LAPORAN TUGAS BESAR PRAKTIKUM
GRAFIKA DAN KOMPUTASI VISUAL

”Simulasi *Game* Obstacle Drift menggunakan OpenGL”

Dibuat untuk memenuhi tugas besar Praktikum Grafika dan Komputasi Visual Asisten Praktikum: Miriam Stefani Abigail Hutapea dan Pujiani Rahayu Agustin

Disusun Oleh:

Akmal Fauzan Azima	24060123120022
Amara Putri Soniaji	24060123120004
Brilianita Eva Syafitri	24060123130051
Cacania Pasu Nalaung Siregar	24060123120033

DEPARTEMEN INFORMATIKA
FAKULTAS SAINS DAN MATEMATIKA
UNIVERSITAS DIPONEGORO
2025

DAFTAR ISI

BAB I
PENDAHULUAN

1.1 Latar Belakang

Kemajuan komputasi grafis mempengaruhi berbagai lini kehidupan, salah satunya hiburan digital. Salah satu hiburan digital yang menarik berbagai kalangan ialah *game*. Baik secara profesional maupun sebagai bagian dari proses pembelajaran, pengembangan *game* semakin banyak menarik individu.

Salah satu teknologi yang dapat digunakan untuk pembuatan *game* adalah OpenGL. OpenGL merupakan antarmuka pemrograman aplikasi (Application Programming Interface) yang bersifat lintas platform dan digunakan secara luas dalam pembuatan aplikasi visual interaktif, khususnya grafik 2D dan 3D (Angel & Shreiner, 2012).

Game yang dapat dirancang dan direalisasikan dengan pemrograman melalui openGL, salah satunya ialah "simulasi *game* obstacle drift". *Game* ini merupakan *game* 3D yang mensimulasikan pergerakan mobil yang harus menghindari rintangan pada lintasan, dengan mengimplementasikan berbagai elemen grafis seperti animasi, deteksi tabrakan (*collision detection*), serta interaksi melalui *input* pengguna. Dengan merancang dan membangun simulasi ini, diharapkan dapat memahami prinsip-prinsip dasar grafika komputer, sekaligus mengembangkan keterampilan dalam membangun aplikasi visual menggunakan teknologi OpenGL.

1.2 Rumusan Masalah

1.2.1 Bagaimana mengimplementasikan fungsi-fungsi dari OpenGL untuk membangun simulasi game?

1.2.2 Bagaimana merancang objek permainan, seperti mobil dan rintangan, agar dapat ditampilkan dan berinteraksi dalam lintasan permainan?

1.2.3 Bagaimana menerapkan logika permainan, termasuk deteksi tabrakan (collision detection) dan sistem kontrol pergerakan mobil melalui input dari pengguna?

1.3 Tujuan

1.3.1 Mengimplementasikan teknologi OpenGL untuk membuat simulasi *game* 3D.

1.3.2 Merancang dan merealisasikan objek-objek permainan yang dapat bergerak serta berinteraksi secara visual di dalam lingkungan permainan.

1.3.3 Menerapkan prinsip-prinsip dasar grafika komputer, seperti transformasi objek, pergerakan dinamis, dan interaksi pengguna, dalam bentuk proyek game “Obstacle Drift”.

BAB II
DASAR TEORI

2.1 Komputasi Grafis

Komputasi grafis adalah bidang ilmu komputer yang berkaitan dengan penciptaan, penyajian, dan manipulasi gambar secara visual dengan menggunakan komputer. Komputasi grafis digunakan secara luas dalam berbagai aplikasi, mulai dari desain grafis, simulasi, visualisasi data, hingga pengembangan *game* dan film animasi (Hearn & Baker, 2014).

Dalam konteks pengembangan *game*, komputasi grafis memegang peran penting dalam menghasilkan visualisasi dunia virtual yang interaktif dan realistis. Komputasi grafis memungkinkan representasi objek dua dimensi (2D) dan tiga dimensi (3D) dengan menggunakan teknik pemodelan, *rendering*, pencahayaan, dan transformasi geometri. Proses dalam komputasi grafis umumnya melibatkan:

- ***Modeling**:* Membuat representasi objek dalam bentuk matematis.
- ***Rendering**:* Mengubah model menjadi gambar melalui proses proyeksi, pencahayaan, dan pewarnaan.
- **Transformasi**: Mengatur posisi, skala, dan rotasi objek di dalam ruang virtual.
- ***Clipping & Rasterization**:* Menentukan bagian dari gambar yang akan ditampilkan pada layar dan mengubah koordinat geometrik menjadi pixel.

Komputasi grafis modern banyak didukung oleh perangkat keras grafis seperti GPU (Graphics Processing Unit) dan pustaka perangkat lunak grafis seperti OpenGL (Hearn & Baker, 2014).

2.2 OpenGL

OpenGL *(Open Graphics Library)* adalah pustaka grafis lintas platform dan lintas bahasa pemrograman yang digunakan

untuk merender grafik 2D dan 3D. OpenGL dikembangkan pertama kali oleh Silicon Graphics Inc. (SGI) pada awal 1990-an dan kini menjadi standar de facto dalam pengembangan grafis komputer, termasuk dalam industri *game* dan simulasi.

OpenGL menyediakan serangkaian API (Application Programming Interface) yang memungkinkan pengembang untuk mengakses fitur-fitur grafis tingkat rendah pada perangkat keras grafis. OpenGL bersifat *state machine*, artinya ia mempertahankan status internal yang memengaruhi cara *rendering* dilakukan (OpenGL ARB, n.d.). Beberapa fitur utama OpenGL antara lain:

- Manipulasi objek 2D dan 3D.
- Pengaturan proyeksi (ortografis dan perspektif).
- Penerapan transformasi geometris (translasi, rotasi, scaling).
- Pengolahan warna dan pencahayaan.
- *Rendering* berbasis primitif grafis (titik, garis, segitiga).
- Buffering dan manajemen viewport.

Dalam pengembangan simulasi *game* ini, digunakan GLUT (OpenGL Utility Toolkit) sebagai pustaka bantu yang menyederhanakan pembuatan jendela, pengaturan *input* keyboard dan mouse, serta pengelolaan peristiwa (event handling).

2.3 Game

Game adalah aplikasi interaktif yang dirancang untuk memberikan hiburan, tantangan, dan pengalaman bermain melalui keterlibatan pengguna dalam sistem aturan dan tujuan tertentu. *Game* dapat bersifat edukatif, simulatif, atau rekreatif, dan dapat diterapkan dalam berbagai platform seperti komputer, konsol, maupun perangkat mobile. Dalam konteks teknis, *game* terdiri dari beberapa elemen penting, antara lain:

- ***Gameplay***: Mekanika interaksi antara pemain dan sistem *game*.

- **Grafik**: Representasi visual yang menampilkan objek, latar, dan animasi.
- **Kontrol**: Sistem *input* yang memungkinkan pemain berinteraksi dengan *game*.
- **Logika *Game***: Aturan dan alur permainan yang menentukan bagaimana *game* berlangsung.
- **Audio**: Efek suara dan musik untuk meningkatkan pengalaman bermain.

Game **Obstacle Drift** yang dikembangkan dalam proyek ini merupakan sebuah *game* simulasi sederhana berbasis 2D, yang menguji ketangkasan pemain dalam menghindari rintangan saat mengemudikan objek kendaraan. *Game* ini dirancang menggunakan OpenGL dan GLUT, dengan fokus pada pengolahan grafis, deteksi tabrakan, dan pengendalian objek secara real-time.

BAB III
PEMBAHASAN

3.1 Primitif Drawing

Primitif drawing adalah teknik dasar dalam pemrograman grafis OpenGL untuk menggambar bentuk-bentuk geometris sederhana seperti titik (GL_POINTS), garis (GL_LINES), segitiga (GL_TRIANGLES), dan persegi (GL_QUADS). Setiap objek kompleks dalam dunia grafis 2D maupun 3D biasanya dibangun dari kombinasi primitif ini. Dengan menggunakan perintah glBegin() dan glEnd(), kita mendefinisikan serangkaian koordinat (glVertex) yang membentuk elemen visual di layar. Teknik ini merupakan fondasi dari semua gambar grafis dalam OpenGL dan penting untuk memahami bagaimana objek dibentuk secara matematis dalam ruang koordinat. Penerapan primitif *drawing* pada kode *game* ini ada beberapa, diantaranya:

3.1.1 Garis (GL_LINES)

```
// Garis tengah jalan
glDisable(GL_LIGHTING);
glColor3fv(putih);
glLineWidth(10.0);
glBegin(GL_LINES);
for (int i = -10; i < roadLength; i += 5) {
    if (i % 10 == 0) {
        glVertex3f(0.0, -0.47, i);
        glVertex3f(0.0, -0.47, i + 3);
    }
}
glEnd();
```

Kode ini membuat garis horizontal yang mewakili jalur jalan atau garis putus-putus pada aspal.

3.1.2 Segiempat (GL_QUADS)

```
// Kiri jalan
glBegin(GL_QUADS);
glVertex3f(-roadWidth * 15, -0.49, -10.0);
glVertex3f(-roadWidth/2, -0.49, -10.0);
glVertex3f(-roadWidth/2, -0.49, roadLength);
glVertex3f(-roadWidth * 15, -0.49, roadLength);
glEnd();

// Kanan jalan
glBegin(GL_QUADS);
glVertex3f(roadWidth/2, -0.49, -10.0);
glVertex3f(roadWidth * 15, -0.49, -10.0);
glVertex3f(roadWidth * 15, -0.49, roadLength);
glVertex3f(roadWidth/2, -0.49, roadLength);
glEnd();
```

Untuk bagian dinding kiri dan kanan jalan. Kode ini membentuk persegi atau persegi panjang dari 4 titik. Kode ini juga digunakan untuk membuat bagian langit, finish line, dan lain-lain.

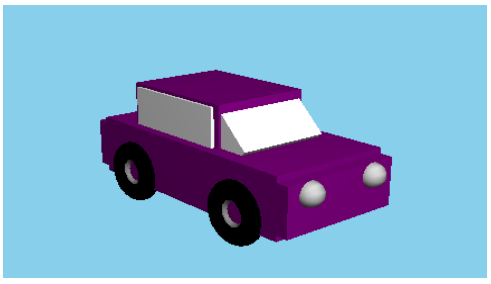
3.1.3 Lingkaran dan Bola

```
// Bulan utama (terang)
glColor3fv(moonBright);
glPushMatrix();
glTranslatef(0.8f, 0.85f, 0.0f);  // Posisi bulan
glutSolidSphere(0.1, 50, 50);
glPopMatrix();

// Bayangan bulan (seolah bulan sabit)
glColor3fv(skyColor);
glPushMatrix();
glTranslatef(0.82f, 0.85f, 0.01f);  // Geser sedikit ke kanan & depan
glutSolidSphere(0.1, 50, 50);
glPopMatrix();
```

Menggunakan `glutSolidSphere` untuk bentuk bulat. Kode ini diterapkan pada objek batu, roda, rintangan, dan elemen dekorasi lainnya. Meskipun bukan bagian dari `glBegin(...)`, fungsi ini menghasilkan bentuk primitif berbasis titik dan segitiga (otomatis oleh GLUT).

3.2 Objek

3.2.1 Mobil 3D



```
void buatMobil() {
    // Bodi utama mobil (kotak)
    glPushMatrix();
    glColor3fv(ungu);
    glScalef(1.0, 0.5, 2.0);
    glutSolidCube(1.0);
    glPopMatrix();

    // Bagian kabin mobil (kotak) - Struktur ungu
    glPushMatrix();
    glColor3fv(ungu);
    glTranslatef(0.0, 0.4, -0.2);
    glScalef(0.8, 0.4, 1.0);
    glutSolidCube(1.0);
    glPopMatrix();

    // Kaca depan (trapesium)
    glPushMatrix();
    glColor3fv(putih);
    glTranslatef(0.0, 0.35, 0.3);
    glRotatef(-45.0, 1.0, 0.0, 0.0);
    glScalef(0.75, 0.25, 0.45);
    glutSolidCube(1.0);
    glPopMatrix();
```

Pembuatan objek mobil memanfaatkan primitif dari GLUT seperti `glutSolidCube` dan `glutSolidSphere`, yang kemudian dimodifikasi menggunakan transformasi dasar OpenGL seperti `glTranslatef`, `glScalef`, dan `glRotatef`. Bentuk dasar mobil terdiri dari body utama dan kabin yang keduanya dibentuk dengan kubus berwarna ungu. Kaca depan dan kaca samping dibuat menggunakan kubus tipis berwarna putih, yang dimiringkan atau diposisikan sesuai dengan letaknya pada mobil. Bagian depan (moncong) dan belakang mobil juga dibentuk dari kubus yang diperpendek, sementara lampu depan dan belakang menggunakan bola kecil berwarna putih dan merah. Empat roda mobil dipasang pada posisi yang sesuai di kiri dan kanan bagian depan serta belakang mobil, ini menggunakan fungsi tambahan buatRoda().Berikut adalah code nya:

```
void buatRoda(float x, float y, float z) {
    glPushMatrix();
    glTranslatef(x, y, z);
    glRotatef(90.0, 0.0, 1.0, 0.0);  // Rotasi pertama untuk orientasi yang benar
    // Kemudian terapkan rotasi roda pada sumbu yang sesuai
    glRotatef(wheelRotation, 0.0, 0.0, 1.0);
```

```
    // Ban (silinder hitam)
    glColor3fv(hitam);
    glutSolidTorus(0.08, 0.18, 20, 20);

    // Pelek (silinder perak)
    glColor3fv(abu);
    glutSolidTorus(0.04, 0.13, 10, 10);

    glPopMatrix();
}
```

3.2.2 Landscape

Landscape adalah representasi visual dari lingkungan atau latar belakang tempat objek-objek dalam 3D. Dalam *game* ini, fungsi `buatLandscape()` digunakan untuk menggambar seluruh lingkungan tempat mobil bergerak. Elemen-elemen yang dibentuk oleh landscape ini mencakup: tanah dasar, rumput, jalan utama, dan dekorasi (rumah & pohon). Kodenya adalah sebagai berikut:

```
// Membuat landscape dengan perumahan dan pohon
void buatLandscape() {
    // Tanah dasar
    glPushMatrix();
    glColor3fv(coklatTanah);
    glBegin(GL_QUADS);
    glVertex3f(-roadWidth * 15, -0.5, -10.0);
    glVertex3f(roadWidth * 15, -0.5, -10.0);
    glVertex3f(roadWidth * 15, -0.5, roadLength);
    glVertex3f(-roadWidth * 15, -0.5, roadLength);
    glEnd();
    glPopMatrix();

    // Rumput di sekeliling jalan
    glPushMatrix();
    glColor3fv(isNight ? rumputMalam : rumputSiang);

    // Kiri jalan
    glBegin(GL_QUADS);
    glVertex3f(-roadWidth * 15, -0.49, -10.0);
    glVertex3f(-roadWidth/2, -0.49, -10.0);
    glVertex3f(-roadWidth/2, -0.49, roadLength);
    glVertex3f(-roadWidth * 15, -0.49, roadLength);
    glEnd();

    // Kanan jalan
    glBegin(GL_QUADS);
    glVertex3f(roadWidth/2, -0.49, -10.0);
    glVertex3f(roadWidth * 15, -0.49, -10.0);
    glVertex3f(roadWidth * 15, -0.49, roadLength);
    glVertex3f(roadWidth/2, -0.49, roadLength);
    glEnd();

    glPopMatrix();

    // Jalan dengan tekstur
    glPushMatrix();
    glDisable(GL_LIGHTING);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, jalanTexture);

    glColor3f(1.0f, 1.0f, 1.0f);
    glBegin(GL_QUADS);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-roadWidth/2, -0.48, -10.0);
    glTexCoord2f(5.0f, 0.0f); glVertex3f(roadWidth/2, -0.48, -10.0);
    glTexCoord2f(5.0f, roadLength / 10.0f); glVertex3f(roadWidth/2, -0.48, roadLength);
    glTexCoord2f(0.0f, roadLength / 10.0f); glVertex3f(-roadWidth/2, -0.48, roadLength);
    glEnd();

    glDisable(GL_TEXTURE_2D);
    glEnable(GL_LIGHTING);   // Nyalakan lagi
    glPopMatrix();

    // Garis tengah jalan
    glDisable(GL_LIGHTING);
    glColor3fv(putih);
    glLineWidth(10.0);
    glBegin(GL_LINES);
    for (int i = -10; i < roadLength; i += 5) {
        if (i % 10 == 0) {
            glVertex3f(0.0, -0.47, i);
            glVertex3f(0.0, -0.47, i + 3);
        }
    }
    glEnd();
    glEnable(GL_LIGHTING);

    glPopMatrix();

    // Gambar dekorasi (rumah dan pohon)
    for (const auto& deco : decorations) {
        // Hanya gambar yang berada pada jarak tertentu dari mobil
        if (deco.z > carZ - 50 && deco.z < carZ + 150) {
            switch (deco.type) {
                case 0: // Rumah tipe 1
                    buatRumahTipe1(deco.x, deco.z, deco.rotasi);
                    break;
                case 1: // Rumah tipe 2
                    buatRumahTipe2(deco.x, deco.z, deco.rotasi);
                    break;
                case 2: // Pohon besar
                    buatPohon(deco.x, deco.z, 1.5);
                    break;
                case 3: // Pohon kecil
                    buatPohonKecil(deco.x, deco.z);
                    break;
            }
        }
    }
}
```

Fungsi dimulai dengan menggambar tanah dasar menggunakan bidang datar `GL_QUADS` yang berwarna coklat, membentang luas ke kiri dan kanan jalan dan memberikan kesan permukaan bumi tempat semua objek berdiri. Kemudian, pada sisi kiri dan kanan jalan, digambar area rumput dengan warna yang menyesuaikan mode siang atau malam (rumputSiang atau rumputMalam). Rumput ini membatasi batas jalur mobil dan sekaligus menjadi indikator area keluar jalur.

Selanjutnya, bagian jalan utama digambar dengan lapisan tekstur menggunakan `glBindTexture` dan `glTexCoord2f`, yang menciptakan tampilan aspal realistis. Jalur ini ditempatkan sedikit lebih tinggi dari tanah agar tidak tenggelam dalam *rendering*. Untuk menambah elemen visual khas jalan raya, digambar pula garis-garis putih (marka jalan) di tengah jalan dengan `GL_LINES` yang muncul terputus-putus secara berkala.

Bagian terakhir dari fungsi ini adalah menempatkan dekorasi lingkungan, seperti rumah tipe 1, rumah tipe 2, pohon besar, dan pohon kecil. Objek-objek ini diambil dari vektor *decorations*, yang telah diisi secara acak sebelumnya. Untuk efisiensi *rendering*, hanya dekorasi yang berada dalam jarak pandang dekat mobil (antara carZ - 50 hingga carZ + 150) yang akan ditampilkan. Masing-masing objek kemudian dipanggil berdasarkan tipenya, dan ditempatkan dengan posisi serta orientasi yang telah diatur.

3.2.3 Rumah

```
// Membuat rumah tipe 1 (rumah sederhana kotak)
void buatRumahTipe1(float x, float z, float rotasi) {
    glPushMatrix();
    glTranslatef(x, 0.0, z);
    glRotatef(rotasi, 0.0, 1.0, 0.0);

    // Badan rumah
    glColor3fv(merahBata);
    glPushMatrix();
    glTranslatef(0.0, 1.0, 0.0);
    glScalef(3.0, 2.0, 2.5);
    glutSolidCube(1.0);
    glPopMatrix();

    // Atap rumah
    glColor3fv(coklatAtap);
    glPushMatrix();
    glTranslatef(0.0, 2.0, 0.0);
    glRotatef(-90, 1.0, 0.0, 0.0);
    glutSolidCone(3.2, 2.5, 4, 10);
    glPopMatrix();
```

Pembuatan rumah tipe 1 dilakukan dengan menggunakan parameter x dan z sebagai penentu posisi horizontal rumah, serta parameter rotasi untuk menentukan orientasi rotasi rumah terhadap sumbu vertikal (sumbu Y). Seluruh transformasi objek rumah dibungkus menggunakan fungsi `glPushMatrix()` dan *`glPopMatrix()`* untuk memastikan bahwa transformasi yang diterapkan pada objek rumah tidak memengaruhi objek lainnya dalam lingkungan 3D.

Bagian badan rumah dibentuk menggunakan objek kubus yang dimodifikasi. Transformasi pertama dilakukan dengan `glTranslatef(0.0, 1.0, 0.0)` untuk menggeser posisi kubus ke atas, sehingga badan rumah tidak berada di bawah bidang dasar. Selanjutnya, dilakukan transformasi skala menggunakan `glScalef(3.0, 2.0, 2.5)` untuk memperbesar dimensi objek menjadi lebih menyerupai bentuk bangunan rumah. Bentuk dasar badan rumah dibuat dengan fungsi `glutSolidCube(1.0)` dan diberi warna menyerupai batu bata dengan `glColor3fv(merahBata)`.

Atap rumah dibuat dengan bentuk kerucut yang dihasilkan melalui fungsi `glutSolidCone()`. Posisi atap digeser ke atas menggunakan `glTranslatef(0.0, 2.0, 0.0)` agar berada di atas badan rumah. Karena arah default dari kerucut adalah sejajar sumbu Z, dilakukan rotasi sebesar -90 derajat terhadap sumbu X menggunakan `glRotatef(-90, 1.0, 0.0, 0.0)` agar kerucut tegak lurus terhadap sumbu Y.

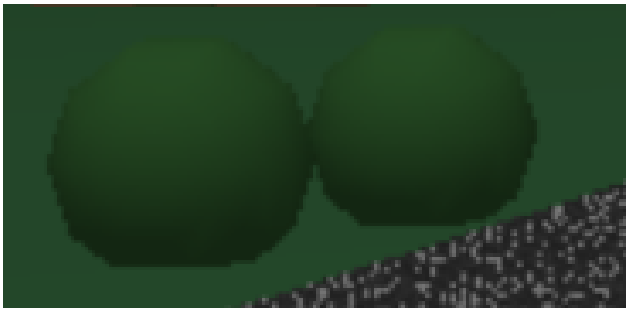
3.2.4 Pohon dan Semak

Pembuatan pohon menggunakan parameter x dan z untuk posisi horizontal pohon, dan scale untuk mengatur ukuran seluruh pohon. Di awal fungsi digunakan `glPushMatrix` dan `glPopMatrix` untuk mencegah transformasi agar tidak mempengaruhi objek lain. Pohon terdri dari batang dan daun, batang pohon berbentuk kubus yang dibuat dengan fungsi *library* GLUT yaitu `glutSolidCube`, sedangkan daun pohon dibentuk dari kerucut dari pemanggilan fungsi `glutSolidCone` yang diposisikan di atas batang, dirottasi dan diberi warna hijau.

```
void buatPohon(float x, float z, float scale) {
    glPushMatrix();
    glTranslatef(x, 0.0, z);
    glScalef(scale, scale, scale);

    // Batang pohon
    glColor3fv(coklatKayu);
    glPushMatrix();
    glTranslatef(0.0, 0.75, 0.0);
    glScalef(0.5, 1.5, 0.5);
    glutSolidCube(1.0);
    glPopMatrix();

    // Daun pohon
    glColor3fv(hijauDaun);
    glPushMatrix();
    glTranslatef(0.0, 1.0, 0.0);
    glRotatef(-90, 1.0, 0.0, 0.0);
    glutSolidCone(1.5, 3.0, 10, 10);
    glPopMatrix();

    glPopMatrix();
}
```

Pembuatan Semak melalui pembuatan fungsi `buatPohonKecil` yang berbentuk seperti bola padat yang dibentuk dari fungsi `library` GLUT `glutSolidSphere` yang diberi warna hijau daun. Bola digeser ke bawah pada sumbu Y `(glTranslatef(-.0, -0.1, 0.0)` agar tampak menempel ke tanah.

```
// Membuat pohon kecil (semak/bush)
void buatPohonKecil(float x, float z) {
    glPushMatrix();
    glTranslatef(x, 0.0, z);

    // Daun pohon kecil (bentuk bola)
    glColor3fv(hijauDaun);
    glPushMatrix();
    glTranslatef(0.0, -0.1, 0.0);
    glutSolidSphere(0.5, 10, 10);
    glPopMatrix();

    glPopMatrix();
}
```



3.2.5 Rintangan Batu

Objek batu ini merupakan rintangan dalam *game* yang digunakan untuk menghalangi laju mobil dan menciptakan tantangan bagi pemain. Batu ditampilkan di jalur mobil secara acak dan harus dihindari oleh pemain. Bila mobil menabrak batu, permainan akan

langsung berakhir (*game over*). Batu digambar dengan menggunakan fungsi `buatBatu(float x, float z)`. Kodenya adalah sebagai berikut:

```
// Membuat batu sebagai rintangan
void buatBatu(float x, float z) {
    glPushMatrix();
    glTranslatef(x, -0.1, z);

    glColor3fv(abuBatu);
    glScalef(1.0, 0.6, 1.0);
    glutSolidSphere(1.0, 10, 10);
```

Batu tidak dibuat secara manual satu per satu tetapi dengan menggunakan fungsi `generateObstacles()` yang berfungsi untuk membuat banyak batu secara otomatis. Fungsi ini berjalan saat *game* dimulai. Jadi, dalam interval setiap 15 satuan di sepanjang sumbu Z, ada peluang sebesar 65% pada setiap titik untuk munculnya sebuah batu. Keberadaan batu ditentukan secara acak menggunakan pernyataan kondisi `rand() % 100 < 65`, yang berarti tidak semua titik akan menghasilkan batu sehingga menciptakan variasi dalam jalur rintangan. Posisi batu pada sumbu X (horizontal) juga dibuat secara acak dalam batas lebar jalan, menggunakan rumus `(rand() % int(roadWidth) - roadWidth / 2) * 0.8f` yang memastikan batu tetap berada dalam jalur yang memungkinkan untuk dilalui mobil. Setiap batu yang berhasil dibuat disimpan dalam vektor obstacles sebagai objek dari struktur Obstacle, dengan nilai type = 0 yang menandakan bahwa rintangan tersebut adalah batu. Kodenya adalah sebagai berikut:

```
// Membuat rintangan
void generateObstacles() {
    srand(time(NULL));

    for (int z = 20; z < roadLength; z += 15) {
        if (rand() % 100 < 65) { // 65% kemungkinan ada rintangan
            Obstacle obs;
            obs.z = z;
            obs.x = (rand() % int(roadWidth) - roadWidth/2) * 0.8f; // posisi acak dalam jalan
            obs.type = 0; // jenis rintangan = 0 (batu)
            obstacles.push_back(obs);
        }
    }
}
```

3.2.6 Bulan

Pembuatan bulan sabit dilakukan dengan dua buah `glutSolidSphere` yang diposisikan berdekatan. Bulan utama dibuat terlebih dahulu dengan warna putih kekuningan menggunakan `glColor3fv(moonBright)`, lalu ditempatkan pada posisi tertentu di langit menggunakan `glTranslatef`. Untuk menciptakan efek bulan sabit, ditambahkan sebuah `glutSolidSphere` kedua yang sedikit digeser ke kanan dan ke depan dengan warna menyerupai langit malam (skyColor), sehingga menutupi sebagian bola pertama dan membentuk ilusi bulan sabit. Berikut adalah code nya:

```
// Warna Bulan sabit
GLfloat moonBright[] = {1.0f, 1.0f, 0.9f};  // Putih kekuningan
GLfloat skyColor[]   = {0.05f, 0.05f, 0.1f}; // Langit malam

// Bagian Bulan utama (terang)
glColor3fv(moonBright);
glPushMatrix();
glTranslatef(0.8f, 0.85f, 0.0f);  // Posisi bulan
glutSolidSphere(0.1, 50, 50);
glPopMatrix();
```

```
// Bagian Bayangan bulan (seolah bulan sabit)
glColor3fv(skyColor);
glPushMatrix();
glTranslatef(0.82f, 0.85f, 0.01f);  // Geser sedikit ke kanan & depan
glutSolidSphere(0.1, 50, 50);
glPopMatrix();
```

3.2.7 Bintang

Pembuatan bintang dilakukan dengan `GL_POINTS`, yang menggambar titik-titik kecil pada layar. Ukuran titik diatur menggunakan `glPointSize(2.0f)` agar bintang tampak lebih jelas. Warna bintang ditentukan oleh `glColor3fv (starColor)`, yaitu berwarna putih `GLfloat starColor[]= {1.0f, 1.0f, 1.0f, 1.0f}`.Berikut adalah codenya:

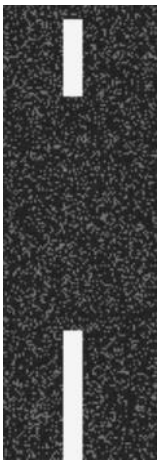
```
glPointSize(2.0f);
glBegin(GL_POINTS);
   glColor3fv(starColor);
   for (int i = 0; i < 100; ++i) {
       glVertex2f(starPos[i][0], starPos[i][1]);
   }
 glEnd();
```

3.2.8 Jalan dan Marka Jalan

Pembuatan jalan menggunakan segi empat `(GL_QUADS)` yang didefinisikan empat titik `glVertex3f`, pengaktifan tekstur dengan `glEnable(GL_TEXTURE_2D)`. Kemudian, untuk mengatur tampilan permukaan jalan yang berkelanjutan, diatur dengan nilai 5.0f dan `roadLength/`0.0f` pada `glTexCoord2f` sehingga tekstur akan terimplementasi di sepanjang lebar dan panjang jalan.

Adapun untuk menggambar marka di tengah jalan digunakan garis `glLineWidth(10.0)` sehingga ketebalan garis menjadi 10 piksel. Kemudian, digunakan loop untuk menggmbar serangkaian garis `(GL_LINES)` pendek. Mengatur perulangan dengan kondisi `if (i % 10 ==0)` agar garis tergambarkan secara putus-putus.

```
// Jalan dengan tekstur
    glPushMatrix();
    glDisable(GL_LIGHTING);
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, jalanTexture);

    glColor3f(1.0f, 1.0f, 1.0f);
    glBegin(GL_QUADS);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-roadWidth/2, -0.48, -10.0);
    glTexCoord2f(5.0f, 0.0f); glVertex3f(roadWidth/2, -0.48, -10.0);
    glTexCoord2f(5.0f, roadLength / 10.0f); glVertex3f(roadWidth/2, -0.48, roadLength);
    glTexCoord2f(0.0f, roadLength / 10.0f); glVertex3f(-roadWidth/2, -0.48, roadLength);
    glEnd();

    glDisable(GL_TEXTURE_2D);
    glEnable(GL_LIGHTING);    // Nyalakan lagi
    glPopMatrix();

    // Garis tengah jalan
    glDisable(GL_LIGHTING);
    glColor3fv(putih);
    glLineWidth(10.0);
    glBegin(GL_LINES);
    for (int i = -10; i < roadLength; i += 5) {
        if (i % 10 == 0) {
            glVertex3f(0.0, -0.47, i);
            glVertex3f(0.0, -0.47, i + 3);
        }
    }
    glEnd();
    glEnable(GL_LIGHTING);

    glPopMatrix();
```



3.3 Transformasi

Transformasi merupakan proses manipulasi objek dalam ruang 3D untuk menentukan posisi, ukuran, dan orientasi objek. Dalam *game* ini, transformasi diterapkan dengan menggunakan fungsi-fungsi OpenGL, seperti `glTranslatef` untuk translasi, `glRotatef` untuk rotasi, dan `glScalef` untuk skala. Salah satu contoh penerapannya adalah pada fungsi `buatPohon()`. Kodenya adalah sebagai berikut:

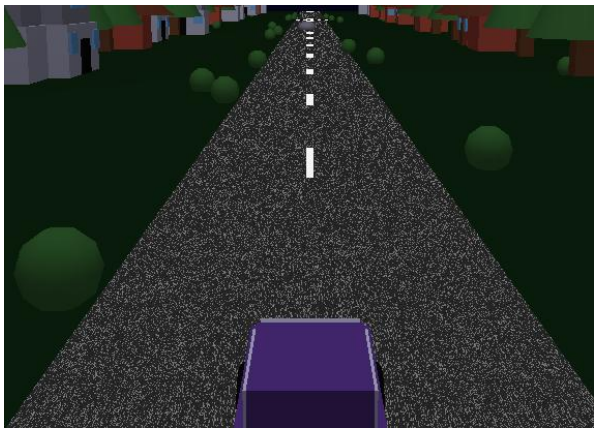
```
// Membuat pohon sebagai rintangan
void buatPohon(float x, float z, float scale) {
    glPushMatrix();
    glTranslatef(x, 0.0, z);
    glScalef(scale, scale, scale);

    // Batang pohon
    glColor3fv(coklatKayu);
    glPushMatrix();
    glTranslatef(0.0, 0.75, 0.0);
    glScalef(0.5, 1.5, 0.5);
    glutSolidCube(1.0);
    glPopMatrix();

    // Daun pohon
    glColor3fv(hijauDaun);
    glPushMatrix();
    glTranslatef(0.0, 1.0, 0.0);
    glRotatef(-90, 1.0, 0.0, 0.0);
    glutSolidCone(1.5, 3.0, 10, 10);
    glPopMatrix();

    glPopMatrix();
}
```

Fungsi `buatPohon(float x, float z, float scale)` digunakan untuk menggambar pohon di posisi tertentu di sepanjang sisi jalan. Transformasi pertama yang diterapkan adalah translasi menggunakan `glTranslatef(x, 0.0, z)`, yang memindahkan posisi dasar pohon ke koordinat (x, z) pada bidang horizontal sehingga pohon bisa tersebar secara acak di sekitar lingkungan. Selanjutnya, menggunakan `glScalef(scale, scale, scale)` untuk menerapkan skala seragam pada seluruh pohon. Nilai scale menentukan besar kecilnya pohon secara keseluruhan (tinggi, lebar, dan kedalaman) agar tidak semua pohon berukuran sama, menambah kesan alami.

Untuk bagian batang, diterapkan transformasi lokal, yaitu `glTranslatef(0.0, 0.75, 0.0)` untuk mengangkat posisi batang ke tengah vertikal dan `glScalef(0.5, 1.5, 0.5)` untuk menjadikan kubus menjadi balok tegak panjang. Pada bagian daun, dilakukan `glTranslatef(0.0, 1.0, 0.0)` untuk menaikkan posisi kerucut sehingga berada di atas batang, dan `glRotatef(-90, 1.0, 0.0, 0.0)` untuk memutar kerucut menghadap ke atas (default kerucut mengarah ke sumbu Z). Semua transformasi ini dibungkus dalam `glPushMatrix()`

dan `glPopMatrix()` untuk memastikan bahwa perubahan posisi, rotasi, dan skala hanya berlaku untuk objek pohon yang sedang digambar.

3.4 Proyeksi dan Animasi



3.4.1 Proyeksi

Dalam OpenGL, proyeksi adalah cara mentransformasikan objek 3D ke layar 2D. Terdapat dua jenis matriks utama :

- *Projection Matrix* : menentukan bagaimana dunia 3D "diproyeksikan" ke layar, menciptakan efek perspektif.
- *Modelview Matrix* : digunakan untuk mentransformasikan objek (translasi, rotasi, skala) atau kamera dalam dunia 3D.

Penerapan dalam kode *game* :

```
void reshape(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (GLfloat)w / (GLfloat)h, 0.1, 1000.0);
    glMatrixMode(GL_MODELVIEW);
}
```

- `glMatrixMode(GL_PROJECTION)`: mengatur bahwa transformasi yang akan dibuat adalah pada projection matrix.
- `gluPerspective(...)`: membuat efek perspektif yaitu objek yang jauh tampak lebih kecil. Parameter :
    a) 45.0 sudut pandang (*field of view*)
    b) aspect ratio : perbandingan lebar dan tinggi
    c) 0.1 dan 1000.0 : jarak pandang minimum dan maksimum *(near & far plane*)

3.4.2 Animasi

Animasi dalam OpenGL dilakukan dengan mengubah nilai transformasi (translasi, rotasi, dll.) secara terus-menerus menggunakan perulangan atau fungsi `idle()` dan `glutTimerFunc`. Penerapan dalam kode *game*:

```
void update(int value) {
    if (gameStarted && !gameOver) {
        carZ += carSpeed;
        wheelRotation -= carSpeed * 50.0f;
        ...
        glutTimerFunc(16, update, 0);
    }
    glutPostRedisplay();
}
```

Animasi Roda dan Pergerakan Mobil

- carZ += carSpeed : memindahkan mobil ke depan (arah sumbu Z).
- `wheelRotation -=` carSpeed * 50.0f : memutar roda sebanding dengan kecepatan.
- `glutTimerFunc(16, update, 0)` : memanggil fungsi update() setiap 16ms (~60 FPS).
- `glutPostRedisplay()`: memberitahu OpenGL untuk menggambar ulang layar.

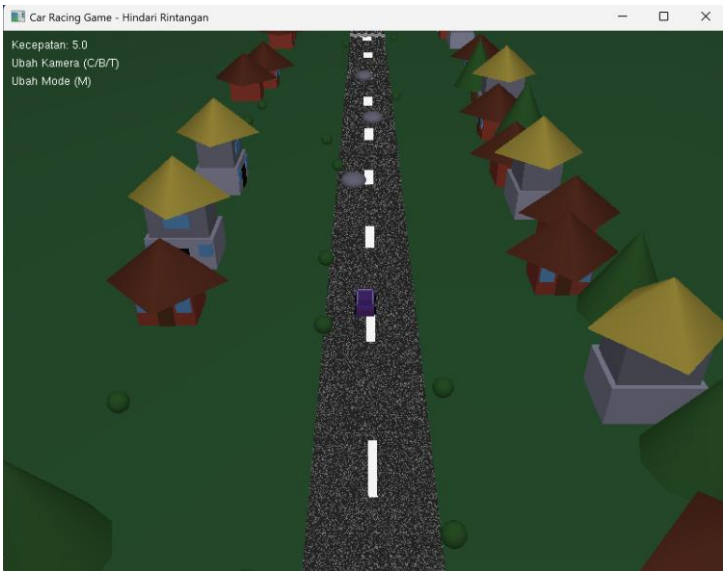
3.5 Kamera

Untuk mengatur posisi dan sudut pandang kamera, menggunakan fungsi `gluLookAt`. Fungsi tersebut memiliki tiga parameter utama, yaitu posisi kamera, titik yang dilihat, dan arah.

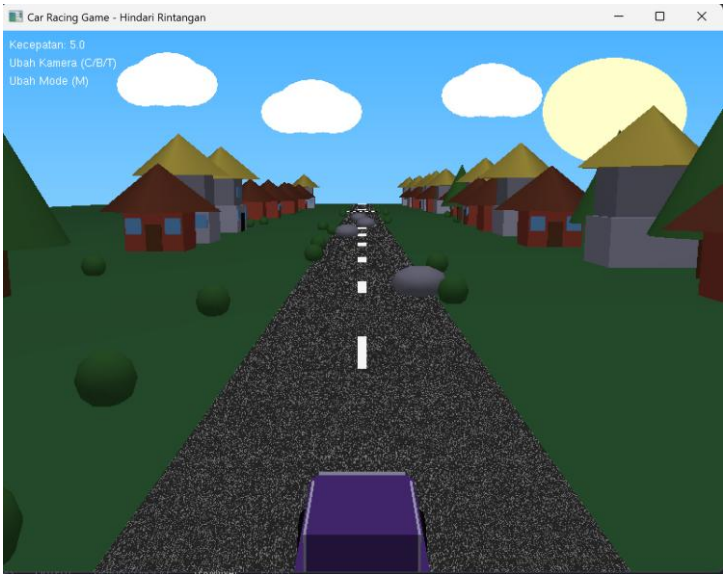
```
// Atur posisi kamera berdasarkan posisi mobil
if (topView) {
    // Kamera tampak atas
    gluLookAt(
        carX, 20.0, carZ - 30.0,  // Posisi kamera
        carX, 0.0, carZ,          // Titik yang dilihat (mobil)
        0.0, 1.0, 0.0             // Vektor up
    );
} else if (thirdPersonView) {
    // Tampilan third-person di belakang mobil
    gluLookAt(
        carX, 1.0 + cameraHeight, carZ - cameraDistance,    // Posisi kamera
        carX, 0.5, carZ + 10.0,                             // Titik yang dilihat (di depan mobil)
        0.0, 1.0, 0.0                                       // Vektor up
    );
} else if(sideTopView){
    gluLookAt(
        // Kamera tampak atas dan samping
        carX + 7.0, 7.0, carZ - 10.0,   // Posisi kamera
        carX, 0.0, carZ,                // Titik yang dilihat: ke arah mobil
        0.0, 1.0, 0.0                   // Vektor up
    );
} else {
    // Tampilan first-person dari dalam mobil
    gluLookAt(
        carX, 0.8, carZ + 0.5,      // Posisi kamera
        carX, 0.8, carZ + 10.0,     // Titik yang dilihat (di depan mobil)
        0.0, 1.0, 0.0               // Vektor up
    );
}
```

Dalam kode di atas, pengaturan kamera disesuaikan dengan mode tampilan yang sedang aktif, yaitu *topView*, *thirdPersonView*, *sideTopView*, dan *firstPersonView*.

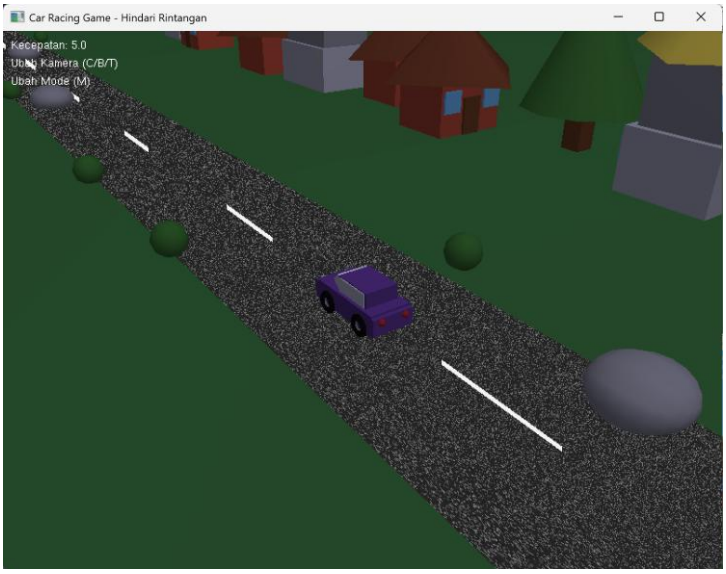
Jika topView bernilai true, kamera ditempatkan di atas mobil dengan koordinat (carX, 20.0, carZ, -30.0) yang memberikan tampilan dari atas sehingga pemain bisa melihat mobil dan lintasan secara luas. Titik yang dilihat adalah posisi mobil (carX, 0.0, carZ), dan vektor atasnya (0.0, 1.0, 0.0) menunjukkan bahwa sumbu Y tetap sebagai arah vertikal. Hasilnya sebagai berikut:



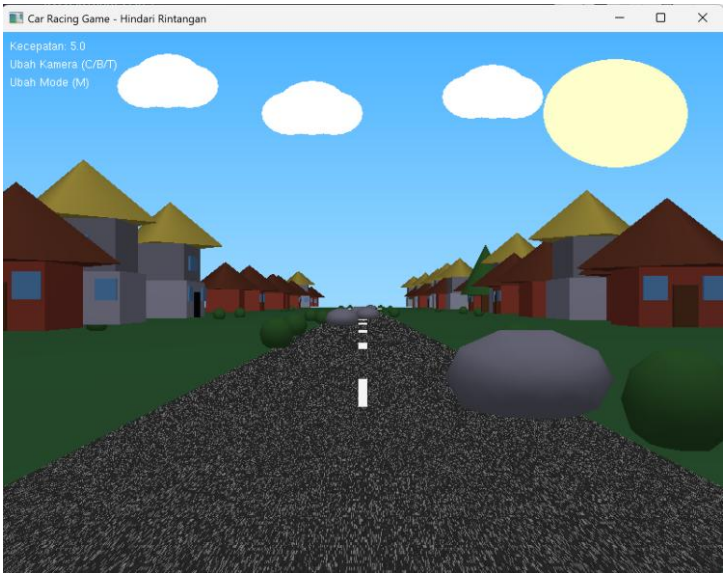
Jika *thirdPersonView* bernilai true, kamera berada di belakang mobil pada posisi (carX, 1.0 + cameraHeight, carZ - cameraDistance). Kamera diarahkan ke depan mobil, yaitu ke titik (carX, 0.5, carZ + 10.0). Ini menghasilkan tampilan *third-person*, di mana kamera mengikuti mobil dari belakang. Hasilnya sebagai berikut:



Jika *sideTopView* bernilai *true*, kamera ditempatkan di samping kanan mobil tinggi dan di belakang, yaitu pada posisi (carX + 7.0, 7.0, carZ - 10.0). Kamera tetap mengarah ke mobil di (carX, 0.0, carZ) yang menciptakan perspektif miring dari samping untuk melihat area sekitar mobil. Hasilnya sebagai berikut:



Jika tidak ada mode khusus yang aktif, kamera akan menggunakan tampilan default, yaitu firstPerson. Posisi kamera ditempatkan sedikit di atas permukaan pada (carX, 0.8, carZ + 0.5) yang seolah pemain duduk di dalam mobil dan diarahkan ke depan dengan titik penglihatan (carX, 0.8, carZ + 10.0). Hasilnya sebagai berikut:



3.6 Depth dan Lighting

3.6.1 Depth (kedalaman)

Depth buffer digunakan untuk memastikan objek yang lebih dekat ditampilkan di depan objek yang lebih jauh. Hal ini menjamin bagian objek yang tertutup oleh objek lain akan tersembunyi(tidak terlihat oleh pengguna). Contohnya dalam *game* adalah sebagai berikut:

- Rintangan yang lebih dekat akan menutupi rintangan di belakang.
- Rumah dan pohon yang berada jauh di belakang akan tidak terlihat bila terhalang oleh objek di depannya.

Berikut adalah code nya:

```
// Inisialisasi Depth Buffer
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
// Mengaktifkan Depth Test
glEnable(GL_DEPTH_TEST);
```

Kode di atas terletak di `main()`, yang mana inisialisasi untuk memberitahu OpenGL agar menyimpan informasi kedalaman pada setiap piksel. Kemudian depth testing diaktifkan, sehingga objek yang lebih dekat ke kamera akan menutupi objek yang lebih jauh secara otomatis.

```
// Membersihkan Depth Buffer sebelum menggambar
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Kode di atas terletak di display agar *depth buffer* tidak menyimpan informasi dari frame sebelumnya.

3.6.2 Lighting

Kode dalam *game* untuk menerapkan cahaya antara lain:

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

Sistem pencahayaan OpenGL `(GL_LIGHTING)` diaktifkan, beserta dengan sumber cahaya pertama `(GL_LIGHT0)`. Ini menyebabkan semua objek 3D yang digambar akan terkena efek cahaya, menciptakan efek terang-gelap tergantung posisi relatif terhadap cahaya. Selanjutnya:

```
GLfloat ambient[] = { 0.1f, 0.1f, 0.15f, 1.0f };
GLfloat diffuse[] = { 0.3f, 0.3f, 0.35f, 1.0f };

glLightfv(GL_LIGHT0, GL_AMBIENT,  ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE,  diffuse);
```

Ambient menjaga agar objek tetap terlihat walau tidak terkena cahaya langsung. Diffuse memberikan efek pada permukaan objek, misalnya mobil, yang menghadap ke arah datangnya cahaya akan tampak lebih terang, sementara sisi lainnya tampak lebih gelap.

```
GLfloat posisiCahaya[] = {0.0, 50.0, 0.0, 1.0};
glLightfv(GL_LIGHT0, GL_POSITION, posisiCahaya);
```

Posisi cahaya berasal dari atas, seperti matahari, sehingga objek akan mendapat pencahayaan lebih banyak dari atas yang menciptakan ilusi datangnya cahaya dari langit. Lalu:

```
glEnable(GL_COLOR_MATERIAL);
glEnable(GL_NORMALIZE);
```

`GL_COLOR_MATERIAL`: membuat warna dari objek `(glColor3fv(...))` dapat berinteraksi dengan pencahayaan.

`GL_NORMALIZE`: menjaga normal vektor tetap akurat walau objek mengalami `glScalef(...)`, sehingga pencahayaan tidak rusak.

```
glDisable(GL_LIGHTING);
// menggambar garis jalan, garis finish, dll
glEnable(GL_LIGHTING);
```

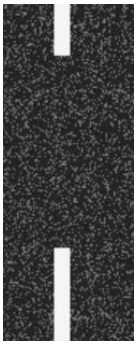
Kode di atas mencegah objek 2D seperti garis jalan, langit, atau teks agar tidak terkena pengaruh pencahayaan, sehingga tetap terlihat jelas dan kontras.

3.7 Tekstur

Penggunaan tekstur di *game* ini terletak pada tekstur aspal jalan. Tekstur dibuat secara prosedular menggunakan OpenGL. Pertama harus deklarasi variable GLuint sebagai variable penyimpan ID tekstur yang digunakan dalam *rendering*. Lalu mendefinisikan ukuran teskturnya sebesar 256x256 piksel. Setiap pksel diisi dengan warna abu-abu dimulai dari warna dasar gelap (baseGray). Kemudian, membuat fungsi `generateAsphaltTexture` untuk menghasilkan tekstur aspal. Di fungsi ini terdapat *nested loop* untuk mengiterasi setiap piksel. Di dalam loop ini, baseGray ditentukan dan ditambahkan variasi warna secara acak agar menimbulkan detail aspal seperti butiran terang atau retakan gelap yang diberi noise.

Untuk warna dasar yaitu abu-abu gelap diatur dalam rentang 50-79 sehingga `baseGray = 50 + rand() % 30`, sedangkan untuk menambahkan kontras dan variasi warna digunakan 2 kondisi if, if untuk perubahan dan if untuk retakan.

```
// Pembuatan tekstur aspal secara prosedural
#define TEXTURE_SIZE 256
GLubyte asphaltData[TEXTURE_SIZE][TEXTURE_SIZE][3];

void generateAsphaltTexture() {
    srand(time(NULL)); // Inisialisasi seed untuk generator angka acak

    // Membuat tekstur aspal kasar secara acak
    for (int i = 0; i < TEXTURE_SIZE; i++) {
        for (int j = 0; j < TEXTURE_SIZE; j++) {
            // Warna abu-abu gelap sebagai dasar aspal
            int baseGray = 50 + rand() % 30; // Rentang 50-79 (warna dasar gelap)

            // Meningkatkan kontras dan variasi
            if (rand() % 4 == 0) { // Perubahan lebih sering
                baseGray += 40 + rand() % 40; // Butiran lebih terang, rentang lebih lebar
            }
            if (rand() % 6 == 0) { // Retakan lebih sering
                baseGray -= 30 + rand() % 20; // Retakan lebih gelap, rentang lebih lebar
            }

            // Menambahkan noise yang lebih mencolok
             baseGray += (rand() % 11) - 5; // Menambahkan nilai antara -5 dan 5

            // Memastikan nilai tetap dalam rentang yang valid
            baseGray = baseGray > 255 ? 255 : baseGray;
            baseGray = baseGray < 0 ? 0 : baseGray;

            asphaltData[i][j][0] = baseGray;
            asphaltData[i][j][1] = baseGray;
            asphaltData[i][j][2] = baseGray;
        }
    }

    // Membuat dan menetapkan tekstur
    glGenTextures(1, &asphaltTexture);
    glBindTexture(GL_TEXTURE_2D, asphaltTexture);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, TEXTURE_SIZE, TEXTURE_SIZE, 0, GL_RGB, GL_UNSIGNED_BYTE, asphaltData);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
}
```



3.8 Toggle

3.8.1 Mode Siang

Untuk mode siang pada *game* ini dibuat melalui fungsi `buatLangitSiang`. Dalam fungsi ini terdapat latar belakang langit siang lengkap dengan matahari dan awan. Langit digambar sebagai sebuah persegi dari primitif `GL_QUADS` dengan gradasi warna biru, di mana bagian bawah lebih terang dan atasnya cenderung gelap. Matahari digambar sebagai bola padat `(glutSolidSphere)` berwarna kuning terang yang diposisikan di kanan atas layar. Lalu, awan ditambahkan dengan memanggil fungsi gambarAwan di beberapa posisi berbeda. Fungsi `buatAwan` digambar dari tiga bola yang dirangkai jadi satu memakai `glutSolidSphere` berwarna putih yang saling tumapng tindih.

```
void buatLangitSiang() {
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
    gluOrtho2D(0.0, 1.0, 0.0, 1.0);

    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();

    glDisable(GL_DEPTH_TEST);
    glDisable(GL_LIGHTING);

    // 1) Langit biru terang (gradasi)
    glBegin(GL_QUADS);
        glColor3f(0.6f, 0.85f, 1.0f);  // Biru muda terang (bawah)
        glVertex2f(0.0f, 0.5f);
        glVertex2f(1.0f, 0.5f);
        glColor3f(0.3f, 0.7f, 1.0f);  // Biru lebih dalam (atas)
        glVertex2f(1.0f, 1.0f);
        glVertex2f(0.0f, 1.0f);
    glEnd();

    // 2) Matahari
    glColor3f(1.0f, 1.0f, 0.8f); // Kuning terang (lebih putih)
    glPushMatrix();
    glTranslatef(0.85f, 0.85f, 0.0f);
    glutSolidSphere(0.1, 50, 50);
    glPopMatrix();

    // 3) Awan-awan
    // -- Pastikan digambar di sini, saat lighting dimatikan
    gambarAwan(0.2f, 0.9f);
    gambarAwan(0.4f, 0.85f);
    gambarAwan(0.65f, 0.88f);

    // Restore OpenGL state
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);

    glPopMatrix();
    glMatrixMode(GL_PROJECTION);
    glPopMatrix();
    glMatrixMode(GL_MODELVIEW);
}
```

```
// Fungsi untuk menggambar awan
void gambarAwan(float x, float y) {
    glPushMatrix();
    glTranslatef(x, y, 0.0f);
    glColor3f(1.0f, 1.0f, 1.0f); // Putih bersih

    // Gambar 3 lingkaran saling tumpuk
    glutSolidSphere(0.04, 20, 20);
    glTranslatef(0.03f, 0.01f, 0.0f);
    glutSolidSphere(0.05, 20, 20);
    glTranslatef(0.03f, -0.01f, 0.0f);
    glutSolidSphere(0.04, 20, 20);

    glPopMatrix();
}
```

3.8.2 Mode Malam

Mode malam pada *game* ini dibuat melalui fungsi `buatLangitMalam()`.Fungsi ini bertanggung jawab untuk menggambarkan suasana malam yang terdiri dari langit malam, bulan sabit, dan bintang-bintang.

1). Langit Malam

Langit malam digambar menggunakan primitif `GL_QUADS`. Warna langit malam diatur dengan gradasi gelap menggunakan warna RGB rendah yaitu `skyColor[] = {0.05f, 0.05f, 0.1f}` yang menciptakan nuansa biru gelap.

2). Bulan Sabit

Bulan digambarkan menggunakan dua buah bola padat `(glutSolidSphere)`:

Bola pertama merupakan bulan terang dengan warna putih kekuningan `moonBright[] = {1.0f, 1.0f, 0.9f}`.

Bola kedua digunakan sebagai bayangan atau penutup sebagian bola terang, menciptakan efek bulan sabit. Bola ini sedikit digeser ke kanan dan depan dengan warna skyColor (gelap), sehingga menutupi sebagian bulan utama.

3). Bintang-bintang

Sebanyak 100 bintang digambar dengan `GL_POINTS` menggunakan posisi acak yang dihasilkan satu kali saja dengan `rand()`. Warna bintang `(starColor)` ditentukan dan ukuran titik diatur dengan `glPointSize(2.0f)` agar tampak jelas.

```
// Membuat langit
void buatLangitMalam() {
    // ===== Inisialisasi STAR POSITIONS SEKALI SAJA =====
    static bool    starsInitialized = false;
    static float   starPos[100][2];    // 100 bintang, masing-masing (x,y)
    if (!starsInitialized) {
        starsInitialized = true;
        srand(42);  // seed tetap agar setiap run reproducible
        for (int i = 0; i < 100; ++i) {
            starPos[i][0] = (rand() % 100) / 100.0f;          // x dalam [0,1)
            starPos[i][1] = 0.5f + (rand() % 50) / 100.0f;    // y dalam [0.5,1)
        }
    }

    // ===== Begin menggambar =====
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
      glLoadIdentity();
      gluOrtho2D(0.0, 1.0, 0.0, 1.0);
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
      glLoadIdentity();
      glDisable(GL_DEPTH_TEST);
      glDisable(GL_LIGHTING);

      // 1) Quad langit malam
      glBegin(GL_QUADS);
        glColor3fv(darkSky);
        glVertex2f(0.0f, 0.5f);
        glVertex2f(1.0f, 0.5f);
        glVertex2f(1.0f, 1.0f);
        glVertex2f(0.0f, 1.0f);
      glEnd();
```

3.9 Interaksi antar Objek

Interaksi antar objek pada *game* ini terbagi menjadi empat bagian, yaitu:

3.9.1 Mobil dan Rintangan

Mobil yang dimainkan oleh pengguna bergerak maju ke arah sumbu Z dan dapat digerakkan ke kiri dan kanan melalui sumbu X. Di sepanjang jalan, terdapat objek rintangan berupa batu yang muncul secara acak. Interaksi terjadi ketika posisi mobil terlalu dekat dengan rintangan, dideteksi melalui perhitungan jarak Euclidean. Jika jarak antara mobil dan rintangan lebih kecil dari batas minimum yang ditentukan, maka dianggap terjadi tabrakan dan status permainan akan berubah menjadi "*Game* Over". Selain itu, terdapat rintangan berupa rumput di sisi kanan dan kiri jalan. Jika mobil melaju terlalu jauh ke samping dan mengenai area rumput, maka itu dianggap sebagai keluar jalur dan dan status permainan akan berubah menjadi "*Game* Over". Interaksi ini dituliskan dalam fungsi `checkCollision()` sebagai berikut:

```
// Memeriksa tabrakan antara mobil dan rintangan
bool checkCollision() {
    float carSize = 1.5f;

    for (const auto& obstacle : obstacles) {
        float dx = carX - obstacle.x;
        float dz = carZ - obstacle.z;
        float distance = sqrt(dx*dx + dz*dz);

        float collisionThreshold = (obstacle.type == 0) ? 1.5f : 1.8f; // Batu

        if (distance < collisionThreshold) {
            return true;
        }
    }

    // Memeriksa jika mobil keluar jalur
    if (fabs(carX) > roadWidth/2) {
        return true;
    }

    return false;
}
```

3.9.2 Mobil dan Finish Line

Finish line merupakan garis akhir dari lintasan balap yang berada pada posisi tertentu di sumbu Z dan ditentukan oleh variabel `finishZ`. Ketika mobil yang dikendalikan pengguna mencapai titik ini, kecepatan mobil dihentikan secara otomatis dengan mengatur nilai `carSpeed` menjadi nol. Sebagai bentuk interaksi visual, pada layar akan ditampilkan teks "FINISH!" sebagai penanda bahwa pengguna telah berhasil menyelesaikan permainan. Kondisi ini tidak memicu status *game*Over sehingga sistem tetap aktif meskipun mobil sudah berhenti. Interaksi ini memungkinkan pemain mengetahui bahwa tujuan permainan telah tercapai. Kode yang menandakan mobil sudah mencapai garis finish berada di fungsi `update()`, yaitu:

```
if (carZ >= finishZ) {
    gameOver = false;
    carSpeed = 0.0f;
}
```

Sedangkan, kode untuk menampilkan teks "FINISH!" berada di fungsi `display()`, yaitu:

```
if (carZ >= finishZ) {
    glColor3f(1.0, 1.0, 0.0);
    renderText(330, 320, "FINISH!");
}
```

3.9.3 Pengguna dan keyboard

Pengguna berinteraksi secara langsung dengan *game* melalui tombol keyboard untuk mengendalikan mobil dan mengatur berbagai mode dalam *game*. Interaksi untuk mengendalikan mocil dituliskan dalam fungsi `specialKeys()`, sebagai berikut:

```
void specialKeys(int key, int x, int y) {
    switch(key) {
        case GLUT_KEY_UP:
            if (!gameStarted && !gameOver) {
                gameStarted = true;
                generateObstacles();
                generateDecorations();
                glutTimerFunc(16, update, 0);
            }
            if (gameStarted && !gameOver) {
                if (carSpeed < maxSpeed) {
                    carSpeed += acceleration;
                }
            }
            break;
        case GLUT_KEY_DOWN:
            if (gameStarted && !gameOver) {
                if (carSpeed > 0) {
                    carSpeed -= acceleration * 2;
                    if (carSpeed < 0) carSpeed = 0;
                }
            }
            break;
        case GLUT_KEY_RIGHT:
            if (gameStarted && !gameOver) {
                carX -= steeringSpeed;
            }
            break;
        case GLUT_KEY_LEFT:
            if (gameStarted && !gameOver) {
                carX += steeringSpeed;
            }
            break;
    }
    glutPostRedisplay();
}
```

Keyboard “up” digunakan untuk menambah kecepatan mobil, “down” untuk menurunkan kecepatan mobil (mengerem), “right” untuk menggeser mobil ke kanan, dan “left” untuk menggeser mobil ke kiri.

Sedangkan, fungsi untuk mengatur berbagai mode dalam *game* dituliskan dalam fungsi `keyboard()`, sebagai berikut:

```
void keyboard(unsigned char key, int x, int y) {
    switch(key) {
        case 27: // ESC
            exit(0);
            break;
        case 'r': // Restart game
        case 'R':
            if (gameOver) {
                carX = 0.0f;
                carZ = 0.0f;
                carSpeed = 0.0f;
                gameStarted = false;
                gameOver = false;
                obstacles.clear();
                generateObstacles();
                generateDecorations();
                glutTimerFunc(16, update, 0);
            }
            break;
        case 'f':
        case 'F':
            if (gameStarted && !gameOver) {
                topView = false;
                thirdPersonView = false;
                sideTopView = false;
            }
            break;
        case 'c':
        case 'C':
            thirdPersonView = !thirdPersonView;
            topView = false; // Pastikan hanya satu view yang aktif
            sideTopView = false;
            break;
        case 'w':
            if (gameStarted && !gameOver) {
                if (carSpeed < maxSpeed) {
                    carSpeed += acceleration;
                }
            }
            break;
        case 's':
            if (gameStarted && !gameOver) {
                if (carSpeed > 0) {
                    carSpeed -= acceleration * 2;
                    if (carSpeed < 0) carSpeed = 0;
                }
            }
            break;
        case 'a':
            if (gameStarted && !gameOver) {
                carX += steeringSpeed;
            }
            break;
        case 'd':
            if (gameStarted && !gameOver) {
                carX -= steeringSpeed;
            }
            break;
        case 't':
        case 'T':
            topView = !topView;
            thirdPersonView = false; // Pastikan hanya satu view yang aktif
            sideTopView = false;
            break;
        case 'b':
        case 'B':
            sideTopView = !sideTopView;
            topView = false; // Pastikan hanya satu view yang aktif
            thirdPersonView = false;
            break;
        case 'm': // Toggle malam/siang
        case 'M':
            isNight = !isNight;
            if (isNight) {
                glClearColor(0.05f, 0.05f, 0.1f, 1.0f); // Warna langit malam
            } else {
                glClearColor(0.6f, 0.85f, 1.0f, 1.0f); // Warna langit siang
            }
            break;
    }

    glutPostRedisplay(); // Redraw layar setelah input keyboard
}
```

Keyboard “esc” digunakan untuk keluar dari *game*, “r” untuk *restart game*, “c” untuk mengganti tampilan kamera *thirdPersonView*, “w” untuk menambah kecepatan mobil, “s” untuk menurunkan kecepatan mobil (mengerem), “a” untuk menggeser mobil ke kanan, “d” untuk menggeser mobil ke kiri, “t” untuk mengganti tampilan kamera topView, “b” untuk mengganti tampilan kamera sideTopView, dan “m” untuk mengganti mode siang/malam.

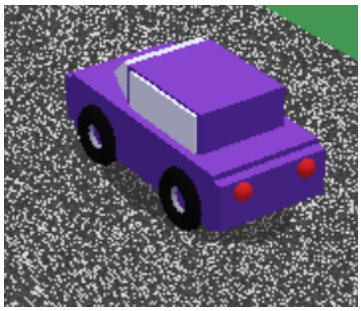
3.9.4 Interaksi dalam game

Interaksi yang terjadi selama permainan berlangsung dikendalikan secara terpusat melalui fungsi `update()`, yang terus dijalankan dalam interval tertentu menggunakan `glutTimerFunc`. Di dalam fungsi ini, posisi mobil pada sumbu Z akan diperbarui berdasarkan kecepatan saat ini dan rotasi roda disesuaikan agar sinkron dengan laju mobil. Fungsi ini juga melakukan pengecekan terhadap tabrakan dengan rintangan atau keluar dari batas jalan melalui pemanggilan `checkCollision()`. Jika terjadi tabrakan, permainan akan dihentikan dengan mengaktifkan status *game*Over, dan layar akan menampilkan informasi bahwa permainan telah selesai. Fungsi ini menciptakan interaksi dinamis antar objek yang terus-menerus diperbarui seiring waktu, memastikan permainan berjalan secara real-time.

3.10 Shadow

Penggunaan *shadow* pada *game* ini terdapat pada setiap objek yang di darat pada mode siang karena *shadow* merupakan efek pada objek yang terkena pancaran sinar matahari sehingga membentuk bayangan dari objek tersebut. Berikut implementasi shadow pada objek-objek di darat:

3.10.1 Mobil



Bayangan mobil digambar dengan bentuk bola pipih atau oval memanjang agar menyerupai bayangan mobil, diatur dengan `glScalef(1.2f, 0.05f, 2.2f)` dan posisinya diatur di bawah mobil dengan `glTranslate(0.0, 0.45, 0.0)`.

```
void buatMobil() {
    // --- Bayangan Mobil ---
    if (!isNight) {
        glDisable(GL_LIGHTING);
        glEnable(GL_BLEND); // Aktifkan blending untuk transparansi
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        glColor4f(0.1f, 0.1f, 0.1f, 0.4f); // Dark gray, semi-transparent
        glPushMatrix();
        glTranslatef(0.0, -0.45, 0.0); // Posisi bayangan sedikit di bawah mobil
        glScalef(1.2f, 0.05f, 2.2f); // Pipih dan lebar, menyesuaikan bentuk mobil
        glutSolidSphere(0.5, 20, 20); // Gunakan bola pipih untuk bayangan oval
        glPopMatrix();
        glDisable(GL_BLEND); // Nonaktifkan blending
        glEnable(GL_LIGHTING);
    }
}
```

3.10.2 Pohon

Bayangan pohon digambar dengan bola pipih `glScalef(1.8f, 0.05f, 1.8f)` agar berbentuk lingkaran, lalu posisinya diatur di bawah pohon dengan `glTranslatef(0.0, -0.1, 0.0)`

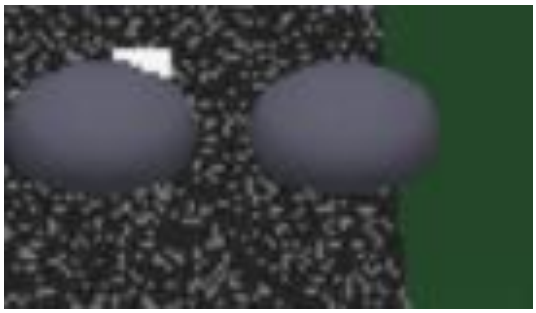
```
void buatPohon(float x, float z, float scale) {
    // --- Bayangan Pohon ---
    if (!isNight) {
        glDisable(GL_LIGHTING);
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        glColor4f(0.1f, 0.1f, 0.1f, 0.4f); // Dark gray, semi-transparent
        glPushMatrix();
        glTranslatef(0.0, -0.1, 0.0); // Posisi bayangan di dasar pohon
        glScalef(1.8f, 0.05f, 1.8f); // Ukuran bayangan disesuaikan dengan alas daun
        glutSolidSphere(0.5, 20, 20); // Bola pipih untuk bayangan melingkar
        glPopMatrix();
        glDisable(GL_BLEND);
        glEnable(GL_LIGHTING);
    }

    glPopMatrix();
}
```

3.10.3 Pohon kecil atau semak-semak

Bayangan pohon digambar dengan bola pipih `glScalef(0.7f, 0.03f, 0.7f)` agar membentuk lingkaran, posisinya diatur di bawah Semak dengan `glTranslatef(0.0, -0.1f, 0.0)`.

```
// --- Bayangan Pohon Kecil ---
if (!isNight) {
    glDisable(GL_LIGHTING);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    glColor4f(0.08f, 0.08f, 0.08f, 0.2f);
    glPushMatrix();
    glTranslatef(0.0, -0.1f, 0.0);
    glScalef(0.7f, 0.03f, 0.7f); // Lebar/panjang 0.7f, ketebalan 0.03f
    glutSolidSphere(0.5, 20, 20); // Gambar bola pipih
    glPopMatrix();
    glDisable(GL_BLEND);
    glEnable(GL_LIGHTING);
}
    glPopMatrix();
}
```

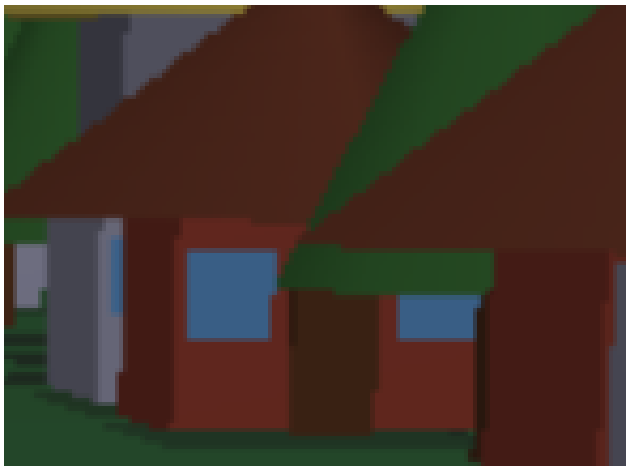
3.10.4 Batu



Bayangan batu digambar sebagai bola pipih `glScalef(1.0f, 0.05f, 1.0f)`, posisinya diatur di bawah batu dengan `glTranslatef(0.0, -0.2, 0.0)`

```
void buatBatu(float x, float z) {
    // --- Bayangan Batu ---
    if (!isNight) {
        glDisable(GL_LIGHTING);
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        glColor4f(0.1f, 0.1f, 0.1f, 0.4f); // Dark gray, semi-transparent
        glPushMatrix();
        glTranslatef(0.0, -0.2, 0.0); // Posisi bayangan di dasar batu
        glScalef(1.0f, 0.05f, 1.0f); // Ukuran bayangan disesuaikan dengan batu
        glutSolidSphere(0.5, 20, 20); // Bola pipih
        glPopMatrix();
        glDisable(GL_BLEND);
        glEnable(GL_LIGHTING);
    }

    glPopMatrix();
}
```

3.10.5 RumahTipe1

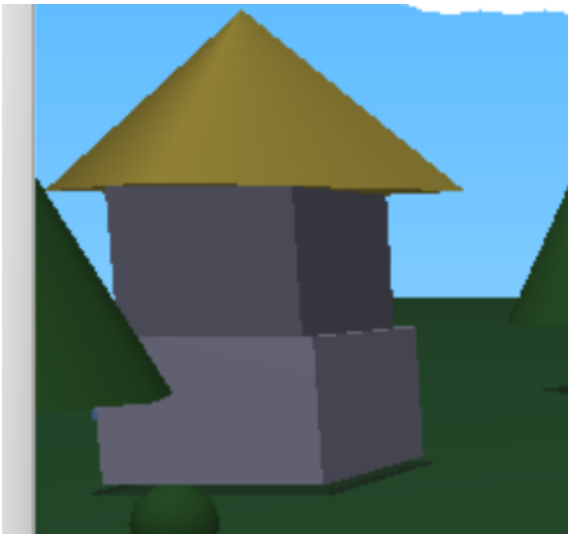


Bayangan rumah tipe 1 digambar dengan kubus pipih `glScalef(3.2f, 0.05f, 2.7f)` agar bentuknya menyerupai

dasar rumah, posisinya diatur di bawah rumah dengan `glTranslatef(0.0, -0.1, 0.0)`.

```
void buatRumahTipe1(float x, float z, float rotasi) {
    // --- Bayangan Rumah Tipe 1 ---
    if (!isNight) {
        glDisable(GL_LIGHTING);
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        glColor4f(0.1f, 0.1f, 0.1f, 0.4f); // Dark gray, semi-transparent
        glPushMatrix();
        glTranslatef(0.0, -0.1, 0.0); // Posisi bayangan di dasar rumah
        glScalef(3.2f, 0.05f, 2.7f); // Ukuran bayangan disesuaikan dengan alas rumah
        glutSolidCube(1.0); // Bentuk kubus pipih
        glPopMatrix();
        glDisable(GL_BLEND);
        glEnable(GL_LIGHTING);
    }

    glPopMatrix();
}
```

3.10.6 RumahTipe2



Bayangan rumah tipe 2 digambar dengan kubus pipih `glScalef(4.2f, 0.05f, 3.2f)`, posisinya diatur di dasar rumah dengan `glTranslatef(0.0, -0.1, 0.0)`.

```
void buatRumahTipe2(float x, float z, float rotasi) {
    // --- Bayangan Rumah Tipe 2 ---
    if (!isNight) {
        glDisable(GL_LIGHTING);
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        glColor4f(0.1f, 0.1f, 0.1f, 0.4f); // Dark gray, semi-transparent
        glPushMatrix();
        glTranslatef(0.0, -0.1, 0.0); // Posisi bayangan di dasar rumah
        glScalef(4.2f, 0.05f, 3.2f); // Ukuran bayangan disesuaikan dengan alas rumah
        glutSolidCube(1.0); // Bentuk kubus pipih
        glPopMatrix();
        glDisable(GL_BLEND);
        glEnable(GL_LIGHTING);
    }

    glPopMatrix();
}
```

BAB IV
SIMPULAN

Hasil perencanaan dan implementasi proyek Simulasi Game Obstacle Drift menunjukkan bahwa OpenGL efektif dalam membangun simulasi game 3D. OpenGL dapat merender objek, mengatur kamera dan proyeksi, serta menerapkan animasi dan interaksi secara *real-time*. Fungsi-fungsi dasar OpenGL berperan penting dalam membentuk dan mengelola objek dalam dunia 3D. Selain itu, objek-objek permainan, seperti mobil, rintangan, dan elemen latar berhasil dirancang secara interaktif dan memungkinkan visualisasi yang dinamis serta realistis di sepanjang lintasan permainan.

Logika permainan berhasil diimplementasikan dengan baik, termasuk sistem kontrol mobil, deteksi tabrakan menggunakan perhitungan jarak Euclidean, dan pengelolaan status permainan seperti "Game Over" atau "Finish". Hal ini menunjukkan integrasi prinsip grafika komputer dan pemrograman logika. Fitur tambahan, seperti *lighting*, proyeksi perspektif, *shadow*, tekstur prosedural, dan mode siang-malam juga menambah pengalaman visual pengguna. Variasi sudut pandang kamera dinamis (top view, third-person, side-top, dan first-person) semakin meningkatkan kenyamanan dan fleksibilitas bermain.