Witold Alda

Computer Graphics. Lab 1.

WebGL: 3D Graphics in Internet browsers

WebGL is a crippled version of OpenGL library, which is dedicated to presenting 3D graphics in Internet browsers. Natural environment for WebGL is Javascript, which in turn is immersed in HTML5 file.

Most of browsers handles WebGL commands properly. However older versions of Internet Explorer, need installing add-ons.

While programming in WebGL one may use a ”low-level” approach – applying directly vertices, buffers, matrix algebra for transformations and projections, as well as shaders – thus resembling current OpenGL style. But it is also possible to employ existing libraries which move graphics programming to an object level. The latter approach is much more intuitive, simpler and allows us to get interesting graphical effects very fast, however for the price of some restrictions in rendering.

In the current lab, as well as in the following ones, we will use mainly ”high-level” libraries. In WebGL there are quite a few of them and they are very popular. On the contrary, in OpenGL such libraries extinct, however this may be temporary.

High Level Libraries

Coding using WebGL functions and shaders only is generally pretty tedious. In practice we often try to escape from such solution and we search for libraries which help us to program more intuitively leaving technical, low level aspects apart.
WebGL offers at least a few such solutions, which make programming easier and cut the code length significantly.
Among popular libraries we may enlist, e.g.:

- GLGE (*http://www.glge.org/*),
- SceneJS （*http://www.scenejs.org/*),
- Babylon.js (*http://www.babylonjs.com/*).

There are however much more of tchem.

Every library has its own philosophy of building and rendering scenes, however all tend achieve high level character of programming and user-friendly programming interface.

In this and subsequent labs we will base on the three.js (*https://github.com/mrdoob/three.js/*) library.

Three.js Library

In three.js library a scene is built of objects. The scene itself is also an object – a container – in which you put all other objects: geometry, but also light sources and cameras.

Basic objects, which we generate according to three.js classes are as follows:

- Renderers (There are at least two of them: *Canvas Renderer* and *WebGL Renderer*, the latter with larger capabilities but also with higher hardware needs.
- Ready-made geometry shapes supplied by the library (such as cube, arbitrary box, torus etc.) as well made by hand as a polygon mesh.
- Objects defined as a viewer (*camera*), its coordinates, points it looks at, viewing angle, etc.
- Light sources which illuminate the scene.
- Materials assigned to geometry shapes, which define their color and the light is reflected and/or refracted.

Three.js objects have their internal variables (properties) which should be assigned desired values, as well as methods. Very useful, however not complete documentation which describes object structure in on web page *threejs.org*

Examples to run

Example 00

Example 00 does not employ any other library, but relies only core WebGL functions. It uses shaders – we include it here only to see how a code with shaders looks like and we don't modify it in any way. We can see that in pure WebGL even simplest graphics needs a relatively long code.

Example 01

Example 01 draws white triangle and square. Current example and subsequent ones in this set use the three.js library, however figures are built manually, starting from vertices – non typical way in three.js.

After a brief code analysis:

1. Change the color of the triangle and square from white to another
2. Using three triangles and a quadrangle build a tree, while using a square and a triangle – build a house. You can of course add a window, door or a chimney, or build something completely different.

Example 02

Example 02 draws a triangle and square with interpolated colors. Read from the example how a color is assigned to a single vertex.

Example 03

Example 03 rotates both objects. Analyze the code and change the rotation speed and/or the axis of rotation.

Example 04

Example 04 demonstrates 3D solids built manually and rotating.