IconLayer

Edit on Codepen

The `IconLayer` renders raster icons at given coordinates.

There are two approaches to load icons. You can pre-generated a sprite image (`iconAtlas`), which packs all your icons into one layout, and a JSON descriptor (`iconMapping`), which describes the position and size of each icon in the `iconAtlas`. You can create sprite images with tools such as TexturePacker. This is the most efficient way to load icons.

It is also possible to ask `IconLayer` to generate `iconAtlas` dynamically. This is slower but might be useful in certain use cases.

Example: pre-packed iconAtlas

```
import DeckGL from '@deck.gl/react';
import {IconLayer} from '@deck.gl/layers';

const ICON_MAPPING = {
  marker: {x: 0, y: 0, width: 128, height: 128, mask: true}
};

function App({data, viewState}) {
  /**
   * Data format:
```

```
import DeckGL, {IconLayer} from 'deck.gl';
import {Octokit} from '@octokit/rest';
const octokit = new Octokit()

function App({data, viewState}) {
  /**
   * Data format:
   * [
   *   {
   *     avatar_url: "https://avatars1.githubusercontent.com/u/7025232?v=4",
   *     contributions: 620,
   *     id: 7025232,
   *     login: "ibgreen",
   *     type: "User",
   *     ...
   *   }
   * ]
   */
  const layer = new IconLayer({
    id: 'icon-layer',
    data: octokit.repos.getContributors({
      owner: 'visgl',
      repo: 'deck.gl'
    }).then(result => result.data),
    // iconAtlas and iconMapping should not be provided
    // getIcon return an object which contains url to fetch icon of each data
point
    getIcon: d => ({
      url: d.avatar_url,
      width: 128,
      height: 128,
      anchorY: 128
    }),
    // icon size is based on data point's contributions, between 2 - 25
    getSize: d => Math.max(2, Math.min(d.contributions / 1000 * 25, 25)),
    pickable: true,
    sizeScale: 15,
    getPosition: d => d.coordinates
  });

  return <DeckGL viewState={viewState}
    layers={[layer]}
    getTooltip={({object}) => object && `${object.login}\n$
{object.contributions}`} />;
}
```

Installation

To install the dependencies from NPM:

```
npm install deck.gl
# or
npm install @deck.gl/core @deck.gl/layers
```

```
import {IconLayer} from '@deck.gl/layers';
new IconLayer({});
```

To use pre-bundled scripts:

```
<script src="https://unpkg.com/deck.gl@^8.0.0/dist.min.js"></script>
<!-- or -->
<script src="https://unpkg.com/@deck.gl/core@^8.0.0/dist.min.js"></script>
<script src="https://unpkg.com/@deck.gl/layers@^8.0.0/dist.min.js"></script>
```

```
new deck.IconLayer({});
```

Properties

Inherits from all Base Layer properties.

`iconAtlas` (String|Texture2D|Image|ImageData|HTMLCanvasElement|HTMLVideoElement|ImageBitmap|Promise|Object, optional)

A pre-packed image that contains all icons.

- If a string is supplied, it is interpreted as a URL or a Data URL.
- One of the following, or a Promise that resolves to one of the following:
  - One of the valid pixel sources for WebGL texture
  - A luma.gl Texture2D instance
  - A plain object that can be passed to the `Texture2D` constructor, e.g. `{width: <number>, height: <number>, data: <Uint8Array>}`. Note that whenever this object shallowly changes, a new texture will be created.

The image data will be converted to a Texture2D object. See `textureParameters` prop for advanced customization.

If you go with pre-packed strategy, this prop is required.

If you choose to use auto packing, this prop should be left empty.

`iconMapping` (Object|String, optional)

Icon names mapped to icon definitions, or a URL to load such mapping from a JSON file. Each icon is defined with the following values:

- `x` (Number, required): x position of icon on the atlas image
- `y` (Number, required): y position of icon on the atlas image
- `width` (Number, required): width of icon on the atlas image
- `height` (Number, required): height of icon on the atlas image
- `anchorX` (Number, optional): horizontal position of icon anchor. Default: half width.
- `anchorY` (Number, optional): vertical position of icon anchor. Default: half height.
- `mask` (Boolean, optional): whether icon is treated as a transparency mask. If `true`, user defined color is applied. If `false`, pixel color from the image is applied. User still can specify the opacity through getColor. Default: `false`

If you go with pre-packed strategy, this prop is required.

If you choose to use auto packing, this prop should be left empty.

`sizeScale` (Number, optional) transition enabled

- Default: `1`

Icon size multiplier.

`sizeUnits` **(String, optional)**

- Default: `pixels`

The units of the size, one of `'meters'`, `'common'`, and `'pixels'`. See unit system.

`sizeMinPixels` **(Number, optional)** transition enabled

- Default: `0`

The minimum size in pixels. When using non-pixel `sizeUnits`, this prop can be used to prevent the icon from getting too small when zoomed out.

`sizeMaxPixels` **(Number, optional)** transition enabled

- Default: `Number.MAX_SAFE_INTEGER`

The maximum size in pixels. When using non-pixel `sizeUnits`, this prop can be used to prevent the icon from getting too big when zoomed in.

`billboard` **(Boolean, optional)**

- Default: `true`

If `true`, the icon always faces camera. Otherwise the icon faces up (z).

`alphaCutoff` **(Number, optional)**

- Default: `0.05`

Discard pixels whose opacity is below this threshold. A discarded pixel would create a "hole" in the icon that is not considered part of the object. This is useful for customizing picking behavior, e.g. setting `alphaCutoff: 0, autoHighlight` will highlight an object whenever the cursor moves into its bounding box, instead of over the visible pixels.

`loadOptions` **(Object, optional)**

On top of the default options, also accepts options for the following loaders:

- ImageLoader if the `iconAtlas` prop is an URL, or if `getIcon` returns URLs for auto-packing

`textureParameters` **(Object)**

Customize the texture parameters.

If not specified, the layer uses the following defaults to create a linearly smoothed texture from `iconAtlas`:

```
{
  [GL.TEXTURE_MIN_FILTER]: GL.LINEAR_MIPMAP_LINEAR,
  [GL.TEXTURE_MAG_FILTER]: GL.LINEAR,
  [GL.TEXTURE_WRAP_S]: GL.CLAMP_TO_EDGE,
  [GL.TEXTURE_WRAP_T]: GL.CLAMP_TO_EDGE
}
```

Data Accessors

`getIcon` (Function, optional)

- Default: `d => d.icon`

Method called to retrieve the icon name of each object, returns string or object.

If you go with pre-packed strategy, then `getIcon` should return a string representing name of the icon, used to retrieve icon definition from given `iconMapping`.

If you choose to use auto packing, then `getIcon` should return an object which contains the following properties.

- `url` (String, required): url to fetch the icon
- `height` (Number, required): max height of icon
- `width` (Number, required): max width of icon
- `id`: (String, optional): unique identifier of the icon, fall back to `url` if not specified
- `anchorX`, `anchorY`, `mask` are the same as mentioned in `iconMapping`

`IconLayer` uses `id` (fallback to `url`) to dedupe icons. For icons with the same id, even if their sizes differ, `IconLayer` will only define one icon according to the first occurrence and ignore the rest of them. Vice versa, for icons with different ids, even if `url`s are the same, the image will be fetched again to create a new definition with different size, anchor, etc.

The image loaded from `url` is always resized to fit the box defined by `[width, height]` while preserving its aspect ratio.

`getPosition` (Function, optional) transition enabled

- Default: `d => d.position`

Method called to retrieve the position of each object, returns `[lng, lat, z]`.

`getSize` (Function|Number, optional) transition enabled

- Default: `1`

The height of each object, in units specified by `sizeUnits` (default pixels).

- If a number is provided, it is used as the size for all objects.
- If a function is provided, it is called on each object to retrieve its size.

`getColor` (Function|Array, optional) transition enabled

- Default: `[0, 0, 0, 255]`

The rgba color is in the format of `[r, g, b, [a]]`. Each channel is a number between 0-255 and `a` is 255 if not supplied.

- If an array is provided, it is used as the color for all objects.
- If a function is provided, it is called on each object to retrieve its color.
- If `mask` = false, only the alpha component will be used to control the opacity of the icon.

`getAngle` (Function|Number, optional) transition enabled

- Default: `0`

The rotating angle of each object, in degrees.

- If a number is provided, it is used as the angle for all objects.
- If a function is provided, it is called on each object to retrieve its angle.

`getPixelOffset` (Function|Array, optional) transition enabled

- Default: `[0, 0]`

Screen space offset relative to the `coordinates` in pixel unit.

- If an array is provided, it is used as the offset for all objects.
- If a function is provided, it is called on each object to retrieve its offset.

Callbacks

`onIconError` (Function)

- Default: `null`

Only used when using auto-packing. If the attempt to fetch an icon returned by `getIcon` fails, this callback is called with the following arguments:

- `event` (Object)
  - `url` (String) - the URL that was trying to fetch
  - `loadOptions` (Object) - the load options used for the fetch
  - `source` (Object) - the original data object that requested this icon
  - `sourceIndex` (Object) - the index of the original data object that requested this icon
  - `error` (Error)

Use binary attributes

This section is about the special requirements when supplying attributes directly to an `IconLayer`.

If `data.attributes.getIcon` is supplied, since its value can only be a typed array, `iconMapping` can only use integers as keys.

Source

modules/layers/src/icon-layer

Edit this page