Viewport

> Read the article detailing deck.gl's Views and Projections system.

A deck.gl `Viewport` is essentially a geospatially enabled camera, and combines a number of responsibilities, which can project and unproject 3D coordinates to the screen.

`Viewport` classes are focused on mathematical operations such as coordinate projection/unprojection, and calculation of `view` and `projection` matrices and other uniforms needed by the WebGL vertex shaders. The basic `Viewport` class is a generic geospatially enabled version of the typical 3D "camera" class you would find in most 3D/WebGL/OpenGL library.

While the `Viewport` class can certainly be used directly if you need and are able to calculate your own projection matrices, you typically do not directly create `Viewport` instances. Instead, `Viewport` classes are created using the View class descriptors and the current `viewState`.

Overview of Viewport Classes

Viewport Class	Description
`Viewport`	The base viewport has to be supplied view and projection matrices. It is typically only instantiated directly if the application needs to work with viewports that have been supplied from external sources, such as the `WebVR` API.
`WebMercatorViewport`	While all `Viewport` subclasses are geospatially enabled, this class renders from a perspective that matches a typical top-down map and is designed to synchronize perfectly with a mapbox-gl base map (even in 3D enabled perspective mode).

Usage

The `Viewport` class is normally not instantiated directly. The `View` class is more commonly

used by applications. deck.gl automatically creates `Viewport`s from `View`s and `viewState` when needed, using the `View.makeViewport` method.

Constructor

```
new Viewport({width: 500, height: 500, viewMatrix, projectionMatrix, ...});
```

General Parameters

- `x` (`Number`, optional) - x position of viewport top-left corner. Default `0`.
- `y` (`Number`, optional) - y position of viewport top-left corner. Default `0`.
- `width` (`Number`) - Width of viewport. Default `1`.
- `height` (`Number`) - Height of viewport. Default `1`.

View Matrix Parameters

- `viewMatrix` (`Array[16]`, optional) - 4x4 view matrix. Defaults to the identity matrix.

Position and Geospatial Anchor Options (Optional)

- `latitude` (`Number`, optional) - Center of viewport on map (alternative to center). Must be provided if constructing a geospatial viewport.
- `longitude` (`Number`, optional) - Center of viewport on map (alternative to center). Must be provided if constructing a geospatial viewport.
- `zoom` (`Number`, optional) - zoom level .
- `focalDistance` (`Number`, optional) - modifier of viewport scale if `zoom` is not supplied. Corresponds to the number of pixels per meter. Default to `1`.
- `position` (`Array[3]`, optional) - Position of viewport camera. Default `[0, 0, 0]`.
- `modelMatrix` (`Array[16]`, optional) - Optional 4x4 model matrix applied to position.

Projection Matrix Parameters.

- `projectionMatrix` (`Array[16]`, optional) - 4x4 projection matrix.

If `projectionMatrix` is not supplied, an attempt is made to build from the remaining parameters. Otherwise the remaining parameters will be ignored.

- `fovy` (`Number`, optional) - Field of view covered by camera, in the perspective case. In degrees. Default `75`.
- `near` (`Number`, optional) - Distance of near clipping plane. Default `0.1`. (Note that in geospatial viewports, this actual distance used is scaled by the height of the screen).
- `far` (`Number`, optional) - Distance of far clipping plane. Default `1000`. (Note that in geospatial viewports, this actual distance used is scaled by the height of the screen).
- `orthographic` (`Boolean`, optional) - whether to create an orthographic or perspective projection matrix. Default `false` (perspective projection).

Methods

`equals`

Parameters:

- `viewport` (Viewport) - The viewport to compare with.

Returns:

- `true` if the given viewport is identical to the current one.

`project`

Projects world coordinates to pixel coordinates on screen.

Parameters:

- `coordinates` (Array) - `[X, Y, Z]` in world units. `Z` is default to `0` if not supplied.
- `opts` (Object)
  - `topLeft` (Boolean, optional) - Whether projected coords are top left. Default to `true`.

Returns:

- `[x, y]` or `[x, y, z]` in pixels coordinates. `z` is pixel depth.
  - If input is `[X, Y]`: returns `[x, y]`.

    - If input is `[X, Y, Z]`: returns `[x, y, z]`.

`unproject`

Unproject pixel coordinates on screen into world coordinates.

Parameters:

- `pixels` (Array) - `[x, y, z]` in pixel coordinates. Passing a `z` is optional.
- `opts` (Object)
    - `topLeft` (Boolean, optional) - Whether projected coords are top left. Default to `true`.
    - `targetZ` (Number, optional) - If pixel depth `z` is not specified in `pixels`, this is used as the elevation plane to unproject onto. Default `0`.

Returns:

- `[X, Y]` or `[X, Y, Z]` in world coordinates.
    - If input is `[x, y]` without specifying `opts.targetZ`: returns `[X, Y]`.
    - If input is `[x, y]` with `opts.targetZ`: returns `[X, Y, targetZ]`.
    - If input is `[x, y, z]`: returns `[X, Y, Z]`.

`projectPosition`

Projects latitude, longitude (and altitude) to coordinates in the common space.

Parameters:

- `coordinates` (Array) - `[lng, lat, altitude]` Passing an altitude is optional.

Returns:

- `[x, y, z]` in WebMercator coordinates.

`unprojectPosition`

Projects a coordinate from the common space to latitude, longitude and altitude.

Parameters:

- `coordinates` (Array) - `[x, y, z]` in the WebMercator world. `z` is optional.

Returns:

- `[longitude, latitude, altitude]`

`getBounds`

Extracts the axis-aligned bounding box of the current visible area.

- `options` (Object, optional)
  - `options.z` (Number, optional) - To calculate a bounding volume for fetching 3D data, this option can be used to get the bounding box at a specific elevation. Default `0`.

Returns:

- `[minX, minY, maxX, maxY]` that defines the smallest orthogonal bounds that encompasses the visible region.

`getFrustumPlanes`

Extract view frustum planes of the current camera. Each plane is defined by its normal `normal` and distance from the origin `distance` (such that point `x` is on the plane if `dot(normal, x) === distance`) in the common space.

Returns:

- `{near: {normal, distance}, far: {normal, distance}, left: {normal, distance}, right: {normal, distance}, top: {normal, distance}, bottom: {normal, distance}}`

```
import {Vector3} from '@math.gl/core';

// Culling tests must be done in common space
const commonPosition = new Vector3(viewport.projectPosition(point));

// Extract frustum planes based on current view.
const frustumPlanes = viewport.getFrustumPlanes();
let outDir = null;

// Check position against each plane
for (const dir in frustumPlanes) {
  const plane = frustumPlanes[dir];
  if (commonPosition.dot(plane.normal) > plane.distance) {
    outDir = dir;
```

```
      break;
    }
  }
  if (outDir) {
    console.log(`Point is outside of the ${outDir} plane`);
  } else {
    console.log('Point is visible');
  }
```

Remarks

- The `Viewport` class and its subclasses are perhaps best thought of as geospatially enabled counterparts of the typical `Camera` classes found in most 3D libraries.
- The `Viewport` class works together with the `project` shader module and generates the uniforms that module needs to project correctly in GLSL code.
- Accordingly, a main function of viewports is to generate WebGL compatible view and projection matrices (column-major format).
- Functions (including projection and unprojection of coordinates) are available both in JavaScript and in GLSL, so that layers can do consistent projection calculations in both GLSL and JavaScript.
- To support pixel project/unproject functions (in addition to the clipspace projection that Camera classes typically manage), the `Viewport` is also aware of the viewport extents.
- In geospatial setups, Viewports can contain geospatial anchors.

Source

modules/core/src/viewports/viewport.ts

Edit this page