Analysis and interactive visualization of neutrino event topologies registered in the OPERA experiment

Sergey Dmitrievsky

Dzhelepov Laboratory of Nuclear Problems

JINR, Dubna, Russia

Task 3

INTEREST @JINR, Wave 9, Project #253

Task 3

3) In this task OPERA emulsion dataset for the tau neutrino appearance studies from the Open Data Portal will be used. A simplified version of a browser based 3D event display that uses the THREE.js graphics library will be provided with missing parts of the source code. It will be suggested to recover the code in order to display tracks and vertices reconstructed in nuclear emulsions in the 10 tau neutrino candidate events.

OPERA papers about the full ν_τ event sample:

[PRL 120, 211801 (2018)]

[Sci Data 8, 218 (2021)]

All 10 ν_τ-candidate events are available on the Open Data Portal (ODP).

OPERA browser-based event display:

Web application on the Open Data Portal

Source code on GitHub

Our JS project for the task 3 is just a simplified version of the OPERA event display application from the ODP.

From *.csv to *.js files

In this task information about interaction vertices and particle tracks is used as well. It is already extracted from the *.csv files and saved to *.js files (to JavaScript structures):

From the **10123059807_Vertices.csv** file

```
posX,posY,posZ,globPosX,globPosY,globPosZ,primary
112653,79333.3,24057,-219.336,121.595,192.288,1
112640,79344.7,24196.4,-219.337,121.596,192.301,0
111359,78992,34618,-219.465,121.561,193.344,0
100040,86241.7,47861.3,-220.597,122.286,194.668,0
```

From the **10123059807_Lines.csv** file

```
trType,posX1,posY1,posZ1,posX2,posY2,posZ2
8,112653,79333.3,24057,112640,79344.7,24196.4
2,112653,79333.3,24057,111168,78975.8,25883.7
2,111168,78975.8,25883.7,109962,78663.4,27396.4
2,109962,78663.4,27396.4,108911,78360.4,28694.1
2,108911,78360.4,28694.1,107877,78065.5,29980.8
2,107877,78065.5,29980.8,106824,77783.2,31278.3
2,106824,77783.2,31278.3,105764,77505.1,32578.3
```

From the **loadEvent10123059807.js** file

```
display3d.resetEvent();

display3d.event().id(10123059807);
display3d.event().date(1272871069000);

display3d.event().vertices3D([
  new Vertex([112653, 79333.3, 24057],   [-219.336, 121.595, 192.288]),
  new Vertex([112640, 79344.7, 24196.4], [-219.337, 121.596, 192.301]),
  new Vertex([111359, 78992, 34618],     [-219.465, 121.561, 193.344]),
  new Vertex([100040, 86241.7, 47861.3], [-220.597, 122.286, 194.668])
]);

display3d.event().tracks3D([
  new Track3D(0, 8,    [112653, 79333.3, 24057], [1000000., 1000000.], [112640, 79344.7, 24196.4]),
  new Track3D(1, 17,   [112653, 79333.3, 24057], [1000000., 1000000.], [111168, 78975.8, 25883.7]),
  new Track3D(2, 17,   [111168, 78975.8, 25883.7], [1000000., 1000000.], [109962, 78663.4, 27396.4]),
  new Track3D(3, 17,   [109962, 78663.4, 27396.4], [1000000., 1000000.], [108911, 78360.4, 28694.1]),
  new Track3D(4, 17,   [108911, 78360.4, 28694.1], [1000000., 1000000.], [107877, 78065.5, 29980.8]),
  new Track3D(5, 17,   [107877, 78065.5, 29980.8], [1000000., 1000000.], [106824, 77783.2, 31278.3]),
  new Track3D(6, 17,   [106824, 77783.2, 31278.3], [1000000., 1000000.], [105764, 77505.1, 32578.3])
```

From the Task 3 index.html file

External js-libraries

Internal js-files

```
<head>

  <title>Browser based 3D event display</title>

  <meta charset="utf-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <link rel="stylesheet" href="css/display3d.css">

  <script src="js/lib/jquery.min.js"></script>
  <script src="js/lib/three.min.js"></script>
  <script src="js/lib/three3DExtras.min.js"></script>
  <script src="js/lib/TrackballControls.js"></script>

  <script src="js/Vertex-def.js"></script>
  <script src="js/Track3D-def.js"></script>
  <script src="js/Event-def.js"></script>
  <script src="js/MgrDraw3D-def.js"></script>
  <script src="js/Display3d-def.js"></script>
  <script src="js/init.js"></script>
  <script src="js/Display3d-fills.js"></script>
  <script src="js/MgrDraw3D-funcAdd.js"></script>

</head>
```

The THREE.js library and extra classes for it.

The modern versions of these libraries require their installation (e.g., with help of the npm manager). In our project we will use an older versions that can be just included in the HTML file!

From the Task 3 index.html file

The toolbar with buttons, etc.

The event display canvas

The first script to be loaded

```
<body id="body-display3d" data-event-id="9190097972">

  <div class="viewer" id="div-viewer">

    <div id="div-col-display-3D">

      <div id="div-toolbar-3D" class="toolbar">

        <label for="input-event-3D">Event:</label>
        <input id="input-event-3D" size="11">

        <div class="btn-group" role="group">
          <button type="button" id="btn-ecc-prev-event" onclick="display3d.loadPrevOrNextEvent(-1);"
                  title="Previous event">
            <img src="graphics/event-prev.png"
                 class="img-responsive"/>
          </button>
          <button type="button" id="btn-ecc-next-event" onclick="display3d.loadPrevOrNextEvent(1);"
                  title="Next event">
            <img src="graphics/event-next.png"
                 class="img-responsive"/>
          </button>
        </div>

      </div>  <!-- "div-toolbar-3D" -->

      <div class="row ecc-display">
        <div id="div-canvas-3D" style="position: relative;">
          <span id="span-canvas-3D-title" class="canvas-title"></span>
          <canvas id="canvas-3D" class="canvas-bordered-white"></canvas>
        </div> <!-- "div-canvas-3D" -->
      </div>

    </div> <!-- "div-col-display-3D" -->

  </div> <!-- "div-viewer" -->

  <div id="div-ScrLoadEvent">
    <script id="script-LoadEvent" src="js/loadFirstEvent.js"> </script>
  </div>

</body>
```

Vertex-def.js

The global system of reference will not be used by us in this task!

```
class Vertex {
  constructor(pos, posGlob) {
    this._pos = pos;          // [posX, posY, posZ] Position in the OPERA brick system of reference (in micrometers)
    this._posGlob = posGlob; // [posGlobX, posGlobY, posGlobZ] Position in the OPERA detector system of reference (in cm)
  };
  pos(ps) {
    if (ps === undefined) return this._pos;
    this._pos = ps;
  };
  posGlob(ps) {
    if (ps === undefined) return this._posGlob;
    this._posGlob = ps;
  };
  static colorFor3D() { return "gold"; };
};
```

Track3D-def.js

The track slopes will not be used by us in this task!

```
class Track3D {

  constructor(id, partId, pos1, slopes, pos2) {

    this._id = id;

    this._partId = partId;   // particle Id

    this._pos1 = pos1;       // [posX, posY, posZ] Position of the first track point
                             // in the OPERA brick system of reference (in micrometers)

    this._pos2 = pos2;       // [posX, posY, posZ] Position of the second track point
                             // in the OPERA brick system of reference (in micrometers)
                             // This point is used only if slopes[0]==slopes[1]==1000000

    // Equations of a track:
    // X = Z*Axy[0] + Bxy[0], Y = Z*Axy[1] + Bxy[1]

    this._Axy = slopes; // [slopeXZ, slopeYZ] --- tangents of the track angles
  };

  id() { return this._id; };

  partId() { return this._partId; };

  pos1(ps) {

    if (ps === undefined) return this._pos1;

    this._pos1 = ps;

  };

  pos2(ps) {

    if (ps === undefined) return this._pos2;

    this._pos2 = ps;

  };

  Axy(ip) {

    if (ip === undefined) return this._Axy;

    return this._Axy[ip];

  };
```

Track colors for different particles

```
static colors(partId) {

  switch (partId) {

    case  1: return "dodgerblue";  // for tracks of muons
    case  2: return "#FF1111";     // for tracks of hadrons
    case  3: return "yellow";      // for tracks of e+/e-
    case  4: return "white";       // for highly ionizing tracks
    case  5: return "white";       // for back highly ionizing tracks
    case  6: return "springgreen"; // for ionizing tracks
    case  7: return "springgreen"; // for back ionizing tracks
    case  8: return "#FF1111";     // for tracks of tau leptons or charm particles
    case  9: return "aqua";          // for tracks of hadrons
    case 10: return "limegreen";     // for tracks of hadrons
    case 11: return "dodgerblue";    // for tracks of hadrons
    case 12: return "magenta";       // for tracks of hadrons
    case 13: return "lawngreen";     // for tracks of hadrons
    case 14: return "white";         // for tracks of hadrons
    case 15: return "gray";          // for tracks of hadrons
    case 16: return "orange";        // for tracks of hadrons or e+/e-
    case 17: return "yellow";        // for tracks of hadrons or e+/e-
    case 18: return "deeppink";      // for tracks e+/e-
    case 19: return "antiquewhite"; // for tracks e+/e-
    case 20: return "#FF1111";       // for tracks of hadrons

    default: return "black";      // for other tracks (not used!)
  }
```

Event-def.js

```
class Event { // OPERA event object will contain experimental data to be displayed in the main window
  constructor() {
    this._id = 0;              // OPERA event id
    this._date = {};           // The OPERA event time (millisecons since 01.01.1970)
    this._vertices3D = [];     // Array of vertices. [0] - the primary neutrino interaction vertex!
    this._tracks3D = [];       // Array of tracks.
  };
  id(jd) {
    if (jd === undefined) return this._id;
    this._id = jd;
  };
  date(ts) {
    if (ts === undefined) return this._date;
    this._date = new Date(ts);
  };
  vertices3D(vertices) {
    if (vertices === undefined) return this._vertices3D;
    this._vertices3D = vertices;
  };
  tracks3D(tracks) {
    if (tracks === undefined) return this._tracks3D;
    this._tracks3D = tracks;
  };
};
```

From the Display3d-def.js file

```
class Display3d { // Browser-based 3D event display

  constructor() {

    this._evListNuTau = [];   // Array of OPERA nu_tau event IDs

    this._evIndex    = -1;    // Index of the loaded event in the array (from 0 to evList.length - 1)
    this._evIndexMax = -1;    // Max index of the loaded event in the array (=== evList.length - 1)
    this._event      = {};    // Loaded (displayed) event

    this._mgrDraw3D = {};    // Manager hired for drawing of (3D) tracks found in emulsion

  };

  evList(evsample, evlist) {

    if (evlist === undefined) return this._evListNuTau;

    this._evListNuTau = evlist;

    this._evIndex = 0;
    this._evIndexMax = evlist.length - 1;

  };

  evIndex(evindex) {

    if (evindex === undefined) return this._evIndex;

    this._evIndex = evindex;

  };

  evIndexMax(evindexmax) {

    if (evindexmax === undefined) return this._evIndexMax;

    this._evIndexMax = evindexmax;

  };

  event(ev) {

    if (ev === undefined) return this._event;

    this._event = ev;

  };

  resetEvent() { this._event = new Event(); };
```

Display3d-fills.js

Event IDs of the 10 ν_τ-candidate events

```
display3d.evList(1, [
9190097972,
9234119599,
10123059807,
11113019758,
11143018505,
11172035775,
11213015702,
12123032048,
12227007334,
12254000036
]);
```

From the MgrDraw3D-def.js file

```
class MgrDraw3D { // Manager intended for drawing of vertices and (3D) tracks found in the emulsion

  constructor() {

    this._camera     = null;

    this._scene      = null;

    this._renderer   = null;    // Draws the scene on the screen (as it is seen by the camera)

    this._tbControls = null;    // Trackball controls can be used to pan and move the camera around

    this._view = 1;             // 0 - XZ, 1 - YZ, and 2 - XY

    //---

    this._primVertDrawPos = new THREE.Vector3(0, 0, 0); // Position of the primary vertex.
                                                        // The code of drawing functions should be properly
                                                        // modified in case change of this position is needed!!!

    this._cameraInitPositions = [];                     // Initialyzed in the initCameras() function!

    this._cameraInitUpDirs = [new THREE.Vector3(1, 0, 0),   // The 'up' directions of the cameras for XZ,
                              new THREE.Vector3(0, 1, 0),   //                                         YZ,
                              new THREE.Vector3(0, 1, 0)];  //                                         and XY views

    this._vertexGeometry = {};

    this._vertexMaterial = {};

    this._groupOfVertices = {};   // three.js group of vertex points

    this._trackLinePars = [];     // Array of line parameters used for drawing of the 3D tracks

    this._groupOfTracks = {};     // three.js group of track lines

  };
```

We will draw the primary vertex of each neutrino event always in the same point of the screen!

The absolute coordinates of all tracks and vertices have to be recalculated properly with respect to this reference point!

From the MgrDraw3D-funcAdd.js file: Camera initialization

```
//dm3D == display3d.mgrDraw3D() !!!
//-------------------------------------

dm3D.initGraphics = function() {

  dm3D.initCamera();

  dm3D.scene( new THREE.Scene() );

  dm3D.initRenderer();

  dm3D.initControls();

  dm3D.groupOfVertices( new THREE.Group() );

  dm3D.groupOfTracks( new THREE.Group() );

  dm3D.initVertexProperties();

  dm3D.initTrackLineProperties();

};
//-------------------------------------------------------------------------------

dm3D.initCamera = function() {

  const primVertDrawPos = dm3D.primVertDrawPos();

  dm3D.cameraInitPositions([
    new THREE.Vector3(primVertDrawPos.x - 200,
                      primVertDrawPos.y + 2000,
                      primVertDrawPos.z + 200),

    new THREE.Vector3(primVertDrawPos.x - 2000,
                      primVertDrawPos.y + 200,
                      primVertDrawPos.z + 200),

    new THREE.Vector3(primVertDrawPos.x - 200,
                      primVertDrawPos.y + 200,
                      primVertDrawPos.z + 7000)
  ]);

  dm3D.camera( new THREE.OrthographicCamera(primVertDrawPos.z - 1800,
                                            primVertDrawPos.z + 4200,
                                            primVertDrawPos.y + 1854,
                                            primVertDrawPos.y - 1854,
                                            primVertDrawPos.x - 50000,
                                            primVertDrawPos.x + 150000) );
};
```

From the MgrDraw3D-funcAdd.js file

Initialization of vertex and track properties

```
dm3D.initVertexProperties = function( ) {

  dm3D.vertexGeometry( new THREE.SphereGeometry(20, 32, 32) );

  dm3D.vertexMaterial( new THREE.MeshBasicMaterial({ color: Vertex.colorFor3D( ) }) );

};
//-----------------------------------------------------------------------------

dm3D.initTrackLineProperties = function( ) {

  dm3D.trackLinePars( )[1] = { // for a muon track

    color:  Track3D.colors(1),
    length: 10*1300,
    width:  12

  };

  dm3D.trackLinePars( )[2] = { // for an hadron track

    color:  Track3D.colors(2),
    length: 10*1300,
    width:  12

  };

  dm3D.trackLinePars( )[3] = { // for an electron track

    color:  Track3D.colors(3),
    length: 3*1300,
    width:  12

  };
```

From the MgrDraw3D-funcAdd.js file

Drawing the vertices

```
dm3D.drawVertices = function() {

  const evVertices = display3d.event().vertices3D();

  const primVertRealPos = evVertices[0].pos();

  const primVertDrawPos = dm3D.primVertDrawPos();

  const nbOfVertices = evVertices.length;

  for (let iv = 0; iv < nbOfVertices; iv++) {

    const vertexPoint = new THREE.Mesh( dm3D.vertexGeometry(), dm3D.vertexMaterial() );

    vertexPoint.position.x = // Relative vertex position
    vertexPoint.position.y = // with respect to
    vertexPoint.position.z = // the primary interaction vertex!

    dm3D.groupOfVertices().add(vertexPoint);

  }

  dm3D.scene().add(dm3D.groupOfVertices());

};
```

Please try to put your code here to calculate the three coordinates of the relative vertex position!

Drawing the vertices

Coordinates in the (absolute) system of reference of the OPERA brick

$V_i\ (x_i, y_i, z_i)$

$V_0\ (x_0, y_0, z_0)$

Coordinates in the (relative) system of reference of the 3D display

$V_i\ (dx_{i0}, dy_{i0}, dz_{i0})$

$V_0\ (0, 0, 0)$

$$dx_{i0} \equiv x_i - x_0,$$
$$dy_{i0} \equiv y_i - y_0,$$
$$dz_{i0} \equiv z_i - z_0.$$

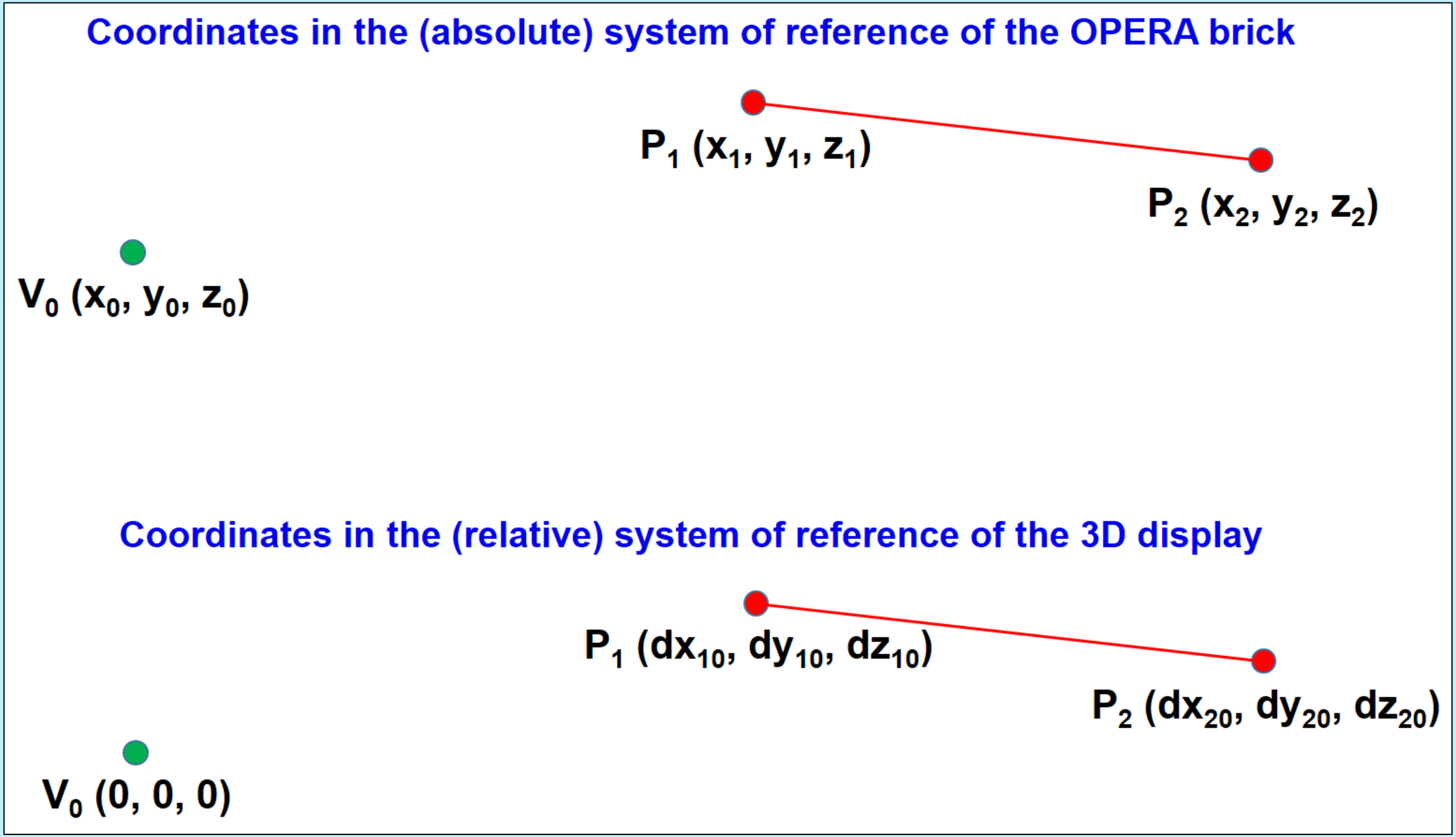
From the MgrDraw3D-funcAdd.js file

Drawing the tracks

```
dm3D.drawTracks = function() {

  const primVertRealPos = display3d.event().vertices3D()[0].pos();

  const primVertDrawPos = dm3D.primVertDrawPos();

  const evTracks = display3d.event().tracks3D();

  const nbOfTracks = evTracks.length;

  for (let it = 0; it < nbOfTracks; it++) {

    const iTrack = evTracks[it];

    const trPos1 = [0, 0, 0]; // Just a 1d array initialization
    const trPos2 = [0, 0, 0]; // Just a 1d array initialization

    for (let ip = 0; ip < 3; ip++) {

      trPos1[ip] = // Relative track positions with respect to
      trPos2[ip] = // the primary interaction vertex

    }

    const trPars = dm3D.trackLinePars()[iTrack.partId()];

    const trackLine = new three3DExtras.tubeLine(trPos1, trPos2, trPars.width, trPars.color);

    dm3D.groupOfTracks().add(trackLine.getObject3D());

  }

  dm3D.scene().add(dm3D.groupOfTracks());

};
```

Please try to put your code here to calculate the three coordinates of the relative trPos1 and trPos2 positions!

Drawing the tracks



Backup slides

Useful links

HTML/CSS/JavaScript/… tutorials:
https://www.w3schools.com/html/default.asp

Documentation for Web developers from the MDN Web Docs:
https://developer.mozilla.org/en-US/docs/Web

THREE.js official web-site:
https://threejs.org/

Example of using the Trackball controls:
https://threejs.org/docs/#examples/en/controls/TrackballControls

Three3DExtras package on GitHub:
https://github.com/jjcc1421/Three3DExtras

How to open the developer tools in the Google Chrome browser