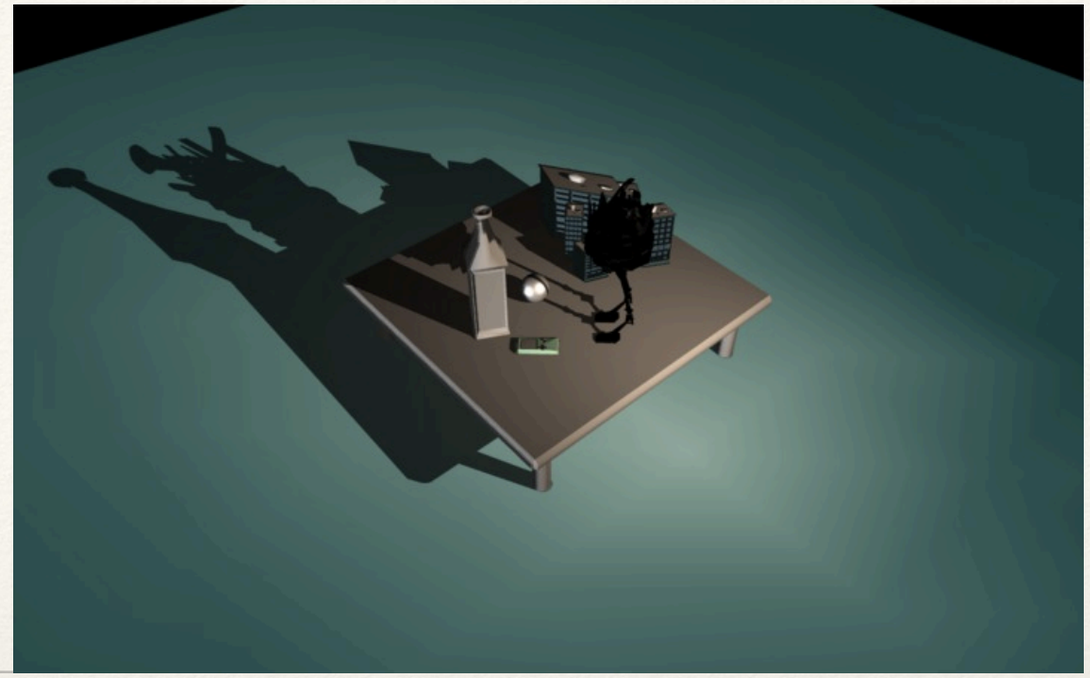
Dr. Sarah Abraham
University of Texas at Austin
Computer Science Department



Importing into Three.js

Elements of Graphics
CS324e

Importing Models

- Many file format loaders available under examples -> js -> loaders
  - Assimp, FBX, OBJ, SVG, Collada, etc
- Copy loading script in local project and include in .html
  - `<script src="js/OBJLoader.js"></script>`

Model Loading

- Create a group to store the loaded object and use the OBJLoader to import

```
let objloader = new THREE.OBJLoader();

let object = new THREE.Group();
objloader.load('path_to_obj', function(obj) {
      object.add(obj);
   }
);
…
scene.add(object);
```

First Class and Anonymous Functions

- Javascript supports first class functions:
  - Functions treated like any other variable
  - Functions can be passed in as an argument to another function
- Javascript supports anonymous functions:
  - Functions without a name are anonymous
  - Must be stored in a variable/passed in as an argument
  - Anonymous functions are first class functions

Consider ObjLoader…

- `load()` has 4 parameters:
  - `url` (String)
    - Contains path/url to object to be loaded
  - `onLoad` (Function)
    - A function called upon successful loading
  - `onProgress` (Function)
    - A function called while loading is in progress
  - `onError` (Function)
    - A function called if an error occurs during loading

Load Callbacks

- `onLoad` receives a Object3D as an argument when successful
- `onProgress` receives an XMLHttpRequest instance containing total and loaded bytes as an argument when loading
- `onError` receives an error as an argument when an error occurs

Anonymous vs Named Functions?

- Possible to pass in named functions rather than anonymous functions

```
function onLoad() { … }

function onProgress() { … }

function onError { … }


loader.load('path_to_obj', onLoad, onProgress,
onError);
```

- But libraries often designed around anonymous functions to support asynchronous calls

Consider Setting an Object's Material

- Must set material properties for a given mesh within `onLoad` function

```
objloader.load(str, function(obj) {
   obj.traverse(function(child) {
      if (child.isMesh) {
        let material = new THREE.MeshBasicMaterial();
        child.material = material;
        }
   });
});
```

The onLoad Function

- Must set material properties for a given mesh within `onLoad` function

```
objloader.load(str, function(obj) {
   obj.traverse(function(child) {
      if (child.isMesh) {
        let material = new THREE.MeshBasicMaterial();
        child.material = material;
        }
   });
});
```

Object Traversal

- Object3Ds have a `traverse` function
- `traverse` takes a **`callback`** function as an argument
- `callback` takes an Object3D as an argument
  - Executes the `callback` code on the argument (Object3D) and all its descendants

Traverse Callback Function

- `callback` called on child and all of child's descendants
- Must set the object's material within this callback to apply to the mesh

```
obj.traverse(function(child) {
 if (child.isMesh) {
    let material = new THREE.MeshBasicMaterial();
    child.material = material;
    }
});
```

Running Locally

- Browsers will not load from external files due to security restrictions
  - Ideally will create a local server to handle this
- Node example:
  - `sudo npm install http-server -g` (may need to install Node package to use npm)
  - `http-server . -p 8000` (start up server in local directory)
- Open this address in a browser: `http://127.0.0.1:8000/`
- https://threejs.org/docs/index.html#manual/en/introduction/How-to-run-things-locally

Loading JSON documents

1. Use `XMLHttpRequest` to access file information
   - Works the same locally as on a server
2. Use `JSON.parse()` to extract json into Javascript objects and arrays
3. Extract relevant information and store in appropriate data structures

XMLHttpRequest: JSON

```
function loadJSON(callback) {
   let request = new XMLHttpRequest();
   request.overrideMimeType("application/json");
   request.open("GET", "path_to_data", true);
   request.send();
   request.onreadystatechange = function () {
      if (request.readyState === 4 && request.status === 200)
      {
         callback(request.responseText);
      }
   };
}
```

Set type of returned data

Set request method, path to data and if asynchronous

On Ready State Change

- `onreadystatechange` is an event handler for when the `readystate` of data changes
  - `readystate` is 4 when operation is complete
  - request status is 200 when request has succeeded

```
request.onreadystatechange = function () {
   if (request.readyState === 4 && request.status === 200)
   {
      callback(request.responseText);
      }
   };
```

Parsing the Request via Callback

anonymous function for callback

jsonData is either a Javascript object or array (depending on json)

```
loadJSON(function(response) {
  jsonData = JSON.parse(response);
  console.log(jsonData.key1);
  console.log(jsonData.key2);
  for (let i = 0; i < jsonData.array1.length; i++) {
    console.log(jsonData.array1[i]);
  }
});
```

Loading XML Documents

- Same concept as JSON loading!

1. Use `XMLHttpRequest` to access file information
   - Works the same locally as on a server
2. Use DOM`Parser().parseFromString()` to extract xml data
3. Extract relevant information by navigating hierarchy and store in appropriate data structures

XMLHttpRequest: XML

```
function loadXML(callback) {
   let request = new XMLHttpRequest();
   request.overrideMimeType(“text/xml”);
   request.open(“GET”, “path_to_data”, true);
   request.send();
   request.onreadystatechange = function () {
      if (request.readyState === 4 && request.status === 200) {
         callback(request.responseText);
      }
   };
}
```

Parsing the Request via Callback

```
loadXML(function (response) {
   parser = new DOMParser();
   xmlData = parser.parseFromString(response, “text/xml”);
   root = xmlData.childNodes[0];
   console.log(root.getElementsByTagName(“element1”)
   [0].firstChild.nodeValue);
   let items = root.getElementsByTagName(“element2”)
   [0].getElementsByTagName(“items”);
   let numitems = items.length;
   for (let i = 0; i < numitems; i++) {
     console.log(items[i].firstChild.nodeValue);
   }
});
```

Other Libraries

- Many, many other libraries for simplifying data transfer and handling in Javascript
  - jQuery used for data traversal and management
  - D3 designed for data visualization and data processing
- All of them built around same principles of callbacks and DOM (document object model) used to model HTML structures

Hands On: Loading Files

- Create or find a small json or xml file and include it in your project directory
- Use XMLHttpRequest to import this file into your webpage's main script
- Parse this file into a Javascript or XML object
- Create an Object3D (or objects) that is based on the data in your file and add it to the main scene