TP: WebGL

R5.A.06: Sensibilisation à la programmation multimédia

Abdelkader Gouaïch

2023/2024

Etape 1: Dessiner un point

Notre objectif est de dessiner un point simple sur une toile.

1. Préparation de l'Environnement:

- Créez un nouveau fichier HTML.
- Ajoutez un élément `<canvas>` avec un identifiant `canvas` dans le corps du document.
- Ajoutez une balise `<style>` pour mettre en forme le corps de la page et le canvas.
- Insérez une balise `<script>` pour y écrire votre code JavaScript.

```
<!-- ... (code précédent) -->
<canvas id="canvas"></canvas>
<style>
  body {
    margin: 0;
  }

  canvas {
    display: block;
  }
</style>
<script>
  // Votre code JavaScript ira ici
</script>
<!-- ... (code suivant) -->
```

2. Configuration Initiale de WebGL:

- Récupérez le contexte WebGL du canvas.
- Vérifiez que votre navigateur supporte WebGL.

```
var canvas = document.getElementById("canvas");
var gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");

if (!gl) {
  alert("WebGL non supporté par votre navigateur");
}
```

3. Création et Compilation des Shaders:

- Écrivez le code source pour les shaders vertex et fragment.
- Compilez les shaders.

```
// ... (code précédent)

var vertexShaderSource = `
    attribute vec4 a_position;
    void main() {
        gl_Position = a_position;
        gl_PointSize = 10.0;
    }
`;

var fragmentShaderSource = `
    precision mediump float;
    void main() {
        gl_FragColor = vec4(0.8, 0.2, 0, 1);  // Orange
    }
`;

function compileShader(source, type) {
  var shader = gl.createShader(type);
  gl.shaderSource(shader, source);
  gl.compileShader(shader);
  if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
    console.error(
      "Erreur de compilation du shader: " + gl.getShaderInfoLog(shader)
    );
    gl.deleteShader(shader);
    return null;
  }
  return shader;
```

```
}

var vertexShader = compileShader(vertexShaderSource, gl.VERTEX_SHADER);
var fragmentShader = compileShader(fragmentShaderSource, gl.FRAGMENT_SHADER);
```

4. Lien des Shaders dans un Programme:

- Liez les shaders compilés dans un programme WebGL.
- Utilisez le programme.

```
// ... (code précédent)

var program = gl.createProgram();
gl.attachShader(program, vertexShader);
gl.attachShader(program, fragmentShader);
gl.linkProgram(program);
gl.useProgram(program);
```

5. Préparation des Données Géométriques:

- Définissez les données de vertex pour votre point.
- Créez un buffer et chargez-y vos données.

```
// ... (code précédent)

var positions = new Float32Array([
  0.0,
  0.0, // Centre du point
]);

var positionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
gl.bufferData(gl.ARRAY_BUFFER, positions, gl.STATIC_DRAW);
```

6. Configuration des Attributs:

- Obtenez la localisation de l'attribut `a_position` du programme.
- Activez l'attribut et configurez-le pour utiliser votre buffer de positions.

```
// ... (code précédent)

var position = gl.getAttribLocation(program, "a_position");
gl.enableVertexAttribArray(position);
gl.vertexAttribPointer(position, 2, gl.FLOAT, false, 0, 0);
```

7. Phase de Dessin:

- Utilisez le programme.
- Effacez le canvas.
- Dessinez le point.

```
// ... (code précédent)

gl.useProgram(program);
gl.clear(gl.COLOR_BUFFER_BIT);
gl.drawArrays(gl.POINTS, 0, 1);
```

Etape 2: dessiner une ligne

Notre objectif est de programmer toutes les étapes pour dessiner des lignes.

1. Préparation de l'environnement:

- Créez un nouveau fichier HTML.
- Ajoutez un élément `<canvas>` avec un identifiant `canvas`.
- Liez un fichier JavaScript externe ou incluez une balise `<script>` dans votre fichier HTML.

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8" />
    <title>TP WebGL</title>
  </head>
  <body>
    <canvas id="canvas" width="500" height="500"></canvas>
    <script src="script.js"></script>
  </body>
</html>
```

2. Configuration initiale de WebGL:

- Dans votre fichier JavaScript, récupérez le contexte WebGL du canvas.
- Vérifiez que votre navigateur supporte WebGL.

```
var canvas = document.getElementById("canvas");
var gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");

if (!gl) {
```

```
  alert("WebGL non supporté par votre navigateur");
}
```

3. Création et Compilation des Shaders:

- Écrivez le code source pour les shaders vertex et fragment.
- Compilez les shaders.

```
// Shader sources
var vertexShaderSource = `
    attribute vec4 a_position;
    void main() {
        gl_Position = a_position;
        gl_PointSize = 2.0;
    }
`;

var fragmentShaderSource = `
    precision mediump float;
    void main() {
        gl_FragColor = vec4(0.8, 0.2, 0, 1);  // Orange
    }
`;

// Compile shader
function compileShader(source, type) {
  var shader = gl.createShader(type);
  gl.shaderSource(shader, source);
  gl.compileShader(shader);
  if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
    console.error(
      "Erreur de compilation du shader: " + gl.getShaderInfoLog(shader)
    );
    gl.deleteShader(shader);
    return null;
  }
  return shader;
}

var vertexShader = compileShader(vertexShaderSource, gl.VERTEX_SHADER);
var fragmentShader = compileShader(fragmentShaderSource, gl.FRAGMENT_SHADER);
```

4. Lien des Shaders dans un Programme:

- Liez les shaders compilés dans un programme WebGL.
- Vérifiez que le lien est réussi.

```
// ... (code précédent)

// Link shaders into a program
var program = gl.createProgram();
gl.attachShader(program, vertexShader);
gl.attachShader(program, fragmentShader);
gl.linkProgram(program);

if (!gl.getProgramParameter(program, gl.LINK_STATUS)) {
  console.error(
    "Erreur de liaison du programme: " + gl.getProgramInfoLog(program)
  );
  gl.deleteProgram(program);
  return;
}
```

5. Préparation des Données Géométriques:

- Définissez les données de vertex pour vos lignes.
- Créez un buffer et chargez-y vos données.

```
// ... (code précédent)

// Data for a single point
var positions = new Float32Array([-1, -1, 1, 1, -1, 1, 1, -1]);

// Create a buffer
var positionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
gl.bufferData(gl.ARRAY_BUFFER, positions, gl.STATIC_DRAW);
```

6. Configuration des Attributs:

- Obtenez la localisation de l'attribut `a_position` du programme.
- Activez l'attribut et configurez-le pour utiliser votre buffer de positions.

```
// ... (code précédent)

var position = gl.getAttribLocation(program, "a_position");
```

```
gl.enableVertexAttribArray(position);
gl.vertexAttribPointer(position, 2, gl.FLOAT, false, 0, 0);
```

7. Phase de Dessin:

- Utilisez votre programme.
- Effacez le canvas.
- Dessinez votre forme géométrique.

```
// ... (code précédent)

<!-- Phade de dessin -->

gl.useProgram(program);
// Draw the point
gl.clear(gl.COLOR_BUFFER_BIT);
gl.drawArrays(gl.LINES, 0, 4);
```

Etape 3: dessiner un triangle

Créer un nouveau fichier nommé `drawTriangle.html`. Reprenez les instructions vues dans les étapes précédents pour dessiner cette fois-ci un triangle.

Pour dessiner triangle nous avons besoin de trois vertex. En deux 2D, chaque vertex aura des coordonnées composés de deux floats.

La primitive de dessin finale sera :

```
// Draw the point
gl.clear(gl.COLOR_BUFFER_BIT);
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Nous vous demandons d'écrire le programme complet en vous aidant des étapes précédentes.

Etape 4: dessiner un rectangle

Nous souhaitons maintenant dessiner un rectangle.

L'instruction :

```
gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
```

va dessiner 2 triangles qui partagent un segment. Le premier triangle est déterminé par les points 0, 1 e 2 et le deuxième triangles sera déterminé par les points

1, 2 et 3. Nous pouvons ainsi dessiner un rectangle en utilisant deux triangles qui nécessitent 4 points avec l'instruction `TRIANGLE_STRIP`.

Créer un nouveau fichier nommé `drawRectangle.html` et donner le code pour dessiner un rectangle en vous aidant des instructions précédentes.

Etape 5: Appliquer une texture

Notre objectif est de créer une forme géométrique simple, dans ce cas, un rectangle, et l'application d'une texture sur cette forme en utilisant WebGL.

Pour cette étape, placez une image nommée 'perso.jpg' dans le même dossier que votre fichier HTML.

1. Préparation de l'environnement:

- Créez un nouveau fichier HTML.
- Ajoutez un élément `<canvas>` avec un identifiant `canvas`.
- Incluez une balise `<style>` pour appliquer des styles de base au canvas et au corps de la page.
- Incluez une balise `<script>` pour écrire votre code JavaScript.

```
<!-- ... (code précédent) -->
<canvas id="canvas"></canvas>
<style>
  body {
    margin: 0;
  }

  canvas {
    display: block;
  }
</style>
<script>
  // Votre code JavaScript ira ici
</script>
```

2. Configuration initiale de WebGL:

- Récupérez le contexte WebGL du canvas.
- Vérifiez que votre navigateur supporte WebGL.

3. Création et Compilation des Shaders:

- Écrivez le code source pour les shaders vertex et fragment.
- Compilez les shaders.

```
// ... (code précédent)

var vertexShaderSource = `
    attribute vec4 a_position;
    attribute vec2 a_texCoord;
    varying vec2 v_texCoord;
    void main() {
        gl_Position = a_position;
        v_texCoord = a_texCoord;
    }
`;

var fragmentShaderSource = `
    precision mediump float;
    varying vec2 v_texCoord;
    uniform sampler2D u_texture;
    void main() {
        gl_FragColor = texture2D(u_texture, v_texCoord);
    }
`;

function compileShader(source, type) {
  var shader = gl.createShader(type);
  gl.shaderSource(shader, source);
  gl.compileShader(shader);
  if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
    console.error(
      "Erreur de compilation du shader: " + gl.getShaderInfoLog(shader)
    );
    gl.deleteShader(shader);
    return null;
  }
  return shader;
}

var vertexShader = compileShader(vertexShaderSource, gl.VERTEX_SHADER);
var fragmentShader = compileShader(fragmentShaderSource, gl.FRAGMENT_SHADER);
```

4. Lien des Shaders dans un Programme:

- Liez les shaders compilés dans un programme WebGL.
- Utilisez le programme.

```
// ... (code précédent)

var program = gl.createProgram();
gl.attachShader(program, vertexShader);
gl.attachShader(program, fragmentShader);
gl.linkProgram(program);
gl.useProgram(program);
```

5. Préparation des Données Géométriques:

- Définissez les données de vertex pour votre rectangle.
- Créez un buffer et chargez-y vos données.

```
// ... (code précédent)

var positions = new Float32Array([
  -0.5, -0.5, 0.0, 1.0, 0.5, -0.5, 1.0, 1.0, 0.5, 0.5, 1.0, 0.0, -0.5, 0.5, 0.0,
  0.0,
]);

var positionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
gl.bufferData(gl.ARRAY_BUFFER, positions, gl.STATIC_DRAW);
```

6. Configuration des Attributs:

- Obtenez la localisation de l'attribut `a_position` et `a_texCoord` du programme.
- Activez les attributs et configurez-les pour utiliser votre buffer de positions.

```
// ... (code précédent)

var position = gl.getAttribLocation(program, "a_position");
gl.enableVertexAttribArray(position);
gl.vertexAttribPointer(
  position,
  2,
  gl.FLOAT,
  false,
```

```
  4 * Float32Array.BYTES_PER_ELEMENT,
  0
);

var texCoord = gl.getAttribLocation(program, "a_texCoord");
gl.enableVertexAttribArray(texCoord);
gl.vertexAttribPointer(
  texCoord,
  2,
  gl.FLOAT,
  false,
  4 * Float32Array.BYTES_PER_ELEMENT,
  2 * Float32Array.BYTES_PER_ELEMENT
);
```

7. Chargement et Configuration de la Texture:

- Créez une texture et chargez une image.
- Une fois l'image chargée, configurez la texture.

```
// ... (code précédent)

var texture = gl.createTexture();
var image = new Image();
image.src = "perso.jpg";
image.onload = function () {
  gl.bindTexture(gl.TEXTURE_2D, texture);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
  gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);

  draw();
};
```

8. Phase de Dessin:

- Définissez une fonction `draw` pour dessiner votre rectangle.
- Appelez cette fonction une fois la texture chargée.

```
// ... (code précédent)

function draw() {
  // Clear the canvas
```

```
  gl.clear(gl.COLOR_BUFFER_BIT);
  gl.useProgram(program);
  // Bind the texture
  gl.bindTexture(gl.TEXTURE_2D, texture);
  // Draw the rectangle
  gl.drawArrays(gl.TRIANGLE_FAN, 0, 4);
}
```