LAPORAN PRAKTIKUM

TUGAS BESAR

GRAFIKA DAN KOMPUTASI VISUAL

Dibuat untuk memenuhi tugas besar Praktikum Grafika dan Komputasi Visual

Asisten Praktikum: Miriam Stefani Abigail Hutapea, Pujiani Rahayu Agustin

Oleh:

[Bramantyo Kunni N]	[24060123130091]
[Bagus Athallah]	[24060123120002]
[Agathan Khairy B. L]	[24060123140129]
[Al Vanza Aries R]	[24060123140169]

DEPARTEMEN INFORMATIKA
FAKULTAS SAINS DAN MATEMATIKA
UNIVERSITAS DIPONEGORO

2025

DAFTAR ISI

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dalam mata kuliah Grafika dan Komputasi Visual, mahasiswa diberikan bimbingan untuk menggunakan OpenGL sebagai alat dalam membuat proyek dengan fokus pada output visual. Tujuan dari mata kuliah ini adalah agar mahasiswa dapat berkreasi dan mengembangkan proyek yang menggabungkan konsep-konsep teori yang telah dipelajari selama perkuliahan. Sebagai bagian dari tugas besar dalam mata kuliah ini, kami membuat sebuah model Gerobak Juara sebagai proyek kami dalam memenuhi persyaratan Grafika dan Komputasi Visual. Proyek Gerobak Juara merupakan proyek yang menantang karena melibatkan implementasi konsep konsep dasar grafika komputer seperti pemodelan 3D, pencahayaan, tekstur, dan transformasi objek. Dengan menggunakan OpenGL, sebuah library grafika 3D, mahasiswa dapat mengimplementasikan berbagai teknik grafika untuk membuat Gerobak Juara 3D yang realistis dan interaktif. Melalui proyek ini, mahasiswa dapat memperdalam pemahaman mereka tentang konsep-konsep dasar dalam grafika komputer dan komputasi visual, serta meningkatkan keterampilan pemrograman dalam lingkungan OpenGL/Dev CPP. Selain itu, proyek ini juga memberikan kesempatan bagi mahasiswa untuk berkreasi, mengembangkan kreativitas, dan mengaplikasikan pengetahuan yang telah dipelajari dalam sebuah proyek nyata.

OpenGL merupakan API (Application Programing Interface) yang paling populer untuk pemgembangan grafis 3D, karena kemampuannya yang tinggi dalam rendering grafis dengan performa optimal dan fleksibilitas yang besar. Pengembangan sebuah perangkat lunak 3D sederhana dapat menggunakan OpenGL dan bahasa pemrograman C++, yang mengimplementasikan berbagai konsep dasar grafika seperti transformasi geometri (translasi, rotasi, dan skala), pewarnaan objek (coloring), proyeksi perspektif, dan penggabungan objek primitif menjadi bentuk kompleks. Dalam game ini, berbagai objek seperti gerobak, karakter manusia, rumah, pagar, pohon, jalan raya, dan rintangan (obstacle) dirancang dari gabungan primitive shapes seperti kubus, silinder, dan bola. Implementasi ini juga mencakup penanganan input keyboard untuk memanipulasi objek secara real-time. Dengan menggabungkan elemen-elemen ini, perangkat lunak yang dikembangkan memberikan pengalaman dasar dalam pengembangan grafis 3D, serta menjadi dasar yang kuat untuk proyek yang lebih kompleks di masa depan.

1.2 Rumusan Masalah

1.2.1 Bagaimana mengimplementasikan model Gerobak Juara 3D yang realistis menggunakan OpenGL dan C++?

1.2.2 Bagaimana menerapkan teknik pencahayaan dan shading untuk menghasilkan visualisasi yang realistis pada model Gerobak Juara?

1.2.3 Bagaimana mengelola dan mengoptimalkan kontrol interaktif dalam lingkungan grafis 3D menggunakan OpenGL?

1.2.4 Bagaimana memastikan proyek ini dapat menjadi dasar yang kuat untuk pengembangan proyek grafis 3D yang lebih kompleks di masa depan?

1.3 Tujuan

1.3.1 Mengimplementasikan model Gerobak Juara 3D yang realistis menggunakan OpenGL dan C++

1.3.2 Menerapkan teknik pencahayaan dan shading untuk menghasilkan visualisasi yang realistis pada model Gerobak Juara

1.3.3 Mengelola dan mengoptimalkan kontrol interaktif dalam lingkungan grafis 3D menggunakan OpenGL

1.3.4 Mengembangkan fondasi yang kuat untuk pengembangan proyek grafis 3D yang lebih kompleks di masa depan dengan menerapkan teknik dan konsep yang telah dipelajari.

BAB II

DASAR TEORI

2.1 Komputasi Grafis

Komputasi grafis adalah cabang ilmu komputer yang mempelajari teknik, algoritma, dan perangkat lunak untuk menghasilkan, memanipulasi, dan menampilkan gambar atau visualisasi secara digital (Posada et al., 2015). Komputasi grafis memungkinkan representasi objek dua dimensi (2D) maupun tiga dimensi (3D) pada layar komputer dengan memanfaatkan proses matematis dan geometris untuk mengatur bentuk, warna, pencahayaan, serta perspektif objek. Bidang ini memainkan peran penting dalam berbagai aplikasi, mulai dari desain grafis, animasi, visualisasi ilmiah, hingga simulasi dan pengembangan game.

Dalam implementasinya, komputasi grafis melibatkan penggunaan perangkat lunak khusus seperti OpenGL atau DirectX, yang menyediakan sekumpulan fungsi untuk membangun, memodifikasi, dan merender objek grafis secara efisien. Komputasi grafis juga mencakup konsep dasar seperti sistem koordinat, transformasi geometri (translasi, rotasi, skala), proyeksi kamera, serta pengolahan warna dan cahaya. Dengan adanya komputasi grafis, komputer dapat digunakan tidak hanya untuk pengolahan data numerik, tetapi juga untuk menghasilkan visualisasi yang lebih interaktif dan informatif bagi pengguna.

2.2 OpenGL

OpenGL (Open Graphics Library) adalah sebuah Application Programming Interface (API) lintas platform dan bahasa pemrograman yang digunakan untuk membuat aplikasi grafis komputer 2D dan 3D (Erwinsyah et al., 2019). Dikembangkan pertama kali oleh Silicon Graphics Inc. pada tahun 1992, OpenGL telah menjadi standar industri yang didukung oleh berbagai sistem operasi dan perangkat keras grafis. OpenGL menggunakan model pemrograman prosedural dengan prinsip state machine, di mana fungsi-fungsi memanipulasi

state internal untuk menghasilkan output grafis. API ini menyediakan fungsi-fungsi dasar untuk operasi rendering seperti transformasi geometri, proyeksi, pencahayaan, tekstur, dan blending yang memungkinkan pengembang membangun aplikasi grafis kompleks dari primitive sederhana.

Arsitektur OpenGL terdiri dari pipeline grafis yang memproses data geometris dan pixel melalui serangkaian tahapan untuk menghasilkan tampilan akhir pada framebuffer. Mulai dari OpenGL 2.0, diperkenalkan shader yang memungkinkan pengembang menulis program kustom dalam bahasa GLSL (OpenGL Shading Language) untuk memanipulasi vertex dan fragment secara fleksibel. Pendekatan ini memberikan kontrol yang lebih besar terhadap proses rendering dan memungkinkan efek visual yang lebih canggih. OpenGL terus berkembang dengan versi terbaru yang mendukung teknik rendering modern seperti compute shader, tesselasi, dan instancing, menjadikannya pilihan yang relevan untuk pengembangan berbagai aplikasi grafis dari visualisasi ilmiah, CAD, game, hingga virtual reality.

2.3 Transformasi Geometri

Transformasi geometri dalam grafika komputer merupakan serangkaian operasi matematika yang digunakan untuk mengubah posisi, orientasi, dan ukuran objek dalam ruang 2D atau 3D. Terdapat tiga jenis transformasi dasar yang menjadi fondasi grafika komputer, yaitu translasi (perpindahan posisi), rotasi (perputaran objek di sekitar sumbu), dan skala (pembesaran atau pengecilan dimensi objek). Dalam implementasinya, transformasi geometri ini direpresentasikan dalam bentuk operasi matriks, yang memungkinkan manipulasi efisien pada titik-titik (vertices) yang membentuk objek grafis. OpenGL menggunakan matriks 4×4 untuk melakukan transformasi, di mana operasi translasi, rotasi, dan skala dapat dikombinasikan menjadi satu matriks transformasi tunggal melalui perkalian matriks, sehingga memudahkan pengelolaan objek kompleks dalam ruang 3D.

Implementasi transformasi geometri dalam OpenGL dilakukan melalui fungsi-fungsi seperti glTranslatef(), glRotatef(), dan glScalef(), yang secara internal memodifikasi matriks model-view untuk mengubah posisi dan bentuk objek sebelum dirender. Konsep penting lainnya dalam transformasi geometri adalah sistem koordinat dan hierarki transformasi. OpenGL menggunakan prinsip stack matriks yang memungkinkan pengembang untuk menyimpan dan memulihkan state transformasi, sehingga transformasi dapat diberlakukan secara hierarkis pada objek yang memiliki hubungan parent-child. Sebagai contoh, dalam pembuatan model gerobak pada game, transformasi rotasi pada roda dapat diaplikasikan relatif terhadap posisi gerobak, sehingga ketika gerobak bergerak (translasi), rodanya tetap berotasi pada posisi yang tepat. Penguasaan transformasi geometri menjadi kunci dalam menciptakan animasi yang realistis dan interaksi objek yang meyakinkan dalam aplikasi grafika komputer.

2.4 Pencahayaan dan Material

Pencahayaan dalam grafika komputer merupakan proses simulasi interaksi cahaya dengan permukaan objek untuk menciptakan kesan realistis pada visual 3D (Krupiński, 2021). Model pencahayaan yang umum digunakan adalah model Phong, yang membagi komponen cahaya menjadi tiga: ambient (cahaya latar yang menerangi seluruh adegan secara merata), diffuse (cahaya yang tersebar ke segala arah saat mengenai permukaan kasar), dan specular (cahaya yang terpantul pada sudut tertentu, menciptakan efek mengkilap pada permukaan). OpenGL menyediakan implementasi pencahayaan melalui pengaturan sumber cahaya (glLight*()) dengan parameter seperti posisi, intensitas, warna, dan atenuasi (pelemahan intensitas berdasarkan jarak), yang kemudian dikalkulasikan terhadap vektor normal permukaan untuk menentukan bagaimana cahaya diproses pada tiap titik permukaan objek.

Material dalam konteks grafika komputer mendefinisikan bagaimana permukaan objek berinteraksi dengan cahaya yang mengenainya, menentukan karakteristik visual objek seperti warna, kilau, dan transparansi. Dalam OpenGL, properti material ditetapkan menggunakan fungsi glMaterial*() yang

memungkinkan pengaturan respon permukaan terhadap berbagai jenis cahaya. Parameter material meliputi ambient (warna objek di bawah cahaya latar), diffuse (warna dasar objek di bawah pencahayaan langsung), specular (warna kilau pada permukaan), shininess (tingkat kekilauan yang menentukan luas area pantulan specular), dan emission (cahaya yang dipancarkan objek sendiri). Kombinasi yang tepat antara pencahayaan dan material memungkinkan penciptaan berbagai efek visual seperti logam mengkilap, plastik kusam, atau kain berpori, yang sangat penting dalam meningkatkan realisme scene 3D. Dalam implementasi game sederhana, pencahayaan yang dirancang dengan baik dapat secara signifikan meningkatkan kualitas visual meskipun menggunakan model objek yang relatif sederhana.

2.5 Deteksi Tabrakan (Collision Detection)

Deteksi tabrakan (collision detection) adalah proses komputasi untuk menentukan apakah dua atau lebih objek dalam ruang virtual bersinggungan atau bertumpuk (Heo et al., 2019). Konsep ini menjadi fundamental dalam pengembangan game dan simulasi interaktif, karena memungkinkan objek merespons secara realistis terhadap interaksi fisik dengan objek lainnya, seperti tumbukan kendaraan dengan rintangan, pergerakan karakter yang dibatasi oleh dinding, atau pengumpulan item. Dalam implementasinya, deteksi tabrakan dapat dibagi menjadi dua fase: broad phase (fase luas) yang menyaring pasangan objek yang berpotensi bertabrakan menggunakan struktur data spasial seperti spatial hashing, quad-tree, atau bounding volume hierarchies (BVH), dan narrow phase (fase sempit) yang melakukan kalkulasi tabrakan secara presisi pada objek-objek yang telah teridentifikasi pada fase pertama. Metode deteksi tabrakan yang umum digunakan meliputi bounding volumes (seperti axis-aligned bounding box/AABB, bounding sphere, atau oriented bounding box/OBB), ray casting, dan pengujian separating axis theorem (SAT).

Pada aplikasi berbasis OpenGL, implementasi deteksi tabrakan umumnya dilakukan di luar pipeline rendering utama, karena OpenGL sendiri tidak menyediakan fungsi bawaan untuk deteksi tabrakan. Pengembang harus

mengimplementasikan algoritma deteksi tabrakan secara terpisah, biasanya dengan memanfaatkan data geometri yang sama yang digunakan untuk rendering objek. Untuk game sederhana, pendekatan yang efisien adalah menggunakan bounding volumes sederhana seperti AABB atau bounding sphere, yang menyederhanakan bentuk objek kompleks menjadi bentuk geometris sederhana untuk mempercepat perhitungan. Saat tabrakan terdeteksi, sistem kemudian dapat menerapkan respons yang sesuai, seperti memantulkan objek, menghentikan pergerakan, mengurangi nilai kesehatan karakter, atau memicu event game lainnya. Optimasi deteksi tabrakan menjadi krusial dalam aplikasi real-time seperti game, di mana perhitungan harus diselesaikan dalam waktu singkat (biasanya kurang dari 16.7 millisecond untuk mencapai 60 fps) tanpa mengorbankan akurasi yang diperlukan untuk gameplay yang memuaskan.

BAB III

PEMBAHASAN

3.1 Fungsi drawGerobak()

```
void drawGerobak() {
    glPushMatrix();
    glScalef(0.3f, 0.3f, 0.3f);
    glTranslatef(-9.0f, 1.5f + (carYOffset / 0.3f), -8.9f); //
Menaikkan posisi Y untuk menghindari tumpang tindih
    glRotatef(90.0f, 0.0f, 1.0f, 0.0f); // Orientasi utama
gerobak

    // Terapkan kemiringan saat belok (pada sumbu Z LOKAL
gerobak)
    // Sumbu Z lokal gerobak adalah sumbu maju/mundurnya setelah
rotasi orientasi
    glRotatef(carTiltAngle, 0.0f, 1.0f, 0.0f);
    Vertex block1[8] = {
        { -1.5, 0.6, 9.85 }, {  1.5, 0.6, 9.85 }, {  2.0, 3.0,
10.3 }, { -2.0, 3.0, 10.3 },
        { -1.5, 0.6, 10.05 }, {  1.5, 0.6, 10.05 }, {  2.0, 3.0,
10.5 }, { -2.0, 3.0, 10.5 }
    }; drawBlockCustom(block1, 216, 64, 64);
    Vertex block2[8] = {
        {  1.3, 0.6, 8.05 }, {  1.5, 0.6, 8.05 }, {  2.0, 3.0,
7.6 }, {  1.8, 3.0, 7.6 },
        {  1.3, 0.6, 9.85 }, {  1.5, 0.6, 9.85 }, {  2.0, 3.0,
10.3 }, {  1.8, 3.0, 10.3 }
    }; drawBlockCustom(block2, 216, 64, 64);
    Vertex block3[8] = {
        { -1.5, 0.6, 8.05 }, { -1.45, 0.6, 8.05 }, { -1.75, 3.0,
7.6 }, { -2.0, 3.0, 7.6  },
        { -1.5, 0.6, 9.85 }, { -1.45, 0.6, 9.85 }, { -1.75, 3.0,
10.3 },{ -2.0, 3.0, 10.3 }
    }; drawBlockCustom(block3, 216, 64, 64);
    Vertex block4[8] = {
        { -1.5, 0.4, 7.85 }, {  1.5, 0.4, 7.85 }, {  1.5, 0.6,
7.85 }, { -1.5, 0.6, 7.85 },
        { -1.5, 0.4, 10.05 }, {  1.5, 0.4, 10.05 }, {  1.5, 0.6,
10.05 }, { -1.5, 0.6, 10.05 }
    }; drawBlockCustom(block4, 163, 29, 29);
    Vertex block5[8] = {
        { -1.5, 0.6, 7.85 }, {  1.5, 0.6, 7.85 }, {  2.0, 3.0,
7.4 }, { -2.0, 3.0, 7.4 },
        { -1.5, 0.6, 8.05 }, {  1.5, 0.6, 8.05 }, {  2.0, 3.0,
7.6 }, { -2.0, 3.0, 7.6 }
    }; drawBlockCustom(block5, 216, 64, 64);
    Vertex block6[8] = {
        { -2.5, 2.8, 10.3 }, { -1.9, 2.5, 10.2 }, { -2.0, 3.0,
10.3 }, { -2.5, 3.0, 10.3 },
        { -2.5, 2.8, 10.5 }, { -1.9, 2.5, 10.4 }, { -2.0, 3.0,
10.5 }, { -2.5, 3.0, 10.5 }
    }; drawBlockCustom(block6, 255, 255, 0);
    Vertex block7[8] = {
```

```
        { -2.5, 2.8, 7.4 }, { -1.9, 2.5, 7.5 }, { -2.0, 3.0, 7.4
}, { -2.5, 3.0, 7.4 },
        { -2.5, 2.8, 7.6 }, { -1.9, 2.5, 7.7 }, { -2.0, 3.0, 7.6
}, { -2.5, 3.0, 7.6 }
    }; drawBlockCustom(block7, 255, 255, 0);
    Vertex block8[8] = {
        { -2.5, 2.8, 7.6 }, { -2.3, 2.8, 7.6 }, { -2.3, 3.0, 7.6
}, { -2.5, 3.0, 7.6 },
        { -2.5, 2.8, 10.3 }, { -2.3, 2.8, 10.3 }, { -2.3, 3.0,
10.3 }, { -2.5, 3.0, 10.3 }
    }; drawBlockCustom(block8, 255, 255, 0);

    glPushMatrix(); glTranslatef(0.5, 0.3, 7.4); drawWheel1();
glPopMatrix();
    glPushMatrix(); glTranslatef(0.5, 0.3, 10.5); drawWheel1();
glPopMatrix();
    glPushMatrix(); glTranslatef(-2.0, 0.0, 8.95); drawWheel2();
glPopMatrix();

    glPushMatrix();
        glTranslatef(0.0, 0.3, 10.5);
        glColor3f(0.8f, 0.8f, 0.8f);
      glRotatef(90.0f, 0.0f, 1.0f, 0.0f);
        drawCylinder(0.1, 0.1, 3.1, 20, 1);
    glPopMatrix();

    Vertex block9[8] = {
        { -2.15, -0.05, 9.2 }, { -1.85, -0.05, 9.2 }, {  -1.2,
0.4, 9.2 }, { -1.5, 0.4, 9.2 },
        { -2.15, -0.05, 9.3 }, { -1.85, -0.05, 9.3 }, { -1.2,
0.4, 9.3 }, { -1.5, 0.4, 9.3 }
    }; drawBlockCustom(block9, 163, 29, 29);
    Vertex block10[8] = {
        { -2.15, -0.05, 8.6 }, { -1.85, -0.05, 8.6 }, {  -1.2,
0.4, 8.6 }, { -1.5, 0.4, 8.6 },
        { -2.15, -0.05, 8.7 }, { -1.85, -0.05, 8.7 }, { -1.2,
0.4, 8.7 }, { -1.5, 0.4, 8.7 }
    }; drawBlockCustom(block10, 163, 29, 29);

    glPushMatrix();
        glTranslatef(-2.0, 0.0, 8.7);
        glColor3f(0.8f, 0.8f, 0.8f);
        drawCylinder(0.05, 0.05, 0.5, 20, 1);
    glPopMatrix();

      glPushMatrix();
          // Penyesuaian posisi agar orang duduk pas di dalam
gerobak
          glTranslatef(0.0f, 1.8f, 9.0f); // Posisi lebih rendah
di gerobak
          glRotatef(-90.0f, 0.0f, 1.0f, 0.0f); // Putar 90°
berlawanan arah jarum jam (sumbu Y)
          glScalef(0.8f, 0.8f, 0.8f); // Sesuaikan skala orang
agar tidak terlalu besar

          // Buat animasi untuk orang bergerak saat gerobak
berbelok
          float personTiltFactor = -0.3f; // Arah berlawanan
```

```
dengan kemiringan gerobak
          glRotatef(carTiltAngle * personTiltFactor, 0.0f, 0.0f,
1.0f);

          buatOrang(); // Gambar orangnya
      glPopMatrix();

    glPopMatrix();
}
```

Fungsi drawGerobak() mengimplementasikan pembuatan objek gerobak 3D menggunakan OpenGL dengan pendekatan pemodelan berbasis vertex. Kode ini dimulai dengan menerapkan transformasi dasar, termasuk penskalaan (0.3f pada semua sumbu) untuk menyesuaikan ukuran gerobak, translasi untuk penempatan awal, dan rotasi 90 derajat pada sumbu Y yang menentukan orientasi gerobak di dunia virtual. Aspek dinamis diterapkan melalui variabel carTiltAngle yang memberikan efek kemiringan saat gerobak berbelok, menciptakan simulasi fisika sederhana yang meningkatkan realisme pergerakan. Struktur utama gerobak dibangun dari sepuluh blok kustom, masing-masing didefinisikan sebagai array vertex tiga dimensi yang diproses melalui fungsi drawBlockCustom() dengan parameter warna RGB yang berbeda, seperti merah kecoklatan (216,64,64) untuk badan utama dan kuning (255,255,0) untuk aksen tertentu.

Komponen-komponen tambahan gerobak mencakup sistem roda yang diimplementasikan melalui fungsi drawWheel1() dan drawWheel2(), serta silinder penghubung yang dibuat menggunakan drawCylinder() dengan parameter yang mengontrol diameter, panjang, dan jumlah segmen. Elemen interaktif ditambahkan dalam bentuk figur pengemudi yang diposisikan di dalam gerobak menggunakan translasi, rotasi, dan penskalaan terpisah. Figur ini menampilkan animasi responsif dengan faktor kemiringan berlawanan (personTiltFactor) yang membuat pengemudi miring ke arah berlawanan saat gerobak berbelok, menciptakan ilusi pergerakan alami. Struktur hierarkis transformasi dikelola melalui pasangan glPushMatrix() dan glPopMatrix() yang memungkinkan isolasi transformasi untuk komponen spesifik, sehingga perubahan pada satu bagian tidak memengaruhi bagian lain. Penggunaan teknik pemodelan berbasis vertex dan transformasi geometri ini mendemonstrasikan prinsip dasar komputasi grafis dalam menciptakan objek 3D interaktif dan animasi untuk aplikasi game.

3.2 Fungsi buatOrang()

```
// Fungsi-fungsi untuk membuat orang
void BuatKepala() {
    glPushMatrix();
    glTranslatef(0.0f, 6.0f, 0.0f);
    glColor3f(1.0f, 0.8f, 0.6f);
    glutSolidSphere(2.0f, 50, 50);
    glPopMatrix();
}

void BuatTopi() {
    glPushMatrix();
    glTranslatef(1.0f, 7.5f, 0.0f);
    glColor3f(0.0f, 0.0f, 1.0f);
    glScalef(3.0f, 0.3f, 2.2f);
    glutSolidCube(2.0f);
    glPopMatrix();
}

void BuatLeher() {
    glPushMatrix();
    glTranslatef(0.0f, 4.0f, 0.0f);
    glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
    glColor3f(1.0f, 0.8f, 0.6f);
    GLUquadricObj *quadratic = gluNewQuadric();
    gluCylinder(quadratic, 0.5f, 0.5f, 1.0f, 32, 32);
    gluDeleteQuadric(quadratic);
    glPopMatrix();
}

void Badan() {
    glPushMatrix();
    glScalef(0.8f, 1.1f, 0.5f);

    // Depan
    glBegin(GL_QUADS);
        glVertex3f(-3.0f, -3.0f, 3.0f);
        glVertex3f(3.0f, -3.0f, 3.0f);
        glVertex3f(3.0f, 3.0f, 3.0f);
        glVertex3f(-3.0f, 3.0f, 3.0f);
    glEnd();

    // Belakang
    glBegin(GL_QUADS);
        glVertex3f(-3.0f, -3.0f, -3.0f);
        glVertex3f(-3.0f, 3.0f, -3.0f);
        glVertex3f(3.0f, 3.0f, -3.0f);
        glVertex3f(3.0f, -3.0f, -3.0f);
    glEnd();

    // Atas
    glBegin(GL_QUADS);
        glVertex3f(-3.0f, 3.0f, -3.0f);
        glVertex3f(-3.0f, 3.0f, 3.0f);
        glVertex3f(3.0f, 3.0f, 3.0f);
        glVertex3f(3.0f, 3.0f, -3.0f);
    glEnd();
```

```
    // Bawah
    glBegin(GL_QUADS);
        glVertex3f(-3.0f, -3.0f, -3.0f);
        glVertex3f(3.0f, -3.0f, -3.0f);
        glVertex3f(3.0f, -3.0f, 3.0f);
        glVertex3f(-3.0f, -3.0f, 3.0f);
    glEnd();

    // Kiri
    glBegin(GL_QUADS);
        glVertex3f(-3.0f, -3.0f, -3.0f);
        glVertex3f(-3.0f, -3.0f, 3.0f);
        glVertex3f(-3.0f, 3.0f, 3.0f);
        glVertex3f(-3.0f, 3.0f, -3.0f);
    glEnd();

    // Kanan
    glBegin(GL_QUADS);
        glVertex3f(3.0f, -3.0f, -3.0f);
        glVertex3f(3.0f, 3.0f, -3.0f);
        glVertex3f(3.0f, 3.0f, 3.0f);
        glVertex3f(3.0f, -3.0f, 3.0f);
    glEnd();

    glPopMatrix();
}

void BuatTangan(bool kiri) {
    glPushMatrix();
    float posisiX = (kiri ? -3.2f : 3.2f);
    glTranslatef(posisiX, 3.0f, 0.0f);

    // Lengan Atas
    glPushMatrix();
    glColor3f(1.0f, 0.0f, 0.0f);
    glScalef(0.7f, 1.6f, 0.7f);
    glTranslatef(0.0f, -1.0f, 0.0f);
    glutSolidCube(2.0f);
    glPopMatrix();

    // Lengan Bawah
    glPushMatrix();
    glColor3f(0.0f, 1.0f, 0.0f);
    glScalef(0.6f, 1.5f, 0.6f);
    glTranslatef(0.0f, -3.0f, 0.0f);
    glutSolidCube(2.0f);
    glPopMatrix();

    // Telapak Tangan
    glPushMatrix();
    glColor3f(0.0f, 0.0f, 1.0f);
    glScalef(0.5f, 0.6f, 0.5f);
    glTranslatef(0.0f, -10.1f, 0.0f);
    glutSolidCube(2.0f);
    glPopMatrix();

    glPopMatrix();
```

```
}

void BuatKakiMenekuk(bool kiri) {
    glPushMatrix();
    float posisiX = (kiri ? -1.2f : 1.2f);

    // Posisi awal kaki
    glTranslatef(posisiX, -3.0f, 0.0f);

    // Rotasi kaki untuk tekukan
    glRotatef(80.0f, 1.0f, 0.0f, 0.0f);

    // Paha
    glPushMatrix();
        glColor3f(1.0f, 0.0f, 0.0f);
        glScalef(1.1f, 2.0f, 1.1f);
        glTranslatef(0.0f, -1.0f, 0.0f);
        glutSolidCube(2.0f);
    glPopMatrix();

    // Betis (ditekuk)
    glPushMatrix();
        glTranslatef(0.0f, -4.0f, -2.0f);
        glRotatef(-70.0f, 1.0f, 0.0f, 0.0f); // Tekukan di lutut

        glColor3f(0.0f, 1.0f, 0.0f);
        glScalef(1.0f, 2.0f, 1.0f);
        glTranslatef(0.0f, -1.5f, 0.0f);
        glutSolidCube(2.0f);

        // Telapak kaki
        glPushMatrix();
            glTranslatef(0.0f, -2.5f, -0.7f);
            glRotatef(20.0f, 1.0f, 0.0f, 0.0f); // Sedikit
menekuk telapak

            glColor3f(0.0f, 0.0f, 1.0f);
            glScalef(1.0f, 0.7f, 1.7f);
            glutSolidCube(2.0f);
        glPopMatrix();
    glPopMatrix();

    glPopMatrix();
}

void buatOrang() {
    glPushMatrix();
    glTranslatef(0.0f, 2.0f, 0.0f);
    glScalef(0.3f, 0.3f, 0.3f);

    Badan();
    BuatTangan(true);
    BuatTangan(false);
    BuatKakiMenekuk(true);   // Kaki kiri menekuk
    BuatKakiMenekuk(false);  // Kaki kanan menekuk
    BuatKepala();
    BuatTopi();
    BuatLeher();
```

```
    glPopMatrix();
}
```

Fungsi buatOrang() merupakan fungsi utama yang menggabungkan seluruh bagian tubuh karakter manusia menggunakan transformasi geometri berbasis matriks dalam OpenGL. Fungsi ini dimulai dengan translasi ke atas sumbu Y sebesar 2.0 satuan dan penskalaan seragam sebesar 0.3 pada ketiga sumbu, yang berfungsi untuk menyesuaikan ukuran keseluruhan karakter dengan lingkungan dunia virtual. Bagian tubuh utama dibangun secara modular menggunakan kombinasi objek primitif seperti kubus (glutSolidCube), bola (glutSolidSphere), dan silinder (gluCylinder). Struktur modular ini memungkinkan penerapan transformasi lokal dengan memanfaatkan pasangan glPushMatrix() dan glPopMatrix(), yang menjaga agar setiap transformasi bagian tubuh tidak mempengaruhi bagian lain.

Bagian badan dibentuk melalui fungsi Badan() dengan pendekatan eksplisit berbasis vertex menggunakan glBegin(GL_QUADS), yang membentuk kubus berwarna dengan enam sisi berbeda. Tangan dibentuk melalui BuatTangan(bool kiri), dengan parameter boolean untuk menentukan sisi (kiri atau kanan). Setiap tangan terdiri atas tiga segmen: lengan atas berwarna merah, lengan bawah hijau, dan telapak biru, yang disusun dengan translasi vertikal dan penskalaan yang tepat untuk menciptakan bentuk tangan yang proporsional. Struktur tangan dibangun tanpa rotasi kompleks namun tetap menghasilkan efek volume tiga dimensi yang realistis.

Untuk bagian kaki, digunakan fungsi BuatKakiMenekuk(bool kiri) yang menerapkan rotasi kompleks untuk mensimulasikan postur duduk atau menekuk. Paha dimiringkan 80 derajat ke depan, kemudian betis ditekuk kembali ke belakang sebesar 70 derajat, menciptakan efek lekukan lutut yang alami. Telapak kaki ditambahkan dengan sedikit rotasi tambahan untuk menampilkan posisi kaki yang fleksibel. Kepala dibentuk dengan bola (glutSolidSphere) yang ditempatkan di atas badan, dilengkapi dengan topi datar berwarna biru yang dibentuk dari kubus pipih. Leher menggunakan silinder kecil untuk transisi anatomi antara

kepala dan badan. Seluruh struktur disusun secara hierarkis dan transformasi lokal yang tepat, mencerminkan prinsip dasar dalam grafika komputer 3D, yaitu komposisi objek kompleks dari bentuk primitif dengan transformasi geometris. Teknik ini juga menunjukkan pemanfaatan warna dan rotasi sebagai elemen visual penting untuk diferensiasi dan ekspresi karakter.

3.3 Fungsi drawGround()

```
void drawGround() {
    glDisable(GL_LIGHTING);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, _textureId);
    glColor3f(1.0f, 1.0f, 1.0f);

    float groundDepth = -0.2f; // Ground berada di bawah jalan
    float groundWidth = 200.0f; // Perlebar ground

    // Perpanjang ground di depan dan belakang gerobak
    float frontZ = carZ - 300.0f; // Perpanjang jauh ke depan
    float backZ = carZ + 100.0f; // Perpanjang ke belakang

    float groundY = groundDepth;

    glBegin(GL_QUADS);
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(-groundWidth, groundY, frontZ);

        glTexCoord2f(10.0f, 0.0f); // Repeat 10x di X
        glVertex3f(groundWidth, groundY, frontZ);

        glTexCoord2f(10.0f, 20.0f); // Repeat 20x di Z
        glVertex3f(groundWidth, groundY, backZ);

        glTexCoord2f(0.0f, 20.0f);
        glVertex3f(-groundWidth, groundY, backZ);
    glEnd();
    glDisable(GL_TEXTURE_2D);
    glEnable(GL_LIGHTING);

}
```

Fungsi drawGround() untuk merender permukaan tanah datar (ground plane) yang menjadi latar dasar dari lingkungan 3D. Fungsi ini secara eksplisit menonaktifkan pencahayaan (glDisable(GL_LIGHTING)) sebelum menggambar permukaan tanah agar warnanya tidak dipengaruhi oleh efek pencahayaan dinamis, dan kemudian mengaktifkan glEnable(GL_TEXTURE_2D) untuk memberi tahu OpenGL bahwa objek ground akan diberi tekstur 2D, glBindTexture(GL_TEXTURE_2D, _textureId) digunakan untuk memilih tektur

yang aktif berdasarkan _textureId. Setelah itu mengatur warna tanah menjadi putih (glColor3f(1.0f, 1.0f, 1.0f)) supaya teksture yang sudah dipilih terlihat.

Secara geometri, tanah dibentuk sebagai sebuah kuadrilateral besar (persegi panjang datar) yang memanjang dalam arah sumbu Z, sesuai dengan posisi carZ, yaitu posisi gerobak atau objek utama dalam dunia 3D. Panjang ground diperluas jauh ke depan (carZ - 300.0f) dan sedikit ke belakang (carZ + 100.0f) untuk menjaga kontinuitas tampilan seiring pergerakan kamera atau gerobak ke depan, menciptakan ilusi dunia yang luas dan terbuka. Lebar ground (groundWidth) disetel hingga 200 unit untuk menyesuaikan dengan lebar jalur atau lingkungan, dan seluruh permukaan digambar pada kedalaman tetap (groundDepth = -0.2f), sedikit di bawah level jalan, agar tidak terjadi tumpang tindih visual dengan objek jalan. Setelah penggambaran selesai, pencahayaan diaktifkan kembali (glEnable(GL_LIGHTING)) dan tekstur 2D dinonaktifkan kembali glDisable(GL_TEXTURE_2D), sehingga objek lain tetap mendapatkan pencahayaan realistis. Pendekatan ini menunjukkan pemahaman dasar tentang kontrol rendering state dan efisiensi dalam menggambar elemen statis besar di dalam dunia 3D.

3.5 Fungsi gambarAwan()

```
void gambarAwan(float x, float y, float z) {
    glColor3d(1.0, 1.0, 1.0); // set warna awan putih
    glPushMatrix();
    glTranslatef(x, y, z);
    glutSolidSphere(1.0, 50, 50);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(x + 1.0, y + 0.5, z);
    glutSolidSphere(1.5, 50, 50);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(x - 1.0, y + 0.5, z);
    glutSolidSphere(2, 50, 50);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(x + 0.5, y + 1.0, z);
    glutSolidSphere(1.0, 50, 50);
    glPopMatrix();
    glPushMatrix();
    glTranslatef(x - 0.5, y + 1.0, z);
    glutSolidSphere(1.0, 50, 50);
    glPopMatrix();
}
```

Fungsi gambarAwan() berfungsi untuk membentuk objek awan 3D dengan tampilan lembut dan natural menggunakan beberapa bola (spheres) putih yang digabungkan. Pemanggilan fungsi ini memerlukan tiga parameter posisi (x, y, z) yang menentukan lokasi awan dalam ruang 3D. Di dalam fungsi, transformasi posisi dilakukan menggunakan glTranslatef() sebelum menggambar setiap bola dengan glutSolidSphere(). Seluruh bola memiliki warna putih (glColor3d(1.0, 1.0, 1.0)), mencerminkan bentuk awan yang umum dalam ilustrasi kartun atau simulasi langit sederhana. Untuk memastikan setiap bola tidak saling memengaruhi transformasinya, masing-masing dibungkus dengan pasangan glPushMatrix() dan glPopMatrix() sebagai teknik manajemen transformasi lokal.

3.6 Fungsi drawBarriers()

```
void drawBarriers() {
    for (int i = 0; i < MAX_BARRIERS; i++) {
        if (barriers[i].active) {            // Use flat road
height
            float barrierY = groundLevel + 0.05f;
            barrier(barriers[i].x, barrierY, barriers[i].z,
barriers[i].scale);
        }
    }
}
```

Fungsi drawBarriers() untuk menggambar seluruh *obstacle* atau penghalang yang ada di dunia permainan. Fungsi ini melakukan iterasi melalui array barriers[] sebanyak MAX_BARRIERS, dan hanya menggambar penghalang yang statusnya aktif (barriers[i].active == true). Setiap penghalang diposisikan secara vertikal sedikit di atas permukaan tanah, yang diatur dengan menambahkan nilai 0.05f ke variabel groundLevel agar tidak tertanam ke dalam tanah, menjaga visualisasi tetap realistis.

3.7 Fungsi rumah()

```
void rumah() {
    { // Blok untuk lantai bawah
        glPushMatrix();
            glColor3ub(90, 81, 102);
            glTranslatef(0.0f, 0.35f, 0.0f);
            float halfWidth  = 5.5;
```

```
            float halfHeight = 0.35f;
            float halfDepth  = 5.5f;

            glBegin(GL_QUADS);
                // Face atas
                glVertex3f(-halfWidth,  halfHeight, -halfDepth);
                glVertex3f( halfWidth,  halfHeight, -halfDepth);
                glVertex3f( halfWidth,  halfHeight,  halfDepth);
                glVertex3f(-halfWidth,  halfHeight,  halfDepth);

                // Face bawah
                glVertex3f(-halfWidth, -halfHeight, -halfDepth);
                glVertex3f( halfWidth, -halfHeight, -halfDepth);
                glVertex3f( halfWidth, -halfHeight,  halfDepth);
                glVertex3f(-halfWidth, -halfHeight,  halfDepth);

                // Face depan
                glVertex3f(-halfWidth, -halfHeight,  halfDepth);
                glVertex3f( halfWidth, -halfHeight,  halfDepth);
                glVertex3f( halfWidth,  halfHeight,  halfDepth);
                glVertex3f(-halfWidth,  halfHeight,  halfDepth);

                // Face belakang
                glVertex3f(-halfWidth, -halfHeight, -halfDepth);
                glVertex3f( halfWidth, -halfHeight, -halfDepth);
                glVertex3f( halfWidth,  halfHeight, -halfDepth);
                glVertex3f(-halfWidth,  halfHeight, -halfDepth);

                // Face kiri
                glVertex3f(-halfWidth, -halfHeight, -halfDepth);
                glVertex3f(-halfWidth, -halfHeight,  halfDepth);
                glVertex3f(-halfWidth,  halfHeight,  halfDepth);
                glVertex3f(-halfWidth,  halfHeight, -halfDepth);

                // Face kanan
                glVertex3f( halfWidth, -halfHeight, -halfDepth);
                glVertex3f( halfWidth, -halfHeight,  halfDepth);
                glVertex3f( halfWidth,  halfHeight,  halfDepth);
                glVertex3f( halfWidth,  halfHeight, -halfDepth);
            glEnd();
        glPopMatrix();
    }

    { // Blok untuk badan rumah
        glPushMatrix();
            glColor3ub(118, 148, 61);

            glTranslatef(0.0f, 4.6f, 0.0f);
            float halfWallWidth  = 4.5f;
            float halfWallHeight = 4.0f;
            float halfWallDepth  = 4.5f;

            glBegin(GL_QUADS);
                // Face atas
                glVertex3f(-halfWallWidth,  halfWallHeight,
-halfWallDepth);
                glVertex3f( halfWallWidth,  halfWallHeight,
-halfWallDepth);
```

```
                glVertex3f( halfWallWidth,  halfWallHeight,
halfWallDepth);
                glVertex3f(-halfWallWidth,  halfWallHeight,
halfWallDepth);

                // Face bawah
                glVertex3f(-halfWallWidth, -halfWallHeight,
-halfWallDepth);
                glVertex3f( halfWallWidth, -halfWallHeight,
-halfWallDepth);
                glVertex3f( halfWallWidth, -halfWallHeight,
halfWallDepth);
                glVertex3f(-halfWallWidth, -halfWallHeight,
halfWallDepth);

                // Face depan
                glVertex3f(-halfWallWidth, -halfWallHeight,
halfWallDepth);
                glVertex3f( halfWallWidth, -halfWallHeight,
halfWallDepth);
                glVertex3f( halfWallWidth,  halfWallHeight,
halfWallDepth);
                glVertex3f(-halfWallWidth,  halfWallHeight,
halfWallDepth);

                // Face belakang
                glVertex3f(-halfWallWidth, -halfWallHeight,
-halfWallDepth);
                glVertex3f( halfWallWidth, -halfWallHeight,
-halfWallDepth);
                glVertex3f( halfWallWidth,  halfWallHeight,
-halfWallDepth);
                glVertex3f(-halfWallWidth,  halfWallHeight,
-halfWallDepth);

                // Face kiri
                glVertex3f(-halfWallWidth, -halfWallHeight,
-halfWallDepth);
                glVertex3f(-halfWallWidth, -halfWallHeight,
halfWallDepth);
                glVertex3f(-halfWallWidth,  halfWallHeight,
halfWallDepth);
                glVertex3f(-halfWallWidth,  halfWallHeight,
-halfWallDepth);

                // Face kanan
                glVertex3f( halfWallWidth, -halfWallHeight,
-halfWallDepth);
                glVertex3f( halfWallWidth, -halfWallHeight,
halfWallDepth);
                glVertex3f( halfWallWidth,  halfWallHeight,
halfWallDepth);
                glVertex3f( halfWallWidth,  halfWallHeight,
-halfWallDepth);
            glEnd();
        glPopMatrix();
    }
```

```
    { // Blok untuk lantai tengah
        glPushMatrix();
            glColor3ub(90, 81, 102);
            glTranslatef(0.0f, 9.0f, 0.0f);
            float halfWidth  = 5.0f;
            float halfHeight = 0.3f;
            float halfDepth  = 5.0f;

            glBegin(GL_QUADS);
                // Face atas
                glVertex3f(-halfWidth,  halfHeight, -halfDepth);
                glVertex3f( halfWidth,  halfHeight, -halfDepth);
                glVertex3f( halfWidth,  halfHeight,  halfDepth);
                glVertex3f(-halfWidth,  halfHeight,  halfDepth);

                // Face bawah
                glVertex3f(-halfWidth, -halfHeight, -halfDepth);
                glVertex3f( halfWidth, -halfHeight, -halfDepth);
                glVertex3f( halfWidth, -halfHeight,  halfDepth);
                glVertex3f(-halfWidth, -halfHeight,  halfDepth);

                // Face depan
                glVertex3f(-halfWidth, -halfHeight,  halfDepth);
                glVertex3f( halfWidth, -halfHeight,  halfDepth);
                glVertex3f( halfWidth,  halfHeight,  halfDepth);
                glVertex3f(-halfWidth,  halfHeight,  halfDepth);

                // Face belakang
                glVertex3f(-halfWidth, -halfHeight, -halfDepth);
                glVertex3f( halfWidth, -halfHeight, -halfDepth);
                glVertex3f( halfWidth,  halfHeight, -halfDepth);
                glVertex3f(-halfWidth,  halfHeight, -halfDepth);

                // Face kiri
                glVertex3f(-halfWidth, -halfHeight, -halfDepth);
                glVertex3f(-halfWidth, -halfHeight,  halfDepth);
                glVertex3f(-halfWidth,  halfHeight,  halfDepth);
                glVertex3f(-halfWidth,  halfHeight, -halfDepth);

                // Face kanan
                glVertex3f( halfWidth, -halfHeight, -halfDepth);
                glVertex3f( halfWidth, -halfHeight,  halfDepth);
                glVertex3f( halfWidth,  halfHeight,  halfDepth);
                glVertex3f( halfWidth,  halfHeight, -halfDepth);
            glEnd();
        glPopMatrix();
    }

    { // Blok untuk badan rumah 2
        glPushMatrix();
            glColor3ub(118, 148, 61);
            glTranslatef(0.0f, 12.4f, 0.0f);
            float halfWallWidth  = 4.5f;
            float halfWallHeight = 4.0f;
            float halfWallDepth  = 4.5f;

            glBegin(GL_QUADS);
                // Face atas
```

```
                glVertex3f(-halfWallWidth,  halfWallHeight,
-halfWallDepth);
                glVertex3f( halfWallWidth,  halfWallHeight,
-halfWallDepth);
                glVertex3f( halfWallWidth,  halfWallHeight,
halfWallDepth);
                glVertex3f(-halfWallWidth,  halfWallHeight,
halfWallDepth);

                // Face bawah
                glVertex3f(-halfWallWidth, -halfWallHeight,
-halfWallDepth);
                glVertex3f( halfWallWidth, -halfWallHeight,
-halfWallDepth);
                glVertex3f( halfWallWidth, -halfWallHeight,
halfWallDepth);
                glVertex3f(-halfWallWidth, -halfWallHeight,
halfWallDepth);

                // Face depan
                glVertex3f(-halfWallWidth, -halfWallHeight,
halfWallDepth);
                glVertex3f( halfWallWidth, -halfWallHeight,
halfWallDepth);
                glVertex3f( halfWallWidth,  halfWallHeight,
halfWallDepth);
                glVertex3f(-halfWallWidth,  halfWallHeight,
halfWallDepth);

                // Face belakang
                glVertex3f(-halfWallWidth, -halfWallHeight,
-halfWallDepth);
                glVertex3f( halfWallWidth, -halfWallHeight,
-halfWallDepth);
                glVertex3f( halfWallWidth,  halfWallHeight,
-halfWallDepth);
                glVertex3f(-halfWallWidth,  halfWallHeight,
-halfWallDepth);

                // Face kiri
                glVertex3f(-halfWallWidth, -halfWallHeight,
-halfWallDepth);
                glVertex3f(-halfWallWidth, -halfWallHeight,
halfWallDepth);
                glVertex3f(-halfWallWidth,  halfWallHeight,
halfWallDepth);
                glVertex3f(-halfWallWidth,  halfWallHeight,
-halfWallDepth);

                // Face kanan
                glVertex3f( halfWallWidth, -halfWallHeight,
-halfWallDepth);
                glVertex3f( halfWallWidth, -halfWallHeight,
halfWallDepth);
                glVertex3f( halfWallWidth,  halfWallHeight,
halfWallDepth);
                glVertex3f( halfWallWidth,  halfWallHeight,
-halfWallDepth);
```

```
            glEnd();
        glPopMatrix();
    }

    { // atap 1
        glPushMatrix();
            glColor3ub(90, 81, 102);
            glTranslatef(0.0f, 16.7f, 0.0f);
            float halfWidth  = 5.0f;
            float halfHeight = 0.3f;
            float halfDepth  = 5.0f;

            glBegin(GL_QUADS);
                // Face atas
                glVertex3f(-halfWidth,  halfHeight, -halfDepth);
                glVertex3f( halfWidth,  halfHeight, -halfDepth);
                glVertex3f( halfWidth,  halfHeight,  halfDepth);
                glVertex3f(-halfWidth,  halfHeight,  halfDepth);

                // Face bawah
                glVertex3f(-halfWidth, -halfHeight, -halfDepth);
                glVertex3f( halfWidth, -halfHeight, -halfDepth);
                glVertex3f( halfWidth, -halfHeight,  halfDepth);
                glVertex3f(-halfWidth, -halfHeight,  halfDepth);

                // Face depan
                glVertex3f(-halfWidth, -halfHeight,  halfDepth);
                glVertex3f( halfWidth, -halfHeight,  halfDepth);
                glVertex3f( halfWidth,  halfHeight,  halfDepth);
                glVertex3f(-halfWidth,  halfHeight,  halfDepth);

                // Face belakang
                glVertex3f(-halfWidth, -halfHeight, -halfDepth);
                glVertex3f( halfWidth, -halfHeight, -halfDepth);
                glVertex3f( halfWidth,  halfHeight, -halfDepth);
                glVertex3f(-halfWidth,  halfHeight, -halfDepth);

                // Face kiri
                glVertex3f(-halfWidth, -halfHeight, -halfDepth);
                glVertex3f(-halfWidth, -halfHeight,  halfDepth);
                glVertex3f(-halfWidth,  halfHeight,  halfDepth);
                glVertex3f(-halfWidth,  halfHeight, -halfDepth);

                // Face kanan
                glVertex3f( halfWidth, -halfHeight, -halfDepth);
                glVertex3f( halfWidth, -halfHeight,  halfDepth);
                glVertex3f( halfWidth,  halfHeight,  halfDepth);
                glVertex3f( halfWidth,  halfHeight, -halfDepth);
            glEnd();
        glPopMatrix();
    }

    { // atap 2
    glPushMatrix();
        glColor3ub(90, 81, 102);
        glTranslatef(0.0f, 17.3f, 0.0f);
        float halfWidth  = 5.2f;
        float halfHeight = 0.3f;
```

```
        float halfDepth  = 5.2f;

        glBegin(GL_QUADS);
            // Face atas
            glVertex3f(-halfWidth,  halfHeight, -halfDepth);
            glVertex3f( halfWidth,  halfHeight, -halfDepth);
            glVertex3f( halfWidth,  halfHeight,  halfDepth);
            glVertex3f(-halfWidth,  halfHeight,  halfDepth);

            // Face bawah
            glVertex3f(-halfWidth, -halfHeight, -halfDepth);
            glVertex3f( halfWidth, -halfHeight, -halfDepth);
            glVertex3f( halfWidth, -halfHeight,  halfDepth);
            glVertex3f(-halfWidth, -halfHeight,  halfDepth);

            // Face depan
            glVertex3f(-halfWidth, -halfHeight,  halfDepth);
            glVertex3f( halfWidth, -halfHeight,  halfDepth);
            glVertex3f( halfWidth,  halfHeight,  halfDepth);
            glVertex3f(-halfWidth,  halfHeight,  halfDepth);

            // Face belakang
            glVertex3f(-halfWidth, -halfHeight, -halfDepth);
            glVertex3f( halfWidth, -halfHeight, -halfDepth);
            glVertex3f( halfWidth,  halfHeight, -halfDepth);
            glVertex3f(-halfWidth,  halfHeight, -halfDepth);

            // Face kiri
            glVertex3f(-halfWidth, -halfHeight, -halfDepth);
            glVertex3f(-halfWidth, -halfHeight,  halfDepth);
            glVertex3f(-halfWidth,  halfHeight,  halfDepth);
            glVertex3f(-halfWidth,  halfHeight, -halfDepth);

            // Face kanan
            glVertex3f( halfWidth, -halfHeight, -halfDepth);
            glVertex3f( halfWidth, -halfHeight,  halfDepth);
            glVertex3f( halfWidth,  halfHeight,  halfDepth);
            glVertex3f( halfWidth,  halfHeight, -halfDepth);
        glEnd();
    glPopMatrix();
}

    { // pintu dan jendela
    glPushMatrix();

        glTranslatef(0.0f, 0.35f, 4.65f);
        glColor3ub(133, 101, 14);

        // Ukuran pintu
        float doorWidth     = 2.5f;
        float doorHeight    = 4.0f;
        float doorThickness = 0.1f;
        float archRadius    = 1.25f;
        int   segments      = 16;    // jumlah segmen lengkungan

        // Variabel pembantu
        float w2 = doorWidth * 0.5f; // setengah lebar
        float t  = doorThickness;
```

```
        float h  = doorHeight;

        // Depan
        glBegin(GL_QUADS);
            glVertex3f(-w2, 0.0f,       0.0f);
            glVertex3f( w2, 0.0f,       0.0f);
            glVertex3f( w2, h,          0.0f);
            glVertex3f(-w2, h,          0.0f);
        glEnd();

        // Belakang
        glBegin(GL_QUADS);
            glVertex3f(-w2, 0.0f,      -t);
            glVertex3f( w2, 0.0f,      -t);
            glVertex3f( w2, h,         -t);
            glVertex3f(-w2, h,         -t);
        glEnd();

        // Sisi bawah
        glBegin(GL_QUADS);
            glVertex3f(-w2, 0.0f,  0.0f);
            glVertex3f(-w2, 0.0f, -t);
            glVertex3f( w2, 0.0f, -t);
            glVertex3f( w2, 0.0f,  0.0f);
        glEnd();

        // Sisi atas
        glBegin(GL_QUADS);
            glVertex3f(-w2, h,  0.0f);
            glVertex3f(-w2, h, -t);
            glVertex3f( w2, h, -t);
            glVertex3f( w2, h,  0.0f);
        glEnd();

        // Sisi kiri
        glBegin(GL_QUADS);
            glVertex3f(-w2, 0.0f,  0.0f);
            glVertex3f(-w2, h,     0.0f);
            glVertex3f(-w2, h,    -t);
            glVertex3f(-w2, 0.0f, -t);
        glEnd();

        // Sisi kanan
        glBegin(GL_QUADS);
            glVertex3f(w2, 0.0f,  0.0f);
            glVertex3f(w2, h,     0.0f);
            glVertex3f(w2, h,    -t);
            glVertex3f(w2, 0.0f, -t);
        glEnd();


        // border
        {    float borderThickness = 0.1f;
        glColor3ub(90, 81, 102);

        // border atas
        glPushMatrix();
            glTranslatef(0.0f, h + borderThickness*0.5f,
```

```
-t*0.5f);
            glScalef(doorWidth + 2*borderThickness,
borderThickness, doorThickness + 2*borderThickness);
            glutSolidCube(1.0f);
        glPopMatrix();

        // 2) Border sisi kiri
        glPushMatrix();
            glTranslatef(-w2 - borderThickness*0.5f, h*0.5f,
-t*0.5f);
            glScalef(borderThickness, doorHeight, doorThickness +
2*borderThickness);
            glutSolidCube(1.0f);
        glPopMatrix();

        // 3) Border sisi kanan
        glPushMatrix();
            glTranslatef(w2 + borderThickness*0.5f, h*0.5f,
-t*0.5f);
            glScalef(borderThickness, doorHeight, doorThickness +
2*borderThickness);
            glutSolidCube(1.0f);
        glPopMatrix();}



        // 1. Bagian kaca
        {
            glPushMatrix();
            glTranslatef(0.0f, 0.0f, -0.08f);
            glColor3ub(107, 215, 255);

            float baseY = h + 0.5f;  // pangkal lengkungan
            // Bagian depan (z=0)
            glBegin(GL_TRIANGLE_FAN);
                glVertex3f(0.0f, baseY, 0.0f); // pusat
lengkungan
                for(int i = 0; i <= segments; i++) {
                    float theta = 3.14159f * i / segments;
                    float xPos = archRadius * cosf(theta);
                    float yPos = archRadius * sinf(theta);
                    glVertex3f(xPos, baseY + yPos, 0.0f);
                }
            glEnd();

            // Bagian belakang (z = -t)
            glBegin(GL_TRIANGLE_FAN);
                glVertex3f(0.0f, baseY, -t);
                for(int i = 0; i <= segments; i++) {
                    float theta = 3.14159f * i / segments;
                    float xPos = archRadius * cosf(theta);
                    float yPos = archRadius * sinf(theta);
                    glVertex3f(xPos, baseY + yPos, -t);
                }
            glEnd();

            // Sisi lengkungan
            glBegin(GL_QUAD_STRIP);
```

```
            for(int i = 0; i <= segments; i++) {
                float theta = 3.14159f * i / segments;
                float xPos = archRadius * cosf(theta);
                float yPos = archRadius * sinf(theta);
                glVertex3f(xPos, baseY + yPos, 0.0f);
                glVertex3f(xPos, baseY + yPos, -t);
            }
        glEnd();

        glPopMatrix();
    }

    // 2. Border
    {
        glColor3ub(255, 255, 255);

        float borderThickness = 0.1f;
        float outerArchRadius = archRadius + borderThickness;
        float archBaseY = h + 0.5f;

        // Bagian depan border
        glBegin(GL_QUAD_STRIP);
        for(int i = 0; i <= segments; i++) {
            float theta = 3.14159f * i / segments;
            float xOuter = outerArchRadius * cosf(theta);
            float yOuter = outerArchRadius * sinf(theta);
            float xInner = archRadius * cosf(theta);
            float yInner = archRadius * sinf(theta);

            glVertex3f(xOuter, archBaseY + yOuter, 0.0f);
            glVertex3f(xInner, archBaseY + yInner, 0.0f);
        }
        glEnd();

        // Bagian belakang border
        glBegin(GL_QUAD_STRIP);
        for(int i = 0; i <= segments; i++) {
            float theta = 3.14159f * i / segments;
            float xOuter = outerArchRadius * cosf(theta);
            float yOuter = outerArchRadius * sinf(theta);
            float xInner = archRadius * cosf(theta);
            float yInner = archRadius * sinf(theta);

            glVertex3f(xOuter, archBaseY + yOuter, -t);
            glVertex3f(xInner, archBaseY + yInner, -t);
        }
        glEnd();

        // Sisi lengkungan border
        glBegin(GL_QUAD_STRIP);
        for(int i = 0; i <= segments; i++) {
            float theta = 3.14159f * i / segments;
            float xPos = outerArchRadius * cosf(theta);
            float yPos = outerArchRadius * sinf(theta);

            glVertex3f(xPos, archBaseY + yPos, 0.0f);
            glVertex3f(xPos, archBaseY + yPos, -t);
        }
```

```
            glEnd();

            // border atas
            glPushMatrix();
                glTranslatef(0.0f, 0.5 + h +
borderThickness*0.5f, -t*0.5f);
                glScalef(doorWidth + 2*borderThickness,
borderThickness, doorThickness + 2*borderThickness);
                glutSolidCube(1.0f);
             glPopMatrix();
        }

        // 3. Tralis
        {
        {
            float tralisThickness = 0.05f;
            float R = archRadius;  // jari-jari lengkungan

            float centerX = 0.0f;
            float centerY = h + 0.5f;
            float centerZ = 0.0f;

            for (int i = 0; i < 3; i++) {
                float theta;
                if (i == 0)
                    theta = 3.14159f * 0.5f;   // 90° (vertikal)
                else if (i == 1)
                    theta = 3.14159f * 0.25f;  // 45°
                else
                    theta = 3.14159f * 0.75f;  // 135°

                // Titik start adalah pusat
                float sx = centerX;
                float sy = centerY;
                float sz = centerZ;
                // Titik end dihitung dari pusat + offset
berbasis theta
                float ex = centerX + R * cosf(theta);
                float ey = centerY + R * sinf(theta);
                float ez = centerZ;

                // Vektor dari start ke end:
                float dx = ex - sx;
                float dy = ey - sy;
                // Panjang vektor (seharusnya sama dengan R)
                float len = sqrt(dx * dx + dy * dy);
                // Vektor normal (perpendicular di bidang XY)
                float nx = -dy / len;
                float ny = dx / len;
                // Offset sebesar setengah ketebalan
                float ox = nx * (tralisThickness * 0.5f);
                float oy = ny * (tralisThickness * 0.5f);

                // Tentukan empat titik quad:
                // v1: start + offset
                float v1x = sx + ox, v1y = sy + oy, v1z = sz;
                // v2: start - offset
                float v2x = sx - ox, v2y = sy - oy, v2z = sz;
```

```
                // v3: end - offset
                float v3x = ex - ox, v3y = ey - oy, v3z = ez;
                // v4: end + offset
                float v4x = ex + ox, v4y = ey + oy, v4z = ez;

                glColor3ub(255, 255, 255);
                glBegin(GL_QUADS);
                    glVertex3f(v1x, v1y, v1z);
                    glVertex3f(v2x, v2y, v2z);
                    glVertex3f(v3x, v3y, v3z);
                    glVertex3f(v4x, v4y, v4z);
                glEnd();
            }
        }
     }

        // gagang pintu
        glPushMatrix();
             glTranslatef(-w2 + 0.2f, h * 0.5f, 0.0f);
             glColor3ub(80, 80, 80);
             glutSolidSphere(0.08, 16, 16);
        glPopMatrix();

    glPopMatrix();
    }

    { // Jendela 1
    glPushMatrix();
        glTranslatef(2.0f, 12.85f, 4.65f);
        float windowWidth  = 2.0f;
        float windowHeight = 3.0f;
        float windowDepth  = 0.1f;
        float frameThickness = 0.2f;
        float crossThickness = 0.1f;

        // Kaca jendela
        glColor3ub(107, 215, 255);
        glBegin(GL_QUADS);
            glVertex3f(-windowWidth * 0.5f, -windowHeight * 0.5f,
0.0f);
            glVertex3f( windowWidth * 0.5f, -windowHeight * 0.5f,
0.0f);
            glVertex3f( windowWidth * 0.5f,  windowHeight * 0.5f,
0.0f);
            glVertex3f(-windowWidth * 0.5f,  windowHeight * 0.5f,
0.0f);
        glEnd();

        // Frame
        glColor3ub(255, 255, 255);
        // Atas
        glPushMatrix();
            glTranslatef(0.0f, windowHeight * 0.5f +
frameThickness * 0.5f, 0.0f);
            glScalef(windowWidth + frameThickness * 2,
frameThickness, windowDepth);
            glutSolidCube(1.0f);
        glPopMatrix();
```

```
        // Bawah
        glPushMatrix();
            glTranslatef(0.0f, -windowHeight * 0.5f -
frameThickness * 0.5f, 0.0f);
            glScalef(windowWidth + frameThickness * 2,
frameThickness, windowDepth);
            glutSolidCube(1.0f);
        glPopMatrix();


        // Kiri
        glPushMatrix();
            glTranslatef(-windowWidth * 0.5f - frameThickness *
0.5f, 0.0f, 0.0f);
            glScalef(frameThickness, windowHeight +
frameThickness * 2, windowDepth);
            glutSolidCube(1.0f);
        glPopMatrix();

        // Kanan
        glPushMatrix();
            glTranslatef(windowWidth * 0.5f + frameThickness *
0.5f, 0.0f, 0.0f);
            glScalef(frameThickness, windowHeight +
frameThickness * 2, windowDepth);
            glutSolidCube(1.0f);
        glPopMatrix();


        {glPushMatrix(); //alas
            glColor3ub(90, 81, 102);

            float sillWidth  = 2.7f;
            float sillHeight = 0.4f;
            float sillDepth  = 0.5f;

            glTranslatef(0.0f, (-windowHeight * 0.5f -
frameThickness * 0.5f) - 0.2f, 0.0f);

            glScalef(sillWidth, sillHeight, sillDepth);
            glutSolidCube(1.0f);
        glPopMatrix();}

        // Frame tengah
        glColor3ub(255, 255, 255);

        // Vertikal
        glPushMatrix();
            glScalef(crossThickness, windowHeight, windowDepth);
            glutSolidCube(1.0f);
        glPopMatrix();

        // Horizontal
        glPushMatrix();
            glScalef(windowWidth, crossThickness, windowDepth);
            glutSolidCube(1.0f);
        glPopMatrix();
```

```
    glPopMatrix();
}

    { // Jendela 2
            glPushMatrix();
            glTranslatef(-2.0f, 12.85f, 4.65f);
            float windowWidth  = 2.0f;
            float windowHeight = 3.0f;
            float windowDepth  = 0.1f;
            float frameThickness = 0.2f;
            float crossThickness = 0.1f;

            // Kaca jendela
            glColor3ub(107, 215, 255);
            glBegin(GL_QUADS);
                glVertex3f(-windowWidth * 0.5f, -windowHeight *
0.5f, 0.0f);
                glVertex3f( windowWidth * 0.5f, -windowHeight *
0.5f, 0.0f);
                glVertex3f( windowWidth * 0.5f,  windowHeight *
0.5f, 0.0f);
                glVertex3f(-windowWidth * 0.5f,  windowHeight *
0.5f, 0.0f);
            glEnd();

            // Frame
            glColor3ub(255, 255, 255);
            // Atas
            glPushMatrix();
                glTranslatef(0.0f, windowHeight * 0.5f +
frameThickness * 0.5f, 0.0f);
                glScalef(windowWidth + frameThickness * 2,
frameThickness, windowDepth);
                glutSolidCube(1.0f);
            glPopMatrix();

            // Bawah
            glPushMatrix();
                glTranslatef(0.0f, -windowHeight * 0.5f -
frameThickness * 0.5f, 0.0f);
                glScalef(windowWidth + frameThickness * 2,
frameThickness, windowDepth);
                glutSolidCube(1.0f);
            glPopMatrix();

            // Kiri
            glPushMatrix();
                glTranslatef(-windowWidth * 0.5f - frameThickness
* 0.5f, 0.0f, 0.0f);
                glScalef(frameThickness, windowHeight +
frameThickness * 2, windowDepth);
                glutSolidCube(1.0f);
            glPopMatrix();

            // Kanan
            glPushMatrix();
                glTranslatef(windowWidth * 0.5f + frameThickness
```

```
* 0.5f, 0.0f, 0.0f);
                glScalef(frameThickness, windowHeight +
frameThickness * 2, windowDepth);
                glutSolidCube(1.0f);
            glPopMatrix();

            {glPushMatrix(); // alas
                glColor3ub(90, 81, 102);

                float sillWidth  = 2.7f;
                float sillHeight = 0.4f;
                float sillDepth  = 0.5f;

                glTranslatef(0.0f, (-windowHeight * 0.5f -
frameThickness * 0.5f) - 0.2f, 0.0f);

                glScalef(sillWidth, sillHeight, sillDepth);
                glutSolidCube(1.0f);
            glPopMatrix();}


            // Frame tengah
            glColor3ub(255, 255, 255);

            // Vertikal
            glPushMatrix();
                glScalef(crossThickness, windowHeight,
windowDepth);
                glutSolidCube(1.0f);
            glPopMatrix();

            // Horizontal
            glPushMatrix();
                glScalef(windowWidth, crossThickness,
windowDepth);
                glutSolidCube(1.0f);
            glPopMatrix();
        glPopMatrix();
    }

    { // brick

        glPushMatrix();
            glTranslatef(0.0f, 4.6f, 0.0f);
            glColor3ub(106, 133, 55);

            // 1
            glPushMatrix();
                glTranslatef(3.2f, 2.0f, 4.51f);
                drawBrick(1.25f, 0.75f, 0.2f);
            glPopMatrix();

            // 2
            glPushMatrix();
                glTranslatef(-2.5f, 1.75f, 4.51f);
                drawBrick(1.25f, 0.75f, 0.2f);
            glPopMatrix();
```

```
            // 3
            glPushMatrix();
                glTranslatef(-3.4f, 2.75f, 4.51f);
                drawBrick(1.25f, 0.75f, 0.2f);
            glPopMatrix();

            // 4
            glPushMatrix();
                glTranslatef(2.4f, -1.3f, 4.51f);
                drawBrick(1.25f, 0.75f, 0.2f);
            glPopMatrix();


            // 5
            glPushMatrix();
                glTranslatef(0.8f, 5.5f, 4.51f);
                drawBrick(1.25f, 0.75f, 0.2f);
            glPopMatrix();

            // 6
            glPushMatrix();
                glTranslatef(-2.9f, 10.8f, 4.51f);
                drawBrick(1.25f, 0.75f, 0.2f);
            glPopMatrix();
        glPopMatrix();
    }
}
```

Fungsi rumah() bertujuan untuk membangun model rumah 3D secara terstruktur menggunakan teknik pemodelan berbasis primitive OpenGL. Proses konstruksi dimulai dari lantai bawah yang dibentuk sebagai balok lebar dan tipis berwarna abu-abu kebiruan, kemudian dilanjutkan dengan badan rumah bawah yang berupa balok hijau kekuningan untuk menampilkan dinding lantai dasar. Di atasnya, lantai tengah ditambahkan dengan dimensi sedikit lebih kecil namun tetap mempertahankan warna dan bentuk yang seragam dengan lantai bawah. Selanjutnya, badan rumah atas dibangun dengan bentuk dan warna yang sama seperti lantai bawah, namun ditempatkan lebih tinggi untuk mensimulasikan rumah bertingkat. Bagian atap terdiri dari dua lapisan balok tipis yang semakin melebar ke atas, menghasilkan efek overhang yang alami pada struktur atap rumah. Pada bagian depan, pintu utama dibuat dengan detail lengkungan kaca biru, border putih, serta tralis dekoratif dan gagang pintu berbentuk bola, sehingga memberikan karakteristik pintu klasik. Di sisi kanan dan kiri lantai atas, dua buah jendela diletakkan simetris dengan desain bingkai putih, kaca biru, dan adanya frame tengah baik vertikal maupun horizontal untuk mempertegas tampilan jendela tradisional.

Ornamen tambahan berupa batu bata berwarna hijau kekuningan diposisikan secara acak di dinding depan bawah untuk memberi tekstur visual yang lebih hidup. Seluruh bagian rumah dibangun secara modular dengan pengelolaan transformasi melalui pasangan glPushMatrix() dan glPopMatrix(), memastikan setiap elemen dapat diposisikan dan diubah secara independen tanpa mengganggu bagian lain. Teknik pemodelan ini menunjukkan pemahaman prinsip komposisi objek 3D, pemanfaatan warna, serta penerapan transformasi geometri secara hierarkis untuk menghasilkan model rumah yang realistis dan ekspresif dalam lingkungan grafika komputer.

3.8 Fungsi drawTrees()

```
void drawTrees() {
    // Parameter pohon dengan jarak yang diperbesar
    float treeOffset = roadWidth/2 + 3.5f;  // Jarak dari tepi
jalan, lebih jauh dari pagar
    float treeSpacing = 40.0f;              // Jarak antar pohon
diperbesar
    float houseOffset = 8.0f;               // UBAH: Lebih dekat
ke jalan (dari 14.0f)

    // Gunakan ketinggian konstan untuk mencegah pohon bergerak
naik turun
    float treeGroundY = -0.2f;
    float houseGroundY = -0.2f;

    // Iterasi melalui setiap segmen jalan aktif, sama seperti
pagar
    for (int segIdx = 0; segIdx < NUM_ROAD_SEGMENTS; segIdx++) {
        if (!roadSegments[segIdx].active) continue;

        // Gunakan segmen jalan yang sama seperti digunakan untuk
pagar
        float segmentZ1 = roadSegments[segIdx].startZ;
        float segmentZ2 = roadSegments[segIdx].endZ;
        float segmentLength = segmentZ1 - segmentZ2;

        // Hitung berapa pohon bisa ditampilkan di segmen ini
        int treesPerSegment = (int)(segmentLength / treeSpacing)
+ 1;

        // Iterasi untuk kedua sisi jalan
        for (int side = -1; side <= 1; side += 2) {
            if (side == 0) continue; // Lewati bagian tengah

            // Iterasi pohon dalam segmen ini
            for (int i = 0; i < treesPerSegment; i++) {
                // Posisi Z absolut untuk setiap pohon dalam
segmen
                float treeZ = segmentZ1 - i * treeSpacing -
```

```
treeSpacing/2;

                // Cek apakah masih dalam segmen ini
                if (treeZ < segmentZ2) continue;

                // Generate ID unik untuk pohon ini berdasarkan
posisi absolutnya
                int treeId = (int)(treeZ * 100.0f) + (side < 0 ?
100000 : 200000);

                // Gunakan treeId untuk konsistensi acak
                float hashValue1 = sin(treeId * 0.3567f) * 0.5f +
0.5f;
                float hashValue2 = sin(treeId * 0.4321f) * 0.5f +
0.5f;
                float hashValue3 = sin(treeId * 0.7123f) * 0.5f +
0.5f;

                // Variasi posisi X dengan jarak dari jalan yang
konsisten
                float treeXOffset = hashValue1 * 1.2f;
                float treeX = side * (treeOffset + treeXOffset);

                // Variasi ukuran pohon
                float trunkHeight = 2.5f + hashValue2 * 1.0f;
                float trunkWidth = 0.25f + hashValue3 * 0.15f;
                float foliageSize = 1.0f + hashValue2 * 0.4f;

                // Variasi warna batang
                float trunkR = 0.35f + hashValue1 * 0.1f;
                float trunkG = 0.2f + hashValue3 * 0.05f;
                float trunkB = 0.1f + hashValue2 * 0.05f;

                // Variasi warna daun
                float foliageR = 0.1f + hashValue3 * 0.05f;
                float foliageG = 0.45f + hashValue1 * 0.15f;
                float foliageB = 0.1f + hashValue2 * 0.05f;

                // Render batang pohon
                glEnable(GL_LIGHTING);
                glColor3f(trunkR, trunkG, trunkB);
                glPushMatrix();
                    glTranslatef(treeX, treeGroundY +
trunkHeight/2, treeZ);
                    glScalef(trunkWidth, trunkHeight,
trunkWidth);
                    glutSolidCube(1.0f);
                glPopMatrix();

                // Render daun (3 lapisan)
                glColor3f(foliageR, foliageG, foliageB);
                glPushMatrix();
                    glTranslatef(treeX, treeGroundY +
trunkHeight*0.7f, treeZ);
                    glScalef(foliageSize * 1.3f, foliageSize *
0.9f, foliageSize * 1.3f);
                    glutSolidSphere(1.0f, 8, 6);
                glPopMatrix();
```

```
                // Lapisan daun tengah
                glColor3f(foliageR + 0.05f, foliageG + 0.05f,
foliageB);
                glPushMatrix();
                    glTranslatef(treeX, treeGroundY +
trunkHeight*0.85f, treeZ);
                    glScalef(foliageSize * 1.1f, foliageSize *
0.8f, foliageSize * 1.1f);
                    glutSolidSphere(1.0f, 8, 6);
                glPopMatrix();

                // Lapisan daun atas
                glColor3f(foliageR + 0.08f, foliageG + 0.08f,
foliageB);
                glPushMatrix();
                    glTranslatef(treeX, treeGroundY +
trunkHeight, treeZ);
                    glScalef(foliageSize * 0.9f, foliageSize *
0.7f, foliageSize * 0.9f);
                    glutSolidSphere(1.0f, 8, 6);
                glPopMatrix();

                // Saat menambahkan rumah, ubah skalanya dan
kondisi penempatannya:
                    if (i % 2 == 0) {  // Setiap pohon kedua
                        if (side > 0) {  // Sisi kanan jalan
                            // Rumah dengan jarak yang lebih
dekat
                            float houseZ = treeZ - 12.0f;  //
UBAH: Letakkan secara eksplisit, tidak di tengah pohon
                            float houseX = side * houseOffset;

                            // TAMBAHKAN: Debug output
                            std::cout << "Placing right house
at X: " << houseX << " Z: " << houseZ << std::endl;

                            glPushMatrix();
                                // Sesuaikan skala dan rotasi
rumah
                                glTranslatef(houseX,
houseGroundY, houseZ);
                                glScalef(0.4f, 0.4f, 0.4f);  //
UBAH: Perbesar skala (dari 0.2f)
                                glRotatef(-90, 0, 1, 0);
                                // Pastikan lighting diatur
dengan benar untuk rumah
                                GLfloat house_ambient[] =
{0.6f, 0.6f, 0.6f, 1.0f};
                                glMaterialfv(GL_FRONT,
GL_AMBIENT, house_ambient);
                                rumah();
                            glPopMatrix();
                        }
                        else if (i % 4 == 0) {  // Sisi kiri,
lebih jarang
                            // Rumah sisi kiri
                            float houseZ = treeZ - 8.0f;  //
```

```
UBAH: Posisi eksplisit
                                float houseX = side * (houseOffset
+ 1.0f);  // UBAH: Lebih dekat (dari 2.0f offset tambahan)

                                glPushMatrix();
                                    glTranslatef(houseX,
houseGroundY, houseZ);
                                    glScalef(0.45f, 0.45f, 0.45f);
// UBAH: Perbesar skala (dari 0.25f)
                                    glRotatef(90, 0, 1, 0);
                                    GLfloat house_ambient[] =
{0.6f, 0.6f, 0.6f, 1.0f};
                                    glMaterialfv(GL_FRONT,
GL_AMBIENT, house_ambient);
                                    rumah();
                                glPopMatrix();
                            }
                        }
                }
            }
        }
}
```

drawTrees() bertujuan untuk menggambar pohon dan rumah secara procedural di sepanjang jalan dalam sebuah lingkungan 3D menggunakan OpenGL. Pohon ditempatkan di kedua sisi jalan pada segmen jalan yang aktif, dengan jarak antar pohon yang tetap dan posisi yang sedikit bervariasi secara konsisten menggunakan fungsi sinus dari ID unik setiap pohon. Variasi ini mencakup ukuran batang, warna batang dan daun, serta ukuran kanopi daun yang terdiri dari tiga lapisan bola. Setiap pohon digambar menggunakan kombinasi glutSolidCube dan glutSolidSphere, dengan lighting diaktifkan. Selain pohon, rumah juga ditambahkan secara selektif rumah sisi kanan muncul di setiap pohon genap dan diletakkan lebih dekat ke jalan, sedangkan rumah sisi kiri lebih jarang muncul dan sedikit lebih jauh. Seluruh proses ini menciptakan tampilan jalan yang dinamis dan alami tanpa perlu menempatkan objek secara manual.

3.9 Fungsi glShadowProjection()

```
void glShadowProjection(float* l, float* e, float* n) {
    float d, c;
    float mat[16];

    d = n[0]*l[0] + n[1]*l[1] + n[2]*l[2];
    c = e[0]*n[0] + e[1]*n[1] + e[2]*n[2] - d;

    mat[0]  = l[0]*n[0]+c;
    mat[4]  = n[1]*l[0];
```

```
    mat[8]  = n[2]*l[0];
    mat[12] = -l[0]*c-l[0]*d;

    mat[1]  = n[0]*l[1];
    mat[5]  = l[1]*n[1]+c;
    mat[9]  = n[2]*l[1];
    mat[13] = -l[1]*c-l[1]*d;

    mat[2]  = n[0]*l[2];
    mat[6]  = n[1]*l[2];
    mat[10] = l[2]*n[2]+c;
    mat[14] = -l[2]*c-l[2]*d;

    mat[3]  = n[0];
    mat[7]  = n[1];
    mat[11] = n[2];
    mat[15] = -d;

    glMultMatrixf(mat);
}

void render() {
    glClearColor(0.0, 0.6, 0.9, 0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glLightfv(GL_LIGHT0, GL_POSITION, l);

    // Draw ground plane
    glDisable(GL_LIGHTING);
    glColor3f(0.8f, 0.8f, 0.8f);
    glBegin(GL_QUADS);
        glNormal3f(0.0, 1.0, 0.0);
        glVertex3f(-1300.0, e[1]-0.1, 1300.0);
        glVertex3f(1300.0, e[1]-0.1, 1300.0);
        glVertex3f(1300.0, e[1]-0.1, -1300.0);
        glVertex3f(-1300.0, e[1]-0.1, -1300.0);
    glEnd();

    // Draw actual objects
    glEnable(GL_LIGHTING);
    glPushMatrix();
        glRotatef(ry, 0, 1, 0);
        glRotatef(rx, 1, 0, 0);
        drawTrees();
        drawBarriers();
        drawGerobak();
    glPopMatrix();

    // Draw shadows
    glPushMatrix();
        glShadowProjection(l, e, n);
        glRotatef(ry, 0, 1, 0);
        glRotatef(rx, 1, 0, 0);
        glDisable(GL_LIGHTING);
        glColor3f(0.4, 0.4, 0.4);
        drawTrees();
        drawBarriers();
        drawGerobak();
```

```
    glPopMatrix();

    glutSwapBuffers();
}

void idle() {
    rx += 0.1f;
    ry += 0.1f;
    glutPostRedisplay();
}
```

glShadowProjection() bagian dari program grafis 3D yang menampilkan objek-objek seperti pohon, pembatas jalan, dan gerobak, lengkap dengan bayangan yang diproyeksikan ke permukaan tanah. Fungsi glShadowProjection() menghitung matriks proyeksi bayangan berdasarkan posisi cahaya (l), titik pada bidang (e), dan normal bidang (n), kemudian menerapkannya menggunakan glMultMatrixf() untuk mengubah transformasi OpenGL agar objek digambar sebagai bayangan. Fungsi render() menggambar keseluruhan adegan: pertama, membersihkan layar dan menggambar bidang tanah tanpa pencahayaan; lalu menggambar objek asli dengan pencahayaan diaktifkan dan rotasi (rx, ry); kemudian menggambar kembali objek-objek tersebut sebagai bayangan dengan lighting dimatikan dan warna abu-abu, sambil menerapkan transformasi proyeksi bayangan. Fungsi idle() digunakan untuk membuat animasi dengan terus menambah sudut rotasi dan memicu render ulang menggunakan glutPostRedisplay(), sehingga adegan tampak berputar terus-menerus secara otomatis.

3.10 Fungsi loadTexture()

```
GLuint loadTexture(Image* image) {
    GLuint textureId;
    glGenTextures(1, &textureId);
    glBindTexture(GL_TEXTURE_2D, textureId);

    // Set texture parameters untuk kualitas yang lebih baik
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

    // Upload texture data
```

```
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->width,
image->height,
                 0, GL_RGB, GL_UNSIGNED_BYTE, image->pixels);

    return textureId;
}

GLuint createDefaultTexture() {
    GLuint textureId;
    glGenTextures(1, &textureId);
    glBindTexture(GL_TEXTURE_2D, textureId);

    // Create simple green checkerboard pattern
    unsigned char defaultTexture[64][64][3];
    for (int i = 0; i < 64; i++) {
        for (int j = 0; j < 64; j++) {
            if ((i/8 + j/8) % 2 == 0) {
                defaultTexture[i][j][0] = 34;  // R - Dark green
                defaultTexture[i][j][1] = 139; // G
                defaultTexture[i][j][2] = 34;  // B
            } else {
                defaultTexture[i][j][0] = 50;  // R - Light green
                defaultTexture[i][j][1] = 205; // G
                defaultTexture[i][j][2] = 50;  // B
            }
        }
    }

    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 64, 64, 0, GL_RGB,
GL_UNSIGNED_BYTE, defaultTexture);

    return textureId;
}

// 2. PERBAIKI initRendering - TAMBAHKAN ERROR CHECKING
void initRendering() {
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glEnable(GL_COLOR_MATERIAL);

    // Load texture dengan error checking
    Image* image = loadBMP("rumput.bmp");
    if (image ) {
        _textureId = loadTexture(image);
        delete image;
        printf("Texture loaded successfully! ID: %d\n",
_textureId);
    } else {
        printf("ERROR: Failed to load rumput.bmp!\n");
```

```
        // Create default texture jika gagal load
        _textureId = createDefaultTexture();
    }
}
```

Kode tersebut mengilustrasikan implementasi sistem pengelolaan tekstur dalam aplikasi grafis OpenGL, yang terdiri dari tiga fungsi utama untuk pemuatan, pembuatan, dan inisialisasi tekstur. Fungsi loadTexture() berperan sebagai pengolah data gambar menjadi tekstur OpenGL yang dapat digunakan untuk rendering, dimulai dengan pembuatan identitas tekstur baru melalui glGenTextures() dan pengikatan tekstur tersebut menggunakan glBindTexture(). Konfigurasi kualitas tekstur diatur melalui parameter GL_TEXTURE_MIN_FILTER dan GL_TEXTURE_MAG_FILTER dengan nilai GL_LINEAR untuk menghasilkan interpolasi linear yang memberikan tampilan lebih halus dibandingkan metode nearest-neighbor, sementara GL_TEXTURE_WRAP_S dan GL_TEXTURE_WRAP_T diatur ke GL_REPEAT agar tekstur dapat diulang secara seamless untuk permukaan yang lebih luas dari dimensi gambar asli. Data piksel tekstur kemudian diunggah ke GPU melalui fungsi glTexImage2D() yang mengkonversi array piksel dari objek Image ke format yang diproses oleh OpenGL, dengan spesifikasi format warna RGB dan tipe data unsigned byte.

Fungsi kedua, createDefaultTexture(), menyediakan mekanisme fallback dengan membuat tekstur checkerboard hijau berukuran 64×64 piksel secara prosedural sebagai pengganti ketika pemuatan tekstur dari file gagal. Pola checkerboard dibuat dengan dua warna hijau berbeda (dark green dan light green) berdasarkan posisi piksel dalam grid 8×8, memberikan visualisasi yang jelas bahwa tekstur default sedang digunakan. Fungsi ketiga, initRendering(), mengintegrasikan sistem tekstur dengan inisialisasi rendering keseluruhan dengan mengaktifkan fitur OpenGL seperti depth testing, pencahayaan, dan normalisasi vektor normal. Fungsi ini juga menerapkan mekanisme penanganan kesalahan yang robust, di mana tekstur "rumput.bmp" dimuat melalui loadBMP(), dan jika berhasil, diproses melalui loadTexture(), namun jika gagal, sistem akan secara otomatis beralih ke tekstur default melalui pemanggilan createDefaultTexture().

Implementasi ini memastikan bahwa aplikasi grafis tetap berfungsi secara visual bahkan ketika file tekstur eksternal tidak tersedia, mendemonstrasikan pendekatan defensive programming dalam pengembangan aplikasi grafis interaktif.

BAB IV

PENUTUP

4.1 Kesimpulan

Proyek game berbasis OpenGL ini berhasil mengimplementasikan model tiga dimensi Gerobak Juara beserta elemen-elemen pendukungnya seperti pohon, rumah, rintangan, dan awan dengan pendekatan pemodelan berbasis vertex dan primitif dasar di OpenGL. Setiap objek dibangun secara modular menggunakan kombinasi kubus, silinder, dan bola, di mana transformasi geometri (translasi, rotasi, dan skala) diatur melalui pasangan glPushMatrix() dan glPopMatrix() untuk menjaga isolasi transformasi pada masing-masing komponen. Contohnya, fungsi drawGerobak() menampilkan detail struktur gerobak, termasuk blok-blok tubuh utama, roda, serta animasi kemiringan saat berbelok, yang keseluruhannya disusun dari array vertex dan diproses melalui drawBlockCustom() dengan parameter warna berbeda. Demikian pula, fungsi buatOrang() merakit tubuh karakter dengan skala dan rotasi yang tepat untuk menghasilkan sosok manusia di dalam gerobak, lengkap dengan topi, lengan, dan kaki yang menekuk untuk menimbulkan kesan sedang duduk.

Penggunaan teknik pencahayaan, shading, dan tekstur meningkatkan realisme tampilan visual. Sistem pencahayaan diatur melalui fungsi seperti glLightfv() untuk menentukan posisi dan sifat cahaya, sementara properti material diatur dengan glMaterial*(), sehingga objek-objek termasuk pohon, gerobak, dan rumah menampilkan efek ambient, diffuse, dan specular sesuai model Phong. Bayangan objek dihasilkan dengan menerapkan proyeksi bayangan menggunakan matriks yang dihitung oleh glShadowProjection(), lalu objek dan bayangan digambar berturut-turut dengan mode pencahayaan aktif dan nonaktif untuk menciptakan efek siluet ke permukaan tanah. Untuk tekstur, fungsi loadTexture() memuat data gambar (misalnya “rumput.bmp”) ke dalam OpenGL, menerapkan parameter GL_LINEAR untuk filtering dan GL_REPEAT untuk wrapping, sedangkan createDefaultTexture() menyediakan tekstur checkerboard hijau sebagai fallback jika pemuatan gambar gagal.

Dari sisi interaktivitas, proyek ini menyediakan kontrol untuk memanipulasi orientasi kamera dan pergerakan objek secara real-time. Fungsi idle() secara kontinu memperbarui sudut rotasi (rx dan ry) untuk menciptakan animasi adegan yang terus berputar, sementara input keyboard (tidak dijelaskan secara eksplisit di bagian yang diunggah) memungkinkan pengguna untuk mengontrol gerakan gerobak dan sudut pandang kamera. Response interaktif ini sejalan dengan tujuan merancang kontrol yang responsif dalam lingkungan grafis 3D menggunakan OpenGL, sehingga pengguna dapat merasakan sensasi mengemudikan gerobak di antara pohon dan rintangan serta memperhatikan bayangan yang dihasilkan.

Secara keseluruhan, proyek ini telah mencapai tujuan yang dirumuskan, yaitu mengimplementasikan model Gerobak Juara 3D yang realistis, menerapkan teknik pencahayaan dan shading untuk visualisasi yang meyakinkan, serta mengelola kontrol interaktif dalam lingkungan OpenGL. Struktur modular dan penggunaan transformasi hierarkis membentuk fondasi yang kuat untuk pengembangan proyek grafis 3D yang lebih kompleks di masa depan. Pengalaman merancang fungsi-fungsi seperti drawTrees(), drawBarriers(), hingga render() dan glShadowProjection() menambah pemahaman mendalam tentang pipeline OpenGL, sehingga modul ini dapat diperluas untuk menambahkan elemen seperti sistem fisika, deteksi tabrakan, atau shader kustom di iterasi berikutnya. Dengan demikian, laporan ini tidak hanya mencerminkan keberhasilan implementasi teknis, tetapi juga membuka peluang eksplorasi lebih lanjut dalam komputasi grafis.

DAFTAR PUSTAKA

Erwinsyah, A., Andriany, L., & Fithra, H. (2019). Three-Dimensional Text Applications with OpenGL. *Journal of Physics: Conference Series*, *1364*(1), 012048.

Heo, Y. J., Kim, D., Lee, W., Kim, H., Park, J., & Chung, W. K. (2019). Collision detection for industrial collaborative robots: A deep learning approach. *IEEE Robotics and Automation Letters*, *4*(2), 740–746.

Krupiński, R. (2021). Simulation and analysis of floodlighting based on 3D computer graphics. *Energies*, *14*(4), 1042.

Posada, J., Toro, C., Barandiaran, I., Oyarzun, D., Stricker, D., De Amicis, R., Pinto, E. B., Eisert, P., Döllner, J., & Vallarino, I. (2015). Visual computing as a key enabling technology for industrie 4.0 and industrial internet. *IEEE Computer Graphics and Applications*, *35*(2), 26–40.

LAMPIRAN

Link video

https://drive.google.com/drive/folders/1A3XioE6vAjZjbK4gWb0XpyadBITaKNsL?usp=sharing

Pembagian Tugas :

Bramantyo Kunni Nurrisqi	• Membuat objek orang • Memberikan teksture dan shadowing
Bagus Athallah	• Membuat objek pohon, gerobak, jalan • Membuat animasi gerak
Agathan Khairy Bowo Laksono	• Membuat objek barrier dan rumah • Membuat animasi gerak
Al Vanza Aries R	• Membuat objek tempat sampah • Menyusun proposal