Module- 3

5.A] Explain in detail the logical classification of input devices?

Answer:

From the perspective of logical devices, inputs originate from within the application program.

The two main characteristics that describe the logical behaviour of input devices are:

- The measurements returned by the device to the user program
- The timing of when these measurements are returned

The API defines six classes of logical input devices as follows:

1. **STRING:** This logical device provides ASCII values of input characters to the user program, typically implemented via a physical keyboard.
2. **LOCATOR**: A locator device supplies a position in world coordinates to the user program, usually implemented with pointing devices such as a mouse or trackball.
3. **PICK:** A pick device returns the identifier of an object on the display to the user program. It is generally implemented with the same physical device as the locator but features a separate software interface. In OpenGL, picking can be accomplished through a selection process.
4. **CHOICE:** A choice device enables the user to select from a discrete number of options. In OpenGL, this can be achieved using various widgets provided by the window system. Widgets are graphical interactive components, such as menus, scrollbars, and buttons. For instance, a menu with multiple selections functions as a choice device, allowing the user to pick from 'n' alternatives.
5. **VALUATORS:** These devices provide analog input to the user program. Graphical systems may use boxes or dials for this purpose.
6. **STROKE:** A stroke device returns an array of locations. For example, pressing a mouse button initiates data transfer into a specified array, and releasing the button concludes this transfer.

5.B] Describe in detail the interactive picture construction techniques?

Answer:

In computer graphics, interactive picture construction techniques are essential for creating and manipulating visual content.

These techniques are commonly used in design and painting applications, allowing users to position objects, constrain their alignment, sketch figures, and drag elements around the screen.

Here's a detailed explanation of these techniques:

1. Basic Positioning Methods

- Specify a location for objects or text using coordinate values.
- Interact with the screen using a pointing device, such as a mouse or touchscreen.
- Position the cursor to set coordinates for placing objects.
- **Example**: Clicking on a location with the mouse to position an image or text at that point.

2. Constraints

- Apply rules to input coordinates to ensure alignment or orientation.
- Common constraints include horizontal or vertical alignment.
- Help maintain consistency and precision in layout.
- **Example**: Aligning a line or shape to the nearest horizontal or vertical axis to ensure it is straight.

3. Grids

- Display a series of rectangular lines on the screen.
- Snap input coordinate positions to the nearest intersection of grid lines.
- Aid in accurate positioning and alignment of objects.
- **Example**: A grid overlay in a drawing application that snaps shapes to grid intersections for precise placement.

4. Gravity Field

- Create a field around lines or objects to assist in alignment.
- Snap input positions near a line to the nearest point on the line.
- Facilitate connections and alignments by automatically adjusting positions.

- **Example**: Drawing a line that snaps to the nearest point on an existing line when the cursor is close to it, with a shaded boundary indicating the gravity area.

5. Rubber Band Methods

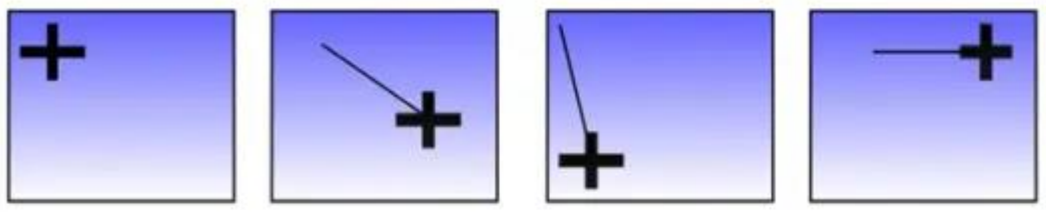
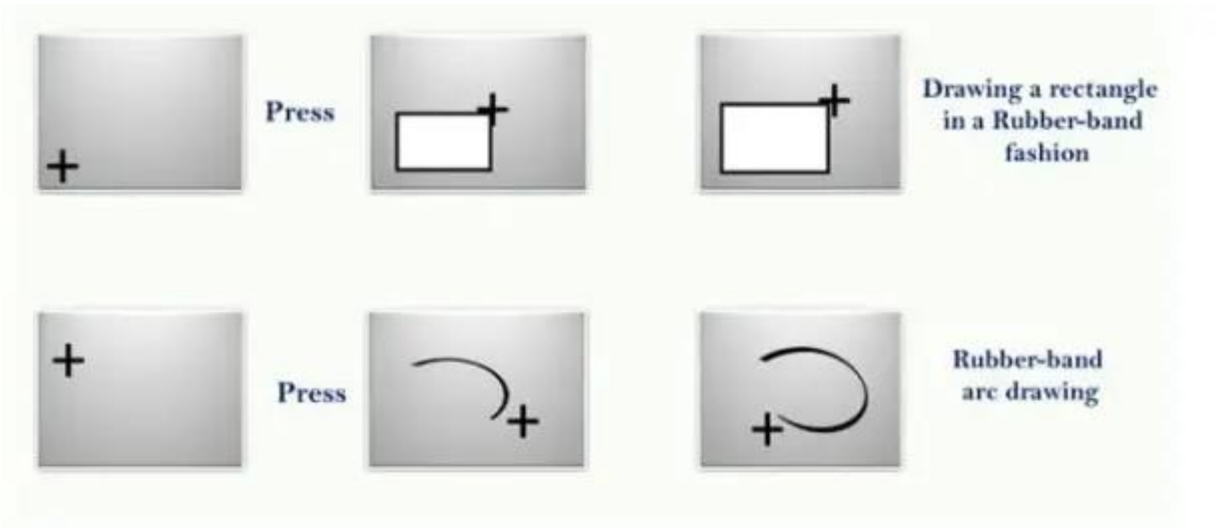
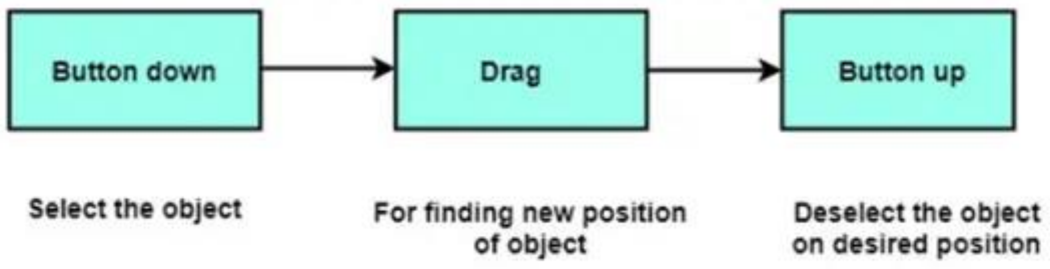


Fig:Demonstration of rubber banding: look at the cross-hair cursor(a) The start point of the line to be drawn is selected (b) 3 different lines are shown depending on the position of the cursor representing the end-point the user selects the desired line.

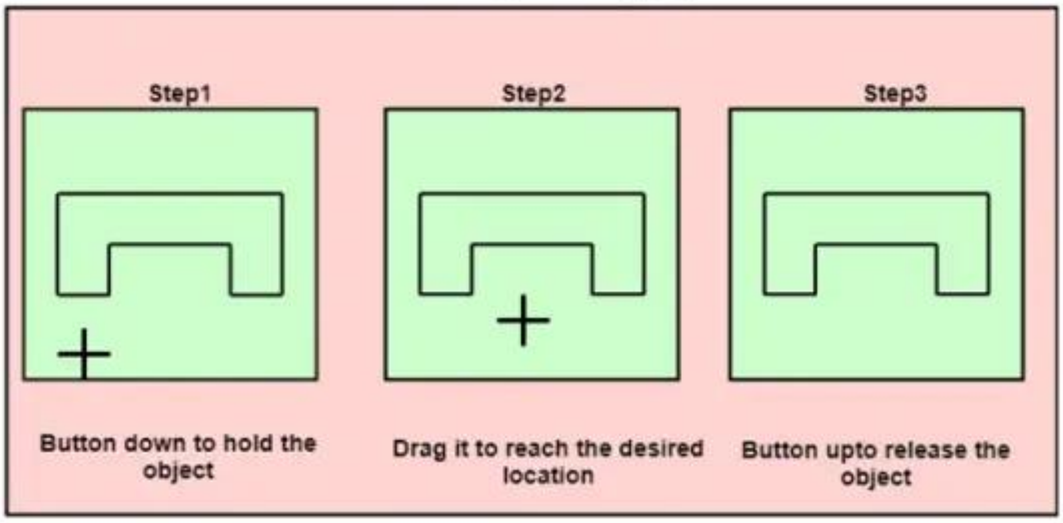


- Stretch a line or shape from a starting position to an endpoint as the cursor moves.
- Allow real-time visualization of the line or shape being created.
- Adjust length and position dynamically based on cursor movement.
- **Example**: Dragging the mouse to draw a line or shape that extends from the initial click to the current cursor position.

6. Dragging


Button down
Drag
Button up
Select the object
For finding new position of object
Deselect the object on desired position

Basic Diagram of Dragging


Step1
Step2
Step3
Button down to hold the object
Drag it to reach the desired location
Button upto release the object

- Move objects by clicking and holding the mouse button while dragging the cursor.
- Reposition objects precisely and intuitively on the screen.
- **Example**: Clicking and dragging an image to move it to a different location on the canvas.

7. Painting and Drawing

- Provide tools for creating and modifying graphical content.
- Options include standard shapes (e.g., circles, arcs) or freehand drawing.
- Adjust attributes like line widths and styles for enhanced customization.
- **Example**: Using a paintbrush tool to create freehand strokes or applying predefined shapes like circles and squares, with options to adjust line thickness and style.

5.B] Explain OpenGL interactive input device functions:
i) GLUT Keyboard Functions
ii) GLUT Mouse Functions

Answer:

In OpenGL, interactive input device functions are used to handle user inputs from keyboards and mice. These functions are part of the GLUT (OpenGL Utility Toolkit) library, which simplifies the creation of OpenGL applications and handling of user inputs.

Here's an explanation of the GLUT keyboard and mouse functions:

i) GLUT Keyboard Functions

GLUT provides several functions to handle keyboard input. These functions allow you to respond to key presses and releases, enabling interactive controls within an OpenGL application.

glutKeyboardFunc(void (*func)(unsigned char key, int x, int y))

- **Description**: Registers a callback function to handle standard ASCII keyboard input.
- **Parameters**:
  - func: A pointer to the callback function that will be called when a key is pressed.
  - key: The ASCII value of the key pressed.
  - x, y: The current mouse coordinates when the key is pressed.
- **Usage**: Used to capture input from standard keys (e.g., 'a', 'b', '1').

ii) GLUT Mouse Functions

GLUT also provides several functions to handle mouse input, allowing users to interact with OpenGL applications using mouse clicks and movements.

glutMouseFunc(void (*func)(int button, int state, int x, int y))

- **Description**: Registers a callback function to handle mouse button events.
- **Parameters**:
  - func: A pointer to the callback function that will be called on mouse button presses and releases.
  - button: The mouse button pressed or released (e.g., GLUT_LEFT_BUTTON, GLUT_RIGHT_BUTTON).
  - state: The state of the button (e.g., GLUT_DOWN, GLUT_UP).
  - x, y: The mouse coordinates when the button event occurs.
- **Usage**: Used to detect and respond to mouse button clicks and releases.

6.B] How are menus and submenus created in OpenGL? Illustrate with an example?

Answer:

Menus are an important feature of any application program. OpenGL provides a feature called ***"Pop-up-menus"*** using which sophisticated interactive applications can be created.

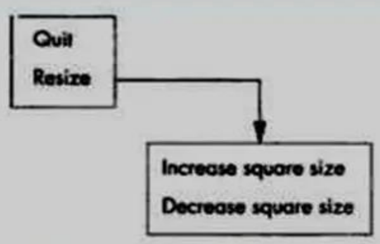
Creating Menus and Submenus in OpenGL

1. **Define the Menu Callback Function**
   - Create a callback function that will handle menu item selections. This function will be called whenever a menu item is selected by the user.
2. **Create Menu Items**
   - Use the glutCreateMenu function to create a menu and specify the callback function. Then, use glutAddMenuEntry to add individual menu items.
3. **Create Submenus**
   - Create a submenu using glutCreateMenu and add items to it using glutAddMenuEntry. Attach this submenu to a parent menu using glutAddSubMenu.
4. **Attach the Menu to a Mouse Button**
   - Use glutAttachMenu to attach the menu to a mouse button (usually the right mouse button).

```
// Function to create the menus
void createMenu() {
   int submenu;

   // Create a submenu
   submenu = glutCreateMenu(menu);
   glutAddMenuEntry("Submenu Item 1", 3);
   glutAddMenuEntry("Submenu Item 2", 4);

   // Create the main menu
```

```
    glutCreateMenu(menu);
    glutAddMenuEntry("Menu Item 1", 1);
    glutAddMenuEntry("Menu Item 2", 2);
    glutAddSubMenu("Submenu", submenu);
    glutAddMenuEntry("Exit", 5);

    // Attach the menu to the right mouse button
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}
// Function to handle menu item selections
void menu(int item) {
    switch (item) {
        case 1:
            printf("Menu Item 1 selected\n");
            break;
        case 2:
            printf("Menu Item 2 selected\n");
            break;
        case 3:
            printf("Submenu Item 1 selected\n");
            break;
        case 4:
            printf("Submenu Item 2 selected\n");
            break;
        case 5:
            printf("Exit selected\n");
            exit(0);
            break;
    }
```

```
}
```



```
sub_menu = glutCreateMenu(size_menu);
glutAddMenuEntry("Increase square size", 2);
glutAddMenuEntry("Decrease square size", 3);
glutCreateMenu(top_menu);
glutAddMenuEntry("Quit",1);
glutAddSubMenu("Resize", sub_menu);
glutAttachMenu(GLUT_RIGHT_BUTTON);
```

Figure: Structure of hierarchical menus

Full Code:-

```
#include <GL/glut.h>
#include <stdio.h>

// Function to handle menu item selections
void menu(int item) {
    switch (item) {
        case 1:
            printf("Menu Item 1 selected\n");
            break;
        case 2:
            printf("Menu Item 2 selected\n");
            break;
        case 3:
            printf("Submenu Item 1 selected\n");
            break;
        case 4:
            printf("Submenu Item 2 selected\n");
            break;
        case 5:
            printf("Exit selected\n");
            exit(0);
            break;
    }
}

// Function to create the menus
void createMenu() {
    int submenu;

    // Create a submenu
    submenu = glutCreateMenu(menu);
    glutAddMenuEntry("Submenu Item 1", 3);
    glutAddMenuEntry("Submenu Item 2", 4);

    // Create the main menu
    glutCreateMenu(menu);
    glutAddMenuEntry("Menu Item 1", 1);
    glutAddMenuEntry("Menu Item 2", 2);
    glutAddSubMenu("Submenu", submenu);
    glutAddMenuEntry("Exit", 5);

    // Attach the menu to the right mouse button
    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

// Display function (for rendering)
void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}
```

```
// Main function
int main(int argc, char** argv) {
   // Initialize GLUT
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(500, 500);
   glutCreateWindow("OpenGL Menu Example");

   // Set up the display function
   glutDisplayFunc(display);

   // Create the menu
   createMenu();

   // Start the GLUT event loop
   glutMainLoop();
   return 0;
}
```

Explanation

1. **Menu Callback Function (menu)**
   - This function is called whenever a menu item is selected. It handles different cases based on the item selected.
2. **Creating Menus (createMenu)**
   - A submenu is created first using glutCreateMenu and populated with items using glutAddMenuEntry.
   - The main menu is created and populated with items, including the submenu using glutAddSubMenu.
   - Finally, the menu is attached to the right mouse button using glutAttachMenu.
3. **Display Function (display)**

    - This function clears the screen and is required by GLUT, though it doesn't do much in this example.

4. **Main Function (main)**
    - Initializes GLUT, sets up the window, and starts the event loop.

6.B] Write a note on OpenGL Animation Procedures?

Answer:

OpenGL Animation Procedures

OpenGL provides several methods to create smooth animations. Here's a simplified overview of the main procedures:

Double Buffering:

- **Purpose**: Double buffering helps to avoid flickering and tearing in animations by using two buffers for drawing. This ensures that the animation looks smooth.
- **How to Activate**: Use the command:
  glutInitDisplayMode(GLUT_DOUBLE);
  This sets up two buffers: one for displaying the current frame (front buffer) and one for drawing the next frame (back buffer).
- **Swapping Buffers**: To make the newly drawn frame visible, use:
  glutSwapBuffers();
  This swaps the front and back buffers.
- **Checking Availability**: To see if double buffering is supported,
  use: GLboolean status; glGetBooleanv(GL_DOUBLEBUFFER, &status);
    - GL_TRUE means double buffering is available.
    - GL_FALSE means it is not available.

Continuous Animation with Idle Function:

- **Purpose**: The glutIdleFunc function lets you continuously run an animation function when there are no other events to process.
- **Setting Up**: Use:
  glutIdleFunc(animationFcn);
    - Replace animationFcn with the name of your function that updates the animation.

- **Stopping Animation**: To stop the animation function from running, use: glutIdleFunc(NULL);

These procedures help make animations smooth and keep the display updated efficiently. Double buffering reduces visual glitches, while the idle function ensures that the animation keeps running when the system is idle.

5.A] Define computer animation. Explain the stages to design animation sequences.

Answer:

- Computer animation generally refers to any time sequence of visual changes in a picture.
- In addition to changing object positions using translations or rotations, a computer- generated animation could display time variations in object size, color, transparency, or surface texture.

Two basic methods for constructing a motion sequence are

1. **real-time animation**
    - In a real-time computer-animation, each stage of the sequence is viewed as it is created.
    - Thus the animation must be generated at a rate that is compatible with the constraints of the refresh rate.
2. **frame-by-frame animation**
    - For a frame-by-frame animation, each frame of the motion is separately generated and stored.
    - Later, the frames can be recorded on film, or they can be displayed consecutively on a video monitor in "real-time playback" mode.

Animation Sequence Design Steps:-

Storyboard Layout:

- Visual outline of the animation sequence.
- Includes rough sketches or a list of ideas representing key moments and transitions.
- Helps plan the overall flow and action of the animation.

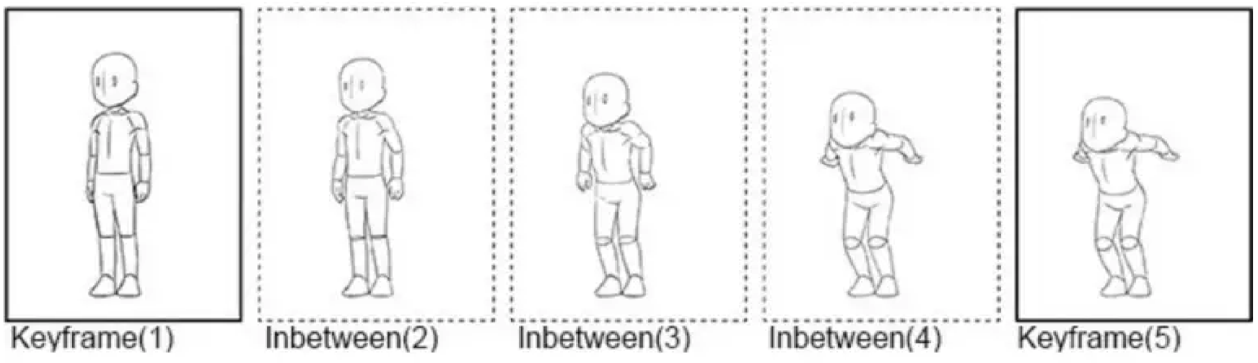
Object Definitions:

- Define objects to be animated using basic shapes (e.g., splines, polygons).
- Specify movements and transformations for each object.
- Ensures accurate representation and movement of objects.

Key-Frame Specifications:

- Detailed drawings or representations of the scene at specific times.
- Capture critical moments and significant changes in the animation.
- Key frames are placed at extreme positions or major points, with more key frames for smooth, fast motions.

Generation of In-Between Frames:



- Frames between key frames that create smooth transitions.
- Number of in-betweens depends on media format (e.g., film needs 24 frames per second).
- Maintains smooth animation by filling gaps between key frames.
- Key frames may be duplicated based on motion speed to maintain the desired frame rate.

Additional Tasks:

- Synchronize with audio, such as background music, sound effects, or voiceovers.
- Ensures that the animation's audio elements match the timing and action of the animation.

Raster Methods for Computer Animation

1. **Creating Animation Sequences**

   - **Real-Time Animation:** Simple animation sequences can be created in real-time by continuously updating and displaying frames. This involves generating each frame on the fly, ensuring that the animation appears smooth and continuous.
   - **Frame-by-Frame Animation:** Alternatively, frames can be created one at a time and saved to files. These frames can later be viewed sequentially or transferred to film for playback.

2. **Displaying Animation**
   - **Sequential Frame Viewing**: After creating the frames, they can be cycled through to display the animation. This method is often used for playback of pre-rendered sequences.
   - **Real-Time Frame Generation:** For continuous animation, frames must be generated quickly enough to maintain smooth motion. This requires efficient rendering techniques to keep up with the display refresh rate.
3. **Double Buffering:** Double buffering is a technique used to produce real-time animation by managing two buffers that handle the frame display and frame creation processes. This technique helps avoid flickering and ensures smooth transitions between frames.
   - **Process:**
     1. **Initialize Buffers:** Two buffers (front buffer and back buffer) are used. The front buffer is currently being displayed on the screen, while the back buffer is where the next frame is being constructed.
     2. **Draw Frames:** While the front buffer is being displayed, the next frame is drawn into the back buffer.
     3. **Swap Buffers:** Once the next frame is complete, the roles of the buffers are swapped. The back buffer becomes the front buffer (now displayed), and the former front buffer is now used for drawing the subsequent frame.
     4. **Repeat:** This process continues, allowing the animation to be displayed smoothly with minimal flicker.
   - **Advantages:**
     - **Eliminates Flicker:** By drawing to a back buffer while the front buffer is displayed, flickering that can occur when updating the display is reduced or eliminated.

        - **Smooth Animation:** Ensures that the transition between frames is smooth, as the entire frame is updated in one go.
    - **Graphics Library Functions:**
        - **Activating Double Buffering:** Libraries like OpenGL provide functions to enable double buffering. For example, in OpenGL, the glutInitDisplayMode(GLUT_DOUBLE) function activates double buffering.
        - **Buffer Swapping:** The glutSwapBuffers() function is used to swap the front and back buffers, making the newly drawn frame visible on the screen.

Module- 5

10 a] What is image segmentation? Classify the image segmentation algorithms.

Image segmentation is the process of partitioning an image into multiple segments or regions, where each segment consists of pixels that share similar attributes such as color, intensity, or texture. The main objective is to simplify the image's representation, making it easier to analyze and process. Segmentation is often used to identify objects, boundaries, and other relevant structures within an image. It is a crucial step in many computer vision tasks, including object detection, recognition, and image editing. Techniques for segmentation range from simple thresholding to more complex methods like clustering, edge detection, and deep learning-based approaches.

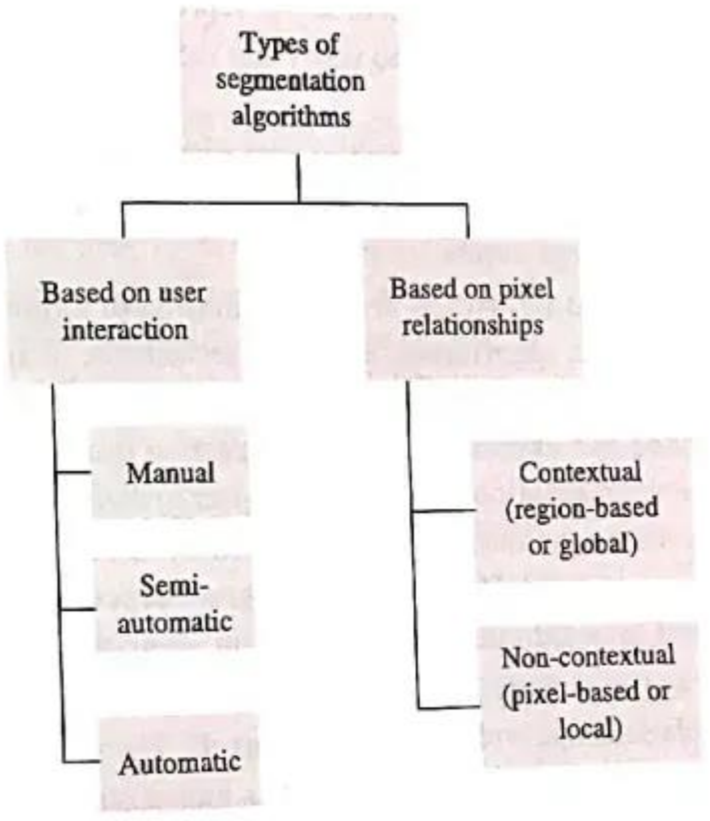

Types of segmentation algorithms
Based on user interaction
Based on pixel relationships
Manual
Semi-automatic
Automatic
Contextual (region-based or global)
Non-contextual (pixel-based or local)

Fig. 9.2 Classification of segmentation algorithms

Classification of Image Segmentation

Image segmentation methods can be classified into three categories based on the level of user interaction:

1. **Manual Segmentation:** In this approach, an expert manually traces the boundaries of the region of interest (ROI) using specialized software. The expert makes all segmentation decisions, and the software may assist by connecting open contours into closed regions, which are then converted into control points and connected by splines. While accurate, manual segmentation is time-consuming, subjective, and prone to variations among different observers.
2. **Automatic Segmentation:** These algorithms perform segmentation without any human intervention, making them ideal for processing large numbers of images. Automatic methods analyze the image and segment structures based on predefined criteria, offering consistent results with minimal user input.
3. **Semi-Automatic Segmentation:** This approach combines both manual and automatic methods. Human intervention is required at the initial stage, typically to provide seed points indicating the ROI. The algorithm then automatically completes the segmentation process. Region-growing

techniques are a common example, where the program processes the segmentation after the initial seeds are placed by the user.

Segmentation algorithms can also be classified based on pixel similarity relationships:

1. **Contextual (Region-Based or Global) Algorithms:** These methods group pixels based on common properties, such as color or texture, by exploiting the relationships among neighboring pixels. They are also known as global algorithms because they consider the entire image context.
2. **Non-Contextual (Pixel-Based or Local) Algorithms:** These algorithms focus on individual pixel properties, ignoring their relationships with neighboring pixels. They aim to identify discontinuities in the image, such as edges or isolated lines, and group them into regions accordingly.

9 b] Explain the basic types of discontinuities in a digital image.

In digital image processing, discontinuities refer to abrupt changes in pixel intensity values, which often correspond to significant features in an image, such as edges, lines, and points. These discontinuities are critical for detecting and analyzing various image features. The basic types of discontinuities in a digital image are:

1. Point Discontinuities

- **Description:** Point discontinuities occur at single pixels where there is a sharp change in intensity compared to the surrounding pixels. These points are often isolated and can be caused by noise, small image details, or features like stars in an astronomical image.
- **Detection:**
  - Point discontinuities can be detected using techniques like the **Laplacian operator** or **high-pass filtering**. These methods emphasize areas with rapid changes in intensity, making isolated points stand out.
  - **Thresholding** is often applied after filtering to identify significant point discontinuities by setting a threshold value above which the intensity changes are considered significant.
- **Applications:** Point discontinuity detection is used in various fields, such as astronomy (to detect stars or cosmic rays), medical imaging (to identify small lesions), and quality control (to detect defects on surfaces).

2. Line Discontinuities

- **Description:** Line discontinuities occur when there is a sudden change in intensity along a narrow, linear region of the image. These lines can be straight or curved, and they represent features such as roads, wires, or the edges of thin objects.
- **Detection:**
    - Line detection often involves using **convolution masks** designed to respond to linear structures in the image. Common masks include the **Sobel operator** and **Prewitt operator**, which calculate gradients in specific directions to highlight lines.
    - Another approach is the **Hough Transform**, which is particularly effective for detecting straight lines by transforming points in the image into a parameter space and identifying lines that correspond to peaks in this space.
- **Applications:** Line detection is crucial in applications like road detection in aerial images, fingerprint recognition, and feature extraction in various pattern recognition tasks.

3. Edge Discontinuities

- **Description:** Edge discontinuities are the most significant and commonly analyzed type of discontinuity in digital images. They represent the boundaries between different regions, such as the border between an object and its background. Edges are often associated with significant changes in intensity, color, or texture.
- **Detection:**
    - Edge detection typically involves computing the gradient of the image intensity function. The gradient represents the rate of change of intensity and points in the direction of the greatest increase.
    - **Gradient-based methods** like the **Sobel**, **Prewitt**, and **Roberts operators** calculate the first derivative of the intensity function to identify edges.
    - **Laplacian-based methods** use the second derivative to detect zero-crossings, which indicate edges.
    - **Canny Edge Detector** is one of the most popular edge detection algorithms. It uses a multi-stage process that includes noise reduction (using Gaussian filtering), gradient calculation, non-

maximum suppression, and edge tracking by hysteresis to detect edges more accurately.

- **Applications:** Edge detection is a fundamental step in many computer vision and image processing tasks, including object recognition, segmentation, medical imaging, and image enhancement.

9 a] Explain in detail stages of Edge detection process with block diagram.

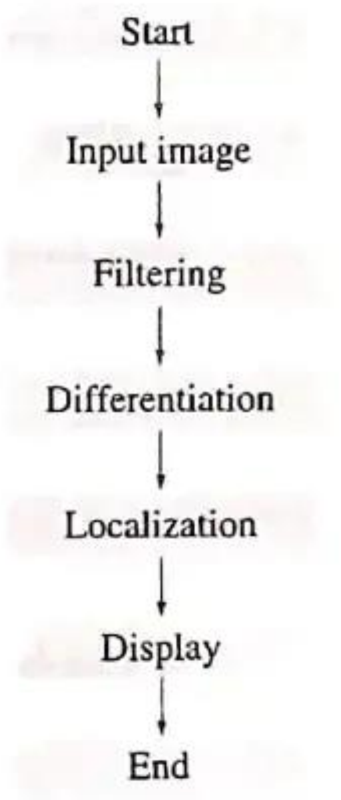


Fig. 9.8 Edge detection process

The diagram represents the edge detection process, which is a crucial step in computer vision and image processing. The goal of edge detection is to identify points in a digital image where the brightness changes sharply, which typically corresponds to object boundaries. Here's a step-by-step explanation of the process depicted in the diagram:

1. Start

- **Initialization**: This is where the edge detection algorithm begins. It may include setting up parameters like threshold values, kernel sizes, and other configurations necessary for the specific edge detection technique being used.

2. Input Image

- **Image Acquisition**: The input to the edge detection process is typically a digital image captured by a camera or obtained from a dataset. The image can be in color (RGB) or grayscale.
- **Grayscale Conversion**: If the input image is in color, it is often converted to grayscale. This simplifies the process because edge detection primarily focuses on intensity changes, and working with a single intensity channel is computationally simpler.

3. Filtering

- **Purpose**: Filtering is essential to reduce noise in the image, which can lead to false edges. Noise in an image can come from various sources like low-light conditions, sensor imperfections, or environmental factors.
- **Types of Filters**:
    - **Gaussian Blur**: The most commonly used filter in edge detection. It applies a Gaussian function to smooth the image, reducing high-frequency noise and small details. The degree of smoothing is controlled by the standard deviation (σ) of the Gaussian function.
    - **Median Filter**: Another popular filter, particularly effective in removing salt-and-pepper noise, which is characterized by random occurrences of black and white pixels.
    - **Bilateral Filter**: This filter smooths the image while preserving edges, as it considers both spatial distance and intensity difference when averaging neighboring pixels.

4. Differentiation

- **Gradient Calculation**: This step involves computing the gradient of the image intensity. The gradient represents the rate of change in intensity, and areas with high gradients are likely to be edges.
- **Operators**:
    - **Sobel Operator**: It uses convolution with a pair of 3×3 kernels (one for horizontal and one for vertical changes) to approximate the gradient. The result is a gradient magnitude and direction for each pixel.
    - **Prewitt Operator**: Similar to Sobel but with slightly different convolution kernels. It's less commonly used because the Sobel operator tends to provide better results.

    - **Roberts Cross Operator**: A simple edge detection operator that uses 2×2 convolution kernels. It's faster but less accurate compared to Sobel or Prewitt.
    - **Canny Edge Detector**: A multi-step process that includes gradient calculation, non-maximum suppression, and edge tracking by hysteresis. It's considered one of the best edge detection methods due to its accuracy and robustness.

5. Localization

- **Non-Maximum Suppression**: After calculating the gradient, the next step is to thin out the edges to ensure that each edge is only one pixel wide. Non-maximum suppression checks the gradient magnitude of each pixel and suppresses any pixel that is not a local maximum in the direction of the gradient.
- **Edge Thresholding**: This step involves applying thresholds to decide which gradients represent edges. There are usually two thresholds:
    - **High Threshold**: Gradients above this value are considered strong edges.
    - **Low Threshold**: Gradients below this value are considered weak edges and may be kept or discarded based on their connectivity to strong edges (in the case of the Canny algorithm).
- **Edge Linking and Hysteresis**: This step ensures that the edges are continuous and connected. Weak edges that are connected to strong edges are preserved, while isolated weak edges are discarded.

6. Display

- **Edge Map Generation**: The final output of the edge detection process is an edge map, where detected edges are highlighted. This edge map is typically a binary image where pixels corresponding to edges are marked as white (1) and non-edge pixels as black (0).
- **Visualization**: The edge map can be overlaid on the original image for visual inspection or used as input to further image processing tasks, such as object recognition, image segmentation, or feature extraction.

7. End

- **Post-Processing**: In some cases, further refinement of the detected edges is performed after the initial edge detection process. This could include morphological operations like dilation or erosion to enhance the edges or fill gaps.

- **Final Output**: The edge detection process concludes with a refined edge map that is ready for use in higher-level vision tasks or for display to the user.

10 b] Explain first order edge detection operators

i) Roberts operator

ii) Prewitt operator

iii) Sobel operator

First-Order Edge Detection Operators

First-order edge detection operators are used to detect edges in an image by identifying areas where the intensity of pixels changes significantly. These operators calculate the gradient of the image intensity at each pixel, which highlights the edges. Here's an explanation of three common first-order edge detection operators:

i) Roberts Operator

The Roberts operator is a simple, first-order edge detection method that calculates the gradient using a pair of 2×2 convolution masks. It detects edges by approximating the gradient of the image intensity at each pixel. The operator uses two kernels:

$$G_x = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix}, \quad G_y = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$$

- (G_x) detects changes in the vertical direction, and (G_y) detects changes in the horizontal direction.
- The magnitude of the gradient is calculated as

$$G = \sqrt{G_x^2 + G_y^2}.$$

- It's sensitive to noise due to its small kernel size and is best suited for detecting diagonal edges.

ii) Prewitt Operator

The Prewitt operator is another first-order edge detection method that uses larger convolution kernels (3×3) to calculate the gradient, making it more robust to noise than the Roberts operator. The Prewitt operator uses two kernels:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}, \quad G_y = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

- (G_x) detects horizontal edges, and (G_y) detects vertical edges.
- The gradient magnitude is calculated similarly as

$$G = \sqrt{G_x^2 + G_y^2}.$$

- Prewitt operator is effective for detecting edges in images with more uniform noise levels and provides a better edge response for edges aligned with the horizontal and vertical directions.

iii) Sobel Operator

The Sobel operator is an extension of the Prewitt operator, with an added emphasis on the central pixels of the convolution kernel. It also uses 3×3 kernels, but the weights are adjusted to give more importance to the center pixel:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- (G_x) detects vertical edges, and (G_y) detects horizontal edges.
- The gradient magnitude is calculated as

$$G = \sqrt{G_x^2 + G_y^2}.$$

- The Sobel operator is widely used due to its effectiveness in reducing noise and its ability to highlight edges with a higher degree of precision, particularly for images with significant noise or uneven lighting.

These operators are foundational in image processing for edge detection and are often used as the first step in more complex image analysis tasks.

10 b] What are template matching masks? Explain any 3 template matching masks?

Answer:

9.4.3.4 Template matching masks

Gradient masks are isotropic and insensitive to directions. Sometimes it is necessary to design direction sensitive filters. Such filters are called *template matching filters*. A few kinds of template matching masks are discussed in this section.

Kirsch masks Kirsch masks are called compass masks because they are obtained by taking one mask and rotating it to the eight major directions: north, north west, west, south west, south, south-east, east, and north-east. The respective masks are given as

$$K_0=\begin{bmatrix}-3&-3&5\\-3&0&5\\-3&-3&5\end{bmatrix} K_1=\begin{bmatrix}-3&5&5\\-3&0&5\\-3&-3&-3\end{bmatrix} K_2=\begin{bmatrix}5&5&5\\-3&0&-3\\-3&-3&-3\end{bmatrix} K_3=\begin{bmatrix}5&5&-3\\5&0&-3\\-3&-3&-3\end{bmatrix}$$

$$K_4=\begin{bmatrix}5&-3&-3\\5&0&-3\\5&-3&-3\end{bmatrix} K_5=\begin{bmatrix}-3&-3&-3\\5&0&-3\\5&5&-3\end{bmatrix} K_6=\begin{bmatrix}-3&-3&-3\\-3&0&-3\\5&5&5\end{bmatrix} K_7=\begin{bmatrix}-3&-3&-3\\-3&0&5\\-3&5&5\end{bmatrix}$$

Each mask is applied to the image and the convolution process is carried out. The magnitude of the final edge is the maximum value of all the eight masks. The edge direction is the direction associated with the mask that produces maximum magnitude.

Robinson compass mask The spatial masks for the Robinson edge operator for all the directions are as follows:

$$R_0=\begin{pmatrix}-1&0&1\\-2&0&2\\-1&0&1\end{pmatrix} R_1=\begin{pmatrix}0&1&2\\-1&0&1\\-2&-1&0\end{pmatrix} R_2=\begin{pmatrix}1&2&1\\0&0&0\\-1&-2&-1\end{pmatrix},$$

$$R_3=\begin{pmatrix}2&1&0\\1&0&-1\\0&-1&-2\end{pmatrix} R_4=\begin{pmatrix}1&0&-1\\2&0&-2\\1&0&-1\end{pmatrix} R_5=\begin{pmatrix}0&-1&-2\\1&0&-1\\2&1&0\end{pmatrix},$$

$$R_6=\begin{pmatrix}-1&-2&-1\\0&0&0\\1&2&1\end{pmatrix} R_7=\begin{pmatrix}-2&-1&0\\-1&0&1\\0&1&2\end{pmatrix}$$

Similar to Kirsch masks, the mask that produces the maximum value defines the direction of the edge. It is sufficient for edge detection. The results of the remaining masks are the negation of the first four masks. Thus the computational effort can be reduced.

Frei-Chen masks Any image can be considered as the weighted sum of the nine Frei-Chen masks. The weights are obtained by a process called projecting process by overlaying a 3×

3 image onto each mask and by summing the multiplication of coincident terms. The first four masks represent the edge space, the next four represent the line subspace, and the last one represents the average subspace. The Frei-Chen masks are given as

$$F_1=\frac{1}{2\sqrt{2}}\begin{pmatrix}1 & \sqrt{2} & 1\\ 0 & 0 & 0\\ -1 & -\sqrt{2} & -1\end{pmatrix} \quad F_2=\frac{1}{2\sqrt{2}}\begin{pmatrix}1 & 0 & -1\\ \sqrt{2} & 0 & -\sqrt{2}\\ 1 & 0 & -1\end{pmatrix}$$

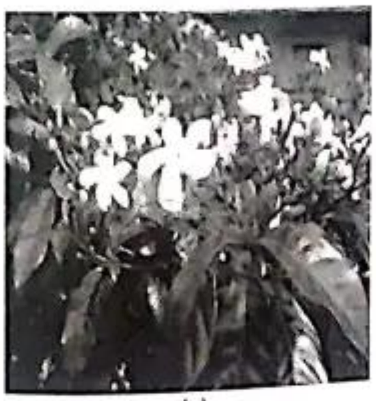
$$F_3=\frac{1}{2\sqrt{2}}\begin{pmatrix}0 & -1 & \sqrt{2}\\ 1 & 0 & -1\\ \sqrt{2} & 1 & 0\end{pmatrix} \quad F_4=\frac{1}{2\sqrt{2}}\begin{pmatrix}\sqrt{2} & -1 & 0\\ -1 & 0 & 1\\ 0 & 1 & \sqrt{2}\end{pmatrix}$$

$$F_5=\frac{1}{2}\begin{pmatrix}0 & 1 & 0\\ -1 & 0 & -1\\ 0 & 1 & 0\end{pmatrix} \quad F_6=\frac{1}{2}\begin{pmatrix}-1 & 0 & -1\\ 0 & 0 & 0\\ 1 & 0 & -1\end{pmatrix}$$

$$F_7=\frac{1}{6}\begin{pmatrix}1 & -2 & 1\\ -2 & 4 & -2\\ 1 & -2 & 1\end{pmatrix} \quad F_8=\frac{1}{6}\begin{pmatrix}-2 & 1 & -2\\ 1 & 4 & 1\\ -2 & 1 & -2\end{pmatrix}$$

$$F_9=\frac{1}{3}\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix}$$

Figures 9.11(a)–9.11(d) show an original image and the images obtained by using the Kirsch, Robinson compass, and Frei-Chen masks, respectively.


(a)

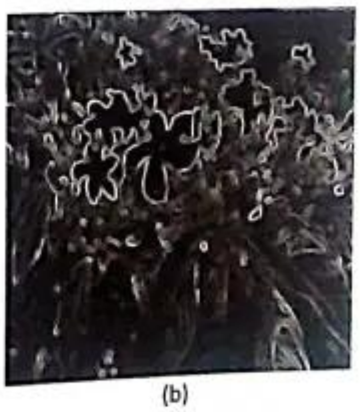

(b)

Fig. 9.11 Template matching masks (a) Original image (b) Image obtained using Krisch mask

DIGITAL IMAGE PROCESSING

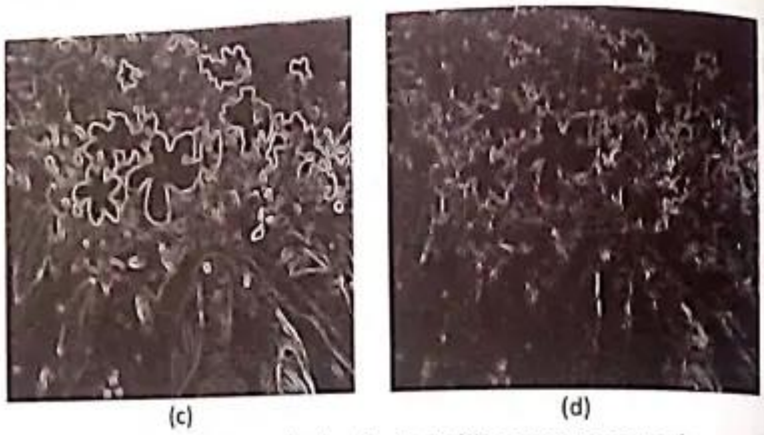


Fig. 9.11 (c) Image obtained using Robinson compass mask
(d) Image obtained using Frei-Chen mask

9 c] Explain canny edge detection algorithm.

Canny Edge Detection Algorithm

The Canny edge detection algorithm is a popular technique used in image processing to detect edges in an image. It is designed to optimize the balance between detecting true edges, accurately locating them, and minimizing the detection of false edges. The steps involved in the Canny edge detection process are as follows:

Key Objectives of the Canny Edge Detection Algorithm:

1. **Good Edge Detection:** The algorithm should detect only the actual edge points in an image, effectively discarding any false edges that do not correspond to real edges.
2. **Good Edge Localization:** The detected edge points should be as close as possible to the actual edges in the image, ensuring precise edge localization.
3. **Single Response to Each Edge:** The algorithm should produce only one edge response per edge, avoiding any false, double, or spurious edges.

Steps in the Canny Edge Detection Algorithm:

1. **Gaussian Smoothing:**

- The first step involves convolving the image with a Gaussian filter to smooth it, reducing noise and minor details that might be mistaken for edges. The gradient of the smoothed image is then computed, which gives the edge magnitude (M(x, y)) and edge orientation (\theta(x, y)). These are stored separately in two arrays.

2. Non-Maxima Suppression:

- After obtaining the edge magnitude and orientation, the edges need to be thinned to ensure they are well-defined. This is done through non-maxima suppression, which examines the gradient direction of each edge point. Instead of analyzing every possible direction (0-360°), the gradient direction is reduced to four sectors by dividing the range into eight equal parts, with two parts forming one sector.
- For a given point (M(x, y)), the edge magnitudes of two neighboring pixels along the same gradient direction are compared. If the magnitude at (M(x, y)) is less than that of its neighbors, the value at (M(x, y)) is suppressed (set to zero). Otherwise, the value is retained.

3. **Hysteresis Thresholding:**

- The final step involves applying hysteresis thresholding to identify strong and weak edges. This technique uses two thresholds:
    - **High Threshold ((T_H)):** If the gradient magnitude at a pixel is greater than (T_H), it is considered a definite edge point.
    - **Low Threshold ((T_L)):** If the gradient magnitude is below (T_L), the pixel is classified as noise and discarded.
    - If the gradient magnitude falls between (T_L) and (T_H), the pixel is considered a weak edge point, and its classification depends on its context.
- Two images are generated using the high and low thresholds. The image with the high threshold contains strong edges but may have gaps. The low threshold image, which may include more noise, is used to bridge these gaps. The neighboring pixels of weak edges in the low-threshold image are examined, and if they connect to strong edges, they are kept, ensuring the edges in the final image are continuous and accurately represent the contours of the original image.

This process ensures that the Canny edge detector effectively detects true edges, accurately locates them, and minimizes the detection of false or duplicate edges, resulting in a clean and precise edge map of the image.