IPB University — Bogor Indonesia | Departemen Ilmu Komputer

Tutorial Pemrograman Grafis
KOM1304 Grafika Komputer dan Visualisasi

Penulis: **Tim Pengajar**

Daftar Isi

Bagian 3: Transformasi Objek Geometri

3.1. Transformasi

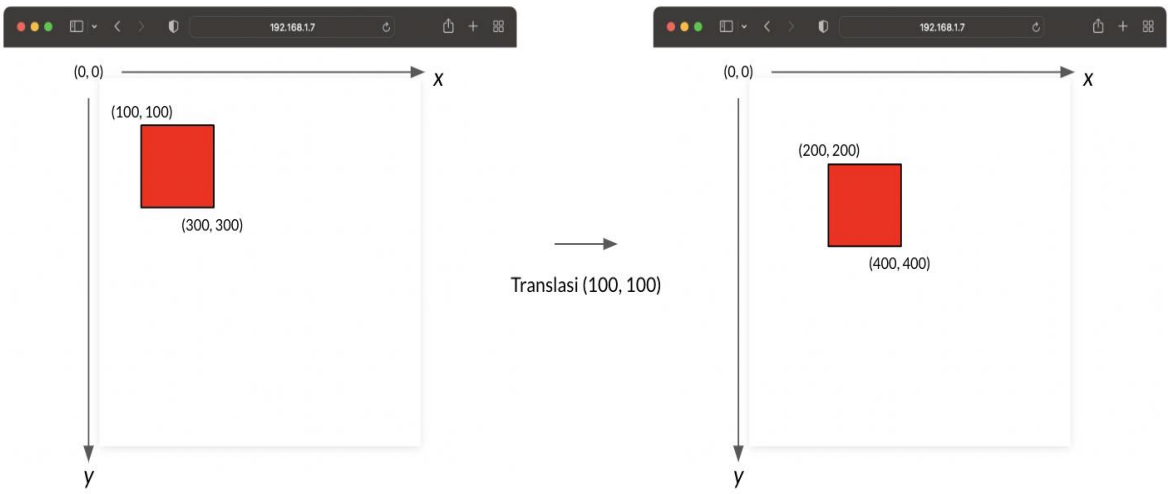
Transformasi merupakan suatu proses yang memodifikasi suatu objek grafis. Suatu objek grafis pada dasarnya terdiri atas sehimpunan verteks yang berada pada koordinat tertentu. Suatu transformasi akan memanipulasi nilai koordinat dari verteks tersebut sehingga objek grafis menjadi berubah. Ada tiga transformasi dasar pada grafika komputer, yaitu **translasi**, **rotasi**, dan **skalasi**.

Transformasi, yang berasal dari konsep matematika **geometri**, pada grafika komputer juga diterapkan dengan menggunakan **operasi matematika menggunakan matriks**. Ada dua matriks yang terlibat, yaitu **matriks koordinat** dan **matriks transformasi**. Matriks koordinat berisi **koordinat verteks** yang akan **ditransformasikan**, sedangkan **matriks transformasi** berisi **informasi transformasi** yang akan dilakukan.

Pada bagian ini, kita akan melihat bagaimana tiga transformasi dasar, yaitu **translasi**, **rotasi**, dan **skalasi**, diterapkan pada suatu objek dua dimensi. Penjelasan akan diberikan dalam bentuk operasi matriks dan penerapannya dalam kode program menggunakan Canvas HTML5.

3.1.1. Translasi

Translasi adalah suatu operasi yang akan memindahkan koordinat objek tanpa mengubah ukuran, bentuk atau orientasi. Dengan demikian, translasi dapat diibaratkan sebagai operasi menggeser suatu objek ke posisi yang baru. Oleh karena itu, informasi yang dibutuhkan dalam suatu transformasi adalah seberapa jauh pergeseran terjadi. Nilai pergeseran ini digunakan untuk setiap sumbu. Jika objek yang ditranslasi adalah objek dua dimensi, translasi akan memerlukan nilai pergeseran untuk sumbu-x dan sumbu-y. Pada contoh di Gambar 3.1., terlihat suatu objek bujur sangkar yang digeser sebanyak 100 satuan di sumbu-x dan 100 satuan di sumbu-y.



Gambar 3.1. Translasi pada suatu objek bujur sangkar sebanyak 100 satuan di sumbu-x dan 100 satuan di sumbu-y. Perhatian perubahan yang terjadi pada nilai-nilai koordinat verteks objek.

Proses di atas dapat ditulis dalam bentuk operasi matriks. Secara umum, operasi matriks untuk translasi adalah berupa perkalian antara matriks koordinat asal (P) dengan matriks transformasi (T) yang menghasilkan matriks koordinat hasil (P'). Operasi ini dilakukan untuk setiap titik (x, y) pada objek untuk menggesernya sebanyak (t_x, t_y).

$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} (1 \times x) + (0 \times y) + (t_x \times 1) \\ (0 \times x) + (1 \times y) + (t_y \times 1) \\ (0 \times x) + (0 \times y) + (1 \times 1) \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ 1 \end{pmatrix}$$

Matriks Transformasi (T)

P: Matriks Koordinat Asal

P': Matriks Koordinat Hasil

Gambar 3.2. Operasi matriks untuk melakukan translasi dari verteks (x, y) sebanyak (t_x, t_y). Operasi ini dilakukan untuk semua verteks pada objek.

Translasi pada contoh di atas dapat dituliskan pada Kode 3.1.

Kode 3.1. Translasi

```
1.  const canvasSketch = require('canvas-sketch');
2.
3.  const settings = {
4.    dimensions: [ 1200, 1200 ]
5.  };
6.
7.  const sketch = () => {
8.    return ({ context, width, height }) => {
9.      context.fillStyle = 'white';
10.     context.fillRect(0, 0, width, height);
11.
12.     context.fillStyle = 'blue';
13.     context.translate(100, 100);
14.     context.fillRect(100, 100, 300, 300);
15.   };
16. };
17.
18. canvasSketch(sketch, settings);
```

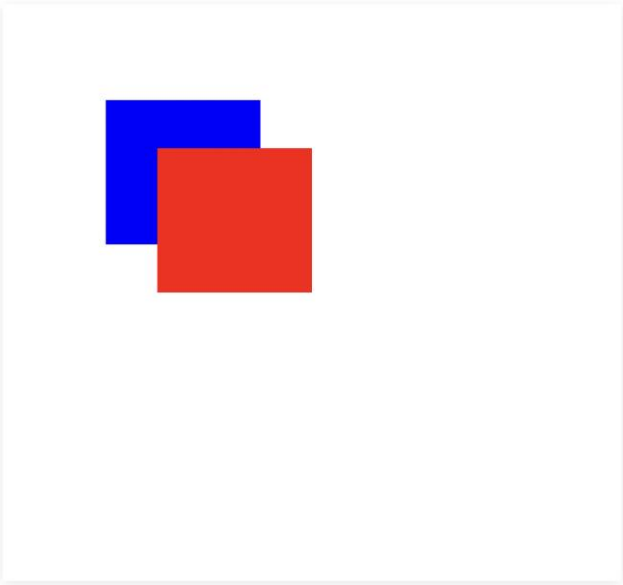
Perintah untuk melakukan translasi dapat dilihat pada baris ke-12 dengan dua argumen, yaitu pergerakan di sumbu-x dan sumbu-y. Nilai ini dapat bernilai positif dan negatif yang akan menentukan arah pergerakan.

Suatu operasi transformasi dapat diterapkan pada lebih dari satu objek secara sekaligus. Misalnya, kita akan membuat dua buah bujur sangkar, bewarna merah dan biru yang masing-masing digambar pada koordinat yang sama dan dioperasikan menggunakan transformasi yang sama.

Kode 3.2. Translasi pada dua bujur sangkar

```
1.  const canvasSketch = require('canvas-sketch');
2.
3.  const settings = {
4.    dimensions: [ 1200, 1200 ]
5.  };
6.  const sketch = () => {
7.    return ({ context, width, height }) => {
8.      context.fillStyle = 'white';
9.      context.fillRect(0, 0, width, height);
10.
11.     context.fillStyle = 'blue';
12.     context.translate(100, 100);
13.     context.fillRect(100, 100, 300, 300);
14.
15.     context.fillStyle = 'red';
16.     context.translate(100, 100);
17.     context.fillRect(100, 100, 300, 300);
18.   };
19. };
20.
21. canvasSketch(sketch, settings);
```

Dengan asumsi kedua objek memiliki koordinat dan translasi yang sama, kedua objek dapat diperkirakan akan digambar bertumpuk. Akan tetapi, pada saat program dijalankan, kedua bujur sangkar tidak bertumpuk. Bujur sangkar kedua yang bewarna merah digeser lebih jauh daripada bujur sangkar kedua (Gambar 3.3.).



Gambar 3.3. Hasil translasi dari Kode 3.2.

Hal ini disebabkan oleh cara kerja API Grafis yang memiliki matriks transformasi global yang akan mengakumulasi transformasi yang dilakukan. Pada Kode 3.2., translasi di baris ke-13 akan menggeser objek sejauh (100, 100). Ketika baris ke-14 dieksekusi, objek yang digambar akan

digeser sejauh (100, 100). Di baris ke-17, translasi kembali dilakukan sejauh (100, 100). Saat baris ini dieksekusi, jarak tersebut akan diakumulasikan dengan translasi sebelumnya sehingga total translasi yang dilakukan adalah sebanyak (200, 200). Oleh karena itulah bujur sangkar kedua digeser sejauh (200, 200). Akumulasi ini tidak hanya terjadi pada translasi, namun juga pada rotasi dan skalasi.

Oleh karena itu, dua buah fungsi diperkenalkan yaitu `context.save()` dan `context.restore()`. Kedua fungsi ini digunakan untuk menyimpan *state* dari suatu kanvas, atau mengembalikan *state* terakhir yang disimpan. State merupakan keadaan atau pengaturan yang dimiliki oleh kanvas, seperti fillStyle, strokeStyle, lineWidth, termasuk di dalamnya adalah isi dari matriks transformasi. Pada Kode 3.2., kedua fungsi di atas dapat digunakan seperti pada Kode 3.3. agar transformasi yang diaplikasi pada objek pertama tidak mempengaruhi transformasi pada objek kedua. Hasilnya, kedua bujur sangkar akan digambar pada koordinat yang sama.

Kode 3.3. Translasi pada dua bujur sangkar

```
1.  const canvasSketch = require('canvas-sketch');
2.
3.  const settings = {
4.    dimensions: [ 1200, 1200 ]
5.  };
6.
7.  const sketch = () => {
8.    return ({ context, width, height }) => {
9.      context.fillStyle = 'white';
10.     context.fillRect(0, 0, width, height);
11.     context.fillStyle = 'blue';
12.
13.     context.save();
14.     context.translate(100, 100);
15.     context.fillRect(100, 100, 300, 300);
16.     context.restore();
17.
18.     context.save();
19.     context.fillStyle = 'red';
20.     context.translate(100, 100);
21.     context.fillRect(100, 100, 300, 300);
22.     context.restore();
23.
24.   };
25. };
26.
27. canvasSketch(sketch, settings);
```

3.1.2. Skalasi

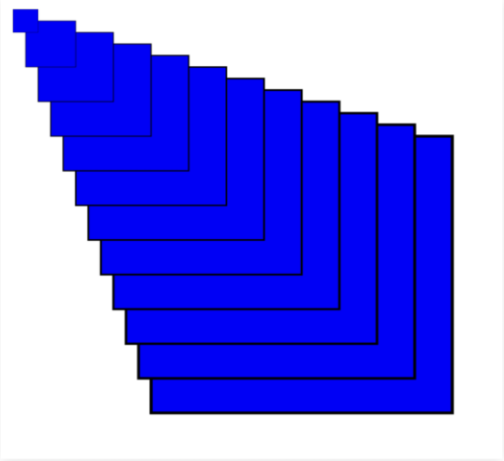
Transformasi berikutnya adalah skalasi yang akan mengalikan koordinat dengan suatu angka skalar (Gambar 3.6). Kode 3.5. memperlihatkan contoh kode yang menggunakan skalasi dengan hasil pada Gambar 3.7.

$$\begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} (S_x \times x) + (0 \times y) + (0 \times 1) \\ (0 \times x) + (S_y \times y) + (0 \times 1) \\ (0 \times x) + (0 \times y) + (1 \times 1) \end{pmatrix} = \begin{pmatrix} S_x \cdot x \\ S_y \cdot y \\ 1 \end{pmatrix}$$

Gambar 3.4. Operasi matriks untuk melakukan skalasi dari verteks (x, y) sebesar (S_x, S_y).

Kode 3.4. Skalasi

```
1.  const canvasSketch = require('canvas-sketch');
2.
3.  const settings = {
4.    dimensions: [ 600, 600 ]
5.  };
6.
7.  const sketch = () => {
8.    return ({ context, width, height }) => {
9.      context.fillStyle = 'white';
10.     context.fillRect(0, 0, width, height);
11.
12.     context.fillStyle = 'blue';
13.     context.strokeStyle = 'black';
14.
15.     for(var i = 12; i >= 1; i--){
16.       context.save();
17.       context.scale(0.3*i, 0.3*i);
18.       context.fillRect(50, 50, 100, 100);
19.       context.strokeRect(50, 50, 100, 100);
20.       context.restore();
21.     }
22.   };
23. };
24. canvasSketch(sketch, settings);
```



Gambar 3.5. Contoh visualisasi polygon rectangle dengan perbesaran 30% dari ukuran objek.

3.1.3. Rotasi

Transformasi dasar yang terakhir adalah rotasi. Mirip dengan translasi, rotasi menggunakan operasi matriks seperti pada Gambar 3.6. Rotasi akan memutar koordinat objek sebesar Θ derajat terhadap titik origin (0, 0) pada kanvas. Kode 3.5. memberikan contoh rotasi yang hasilnya dapat dilihat pada Gambar 3.7.

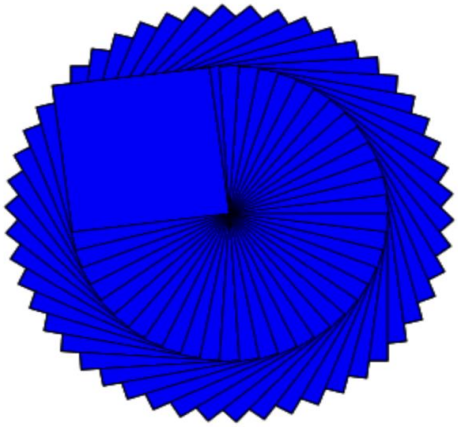
$$\begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} (cos\theta \times x) + (-sin\theta \times y) + (0 \times 1) \\ (sin\theta \times x) + (cos\theta \times y) + (0 \times 1) \\ (0 \times x) + (0 \times y) + (1 \times 1) \end{pmatrix} = \begin{pmatrix} x \times cos\theta - y \times sin\theta \\ x \times sin\theta + y \times cos\theta \\ 1 \end{pmatrix}$$

Gambar 3.6. Operasi matriks untuk melakukan rotasi dari verteks (x, y) sebesar Θ .

Kode 3.5. Rotasi

```
1.  const canvasSketch = require('canvas-sketch');
2.
3.  const settings = {
4.    dimensions: [ 600, 600 ]
5.  };
6.
7.  const sketch = () => {
8.    return ({ context, width, height }) => {
9.      context.fillStyle = 'white';
10.     context.fillRect(0, 0, width, height);
11.
12.     context.fillStyle = 'blue';
13.     context.strokeStyle = 'black';
14.
15.     const num = 50;
16.     var size = 100;
17.
18.     for(var i = 0; i < num*3; i++){
19.       context.save();
20.       context.translate(width*0.5, height*0.5);
21.       context.rotate((2*Math.PI / num) * i);
22.       context.fillRect(-size, -size, size, size);
23.       context.strokeRect(-size, -size, size, size);
24.       context.restore();
25.     }
26.   };
27. };
28. canvasSketch(sketch, settings);
```

Pelajari lebih lanjut tentang mengombinasikan translasi, skalasi, dan rotasi di:
https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Transformations



Gambar 3.7. Hasil kompilasi Kode 3.5.

3.1.4. Transformasi (*custom* pada Canvas HTML5)

Metode transformasi berikut memungkinkan modifikasi langsung ke matriks transformasi:

`transform(a, b, c, d, e, f)`
lakukan perkalian matriks CTM (Current Trasnformation Matriks) dengan matriks yang digambarkan di argument metode do atas. Matriks transformasi dideskripsikan oleh matriks (sesuai dengan parameter pada method transform).

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

Jika salah satu argumennya adalah *Infinity*, matriks transformasi harus ditandai sebagai infinite alih-alih metode yang memberikan pengecualian.

Keterangan dari parameter-parameter tersebut adalah:

- a (m11) merupakan bentuk *scaling* secara horizontal
- b(m21) meruapakan bentuk skewing secara horizontal
- c(m12) merupakan bentuk skewing secara vertical
- d(m22) merupakan bentuk scaling secara vertical
- e(m13) merupakan bentuk moving secara horizontal
- f(m23) merupakan bentuk moving secara vertical

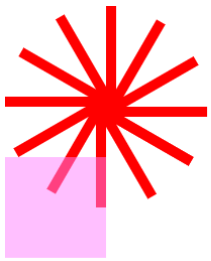
Method lainnya yaitu setTransform(a, b, c, d, e, f) untuk me-reset CTM ke bentuk matriks identitas dan kemudian memanggil metode transform() dengan argument yang sama. Artinya ada pembatalan transformasi yang tersimpan (*current*) kemudian menetapkan transformasi lain dalam 1 langkah. Sedangkan resetTransform(a, b, c, d, e, f) mengembalikan CTM ke matriks identitas atau sama dengan memanggil sintaks: ctx.setTransform(1, 0, 0, 1, 0, 0)

Kode 3.6. transformasi menggunakan matriks (`transform()` dan `setTransform()`)

```
1.  const canvasSketch = require('canvas-sketch');
2.
3.   const settings = {
4.     dimensions: [ 1200, 1200 ]};
```

```
5.
6.  const sketch = () => {
7.    return ({ context, width, height }) => {
8.      context.fillStyle = 'white';
9.      context.fillRect(0, 0, width, height);
10.
11.     const sin = Math.sin(Math.PI / 6);
12.     const cos = Math.cos(Math.PI / 6);
13.     context.translate(100, 100);
14.
15.     for (let i = 0; i <= 12; i++) {
16.       context.transform(cos, sin, -sin, cos, 0, 0);
17.       context.fillStyle = 'red';
18.       context.fillRect(0,0, 100, 10);
19.     }
20.     context.setTransform(-1, 0, 0, 1, 100, 100);
21.     context.fillStyle = "rgba(255, 128, 255, 0.5)";
22.     context.fillRect(0, 50, 100, 100);
23.   };
24.  };
25.
26. canvasSketch(sketch, settings);
```

Hasil dari kode 3.6 di atas dapat dilihat pada gambar 3.8 berikut:



Gambar 3.8. Hasil kompilasi Kode 3.6.

Bagaimana jika pada baris ke 21, method setTransform diganti menjadi transform() saja?

3.1.5. Properti tambahan dalam transformasi (*basic animation*)

Karena kita menggunakan JavaScript untuk mengontrol <canvas> elemen, juga sangat mudah untuk membuat animasi (interaktif). Berikutnya kita akan melihat contoh bagaimana melakukan animasi dasar yang dapat diterapkan di objek canvas.

Kendala atau batasan terbesar adalah, begitu sebuah object digambar, maka kondisinya adalah statis. Jika melakukan animasi maka perlu dilakukan pemindahan object (transformasi), atau menggambar ulang object sebelumnya diposisi yang berbeda dan menghapus (clear) semua yang digambar sebelumnya. Dibutuhkan banyak waktu untuk menggambar ulang frame yang kompleks dan kinerjanya sangat tergantung pada kecepatan komputer yang dijalankannya.

Berikut adalah Langkah-langkah dalam mengambar object yang ditrasnformasikan dalam bentuk animasi:

1. Kosongkan kanvas Kecuali bentuk yang akan kita gambar merupakan object yang tidak bergerak (misalnya gambar latar belakang). Kita perlu menghapus bentuk apa pun yang telah digambar sebelumnya. Cara termudah untuk melakukan ini adalah menggunakan metode clearRect().
2. Simpan kondisi canvas untuk setiap perubahan yang dilaukan (seperti style dan transormasi)
3. Gambar objek animasi
4. Restore canvas state, jika data objek sudah disimpan, lakukan restore sebelum menggambar objek/*frame* yang baru

Object/shapes digambar ke canvas dengan menggunakan metode canvas-sketch baik secara langsung atau dengan memanggil fungsi custom. Dalam keadaan normal, kita hanya melihat hasil ini muncul di kanvas ketika kode selesai dijalankan. Misalnya, tidak mungkin untuk melakukan animasi dari dalam for loop. Itu berarti kita membutuhkan cara untuk menjalankan fungsi menggambar kita selama periode waktu tertentu. Salah satu method untuk mengontrol object animasi dalam bentuk loop adalah: setInterval().

Berikut ini adalah kode sederhana untuk membuat animasi untuk menggambarkan rectangle yang dirotasi dengan suatu sudut tertentu

Kode 3.7. Transformasi menggunakan animasi setInterval().

```
1.  const canvasSketch = require('canvas-sketch');
2.
3.  const settings = {
4.    dimensions: [ 600, 600 ]};
5.
6.  const sketch = () => {
7.    return ({ context, width, height }) => {
8.      context.fillStyle = 'blue';
9.      context.fillRect(0, 0, width, height);
10.
11.     var ang = 0; //angle
12.     var fps = 1000/0.8; //jumlah frame per detik
13.
14.     setInterval(function(){
15.      context.save();
16.      context.clearRect(0, 0, width, height)
17.      context.beginPath();
18.      context.translate(300, 300);
19.      context.rotate(Math.PI / 180 * (ang +=6));
20.      context.fillRect(0, 0, 200, 10);
21.      context.restore();
22.
23.     }, fps);
24.
25.    };
26.  };
27.
28. canvasSketch(sketch, settings)
```

Gambar 3.9. Hasil kompilasi Kode 3.7

3.1. Latihan 1: Transformasi Obyek

Pada Tugas kelompok sebelumnya, telah diberikan instruksi untuk menggunakan objek-objek geometri yang telah dipelajari untuk menggambar inisial nama kalian (2-3 huruf) dengan menggunakan Canvas-Sketch (atau HTML Canvas).

Berikutnya, setelah memahami materi mengenai transformasi sederhana (Translasi, Rotasi, Skalasi), terapkan masing-masing inisial nama anggota kelompok untuk dapat:
a. Ditranslasikan sejauh nilai tertentu
b. Diskalasikan sejauh dan sebesar nilai tertentu
c. Dirotasikan sejauh sudut putar tertentu pada pusat obyek dan,
d. Dirotasikan sejauh sudut putar tertentu pada titik di sekitar obyek sebagai titik putar

Catatan: untuk setiap kelompok harus berbeda nilai translasi, rotasi dan skalasi

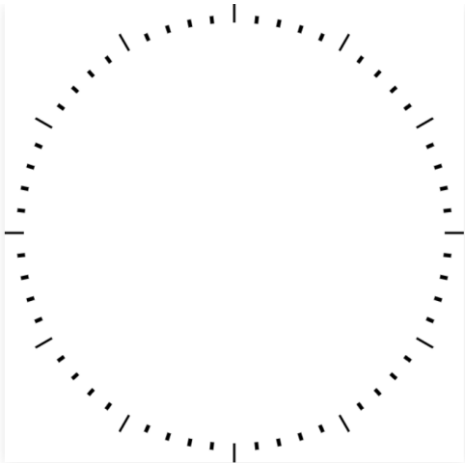
3.2. Latihan 2: Animasi sederhana Stopwatch

Petunjuk Pengerjaan Tugas
- Kerjakan soal berikut menggunakan *template* tugas yang tersedia di IPBM.
- Kumpulkan *screenshot* hasil kerja melalui google form yang diberikan

Diskusi dan tanya jawab seputar pengerjaan tugas atau Canvas API dapat dilakukan di *IPBM* dan tenggat waktu pengerjaan tugas dapat dilihat di IPBM.

Buatlah suatu grafika berupa objek stopwatch (menggunakan kode 3.8) yang dimodifikasi.

Langkah pertama adalah membuat objek garis yang kemudian di rotasi dengan pengulangan sehingga menghasilkan bentuk seperti gambar berikut ini:

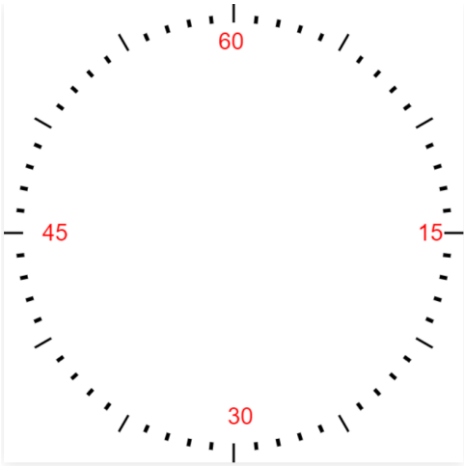


Gambar 3.10. Second marks untuk stopwatch.

Berikut contoh kode yang digunakan untuk menghasilkan frame (Sebagian) seperti pada gambar 10:

```
1.  // Second marks
2.    context.save();
3.    context.translate(300, 300);
4.    context.lineWidth = 5;
5.    for (let i = 0; i < 60; i++) {
6.      if (i % 5 !== 0) {
7.        context.beginPath();
8.        context.moveTo(275, 0);
9.        context.lineTo(285, 0);
10.       context.stroke();
11.     }
12.     context.rotate(Math.PI / 30);
13.   }
14.   context.restore();
```


Langkah berikutnya, tambahkan font untuk angka yang menunjukkan informasi posisi jarum stopwatch berada seperti pada gambar berikut:



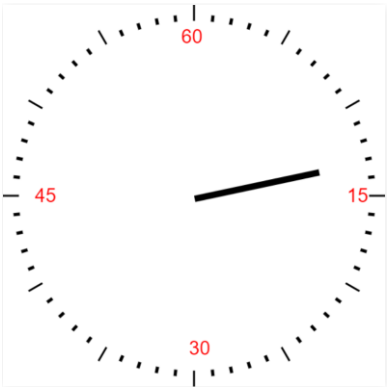
Gambar 3.11. Penambahan label angka

Seabgai contoh untuk menambahkan font ke dalam canvas adalah sebagai berikut:

```
1.  context.save();
```

```
2.  context.font = "30px Arial";
3.  context.fillStyle = "red";
4.  context.fillText("60", (width/2)-20, 60);
5.  context.restore();
```

Dan terakhir adalah menambahkan jarum stopwatch yang dimodifikasi dari Kode 3.8 sehingga hasilnya menjadi seperti gambar 3.12 berikut:



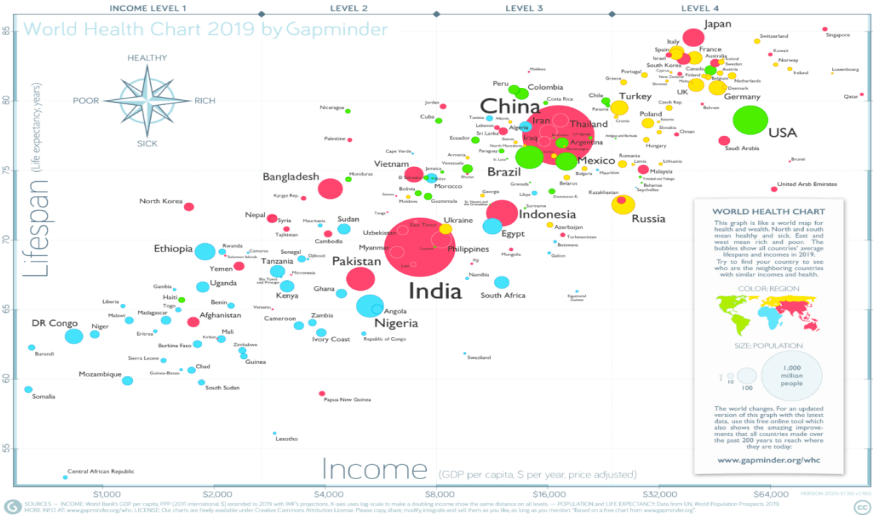
Gambar 3.12. Penambahan jarum stopwatch yang diberikan efek/propertis animasi.

3.3. Latihan 3: Visualisasi Bubble

Petunjuk Pengerjaan Tugas
- Kerjakan soal berikut menggunakan *template* tugas yang tersedia di LMS/IPBM.
- Kumpulkan *screenshot* hasil kerja melalui google form yang diberikan
- Diskusi dan tanya jawab seputar pengerjaan tugas atau Canvas API dapat dilakukan di *IPBM* dan tenggat waktu pengerjaan tugas dapat dilihat di IPBM.

Gapminder[1] merupakan suatu aplikasi visualisasi yang menampilkan data dalam plot dua dimensi. Setiap titik data direpresentasikan dalam bentuk lingkaran-lingkaran dengan ukuran yang berbeda-beda. Visualisasi ini disebut sebagai visualisasi *bubble* (Gambar 3.8.).

[1] https://www.gapminder.org/tools/#$chart-type=bubbles&url=v1



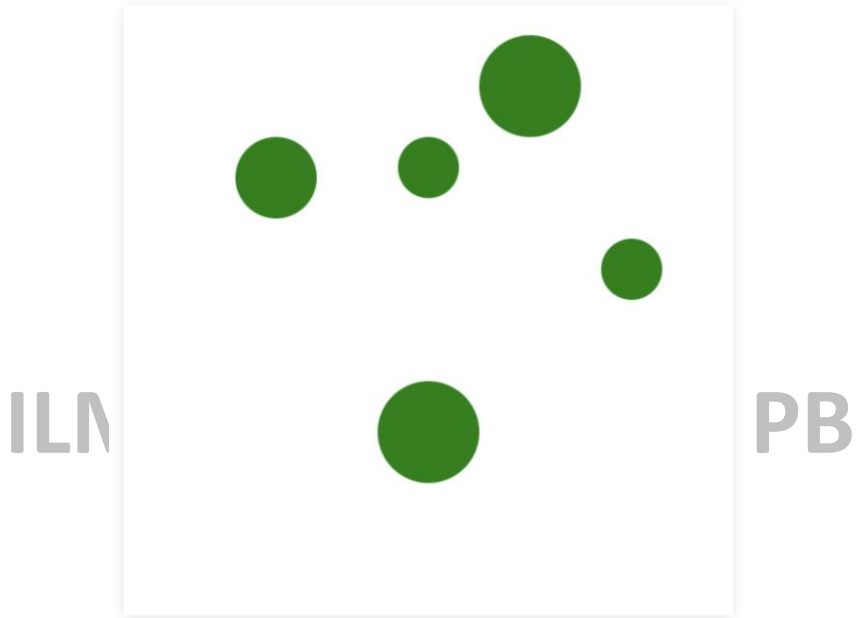
Gambar 3.8. Contoh visualisasi *bubble* pada Gapminder. .

Kode 3.6. telah memiliki fungsionalitas dasar untuk menampikan data dari sebuah matriks dua dimensi dalam bentuk plot lingkaran sederhana. Setiap baris pada matriks mewakili koordinat objek (sumbu-*x* dan sumbu-*y*) dan radius dari lingkaran yang dihasilkan. Hasil kompolasi kode 3.6. dapat dilihat pada Gambar 3.9.

Kode 3.6. Kode dasar visualisasi *bubble*

```
1.  const canvasSketch = require('canvas-sketch');
2.
3.  const settings = {
4.    dimensions: [ 600, 600 ]
5.  };
6.
7.  const sketch = () => {
8.    return ({ context, width, height }) => {
9.      context.fillStyle = 'white';
10.     context.fillRect(0, 0, width, height);
11.
12.     var data = [
13.       [400, 80, 50],
14.       [150, 170, 40],
15.       [500, 260, 30],
16.       [300, 160, 30],
17.       [300, 420, 50]
18.     ];
19.
20.     for(var i = 0; i < data.length; i++){
21.       context.save();
22.
23.       context.translate(data[i][0], data[i][1]);
24.       context.scale(data[i][2], data[i][2]);
25.
26.       var centerX = data[i][0];
27.       var centerY = data[i][1];
```

```
28.       var radius = data[i][2];
29.
30.       context.beginPath();
31.       context.arc(0, 0, 1, 0, 2 * Math.PI, false);
32.       context.fillStyle = 'green';
33.       context.fill();
34.
35.       context.restore();
36.     }
37.   };
38. };
39.
40. canvasSketch(sketch, settings);
```



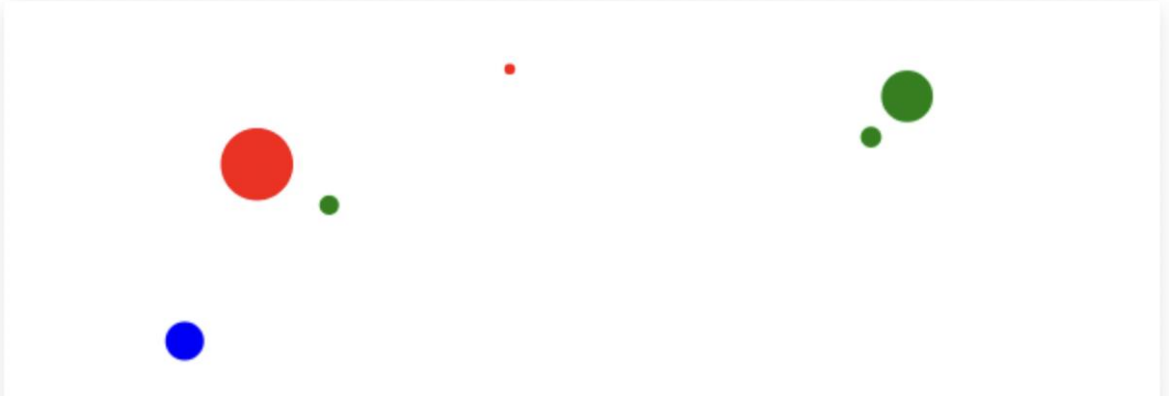
Gambar 3.9. Hasil kompilasi Kode 3.6.

Modifikasilah kode program tersebut agar visualisasi kode tersebut agar menjadi lebih mirip dengan visualisasi *bubble* pada Gapminder. Misalnya:

- Atur bentuk visualisasi agar koordinat (0, 0) berada pada pojok kiri bawah visualisasi.
- Tambahkan data dengan informasi warna yang akan ditampilkan.
- Buat sumbu *x* dan *y* pada visualisasi.
- Berikan teks-teks keterangan (referensi).
- Buat agar kode program membaca data dari sebuah *file*, tidak ditulis di dalam kode program (referensi).

```
const canvasSketch = require('canvas-sketch');

const settings = {
  dimensions: [ 600, 600 ]
};
```

```
const sketch = () => {
  return ({ context, width, height }) => {
    context.fillStyle = 'white';
    context.fillRect(0, 0, width, height);

    const maxLifespan = 85;
    const maxIncome = 64000;
    const maxPopulation = 1200000000;

    var data = [
      [14000, 73, 280000000, 1],  // Asia
      [50000, 78, 200000000, 2],  // Eropa
      [10000, 60, 150000000, 3],  // Afrika
      [18000, 70, 75000000, 2],   // Eropa
      [28000, 80, 40000000, 1],   // Asia
      [48000, 75, 80000000, 2],   // Eropa
    ];

    for(var i = 0; i < data.length; i++){
      var centerX = (data[i][0]/maxIncome)*width;
      var centerY = (data[i][1]/maxLifespan)*height;
      var radius = (data[i][2]/maxPopulation)*80;

      switch(data[i][3]){
        case 1:
          context.fillStyle = 'red';
          break;
        case 2:
          context.fillStyle = 'green';
          break;
        case 3:
          context.fillStyle = 'blue';
          break;
        default:
          context.fillStyle = 'green';
      }

      context.beginPath();
      context.arc(centerX, centerY, radius, 0, 2 * Math.PI, false);
      context.fill();
    }
  };
};

canvasSketch(sketch, settings);
```



Year	Country Name	Population	GPD Per Kapita	Lifespan	Continent
1999	Indonesia	280000000	14000	73	Asia
2000	Indonesia	280000000	14000	73	Asia
2001	Indonesia	280000000	14000	73	Asia
2002	Indonesia	280000000	14000	73	Asia