Introduction to Computer Graphics with WebGL

Week7

Instructor: Hooman Salamat

Advanced geometry loading techniques: JavaScript Object Notation (JSON) and AJAX

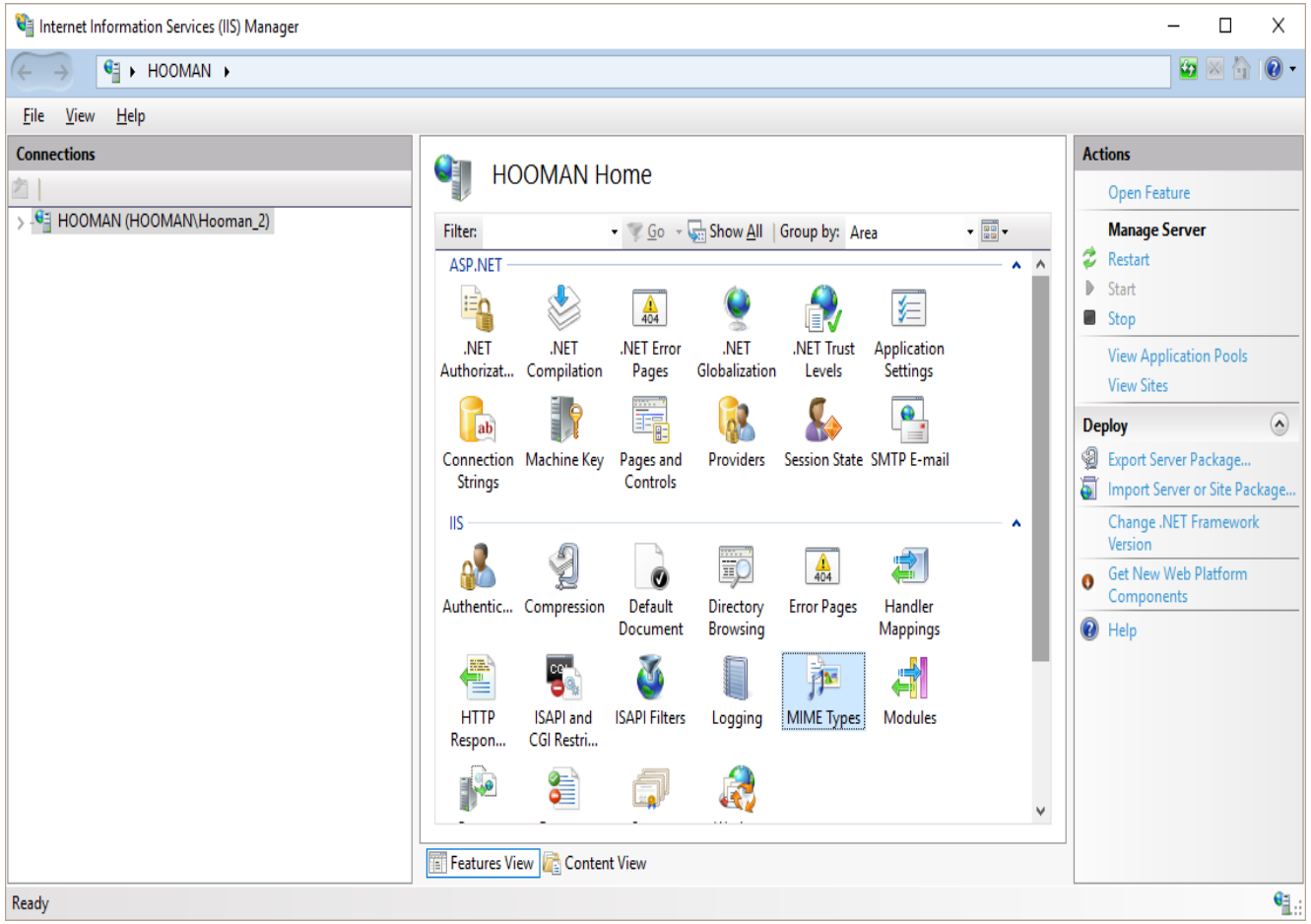
- So far, we have rendered very simple objects. Now let's study a way to load the geometry (vertices and indices) from a file instead of declaring the vertices and the indices every time we call initBuffers.
- To achieve this, we will make asynchronous calls to the web server using AJAX. We will retrieve the file with our geometry from the web server and then we will use the built-in JSON parser to convert the context of our files into JavaScript objects. In our case, these objects will be the vertices and indices array.

Installing IIS

- To install IIS:
- In Windows, access the Control Panel and click **Add or Remove Programs**.
- In the Add or Remove Programs window, click **Add/Remove Windows Components**.
- Select the **Internet Information Services (IIS)** check box, click **Next**, then click **Finish**.

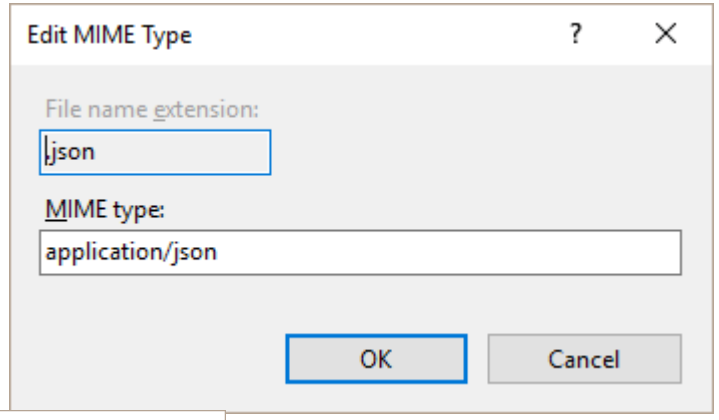
IIS Manager

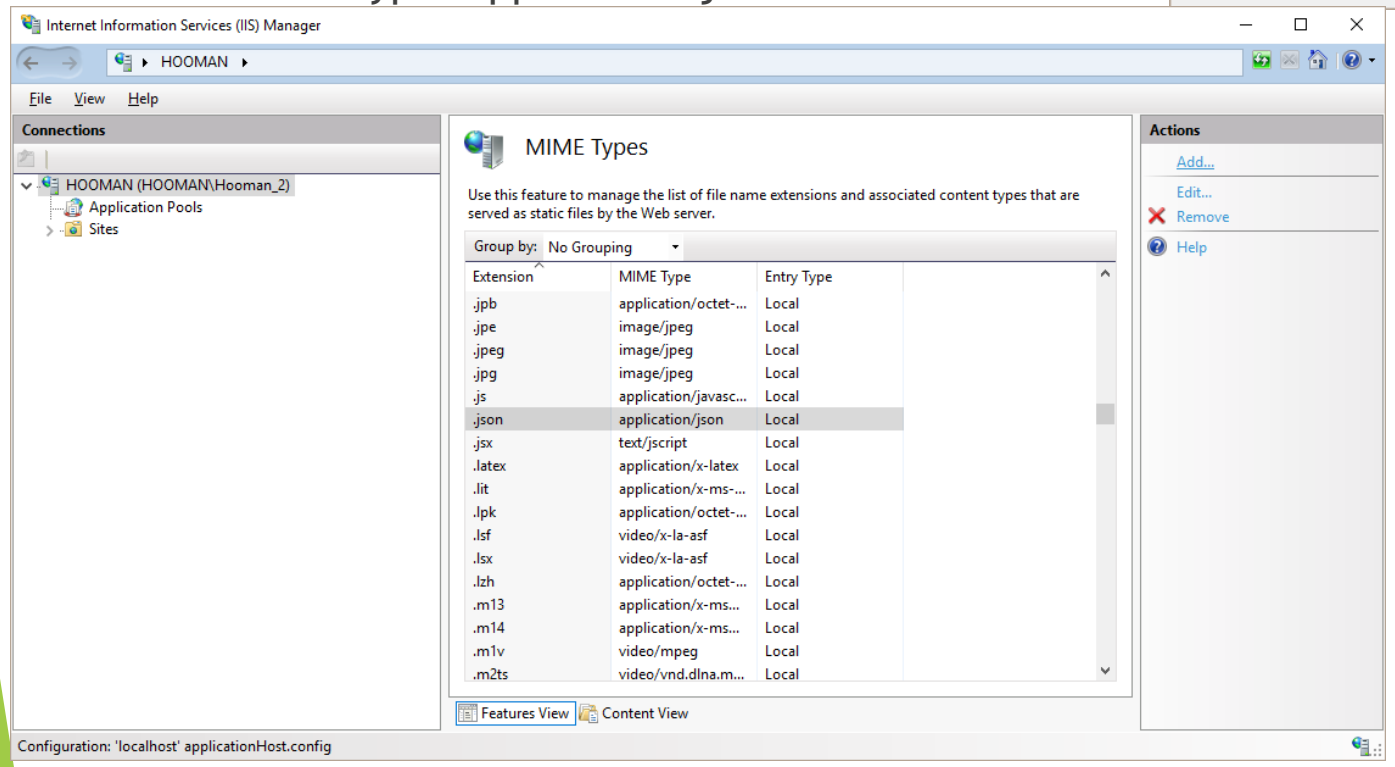
- Open IIS Manager
- Display properties for the IIS Server



IIS JSON Support

- Click **MIME Types** and then add the JSON extension:
  - File name extension: .json
  - MIME type: application/json





Introduction to JSON – JavaScript Object Notation

- **JSON** stands for **JavaScript Object Notation**. It is a lightweight, text-based, open format used for data interchange. JSON is commonly used as an alternative to XML.
- The *JSON format* is language-agnostic. This means that there are parsers in many languages to read and interpret JSON objects. Also, JSON is a subset of the object literal notation of JavaScript. Therefore, we can define JavaScript objects using JSON.

Defining JSON-based 3D models

- Assume for example that we have the model object with two arrays vertices and indices Say that these arrays contain the information described in the cone example as follows:

```
vertices =[
    -1.5, -0.809017, -0.587785,
    -1.5, -0.309017, -0.951057,
    -1.5, 0.309017,  -0.951057,
    -1.5, 0.809017,  -0.587785];
indices = [
    0, 1, 2,
    0, 2, 3];
```

- Following the JSON notation, we would represent these two arrays as an object, as follows:

```
var model = {
"vertices" : [
  -1.5, -0.809017, -0.587785,
  -1.5, -0.309017, -0.951057,
  -1.5, 0.309017,  -0.951057,
  -1.5, 0.809017,  -0.587785],
"indices" : [
  0, 1, 2,
  0, 2, 3]};
```

JSON Syntax Rules

- The extent of a JSON object is defined by curly brackets {}
- Attributes in a JSON object are separated by comma ,
- There is no comma after the last attribute
- Each attribute of a JSON object has two parts: a key and a value
- The name of an attribute is enclosed by quotation marks " "
- Each attribute key is separated from its corresponding value with a colon :
- Attributes of the type Array are defined in the same way you would define them in JavaScript

JSON encoding and decoding

- Most modern web browsers support native JSON encoding and decoding through the built-in JavaScript object JSON. Let's examine the methods available inside this object:

Method	Description
var myText = JSON.stringify(myObject)	We use JSON.stringify for converting JavaScript objects to JSON-formatted text.
var myObject = JSON.parse(myText)	We use JSON.parse for converting text into JavaScript objects.

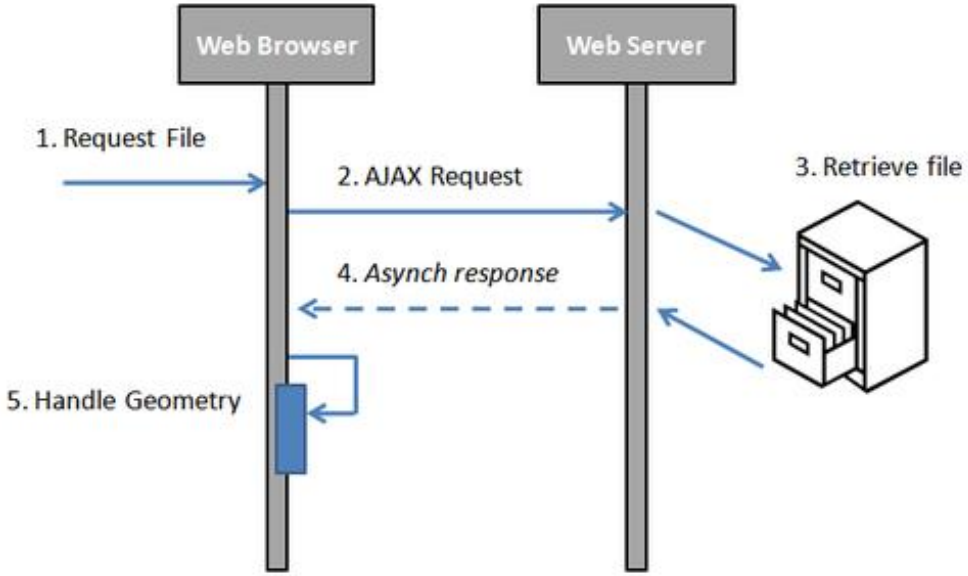
How to encode with the JSON notation

- Go to your Internet browser and open the interactive JavaScript console. (PC: Ctrl+Shift+J, Mac: Command+Alt+J)
- Create a JSON object by typing: var model = {"vertices":[0,0,0,1,1,1], "indices":[0,1]};
- Verify that the model is an object by writing: typeof(model)
- Now, let's print the model attributes. Write this in the console (press **Enter** at the end of each line):
  - model.vertices
  - model.indices
- Now, let's create a JSON text:
  - var text = JSON.stringify(model)
  - alert(text)

How to decode with the JSON notation

- What happens when you type text.vertices?
- As you can see, you get an error message saying that text.vertices is not defined. This happens because text is not a JavaScript object but a string with the peculiarity of being written according to JSON notation to describe an object. Everything in it is text and therefore it does not have any fields.
- Now let's convert the JSON text back to an object. Type the following:
  - var model2 = JSON.parse(text)
  - typeof(model2)
- Type: model2.vertices
- this is the way we will define our geometry to be loaded from external files.

Asynchronous loading with AJAX



Asynchronous loading of files by the web browser using AJAX

- **Request file**: First of all, we should indicate the filename that we want to load. Remember that this file contains the geometry that we will be loading from the web server instead of coding the JavaScript arrays (vertices and indices) directly into the web page.
- AJAX request: We need to write a function that will perform the AJAX request. Let's call this function loadFile. The code can look like this:
- If the readyState is 4, it means that the file has finished downloading.

```
function loadModel(filename){
    var request = new XMLHttpRequest();
    var resource = "http://" + document.domain + filename
    request.open("GET",filename);
    request.onreadystatechange = function() {
        console.info(request.readyState +' - '+request.status);
      if (request.readyState == 4) {
    if(request.status == 200) { //OK
handleLoadedModel(filename,JSON.parse(request.responseText));}
else if (document.domain.length == 0 && request.status == 0){ //OK but local, no web
server
            handleLoadedModel(filename,JSON.parse(request.responseText)); }
        else{
            alert ('There was a problem loading the file :' + filename);
            alert ('HTML error code: ' + request.status);} }   }
    request.send(); }
```

Continued…

- Retrieve file: The web server will receive and treat our request as a regular HTTP request. As a matter of fact, the server does not know that this request is asynchronous (it is asynchronous for the web browser as it does not wait for the answer). The server will look for our file and whether it finds it or not, it will generate a response. This will take us to step 4.
- Asynchronous response: Once a response is sent to the web browser, the callback specified in the loadFile function is invoked. This callback corresponds to the request method onreadystatechange. This method examines the answer. If we obtain a status different from 200 (OK according to the HTTP specification), it means that there was a problem. Hopefully the specific error code that we get on the status variable (instead of 200) can give us a clue about the error. For instance, code 404 means that the resource does not exist. In that case, you would need to check if there is a typo, or you are requesting a file from a directory different from the directory where the page is located on the web server. Different error codes will give you different alternatives to treat the respective problem. Now if we get a 200 status, we can invoke the handleLoadedGeometry function.

Continued…

- Handling the loaded model: In order to keep our code looking pretty, we can create a new function to process the file retrieved from the server. Let's call this handleLoadedGeometry function. Please notice that in the previous segment of code, we used the JSON parser in order to create a JavaScript object from the file before passing it along to the handleLoadedGeometry function. This object corresponds to the second argument (model) as we can see here. The code for the handleLoadedGeometry function looks like this:
- function handleLoadedGeometry: If you look closely, this function is very similar to one of our functions that we saw previously: the initBuffers function. This makes sense because we cannot initialize the buffers until we retrieve the geometry data from the server.

```
function handleLoadedModel(filename,payload) {
    model = payload; //save our model in a global variable so we can retrieve it in drawScene
    alert(filename + ' has been retrieved from the server');
    modelVertexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, modelVertexBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(model.vertices), gl.STATIC_DRAW);
    modelIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, modelIndexBuffer);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(model.indices), gl.STATIC_DRAW);
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, null);
    gl.bindBuffer(gl.ARRAY_BUFFER,null);
    gl.uniform3f(prg.modelColor,model.color[0], model.color[1],model.color[2]);
    modelLoaded = true;
}
```