INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Université d'Ottawa · University of Ottawa

Visualizing Ordered Sets
Using Force-Directed Placement

By

Hassan Hajjdiab

A Masters Thesis
submitted to the school of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
Masters of Science in Computer Science

Abstract

The main aim of this thesis is to study and develop algorithms to visualise ordered sets and secondly to develop algorithms to visualise the faces of complicated ordered sets.

To obtain a "pleasing" drawing for an order is an important issue and deserves attention. To this end, the ordered set may be drawn with minimum edge crossing, same edge length, and it may reflect geometrical symmetry [BaEaTaTo94].

A leading problem is to visualise the faces of an ordered set with discrete structure; some ordered sets, like the *projective plane*, have a discrete structure that is difficult to visualise and understand.

In this thesis we are extending to three dimensions the concept of Eades [Ea84]. The ordered set is treated as a physical object and the vertices are considered as metal spheres with charges while the edges are considered as mechanical springs. The rules of physics are applied to the system and each vertex moves according to the net force applied to it until the system reaches its equilibrium position and the net force on each vertex is zero.

Acknowledgements

It is my pleasure to acknowledge the many people who offered their help and advice throughout the course of my work.

First of all, I would like to express my sincere gratitude to my supervisor Dr. **Ivan Rival**, for his knowledge, guidance and encouragement. It has been his kindness above all that has given me the many constructive opportunities I needed to complete this work.

I would like to thank **K.Sugiyama** and **K.Misue** for their help and valuable information that they provided during my work.

Dedication

This work is lovingly dedicated to my parents who put their whole heart into helping me accomplish this work. For their priceless support, I thank them.

Contents

List of Figures

Chapter 1

Overview

Ordered sets are widely used for representing hierarchical structures. They occur in computation, in scheduling, in sorting, in social science and even in geography.
Visualising the ordered sets plays an important role in understanding the relation between entities. To visualise an ordered set, we usually draw a graph on the plane with vertices corresponding to the elements of the ordered set, with edges drawn monotonically with respect to a fixed direction in order to represent relation between elements. This, in general, is called a *drawing* of an ordered set.
The *Force and Energy Controlled Placement* algorithm is a well known heuristic approach to draw graphs. Most of the previous work of the *Force and Energy Controlled Placement* algorithm (see [Ea84], [FrRe91], [KaKa89], [Sa96], [SuMi94] and [DaHa89]) deals with graph drawing. We will apply this algorithm for *upward drawing* of ordered sets in 3D.

Some orders have complicated structure so a traditional drawing cannot provide a visualisable structure. One complicated order is the *projective plane*. Many research efforts were devoted to study the topological and geometrical structure of the *projective plane* (see [Ha96] and [GrTu87]). The *projective plane* can be drawn with three "*handles*" as a graph and with four *"handles"* as an ordered set. In this thesis we will introduce a triangulation algorithm to triangulate the faces of the *projective plane* and then apply the *Force Directed Placement* algorithm to visualise its structure as an ordered set.

Chapter 2 entitled "Force and Energy Controlled Placement", investigates the *Force and Energy Controlled Placement* algorithms. It starts with theoretical background from physics; some laws are presented such as **Hooke's** law, **Coulomb's** law and the law of equilibrium. Also the *Force and Energy Controlled Placement* algorithms are introduced. The algorithms are the following:

- the *spring embedding* algorithm of **Eades** [Ea84] and **Kamada** and **Kawai** [KaKa89]
- the *temperature scheme* algorithms of **Fruchterman** and **Reingold** [FrRe91] and that of **Frick, Ludwig** and **Mehldau** [FLM95]
- the *simulated annealing* algorithm by **Davidson** and **Harel** [DaHa89]
- the *magnetic fields* algorithm by **Sugiyama** and **Misue** [SuMi95]

Chapter 3 , "Visualising Ordered Sets**",** provides an introduction to ordered sets, its properties, graphical data structures and upward drawing. Then we treat the following items:

- algorithm to draw ordered sets using *force directed placement* algorithm
- spherical drawing of an ordered set
- experimental results

Chapter 4 ," Triangulation", presents algorithms to visualise the faces of the ordered sets using triangulation. Experimental results are provided and the $K_{3,3}$, *cube, double cube* and the *projective plane* are triangulated.

Chapter 5 , "Implementation", discuses the tool *Order Visualiser* that we developed to get our results. First the interface of the tool is introduced then an example on using the tool is presented.

Finally, **Chapter 6 ,** "Summary and Future Work", provides a summary of the thesis and presents future work to be done.

Chapter 2

Force and Energy Controlled Placement

2.1 Introduction

Many molecules of substances in nature have a high degree of uniformity. This uniformity is due to the force and energy effects on the particles of the molecules. The particles move according to the forces applied to them and come to a position when the forces balance each other and the molecule is in minimal energy state.

The idea of the *Force and Energy Controlled Placement* heuristic is to simulate the physical model that exists in nature. The nodes are considered as particles. Starting from arbitrary position, the particles move to new positions where the energy is lower until the system comes to rest.

2.2 Theoretical background

In this section, some of the basic laws of physics are discussed. These laws are the basic theories that the *Force and Energy Controlled Placement* algorithm is based on. The first law we are going to discuss is **Hooke's** law the next one is **Coulomb's** law and the last is the law of equilibrium.

2.2.1 Hooke's law

The performance of the spring is described by a well known law called **Hooke's** law. This law can be expressed mathematically as follows [HaRe88]:

$$F = k\,\Delta l \tag{2.1}$$

where F : Is the applied tension force

Δl : Resulting compression or elongation

k : Spring constant

The spring may be in three states: the equilibrium state when $\Delta l = 0$, the elongation state when $\Delta l > 0$ and the compression state when $\Delta l < 0$.

The following figure shows the spring in its three states:

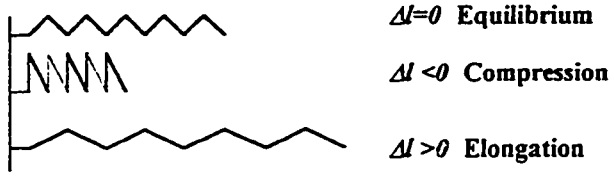


Figure 1. Spring states

The spring stores energy when it is in the compression or elongation states, no energy is stored when it is in the equilibrium state. The energy stored in the spring is given by :

$$E = \frac{1}{2} K(\Delta l)^2 \tag{2.2}$$

2.2.2 Coulomb's law

The quantitative relationship for the interaction forces between any two electric charges A and B is given by **Coulomb's** law [HaRe88]. This law may be expressed mathematically by

$$F = \frac{C q_A q_B}{d^2} \tag{2.3}$$

where

q_A the measure of charges on point A

q_B is the measure of charges on point B

C is called the proportionality constant

d is the distance between A and B

Electric field force F is , of course, a vector quantity, and the formula above gives only its magnitude. The direction of F is always along the line joining A and B, and the force is attractive if q_A and q_B have opposite signs and repulsive if they have same sign. The following figure shows the different states of particles A and B.

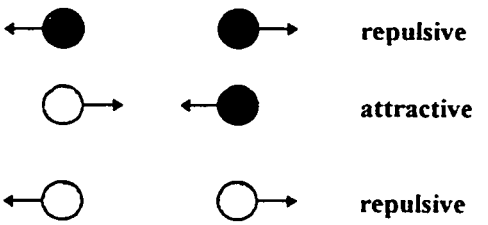


Figure 2. Charges

2.2.3 Equilibrium

An object which has no net force acting on it is said to be in translational equilibrium [HaRe88]. The condition for translational equilibrium may be expressed in the form

$$\sum \bar{F} = 0 \tag{2.4}$$

In many equilibrium situations all the various forces lie in the same plane . So, we can establish a set of x-y-z coordinate axes wherever convenient in the plane and then resolve each force F into the components Fx , Fy and Fz. Thus we can replace Eq(2.3), which is a vector equation, with three scalar equations

$$\sum Fx = 0 \tag{2.5}$$

$$\sum Fy = 0 \tag{2.6}$$

$$\sum Fz = 0 \tag{2.7}$$

In three dimensional space any system is in equilibrium if each of the total forces applied along the *x-axis* , *y-axis* and *z-axis* is zero.

2.3 Force and Energy Controlled Placement Heuristics

2.3.1 Spring Embedding

The earliest heuristics of force-directed placement is based on the work of **Eades** [Ea84] which is , in turn, evolved from an algorithm for placing components on a carrier ,such as printed circuit board or ceramic substrate. The algorithm is called *Force Directed Component Placement* [QuBr79].

Eades describes his algorithm in this way:

> "The basic idea is as follows . To embed a graph we replace the vertices by steel rings and replace each edge with a spring to form a mechanical system, ... The vertices are placed in some initial layout and let go so that the spring forces on the rings move the system to a minimal energy state. The algorithm outputs the positions of the vertices in this stable state."

Eades noticed that **Hooke's** Law springs are too strong when the vertices are far apart; he introduced logarithmic forces as follows:

$$F_a = C_1 \log\left(\frac{d}{C_2}\right) \qquad \textbf{(2.8)}$$

where d is the length of spring

C_1 and C_2 are constants.

Secondly, nonadjacent vertices repel each other by the inverse square law

$$F_r = \frac{C_3}{d^2} \qquad \textbf{(2.9)}$$

where C_3 is constant

d is distance between the vertices.

As a result, the graph is modelled as a physical system of rings and springs.
Thus **Eades** simulated the graph by a mechanical system, his algorithm is as follows:

Algorithm SPRING (*G*:graph);
1. place vertices of *G* in random locations;
2. repeat *M* times
   - calculate the force on each vertex;
   - move the vertex by C_4 times (*force on vertex*);
3. draw graph on CRT or plotter

The values $C_1 = 2.0$, $C_2 = 1.0$, $C_3 = 1.0$ and $C_4 = 0.1$ were appropriate for most graphs, he noticed also that almost all graphs achieve minimal energy state after 100 iterations , that is $M = 100$.
In his paper **Eades** [Ea84] shows some successes and limitations of the algorithm. Figure 3 shows that symmetries (graph automorphisms) are usually displayed. Figure 3(a) is the same as 3(b); the different layout is due to different initial position.
The algorithm is successful with regular grid graphs and trees. Figure 4(a) shows a grid graph of 16 vertices, while 4(b) shows a binary tree with 19 vertices.
The algorithm also produces good layout for most sparse graphs. Figure 5 shows some sparse graphs as drawn by **Eades** [Ea84]
Some problems with the algorithm may be illustrated in Figure 6. There is a triangle with vertices that are almost collinear. This problem can be solved by choosing different values of C_1, C_2 and C_3.

Kamada and **Kawai** [KaKa89] have their own variant on **Eades's** algorithm. They also modelled a graph as a system of springs, but whereas **Eades** abandoned **Hooke's** law, **Kamada** and **Kawai** solved partial differential equations based on it to optimise layout. They saw the graph drawing problem as a process of reducing the total energy of a

system of springs connecting the nodes. By reducing the total energy of the springs, the nodes would be near their ideal distances from each other. They modelled the graphs with a dynamic system in which n ($=|V|$) particles are mutually connected by springs. The position of the particles in the plane p_1, p_2, ..., p_n corresponds to vertices v_1, v_2, ..., $v_n \in V$ respectively.

The total energy of the system is given by:

$$E = \frac{1}{2}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} K_{ij}(\,|p_i - p_j| - l_{ij}^2)$$

where p_i is the position of node v_i. k_{ij} is the spring constant between p_i and p_j and l_{ij} is the ideal distance between nodes v_i and v_j.

To minimise the global energy E of the system, the following partial differential equation should be solved:

$$\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial y_i} = 0 \quad for\ 1 \le i \le n$$

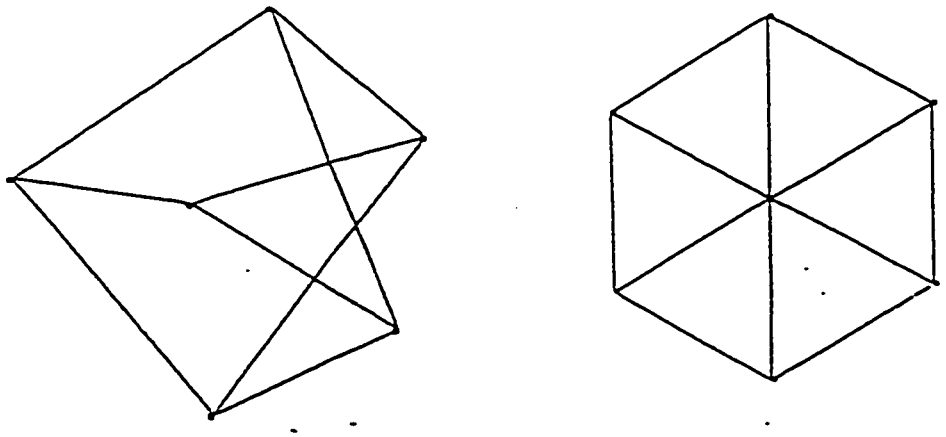
To find the position p_i for all nodes. They solved the equation using a numerical algorithm called **Newton-Raphson** [BoPr91] algorithm.

The algorithm produces good layout for symmetric graphs. Figure 7 (a) and (d) shows one picture of the regular *hexahedron* (*cube*) graph and regular *dodecahedron* graph respectively. They look as if they were the projected images of a *cube* and *dodecahedron*. Figure 7 (b) shows a graph which has the triangulated internal structure.

Also asymmetric pictures are visualised pleasingly. Figure 8 shows two pictures of asymmetric graphs. In these cases needless edge crossings are avoided.

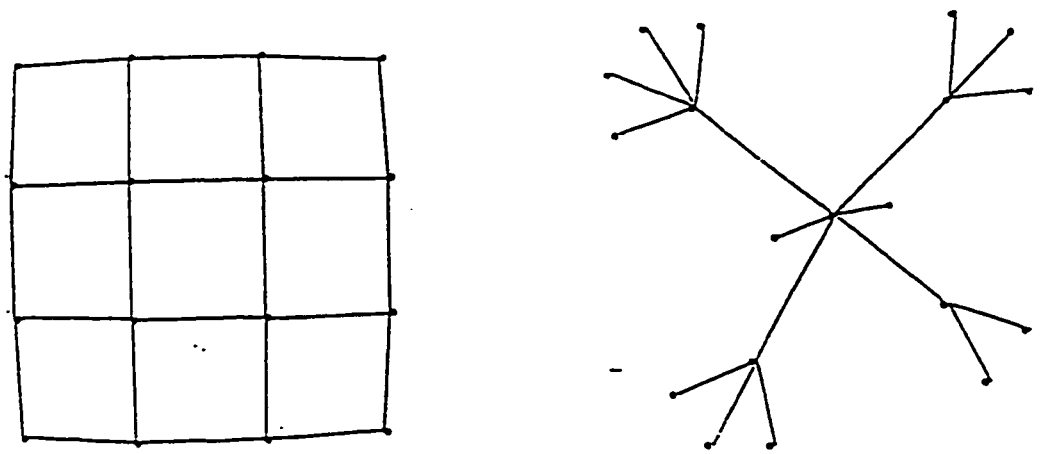
Eades decided that it was important only for a vertex to be near its immediate neighbours and so calculated attractive forces only between neighbours, but **Kamada** and **Kawai's** algorithm adds the concept of an ideal distance between vertices that are not neighbours: the ideal distance between two vertices is proportional to the length of shortest path between them [BeHo87]. However, trying to place a vertex at its ideal distance from all other vertices requires an $O(|V|^2)$ algorithm.

Both approaches have similar results and in many cases the resulting layout shows existing symmetry. One interesting example of the *Spring Embedding algorithm* is the *Grid Graph*. Figure 9 (see **Sander** [Sa96]) shows the *Grid Graph* drawn randomly. As the iterations are animated , there is the impression of three-dimensional unfolding process.



(a) (b)

Figure 3. Automorphism as drawn by **Eades**[Ea84]



(a) (b)

Figure 4. Graphs adopted from **Eades** [Ea84] (a) grid graph (b) binary tree

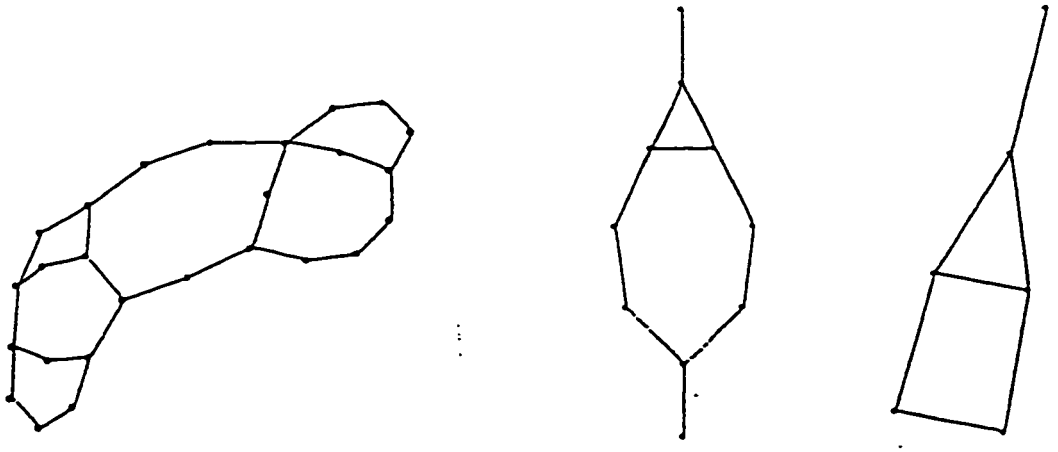


Figure 5. Sparse graphs as drawn by **Eades** [Ea84]

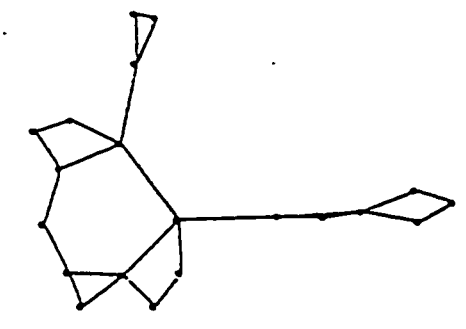


Figure 6. A triangle with vertices that are almost collinear as drawn by **Eades** [Ea84]

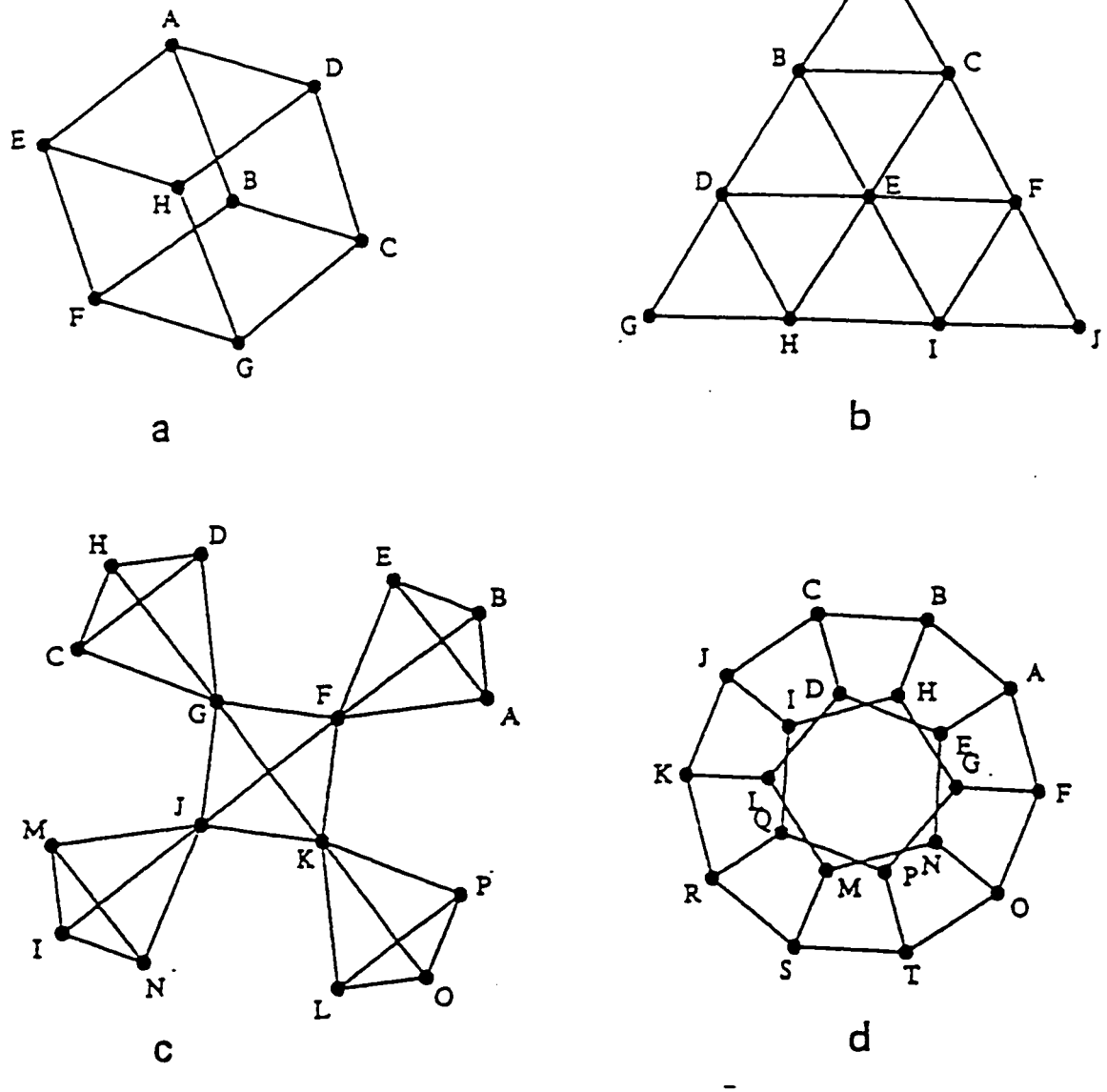


Figure 7. Symmetric graphs adopted from Kamada and Kawai [KaKa89]

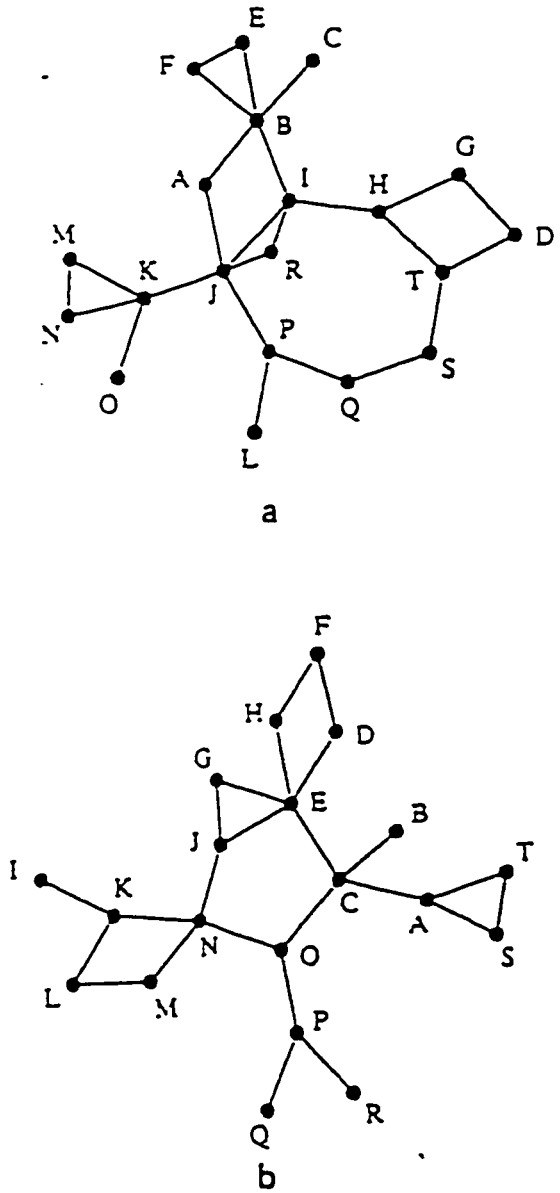


Figure 8. Asymmetric graphs as drawn by **Kamada** and **Kawai** [KaKa89]

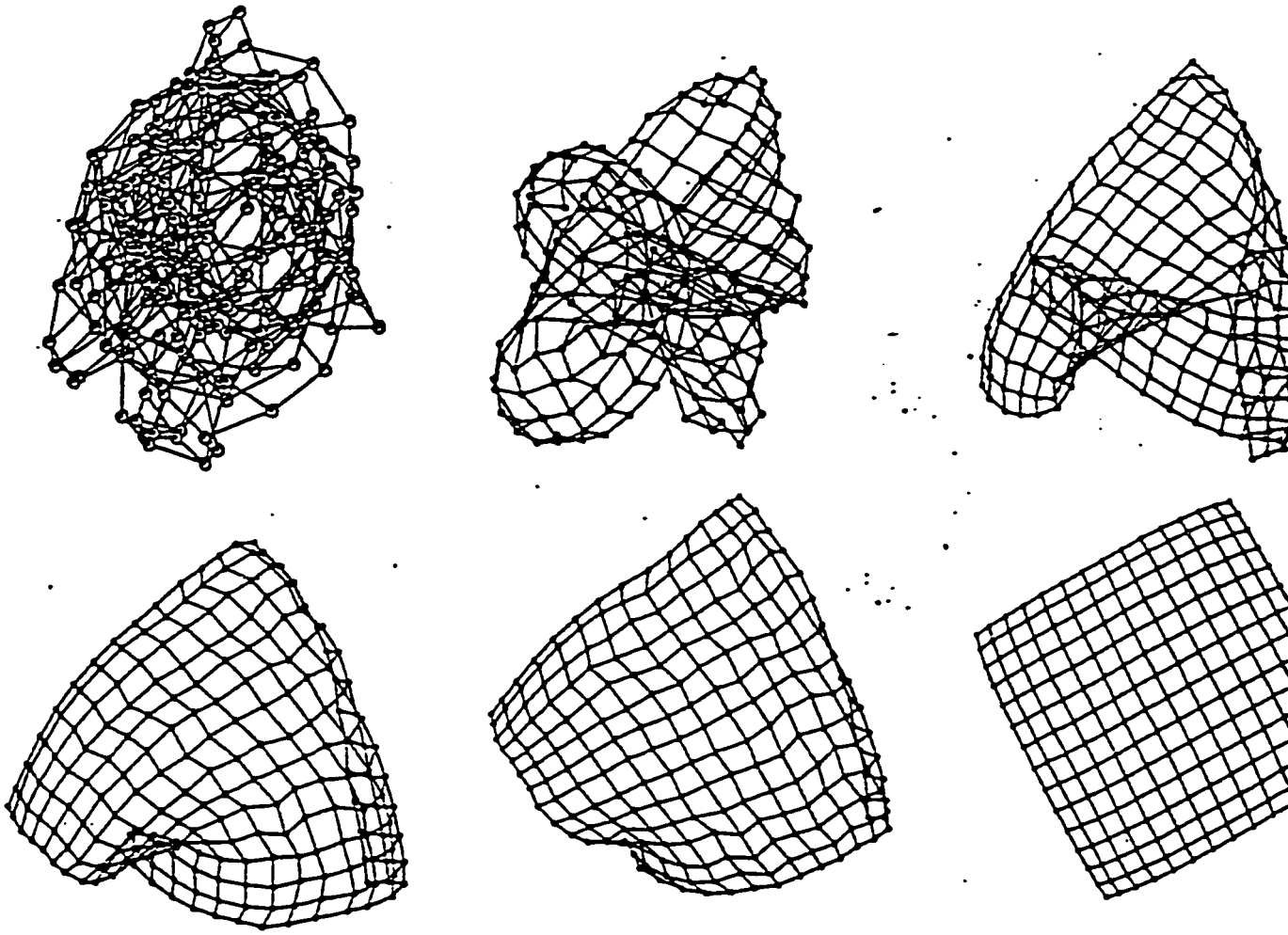


Figure 9. Animation of Spring Embedding of Grid Graph as drawn by **Sander** [Sa96]

2.3.2 Simulated Annealing

Davidson, **Harel** and **Sardas** ([DaHa89] and [HaSa93]) applied a randomised optimisation method from statistical mechanics called simulated annealing [MRRTT]. See ([Ha85],[LA87],[JAMS87]) for more surveys on the method and its use. In addition to the global energy E, there is a global temperature T which is lowered as the iterations progress. In each step, a random move is tried at some node. If the global energy E gets smaller with the new position of the node, the move is done. If E is enlarged by ΔE, the move is accepted with probability

$$P = e^{-\frac{\Delta E}{T}}$$

otherwise the move is rejected. The uphill changes of the energy prevent the layout to go towards a local minimum very early. By lowering the temperature T in each step, uphill changes get more improbable as the algorithm progresses.

The SA algorithm is applied as follows:

- the set of configurations or states of the system (initial configuration is chosen at random)
- a generation rule for new configuration , usually new configuration is obtained by choosing a random configuration from the current one.
- the cost function to be minimised over the configuration space , usually global energy E.
- the cooling schedule of the control parameter, including initial values and rules for when and how to change it (decreasing the temperature T)
- the termination condition, usually based on the time and the values of the cost function or control parameter.

The SA algorithm can be described as follows:

1- Choose an initial configuration α and initial temperature T;

2- Repeat the following (usually fixed number of times)

(a) choose a new configuration α' from the neighbourhood of α

(b) let E and E' be the values of the cost function of α and α' respectively

$$if\ E < E'\ or\ random\ < e^{\frac{(E-E')}{T}}\ \ take\ \alpha'$$

3- decrease temperature T

4- if termination condition is satisfied then stop else go to 2.

As long as the temperature decreased slowly enough, this randomised method results in uniform and symmetric layouts. The method has the advantage that no vector calculations are needed, because no force vectors need be calculated. Any complex scalar formula for the energy is allowed. Taking into account the border of the layout x_{min}, x_{max}, y_{min}, y_{max}, or the number of crossings and overlapping. Typical formulas are

$$E_{global} = \sum_{\substack{u,w \in V \\ v \neq w}} E_{rep}(v,w) + \sum_{(v,w)\in E} E_{att}(v,w) + \sum_{v \in V} E_{border}(v) + E_{lap} + E_{cross}$$

where

$$E_{rep}(v,w) = \frac{\lambda_{rep}}{|\Delta(v,w)|^2}$$

$$E_{att}(v,w) = \lambda_{att}|\Delta(v,w)|^2$$

$$E_{border} = \frac{\lambda_{border}}{(x(v)-x_{\min})^2} + \frac{\lambda_{border}}{(x(v)-x_{\max})^2} + \frac{\lambda_{border}}{(y(v)-y_{\min})^2} + \frac{\lambda_{border}}{(y(v)-y_{\max})^2}$$

$$E_{lap} = \lambda_{llap} * number\ of\ overlappings$$

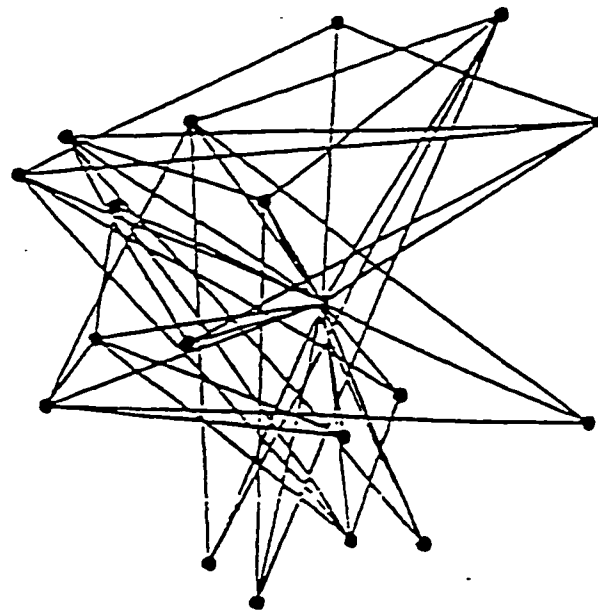
$$E_{cross} = \lambda_{cross} * number\ of\ crossings$$

where λ_{rep}, λ_{att}, λ_{border}, λ_{llap} and λ_{cross} are parameters for tuning the model

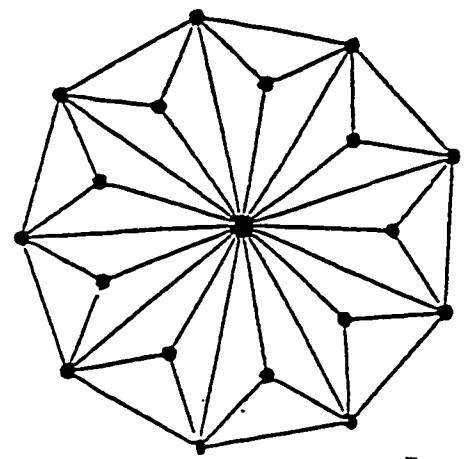
The algorithm produces good layouts for planar graphs. Figure 10(a) shows planar graph in initial random position , 10(b) shows the final layout.

Another example of a planar graph is Figure 11. Figure 11(a) shows the input with random locations of vertices. Figure 11(i) shows the final output. Figures 11(b) through 11(h) are seven intermediate steps of the graph.

Figure 12 shows a drawing of a graph containing 37 nodes and 68 edges. The graph is nonplanar, but nevertheless the algorithm deals with it well. The algorithm does not always do as well on other graphs. For example, Figure 13 is a complete binary tree of depth six. It contains 62 edges. The drawing contains two crossings, and the structure of the tree is rather difficult to visualise.



(a)



(b)

Figure 10. Planar graph as drawn by **Davidson and Harel** [DaHa91]

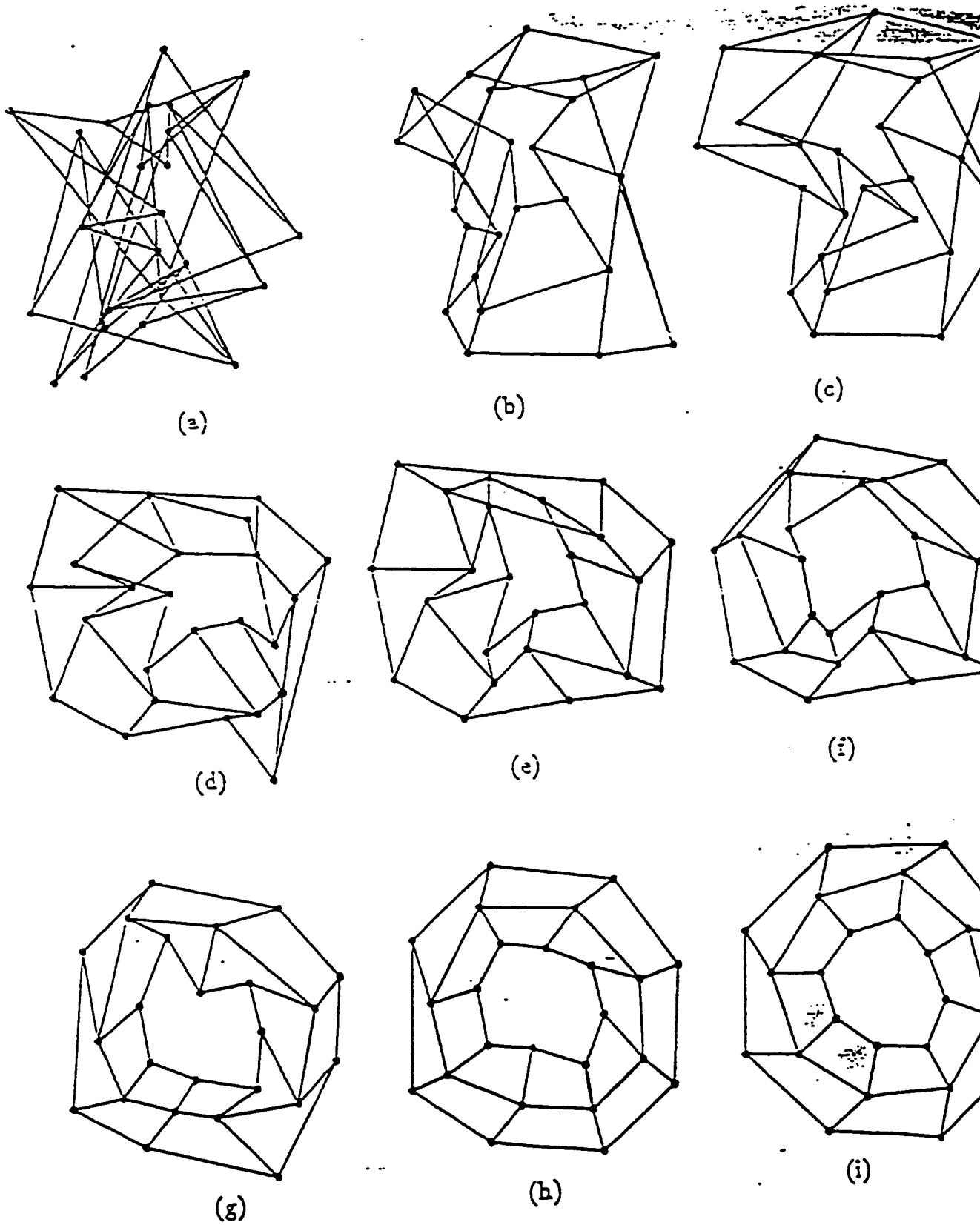


Figure 11. Another planar graph drawn by **Davidson and Harel** [DaHa91]

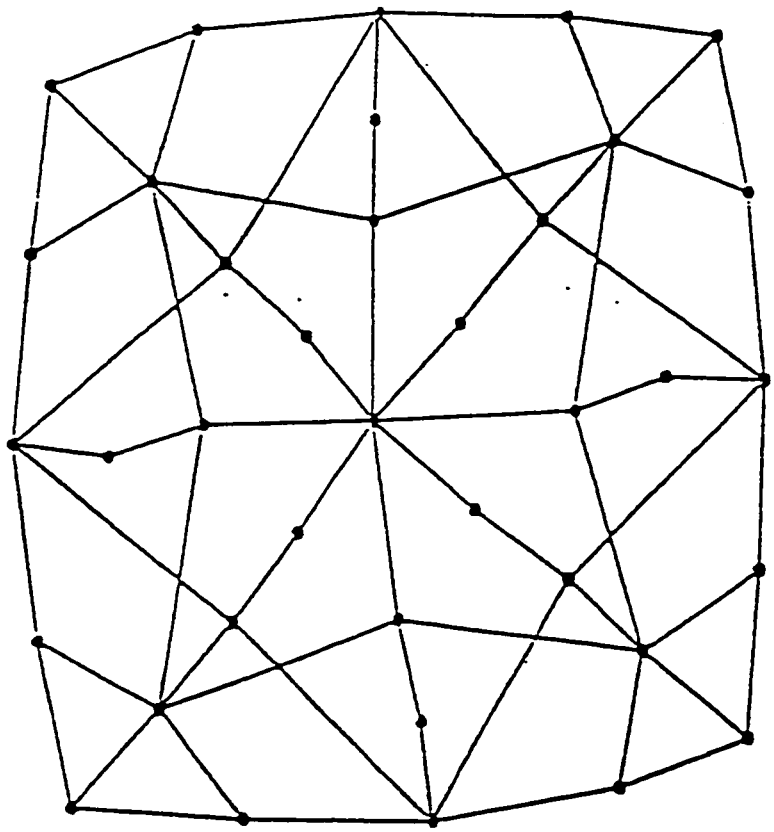


Figure 12. Nonplanar graph as drawn by Davidson and Harel [DaHa89]

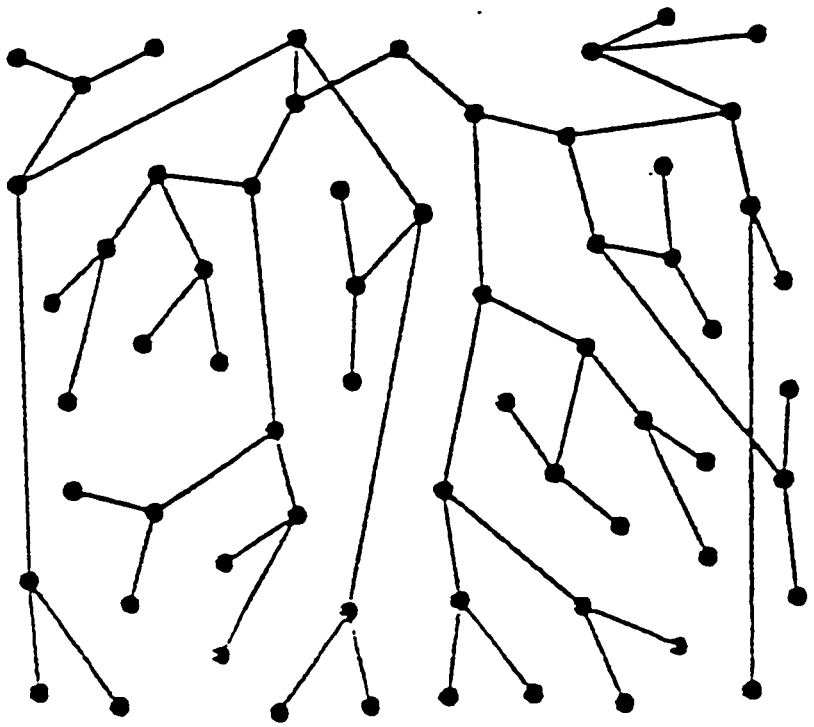


Figure 13. Complete binary tree of depth six as drawn by Davidson and Harel [DaHa89]

2.3.3 Temperature Schemes

Fruchterman and **Reingold** [FrRe91] used the concept of cooling temperature to the spring embedding. Vertices are moved in the direction of resultant force by $\delta(T)$, but the value of $\delta(T)$ is controlled by the global temperature T. As T decreases $\delta(T)$ decreases, when $T = 0$ and $\delta(T) = 0$ and all vertices are at rest.
The attractive and repulsive forces that they used are different than those chosen by Eades, they are as follows:

$$F_a = \frac{d^2}{K}$$

$$F_r = -\frac{K^2}{d}$$

where K is the optimal distance between vertices

d is the distance between vertices

The most natural and pleasing results of the algorithm may be those from highly symmetrical graphs. Figure 15 shows K_4, K_5 and K_6 graphs. Note that K_4 is planar although this drawing is not planar.

Figure 16(a) shows the *cube* and 16(b) shows the *hypercube*. The four-dimensional layout of the *hypercube* is not visualisable.

Frick, Ludwig and **Mehldau** [FLM95] expanded the concept of **Fruchterman** and **Reingold** by introducing local temperature $T(v)$ for each node. Each node v is moved according to the value $\delta(T(v))$. A global temperature is introduced as:

$$T_{global} = \frac{1}{n}\sum_{i=1}^{n} T(v_i)$$

The algorithm terminates when T_{global} is less than a threshold temperature $T_{threshold}$. The temperature of each node $T(v)$ is assigned according to the movement of node v. The new impulse vector, $I_{new}(v)$ is compared with the old impulse vector $I_{old}(v)$ (Figure 14). If both vectors point into the same direction then the local temperature T(v) is increased since the ideal position is more likely to be found in this direction. If both point into opposite directions, T(v) is decreased since node v is more likely to oscillate around its ideal position. If a node oscillates several times in the same direction, it is more likely oscillating around its ideal point and T(v) is decreased.

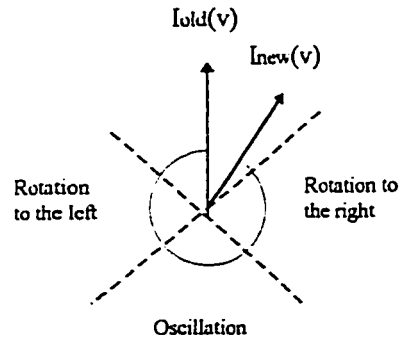


Figure 14. Detection of Rotation and Oscillation

The quality of output is similar to that of the spring embedding . Figure 17 shows the unfolding of a triangular mesh which is similar to that in Figure 9, the unfolding of a *Grid Graph.* Figure 18 shows a sparse graph (the cycle graph) in development stages. The algorithm also produces good layout for binary trees. In Figure 19 binary trees of size $|V| = 31, 63, 127$ and 255 are shown.

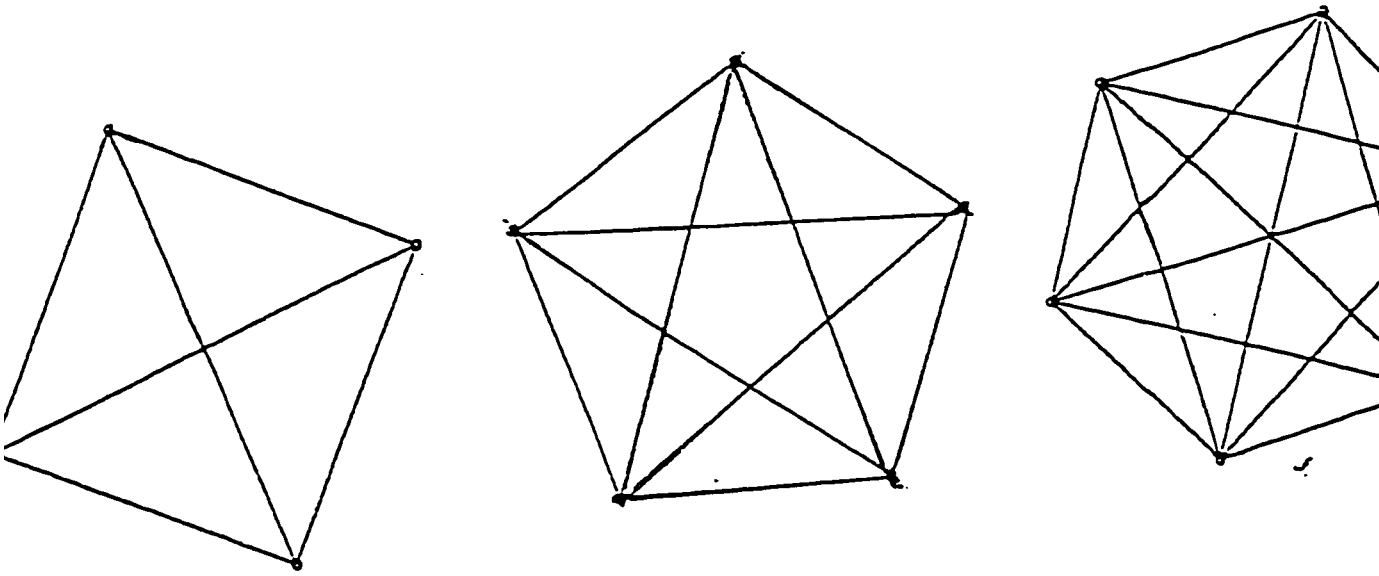


Figure 15. Highly symmetrical graphs as drawn by **Fruchterman** and **Reingold** [FrRe91]

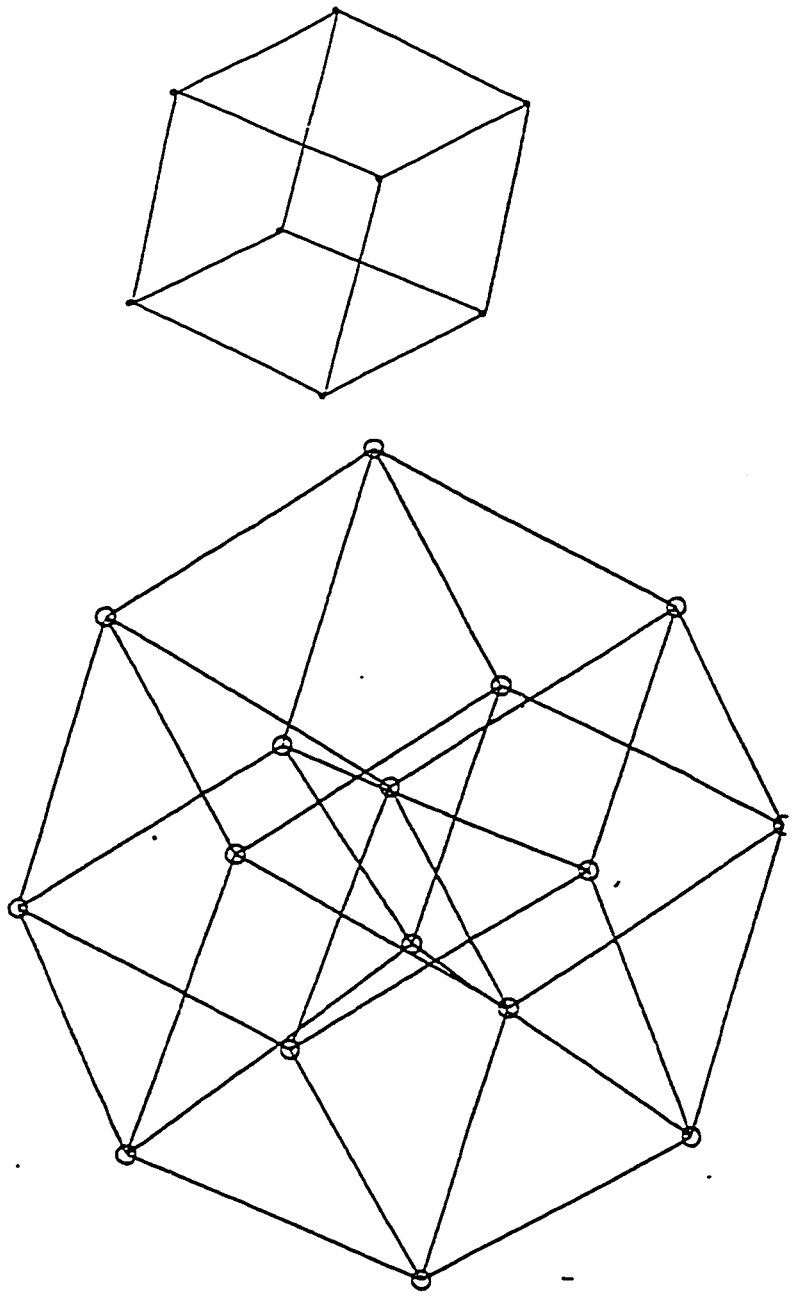


Figure 16. *Cube* and *hypercube* as drawn by **Fruchterman and Reingold** [FrRe91]

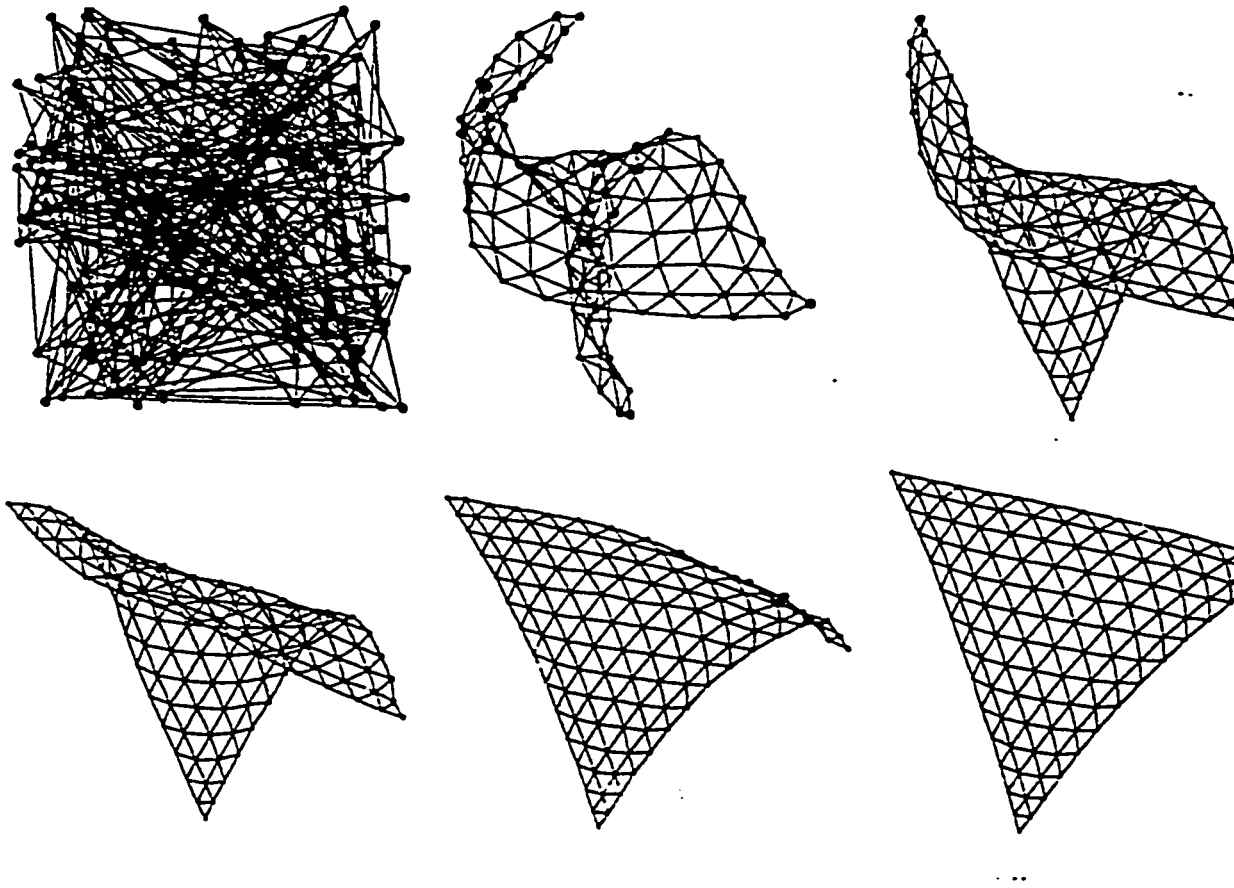


Figure 17: Unfolding of a triangular mesh as drawn by **Frick** [FLM95]

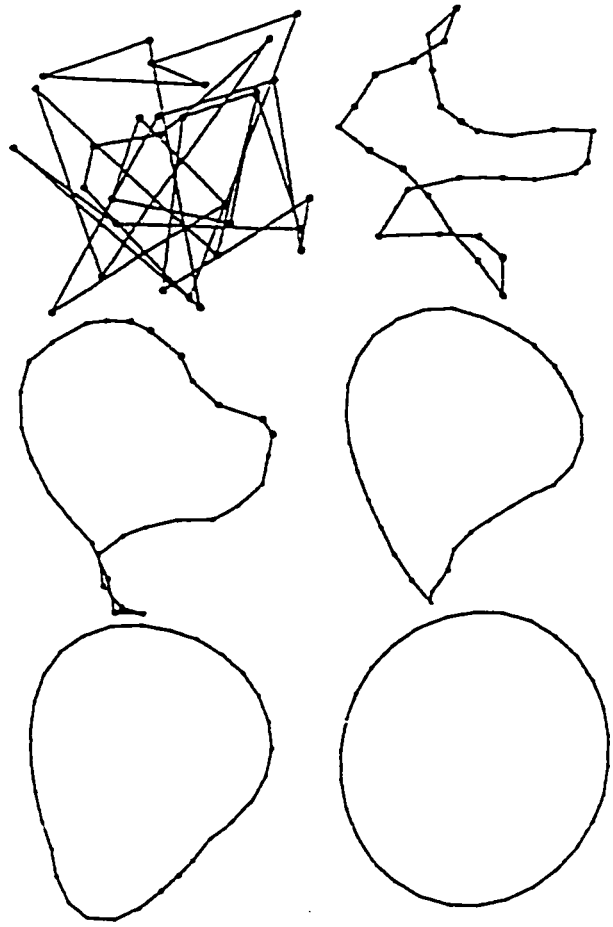


Figure 18. The cycle graph as drawn by Frick [FLM95]

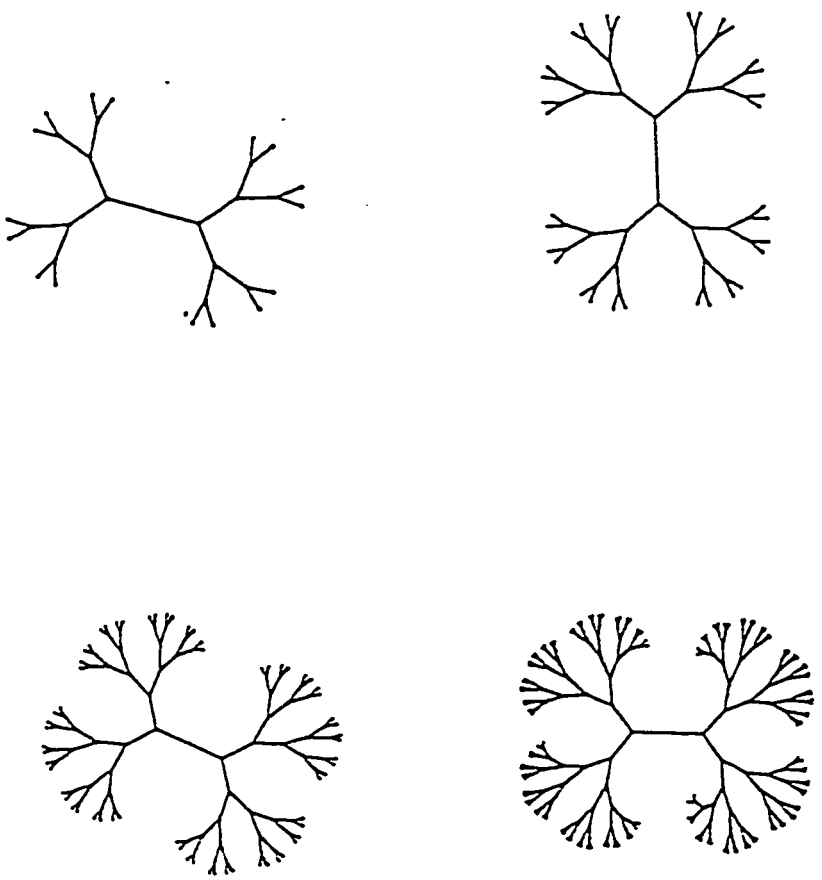


Figure 19. Binary trees of size $|V|=31, 63, 127$ and 255 as drawn by Frick [FLM95]

2.3.4 Magnetic Fields

Recently, **Misue** and **Sugiyama** [SuMi94,SuMi95] proposed an extension according to which the spring embedder algorithm takes edge direction into account . They considered the edges as springs, but also as magnetic needles which are oriented according to a magnetic field. Additional force called the magnetic force is added. While the spring force is proportional to the length of the spring, the magnetic force is proportional to the angle α between the edge and magnetic field, and is directed orthogonally to the edge. The edge is rotated according to the force. The magnetic force can be described by:

$$F_m = c_m b d^{\alpha} \theta^{\beta}$$

where d is the distance between pair of vertices, b is the strength of a magnetic field, θ ($-\pi < \theta < \pi$) is the angle in radians from the orientation of the field to the orientation of the magnetic edge, α, β and c_m are parameters for tuning the model.

The magnetic force becomes zero when the edge points exactly in the direction of the field (Figure20).

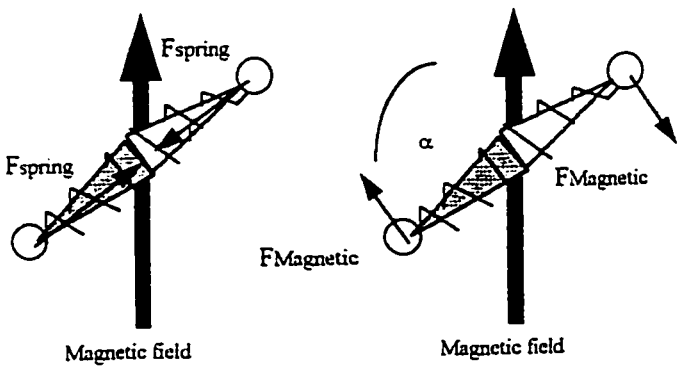


Figure 20. Spring Force and Magnetic Force

They considered three types of standard fields: parallel, polar and concentric; and two types of compound fields: orthogonal and polar-concentric (Figure 21). The compound magnetic fields are composed from standard magnetic fields.

Each standard magnetic field $B(x,y)$ at (x,y) is given by:

$B(x,y) = b\, m(x,y)$

where b is the strength of the magnetic field (constant for uniform fields)

$m(x,y)$ is the orientation of the magnetic field at any point (x,y)

the value of $m(x,y)$ for different fields is as follows:

- parallel field

$$m(x,y) = \begin{cases} (0,1) & : North \\ (-1,0) & : West \\ (0,-1) & : South \\ (1,0) & : East \end{cases}$$

- polar field

$$m(x,y) = \frac{(x,y)}{|(x,y)|}$$

- concentric field

$$m(x,y) = \begin{cases} \dfrac{(y,-x)}{|(x,y)|} : & clockwise \\ \dfrac{(-y,x)}{|(x,y)|} : & anticlockwise \end{cases}$$

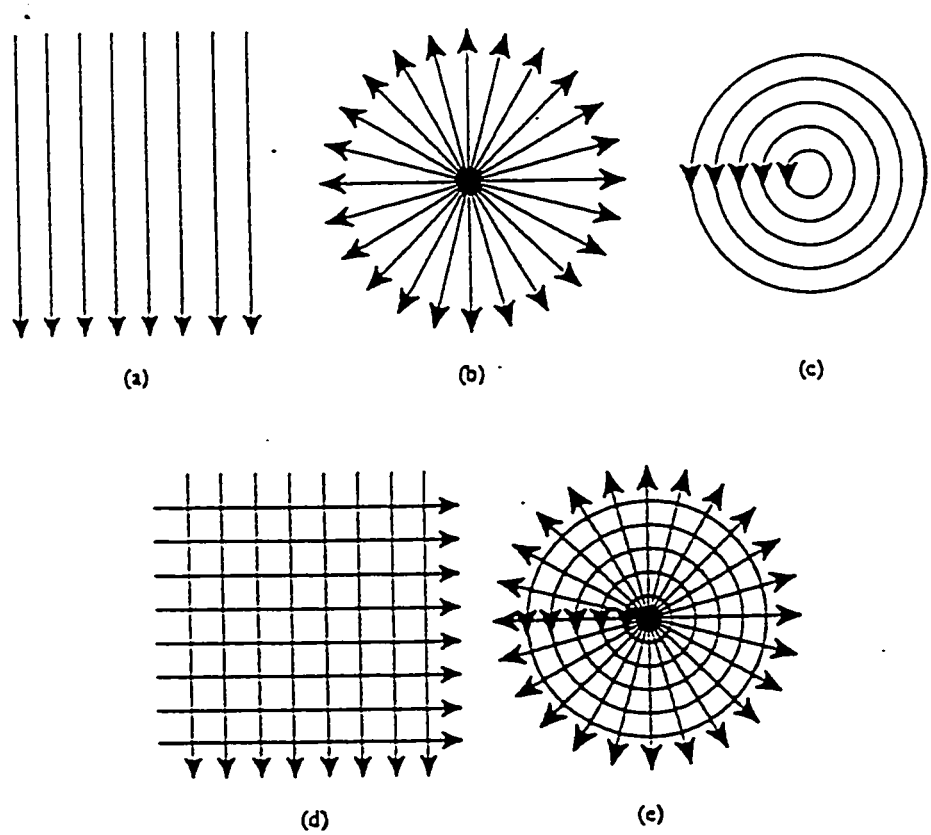


Figure 21. Magnetic fields (a) parallel (b) polar (c) concentric (d) orthogonal (e) polar-concentric

The output of the algorithm changes as the strength and the direction of the magnetic field change. Figure 22 shows the layout of a directed acyclic graph in parallel field with different intensity *b*. Figure 23 shows an edge-bipartite rooted tree in the orthogonal field. Figure 24 shows cyclic directed graph in the concentric field.

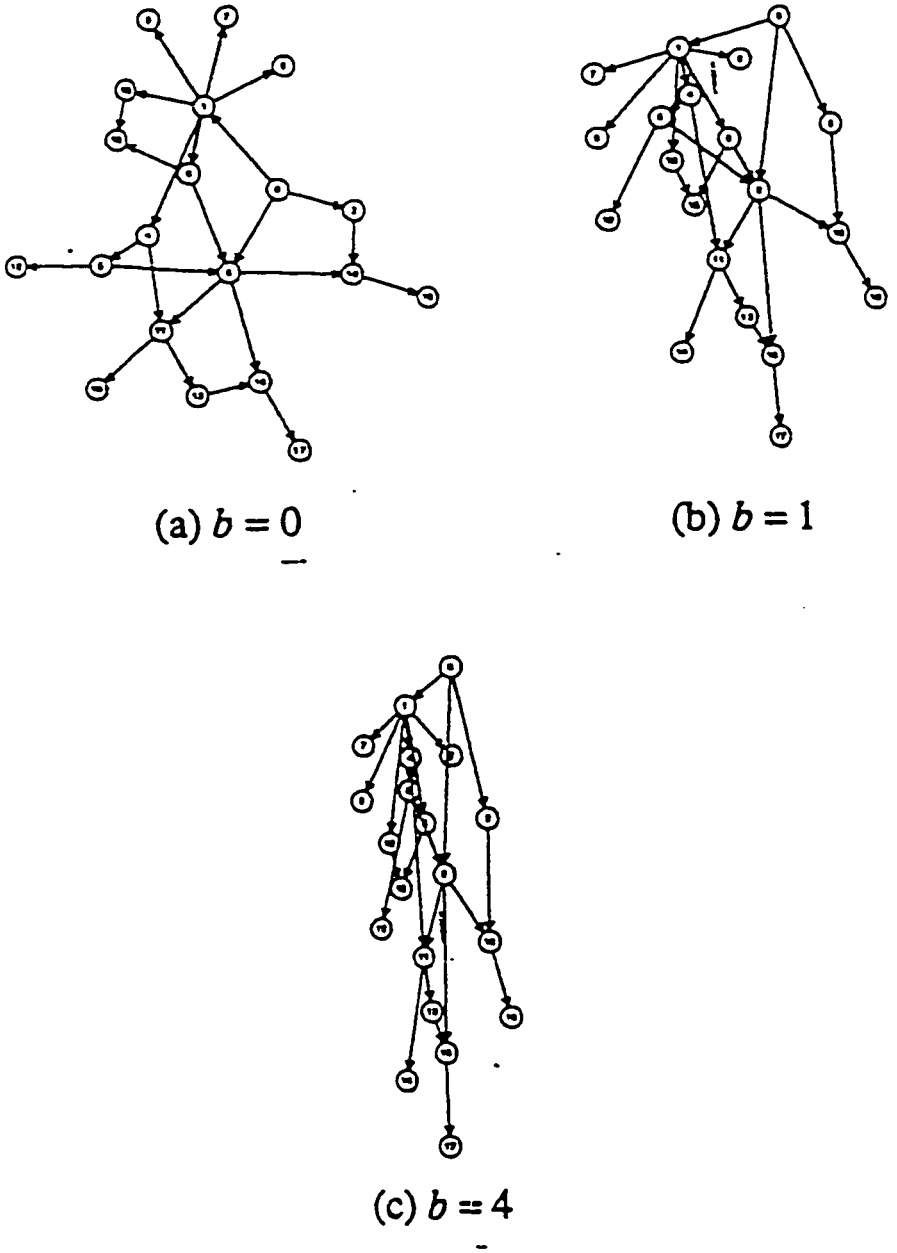


Figure 22. Layouts of an acyclic directed graph in the parallel field from Sugiyama and Misue [SuMi95]

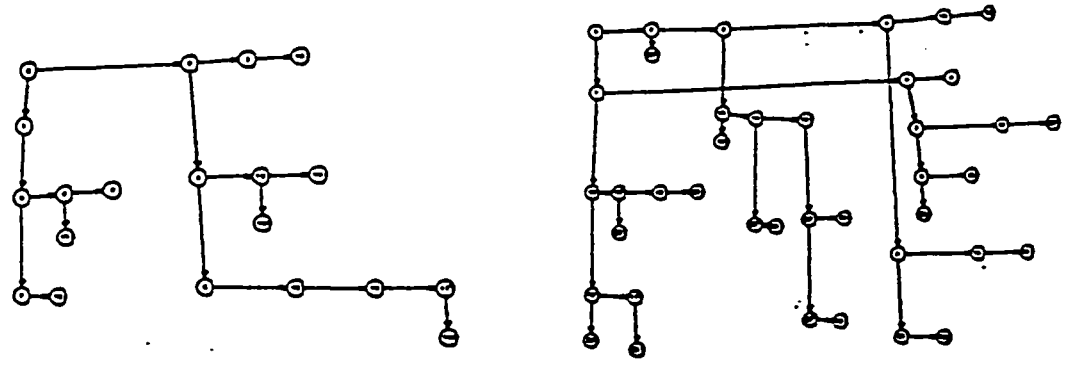


Figure 23. Layouts of an edge-bipartite rooted tree in orthogonal field from **Sugiyama and Misue** [SuMi95]

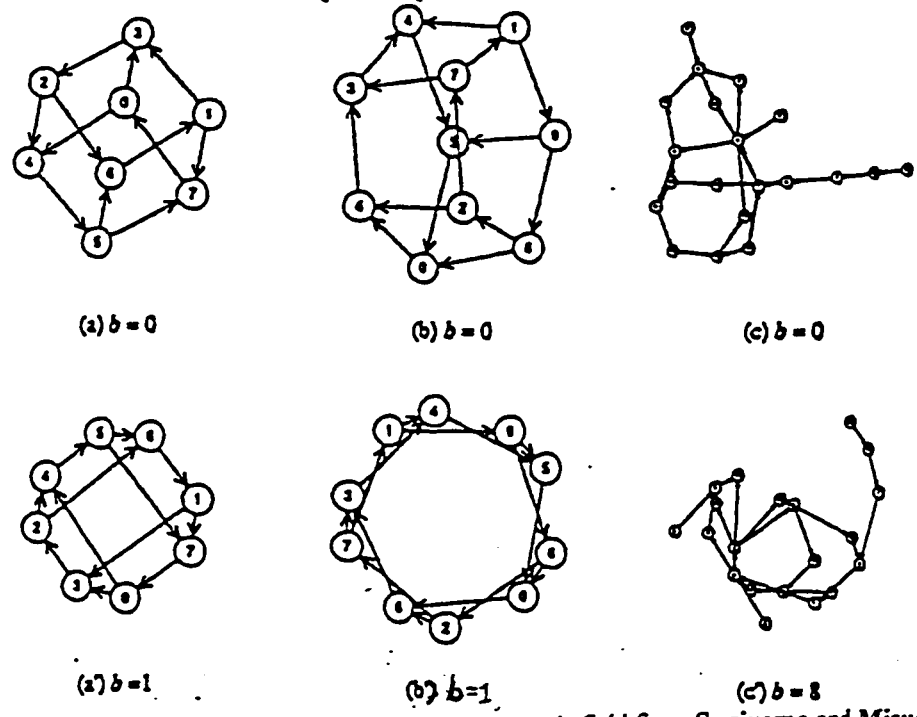


Figure 24. Layouts of three cyclic directed graphs in the concentric field from **Sugiyama and Misue** [SuMi95]

Chapter 3

Visualising Ordered Sets

3.1 Graphs

A graph G is a pair of sets (V,E) where V is nonempty, and E is a (possibly empty) set of unordered pairs of elements of V. The elements of V are called the vertices of G and the elements of E are called the edges of G. A graph may be finite or infinite depending whether the set V is finite or infinite. In our work we will consider finite graphs.

In a graph it is not allowed that two vertices are joined by more than one edge. If we allow such multiple edges , the structure is called a multigraph. In our work only graphs are considered.

If x and y are vertices of a graph G, we say x is adjacent to y if there is an edge between x and y. We say also x and y are neighbours. We say that a vertex x is incident with an edge e if x is an endpoint of e. We also say that e is incident with x whenever x is an endpoint of e.

There are several aesthetics for obtaining pleasing drawings of general undirected graphs [BaEaTaTo 94]. The main such aesthetics are:

- display symmetry
- avoid edge crossings
- avoid bends in edges
- keep edge lengths uniform
- distribute vertices uniformly

3.2 Ordered Sets

Ordered sets are special kind of graphs with two abstract properties. The first is *antisymmetry* or, loosely speaking, "inequality" [Ri96,Ri89]. Mathematically *antisymmetry* means that,

if a is bigger than b, then b is not bigger than a.

For example *4* is more than *2* and *2* is not more than *4*, that is *antisymmetry*.

The second abstract property of order is *transitivity*,

if a is smaller than b and b is smaller than c, then a is smaller than c.

The natural number *1* is less than *3* and *3* is less than *4*, it follows by transitivity *1* is less than *4*.

There are three graphical data structures for ordered sets: *comparability*, *covering* and *diagram*[Ri89]. The *comparability graph* is an undirected graph in which an edge joins two vertices a and b precisely if either $a < b$ or $b < a$. Figure 25 shows the comparability graph of 2^3, the ordered set of all subsets of $\{a,b,c\}$ ordered by inclusion.

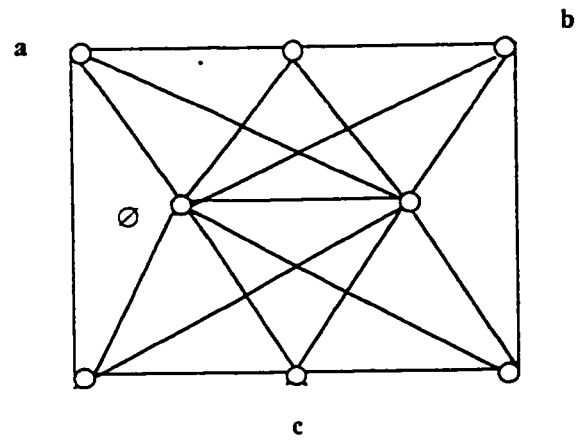


Figure 25. Comparability graph

The profusion of edges in the comparability graph may be avoided by exploiting the 'transivity' of an order. For elements a and b in an ordered set P we say that a covers b or b covered by a, if $a > b$ and if, for each x in P, $a > x \geq b$ implies $x=b$. We also call a an *upper cover* of b, and b a *lower cover* of a. The *covering graph* of P is an undirected graph whose vertices are elements of P and in which an edge joins two vertices a and b precisely if a covers b or b covers a (Figure 26).

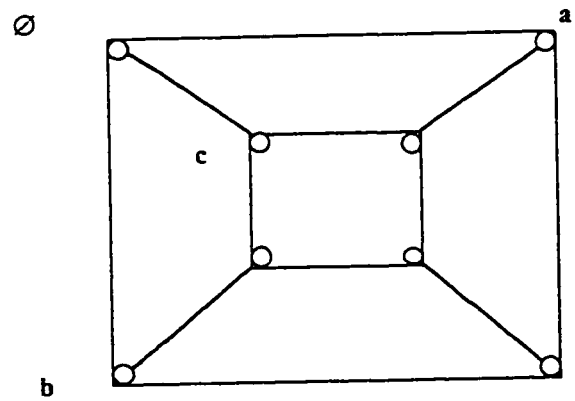


Figure 26. covering graph

Antisymmetry of the order relation makes possible an orientation of the covering graph from which the comparability relations may be readily inferred. To this end we orient any edge $a > b$ of the covering graph so that it makes an angle θ with the horizontal line satisfying $0° < \theta < 180°$. This is a diagram of P. Thus, the elements of P are represented by small circles on the plane so arranged that any circle corresponding to an upper cover a of b is situated higher in the plane than the circle corresponding to b and is joined to it by monotonic arc (that is, an arc with no repeated y-coordinates) (Figure 27).

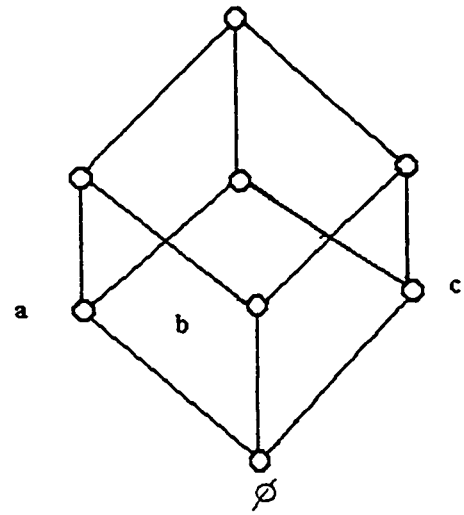


Figure 27. Digraph

3.2.1 Upward Drawing

It is customary to identify an ordered set with its geometrical representation, called its upward drawing. According to this convention, the elements of the ordered set P are drawn on a surface, traditionally a plane, as disjoint small circles, arranged in such a way that for a, b belonging to P, the circle corresponding to a is higher than the circle corresponding to b whenever $a > b$ and an arc, monotonic with respect to a fixed direction, usually south to north, is drawn to join them only if a covers b. These arcs are drawn, of course, to avoid the incidence of any other circle on it, to avoid unwanted comparabilities and when possible, to avoid intersection, except where two arcs meet at a circle. Figure 28(i) shows an upward drawing where $a > b$ and a covers b, figure 28(ii) shows an upward drawing where $d > c$ but d does not cover c.

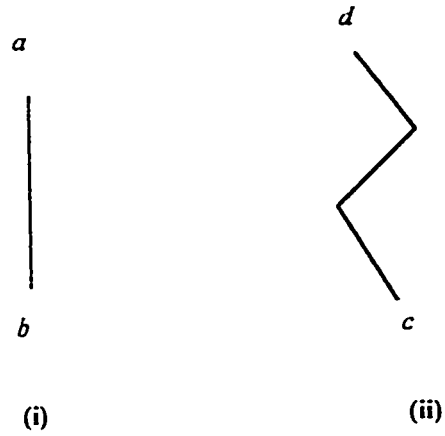


Figure 28. Upward drawing of ordered sets, In(i) a covers b, in (ii) d > c but d does not cover c

An ordered set is planar if it has an upward drawing on the plane with no arc crossing, although they may meet at a vertex with which they are incident. Usually, by a planar upward drawing of an ordered set, we mean a drawing in which all edges are straight lines. It is a fundamental fact that every planar ordered set has a straight line planar representation (see Figure 29).

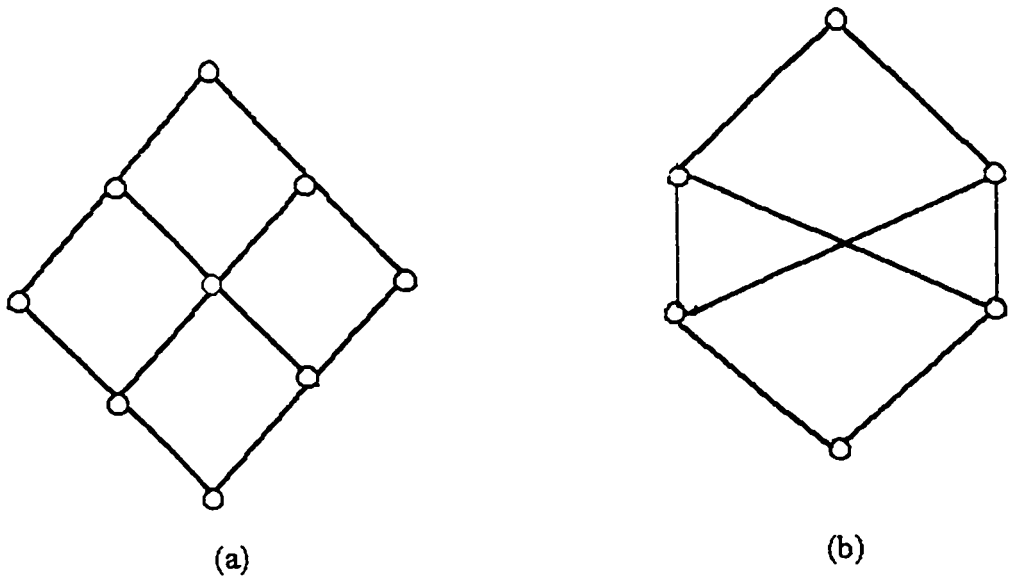


Figure 29. (a) A planar ordered set (b) A nonplanar ordered set

An ordered set is spherical if it has an upward drawing on the sphere

$S = \{ (x,y,z) : x^2 + y^2 - z^2 = 1 \}$

such that all edges are monotonic paths with respect to a fixed direction, say the positive direction of the z-axis, that is northerly direction, and no two edges cross. The ordered set in Figure 29(b) is a spherical order and it can be drawn on a sphere as shown in Figure 30.

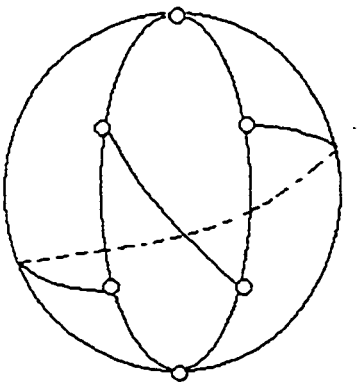


Figure 30. Spherical order

3.3 Drawing on a Sphere

In this algorithm, we will modify the *Force-directed placement* algorithm to draw ordered sets and graphs on a sphere. The decision problem whether an ordered set has an upward drawing on a sphere is an NP-complete problem [HaKiRi96]. Our algorithm is a heuristic approach that tries to draw an ordered set on a sphere with minimal edge crossing.

The idea of the algorithm is that we introduced a sphere with center at (0,0,0) and with radius R_s. When the *Force-directed placement* algorithm is applied, the vertices will repel each other but they will be attracted to the sphere with certain attractive force f_s. After a certain number of iterations all the vertices will eventually lie on the surface of the sphere and hence we get a spherical drawing of the ordered set. The concept of our sphere is that the vertex can penetrate the sphere from outside to inside or from inside to outside depending on the direction and magnitude of the resultant forces. The radius of the sphere should be chosen in a way that all vertices are able to stick to the surface of the sphere. We define the force of the sphere to be

$$f_s = k_s \log(\frac{d}{R_s})$$

where k_s is sphere constant, d is the distance of the vertex from the center of the sphere and R_s is the radius of the sphere.

The direction of f_s is in the same direction as the directed line from the center of the sphere to the vertex v. The value $|(d-R_s)|$ represent the distance of the vertex from the surface of the sphere. As $|(d-R_s)|$ decreases f_s decreases and the system stabilises more.

$$f_s \begin{cases} > 0 \ \ if\ d > R_s & \textbf{vertex outside the sphere} \\ = 0 \ \ if\ d = R_s & \textbf{vertex on the surface of the sphere} \\ < 0 \ \ if\ d < R_s & \textbf{vertex inside the sphere} \end{cases}$$

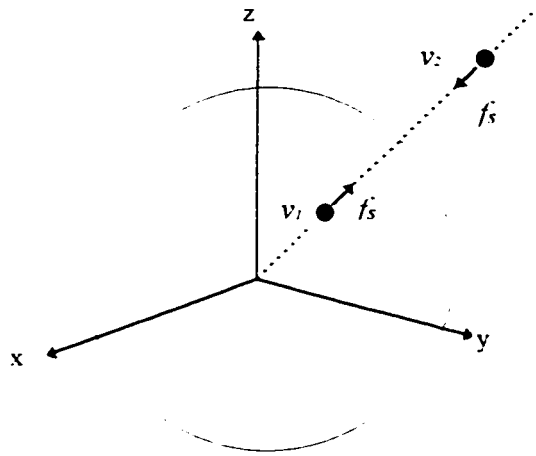


Figure 31. Spherical Drawing

In Figure 31, the force f_s on vertex v_1 is toward the outside of the sphere, while the sense of f_s on vertex v_2 is toward the inside.

3.4 The Frame

The order should be confined to the frame specified by the user. **Fruchterman** and **Reingold** [FrRe90] placed dummy vertices around the perimeter of the graph that exerted repulsive forces and these dummy vertices could not move. They consider the frame as an immovable object, modelling it as four 'walls', each of which exerts a normal force exactly equal to the force pushing any vertex beyond it, thus stopping it like a real wall. They introduced many approaches to model the frame as physical walls. The first approach is the 'sticky' vertex that adheres to the spot on the wall where it first strikes

and this is called inelastic collision. The vertex sticks to the wall until the force vector applied on it has only components to move it away from the wall (see Figure 32(a)).
Another approach is the elastic collision Figure 32(b)). In this approach the order and the orientation in which the vertex hits the wall are important. This may add extensive computation since the vertex may bounce many times. This approach is computation-intensive method and it uses the same concept of ray-tracing [HeBa86] used in computer graphics.

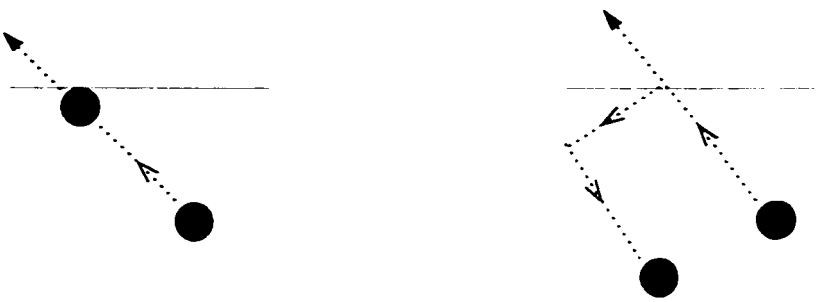


Figure 32. (a) Inelastic collision (b) Elastic collision

Our method is similar but since we are working in three dimensions, we chose to confine our graph inside a sphere. We call our frame the *spherical frame* (Figure 33). The center of the sphere is at (0,0,0) and the radius can be controlled . Originally, all the graph is drawn inside the sphere, when a vertex tries to cross the borders the applied forces considered are only the centrifugal and tangential forces and the distance from the origin does not exceed the radius of the sphere. In this way no vertex can cross the borders of the sphere.

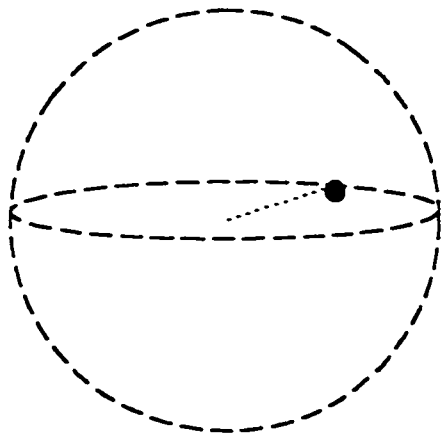


Figure 33. Spherical frame

3.5 Centralising the Order

The whole order is drawn at the center of the screen. We used a function called *centralise* to draw the graph at the center of the screen. The idea is to pass through the coordinates of our vertices one time and find the highest and lowest x,y and z coordinates, then we consider that the central point of our drawing is located at the point $O'(x_1, y_1, z_1)$.

where $x_1 = \frac{x_{max} - x_{min}}{2}$, $y_1 = \frac{y_{max} - y_{min}}{2}$ and $z_1 = \frac{z_{max} - z_{min}}{2}$

z_{max}, y_{max}, x_{max} are the maximum ordinate on the z-axis, y-axis and x-axis respectively.

z_{min}, y_{min}, x_{min} are the minimum ordinate on the z-axis, y-axis and x-axis respectively

To draw our graph at the center of the screen we simply translate the center from $O'(x_1, y_1, z_1)$ back to $O(0,0,0)$.

The algorithm is as follows:

step1: Find z_{max}, y_{max}, x_{max}

z_{min}, y_{min}, x_{min}

step2: Find x_1, y_1, z_1

step3 : $\forall\ i \in \mathbf{G}$ do begin

$x_{new(i)} = x_{old(i)} - x_1$

$y_{new(i)} = y_{old(i)} - y_1$

$z_{new(i)} = z_{old(i)} - z_1$

End

3.6 The Algorithm

The algorithm , in more detail, for drawing an ordered set in three dimensions, is as follows :

Repeat

For each vertex v_i in **G**

1- Find the x, y, and z components of the total resultant force

2- move vertex v_i in the x-axis and y-axis by:

$$xv_i = xv_i + \alpha\ Ftx_i$$

$$yv_i = yv_i + \alpha\ Fty_i$$

3- $\{\forall\ v_j \in \mathbf{G} / v_j$ covers $v_i\}$ if $zv_j > zv_i + \alpha\ Ftz_i$ then move v_i by

$$zv_i = zv_i + \alpha\ Ftz_i$$

Until (Resultant force on each vertex < $\mathbf{F}_{\mathbf{threshold}}$)

$\mathbf{F}_{\mathbf{threshold}}$ and α are two constants used throughout the algorithm. In the implementation their values are 1 and 0.1 respectively.

3.7 Experimental Results

In this section we provide some experimental examples of some orders. Most of the results that we get are pleasing (see section 3.1) and symmetrical.

The forces considered are of two types:

(1) F_a: attractive or repulsive forces caused by the springs between neighbours

(2) F_r : repulsive forces between every pair of non-neighbouring vertices

The value of these forces is given by:

$$F_a = C_s \log(\frac{d}{k})$$

$$F_r = \frac{C_r}{d^2}$$

Where d is the distance between a pair of vertices

k is the "ideal" distance between neighbours

C_s is spring constant

C_r is the repulsion constant

The default values of the parameters are set as $C_s = 1.0$, $C_r = 1.0$ and $k = 1.0$.

We experimented with many orders, some of which appear to be highly symmetrical like the *Double cube*, *Grid* , and the *Cube* . The time needed by the tool to produce the results ranges from 0.32 to 4 seconds. The language used to implement the algorithm is Visual

Basic 3.1 (see **Chapter 5**). The machine used was a Pentium P60Mhz CPU speed and 8 MB memory.

The following table shows the number of iterations, number of vertices, number of edges and the time in seconds to produce each order.

ORDER	NUMBER OF ITERATIONS	NUMBER OF VERTICES	NUMBER OF EDGES	TIME IN SECONDS
Cube	50	8	12	0.73
Double Cube	70	14	23	3.08
Grid 16	70	16	24	3.8
Spider	50	15	19	2.5
$K_{3,3}$	50	6	9	0.32
$K_{3,3}$ with top and bottom	50	8	15	0.75
Grid 9	60	9	12	1.1
Non-planar order	50	6	8	0.4
Projective Plane	50	16	35	3.8

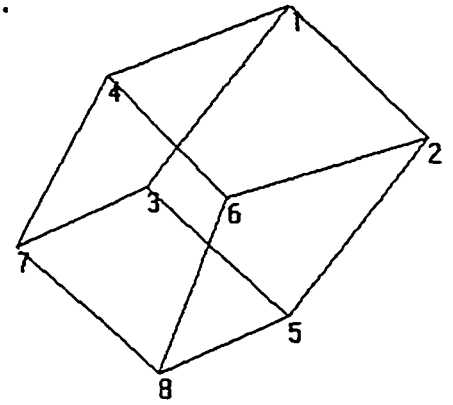


Figure 34. cube

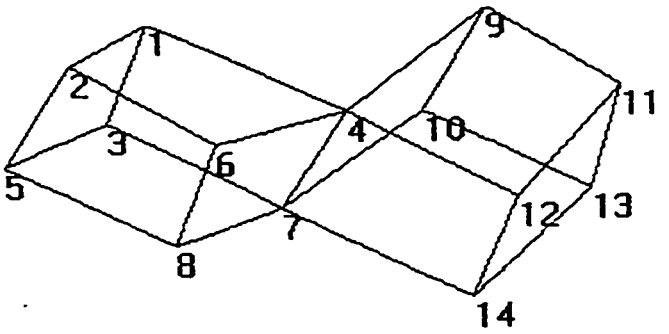


Figure 35. Double cube

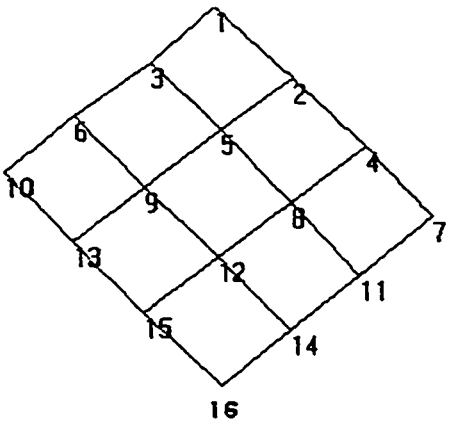


Figure 36. Grid 16

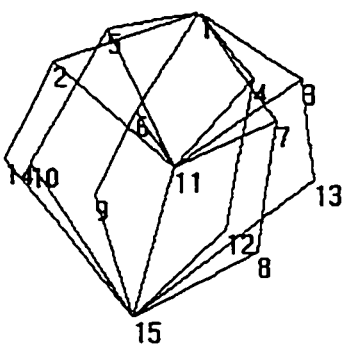


Figure 37. Spider

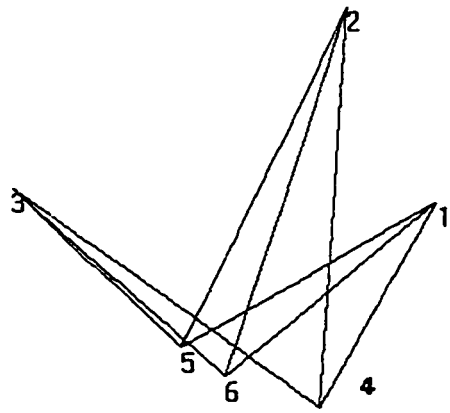


Figure 38. $K_{3,3}$

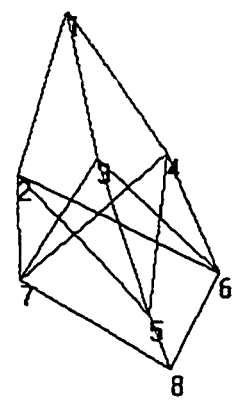


Figure 39. $K_{3,3}$ with top and bottom

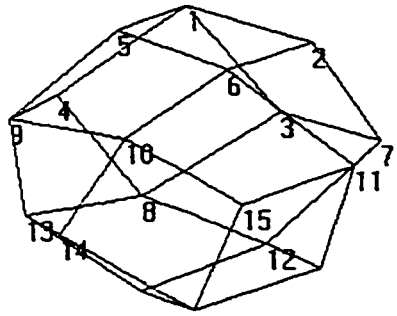


Figure 40. Order 5D

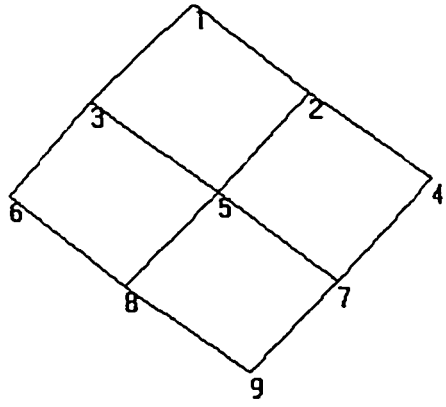


Figure 41. Grid 9

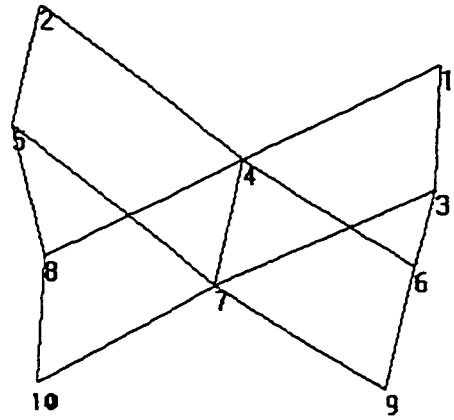


Figure 42. Nonplanar order

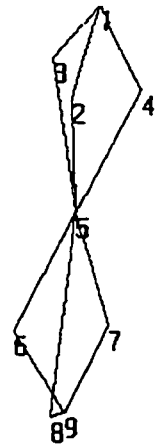


Figure 43. Other order

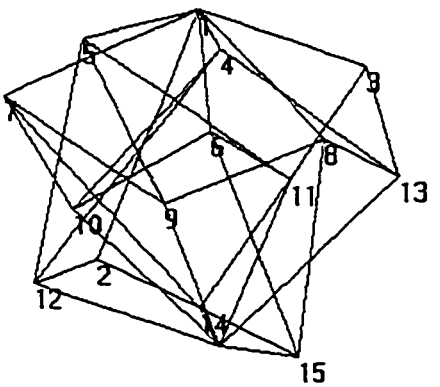


Figure 44. **Projective plane**

The $K_{3,3}$ and the *projective plane* (Figure 38 and 44) are not easy to understand and visualise. In the next section we will introduce the triangulation algorithm to visualise the faces.

Chapter 4

Triangulation

4.1 Introduction

We can enhance the *Force Directed Placement* algorithm with triangulation. In this chapter we introduce an algorithm to triangulate ordered sets and then apply the *Force Directed Placement* algorithm.

To triangulate the ordered set we propose two constraints. The two constraints are:

- global order is preserved, if a vertex a is higher than a vertex b in the original triangulation, then a is higher than b in the final triangulation (if $a > b$ in the original order then $a > b$ in any triangulation)
- the resulting triangulated ordered set is always monotonic. If $a > b$ in the original ordered set, $a > b$ in any triangulation and the path from a to b is monotonic.

We propose an algorithm to triangulate ordered sets by inserting additional edges and vertices to its embedding covering graph.

4.2 Algorithm

The algorithm starts with the polygonal embedding of *cover(p)* which we call G. Graph G is triangulated by adding new edges between the vertices so that all the faces are triangulated. After triangulation, the *Force Directed Placement* algorithm is applied.

The **Simple Triangulation Algorithm** is as follows:

Step 1: triangulate the polygonal embedding of *cover(p)* (call it graph *G*) on the plane by inserting edges between vertices:

1- insert edges to destroy k-gons in each face, where $k \geq 4$

Step 2: input all triangles in a queue call it $T=\{t_1, t_2, \ldots, t_n\}$

Step 3: apply the *Force Directed Placement* algorithm on T

This, of course, can certainly be done provided G is a planar graph.

The quality of output may be enhanced by adding more triangles. To do that, the triangulation algorithm starts with a queue T that contains a set of triangles that describes the faces of the ordered set. At each triangulation level the queue is updated with a new generation of triangles.

The **Triangle Refinement Algorithm** is as follows:

Step 1: start with queue $T = \{ t_1, t_2, \ldots, t_k\}$

while there are non triangulated triangles do:

Step 2 : *remove*(t_i) {remove triangle t_i from the queue }

Step 3 : - let ***a,b*** & **c** be the vertices of the triangle t_i

- remove covering relation between a, b & c

{ a, b & c are non-comparable now}

Step 4 : - create new vertices d, e & f

- assign the following covering relation between ***a,b,c,d,e*** & f

a covers d, d covers b, b covers e, e covers c, f covers c,

d covers e, d covers f, f covers e and a covers f

Step 5 : - assign the coordinates of *d,e,f* as follows

$x_d = (x_a+x_b) / 2,\ y_d = (y_a+y_b) / 2,\ z_d = (z_a+z_b) / 2$

$x_e = (x_b+x_c) / 2,\ y_e = (y_b+y_c) / 2,\ z_e = (z_b+z_c) / 2$

$x_f = (x_a+x_c) / 2,\ y_f = (y_a+y_c) / 2,\ z_f = (z_a+z_c) / 2$

Step 6 : -push into the queue the following triangles:

adf, dbe, def, efc

Endwhile

Step 7: -apply *Force Directed Placement* algorithm

After each triangulation level each triangle is split into four triangles and the number of vertices is increased from 3 to 6 (Figure 45(a)).

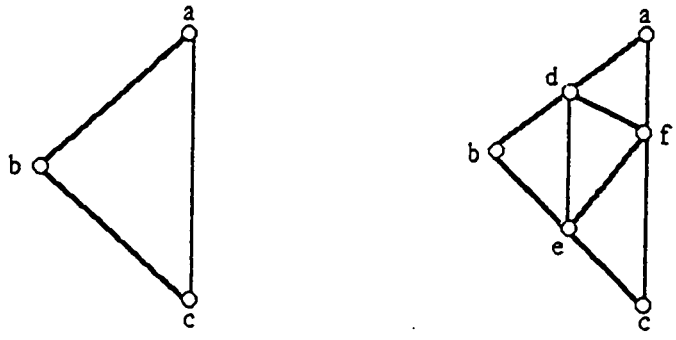


Figure 45. Triangulation (a) Initial triangles (b) After triangulation

After triangulating the triangles in queue *T*, no correction is needed. Consider the case in Figure 46(a). Adjacent triangles share some vertices after triangulation. Figure 46(a) shows two triangles with four vertices; after triangulation, the number of triangles is 8 while the number of vertices is 9 (see Figure 46(b)). Edge 1~4 is a common edge between triangles 124 and 134, vertex 9 is a common vertex between triangles 698 and 597 and no correction is needed.

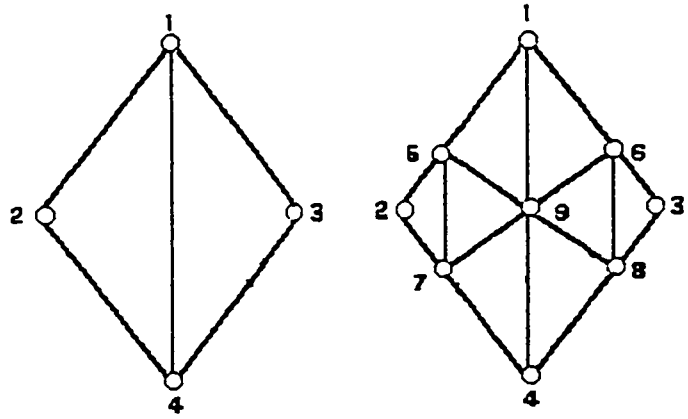
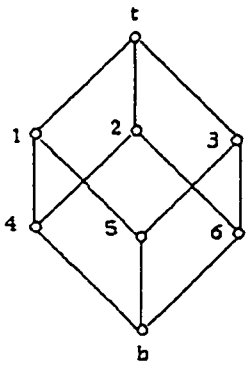


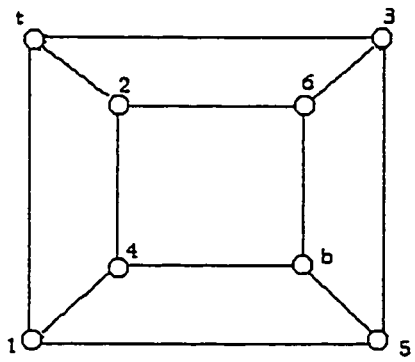
Figure 46. Triangulation (a) Initial triangles (b) After triangulation

As an example, consider the *cube* ordered set. Figure 47 shows the *cube* and its planar graph. In step1 of the algorithm, edges are inserted in each face to destroy k-gons, $k \geq 4$. The dark lines in Figure 48(a) are edges inserted to destroy k-gons, $k \geq 4$ in all faces. Each face consists of 3-gons and no parallel edges are introduced. In step 2 the set of triangles is entered in a queue T. Finally in step 3 the *Force Directed Placement* algorithm is applied.

By following the algorithm, T can be triangulated again by inserting three vertices in each triangle. The result of this triangulation is shown in Figure 48(b) , no parallel edges are created during this step. The output after triangulation is shown in Figure 48(c).

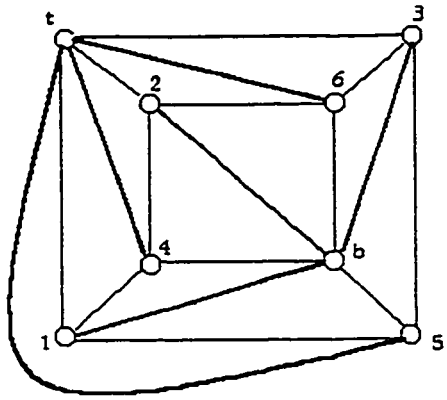


Cube

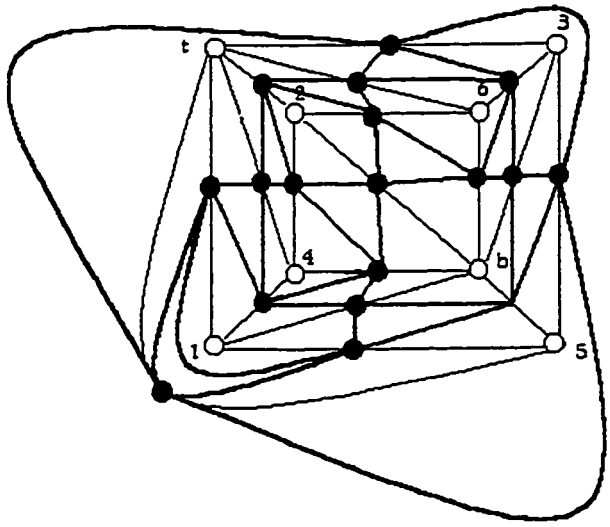


Planar graph

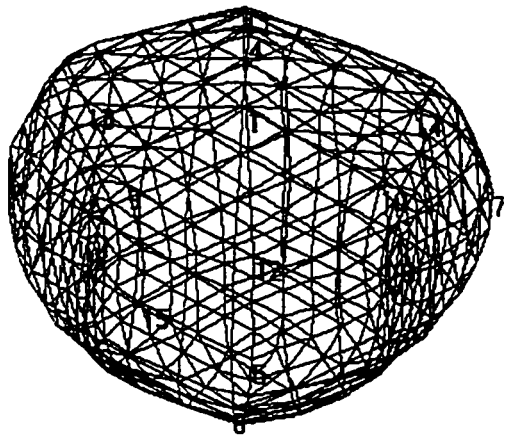
Figure 47. Triangulating the *cube*



(a) Triangulation



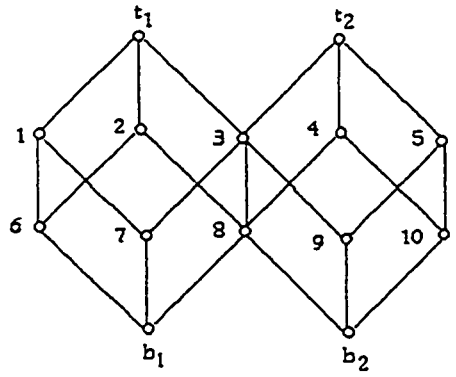
(b) After one triangulation refinement



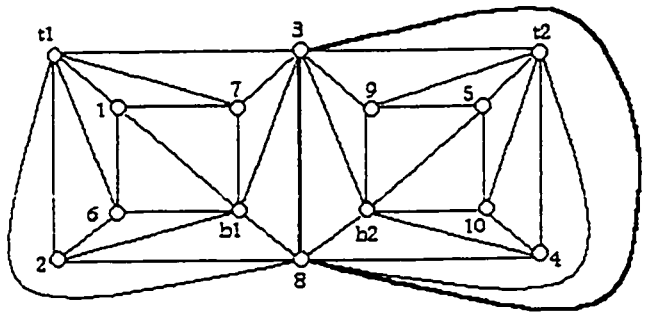
(c) Output of the cube after three triangle refinements

Figure 48. Triangulated cube

Another example is the *double cube*. The double cube and its polygonal embedding are shown in Figure 49. In step 1 of the algorithm, edges are inserted to destroy $\geq$ 4-gons , the result is shown in Figure 49(b). A parallel edge is introduced between vertices 3 and 8. This can be avoided by choosing another triangulation as shown in Figure 50(a). In step 2 the set of triangles are entered in a queue T. T can be triangulated by inserting new three vertices in each triangle. After applying the *Force Directed Placement* algorithm the result is shown in Figure 50(c).

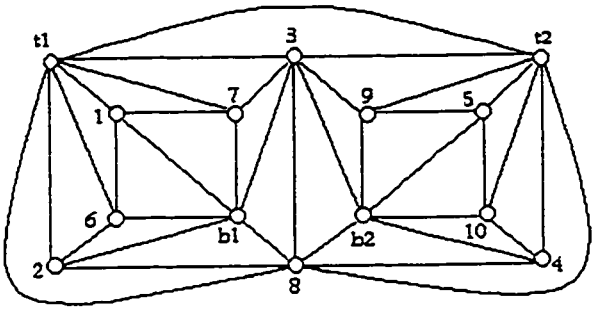


(a) Double cube

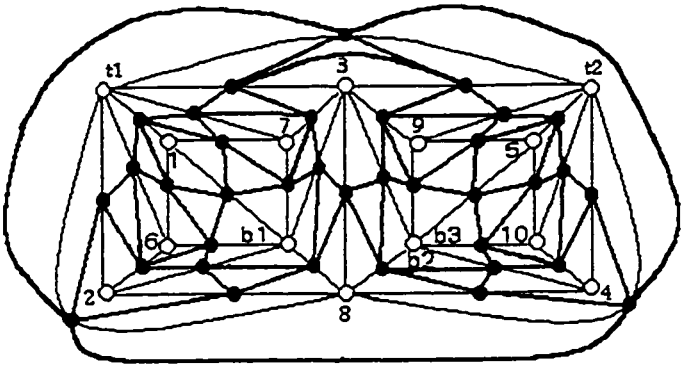


(b) Triangulation with parallel edges

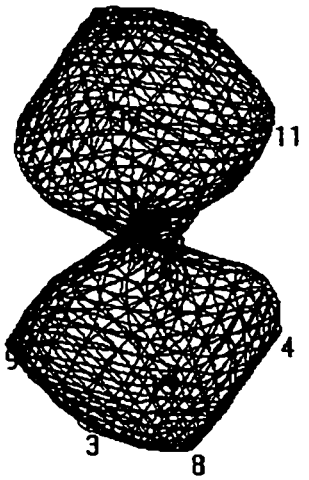
Figure 49. Triangulating the double *cube*



(a) Triangulation without parallel edges



(b) Undirected graph after one triangle refinement



Top view

(c) Output of double cube after three triangle refinements

Figure 50. *Double cube* after applying the *Force Directed Placement* algorithm

Sometimes we cannot avoid parallel edges, especially if the graph is not planar. Therefore, the algorithm should be able to treat parallel edges. Also after triangulation some parallel edges may be introduced. This is especially the case if the graph has *genus* ≥ 1, for example, $K_{3,3}$ (Figure 51). If there exist parallel edges, then the triangulation is corrected by destroying parallel edges. Notice that faces of graphs of *genus* ≥ 1 may contain a vertex more than once.

After triangulation and correction, the *Force Directed Placement* algorithm is applied.

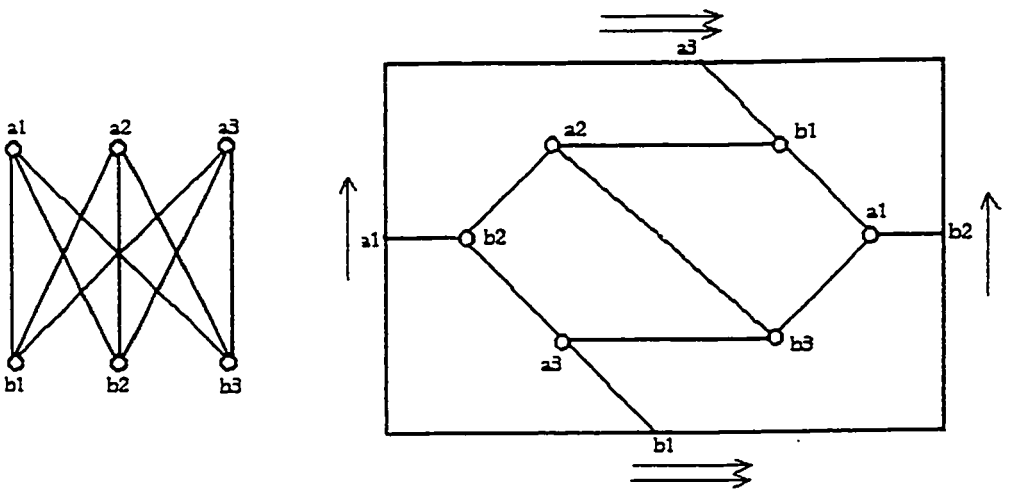


Figure 51. (a) $K_{3,3}$ (b) Embedding on polygonal model of torus

The **Parallel Edge Triangulation Algorithm** is as follows:

Step 1: triangulate the polygonal embedding of *cover(p)* (call it graph G) on the plane by inserting edges between vertices:

1- insert edges to destroy k-gons in each face, where $k \geq 4$

Step 2: check the adjacency list of the triangulated graph G and mark parallel edges to be corrected (parallel edges to be corrected are parallel edges created during **Step1**)

Step 3: for each parallel edge a~b to be corrected do the following:

1- check the right and left neighbouring regions of a~b and treat the following cases:

case1: (i) n parallel edges (n ≥1) adjacent to two 3-gons ($a\ x_1\ b$ and $a\ x_2\ b$) from left and right (Figure 52 a).

- split the parallel edges by inserting new vertices $v_1, v_2, \ldots, v_n$
- add new edges between vertices as follows : x_1~v_1, v_1~v_2, v_2~v_3, …, v_n~x_2 (Figure 52 b)

(ii) 1 parallel edge adjacent to two 3-gons ($a\ x_1\ b$ and $a\ x_2\ b$, x_1=x_2) from left and right (Figure 52 c).

- split the parallel edge by inserting new vertex v_1
- add new edges between vertices as follows : x_1~v_1, x_2~v_1
- split the parallel edges x_1~$v_1$and x_2~v_1 by inserting new vertices v_2 and v_3
- add new edges between vertices as follows : a~v_2, b~v_2, a~v_3 and b~v_3 (Figure 52 d)

case2: n parallel edges ($n \geq 1$) adjacent to one 3-gons ($a\,x\,b$) (Figure 52 e).

- split the parallel edges by inserting new vertices $v_1, v_2, \ldots, v_n$
- add new edges between vertices as follows:

$x_1 \sim v_1, v_1 \sim v_2, v_2 \sim v_3, \ldots, v_{n-1} \sim v_n$ (Figure 52 f)

case3: (i) n parallel edges (Figure 52 g).

- split the parallel edges by inserting new vertices $v_1, v_2, \ldots, v_n$
- add new edges between vertices as follows:

$v_1 \sim v_2, v_2 \sim v_3, \ldots, v_{n-1} \sim v_n$ (Figure 52 h)

(ii) 1 parallel edge (Figure 52 i).

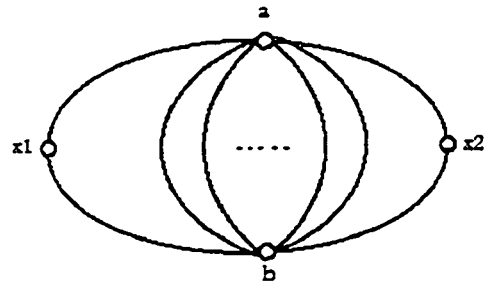
- add new parallel edge between vertices a and b:
- split the parallel edges by inserting new vertices v_1, v_2
- add new edge between vertices as follows:

$v_1 \sim v_2$ (Figure 52 j)

Step 4: input all triangles in a queue which is called $T=\{t_1, t_2, \ldots, t_n\}$

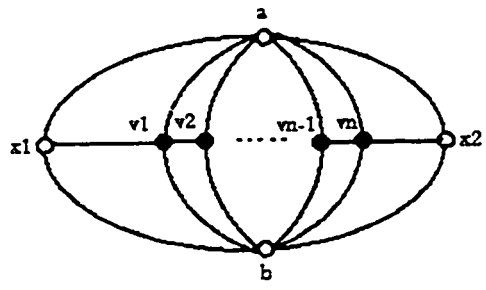
Step 5: apply the *Force Directed Placement* algorithm on T

Case 1:



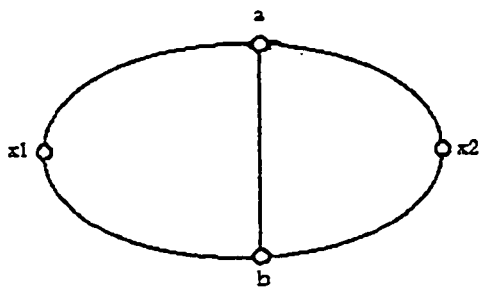
(i) Input (x1≠ x2 and n ≥ 1) or (x1 = x2 and n > 1)

(a)



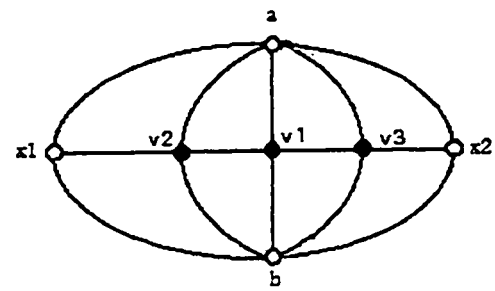
Output

(b)



(ii) Input (x1=x2 and n=1)

(c)

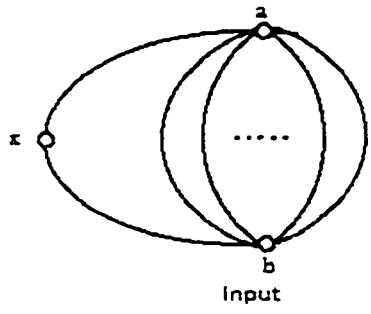


Output

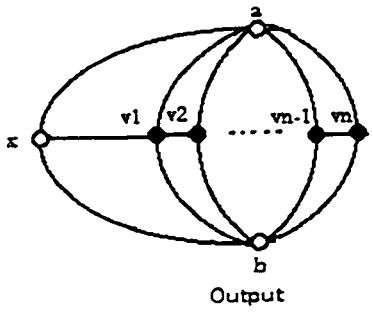
(d)

Figure 52. Parallel edge correction

Case 2:

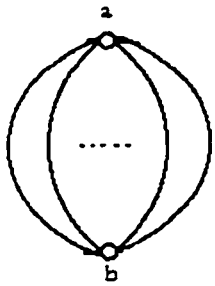


(e)

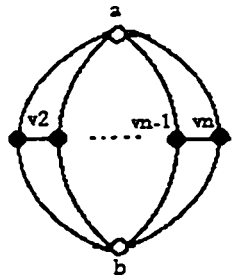


(f)

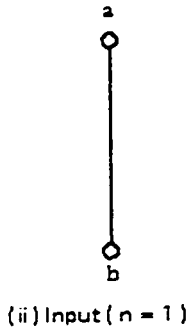
Case 3:



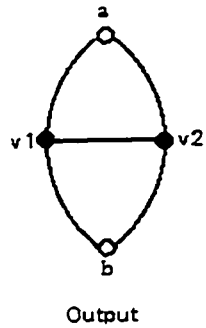
(g)



(h)



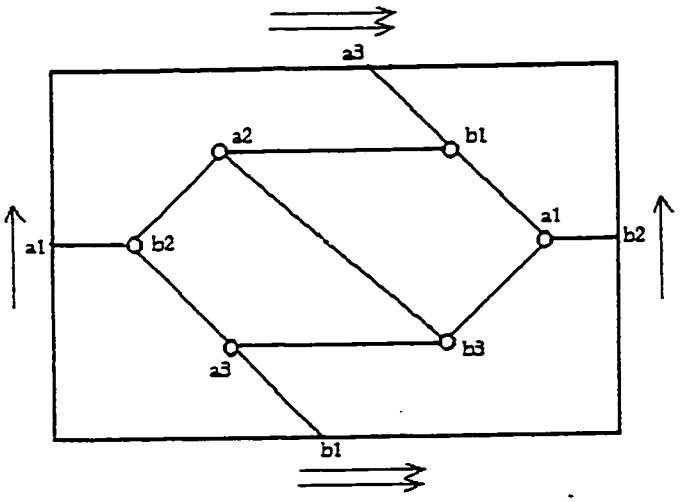
(i)



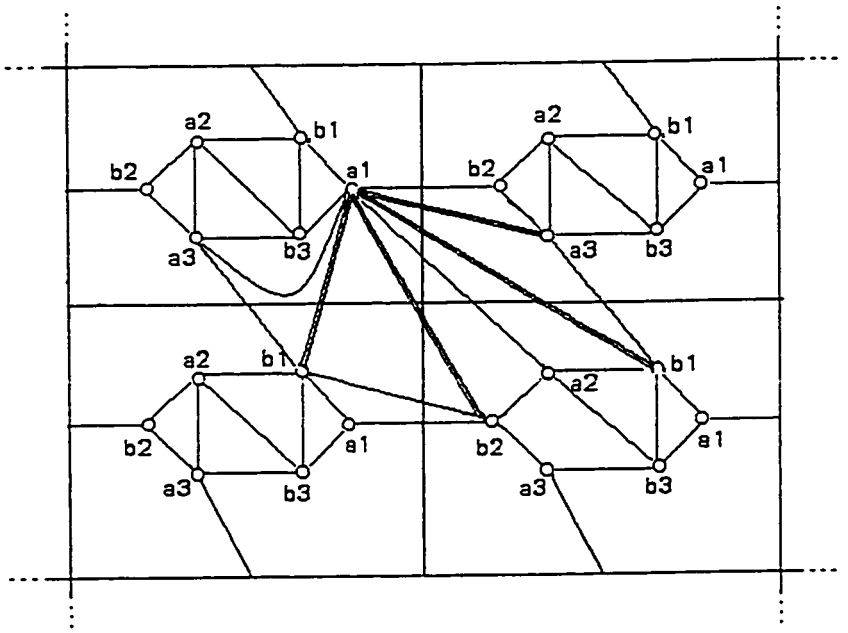
(j)

Figure 52.(continued) Parallel edge correction

As an example, consider the $K_{3,3}$ ordered set. Its polygonal embedding is graph G in Figure 53(a). Figure 53(b) shows the graph G after triangulation, some parallel edges are introduced. The double edges are the parallel edges. Figure 54 shows graph G after destroying the parallel edges. In Figure 54(a) parallel edge a_1~a_3 is destroyed by inserting vertex v_1 and creating edges v_1~b_2 and v_1~b_1. In Figure 54(b) parallel edge a_1~a_3 is destroyed by inserting vertex v_2 and creating edges v_2~a_2 and v_2~v_1. Parallel edge a_1~b_2 is destroyed by inserting new vertex v_3 and creating two edges v_3~b_1 and v_3~a_2(Figure 54(c)) . Finally, parallel edge a_1~b_1 is destroyed by inserting new vertex v_4 and creating two edges v_4~a_3 and v_4~v_3 (Figure 54 (d)). The polygonal embedding of the triangulation is shown in Figure 55(a) and the embedding on a torus is shown in Figure 55(b). The polygonal embedding of the correction is shown in Figure 56. After applying the *Force Directed Placement* algorithm the output is as shown in Figure 57.

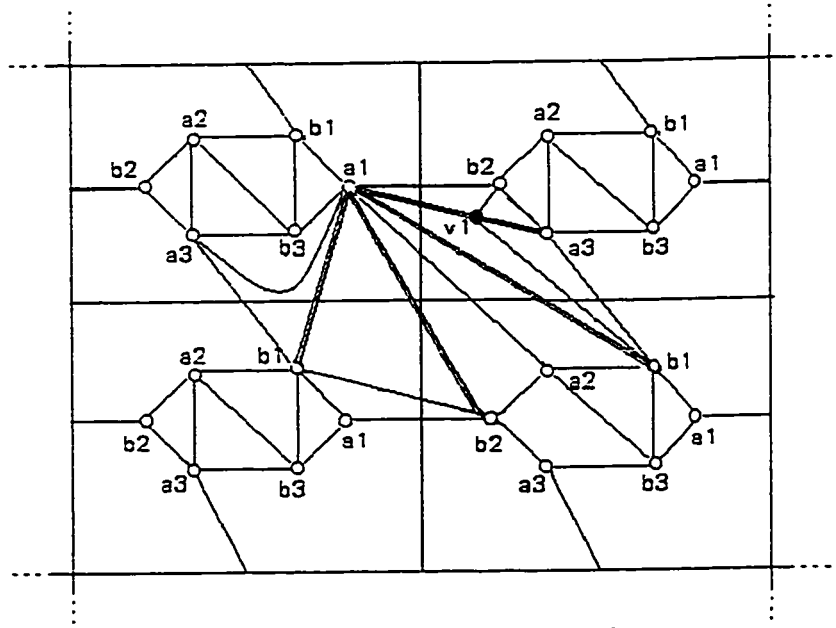


(a) Polygonal embedding of $K_{3,3}$ with the faces: $(\{a2,b2,a3,b3\},\{a1,b1,a2,b3\},\{a2,b3,a3,b1,b2,a2,b1,a3,b2\})$

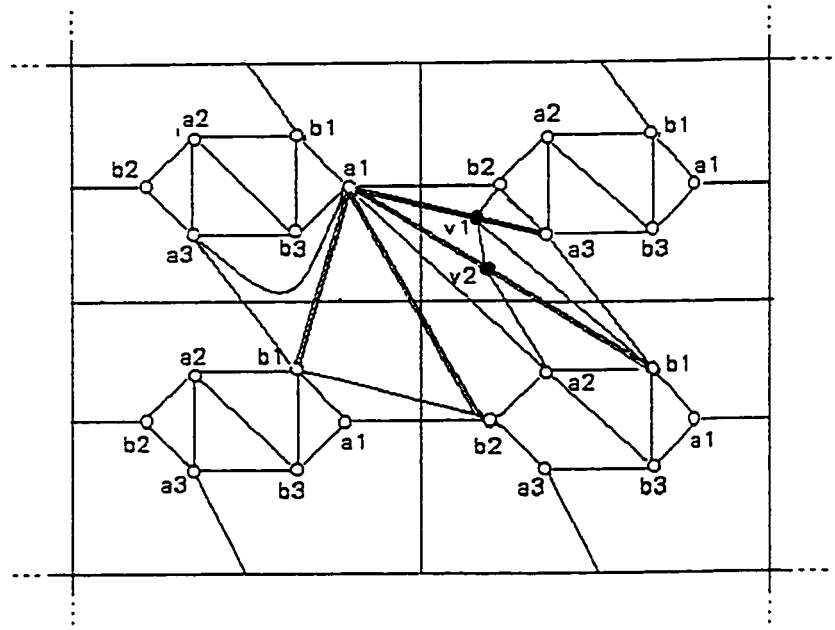


(b) A partial "triangulation" of the universal covering space of this polygonal embedding

Figure 53. Triangulating the $K_{3,3}$

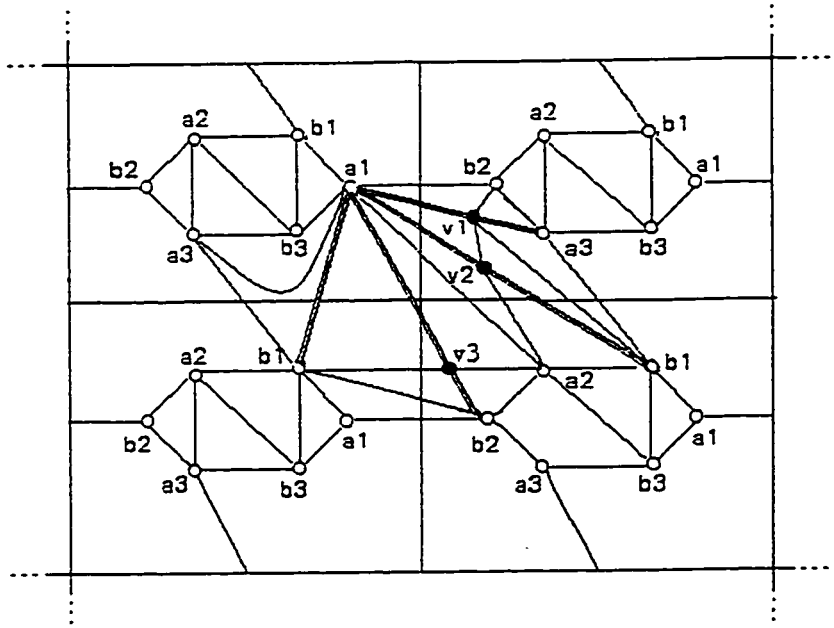


(a) Destroying parallel edge a1~a3

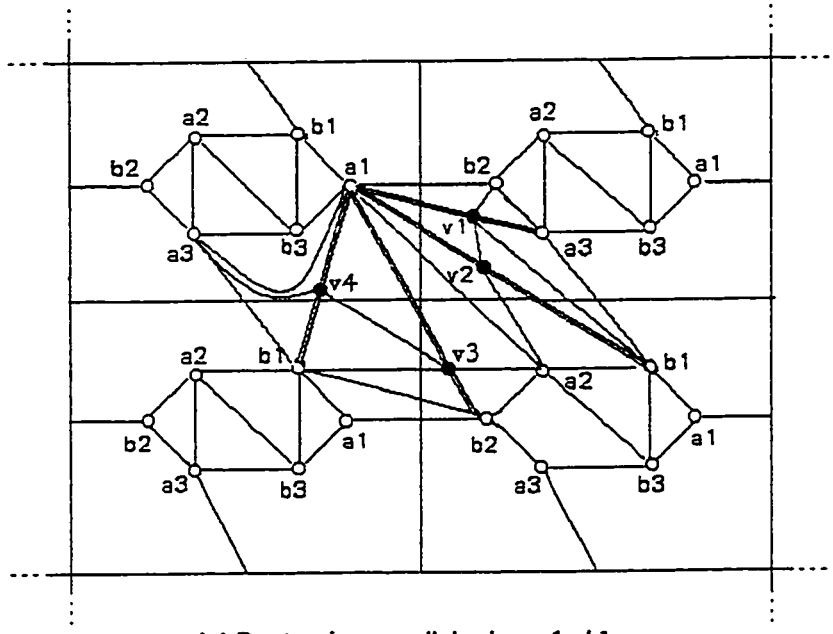


(b) Destroying parallel edge a1~b1

Figure 54. Correcting parallel edges

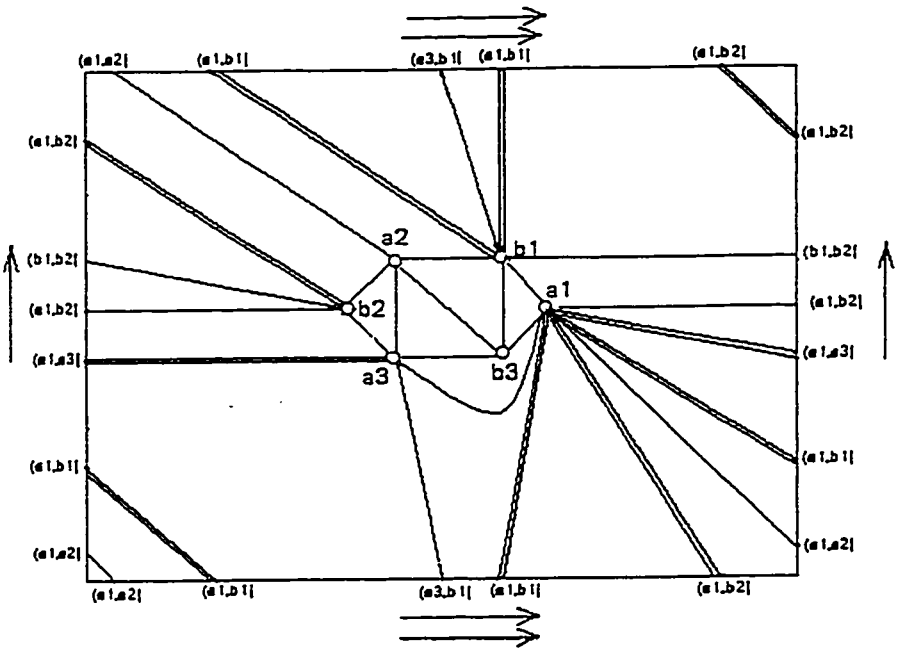


(c) Destroying parallel edge a1–b2

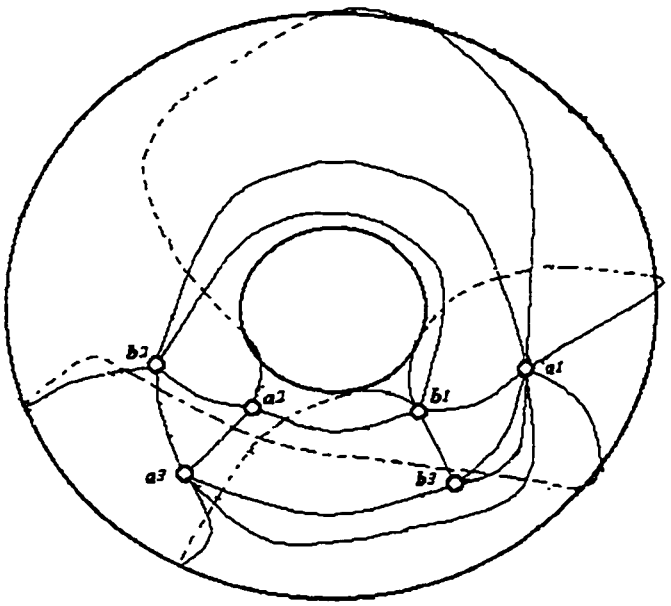


(d) Destroying parallel edge a1–b1

Figure 54. (continued) Correcting parallel edges



(a) Polygonal embedding of the triangulation



(b) Embedding on a torus

Figure 55. Embedding of the triangulation

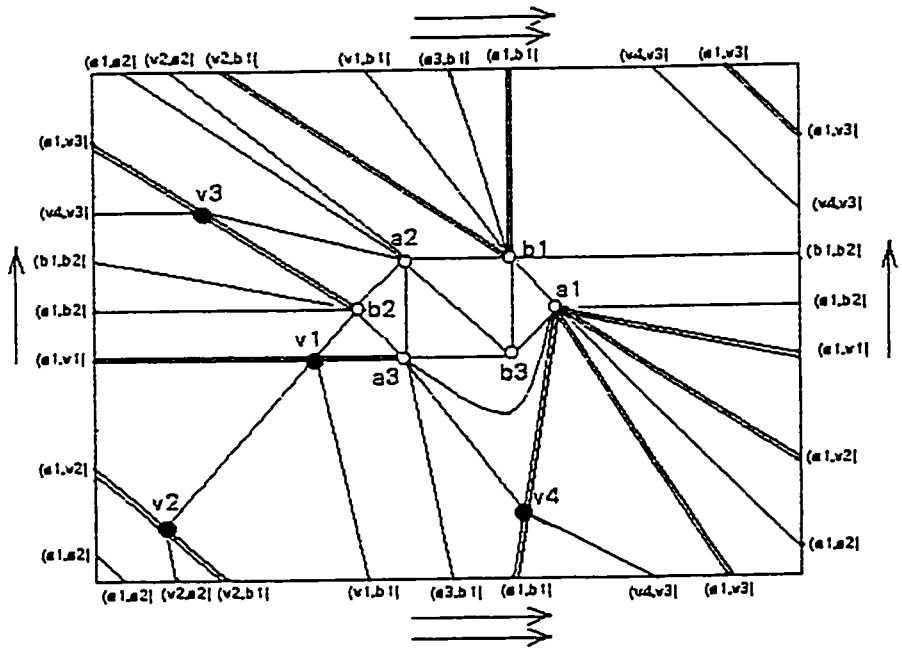
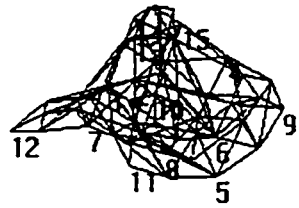
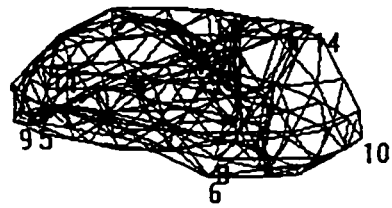


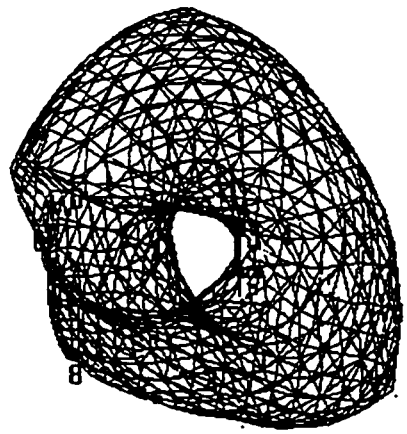
Figure 56. Polygonal embedding of the corrected triangulation



(a) After one triangle refinement



(b) After two triangle refinements



(c) After three triangle refinements

Figure 57. Output after applying the *Force Directed Placement* algorithm

4.3 Triangulating some orders

In this section the triangulation algorithm is applied to the *cube*, *double cube*, $K_{3,3}$ and the *projective plane*. The system used during the experiment is a Pentium P60 MHZ PC with 8 MB RAM.

4.3.1 Cube

The *cube* has a discrete structure, we will try to view it as smooth structure using our triangulation algorithm.

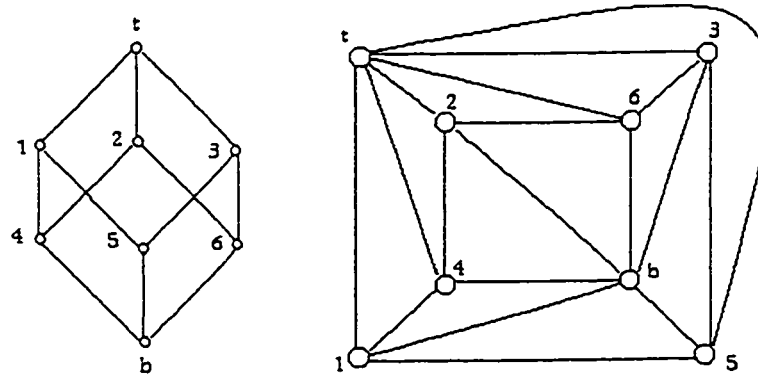


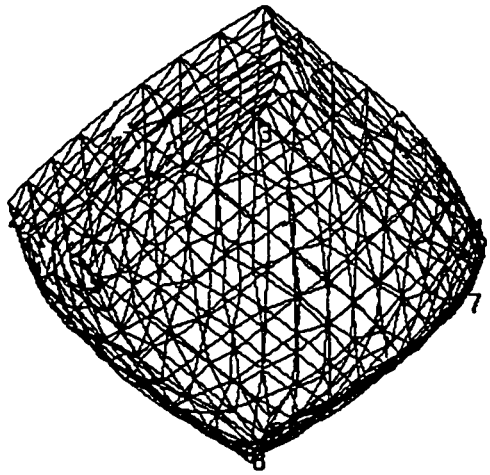
Figure 58. **(a) Cube** **(b)Triangulation**

After applying the triangulation algorithm the *cube* is triangulated as shown in Figure 58(b). The set of triangles is as follows:

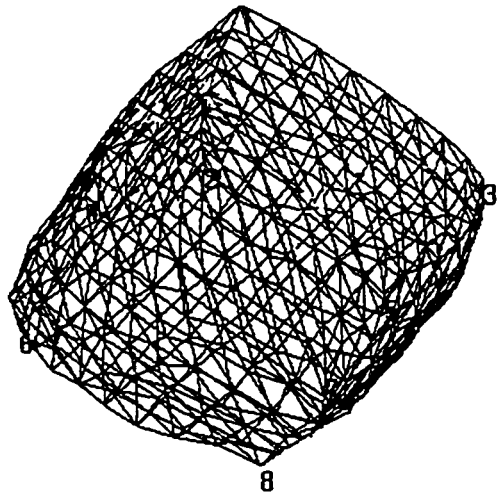
$T = \{(t,1,4),(t,2,4),(t,3,6),(t,2,6),(t,1,5),((t,3,5),(2,4,b),(2,6,b),(3,5,b),(3,6,b),(1,4,b),(1,5,b)\}$. The value of the constants (K_{spring} and C_r) affect the final shape of the cube. Figure 59 shows a cube after 3 levels of triangulation where we got 378 vertices and 768 triangles with K_{spring} =4 , C_r = 1 and L_s = 2.

The *cube* appears here as a discrete structure , we changed the constants to $K_{spring} = 1$, $C_r = 4$ and $L_s = 2.5$. The results appeared to be more differentiable (Figure 60).

The time required to produce the orders in Figures 59 and 60 is around 360 seconds. Figure 59 has a discrete structure while Figure 60 has sphere like structure.

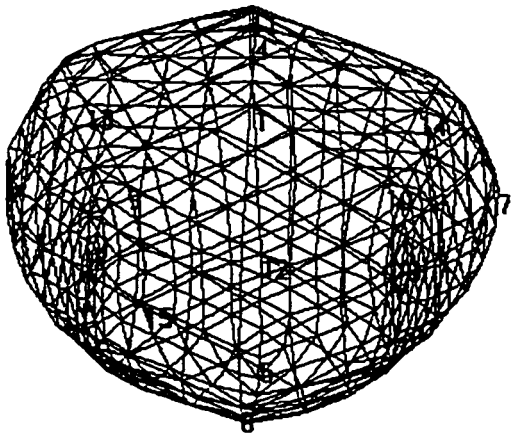


Side view

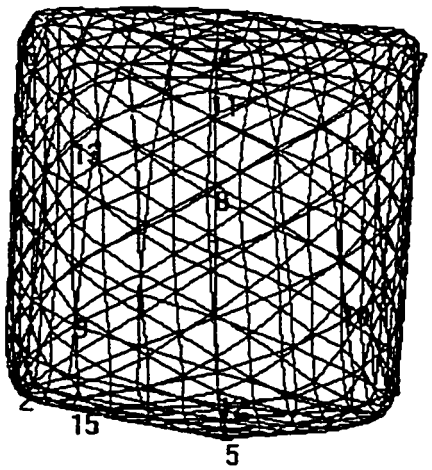


Another side view (after 90 degree rotation)

Figure 59. *Cube* ($C_r = 1$, $K_{spring} = 4$, $L_s = 2.5$)



Side view



Top view

Figure 60. *Cube* ($C_r = 4$, $K_{spring} = 1$, $L_s = 2$)

4.3.2 Double cube

Another order that we want to triangulate is the *double cube* . The *double cube* order is shown in Figure 61(a), its triangulated polygonal embedding is shown in Figure 61 (b).

After applying the triangulation algorithm, the set of triangles is as follows:

T = { $(t_1,2,6),(t_1,1,6),(t_1,1,7),(3,7,b_1),(3,8,b_1),(t_1,3,7),(2,6,b_1)$,$(2,8,b_1),(3,9,t_2),(3,8,b_2)$,$(3,9,b_2),(t_2,5,9),(t_2,5,10),(t_2,4,10),(5,9,t_2),(5,10,b_2),(4,10,b_2),(4,8,b_2),(t_1,t_2,3),(t_1,2,8),(t_2,4,8)$,$(t_1,t_2,8)$ }.

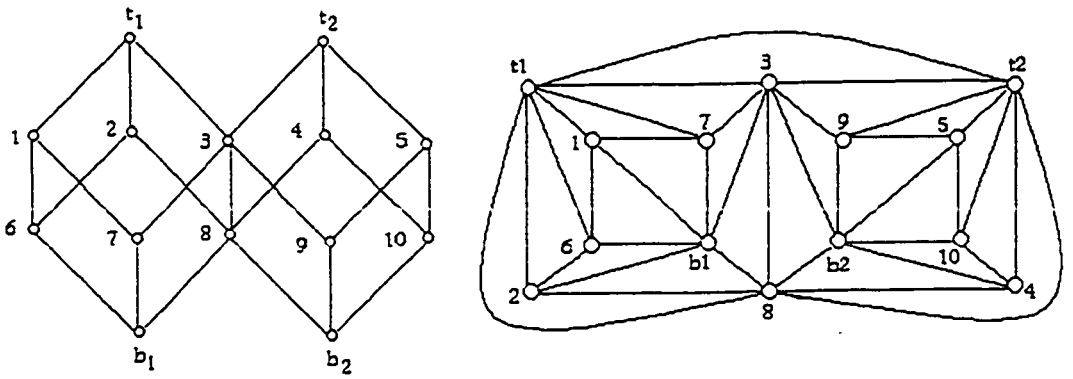
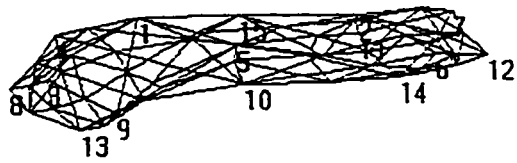


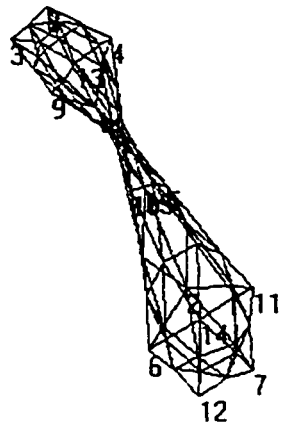
Figure 61. (a) Double cube (b) Triangulation

We started with the triangles set T and applied our algorithm with the following constants L_s = 1, K_{spring} = 2 and C_r = 2. After one triangle refinement the result is as shown in Figure 62.

The result seems to be not smooth and the faces are not clear. Another triangle refinement was applied; the result is more smooth as shown in Figure 63.
Another triangle refinement was applied. The result is shown in Figure 64. Compared to Figure 62, the *double cube* is changed from a non smooth structure to a smooth structure by applying more triangle refinements.

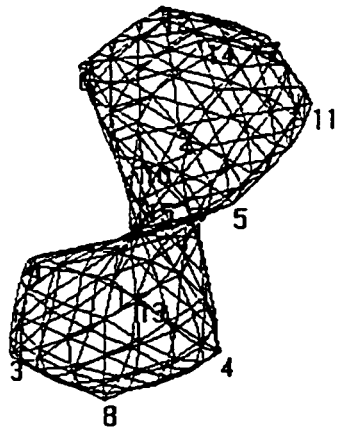


Side view

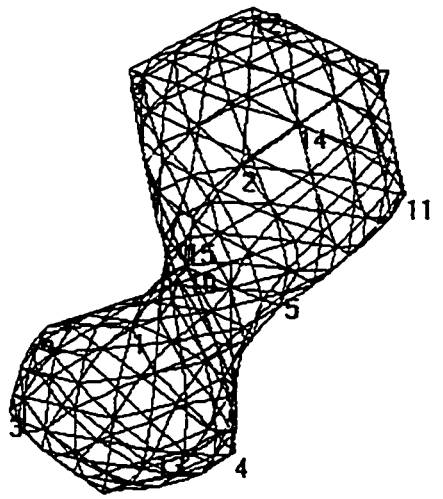


Top view

Figure 62. Double cube after one triangle refinement

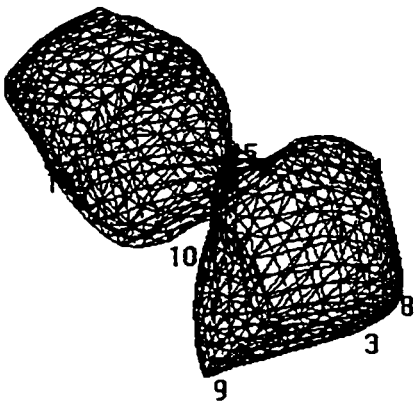


Side view

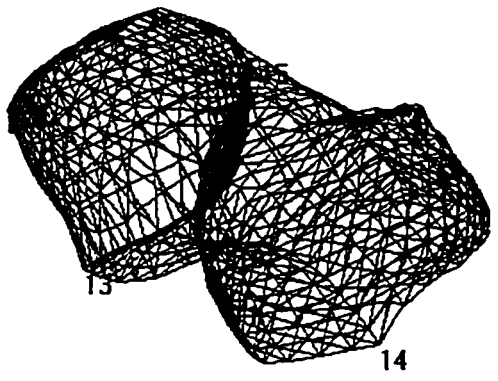


Top view

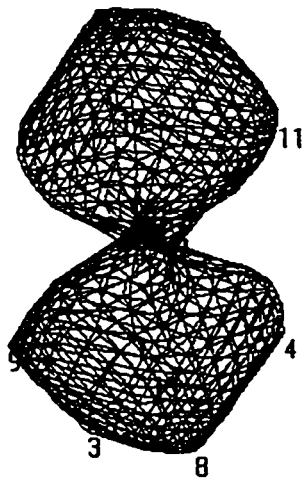
Figure 63. Double cube after two triangle refinements



Side view



Side view



Top view

Figure 64. Double cube after three triangle refinements

4.3.3 $K_{3,3}$

The $K_{3,3}$ (Figure 65(a)) has complicated structure that makes it difficult to visualise. The $K_{3,3}$ can be embedded on polygonal model of a tourus as shown in Figure 65(b).

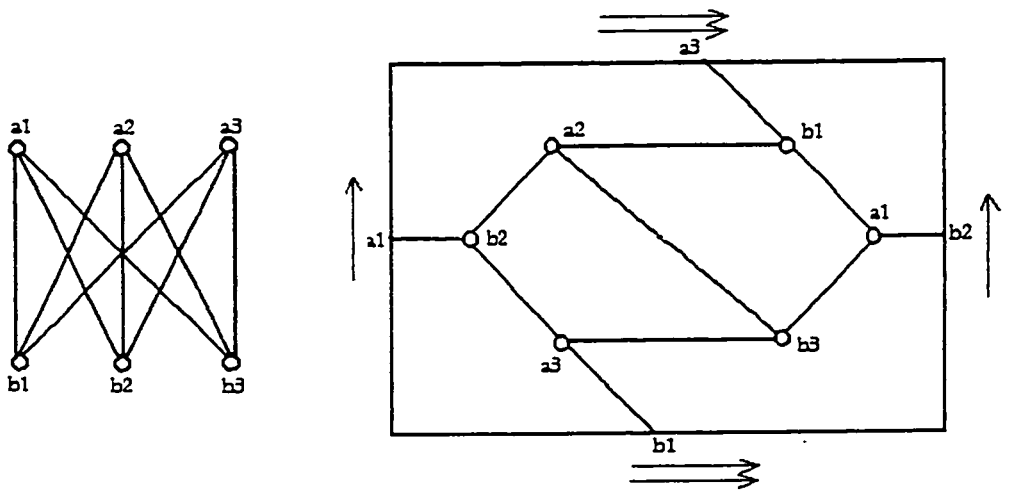


Figure 65. (a) $K_{3,3}$ (b) Embedding on polygonal model of torus

The triangulation algorithm is applied and the $K_{3,3}$ is triangulated (Figure 66) such that the global order is not violated and no parallel edges are allowed.

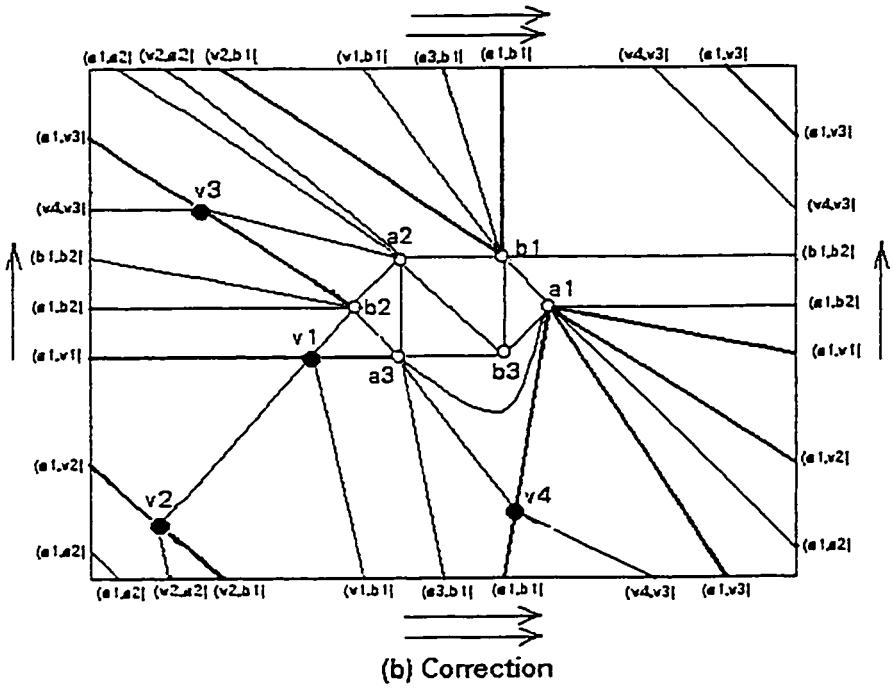


Figure 66. Triangulated embedding of $K_{3,3}$ on polygonal model of a torus

The algorithm is applied with the following constants : $K_{spring} = 2$, $C_r = 2$ and $L_s = 1$, After three levels of triangulation and 600s the output is clearly a torus as shown in Figure 67.

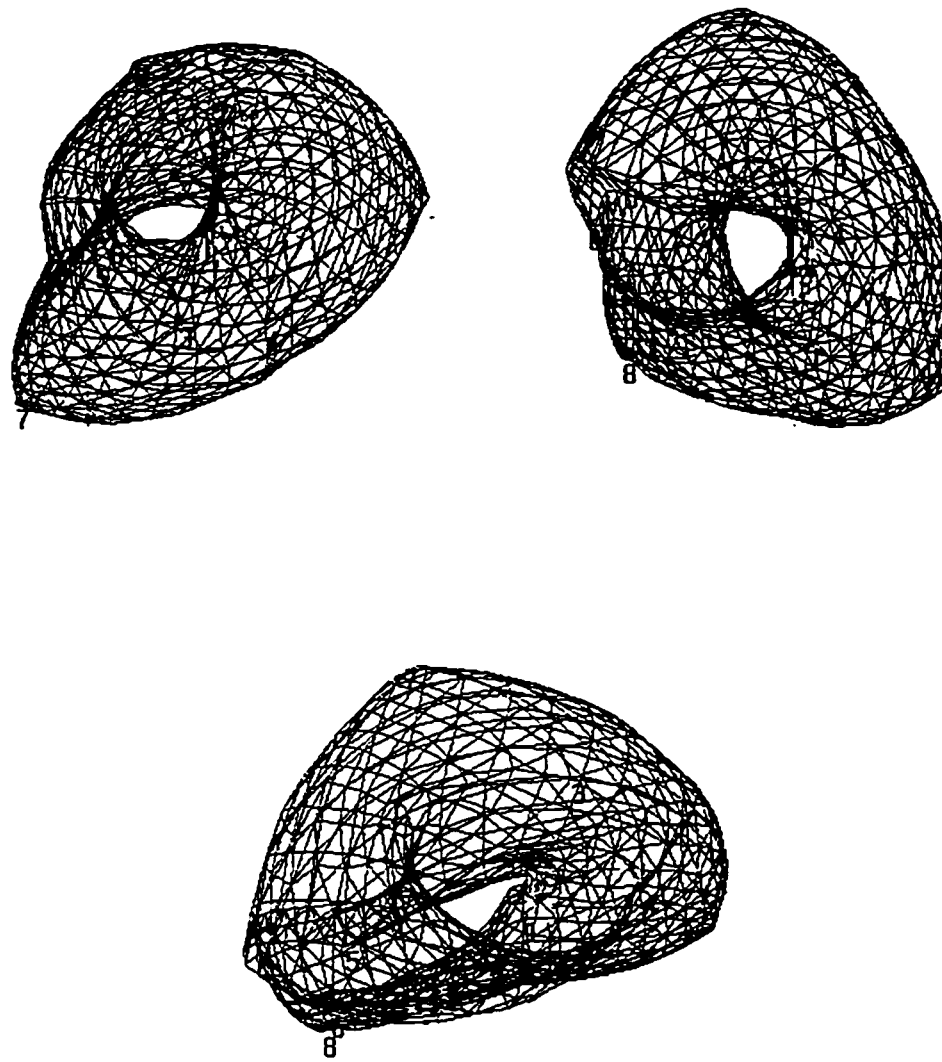


Figure 67. $K_{3,3}$ with $K_{spring} = 2$, $C_r = 2$, $L_s = 1$

4.3.4 Projective plane

The projective plane has a complicated structure that makes it one interesting order to visualise. To date there is still no satisfactory actual drawing either of its undirected graph or its upward drawing on a surface with crossing edges. Figure 68 shows an upward drawing of the *projective plane* on the plane. As it is shown it is difficult to visualise the relation between vertices.

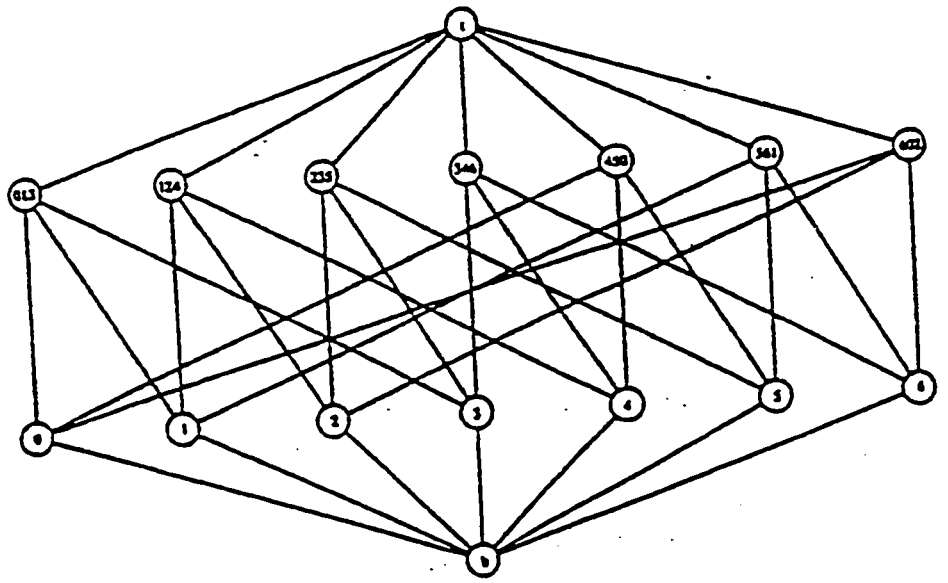


Figure 68. An upward drawing of the *projective plane* on the plane

One triangulation of the *projective plane* is as shown in Figure 69. After two triangle refinements this produces the upward drawing illustrated in Figure 70.

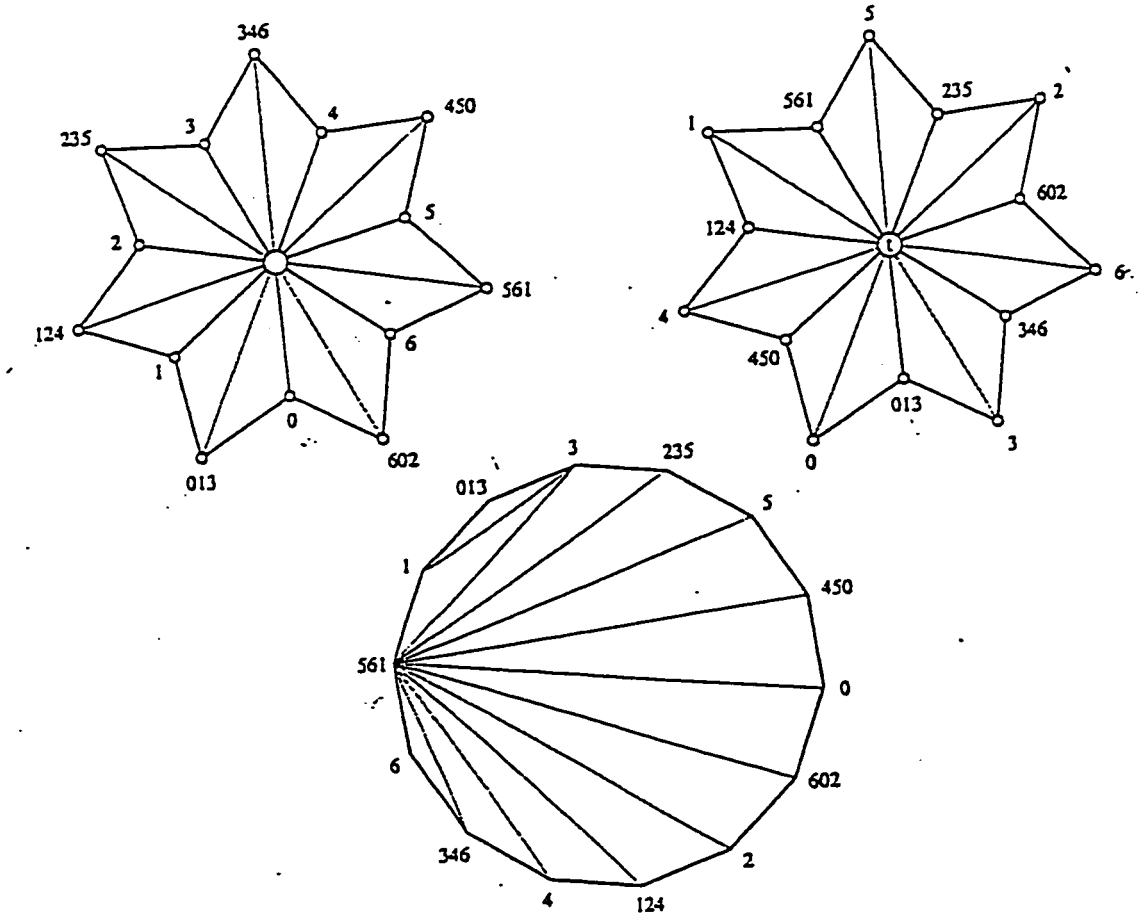
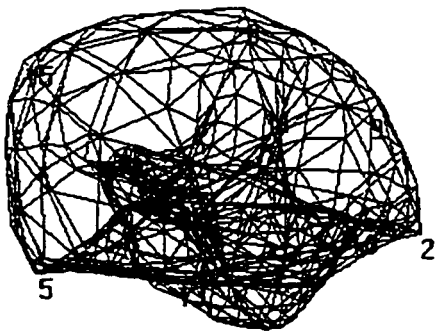
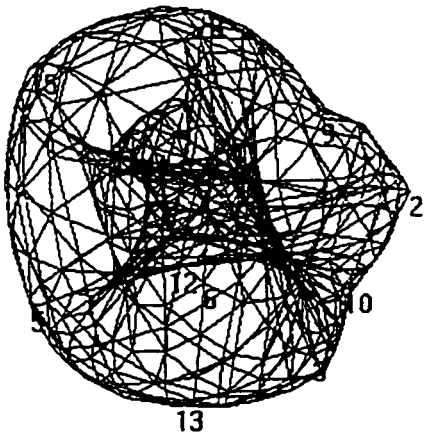


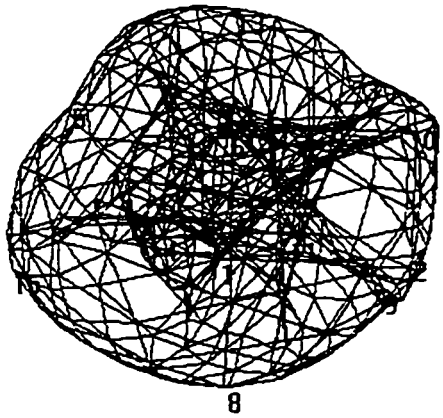
Figure 69. One triangulation of the *projective plane*



Side view



Top view



Bottom view

Figure 70. *Projective plane* of Figure 69.

The *projective plane* is then triangulated as shown in Figure 71. No parallel edges are added and we added two temporary vertices (*a* and *b*) in order to avoid adding parallel edges.

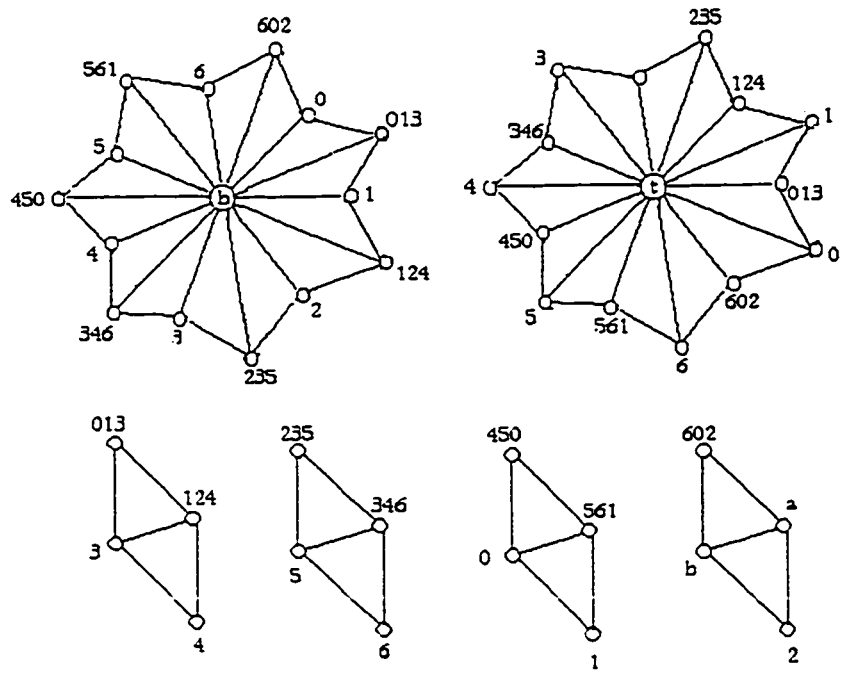


Figure 71. Triangulating the projective plane

First we applied the algorithm with the triangles as shown in Figure 71. After 300 seconds and two levels of triangulation we got 576 triangles and 214 vertices and the output is as shown in Figure 72.

At this writing we have no automatic technique to detect "handles". Nevertheless, visually we can detect handles. In Figure 72 (a)(Top view) three handles are shown. In

order to enhance the visibility of the projective plane we then applied other methods. The first method is to let the vertices of the handles be neutral with respect to any vertex that does not belong to a handle; in other words we changed the equation for calculating the repulsive force between vertices to be as follows:

$$Fr_{ij} = \begin{cases} 0 \ \textit{if } v_i \textit{ belongs to any handle and } v_j \textit{ does not} \\ \dfrac{Cr}{D^2_{ij}} \ \textit{if } v_i \textit{ does not belong to any handle but } v_j \textit{ does} \end{cases}$$

The vertices located in the handles can repel each other , also the vertices outside the handles can repel each other, but vertices from the handles cannot repel vertices outside, while vertices from outside can repel vertices in the handles.

The equation for calculating the repulsive forces can be given as :

$$Fr_{ij} = u(1 - handle(i)) * \frac{Cr}{D^2_{ij}}$$

Where *u(1-handle(i))* is the unit step function [Kr88] defined as

$$u(t) = \begin{cases} 1 \ if \ t > 0 \\ 0 \ otherwise \end{cases}$$

and *handle(i)* is defined as follows

$$handle(i) = \begin{cases} 1 \ if \ v_i \in \ to \ a \ handle \\ 0 \ otherwise \end{cases}$$

We applied the algorithm with the constants $K_{spring} = 2$, $C_r = 5$ and $L_s = 1$ for 400 seconds, the result is as shown in Figure 73(a & b). The output is more visualizable and the handles are more clear.

The output still needs some modifications to enhance the view.
To enhance the visibility we tried another method. The spherical drawing algorithm is applied on the vertices that do not belong to the handles .

We changed C_r from 5 to 2 and maintained the same other constants as before and applied the algorithm for 300 seconds the result was pleasing and the *projective plane* is shown as a sphere with four handles Figure 74. Figures 74(b) and (d) are outline drawings of Figures 74(a) and (c) respectively.
As an undirected graph it is known that the *projective plane* has *genus* three. We have displayed a *genus* four upward drawing. We do not know whether there is a *genus* three upward drawing.

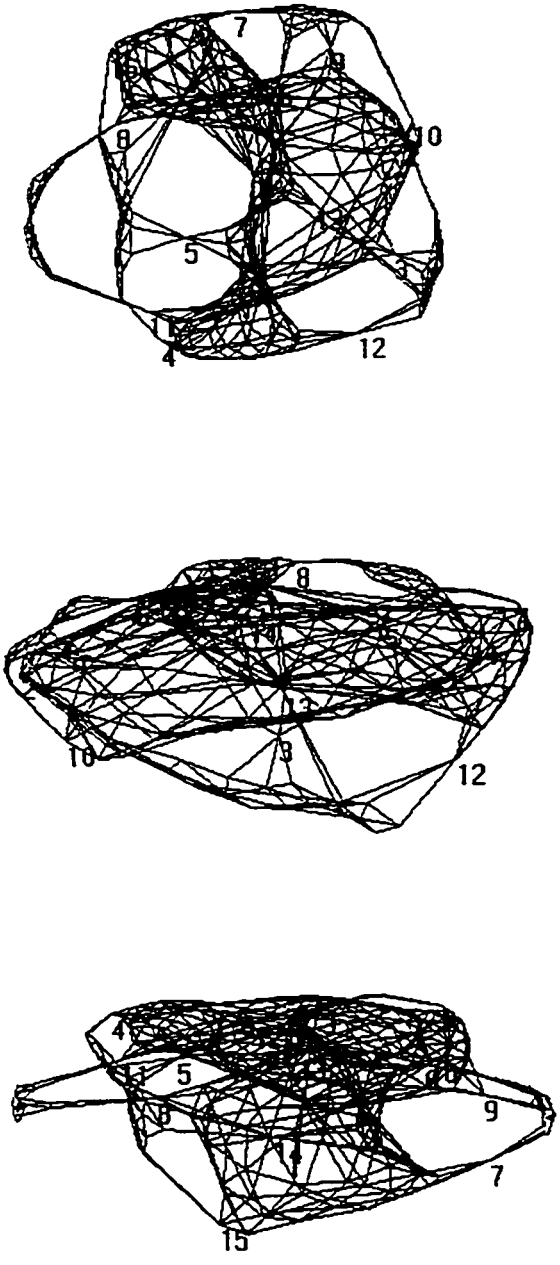


Figure 72. Projective plane ($K_{spring} = 3$, $C_r = 1$, $L_s = 1$)

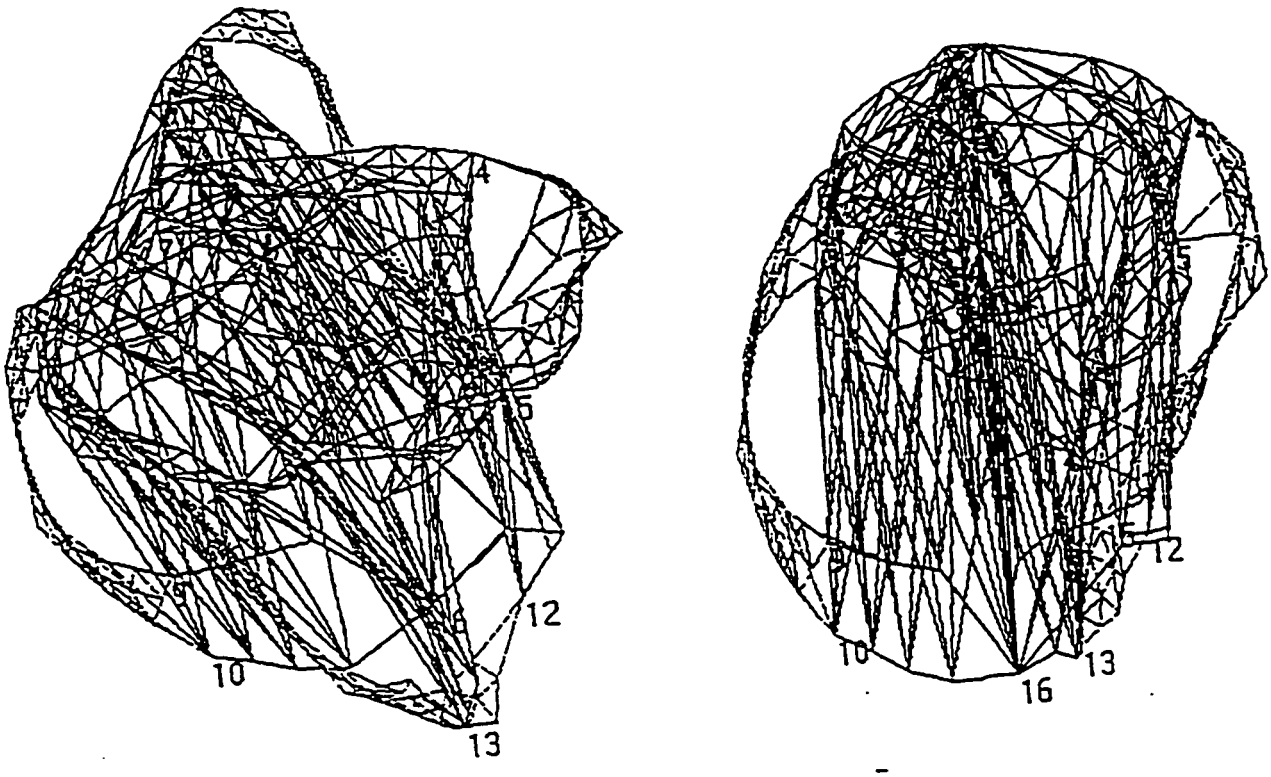
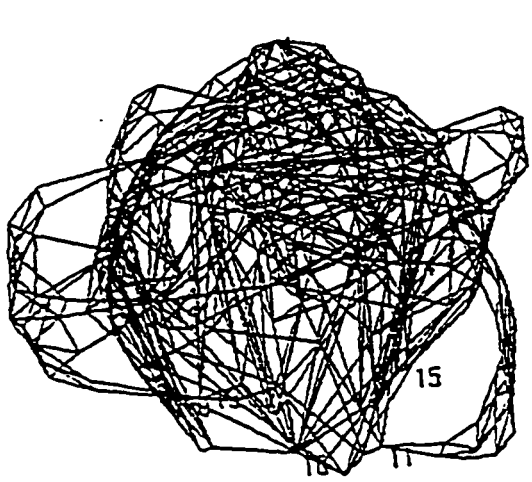
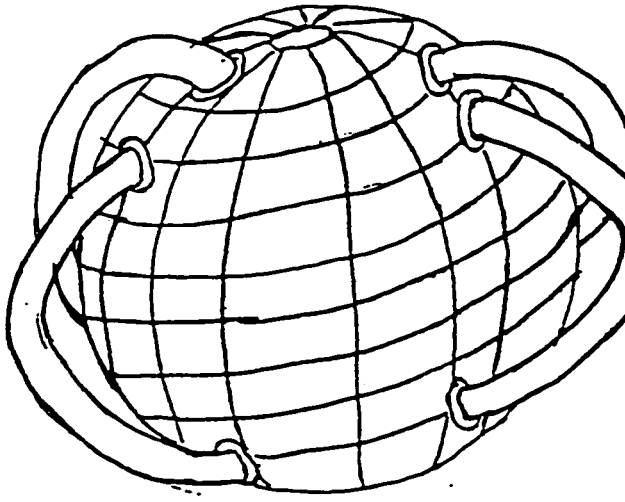


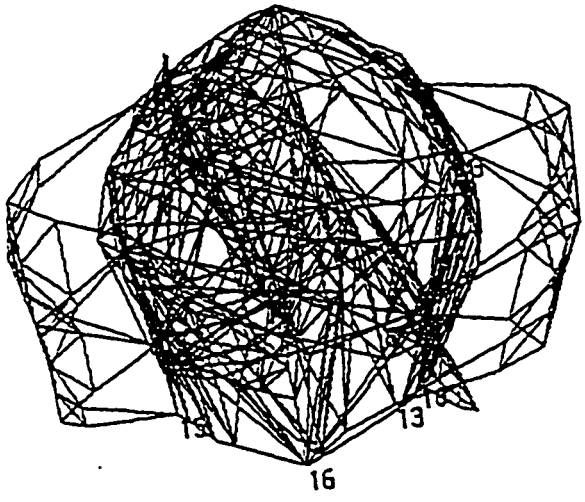
Figure 73. ***Projective plane*** **with handles not affecting the sphere**



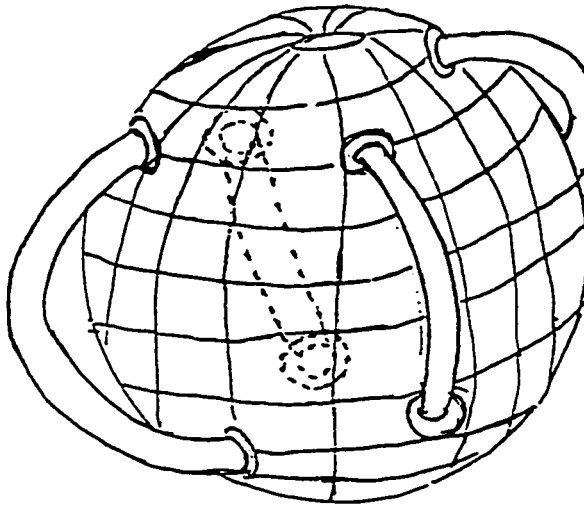
(a)



(b)



(c)



(d)

Figure 74. *Projective plane* ($K_{spring} = 2$, $K_{sphere} = 0.2$, $R_{sphere} = 1000$, $C_r = 2$)

Chapter 5

Implementation

5.1 Interface

In visualising 3D graphs, an important aspect resides in the ability to dynamically change the view of the graph. The interface to our system is user friendly and the user is able to enter the order by only using the mouse and in a graphical style, that is, without keyboarding.

The algorithm is implemented using Visual Basic 3.0. The 3D layout is presented with visual clues (colour, perspective and light). The user may choose to view the development of the graph step-by-step.

The interface of the program is the following:

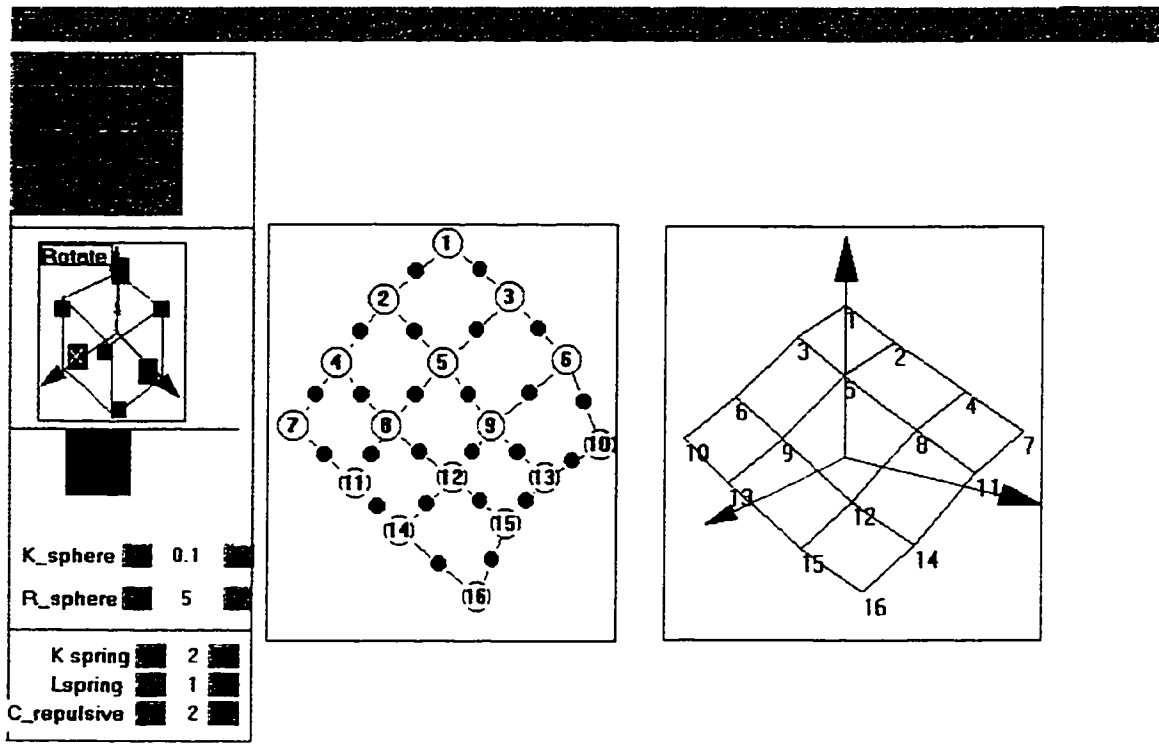


Figure 75. Interface

Using the ***controlbox*** (Figure 75, left), the user is able to control the input and output by using the following buttons:

Show order: shows the graph in its initial state in the output screen

Order: consider the input as an ordered set

Random: initiates the graph or order with random positioning

Pause: pause the current executing function

Zoom+: enlarge the view in the output screen

Zoom-: reduce the view in the output screen

Rotate: used to rotate the graph or order in the output screen around the *x,y* and *z axis* and around the *xy-axis*, *xz-axis* , *yz-axis* and *xyz-axis*.

Ksphere: change the sphere constant

Rsphere: change the radius of the sphere

Kspring: change the spring constant

C-repulsive: change the repulsive force constants

The pull-down menu contains two choices : **File** and **Options** (Figure 76).

The File option contains the following:

New : to enter new order, the current order is erased.

Open File : displays a saved input on the input screen.

Save file : saves the current input in a file.

Exit : terminates the execution of the program.

The Options choice contains the following:

Colors : displays the colour icon (Figure 79 (b)) which is used to change the colour of the order in the output screen

Graph : treats the input as a graph.

Figure 76. Pull-down menu

Using the ***Inputscreen*** (Figure 75, middle), the user may input or update the initial graph or order by using the mouse. A new vertex can be inserted by clicking on the screen, two vertices may be joined by an edge by clicking on both of them one after the other.

The ***Outputscreen*** (Figure 75, right) is used for output only and the user can change the backcolor, the vertex colour and the edge colour by using the ***colors*** option from the main menu.

5.1.1 Input screen

The order can be entered through the input screen (Figure 77). By clicking on the input screen a node is automatically created, the node can be removed by clicking on the node using the right button of the mouse. The edge between two nodes can be created by clicking once on both nodes, the edge can be remove by clicking on the circle on the middle of the edge. In Figure 77 a grid graph is entered as an ordered set.

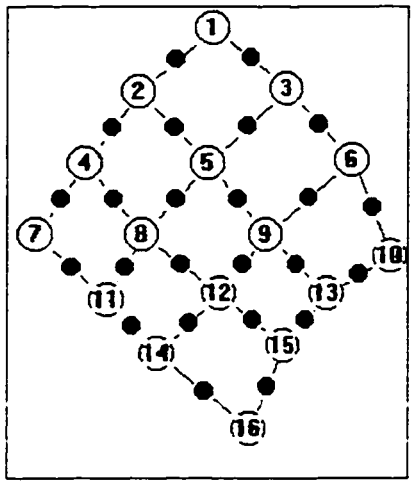


Figure 77. Input Screen

5.1.2 Output screen

The output screen is used only for outputting the ordered set, the order can be visualised by using the control box options. The zoom buttons **Zoom +** and **Zoom -** are used for

zooming the order for better view, also it can be rotated around the x, y, z, xy, xz, yz and xyz axis. The rotation is performed by pressing on the corresponding axis in the rotation icon (Figure 79(a)).

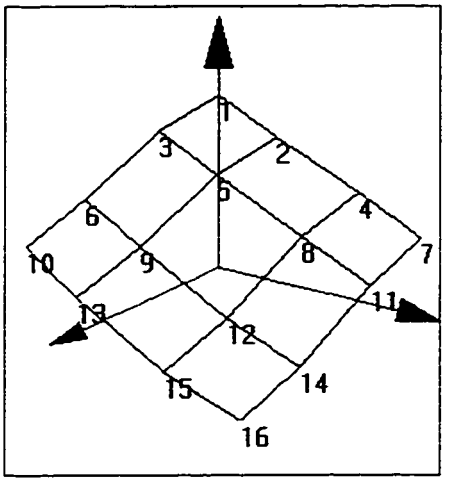


Figure 78. Output screen

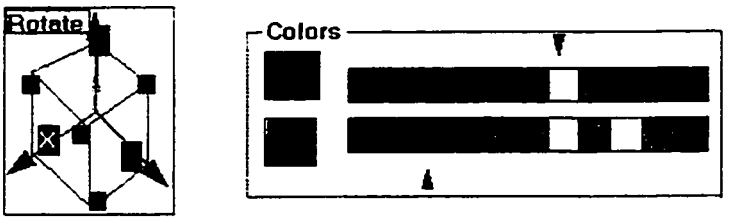


Figure 79. (a) Rotation icon (b) Colour icon

5.2 Using the Tool

In this section we provide an example on how to use the tool for inputting the **Grid 16** ordered set. The steps are as follows:

Step 1 : create 16 nodes on the ***Input Screen*** by clicking 16 times using the left button of the mouse.

Step 2 : input the covering relation between nodes. To have node 1 cover nodes 2 and 3 first place node 1 above nodes 2 and 3, then draw edges from 1 to 2 and from 1 to 3.

Step 3 : press ***Show order*** button, the order will be displayed randomly on the output screen

Step 4 : press ***Order*** button, the *Force-Directed Placement* algorithm will be applied to the order and the output screen shows the development of the order.

Step 5 : press ***Pause*** button to stop applying the algorithm on the order.

The constants ($C_{repulsive}$, K_{spring}, L_{spring}, K_{sphere} and L_{sphere}) may be changed by the user either when the algorithm is running or when the order is in the pause state. The rotation icon (Figure 79(a)) is active only when the order is in the pause state. The user may chose to rotate the order around the x, y, z, xy, xz, yz or xyz axis.

The **Zoom+** and **Zoom-** buttons are active when the algorithm is running or when the order is in the pause state. They are used to enlarge and reduce the view of the order in the output screen.

Figure 80 shows the development of the ***Grid 16*** order. Figure 80(a) shows the order in initial random positon, Figure 80(f) shows the final drawing after 25 iterations. Figures 80(b) to 80(e) are intermediate states.

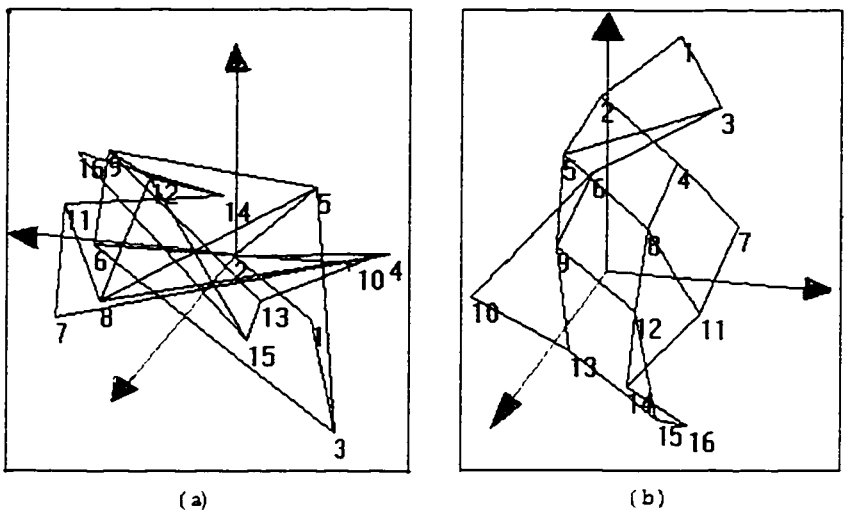


Figure 80 . Grid 16 in development stages

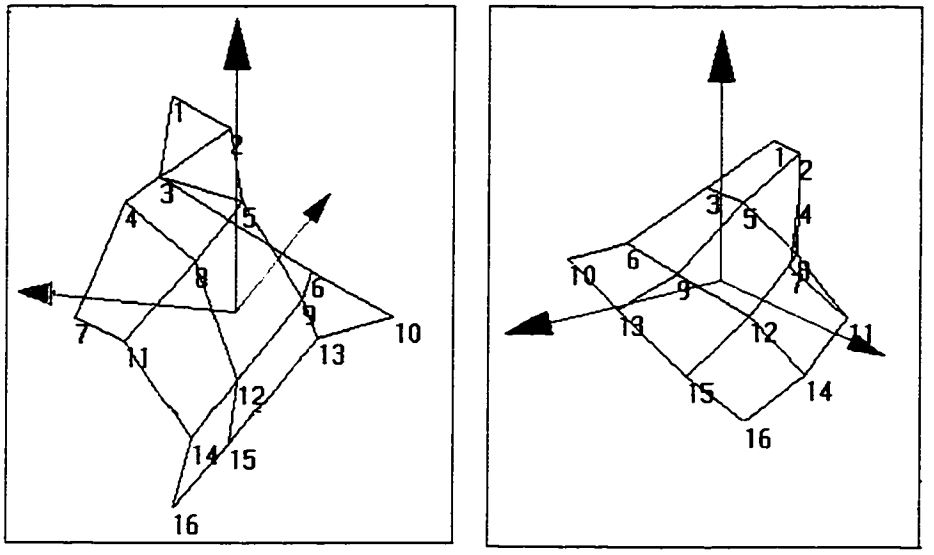


Figure 80 . (continued) Grid 16 in development stages

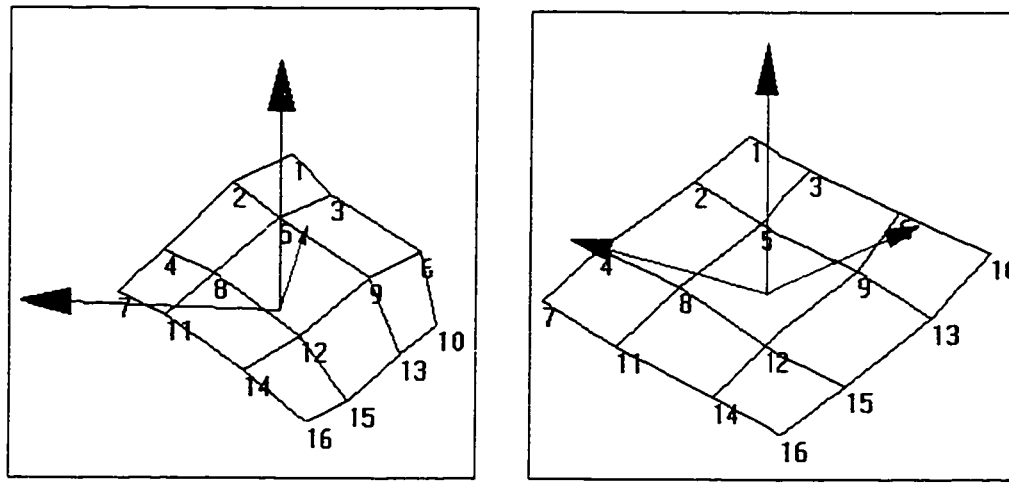


Figure 80 . (continued) Grid 16 in development stages

Chapter 6

Summary and Future Work

6.1 Summary

In this thesis we introduced algorithms to draw and visualise ordered sets using *Force-Directed Placement* algorithm. Most of the Force-Directed Placement algorithms deal with graph drawing . We implemented a tool that draws orders in 3D and the user is able to manipulate the input and the output in a natural and user friendly fashion.
The complexity of our algorithm can be summarised as follows: the individual iteration has a time complexity of $\Theta(|V|^2+|E|)$ for the basic algorithm(i.e. to calculate the attractive and repulsive forces) and $\Theta(|V|+|E|)$ for centralising our graph in the middle of the screen. The spherical drawing adds no time complexity to the algorithm, but another force called f_s (force of the sphere) is added to our set of forces. The number of iterations needed varies according to the initial layout of the graph, but experimentally it was found that the number of iterations is a function of number of edges and number of vertices.

6.2 Future work

Although the *Force-Directed Placement* algorithm gives pleasing results in many cases and it deterministically converges to a minimum energy state, its theoretical behaviour is not clear. Experimentally the time complexity is $O(|V|^3)$, but we cannot give a formal proof.

Another interesting problem for future work is applying our triangulation algorithm with other *Force and Energy Controlled Placement* algorithms and compare the results in terms of quality of output and speed.

References

[BeHo87] Becker, B. and Hotz, G., On optimal layout of planar graphs with fixed boundary, SIAM J. Comput. 16(1987), 946-972.

[BoPr91] Boehm, W., Prautzsch, H. Numerical Methods, A.K. Peters, Vieweg, 1991.

[DaHa89] Davidson R. and D. Harel, "Drawing Graphs Nicely Using Simulated Annealing", Technical report, The Weizmann Institute of Science, Rehovot, Israel, 1989.

[DaHa91] Davidson R. and D. Harel, "Drawing Graphs Nicely Using Simulated Annealing", Technical report, The Weizmann Institute of Science, Rehovot, Israel, (revised revision) 1991.

[Ea84] Eades, .P: A heuristic for Graph Drawing, Congressus Numerantium, 42, 149-160 (1984)

[FLM95] Frick, A. ; Ludwig, A. ; Mehldau, H. : A Fast Adaptive Layout Algorithm for Undirected Graphs, In proceedings of Graph Drawing 94, volume 894 of LNCS, pages 388-403. Springer, 1994.

[FrRe90] Fruchterman, T.M.J; Reingold, E.M. : Graph Drawing by Force Directed Placement, Technical report, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, 1990.

[FrRe91] Fruchterman, T.M.J; Reingold, E.M. : Graph Drawing by Force Directed Placement, Software-Practice and Experience 21, pp.1129-1164, 1991

[GrTu87] Gross, J.L., and Tuker, T.W., Topological graph theory, Awiley-Interscience Publication, John Wiley & Sons, New York, 1987.

[Ha96] Hashemi, M.S. :"ON THE SURFACE GEOMETRY OF ORDERED SETS ", A Ph.D. Thesis for department of Mathematics and Statistics, University of Ottawa, Ottawa, Canada, 1969.

[Ha85] Hajek, B., "A Tutorial Survey of Theory and Applications of Simulated Annealing", Proc. 24th Conf. Decis. Cont., 1985, pp. 775-760.

[HaKiRi95] Hashemi, M.S. ; Kisielewicz, A; Rival, I. : Upward Drawing on Planes and Spheres, ORDER (to appear).

[HaRe88] Halliday, D.; Resnick, R. :Fundamentals of Physics, third edition, John Wiley and Sons, 1988

[HaRi90] Harstfield, N. ; Ringel, G. : Pearls in Graph Theory, A comprehensive Introduction, Academic Press, Inc. 1990.

[HaSa93] Harel, D., Sardas, M. "Randomized Graph Drawing with Heavy-Duty Preprocessing ", Technical report, The Weizmann Institute of Science, Rehovot, Israel, 1993.

[HeBa86] D. Hearn and M.P. Baker, Computer Graphics, Printice-Hall, Englewood Cliffs, NJ, 1986.

[JASM87] Johnson, D.S., C. R. Aragon, L. A. McGeoch and C. Schevon, "Optimization by simulated annealing: An Experimental Evaluation, Part I (Graph Partitioning)", Oper. Res., 37 (1989), 865-892.

[KaKa89] Kamada, T.; Kawai, S. : An Algorithm for Drawing General Undirected Graphs, Information Processing Letters, 31, pp. 7-15, 1989.

[LA87] Laarhoven, P.J.M., and Aarts, E.H.L., Simulated Annealing: Theory and applications, D. Reidel Publishing Co., Dordrecht, 1987.

[MRRTT53] Metropolis, N.A. Rosenbluth, M. Rosenbluth, A,Teller and E.Teller " Equation of state Calculations by Fast Computing Machines", J. Chem, phys. 21 (1953), 1087, 1091

[QuBr79] Quinn, Jr. ,N.R; Breur, M.A. : A force directed placement procedure for printed circuit boards, IEEE Transaction on circuits and systems, CAS-26, (6), 377-388 (1979)

[Ri96] Rival, I. : Lectures on Ordered Sets, University of Ottawa, Ottawa, Canada, 1996 (*http:www.csi.uottawa.ca:80 dept algorithms order*)

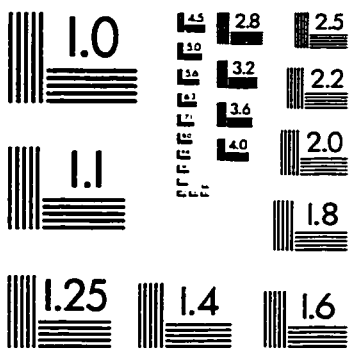
[Ri89] Rival, I. : Algorithms and Order, Kluwer Academic Publishers, pp. 3-11, 1989

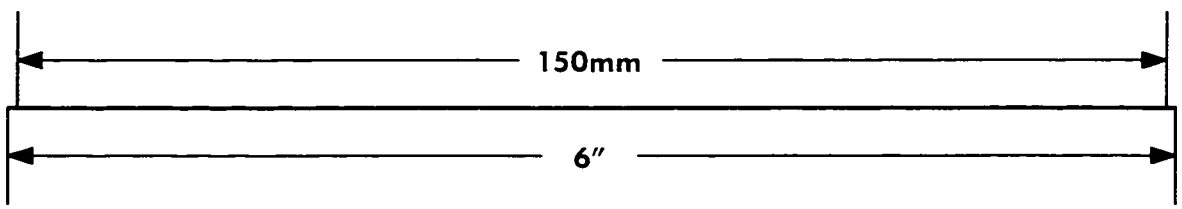
[Sa96] Sander, G.: Graph Layout for Applications in Compiler Construction, Research Report, FB 14 Informatik, University of Saarlandes, Germany, 1996

[SuMi94] Sugiyama, K.; Misue, K. : Graph Drawing by Magnetic-Spring Model, Research Report ISIS-RR-94-14E, Inst. Social Information Science, Fujitsu Labs. Ltd., 1994.

[SuMi95] Sugiyama, K.; Misue, K.: A Simple and Unified Method for Drawing Graphs: Magnetic-Spring Algorithm, Lecture Notes in Computer Science, pp. 364-375, 1995

TEST TARGET (QA-3)





APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989