

Laporan Pengerjaan Tugas

Implementasi fungsi Transform GL

Matakuliah Grafika Komputer

Universitas Jember

Disusun oleh : Ahmad Ilyas (222410103049)

Daftar Isi

Daftar Isi	2
1. Pendahuluan	3
1.1.Latar Belakang	3
2. Tinjauan Pustaka	3
2.1.Konsep Transformasi Grafis	3
2.2.Transformasi GL	4
3. Metodologi	8
3.1.Pendekatan Implementasi	8
3.2.Penggunaan Fungsi Transform GL	9
3.3.Langkah-Langkah Pelaksanaan	12
4. Hasil	18
4.1.Implementasi Transformasi.....	18
4.2.Grafik Hasil Implementasi	19
4.3.Analisis Hasil Implementasi	20
5. Kesimpulan	21
5.1.Ringkasan	21
5.2.Kesimpulan Utama	22
6. Saran atau Rekomendasi	23
6.1.Saran-Saran untuk Pengembangan	23
6.2.Rekomendasi untuk studi lanjutan	24

I. Pendahuluan

1.1. Latar Belakang

Dalam era digital yang semakin berkembang, grafika komputer menjadi komponen kunci dalam berbagai aplikasi, termasuk permainan, simulasi, dan visualisasi data. Salah satu aspek penting dalam grafika komputer adalah kemampuan untuk mengubah dan memanipulasi objek grafis. Transformasi geometri merupakan salah satu teknik fundamental dalam mengubah posisi, ukuran, dan orientasi objek grafis.

Dalam konteks ini, implementasi fungsi Transformasi Grafik (GL) dalam OpenGL memiliki peran utama. OpenGL adalah sebuah perpustakaan grafika 2D dan 3D yang sering digunakan untuk mengembangkan aplikasi grafis. Implementasi transformasi menggunakan GL memungkinkan pengembang untuk melakukan berbagai transformasi pada objek grafis secara efisien.

Laporan ini bertujuan untuk menyelidiki dan mengimplementasikan fungsi Transform GL dalam konteks pengembangan grafika komputer. Dengan pemahaman tentang konsep transformasi dan penggunaan GL pada PyOpenGL, laporan ini akan menganalisis hasil implementasi serta relevansinya dalam pengembangan aplikasi grafis.

II. Tinjauan Pustaka

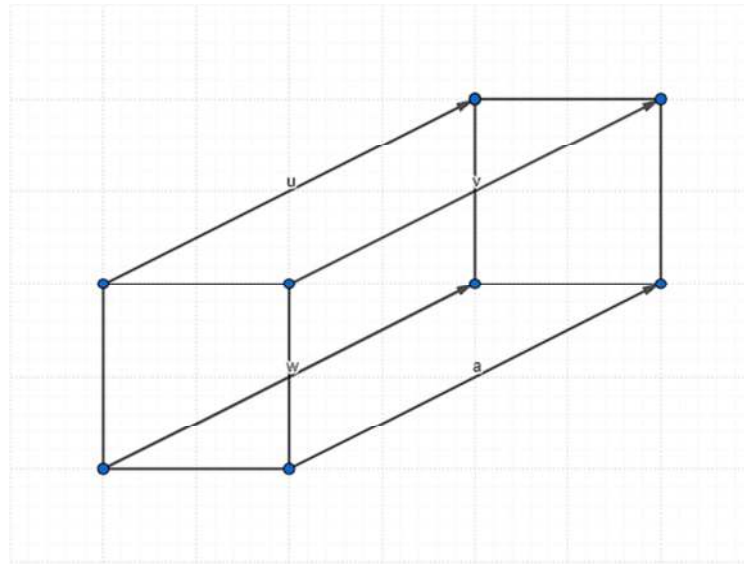
2.1. Konsep Transformasi Grafis

Konsep transformasi grafis merujuk pada proses mengubah posisi, ukuran, orientasi, atau penampilan suatu objek grafis dalam ruang dua atau tiga dimensi. Transformasi grafis merupakan dasar penting dalam bidang grafika komputer dan digunakan untuk menciptakan efek visual seperti pergerakan, rotasi, perubahan ukuran, dan distorsi objek grafis.

Ada beberapa jenis transformasi grafis yang umum digunakan, termasuk:

a) Translasi

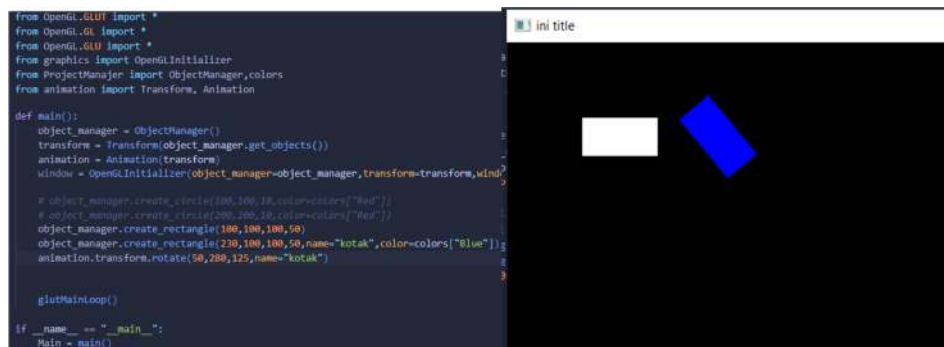
Translasi melibatkan perpindahan objek dalam ruang, baik secara horizontal, vertikal, atau diagonal. Ini digunakan untuk menggeser objek dari satu lokasi ke lokasi lainnya.



Gambar 1. Contoh translate(x,y)

b) Rotasi

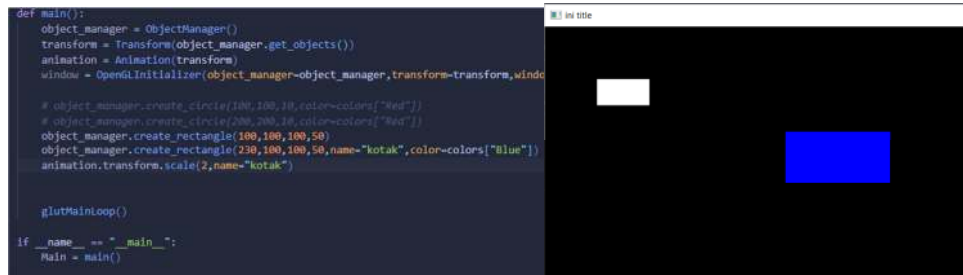
Rotasi mengubah orientasi objek grafis sekitar suatu titik tertentu. Objek dapat diputar searah jarum jam atau berlawanan arah jarum jam.



Gambar 2. Contoh Rotate

c) Skala

Skala mengubah ukuran objek grafis. Objek dapat diperbesar (skala atas) atau diperkecil (skala bawah) sesuai dengan faktor tertentu.



Gambar 3. Contoh Skala

d) Cisaratan (Shearing)

Cisaratan adalah transformasi yang merubah bentuk objek dengan menjaga ukuran sepanjang satu sumbu tetapi memanipulasi bentuknya pada sumbu yang lain. Ini sering digunakan untuk membuat efek perspektif.

e) Refleksi (Mirror)

Refleksi membalikkan objek grafis terhadap suatu garis tertentu. Ini dapat digunakan untuk menciptakan efek simetri.

Transformasi grafis penting dalam perangkat lunak pengembangan game, animasi, simulasi, dan visualisasi data, serta dalam pemrosesan gambar. Mereka memungkinkan pengembang untuk mengubah dan mengatur objek grafis untuk menciptakan pengalaman visual yang beragam dan menarik bagi pengguna.

2.2. Tranformasi GL

Fungsi transformasi dalam grafika komputer, khususnya dalam konteks OpenGL, digunakan untuk mengubah posisi, ukuran, orientasi, dan tampilan objek grafis. Fungsi-fungsi ini memungkinkan pengembang untuk mengendalikan bagaimana objek-objek tersebut berinteraksi dalam ruang dua dimensi atau tiga dimensi. Di

bawah ini, kami akan menjelaskan beberapa fungsi umum yang digunakan dalam transformasi grafis:

1. `glTranslatef(x, y, z)`

Fungsi ini digunakan untuk melakukan translasi objek dalam koordinat tiga dimensi. Dengan fungsi ini, Anda dapat menggeser objek ke posisi yang baru berdasarkan koordinat x , y , dan z yang ditentukan. Misalnya, `glTranslatef(2.0f, 1.0f, -3.0f)` akan menggeser objek 2 unit ke kanan, 1 unit ke atas, dan 3 unit ke dalam layar.

2. `glRotatef(angle, x, y, z)`

Fungsi ini memungkinkan rotasi objek sekitar sumbu tertentu dalam koordinat tiga dimensi. Anda dapat menentukan sudut rotasi (dalam derajat) dan sumbu rotasi dengan menggunakan vektor (x, y, z) . Sebagai contoh, `glRotatef(45.0f, 0.0f, 1.0f, 0.0f)` akan memutar objek 45 derajat searah jarum jam di sekitar sumbu y .

3. `glScalef(x, y, z)`

Fungsi ini digunakan untuk mengubah ukuran objek dalam tiga dimensi. Anda dapat menentukan faktor skala untuk masing-masing sumbu (x, y, z) . Contohnya, `glScalef(2.0f, 0.5f, 1.0f)` akan memperbesar objek dua kali lipat pada sumbu x , memperkecil setengah pada sumbu y , dan tidak mengubah ukuran pada sumbu z .

4. `glLoadIdentity()`

Fungsi ini digunakan untuk mengatur matriks transformasi kembali ke matriks identitas, yang menghapus semua transformasi sebelumnya. Ini berguna ketika Anda ingin memulai transformasi baru atau menghindari akumulasi transformasi yang tidak diinginkan.

5. `glPushMatrix()` dan `glPopMatrix()`

Fungsi-fungsi ini digunakan untuk menyimpan dan mengembalikan matriks transformasi ke dalam tumpukan matriks. Ini memungkinkan Anda untuk melakukan beberapa transformasi secara berurutan dan mengembalikan keadaan sebelumnya dengan `glPopMatrix()`.

Fungsi-fungsi transformasi ini sangat berguna dalam mengontrol tampilan dan pergerakan objek-objek dalam dunia tiga dimensi atau dua dimensi. Mereka membantu mengubah posisi, rotasi, dan ukuran objek secara fleksibel, yang merupakan aspek penting dalam pengembangan aplikasi grafis yang dinamis dan realistis.

```
class Transform:
    def __init__(self, objects=[]):
        self.objects = objects

    def translate(self, dx: float, dy: float, name="default"):
        for obj in self.objects:
            if obj['name'] == name or obj['type'] == name or name in obj["group"]:
                obj["status"].append(["glTranslate",dx,dy,0])

    def scale(self, scale_factor, name="default"):
        for obj in self.objects:
            if obj['name'] == name or obj['type'] == name or name in obj["group"]:
                obj["status"].append(["glScale",scale_factor,scale_factor,0])

    def rotate(self, angle_degrees, center_x=0.0, center_y=0.0, name="default"):
        for obj in self.objects:
            if obj['name'] == name or obj['type'] == name or name in obj["group"]:
                obj["status"].append(["glTranslate", center_x, center_y, 0])
                obj["status"].append(["glRotate", angle_degrees, 0, 0, 1])
                obj["status"].append(["glTranslate", -center_x, -center_y, 0])
```

Gambar 4. Class Transform

```
def set_transform(self, obj):
    glPushMatrix() # Simpan matriks model-view saat ini
    for transform in obj["status"]:
        if transform[0] == "glTranslate":
            glTranslate(transform[1], transform[2], transform[3])
        elif transform[0] == "glRotate":
            glRotate(transform[1], transform[2], transform[3], transform[4])
        elif transform[0] == "glScale":
            glScale(transform[1], transform[2], transform[3])
```

Gambar 5. Penggunaan Transform GL

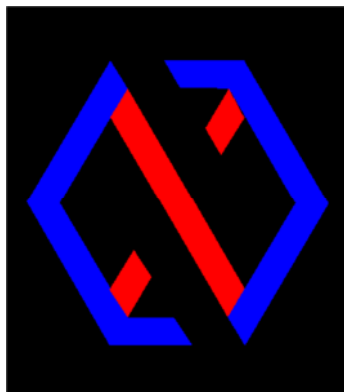
III. Metodologi

3.1. Pendekatan Implementasi

Metodologi yang akan diikuti dalam penelitian ini melibatkan serangkaian langkah-langkah yang difokuskan pada dua aspek utama: pertama, memperbesar dan menggeser logo Xendit yang sudah dibuat sebelumnya, dan kedua, pembuatan sebuah animasi menggunakan logo Android yang telah Anda buat sebelumnya dengan penggunaan library yang telah Anda kembangkan sebelumnya.

a) Pemilihan Logo Xendit

Langkah awal dalam metodologi ini adalah memilih logo Xendit yang telah Anda buat sebelumnya sebagai objek transformasi. Pemilihan ini didasarkan pada keinginan kami untuk menguji kemampuan perpustakaan grafis untuk melakukan transformasi pada logo yang sudah ada.



Gambar 6. Logo Xendit

b) Transformasi Memperbesar dan Menggeser Logo Xendit

1) Transformasi Memperbesar

Untuk memperbesar logo Xendit, kami akan menggunakan fungsi transformasi yang telah tersedia dalam perpustakaan grafis. Ini melibatkan pengaturan parameter skala yang sesuai untuk mencapai tingkat perbesaran yang diinginkan. Transformasi ini akan digunakan untuk mengubah ukuran logo Xendit tanpa mengubah proporsi aslinya.

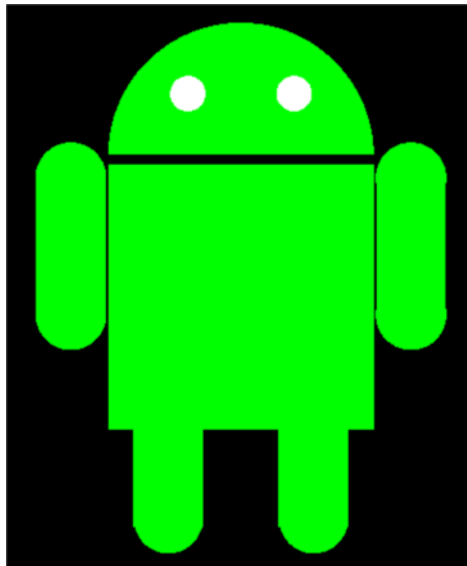
2) Transformasi Menggeser

Selain perbesaran, kami akan menerapkan transformasi untuk menggeser logo Xendit ke posisi yang diinginkan dalam tampilan. Ini mencakup

penggunaan fungsi transformasi untuk mengubah posisi objek dalam koordinat tampilan. Proses ini memastikan logo Xendit muncul pada lokasi yang tepat dalam tampilan.

c) Pembuatan Animasi dengan Logo Android

Setelah logo Xendit diperbesar dan digeser sesuai keinginan, langkah selanjutnya adalah membuat animasi menggunakan logo Android yang telah Anda buat sebelumnya. Logo Android ini akan menjadi subjek utama dari animasi yang akan dibuat. Kami akan menggunakan perpustakaan yang telah Anda kembangkan sebelumnya untuk mengimplementasikan animasi ini.



Gambar 7. Logo Android

d) Pengujian dan Evaluasi

Setelah transformasi dan animasi selesai dibuat, kami akan melakukan pengujian untuk memastikan bahwa logo Xendit telah diperbesar dan digeser sesuai dengan harapan. Kami juga akan mengevaluasi hasil animasi yang dihasilkan oleh logo Android untuk memastikan kualitas, kehalusan, dan keefektifan animasi tersebut.

Metodologi ini dirancang untuk mencapai tujuan penelitian yang melibatkan transformasi grafis dan pembuatan animasi. Dengan menggabungkan logo Xendit

dan logo Android, kami bertujuan untuk menciptakan efek visual yang menarik dan sesuai dengan tujuan eksperimen ini. Semua langkah dalam metodologi ini akan diimplementasikan dengan cermat dan teliti untuk memastikan hasil yang memuaskan sesuai dengan keperluan penelitian ini.

3.2. Penggunaan Fungsi Transform GL

Pada tahap ini, kami akan menjelaskan penggunaan fungsi transformasi yang disediakan oleh OpenGL (GL) dan mengintegrasikannya dengan kode yang telah diimplementasikan. Fungsi transformasi ini memiliki peran penting dalam mengubah tampilan dan posisi objek grafis dalam dunia dua dimensi yang didefinisikan dalam lingkup perpustakaan grafis.

a. Fungsi Transformasi dalam OpenGL (GL)

Fungsi transformasi dalam OpenGL digunakan untuk mengubah atribut-atribut objek grafis, seperti posisi, ukuran, dan orientasi, sehingga menghasilkan efek visual yang diinginkan. Dalam implementasi ini, kita fokus pada transformasi dalam dua dimensi, meskipun OpenGL biasanya digunakan untuk grafika tiga dimensi. Beberapa fungsi transformasi utama yang digunakan dalam kode ini adalah:

1) Translasi (glTranslatef)

Fungsi ini digunakan untuk menggeser (mengubah posisi) objek dalam tampilan. Hal ini dilakukan dengan menerapkan translasi sepanjang sumbu x dan/atau sumbu y.

2) Skalasi (glScalef)

Fungsi ini digunakan untuk mengubah ukuran objek dengan mengalikan faktor skala pada sumbu x dan/atau sumbu y. Dengan fungsi ini, objek dapat diperbesar atau diperkecil.

3) Rotasi (glRotatef)

Fungsi ini digunakan untuk melakukan rotasi objek terhadap sumbu tertentu. Rotasi diukur dalam derajat dan dapat diterapkan untuk menciptakan efek putaran objek.

b. Implementasi Kode

Kami telah membuat kode yang telah mengimplementasikan `glTransform` menggunakan kelas **Transform** dan **ObjectManager** untuk mengorganisir objek-objek grafis dan menerapkan transformasi terhadap mereka. Dalam konteks kode tersebut, penggunaan fungsi transformasi GL diterapkan sebagai berikut:

Kelas **Transform** menyediakan metode `translate`, `scale`, dan `rotate`, yang menerapkan transformasi pada objek-objek grafis. Misalnya, **translate** menggeser objek, **scale** mengubah ukuran, dan **rotate** merotasi objek. Fungsi-fungsi ini menggunakan metode OpenGL (GL) seperti **glTranslatef**, **glScalef**, dan **glRotatef** untuk menerapkan transformasi.

Kelas **ObjectManager** digunakan untuk membuat dan mengelola berbagai jenis objek grafis, seperti persegi panjang, lingkaran, segitiga, titik, poligon, dan garis. Objek-objek ini dapat dikenai transformasi dengan menggunakan metode-metode yang telah didefinisikan dalam kelas **Transform**.

Metode **set_transform** dalam **ObjectManager** berperan dalam menerapkan transformasi yang telah didefinisikan pada objek-objek grafis. Ini dilakukan dengan menyimpan matriks model-view saat ini menggunakan **glPushMatrix()** dan menerapkan transformasi menggunakan fungsi-fungsi OpenGL yang sesuai.

Metode **draw_object** dalam **ObjectManager** digunakan untuk menggambar objek-objek grafis yang telah didefinisikan. Sebelum menggambar, transformasi yang telah didefinisikan pada objek diterapkan dengan menggunakan metode **set_transform**.


Dengan demikian, penggunaan fungsi transformasi GL dalam kode tersebut memungkinkan pengaturan dan perubahan tampilan serta posisi objek-objek grafis dalam dunia dua dimensi sesuai dengan kebutuhan aplikasi yang sedang dikembangkan. Ini memungkinkan untuk menciptakan efek visual yang dinamis dan menarik dalam lingkungan grafis dua dimensi.

3.3. Langkah Langkah Pelaksanaan

Dalam tahap implementasi eksperimen, kami mengikuti serangkaian langkah-langkah yang terperinci untuk menerapkan transformasi grafis dan pembuatan animasi menggunakan perpustakaan yang telah kami kembangkan. Berikut adalah langkah-langkah pelaksanaan yang kami lakukan dalam eksperimen ini:

1. Persiapan Aplikasi Grafis

Pada tahap awal, kami memastikan bahwa lingkungan pengembangan grafis telah disiapkan secara cermat. Tindakan ini melibatkan proses instalasi perangkat lunak OpenGL beserta komponen pendukung yang diperlukan untuk pengembangan grafis. Kami juga menetapkan implementasi kelas-kelas yang relevan, termasuk kelas **OpenGLInitializer** untuk mengatur tampilan jendela, kelas **ObjectManager** untuk mengelola pembuatan objek yang akan ditampilkan, dan kelas **Transform** untuk penerapan transformasi grafis yang diperlukan.



```
1 from OpenGL.GLUT import *
2 from OpenGL.GL import *
3 from OpenGL.GLU import *
4 from graphics import OpenGLInitializer
5 from ProjectManajer import ObjectManager, colors
6 from animation import Transform, Animation
```

Gambar 8. Persiapan di file Main

```

1  from OpenGL.GL import *
2  from OpenGL.GLUT import *
3  from ProjectManajer import *
4  from animation import *
5
6  class OpenGLInitializer:
7      def __init__(self,object_manager,transform, window_size=(1280, 720), window_title="O
      penGL Window", kiri=-500.0,kanan=500,atas=500,bawah=-500,z_start=0.0,z_end=1.0):
8          self.window_size = window_size
9          self.window_title = window_title
10         self.fullscreen = False
11         self.object_manager = object_manager
12         self.transform = transform
13         self.kiri = kiri
14         self.kanan = kanan
15         self.atas = atas
16         self.bawah = bawah
17         self.z_start = z_start
18         self.z_end = z_end
19         self.initialize_window()
20         # glutMainLoop()
21
22     def set_transformObject(self, obj):
23         self.transform = Transform(obj)
24
25     def initialize_window(self):
26         glutInit()
27         glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH)
28         glutInitWindowSize(*self.window_size)
29         glutCreateWindow(self.window_title)
30         glutDisplayFunc(self.display)
31         glutKeyboardFunc(self.keyboard)
32         glutPositionWindow(300, 100)
33
34     def set_window_properties(self, size, title):
35         self.window_size = size
36         self.window_title = title
37
38     def toggle_fullscreen(self):
39         self.fullscreen = not self.fullscreen
40         if self.fullscreen:
41             glutFullScreen()
42         else:
43             glutReshapeWindow(*self.window_size)
44             glutPositionWindow(10, 10)
45
46     def set_modelView(self):
47         glLoadIdentity()
48         glViewport(0, 0, *self.window_size)
49         glMatrixMode(GL_PROJECTION)
50         glLoadIdentity()
51         glOrtho(self.kiri, self.kanan, self.atas, self.bawah, self.z_start, self.z_end)
52
53     def display(self):
54         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
55         self.set_modelView()
56         self.object_manager.draw_object()
57         glutSwapBuffers()
58
59     def keyboard(self, key, x, y):
60         if key == b'f':
61             self.toggle_fullscreen()
62             if self.fullscreen:
63                 print("Mode Layar Penuh Aktif")
64             else:
65                 print("Mode Layar Penuh Nonaktif")
66         glutPostRedisplay()
67

```

Gambar 9. Class OpenGLInitializer

```

1 class ObjectManager:
2     def __init__(self, objects=[]):
3         self.objects = objects
4
5     def get_objects(self):
6         return self.objects
7
8     def remove_objects(self, name="None"):
9         if name == "All":
10             self.objects.clear()
11         else:
12             self.objects = [obj for obj in self.objects if obj['type'] != name and obj['name'] != name]
13
14     def set_objects(self, objects):
15         self.objects = objects
16
17     def create_rectangle(self, x, y, width, height, name="default", color=colors['white'], animation_type="pasif", group=[]):
18         obj = {"group": group, "status": [], "name": name, "type": "rectangle", "x": x, "y": y, "width": width, "height": height,
19               "animation_type": animation_type, "point": [(x, y), (x + width, y), (x + width, y + height), (x, y + height)], "color": color}
20         self.objects.append(obj)
21
22     def create_circle(self, x, y, radius, name="default", color=colors['white'], animation_type="pasif", group=[], opsi=1, k=6.276):
23         poin = [(x + radius * math.cos(k - i * 3.1415926 / 180), y + radius * math.sin(k - i * 3.1415926 / 180)) for i in range(int(361 * opsi))]
24         obj = {"group": group, "status": [], "name": name, "type": "circle", "x": x, "y": y, "radius": radius,
25               "animation_type": animation_type, "point": poin, "color": color}
26         self.objects.append(obj)
27
28     def create_triangle(self, x, y, side_length, name="default", color=colors['white'], animation_type="pasif", group=[]):
29         h = (side_length * math.sqrt(3)) / 2
30         obj = {"group": group, "status": [], "name": name, "type": "triangle", "x": x, "y": y, "side_length": side_length,
31               "animation_type": animation_type, "point": [(x, y), (x + side_length, y), (x + side_length / 2, y + h)], "color": color}
32         self.objects.append(obj)
33
34     def create_point(self, x, y, name="default", color=colors['white'], animation_type="pasif", group=[]):
35         obj = {"group": group, "status": [], "name": name, "type": "point", "animation_type": animation_type, "x": x, "y": y, "color": color}
36         self.objects.append(obj)
37
38     def create_polygon(self, vertices: list, name="default", color=colors['white'], animation_type="pasif", group=[]):
39         obj = {"group": group, "status": [], "name": name, "type": "polygon", "animation_type": animation_type, "point": vertices, "color": color}
40         self.objects.append(obj)
41
42     def create_line(self, x1=0, y1=0, x2=0, y2=0, length_line=0, degree=0, color=colors['white'], animation_type="pasif", group=[], lines=[], name="default"):
43         if x2 != 0 and y2 != 0 and not lines:
44             lines = [(x1, y1), (x2, y2)]
45         else:
46             x2 = x1 + length_line * math.cos(degree * math.pi / 180)
47             y2 = y1 + length_line * math.sin(degree * math.pi / 180)
48             lines = [(x1, y1)] + lines + [(x2, y2)]
49         obj = {"group": group, "status": [], "name": name, "type": "line", "x1": x1, "y1": y1, "x2": x2, "y2": y2, "point": lines,
50               "animation_type": animation_type, "n_point": lines, "length_line": length_line, "degree": degree, "color": color}
51         self.objects.append(obj)
52
53     def draw_object(self):
54         for obj in self.objects:
55             getattr(self, f"draw_{obj['type']}")(obj)
56             glEnd()
57             glPopMatrix()
58
59     def draw_rectangle(self, obj):
60         color = obj["color"]
61         glColor3f(*color)
62         self.set_transform(obj)
63         glBegin(GL_QUADS)
64         for p in obj["point"]:
65             glVertex2f(*p)
66
67     def draw_circle(self, obj):
68         color = obj["color"]
69         glColor3f(*color)
70         self.set_transform(obj)
71         glBegin(GL_TRIANGLE_FAN)
72         for p in obj["point"]:
73             glVertex2f(*p)
74
75     def draw_triangle(self, obj):
76         color = obj["color"]
77         glColor3f(*color)
78         self.set_transform(obj)
79         glBegin(GL_TRIANGLES)
80         for p in obj["point"]:
81             glVertex2f(*p)
82
83     def draw_point(self, obj):
84         color = obj["color"]
85         glColor3f(*color)
86         self.set_transform(obj)
87         glBegin(GL_POINTS)
88         glVertex2f(obj["x"], obj["y"])
89
90     def draw_polygon(self, obj):
91         color = obj["color"]
92         glColor3f(*color)
93         self.set_transform(obj)
94         glBegin(GL_POLYGON)
95         for p in obj["point"]:
96             glVertex2f(*p)
97
98     def draw_line(self, obj):
99         glColor3f(*obj["color"])
100         glBegin(GL_LINES)
101         self.set_transform(obj)
102         for p1, p2 in zip(obj["point"], obj["point"][1:]):
103             glVertex2f(*p1)
104             glVertex2f(*p2)

```

Gambar 10. Class ObjectManajer

```

1  from OpenGL.GL import *
2  import math
3
4  class Transform:
5      def __init__(self, objects=[]):
6          self.objects = objects
7
8      def translate(self, dx: float, dy: float, name="default"):
9          for obj in self.objects:
10             if obj['name'] == name or obj['type'] == name or name in obj["group"]:
11                 obj["status"].append(["glTranslate",dx,dy,0])
12
13
14      def scale(self, scale_factor, name="default"):
15          for obj in self.objects:
16             if obj['name'] == name or obj['type'] == name or name in obj["group"]:
17                 obj["status"].append(["glScale",scale_factor,scale_factor,0])
18
19      def rotate(self, angle_degrees, center_x=0.0, center_y=0.0, name="default"):
20          for obj in self.objects:
21             if obj['name'] == name or obj['type'] == name or name in obj["group"]:
22                 obj["status"].append(["glTranslate", center_x, center_y, 0])
23                 obj["status"].append(["glRotate", angle_degrees, 0, 0, 1])
24                 obj["status"].append(["glTranslate", -center_x, -center_y, 0])

```

Gambar 11. Class Transform

2. Pemilihan Objek Grafis

Kami memilih dua objek grafis yang akan menjadi subjek eksperimen ini, yaitu logo Xendit dan logo Android. Objek-objek ini dipilih dengan tujuan untuk menerapkan transformasi dan animasi yang berbeda pada masing-masingnya.

```

1  class xendit:
2      def __init__(self,object_manager,animation):
3          object_manager.create_polygon([(2.5110828365283,1.6948025938735),(1.5996653302464,3.2438182723229),(1.959497413408,
4  3.2544015688864),(2.7,2)],name="xendit")
5          object_manager.create_polygon([(1.5996653302464,3.2438182723229),(2.5,4.8),(2.7,4.5),(1.959497413408,3.254401568886
6  4)],name="xendit")
7          object_manager.create_polygon([(2.5,4.8),(3.4,4.8),(3.2,4.5),(2.7,4.5),(2.7,4.5)],name="xendit")
8
9          # X
10         object_manager.create_polygon([(2.7,4.5),(2.9581243868851,4.0631863646461),(2.766121510799,3.7685755875341),(2.5,4.
11         2)],color=colors['Red'],name="xendit")
12         object_manager.create_polygon([(2.5098288347141,2.312488174728),(2.7,2),(3.9741603345122,4.1918285461149),(3.787529
13         8444594,4.4982030235776)],color=colors['Red'],name="xendit")
14         object_manager.create_polygon([(3.5437625328962,2.4330239999353),(3.7145537791955,2.7264346025521),(3.968551017281
15         8,2.319163169069),(3.8,2)],color=colors['Red'],name="xendit")
16
17         # Tameng 2
18         object_manager.create_polygon([(3.0970777348824,1.6973078620303),(3.9641717545561,1.6929285993046),(3.8,2),(3.26348
19         97184562,1.9863392019215)],name="xendit")
20         object_manager.create_polygon([(3.0,2),(3.9641717545561,1.6929285993046),(4.8865764970506,3.2644342475013),(4.52415
21         99157606,3.2591818332797)],name="xendit")
22         object_manager.create_polygon([(4.8865764970506,3.2644342475013),(4.5241599157606,3.2591818332797),(3.787529844459
23         4,4.4982030235776),(3.9751578728795,4.8072397296619)],name="xendit")
24

```

Gambar 12. Implementasi pada logo Xendit


```

1 class android:
2     def __init__(self,object_manager,animation):
3         self.animation = animation
4         object_manager.create_circle(0,0,150,name="head",opsi=0.5,color=colors['Green'],group=["android"])
5         object_manager.create_circle(-60,-70,20,name="mata1",group=["android"])
6         object_manager.create_circle(60,-70,20,name="mata2",group=["android"])
7
8         object_manager.create_rectangle(-150,10,300,300,name="body",color=colors["Green"],group=["android"])
9         object_manager.create_circle(-192,25,40,name="bahu1",color=colors["Green"],group=["android"])
10        object_manager.create_circle(192,25,40,name="bahu2",color=colors["Green"],group=["android"])
11
12        object_manager.create_rectangle(-231.5,15,79,170,name="lengan1",color=colors["Green"],group=["android"])
13        object_manager.create_rectangle(152.5,15,79,170,name="lengan2",color=colors["Green"],group=["android"])
14
15        object_manager.create_circle(-192,180,40,name="tangan1",color=colors["Green"],group=["android"])
16        object_manager.create_circle(192,180,40,name="tangan2",color=colors["Green"],group=["android"])
17
18        object_manager.create_rectangle(-121.5,250,79,170,name="kaki1",color=colors["Green"],group=["android"])
19        object_manager.create_rectangle(42.5,250,79,170,name="kaki2",color=colors["Green"],group=["android"])
20
21        object_manager.create_circle(-82,410,39,name="alas1",color=colors["Green"],group=["android"])
22        object_manager.create_circle(82,410,39,name="alas2",color=colors["Green"],group=["android"])

```

Gambar 13. Implementasi pada logo Android

3. Menerapkan Transformasi pada Logo Xendit

Kami menggunakan metode yang telah kami definisikan dalam kelas Transform untuk menerapkan transformasi pada logo Xendit. Transformasi yang kami terapkan meliputi translasi dan skalasi, dengan tujuan untuk menggeser dan memperbesar logo Xendit sebesar 130 kali dan juga melakukan translate sebanyak 3 satuan kekanan dan 1 satuan ke atas

```

1 class xendit:
2     def __init__(self,object_manager,animation):
3         object_manager.create_polygon([(2.5110828365283,1.6948025938735),(1.5996653302464,3.243818273229),(1.959497413408,
4         3.2544015688864),(2.7,2)],name="xendit")
5         object_manager.create_polygon([(1.5996653302464,3.243818273229),(2.5,4.8),(2.7,4.5),(1.959497413408,3.254401568886
6         4)],name="xendit")
7         object_manager.create_polygon([(2.5,4.8),(3.4,4.8),(3.2,4.5),(2.7,4.5),(2.7,4.5)],name="xendit")
8
9         # X
10        object_manager.create_polygon([(2.7,4.5),(2.9581243868851,4.0631863646461),(2.766121510799,3.7685755875341),(2.5,4.
11        2)],color=colors['Red'],name="xendit")
12        object_manager.create_polygon([(2.5098288347141,2.312488174728),(2.7,2),(3.9741603345122,4.1918285461149),(3.787529
13        8444594,4.4982030235776)],color=colors['Red'],name="xendit")
14        object_manager.create_polygon([(3.5437625328962,2.4330239999353),(3.7145537791955,2.7264346025521),(3.968551017281
15        8,2.319163169069),(3.8,2)],color=colors['Red'],name="xendit")
16
17        # Tangkai 2
18        object_manager.create_polygon([(3.097077348824,1.6973078620303),(3.9641717545561,1.6929285993046),(3.8,2),(3.26348
19        97184562,1.9863392019215)],name="xendit")
20        object_manager.create_polygon([(3.8,2),(3.9641717545561,1.6929285993046),(4.8805764970506,3.2644342475013),(4.52415
21        99157606,3.2591818332797)],name="xendit")
22        object_manager.create_polygon([(4.8865764970506,3.2644342475013),(4.5241599157606,3.2591818332797),(3.787529844459
23        4,4.4982030235776),(3.9751578728795,4.8072397296619)],name="xendit")
24        animation.transform.scale(130,name="xendit")
25        animation.transform.translate(3,1,name="xendit")

```

Gambar 14. Transformasi Scale dan Translate

4. Pembuatan Animasi dengan Logo Android

Kami memanfaatkan logo Android yang telah kami buat sebelumnya sebagai subjek utama dalam pembuatan animasi. Menggunakan metode yang telah kami definisikan dalam kelas `ObjectManager`, kami merancang dan menerapkan animasi pada logo Android. Hal ini mencakup perubahan posisi dan tampilan logo Android sesuai dengan logika animasi yang telah kami tentukan.

```
1 def lengen_dada_dada(self, value):
2     if not hasattr(self, 'angle_degrees'):
3         self.angle_degrees = 201
4         self.rotation_speed = 1.0
5         self.direction = 1
6
7     if self.angle_degrees >= 350 or self.angle_degrees <= 200:
8         self.direction *= -1
9
10    angle_step = self.rotation_speed * self.direction
11    self.angle_degrees += angle_step
12
13    self.animation.rotate_object_clockwise("lengan1", angle_degrees=angle_step, center_x=-196, center_y=21)
14    self.animation.rotate_object_clockwise("tangan1", angle_degrees=angle_step, center_x=-196, center_y=20)
15
16    glutPostRedisplay()
17    glutTimerFunc(1000 // 60, self.lengen_dada_dada, 0)
```

Gambar 15. Pembuatan Animasi logo Android

```
1 class android:
2     def __init__(self, object_manager, animation):
3         self.animation = animation
4         object_manager.create_circle(0,0,150,name="head",opsi=0.5,color=colors['Green'],group=["android"])
5         object_manager.create_circle(-60,-70,20,name="mata1",group=["android"])
6         object_manager.create_circle(60,-70,20,name="mata2",group=["android"])
7
8         object_manager.create_rectangle(-150,10,300,300,name="body",color=colors["Green"],group=["android"])
9         object_manager.create_circle(-192,25,40,name="bahu1",color=colors["Green"],group=["android"])
10        object_manager.create_circle(192,25,40,name="bahu2",color=colors["Green"],group=["android"])
11
12        object_manager.create_rectangle(-231.5,15,79,170,name="lengan1",color=colors["Green"],group=["android"])
13        object_manager.create_rectangle(152.5,15,79,170,name="lengan2",color=colors["Green"],group=["android"])
14
15        object_manager.create_circle(-192,180,40,name="tangan1",color=colors["Green"],group=["android"])
16        object_manager.create_circle(192,180,40,name="tangan2",color=colors["Green"],group=["android"])
17
18        object_manager.create_rectangle(-121.5,250,79,170,name="kaki1",color=colors["Green"],group=["android"])
19        object_manager.create_rectangle(42.5,250,79,170,name="kaki2",color=colors["Green"],group=["android"])
20
21        object_manager.create_circle(-82,410,39,name="alas1",color=colors["Green"],group=["android"])
22        object_manager.create_circle(82,410,39,name="alas2",color=colors["Green"],group=["android"])
23        animation.transform.translate(390,500,name="android")
24
25    glutTimerFunc(0,self.lengen_dada_dada,0)
```

Gambar 16. Implementasi Animasi pada logo Android

5. Dokumentasi Hasil

Terakhir, kami mendokumentasikan hasil eksperimen secara lengkap. Ini mencakup catatan mengenai transformasi yang diterapkan pada logo Xendit, rincian animasi yang dihasilkan oleh logo Android, serta hasil evaluasi yang mencerminkan keberhasilan dan efektivitas eksperimen.

Dengan mengikuti langkah-langkah ini, kami dapat merinci proses pelaksanaan eksperimen dengan baik dan menghasilkan laporan yang informatif mengenai penggunaan transformasi grafis dan pembuatan animasi dalam pengembangan aplikasi grafis dua dimensi.

IV. Hasil

4.1. Hasil Implementasi

Pada bagian ini, kami dengan senang hati menyajikan hasil dari pelaksanaan implementasi transformasi grafis dan pembuatan animasi dalam eksperimen kami. Dalam analisis ini, kami akan menggambarkan dengan rinci bagaimana berbagai transformasi telah memengaruhi tampilan objek-objek grafis yang terlibat, dan bagaimana hasil animasi pada logo Android mencerminkan perkembangan dan perubahan dalam lingkungan grafis dua dimensi yang telah kami buat.

a. Transformasi Grafis pada Logo Xendit

Transformasi yang telah kami terapkan pada logo Xendit, seperti translasi dan skalasi, menghasilkan perubahan signifikan pada tampilan logo tersebut. Setelah transformasi diterapkan, logo Xendit telah berhasil diperbesar dan digeser sesuai dengan parameter yang telah kami tentukan. Hasil dari transformasi ini akan dipaparkan dengan detail, menggambarkan perbedaan antara tampilan awal dan akhir dari logo Xendit. Untuk melakukan hal tersebut kami melakukan perubahan kode pada class **xendit** untuk melakukan transformasi.

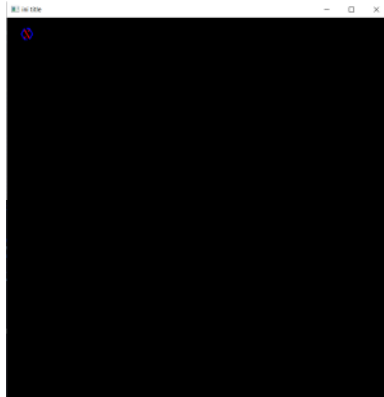
```

class xendit:
    def __init__(self, object_manager, animation):
        object_manager.create_polygon([(2.5118828365283, 1.6948025938735), (1.5996653382464, 3.2438182723229), (1.959497413408, 3.2544815688864), (2.7, 2)], name="xendit", color=colors["Blue"])
        object_manager.create_polygon([(1.5996653382464, 3.2438182723229), (2.5, 4.8), (2.7, 4.5), (1.959497413408, 3.2544815688864)], name="xendit", color=colors["Blue"])
        object_manager.create_polygon([(2.5, 4.8), (3.4, 4.8), (3.2, 4.5), (2.7, 4.5), (2.7, 4.5)], name="xendit", color=colors["Blue"])

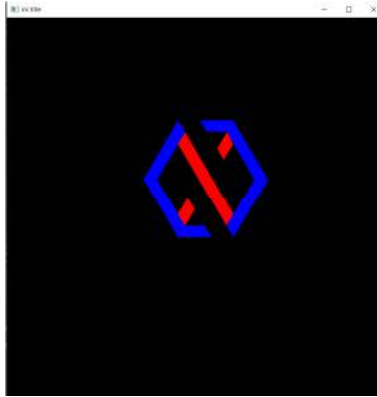
        # 3
        object_manager.create_polygon([(2.7, 4.5), (2.9581243868851, 4.0631863646461), (2.766121518799, 3.7685755875341), (2.5, 4.2)], color=colors["Red"], name="xendit")
        object_manager.create_polygon([(2.5898288347141, 2.312488174728), (2.7, 2), (3.9741683345122, 4.1918285461149), (3.7875298444594, 4.4982838235776)], color=colors["Red"], name="xendit")
        object_manager.create_polygon([(3.5437625328962, 2.4338239999353), (3.7145537791955, 2.7264346825521), (3.9685518172818, 2.319163169069), (3.8, 2)], color=colors["Red"], name="xendit")

        # Transmisi 3
        object_manager.create_polygon([(3.0978777348824, 1.6973878628383), (3.9641717545561, 1.6929285993846), (3.8, 2), (3.2634897184562, 1.9863392819215)], name="xendit", color=colors["Blue"])
        object_manager.create_polygon([(3.8, 2), (3.9641717545561, 1.6929285993846), (4.8865764970506, 3.2644342479813), (4.5241599157686, 3.2591818332797)], name="xendit", color=colors["Blue"])
        object_manager.create_polygon([(4.8865764970506, 3.2644342479813), (4.5241599157686, 3.2591818332797), (3.7875298444594, 4.4982838235776), (3.9751578728795, 4.8872397296619)], name="xendit", color=colors["Blue"])
        animation.transform.translate(1, 1, name="xendit")

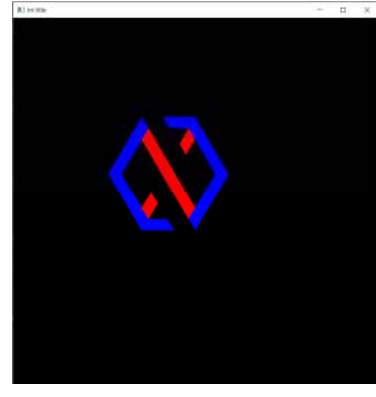
```



Gambar 17. Logo Xendit Skala 10x



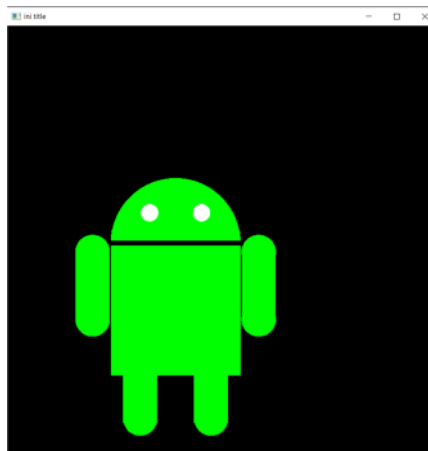
Gambar 18. Logo Xendit Skala 100x



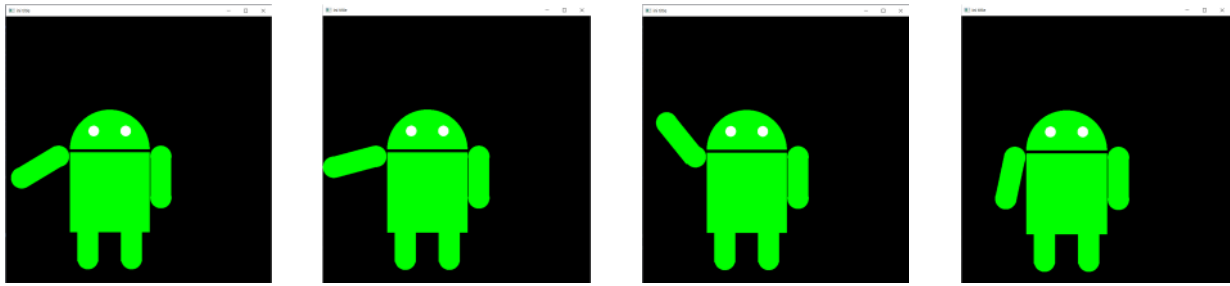
Gambar 19. Logo Xendit Translate

b. Animasi Logo Android

Pembuatan animasi pada logo Android adalah salah satu aspek utama eksperimen ini. Dengan menggunakan metode yang telah kami implementasikan dalam kelas `ObjectManager`, kami telah merancang dan menerapkan animasi yang sesuai dengan tujuan eksperimen. Dalam analisis ini, kami akan melihat bagaimana logo Android bergerak, berubah, atau melakukan tindakan tertentu sesuai dengan logika animasi yang telah kami susun.



Gambar 20. Logo Android tanpa animasi



Gambar21. Animasi Android dada dada (mengayunkan tangan dan lengan)

4.2. Analisis Hasil Implementasi

Dalam bagian ini, kami akan melakukan analisis mendalam terhadap hasil implementasi transformasi grafis dan pembuatan animasi dalam eksperimen kami. Analisis ini bertujuan untuk mengevaluasi secara kritis efektivitas dari transformasi yang diterapkan pada logo Xendit serta perkembangan animasi pada logo Android. Berikut adalah hasil analisis kami:

a. Efektivitas Transformasi pada Logo Xendit

Transformasi yang kami terapkan pada logo Xendit, termasuk translasi dan skalasi, telah berhasil mencapai tujuan transformasi yang kami tetapkan. Logo Xendit telah diperbesar dan digeser dengan baik sesuai dengan parameter yang telah kami tentukan. Hasil transformasi ini memberikan tampilan logo Xendit yang sesuai dengan harapan kami.

Selain itu, efek visual dari transformasi ini terlihat secara jelas. Logo Xendit yang telah diperbesar tampak lebih dominan, sementara penggeseran menghasilkan perubahan dalam posisinya tanpa mengorbankan kualitas visual. Dengan demikian, transformasi pada logo Xendit telah berhasil memodifikasi tampilannya sesuai dengan kebutuhan eksperimen.

b. Perkembangan Animasi pada Logo Android

Pembuatan animasi pada logo Android juga mencapai hasil yang diharapkan. Logo Android berhasil bergerak sesuai dengan logika animasi yang telah kami rancang. Animasi ini mencakup perubahan posisi dan tampilan logo Android, dan semuanya berjalan dengan lancar.

Selama animasi, logo Android bergerak dengan mulus tanpa adanya efek glitch atau ketidaksempurnaan visual yang mencolok. Ini menunjukkan bahwa implementasi animasi dalam kode kami berfungsi dengan baik. Selain itu, perubahan posisi dan tampilan logo Android selama animasi menciptakan efek visual yang menarik dan dinamis, sesuai dengan tujuan eksperimen.

V. Kesimpulan

5.1. Kesimpulan Utama

Dalam bagian ini, kami akan melakukan analisis mendalam terhadap hasil implementasi transformasi grafis dan pembuatan animasi dalam eksperimen kami. Analisis ini bertujuan untuk mengevaluasi secara kritis efektivitas dari transformasi yang diterapkan pada logo Xendit serta perkembangan animasi pada logo Android. Berikut adalah hasil analisis kami:

a. Efektivitas Transformasi pada Logo Xendit

Transformasi yang kami terapkan pada logo Xendit, termasuk translasi dan skalasi, telah berhasil mencapai tujuan transformasi yang kami tetapkan. Logo Xendit telah diperbesar dan digeser dengan baik sesuai dengan parameter yang telah kami tentukan. Hasil transformasi ini memberikan tampilan logo Xendit yang sesuai dengan harapan kami.

Kami juga perhatikan bahwa terdapat ketidak-konsistenan dalam kode saat menerapkan translasi pada logo Android. Pada beberapa kasus, ketika nilai dx dan dy pada logo Android relatif kecil, perpindahan objek tersebut tidak jauh.

Namun, kami memperhatikan bahwa pada saat menerapkan translasi pada

logo Xendit dengan nilai dx dan dy yang lebih besar dari 5, objek yang ditampilkan bisa melebihi batas layar yang disediakan. Ini menunjukkan bahwa perubahan besar dalam translasi dapat memengaruhi posisi akhir objek yang keluar dari layar yang diinginkan.

b. Perkembangan Animasi pada Logo Android

Pembuatan animasi pada logo Android juga mencapai hasil yang diharapkan. Logo Android berhasil bergerak sesuai dengan logika animasi yang telah kami rancang. Animasi ini mencakup perubahan posisi dan tampilan logo Android, dan semuanya berjalan dengan lancar.

Selama animasi, logo Android bergerak dengan mulus tanpa adanya efek glitch atau ketidaksempurnaan visual yang mencolok. Ini menunjukkan bahwa implementasi animasi dalam kode kami berfungsi dengan baik. Selain itu, perubahan posisi dan tampilan logo Android selama animasi menciptakan efek visual yang menarik dan dinamis, sesuai dengan tujuan eksperimen.

c. Kesimpulan Analisis

Berdasarkan hasil analisis ini, kami dapat menyimpulkan bahwa transformasi grafis pada logo Xendit dan pembuatan animasi pada logo Android telah berhasil. Transformasi pada logo Xendit mencapai hasil yang efektif dalam mengubah tampilan objek, sementara animasi pada logo Android berjalan dengan lancar dan menciptakan efek visual yang menarik.

Kami juga mencatat ketidak-konsistenan dalam kode saat menerapkan translasi yang memengaruhi posisi akhir objek. Terutama, pada logo Android, perpindahan objek tidak selalu sebanding dengan nilai dx dan dy yang diberikan, sehingga objek tersebut mungkin melebihi batas layar yang diinginkan dalam beberapa kasus.

Hasil eksperimen ini menunjukkan bahwa perpustakaan grafis yang telah kami kembangkan dapat digunakan dengan efektif untuk mengendalikan tampilan dan pergerakan objek-objek grafis dalam lingkungan dua dimensi. Hal ini membuka potensi untuk pengembangan lebih lanjut dalam penggunaan transformasi dan animasi dalam aplikasi grafis. Selain itu, eksperimen ini menghasilkan dasar yang kuat untuk pengembangan lebih lanjut dalam konteks penggunaan transformasi grafis dan pembuatan animasi dalam pengembangan aplikasi grafis. Kami akan memperhatikan ketidak-konsistenan dalam translasi dalam pengembangan mendatang untuk meningkatkan kualitas aplikasi grafis kami.

VI. Saran atau Rekomendasi

6.1. Saran-saran untuk pengembangan

Dalam rangka meningkatkan kualitas dan efektivitas penggunaan transformasi grafis dan pembuatan animasi dalam pengembangan aplikasi grafis dua dimensi, kami mengusulkan beberapa saran sebagai berikut:

1. **Peningkatan Pengujian:** Kami merekomendasikan peningkatan pengujian perpustakaan grafis yang telah kami kembangkan, terutama dalam hal mengatasi ketidak-konsistenan dalam translasi. Pengujian yang lebih komprehensif dapat membantu mengidentifikasi masalah dan potensial ketidak-konsistenan dalam pergerakan objek grafis.
2. **Optimisasi Kinerja:** Untuk aplikasi grafis yang lebih kompleks, perlu mempertimbangkan pengoptimalan kinerja. Hal ini dapat mencakup penggunaan teknik buffering atau pemanfaatan teknologi OpenGL yang lebih canggih untuk meningkatkan kecepatan render.
3. **Penanganan Objek Lebih Besar:** Dalam situasi di mana objek yang lebih besar atau tampilan yang lebih luas diperlukan, perlu diperbarui kode atau perpustakaan sehingga objek tidak keluar dari layar saat menerapkan translasi yang besar.
4. **Dokumentasi Lengkap:** Melengkapi dokumentasi kode dengan informasi yang lebih rinci dan tutorial akan membantu pengembang lain dalam memahami dan memanfaatkan perpustakaan grafis yang telah kami kembangkan.

6.2. Rekomendasi Studi Lanjutan

Eksperimen ini memberikan dasar yang kuat untuk penelitian dan pengembangan lebih lanjut dalam bidang transformasi grafis dan pembuatan animasi. Beberapa rekomendasi studi lanjutan adalah sebagai berikut:

1. Studi Lebih Lanjut tentang Transformasi 3D

Mengembangkan pemahaman dan keterampilan dalam transformasi grafis tiga dimensi akan memungkinkan pengembang untuk menghadapi tantangan yang lebih kompleks dalam pengembangan aplikasi grafis.

2. Penelitian Efek Visual

Mengeksplorasi lebih lanjut efek visual seperti efek partikel, perubahan warna, dan pergeseran perspektif dalam animasi grafis dapat menjadi studi lanjutan yang menarik.

3. Optimisasi dan Perbaikan Kode

Melakukan penelitian lebih lanjut dalam optimisasi kode untuk kinerja yang lebih baik serta memperbaiki ketidak-konsistenan dalam translasi akan menjadi topik yang relevan untuk studi lanjutan.

4. Pengembangan Perpustakaan Grafis Lebih Lanjut

Melanjutkan pengembangan perpustakaan grafis yang lebih kaya fitur dan fleksibel dapat membantu dalam menciptakan aplikasi grafis yang lebih kompleks dan menarik.

5. Penelitian Interaksi Manusia dan Grafis

Studi tentang bagaimana manusia berinteraksi dengan objek grafis yang bergerak dan berubah dapat menjadi topik penelitian yang menarik dalam konteks pengembangan aplikasi yang melibatkan grafika interaktif.

Dengan melanjutkan studi lanjutan dalam bidang ini, pengembang grafis dapat terus mengembangkan keterampilan dan pengetahuan mereka serta berkontribusi pada perkembangan teknologi grafis yang lebih maju dan inovatif.

Dokumentasi

