APLIKASI GRAFIKA KOMPUTER MENGGUNAKAN PYTHON

Tugas Mata Kuliah

GRAFIKA KOMPUTER

KELAS IF-D

Semester Genap TA. 2024/2025

Oleh:

1. Arvidion Havas O	123220067	(Ketua)
2. Faaris Sayyid A.G	123220059	(Anggota)
3. Jovano Dion Manuel	123220103	(Anggota)

JURUSAN INFORMATIKA – PRODI INFORMATIKA
FAKULTAS TEKNIK INDUSTRI
UNIVERSITAS PEMBANGUNAN NASIONAL "VETERAN"
YOGYAKARTA
2025

HALAMAN PENGESAHAN PRESENTASI

APLIKASI GRAFIKA KOMPUTER MENGGUNAKAN PYTHON

Disusun oleh:

KELOMPOK

Nama	No.Mhs.	Tanda Tangan
Arvidion Havas Oktavian	123220067	
Faaris Sayyid Al-Ghozali	123220059	
Jovano Dion Manuel	123220103	

telah dipresentasikan pada tanggal 11 Juni 2025

Menyetujui
Dosen Pengampu

Herry Sofyan, S.T., M.Kom

DAFTAR ISI

BAB I
TEORI DASAR

1.1. Pendahuluan

Grafika komputer adalah cabang dari ilmu komputer yang berfokus pada pembuatan, manipulasi, dan representasi visual data secara digital. Bidang ini mencakup teknik untuk menggambar, menampilkan, dan memproses gambar dalam format digital, baik dua dimensi (2D) maupun tiga dimensi (3D). Grafika komputer digunakan dalam berbagai bidang, termasuk game, simulasi, desain produk, pemodelan, animasi, dan visualisasi data ilmiah.

Hubungan antara grafika komputer dan ilmu komputer sangat erat, karena grafika komputer memerlukan pemahaman mendalam tentang algoritma, struktur data, matematika komputer, serta arsitektur perangkat keras. Melalui grafika komputer, konsep abstrak dalam komputasi dapat divisualisasikan sehingga lebih mudah dipahami oleh manusia. Grafika komputer juga berperan penting dalam pengembangan antarmuka pengguna (user interface), sistem interaktif, serta kecerdasan buatan yang memerlukan pemrosesan visual.

Manfaat dari grafika komputer sangat luas, mulai dari peningkatan kualitas komunikasi visual, efisiensi dalam desain dan rekayasa, hingga mendukung pembelajaran interaktif. Dalam dunia profesional, grafika komputer mempercepat proses pengambilan keputusan melalui visualisasi yang informatif dan menarik.

1.2. Sistem Grafika

Sistem grafika terdiri dari perangkat keras dan perangkat lunak yang bekerja sama untuk menghasilkan gambar di layar. Komponen pentingnya meliputi CPU, GPU (Graphics Processing Unit), monitor, dan perangkat input (seperti mouse dan keyboard). Sistem ini juga mencakup algoritma untuk menghasilkan bentuk-bentuk visual dari data, serta antarmuka untuk interaksi pengguna.

1.3. Teknologi Display

Teknologi display digunakan untuk menampilkan gambar digital di layar. Dua sistem utama dalam teknologi display adalah:

1.3.1. Raster-Scan System

Raster-scan adalah metode paling umum digunakan pada monitor modern seperti CRT, LCD, dan LED. Sistem ini bekerja dengan menyusun gambar berdasarkan grid piksel dalam baris horizontal dari kiri ke kanan dan atas ke bawah. Gambar dibentuk melalui penyinaran titik-titik cahaya secara berurutan dan cepat, sehingga mata manusia melihat gambar utuh secara kontinu.

1.3.2. Random-Scan System

Random-scan, juga dikenal sebagai vector display, menggambar langsung garis demi garis, bukan berdasarkan piksel. Sistem ini lebih cocok untuk aplikasi teknik seperti CAD karena menghasilkan garis yang lebih halus. Namun, teknologi ini lebih mahal dan tidak mendukung gambar berwarna penuh secara efisien.

1.3.3. Peralatan Input Interaktif

Perangkat input interaktif memungkinkan pengguna memberikan perintah dan berinteraksi langsung dengan sistem grafika. Contohnya:

- Mouse dan Keyboard – Untuk memilih, menggambar, atau memasukkan data.
- Tablet Grafis – Untuk menggambar lebih presisi secara digital.
- Joystick – Untuk navigasi 3D atau dalam game.
- Touchscreen – Untuk input langsung dengan jari atau stylus.

1.4. Output Primitif

Output primitif adalah elemen-elemen dasar yang membentuk gambar dalam grafika komputer.

1.4.1. Titik dan Garis

Titik adalah unit terkecil dalam grafika komputer. Garis dibentuk dari dua titik yang dihubungkan, dan digunakan untuk membuat bentuk lebih kompleks.

1.4.2. Algoritma Pembentukan Garis

Algoritma yang digunakan untuk menggambar garis antara dua titik, seperti:

- Algoritma DDA (Digital Differential Analyzer) – Menggunakan pendekatan berbasis gradien untuk menggambar garis.
- Algoritma Bresenham – Lebih efisien karena hanya menggunakan operasi integer, sangat cocok untuk raster-scan.

1.4.3. Algoritma Pembentukan Lingkaran dan Elips

Algoritma Bresenham Circle – Untuk membentuk lingkaran dengan efisiensi tinggi. Algoritma Midpoint – Untuk lingkaran dan elips dengan menggunakan titik tengah dan simetri untuk mengurangi perhitungan.

1.4.4. Fill Area Primitif

Fill area digunakan untuk mengisi bentuk tertutup dengan warna atau pola.

- Boundary Fill – Mengisi area dengan warna hingga batas tertentu.
- Flood Fill – Mengisi area berdasarkan warna area awal yang sama.

1.5. Atribut Output Primitif

Dalam grafika komputer, output primitif seperti titik, garis, dan kurva tidak hanya ditentukan oleh posisi geometrisnya saja, tetapi juga oleh atribut visual yang menyertainya. Atribut-atribut ini memungkinkan objek grafis terlihat lebih nyata dan dapat dikustomisasi tampilannya sesuai kebutuhan pengguna.

1.5.1. Atribut Titik

Titik merupakan elemen grafis paling dasar dalam sistem grafika komputer. Meskipun sederhana, titik memiliki beberapa atribut penting yang memengaruhi tampilannya di layar. Salah satu atribut utama adalah warna titik, yang dapat ditentukan dalam bentuk warna solid seperti merah, hijau, atau biru, maupun berdasarkan kombinasi nilai RGB untuk menghasilkan warna yang lebih kompleks. Selain warna, ukuran titik juga merupakan atribut yang krusial, terutama dalam sistem raster. Dalam sistem ini, ukuran titik dapat diperbesar dengan menggambar beberapa piksel di sekitar koordinat utama agar titik terlihat lebih besar dan jelas. Ukuran ini berguna saat titik perlu ditonjolkan sebagai elemen penting dalam visualisasi. Tidak hanya ukuran, bentuk titik pun dapat divariasikan tergantung pada antarmuka grafis yang digunakan. Titik bisa ditampilkan sebagai persegi kecil,

lingkaran, atau bahkan simbol tertentu untuk memberikan makna visual tambahan, misalnya dalam grafik data atau peta interaktif.

1.5.2. Atribut Kurva dan Garis

Kurva dan garis merupakan komponen utama dalam membentuk objek visual yang lebih kompleks seperti poligon, diagram, dan bentuk geometris lainnya. Atribut pertama yang melekat pada garis adalah warnanya. Seperti pada titik, warna garis bisa disesuaikan menggunakan model RGB atau sistem palet warna lain yang tersedia dalam aplikasi. Warna yang berbeda dapat digunakan untuk membedakan kategori informasi atau memberi penekanan visual tertentu. Selain itu, garis memiliki atribut ketebalan yang menentukan seberapa tipis atau tebal garis akan terlihat. Ketebalan ini sangat berguna dalam desain visual, misalnya untuk membedakan garis utama dan garis bantu dalam skema atau diagram teknis. Jenis garis atau *line style* juga merupakan atribut penting. Garis bisa ditampilkan dalam bentuk solid (garis utuh), putus-putus, atau titik-titik, tergantung konteks penggunaannya. Dalam grafika modern, juga diterapkan teknik anti-aliasing, yaitu metode untuk menghaluskan tampilan garis agar tidak tampak bergerigi, khususnya pada garis diagonal. Teknik ini meningkatkan kualitas visual dengan membuat garis tampak lebih halus dan natural di mata pengguna.

1.5.3. Warna dan Grayscale

Atribut warna adalah salah satu aspek paling vital dalam grafika komputer karena sangat memengaruhi tampilan dan daya tarik visual suatu objek. Warna dalam grafika umumnya direpresentasikan menggunakan model RGB, yaitu kombinasi dari tiga warna dasar: merah (Red), hijau (Green), dan biru (Blue). Dengan mencampurkan ketiganya dalam berbagai proporsi, sistem ini mampu menghasilkan jutaan warna berbeda, menjadikannya sangat fleksibel untuk berbagai kebutuhan desain visual. Selain RGB, grafika komputer juga sering menggunakan representasi grayscale, yang menampilkan warna dalam tingkatan keabuan antara hitam dan putih. Grayscale banyak digunakan dalam pengolahan citra awal, pemrosesan citra medis, atau ketika tampilan warna tidak diperlukan. Di luar itu, terdapat juga model warna alternatif seperti CMY(K) yang umum dipakai

dalam pencetakan, serta HSV dan HSL yang memisahkan informasi warna berdasarkan rona (hue), kejenuhan (saturation), dan pencahayaan (value/lightness), sehingga lebih intuitif digunakan dalam pengaturan warna oleh desainer. Penggunaan model warna yang tepat sangat membantu dalam meningkatkan kualitas visual, keterbacaan, dan interpretasi data dalam aplikasi grafis..

1.6. Transformasi Dua Dimensi

1.6.1. Translasi

Translasi merupakan salah satu bentuk transformasi dua dimensi yang digunakan untuk menggeser posisi suatu objek dari satu tempat ke tempat lain pada bidang datar. Proses ini tidak mengubah bentuk, ukuran, maupun orientasi objek, melainkan hanya memindahkan lokasinya. Misalnya, jika sebuah kotak berada di suatu posisi awal, translasi dapat digunakan untuk memindahkannya ke posisi baru dengan menentukan seberapa jauh kotak itu digeser ke kanan, kiri, atas, atau bawah. Arah dan jarak pergeseran ditentukan oleh nilai translasi pada sumbu horizontal dan vertikal. Translasi banyak digunakan dalam pergerakan objek dalam animasi, pemetaan antarmuka pengguna, dan penyusunan elemen dalam ruang kerja desain.

1.6.2. Translasi

Rotasi adalah transformasi yang memutar objek terhadap suatu titik pusat, yang biasanya adalah titik asal sistem koordinat atau titik tertentu yang ditentukan secara eksplisit. Ketika objek diputar, setiap titik pada objek tersebut bergerak dalam lintasan melingkar dengan jarak tetap terhadap titik pusat rotasi. Sudut putar dapat ditentukan dalam arah searah jarum jam maupun berlawanan arah jarum jam. Rotasi sangat berguna dalam menciptakan efek visual seperti memutar gambar, menyusun pola simetris, atau menyimulasikan pergerakan objek dalam dunia nyata. Sebagai contoh, memutar panah ke arah utara dalam sebuah peta digital merupakan aplikasi dari transformasi rotasi.

1.6.3. Translasi

Skala adalah transformasi yang digunakan untuk mengubah ukuran suatu objek, baik dengan memperbesar maupun memperkecilnya. Transformasi ini dapat dilakukan secara seragam di seluruh arah, sehingga bentuk objek tetap proporsional, atau secara tidak seragam, yang dapat menyebabkan objek tampak memanjang atau menyempit di salah satu sumbu. Misalnya, jika sebuah gambar diperbesar dua kali lipat di kedua arah, maka bentuk dan proporsinya tetap sama, hanya ukurannya yang berubah. Namun, jika hanya diperbesar ke samping, gambar akan terlihat melebar. Skala berguna dalam menyesuaikan tampilan visual objek terhadap ukuran layar, memperjelas detail tertentu, atau menciptakan efek perspektif dalam grafik dua dimensi.

1.6.4. Matriks Transformasi

Dalam grafika komputer, semua transformasi seperti translasi, rotasi, dan skala dapat diwakili dalam bentuk matriks. Matriks transformasi ini memungkinkan sistem untuk memproses berbagai jenis transformasi dengan cara yang seragam dan efisien. Dengan menggunakan matriks, kita bisa menggabungkan beberapa transformasi menjadi satu operasi yang lebih cepat dan mudah diterapkan pada objek grafis. Teknik ini disebut komposisi transformasi, dan sangat berguna ketika objek perlu digeser, diputar, dan diperbesar secara bersamaan. Pendekatan matriks juga mempermudah pemrograman grafis karena transformasi dapat dilakukan secara berurutan dan disimpan dalam bentuk yang mudah dimanipulasi oleh perangkat lunak grafika. Matriks transformasi menjadi pondasi penting dalam pengolahan citra, pemodelan objek, dan rendering visual dalam aplikasi grafika modern.

BAB II
PERANCANGAN APLIKASI

2.1. Perancangan Menu

Aplikasi grafika komputer ini memiliki menu interaktif yang didesain dalam bentuk toolbar di sisi kanan layar. Toolbar terbagi menjadi beberapa bagian, seperti Basic Tools (Select, Brush, Eraser), Shape Tools (Garis, Persegi, Oval, Segitiga, Bintang, Segilima), Color Tools (pemilihan warna border dan fill), serta Transform Tools (Translasi, Skala, Rotasi, Cermin, dan Shear). Pengguna cukup memilih salah satu alat, lalu melakukan interaksi langsung pada canvas gambar. Semua kontrol menggunakan elemen GUI berbasis Tkinter seperti Radiobutton, Entry, dan Button.

2.2. Perancangan Antar Muka Pengguna

Antarmuka pengguna (UI) dirancang menggunakan Tkinter dan dibagi menjadi dua area utama, yaitu area gambar (canvas) di sebelah kiri dan toolbar kontrol di sebelah kanan. Canvas memiliki ukuran tetap dan digunakan untuk menggambar bentuk 2D secara langsung menggunakan mouse. Toolbar berisi berbagai kontrol yang tertata dalam kelompok-kelompok berdasarkan fungsinya. Misalnya, pengguna dapat memilih bentuk tertentu, mengatur warna garis dan isian, lalu mengklik pada canvas untuk menggambar. Transformasi objek dapat dilakukan dengan memasukkan nilai parameter dan memilih objek terlebih dahulu. Desain antarmuka ini mengutamakan kemudahan penggunaan serta alur kerja yang intuitif.

BAB III
IMPLEMENTASI PROGRAM

3.1. Perancangan Menu

- Laptop Lenovo Legion 5
- Prosesor AMD Ryzen 5 5600H
- RAM 16 GB
- Sistem Operasi Windows 11

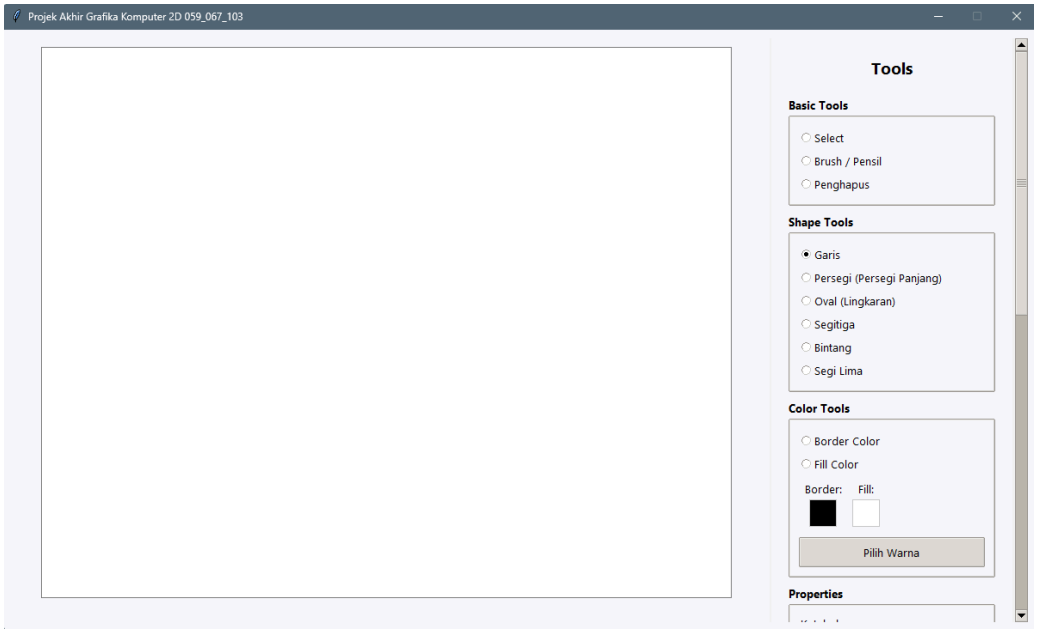
3.2. Perangkat Lunak yang Digunakan

- Bahasa Pemrograman : Python
- Library : Tkinter, Pillow, NumPy
- Code Editor : Visual Studio Code

3.3. Tampilan dan Modul Program

3.3.1. Tampilan

Antarmuka pengguna aplikasi ini dirancang untuk kemudahan penggunaan dalam menggambar dan memanipulasi objek 2D. Modul program diorganisasikan untuk memisahkan logika antarmuka, manajemen kanvas, penanganan peristiwa, dan fungsi transformasi.



Gambar 3.3.1. Tampilan UI Aplikasi

Antarmuka aplikasi terdiri dari dua area utama: area kanvas di sebelah kiri dan panel alat (toolbar) di sebelah kanan :

- Area Kanvas: Area utama di mana pengguna dapat menggambar berbagai bentuk 2D seperti garis, persegi, oval, segitiga, bintang, dan segi lima. Kanvas juga menjadi tempat objek yang dipilih ditampilkan bersama dengan kotak seleksi dan handle transformasi.
- Panel Alat (Toolbar) Kanan: Berisi semua kontrol interaktif yang dikelompokkan menjadi beberapa bagian:
- Basic Tools: Pilihan untuk mode "Select" (memilih dan memanipulasi objek yang sudah ada), "Brush / Pensil" (menggambar bebas), dan "Penghapus".
- Shape Tools: Tombol radio untuk memilih jenis bentuk 2D yang ingin digambar.
- Color Tools: Pilihan untuk menentukan "Border Color" dan "Fill Color" objek. Terdapat juga pratinjau warna dan tombol "Pilih Warna" untuk membuka color chooser.
- Properties: Slider untuk mengatur "Ketebalan" garis (brush/pensil, border objek) dan checkbox untuk "Batasi Persegi" atau "Batasi Lingkaran" saat menggambar bentuk.
- Transform: Bagian ini berisi input untuk melakukan transformasi geometris pada objek yang dipilih:
- Translate: Input untuk nilai X dan Y untuk menggeser objek.
- Scale: Input untuk nilai X dan Y untuk mengubah ukuran objek.
- Rotate: Input untuk sudut rotasi objek.
- Mirror: Tombol untuk mencerminkan objek secara "Horizontal" atau "Vertical".
- Shear: Input untuk nilai X dan Y untuk menerapkan transformasi shear.
- Clear Canvas Button: Tombol untuk menghapus semua objek dari kanvas.

3.3.2. Modul Program

Struktur program dibagi menjadi beberapa modul Python untuk pengelolaan kode yang lebih baik:

paint_app.py

Ini adalah file utama yang menginisialisasi aplikasi Tkinter, mengatur gaya, mendeklarasikan variabel-variabel global, dan mengelola manajer utama (CanvasManager, ToolbarManager, EventHandler). Fungsi-fungsi utama seperti pick_active_color, on_thickness_change, dan fungsi-fungsi untuk menerapkan transformasi (apply_translation, apply_scale, apply_rotation, apply_mirror, apply_shear) juga berada di sini. Implementasi flood_fill juga ada di dalam kelas PaintApp.

canvas_manager.py

Modul ini bertanggung jawab atas semua interaksi dan tampilan pada kanvas gambar.

a. Fungsi __init__

   menginisialisasi kanvas Tkinter dan objek PIL.Image yang digunakan untuk rendering gambar.

```
def __init__(self, root):
        self.root = root
        self.root.title("Simple Paint - Grafika Komputer 2D
Shapes")
        self.root.geometry("1200x700")
        self.root.resizable(False, False)
        self.root.configure(bg="#f5f7fa")

        # Setup style
        self.setup_style()

        # Setup variables harus dilakukan sebelum
setup_managers
        self.setup_variables()
        self.setup_managers()
        self.bind_events()
```

b. setup_image()

   Mengatur ulang gambar PIL.

```
def setup_image(self):
        """Setup PIL image untuk drawing"""
        self.image = Image.new("RGB", (self.width,
self.height), self.bg_color)
        self.draw = ImageDraw.Draw(self.image)
        self.update_canvas()
```

c. clear_canvas()

Membersihkan kanvas dan mereset semua status pemilihan.

```
def clear_canvas(self):
        """Bersihkan canvas"""
        self.image = Image.new("RGB", (self.width,
self.height), self.bg_color)
        self.draw = ImageDraw.Draw(self.image)
        self.canvas.delete("all")
        self.selected_item = None
        self.selection_box = None
        self.resize_handles = []
        self.active_handle = None
        self.original_coords = None
        self.update_canvas()
```

d. update_canvas()

Memperbarui tampilan gambar di kanvas Tkinter dari objek PIL.Image.

```
def update_canvas(self):
        """Update tampilan canvas"""
        self.photo = ImageTk.PhotoImage(self.image)
        self.canvas.delete("image")
        self.canvas.create_image(0, 0, anchor=tk.NW,
image=self.photo, tags="image")
```

e. show_selection()

Menggambar kotak seleksi dan resize handles di sekitar objek yang dipilih.

```
def show_selection(self):
        """Tampilkan selection box dan handles"""
        if not self.selected_item:
            return

        # Hapus selection box dan handles yang ada
        if self.selection_box:
            self.canvas.delete(self.selection_box)
        for handle in self.resize_handles:
            self.canvas.delete(handle)

        # Dapatkan bounding box dari item yang dipilih
        bbox = self.canvas.bbox(self.selected_item)
        if not bbox:
            return

        x1, y1, x2, y2 = bbox
        cx = (x1 + x2) / 2
        cy = (y1 + y2) / 2

        # Buat selection box
        self.selection_box = self.canvas.create_rectangle(
            x1, y1, x2, y2,
```

```
            outline='blue', dash=(4, 4))

        # Reset resize handles
        self.resize_handles = []

        # Tambah handle rotasi
        rotation_y = y1 - 25  # Posisi handle rotasi di atas
objek
        rotation_handle = self.canvas.create_oval(
            cx-5, rotation_y-5,
            cx+5, rotation_y+5,
            fill='green', outline='blue',
            tags=('handle', 'rotation'))
        self.resize_handles.append(rotation_handle)

        # Garis penghubung ke handle rotasi
        rotation_line = self.canvas.create_line(
            cx, y1, cx, rotation_y,
            fill='blue', dash=(2, 2))
        self.resize_handles.append(rotation_line)

        # Handle resize normal
        handle_positions = [
            ('nw', x1, y1), ('n', cx, y1), ('ne', x2, y1),
            ('w', x1, cy), ('e', x2, cy),
            ('sw', x1, y2), ('s', cx, y2), ('se', x2, y2)
        ]

        for pos, x, y in handle_positions:
            handle = self.canvas.create_rectangle(
                x-3, y-3, x+3, y+3,
                fill='white', outline='blue',
                tags=('handle', pos))
            self.resize_handles.append(handle)
```

f. clear_selection()

Menghapus kotak seleksi dan resize handles.

```
def clear_selection(self):
        """Hapus selection box dan handles"""
        if self.selection_box:
            self.canvas.delete(self.selection_box)
        for handle in self.resize_handles:
            self.canvas.delete(handle)
        self.selected_item = None
        self.selection_box = None
        self.resize_handles = []
        self.active_handle = None
        self.original_coords = None
```

g. create_preview_shape(shape, event)

Membuat pratinjau bentuk (garis putus-putus) saat pengguna menggerakkan mouse untuk menggambar.

```
def create_preview_shape(self, shape, event):
        """Buat preview shape saat mouse bergerak"""
        if self.preview_shape:
            self.canvas.delete(self.preview_shape)

        x, y = event.x, event.y

        if shape == "line":
            self.preview_shape = self.canvas.create_line(
                x, y, x+1, y+1,
                fill=self.app.current_color, width=1,
                dash=(4, 4))
        elif shape == "rectangle":
            self.preview_shape =
self.canvas.create_rectangle(
                x, y, x+50, y+50,
                outline=self.app.current_color, width=1,
                dash=(4, 4))
        elif shape == "oval":
            self.preview_shape = self.canvas.create_oval(
                x, y, x+50, y+50,
                outline=self.app.current_color, width=1,
                dash=(4, 4))
        elif shape == "triangle":
            points = [x, y+50, x+50, y+50, x+25, y]
            self.preview_shape = self.canvas.create_polygon(
                points, outline=self.app.current_color,
                fill="", width=1, dash=(4, 4))
        elif shape == "star":
            # Buat preview bintang
            radius_out = 25  # Radius luar tetap
            radius_in = radius_out * 0.4  # Radius dalam
            points = []
            for i in range(10):
                angle = math.pi/2 + (2*math.pi*i)/10
                radius = radius_out if i % 2 == 0 else
radius_in
                px = x + radius * math.cos(angle)
                py = y - radius * math.sin(angle)
                points.extend([px, py])
            self.preview_shape = self.canvas.create_polygon(
                points, outline=self.app.current_color,
                fill="", width=1, dash=(4, 4))
        elif shape == "pentagon":
            # Buat preview pentagon
            radius = 25  # Radius tetap
            points = []
            for i in range(5):
                angle = math.pi/2 + (2*math.pi*i)/5
                px = x + radius * math.cos(angle)
                py = y - radius * math.sin(angle)
```

```
                points.extend([px, py])
            self.preview_shape = self.canvas.create_polygon(
                points, outline=self.app.current_color,
                fill="", width=1, dash=(4, 4))
```

event_handlers.py

Modul ini menangani semua peristiwa mouse (tekan, gerak, lepas) yang terjadi di kanvas.

a. on_button_press(event)

   Menentukan apakah pengguna mengklik handle seleksi atau memulai gambar baru. Jika mode "select" aktif, ia memanggil handle_selection_press ().

```
def on_button_press(self, event):
        """Handle mouse button press event"""
        shape = self.app.current_shape.get()

        # Hapus preview shape
        if self.app.canvas_manager.preview_shape:

self.app.canvas_manager.canvas.delete(self.app.canvas_manage
r.preview_shape)
            self.app.canvas_manager.preview_shape = None

        if shape == "select":
            self.handle_selection_press(event)
            return

        # Reset seleksi jika memulai gambar baru
        self.reset_selection()
        self.start_drawing(event, shape)
```

b. handle_selection_press(event)

   Mengidentifikasi objek yang diklik atau handle transformasi.

```
def handle_selection_press(self, event):
        """Handle ketika tool select ditekan"""
        self.start_x = event.x
        self.start_y = event.y

        # Check if clicking on a handle
        items =
self.app.canvas_manager.canvas.find_overlapping(
            event.x-2, event.y-2, event.x+2, event.y+2)
```

```
        for item in items:
            tags =
self.app.canvas_manager.canvas.gettags(item)
            if 'handle' in tags:
                if 'rotation' in tags:
                    self.resize_mode = 'rotation'
                else:
                    self.resize_mode = [tag for tag in tags
if tag in ['nw','n','ne','e','se','s','sw','w']][0]
                self.app.canvas_manager.active_handle = item
                return

        # If not clicking handle, proceed with normal
selection
        items =
self.app.canvas_manager.canvas.find_overlapping(
            event.x-2, event.y-2, event.x+2, event.y+2)

        if items:
            selected = items[-1]  # Get topmost item
            if selected not in
[self.app.canvas_manager.selection_box,

*self.app.canvas_manager.resize_handles]:
                self.select_item(selected, event)
        else:
            self.app.canvas_manager.clear_selection()
```

c. select_item(item, event)

Menandai objek sebagai terpilih dan menampilkan selection box.

```
def select_item(self, item, event):
        """Pilih item dan setup untuk transformasi"""
        # Hapus seleksi sebelumnya
        self.reset_selection()

        self.app.canvas_manager.selected_item = item
        # Reset dan simpan koordinat asli item
        self.app.canvas_manager.original_coords = list(
            self.app.canvas_manager.canvas.coords(item))
        self.app.canvas_manager.show_selection()

        # Simpan posisi awal untuk drag
        self.drag_data["x"] = event.x
        self.drag_data["y"] = event.y
```

d. reset_selection()

Mereset status seleksi.

```
def reset_selection(self):
        """Reset state seleksi"""
        if self.app.canvas_manager.selection_box:

self.app.canvas_manager.canvas.delete(self.app.canvas_manage
r.selection_box)
        for handle in
self.app.canvas_manager.resize_handles:
            self.app.canvas_manager.canvas.delete(handle)
        self.app.canvas_manager.selected_item = None
        self.app.canvas_manager.original_coords = None
```

e. on_button_move(event)

Menangani pergerakan mouse saat tombol ditekan, baik untuk menggambar atau memanipulasi objek yang dipilih (move_selected(), resize_selected()).

```
def on_button_move(self, event):
        """Handle mouse movement dengan button ditekan"""
        shape = self.app.current_shape.get()

        if shape == "select":
            if self.app.canvas_manager.selected_item:
                if self.app.canvas_manager.active_handle:
                    self.resize_selected(event)
                else:
                    self.move_selected(event)
            return

        if not self.app.canvas_manager.temp_item:
            return

        if shape in ["brush", "eraser"]:
            self.continue_brush(event, shape)
        else:
            self.continue_shape(event, shape)
```

f. move_selected(event)

Menggeser objek yang dipilih.

```
def move_selected(self, event):
        """Gerakkan objek yang dipilih"""
        # Hitung perubahan posisi
        dx = event.x - self.drag_data["x"]
        dy = event.y - self.drag_data["y"]

        # Update posisi objek

self.app.canvas_manager.canvas.move(self.app.canvas_manager.
selected_item, dx, dy)

        # Update selection box dan handles
        self.app.canvas_manager.show_selection()
```

```
        # Update posisi drag terakhir
        self.drag_data["x"] = event.x
        self.drag_data["y"] = event.y
```

g. resize_selected(event)

Mengubah ukuran objek yang dipilih berdasarkan handle yang digeser.

```
def resize_selected(self, event):
        """Handle resize dan rotasi objek"""
        if not self.app.canvas_manager.selected_item or not
self.resize_mode:
            return

        canvas = self.app.canvas_manager.canvas
        item = self.app.canvas_manager.selected_item

        if self.resize_mode == "rotation":
            # Dapatkan koordinat objek dan titik pusat
            bbox = canvas.bbox(item)
            center_x = (bbox[0] + bbox[2]) / 2
            center_y = (bbox[1] + bbox[3]) / 2

            # Hitung sudut berdasarkan posisi mouse relatif
terhadap pusat
            dx = event.x - center_x
            dy = center_y - event.y  # Dibalik karena y-axis
terbalik di canvas
            angle = math.degrees(math.atan2(dy, dx))

            # Dapatkan sudut awal jika belum ada
            if not hasattr(self, 'start_angle'):
                self.start_angle = angle
                self.last_angle = 0

            # Hitung perubahan sudut dari posisi terakhir
            delta_angle = angle - self.start_angle -
self.last_angle
            self.last_angle = angle - self.start_angle

            # Rotasi menggunakan fungsi rotate_shape
            rotate_shape(canvas, item, delta_angle)
            self.app.canvas_manager.show_selection()
            return

        # Handle normal resizing
        dx = event.x - self.resize_start["x"]
        dy = event.y - self.resize_start["y"]

        coords = list(self.original_coords)
        shape_type =
self.app.canvas_manager.canvas.type(self.app.canvas_manager.
selected_item)

        if shape_type == "line":
```

```
            if self.resize_mode == "nw":
                coords[0] += dx
                coords[1] += dy
            elif self.resize_mode == "se":
                coords[2] += dx
                coords[3] += dy

        elif shape_type in ["rectangle", "oval"]:
            x1, y1, x2, y2 = coords

            # Handle different resize modes
            if "n" in self.resize_mode: y1 += dy
            if "s" in self.resize_mode: y2 += dy
            if "w" in self.resize_mode: x1 += dx
            if "e" in self.resize_mode: x2 += dx

            # Update coordinates
            coords = [x1, y1, x2, y2]

            # Maintain aspect ratio if shift is held
            if self.app.square_var.get():
                width = abs(x2 - x1)
                height = abs(y2 - y1)
                if width > height:
                    if "n" in self.resize_mode: y1 = y2 -
width
                    if "s" in self.resize_mode: y2 = y1 +
width
                else:
                    if "w" in self.resize_mode: x1 = x2 -
height
                    if "e" in self.resize_mode: x2 = x1 +
height

        elif shape_type == "polygon":  # Triangle, Star,
Pentagon
            vertex = int(self.resize_mode[1])  # Get vertex
number from tag
            coords[vertex*2] += dx
            coords[vertex*2 + 1] += dy

        # Update the shape

self.app.canvas_manager.canvas.coords(self.app.canvas_manage
r.selected_item, *coords)

        # Update selection box and handles
        self.app.canvas_manager.show_selection()

        # Update start position for next movement
        self.resize_start = {"x": event.x, "y": event.y}
```

h. on_button_release(event)

Finalisasi operasi menggambar atau transformasi.

```
def on_button_release(self, event):
        """Handle mouse button release"""
        shape = self.app.current_shape.get()

        if shape == "select":
            if self.app.canvas_manager.selected_item:
                self.app.canvas_manager.original_coords =
list(
                    self.app.canvas_manager.canvas.coords(

self.app.canvas_manager.selected_item))
                # Reset transform states
                self.app.canvas_manager.active_handle = None
                self.resize_mode = None
            return

        if not self.app.canvas_manager.temp_item:
            return

        # Finalize shape
        if shape in ["brush", "eraser"]:
            self.finalize_brush()
        else:
            self.finalize_shape()
```

i. start_drawing (event, shape), start_brush(event, thickness), start_eraser (event, thickness), start_shape (event, shape, thickness)

Fungsi-fungsi untuk memulai berbagai mode gambar.

```
def start_drawing(self, event, shape):
        """Mulai operasi menggambar"""
        self.start_x = event.x
        self.start_y = event.y
        thick = self.app.thickness.get()

        if shape == "brush":
            self.start_brush(event, thick)
        elif shape == "eraser":
            self.start_eraser(event, thick)
        elif shape in ["line", "rectangle", "oval",
"triangle", "star", "pentagon"]:
            self.start_shape(event, shape, thick)

    def start_brush(self, event, thickness):
        """Mulai menggambar dengan brush"""
        self.app.canvas_manager.brush_points = [(event.x,
event.y)]
        self.app.canvas_manager.temp_item =
self.app.canvas_manager.canvas.create_line(
            event.x, event.y, event.x+1, event.y+1,
            fill=self.app.current_color, width=thickness,
            capstyle=tk.ROUND, smooth=True)

    def start_eraser(self, event, thickness):
```

```
        """Mulai menghapus dengan eraser"""
        self.app.canvas_manager.brush_points = [(event.x,
event.y)]
        eraser_width = max(10, thickness*5)
        self.app.canvas_manager.temp_item =
self.app.canvas_manager.canvas.create_line(
            event.x, event.y, event.x+1, event.y+1,
            fill=self.app.canvas_manager.bg_color,
width=eraser_width,
            capstyle=tk.ROUND, smooth=True)

    def start_shape(self, event, shape, thickness):
        """Mulai menggambar bentuk"""
        if shape == "line":
            self.app.canvas_manager.temp_item =
self.app.canvas_manager.canvas.create_line(
                self.start_x, self.start_y, event.x,
event.y,
                fill=self.app.current_color,
width=thickness)
        elif shape == "rectangle":
            self.app.canvas_manager.temp_item =
self.app.canvas_manager.canvas.create_rectangle(
                self.start_x, self.start_y, event.x,
event.y,
                outline=self.app.current_color,
width=thickness, fill='')
        elif shape == "oval":
            self.app.canvas_manager.temp_item =
self.app.canvas_manager.canvas.create_oval(
        self.start_x, self.start_y, event.x, event.y,
                outline=self.app.current_color,
width=thickness, fill='')
        elif shape == "triangle":
            self.app.canvas_manager.temp_item =
self.app.canvas_manager.canvas.create_polygon(
                self.start_x, self.start_y,
                self.start_x, self.start_y,
                self.start_x, self.start_y,
                outline=self.app.current_color, fill="",
width=thickness)
        elif shape == "star":
            # Buat bintang dengan 5 titik
            points = [self.start_x, self.start_y] * 10  # 10
titik untuk bintang 5 sudut
            self.app.canvas_manager.temp_item =
self.app.canvas_manager.canvas.create_polygon(
                points, outline=self.app.current_color,
fill="", width=thickness)
        elif shape == "pentagon":
            # Buat segi lima dengan 5 titik
            points = [self.start_x, self.start_y] * 5  # 5
titik untuk pentagon
            self.app.canvas_manager.temp_item =
self.app.canvas_manager.canvas.create_polygon(
                points, outline=self.app.current_color,
```

```
fill="", width=thickness)
```

j. continue_brush(event, shape), continue_shape(event, shape)

Fungsi-fungsi untuk melanjutkan gambar atau bentuk saat mouse bergerak.

```
def continue_brush(self, event, shape):
        """Lanjutkan menggambar brush/eraser"""

self.app.canvas_manager.brush_points.append((event.x,
event.y))
        points_flat = [coord for point in
self.app.canvas_manager.brush_points
                       for coord in point]
        self.app.canvas_manager.canvas.coords(
            self.app.canvas_manager.temp_item, *points_flat)

    def continue_shape(self, event, shape):
        """Lanjutkan menggambar bentuk"""
        x0, y0 = self.start_x, self.start_y
        x1, y1 = event.x, event.y

        if shape == "line":
            self.app.canvas_manager.canvas.coords(
                self.app.canvas_manager.temp_item, x0, y0,
x1, y1)
        elif shape == "rectangle":
            if self.app.square_var.get():
                side = min(abs(x1 - x0), abs(y1 - y0))
                x1 = x0 + side if x1 >= x0 else x0 - side
                y1 = y0 + side if y1 >= y0 else y0 - side
            self.app.canvas_manager.canvas.coords(
                self.app.canvas_manager.temp_item, x0, y0,
x1, y1)
        elif shape == "oval":
            if self.app.circle_var.get():
                diameter = min(abs(x1 - x0), abs(y1 - y0))
                x1 = x0 + diameter if x1 >= x0 else x0 -
diameter
                y1 = y0 + diameter if y1 >= y0 else y0 -
diameter
            self.app.canvas_manager.canvas.coords(
                self.app.canvas_manager.temp_item, x0, y0,
x1, y1)
        elif shape == "triangle":
            x_min = min(x0, x1)
            x_max = max(x0, x1)
            y_min = min(y0, y1)
            y_max = max(y0, y1)
            points = [
                x_min, y_max,
                x_max, y_max,
                (x_min + x_max)/2, y_min
            ]
            self.app.canvas_manager.canvas.coords(
                self.app.canvas_manager.temp_item, *points)
```

```
        elif shape == "star":
            # Hitung radius luar dan dalam untuk bintang
            radius_out = ((x1 - x0)**2 + (y1 - y0)**2)**0.5
            radius_in = radius_out * 0.4  # Radius dalam 40%
dari radius luar
            center_x = x0
            center_y = y0

            # Buat 10 titik untuk bintang (5 sudut luar dan
5 sudut dalam)
            points = []
            for i in range(10):
                angle = math.pi/2 + (2*math.pi*i)/10  #
Mulai dari sudut 90 derajat
                radius = radius_out if i % 2 == 0 else
radius_in
                px = center_x + radius * math.cos(angle)
                py = center_y - radius * math.sin(angle)  #
Minus karena koordinat Y terbalik
                points.extend([px, py])

            self.app.canvas_manager.canvas.coords(
                self.app.canvas_manager.temp_item, *points)
        elif shape == "pentagon":
            # Hitung radius untuk pentagon
            radius = ((x1 - x0)**2 + (y1 - y0)**2)**0.5
            center_x = x0
            center_y = y0

            # Buat 5 titik untuk pentagon
            points = []
            for i in range(5):
                angle = math.pi/2 + (2*math.pi*i)/5  # Mulai
dari sudut 90 derajat
                px = center_x + radius * math.cos(angle)
                py = center_y - radius * math.sin(angle)  #
Minus karena koordinat Y terbalik
                points.extend([px, py])

            self.app.canvas_manager.canvas.coords(
                self.app.canvas_manager.temp_item, *points)
```

k. finalize_brush(), finalize_shape()

Fungsi-fungsi untuk menyelesaikan gambar.

```
def finalize_brush(self):
        """Finalisasi brush stroke"""
        self.app.canvas_manager.temp_item = None
        self.app.canvas_manager.brush_points = []

    def finalize_shape(self):
        """Finalisasi bentuk yang digambar"""
        self.app.canvas_manager.temp_item = None
```

l. on_mouse_move(event)

Menampilkan pratinjau bentuk saat mouse bergerak tanpa tombol ditekan.

```
def on_mouse_move(self, event):
        """Handle mouse movement tanpa button ditekan"""
        shape = self.app.current_shape.get()
        if shape in ["rectangle", "oval", "triangle",
"line", "star", "pentagon"]:

self.app.canvas_manager.create_preview_shape(shape, event)
```

transformations.py

Modul ini berisi fungsi-fungsi matematis untuk melakukan transformasi geometris pada koordinat objek.

a. calculate_fix_point(coords)

Menghitung titik pusat objek untuk transformasi.

```
def calculate_fix_point(coords):
    """
    Menghitung fix point (titik pusat) untuk transformasi
    Args:
        coords: List koordinat [x1,y1, x2,y2, ...]
    Returns:
        (center_x, center_y): Tuple berisi koordinat titik
pusat
    """
    if not coords:
        return 0, 0

    # Konversi list koordinat menjadi list titik [(x1,y1),
(x2,y2), ...]
    points = [(coords[i], coords[i+1]) for i in range(0,
len(coords), 2)]

    # Hitung titik pusat sebagai rata-rata koordinat
    center_x = sum(x for x, _ in points) / len(points)
    center_y = sum(y for _, y in points) / len(points)

    logger.debug(f"Calculate fix point: coords={coords},
center=({center_x}, {center_y})")
    return center_x, center_y
```

b. translate_points(coords, dx, dy)

Menerapkan translasi

```
def translate_points(coords, dx, dy):
    """
```

```
    Translasi semua titik dengan offset dx, dy
    Args:
        coords: List koordinat [x1,y1, x2,y2, ...]
        dx: Pergeseran pada sumbu x
        dy: Pergeseran pada sumbu y
    Returns:
        List koordinat setelah translasi
    """
    new_coords = []
    for i in range(0, len(coords), 2):
        new_coords.extend([coords[i] + dx, coords[i+1] +
dy])
    return new_coords
```

c. scale_points(coords, sx, sy, center_x=None, center_y=None)

Menerapkan penskalaan.

```
def scale_points(coords, sx, sy, center_x=None,
center_y=None):
    """
    Scaling semua titik dengan faktor sx, sy dari titik
pusat
    Args:
        coords: List koordinat [x1,y1, x2,y2, ...]
        sx: Faktor skala pada sumbu x
        sy: Faktor skala pada sumbu y
        center_x: Titik pusat x (optional)
    center_y: Titik pusat y (optional)
    Returns:
        List koordinat setelah scaling
    """
    if center_x is None or center_y is None:
        center_x, center_y = calculate_fix_point(coords)

    new_coords = []
    for i in range(0, len(coords), 2):
        # Translasi ke origin
        x = coords[i] - center_x
        y = coords[i+1] - center_y
        # Scaling
        x *= sx
        y *= sy
        # Translasi kembali
        x += center_x
        y += center_y
        new_coords.extend([x, y])
    return new_coords
```

d. create_rotation_matrix(angle_degrees)

Membuat matriks rotasi 2D.

```
def create_rotation_matrix(angle_degrees):
```

```
    """Create 2D rotation matrix"""
    angle_rad = math.radians(angle_degrees)
    cos_val = math.cos(angle_rad)
    sin_val = math.sin(angle_rad)
    return np.array([
        [cos_val, -sin_val],
        [sin_val, cos_val]
    ])
```

e. rotate_points_matrix(points, center, angle_degrees)

Merotasi kumpulan titik menggunakan matriks rotasi.

```
def rotate_points_matrix(points, center, angle_degrees):
    """
    Rotate points around center using matrix transformation
    Args:
        points: List of points as [[x1, y1], [x2, y2], ...]
        center: [cx, cy] center point
        angle_degrees: Rotation angle in degrees
    Returns:
        Rotated points in same format as input
    """
    # Convert to numpy arrays
    points_array = np.array(points)
    center_array = np.array(center)

    # Create rotation matrix
    rotation_matrix = create_rotation_matrix(angle_degrees)

    # Center points at origin
    centered = points_array - center_array

    # Apply rotation
    rotated = np.dot(centered, rotation_matrix.T)

    # Move back to original position
    result = rotated + center_array

    return result.tolist()
```

f. apply_transformation(points, matrix, center)

Menerapkan matriks transformasi umum ke titik-titik.

```
def apply_transformation(points, matrix, center):
    """
    Apply transformation matrix to points
    Args:
        points: List of points as [[x1, y1], [x2, y2], ...]
        matrix: 2x2 transformation matrix
        center: [cx, cy] center point
    Returns:
        Transformed points in same format as input
    """
```

```
    # Convert to numpy array and reshape
    points_array = np.array(points).reshape(-1, 2)
    center_array = np.array(center)

    # Center points at origin
    centered = points_array - center_array

    # Apply transformation
    transformed = np.dot(centered, matrix.T)

    # Move back to original position
    result = transformed + center_array

    return result.flatten().tolist()
```

g. rotate_shape(canvas, item_id, angle_degrees)

Fungsi khusus untuk merotasi bentuk yang ada di kanvas, yang menangani perbedaan perilaku untuk berbagai jenis bentuk (persegi/oval diubah menjadi poligon untuk rotasi yang lebih baik).

```
def rotate_shape(canvas, item_id, angle_degrees):
    """
    Rotate a shape on the canvas
    Args:
        canvas: Tkinter canvas
        item_id: ID of the item to rotate
        angle_degrees: Rotation angle in degrees
    """
    shape_type = canvas.type(item_id)
    coords = list(canvas.coords(item_id))

    logger.debug(f"Rotating shape: type={shape_type},
id={item_id}, angle={angle_degrees}")
    logger.debug(f"Original coords: {coords}")

    if not coords:
        logger.warning("No coordinates found for shape")
        return

    # Initialize or get rotation tracking
    if not hasattr(canvas, f'total_rotation_{item_id}'):
        setattr(canvas, f'total_rotation_{item_id}', 0)
        setattr(canvas, f'original_coords_{item_id}',
coords.copy())
        setattr(canvas, f'original_center_{item_id}', None)

    # Update total rotation
    current_rotation = getattr(canvas,
f'total_rotation_{item_id}')
    new_rotation = (current_rotation + angle_degrees) % 360
    setattr(canvas, f'total_rotation_{item_id}',
new_rotation)
```

```
    # Get original coordinates and center
    original_coords = getattr(canvas,
f'original_coords_{item_id}')
    original_center = getattr(canvas,
f'original_center_{item_id}')

    if shape_type in ["rectangle", "oval"]:
        x1, y1, x2, y2 = original_coords

        # Calculate center if not stored
        if original_center is None:
            center_x = (x1 + x2) / 2
            center_y = (y1 + y2) / 2
            original_center = [center_x, center_y]
            setattr(canvas, f'original_center_{item_id}',
original_center)

        if not hasattr(canvas, f'polygon_{item_id}'):
            # Store original properties
            props = {
                'fill': canvas.itemcget(item_id, "fill"),
                'outline': canvas.itemcget(item_id,
"outline"),
                'width': canvas.itemcget(item_id, "width")
            }

            if shape_type == "rectangle":
                # Create points for rectangle
                points = [
                    [x1, y1],  # Top-left
                    [x2, y1],  # Top-right
                    [x2, y2],  # Bottom-right
                    [x1, y2]   # Bottom-left
                ]
            else:  # oval
                # Create points for oval approximation
                num_points = 36
                points = []
                rx = (x2 - x1) / 2
                ry = (y2 - y1) / 2
                cx, cy = original_center

                for i in range(num_points):
                    theta = (2 * math.pi * i) / num_points
                    x = cx + rx * math.cos(theta)
                    y = cy + ry * math.sin(theta)
                    points.append([x, y])

            # Create polygon
            polygon_points = [coord for point in points for
coord in point]
            polygon_id = canvas.create_polygon(
                polygon_points,
                fill=props['fill'],
                outline=props['outline'],
                width=props['width'],
```

```
                smooth=(shape_type == "oval")
            )

            # Hide original shape and store polygon
reference
            canvas.itemconfig(item_id, state='hidden')
            setattr(canvas, f'polygon_{item_id}',
polygon_id)
            setattr(canvas, f'original_points_{item_id}',
points)

        # Rotate the shape
        original_points = getattr(canvas,
f'original_points_{item_id}')
        polygon_id = getattr(canvas, f'polygon_{item_id}')

        # Apply rotation from original position
        rotated_points =
rotate_points_matrix(original_points, original_center,
new_rotation)

        # Update polygon coordinates
        flat_points = [coord for point in rotated_points for
coord in point]
        canvas.coords(polygon_id, *flat_points)

    elif shape_type == "line":
        x1, y1, x2, y2 = original_coords
        points = [[x1, y1], [x2, y2]]

        if original_center is None:
            center_x = (x1 + x2) / 2
            center_y = (y1 + y2) / 2
            original_center = [center_x, center_y]
            setattr(canvas, f'original_center_{item_id}',
original_center)

        rotated_points = rotate_points_matrix(points,
original_center, new_rotation)
        canvas.coords(item_id, *[coord for point in
rotated_points for coord in point])

    elif shape_type == "polygon":
        points = [[coords[i], coords[i+1]] for i in range(0,
len(coords), 2)]

        if original_center is None:
            center_x = sum(x for x, y in points) /
len(points)
            center_y = sum(y for x, y in points) /
len(points)
            original_center = [center_x, center_y]
            setattr(canvas, f'original_center_{item_id}',
original_center)

        rotated_points = rotate_points_matrix(points,
```

```
original_center, new_rotation)
        canvas.coords(item_id, *[coord for point in
rotated_points for coord in point])

    logger.debug(f"Rotation complete: angle={new_rotation},
center={original_center}")
```

h. rotate_point(x, y, cx, cy, angle_rad)

Fungsi bantu untuk merotasi satu titik.

```
def rotate_points(coords, angle_degrees, center_x=None,
center_y=None):
    """
    Rotasi koordinat dengan sudut dan titik pusat tertentu
    Args:
        coords: List koordinat [x1,y1, x2,y2, ...]
        angle_degrees: Sudut rotasi dalam derajat
        center_x: Titik pusat x (optional)
        center_y: Titik pusat y (optional)
    Returns:
        List koordinat setelah rotasi
    """
    if not coords:
        return coords

    # Tentukan titik pusat jika tidak disediakan
    if center_x is None or center_y is None:
        center_x, center_y = calculate_fix_point(coords)

    logger.debug(f"Rotating points around ({center_x},
{center_y}) by {angle_degrees}°")
    logger.debug(f"Original coords: {coords}")

    # Konversi sudut ke radian
    angle_rad = math.radians(angle_degrees)

    # Rotasi setiap titik
    rotated_points = []
    for i in range(0, len(coords), 2):
        x, y = coords[i], coords[i+1]
        new_x, new_y = rotate_point(x, y, center_x,
center_y, angle_rad)
        rotated_points.extend([new_x, new_y])

    logger.debug(f"Rotated coords: {rotated_points}")
    return rotated_points
```

i. get_rotated_bbox(x1, y1, x2, y2, cx, cy, angle_rad)

Menghitung bounding box setelah rotasi.

```
def get_rotated_bbox(x1, y1, x2, y2, cx, cy, angle_rad):
    """
    Calculate the bounding box of a rotated rectangle
    """
    # Get all four corners
    corners = [
        (x1, y1),  # Top-left
        (x2, y1),  # Top-right
        (x2, y2),  # Bottom-right
        (x1, y2)   # Bottom-left
    ]

    # Rotate each corner
    rotated_corners = [rotate_point(x, y, cx, cy, angle_rad)
for x, y in corners]

    # Get min/max coordinates
    xs = [x for x, y in rotated_corners]
    ys = [y for x, y in rotated_corners]

    return min(xs), min(ys), max(xs), max(ys)
```

j. mirror_points(coords, axis="x", center_x=None, center_y=None)

Menerapkan pencerminan.

```
ef mirror_points(coords, axis="x", center_x=None,
center_y=None):
    """
    Pencerminan semua titik terhadap sumbu x atau y
    Args:
        coords: List koordinat [x1,y1, x2,y2, ...]
        axis: Sumbu pencerminan ("x" atau "y")
        center_x: Titik pusat x (optional)
        center_y: Titik pusat y (optional)
    Returns:
        List koordinat setelah pencerminan
    """
    if center_x is None or center_y is None:
        center_x, center_y = calculate_fix_point(coords)

    new_coords = []
    for i in range(0, len(coords), 2):
        x = coords[i]
        y = coords[i+1]

        if axis == "x":
            # Pencerminan terhadap sumbu x (flip vertical)
            new_coords.extend([x, 2*center_y - y])
        else:  # axis == "y"
            # Pencerminan terhadap sumbu y (flip horizontal)
            new_coords.extend([2*center_x - x, y])

    return new_coords
```

k. shear_points(coords, shx=0, shy=0, center_x=None, center_y=None)

Menerapkan shear.

```
def shear_points(coords, shx=0, shy=0, center_x=None,
center_y=None):
    """
    Shear transformation pada semua titik
    Args:
        coords: List koordinat [x1,y1, x2,y2, ...]
        shx: Faktor shear pada sumbu x
        shy: Faktor shear pada sumbu y
        center_x: Titik pusat x (optional)
        center_y: Titik pusat y (optional)
    Returns:
        List koordinat setelah shear
    """
    if center_x is None or center_y is None:
        center_x, center_y = calculate_fix_point(coords)

    new_coords = []
    for i in range(0, len(coords), 2):
        # Translasi ke origin
        x = coords[i] - center_x
        y = coords[i+1] - center_y

        # Aplikasikan shear
        new_x = x + shx * y
        new_y = shy * x + y

        # Translasi kembali
        new_coords.extend([new_x + center_x, new_y +
center_y])

    return new_coords
```

ui_components.py

Modul ini bertanggung jawab untuk membangun dan mengatur semua komponen antarmuka pengguna di toolbar.

a. __init__ dan setup_style()

Mengatur gaya widget Tkinter.

```
def __init__(self, parent, app):
        self.parent = parent
        self.app = app
        self.setup_style()
        self.create_toolbar()

    def setup_style(self):
        """Setup style untuk widget ttk"""
        self.style = ttk.Style(self.parent)
```

```
        self.style.theme_use('clam')
        self.style.configure('TFrame', background='#f5f7fa')
        self.style.configure('TLabel', background='#f5f7fa',
font=('Segoe UI', 10))
        self.style.configure('TButton', font=('Segoe UI',
10), padding=6)
        self.style.configure('TCheckbutton',
background='#f5f7fa', font=('Segoe UI', 10))
        self.style.configure('TRadiobutton',
background='#f5f7fa', font=('Segoe UI', 10))
        self.style.configure('TLabelframe',
background='#f5f7fa')
        self.style.configure('TLabelframe.Label',
background='#f5f7fa', font=('Segoe UI', 10, 'bold'))
        self.style.map('TButton', background=[('active',
'#ddd')])
```

b. create_toolbar()

Membuat struktur dasar toolbar, termasuk canvas dan scrollbar untuk memungkinkan scrolling.

```
def create_toolbar(self):
        """Membuat toolbar dengan semua komponennya"""
        # Toolbar container
        toolbar_container = ttk.Frame(self.parent,
width=300)
        toolbar_container.pack(side=tk.RIGHT, fill=tk.Y,
padx=(10, 0))
        toolbar_container.pack_propagate(False)

        # Canvas dan scrollbar untuk toolbar
        toolbar_canvas = tk.Canvas(toolbar_container)
        scrollbar = ttk.Scrollbar(toolbar_container,
orient="vertical",

command=toolbar_canvas.yview)
        self.toolbar = ttk.Frame(toolbar_canvas,
style='TFrame', padding=20)

        # Konfigurasi scrolling

toolbar_canvas.configure(yscrollcommand=scrollbar.set)
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        toolbar_canvas.pack(side=tk.LEFT, fill=tk.BOTH,
expand=True)

        # Buat window di dalam canvas untuk toolbar
        toolbar_window = toolbar_canvas.create_window((0,
0), window=self.toolbar,

anchor="nw", width=280)

        # Update scroll region saat ukuran toolbar berubah
        def configure_scroll_region(event):
```

```
toolbar_canvas.configure(scrollregion=toolbar_canvas.bbox("a
ll"))
        self.toolbar.bind('<Configure>',
configure_scroll_region)

        # Bind mouse wheel untuk scroll
        def on_mousewheel(event):

toolbar_canvas.yview_scroll(int(-1*(event.delta/120)),
"units")
        toolbar_canvas.bind_all("<MouseWheel>",
on_mousewheel)

        self.create_tool_sections()
```

c. create_tool_sections()

Memanggil fungsi-fungsi untuk membuat setiap bagian alat (Basic Tools, Shape Tools, Color Tools, Properties, Transform, Clear Canvas Button).

```
  def create_tool_sections(self):
        """Membuat semua section tools di toolbar"""
        # Judul toolbar
        lbl_title = ttk.Label(self.toolbar, text="Tools",
font=('Segoe UI', 14, 'bold'))
        lbl_title.pack(pady=(0, 20))

        # Basic Tools Section
        self.create_basic_tools()

        # Shape Tools Section
        self.create_shape_tools()

        # Color Tools Section
        self.create_color_tools()

        # Properties Section
        self.create_properties_section()

        # Transform Section
        self.create_transform_section()

        # Clear Canvas Button
        self.create_clear_button()
```

d. create_basic_tools (), create_shape_tools (), create_color_tools (), create_properties_section (), create_transform_section (), create_clear_button (), create_transform_frame ()

Fungsi-fungsi ini masing-masing membuat kelompok widget yang spesifik untuk kategori alat yang berbeda di toolbar.

```
def create_basic_tools(self):
        """Membuat section Basic Tools"""
        basic_tools = [
            ("Select", "select"),
            ("Brush / Pensil", "brush"),
            ("Penghapus", "eraser"),
        ]

        basic_tools_frame = ttk.LabelFrame(self.toolbar,
text="Basic Tools", padding=10)
        basic_tools_frame.pack(fill=tk.X, pady=(0, 10))

        for text, value in basic_tools:
            rb = ttk.Radiobutton(basic_tools_frame,
text=text,

variable=self.app.current_shape, value=value)
            rb.pack(anchor='w', pady=2)

def create_shape_tools(self):
        """Membuat section Shape Tools"""
        shape_tools = [
            ("Garis", "line"),
            ("Persegi (Persegi Panjang)", "rectangle"),
            ("Oval (Lingkaran)", "oval"),
            ("Segitiga", "triangle"),
            ("Bintang", "star"),
            ("Segi Lima", "pentagon")
        ]

        shape_tools_frame = ttk.LabelFrame(self.toolbar,
text="Shape Tools", padding=10)
        shape_tools_frame.pack(fill=tk.X, pady=(0, 10))

        for text, value in shape_tools:
            rb = ttk.Radiobutton(shape_tools_frame,
text=text,

variable=self.app.current_shape, value=value)
            rb.pack(anchor='w', pady=2)

    def create_color_tools(self):
        """Membuat section Color Tools"""
        color_tools_frame = ttk.LabelFrame(self.toolbar,
text="Color Tools", padding=10)
        color_tools_frame.pack(fill=tk.X, pady=(0, 10))

        for text, value in [("Border Color", "border"),
("Fill Color", "fill")]:
            rb = ttk.Radiobutton(color_tools_frame,
text=text,

variable=self.app.color_mode, value=value)
```

```
            rb.pack(anchor='w', pady=2)

        # Preview warna
        color_preview_frame = ttk.Frame(color_tools_frame)
        color_preview_frame.pack(fill=tk.X, pady=(5, 0))

        # Border color preview
        border_frame = ttk.Frame(color_preview_frame)
        border_frame.pack(side=tk.LEFT, padx=5)
        ttk.Label(border_frame, text="Border:").pack()
        self.app.color_display = tk.Canvas(border_frame,
width=30, height=30,

bg=self.app.current_color,

highlightthickness=1, highlightbackground="#ccc")
        self.app.color_display.pack(pady=2)

        # Fill color preview
        fill_frame = ttk.Frame(color_preview_frame)
        fill_frame.pack(side=tk.LEFT, padx=5)
        ttk.Label(fill_frame, text="Fill:").pack()
        self.app.fill_color_display = tk.Canvas(fill_frame,
width=30, height=30,

bg=self.app.fill_color,

highlightthickness=1, highlightbackground="#ccc")
        self.app.fill_color_display.pack(pady=2)

        # Tombol pilih warna
        pick_color_btn = ttk.Button(color_tools_frame,
text="Pilih Warna",

command=self.app.pick_active_color)
        pick_color_btn.pack(fill=tk.X, pady=(10, 0))

    def create_properties_section(self):
        """Membuat section Properties"""
        properties_frame = ttk.LabelFrame(self.toolbar,
text="Properties", padding=10)
        properties_frame.pack(fill=tk.X, pady=(0, 10))

        # Ketebalan garis
        ttk.Label(properties_frame,
text="Ketebalan:").pack(anchor='w')
        thickness_frame = ttk.Frame(properties_frame)
        thickness_frame.pack(fill=tk.X, pady=(0, 5))

        self.app.thickness_scale =
ttk.Scale(thickness_frame, from_=1, to=20,

orient=tk.HORIZONTAL,

variable=self.app.thickness,
```

```
command=self.app.on_thickness_change)
        self.app.thickness_scale.pack(side=tk.LEFT,
fill=tk.X, expand=True)

        self.app.thickness_label =
ttk.Label(thickness_frame,

text=str(self.app.thickness.get()))
        self.app.thickness_label.pack(side=tk.RIGHT,
padx=(5, 0))

        # Checkbox untuk batasi bentuk
        ttk.Checkbutton(properties_frame, text="Batasi
Persegi",

variable=self.app.square_var).pack(anchor='w')
        ttk.Checkbutton(properties_frame, text="Batasi
Lingkaran",

variable=self.app.circle_var).pack(anchor='w')

    def create_transform_section(self):
        """Membuat section Transform"""
        transform_frame = ttk.LabelFrame(self.toolbar,
text="Transform", padding=10)
        transform_frame.pack(fill=tk.X, pady=(0, 10))

        # Translate
        translate_frame = ttk.Frame(transform_frame)
        translate_frame.pack(fill=tk.X, pady=(0, 10))

        ttk.Label(translate_frame,
text="Translate:").pack(anchor='w')

        tx_frame = ttk.Frame(translate_frame)
        tx_frame.pack(fill=tk.X, pady=2)
        ttk.Label(tx_frame, text="X:").pack(side=tk.LEFT)
        self.app.translate_x = ttk.Entry(tx_frame, width=8)
        self.app.translate_x.pack(side=tk.LEFT, padx=5)

        ty_frame = ttk.Frame(translate_frame)
        ty_frame.pack(fill=tk.X, pady=2)
        ttk.Label(ty_frame, text="Y:").pack(side=tk.LEFT)
        self.app.translate_y = ttk.Entry(ty_frame, width=8)
        self.app.translate_y.pack(side=tk.LEFT, padx=5)

        ttk.Button(translate_frame, text="Apply
Translation",

command=self.app.apply_translation).pack(fill=tk.X,
pady=(5,0))

        # Scale
        scale_frame = ttk.Frame(transform_frame)
        scale_frame.pack(fill=tk.X, pady=(0, 10))
```

```
        ttk.Label(scale_frame,
text="Scale:").pack(anchor='w')

        sx_frame = ttk.Frame(scale_frame)
        sx_frame.pack(fill=tk.X, pady=2)
        ttk.Label(sx_frame, text="X:").pack(side=tk.LEFT)
        self.app.scale_x = ttk.Entry(sx_frame, width=8)
        self.app.scale_x.pack(side=tk.LEFT, padx=5)

        sy_frame = ttk.Frame(scale_frame)
        sy_frame.pack(fill=tk.X, pady=2)
        ttk.Label(sy_frame, text="Y:").pack(side=tk.LEFT)
        self.app.scale_y = ttk.Entry(sy_frame, width=8)
        self.app.scale_y.pack(side=tk.LEFT, padx=5)

        ttk.Button(scale_frame, text="Apply Scale",

command=self.app.apply_scale).pack(fill=tk.X, pady=(5,0))

        # Rotate
        rotate_frame = ttk.Frame(transform_frame)
        rotate_frame.pack(fill=tk.X, pady=(0, 10))

        ttk.Label(rotate_frame,
text="Rotate:").pack(anchor='w')

        angle_frame = ttk.Frame(rotate_frame)
        angle_frame.pack(fill=tk.X, pady=2)
        ttk.Label(angle_frame,
text="Angle:").pack(side=tk.LEFT)
        self.app.rotate_angle = ttk.Entry(angle_frame,
width=8)
        self.app.rotate_angle.pack(side=tk.LEFT, padx=5)
        ttk.Label(angle_frame,
text="degrees").pack(side=tk.LEFT)

        ttk.Button(rotate_frame, text="Apply Rotation",

command=self.app.apply_rotation).pack(fill=tk.X, pady=(5,0))

        # Mirror
        mirror_frame = ttk.Frame(transform_frame)
        mirror_frame.pack(fill=tk.X, pady=(0, 10))
        ttk.Label(mirror_frame,
text="Mirror:").pack(anchor='w')
        mirror_buttons = ttk.Frame(mirror_frame)
        mirror_buttons.pack(fill=tk.X, pady=2)
        ttk.Button(mirror_buttons, text="Horizontal",
                   command=lambda:
self.app.apply_mirror("y")).pack(side=tk.LEFT, expand=True,
padx=2)
        ttk.Button(mirror_buttons, text="Vertical",
                   command=lambda:
self.app.apply_mirror("x")).pack(side=tk.LEFT, expand=True,
padx=2)
```

```
        # Shear
        shear_frame = ttk.Frame(transform_frame)
        shear_frame.pack(fill=tk.X, pady=(0, 10))
        ttk.Label(shear_frame,
text="Shear:").pack(anchor='w')

        shx_frame = ttk.Frame(shear_frame)
        shx_frame.pack(fill=tk.X, pady=2)
        ttk.Label(shx_frame, text="X:").pack(side=tk.LEFT)
        self.app.shear_x = ttk.Entry(shx_frame, width=8)
        self.app.shear_x.pack(side=tk.LEFT, padx=5)

        shy_frame = ttk.Frame(shear_frame)
        shy_frame.pack(fill=tk.X, pady=2)
        ttk.Label(shy_frame, text="Y:").pack(side=tk.LEFT)
        self.app.shear_y = ttk.Entry(shy_frame, width=8)
        self.app.shear_y.pack(side=tk.LEFT, padx=5)

        ttk.Button(shear_frame, text="Apply Shear",

command=self.app.apply_shear).pack(fill=tk.X, pady=(5,0))

    def create_clear_button(self):
        """Membuat tombol Clear Canvas"""
        clear_btn = ttk.Button(self.toolbar, text="Bersihkan
Kanvas",
                               command=self.app.clear_canvas)
        clear_btn.pack(fill=tk.X, pady=20)

    def create_transform_frame(self):
        """Buat frame untuk transformasi"""
        transform_frame = ttk.LabelFrame(self.right_frame,
text="Transformations", padding=5)
        transform_frame.pack(fill=tk.X, padx=5, pady=5)

        # Translation
        translate_frame = ttk.Frame(transform_frame)
        translate_frame.pack(fill=tk.X, pady=2)
        ttk.Label(translate_frame,
text="Translate:").pack(side=tk.LEFT)
        self.app.translate_x = ttk.Entry(translate_frame,
width=5)
        self.app.translate_x.pack(side=tk.LEFT, padx=2)
        self.app.translate_y = ttk.Entry(translate_frame,
width=5)
        self.app.translate_y.pack(side=tk.LEFT, padx=2)
        ttk.Button(translate_frame, text="Apply",

command=self.app.apply_translation).pack(side=tk.LEFT,
padx=2)

        # Scale
        scale_frame = ttk.Frame(transform_frame)
        scale_frame.pack(fill=tk.X, pady=2)
        ttk.Label(scale_frame,
text="Scale:").pack(side=tk.LEFT)
```

```
        self.app.scale_x = ttk.Entry(scale_frame, width=5)
        self.app.scale_x.pack(side=tk.LEFT, padx=2)
        self.app.scale_y = ttk.Entry(scale_frame, width=5)
        self.app.scale_y.pack(side=tk.LEFT, padx=2)
        ttk.Button(scale_frame, text="Apply",

command=self.app.apply_scale).pack(side=tk.LEFT, padx=2)

        # Rotate
        rotate_frame = ttk.Frame(transform_frame)
        rotate_frame.pack(fill=tk.X, pady=2)
        ttk.Label(rotate_frame,
text="Rotate:").pack(side=tk.LEFT)
        self.app.rotate_angle = ttk.Entry(rotate_frame,
width=5)
        self.app.rotate_angle.pack(side=tk.LEFT, padx=2)
        ttk.Button(rotate_frame, text="Apply",

command=self.app.apply_rotation).pack(side=tk.LEFT, padx=2)

        # Mirror
        mirror_frame = ttk.Frame(transform_frame)
        mirror_frame.pack(fill=tk.X, pady=2)
        ttk.Label(mirror_frame,
text="Mirror:").pack(side=tk.LEFT)
        ttk.Button(mirror_frame, text="Horizontal",
                   command=lambda:
self.app.apply_mirror("y")).pack(side=tk.LEFT, padx=2)
        ttk.Button(mirror_frame, text="Vertical",
                   command=lambda:
self.app.apply_mirror("x")).pack(side=tk.LEFT, padx=2)

        # Shear
        shear_frame = ttk.Frame(transform_frame)
        shear_frame.pack(fill=tk.X, pady=2)
        ttk.Label(shear_frame,
text="Shear:").pack(side=tk.LEFT)
        self.app.shear_x = ttk.Entry(shear_frame, width=5)
        self.app.shear_x.pack(side=tk.LEFT, padx=2)
        self.app.shear_y = ttk.Entry(shear_frame, width=5)
        self.app.shear_y.pack(side=tk.LEFT, padx=2)
        ttk.Button(shear_frame, text="Apply",

command=self.app.apply_shear).pack(side=tk.LEFT, padx=2)

        # Fill threshold
        threshold_frame = ttk.Frame(transform_frame)
        threshold_frame.pack(fill=tk.X, pady=2)
        ttk.Label(threshold_frame, text="Fill
Threshold:").pack(side=tk.LEFT)
        threshold_scale = ttk.Scale(threshold_frame,
from_=0, to=255,

variable=self.app.fill_threshold, orient=tk.HORIZONTAL)
        threshold_scale.pack(side=tk.LEFT, fill=tk.X,
expand=True, padx=2)
```

BAB IV
KESIMPULAN DAN SARAN

4.1. Kesimpulan

Berdasarkan proses perancangan dan implementasi yang telah dilakukan, dapat disimpulkan bahwa aplikasi grafika komputer 2D berbasis Python dan Tkinter ini berhasil dikembangkan dengan baik. Aplikasi memungkinkan pengguna untuk menggambar berbagai bentuk geometri dasar seperti garis, persegi, lingkaran, segitiga, bintang, dan segi

lima dengan cara interaktif melalui antarmuka pengguna yang intuitif. Selain itu, aplikasi ini juga dilengkapi dengan berbagai fitur transformasi dua dimensi seperti translasi, rotasi, skala, pencerminan (mirror), dan shear. Seluruh proses penggambaran dan manipulasi objek dapat dilakukan dengan mudah menggunakan mouse dan input yang disediakan pada toolbar.

Penggunaan pendekatan modular dalam pengembangan program terbukti efektif dalam menjaga keteraturan dan skalabilitas kode. Setiap fungsi utama dipisahkan dalam modul yang berbeda, seperti pengelolaan canvas, pengolahan event, transformasi bentuk, dan pembuatan antarmuka. Dengan demikian, struktur program menjadi lebih mudah dipahami, dikembangkan, dan diuji secara terpisah. Aplikasi ini juga telah mendemonstrasikan pemahaman yang baik terhadap konsep-konsep dasar dalam grafika komputer, seperti output primitif, transformasi geometri, dan interaksi pengguna.

4.2. Saran

Untuk pengembangan lebih lanjut, terdapat beberapa saran yang dapat dijadikan pertimbangan agar aplikasi menjadi lebih lengkap dan user-friendly. Pertama, aplikasi dapat ditambahkan fitur penyimpanan dan pembukaan file gambar dalam format umum seperti PNG atau JPEG, agar hasil kerja pengguna dapat disimpan dan dibuka kembali. Kedua, penambahan fitur undo dan redo akan sangat membantu pengguna dalam mengoreksi kesalahan tanpa perlu menghapus ulang secara manual. Ketiga, tampilan antarmuka dapat dibuat lebih dinamis dan responsif, termasuk dengan menambahkan ikon atau tooltip agar pengguna lebih mudah memahami fungsi setiap alat.

Selain itu, pengembangan ke arah penggunaan sistem koordinat kartesius secara eksplisit juga dapat menambah nilai edukatif aplikasi ini, terutama dalam konteks pembelajaran grafika komputer. Terakhir, untuk pengalaman pengguna yang lebih baik, dukungan penggunaan stylus (pena digital) dan peningkatan performa render melalui integrasi dengan pustaka grafis yang lebih tinggi seperti Pygame atau OpenGL dapat menjadi langkah selanjutnya dalam pengembangan.