

JavaScript and WebGL

CS425: Computer Graphics I

Fabio Miranda

<https://fmiranda.me>

Overview

- Web technologies
- Web environment
- JavaScript
- WebGL

Web technologies

HTML



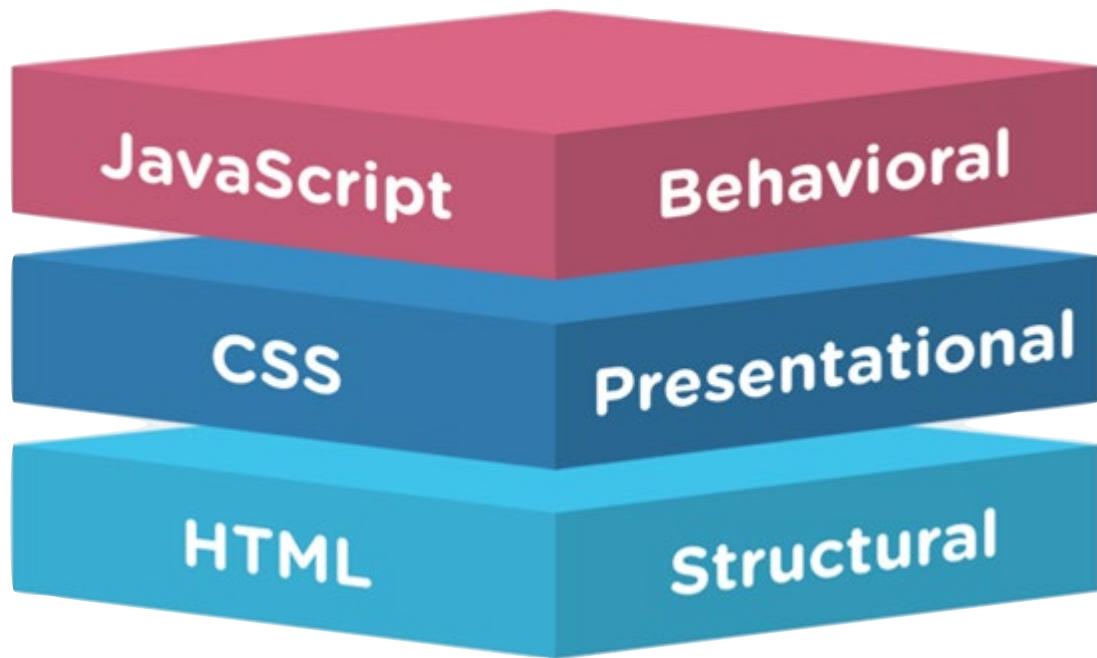
CSS



JS



Web technologies



- **JavaScript:** manage user interaction with the structure and presentation.
- **CSS:** manage presentation
- **HTML5:** mark-up language to structure the content of web pages.

HTML: First example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>First example</title>
</head>
<body>

Content here

</body>
</html>
```

Elements:
<element>
</element>

Examples:
html, head, body

HTML: First example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>First example</title>
</head>
<body>

Content here

</body>
</html>
```

Attributes:

<... attribute="...">

Examples:

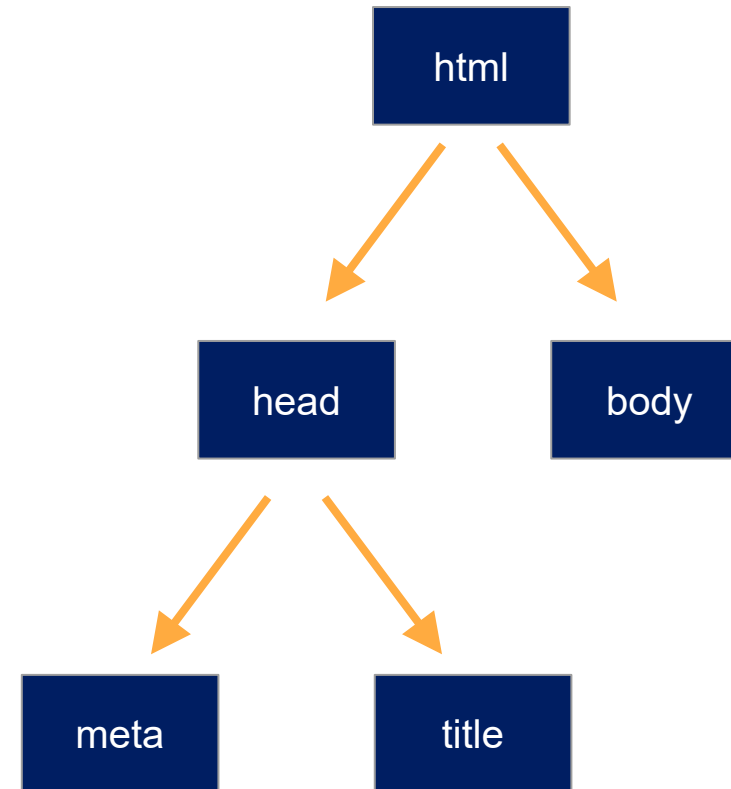
lang, charset, id

HTML: First example

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>First example</title>
</head>
<body>

Content here

</body>
</html>
```



HTML: Elements and attributes

Main root: `<html>`

Metadata: `<base>`, `<head>`, `<link>`, `<meta>`, `<style>`, `<title>`

Root: `<body>`

Content sectioning: `<address>`, `<main>`, ..., `<h1>`

Text content: `<div>`, `<figure>`, `<hr>`, ``

Image and multimedia: `<audio>`, ``, ..., `<video>`

Scripting: `<canvas>`, `<script>`

...

Cascading Style Sheets (CSS)

```
body {  
  margin: 10px;  
  font-size: 20px;  
}
```

Selectors:

selector {

...

}

Examples:

body, #first, .special

Cascading Style Sheets (CSS)

Different selectors

```
h1, h2 {  
    ...  
}
```

```
.classe p {  
    ...  
}
```

```
div > p {  
    ...  
}
```

```
#id1 ~ #id2 {  
    ...  
}
```

```
div[attr] {  
    ...  
}
```

```
div[attr="val"] {  
    ...  
}
```

Cascading Style Sheets (CSS)

```
body {  
  margin: 10px;  
  font-size: 20px;  
}
```

Properties:

key: value;

Examples:

margin, color, font-size

Putting it all together

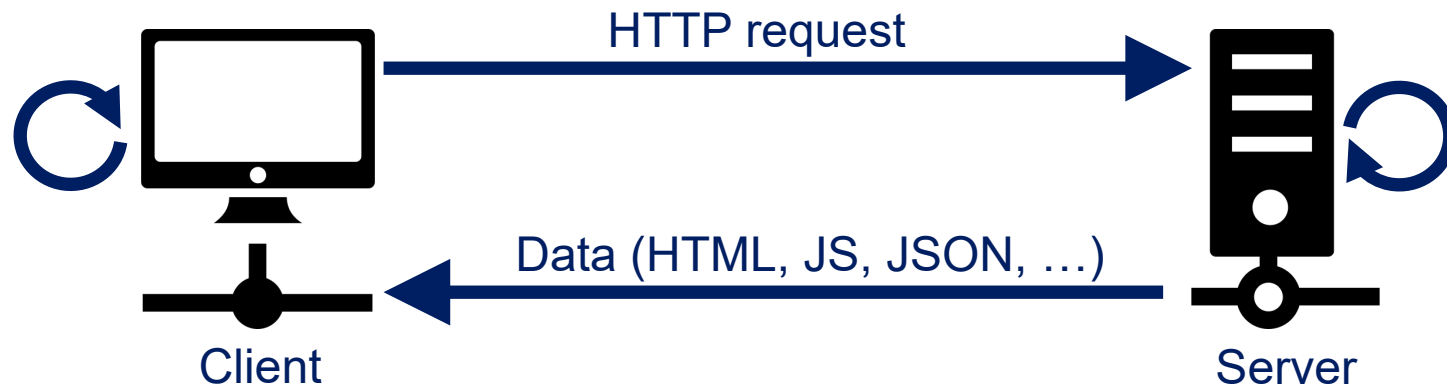
```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>First example</title>
</head>
<style>
  body { background-color: darkblue;}
  div { background-color: mediumslateblue;}
  #mydiv { background-color: red;}
  .myclass { background-color: mediumseagreen;}
</style>
<body>
  <div id="mydiv">This is a div.</div>
  <div id="myseconddiv">This is another div.</div>
  <div class="myclass">This is another div.</div>
</body>
</html>
```



Client and server

Client-side:

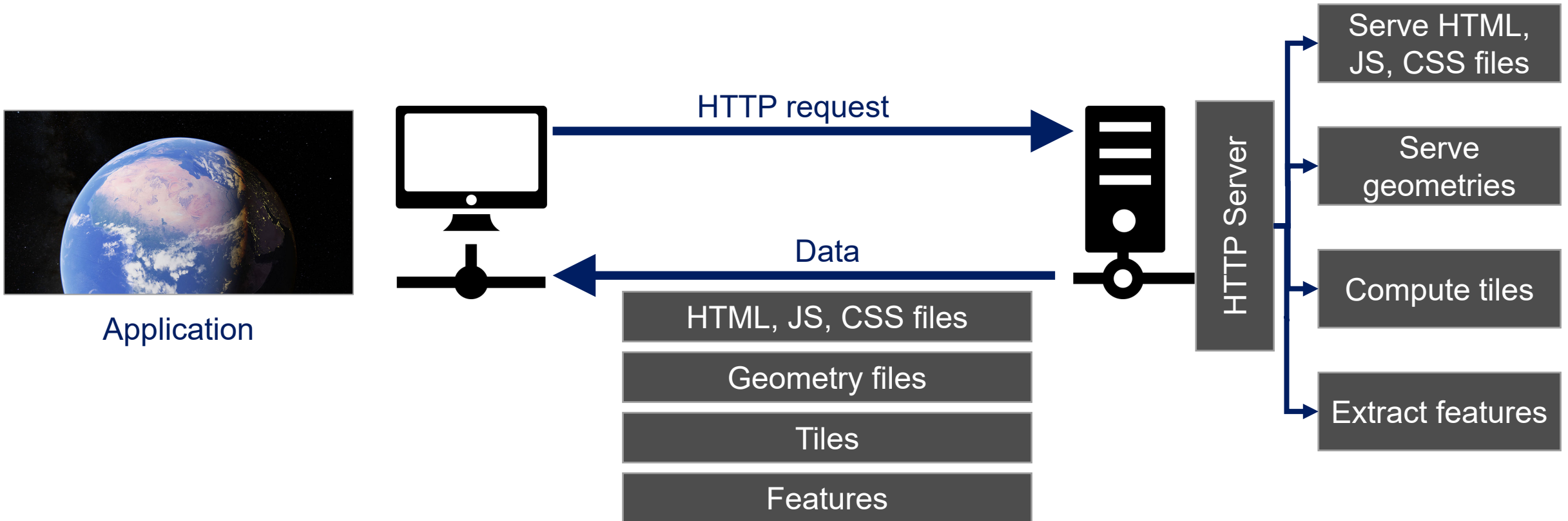
- Rendering
- Interaction
- Light-weight aggregation
- Filtering



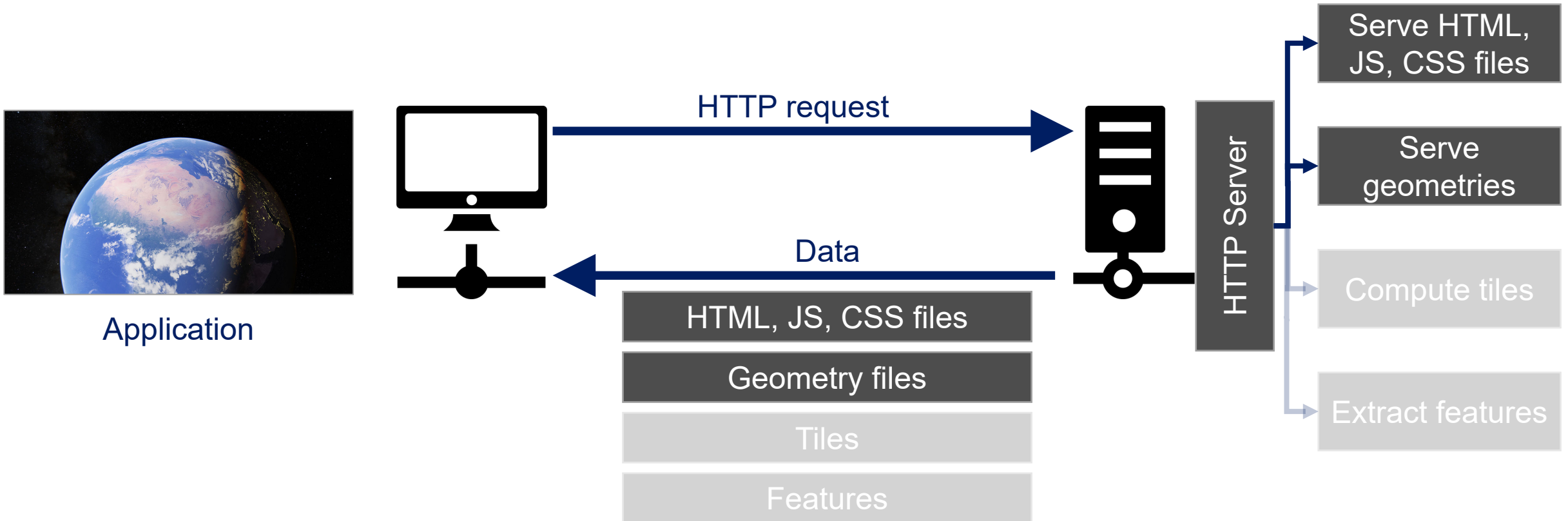
Server-side:

- Database query
- Feature extraction
- Data mining

Client and server



CS425: Client and server



CS425: Client and server

```
example@DESKTOP MINGW64 ~/example
$ python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Detailed steps: <https://mzl.la/3bSLff0>

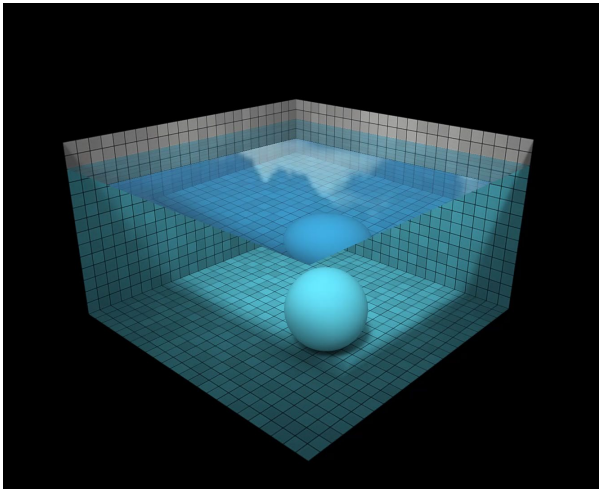
JavaScript: a client-side programming language



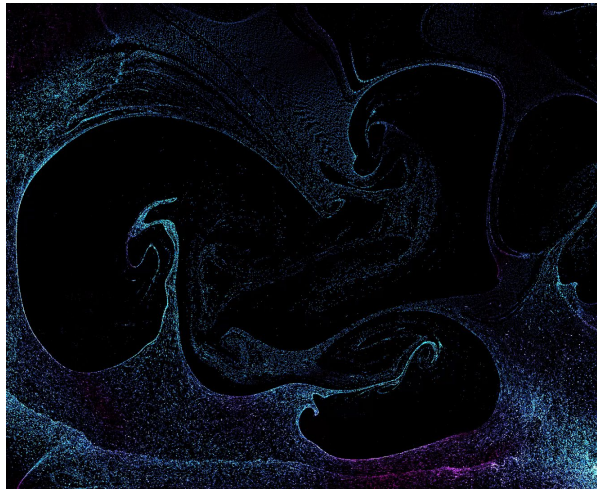
- Interpreted object-oriented language.
- Loosely typed language.
 - Does not require a variable type to be specified.
- Add, delete, and modify nodes from the document tree.
- Integration with other frameworks and toolkits:
 - Qt
 - Swift

JavaScript: a client-side programming language

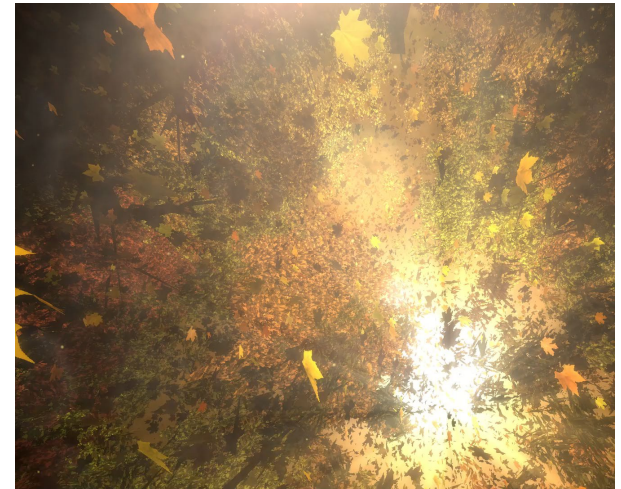
- Is JavaScript slow?
 - JavaScript engines in browsers are getting much faster.
 - Not an issue for graphics, since we transfer the data to the GPU with WebGL.



<http://madebyevan.com/webgl-water/>



<https://haxiomic.github.io/projects/webgl-fluid-and-particles/>



<http://oos.moxiecode.com/js-webgl/autumn/>

JavaScript basics

- Two scopes:
 - Local
 - Global
- Variable created inside a function with 'var' keyword: local to function.
 - Created and destroyed every time function is called.
 - BUT: variables declared without 'var' keyword are always global.
- Variable created outside a function: global

JavaScript basics

- Inserting JavaScript code in a web page:
 - Inside an HTML tag script.
 - In an external file.
 - As an HTML attribute value.

```
<script type="text/javascript">  
    alert("Here is an example.");  
</script>
```

```
<script type="text/javascript" src="file.js"></ script>
```

Statements, comments, and variables

- Statements: separated by new line or semicolon.
- Comment:
 - Single line: `// here is a comment`
 - Multi line: `/* here is a comment */`
- Loops and iteration:
 - `for`, `for...in`, `for...of`, `do...while`, `while`.
- Variables:
 - Assignment operator (`=`) to assign values.

Variable scope

```
var message = 'Hi';

function modify1(){
  var message = 'Hello';
}

function modify2(){
  message = 'Ola';
}

modify1();
console.log(message);

modify2();
console.log(message);
```

Hi

Ola

Functions

- Different ways to define functions:
 - Named
 - Anonymous
- Function expressions cannot be used before they appear in the code.

Function declaration

Function expression

```
function namedFunction1() {  
    console.log('Named function 1');  
}  
  
var myNamedFunction = function namedFunction2() {  
    console.log('Named function 2');  
}  
  
var myAnonFunction = function() {  
    console.log('Anonymous function');  
}
```

Anonymous function

Functions

- Function declarations load before any code is executed, while function expressions load only when the interpreter reaches that line.
- Function expressions: closures, arguments to other functions

```
alert(foo());  
function foo() { return 5; }
```

Function declaration: error in this case, as foo wasn't loaded yet.

```
alert(foo());  
var foo = function() { return 5; }
```

Function expression: alerts 5.
Declarations are loaded before any code can run.

Functions



- Functions are first-class objects:
 - Supports passing functions to other functions.
 - Returning them as values from other functions.
 - Assigning them to variables or data structures.
- Closure:
 - Function that maintains the local variables of a function, after the function has returned.

Closure example

```
function sayHi(name){  
    var whatToSay = 'Hi ' + name;  
  
    return function(){  
        console.log(whatToSay);  
    }  
}  
  
var say = sayHi('Bob');  
say();
```

A closure: a function inside a function

No matter where it is executed, closure function will always remember variables from sayHi.

Data types: numbers and strings

- Numbers: a primitive data type (32-bit float).
- String: sequence of characters.
- Booleans.

```
var aux1 = 3.0;  
var aux2 = 3;  
var aux3 = '3';  
  
console.log(aux1+aux2+aux3);  
console.log(aux3+aux2+aux1);
```

"63"

"333"

Objects

- In JavaScript, objects are a collection of properties with a name and a value.

```
var myObject = new Object();  
console.log(myObject);
```

```
myObject.name = "My Object";  
console.log(myObject);
```

Object { }

Object { name: "My Object" }

Arrays

- List-like objects.

```
var cities = ['NYC', 'Chicago'];  
  
console.log(cities[0]);  
console.log(cities[cities.length-1]);  
  
cities.forEach(function(item, index) {  
    console.log(item, index);  
})
```

NYC

Chicago

NYC 0
Chicago 1

Arrays

- List-like objects.

```
cities.push('LA'); // in place  
console.log(cities);
```

["NYC", "Chicago", "LA"]

```
cities.pop(); // in place  
console.log(cities);
```

["NYC", "Chicago"]

```
var pos = cities.indexOf('Chicago');  
console.log(pos);
```

1

```
cities.splice(pos, 1);  
console.log(cities);
```

["NYC"]

Example: map

```
var a = [1, 2, 3];

for(var i=0; i<a.length; i++){
    a[i] = a[i] * 2;
}

for(var i=0; i<a.length; i++){
    console.log(a[i]);
}
```

```
var a = [1, 2, 3];

function map(f, a){
    for(var i=0; i<a.length; i++){
        a[i] = f(a[i]);
    }
}

map(function(x){return x * 2;}, a);
map(alert, a);
```

Example: reduce

```
var nums = [1, 2, 3, 4];

function sum(a){
  var sum = 0;
  for(var i=0; i<a.length; i++){
    sum += a[i];
  }
  return sum;
}

function mult(a){
  var mult=1;
  for(var i=0; i<a.length; i++){
    mult *= a[i];
  }
  return mult;
}

console.log(sum(nums));
console.log(mult(nums));
```

```
var nums = [1, 2, 3, 4];

function reduce(f, a, init){
  var s = init;
  for(var i=0; i<a.length; i++){
    s = f(s, a[i]);
  }
  return s;
}

function add(a, b){
  return a+b;
}

function mult(a, b){
  return a*b;
}

console.log(reduce(add, nums, 0));
console.log(reduce(mult, nums, 1));
```


Manipulating documents

- So far: HTML, CSS, JavaScript.
- But how can we use JavaScript to modify nodes from DOM?
- Answer: **document object**.
- When an HTML document is loaded by a browser, it becomes a **document object**, containing the root node of the HTML document.

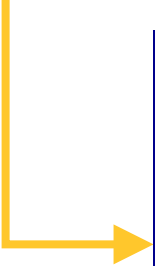
Document object

```
>> document
< HTMLDocument https://www.google.com/
  URL: "https://www.google.com/"
  ▶ __wizdispatcher: Object { La: trigger(c) ↗, Fa: {...}, Aa: false, ... }
  ▶ __wizmanager: Object { w0: false, JN: (1) [...], Ha: 10, ... }
  ▶ activeElement: <body id="gsr" class="hp vasq big" jsmodel="TvHxbe" jsaction="VM8bg:.CLIENT;hWT9Jb:.CL...:CLIENT;kWlxhc:.CLIENT">
    alinkColor: ""
  ▶ all: HTMLAllCollection { 0: html , 1: head , 2: meta , ... }
  ▶ anchors: HTMLCollection { length: 0 }
  ▶ applets: HTMLCollection { length: 0 }
  baseURI: "https://www.google.com/"
  bgColor: ""
  ▶ body: <body id="gsr" class="hp vasq big" jsmodel="TvHxbe" jsaction="VM8bg:.CLIENT;hWT9Jb:.CL...:CLIENT;kWlxhc:.CLIENT">
    characterSet: "UTF-8"
    charset: "UTF-8"
    childElementCount: 1
```

DOM elements using selectors

```
var allDivs = document.querySelector('div');  
var myDiv = document.querySelector('#mydiv');  
var mySecondDiv = document.querySelector('#myseconddiv');  
var myClass = document.querySelector('.myclass');
```

```
mySecondDiv.textContent = 'This is a modified div.';
```



This is a div.
This is a modified div.
This is another div.

DOM elements using selectors

```
var newDiv = document.createElement('div');  
newDiv.textContent = 'This is a new div.';  
newDiv.className = 'myclass';  
document.querySelector('body').appendChild(newDiv);
```



This is a div.
This is a modified div.
This is another div.
This is a new div.

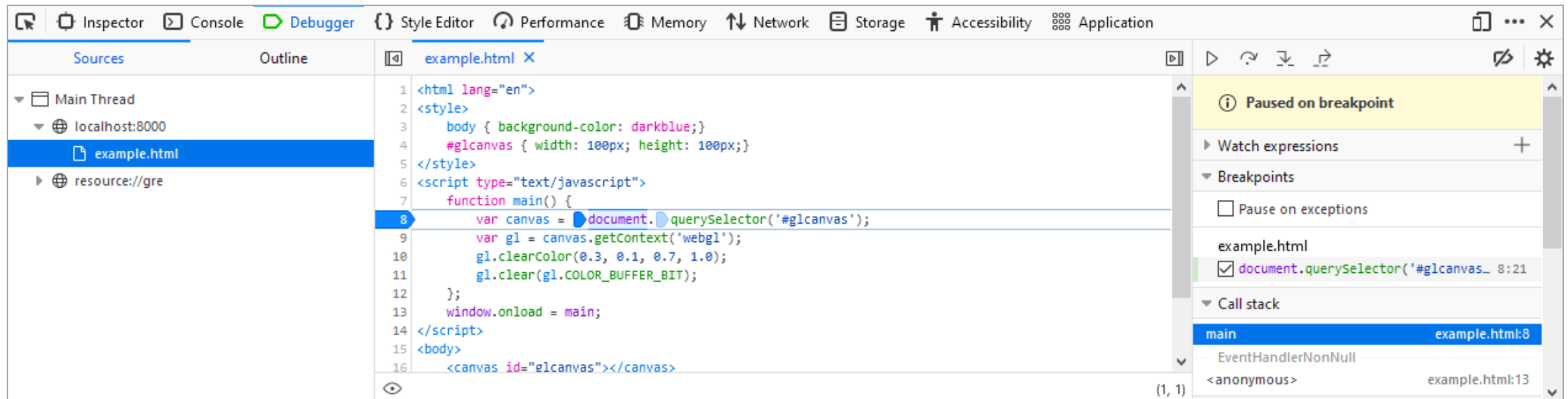
Event handlers

- Events are actions like being clicked, pressed keys, getting focus, etc.
- Different ways to specify handlers for a particular event:

```
<button onclick="handleClick()">
```

```
document.querySelector("button").onclick = function(event) {}
```

Debugging JavaScript



Finally drawing something

- Several ways to draw graphics on the web:
 - SVG
 - XML-based format for vector images.
 - Simple option for small data.
 - Easy event and CSS integration.
 - Canvas
 - HTML element.
 - No object-level interaction.
 - WebGL
 - Complex 3D geometries.
 - Uses rendering pipeline.
 - Hardware acceleration.

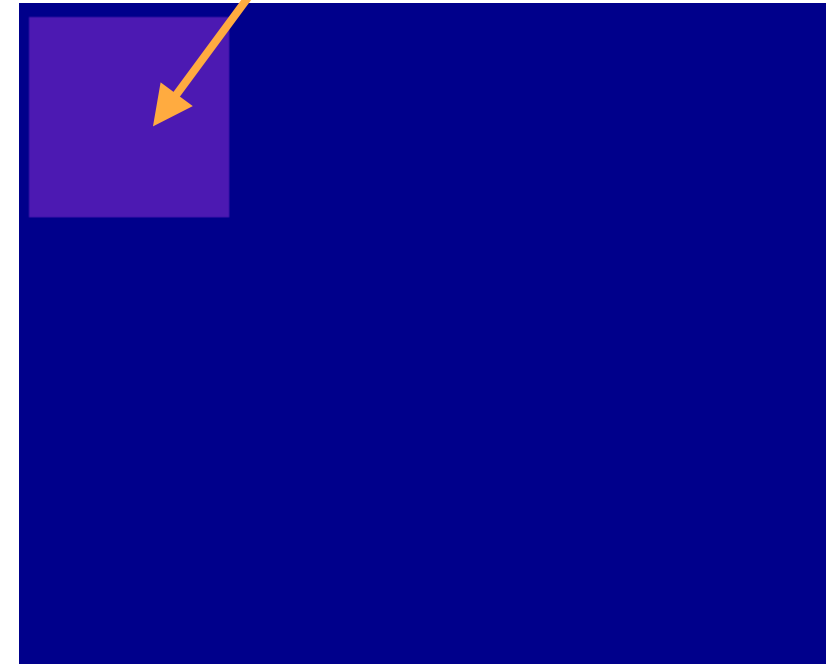
WebGL: a bird's-eye view

- API for rendering graphics within a web browser without plug-ins.
- Hardware accelerated.
- Shader based (no fixed-function API).
 - Fixed function pipeline: set of calls for matrix transformation, lighting.
 - Programmable pipeline: shaders for vertex and fragment processing.
- WebGL 2.0 based on OpenGL ES 3.0.

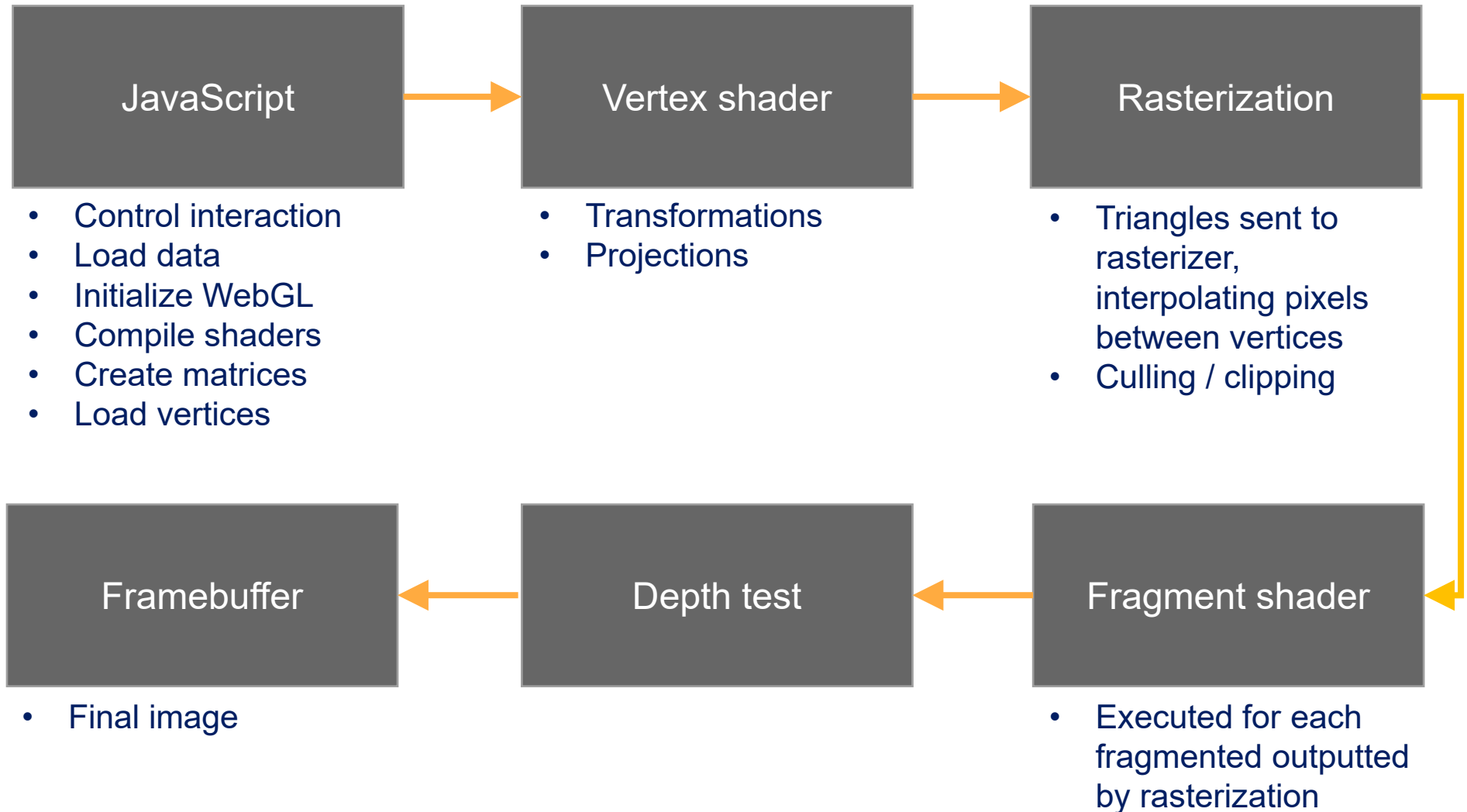
WebGL: a bird's-eye view

```
<html lang="en">
<style>
  body { background-color: darkblue;}
  #glcanvas { width: 100px; height: 100px;}
</style>
<script type="text/javascript">
  function main() {
    var canvas = document.querySelector('#glcanvas');
    var gl = canvas.getContext('webgl2');
    gl.clearColor(0.3, 0.1, 0.7, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);
  };
  window.onload = main;
</script>
<body>
  <canvas id="glcanvas"></canvas>
</body>
</html>
```

WebGL canvas



WebGL: a bird's eye view



Lab

- Web server:
 - Python: <https://mzl.la/3bSLff0>
- Development:
 - Visual Studio Code: <https://code.visualstudio.com/>
 - Atom: <https://atom.io/>
 - Sublime: <https://www.sublimetext.com/>
- Double check if your browser supports WebGL:
<https://get.webgl.org/>

Useful links



<https://developer.mozilla.org/en-US/docs/Web/HTML>

<https://developer.mozilla.org/en-US/docs/Web/CSS>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript>