QUESTION ONE (30 MARKS)

a) 2 Applications of Computer Graphics in Health Sector (4 marks)

- **Medical Imaging:** Used in CT scans, MRIs, and X-rays to visualize internal organs in 2D/3D.
- **Surgical Simulation:** Allows practice of surgical procedures using 3D models for training.

b) Flood Fill a Trapezium (4 marks)

```
oid display() {
glClear(GL_COLOR_BUFFER_BIT);

// Draw black border trapezium
glColor3f(0.0f, 0.0f, 0.0f); // Black
glBegin(GL_LINE_LOOP);
glVertex2i(100, 100); // base a = 12cm
glVertex2i(220, 100); // base b = 20cm
glVertex2i(260, 200); // top
glVertex2i(60, 200); // top
glEnd();

glFlush();

// Flood fill from inside point
float fillColor[] = {0.5f, 0.0f, 0.5f}; // Pink (#800080)
float borderColor[] = {0.0f, 0.0f, 0.0f}; // Black
floodFill(150, 150, fillColor, borderColor);
}
```

c) Midpoint Scan Convert for y=50−x2y = 50 - x^2y=50−x2 (6 marks)

Short working:

- Rewrite as *x2=50−yx^2 = 50 - y*x2=50−y, thus symmetry about y-axis.
- Plot for *x=−5x = -5*x=−5 to *x=5x = 5*x=5, compute y = 50 - x^2.
- Use midpoint decision formula as in parabola plotting.

Key plotting logic:

```
void display() {
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0f, 0.0f, 0.0f); // Red

glBegin(GL_POINTS);
for (int x = -5; x <= 5; x++) {
int y = 50 - (x * x); // Compute parabola y = 50 - x^2
glVertex2i(x, y);
}
glEnd();

glFlush();
}
```

d) Vector vs Raster Graphics (4 marks)

Feature	Vector Graphics	Raster Graphics
Representation	Shapes/paths (mathematical)	Pixels (grid)
Scalability	Infinitely scalable	Loses quality when scaled
File Size	Usually smaller	Larger
Best Use	Logos, illustrations	Photos, complex textures

e) Mandelbrot Set in Fractal Geometry (3 marks)

- It's a set of complex numbers that do not diverge when iterated through the equation $zn+1=zn2+cz_\{n+1\} = z_n\^2 + c$zn+1 =zn2 +c.
- Used to generate complex, self-similar fractal images.
- Common in procedural textures and landscapes.

f) Three Display Technologies (6 marks)

1. **CRT (Cathode Ray Tube):** Uses electron beams to light up phosphor pixels.
2. **LCD (Liquid Crystal Display):** Uses liquid crystals and backlight; efficient and thin.

3. **OLED (Organic LED):** Emits light directly; better contrast and flexibility.

g) Tessellation in 3D Graphics (3 marks)

- Tessellation divides complex surfaces into triangles/polygons for rendering.
- Crucial in 3D modeling and terrain rendering for smoother surfaces.

QUESTION TWO (20 MARKS)

a) Triangle with Vertices (-1,6), (2,0), (-4,9) [2 marks]

Draw triangle logic (snippet):

```
void DDA(int x1, int y1, int x2, int y2) {
float dx = x2 - x1, dy = y2 - y1;
float steps = fabs(dx) > fabs(dy) ? fabs(dx) : fabs(dy);
float xInc = dx / steps;
float yInc = dy / steps;
float x = x1, y = y1;

glBegin(GL_POINTS);
for (int i = 0; i < steps; i++) {
glVertex2i(round(x), round(y));
x += xInc;
y += yInc;
}
glEnd();
glFlush();
}
```

b) Orthogonal vs Perspective Transformation (4 marks)

Feature	Orthogonal	Perspective
Projection	Parallel lines remain parallel	Lines converge at vanishing point
Depth Effect	No depth	Depth perception present
Usage	CAD, schematics	3D games, simulations

c) Shading Techniques (4 marks)

i) Gouraud Shading:

- Calculates color at vertices, then interpolates across surface.
- Faster, but can miss specular highlights.

ii) Phong Shading:

- Interpolates normals, then computes color per pixel.
- More accurate lighting and highlights.

d) Line Drawing: (i) DDA and (ii) Bresenham (6 marks)

(i) DDA Key Step:

```
void DDA(int x1, int y1, int x2, int y2) {
float dx = x2 - x1, dy = y2 - y1;
float steps = fabs(dx) > fabs(dy) ? fabs(dx) : fabs(dy);
float xInc = dx / steps;
float yInc = dy / steps;
float x = x1, y = y1;

glBegin(GL_POINTS);
for (int i = 0; i < steps; i++) {
glVertex2i(round(x), round(y));
x += xInc;
y += yInc;
}
glEnd();
glFlush();
}
```

(ii) Bresenham Key Step:

```
void Bresenham(int x1, int y1, int x2, int y2) {
int dx = x2 - x1;
int dy = y2 - y1;
int p = 2 * dy - dx;
int x = x1, y = y1;

glBegin(GL_POINTS);
while (x <= x2) {
glVertex2i(x, y);
```

```
x++;
if (p < 0)
p += 2 * dy;
else {
y++;
p += 2 * (dy - dx);
}
}
glEnd();
glFlush();
}
```

e) Concave Polygon Splitting via Rotation (4 marks)

- Rotate polygon around pivot.
- Use edge-crossing method to find ears or notches.
- Split recursively until convex.

Q3a – Importance of Bézier-Spline Curves

- Bézier-Spline curves allow **smooth, precise control** over shapes in design.
- They are used in **automobile design**, animation, font design, and CAD.
- Curves are defined by **control points** and are **scalable** without loss of smoothness.

☑ Q3b – Chessboard in OpenGL (8x8, Green & White)

```
Q3b

void drawChessboard() {
int size = 50;
for (int i = 0; i < 8; i++) {
for (int j = 0; j < 8; j++) {
if ((i + j) % 2 == 0)
glColor3f(0.0f, 0.5f, 0.0f); // Green #008000
else
glColor3f(1.0f, 1.0f, 1.0f); // White

int x = j * size;
int y = i * size;
glBegin(GL_QUADS);
glVertex2i(x, y);
glVertex2i(x + size, y);
glVertex2i(x + size, y + size);
```

```
glVertex2i(x, y + size);
glEnd();
}
}
glFlush();
}
```

☑ Q3c(i) – Midpoint Circle Algorithm (radius = 5, center at (0,3))

```
q3ci
void drawCirclePoints(int xc, int yc, int x, int y) {
glVertex2i(xc + x, yc + y);
glVertex2i(xc - x, yc + y);
glVertex2i(xc + x, yc - y);
glVertex2i(xc - x, yc - y);
glVertex2i(xc + y, yc + x);
glVertex2i(xc - y, yc + x);
glVertex2i(xc + y, yc - x);
glVertex2i(xc - y, yc - x);
}

void drawMidpointCircle(int xc, int yc, int r) {
int x = 0, y = r;
int p = 1 - r;

glBegin(GL_POINTS);
drawCirclePoints(xc, yc, x, y);
while (x < y) {
x++;
if (p < 0)
p += 2 * x + 1;
else {
y--;
p += 2 * (x - y) + 1;
}
drawCirclePoints(xc, yc, x, y);
}
glEnd();
glFlush();
}
```

☑ Q3c(ii) – Rotate Circle by 90° and Fill with Purple

```
void display() {
glClear(GL_COLOR_BUFFER_BIT);
```

```
glColor3f(0.5f, 0.0f, 0.5f); // Purple (#800080)
glPushMatrix();
glTranslatef(0, 3, 0); // Move center to origin
glRotatef(90, 0, 0, 1); // Rotate 90 degrees
glTranslatef(0, -3, 0); // Move back
drawMidpointCircle(0, 3, 5);
glPopMatrix();

glFlush();
}
```

☑ Q3d – Explain the Concepts

- **Frame buffer:** A memory buffer that holds pixel color data for display.
- **Bit frame:** Data structure used to store 1-bit monochrome images.
- **Pixel:** Smallest unit of an image display, holds color and brightness info.
- **Digital image representation:** A grid of pixels represented in numeric format, often with RGB or grayscale values.

☑ Q3e – Two Ways to Generate Polyhedra in OpenGL

1. **Manual Definition**: Use glBegin(GL_QUADS) or glBegin(GL_TRIANGLES) to manually specify faces.
2. **Modeling Libraries**: Use GLUT/GLU/GLM to load or construct standard 3D objects like cubes, spheres, etc.

☑ Q4a – Five Factors Influencing Raster vs Vector Use

1. **Scalability** – Vector images scale without quality loss; raster becomes pixelated.
2. **File Size** – Vectors are smaller for simple graphics; raster larger but more detailed.
3. **Rendering Speed** – Raster renders faster but may require more memory.
4. **Editability** – Vector is ideal for precise edits (e.g., logos); raster suits detailed images (e.g., photos).
5. **Use Case** – Raster suits web/photos; vector suits CAD, diagrams, animations.

☑ Q4b – Gupta-Sproull Line Drawing Snippet (20,10 to 30,18)

```
void drawGuptaSproullLine(int x0, int y0, int x1, int y1) {
// Anti-aliased line using intensity weighting (simplified illustration)
float dx = x1 - x0;
float dy = y1 - y0;
float gradient = dy / dx;
for (int x = x0; x <= x1; x++) {
float y = y0 + gradient * (x - x0);
glColor3f(1.0, 1.0, 1.0); // White core
glBegin(GL_POINTS);
glVertex2i(x, round(y));
glEnd();

// Draw soft edges (simulate intensity drop-off)
glColor3f(0.7, 0.7, 0.7);
glBegin(GL_POINTS);
glVertex2i(x, round(y) + 1);
glVertex2i(x, round(y) - 1);
glEnd();
}
glFlush();
}
```

☑ Q4b(i) – Gupta-Sproull Antialiasing Explanation

- Smooths jagged edges of lines by blending edge pixels based on distance from the ideal line.
- Uses **intensity weighting** to reduce the **stair-step effect** ("aliasing").

☑ Q4c – JKUAT IS FUN TUPU Banner with Fonts and Colors

```
Q4c
void drawBannerText() {
glClearColor(0.6f, 0.4f, 0.2f, 1.0f); // Brown background
glClear(GL_COLOR_BUFFER_BIT);

glColor3f(0.0f, 0.5f, 0.0f); // Green for "JKUAT"
drawText("JKUAT", 10, 200); // Placeholder text function
```

```
glColor3f(1.0f, 0.0f, 0.0f); // Red for "IS FUN TUPU"
drawText(" IS FUN TUPU", 80, 200);

glFlush();
}
```

☑ Q4c(ii) – Pseudocode for Blender Banner (JKUAT IS FUN TUPU)

```
# Blender Python (pseudocode)
import bpy

bpy.ops.object.text_add(location=(0, 0, 0))
bpy.context.object.data.body = "JKUAT IS FUN TUPU"
bpy.context.object.data.extrude = 0.1

# Split and color text
text_obj = bpy.context.object
text_obj.data.materials.append(create_material("JKUAT", "#008000"))
text_obj.data.materials.append(create_material("IS FUN TUPU", "#FF0000"))

# Set background plane
bpy.ops.mesh.primitive_plane_add(size=20, location=(0, 0, -0.1))
set_material_to_object(bpy.context.object, "#A52A2A") # Brown
```

☑ Q4d – Draw Red Polygon (Same Coordinates as Earlier)

```
void drawPolygon() {
glColor3f(1.0f, 0.0f, 0.0f); // Red #FF0000
glBegin(GL_POLYGON);
glVertex2i(8, 4);
glVertex2i(2, 4);
glVertex2i(0, 8);
glVertex2i(3, 12);
glVertex2i(7, 12);
glVertex2i(10, 8);
glEnd();
glFlush();
}
```

☑ Q5a – Color Look-Up Tables (LUTs) in 8-bit Image Types (4 marks)

Concept:

A Color LUT is a mapping table used to translate pixel values into RGB colors. In an 8-bit image, each pixel holds a value from 0 to 255, and each value maps to a specific RGB color via the LUT.

Example:

```
unsigned char LUT[256][3]; // LUT[i][0]=R, [1]=G, [2]=B
unsigned char pixel = 45; // Pixel intensity
setColor(LUT[pixel][0], LUT[pixel][1], LUT[pixel][2]); // Use mapped color
```

☑ Q5b – Pseudocode for Clipping Algorithms (6 marks)

(i) Cohen-Sutherland Polygon Clipping

```
while (line outside window) {
assign region codes to endpoints;
if (both codes == 0)
accept line;
else if (logical AND of codes ≠ 0)
reject line;
else
move one point to window edge;
}
```

(ii) Liang-Barsky Line Clipping

```
calculate p1, p2, ..., p4 and q1, q2, ..., q4;
for each boundary:
if pi == 0 and qi < 0 → reject;
else if pi < 0 → update u1;
else if pi > 0 → update u2;
if u1 > u2 → reject line;
else clip between u1 and u2;
```

(iii) Nicholl-Lee-Nicholl Line Clipping

```
choose one endpoint as reference;
classify region of other endpoint;
clip directly using geometry;
reduces unnecessary comparisons;
```

☑ Q5c – Homogeneous Coordinates (3 marks)

- **To Homogeneous**:
  (x,y,z)→(x,y,z,1)(x, y, z) → (x, y, z, 1)(x,y,z)→(x,y,z,1)
- **From Homogeneous**:
  (x,y,z,w)→(xw,yw,zw)(x, y, z, w) → \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)(x,y,z,w)→(wx ,wy ,wz) if *w≠0w ≠ 0*w=0

Used for easier matrix transformations like translation and perspective projection.

☑ Q5d – Prove: Rotation Matrix Multiplication is Commutative (3 marks)

Let:

- *R1=Rotation about Z by θR_1 = \text{Rotation about Z by } \theta*R1 =Rotation about Z by θ
- *R2=Rotation about Z by ϕR_2 = \text{Rotation about Z by } \phi*R2 =Rotation about Z by ϕ

Then:

*R1·R2=[cosθ−sinθ0sinθcosθ0001]·[cosϕ−sinϕ0sinϕcosϕ0001]=[cos(θ+ϕ)−sin(θ+ϕ)0sin(θ+ϕ)cos(θ+ϕ)0001]=R2·R1R_1 \cdot R_2 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta+\phi) & -\sin(\theta+\phi) & 0 \\ \sin(\theta+\phi) & \cos(\theta+\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} = R_2 \cdot R_1*R1 ·R2 = cosθsinθ0 −sinθcosθ0 001 · cosϕsinϕ0 −sinϕcosϕ0 001 = cos(θ+ϕ)sin(θ+ϕ)0 −sin(θ+ϕ)cos(θ+ϕ)0 001 =R2 ·R1

✅ Therefore, rotation about the same axis is **commutative**.

☑ Q5e – Cyrus-Beck Algorithm in 3D Clipping (OpenGL Snippet)

For a 3D convex polyhedron:

```
Q5e

void applyCyrusBeckClip3D(Point3D lineStart, Point3D lineEnd, Plane planes[], int numPlanes) {
float tEnter = 0, tLeave = 1;
Vector3D D = lineEnd - lineStart;

for (int i = 0; i < numPlanes; i++) {
Vector3D N = planes[i].normal;
float NdotD = dot(N, D);
float NdotP = dot(N, (planes[i].point - lineStart));

if (NdotD != 0) {
float t = NdotP / NdotD;
if (NdotD < 0) tEnter = max(tEnter, t);
else tLeave = min(tLeave, t);
} else if (NdotP < 0)
return; // Line is outside and parallel
}

if (tEnter < tLeave) {
Point3D clippedStart = lineStart + tEnter * D;
Point3D clippedEnd = lineStart + tLeave * D;
drawClippedLine(clippedStart, clippedEnd);
}
}
```

Group 1: Bézier Curves in OpenGL

Question Requirements:

1. Describe Bézier curves and how they are implemented in OpenGL.
2. Show 2D and 3D examples.
3. Explain clipping a Bézier curve against a screen rectangle.

Answer:

1. **Bézier Curve Basics**: Defined by control points. OpenGL uses glMap1f and glEvalCoord1f to compute and draw the curve.
2. **Code (2D Example)**:

```
cpp
Copy
Download
GLfloat ctrlPoints[4][3] = {
  {0.0, 0.0, 0.0}, // Control points (x, y, z)
  {1.0, 2.0, 0.0},
  {3.0, 3.0, 0.0},
  {4.0, 0.0, 0.0}
};

glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, &ctrlPoints[0][0]);
glEnable(GL_MAP1_VERTEX_3);

glBegin(GL_LINE_STRIP);
for (int i = 0; i <= 30; i++) {
  glEvalCoord1f((GLfloat)i / 30.0); // Draw 30 segments
}
glEnd();
```

3. **3D Example**: Add a Z-coordinate (e.g., {1.0, 2.0, 1.0}).
4. **Clipping**: Use glViewport(x, y, width, height) to define the visible area.

Group 2: Bresenham Circle Algorithm
Question Requirements:

1. Draw a circle with radius 6 cm at (3,5).
2. Draw another circle at (-1,-1) filled with cyan.

Answer:
Code:

```
cpp
Copy
Download
```

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

void drawCircle(int radius, int xc, int yc) {
    int x = 0, y = radius;
    int d = 3 - 2 * radius;

    glBegin(GL_POINTS);
    while (x <= y) {
        // Plot symmetrical points in all octants
        glVertex2i(xc + x, yc + y);
        glVertex2i(xc - x, yc + y);
        glVertex2i(xc + x, yc - y);
        glVertex2i(xc - x, yc - y);
        glVertex2i(xc + y, yc + x);
        glVertex2i(xc - y, yc + x);
        glVertex2i(xc + y, yc - x);
        glVertex2i(xc - y, yc - x);

        if (d < 0) d += 4 * x + 6;
        else { d += 4 * (x - y) + 10; y--; }
        x++;
    }
    glEnd();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0); // Red
    drawCircle(6, 3, 5);

    // Cyan-filled circle
    glColor3f(0.0, 1.0, 1.0); // Cyan = (0,1,1)
    glBegin(GL_POLYGON);
    for (int angle = 0; angle < 360; angle++) {
        float rad = angle * 3.14159 / 180;
        glVertex2f(-1 + 6 * cos(rad), -1 + 6 * sin(rad));
    }
    glEnd();
    glFlush();
```

```
}

int main(int argc, char** argv) {
   glutInit(&argc, argv);
   glutCreateWindow("Circle");
   glutDisplayFunc(display);
   gluOrtho2D(-10, 10, -10, 10); // Adjust window coordinates
   glutMainLoop();
   return 0;
}
```

Group 3: Scene Graph & BSP Tree
Question Requirements:

1. Diagram a scene graph for a bicycle.
2. Illustrate BSP tree for Koch triangles.

Answer:

1. **Bicycle Scene Graph**:

```
plaintext
Copy
Download
Root
├─ Frame (Body)
├─ Front Wheel (Rotation Node)
└─ Back Wheel (Rotation Node)
```

2. **Koch Triangle with BSP**:
   Code (Recursive Splitting):

```
cpp
Copy
Download
void koch(float x1, float y1, float x2, float y2, int depth) {
   if (depth == 0) {
      glBegin(GL_LINES);
      glVertex2f(x1, y1);
      glVertex2f(x2, y2);
      glEnd();
```

```
  } else {
    // Split line into 4 segments
    float dx = (x2 - x1) / 3;
    float dy = (y2 - y1) / 3;
    float xA = x1 + dx;
    float yA = y1 + dy;
    float xB = x2 - dx;
    float yB = y2 - dy;
    float xC = (xA + xB) / 2 - dy * sqrt(3)/2;
    float yC = (yA + yB) / 2 + dx * sqrt(3)/2;

    koch(x1, y1, xA, yA, depth - 1);
    koch(xA, yA, xC, yC, depth - 1);
    koch(xC, yC, xB, yB, depth - 1);
    koch(xB, yB, x2, y2, depth - 1);
  }
}
```

Group 4: Midpoint Circle Algorithm

Question Requirements:

1. Draw a circle (radius=3 cm, center=(0,2)) using Midpoint algorithm.
2. Rotate it 60° and fill with purple.
3. Split a concave polygon using rotational method.

Answer:

1. Midpoint Circle Code:

cpp
Copy
Download

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

void drawMidpointCircle(int radius, int xc, int yc) {
  int x = 0, y = radius;
  int d = 1 - radius;

  glBegin(GL_POINTS);
```

```
    while (x <= y) {
        glVertex2i(xc + x, yc + y);
        glVertex2i(xc - x, yc + y);
        glVertex2i(xc + x, yc - y);
        glVertex2i(xc - x, yc - y);
        glVertex2i(xc + y, yc + x);
        glVertex2i(xc - y, yc + x);
        glVertex2i(xc + y, yc - x);
        glVertex2i(xc - y, yc - x);

        if (d < 0) d += 2 * x + 3;
        else { d += 2 * (x - y) + 5; y--; }
        x++;
    }
    glEnd();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Draw original circle (radius=3 at (0,2))
    glColor3f(1.0, 0.0, 1.0); // Purple
    glPushMatrix();
    glTranslatef(0, 2, 0); // Move to (0,2)
    glutSolidSphere(3, 100, 100); // Simplified fill
    glPopMatrix();

    // Rotated circle (60 degrees)
    glPushMatrix();
    glTranslatef(0, 2, 0);
    glRotatef(60, 0, 0, 1); // Rotate 60° around Z-axis
    glutSolidSphere(3, 100, 100);
    glPopMatrix();

    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
```

```
    glutCreateWindow("Circle");
    gluOrtho2D(-10, 10, -10, 10); // Set coordinate system
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Group 5: Gupta-Sproull & Xiaolin Wu Algorithms
Question Requirements:

1. Draw a line (20,10 to 30,18) using Gupta-Sproull. Explain anti-aliasing.
2. Draw a line (15,10 to 23,18) using Xiaolin Wu. Demonstrate anti-aliasing.

Answer:
1. Gupta-Sproull Code (Simplified):

```
cpp
Copy
Download
void drawGuptaSproullLine(int x0, int y0, int x1, int y1) {
    int dx = x1 - x0, dy = y1 - y0;
    float d = sqrt(dx*dx + dy*dy); // Line length

    for (float t = 0; t <= 1; t += 0.01) {
        int x = x0 + t * dx;
        int y = y0 + t * dy;
        float intensity = 1.0 - (t * (1 - t)); // Simple anti-aliasing
        glColor3f(intensity, intensity, intensity);
        glBegin(GL_POINTS);
        glVertex2i(x, y);
        glEnd();
    }
}

// Call this in display():
glColor3f(1.0, 1.0, 1.0);
drawGuptaSproullLine(20, 10, 30, 18);
```

2. Xiaolin Wu Code:

```
cpp
Copy
Download
```

```
void drawXiaolinWuLine(int x0, int y0, int x1, int y1) {
  int dx = x1 - x0, dy = y1 - y0;
  float gradient = (float)dy / dx;

  for (int x = x0; x <= x1; x++) {
    float y = y0 + gradient * (x - x0);
    float frac = y - floor(y);
    glColor3f(1.0 - frac, 1.0 - frac, 1.0 - frac);
    glVertex2i(x, (int)y);
    glColor3f(frac, frac, frac);
    glVertex2i(x, (int)y + 1);
  }
}

// Call this in display():
glColor3f(1.0, 1.0, 1.0);
drawXiaolinWuLine(15, 10, 23, 18);
```

Anti-aliasing Explanation:

- Both algorithms reduce jagged edges by blending pixel colors based on proximity to the ideal line.

Group 6: Text Banner
Question Requirements:

1. Draw "JKUAT ROCKS" with "JKUAT" in green, "OCKS" in red, and brown background.
2. Increase text size if needed.

Answer:
Code:
cpp
Copy
Download

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

void drawText(float x, float y, char* text) {
  glRasterPos2f(x, y);
  for (char* c = text; *c != '\0'; c++)
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *c);
```

```
}

void display() {
   glClearColor(0.6, 0.3, 0.1, 1.0); // Brown background (R=0.6, G=0.3, B=0.1)
   glClear(GL_COLOR_BUFFER_BIT);

   // "JKUAT" in green
   glColor3f(0.0, 1.0, 0.0);
   drawText(-2.0, 0.0, "JKUAT");

   // "OCKS" in red
   glColor3f(1.0, 0.0, 0.0);
   drawText(0.5, 0.0, "OCKS");

   glFlush();
}

// To increase text size (use scaling):
glPushMatrix();
glScalef(2.0, 2.0, 1.0); // Double the size
drawText(-2.0, 0.0, "JKUAT");
glPopMatrix();
```

Group 7: Bar Chart for Fruit Preferences

Question Requirements:

1. Draw a bar chart for fruit preference data (150 respondents).
2. Bars must match fruit colors, label axes (x: black, y: red).
3. Start the graph at (5,5).

Answer:

cpp
Copy
Download

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

// Data: Avocado(36), Orange(41), Banana(19), Kiwi(28), Mango(30), Grapes(16)
float data[] = {36, 41, 19, 28, 30, 16};
const char* labels[] = {"Avocado", "Orange", "Banana", "Kiwifruit", "Mango", "Grapes"};
```

```
float colors[][3] = {{0.3,0.6,0.2}, {1,0.5,0}, {1,1,0}, {0.5,0.3,0.5}, {1,0.8,0}, {0.5,0,1}};

void drawBarChart() {
   float barWidth = 0.8; // Width of each bar
   float startX = 5, startY = 5; // Start at (5,5)

   // Draw axes (X: black, Y: red)
   glColor3f(0,0,0);
   glBegin(GL_LINES);
   glVertex2f(startX, startY); glVertex2f(startX + 6, startY); // X-axis
   glEnd();
   glColor3f(1,0,0);
   glBegin(GL_LINES);
   glVertex2f(startX, startY); glVertex2f(startX, startY + 45); // Y-axis
   glEnd();

   // Draw bars
   for (int i = 0; i < 6; i++) {
      glColor3fv(colors[i]);
      glRectf(startX + i + 0.1, startY, startX + i + barWidth, startY + data[i]);
   }
}

void display() {
   glClear(GL_COLOR_BUFFER_BIT);
   drawBarChart();
   glFlush();
}

int main(int argc, char** argv) {
   glutInit(&argc, argv);
   glutCreateWindow("Bar Chart");
   gluOrtho2D(0, 20, 0, 60); // Adjust window to fit data
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```

Group 8: Line Graph for Daily Earnings
Question Requirements:

1. Draw a line graph with data points as asterisks connected by lines.
2. Thicken lines to width 2, use boxes as markers.
3. Blue line, red circles, cream background.

Answer:

cpp
Copy
Download

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

// Data: Mon(590), Tue(850), Wed(940), Thurs(1070), Fri(800), Sat(1020)
float data[] = {590, 850, 940, 1070, 800, 1020};

void drawLineGraph() {
   // Plot data points as asterisks
   glColor3f(1,1,1);
   for (int i = 0; i < 6; i++) {
      glRasterPos2f(i * 2, data[i]/50.0);
      glutBitmapCharacter(GLUT_BITMAP_9_BY_15, '*');
   }

   // Draw lines
   glLineWidth(2);
   glBegin(GL_LINE_STRIP);
   for (int i = 0; i < 6; i++)
      glVertex2f(i * 2, data[i]/50.0);
   glEnd();
}

void display() {
   glClearColor(1.0, 0.98, 0.82, 1.0); // Cream background (#FFFDD0)
   glClear(GL_COLOR_BUFFER_BIT);

   // Case 3: Blue line, red circles
   glColor3f(0,0,1);
   glLineWidth(2);
   glBegin(GL_LINE_STRIP);
```

```
    for (int i = 0; i < 6; i++)
        glVertex2f(i * 2, data[i]/50.0);
    glEnd();

    glColor3f(1,0,0);
    for (int i = 0; i < 6; i++) {
        glBegin(GL_POLYGON);
        for (int angle = 0; angle < 360; angle += 30)
            glVertex2f(i*2 + 0.1*cos(angle), data[i]/50.0 + 0.1*sin(angle));
        glEnd();
    }
    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutCreateWindow("Line Graph");
    gluOrtho2D(0, 12, 0, 25); // Adjust window to fit scaled data
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Group 9: Pie Chart for Fruit Data
Question Requirements:

1. Draw a pie chart with fruit percentages.
2. Labels outside the pie, colors matching fruits.
3. Convert background to grayscale.

Answer:

cpp
Copy
Download

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

// Data: Total = 150
float angles[] = {36/150.0*360, 41/150.0*360, 19/150.0*360, 28/150.0*360,
30/150.0*360, 16/150.0*360};
```

```
float colors[][3] = {{0.3,0.6,0.2}, {1,0.5,0}, {1,1,0}, {0.5,0.3,0.5}, {1,0.8,0}, {0.5,0,1}};

void drawPieChart() {
    float radius = 3.0;
    float startAngle = 0.0;

    for (int i = 0; i < 6; i++) {
        glColor3fv(colors[i]);
        glBegin(GL_POLYGON);
        glVertex2f(0, 0);
        for (float angle = startAngle; angle < startAngle + angles[i]; angle += 1) {
            float rad = angle * M_PI / 180;
            glVertex2f(radius * cos(rad), radius * sin(rad));
        }
        glEnd();
        startAngle += angles[i];
    }
}

void display() {
    glClearColor(0.5, 0.5, 0.5, 1.0); // Gray background
    glClear(GL_COLOR_BUFFER_BIT);
    drawPieChart();
    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutCreateWindow("Pie Chart");
    gluOrtho2D(-5, 5, -5, 5);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Group 10: Color Lookup Table & Scatter Plot

Question Requirements:

1. Implement a color lookup table.
2. Draw a scatter plot (temperature vs. ice cream sales) with a trend line.

Answer:

1. Color Lookup Table (Simplified):

cpp
Copy
Download

```
// Define a lookup table with 256 entries
GLfloat colorTable[256][3];

void buildColorTable() {
   for (int i = 0; i < 256; i++) {
      colorTable[i][0] = i / 255.0; // Red
      colorTable[i][1] = 0.5;       // Green (fixed)
      colorTable[i][2] = 1.0 - i/255.0; // Blue
   }
}

// Usage:
glColor3fv(colorTable[index]);
```

2. Scatter Plot with Trend Line:

cpp
Copy
Download

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

float temp[] = {14.2, 16.4, 11.9, 15.2, 18.5, 22.1, 19.4, 25.1, 23.4, 18.1, 22.6, 17.2};
float sales[] = {215, 325, 185, 332, 406, 522, 412, 614, 544, 421, 445, 408};

void drawLine(int x1, int y1, int x2, int y2) {
   // Bresenham line algorithm
   int dx = abs(x2 - x1), dy = abs(y2 - y1);
   int sx = (x1 < x2) ? 1 : -1, sy = (y1 < y2) ? 1 : -1;
   int err = dx - dy;

   while (true) {
```

```
        glVertex2i(x1, y1);
        if (x1 == x2 && y1 == y2) break;
        int e2 = 2 * err;
        if (e2 > -dy) { err -= dy; x1 += sx; }
        if (e2 < dx) { err += dx; y1 += sy; }
    }
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Plot data points (blue)
    glColor3f(0, 0, 1);
    glBegin(GL_POINTS);
    for (int i = 0; i < 12; i++)
        glVertex2f(temp[i], sales[i]/100.0); // Scale sales to fit window
    glEnd();

    // Trend line (orange)
    glColor3f(1, 0.65, 0);
    glBegin(GL_LINES);
    drawLine(10, 200/100, 26, 700/100); // Example coordinates
    glEnd();

    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutCreateWindow("Scatter Plot");
    gluOrtho2D(10, 26, 0, 7); // Adjusted for temperature (X) and sales (Y)
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Group 11: Frame Buffer & Transformations
Question Requirements:

1. Calculate the frame buffer address for pixel (x, y).
2. Draw a figure, translate it, rotate it, and apply colors.

Answer:

1. Frame Buffer Address Calculation:

- Screen resolution: 12 inches * 120 PPI = 1440 pixels (width), 14 inches * 120 PPI = 1680 pixels (height).
- Each pixel = 4 bits = 0.5 bytes.
- Address of pixel (x, y):

Copy
Download

```
address = (y * 1440 + x) * 0.5
```

2. OpenGL Code:

cpp
Copy
Download

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

void drawFigure() {
    // Original coordinates: A(0,4), B(3,4), C(4,0), D(0,0)
    glBegin(GL_POLYGON);
    glVertex2f(0, 4);
    glVertex2f(3, 4);
    glVertex2f(4, 0);
    glVertex2f(0, 0);
    glEnd();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Translated figure (2,2)
    glPushMatrix();
    glTranslatef(2, 2, 0);
    glColor3f(0, 1, 0); // Green border
    glLineWidth(2);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    drawFigure();
```

```
    // Fill with cream
    glColor3f(1, 0.98, 0.82);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    drawFigure();
    glPopMatrix();

    // Rotate by 55 degrees
    glPushMatrix();
    glTranslatef(2, 2, 0); // Translate first
    glRotatef(55, 0, 0, 1); // Rotate around (2,2)
    glColor3f(0, 1, 0);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    drawFigure();
    glPopMatrix();

    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutCreateWindow("Transformations");
    gluOrtho2D(-5, 10, -5, 10);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Group 12: Transformation Matrices & Shaded Triangle
Question Requirements:

1. Prove commutativity of transformation matrices.
2. Draw a rotated triangle with mixed primary colors.

Answer:
1. Commutativity Proof:

- **Translations**: T1 + T2 = T2 + T1 (commutative).
- **Scalings**: S1 * S2 = S2 * S1 (commutative).

- **Rotations**: Only commutative if around the same axis.

2. OpenGL Code:

cpp
Copy
Download

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

void drawTriangle() {
    glBegin(GL_TRIANGLES);
    glColor3f(1, 0, 0); // Red
    glVertex2f(-1.6, 2.0);
    glColor3f(0, 1, 0); // Green
    glVertex2f(2.0, 0.0);
    glColor3f(0, 0, 1); // Blue
    glVertex2f(-4.9, 9.0);
    glEnd();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Rotate triangle by -45 degrees
    glPushMatrix();
    glRotatef(-45, 0, 0, 1);
    drawTriangle();
    glPopMatrix();

    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutCreateWindow("Triangle");
    gluOrtho2D(-10, 10, -10, 10);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Group 13: Boundary-Fill & Parabola Scan Conversion

Question Requirements:

1. Write a boundary-fill procedure for an 8-connected region.
2. Scan convert the parabola $y=50-x2y=50-x2$ using the midpoint method.

Answer:

1. Boundary-Fill Code:

cpp
Copy
Download

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

void boundaryFill(int x, int y, float* fillColor, float* borderColor) {
   float current[3];
   glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, current);

   // Check if pixel is not border and not filled
   if (current[0] != borderColor[0] || current[1] != borderColor[1] || current[2] != 
borderColor[2]) {
      if (current[0] != fillColor[0] || current[1] != fillColor[1] || current[2] != fillColor[2]) {
         glBegin(GL_POINTS);
         glColor3fv(fillColor);
         glVertex2i(x, y);
         glEnd();
         glFlush();

         // 8-connected recursion
         boundaryFill(x+1, y, fillColor, borderColor);
         boundaryFill(x-1, y, fillColor, borderColor);
         boundaryFill(x, y+1, fillColor, borderColor);
         boundaryFill(x, y-1, fillColor, borderColor);
         boundaryFill(x+1, y+1, fillColor, borderColor);
         boundaryFill(x-1, y-1, fillColor, borderColor);
         boundaryFill(x+1, y-1, fillColor, borderColor);
         boundaryFill(x-1, y+1, fillColor, borderColor);
      }
   }
}
```

2. Midpoint Parabola Code:

```cpp
Copy
Download
void drawParabola() {
   int x, y;
   for (x = -4; x <= 4; x++) {
      y = 50 - x*x; // Equation: y = 50 - x^2
      glBegin(GL_POINTS);
      glVertex2i(x, y);
      glEnd();
   }
}

// Call in display():
glColor3f(1,1,1);
drawParabola();
```

Group 14: Color & Texture
Question Requirements:

1. Set OpenGL display color to green and fill a circle with texture.

Answer:

```cpp
Copy
Download
#include <GL/glut.h>

void display() {
   glClear(GL_COLOR_BUFFER_BIT);

   // Green border
   glColor3f(0.0, 1.0, 0.0);
   glLineWidth(3);
   glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
   glutSolidSphere(4.0, 50, 50);

   // Textured fill (simplified)
   glColor3f(0.5, 0.5, 0.5); // Gray texture placeholder
```

```
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glutSolidSphere(4.0, 50, 50);

    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutCreateWindow("Circle");
    gluOrtho2D(-10, 10, -10, 10);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Group 15: Ellipse & Transformations
Question Requirements:

1. Draw an ellipse $(x-2)^2/36+(y+1)^2/25=1$ i.e. $36(x-2)^2+25(y+1)^2=1$.
2. Apply flood-fill, shear, and anti-aliasing.

Answer:
1. Ellipse Code:

```
cpp
Copy
Download
void drawEllipse() {
    float a = 6, b = 5; // Semi-axes
    for (float theta = 0; theta < 360; theta += 1) {
        float rad = theta * 3.14159 / 180;
        float x = 2 + a * cos(rad); // Center at (2, -1)
        float y = -1 + b * sin(rad);
        glVertex2f(x, y);
    }
}

// Call in display():
glBegin(GL_POINTS);
drawEllipse();
glEnd();
```

2. Shear Transformation:

```
cpp
Copy
Download
// Shear matrix: X' = x + 2y, Y' = y + 2x
for (float theta = 0; theta < 360; theta += 1) {
   float rad = theta * 3.14159 / 180;
   float x = 2 + 6 * cos(rad);
   float y = -1 + 5 * sin(rad);
   float xShear = x + 2*y;
   float yShear = y + 2*x;
   glVertex2f(xShear, yShear);
}
```

Group 16: Coordinate Systems & Chessboard

Question Requirements:

1. Describe coordinate systems (OCS, WCS, VCS, etc.) and explain homogeneous coordinates.
2. Draw an 8x8 chessboard with brown and white squares.

Answer:

1. Coordinate Systems:

- **OCS (Object Coordinate System)**: Local to an object (e.g., a cube centered at its origin).
- **WCS (World Coordinate System)**: Global space where all objects are placed.
- **VCS (Viewing Coordinate System)**: Camera/viewer's perspective.
- **CCS (Clipping Coordinate System)**: Defines the visible region.
- **NDC (Normalized Device Coordinates)**: Coordinates normalized to [-1, 1].
- **DCS (Device Coordinate System)**: Final screen/pixel coordinates.

Homogeneous Coordinates: Allow transformations (translation, rotation, scaling) to be represented as matrix multiplications. A point (x,y) is written as $(x,y,1)$ in 2D.

2. Chessboard Code:

```
cpp
Copy
Download
```

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

void drawChessboard() {
    bool isBrown = true;
    for (int row = 0; row < 8; row++) {
        for (int col = 0; col < 8; col++) {
            if ((row + col) % 2 == 0)
                glColor3f(0.6, 0.3, 0.1); // Brown
            else
                glColor3f(1.0, 1.0, 1.0); // White

            glBegin(GL_QUADS);
                glVertex2i(col, row);
                glVertex2i(col + 1, row);
                glVertex2i(col + 1, row + 1);
                glVertex2i(col, row + 1);
            glEnd();
        }
    }
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    drawChessboard();
    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutCreateWindow("Chessboard");
    gluOrtho2D(0, 8, 0, 8);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Group 17: Mandelbrot Set
Question Requirements:

1. Explain the Mandelbrot Set.
2. Provide OpenGL code to draw it.

Answer:

1. Mandelbrot Set:

A fractal defined by complex numbers c where the sequence $z_{n+1} = z_n^2 + c$ does not diverge (starting from $z_0 = 0$).

2. OpenGL Code (Simplified):

cpp
Copy
Download

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

void drawMandelbrot() {
  glBegin(GL_POINTS);
  for (float x = -2.0; x < 1.0; x += 0.003) {
    for (float y = -1.5; y < 1.5; y += 0.003) {
      std::complex<float> c(x, y);
      std::complex<float> z(0, 0);
      int iter = 0;

      while (abs(z) < 2 && iter < 100) {
        z = z * z + c;
        iter++;
      }

      if (iter == 100) glColor3f(0, 0, 0); // Black for Mandelbrot
      else glColor3f(1, 1, 1); // White for divergence
      glVertex2f(x, y);
    }
  }
  glEnd();
}

void display() {
  glClear(GL_COLOR_BUFFER_BIT);
  drawMandelbrot();
  glFlush();
}
```

```
int main(int argc, char** argv) {
   glutInit(&argc, argv);
   glutCreateWindow("Mandelbrot");
   gluOrtho2D(-2, 1, -1.5, 1.5);
   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```

Group 18: Polyhedra & Teapot
Question Requirements:

1. Generate polyhedra using tessellation and GLUT.
2. Draw the Utah teapot.

Answer:
1. Surface Tessellation (Tetrahedron):

cpp
Copy
Download

```
void drawTetrahedron() {
   glBegin(GL_TRIANGLES);
   // Face 1
   glVertex3f(0, 1, 0);
   glVertex3f(-1, -1, 1);
   glVertex3f(1, -1, 1);

   // Face 2
   glVertex3f(0, 1, 0);
   glVertex3f(1, -1, 1);
   glVertex3f(1, -1, -1);

   // Face 3
   glVertex3f(0, 1, 0);
   glVertex3f(1, -1, -1);
   glVertex3f(-1, -1, -1);

   // Face 4
   glVertex3f(0, 1, 0);
```

```
    glVertex3f(-1, -1, -1);
    glVertex3f(-1, -1, 1);
    glEnd();
}
```

2. Utah Teapot using GLUT:

cpp
Copy
Download

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

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.5, 0.0); // Orange teapot
    glutSolidTeapot(1.0); // Radius = 1.0
    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutCreateWindow("Teapot");
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Group 19: 3D Clipping & Cohen-Sutherland/Cyrus-Beck

Question Requirements:

1. Illustrate 3D clipping and viewing.
2. Apply Cohen-Sutherland and Cyrus-Beck algorithms.

Answer:

1. 3D Clipping:

- Use glClipPlane to define clipping planes.

2. Code (Simplified):

cpp

Copy
Download

```
glEnable(GL_CLIP_PLANE0);
GLdouble eq[] = {0, 0, 1, 0}; // Clip everything behind Z=0
glClipPlane(GL_CLIP_PLANE0, eq);

// Draw a cube
glutSolidCube(2.0);
```

Algorithm Summary:

- **Cohen-Sutherland**: Assign region codes to vertices (e.g., 6-bit for 3D). Clip lines against view volume.
- **Cyrus-Beck**: Parametric line-clipping for convex polygons.

Group 20: Koch Curve & Snowflake
Question Requirements:

1. Describe the Koch curve.
2. Draw a 2D Koch snowflake.

Answer:
1. Koch Curve: Replace each line segment with 4 segments (recursive).
2. Code:
cpp
Copy
Download

```
void koch(float x1, float y1, float x2, float y2, int depth) {
  if (depth == 0) {
    glVertex2f(x1, y1);
    glVertex2f(x2, y2);
  } else {
    // Split into 4 segments and recurse
    float dx = (x2 - x1)/3, dy = (y2 - y1)/3;
    float xA = x1 + dx, yA = y1 + dy;
    float xB = x2 - dx, yB = y2 - dy;
    float xC = (xA + xB)/2 - dy * sqrt(3)/2;
    float yC = (yA + yB)/2 + dx * sqrt(3)/2;

    koch(x1, y1, xA, yA, depth-1);
```

```
    koch(xA, yA, xC, yC, depth-1);
    koch(xC, yC, xB, yB, depth-1);
    koch(xB, yB, x2, y2, depth-1);
  }
}

// Draw snowflake: Call koch() 3 times around a triangle.
```

Group 21: Concave Polygon Splitting & Shading
Question Requirements:

1. Split a concave polygon using rotational method.
2. Describe Gouraud, Phong, and Faceted shading.

Answer:
1. Concave Polygon Splitting:

```
cpp
Copy
Download
void splitConcavePolygon() {
  // Iterate through vertices, find "ears," and split
  // (Implementation requires complex vertex traversal)
}
```

2. Shading Techniques:

- **Gouraud**: Interpolate colors across polygon vertices.
- **Phong**: Interpolate normals and compute per-pixel lighting.
- **Faceted**: Flat shading with one color per polygon.

Group 22: Hilbert Curve
Question Requirements:

1. Draw a Hilbert curve.

Answer:
Code:

```
cpp
Copy
Download
```

```
void hilbert(float x, float y, float size, int depth, int dir) {
   if (depth == 0) return;
   hilbert(x, y, size/2, depth-1, (dir+3)%4);
   drawLine(x, y, dir, size); // Implement drawLine()
   hilbert(x + dx, y + dy, size/2, depth-1, dir);
   // Repeat for all quadrants
}
```

Group 23: Filled Polygon Operations
Question Requirements:

1. Draw a filled polygon, scale it, and fill with asterisks.

Answer:
1. Draw Polygon:

```
cpp
Copy
Download
glBegin(GL_POLYGON);
glVertex2f(8,4); glVertex2f(2,4); glVertex2f(0,8);
glVertex2f(3,12); glVertex2f(7,12); glVertex2f(10,8);
glEnd();
```

2. Scaling:

```
cpp
Copy
Download
glPushMatrix();
glScalef(2.0, 2.0, 1.0); // Scale by factor 2
// Redraw polygon
glPopMatrix();
```

Group 24: Phong & Gouraud Shading in Blender
Question Requirements:

1. Create videos for both shading models.

Answer:
Steps in Blender:

1. Add object (e.g., sphere).

2. **Gouraud**: Set material shading to "Smooth" and enable "Auto Smooth."
3. **Phong**: Use a Principled BSDF shader with high specular.
4. Render animation and export video.

Group 25: Triangular Meshes for Pyramid
Question Requirements:

1. Build an Egyptian pyramid using triangular meshes.

Answer:
OpenGL Code:

```
cpp
Copy
Download
void drawPyramid() {
  glBegin(GL_TRIANGLES);
  // Base (4 triangles)
  glVertex3f(-3.5, 0, -2); glVertex3f(3.5, 0, -2); glVertex3f(0, 0, 4);
  // Sides
  glVertex3f(-3.5, 0, -2); glVertex3f(0, 3, 0); glVertex3f(3.5, 0, -2);
  // Repeat for other faces
  glEnd();
}
```