PRAKTIKUM 6

Tugas praktikum Mata Kuliah Grafika dan Komputasi Visual

Disusun Oleh :

Syahreza Abror Alvarizqi

(24060123140053)

DEPARTEMEN INFORMATIKA

FAKULTAS SAINS DAN MATEMATIKA

UNIVERSITAS DIPONEGORO

SEMARANG

2025

1. **Penerapan Tekstur pada Objek Kubus 3D Menggunakan OpenGL dengan GLFW dan GLEW**

Main.cpp

```
#include <GL/glew.h>
#include <GLFW/glfw3.h>
#include <iostream>
#include <cstdlib>
#include "imageloader.h"

const float BOX_SIZE = 7.0f;
float angle = 0.0f;
GLuint textureId;

void handleKeypress(GLFWwindow* window, int key, int scancode, int
action, int mods) {
    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
        glfwSetWindowShouldClose(window, GLFW_TRUE);
}

GLuint loadTexture(Image* image) {
    GLuint textureId;
    glGenTextures(1, &textureId);
    glBindTexture(GL_TEXTURE_2D, textureId);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->width, image->height, 0,
GL_RGB, GL_UNSIGNED_BYTE, image->pixels);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    return textureId;
}

void initRendering() {
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glEnable(GL_COLOR_MATERIAL);

    Image* image = loadBMP("852.bmp");
    textureId = loadTexture(image);
    delete image;
}
```

```
void drawScene() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0f, 0.0f, -20.0f);
    GLfloat ambientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientLight);

    GLfloat lightColor[] = { 0.7f, 0.7f, 0.7f, 1.0f };
    GLfloat lightPos[] = { -2 * BOX_SIZE, BOX_SIZE, 4 * BOX_SIZE, 1.0f
};

glLightfv(GL_LIGHT0, GL_DIFFUSE, lightColor);
glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
glRotatef(-angle, 1.0f, 1.0f, 0.0f);
glBegin(GL_QUADS);
glColor3f(1.0f, 1.0f, 0.0f);
glNormal3f(0.0, 1.0f, 0.0f);
glVertex3f(-BOX_SIZE / 2, BOX_SIZE / 2, -BOX_SIZE / 2);
glVertex3f(-BOX_SIZE / 2, BOX_SIZE / 2, BOX_SIZE / 2);
glVertex3f(BOX_SIZE / 2, BOX_SIZE / 2, BOX_SIZE / 2);
glVertex3f(BOX_SIZE / 2, BOX_SIZE / 2, -BOX_SIZE / 2);

glColor3f(1.0f, 0.0f, 1.0f);
glNormal3f(0.0, -1.0f, 0.0f);
glVertex3f(-BOX_SIZE / 2, -BOX_SIZE / 2, -BOX_SIZE / 2);
glVertex3f(BOX_SIZE / 2, -BOX_SIZE / 2, -BOX_SIZE / 2);
glVertex3f(BOX_SIZE / 2, -BOX_SIZE / 2, BOX_SIZE / 2);
glVertex3f(-BOX_SIZE / 2, -BOX_SIZE / 2, BOX_SIZE / 2);
glNormal3f(-1.0, 0.0f, 0.0f);
glColor3f(0.0f, 1.0f, 1.0f);
glVertex3f(-BOX_SIZE / 2, -BOX_SIZE / 2, -BOX_SIZE / 2);
glVertex3f(-BOX_SIZE / 2, -BOX_SIZE / 2, BOX_SIZE / 2);
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3f(-BOX_SIZE / 2, BOX_SIZE / 2, BOX_SIZE / 2);
glVertex3f(-BOX_SIZE / 2, BOX_SIZE / 2, -BOX_SIZE / 2);
glNormal3f(1.0, 0.0f, 0.0f);
glColor3f(1.0f, 0.0f, 0.0f);
glVertex3f(BOX_SIZE / 2, -BOX_SIZE / 2, -BOX_SIZE / 2);
glVertex3f(BOX_SIZE / 2, BOX_SIZE / 2, -BOX_SIZE / 2);
glColor3f(0.0f, 1.0f, 0.0f);
glVertex3f(BOX_SIZE / 2, BOX_SIZE / 2, BOX_SIZE / 2);
glVertex3f(BOX_SIZE / 2, -BOX_SIZE / 2, BOX_SIZE / 2);
glEnd();
```

```
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, textureId);
glColor3f(1.0f, 1.0f, 1.0f);
glBegin(GL_QUADS);
glNormal3f(0.0, 0.0f, 1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f(-BOX_SIZE / 2, -BOX_SIZE / 2,
BOX_SIZE / 2);
glTexCoord2f(1.0f, 0.0f); glVertex3f(BOX_SIZE / 2, -BOX_SIZE / 2,
BOX_SIZE / 2);
glTexCoord2f(1.0f, 1.0f); glVertex3f(BOX_SIZE / 2, BOX_SIZE / 2,
BOX_SIZE / 2);
glTexCoord2f(0.0f, 1.0f); glVertex3f(-BOX_SIZE / 2, BOX_SIZE / 2,
BOX_SIZE / 2);

glNormal3f(0.0, 0.0f, -1.0f);
glTexCoord2f(0.0f, 0.0f); glVertex3f(-BOX_SIZE / 2, -BOX_SIZE / 2,
-BOX_SIZE / 2);
glTexCoord2f(1.0f, 0.0f); glVertex3f(-BOX_SIZE / 2, BOX_SIZE / 2,
-BOX_SIZE / 2);
glTexCoord2f(1.0f, 1.0f); glVertex3f(BOX_SIZE / 2, BOX_SIZE / 2,
-BOX_SIZE / 2);
glTexCoord2f(0.0f, 1.0f); glVertex3f(BOX_SIZE / 2, -BOX_SIZE / 2,
-BOX_SIZE / 2);
glEnd();
glDisable(GL_TEXTURE_2D);
}

int main() {
if (!glfwInit()) {
std::cerr << "Failed to initialize GLFW\n";
return EXIT_FAILURE;
}

GLFWwindow* window = glfwCreateWindow(800, 600, "GLFW Texture Box",
NULL, NULL);
if (!window) {
std::cerr << "Failed to create window\n";
glfwTerminate();
return EXIT_FAILURE;
}
glfwMakeContextCurrent(window);
glewExperimental = GL_TRUE;
if (glewInit() != GLEW_OK) {
std::cerr << "Failed to initialize GLEW\n";
return EXIT_FAILURE;
```

```
}
glfwSetKeyCallback(window, handleKeypress);
glEnable(GL_DEPTH_TEST);
glMatrixMode(GL_PROJECTION);

glLoadIdentity();
gluPerspective(45.0, 800.0 / 600.0, 1.0, 200.0);
initRendering();
while (!glfwWindowShouldClose(window)) {
angle += 0.2f;
if (angle > 360) angle -= 360;
drawScene();
glfwSwapBuffers(window);
glfwPollEvents();
}
glfwDestroyWindow(window);
glfwTerminate();
return 0;
}
```

- **OpenGL** menyediakan kemampuan untuk memberikan tekstur pada objek 3D agar tampil lebih realistis. Tekstur adalah gambar 2D yang dibungkus (mapped) ke permukaan objek 3D. Penerapan tekstur umumnya dilakukan dengan:

- **glBindTexture** untuk mengikat tekstur.
- **glTexCoord2f** untuk menetapkan koordinat gambar ke tiap vertex.
- **glTexImage2D** untuk memuat data gambar ke memori GPU.

- **Langkah Percobaan**

1. Inisialisasi GLFW dan GLEW.
2. Buat window dengan GLFW.
3. Load gambar BMP menggunakan fungsi `loadBMP()` dari `imageloader.h/cpp`.
4. Konversi gambar menjadi tekstur OpenGL menggunakan `glTexImage2D`.
5. Aktifkan lighting dan depth testing.
6. Gambar kubus dan terapkan tekstur pada sisi depan dan belakang kubus.

- **Hasil dan analisis**

Program berhasil menampilkan **kubus 3D** yang berputar, di mana dua sisi (depan dan belakang) diberikan **tekstur dari gambar 852.bmp**. Sisi lainnya diwarnai menggunakan kombinasi warna RGB biasa.

Fitur penting lain yang berhasil diterapkan:

- **Pencahayaan (lighting)**: OpenGL menggunakan `GL_LIGHT0` untuk memberikan efek realistik terhadap arah dan warna cahaya.
- **Depth testing** (`glEnable(GL_DEPTH_TEST)`) mencegah objek tumpang tindih secara visual.
- **Rotasi objek** menggunakan `glRotatef()` agar terlihat dinamis.

2. **Proyeksi Bayangan (Shadow Projection) dalam OpenGL**

Main2.cpp

```
#include <math.h>
#include <stdio.h>
#include <GL/glew.h>
#include <GLFW/glfw3.h>
double rx = 0.0;
double ry = 0.0;
float l[] = { 0.0, 80.0, 0.0 };
float n[] = { 0.0, -40.0, 0.0 };
float e[] = { 0.0, -60.0, 0.0 };
void help() {
printf("Shadow projection example using a cone object\n");
printf("Controls:\n");
printf("W/S - Move light up/down\n");
printf("A/D - Move light left/right\n");
printf("Q/E - Move light forward/backward\n");
printf("ESC - Exit\n");
}
void draw() {

GLUquadric* quadric = gluNewQuadric();
gluQuadricDrawStyle(quadric, GLU_FILL);
gluCylinder(quadric, 20.0, 0.0, 50.0, 40, 50);
gluDeleteQuadric(quadric);
}
void glShadowProjection(float *l, float *e, float *n) {
float d, c;
float mat[16];
d = n[0]*l[0] + n[1]*l[1] + n[2]*l[2];
c = e[0]*n[0] + e[1]*n[1] + e[2]*n[2] - d;
mat[0] = l[0]*n[0]+c;
mat[4] = n[1]*l[0];
mat[8] = n[2]*l[0];
mat[12] = -l[0]*c-l[0]*d;
mat[1] = n[0]*l[1];
mat[5] = l[1]*n[1]+c;
```

```
mat[9] = n[2]*l[1];
mat[13] = -l[1]*c-l[1]*d;
mat[2] = n[0]*l[2];
mat[6] = n[1]*l[2];
mat[10] = l[2]*n[2]+c;
mat[14] = -l[2]*c-l[2]*d;
mat[3] = n[0];
mat[7] = n[1];
mat[11] = n[2];
mat[15] = -d;
glMultMatrixf(mat);
}
void render() {
glClearColor(0.0, 0.6, 0.9, 0.0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLightfv(GL_LIGHT0, GL_POSITION, l);
glDisable(GL_LIGHTING);
glColor3f(1.0, 1.0, 0.0);

glPointSize(5.0f);
glBegin(GL_POINTS);
glVertex3f(l[0], l[1], l[2]);
glEnd();
glColor3f(0.8, 0.8, 0.8);
glBegin(GL_QUADS);
glNormal3f(0.0, 1.0, 0.0);
glVertex3f(-1300.0, e[1]-0.1, 1300.0);
glVertex3f(1300.0, e[1]-0.1, 1300.0);
glVertex3f(1300.0, e[1]-0.1, -1300.0);
glVertex3f(-1300.0, e[1]-0.1, -1300.0);
glEnd();
glPushMatrix();
glRotatef(ry, 0, 1, 0);
glRotatef(rx, 1, 0, 0);
glEnable(GL_LIGHTING);
glColor3f(0.0, 0.0, 0.8);
draw();
glPopMatrix();
glPushMatrix();
glShadowProjection(l, e, n);
glRotatef(ry, 0, 1, 0);
glRotatef(rx, 1, 0, 0);
glDisable(GL_LIGHTING);
glColor3f(0.4, 0.4, 0.4);
draw();
```

```
glPopMatrix();
}
void keypress(GLFWwindow* window, int key, int scancode, int action, int
mods) {
if (action == GLFW_PRESS || action == GLFW_REPEAT) {
switch (key) {
case GLFW_KEY_ESCAPE:
glfwSetWindowShouldClose(window, GLFW_TRUE);
break;

case GLFW_KEY_S:
l[1] -= 5.0;
break;
case GLFW_KEY_W:
l[1] += 5.0;
break;
case GLFW_KEY_A:
l[0] -= 5.0;
break;
case GLFW_KEY_D:
l[0] += 5.0;
break;
case GLFW_KEY_Q:
l[2] -= 5.0;
break;
case GLFW_KEY_E:
l[2] += 5.0;
break;
case GLFW_KEY_H:
help();
break;
}
}
}
void idle(GLFWwindow* window) {
rx += 0.1;
ry += 0.1;
render();
glfwSwapBuffers(window);
}
void resize(GLFWwindow* window, int width, int height) {
glViewport(0, 0, width, height);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60.0f, (float)width/(float)height, 1.0f, 400.0f);
```

```
glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char* argv[]) {
if (!glfwInit()) {
fprintf(stderr, "Failed to initialize GLFW\n");
return -1;
}
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 2);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 1);
glfwWindowHint(GLFW_SAMPLES, 4);
GLFWwindow* window = glfwCreateWindow(400, 400, "Shadow Projection",
NULL, NULL);
if (!window) {
fprintf(stderr, "Failed to create GLFW window\n");
glfwTerminate();
return -1;
}
glfwMakeContextCurrent(window);
glfwSetKeyCallback(window, keypress);
glfwSetFramebufferSizeCallback(window, resize);
if (glewInit() != GLEW_OK) {
fprintf(stderr, "Failed to initialize GLEW\n");
return -1;
}
glEnable(GL_NORMALIZE);
glEnable(GL_LIGHTING);
glEnable(GL_COLOR_MATERIAL);
glEnable(GL_DEPTH_TEST);
glEnable(GL_LIGHT0);
float lightAmbient[] = {0.2f, 0.2f, 0.2f, 1.0f};
float lightDiffuse[] = {0.8f, 0.8f, 0.8f, 1.0f};
glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmbient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDiffuse);

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(60.0f, 1.0f, 1.0f, 400.0f);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.0, 0.0, -150.0);
help();
while (!glfwWindowShouldClose(window)) {
idle(window);
glfwPollEvents();
```

```
}
glfwDestroyWindow(window);
glfwTerminate();
return 0;
}
```

- Mempelajari bagaimana bayangan (shadow) dari objek 3D dapat dibuat menggunakan teknik **proyeksi bayangan manual** dalam OpenGL, serta memahami bagaimana posisi sumber cahaya mempengaruhi posisi bayangan.

- Langkah percobaan

1. **Inisialisasi GLFW dan GLEW**
   Program memulai dengan inisialisasi GLFW dan pembuatan jendela OpenGL.
2. **Pengaturan Kamera dan Pencahayaan**
   Kamera diposisikan dengan `gluPerspective` dan translasi tampilan.
   Pencahayaan diaktifkan dengan `GL_LIGHT0`.
3. **Membuat Objek 3D (Kerucut)**
   Objek utama adalah sebuah **kerucut (cone)** yang dibuat menggunakan `gluCylinder`.
4. **Menggambar Bidang Datar**
   Bidang datar tempat bayangan ditampilkan digambar menggunakan `GL_QUADS`.
5. **Menghitung dan Menggambar Bayangan**
   Fungsi `glShadowProjection(l, e, n)` menghitung matriks proyeksi bayangan berdasarkan:

- `l`: posisi cahaya
- `e`: titik pada bidang
- `n`: normal bidang

6. **Interaksi Keyboard**

- `W/S`: Naik/turun sumber cahaya
- `A/D`: Geser kiri/kanan
- `Q/E`: Geser maju/mundur

7. **Loop Render**
   Objek dan bayangannya digambar ulang setiap frame dengan rotasi kecil agar terlihat dinamis.

- **Hasil**
  Program berhasil menampilkan sebuah kerucut berwarna biru dengan bayangan abu-abu yang diproyeksikan ke bidang datar. Bayangan bergerak dinamis sesuai perubahan posisi sumber cahaya menggunakan tombol W, A, S, D, Q, E.

3. Interaksi Antar Objek pada Gambar (Simulasi Tendangan Bola)

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <GL/glew.h>
#include <GLFW/glfw3.h>
#include <GL/glu.h>
#define PI 3.14f

float angle = 0.0f, deltaAngle = 0.0f, ratio;
float x = -5.0f, y = 12.0f, z = 40.0f;
float lx = 0.0f, ly = 0.0f, lz = -1.0f;
int deltaMove = 0, w, h;
static int rotAngleX = 0, rotAngleY = 0, rotAngleZ = 0;
float posXKaki = 10, posXBola = -10, posYKaki = 6, posYBola = -5;
float rotKaki = 0.0f;
int kick = 0, roll = 0, touch = 0;

float jarak = 1;
GLUquadricObj *IDquadric;
const GLfloat light_ambient[] = { 0.5f, 0.5f, 0.5f, 0.0f };
const GLfloat light_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
const GLfloat light_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
const GLfloat light_position[] = { 0.0f, 20.0f, 10.0f, 1.0f };
const GLfloat mat_ambient[] = { 0.7f, 0.7f, 0.7f, 1.0f };
const GLfloat mat_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };
const GLfloat mat_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
const GLfloat high_shininess[] = { 100.0f };
void init() {
glEnable(GL_DEPTH_TEST);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
IDquadric = gluNewQuadric();
gluQuadricNormals(IDquadric, GLU_SMOOTH);
gluQuadricTexture(IDquadric, GL_TRUE);
}
void reshape(GLFWwindow* window, int w1, int h1) {
if (h1 == 0) h1 = 1;
w = w1;
h = h1;
ratio = 1.0f * w / h;
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glViewport(0, 0, w, h);
```

```
46.gluPerspective(45, ratio, 0.1, 1000);
47.glMatrixMode(GL_MODELVIEW);
48.glLoadIdentity();
49.gluLookAt(x, y, z, x + lx, y + ly, z + lz, 0.0f, 1.0f, 0.0f);
50.}
51.void orientMe(float ang) {
52.lx = sin(ang/10);
53.lz = -cos(ang/10);
54.glLoadIdentity();
55.
56.gluLookAt(x, y, z, x + lx, y + ly, z + lz, 0.0f, 1.0f, 0.0f);
57.}
58.void moveMeFlat(int i) {
59.x = x + i*(lx)*0.1f;
60.z = z + i*(lz)*0.1f;
61.glLoadIdentity();
62.gluLookAt(x, y, z, x + lx, y + ly, z + lz, 0.0f, 1.0f, 0.0f);
63.}
64.void keyCallback(GLFWwindow* window, int key, int scancode, int
   action,
65.int mods) {
66.if (action == GLFW_PRESS || action == GLFW_REPEAT) {
67.switch (key) {
68.case GLFW_KEY_W: rotAngleX += 2; break;
69.case GLFW_KEY_S: rotAngleX -= 2; break;
70.case GLFW_KEY_A: rotAngleY += 2; break;
71.case GLFW_KEY_D: rotAngleY -= 2; break;
72.case GLFW_KEY_Q: rotAngleZ += 2; break;
73.case GLFW_KEY_E: rotAngleZ -= 2; break;
74.case GLFW_KEY_O:
75.posXKaki -= 1;
76.if (posXBola < -2.9f) posXBola += 1;
77.break;
78.case GLFW_KEY_P:
79.posXKaki += 1;
80.posXBola -= 1;
81.break;
82.case GLFW_KEY_K: kick = 1; break;
83.case GLFW_KEY_SPACE:
84.rotAngleX = rotAngleY = rotAngleZ = 0;
85.posXKaki = 10; posXBola = -10; posYKaki = 6; posYBola =
86.-5;
87.rotKaki = kick = roll = 0;
88.break;
89.case GLFW_KEY_ESCAPE: glfwSetWindowShouldClose(window,
```

```
90.GLFW_TRUE); break;
91.}
92.}
93.
94.}
95.void specialKeyCallback(GLFWwindow* window, int key, int scancode,
   int
96.action, int mods) {
97.if (action == GLFW_PRESS) {
98.switch (key) {
99.case GLFW_KEY_UP: deltaMove = 1; break;
100.      case GLFW_KEY_DOWN: deltaMove = -1; break;
101.      case GLFW_KEY_LEFT: deltaAngle = -0.01f; break;
102.      case GLFW_KEY_RIGHT: deltaAngle = 0.01f; break;
103.      }
104.      } else if (action == GLFW_RELEASE) {
105.      switch (key) {
106.      case GLFW_KEY_UP: if (deltaMove > 0) deltaMove = 0; break;
107.      case GLFW_KEY_DOWN: if (deltaMove < 0) deltaMove = 0; break;
108.      case GLFW_KEY_LEFT: if (deltaAngle < 0.0f) deltaAngle =
109.      0.0f; break;
110.      case GLFW_KEY_RIGHT: if (deltaAngle > 0.0f) deltaAngle =
111.      0.0f; break;
112.      }
113.      }
114.      }
115.      void lighting() {
116.      glEnable(GL_DEPTH_TEST);
117.      glDepthFunc(GL_LESS);
118.      glEnable(GL_LIGHT0);
119.      glEnable(GL_NORMALIZE);
120.      glEnable(GL_COLOR_MATERIAL);
121.      glEnable(GL_LIGHTING);
122.      glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
123.      glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
124.      glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
125.      glLightfv(GL_LIGHT0, GL_POSITION, light_position);
126.      glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
127.      glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
128.      glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
129.      glMaterialfv(GL_FRONT, GL_SHININESS, high_shininess);
130.      }
131.
132.      void Grid() {
133.      double i;
```

```
134.      const float Z_MIN = -50, Z_MAX = 50;
135.      const float X_MIN = -50, X_MAX = 50;
136.      const float gap = 2;
137.      glColor3f(0.5f, 0.5f, 0.5f);
138.      glBegin(GL_LINES);
139.      for (i = Z_MIN; i < Z_MAX; i += gap) {
140.      glVertex3f(i, 0, Z_MIN);
141.      glVertex3f(i, 0, Z_MAX);
142.      }
143.      for (i = X_MIN; i < X_MAX; i += gap) {
144.      glVertex3f(X_MIN, 0, i);
145.      glVertex3f(X_MAX, 0, i);
146.      }
147.      glEnd();
148.      }
149.      void Balok(float panjang, float lebar, float tinggi) {
150.      glPushMatrix();
151.      float p = panjang/2;
152.      float l = lebar/2;
153.      float t = tinggi/2;
154.      // Front face
155.      glBegin(GL_QUADS);
156.      glVertex3f(-p, 0, l);
157.      glVertex3f(p, 0, l);
158.      glVertex3f(p, -t*2, l);
159.      glVertex3f(-p, -t*2, l);
160.      glEnd();
161.      // Back face
162.      glBegin(GL_QUADS);
163.      glVertex3f(-p, 0, -l);
164.      glVertex3f(p, 0, -l);
165.      glVertex3f(p, -t*2, -l);
166.
167.      glVertex3f(-p, -t*2, -l);
168.      glEnd();
169.      // Top face
170.      glBegin(GL_QUADS);
171.      glVertex3f(-p, 0, -l);
172.      glVertex3f(p, 0, -l);
173.      glVertex3f(p, 0, l);
174.      glVertex3f(-p, 0, l);
175.      glEnd();
176.      // Bottom face
177.      glBegin(GL_QUADS);
178.      glVertex3f(-p, -t*2, -l);
```

```
179.      glVertex3f(p, -t*2, -l);
180.      glVertex3f(p, -t*2, l);
181.      glVertex3f(-p, -t*2, l);
182.      glEnd();
183.      // Right face
184.      glBegin(GL_QUADS);
185.      glVertex3f(-p, -t*2, -l);
186.      glVertex3f(-p, -t*2, l);
187.      glVertex3f(-p, 0, l);
188.      glVertex3f(-p, 0, -l);
189.      glEnd();
190.      // Left face
191.      glBegin(GL_QUADS);
192.      glVertex3f(p, -t*2, -l);
193.      glVertex3f(p, -t*2, l);
194.      glVertex3f(p, 0, l);
195.      glVertex3f(p, 0, -l);
196.      glEnd();
197.      glPopMatrix();
198.      }
199.      void pergerakanKaki() {
200.
201.      if (kick == 1) {
202.      if (rotKaki <= 45) rotKaki += 0.03f;
203.      if (rotKaki > 44.9f) kick = 2;
204.      }
205.      if (posXBola > -2.9f) touch = 1;
206.      else if (posXBola < -12) touch = 0;
207.      if (kick == 2) {
208.      if (rotKaki >= -90) {
209.      rotKaki -= 0.2f;
210.      if (rotKaki < 1 && touch == 1) roll = 1;
211.      }
212.      if (rotKaki < -90) kick = 3;
213.      }
214.      if (kick == 3) {
215.      if (rotKaki <= 0) rotKaki += 0.05f;
216.      if (rotKaki > -1) kick = 0;
217.      }
218.      }
219.      void pergerakanBola() {
220.      if (roll == 1) {
221.      if (jarak > 0) {
222.      posXBola -= 0.03f;
223.      jarak -= 0.01f;
```

```
224.      }
225.      if (jarak < 0) {
226.      roll = 0;
227.      jarak = 1;
228.      }
229.      }
230.      }
231.      void Object() {
232.      glPushMatrix();
233.      glTranslatef(posXKaki, posYKaki, 0);
234.
235.      glPushMatrix();
236.      pergerakanKaki();
237.      glRotatef(rotKaki, 0, 0, 1);
238.      glColor3f(1, 1, 1);
239.      Balok(2, 3, 6);
240.      glPopMatrix();
241.      glPushMatrix();
242.      pergerakanBola();
243.      glColor3f(0.8f, 0.4f, 0.0f);
244.      glTranslatef(posXBola, posYBola, 0);
245.      gluSphere(IDquadric, 1.0, 20, 20);
246.      glPopMatrix();
247.      glPopMatrix();
248.      }
249.      void display(GLFWwindow* window) {
250.      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
251.      if (deltaMove) moveMeFlat(deltaMove);
252.      if (deltaAngle) {
253.      angle += deltaAngle;
254.      orientMe(angle);
255.      }
256.      glPushMatrix();
257.      glRotated(rotAngleX, 1, 0, 0);
258.      glRotated(rotAngleY, 0, 1, 0);
259.      glRotated(rotAngleZ, 0, 0, 1);
260.      Grid();
261.      Object();
262.      glPopMatrix();
263.      glfwSwapBuffers(window);
264.      }
265.
266.      int main(int argc, char **argv) {
267.      if (!glfwInit()) {
268.      fprintf(stderr, "Failed to initialize GLFW\n");
```

```
269.      return -1;
270.      }
271.      GLFWwindow* window = glfwCreateWindow(640, 480, "Tendangan",
   NULL,
272.      NULL);
273.      if (!window) {
274.      fprintf(stderr, "Failed to create GLFW window\n");
275.      glfwTerminate();
276.      return -1;
277.      }
278.      glfwMakeContextCurrent(window);
279.      glfwSetKeyCallback(window, keyCallback);
280.      glfwSetWindowSizeCallback(window, reshape);
281.      glewExperimental = GL_TRUE;
282.      if (glewInit() != GLEW_OK) {
283.      fprintf(stderr, "Failed to initialize GLEW\n");
284.      return -1;
285.      }
286.      lighting();
287.      init();
288.      reshape(window, 640, 480);
289.      while (!glfwWindowShouldClose(window)) {
290.      display(window);
291.      glfwPollEvents();
292.      if (glfwGetKey(window, GLFW_KEY_UP) == GLFW_PRESS) deltaMove =
293.      1;
294.      if (glfwGetKey(window, GLFW_KEY_DOWN) == GLFW_PRESS) deltaMove
   =
295.      -1;
296.      if (glfwGetKey(window, GLFW_KEY_LEFT) == GLFW_PRESS)
   deltaAngle
297.      = -0.01f;
298.      if (glfwGetKey(window, GLFW_KEY_RIGHT) == GLFW_PRESS)
   deltaAngle
299.
300.      = 0.01f;
301.      if (glfwGetKey(window, GLFW_KEY_UP) == GLFW_RELEASE &&
   deltaMove
302.      > 0) deltaMove = 0;
303.      if (glfwGetKey(window, GLFW_KEY_DOWN) == GLFW_RELEASE &&
304.      deltaMove < 0) deltaMove = 0;
305.      if (glfwGetKey(window, GLFW_KEY_LEFT) == GLFW_RELEASE &&
306.      deltaAngle < 0.0f) deltaAngle = 0.0f;
307.      if (glfwGetKey(window, GLFW_KEY_RIGHT) == GLFW_RELEASE &&
308.      deltaAngle > 0.0f) deltaAngle = 0.0f;
```

```
309.      }
310.      glfwDestroyWindow(window);
311.      glfwTerminate();
312.      return 0;
313.      }
```

- Mempelajari dan memahami bagaimana interaksi antar objek dapat diimplementasikan dalam pemrograman grafika komputer menggunakan **OpenGL** dan **GLFW**. Praktikum ini secara khusus mensimulasikan interaksi antara **kaki karakter** dan sebuah **bola** dalam bentuk animasi tendangan.

- **Langkah percobaan**

1. **Fungsi Utama:**

- main(): Inisialisasi window, callback input, dan loop utama untuk merender tampilan serta menangani logika gerakan.

2. **Rendering Objek:**

- Fungsi Object() menggambar dua objek utama:
  - **Kaki**: digambar menggunakan fungsi Balok().
  - **Bola**: digambar menggunakan gluSphere().

3. **Interaksi:**

- Kaki bergerak melalui kombinasi tombol O, P, dan K.
- Bola hanya akan bergerak jika terdapat **kontak** antara kaki dan bola saat aksi tendangan (kick == 2) dan bola berada pada posisi tertentu (posXBola > -2.9f).

4. **Logika Animasi:**

- pergerakanKaki() mengatur animasi rotasi kaki saat menendang.
- pergerakanBola() mengatur perpindahan bola jika terjadi interaksi.

5. **Input Keyboard:**

- Tombol panah menggerakkan kamera (navigasi).
- W, A, S, D, Q, E: rotasi tampilan.
- K: memicu tendangan.
- SPACE: reset posisi semua objek.
- **Hasil**

Setelah program dijalankan, tampilan menunjukkan grid sebagai lantai dan dua objek utama: **kaki dan bola**. Ketika tombol K ditekan, kaki melakukan gerakan menendang.

Jika kaki mengenai bola (terdeteksi oleh posisi X), maka bola bergulir menjauh, menandakan telah terjadi **interaksi antar objek**.

4. Interaksi Antar Objek pada Gambar (Memegang Objek)

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <GL/glew.h>
#include <GLFW/glfw3.h>
#include <GL/glu.h>
#define PI 3.14f
float angle = 0.0f, deltaAngle = 0.0f, ratio;
float x = 5.0f, y = 10.0f, z = 40.0f;
float lx = 0.0f, ly = 0.0f, lz = -1.0f;
int deltaMove = 0, w, h;
static int rotAngleX = 0, rotAngleY = 0, rotAngleZ = 0;
float posXBadan = 10, posXKotak = 0, posYBadan = 7, posYKotak = 6;
float rotTangan1 = 0.0f, rotTangan2 = 0.0f, rotTangan3 = 0.0f,
rotTangan4 = 0.0f;
int kick = 0, roll = 0, hit = 0, gerakTangan = 0, drop = 0, bring = 0,
grab = 0, tabrak = 0;
float jarak = 1;

GLUquadricObj *IDquadric;
const GLfloat light_ambient[] = { 0.5f, 0.5f, 0.5f, 0.0f };
const GLfloat light_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
const GLfloat light_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
const GLfloat light_position[] = { 0.0f, 20.0f, 10.0f, 1.0f };
const GLfloat mat_ambient[] = { 0.7f, 0.7f, 0.7f, 1.0f };
const GLfloat mat_diffuse[] = { 0.8f, 0.8f, 0.8f, 1.0f };
const GLfloat mat_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
const GLfloat high_shininess[] = { 100.0f };
void init(void) {
glEnable(GL_DEPTH_TEST);
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
IDquadric = gluNewQuadric();
gluQuadricNormals(IDquadric, GLU_SMOOTH);
gluQuadricTexture(IDquadric, GL_TRUE);
}
void reshape(GLFWwindow* window, int w1, int h1) {
if (h1 == 0) h1 = 1;
w = w1;
h = h1;
```

```
ratio = 1.0f * w / h;
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glViewport(0, 0, w, h);
gluPerspective(45, ratio, 0.1, 1000);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(x, y, z, x + lx, y + ly, z + lz, 0.0f, 1.0f, 0.0f);
}
void orientMe(float ang) {
lx = sin(ang/10);
lz = -cos(ang/10);
glLoadIdentity();
gluLookAt(x, y, z, x + lx, y + ly, z + lz, 0.0f, 1.0f, 0.0f);

}
void moveMeFlat(int i) {
x = x + i*(lx)*0.1f;
z = z + i*(lz)*0.1f;
glLoadIdentity();
gluLookAt(x, y, z, x + lx, y + ly, z + lz, 0.0f, 1.0f, 0.0f);
}
void keyCallback(GLFWwindow* window, int key, int scancode, int action,
int mods) {
if (action == GLFW_PRESS || action == GLFW_REPEAT) {
switch (key) {
case GLFW_KEY_W: rotAngleX += 2; break;
case GLFW_KEY_S: rotAngleX -= 2; break;
case GLFW_KEY_A: rotAngleY += 2; break;
case GLFW_KEY_D: rotAngleY -= 2; break;
case GLFW_KEY_Q: rotAngleZ += 2; break;
case GLFW_KEY_E: rotAngleZ -= 2; break;
case GLFW_KEY_O:
if (drop == 0) {
if (posXBadan > 4) {
posXBadan -= 1;
if (bring == 1) posXKotak -= 1;
}
} else {
if (posXBadan > posXKotak + 3) {
posXBadan -= 1;
}
}
break;
case GLFW_KEY_P:
```

```
posXBadan += 1;
if (bring == 1) posXKotak += 1;
break;
case GLFW_KEY_G: gerakTangan = 1; break;
case GLFW_KEY_T:
if (posXBadan >= 8) {
drop = 1;

gerakTangan = 3;
}
break;
case GLFW_KEY_SPACE:
rotAngleX = rotAngleY = rotAngleZ = 0;
posXBadan = 10; posXKotak = 0; posYBadan = 7; posYKotak
= 6;
rotTangan1 = rotTangan2 = rotTangan3 = rotTangan4 = kick
= roll = gerakTangan = drop = hit = bring = grab = 0;
break;
case GLFW_KEY_ESCAPE: glfwSetWindowShouldClose(window,
GLFW_TRUE); break;
}
}
}
void specialKeyCallback(GLFWwindow* window, int key, int scancode, int
action, int mods) {
if (action == GLFW_PRESS) {
switch (key) {
case GLFW_KEY_UP: deltaMove = 1; break;
case GLFW_KEY_DOWN: deltaMove = -1; break;
case GLFW_KEY_LEFT: deltaAngle = -0.01f; break;
case GLFW_KEY_RIGHT: deltaAngle = 0.01f; break;
}
} else if (action == GLFW_RELEASE) {
switch (key) {
case GLFW_KEY_UP: if (deltaMove > 0) deltaMove = 0; break;
case GLFW_KEY_DOWN: if (deltaMove < 0) deltaMove = 0; break;
case GLFW_KEY_LEFT: if (deltaAngle < 0.0f) deltaAngle =
0.0f; break;
case GLFW_KEY_RIGHT: if (deltaAngle > 0.0f) deltaAngle =
0.0f; break;
}
}
}
void lighting() {
glEnable(GL_DEPTH_TEST);
```

```
glDepthFunc(GL_LESS);
glEnable(GL_LIGHT0);
glEnable(GL_NORMALIZE);
glEnable(GL_COLOR_MATERIAL);
glEnable(GL_LIGHTING);
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, high_shininess);
}
void Grid() {
double i;
const float Z_MIN = -50, Z_MAX = 50;
const float X_MIN = -50, X_MAX = 50;
const float gap = 2;
glColor3f(0.5f, 0.5f, 0.5f);
glBegin(GL_LINES);
for (i = Z_MIN; i < Z_MAX; i += gap) {
glVertex3f(i, 0, Z_MIN);
glVertex3f(i, 0, Z_MAX);
}
for (i = X_MIN; i < X_MAX; i += gap) {
glVertex3f(X_MIN, 0, i);
glVertex3f(X_MAX, 0, i);
}
glEnd();
}
void Grid2() {
glColor3f(1.0f, 1.0f, 1.0f);
glBegin(GL_QUADS);
glVertex3f(-50, 0, 50);
glVertex3f(-50, 0, -50);

glVertex3f(50, 0, -50);
glVertex3f(50, 0, 50);
glEnd();
}
void Balok(float panjang, float lebar, float tinggi) {
glPushMatrix();
float p = panjang/2;
```

```
float l = lebar/2;
float t = tinggi/2;
glBegin(GL_QUADS);
glVertex3f(-p, 0, l);
glVertex3f(p, 0, l);
glVertex3f(p, -t*2, l);
glVertex3f(-p, -t*2, l);
glEnd();
glBegin(GL_QUADS);
glVertex3f(-p, 0, -l);
glVertex3f(p, 0, -l);
glVertex3f(p, -t*2, -l);
glVertex3f(-p, -t*2, -l);
glEnd();
glBegin(GL_QUADS);
glVertex3f(-p, 0, -l);
glVertex3f(p, 0, -l);
glVertex3f(p, 0, l);
glVertex3f(-p, 0, l);
glEnd();
glBegin(GL_QUADS);
glVertex3f(-p, -t*2, -l);
glVertex3f(p, -t*2, -l);
glVertex3f(p, -t*2, l);
glVertex3f(-p, -t*2, l);
glEnd();

glBegin(GL_QUADS);
glVertex3f(-p, -t*2, -l);
glVertex3f(-p, -t*2, l);
glVertex3f(-p, 0, l);
glVertex3f(-p, 0, -l);
glEnd();
glBegin(GL_QUADS);
glVertex3f(p, -t*2, -l);
glVertex3f(p, -t*2, l);
glVertex3f(p, 0, l);
glVertex3f(p, 0, -l);
glEnd();
glPopMatrix();
}
void perubahKotak() {
if (drop == 1 && grab == 1) {
if (posYKotak >= 3) {
posYKotak -= 0.01f;
```

```
}
if (posYKotak < 3) {
bring = 0;
hit = 0;
grab = 0;
}
}
}
void pengubahTangan() {
if (posXBadan != 4) {
hit = 0;
} else {
hit = 1;
}
if (hit == 1 && grab == 1) {
bring = 1;

}
if (gerakTangan == 1) {
if (rotTangan1 >= -90) {
rotTangan1 -= 0.1f;
}
if (rotTangan1 < -90) {
gerakTangan = 2;
}
}
if (gerakTangan == 2) {
if (rotTangan2 >= -90) {
rotTangan2 -= 0.1f;
}
if (rotTangan2 < -90 && hit == 1) {
grab = 1;
}
}
if (gerakTangan == 3) {
if (rotTangan2 <= 0) {
rotTangan2 += 0.1f;
}
if (rotTangan2 > 0) {
gerakTangan = 4;
}
}
if (gerakTangan == 4) {
if (rotTangan1 <= 0) {
rotTangan1 += 0.1f;
```

```
}
if (rotTangan1 > 0) {
gerakTangan = 0;
}
}
}

void Object() {
glPushMatrix();
glColor3f(0.1f, 0.1f, 0.2f);
glTranslatef(0, 3, 0);
Balok(5, 5, 3);
glPopMatrix();
glPushMatrix();
perubahKotak();
glColor3f(0.8f, 0.3f, 0.3f);
glTranslatef(posXKotak, posYKotak, 0);
Balok(3, 3, 3);
glPopMatrix();
glPushMatrix();
pengubahTangan();
glColor3f(0.3f, 0.3f, 0.8f);
glTranslatef(posXBadan, posYBadan, 0);
Balok(3, 3, 7);
glPushMatrix();
glColor3f(0.2f, 0.5f, 0.2f);
glTranslatef(0, -2, 2.5f);
glRotatef(rotTangan1, 1, 0, 0);
glRotatef(rotTangan2, 0, 0, 1);
Balok(2, 2, 4);
glPopMatrix();
glPushMatrix();
glColor3f(0.2f, 0.5f, 0.2f);
glTranslatef(0, -2, -2.5f);
glRotatef(-rotTangan1, 1, 0, 0);
glRotatef(rotTangan2, 0, 0, 1);
Balok(2, 2, 4);
glPopMatrix();
glPopMatrix();
}

void display(GLFWwindow* window) {
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
if (deltaMove) moveMeFlat(deltaMove);
if (deltaAngle) {
```

```
angle += deltaAngle;
orientMe(angle);
}
glPushMatrix();
glRotated(rotAngleX + 10, 1, 0, 0);
glRotated(rotAngleY, 0, 1, 0);
glRotated(rotAngleZ, 0, 0, 1);
Grid();
Grid2();
Object();
glPopMatrix();
glfwSwapBuffers(window);
}
int main(int argc, char **argv) {
if (!glfwInit()) {
fprintf(stderr, "Failed to initialize GLFW\n");
return -1;
}
GLFWwindow* window = glfwCreateWindow(640, 480, "Grab", NULL, NULL);
if (!window) {
fprintf(stderr, "Failed to create GLFW window\n");
glfwTerminate();
return -1;
}
glfwMakeContextCurrent(window);
glfwSetKeyCallback(window, keyCallback);
glfwSetWindowSizeCallback(window, reshape);

glewExperimental = GL_TRUE;
if (glewInit() != GLEW_OK) {
fprintf(stderr, "Failed to initialize GLEW\n");
return -1;
}
lighting();
init();
reshape(window, 640, 480);
while (!glfwWindowShouldClose(window)) {
display(window);
glfwPollEvents();
if (glfwGetKey(window, GLFW_KEY_UP) == GLFW_PRESS) deltaMove =
1;
if (glfwGetKey(window, GLFW_KEY_DOWN) == GLFW_PRESS) deltaMove =
-1;
if (glfwGetKey(window, GLFW_KEY_LEFT) == GLFW_PRESS) deltaAngle
= -0.01f;
```

```
if (glfwGetKey(window, GLFW_KEY_RIGHT) == GLFW_PRESS) deltaAngle
= 0.01f;
if (glfwGetKey(window, GLFW_KEY_UP) == GLFW_RELEASE && deltaMove
> 0) deltaMove = 0;
if (glfwGetKey(window, GLFW_KEY_DOWN) == GLFW_RELEASE &&
deltaMove < 0) deltaMove = 0;
if (glfwGetKey(window, GLFW_KEY_LEFT) == GLFW_RELEASE &&
deltaAngle < 0.0f) deltaAngle = 0.0f;
if (glfwGetKey(window, GLFW_KEY_RIGHT) == GLFW_RELEASE &&
deltaAngle > 0.0f) deltaAngle = 0.0f;
}
glfwDestroyWindow(window);
glfwTerminate();
return 0;
}
```

➢ OpenGL adalah API untuk merender grafik 2D dan 3D. GLFW digunakan untuk mengatur window dan input perangkat, sedangkan GLEW membantu mengakses fungsi ekstensi dari OpenGL. Dalam grafika komputer, penting untuk memahami:

- Transformasi Geometri (translasi, rotasi, skala)
- Kamera dan perspektif (gluLookAt)
- Interaksi input melalui keyboard
- Konsep pencahayaan (lighting)
- Interaksi antar objek (collision, grab/drop)

➢ Program ini mensimulasikan **interaksi antara objek berupa karakter dan sebuah kotak** (balok kecil). Karakter dapat bergerak ke kanan dan kiri, menggerakkan tangan, mengambil kotak, dan menjatuhkannya.

1. ***Fitur Utama:***

- **Pergerakan Karakter**: dengan tombol O dan P
- **Gerakan Tangan (Arm)**: dengan tombol G dan T
- **Pengambilan Kotak**: karakter harus dekat dengan kotak, tekan G untuk menjangkau dan T untuk mengangkat
- **Menjatuhkan Kotak**: tombol T ketika sudah membawa kotak
- **Reset Posisi**: tombol Space
- **Rotasi Scene**: tombol W, A, S, D, Q, E
- **Navigasi Kamera**: tombol panah (arrow keys)

- Hasil

Program berhasil:

- Menampilkan karakter dan kotak dalam lingkungan 3D.
- Memungkinkan karakter mendekati, mengambil, dan menjatuhkan kotak.
- Mendukung gerakan kamera dan rotasi objek.
- Implementasi interaktif tangan dalam animasi pengambilan objek.

Tampilan program akan menunjukkan interaksi real-time antara karakter dan objek kotak, lengkap dengan pencahayaan dan bayangan.