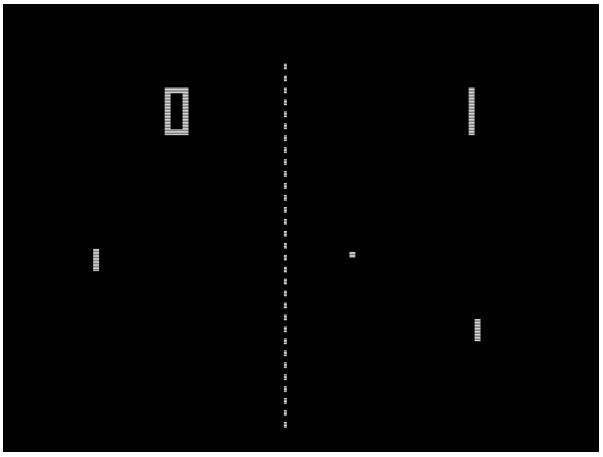
Computer Graphics - WebGL, Teil 3

1 Einführung

In dieser Übung sollen sie ein einfaches Pong Spiel in WebGL entwickeln. Bei Pong steuern sie ein Paddle und müssen versuchen den Ball zu treffen, so dass er nicht über ihre Grundlinie gelangt. Als Ausgangslage dient uns eine Version der letzten Übung in der Sie ein Quadrat zeichnen können und die Farbe über ein uniform Variable setzen (oder über attribute und varying). Diese Version steht sonst auch auf Illias zur Verfügung.



Screenshot des Original Atari Pong Spiels von 1972

2 Transformationen

Für 3D Transformationen werden in WebGL $4x4$ Transformationsmatrizen verwendet, für 2D Transformationen $3x3$ Matrizen. Die Shader unterstützen Operationen auf Matrizen direkt, in WebGL sonst sind aber keine Operationen dafür vorhanden. Es gibt jedoch mehrere JavaScript Bibliotheken, die diese Funktionalität anbieten, wir verwenden für die Übungen *gl-matrix.js*. Die Datei ist auf Illias in Ihrem Startprojekt von letzter Woche bereits eingebunden. Sie ist

sonst auch auf `https://github.com/toji/gl-matrix` zu finden. In der HTML Datei sollte das File auch schon referenziert werden:

```
<script type="text/javascript" src="gl-matrix.js"></script>
```

Die Transformationen müssen wieder vom JavaScript Program an den Vertex Shader übergeben werden. Dafür verwenden wir wiederum uniform Variablen.

Als erstes möchten wir die Koordinaten des Spielfelds angeben können, bisher haben wir unsere Objekte ja in normalisierten Bildschirmkoordinaten im Bereich von -1 nach 1 angegeben. Für dieses *Weltkoordinatensystem* möchten wir etwas wählen, was uns die Berechnungen vereinfacht. Wir behalten deshalb (0,0) als Zentrum des Bildes, wählen dann aber Pixelkoordinaten von $-w/2$ nach $w/2$ bzw. von $-h/2$ nach $h/2$ für die x- und y-Achsen.

Um eine Matrix zu erzeugen, die uns diese Abbildung liefert können wir eine Skalierung (siehe unten) verwenden:

```
// Set up the world coordinates
var projectionMat = mat3.create();
mat3.fromScaling(projectionMat, [2.0/gl.drawingBufferWidth, 2.0/gl.
   drawingBufferHeight]);
gl.uniformMatrix3fv(ctx.uProjectionMatId, false, projectionMat);
```

Die Matrix müssen Sie nun entsprechend im Vertex Shader einbauen.

Um die Objekte zu verändern, verwenden wir eine zweite 3x3 Matrix `modelMat`. Wir verwenden nur diese Matrix für alle Objekt- (und später Kamera-) Transformationen und passen diese für jedes gezeichnete Objekt an.

Die gl-Matrix Bibliothek bietet verschiedene Möglichkeiten an, eine 3x3 Matrix für eine spezielle Abbildung zu erzeugen, wie zum Beispiel

- `mat3.fromRotation(out, rad)`
- `mat3.fromScaling(out, v)`
- `mat3.fromTranslation(out, v)`
- `mat3.fromValues(m00, m01, m02, m10, m11, m12, m20, m21, m22)`

Alle diese Operationen füllen die mitgegebene Matrix `out` mit der entsprechenden Transformationsmatrix. Weitere Angaben finden Sie auf der Dokumentationsseite der Bibliothek `http://glmatrix.net/`.

Aufgabe 1: Ergänzen Sie das Program mit den Matrizen und zeichnen Sie ein 100x100 Quadrat in der Mitte des Bildes. Verwenden Sie dazu zwei Transformationsmatrizen `uProjectionMat`

und `uModelMat` im Vertex Shader. Wie müssen Sie nun die aktuelle WebGL Position (`gl_Position`) berechnen?

3 Zusammengesetzte Transformationen

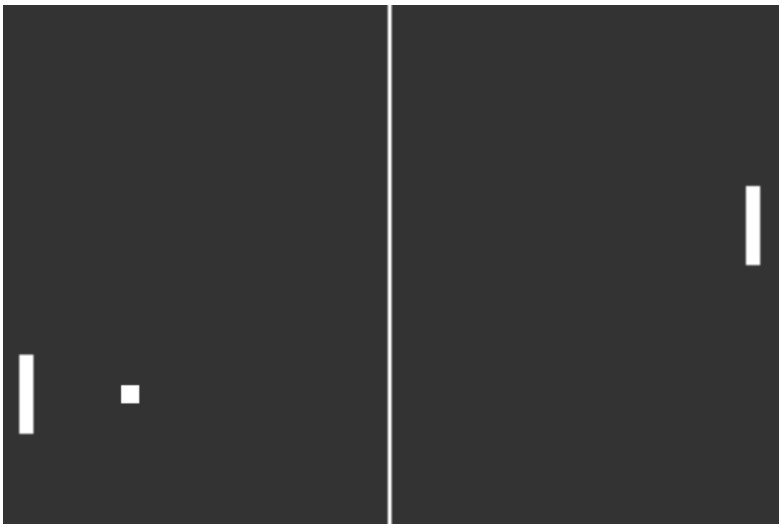
In Grafik Programmen werden oft jeweils mehrere Transformation hintereinander ausgeführt. Wie Sie ja bereits wissen, werden dabei die Matrizen multipliziert. Die gl-Matrix Bilbiothek stellt Funktionen zur Verfügung, die die neue Transformation auf eine bereits bestehenden Transformation anwendet und zurückgibt:

- `mat3.rotate(out, a, rad)`
- `mat3.scale(out, a, v)`
- `mat3.translate(out, a, v)`

Um zusammengesetzte Transformationen zu verwenden, berechnen wie die Matrix, die dieser Transformation entspricht (als `modelMat`) und setzen Sie dann entsprechen für den Shader. Im Shader werden keine weiteren Transformationsmatrizen verwendet.

Aufgabe 2: Verändern Sie das Programm, sodass zwei 100x100 Quadrate an verschiedenen Orten gezeichnet werden.

Aufgabe 3: Erweitern Sie nun das Programm, so dass die Paddles, der Ball und die Mittellinie (nicht gestrichelt) gezeichnet werden indem Sie das ursprüngliche Quadrat transformieren und mit verschiedenen Transformationen zeichnen. Ihr Screen sollte nun wie folgt aussehen:



Es empfiehlt sich die Positionen des Balls und der Paddels in einer eigenen Datenstruktur zu speichern um dann einfacher die Spiellogik implementieren zu können.

Animation

Für das Spiel müssen die Objekte dann animiert werden, da sich der Ball und die Paddles ja bewegen sollen. Um eine Animation zu programmieren kann die Funktion

```
window.requestAnimationFrame(callback);
```

verwendet werden. Diese informiert den Browser, das eine Animation gewünscht wird und ruft die Callback Funktion vor dem nächsten redraw auf. In der Callback Funktion muss dann wiederum `requestAnimationFrame` aufgerufen . Die Callback Funktion wird vom System mit einem Timestamp Parameter aufgerufen, der es ermöglicht herauszufinden wie viel Zeit seit dem letzten Aufruf vergangen ist. Dies sollte dann bei der Bewegung des Balls und der Paddles berücksichtigt werden.

```
function drawAnimated(timeStamp) {
   // calculate time since last call
   // move or change objects

   draw();
   // request the next frame
   window.requestAnimationFrame(drawAnimated);
}
```

Spiel

Als weitere Komponente für das Spiel sollen sie das Paddle noch bewegen können. Es gibt keine Möglichkeit den Status einer Taste in javascript direkt abzufragen, jedoch können sie mit den normalen Events (keydown, keyup) arbeiten und einen Event Handler entweder im HTML Code oder mittels `windows.addEventHandler()` hinzufügen. Diese Funktionen sind im *pong_start* Projekt auf Illias bereits eingebaut.

```
// Key Handling
var key = {
    _pressed: {},

    LEFT: 37,
    UP: 38,
    RIGHT: 39,
    DOWN: 40
};
```

```
function isDown (keyCode) {
    return key._pressed[keyCode];
}

function onKeydown(event) {
    key._pressed[event.keyCode] = true;
}

function onKeyup(event) {
    delete key._pressed[event.keyCode];
}

}
```

Aufgabe 5: Steuern sie eines der Paddles mit den Pfeiltasten auf der Tastatur.

Nun können sie noch die Spiellogik ergänzen. Dazu gehört:

- Bewegen des Computer Paddles
- Abprallen des Balls an der oberen und unteren Kante
- Detektieren wann der Ball die Paddles trifft und davon abprallt
- Detektieren wann der Ball nicht geschlagen wird.

Aufgabe 6: Programmieren Sie das Spiel fertig.

Aufgabe 7: Weitere Möglichkeiten[1]:

- Zeigen Sie die Leben/Versuche (3) jedes Spielers an.
- Machen Sie das Spiel farbiger.
- Ändern Sie die Farbe anhand des x-Wertes der Pixelposition
- Verwenden Sie Kreise als Bälle
- Programmieren Sie einen besonders intelligenten Gegner.
- Wieso Pong, wenn Sie auch Breakout implementieren könnten?
- usw.

[1] optional