COMPUTER GRAPHICS AND IMAGE PROCESSING:LAB MANUAL

Course Outcome (Course Skill Set)

At the end of the course the student will be able to:

CO 1: Use openGL /OpenCV for the development of mini Projects.

CO 2: Analyze the necessity mathematics and design required to demonstrate basic geometric transformation techniques.

C0 3: Demonstrate the ability to design and develop input interactive techniques.

C0 4: Apply the concepts to Develop user friendly applications using Graphics and IP concepts.

1. Develop a program to draw a line using Bresenham's line drawing technique

```
#include <GL/glut.h>
#include <stdio.h>
int x0 =80, y0 = 70, x1 = 150, y1 = 180;
void setPixel(int x,int y){
    glBegin(GL_POINTS);
    glVertex2i(x,y);
    glEnd();
    glFlush();
}
void breshamline(int x0,int y0,int x1,int y1)
{
    int dx= x1-x0;
    int dy = y1-y0;
    int d=2*dy-dx;
    int y=y0;
    for(int x=x0;x<=x1;x++)
    {
        setPixel(x,y);
        if(d>0)
        {
            y++;
            d=d+(2*(dy-dx));
        }else{
            d=d+2*-dy;

        }
    }
}
void display(){
    glClear(GL_COLOR_BUFFER_BIT);
    breshamline(x0,y0,x1,y1);
    glFlush();
}
void init(){
glClearColor(1.0,1.0,1.0,1.0);
glColor3f(0.0,0.0,0.0);
gluOrtho2D(0.5,200.0,0.0,200.0);
}
```

```
int main(int argc,char**argv){
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(400,400);
    glutInitWindowPosition(100,100);
    glutCreateWindow("4MT21CS146");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return(0);
}
```

2. Develop a program to demonstrate basic geometric operations on the 2D object

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

float red = 1.0f;
float green = 0.0f;
float blue = 0.0f;


void initGL() {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    gluPerspective(45.0f, 1.0f, 1.0f, 100.0f); // Perspective projection for 3D view
    glMatrixMode(GL_MODELVIEW);
    gluLookAt(1.0f, 1.0f, 3.0f,    // Eye position (slightly to the right and above the
object)
              0.0f, 0.0f, 0.0f,    // Look-at position (centered at the object)
              0.0f, 1.0f, 0.0f);   // Up vector (positive Y-axis)
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Add GL_DEPTH_BUFFER_BIT for
depth testing

    glPushMatrix();

    glBegin(GL_POLYGON);
    glColor3f(red, green, blue);
    // Front face
    glVertex2f(-0.5f, -0.5f);
    glVertex2f(0.5f, -0.5f);
    glVertex2f(0.5f, 0.5f);
    glVertex2f(-0.5f, 0.5f);

    glEnd();

    glPopMatrix();

    glutSwapBuffers(); // Use double buffering for smoother animation
}
```

```
void menu(int option) {
    switch (option) {
        case 1:
            red = 1.0f;
            green = 0.0f;
            blue = 0.0f;
            break;
        case 2:
            red = 0.0f;
            green = 1.0f;
            blue = 0.0f;
            break;
        case 3:
            red = 0.0f;
            green = 0.0f;
            blue = 1.0f;
            break;
        case 4:
            red = 1.0f;
            green = 1.0f;
            blue = 0.0f;
            break;
        case 5:
            red = 1.0f;
            green = 0.0f;
            blue = 1.0f;
            break;
        case 6:
            red = 0.0f;
            green = 1.0f;
            blue = 1.0f;
            break;
    }
    glutPostRedisplay();
}



int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH); // Enable double buffering
and depth testing
    glutCreateWindow("3D Tetrahedron_4MT21CS138");
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(50, 50);
    glutDisplayFunc(display);

    initGL();

    glutCreateMenu(menu);
    glutAddMenuEntry("Red", 1);
    glutAddMenuEntry("Green", 2);
    glutAddMenuEntry("Blue", 3);
    glutAddMenuEntry("Yellow", 4);
```

```
    glutAddMenuEntry("Magenta", 5);
    glutAddMenuEntry("Cyan", 6);

    glutAttachMenu(GLUT_RIGHT_BUTTON);

    glutMainLoop();

    return 0;
}
```

3. Develop a program to demonstrate basic geometric operations on the 3D object

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

float red = 1.0f;
float green = 0.0f;
float blue = 0.0f;

void initGL() {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    gluPerspective(45.0f, 1.0f, 1.0f, 100.0f); // Perspective projection for 3D view
    glMatrixMode(GL_MODELVIEW);
    gluLookAt(1.0f, 1.0f, 3.0f,    // Eye position (slightly to the right and above the
object)
              0.0f, 0.0f, 0.0f,    // Look-at position (centered at the object)
              0.0f, 1.0f, 0.0f);   // Up vector (positive Y-axis)
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Add GL_DEPTH_BUFFER_BIT for
depth testing

    glPushMatrix();

    glBegin(GL_TRIANGLES);
    glColor3f(red, green, blue);
    // Front face
    glVertex3f(0.0f, 0.5f, 0.0f);
    glVertex3f(-0.5f, -0.5f, 0.5f);
    glVertex3f(0.5f, -0.5f, 0.5f);
    // Right face
    glVertex3f(0.0f, 0.5f, 0.0f);
    glVertex3f(0.5f, -0.5f, 0.5f);
    glVertex3f(0.5f, -0.5f, -0.5f);
    // Back face
    glVertex3f(0.0f, 0.5f, 0.0f);
    glVertex3f(0.5f, -0.5f, -0.5f);
    glVertex3f(-0.5f, -0.5f, -0.5f);
    // Left face
    glVertex3f(0.0f, 0.5f, 0.0f);
```

```
    glVertex3f(-0.5f, -0.5f, -0.5f);
    glVertex3f(-0.5f, -0.5f, 0.5f);
    glEnd();

    glPopMatrix();
    glutSwapBuffers(); // Use double buffering for smoother animation
}

void menu(int option) {
    switch (option) {
        case 1:
            red = 1.0f;
            green = 0.0f;
            blue = 0.0f;
            break;
        case 2:
            red = 0.0f;
            green = 1.0f;
            blue = 0.0f;
            break;
        case 3:
            red = 0.0f;
            green = 0.0f;
            blue = 1.0f;
            break;
        case 4:
            red = 1.0f;
            green = 1.0f;
            blue = 0.0f;
            break;
        case 5:
            red = 1.0f;
            green = 0.0f;
            blue = 1.0f;
            break;
        case 6:
            red = 0.0f;
            green = 1.0f;
            blue = 1.0f;
            break;
    }
    glutPostRedisplay();
}



int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH); // Enable double buffering
and depth testing
    glutCreateWindow("3D Tetrahedron_4MT21CS138");
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(50, 50);
    glutDisplayFunc(display);
```

```
    initGL();
    glutCreateMenu(menu);
    glutAddMenuEntry("Red", 1);
    glutAddMenuEntry("Green", 2);
    glutAddMenuEntry("Blue", 3);
    glutAddMenuEntry("Yellow", 4);
    glutAddMenuEntry("Magenta", 5);
    glutAddMenuEntry("Cyan", 6);

    glutAttachMenu(GLUT_RIGHT_BUTTON);

    glutMainLoop();

    return 0;
}
```

4. Develop a program to demonstrate 2D transformation on basic objects

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

GLfloat vertices[] = {
-0.5f, -0.5f,
0.5f, -0.5f,
0.5f, 0.5f,
-0.5f, 0.5f
};
GLfloat tx = 0.0f; // Translation in x
GLfloat ty = 0.0f; // Translation in y
GLfloat angle = 0.0f; // Rotation angle
GLfloat sx = 1.0f; // Scaling in x
GLfloat sy = 1.0f; // Scaling in y
void display() {
glClear(GL_COLOR_BUFFER_BIT);
glPushMatrix();
// Apply transformations
glTranslatef(tx, ty, 0.0f);
glRotatef(angle, 0.0f, 0.0f, 1.0f);
glScalef(sx, sy, 1.0f);
// Draw the rectangle
glBegin(GL_POLYGON);
for (int i = 0; i < 4; ++i) {

glVertex2f(vertices[i*2], vertices[i*2+1]);
}
glEnd();
glPopMatrix();
glutSwapBuffers();
}
void keyboard(unsigned char key, int x, int y) {
```

```
switch (key) {
case 'w':
ty += 0.1f; // Move up
break;
case 's':
ty -= 0.1f; // Move down
break;
case 'a':
tx -= 0.1f; // Move left
break;
case 'd':
tx += 0.1f; // Move right
break;
case 'q':
angle += 5.0f; // Rotate counter-clockwise
break;
case 'e':
angle -= 5.0f; // Rotate clockwise
break;
case 'z':
sx += 0.1f; // Scale up in x
break;
case 'x':
sx -= 0.1f; // Scale down in x
break;
case 'c':
sy += 0.1f; // Scale up in y
break;
case 'v':
sy -= 0.1f; // Scale down in y
break;
case 27: // Escape key
exit(0);
break;
}
glutPostRedisplay();
}
void init() {
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
}

int main(int argc, char** argv) {
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE |
GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(100, 100);
glutCreateWindow("4MT21CS146");
init();
glutDisplayFunc(display);
glutKeyboardFunc(keyboard);
```

```
glutMainLoop();
return 0;
}
```

5. Develop a program to demonstrate 3D transformation on 3D objects

```
#include <GL/glut.h>
#include <math.h>
GLfloat vertices[] = {
    -0.5f, -0.5f, -0.5f,
    0.5f, -0.5f, -0.5f,
    0.5f, 0.5f, -0.5f,
    -0.5f, 0.5f, -0.5f,
    -0.5f, -0.5f, 0.5f,
    0.5f, -0.5f, 0.5f,
    0.5f, 0.5f, 0.5f,
    -0.5f, 0.5f, 0.5f
};
GLubyte indices[] = {
    0, 1, 2, 3,
    3, 2, 6, 7,
    1, 0, 4, 5,
    2, 1, 5, 6,
    0, 3, 7, 4,
    7, 6, 5, 4
};
GLfloat colors[] = {
    1.0f, 1.0f, 0.0f, 1.0f,
    0.0f, 1.0f, 0.0f, 1.0f,
    0.0f, 0.0f, 1.0f, 1.0f,
    1.0f, 0.0f, 1.0f, 1.0f,
    1.0f, 1.0f, 1.0f, 1.0f,
    0.0f, 1.0f, 1.0f, 1.0f,
    1.0f, 0.0f, 0.0f, 1.0f,
    0.5f, 0.5f, 0.5f, 1.0f
};
GLfloat tx = 0.0f;
GLfloat ty = 0.0f;
GLfloat tz = 0.0f;
GLfloat angleX = 0.0f;
GLfloat angleY = 0.0f;
GLfloat angleZ = 0.0f;
GLfloat sx = 1.0f;
GLfloat sy = 1.0f;
GLfloat sz = 1.0f;

void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();

    glTranslatef(tx, ty, tz);
    glRotatef(angleX, 1.0f, 0.0f, 0.0f);
    glRotatef(angleY, 0.0f, 1.0f, 0.0f);
```

```
    glRotatef(angleZ, 0.0f, 0.0f, 1.0f);
    glScalef(sx, sy, sz);

    glBegin(GL_QUADS);
    for (int i = 0; i < 6; ++i) {
        glColor4fv(&colors[i * 4]);
        for (int j = 0; j < 4; ++j) {
            glVertex3fv(&vertices[indices[i * 4 + j] * 3]);
        }
    }
    glEnd();

    glPopMatrix();
    glutSwapBuffers();
}

void keyboard(unsigned char key, int x, int y) {
    switch (key) {
        case 'w':
            ty += 0.1f; // Move up
            break;
        case 's':
            ty -= 0.1f; // Move down
            break;
        case 'a':
            tx -= 0.1f; // Move left
            break;
        case 'd':
            tx += 0.1f; // Move right
            break;
        case 'q':
            angleY += 5.0f; // Rotate around Y-axis
            break;
        case 'e':
            angleY -= 5.0f; // Rotate around Y-axis
            break;
        case 'z':
            sx += 0.1f; // Scale on X-axis
            break;
        case 'x':
            sx -= 0.1f; // Scale on X-axis
            break;
        case 'c':
            sy += 0.1f; // Scale on Y-axis
            break;
        case 'v':
            sy -= 0.1f; // Scale on Y-axis
            break;
        case 'i':
            tz -= 0.1f; // Move closer
            break;
        case 'o':
            tz += 0.1f; // Move farther
            break;
```

```
        case 'u':
            angleX += 5.0f; // Rotate around X-axis
            break;
        case 'j':
            angleX -= 5.0f; // Rotate around X-axis
            break;
        case 'k':
            angleZ += 5.0f; // Rotate around Z-axis
            break;
        case 'l':
            angleZ -= 5.0f; // Rotate around Z-axis
            break;
        case 27:
            exit(0); // Exit the program
            break;
    }
    glutPostRedisplay();
}

void init() {
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glViewport(0, 0, 500, 500);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0f, 1.0f, 1.0f, 100.0f);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
    glEnable(GL_DEPTH_TEST);
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("4MT21CS146");

    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);

    init();

    glutMainLoop();

    return 0;
}
```

6. Develop a program to demonstrate Animation effects on simple objects.

```
#include <GL/glut.h>
#include <math.h>
// Angle for rotation
```

```
float angle = 0.0;
// Position for translation
float posX = -0.8, posY = 0.0;
// Speed for translation
float speedX = 0.01;
void init() {
// Set the background color
glClearColor(0.0, 0.0, 0.0, 1.0);
}
void display() {
// Clear the screen
glClear(GL_COLOR_BUFFER_BIT);
// Save the current matrix
glPushMatrix();
// Move and draw the triangle
glTranslatef(posX, posY, 0.0);
glBegin(GL_TRIANGLES);
glColor3f(1.0, 0.0, 0.0);
glVertex2f(-0.1, -0.1);
glVertex2f(0.1, -0.1);
glVertex2f(0.0, 0.1);
glEnd();
// Restore the previous matrix
glPopMatrix();
// Save the current matrix
glPushMatrix();
// Rotate and draw the square
glTranslatef(0.5, 0.0, 0.0);
glRotatef(angle, 0.0, 0.0, 1.0);
glBegin(GL_QUADS);
glColor3f(0.0, 0.0, 1.0);
glVertex2f(-0.1, -0.1);

glVertex2f(0.1, -0.1);
glVertex2f(0.1, 0.1);
glVertex2f(-0.1, 0.1);
glEnd();
// Restore the previous matrix
glPopMatrix();
// Swap the buffers to display the scene
glutSwapBuffers();
}
void update(int value) {
// Update the rotation angle
angle += 2.0;
if (angle > 360) {
angle -= 360;
}
// Update the position for translation
posX += speedX;
if (posX > 0.8 || posX < -0.8) {
speedX = -speedX;
}
// Redraw the scene
```

```
glutPostRedisplay();
// Call update again after 16 milliseconds
glutTimerFunc(16, update, 0);
}
int main(int argc, char** argv) {
// Initialize GLUT
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowSize(800, 600);
// Create the window
glutCreateWindow("Simple Animation");
// Initialize OpenGL
init();

// Set the display function
glutDisplayFunc(display);
// Set the update function
glutTimerFunc(16, update, 0);
// Start the main loop
glutMainLoop();
return 0;
}
```

7. Write a Program to read a digital image. Split and display image into 4 quadrants, up, down, right and left.

```
import cv2

# Function to split the image into four quadrants
def split_image(image):
    height, width, _ = image.shape
    half_height = height // 2
    half_width = width // 2

    # Split the image into four quadrants
    top_left = image[:half_height, :half_width]
    top_right = image[:half_height, half_width:]
    bottom_left = image[half_height:, :half_width]
    bottom_right = image[half_height:, half_width:]

    return top_left, top_right, bottom_left, bottom_right

# Function to display images
def display_images(images, window_names):
    for img, name in zip(images, window_names):
        cv2.imshow(name, img)

    print("Press any key to terminate.")
    cv2.waitKey(0)
    cv2.destroyAllWindows()

# Read the image
image_path = "ivan.jpg"  # Replace "image.jpg" with the path to your image
```

```
image = cv2.imread(image_path)

if image is None:
    print("Failed to load the image.")
else:
    # Split the image into quadrants
    top_left, top_right, bottom_left, bottom_right = split_image(image)

    # Display the quadrants
    display_images([top_left, top_right, bottom_left, bottom_right], ["top_left", "Top
Right", "Bottom Left", "Bottom Right"])
```

8.a program to show rotation, scaling, and translation on an image.

```
import cv2
import numpy as np

# Load the image
image_path = "face.jpg"  # Replace with the path to your image
img = cv2.imread(image_path)

# Get the image dimensions
height, width, _ = img.shape

# Define the transformation matrices
rotation_matrix = cv2.getRotationMatrix2D((width/2, height/2), 45, 1)  # Rotate by 45
degrees
scaling_matrix = np.float32([[1.5, 0, 0], [0, 1.5, 0]])  # Scale by 1.5x
translation_matrix = np.float32([[1, 0, 100], [0, 1, 50]])  # Translate by (100, 50)

# Apply transformations
rotated_img = cv2.warpAffine(img, rotation_matrix, (width, height))
scaled_img = cv2.warpAffine(img, scaling_matrix, (int(width*1.5), int(height*1.5)))
translated_img = cv2.warpAffine(img, translation_matrix, (width, height))

# Display the original and transformed images
cv2.imshow("Original Image", img)
cv2.imshow("Rotated Image", rotated_img)
cv2.imshow("Scaled Image", scaled_img)
cv2.imshow("Translated Image", translated_img)
# Wait for a key press and then close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

9. Read an image and extract and display low-level features such as edges, textures using filtering techniques.

```
import cv2
import numpy as np

# Load the image
image_path = "car.jpg"  # Replace with the path to your image
```

```
img = cv2.imread(image_path)

# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Apply Laplacian filter to extract edges
laplacian_edges = cv2.Laplacian(img, cv2.CV_64F)
laplacian_edges = cv2.normalize(laplacian_edges, None, 0, 255, cv2.NORM_MINMAX,
dtype=cv2.CV_8U)

# Display edges extracted using Laplacian filter
cv2.imshow("Edges (Laplacian Filter)", laplacian_edges)

# Edge detection
edges = cv2.Canny(gray, 100, 200)  # Use Canny edge detector

# Texture extraction
kernel = np.ones((5, 5), np.float32) / 25  # Define a 5x5 averaging kernel
texture = cv2.filter2D(gray, -1, kernel)  # Apply the averaging filter for texture
extraction

# Display the original image, edges, and texture
cv2.imshow("Original image", img)
cv2.imshow("Edges", edges)
cv2.imshow("Texture", texture)

# Wait for a key press and then close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

10. Write a program to blur and smoothing an image.

```
import cv2

# Load the image
image = cv2.imread('org.jpg')

# Gaussian Blur
gaussian_blur = cv2.GaussianBlur(image, (5, 5), 0)

# Median Blur
median_blur = cv2.medianBlur(image, 5)

# Bilateral Filter
bilateral_filter = cv2.bilateralFilter(image, 9, 75, 75)

# Display the original and processed images
cv2.imshow('Original Image', image)
cv2.imshow('Gaussian Blur', gaussian_blur)
cv2.imshow('Median Blur', median_blur)
cv2.imshow('Bilateral Filter', bilateral_filter)
```

```
# Wait for a key press to close the windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

11. Write a program to contour an image.

```
import cv2
import numpy as np

# Load the image
image = cv2.imread('dar.jpg')

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply binary thresholding
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)

# Find contours
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Create a copy of the original image to draw contours on
contour_image = image.copy()

# Draw contours on the image
cv2.drawContours(contour_image, contours, -1, (0, 255, 0), 2)

# Display the original and contour images
cv2.imshow('original image', image)
cv2.imshow('Contours', contour_image)
# Wait for a key press to close the windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

12. Write a program to detect a face/s in an image.

```
import cv2
import matplotlib.pyplot as plt

# Load the pre-trained Haar cascade classifier for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')

# Read the image
image = cv2.imread('sop2.jpg')
if image is None:
    print("Error: Could not read the image.")
    exit()

# Convert the image to grayscale
```

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Detect faces in the image
faces = face_cascade.detectMultiScale(gray_image, scaleFactor=1.1, minNeighbors=5,
minSize=(30, 30), flags=cv2.CASCADE_SCALE_IMAGE)

# Draw rectangles around the detected faces
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Display the image with detected faces using Matplotlib
plt.figure(figsize=(10, 7))

plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Detected Faces (by 4MT21CS146)')
plt.axis('off')
plt.show()
```