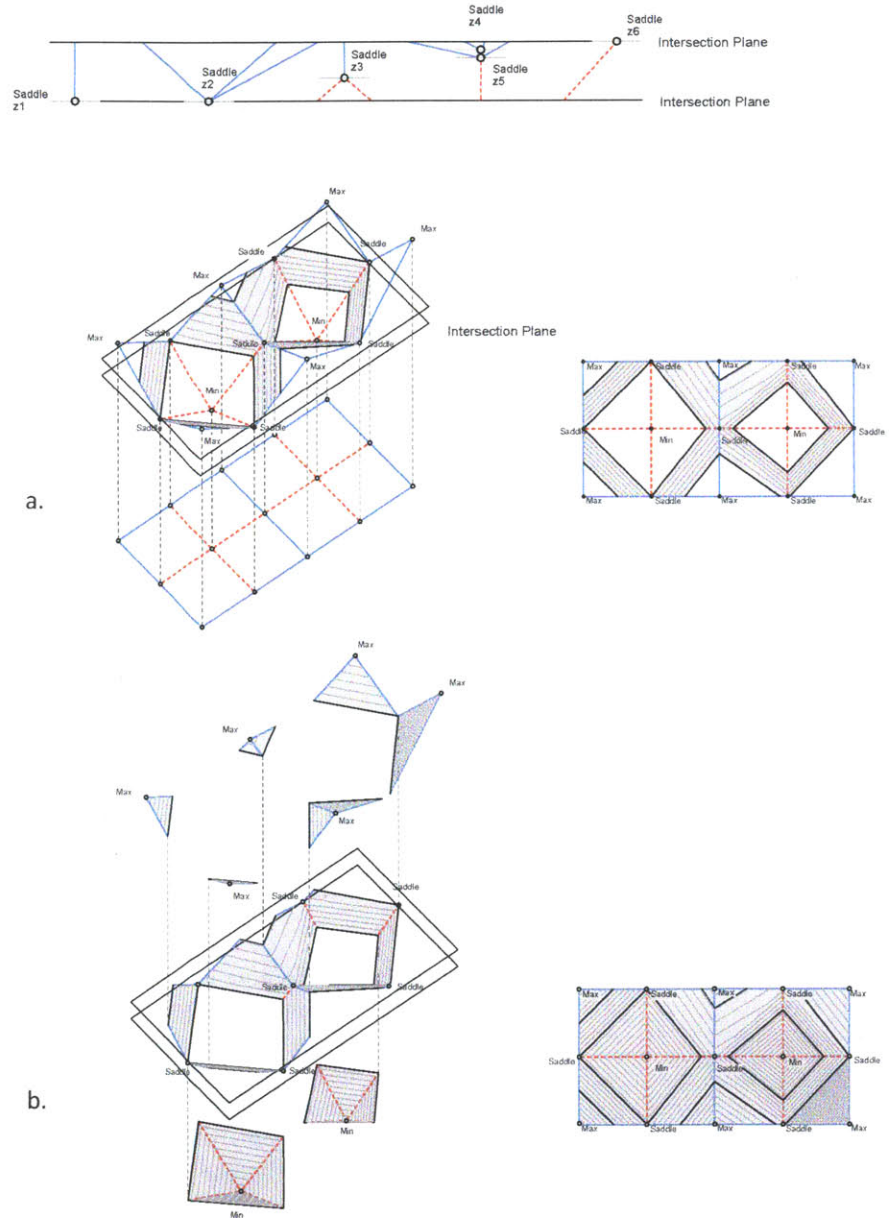


Figure. 3.17. a. Middle part of the graph
b. Completing the contours by adding all parts together

By Extracting the contours from each part we will be able to have continuous closed curves to reconstruction of the surface. The importance of this algorithm relies on the fact that since these contours are extracted from a particular curves in the graph, there is no order in terms of connecting the curves and making a continuous closed contour curve which travels along the whole surface (Fig. 3.17)

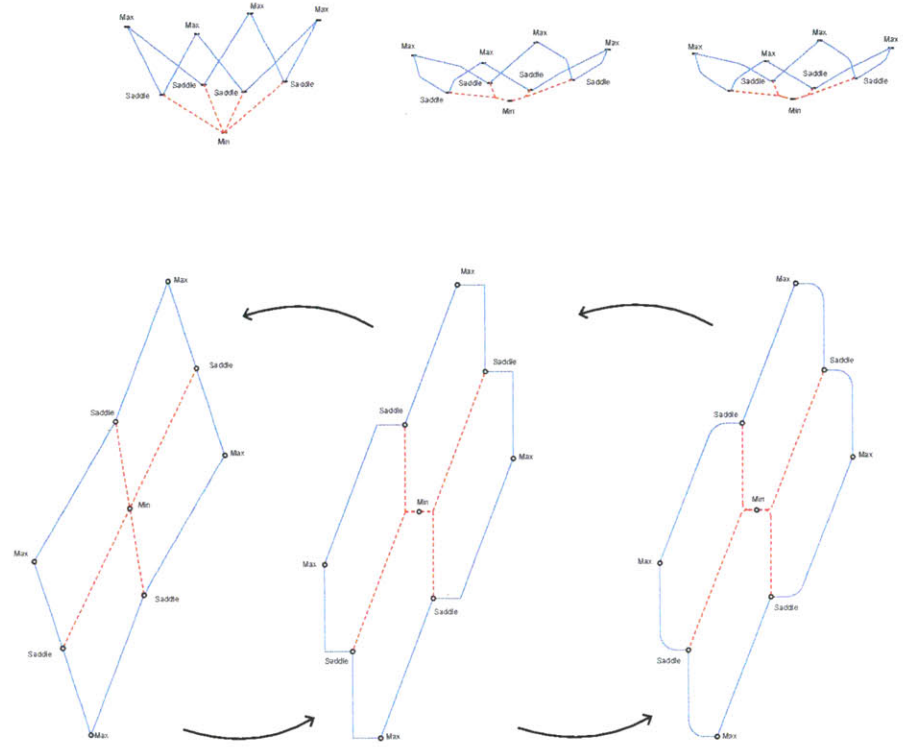


Figure. 3.18. Topologically Similar networks

There is a possibility of translating any geometrical pattern and grid into landscape organization using surface network graph

Developing the contour extraction algorithm allows us to extract the contours from any given network graph and consequently reconstruct the surface using the contours. This means any given graph that is topologically related to the idea of surface network can be used to generate topography. (Fig. 3.18). Shows some design possibilities of a simple network and interchangeability of them. This is an important point for designs, since one graph can provide multiple design options.

Lets take a look at a simple design possibility of network graph. In This example, designer starts with a simple module of surface network graph and aggregates them to create a topography. Simply this aggregation can be used for contour extraction algorithm and eventually the surface topography will be constructed based on that. (Figure. 3.19)

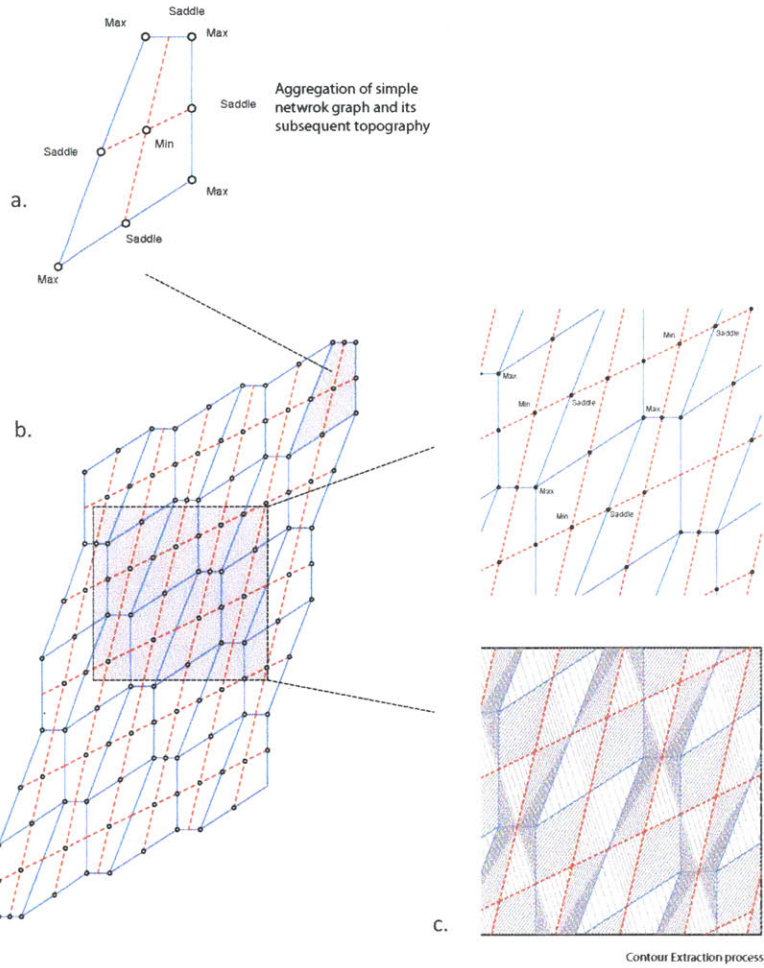


Figure 3.19. a. Simple module of Surface network Graph. b. Aggregation of the module c. Contour extraction result d. Final topography

The key point in generating the surface with this technique is the variability of the result based on the contour manipulation. Secondary design algorithms can be used in this step to change the simple definition of contour in different types of curves. (Figure. 3.20)

In current example the regular sharp connection of contours are changed step by step using chamfering algorithm. This algorithm searches for intersection between the contours and chamfers them based on a fraction of the length of each intersection lines. The result is a topography with different resolution on courses and ridges.

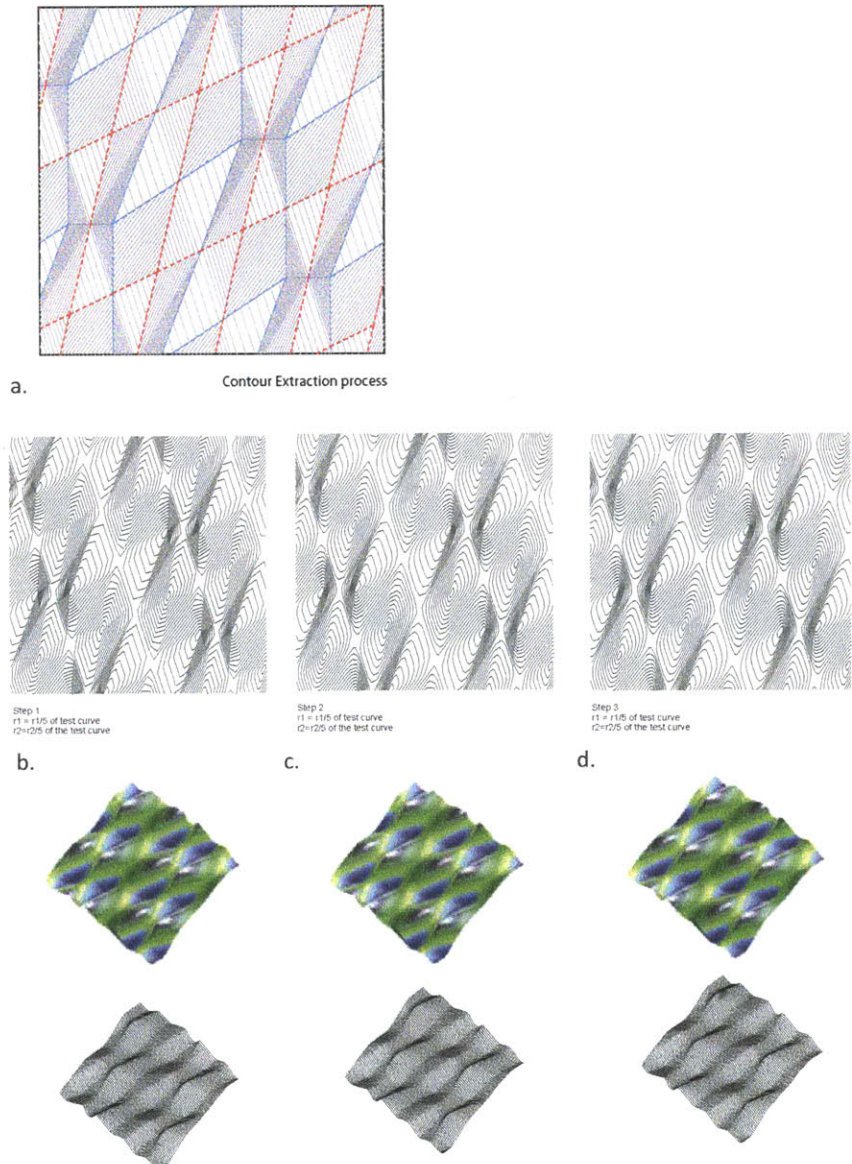


Figure. 3.20. a. Extracted contours from the network graph with sharp angle connections b. Chamfer algorithm to change the corners step 1, c. Chamfered Algorithm Step 2, d. Chamfered algorithm Step 3.

3.4. Summary

In this chapter the general idea of surface construction using graphs was introduced. The material for this chapter was gathered from different science fields such as computation a geometry, geo-computation and geographical information science. Consequently, the history of surface data structure was visited. The term critical graph and it use in geo-computation was the main area of containers of this chapter. Further on the process of topography generation from critical graph was explained and some example of the use of such techniques in terms of design was provided.

Chapter Four: Generating Performative Surfaces

4.1. introduction

In this chapter the author will introduce three main algorithms based on the foundation of the surface network graphs. The surface network graphs are not directly used in the generation of these graphs but lessons learned from the performance of the surface was necessary for the results of this chapter. The Algorithms provided in this chapter are chronologically related meaning that they complement each other and the earlier algorithms are the ancestors of the last algorithm.

4.2. Performative Aggregation of Critical Graph

In this section, let's take a look at the properties of a single critical graph. mentioned in chapter 3, each complete graph consists of three types of points: Maximum, Minimum, and saddle points. and two types of lines: Ridge lines are the ones which connect saddle points to minimum points. Coarse lines, on the contrary, connect saddle points to maximum points. change in the height properties of each of the graph's points will change the whole properties of the graph as we can no longer call that a complete network (Fig. 4.1)

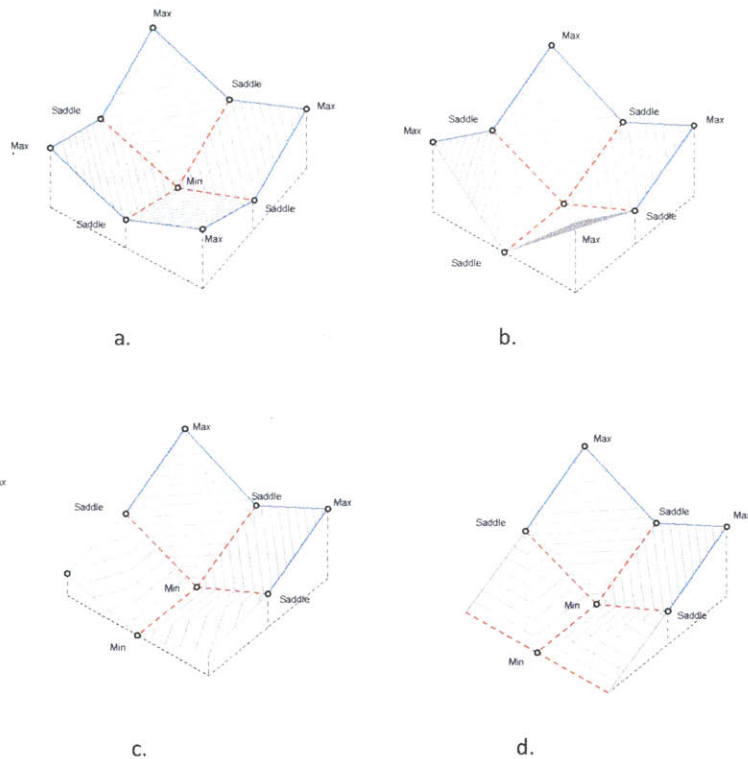
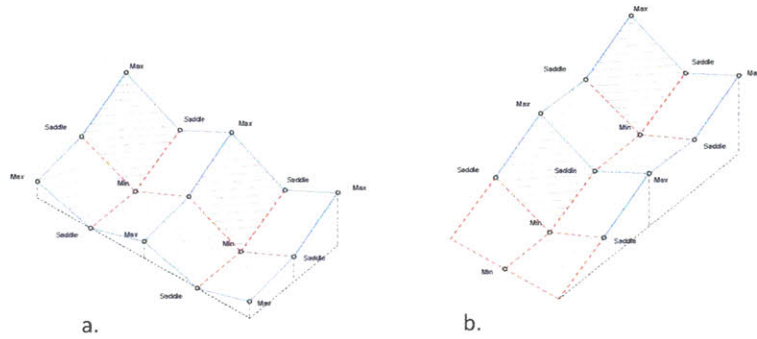


Figure. 4.1. Change in height of each point of the graph changes the properties of the graph. a. complete graph b. Graph with incomplete ridge coarse lines c. graph with incomplete ridge and coarse lines d. graph with new ridge lines.

aggregation of the surface network graph is a method of reconstruction of surface. observation from the characteristics of the network shows the possibility of reconstruction using only ridge lines (Fig. 4.2). if we reverse the order of surface reconstruction, The key point is the change in height of the points. if we start from the zero elevation height , in order to construct surface, we need to create ridge lines. This observation is the basis for deriving surface reconstruction algorithms which are provided in this research.

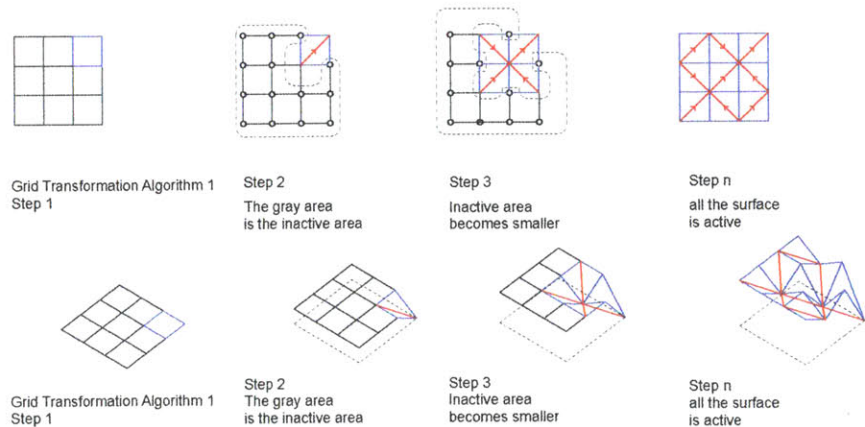
Figure. 4.2. a. Side by Side Aggregation of the Surface network units. b. descending aggregation of the surface networks.

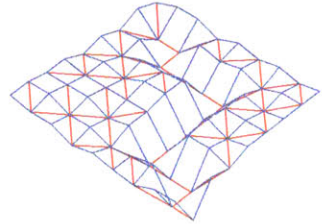
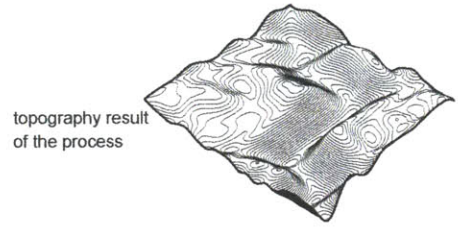


4.3. Surface Generative algorithm version 1

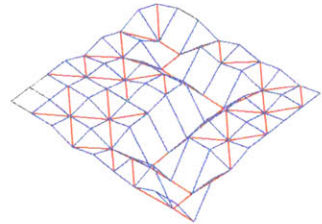
This algorithm is based on point grid transformation in three dimensional space. The assumptions in this approach is that the surface will be constructed on a starting point grid field. Surface reconstruction is achieved through activating each cell on th point grid to create ridge line. the Algorithm is operated in each step and continues till the last cell of the point grid is activated. In order to create the first ridge line, all the points of the point grid are transported to higher elevation except the starting point (Fig. 4.3). in the next step all the transported points, except the ones that are connected to the both sides of the ridge, are transported again to higher elevation. this technique generates connected ridge lines as well as coarse lines on the surface. the process continues, untill all the surface truns into a connected networks of ridges and coarse lines (Fig. 4.4)

Figure. 4.3. Step by Step Transformation Algorithm

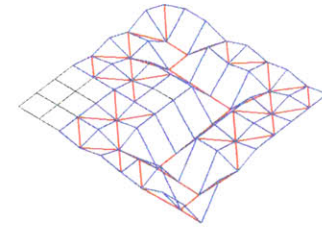




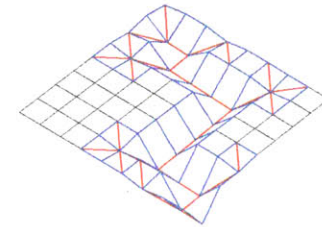
Step n



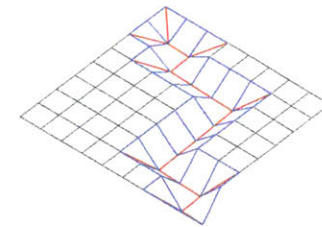
Step 4



Step 3



Step 2



Step 1

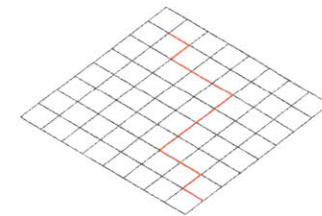


Figure. 4.4. Surface transformation Algorithm

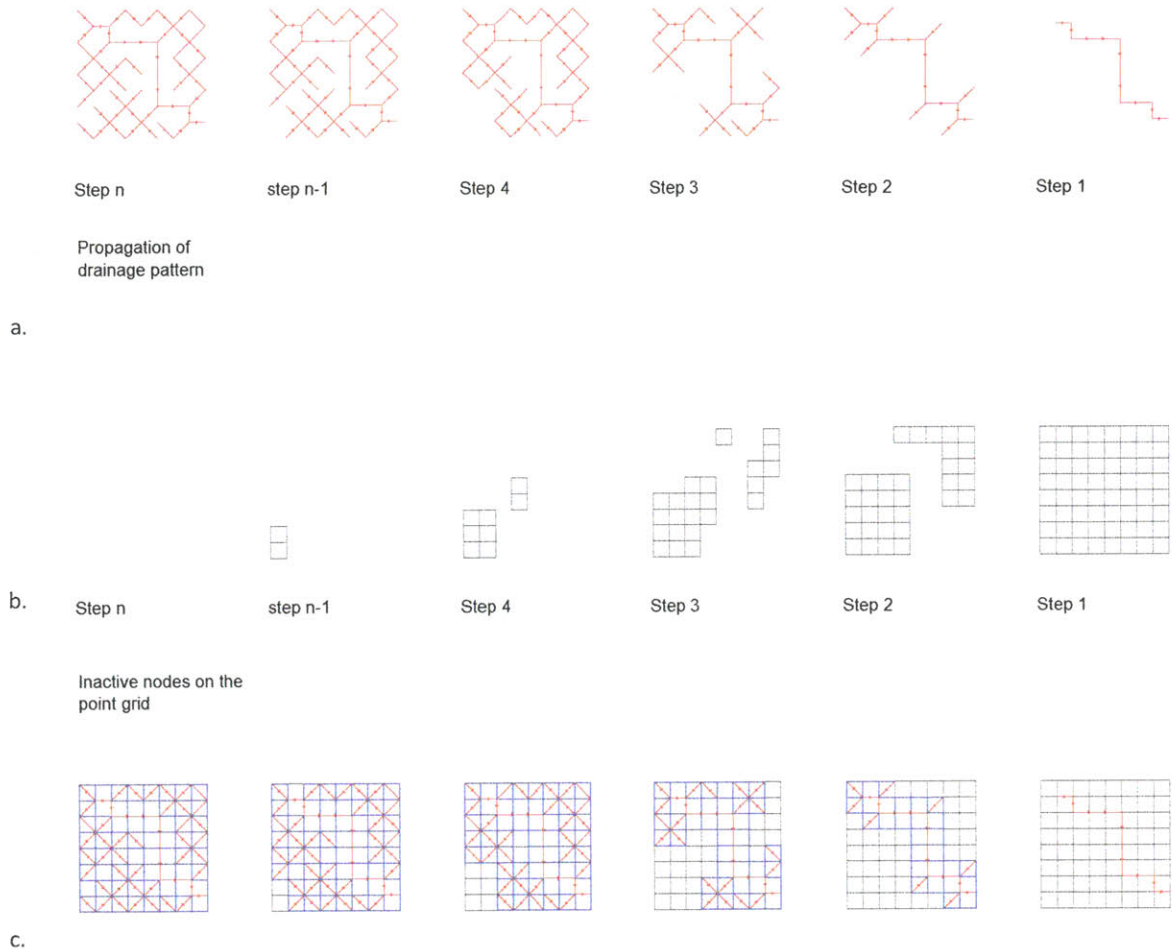


Figure. 4.5. Different results of algorithm through the completion process. a. Propagation process b. Cell activation process c. Surface transformation process

The process of transformation has different steps. In Fig. 4.5, The propagation of ridge lines are shown as well as the activation steps for each cell till the last cell. The most important factor that need to be mentioned is the whole process is starts with drawing a line on the flat point grid. This will provide a control for the designer to transform the surface based on the original flow pattern that s/he input to the surface. Figure 4.6 Shows the physical model of the process.

4.4. Surface Generative algorithm version 2

This algorithm is in connuatio of the previous algorithm with some modification. The previous algorithm results in more dynamic surface generation which might not be ideal for architectural porposes. in this respect, the Author modified the process of algorithm to acheive smoother surface. Accordigly, in this algorithm the transformation of the grid into higher elevation is modified through just moving the points that are not shares with the activated cell.

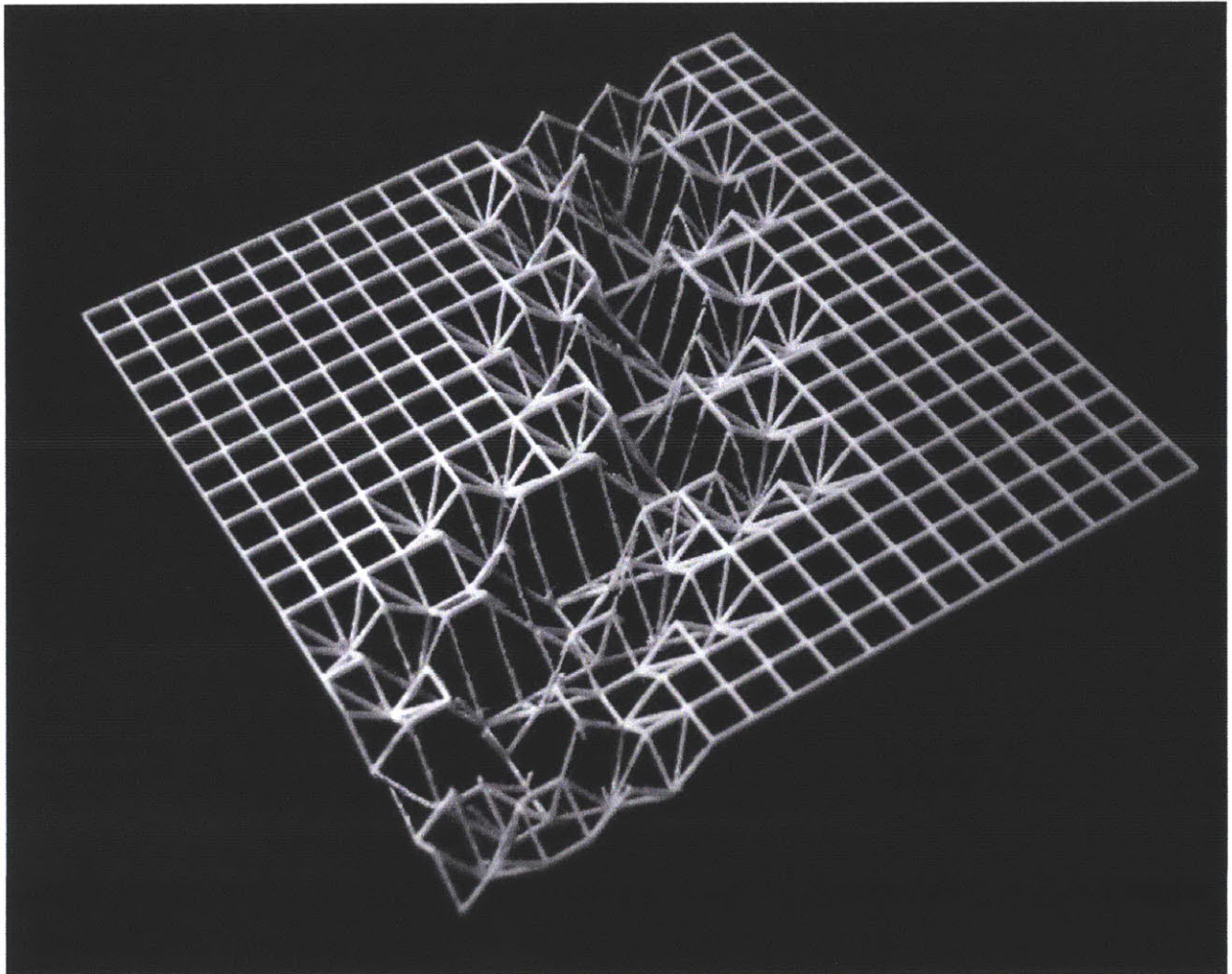
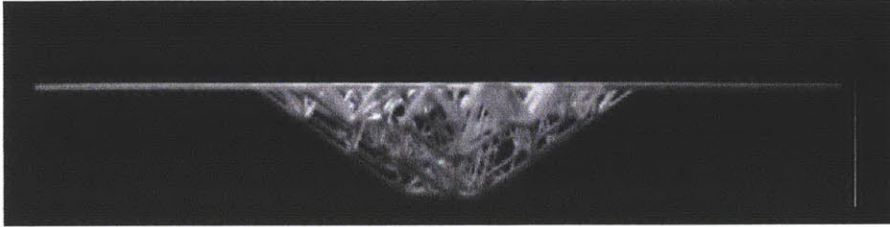
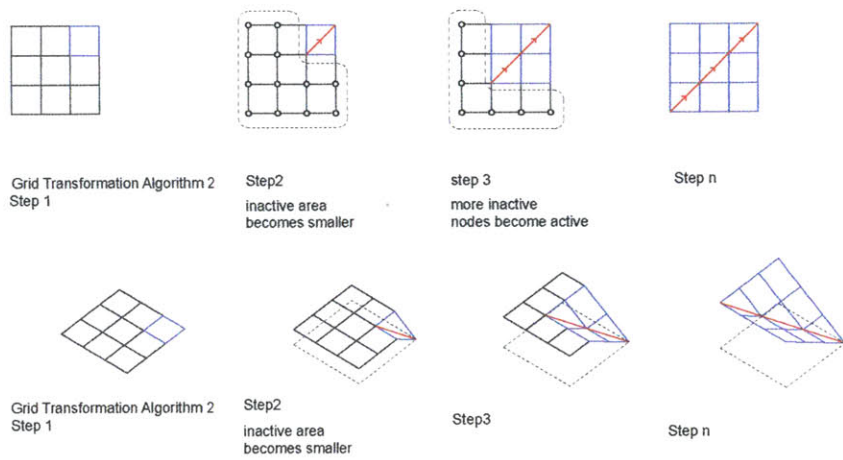


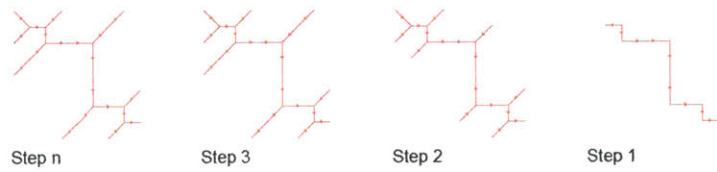
Figure. 4.6. Physical Model of the transformation process

This process helps creating smoother and cleaner ridges and eventually cleaner topography. in this algorithm, similar to previous one, the designer inputs the flow pattern drawing and surface transformation is based on ridge generation process (Fig. 4.7, 4.8, 4.9)

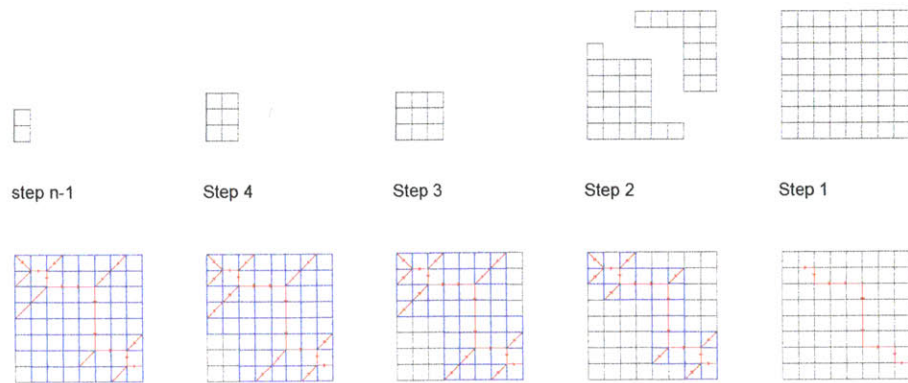
Fig. 4.7, 4.8. Step by Step process of activation of cells.



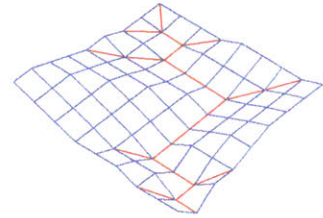
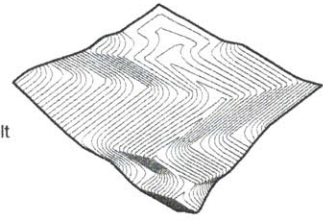
Propagation of drainage pattern



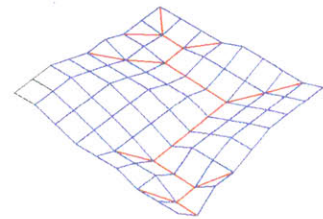
Inactive nodes on the point grid



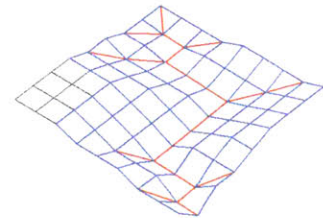
topography result
of the process



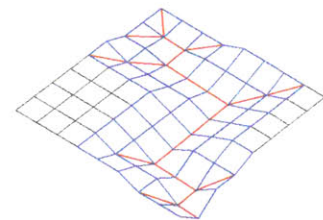
Step n



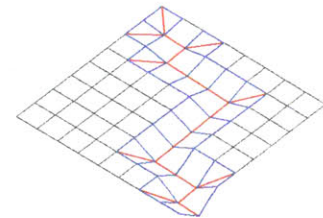
Step 4



Step 3



Step 2



Step 1

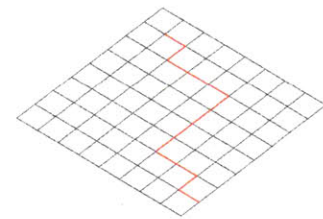


Figure. 4.9. Surface transformation Algorithm

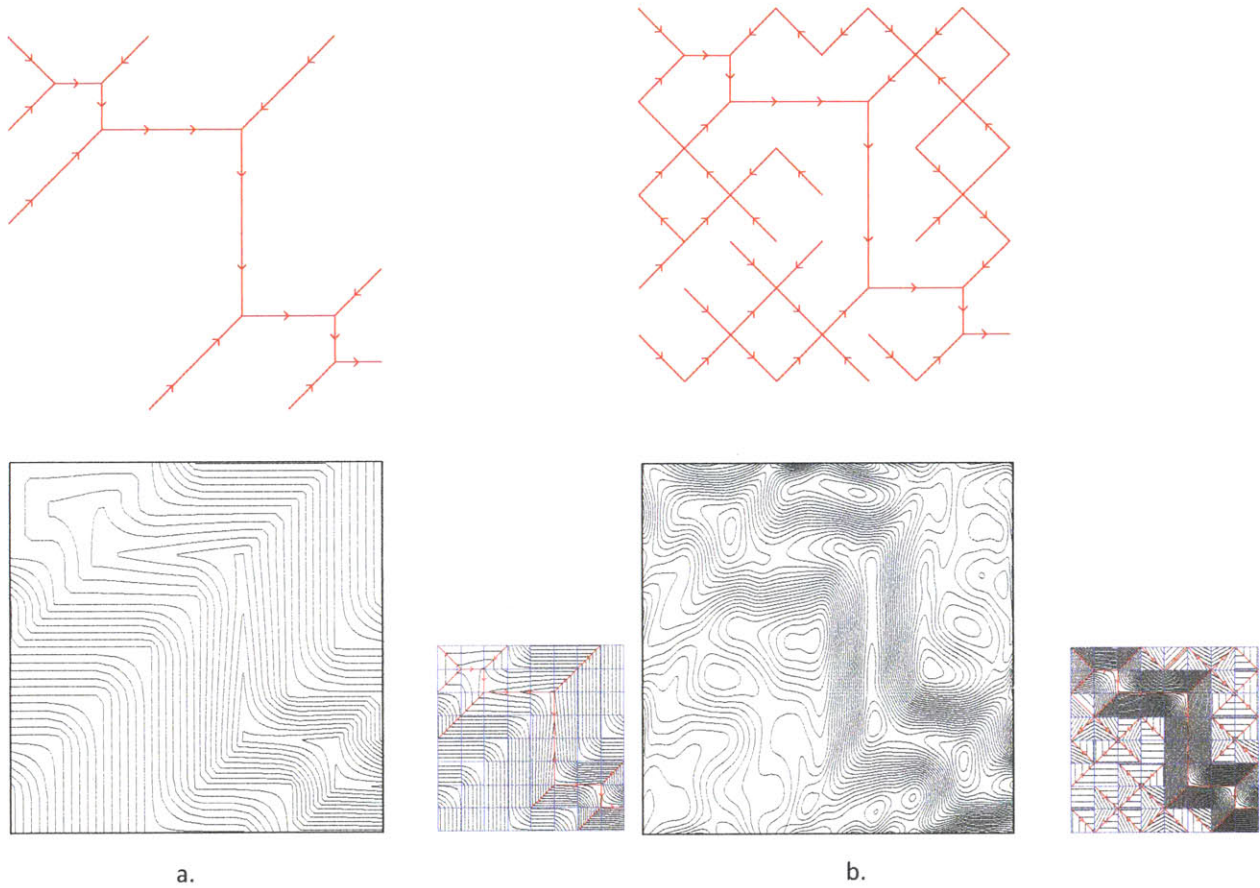


Figure 4.10. a. Algorithm 2 b. Algorithm 1

Comparing the topography results from the two algorithms reveals their inherent differences (Fig. 4.10) obviously, the second algorithm generates smoother and cleaner surface, but this shouldn't overshadow the interesting features of the first algorithm. The first algorithm, regardless of its complex geometry can transfer flow in multiple directions and create more dynamic flow of water. The second algorithm, on the other hand, provides with a faster and shorter paths for the flow. The most important disadvantage of both is the low resolution of the input drawing and also the result. Moreover, the drawing cannot handle angles more or less than 90 degrees and this is a limitation for design and generating performative surfaces.

4.5. Surface Generative algorithm version 3

This the third and last algorithm provide in the research which is developed to overcome the problems and disadvantages of the previous algorithms. flow of water always follows the shortest paths on the surface or topography. this algorithm is founded based on the idea of shortest distance of point grids to the input flow drawings. similar to previous algorithms the surface is a result of transformation of point grid in three dimensional space. (Fig. 4.11). in this the shortest distance of each point with respect to the flow patterns is calculated (Fig. 4.12). unitizing the shortest direction for each point on the grid will result in a discontinuous field of lines and points. in order to create a continuous field of lines and points Author developed a rationalization algorithm for the flow direction. this simple algorithm creates a connected network of lines and points which also represent the direction of the flow at each point of the point grid (Fig. 4.13).

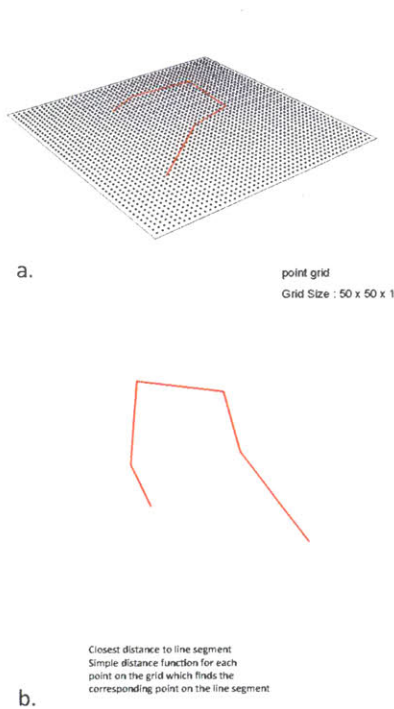
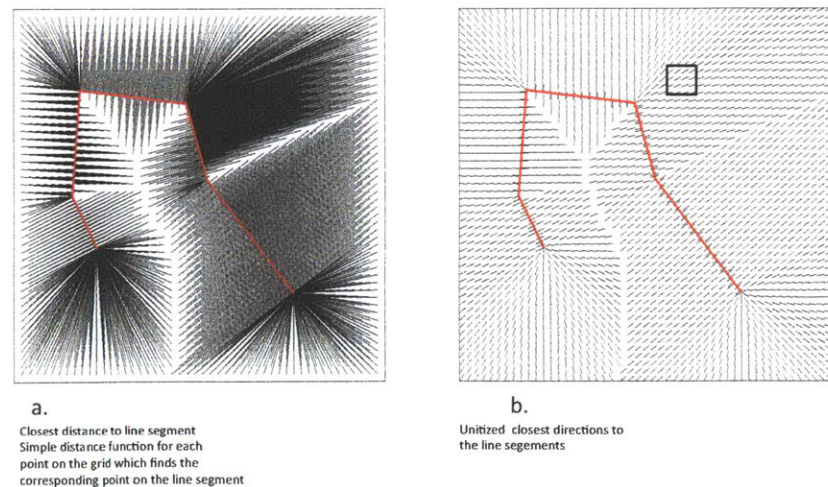


Figure. 4.11. a. Point grid b. input Flow pattern on the point grid

Figure. 4.12. a. Point grid b. input Flow pattern on the point grid

4.5.1. Rationalized drainage direction of a surface

In order to explain the process of this rationaliaion, lets have a closer look at the unitized field of ponts and flow directions [Figure. 44]. for each point there exist eight souraounding neighbors. the direction which connects each point to its souraounding points is considered a primary direction. this allows us to create a connected network of points and lines. in order to rationalize the direction of the flow, the angle of the unitized vector is complared to the primary direction for each point. the mimum of these angles is chosen and the corrsponding primary direction is drawn to create a connected network of flow. [Fig. 4.13).



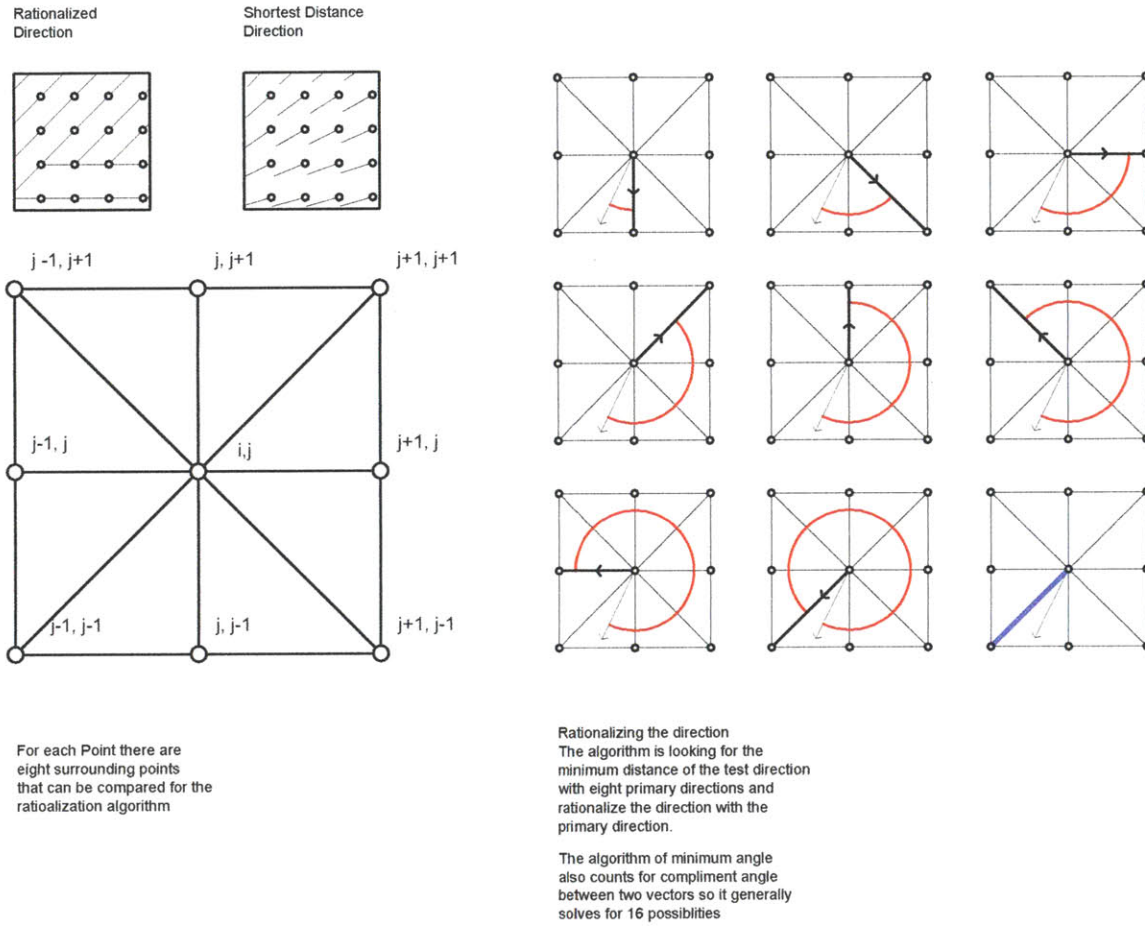


Figure. 4.13. each cell is compared to eight primary directions of the flow to rationalize the unitized direction vector

This technique is used to test the flow representation on the surface. For this reason a complex surface was chosen to reflect the direction of the flow. First a 2 dimensional point grid projected on the surface. From each projected points the steepest path was calculated based on the resolution mentioned in chapter 3 (Fig. 4.14, 4.15, 4.16). Different steps of flow will result in different paths on the surface.

If we use the same technique of rationalization for the flow direction, we can reconstruct the geometry of the surface with a connected network which represents the direction of the flow on the surface as well. In order to cover the whole area of the surface, this technique needs to be applied in two opposite directions: steepest path up, and steepest path down. This will cover the whole area of the surface [Figure. 46]

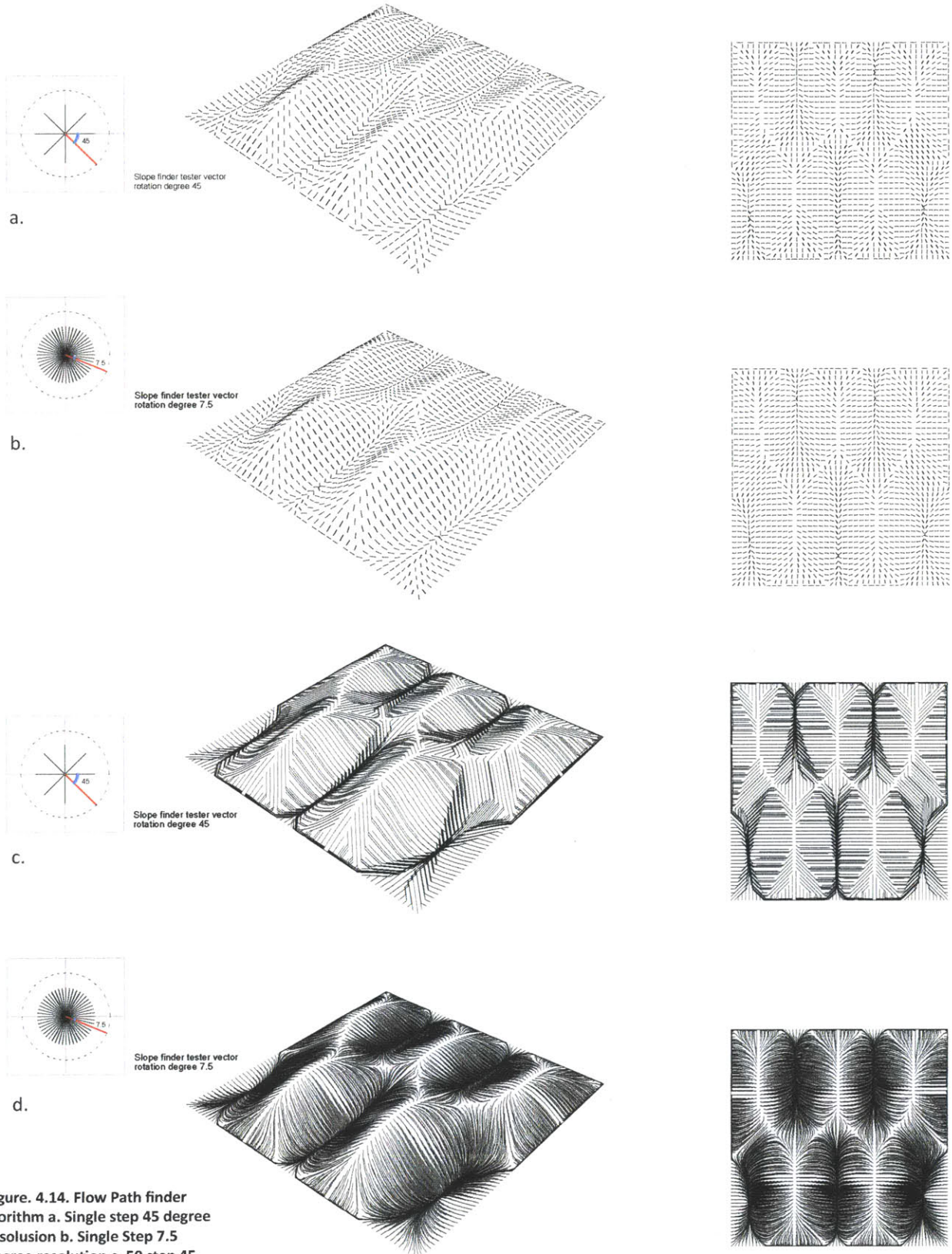


Figure. 4.14. Flow Path finder algorithm a. Single step 45 degree resolution b. Single Step 7.5 degree resolution c. 50 step 45 degree resolution d. 50 step 7.5 degree resolution

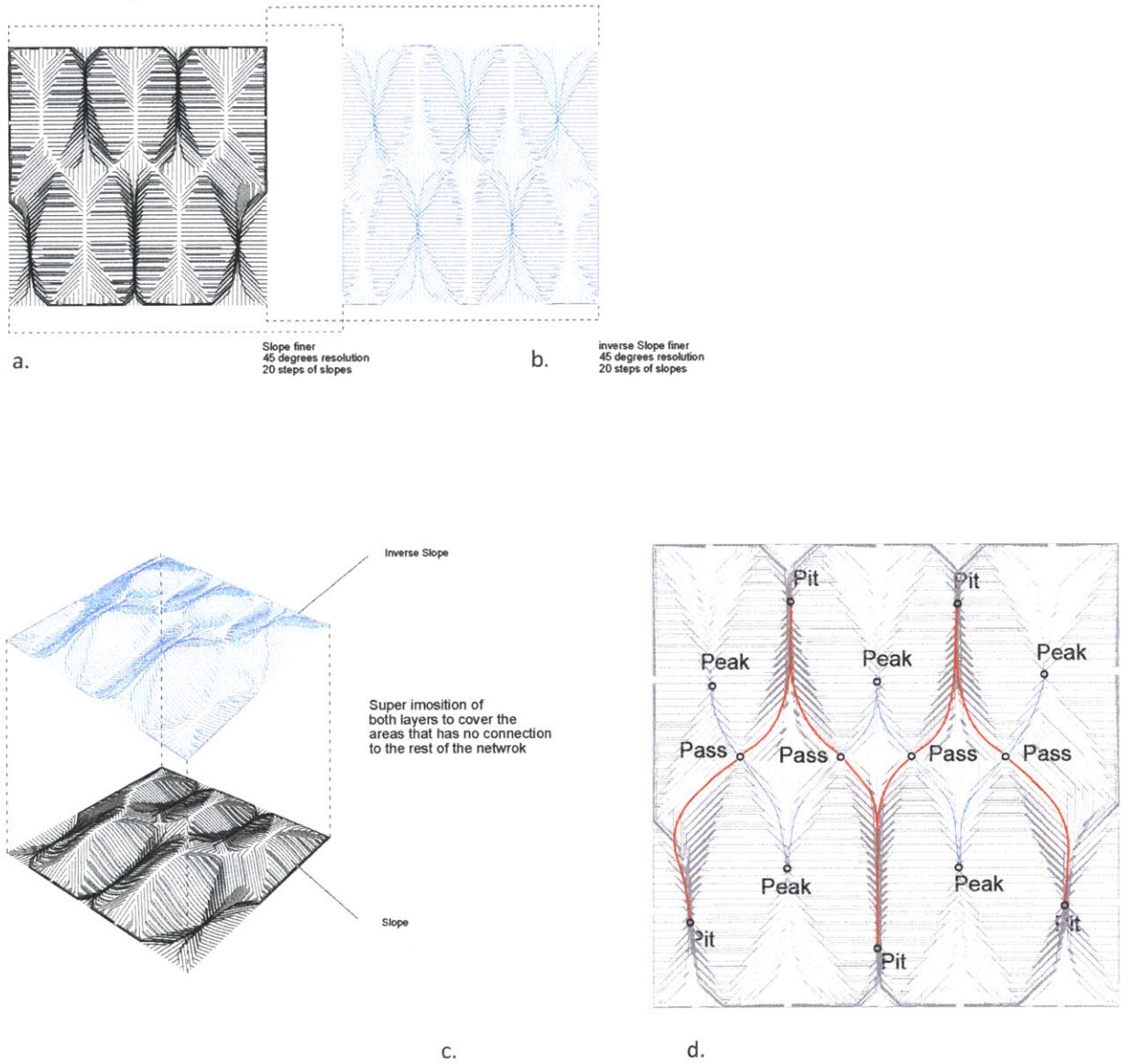
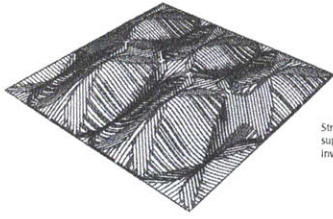


Figure 4.15. Slope Finder algorithm a. upward direction b. Downward direction c. superimposition of both d. surface network graph layered on top of the flow pattern



Structural model made based on
superimposition of watersheds and
inverse watersheds

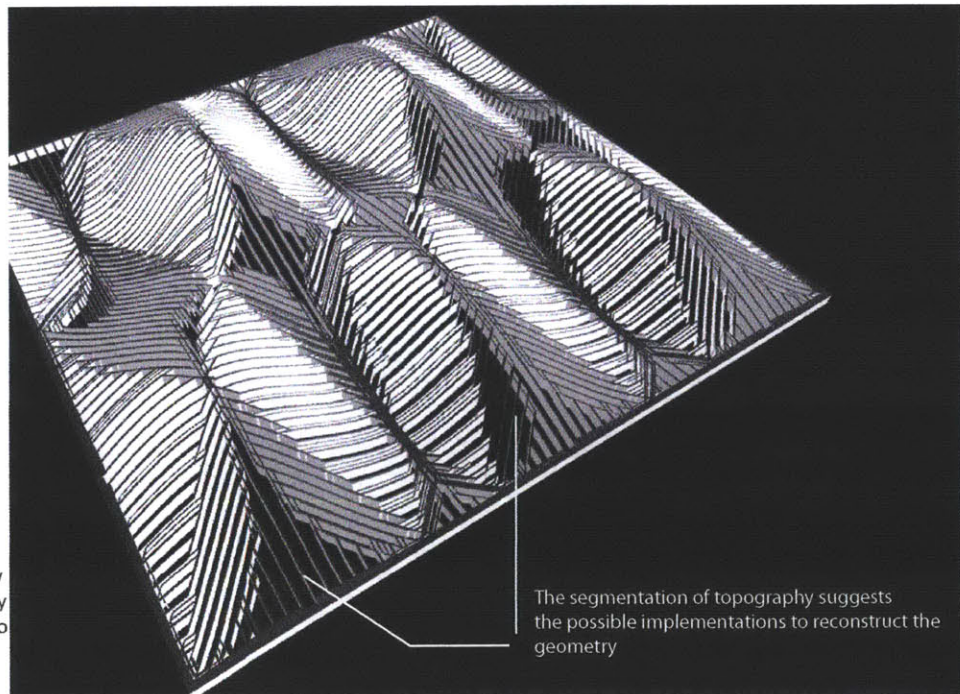
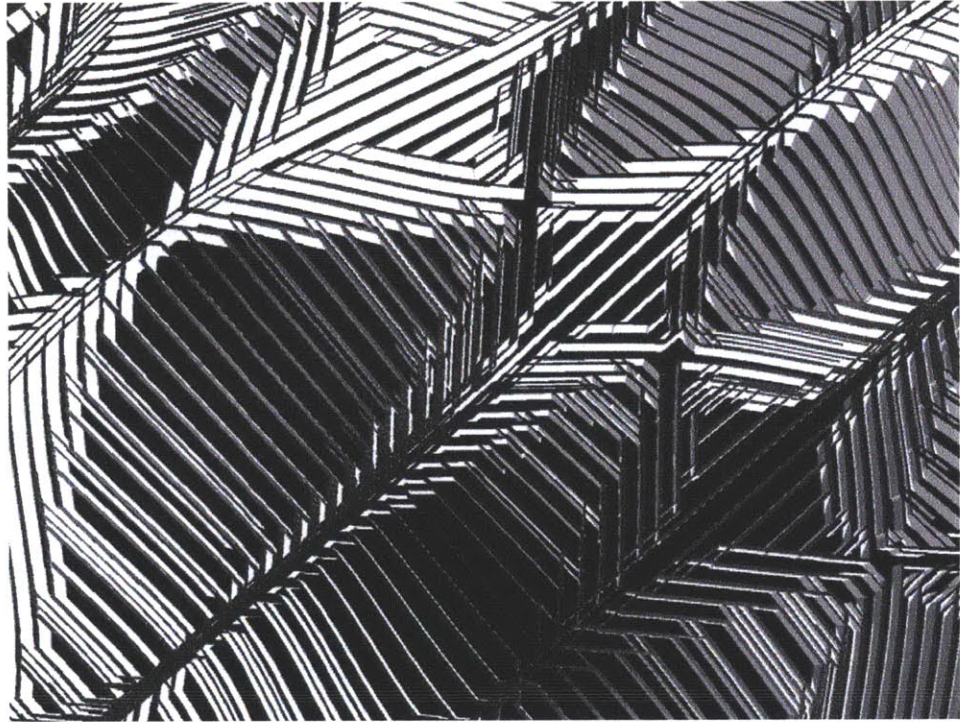


Figure. 4.16.spatializing the flow
patterns. note that the geometry
is not very clean since there is no
rationalization applied yet.

The segmentation of topography suggests
the possible implementations to reconstruct the
geometry

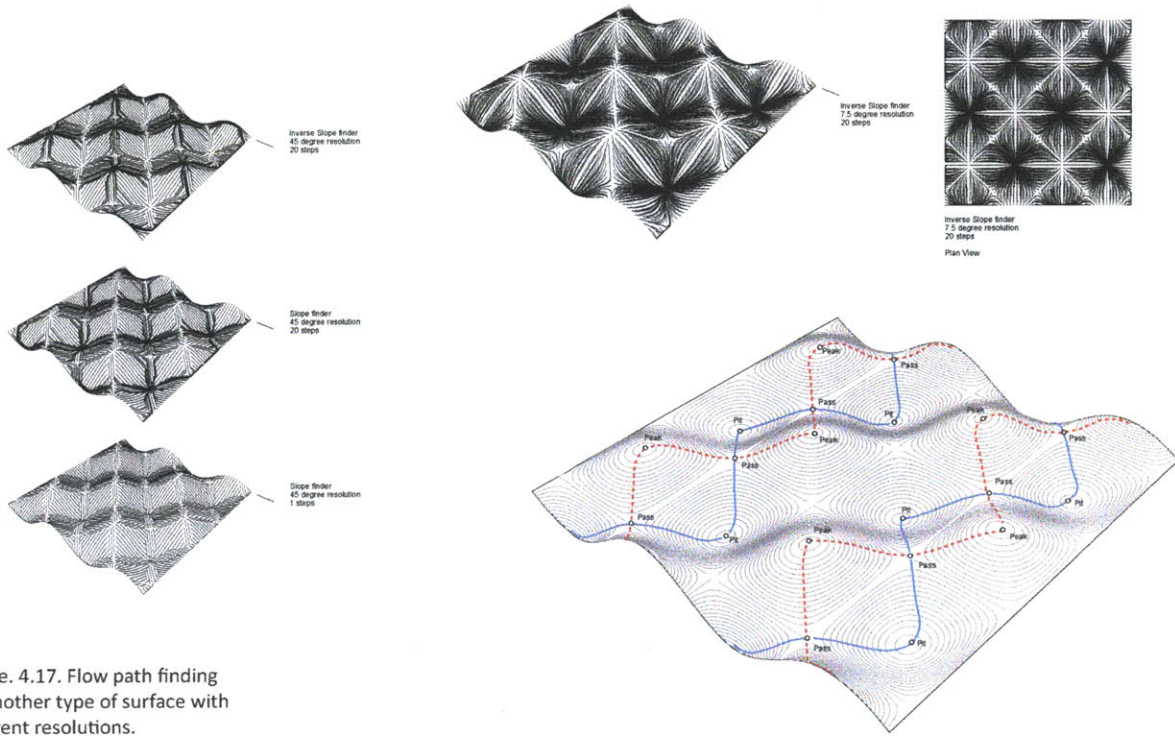
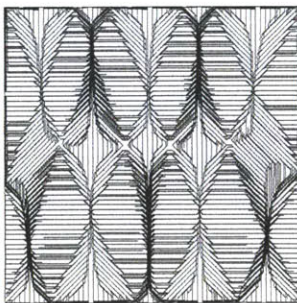


Figure. 4.17. Flow path finding on another type of surface with different resolutions.

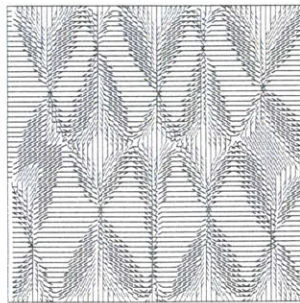
Figure. 4.18. a. Flow pattern finding algorithm 45 degree angle resolution 50 steps of slope finding b. 45 degree of resolution first step of slope finding c. Con-

The same technique was also used for different types of surface to see the possible differences. This technique also helps recognizing the surface network graph of the surface (Fig. 4.17). by using this technique it is quite easy to cover the whole area of the surface with straight elements which meet each other at 45 degree angle in plan projections (Fig. 4.18, 4.19, 4.20) .



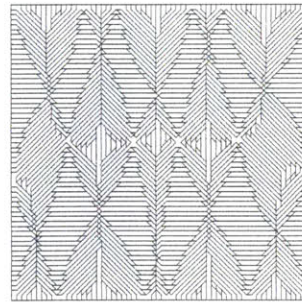
Original Water flow Algorithm result for a grid 60 X 60 on a given topography

a.



First step of rationalization of the channels

b.



Final Step of Rationalization based on 45 degree resolution

c.

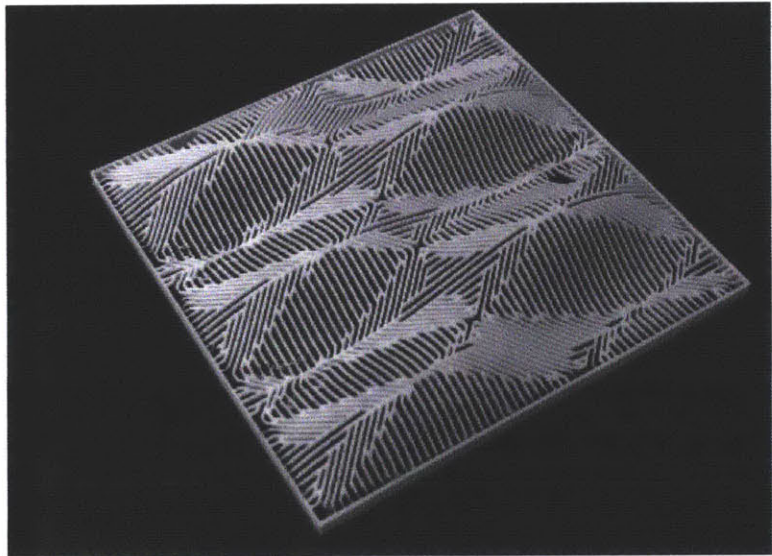
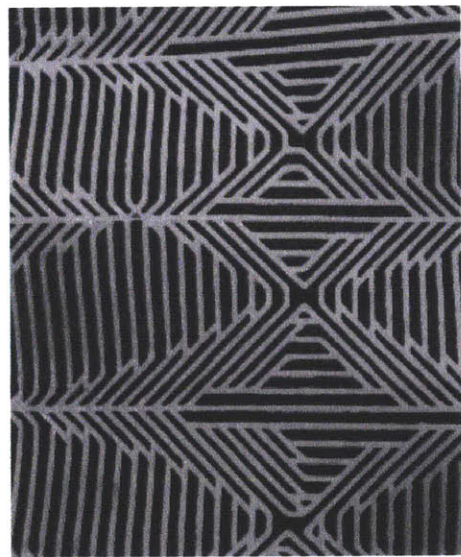
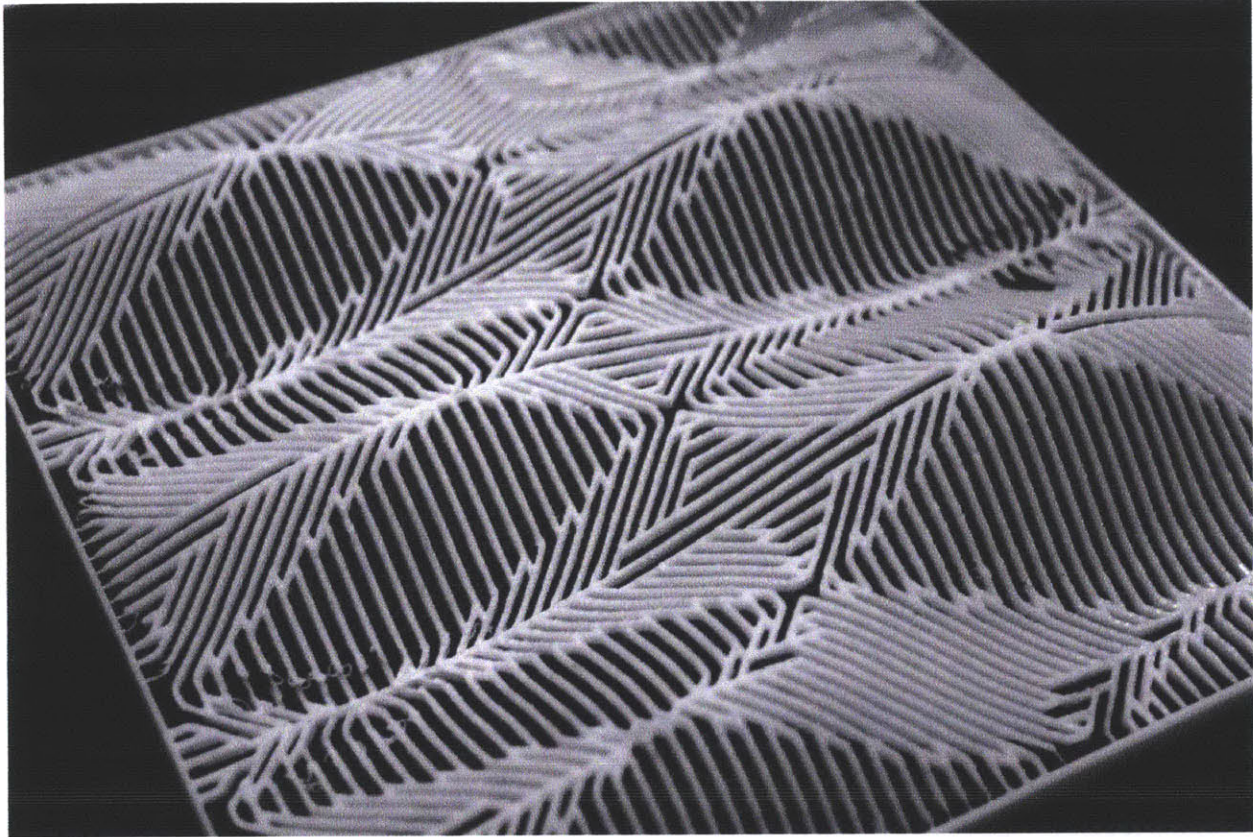


Figure. 4.19. Physical Model of
the rationalized surface based on
flow direction.

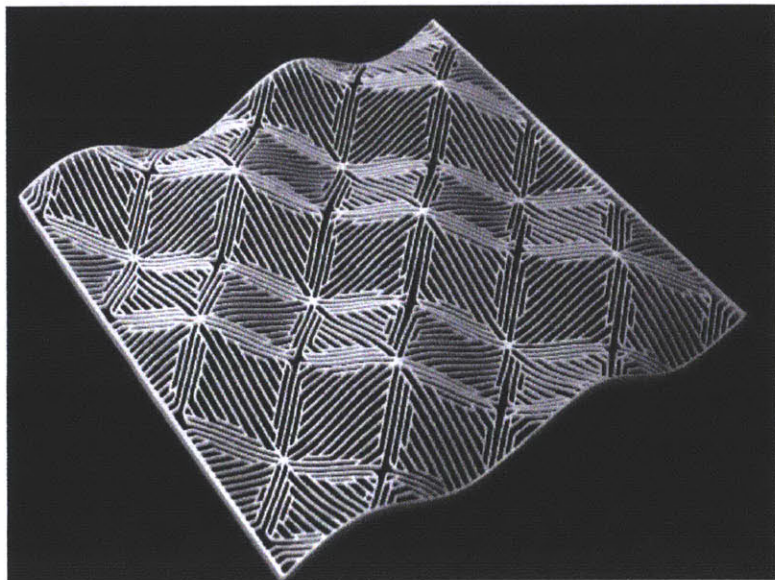
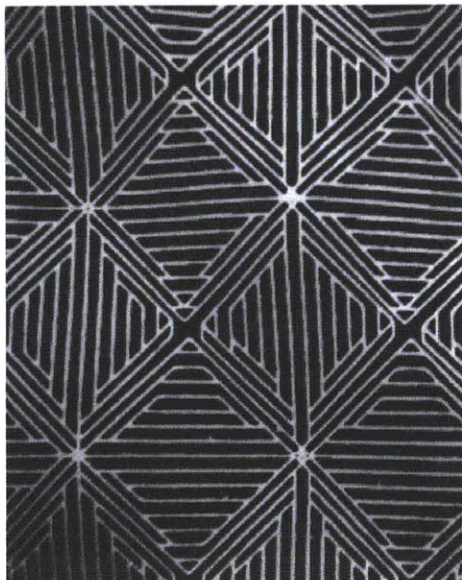
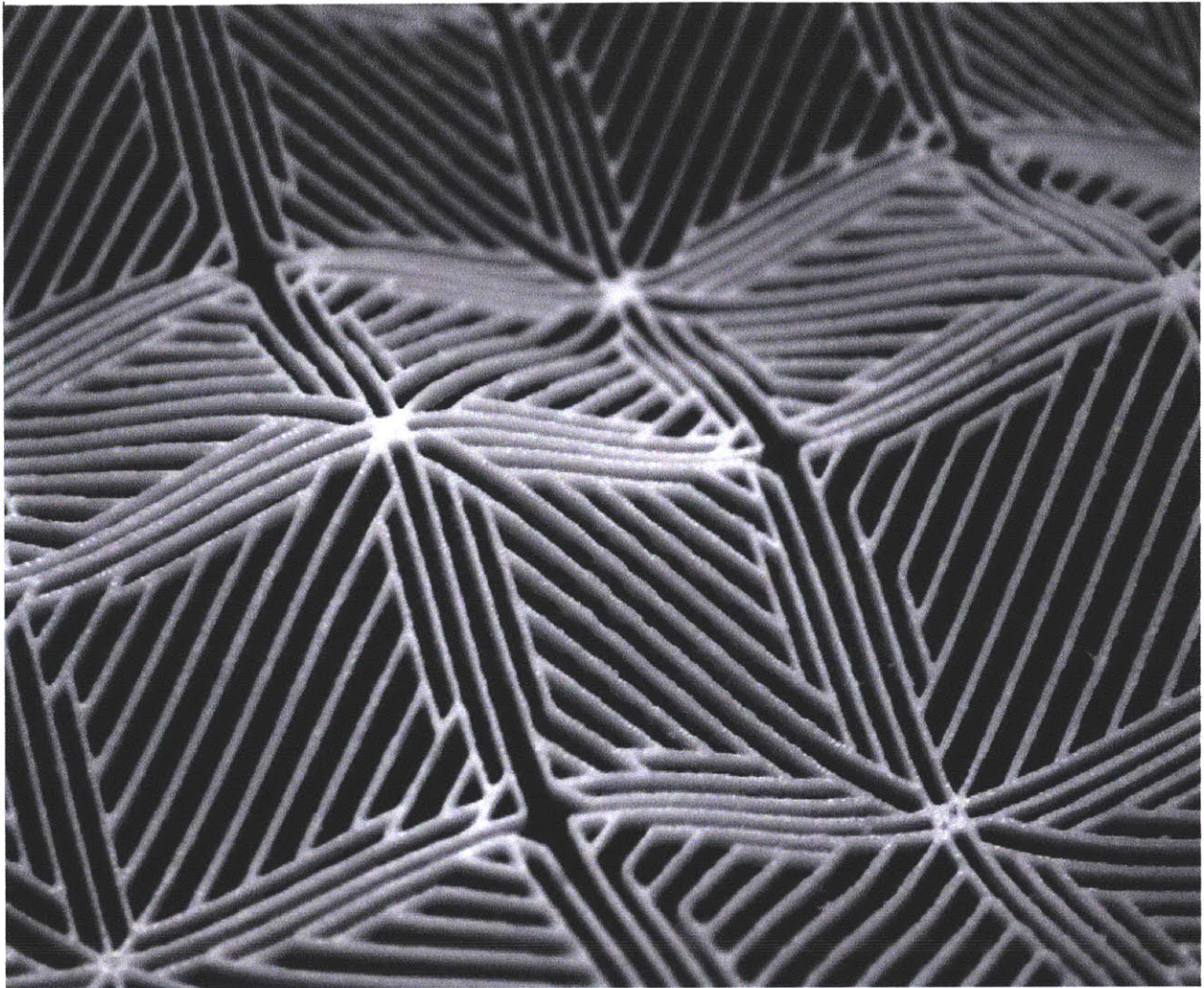
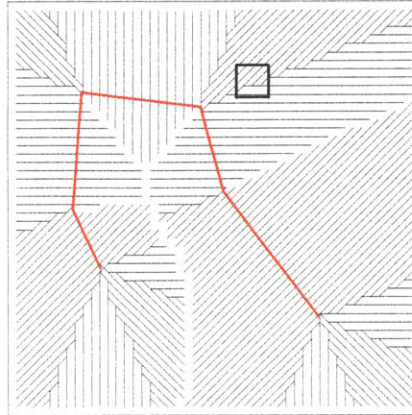
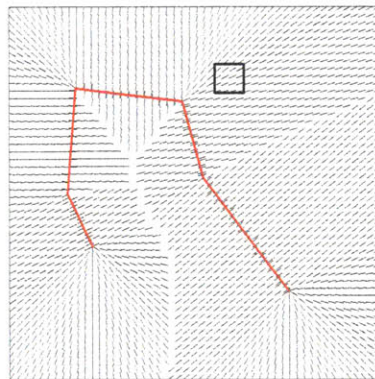


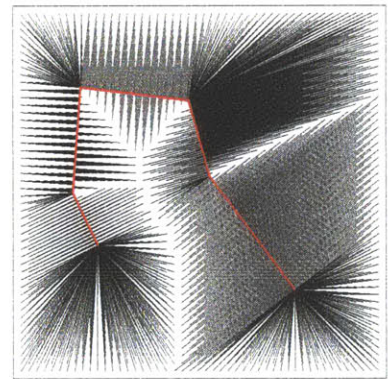
Figure. 4.20. Physical Model of the rationalized surface based on flow direction different surface geometry



Rationalized directions of the closest point



Unitized closest directions to the line segments



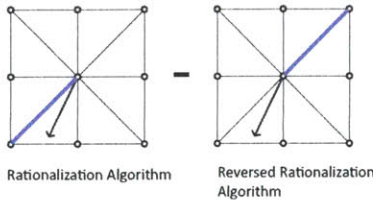
Closest distance to line segment
Simple distance function for each point on the grid which finds the corresponding point on the line segment

Figure. 4.21. a. Rationalized connected network b. Unitized shortest distance direction c. Shortest distance drawn from each point on the grid to the corresponding segment of the

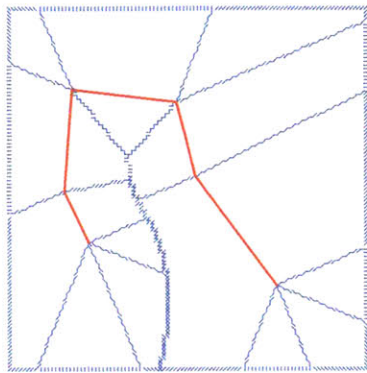
4.5.2. Algorithm description

If we use the same technique for the two dimensional point grid, we can have a connected network of lines which author tends to call the influence area of each segment of the flow drawings (Fig. 4.21). This simplifies the surface generation algorithm since the point grid has been divided into discrete segments which are under the influence of each segment of the line. One of the key points in constructing connected drainage network is that the process of rationalized direction must be applied twice in two complete opposite directions from each other to cover the whole area of the point grids (Fig. 4.22)

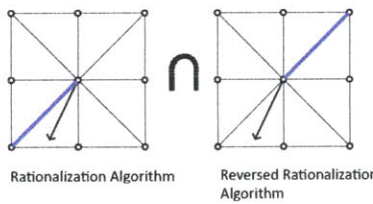
Now that we are dealing with each segment separately, we can measure the shortest distance from each point and translate that into third dimension adding to its height component (Fig. 4.23, 4.24, 4.25). Changing the z_p parameter can result in different surface edge heights and slopes. 55



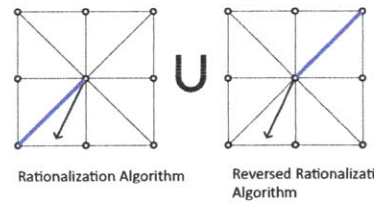
Rationalization Algorithm Reversed Rationalization Algorithm



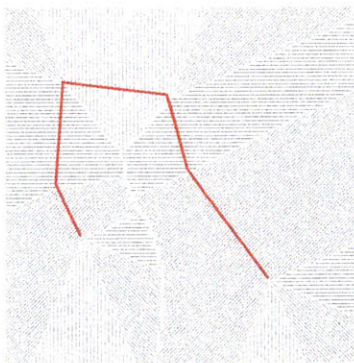
a. Points which are cover



Rationalization Algorithm Reversed Rationalization Algorithm

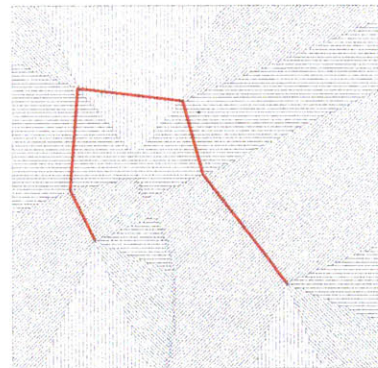


Rationalization Algorithm Reversed Rationalization Algorithm



Areas of over lapping between rational and reverse rational distance algorithm

b.



superimposed both rational and its reversed algorithm on the point grid to cover all connections between grids

c.

Figure 4.22. a. Rationalized connected network b. Unitized shortest distance direction c. Shortest distance drawn from each point on the grid to the corresponding segment of the line.

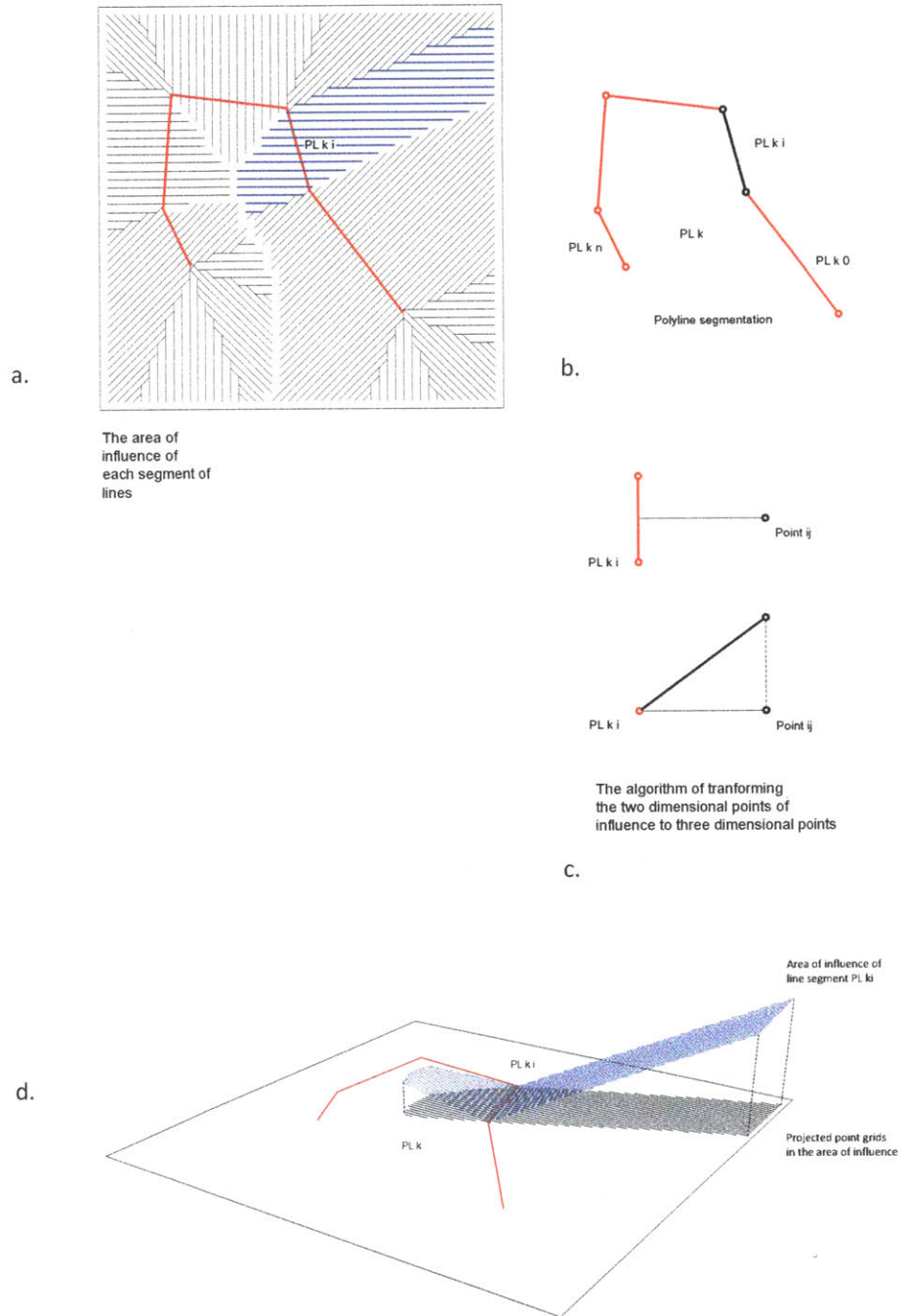
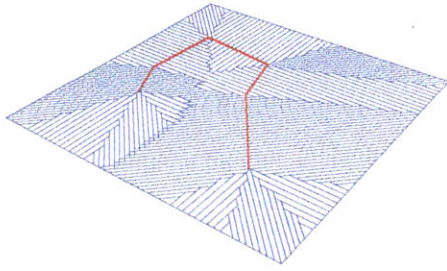
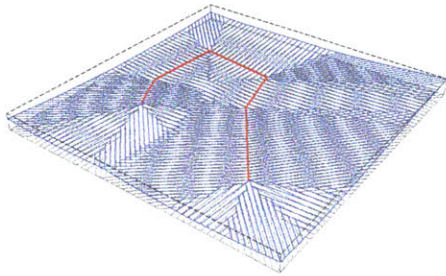


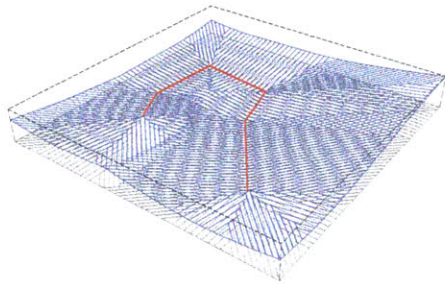
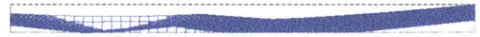
Figure 4.23. a. area of influence of each segment of the line b. line segments of an input polyline c. plan distace from the point to corresponding line segment d. linear translation of distance to height e. three



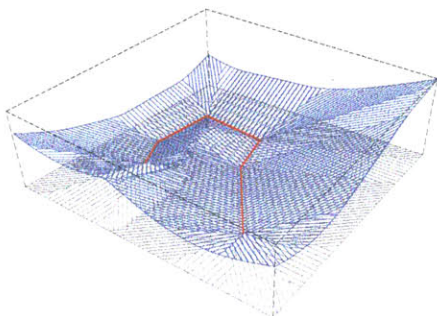
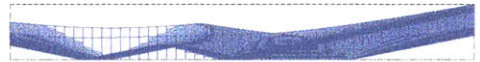
iteration 01
slope = % 0.0



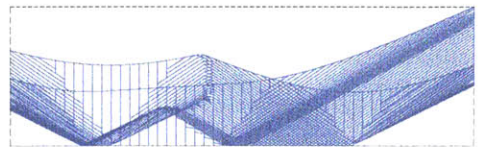
iteration 01
slope = % 0.1



iteration 01
slope = % 0.3



iteration 01
slope = % 0.6



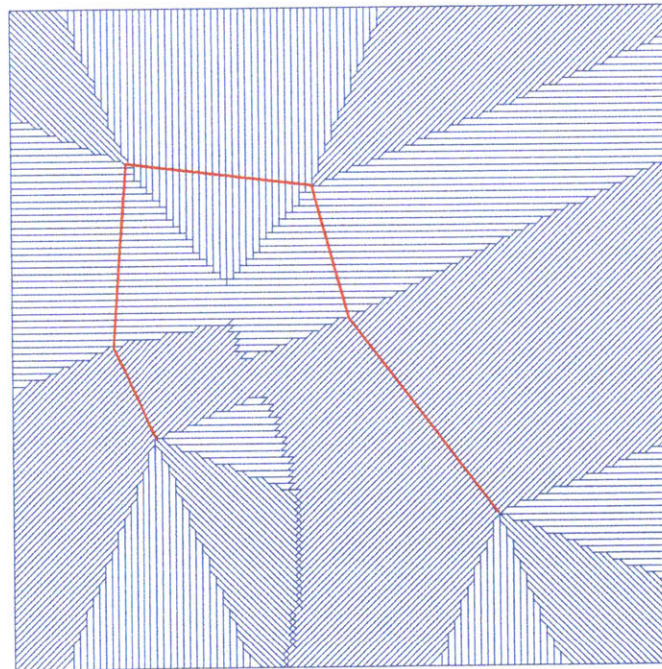
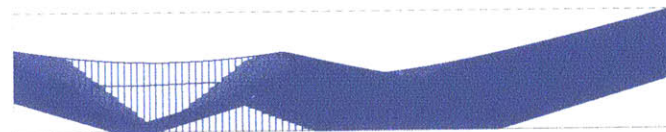
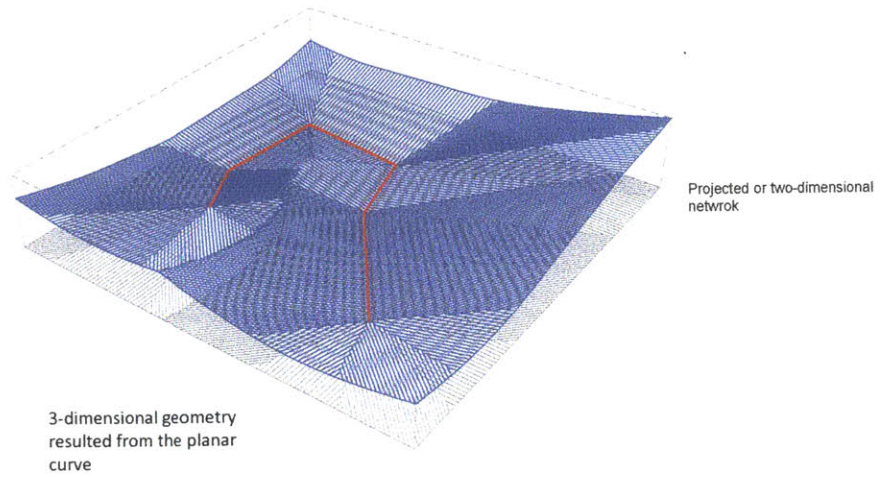


Figure. 4.24, 4.25. Different fraction of z creates different surface slope from completely flat to one to one relationship between the plan distance and height

4.5.3. Global Drainage

So far we achieved surface generation algorithm using shortest distance in plan projection. This allows designers to start with planar curves and construct surfaces which drain into the planar curves (Fig. 4.26). What if the global drainage of the surface is intended. In other word, how to construct the surface that can drain along a curve. For this reason we need to first reconstruct the curve in three dimension as if a drop of water starts at the highest point of the curve, it follows the slope of the curve without changing its oath along the curve till it arrives to the lowest height of the curve.

There are three major types of curve, which can be used in this algorithm: poly lines, control point curves, and branching control point curves [Figure. 58]. The process of constructing spatial curve is quite simple. Each time the control points of the curves are calculated and based on the input slope parameter and length of the curve, the height of each control point is adjusted. The result is a spatial curve which cannot sit on a single plane in three dimension space. For branching geometry the method is still the same with a difference that in branching poly line, each time, the designer must draw the poly line from the root.

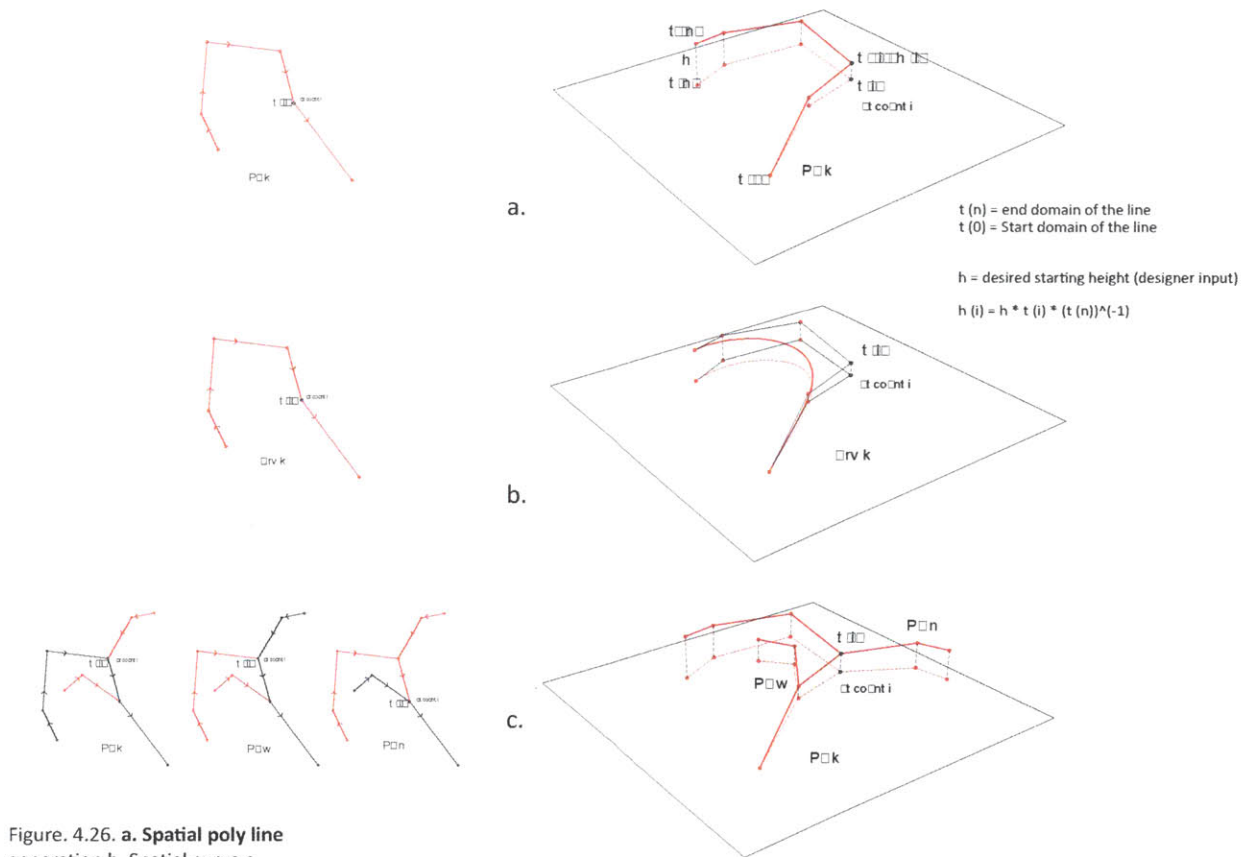
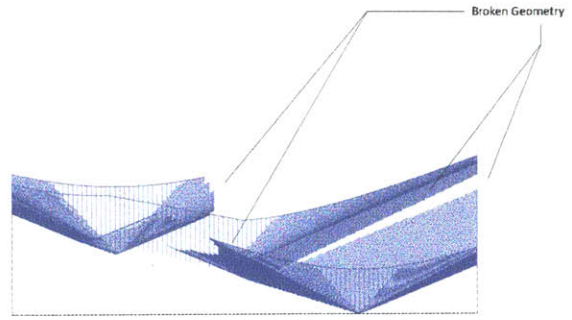
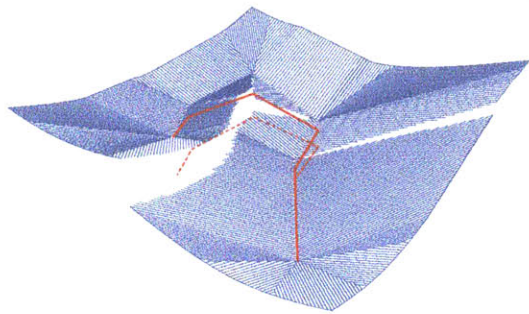
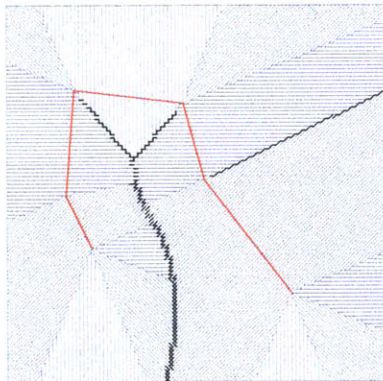


Figure. 4.26. a. Spatial poly line generation b. Spatial curve c. Spatial branching poly lines and control poly lines



Break in the geometry
the case of literal translation of
slopped side algorithm into
slopped curve simultaneously

4.5.4. Point grid pre-transformation

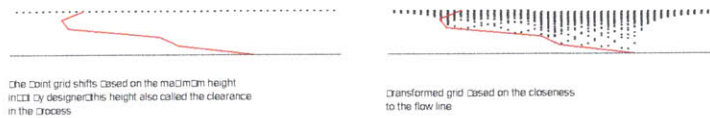


The broken part of geometry are highlighted in dark
from plan point of view

Direct application of height transformation of grids into the spatial curve might result into break in geometry (Fig. 4.27). In order to overcome this problem we need to transform the original point grid prior to shortest distance algorithm. For this propose, first we need to construct the curves in three dimension, then transfer the point grid to the highest point of the curve then relocate the points based on the distance to the projected curve on the plane (Fig. 4.28)

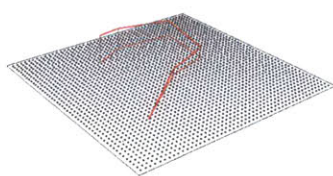
After this step, the point grid is ready to be used as a basis for linear transformation in height. In other word, the superimposition of two technique of re-transformation and height change together will result in a surface which has a global direction of drainage (Fig. 4.29)

Figure. 4.27. Break in geometry resulted from direct translation in plan and height based on spatial curve

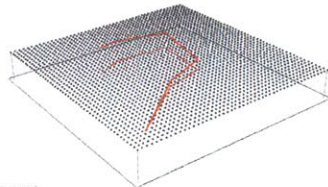


The Point grid shifts based on the maximum height in the process
this height also called the clearance in the process

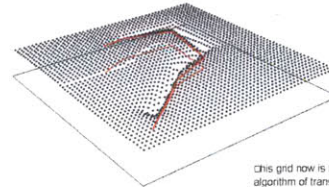
Transformed grid based on the closeness to the flow line



Algorithm of transforming the curve and point grid



Algorithm of transforming the curve and point grid



This grid now is used as a base for the second algorithm of transformation
Algorithm of transforming the curve and point grid

Figure. 4.28. Point Grid Pre-transformation based on the spatial curve

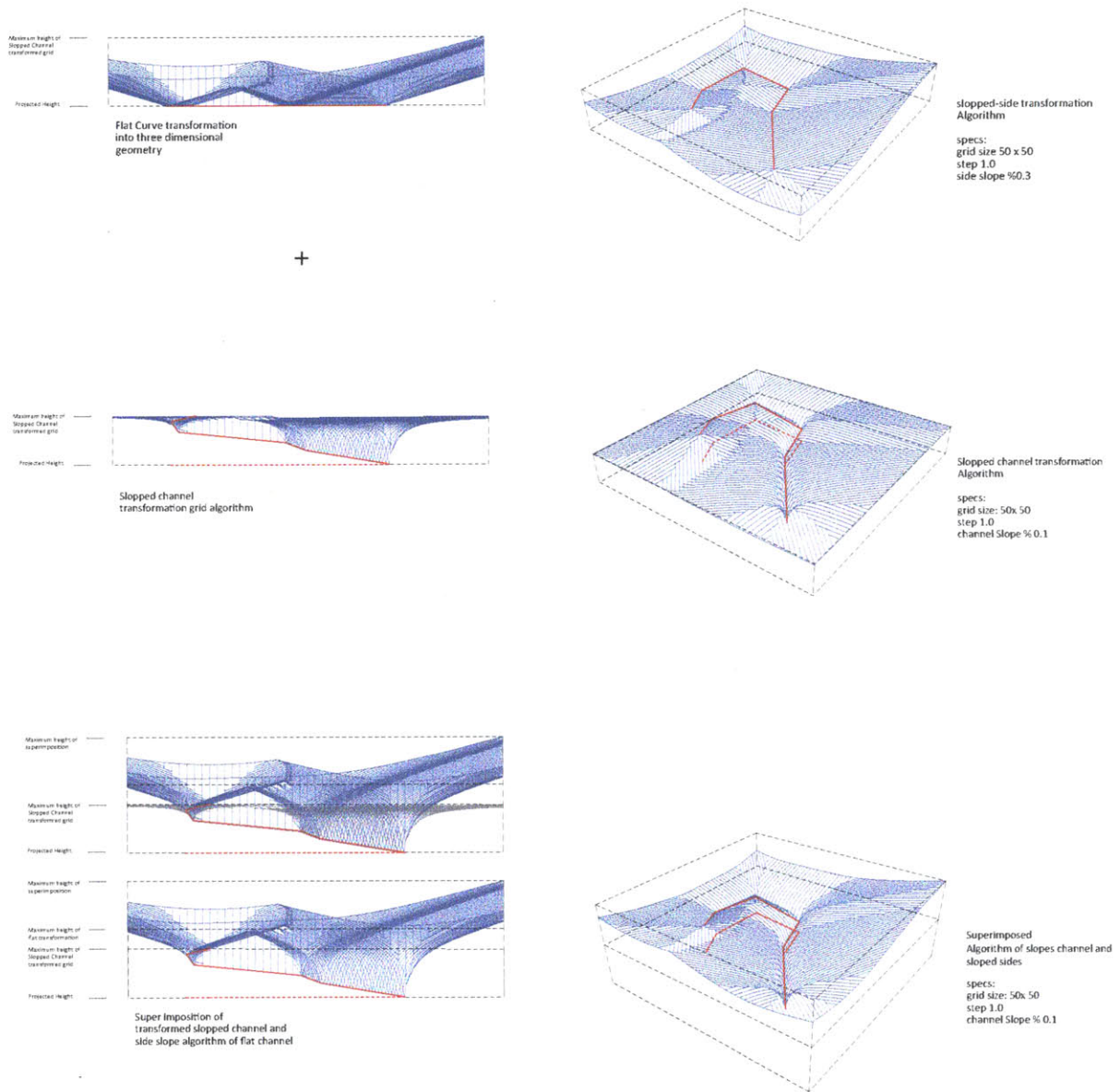


Figure. 4.29. Superimposition of two steps of the algorithm: Linear height change and re-transformation of point grid due to spatial curve

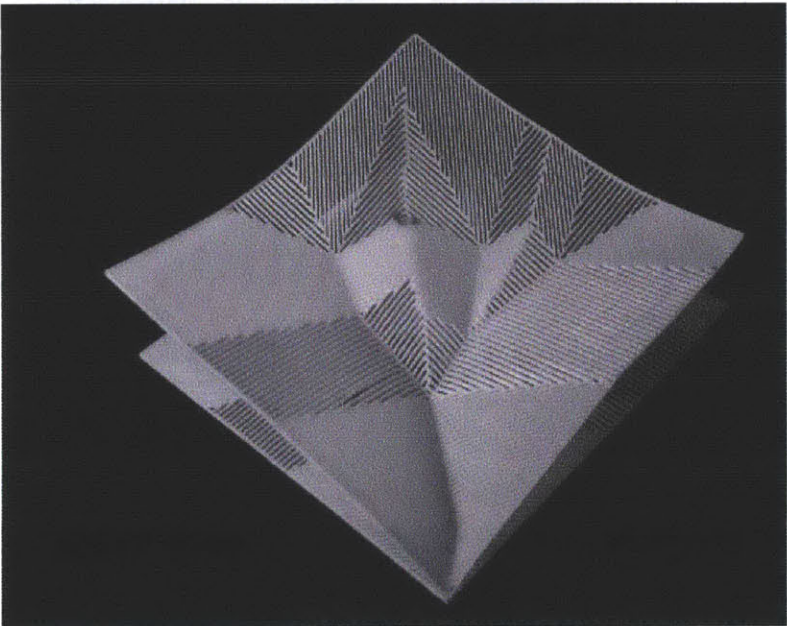
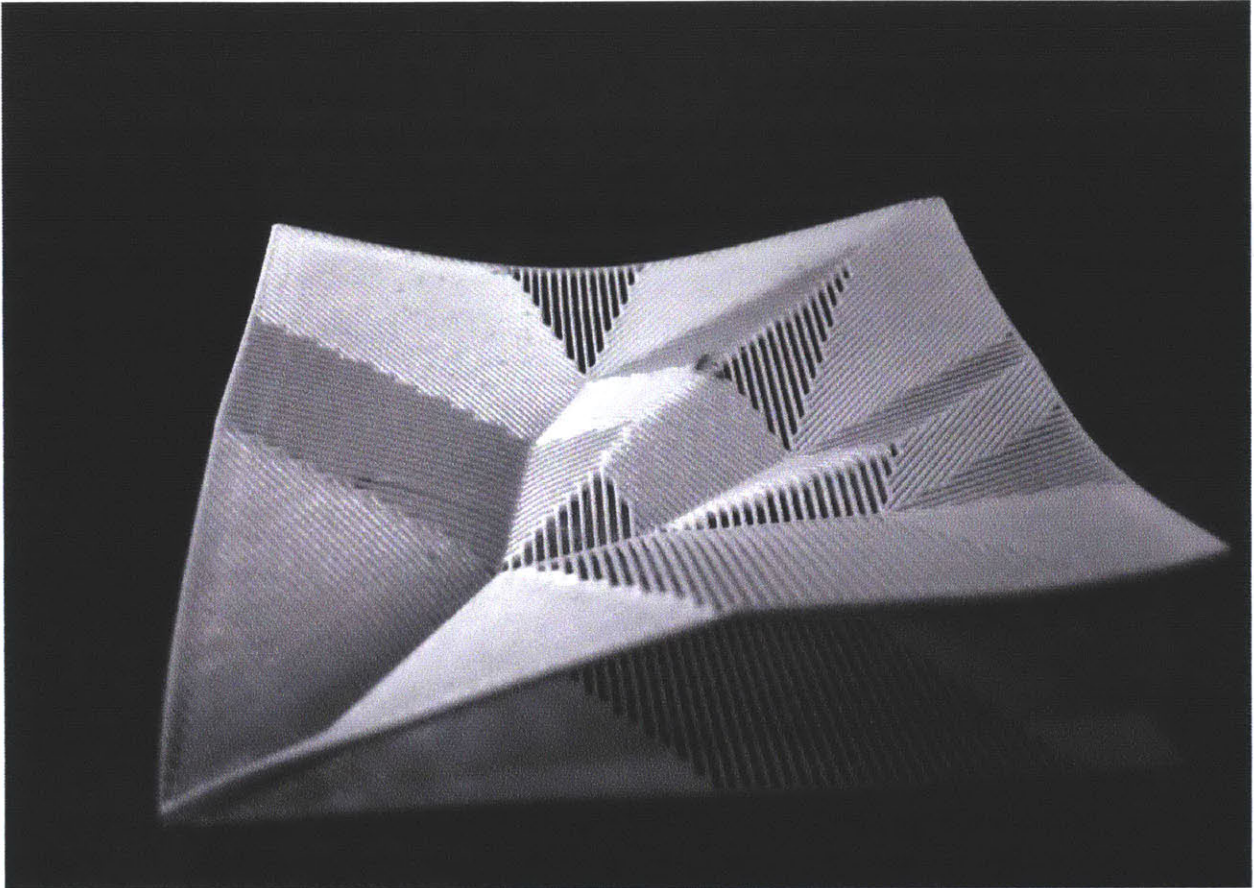


Figure.4.30. Design Sample Using braching polylines. Courtesy of Masoud Akbarzadeh

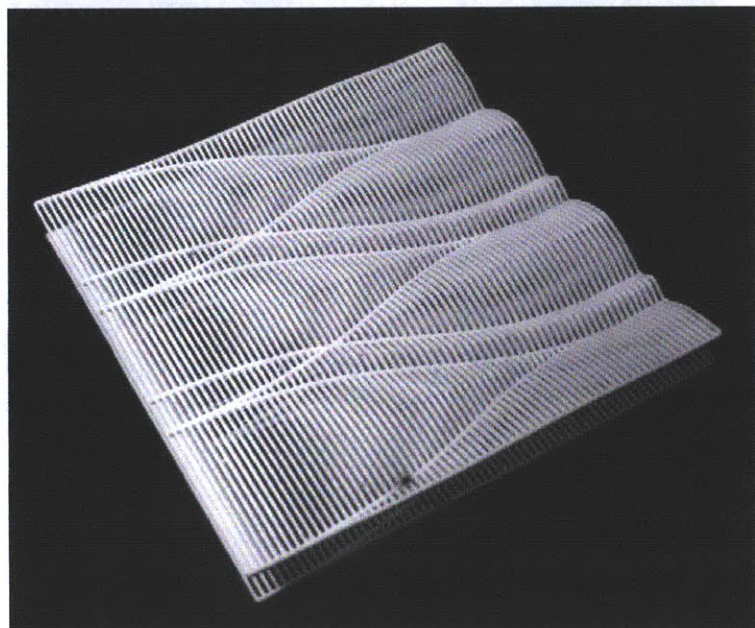
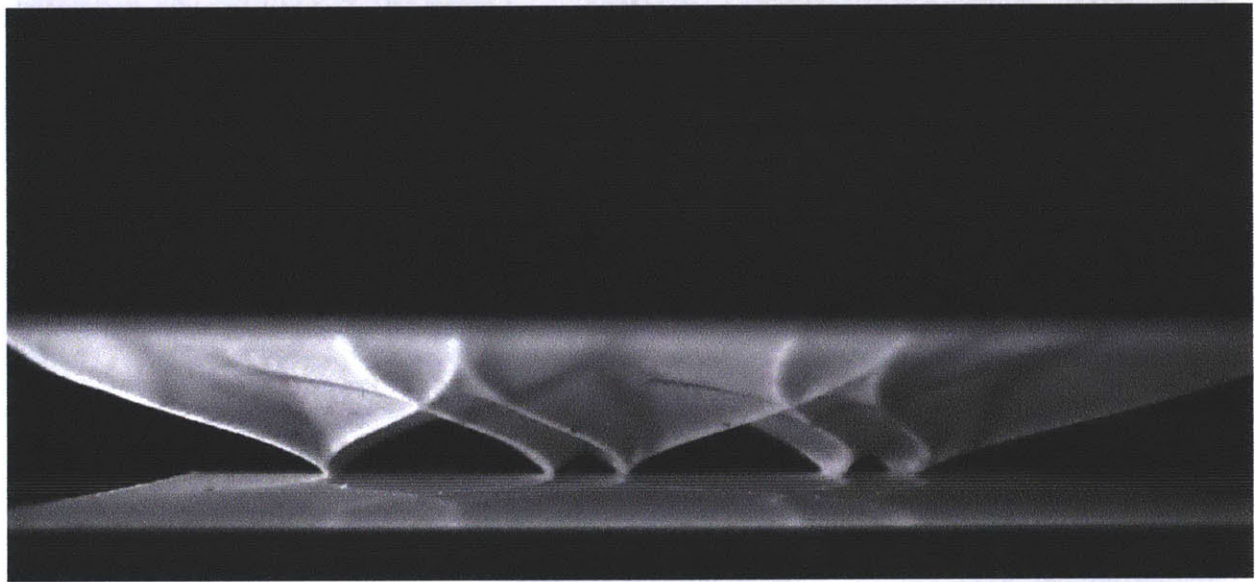
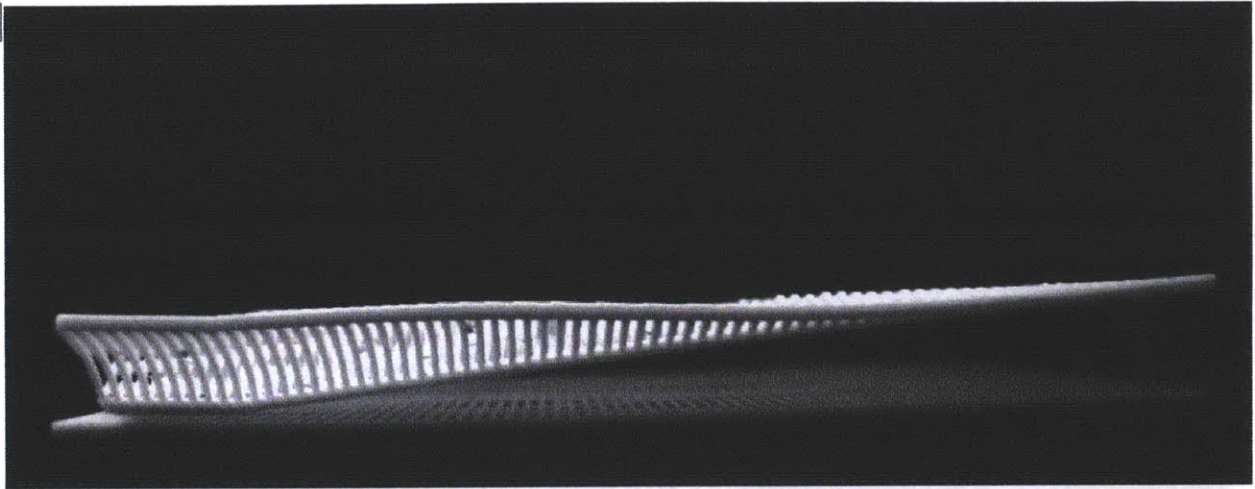


Figure 4.31. Design sample using only plan drawings of curves.
Credit: Joel Lamere

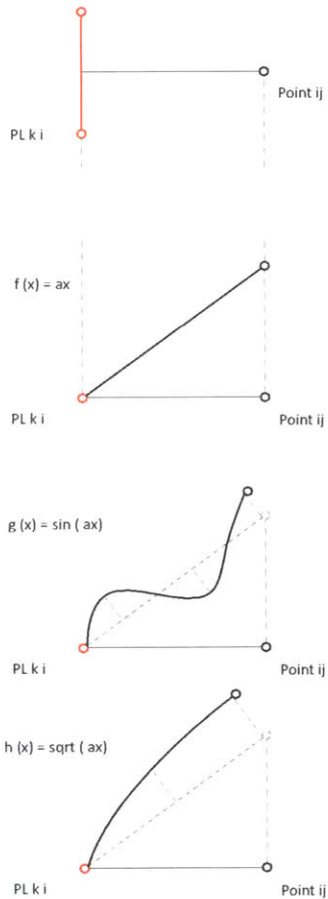


Figure. 4.32. Linear versus non-linear transformation of point into 3-D space

4.5.5. Non-Linear Transformation of height

The inherent potential of this algorithm is in its linear transformation of point grids from two dimensional space to three dimension. This transformation can be easily changed into other non-linear or trigonometric functions (Fig. 4.32). This adds to the variety of different complex geometries that can be constructed using this algorithms (Fig. 4.33).

4.6. Summary

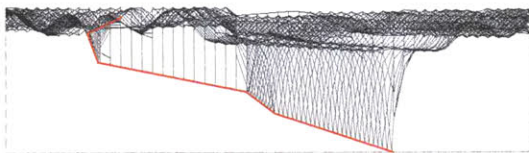
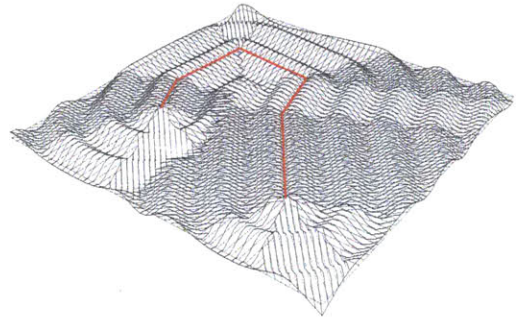
In this chapter three algorithm for reconstruction of surface using two dimensional curves were offered. Algorithms version 1. And 2. Operate based on the behavior of surface patches. In both algorithms a two dimensional point grid transforms in the space and generates three dimensional surface. They both emphasize on the idea of generating ridge lines in topography that directs the flow on surface. These algorithms are similar in the definition, but their resulting geometry is slightly different. The algorithm version 2 generates cleaner topography in comparison with Algorithm version 1.

Algorithm version 3. Is based on the idea of shortest distance of flow. According to this notion, the point grids on the two dimensional transform into three dimensional space through linear transformation of their closest distance to the input flow line. The Algorithm Version 3. can generate complex surfaces with the property of directing the flow based on the original two-dimensional input drawing.

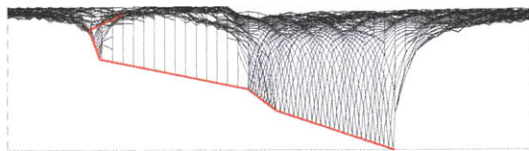
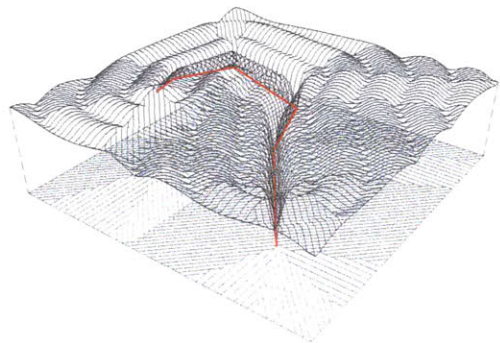
Designing Performative Surfaces



lat rve
 se cs
 side slo e
 ide nction in



lo ed rve
 se cs
 side slo e
 channel slo e
 ide slo e nction in



lo ed rve
 se cs
 side slo e
 channel slo e
 ide slo e nction in

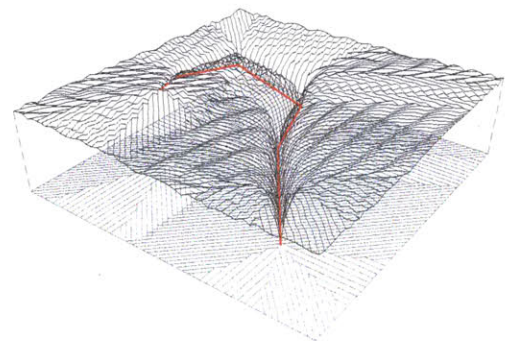


Figure 4.33. Use of non-linear transformation in generating surface geometry

Chapter Five: Conclusions

5. Conclusions

5.1. Summary

This research presents a new tool for designing complex surfaces using only plan drawings.

- Using simple and adjustable plan drawings to start

This tool uses simple input drawings to start and interpolates them through a series of computational algorithms to create 3 dimensional geometry. These drawings are easily adjustable and the result of is quickly accessible.

- Easy tool for design and sketch

This tool provides designers with a powerful tool to explore unique geometries using plan drawings. It works with different groups of curves and result is very fast.

- Unique Geometries not achievable using existing tools

The geometries resulted from this tool are unique and existing tools and methods such as interpolation curves, contour methods, and triangulation are capable of producing such geometries in a fast and intuitive way as this tool creates.

- Constructible elements in result

This tool uses rationalized directional connection between the cells. This limits the connection angles to 45 degree (projected in plan) and members to 2 dimensional elements. As a result construction of these surfaces are easily possible using 2 and a half axis milling machines.

- Water collection performance

Since the whole idea of the surface generation is based on the flow patterns, the first and foremost exquisite result of this tool is types of geometries which can collect and direct water. This is an extremely important performance in designing complex geometries, in large scale, such as airport roofs and sheds and buildings which extend horizontally.

- Structural performance

One of the assumptions made through the process of completing the algorithm was the shortest distance of flow in generating the geometry. This concept has structural values as well. Since forces in structures always search for shortest paths in the system to transfer the loads to the ground. Undoubtedly, this claim requires further exploration of structural properties of such surfaces.

5.2. Discussions and further developments

Definitely there are multiple directions to complete and utilize this tool in future. This tool requires some criteria to evaluate its performance and capabilities. The evaluations in terms of design and analysis resulted in the following discussions:

- Edge control

At the moment, the designer only can control the clearance or the maximum height of the roof. This property can be advanced with further improvement of the algorithm to provide the designers with more control over the edge geometry.

- Topological improvements

There are some topological improvements that can be added to the algorithm of surface generation such as holes and openings. This is a very simple improvement that can be added to the algorithm.

- Structural potpourris

Finally, There are some inherent structural properties in the result of this algorithm as it always results in non linear folded plate surfaces which requires further explorations. The value of this research is in the simplicity of the process of design. Starting with simple 2 dimensional curves and generating three dimensional geometry which has structural properties as well.

References

Cayley, A. (1859) "On contour and slope lines. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science. XVIII, pp. 264-268.

Eavns, Robin, (1995), *The Projective Cast: architecture and its three geometries*, MIT press, pp. 366-370

Howard, Arthur David. (1967) "Drainage analysis in geologic interpretation: a summation," *American Association of Petroleum Geologists Bulletin*, v. 51, p. 2246-2259.

Huber, Stefan, (2011) *Computing Straight Skeletons and Motorcycle Graphs: Theory and Practice*, Department of Computer Science, University of Salzburg, Austria

Maxwell, J.C. (1870) "On contour lines and measurements of heights," *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science* 40, pp.421-427

Morse, S. P. (1965), "A mathematical model for the analysis of contour line data", *Technical Report*, Dept. Electrical Engineering, New York University, New York, pp. 400-124

Nelson, Stephen A. *Physical Geology*, EENS 111, Tulane University, <http://www.tulane.edu/~sanelson/eens1110/streams.htm>

Pfaltz, J. L. (1976) "Surface networks." *Geographical Analysis* 8(1), pp.77-93.
Rinaldo, Andrea, and Ignacio Rodriguez-Iturbe,(2001) *Fractal River Basins: Chance and Self-Organization*, Cambridge University Press

Schneider, Bernard and Jo Wood,(2004) *Construction of Metric Surface Networks from Raster-Based DEMs, Topological Data Structures for Surfaces*, ed. Sanjay Rana, John Wiley and Sons, Ltd, pp. 53-70

Schneider, B. (2003) "Surface networks: extension of the topology and extraction from bilinear surface patches," 7th International Conference on Geocomputation.

Figures

Figure 1.0. Arthur David Howard, Drainage Analysis in Geologic Interpretation A Summation [Howard 1967].

Figure 1.1. parallel projection
vs Conical Projections

Figure 1.2. Surface geometry of the Earth, dendritic drainage pattern [A].

Figure 1.3. Interpolation of Sectional Curves. a. Design curves / Required curves to start. b. Plan location of each sectional curve. c. Interpolated result/ Surface of the curves.

Figure 1.4. Contour manipulation
a. design requirement elements. b. plan representation of contoured based design. c. Surface result of the contour drawn plan

Figure 1.5. Digital Elevation Model representation of a surface. a. Design elements as pixels and their colors. b. Image representation of surface as pixels ranging from black to white. c. Surface result of elevation data

Figure 1.6. Triangulation. a. Vertices and their corresponding triangles. b. Plan representation of triangulated vertices. c. Mesh Surface Representation of triangles.

Figure 1.7. Plan drawings of a drainage pattern

Figure 1.4. Contour manipulation
a. design requirement elements. b. plan representation of contoured based design. c. Surface result of the contour drawn plan

Figure 1.5. Digital Elevation Model representation of a surface. a. Design elements as pixels and their colors. b. Image representation of surface as pixels ranging from black to white. c. Surface result of elevation data

Figure 1.6. Triangulation. a. Vertices and their corresponding triangles. b. Plan representation of triangulated vertices. c. Mesh Surface Representation of triangles.

Figure 1.7. Plan drawings of a drainage pattern

Figure 2.1. Performative surface algorithms. a. Algorithm 1. b. Algorithm 2. c. Algorithm 3.

Figure 2.2. a. Critical Graph consisting of maximum, minimum and saddle points and the lines connecting these points to each other. b. simplified version of critical graph

Figure 2.3. a. Surface Network Graph b. Contour Extraction algorithm from Surface Network Graph

Figure 2.4. a. Surface Network Module b. Surface Network Module Transformed

Figure 2.5. Surface Generation Algorithm 3.

Figure. 3.1. a. Circular Indicatrix b. Hyperbolic Indicatrix c. Parabolic Indicatrix

Figure. 3.2. a. indicatrix or contour is a closed curve b. It encompasses other smaller indicatrices inside c. By Travelling in z direction the indicatrices become smaller and smaller and finally become a peak point

Fig. 3.3. a. indicatrix or contour is a closed curve b. It encompasses other smaller indicatrices inside c. By Travelling in (-z) direction the indicatrices become smaller and smaller and finally become a minimum point

Figure. 3.4. a. There is a point in topography where four indicatrices meet each other. b. the contours in this point look like hyperbola c. Saddle point of topography

Figure. 3.5. a. The drainage patterns are always perpendicular on contour lines b. There is only one path exist, If we travel from saddle point with steepest paths upward or downward which connect saddle point to maxima and minima. c. series of drainage paths on the topography. d. plan view of slope paths and contour paths

Figure. 3.6. a, b. Local maxima and local minima and saddle points on the topography. c, d dale or valley. e, f. Hills

Figure. 3.7. Relationship between Peak, Pits and passes (Maxima, minima and Saddle points)

Figure.3.8. a. Saddle point Mathematical definition. b. Local Maximum. c. Local Minimum d. Coarse line, the line which connects saddle point to maximum points. e. Ridge lines which connects saddle points to minima

Figure. 3.9. a. Projecting a point grid onto the geometry of a surface to construct square surface patches b. locations of extremum points of the surface as well as the saddle points

Figure. 3.10. a. Surface patch and central vertex and its surrounding. b. Saddle point on the surface. c. Saddle point on the grid. d. Minimum point e. Maximum Point.

Figure. 3.11. Locating the saddle points on the topography and finding the principal directions to go up or down

Figure. 3.12. Different resolution of path finding. a. 45 Degree Resolution, top path downward, bottom, path upward. b. 30 degree resolution, top path downward, bottom, path upward. c. 7.5 degree resolution, top, path downward, bottom, path upward

Figure. 3.13. a. **axometric** view of the surface with its surface network graph. b. plan view of the surface with surface network graph.

Figure. 3.14. a. axometric and elevation view of general surface network diagram. b. Plan view of different types of surface network after [Rana 2002].

Figure.3.15. Subdivision of the graph into three main parts: upper, middle and lower parts.

Figure. 3.16. a. The lower part of the graph and lower bound of the middle part b. upper part of the graph with the upper bound of the middle part.

Figure. 3.17. a. Middle part of the graph
b. Completing the contours by adding all parts together

Figure. 3.18. Topologically Similar networks

Figure. 3.19. a. Simple module of Surface network Graph. b. Aggregation of the module c. Contour extraction result d. Final topography

Figure. 3.20. a. Extracted contours from the network graph with sharp angle connections b. Chamfer algorithm to change the corners step 1, c. Chamfered Algorithm Step 2, d. Chamfered algorithm Step 3.

Figure. 4.1. Change in height of each point of the graph changes the properties of the graph. a. complete graph b. Graph with incomplete ridge coarse lines c. graph with incomplete ridge and coarse lines d. graph with new ridge lines.

Figure. 4.3. Step by Step Transformatin Algorithm

Figure. 4.4. Surface transformation Algorithm

Figure. 4.5. Different results of algorithm through the completion process. a. Propagation process b. Cell activation process c. Surface transformation process

Figure. 4.6. Physical Model of the transformation process

Fig. 4.7, 4.8. Step by Step process of activation of cells.

Figure. 4.9. Surface transformation Algorithm

Figure. 4.10. a. Algorithm 2 b. Algorithm 1

Figure. 4.11. a. Point grid b. input Flow pattern on the point grid

Figure. 4.12. a. Point grid b. input Flow pattern on the point grid

Figure. 4.13. each cell is compared to eight primary directions of the flow to rationalize the unitized direction vector

Figure. 4.14. Flow Path finder algorithm a. Single step 45 degree resolution b. Single Step 7.5 degree resolution c. 50 step 45 degree resolution d. 50 step 7.5 degree resolution

Figure. 4.15. Slope Finder algorithm a. upward direction b. Downward direction c. superimposition of both d. surface network graph layered on top of the flow pattern

Figure. 4.16. spatializing the flow patterns. note that the geometry is not very clean since there is no rationalization applied yet.

Figure. 4.17. Flow path finding on another type of surface with different resolutions.

Figure. 4.18. a. Flow pattern finding algorithm 45 degree angle resolution 50 steps of slope finding b. 45 degree of resolution first step of slope finding c. Connected networks using direction rationalization technique

Figure. 4.19. Physical Model of the rationalized surface based on flow direction.

Figure. 4.20. Physical Model of the rationalized surface based on flow direction different surface geometry

Figure. 4.21. a. Rationalized connected network b. Unitized shortest distance direction c. Shortest distance drawn from each point on the grid to the corresponding segment of the line.

Figure. 4.22. a. Rationalized connected network b. Unitized shortest distance direction c. Shortest distance drawn from each point on the grid to the corresponding segment of the line.

Figure. 4.23. a. area of influence of each segment of the line b. line segments of an input polyline c. plan distance from the point to corresponding line segment d. linear translation of distance to height e. three dimensional

Figure. 4.24, 4.25. Different fraction of z creates different surface slope from completely flat to one to one relationship between the plan distance and height

Figure. 4.26. **a. Spatial poly line** generation b. Spatial curve c. Spatial branching poly lines and control poly lines

Figure. 4.27. Break in geometry resulted from direct translation in plan and height based on spatial curve

Figure. 4.28. Point Grid Pre-transformation based on the spatial curve

Figure. 4.29. Superimposition of two steps of the algorithm: Linear height change and re-transformation of point grid due to spatial curve

Figure.4.30. Design Sample Using braching polylines. Courtesy of Masoud Akbarzadeh

Figure. 4.31. Design sample using only plan drawings of curves.
Credit: Joel Lamere

Figure. 4.32. Linear versus non-linear transformation of point into 3-D space

Figure. 4.33. Use of non-linear transformation in generating surface geometry

Appendix

```
# This component is Written by
# Masoud Akbarzadeh @ MIT Departement of Architceture
# SMArchs Computation 2012
# All right reserved
# Perfrmative Surfaces, use of Flow as mean to draw complex geometries
# Please Do not distribute without the permission of the Author
```

```
import rhinoscriptsyntax as rs
import math
```

```
slope=IN_slope
x=IN_x
y=IN_y
step=IN_steps
OUT_structure=[]
OUT_projected=[]
z=IN_height
crvs_s=IN_sloped_curves
crvs_f=IN_Flat_curves
d=IN_relative_thickness
```

```
OUT_structure2=[]
```

```
crvs=[]
crvs_f=[]
crvs_s=[]
```

```
crvs_s=IN_sloped_curves
crvs_f=IN_Flat_curves
```

```
#crvs_s=rs.GetObjects("please enter the curves which meant to be sloped")
#crvs_f=rs.GetObjects("please enter the flat curves")
```

```
if crvs_s:
    for i in range(len(crvs_s)):
        crvs.append([crvs_s[i], [1]])
```

```
if crvs_f:
    for i in range(len(crvs_f)):
        crvs.append([crvs_f[i], [0]])
```

```
rs.EnableRedraw(False)
```

```
def slopizer(crv):
    cptt=[]
    dist=[]
    ptcounts=[]
    cpt=rs.CurvePoints(crv[0])
    pts=rs.CurveStartPoint(crv[0])
    cdom=rs.CurveDomain(crv[0])
    length=rs.CurveLength(crv[0])
```

```

if crv[1]==[1]:
    for i in range(len(cpt)):
        ptcnts.append(cpt[i])
        param=rs.CurveClosestPoint(crv[0],ptcnts[i])
        cptt.append([ptcnts[i][0]
,ptcnts[i][1],(slope)*param*length/cdom[1]])
    else:
        for i in range(len(cpt)):
            ptcnts.append(cpt[i])
            param=rs.CurveClosestPoint(crv[0],ptcnts[i])
            cptt.append([ptcnts[i][0] ,ptcnts[i][1],0])
#rs.CurrentLayer("LINE 02")
maxh=[]
for i in range(len(cptt)):
    maxh.append(cptt[i][2])
mah=max(maxh)
deg=rs.CurveDegree(crv[0])
return [rs.AddCurve(cptt, deg),mah]

```

```

def vectorize(pt,crvs):
    dist=[]
    for k in range(len(crvs)):
        param=rs.CurveClosestPoint(crvs[k][0],pt)
        pttest=rs.EvaluateCurve(crvs[k][0],param)
        dist.append(rs.Distance(pt,pttest))

    for k in range(len(crvs)):
        param=rs.CurveClosestPoint(crvs[k][0],pt)
        pttest=rs.EvaluateCurve(crvs[k][0],param)
        distt=rs.Distance(pt,pttest)
        if min(dist)==distt:
            return [min(dist),crvs[k]]

```

```

dist={}
pth={}
pts={}
pths={}

```

```

points={}
dist={}
pth1={}
h=[]

```

```

for crv in (crvs):
    test=slopizer(crv)
    h.append(test[1])
mah=max(h)

```

```

for i in range(x+1):
    for j in range(y+1):
        points[(i,j)]=[i*step,j*step,0]
        pth1[(i,j)]=[points[(i,j)][0], points[(i,j)][1],mah]
        dist[(i,j)]=vectorize(points[(i,j)],crvs)

```

```

pth2={}
prox=2
dist2={}

```

```

for i in range(len(crvs)):
    curve=slopizer(crvs[i])
    OUT_structure.append(curve[0])

```

```

str={}
for i in range(x+1):
    for j in range(y+1):
        param=rs.CurveClosestPoint(dist[(i,j)][1][0],points[(i,j)])
        pttest=rs.EvaluateCurve(dist[(i,j)][1][0],param)
        line=rs.AddLine(pttest,[pttest[0], pttest[1], 100])
        crv2=slopizer(dist[(i,j)][1])
        int=rs.CurveCurveIntersection(line,crv2[0])
        rs.DeleteObject(line)
        pttest2=int[0][1]
        dist2[(i,j)]=rs.Distance(pttest, pttest2)
        dist2[(i,j)]=mah-dist2[(i,j)]
        pth2[(i,j)]=[pth1[(i,j)][0], pth1[(i,j)][1],mah-
dist2[(i,j)]/(dist[(i,j)][0]+1)]

```

```

smooth=IN_smooth

```

```

if smooth>0:

```

```

    for o in range(smooth):
        for i in range(x+1):
            for j in range(y+1):

```

```

                if 0<i<x and 0<j<y:

```

```

                    pth2[(i,j)]=[pth2[(i,j)][0],pth2[(i,j)][1],(pth2[(i+1,j)][2]+
                    pth2[(i+1,j+1)][2]+
                    pth2[(i,j+1)][2]+ pth2[(i-1,j+1)][2]+pth2[(i-1,j)][2]+
                    pth2[(i-1,j-1)][2]+
                    pth2[(i,j-1)][2]+pth2[(i+1,j-1)][2])/8]
                    #rs.AddPoint(pth2[(i,j)])

```

```

                    elif i==0:
                        if 0<j<y:

```

```

                            pth2[(i,j)]=[pth2[(i,j)][0],pth2[(i,j)][1],(pth2[(i+1,j)][2]+
                            pth2[(i+1,j+1)][2]+
                            pth2[(i,j+1)][2]+pth2[(i,j-1)][2]+pth2[(i+1,j-
                            1)][2])/5]
                            #rs.AddPoint(pth2[(i,j)])

```

```

                    elif i==x:

```

```

        if 0<j<y:
            pth2[(i,y-
1)]=[pth2[(i,j)][0],pth2[(i,j)][1],(pth2[(i,j+1)][2]+ pth2[(i-
1,j+1)][2]+pth2[(i-1,j)][2]+
                pth2[(i-1,j-1)][2]+pth2[(i,j-1)][2])/5]
            #rs.AddPoint(pth2[(i,j)])

    elif j==0:
        if 0<i<x:

pth2[(i,j)]=[pth2[(i,j)][0],pth2[(i,j)][1],(pth2[(i+1,j)][2]+
pth2[(i+1,j+1)][2]+
                pth2[(i,j+1)][2]+ pth2[(i-1,j+1)][2]+pth2[(i-
1,j)][2])/5]
                #rs.AddPoint(pth2[(i,j)])

    elif j==y:
        if 0<i<x:

pth2[(i,j)]=[pth2[(i,j)][0],pth2[(i,j)][1],(pth2[(i+1,j)][2]+
                pth2[(i-1,j)][2]+ pth2[(i-1,j-1)][2]+
                pth2[(i,j-1)][2]+pth2[(i+1,j-1)][2])/5]
                #rs.AddPoint(pth2[(i,j)])

        if i==0:
            if j==0:

pth2[(i,j)]=[pth2[(i,j)][0],pth2[(i,j)][1],(pth2[(i+1,j)][2]+
pth2[(i+1,j+1)][2]+
                pth2[(i,j+1)][2])/3]
                #rs.AddPoint(pth2[(i,j)])

        if i==x:
            if j==y:
                pth2[(i,j)]=[pth2[(i,j)][0],pth2[(i,j)][1],(pth2[(i-
1,j)][2]+ pth2[(i-1,j-1)][2]+
                pth2[(i,j-1)][2])/3]
                #rs.AddPoint(pth2[(i,j)])

        if i==0:
            if j==y:

pth2[(i,j)]=[pth2[(i,j)][0],pth2[(i,j)][1],(pth2[(i+1,j)][2]+
                +pth2[(i,j-1)][2]+pth2[(i+1,j-1)][2])/3]
                #rs.AddPoint(pth2[(i,j)])

        if j==0:
            if i==x:
                pth2[(i,j)]=[pth2[(i,j)][0],pth2[(i,j)][1],(
                pth2[(i,j+1)][2]+ pth2[(i-1,j+1)][2]+pth2[(i-
1,j)][2])/3]
                #rs.AddPoint(pth2[(i,j)])

pth={}
for i in range(x+1):
    for j in range(y+1):
        param=rs.CurveClosestPoint(dist[(i,j)][1][0],points[(i,j)])
        pttest=rs.EvaluateCurve(dist[(i,j)][1][0],param)

```



```

#####YOU CAN CHANGE THE MATH HERE IN THE NEXT LINE#####
#pth[(i,j)]=[pth2[(i,j)]] [0],
pth2[(i,j)]] [1],pth2[(i,j)]] [2]+math.tan(math.pi/8)*(dist[(i,j)]] [0])*z]
#pth[(i,j)]=[pth2[(i,j)]] [0],
pth2[(i,j)]] [1],pth2[(i,j)]] [2]+math.sqrt(dist[(i,j)]] [0])*z]
pth[(i,j)]=[pth2[(i,j)]] [0],
pth2[(i,j)]] [1],pth2[(i,j)]] [2]+(dist[(i,j)]] [0])*z]

for j in range(y):
    #rs.CurrentLayer("LINE 00")
    OUT_projected.append(rs.AddLine(points[(0,j)],points[(0,j+1)]))
    OUT_projected.append(rs.AddLine(points[(x,j)],points[(x,j+1)]))
for i in range(x):
    #rs.CurrentLayer("LINE 00")
    OUT_projected.append(rs.AddLine(points[(i,y)],points[(i+1,y)]))
    OUT_projected.append(rs.AddLine(points[(i,0)],points[(i+1,0)]))

def director(pts,pttest):
    vec=rs.VectorCreate(pttest, pts[0])
    angle=[]
    vecs=[]
    for i in range(1,len(pts)):
        vecs.append(rs.VectorCreate(pts[i], pts[0]))
    for i in range(len(vecs)):
        angle.append(rs.VectorAngle(vecs[i],vec))
    if min(angle)==angle[0]:
        return [vecs[0], 0]
    if min(angle)==angle[1]:
        return [vecs[1], 1]
    if min(angle)==angle[2]:
        return [vecs[2], 2]
    if min(angle)==angle[3]:
        return [vecs[3], 3]
    if min(angle)==angle[4]:
        return [vecs[4], 4]
    if min(angle)==angle[5]:
        return [vecs[5], 5]
    if min(angle)==angle[6]:
        return [vecs[6], 6]
    if min(angle)==angle[7]:
        return [vecs[7], 7]
    #print max(angle)

for i in range(1,x):
    for j in range(1,y):
        pts[(i,j)]=[points[(i,j)],points[(i+1,j)], points[(i+1,j+1)],
        points[(i,j+1)], points[(i-1,j+1)], points[(i-1,j)], points[(i-1,j-
1)],
        points[(i,j-1)],points[(i+1,j-1)]]
        pths[(i,j)]=[pth[(i,j)],pth[(i+1,j)], pth[(i+1,j+1)],
        pth[(i,j+1)], pth[(i-1,j+1)], pth[(i-1,j)], pth[(i-1,j-1)],
        pth[(i,j-1)],pth[(i+1,j-1)]]

```

```

for i in range(1,x):
    for j in range(1,y):
        param=rs.CurveClosestPoint(dist[(i,j)][1][0],points[(i,j)])
        pttest=rs.EvaluateCurve(dist[(i,j)][1][0],param)
        vec=rs.VectorCreate(pttest,points[(i,j)])
        vec=rs.VectorUnitize(vec)
        pttest=rs.PointAdd(points[(i,j)],vec)
        vec=director(pts[(i,j)],pttest)
        vecl=rs.VectorLength(vec[0])
        #rs.CurrentLayer("LINE 00")
        pttest=rs.PointAdd(points[(i,j)],vec[0])
        if dist[(i,j)][0]< vecl:
            line=rs.AddLine(pttest,points[(i,j)])
            ptend=rs.CurveEndPoint(dist[(i,j)][1][0])
            ptstart=rs.CurveStartPoint(dist[(i,j)][1][0])
            distend=rs.Distance(points[(i,j)],ptend)
            diststart=rs.Distance(points[(i,j)],ptstart)
            int=rs.CurveCurveIntersection(dist[(i,j)][1][0],line)
            if int:
                ptt=int[0][1]
                ptt2=[ptt[0], ptt[1], ptt[2]+100]
                linet=rs.AddLine(ptt, ptt2)
                crv2=slopizer(dist[(i,j)][1])
                int=rs.CurveCurveIntersection(crv2[0],linet)
                rs.DeleteObject(linet)
                rs.DeleteObject(line)
                #rs.CurrentLayer("LINE 00")

OUT_projected.append(rs.AddLine(points[(i,j)], [int[0][1][0],int[0][1][1], 0]
))

    #rs.CurrentLayer("LINE 02")
    if vec[1]==0:
        OUT_structure.append(rs.AddLine(pth[(i,j)],int[0][1]))
    if vec[1]==1:
        OUT_structure.append(rs.AddLine(pth[(i,j)],int[0][1]))
    if vec[1]==2:
        OUT_structure.append(rs.AddLine(pth[(i,j)],int[0][1]))
    if vec[1]==3:
        OUT_structure.append(rs.AddLine(pth[(i,j)],int[0][1]))
    if vec[1]==4:
        OUT_structure.append(rs.AddLine(pth[(i,j)],int[0][1]))
    if vec[1]==5:
        OUT_structure.append(rs.AddLine(pth[(i,j)],int[0][1]))
    if vec[1]==6:
        OUT_structure.append(rs.AddLine(pth[(i,j)],int[0][1]))
    if vec[1]==7:
        OUT_structure.append(rs.AddLine(pth[(i,j)],int[0][1]))
    else:
        #rs.CurrentLayer("LINE 02")
        if vec[1]==0:
            OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i+1,j)]))
        if vec[1]==1:

OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i+1,j+1)]))
    if vec[1]==2:
        OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i,j+1)]))
    if vec[1]==3:

```

```

        OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i-
1,j+1)]))
        if vec[1]==4:
            OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i-1,j)]))
        if vec[1]==5:
            OUT_structure.append(rs.AddLine(pth[(i,j)], pth[(i-1,j-
1)]))
        if vec[1]==6:
            OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i,j-1)]))
        if vec[1]==7:
            OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i+1,j-
1)]))

else:
    #rs.CurrentLayer("LINE 00")
    OUT_projected.append(rs.AddLine(pttest,points[(i,j)]))
    #rs.CurrentLayer("LINE 02")
    if vec[1]==0:
        OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i+1,j)]))
    if vec[1]==1:
        OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i+1,j+1)]))
    if vec[1]==2:
        OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i,j+1)]))
    if vec[1]==3:
        OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i-1,j+1)]))
    if vec[1]==4:
        OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i-1,j)]))
    if vec[1]==5:
        OUT_structure.append(rs.AddLine(pth[(i,j)], pth[(i-1,j-1)]))
    if vec[1]==6:
        OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i,j-1)]))
    if vec[1]==7:
        OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i+1,j-1)]))
    #rs.CurrentLayer("LINE 00")

    if vec[1]==0:
        OUT_projected.append(rs.AddLine(points[(i,j)],points[(i-1,j)]))
    if vec[1]==1:
        OUT_projected.append(rs.AddLine(points[(i,j)],points[(i-1,j-1)]))
    if vec[1]==2:
        OUT_projected.append(rs.AddLine(points[(i,j)],points[(i,j-1)]))
    if vec[1]==3:
        OUT_projected.append(rs.AddLine(points[(i,j)],points[(i+1,j-1)]))
    if vec[1]==4:
        OUT_projected.append(rs.AddLine(points[(i,j)],points[(i+1,j)]))
    if vec[1]==5:
        OUT_projected.append(rs.AddLine(points[(i,j)],
points[(i+1,j+1)]))
    if vec[1]==6:
        OUT_projected.append(rs.AddLine(points[(i,j)],points[(i,j+1)]))
    if vec[1]==7:
        OUT_projected.append(rs.AddLine(points[(i,j)],points[(i-1,j+1)]))

#
    rs.CurrentLayer("LINE 02")
    if vec[1]==0:
        OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i-1,j)]))
    if vec[1]==1:

```

```

        OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i-1,j-1)]))
if vec[1]==2:
    OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i,j-1)]))
if vec[1]==3:
    OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i+1,j-1)]))
if vec[1]==4:
    OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i+1,j)]))
if vec[1]==5:
    OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i+1,j+1)]))
if vec[1]==6:
    OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i,j+1)]))
if vec[1]==7:
    OUT_structure.append(rs.AddLine(pth[(i,j)],pth[(i-1,j+1)]))

#rs.CurrentLayer("LINE 02")
for j in range(y):
    OUT_structure.append(rs.AddLine(pth[(0,j)],pth[(0,j+1)]))
    OUT_structure.append(rs.AddLine(pth[(x,j)],pth[(x,j+1)]))
for i in range(x):
    OUT_structure.append(rs.AddLine(pth[(i,y)],pth[(i+1,y)]))
    OUT_structure.append(rs.AddLine(pth[(i,0)],pth[(i+1,0)]))

str2={}
maxh=[]

for i in range(x+1):
    for j in range(y+1):
        maxh.append(pth[(i,j)][2])

maxz=max(maxh)
minz=min(maxh)

pthd={}

pthds={}

if maxz==0:
    for i in range(x+1):
        for j in range(y+1):
            pthd[(i,j)]=[pth[(i,j)][0], pth[(i,j)][1], pth[(i,j)][2]]
else:
    for i in range(x+1):
        for j in range(y+1):
            pthd[(i,j)]=[pth[(i,j)][0], pth[(i,j)][1],
pth[(i,j)][2]+d*math.sqrt((1-ptd[(i,j)][2]/maxz))]

for i in range(1,x):
    for j in range(1,y):
        pthds[(i,j)]=[pthd[(i,j)],pthd[(i+1,j)], pthd[(i+1,j+1)],
pthd[(i,j+1)], pthd[(i-1,j+1)], pthd[(i-1,j)], pthd[(i-1,j-1)],
pthd[(i,j-1)],pthd[(i+1,j-1)]]

```

```

for i in range(1,x):
    for j in range(1,y):
        param=rs.CurveClosestPoint(dist[(i,j)][1][0],points[(i,j)])
        pttest=rs.EvaluateCurve(dist[(i,j)][1][0],param)
        vec=rs.VectorCreate(pttest,points[(i,j)])
        vec=rs.VectorUnitize(vec)
        pttest=rs.PointAdd(points[(i,j)],vec)
        vec=director(pts[(i,j)],pttest)
        vecl=rs.VectorLength(vec[0])
        pttest=rs.PointAdd(points[(i,j)],vec[0])
        #rs.CurrentLayer("LINE 03")
        if dist[(i,j)][0]< vecl:
            line=rs.AddLine(pttest,points[(i,j)])
            int=rs.CurveCurveIntersection(dist[(i,j)][1][0],line)
            if int:
                ptt=int[0][1]
                ptt2=[ptt[0], ptt[1], ptt[2]+100]
                linet=rs.AddLine(ptt, ptt2)
                crv2=slopizer(dist[(i,j)][1])
                int2=rs.CurveCurveIntersection(crv2[0],linet)
                rs.DeleteObject(line)
                intt=int2[0][1]
                if maxz==0:
                    intt=[int2[0][1][0],int2[0][1][1],int2[0][1][2]]
                else:
                    intt=[int2[0][1][0],int2[0][1][1],int2[0][1][2]+d*math.sqrt((1-
intt=[int2[0][1][0],int2[0][1][1],int2[0][1][2]+d*math.sqrt((1-
int2[0][1][2]/maxz))]
                #rs.CurrentLayer("LINE 01")
                cpt2=rs.CurvePoints(crv2[0])
                cptt2=[]
                for b in range(len(cpt2)):
                    if maxz==0:
                        cptt2.append([cpt2[b][0] ,cpt2[b][1], cpt2[b][2]])
                    else:
                        cptt2.append([cpt2[b][0] ,cpt2[b][1],
cpt2[b][2]+d*math.sqrt((1-cpt2[b][2]/maxz))])
                deg=rs.CurveDegree(crv2[0])
                crv3=rs.AddCurve(cptt2, deg)
                OUT_structure2.append(crv3)
                int3=rs.CurveCurveIntersection(crv3,linet)
                intt=int3[0][1]
                rs.DeleteObject(linet)
                #rs.CurrentLayer("LINE 01")
                if vec[1]==0:
                    OUT_structure2.append(rs.AddLine(pthd[(i,j)],intt))
                if vec[1]==1:
                    OUT_structure2.append(rs.AddLine(pthd[(i,j)],intt))
                if vec[1]==2:
                    OUT_structure2.append(rs.AddLine(pthd[(i,j)],intt))
                if vec[1]==3:
                    OUT_structure2.append(rs.AddLine(pthd[(i,j)],intt))
                if vec[1]==4:
                    OUT_structure2.append(rs.AddLine(pthd[(i,j)],intt))
                if vec[1]==5:
                    OUT_structure2.append(rs.AddLine(pthd[(i,j)],intt))
                if vec[1]==6:

```

```

        OUT_structure2.append(rs.AddLine(pthd[(i,j)],intt))
    if vec[1]==7:
        OUT_structure2.append(rs.AddLine(pthd[(i,j)],intt))
else:
    #rs.CurrentLayer("LINE 01")
    if vec[1]==0:
OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i+1,j)]))
        if vec[1]==1:
OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i+1,j+1)]))
        if vec[1]==2:
OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i,j+1)]))
        if vec[1]==3:
            OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i-
1,j+1)]))
        if vec[1]==4:
            OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i-
1,j)]))
        if vec[1]==5:
            OUT_structure2.append(rs.AddLine(pthd[(i,j)], pthd[(i-
1,j-1)]))
        if vec[1]==6:
            OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i,j-
1)]))
        if vec[1]==7:
            OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i+1,j-
1)]))
    else:
        #rs.CurrentLayer("LINE 01")
        if vec[1]==0:
            OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i+1,j)]))
        if vec[1]==1:
OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i+1,j+1)]))
        if vec[1]==2:
            OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i,j+1)]))
        if vec[1]==3:
            OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i-
1,j+1)]))
        if vec[1]==4:
            OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i-1,j)]))
        if vec[1]==5:
            OUT_structure2.append(rs.AddLine(pthd[(i,j)], pthd[(i-1,j-
1)]))
        if vec[1]==6:
            OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i,j-1)]))
        if vec[1]==7:
            OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i+1,j-
1)]))
    #rs.CurrentLayer("LINE 01")
    if vec[1]==0:
        OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i-1,j)]))
    if vec[1]==1:
        OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i-1,j-1)]))

```

```

if vec[1]==2:
    OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i,j-1)]))
if vec[1]==3:
    OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i+1,j-1)]))
if vec[1]==4:
    OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i+1,j)]))
if vec[1]==5:
    OUT_structure2.append(rs.AddLine(pthd[(i,j)], pthd[(i+1,j+1)]))
if vec[1]==6:
    OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i,j+1)]))
if vec[1]==7:
    OUT_structure2.append(rs.AddLine(pthd[(i,j)],pthd[(i-1,j+1)]))

for j in range(y):
    OUT_structure2.append(rs.AddLine(pthd[(0,j)],pthd[(0,j+1)]))
    OUT_structure2.append(rs.AddLine(pthd[(x,j)],pthd[(x,j+1)]))
for i in range(x):
    OUT_structure2.append(rs.AddLine(pthd[(i,y)],pthd[(i+1,y)]))
    OUT_structure2.append(rs.AddLine(pthd[(i,0)],pthd[(i+1,0)]))

#OUT_structure.append(str.values())

print "Successfull Process: Roof Completed! "

rs.EnableRedraw(True)

```