Computer Graphics 1

1 Introduction

Summer Semester 2021

Ludwig-Maximilians-Universität München

Welcome!

Tutorial 1: Introduction

- Initial Setups
  - Git and GitHub
  - Code Editing and Markdown
- Basics of Modern JavaScript
- TypeScript, Node.js and Its Ecosystem
- 3D in WebGL and three.js
- Summary

Minimum Environment

The minimum environment setup for the CG1 tutorials:

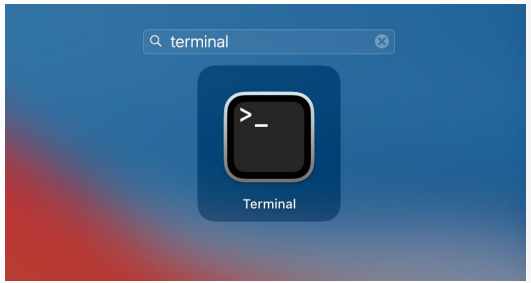
- git: https://git-scm.com/
- Node.js: https://nodejs.org/en/
- Visual Studio Code (recommended): https://code.visualstudio.com/

We will explain them one by one.

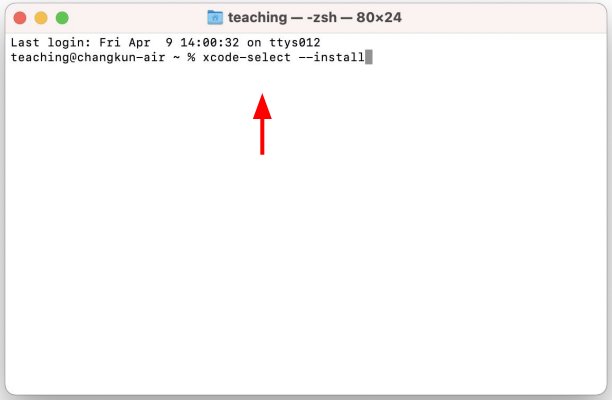
**Linux users should be able to setup the above environment without instructions.*

Install Git (for macOS users)

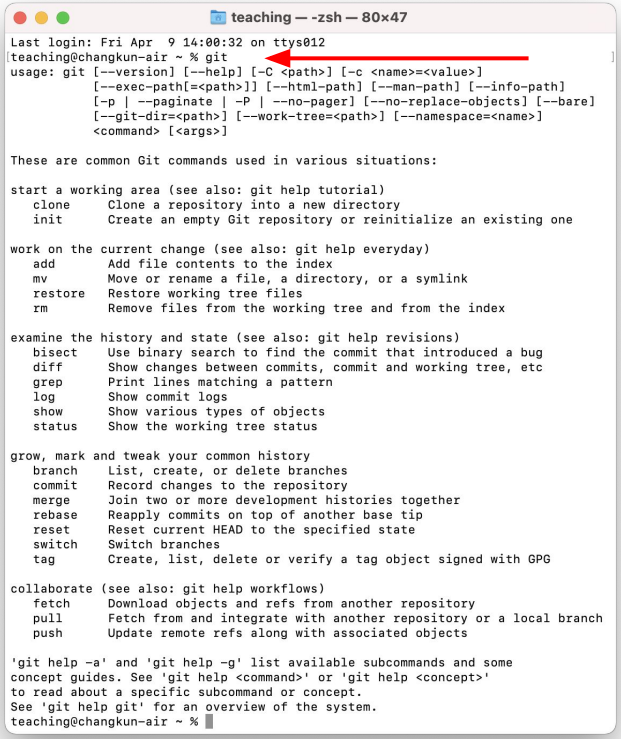
- Find terminal in the system (though Launchpad or Spotlight search)
- Verify if the **`git`** command is already available in the system
- If not, type **`xcode-select --install`** and follow the prompts, confirm and wait until installation is finished
- (Again) Verify if the **`git`** command is available from the terminal



Find terminal



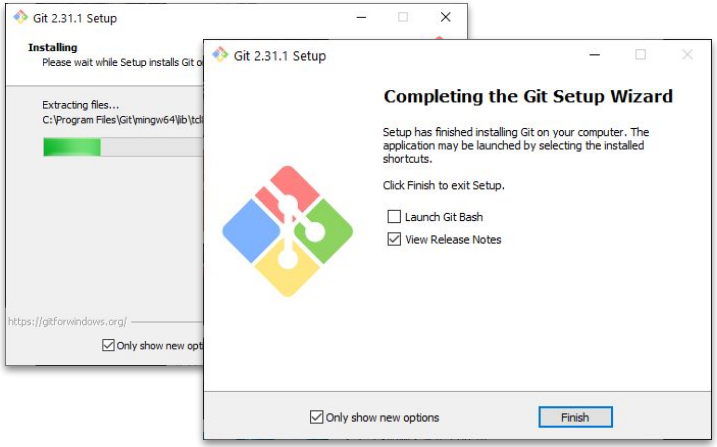
Install development environment on macOS



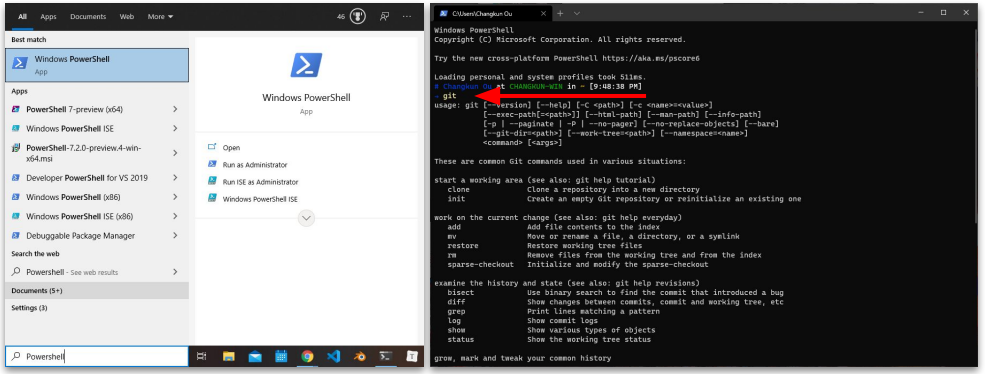
Verify availability of git

Install Git (for Window 10 users)

- Find a terminal (PowerShell) in the system
- Verify if the **`git`** command is already available in the system
- If not, download and install git, confirm and wait until installation is finished
- (Again) Verify if the **`git`** command is available from the terminal



Install Git on Windows

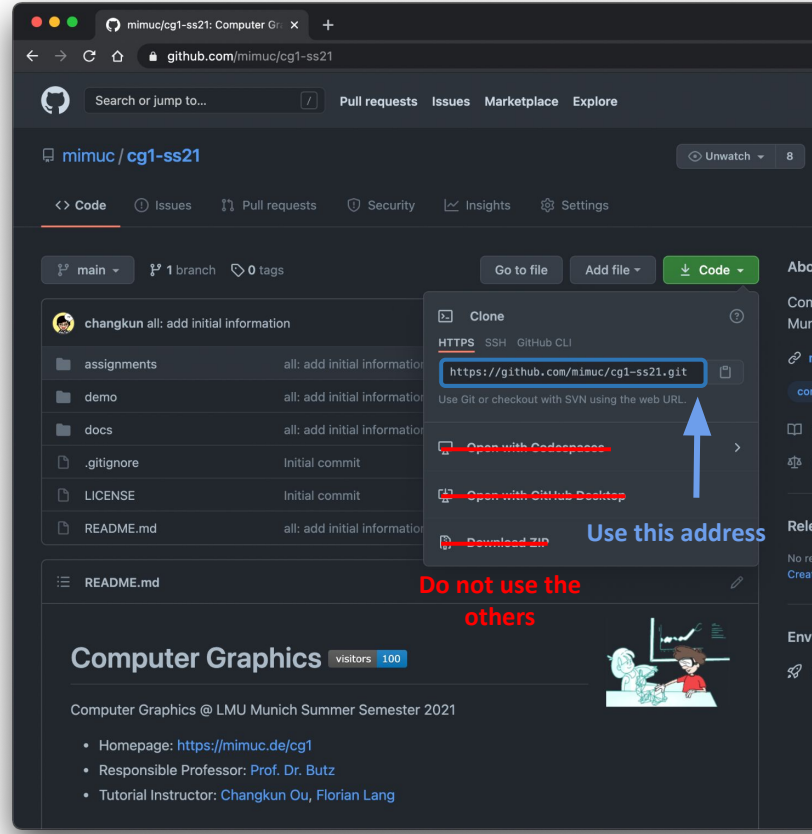


Verify availability of git

Clone from GitHub

The first thing to get started is to download our GitHub repository. Use git to clone the repository in the terminal:

```
$ git clone https://github.com/mimuc/cg1.git
Cloning into 'cg1'...
remote: Enumerating objects: 25, done.
remote: Counting objects: 100% (25/25), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 25 (delta 6), reused 18 (delta 3),
pack-reused 0
Receiving objects: 100% (25/25), 54.13 KiB | 701.00 KiB/s,
done.
Resolving deltas: 100% (6/6), done.
```



Install Visual Studio Code (VSCode)

Download VSCode and install it to the system

Add code command to the terminal

In the terminal

- windows user:

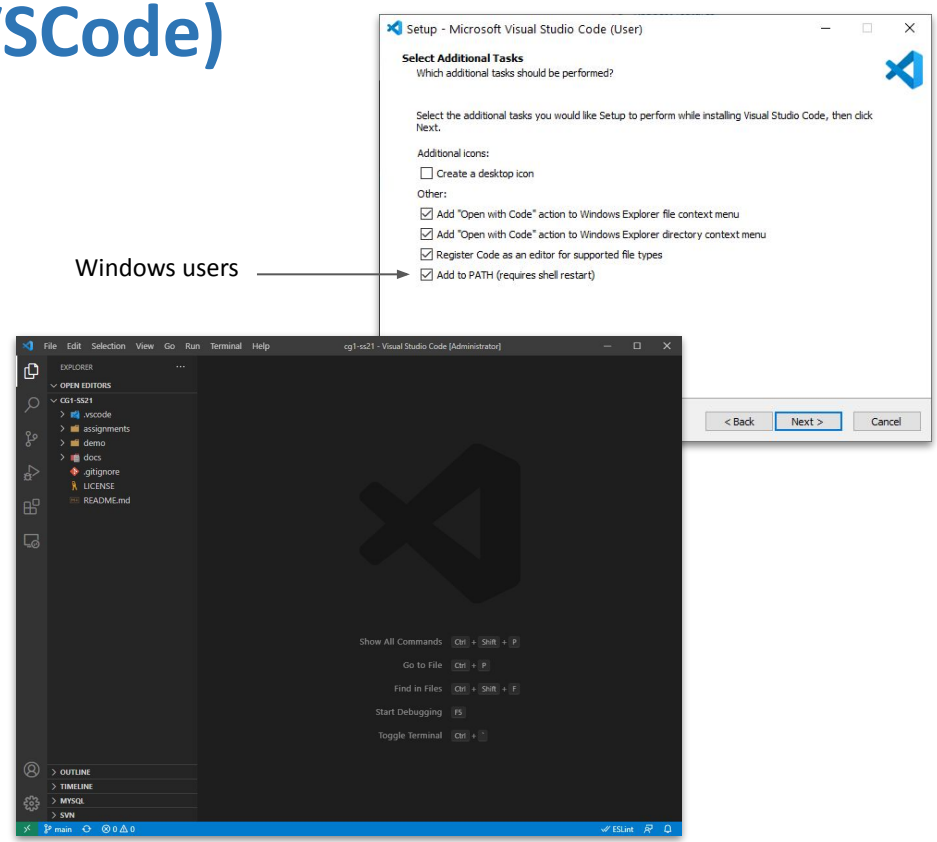
  ```
  $ code .\cg1-ss21\
  ```

- macOS user:

  ```
  $ code cg1-ss21
  ```

Then the entire workspace should be opened

Windows users



Other code editor alternatives:

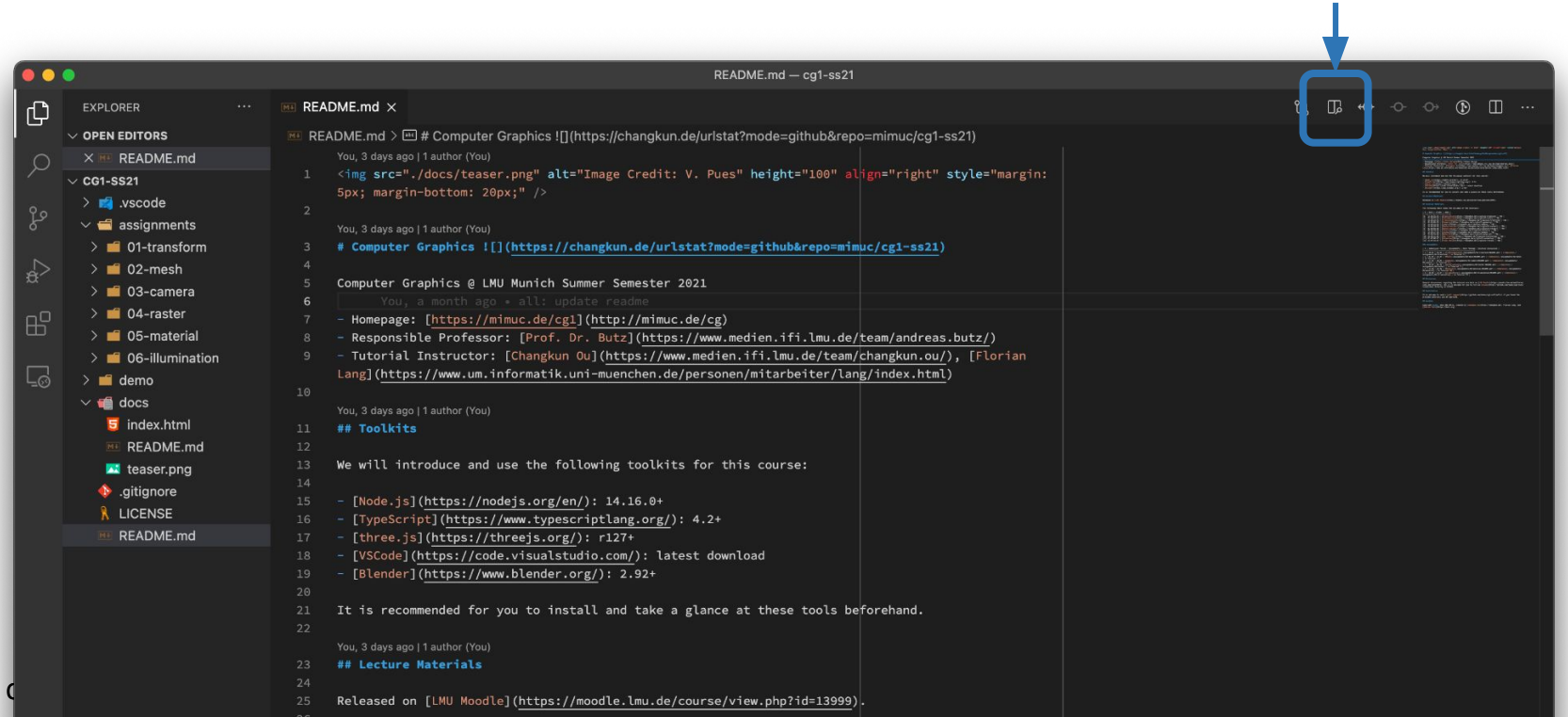
- Vim
- WebStorm

*VSCode configurations are already included and activated in the workspace configuration.

Markdown

Markdown is a lightweight markup ~~language~~ for creating formatted text, and widely used.

VSCode can rendering a Markdown file (.md), side by side:



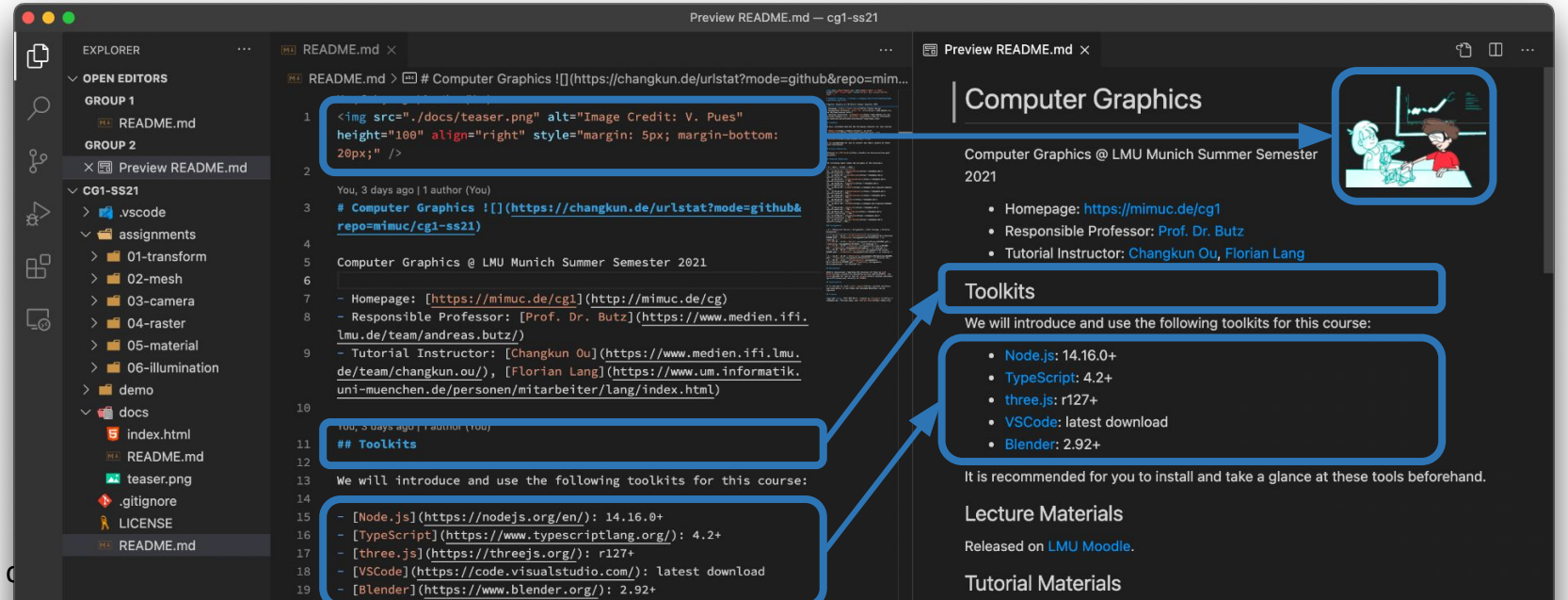
Markdown

Markdown is a lightweight markup ~~language~~ for creating formatted text, and widely used.

VSCode can rendering a Markdown file (.md), side by side

⇒ It is unnecessary to learn it systematically, instead: use it by looking at the provided examples in the README.md file

Install a plugin for displaying mathematics formula: Markdown All in One.

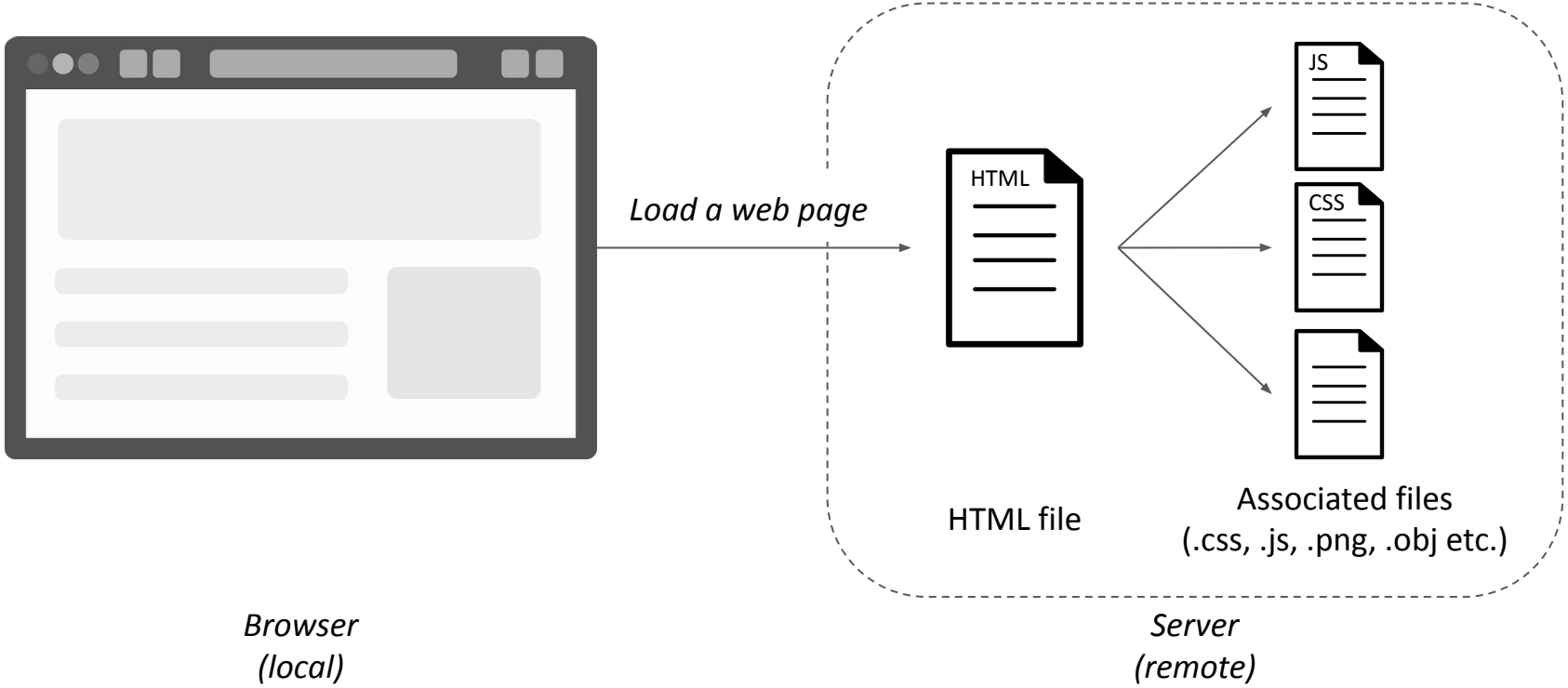


Tutorial 1: Introduction

- Initial Setups
- Basics of Modern JavaScript
  - Browser execution environment
  - Programming building blocks
- TypeScript, Node.js and Its Ecosystem
- 3D in WebGL and three.js
- Summary

JavaScript

JavaScript is a programming language that is originally designed for manipulating elements on a web page



Open JavaScript Console (in Chrome)

Keyboard shortcuts for opening a JS console:

- macOS: alt+command+J
- Windows: Ctrl+Shift+J

(Same shortcut in Firefox)

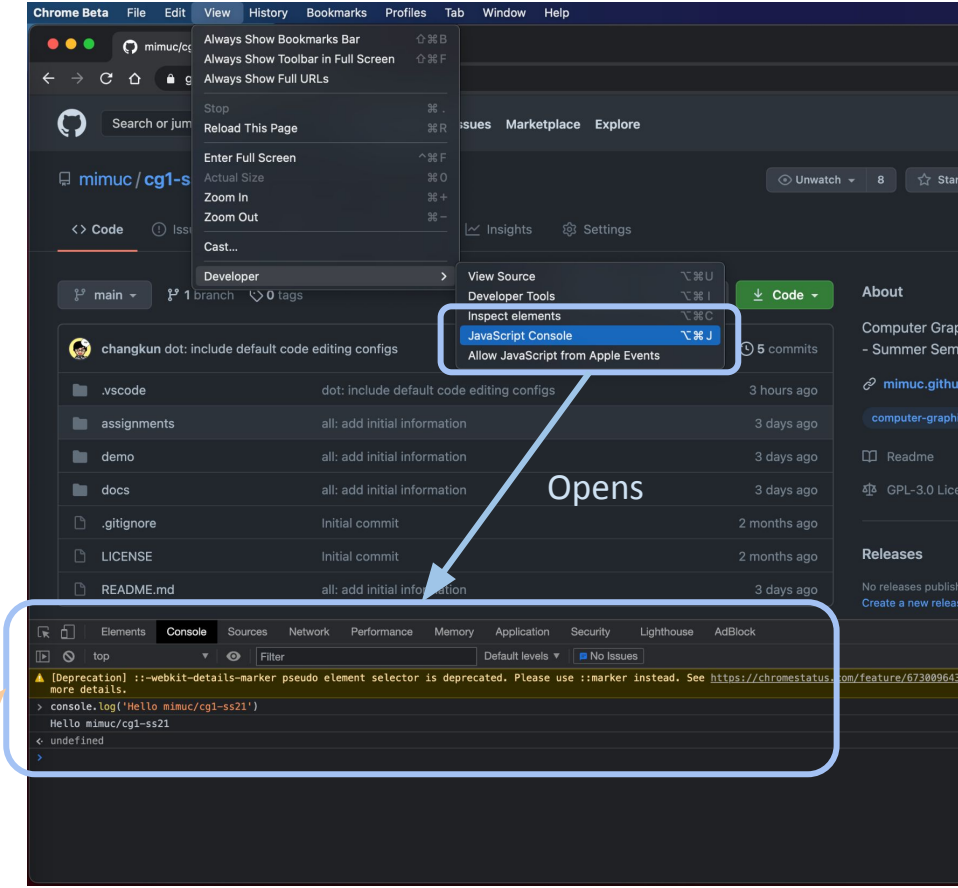
Try to type:

```
console.log('Hello mimuc/cg1-ss21')
```

See if the same result can be reproduced.

The console is helpful for debugging and we can see what went wrong.

Write JavaScript code here



Some Browsers act differently. If you get an error, try a different browser first.

Basic JavaScript Concepts

- ***constant***: immutable data

```
const c = 3.14; // cannot be changed later
```

- ***variable***: mutable data

```
let v = 0;      // can be changed later
```

- ***function***: a code block maps a list of parameters to a list of return values

```
function Foo(p1, p2, p3) { … }    (normal function)
const Bar = (p1, p2, p3) => { … } (arrow  function)
```

Q: What are the differences?

- ***flow control***: `if/else/switch/for` statements (in almost every-language)

```
const a = 1;
const b = 2;
if (a > b) {
   console.log(a);
} else {
   console.log(b); // prints
}
```

```
let difficulty = 'myth';
switch (difficulty) {
   case 'easy':
       console.log('CG1 is very easy');
       break;
   default:
       console.log('I do not know!');
       break;
}
```

```
for (let i = 0; i < 10; i++) {
   console.log(`CG${i}`);
}
// Prints:
// CG0
// CG1
// …
// CG9
```

Data Types

- ***boolean:*** `true | false`
- ***number:*** `3.1415`
- ***string***: `'Hello CG1!'`
- ***array***: `[1, 2, 3, 4]`
- ***object***: `{course: 'MIMUC/CG1', year: 2021, difficulty: 'ultra-easy'}`
- ***Special values in JavaScript:*** `null, undefined` ***(Advise: just use*** `null`***)***

```
let unknown = null;
```

- ***Type inspection:*** `typeof`

```
typeof(3.1415) // "number"
typeof(true)   // "boolean"
typeof('CG1')  // "string"
```

Error Handling

A error can stop JavaScript from execution. Handling errors can prevent the interruption.

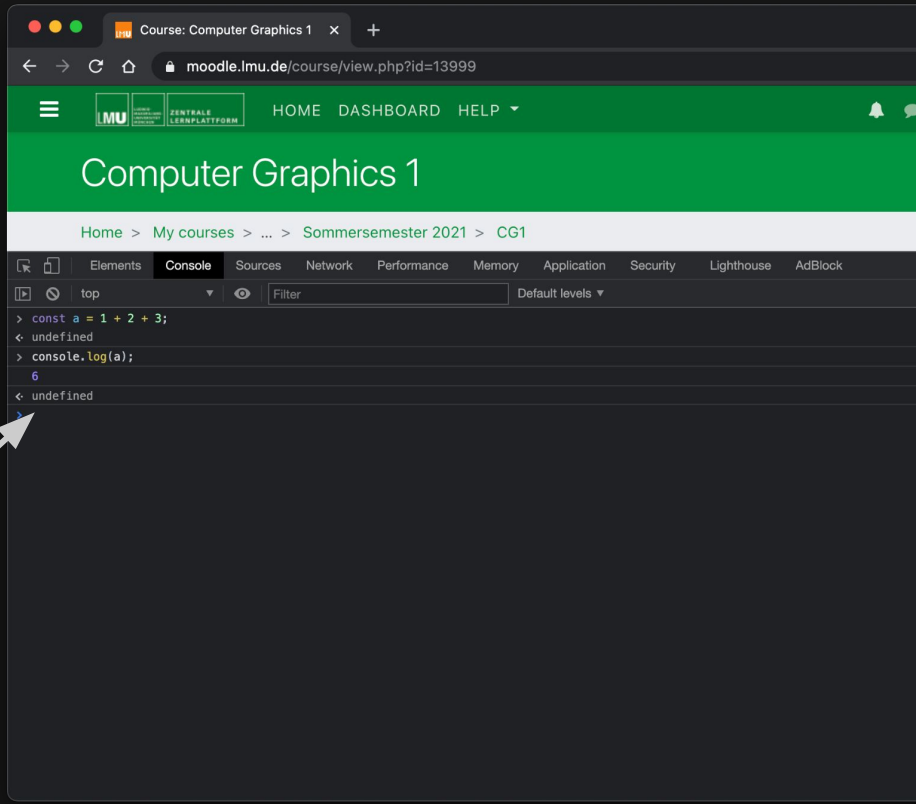
- Try block: the main code for execution
- Catch block: be executed when there is an error
- Finally block: executes always

Example:

```
try {
   console.log('works');      // prints 'works'
   throw new Error('throw an error!');
   console.log('not work');   // will not be printed
} catch(err) {
   console.log(err);          // prints throwed value: "throw an error"
} finally {
   console.log(always work'); // always prints 'always work'
}
```

Breakout: Try JavaScript in A Browser

- Open the JavaScript console
- Write some code using these concepts that we just covered:
  - variable
  - constant
  - function
  - flow control
  - data types
  - error handling
- Search and explain why there is an "undefined" output in the console using Google.



Downside of JavaScript

Consider a naive function: `function foo(a, b) { … }`

This can be confusing!

As a user/caller:

- What is the type of the parameters I am supposed to pass to the function?
- What does this function return to me?
- What will happen if I do not provide the expected parameters?
- ...

As an implementer:

- How will the caller use this function?
- How should I handle invalid inputs? Throw an error? Return a default result? Something else?
- How should I document the behavior of the function? What if I don't?
- ...

Tutorial 1: Introduction

- Initial Setups
- Basics of Modern JavaScript
- TypeScript, Node.js and Its Ecosystem
- WebGL and three.js
- Summary

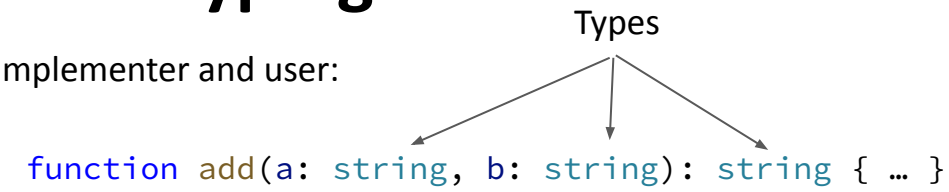
TypeScript

TypeScript is a superset of JavaScript, or *a statically typed JavaScript*

- It cannot be executed directly in a browser but will be compiled to JavaScript, automatically
- (Almost) everything that works in JavaScript also works in TypeScript
- But: TypeScript forces us to think about the type ⇒ increases readability and debuggability
- Key supplies beyond JavaScript:
  - Static and strong typing
  - Interface
  - etc.

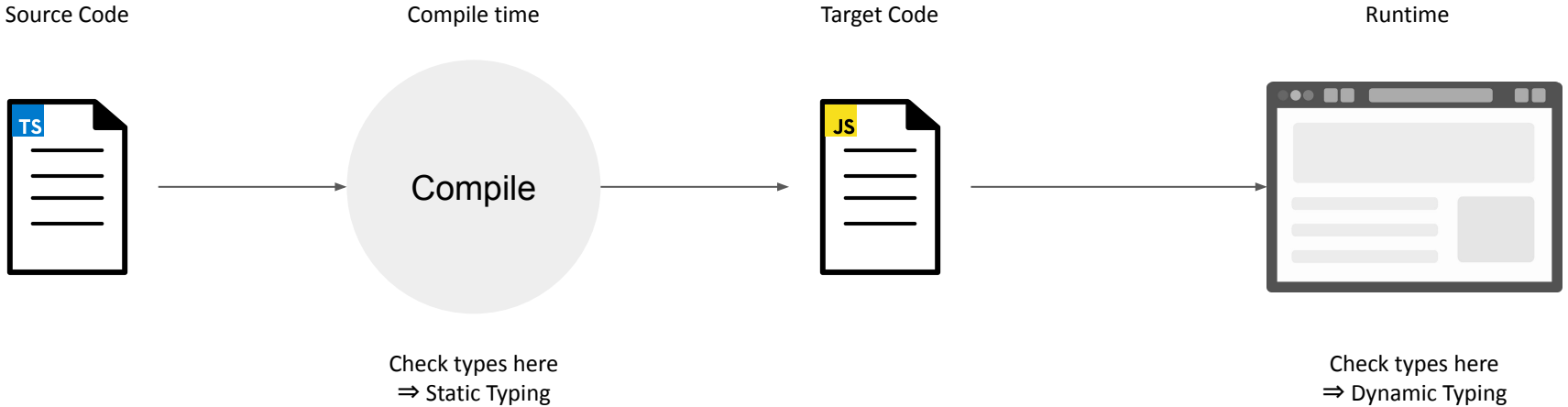
Dynamic vs. Static Typing

This is much better for both implementer and user:



The definition of the function already tells a lot of information:

It adds two strings and returns the resulting string



Weak vs. Strong Typing

- There is no precise technical definition regarding weak, loosely or strong typing
- We commonly agree that JavaScript is a *weakly typed* language, and TypeScript is strongly typed
- In JavaScript:

```
console.log(4 + '2') // 42
console.log(4 * '2') // 8
```

- In TypeScript:

```
console.log(4 + '2') // not allowed
console.log(4 * '2') // not allowed
```

- Weakly typed languages are convenient when coding but can lead to unexpected behavior:

```
console.log(1 === '1') // ???
console.log(1 == '1')  // ???
```

  We never know the answer until we run it

- Advise: use TypeScript to add more restrictions but bring a lot more convenience for the future

**Aside: Java is a strongly typed language*

Class in TypeScript

class: a special "function" with a `constructor()` which is auto-executed when a new object of that class is created

```
class MyClass {
 p1: string;
 p2: number;
 pn: object[];
 constructor(p1: string, p2: number, ...pn: object[]) {
   this.p1 = p1;
   this.p2 = p2;
   this.pn = [...pn];
 }
 f() {
   console.log(this.p1, this.p2, this.pn);
 }
}
const m = new MyClass('1', 2, 1, 2);
m.f(); // '1', 2, [1, 2]
```

Interface in TypeScript

Interface: declares a shape of an object:

```
interface User {
 name: string;
 id: number;
}
class UserAccount {
 name: string;
 id: number;
 constructor(name: string, id: number) {
   this.name = name;
   this.id = id;
 }
}
const user: User = new UserAccount('Murphy', 1);
console.log(user); // UserAccount {name: "Murphy", id: 1}
```

Data Types in TypeScript

Declare types:

```
const b: boolean = false;
const n: number  = 3.1415;
const s: string  = 'Hello CG1';
const a: Array<number> = [1, 2, 3, 4];
const o: object = {course: 'mimuc/cg1', year: 2021, difficulty: 'ultra-easy'};
```

With the function `typeof`***, a parameter can accept multiple different types of arguments:***

```
/**
 * elements counts the number of elements of a given argument.
 * @param s is either a string or an array of string
 * @returns the number of character elements of the given string or array.
 */
function elements(s: string | string[]) {
 if (typeof s === 'string') {
   return s.length;
 }
 let sum = 0;
 for (let i = 0; i < s.length; i++) {
   sum += s[i].length;
 }
 return sum;
}
```

Core Concepts in (Almost) Every Programming Language

- Constant
- Variable
- Function
- Flow control
- Class and interface
- Types
- Error handling

That's it. That's all we need to know about using JavaScript/TypeScript (Just 1% of the whole language)

for graphics programming in CG1. The remaining key question is:

What do we need to use TypeScript?

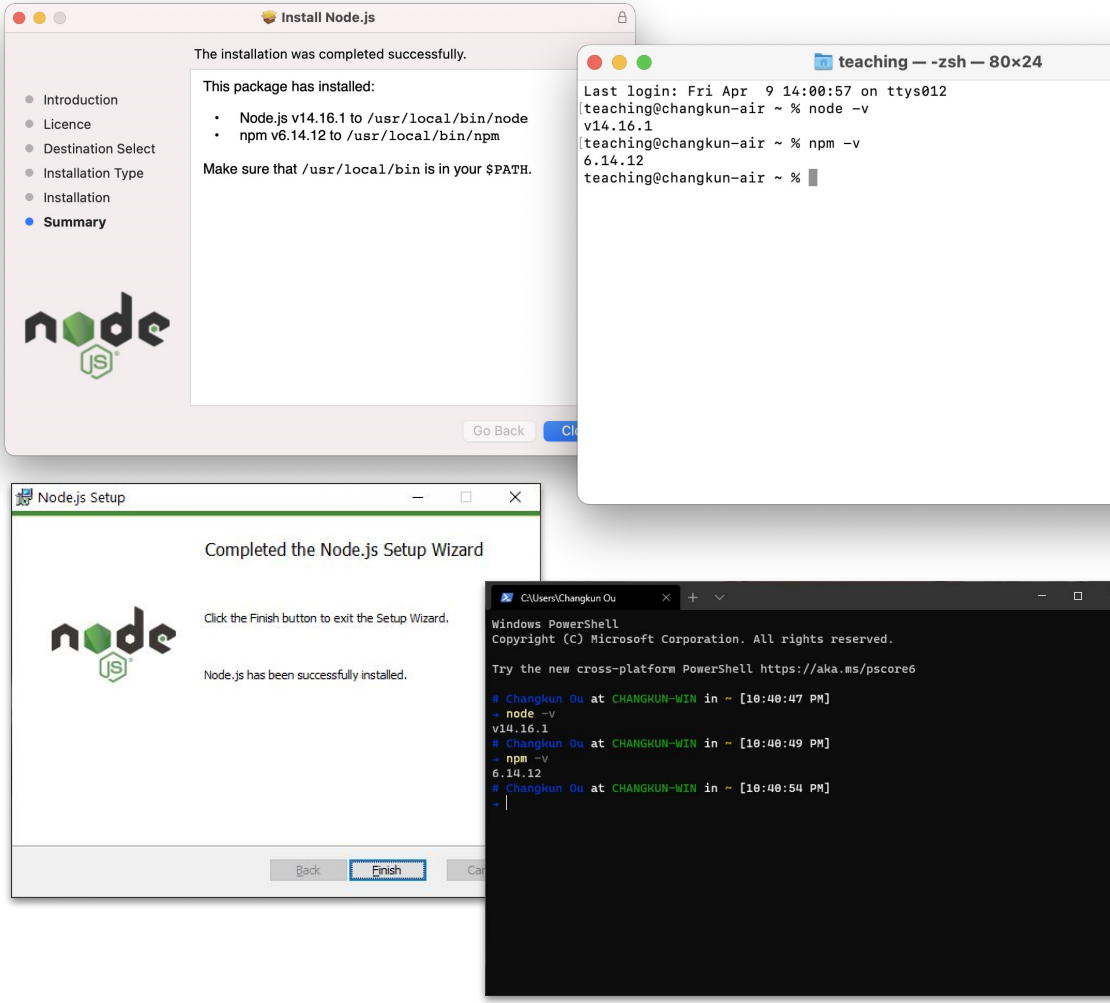
Install Node.js

Download and install the stable version:

https://nodejs.org/en/

After the installation, verify if the following two commands are available from the terminal:

```
$ node -v
v14.16.1
$ npm -v
v6.14.12
```



NodeJS and NPM

- What are those?
  - JavaScript is a (standardized) language, and Node.js is an implementation/runtime.
- We need them to:
  - Generate files automatically and compile TypeScript to JavaScript for execution
  - Start a local server to serve all the files (like a server on the internet)
  - Better coding experience, e.g., automatically refresh the page when the code has changed
  - Better engineering practices, e.g., dependency management
  - …

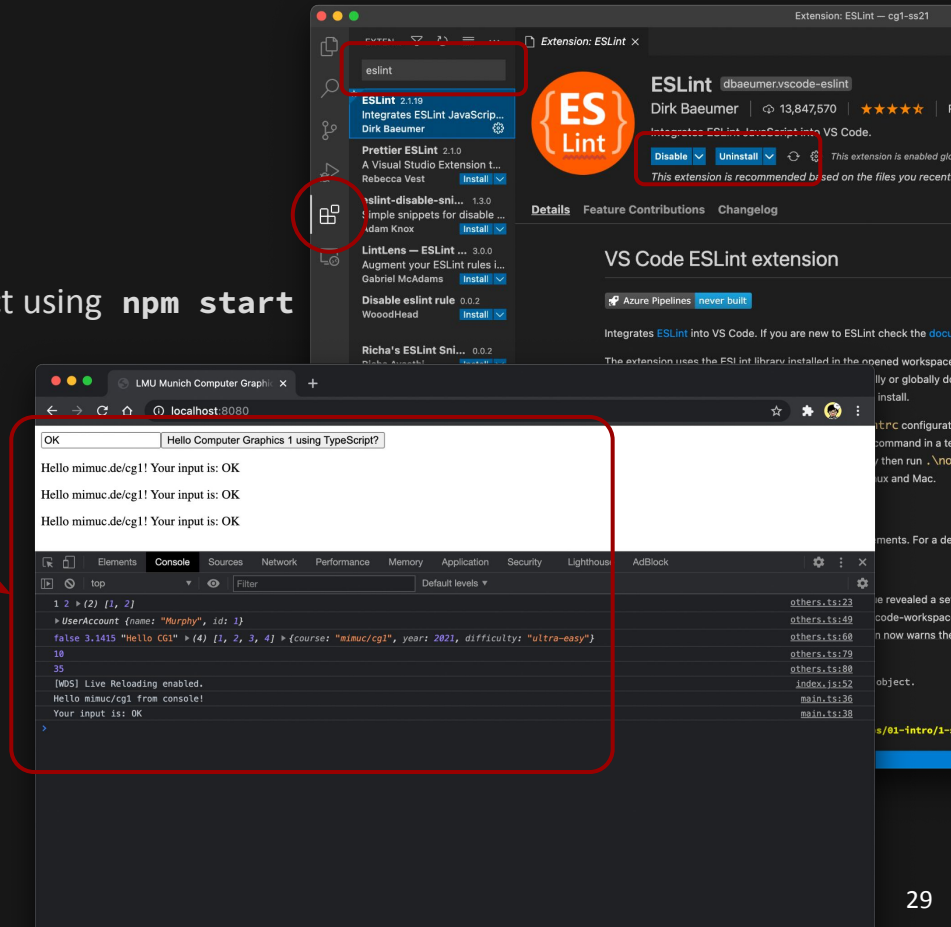
- NPM manages declared dependencies in **`package.json`**, and saves dependencies in **`node_modules`** folder.
- Basic usage:

`$ npm i`	install everything we need for coding (required inside the folder with file `package.json`)
`$ npm start`	start command of a project (in the CG1 provided code skeletons, it also compiles TypeScript)

Breakout: Setup (Enhanced) Coding Environment

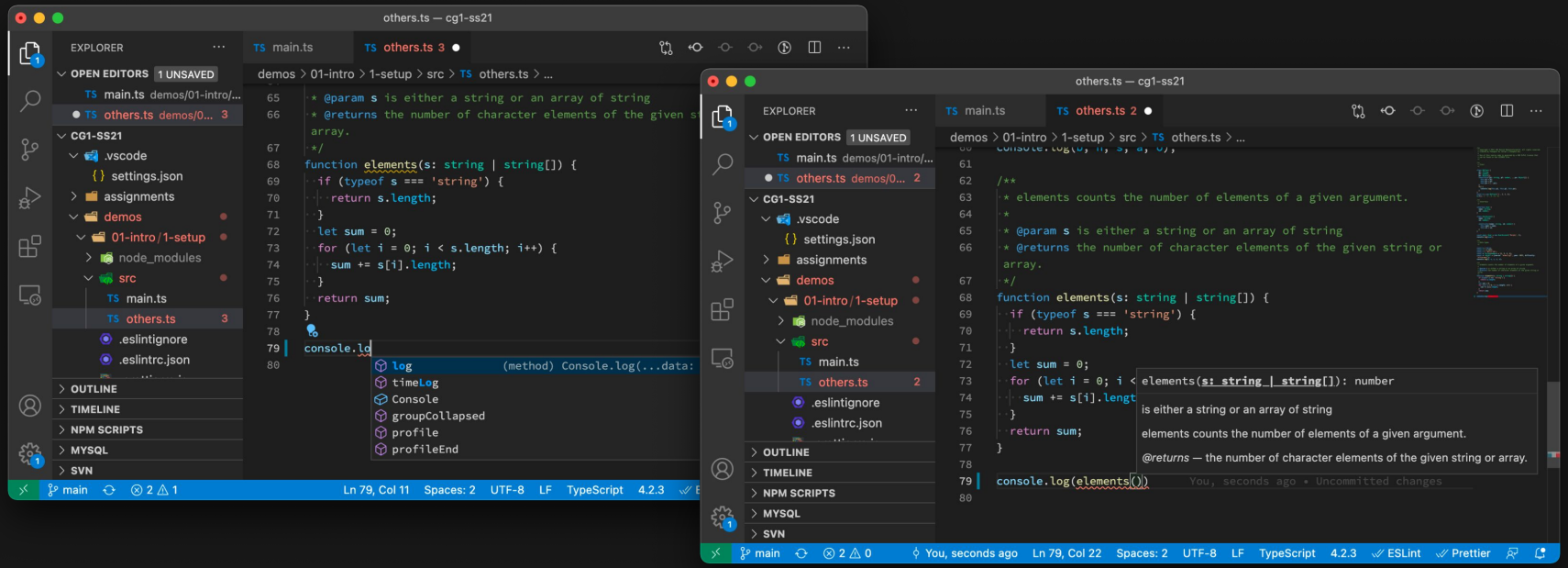
- **Install ESLint plugin and activate it in VSCode**
- Navigate to folder
  - **`Windows: cd demos\01-intro\1-setup`**
  - **`macOS:   cd demos/01-intro/1-setup`**
- Install all dependencies using **`npm i`** and run the project using **`npm start`**
- Double check if these features works:
  - AutoCompletion
  - AutoFix on save
  - See if the provided code gets executed
- Read the code and comments
- Answer the question (with help from Google):
  - What does this mean? In `src/main.ts`, line 9:

    ```
    import './others';
    ```



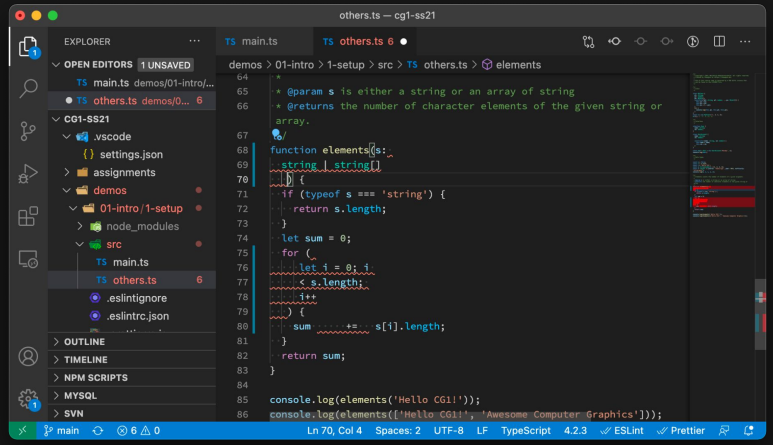
Feature: AutoCompletion

- Prompts regarding variables, function, class methods, etc
- Show documents about how to use a function/method etc

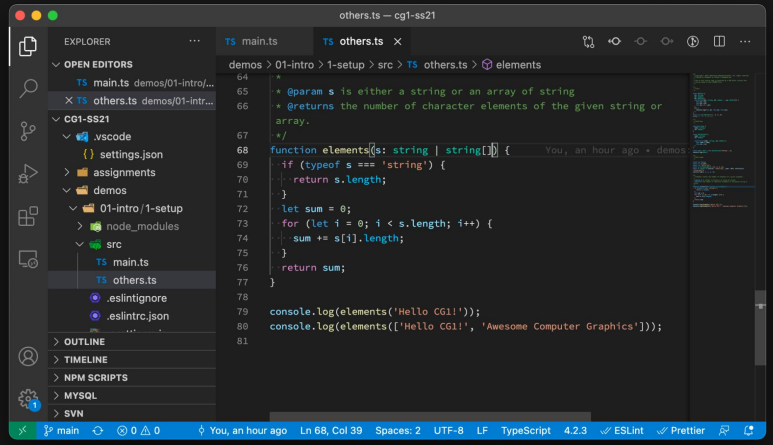


Feature: AutoFix

- Fix code format automatically when we save the file (ctrl+s or cmd+s)
- We use widely accepted Google TypeScript Style Guide
- With this feature enabled, we don't need worry anything about how we should place spaces, tabs, or others. **JUST SAVE THE CODE** and the tool will do everything else for us



Fix Code Format



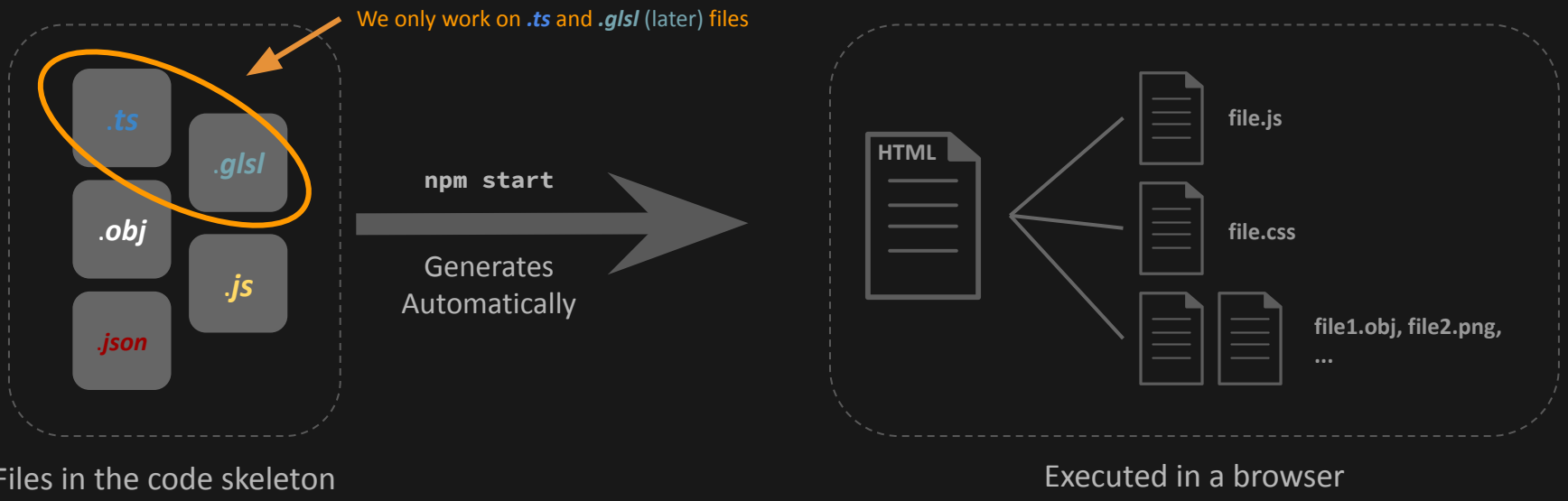
Aside: How does the code skeleton work?

Traditionally, we need to work on *.html*, *.css*, and *.js* files together, and resolve every detail separately.

In modern world (2021), front-end web development had moved on from that and switched to pure JavaScript based development, or even better - TypeScript based development: Write everything in TypeScript, and use tools to organize and generate all other files automatically.

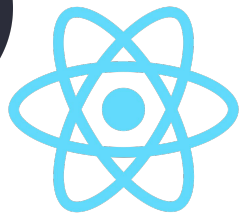
Our code skeleton can generates/transpiles all files (using *Webpack*) for us, then serves it in a browser.

**Conveniently: refresh the page automatically whenever we made changes to the code so we don't have to refresh it by hand*



Ecosystem

- JavaScript and TypeScript are not limited to manipulate elements on a web page, they also can:
  - Create desktop application using Electron
  - Create mobile application using React Native
  - Create modern progressive web application using React or Vue
  - Create backend server using Express (although we have better choices :)
  - … even create immersive XR application with WebXR
  - Check out W3C standards and see more possibilities there
- But those would be another story beyond CG1 😁
- We will focus on WebGL and the library *three.js* build on top of it


Tutorial 1: Introduction

- Initial Setups
- Basics of Modern JavaScript
- TypeScript, Node.js and Its Ecosystem
- WebGL and three.js
- Summary

WebGL

WebGL is a web standard based on OpenGL ES (and OpenGL is a legacy graphics standard) for creating 2D and 3D graphics on the web

Check whether a browser supports WebGL: https://webglreport.com/

Documents: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API

But, we are still not going to discuss WebGL directly from Day 1 because:

- The API design is too complicated for "getting started" purpose
- APIs are too difficulty to use, and breaks in between WebGL and WebGL2
- There is already a fantastic library which reduces the complexity of using WebGL
- ...

Is there a better choice?

three.js: A JavaScript 3D Library

three.js r127

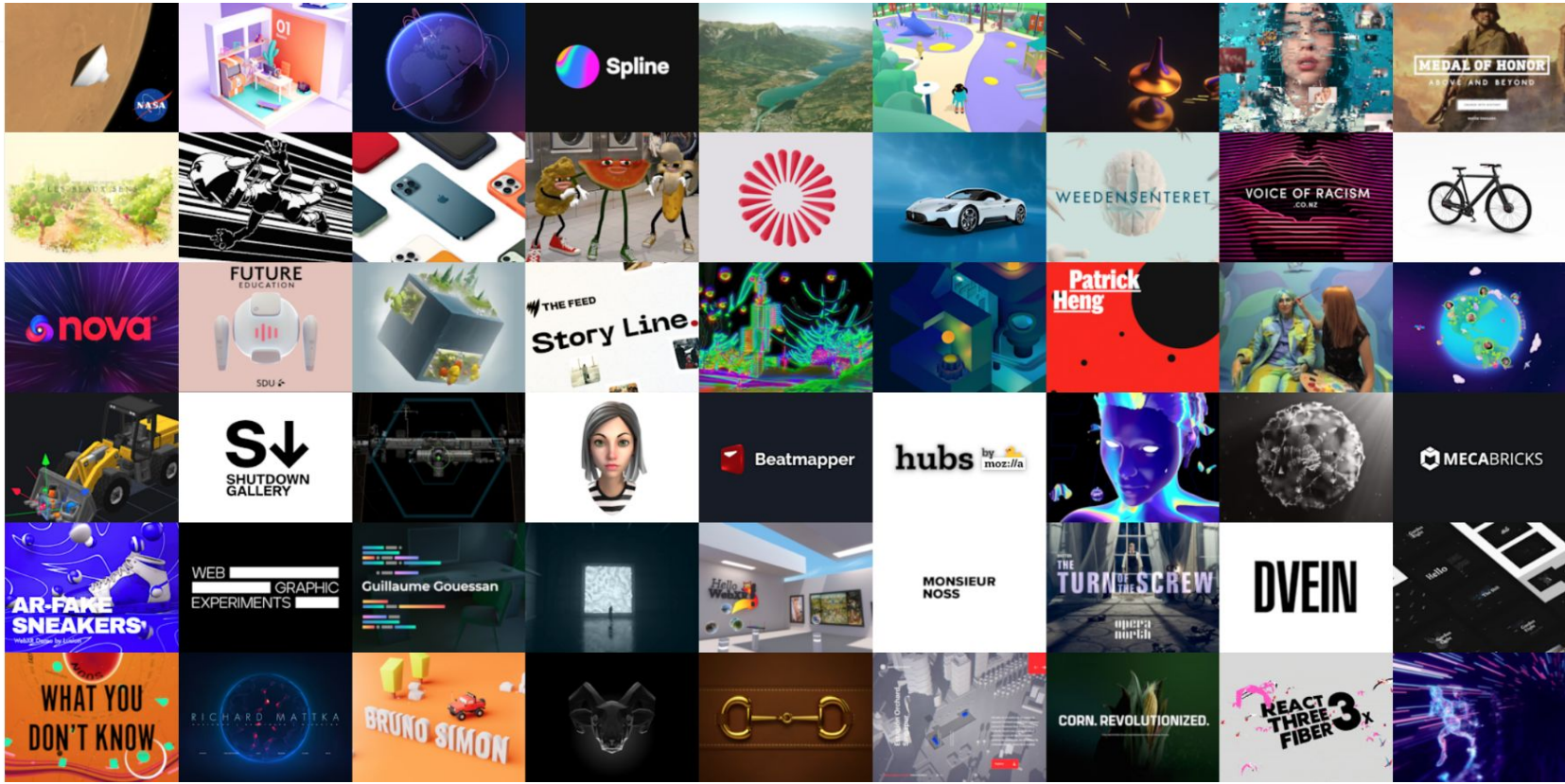
Learn
documentation
examples
editor

Community
questions
discord
forum
slack
twitter

Code
github
download
devtools

Resources
Three.js Fundamentals
Three.js Journey
Learn Three.js
初めてのThree.js

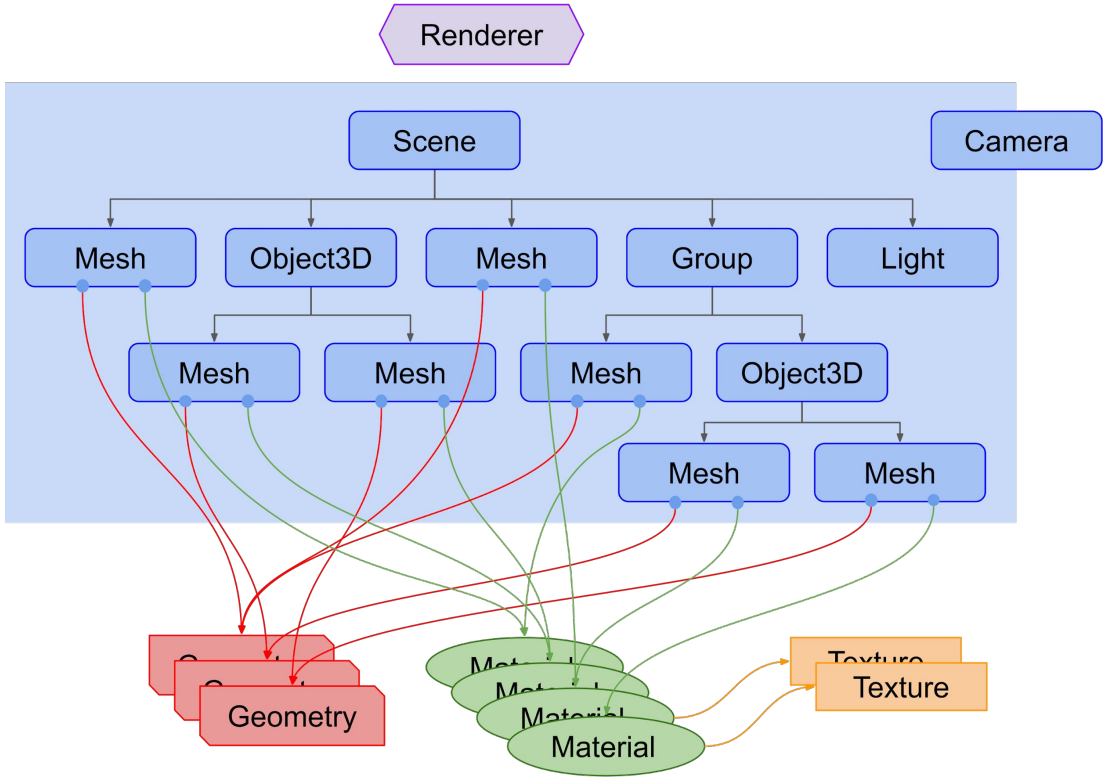
Merch
T-Shirts



three.js

The core concepts in three.js are:

- Renderer
- Scene
- Camera
- Mesh
- Geometry
- Material
- Light
- …



Renderer and Scene

A ***Renderer*** is created by `WebGLRenderer`, and renders a scene in the browser.

HTML manages everything using *DOM tree*. Therefore it is necessary to add a renderer DOM element to the `document.body`.

(Don't pay too much attention to how DOM works) Q: What happens if we don't pass the `{antialias: true}` to the renderer? Try it out in the breakout.

```
constructor() {
  const container = document.body;
  ...

  // Create renderer and add to the container
  this.renderer = new WebGLRenderer({antialias: true});
  this.renderer.setPixelRatio(window.devicePixelRatio);
  container.appendChild(this.renderer.domElement);
  ...
}
```

A ***Scene*** is created by `Scene`, and represents a graph of the whole scene

```
// Create scene
this.scene = new Scene();
```

Camera and OrbitControl

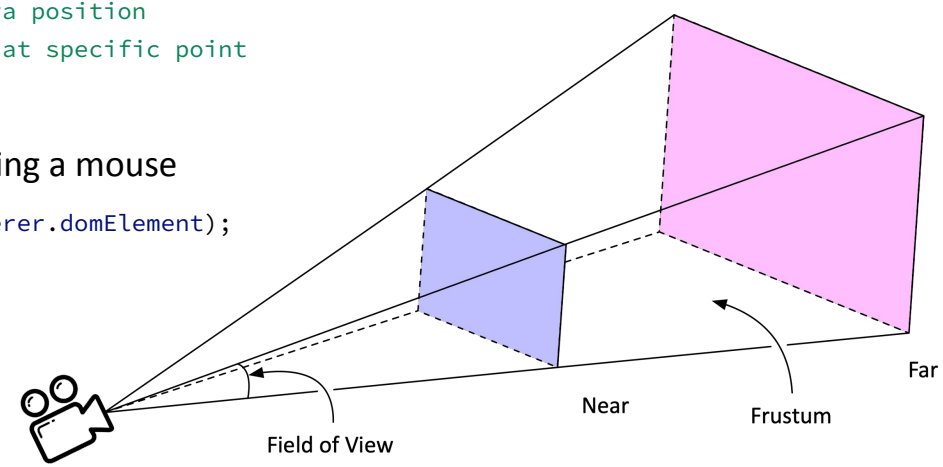
There are many different types of camera, which we will discuss later. The most frequently used camera is

`PerspectiveCamera`

```
this.camera = new PerspectiveCamera(
  cameraParam.fov,    // field of view
  cameraParam.aspect, // screen width / screen height
  cameraParam.near,   // near plane
  cameraParam.far     // far  plane
);
this.camera.position.copy(cameraParam.position); // camera position
this.camera.lookAt(cameraParam.lookAt);          // look at specific point
```

`OrbitControl` allows us to rotate the camera view using a mouse

```
this.controls = new OrbitControls(this.camera, this.renderer.domElement);
```



Animation Frames and Render Loop

An animation is a series of rendered images, `requestAnimationFrame` is a request to the browser that we want animate something.

The animate callback will be executed by the browser if anything is updated (loop occurs).

The render() draws the scene according to the camera's definition.

```
// The render loop
render() {
  this.renderer.setSize(window.innerWidth, window.innerHeight);
  this.controls.update();
  this.renderer.render(this.scene, this.camera);
  window.requestAnimationFrame(() => this.render());
}
```

What we have so far...

See code example from demos/01-intro/2-basic, line 24 - 91:

```
class SimpleWorld {
  // A WebGLRenderer for rendering the world
  renderer: WebGLRenderer;
  ...
  constructor() {
    const container = document.body;
    ...
  }
  // The render loop
  render() {
    ...
    window.requestAnimationFrame(() => this.render());
  }
```

These code are very repetitive and almost the same in every project.

We just need focus on writing **core part** of a project: *managing geometric objects and handle their actions*.

Breakout: Create A Simple 3D Scene (without knowing graphics)

Enter folder `demos/01-intro/2-basic`

- `Windows: cd demos\01-intro\2-basic`
- `macOS: cd demos/01-intro/2-basic`

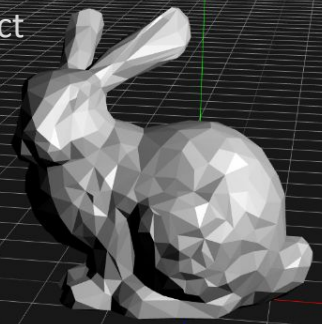
Use `npm i` to install dependencies and `npm start` to run the project

Read the code and understand how the code is executed

Find TODO: comments in the `src/main.ts`

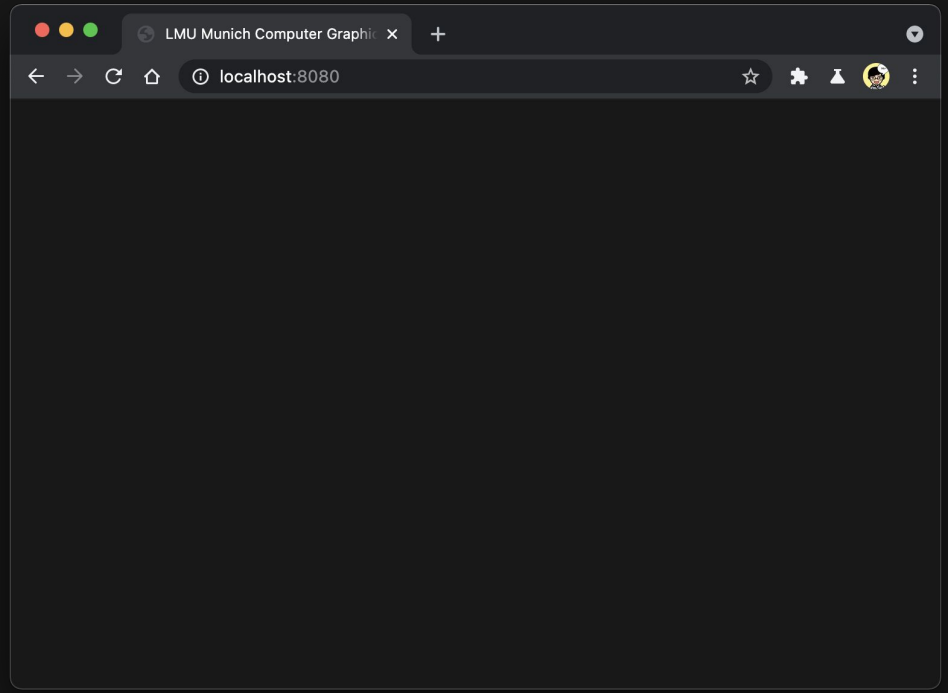
Activate the four pieces (uncomment one by one):

1. Create a `GridPlane`
2. Create an `AxesHelper`
3. Load and render a bunny from `bunny.obj` file
4. Create a `PointLight`



Step 0: Run the Project

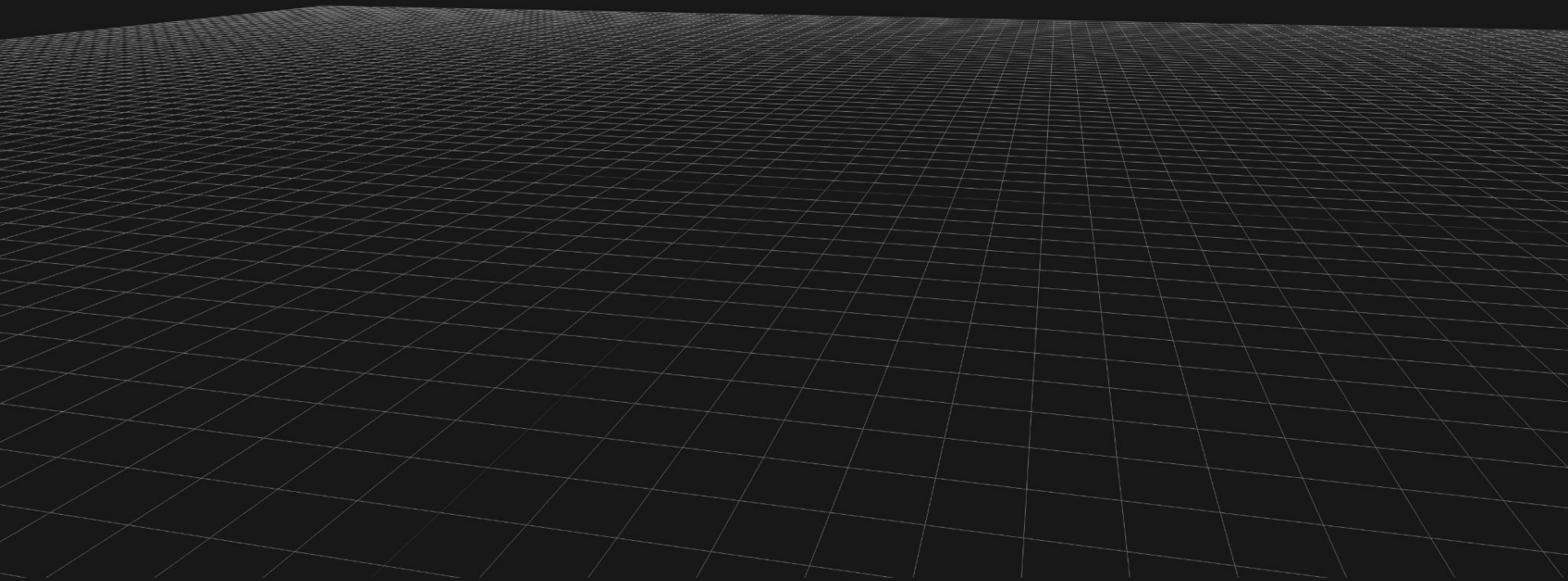
```
$ npm i
added 1164 packages, and audited 1165 packages in 8s
...
found 0 vulnerabilities

$ npm start

> cg1-demo@1.0.0 start
> npx webpack serve
...
ℹ ｢wds｣: Project is running at http://localhost:8080/
...
webpack 5.31.2 compiled successfully in 3622 ms
ℹ ｢wdm｣: Compiled successfully.
```



The initial screen is black.

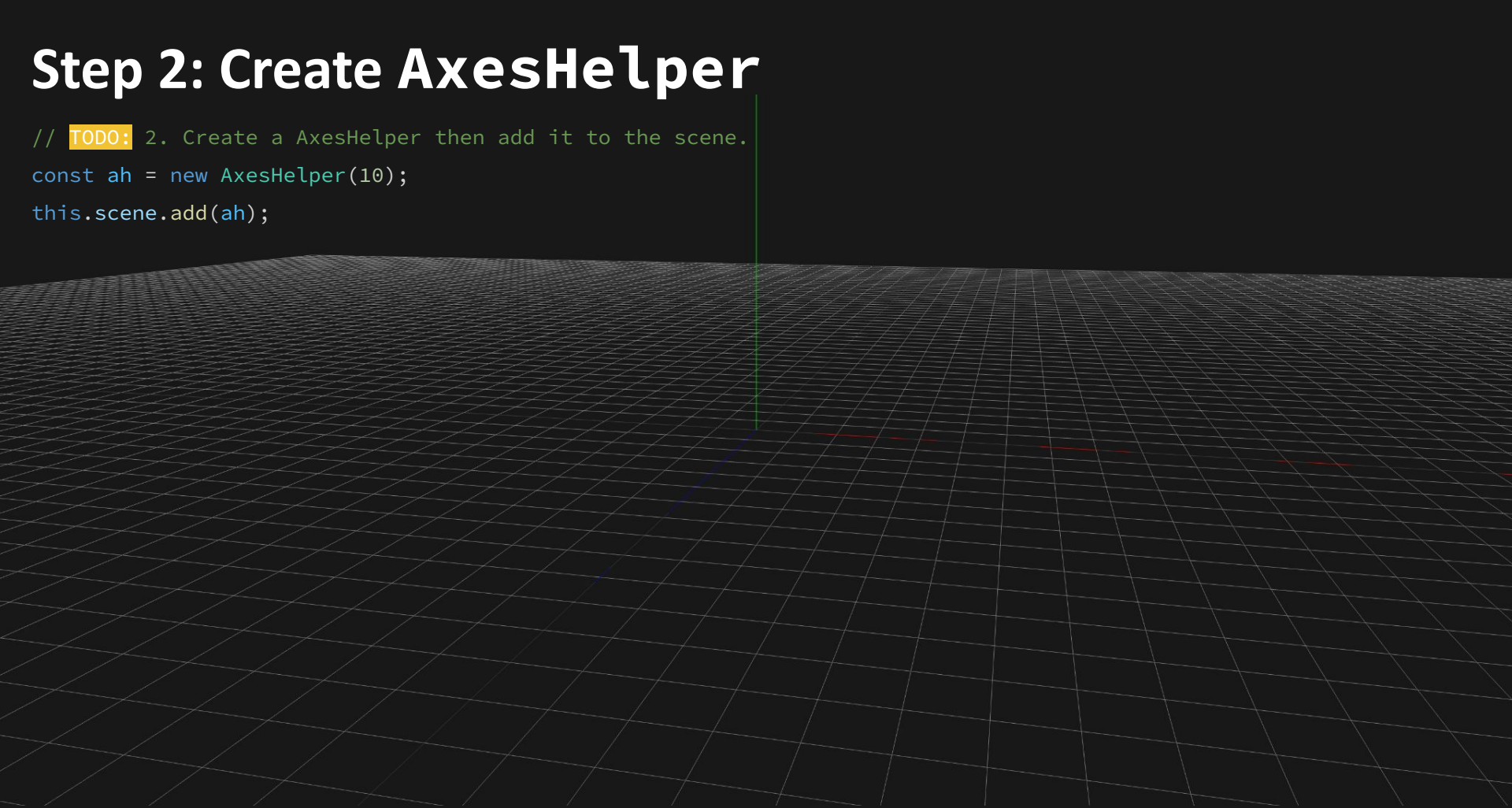
Step 1: Create `GridPlane`

```
// TODO: 1. Create a GridHelper then add it to the scene.
const gh = new GridHelper(gridParam.size, gridParam.divisions);
this.scene.add(gh);
```



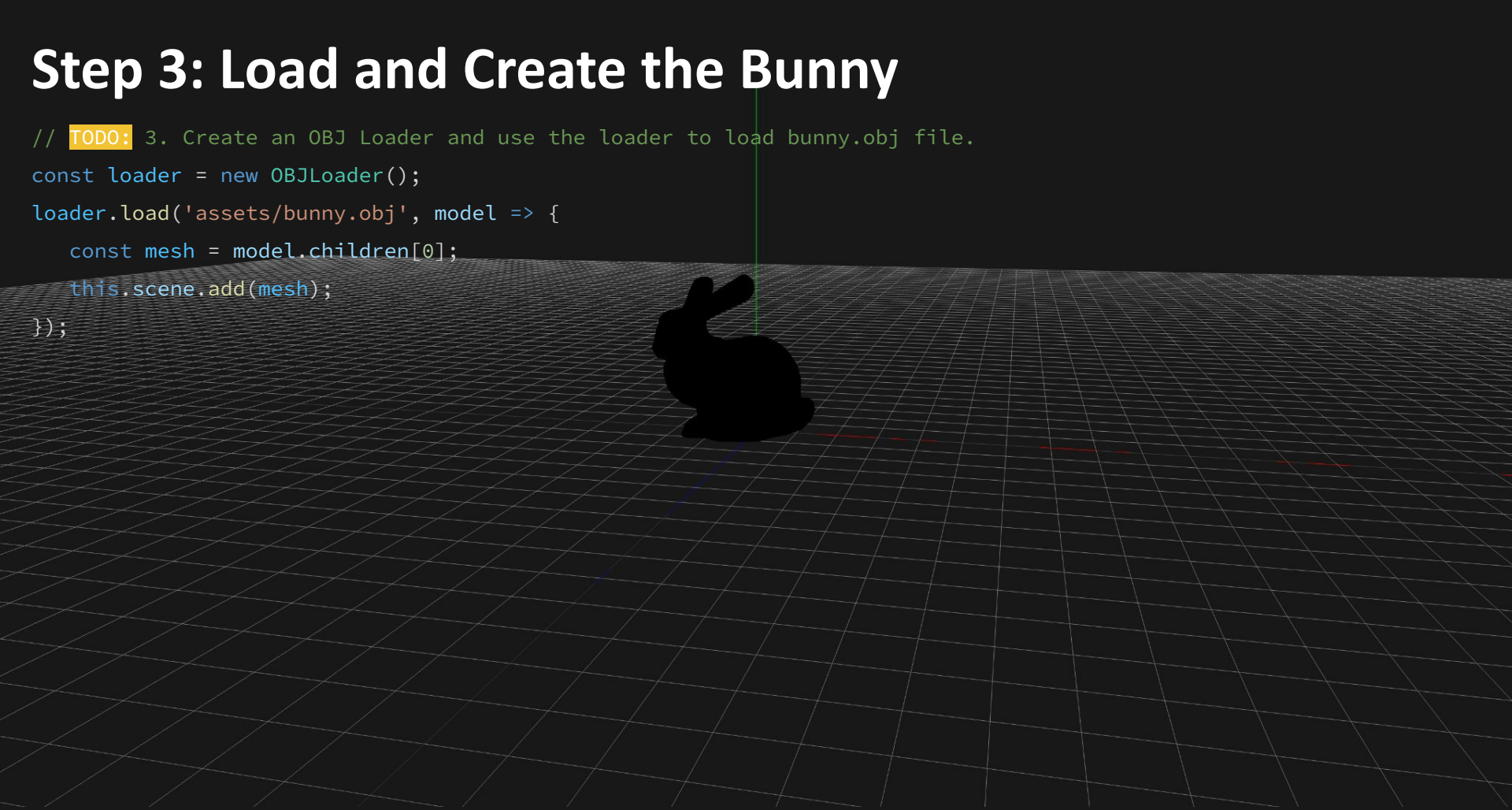
Step 2: Create AxesHelper

```
// TODO: 2. Create a AxesHelper then add it to the scene.
const ah = new AxesHelper(10);
this.scene.add(ah);
```



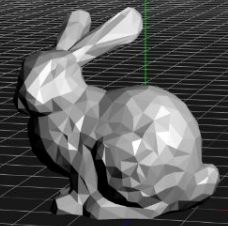
Step 3: Load and Create the Bunny

```
// TODO: 3. Create an OBJ Loader and use the loader to load bunny.obj file.
const loader = new OBJLoader();
loader.load('assets/bunny.obj', model => {
    const mesh = model.children[0];
    this.scene.add(mesh);
});
```



Step 4: Create PointLight

```
// TODO: 4. Create a PointLight and add to the group, then
// create a PointLightHelper and also adds to the light group.
const light = new PointLight(
  lightParams.color,
  lightParams.intensity,
  lightParams.distance
);
light.position.copy(lightParams.position);
g.add(light);

const helper = new PointLightHelper(light, 0.1);
g.add(helper);

this.scene.add(g);
```



Awesome! Did I just learned everything about graphics?

Apparently: **No!**

What else would be expected to learn?

- Using a library is just a matter of reading a document, understand fundamental principles helps us live longer and will not limit the skill to that specific library/engine
- By the end of the course, these questions can be easily answered:
  - What does .obj mean? How was the `bunny.obj` created? How to move the bunny in the scene?
  - How to put colors on the ground and the bunny?
  - What exactly is a *point light* and how does its parameters change the rendering result?
  - How *exactly* did three.js convert a bunny.obj file and rendered it in a browser?
  - Why are there no *shadows* and how can I create them?
  - How to make the scene more photo-realistic and pleasing?
  - What else do I need in order to create a character, animation, or even build a 3A-level game?
  - …

Why didn't we use XYZ?

"JavaScript is a joke!"

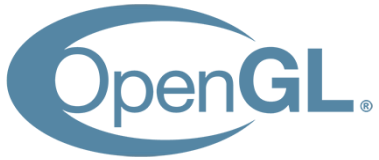
"I don't like TypeScript!"

"Unity is more popular!"

"Unreal Engine 5 is awesome!"

"I don't want write programs!"

…



- OpenGL/WebGL/Vulkan are cross platform standards
- DirectX is dedicated for Microsoft, and Metal is only for Apple
- Unity, Unreal, three.js … are engines/library build on top of them
- We minimize the setup cost and learn graphics **fundamentals**, thus we use three.js to get started but the final goal is to avoid using APIs.

  (you will see what that means :)

Why might 3D programming be non-trivial?

- Math is absolutely important, and not an easy task for most of people
- Geometric imagination and graphics creation are sometimes not easy, i.e. viewing 3D through 2D
- Tweaking is (super) time consuming and tedious, e.g. create an eyeball that is not just a sphere
- If there is a mistake, even a tiny arithmetic error, it is very likely to just get a black screen
  - WARNING: this is very frustrated sometimes :(
  - Debugging/testing are also not easy, and most of the time need a person to "see" what went wrong (e.g. "game tester")
- Different platforms with different APIs (DirectX v.s. Metal), and massive API breaking changes over time
- Interdisciplinary. Knowledge in physics, biology, and more might be needed

Don't panic! 🙂

Tutorial 1: Introduction

- Initial Setups
- Basics of Modern JavaScript
- TypeScript, Node.js and Its Ecosystem
- WebGL and three.js
- Summary

Summary

- We covered:
  - How to setup modern JavaScript/TypeScript programming environment with Git, GitHub, Markdown, and Node.js
  - How TypeScript code-skeleton get executed in a browser
  - How graphics programming could look like and how to create objects step by step
- Overwhelmed? Don't worry, we will repeat and practice them more

Resources

- Helpful short introduction courses (for CG1)
  - **Git** in 10 minutes: https://guides.github.com/introduction/git-handbook/
  - **GitHub** in 5 minutes: https://guides.github.com/introduction/flow/
  - **Markdown** in 3 minutes: https://guides.github.com/features/mastering-markdown/
  - **TypeScript** in 5 minutes: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html
  - **VSCode** in 30 minutes: https://code.visualstudio.com/docs/getstarted/introvideos
  - More? Check this: https://learnxinyminutes.com/
  - Search on YouTube if the documents are still tedious to read
- (Lifetime) Systematic References
  - The Modern JavaScript Tutorial https://javascript.info/
  - The TypeScript Handbook https://www.typescriptlang.org/assets/typescript-handbook.pdf
  - Pro Git https://git-scm.com/book/en/v2
  - Documentation for Visual Studio Code https://code.visualstudio.com/docs

Next

Transformation