Computer Graphics - WebGL, Teil 1

Thomas Koller *

23. September 2022

1 JavaScript

Die Programmierübungen in diesem Modul werden mit JavaScript durchgeführt. Dabei werden nur die einfachsten Sprachelemente benutzt, trotzdem empfiehlt es sich, ein JavaScript-Tutorial durchzuarbeiten oder ein entsprechendes Buch zu verwenden, zum Beispiel:

- Eric T. Freeman, Elisabeth Robson
  Head First JavaScript Programming
  O'Reilly, Sebastopol 2014
  ISBN: 978-1-449-34013-1
- David Flanagan
  JavaScript. The Definitive Guide
  O'Reilly, Sebastopol 2011, 6th edition
  ISBN: 978-0-596-80552-4

Es wird empfohlen, eine Entwicklungsumgebung für JavaScript zu verwenden. Für die Produkte von jetbrains (`www.jetbrains.com`) erhalten sie mit ihrer HSLU-Email kostenfreie Lizenzen, WebStorm ist eine Entwicklungsumgebung von jetbrains für JavaScript. Natürlich dürfen Sie auch gerne andere Umgebungen wie zum Beispiel eclipse oder netbeans verwenden.

2 Einführung

In dieser Übung sollen Sie erste Objekte mit WebGl zeichnen. Um den Einstieg einfach zu halten, werden wir zuerst die Konzepte anhand eines 2D-Beispiels anschauen und danach das Ganze nach 3D erweitern.

WebGL verfügt über eine programmierbare Graphik-Pipeline. Dies bedingt, dass bereits zum Zeichnen einfacher Objekte einiges programmiert werden muss, dafür können dann später interessante Graphikeffekte relativ einfach integriert werden.

*Bearbeitung von Joachim Wirth

Schritt 1: WebGL aufsetzen, Hintergrund zeichnen

In einem ersten Schritt soll WebGL aufgesetzt und mit den ersten WebGL-Befehlen der Hintergrund gezeichnet werden. Das Programm besteht dabei aus aus einem HTML-Dokument, das die Zeichenfläche (Canvas) aufsetzt und die JavaScript-Dateien definiert, die das eigentliche Programm enthalten. Der folgende Code definiert die Zeichenfläche und lädt die JavaScript-Dateien, die für das Projekt benötigt werden. Diese werden im folgenden genauer vorgestellt.

```
<!DOCTYPE html >
<html lang ="en">
<head>
   <meta charset="UTF-8">
   <title >Computer Graphics </ title >
   <script type="text/javascript" src="webgl-debug.js"></script>
   <script type="text/javascript" src="shaderUtils.js"></script>
   <script type="text/javascript" src="gl-matrix.js"></script>
   <script type="text/javascript" src="exercise-1.js"></script>
</head >
<body>
   <h1>Computer Graphics</h1>
   <canvas id="myCanvas" width="800" height="600"></canvas>
</body>
</html>
```

Im Javascript-Code muss nun zuerst das Canvas-Objekt aus dem Dokument gefunden und dann die `getContext()`-Methode aufgerufen werden, um das Kontext-Objekt zu erhalten. Dafür wird der Parameter „webgl“ benötigt, damit ein WebGL Kontext zurückgegeben wird und nicht ein 2D-Kontext, der über ein anderes Interface angesprochen wird. Sämtliche WebGl-Funktionen werden nun als Methoden des Kontext-Objekts aufgerufen. Der folgenden Code zeigt die Basisstruktur der Javascript-Hauptdatei:

```
// Computer Graphics
// WebGL Exercises

// Register function to call after document has loaded
window.onload = startup;

// the gl object is saved globally
var gl;

// we keep all local parameters for the program in a single object
var ctx =
{
    shaderProgram: -1,
    // add local parameters for attributes and uniforms here
};
```

```
// we keep all the parameters for drawing a specific object together
var rectangleObject =
{
    buffer: -1
};

/**
 * Startup function to be called when the body is loaded
 */

function startup ()
{
    "use strict";
    var canvas = document.getElementById ("myCanvas");
    gl = createGLContext (canvas);
    initGL();
    draw();
}

/**
 * InitGL should contain the functionality that needs to be executed only once
 */

function initGL ()
{
    "use strict";

    ctx.shaderProgram = loadAndCompileShaders (gl, 'vertex-shader.glsl',
        'fragment-shader.glsl');
    setUpAttributesAndUniforms();
    setUpBuffers();

    // set the clear color here
    gl.clearColor (1, 0, 0, 1);
    // add more necessary commands here
}

/**
 * Setup all the attribute and uniform variables
 */

function setUpAttributesAndUniforms ()
{
    "use strict";
    // add code here to get the ids of attributes and uniform variables
    // from the shaders
}
```

```
/**
 * Setup the buffers to use. If more objects are needed this should be
 * split in a file per object.
 */

function setUpBuffers ()
{
    "use strict";
    // add code here to setup the buffers for drawing an object
}

/**
 * Draw the scene.
 */

function draw ()
{
    "use strict";
    console.log ("Drawing");
    gl.clear(gl.COLOR_BUFFER_BIT);
    // add drawing routines here
}
```

In den Funktionen `initGL()` und `draw()` kann mittels `gl.someFunction()` auf Befehle von WebGL zugegriffen werden.

Debug-Version der WebGL-Bibliothek

WebGL gibt keine Fehlermeldungen zurück, jedoch kann jeweils mit dem Befehl `glError` abgefragt werden, ob ein Fehler existiert. Dies macht jedoch das Debuggen mühsam. Daher gibt es eine Bibliothek `webgl-debug.js`, die nach jedem `gl`-Befehl automatisch diese Funktion aufruft und allfällige Fehler auf die Console schreibt, siehe auch:

`http://www.khronos.org/webgl/wiki/Debugging`

Im obigen Code ist dies bereits eingebaut. Die aufgerufene Funktion `createGLContext` ist im file `shaderUtils.js` definiert. Im JavaScript code:

```
function createGLContext (canvas)
{
    "use strict";
    // get the gl drawing context
    var context = canvas.getContext ("webgl");
    if (! context)
    {
        alert ("Failed to create GL context");
    }
    // wrap the context to a debug context to get error messages
    return WebGLDebugUtils.makeDebugContext (context);
}
```

Aufgabe 1

Im Code oben wird der Hintergrund des Canvas rot eingefärbt. Ändern Sie die Hintergrundfarbe auf hellgrau.

Frage: Wieso befindet sich `gl.clearColor()` in der Funktion `initGL()`, `gl.clear()` aber in der Funktion `draw()`?

Schritt 2: Zeichnen eines Rechtecks

Shader-Programme

Als nächstes soll ein Rechteck gezeichnet werden. Dazu müssen nun der Vertex-Shader und der Fragment-Shader implementiert werden, da WebGL eine vollständig programmierbare Grafik-Pipeline beinhaltet. Am einfachsten werden die Shader von externen Files geladen. In der Datei `shaderUtils.js` befindet sich die Funktion `loadAndCompileShaders()` zur Verfügung, die die Shader lädt, kompiliert und zusammenfügt (linking), dies ist im Code der Funktion `loadAndCompileShaders()` ersichtlich, die auch im file `shaderUtils.js` definiert ist. Falls es beim Kompilieren der Shader zu einem Problem kommt, wird ein Fenster mit dem entsprechenden Fehler angezeigt.

Der Vertex- und der Fragment-Shader werden in der OpenGL Shading Language (GLSL) implementiert. Diese benutzt eine C-ähnliche Syntax.

Der Vertex-Shader wird für jeden Vertex aufgerufen. Seine Hauptaufgabe besteht darin, die Position des Vertex auf dem Canvas zu berechnen. Diese Position wird in sogenannten normalized screen coordinates angegeben, die von -1 zu +1 gehen.

Der Fragment-Shader wird für jeden zu zeichnenden Pixel (jedes Fragment) aufgerufen. Seine Hauptaufgabe besteht darin, die Farbe des Pixels zu setzen.

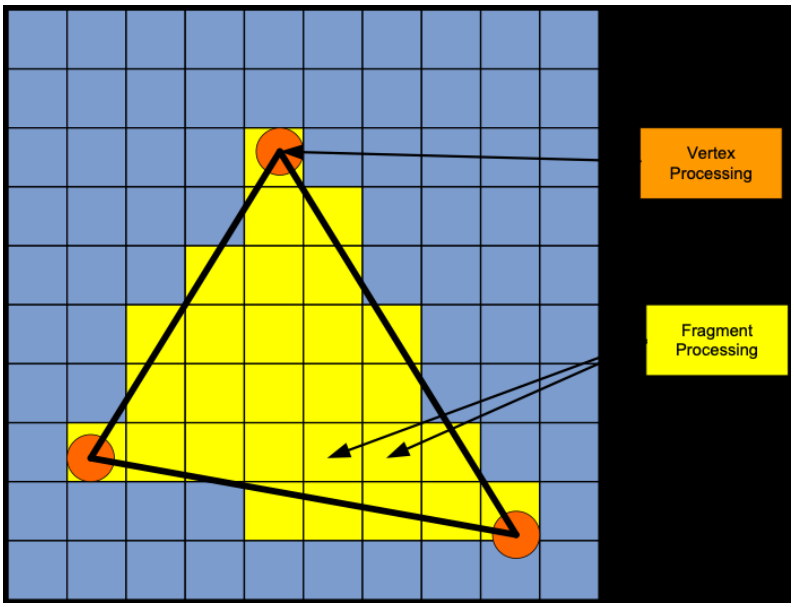

Abbildung 1: Funktionsweise der Shader: Der Vertex-Shader wird für jeden Vertex aufgerufen, der Fragment-Shader für jedes Pixel.

Zeichnen von Objekten

Zum Zeichnen einer Figur muss zuerst spezifiziert werden, wie die Information vom JavaScript-Programm zu den Shader-Programmen gelangt. In dieser Übung möchten wir Vertex-Positionen übergeben, die dann als Endpunkte von Linien oder Eckpunkte von Dreiecken dienen. Wir benutzen daher eine Variante, in der ein ganzer Buffer von Werten übergeben werden kann. In einer anderen Variante könnten wir jeweils einzelne Werte übergeben, dies eignet sich jedoch nicht für Positionen. Es ist folgendes Vorgehen notwendig:

1. Spezifikation des entsprechenden Attributes im Vertex-Shader-Programm
2. Abfragen des Index dieses Attributs im JavaScript-Programm
3. Erzeugen der Vertex Daten (als Float32Array)
4. Erzeugen des WebGL Buffers
5. Verbinden des Buffers mit dem Attribut Index
6. Zeichnen des Arrays

Im Beispiel des Vertex-Shaders oben ist bereits eine Attribut-Variable `aVertexPosition` definiert (Schritt 1). Um dieser Variablen Werte zuordnen zu können, muss sie im JavaScript Program abgefragt werden (Schritt 2):

```
// we keep all local parameters for the program in a single object

var ctx =
{
    shaderProgram: -1,
    aVertexPositionId: -1
};

...

function setupAttributes ()
{
    // finds the index of the variable in the program
    ctx.aVertexPositionId = gl.getAttribLocation (ctx. shaderProgram,
        "aVertexPosition");
}
```

Der Vertex-Buffer muss erzeugt und mit einem `Float32Array`-Objekt gefüllt werden (Schritte 3 und 4):

```
// we keep all the parameters for drawing a specific object together

var rectangleObject =
{
    buffer: -1
};

...

function setupBuffers ()
{
    rectangleObject.buffer = gl.createBuffer ();
    var vertices =
    [
        0, 0,
        ...
    ];
    gl.bindBuffer (gl.ARRAY_BUFFER, rectangleObject.buffer);
    gl.bufferData (gl.ARRAY_BUFFER, new Float32Array (vertices),
        gl.STATIC_DRAW);
}
```

Nun muss noch der Buffer mit dem Attribut verbunden werden, und in WebGL muss angegeben werden, dass die Werte für das Attribut vom Buffer übernommen werden soll (Schritt 5).

```
function draw ()
{
    gl.clear (gl.COLOR_BUFFER_BIT);
    gl.bindBuffer (gl.ARRAY_BUFFER, rectangleObject. buffer);
    gl.vertexAttribPointer (ctx.aVertexPositionId, 2, gl.FLOAT,
        false, 0, 0);
    gl.enableVertexAttribArray (ctx.aVertexPositionId);
    ...
}
```

Schlussendlich kann das Array gezeichnet werden (Schritt 6). WebGl wird nun für jeden Wert des Arrays den Vertex-Shader aufrufen.

```
function draw ()
{
    ...
    gl.drawArrays (gl.LINE_LOOP, 0, 4);
}
```

Aufgabe 2

Fügen sie die vorgegebenen Programmteile zusammen und ergänzen Sie sie, so dass ein ungefüllt weisses Rechteck auf dem Hintergrund gezeichnet wird.

Aufgabe 3

Verändern Sie das Program, so dass ein gefülltes Rechteck gezeichnet wird. Dazu müssen Sie `gl.drawArrays()` eine andere Konstante übergeben. Die möglichen Konstanten sind in Abbildung 2 spezifiziert.

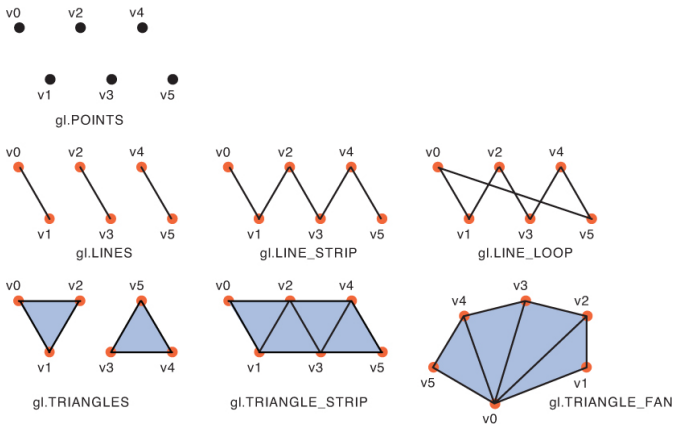

Abbildung 2: Die verschiedenen Parameterwerte von `drawArrays` werden verwendet, um Punkte, Linien und Dreiecke darzustellen.

Falls Sie bereits fertig sind, können Sie mit den folgenden Aufgaben fortfahren, die dann auch Teil der nächsten Übung sind. Dort werden die dazu benötigten Schritte dann genauer erklärt.

Aufgabe 4*

Ergänzen sie das Program, so dass Sie die Farbe des Rechtecks übergeben können.

Aufgabe 5*

Zeichnen sie mehrere Rechtecke in verschiedenen Farben.