Lab 8 | 3D Web Maps

First, I will make a distinction between 3D web mapping and 3D modeling. Both are extremely useful concepts in geovisualization, although we won't be covering the latter concept in this lab: 3D modeling has its own sets of standards, principles, and software that go far beyond the scope of this web mapping class. There is substantial overlap sometimes, however, perhaps most visibly with KML and Google Earth API (which we also haven't covered since we're working with open-access APIs). While we have discussed KML in terms of its usefulness to simple web mapping, it can also be used to store three-dimensional data (those of you who are interested can explore options here: https://developers.google.com/kml/documentation/kml_21tutorial). Furthermore, it appears that many major GIS organizations that have developed 3D views are working on incorporating 3D models from KML.

Anyway, you are already somewhat familiar with 3D "views" with Esri's ArcGIS JavaScript API, through in-class examples and demonstrations (in version 4 that we used, this was essentially as simple as switching to "SceneView" rather than the default "MapView") where we "redrew" the reference/base map over a 3D object. Plenty of other web GIS APIs have 3D mapping options, although most of them do require loading in additional JavaScript libraries to support 3D rendering and effects. Many examples you encounter at this time are concerned with displaying 2D data on a 3D visualization framework, such as "volumetric" mapping of phenomena. Others are focused on stylizing "base map" features such as buildings and their shadows, to give additional context to 2D data.

In this assignment, you will take a basic 2D University of Iowa campus map and transform it with realistic 3D effects, using the same underlying data set. Basically, University of Iowa facilities has this incredibly informative GIS layer about trees. They love so much they made an app just for it: https://maps.facilities.uiowa.edu/trees/

It's nice and all, but 3D views might make it more interesting. We will utilize ArcGIS JavaScript API once again this time to take advantage of something unique that they offer: pre-developed, realistic 3D models of all kinds of real-life objects, including detailed models for tree species, https://developers.arcgis.com/javascript/latest/guide/esri-web-style-symbols-3d/#vegetation. These sorts of items (and Esri has a long and growing list) help to build out 3D scenes. You can draw dynamically from multiple variables in a GIS database (for example tree **species** and **height**) to render more accurate representations in a 3D view. (Theoretically, using a similar method, you might also create your own unique models and then populate them into your web map view based on a field in the attributes, but that's a bit more complex than we're ready to take on). So, let's get back to these trees in Iowa City:

First, we need to find their data. If you use at the Developer Tools and view the sources, you'll notice that one is their server: https://maps.facilities.uiowa.edu/arcgis/rest/services/.

More specifically, the application pulls in the trees layer, https://maps.facilities.uiowa.edu/arcgis/rest/services/Features/Trees/MapServer/1

This is the *same* type of format we had worked with in previous labs. There are some updates, but once again, I'm keeping this easy enough to follow along.

Please read the instructions carefully and answer the questions in a word document. Then, publish your work to your GitHub Pages site as usual and submit your answers as a text or Word file through Canvas.

Create the Scene View

- First, we set up a classic SceneView in ArcGIS JavaScript API.
- Open a new text editor and start a new HTML file
  Once again, please use something like Notepad++ or Atom.io that will colorize your document, NOT Word or Notepad. Alternatively you could use the ArcGIS API for JavaScript Sandbox, but you will not be able to get feedback for errors this way

We are going to be following a familiar path: adding a head section that contains the .js and .css files from ArcGIS JavaScript API as well as styles for our map. We'll follow that in the head section with our own custom code, beginning with the modules we will use (the "require" section); followed by the functions executing those modules; then a list of variables (like feature services); and finally we'll "add" any layers or widgets to the map.

- First, add the following code to your empty file:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="initial-scale=1,maximum-scale=1,user-scalable=no"/>
    <title>Visualize features with realistic WebStyleSymbols - 4.14</title>
    <link rel="stylesheet"
href="https://js.arcgis.com/4.14/esri/themes/light/main.css"/>
    <script src="https://js.arcgis.com/4.14/"></script>
    <style>
      html,
      body,
      #viewDiv {padding: 0; margin: 0; height: 100%; width: 100%; }
   </style>
    <script>
require(["esri/WebScene","esri/views/SceneView","esri/layers/FeatureLayer",
"esri/layers/SceneLayer","esri/tasks/support/Query"],
function(WebScene, SceneView, FeatureLayer, SceneLayer, Query) {
        // Create the web scene
        var map = new WebScene({
          basemap: "satellite",
          ground: "world-elevation"
        });
        // Create the view
        var view = new SceneView({
          container: "viewDiv",
          map: map,
          camera: {
            position: {
              latitude: 41.6608232,
              longitude: -91.537668,
              z: 290
            },
            tilt: 60,
            heading: 330
          },
          environment: {
            lighting: {
              date: new Date("April 15, 2020 16:00:00 EDT"),
              directShadowsEnabled: true,
              ambientOcclusionEnabled: true
            }
          }
        });

    </script>
  </head>
 <body>
    <div id="viewDiv"></div>
 </body>
</html>
```

- Notice that relative to the last ArcGIS JS lab, we have added something new to our SceneView map parameters: in addition to setting the center of the map, we have introduced additional control though a new element, “camera.” This doesn’t merely involve picking a center starting position for the map with longitude (x coordinate) and latitude (y coordinate) as well as scale (zoom), rather we are setting the starting position of a hypothetical camera that “shows” the 3D scene, and there’s a new element to set the height above the ground (z). Additionally, “camera” comes with its own sub-properties: tilt and heading, where you can control the user’s specific starting orientation. A tilt of 0 degrees would be an overhead view looking straight down and 90 degrees would be looking “sideways,” perpendicular to the ground. Usually, you will want an oblique angle in between those figures. I used 60 in the code above.

- Next, I will turn your attention to another sub-property of the camera: lighting. There are a number of ways to use this function, but perhaps the most interesting aspect of it is that you can input a date and time, and Esri will calculate the sun angle and return shadows – from your 3D data – that correspond to that time. I have set this already to April 15th at 3:00PM.

- This works for the basics, but no layers are actually added to the map with just the code above. Farther down the page, you will need to add in the layers.

In the following steps, we’ll add the trees layer that I discussed in the introduction. Just as in the previous assignments, we can point directly to the data layer we want, and there’s no need to control or own the data to use it (and manipulate it) for our own purposes. Once we point to it and pull it into our web map, then we can begin the process of transforming it. In the previous ArcGIS JS API lab, you used a “simple renderer” to transform the Lincoln-Lancaster County GIS data on property assessed values into graduated symbols. Essentially, we’re doing the same thing here, but just a little more complex! Rather than utilizing simple renderers, we’re actually using unique value renderers, where we can name any number of quantitative and qualitative differences in our dataset and give them unique symbols. Furthermore, we will set this renderer to draw features based on qualitative differences (tree types), and also scale them proportionally based on another field (their height).

- First, press return several times to enter some space just before the </script> part of the code above. After we’ve set up the basics, we add a variable for the trees below that (I'm calling it "vegetationLayer):

```
//define and add vegetation layer
          var vegetationLayer = new FeatureLayer({
          url:
"https://maps.facilities.uiowa.edu/arcgis/rest/services/Features/Trees/MapServer/1",
          elevationInfo: {
            mode: "on-the-ground"
          },
          renderer: {
            type: "unique-value", // autocasts as new UniqueValueRenderer()
            field: "Scientific",
            defaultSymbol: {
              type: "web-style", // autocasts as new WebStyleSymbol()
              name: "Unknown",
              styleName: "EsriRealisticTreesStyle"
            },
            uniqueValueInfos: uniqueValueInfos,
            visualVariables: [
              {
                type: "size",
                field: "Height",
                axis: "height" // take the real height of the plant from the SIZE field
              },
              {
                type: "rotation",
                valueExpression: "random() * 360" // we use a random rotation, so that
plants look different
              }
            ]
          }
        });

        map.add(
          vegetationLayer
        );
});
```

- We call in the feature layer, as usual, but then add a comma and indicate that its "elevationInfo" will be set to "on-the-ground," to clamp features to the surface, regardless of any other internal data. Then, we set the "renderer" to allow US to customize the appearance. This is where it gets slightly tricker than previous exercises. Notice from the above code that we:
    - Set type to "unique-value"

    - Set field to “Scientific,” which is (if you look at the data source) a field in the trees data indicating the scientific name for the tree as recorded by University of Iowa
    - Create a default symbol, an “unknown,” gray tree.
    - Change tree size directly based on the “Height” field in the data using visual variables (just as we did with graduated symbols based on property assessment earlier)
    - We rotate the trees randomly so it looks a bit nicer and finally,
    - We ADD the trees layer to the map.

- But what else is happening in the renderer? There is a property called “uniqueValueInfos” that looks for differences across a particular field in the attributes to draw features differently. This is normally how we set up unique value renderers. We have this set to a variable called “uniqueValueInfos.” But we need to define what attributes we’re looking for and how the browser should render them. Note: Iowa has their data published in an Excel file too.

- This is pretty simple: we just say what value we want to pull from the tree GIS layer in the “value” field (remember that we’re using the “Scientific” field). And then we match it to a corresponding 3D Tree style from Esri’s list here (this is just an example to highlight the link; do NOT copy + paste yet) by assigning a name from the “EsriRealisticTreesStyle” group (fortunately Esri uses scientific names that are pretty close to what we want anyway):

```
var uniqueValueInfos = [
          {
            value: "Crataegus sp.",
            symbol: {
              type: "web-style", // autocasts as new
WebStyleSymbol()
              name: "Crataegus",
              styleName: "EsriRealisticTreesStyle"
            }
          }…
```

This is the attribute name in Iowa’s GIS data

This is Esri realistic trees object name you use for that attribute

    - I am providing the code below for matching up quite a few types of trees from the database with Esri. We will want to put this uniqueValueInfos variable section in between the two sections of code we have added previously. *I know there has been some confusion about why we place sections of code where we do. But the logic here is simple: we usually* *need to define something before we try to use/“execute” it*.

- Put this in between the previous code sections:

```
var uniqueValueInfos = [
          {
            value: "Crataegus sp.",
            symbol: {
              type: "web-style", // autocasts as new WebStyleSymbol()
              name: "Crataegus",
              styleName: "EsriRealisticTreesStyle"
            }
          },
          {
            value: "Fraxinus pennsylvanica",
            symbol: {
              type: "web-style", // autocasts as new WebStyleSymbol()
              name: "Fraxinus",
              styleName: "EsriRealisticTreesStyle"
            }
          },
          {
            value: "Picea abies",
            symbol: {
              type: "web-style", // autocasts as new WebStyleSymbol()
              name: "Picea",
              styleName: "EsriRealisticTreesStyle"
            }
          },
                    {
            value: "Gleditsia triacanthos",
            symbol: {
              type: "web-style", // autocasts as new WebStyleSymbol()
              name: "Robinia",
              styleName: "EsriRealisticTreesStyle"
            }
          },
                              {
            value: "Fagus grandifolia",
            symbol: {
              type: "web-style", // autocasts as new WebStyleSymbol()
              name: "Fagus",
              styleName: "EsriRealisticTreesStyle"
            }
          },
                              {
            value: "Quercus macrocarpa",
            symbol: {
              type: "web-style", // autocasts as new WebStyleSymbol()
              name: "Quercus",
              styleName: "EsriRealisticTreesStyle"
            }
          },
                    {
            value: "Pinus strobus",
            symbol: {
              type: "web-style", // autocasts as new WebStyleSymbol()
              name: "Pinus",
              styleName: "EsriRealisticTreesStyle"
            }
          },
          {
            value: "Liriodendron tulipifera",
            symbol: {
              type: "web-style", // autocasts as new WebStyleSymbol()
              name: "Philodendron",
              styleName: "EsriRealisticTreesStyle"
            }
          }
        ];
```

Your Turn

Now that you have all the relevant code in place, you can save and open your HTML file. (It might take a moment to load your scene depending on your Internet connection speed). Your web application is focused on Hubbard Park in Iowa City, and the only data layer is the trees layer. You can easily add more feature layers by changing the

map.add(vegetationLayer);

Section to something like (let’s say a second layer is called “SecondLayer”) --

map.addMany([vegetationLayer, secondLayer]);

You just add additional layers by separating variables with commas, and putting them in brackets instead. Your task now is to make these changes and explain briefly HOW you did them for the points:

- Change the view by selecting a different camera height, viewing angle and orientation (direction). [10 points]

- Change all the currently-identified trees to obviously inappropriate vegetation species (e.g. palm, cactus, etc.) by purposefully misidentifying their “name” from the ArcGIS list (I’ve provided the link to the styles in the introduction of this assignment). [10 points]

- Change the shadow effects to correspond to a time of your choosing (as long as it is during daylight hours). [10 points]

- Add a second FeatureLayer from the same server location. You do NOT have to specify any renderer changes, just create the variable, point to the URL, make the requisite changes to the “add” section at the end, and bring in the layer as-is. (Notably, the bicycle path layer is in the starting view, so that one might be easiest). You may need to consult the documentation or online examples or previous labs to help you if you’ve forgotten how to add a FeatureLayer. [10 points]

Submitting Your Assignment

- Upload your finished document to GitHub. Then, send me a text file or word document through Canvas that includes the link, as well as your written answers.