Chapter 2
Discrete Spaces: Graphs, Lattices, and Digital Spaces

Abstract The best way to describe discrete objects is to use graphs. A graph consists of vertices and edges. The vertex usually represents a part of an object, an whole object, or the location of an object; the edge represents a relationship between two vertices. A graph can be defined as $G = (V, E)$ where V is a set of vertices and E is a set of edges, each of which links two vertices. For a certain geometric object, e.g. a rectangle, one can draw four points on the corners and link them using four edges. The drawback of using edges is that the edge is not a geometric line and it usually does not carry a distance. A geometric space also requires a measurement of distance (the length between two points), called a metric. Therefore, people prefer to use specialized graphs such as triangulated graphs and grid graphs, to represent an object in discrete space. In this chapter, we introduce the discrete spaces made by graphs, lattices, and grid points. We briefly review some of the basic concepts related to discrete objects and discrete spaces.

Keywords Graph · Lattice · Space · Digital space · Discrete space · Algorithms

2.1 Objects in Discrete Spaces

Objects are things that are visible or tangible. An object usually has a form that is relatively stable. Geometry is the study of objects and their properties in space. Due to the fact that computers can only take a discrete or finite number of objects, discrete geometry has become more important in recent years.

There are always two ways to view an object: (1) A contiguous interpolation of discrete points to from a continuous object, (2) A discretizing or sampling of a continuous object to get its discrete representation. In this chapter, we assume that our space is discrete. In the next chapter, we discuss spaces and objects that are continuous.

The recent motivation behind studies in discrete geometry is due to the need of computer graphics and computer vision. There are a great deal of applications, especially in medical imaging, that require a high level of math including topology in a digital format. Images are stored in digital spaces, discrete spaces, and even with finite topologies.

The simplest way to describe a discrete object is to use graphs. A graph consists of vertices and edges. The vertex usually represents a part of an object, a whole object, or the location of an object; the edge represents a relationship between two vertices. For instance, a rectangle can be represented by drawing four points on the corners and linking them using four edges.

In this chapter, we start with an introduction of an undirected graph $G = (V, E)$, where V denotes the set of vertices and E denotes the edges between the vertices. Then we introduce the lattice that has geometrically regular assigned locations for each vertex. The edge in lattices is typically assumed. Note, the lattice we deal with here differs from an algebraic lattice, which has a partial order assigned on each vertex.

Then, we focus on the grid space that is the simplest lattice, similar to arrays in computers. This grid space is called digital spaces.

Another popular discrete space is called the triangulated space in two-dimensional spaces (2D). This space contains only triangles as 2D elements. In mathematics, a triangle is called a two-dimensional simplex (2-simplex), while a point is called a 0-simplex and a line-segment is called 1-simplex.

In undirected graphs, we can view that vertices are 0-simplices, and edges are 1-simplices. However, to describe 2-simplices, one needs special structures to define it. On the other hand, edges in a graph cannot just be viewed as line-segments because edges in graphs can have more general meanings.

2.2 Graphs and Simple Graphs

A graph G consists of two sets V and E, where V is the set of vertices and E is the set of pairs of vertices called edges. An edge is said to be incident to the vertices if it joins [2, 11, 16]. In this book, assume that $G = (V, E)$ is an undirected graph, which means that if $(a, b) \in E$ then $(b, a) \in E$, or $(a, b) = (b, a)$. a, b are also called ends, endpoints, or end vertices of edge (a, b). It is possible for a vertex in a graph to not belong to any edge. V and E are usually finite, and the order of a graph is $|V|$, which is the number of vertices. The size of a graph is linear to $\max\{|V|, |E|\}$, meaning that it requires this much memory to store the graph.

The degree of a vertex is the number of edges that incident with (or link to) it. A loop is an edge that links to the same vertex. In such a case, the degree would be counted twice. Figure 2.1 shows two examples of graphs. Figure 2.1a shows a directed graph, where the edge has an arrow. Figure 2.1b shows an undirected graph.

2.2.1 Basic Concepts of Graphs

Graph $G = (V, E)$ is called a simple graph if every pair of vertices has at most one edge that is incident to these two vertices and there is no loop $(a, a) \in E$ for any $a \in V$. See Fig. 2.2.

If (p, q) is in E, then p is said to be adjacent to q. Let $p_0, p_1, ..., p_{n-1}, p_n$ be $n+1$ vertices in V. If (p_{i-1}, p_i) is in E for all $i = 1, ..., n$, then $\{p_0, p_1, ..., p_{n-1}, p_n\}$ is

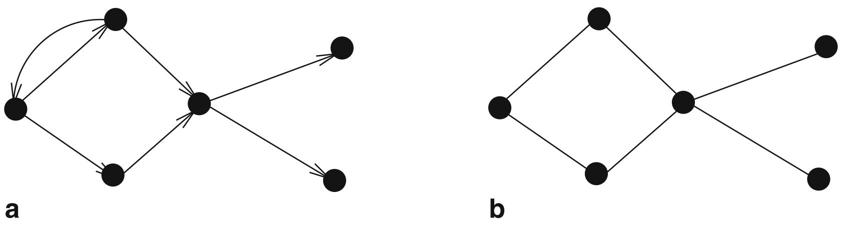
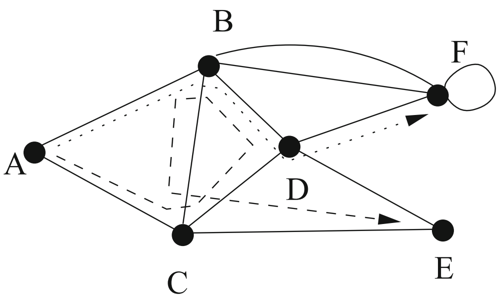


Fig. 2.1 Example of graphs: **a** A directed graph, and **b** An undirected graph

Fig. 2.2 Paths: A path $\{A, C, D, B, C, E\}$; A simple path $\{A, B, D, F\}$; and A loop at F



called a path. If $p_0, p_1, ..., p_{n-1}, p_n$ are distinct vertices, the path is called a simple path.

A simple path $\{p_0, p_1, ..., p_{n-1}, p_n\}$ is closed if (p_0, p_n) is an edge in E. A closed path is also called a cycle. Two vertices p and q are connected if there is a path $\{p_0, p_1, ..., p_{n-1}, p_n\}$ where $p_0 = p$ and $p_n = q$. G is called connected if every pair of vertices in G is connected. In this book, it is always assumed that G is connected (unless otherwise specified).

Let S be a set. If S' is a subset of S, then their relationship is denoted by $S' \subseteq S$. If S is not a subset of S', then S' is called a proper-subset of S, denoted by $S' \subset S$.

Suppose $G' = (V', E')$ is a graph where $V' \subseteq V$ and $E' \subseteq E$ for graph $G = (V, E)$. Then, $G' = (V', E')$ is called a subgraph of G.

If E' consists of all edges in G whose joining vertices are in V', then the subgraph $G' = (V', E')$ is called a partial-graph of G and their relationship is denoted by $G' \preceq G$. If V' is a proper-subset of V, then the relationship is denoted by $G' \prec G$. It is noted that for a certain subset V' of V, the partial-graph G' with vertices V' is uniquely defined. [1] A path is a subgraph. For example in Fig. 2.2, $\{A, B, D, C, E\}$ is a path. Let $V' = \{A, B, D, C, E\}$ and $E' = \{(A, B), (B, D), (D, C), (C, E)\}$. $G' = (V', E')$ is a subgraph, but it is not a partial graph because it does not include the edge (D, E). So, $G'' = (V', E' \cup \{(D, E)\})$ is a partial graph.

[1] In [6], we made an error in revising the meanings of the two concepts: Subgraphs and Partial graphs.

2.2.2 Special Graphs

In this subsection, we present some examples of special graphs that may be used later in this book. The detailed description for these concepts can also be found in [2, 11].

Complete Graph In a complete graph, each pair of vertices is joined by an edge. A triangle is a complete graph with three vertices. A complete graph with five vertices contains 10 edges. See Fig. 2.3a. A complete graph with n vertices is denoted as K_n.

Bipartite Graph In a bipartite graph, the vertices can be divided into two sets, X and Y, so that every edge has one vertex in each of the two sets, i.e. no edge is inside set X (or Y) alone. See Fig. 2.3b. A Bipartite graph with n vertices in X and m vertices in Y is denoted as $K_{n,m}$.

Weighted Graph In a weighted graph, each edge can be assigned a weight. The weights are usually real numbers that could indicate distance if the vertices are cities. See Fig. 2.3c.

Planar Graph A planar graph is a graph that can be drawn in a plane with no crossing edges. A graph with crossing edges may or may not be a planar (plane-able) graph. In the following subsection, we present a theorem related to planar graphs. See Fig. 2.3d.

Tree A tree is a connected graph with no cycles. A forest is a graph with no cycles. See Fig. 2.3e.

2.3 Basic Topics and Results in Graph Theory

Graph theory was established by Euler, who solved the well-known —Seven Bridges of Konigsberg problem in his time. See Fig. 2.4a. In this map, Euler was asked to draw a path where each bridge would be traveled just once. Euler found that it was impossible, since he represented this problem as a 4 vertices and 7 edges graph, Fig. 2.4b. When one passes a vertex, he must go through two edges, one in and one out. That is to say, if such a path exists, each vertex must contain even number of edges. This property became the first theorem in graph theory.

Two of the most famous problems in graph theory are the Four Color Problem and the Traveling Salesman Problem. The former deals with a well-known “rule” in map printing. Only four colors are needed in a map, a planar graph, in which all adjacent points are colored by different colors. It is believed that this problem was solved positively by a computer program. However, even now, no one is able to verify the correctness of the computer program [2].

As for the Traveling Salesman Problem, its goal is to find the shortest path (for n cities) when a salesperson only travels to each city once. The path finding procedure exists but any of these solutions require extremely long computation time to complete

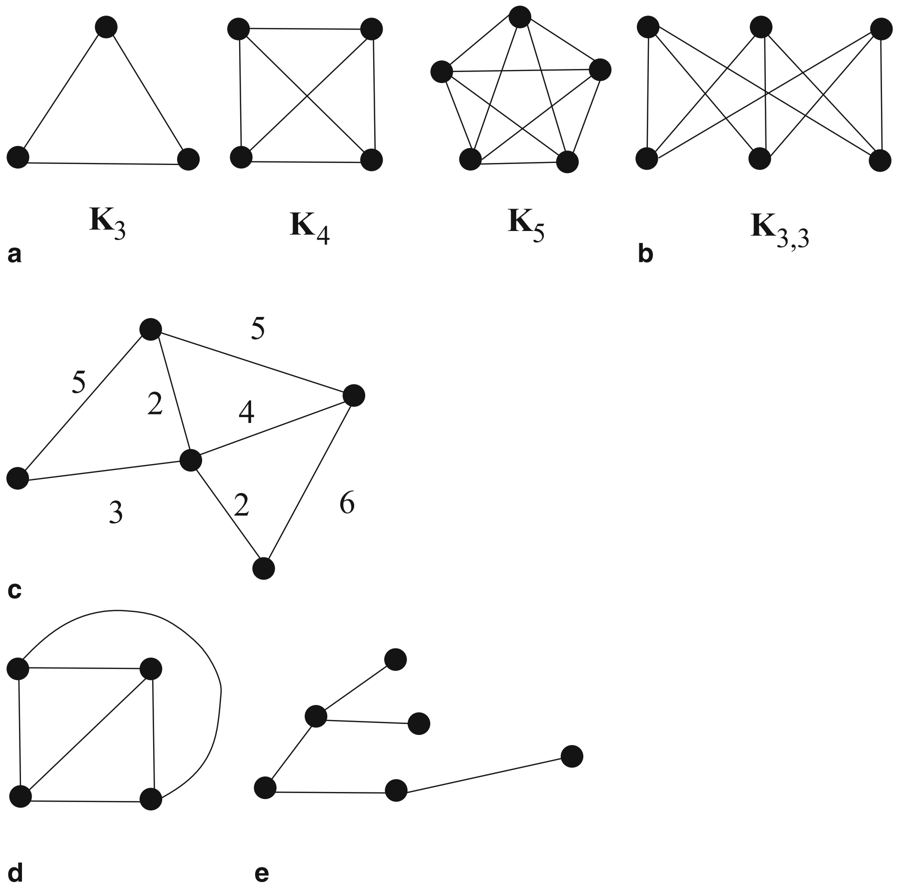


Fig. 2.3 Some special graphs: **a** A complete graph K_3, K_4, K_5, **b** A bipartite graph $K_{3,3}$, **c** A weighted graph, **d** A planar graph K_4, and **e** A tree

the job. Today, no one knows if there is an efficient algorithm to solve this problem. It is related to the $P = ?NP$ problem [8].

2.3.1 Graph Representation, Searching Graph, and Graph Coloring

To solve a complex problem in graphs, for example the graph contains hundreds or even thousands of vertices, we must rely on computers. But how do we represent a certain graph in the computer? There are two ways to do this: the adjacency matrix and the adjacency list.

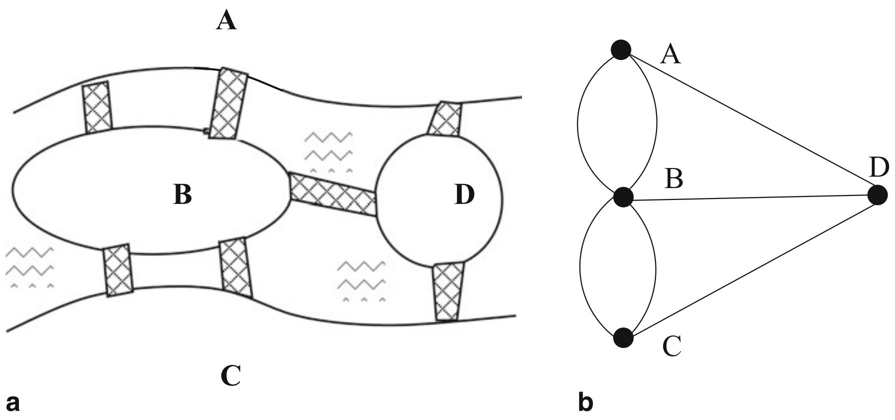


Fig. 2.4 Seven Bridges of Konigsberg: **a** An original map, and **b** The equivalent representation in graphs

In the adjacency matrix, we assume Graph G has n vertices. An $n \times n$ $\{0, 1\}$ matrix $M = \{m_{ij}\}$ is used to hold the information of adjacency: $m_{ij} = 1$ if and only if vertex i and j are adjacent. This is the simplest way, but if there are not many edges in G, then there needs to be a large amount of storage space to represent matrix M. For instance, if $n = 100$, we need 10,000 memory units to represent M.

Another way that could save storage space is called the adjacency list. For vertex v in G, we just attach all adjacent points of v to v. So each link becomes a linked list that is lead by v.

$$v_i \to u_{i1} \to u_{i2} \cdots \to u_{ik}$$

v_i represents the ith vertex. The length of each list is not the same, so we can save a lot of space. However, the process of calculating may become a little more difficult.

To find whether a graph G is connected, we need to search the graph. The best way is called the depth-first search or breadth-first search technique. In mathematics and computer science, if one needs a procedure to solve a problem, the procedure is called an algorithm. An algorithm usually contains several steps of instructions that solves a problem. The basic idea of the depth first search algorithm is to find the set of connected vertices until no more vertices can be found. Then, we go back to each vertex we visited to see if there are any other ways to go. We continue, where possible, until there are no possible ways left. This procedure requires a special data structure to hold the vertices we visited. We discuss this further in Chaps. 4–6.

Graph coloring assigns different colors to adjacent vertices. It usually tries to assign the minimum number of colors. The most famous problem is called the four color problem for planar graphs, as we have mentioned before. This problem was believed to be solved in 1975 with the help of computer programs. Since some errors were found and fixed in the program, some mathematicians are still looking for pure,

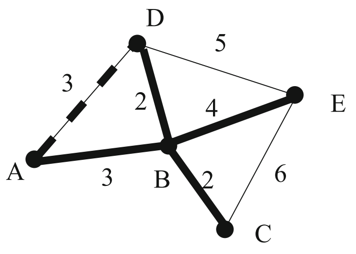


Fig. 2.5 Find a minimum spanning tree T

mathematical proofs. However, a famous theorem states: Five colors are enough for a planar graph [11].

2.3.2 *The Minimum Spanning Tree*

Given a connected graph $G = (V, E)$, a spanning tree is a subgraph of G, which is a tree and contains all the vertices of G. This special tree is called a spanning tree since it spans every vertex.

A graph may have several different spanning trees. If G is a weighted graph, a minimum spanning tree (MST) is the one that has the minimum total weight. The simplest method for finding a MST is called Kruskal's Algorithm. Its principles are as follows (Fig. 2.5):

Algorithm 2.1: Kruskal's Algorithm. Find a minimum spanning tree T for graph G.

```
Step 1  Sort the edges based on the weights of the edges from smallest to largest.
Step 2  Set initial tree $T$ to be empty.
Step 3  Select an edge from the sorted edge list and add it to $T$ if such an added
        edge does not generate a cycle.
Step 4  $T$ would be the minimal spanning tree if $|V| - 1$ edges are added to $T$.
```

The proof of Algorithm 2.1 can be found in [8].

2.3.3 *The Shortest Path**

Finding the shortest path for each pair of vertices in a graph is one of the most common problems in the real world. A method of finding the shortest paths from a single source vertex to all of the other vertices in a weighted directed graph is called the Bellman–Ford algorithm.

Let $G = (V, E)$ and $|V| = n$. We use W(e)=W(u,v) to represent the weight on an edge $e = (u, v)$. The principle of the algorithm is to reach a vertex v using k edges from the source vertex S and maintain the shortest path using at most k edges. In

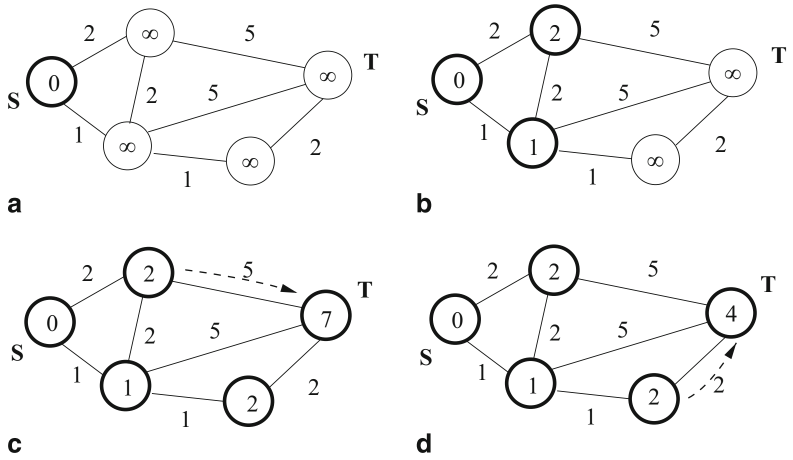


Fig. 2.6 Bellman-Ford Algorithm for finding the shortest paths: **a** Original graph, **b** Move to the direct neighbors and no change in relaxation, **c** Move to the next neighbors, and **d** Value changed in relaxation

other words, from S, if we use one edge, we can only get to the neighboring cities. If we use two edges, we can get to the neighboring cities of the neighboring cities. Then, we update the shortest distance on all vertices we can reach on the path with at most two edges. Continuing this idea, we can reach all vertices by using at most $n - 1$ edges and we can finish our task.

In the detailed algorithm, we always mark or record the distance from the source to every other vertex at the vertex. It is obvious that we always mark the smallest value (the shortest one, by using k edges). When we have used $(n - 1)$ edges, we would have the solution.

Algorithm 2.2 : (The Bellman–Ford Algorithm) Find the shortest paths from a vertex S to all vertices in graph G (Fig. 2.6).

Step 1 Mark all vertices other than S as ∞, where $d(v)$ is the distance from S to v and $d(S) = 0$.

Step 2 For each vertex, follow an iterative procedure called relaxation: for each v where u is adjacent to v, check if $d(v) > d(u) + W(u, v)$. If so, then $d(v) = d(u) + W(u, v)$. Repeat Step 2 for $(n - 1)$ times.

Another algorithm is called Dijkstra's algorithm. Dijkstra's algorithm runs faster than the Bellman–Ford algorithm, but Dijkstra's algorithm is unable to deal with graphs that have some negative weight edges. The Bellman–Ford algorithm is also easier to understand. It checks all possible links $n - 1$ times. During each iteration, we get the shortest path passing through k edges. Until we pass $n - 1$ edges, we naturally get the shortest path from S. The idea behind this algorithm is called dynamic programming, which means dynamically using the results already calculated.

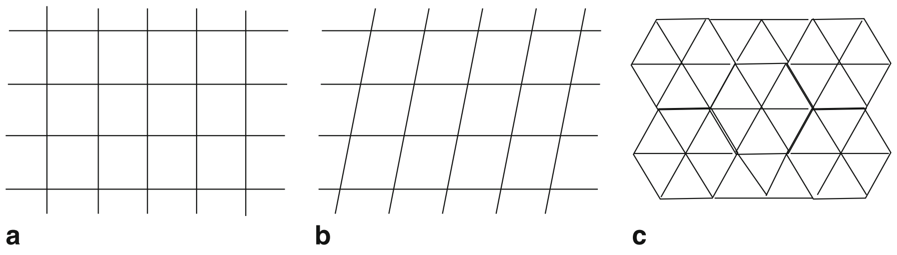


Fig. 2.7 Lattice graphs: **a** Squares, **b** Parallelograms, and **c** Triangles

*2.3.4 Graph Homomorphism and Graph Isomorphism**

Homomorphism and isomorphism are related to functions between to two graphs. A graph homomorphic mapping is a function from the vertex set of G to the vertex set of G' so that if (a, b) is an edge of G, then $(f(a), f(b))$ is an edge of G'. If such a mapping exists, we say that G, G' are homomorphic.

If f is a 1-to-1 mapping (bijection), then f is called isomorphic. Homomorphism is called an edge-preserving mapping whereas isomorphism is an edge-preserving bijection.

Graph homomorphism has some applications to graph coloring problems. However, graph isomorphism mainly describes two graphs having "the same structure."

When the preserving edge can be allowed to shrink into a vertex, the function/mapping is called immersion and embedding. Graph immersion is an important concept to discrete surface reconstruction in this book. See Chap. 11.

A famous unsolved problem in computer science states: Given two graphs with the same number of vertices, is there a polynomial time algorithm to decide if these two graphs are isomorphic? This problem is called the graph isomorphism problem [1, 5].

2.4 Lattice Graphs, Triangulated Space, and Grid Space

To view a general graph as a discrete space is sometime too general. There are much more specific graphs that are better for the purpose of defining discrete spaces. These are called lattice graphs. A lattice graph is a simple graph with a distance measurement (called metric) of a geometric object. Lattice graphs are regular graphs, meaning that each edge has the same weight or represents the same distance in Euclidean space as in other spaces.

For instance, in regular triangles, each edge has the same length as in hexagons and parallelograms. (See Fig. 2.7) A more general 2D shape is called a polygon, which is formed by multiple edges as a closed boundary. In Chap. 5, we show more examples of these shapes.

A metric used to measure distances is dependent on geometric objects. For instance, the metrics on the plane and the sphere are different. The weight of an edge usually means the length of the shortest path between the two lattice points. In other words, the edge is usually the minimum distance curve in the space, called the geodesic curves between two adjacent lattice points. In computer graphics, we call lattices the meshes. Therefore, meshes on surfaces are good examples of lattices.

Two of the most popular lattice graphs are the triangulated space and the grid space. See Fig. 2.7b and c. Since the triangle is the simplest shape containing 2D information. It is called a 2D simplex. Adding another point in a new dimension, we will have a 3D simplex that contains four end points. Any geometric shape can be partitioned using simplexes in general. So mathematicians treat the simplex as the most significant discretization unit. However, this is difficult to represent in computers since the computer memory or disk storage are arranged as arrays, which is similar to grid spaces.

A grid space is a special lattice in which each point is at the integer coordinate location in Euclidean space and the edges are usually parallel to the coordinate lines. In other words, a grid graph is a graph whose vertices correspond to the points with integer coordinates. For instance, in a 2D plane, x-coordinates are in the range 1,...,n, and y-coordinates are in the range 1,...,n. A grid space is similar to a TV screen or a mathematical 2D array.

The grid space is also called the grid-cell space, which is the main topic of concern in this book—digital space. However, digital space has more inner-meanings than a grid graph, and we discuss it in further details in the following sections of this chapter.

In summary, we can usually view a lattice as a graph embedded in Euclidean space. Lattices only contain points and edges, where edges are usually straight lines. Examples, including meshes for computer graphics, are very popular. The lattice graph differs from the algebraic lattice, which is defined on a partially ordered set where any two elements have a supremum and infimum in the algebraic lattice [15].

2.5 Basic Concepts of Digital Spaces

Digital space has two definitions. First, in the narrow sense, a digital space is a discrete space in which each point can be defined as an integer vector, i.e. each component of the vector is an integer. Second, in the general sense, the space is a digitized space or discretely sampled space that is saved in digital form. In this book, we usually reference the first definition of grid space when discussing digital space.

2.5.1 2D and 3D Digital Spaces

Let us consider a two-dimensional digital space Σ_2. It contains all integer points of a Euclidean plane, $\mathbf{E}_2$. A point P (x, y) in Σ_2 has two horizontal $(x, y \pm 1)$ and two