Computer Graphics - WebGL, Teil 2

1 Farbe

Uniform Variablen

Bisher habe wir für die Übergabe von Werten zwischen dem Programm und den Shadern *Attribut*-Variablen verwendet. Attribute erlauben es Daten zu einem Vertex mitzugeben, im ersten Teil der Übung war dies die Position der Vertex. Wenn wir farbige Objekte zeichnen wollen, so müssen wir unterscheiden ob wir die Farbe auch pro Vertex übergeben wollen oder ob wir eine Farbe pro Objekt (Dreieck) übergeben. Wir schauen uns zuerst den zweiten Fall an.

Für die Übergabe von Werten für ein ganzes Objekt gibt es in WebGL Variablen vom Typ `uniform`. Diese können sowohl im Vertex, wie auch im Fragment-Shader verwendet werden, vergleichen Sie dazu die entsprechende Folien in der WebGL Einführung. Da die Farben dann einem Pixel (bzw. Fragment) zugeordnet werden, müssen wir den Wert also an den Fragment-Shader übergeben. Dazu wird dort eine Variable vom Typ uniform deklariert.

```
precision mediump float;
uniform vec4 uColor;
void main() {
...
};
```

Um diese Variable im Programm setzen zu können, muss zuerst auch wieder Ihr Speicherort (Location) abgefragt werden. Danach kann mit den `uniform[234][fi](...)` Befehlen der Wert gesetzt werden. Je nach Typ werden verschieden Versionen des `uniform` Befehl's verwendet, zum Beispiel `uniform4f(vUniform, 1.0, 2.0, 3.0, 4.0)` um 4 Float Werte zu übergeben.

```
var ctx = {
  ...
  uColorId: -1
}
...
function setupAttributesAndUniforms() {
```

```
    ...
    ctx.uColorId = gl.getUniformLocation(shaderProgram, "uColor");
 }
 ...
 function draw() {
  gl.uniform4f(ctx.uColorId, 1.0, 0.0, 0.0, 1.0);
  gl.drawArrays(...)
}
```

Aufgabe 1: Verändern Sie das Programm aus der letzten Übung, sodass das sie die Farbe des Rechtecks im JavaScript Programm setzen können.

1.1 Varying Variablen

Bei den Beleuchtungen später werden wir sehen, dass sinnvollerweise jeder Ecke eines Objekts eine andere Farbe zugeordnet werden sollte. Dazu können die Farben wie die Positionen der Vertices als Attribute übergeben werden. Allerdings ist es nicht möglich sie direkt an den Fragment-Shader zu übergeben, da dieser ja die Positionen nicht mehr kennt. Sie müssen also vom JavaScript Program an den Vertex Shader und dann an den Fragment Shader weitergeleitet werden.

Dazu gibt es den Typ `varying`. Diese Variablen werden im Vertex Shader gesetzt und können dann im Fragment Shader ausgelesen werden. Die Variablen müssen in beiden Shadern gleich deklariert werden, sonst wird das Linken des Programms fehlschlagen.

```
varying vec4 vColor;
```

Aufgabe 2: Passen Sie nun das Programm so an, dass das Rechteck verschiedene Farben an den Eckpunkten erhält. Wie wird die Farbe im Innern des Rechtecks berechnet?

Hinweis: Um die Farben als Attribute zu übergeben, können Sie einen zweiten Buffer verwenden, mit den Farbdaten füllen und dem Attribut für die Farbe zuordnen. Beim Zeichnen mittels `drawArrays` müssen dann beide Buffer *enabled* sein. Ein andere Möglichkeit besteht darin nur ein Buffer zu verwenden, der sowohl die Positionen wie auch die Farben der Vertices enthält.

2 Texture Mapping

Um Objekte nicht nur an den Eckpunkten sondern auch im Innern einzufärben kann Texture Mapping verwendet werden. Dazu wird ein Bild auf die zu zeichnende Fläche projiziert.

Laden der Textur

Zum Laden einer Texture wird am besten ein HTMLImageElement Objekt verwendet welches mit `new Image()` erzeugt werden kann. WebGl benutzt des weiteren Texture Objekte (WebGLTextureObject), die mit `createTexture()` erzeugt werden können. Ein TextureObjekt wird mit `bindTexture()` gebunden, worauf der Inhalt und die Parameter der Textur spezifiziert werden können.

```
// keep texture parameters in an object so we can mix textures and objects
var lennaTxt = {
    textureObj: {}
};

/**
 * Initialize a texture from an image
 * @param image the loaded image
 * @param textureObject WebGL Texture Object
 */
function initTexture(image, textureObject) {
    // create a new texture
    gl.bindTexture(gl.TEXTURE_2D, textureObject);

    // set parameters for the texture
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
    gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
    gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.
    LINEAR_MIPMAP_NEAREST);
    gl.generateMipmap(gl.TEXTURE_2D);

    // turn texture off again
    gl.bindTexture(gl.TEXTURE_2D, null);
}

/**
 * Load an image as a texture
 */
function loadTexture() {
    var image = new Image();
    // create a texture object
    lennaTxt.textureObj = gl.createTexture();
    image.onload = function() {
        initTexture(image, lennaTxt.textureObj);
        // make sure there is a redraw after the loading of the texture
        draw();
    };
    // setting the src will trigger onload
    image.src = "lena512.png";
}
```

Spezifizieren der Texture Koordinaten

Zu jedem Vertex des Objekts müssen nun die Texture Koordinaten angegeben werden, die beschreiben welche Position des Bildes für diesen Vertex verwendet wird. Dies geschieht am Besten in einem eigenen Buffer:

```
// create the texture coordinates for the object
    var textureCoord = [
        0.0, 0.0,
        ...
    ];
    rectangleObject.textureBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, rectangleObject.textureBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoord), gl.
    STATIC_DRAW);
```

Anpassung der Shader

Das eigentliche Texture Mapping wird im Fragment Shader ausgeführt. Damit die Texture Koordinaten dort zur Verfügung stehen müssen sie vom JavaScript Program an den Vertex Shader und dann weiter an den Fragment Shader übergeben werden. Das Gleiche haben wir auch schon für die Farbwerte gemacht.

Vertex Shader

```
attribute vec2 aVertexTextureCoord;
...
varying vec2 vTextureCoord;

void main() {
    ...
    vTextureCoord = aVertexTextureCoord;
}
```

Fragment Shader

Im Fragment Shader wird nun nicht mehr eine Farbe an das Fragment zugewiesen, sondern der Wert wird aus der Textur ausgelesen und zwar an der Stelle, die durch die Texture Koordinaten bestimmt werden. Dafür wird noch eine uniform Variable sampler2D benutzt, die angibt welche Texture verwendet werden soll (siehe nächstes Kapitel).

```
precision mediump float;
```

```
varying vec2 vTextureCoord;
uniform sampler2D uSampler;

void main() {
    gl_FragColor = texture2D(uSampler, vTextureCoord);
}
```

Zeichnen

WebGL unterstützt mehrere Texturen, die auch gleichzeitig verwendet werden können. Die gewünschte TextureId muss aktiviert werden (`gl.activeTexture`), und dann kann die bereits geladene Texture als 2D Textur darauf gebunden werden. Zudem muss die im Fragment Shader verwendete Sampler Variable auf den Wert der TextureId gesetzt werden, i.e. 0 für `gl.TEXTURE0`, 1 für `gl.TEXTURE1` usw. Die Id der sampler Variable muss wie üblich mit `gl.getUniformLocation(...)` ermittelt werden.

```
    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, lennaTxt.textureObj);
    gl.uniform1i(ctx.uSampler2DId, 0);
```

Aufgabe 3: Zeichnen sie ein Rechteck mit Texture Mapping.