Chapter 6

Theory of Computation and Computer Graphics

Krishna Gaire (Data Engineer @ CGT)

Outlines

6.1 Introduction to finite automata
6.2 Introduction to context free language
6.3 Turing machine

6.4 Introduction of computer graphics
6.5 Two-dimensional transformation
6.6 Three-dimensional transformation

6.1 Introduction to finite automata

Introduction to Finite Automata and Finite State Machine
Equivalence of DFA and NDFA
Regular Expressions
Minimization of Finite State Machines
Equivalence of Regular Expression and Finite Automata

Pumping lemma for regular language.

Formal languages

An alphabet $\sum$ is a set of symbols:

e.g. $\sum = \{a, b, c\}$

A string ω is a sequence of symbols, e.g $\omega = abcb$.

The empty string ε consists of zero symbols.

The Kleene closure $\sum^*$ ('sigma star') is the (infinite) set of all strings that can be formed from $\sum$:

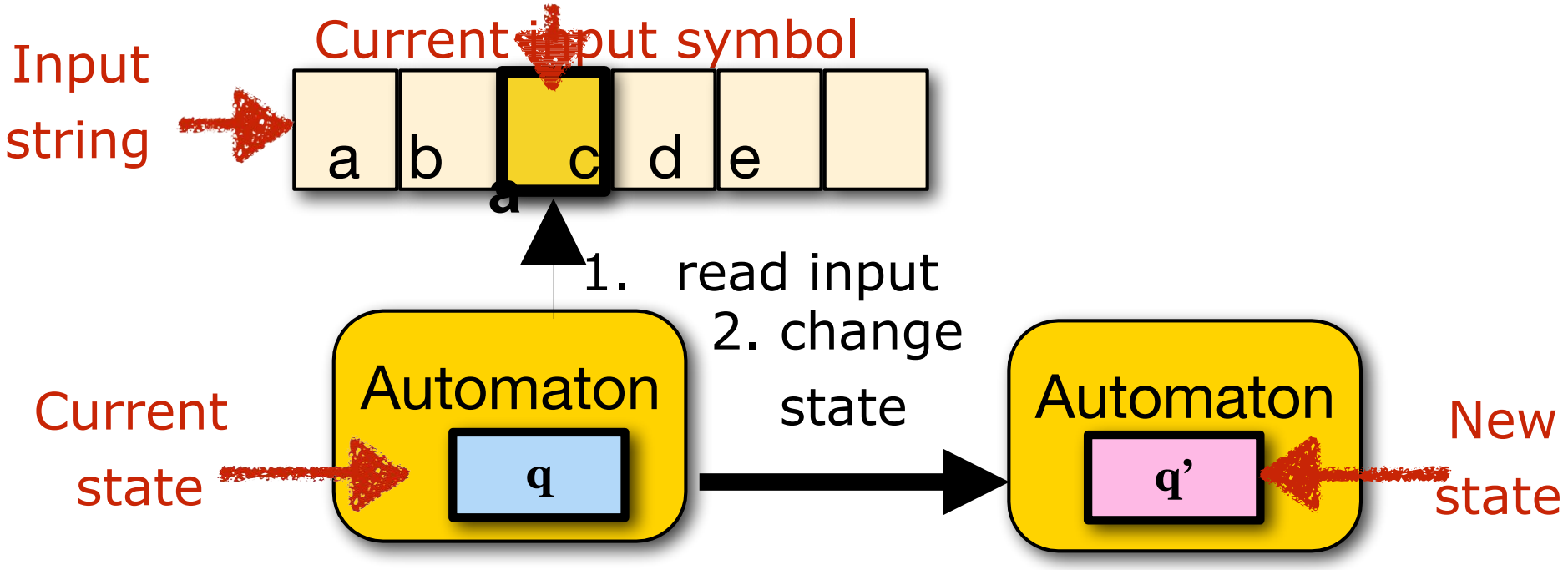
$\sum^* = \{\varepsilon, a, b, c, aa, ab, ba, aaa, ...\}$

A language $L \subseteq \sum^*$ over $\sum$ is also a set of strings.

Typically we only care about proper subsets of $\sum^*$ ($L \subset \Sigma$).

Automata and languages

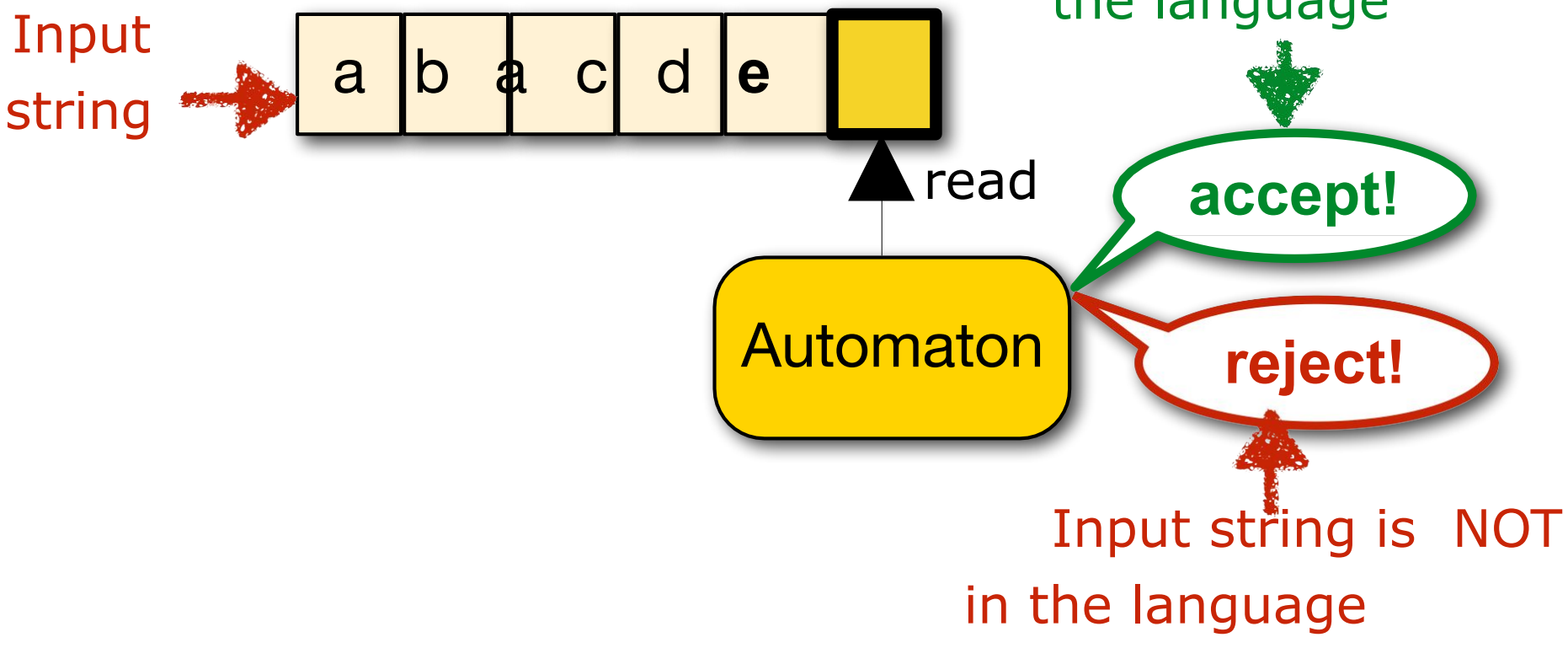
An automaton is an abstract model of a computer. It *reads* an input string symbol by symbol.
It *changes* its internal state depending on the current input symbol and its current internal state.



Automata and languages

The automaton either accepts or rejects the input string.

Every automaton defines a language (the set of strings it accepts).



Automata and languages

Different types of automata define
different language classes:

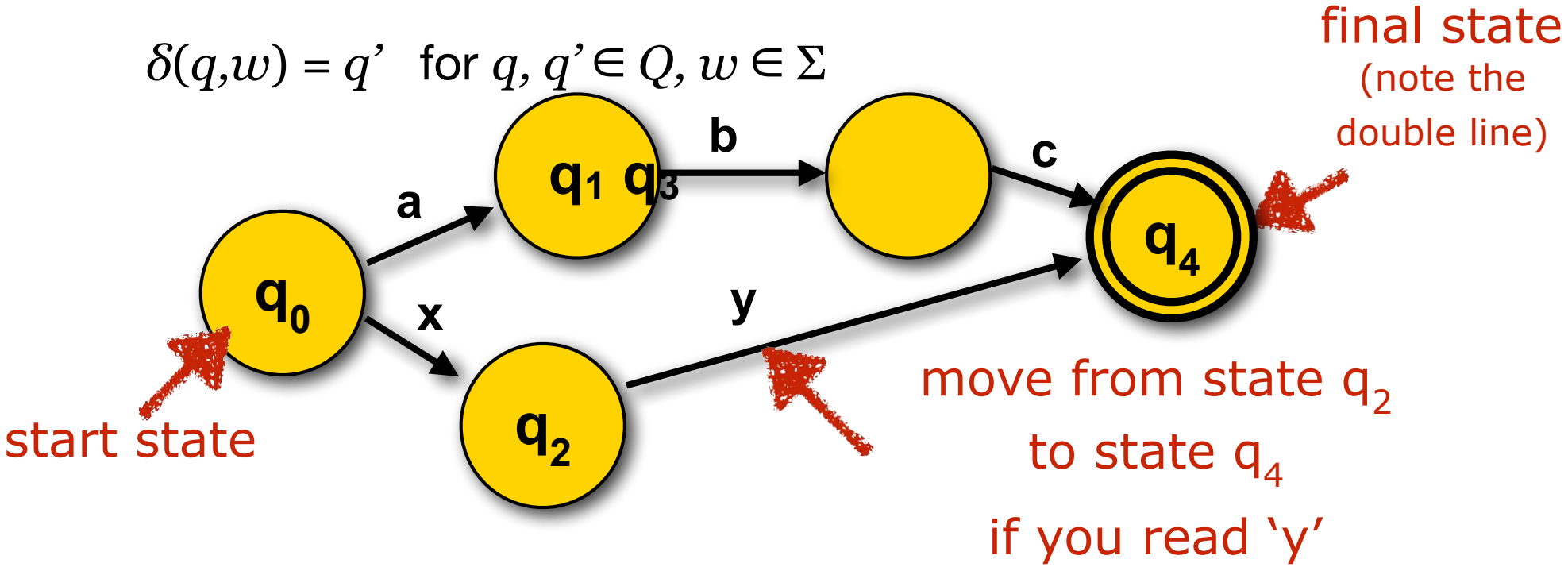
- **Finite-state** automata define **regular** languages
- **Pushdown** automata define **context-free** languages
- **Turing machines** define **recursively enumerable** languages

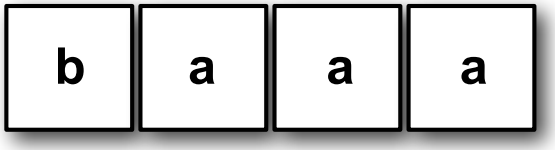
Finite-state automata

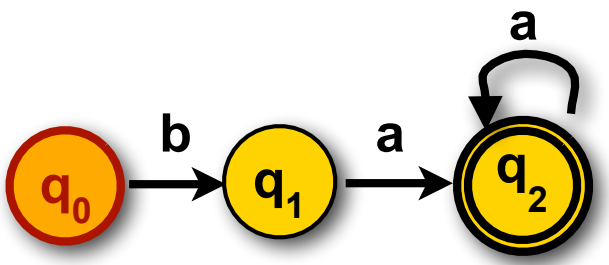
A (deterministic) finite-state automaton (FSA) consists of:

- a **finite set of states** $Q = \{q_0 ... q_N\}$, including **a start state** q_0 and one (or more) **final (=accepting) states** (say, q_N)
- a (**deterministic**) transition function

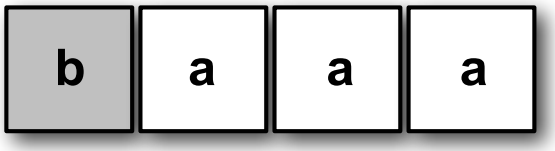
$\delta(q,w) = q'$ for $q, q' \in Q, w \in \Sigma$

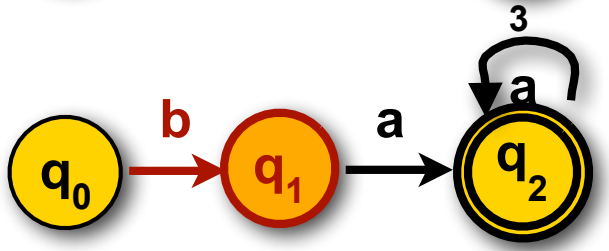


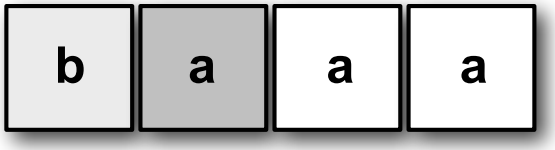


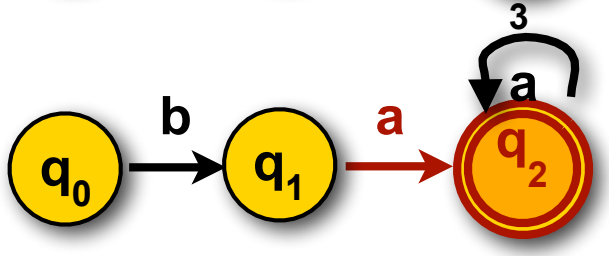


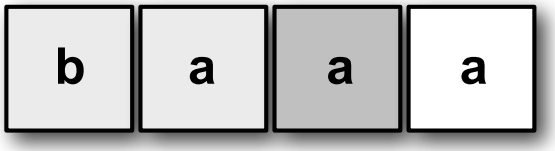
Start in q_0

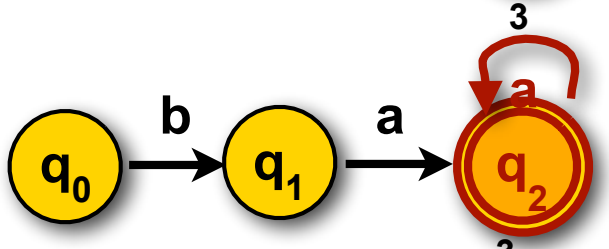


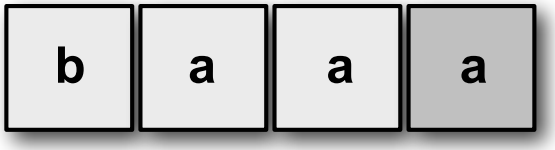


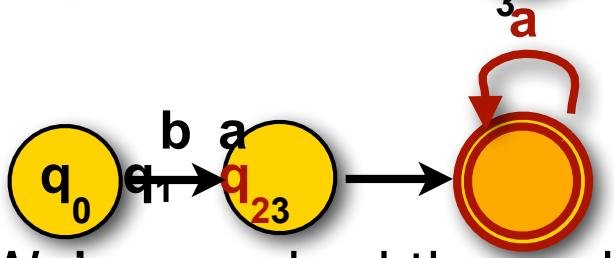








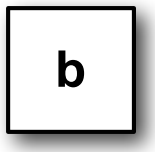


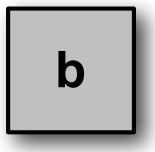


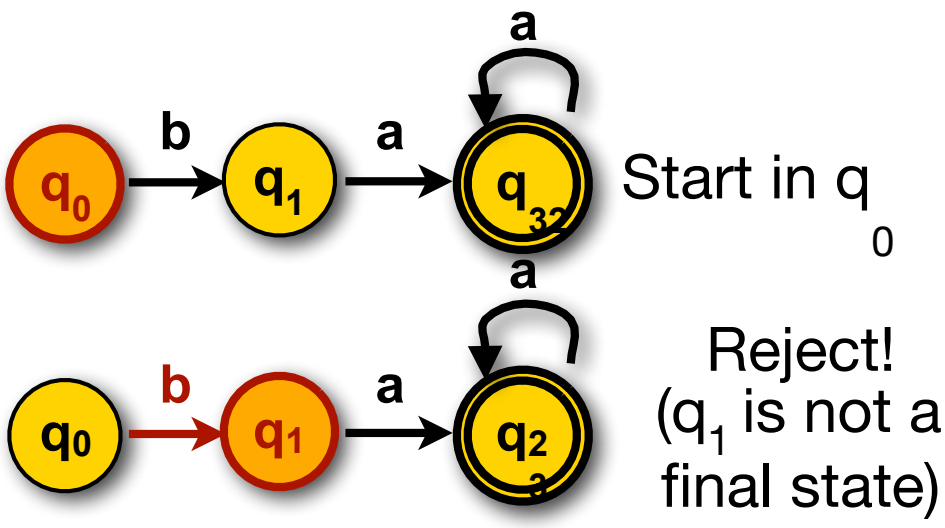
Accept

We've reached the end of the string, and are in an accepting state.

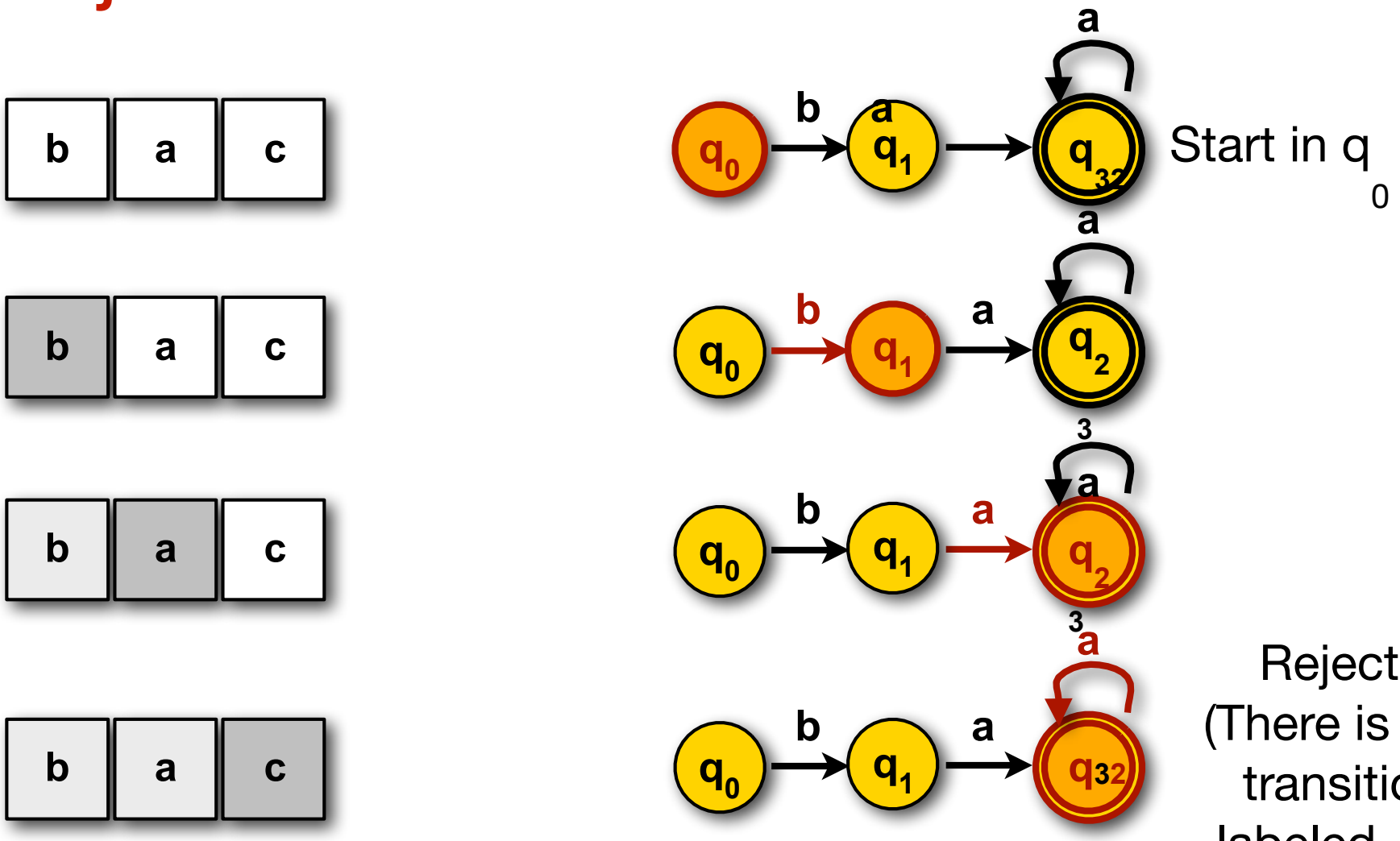
Rejection: Automaton does not end up in accepting state







Rejection: Transition not defined



Finite State Automata (FSAs)

A finite-state automaton $M = \langle Q, \Sigma, q_0, F, \delta \rangle$ consists of:

- A finite set of states $Q = \{q_0, q_1, .., q_n\}$
- A finite alphabet Σ of input symbols (e.g. $\Sigma = \{a, b, c, ...\}$)
- A designated start state $q_0 \in Q$
- A set of final states $F \subseteq Q$
- A transition function δ:
  - The transition function for a deterministic (D)FSA: $Q \times \Sigma \rightarrow Q$

    $\delta(q,w) = q'$ for $q, q' \in Q, w \in \Sigma$

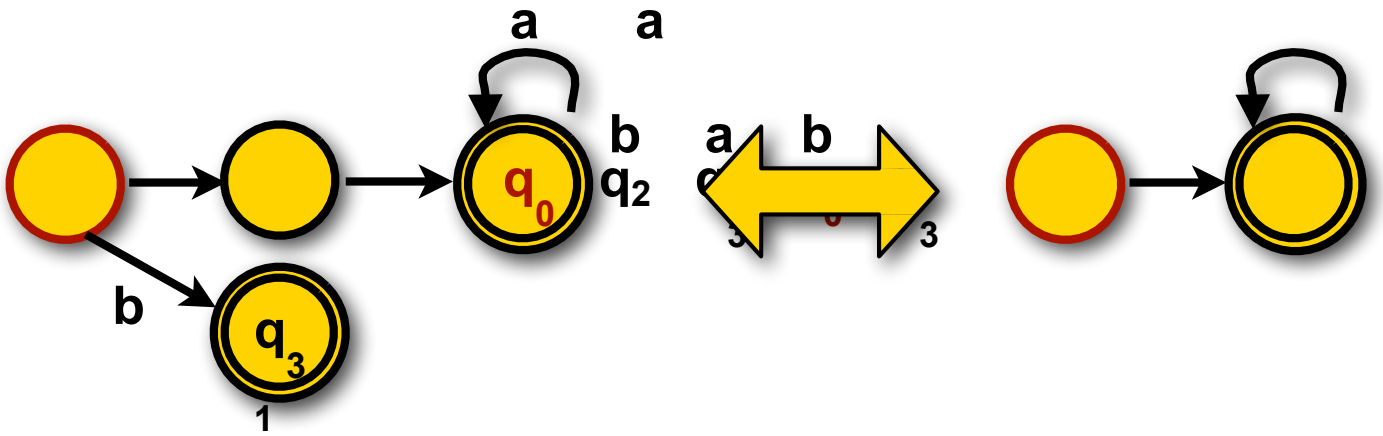
    If the current state is q and the current input is w, go to q'
  - The transition function for a nondeterministic (N)FSA: $Q \times \Sigma \rightarrow 2^Q$

    $\delta(q,w) = Q'$ for $q \in Q, Q' \subseteq Q, \quad w \in \Sigma$

    If the current state is q and the current input is w, go to any $q' \in Q'$

Finite State Automata (FSAs)

Every NFA can be transformed into an equivalent DFA:



Recognition of a string *w* with a DFA is linear in the length of *w*

Finite-state automata define the class of regular languages

$L_1 = \{ a^n b^m \} = \{ab, aab, abb, aaab, abb, \ldots\}$ is a regular language,

$L_2 = \{ a^n b^n \} = \{ab, aabb, aaabbb, \ldots\}$ is not (it's context-free).

You cannot construct an FSA that accepts all the strings in L_2 and nothing else.

Regular Expressions

Regular expressions can also be used to define a regular language.

Simple patterns:

- **Standard characters** match themselves: *'a'*, *'1'*
- **Character classes**: *'[abc]'*, *'[0-9]'*, **negation:** *'[^aeiou]'*
  (Predefined: *\s* (whitespace), *\w* (alphanumeric), etc.)
- **Any character** (except newline) is matched by '.'

Complex patterns: (e.g. ^[*A-Z*]([*a-z*])+\s)

- **Group:** '(…)'
- **Repetition:** 0 or more times: *'*'*, 1 or more times: *'+'*
- **Disjunction:** *'...|...'*
- **Beginning of line** *'^'* and **end of line** '$'

Useful website for regex

https://www.w3schools.com/

https://regexr.com/

MCQs

MCQs:

1. Which of the following is a component of a finite automaton?
   a. Alphabet b. Transition function c. States d. **All of the above**

2. In a DFA, for a given state and input, the transition to the next state is:
   a) Non-deterministic b) **Unique** c) Undefined d) Multiple

3. Which of the following automata can have epsilon (ε) transitions?
   a) DFA b) **NFA** c) Both DFA and NFA d) Neither DFA nor NFA

Lecture 2

- DFA and NFA (Continue from lecture 1 ...)
- Grammar
- Context-Free Languages
- Chomsky Normal Form
- Greibach Normal Form (GNF)
- Backus-Naur Form (BNF)
- Pushdown Automata (PDA)
- Pumping lemma for context free language
- Properties of context free Language.

DFA and NFA

Deterministic Finite Automaton (DFA) and Non-Deterministic Finite Automaton (NFA) are fundamental concepts in the theory of computation, particularly in the study of formal languages and automata theory.

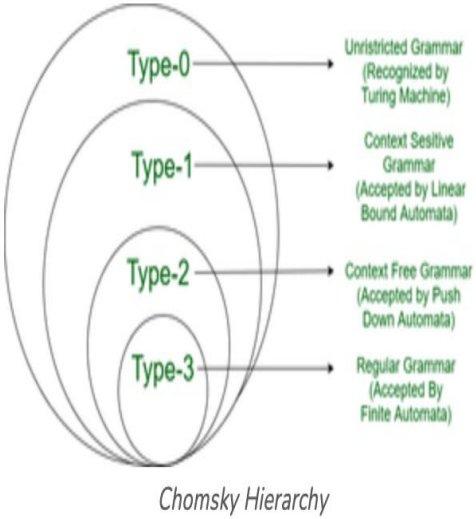
They are used to model computation and recognize patterns within input strings. Here’s a detailed explanation of both:

A **DFA** is a finite state machine where for each state and input symbol, there is exactly one transition to a next state. This makes the DFA's behavior predictable and straightforward.

An **NFA** is similar to a DFA but with a key difference: for a given state and input symbol, it can transition to any number of next states, including zero states. This introduces the concept of non-determinism.

Grammar

The list provided describes the four levels of the Chomsky hierarchy, a classification of formal grammars in computational theory, based on their generative power. Here's a brief explanation of each:



Chomsky Hierarchy

0. Unrestricted Grammar : These are the most general type of grammars and have no restrictions on their production rules. They can generate any language that can be recognized by a **Turing machine**, meaning they are as powerful as any computational system.

1. Context Sensitive Grammar : These grammars have production rules of the form $\alpha A \beta \rightarrow \alpha \gamma \beta$ αAβ→αγβ, where A A is a non-terminal, and α α, β β, and γ γ are strings of terminals and/or non-terminals, with γ γ being non-empty.

2. Context Free Grammar : Context-free grammars are powerful enough to describe many syntactical constructs in programming languages and natural languages.

3. Regular Grammar : These grammars have the most restricted form of production rules. They are either of the form A→aBA \rightarrow aBA→aB or A→aA \rightarrow aA→a, where AAA and BBB are non-terminals and aaa is a terminal. Regular grammars generate regular languages, which can be represented by regular expressions and are recognized by finite automata.

Each level of the hierarchy is a superset of the one below it, meaning every regular grammar is also context-free, every context-free grammar is context-sensitive, and every context-sensitive grammar is unrestricted.

Context-Free Languages

The class of context-free languages consists of languages that have some sort of **recursive structure**.

We will see two equivalent methods to obtain this class.

1. We start with **context-free grammars**, which are used for defining the syntax of programming languages and their compilation.

2. Then we introduce the notion of **(nondeterministic) pushdown automata**, and show that these automata have the same power as context-free grammars.

Context-Free Languages

The class of context-free languages consists of languages that have some sort of **recursive structure**.

We will see two equivalent methods to obtain this class.

1. We start with **context-free grammars**, which are used for defining the syntax of programming languages and their compilation.

2. Then we introduce the notion of **(nondeterministic) pushdown automata**, and show that these automata have the same power as context-free grammars.

We start with an example. Consider the following five (substitution) rules:

$$S \rightarrow AB$$
$$A \rightarrow a \quad A \rightarrow aA \quad B \rightarrow b \quad B \rightarrow bB$$

Here, S, A, and B are *variables*, S is the *start variable*, and a and b are *terminals*. We use these rules to derive strings consisting of terminals (i.e., elements of $\{a, b\}^*$), in the following manner:

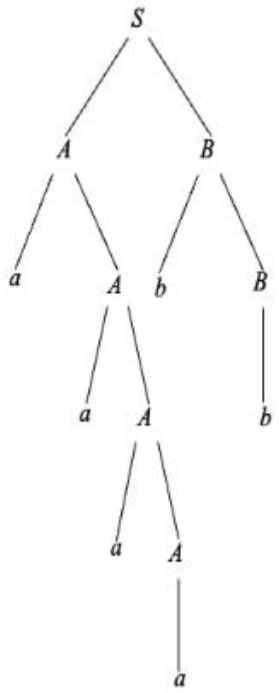
1. Initialize the *current string* to be the string consisting of the start variable S.

2. Take any variable in the current string and take any rule that has this variable on the left-hand side. Then, in the current string, replace this variable by the right-hand side of the rule.

3. Repeat 2. until the current string only contains terminals.

For example, the string *aaaabb* can be derived in the following way:

$$\begin{aligned} S &\Rightarrow AB \\ &\Rightarrow aAB \\ &\Rightarrow aAbB \\ &\Rightarrow aaAbB \\ &\Rightarrow aaaAbB \\ &\Rightarrow aaaabB \\ &\Rightarrow aaaabb \end{aligned}$$



This derivation can also be represented using a ***parse tree,*** as in the figure

Definition

A context-free grammar is a **4-tuple** $G = (V, \Sigma, R, S)$, where

1. V is a finite set, whose elements are called *variables*,

2. Σ is a finite set, whose elements are called *terminals*,

3. $V \cap \Sigma = \varnothing$,

4. S is an element of V ; it is called the *start variable*,

5. R is a finite set, whose elements are called *rules*. Each rule has the form $A \longrightarrow w$, where $A \in V$ and $w \in (V \cup \Sigma)^*$.

In our example, we have $V = \{S, A, B\}$, $\Sigma = \{a, b\}$, and

$$R = \{S \longrightarrow AB,\ A \longrightarrow a,\ A \longrightarrow aA,\ B \longrightarrow b,\ B \longrightarrow bB\}.$$

Chomsky Normal Form

A **context-free grammar** $G = (V, \Sigma, R, S)$ is said to be in *Chomsky normal form*, if every rule in R has one of the following three forms:

1. $A \rightarrow BC$, where A, B, and C are elements of V , $B \neq S$, and $C \neq S$.

2. $A \rightarrow a$, where A is an element of V and a is an element of Σ.

3. $S \rightarrow \varrho$, where S is the start variable.

You should convince yourself that, for such a grammar, R contains the rule $S \rightarrow \varrho$ if and only if $\varrho \in L(G)$.

Greibach Normal Form (GNF)

GNF stands for Greibach normal form. A CFG(context free grammar) is in GNF(Greibach normal form) if all the production rules satisfy one of the following conditions:

- A start symbol generating ε. For example, S → ε.
- A non-terminal generating a terminal. For example, A → a.
- A non-terminal generating a terminal which is followed by any number of non-terminals. For example, S → aASB.

For Example :

G1 = {S → aAB | aB, A → aA| a, B → bB | b}

G2 = {S → aAB | aB, A → aA | ε, B → bB | ε}

Backus-Naur Form (BNF)

BNF stands for **Backus Naur Form** notation. BNF may be a meta-language (a language that cannot describe another language) for primary languages.

Left side → definition

$S \rightarrow aSa$,

$S \rightarrow bSb$

$S \rightarrow c$

BNF : $S \rightarrow aSa \mid bSb \mid c$

Push Down Automata (PDA)

Pushdown automata is a way to implement a CFG in the same way we design DFA for a regular grammar. A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information with an "**external stack memory**"

The PDA can be defined as a collection of **7 components**:

Q: the finite set of states

∑: the input set

Γ: a stack symbol which can be pushed and popped from the stack

q0: the initial state

Z: a start symbol which is in Γ.

F: a set of final states

δ: mapping function which is used for moving from current state to next state.

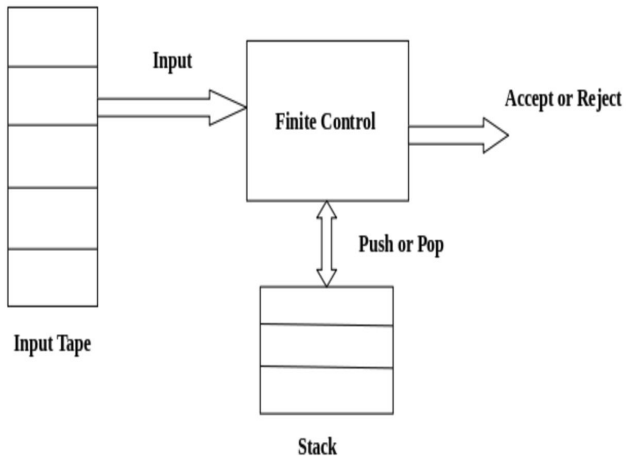


Fig: Pushdown Automata

Pumping lemma for context free language

Pumping Length (for CFL) is used to prove that a language is not context free

Let L be a context-free language. Then there exists an integer $p \geq 1$, called the pumping length, such that the following holds:

Every string s in L, with $|s| \geq p$, can be written as s = uvxyz, such that
1. $|vy| \geq 1$ (i.e., v and y are not both empty),
2. $|vxy| \leq p$, and
3. $uvixyiz \in L$, for all $i \geq 0$.

Pumping lemma for CFL : 5 Pieces
Pumping lemma for Regular Langauge : 3 Pieces

Properties of context free Language.

1. Union
2. Concentration
3. Transpose
4. Kleenstar
5. Intersection
6. Complement.

Key Points

Finite Automata: Understand the concept of finite automata (FA) and finite state machines. An FA is represented by a **5-tuple** (Q, Σ, δ, q0, F).

DFA: Deterministic Finite Automata - has one unique transition for each symbol and state. Represented by a **5-tuple** (Q, Σ, δ, q0, F), where $\delta: Q \times \Sigma \rightarrow Q$.

NDFA: Non-Deterministic Finite Automata - can have multiple transitions for the same symbol and state. Represented by a 5-tuple (Q, Σ, δ, q0, F), where $\delta: Q \times \Sigma \rightarrow 2^Q$.

Equivalence of DFA and NDFA: Both recognize the same class of languages (regular languages). Any NDFA can be converted to an equivalent DFA.

Minimization of Finite State Machines: Techniques to reduce the number of states in a DFA while preserving the language it recognizes.

Regular Expressions: Representations of regular languages using symbols and operators. E.g., a(b|c)*.

Equivalence of Regular Expressions and Finite Automata: Both describe the same set of languages. A regular expression can be converted to an FA and vice versa.

Pumping Lemma for Regular Languages: A property used to prove that certain languages are not regular. It states that for any regular language, there exists a length p such that any string longer than **p can be split into three parts,** and the middle part can be pumped (repeated) to produce new strings in the language.

Context-Free Grammar (CFG): A type of grammar where each production rule has a single non-terminal on the left-hand side. Represented by a **4-tuple** (V, Σ, R, S).

Derivative Trees: Visual representations of the derivations of strings in a language, showing how a string is generated by the grammar.

Bottom-Up Approach: Building the parse tree from the leaves to the root.

Top-Down Approach: Building the parse tree from the root to the leaves.

Leftmost Derivation: Replacing the leftmost non-terminal first during derivation.

Rightmost Derivation: Replacing the rightmost non-terminal first during derivation.

Language of a Grammar: The set of all strings that can be generated using the grammar.

Parse Tree Construction: Creating a tree that represents the structure of a string according to a grammar.

Ambiguous Grammar: A grammar that can generate the same string in multiple ways, resulting in different parse trees.

Chomsky Normal Form (CNF): A form of CFG where each production rule is either A → BC or A → a, making the grammar simpler and easier to parse.

Greibach Normal Form (GNF): A form of CFG where each production rule is $A \rightarrow a\alpha$, useful for certain parsing algorithms.

Backus-Naur Form (BNF): A notation used to express the grammar of a language in a formal way, often used in programming language specifications.

Pushdown Automata (PDA): A type of automaton that uses a stack to handle context-free languages. Represented by a **7-tuple** $(Q, \Sigma, \Gamma, \delta, q0, Z0, F)$, where Γ is the stack alphabet and $Z0$ is the initial stack symbol.

Pumping Lemma for Context-Free Languages: A property used to prove that certain languages are not context-free. It states that for any context-free language, there exists a length p such that any string longer than p can be **split into five parts**, and certain parts can be pumped to produce new strings in the language.

Closure Properties of Regular Languages: Regular languages are closed under operations like union, intersection, and complementation.

Closure Properties of Context-Free Languages: Context-free languages are closed under operations like union and concatenation but not under intersection and complementation.

Applications: Real-world applications of finite automata and context-free languages in areas like compiler design, formal verification, and natural language processing.

More Resources at:

Book :
https://cglab.ca/~michiel/TheoryOfComputation/TheoryOfComputation.pdf

https://www.geeksforgeeks.org/introduction-of-pushdown-automata/

https://www.javatpoint.com/pushdown-automata

Next Lecture

Turing machine …

6.4 Introduction of computer graphics

- Overview of Computer Graphics
- Graphics Hardware (Display Technology, Architecture of Raster-Scan Displays, Vector Displays, Display Processors, output device and Input Devices)
- Graphics Software and Software standards.

Applications of Computer Graphics

There are many applications of computer graphics discussed below

Presentation Graphics – It is used to summarize financial statistical scientific or economic data. For example- Bar charts systems and line charts.

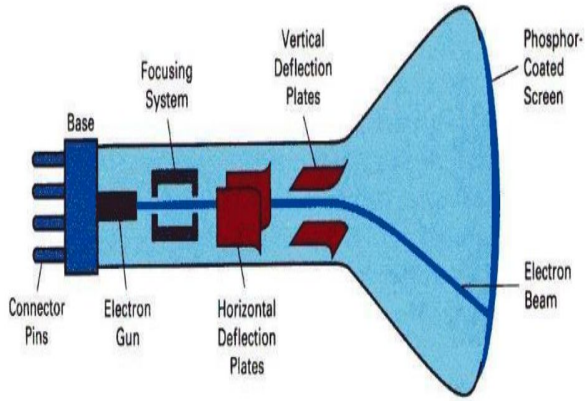
Entertainment- It is used in motion pictures, music videos, and television gaming.

Education and training- It is used to understand the operations of complex systems. It is also used for specialized systems such as framing for captains, pilots, and so on.

Visualization- To study trends and patterns. For example- Analyzing satellite photos of earth

6.4 Introduction of computer graphics

An electron gun emits a beam of electrons which passes through focusing and deflection systems and hits on the phosphor-coated screen. The number of points displayed on a CRT is referred to as resolutions (eg. 1024x768).



Different phosphors emit small light spots of different colors, which can combine to form a range of colors.

A common methodology for color CRT display is the Shadow-mask meth

The light emitted by phosphor fades very rapidly, so it needs to redraw the picture repeatedly.

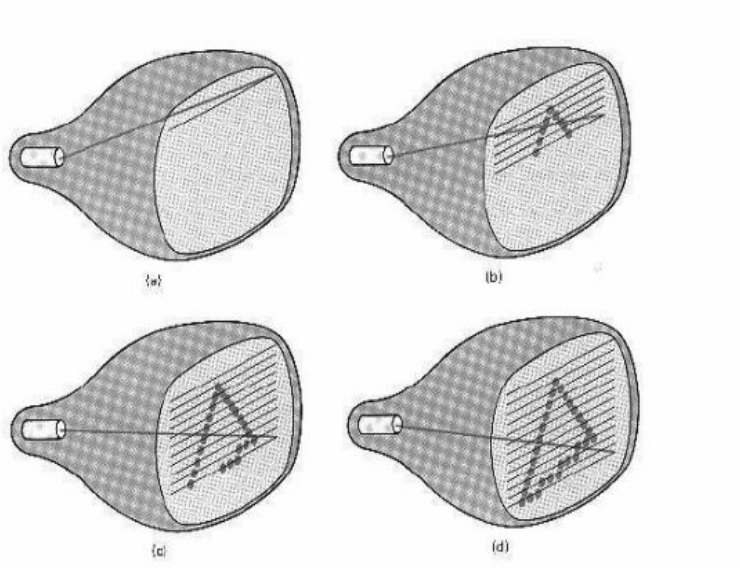
Overview of Computer Graphics

There are 2 kinds of redrawing mechanisms:

1. Raster-Scan
2. Random-Scan (Vector Display)

Architecture of Raster-Scan Displays

The electron beam is swept across the screen one row at a time from top to bottom. As it moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots.



This scanning process is called refreshing.

Each complete scanning of a screen is normally called a **frame**.

Architecture of Raster-Scan Displays

The refreshing rate, called the **frame rate**, is normally 60 to 80 frames per second, or described as **60 Hz to 80 Hz.**

Picture definition is stored in a memory area called the **frame buffer.**

This frame buffer stores the intensity values for all the screen points. Each screen point is called a **pixel** (picture element).

On black and white systems, the frame buffer storing the values of the pixels is called a **bitmap**.

Each entry in the bitmap is a 1-bit data which determine the on (1) and off (0) of the intensity of the pixel.

Architecture of Raster-Scan Displays

On color systems, the frame buffer storing the values of the pixels is called a **pixmap** (Though nowadays many graphics libraries name it as bitmap too).

Each entry in the pixmap 2 occupies a number of bits to represent the color of the pixel.

For a true color display, the number of bits for each entry is 24 (8 bits per red/green/blue channel.

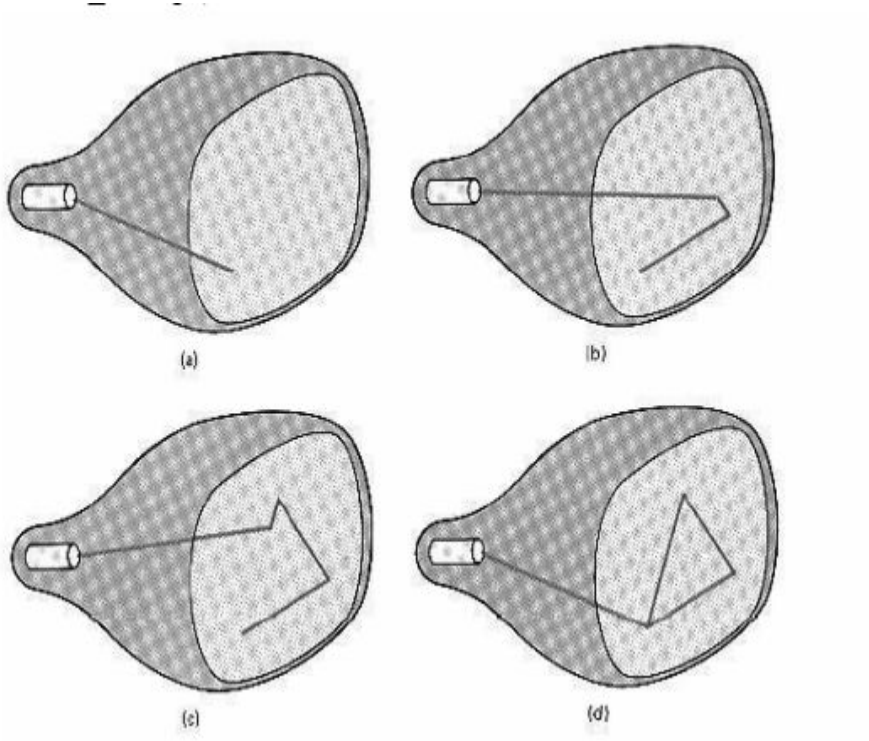
each channel 28=256 levels of intensity value, ie. 256 voltage settings for each of the red/green/blue electron guns).

Raster-scan system uses **shadow-mask** method because they produce wide range of colors.

Random-Scan (Vector Display)

The CRT's electron beam is directed only to the parts of the screen where a picture is to be drawn. The picture definition is stored as a set of line-drawing commands in a refresh display file or a refresh buffer in memory.

Random-scan generally have higher resolution than raster systems and can produce smooth line drawings, however it cannot display realistic shaded scenes.



Unlike raster displays, vector displays do not require refreshing.

Display Processors

Display Processor is the interpreter or a hardware that converts display processor code into picture.

The Display Processor converts the digital information from CPU to analog values.

The main purpose of the Digital Processor is to free the CPU from most of the graphic chores.

The Display Processor digitize a picture definitions given in an application program into a set of pixel intensity values for storage in the frame buffer.

This digitization process is called Scan Conversion.

Display Technology

CRT (Cathode Ray Tube): Uses electron beams to light up phosphors on the screen. Traditional and bulky.

LCD (Liquid Crystal Display): Uses liquid crystals and backlighting. Slim and energy-efficient.

LED (Light Emitting Diode): Uses LEDs for backlighting in LCD screens or directly as pixels in OLED displays.

OLED (Organic LED): Each pixel emits its own light, offering better color and contrast.

Output Device & Input Devices

Input Devices

Keyboard and Mouse: Basic input devices for user interaction with graphics software.

3D Mice: Allow for navigation and interaction within three-dimensional space.

Scanners: Devices for digitizing physical images or documents into digital formats.

Output Devices

Printers: Includes laser and inkjet printers for producing hard copies of graphics.

Plotters: Devices used for printing vector graphics, useful in engineering and architectural drawings.

Graphics Software and Software Standards

Graphics Software Types: Includes graphics editors like Adobe Photoshop, 3D modeling software like Blender, and CAD software like AutoCAD.

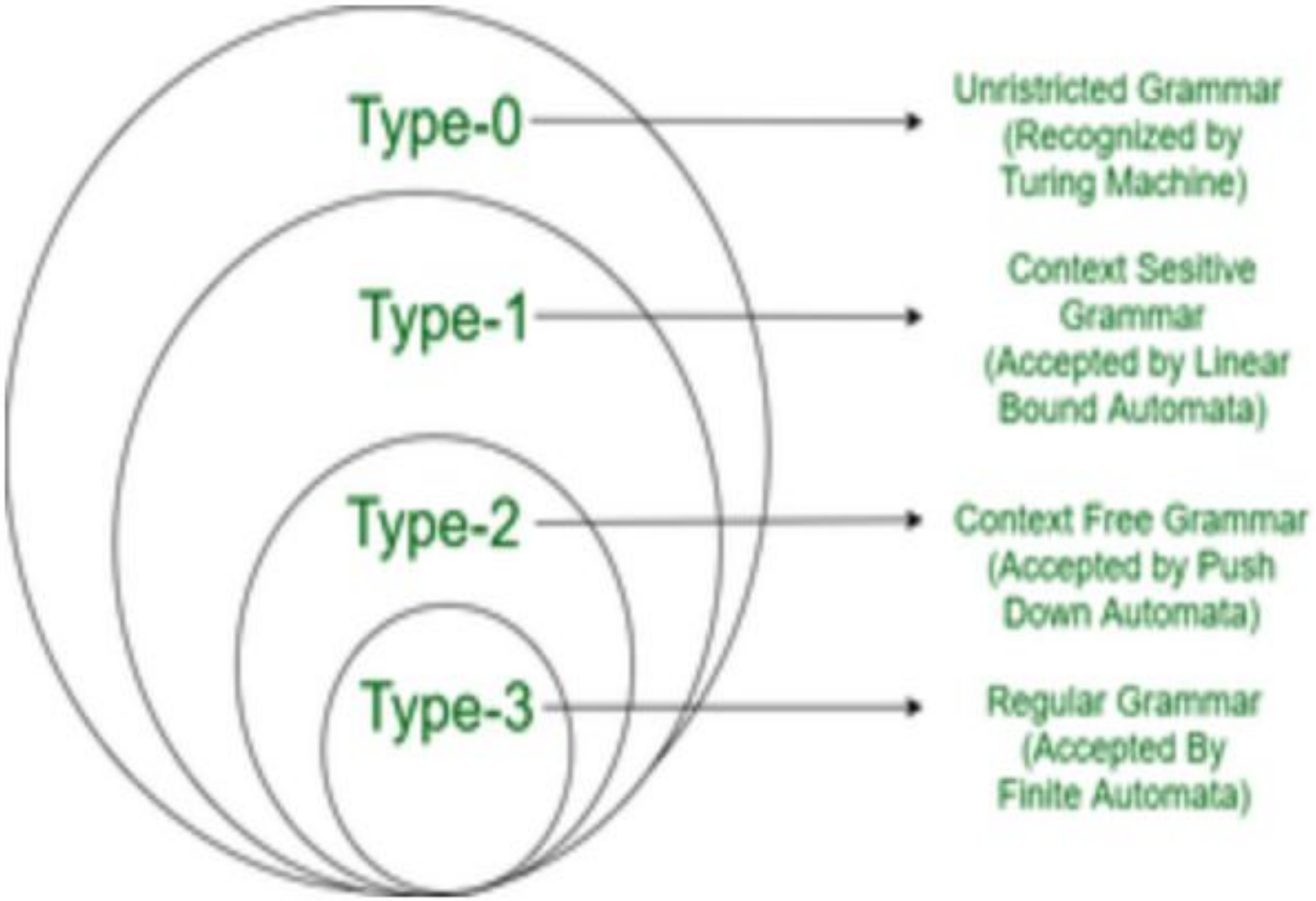
Graphics Software Standards: Important standards include OpenGL, Direct3D, Vulkan, and SVG.

OpenGL (Open Graphics Library): A cross-platform API for rendering 2D and 3D graphics.

Pygame is library available in python.

Remaining topics on Computer Graphics on different slides

Turing Machine …



Chomsky Hierarchy

Turing Machine …

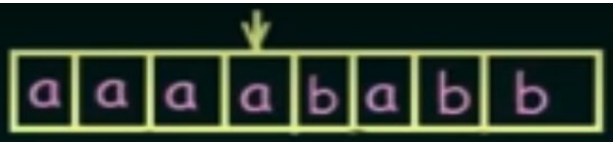
Turing Machine was invented by Alan Turing in 1936 and it is used to accept **Recursive Enumerable Languages** (generated by Type-0 Grammar).

In the context of automata theory and the theory of computation, Turing machines are used to study the properties of algorithms and to determine what problems can and cannot be solved by computers.

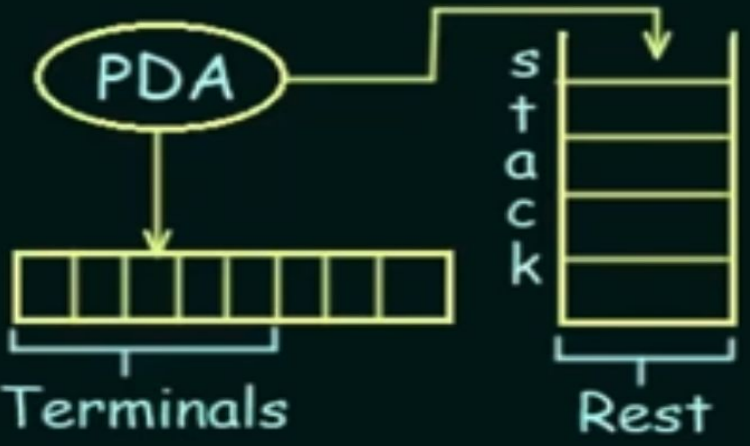
They provide a way to model the behavior of algorithms and to analyze their computational complexity, which is the amount of time and memory they require to solve a problem.

Turing Machine …

FSM: The Input String

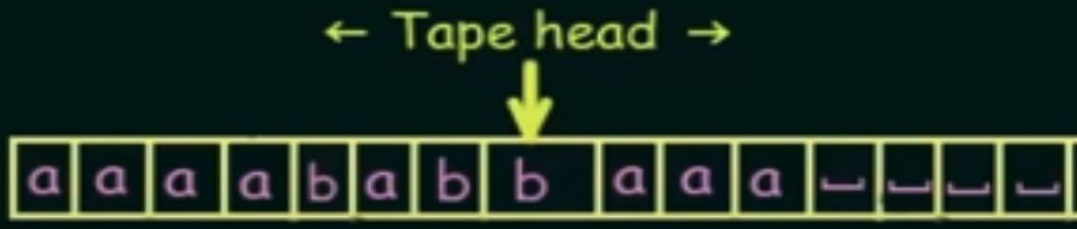


PDA: -> The Input String
-> A Stack



TURING MACHINE:

-> A Tape



Tape Alphabets: $\Sigma = \{ 0,1,a,b,x,Z_0 \}$

The Blank ␣ is a special symbol. ␣ $\notin \Sigma$

The blank is a special symbol used the fill the infinite tape

Turing Machine ...

A **Turing machine** consists of a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ where

Q a finite set of states

Σ the finite input alphabet, not including the blank symbol $\sqcup$

Γ the finite **tape alphabet**, where $\Sigma \subset \Gamma$ and $\sqcup \in \Gamma$

δ a transition function δ that, given the current state and the tape symbol being read, determines the machine's next state, the next tape symbol to be written, and the direction for the tape head to move. In other words,

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}.$$

$q_0 \in Q$ the **initial state**

q_a the **accepting state**

q_r the **rejecting state**

It helps to think of a Turing machine as consisting of a one-way infinite **tape** (i.e. array) of symbols from Γ, where at the beginning of the computation the tape consists of input symbols $w_1 \cdots w_n$, followed by an infinite sequence of $\sqcup$ symbols. The **tape head** reads one tape cell and moves either right or left according to δ. Before moving, it first writes a new symbol on the cell being read.

Turing Machine

Introduction to Turing Machines: A theoretical model for computation, with components like tape, head, state register, and instruction table.

Notations of Turing Machine: Symbols and transition functions define the machine's operations.

Acceptance of a String by a Turing Machine: A string is accepted if the machine reaches a final state.

Turing Machine as a Language Recognizer: Recognizes languages by accepting all strings in the language.

Turing Machine as a Computing Function: Computes functions mapping input strings to output strings.

Turing Machine

Turing Machine with Multiple Tracks: Uses a single tape with multiple tracks to simulate complex computations.

Turing Machine with Multiple Tapes: Uses multiple tapes and heads to simplify complex computations.

Non-Deterministic Turing Machines: Can make multiple moves from a state and symbol, accepting if any path leads to a final state.

Church-Turing Thesis: Any function computable by an algorithm can be computed by a Turing machine.

Universal Turing Machine: Can simulate any other Turing machine, foundational for general-purpose computing.

Turing Machine

Computational Complexity: Studies resources (time and space) needed for Turing Machines to solve problems.

Intractability: Problems that cannot be solved in polynomial time, like NP-complete problems(nondeterministic polynomial time).i.e Travelling Salesman Problem (TSP)

Reducibility: Solving one problem using the solution to another, classifying problems by complexity.

Numerical

Rotate a triangle placed at A(0,0), B(1,1) and C(5,2) by an angle 45 with respect to point P(-1,-1).

The calculations available for computer graphics can be performed only at origin. It is a case of composite transformation which means this can be performed when more than one transformation is performed.

The following composite transformation matrix would be performed as follows.

- First bring the Point P(-1,-1) to the origin => which means translation towards origin => towards origin will be negative translation. So tx and ty values would be negative. So tx =- (-1) = 1 and ty = – (-1) = 1.
- Perform the rotation of 45 degrees.
- Send the point P(-1,-1) back => which mean translation away from origin => Away from origin would be positive translation therefore tx and ty will be positive. So tx = -1 and ty = -1.
- The **most important point: the composite matrix should be written from right to left , So** the composite rotation matrix would be = T(x) * R(Theta) * T(-x) and **NOT** *T(-x) * R(Theta) * T(x)*

The composite rotation matrix would be = Positive translation * Rotation (45degree) * Negative translation.

$$\begin{bmatrix} 1 & 0 & -(-1) \\ 0 & 1 & -(-1) \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \text{Cos}(45) & -\text{Sin}(45) & 0 \\ \text{Sin}(45) & \text{Cos}(45) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & -(1) \\ 0 & 1 & -(1) \\ 0 & 0 & 1 \end{bmatrix}$$

Translation -ve

Rotation Matrix

Translation +ve

Numerical

Composite Matrices

$$\begin{bmatrix} 1 & 0 & -(-1) \\ 0 & 1 & -(-1) \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & -(1) \\ 0 & 1 & -(1) \\ 0 & 0 & 1 \end{bmatrix}$$

Translation -ve

Rotation Matrix

Translation +ve

The composite Transformation

Multiply the resultant rotation matrix with the triangle matrix.

$$\begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & -1 \\ 1/\sqrt{2} & 1/\sqrt{2} & (2/\sqrt{2}) - 1 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} A & B & C \\ 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

Resultant Component Matrix

Triangle Matrix

Numerical

The final resultant matrix will be as follows.

A’	B’	C’
-1	-1	$(3/\sqrt{2})-1$
$(2/\sqrt{2})-1$	$(4/\sqrt{2})-1$	$(9/\sqrt{2})-1$
1	1	1

A’(-1, $(2/\sqrt{2})-1$) , B’ (-1, $(4/\sqrt{2})-1$) and C’ ($(3/\sqrt{2})-1$, $(9/\sqrt{2})-1$)

Best of Luck

UNIT-1 : 2D AND 3D TRANSFORMATION & VIEWING

2D Transformation

Transformation means changing some graphics into something else by applying rules. We can have various types of transformations such as translation, scaling up or down, rotation, shearing, etc. When a transformation takes place on a 2D plane, it is called 2D transformation.

Transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation.

Homogenous Coordinates

To perform a sequence of transformation such as translation followed by rotation and scaling, we need to follow a sequential process −

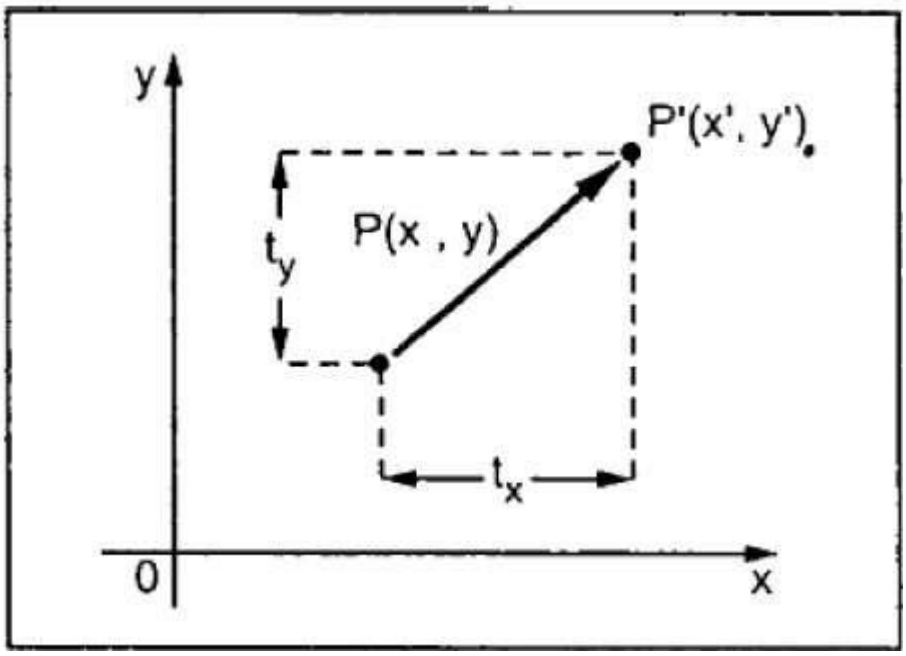
- Translate the coordinates,
- Rotate the translated coordinates, and then
- Scale the rotated coordinates to complete the composite transformation.
-

To shorten this process, we have to use 3×3 transformation matrix instead of 2×2 transformation matrix. To convert a 2×2 matrix to 3×3 matrix, we have to add an extra dummy coordinate W.

In this way, we can represent the point by 3 numbers instead of 2 numbers, which is called **Homogenous Coordinate** system. In this system, we can represent all the transformation equations in matrix multiplication. Any Cartesian point P(X, Y) can be converted to homogenous coordinates by P' (X_h, Y_h, h).

Translation

A translation moves an object to a different position on the screen. You can translate a point in 2D by adding translation coordinate (t_x, t_y) to the original coordinate (X, Y) to get the new coordinate (X', Y').



From the above figure, you can write that –

$$\mathbf{X' = X + t_x}$$

$$\mathbf{Y' = Y + t_y}$$

The pair (t_x, t_y) is called the translation vector or shift vector. The above equations can also be represented using the column vectors.

$$P = \begin{bmatrix} X \\ Y \end{bmatrix} \quad p' = \begin{bmatrix} X' \\ Y' \end{bmatrix} \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

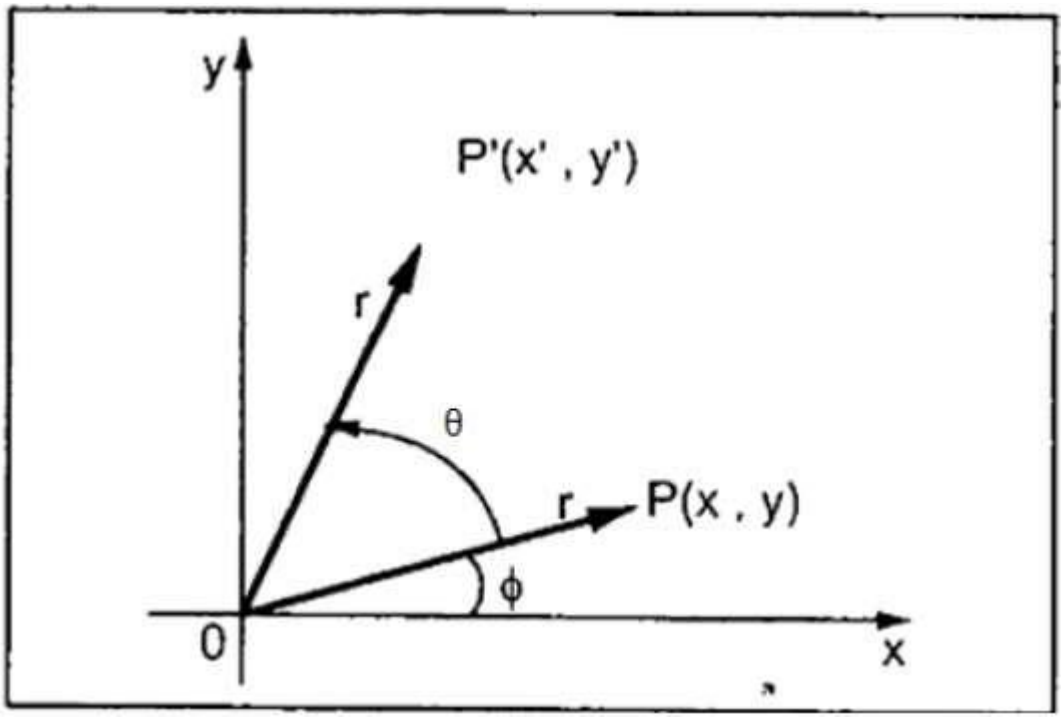
We can write it as –

$$\mathbf{P' = P + T}$$

Rotation

In rotation, we rotate the object at particular angle θ (theta) from its origin. From the following figure, we can see that the point P(X, Y) is located at angle φ from the horizontal X coordinate with distance r from the origin.

Let us suppose you want to rotate it at the angle θ. After rotating it to a new location, you will get a new point P' (X', Y').



Using standard trigonometric the original coordinate of point P(X, Y) can be represented as –

$$X = r\cos\phi \ldots\ldots (1)$$

$$Y = r\sin\phi \ldots\ldots (2)$$

Same way we can represent the point P′ (X′, Y′) as –

$$x' = r\cos(\phi + \theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta \ldots\ldots (3)$$

$$y' = r\sin(\phi + \theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta \ldots\ldots (4)$$

Substituting equation (1) & (2) in (3) & (4) respectively, we will get

$$x' = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta$$

Representing the above equation in matrix form,

$$[X'Y'] = [XY]\begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} OR$$

P′ = P . R

Where R is the rotation matrix

Where R is the rotation matrix

$$R = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix}$$

The rotation angle can be positive and negative.

For positive rotation angle, we can use the above rotation matrix. However, for negative angle rotation, the matrix will change as shown below –

$$R = \begin{bmatrix} cos(-\theta) & sin(-\theta) \\ -sin(-\theta) & cos(-\theta) \end{bmatrix}$$

$$= \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} (\because cos(-\theta) = cos\theta \; and \; sin(-\theta) = -sin\theta)$$

Scaling

To change the size of an object, scaling transformation is used. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result.

Let us assume that the original coordinates are (X, Y), the scaling factors are (S_X, S_Y), and the produced coordinates are (X', Y'). This can be mathematically represented as shown below –

$$\mathbf{X' = X \cdot S_X \text{ and } Y' = Y \cdot S_Y}$$

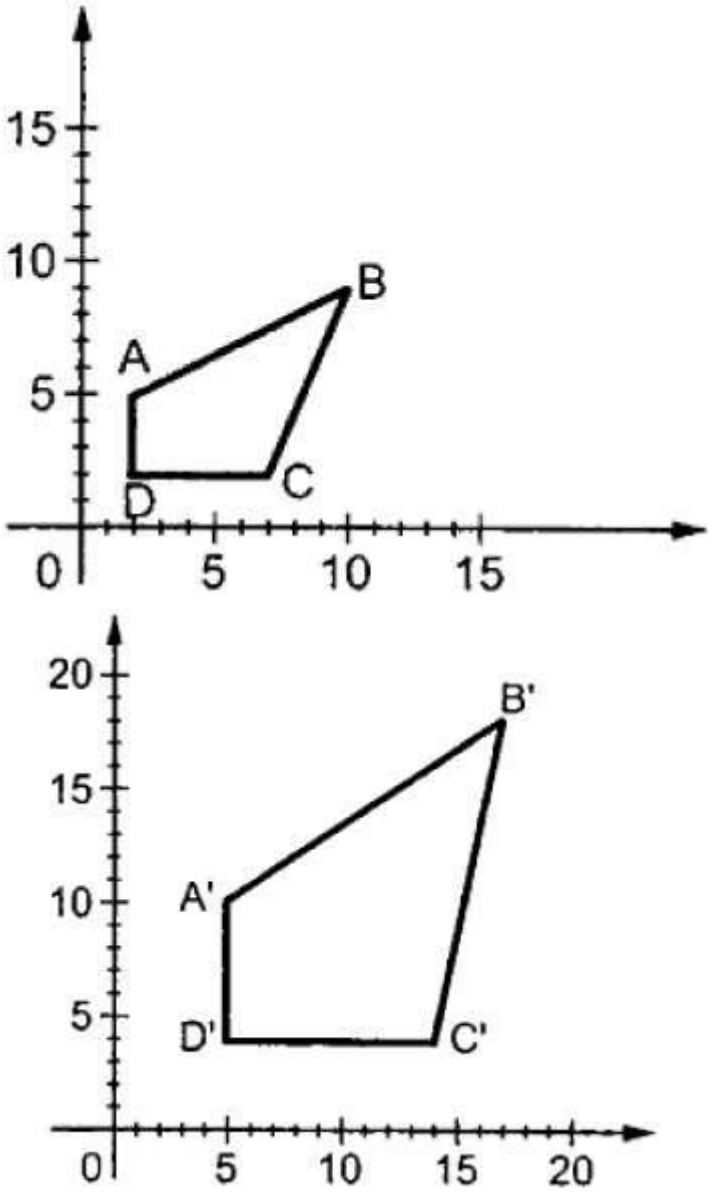
The scaling factor S_X, S_Y scales the object in X and Y direction respectively. The above equations can also be represented in matrix form as below –

$$(X'Y')=(XY)[S_x00S_y](X'Y')=(XY)[Sx00Sy]$$

OR

$$\mathbf{P' = P \cdot S}$$

Where S is the scaling matrix. The scaling process is shown in the following figure.

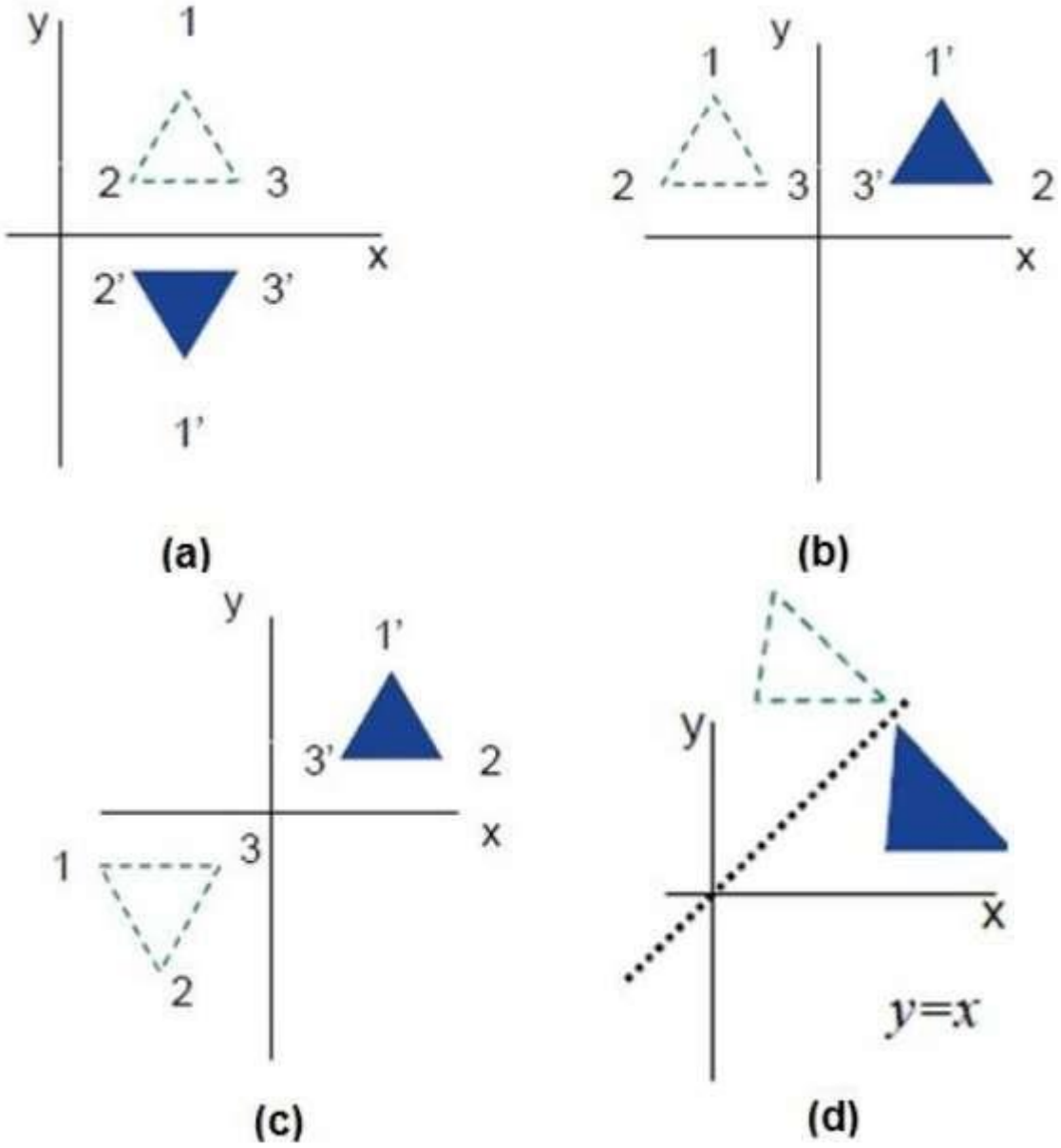


If we provide values less than 1 to the scaling factor S, then we can reduce the size of the object. If we provide values greater than 1, then we can increase the size of the object.

Reflection

Reflection is the mirror image of original object. In other words, we can say that it is a rotation operation with 180°. In reflection transformation, the size of the object does not change.

The following figures show reflections with respect to X and Y axes, and about the origin respectively.

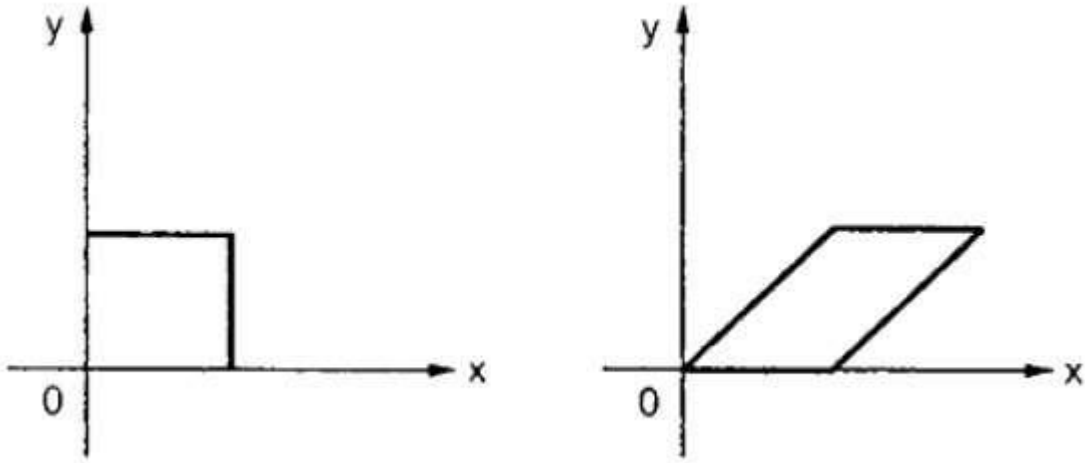


Shear

A transformation that slants the shape of an object is called the shear transformation. There are two shear transformations **X-Shear** and **Y-Shear**. One shifts X coordinates values and other shifts Y coordinate values. However; in both the cases only one coordinate changes its coordinates and other preserves its values. Shearing is also termed as **Skewing**.

X-Shear

The X-Shear preserves the Y coordinate and changes are made to X coordinates, which causes the vertical lines to tilt right or left as shown in below figure.



(a) Original object (b) Object after x shear

The transformation matrix for X-Shear can be represented as –

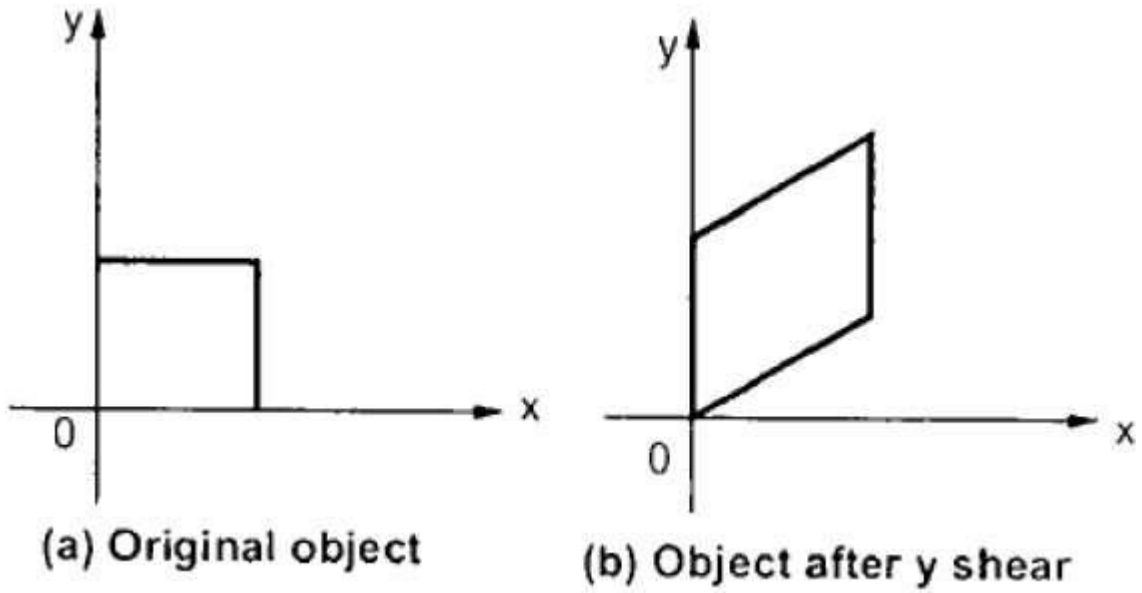
$$X_{sh} = \begin{bmatrix} 1 & shx & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Y' = Y + Sh_y \cdot X$$

$$X' = X$$

Y-Shear

The Y-Shear preserves the X coordinates and changes the Y coordinates which causes the horizontal lines to transform into lines which slopes up or down as shown in the following figure.



(a) Original object (b) Object after y shear

The Y-Shear can be represented in matrix from as –

$$Y_{sh} \begin{bmatrix} 1 & 0 & 0 \\ shy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$X' = X + Sh_X \cdot Y$$

$$Y' = Y$$

Composite Transformation

If a transformation of the plane T1 is followed by a second plane transformation T2, then the result itself may be represented by a single transformation T which is the composition of T1 and T2 taken in that order. This is written as T = T1·T2.

Composite transformation can be achieved by concatenation of transformation matrices to obtain a combined transformation matrix.

A combined matrix –

[T][X] = [X] [T1] [T2] [T3] [T4] …. [Tn]

Where [Ti] is any combination of

- Translation
- Scaling
- Shearing
- Rotation
- Reflection

The change in the order of transformation would lead to different results, as in general matrix multiplication is not cumulative, that is [A] . [B] ≠ [B] . [A] and the order of multiplication. The basic purpose of composing transformations is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformation, one after another.

For example, to rotate an object about an arbitrary point (X_p, Y_p), we have to carry out three steps −

- Translate point (X_p, Y_p) to the origin.
- Rotate it about the origin.
- Finally, translate the center of rotation back where it belonged.

Explain Viewing transformation pipeline

The Viewing Transformation Pipeline:-

We know that the picture is stored in the computer memory using any convenient Cartesian co-ordinate system, referred to as World Co-Ordinate System (WCS). However, when picture is displayed on the display device it is measured in Physical Device Co-Ordinate System (PDCS) corresponding to the display device. Therefore, displaying an image of a picture involves mapping the co-ordinates of the Points and lines that form the picture into the appropriate physical device co-ordinate where the image is to be displayed. This mapping of co-ordinates is achieved with the use of co-ordinate transformation known as viewing transformation.

The viewing transformation which maps picture co-ordinates in the WCS to display co-ordinates in PDCS is performed by the following transformations.

- Converting world co-ordinates to viewing co-ordinates.
- Normalizing viewing co-ordinates.
- Converting normalized viewing co-ordinates to device co-ordinates.

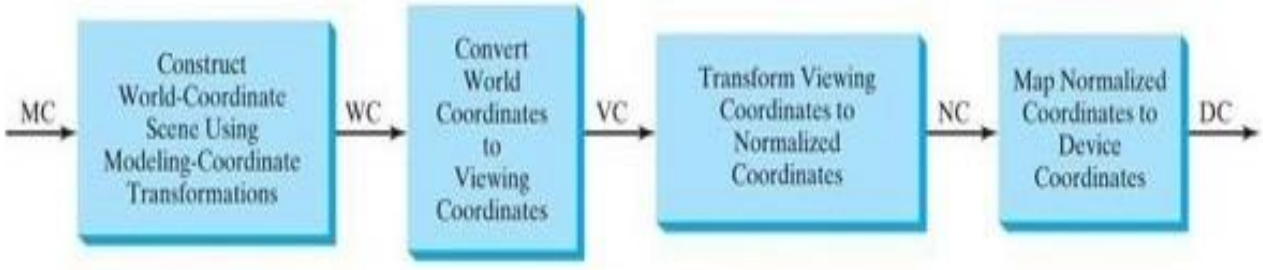
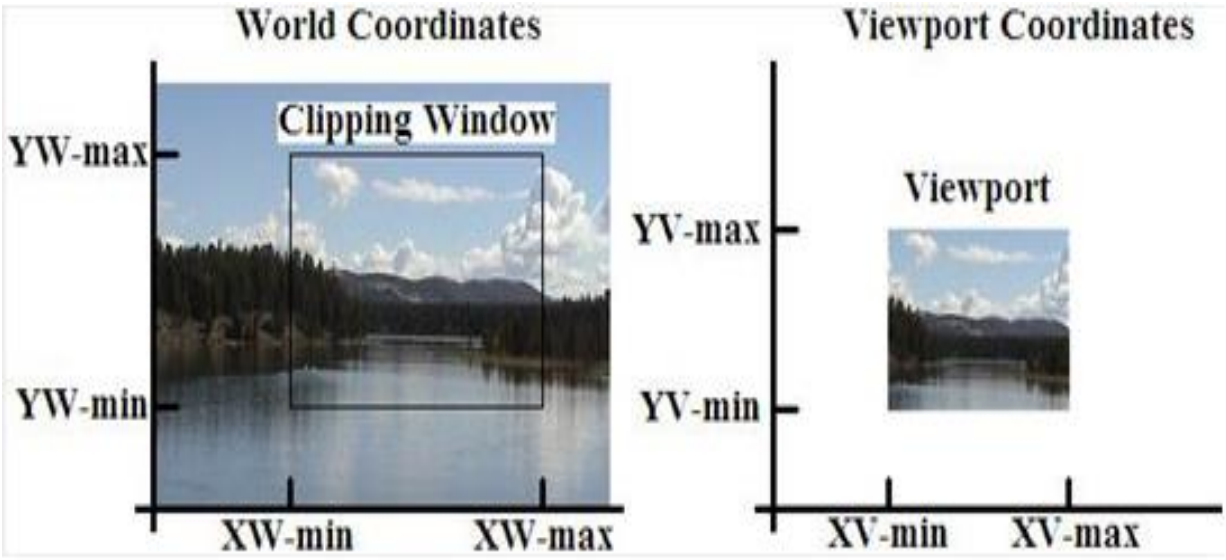


Fig. (c) Two-dimensional viewing transformation pipeline



1. This transformation involves developing formulas that start with a point in the world window, say (xw, yw).
2. The formula is used to produce a corresponding point in viewport coordinates, say (xv, yv). We would like for this mapping to be "proportional" in the sense that if xw is 30% of the way from the left edge of the world window, then xv is 30% of the way from the left edge of the viewport.
3. Similarly, if yw is 30% of the way from the bottom edge of the world window, then yv is 30% of the way from the bottom edge of the viewport. The picture below shows this proportionality.

For proportionality in x:

For proportionality in y:

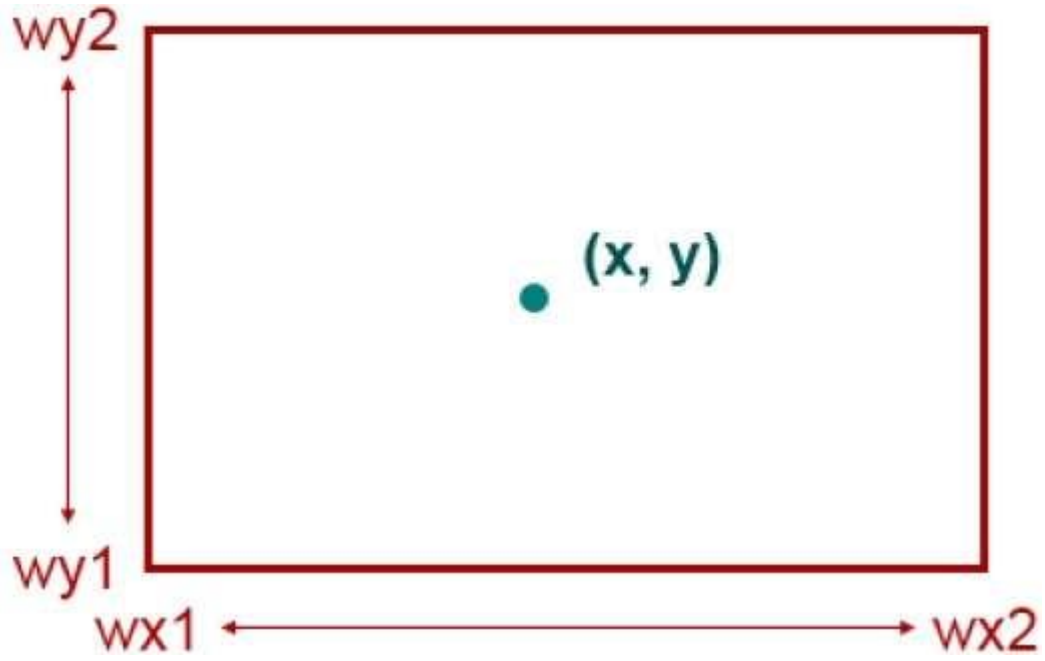
Viewing & Clipping

The primary use of clipping in computer graphics is to remove objects, lines, or line segments that are outside the viewing pane. The viewing transformation is insensitive to the position of points relative to the viewing volume – especially those points behind the viewer – and it is necessary to remove these points before generating the view.

Point Clipping

Clipping a point from a given window is very easy. Consider the following figure, where the rectangle indicates the window. Point clipping tells us whether the given point (X, Y) is within the given window or not; and decides whether we will use the minimum and maximum coordinates of the window.

The X-coordinate of the given point is inside the window, if X lies in between Wx1 ≤ X ≤ Wx2. Same way, Y coordinate of the given point is inside the window, if Y lies in between Wy1 ≤ Y ≤ Wy2.

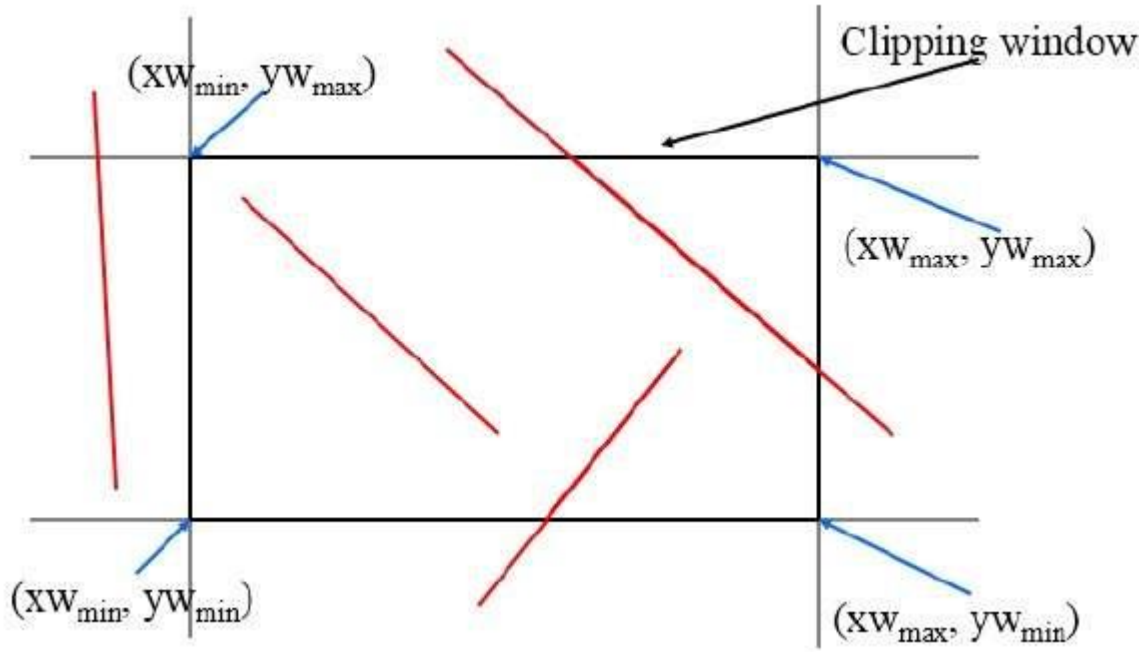


Line Clipping

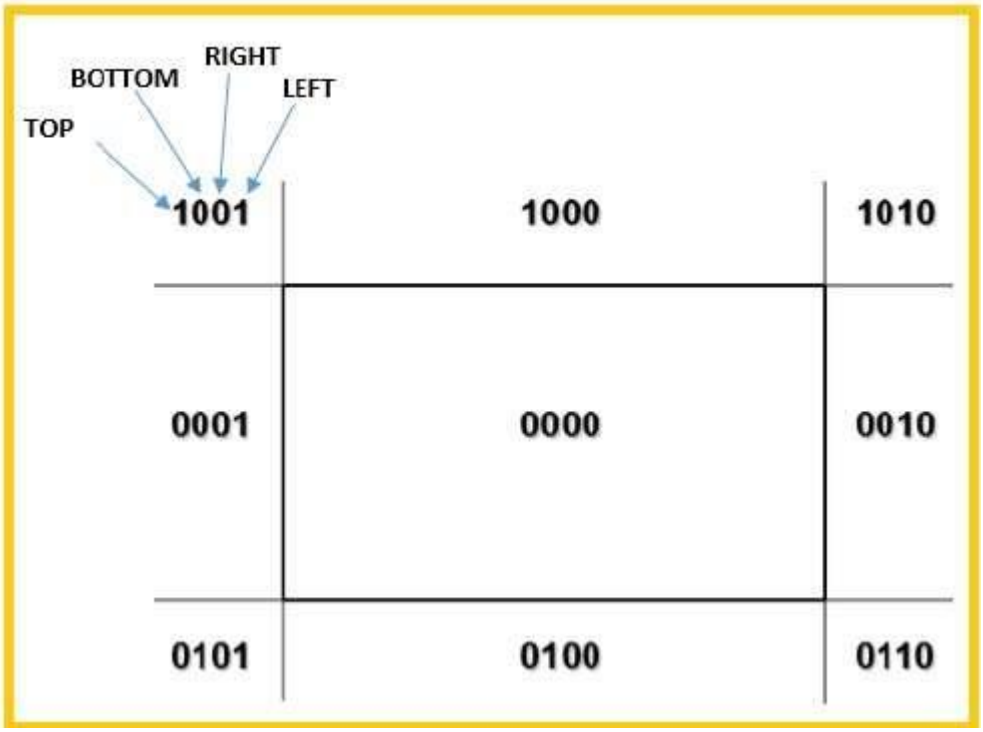
The concept of line clipping is same as point clipping. In line clipping, we will cut the portion of line which is outside of window and keep only the portion that is inside the window.

Cohen-Sutherland Line Clippings

This algorithm uses the clipping window as shown in the following figure. The minimum coordinate for the clipping region is (XW_{min}, YW_{min})(XWmin,YWmin) and the maximum coordinate for the clipping region is (XW_{max}, YW_{max})(XWmax,YWmax).



We will use 4-bits to divide the entire region. These 4 bits represent the Top, Bottom, Right, and Left of the region as shown in the following figure. Here, the **TOP** and **LEFT** bit is set to 1 because it is the **TOP-LEFT** corner.



There are 3 possibilities for the line –

- Line can be completely inside the window (This line should be accepted).
- Line can be completely outside of the window (This line will be completely removed from the region).
- Line can be partially inside the window (We will find intersection point and draw only that portion of line that is inside region).

Algorithm

Step 1 – Assign a region code for each endpoints.

Step 2 – If both endpoints have a region code **0000** then accept this line.

Step 3 – Else, perform the logical **AND**operation for both region codes.

Step 3.1 – If the result is not **0000,** then reject the line.

Step 3.2 – Else you need clipping.

Step 3.2.1 – Choose an endpoint of the line that is outside the window.

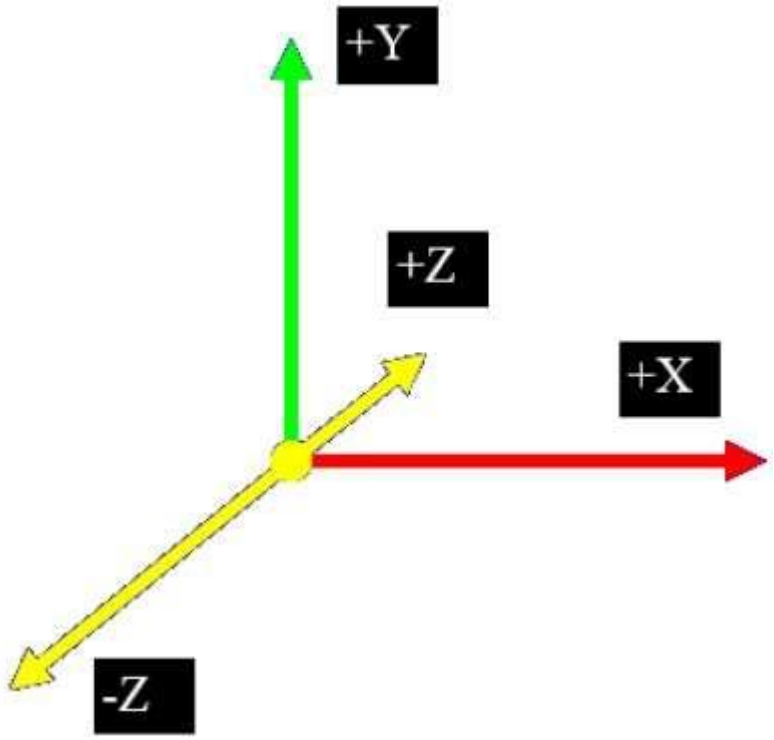
Step 3.2.2 – Find the intersection point at the window boundary (base on region code).

Step 3.2.3 – Replace endpoint with the intersection point and update the region code.

3D Computer Graphics

In the 2D system, we use only two coordinates X and Y but in 3D, an extra coordinate Z is added. 3D graphics techniques and their application are fundamental to the entertainment, games, and computer-aided design industries. It is a continuing area of research in scientific visualization.

Furthermore, 3D graphics components are now a part of almost every personal computer and, although traditionally intended for graphics-intensive software such as games, they are increasingly being used by other applications.

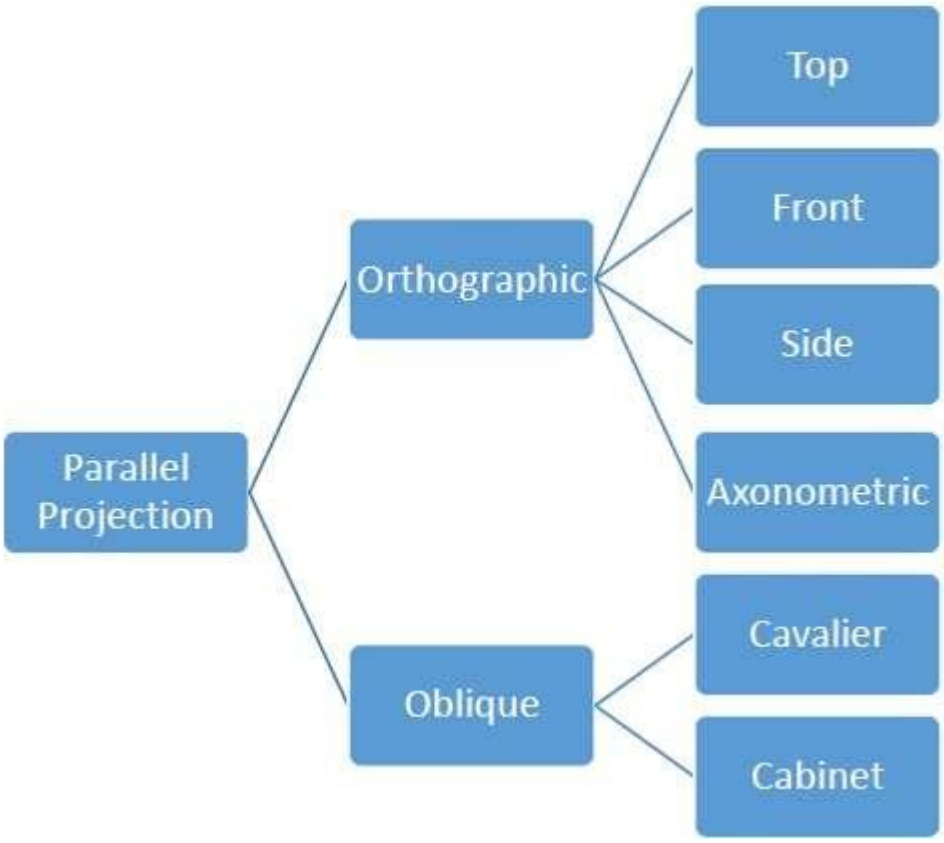


Parallel Projection

Parallel projection discards z-coordinate and parallel lines from each vertex on the object are extended until they intersect the view plane. In parallel projection, we specify a direction of projection instead of center of projection.

In parallel projection, the distance from the center of projection to project plane is infinite. In this type of projection, we connect the projected vertices by line segments which correspond to connections on the original object.

Parallel projections are less realistic, but they are good for exact measurements. In this type of projections, parallel lines remain parallel and angles are not preserved. Various types of parallel projections are shown in the following hierarchy.



Orthographic Projection

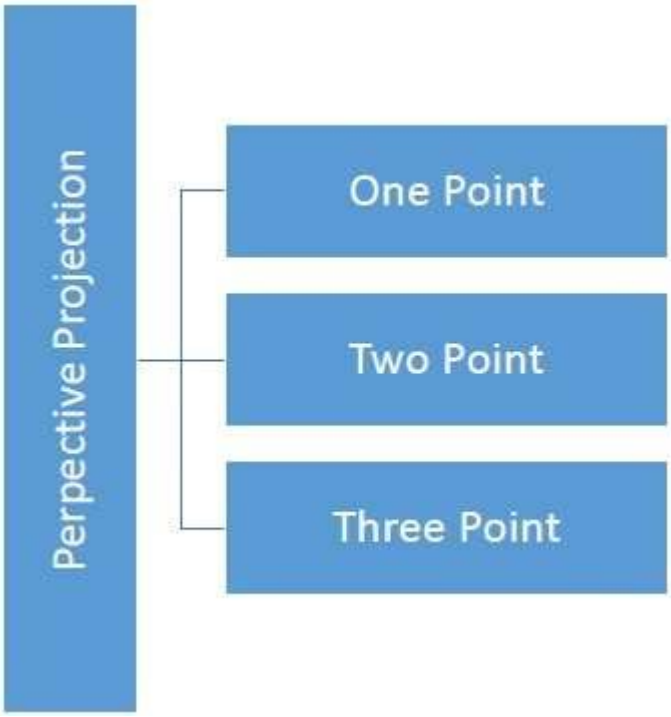
In orthographic projection the direction of projection is normal to the projection of the plane. There are three types of orthographic projections –

- Front Projection
- Top Projection
- Side Projection

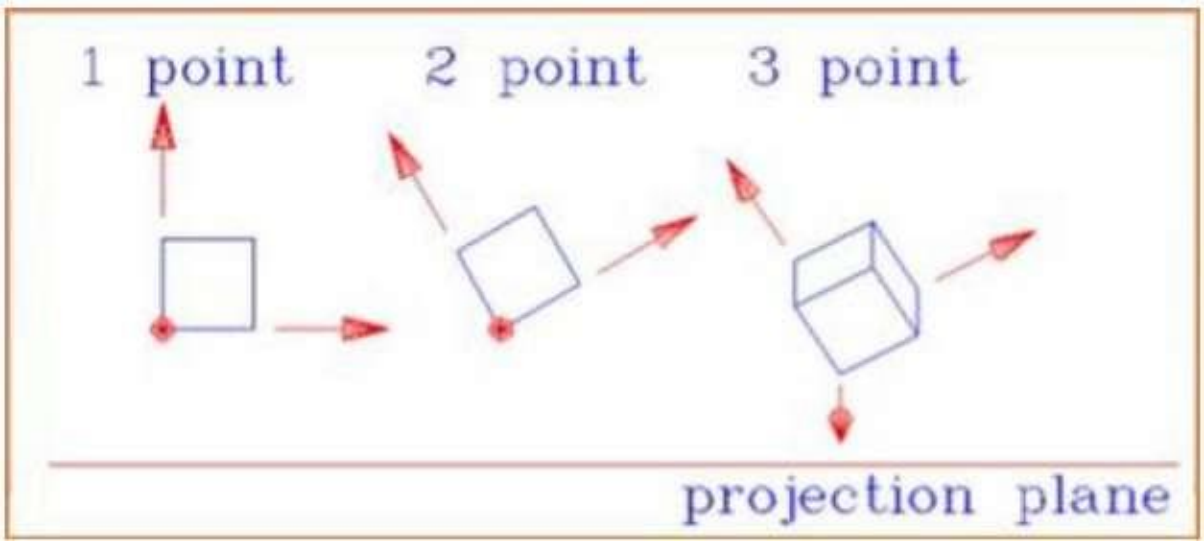
In perspective projection, the distance from the center of projection to project plane is finite and the size of the object varies inversely with distance which looks more realistic.

The distance and angles are not preserved and parallel lines do not remain parallel. Instead, they all converge at a single point called **center of projection** or **projection reference point**. There are 3 types of perspective projections which are shown in the following chart.

- **One point** perspective projection is simple to draw.
- **Two point** perspective projection gives better impression of depth.
- **Three point** perspective projection is most difficult to draw.



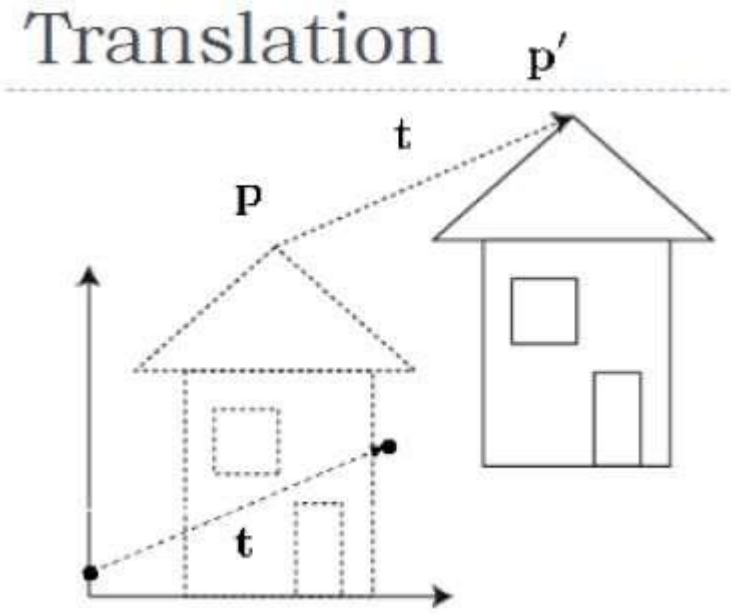
The following figure shows all the three types of perspective projection –



Translation

In 3D translation, we transfer the Z coordinate along with the X and Y coordinates. The process for translation in 3D is similar to 2D translation. A translation moves an object into a different position on the screen.

The following figure shows the effect of translation –



A point can be translated in 3D by adding translation coordinate (t_x, t_y, t_z) to the original coordinate (X, Y, Z) to get the new coordinate (X′, Y′, Z′).

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

$$P' = P \cdot T$$

$$[X'\ Y'\ Z'\ 1] = [X\ Y\ Z\ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

$$= [X + t_x \quad Y + t_y \quad Z + t_z \quad 1]$$

3D Transformation

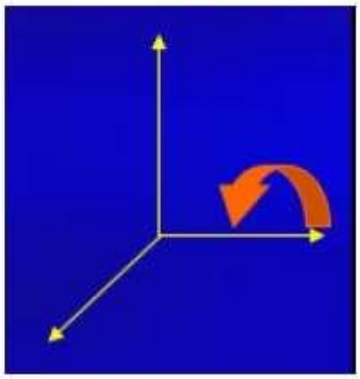
Rotation

3D rotation is not same as 2D rotation. In 3D rotation, we have to specify the angle of rotation along with the axis of rotation. We can perform 3D rotation about X, Y, and Z axes. They are represented in the matrix form as below −

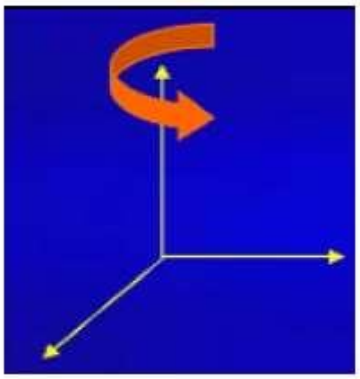
$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_z(\theta)$$

$$= \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

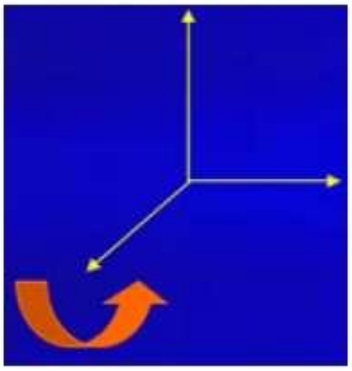
The following figure explains the rotation about various axes –



Rotation about x-axis



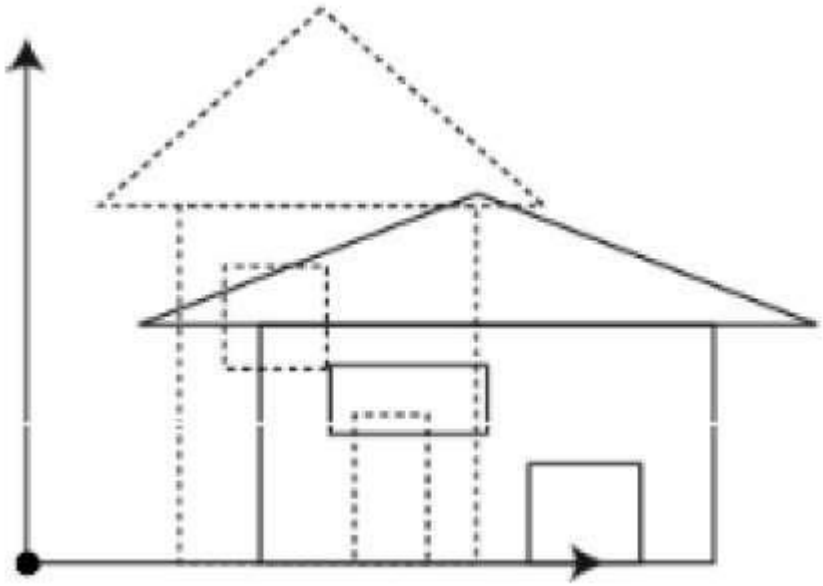
Rotation about y-axis



Rotation about z-axis

Scaling

You can change the size of an object using scaling transformation. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result. The following figure shows the effect of 3D scaling –



In 3D scaling operation, three coordinates are used. Let us assume that the original coordinates are (X, Y, Z), scaling factors are (S_X,S_Y,S_Z)(SX,SY,Sz) respectively, and the produced coordinates are (X', Y', Z'). This can be mathematically represented as shown below –

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

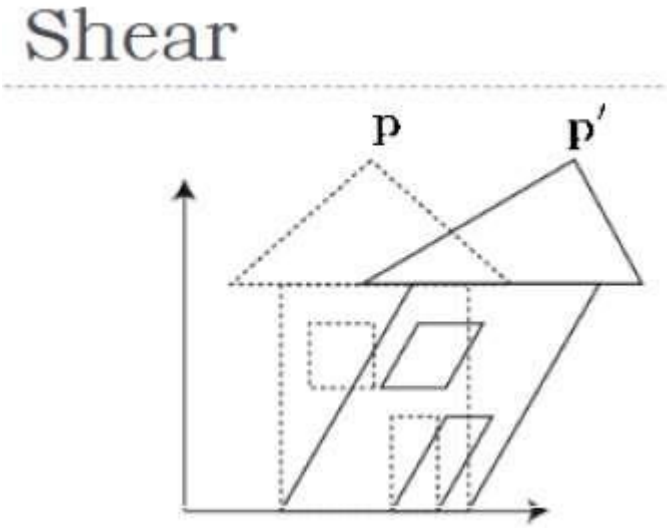
$$P' = P \cdot S$$

$$[X' \;\; Y' \;\; Z' \;\; 1] = [X \;\; Y \;\; Z \;\; 1] \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= [X.S_x \;\; Y.S_y \;\; Z.S_z \;\; 1]$$

Shear

A transformation that slants the shape of an object is called the **shear transformation**. Like in 2D shear, we can shear an object along the X-axis, Y-axis, or Z-axis in 3D.



As shown in the above figure, there is a coordinate P. You can shear it to get a new coordinate P', which can be represented in 3D matrix form as below –

$$Sh = \begin{bmatrix} 1 & sh_x^y & sh_x^z & 0 \\ sh_y^x & 1 & sh_y^z & 0 \\ sh_z^x & sh_z^y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P' = P \cdot Sh$$

$$X' = X + Sh_x^y Y + Sh_x^z Z$$

$$Y' = Sh_y^x X + Y + sh_y^z Z$$

$$Z' = Sh_z^x X + Sh_z^y Y + Z$$

Transformation Matrices

Transformation matrix is a basic tool for transformation. A matrix with n x m dimensions is multiplied with the coordinate of objects. Usually 3 x 3 or 4 x 4 matrices are used for transformation. For example, consider the following matrix for various operation.

$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$	$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$Sh = \begin{bmatrix} 1 & sh_x^y & sh_x^z & 0 \\ sh_y^x & 1 & sh_y^z & 0 \\ sh_z^x & sh_z^y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Translation Matrix	**Scaling Matrix**	**Shear Matrix**
$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$R_z(\theta) = \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Rotation Matrix		

1. Which of the following statements define Computer Graphics?
a) It refers to designing plans
b) It means designing computers
c) It refers to designing images
d) None of the mentioned

2. Among the given scientists/inventor who is known as the father of Computer Graphics?
a) Nikola Tesla
b) Ivan Sutherland
c) Ada Lovelace
d) Marie Curie

3. Which of the following are the features of Computer Graphics?
a) Creation and deletion of images by computer only
b) Deletion and manipulation of graphical images by computer
c) Creation and manipulation of graphics by computer
d) Creation of artificial images by computer only

4. Which of the following is a Computer Graphics type?
a) Raster and Vector
b) Raster and Scalar
c) Scalar only
d) All of the above

5. Who is the first user of computer graphics?
a) William Fetter
b) Ivan Edward Sutherland
c) Ada Lovelace
d) Nicholas Williams

6. Which of the following is the purpose for using clipping in computer graphics?
a) copying
b) zooming
c) adding graphics
d) removing objects and lines

7. In a graphical system, an array of pixels in the picture are stored in which of the following locations?
a) Frame buffer
b) Processor
c) Memory
d) All of the mentioned

8. Bitmap is a collection of _____ that describes an image.
a) pixels
b) algorithms
c) bits
d) colors

9. Which of the following devices provides positional information to the graphics system?
a) Pointing devices
b) Both Input devices and Pointing devices
c) Output devices
d) Input devices

10. Which of the following is defined as the number of pixels stored in the frame buffer of a graphics system?
a) Resalution
b) Resolution
c) Depth
d) None of the mentioned

11. Which of the following is a primary output device of a graphics system?
a) Printer
b) Scanner
c) Video monitor
d) Neither Scanner nor Video monitor

12. Which of the following is used in graphics workstations as input devices to accept voice commands?

a) Speech recognizers
b) Touch panels
c) None of the mentioned
d) All of the mentioned

13. Which of the following is defined as the process of elimination of parts of a scene outside a window or a viewport?
a) editing
b) cutting
c) plucking
d) clipping

14. Which of the following is commonly known as frame buffer on a black and white system with one bit per pixel?
a) Bitmap
b) Pix map
c) Multi map
d) All of the mentioned

15. What does an aspect ratio mean?
a) Ratio of vertical points to horizontal points
b) Ratio of vertical points to horizontal points and horizontal points to vertical points
c) Number of pixels
d) Ratio of horizontal points to vertical points

16. Each screen point is referred to as ..... ?
a) Resolution
b) Pixel
c) Persistence
d) Dot Pitch

17. On a monochromatic monitor, the frame buffer is known as ..... ?
a) Display file
b) Pixmap
c) Bitmap
d) Refresh buffer

18. Color information can be stored in
a) Main memory
b) Secondary memory
c) Graphics card
d) Frame buffer

19. The range that specifies the gray or grayscale levels is
a) The value range from -1 to 1
b) The value range from 0 to -1
c) The value range from 0 to 1
d) Any one of the above

20. In which system, the Shadow mask methods are commonly used
a) Raster-scan system
b) Random-scan system
c) Only b
d) Both a and b

21. he process of digitizing a given picture definition into a set of pixel-intensity for storage in the frame buffer is called
a) Rasterization
b) Encoding
c) Scan conversion
d) True color system

22. Random-scan system mainly designed for
a) Realistic shaded screen
b) Fog effect
c) Line-drawing applications
d) Only b

23. The primary output device in a graphics system is_____
a) Scanner
b) Video monitor
c) Neither a nor b

d) Printer

24. On a black and white system with one bit per pixel, the frame buffer is commonly called as
a) Pix map
b) Multi map
c) Bitmap
d) All of the mentioned

25. What technology does a CRT (Cathode Ray Tube) use to create images on the screen?

a. Liquid crystals

b. Electron beams

c. LEDs

d. Organic LEDs

26. Which of the following is NOT a characteristic of CRT displays?

a. Uses electron beams

b. Slim design

c. Traditional and bulky

d. Lights up phosphors on the screen

27. What does an electron gun emit in a CRT display?

a. Photons

b. Protons

c. Electrons

d. Neutrons

28 . What material is the screen coated with in a CRT display to create the visible image?

a. Liquid crystals

b. Organic LEDs

c. Phosphor

d. Electrons

Turing Machine MCQs

1. What is the primary purpose of a Turing Machine?

a. To solve linear equations

b. To formalize the concept of computation and algorithms

c. To play chess

d.To model physical phenomena

Answer: B) To formalize the concept of computation and algorithms

2**.** In Turing Machine notation, what does the symbol Σ\SigmaΣ represent?

a. Tape alphabet
b. Input alphabet
c. Blank symbol
d. Set of states

Answer: B) Input alphabet

3**.** When does a Turing Machine accept a string?

a. When it enters a non-final state
b. When it halts and does not move
c. When it enters a final state after processing the string
d. When it writes a blank symbol

Answer: C) When it enters a final state after processing the string

4**.** What does a Turing Machine recognize?

a. All possible inputs
b. The shortest possible string
c. All and only the strings in a language
d. Only the longest string

Answer: C) All and only the strings in a language

5**.** What is a Turing Machine doing when it enumerates a language?

a. Sorting strings alphabetically
b. Listing all strings in the language
c. Generating random strings
d. Comparing strings

Answer: B) Listing all strings in the language

6. What is an advantage of a Turing Machine with multiple tracks?

a. It can move faster
b. It can simulate more complex computations
c. It uses less tape
d. It has more states

Answer: B) It can simulate more complex computations

7. What is the primary advantage of a Turing Machine with multiple tapes?

a. It requires less memory
b. It simplifies complex computations
c. It is slower
d. It has fewer states

Answer: B) It simplifies complex computations

8 **.**How does a non-deterministic Turing Machine differ from a deterministic one?

a. It makes a single move from a given state and symbol
b. It can make multiple moves from a given state and symbol
c. It only accepts finite strings
d. It cannot accept strings

Answer: B) It can make multiple moves from a given state and symbol

9**.** What does the Church-Turing Thesis state?

a. Any function computable by an algorithm can be computed by a Turing machine
b. Only numerical functions can be computed by a Turing machine
c. Turing machines are faster than all other computing models
d. Turing machines cannot simulate real-world processes

Answer: A) Any function computable by an algorithm can be computed by a Turing machine

10. What is the significance of a Universal Turing Machine?

a. It is the fastest type of Turing machine
b. It can simulate any other Turing machine
c. It can only solve arithmetic problems
d. It cannot be constructed physically

Answer: B) It can simulate any other Turing machine

11. What does computational complexity study?

a. The speed of hardware components
b. The resources required for a Turing Machine to solve a problem
c. The physical size of Turing machines
d. The number of states in a Turing machine

Answer: B) The resources required for a Turing Machine to solve a problem

12. What is an intractable problem?

a. A problem that cannot be solved
b. A problem that cannot be solved in polynomial time
c. A problem with no inputs
d. A problem with infinite solutions

Answer: B) A problem that cannot be solved in polynomial time

13. Why is reducibility important in computational theory?

a. It helps in optimizing algorithms
b. It helps in classifying problems based on their computational complexity
c. It reduces the number of states in a Turing machine
d. It decreases the tape length

Answer: B) It helps in classifying problems based on their computational complexity

14. Turing machine can be represented using the following tools:

a. Transition graph
b. Transition table
c. Queue and Input tape
d. All of the mentioned

Answer: D) We can represent a turing machine, graphically, tabularly and diagramatically.

15. We translate a two-dimensional point by adding

a. Translation distances
b. Translation difference
c. X and Y
d. Only a

Answer: D) We can translate 2D point by adding translation distances dx and dy.

16. The translation distances (dx, dy) is called as

a. Translation vector
b. Shift vector
c. Both a and b
d. Neither a nor b

Answer C : The translation distances (dx, dy) from its original position is called as translation vector or shift vector.

17. The two-dimensional translation equation in the matrix form is

a) P′=P+T

b) P′=P-T

c) P′=P*T

d) P′=p

Answer A: The 2D translation equation is P′=P+T.

18. _____ is a rigid body transformation that moves objects without deformation.

a) Rotation

b) Scaling

c) Translation

d) All of the mentioned

Answer C : Translation a rigid body transformation that moves objects without deformation.

19. The basic geometric transformations are

a.Translation b. Rotation c. Scaling d.All of the mentioned

Answer D : All of the above

20. Composite transformations increases the number of operations performed in a series of transformation.

a. True
b. False

Answer B