ArcGIS API for JavaScript
Creating Custom Layers

Yann Cabon, Jacob Wasilkovski, Thomas Other
Agenda

- Introduction to layers in 4.x
- Building a custom dynamic layer
- Building a custom tile layer
- Layer loading and attribution
- Road-ahead: Custom 2D LayerViews
- A practical 3D example
Introduction to Layers
Introduction to Layers extensibility in 4.x

- They don’t have a visual representation, they are a data access
- Create a custom layer
  - To connect to a service not supported (or not yet) by the API
  - To work on the data client-side before it’s being displayed
  - To mash up multiple services to create new visualizations
- We will cover:
  - Dynamic layer
  - Tile
- 4.4 will introduce new layer classes designed for extensibility
Custom Dynamic Layers
Dynamic Layer

- displays an image that covers the view. At the end of a user interaction, a new image is requested.
  - 1 export in MapView
  - 2 exports in SceneView
- Pro: One export on the service
- Con: Export is different each time, so cannot be cached

Dynamic Layer

- **Extensibility hooks:**
  - `BaseDynamicLayer.getImageUrl`
    - Easiest method that creates a URL for a extent and size
  - `BaseDynamicLayer.fetchImage`
    - Method that do the actual image fetching
    - To extend if you need to transform data.
    - Default implementation fetch the image at the URL returned by `getImageUrl`
Custom Dynamic Layer
Consuming a WMS service
Custom Tile Layers
Tile Layer

- Displays adjacent images stitched together to cover the view.
- New tiles are requested as the user interacts with the view.

Pros:
- Efficient as tiles are cacheable resources on the server and web browser.
- Tiles appear as you pan and zoom.

Con:
- A tile resource has to be less dynamic to have an efficient layer.
Tile Layer

• Extensibility hooks:
  - `BaseTileLayer.getTileUrl`
    - Easiest method that creates a url for a tile id: level / row / col
  - `BaseTileLayer.fetchImage`
    - Method that do the actual image fetching
    - To extend if you need to transform data.
    - Default implementation fetch the image at the URL returned by `getTileUrl`

Custom Tile Layer

getTileUrl
Custom Tile Layer

fetchTile
loading and integration
Loading and integration

- **Layers are loadable resources**

- **Layers need to load information before being displayed**
  - Test if the service exists and validity
  - Properties of the layer?
    - Extent
    - Title
    - Attribution
Loadable Integration

BlendLayer – service mashup
More examples

Brick Layers
Future Work
Future Work

• Custom layers are both working in 2D and 3D
• Relatively easy to extend
• But have limitations
  – Suited for static content
  – No control over when it redraws
  – Limited to image formats
Future Work

• For 3D, doing more with `externalRenderers`
    - Session https://devsummit2018.schedule.esri.com/schedule/1096560065

• For 2D, working on custom layerviews base classes
  - LayerViews are responsible to call the layers’ APIs to get data and redraw
  - Provide a Canvas2D API to draw anything you want.
  - UC release – July
Custom 2D LayerView Prototype
Custom (non-tiled) layer: Canvas-Flowmap-Layer

- Flows between "origins" and "destinations".
- Bézier curves. Not geodesic lines in real space.
Custom (non-tiled) layer for JSAPI v3

Canvas-Flowmap-Layer

- extends 'esri/layers/GraphicsLayer' with:

```javascript
// inside a new Dojo module...
declare([GraphicsLayer], {/* properties and methods */});
```

- adds `<canvas>` element to very specific DOM position
- converts spatial coordinates to screen x,y position for canvas drawing
Custom (non-tiled) layer for JSAPI v3

Canvas-Flowmap-Layer

- also manages many map event listeners

```javascript
// when user finishes zooming or panning the map
on.pausable(this._map, 'extent-change', /* function to redraw the canvas */);

// when user begins zooming the map
on.pausable(this._map, 'zoom-start', /* function to clear the canvas */);

// when user is actively panning the map
on.pausable(this._map, 'pan', /* function to move the canvas */);

// when map is resized in the browser
on.pausable(this._map, 'resize', /* function to resize the canvas */);
```
Custom (non-tiled) layer for JSAPI v4

Canvas-Flowmap-Layer

(work in progress)
Custom (non-tiled) layer for JSAPI v4

Canvas-Flowmap-Layer

- extends 'esri/layers/Layer' and 'esri/views/2d/layers/BaseLayerView2D' with:

```javascript
// inside a new Dojo module...
var myCustomLayerView = BaseLayerView2D.createSubclass({/* properties and methods */});
var myCustomLayer = Layer.createSubclass({/* properties and methods */});
return myCustomLayer;
```

- converts spatial coordinates to screen x,y position for canvas drawing
Custom (non-tiled) layer for JSAPI v4
Canvas-Flowmap-Layer

- does not manually add `<canvas>` element to very specific DOM position
- does not manually manage many map event listeners
Topics

- Typescript
- 2D / 3D Viewing
- Tiled Layers
- Lerc Encoding
- Custom Avalanche Tile Layer
  - Extending TileLayer
  - Up-Sampling / Canvas drawing
  - Slope Calculation/Classification
- Building App / Demo
- Conclusion
Typescript

Type Safety for Large Code Bases

- Superset of JavaScript
- Type existing JavaScript
- Adds lots of syntactic sugar
var map = new Map({
    basemap: "streets",
    layers: [new FeatureLayer(
        "...Germany/FeatureServer/0"
    )]
});

viewLeft = new MapView({
    container: "viewDivLeft",
    map: map
});

viewRight = new SceneView({
    container: "viewDivRight",
    map: map
});
Tiled Layers

Tiling Schemes

- Span full globe with different levels of detail
- Quadratic tiles $256 \times 256$
- Width/Height prop. to $2^\text{Level}$
- Level can go down to 15 (1 mile), sometimes even to 22 (10 yards)
Custom Tile Layer

Let's create a custom avalanche layer that calculates slope angles on the fly.

- Use existing elevation service for sampling
- Classify slope angles according to incline:
  - Low danger: < 30°
  - Moderate danger: 30 - 35°
  - High danger: 35 - 40°
  - Very high danger: 40 - 45°
  - Always avoid: > 45°
- Display resulting image on top of terrain
LERC Encoding

Limited Error Raster Compression

- LERC is an open-source image or raster format
- User defined maximum compression error per pixel while encoding
- Elevation data published by Esri is LERC encoded
- JavaScript implementation available publicly
  - https://github.com/Esri/lerc/
Custom Tile Layer
Extending the Base Class

// esri.core.accessorSupport
import { declared, subclass, property } from "esri/core/accessorSupport/decorators";

// esri.layers
import BaseTileLayer = require("esri/layers/BaseTileLayer");

@subclass()
class CustomTileLayer extends declared(BaseTileLayer) {

    @property({ type: String })
    url: string = null;

    @property({ type: Boolean })
    shade: boolean = false;

    @property({ type: Number })
    shadeDirection: number = 1;

    // Generate the tile url for a given level, row and column
    public getTileUrl(level: number, row: number, col: number) { ... }

    // Fetches tiles for the specified level and size.
    public fetchTile(level: number, row: number, col: number) { ... }
}

export = CustomTileLayer;
Custom Tile Layer

Up Sampling

- Reduce memory / computational load
- Use tile information from a tile with a lower resolution (higher level)

```javascript
public fetchTile(level: number, row: number, col: number) {
    let uLevel = level > 2 ? level - 2 : 0;
    if (level > 16) {
        uLevel = 14;
    }
    const uRate = level - uLevel;
    const uRow = row >> uRate;
    const uCol = col >> uRate;
    const uFactor = 1 << uRate;
    // call getTileUrl() method to construct the URL to tiles
    // for a given level, row and col provided by the LayerView
    var url = this.getTileUrl(uLevel, uRow, uCol);
    return esriRequest(url, { responseType: "array-buffer" })
        .then(response => {
            ...
        });
}
```
public fetchTile(level: number, row: number, col: number) {
    const canvas = document.createElement("canvas");
    const context = canvas.getContext("2d");
    const width = Math.floor(this.tileInfo.size[0] / uFactor);
    const height = Math.floor(this.tileInfo.size[1] / uFactor);
    canvas.width = width;
    canvas.height = height;
    const imageData = context.createImageData(width, height);
    for (var i = 0; i < width * height; i++) {
        var channel = Math.round(Math.random() * 3);
        imageData.data[i * 4] = channel % 3 === 0 ? 255 : 0;
        imageData.data[i * 4 + 1] = channel % 3 === 1 ? 255 : 0;
        imageData.data[i * 4 + 2] = channel % 3 === 2 ? 255 : 0;
        imageData.data[i * 4 + 3] = 32;
    }
    context.putImageData(imageData, 0, 0);
    return canvas;
}
const lerc = LercDecode.decode(response.data, { noDataValue: 0 });

const sLeft = Math.floor((col - uCol * uFactor) * width);
const sTop = Math.floor((row - uRow * uFactor) * height);

const kernel = [ { x: -1, y: 0 }, { x: 1, y: 0 }, { x: 0, y: -1 }, { x: 0, y: 1 } ];

const lengthPerPixel = 40075000 / lerc.width / (1 << uLevel) / 2;

for (let i = 0; i < width * height; i++) {
  const x = i % width;
  const y = Math.floor(i / width);

  let sMax = -Infinity;
  let sMin = Infinity;

  for (const c of kernel) {
    const sample = lerc.pixels[sLeft + x + c.x + (sTop + y + c.y) * lerc.width];
    sMax = Math.max(sMax, sample);
    sMin = Math.min(sMin, sample);
  }

  const slope = Math.atan((sMax - sMin) / 3 / lengthPerPixel) / Math.PI * 180;

  data[i * 4] = slope * 10;
  data[i * 4 + 1] = data[i * 4 + 2] = 0;
  data[i * 4 + 3] = 128;
}
slope = Math.atan((sMax - sMin) / 3 / lengthPerPixel) / Math.PI * 180;

if (slope > 45) {
    data[i * 4] = 172;
    data[i * 4 + 1] = 16;
    data[i * 4 + 2] = 64;
} else if (slope > 40) {
    data[i * 4] = 255;
    data[i * 4 + 1] = 0;
    data[i * 4 + 2] = 0;
} else if (slope > 35) {
    data[i * 4] = 255;
    data[i * 4 + 1] = 128;
    data[i * 4 + 2] = 0;
} else if (slope > 30) {
    data[i * 4] = 255;
    data[i * 4 + 1] = 255;
    data[i * 4 + 2] = 0;
} else {
    data[i * 4] = data[i * 4 + 1] = data[i * 4 + 2] = 0;
}

data[i * 4 + 3] = slope < 30 ? 0 : 128;
for (let i = 0; i < width * height; i++) {
    const x = i % width;
    const y = Math.floor(i / width);

    let sMax = -Infinity;
    let sMin = Infinity;

    const center = lerc.pixels[sLeft + x + (sTop + y) * lerc.width];

    let shade = false;

    for (const c of kernel) {
        const sample = lerc.pixels[sLeft + x + c.x + (sTop + y + c.y) * lerc.width];

        sMax = Math.max(sMax, sample);
        sMin = Math.min(sMin, sample);

        if (sample - center < 0 && (c.x === 1)) {
            shade = true;
        }
    }

    data[i * 4] = 0;
    data[i * 4 + 1] = 0;
    data[i * 4 + 2] = 0;
    data[i * 4 + 3] = shade ? 32 : 0;
}
Custom Tile Layer

Building a Custom Application

```javascript
require(['
  './AvalancheTileLayer.js',
  'esri/Map',
  'esri/layers/SceneLayer',
  'esri/renderers/UniqueValueRenderer'
], function(
  AvalancheTileLayer,
  Map,
  SceneLayer
) {
  var avalancheTileLayer = new AvalancheTileLayer({
    url: '//elevation3d.arcgis.com/arcgis/rest/...',
    title: 'Avalanche Hazard Layer'
  });

  var map = new Map({
    basemap: 'topo',
    ground: 'world-elevation',
    layers: [ customTileLayer ]
  });

  var view = new SceneView({
    container: 'viewDiv',
    map: map,
  });
});
```
Custom Tile Layer

Conclusion

Possible improvements:
- Sample more points for slope calculation
- Sample across tile boundaries
- Detect highest available level

Possible enhancements:
- Add weather information
- Add snow layering information
Slides

Implementation