## First Steps

**Creating A Project**

First you'll need to set up where you want the project to be located.

1. Using the text editor of your choice, open up your project folder. I will be using Visual Studio Code for this series.
2. Create a folder where you'll be storing your project files
3. Set up the file folder structure to be something like this:
   * dist
   * public
   * src
4. Main files
   * Go into your src folder and create an app.ts file
   * Go into your public folder and create an index.html file
   * Your html file should look something like:

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Title of Your Project</title>

</head>

<body>

</body>

</html>

Notice that we currently don't have anything within the body of our html file. We'll be generating a canvas within our app.ts file later on.

**Installing Babylon.js**

1. Generate the package.json

npm -y init

You can fill these out now or just keep pressing *Enter* and fill these out in your package.json later. 2. Open up the terminal (in VSCode you can do this by going to the top bar > Terminal > New Terminal).

1. Install Babylon.js

npm install --save-dev @babylonjs/core

This will install all of the dependencies for babylon that you'll need.

1. Typescript Support

tsc --init

This will create a default tsconfig.json. You can replace the contents with:

{

"compilerOptions": {

"target": "es6",

"module": "commonjs",

"noResolve": false,

"noImplicitAny": false,

"sourceMap": true,

"preserveConstEnums":true,

"lib": [

"dom",

"es6"

],

"rootDir": "src"

}

}

**Setting Up Webpack**

**Installing Dependencies**  
Now that we have our package.json generated, we need to install the dev dependencies for using webpack.

npm install --save-dev typescript webpack ts-loader webpack-cli

(if this line doesn't work the first time, try one more time) **Configure Webpack**  
Now we will need to configure webpack to know what to do. Create a webpack.config.js file within your root directory. This is an example of what my configuration looks like:

const path = require("path");

const fs = require("fs");

const appDirectory = fs.realpathSync(process.cwd());

module.exports = {

entry: path.resolve(appDirectory, "src/app.ts"), //path to the main .ts file

output: {

filename: "js/bundleName.js", //name for the javascript file that is created/compiled in memory

},

resolve: {

extensions: [".tsx", ".ts", ".js"],

},

module: {

rules: [

{

test: /\.tsx?$/,

use: "ts-loader",

exclude: /node\_modules/,

},

],

},

mode: "development",

};

**Plugins**  
Additionally, we will install some plugins that will help with updating when running locally, and cleaning our bundle.

npm install --save-dev html-webpack-plugin

npm install --save-dev clean-webpack-plugin

npm install --save-dev webpack-dev-server

After we have these installed, we'll need to add them to our webpack.config.js. Our updated webpack.config.js should look something like this now:

const path = require("path");

const fs = require("fs");

const HtmlWebpackPlugin = require("html-webpack-plugin");

const { CleanWebpackPlugin } = require("clean-webpack-plugin");

const appDirectory = fs.realpathSync(process.cwd());

module.exports = {

entry: path.resolve(appDirectory, "src/app.ts"), //path to the main .ts file

output: {

filename: "js/bundleName.js", //name for the js file that is created/compiled in memory

},

resolve: {

extensions: [".tsx", ".ts", ".js"],

},

devServer: {

host: "0.0.0.0",

port: 8080, //port that we're using for local host (localhost:8080)

disableHostCheck: true,

contentBase: path.resolve(appDirectory, "public"), //tells webpack to serve from the public folder

publicPath: "/",

hot: true,

},

module: {

rules: [

{

test: /\.tsx?$/,

use: "ts-loader",

exclude: /node\_modules/,

},

],

},

plugins: [

new HtmlWebpackPlugin({

inject: true,

template: path.resolve(appDirectory, "public/index.html"),

}),

new CleanWebpackPlugin(),

],

mode: "development",

};

Now when we build and run our project, if we make any updates, the browser will refresh so that we can see our changes. In addition, the HTML Webpack plugin is taking the javascript bundle file that is compiled and injects it into our index.html file. This .js bundle will show up inside of the dist folder.

**Creating A Scene**

We'll be setting up the app.ts file to be the main entry point for our project.

**Set up and create the App class**

The App class will serve as our entire game application. This is a very simple example of how to set up a scene and should be separated out into different functions and make use of class variables for your project as you progress.

class App {

constructor() {

// create the canvas html element and attach it to the webpage

var canvas = document.createElement("canvas");

canvas.style.width = "100%";

canvas.style.height = "100%";

canvas.id = "gameCanvas";

document.body.appendChild(canvas);

// initialize babylon scene and engine

var engine = new Engine(canvas, true);

var scene = new Scene(engine);

var camera: ArcRotateCamera = new ArcRotateCamera("Camera", Math.PI / 2, Math.PI / 2, 2, Vector3.Zero(), scene);

camera.attachControl(canvas, true);

var light1: HemisphericLight = new HemisphericLight("light1", new Vector3(1, 1, 0), scene);

var sphere: Mesh = MeshBuilder.CreateSphere("sphere", { diameter: 1 }, scene);

// hide/show the Inspector

window.addEventListener("keydown", (ev) => {

// Shift+Ctrl+Alt+I

if (ev.shiftKey && ev.ctrlKey && ev.altKey && ev.keyCode === 73) {

if (scene.debugLayer.isVisible()) {

scene.debugLayer.hide();

} else {

scene.debugLayer.show();

}

}

});

// run the main render loop

engine.runRenderLoop(() => {

scene.render();

});

}

}

new App();

At this point, you should be seeing a lot of errors. This is because we haven't imported the babylonjs dependencies. Add to the top of your file:

import "@babylonjs/core/Debug/debugLayer";

import "@babylonjs/inspector";

import "@babylonjs/loaders/glTF";

import { Engine, Scene, ArcRotateCamera, Vector3, HemisphericLight, Mesh, MeshBuilder } from "@babylonjs/core";

**Bundling the Project and Running Locally**

Now that we have our project set up, how do we run it locally? We need to set up tasks in our package.json. In your package.json replace your "scripts" with this:

"scripts": {

"build": "webpack",

"start": "webpack serve --mode development --open 'Chrome'"

},

Now all we need to do is:

npm run build

npm run start

Then when you visit http://0.0.0.0:8080 in our browser, you should see a sphere!