Prof. Dr. Michael Krone
Marco Schäfer
Marcel Bock
Big Data Visual Analytics in Life Sciences
Department of Computer Science & IBMI
Faculty of Science

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Scientific Visualization **SS/2021**
Assignment 1 **Due: May 6th, 2021, 9:00 am**

Hello Cube (6+1 points)

In this assignment you will setup your development environment based on a little project skeleton to make programming a bit easier. This development environment is necessary for all upcoming tasks including assignments and tutorials. After that you will create your first own 3D cube!

Important Notice: This first exercise sheet should be done by each one of you – do not submit it in pairs! We will start working in pairs starting with exercise sheet 2.

1 Checking your system (1 point)

All programming exercises will be web-based using JavaScript as the main programming language. JavaScript is natively available on all modern systems and browsers, that is, you can use the operating system of your choice. Since Scientific Visualization focuses on interactive, three-dimensional depictions of data, we will need WebGL for GPU-accelerated 3D graphics. To make sure that everything runs correctly on your device, you should open the WebGL Report website in your browser: `https://webglreport.com/`

- Both tabs (WebGL 1 & WebGL 2) should read "This browser supports WebGL 1/2" in the upper bar. If not, you should update your browser or use another one (e.g., Firefox, Chrome, or Opera).
- Copy the information that is shown in the upper left part on each tab (only from *"Platform: ..."* to *"Major Performance Caveat: ..."* – **not** the *"Supported Extensions: ..."*) to a simple text document (a simple .txt file is fine).

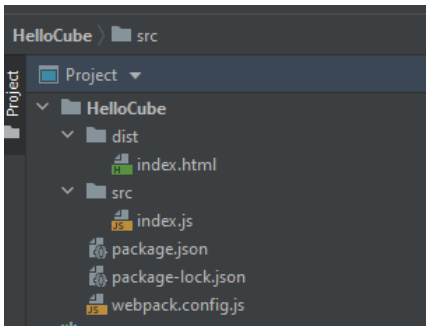
2 Let's start with the development environment (2.5 points)

Basic Setup

As first step download the project skeleton from ILIAS and unzip it to a place of your choice. If not already done, install a code editor you want to work with [e.g., Webstorm (Student License freely available), Visual Studio Code, Eclipse, Atom, Notepad++, or any other].

- **download and unzip project skeleton**:
  Ilias course → Assignments → Assignment_1 → assignment1_code_skeleton.zip

The structure of the unzipped skeleton should look like this:



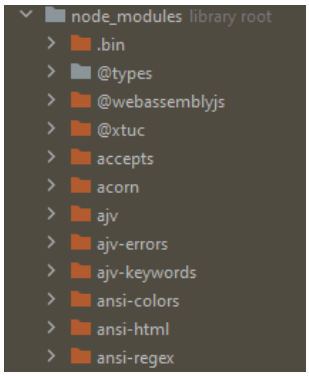
We will work with a web server that will serve us our webpage including HTML documents (.html), JavaScripts (.js) and cascading style-sheets (.css). We need it also to do more advanced exercises in later tasks (e.g., file reading). The first step is to install *node.js* as a platform for building network applications. The installation also brings the package manager *npm*, which is used to easily install needed components for our application.

- **install node.js**: `https://nodejs.org/en/download/`
  (choose the installer corresponding to your operating system)

Now navigate to the project folder (`./HelloCube/`), open a console/terminal and type in the following line: (be sure your current working directory of the console is set to project folder path):

- **npm install** (info: install all packages listed in package.json)

Now the project folder should contain a new directory *node_modules*. This is the folder with all the installed packages from the step above.



The installed packages are *webpack*; *webpack-cli*; *webpack-dev-server* and *three*. The *node_modules* folder contains more packages because of the dependencies of the mentioned main packages. We are using *webpack* as a module bundler. It searches for all dependencies of our web-application and bundles them in one file, this is, we have not to separately import every library or module. Also, we use *webpack-dev-server* as our webserver to serve our web-application and the therefore needed files. It also has the advantage to automatically reload our webpage if the code changes (easier development). The last package is *three* which is a JavaScript 3D library we use to create our visualizations.

Server Setup

Now we configure webpack and our web-server. Open *webpack.config.js* and have a look on the following lines of code and write them into the config-file: (**Info:** Whenever your decide to copy the code be sure that all quotes ["] are the same and that there are no unnecessary white-spaces in the paths marked with these quotes)

```
// use "path" package
const path = require ("path");
module . exports = {
    // a empty placeholder to define custom rules and more
    // (e.g. set loaders for specific file types)
    module : {
        rules : [
        ],
    },
    optimization : {
        minimize : false
    },
    // set the webpack mode
    mode : "development",
    // set the entry point (main file of the web-application)
    entry : "./src/index.js",
    // define the name of the bundle file and its location to store it
    output : {
        // name of the bundle file
        filename : "main.js",
        // location to store it (here : "working dir"/dist )
        path : path . resolve ( __dirname , "./dist")
    },
    devtool : "inline-source-map",

    // configure the webserver
    devServer : {
        writeToDisk : true ,
        // set the path to the folder containing the web -app
            contentBase : "./dist",
        // you can also use another port, it is just custom to use 8080
        port : 8080 ,
        open : false
    },
};
```

Startpage Setup

As next step we add missing code to our *index.html* which is the automatically called web-page when the url `http://localhost:8080` is requested. Open *index.html*, read and add the following missing line of code to the <*body*> section:

```
<body>
    <script type="module" src="main.js"></script>
</body>
```

This will tell your browser to load the JavaScript file *main.js*, which is our by webpack bundled index.js file with all its dependencies.

Checkpoint 1: Test current state

Now test the current state by typing *npm run start* into the system console and call the url `http://localhost:8080` in your browser. Open the web console (normally F12, depends on your browser) and try to disable caching of websites or use a private tab to avoid caching. Reload the page. You should see a blank page with *Hello Cube - Scientific Visualization 2021 - Assignment 1* in the upper left corner and **no ERROR** in the web console. Otherwise check for typos and check the instructions so far again. **Info:** While debugging your code, you have not to restart the web-server and to reload the web-page to let your changes take place this happens automatically by the webpack-dev-server as mentioned earlier.

3 Creating your first own cube in a 3D Scene (2.5 points)

After we setup the development environment we work now on the main visualization part. To get the first 3D object on screen we have to add some instructions to our still almost empty *index.js* file to:

- set a new **Scene**
- add a **Camera**
- create a **Geometry** (cube)
- assign a **Material** to the geometry
- create a **Mesh** of the geometry
- create a **Renderer**

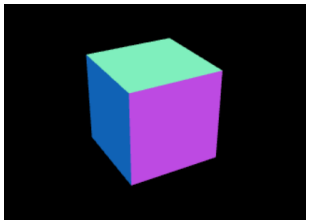
Add to *index.js* the following lines of source code. As question task you should solve/answer for yourself: Look for a more detailed description of the different objects and functions. What they are good for? Can additional parameter be passed to these function/objects? If yes which one?

```
// defining the variables
let camera, scene, renderer;
let geometry, material, mesh;

//creating the scene
scene = new THREE.Scene();
// adding a camera PerspectiveCamera( fov, aspect, near, far)
camera = new THREE.PerspectiveCamera( 70, window.innerWidth / window.
    innerHeight, 0.01, 10 );
// set the camera position x,y,z in the Scene
camera.position.set(0,0,1);

// create a geometry (a cube [width, height, depth])
geometry = new THREE.BoxGeometry(0.2,0.2,0.2 );
// create a material
material = new THREE.MeshNormalMaterial();

// create a mesh with the defined geometry and material
mesh = new THREE.Mesh( geometry, material );
// add mesh to current scene
scene.add( mesh );

/* +++ create a WebGLRenderer +++
 * enables antialiasing (nicer geometry: borders without stair effect)
```

```
 */
renderer = new THREE.WebGLRenderer( {antialias: true}   );
// get and set window dimension for the renderer
renderer.setSize( window.innerWidth, window.innerHeight );
// add dom object(renderer) to the body section of the index.html
document.body.appendChild( renderer.domElement );
/* rotates the cube on x and y
 * otherwise there is only a rectangle to see!
 */
mesh.rotation.x += 0.5;
mesh.rotation.y += 0.5;

/* +++ render the scene +++
 * gives the renderer the scene with the geometry
 * and sets also the camera
 */
renderer.render( scene, camera );
```

Checkpoint 2: Test final state

After you saved your code you should see the following colored cube on the web-page:
(if not, see instructions of section 2 Checkpoint 1: Test current state)



4 Bonus exercise (1 point)

Try to let the cube rotate continuously on two axes over the time.

Handing in

For the Hand-in, write your name within all code files. Copy the text file with the information you collected in section 1 to your project folder. Create a ZIP archive of your project folder and **PLEASE EXCLUDE** the folder **NODE_MODULES** from your archive. Name it with your last and first name, and the assignment number,
e.g. *schaefer_marco_assignment1.zip*, and upload it to Ilias: `Assignments` → `Assignment_1`.