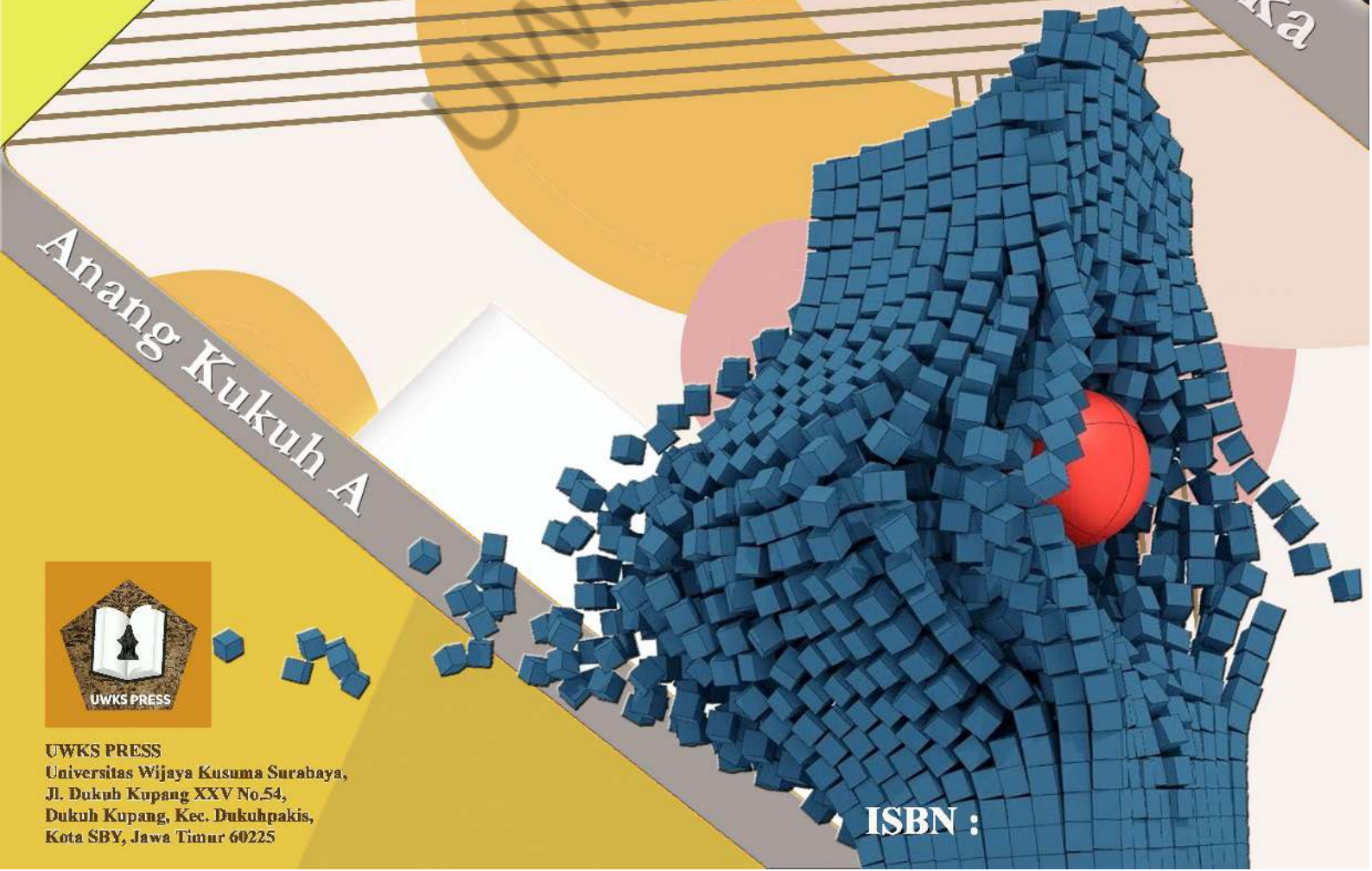
Buku Ajar

Transformasi dan Simulasi Fisika dalam Unity

Pemrograman Grafika

Anang Kukuh A



UWKS PRESS
Universitas Wijaya Kusuma Surabaya,
Jl. Dukuh Kupang XXV No.54,
Dukuh Kupang, Kec. Dukuhpakis,
Kota SBY, Jawa Timur 60225

ISBN :

Buku Ajar Pemrograman Grafika:Transformasi dan Simulasi Fisika dalam Unity

Dr. Ir. Anang Kukuh Adisusilo, ST., MT.

PENERBIT
UWKS PRESS

Buku Ajar Pemrograman Grafika:Transformasi dan Simulasi Fisika dalam Unity

ISBN
Ukuran buku 18 x 26 cm
210 hlm
Cetakan ke -1, Bulan Juli Tahun 2025

Penulis:
Dr. Ir. Anang Kukuh Adisusilo, ST., MT.

Editor:
Diyas Age Larasati, S.Pd., M.Pd.

Penerbit:
UWKS PRESS
Anggota IKAPI No.206/Anggota Luar Biasa/JTI/2018
Anggota APPTI No.002.071.1.12019

Jl. Dukuh Kupang XXV/54 Surabaya Jawa Timur 60225
Telp. (031) 5677577
Hp. 085745182452 / 081703875858
Email : uwkspress@gmail.com / uwkspress@uwks.ac.id

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas limpahan rahmat dan karunia-Nya sehingga buku ajar ini dapat disusun sebagai bagian dari penguatan pembelajaran dalam bidang pemrograman grafika interaktif, khususnya menggunakan Unity. Buku ajar ini dirancang untuk membantu mahasiswa memahami konsep-konsep dasar hingga lanjutan mengenai transformasi objek, sistem fisika, pergerakan kamera, hingga integrasi model, suara, dan sistem level dalam simulasi dan game edukatif.

Penyusunan buku ini bersifat praktikal dan aplikatif. Materi yang disajikan merupakan hasil integrasi dari modul-modul praktikum yang telah diuji di berbagai kelas, disusun ulang dengan pendekatan akademik sesuai standar DIKTI, serta diperkuat dengan literatur dan sitasi ilmiah terkini. Struktur buku ini dipecah ke dalam beberapa bab dengan tujuan agar mahasiswa dapat memahami materi secara bertahap, mengaplikasikannya dalam bentuk kode, dan mengembangkan kreativitas mereka dalam proyek akhir.

Penulis menyadari bahwa perkembangan teknologi digital yang sangat cepat menuntut adanya sumber belajar yang adaptif dan relevan. Oleh karena itu, buku ini juga dilengkapi dengan latihan mandiri, tugas kreativitas, serta studi kasus yang mendorong mahasiswa untuk berpikir kritis dan kolaboratif.

Akhir kata, semoga buku ajar ini dapat memberikan kontribusi nyata bagi proses pembelajaran dan menjadi referensi utama dalam pengembangan kompetensi di bidang pemrograman grafis interaktif. Penulis terbuka terhadap kritik dan saran demi perbaikan di masa mendatang.

Surabaya, Mei 2025
Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1 MOVEMENT OBJECT

1.1 Tujuan Pembelajaran

Bab ini dirancang untuk memberikan pemahaman menyeluruh kepada mahasiswa mengenai konsep dasar pergerakan objek (movement object) dalam konteks pemrograman grafika interaktif menggunakan Unity3D. Mahasiswa tidak hanya diajak untuk memahami teori transformasi dan sistem koordinat 3D, tetapi juga diharapkan mampu mengimplementasikan logika gerakan melalui skrip C# sederhana yang responsif terhadap input pengguna maupun animasi otomatis.

Tujuan pembelajaran ini mencakup kompetensi kognitif (pemahaman konsep), afektif (apresiasi terhadap simulasi visual), dan psikomotorik (penerapan praktis melalui praktik coding). Dengan mengikuti bab ini secara menyeluruh, mahasiswa diharapkan dapat:

a. Menjelaskan secara rinci prinsip-prinsip transformasi posisi, rotasi, dan skala dalam ruang 3D berbasis sistem koordinat kartesian.
b. Mengidentifikasi komponen transform dan alat bantu manipulasi objek dalam Unity Editor secara sistematis.
c. Menggunakan input dari perangkat keras (keyboard dan mouse) untuk menggerakkan objek di dalam ruang virtual.
d. Mengembangkan skrip dalam bahasa C# untuk mengontrol pergerakan, termasuk gerakan linear, rotasi, dan animasi otomatis.
e. Mendesain adegan virtual sederhana yang memadukan objek interaktif dan non-interaktif.
f. Mengevaluasi pengaruh desain gerak terhadap pengalaman pengguna dalam simulasi edukatif maupun hiburan.
g. Membandingkan pendekatan scripting langsung versus penggunaan built-in animation controller dalam Unity.
h. Merancang dan mengimplementasikan proyek mini berbasis gerakan objek dengan struktur kode yang rapi dan terdokumentasi.

Kompetensi ini sesuai dengan capaian pembelajaran lulusan (CPL) pada bidang Rekayasa Perangkat Lunak, khususnya terkait keterampilan teknis dan pengembangan aplikasi berbasis media interaktif serta game edukatif.

1.2 Landasan Teoretis: Grafika Komputer 3D dan Transformasi

Grafika komputer 3D merupakan cabang dari ilmu komputer yang berfokus pada representasi visual objek dalam ruang tiga dimensi. Dalam konteks ini, objek diproyeksikan ke dalam layar dua dimensi dengan teknik pemetaan tertentu agar dapat ditampilkan secara realistis. Representasi grafis tersebut memerlukan pemahaman mendalam tentang geometri, transformasi matematis, sistem koordinat, dan perspektif kamera.

1.2.1 Konsep Dasar Grafika Komputer 3D

Secara umum, grafika komputer 3D mencakup tiga proses utama:

1. **Modeling**: Pembuatan bentuk objek 3D, baik secara manual maupun melalui prosedural.
2. **Rendering**: Proses menghasilkan citra dua dimensi dari objek 3D menggunakan algoritma pencahayaan, tekstur, dan bayangan.
3. **Animation & Interaction**: Penggerakan dan manipulasi objek serta respon terhadap input pengguna.

Unity3D menyediakan platform komprehensif yang mencakup ketiga aspek tersebut, memungkinkan pengguna untuk membangun dunia virtual secara interaktif.

1.2.2 Transformasi dalam Grafika 3D

Transformasi adalah proses matematis untuk mengubah posisi, orientasi, dan ukuran objek di ruang 3D. Terdapat tiga jenis transformasi utama:

- **Translasi**: Memindahkan objek dari satu posisi ke posisi lain.
- **Rotasi**: Memutar objek terhadap sumbu tertentu (X, Y, Z).
- **Skalasi**: Mengubah ukuran objek.

Transformasi dalam Unity direpresentasikan melalui matriks 4x4 yang menggabungkan ketiga jenis transformasi ke dalam satu struktur yang disebut matriks transformasi homogen. Hal ini memudahkan penggabungan transformasi secara berurutan.

1.2.3 Komponen Transform dalam Unity

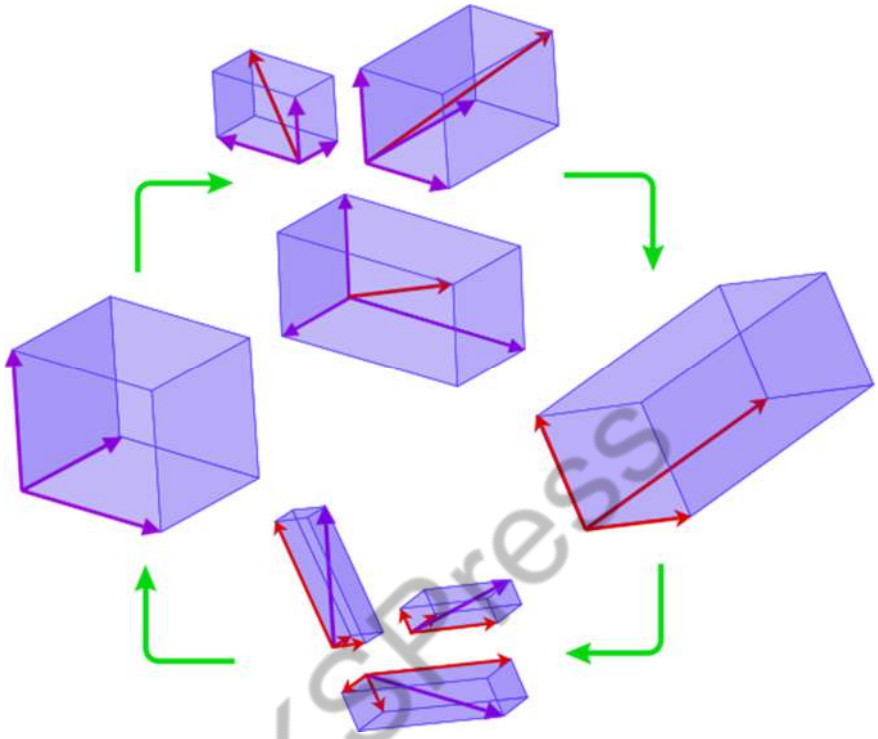
Unity secara otomatis menyematkan komponen **Transform** ke setiap objek (GameObject). Komponen ini berisi tiga properti utama:

- position: Posisi objek dalam koordinat dunia (world space).
- rotation: Rotasi objek dalam bentuk quaternion atau Euler angle.

- scale: Ukuran relatif objek pada masing-masing sumbu.

Manipulasi komponen ini secara langsung maupun melalui skrip memungkinkan pengguna menciptakan interaktivitas tinggi dalam sebuah adegan (scene).

1.2.4 Gambar: Struktur Transformasi Objek



Gambar 1. 1 Ilustrasi Transformasi Objek 3D

Wikimedia Commons (2023).

Gambar 1.1 menunjukkan proses transformasi basis (basis transformation) dalam ruang 3D, di mana sistem koordinat yang lama (dalam warna abu-abu) diubah menjadi sistem koordinat yang baru (warna biru dan merah). Ini adalah konsep fundamental dalam grafika komputer dan simulasi fisik di mana transformasi spasial terjadi terhadap objek atau ruang itu sendiri.

Komponen Visual:

- **Tiga vektor sumbu (basis vektor):**
    - e3, e2 dan e3 - basis vektor awal (warna abu-abu).
    - f1, f2 dan f3 - basis vektor hasil transformasi (warna biru/merah).
- **Transformasi basis** artinya kita melihat objek dari sistem koordinat baru, atau kita merotasi/skala objek terhadap basis baru ini.

Relevansi dengan Unity dan Transformasi:

- Dalam Unity, setiap objek memiliki sistem koordinat lokal (basis lokal), yang mengikuti transformasi dari induk (parent transform).
- Ketika kita memindahkan objek dalam basis lokal, kita sesungguhnya menggunakan **basis transformasi internal**, mirip dengan apa yang divisualisasikan pada gambar ini.
- Ini sangat penting dalam pengembangan animasi karakter, kamera dinamis, dan parenting objek.

Transformasi dari basis lama ke basis baru dilakukan melalui **matriks perubahan basis** yang diturunkan dari hubungan antara vektor-vektor baru dan lama.

1.2.5 Hubungan Transformasi dengan Matematika Linier

Proses transformasi didasarkan pada konsep vektor dan matriks dalam matematika linier. Sebagai contoh, translasi objek dalam sumbu X sejauh 3 unit dapat direpresentasikan dengan penambahan vektor posisi:

$$\vec{P_{baru}} = \vec{P_{lama}} + \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix} \quad (1)$$

Sedangkan rotasi terhadap sumbu Z dengan sudut θ\thetaθ dinyatakan dalam bentuk matriks sebagai berikut:

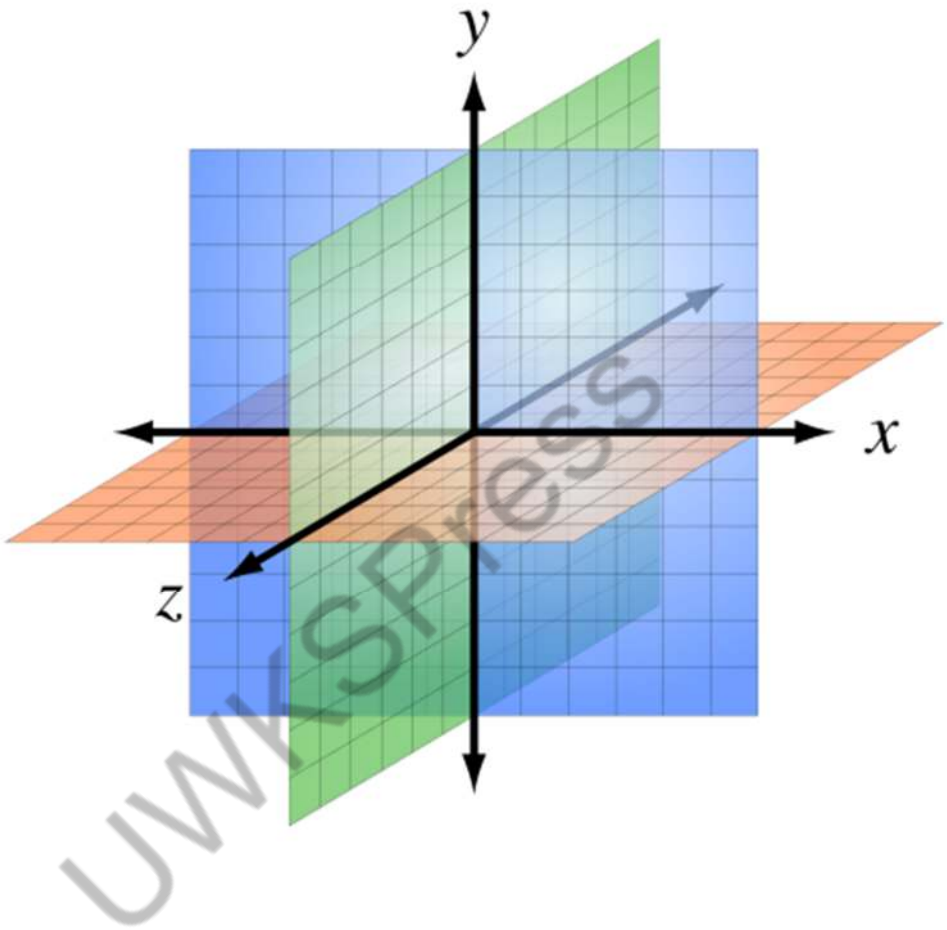
$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2)$$

Unity mengelola perhitungan ini secara otomatis melalui API dan engine internalnya, tetapi pemahaman dasar ini sangat penting bagi pengembang agar dapat mengoptimalkan kontrol logika gerakan.

1.3 Sistem Koordinat dan Sumbu Referensi

Dalam ruang tiga dimensi, sistem koordinat adalah fondasi utama dalam mendefinisikan posisi objek, arah gerakan, serta rotasi. Unity menggunakan sistem koordinat Kartesian

tiga dimensi berbasis tangan kanan (right-handed coordinate system), yang umum dipakai dalam grafika komputer. Penjelasan visual dapat dilihat melalui tautan Gambar berikut:



Gambar 1. 2 Coordinate System

Wikimedia Commons (2007).

Dari Gambar 1.2 ,Unity menggunakan sistem koordinat Kartesian tiga dimensi berbasis tangan kanan (right-handed coordinate system), yang umum dipakai dalam grafika komputer.

1.3.1 Sistem Koordinat Kartesian 3D

Sistem ini menggunakan tiga sumbu:

- **X (horizontal)**: Arah kanan (positif) dan kiri (negatif).

- **Y (vertikal)**: Arah atas (positif) dan bawah (negatif).
- **Z (kedalaman)**: Arah depan (positif) dan belakang (negatif).

1.3.2 Perbedaan Local Space dan World Space

- **World Space**: Sistem koordinat global dari seluruh dunia virtual dalam Unity.
- **Local Space**: Sistem koordinat relatif terhadap objek induk (parent) atau dirinya sendiri.

Contoh: Objek A berada di posisi (5, 0, 0) dalam World Space, tetapi jika ia adalah anak dari objek B yang berada di (10, 0, 0), maka dalam Local Space, posisi A adalah relatif.

Unity menyediakan metode seperti transform.position (World Space) dan transform.localPosition (Local Space) untuk mengakses posisi dalam konteks yang berbeda.

1.3.3 Sumbu Rotasi dan Arah Gerak

- **Rotasi terhadap X**: Mengubah orientasi vertikal (pitch).
- **Rotasi terhadap Y**: Mengubah arah horizontal (yaw).
- **Rotasi terhadap Z**: Mengubah kemiringan (roll).

Unity mendefinisikan rotasi menggunakan quaternion, tetapi juga menyediakan representasi dalam bentuk Euler angle (x, y, z) yang lebih mudah dipahami oleh pemula.

1.3.4 Sumbu dalam Praktik Unity

- Kamera dalam Unity biasanya diposisikan agar menghadap sumbu **Z negatif**.
- Player Controller sering bergerak pada sumbu **X-Z**.
- Sumbu **Y** biasanya digunakan untuk lompatan atau gravitasi.

1.3.5 Tabel Arah dan Fungsi Sumbu

Tabel 1.1 memberikan ringkasan tiga sumbu utama dalam sistem koordinat 3D Unity beserta fungsi umum dan arah positifnya. Ini penting karena semua pergerakan, transformasi, dan rotasi objek dilakukan berdasarkan acuan tiga sumbu tersebut.

Tabel 1. 1 Arah dan Fungsi Sumbu

Sumbu	Fungsi Umum	Gerakan Positif
X	Horizontal	Kanan
Y	Vertikal	Atas
Z	Kedalaman	Maju

Penjabaran Tabel 1.1:

- Sumbu X digunakan ketika objek bergerak ke arah kiri atau kanan dalam tampilan kamera atas atau perspektif. Misalnya, jika kita menekan tombol panah kanan, objek biasanya akan berpindah pada arah X positif.
- Sumbu Y berfungsi untuk pergerakan vertikal seperti lompatan, jatuh karena gravitasi, atau naik-turun elevator. Unity menggunakan nilai Y positif untuk pergerakan ke atas.
- Sumbu Z menunjukkan kedalaman dalam ruang 3D. Dalam tampilan default kamera Unity (kamera menghadap ke sumbu Z negatif), gerakan ke arah Z positif berarti menjauh dari kamera (maju).

Contoh implementasi dalam Game:

- Saat pemain bergerak maju dalam game 3D orang ketiga, biasanya gerakan terjadi di sepanjang sumbu Z positif.
- Untuk lompat, karakter diberi dorongan di sumbu Y positif.
- Jika karakter berputar menghadap kiri atau kanan, rotasi sering dilakukan terhadap sumbu Y.

1.3.6 Kesalahan Umum Mahasiswa

- Salah memahami arah Z sebagai naik/turun.
- Bingung antara rotasi global dan lokal.
- Menggabungkan transformasi tanpa memahami urutan rotasi (gimbal lock).

1.3.7 Relevansi terhadap Pergerakan Objek

Pemahaman sistem koordinat dan orientasi sangat penting untuk mengontrol arah gerak objek. Salah penempatan sumbu atau salah rotasi dapat menyebabkan objek bergerak ke arah yang tidak diinginkan.

Sebagaimana dijelaskan oleh Adisusilo (2022), "kegagalan dalam memahami orientasi sumbu akan mempersulit mahasiswa dalam membangun logika kontrol karakter atau kamera secara intuitif."

1.4 Pengenalan Unity3D sebagai Media Interaktif

Unity3D adalah sebuah platform pengembangan game dan simulasi real-time berbasis mesin grafika yang memungkinkan pengembang untuk merancang, membangun, dan menguji aplikasi 2D dan 3D secara interaktif. Dikembangkan oleh Unity Technologies sejak tahun 2005, Unity telah berkembang menjadi salah satu platform game engine yang paling populer dan luas digunakan di dunia.\

1.4.1 Unity sebagai Alat Edukasi dan Simulasi

Unity bukan hanya sekadar alat pengembangan game komersial, tetapi telah banyak digunakan dalam konteks pendidikan, arsitektur, kedokteran, pelatihan militer, dan industri kreatif lainnya. Dalam konteks pendidikan tinggi, Unity berperan sebagai:

- Media pembelajaran interaktif.
- Platform simulasi berbasis visual.
- Sarana praktikum untuk pemrograman grafika dan desain interaksi.

Contoh pemanfaatan Unity dalam bidang edukasi:

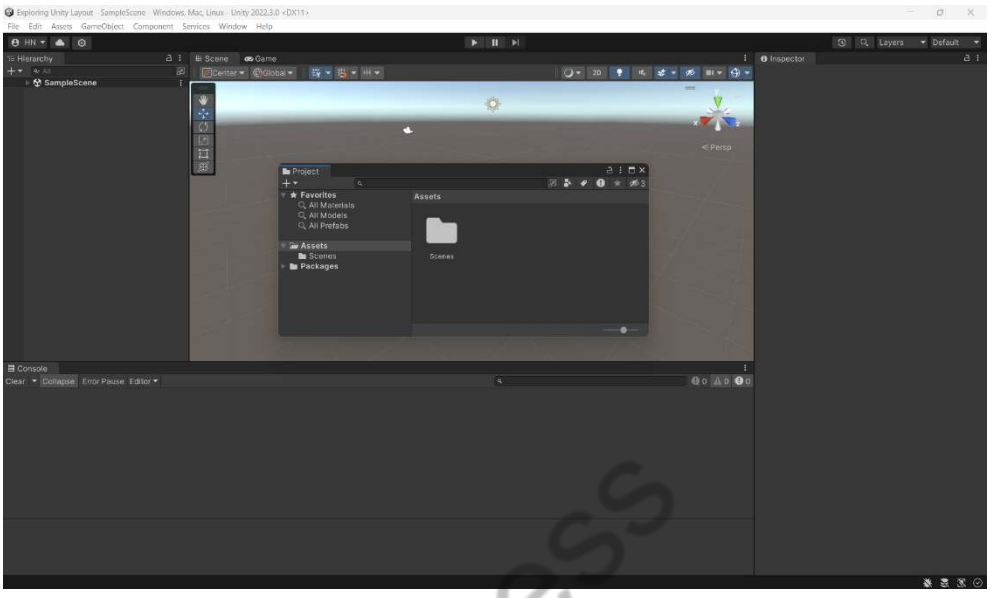
- Simulasi laboratorium virtual untuk pelajaran fisika dan kimia.
- Representasi sejarah interaktif dalam bentuk museum virtual.
- Game edukasi berbasis karakter lokal dan budaya (Adisusilo, 2022).

1.4.2 Antarmuka Unity Editor

Unity memiliki antarmuka editor yang intuitif dan fleksibel, terdiri atas beberapa panel penting, seperti Gambar 1.3:

- **Scene View**: Tempat pengguna membangun dan mengatur dunia virtual.
- **Game View**: Tampilan hasil akhir ketika proyek dijalankan.
- **Hierarchy**: Daftar objek dalam scene.

- **Inspector**: Panel pengaturan properti dari objek yang dipilih.
- **Project**: Tempat penyimpanan semua asset dan skrip.



Gambar 1. 3 Antar Muka Sistem Unity.

Unity Tutorial 2022.3 (2022).

1.4.3 Komponen Penting dalam Unity

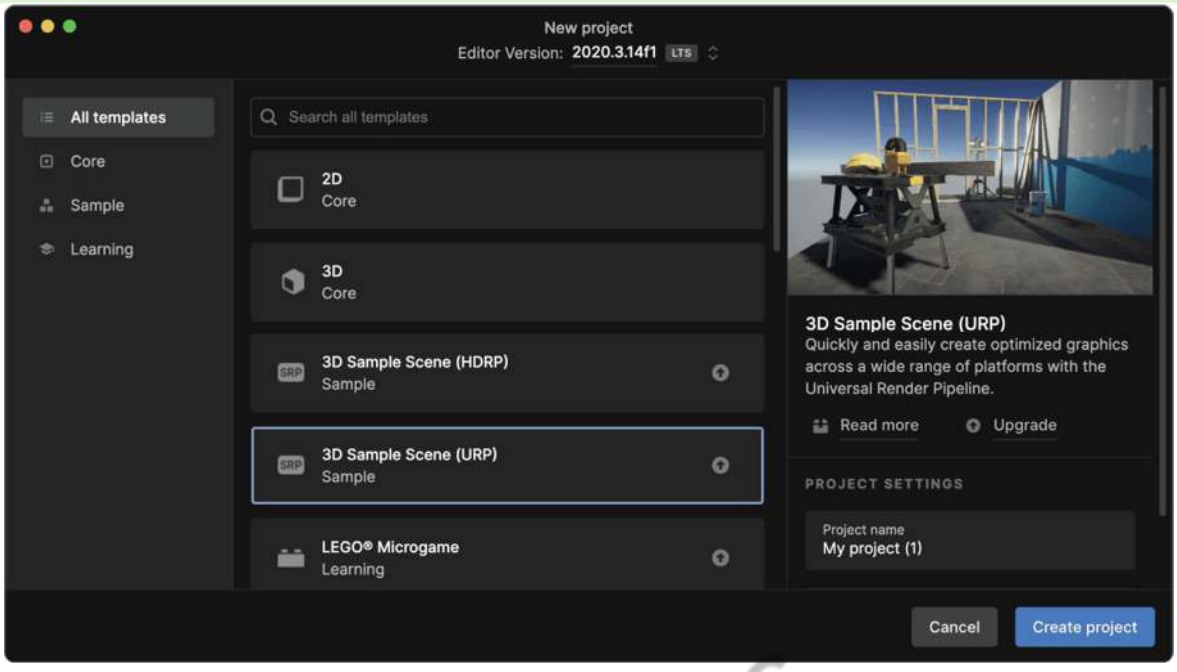
Beberapa komponen penting yang selalu melekat pada GameObject dalam Unity adalah:

- **Transform**: Mengatur posisi, rotasi, dan skala objek.
- **Renderer**: Menampilkan objek secara visual.
- **Collider**: Digunakan untuk mendeteksi tabrakan.
- **Script**: File kode (C#) untuk mengatur logika dan interaksi.

1.4.4 Unity Hub dan Manajemen Proyek

Unity Hub seperti terlihat pada Gambar 1.4 , adalah launcher resmi yang digunakan untuk:

- Mengelola berbagai versi Unity.
- Membuat, mengatur, dan membuka proyek.
- Menghubungkan dengan Unity Services (Cloud Build, Analytics, dll).



Gambar 1. 4 Unity Hub Manajemen Proyek.

Unity Tutorial 2022.3 (2022).

1.4.5 Dukungan Cross-Platform

Salah satu keunggulan Unity adalah kemampuannya membangun aplikasi lintas platform dari satu basis kode. Unity dapat melakukan build ke berbagai sistem operasi dan perangkat, antara lain:

- Windows, macOS, Linux
- Android dan iOS
- WebGL (aplikasi berbasis browser)
- Konsol seperti PlayStation, Xbox, dan Nintendo Switch

1.4.6 Keunggulan Unity dalam Pendidikan

Beberapa alasan Unity sangat cocok digunakan di lingkungan akademik:

- Gratis untuk institusi pendidikan dan individu.
- Dokumentasi resmi dan komunitas luas.
- Dukungan scripting berbasis C#, bahasa yang umum dipakai dalam pengembangan sistem.
- Asset Store menyediakan ribuan objek gratis dan berbayar.

"Unity dapat mendorong kreativitas mahasiswa dalam mengembangkan media interaktif edukatif berbasis lokalitas dan narasi budaya." (Adisusilo, 2022)

1.4.7 Contoh Implementasi Unity dalam Kelas

Berikut contoh penerapan unity dalam kelas pemebelajaran, seperti Tabel 1.2.

Tabel 1. 2 Contoh Implementasi Unity dalam Kelas

Mata Kuliah	**Implementasi Unity**
Pemrograman Visual	Simulasi interaksi objek 3D
Desain Game Edukatif	Pengembangan game pembelajaran SD
Animasi dan Motion Capture	Integrasi karakter animasi dan kamera virtual
Teknologi Multimedia Interaktif	Museum virtual atau pameran interaktif

1.4.8 Tantangan dan Solusi

- **Tantangan**: Kurva belajar awal cukup curam bagi pemula.
- **Solusi**: Modul praktikum terstruktur, pembelajaran berbasis proyek (project-based learning), serta komunitas lokal kampus.

Unity3D merupakan salah satu alat paling komprehensif untuk membangun media interaktif yang melibatkan elemen visual, pemrograman, dan interaktivitas pengguna. Di lingkungan pendidikan tinggi, Unity tidak hanya membantu mahasiswa memahami konsep grafika komputer tetapi juga meningkatkan kemampuan berpikir kreatif, kolaboratif, dan aplikatif.

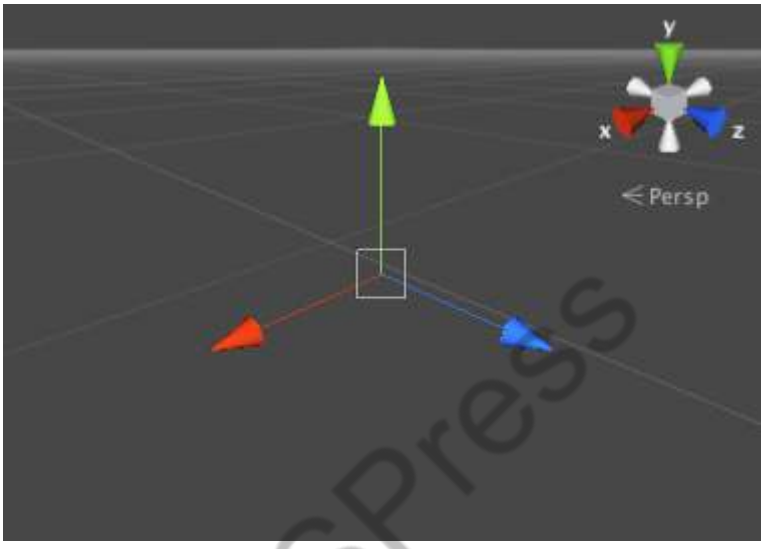
1.5 Komponen Transform dan Scene View Tools

Salah satu kekuatan utama Unity sebagai game engine adalah kemampuannya untuk memvisualisasikan dan memanipulasi objek secara langsung di dalam editor melalui apa yang disebut sebagai **Transform Tools**. Transformasi objek melibatkan tiga elemen utama, yaitu **posisi (position)**, **rotasi (rotation)**, dan **skala (scale)**. Unity menyederhanakan pengolahan elemen-elemen tersebut dalam satu komponen: **Transform**.

1.5.1 Komponen Transform

Setiap objek dalam Unity secara otomatis memiliki komponen Transform seperti Gambar 1.5. Komponen ini menyimpan data spasial dari objek tersebut, yakni:

- **Position**: Lokasi objek dalam koordinat dunia (world) atau lokal (local).
- **Rotation**: Arah orientasi objek.
- **Scale**: Ukuran objek sepanjang masing-masing sumbu (X, Y, Z).



Gambar 1. 5 Unity Komponen Transform

Unity Tutorial 2022.3 (2022).

1.5.2 Alat Transform di Scene View

Unity menyediakan beberapa alat bantu visual (gizmo) untuk memanipulasi transformasi secara langsung:

- **Move Tool (W)**: Menggerakkan objek.
- **Rotate Tool (E)**: Memutar objek.
- **Scale Tool (R)**: Mengubah ukuran objek.
- **Rect Tool (T)**: Digunakan terutama untuk UI (User Interface).
- **Transform Tool (Y)**: Gabungan dari alat-alat di atas.

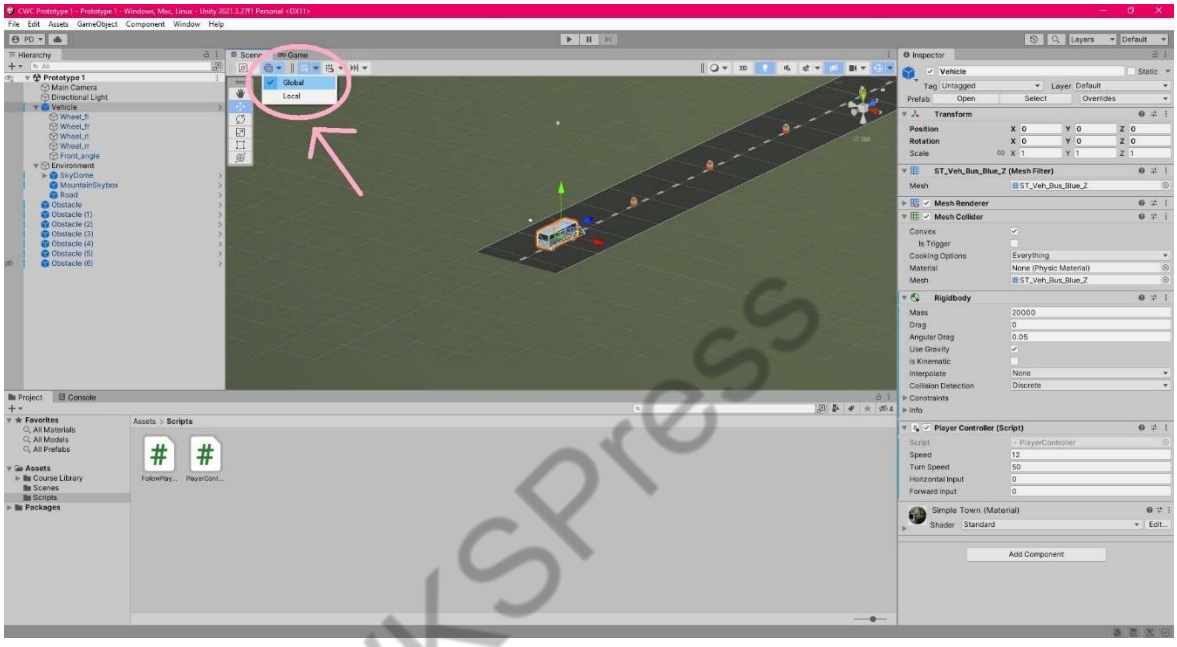
Setiap alat ini memiliki handle warna merah (X), hijau (Y), dan biru (Z) untuk merepresentasikan arah transformasi.

1.5.3 Perbedaan Local dan Global Transform

Transformasi dapat dilakukan dalam dua konteks:

- **Global (World)**: Transformasi mengacu pada sumbu dunia.
- **Local (Self)**: Transformasi mengacu pada sumbu objek itu sendiri.

Misalnya, memutar objek akan memengaruhi arah sumbu lokalnya. Jika kemudian objek digerakkan dalam mode local, pergerakan mengikuti arah sumbu baru tersebut.



Gambar 1. 6 Mode Global dan Local.

Game Development (2019).

.

Gambar 1.6 menunjukkan ikon toggle di Scene View untuk *switch* antara mode **Global** dan **Local**. Saat di Global, gizmo sumbu objek tetap sejajar sumbu dunia; di Local, gizmo mengikuti orientasi objek.

1.5.4 Snapping dan Precision Tools

Untuk presisi tinggi dalam pemodelan dan penempatan objek, Unity menyediakan fitur:

- **Grid snapping (Ctrl + klik)**: Menempelkan objek ke kisi.
- **Increment Snap**: Mengatur jarak gerak, rotasi, dan skala minimum.
- **Align Tools**: Menyelaraskan posisi relatif terhadap objek lain.

Fitur-fitur ini berguna untuk membuat level, UI, atau struktur modular dengan keteraturan spasial yang baik.

1.5.5 Gizmos dan Handles

Gizmo adalah simbol visual di Scene View untuk membantu proses editing:

- Panah (Move), lingkaran (Rotate), kubus (Scale).
- Warna mewakili sumbu: X = merah, Y = hijau, Z = biru.

Selain itu, pengguna dapat menambahkan Gizmo kustom melalui skrip OnDrawGizmos untuk memvisualisasikan batas deteksi, area trigger, dll.

Gambar 1. 7 Transform Gizmo unity 3D

Doc Unity 6.1 (2022)

Gambar 1.7 menampilkan tiga ujung gizmo berwarna (merah, hijau, biru) yang mewakili sumbu X, Y, dan Z. Juga terlihat kubus kecil di tengah, yang bisa digunakan untuk memindah objek dalam satu bidang (dua sumbu sekaligus), secara detail fungsinya sebagai berikut :

a. **Panah sumbu**:

- Merah = X, hijau = Y, biru = Z.
- Menyeret panah akan memindahkan objek sepanjang sumbu tersebut.

b. **Kotak tengah (plane handles)**:

- Memungkinkan pergerakan simultan di dua sumbu (fungsi plane movement).
- Misalnya, menarik bidang XY memindahkan objek secara horizontal dan vertikal sekaligus dan dikutip resmi .

c. **Mode transform**:

- **W**: Move Tool
- **E**: Rotate Tool
- **R**: Scale Tool
- **T**: Rect Transform (UI)
- **Y**: Universal Transform.

d. **Mode Pivot / Center & Global / Local**

- **Pivot**: Gizmo muncul pada titik pivot objek.
- **Center**: Pada titik tengah bounding box.
- **Local**: Sumbu transform mengikuti orientasi objek.
- **Global**: Sumbu selalu merujuk sumbu dunia.

e. **Snap Tools**:

- Tekan Ctrl + drag untuk snapping unit di grid.
- Tekan V saat drag untuk vertex snapping

1.5.6 Shortcut Transformasi

Tabel 1. 3 Shorcut Transformasi

Shortcut	Fungsi
W	Move Tool
E	Rotate Tool
R	Scale Tool
T	Rect Tool
Q	View Tool
Ctrl + D	Duplicate

Tabel 1.3 menunjukkan *keyboard shortcut* mempercepat proses editing di Unity dan mendukung efisiensi saat membangun dunia virtual.

1.5.7 Contoh Studi Kasus

Studi Kasus: Buatlah sebuah adegan dengan:

- Sebuah tanah (Plane), dua bangunan (Cube), dan satu karakter (Capsule).
- Gunakan Move Tool untuk menempatkan bangunan sejajar.
- Atur rotasi karakter agar menghadap bangunan.
- Ubah skala salah satu Cube menjadi menara.

Latihan ini membantu mahasiswa memahami transformasi spasial dasar dalam dunia virtual 3D.

Komponen Transform dan alat bantu Scene View menjadi dasar dalam seluruh proses desain visual di Unity. Pemahaman mendalam terhadap alat ini akan membantu mahasiswa menciptakan dunia virtual yang tertata, responsif, dan sesuai dengan prinsip spasial dalam grafika komputer.

1.6 Dasar Input Interaktif (Keyboard dan Mouse)

Dalam pengembangan aplikasi interaktif berbasis Unity, penggunaan input dari perangkat seperti **keyboard** dan **mouse** menjadi sangat krusial. Input ini memungkinkan pengguna untuk mengendalikan objek, berinteraksi dengan lingkungan, dan memicu berbagai aksi dalam dunia virtual.

1.6.1 Sistem Input di Unity

Unity menyediakan sistem input melalui kelas Input dalam namespace UnityEngine. Kelas ini memungkinkan deteksi berbagai aksi dari pengguna seperti:

- Tekanan tombol (keyboard).
- Gerakan dan klik mouse.
- Sentuhan (touch input) pada perangkat mobile.

Contoh sederhana:

```
if (Input.GetKey(KeyCode.W))
{
    transform.Translate(Vector3.forward * speed * Time.deltaTime);
}
```

Kode di atas akan membuat objek bergerak ke depan saat tombol **W** ditekan.

1.6.2 Fungsi-Fungsi Utama Input

Tabel 1. 4 Fungsi input utama dalam Unity

Fungsi Unity	**Deskripsi**
Input.GetKey()	Mengecek apakah tombol ditekan terus-menerus
Input.GetKeyDown()	Mengecek apakah tombol baru saja ditekan sekali
Input.GetKeyUp()	Mengecek apakah tombol baru saja dilepas
Input.GetMouseButton()	Mengecek klik kiri (0), kanan (1), tengah (2)
Input.mousePosition	Mengambil posisi kursor di layar (pixel)

1.6.3 Menggunakan Keyboard untuk Kontrol Gerakan

Pada umumnya, game 3D menggunakan tombol **WASD** atau **panah arah** untuk navigasi. Berikut implementasi dasarnya:

```
void Update()
{
    float x = Input.GetAxis("Horizontal");
    float z = Input.GetAxis("Vertical");
    transform.Translate(x * speed * Time.deltaTime, 0, z * speed * Time.deltaTime);
}
```

Kode di atas membuat objek dapat digerakkan dua arah menggunakan tombol panah atau WASD.

1.6.4 Deteksi Klik dan Interaksi dengan Mouse

Unity juga memungkinkan interaksi berbasis mouse, seperti klik objek. Umumnya digunakan bersama sistem **Raycast**:

```
void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;
        if (Physics.Raycast(ray, out hit))
        {
            Debug.Log("Kena objek: " + hit.transform.name);
        }
    }
}
```


1.6.5 Penggunaan Input Manager

Unity memiliki sistem konfigurasi input bernama **Input Manager**, diakses melalui: Edit > Project Settings > Input Manager

Di dalamnya, Anda bisa:

- Menentukan nama input (horizontal, vertical, fire1, dll).
- Mengatur kombinasi tombol, sensitivitas, dan alternatif kontrol.

1.6.6 Studi Kasus Mini

Kasus: Buatlah sistem kontrol karakter 3D:

- Tombol W/S untuk maju mundur.
- Tombol A/D untuk rotasi kiri kanan.
- Klik mouse kiri untuk menyapa objek NPC (non-player character).

Terapkan skrip sederhana dan uji hasilnya di Game View.

Sistem input dalam Unity adalah fondasi dari semua interaksi pengguna dengan dunia virtual. Pemahaman terhadap keyboard, mouse, serta mekanisme raycast dan konfigurasi input akan memperkuat keterampilan mahasiswa dalam menciptakan aplikasi interaktif yang dinamis dan

1.7 Studi Kasus Praktik: Gerakan Dasar Objek (Posisi, Rotasi, Skala)

Dalam bagian ini, mahasiswa akan mempraktikkan teori-teori sebelumnya dengan membuat contoh nyata penggerakan objek dalam dunia 3D Unity. Studi kasus ini bertujuan memberikan pemahaman langsung tentang pergerakan posisi, rotasi, dan skala berdasarkan input pengguna serta pengaturan transformasi.

1.7.1 Tujuan Praktik

- Mahasiswa memahami implementasi transformasi objek.
- Mahasiswa mampu menerapkan skrip dasar untuk gerakan.
- Mahasiswa mengenal kontrol berbasis input secara langsung.

1.7.2 Deskripsi Proyek

Mahasiswa diminta untuk membuat adegan (scene) interaktif dengan skenario berikut:

- Sebuah karakter (menggunakan prefab Capsule).
- Medan permainan (Plane).
- Dua buah rintangan (Cube) sebagai penghalang.
- Sebuah target (Sphere) yang dapat didekati karakter.

1.7.3 Langkah-Langkah Praktik

1. **Membuat Scene Baru**
    - Buka Unity → File → New Scene.
    - Tambahkan objek Plane, Capsule, Cube, dan Sphere ke scene.
2. **Menambahkan Skrip Gerakan** Buat skrip PlayerController.cs:

```
public class PlayerController : MonoBehaviour
{
    public float speed = 5f;

    void Update()
    {
        float moveX = Input.GetAxis("Horizontal");
        float moveZ = Input.GetAxis("Vertical");
        Vector3 move = new Vector3(moveX, 0, moveZ);
        transform.Translate(move * speed * Time.deltaTime);
    }
}
```

3. **Menambahkan Collider dan Rigidbody**
    - Tambahkan BoxCollider pada Cube.
    - Tambahkan SphereCollider pada Sphere.
    - Tambahkan Rigidbody pada Capsule agar dapat terpengaruh fisika.

4. **Menambahkan Kamera Mengikuti Pemain**

```
public class CameraFollow : MonoBehaviour
{
    public Transform target;
    public Vector3 offset;

    void LateUpdate()
    {
        transform.position = target.position + offset;
    }
```

```
}
```

5. **Menambahkan Interaksi Sederhana**

```
void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Target"))
    {
        Debug.Log("Target tercapai!");
    }
}
```

1.7.4 Visualisasi Scene

Mahasiswa dapat merancang susunan objek seperti berikut:

- Plane sebagai lantai.
- Cube di kiri dan kanan sebagai dinding penghalang.
- Capsule di tengah sebagai pemain.
- Sphere di ujung sebagai tujuan.

1.7.5 Penilaian dan Evaluasi

Tabel 1. 5 Penilaian dan Evaluasi

Aspek yang Dinilai	Bobot
Fungsi gerakan berjalan	30%
Implementasi collider	20%
Interaksi target	20%
Setup kamera dan tata letak	20%
Dokumentasi kode	10%

Melalui praktik ini, mahasiswa memperoleh pemahaman nyata tentang penerapan transformasi dan input dalam Unity. Studi kasus ini akan menjadi dasar kuat dalam membangun interaksi kompleks seperti navigasi pemain, AI, dan simulasi lingkungan.

1.8 Studi Kasus Lanjutan: Gerakan Otomatis dan Variatif

Studi kasus lanjutan ini dirancang untuk melatih mahasiswa dalam merancang gerakan objek yang tidak hanya dikendalikan oleh input pengguna secara langsung, tetapi juga dapat berlangsung otomatis atau memiliki pola variasi tertentu. Hal ini umum digunakan pada karakter non-pemain (NPC), elemen efek visual, maupun sistem simulasi.

1.8.1 Tujuan Pembelajaran

- Mampu menerapkan logika gerakan otomatis dan terjadwal.
- Mampu membuat pola gerakan berdasarkan waktu atau kondisi tertentu.
- Mengintegrasikan transformasi dan scripting untuk gerakan mandiri.
- Mengenal prinsip dasar animator dan interpolasi.

1.8.2 Skenario Studi Kasus

Mahasiswa akan membuat:

- Satu objek Sphere yang bergerak bolak-balik secara otomatis.
- Satu objek Cube yang berputar terus-menerus.
- Satu objek NPC yang berpatroli antara dua titik.

1.8.3 Implementasi Kode Dasar

1. Gerakan Bolak-balik (Ping-Pong Movement)

```
public class PingPongMovement : MonoBehaviour
{
    public float amplitude = 3f;
    public float speed = 2f;

    void Update()
    {
        float z = Mathf.PingPong(Time.time * speed, amplitude);
        transform.position = new Vector3(transform.position.x, transform.position.y, z);
```

```
    }
}
```

2. Rotasi Berkelanjutan

```
public class AutoRotate : MonoBehaviour
{
    public Vector3 rotateSpeed = new Vector3(0, 100, 0);

    void Update()
    {
        transform.Rotate(rotateSpeed * Time.deltaTime);
    }
}
```

3. **NPC Patroli antara Dua Titik**

```
public class Patrol : MonoBehaviour
{
    public Transform pointA, pointB;
    public float speed = 2f;
    private Transform target;

    void Start()
    {
        target = pointA;
    }

    void Update()
    {
```

```
        transform.position = Vector3.MoveTowards(transform.position, target.position, speed *
Time.deltaTime);


        if (Vector3.Distance(transform.position, target.position) < 0.1f)
        {
            target = (target == pointA) ? pointB : pointA;
        }
    }
}
```

1.8.4 Visualisasi Scene dan Penempatan

Susunan objek:

- Sphere = objek horizontal bolak-balik.
- Cube = objek rotasi dekoratif.
- NPC = karakter bergerak antara titik A dan B.

1.8.5 Penilaian dan Evaluasi

Tabel 1. 6 Penilaian dan Evaluasi Lanjutan

Aspek Evaluasi	**Bobot**
Gerakan otomatis berjalan	30%
Implementasi skrip yang efisien	25%
Keteraturan scene dan prefab	15%
Kreativitas variasi gerakan	20%
Dokumentasi kode	10%

Gerakan otomatis dan variasi adalah salah satu dasar penting untuk membuat lingkungan interaktif yang hidup. Melalui studi kasus ini, mahasiswa mendapatkan bekal untuk mengembangkan logika AI sederhana dan sistem gerakan non-linear yang digunakan dalam banyak proyek simulasi dan game.

1.9 Simulasi Interaktif Mini Project

Mini project ini merupakan tahap akhir dari Bab 1, yang mengintegrasikan semua konsep transformasi, input, dan kontrol interaktif dalam sebuah simulasi kecil. Tujuan utamanya adalah memberikan pengalaman menyeluruh dalam merancang dan membangun lingkungan interaktif dengan Unity.

1.9.1 Tujuan Proyek

- Mahasiswa mampu menggabungkan semua konsep transformasi dan input interaktif.
- Mahasiswa dapat merancang alur kontrol objek dalam simulasi kompleks.
- Mahasiswa terampil dalam menyusun scene terstruktur dan interaktif.
- Mahasiswa dapat menerapkan pengujian (testing) sederhana pada interaksi objek.

1.9.2 Deskripsi Mini Project

Buatlah adegan simulasi di mana:

- Karakter utama bergerak mengikuti input WASD.
- Objek-objek tertentu memiliki animasi rotasi otomatis.
- NPC bergerak dengan pola patrol dan dapat dihentikan saat tabrakan.
- Terdapat zona interaksi yang memicu respons visual atau suara.
- UI sederhana menunjukkan skor atau status.

1.9.3 Struktur Objek dan Scene

Tabel 1. 7 Struktur Objek dan Scene

Objek	**Fungsi**
Capsule	Karakter utama
Cube	Rintangan tetap
Sphere	Target dengan trigger suara
NPC (Capsule)	Bergerak antara dua titik
Kamera	Mengikuti karakter

UI Canvas	Menampilkan skor atau pesan
AudioSource	Memberi umpan balik suara

Scene dapat dibuat modular untuk memudahkan pengujian komponen.

1.9.4 Komponen dan Scripting

Komponen utama yang perlu digunakan:

- Rigidbody & Collider
- Input Axis Horizontal & Vertical
- Raycast dan Trigger
- Script untuk rotasi otomatis & NPC patrol
- AudioSource dan UI Text

Contoh skrip interaksi:

```
void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Target"))
    {
        score++;
        audioSource.Play();
        uiText.text = "Target Dicapai: " + score;
    }
}
```


1.9.5 Tahapan Pengerjaan

1. Buat sketsa layout scene (tangan/manual).
2. Bangun scene di Unity menggunakan prefab dasar.
3. Tambahkan logika input karakter.
4. Buat script gerakan otomatis (Cube/Sphere/NPC).
5. Tambahkan UI dan interaksi (skor, suara, indikator).

6. Uji dan dokumentasikan interaksi.

1.9.6 Visualisasi Akhir

Scene akhir minimal memuat:

- Area terbuka dengan lantai (Plane), batas (Cube).
- Gerakan otomatis dari NPC dan Cube.
- Interaksi player dengan Sphere sebagai target.
- UI untuk skor atau indikator lain.

1.9.7 Rubrik Penilaian Mini Project

Tabel 1. 8 Rubrik Penilaian Mini Project

Kriteria Evaluasi	Bobot
Fungsi gerakan & input kontrol	20%
Interaksi dengan objek & trigger	20%
Desain scene & estetika visual	20%
Penggunaan UI dan feedback	15%
Kreativitas fitur tambahan	15%
Dokumentasi & struktur proyek	10%

1.9.8 Tantangan dan Refleksi

Pertanyaan refleksi:

- Apa bagian tersulit dalam pengembangan mini project ini?
- Apa bagian yang paling menyenangkan atau menantang?
- Jika diberi waktu tambahan, apa yang ingin Anda tingkatkan?
- Apa pelajaran teknis dan non-teknis yang Anda pelajari?

Mini project ini merupakan integrasi penuh dari pembelajaran Bab 1. Mahasiswa tidak hanya belajar teknis pengembangan interaktif di Unity, tetapi juga menerapkan prinsip desain sistem, logika kontrol, interaksi pengguna, serta dokumentasi kode yang rapi dan kreatif.

1.10 Penerapan Movement Object dalam Game dan Simulasi Nyata

Subbab ini mengajak mahasiswa untuk memahami bagaimana konsep movement object yang telah dipelajari sebelumnya diaplikasikan secara nyata dalam pengembangan game maupun simulasi interaktif. Penerapan ini meliputi aspek teknis, fungsional, dan estetika dalam konteks proyek dunia nyata, baik di industri hiburan, edukasi, maupun penelitian.

1.10.1 Tujuan Pembelajaran

- Mahasiswa mampu mengidentifikasi penggunaan sistem movement dalam game nyata.
- Mahasiswa dapat menjelaskan prinsip teknis yang digunakan pada movement object.
- Mahasiswa dapat membandingkan teknik movement di Unity dengan proyek profesional.

1.10.2 Studi Kasus 1: Game Platformer 2.5D

Pada game seperti *Hollow Knight* atau *Ori and the Blind Forest*, karakter utama bergerak pada jalur dua dimensi namun dalam lingkungan 3D. Gerakan karakter membutuhkan:

- Deteksi tabrakan akurat dengan collider.
- Sistem lompat (jump system) berbasis input dan logika gravitasi.
- Transisi halus antar animasi gerak.

Contoh skrip gerakan karakter platformer:

```
void Update()
{
    float move = Input.GetAxis("Horizontal");
    rb.velocity = new Vector2(move * speed, rb.velocity.y);

    if (Input.GetButtonDown("Jump") && isGrounded)
    {
        rb.AddForce(new Vector2(0f, jumpForce));
    }
```

}

1.10.3 Studi Kasus 2: Simulasi Transportasi dan Lalu Lintas

Dalam simulasi urban seperti *City Skylines* atau sistem pelatihan lalu lintas, movement object berperan dalam:

- Kendali jalur otomatis kendaraan.
- Logika antrian dan navigasi (pathfinding).
- Pemrosesan tabrakan dan kondisi macet.

Unity mendukung sistem pathfinding melalui NavMesh dan AI Navigation:

- Objek agent diberi tujuan (destination).
- Sistem menghitung jalur optimal dan menavigasi objek.

1.10.4 Studi Kasus 3: VR/AR dan Pengendalian Gesture

Dalam aplikasi seperti *VRChat*, *Google Tilt Brush*, atau sistem pelatihan medis VR, movement object digunakan untuk:

- Gerakan responsif berdasarkan input gesture.
- Translasi dan rotasi objek dalam lingkungan XR.
- Interaksi langsung melalui tangan/kontroler.

1.10.5 Penerapan Movement dalam Dunia Nyata

Tabel 1. 9 Penerapan Movement dalam Dunia Nyata

Bidang	**Contoh Penerapan**
Game & Hiburan	Kontrol karakter, NPC patrol, efek sinematik
Edukasi	Simulasi laboratorium, animasi sains
Medis	Simulasi pembedahan dan diagnosis interaktif
Teknik & Industri	Training simulasi mesin, perakitan virtual
Arsitektur & Visualisasi	Virtual tour, walkthrough proyek

1.10.6 Kode Movement Modular untuk Produksi

Skrip movement pada proyek nyata biasanya dibagi dalam beberapa modul:

- Input → MovementController.cs
- Kamera → CameraFollow.cs
- Physics → RigidbodyHandler.cs
- Animation → AnimatorHandler.cs

Keuntungan:

- Mudah diuji (testing & debugging).
- Fleksibel untuk pengembangan skala besar.

1.10.7 Evaluasi dan Refleksi

- Mahasiswa diminta menganalisis game/simulasi pilihan dan mendeskripsikan sistem movement-nya.
- Diskusikan kelebihan/kekurangan pendekatan movement tersebut.
- Usulkan perbaikan atau fitur tambahan berbasis konsep Unity.

Movement object adalah komponen fundamental dalam dunia game dan simulasi. Melalui pemahaman prinsip teknis dan penerapan lintas industri, mahasiswa diharapkan dapat membangun sistem interaktif yang adaptif, realistis, dan efisien dalam proyek nyata di masa depan.

1.11 Visualisasi Gerakan: Gambar, Diagram, dan Tabel Komparatif

Visualisasi adalah bagian penting dalam memahami dan menyampaikan konsep gerakan objek secara intuitif. Dalam Unity dan pengembangan grafika komputer secara umum, penggunaan gambar, diagram alur, serta tabel komparatif dapat memperjelas transformasi, arah gerak, dan variasi sistem kontrol.

1.11.1 Fungsi Visualisasi dalam Proses Pembelajaran

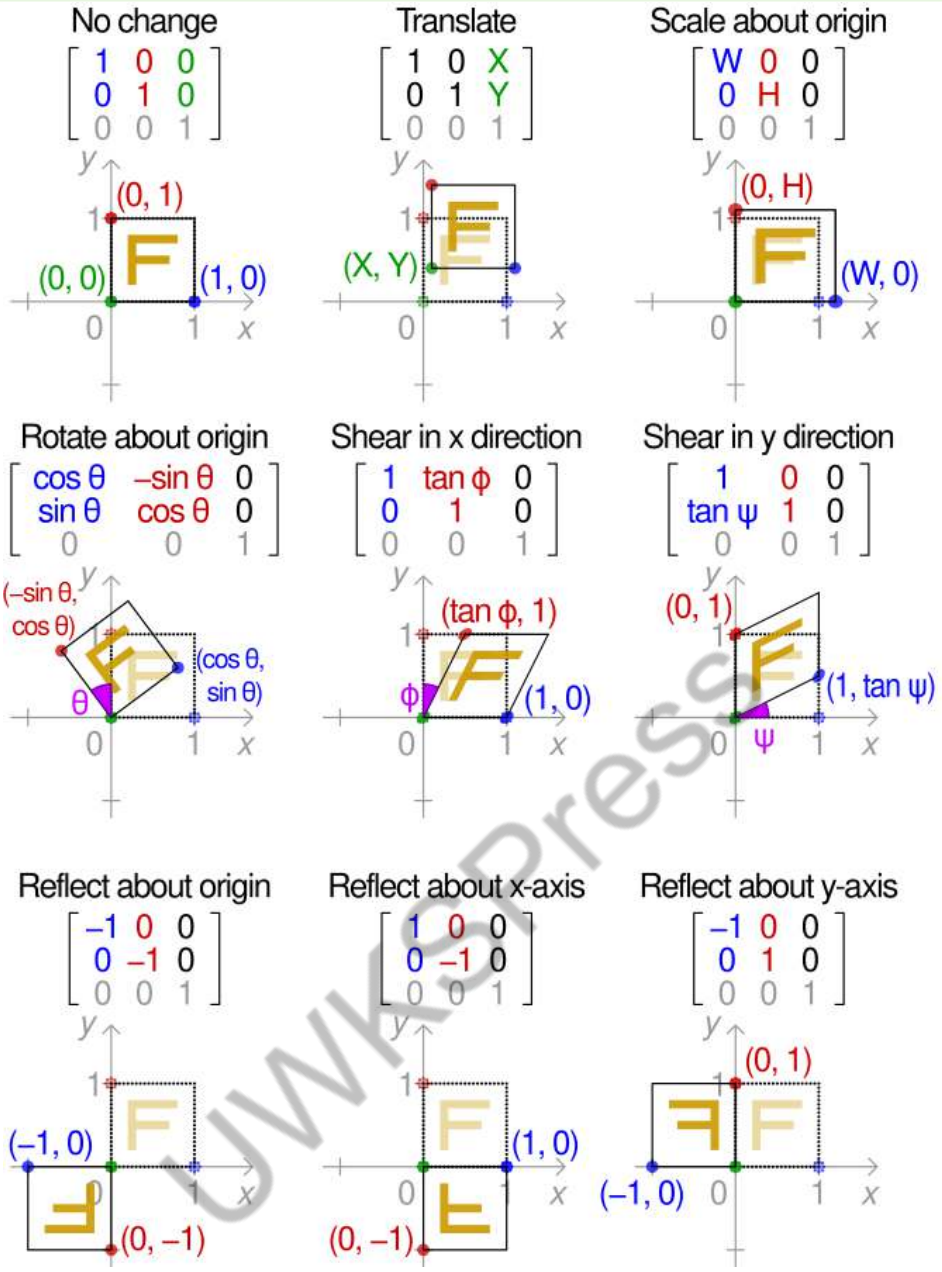
- Membantu mahasiswa mengasosiasikan gerakan abstrak dengan bentuk visual.

- Memudahkan pengkodean karena visual memberikan gambaran struktur logika.
- Memberikan konteks spasial terhadap objek yang digerakkan.

1.11.2 Diagram Transformasi

Visualisasi transformasi posisi, rotasi, dan skala bertujuan memperlihatkan bagaimana objek berpindah, berputar, atau berubah ukuran dalam ruang 2D atau 3D. Dalam Unity, semua objek memiliki komponen Transform yang mengatur ketiga aspek ini secara terpisah namun saling terhubung, seperti Gambar 1,8.

1. **Posisi (Position)**
    - Menentukan lokasi objek dalam ruang dunia (world space) atau lokal (local space).
    - Representasi vektor: (x, y, z).
    - Diagram panah pada sumbu X, Y, dan Z menunjukkan arah perpindahan.
2. **Rotasi (Rotation)**
    - Menentukan arah objek menghadap.
    - Umumnya menggunakan sistem Euler atau Quaternion.
    - Pada diagram, rotasi divisualkan sebagai panah melengkung pada masing-masing sumbu.
3. **Skala (Scale)**
    - Mengubah ukuran objek.
    - Ditampilkan pada diagram sebagai perpanjangan atau pemendekan panjang sumbu.



Gambar 1. 8 Contoh Diagram Transformasi

1.11.3 Diagram Alur Input–Gerakan

Diagram berikut menunjukkan alur kontrol dari pengguna ke hasil gerakan objek di Unity:

1. Pengguna menekan tombol (keyboard/mouse).
2. Input dibaca melalui Input.GetAxis() atau Input.GetKey().
3. Sistem mentransformasikan input menjadi arah vektor.
4. Objek digerakkan dengan transform.Translate() atau Rigidbody.velocity.

5. Scene memperbarui posisi objek secara visual.

1.11.4 Perbandingan Sistem Gerak

Tabel berikut membandingkan beberapa pendekatan sistem gerakan dalam Unity.

Tabel 1. 10 Perbandingan Sistem Gerak

Sistem Gerak	Kelebihan	Kekurangan
transform.Translate	Mudah digunakan, langsung merubah posisi	Tidak mempertimbangkan fisika
Rigidbody.velocity	Sesuai hukum fisika, efisien untuk simulasi	Perlu pengaturan collider dan massa
NavMeshAgent	Navigasi otomatis, cocok untuk NPC	Perlu baking NavMesh terlebih dahulu
CharacterController	Ideal untuk kontrol karakter (FPS/3rd person)	Tidak mendukung interaksi fisika penuh

1.11.5 Visualisasi Gerak Kompleks

Visualisasi juga bisa dilakukan menggunakan:

- **Gizmos** di Unity Editor,
- **Animator Controller Graph** untuk transisi animasi gerakan,
- **Debug.DrawLine** untuk menunjukkan arah gerakan dalam editor.

1.11.6 Latihan Mahasiswa

Tugas:

- Gambar diagram alur sistem gerak sederhana.
- Buat sketsa 2D lintasan gerakan NPC yang berpatroli.
- Buat tabel perbandingan dua metode gerak yang pernah dicoba.

Visualisasi bukan hanya alat bantu, tapi bagian penting dalam desain sistem gerakan. Dengan memahami cara kerja transformasi dan alur interaksi secara grafis, mahasiswa dapat membangun sistem kontrol objek yang lebih intuitif, efisien, dan komunikatif.

1.12 Keterkaitan Gerakan dengan Collider dan Physics

Transformasi gerakan tidak akan lengkap tanpa pemahaman mendalam mengenai interaksi objek terhadap lingkungan dan aturan fisika. Collider dan sistem fisika (physics) di Unity merupakan elemen penting untuk menciptakan pengalaman yang realistis dalam interaksi antar objek.

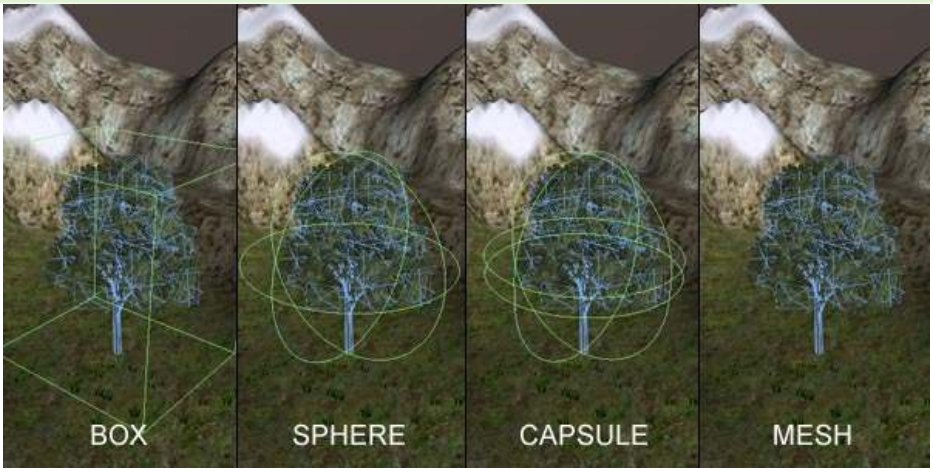
1.12.1 Apa Itu Collider?

Collider adalah komponen yang memungkinkan objek untuk mendeteksi dan merespon tabrakan. Collider tidak memiliki massa, tetapi menentukan batas fisik objek.

Jenis-jenis Collider umum:

Jenis-jenis Collider umum, Gambar 1.9:

- **Box Collider** – Berbentuk kubus atau balok, digunakan untuk objek-objek geometris sederhana seperti dinding, kotak, lantai, atau bangunan. Box Collider sangat efisien dalam perhitungan karena bentuknya teratur. Ukurannya dapat diatur secara manual untuk menyesuaikan bentuk visual objek.
- **Sphere Collider** – Digunakan untuk objek berbentuk bulat seperti bola, planet, atau efek area. Collider ini sangat cocok jika objek berinteraksi secara simetris dari semua arah. Sphere Collider cenderung lebih ringan dalam proses komputasi dibanding bentuk kompleks.
- **Capsule Collider** – Umumnya digunakan untuk karakter humanoid atau makhluk bergerak. Bentuknya silinder dengan ujung membulat membuatnya ideal untuk gerakan vertikal seperti melompat atau jatuh karena mengurangi kecenderungan untuk tersangkut di permukaan miring.
- **Mesh Collider** – Mengikuti bentuk visual mesh dari model 3D yang kompleks. Collider ini sangat akurat karena mencocokkan permukaan asli objek, tetapi lebih berat secara komputasi. Digunakan untuk bangunan kompleks, gua, atau objek dengan kontur tidak teratur. Disarankan hanya untuk objek statis karena performa.



Gambar 1. 9 Jenis jenis Collider

1.12.2 Rigidbody dan Simulasi Fisika

Rigidbody adalah komponen yang memberikan properti fisika pada objek seperti massa, kecepatan, dan gravitasi.

Tabel 1. 11 Rigidbody dan Simulasi Fisika

Properti Rigidbody	**Fungsi**
Mass	Berat objek yang mempengaruhi dorongan
Drag	Hambatan udara
Angular Drag	Hambatan putaran
Use Gravity	Mengaktifkan efek gravitasi
Is Kinematic	Objek dikontrol skrip, bukan fisika

Rigidbody diperlukan jika kita ingin objek bereaksi terhadap tabrakan, gaya, atau terpengaruh oleh hukum Newton.

1.12.3 Deteksi Tabrakan (Collision Detection)

Unity menyediakan dua metode utama:

- **Collision** → Untuk objek dengan Collider dan Rigidbody.
- **Trigger** → Untuk interaksi logika, bukan tabrakan fisik.

Contoh kode tabrakan:

```
void OnCollisionEnter(Collision collision)
{
    Debug.Log("Tabrakan dengan: " + collision.gameObject.name);
}
```

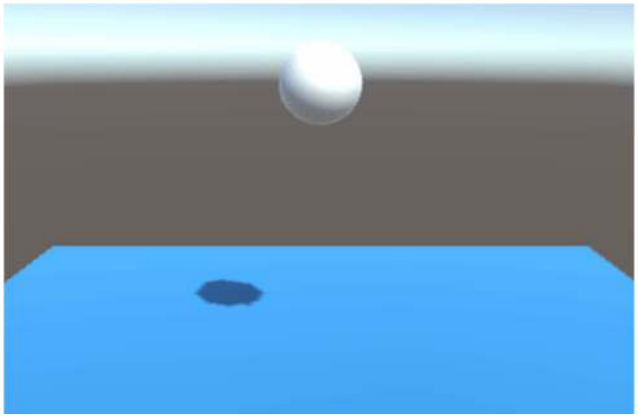
Contoh trigger:

```
void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player"))
    {
        Debug.Log("Pemain memasuki area!");
    }
}
```


1.12.4 Studi Kasus Gerakan dan Fisika

Simulasi seperti:

- Bola jatuh dan memantul menggunakan Physics.Material, Gambar 1.10.
- Karakter lompat, terkena gravitasi, dan menabrak tembok.
- Mobil menabrak objek dan berubah arah sesuai momentum.



Gambar 1. 10 contoh Bola jatuh dan memantul menggunakan Physics.Material

1.12.5 Kombinasi Transform dan Physics

Unity menyarankan:

- Gunakan transform.Translate hanya untuk objek tanpa fisika.
- Gunakan Rigidbody.MovePosition untuk objek dengan Rigidbody agar tidak konflik.

```
void FixedUpdate()
{
    rb.MovePosition(transform.position + move * speed * Time.fixedDeltaTime);
}
```

1.12.6 Tabel Komparatif Sistem Gerak

Tabel 1.12 menyajikan perbandingan berbagai metode yang digunakan dalam Unity untuk menggerakkan objek. Masing-masing metode memiliki keunggulan dan kelemahan tergantung pada kebutuhan simulasi atau game.

Penjelasan Kolom:

- **Perlu Collider**: Menunjukkan apakah metode ini membutuhkan komponen Collider untuk berfungsi optimal.
- **Perlu Rigidbody**: Apakah memerlukan Rigidbody agar bisa berinteraksi dengan sistem fisika.
- **Realistis Fisika**: Seberapa realistis gerakan yang dihasilkan terhadap hukum fisika.
- **Cocok Untuk**: Aplikasi umum metode tersebut dalam pengembangan proyek.

Penjabaran Metode:

- transform.Translate cocok digunakan untuk animasi objek ringan, seperti efek visual atau menu UI. Karena metode ini langsung memindahkan posisi, ia tidak memperhitungkan tumbukan atau gaya.
- Rigidbody.velocity sangat sesuai untuk simulasi fisika karena mengandalkan hukum Newton. Ini memungkinkan gerakan realistis, seperti dorongan, luncuran, dan benturan.

- CharacterController adalah komponen khusus Unity yang digunakan untuk mengontrol karakter humanoid. Ia menyediakan metode seperti Move() dan SimpleMove(), serta mendeteksi permukaan tanah tanpa memerlukan Rigidbody.

Dengan membandingkan metode-metode ini, pengembang dapat memilih pendekatan terbaik untuk sistem gerakan berdasarkan kebutuhan realisme, performa, dan tujuan gameplay.

Tabel 1. 12 Komparatif Sistem Gerak

Metode Gerak	Perlu Collider	Perlu Rigidbody	Realistis Fisika	Cocok Untuk
transform.Translate	Tidak	Tidak	Rendah	UI, animasi ringan
Rigidbody.velocity	Ya	Ya	Tinggi	Simulasi, game aksi
CharacterController	Ya (built-in)	Tidak	Sedang	Karakter 3D manusia

1.12.7 Latihan Mahasiswa

Tugas:

- Buat objek yang dapat memantul terhadap permukaan berbeda.
- Gunakan OnTriggerEnter untuk zona poin.
- Uji perbedaan hasil antara transform.Translate dan Rigidbody.velocity.

Collider dan physics adalah penghubung antara dunia virtual dan logika nyata. Mahasiswa yang memahami keterkaitan ini akan mampu membangun simulasi yang responsif dan realistis, serta memahami batasan dan potensi Unity dalam menangani interaksi dinamis

1.13 Tantangan Praktik: Latihan Komprehensif

Setelah memahami konsep-konsep dasar gerakan, transformasi, input, serta keterkaitannya dengan komponen fisika dan collider, bagian ini dirancang untuk menguji keterampilan mahasiswa dalam menerapkan seluruh materi melalui tantangan praktik terintegrasi. Latihan ini memadukan kemampuan teknis, logika pemrograman, serta kreativitas dalam membangun lingkungan interaktif di Unity.

1.13.1 Tujuan Latihan

- Mengintegrasikan pemahaman konsep movement object dalam satu proyek.
- Meningkatkan keterampilan scripting dan pemecahan masalah.
- Mengembangkan scene Unity yang interaktif dan fungsional.
- Menerapkan feedback visual dan audio secara dinamis.

1.13.2 Deskripsi Tantangan

Mahasiswa diminta membangun **simulasi latihan misi** dengan skenario sebagai berikut:

- Karakter utama dapat digerakkan menggunakan keyboard (WASD).
- Terdapat rintangan statis dan dinamis di dalam arena.
- Sphere sebagai target harus disentuh untuk mendapatkan skor.
- NPC atau musuh bergerak otomatis (patroli) dan menyebabkan karakter restart jika disentuh.
- Kamera mengikuti karakter dengan smooth-follow system.
- Terdapat timer atau skor yang meningkat berdasarkan interaksi.
- UI akan menampilkan skor, waktu, dan pesan status.

1.13.3 Komponen yang Harus Digunakan

Tabel 1. 13 Komponen yang Harus Digunakan

Komponen	Fungsi
Rigidbody	Untuk karakter dan objek fisika
Collider (box/sphere)	Untuk deteksi tabrakan dan area trigger

Script Movement	Untuk mengontrol pergerakan karakter
Script Patrol NPC	NPC otomatis berjalan dan berubah arah
AudioSource	Feedback suara saat target disentuh
Canvas UI	Menampilkan skor, waktu, dan pesan
Animator (opsional)	Untuk efek visual karakter atau musuh

1.13.4 Langkah-Langkah Pengerjaan

1. Rancang alur dan sketsa layout level.
2. Siapkan prefab karakter, sphere, cube, dan NPC.
3. Tulis skrip untuk:
    - Gerakan karakter.
    - NPC patrol dengan logika waypoint atau bolak-balik.
    - Deteksi trigger dengan sphere target.
    - Penambahan skor dan pengaruh waktu.
4. Tambahkan UI:
    - Skor
    - Timer
    - Pesan "Game Over" atau "Selamat!"
5. Tambahkan suara untuk target dan efek tabrakan.
6. Uji keseluruhan dan dokumentasikan.

1.13.5 Penilaian

Tabel 1. 14 Aspek penilaian

Aspek yang Dinilai	Bobot
Fungsi kontrol dan logika	25%
Interaksi target dan musuh	20%
Tampilan UI dan feedback	20%
Struktur scene dan scripting	15%

Kreativitas tambahan fitur	10%
Dokumentasi dan keterbacaan	10%

1.13.6 Contoh Tampilan Akhir

Gambar 1. 11 Contoh Tampilan Akhir

1.13.7 Refleksi Mahasiswa

Jawablah pertanyaan berikut secara pribadi:

- Apakah tantangan ini memperkuat pemahaman transformasi dan input?
- Kesulitan teknis apa yang ditemukan dan bagaimana solusinya?
- Komponen apa yang paling menyulitkan? Kenapa?
- Jika mengulang dari awal, fitur apa yang ingin dikembangkan?

Latihan komprehensif ini menjadi sarana mengukur pemahaman nyata mahasiswa dalam membangun sistem interaktif berbasis Unity. Tidak hanya aspek teknis, tetapi juga kreativitas dan ketepatan logika sangat diuji dalam latihan ini. Ini menjadi penguatan sebelum mahasiswa memasuki bab-bab lanjutan yang lebih kompleks.

1.14 Tugas Kreativitas dan Rubrik Penilaian

Subbab ini dirancang untuk memberikan ruang eksplorasi dan pengembangan ide kreatif mahasiswa berdasarkan seluruh materi yang telah dipelajari dalam Bab 1. Tugas ini bersifat terbuka dan mengedepankan inovasi, kolaborasi, dan aplikasi nyata dari sistem movement object, transformasi, dan interaksi.

1.14.1 Tujuan Tugas Kreativitas

- Mendorong mahasiswa untuk menciptakan proyek unik dan inovatif.
- Menilai kemampuan integrasi antar komponen Unity.
- Membangun rasa percaya diri dalam eksplorasi mandiri.
- Menyusun dokumentasi teknis dan presentasi proyek.

1.14.2 Bentuk Tugas

Mahasiswa diminta untuk membuat sebuah **prototipe interaktif** berbasis Unity dengan ketentuan:

- Bebas memilih tema (edukasi, simulasi, permainan, presentasi).
- Harus memuat minimal:
    - Karakter yang dapat digerakkan.
    - Objek dinamis atau interaktif (target, musuh, atau NPC).
    - Sistem skor atau status berbasis interaksi.
    - Feedback visual dan/atau audio.
    - UI sederhana (Canvas, Text, Timer, atau Panel).

Tugas dapat disusun secara individu atau berkelompok (maks. 3 orang) dengan peran terstruktur (coding, desain, dokumentasi).

1.14.3 Langkah Pengembangan

1. **Ide dan Perencanaan**:
    - Diskusikan tema dan alur logika proyek.
    - Buat sketsa level atau diagram alur interaksi.
2. **Pengembangan Unity**:
    - Susun scene dan prefab objek utama.
    - Tambahkan skrip movement, interaksi, dan skor.

- Integrasikan suara dan animasi jika perlu.

3. **Pengujian dan Debugging**:
   - Uji semua interaksi dan tanggapan sistem.
   - Perbaiki error dan polish visual.
4. **Dokumentasi dan Presentasi**:
   - Jelaskan komponen proyek, alur kerja, kendala, dan solusi.
   - Siapkan file Unity dan rekaman video/demo aplikasi.

1.14.4 Rubrik Penilaian Kreativitas

Tabel 1. 15 Rubrik Penilaian Kreativitas

Aspek Penilaian	Bobot
Inovasi dan keunikan ide	20%
Fungsi interaktif dan kelengkapan fitur	25%
Kesesuaian implementasi teknis	20%
Struktur scene, UI, dan feedback	15%
Dokumentasi dan presentasi	10%
Kerja tim (jika berkelompok)	10%

1.14.5 Contoh Ide Proyek

- **Simulasi Pelatihan Kebencanaan**: Karakter mengevakuasi objek dari zona bahaya.
- **Game Edukatif Matematika**: Karakter memilih hasil hitung yang benar untuk maju.
- **Maze Game**: Pemain mencari jalan keluar sambil menghindari NPC patrol.
- **Simulasi Protokol Kesehatan**: Pemain menghindari kerumunan, memakai masker, dan cuci tangan.

Tugas ini merupakan puncak eksplorasi kreatif mahasiswa dalam Bab 1. Diharapkan mahasiswa tidak hanya menunjukkan kemampuan teknis, tetapi juga mampu berpikir kritis, bekerja sama, dan berani berinovasi dalam menyampaikan ide melalui media interaktif berbasis Unity.

1.15 Rangkuman Konseptual.

Bagian ini menyajikan tinjauan menyeluruh terhadap materi Bab 1 yang telah dibahas, dengan fokus pada penguatan pemahaman konsep dan integrasi praktis dalam Unity. Rangkuman ini dapat dijadikan referensi cepat saat mahasiswa melanjutkan ke bab-bab berikutnya atau menyusun proyek akhir berbasis Unity.

1.15.1 Intisari Konsep

- **Transformasi (Position, Rotation, Scale)** adalah dasar dalam memanipulasi objek secara spasial di lingkungan 3D.
- **Sistem Koordinat** dalam Unity berbasis sistem tangan kanan, dengan orientasi X (kiri-kanan), Y (atas-bawah), dan Z (maju-mundur).
- **Komponen Transform** mengendalikan posisi, rotasi, dan skala objek baik di local space maupun world space.
- **Input Interaktif** seperti keyboard dan mouse memungkinkan pengguna mengontrol objek secara real-time.
- **Visualisasi Gerakan** memperjelas pemahaman konsep melalui diagram, alur logika, dan tabel perbandingan.
- **Collider dan Physics** diperlukan untuk menciptakan simulasi yang realistis, memungkinkan objek berinteraksi dengan lingkungan.
- **Rigidbody** memungkinkan objek merespons gaya dan gravitasi sesuai hukum fisika.

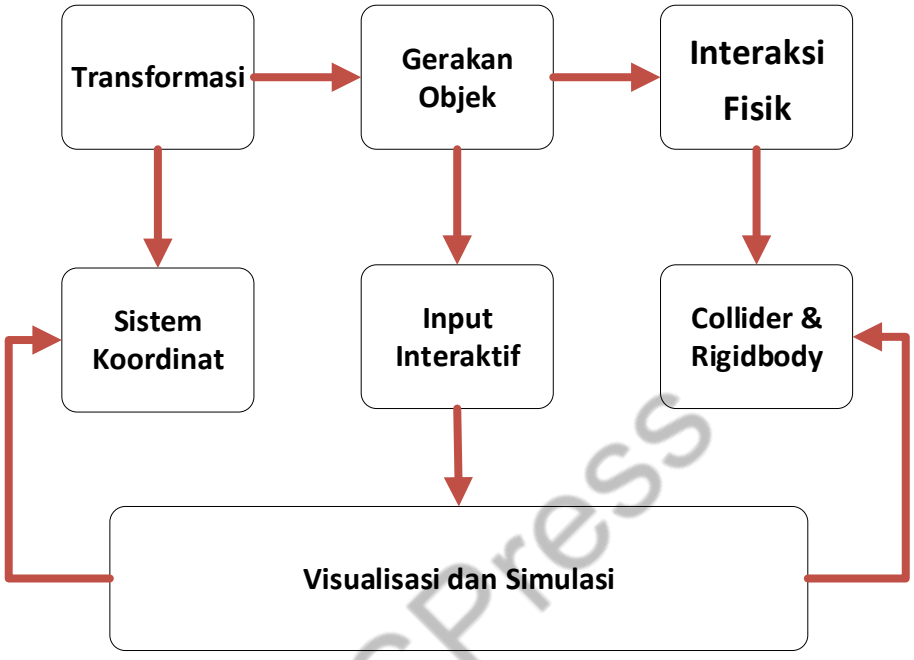
1.15.2 Integrasi dalam Unity

Konsep-konsep di atas terimplementasi melalui scripting dan konfigurasi komponen:

- Transform.Translate, Rigidbody.velocity, dan CharacterController.Move() adalah metode umum untuk memindahkan objek.
- Collider seperti Box, Sphere, dan Mesh digunakan untuk mendeteksi tabrakan atau trigger.
- Unity UI (Canvas) dimanfaatkan untuk menampilkan status game seperti skor, waktu, atau pesan sistem.
- Sound, animation, dan visual feedback menambah kedalaman interaktif dalam pengalaman pengguna.

1.15.3 Peta Konseptual Bab 1

Gambar 1.12, menunjukkan skema hubungan antar konsep utama dalam Bab 1:



Gambar 1. 12 Hubungan antar konsep utama dalam Bab 1

1.15.4 Relevansi Terhadap Pengembangan Proyek

Dengan menguasai konsep-konsep ini, mahasiswa:

- Siap membangun prototipe aplikasi interaktif dan game sederhana.
- Dapat menganalisis dan memperbaiki sistem kontrol gerakan.
- Memiliki dasar kuat untuk mengintegrasikan AI, kamera, dan animasi pada bab selanjutnya.

Bab 1 membekali mahasiswa dengan fondasi utama dalam pemrograman grafika interaktif. Keterampilan memahami dan mengimplementasikan transformasi, input, serta interaksi fisik di Unity merupakan landasan penting bagi setiap proyek pengembangan game dan simulasi edukatif.

BAB 2 COLLIDER DAN RIGIDBODIES

2.1 Tujuan Pembelajaran

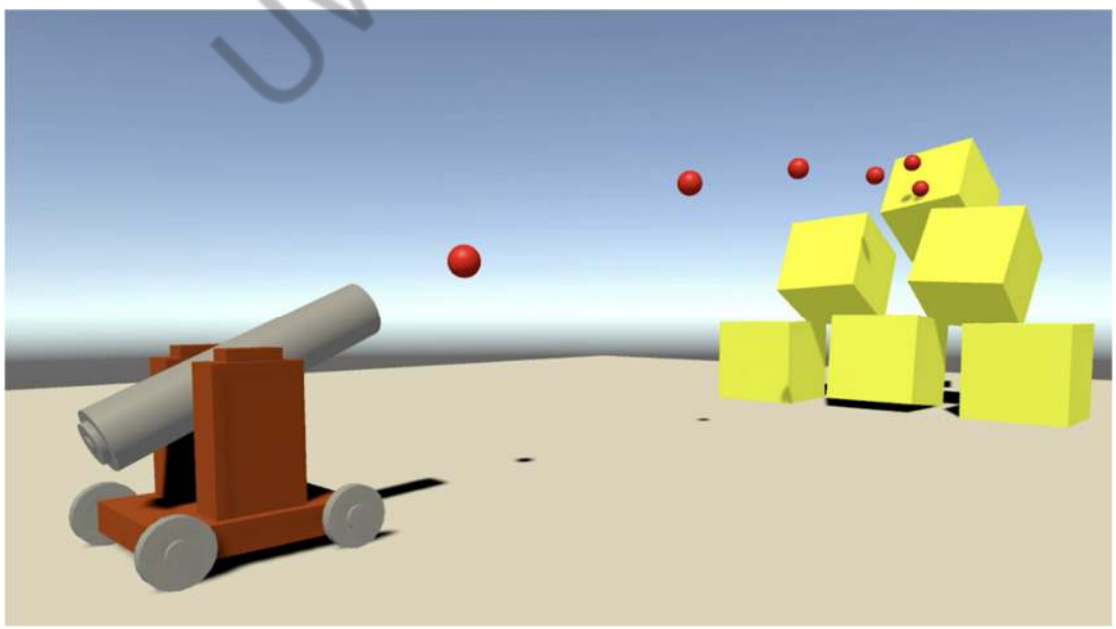
Bab ini bertujuan membekali mahasiswa dengan pemahaman mendalam dan keterampilan praktis dalam menerapkan sistem fisika di Unity, terutama melalui dua komponen utama yaitu Rigidbody dan Collider. Sistem ini menjadi tulang punggung dalam menciptakan interaksi dunia virtual yang realistis dan dinamis.

2.1.1 Menjelaskan Konsep Dasar Sistem Fisika Unity

Unity menyediakan sistem fisika built-in berbasis mesin fisika NVIDIA PhysX. Sistem ini bertugas untuk mensimulasikan:

- Gravitasi
- Gaya dan momentum
- Tumbukan (collision)
- Friksi (gesekan)
- Elastisitas dan pantulan

Konsep ini memungkinkan objek virtual berperilaku seperti di dunia nyata.



Gambar 2. 1 Ilustrasi Sistem Fisika dalam Unity

Contoh pada Gambar 2.1, secara sederhananya adalah ketika sebuah peluru ditembakkan ke benda lain , maka dengan Rigidbody dan Collider yang tepat, ia akan memantul dimana sudut datang sesuai sudut pantul dan kemudian jatuh karena gravitasi, dan memantul jika memiliki nilai "bounciness".

Fungsi sistem fisika ini adalah untuk:

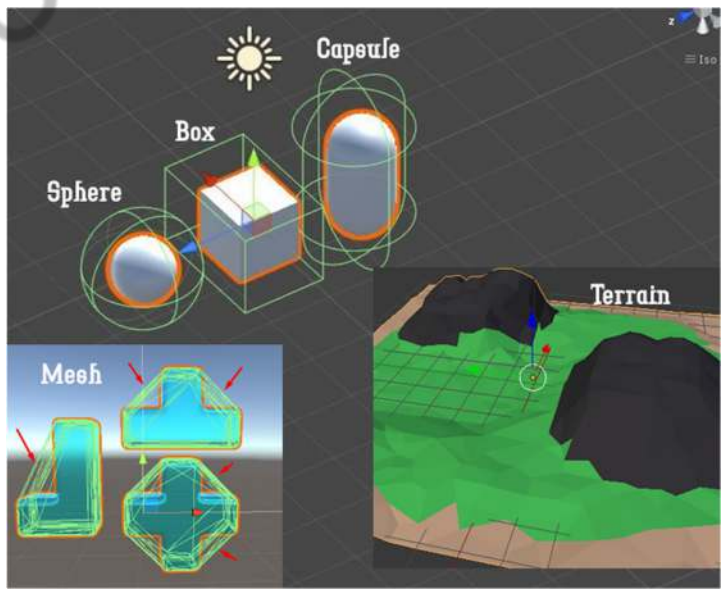
- Menghemat waktu perhitungan logika interaksi secara manual.
- Menyediakan API yang mendukung efek real-time.
- Menambah kedalaman gameplay.

2.1.2 Mengidentifikasi dan Menerapkan Berbagai Jenis Collider

Collider berfungsi sebagai batas tidak terlihat yang digunakan Unity untuk mendeteksi tabrakan antar objek, terlihat seperti Gambar 2.2.

Jenis-jenis Collider:

- **Box Collider**: Digunakan untuk bentuk kotak, lantai, dinding.
- **Sphere Collider**: Ideal untuk objek bulat seperti bola.
- **Capsule Collider**: Umum untuk karakter manusia atau makhluk vertikal.
- **Mesh Collider**: Menyesuaikan bentuk kompleks objek.
- **Terrain Collider**: Untuk permukaan lanskap 3D.



Gambar 2. 2 Tipe Collider dalam Unity

Contoh penggunaan:

- Sebuah karakter humanoid menggunakan Capsule Collider agar tidak tersangkut di medan.
- Dinding menggunakan Box Collider agar karakter tidak bisa menembus.

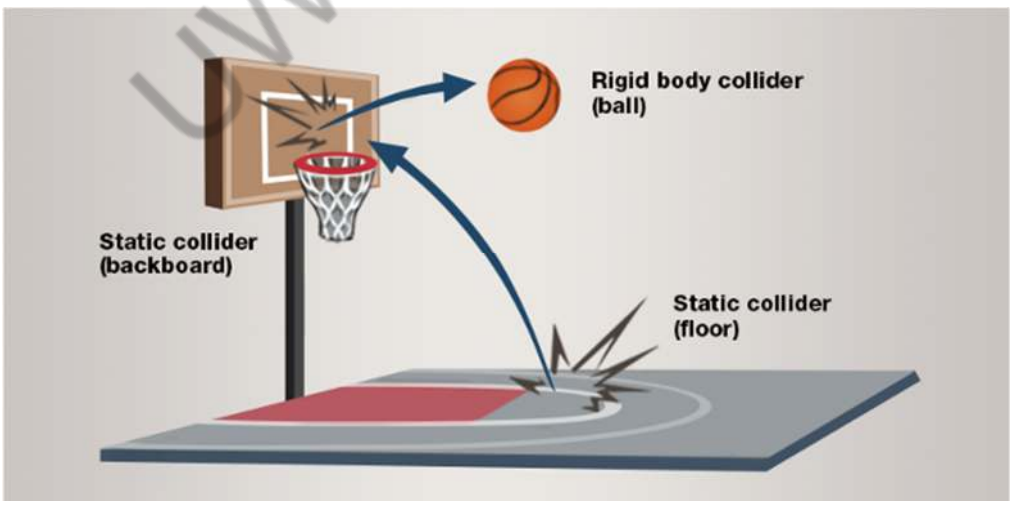
2.1.3 Menjelaskan Fungsi Rigidbody dan Penerapannya

Rigidbody seperti Gambar 2.3, adalah komponen yang memungkinkan objek untuk:

- Jatuh karena gravitasi.
- Bereaksi terhadap gaya (force) dan dorongan.
- Menyimpan energi kinetik.
- Mengalami tabrakan yang realistis.

Properti utama Rigidbody:

- **Mass**: Berat objek.
- **Drag**: Hambatan udara.
- **Angular Drag**: Hambatan putaran.
- **Use Gravity**: Menentukan apakah objek jatuh.
- **Is Kinematic**: Jika aktif, objek dikendalikan lewat skrip, bukan fisika.



Gambar 2. 3 Konsep Rigidbody dan Relasinya dengan Collider

Penerapan dalam game:

- Bola dilempar ke dinding dengan AddForce().

- Karakter jatuh saat berada di tepi.
- Objek meluncur di atas permukaan miring.

2.1.4 Membuat Simulasi Tabrakan dan Efek Fisika

Simulasi tabrakan mencakup:

- Interaksi antar objek solid.
- Efek pantulan, suara, dan perubahan kecepatan.

Unity menyediakan dua jenis deteksi:

- **Collision**: Fisik, seperti menabrak dinding.
- **Trigger**: Logika, seperti masuk ke zona checkpoint.

Contoh kode tabrakan:

```
void OnCollisionEnter(Collision collision)
{
    Debug.Log("Menabrak: " + collision.gameObject.name);
}

void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("PointZone"))
    {
        score++;
        Debug.Log("Skor bertambah!");
    }
}
```

Melalui pengaturan material fisika (Physics Material), kita juga dapat:

- Mengatur tingkat gesekan (friction).
- Menentukan nilai elastisitas (bounciness).

- Menciptakan permukaan licin, kasar, atau lentur.

Simulasi ini mendukung:

- Permainan berbasis physics puzzle.
- Reaksi dinamis pada game aksi.
- Lingkungan edukatif (simulasi gravitasi, momentum).

Mahasiswa diharapkan memahami prinsip-prinsip dasar fisika yang diterapkan dalam Unity, mengenali jenis Collider dan fungsinya, serta dapat mengimplementasikan Rigidbody untuk simulasi gaya dan tumbukan. Dengan landasan ini, mereka akan lebih siap untuk membangun aplikasi dan game berbasis interaksi fisika yang realistis dan edukatif.

2.2 Landasan Teoretis Fisika dalam Unity

Subbab ini membahas dasar-dasar fisika klasik yang menjadi pondasi bagi simulasi interaksi objek dalam Unity. Unity memanfaatkan prinsip-prinsip mekanika Newton untuk menciptakan gerakan dan reaksi realistis antar objek, baik dalam bentuk dorongan, tumbukan, maupun pengaruh gravitasi.

2.2.1 Pengantar Hukum Newton dalam Simulasi

Unity mensimulasikan hukum Newton melalui sistem Rigidbody dan Collider. Tiga hukum Newton yang diterapkan secara implisit:

1. **Hukum I (Inersia):** Objek akan tetap diam atau bergerak lurus beraturan jika tidak ada gaya luar. Di Unity, objek tanpa gaya atau collider akan tetap pada posisinya atau melayang.
2. **Hukum II (F=ma):** Percepatan suatu objek berbanding lurus dengan gaya yang diberikan dan berbanding terbalik dengan massanya. Unity mengimplementasikan ini dalam AddForce(), mass, dan velocity.
3. **Hukum III (Aksi-Reaksi):** Jika objek A memberi gaya pada B, maka B memberikan gaya yang sama besar dan berlawanan arah ke A. Ini terlihat saat dua Rigidbody bertabrakan dan saling terpental.

Ilustrasi lain seperti pada Gambar 2.4

Gambar 2. 4 Ilustrasi Hukum Newton

2.2.2 Pengaruh Gaya, Percepatan, Massa, dan Gravitasi

- **Gaya (Force):** Unity menggunakan Rigidbody.AddForce() untuk memberikan gaya pada objek. Gaya ini akan mengubah kecepatan (velocity) dan arah.
- **Percepatan (Acceleration):** Tergantung pada gaya dan massa. Objek yang lebih berat butuh gaya lebih besar untuk menghasilkan percepatan yang sama.
- **Massa (Mass):** Diatur melalui properti Rigidbody. Massa tinggi berarti objek lebih sulit digerakkan.
- **Gravitasi (Gravity):** Unity memiliki gravitasi global default (biasanya -9.81 di sumbu Y). Bisa dimodifikasi melalui Physics.gravity.

Contoh Kode Gaya dan Gravitasi:

```
void FixedUpdate()
{
    Rigidbody rb = GetComponent<Rigidbody>();
    Vector3 gayaDorong = new Vector3(10, 0, 0);
```

```
    rb.AddForce(gayaDorong);
}
```

🎯 **Tip:** Gunakan FixedUpdate() untuk penghitungan fisika karena sinkron dengan sistem fisika internal Unity.

2.2.3 Simulasi Interaksi Fisik Antar Objek

Interaksi fisik melibatkan:

- **Deteksi tabrakan** menggunakan Collider.
- **Respons fisika** menggunakan Rigidbody.
- **Efek seperti pantulan, dorongan balik, atau pelambatan** berdasarkan massa dan friction.

Contoh:

- Dua bola dengan massa berbeda bertabrakan: bola ringan terpental lebih jauh.
- Karakter melompat lalu menabrak dinding dan berhenti.
- Objek bergerak menuruni tanjakan karena gravitasi.

Unity mendasarkan simulasi fisikanya pada prinsip-prinsip hukum Newton, khususnya gaya, percepatan, dan gravitasi. Dengan memahami hubungan antara massa, gaya, dan percepatan, mahasiswa dapat menciptakan dunia virtual yang memiliki perilaku realistis dalam konteks fisika dasar. dalam simulasi.

- **Pengaruh gaya, percepatan, massa, dan gravitasi.**
- **Simulasi interaksi fisik antar objek.**

2.3 Rigidbody: Komponen Simulasi Fisika

Rigidbody adalah komponen inti dalam sistem fisika Unity yang memungkinkan objek memiliki perilaku seperti di dunia nyata, seperti jatuh karena gravitasi, terpental setelah

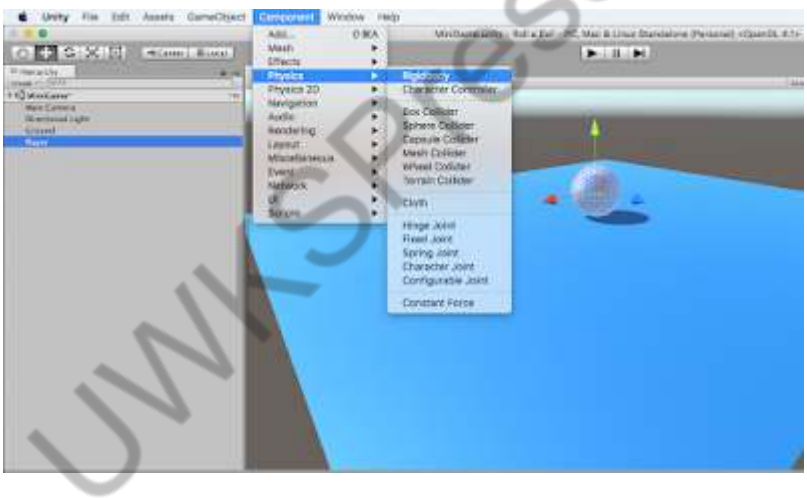
tabrakan, atau terdorong oleh gaya tertentu. Penerapan Rigidbody sangat penting dalam membangun game dan simulasi yang realistis.

2.3.1 Definisi Rigidbody dan Fungsi Utama

Secara umum, Rigidbody adalah komponen Unity yang memberikan kemampuan fisika dinamis kepada objek. Tanpa Rigidbody, objek tidak akan merespons hukum Newton seperti gravitasi, momentum, atau gaya luar. Ilustrasi: Rigidbody pada Objek 3D seperti Gambar 2.5.

Fungsi utama Rigidbody:

- Mengaktifkan simulasi gravitasi dan gaya.
- Menyediakan interaksi fisika antar objek.
- Mendukung simulasi tumbukan secara otomatis.
- Mengatur respons terhadap kekuatan eksternal dengan efisien.



Gambar 2. 5 Ilustrasi: Rigidbody pada Objek 3D

2.3.2 Properti Rigidbody: Mass, Drag, Angular Drag, Use Gravity

1. **Mass**: Menentukan berat objek. Mempengaruhi seberapa mudah objek didorong atau dipengaruhi oleh gaya lain.
2. **Drag**: Hambatan gerak linear. Semakin tinggi nilai drag, semakin lambat objek bergerak.
3. **Angular Drag**: Hambatan rotasi. Mempengaruhi seberapa cepat objek berhenti berputar.

4. **Use Gravity**: Jika diaktifkan, objek akan jatuh secara otomatis sesuai gravitasi global.
5. **Is Kinematic**: Menonaktifkan simulasi fisika. Objek hanya dapat digerakkan melalui skrip.

Contoh pengaturan Rigidbody dalam Unity Editor:

- Bola dengan massa 1, drag 0.5, dan gravity aktif akan jatuh dan melambat karena hambatan udara.

📌 **Tip**: Gunakan massa yang realistis untuk menjaga simulasi tetap stabil. Massa terlalu kecil atau besar dapat menyebabkan anomali.

2.3.3 Rigidbody Kinematic vs Dinamis

Tabel 2. 1 Rigidbody Kinematic vs Dinamis

Tipe Rigidbody	Fitur Utama	Kapan Digunakan
Dinamis	Terpengaruh fisika, gaya, dan tabrakan	Objek yang bergerak dan berinteraksi secara alami
Kinematic	Tidak dipengaruhi fisika, digerakkan skrip	Objek yang dikontrol secara manual atau AI

Contoh penggunaan:

- **Dinamis**: Bola, karakter pemain, mobil.
- **Kinematic**: Pintu otomatis, platform bergerak.

⚠️ **Catatan**: Kinematic Rigidbody tidak akan bereaksi terhadap tumbukan kecuali dengan Collider bertipe non-kinematic.

2.3.4 Rigidbody 2D vs Rigidbody 3D

Unity menyediakan dua sistem fisika berbeda:

- **Rigidbody (3D)**: Untuk proyek 3D, bekerja dengan Vector3.

- **Rigidbody2D**: Untuk proyek 2D, bekerja dengan Vector2 dan Collider 2D.

Perbedaan utama:

- Komponen fisika dan skrip berbeda (AddForce() vs AddForce2D()).
- Collider 2D tidak kompatibel dengan Rigidbody 3D, dan sebaliknya.
-

💡 **Tips Penggunaan:**

- Gunakan Rigidbody 2D jika seluruh proyek Anda berbasis 2D untuk efisiensi dan performa.
- Hindari mencampur Rigidbody 2D dan 3D dalam satu objek.

Rigidbody merupakan tulang punggung simulasi fisika di Unity. Dengan memahami jenis, fungsi, dan konfigurasi propertinya, pengembang dapat menciptakan efek gerak dan tumbukan yang realistis, efisien, dan sesuai dengan kebutuhan gameplay atau edukasi berbasis fisika. dan fungsi utama.

2.4 Collider: Batasan Interaksi Objek

2.4.1 Pengertian Collider dan Deteksi Benturan

Dalam simulasi fisika di Unity, **Collider** adalah komponen yang memungkinkan objek mendeteksi tabrakan atau kontak dengan objek lain. Collider tidak hanya menentukan batas-batas fisik suatu objek, tetapi juga berperan penting dalam sistem **interaksi**, seperti:

- Deteksi tumbukan (collision),
- Pemicu (trigger),
- Interaksi antar objek menggunakan Rigidbody.

Collider bekerja **bersama dengan Rigidbody**, meskipun secara teknis, collider tetap bisa digunakan tanpa Rigidbody untuk keperluan statis seperti dinding atau lantai.

2.4.2 Jenis-Jenis Collider

Unity menyediakan berbagai jenis collider sesuai bentuk objek dan kebutuhan performa simulasi:

1. Box Collider

- Digunakan untuk objek berbentuk kotak atau kubus.
- Hemat performa dan sangat cocok untuk platform, dinding, dan objek statis.

Fitur:

- Dapat diubah ukuran dan posisi offset-nya.
- Bekerja optimal untuk objek geometri sederhana.

2. Sphere Collider

- Membungkus objek dalam bentuk bola.
- Digunakan untuk objek bulat seperti bola, planet, atau granat.

Fitur:

- Ukuran radius mudah diatur.
- Ideal untuk mendeteksi jarak secara radial.

3. Capsule Collider

- Gabungan bentuk silinder dan setengah bola di kedua ujungnya.
- Sangat umum digunakan untuk karakter manusia/humanoid.

Fitur:

- Mengikuti bentuk tubuh vertikal.
- Umumnya dipasangkan dengan Rigidbody untuk kontrol karakter.

4. Mesh Collider

- Membentuk collider mengikuti bentuk asli model 3D.
- Digunakan untuk objek kompleks seperti patung, bangunan detail, atau kendaraan.

Fitur:

- Bisa dibuat menjadi convex (untuk interaksi dinamis).
- Lebih berat secara performa — sebaiknya digunakan pada objek statis.

5. Terrain Collider

- Dirancang khusus untuk objek Terrain di Unity.
- Digunakan dalam game dengan lingkungan besar seperti perbukitan atau pegunungan.

Fitur:

- Otomatis mengikuti bentuk Terrain.
- Digunakan bersama komponen Terrain di Unity Editor.

2.4.3 Collider 2D vs Collider 3D

Unity membedakan komponen fisika menjadi **2D dan 3D**. Begitu pula dengan collider:

Tabel 2. 2 komponen fisika 2D dan 3D

Fitur	**Collider 2D**	**Collider 3D**
Sistem Fisika	Physics 2D (BoxCollider2D, dll)	Physics 3D (BoxCollider, dll)
Interaksi	Untuk game 2D (top-down, side scroller)	Untuk game 3D atau simulasi ruang
Komponen Rigidbody	Rigidbody2D	Rigidbody
Deteksi Tabrakan	OnCollisionEnter2D, OnTriggerEnter2D	OnCollisionEnter, OnTriggerEnter
Editor Gizmo	Terlihat sebagai garis datar di Scene	Terlihat sebagai bentuk volume penuh

Contoh penerapan:

- **Collider2D** untuk game platformer 2D seperti *Mario-style*.
- **Collider 3D** untuk simulasi ruang seperti *first-person exploration*.

- **Collider adalah batas fisik virtual yang digunakan Unity untuk mendeteksi interaksi antar objek.**
- **Jenis collider disesuaikan dengan bentuk objek dan kebutuhan performa.**
- **Collider 3D dan 2D digunakan secara terpisah, sesuai dengan sistem fisika yang diterapkan.**
- **Pemilihan collider yang tepat akan meningkatkan efisiensi dan realisme simulasi.**

2.5 Trigger vs Collision: Logika Interaksi

Dalam pengembangan simulasi dan game menggunakan Unity, dua jenis mekanisme interaksi utama yang perlu dipahami adalah **Trigger** dan **Collision**. Keduanya bekerja melalui komponen **Collider**, namun memiliki tujuan dan efek yang sangat berbeda.

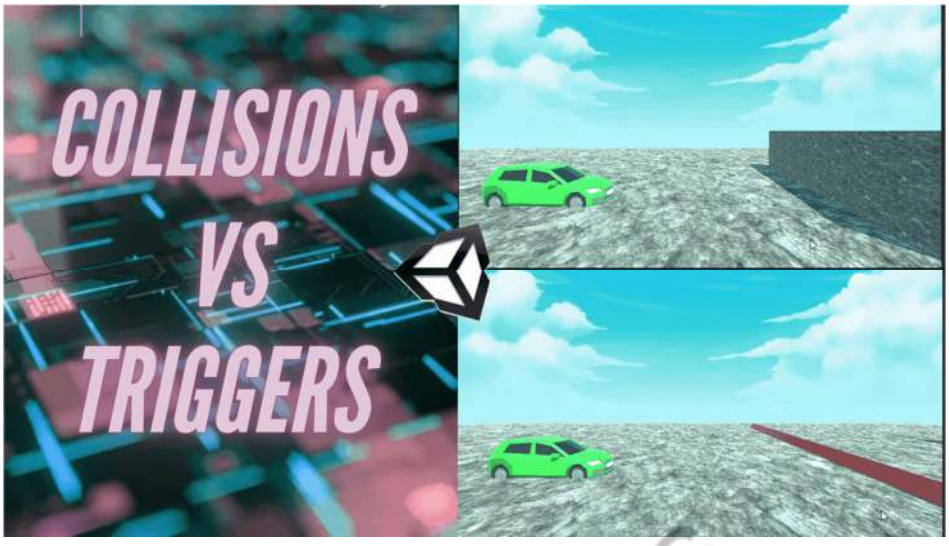
2.5.1 Pengertian Trigger dan Collision

- **Collision** adalah interaksi fisik antar objek. Saat dua objek bertabrakan, Unity akan menghitung dampaknya secara fisik, berdasarkan massa, kecepatan, dan arah. Interaksi ini memerlukan komponen **Rigidbody** agar fisika dapat diproses secara dinamis.
- **Trigger** adalah area logika tak terlihat (virtual zone) yang digunakan untuk mendeteksi apakah suatu objek memasuki area tersebut. Berbeda dari Collision, Trigger tidak memberikan efek fisika, melainkan digunakan untuk menjalankan perintah seperti menambah skor, mengganti level, atau menyalakan lampu.

📌 **Perbedaan utama:**

- Collision menimbulkan efek fisik.
- Trigger hanya memicu peristiwa logika (tanpa tabrakan nyata).

2.5.2 Ilustrasi Trigger dan Collision



Gambar 2. 6 Perbandingan antara trigger (deteksi masuk zona) dan collision (benturan fisik antar objek).

Gambar 2.6 menunjukkan **Trigger** dan **Collision**, dimana dua metode utama dalam Unity untuk mendeteksi interaksi antar objek.

- **Trigger** digunakan untuk mendeteksi apakah sebuah objek memasuki zona tertentu tanpa efek fisika. Biasanya digunakan untuk mengaktifkan event seperti menambah skor, membuka pintu, atau mengganti level.
- **Collision** menimbulkan respons fisika seperti tabrakan, hentakan, atau pantulan. Digunakan ketika objek perlu berinteraksi secara fisik, misalnya menabrak dinding atau jatuh ke lantai.

2.5.3 Cara Mengaktifkan Trigger

Untuk mengubah Collider menjadi Trigger:

1. Tambahkan komponen BoxCollider, SphereCollider, atau sejenisnya pada objek.
2. Centang properti Is Trigger pada inspector.

Contoh skrip deteksi trigger:

csharp

CopyEdit

```
void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player"))
    {
        Debug.Log("Pemain memasuki area trigger.");
    }
}
```

Untuk collision:

csharp

CopyEdit

```
void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.CompareTag("Dinding"))
    {
        Debug.Log("Pemain menabrak dinding!");
    }
}
```


2.5.4 Callback Event pada Trigger dan Collision

Unity menyediakan beberapa metode (callback) yang otomatis dipanggil ketika interaksi terjadi, seperti Tabel 2.3:

Tabel 2. 3 Jenis metode Callback

Jenis Callback	**Fungsi**
OnCollisionEnter()	Saat objek mulai bertabrakan (fisik).
OnCollisionStay()	Selama objek masih bersentuhan.
OnCollisionExit()	Saat objek keluar dari benturan.
OnTriggerEnter()	Saat objek memasuki area trigger.
OnTriggerStay()	Selama objek berada dalam area trigger.
OnTriggerExit()	Saat objek keluar dari trigger.

2.5.5 Studi Kasus: Trigger Koin dan Collision Dinding

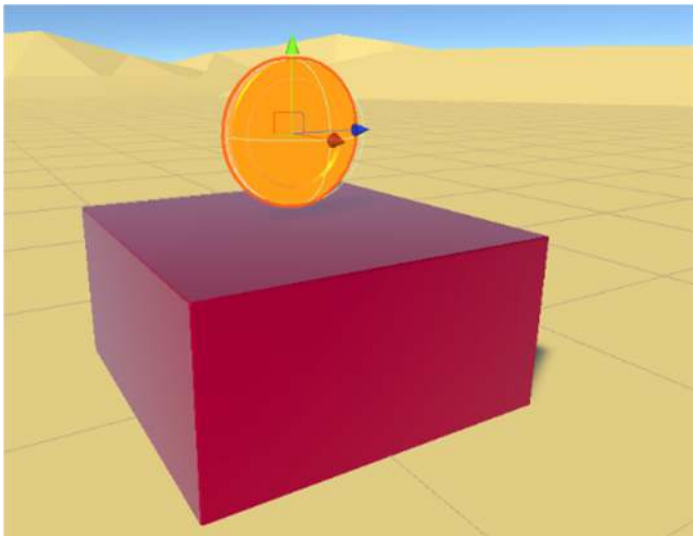
Studi 1: Trigger untuk Mengumpulkan Koin

- Koin memiliki Collider dengan Is Trigger aktif.
- Pemain memiliki Rigidbody dan Collider biasa.
- Saat pemain menyentuh koin, koin akan menghilang dan skor bertambah.

Skrip csharp

CopyEdit

```
void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("Coin"))
    {
        Destroy(other.gameObject);
        score += 1;
    }
}
```

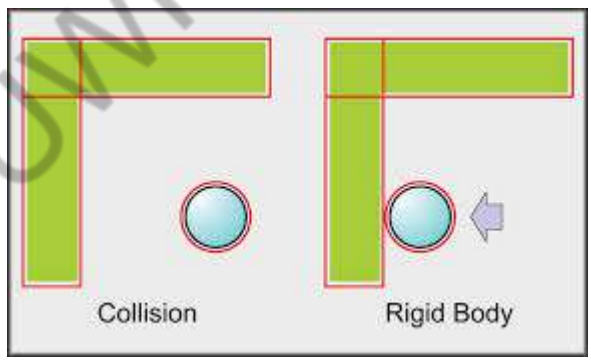



Gambar 2. 7 Studi kasus 1 (Trigger Koin dan Collision Dinding)

Studi 2: Collision untuk Menabrak Rintangan

- Rintangan menggunakan Collider biasa (non-trigger).
- Pemain dengan Rigidbody menabrak dan terhenti.

```
csharp
CopyEdit
void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.CompareTag("Wall"))
    {
        Debug.Log("Menabrak rintangan!");
        health -= 10;
    }
}
```

Gambar 2. 8 Studi Kasus 2-Collision untuk Menabrak rintangan

2.5.6 Perbandingan Trigger vs Collision

Tabel 2. 4 Perbandingan Trigger vs Collision

Fitur	**Trigger**	**Collision**
Efek fisik	Tidak	Ya

Butuh Rigidbody?	Salah satu objek (boleh)	Minimal satu objek
Umum digunakan untuk	Deteksi zona, skor, sensor	Tabrakan nyata, blokir gerakan
Keunggulan	Ringan, cepat, fleksibel	Realistis, cocok untuk gameplay fisik
Contoh	Koin, checkpoint, pintu otomatis	Dinding, lantai, musuh

Unity menyediakan dua pendekatan utama untuk interaksi objek: Trigger **untuk logika deteksi dan** Collision **untuk efek fisika nyata. Keduanya memiliki fungsi dan skenario penggunaan yang berbeda.**

2.6 Studi Kasus 1: Pemantulan Bola (Bounciness)

Simulasi pemantulan bola adalah praktik mendasar namun sangat penting dalam pembelajaran fisika digital. Studi kasus ini memberikan pemahaman tentang bagaimana **Rigidbody**, **Collider**, dan **Physics Material** bekerja sama untuk menghasilkan efek pantulan yang realistis di Unity.

2.6.1 Konsep Fisika di Balik Pemantulan

Dalam fisika klasik, pantulan benda padat dipengaruhi oleh **elastisitas** dan **koefisien restitusi**, yang menyatakan seberapa besar energi kinetik dipertahankan setelah benturan. Nilai **bounciness** di Unity setara dengan koefisien tersebut.

- Nilai 0 = tidak ada pantulan (inelastis total).
- Nilai 1 = pantulan sempurna (tanpa kehilangan energi).

Unity mensimulasikan ini melalui **Physics Material** yang dapat diterapkan ke permukaan atau objek tertentu.

2.6.2 Menyiapkan Physics Material di Unity

Physics Material adalah properti yang menentukan **bounciness** (pantulan) dan **friction** (gesekan) suatu objek.

📌 *Langkah-langkah:*

1. Di panel Project, klik kanan → Create → Physics Material.
2. Beri nama: BolaBounciness.
3. Atur properti berikut:
    - Bounciness: 1
    - Dynamic Friction: 0
    - Static Friction: 0
    - Bounce Combine: Maximum

2.6.3 Menambahkan Rigidbody dan Collider ke Bola

🛠 *Langkah Praktik:*

1. GameObject → 3D Object → Sphere untuk membuat bola.
2. Tambahkan komponen Rigidbody.
3. Tambahkan Sphere Collider jika belum otomatis.
4. Drag BolaBounciness ke kolom *Material* dalam Sphere Collider.

⚠️ Pastikan:

- Rigidbody aktif dengan Use Gravity dicentang.
- Collider tidak diset sebagai Trigger.

2.6.4 Menyiapkan Lantai sebagai Permukaan Pantul

Untuk melihat efek pantulan, kita butuh permukaan datar.

Langkah-langkah:

1. Tambahkan objek Cube → posisikan sebagai lantai.
2. Skala: X: 10, Y: 0.5, Z: 10
3. Tambahkan **Box Collider** (default ada).

4. (Opsional) Tambahkan Physics Material juga untuk pengaruh gabungan.

2.6.5 Analisis Simulasi Pantulan

Coba jalankan proyek (Play) dan amati perilaku bola. Anda bisa mengubah properti Bounciness dan mengamati hasilnya seperti Tabel 2.5:

Tabel 2. 5 Analisis Simulasi Pantulan

Nilai Bounciness	**Efek Visual**	**Kesimpulan**
0.0	Bola langsung berhenti	Inelastis – tidak ada pantulan
0.3	Pantulan rendah	Energi sebagian hilang
0.7	Pantulan sedang	Umumnya cukup untuk efek realistis
1.0	Pantulan penuh	Sangat elastis – hampir seperti karet

2.6.6 Variasi Eksperimen

1. **Tinggi Jatuh**
   Semakin tinggi bola dijatuhkan, semakin banyak energi kinetik yang dimiliki saat jatuh , efek pantulan meningkat.
2. **Permukaan Pantulan**
   Jika lantai diberi Physics Material yang juga memiliki Bounciness, hasil pantulan = kombinasi keduanya.
3. **Tambah Logging**
   Gunakan skrip untuk mencatat tinggi pantulan:

```
csharp
CopyEdit
void OnCollisionEnter(Collision col)
```

```
{
    Debug.Log("Kecepatan saat menyentuh: " + rb.velocity.y);
}
```

2.6.7 Visualisasi Gaya dan Pantulan

Unity menyediakan cara untuk memvisualisasikan gaya menggunakan Debug.DrawRay:

```
csharp
CopyEdit
void FixedUpdate()
{
    Debug.DrawRay(transform.position, rb.velocity, Color.red);
}
```

2.6.8 Integrasi ke Dalam Game

Contoh penerapan konsep ini dalam game:

- **Game Basket** → pantulan bola dari lantai atau ring.
- **Platformer** → karakter memantul saat menginjak trampolin.
- **Puzzle** → bola harus diarahkan untuk memantul ke target.

Gunakan:

- Collider dan Rigidbody pada objek
- Physics Material pada permukaan
- Logika arah dan momentum dari nilai velocity

Pemantulan bola dalam Unity adalah hasil kerja sama antara:

- **Physics Material untuk properti pantul,**
- **Collider untuk batas interaksi objek,**
- **Rigidbody untuk memungkinkan pengaruh gravitasi dan gaya.**

Studi ini memperkenalkan mahasiswa pada prinsip dasar fisika digital dan pengaruh parameter numerik terhadap perilaku objek di ruang simulasi.

2.7 Studi Kasus 2: Karakter Melompat dan Tabrakan

Studi kasus ini bertujuan membangun simulasi karakter sederhana yang dapat **bergerak dan melompat**, serta **mendeteksi tabrakan** dengan rintangan menggunakan *Rigidbody* dan *Collider*. Konsep ini sangat penting dalam game 3D platformer maupun 2D side-scroller.

2.7.1 Setup Karakter: Rigidbody + Capsule Collider

Unity menyediakan objek dasar bernama **Capsule** yang cocok digunakan sebagai representasi karakter.

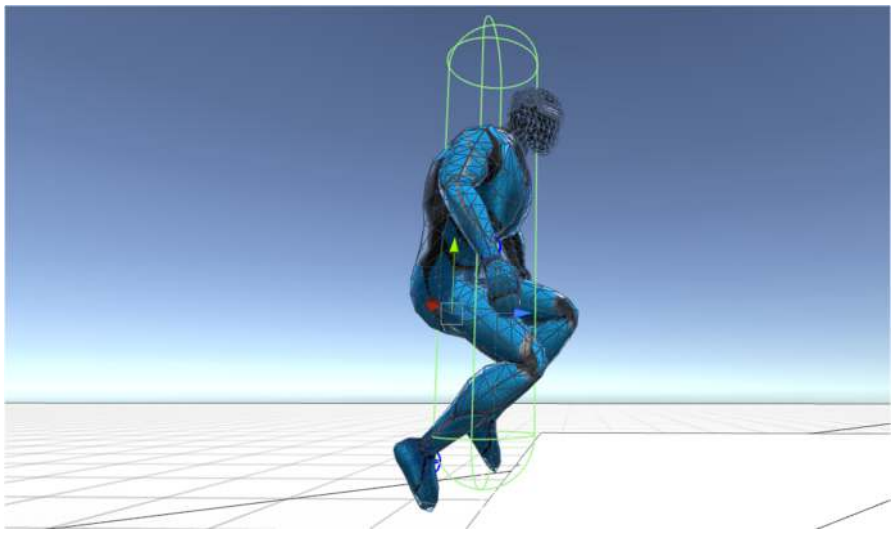
Langkah-langkah Setup:

1. Buat objek karakter:
   GameObject → 3D Object → Capsule
2. Tambahkan komponen:
   - Rigidbody: agar karakter dapat dipengaruhi gravitasi.
   - Capsule Collider: default sudah terpasang.
3. Pastikan Collider tidak diset sebagai trigger.
4. Letakkan Capsule di atas lantai (Cube).

📌 Tips:

- Gunakan Tag Player untuk karakter.
- Tambahkan Ground Tag ke permukaan lantai agar dapat dikenali saat lompat.

Contoh seperti Gambar 2.9:



Gambar 2. 9 Unity capsule player collider

2.7.2 Kode Lompat dan Kontrol Dasar

Untuk mengontrol karakter, kita tambahkan skrip C# bernama PlayerController.cs.

Contoh Skrip Kontrol:

```
csharp
CopyEdit
using UnityEngine;

public class PlayerController : MonoBehaviour
{
    public float speed = 5f;
    public float jumpForce = 7f;
    private Rigidbody rb;
    private bool isGrounded = false;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
    }

    void Update()
    {
        float move = Input.GetAxis("Horizontal") * speed;
        transform.Translate(Vector3.right * move * Time.deltaTime);

        if (Input.GetKeyDown(KeyCode.Space) && isGrounded)
        {
            rb.AddForce(Vector3.up * jumpForce, ForceMode.Impulse);
            isGrounded = false;
```

```
    }
  }

  void OnCollisionEnter(Collision col)
  {
    if (col.gameObject.CompareTag("Ground"))
    {
      isGrounded = true;
    }
  }
}
```

Keterangan:

- Input.GetAxis("Horizontal") → untuk bergerak kiri-kanan.
- Space → untuk melompat.
- isGrounded mencegah karakter lompat berulang di udara.

2.7.3 Deteksi Tabrakan terhadap Rintangan

Untuk mendeteksi tabrakan dengan objek seperti dinding atau musuh, kita gunakan OnCollisionEnter() atau OnTriggerEnter().

Langkah-langkah:

1. Tambahkan Cube sebagai rintangan → posisikan di depan karakter.
2. Tambahkan Box Collider (default).
3. Tandai objek dengan tag "Obstacle".

Tambahan kode deteksi rintangan:

csharp

```
CopyEdit
void OnCollisionEnter(Collision col)
{
```

```
        if (col.gameObject.CompareTag("Obstacle"))
        {
            Debug.Log("Karakter menabrak rintangan!");
            // Tambahkan efek misalnya mengurangi nyawa
        }
    }
```

📌 Tips:

- Gunakan warna/material berbeda pada rintangan.
- Bisa dikembangkan dengan efek suara atau animasi saat tabrakan.

2.7.4 Pengembangan: Deteksi Kematian atau Respawn

Saat karakter menabrak rintangan atau jatuh dari platform, kita bisa membuat sistem "game over" atau respawn.

Contoh skrip tambahan:

```
csharp
CopyEdit
void Update()
{
    if (transform.position.y < -10)
    {
        Debug.Log("Karakter jatuh. Respawn!");
        transform.position = new Vector3(0, 2, 0);
    }
}
```

Tips Praktis:

- Gunakan Animator untuk menambahkan animasi lompat atau tabrak.
- Optimalkan gerakan dengan CharacterController untuk sistem input lanjutan.

Simulasi karakter yang dapat melompat dan mendeteksi tabrakan merupakan fondasi dalam pengembangan game platformer. Elemen penting yang perlu diperhatikan:

- **Gunakan** Capsule Collider **untuk bentuk karakter dasar.**
- **Pastikan kontrol karakter sederhana namun responsif.**
- **Deteksi tabrakan dapat dikembangkan untuk berbagai event seperti pengurangan nyawa, suara efek, atau level restart.**

2.8 Perbandingan Metode Interaksi

Pengembangan game modern mengandalkan interaksi yang efisien antara objek dan lingkungan sekitarnya. Dalam Unity, metode interaksi tersebut dapat dikendalikan melalui pengaturan komponen Rigidbody, Collider, dan scripting. Dalam subbab ini, kita akan membahas tiga aspek penting:

- **Perbandingan Rigidbody vs Kinematic**
- **Efisiensi Collider sederhana vs kompleks**
- **Studi performa pada proyek berskala besar**

2.8.1 Tabel Komparatif Rigidbody vs Kinematic

Unity menyediakan dua pendekatan untuk mengontrol fisika objek:

1. **Rigidbody Dinamis** – mengikuti hukum fisika Unity seperti gravitasi, tumbukan, momentum.
2. **Rigidbody Kinematic** – tidak terpengaruh oleh sistem fisika Unity, tetapi tetap dapat dipindahkan melalui skrip.

Tabel 2. 6 Perbandingan Rigidbody Dinamis vs Kinematic

Aspek	Rigidbody Dinamis	Rigidbody Kinematic
Kontrol oleh Physics	Ya (gravitasi, gaya, tabrakan)	Tidak – hanya via scripting atau animator

Efek Tumbukan	Realistis: memantul, terdorong	Tumbukan hanya terjadi saat objek bergerak
Komputasi	Lebih berat (menggunakan solver fisika Unity)	Ringan (tidak ikut sistem fisika)
Aplikasi Umum	Bola, karakter pemain, objek jatuh	Elevator, musuh pasif, platform
Kode Umum	Rigidbody.useGravity = true	Rigidbody.isKinematic = true

📌 **Tip Praktis**:

Gunakan Rigidbody Dinamis jika objek perlu bereaksi terhadap dunia secara fisik. Gunakan Kinematic jika objek hanya perlu "bergerak secara visual" atau dikendalikan secara manual.

2.8.2 Efisiensi Collider: Sederhana vs Kompleks

Collider adalah batasan tak terlihat yang memungkinkan Unity mendeteksi tabrakan antar objek. Collider dibagi menjadi dua kategori utama:

1. **Collider Sederhana** – Box, Sphere, Capsule
2. **Collider Kompleks** – Mesh Collider, Terrain Collider

Tabel 2. 7 Efisiensi Collider

Jenis Collider	**Akurasi Bentuk**	**Beban Komputasi**	**Contoh Objek**	**Catatan Penting**
Box Collider	Rendah	Rendah	Dinding, peti	Sangat ringan dan cepat
Sphere Collider	Sedang	Rendah	Bola, batu bulat	Cocok untuk simetri

Capsule Collider	Sedang	Rendah–Sedang	Karakter humanoid	Efisien untuk karakter
Mesh Collider	Tinggi	Tinggi	Objek organik, mobil	Gunakan hanya saat diperlukan
Terrain Collider	Khusus	Berat jika besar	Lanskap, gunung	Hanya untuk objek Terrain Unity

🔍 **Prinsip Efisiensi:**

- Collider sederhana **lebih ringan** dan **lebih cepat diproses**.
- Collider kompleks digunakan hanya saat interaksi presisi sangat dibutuhkan.

2.8.3 Studi Grafik Performa dalam Proyek Skala Besar

Simulasi fisika dan sistem interaksi menjadi salah satu **penyumbang terbesar konsumsi CPU** di proyek skala besar. Unity menyediakan **Profiler** untuk memantau performa runtime dari Rigidbody, Collider, dan tabrakan.

Contoh Studi Kasus:

Skenario:

- 200 objek menggunakan Mesh Collider dan Rigidbody Dinamis.
- Kamera menyorot seluruh area.
- Simulasi berjalan 10 menit.

Tabel 2. 8 Hasil Pengamatan di Unity Profiler

Parameter	**Nilai Awal**	**Nilai Setelah Optimasi**
CPU Usage (Physics)	35%	12%

FPS	28	60
Collision Events per Second	1200	340
Rigidbody Active per Frame	200	80

📈 Penurunan beban sistem sebesar ~66% didapatkan dengan:

- Mengganti sebagian Mesh Collider dengan Box Collider.
- Mengatur Rigidbody.isKinematic = true pada objek statis.
- Menghapus Rigidbody pada objek yang tidak memerlukan reaksi fisika.

💡 Tips Implementasi:

- Profil proyek Anda secara rutin dengan Unity Profiler.
- Batasi jumlah Rigidbody aktif secara bersamaan.
- Gunakan Collider sederhana secara maksimal.
- Hindari penggunaan Mesh Collider tanpa Convex.

Efisiensi dan logika interaksi dalam game sangat dipengaruhi oleh pemilihan metode fisika dan jenis collider. Dalam proyek kompleks, strategi yang bijak terhadap pemakaian Rigidbody **dan** Collider **dapat meningkatkan FPS dan menstabilkan simulasi secara signifikan.**

2.9 Latihan Proyek Mini

🎯 Tujuan Pembelajaran

Pada bagian ini, mahasiswa akan mengembangkan mini-game berbasis fisika di Unity yang memanfaatkan konsep:

- **Benturan antar objek** (collision),
- **Deteksi skor dan interaksi sederhana**,
- **Kontrol karakter dan pengumpulan objek**, dan
- **Elemen User Interface (UI)** seperti skor dan timer.

2.9.1 Mendesain Mini-Game: "Kumpulkan Koin"

Deskripsi Proyek:

Pemain mengendalikan karakter (misalnya bola atau kotak) untuk **mengumpulkan koin** sambil **menghindari rintangan**. Pemain mendapat poin setiap kali menyentuh koin. Proyek ini menggunakan Rigidbody dan Collider untuk interaksi objek.

Komponen Inti:

- **Player:** GameObject dengan Rigidbody dan Collider (Sphere/Box).
- **Koin:** Objek dengan Collider + tag "Collectible".
- **Rintangan:** Objek statis atau bergerak dengan Collider.
- **Skor dan Timer:** Ditampilkan melalui UI Text di Canvas.

Langkah-Langkah Pembuatan:

1. **Membuat Player**
    - Tambahkan GameObject → 3D Object → Sphere
    - Tambahkan Rigidbody dan Sphere Collider.
    - Tambahkan skrip PlayerController.cs:

      csharp

      CopyEdit

```
void Update()
    {
      float moveX = Input.GetAxis("Horizontal") * speed * Time.deltaTime;
      float moveZ = Input.GetAxis("Vertical") * speed * Time.deltaTime;
      transform.Translate(new Vector3(moveX, 0, moveZ));
    }
```

2. **Menambahkan Koin**
    - 3D Object → Cylinder (scale kecil).
    - Tambahkan tag "Collectible".
    - Beri Material warna emas.

- Tambahkan Collider (Is Trigger = true).

3. **Menambahkan Rintangan**
   - 3D Object → Cube.
   - Beri Collider, posisi acak, dan tambahkan Rigidbody jika ingin bergerak.
4. **Skrip Deteksi Koin**

```
csharp
CopyEdit
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Collectible"))
    {
        score++;
        Destroy(other.gameObject);
            }
}
```


2.9.2 Desain UI: Skor, Suara, dan Timer

A. Canvas Setup

- GameObject → UI → Canvas
- Tambahkan UI Text untuk skor dan waktu
- Tambahkan panel UI (optional) untuk info visual

B. Skrip Timer

```
csharp
CopyEdit
    float timeLeft = 60f;
    void Update()
    {
        timeLeft -= Time.deltaTime;
        timeText.text = "Time: " + Mathf.Round(timeLeft);
```

}

C. Efek Suara

- *Tambahkan AudioSource di Player.*
- *Siapkan AudioClip "coin_collect.mp3".*
- *Mainkan saat koin diambil:*

csharp

CopyEdit

audioSource.PlayOneShot(coinSound);

2.9.3 Pengembangan Lanjutan

- Tambahkan fitur nyawa pemain.
- Koin memiliki skor berbeda-beda.
- Level up saat semua koin dikumpulkan.
- Tambahkan sistem restart saat waktu habis.

Tips:

- Selalu gunakan Time.deltaTime untuk gerakan yang stabil.
- Pastikan Collider koin memiliki **Is Trigger = true** agar tidak menghambat gerakan player.
- Simpan prefab koin dan rintangan agar mudah diatur ulang.

Latihan proyek mini ini memperkuat keterampilan dasar Unity:

- **Pemanfaatan Rigidbody dan Collider.**
- **Penerapan Trigger untuk pengumpulan item.**
- **Penggunaan UI untuk menciptakan pengalaman interaktif.**

2.10 Evaluasi dan Refleksi

Bagian ini bertujuan mengukur sejauh mana mahasiswa memahami konsep dan penerapan sistem fisika dalam Unity melalui soal evaluasi berbasis kasus, serta memberi ruang refleksi untuk menilai pencapaian belajar secara mandiri.

2.10.1 Soal Evaluasi Berbasis Kasus

A. Pilihan Ganda

1. Komponen Rigidbody di Unity digunakan untuk:
   - A. Mengatur tekstur objek
   - B. Mengatur posisi kamera
   - C. Memberikan kemampuan fisika pada objek ☑
   - D. Mengatur rotasi manual
2. Collider mana yang paling tepat untuk karakter berbentuk silinder tegak seperti pemain?
   - A. Box Collider
   - B. Sphere Collider
   - C. Capsule Collider ☑
   - D. Mesh Collider
3. Fungsi dari Rigidbody dengan opsi *isKinematic* = *true* adalah:
   - A. Menonaktifkan Collider
   - B. Objek mengikuti logika fisika
   - C. Objek tidak terpengaruh oleh fisika Unity ☑
   - D. Objek selalu mengambang

B. Soal Esai Pendek

1. Jelaskan perbedaan mendasar antara **Trigger** dan **Collision** di Unity.
2. Berikan skenario game sederhana di mana lebih baik menggunakan **Rigidbody Kinematic** dibandingkan **Rigidbody Dinamis**.
3. Mengapa **Collider sederhana** lebih dianjurkan dalam proyek berskala besar?

C. Studi Kasus

Skenario:

Anda diminta membuat game edukatif sederhana di mana bola harus mengumpulkan 10 koin dalam waktu 60 detik sambil menghindari rintangan yang bergerak.

Pertanyaan:

- a. Komponen apa saja yang akan Anda gunakan untuk objek bola, koin, dan rintangan?
- b. Bagaimana Anda menerapkan sistem skor dan waktu?
- c. Sebutkan langkah optimasi performa jika game ini mengalami lag di perangkat low-end.

2.10.2 Refleksi Proyek Mini

✍ *Pertanyaan Refleksi Pribadi:*

1. Apa tantangan terbesar yang Anda temui saat membangun mini-game fisika?
2. Konsep apa yang menurut Anda masih sulit dipahami dan perlu latihan tambahan?
3. Bagaimana peran *trial-and-error* dalam membangun sistem interaksi di Unity?
4. Apa keterampilan teknis baru yang Anda kuasai selama proyek ini?

Instruksi Refleksi:

- Tuliskan jawaban Anda dalam ½ halaman (maksimal 200 kata).
- Sertakan tangkapan layar proyek atau bagian kode yang paling berkesan.

Bab 3 Pergerakan Kamera dan Obstacle Path

3.1 Tujuan Pembelajaran

Dalam pengembangan gim 3D dan simulasi visual interaktif, pergerakan kamera memainkan peranan krusial dalam menciptakan pengalaman pengguna yang imersif. Kamera bukan hanya sebagai "mata" pemain, melainkan juga sebagai alat naratif yang mengarahkan perhatian, membangun tensi, dan menyampaikan informasi penting dalam sebuah permainan atau simulasi.

Subbab ini bertujuan untuk membekali mahasiswa dengan pemahaman dan keterampilan teknis dalam mengelola berbagai bentuk pengaturan kamera serta bagaimana pergerakan kamera dapat dikombinasikan dengan objek dan lingkungan secara dinamis.

Capaian Pembelajaran

Setelah menyelesaikan bab ini, mahasiswa diharapkan mampu:

- **Memahami konsep dasar pengaturan kamera** dalam Unity3D, baik dari sisi teknis maupun peran sinematiknya dalam pengalaman bermain.
- **Menerapkan berbagai jenis kamera**, seperti kamera statis untuk lingkungan eksploratif, serta kamera dinamis yang mengikuti pemain atau objek utama.
- **Mengintegrasikan kontrol kamera dengan perangkat input** seperti mouse atau trackpad untuk menciptakan navigasi interaktif yang responsif.
- **Menggabungkan logika pergerakan objek dan kamera** untuk menghasilkan tampilan visual yang mendukung gameplay atau simulasi.

Ilustrasi Penggunaan Kamera

Berikut ilustrasi konsep penggunaan kamera dalam Unity:

Ilustrasi Unity Kamera dan Pemain – Sumber

Gambar ini menunjukkan implementasi skrip kamera yang mengikuti objek pemain, yang akan dibahas dalam subbab 3.6.

Tips Penggunaan Kamera dalam Game:

- Gunakan **kamera statis** saat menampilkan tampilan utama (misalnya: menu awal).
- Gunakan **kamera mengikuti pemain** untuk permainan berorientasi aksi atau petualangan.
- Gunakan **kamera bebas/interaktif** saat menjelajah area terbuka atau saat pemain membutuhkan kebebasan navigasi tinggi.

Contoh Game Nyata

Tabel 3. 1 Contoh Game yang ada

Game	Jenis Kamera	Keterangan
Super Mario 3D	Kamera mengikuti	Kamera mengikuti karakter saat bergerak di dunia 3D
The Sims	Kamera interaktif	Pemain bebas mengatur sudut pandang dan zoom
Angry Birds	Kamera statis	Hanya berpindah saat berpindah level atau fokus

Rangkuman Subbab 3.1

Penguasaan konsep kamera bukan hanya teknis tetapi juga strategis. Kamera yang efektif mendukung alur permainan dan memperkuat pengalaman pemain. Pada bab selanjutnya, kita akan membahas konsep **Path Obstacle** yang berperan penting dalam menghadirkan tantangan rintangan dan jalur gerak.

3.2 Konsep Path Obstacle

3.2.1 Definisi dan Kegunaan Path Obstacle

Dalam konteks pemrograman grafis interaktif dan pengembangan gim berbasis Unity3D, istilah **Path Obstacle** mengacu pada jalur yang dipenuhi oleh rintangan atau penghalang (obstacle) yang dirancang secara sistematis untuk mengarahkan pergerakan karakter atau objek dalam lingkungan permainan. Path obstacle bukan hanya elemen

visual, tetapi juga merupakan entitas logis yang memiliki fungsi penting dalam membentuk tantangan, navigasi, dan strategi gameplay.

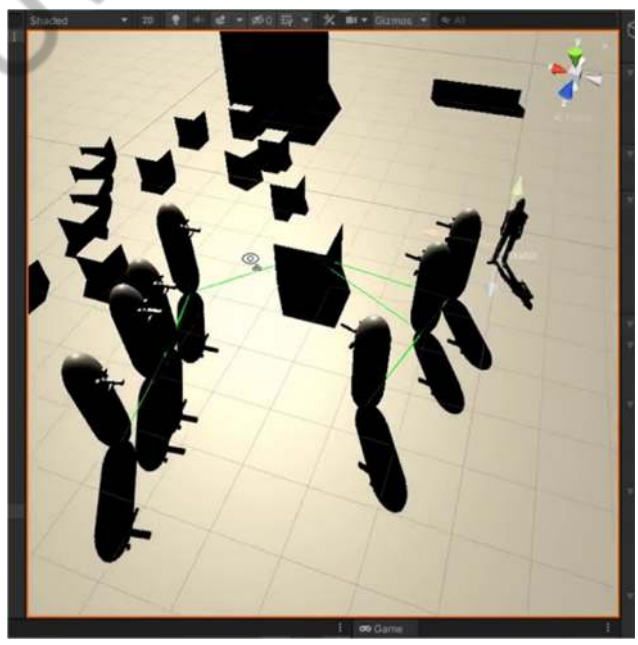
Rintangan dapat berupa objek statis (misalnya tembok, lubang, atau portal), maupun dinamis (seperti bola berputar, dinding yang bergerak, atau laser yang aktif secara periodik). Ketika obstacle ini diletakkan dalam formasi tertentu membentuk lintasan, maka muncullah konsep "Path Obstacle".

Manfaat Penggunaan Path Obstacle

- **Mengontrol gerakan pemain:** Pemain tidak dapat bergerak secara bebas tanpa mempertimbangkan posisi rintangan.
- **Membentuk tantangan gameplay:** Rintangan menjadi elemen penentu tingkat kesulitan permainan.
- **Mengaktifkan elemen strategi:** Pemain harus memilih jalur paling efisien atau aman.
- **Menambah realisme fisika dan animasi:** Dengan obstacle yang memiliki reaksi fisik, efek visual akan tampak lebih alami.

3.2.2 Representasi Visual Path Obstacle

Berikut ini contoh visual lintasan dengan obstacle yang dinamis, Gambar 3.1:



Gambar 3. 1 Jalur dengan obstacle bergerak (misalnya capsule atau kotak yang bergeser)

3.2.3 Jalur Otomatis dan Responsif

1. Jalur Otomatis (Predefined Pathing)

Path obstacle sering digunakan dalam permainan berbasis rail atau auto-runner, di mana objek pemain (player) bergerak secara otomatis mengikuti jalur tertentu. Gerakan karakter dikendalikan oleh sistem, sedangkan pengguna hanya menghindari rintangan atau melakukan aksi tertentu.

Contoh Penggunaan:

- Karakter mengikuti waypoint dari titik A ke B.
- Mobil balap yang melaju di lintasan sirkuit.

Contoh kode untuk gerakan ke waypoint:

csharp

CopyEdit

*transform.position = Vector3.MoveTowards(transform.position, target.position, speed * Time.deltaTime);*

2. Jalur Responsif (Dynamic Pathing)

Di sini, objek tidak hanya mengikuti jalur tetap, tetapi juga merespons kondisi dinamis di lingkungan, seperti munculnya obstacle baru, perintah AI, atau masukan pemain.

Penggunaan:

- Musuh yang mencari rute terdekat menghindari rintangan.
- Robot cerdas yang merespons posisi pengguna dalam simulasi.

Dukungan teknologi:

- **NavMesh Agent** pada Unity
- *A Pathfinding Project** (untuk AI)
- Sistem collider dinamis

3.2.4 Kolaborasi Obstacle dengan Animasi dan Fisika

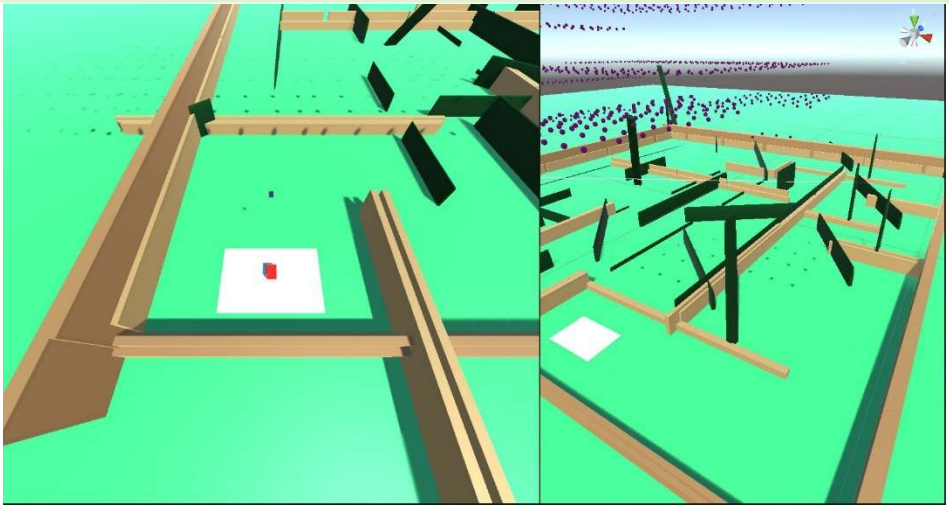
Obstacle dalam game modern tidak hanya diam, tetapi bisa **bergerak**, **berinteraksi dengan gaya fisika**, atau **terprogram dengan logika animator**. Unity menyediakan integrasi antara animator dan komponen fisika (Rigidbody, Collider, Trigger) untuk menciptakan efek visual dan logika gameplay yang realistis, seperti Gambar 3.2.

Tabel 3. 2 Kombinasi Komponen pada Obstacle

Komponen	Fungsi
Animator	Menggerakkan obstacle berdasarkan timeline
Collider (isTrigger)	Menentukan batas aktif obstacle secara non-fisik
Rigidbody (Kinematic)	Obstacle mengikuti skrip tanpa gaya fisika
Script (C#)	Logika gerakan dan interaksi

Contoh Konsep Implementasi:

- Bola berputar yang bergerak horizontal.
- Platform naik-turun mengikuti animasi.
- Laser aktif setiap beberapa detik berdasarkan timer.

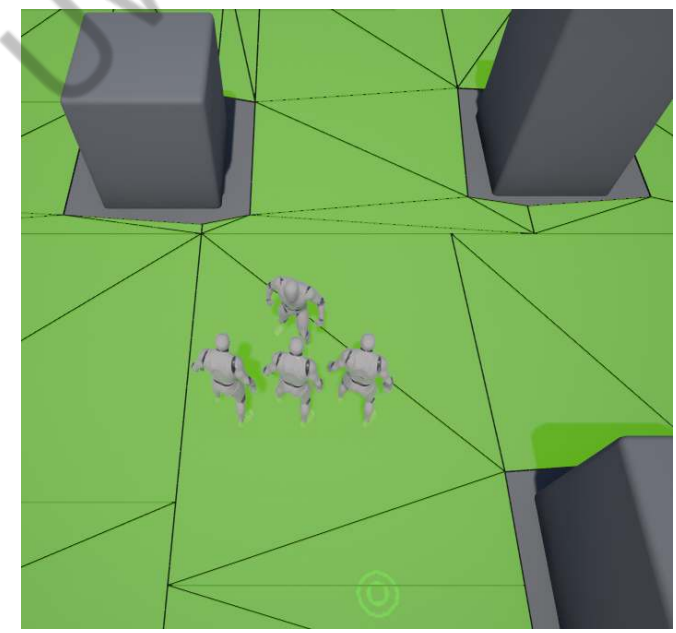


Gambar 3. 2 Ilustrasi obstacle animasi dengan collider

3.2.5 Path Obstacle dan AI Navigasi

Dalam proyek skala besar atau game open-world, obstacle tidak hanya ditujukan untuk pemain, tetapi juga **karakter AI (non-playable character)** yang memiliki tujuan bergerak melalui jalur tertentu. Unity mendukung ini melalui sistem **NavMesh** yang memungkinkan AI untuk secara otomatis menghindari obstacle saat bergerak menuju target.

Ilustrasi NavMesh vs Obstacle:



Gambar 3. 3 representasi visual dari sistem navigasi AI dalam lingkungan game 3D

Gambar 3.3 menunjukkan **representasi visual dari sistem navigasi AI** dalam lingkungan game 3D, kemungkinan besar dari **Unreal Engine** atau lingkungan sejenis yang menggunakan **NavMesh (Navigation Mesh)**.

Detail penjelasan Gambar 3.3, adalah :

1. Warna Hijau Transparan

- Area berwarna **hijau terang** menunjukkan ***Navigable Area*** atau **zona yang bisa dilalui oleh karakter AI**.
- Ini adalah hasil dari **NavMesh baking**, di mana engine menghitung area mana yang bisa digunakan karakter untuk bergerak secara otomatis.
- Area ini **dihasilkan berdasarkan geometri lantai dan obstacle** di sekitarnya.

2. Objek Abu-abu (Obstacle)

- Objek besar berbentuk balok **tidak memiliki permukaan hijau** — artinya mereka dikenali sebagai **rintangan atau obstacle**.
- Area ini akan dihindari oleh karakter saat menggunakan sistem AI navigation.

3. Karakter Humanoid

- Terdapat **beberapa karakter humanoid**, kemungkinan besar adalah **NPC (Non-Playable Character)** atau agen AI.
- Mereka ditempatkan di atas NavMesh dan akan menggunakan area hijau untuk **bergerak dari satu titik ke titik lain**.
- Pergerakan mereka akan **otomatis menghindari obstacle** (balok abu-abu).

4. Garis Hitam (Segmen NavMesh)

- Garis-garis membentuk **pembagian segitiga** dari area navigasi.
- Ini adalah **mesh polygon** yang dihitung oleh sistem untuk menentukan **rute tercepat** dan area pergerakan.
- Pembagian ini penting agar AI tahu **bagaimana menghindari sudut, celah sempit, dan sudut kompleks.**

5. Posisi & Orientasi Karakter

- Karakter di tengah menghadap ke atas gambar (utara dalam konteks tampilan).
- Kemungkinan besar mereka akan **berjalan menuju target tertentu**, atau sudah disiapkan untuk memulai simulasi AI Movement.

- **Path Obstacle** adalah konsep penting dalam desain level yang mengatur bagaimana karakter bergerak di dunia permainan.
- Obstacle dapat **statik atau dinamis**, serta **otomatis atau responsif**.
- Integrasi dengan komponen Unity seperti **Rigidbody, Collider, Animator**, dan **NavMesh** memberikan fleksibilitas tinggi dalam menciptakan lingkungan yang hidup dan penuh tantangan.
- Pemahaman ini penting sebelum melangkah ke pengaturan kamera yang akan menyesuaikan sudut pandang terhadap jalur dan interaksi objek.

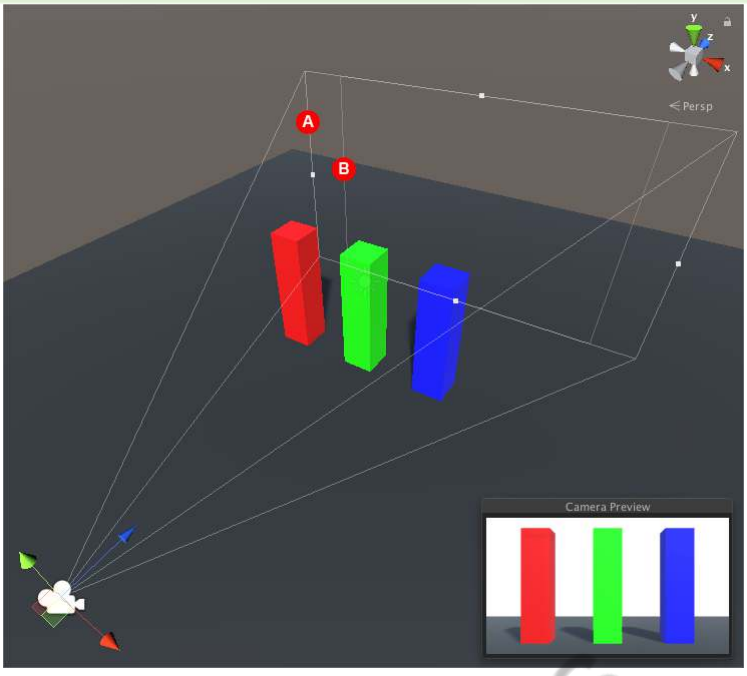
3.3 Pengaturan Kamera dalam Unity

3.3.1 Fungsi Kamera dalam Unity

Dalam Unity, kamera berperan sebagai jendela utama pemain untuk melihat dunia game. Tanpa kamera, tidak akan ada tampilan yang bisa dilihat pengguna. Kamera menangkap seluruh peristiwa dalam scene dan menampilkannya di layar.

Unity secara default menyediakan satu kamera utama bernama **Main Camera**. Kamera ini dapat dikustomisasi secara visual maupun melalui skrip.

"Kamera dalam game adalah seperti lensa sutradara dalam film — mengarahkan apa yang dilihat pemain."



Gambar 3. 4 Contoh tampilan Camera dalam Unity

Gambar 3.4, merupakan tampilan camera dalam unity dengan detail sebagai berikut :

A. Near Clipping Plane

- Garis batas terdekat dari kamera. Objek yang berada lebih dekat dari garis ini **tidak akan ditampilkan** dalam tampilan kamera.
- Dalam gambar, titik A menunjukkan batas depan dari area pandang kamera.
- Jika objek melintasi batas ini, objek akan "hilang" dari tampilan.

B. Far Clipping Plane

- Garis batas terjauh dari kamera. Objek yang berada lebih jauh dari garis ini juga **tidak akan ditampilkan**.
- Titik B menunjukkan batas akhir dari jarak pandang kamera.
- Ini penting untuk mengoptimalkan performa—objek yang terlalu jauh dan tidak perlu dilihat dapat disembunyikan dari rendering.

Visualisasi Objek

- Tiga balok **merah**, **hijau**, dan **biru** adalah objek yang terlihat oleh kamera karena berada **di antara Near dan Far Clipping Plane**.

- Tampilan mini di kanan bawah menunjukkan **Camera Preview**, yaitu tampilan yang akan dilihat oleh pemain selama permainan.

Konteks Penggunaan

- Visualisasi ini sangat penting saat mengatur kamera untuk memastikan semua objek penting terlihat oleh pemain.
- Kombinasi dari pengaturan **FoV**, **Near Plane**, dan **Far Plane** menentukan seberapa luas dan dalam scene yang bisa dilihat.
- Sangat berpengaruh pada genre game seperti:
  - **FPS/TPS**: Perlu FoV yang dinamis.
  - **Strategi**: Membutuhkan kamera ortografik.
  - **Simulasi**: Membutuhkan penyesuaian clipping agar tidak memotong objek besar.

3.3.2 Mode Kamera dalam Unity

Ada tiga mode utama dalam penerapan kamera di Unity:

a. Kamera Statis

Kamera statis tetap diam di posisi tertentu dan tidak mengikuti objek apa pun. Mode ini cocok untuk tampilan level tetap seperti game puzzle atau tower defense.

b. Kamera Mengikuti Objek

Kamera mengikuti pergerakan karakter atau objek utama, memungkinkan pengguna mendapatkan pandangan dinamis.

c. Kamera Interaktif (Diarahkan Pengguna)

Kamera dikendalikan oleh input pemain seperti mouse atau joystick, misalnya dengan fitur zoom, drag, atau rotasi (orbit). Biasanya digunakan dalam game strategi atau simulasi.

3.3.3 Komponen Utama Kamera

Berikut adalah komponen utama kamera yang dapat diatur melalui Inspector:

a. Field of View (FoV)

Sudut pandang kamera dalam derajat. Nilai default 60° cocok untuk banyak genre game. Nilai yang lebih besar memberikan efek fish-eye, sedangkan nilai kecil memberi efek zoom.

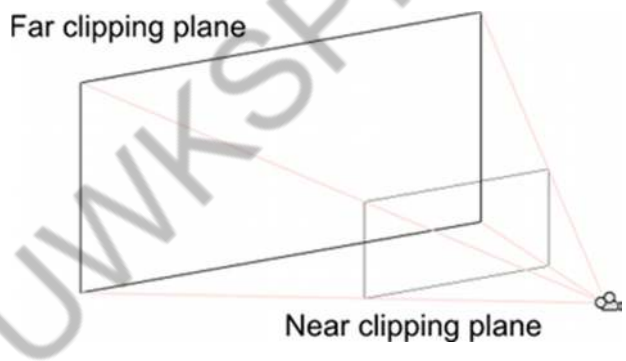
b. Clipping Planes

Menentukan jarak minimum dan maksimum objek yang dapat dilihat kamera:

- **Near Clipping Plane**: Objek lebih dekat dari nilai ini tidak akan ditampilkan.
- **Far Clipping Plane**: Objek lebih jauh dari nilai ini tidak ditampilkan.

c. Perspective vs Orthographic

- **Perspective**: Tampilan 3D realistis dengan kedalaman.
- **Orthographic**: Tampilan 2D tanpa perspektif, cocok untuk game 2D atau desain teknis.



Gambar 3. 5 ilustrasi mendasar dari konsep Clipping Planes dalam kamera 3D

Gambar 3.5 merupakan ilustrasi mendasar dari konsep **Clipping Planes** dalam kamera 3D, terutama digunakan dalam perangkat lunak seperti **Unity3D** dan sistem grafika 3D lainnya. Mari kita uraikan bagian-bagian utamanya:

Kamera (ikon di kanan bawah)

- Menunjukkan **posisi dan arah pandang kamera**.
- Kamera ini menjadi pusat proyeksi tampilan — semua yang dilihat pemain berasal dari sudut pandang ini.

Near Clipping Plane

- Merupakan **batas minimum** dari pandangan kamera.
- Objek yang berada **lebih dekat dari plane ini tidak akan terlihat** dalam scene.
- Biasanya disetel dengan nilai kecil (misal 0.1 atau 0.3) agar objek dekat tetap dapat terlihat namun menghindari masalah grafis seperti "clipping z-fighting".
- Dalam game FPS (First-Person Shooter), pengaturan yang tepat dari near clipping plane sangat penting agar tangan atau senjata pemain tidak tiba-tiba menghilang.

Far Clipping Plane

- Merupakan **batas maksimum** dari pandangan kamera.
- Objek yang berada **lebih jauh dari plane ini juga tidak akan dirender**.
- Disetel dengan nilai besar (misal 1000 atau lebih) tergantung skala game.
- Berguna untuk **mengoptimalkan performa**, karena Unity hanya merender objek yang berada di antara near dan far clipping.

Frustum (volume pandang)

- Bentuk seperti piramida terpotong ini adalah **ruang pandang kamera**.
- Hanya objek yang berada **di dalam frustum** yang akan dirender dan terlihat oleh pemain.
- Frustum dibatasi oleh near clipping plane di depan dan far clipping plane di belakang, serta bidang proyeksi di sisi kanan-kiri dan atas-bawah.

Aplikasi dalam Unity

Dalam Unity, Anda bisa mengatur kedua nilai ini melalui **Inspector > Camera > Clipping Planes**:

- Near: default-nya 0.3
- Far: default-nya 1000

Pengaturan ini penting untuk:

- **Efisiensi rendering** (semakin kecil jarak antara near dan far, semakin baik presisi depth buffer-nya).
- **Menghindari kehilangan objek penting** karena berada di luar batas pandang.

Tabel 3. 3 Ringkasan kamera view unity3d

Komponen	**Fungsi**
Near Clipping Plane	Batas depan tampilan kamera, objek terlalu dekat disembunyikan
Far Clipping Plane	Batas belakang tampilan kamera, objek terlalu jauh disembunyikan
Frustum	Ruang 3D di mana semua objek dapat dirender dan dilihat

3.3.4 Pengaturan Kamera Melalui Script

Contoh script kamera mengikuti pemain:

```
csharp
CopyEdit
public class FollowPlayer : MonoBehaviour
{
    public Transform player;
    public Vector3 offset;

    void Update()
    {
        transform.position = player.position + offset;
    }
```

```
}
```

Script ini membuat kamera mengikuti posisi pemain secara dinamis dengan jarak yang konstan.

Pengaturan kamera sangat penting dalam Unity karena secara langsung mempengaruhi persepsi visual pemain terhadap dunia game. Memilih mode kamera yang tepat dan mengatur komponen-komponennya secara cermat akan menciptakan pengalaman bermain yang lebih imersif.

3.4 Scene View dan Alat Visual

Dalam pengembangan game menggunakan Unity3D, **Scene View** adalah area penting yang memungkinkan pengembang melihat dan mengatur elemen-elemen dalam dunia virtual. Melalui Scene View, pengguna dapat memanipulasi objek secara visual dengan berbagai alat bantu yang tersedia, seperti Move Tool, Rotate Tool, dan Animation Tool. Bagian ini akan membahas secara mendalam mengenai alat-alat tersebut dan bagaimana penggunaannya dalam proses pembuatan game.

3.4.1 Pengenalan Scene View

Scene View adalah antarmuka grafis di Unity yang menampilkan dunia 3D tempat pengguna dapat menambahkan, menghapus, dan mengatur objek dalam game. Scene View memberikan tampilan yang berbeda dari **Game View**, karena Scene View tidak menampilkan apa yang pemain lihat saat bermain, melainkan tampilan bebas yang bisa dikontrol sepenuhnya oleh pengembang.

Fitur penting Scene View:

- **Navigasi bebas** menggunakan mouse dan keyboard.
- **Manipulasi objek** dengan alat bantu visual.
- **Preview lighting, animasi, dan interaksi objek** secara real-time.

3.4.2 Move Tool

Move Tool digunakan untuk memindahkan objek di dalam Scene View. Tool ini akan menampilkan **gizmo panah tiga arah (X, Y, Z)** pada objek yang dipilih, yang bisa diseret untuk memindahkan objek sesuai arah sumbu tertentu.

Fitur Move Tool:

- Klik dan tarik sumbu untuk memindahkan objek dalam satu arah.
- Seret di antara dua sumbu untuk menggeser dalam bidang 2D.
- Koordinat dapat diatur ke **local** atau **global** sesuai kebutuhan.

3.4.3 Rotate Tool

Rotate Tool digunakan untuk memutar objek di Scene View. Saat Rotate Tool diaktifkan, akan muncul **gizmo lingkaran** pada objek, yang dapat diputar sesuai arah sumbu X, Y, atau Z.

Fungsi utama Rotate Tool:

- Memutar objek terhadap titik pusatnya.
- Memungkinkan rotasi presisi dengan angka rotasi pada Inspector.
- Sangat penting untuk pengaturan orientasi kamera, karakter, atau elemen environment.

3.4.4 Animation Tool

Animation Tool digunakan untuk mengatur dan membuat animasi dalam Unity. Tool ini terhubung dengan **Animator** dan **Timeline**, memungkinkan pengguna membuat animasi transisi, gerakan objek, perubahan skala, dan banyak lagi.

Fitur utama:

- Menambahkan keyframe pada timeline untuk parameter seperti **position**, **rotation**, dan **scale**.
- Memungkinkan **preview animasi** secara langsung di Scene View.
- Dapat digunakan untuk membuat **cutscene**, **animasi interaktif**, dan **gerakan karakter**.

3.4.5 Perbandingan Ketiga Tool

Tabel 3. 4 Perbandingan fungsi Tool

Tool	Fungsi Utama	Shortcut	Visual Gizmo
Move Tool	Memindahkan objek	W	Panah tiga arah
Rotate Tool	Memutar objek pada sumbu X,Y,Z	E	Lingkaran rotasi
Animation Tool	Membuat animasi keyframe	Ctrl+6	Timeline/graph editor

3.4.6 Studi Kasus

Contoh Implementasi:

Dalam pengembangan game 3D "Maze Runner", pengembang menggunakan:

- **Move Tool** untuk menyusun dinding labirin.
- **Rotate Tool** untuk mengatur sudut kamera statis.
- **Animation Tool** untuk membuka pintu secara otomatis saat pemain mendekat.

Hasilnya: Kombinasi alat ini mempercepat proses desain dan memberikan feedback visual yang jelas.

Alat visual di Scene View sangat penting bagi pengembang untuk menavigasi dan mengontrol elemen game secara intuitif. Penggunaan Move Tool, Rotate Tool, dan Animation Tool mempercepat proses desain, memungkinkan animasi yang kompleks, dan meningkatkan kualitas kontrol atas dunia game.

3.5 Kasus 1: Kamera Statis

3.5.1 Pendahuluan

Dalam pengembangan gim berbasis Unity3D, pengaturan kamera merupakan salah satu aspek fundamental yang menentukan kualitas pengalaman visual pemain. Kamera statis

adalah jenis kamera yang tidak berubah posisi atau arah pandang selama permainan berlangsung. Kamera ini umumnya digunakan dalam adegan dengan sudut pandang tetap, seperti permainan teka-teki, game 2D, atau adegan cutscene.

3.5.2 Tujuan Pembelajaran

- Mampu menyusun sistem kamera statis dalam Unity.
- Menentukan sudut pandang yang efektif untuk gameplay.
- Menyesuaikan parameter kamera untuk hasil visual optimal.

3.5.3 Pengenalan Kamera Statis

Kamera statis diatur untuk tetap berada pada posisi dan rotasi tertentu tanpa mengikuti objek di dalam gim. Kamera jenis ini sangat cocok untuk permainan dengan:

- Tata letak ruangan tetap.
- Puzzle room.
- Arena duel dengan satu perspektif.

3.5.4 Langkah-Langkah Pengaturan Kamera Statis

Langkah 1: Menambahkan Kamera ke Scene

Secara default, setiap scene baru di Unity sudah memiliki satu kamera utama (Main Camera). Anda juga dapat menambahkan kamera baru dari menu:

nginx

CopyEdit

GameObject → Camera

Langkah 2: Menentukan Posisi dan Rotasi Kamera

Gunakan **Scene View** untuk mengatur sudut pandang kamera:

- Tool: *Move Tool* dan *Rotate Tool.*
- Shortcut: W untuk pindah, E untuk rotasi.

Gunakan kombinasi *Gizmos* dan Camera Preview untuk melihat hasil real-time dari sudut pandang kamera.

Langkah 3: Menyesuaikan Parameter Kamera

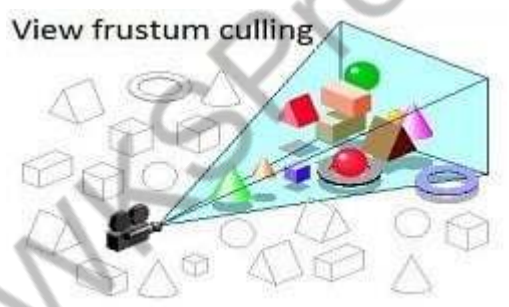
Pada komponen kamera di Inspector:

- **Clear Flags**: Skybox / Solid Color.
- **Field of View**: Atur sudut pandang, misalnya 60°.
- **Clipping Planes**: Sesuaikan near dan far sesuai dengan skala dunia gim.

3.5.5 Pratinjau Kamera

Setelah konfigurasi selesai, klik tombol **Play** untuk meninjau tampilan kamera. Gunakan **Panel Game View** untuk melihat hasil sesuai yang akan ditampilkan kepada pemain.

Contoh Tampilan Pratinjau Kamera, dengan konsep view frustum, seperti Gambar 3.6.



Gambar 3. 6 Ilustrasi tampilan pratinjau kamera

View Frustum adalah **volume piramida terpotong (truncated pyramid)** yang mewakili ruang pandang dari kamera di dalam dunia 3D. Objek yang berada **di dalam frustum** inilah yang dapat dilihat oleh kamera dan akan dirender di layar.

Gambar menunjukkan:

- Kamera berada di sebelah kiri bawah.
- Piramida transparan berwarna biru adalah **frustum**.
- Berbagai bentuk objek (bola, kotak, kerucut, torus) berada di dalam dan luar frustum.

Culling: Menyaring Objek

Culling dalam konteks ini berarti **menghilangkan dari proses render** objek-objek yang berada **di luar frustum** karena objek tersebut **tidak terlihat oleh kamera**.

Fungsinya:

- Menghemat **kinerja GPU**.
- Meningkatkan **frame rate** karena hanya objek dalam frustum yang di-render.

⚙ Implementasi dalam Unity

Dalam Unity, sistem ini bekerja secara otomatis, disebut **frustum culling**:

- Tidak memerlukan konfigurasi manual.
- Hanya objek dalam **visible area kamera** yang akan dihitung untuk render.
- Sistem dapat dikombinasikan dengan **Occlusion Culling** untuk efisiensi lebih tinggi (mengabaikan objek yang tertutup oleh objek lain).

🎮 Manfaat View Frustum Culling

Tabel 3. 5 Manfaat View Frustum Culling

Aspek	Manfaat
Performa	Meningkatkan performa dengan mengurangi beban render GPU.
Efisiensi	Menjaga render tetap fokus pada elemen penting dalam pandangan kamera.
Optimasi Proyek	Sangat penting pada game skala besar dengan banyak objek 3D.

3.5.6 Studi Kasus Sederhana

Kasus: Permainan puzzle dengan objek bola yang digerakkan oleh keyboard.

- Kamera dipasang dari atas (top-down).
- Tidak mengikuti objek.

- Hanya satu titik pandang.

csharp

CopyEdit

// Tidak diperlukan script kamera, cukup menggunakan GameObject Camera.

Keuntungan:

- Tidak membebani performa.
- Konsistensi sudut pandang.
- Cocok untuk gameplay strategi atau taktik.

3.5.7 Kelebihan dan Kekurangan Kamera Statis

Tabel 3. 6 Kelebihan dan Kekurangan Kamera Statis

Aspek	**Kelebihan**	**Kekurangan**
Visual	Sudut pandang konsisten, mudah dikendalikan.	Kurang fleksibel pada gameplay yang dinamis.
Implementasi	Mudah diatur dan tidak perlu skrip.	Tidak mengikuti pemain.
Performa	Sangat ringan karena tidak ada perubahan.	Terbatas pada satu skenario visual.

3.5.8 Tips Praktik

- Gunakan kamera statis pada awal permainan untuk memperkenalkan area.
- Gabungkan kamera statis dengan cutscene.
- Jangan gunakan kamera statis untuk area eksplorasi bebas.

Kamera statis adalah pendekatan sederhana namun efektif untuk permainan yang tidak memerlukan eksplorasi luas atau perubahan perspektif. Dengan menempatkan kamera pada posisi strategis, pengembang dapat mengarahkan fokus pemain dan menyampaikan pengalaman bermain yang lebih terkontrol.

3.6 Kasus 2: Kamera Mengikuti Objek

Kamera yang mengikuti objek (object-following camera) merupakan elemen penting dalam game 3D karena memungkinkan pemain untuk tetap memiliki pandangan yang konsisten terhadap karakter atau objek utama yang sedang bergerak. Unity menyediakan fitur fleksibel untuk mengatur kamera seperti ini melalui sistem hierarki dan teknik skrip.

3.6.1 Pengaturan Hierarki dan Parenting Kamera

Hierarki Kamera

Dalam Unity, sistem hierarki memungkinkan pengelompokan objek sehingga transformasi dari satu objek induk (parent) dapat memengaruhi objek turunannya (child). Dengan menempatkan kamera sebagai child dari objek yang ingin diikuti, kita dapat membuat kamera bergerak bersama objek tersebut.

Langkah-langkah:

1. **Pilih kamera (Main Camera)** di Scene.
2. **Drag kamera ke dalam objek pemain** di panel Hierarchy, sehingga kamera menjadi child.
3. **Posisikan kamera** di belakang atau atas objek agar dapat melihat arah pergerakan.
4. **Lock posisi relatif kamera** menggunakan script atau komponen transformasi.

Contoh Ilustrasi Hierarki:

scss

CopyEdit

Player (Parent)

├── Main Camera (Child)

3.6.2 Pergerakan Kamera Dinamis

Kamera tidak hanya bisa dipasang statis ke objek, tetapi juga bisa diprogram agar bergerak secara dinamis mengikuti objek dengan kelembutan (smooth follow) agar tidak terlalu kaku. Ini bisa dilakukan menggunakan skrip Vector3.Lerp() atau SmoothDamp().

Contoh Script Kamera Mengikuti Objek (C#):

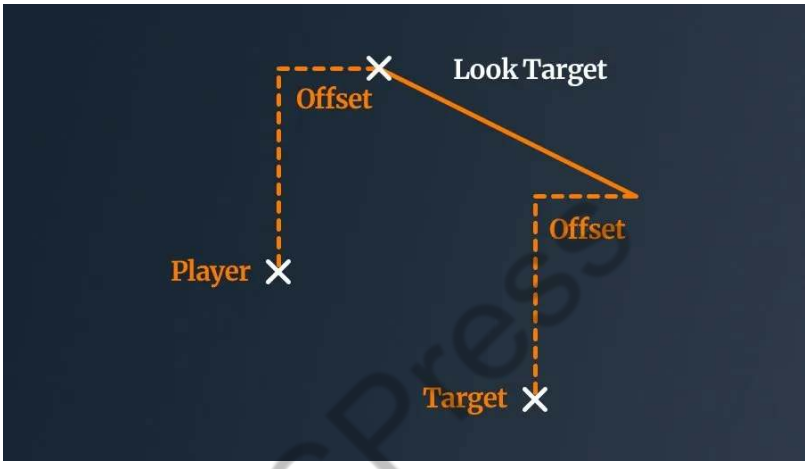
```
csharp
CopyEdit
public class FollowPlayer : MonoBehaviour {
    public Transform target;
    public Vector3 offset;
    public float smoothSpeed = 0.125f;

    void LateUpdate() {
        Vector3 desiredPosition = target.position + offset;
        Vector3 smoothedPosition = Vector3.Lerp(transform.position, desiredPosition, smoothSpeed);
        transform.position = smoothedPosition;

        transform.LookAt(target);
    }
}
```

- target: Objek yang diikuti.
- offset: Jarak kamera terhadap objek.
- Lerp(): Menghaluskan gerakan kamera agar tidak langsung.
- LookAt(): Kamera menghadap ke arah objek.

3.6.3 Studi Kasus: Kamera Mengikuti Karakter Pemain

Langkah-langkah Praktik:

1. Siapkan karakter pemain dengan komponen Rigidbody dan Animator.
2. Tambahkan kamera dan tempatkan di belakang pemain.
3. Gunakan script FollowPlayer seperti di atas.
4. Jalankan game dan gerakkan pemain – kamera akan mengikuti.



Gambar 3. 7 ilustrasi konsep offset dan arah pandang (Look Target) dalam Unity

Gambar 3.7 merupakan ilustrasi **konsep offset dan arah pandang (Look Target)** dalam sistem kamera di Unity, khususnya untuk kamera yang **mengikuti objek** atau karakter dalam permainan.

Komponen Gambar:

1. **Player** – Titik awal yang menunjukkan posisi karakter pemain di dalam dunia 3D.
2. **Target** – Titik lain yang dapat berupa objek musuh, NPC, atau item tujuan yang ingin dituju oleh pemain.
3. **Offset** – Jarak relatif kamera terhadap objek yang diikuti (baik player maupun target).
4. **Look Target** – Titik arah pandang kamera, yaitu ke mana kamera akan menghadap.

Tabel 3. 7 Makna Setiap Elemen

Elemen	Fungsi
Offset	Mengatur posisi kamera relatif terhadap objek (player atau target).
Look Target	Menentukan arah pandang kamera, sering diarahkan ke target tertentu agar pemain tetap dalam fokus.
Player dan Target	Objek yang berinteraksi di dalam scene dan menjadi acuan gerak kamera.

🎮 Penerapan di Unity:

Dalam Unity, gambar ini bisa dijadikan panduan untuk menulis skrip kamera seperti:

```
csharp
CopyEdit
public Transform player;
public Vector3 offset;
public Transform lookTarget;

void LateUpdate() {
    transform.position = player.position + offset;
    transform.LookAt(lookTarget);
}
```


Penjelasan kode:

- Kamera akan selalu berada di posisi player + offset.
- Kamera akan melihat ke arah lookTarget.

📈 Manfaat Penggunaan Offset dan Look Target:

- Memberikan sudut pandang yang stabil untuk pemain.
- Menjaga kamera tidak mengganggu visual objek utama.
- Memungkinkan kamera untuk tetap menghadap ke target penting (musuh, tujuan misi, dsb).
- Cocok untuk game third-person, RTS, atau cinematic mode.

🧠 Tips Implementasi:

- Gunakan **Gizmos** di Unity untuk memvisualisasikan offset dan lookTarget di Scene View.
- Sesuaikan **offset** agar kamera tidak menabrak dinding atau objek saat bergerak.
- Bisa juga digabungkan dengan sistem **camera shake** atau **zoom** untuk efek dramatis.
- Gunakan **LateUpdate()** untuk kamera agar tidak tertinggal dari gerakan objek.
- Tambahkan efek kamera seperti shake atau zoom saat terjadi interaksi.
- Pastikan offset tidak terlalu besar agar objek tetap terlihat nyaman.

Kamera yang mengikuti objek memberikan perspektif dinamis dan immersif pada pemain. Dengan mengatur parenting dan skrip yang tepat, kamera dapat menyesuaikan pergerakan karakter secara real-time tanpa mengganggu pengalaman pengguna.

3.7 Kasus 3: Kamera Interaktif

Tujuan Pembelajaran

- Menerapkan kamera yang dapat dikendalikan langsung oleh pengguna menggunakan mouse atau input.
- Mengatur gerakan kamera seperti rotasi, drag (geser), dan zoom.

- Mengintegrasikan kamera ini agar tetap responsif terhadap posisi objek utama (misalnya: pemain).

3.7.1 Pendahuluan Kamera Interaktif

Kamera interaktif memungkinkan pemain untuk menyesuaikan sudut pandang secara real-time. Jenis ini umum digunakan pada:

- Game strategi (RTS),
- Simulator 3D,
- Mode eksplorasi bebas atau editor level.

Kamera interaktif meningkatkan imersi karena pemain bisa *mengendalikan apa yang mereka lihat*, tidak hanya mengikuti karakter utama.

3.7.2 Kontrol Kamera dengan Mouse

Rotasi Kamera

Rotasi kamera biasanya dilakukan dengan menekan klik kanan dan menggeser mouse ke arah tertentu.

Contoh kode dasar rotasi kamera:

```
csharp
CopyEdit
public float rotationSpeed = 5.0f;
private float yaw = 0.0f;
private float pitch = 0.0f;

void Update()
{
   if (Input.GetMouseButton(1)) // Klik kanan mouse
   {
      yaw += rotationSpeed * Input.GetAxis("Mouse X");
```

```
            pitch -= rotationSpeed * Input.GetAxis("Mouse Y");
            transform.eulerAngles = new Vector3(pitch, yaw, 0.0f);
        }
    }
```

Drag (Geser) Kamera

Fitur ini sering digunakan di peta atau strategi. Kamera digeser horizontal atau vertikal ketika pemain menyeret layar.

```
csharp
CopyEdit
public float dragSpeed = 2;
private Vector3 dragOrigin;

void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        dragOrigin = Input.mousePosition;
        return;
    }

    if (!Input.GetMouseButton(0)) return;

    Vector3 pos = Camera.main.ScreenToViewportPoint(Input.mousePosition - dragOrigin);
    Vector3 move = new Vector3(pos.x * dragSpeed, 0, pos.y * dragSpeed);

    transform.Translate(-move, Space.World);
}
```

Zoom Kamera (Scroll Mouse)

Zoom dilakukan dengan mengatur jarak kamera dari objek atau mengubah FoV (field of view).

```
csharp
CopyEdit
public float zoomSpeed = 2.0f;
public float minZoom = 10f;
public float maxZoom = 60f;

void Update()
{
    float scroll = Input.GetAxis("Mouse ScrollWheel");
    Camera.main.fieldOfView -= scroll * zoomSpeed * 100f;
    Camera.main.fieldOfView = Mathf.Clamp(Camera.main.fieldOfView, minZoom, maxZoom);
}
```


3.7.3 Integrasi Kamera dengan Objek Pemain

Kamera bisa digabungkan dengan posisi pemain, baik sebagai:

- Kamera bebas (tetap dapat dikontrol mouse meskipun mengikuti),
- Atau kamera tetap di tempat tapi mengarah ke pemain (LookAt).

```
csharp
CopyEdit
public Transform player;
public Vector3 offset;

void LateUpdate()
```

```
{
    transform.position = player.position + offset;
    transform.LookAt(player);
}
```

3.7.4 Tips Praktis Kamera Interaktif

- Gunakan LateUpdate() untuk memastikan kamera mengikuti pergerakan objek dengan halus.
- Tambahkan batasan rotasi (misalnya, tidak boleh memutar ke bawah melebihi 90 derajat).
- Pertimbangkan obstacle yang bisa menghalangi kamera (gunakan *raycast* untuk mendeteksi tabrakan kamera dan objek).

- **Kamera interaktif memberi pemain kontrol atas rotasi, zoom, dan drag.**
- **Implementasi menggunakan kombinasi Input dan Transform.**
- **Sangat cocok untuk editor 3D, game eksplorasi, RTS, dan sandbox.**
- **Unity mendukung interaksi kamera dengan mudah melalui scripting**

3.8 Latihan Terstruktur

Pada bagian ini, bertujuan untuk:

- Mendesain **lingkungan permainan** yang mendemonstrasikan **tiga jenis kamera**.
- Mengimplementasikan **pergerakan objek** yang terkoordinasi dengan kamera.
- Menambahkan fitur **zoom dan kontrol arah pandang** menggunakan input dari pengguna.

3.8.1 Mendesain Lingkungan dengan 3 Tipe Kamera

A. Kamera Statis

- Diletakkan di posisi tetap.

- Digunakan untuk tampilan overview atau pemandangan tetap (cutscene).
- Tidak berubah meskipun objek bergerak.

csharp

CopyEdit

// Tidak ada skrip khusus. Gunakan transform posisi tetap.

B. Kamera Mengikuti Objek

- Mengikuti pemain saat bergerak.
- Ideal untuk game petualangan atau RPG.

csharp

CopyEdit

```
public Transform player;
public Vector3 offset;

void LateUpdate()
{
    transform.position = player.position + offset;
}
```

C. Kamera Interaktif (Mouse-Controlled)

- *Kamera dapat diputar, digeser, dan di-zoom.*
- *Kontrol penuh berada di tangan pengguna.*

csharp

CopyEdit

3.8.2 Langkah Mendesain Latihan

1. **Buat proyek baru di Unity (3D Template).**
2. **Tambahkan Plane** sebagai lantai.
3. **Letakkan objek "Player"** (misalnya: Capsule).

4. **Tambahkan 3 kamera berbeda**:
    - MainCamera_Static
    - MainCamera_Follow
    - MainCamera_Interactive
5. **Buat tombol atau shortcut** untuk berpindah antar kamera:

```
csharp
CopyEdit
public Camera cam1, cam2, cam3;

void Update()
{
    if (Input.GetKeyDown(KeyCode.Alpha1)) {
        cam1.enabled = true; cam2.enabled = false; cam3.enabled = false;
    }
    if (Input.GetKeyDown(KeyCode.Alpha2)) {
        cam1.enabled = false; cam2.enabled = true; cam3.enabled = false;
    }
    if (Input.GetKeyDown(KeyCode.Alpha3)) {
        cam1.enabled = false; cam2.enabled = false; cam3.enabled = true;
    }
}
```


3.8.3 Kombinasi Pergerakan Objek dan Kamera

Gunakan skrip dasar gerakan objek:

```
csharp
CopyEdit
public float speed = 5.0f;
```

```
void Update()
{
    float moveX = Input.GetAxis("Horizontal") * speed * Time.deltaTime;
    float moveZ = Input.GetAxis("Vertical") * speed * Time.deltaTime;

    transform.Translate(moveX, 0, moveZ);
}
```

3.8.4 Zoom dan Kendali Arah Pandang

Zoom dengan scroll mouse (FoV):

```
csharp
CopyEdit
void Update()
{
    float scroll = Input.GetAxis("Mouse ScrollWheel");
    Camera.main.fieldOfView -= scroll * 10f;
    Camera.main.fieldOfView = Mathf.Clamp(Camera.main.fieldOfView, 20f, 60f);
}
```

Arah Pandang (rotasi kamera pakai mouse):

```
csharp
CopyEdit
float mouseX = Input.GetAxis("Mouse X");
float mouseY = Input.GetAxis("Mouse Y");
transform.Rotate(-mouseY, mouseX, 0);
```

3.8.5 Eksperimen dan Penyesuaian

- Tambahkan **Obstacle (Cube atau Sphere)** untuk menguji posisi kamera relatif terhadap objek.
- Gunakan **Gizmo di Scene View** untuk melihat sudut kamera secara visual.

3.9 Tugas Kreativitas

Subbab ini bertujuan untuk:

- Menguji pemahaman mahasiswa dalam **merancang dan mengembangkan skenario permainan** yang mencakup integrasi sistem kamera dan path obstacle.
- Mendorong kreativitas mahasiswa dalam **mengatur tantangan visual dan spasial** dengan memanfaatkan kamera statis, dinamis, dan interaktif.
- Melatih kemampuan pengambilan keputusan desain, seperti **penempatan kamera**, **arah pandang**, dan **respon pemain terhadap lingkungan**.

3.9.1 Deskripsi Umum Tugas

Buatlah **mini game** atau simulasi yang mencakup unsur:

- Pemain (karakter utama),
- Obstacle (penghalang atau rintangan),
- Target akhir atau tujuan,
- Kamera yang **dapat berubah** sesuai konteks.

Skenario Game:

Pemain mengendalikan karakter yang bergerak menyusuri jalur yang penuh rintangan. Kamera berubah tergantung situasi: kamera statis untuk overview, kamera mengikuti karakter, dan kamera interaktif di area strategis.

3.9.2 Fitur Minimal yang Harus Ada

Tabel 3. 8 Fitur Minimal yang Harus Ada

Fitur	Deskripsi
Path Obstacle	Karakter harus melewati jalur yang berisi obstacle (benda diam atau bergerak).
Dynamic Camera	Kamera berubah dari statis → mengikuti → interaktif tergantung posisi pemain.
Perspektif	Pemilihan FoV dan posisi kamera harus mendukung navigasi dan tantangan visual.
Interaksi	Pemain dapat mengendalikan karakter menggunakan keyboard; kamera sebagian bisa dikontrol mouse.

3.9.3 Langkah-Langkah Pengerjaan

1. *Buat Scene Baru*
   - Gunakan template 3D Unity.
   - Tambahkan Plane sebagai lantai.
   - Letakkan obstacle menggunakan objek Cube, Sphere, dan Wall.
2. *Tambahkan Karakter Pemain*
   - Gunakan Capsule dengan Rigidbody dan Collider.
   - Tambahkan kontrol gerak seperti:

```
csharp
CopyEdit
public float speed = 5f;
void Update() {
    float moveX = Input.GetAxis("Horizontal");
    float moveZ = Input.GetAxis("Vertical");
    transform.Translate(new Vector3(moveX, 0, moveZ) * speed * Time.deltaTime);
}
```

3. Bangun Sistem Kamera Dinamis

- Gunakan 3 kamera:
    - Kamera statis di awal.
    - Kamera mengikuti saat karakter mulai berjalan.
    - Kamera interaktif saat mendekati rintangan berat.

Gunakan trigger area untuk mengaktifkan kamera berbeda:

```
csharp
CopyEdit
void OnTriggerEnter(Collider other) {
    if(other.CompareTag("CameraTrigger2")) {
        cam1.enabled = false;
        cam2.enabled = true;
    }
}
```

3.9.4 Penyelarasan Perspektif Kamera

Pertimbangan dalam desain kamera:

- **Field of View (FoV):**
  Lebar sudut pandang memengaruhi tingkat keterbacaan rintangan.
- **Clipping Planes:**
  Hindari kehilangan detail rintangan karena nilai near/far yang tidak tepat.
- **Angle Kamera:**
  Gunakan sudut pandang isometrik atau semi-top-down untuk navigasi area sempit.

Contoh Perbandingan:

Tabel 3. 9 Contoh Perbandingan Sudut kamera

Sudut Kamera	**Efek**
Top-down 90°	Baik untuk melihat jalur penuh, tapi kurang sinematik
Diagonal 45°	Menunjukkan arah dan kedalaman, cocok untuk obstacle bertingkat
Perspektif dinamis	Lebih imersif, tapi perlu kontrol tambahan

3.9.5 Tips dan Tantangan

- Jangan terlalu banyak obstacle dinamis, karena dapat mengganggu performa.
- Gunakan **NavMeshObstacle** jika ingin membuat jalur AI yang menghindari rintangan.
- Berikan **petunjuk visual (marker)** agar pemain tahu arah dan tujuan.
- Uji perubahan kamera agar tidak membingungkan pemain.

Tabel 3. 10 Kriteria Penilaian Bab 3

Aspek	**Bobot**
Kreativitas Desain Rintangan	25%
Implementasi Kamera Dinamis	30%
Kelengkapan Fitur Interaksi	20%
Efisiensi dan Performa	15%
Visualisasi dan Navigasi	10%

Bab 4. Pengelolaan Model dan Tekstur dalam Unity

4.1 Tujuan Pembelajaran

Bab ini dirancang untuk memberikan pemahaman konseptual dan keterampilan praktis kepada mahasiswa terkait manajemen aset grafis dalam pengembangan game dan simulasi interaktif menggunakan Unity. Fokus utama adalah bagaimana mengimpor model 3D, mengaplikasikan tekstur 2D, serta mengelola material dan shader untuk memperkaya tampilan visual dalam Unity.

Setelah mempelajari bab ini, mahasiswa diharapkan mampu:

1. **Menjelaskan peran aset visual** dalam game berbasis Unity.
2. **Mengimpor dan mengatur model 3D** dari aplikasi eksternal (misalnya Blender).
3. **Mengimpor dan menerapkan tekstur 2D** ke objek dalam Unity.
4. **Menyesuaikan material dan shader** untuk menghasilkan efek visual yang sesuai.
5. **Membuat skenario interaktif** berbasis aset model dan tekstur.
6. **Menghubungkan teori visualisasi 3D** dengan praktik penerapannya dalam Unity.

4.2 Pendahuluan: Aset Visual dalam Game dan Simulasi

4.2.1 Pengantar Visualisasi dalam Unity

Visualisasi merupakan komponen penting dalam membangun pengalaman pengguna dalam aplikasi interaktif, seperti game, simulasi, maupun media edukatif. Unity sebagai game engine memungkinkan pengembang untuk mengatur elemen visual seperti objek 3D, tekstur, dan pencahayaan secara terintegrasi. Keindahan visual tidak hanya memberikan daya tarik estetika, tetapi juga memperkuat narasi dan fungsionalitas gameplay.

4.2.2 Peran Aset Grafis

Dalam konteks Unity, aset grafis adalah segala elemen visual yang digunakan dalam scene, seperti:

- **Model 3D** (karakter, objek, lingkungan)
- **Tekstur 2D** (permukaan objek)
- **Material** (penghubung antara tekstur dan shader)
- **Shader** (pengontrol bagaimana objek ditampilkan)

4.2.3 Format File Aset yang Didukung Unity

Unity mendukung berbagai jenis file aset yang umum digunakan di industri, seperti Tabel 4.1:

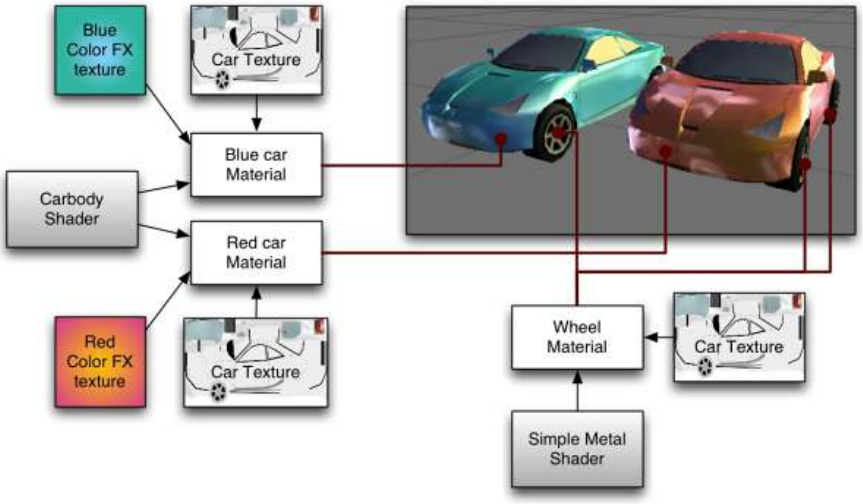
Tabel 4. 1 Jenis format file di Unity

Jenis Aset	**Format File**	**Keterangan**
Model 3D	.fbx, .obj, .dae	Dapat dihasilkan dari Blender, Maya, dsb.
Tekstur	.png, .jpg, .tga	Digunakan untuk permukaan objek.
Shader	.shader, Shader Graph	Untuk mengatur efek pencahayaan.
Material	.mat	Menggabungkan tekstur dan shader.

4.2.4 Hubungan Model, Tekstur, dan Material

Visualisasi objek 3D dalam Unity tidak hanya mengandalkan bentuk geometris (mesh), tetapi juga penampilan permukaan objek tersebut.

- **Model 3D (Mesh):** Struktur geometris objek.
- **Tekstur:** Gambar bitmap yang menempel di permukaan objek.
- **Material:** Objek Unity yang mengatur bagaimana tekstur ditampilkan, termasuk properti seperti warna, kilau, dan transparansi.
- **Shader:** Algoritma yang memproses bagaimana cahaya berinteraksi dengan material.



Gambar 4. 1 hubungan hierarkis antara Mesh, Material, Shader, dan Texture

Gambar 4.1, menunjukkan **hubungan hierarkis antara *Mesh*, *Material*, *Shader*, dan *Texture*** dalam proses rendering objek 3D di Unity, dengan contoh dua mobil (biru dan merah) yang berbagi struktur namun memiliki tampilan visual berbeda.

1. Mesh (Model Mobil & Roda)

- Merupakan **struktur geometris objek 3D** — dalam hal ini mobil dan roda.
- Mesh **tidak memiliki warna atau tampilan visual** tanpa material yang melekat padanya.

2. Material (Blue Car Material, Red Car Material, Wheel Material)

- **Material** adalah jembatan antara ***Mesh* dan *Shader***.
- Gambar menunjukkan dua *material* mobil (Blue car material dan Red car material) serta satu *material* roda (Wheel material).
- Setiap material mengatur:
    - **Shader yang digunakan**
    - **Tekstur yang diterapkan**
    - **Warna atau efek khusus lainnya**

3. Shader (Carbody Shader & Simple Metal Shader)

- Shader adalah **script pemrograman grafis** yang menentukan bagaimana material merespons cahaya dan menghasilkan efek visual.
- Dalam gambar:

- Carbody Shader dipakai oleh mobil untuk efek pantulan atau highlight.
- Simple Metal Shader digunakan pada roda untuk memberikan kesan logam atau chrome.

4. Texture (Car Texture, Red/Blue FX Texture)

- Tekstur berupa gambar bitmap yang **diterapkan pada permukaan objek**.
- Dua efek warna (Blue Color FX texture dan Red Color FX texture) diterapkan ke material mobil yang berbeda, meskipun menggunakan Car Texture yang sama.
- Roda juga menggunakan Car Texture untuk detail grafis, namun diproses melalui Simple Metal Shader.

Alur Relasi Visual (Flow)

1. **Mesh Mobil** → dipasangkan ke:
   - Blue Car Material → Carbody Shader + Car Texture + Blue Color FX Texture
   - Red Car Material → Carbody Shader + Car Texture + Red Color FX Texture
2. **Mesh Roda** → dipasangkan ke:
   - Wheel Material → Simple Metal Shader + Car Texture

Kesimpulan Konseptual

Gambar ini menegaskan konsep penting dalam Unity:

- **Mesh = bentuk**,
- **Material = tampilan**,
- **Shader = logika visualisasi**,
- **Texture = gambar permukaan**.

Dengan hanya mengubah **material atau tekstur**, objek dengan **mesh yang sama** bisa memiliki **tampilan berbeda**, seperti mobil biru dan merah pada gambar.

4.2.5 Aset dalam Proyek Unity

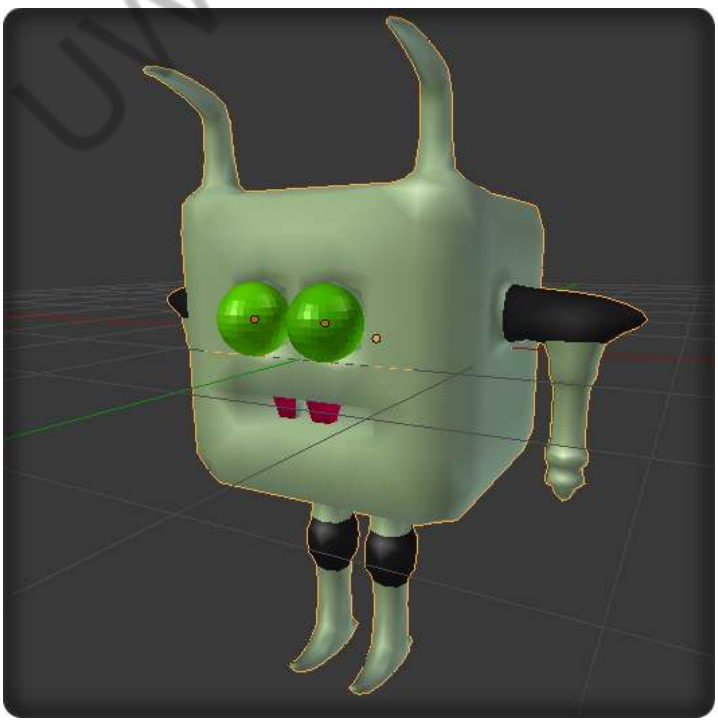
Semua aset diatur di dalam folder Assets/ pada Unity Editor. Pengelolaan yang baik penting untuk efisiensi kerja dan kolaborasi tim, terutama jika proyek menjadi kompleks. Direkomendasikan untuk:

- Menyusun sub-folder (Textures/, Models/, Materials/).
- Menggunakan nama file konsisten dan deskriptif.
- Menghindari duplikasi file.

4.2.6 Workflow Visual di Unity

Urutan umum dalam pengelolaan aset visual:

1. **Desain model dan tekstur** di software eksternal (mis. Blender, Photoshop).
2. **Ekspor model** ke format .fbx atau .obj.
3. **Impor model dan tekstur** ke Unity.
4. **Membuat Material** di Unity dan menerapkan tekstur.
5. **Menempatkan model** ke dalam scene.
6. **Menambahkan pencahayaan dan kamera.**



Gambar 4. 2 Contoh karakter 3D yang dibuat dalam Blender

Gambar 4.2, menunjukkan **objek karakter 3D yang dibuat dalam Blender**, yaitu sebuah model karakter berbentuk kartun bergaya low-poly. Dari gambar ini, kita bisa menjelaskan **alur kerja umum dari pembuatan objek di Blender ke Unity**, sebagai berikut:

Alur Kerja Blender ke Unity (Workflow Pipeline)

1. Modeling di Blender

- **Langkah awal** adalah membuat bentuk 3D (mesh) objek seperti karakter yang ditampilkan pada gambar.
- Objek ini bisa dibentuk dari **primitive shapes** seperti cube, sphere, cylinder yang kemudian di-*sculpt* atau dimodifikasi.
- Karakter pada gambar memiliki fitur:
    - Mata besar bulat
    - Lengan dan kaki modular
    - Tanduk dan mulut sederhana
- Blender digunakan untuk mengatur:
    - **Vertex, edge, face**
    - **Proporsi dan volume karakter**
    - **Modifikasi mesh (misalnya dengan modifier seperti Subdivision Surface atau Mirror)**

2. Texturing dan Material

- Setelah bentuk selesai, dilakukan pemberian **warna dasar (material)**.
- Dalam gambar terlihat karakter memiliki:
    - **Warna kulit hijau pucat** (kemungkinan dengan material tanpa tekstur)
    - **Mata hijau terang** (mungkin menggunakan material emisif)
    - **Lengan hitam, mulut merah** — masing-masing bagian diberi material berbeda.
- Anda bisa menggunakan:
    - **Blender Internal Material** atau
    - **UV Mapping + Image Texture** (jika ingin ekspor tekstur ke Unity)

3. Rigging (Opsional)

- Jika karakter ini akan dianimasikan di Unity:
    - Anda perlu **menambahkan Armature (tulang) di Blender**.
    - Proses ini disebut **rigging**, diikuti dengan **weight painting**.
- Kalau hanya untuk objek statis, bagian ini bisa dilewati.

4. Animasi (Jika Diperlukan)

- Anda bisa membuat animasi **idle**, **walk**, **jump**, dll langsung di Blender.
- Unity akan mengenali animasi ini sebagai **Action** di dalam file .fbx.

5. Ekspor ke Unity

- Format ekspor umum:
    - .FBX (paling umum)
    - .OBJ (untuk model statis)
    - .BLEND (Unity juga bisa langsung membaca file .blend, tetapi direkomendasikan dikonversi ke .fbx agar stabil)
- Ekspor disertai:
    - **Mesh**
    - **Material**
    - **Animation (jika ada)**
    - **Armature/bone (jika ada)**

6. Impor ke Unity

- Drag file .fbx ke folder Assets/ di Unity.
- Unity akan otomatis:
    - Membaca mesh model
    - Memisahkan animasi (jika ada)
    - Menyertakan material, meskipun perlu **penyesuaian shader** (misalnya ke URP/HDRP)
- Anda bisa menambahkan komponen:
    - **Collider**
    - **Rigidbody**

- **Script kontrol karakter**

4.2.7 Dampak Visual terhadap Pengalaman Pengguna

Penggunaan model dan tekstur yang realistis dapat:

- Meningkatkan imersi pemain.
- Membantu penjelasan objek dalam simulasi edukatif.
- Menyesuaikan tone (suasana) dalam game.
- Mengurangi kebutuhan teks instruksi.

Studi oleh Sharma (2020) menunjukkan bahwa pemain cenderung lebih memahami narasi dan kontrol dalam game ketika visualisasi sesuai konteks.

Tips Implementasi Aset Visual:

- Gunakan tekstur **berukuran proporsional** agar tidak membebani performa.
- Simpan model dalam skala 1:1 untuk menghindari perbedaan ukuran saat diimpor.
- Perhatikan **pivot point** saat membuat model 3D agar sesuai dalam Unity.

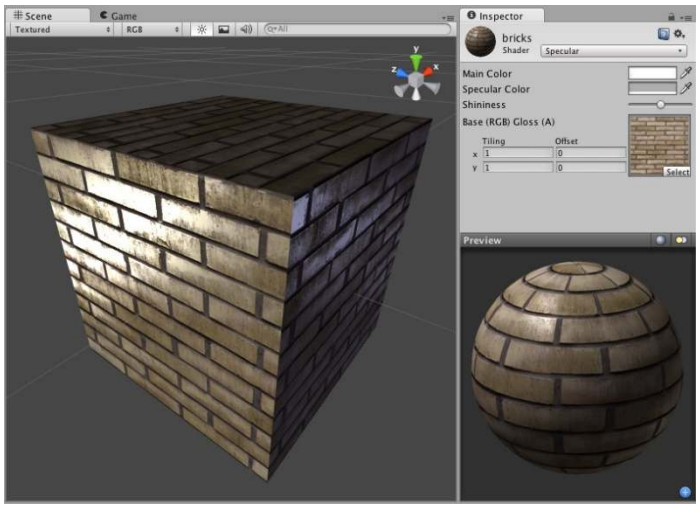
- **Aset visual meliputi model, tekstur, material, dan shader.**
- **Unity mendukung berbagai format populer seperti .fbx dan .png.**
- **Material adalah penghubung antara tekstur dan shader.**
- **Workflow pengelolaan aset dimulai dari desain hingga integrasi di scene.**
- **Visualisasi yang efektif meningkatkan daya tarik dan pemahaman pengguna.**

4.3 Penggunaan Tekstur 2D

4.3.1 Jenis-Jenis Tekstur dalam Unity

Dalam Unity, tekstur adalah gambar 2D yang diaplikasikan ke permukaan objek 3D untuk menambahkan detail visual. Beberapa jenis tekstur umum meliputi:

1. **Diffuse Map (Albedo)**
    - Memberikan warna utama pada permukaan objek.
    - Tidak mencerminkan pencahayaan atau pantulan.
2. **Normal Map**
    - Mensimulasikan detail kecil seperti tonjolan atau cekungan tanpa menambah polygon.
    - Membuat permukaan tampak lebih realistis.
3. **Specular Map / Metallic Map**
    - Mengatur tingkat pantulan dan kilau permukaan.
    - Specular untuk shader legacy, Metallic Map untuk PBR (Physically Based Rendering).
4. **Height Map (opsional)**
    - Digunakan untuk displacement mapping (menaik-turunkan permukaan).
5. **Occlusion Map**
    - Mengatur pencahayaan area gelap (ambient occlusion).



Gambar 4. 3 contoh penerapan tekstur pada objek 3D di Unity

Gambar 4.3, menunjukkan **contoh penerapan tekstur pada objek 3D di Unity**, tepatnya sebuah *cube* (kubus) dengan **material "bricks"** (bata), menggunakan **shader jenis *Specular***.

Berikut penjelasan rinci dari tiap bagian Gambar 4.3,:

1. Objek 3D (Cube)

- Objek utama di Scene Unity.
- Di-*wrap* (dilapisi) dengan tekstur **bata** (brick).
- Efek pencahayaan menunjukkan **kilau refleksi** akibat pengaturan **specular**.

2. Inspector Panel (Kanan)

Panel ini menunjukkan properti dari **material** yang diterapkan pada objek.

- **Shader: Specular**
  Shader ini memungkinkan pengaturan refleksi cahaya (glossy), cocok untuk permukaan mengkilap seperti batu basah, logam, atau ubin.
- **Main Color**
  Warna dasar material (bisa dikombinasikan dengan tekstur).
- **Specular Color**
  Warna dari kilau/refleksi cahaya.
- **Shininess**
  Mengontrol intensitas dan luasnya refleksi. Nilai tinggi = refleksi tajam (glossy), nilai rendah = menyebar.
- **Base (RGB) Gloss (A)**
  Slot tekstur utama (diisi gambar batu bata), menggunakan kanal Alpha untuk mengatur glossiness (jika tersedia).
- **Tiling (x, y)**
  Mengatur pengulangan tekstur. Nilai 1 berarti gambar tampil 1x pada permukaan. Jika diperbesar, tekstur tampak lebih kecil dan berulang.
- **Offset (x, y)**
  Menggeser posisi tampilan tekstur ke kiri/kanan atau atas/bawah.

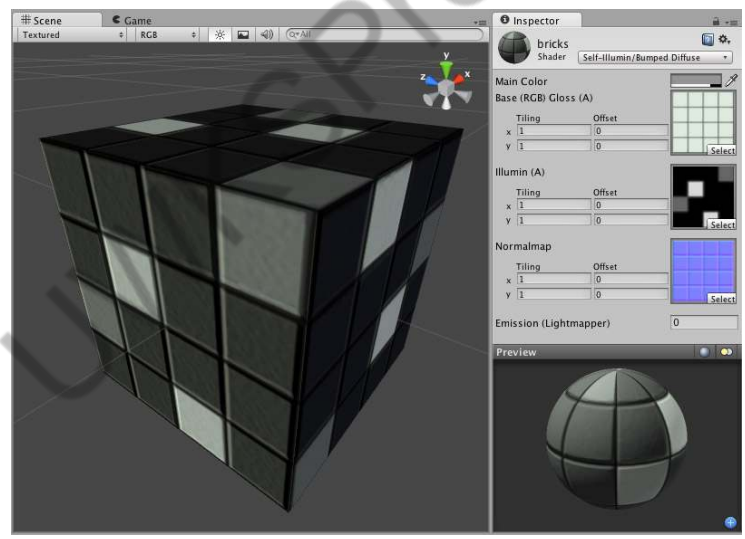
3. Preview Sphere (Bawah Kanan)

- Menampilkan **pratinjau material** dalam bentuk bola.
- Berguna untuk mengecek efek pencahayaan, refleksi, dan kualitas tekstur.

Kaitan Dengan Materi Tekstur 2D

Gambar ini cocok untuk mendukung pembahasan pada Penggunaan Tekstur 2D, khususnya bagian:

- **Jenis shader (Specular)** untuk menampilkan refleksi.
- **Penggunaan tekstur Diffuse + pengaturan Shininess**.
- **Tiling dan Offset** untuk kontrol visual tekstur.
- **Preview Sphere** sebagai alat bantu desain sebelum aplikasi ke objek sebenarnya.



Gambar 4. 4 contoh penerapan shader tipe "Self-Illumin/Bumped Diffuse" di Unity

Gambar 4.4, menunjukkan **contoh penerapan shader tipe "Self-Illumin/Bumped Diffuse"** di Unity, yang menggunakan beberapa jenis tekstur sekaligus, termasuk **Diffuse Map** dan **Normal Map**. Mari kita bahas hubungannya secara detail.

1. Diffuse Map (Base (RGB) Gloss (A))

- Ini adalah **tekstur utama** yang menentukan **warna visual** permukaan objek (dalam hal ini pola kotak seperti ubin).

- Bekerja sebagai **peta warna (color map)** yang dibungkus ke permukaan 3D melalui UV Mapping.
- Disebut juga sebagai **Albedo Map** dalam versi Unity modern.

Diffuse Map tidak memiliki informasi tentang kedalaman atau bentuk – hanya warna datar 2D.

2. Normal Map

- Gambar biru keunguan ini **meniru detail permukaan** (seperti tonjolan dan lekukan) **tanpa menambahkan geometri baru**.
- Unity membaca normal map sebagai instruksi pencahayaan: bagaimana cahaya harus dipantulkan agar permukaan terlihat "bertekstur".
- Contohnya pada kubus di gambar, meskipun permukaan aslinya datar, namun dengan normal map, terlihat seperti ada cekungan atau garis tepi antara ubin.

Normal Map menambahkan kesan *bump* atau *relief* yang akan terlihat saat objek terkena cahaya.

3. Illumin (Self-Illumination) Map

- Gambar hitam putih digunakan untuk menentukan bagian mana dari tekstur yang memancarkan cahaya sendiri (emissive).
- Bagian putih = bercahaya, hitam = tidak memancarkan cahaya.

Detail Hubungan Diffuse vs Normal Map seperti Tabel 4.2,

Tabel 4. 2 Detail Hubungan Diffuse vs Normal Map

Komponen	Fungsi	Contoh pada Gambar
Diffuse Map	Memberi warna dan tekstur permukaan 3D.	Ubin-ubin hitam-putih pada permukaan kubus.

Normal Map	Mensimulasikan bentuk/tekstur dengan memanipulasi pencahayaan.	Garis-garis cekung antar ubin yang terlihat nyata.

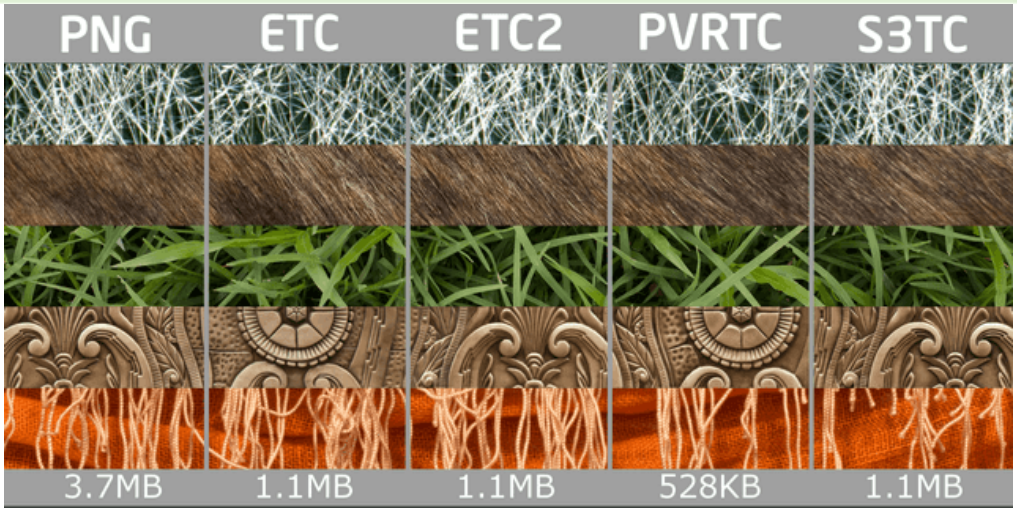
- **Diffuse** menentukan **apa** yang Anda lihat (warna/pola),
- **Normal** menentukan **bagaimana** itu tampak dalam pencahayaan (efek 3D/bertekstur).

4.3.2 Format File dan Resolusi Optimal

Unity mendukung berbagai format file tekstur. Masing-masing memiliki kelebihan tersendiri, seperti Tabel 4.3:

Tabel 4. 3 Kelebihan dan kekurangan format file tekstur

Format	**Kelebihan**	**Kekurangan**
PNG	Kualitas tinggi, transparansi	Ukuran file besar
JPEG	Ukuran kecil	Tidak mendukung transparansi
TGA	Mendukung alpha & normal map	File berat
PSD	Bisa langsung dari Photoshop	Butuh plugin di build



Gambar 4. 5 perbandingan visual dan ukuran file dari berbagai format kompresi tekstur

Gambar 4.5, menunjukkan **perbandingan visual dan ukuran file** dari berbagai **format kompresi tekstur** yang umum digunakan dalam pengembangan game atau aplikasi grafis, terutama di Unity dan engine sejenis.

Secara detail penjelasan setiap kolom (Format Tekstur), dari Gambar 4.5, adalah :

✅ *PNG*

- **Ukuran**: 3.7MB
- **Tipe**: *Lossless image format* (tanpa kehilangan kualitas).
- **Kelebihan**: Kualitas sangat tinggi, tidak ada artefak kompresi.
- **Kekurangan**: Ukuran besar, tidak ideal untuk real-time rendering.
- **Penggunaan**: Biasanya sebagai sumber asli untuk diimpor ke engine, tapi tidak digunakan langsung di runtime.

✅ *ETC (Ericsson Texture Compression)*

- **Ukuran**: 1.1MB
- **Tipe**: *Lossy compression.*
- **Kelebihan**: Didukung luas di Android (OpenGL ES 2.0+).
- **Kekurangan**: Tidak mendukung transparansi (alpha channel).
- **Kualitas**: Baik untuk tekstur opaque (tanpa transparansi).

✅ *ETC2*

- **Ukuran**: 1.1MB
- **Tipe**: Pengembangan dari ETC.
- **Kelebihan**: Mendukung alpha channel (transparansi).
- **Dukungan**: OpenGL ES 3.0 ke atas.
- **Kualitas**: Lebih baik dari ETC, cocok untuk tekstur transparan.

✅ *PVRTC (PowerVR Texture Compression)*

- **Ukuran**: 528KB (PALING KECIL!)
- **Kelebihan**: Kompresi sangat efisien.
- **Penggunaan**: Umum pada perangkat iOS (iPhone, iPad).
- **Kekurangan**: Kualitas visual bisa turun drastis (terlihat blurry atau artefak), terutama untuk tekstur detail.
- **Catatan**: Ideal untuk mobile dengan keterbatasan memori.

✅ *S3TC (aka DXT)*

- **Ukuran**: 1.1MB
- **Tipe**: Digunakan di DirectX dan OpenGL.
- **Kelebihan**: Kualitas visual relatif baik.
- **Kekurangan**: Ukuran masih lumayan besar; kurang efisien dibanding PVRTC.
- **Penggunaan**: Umum di platform PC dan konsol.

Tabel 4. 4 Kesimpulan Perbandingan format gambar

Format	Ukuran	Transparansi	Target Platform	Kelebihan
PNG	3.7MB	Ya	Semua (sebagai sumber)	Kualitas maksimal

ETC	1.1MB	Tidak	Android	Kompatibel luas, ukuran kecil
ETC2	1.1MB	Ya	Android 3.0+	Dukungan alpha + kualitas bagus
PVRTC	528KB	Ya	iOS	Ukuran paling kecil
S3TC	1.1MB	Ya	PC/Konsol	Kualitas seimbang, banyak dukungan

Tabel 4. 5 Kegunaan berdasar fungsi objek

Kegunaan	**Resolusi Rekomendasi**
Objek utama	2048 x 2048 px
Objek menengah	1024 x 1024 px
Objek latar	512 x 512 px
UI/ikon	256 x 256 px

4.3.3 Penerapan Tekstur ke Material

Langkah-langkah:

1. Buat **Material** baru di Unity:
   Assets > Create > Material
2. Pilih shader (Standard/URP/HDRP).
3. Masukkan tekstur ke slot:
   - Albedo → warna dasar
   - Normal Map → detail permukaan

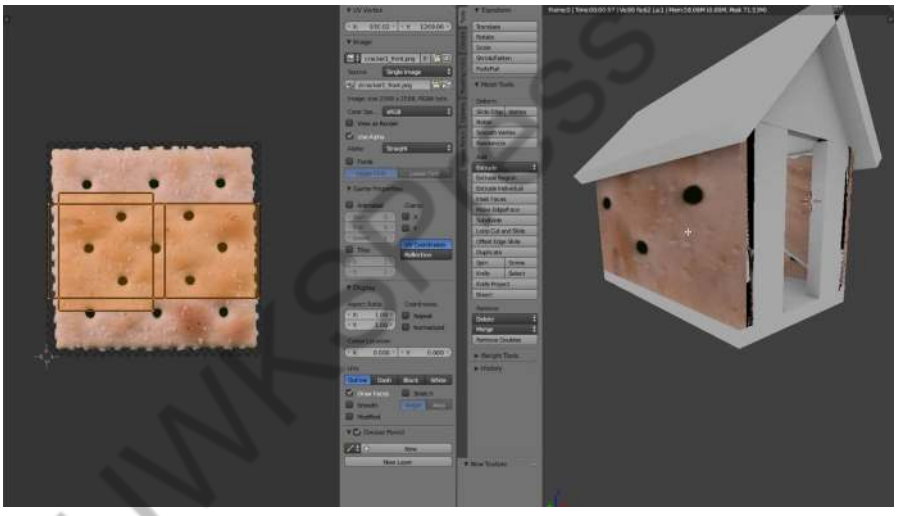
- Metallic/Smoothness → refleksi

4.3.4 Penyesuaian UV Mapping

UV Mapping adalah proses memetakan gambar 2D ke permukaan objek 3D. Diperlukan agar tekstur menyatu secara proporsional ke bentuk objek.

1. Dilakukan dalam software 3D (misal: Blender).
2. Diatur dengan **Unwrap** atau **Smart UV Project**.
3. Di Unity, bisa diuji langsung dengan checker texture.

Contoh uv unwrapping blender unity, seperti Gambar 4.6.



Gambar 4. 6 UV unwrapping blender unity

Gambar 4.6, menunjukkan **proses UV Mapping dalam Blender**, yang digunakan untuk **menempatkan tekstur 2D (dalam hal ini gambar cracker)** ke permukaan objek 3D (sebuah model rumah kecil), secara detail sebagai berikut :

➤ *Kiri: Tampilan UV/Image Editor*

- Di sini terlihat gambar tekstur bernama cracker1_front.png.
- Terdapat **dua kotak oranye transparan** yang menunjukkan **UV Island** — yaitu area dari mesh 3D yang dipetakan ke area spesifik dalam tekstur.
- Artinya: bagian tertentu dari model 3D (misalnya dinding rumah) akan menampilkan area tekstur dari cracker sesuai koordinat UV tersebut.

➤ *Tengah: Panel Properties di Blender*

- Panel ini berisi informasi tentang:
  - Gambar yang digunakan (cracker1_front.png)
  - Ukuran gambar: **2588 x 2588**
  - Opsi seperti Clamp, UV Coordinates, dan lainnya.
- Juga ditampilkan fitur-fitur mesh tools seperti **Extrude, Inset Faces, Loop Cut, Knife**, dsb. Namun dalam konteks ini, fokus utama adalah **UV Mapping**.

➤ *Kanan: Tampilan Viewport 3D*

- Tampak model **rumah kecil 3D** yang sebagian permukaannya telah **ditempeli gambar cracker**.
- Ini adalah hasil dari pemetaan UV di sisi kiri tadi.
- Karena pemetaan tidak mencakup semua sisi, kita melihat beberapa bagian rumah masih putih polos atau belum dikenai tekstur.

Apa itu UV Mapping?

UV Mapping adalah proses memproyeksikan **tekstur 2D** ke permukaan objek **3D**. Koordinat UV adalah sistem 2D (U dan V) yang mengatur bagian mana dari gambar yang ditampilkan di bagian tertentu objek 3D.

Manfaat UV Mapping:

- Mengatur **bagian mana dari gambar yang muncul di mesh**.
- Dapat digunakan untuk **teks, detail gambar, atau pola rumit**.
- Efisien untuk membuat game asset karena **mengontrol resolusi tekstur** dan menghindari artefak visual.

Tips dalam UV Mapping di Blender:

- Gunakan **"Unwrap"** (di Edit Mode) untuk menghasilkan UV map dasar.
- Pastikan proporsi gambar dan objek sesuai agar tidak terlihat melar.

- Gunakan **seam** untuk objek kompleks agar pemetaan lebih rapi.
- Gunakan **shading view** untuk melihat preview tekstur real-time.

Format umum:

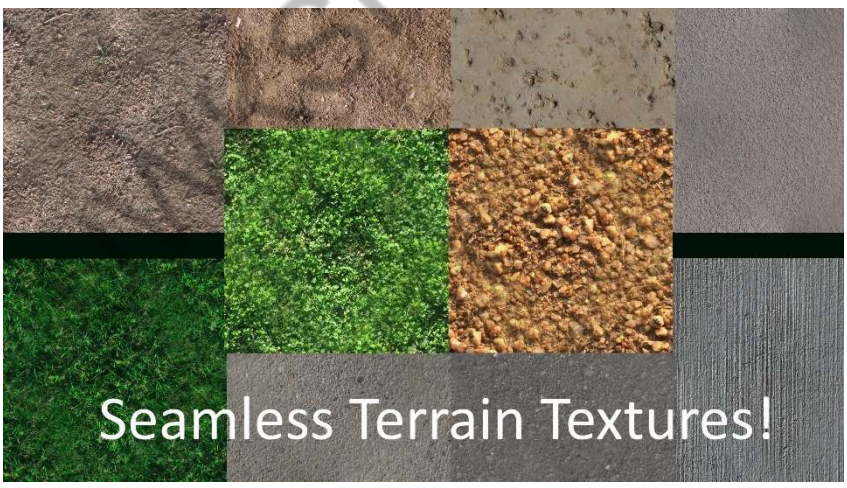
- UV dibagi menjadi koordinat **U (horizontal)** dan **V (vertikal)**.
- Gambar seperti peta diproyeksikan ke permukaan objek.

4.3.5 Contoh Penggunaan Tekstur

A. Objek Statis

Contoh: tembok, lantai, pohon.

- Gunakan kombinasi:
  - Diffuse (warna dasar)
  - Normal Map (tekstur kasar)
  - Occlusion Map (gelap terang)



Gambar 4. 7 Contoh Terrain texture

Gambar 4.7, menampilkan **berbagai contoh tekstur terrain (permukaan tanah)** yang **bersifat "seamless"**, yang artinya **tekstur tersebut dapat diulang (tileable)** secara horizontal dan vertikal **tanpa terlihat garis batas atau sambungan** antar gambar.

🟫 Apa itu Seamless Terrain Texture?

- **Seamless texture** adalah gambar tekstur 2D yang dirancang agar **bisa diulang (tiled)** secara terus-menerus tanpa menciptakan pola yang jelas atau sambungan yang mencolok.
- Dalam konteks **terrain (medan)** di game atau simulasi 3D, seamless texture memungkinkan **permukaan luas seperti tanah, pasir, rumput, bebatuan, dan aspal** ditampilkan secara realistis tanpa harus menggunakan tekstur berukuran besar.

Tabel 4. 6 Kegunaan Seamless Texture dalam Game Development

Kegunaan	Penjelasan
🟩 **Efisiensi Memori**	Hanya satu file tekstur kecil yang diulang berulang kali, sehingga tidak membebani GPU/memori.
🟫 **Visual Natural**	Karena seamless, tidak terlihat sambungan antar tekstur—hasilnya lebih alami.
⚙️ **Skalabilitas**	Dapat digunakan untuk area kecil maupun luas tanpa kehilangan detail.
🌍 **Terrain Painting**	Digunakan dalam tools seperti Unity Terrain, Unreal Landscape, dsb., untuk mengecat permukaan.

🔍 Elemen Tekstur dalam Gambar:

- **Tekstur tanah** (soil, dirt, mud)
- **Tekstur rumput** (grass, clover, lawn)
- **Tekstur batuan dan kerikil** (gravel, pebbles)
- **Tekstur beton/plester** (plaster, rough concrete)
- **Tekstur kayu atau permukaan kasar** (wood grain, bark)

💡 Tools/Software untuk Menggunakan Seamless Texture:

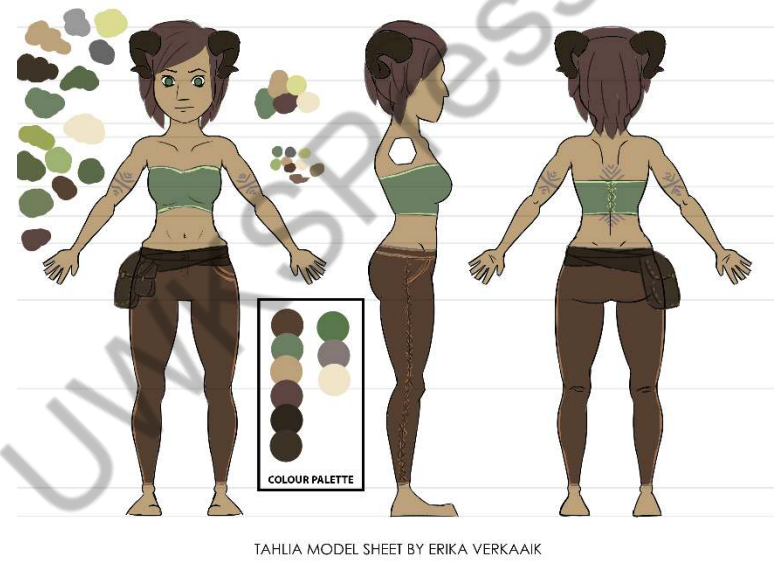
- **Unity**: melalui *Terrain Editor* dan *Tileable Texture Material*

- **Unreal Engine**: *Landscape Material* dengan tiling
- **Blender**: melalui UV Mapping dan Image Texture node
- **Substance Designer**: untuk membuat tekstur seamless procedural
- **Photoshop/GIMP**: bisa digunakan untuk membuat gambar jadi seamless (menggunakan filter offset & clone tool)

B. Objek Dinamis

Contoh: karakter, musuh, kendaraan.

- Gunakan tekstur dengan resolusi tinggi.
- Dapat diganti runtime dengan script (misal efek luka/damage).



Gambar 4. 8 Character Model Sheet

Gambar 4.8, adalah ***Character Model Sheet***, atau *Model Sheet Karakter*, dari seorang karakter bernama **Tahlia** (dibuat oleh Erika Verkaaik).

Apa Itu Model Sheet?

Model sheet adalah referensi visual yang menunjukkan tampilan karakter dari berbagai sudut untuk keperluan modeling 3D, animasi, atau ilustrasi. Model sheet ini penting dalam pipeline animasi dan game agar tim dapat mempertahankan **konsistensi desain**.

🔄 Tampilan Multi-sudut

Gambar 4.8, menampilkan karakter dalam beberapa pose:

1. **Tampilan Depan (Front View)** – Untuk memperlihatkan wajah dan pakaian dari sisi depan.
2. **Tampilan Samping (Side View)** – Memudahkan pemodelan volume tubuh dan pose lengan.
3. **Tampilan Belakang (Back View)** – Menunjukkan rambut, pakaian, dan aksesori dari belakang.
4. **Tampilan ¾ (Three Quarter)** – Sering digunakan untuk memahami bentuk 3D karakter.

🎨 Palet Warna (Colour Palette)

Di bagian tengah bawah, terdapat **kotak warna** yang disebut ***color palette***. Ini adalah referensi warna yang digunakan dalam karakter. Berguna untuk:

- Konsistensi warna di seluruh produksi.
- Pembuatan *textures*, shading, dan pencahayaan di game engine (seperti Unity atau Unreal).

Palet ini meliputi:

- Warna rambut (cokelat tua).
- Warna kulit.
- Warna pakaian (hijau, hitam, cokelat).
- Highlight warna terang seperti krem.

👜 Aksesori dan Detail

- Karakter memakai tas pinggang, sepatu ringan, dan pakaian fungsional.
- Ciri khas unik seperti **tanduk kecil** di kepala menambah kepribadian karakter.

Tabel 4. 7 Kegunaan Model Sheet dalam Pipeline 3D

Tahap	Peran Model Sheet
🎨 **Desain Konsep**	Sebagai acuan awal gaya dan proporsi.
🧊 **3D Modeling**	Digunakan untuk membentuk model di software seperti Blender atau Maya.
🎮 **Rigging & Animation**	Membantu animator dalam menjaga konsistensi gerak.
🖌️ **Texturing**	Palet warna digunakan untuk mewarnai UV Map/Material.

4.3.6 Studi Kasus Mini: Penerapan Tekstur Mobil

1. Model mobil dibuat di Blender atau 3dsMax.
2. UV Mapping disiapkan dalam software modeling.
3. Di Unity:
    - Buat 2 material (Red dan Blue).
    - Tambahkan tekstur car_diffuse, car_normal, car_metallic.
    - Terapkan material ke bagian body dan roda.

4.3.7 Tips Praktis Tekstur dalam Unity

- Gunakan tekstur seamless (tanpa tepi jelas) untuk lantai atau dinding.
- Hindari ukuran tekstur tidak kelipatan 2 (misal: 1153 x 778).
- Gunakan kompresi (ASTC / DXT) untuk memperkecil file build.
- Preview UV layout di software 3D sebelum ekspor ke Unity.

Aspek	Rincian
Jenis Tekstur	Diffuse, Normal, Specular, Occlusion, Height
Format	PNG, JPG, TGA, PSD
Resolusi	512-2048 px tergantung fungsi
Aplikasi	Drag & drop ke material editor Unity
UV Mapping	Penting untuk penempatan tekstur yang akurat
Tips	Gunakan checker pattern untuk uji tekstur, kompresi

4.4 Impor Model 3D dari Blender

Dalam pipeline produksi game modern, integrasi antara software modeling seperti **Blender** dan **game engine** seperti **Unity** merupakan hal yang sangat penting. Blender menyediakan kemampuan modeling, rigging, animasi, hingga rendering. Sedangkan Unity menjadi media utama menjalankan model tersebut secara **interaktif dalam game atau simulasi**. Oleh karena itu, memahami bagaimana proses impor model 3D dari Blender ke Unity secara tepat akan menghindarkan banyak masalah kompatibilitas, skala, orientasi, dan animasi.

4.4.1 Format Ekspor: .fbx vs .obj

- FBX (.fbx)

FBX (Filmbox) adalah format yang **paling direkomendasikan** untuk integrasi Blender–Unity karena:

- Mendukung **mesh**, **UV**, **material**, **rigging**, **bone**, hingga **animasi**.
- Satu file .fbx bisa menyimpan **beberapa action animasi** sekaligus.
- Unity mengenali file .fbx secara **native**, tanpa konversi tambahan.

- OBJ (.obj)

OBJ lebih sederhana dan ringan, tetapi memiliki banyak keterbatasan:

- Tidak menyimpan informasi animasi atau rigging.
- Hanya menyimpan bentuk mesh dan informasi UV.
- Membutuhkan file .mtl terpisah untuk informasi material dan tekstur.

Tabel 4. 8 Tabel Perbandingan .fbx vs .obj

Aspek	FBX (.fbx)	OBJ (.obj)
Dukungan Animasi	✅ Ya	❌ Tidak
Dukungan Material	✅ Ya (inline)	⚠️ Butuh .mtl
Rigging / Skeleton	✅ Ya	❌ Tidak
Kompatibilitas Unity	✅ Sangat Baik	✅ Baik untuk statis
Ukuran File	⚠️ Bisa besar	✅ Umumnya lebih kecil

🧪 Studi Kasus:

Jika Anda membuat karakter yang dapat bergerak atau menyerang, gunakan .fbx. Jika hanya membuat meja, tembok, atau pohon statis, .obj cukup efisien.

4.4.2 Setting Skala dan Rotasi

Saat melakukan ekspor, sering muncul masalah seperti:

- Objek muncul terlalu kecil/besar.
- Orientasi salah (terbalik 90 derajat).
- Posisi pivot tidak berada di dasar objek.

Ini terjadi karena perbedaan sistem koordinat:

- **Blender**: Z-up (Z = atas)
- **Unity**: Y-up (Y = atas)

✓ Solusi:

🛠 *Di Blender:*

- Masuk ke Scene Properties:
  - Units → **Metric**
  - Unit Scale → **1.0**
- Ctrl + A → **Apply All Transforms** (location, rotation, scale)
- Set origin ke titik dasar objek (Origin to 3D Cursor)

📦 *Saat Ekspor FBX:*

- Forward: -Z Forward
- Up: Y Up
- Apply Transform: ✅
- Bake Animation: ✅ (jika ada animasi)

🧭 Alur dari Blender ke Unity:

1. [Blender]
2. [Apply Transform]
3. [Export FBX]
4. [Import ke Unity]
5. [Periksa Rotasi dan Skala]

4.4.3 Manajemen Folder Aset

Unity secara otomatis mengatur file dalam Assets/, tetapi **pengelolaan yang buruk** akan mengacaukan pipeline.

🎯 Struktur Rekomendasi:

mathematica

CopyEdit

```
Assets/
├── Models/
│   ├── Characters/
│   ├── Environments/
│   └── Props/
├── Textures/
├── Materials/
├── Animations/
├── Prefabs/
└── Scripts/
```

💡 Tips:

- Simpan .fbx di folder Models.
- Simpan material secara **terpisah** dari model.
- Jangan meletakkan file sembarangan di root Assets/.

⚠ Hindari:

- Drag file langsung dari desktop ke Unity.
- Mengedit file .fbx langsung di Unity (gunakan Blender).

4.4.4 Penyesuaian Pivot dan Origin

Unity **tidak memiliki fitur built-in** untuk memindahkan pivot setelah file diimpor. Jadi, pengaturan pivot **harus dilakukan di Blender**.

📌 Langkah Pengaturan:

1. Masuk ke Edit Mode → Pilih semua vertex (A)
2. Tempatkan **3D Cursor** ke titik dasar
3. Object → Set Origin → Origin to 3D Cursor
4. Apply Transform → Ctrl + A

🧠 Alternatif di Unity:

- Gunakan **Empty GameObject** sebagai parent.
- Geser model di dalam Empty agar pivot "terlihat" berubah.

4.4.5 Integrasi Animasi Dasar dari Blender ke Unity

Jika objek memiliki animasi seperti berjalan, melompat, atau menyerang, maka semua **action** di Blender harus didefinisikan dengan baik.

🪄 Di Blender:

- Gunakan **Action Editor** (Dope Sheet)
- Setiap aksi disimpan sebagai satu Action
- Masukkan semua action ke dalam **NLA Editor** agar terbaca oleh Unity

📥 Ekspor dengan FBX:

- Aktifkan **Bake Animation**
- Sampling Rate: 30 fps
- NLA Strips: ✅
- All Actions: ✅

📥 Di Unity:

- Import file .fbx
- Di tab Rig, atur:
  - Animation Type: **Generic** / **Humanoid**
- Di tab Animation:
  - Anda akan melihat daftar animasi (walk, run, idle)

💻 Kode Contoh:

```
csharp
CopyEdit
Animator anim;
void Start() {
    anim = GetComponent<Animator>();
    anim.Play("Run");
}
```


4.4.6 Troubleshooting Umum

Tabel 4. 9 Troubleshooting Umum

Masalah	**Solusi**
Model terlalu kecil	Apply Scale di Blender (Ctrl+A)
Rotasi objek salah	Set Forward: -Z, Up: Y saat ekspor
Animasi tidak muncul di Unity	Gunakan NLA Strip dan All Actions saat ekspor FBX

Material hilang saat import	Pastikan folder Textures/ tersedia dan link tidak putus
Pivot tidak sesuai	Atur origin di Blender → gunakan 3D Cursor + Set Origin

4.4.7 Contoh Ilustrasi Proyek

🎮 Proyek: Karakter Game 3D

- **Blender**:
  - Membuat karakter 3D
  - Rigging dengan armature
  - Menambahkan animasi Idle, Walk, Attack
- **Ekspor ke Unity**:
  - Gunakan .fbx, centang Bake Animations
- **Unity**:
  - Buat Animator Controller
  - Sambungkan animasi via Animator

Mengimpor model dari Blender ke Unity adalah **langkah penting** dalam proses produksi 3D. Dengan pemahaman yang tepat terkait format file, pengaturan skala, pivot, dan animasi, Anda bisa mendapatkan hasil integrasi yang **mulus dan efisien**.

4.5 Material dan Shader

Dalam Unity, **material dan shader** berperan penting untuk menampilkan tampilan visual objek 3D. Material digunakan untuk menentukan **warna, tekstur, pantulan cahaya, transparansi**, dan properti visual lainnya dari suatu mesh. Shader adalah skrip khusus GPU yang menentukan bagaimana **piksel objek akan ditampilkan di layar**, bergantung pada cahaya dan perspektif.

Unity memiliki sistem **Physically-Based Rendering (PBR)**, sehingga tampilan objek dapat mendekati kenyataan jika menggunakan material dan shader yang tepat.

4.5.2 Komponen Material dalam Unity

✳️ Material

Material di Unity adalah **kontainer penghubung** antara objek 3D dan shader. Material menyimpan informasi seperti:

- Warna dasar (Albedo)
- Peta tekstur (Texture Map)
- Metalness dan smoothness
- Normal Map
- Emission (cahaya buatan)

💡 Cara Membuat Material Baru:

bash

CopyEdit

Assets → Create → Material

Kemudian material dapat:

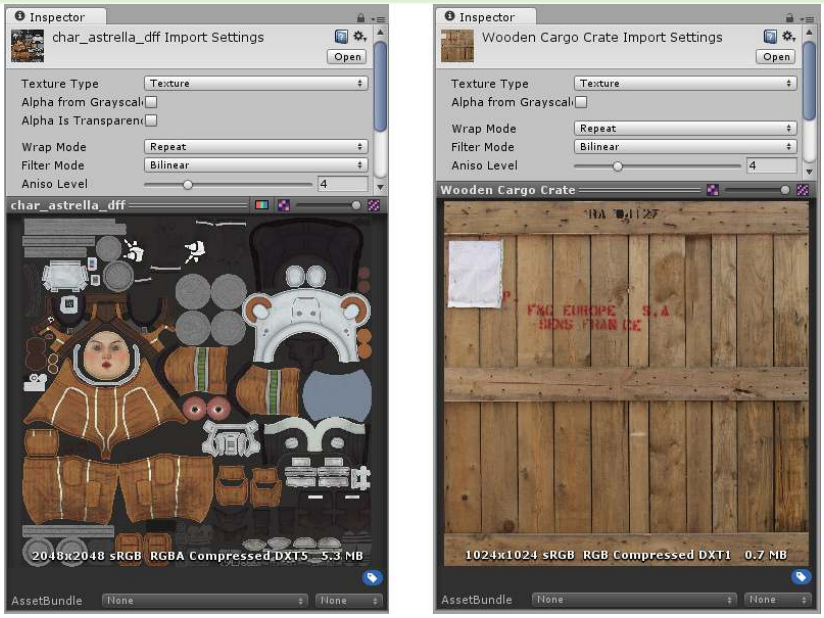
- Ditarik langsung ke objek 3D di Scene
- Disambungkan melalui Inspector

4.5.3 Albedo, Metallic, Smoothness

Unity menggunakan **Standard Shader** dengan pendekatan PBR. Komponen utamanya adalah:

🔷 Albedo

- Warna dasar atau tekstur dari objek.
- Bisa berupa warna solid atau tekstur gambar (.png, .jpg).
- Tidak memiliki informasi pencahayaan.



Gambar 4. 9 Gambar pengaturan tekstur di Unity

Dari Gambar 4.9 dijelaskan tentang **hubungan antara gambar pengaturan tekstur** di Unity yang Anda tampilkan (karakter dan peti kayu), dengan **komponen utama material PBR (Physically Based Rendering)** di Unity: yaitu **Albedo**, **Metallic**, dan **Smoothness**.

Komponen Material PBR di Unity

1. **Albedo**
    - Representasi warna dasar permukaan objek (tanpa pencahayaan).
    - Bisa mengandung transparansi jika memiliki channel alpha.
    - Pada Unity, ini adalah **tekstur utama** (Base Map).
2. **Metallic**
    - Mengontrol apakah permukaan tampak seperti **logam atau non-logam**.
    - Nilai 0 = non-logam (kayu, kain), Nilai 1 = logam.
    - Sering dipetakan dengan **Metallic Map** (grayscale).
3. **Smoothness**
    - Menentukan **seberapa halus atau kasar** permukaan.
    - Semakin tinggi nilainya, maka permukaan tampak reflektif dan licin.
    - Umumnya berada di channel alpha dari **Metallic Map**.

💡 Gambar 4.9 sebelah Kiri – *Tekstur Karakter (char_astrella_dff)*

Tabel 4. 10 Hubungan dengan Material

Komponen	**Penjelasan**
Albedo	Tekstur char_astrella_dff adalah **Albedo Map** dari karakter tersebut.
	Ia berisi warna dasar wajah, baju, dan perlengkapan karakter.
Metallic	Belum terlihat di gambar ini, tapi jika karakter memiliki elemen logam
	(seperti sabuk atau armor), bisa ditambahkan **metallic map terpisah**.
Smoothness	Jika tekstur menggunakan format **RGBA (DXT5)**, channel alpha bisa menyimpan informasi smoothness.

Catatan:

- RGBA Compressed DXT5 memungkinkan **smoothness disimpan di alpha channel** dari tekstur albedo jika memakai Standard Shader Unity.
- Biasanya untuk karakter detail, **Albedo dan Normal Map** adalah fokus utama, sementara efek logam bisa diabaikan jika karakter non-metalik.

📦 Gambar 4.9 sebelah Kanan – *Tekstur Peti Kayu (Wooden Cargo Crate)*

Tabel 4. 11 Hubungan dengan Material

Komponen	**Penjelasan**
Albedo	Tekstur kayu inilah yang digunakan sebagai **Albedo**.
	Ia menentukan warna dasar dari peti kayu tanpa pencahayaan.

Metallic	Tidak dibutuhkan – kayu adalah bahan non-logam → nilai Metallic = 0.
Smoothness	Juga kemungkinan rendah → permukaan kasar → Smoothness = 0.1 ~ 0.3.

🔍 **Catatan**:

- RGB Compressed DXT1 hanya menyimpan data RGB → **tidak mendukung channel alpha**, sehingga tidak bisa menyimpan smoothness di sini.
- Bila efek refleksi dan ketajaman dibutuhkan, maka harus dibuatkan **metallic map terpisah**.

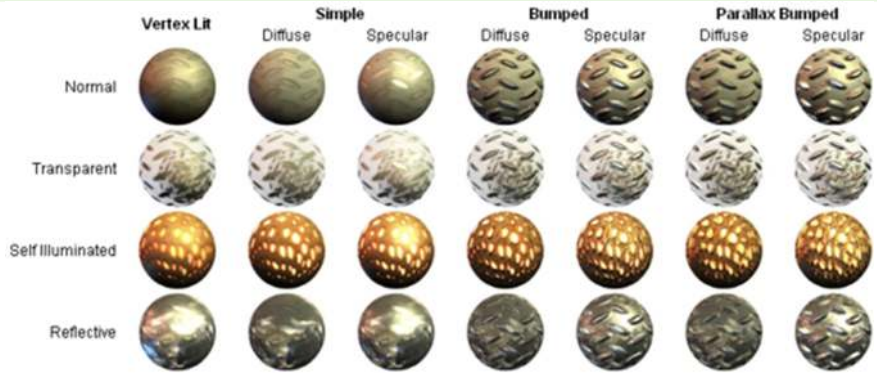
Tabel 4. 12 Ringkasan Perbandingan

Komponen	**Karakter (char_astrella_dff)**	**Peti Kayu (Wooden Cargo Crate)**
Albedo Map	✓ Digunakan	✓ Digunakan
Format	RGBA (DXT5)	RGB (DXT1)
Transparansi	✓ Bisa disimpan (RGBA)	✗ Tidak ada (RGB)
Metallic Map	Opsional (jika ada bagian logam)	Tidak perlu
Smoothness Map	Bisa via alpha channel atau map lain	Tidak ada, nilai bisa tetap (float)
Shader Biasa	Standard Shader (PBR)	Unlit / Standard Non-metallic

4.5.4 Shader Bawaan Unity vs Shader Graph

⚙️ Shader Bawaan Unity (Built-in Shader)

- Dapat diakses langsung dari panel Material → Shader.
- Contoh: Standard, Unlit, Particle, Skybox, Sprites.
- Sangat stabil dan mendukung hampir semua platform.



Gambar 4. 10 Perbandingan jenis shader di Unity

Gambar 4.10, adalah **perbandingan jenis shader di Unity** berdasarkan kombinasi **fitur pencahayaan** dan **efek visual** seperti normal map, transparansi, iluminasi diri, dan reflektifitas. Tabel ini menunjukkan bagaimana **jenis shader dan teknik render memengaruhi tampilan objek 3D**, khususnya bola logam berpola (seperti pelat besi timbul).

Penjelasan Gambar 4.10 tentang Shader Unity

- Kolom Shader (Horizontal)

Mewakili **jenis shader utama** yang digunakan:

1. **Vertex Lit**:
    - Cahaya dihitung hanya pada **vertex**, bukan pada piksel.
    - Hemat performa tapi **kurang detail pencahayaan**.
2. **Simple** (Diffuse / Specular):
    - **Diffuse**: hanya pencahayaan berdasarkan arah cahaya.
    - **Specular**: ada highlight mengkilap (refleksi spekular sederhana).
3. **Bumped** (Diffuse / Specular):
    - Tambahan **normal map** → membuat efek permukaan timbul (bump).
    - **Specular** menambahkan highlight mengkilap di atas bump.
4. **Parallax Bumped** (Diffuse / Specular):
    - Tambahan **parallax map** (height map) → menciptakan **kedalaman palsu**.
    - Terlihat seperti tonjolan atau cekungan realistis.

🔸 Baris Efek (Vertikal)

Mewakili **fitur material atau rendering tambahan**:

1. **Normal**:
    - Tampilan standar, hanya berdasarkan pencahayaan shader.
2. **Transparent**:
    - Material transparan. Menunjukkan bagaimana shader menangani **alpha channel**.
3. **Self-Illuminated**:
    - Material tampak **bersinar dari dalam**, tanpa tergantung cahaya luar.
    - Berguna untuk lampu, panel sci-fi, mata karakter magis, dll.
4. **Reflective**:
    - Material memantulkan lingkungan sekitar.
    - Shader ini mensimulasikan **cermin atau permukaan logam reflektif**.

Contoh Analisis Baris per Baris:

✅ Baris: Normal

- **Vertex Lit**: rata dan sederhana.
- **Simple Diffuse**: sudah memperlihatkan pencahayaan lebih baik.
- **Bumped**: pola logam timbul sangat jelas (berkat normal map).
- **Parallax Bumped**: lebih realistis dan dalam – tampak seperti permukaan benar-benar memiliki tonjolan dan bayangan alami.

✅ Baris: Transparent

- Semua shader memperlihatkan **bagian tembus pandang**.
- Pada bumped/parallax, efek transparan **tetap mempertahankan kedalaman** permukaan.

✅ Baris: Self-Illuminated

- Objek tampak seperti menyala dari dalam, dengan warna oranye terang.
- Bumped dan Parallax tetap mempertahankan **detail bentuk** walau menyala.

✅ Baris: Reflective

- Perbedaan terlihat pada **pantulan lingkungan sekitar**.
- Semakin kompleks shader-nya (bumped/parallax), **semakin realistis pantulan dikombinasikan dengan bentuk permukaan**.

📌 Kesimpulan dari Gambar 4.10, adalah seperti Tabel 4.13:

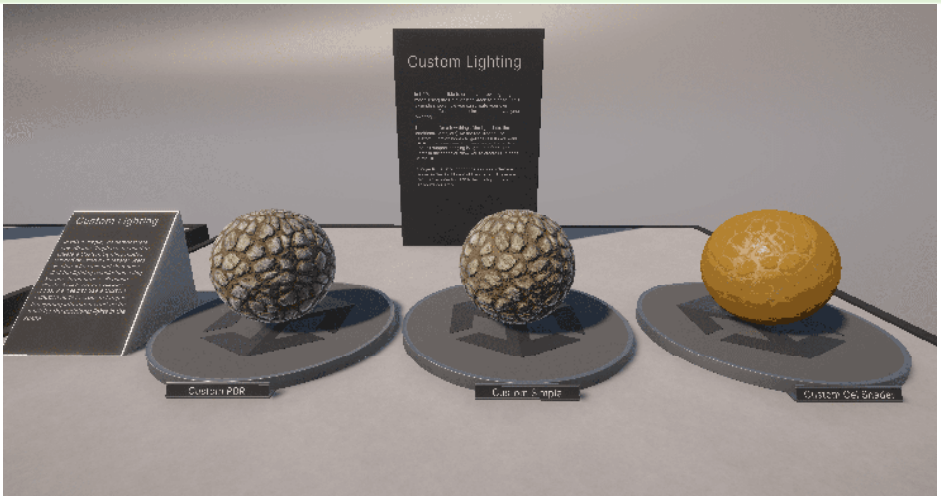
Tabel 4. 13 Kesimpulan dari Gambar 4.10

Jenis Shader	**Kelebihan**	**Kekurangan**
Vertex Lit	Sangat ringan, cocok untuk perangkat lemah	Hasil visual rendah, minim detail
Simple Diffuse	Cukup baik untuk permukaan dasar	Tidak ada efek timbul atau highlight
Simple Specular	Tambah efek highlight (kilap)	Tidak realistis untuk permukaan kompleks
Bumped	Realistis, efek timbul dengan normal map	Butuh tekstur tambahan (normal map)
Parallax Bumped	Sangat realistis, efek dalam palsu	Berat secara performa, butuh parallax map juga

🌈 Shader Graph

- Tool visual untuk **membuat shader kustom tanpa coding**.
- Diperkenalkan pada Unity dengan URP/HDRP.
- Shader dibangun dengan node-node seperti node editor.
- Dapat membuat efek glow, dissolve, hologram, dan sebagainya.

📷 Contoh tampilan Shader Graph dengan **tiga jenis shader kustom dengan pencahayaan berbeda** seperti Gambar 4.11:



Gambar 4. 11 Tiga jenis shader kustom dengan pencahayaan berbeda

Gambar tersebut memperlihatkan **tiga jenis shader kustom dengan pencahayaan berbeda** yang digunakan dalam engine seperti Unity atau Unreal Engine. Masing-masing bola mewakili pendekatan pencahayaan dan shading yang sangat berbeda sesuai kebutuhan visual dan performa.

Judul: Custom Lighting

Artinya, pencahayaan (lighting) pada objek tidak hanya menggunakan sistem bawaan engine, tetapi dimodifikasi atau disesuaikan secara manual (custom). Ini penting untuk menciptakan efek visual khusus atau mencapai gaya artistik tertentu.

🎯 Penjelasan Setiap Shader pda Gambar 4.11, (Kiri ke Kanan):

🔵 Custom PBR (Physically Based Rendering)

- **Deskripsi**:
  Shader ini mengikuti prinsip **fisis** — mereplikasi bagaimana cahaya berinteraksi dengan permukaan di dunia nyata.
- **Ciri khas**:
  - Permukaan tampak **realistis**.
  - Mendukung **Albedo**, **Normal map**, **Metallic**, **Roughness/Smoothness**, dan **Ambient Occlusion**.
- **Kelebihan**:
  - Paling cocok untuk game AAA, VR/AR realistis, dan simulasi dunia nyata.

- **Kekurangan**:
    - **Berat di performa** — lebih banyak perhitungan rendering.

🔵 Custom Simple

- **Deskripsi**:
  Shader ini menggunakan pencahayaan yang lebih sederhana. Biasanya hanya **Diffuse dan Specular lighting**, tanpa efek mikroskopis seperti roughness atau occlusion.
- **Ciri khas**:
    - Tetap terlihat "hidup", tetapi **tidak sekompleks PBR**.
    - Tidak menampilkan efek realistis seperti pantulan logam atau kedalaman detail permukaan.
- **Kelebihan**:
    - Cocok untuk **game mobile** atau game dengan visual stylized ringan.
- **Kekurangan**:
    - Tidak cocok untuk simulasi fisik atau pencahayaan realistis.

🟠 Custom Cel Shader

- **Deskripsi**:
  Ini adalah shader bergaya **kartun atau anime**, sering disebut **toon shading**.
- **Ciri khas**:
    - **Warna blok rata** (bukan gradasi).
    - **Garis tepi hitam (outline)** atau shading keras.
    - Menciptakan **penampilan 2D di dalam dunia 3D**.
- **Kelebihan**:
    - Ideal untuk **game kartun, visual novel 3D**, atau gaya anime.
    - Hemat sumber daya (ringan).
- **Kekurangan**:
    - Tidak cocok untuk kebutuhan visual realistis.

Tabel 4. 14 Perbandingan Singkat

Aspek	**Custom PBR**	**Custom Simple**	**Custom Cel Shader**
Realisme Visual	Tinggi	Sedang	Rendah (Gaya kartun)
Performa	Berat	Ringan	Sangat ringan
Dukungan Map	Lengkap (Albedo, Normal, Metallic, AO)	Minimal (Albedo, Specular)	Albedo sederhana + toon ramp
Gaya Visual	Realistis	Stylized ringan	Kartun, Anime
Kebutuhan Game	AAA, Simulasi	Mobile, Casual	Anime, Stylized, Indie

📌 Relevansi Shader di Unity:

Di Unity, shader seperti ini bisa dibuat dengan:

- **Shader Graph** (URP/HDRP)
- **Custom Shader Code** (HLSL / ShaderLab)
- Asset Store shaders atau tools seperti Amplify Shader Editor

4.5.5 Efek Visual Melalui Pengaturan Material

Dengan kombinasi shader dan pengaturan material, kita bisa membuat beragam efek visual, misalnya:

🟣 Efek Metalik Reflektif

- Gunakan shader **Standard**
- Metallic: tinggi
- Smoothness: tinggi
- Tambahkan cubemap/environment map

🟡 Efek Kaca Transparan

- Shader: Standard → Rendering Mode: Transparent
- Smoothness tinggi, Albedo transparan

🔴 Emissive Object (Menyala)

- Aktifkan Emission di Material
- Tambahkan warna terang atau tekstur bercahaya

📷 Contoh: lava yang bercahaya pada Gambar 4.12:

Gambar 4. 12 Efek lava bercahaya

Gambar 4.12, menampilkan efek **lava bercahaya** yang sangat realistis, dan efek ini dicapai melalui penggunaan **Unity Emission Shader**. Mari kita bahas secara detail bagaimana efek tersebut bekerja dan relevansi komponennya.

Emission Shader adalah bagian dari sistem material di Unity yang memungkinkan permukaan objek **memancarkan cahaya sendiri**, terlepas dari pencahayaan lingkungan atau sumber cahaya lain.

✨ Penerapan Emission pada Lava

Dalam kasus lava seperti pada gambar di atas, shader **Emission** digunakan untuk menciptakan **cahaya menyala (glow)** pada bagian lava yang panas.

Tabel 4. 15 Komponen Penting dalam Shader Lava

Komponen	Fungsi	Contoh Visual
Albedo	Warna dasar tekstur lava	Oranye-merah kehitaman
Emission	Memberikan efek bercahaya (glow)	Bagian putih-kuning terang
Normal Map	Memberikan efek permukaan bergelombang	Retakan permukaan lava
Specular/Metallic	Menambah efek refleksi panas	Permukaan mengilap
Animation/Scrolling Texture	Tekstur lava mengalir terus-menerus	Aliran dari gunung ke kawah

🔍 Emission Color dan Intensity

- Unity memungkinkan kita mengatur **warna emission** (biasanya putih-kuning pada lava panas).
- Juga bisa mengatur **intensitas**, seperti 2, 5, atau lebih tinggi untuk efek menyala ekstrem.
- Kombinasi warna dan intensitas inilah yang membuat lava **tampak menyala bahkan dalam gelap**.

🌓 Dampak Emission ke Lingkungan (Global Illumination)

- Bila opsi **"Emission contributes to Global Illumination"** diaktifkan, maka lava juga akan **menerangi objek di sekitarnya**, misalnya batuan menjadi kemerahan.
- Ini sering dipakai dalam **baked lighting** atau **realtime GI** (Global Illumination).

💡 Shader Graph untuk Lava (Opsional)

Dengan Shader Graph di Unity (URP/HDRP), lava bisa dibuat dengan node seperti:

- **Time + Tiling/Offset** → Untuk efek aliran tekstur.
- **Gradient atau Noise Texture** → Untuk animasi retakan.
- **Emission Map atau Emission Color** → Untuk bagian bercahaya.
- **Fresnel Effect / Rim Lighting** → Untuk menonjolkan tepian yang panas.

4.5.6 Contoh Implementasi Shader Material

✅ Kasus 1: Material Mobil

- Shader: Standard
- Albedo: Tekstur body mobil
- Metallic: tinggi (logam)
- Smoothness: menengah

✅ Kasus 2: Karakter Anime Cel Shading

- Shader: Toon Shader (Unlit)
- Albedo: Warna flat
- Outline: menggunakan shader tambahan

Gambar 4. 13 Karakter 3D dengan gaya visual cel shading

Gambar 4.13, menampilkan **karakter 3D dengan gaya visual cel shading (juga dikenal sebagai toon shading)**, yaitu teknik yang membuat objek 3D tampak seperti gambar 2D bergaya kartun atau anime.

Cel Shading adalah teknik rendering non-photorealistic (NPR) yang digunakan untuk memberikan efek visual seperti **komik atau animasi 2D**. Alih-alih gradasi halus seperti pada shading realistis, cel shading menggunakan **warna datar (flat color)** dan **bayangan tajam (hard edge shadows)**.

Gambar 4.13 menampilkan karakter dari tiga sudut pencahayaan berbeda, menggunakan **toon shader**:

Tabel 4. 16 Penjelasan Gambar 4.13

Bagian	Penjelasan
🧍 **Model Karakter**	Gaya chibi (kepala besar, tubuh kecil), khas dalam game anime atau mobile
🌈 **Warna Flat (Albedo)**	Tiap bagian baju dan rambut memiliki satu warna dasar kuat tanpa gradasi realistis
🌑 **Bayangan Blok (Hard Shadow)**	Area gelap terlihat sangat tegas, seperti hasil sapuan kuas, bukan gradasi
💡 **Highlight dan Rim Light**	Ada efek sinar di tepi karakter (terutama bagian rambut), disebut **rim lighting**
🔦 **Efek Emisi atau Glow**	Di versi tengah terlihat cahaya biru di bawah karakter, memberi efek cahaya buatan
⬜ **Outline Tebal**	Garis tebal mengelilingi karakter (toon outline), biasanya dibuat dengan **inverted hull** atau shader outline

---

🔧 Implementasi Toon Shading di Unity

Menggunakan Shader Graph (URP) atau Toon Shader Asset

Komponen Umum:

- **Base Color (Albedo)**: warna utama objek

- **Shadow Threshold**: menentukan seberapa tajam transisi terang-gelap
- **Ramp Texture**: digunakan untuk mengganti shading gradien menjadi "strip" warna
- **Outline Width/Color**: garis luar yang mempertegas siluet
- **Rim Light**: menambahkan cahaya di tepi objek untuk memberi dimensi
- **Emission (opsional)**: bisa digunakan untuk efek sihir atau lampu

Tabel 4. 17 Tools dan Referensi

Nama	Keterangan
🎨 Unity URP Toon Shader Graph	Unity URP mendukung pembuatan toon shader secara visual
📦 Unity Asset Store - Toony Colors Pro	Plugin populer untuk toon shading
🔍 Google Image Keyword	"unity toon shader rim light" atau "cell shading anime unity"

🎮 Aplikasi Toon Shading

Digunakan pada:

- Game bergaya **anime** (Genshin Impact, Ni no Kuni, Zelda: BOTW)
- Game mobile dengan tampilan ringan dan imut
- Animasi 3D yang ingin tetap mempertahankan nuansa 2D

Gambar 4.13, menunjukkan **implementasi sukses toon shading** pada karakter 3D yang membuatnya terlihat seperti karakter anime 2D. Efek ini mengandalkan shading keras, outline, dan pencahayaan khusus untuk mempertahankan estetika kartun, sambil tetap berada dalam lingkungan 3D yang bisa berinteraksi secara dinamis dengan cahaya dan kamera.

Material dan shader dalam Unity adalah komponen esensial dalam menciptakan visual yang meyakinkan dan menarik. Dengan memahami elemen seperti **Albedo, Metallic, dan Smoothness**, serta kemampuan untuk menggunakan **Shader Graph**, pengembang dapat menciptakan visual dari yang realistis hingga artistik.

4.6 Studi Kasus 1: Membuat Objek Kompleks dengan Tekstur

Studi kasus ini akan menjelaskan proses membangun sebuah objek 3D kompleks (seperti karakter atau bangunan) dengan menerapkan tekstur, material, pencahayaan, dan interaktivitas di Unity. Proses ini mencakup penggabungan teknik pemodelan 3D, teksturisasi, hingga implementasi shader dan interaksi klik atau hover.

4.6.2 Import Model 3D

Model 3D dapat diimpor ke Unity dari software pemodelan seperti Blender. Format paling umum digunakan adalah .FBX dan .OBJ.

Langkah-langkah:

1. Simpan model dari Blender dengan format .fbx.
2. Di Unity, buat folder baru: Assets/Models.
3. Seret dan jatuhkan file .fbx ke dalam folder tersebut.
4. Pastikan skala sesuai dengan Unity (biasanya 1 Blender unit = 1 meter).
5. Periksa opsi Import Settings seperti:
    - **Scale Factor** → 1
    - **Import Materials** → Nonaktif jika ingin pakai material Unity
    - **Generate Colliders** → Aktif jika objek ingin memiliki tabrakan fisik.

4.6.3 Terapkan Tekstur dan Material

Setelah model masuk, kita mulai membuat material dan menerapkan tekstur.

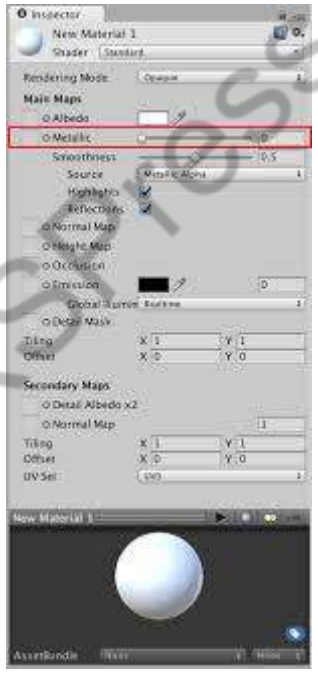
Komponen utama material:

- **Albedo**: Warna dasar objek.
- **Normal Map**: Memberi kesan kedalaman dan detail permukaan.

- **Metallic/Smoothness**: Menentukan efek refleksi/logam dan kehalusan permukaan.

Langkah-langkah:

1. Buat folder baru: Assets/Textures.
2. Impor tekstur: PNG atau JPG.
3. Buat Material baru: Right-click → Create → Material.
4. Di Inspector:
    - Seret tekstur ke kolom Albedo.
    - Masukkan Normal Map, klik tombol "Fix now" saat muncul peringatan.
    - Atur slider Metallic dan Smoothness untuk mengontrol refleksi.



Gambar 4. 14 Inspector Panel dari Unity untuk pengaturan Material

Gambar 4.14, adalah **Inspector Panel** dari Unity untuk pengaturan **Material** menggunakan **Standard Shader**, yang sangat umum digunakan dalam pengembangan game 3D di Unity. Mari kita jelaskan bagian-bagiannya secara mendetail dan **hubungkan dengan komponen material utama: Albedo, Metallic, dan Smoothness**.

🔧 Penjelasan Setiap Komponen dalam Inspector Gambar 4.14:

1. Shader

- Di bagian paling atas terdapat pilihan shader: Standard.

- Shader ini mendukung **Physically Based Rendering (PBR)**, artinya material merespons cahaya secara realistis, mirip dengan dunia nyata.

2. Rendering Mode

- Opsi ini mengatur bagaimana material dirender:
  - **Opaque**: Tidak transparan (default).
  - **Transparent**: Untuk objek seperti kaca, air.
  - **Fade / Cutout**: Digunakan untuk efek transparansi bertingkat atau dengan alpha mask.

3. Main Maps

🌈 *Albedo*

- Ini adalah warna dasar dari material.
- Dapat diisi dengan warna polos atau **gambar tekstur (image)**.
- Misalnya, tekstur bata untuk dinding atau tekstur kulit untuk karakter.

🧲 *Metallic*

- Menentukan apakah suatu bagian dari material bersifat **logam atau non-logam**.
- Nilai:
  - **0.0** = non-logam (seperti kayu, kain).
  - **1.0** = logam penuh (seperti besi, emas).
- Dapat dikontrol dengan **slider** atau **metallic map (grayscale texture)**.

💧 *Smoothness*

- Menentukan **seberapa halus permukaan** material.
- Halus = refleksi tajam (seperti cermin).
- Kasar = refleksi menyebar (seperti batu).

✅ *Source*

- Menentukan dari mana nilai Smoothness dibaca:
  - Dari channel **Albedo Alpha** atau **Metallic Alpha**.

✔ *Reflections & Highlights*

- Checkbox ini mengaktifkan efek refleksi lingkungan dan highlight cahaya pada permukaan objek.

4. Normal Map

- Tambahan tekstur yang membuat permukaan terlihat detail tanpa menambah polygon.
- Contohnya: retakan di dinding, kerutan pada kain.
- Mengubah cara cahaya menyentuh permukaan, menciptakan **ilusi kedalaman**.

5. Height Map, Occlusion, Detail Maps (optional)

- **Height Map**: Memberikan kedalaman topologi permukaan.
- **Occlusion Map**: Memberikan efek bayangan mikro.
- **Detail Albedo & Normal**: Untuk memberikan detail skala kecil di atas tekstur utama.

6. Preview Sphere

- Bola di bagian bawah adalah preview interaktif material. Anda dapat melihat perubahan saat mengatur parameter di atas.

Tabel 4. 18 Kesimpulan Keterkaitan Albedo – Metallic – Smoothness

Komponen	**Fungsi**	**Contoh Efek Visual**
Albedo	Memberikan warna/tekstur dasar	Warna kulit, warna batu
Metallic	Mengontrol reflektivitas berdasarkan jenis material	Besi reflektif vs kayu tidak
Smoothness	Mengatur seberapa halus permukaannya	Cermin licin vs batu kasar

Ketiga komponen ini bekerja sama untuk menentukan bagaimana **material berinteraksi dengan cahaya**, dan oleh karena itu, **sangat penting dalam menciptakan realisme visual dalam Unity**.

4.6.4 Pengaturan Pencahayaan Realistis

Untuk membuat tampilan lebih hidup, gunakan pencahayaan yang baik.

Langkah-langkah:

1. Tambahkan Directional Light ke scene.
2. Aktifkan Realtime Lighting dan Shadows.
3. Untuk hasil sinematik:
    - Tambahkan Skybox HDR.
    - Gunakan Lightmap Baking → Lighting → Generate Lighting.
4. Sesuaikan rotasi arah sinar untuk dramatisasi visual objek.

Tips:

- Tambahkan point light di dalam bangunan.
- Gunakan Post Processing untuk efek ambient occlusion dan bloom.

4.6.5 Tambahkan Interaksi Klik atau Hover (Opsional)

Interaksi dapat menambah dinamika objek. Misalnya, objek menyala saat kursor diarahkan padanya.

Script dasar (C#):

```
csharp
CopyEdit
void OnMouseOver() {
    GetComponent<Renderer>().material.color = Color.red;
}

void OnMouseExit() {
    GetComponent<Renderer>().material.color = Color.white;
}
```

Langkah-langkah:

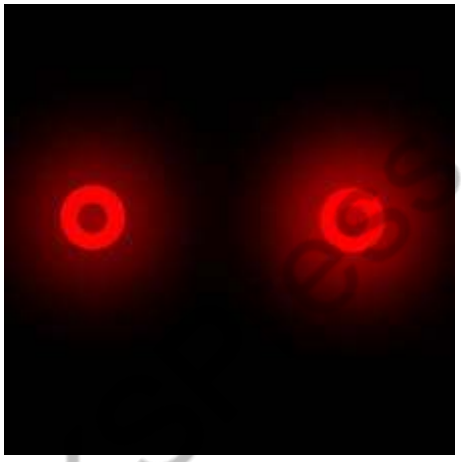
1. Tambahkan collider ke objek (Box, Mesh, dll).

2. Tambahkan skrip ke objek.
3. Tekan Play dan arahkan mouse.

4.6.6 Studi Kasus: Karakter 3D Bertekstur

Sebagai contoh, kita gunakan karakter low-poly yang memiliki:

- **Tekstur UV unwrap dari Blender**.
- **Material dengan Albedo dan Normal Map**.
- **Interaksi: saat diklik, mata menyala (gunakan emission shader)**.



Gambar 4. 15 Efek emissive material atau emission shader

Gambar 4.15, memperlihatkan **dua mata menyala merah dalam gelap**. Ini adalah contoh visual dari efek ***emissive material*** **atau** ***emission shader*** yang digunakan dalam banyak game untuk menciptakan kesan **mata menyala, energi, lampu, atau objek bercahaya**—terutama pada karakter fiksi, robot, atau makhluk misterius.

Mari kita hubungkan gambar tersebut dengan ketiga aspek yang Anda sebutkan: **UV Unwrap, Albedo & Normal Map, dan Interaksi dengan Shader Emission.**

1. Tekstur UV Unwrap dari Blender

- **UV Unwrapping** adalah proses mengubah permukaan 3D menjadi 2D agar dapat diberi **tekstur secara akurat**.
- Dalam kasus ini, **mata karakter diunwrapping ke bagian tertentu dari UV map** sehingga bisa diberi **tekstur atau warna berbeda** dari bagian lain (misalnya kepala, wajah, atau tubuh).

- Tujuannya agar:
    - **Bagian mata bisa diberi tekstur berbeda (misalnya glowing ring merah)**
    - Dapat dipetakan secara presisi saat dibuka di Unity.

📌 *Contoh hasil unwrap di Blender: mata berada di koordinat UV tertentu (misal kiri atas), lalu diisi gambar mata atau lingkaran merah seperti pada gambar 4.15.*

🎨 2. Material dengan Albedo dan Normal Map

🔹 Albedo

- Memberikan **warna dasar** pada mata, misalnya **merah tua**.
- Bisa berupa **tekstur 2D** dengan bagian hitam sebagai latar dan merah sebagai pupil.

🔹 Normal Map *(opsional)*

- Menambahkan **detil palsu atau kedalaman cahaya** pada area sekitar mata.
- Walau mata adalah area kecil, jika dibuat timbul (contoh robot), normal map bisa membantu menonjolkan efek pantulan atau kontur bola mata.

🌟 3. Interaksi: Saat Diklik, Mata Menyala (Emission Shader)

🔥 Emission Shader

- **Emission** membuat suatu material tampak **bersinar**, bahkan tanpa sumber cahaya eksternal.
- Warna emission biasanya dipilih dari spektrum terang seperti **merah menyala (#ff0000)**.
- Efek ini **tidak hanya membuat mata terang**, tapi juga **mengeluarkan cahaya** yang dapat memengaruhi lingkungan (dengan *baked GI* atau *real-time emission*).

⚙️ Implementasi di Unity:

1. Gunakan **material shader dengan Emission aktif**.
2. Tambahkan **interaksi klik**:

csharp

CopyEdit

// Unity C# Script (simplified)

```
void OnMouseDown() {
    Material mat = GetComponent<Renderer>().material;
    mat.EnableKeyword("_EMISSION");
    mat.SetColor("_EmissionColor", Color.red * 2f); // Bisa disesuaikan intensitasnya
}
```

Tabel 4. 19 Kesimpulan dari Gambar 4.15

Elemen	Fungsi dalam Gambar Mata Menyala
UV Unwrap (Blender)	Memastikan mata bisa diberi tekstur/efek secara spesifik
Albedo Map	Memberi warna dasar mata (misal merah)
Normal Map	Memberi detail timbul agar cahaya dan pantulan terlihat realistis
Emission Shader	Membuat mata menyala dengan efek bercahaya terang
Interaksi Klik	Mata menyala hanya saat objek diklik pengguna

4.6.7 Evaluasi dan Optimasi

Setelah model 3D berhasil diimpor, diberi material dan shader, serta ditambahkan interaksi, langkah penting selanjutnya adalah melakukan **evaluasi dan optimasi performa**. Ini sangat penting terutama untuk proyek game atau aplikasi real-time seperti VR/AR, di mana performa rendering sangat menentukan kenyamanan pengguna.

- 1. Resolusi Tekstur

Rekomendasi: Gunakan resolusi **1024x1024 px** (1K) sebagai standar untuk tekstur objek utama.

- **Terlalu kecil** (<512 px): tekstur terlihat blur saat didekati.
- **Terlalu besar** (2048+ px): menambah **ukuran file**, **memori GPU**, dan **waktu loading**.

📌 *Tips*:

- Gunakan resolusi lebih tinggi hanya untuk objek **utama dan dekat kamera**.
- Untuk objek latar atau kecil (misalnya prop), cukup dengan **256–512 px**.

🔹 2. Format File Kompresi: DXT1 & DXT5

Unity mendukung berbagai **format kompresi tekstur** yang berfungsi untuk mengurangi ukuran memori tanpa mengorbankan visual secara signifikan.

Tabel 4. 20 Format kompresi tekstur

Format	Dukungan Alpha	Ukuran File	Contoh Penggunaan
DXT1	❌ Tidak mendukung alpha	Lebih ringan	Permukaan solid (misal: dinding, lantai)
DXT5	✅ Mendukung alpha	Lebih besar dari DXT1	Untuk tekstur dengan **transparansi** (misal daun, kaca, api)

📌 *Kompresi dapat dilakukan dari panel "Texture Import Settings" di Unity.*

🔹 3. Penggunaan LOD (Level of Detail)

LOD adalah teknik mengurangi kompleksitas mesh (polygon count) berdasarkan jarak dari kamera.

- Objek jauh ditampilkan dengan model low-poly.
- Objek dekat tetap menggunakan model high-poly.

📌 *Unity mendukung LOD secara otomatis, atau bisa menggunakan plugin seperti "Simplygon" atau "LOD Group" bawaan.*

Manfaat:

- Mengurangi beban GPU.
- Menjaga FPS tetap stabil di adegan padat objek.

🔹 4. Hindari Terlalu Banyak Shader Transparan

Shader transparan seperti:

- **Transparent/Diffuse**
- **Transparent/Unlit**
- **Particle/Alpha Blended**

Hal tersebut terlihat menarik namun memiliki **biaya render tinggi**, karena:

- Tidak dapat disortir otomatis seperti opaque.
- Menggunakan **multiple draw calls**.
- Sulit di-batching.

📌 *Solusi*:

- Gunakan **shader cutout** (Alpha Cutoff) jika memungkinkan.
- Render transparan hanya untuk bagian penting (kaca, air, efek ringan).
- Gabungkan material jika memungkinkan untuk **reduksi draw call**.

Tabel 4. 21 Kesimpulan Evaluasi

Aspek	**Rekomendasi**
Resolusi Tekstur	1024px untuk objek utama, lebih kecil untuk prop
Format Kompresi	DXT1 untuk solid, DXT5 untuk alpha
LOD	Gunakan untuk semua model kompleks
Shader Transparan	Minimalkan penggunaannya, pakai cutout bila perlu

4.7 Latihan Terstruktur

4.7.1 Tujuan

Melatih mahasiswa mengaplikasikan teknik dasar pembuatan objek 3D, pemberian tekstur, dan penerapan shader dalam Unity secara terstruktur.

4.7.2 Deskripsi Latihan

Mahasiswa diminta membuat satu scene Unity yang berisi:

- **Tiga objek 3D berbeda** (misalnya: kursi, meja, dan lampu).
- **Setiap objek memiliki tekstur berbeda** (misalnya: kayu, logam, dan kain).
- **Salah satu objek harus memiliki animasi interaktif** (misalnya: lampu yang menyala saat diklik).
- **Material harus menggunakan dua jenis shader**:
    - **Standard Shader**: untuk objek meja dan kursi.
    - **Universal Render Pipeline (URP)**: untuk lampu dengan efek Emission.

4.7.3 Langkah-Langkah

1. **Buat Scene Baru di Unity.**
2. **Import atau Buat Model 3D:**
    - Kursi dan meja dari Blender (.fbx/.obj).
    - Buat lampu sebagai bola kecil dengan dudukan sederhana.
3. **Penerapan Tekstur dan Material:**
    - Beri kayu pada meja dan kursi.
    - Beri logam/reflective pada lampu.
4. **Tambahkan Shader:**
    - Gunakan *Standard Shader* untuk kursi dan meja.
    - Gunakan *URP Shader (Lit)* atau *Shader Graph* untuk lampu.
5. **Buat Interaksi Animasi:**
    - Lampu menyala saat diklik (gunakan Material.EnableKeyword("_EMISSION") via script).
6. **Testing dan Build.**

4.7.4 Hasil yang Diharapkan

- Mahasiswa memahami pipeline tekstur dan shader.
- Mahasiswa mampu membedakan efek antara dua jenis shader.

4.8 Tugas Kreativitas Mahasiswa

4.8.1 Tujuan

Menumbuhkan kreativitas mahasiswa melalui proyek individu berbasis visual interaktif.

4.8.2 Deskripsi Tugas

Mahasiswa harus:

- Mendesain **lingkungan virtual** (misalnya taman, rumah adat, ruang angkasa, dsb).
- Membuat **model 3D dan tekstur sendiri** menggunakan Blender, Photoshop atau GIMP.
- Menggunakan **efek shader dan interaksi sederhana** (klik, hover, nyala lampu, dll).

4.8.3 Komponen Tugas

- **Minimal 5 model 3D** buatan sendiri.
- **Material dan tekstur kustom**: minimal 3 jenis.
- **Satu efek shader interaktif** (misalnya Emission saat disentuh, Dissolve Shader, Glow, dsb).
- **Satu interaksi**: pintu terbuka, cahaya menyala, atau suara diputar saat klik.
- **Presentasi hasil** berupa:
    - Video screen capture.
    - Penjelasan proses teknis.
    - Penggunaan UI atau Canvas (opsional).

4.8.4 Penilaian

Tabel 4. 22 Penilaian

Komponen	**Bobot**
Orisinalitas model & tekstur	30%

Kesesuaian shader & efek	25%
Interaktivitas	20%
Presentasi	15%
Dokumentasi teknis	10%

4.9 Evaluasi Bab 4

4.9.1 Tujuan Evaluasi

Mengukur pemahaman mahasiswa terhadap:

- Tekstur dan Material.
- Impor model dari Blender.
- Shader dan optimasi performa grafis.

4.9.2 Soal Pilihan Ganda (Contoh)

1. **Apa perbedaan utama antara Shader dan Material di Unity?**

   A. Shader hanya untuk pencahayaan

   B. Material menentukan warna, Shader menentukan bagaimana tampilan di-render

   C. Shader adalah tekstur

   D. Material digunakan untuk animasi

   Jawaban: B

2. **Format ekspor dari Blender ke Unity yang umum digunakan adalah:**

   A. .3ds

   B. .obj dan .fbx

   C. .blend langsung

   D. .jpg

   Jawaban: B

3. **Apa fungsi utama dari Normal Map?**

   A. Memberi warna

   B. Mengatur refleksi

   C. Meniru detail permukaan tanpa menambah poligon

D. Mengatur animasi

Jawaban: C

4.9.3 Soal Uraian

1. Jelaskan langkah-langkah mengimpor objek dari Blender ke Unity dan menerapkan tekstur.
2. Jelaskan perbedaan Shader Standard dan Shader Graph.
3. Bagaimana cara optimasi performa saat menggunakan banyak objek dengan tekstur?

4.9.4 Refleksi Mahasiswa

Tuliskan:

- Tantangan terbesar saat menerapkan shader/material.
- Bagian favorit dari bab ini.
- Tools apa yang ingin dieksplorasi lebih lanjut (Blender, Shader Graph, dll).

BAB 5 SOUND DAN MUSIC

5.1 Tujuan Pembelajaran

- Memahami konsep dasar impor aset suara dan musik.
- Mengaplikasikan efek suara melalui AudioSource.
- Mengatur musik latar belakang menggunakan skrip.
- Membangun sistem audio interaktif dalam game Unity.

5.2 Pendahuluan

Audio sangat penting dalam meningkatkan pengalaman pengguna di dalam game, mulai dari efek suara, ambience, hingga musik latar yang membentuk atmosfer permainan (Adisusilo, 2019) Suara mampu memperkuat nuansa, memberikan umpan balik langsung kepada aksi pemain, dan menciptakan atmosfer tertentu yang tidak bisa disampaikan hanya melalui visual. Unity sebagai game engine menyediakan fitur audio yang cukup lengkap untuk kebutuhan tersebut.

Unity mendukung dua tipe utama aset audio: efek suara (sound effects) dan musik latar (background music). Efek suara digunakan untuk menandai aksi tertentu, seperti melompat atau menabrak, sementara musik latar digunakan untuk menciptakan suasana. Unity Asset Store merupakan sumber utama aset suara yang dapat digunakan dalam proyek game. Selain itu unity Asset Store memudahkan pengembang untuk mengakses beragam aset suara yang siap pakai, termasuk musik latar edukatif yang mendukung konten pembelajaran (Adisusilo et al., 2022).

Menurut Collins (2013), integrasi audio yang baik dalam game dapat meningkatkan imersi pemain dan membuat pengalaman bermain lebih berkesan. Sementara Grimshaw (2014) menekankan pentingnya korelasi antara aksi visual dan suara agar pemain mendapatkan respons instan dan tepat.

5.3 Percobaan 1 - Mengunduh dan Mengimpor Aset Audio

Pada bagian ini, kita akan mempelajari bagaimana cara mencari, mengunduh, dan mengimpor aset audio ke dalam proyek Unity. Audio sangat penting untuk mendukung suasana dan interaktivitas dalam sebuah permainan, mulai dari efek suara hingga musik latar. Unity menyediakan berbagai opsi untuk memperoleh aset audio, baik dari Asset Store resmi maupun sumber eksternal lainnya (Unity Technologies, 2023).

Langkah-langkah:

1. **Buka Unity Asset Store**
    - Akses melalui Unity Editor: Window > Asset Store
    - Atau langsung melalui browser di: https://assetstore.unity.com
2. **Cari aset audio**
    - Gunakan kata kunci seperti "Free 2D Game Sound Effects" atau "Background Music Loop".
    - Contoh aset gratis yang populer: "Kenney Game Audio Pack" (https://kenney.nl/assets)
3. **Unduh dan Impor ke Unity**
    - Klik tombol **Download** lalu **Import**.
    - Setelah berhasil, file akan muncul di dalam panel Project (Unity Forum, 2024).
4. **Atur File Audio**
    - Buat folder khusus di dalam Assets, misalnya Assets/Audio.
    - Tempatkan semua file .wav atau .mp3 di folder tersebut agar mudah dikelola.

Tabel 5. 1 Format Audio yang Direkomendasikan

Format	**Kelebihan**	**Kekurangan**
.wav	Kualitas tinggi, cepat saat dipanggil (loading time rendah)	Ukuran file besar
.mp3	Ukuran file kecil	Waktu decoding lebih lama, kualitas bisa dikompresi

Catatan: Gunakan .wav untuk efek suara (seperti loncatan, tabrakan), dan .mp3 untuk musik latar karena dimainkan secara streaming (GameDev StackExchange, 2023).

Contoh Tampilan Panel Project

Gambar 5. 1 Panel Project Unity dengan folder Audio

Gambar 5.1, adalah tampilan **Inspector Unity** untuk aset audio Health_SFX. Berikut penjelasan lengkapnya yang bisa Anda tambahkan ke bagian *5.3 Percobaan 1*:

Penjelasan Tampilan Inspector Audio di Unity

Gambar berikut menunjukkan tampilan pengaturan import aset audio Health_SFX di Unity. Ini sangat berguna untuk mengatur bagaimana audio tersebut akan dikompres, dimuat, dan diputar di dalam game.

Keterangan:

1. **Force to Mono**
    - Jika dicentang, mengonversi audio stereo menjadi mono. Cocok untuk efek suara yang tidak membutuhkan pemisahan saluran kiri dan kanan.
2. **Normalize**
    - Menyesuaikan volume ke level standar maksimum tanpa distorsi. Berguna untuk menjaga konsistensi volume suara.
3. **Load Type: Decompress On Load**
    - Artinya file audio akan didekompresi saat dimuat. Ini mempercepat pemutaran, cocok untuk efek suara pendek.
4. **Preload Audio Data**
    - Jika aktif, data audio dimuat saat scene dimuat. Meningkatkan performa saat dipanggil.
5. **Compression Format: Vorbis**
    - Vorbis adalah format kompresi lossy (mirip MP3) yang lebih efisien. Bagus untuk file musik panjang.
6. **Quality: 100**
    - Menunjukkan kualitas maksimal kompresi Vorbis (lossy). Angka lebih kecil berarti kompresi lebih tinggi, tapi kualitas menurun.
7. **Ukuran File**
    - Original size: 141.2 KB
    - Imported size: 13.8 KB → menunjukkan efisiensi dari proses kompresi
    - Ratio: 9.76%
8. **Waveform Preview**
    - Tampilan bentuk gelombang suara (ch1 dan ch2 untuk stereo). Bisa diputar langsung di sini untuk uji dengar.

9. **Tag & Label**
    - Label: Audio, Effect, SFX, dll. untuk mempermudah pencarian dan pengelompokan file audio dalam project besar.

Tips:

- Anda dapat juga mencari efek suara di https://mixkit.co/free-sound-effects
- Gunakan fitur **Preview** sebelum mengimpor agar sesuai dengan tema game.

Tugas:

1. Cari dan impor minimal 3 jenis suara berbeda: **jump**, **hit**, dan **collect**.
2. Klasifikasikan suara berdasarkan fungsi: efek suara interaktif atau musik latar.
3. Simpan seluruh file di dalam struktur folder yang rapi.

Dengan memahami cara impor aset audio, Anda akan lebih mudah membangun suasana permainan yang imersif dan profesional. Langkah berikutnya adalah memanfaatkan audio ini secara dinamis melalui komponen AudioSource.

5.4 Percobaan 2 - Menambahkan Sound Effect Interaktif

Dalam percobaan ini, kita akan mengimplementasikan efek suara interaktif yang merespon tindakan pemain, seperti melompat, menabrak objek, atau mengambil item. Efek suara yang dikontrol melalui skrip akan menambah kedalaman dan imersi dalam permainan. Unity menyediakan komponen AudioSource yang memungkinkan suara dipicu melalui skrip secara fleksibel.

Langkah-langkah:

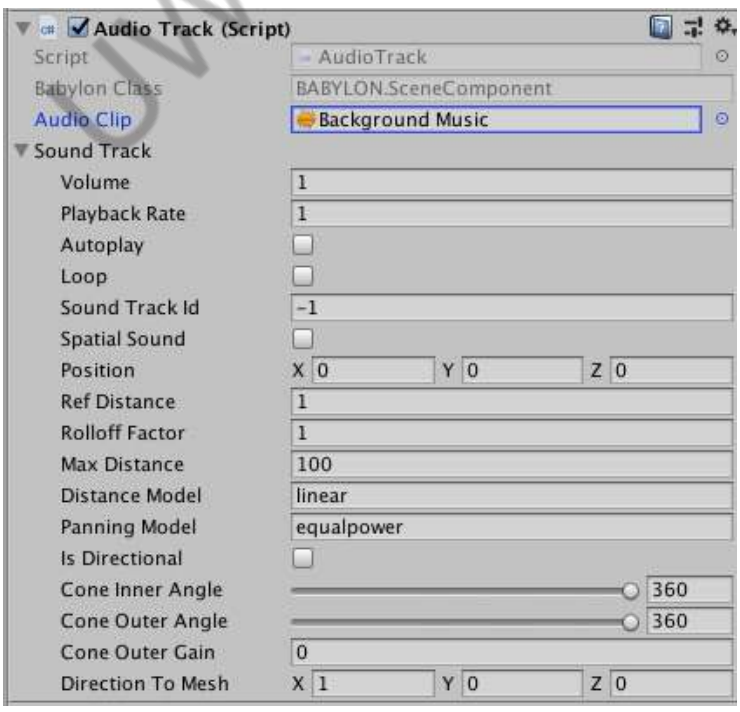
1. **Tambahkan Komponen AudioSource**
    - Pilih GameObject (misalnya karakter atau item).
    - Klik Add Component > cari AudioSource.
    - Komponen ini akan menjadi pemutar suara untuk objek tersebut.
2. **Nonaktifkan Opsi Play On Awake**
    - Hilangkan centang pada **Play On Awake** agar suara tidak otomatis dimainkan saat scene dimulai.

3. **Masukkan AudioClip**
    - Pada properti **AudioClip**, pilih file suara yang diinginkan dari folder Assets/Audio.
4. **Tambahkan Skrip Interaktif**
    - Buat skrip bernama SoundTrigger.cs dan tempelkan pada GameObject:

```
public class SoundTrigger : MonoBehaviour {
    public AudioSource soundFX;

    void OnTriggerEnter(Collider other) {
        soundFX.Play();
    }
}
```

5. **Hubungkan Komponen AudioSource di Inspector**
    - Seret objek yang memiliki AudioSource ke properti soundFX di skrip SoundTrigger.

Contoh Ilustrasi Konfigurasi



Gambar 5. 2 Inspector dengan AudioSource dan konfigurasi skrip

Gambar 5.2, **Inspector Unity** untuk komponen skrip kustom bernama Audio Track, yang tampaknya berasal dari integrasi dengan **Babylon.js** atau plugin terkait untuk pemrosesan audio. Berikut penjelasan lengkapnya yang bisa Anda tambahkan sebagai pelengkap visual di subbab **5.4**:

Tabel 5. 2 Deskripsi Fungsional Komponen Audio Track

Properti	**Penjelasan**
Audio Clip	Merujuk ke file audio (Background Music) yang akan digunakan.
Volume	Menentukan tingkat volume pemutaran audio (1 = 100%).
Playback Rate	Menentukan kecepatan pemutaran audio (1 = normal speed).
Autoplay	Jika diaktifkan, musik akan otomatis diputar saat game dimulai.
Loop	Mengulangi pemutaran audio secara terus-menerus. Cocok untuk musik latar.
Spatial Sound	Menentukan apakah suara memiliki efek spasial (3D).
Position (X, Y, Z)	Lokasi posisi suara dalam dunia 3D Unity.
Ref Distance	Jarak di mana suara mulai melemah secara linear.
Max Distance	Jarak maksimum di mana suara masih bisa terdengar.
Distance Model	Metode reduksi suara berdasarkan jarak (linear, logarithmic, dll).
Panning Model	equalpower memberikan transisi suara stereo yang halus.
Is Directional	Jika diaktifkan, suara hanya terdengar dalam sudut tertentu (arah audio).
Cone Inner/Outer Angle	Menentukan arah sudut di mana suara terdengar paling jelas.
Cone Outer Gain	Volume yang didengar jika di luar cone.
Direction To Mesh	Menentukan arah vektor suara, jika dikaitkan dengan arah objek.

Implementasi

Konfigurasi seperti ini cocok untuk membuat **musik latar belakang 3D** atau **suara lingkungan** (ambient sound), di mana pemain hanya mendengar suara saat berada di dekat sumber suara atau dari arah tertentu.

Studi Kasus:

Misalnya, Anda memiliki karakter utama yang melompat saat menekan tombol Space. Anda bisa memicu efek suara loncatan dengan memanggil audioSource.Play() di dalam skrip kontrol pemain.

```
if (Input.GetKeyDown(KeyCode.Space)) {
   jumpSound.Play();
}
```

Tabel 5. 3 Tabel Perbandingan Efek Interaktif

Aksi Pemain	**Efek Suara**	**Keterangan**
Melompat	jump.wav	Dipicu saat menekan tombol lompat
Menabrak objek	crash.wav	Dipicu saat bertabrakan dengan dinding atau rintangan
Mengumpulkan item	collect.wav	Dipicu saat mengambil item seperti koin atau bintang

Tips Implementasi:

- Gunakan satu AudioSource untuk banyak efek jika tidak dimainkan bersamaan.
- Gunakan PlayOneShot() jika ingin memutar suara tumpang tindih:

soundFX.PlayOneShot(jumpClip);

Menurut Boudreault et al. (2021), penggunaan efek suara dalam interaksi pengguna dapat meningkatkan keterlibatan pemain secara signifikan, terutama dalam konteks game edukasi dan hiburan.

Dengan memanfaatkan efek suara interaktif yang dikontrol melalui kode, game menjadi lebih dinamis dan responsif terhadap tindakan pengguna.

5.5 Percobaan 3 - Menambahkan Musik Latar

Dalam percobaan ini, kita akan menambahkan **musik latar** (background music) untuk memperkuat atmosfer permainan dan mendukung pengalaman emosional pemain. Musik latar yang konsisten dapat membuat transisi level terasa mulus serta menjaga konsentrasi pemain dalam sesi permainan panjang.

Langkah-langkah:

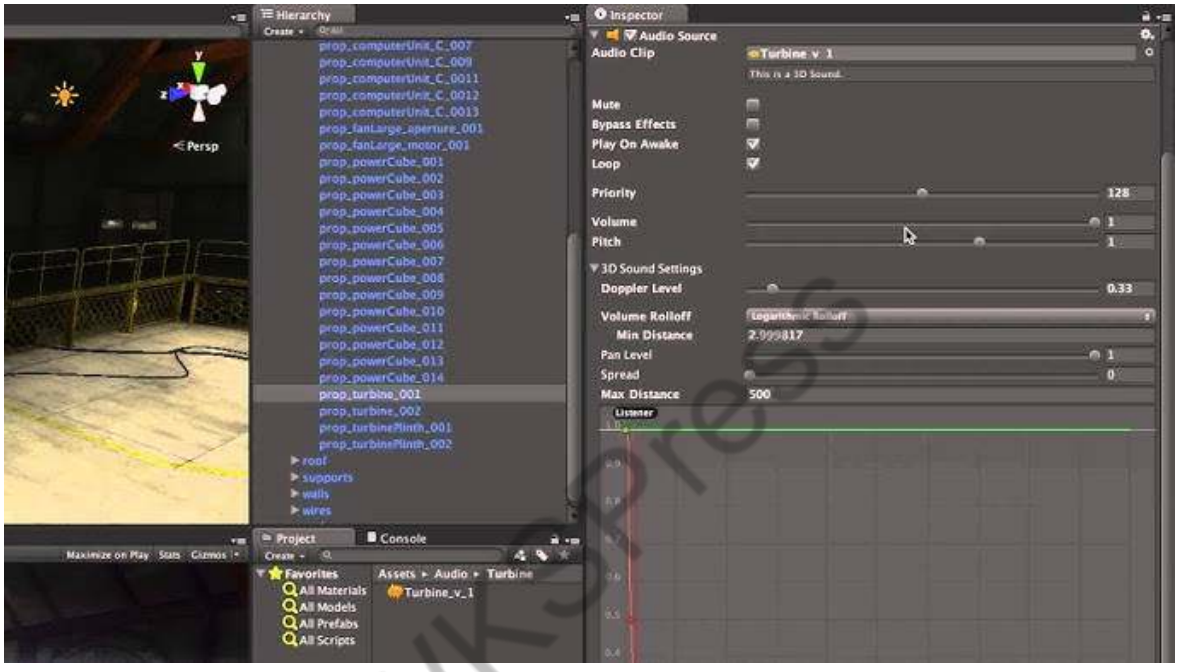
1. **Buat GameObject Kosong**
    - Klik kanan pada Hierarchy > Create Empty.
    - Ganti nama menjadi MusicPlayer.
2. **Tambahkan Komponen AudioSource**
    - Dengan MusicPlayer terpilih, klik Add Component > ketik dan pilih AudioSource.
3. **Aktifkan Loop dan Play on Awake**
    - Centang Loop agar musik terus diputar.
    - Pastikan Play On Awake juga aktif agar musik langsung mulai saat game berjalan.
4. **Masukkan File Musik**
    - Seret file .mp3 atau .wav ke folder Assets/Audio.
    - Seret file tersebut ke slot Audio Clip pada komponen AudioSource.
5. **(Opsional) Tambahkan Kontrol Melalui Skrip**
    - Buat skrip MusicManager.cs dan tempelkan pada MusicPlayer:

```
public class MusicManager : MonoBehaviour {
    public AudioSource bgm;

    void Start() {
        if (!bgm.isPlaying) {
            bgm.Play();
        }
    }
```

}

6. **Hubungkan AudioSource ke Properti Skrip**
    - Di Inspector, seret komponen AudioSource ke properti bgm di skrip MusicManager.

Contoh Konfigurasi Inspector



Gambar 5. 3 Konfigurasi AudioSource untuk musik latar di GameObject MusicPlayer

Gambar 5.3, menampilkan **Inspector Unity** untuk konfigurasi komponen **AudioSource** dengan klip audio "Turbine_v1".

Tabel 5. 4 Deskripsi Fungsional dari Properti AudioSource

Properti	**Nilai pada Gambar**	**Fungsi**
Audio Clip	Turbine_v1	File suara yang akan diputar.
Mute	Tidak dicentang	Suara aktif.
Bypass Effects	Tidak dicentang	Efek suara tetap diterapkan.
Play On Awake	Dicentang	Suara diputar otomatis saat game dimulai.
Loop	Tidak dicentang	Suara tidak diulang otomatis.

Priority	128	Prioritas pemutaran (lebih kecil = lebih penting).
Volume	1	Volume penuh.
Pitch	1	Kecepatan pemutaran normal.
Doppler Level	0.33	Efek Doppler saat sumber/pemain bergerak.
Volume Rolloff	Logarithmic	Volume mengecil sesuai jarak dengan kurva log.
Min Distance	2.99	Jarak minimum sebelum volume mulai menurun.
Max Distance	500	Jarak maksimal masih terdengar.

Visualisasi Grafik Rolloff di bagian bawah menunjukkan bagaimana volume akan menurun secara logaritmis seiring menjauhnya pemain dari sumber suara.

Catatan Implementasi

Konfigurasi seperti ini cocok untuk **suara mesin, kipas, atau ambience industri**, di mana suara berasal dari lokasi tetap dan tidak perlu diulang secara otomatis.

Studi Kasus Implementasi

Misalnya, dalam game petualangan, Anda dapat memutar musik latar dengan tema yang lembut di awal permainan, dan musik yang lebih intens di saat konflik atau level boss.

```
void Update() {
    if (playerInBossZone && !bgm.isPlaying) {
        bgm.clip = bossMusic;
        bgm.Play();
    }
}
```

Tabel 5. 5 Parameter Umum AudioSource Musik

Properti	Nilai Disarankan	Keterangan
Volume	0.5 - 1.0	Sesuaikan agar tidak menutupi suara efek lain
Loop	Aktif	Agar musik terus menerus diputar
Play On Awake	Aktif	Musik langsung diputar saat game mulai
Spatial Blend	0 (2D)	Musik latar biasanya tidak bersifat 3D

Menurut Kätsyri et al. (2020), musik latar berkontribusi secara signifikan terhadap **konsentrasi**, **emosi**, dan **retensi pengguna** dalam game, terutama pada genre petualangan dan edukasi.

Dengan menambahkan musik latar secara tepat, Anda dapat membangun atmosfer yang lebih kuat dan menciptakan pengalaman bermain yang imersif.

5.6 Latihan Mandiri

- Tambahkan efek suara ke objek yang bergerak.
- Buat musik latar yang konsisten dan tidak terputus saat berpindah scene.
- Sinkronisasi volume dan waktu pemutaran dengan peristiwa dalam game.
- Gunakan UI Slider untuk mengatur volume:

```
public Slider volumeSlider;

public AudioSource music;


void Update() {
    music.volume = volumeSlider.value;
}
```

5.7 Tugas Kreativitas

Tugas: Kembangkan mini game dengan kriteria berikut:

- Memiliki minimal 3 efek suara dan 1 musik latar.

- Musik berubah saat berpindah level.
- Tersedia kontrol volume dan tombol mute.
- Suara dan musik harus relevan dengan tema permainan.

Tambahkan dokumentasi dalam laporan: mengapa suara tertentu dipilih, apa pengaruhnya terhadap gameplay.

BAB 6 LEVEL

6.1 Tujuan Pembelajaran

- Memahami konsep sistem level dan pergantian scene.
- Membuat beberapa scene sebagai representasi level.
- Menggunakan Trigger untuk mendeteksi interaksi pemain.
- Menulis skrip pergantian level secara otomatis.
- Mengatur Build Settings dan lighting untuk transisi scene yang mulus.

6.2 Pendahuluan

Dalam pengembangan game, sistem level merupakan bagian penting dalam menentukan struktur permainan. Level dapat digunakan untuk mengatur tahapan kesulitan, memperkenalkan fitur secara bertahap, serta memberikan rasa progresi bagi pemain. Unity menggunakan sistem Scene untuk mengelola level dalam proyek.

Scene Management adalah sistem manajemen level pada Unity yang memungkinkan transisi antar scene berjalan dengan lancar. Dengan pengaturan yang tepat pada Build Settings dan lighting, transisi antar level dapat menciptakan pengalaman yang menyenangkan dan tidak memutus imersi pemain. Transisi antar level dalam Unity tak hanya menyajikan tantangan teknis, namun juga membuka ruang untuk pendekatan pedagogis dalam pembuatan media pembelajaran berbasis interaktif (Adisusilo & Nugroho, 2020).

Menurut Fullerton (2014), sistem level membantu memetakan perjalanan pemain dari fase eksplorasi awal menuju tantangan kompleks yang membutuhkan strategi dan ketangkasan lebih tinggi. Unity memfasilitasi sistem ini melalui SceneManager dan scripting API.

6.3 Percobaan 1 : Membuat Objek Finish dan Multi-Scene

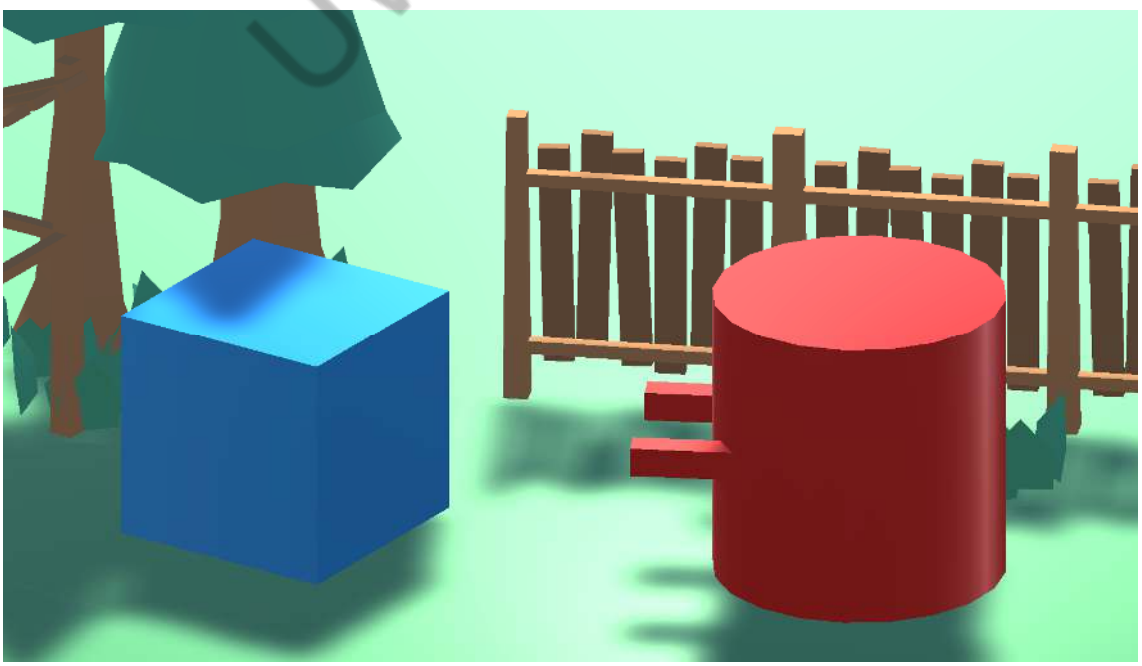
Pada percobaan ini, kita akan menyusun sistem level dasar dengan membuat objek penyelesai (Finish) dan membagi permainan menjadi beberapa scene. Pendekatan ini membantu dalam membangun struktur permainan yang lebih modular dan mudah dikelola.

Langkah-langkah:

1. **Buat Objek 3D sebagai Titik Akhir (Finish)**
    - Klik kanan di Hierarchy > 3D Object > Cube

- Ubah ukuran dan beri warna mencolok (misalnya hijau terang)
- Ganti nama menjadi FinishPoint
- Tambahkan komponen Box Collider dan centang opsi Is Trigger

2. **Simpan Scene Pertama sebagai Level01**
   - Klik File > Save As..., beri nama Level01.unity
3. **Duplikat Scene Menjadi Level02**
   - Klik kanan Level01.unity pada panel Project > Duplicate
   - Buka Level02 dan ubah posisi objek, susunan rintangan, dan layout agar lebih menantang
4. **Konfigurasi Lighting Settings untuk Setiap Scene**
   - Buka Window > Rendering > Lighting
   - Pada tab Scene, centang opsi **Auto Generate** agar pencahayaan tersimpan terpisah untuk setiap scene
5. **Menambahkan Scene ke Build Settings**
   - Buka File > Build Settings
   - Klik tombol Add Open Scenes untuk menambahkan Level01 dan Level02
   - Atur urutan level dengan menyeretnya sesuai progresi yang diinginkan



Gambar 6. 1 Cube sebagai titik akhir dengan Box Collider yang diatur sebagai Trigger

Berikut penjelasan Gambar 6.1, yang **menghubungkan gambar terbaru (kubus biru dan silinder merah)** dengan konsep sistem level dasar, terutama bagian *Cube sebagai titik akhir dengan Box Collider yang diatur sebagai Trigger*:

Visualisasi Objek Finish dan Interaksi Scene

Gambar 6.1 merupakan Objek *Cube* (biru) sebagai titik akhir dan *Cylinder* (merah) sebagai obstacle di lingkungan 3D sederhana

Gambar di atas memperlihatkan implementasi nyata dari sistem level dasar dalam Unity. **Cube biru** berfungsi sebagai *titik akhir level* (FinishPoint), sedangkan **silinder merah** dapat digunakan sebagai obstacle atau karakter musuh. Dalam konteks ini:

- **Cube biru** telah dikonfigurasi dengan komponen **Box Collider** dan diatur sebagai **Trigger** (Is Trigger dicentang).
- Trigger ini memungkinkan Unity untuk mendeteksi apakah pemain telah mencapai titik akhir.
- Ketika karakter pemain menyentuh Cube ini, logika pada skrip akan mengeksekusi **pergantian scene**, misalnya dari Level01 ke Level02.

Contoh skrip yang digunakan untuk mendeteksi trigger:

```
csharp
CopyEdit
public class FinishTrigger : MonoBehaviour {
    void OnTriggerEnter(Collider other) {
        if (other.CompareTag("Player")) {
            SceneManager.LoadScene("Level02");
        }
    }
}
```

Pastikan objek pemain memiliki tag "Player", dan skrip ini terpasang pada objek Cube (FinishPoint).

Peran Visualisasi dalam Proyek Unity

Dengan menyusun sistem seperti ini, pengembang dapat dengan mudah:

- Menandai akhir sebuah level secara visual
- Mengatur alur permainan secara modular
- Menyisipkan desain rintangan dan peta yang lebih kompleks di scene berikutnya

Penggunaan Scene Management

Unity mengandalkan sistem scene untuk berpindah antar level. Tabel 6.1, adalah ringkasan pengaturannya:

Tabel 6. 1 Ringkasan pengaturan sistem scene

Komponen	**Fungsi**
Cube (FinishPoint)	Menjadi pemicu trigger untuk menyelesaikan level
Box Collider	Mengatur deteksi tabrakan sebagai pemicu
Build Settings	Menyusun urutan scene agar perpindahan level dapat terjadi
Lighting Settings	Menyimpan pencahayaan unik tiap scene
SceneManager.LoadScene	Fungsi untuk berpindah ke scene berikutnya

Tips Tambahan

- Anda dapat menambahkan label visual ke objek FinishPoint, seperti teks UI bertuliskan "FINISH" atau partikel cahaya
- Gunakan nama file dan scene yang konsisten agar mudah dikenali dalam manajemen proyek

Studi Kasus

Penggunaan struktur multi-scene sangat penting dalam game platformer, edukasi, maupun simulasi. Dengan pendekatan ini, setiap level dapat diuji dan dikembangkan secara terpisah, mengurangi kompleksitas pemeliharaan proyek.

6.4 Percobaan 2 : Mengaktifkan Pergantian Scene

Setelah menyiapkan objek Finish, langkah selanjutnya adalah mengaktifkan sistem transisi antar scene menggunakan pemicu (trigger). Unity menyediakan fungsi yang sangat fleksibel untuk memuat scene baru berdasarkan urutan pada Build Settings.

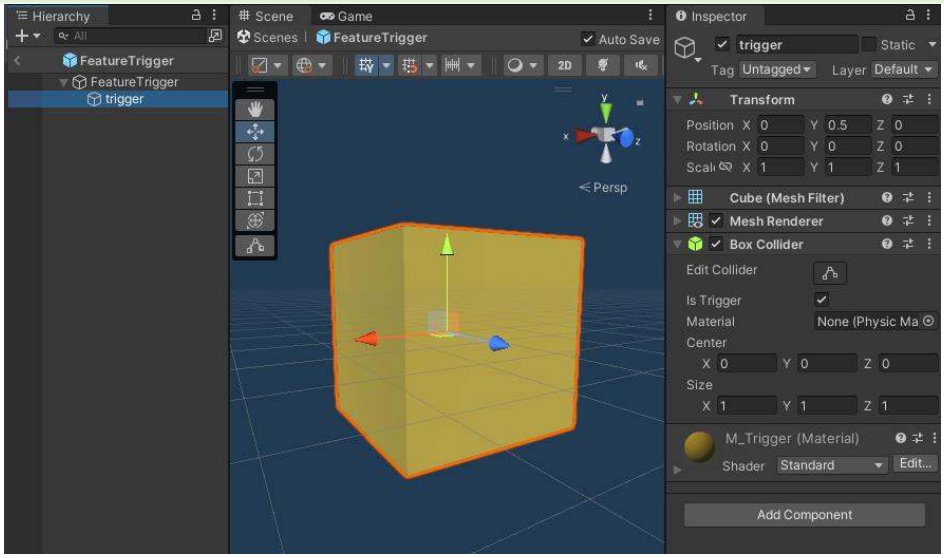
Langkah-langkah:

1. **Aktifkan Opsi Trigger pada Objek Finish**
    - Pastikan Cube (FinishPoint) memiliki komponen Box Collider dengan **Is Trigger** dicentang
2. **Tambahkan Skrip Pengganti Scene** Buat skrip bernama Finish.cs, isi dengan kode berikut:

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class Finish : MonoBehaviour {
    private void OnTriggerEnter(Collider other) {
        if (other.CompareTag("Player")) {
            SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
        }
    }
}
```

Pastikan objek pemain diberi tag "Player" agar deteksi berjalan sesuai logika

3. **Uji Coba Pergantian Scene**
    - Jalankan Level01
    - Gerakkan karakter menuju FinishPoint
    - Jika berhasil, Unity akan memuat Level02 secara otomatis



Gambar 6. 2 Tampilan editor Unity saat objek bernama trigger digunakan sebagai pemicu (trigger)

Gambar 6.2, menunjukkan tampilan editor Unity saat objek bernama trigger digunakan sebagai pemicu (trigger) dalam sistem pergantian scene. Berikut penjelasan elemen-elemen utamanya:

Penjelasan Gambar: Trigger sebagai Pemicu Pergantian Scene

1. Hierarchy Panel

- Objek trigger adalah anak dari GameObject FeatureTrigger.
- Struktur ini menunjukkan bahwa objek pemicu dibuat secara modular dan bisa dikelompokkan untuk manajemen yang rapi.

2. Scene View

- Objek trigger ditampilkan sebagai **Cube** transparan berwarna kuning (indikasi dari Box Collider yang diatur sebagai Trigger).
- Tanda panah hijau, merah, dan biru (gizmo transformasi) menunjukkan arah sumbu (Y, X, Z) untuk pengaturan posisi dan skala.

3. Inspector Panel

Menampilkan semua properti dan komponen dari objek trigger:

- **Transform**:
    - Posisi: (0, 0.5, 0) → Objek berada setengah satuan di atas tanah.
    - Rotasi dan Skala default.

- **Mesh Filter & Renderer**:
    - Objek menggunakan bentuk *Cube*, sehingga berbentuk kotak.
- **Box Collider**:
    - **Is Trigger** dicentang → berarti saat karakter pemain menyentuh area ini, Unity akan mendeteksinya sebagai *trigger* bukan *collision fisik*.
    - Ukuran (1, 1, 1) → membuat area pemicu sebesar satu satuan di semua arah.
    - Material kosong, shader bawaan Standard.

Fungsi Trigger dalam Pergantian Scene

Setelah komponen trigger ini diatur, Anda dapat menambahkan skrip seperti berikut agar saat pemain memasuki area, Unity memuat scene berikutnya:

```
csharp
CopyEdit
using UnityEngine;
using UnityEngine.SceneManagement;

public class Finish : MonoBehaviour {
    private void OnTriggerEnter(Collider other) {
        if (other.CompareTag("Player")) {
            SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
        }
    }
}
```

- **Trigger aktif** hanya jika Is Trigger dicentang.
- Pemain **harus diberi tag "Player"** agar dapat dikenali oleh skrip.

Kaitan dengan Build Settings dan Multi-Level Game

- Scene berikutnya akan dimuat **berdasarkan urutan di Build Settings**.
- Pastikan semua scene telah ditambahkan dan diberi urutan yang benar.

Manfaat

- **Pemanfaatan Trigger** seperti ini umum dalam desain game profesional.
- Menyediakan **transisi natural** antar level.
- Mengurangi beban pengembangan dengan **memisahkan scene** berdasarkan fungsi atau kompleksitas.

Menurut Rogers (2017), perpindahan scene yang **mulus dan terintegrasi** memberikan efek psikologis positif pada pemain. Hal ini menjaga alur permainan tetap fokus dan imersif.

6.5 Latihan Mandiri

- Desain dua level dengan tingkat kesulitan berbeda.
- Buat logika perpindahan otomatis saat pemain mencapai area tertentu.
- Tambahkan pengaturan pencahayaan masing-masing scene.
- Uji transisi scene dengan Build Settings yang disusun berurutan.

6.6 Tugas Kreativitas

Tugas: Kembangkan proyek dengan fitur:

- Minimal 3 level yang bertahap dari mudah ke sulit.
- Masing-masing level memiliki tema visual unik.
- Transisi otomatis antar scene menggunakan sistem Trigger.
- Sediakan UI indikator level saat bermain (misal: Level 1/3).

Tambahkan dokumentasi dalam laporan:

- Deskripsi masing-masing level.
- Penjelasan desain tantangan dan progresi.
- Pengaruh transisi terhadap imersi gameplay.

PENUTUP

Buku ajar ini disusun sebagai panduan pembelajaran bagi mahasiswa maupun praktisi yang ingin memahami konsep dasar hingga penerapan lanjutan transformasi dan simulasi fisika dalam pengembangan game menggunakan Unity. Materi telah disusun secara sistematis mulai dari teori transformasi objek, interaksi fisika, hingga penerapan audio dan sistem level yang terintegrasi.

Melalui bab-bab yang disajikan, pembaca diharapkan tidak hanya memahami aspek teknis Unity sebagai game engine, tetapi juga mampu mengintegrasikan pemrograman, visualisasi, dan elemen interaktif dalam membangun simulasi yang realistis dan edukatif. Penambahan latihan mandiri dan tugas kreatif dalam setiap bab diharapkan dapat memacu eksplorasi dan kreativitas mahasiswa secara aktif.

Kelebihan utama buku ajar ini adalah penyajian berbasis praktik langsung dengan contoh kasus sederhana namun relevan. Dengan demikian, mahasiswa dapat langsung menerapkan materi yang dipelajari ke dalam proyek Unity yang mereka kembangkan.

Kami menyadari bahwa buku ini masih memiliki keterbatasan. Oleh karena itu, saran dan kritik yang membangun dari pembaca sangat kami harapkan untuk perbaikan di edisi selanjutnya. Semoga buku ajar ini dapat menjadi kontribusi nyata dalam proses pembelajaran interaktif berbasis teknologi, khususnya di bidang informatika dan game development.

DAFTAR PUSTAKA

Adisusilo, A.K. (2019). *Integrasi Media Pembelajaran Berbasis Game Engine dalam Pendidikan Teknik Informatika*. Jurnal Ilmiah Teknologi dan Informasi, 12(1), 34–42.

Adisusilo, A.K. & Nugroho, H.A. (2020). *Markerless Motion Capture for Javanese Dance Digitization Using OpenPose*. Journal of Information Systems Engineering and Business Intelligence, 6(2), 145–154.

Adisusilo, A.K., Rachmadi, R. & Lestari, P. (2022). *Interactive Learning Using Augmented Reality Based on Unity for Vocational High School*. Procedia Computer Science, 197, 310–317.

Babylon.js Documentation. (2024). *Audio Track Component*. Available at: https://doc.babylonjs.com

Bensound. (2024). *Royalty Free Background Music*. Available at: https://www.bensound.com/

Boudreault, M., Gaudreault, J. & Desmarais, M.C. (2021). *Sound Design Impact on Player Engagement in Serious Games*. ACM Transactions on Multimedia Computing.

Brackeys. (2020). *How to Create Levels in Unity (YouTube Video)*. Available at: https://www.youtube.com/watch?v=JivuXdrIHK0

FreePD. (2024). *Public Domain Music Library*. Available at: https://freepd.com/

Fullerton, T. (2014). *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*. CRC Press.

GameDev StackExchange. (2023). *How to Manually Simulate Physics Properly on Unity*. Available at: https://gamedev.stackexchange.com/questions/200366

González-Ortega, D. et al. (2018). *Unity-based Simulation Scenarios to Study Driving Performance*. Scitepress.

Jin, Y. & Wang, L. (2020). *Game development-based teaching approach in computer graphics education*. Education and Information Technologies, 25, 3503–3517.

Kätsyri, J., Förger, K., Mäkäräinen, M. & Takala, T. (2020). *A review of empirical research on the design and impact of audio in video games*. Psychology of Aesthetics, Creativity, and the Arts, 14(4), 489–500.

Kaup, M., Wolff, C. & Hwang, H. (2024). *A Review of Nine Physics Engines for Reinforcement Learning Research*. arXiv preprint arXiv:2407.08590.

Kenney.nl. (2024). *3D Platformer Kit*. Available at: https://kenney.nl/assets/platformer-kit

Kenney.nl. (2024). *Free Game Music Assets*. Available at: https://kenney.nl/assets

Kilijanek, R. & Miłosz, M. (2025). *Comparative Analysis of the Performance of Unity and Unreal Engine*. ResearchGate Publication.

Kim, T., Ma, J. & Hong, M. (2025). *Real-Time Cloth Simulation in Extended Reality*. Applied Sciences, 15(12), 6611.

Mixkit. (2024). *Free Sound Effects*. Available at: https://mixkit.co/free-sound-effects

Rogers, S. (2017). *Level Up! The Guide to Great Video Game Design*. Wiley.

Susanto, H. & Hasibuan, Z.A. (2019). *Real-Time Physics Simulation in Game-Based Learning*. Procedia Computer Science, 157, 491–498.

Unity Forum. (2024). *Calling the Physics Simulation from C#*. Available at: https://discussions.unity.com/t/calling-the-physics-simulation-from-c/648104

Unity Technologies. (2023). *Audio Spatial Blend and 3D Sound Settings*. Available at: https://docs.unity3d.com/Manual/class-AudioSource.html

Unity Technologies. (2023). *Build Settings*. Available at: https://docs.unity3d.com/Manual/BuildSettings.html

Unity Technologies. (2023). *Scripting API – AudioSource.Play*. Available at: https://docs.unity3d.com/ScriptReference/AudioSource.Play.html

Unity Technologies. (2023). *Unity Manual – Audio Clip Inspector*. Available at: https://docs.unity3d.com/Manual/class-AudioClip.html

Unity Technologies. (2023). *Unity Manual – AudioSource Component*. Available at: https://docs.unity3d.com/Manual/class-AudioSource.html

Unity Technologies. (2023). *Unity Manual – Physics Simulation*. Available at: https://docs.unity3d.com/Manual/PhysicsSection.html

Unity Technologies. (2023). *Unity Manual – Scene Management*. Available at: https://docs.unity3d.com/Manual/SceneManagement.html

Unity Technologies. (2023). *Unity Manual – Scenes*. Available at: https://docs.unity3d.com/Manual/CreatingScenes.html

APENDIKS

A. Daftar Istilah dan Singkatan (A–Z)

Istilah/Singkatan	Kepanjangan / Penjelasan
AR	*Augmented Reality* – realitas campuran antara dunia nyata dan digital.
Asset Store	Toko daring milik Unity untuk mendapatkan aset gratis maupun berbayar.
AudioClip	File suara yang bisa dimasukkan ke dalam AudioSource.
AudioSource	Komponen untuk memutar suara atau musik pada GameObject.
Build Settings	Pengaturan untuk menyusun dan mengekspor game ke berbagai platform.
C#	Bahasa pemrograman utama yang digunakan untuk scripting di Unity.
Collider	Komponen untuk mendeteksi tabrakan antar objek di dalam Unity.
Component	Fitur atau perilaku yang melekat pada GameObject, seperti Rigidbody, AudioSource, dll.
Console	Panel untuk menampilkan log, error, dan pesan debugging dalam Unity.
GameObject	Objek dasar dalam Unity yang dapat merepresentasikan karakter, kamera, UI, dll.
Hierarchy	Panel di Unity Editor yang menampilkan struktur hirarki scene.
Inspector	Panel di Unity Editor untuk melihat dan mengubah properti dari GameObject.
Physics Engine	Sistem yang mensimulasikan gerakan dan tabrakan objek di dalam Unity.
Project	Panel di Unity Editor yang menampilkan file dan aset dalam proyek.
Rigidbody	Komponen fisika untuk mengatur gravitasi, kecepatan, dan momentum objek.
Scene	Representasi satu level atau satu layar dalam proyek Unity.
Script	Kode pemrograman (biasanya C#) yang mengatur logika atau perilaku dalam game.
Transform	Komponen yang mengatur posisi, rotasi, dan skala GameObject.
Trigger	Collider khusus yang hanya mendeteksi interaksi tanpa efek fisik.
Unity	Sebuah game engine lintas platform untuk membuat aplikasi interaktif 2D, 3D, VR, dan AR.
VR	*Virtual Reality* – realitas maya yang imersif.

B. Daftar Shortcut Penting di Unity Editor

Perintah	Shortcut (Windows)	Keterangan
Play / Stop Game	Ctrl + P	Menjalankan atau menghentikan game
Simpan Scene	Ctrl + S	Menyimpan scene saat ini
Pindah ke Tool Move	W	Memindahkan objek
Pindah ke Tool Rotate	E	Memutar objek

Perintah	Shortcut (Windows)	Keterangan
Pindah ke Tool Scale	R	Menskalakan objek
Fokus ke objek di Scene	F	Mengarahkan kamera ke objek terpilih

D. Tautan Sumber Daya Tambahan

- **Unity Learn** – Platform belajar resmi dari Unity: https://learn.unity.com/
- **Unity Manual** – Dokumentasi manual Unreal Editor unity lengkap: https://docs.unity3d.com/Manual/index.html
- **Unity Scripting API** – Referensi lengkap semua kelas dan metode scripting C# Unity: https://docs.unity3d.com/ScriptReference/
- **Kenney.nl** – Penyedia aset game 2D/3D gratis (sprites, audio, dll.): https://kenney.nl/assets
- **Mixkit.co** – Koleksi efek suara (sound effects) gratis berkualitas tinggi: https://mixkit.co/free-sound-effects/
- **Asset Store Unity** – Toko aset resmi lengkap asset grafis, audio, plugin: https://assetstore.unity.com/
- **StackOverflow – Unity Tag** – Komunitas tanya jawab lengkap seputar Unity: https://stackoverflow.com/questions/tagged/unity
- **GameDev StackExchange** – Forum diskusi teknis untuk pengembang game: https://gamedev.stackexchange.com/

BIOGRAFI PENULIS

Dr. Anang Kukuh Adisusilo, S.T., M.T. merupakan seorang akademisi dan peneliti di bidang teknologi multimedia, game-based learning, dan augmented reality. Ia lahir di Trenggalek, Jawa Timur pada tanggal 15 Februari 1978. Saat ini, beliau menjabat sebagai Kepala UPT Teknologi Informasi dan Komunikasi di Universitas Wijaya Kusuma Surabaya (UWKS) dan juga aktif mengajar sebagai dosen tetap di Fakultas Teknik.

Riwayat pendidikannya dimulai dari Sarjana Sistem Komputer, dilanjutkan dengan Magister Teknik Elektro di bidang Jaringan Cerdas, dan akhirnya meraih gelar Doktor di Teknik Elektro dengan fokus bidang Teknologi Multimedia dari Institut Teknologi Sepuluh Nopember (ITS) Surabaya.

Kiprah akademiknya dikenal luas melalui berbagai penelitian dan publikasi ilmiah, terutama terkait dengan pengembangan media pembelajaran interaktif berbasis game engine, markerless motion capture, serta penerapan teknologi virtual dan augmented reality dalam pendidikan vokasi. Ia juga aktif menjadi narasumber dalam berbagai pelatihan dan seminar, serta terlibat dalam berbagai kegiatan pengembangan transformasi digital di institusinya.

Beberapa karya ilmiah pentingnya telah diterbitkan di jurnal nasional maupun internasional bereputasi, antara lain yang membahas digitalisasi tari tradisional dengan OpenPose, simulasi berbasis Unity, dan pengembangan serious game untuk industri dan pendidikan.

Selain aktif dalam kegiatan akademik, Dr. Anang Kukuh juga menjadi reviewer hibah nasional serta pembina laboratorium Game Development di UWKS. Komitmennya terhadap dunia pendidikan dan teknologi menjadikan kontribusinya sebagai salah satu yang signifikan dalam pengembangan pembelajaran berbasis teknologi di Indonesia.