LAPORAN PENGANTAR GRAFIKA KOMPUTER

*“OPENGL DAN WEBGL”*



**Disusun Oleh :**

Nadhif Yonda Raditya

421051029

3 D4 Game Tech

**PROGRAM STUDI D4 TEKNOLOGI GAME**

**DEPARTEMEN TEKNOLOGI MULTIMEDIA KREATIF**

**POLITEKNIK ELEKTRONIKA NEGERI SURABAYA**

**2017**

**APA ITU OPENGL DAN WEBGL?**

Secara singkat, OpenGL merupakan suatu bahasa yang dapat merender grafik 2D dan 3D dan bersifat multi-platform. OpenGL sendiri biasanya berinteraksi dengan Graphics Processing Unit (GPU) untuk mendapatkan kecepatan rendering dari hardwarenya.

Sementara WebGL merupakan API berbasis JavaScript yang terdapat pada browser web yang kompatibel tanpa perlu menginstall plugin yang lain.

**KOORDINAT PADA OPEN GL DAN WEBGL**

*OpenGL:*

Sebuah titik secara umum sering disebut dimensi ke 0, garis dimensi ke 1, bidang dimensi ke 2, kubus dimensi ke 3, dan seterusnya. Untuk kebutuhan grafik umumnya kita menggunakan dimensi ke 3. Kita mempunyai sistem koordinat untuk 2 dimensi dan 3 dimensi yang dibedakan dengan jumlah sumbu koordinat yang dimiliki oleh masing-masing sistem. Sistem 2 Dimensi memiliki 2 sumbu koordinat, yaitu sumbu x dan sumbu y, sedangkan sistem koordinat 3 dimensi memiliki 3 buah sumbu koordinat, yaitu x, y dan z.

**Konversi Koordinat**

1. Konversi dari koordinat Nyata ke Sistem Konversi dengan memperhatikan dimensi dari sistem itu sendiri. Koordinat nyata mempunyai sistem 3 dimensi, sedangkan koordinat sistem ada 2 pilihan yaitu 2 dimensi dan 3 dimensi.

Real World Koordinat sistem (1)

Koordinat Sistem (2)

Koordinat Sistem (3)

2. Konversi dari koordinat sistem ke layer.

Masalah yang penting untuk diperhatikan di sini adalah perbedaan nilai koordinat antar koordinat sistem dengan koordinat layar, di mana koordinat sistem bisa memiliki nilai negatif, sedangkan koordinat layar hanya mengenal nilai positif.

Kalau pada koordinat sistem ada salah satu sumbu yang bernilai negatif maka data tersebut (pixel) tidak akan tampil pada layar. Dengan kata lain, pixel tampil di luar layar.

*WebGL:*

Dalam WebGL ke-3 titik koordinat ini dibuat sebagai *typed array* yang bernama *Float32Array.* WebGL memerlukan array dengan tipe data sama seperti koordinat vertex atau warna dan biasanya memproses data-data ini dalam jumlah yang besar untuk membentuk objek 3D. Dengan alasan performansi maka dalam JavaScript di kenalkan Float32Array yaitu array khusus dengan tipe data float 32bit. Operasi yang didukung array tipe ini lihat link mozilla tentang Float32Array.

**TRANSFORMASI**

***OPENGL***

1. *Translation (Perpindahan)*

Pengertian translasi adalah perpindahan objek dari titik P(x, y) ke titik P’(x’, y’) secara linier. Dalam translasi dua dimensi diperlukan dua parameter, yaitu pemindahan kearah sumbu x dan pemindahan kearah sumbu y. Translasi (dx, dy) didefinisikan dengan :

* x’= x + dx
* y’ = y + dy

*Bentuk umum :*

glTranslatef( x, y, z )

glTranslated( x, y, z )

glVertex3f(1.0f, -1.0f, -10.0f );

glColor3f( 1.0, 0.0, 0.0 );

glVertex3f(-1.0f, -1.0f, -10.0f );

glEnd();

1. *Scale (Skala)*

Definisi dari penskalaan m adalah perpindahan objek dari titik P ke titik P’, dimana jarak titik P’ adalah m kali titik P atau suatu operasi yang membuat suatu objek berubah ukurannya baik menjadi mengecil ataupun membesar secara seragam atau tidak seragam tergantung pada faktor penskalaan yang diberikan. Proses penskalaan dapat didefinisikan dengan :

* x’ = mx x
* y’ = my y

*Bentuk umum :*

glScalef( xf, yf, zf );

1. *Rotation (Rotasi)*

Definisi rotasi adalah perpindahan objek dari titik P(x, y) ke titik P’(x’,y’) yang berupa pemindahan berputar sebesar sudut θ atau suatu operasi yang menyebabkan objek bergerak berputar pada titik pusat atau pada sumbu putar yang dipilih berdasarkan sudut putaran tertentu. Operasi rotasi dapat diuraikan sebagai berikut :

Rotasi dari titik P yang berada di sumbu x, P(x, 0) menjadi P’(x’, y’) dapat digambarkan dengan :

* x’ = x .cos θ
* y’ = x.sin θ

Rotasi dari titik P yang berada di sumbu y, P(y, 0) menjadi P’(x’, y’) dapat digambarkan dengan :

* x’ = -y .cos θ
* y’ = y.sin θ

Rotasi dari titik P(x, y) menjadi P’(x’, y’) merupakan kombinasi dari kedua macam transformasi di atas R = Rx + Ry.

* x’ = x .cos θ – y.sin θ
* y’ = x.sin θ + y.cos θ

*Bentuk umum :*

glRotatef( sudut, sx, sy, sz );

***WEBGL***

1. *Translation (Perpindahan)*

Pada WebGL, dibuat variabel yang menampung translasi tersebut.

var translation = [0, 0];

var width = 100;

var height = 30;

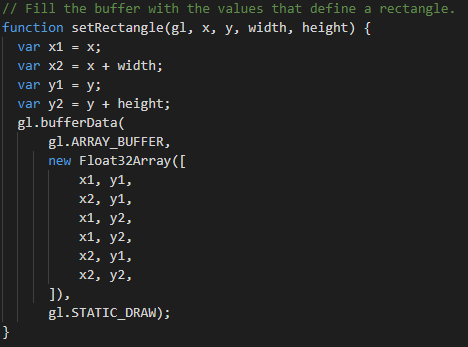
var color = [Math.random(), Math.random(), Math.random(), 1];

Variabel tersebut menampung koordinat dari objek. Selanjutnya kita hanya perlu menggambar ulang koordinat yang telah dirubah untuk mengupdate lokasi.

// Setup a rectangle

setRectangle(gl, translation[0], translation[1], width, height);

Berikut pseudo code untuk mengupdate posisi dari objek:



1. *Scale (Skala)*

Proses scaling pada WebGL hanyalah berupa perkalian dari posisi awal objek.

uniform vec2 u\_scale;

void main() {

// Scale the positon

vec2 scaledPosition = a\_position \* u\_scale;

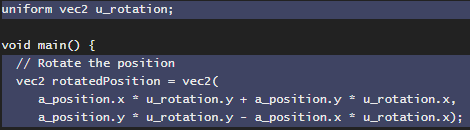
Dan kemudian tambahkan JavaScript untuk mengatur scale saat kita draw.

// Set the scale.

gl.uniform2fv(scaleLocation, scale);

1. *Rotation (Rotasi)*

Tambahkan pada shader, variabel untuk rotasi dan kemudian buat variabel rotasi tersebut.



Kemudian tambah JavaScript yang berfungsi untuk mengatur rotasi tersebut.



**MODEL, VIEW, & PROJECTION**

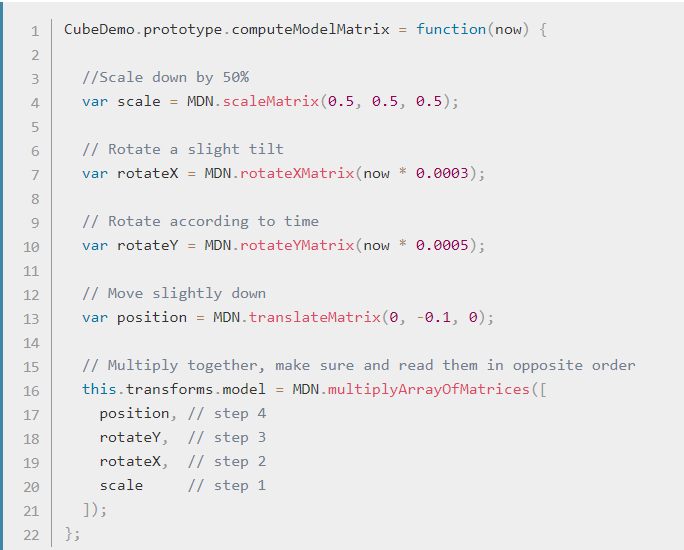
***WEBGL***

*Model*

Menempatkan titik langsung ke dalam ruang klip adalah penggunaan yang terbatas. Apa yang lebih baik adalah mengambil data model dan mengubahnya menjadi ruang klip. Kubus yang sederhana adalah contoh mudah bagaimana melakukan ini. Data kubus terdiri dari posisi vertex, warna wajah kubus, dan urutan posisi vertex yang membentuk poligon individu (dalam kelompok 3). Posisi dan warna disimpan dalam buffer, dikirim ke shader sebagai atribut, dan kemudian dioperasikan secara terpisah.

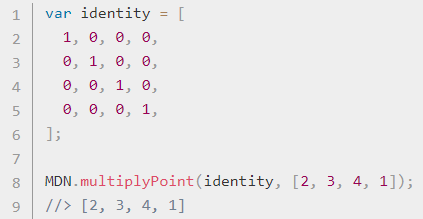
Akhirnya sebuah matriks model tunggal diatur yang mewakili transformasi yang akan dilakukan pada setiap posisi yang membentuk model untuk memindahkannya ke ruang yang benar. Dalam kasus ini, untuk setiap kerangka animasi serangkaian matriks skala, rotasi, dan terjemahan memindahkan data ke tempat yang diinginkan dalam ruang klip. Kubus adalah ukuran ruang klip (-1, -1, -1) sampai (1,1,1) sehingga perlu dikecilkan agar sesuai. Matriks ini dikirim langsung ke shader, telah dikalikan dengan JavaScript sebelumnya.

Contoh kode berikut mendefinisikan sebuah metode pada objek CubeDemo yang akan menciptakan matriks model. Menggunakan fungsi khusus untuk membuat dan mengalikan matriks seperti yang didefinisikan dalam kode bersama MDN WebGL.

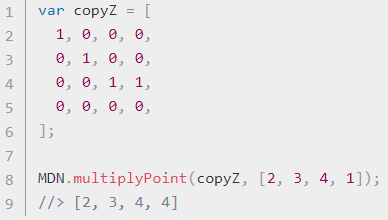


*Projection*

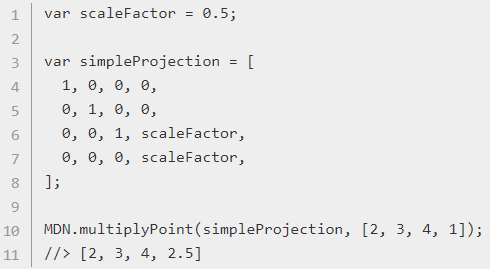
Langkah terakhir mengisi komponen w sebenarnya dapat dilakukan dengan matriks sederhana. Mulailah dengan matriks identitas:



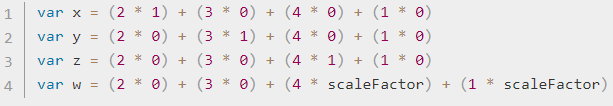
Lalu pindahkan kolom terakhir 1 sampai satu spasi.



Namun pada contoh terakhir kita melakukan (z + 1) \* scaleFactor:



Memecahnya sedikit lebih jauh kita bisa melihat bagaimana ini bekerja:

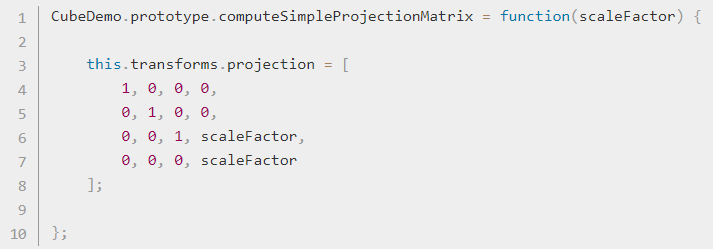


Baris terakhir dapat disederhanakan dan anjak keluar scalefactor, kita mendapatkan ini:



Yang persis sama dengan skalaf (z + 1) \* yang kita gunakan pada contoh sebelumnya.

Dalam kotak demo, metode tambahan. SimulatorSimpleProjectionMatrix () ditambahkan. Ini disebut dalam metode .draw () dan memiliki faktor skala yang melewatinya. Hasilnya harus identik dengan contoh terakhir:



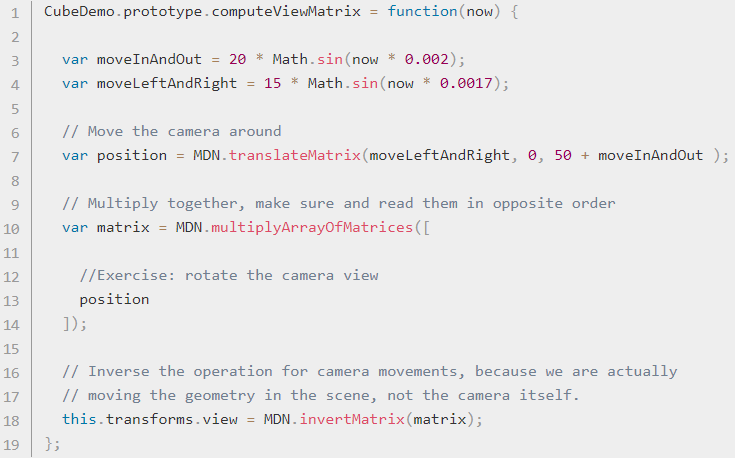
*View*

Langkah terakhir dalam semua ini adalah membuat matriks tampilan. Saat ini kita bisa memindahkan kubus di sekitar ruang dunia. Kita bisa memproyeksikan segalanya untuk memiliki perspektif, tapi kita tetap tidak bisa menggerakkan kamera.

Matriks terakhir adalah matriks tampilan yang mewakili lokasi kamera di luar angkasa. Bayangkan syuting film dengan kamera fisik. Matriks ini mewakili posisi dan rotasi kamera fisik itu.

Berbeda dengan matriks model, yang secara langsung mengubah simpul model, matriks tampilan menggerakkan kamera abstrak. Pada kenyataannya vertex shader masih hanya menggerakkan model dan "kamera" tetap pada tempatnya. Agar hal ini dapat bekerja dengan benar, kebalikan dari matriks transformasi harus digunakan. Matriks invers pada dasarnya membalikkan sebuah transformasi. Jadi jika kita memindah tampilan kamera ke depan, ia memindahkan semua benda di tempat kejadian kembali.

Metode ComputeViewMatrix () berikut menjiwai matriks tampilan dengan memindahkannya masuk dan keluar, dan kiri dan kanan.

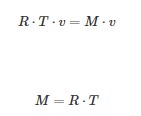


***OPENGL***

*Matriks model*

Seperti yang disarankan sebelumnya, kita bisa menerapkan berbagai transformasi geometris pada benda dengan menggunakan matriks. Jika kita perlu memutar objek, kita mengalikan koordinatnya dengan matriks rotasi, sama halnya untuk terjemahan dan penskalaan. Urutan di mana kita menerapkan transformasi ini ke objek sangat penting. Kami akan mencapai efek yang berbeda jika kita menerjemahkan dan menerapkan rotasi ke objek daripada jika kita memulai dengan memutar objek dan menerjemahkan hasilnya.

Dari sudut pandang matematis yang menerapkan dua, atau lebih, transformasi ke objek dapat dilakukan dengan mengalikan koordinat objek dengan transformasi matriks elementer satu per satu. Sebagai alternatif, kita bisa menggunakan satu matriks tunggal, yang berisi semua transformasi di atas, untuk mengalikan koordinat objek:



V adalah vektor kolom yang berisi simpul dari objek kita. Agar benar mengubah suatu objek, transformasi harus diterapkan ke setiap simpul objek.

Dalam persamaan di atas, kita telah mengganti produk dari dua matriks transformasi, R (rotasi) dan T (translasi), dengan matriks transformasi tunggal, M, dengan menggunakan properti asosiatif dari perkalian matriks.

Matriks M, yang berisi setiap terjemahan, rotasi atau penskalaan, yang diterapkan pada suatu objek diberi nama matriks model di OpenGL. Pada dasarnya, alih-alih menurunkan pipeline OpenGL dua, atau lebih, matriks transformasi geometris kita akan mengirimkan matriks tunggal untuk efisiensi. Ingat bahwa program vertex shader dijalankan untuk setiap simpul? Ini lebih efisien untuk melipatgandakan koordinat setiap titik dengan matriks model tunggal daripada melakukannya dengan dua atau lebih matriks.

*Matriks view*

Secara default, di OpenGL, penampil diposisikan pada sumbu z, seperti menggunakan kamera untuk mengambil gambar. Bayangkan bahwa kamera Anda menunjuk ke asal sistem Cartesian. Arah ke atas sejajar dengan sumbu Oy dan dalam arti positif Oy.

Matriks tampilan di OpenGL mengontrol cara kita melihat sebuah adegan. Pada artikel ini kita akan menggunakan matriks tampilan yang mensimulasikan sebuah kamera bergerak, biasanya bernama lookAt.

Hal ini di luar tujuan artikel ini untuk mendapatkan dan menyajikan cara kita membuat matriks tampilan, cukup untuk mengatakan bahwa itu adalah matriks 4x4, seperti matriks model, dan secara unik ditentukan oleh 3 parameter:

* Mata, atau posisi penampil;
* Bagian tengah, atau titik di mana kamera kita berada;
* Bagian atas, yang menentukan arah penampil.

Default di OpenGL adalah: mata pada (0, 0, -1); pusat di (0, 0, 0) dan atas diberikan oleh arah positif dari sumbu Oy (0, 1, 0).

Misalkan kita memiliki fungsi C ++ generik yang diberikan pada mata, pusat dan bagian atas akan mengembalikan matriks tampilan 4x4 untuk kita.

Matriks tampilan, V, mengalikan matriks model dan, pada dasarnya menyelaraskan dunia (objek dari sebuah adegan) ke kamera. Untuk simpul generik, v, inilah cara kita menerapkan transformasi tampilan dan model:



*Matriks projection*

Secara default, di OpenGL, sebuah objek akan tampak memiliki ukuran yang sama di manapun kamera berada. Ini bertentangan dengan pengalaman sehari-hari kita, di mana objek yang dekat dengan kita (ke kamera) terlihat lebih besar daripada objek yang berada pada jarak yang lebih jauh. Bayangkan cara bola mendekati Anda, bola akan tampak lebih besar dan lebih besar karena mata Anda lebih dekat.

Masalah lain dengan default dari OpenGL adalah bahwa untuk benar-benar melihat (menggambar) sesuatu di layar, objek yang ingin kita gambar harus berada di dalam kubus [-1, +1] x [-1, +1] x [-1, +1]. Setiap bagian dari scene kita yang berada di luar unit cube akan terpotong.

Kita dapat memperbesar atau mengecilkan volume kliping melalui penggunaan matriks proyeksi, ketika volume kliping kita berbentuk kotak paralel atau kotak, kita mengatakan bahwa kita menggunakan proyeksi ortografi. Proyeksi ortografi tidak akan mengubah ukuran benda tidak peduli di mana kamera diposisikan. Ini adalah karakteristik yang diinginkan untuk program CAD atau untuk game 2D.