Testing with NPM

This article shows how to get three.js into a node.js environment so that you can execute automated tests. Tests can be run on the command line, or by automated CI tools like Travis.

The short version

If you're comfortable with node and npm,

```
$ npm install three --save-dev
```

✓

and add

```
var THREE = require('three');
```

to your test.

Create a testable project from scratch

If you're not familiar with these tools, here's a quick guide (for linux, the installation process will be slightly different using windows, but the NPM commands are identical).

Basic setup

1. Install npm and nodejs. The shortest path typically looks something like

```
$ sudo apt-get install -y npm nodejs-legacy
# fix any problems with SSL in the default registry URL
$ npm config set registry http://registry.npmjs.org/
```

2. Make a new project directory

   ```
   $ mkdir test-example; cd test-example
   ```

3. Ask npm to create a new project file for you:

   ```
   $ npm init
   ```

   and accept all defaults by hitting Enter on all the prompts. This will create package.json.

4. Try and start the test feature with

   ```
   $ npm test
   ```

   This will fail, which is expected. If you look in the package.json, the definition of the test script is

   ```
   "test": "echo \"Error: no test specified\" && exit 1"
   ```

Add mocha

We're going to use mocha.

1. Install mocha with

   ```
   $ npm install mocha --save-dev
   ```

   Notice that node_modules/ is created and your dependencies appear in there. Also notice that your package.json has been updated: the property devDependencies is added and updated by the use of --save-dev.

2. Edit package.json to use mocha for testing. When test is invoked, we just want to run mocha and specify a verbose reporter. By default this will run anything in test/ (not having directory test/ can run into npm ERR!, create it by mkdir test)

   ```
   "test": "mocha --reporter list"
   ```

3. Rerun the test with

   ```
   $ npm test
   ```

   This should now succeed, reporting 0 passing (1ms) or similar.

Add three.js

1. Let's pull in our three.js dependency with

   ```
   $ npm install three --save-dev
   ```
   ✓

   - If you need a different three version, use

     ```
     $ npm show three versions
     ```
     ✓

     to see what's available. To tell npm the right one, use

     ```
     $ npm install three@0.84.0 --save
     ```

     (0.84.0 in this example). --save makes this a dependency of this project, rather than dev dependency. See the docs here for more info.

2. Mocha will look for tests in test/, so let's

   ```
   $ mkdir test
   ```

3. Finally we actually need a JS test to run. Let's add a simple test that will verify that the three.js object is available and working. Create test/verify-three.js containing:

   ✓

   ```
   var THREE = require('three');
   var assert = require("assert");
   ```

```
describe('The THREE object', function() {
  it('should have a defined BasicShadowMap constant', function() {
    assert.notEqual('undefined', THREE.BasicShadowMap);
  }),

  it('should be able to construct a Vector3 with default of x=0',
function() {
    var vec3 = new THREE.Vector3();
    assert.equal(0, vec3.x);
  })
})
```

4. Finally let's test again with $ npm test. This should run the tests above and succeed, showing something like:

```
The THREE object should have a defined BasicShadowMap constant: 0ms
The THREE object should be able to construct a Vector3 with default
of x=0: 0ms
2 passing (8ms)
```

Add your own code

You need to do three things:

1. Write a test for the expected behaviour of your code, and place it under test/. Here is an example from a real project.

2. Export your functional code in such a way that nodejs can see it, for use in conjunction with require. See it here.
3. Require your code into the test file, in the same way we did a require('three') in the example above.

Items 2 and 3 will vary depending on how you manage your code. In the example of Physics.js given above, the export part is right at the end. We assign an object to module.exports:

```
//=============================================================================

// make available in nodejs
//=============================================================================

if (typeof exports !== 'undefined')
{
  module.exports = Physics;
}
```

Dealing with dependencies

If you're already using something clever like require.js or browserify, skip this part.

Typically a three.js project is going to run in the browser. Module loading is hence done by the browser executing a bunch of script tags. Your individual files don't have to worry about dependencies. In a nodejs context however, there is no index.html binding everything together, so you have to be explicit.

If you're exporting a module that depends on other files, you're going to have to tell node to load them. Here is one approach:

1. At the start of your module, check to see if you're in a nodejs environment.
2. If so, explicitly declare your dependencies.
3. If not, you're probably in a browser so you don't need to do anything else.

Example code from Physics.js:

```
//===============================================================================

// setup for server-side testing
//===============================================================================

if (typeof require === 'function') // test for nodejs environment
{
  var THREE = require('three');
  var MY3 = require('./MY3.js');
}
```