Richard Rivaldo
13519185 – K04

Bentuk 3D

Small Notes on WebGL:

- WebGL diturunkan dari OpenGL yang lebih dahulu masuk ke dalam dunia grafika komputer. Oleh karena itu, WebGL memiliki banyak kemiripan cara kerja dengan OpenGL.
- Keberadaan WebGL sangat didukung oleh pengembangan HTML5 yang saat itu juga memperkenalkan komponen baru, yaitu *canvas*.
- Keuntungan WebGL adalah WebGL memiliki manajemen memori yang otomatis (tidak seperti OpenGL), bekerja dengan cepat karena memiliki akses terhadap perangkat grafis, portabel dan memberikan publisitas karya yang lebih baik, tidak membutuhkan kompilasi, serta mudah diintegrasikan ke dalam *website* karena dapat dikembangkan dengan menggunakan JavaScript. Alhasil, banyak sekali pustaka dan kakas yang dimiliki oleh WebGL.
- WebGL juga menggunakan GLSL (*Graphics Library Shading Language*) yang menggunakan versi standar OpenGL ES 2.0.
- Biasanya, pada berbagai macam media grafika komputer tersedia tiga primitif atau bentuk dasar, yaitu titik, garis, dan segitiga. Hal ini dikarenakan ketiga bentuk ini lebih mudah dirasterisasi oleh GPU dibandingkan bentuk lainnya.
- Dalam menggambarkan objek, WebGL menggunakan koordinat dengan prinsip *right-hand*. Selain itu, pada WebGL juga terdefinisi beberapa tipe data vector dan matriks seperti *vec2*, *vec3*, *vec4*, dan *mat4*.

1. Apa saja yang harus dilakukan saat ingin merender suatu bentuk kubus?

   Pada kenyataannya, WebGL hanyalah sebuah *state-machine* rasterisasi biasa yang menggambarkan titik, garis, dan segitiga berdasarkan kode program yang kita berikan. WebGL berjalan di atas GPU dan oleh karena itu, maka diperlukan penghubung *low-level* antara kode program yang dibuat dengan menggunakan JavaScript dengan GPU. Hal ini merupakan peran dari GLSL yang pada dasarnya merupakan bentuk C/C++ yang sangat ketat.
   Penghubung tersebut dibuat dalam bentuk *shaders*. *Shaders* secara sederhana dapat dikatakan sebagai program yang memungkinkan komputer untuk memberikan visualisasi tertentu. *Shaders* terbagi menjadi dua, yaitu *Vertex Shader* dan *Fragment Shader*. *Vertex Shader* adalah *shader* yang mengatur posisi dari *vertex* dan bagaimana *vertex* tersebut diciptakan. Adapun *Fragment Shader* mengatur warna dari setiap *pixels* pada permukaan primitif yang akan digambar.
   Shaders bisa menerima data dalam 4 buah bentuk. *Buffer* adalah alokasi memori kontigu yang berisi banyak informasi visualisasi yang ingin dibangun. Untuk menggunakan informasi tersebut, maka digunakanlah *attribute* yang biasanya berhubungan dengan *Vertex Shader*. Ada juga *uniform* yang merupakan masukan identik untuk kedua *shader* tersebut. Selain itu, ada juga mekanisme *message passing* antara kedua *shader* dengan menggunakan *varying* yang nilainya berubah-ubah tergantung objek yang sedang dirasterisasi. Terakhir, *shader* juga dapat menerima data dalam bentuk *texture* dari permukaan primitif yang divisualisasikan.

Keseluruhan spesifikasi di atas tergabung ke dalam sebuah *program*, dan untuk melakukan visualisasi apapun perlu didefinisikan terlebih dahulu program ini. Oleh karena itu, setelah melakukan *setup* struktur komponen HTML yang telah mengandung *canvas*, maka akan dilakukan pendefinisian program untuk membuat sebuah kubus. Dalam hal ini, pendefinisian program tersebut dilakukan terhadap sebuah WebGL *Rendering Context* tertentu. Maka, harus didapatkan terlebih dahulu konteks pada kanvas tersebut, baru setelahnya mendefinisikan kebutuhan-kebutuhan di atas.

Setelah objek *shader* dibuat dan didefinisikan spesifikasinya dalam GLSL, kemudian untuk setiap *shader* akan dipasangkan spesifikasi tersebut ke objeknya masing-masing. Setelah itu, barulah *shader* tersebut dikompilasi ke dalam WebGL. Ketika program sudah dibuat, maka *shader* yang telah lolos kompilasi akan dipasang ke program tersebut, dan program tersebut dikaitkan dengan WebGL *rendering context* kanvas yang ingin digambarkan.

Kemudian, perlu didefinisikan data *vertices* yang ingin digambarkan. Jika program hanya memuat bangun segitiga saja, maka *vertices* bisa langsung didefnisikan sebagai kumpulan titik-titik di mana segitiga tersebut ingin digambarkan. Untuk kasus visualisasi kubus, maka persegi pembangunnya bisa dibentuk dari penggabungan dua segitiga simetris.

Oleh karena itu, juga dibutuhkan pendefinisian tipe data *indices* yang menggambarkan titik-titik mana yang akan dihubungkan dan dibentuk menjadi segitiga. Dalam visualisasi kubus, maka untuk setiap permukaannya hanya perlu didefinisikan 4 buah titik. Pendefinisian setiap titik bisa dilengkapi atribut warna ataupun *texture mapping coordinate* yang akan memberikan tambahan properti pada *pixel* yang diraster.

Kemudian, data tersebut di-*bind* ke dalam *buffer* yang masing-masing dibuat terpisah, satu untuk data *vertices*, dan satu untuk data *indices*. Untuk memberikan data ini ke dalam *shader*, maka perlu didapat terlebih dahulu posisi atribut yang ingin dituju pada *shader*, Kemudian, setiap *vertex* nantinya bisa menunjuk atribut tersebut di dalam memori dengan spesifikasi tertentu, sesuai dengan implementasi yang dilakukan.

Selain itu, juga diperlukan untuk memberitahu konteks awal bahwa kita menggunakan program yang spesifikasinya telah kita tentukan di atas. Terakhir, kita hanya perlu mendefinisikan *render loop* yang digunakan WebGL dalam memberikan visualisasi. Jika sumber data yang menjadi target penggambaran hanyalah *vertices*, maka bisa digunakan *drawArrays*, dan untuk *indices* bisa digunakan *drawElements*.

Beberapa modifikasi perlu dilakukan terhadap program sehingga visualisasi yang dihasilkan tidak tampak aneh. Misalnya, dilakukan *enable* terhadap atribut *DEPTH_TEST* supaya objek tidak terlihat tembus pandang. Untuk objek yang ingin diberikan animasi rotasi, maka perlu didefinisikan beberapa buah matriks yang mengandung informasi data *vertex* secara dinamis. Matriks ini harus dikirimkan ke dalam *shader* melalui atribut dan operasi rotasi kemudian didefinisikan di dalam *rendering loop* awal.

Terakhir, pendefinisian tekstur secara penuh berarti menggantikan atribut warna biasa dari sebuah *vertex* pada *Fragment Shader*. Pendefinisian tekstur mirip dengan pendefinisian *buffer* awal. Akan ditentukan beberapa parameter dan properti dari tekstur tersebut, dan kemudian *texture* ini di-*bind* ke dalam daftar tekstur yang ada untuk digunakan nantinya dalam proses *rendering*.

Richard Rivaldo
13519185 – K04

2. Jika ingin merubah bentuk kubus menjadi bentuk limas segi empat, apa saja yang harus dilakukan?

   Pendefinisian sebuah limas akan melibatkan jumlah segitiga yang lebih sedikit dibandingkan dengan visualisasi yang dibutuhkan sebuah kubus. Oleh karena itu, sebenarnya hanya data *vertices* dan *indices* yang diberikan kepada *shader* saja yang utamanya perlu diubah untuk mengubah kubus menjadi limas segi empat. Jumlah segitiga yang digunakan pada limas segi empat hanyalah 6 buah, setengah jumlah dari visualisasi kubus. Bahkan, jika alas dari limas segi empat ini tidak perlu digambarkan, maka limas segi empat ini bisa dibentuk hanya dengan memberikan data *vertices* saja dan jumlah segitiga yang dibutuhkan untuk melakukan visualisasi tersebut hanyalah 4 buah.