

|  |
| --- |
| MODUL 4 |
| Komputer Grafik 2D  TRANSFORMASI 2D (Translasi, SCALE & BaSIC Animasi  D4 Teknik Informatika  Jurusan Teknik Komputer dan Informatika  POLITEKNIK NEGERI BANDUNG |
|  |
| Muhammad samudera bagja 058 | KOMPUTER GRAFIK | FEB, 05 2025 |

Contents

[Konsep TRANSFORMASI 0](#__RefHeading___Toc433_198171685)

[TRANSLASI 0](#__RefHeading___Toc435_198171685)

[SCALING 2](#__RefHeading___Toc437_198171685)

[ROTASI 3](#__RefHeading___Toc439_198171685)

[HOMOGENEOUS MATRIKS 7](#__RefHeading___Toc441_198171685)

[PENGUMPULAN 13](#__RefHeading___Toc443_198171685)

# Konsep TRANSFORMASI

Text

Description automatically generated

# TRANSLASI

Text, letter

Description automatically generated

Graphical user interface, text, application

Description automatically generated

A picture containing shape

Description automatically generated

# SCALING

Text

Description automatically generated with medium confidence

Text

Description automatically generated

Text

Description automatically generated

Text, letter

Description automatically generated

# ROTASI

Text, letter

Description automatically generated

Text, table

Description automatically generated

Text

Description automatically generated

Text, letter

Description automatically generated

Diagram

Description automatically generatedA picture containing text, antenna

Description automatically generated

# HOMOGENEOUS MATRIKS

Text, letter

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text, letter

Description automatically generated

Text, letter

Description automatically generated

A picture containing diagram

Description automatically generated

A picture containing table

Description automatically generated

A picture containing diagram

Description automatically generated

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

Task 0: Analisa TrANSFORMASI

**Tugas Task 0: Buat Lesson Learnt untuk pertanyaan-pertanyaan berikut**

1. Bedah code Transformasi.cs pada file template yang diberikan oleh Dosen
2. Adaptasi Transformasi.cs pada code praktikum modul 3 anda
3. Apa yang anda pahami tentang homogenous matriks / coordinates
4. Apa yang anda pahami dari fungsi transformPoints
5. Mana yang lebih efisien melakukan transformPoints untuk semua list vector sebuah bentuk, atau cukup melakukan transformPoints, titik-titik pembangun bentuk?

|  |
| --- |
| Lesson Learnt |
| **1. Bedah Code Transformasi.cs**  File Transformasi.cs berisi implementasi transformasi geometri 2D menggunakan matriks 3x3. Berikut adalah penjelasan dari beberapa bagian penting dalam kode tersebut:  **a. Matrix Operations**   * **Matrix3x3Identity**: Membuat matriks identitas 3x3. * **Matrix3x3Summation**: Menambahkan dua matriks 3x3. * **Matrix3x3Subtraction**: Mengurangkan dua matriks 3x3. * **Matrix3x3Multiplication**: Mengalikan dua matriks 3x3.   **b. Geometric Transformations**   * **Translation**: Melakukan translasi (pergeseran) pada titik dengan menambahkan nilai x dan y ke koordinat titik tersebut. * **Scaling**: Melakukan scaling (penskalaan) pada titik dengan mengalikan koordinat titik dengan faktor skala x dan y. * **RotationClockwise** dan **RotationCounterClockwise**: Melakukan rotasi searah jarum jam dan berlawanan arah jarum jam pada titik dengan sudut tertentu. * **Shearing**: Melakukan shearing (geser) pada titik dengan faktor geser xdan y. * **ReflectionToX**, **ReflectionToY**, dan **ReflectionToOrigin**: Melakukan refleksi terhadap sumbu X, sumbu Y, dan titik origin.   **c. GetTransformPoint**   * Fungsi ini mengambil daftar titik (List<Vector2>) dan mengaplikasikan transformasi matriks pada setiap titik tersebut. Hasilnya adalah daftar titik yang telah ditransformasi.   **2. Adaptasi Transformasi.cs pada Code Praktikum Modul 3**  Untuk mengadaptasi Transformasi.cs ke dalam code praktikum modul 3, perlu memastikan bahwa fungsi-fungsi transformasi yang ada di Transformasi.cs dapat digunakan untuk memanipulasi bentuk-bentuk geometri yang telah dibuat. Berikut adalah langkah-langkahnya:   1. **Inisialisasi Matriks Transformasi**:    * Buat matriks identitas untuk memulai transformasi. 2. **Aplikasikan Transformasi**:    * Gunakan fungsi-fungsi seperti Translation, Scaling, RotationClockwise, dll., untuk mengaplikasikan transformasi pada matriks. 3. **Transformasikan Titik-Titik**:    * Gunakan fungsi GetTransformPoint untuk mengaplikasikan matriks transformasi pada setiap titik yang membentuk bentuk geometri. 4. **Gambar Bentuk yang Telah Ditransformasi**:    * Setelah titik-titik ditransformasi, gunakan fungsi PutPixelAll dari GraphicsUtils untuk menggambar bentuk yang telah ditransformasi.   **3. Homogenous Matriks / Coordinates**  **Homogenous Coordinates** adalah sistem koordinat yang digunakan dalam grafik komputer untuk mempermudah operasi transformasi seperti translasi, rotasi, dan scaling. Dalam sistem ini, setiap titik dalam ruang 2D direpresentasikan sebagai vektor 3D (x, y, 1). Dengan menggunakan homogenous coordinates, semua transformasi (translasi, rotasi, scaling) dapat direpresentasikan sebagai perkalian matriks, yang memungkinkan untuk menggabungkan beberapa transformasi menjadi satu matriks transformasi.  **4. Fungsi GetTransformPoint**  Fungsi GetTransformPoint digunakan untuk mengaplikasikan matriks transformasi pada setiap titik dalam daftar titik (List<Vector2>). Fungsi ini mengambil matriks transformasi dan daftar titik sebagai input, kemudian mengembalikan daftar titik yang telah ditransformasi.  Proses yang dilakukan oleh fungsi ini adalah:   * Mengalikan setiap titik dengan matriks transformasi. * Mengembalikan titik-titik yang telah ditransformasi.   **5. Efisiensi Transformasi Titik**  **Lebih efisien melakukan transformasi pada titik-titik pembangun bentuk**daripada melakukan transformasi pada semua titik dalam bentuk. Alasan utamanya adalah:   * **Jumlah Titik yang Lebih Sedikit**: Titik-titik pembangun bentuk (seperti titik sudut pada persegi atau segitiga) biasanya lebih sedikit daripada semua titik yang membentuk bentuk tersebut. Dengan mentransformasi titik-titik pembangun, Kita hanya perlu melakukan transformasi pada beberapa titik, bukan pada ratusan atau ribuan titik. * **Kemudahan dalam Menggambar**: Setelah titik-titik pembangun ditransformasi, kita dapat menggambar bentuk tersebut dengan menghubungkan titik-titik tersebut menggunakan algoritma garis atau bentuk lainnya. Ini lebih efisien daripada mentransformasi setiap piksel dalam bentuk. * **Kinerja yang Lebih Baik**: Dengan mengurangi jumlah titik yang perlu ditransformasi, Kita dapat meningkatkan kinerja aplikasi, terutama jika kita bekerja dengan bentuk yang kompleks atau dalam lingkungan real-time seperti game. |

Task 1: EksPLORASI TrANSFORMASIx

**Tugas Task 1 (Karya 1)**

1. Copy Karya 3 pada Modul 3 ke Karya 1 Pada Modul 4
2. Persentasikan Bentuk Bentuk Dasar Transformasi (Translasi, Scaling, Rotasi, Translasi + Scaling, Scaling+Translasi)

|  |
| --- |
| Lesson Learnt |
| A screenshot of a video game  Description automatically generated  **1. Memahami Transformasi dalam Grafika Komputer**  Dalam pengembangan grafika komputer, transformasi adalah proses memanipulasi objek dengan cara tertentu, seperti translasi, rotasi, dan scaling. Kode ini menerapkan berbagai bentuk transformasi pada objek 2D menggunakan Godot dan C#.  **2. Implementasi Transformasi Dasar**  **a. Translasi**  Translasi adalah pergeseran objek dalam ruang 2D tanpa mengubah bentuknya. Diterapkan dengan metode ApplyTranslation(x, y) yang memodifikasi matriks transformasi dengan menambahkan offset tertentu.  private void ApplyTranslation(float x, float y)  {  Vector2 coord = new Vector2(0, 0);  \_transformasi.Translation(\_transformMatrix, x, y, ref coord);  }  **b. Scaling**  Scaling adalah perubahan ukuran objek. Diterapkan dengan ApplyScaling(x, y), yang menggunakan titik pusat sebagai referensi.  private void ApplyScaling(float x, float y)  {  Vector2 coord = new Vector2(WorldOriginX, WorldOriginY);  \_transformasi.Scaling(\_transformMatrix, x, y, coord);  }  **c. Rotasi**  Rotasi mengubah orientasi objek searah atau berlawanan jarum jam berdasarkan sudut tertentu.  private void ApplyRotationClockwise(float angle)  {  Vector2 coord = new Vector2(WorldOriginX, WorldOriginY);  \_transformasi.RotationClockwise(\_transformMatrix, angle, coord);  }  **3. Penggunaan Transformasi Gabungan**  Dalam kode, beberapa kombinasi transformasi diterapkan:   * **Translasi + Scaling**: * ApplyScaling(2f, 2f);   ApplyTranslation(30, 30);   * **Scaling + Translasi**: * ApplyTranslation(30, 30);   ApplyScaling(2f, 2f);  Urutan operasi transformasi berpengaruh terhadap hasil akhir.  **4. Menggambar Objek dengan Transformasi**  Setelah transformasi diterapkan, objek digambar dengan metode PutPixelAll(), memastikan bahwa perubahan terlihat di layar.  List<Vector2> transformedShape = \_transformasi.GetTransformPoint(\_transformMatrix, shape);  PutPixelAll(transformedShape, colorShape);  **5. Penerapan dalam Kuadran Kartesian**   * **Kuadran I**: Translasi positif pada X dan negatif pada Y. * **Kuadran II**: Scaling lalu translasi. * **Kuadran III**: Translasi lalu scaling. * **Kuadran IV**: Rotasi lalu translasi.   **6. Kesimpulan**   * Transformasi adalah konsep fundamental dalam grafika komputer. * Urutan operasi transformasi memengaruhi hasil akhir. * Menggunakan matriks transformasi memungkinkan manipulasi bentuk secara efisien.   Dengan memahami dan mempraktikkan kode ini, kita dapat lebih mudah mengimplementasikan transformasi dalam pengembangan aplikasi grafis menggunakan Godot. |

Task 2: KARYA 2D Bunga transformasi

**Tugas Task 2 Karya 2**

1. Buatlah sebuah bunga yang terdiri dari lingkaran pusat dan 4 kelopak ellips (syarat buat satu kelopak terlebih dahulu lalu gunakan transformasi rotate untuk membuat kelopak lainnya)
2. Buatlah sebuah bunga yang terdiri dari lingkaran pusat dan 8 kelopak ellips (syarat buat satu kelopak terlebih dahulu lalu gunakan transformasi rotate untuk membuat kelopak lainnya)
3. Setiap bunga dibuat menjadi fungsi parametrik atau object (OOP) untuk memudahkan penduplikasian, duplikasikan bunga tersebut sebanyak 4 untuk bunga tipe 1, dan 6 untuk bunga tipe 2.

|  |
| --- |
| Lesson Learnt |
| A group of white circles with yellow center  Description automatically generated  **Lesson Learned - Membuat Bunga dengan Transformasi dan OOP di Godot (C#)**  **1. Membuat Bunga dengan 4 Kelopak Menggunakan Rotasi**  Dalam proses pembuatan bunga, kita menggunakan **ellips sebagai kelopak** dan sebuah **lingkaran pusat** untuk bagian tengah bunga. Untuk membuat kelopak:   * Kita cukup membuat **satu ellips** terlebih dahulu. * Kemudian, kita menerapkan **transformasi rotasi** untuk menggandakan kelopak tersebut di sekitar titik pusat bunga. * Rotasi dilakukan pada sudut **0°, 45°, 90°, dan 135°** untuk mendapatkan total **4 kelopak**.   **Implementasi**   * Menggunakan metode RotatePoint() untuk merotasi kelopak. * Menempatkan hasil rotasi ke dalam loop agar kelopak dapat muncul di posisi yang sesuai.   **2. Membuat Bunga dengan 8 Kelopak Menggunakan Rotasi**  Konsepnya sama dengan bunga berkelopak 4, tetapi dengan jumlah rotasi lebih banyak.   * Kita tetap membuat **satu ellips terlebih dahulu**. * Kemudian, kita menerapkan **rotasi ke sudut 0°, 22.5°, 45°, 67.5°, 90°, 112.5°, 135°, dan 157.5°**. * Hasilnya adalah **8 kelopak** yang lebih rapat dan menyerupai bunga dengan bentuk lebih kompleks.   **3. Menggunakan OOP untuk Memudahkan Duplikasi Bunga**  Agar lebih mudah dalam duplikasi bunga, kita menggunakan konsep **Object-Oriented Programming (OOP)** dengan membuat kelas Bunga.   * Kelas ini memiliki atribut:   + Pusat (koordinat pusat bunga)   + Ukuran (ukuran bunga)   + JumlahKelopak (jumlah kelopak, bisa diubah secara dinamis) * Kelas ini memiliki metode Gambar(), yang akan menggambar bunga dengan jumlah kelopak sesuai parameter yang diberikan.   **Keuntungan OOP**   * **Mudah diduplikasi**: Kita bisa membuat banyak objek bunga dengan mudah tanpa perlu menulis ulang kode. * **Fleksibel**: Kita bisa mengubah jumlah kelopak dan posisi dengan mudah. * **Lebih terstruktur**: Kode lebih rapi dan mudah dipahami.   **4. Mengubah Jumlah Kelopak dengan Input Keyboard**  Pada program ini, kita menambahkan fitur **input dari keyboard**:   * **Tekan tombol "1"** → Kembali ke jumlah kelopak awal (**8 untuk tipe 1, 4 untuk tipe 2**). * **Tekan tombol "2"** → Semua bunga berubah menjadi **8 kelopak**. * Menggunakan metode SetJumlahKelopak(int jumlah) untuk mengubah jumlah kelopak setiap bunga. * Memanggil QueueRedraw() agar gambar diperbarui setiap kali tombol ditekan.   **Kesimpulan**   1. **Rotasi** sangat berguna untuk menggandakan kelopak bunga tanpa perlu menggambar setiap kelopak secara manual. 2. **Menggunakan OOP** membuat kode lebih modular, mudah diperluas, dan lebih terstruktur. 3. **Interaksi dengan pengguna** (menggunakan input keyboard) memungkinkan modifikasi bunga secara dinamis tanpa harus menjalankan ulang program. 4. **Penerapan List** untuk menyimpan objek bunga memudahkan pengelolaan banyak bunga sekaligus.   Dengan memahami konsep ini, kita dapat dengan mudah mengembangkan sistem grafik berbasis transformasi dan OOP untuk aplikasi lainnya, seperti animasi, simulasi objek berulang, atau sistem procedural generation di Godot. |

Task 3: KARYA 2D Bunga transformasi ANIMASI

Tugas Task 3 Karya 3 (Animasinya dapat dilanjutkan minggu depan)

1. Semenarik mungkin tuangkan list ide-ide animasi untuk bunga tipe 1 dan tipe 2
2. Urutkan ide tersebut dari yang paling sederhana hingga yang paling kompleks. lalu pilih 2 ide animasi sederhana dan animasi kompleks.
3. Buatlah Design atau storyboard untuk animasi sederhana setiap bunga tipe 1 dan tipe 2 untuk (misal, untuk frame 1-60, bunga translasi +x dengan jarak 10 lalu berotasi 360 derajat lalu kembali ke tempat semula, dst).
4. Buatlah Design atau storyboard untuk animasi kompleks setiap bunga tipe 1 dan tipe 2 untuk (misal, untuk frame 1-60, bunga translasi +x dengan jarak 10 lalu berotasi 360 derajat lalu kembali ke tempat semula, dst).
5. Pada karya 3 implementasikan ide tersebut (untuk minggu depan).

|  |
| --- |
| Lesson Learnt |
| Disini saya menggunakan beberapa bunga yang saya modifikasi, pada tugas sebelumnya saya hanya menggunakan lingkaran dan elips, disini saya menggunakan variasi dengan menjadikannya sebagai titik titik dan menganimasikannya secara random, saya menganimasikannya dengan scaling, translasi,   * 1. Struktur Animasi dan Pembuatan Bunga **Posisi Awal**: Setiap bunga memiliki posisi acak dalam FieldSize. * **Ukuran**: Ukuran bunga bervariasi antara MinFlowerSize dan MaxFlowerSize. * **Jumlah Kelopak**: Setiap bunga memiliki 4 atau 8 kelopak secara acak. * **Parameter Animasi**:   + **Sway (Goyangan)**: Menggunakan swaySpeed dan swayAmount untuk efek angin.   + **Grow Pulse (Pulsasi Ukuran)**: Bunga sedikit membesar dan mengecil dengan growPulseSpeed dan growPulseAmount.   + **Rotasi**: Bunga berputar searah atau berlawanan dengan rotationSpeed.   Kode ini secara keseluruhan sudah cukup terstruktur untuk mengelola animasi banyak bunga dengan variasi unik.  **2. Teknik Animasi yang Digunakan**  Animasi dalam kode ini dilakukan dengan cara:   1. **Perubahan Posisi (Swaying Effect)**    * Menggunakan fungsi sinus untuk memberikan efek goyangan akibat angin:   float swayX = Mathf.Sin(time \* \_swaySpeed) \* \_swayAmount;  float swayY = Mathf.Cos(time \* \_swaySpeed \* 0.7f) \* (\_swayAmount \* 0.5f);   * + Kombinasi Sin dan Cos menciptakan pergerakan yang lebih alami.  1. **Pulsasi Ukuran (Growth Pulse)**    * Menggunakan fungsi sinus untuk membuat bunga tampak membesar dan mengecil:   \_currentScale = 1f + Mathf.Sin(time \* \_growPulseSpeed) \* \_growPulseAmount;   * + Efek ini menciptakan ilusi pernapasan pada bunga.  1. **Rotasi**    * Rotasi bunga dihitung dengan:   \_currentRotation = (time \* \_rotationSpeed) % 360f;   * + Ini menyebabkan bunga terus berputar seiring waktu.  1. **Penerapan Transformasi**    * Semua efek ini digabungkan dalam satu transformasi:   Transform2D transform = Transform2D.Identity  .Scaled(new Vector2(\_currentScale, \_currentScale))  .Rotated(Mathf.DegToRad(\_currentRotation));  **3. Kendala dalam Membuat Animasi Smooth**  **Masalah utama** yang dihadapi adalah animasi yang belum terasa halus. Beberapa kemungkinan penyebabnya:   * **Delta Time Tidak Dipakai dengan Konsisten**   + Di \_Process(double delta), delta sudah diterapkan dengan benar untuk rotasi, tetapi bisa dicek ulang apakah Update juga mempertimbangkan delta dengan baik. * **Kuantisasi Waktu yang Terlalu Kasar**   + Jika frame rate tidak stabil, gerakan berbasis sin(time \* speed) bisa terasa patah-patah.   + Bisa dicoba menggunakan delta lebih eksplisit:   float swayX = Mathf.Sin((\_elapsedTime + delta) \* \_swaySpeed) \* \_swayAmount;   * **Transformasi Tidak Bertahap**   + Jika bunga langsung diperbarui dalam satu langkah besar tanpa interpolasi, hasilnya bisa terlihat tidak smooth.   + **Solusi:** Gunakan Tween dari Godot untuk transisi yang lebih halus.   **4. Optimasi Potensial**   1. **Gunakan Tween untuk Animasi yang Lebih Halus**    * Godot memiliki sistem Tween untuk animasi interpolasi.    * Daripada mengandalkan perhitungan manual sinus, bisa menggunakan:   var tween = create\_tween()  tween.tween\_property(self, "rotation", 360, 2).set\_trans(Tween.TRANS\_SINE).set\_ease(Tween.EASE\_IN\_OUT)   * + Ini bisa membantu menghindari efek animasi yang patah-patah.  1. **Gunakan Interpolasi Linear**    * Saat menggambar, interpolasi linear bisa digunakan agar transisi lebih halus:   Vector2 smoothOffset = \_currentOffset.Lerp(targetOffset, 0.1f);   * + Ini memastikan bunga tidak langsung berpindah drastis.  1. **Kurangi Frekuensi Pembaruan Animasi**    * Jika animasi terasa terlalu cepat dan tidak smooth, bisa diperlambat dengan mengurangi nilai speed di \_swaySpeed dan \_growPulseSpeed. |

# PENGUMPULAN

Ikuti Format yang diberikan di Google Classroom.