Laporan Project Akhir Grafika Komputer

Laporan ini dibuat untuk menyelesaikan tugas mata kuliah Grafika Komputer  
yang dibina oleh : **Muhammad Ilham Perdana, S.Tr.T., M.T**

Disusun Oleh:

Achmad Ardi Sukmadi 202310370311049

****

**UNIVERSITAS MUHAMMADIYAH MALANG**

**FAKULTAS TEKNIK**

**PROGRAM STUDI INFORMATIA**

**2025**

**Daftar isi**

[1. Soal 3](#_Toc203830481)

[1.1 Modul A Objek 2D 3](#_Toc203830482)

[1.1.1 Fungsi Penggambaran Objek 3](#_Toc203830483)

[1.1.2 Fungsi Warna dan ketebalan 3](#_Toc203830484)

[1.1.3 Tranformasi Geometri 3](#_Toc203830485)

[1.1.4 Windowing dan Clipping 3](#_Toc203830486)

[1.2 Modul B: Objek 3D 4](#_Toc203830487)

[1.2.1 Visualisasi Objek 3D 4](#_Toc203830488)

[1.2.2 Transformasi Objek 3D 4](#_Toc203830489)

[1.2.3 Shading dan Pencahayaan 4](#_Toc203830490)

[1.2.4 Kamera dan Perspektif 4](#_Toc203830491)

[2. Code 5](#_Toc203830492)

[2.1 Code Modul A 2D 5](#_Toc203830493)

[2.2 Code Modul B 3D 12](#_Toc203830494)

[3. Output 24](#_Toc203830495)

[3.1 Output Modul A 2D 24](#_Toc203830496)

[3.1.1 Output Terminal 24](#_Toc203830497)

[3.1.2 Output Visual 25](#_Toc203830498)

[3.2 Output Modul A 3D 29](#_Toc203830499)

[3.2.1 Output Terminal 29](#_Toc203830500)

[3.2.2 Output Visual 29](#_Toc203830501)

[4. Deskripsi Penjelasan Masing-Masing Output 34](#_Toc203830502)

[4.1 Output Modul A 2D 34](#_Toc203830503)

[4.2 Output Modul B 3D 35](#_Toc203830504)

[5. Referensi/Sumber Pengerjaan Project 37](#_Toc203830505)

# Soal

## Modul A Objek 2D

### Fungsi Penggambaran Objek

1. Gambar Objek Dasar:
2. Titik
3. Garis
4. Persegi
5. Ellipse
6. Koordinat input dilakukan dengan klik mouse pada canvas Opengl

### Fungsi Warna dan ketebalan

1. Pengguna dapat memilih:
2. Warna objek (melalui tombola tau shortcut keyboard)
3. Ketebalan garis (jika menggunakan GL\_LINES atau GL\_LINE\_LOOP).

### Tranformasi Geometri

1. Objek yang telah digambar dapat dikenai transformasi:
2. Translasi
3. Rotasi
4. Scaling
5. Transformasi dilakukan melalui keyboard, tombol menu, atau shortcut.

### Windowing dan Clipping

1. Pengguna dapat menentukan window aktif (misalnya dengan klik 2 titik sudut sebagai batas window).
2. Objek yang:
3. Masuk ke window: akan berubah warna (misalnya menjadi hijau).
4. Di luar window: akan dikenai clipping , hanya bagian dalam window yang ditampilkan (gunakan algoritma Coben-Sutherland atau Liang-Barsky untuk garis).
5. Window dapat digeser atau diuabah ukurannya.

## Modul B: Objek 3D

### Visualisasi Objek 3D

Tampilkan Minimal **1 objek 3D** berbentuk:

1. Kubus atau Piramida dan bisa yang lain juga
2. Dibaca dari file .obj atau Digambar manual (vertex, face)

### Transformasi Objek 3D

Translasi, Rotasi (dengan keyboard atau mouse drag)

### Shading dan Pencahayaan

1. Implementasi model pencahayaan sederhana (phong /gouraud) Menggunakan :

* Ambient Light
* Diffuse Light
* Specular Light

### Kamera dan Perspektif

1. Mengatur posisi kamera (gluLookAt)
2. Menggunakan proyeksi perspektif (gluPerspective) untuk 3D

# Code

## Code Modul A 2D

# Import library yang diperlukan  
import sys  
import os  
from math import sin, cos, radians  
  
try:  
 from OpenGL.GL import \*  
 from OpenGL.GLUT import \*  
 from OpenGL.GLU import \*  
except ImportError:  
 print("Error: PyOpenGL tidak terinstal.")  
 print("Silakan instal dengan perintah: pip install PyOpenGL PyOpenGL\_accelerate")  
 sys.exit(1)  
  
# =============================================================================  
# 1. PENGELOLAAN STATE DAN VARIABEL GLOBAL  
# =============================================================================  
  
# Dimensi window  
window\_width = 1280  
window\_height = 720  
  
# Variabel untuk transformasi objek  
rotation\_x = 0.0  
rotation\_y = 0.0  
translate\_x = 0.0  
translate\_y = 0.0  
translate\_z = 0.0 # Direset setiap load model  
scale\_factor = 1.0  
object\_color = [0.6, 0.7, 1.0]  
  
# Variabel untuk interaksi mouse  
mouse\_down = False  
last\_mouse\_x = 0  
last\_mouse\_y = 0  
  
# Struktur data untuk menyimpan model 3D yang dimuat  
model = {  
 "vertices": [],  
 "normals": [],  
 "faces": [],  
 "center": (0.0, 0.0, 0.0) # Pusat geometris model  
}  
  
  
# =============================================================================  
# 2. FUNGSI IMPORT, EXPORT, DAN MANIPULASI MODEL  
# =============================================================================  
  
def center\_model\_and\_reset\_transform():  
 *"""Menghitung pusat model, memindahkannya ke origin, dan mereset transformasi."""* global model, rotation\_x, rotation\_y, translate\_x, translate\_y, translate\_z, scale\_factor  
 if not model["vertices"]:  
 return  
  
 # Hitung bounding box  
 min\_x = min(v[0] for v in model["vertices"])  
 max\_x = max(v[0] for v in model["vertices"])  
 min\_y = min(v[1] for v in model["vertices"])  
 max\_y = max(v[1] for v in model["vertices"])  
 min\_z = min(v[2] for v in model["vertices"])  
 max\_z = max(v[2] for v in model["vertices"])  
  
 # Hitung pusat geometris  
 center\_x = (min\_x + max\_x) / 2.0  
 center\_y = (min\_y + max\_y) / 2.0  
 center\_z = (min\_z + max\_z) / 2.0  
 model["center"] = (center\_x, center\_y, center\_z)  
  
 # Pindahkan semua vertex sehingga pusatnya ada di (0,0,0)  
 new\_vertices = []  
 for v in model["vertices"]:  
 new\_vertices.append((v[0] - center\_x, v[1] - center\_y, v[2] - center\_z))  
 model["vertices"] = new\_vertices  
  
 # Hitung jarak terjauh dari origin untuk menentukan posisi kamera awal  
 max\_dist = max(max(abs(v[i]) for v in model["vertices"]) for i in range(3)) if model["vertices"] else 1  
  
 # Reset transformasi  
 rotation\_x, rotation\_y = 0.0, 0.0  
 translate\_x, translate\_y = 0.0, 0.0  
 # Atur translate\_z agar objek terlihat sepenuhnya  
 translate\_z = -max\_dist \* 2.5  
 scale\_factor = 1.0  
 print(f"Model dipusatkan di {model['center']}. Transformasi direset.")  
  
  
def load\_obj(filename):  
 *"""Memuat model 3D dari sebuah file .obj."""* global model  
 if not os.path.exists(filename):  
 print(f"Error: File '{filename}' tidak ditemukan.")  
 return  
  
 temp\_vertices, temp\_normals, temp\_faces = [], [], []  
 with open(filename, 'r') as f:  
 for line in f:  
 parts = line.strip().split()  
 if not parts: continue  
  
 if parts[0] == 'v':  
 temp\_vertices.append(tuple(map(float, parts[1:4])))  
 elif parts[0] == 'vn':  
 temp\_normals.append(tuple(map(float, parts[1:4])))  
 elif parts[0] == 'f':  
 face = []  
 for part in parts[1:]:  
 indices = part.split('/')  
 v\_idx = int(indices[0]) - 1  
 vn\_idx = int(indices[2]) - 1 if len(indices) > 2 and indices[2] else -1  
 face.append((v\_idx, vn\_idx))  
 temp\_faces.append(tuple(face))  
  
 model = {"vertices": temp\_vertices, "normals": temp\_normals, "faces": temp\_faces}  
 center\_model\_and\_reset\_transform() # Pusatkan model setelah dimuat  
 print(f"Model '{filename}' berhasil dimuat: {len(model['vertices'])} vertices, {len(model['faces'])} faces.")  
 glutPostRedisplay()  
  
  
def export\_obj(filename):  
 *"""Mengekspor model saat ini (dengan transformasi) ke file .obj."""* if not model["vertices"]:  
 print("Tidak ada model untuk diekspor.")  
 return  
  
 with open(filename, 'w') as f:  
 f.write("# Diekspor oleh Aplikasi Grafika 3D \n")  
 f.write(f"# Vertices: {len(model['vertices'])}\n")  
 f.write(f"# Normals: {len(model['normals'])}\n")  
 f.write(f"# Faces: {len(model['faces'])}\n\n")  
  
 # Tulis semua vertex asli (sebelum transformasi)  
 for v in model["vertices"]:  
 # Tambahkan kembali offset pusat sebelum menulis  
 orig\_v = (v[0] + model['center'][0], v[1] + model['center'][1], v[2] + model['center'][2])  
 f.write(f"v {orig\_v[0]:.6f} {orig\_v[1]:.6f} {orig\_v[2]:.6f}\n")  
 f.write("\n")  
  
 # Tulis semua normal asli  
 for n in model["normals"]:  
 f.write(f"vn {n[0]:.6f} {n[1]:.6f} {n[2]:.6f}\n")  
 f.write("\n")  
  
 # Tulis semua face dengan format v//vn  
 for face in model["faces"]:  
 face\_str = "f " + " ".join([f"{v\_idx + 1}//{vn\_idx + 1}" for v\_idx, vn\_idx in face])  
 f.write(face\_str + "\n")  
  
 print(f"Model berhasil diekspor ke '{filename}' dengan data normal.")  
  
  
def load\_default\_cube():  
 *"""Memuat data kubus default ke dalam struktur model."""* global model  
 model = {  
 "vertices": [  
 (1, -1, -1), (1, 1, -1), (-1, 1, -1), (-1, -1, -1),  
 (1, -1, 1), (1, 1, 1), (-1, -1, 1), (-1, 1, 1)  
 ],  
 "normals": [  
 (0, 0, -1), (0, 0, 1), (1, 0, 0), (-1, 0, 0), (0, 1, 0), (0, -1, 0)  
 ],  
 "faces": [  
 ((0, 0), (1, 0), (2, 0), (3, 0)), # Belakang  
 ((4, 1), (5, 1), (7, 1), (6, 1)), # Depan  
 ((0, 2), (4, 2), (6, 2), (3, 2)), # Kanan  
 ((1, 3), (5, 3), (7, 3), (2, 3)), # Kiri  
 ((1, 4), (5, 4), (4, 4), (0, 4)), # Atas  
 ((3, 5), (2, 5), (7, 5), (6, 5)) # Bawah  
 ]  
 }  
 center\_model\_and\_reset\_transform() # Pusatkan kubus juga  
  
  
# =============================================================================  
# 3. DOKUMENTASI DAN BANTUAN  
# =============================================================================  
  
def print\_instructions():  
 *"""Mencetak panduan penggunaan ke konsol."""* print("=" \* 60)  
 print(" Aplikasi Grafika 3D Interaktif - PyOpenGL v1.5")  
 print("=" \* 60)  
 print("--- FILE ---")  
 print(" [I] Import File .obj (Ketik nama file di konsol)")  
 print(" [O] Export File .obj (Ketik nama file di konsol)")  
 print("\n--- KONTROL OBJEK ---")  
 print(" Rotasi : Klik kiri dan seret mouse")  
 print(" Zoom : Scroll mouse wheel")  
 print(" Translasi : W, A, S, D")  
 print("\n--- UBAH WARNA ---")  
 print(" [1] Merah | [2] Hijau | [3] Biru | [4] Kuning | [5] Jingga | [6] Default")  
 print("\n--- KONTROL APLIKASI ---")  
 print(" Keluar : Tekan tombol ESC")  
 print("=" \* 60)  
  
  
# =============================================================================  
# 4. FUNGSI MENGGAMBAR OBJEK  
# =============================================================================  
  
def draw\_model():  
 *"""Menggambar model yang saat ini dimuat dengan shading yang benar."""* if not model["faces"]: return  
  
 for face in model["faces"]:  
 # Gambar sebagai triangles atau quads tergantung jumlah vertex  
 if len(face) == 3:  
 glBegin(GL\_TRIANGLES)  
 elif len(face) == 4:  
 glBegin(GL\_QUADS)  
 else:  
 glBegin(GL\_POLYGON)  
  
 for v\_idx, vn\_idx in face:  
 # Terapkan normal untuk SETIAP vertex untuk smooth shading  
 if model["normals"] and vn\_idx != -1:  
 glNormal3fv(model["normals"][vn\_idx])  
 glVertex3fv(model["vertices"][v\_idx])  
 glEnd()  
  
  
# =============================================================================  
# 5. FUNGSI CALLBACK UTAMA OPENGL/GLUT  
# =============================================================================  
  
def display():  
 *"""Fungsi display utama, dipanggil setiap kali layar perlu digambar ulang."""* glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)  
 glLoadIdentity()  
 gluLookAt(0, 0, 5, 0, 0, 0, 0, 1, 0)  
  
 light\_position = [2.0, 3.0, 4.0, 1.0]  
 glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position)  
  
 # Terapkan transformasi interaktif. Rotasi kini terjadi di sekitar (0,0,0)  
 # karena modelnya sudah dipusatkan.  
 glPushMatrix() # Simpan matriks saat ini  
 glTranslatef(translate\_x, translate\_y, translate\_z)  
 glRotatef(rotation\_x, 1, 0, 0)  
 glRotatef(rotation\_y, 0, 1, 0)  
 glScalef(scale\_factor, scale\_factor, scale\_factor)  
  
 glColor3fv(object\_color)  
 draw\_model()  
 glPopMatrix() # Kembalikan matriks  
  
 glutSwapBuffers()  
  
  
def reshape(w, h):  
 *"""Callback saat window diubah ukurannya."""* global window\_width, window\_height  
 window\_width, window\_height = w, h  
 if h == 0: h = 1  
 glViewport(0, 0, w, h)  
 glMatrixMode(GL\_PROJECTION)  
 glLoadIdentity()  
 gluPerspective(45.0, float(w) / float(h), 0.1, 500.0) # Perbesar zFar  
 glMatrixMode(GL\_MODELVIEW)  
 glLoadIdentity()  
  
  
def init():  
 *"""Inisialisasi state OpenGL."""* glClearColor(1.0, 1.0, 1.0, 1.0)  
 glEnable(GL\_DEPTH\_TEST)  
 glEnable(GL\_LIGHTING)  
 glEnable(GL\_LIGHT0)  
 glShadeModel(GL\_SMOOTH)  
  
 # PERBAIKAN: Aktifkan normalisasi otomatis untuk vektor normal.  
 # Ini memastikan pencahayaan tetap akurat bahkan setelah objek di-skala.  
 glEnable(GL\_NORMALIZE)  
  
 light\_ambient = [0.2, 0.2, 0.2, 1.0]  
 light\_diffuse = [1.0, 1.0, 1.0, 1.0]  
 light\_specular = [1.0, 1.0, 1.0, 1.0]  
 glLightfv(GL\_LIGHT0, GL\_AMBIENT, light\_ambient)  
 glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light\_diffuse)  
 glLightfv(GL\_LIGHT0, GL\_SPECULAR, light\_specular)  
  
 mat\_ambient = [0.7, 0.7, 0.7, 1.0]  
 mat\_diffuse = [0.8, 0.8, 0.8, 1.0]  
 mat\_specular = [1.0, 1.0, 1.0, 1.0]  
 mat\_shininess = [100.0]  
 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, mat\_shininess)  
  
 glEnable(GL\_COLOR\_MATERIAL)  
 glColorMaterial(GL\_FRONT, GL\_DIFFUSE)  
  
 load\_default\_cube()  
 print\_instructions()  
  
  
# =============================================================================  
# 6. FUNGSI CALLBACK INPUT (MOUSE DAN KEYBOARD)  
# =============================================================================  
  
def keyboard(key, x, y):  
 *"""Callback untuk input keyboard."""* global translate\_x, translate\_y, scale\_factor, object\_color  
  
 try:  
 key\_char = key.decode("utf-8").lower()  
 except UnicodeDecodeError:  
 return  
  
 step = 0.2  
 scale\_step = 1.1  
  
 if key == b'\x1b':  
 print("Keluar dari aplikasi.")  
 glutLeaveMainLoop()  
 elif key\_char == 'i':  
 filename = input(">>> Masukkan nama file .obj untuk diimpor: ")  
 load\_obj(filename)  
 elif key\_char == 'o':  
 filename = input(">>> Masukkan nama file .obj untuk diekspor: ")  
 export\_obj(filename)  
 elif key\_char == 'w':  
 translate\_y += step  
 elif key\_char == 's':  
 translate\_y -= step  
 elif key\_char == 'a':  
 translate\_x -= step  
 elif key\_char == 'd':  
 translate\_x += step  
 elif key\_char in ['=', '+']:  
 scale\_factor \*= scale\_step  
 elif key\_char == '-':  
 scale\_factor /= scale\_step  
 elif key\_char == '1':  
 object\_color = [1.0, 0.3, 0.3]; print("Warna: Merah")  
 elif key\_char == '2':  
 object\_color = [0.3, 1.0, 0.3]; print("Warna: Hijau")  
 elif key\_char == '3':  
 object\_color = [0.3, 0.3, 1.0]; print("Warna: Biru")  
 elif key\_char == '4':  
 object\_color = [1.0, 1.0, 0.3]; print("Warna: Kuning")  
 elif key\_char == '5':  
 object\_color = [1.0, 0.3, 1.0]; print("Warna: Jingga")  
 elif key\_char == '6':  
 object\_color = [0.6, 0.7, 1.0]; print("Warna: Biru Muda (Default)")  
 glutPostRedisplay()  
  
  
def mouse\_click(button, state, x, y):  
 *"""Callback untuk klik mouse (rotasi)."""* global mouse\_down, last\_mouse\_x, last\_mouse\_y  
 if button == GLUT\_LEFT\_BUTTON:  
 if state == GLUT\_DOWN:  
 mouse\_down = True  
 last\_mouse\_x, last\_mouse\_y = x, y  
 elif state == GLUT\_UP:  
 mouse\_down = False  
  
  
def mouse\_motion(x, y):  
 *"""Callback untuk gerakan mouse (rotasi)."""* global rotation\_x, rotation\_y, last\_mouse\_x, last\_mouse\_y  
 if mouse\_down:  
 dx, dy = x - last\_mouse\_x, y - last\_mouse\_y  
 rotation\_y += dx \* 0.5  
 rotation\_x += dy \* 0.5  
 last\_mouse\_x, last\_mouse\_y = x, y  
 glutPostRedisplay()  
  
  
def mouse\_wheel(wheel, direction, x, y):  
 *"""Callback untuk scroll mouse (zoom/skala)."""* global scale\_factor  
 scale\_step = 1.1  
 if direction > 0:  
 scale\_factor \*= scale\_step  
 elif direction < 0:  
 scale\_factor /= scale\_step  
 glutPostRedisplay()  
  
  
# =============================================================================  
# 7. FUNGSI MAIN  
# =============================================================================  
  
def main():  
 *"""Fungsi utama untuk menginisialisasi GLUT dan memulai loop."""* glutInit(sys.argv)  
 glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE | GLUT\_DEPTH)  
 glutInitWindowSize(window\_width, window\_height)  
 glutInitWindowPosition(100, 100)  
 glutCreateWindow(b"Aplikasi Grafika 3D Interaktif - OpenGL")  
  
 glutDisplayFunc(display)  
 glutReshapeFunc(reshape)  
 glutKeyboardFunc(keyboard)  
 glutMouseFunc(mouse\_click)  
 glutMotionFunc(mouse\_motion)  
 glutMouseWheelFunc(mouse\_wheel)  
  
 init()  
 glutMainLoop()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

## Code Modul B 3D

# Import library yang diperlukan  
import sys  
import copy  
from math import sin, cos, pi, sqrt, radians, degrees  
  
try:  
 from OpenGL.GL import \*  
 from OpenGL.GLUT import \*  
 from OpenGL.GLU import \*  
except ImportError:  
 print("Error: PyOpenGL tidak terinstal.")  
 print("Silakan instal dengan perintah: pip install PyOpenGL PyOpenGL\_accelerate")  
 sys.exit(1)  
  
# =============================================================================  
# 1. PENGELOLAAN STATE DAN VARIABEL GLOBAL  
# =============================================================================  
  
# Dimensi window  
window\_width = 1280  
window\_height = 720  
  
# List untuk menyimpan semua objek yang digambar  
objects = []  
selected\_indices = []  
clipboard = []  
  
# State aplikasi  
current\_mode = 'select'  
is\_drawing = False  
is\_dragging\_selection = False  
drag\_last\_pos = {'x': 0, 'y': 0}  
  
# Atribut objek baru  
current\_color = (0.0, 0.0, 0.0)  
current\_thickness = 1.0  
  
# Variabel sementara  
temp\_vertex = None  
ghost\_object = None  
selection\_box = None  
  
# Definisi clipping window  
clipping\_window = {  
 'xmin': 100, 'ymin': 100, 'xmax': 500, 'ymax': 400,  
 'active': False,  
 'color': (1.0, 0.0, 0.0),  
}  
  
# Konstanta Cohen-Sutherland  
C\_INSIDE, C\_LEFT, C\_RIGHT, C\_BOTTOM, C\_TOP = 0, 1, 2, 4, 8  
  
  
# =============================================================================  
# 2. DOKUMENTASI DAN BANTUAN  
# =============================================================================  
  
def print\_instructions():  
 *"""Mencetak panduan penggunaan ke konsol."""* print("=" \* 60)  
 print(" Aplikasi Grafika 2D Interaktif - PyOpenGL v1.7.1")  
 print("=" \* 60)  
 print("--- MODE ---")  
 print(" [P] Titik | [L] Garis | [R] Persegi | [E] Elips")  
 print(" [ESC] Kembali ke Mode Seleksi (Select)")  
 print("\n--- SELEKSI & TRANSFORMASI ---")  
 print(" Klik pada objek untuk memilih.")  
 print(" Klik & Seret di area kosong untuk memilih beberapa objek.")  
 print(" Klik & Tahan pada objek terpilih untuk menggesernya.")  
 print(" [Shift+Klik] atau [Shift+Seret] untuk menambah/mengurangi objek.")  
 print(" [Ctrl+A] untuk memilih semua objek.")  
 print(" Gunakan Panah untuk Translasi, [Q/A] untuk Rotasi, [W/S] untuk Skala.")  
 print("\n--- MANAJEMEN OBJEK ---")  
 print(" [Ctrl+C] : Copy objek terpilih.")  
 print(" [Ctrl+V] : Paste objek dari clipboard.")  
 print(" [DELETE] / [BACKSPACE] : Hapus objek yang dipilih.")  
 print(" [Shift+DELETE] : Hapus SEMUA objek (Clear All).")  
 print("\n--- WARNA & KETEBALAN ---")  
 print(" [1] Hitam | [2] Merah | [3] Hijau | [4] Biru")  
 print(" [+/-] Ubah Ketebalan Garis")  
 print("\n--- WINDOWING & CLIPPING ---")  
 print(" [C] Buat Window (Klik & Seret) | [D] Nonaktifkan Window")  
 print(" [G] Masuk mode Geser/Resize Window (Gunakan Panah / Shift+Panah)")  
 print("=" \* 60)  
  
  
# =============================================================================  
# 3. FUNGSI HELPER, MATEMATIKA, DAN ALGORITMA  
# =============================================================================  
  
def create\_object(obj\_type, vertices, color, thickness):  
 *"""Membuat dictionary objek baru dan menambahkannya ke list."""* global objects, selected\_indices  
 new\_obj = {  
 'type': obj\_type,  
 'vertices': vertices,  
 'color': color,  
 'thickness': thickness,  
 'transform': {'translate': [0, 0], 'rotate': 0.0, 'scale': [1.0, 1.0]}  
 }  
 objects.append(new\_obj)  
 selected\_indices = [len(objects) - 1]  
  
  
def copy\_selected\_objects():  
 *"""Menyalin objek terpilih ke clipboard."""* global clipboard  
 if not selected\_indices:  
 print("Tidak ada objek yang dipilih untuk di-copy.")  
 return  
 clipboard.clear()  
 for index in selected\_indices:  
 clipboard.append(copy.deepcopy(objects[index]))  
 print(f"{len(clipboard)} objek di-copy ke clipboard.")  
  
  
def paste\_objects():  
 *"""Menempelkan objek dari clipboard."""* global objects, selected\_indices  
 if not clipboard:  
 print("Clipboard kosong.")  
 return  
 new\_indices = []  
 for obj\_to\_paste in clipboard:  
 new\_obj = copy.deepcopy(obj\_to\_paste)  
 new\_obj['transform']['translate'][0] += 15  
 new\_obj['transform']['translate'][1] += 15  
 objects.append(new\_obj)  
 new\_indices.append(len(objects) - 1)  
 selected\_indices = new\_indices  
 print(f"{len(new\_indices)} objek di-paste.")  
 glutPostRedisplay()  
  
  
def delete\_selected\_objects():  
 *"""Menghapus semua objek yang sedang dipilih."""* global objects, selected\_indices  
 if not selected\_indices: return  
 print(f"Menghapus {len(selected\_indices)} objek terpilih...")  
 objects = [obj for i, obj in enumerate(objects) if i not in selected\_indices]  
 selected\_indices.clear()  
 glutPostRedisplay()  
  
  
def clear\_all():  
 *"""Menghapus semua objek dari canvas."""* global objects, selected\_indices  
 print("Menghapus semua objek...");  
 objects.clear();  
 selected\_indices.clear();  
 glutPostRedisplay()  
  
  
def select\_all():  
 *"""Memilih semua objek di canvas."""* global selected\_indices  
 selected\_indices = list(range(len(objects)))  
 print(f"Memilih semua ({len(selected\_indices)}) objek.");  
 glutPostRedisplay()  
  
  
def get\_object\_center(obj):  
 if not obj['vertices']: return (0, 0)  
 if obj['type'] in ['point', 'ellipse', 'freehand']: return obj['vertices'][0]  
 x\_coords = [v[0] for v in obj['vertices']];  
 y\_coords = [v[1] for v in obj['vertices']]  
 return (sum(x\_coords) / len(x\_coords), sum(y\_coords) / len(y\_coords))  
  
  
def get\_transformed\_vertex(vertex, obj):  
 center = get\_object\_center(obj);  
 tr = obj['transform']  
 vx, vy = vertex[0] - center[0], vertex[1] - center[1]  
 vx, vy = vx \* tr['scale'][0], vy \* tr['scale'][1]  
 angle\_rad = radians(tr['rotate']);  
 cos\_a, sin\_a = cos(angle\_rad), sin(angle\_rad)  
 rvx = vx \* cos\_a - vy \* sin\_a;  
 rvy = vx \* sin\_a + vy \* cos\_a  
 final\_x = rvx + center[0] + tr['translate'][0];  
 final\_y = rvy + center[1] + tr['translate'][1]  
 return (final\_x, final\_y)  
  
  
def get\_object\_aabb(obj):  
 if not obj['vertices']: return None  
 if obj['type'] == 'ellipse':  
 center\_x, center\_y = obj['vertices'][0]  
 rx = abs(obj['vertices'][1][0] - center\_x);  
 ry = abs(obj['vertices'][1][1] - center\_y)  
 verts\_to\_check = [(center\_x + rx, center\_y), (center\_x - rx, center\_y), (center\_x, center\_y + ry),  
 (center\_x, center\_y - ry)]  
 else:  
 verts\_to\_check = obj['vertices']  
 transformed\_verts = [get\_transformed\_vertex(v, obj) for v in verts\_to\_check]  
 min\_x = min(v[0] for v in transformed\_verts);  
 max\_x = max(v[0] for v in transformed\_verts)  
 min\_y = min(v[1] for v in transformed\_verts);  
 max\_y = max(v[1] for v in transformed\_verts)  
 return (min\_x, min\_y, max\_x, max\_y)  
  
  
def get\_inverse\_transformed\_point(x, y, obj):  
 center = get\_object\_center(obj);  
 tr = obj['transform']  
 px, py = x - tr['translate'][0], y - tr['translate'][1]  
 px, py = px - center[0], py - center[1]  
 angle\_rad = radians(-tr['rotate']);  
 cos\_a, sin\_a = cos(angle\_rad), sin(angle\_rad)  
 rpx = px \* cos\_a - py \* sin\_a;  
 rpy = px \* sin\_a + py \* cos\_a  
 px, py = rpx, rpy  
 sx = tr['scale'][0] if tr['scale'][0] != 0 else 1.0;  
 sy = tr['scale'][1] if tr['scale'][1] != 0 else 1.0  
 px, py = px / sx, py / sy  
 final\_px, final\_py = px + center[0], py + center[1]  
 return final\_px, final\_py  
  
  
def dist\_sq(p1, p2):  
 return (p1[0] - p2[0]) \*\* 2 + (p1[1] - p2[1]) \*\* 2  
  
  
def is\_point\_on\_object(x, y, obj):  
 ix, iy = get\_inverse\_transformed\_point(x, y, obj)  
 tolerance\_sq = (obj['thickness'] \* 3 + 3) \*\* 2  
 obj\_type, verts = obj['type'], obj['vertices']  
 if obj\_type == 'point':  
 return dist\_sq((ix, iy), verts[0]) < tolerance\_sq \* 2  
 elif obj\_type == 'line':  
 p, v, w = (ix, iy), verts[0], verts[1];  
 l2 = dist\_sq(v, w)  
 if l2 == 0: return dist\_sq(p, v) < tolerance\_sq  
 t = max(0, min(1, ((p[0] - v[0]) \* (w[0] - v[0]) + (p[1] - v[1]) \* (w[1] - v[1])) / l2))  
 proj = (v[0] + t \* (w[0] - v[0]), v[1] + t \* (w[1] - v[1]))  
 return dist\_sq(p, proj) < tolerance\_sq  
 elif obj\_type == 'rectangle':  
 x\_coords = sorted([verts[0][0], verts[1][0]]);  
 y\_coords = sorted([verts[0][1], verts[1][1]])  
 return (x\_coords[0] <= ix <= x\_coords[1] and y\_coords[0] <= iy <= y\_coords[1])  
 elif obj\_type == 'ellipse':  
 center = verts[0];  
 rx, ry = abs(verts[1][0] - center[0]), abs(verts[1][1] - center[1])  
 if rx == 0 or ry == 0: return False  
 val = ((ix - center[0]) \*\* 2 / rx \*\* 2) + ((iy - center[1]) \*\* 2 / ry \*\* 2)  
 return val <= 1.1  
 elif obj\_type == 'freehand':  
 for i in range(len(verts) - 1):  
 if is\_point\_on\_object(x, y,  
 {'type': 'line', 'vertices': [verts[i], verts[i + 1]], 'thickness': obj['thickness'],  
 'transform': obj['transform']}):  
 return True  
 return False  
  
  
def is\_object\_fully\_inside\_window(obj):  
 if not obj['vertices']: return False  
 aabb = get\_object\_aabb(obj)  
 if not aabb: return False  
 return (clipping\_window['xmin'] <= aabb[0] and aabb[2] <= clipping\_window['xmax'] and  
 clipping\_window['ymin'] <= aabb[1] and aabb[3] <= clipping\_window['ymax'])  
  
  
def compute\_outcode(x, y):  
 code = C\_INSIDE  
 if x < clipping\_window['xmin']:  
 code |= C\_LEFT  
 elif x > clipping\_window['xmax']:  
 code |= C\_RIGHT  
 if y < clipping\_window['ymin']:  
 code |= C\_BOTTOM  
 elif y > clipping\_window['ymax']:  
 code |= C\_TOP  
 return code  
  
  
def cohen\_sutherland\_clip(x1, y1, x2, y2):  
 outcode1, outcode2 = compute\_outcode(x1, y1), compute\_outcode(x2, y2)  
 accept = False  
 while True:  
 if not (outcode1 | outcode2):  
 accept = True; break  
 elif (outcode1 & outcode2):  
 break  
 else:  
 x, y = 0, 0  
 outcode\_out = outcode1 if outcode1 else outcode2  
 if outcode\_out & C\_TOP:  
 x = x1 + (x2 - x1) \* (clipping\_window['ymax'] - y1) / (y2 - y1); y = clipping\_window['ymax']  
 elif outcode\_out & C\_BOTTOM:  
 x = x1 + (x2 - x1) \* (clipping\_window['ymin'] - y1) / (y2 - y1); y = clipping\_window['ymin']  
 elif outcode\_out & C\_RIGHT:  
 y = y1 + (y2 - y1) \* (clipping\_window['xmax'] - x1) / (x2 - x1); x = clipping\_window['xmax']  
 elif outcode\_out & C\_LEFT:  
 y = y1 + (y2 - y1) \* (clipping\_window['xmin'] - x1) / (x2 - x1); x = clipping\_window['xmin']  
 if outcode\_out == outcode1:  
 x1, y1 = x, y; outcode1 = compute\_outcode(x1, y1)  
 else:  
 x2, y2 = x, y; outcode2 = compute\_outcode(x2, y2)  
 return (True, x1, y1, x2, y2) if accept else (False, 0, 0, 0, 0)  
  
  
# =============================================================================  
# 4. FUNGSI MENGGAMBAR OBJEK (HANYA VISUAL)  
# =============================================================================  
  
def draw\_point(vertices, color, thickness):  
 glPointSize(thickness \* 5);  
 glColor3fv(color);  
 glBegin(GL\_POINTS);  
 glVertex2fv(vertices[0]);  
 glEnd()  
  
  
def draw\_line(vertices, color, thickness, clip=False):  
 x1, y1 = vertices[0];  
 x2, y2 = vertices[1]  
 if clip and clipping\_window['active']:  
 visible, nx1, ny1, nx2, ny2 = cohen\_sutherland\_clip(x1, y1, x2, y2)  
 if not visible: return  
 x1, y1, x2, y2 = nx1, ny1, nx2, ny2  
 glLineWidth(thickness);  
 glColor3fv(color);  
 glBegin(GL\_LINES);  
 glVertex2f(x1, y1);  
 glVertex2f(x2, y2);  
 glEnd()  
  
  
def draw\_rectangle(vertices, color, thickness, clip=False):  
 x1, y1 = vertices[0];  
 x2, y2 = vertices[1]  
 lines = [((x1, y1), (x2, y1)), ((x2, y1), (x2, y2)), ((x2, y2), (x1, y2)), ((x1, y2), (x1, y1))]  
 for line in lines: draw\_line(line, color, thickness, clip)  
  
  
def draw\_ellipse(vertices, color, thickness, clip=False):  
 center\_x, center\_y = vertices[0];  
 rx = abs(vertices[1][0] - center\_x);  
 ry = abs(vertices[1][1] - center\_y)  
 num\_segments = 100  
 glLineWidth(thickness);  
 glColor3fv(color);  
 glBegin(GL\_LINE\_LOOP)  
 for i in range(num\_segments):  
 theta = 2.0 \* pi \* i / num\_segments  
 x = rx \* cos(theta) + center\_x;  
 y = ry \* sin(theta) + center\_y  
 if clip and clipping\_window['active'] and not (  
 clipping\_window['xmin'] <= x <= clipping\_window['xmax'] and clipping\_window['ymin'] <= y <=  
 clipping\_window['ymax']):  
 glEnd();  
 glBegin(GL\_LINE\_LOOP);  
 continue  
 glVertex2f(x, y)  
 glEnd()  
  
  
def draw\_freehand(vertices, color, thickness, clip=False):  
 glLineWidth(thickness);  
 glColor3fv(color);  
 glBegin(GL\_LINE\_STRIP)  
 for v in vertices:  
 if clip and clipping\_window['active'] and not (  
 clipping\_window['xmin'] <= v[0] <= clipping\_window['xmax'] and clipping\_window['ymin'] <= v[1] <=  
 clipping\_window['ymax']):  
 glEnd();  
 glBegin(GL\_LINE\_STRIP);  
 continue  
 glVertex2fv(v)  
 glEnd()  
  
  
def draw\_clipping\_window():  
 if clipping\_window['active']:  
 glEnable(GL\_LINE\_STIPPLE);  
 glLineStipple(4, 0xAAAA);  
 glColor3fv(clipping\_window['color']);  
 glLineWidth(2.0)  
 glBegin(GL\_LINE\_LOOP)  
 glVertex2f(clipping\_window['xmin'], clipping\_window['ymin']);  
 glVertex2f(clipping\_window['xmax'], clipping\_window['ymin'])  
 glVertex2f(clipping\_window['xmax'], clipping\_window['ymax']);  
 glVertex2f(clipping\_window['xmin'], clipping\_window['ymax'])  
 glEnd()  
 glDisable(GL\_LINE\_STIPPLE)  
  
  
def draw\_selection\_box():  
 if selection\_box:  
 x1, y1, x2, y2 = selection\_box  
 glColor3f(0.3, 0.5, 0.9);  
 glLineWidth(1.0)  
 glBegin(GL\_LINE\_LOOP)  
 glVertex2f(x1, y1);  
 glVertex2f(x2, y1);  
 glVertex2f(x2, y2);  
 glVertex2f(x1, y2)  
 glEnd()  
 glEnable(GL\_BLEND);  
 glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA)  
 glColor4f(0.3, 0.5, 0.9, 0.2)  
 glBegin(GL\_QUADS)  
 glVertex2f(x1, y1);  
 glVertex2f(x2, y1);  
 glVertex2f(x2, y2);  
 glVertex2f(x1, y2)  
 glEnd()  
 glDisable(GL\_BLEND)  
  
  
# =============================================================================  
# 5. FUNGSI CALLBACK UTAMA OPENGL/GLUT  
# =============================================================================  
  
def display():  
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  
 glLoadIdentity()  
 for i, obj in enumerate(objects):  
 display\_color = obj['color']  
 if i in selected\_indices: display\_color = (0.9, 0.5, 0.0)  
 if clipping\_window['active'] and is\_object\_fully\_inside\_window(obj): display\_color = (0.1, 0.8, 0.2)  
 glPushMatrix()  
 center = get\_object\_center(obj)  
 glTranslatef(obj['transform']['translate'][0], obj['transform']['translate'][1], 0)  
 glTranslatef(center[0], center[1], 0)  
 glRotatef(obj['transform']['rotate'], 0, 0, 1)  
 glScalef(obj['transform']['scale'][0], obj['transform']['scale'][1], 1)  
 glTranslatef(-center[0], -center[1], 0)  
 obj\_type = obj['type']  
 if obj\_type == 'point':  
 draw\_point(obj['vertices'], display\_color, obj['thickness'])  
 elif obj\_type == 'line':  
 draw\_line(obj['vertices'], display\_color, obj['thickness'], clip=True)  
 elif obj\_type == 'rectangle':  
 draw\_rectangle(obj['vertices'], display\_color, obj['thickness'], clip=True)  
 elif obj\_type == 'ellipse':  
 draw\_ellipse(obj['vertices'], display\_color, obj['thickness'], clip=True)  
 elif obj\_type == 'freehand':  
 draw\_freehand(obj['vertices'], display\_color, obj['thickness'], clip=True)  
 glPopMatrix()  
  
 if is\_drawing and ghost\_object:  
 obj = ghost\_object;  
 color = (0.5, 0.5, 0.5)  
 if obj['type'] == 'draw\_line':  
 draw\_line(obj['vertices'], color, obj['thickness'])  
 elif obj['type'] == 'draw\_rectangle':  
 draw\_rectangle(obj['vertices'], color, obj['thickness'])  
 elif obj['type'] == 'draw\_ellipse':  
 draw\_ellipse(obj['vertices'], color, obj['thickness'])  
 elif obj['type'] == 'define\_window':  
 draw\_rectangle(obj['vertices'], clipping\_window['color'], 1.5)  
  
 draw\_clipping\_window()  
 draw\_selection\_box()  
 glutSwapBuffers()  
  
  
def reshape(w, h):  
 global window\_width, window\_height  
 window\_width, window\_height = w, h  
 glViewport(0, 0, w, h if h > 0 else 1)  
 glMatrixMode(GL\_PROJECTION);  
 glLoadIdentity();  
 gluOrtho2D(0.0, w, 0.0, h);  
 glMatrixMode(GL\_MODELVIEW);  
 glLoadIdentity()  
  
  
def init():  
 glClearColor(1.0, 1.0, 1.0, 1.0);  
 glEnable(GL\_BLEND);  
 glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA)  
 glEnable(GL\_LINE\_SMOOTH);  
 print\_instructions()  
  
  
# =============================================================================  
# 6. FUNGSI CALLBACK INPUT (MOUSE DAN KEYBOARD)  
# =============================================================================  
  
def mouse\_click(button, state, x, y):  
 global current\_mode, is\_drawing, temp\_vertex, ghost\_object, selected\_indices, selection\_box  
 global is\_dragging\_selection, drag\_last\_pos  
 y = window\_height - y  
  
 if button == GLUT\_LEFT\_BUTTON:  
 if state == GLUT\_DOWN:  
 mods = glutGetModifiers()  
 if current\_mode == 'select':  
 clicked\_on\_object = False  
 for i in selected\_indices:  
 if is\_point\_on\_object(x, y, objects[i]):  
 is\_dragging\_selection = True  
 drag\_last\_pos = {'x': x, 'y': y}  
 clicked\_on\_object = True  
 break  
 if not is\_dragging\_selection:  
 for i in range(len(objects) - 1, -1, -1):  
 if is\_point\_on\_object(x, y, objects[i]):  
 if mods == GLUT\_ACTIVE\_SHIFT:  
 if i in selected\_indices:  
 selected\_indices.remove(i)  
 else:  
 selected\_indices.append(i)  
 else:  
 selected\_indices = [i]  
 clicked\_on\_object = True  
 break  
 if not clicked\_on\_object:  
 is\_drawing = True  
 selection\_box = (x, y, x, y)  
 if mods != GLUT\_ACTIVE\_SHIFT:  
 selected\_indices.clear()  
 else:  
 is\_drawing = True;  
 temp\_vertex = (x, y)  
 if current\_mode == 'draw\_point':  
 create\_object('point', [(x, y)], current\_color, current\_thickness); is\_drawing = False  
 elif current\_mode == 'draw\_freehand':  
 create\_object('freehand', [temp\_vertex], current\_color, current\_thickness)  
 elif current\_mode in ['draw\_line', 'draw\_rectangle', 'draw\_ellipse', 'define\_window']:  
 ghost\_object = {'type': current\_mode, 'vertices': [temp\_vertex, temp\_vertex],  
 'color': current\_color, 'thickness': current\_thickness}  
 elif state == GLUT\_UP:  
 is\_dragging\_selection = False  
 if selection\_box:  
 x1, y1, x2, y2 = selection\_box  
 sel\_xmin, sel\_xmax = min(x1, x2), max(x1, x2)  
 sel\_ymin, sel\_ymax = min(y1, y2), max(y1, y2)  
 newly\_selected = set(selected\_indices)  
 for i, obj in enumerate(objects):  
 aabb = get\_object\_aabb(obj)  
 if aabb and not (  
 sel\_xmax < aabb[0] or sel\_xmin > aabb[2] or sel\_ymax < aabb[1] or sel\_ymin > aabb[3]):  
 newly\_selected.add(i)  
 selected\_indices = list(newly\_selected)  
 print(f"{len(selected\_indices)} objek terpilih.")  
 if is\_drawing and ghost\_object:  
 if current\_mode in ['draw\_line', 'draw\_rectangle', 'draw\_ellipse']:  
 create\_object(ghost\_object['type'].replace('draw\_', ''), ghost\_object['vertices'], current\_color,  
 current\_thickness)  
 elif current\_mode == 'define\_window':  
 vx = sorted([ghost\_object['vertices'][0][0], ghost\_object['vertices'][1][0]])  
 vy = sorted([ghost\_object['vertices'][0][1], ghost\_object['vertices'][1][1]])  
 clipping\_window.update({'xmin': vx[0], 'ymin': vy[0], 'xmax': vx[1], 'ymax': vy[1], 'active': True})  
 print("Clipping window didefinisikan.");  
 current\_mode = 'select'  
 is\_drawing = False;  
 ghost\_object = None;  
 temp\_vertex = None;  
 selection\_box = None  
 glutPostRedisplay()  
  
  
def mouse\_motion(x, y):  
 global selection\_box, drag\_last\_pos  
 y = window\_height - y  
 if is\_dragging\_selection:  
 dx = x - drag\_last\_pos['x'];  
 dy = y - drag\_last\_pos['y']  
 for index in selected\_indices:  
 objects[index]['transform']['translate'][0] += dx  
 objects[index]['transform']['translate'][1] += dy  
 drag\_last\_pos = {'x': x, 'y': y}  
 glutPostRedisplay()  
 return  
 if not is\_drawing: return  
 if selection\_box:  
 x1, y1, \_, \_ = selection\_box  
 = (x1, y1, x, y)  
 elif current\_mode == 'draw\_freehand':  
 if objects and objects[-1]['type'] == 'freehand': objects[-1]['vertices'].append((x, y))  
 elif ghost\_object and temp\_vertex:  
 ghost\_object['vertices'][1] = (x, y)  
 glutPostRedisplay()  
  
  
def keyboard(key, x, y):  
 global current\_mode, current\_color, current\_thickness  
 mods = glutGetModifiers()  
  
 # --- PERBAIKAN: Cek byte code untuk shortcut Ctrl ---  
 # Kode ini mendeteksi karakter kontrol ASCII yang dikirim saat Ctrl+key ditekan.  
 if key == b'\x01': # Ctrl+A  
 select\_all();  
 return  
 if key == b'\x03': # Ctrl+C  
 copy\_selected\_objects();  
 return  
 if key == b'\x16': # Ctrl+V  
 paste\_objects();  
 return  
  
 if key == b'\x08' or key == b'\x7f': # Backspace atau Delete  
 if mods == GLUT\_ACTIVE\_SHIFT:  
 clear\_all()  
 else:  
 delete\_selected\_objects()  
 return  
  
 # Decode ke char setelah cek control keys  
 try:  
 key\_char = key.decode("utf-8").lower()  
 except UnicodeDecodeError:  
 return # Abaikan tombol yang tidak bisa di-decode  
  
 modes = {'p': 'draw\_point', 'l': 'draw\_line', 'r': 'draw\_rectangle', 'e': 'draw\_ellipse', 'f': 'draw\_freehand',  
 'g': 'move\_window'}  
 # 'c' dipisah untuk menghindari konflik dengan Ctrl+C  
 if key\_char == 'c':  
 current\_mode = 'define\_window'; print(f"Mode: define\_window")  
 elif key\_char in modes:  
 current\_mode = modes[key\_char]; print(f"Mode: {current\_mode}")  
 elif key == b'\x1b':  
 current\_mode = 'select'; print("Mode: Select")  
 elif key\_char == 'd':  
 clipping\_window['active'] = False; print("Clipping window dinonaktifkan.")  
 elif key\_char == '1':  
 current\_color = (0.0, 0.0, 0.0); print("Warna: Hitam")  
 elif key\_char == '2':  
 current\_color = (1.0, 0.0, 0.0); print("Warna: Merah")  
 elif key\_char == '3':  
 current\_color = (0.0, 1.0, 0.0); print("Warna: Hijau")  
 elif key\_char == '4':  
 current\_color = (0.0, 0.0, 1.0); print("Warna: Biru")  
 elif key\_char in ['+', '=']:  
 current\_thickness += 0.5; print(f"Ketebalan: {current\_thickness}")  
 elif key\_char == '-':  
 current\_thickness = max(1.0, current\_thickness - 0.5); print(f"Ketebalan: {current\_thickness}")  
  
 if selected\_indices:  
 for index in selected\_indices:  
 obj = objects[index]  
 if key\_char == 'q':  
 obj['transform']['rotate'] += 5.0  
 elif key\_char == 'a':  
 obj['transform']['rotate'] -= 5.0  
 elif key\_char == 'w':  
 obj['transform']['scale'][0] \*= 1.1; obj['transform']['scale'][1] \*= 1.1  
 elif key\_char == 's':  
 obj['transform']['scale'][0] \*= 0.9; obj['transform']['scale'][1] \*= 0.9  
 glutPostRedisplay()  
  
  
def special\_keys(key, x, y):  
 step = 5.0;  
 mods = glutGetModifiers()  
 if current\_mode == 'select' and selected\_indices:  
 for index in selected\_indices:  
 transform = objects[index]['transform']['translate']  
 if key == GLUT\_KEY\_UP:  
 transform[1] += step  
 elif key == GLUT\_KEY\_DOWN:  
 transform[1] -= step  
 elif key == GLUT\_KEY\_LEFT:  
 transform[0] -= step  
 elif key == GLUT\_KEY\_RIGHT:  
 transform[0] += step  
 elif current\_mode == 'move\_window' and clipping\_window['active']:  
 if mods == GLUT\_ACTIVE\_SHIFT:  
 if key == GLUT\_KEY\_UP:  
 clipping\_window['ymax'] += step  
 elif key == GLUT\_KEY\_DOWN:  
 clipping\_window['ymax'] -= step  
 elif key == GLUT\_KEY\_LEFT:  
 clipping\_window['xmin'] -= step  
 elif key == GLUT\_KEY\_RIGHT:  
 clipping\_window['xmax'] += step  
 else:  
 if key == GLUT\_KEY\_UP:  
 clipping\_window['ymin'] += step; clipping\_window['ymax'] += step  
 elif key == GLUT\_KEY\_DOWN:  
 clipping\_window['ymin'] -= step; clipping\_window['ymax'] -= step  
 elif key == GLUT\_KEY\_LEFT:  
 clipping\_window['xmin'] -= step; clipping\_window['xmax'] -= step  
 elif key == GLUT\_KEY\_RIGHT:  
 clipping\_window['xmin'] += step; clipping\_window['xmax'] += step  
 glutPostRedisplay()  
  
  
# =============================================================================  
# 7. FUNGSI MAIN  
# =============================================================================  
  
def main():  
 glutInit(sys.argv)  
 glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE | GLUT\_DEPTH)  
 glutInitWindowSize(window\_width, window\_height)  
 glutInitWindowPosition(100, 100)  
 glutCreateWindow(b"Aplikasi Grafika 2D Interaktif - OpenGL v1.7.1")  
 glutDisplayFunc(display);  
 glutReshapeFunc(reshape);  
 glutKeyboardFunc(keyboard)  
 glutSpecialFunc(special\_keys);  
 glutMouseFunc(mouse\_click);  
 glutMotionFunc(mouse\_motion)  
 init()  
 glutMainLoop()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

# Output

## Output Modul A 2D

### Output Terminal

A screenshot of a computer program

Description automatically generated

### Output Visual

A screenshot of a computer

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA black and orange lines

Description automatically generatedA screenshot of a computer screen

Description automatically generatedA group of black circles

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

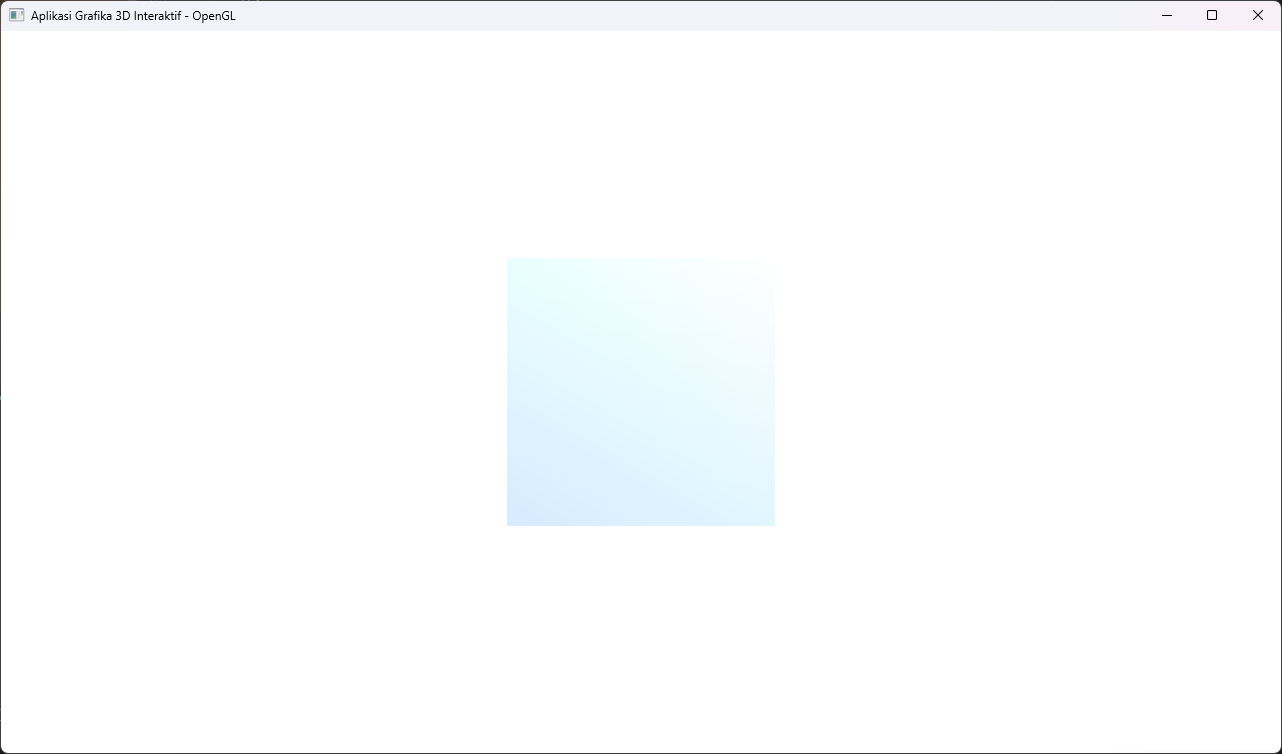
## Output Modul A 3D

### Output Terminal

A screenshot of a computer program

Description automatically generated

### Output Visual



A grey cube with a white background

Description automatically generated

A grey cube with a white background

Description automatically generated

A grey cube on a white background

Description automatically generated

A cube with a shadow

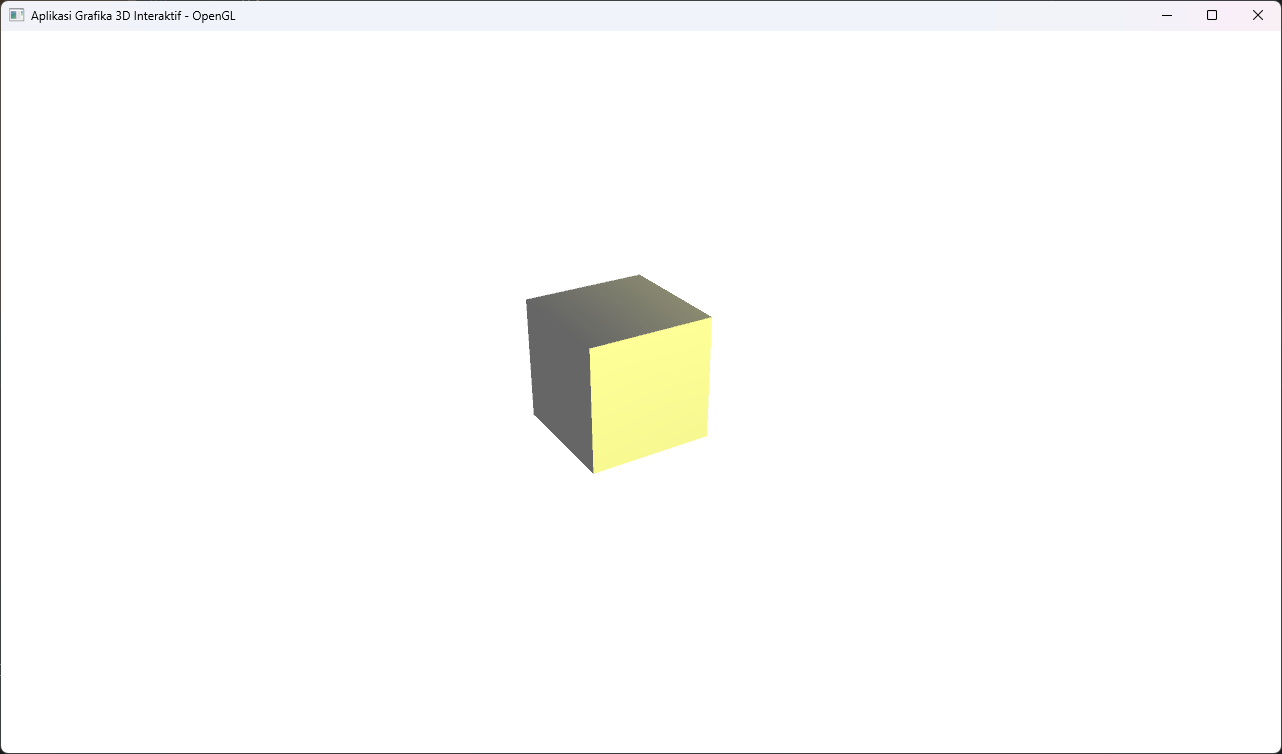
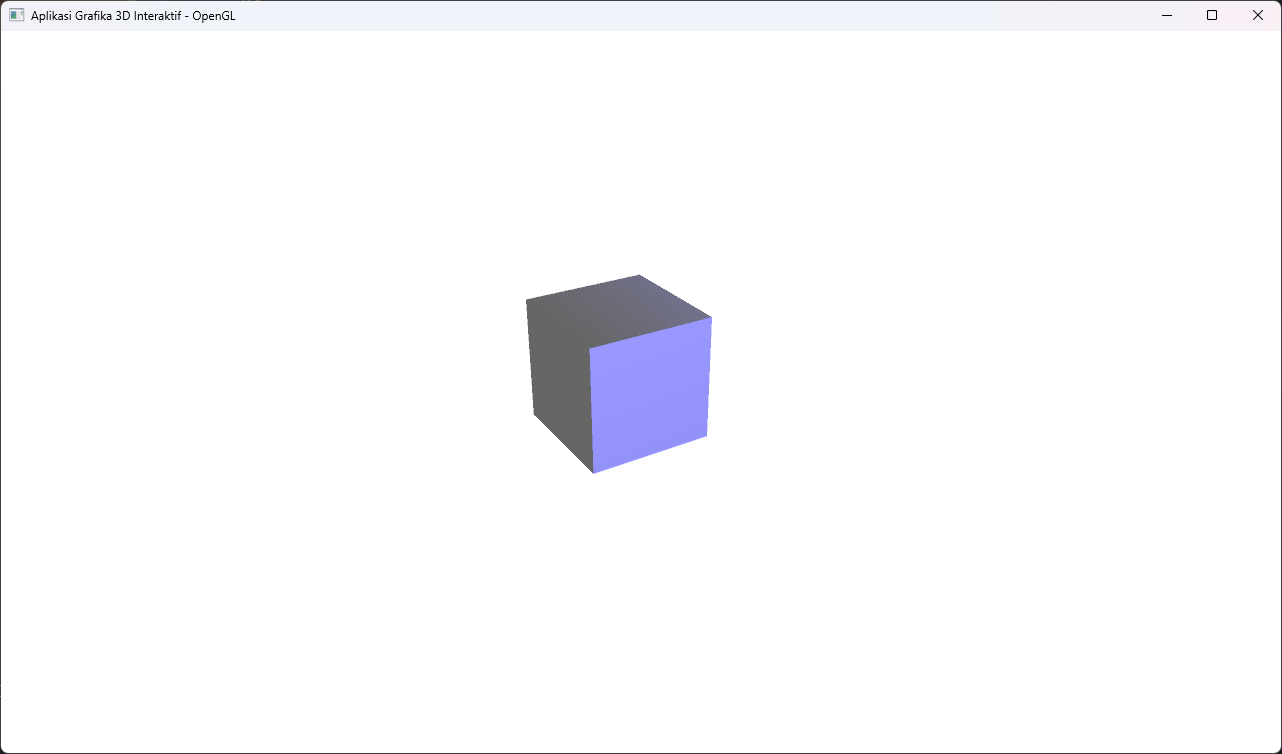
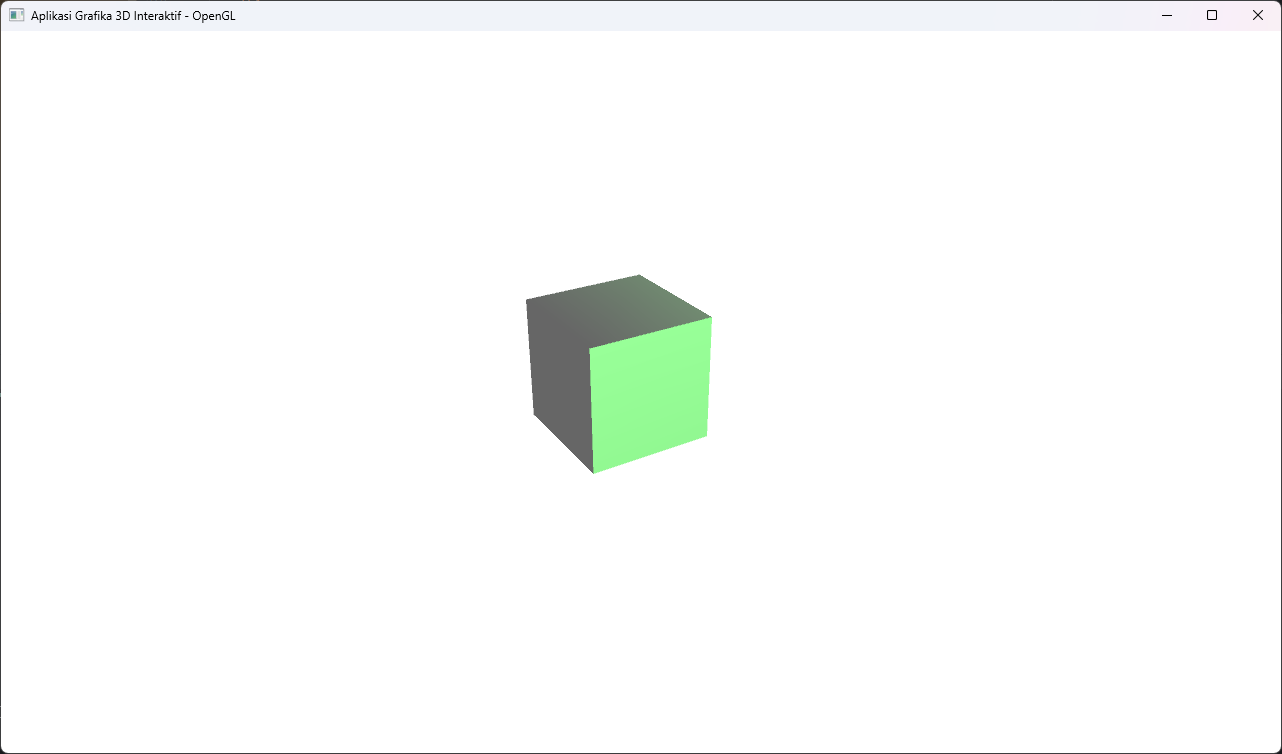
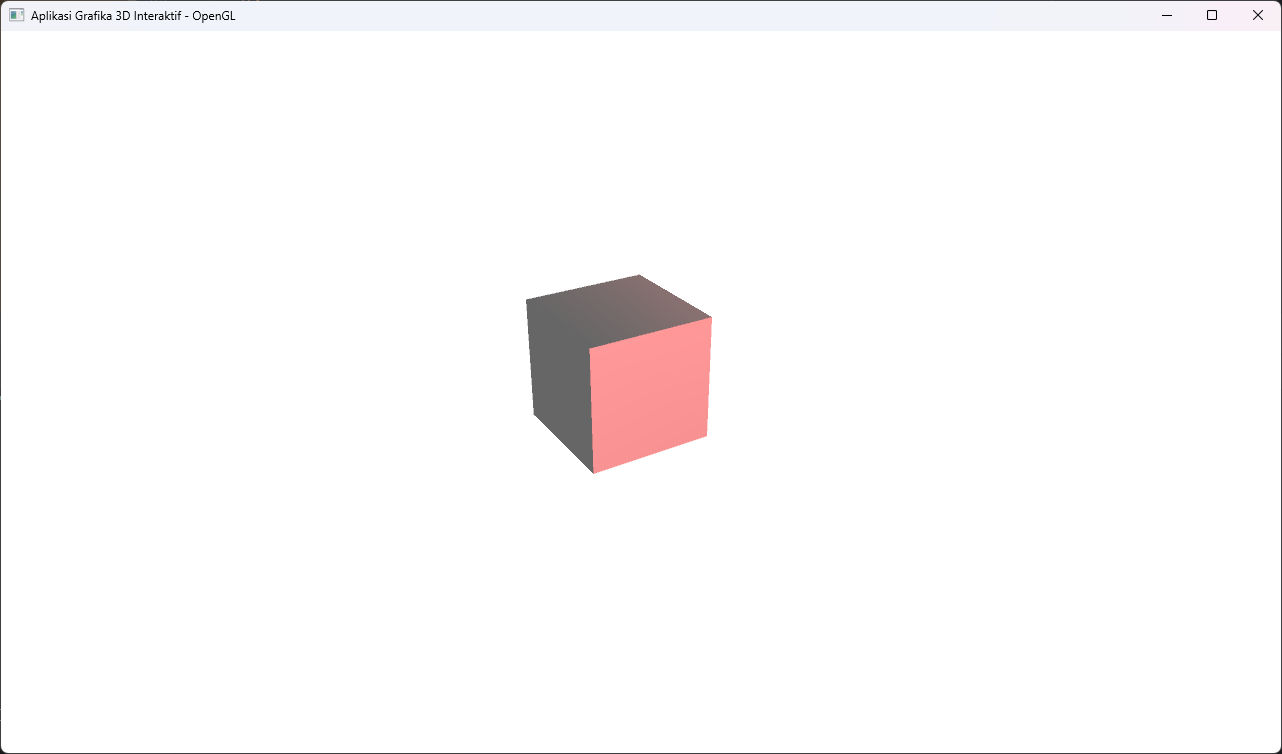
Description automatically generated

A white cube with a blue background

Description automatically generated

A grey cube with a white background

Description automatically generated



# Deskripsi Penjelasan Masing-Masing Output

## Output Modul A 2D

Output dari program Modul A menunjukkan bahwa seluruh fungsionalitas yang disyaratkan dalam soal telah berhasil diimplementasikan. Penjelasan untuk setiap output visual adalah sebagai berikut:

* Output Terminal

Saat program dijalankan, terminal akan menampilkan panduan penggunaan aplikasi secara lengkap. Ini mencakup informasi mengenai:

* + **Mode Menggambar**: Tombol untuk beralih mode menggambar Titik [P], Garis [L], Persegi [R], dan Elips [E].
  + **Seleksi & Transformasi**: Cara menyeleksi, menggeser, melakukan translasi (Panah), rotasi (Q/A), dan skala (W/S) pada objek.
  + **Manajemen Objek**: Shortcut untuk menyalin (Ctrl+C), menempel (Ctrl+V), dan menghapus (DELETE) objek.
  + **Atribut**: Pilihan untuk mengubah warna dan ketebalan garis.
  + **Windowing & Clipping**: Perintah untuk membuat [C], menonaktifkan [D], dan menggeser [G] window.
* Tampilan Awal Aplikasi

Gambar ini menunjukkan jendela aplikasi saat pertama kali dibuka. Terdapat sebuah kanvas putih kosong yang siap digunakan sebagai area untuk menggambar objek-objek 2D.

* Penggambaran dan Seleksi Objek Dasar

Serangkaian gambar output menunjukkan keberhasilan program dalam menggambar berbagai objek dasar sesuai spesifikasi soal, yaitu **Titik**, **Garis**, **Persegi**, dan **Elips**. Input koordinat untuk menggambar objek-objek ini dilakukan melalui klik mouse pada kanvas. Pada setiap gambar, terlihat satu objek berwarna oranye, yang menandakan bahwa objek tersebut sedang dalam mode terseleksi.

* Perubahan Atribut dan Manajemen Objek

Output visual menunjukkan beberapa objek dengan warna yang berbeda-beda (merah, hijau, biru) dalam satu kanvas. Hal ini membuktikan bahwa fungsi untuk mengubah warna objek telah berhasil diimplementasikan sesuai kebutuhan. Selain itu, adanya beberapa objek identik menunjukkan fungsionalitas seperti *copy-paste* juga berjalan.

* Transformasi Geometri

Gambar output menampilkan sebuah elips yang telah terseleksi (oranye) dan posisinya sedikit bergeser atau berotasi dari elips lain yang identik. Ini menunjukkan implementasi dari fitur transformasi geometri, di mana objek yang telah digambar dapat dikenai **translasi**, **rotasi**, atau **scaling**.

* Windowing dan Clipping

Gambar output ini secara efektif mendemonstrasikan fungsionalitas *windowing* dan *clipping*:

* + Sebuah *window* aktif telah berhasil ditentukan, ditandai dengan area persegi panjang bergaris putus-putus merah.
  + Objek yang sepenuhnya berada di dalam *window* (sebuah persegi) berubah warna menjadi hijau, sesuai dengan spesifikasi soal.
  + Objek yang posisinya sebagian di dalam dan sebagian di luar *window* (objek garis dan persegi besar) akan dikenai *clipping*. Hanya bagian dari objek yang berada di dalam *window* yang ditampilkan, sementara sisanya dihilangkan. Ini membuktikan implementasi algoritma *clipping* seperti Cohen-Sutherland telah berhasil.

## Output Modul B 3D

Output dari program Modul B menunjukkan keberhasilan implementasi aplikasi untuk visualisasi dan manipulasi objek 3D yang interaktif.

* **Output Terminal** Saat program dijalankan, terminal menampilkan panduan penggunaan aplikasi 3D. Panduan ini mencakup kontrol untuk:
  + **Manajemen File**: Impor [I] dan Ekspor [O] model dalam format .obj.
  + **Kontrol Objek**: Rotasi menggunakan mouse, Zoom dengan *scroll wheel*, dan Translasi dengan tombol W, A, S, D.
  + **Perubahan Warna**: Shortcut keyboard untuk mengubah warna objek menjadi Merah, Hijau, Biru, dan lainnya.
* **Visualisasi dan Shading Objek 3D** Output visual menampilkan sebuah objek kubus 3D yang dirender pada kanvas. Masing-masing sisi kubus memiliki tingkat kecerahan warna yang berbeda. Hal ini menunjukkan bahwa implementasi model pencahayaan sederhana, yang mencakup komponen *Ambient*, *Diffuse*, dan *Specular Light*, telah berhasil diterapkan. Efek *shading* ini memberikan kesan kedalaman dan volume pada objek.
* **Transformasi Objek 3D** Serangkaian gambar output menampilkan kubus dari berbagai sudut, ukuran, dan posisi. Ini membuktikan bahwa fitur transformasi 3D telah diimplementasikan dengan benar:
  + **Rotasi**: Objek dapat diputar secara bebas menggunakan klik dan seret mouse.
  + **Translasi**: Objek dapat digeser pada sumbu X dan Y menggunakan tombol W, A, S, D di keyboard.
  + **Skala (Zoom)**: Ukuran objek dapat diperbesar atau diperkecil menggunakan *scroll wheel* pada mouse.
* **Perubahan Warna Objek** Gambar-gambar output menunjukkan kubus yang sama ditampilkan dalam berbagai warna, seperti merah, hijau, biru, dan kuning. Fungsionalitas ini, yang diakses melalui *shortcut keyboard, berhasil mengubah warna dasar material objek sambil tetap mempertahankan kalkulasi shading dari sumber cahaya.*
* **Kamera dan Perspektif** Seluruh rendering objek 3D menggunakan proyeksi perspektif (gluPerspective) dan posisi kamera diatur menggunakan gluLookAt. Hal ini memberikan tampilan yang realistis, di mana bagian objek yang lebih jauh dari kamera akan tampak lebih kecil, sesuai dengan cara kerja pandangan mata manusia.

# Referensi/Sumber Pengerjaan Project

de Vries, J. (n.d.). Retrieved from LearnOpenGL: https://learnopengl.com/

*Line Clipping | Set 1 (Cohen–Sutherland Algorithm)*. (2024, March 12). Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/line-clipping-set-1-cohen-sutherland-algorithm/

*OpenGL Lighting*. (n.d.). Retrieved from Tutorialspoint: https://www.tutorialspoint.com/opengl/opengl\_lighting.html

*PyOpenGL 3.1.7 documentation*. (n.d.). Retrieved from PyOpenGL: http://pyopengl.sourceforge.net/documentation/index.html

Wikipedia contributors. (2025, June 2). *Wavefront .obj file*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Wavefront\_.obj\_file