LAPORAN PRAKTIKUM TRANSFORMASI GEOMETRI 3 DIMENSI
GRAFIKA KOMPUTER

DOSEN PENGAMPU:

Dr. Putu Hendra Suputra, S.Kom., M.Cs.

DISUSUN OLEH:

I Gede Ryandika Pramudia Wardana (2315101012/5A)

PROGRAM STUDI S1 ILMU KOMPUTER

JURUSAN TEKNIK INFORMATIKA

FAKULTAS TEKNIK DAN KEJURUAN

UNIVERSITAS PENDIDIKAN GANESHA

2025

DAFTAR ISI

BAB I
PENDAHULUAN

1.1 Latar Belakang

Grafika komputer merupakan salah satu bidang ilmu komputer yang berkaitan dengan proses pembuatan dan manipulasi gambar (visual) secara digital. Dalam perkembangannya, visualisasi 3 dimensi (3D) menjadi elemen krusial dalam berbagai industri, mulai dari simulasi, desain arsitektur, hingga pengembangan *video game*. Namun, tantangan utama dalam grafika komputer adalah bagaimana merepresentasikan objek yang memiliki tiga sumbu (panjang, lebar, dan kedalaman/tinggi) ke dalam layar monitor yang hanya bersifat 2 dimensi (datar).

Untuk menjembatani perbedaan dimensi tersebut, diperlukan pemahaman mendalam mengenai konsep matematika, khususnya matriks dan vektor. Konsep Transformasi Geometri (Translasi, Rotasi, dan Skala) memungkinkan sebuah objek untuk dimanipulasi posisinya dalam ruang maya. Selain itu, teknik proyeksi diperlukan untuk memetakan koordinat 3D tersebut agar dapat digambar menjadi piksel-piksel pada layar.

Dalam praktikum ini, penulis membangun sebuah simulasi *rendering* objek 3D sederhana menggunakan bahasa pemrograman Python dan pustaka Pygame. Proyek ini menerapkan algoritma dasar pembentukan garis (Bresenham) serta operasi matriks homogen untuk mensimulasikan pergerakan objek secara interaktif, sehingga pemahaman mengenai logika di balik *engine* grafis dapat dipahami secara mendasar.

1.2 Rumusan Masalah

Berdasarkan latar belakang di atas, rumusan masalah dalam praktikum ini adalah:

1. Bagaimana cara merepresentasikan struktur data objek 3 dimensi (titik dan garis) ke dalam program komputer?
2. Bagaimana menerapkan algoritma garis Bresenham untuk menghubungkan titik-titik koordinat objek?
3. Bagaimana mengimplementasikan rumus matematika transformasi geometri (Translasi, Rotasi, dan Skala) menggunakan operasi matriks homogen 4x4?
4. Bagaimana membuat sistem interaktif di mana pengguna dapat mengontrol transformasi objek menggunakan *input* keyboard?

1.3 Tujuan

Tujuan dari pelaksanaan praktikum dan penyusunan laporan ini adalah:

1. Memahami konsep dasar struktur data *wireframe* (kerangka) yang terdiri dari *vertices* (titik) dan *edges* (garis).

2. Mampu mengimplementasikan algoritma Bresenham untuk penggambaran garis yang efisien.
3. Mampu menyusun fungsi perkalian matriks untuk melakukan transformasi geometri 3D secara komposit.
4. Menghasilkan aplikasi visualisasi 3D sederhana yang mampu melakukan animasi rotasi, perpindahan posisi, dan perubahan ukuran objek secara *real-time*.

BAB II
PEMBAHASAN

2.1 Teori Graf

Teori Graf dalam Representasi Objek 3D. Dalam konteks grafika komputer, representasi objek yang digunakan pada praktikum ini mengadopsi konsep dasar Teori Graf. Objek 3D direpresentasikan sebagai sebuah Wireframe (kerangka kawat), yang secara matematis adalah sebuah graf $G = (V, E)$.

1. **Vertices (V):** Merupakan himpunan titik-titik sudut yang memiliki koordinat $(x, y, z$ di dalam ruang 3 dimensi. Dalam kode program, ini disimpan dalam *list* self.vertices. Setiap titik merepresentasikan simpul (node) dari graf.
2. **Edges (E):** Merupakan himpunan garis yang menghubungkan sepasang *vertices*. Dalam kode program, ini disimpan dalam *list* self.edges yang berisi pasangan indeks titik (misalnya menghubungkan titik ke-0 dengan titik ke-1). Ini merepresentasikan sisi (rusuk) dari graf.

Dengan menghubungkan *vertices* berdasarkan aturan yang ada pada *edges* menggunakan algoritma garis, komputer dapat memvisualisasikan bentuk kerangka dari bangun ruang seperti kubus, balok, atau limas.

2.2 Algoritma Garis Bressenham

Algoritma Bresenham adalah algoritma konversi raster (rasterisasi) yang dikembangkan oleh Jack E. Bresenham pada tahun 1962. Algoritma ini digunakan untuk menentukan titik-titik piksel dalam grid 2 dimensi yang harus dinyalakan agar membentuk pendekatan garis lurus terbaik antara dua titik koordinat yang diberikan, yaitu $(x_1,\ y_1)$ dan $(x_2,\ y_2)$.

Kelebihan utama dari algoritma Bresenham dibandingkan algoritma garis lainnya (seperti DDA atau *Digital Differential Analyzer*) adalah efisiensi komputasinya. Algoritma ini bekerja menggunakan aritmatika bilangan bulat (*integer arithmetic*) yang hanya melibatkan operasi penjumlahan, pengurangan, dan penggandaan bit (*bit shifting*), tanpa memerlukan operasi pembagian atau bilangan pecahan (*floating point*) yang memakan memori lebih besar pada prosesor komputer.

Prinsip kerja algoritma ini didasarkan pada **Parameter Keputusan** (P_k). Pada setiap langkah pengambilan sampel piksel di sepanjang sumbu utama (misalnya sumbu x), algoritma menentukan piksel berikutnya berdasarkan tanda dari parameter keputusan tersebut:

- Jika $P_k < 0$, maka titik selanjutnya berada pada posisi $(x_{k+1},\ y_k)$.
- Jika $P_k \geq 0$, maka titik selanjutnya berada pada posisi $(x_{k+1},\ y_{k+1})$.

Dalam implementasi program ini, algoritma Bresenham digunakan pada fungsi buatGarisBressenham untuk menghubungkan titik-titik sudut (*vertices*) objek 3D yang telah diproyeksikan ke layar 2D, sehingga membentuk kerangka visual (*wireframe*) yang utuh.

2.3 Matriks Transformasi

Transformasi geometri dalam ruang 3 dimensi melibatkan manipulasi posisi, orientasi, dan ukuran objek. Untuk mempermudah komputasi komputer, transformasi ini direpresentasikan dalam bentuk matriks.

Agar operasi translasi (pergeseran) dapat dilakukan dengan perkalian matriks (sama seperti rotasi dan skala), digunakan sistem **Koordinat Homogen**. Dalam sistem ini, sebuah titik 3D (x, y, z) direpresentasikan sebagai vektor 4 elemen $(x, y, z, 1)$. Hal ini memungkinkan penggunaan matriks transformasi berukuran 4×4.

Berikut adalah jenis-jenis transformasi yang diterapkan dalam praktikum ini:

A. **Translasi (Pergeseran)**. Translasi adalah memindahkan objek dari satu posisi ke posisi lain dengan menambahkan *offset* (t_x, t_y, t_z)\$ pada koordinat asli. Matriks translasinya adalah:

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

B. **Skala (Penskalaan).** Skala adalah mengubah ukuran objek (memperbesar atau memperkecil) berdasarkan faktor skala (s_x, s_y, s_z). Matriks skalanya adalah:

$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

C. Rotasi (Perputaran) Rotasi memutar objek terhadap sumbu tertentu (X, Y, atau Z) sebesar sudut θ.

- **Rotasi Sumbu X**

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **Rotasi Sumbu Y**

$$R_x(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **Rotasi Sumbu Z**

$$R_x(\theta) = \begin{bmatrix} cos\,\theta & -sin\,\theta & 0 & 0 \\ sin\,\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

E. **Matriks Komposit.** Untuk melakukan beberapa transformasi sekaligus (misalnya objek diputar lalu digeser), kita mengalikan matriks-matriks tersebut menjadi satu matriks gabungan (Komposit).

$$M_{final} = M_{translasi} \cdot M_{rotasi} \cdot M_{skala}$$

Dalam program, ini diimplementasikan melalui fungsi kompositMatriks yang memungkinkan efisiensi tinggi, karena komputer hanya perlu mengalikan setiap titik dengan satu matriks final (m_final), bukan mengalikan dengan tiga matriks terpisah secara berulang-ulang

2.4 Menyiapkan Environment

Hal yang saya siapkan dan gunakan untuk praktikum ini adalah:

1. IDE/text editor Visual Studio Code yang saya sudah pernah saya setup sebelumnya, dapat diunduh melalui tautan berikut: https://code.visualstudio.com/Download
2. Bahasa pemrograman Python yang saya sudah lakukan setup juga sebelumnya, dapat diunduh melalui tautan berikut: https://www.python.org/downloads/
3. Library Pygame, yang saya unduh melalui terminal VSCode.
4. Library math untuk melakukan operasi perhitungan matriks serta transformasi geometri.
5. Dua buah file berformat .txt yang berisikan informasi koordinat 3 dimensi serta kemana saya setiap koordinat tersebut terhubung. Disini saya menggunakan bentuk kubus.

File titik.txt:

-50,-50,-50

50,-50,-50

50,50,-50

-50,50,-50

-50,-50,50

50,-50,50

50,50,50

-50,50,50

File garis.txt

0,1

1,2

2,3

3,0

4,5

5,6

6,7

7,4

0,4

1,5

2,6

3,7

Kemudian saya menyiapkan latar dan warna dasar untuk program ini berjalan dengan cara berikut:

```python
# Import library yang dibutuhkan
import pygame
import sys
import math

# Mengatur ukuran window dan warna dasar
WIDTH, HEIGHT = 800, 600
TITIK_PUSAT = (WIDTH // 2, HEIGHT // 2)

PUTIH = (255, 255, 255)
HITAM = (0, 0, 0)
MERAH = (255, 0, 0)
HIJAU = (0, 255, 0)
BIRU = (0, 0, 255)

pygame.init()
layar = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Tugas Grafika Komputer: Transformasi 3D")
clock = pygame.time.Clock()

```

Kemudian saya memasukkan kode dan algoritma untuk membuat titik/memanipulasi pixel serta kode dan algoritma untuk membuat garis yaitu algoritma Bressenham. Kedua kode/algortima ini saya ambil dari kode saya sebelumnya (praktikum memanipulasi piksel dan membuat garis dan lingkaran menggunakan beberapa algoritma):

```python

# Algoritma memanipulasi pixel/membuat titik
Windsurf: Refactor | Explain | Generate Docstring | ×
def buatPixel(x, y, color):
    if 0 <= x < WIDTH and 0 <= y < HEIGHT:
        layar.set_at((int(x), int(y)), color)

```

```python
# Algoritma menggambar garis Bresenham
Windsurf: Refactor | Explain | Generate Docstring | ×
def buatGarisBressenham(p1, p2, color):
    x1, y1 = int (p1[0]), int (p1[1])
    x2, y2 = int (p2[0]), int (p2[1])

    # cari delta
    dx = abs(x2 - x1)
    dy = abs(y2 - y1)

    x, y = x1, y1
    pk = 2* dy - dx

    arah_x = 1 if x1 < x2 else -1
    arah_y = 1 if y1 < y2 else -1

    if abs(dy) > abs(dx):
        pk = 2 * abs(dx) - abs(dy)
        for k in range(dy):
            if pk < 0:
                y = y + arah_y
                pk = pk + 2 * abs(dx)
            else:
                x = x + arah_x
                y = y + arah_y
                pk = pk + 2 * abs(dx) - 2 * abs(dy)
            buatPixel(x, y, color)
    else:
        pk = 2 * abs(dy) - abs(dx)
        for k in range(dx):
            if pk < 0:
                x = x + arah_x
                pk = pk + 2 * abs(dy)
            else:
                x = x + arah_x
                y = y + arah_y
                pk = pk + 2 * abs(dy) - 2 * abs(dx)
            buatPixel(x, y, color)

```

Kemudian saya membuat class khusus bernama render3D. Class ini saya buat dengan tujuan untuk menampilkan objek sekaligus melakukan operasi transformasi geometri disini, untuk sekarang saya menggunakan class ini untuk menampilkan hasil *plotting* terlebih dahulu.

```python
Windsurf: Refactor | Explain
class render3D:
    # Inisialisasi array untuk menyimpan titik dan garis
    Windsurf: Refactor | Explain | Generate Docstring | ×
    def __init__(self):
        self.vertices = []
        self.edges = []

```

Inisialisasi array untuk menyimpan titik dan garis.

```python
    # Load data dari file titik dan garis
    Windsurf: Refactor | Explain | Generate Docstring | ×
    def load_data(self, file_titik, file_garis):
        # membaca file yang telah dibuat
        # parsing satu-persatu
        try:
            with open(file_titik, 'r') as f:
                for line in f:
                    parts = line.strip().split(',')
                    if len(parts) == 3:
                        self.vertices.append([float(parts[0]), float(parts[1]), float(parts[2])])
                    elif len(parts) != 3:
                        print(f"Baris tidak valid di file titik: {line.strip()}")

            with open(file_garis, 'r') as f:
                for line in f:
                    parts = line.strip().split(',')
                    if len(parts) == 2:
                        self.edges.append([int(parts[0]), int(parts[1])])
                    elif len(parts) != 2:
                        print(f"Baris tidak valid di file garis: {line.strip()}")
            print(f"Data Loaded: {len(self.vertices)} titik, {len(self.edges)} garis.")
        except Exception as e:
            print(f"Error loading data: {e}")

```

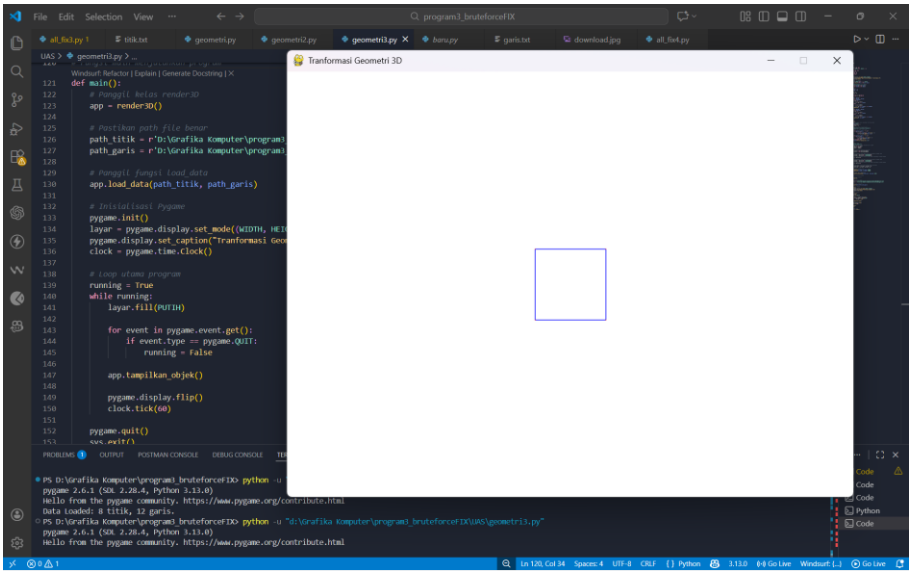
Mengambil/*load* data dari file titik.txt dan garis.txt.

```python
    # Menampilkan objek
    Windsurf: Refactor | Explain | Generate Docstring | ×
    def tampilkan_objek(self):
        # Loop menelusuri setiap pasangan garis (egde)
        for edge in self.edges:
            index1 = edge[0]
            index2 = edge[1]

            # Mendapatkan koordinat titik asli
            p1_asli = self.vertices[index1]
            p2_asli = self.vertices[index2]

            # Manambahkan offset agar titik (0,0) berada di tengah layar
            x1_layar = p1_asli[0] + TITIK_PUSAT[0]
            y1_layar = p1_asli[1] + TITIK_PUSAT[1]
            # z tidak digunakan untuk 2D, nanti pada tranformasi 3D akan dipakai
            z1 = p1_asli[2]

            # Menambahkan offset agar titik (0,0) berada di tengah layar
            x2_layar = p2_asli[0] + TITIK_PUSAT[0]
            y2_layar = p2_asli[1] + TITIK_PUSAT[1]
            # z tidak digunakan untuk 2D, nanti pada tranformasi 3D akan dipakai
            z2 = p2_asli[2]

            # Menyiapkan format titik untuk fungsi Bressenham
            titik_start = [x1_layar, y1_layar, z1]
            titik_end = [x2_layar, y2_layar, z2]

            # gambar elemen visual
            buatPixel(x1_layar, y1_layar, MERAH)
            buatPixel(x2_layar, y2_layar, MERAH)

            # Gambar garis penghubung
            buatGarisBressenham(titik_start, titik_end, BIRU)

```

Setelah mendapatkan data dari koordinat titik dan garis, lakukan penggambaran/*plotting* titik dan membuat garis penghubung menggunakan algoritma Bressenham. Saya membuat titik pusat (0, 0) berada di tengah layar.

```python
# Fungsi main menjalankan program
Windsurf: Refactor | Explain | Generate Docstring | ×
def main():
    # Panggil kelas render3D
    app = render3D()

    # Pastikan path file benar
    path_titik = r'D:\Grafika Komputer\program3_bruteforceFIX\UAS\titik.txt'
    path_garis = r'D:\Grafika Komputer\program3_bruteforceFIX\UAS\garis.txt'

    # Panggil fungsi load_data
    app.load_data(path_titik, path_garis)

    # Inisialisasi Pygame
    pygame.init()
    layar = pygame.display.set_mode((WIDTH, HEIGHT))
    pygame.display.set_caption("Tranformasi Geometri 3D")
    clock = pygame.time.Clock()

    # Loop utama program
    running = True
    while running:
        layar.fill(PUTIH)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False

        app.tampilkan_objek()

        pygame.display.flip()
        clock.tick(60)

    pygame.quit()
    sys.exit()

# Panggil fungsi main
if __name__ == "__main__":
    main()
```

Fungsi main (fungsi utama) yang memanggil seluruh fungsi dan menjalankan program.



Hasil program ketika dijalankan.

2.5 Membuat Algoritma Transformasi Geometri

Operasi transformasi geometri disini menggunakan bentuk matriks homogen. Untuk membuat transformasi geometri lebih sederhana, pertama saya akan mengubah koordinat titik menjadi vektor homogen kemudian mengalikan vektor homogen tersebut dengan matriks homogen 3D dimensi. Berikut algoritma perkaliannya:

```
61  # Membuat fungsi perkalian matriks homogen dengan titik 3D
    Windsurf: Refactor | Explain | Generate Docstring | ×
62  def perkalianMatriks(matriks, titik):
63      # Mengonversi menjadi Homogeneous [x,y,z,1]
64      vec = [titik[0], titik[1], titik[2], 1]
65      hasil = [0, 0, 0, 0]
66
67      for i in range(4):
68          total = 0
69          for j in range(4):
70              total += matriks[i][j] * vec[j]
71          hasil[i] = total
72
73      return [hasil[0], hasil[1], hasil[2]]
```

Algoritma perkalian matriks dengan titik/vektor.

```
125 # Perkalian matriks dengan matriks
    Windsurf: Refactor | Explain | Generate Docstring | ×
126 def kompositMatriks(matriksA, matriksB):
127     hasil = [[0 for _ in range(4)] for _ in range(4)]
128     for i in range(4):
129         for j in range(4):
130             total = 0
131             for k in range(4):
132                 total += matriksA[i][k] * matriksB[k][j]
133             hasil[i][j] = total
134     return hasil
```

Algoritma perkalian matriks dengan matriks.

Setelah itu, saya akan membuat algoritma rumus operasi masing-masing dari transformasi geometri (skala, rotasi, translasi).

```
75  # Inisialisasi matriks operasi transformasi dasar
    Windsurf: Refactor | Explain | Generate Docstring | ×
76  def skala(sx, sy, sz):
77      return [
78          [sx, 0, 0, 0],
79          [0, sy, 0, 0],
80          [0, 0, sz, 0],
81          [0, 0, 0, 1]
82      ]
83
```

Matriks operasi skala.

```python
# Rotasi sumbu x
Windsurf: Refactor | Explain | Generate Docstring | ×
def rotasi_x(sudut):
    c = math.cos(sudut)
    s = math.sin(sudut)
    return [
        [1, 0, 0, 0],
        [0, c, -s, 0],
        [0, s, c, 0],
        [0, 0, 0, 1]
    ]
```

Matriks operasi rotasi sumbu x.

```python
# Rotasi sumbu y
Windsurf: Refactor | Explain | Generate Docstring | ×
def rotasi_y(sudut):
    c = math.cos(sudut)
    s = math.sin(sudut)
    return [
        [c, 0, s, 0],
        [0, 1, 0, 0],
        [-s, 0, c, 0],
        [0, 0, 0, 1]
    ]

```

Matriks operasi rotasi sumbu y.

```python
# Rotasi sumbu z
Windsurf: Refactor | Explain | Generate Docstring | ×
def rotasi_z(sudut):
    c = math.cos(sudut)
    s = math.sin(sudut)
    return [
        [c, -s, 0, 0],
        [s, c, 0, 0],
        [0, 0, 1, 0],
        [0, 0, 0, 1]
    ]

```

Matriks operasi rotasi sumbu z.

```python
def translasi(tx, ty, tz):
    return [
        [1, 0, 0, tx],
        [0, 1, 0, ty],
        [0, 0, 1, tz],
        [0, 0, 0, 1]
    ]
```

Matriks operasi translasi.

Sebelum melakukan operasinya, saya menambahkan beberapa variabel berikut pada def __init__ () dan def load_data():

```
136
    Windsurf: Refactor | Explain
137 class render3D:
138     # Inisialisasi array untuk menyimpan titik dan garis
        Windsurf: Refactor | Explain | Generate Docstring | ×
139     def __init__(self):
140         self.vertices = []
141         self.edges = []
142 
143         # Inisialisasi parameter transformasi awal/asli
144         self.posisi = [0, 0, 0]
145         self.rotasi = [0, 0, 0]
146         self.skala = [1, 1, 1]
147 
148         # Menambahkan font
149         self.font = pygame.font.SysFont("Arial", 12)
150 
151         # Menambahkan label ke setiap titik
152         self.labels = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
153 
154         # Menambahkan tempat menyimpan titik yang sudah dihitung
155         self.transformed_vertices = []
156 
```

Menambahkan *self.transformed_vertices* = [] sebagai tempat menyimpan titik yang sudah di transformasi. Juga menambahkan label untuk mengetahui setiap titik.

```
157     # Load data dari file titik dan garis
        Windsurf: Refactor | Explain | Generate Docstring | ×
158     def load_data(self, file_titik, file_garis):
159         # membaca file yang telah dibuat
160         # parsing satu-persatu
161         try:
162             with open(file_titik, 'r') as f:
163                 for line in f:
164                     parts = line.strip().split(',')
165                     if len(parts) == 3:
166                         self.vertices.append([float(parts[0]), float(parts[1]), float(parts[2])])
167                     elif len(parts) != 3:
168                         print(f"Baris tidak valid di file titik: {line.strip()}")
169 
170             with open(file_garis, 'r') as f:
171                 for line in f:
172                     parts = line.strip().split(',')
173                     if len(parts) == 2:
174                         self.edges.append([int(parts[0]), int(parts[1])])
175                     elif len(parts) != 2:
176                         print(f"Baris tidak valid di file garis: {line.strip()}")
177             print(f"Data Loaded: {len(self.vertices)} titik, {len(self.edges)} garis.")
178         except Exception as e:
179             print(f"Error loading data: {e}")
180 
181         self.hitung_transformasi()
```

Memanggil fungsi *hitung_transformasi()* yang akan dibuat nanti, ini bertujuan jika sudah selesai membaca dan file tersebut ada & isinya sesuai, maka selanjutnya akan masuk ke operasi tersebut.

```
Windsurf: Refactor | Explain | Generate Docstring | ×
def hitung_transformasi(self):
    # 1. Matriks Skala
    m_skala = skala(self.skala[0], self.skala[1], self.skala[2])

    # 2. Matriks Rotasi
    m_rot_x = rotasi_x(math.radians(self.rotasi[0]))
    m_rot_y = rotasi_y(math.radians(self.rotasi[1]))
    m_rot_z = rotasi_z(math.radians(self.rotasi[2]))

    # Gabungkan Rotasi (urutan z * y * x)
    m_rot_gabung = kompositMatriks(m_rot_y, m_rot_x)
    m_rot_gabung = kompositMatriks(m_rot_z, m_rot_gabung)

    # Posisi objekt relatif terhadapad (0, 0, 0)
    m_trans = translasi(self.posisi[0], self.posisi[1], self.posisi[2])

    # Gabungkan semua transformasi
    m_final = kompositMatriks(m_rot_gabung, m_skala)
    m_final = kompositMatriks(m_trans, m_final)

    # terapkan ke semua titik
    self.transformed_vertices = []
    for v in self.vertices:
        v_baru = perkalianMatriks(m_final, v)
        self.transformed_vertices.append(v_baru)

```

Membuat fungsi baru, fungsi *hitung_transformasi()* untuk menghitung transformasi geometri.

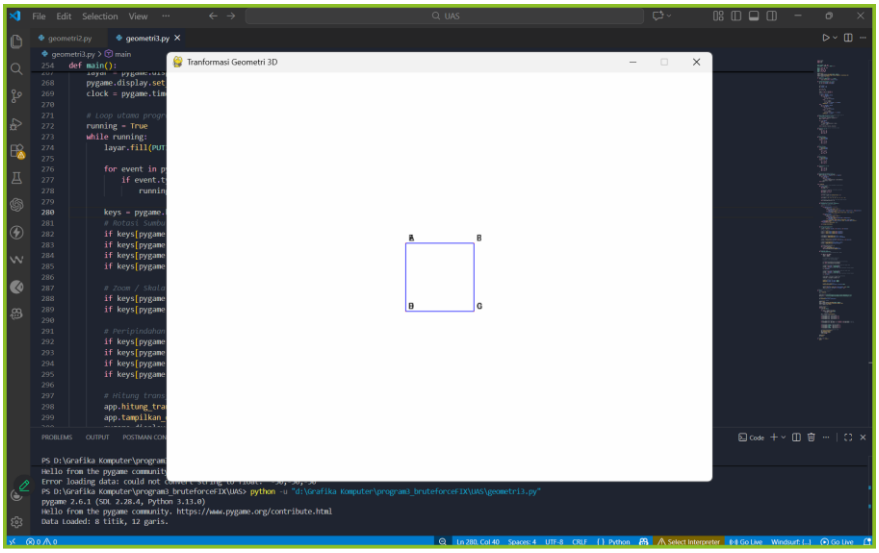
```
def tampilkan_objek(self):
    # Loop menelusuri setiap pasangan garis (egde)

    for edge in self.edges:
        index1 = edge[0]
        index2 = edge[1]

        # # Mendapatkan koordinat titik asli
        # p1_asli = self.vertices[index1]
        # p2_asli = self.vertices[index2]

        # Menggunakan koordinat titik yang sudah ditransformasi
        p1 = self.transformed_vertices[index1]
        p2 = self.transformed_vertices[index2]

        # Manambahkan offset agar titik (0,0) berada di tengah layar
        x1_layar = int(p1[0] + TITIK_PUSAT[0])
        y1_layar = int(p1[1] + TITIK_PUSAT[1])
        # z tidak digunakan untuk 2D, nanti pada tranformasi 3D akan dipakai
        z1 = int(p1[2])

        # Menambahkan offset agar titik (0,0) berada di tengah layar
        x2_layar = int(p2[0] + TITIK_PUSAT[0])
        y2_layar = int(p2[1] + TITIK_PUSAT[1])
        # z tidak digunakan untuk 2D, nanti pada tranformasi 3D akan dipakai
        z2 = int(p2[2])

        # Menyiapkan format titik untuk fungsi Bressenham
        titik_start = [x1_layar, y1_layar, z1]
        titik_end = [x2_layar, y2_layar, z2]

        # gambar elemen visual
        buatPixel(x1_layar, y1_layar, MERAH)
        buatPixel(x2_layar, y2_layar, MERAH)

        # Gambar garis penghubung
        buatGarisBressenham(titik_start, titik_end, BIRU)

        # Tampilkan teks
        teks_A = self.font.render(self.labels[index1], True, HITAM)
        layar.blit(teks_A, (x1_layar + 5, y1_layar - 15))

```

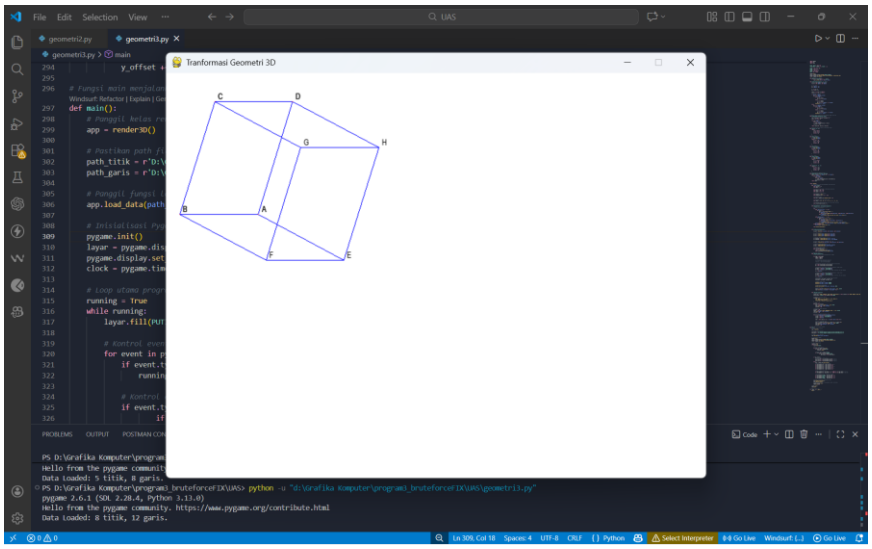
Memperbarui isi fungsi *tampilkan_objek()*, tidak lagi menggunakan titik asli melainkan menggunakan titik yang sudah di transformasi untuk ditampilkan, sekaligus menampilkan label setiap titik.

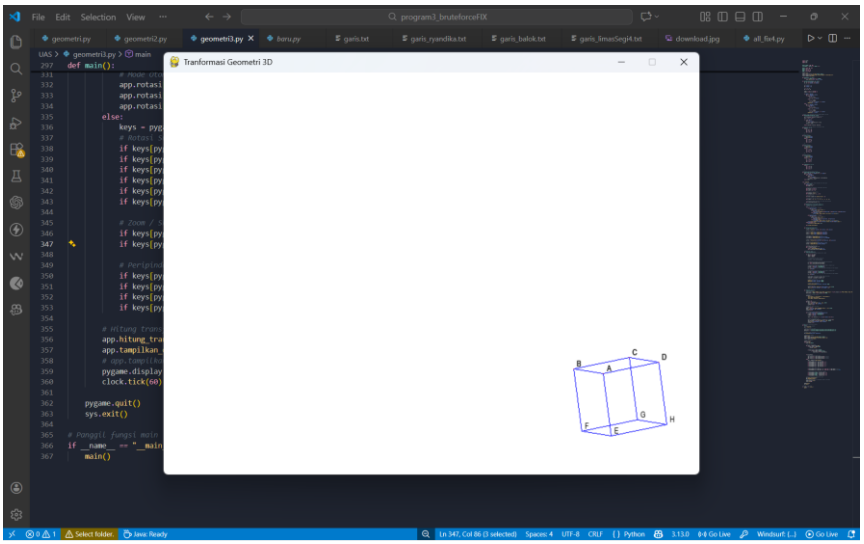
```
289  def main():
306      # Loop utama program
307      running = True
308      while running:
309          layar.fill(PUTIH)
310
311          for event in pygame.event.get():
312              if event.type == pygame.QUIT:
313                  running = False
314
315          keys = pygame.key.get_pressed()
316          # Rotasi Sumbu x dan y (Huruf WASD)
317          if keys[pygame.K_w]:  app.rotasi[0] -= 2 # x
318          if keys[pygame.K_s]:  app.rotasi[0] += 2
319          if keys[pygame.K_a]:  app.rotasi[1] -= 2 # y
320          if keys[pygame.K_d]:  app.rotasi[1] += 2
321          if keys[pygame.K_q]:  app.rotasi[2] -= 1 # z
322          if keys[pygame.K_e]:  app.rotasi[2] += 1
323
324          # Zoom / Skala (Huruf Z/X)
325          if keys[pygame.K_z]: app.skala = [s + 0.05 for s in app.skala] # zoom in
326          if keys[pygame.K_x]: app.skala = [s - 0.05 for s in app.skala] # zoom out
327
328          # Peripindahan posisi (translasi) (Panah)
329          if keys[pygame.K_LEFT]:  app.posisi[0] -= 2
330          if keys[pygame.K_RIGHT]: app.posisi[0] += 2
331          if keys[pygame.K_UP]:    app.posisi[1] -= 2
332          if keys[pygame.K_DOWN]:  app.posisi[1] += 2
333
334          # Hitung transformasi berdasarkan input
335          app.hitung_transformasi()
336          app.tampilkan_objek()
337          pygame.display.flip()
338          clock.tick(60)
339
340      pygame.quit()
341      sys.exit()
342
343  # Panggil fungsi main
344  if __name__ == "__main__":
345      main()
```

Memperbarui loop utama program untuk menambahkan operasi transformasi geometri sekaligus menambahkan kontrol untuk melakukan transformasi. Berikut mapping tombol keyboard untuk melakukan transformasi geometri:

W	Rotasi ke atas (sumbu x)
A	Rotasi ke kiri (sumbu x)
S	Rotasi ke bawah (sumbu y)
D	Rotasi ke kanan (sumbu y)
Q	Rotasi ke kiri/*counter-clockwise* (sumbu z)
E	Rotasi ke kanan/*clockwise* (sumbu z)
Z	Skala perbesar (zoom in)
X	Skala perkecil (zoom out)
↑	Translasi/perpindahan ke atas
←	Translasi/perpindahan ke kiri
→	Translasi/perpindahan ke kanan
↓	Translasi/perpindahan ke bawah







Hasil transformasi.

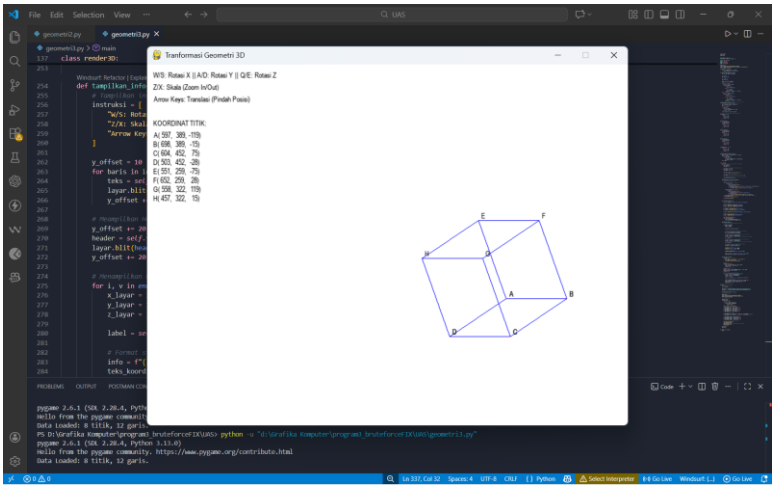
2.6 Penambahan Kreatifitas dan Fitur

Dengan akhir langkah sebelumnya, transformasi geometri dasar sudah berhasil dilakukan. Selanjutnya bisa penambahan fitur dan kreativitas. Seperti menambahkan titik koordinat sekarang atau mengubah bentuk objek. Berikut beberapa penambahan yang saya lakukan:

1. Penambahan fungsi header informasi, ditambahkan pada class objek3D.

```
class render3D:

    Windsurf: Refactor | Explain | Generate Docstring | ×
    def tampilkan_info(self):
        # Tampilkan instruksi kontrol
        instruksi = [
            "W/S: Rotasi X || A/D: Rotasi Y || Q/E: Rotasi Z",
            "Z/X: Skala (Zoom In/Out)",
            "Arrow Keys: Translasi (Pindah Posisi)"
        ]

        y_offset = 10
        for baris in instruksi:
            teks = self.font.render(baris, True, HITAM)
            layar.blit(teks, (10, y_offset))
            y_offset += 20

        # Meampilkan Header Koordinat
        y_offset += 20
        header = self.font.render("KOORDINAT TITIK: ", True, HITAM)
        layar.blit(header, (10, y_offset))
        y_offset += 20

        # Menampilkan Daftar Koordinat Titik
        for i, v in enumerate(self.transformed_vertices):
            x_layar = int(v[0] + TITIK_PUSAT[0])
            y_layar = int(v[1] + TITIK_PUSAT[1])
            z_layar = int(v[2])

            label = self.labels[i] if i < len(self.labels) else str(i)

            # Format string agar rapi
            info = f"{label}({x_layar:>4}, {y_layar:>4}, {z_layar:>4})"
            teks_koordinat = self.font.render(info, True, HITAM)
            layar.blit(teks_koordinat, (10, y_offset))
            y_offset += 15
```

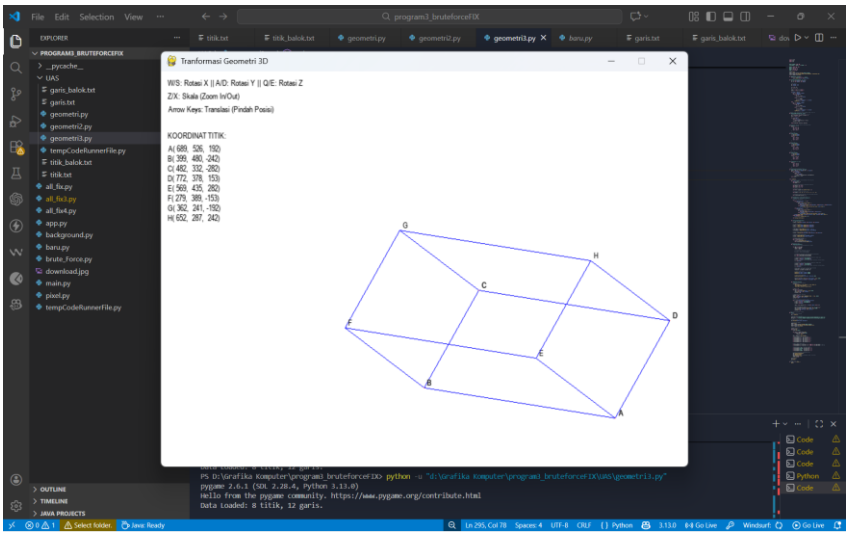
Saya menambahkan informasi untuk konsol gerakannya serta informasi koordinat sekarang.



Hasilnya.

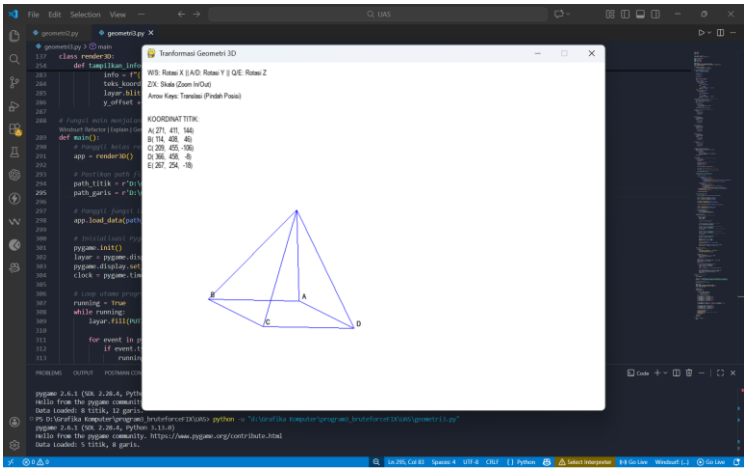
2. Penambahan file koordinat baru untuk memuat objek lain selain kubus.
   Selain kubus, saya mencoba menambahkan beberapa bentuk lain seperti berikut:
   titik_balok.txt:
   -150,-50,-50
   150,-50,-50
   150,50,-50
   -150,50,-50
   -150,-50,50
   150,-50,50
   150,50,50
   -150,50,50

   garis_balok.txt:
   0,1
   1,2
   2,3
   3,0
   4,5
   5,6
   6,7
   7,4
   0,4
   1,5
   2,6
   3,7



   titik_limasSegi4.txt:
   -50,50,-50
   50,50,-50
   50,50,50
   -50,50,50
   0,-50,0

garis_limasSegi4.txt:
0,1
1,2
2,3
3,0
0,4
1,4
2,4
3,4

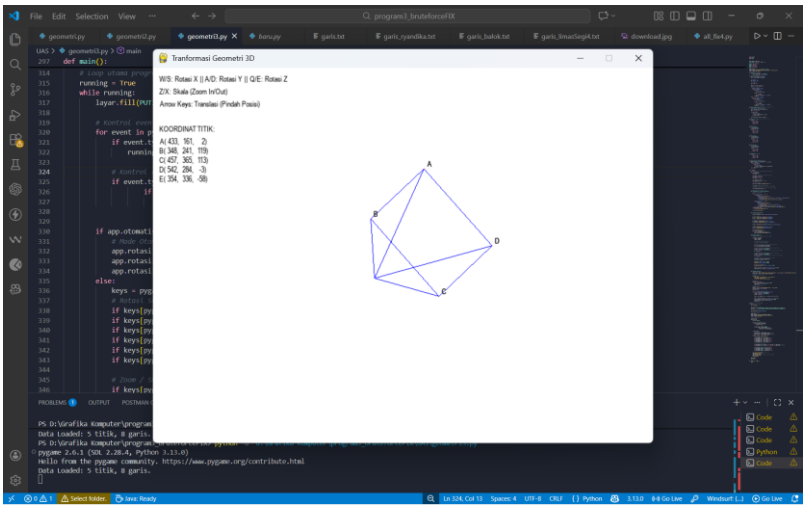


Hasilnya.

4. Penambahan gerak animasi otomatis (toggle).

```
297  def main():
314      # Loop utama program
315      running = True
316      while running:
317          layar.fill(PUTIH)
318
319          # Kontrol event manual
320          for event in pygame.event.get():
321              if event.type == pygame.QUIT:
322                  running = False
323
324              # Kontrol event toggle otomatis
325              if event.type == pygame.KEYDOWN:
326                      if event.key == pygame.K_SPACE:
327                          app.otomatis = not app.otomatis
328
329
330          if app.otomatis:
331              # Mode Otomatis: Putar terus
332              app.rotasi[0] += app.kecepatan_auto[0] # Putar X
333              app.rotasi[1] += app.kecepatan_auto[1] # Putar Y
334              app.rotasi[2] += app.kecepatan_auto[2] # Putar Z
335          else:
336              keys = pygame.key.get_pressed()
337              # Rotasi Sumbu x dan y (Huruf WASD)
338              if keys[pygame.K_w]:  app.rotasi[0] -= 2 # x
339              if keys[pygame.K_s]:  app.rotasi[0] += 2
340              if keys[pygame.K_a]:  app.rotasi[1] -= 2 # y
341              if keys[pygame.K_d]:  app.rotasi[1] += 2
342              if keys[pygame.K_q]:  app.rotasi[2] -= 1 # z
343              if keys[pygame.K_e]:  app.rotasi[2] += 1
344
345              # Zoom / Skala (Huruf Z/X)
346              if keys[pygame.K_z]: app.skala = [s + 0.05 for s in app.skala] # zoom in
347              if keys[pygame.K_x]: app.skala = [s - 0.05 for s in app.skala] # zoom out
348
349              # Peripindahan posisi (translasi) (Panah)
350              if keys[pygame.K_LEFT]:  app.posisi[0] -= 2
351              if keys[pygame.K_RIGHT]: app.posisi[0] += 2
352              if keys[pygame.K_UP]:    app.posisi[1] -= 2
353              if keys[pygame.K_DOWN]:  app.posisi[1] += 2
354
```

Menambahkan tekan tombol spasi pada loop utama (main) untuk mengaktifkan toggle transformasi geometri otomatis.



Hasilnya.

BAB III
PENUTUP

3.1 Kesimpulan

Berdasarkan hasil praktikum dan implementasi kode yang telah dilaksanakan, dapat disimpulkan bahwa pembangunan simulasi objek 3 dimensi sederhana berhasil diwujudkan dengan menerapkan konsep dasar grafika komputer. Representasi objek dilakukan menggunakan struktur data *wireframe* yang mendefinisikan himpunan titik (*vertices*) dan garis (*edges*), yang memungkinkan manipulasi data bentuk secara fleksibel melalui pembacaan *file* eksternal. Dari sisi komputasi matematis, penggunaan matriks homogen 4x4 terbukti sangat efektif dalam menangani operasi transformasi geometri; teknik matriks komposit memungkinkan penggabungan operasi rotasi, skala, dan translasi menjadi satu kesatuan kalkulasi yang efisien. Lebih lanjut, visualisasi objek ke layar monitor 2 dimensi dicapai melalui metode proyeksi ortogonal sederhana dengan memanfaatkan algoritma garis Bresenham, sementara interaktivitas program dibangun melalui mekanisme *event loop* yang responsif terhadap *input* pengguna, sehingga menghasilkan animasi transformasi yang berjalan secara *real-time*.

3.2 Saran

Untuk pengembangan program ini di masa mendatang agar lebih mendekati standar aplikasi grafis modern, terdapat beberapa aspek yang perlu ditingkatkan. Pertama, penerapan rumus proyeksi perspektif sangat disarankan untuk menggantikan proyeksi ortogonal saat ini, hal ini bertujuan memberikan efek kedalaman visual (*depth perception*) di mana objek yang berada lebih jauh akan terlihat lebih kecil secara proporsional. Selain itu, pengembangan metode *rendering* dari sekadar kerangka garis (*wireframe*) menjadi bentuk *solid* yang memiliki permukaan warna perlu dilakukan, misalnya dengan menerapkan *Painter's Algorithm* untuk menangani tumpang tindih sisi objek. Terakhir, dari sisi performa komputasi, penggunaan pustaka numerik khusus seperti NumPy disarankan untuk menangani operasi aritmatika matriks apabila jumlah titik objek bertambah signifikan, guna menjaga efisiensi dan kecepatan pemrosesan data.

3.3 Link Youtube dan Github

Github: https://github.com/IGedeRyandikaPramudiaWardana/Uas_GrafikaKomputer.git

Youtube: https://youtu.be/Hw7Tju-dz0Y