

## Chapter 3

# Matrices and Matrix Operations

### 3.1 Matrices and Matrix Modeling

Matrices are used to model phenomenon requiring multiple dimensional data. Applications areas exist in almost every branch of physics including classical mechanics, statics, dynamics, electromagnetism, optics, and quantum mechanics. We also use matrices in computer graphics to create 2D and 3D models. In this section, we focus on some of the most accessible matrix modeling techniques. These will help us build intuition on how to construct matrices in later sections when we increase the complexity of our models.

#### Definition 3.1: Entry-by-entry definition of an $m \times n$ matrix

Let  $m, n \in \mathbb{N}$ . An  $m \times n$  **matrix**  $A$  is a rectangular array of real numbers with  $m$  rows and  $n$  columns. We can write the general structure of an  $m \times n$  matrix  $A$  as:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Each of the numbers in this array is called an **entry**, **element** or **coefficient** of the matrix. Each entry has a row and column index, which identifies where in the matrix that coefficient is stored.

Notice from our definition above, that all vectors are matrices but not all matrices are vectors.

#### Definition 3.2: Dimensions of a Matrix

The **size** of a matrix  $A$  is given in the form  $m \times n$ , where  $m$  is the number of rows and  $n$  gives the number of columns of the matrix. We call  $m$ , the number listed to the left of the  $\times$  symbol, the **row dimension** of the matrix. On the other hand, we call  $n$ , the number listed to the right of the  $\times$  symbol, the **column dimension** of the matrix.

**EXAMPLE 3.1.1**

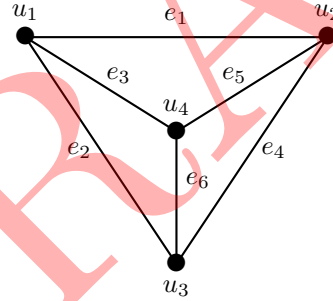
An **undirected graph**  $\mathcal{G}$  consists of two finite sets  $\mathcal{N}$  and  $\mathcal{E}$ . The set  $\mathcal{N}$  includes a list of objects known as **nodes** or **vertices**. The set  $\mathcal{E}$  contains a list of **undirected edges**. Each edge specifies connectivity relationships between vertices of our graph as a subset of  $\mathcal{N}$ . Let's create an undirected graph with four nodes and six undirected edges given by

$$\mathcal{N} = \{u_1, u_2, u_3, u_4\} \quad \text{and} \quad \mathcal{E} = \left\{ \{u_1, u_2\}, \{u_1, u_3\}, \{u_1, u_4\}, \{u_2, u_3\}, \{u_2, u_4\}, \{u_3, u_4\} \right\}.$$

We assume each element of  $\mathcal{E}$  is listed once for each intended edge on the graph. To enumerate the edges, set the first listed element of  $\mathcal{E}$  to be  $e_1 = \{u_1, u_2\}$  and define the second listed element as  $e_2 = \{u_1, u_3\}$ . Continue until the end of the list of edges, concluding with  $e_6 = \{u_3, u_4\}$ .

The vertices  $u_j$  and  $u_k$  associated with edge  $e_i = \{u_j, u_k\}$  are known as the **end vertices** of edge  $e_i$ . For example, the end vertices of edge  $e_3 = \{u_1, u_4\}$  are nodes  $u_1$  and  $u_4$ . In this definition, we do not differentiate between the listed order for the end vertices. In particular,  $e_3 = \{u_1, u_4\} = \{u_4, u_1\}$ , with no distinction between which node is at the “start” of this edge and which node is at the “end” of the edge. Because each edge determines only connectivity between nodes and does not specify a direction for the connection, we call set  $\mathcal{G} = (\mathcal{N}, \mathcal{E})$  an undirected graph.

A very convenient way to represent the set-theoretic definition of undirected graphs is to use a **graph diagram**. To do so, we represent the vertices of the graph as dots and draw each edge as a line segment connecting its end vertices. For the graph defined in this example, our corresponding graph diagram is given below.



We can use this diagram to create an **undirected incidence matrix**  $A \in \mathbb{R}^{6 \times 4}$  that models the connectivity between vertices. In this case, matrix  $A$  has six rows and four columns. Row  $i$  of our matrix corresponds to edge  $i$  while column  $k$  of our matrix corresponds to vertex  $k$  for  $i = 1, 2, 3, 4, 5, 6$  and  $k = 1, 2, 3, 4$ . The entry-by-entry definition of  $A$  is given by

$$a_{ik} = \begin{cases} 1 & \text{if edge } e_i \text{ touches node } n_k, \\ 0 & \text{otherwise.} \end{cases}$$

To create this undirected incidence matrix, we set up the table which we use to read of the entries of the incidence matrix.

Undirected Incidence  
Matrix Table

	$u_1$	$u_2$	$u_3$	$u_4$
$e_1$	1	1	0	0
$e_2$	1	0	1	0
$e_3$	1	0	0	1
$e_4$	0	1	1	0
$e_5$	0	1	0	1
$e_6$	0	0	1	1

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

This matrix  $A$  fully encodes all relevant aspects of our undirected graph including the number of vertices, the number of edges, and the interconnection between each vertex. Also implicit in this matrix definition is our desired enumeration scheme where row one of the matrix corresponds to edge one of the graph. We will study techniques to ascertain many useful properties about a given undirected graph by analyzing the corresponding incidence matrix.

Undirected graphs are very useful when analyzing problems involving general connectivity between distinct objects. If we notice that such a problem does not depend on assigning a direction to component interconnects, we can use undirected graphs as a tool to encode system-wide connectivity. However, there are a number of modeling contexts in which we may want to assign a direction to each edge. For this purpose, we can create directed graphs.

#### EXAMPLE 3.1.2

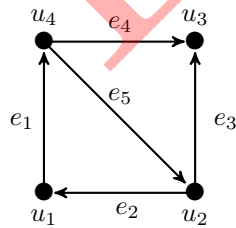
A **directed graph** consists of two sets  $\mathcal{N}$  and  $\mathcal{E}$ . The set  $\mathcal{N}$  contains a finite number of objects, known as **vertices** or **nodes**. The set  $\mathcal{E}$  contains a finite list of **directed edges**. Each directed edge is an ordered pair in  $\mathcal{N} \times \mathcal{N}$ . Let's create a directed graph with four vertices and five edges given by

$$\mathcal{N} = \{u_1, u_2, u_3, u_4\} \quad \text{and} \quad \mathcal{E} = \{(u_1, u_4), (u_2, u_1), (u_2, u_3), (u_4, u_3), (u_4, u_2)\}.$$

Again, we assume each element of  $\mathcal{E}$  is listed once for each intended edge on our graph. We enumerate our edges by assigning indices in the order in which the edges are listed. In this case, we have  $e_1 = (1, 4)$ ,  $e_2 = (2, 1)$  all the way through  $e_5 = (4, 2)$ .

If  $e_i = (u_j, u_k) \in \mathcal{E}$ , we conclude that edge  $i$  is **incident out of** vertex  $u_j$  and **incident into** vertex  $u_k$ . In this case, we call  $u_j$  the **initial vertex** of edge  $e_i$  and while  $u_k$  is the **terminal vertex** of this edge. For example, we see that edge  $e_4 = (u_4, u_3)$  has initial vertex  $u_4$  and terminal vertex  $u_3$ .

Just like with undirected graphs, we can visualize the connectivity between nodes of a directed graph using a graph diagram.



To draw a directed edge, we draw an arrow pointing out of the initial node and pointing into the terminal node. We can then construct a *directed incidence matrix*

to model the connectivity of the vertices. The individual coefficients of the incidence matrix  $A$  are given by

$$a_{ik} = \begin{cases} 1 & \text{if edge } e_i \text{ leaves node } u_k, \\ -1 & \text{if edge } e_i \text{ enters node } u_k, \\ 0 & \text{otherwise.} \end{cases}$$

We let the rows of this matrix represent the edges and the columns represent the nodes of our digraph. For this directed graph with 5 edges and 4 nodes, we create the table which we use to read of the individual entries of our  $5 \times 4$  incidence matrix.

Directed Incidence  
Matrix Table

	$u_1$	$u_2$	$u_3$	$u_4$
$e_1$	1	0	0	-1
$e_2$	-1	1	0	0
$e_3$	0	1	-1	0
$e_4$	0	0	-1	1
$e_5$	0	-1	0	1

$$A = \begin{bmatrix} 1 & 0 & 0 & -1 \\ -1 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

For an incidence matrix  $A \in \mathbb{R}^{m \times n}$  corresponding to a digraph  $\mathcal{G} = \{\mathcal{N}, \mathcal{E}\}$ , the row dimension  $m$  represents the number of edges of the graph and the column dimension  $n$  represents the number of nodes existing between these nodes. Notice that for both undirected and directed graphs, we do not specify any coordinate system for the vertices of our graph. These modeling tools focus only on connectivity between objects and do not require specific locations for each node. There are a number of applications in which we might want to specify locations for each node in addition to connectivity between node. An example of one such application area is in 2D and 3D computer graphics.

### EXAMPLE 3.1.3

One of the subfields of computer science is that of 2D computer graphics. This branch focuses on generating computer-based models for two dimensional shapes which can be very useful in drawing and printing applications as well as 2D animation. Vector models can be particularly effective to encoding 2D geometric shapes. We begin our discussion of 2D computer graphics by focusing on **wireframe models**. We create a wireframe model by specifying a **vertex matrix** and **edge table**. The vertex matrix encodes the coordinates of each vertex in our model and the edge table dictates the connectivity between the vertices.

In this example, let's create a triangle using a wireframe model. We begin by looking back at Example 2.1.1. In that case, we had three vertices, which we labeled:

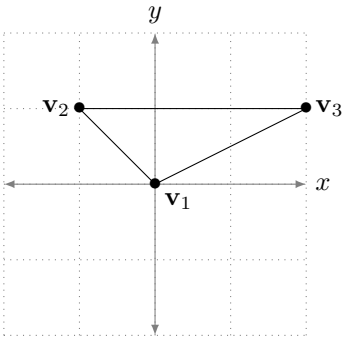
$$\mathbf{v}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \quad \mathbf{v}_3 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

We can encode this information in a *Vertex Table*.

2D Wireframe Vertex Table

	Vertex 1	Vertex 2	Vertex 3
1st Coordinate	0	-1	2
2nd Coordinate	0	1	1

We can also indicate the edges (connections) between vertices using an *Edge Table*.



2D Wireframe Edge Table

Edge	Start Vertex	End Vertex
1	1	2
2	2	3
3	3	1

Notice that we specify the start and end vertices for each edge separately. In this case, we do not care about creating directed edges. Thus the corresponding visual representation of our triangle contains lines connecting each vertex (thought of as “wires”). If we wanted to, we could use arrows that run from the starting vertex toward the end vertex, known as a directed edge. However, this is unnecessary for this model. Also notice that the model has no “area” since the shape contains only vertices and edges. This is why we call this a “frame” since it only specifies the outer region of the shape. It is up to us to fill in this shape if we so desire.

From this model, we can generate a vertex matrix

$$V = \begin{bmatrix} 0 & -1 & 2 \\ 0 & 1 & 1 \end{bmatrix}$$

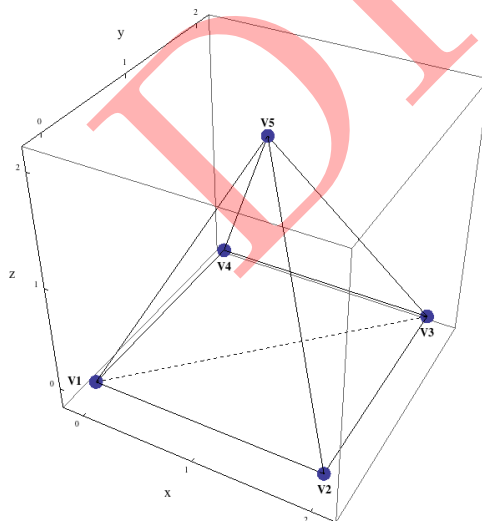
As we will see, we can then apply a variety of geometric transformations on this triangle using matrix-matrix multiplication on this matrix  $V$ . As long as the edge configurations don’t change, we can keep the edge table untouched while executing any geometric transformation.

#### EXAMPLE 3.1.4

In our example above, we discussed wireframe models used in 2D computer graphics. However, sometimes we might like the ability to create interesting 3D computer graphics. We can generalize our 2D wireframe model to create a 3D wireframe model that encodes three dimensional data. Let’s begin by creating a pyramid with a square base using the same wireframe model structure. In this case, we need 5 vertices, which we will label:

$$\mathbf{v}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{v}_3 = \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix}, \quad \mathbf{v}_4 = \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix}, \quad \mathbf{v}_5 = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}.$$

We can visualize this pyramid as follows:



In this figure, the vertices of the pyramid are labeled according to our specified vector indices  $\mathbf{v}_i$  for  $i = 1, 2, \dots, 5$ . The edges are drawn in black as lines. Now, let's create our Vertex Table for this pyramid:

3D Wireframe Vertex Table					
	Vertex 1	Vertex 2	Vertex 3	Vertex 4	Vertex 5
1st Coordinate	0	2	2	0	1
2nd Coordinate	0	0	2	2	1
3rd Coordinate	0	0	0	0	2

Just as before, we need to indicate which vertices are connected by specifying our corresponding *Edge Table*:

3D Wireframe Edge Table		
Edge	Start Vertex	End Vertex
1	1	2
2	2	3
3	3	4
4	3	1
5	4	1
6	1	5
7	2	5
8	3	5
9	4	5

Each edge is a single, undirected line in three space that connects the start vertex to the end vertex. Strictly speaking, the wireframe model we've created above does not contain information about polygon faces. In fact, this model encodes only the vertices and edges. Again, we can think of this as an empty frame in three dimensions which no connections between the edges (except at the vertices).

For now, let's generate the vertex matrix  $V$  corresponding to our pyramid:

$$V = \begin{bmatrix} 0 & 2 & 2 & 0 & 1 \\ 0 & 0 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

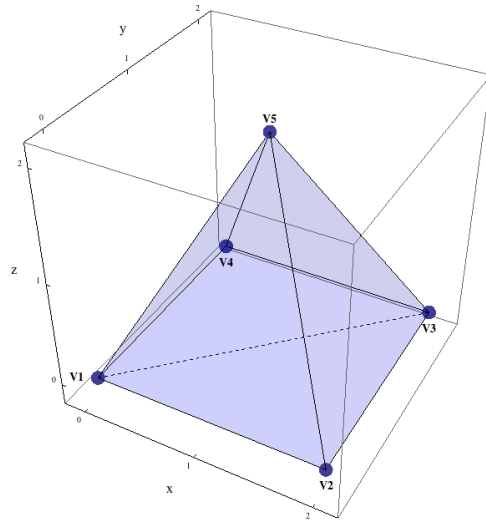
As we will see, we can use matrix-matrix multiplication to perform a range of geometric operations on these vertices.

#### EXAMPLE 3.1.5

In addition to creating wireframe models for shapes in three dimensions, it is often very helpful to create 3D computer graphics made up of polygons with vertices and faces. Let's generalize our 3D wireframe model to be a 3D polygon model that includes a description of faces (rather than edges). Such models are known as polygon meshes. In this case, we discuss the so-called Face-Vertex mesh model. Again we start with our square-based pyramid from the example above with vertex matrix

$$V = \begin{bmatrix} 0 & 2 & 2 & 0 & 1 \\ 0 & 0 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

This time, we visualize our pyramid as follows:



In this figure, the vertices of the pyramid are labeled according to our initial model. The faces of the pyramid are drawn in blue. The edges are drawn in black lines. This time, instead of creating Edge Table we will create a *Face List*:

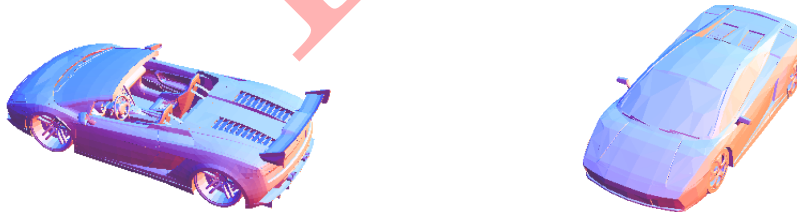
3D Polygon Face Table

Face	1st vertex	2nd Vertex	3rd Vertex
1	1	2	4
2	2	3	4
3	1	2	5
4	2	3	5
5	3	4	5
6	4	1	5

This model creates all faces using two-dimensional triangles that are created by connecting a set of three vertices. There are many advantages to using the face-vertex model (to see more about this, search the wikipedia article on “Polygon Mesh”). As we will see, we can use matrix-matrix multiplication to perform a range of geometric operations on these vertices.

#### EXAMPLE 3.1.6

Let’s take a look at some much more sophisticated graphics that can be created using polygon meshes. First, let’s look at two different 3D models of a Lamborghini Gallardo (one convertible and one hard top):



The convertible model contains 312,411 vertices. This data file was created by jarred1997 and is available at [www.thingiverse.com/thing:125339](http://www.thingiverse.com/thing:125339). Let’s also look

at two different models of the so-called Stanford Bunny (one very fine model and one course model):



The detailed Stanford bunny can be found at <http://www.thingiverse.com/thing:287082>. The simple Stanford bunny is available at [www.thingiverse.com/thing:466857](http://www.thingiverse.com/thing:466857).

### Entries of a matrix

The fundamental building block of all real matrices is the individual entry. Each entry, also referred to as an **element**, of a matrix is composed of three unique pieces of information including

- i. a row index  $i \in \{1, 2, \dots, m\}$
- ii. a column index  $k \in \{1, 2, \dots, n\}$
- iii. a real number given by  $a_{ik} \in \mathbb{R}$

When specifying the individual entries of a matrix, we must identify all three of these pieces of information. Using subscript notation, the row index always appears first and the column index always appears second.

#### Definition 3.3: Entry Operator

Let  $A \in \mathbb{R}^{m \times n}$ . Define the map

$$\text{Entry}_{ik}(A) = a_{ik} = A(i, k)$$

for  $i, k \in \mathbb{N}$  with  $1 \leq i \leq m$  and  $1 \leq k \leq n$ .

The entry operator takes in a row index, column index and matrix to produce the real number stored in position  $(i, k)$  of our given matrix.



**EXAMPLE 3.1.7**

If  $A \in \mathbb{R}^{6 \times 6}$ , then the value of the element with row index 5 and column index 3 is

$$\text{Entry}_{53}(A) = a_{53}$$

This is equivalent to viewing the full matrix and picking out the element in the fifth row and third column

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\ a_{51} & a_{52} & \boxed{a_{53}} & a_{54} & a_{55} & a_{56} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} \end{bmatrix}$$

**EXAMPLE 3.1.8**

We can create a model for digital images that utilizes matrices. To begin our model construction, we will define a continuous physical image as a function  $f(x, y)$  where  $f : D \rightarrow \mathbb{R}$ . In this case, we can write the domain of input points as

$$D = \{(x, y) : x_\ell \leq x \leq x_r \text{ and } y_\ell \leq y \leq y_r\} \subseteq \mathbb{R}^2.$$

Each point  $(x, y)$  represents a spatial coordinate in the frame of our picture. The real-valued output  $f(x, y)$  is the amplitude of the light at each point  $(x, y)$ . The amplitude of light at any point indicates the intensity (or brightness) of the image at that point. In the case of our continuous physical image, we consider the domain space  $D$  that defines our frame to be continuous and we allow for the amplitude function to take an infinite range of values in  $\mathbb{R}$ .

In order to convert a continuous image  $f(x, y)$  into a corresponding digital image  $f(\mathbf{x}, \mathbf{y})$ , we must digitize the image. This process requires two steps: sampling and quantization. When we sample our continuous image, we convert the input domain  $D$  into a set of finite, discrete values. Next, to quantize our image, we digitize the amplitudes by assigning finite, discrete values to captured amplitudes.

The result of sampling and quantization is a matrix of real numbers  $A$  that stores the digital image. In this case, we assume that the continuous image  $f(x, y)$  is sampled so that the resulting digital image has  $m$  rows and  $n$  columns. We say that the digital image matrix  $A$  is of size  $m \times n$ . Again, we emphasize that the values of the coordinates  $(\mathbf{x}, \mathbf{y})$  are discrete quantities. For notational convenience and clarity, we will use integer values for the discrete coordinate values. In our convention, we define the origin of the digital image to be at the point  $(\mathbf{x}, \mathbf{y}) = (1, 1)$ . The next coordinate along the first row is  $(\mathbf{x}, \mathbf{y}) = (1, 2)$  which signified the second sampled point in the first row.

For any coordinate  $(\mathbf{x}, \mathbf{y})$ , the first element  $\mathbf{x}$  refers to the row number of the captured data and the second coordinate  $\mathbf{y}$  refers to the column number. We notice that  $\mathbf{x}$  will range from 1 to  $m$  and  $\mathbf{y}$  will range from 1 to  $n$ . Notice that in this model, the coordinate pair  $(\mathbf{x}, \mathbf{y})$  does not necessarily correspond to the continuous spacial coordinates  $(x, y)$ . The spacial coordinates  $(x, y)$  are used to signify the physical location of any point in our frame while the sampled digital coordinates  $(\mathbf{x}, \mathbf{y})$  encode the location of the data point within our digital image.

	1	2	3	...	n
1	.	.	.	.	.
2	.	.	.	.	.
3	.	.	.	.	.
⋮	.	.	.	.	.
m	.	.	.	.	.

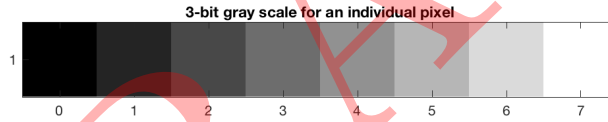
In this case, the point  $(r, c)$  is known as a pixel coordinate. The coordinate system we discussed above leads to the following representation of our digital image function:

$$A = \begin{bmatrix} f(1,1) & f(1,2) & \cdots & f(1,n) \\ f(2,1) & f(2,2) & \cdots & f(2,n) \\ \vdots & \vdots & \ddots & \vdots \\ f(m,1) & f(m,2) & \cdots & f(m,n) \end{bmatrix}$$

The notation  $f(r, c)$  represents the quantized amplitude value located in row  $r$  and column  $c$ . See Section 5.9.2: The camera and the image plane on page 237 of Coding the Matrix by Philip N. Klein

#### EXAMPLE 3.1.9

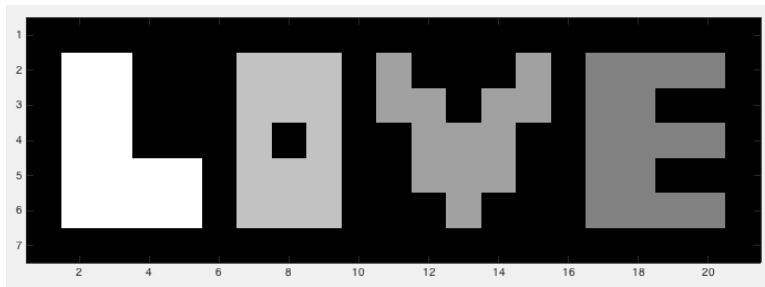
Let's create a digital image using a  $7 \times 21$  with a total of 147 pixels. Our digital image will be broken up into 7 rows of pixels down our image and 21 columns of pixels across. We use a 3-bit gray scale encoding to store the individual pixel values. This means each pixel stores a binary number representing an integer value in the set  $N_3 = \{0, 1, 2, \dots, 7\}$ . These encodings correspond to the spectrum of shades of gray displayed in the diagram below.



Our digital image is a diagram of the word “LOVE” with all upper case letters. To create a digital image of this word, we store our image in the following matrix.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 7 & 7 & 0 & 0 & 0 & 5 & 5 & 5 & 0 & 3 & 0 & 0 & 0 & 3 & 0 & 3 & 3 & 3 & 0 \\ 0 & 7 & 7 & 0 & 0 & 0 & 5 & 5 & 5 & 0 & 3 & 3 & 0 & 3 & 3 & 0 & 3 & 3 & 0 & 0 \\ 0 & 7 & 7 & 0 & 0 & 0 & 5 & 0 & 5 & 0 & 0 & 3 & 3 & 3 & 0 & 0 & 3 & 3 & 3 & 0 \\ 0 & 7 & 7 & 7 & 7 & 0 & 5 & 5 & 5 & 0 & 0 & 3 & 3 & 3 & 0 & 0 & 3 & 3 & 0 & 0 \\ 0 & 7 & 7 & 7 & 7 & 0 & 5 & 5 & 5 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 3 & 3 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Now, we can visualize our image as follows:



Notice that zero entries in our digital image correspond to black pixels (no light) while entries with a value of 7 correspond to white pixels. This is yet another demonstration of the power of discretization in approximating continuous behavior.



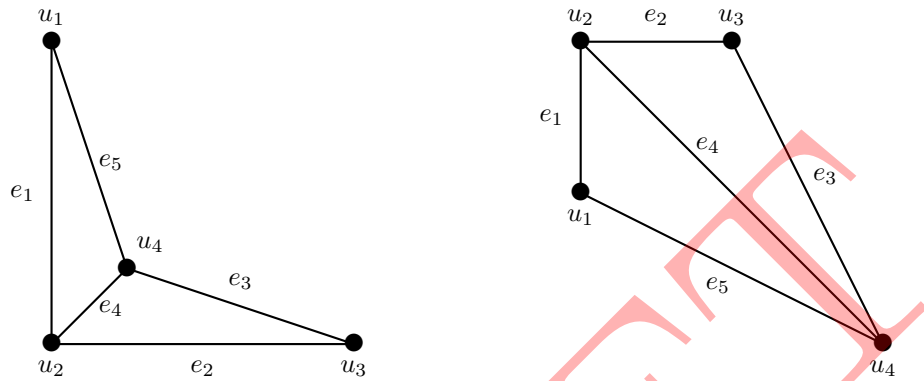
DRAFT

### Definition 3.4: Equal Matrices

Two matrices are defined to be **equal** if they have the same dimensions and if entries with the same row and column index have the same numerical value.

#### EXAMPLE 3.1.10

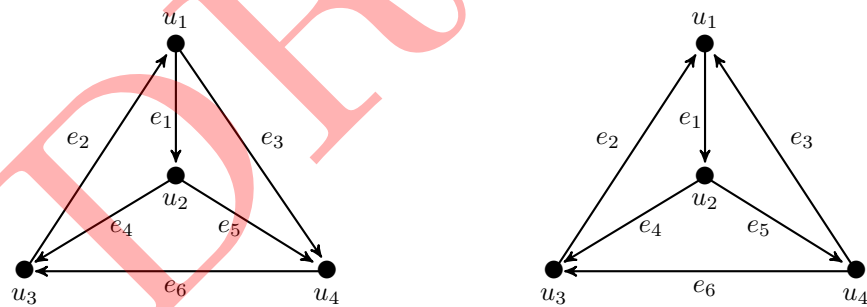
Consider the undirected graphs defined in the following diagram.



The corresponding incidence matrices for each of these graphs are equal. The fact that these incidence matrices are equal indicates that the connectivity between the nodes of each graph are identical. This is a powerful observation: graphs model the connectivity between nodes in a system. Graphs are not used to encode the exact location of each node within a specific coordinate system. In other words, graphs are useful models to encode connectivity but not particularly useful to encode specific location.

#### EXAMPLE 3.1.11

Consider the directed graphs defined in the following diagram.



The corresponding incidence matrices for each of these graphs are NOT equal.

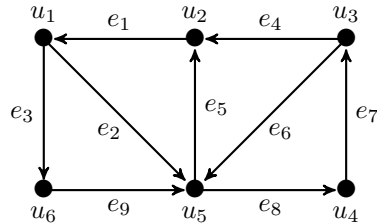
Any time we are looking to compare two matrices to see if these matrices are equal, the first thing we need to do is to check the dimensions.

## Matrix Modeling- Suggested Problems

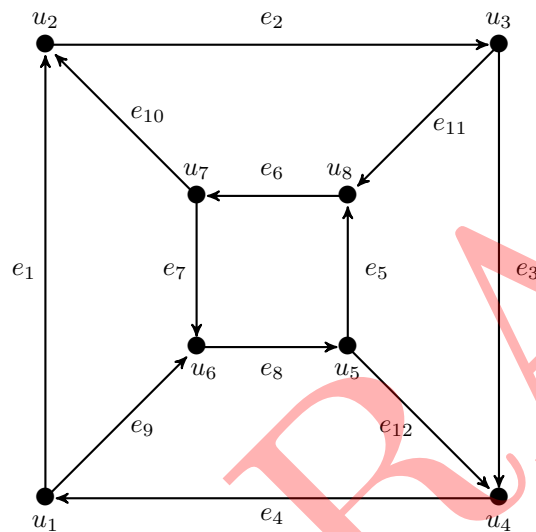
For all the problems below, be sure to explicitly state the dimensions of the matrices you use for each model.

1. Create the incidence matrix  $A$  associated with the following directed graphs:

i.



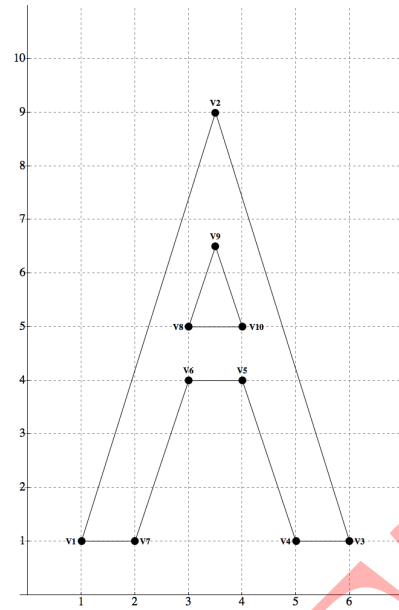
ii.



- A. Write out matrix  $A$  in full and specify its dimensions
- B. Take the dot product between row 1 and 5 of matrix  $A$ . What do these vectors tell you about the graph?
- C. Take the dot product between columns 2 and 4 of matrix  $A$ . What do these vectors indicate about the graph?

\*Author's Note: For future iterations of this problem, create a library of graphs associated with "real-world" problems. Ideally, these will be connected to other application examples in this book.

2. Create a wireframe model for the letter A given in the figure to the below.



- Specify the vertex table and edge table
- Write the vertex matrix  $V$  from the vertex table
- Create any polygon of your choosing and encode as a wireframe model.