ਜਗਤ ਗੁਰੂ ਨਾਨਕ ਦੇਵ
ਪੰਜਾਬ ਸਟੇਟ ਓਪਨ ਯੂਨੀਵਰਸਿਟੀ
ਪਟਿਆਲਾ

JAGAT GURU NANAK DEV
PUNJAB STATE OPEN UNIVERSITY, PATIALA

(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)

The Motto of the University

(SEWA)

SKILL ENHANCEMENT **EMPLOYABILITY** **WISDOM** **ACCESSIBILITY**

Bachelor of Computer Applications (BCA)

Course : Computer Graphics Lab

Course Code: BCA-4-04P

ADDRESS: C/28, THE LOWER MALL, PATIALA-147001
WEBSITE: www.psou.ac.in

JAGAT GURU NANAK DEV

PUNJAB STATE OPEN UNIVERSITY PATIALA

(Established by Act No.19 of 2019 of Legislature of the State of Punjab)

PROGRAMME COORDINATOR :

Dr. Monika Pathak

Assistant Professor, School of Sciences and Emerging Technologies

Jagat Guru Nanak Dev Punjab State Open University, Patiala

PROGRAMME CO-COORDINATOR & COURSE COORDINATOR :

Dr. Gaurav Dhiman

Assistant Professor, School of Sciences and Emerging Technologies

Jagat Guru Nanak Dev Punjab State Open University, Patiala

JAGAT GURU NANAK DEV

PUNJAB STATE OPEN UNIVERSITY PATIALA

(Established by Act No.19 of 2019 of Legislature of the State of Punjab)

PREFACE

Jagat Guru Nanak Dev Punjab State Open University, Patiala was established in Decembas 2019 by Act 19 of the Legislature of State of Punjab. It is the first and only Open Universit of the State, entrusted with the responsibility of making higher education accessible to all especially to those sections of society who do not have the means, time or opportunity to pursue regular education.

In keeping with the nature of an Open University, this University provides a flexible education system to suit every need. The time given to complete a programme is double the duration of a regular mode programme. Well-designed study material has been prepared in consultation with experts in their respective fields.

The University offers programmes which have been designed to provide relevant, skill-based and employability-enhancing education. The study material provided in this booklet is self instructional, with self-assessment exercises, and recommendations for further readings. The syllabus has been divided in sections, and provided as units for simplification.

The Learner Support Centres/Study Centres are located in the Government and Government aided colleges of Punjab, to enable students to make use of reading facilities, and for curriculum-based counselling and practicals. We, at the University, welcome you to be a part of this institution of knowledge.

Prof. G. S. Batra,

Dean Academic Affairs

Bachelor of Computer Applications (BCA)
BCA-4-04P: Computer Graphics Lab

Total Marks: 50
External Marks: 35
Internal Marks: 15
Credits: 2
Pass Percentage: 40%

Course: Computer Graphics Lab	
Course Code: BCA-4-04P	
Course Outcomes (COs) After the completion of this course, the students will be able to:	
CO1	Implement the basic concepts of computer graphics.
CO2	Design & Implement scan conversion problems using Python Programming
CO3	Apply clipping and filling techniques for modifying an object.
CO4	Understand the concepts of different type of geometric transformation of objects in 2D and 3D.
CO5	Understand the practical implementation of modeling, rendering, viewing of objects in 2D.

Detailed List of Programs:

Program No.	Name of Program
P1	Write a program to draw basic geometric shapes (lines, circles, rectangles) using a graphics library.
P2	Implement a program that allows the user to interactively draw and manipulate shapes on a canvas.
P3	WAP that demonstrates 2D transformations (translation, rotation, scaling) on a set of objects.
P4	Extend the program to include 3D transformations and demonstrate their effects.

P5	Implement line-drawing algorithms.
P6	Implement circle-drawing algorithms.
P7	Develop a program for polygon filling using scanline or other suitable algorithms.
P8	Create a program that renders objects with different shading models.
P9	Explore the impact of lighting models on the visual appearance of 3D objects in a scene.
P10	Implement texture mapping on a 3D model, and observe the changes in the rendered output.

Bachelor of Computer Applications (BCA)
Computer Graphics Lab

1. **Write a program to draw basic geometric shapes (lines, circles, rectangles) using a graphics library.**

```
#include <SDL2/SDL.h>
#include <stdio.h>

void drawShapes(SDL_Renderer *renderer) {
    // Draw a line
    SDL_SetRenderDrawColor(renderer, 255, 0, 0, 255); // Red color
    SDL_RenderDrawLine(renderer, 100, 100, 200, 200);

    // Draw a rectangle
    SDL_Rect rect = {300, 100, 200, 150};
    SDL_SetRenderDrawColor(renderer, 0, 255, 0, 255); // Green color
    SDL_RenderDrawRect(renderer, &rect);

    // Draw a filled rectangle
    SDL_SetRenderDrawColor(renderer, 0, 0, 255, 255); // Blue color
    SDL_RenderFillRect(renderer, &rect);

    // Draw a circle
    int centerX = 500, centerY = 300, radius = 100;
    SDL_SetRenderDrawColor(renderer, 255, 255, 0, 255); // Yellow color
    for (int w = 0; w < radius * 2; w++) {
        for (int h = 0; h < radius * 2; h++) {
            int dx = radius - w; // horizontal offset
            int dy = radius - h; // vertical offset
            if ((dx * dx + dy * dy) <= (radius * radius)) {
                SDL_RenderDrawPoint(renderer, centerX + dx, centerY + dy);
            }
        }
    }
}

int main(int argc, char *argv[]) {
    SDL_Init(SDL_INIT_VIDEO);

    SDL_Window *window = SDL_CreateWindow("Basic Shapes",
SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED, 800, 600,
SDL_WINDOW_SHOWN);
    SDL_Renderer *renderer = SDL_CreateRenderer(window, -1,
SDL_RENDERER_ACCELERATED);
```

```
    SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255); // White color
    SDL_RenderClear(renderer);

    drawShapes(renderer);

    SDL_RenderPresent(renderer);

    SDL_Delay(5000); // Pause for 5 seconds

    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();

    return 0;
}
```

2. **Implement a program that allows the user to interactively draw and manipulate shapes on a canvas.**

```
#include <SDL2/SDL.h>
#include <stdio.h>

void handleMouseEvents(SDL_Event *event, SDL_Renderer *renderer, int *drawing,
SDL_Rect *rect) {
    switch (event->type) {
        case SDL_MOUSEBUTTONDOWN:
            if (event->button.button == SDL_BUTTON_LEFT) {
                rect->x = event->button.x;
                rect->y = event->button.y;
                *drawing = 1;
            }
            break;
        case SDL_MOUSEMOTION:
            if (*drawing) {
                rect->w = event->motion.x - rect->x;
                rect->h = event->motion.y - rect->y;
            }
            break;
        case SDL_MOUSEBUTTONUP:
            if (event->button.button == SDL_BUTTON_LEFT) {
                *drawing = 0;
            }
            break;
    }
}
```

```
int main(int argc, char *argv[]) {
    SDL_Init(SDL_INIT_VIDEO);

    SDL_Window *window = SDL_CreateWindow("Interactive Drawing",
SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED, 800, 600,
SDL_WINDOW_SHOWN);
    SDL_Renderer *renderer = SDL_CreateRenderer(window, -1,
SDL_RENDERER_ACCELERATED);

    int quit = 0;
    SDL_Event event;
    int drawing = 0;
    SDL_Rect rect = {0, 0, 0, 0};

    while (!quit) {
        while (SDL_PollEvent(&event) != 0) {
            if (event.type == SDL_QUIT) {
                quit = 1;
            }
            handleMouseEvents(&event, renderer, &drawing, &rect);
        }

        SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255); // White color
        SDL_RenderClear(renderer);

        if (drawing) {
            SDL_SetRenderDrawColor(renderer, 0, 0, 255, 255); // Blue color
            SDL_RenderDrawRect(renderer, &rect);
        }

        SDL_RenderPresent(renderer);
    }

    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();

    return 0;
}
```

3. **WAP that demonstrates 2D transformations (translation, rotation, scaling) on a set of objects.**

```
#include <SDL2/SDL.h>
#include <stdio.h>
```

```
#include <math.h>

void drawRect(SDL_Renderer *renderer, SDL_Rect *rect) {
   SDL_RenderDrawRect(renderer, rect);
}

void translateRect(SDL_Rect *rect, int dx, int dy) {
   rect->x += dx;
   rect->y += dy;
}

void rotateRect(SDL_Renderer *renderer, SDL_Rect *rect, double angle) {
   SDL_Point center = {rect->x + rect->w / 2, rect->y + rect->h / 2};
   SDL_RenderDrawRect(renderer, rect);
   SDL_RenderCopyEx(renderer, NULL, rect, NULL, angle, &center,
SDL_FLIP_NONE);
}

void scaleRect(SDL_Rect *rect, double sx, double sy) {
   rect->w *= sx;
   rect->h *= sy;
}

int main(int argc, char *argv[]) {
   SDL_Init(SDL_INIT_VIDEO);

   SDL_Window *window = SDL_CreateWindow("2D Transformations",
SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED, 800, 600,
SDL_WINDOW_SHOWN);
   SDL_Renderer *renderer = SDL_CreateRenderer(window, -1,
SDL_RENDERER_ACCELERATED);

   SDL_Rect rect = {200, 150, 100, 50};
   int quit = 0;
   SDL_Event event;

   while (!quit) {
      while (SDL_PollEvent(&event) != 0) {
         if (event.type == SDL_QUIT) {
            quit = 1;
         }
         if (event.type == SDL_KEYDOWN) {
            switch (event.key.keysym.sym) {
               case SDLK_RIGHT:
                  translateRect(&rect, 10, 0);
                  break;
```

```
                case SDLK_LEFT:
                    translateRect(&rect, -10, 0);
                    break;
                case SDLK_UP:
                    translateRect(&rect, 0, -10);
                    break;
                case SDLK_DOWN:
                    translateRect(&rect, 0, 10);
                    break;
                case SDLK_r:
                    rotateRect(renderer, &rect, 45);
                    break;
                case SDLK_s:
                    scaleRect(&rect, 1.5, 1.5);
                    break;
            }
        }
    }

    SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255); // White color
    SDL_RenderClear(renderer);

    SDL_SetRenderDrawColor(renderer, 0, 0, 255, 255); // Blue color
    drawRect(renderer, &rect);

    SDL_RenderPresent(renderer);
  }

  SDL_DestroyRenderer(renderer);
  SDL_DestroyWindow(window);
  SDL_Quit();

  return 0;
}
```

4. **Extend the program to include 3D transformations and demonstrate their effects.**

```
#include <SDL2/SDL.h>
#include <GL/glew.h>
#include <GL/gl.h>
#include <GL/glu.h>

void initOpenGL() {
  glEnable(GL_DEPTH_TEST);
  glMatrixMode(GL_PROJECTION);
```

```
    glLoadIdentity();
    gluPerspective(45.0, 4.0/3.0, 1.0, 100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void renderCube() {
    glBegin(GL_QUADS);

    glColor3f(1.0, 0.0, 0.0); // Red
    glVertex3f(1.0, 1.0, -1.0);
    glVertex3f(-1.0, 1.0, -1.0);
    glVertex3f(-1.0, 1.0, 1.0);
    glVertex3f(1.0, 1.0, 1.0);

    glColor3f(0.0, 1.0, 0.0); // Green
    glVertex3f(1.0, -1.0, 1.0);
    glVertex3f(-1.0, -1.0, 1.0);
    glVertex3f(-1.0, -1.0, -1.0);
    glVertex3f(1.0, -1.0, -1.0);

    glColor3f(0.0, 0.0, 1.0); // Blue
    glVertex3f(1.0, 1.0, 1.0);
    glVertex3f(-1.0, 1.0, 1.0);
    glVertex3f(-1.0, -1.0, 1.0);
    glVertex3f(1.0, -1.0, 1.0);

    glColor3f(1.0, 1.0, 0.0); // Yellow
    glVertex3f(1.0, -1.0, -1.0);
    glVertex3f(-1.0, -1.0, -1.0);
    glVertex3f(-1.0, 1.0, -1.0);
    glVertex3f(1.0, 1.0, -1.0);

    glColor3f(0.0, 1.0, 1.0); // Cyan
    glVertex3f(-1.0, 1.0, 1.0);
    glVertex3f(-1.0, 1.0, -1.0);
    glVertex3f(-1.0, -1.0, -1.0);
    glVertex3f(-1.0, -1.0, 1.0);

    glColor3f(1.0, 0.0, 1.0); // Magenta
    glVertex3f(1.0, 1.0, -1.0);
    glVertex3f(1.0, 1.0, 1.0);
    glVertex3f(1.0, -1.0, 1.0);
    glVertex3f(1.0, -1.0, -1.0);

    glEnd();
```

```
}

int main(int argc, char *argv[]) {
    SDL_Init(SDL_INIT_VIDEO);
    SDL_Window *window = SDL_CreateWindow("3D Transformations",
SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED, 800, 600,
SDL_WINDOW_OPENGL);
    SDL_GLContext glContext = SDL_GL_CreateContext(window);
    glewInit();

    initOpenGL();

    int quit = 0;
    SDL_Event event;

    float angle = 0.0f;

    while (!quit) {
        while (SDL_PollEvent(&event) != 0) {
            if (event.type == SDL_QUIT) {
                quit = 1;
            }
        }

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

        glPushMatrix();
        glTranslatef(0.0f, 0.0f, -5.0f);
        glRotatef(angle, 1.0f, 1.0f, 1.0f);
        renderCube();
        glPopMatrix();

        SDL_GL_SwapWindow(window);
        angle += 0.5f;
    }

    SDL_GL_DeleteContext(glContext);
    SDL_DestroyWindow(window);
    SDL_Quit();

    return 0;
}
```

5. **Implement line-drawing algorithms.**

```
#include <SDL2/SDL.h>
#include <stdio.h>

void drawLine(SDL_Renderer *renderer, int x1, int y1, int x2, int y2) {
    int dx = abs(x2 - x1);
    int dy = abs(y2 - y1);
    int sx = (x1 < x2) ? 1 : -1;
    int sy = (y1 < y2) ? 1 : -1;
    int err = dx - dy;

    while (1) {
        SDL_RenderDrawPoint(renderer, x1, y1);
        if (x1 == x2 && y1 == y2) break;
        int e2 = err * 2;
        if (e2 > -dy) {
            err -= dy;
            x1 += sx;
        }
        if (e2 < dx) {
            err += dx;
            y1 += sy;
        }
    }
}

int main(int argc, char *argv[]) {
    SDL_Init(SDL_INIT_VIDEO);

    SDL_Window *window = SDL_CreateWindow("Line Drawing",
SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED, 800, 600,
SDL_WINDOW_SHOWN);
    SDL_Renderer *renderer = SDL_CreateRenderer(window, -1,
SDL_RENDERER_ACCELERATED);

    SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255); // White color
    SDL_RenderClear(renderer);

    SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255); // Black color
    drawLine(renderer, 100, 100, 700, 500);

    SDL_RenderPresent(renderer);

    SDL_Delay(5000); // Pause for 5 seconds

    SDL_DestroyRenderer(renderer);
```

```
    SDL_DestroyWindow(window);
    SDL_Quit();

    return 0;
}
```

6. Implement circle-drawing algorithms.

```
#include <SDL2/SDL.h>
#include <stdio.h>

void drawCircle(SDL_Renderer *renderer, int centerX, int centerY, int radius) {
    int x = radius;
    int y = 0;
    int radiusError = 1 - x;

    while (x >= y) {
        SDL_RenderDrawPoint(renderer, centerX + x, centerY + y);
        SDL_RenderDrawPoint(renderer, centerX - x, centerY + y);
        SDL_RenderDrawPoint(renderer, centerX + x, centerY - y);
        SDL_RenderDrawPoint(renderer, centerX - x, centerY - y);
        SDL_RenderDrawPoint(renderer, centerX + y, centerY + x);
        SDL_RenderDrawPoint(renderer, centerX - y, centerY + x);
        SDL_RenderDrawPoint(renderer, centerX + y, centerY - x);
        SDL_RenderDrawPoint(renderer, centerX - y, centerY - x);
        y++;
        if (radiusError < 0) {
            radiusError += 2 * y + 1;
        } else {
            x--;
            radiusError += 2 * (y - x + 1);
        }
    }
}

int main(int argc, char *argv[]) {
    SDL_Init(SDL_INIT_VIDEO);

    SDL_Window *window = SDL_CreateWindow("Circle Drawing",
SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED, 800, 600,
SDL_WINDOW_SHOWN);
    SDL_Renderer *renderer = SDL_CreateRenderer(window, -1,
SDL_RENDERER_ACCELERATED);

    SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255); // White color
    SDL_RenderClear(renderer);
```

```
    SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255); // Black color
    drawCircle(renderer, 400, 300, 100);

    SDL_RenderPresent(renderer);

    SDL_Delay(5000); // Pause for 5 seconds

    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();

    return 0;
}
```

7. Develop a program for polygon filling using scanline or other suitable algorithms.

```
#include <SDL2/SDL.h>
#include <stdio.h>

typedef struct {
    int x;
    int y;
} Point;

void drawPolygon(SDL_Renderer *renderer, Point *points, int n) {
    for (int i = 0; i < n; i++) {
        SDL_RenderDrawLine(renderer, points[i].x, points[i].y, points[(i + 1) % n].x,
points[(i + 1) % n].y);
    }
}

void fillPolygon(SDL_Renderer *renderer, Point *points, int n) {
    int minY = points[0].y, maxY = points[0].y;

    // Find the min and max Y
    for (int i = 1; i < n; i++) {
        if (points[i].y < minY) minY = points[i].y;
        if (points[i].y > maxY) maxY = points[i].y;
    }

    for (int y = minY; y <= maxY; y++) {
        int intersections[64], count = 0;

        for (int i = 0; i < n; i++) {
```

```
            int j = (i + 1) % n;
            if ((points[i].y <= y && points[j].y > y) || (points[j].y <= y && points[i].y >
y)) {
                float x = points[i].x + (y - points[i].y) * (points[j].x - points[i].x) /
(float)(points[j].y - points[i].y);
                intersections[count++] = (int)x;
            }
        }

        // Sort intersections
        for (int i = 0; i < count - 1; i++) {
            for (int j = i + 1; j < count; j++) {
                if (intersections[i] > intersections[j]) {
                    int temp = intersections[i];
                    intersections[i] = intersections[j];
                    intersections[j] = temp;
                }
            }
        }

        for (int i = 0; i < count; i += 2) {
            SDL_RenderDrawLine(renderer, intersections[i], y, intersections[i + 1], y);
        }
    }
}

int main(int argc, char *argv[]) {
    SDL_Init(SDL_INIT_VIDEO);

    SDL_Window *window = SDL_CreateWindow("Polygon Filling",
SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED, 800, 600,
SDL_WINDOW_SHOWN);
    SDL_Renderer *renderer = SDL_CreateRenderer(window, -1,
SDL_RENDERER_ACCELERATED);

    SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255);
    SDL_RenderClear(renderer);

    SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255);
    Point points[] = {{100, 100}, {200, 50}, {300, 100}, {350, 200}, {250, 250}, {150,
200}};
    int n = sizeof(points) / sizeof(points[0]);
    fillPolygon(renderer, points, n);
    drawPolygon(renderer, points, n);

    SDL_RenderPresent(renderer);
```

```
    SDL_Delay(5000);

    SDL_DestroyRenderer(renderer);
    SDL_DestroyWindow(window);
    SDL_Quit();

    return 0;
}
```

8. Create a program that renders objects with different shading models.

```
#include <SDL2/SDL.h>
#include <GL/glew.h>
#include <GL/gl.h>
#include <GL/glu.h>

void initOpenGL() {
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_COLOR_MATERIAL);
    glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);

    GLfloat lightPos[] = {0.0f, 0.0f, 2.0f, 1.0f};
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, 4.0 / 3.0, 1.0, 100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void drawCube() {
    glBegin(GL_QUADS);

    // Front face
    glNormal3f(0.0, 0.0, 1.0);
    glVertex3f(-1.0, -1.0, 1.0);
    glVertex3f(1.0, -1.0, 1.0);
    glVertex3f(1.0, 1.0, 1.0);
    glVertex3f(-1.0, 1.0, 1.0);

    // Back face
    glNormal3f(0.0, 0.0, -1.0);
```

```
    glVertex3f(-1.0, -1.0, -1.0);
    glVertex3f(-1.0, 1.0, -1.0);
    glVertex3f(1.0, 1.0, -1.0);
    glVertex3f(1.0, -1.0, -1.0);

    // Top face
    glNormal3f(0.0, 1.0, 0.0);
    glVertex3f(-1.0, 1.0, -1.0);
    glVertex3f(-1.0, 1.0, 1.0);
    glVertex3f(1.0, 1.0, 1.0);
    glVertex3f(1.0, 1.0, -1.0);

    // Bottom face
    glNormal3f(0.0, -1.0, 0.0);
    glVertex3f(-1.0, -1.0, -1.0);
    glVertex3f(1.0, -1.0, -1.0);
    glVertex3f(1.0, -1.0, 1.0);
    glVertex3f(-1.0, -1.0, 1.0);

    // Right face
    glNormal3f(1.0, 0.0, 0.0);
    glVertex3f(1.0, -1.0, -1.0);
    glVertex3f(1.0, 1.0, -1.0);
    glVertex3f(1.0, 1.0, 1.0);
    glVertex3f(1.0, -1.0, 1.0);

    // Left face
    glNormal3f(-1.0, 0.0, 0.0);
    glVertex3f(-1.0, -1.0, -1.0);
    glVertex3f(-1.0, -1.0, 1.0);
    glVertex3f(-1.0, 1.0, 1.0);
    glVertex3f(-1.0, 1.0, -1.0);

    glEnd();
}

int main(int argc, char *argv[]) {
    SDL_Init(SDL_INIT_VIDEO);
    SDL_Window *window = SDL_CreateWindow("Shading Models",
SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED, 800, 600,
SDL_WINDOW_OPENGL);
    SDL_GLContext glContext = SDL_GL_CreateContext(window);
    glewInit();

    initOpenGL();
```

```
    int quit = 0;
    SDL_Event event;
    float angle = 0.0f;
    int flatShading = 1;

    while (!quit) {
        while (SDL_PollEvent(&event) != 0) {
            if (event.type == SDL_QUIT) {
                quit = 1;
            }
            if (event.type == SDL_KEYDOWN) {
                if (event.key.keysym.sym == SDLK_s) {
                    flatShading = !flatShading;
                    glShadeModel(flatShading ? GL_FLAT : GL_SMOOTH);
                }
            }
        }

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

        glPushMatrix();
        glTranslatef(0.0f, 0.0f, -5.0f);
        glRotatef(angle, 1.0f, 1.0f, 1.0f);
        drawCube();
        glPopMatrix();

        SDL_GL_SwapWindow(window);
        angle += 0.5f;
    }

    SDL_GL_DeleteContext(glContext);
    SDL_DestroyWindow(window);
    SDL_Quit();

    return 0;
}
```

9. Explore the impact of lighting models on the visual appearance of 3D objects in a scene.

```
#include <SDL2/SDL.h>
#include <GL/glew.h>
#include <GL/gl.h>
#include <GL/glu.h>
```

```
void initOpenGL() {
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_COLOR_MATERIAL);
    glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);

    GLfloat lightPos[] = {0.0f, 0.0f, 2.0f, 1.0f};
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, 4.0 / 3.0, 1.0, 100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void drawCube() {
    glBegin(GL_QUADS);

    // Front face
    glNormal3f(0.0, 0.0, 1.0);
    glVertex3f(-1.0, -1.0, 1.0);
    glVertex3f(1.0, -1.0, 1.0);
    glVertex3f(1.0, 1.0, 1.0);
    glVertex3f(-1.0, 1.0, 1.0);

    // Back face
    glNormal3f(0.0, 0.0, -1.0);
    glVertex3f(-1.0, -1.0, -1.0);
    glVertex3f(-1.0, 1.0, -1.0);
    glVertex3f(1.0, 1.0, -1.0);
    glVertex3f(1.0, -1.0, -1.0);

    // Top face
    glNormal3f(0.0, 1.0, 0.0);
    glVertex3f(-1.0, 1.0, -1.0);
    glVertex3f(-1.0, 1.0, 1.0);
    glVertex3f(1.0, 1.0, 1.0);
    glVertex3f(1.0, 1.0, -1.0);

    // Bottom face
    glNormal3f(0.0, -1.0, 0.0);
    glVertex3f(-1.0, -1.0, -1.0);
    glVertex3f(1.0, -1.0, -1.0);
```

```
    glVertex3f(1.0, -1.0, 1.0);
    glVertex3f(-1.0, -1.0, 1.0);

    // Right face
    glNormal3f(1.0, 0.0, 0.0);
    glVertex3f(1.0, -1.0, -1.0);
    glVertex3f(1.0, 1.0, -1.0);
    glVertex3f(1.0, 1.0, 1.0);
    glVertex3f(1.0, -1.0, 1.0);

    // Left face
    glNormal3f(-1.0, 0.0, 0.0);
    glVertex3f(-1.0, -1.0, -1.0);
    glVertex3f(-1.0, -1.0, 1.0);
    glVertex3f(-1.0, 1.0, 1.0);
    glVertex3f(-1.0, 1.0, -1.0);

    glEnd();
}

int main(int argc, char *argv[]) {
    SDL_Init(SDL_INIT_VIDEO);
    SDL_Window *window = SDL_CreateWindow("Lighting Models",
SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED, 800, 600,
SDL_WINDOW_OPENGL);
    SDL_GLContext glContext = SDL_GL_CreateContext(window);
    glewInit();

    initOpenGL();

    int quit = 0;
    SDL_Event event;
    float angle = 0.0f;
    int lightingModel = 0;

    while (!quit) {
        while (SDL_PollEvent(&event) != 0) {
            if (event.type == SDL_QUIT) {
                quit = 1;
            }
            if (event.type == SDL_KEYDOWN) {
                if (event.key.keysym.sym == SDLK_l) {
                    lightingModel = (lightingModel + 1) % 3;

                    if (lightingModel == 0) {
```

```
                glLightfv(GL_LIGHT0, GL_DIFFUSE, (GLfloat[]){1.0, 1.0, 1.0,
1.0});
                glLightfv(GL_LIGHT0, GL_SPECULAR, (GLfloat[]){0.0, 0.0, 0.0,
1.0});
            } else if (lightingModel == 1) {
                glLightfv(GL_LIGHT0, GL_DIFFUSE, (GLfloat[]){1.0, 0.0, 0.0,
1.0});
                glLightfv(GL_LIGHT0, GL_SPECULAR, (GLfloat[]){1.0, 1.0, 1.0,
1.0});
            } else if (lightingModel == 2) {
                glLightfv(GL_LIGHT0, GL_DIFFUSE, (GLfloat[]){0.0, 0.0, 1.0,
1.0});
                glLightfv(GL_LIGHT0, GL_SPECULAR, (GLfloat[]){1.0, 1.0, 1.0,
1.0});
            }
        }
    }
}

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

glPushMatrix();
glTranslatef(0.0f, 0.0f, -5.0f);
glRotatef(angle, 1.0f, 1.0f, 1.0f);
drawCube();
glPopMatrix();

SDL_GL_SwapWindow(window);
angle += 0.5f;
}

SDL_GL_DeleteContext(glContext);
SDL_DestroyWindow(window);
SDL_Quit();

return 0;
}
```

10. Implement texture mapping on a 3D model, and observe the changes in the rendered output.

```
#include <SDL2/SDL.h>
#include <GL/glew.h>
#include <GL/gl.h>
```

```
#include <GL/glu.h>
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"

GLuint loadTexture(const char *filename) {
    GLuint texture;
    int width, height, nrChannels;
    unsigned char *data = stbi_load(filename, &width, &height, &nrChannels, 0);

    if (data) {
        glGenTextures(1, &texture);
        glBindTexture(GL_TEXTURE_2D, texture);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, nrChannels
== 4 ? GL_RGBA : GL_RGB, GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);
        stbi_image_free(data);
    } else {
        printf("Failed to load texture\n");
    }

    return texture;
}

void initOpenGL(GLuint *texture) {
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_COLOR_MATERIAL);
    glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);

    GLfloat lightPos[] = {0.0f, 0.0f, 2.0f, 1.0f};
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos);

    *texture = loadTexture("texture.jpg");
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, *texture);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, 4.0 / 3.0, 1.0, 100.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}

void drawTexturedCube() {
    glBegin(GL_QUADS);
```

```
    // Front face
    glNormal3f(0.0, 0.0, 1.0);
    glTexCoord2f(0.0, 0.0); glVertex3f(-1.0, -1.0, 1.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(1.0, -1.0, 1.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(1.0, 1.0, 1.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-1.0, 1.0, 1.0);

    // Back face
    glNormal3f(0.0, 0.0, -1.0);
    glTexCoord2f(0.0, 0.0); glVertex3f(-1.0, -1.0, -1.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-1.0, 1.0, -1.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(1.0, 1.0, -1.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(1.0, -1.0, -1.0);

    // Top face
    glNormal3f(0.0, 1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-1.0, 1.0, -1.0);
    glTexCoord2f(0.0, 0.0); glVertex3f(-1.0, 1.0, 1.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(1.0, 1.0, 1.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(1.0, 1.0, -1.0);

    // Bottom face
    glNormal3f(0.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-1.0, -1.0, -1.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(1.0, -1.0, -1.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(1.0, -1.0, 1.0);
    glTexCoord2f(0.0, 0.0); glVertex3f(-1.0, -1.0, 1.0);

    // Right face
    glNormal3f(1.0, 0.0, 0.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(1.0, -1.0, -1.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(1.0, 1.0, -1.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 1.0);
    glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 1.0);

    // Left face
    glNormal3f(-1.0, 0.0, 0.0);
    glTexCoord2f(0.0, 0.0); glVertex3f(-1.0, -1.0, -1.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(-1.0, -1.0, 1.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(-1.0, 1.0, 1.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-1.0, 1.0, -1.0);

    glEnd();
}
```

```
int main(int argc, char *argv[]) {
    SDL_Init(SDL_INIT_VIDEO);
    SDL_Window *window = SDL_CreateWindow("Texture Mapping",
SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED, 800, 600,
SDL_WINDOW_OPENGL);
    SDL_GLContext glContext = SDL_GL_CreateContext(window);
    glewInit();

    GLuint texture;
    initOpenGL(&texture);

    int quit = 0;
    SDL_Event event;
    float angle = 0.0f;

    while (!quit) {
        while (SDL_PollEvent(&event) != 0) {
            if (event.type == SDL_QUIT) {
                quit = 1;
            }
        }

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

        glPushMatrix();
        glTranslatef(0.0f, 0.0f, -5.0f);
        glRotatef(angle, 1.0f, 1.0f, 1.0f);
        drawTexturedCube();
        glPopMatrix();

        SDL_GL_SwapWindow(window);
        angle += 0.5f;
    }

    glDeleteTextures(1, &texture);
    SDL_GL_DeleteContext(glContext);
    SDL_DestroyWindow(window);
    SDL_Quit();

    return 0;
}
```