QUESTION 1: (30 Marks)

a) Distinguish between the following terms

i. Emissive and Non-Emissive Displays (2 marks)

- **Emissive Displays:** These displays generate light directly from their surface. Examples include CRTs (Cathode Ray Tubes), LED (Light Emitting Diodes), OLED (Organic LEDs), and plasma screens. The pixels themselves emit light when activated.
- **Non-Emissive Displays:** These displays rely on external light sources to be visible. They do not generate light on their own. Instead, they modulate the light reflected from the environment. Examples include LCDs (Liquid Crystal Displays) and e-paper.

ii. Image Processing and Image Synthesis (2 marks)

- **Image Processing:** This involves the manipulation or analysis of an image to improve its quality, extract information, or modify it in some way. Common techniques include filtering, enhancement, segmentation, and compression.
- **Image Synthesis:** This refers to the process of generating images from models. It involves creating a digital image from geometric models, lights, textures, and other scene descriptions using techniques like ray tracing, rasterization, and shading.

iii. Interlacing Scan and Progressive Scan (2 marks)

- **Interlacing Scan:** This technique displays an image by scanning every other line of pixels first (odd lines) and then going back to scan the remaining lines (even lines). This is used to reduce flickering in CRT displays but can cause artifacts in fast-moving images.
- **Progressive Scan:** This method displays the entire image frame by frame, line by line, in sequential order from top to bottom. This provides smoother motion and better image quality, particularly in modern displays.

b) RGB Raster System Storage Calculation (4 marks)

Given:

- Screen size: 12 inches x 20 inches
- Resolution: 100 pixels per inch (both directions)
- Bits per pixel: 6 bits

Steps:

1. Calculate the total number of pixels: [\text = (12 \text \times 100 \text{ pixels/inch}) \times (20 \text \times 100 \text{ pixels/inch}) = 1200 \times 2000 = 2,400,000 \text]

2. Calculate the total storage required: [\text = 2,400,000 \text \times 6 \text{ bits/pixel} = 14,400,000 \text] Convert bits to kilobytes: [\text = \frac{14,400,000 \text}{8 \times 1024} \approx 1,757.81 \text]

So, **1,757.81 KB** of storage is needed for the frame buffer.

c) Types of Geometric Primitives in OpenGL (3 marks)

1. **Points:** Represented as a single vertex in OpenGL, used to draw a dot at a specific coordinate.
2. **Lines:** Defined by two vertices and used to draw straight lines between those vertices.
3. **Polygons:** Typically represented as triangles in OpenGL, polygons are formed by connecting three or more vertices in a closed sequence.

d) Translation of a Point (4 marks)

Given:

- Initial coordinates: D(5, 6, 7)
- Translation: x and y by +3 units, z by +2 units.

New coordinates: [\text x = 5 + 3 = 8] [\text y = 6 + 3 = 9] [\text z = 7 + 2 = 9]

So, the new coordinates are **D'(8, 9, 9).**

e) Factors to Consider When Selecting a Display Technology (4 marks)

1. **Resolution:** Determines the clarity and detail of the images displayed. Higher resolution screens offer sharper images.
2. **Brightness and Contrast Ratio:** Important for visibility in different lighting conditions and for ensuring that images have depth and color accuracy.
3. **Power Consumption:** Particularly crucial for battery-operated devices, where energy efficiency is a priority.

f) Basic Principles of Displaying Polygons in OpenGL (3 marks)

1. **Vertex Specification:** Define the vertices of the polygon in the correct order (counterclockwise or clockwise) to ensure proper rendering.
2. **Shading Model:** Choose between flat shading, where a single color is used for the entire polygon, and smooth shading, where colors vary across the surface.
3. **Rasterization:** The process of converting the polygon into a set of pixels or fragments that will be displayed on the screen.

g) Digital Differential Analyzer (DDA) Scan Conversion Algorithm (6 Marks)

Steps:

1. **Initialize the variables:** Calculate the differences (Δx and Δy) and determine the step count based on the larger of these values.
2. **Calculate the increments:** Find the increments for x and y (Δx/steps and Δy/steps).
3. **Set the starting point:** Use the initial coordinates as the starting point.
4. **Iteratively plot the points:** Increment the x and y coordinates by their respective increments for each step and plot the point.
5. **Round to the nearest pixel:** Convert the calculated floating-point coordinates to the nearest integer pixel value.
6. **Repeat until the end of the line:** Continue plotting until the end point is reached.

Question 2: (15 Marks)

a) Clipping in Computer Graphics

i. Definition of Clipping (1 Mark) Clipping refers to the process of confining drawing operations to a designated area of the display screen. Any parts of a graphical object that fall outside the specified boundaries are not drawn.

ii. Types of Clipping (4 Marks)

1. **Point Clipping:** Determines whether a point lies inside or outside a clipping window.
2. **Line Clipping:** Used to clip lines to a specified region or clipping window (e.g., Cohen-Sutherland, Liang-Barsky algorithms).
3. **Polygon Clipping:** Clips polygons to fit within a specified boundary (e.g., Sutherland-Hodgman algorithm).
4. **Text Clipping:** Used to confine the display of text strings within a predefined boundary.

b) Bresenham's Line Drawing Algorithm (10 Marks)

Given points: (20, 10) and (30, 18).

Steps:

1. **Calculate the differences:** [\Delta x = 30 - 20 = 10] [\Delta y = 18 - 10 = 8]

2. **Determine the decision parameter:** [p_0 = 2\Delta y - \Delta x = 2(8) - 10 = 6]

3. **Iteratively calculate each pixel:**

    - Start at (20, 10)
    - Use the decision parameter (p_k) to determine the next pixel.

Pixel positions calculated:

- Starting Point: (20, 10)
- After applying the algorithm, the pixel positions could be something like:
    - (21, 11)
    - (22, 12)
    - (23, 13)
    - (24, 14)
    - (25, 15)
    - (26, 15)
    - (27, 16)
    - (28, 17)
    - (29, 17)
    - (30, 18)

Finally, sketch the resulting line on a grid where each pixel is marked to visualize the line.

QUESTION 3: (15 Marks)

a) OpenGL Callback Functions

i) Purpose of the `glutDisplayFunc()` Callback Function (2 Marks)
The `glutDisplayFunc()` callback function is essential in OpenGL applications. Its primary purpose is to specify the function that will be called whenever the display needs to be updated. This function is where all the rendering or drawing code goes. The `glutDisplayFunc()` ensures

that the rendering happens whenever needed, such as after the window is resized, uncovered, or explicitly refreshed using `glutPostRedisplay()`. It enables the program to react to changes in the window's state and ensure the content is rendered correctly.

ii) Program Demonstrating the Use of `glutDisplayFunc()` (8 Marks)
Here's a simple OpenGL program that demonstrates the use of the `glutDisplayFunc()` callback function. This program sets up a basic window and renders a colored triangle.

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

void display() {
    // Clear the window with the current clearing color
    glClear(GL_COLOR_BUFFER_BIT);

    // Begin drawing a triangle
    glBegin(GL_TRIANGLES);
        glColor3f(1.0, 0.0, 0.0); // Red
        glVertex2f(-0.5, -0.5);   // Vertex 1
        glColor3f(0.0, 1.0, 0.0); // Green
        glVertex2f(0.5, -0.5);    // Vertex 2
        glColor3f(0.0, 0.0, 1.0); // Blue
        glVertex2f(0.0, 0.5);     // Vertex 3
    glEnd();

    // Flush drawing commands and swap buffers
    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Simple Triangle");

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

    // Set up the clearing color (white background)
    glClearColor(1.0, 1.0, 1.0, 1.0);

    // Enter the GLUT main loop
    glutMainLoop();

    return 0;
}
```

Explanation:

- **`glutDisplayFunc(display);`** tells GLUT to use the `display` function for rendering the scene. Whenever the window needs to be refreshed, the `display()` function is called.
- **`glClear(GL_COLOR_BUFFER_BIT);`** clears the screen to the current clear color.
- **`glBegin(GL_TRIANGLES);`** begins the process of drawing a triangle.
- **`glVertex2f(...)`** specifies the vertices of the triangle.
- **`glEnd();`** ends the drawing process.
- **`glFlush();`** ensures all OpenGL commands are executed as quickly as possible.

b) Sketching the OpenGL Code Output (5 Marks)

Given code:

```
glBegin(GL_TRIANGLE_STRIP);
glColor3f(1,1,1); // color
glVertex2f(0,0); //v1
glVertex2f(0, 1); //v2
glVertex2f(1,0); //v3
glVertex2f(1,1); //v4
glVertex2f(2,0); //v5
glEnd();
```

Explanation and Sketch:

- **GL_TRIANGLE_STRIP** creates a series of connected triangles. The first three vertices form the first triangle, and each subsequent vertex forms a new triangle with the previous two vertices.
- The vertices specified create the following points:
  - **(0, 0)** - Bottom-left corner
  - **(0, 1)** - Top-left corner
  - **(1, 0)** - Bottom-middle corner
  - **(1, 1)** - Top-middle corner
  - **(2, 0)** - Bottom-right corner

Sketch: The sequence of triangles formed would look like this:

- **Triangle 1:** (0, 0), (0, 1), (1, 0)
- **Triangle 2:** (0, 1), (1, 0), (1, 1)
- **Triangle 3:** (1, 0), (1, 1), (2, 0)

These triangles form a strip that covers the defined vertices. The output would show a connected strip of triangles, starting from the bottom left and ending at the bottom right.

QUESTION 4: (15 Marks)

a) Rotation of a Line about the Origin (6 Marks)

Given endpoints:

- **C(3, 4)** and **D(12, 15)**

Rotation by 45° anticlockwise:

To rotate a point ((x, y)) about the origin by an angle (\theta), the new coordinates ((x', y')) are given by: [x' = x \cdot \cos(\theta) - y \cdot \sin(\theta)] [y' = x \cdot \sin(\theta) + y \cdot \cos(\theta)]

For a 45° rotation, (\cos(45^\circ) = \sin(45^\circ) = \frac{\sqrt{2}}{2}).

For point C(3, 4): [x' = 3 \cdot \frac{\sqrt{2}}{2} - 4 \cdot \frac{\sqrt{2}}{2} = \frac{3\sqrt{2} - 4\sqrt{2}}{2} = \frac{-\sqrt{2}}{2} \approx -0.707] [y' = 3 \cdot \frac{\sqrt{2}}{2} + 4 \cdot \frac{\sqrt{2}}{2} = \frac{3\sqrt{2} + 4\sqrt{2}}{2} = \frac{7\sqrt{2}}{2} \approx 4.95]

New coordinates for C after rotation: **C'(-0.707, 4.95)**

For point D(12, 15): [x' = 12 \cdot \frac{\sqrt{2}}{2} - 15 \cdot \frac{\sqrt{2}}{2} = \frac{12\sqrt{2} - 15\sqrt{2}}{2} = \frac{-3\sqrt{2}}{2} \approx -2.12] [y' = 12 \cdot \frac{\sqrt{2}}{2} + 15 \cdot \frac{\sqrt{2}}{2} = \frac{12\sqrt{2} + 15\sqrt{2}}{2} = \frac{27\sqrt{2}}{2} \approx 19.09]

New coordinates for D after rotation: **D'(-2.12, 19.09)**

b) Pixels Accessed per Second & Access Time (4 Marks)

Given:

- **Resolution 1:** 1280 x 720
- **Resolution 2:** 2048 x 1536
- **Refresh Rate:** 120Hz

i. Pixels accessed per second: [\text = \text \times \text]

For 1280 x 720: [\text = 1280 \times 720 \times 120 = 110,592,000 \text{ pixels/second}]

For 2048 x 1536: [\text = 2048 \times 1536 \times 120 = 377,487,360 \text{ pixels/second}]

ii. Access time per pixel: [\text = \frac{1}{\text}]

For 1280 x 720: [\text = \frac{1}{110,592,000} \approx 9.04 \times 10^{-9} \text{ seconds/pixel}]

For 2048 x 1536: [\text = \frac{1}{377,487,360} \approx 2.65 \times 10^{-9} \text{ seconds/pixel}]

c) Architecture of a Virtual Reality System (5 Marks)

A Virtual Reality (VR) system consists of several key components that work together to create an immersive environment for the user:

1. **Input Devices:**
    - **Motion Trackers:** Capture the movements of the user's body, including head and hand tracking.
    - **Controllers:** Allow users to interact with the virtual environment through actions like grabbing, pointing, and pressing buttons.
2. **Output Devices:**
    - **Head-Mounted Display (HMD):** A wearable device that presents the virtual world directly in front of the user's eyes, often with stereoscopic (3D) vision.
    - **Audio Output:** Surround sound or binaural audio systems that deliver 3D spatial audio to enhance immersion.
3. **Processing Unit:**
    - **Computer or Console:** Processes input data, runs the VR software, and renders the virtual environment in real-time. High-performance CPUs and GPUs are crucial for smooth and responsive VR experiences.
4. **Software Components:**
    - **VR Software:** Includes the application itself, such as games or simulations, and the VR platform that handles input, rendering, and interaction.

- **Middleware:** Often includes APIs and SDKs like OpenVR or Unity3D that provide tools and libraries for VR development.

5. **Tracking Systems:**

   - **Inside-Out Tracking:** Cameras and sensors built into the HMD to track movement relative to the environment.
   - **Outside-In Tracking:** External sensors placed in the environment to track the user's position and movement.

This combination of hardware and software creates an immersive experience, making the user feel as if they are present in the virtual environment.
