Tugas Besar
IF3260 - Grafika Komputer

WebGL Part 1:
2D Primitive Elements

Dipersiapkan oleh
Asisten Grafika dan Inteligensi Buatan

Tanggal Mulai

Jumat, 15 Maret 2024, 12.00 WIB

Tanggal Akhir

Jumat, 29 Maret 2024, 23.59 WIB

Table of Contents

Deskripsi Tugas

Tugas besar ini akan menjadi pengenalan terhadap indahnya dunia grafika komputer dan melihatkan secara konkrit bagaimana permainan video tiga dimensi yang sering kamu mainkan itu di-*render* dalam layar monitor dua dimensi. Target platform yang akan digunakan adalah **WebGL**. Berikut adalah cakupan materi yang akan digunakan pada tubes ini:

- **WebGL 2D**
- **Line, Polygon**
- **Transformations (translation, rotation, dilation)**
- **Geometry**
- **Input and Output**

Spesifikasi Tugas

Tugas besar terdiri atas beberapa bagian, bagian pertama kali ini akan membuat kloningan photoshop.

1. Dasar WebGL

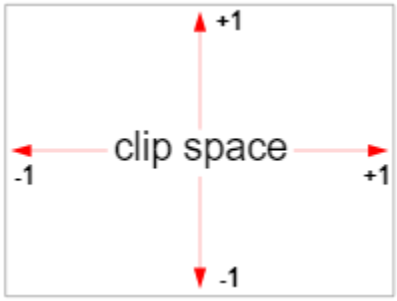
WebGL berjalan pada GPU di komputer, oleh karena itu, kamu perlu memberikan kode yang akan dijalankan di GPU tersebut. Kode tersebut diberikan dalam bentuk pasangan fungsi: *vertex shader* dan *fragment shader*. Keduanya ditulis dalam bahasa seperti C/C++ yang disebut sebagai GL Shader Language (GLSL). Dua fungsi tersebut dipasangkan menjadi *program*. Hampir semua API dari WebGL pada dasarnya mengatur *state* dari *program* tersebut. Setiap kali benda yang ingin kamu gambar, kamu perlu mengatur beberapa *state* lalu mengeksekusi program tersebut dengan memanggil `gl.drawArrays` atau `gl.drawElements` yang mengeksekusi *shader* kamu di dalam GPU.

Setiap data yang mau diakses oleh fungsi tersebut harus diberikan ke GPU terlebih dahulu. Ada 4 cara shader dalam menerima data:

1. **Attributes** dan **Buffers**: *buffers* merupakan *arrays* berisikan data binary yang di-*upload* ke GPU, sedangkan *attributes* memberitahukan bagaimana cara mengambil data tersebut dari *buffer*. Buffer **tidak dapat diakses secara random**, jadi *vertex shader* dijalankan beberapa kali dengan jumlah perulangan tertentu, dan setiap kali dieksekusi, nilai berikutnya dari *buffer* yang ditentukan akan diambil dan di-*assign* ke dalam atribut.
2. **Uniforms**: merupakan ***global variable*** yang kamu atur sebelum mengeksekusi program.
3. **Textures**: merupakan *arrays* berisikan data yang **dapat diakses secara random**.
4. **Varyings**: memberikan data dari *vertex shader* ke *fragment shader*. Tergantung dari apa yang akan di-*render*, nilai yang diatur *varying* oleh *vertex shader* akan di interpolasi ketika *fragment shader* dieksekusi.

WebGL hanya peduli dalam 2 hal: koordinat *clip space* dan warna. Sebagai programmer yang menggunakan WebGL, kamu perlu memberikan WebGL dua hal tersebut dengan memberikan 2 shaders untuk melakukan hal itu. *Vertex shader* berguna untuk memberikan

koordinat *clip space*, sedangkan *fragment shader* berguna untuk memberikan warna. Koordinat *clip space* selalu mulai dari -1 hingga +1, mau seberapa besarpun *canvas*-mu.



2. Contoh Sederhana WebGL

Secara sederhana, berikut contoh program WebGL, dimulai dari *vertex shader*:

```
// sebuah atribut akan menerima data dari buffer
attribute vec4 a_position;

// semua shader punya fungsi utama (main)
void main() {
 // gl_Position merupakan variabel spesial yang harus diatur oleh vertex shader
 // (anggap saja sebagai return result koordinat clip space)
 gl_Position = a_position;
}
```

Selanjutnya untuk *fragment shader*:

```
// fragment shader tidak punya default precision, jadi kita harus memilihnya.
// mediump merupakan default bagus yang berarti "medium precision".
// selain itu juga ada highp (high precision) dan lowp (low precision).
precision mediump float;

void main() {
 // gl_FragColor merupakan variabel spesial yang harus diatur
 // oleh fragment shader (anggap saja sebagai return warna)
 gl_FragColor = vec4(1, 0, 0.5, 1); // mengembalikan warna ungu kemerahan
}
```

Warna dalam WebGL berada pada rentang 0 hingga 1, sehingga kode diatas mengatur gl_FragColor menjadi 1 (r)ed, 0 (g)reen, 0.5 (b)lue, dan 1 (a)lpha, yang berarti warna ungu kemerahan tidak transparan.

Setelah kita memiliki sebuah kode sumber GLSL, mari kita mulai dengan membuat program WebGL.

1. Buat elemen canvas HTML:

```
<canvas id="c"></canvas>
```

2. Kita bisa mengambil canvas tersebut dengan javascript

```
let canvas = document.querySelector("#c");
```

3. Buat WebGLRenderingContext

```
var gl = canvas.getContext("webgl");
if (!gl) {
   // webgl tidak tersedia
   ...
```

4. Untuk kompilasi dua shaders tersebut ke dalam GPU, kita memerlukan shader tersebut dalam bentuk string. GLSL dalam bentuk string bisa dibuat dalam berbagai macam cara: *concatenating*, AJAX untuk mengunduh shader, *multiline template strings*, atau tag HTML <script> dengan tipe notjs. Misalkan kita menggunakan script tag pada HTML sebagai berikut:

```
<script id="vertex-shader-2d" type="notjs">

 // sebuah atribut akan menerima data dari buffer
 attribute vec4 a_position;

 // semua shader punya fungsi utama (main)
 void main() {
 // gl_Position merupakan variabel spesial yang harus diatur oleh vertex shader
 // (anggap saja sebagai return result koordinat clip space)
 gl_Position = a_position;
}

</script>

<script id="fragment-shader-2d" type="notjs">
 // fragment shader tidak punya default precision, jadi kita harus memilihnya.
 // mediump merupakan default bagus yang berarti "medium precision".
 // selain itu juga ada highp (high precision) dan lowp (low precision).
 precision mediump float;

 void main() {
   // gl_FragColor merupakan variabel spesial yang harus diatur
   // oleh fragment shader (anggap saja sebagai return warna)
```

```
  gl_FragColor = vec4(1, 0, 0.5, 1); // mengembalikan warna ungu kemerahan
}

</script>
```

Fun fact: Kebanyakan 3D engine membuat shader GLSL secara otomatis ketika dibutuhkan menggunakan template, concat, dan sebagainya.

5. Setelah kita mendapatkan shader dalam bentuk string, kita perlu membuat shader, upload kode sumber GLSL dalam bentuk string ke shader, dan kompilasi shader tersebut. Buat fungsi untuk melakukan hal tersebut terlebih dahulu

```
function createShader(gl, type, source) {
  let shader = gl.createShader(type);
  gl.shaderSource(shader, source);
  gl.compileShader(shader);
  let success = gl.getShaderParameter(shader, gl.COMPILE_STATUS);
  if (success) {
    return shader;
  }
  console.log(gl.getShaderInfoLog(shader));
  gl.deleteShader(shader);
}
```

Kita bisa memanggil fungsi tersebut untuk membuat 2 *shader*:

```
let vertexShaderSource = document.querySelector("#vertex-shader-2d").text;
let fragmentShaderSource = document.querySelector("#fragment-shader-2d").text;

let vertexShader = createShader(gl, gl.VERTEX_SHADER, vertexShaderSource);
let fragmentShader = createShader(gl, gl.FRAGMENT_SHADER,
fragmentShaderSource);
```

6. Selanjutnya, kita perlu menggabungkan dua *shader* tersebut menjadi sebuah *program*. Buat fungsi untuk membuat program terlebih dahulu:

```
function createProgram(gl, vertexShader, fragmentShader) {
    let program = gl.createProgram();
    gl.attachShader(program, vertexShader);
    gl.attachShader(program, fragmentShader);
    gl.linkProgram(program);
    let success = gl.getProgramParameter(program, gl.LINK_STATUS);
```

```
    if (success) {
      return program;
    }

    console.log(gl.getProgramInfoLog(program));
    gl.deleteProgram(program);
}
```

Lalu panggil fungsi dengan dua shader yang telah dibuat sebelumnya:

```
let program = createProgram(gl, vertexShader, fragmentShader);
```

Selamat, kamu telah punya program WebGL. Selanjutnya, kita perlu memasukkan data ke dalam program yang telah dibuat. Kebanyakan API dari WebGL sebenarnya digunakan untuk mengatur *state* untuk memberikan data ke program WebGL kita. Pada kasus diatas, kita hanya memiliki satu input, yaitu `a_position` berupa atribut. Bagaimana cara kita memasukkan data ke atribut?

Pertama, kita perlu mengambil lokasi atribut tersebut di dalam program. Pengambilan lokasi harusnya cuma dilakukan sekali ketika inisialisasi, bukan di dalam *render loop*.

```
let positionAttributeLocation = gl.getAttribLocation(program,
"a_position");
```

Masih ingat kan kalau atribut mendapatkan datanya dari buffer? Oleh karena itu, kita perlu membuat buffer.

```
let positionBuffer = gl.createBuffer();
```

Sebelum kita memberikan data, kita perlu melakukan *bind resource* (dalam hal ini buffer) ke sebuah *bind point* terlebih dahulu. Anggap saja *bind points* sebagai *internal global variables* di dalam WebGL. Jadi, kita *bind resource* ke *bind point*, kemudian semua fungsi WebGL akan mengacu *resource* ke *bind point* tersebut. Oleh karena itu, mari kita *bind position buffer*.

```
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
```

Baru kita bisa memasukkan data ke dalam *buffer* tersebut dengan mengacunya melalui *bind point*.

```
// tiga titik 2d
let positions = [
```

```
    0, 0,
    0, 0.5,
    0.7, 0,
];
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(positions),
gl.STATIC_DRAW);
```

Pertama, kita punya `positions` yang merupakan JavaScript array, sedangkan WebGL membutuhkan *strongly typed* data, sehingga kita perlu membuat *array* berisi *32 bit* angka desimal *(floating point)* dengan `new Float32Array(positions)`. Selanjutnya, `gl.bufferData` dipanggil untuk menyalin data tersebut ke `positionBuffer` pada GPU yang telah di *bind* ke `gl.ARRAY_BUFFER` sebelumnya.

3. Tugas

Mahasiswa ditugaskan menggunakan WebGL murni untuk mengimplementasikan web dengan fitur menggambar, mengedit, dan memvisualisasi sejumlah model pada kanvas. Berikut daftar spesifikasi yang harus diperhatikan:

- Fungsi-fungsi WebGL yang tidak primitive harus dibuat sendiri. Dijelaskan pula secara singkat dan seperlunya dalam readme
- Model yang harus diimplementasikan, beserta metode spesialnya:
    - Garis: Ubah panjang
    - Persegi: Ubah panjang sisi
    - Persegi panjang: Ubah panjang atau ubah lebar
    - Polygon: Penambahan dan penghapusan titik sudut
- Untuk setiap model, harus dapat dilakukan:
    - Transformasi geometri minimal 2 dari: translasi, dilatasi, rotasi, shear
    - Menggerakkan salah satu titik sudut dengan slider atau drag and drop
    - Mengubah warna salah satu atau semua titik sudut
    - Save sebuah model yang telah dibuat, format dibebaskan kepada mahasiswa, asal dapat di load kembali dan editable pada web yang diimplementasikan. Sediakan setidaknya 2 (dua) model yang siap untuk di-load pada repo.
- Implementasikan minimal satu dari fitur lanjutan pada poin berikutnya. Pengerjaan lebih dari satu fitur lanjutan dianggap bonus nilai. Daftar fitur di bawah juga sebatas saran, mahasiswa dapat mengimplementasikan fitur selain ini dan asalkan

didokumentasikan dengan baik, dapat mengklaim sebagai fitur bonus untuk dinilai asisten:

- Contoh fitur lanjutan:
    - Implementasi algoritma untuk menggambar polygon sedemikian sehingga dengan urutan penambahan titik yang berubah pun, gambar akhir polygon tetap sama yang merupakan convex hull dari titik-titiknya.
    - Integrasi animation pada salah satu fitur yang ada
    - Fitur penguncian pada salah satu aspek, misalnya sudut suatu titik dalam polygon dapat di-lock sehingga saat di-drag atau dipindahkan, titik tersebut masih bersudut sama. Contoh lain penguncian keliling, luas, atau kesebangunan
    - Menghasilkan model baru hasil irisan atau union dari 2 model

Penilaian

1. Source code akan dinilai dari kesesuaian dengan spesifikasi, serta **kontribusi individu** lewat ***commit history***. Semakin sedikit *commit* dan *change* mengakibatkan nilai kontribusi individu menjadi kecil.

2. Demo akan dilakukan dengan asisten, teknis akan diumumkan kemudian, kemungkinan menggunakan virtual meeting (zoom/gmeet) atau rekam video dengan batasan waktu.

3. Keindahan website **TIDAK** dinilai.

4. Diperbolehkan menggunakan library untuk mengubah penampilan atau membangun website seperti React atau library CSS. Namun, penggunaan library eksternal apapun untuk WebGL **TIDAK** diperbolehkan.

5. Berikut adalah distribusi untuk nilai tugas besar IF3270 Grafika Komputer:
    a. Implementasi menggambar garis, persegi, persegi panjang, dan polygon: **30** poin
    b. Transformasi geometri: **20** poin
    c. Menggerakkan titik sudut: **10** poin
    d. Mengubah warna: **10** poin
    e. Save model: **10** poin
    f. Fitur lanjutan wajib: **20** poin
    g. Fitur lanjutan kedua dan seterusnya: bonus **(max 10 poin)**

    Note: Ketidaklengkapan laporan dapat mengurangi penilaian.

Pengumpulan dan Deliverables

1. Untuk tugas ini Anda diwajibkan menggunakan version control system **git** dengan menggunakan sebuah repository **private** di Github Classroom "**Grafkom**" (gunakan email stei.itb.ac.id agar gratis). Berikut adalah Link Github Classroom.

2. Setiap kelompok wajib membuat tim Github Classroom dengan **nama yang sama pada spreadsheet kelompok.**

3. Setiap tugas besar **wajib untuk membuat laporan tugas besar** berisi deskripsi, hasil, serta manual/contoh fungsionalitas program yang dibuat, **BUKAN** berisi source code.

4. Repository minimal berisi
    a. Folder **src**: html, css, dan js file maupun file tambahan yang diperlukan
    b. Folder **test**: Contoh model yang dapat di load
    c. Folder **doc**: Berisi laporan **dalam pdf**
    d. **README.md**: Deskripsi singkat cara menjalankan program

5. Satu kelompok beranggotakan 3 orang. Anggota bebas selama berasal dari kelas yang sama. Isikan nama kelompok pada Link Spreadsheet
   Deadline Pengisian Kelompok Minggu, 17 Maret 2024, 23.59 WIB

6. **Mulai** Jumat, 15 Maret 2024, 18.00 WIB
   Deadline Jumat, 29 Maret 2024, 23.59 WIB

7. Pengumpulan dilakukan dengan membuat **release** dengan major version **1** seperti **v1.x**. Versi patch dapat digunakan jika ada perbaikan. Contoh: Jika ada revisi pada versi **v1.0**, selama tidak melewati *deadline*, dipersilahkan melakukan revisi kode dan melakukan *release* lagi dengan versi **v1.1**. Penilaian akan dilakukan terhadap versi rilis terakhir.

8. Berikut adalah pelanggaran yang dapat menyebabkan **pengurangan nilai**
    - **Plagiarisme & Kerjasama Antar Kelompok**
    - **Perubahan kode setelah deadline**
    - **Tidak membuat release**

9. Jika ada pertanyaan atau masalah pengerjaan harap segera menggunakan sheets QnA mata kuliah IF3270 Grafika Komputer pada Link QnA. Tidak diperkenankan mengontak asisten secara langsung.

Referensi

1. WebGL Fundamentals
2. WebGL API
3. WebGL Examples

Changelog

Berikut adalah list perubahan pada dokumen tugas besar. Untuk perubahan signifikan juga akan dikirim melalui milis

0. Timestamp - Dokumen X - Bagian Y