

BES500831

## **Unlocked potential with Point Clouds and Forge**

### **Learning Objectives**

1. How is CAD vs Scan used in the industry and how to capture a simple point cloud
2. Learn how to visualize point clouds in Forge Viewer with alignment with BIM / CAD model
3. Learn how to download and view 3D-Tiles in many 3D-Engine
4. Learn how to merge points cloud in a Revit file by using Forge Design Automation

### **Description**

Point cloud is now a technology well known in AEC industry including scanning and photogrammetry. With points cloud, you could save a lot of time in various use cases like digital twins, as-build comparing or retro-engineering. Collecting data from the reality to virtual environment is always time consuming and painful. Discover in a smooth workflow, how to merge points cloud on Forge platform with BIM models to improve collaboration and save time.

## Speaker(s)

Alexandre Piro - Piro CIE



Alexandre Piro studied mechanical engineering and manufacturing. Graduated in 2012, he started to work in aeronautics on tooling and efficiency improvement. In 2016, he began Augmented Reality development with CAD models and after 6 months he joined Piro CIE. He spent one year of research and POC developments on Augmented and Virtual Reality. Since 2018, he leads AR/VR projects and web workflows to be more flexible with data exchange. He also started to work on Autodesk Forge to explore another aspect of CAD and 3D manipulation outside of editors. He is regularly working with the Forge Dev Team and participated in 3 Forge Accelerators. He is able to develop on different parts of the Forge APIs.

Brian Nickel - Allied BIM



Fresh out of architecture school, Brian Nickel began designing plumbing systems for mega homes in the resort town of Big Sky, Montana. He found that the processes for turning 3D models into 40,000-square foot mansions were fraught with inefficiencies, especially for the parts of homes that are prefabricated like plumbing and HVAC. Today, he is co-founder of Allied BIM, a software company focused on enhancing and revolutionizing the construction industry.

Michael Beale - Autodesk



Michael Beale is a senior software engineer at Autodesk, where he has been a globe-trotting technical advocate for Forge. Michael focuses on connecting Autodesk cloud data and Autodesk Forge APIs to the browser. He's also contributed to Autodesk Homestyler, the Forge Large Model Viewer (LMV), Autodesk Prooflab, Stereo-Panorama service with webVR and recently 3D formats, such as glTF and 3D-Tiles-Next.

## Introduction

CAD has been used in the industry for decades, helping designers or engineers to be more accurate, to work and iterate faster, and sharing data with all actors of the project. However, although BIM appeared a few years ago, there is still a lot of progress to be made in the AEC field. Many companies are not well trained in 3D modeling or are not implementing BIM processes in their projects, resulting in time and material waste, inaccuracy or poor workmanship.

Fortunately, the ease of access to softwares and new technologies is changing the way we can work with CAD and BIM processes.

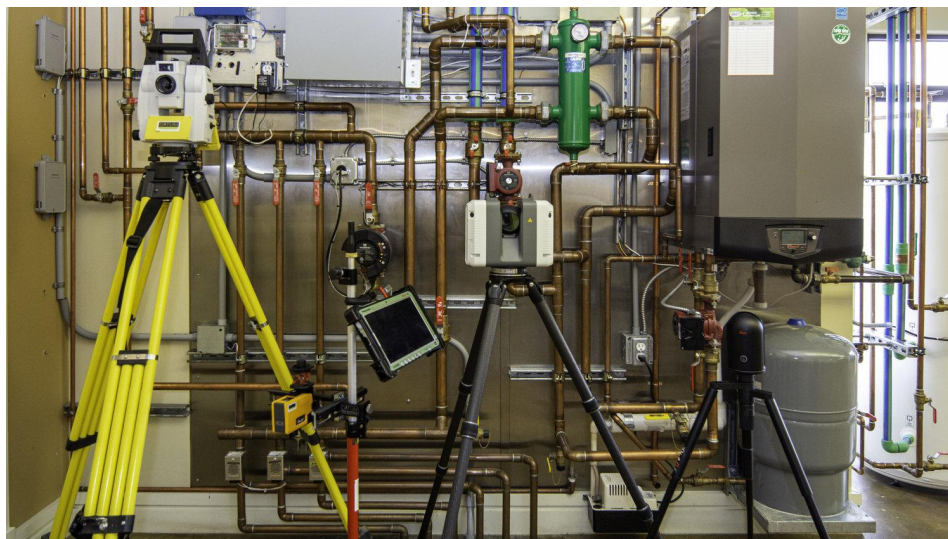
## How is CAD vs Scan used in the industry and how to capture a simple point cloud ?

### Why do we use point clouds ?

Point cloud is now a well known technology in the industry and AEC. It can be used for various applications such as site survey, industrial inspections or construction progress. The accuracy and the quickness to create large models that match the reality, offer the possibility to draw models as built, to compare with 3D models or keep track of different states during a process.

Many tools are available on the market to capture point clouds from reality, using different technologies such as lasers, photogrammetry or lidar. These devices can be embedded on drones or robots and thanks to artificial intelligence, we can automate their paths.

The combination of point cloud technology with 3D models offers great opportunities to enhance workflows, reduce wastes and improve quality.



Different hardware used to capture reality



Autonomous robot scanning a building

### Built-in Lidar is a game-changer

In 2020, Apple released the new iPad Pro which includes a Lidar scanner along with cameras. This scanner enables capabilities never being possible on a standard mobile device. It can measure the distance to surrounding objects up to 5 meters. Improving pictures quality with depth of field (especially in combination with the Night Mode) or speeding Augmented Reality scan with a better accuracy are parts of the new features the Lidar scanner offers.

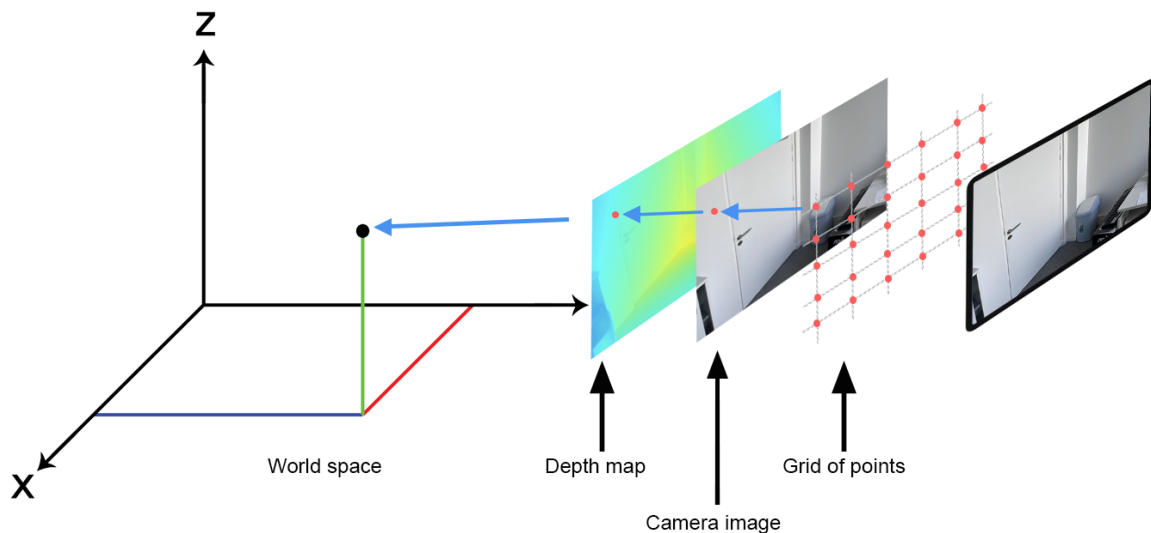


Camera system and Lidar scanner on iPad Pro



## Reality Capture

Using the depth mask accessible from the Metal API, we can easily get the distance from objects. Then, we need to project a point from the camera position with the looking direction and we can calculate 3D coordinates of points. Using the camera image, we can retrieve the RGB values of this point.



Camera system and Lidar scanner on iPad Pro

## Exporting the point cloud

Now, we have a collection of points defined by six values, XYZ coordinates and RGB colors. Many 3D formats can be used to create a point cloud as we only need to describe vertices. In our example, we use PLY format which is a simple text file.

A PLY file starts with the "header" attribute, which specifies the elements of a mesh and their types, followed by the list of elements itself.

```
ply
format ascii 1.0
element vertex 133874
property float x
property float y
property float z
property uchar red
property uchar green
property uchar blue
property uchar alpha
element face 0
property list uchar int vertex_indices
end_header
-1.3394927 0.26312542 -0.8880676 229 224 215 255
-1.3008901 0.27280927 -0.95054207 239 234 223 255
-1.2463698 0.2807801 -0.9852500 236 231 220 255
-1.1901929 0.28937304 -1.0245777 237 232 223 255
-1.1306784 0.29832834 -1.0648515 236 231 222 255
-1.0682554 0.30785447 -1.1083419 235 230 221 255
-1.0021175 0.31782663 -1.1532872 236 231 220 255
-0.9320292 0.32825118 -1.1995732 234 229 218 255
-0.85784256 0.3391862 -1.2476357 234 229 218 255
-0.77918076 0.35043448 -1.2952955 235 230 219 255
-0.6960801 0.36258024 -1.3482238 238 231 218 255
-0.6080649 0.3754163 -1.403952 236 229 216 255
-0.5144205 0.38953382 -1.4675708 245 239 224 255
-0.41817343 0.3996024 -1.4912941 248 242 227 255
```

## Learn how to visualize point clouds in Forge Viewer with alignment with BIM / CAD model

### Point cloud integration

Forge Viewer is built on top of three.js, a popular javascript 3D library. It provides a function to create point clouds.

We are going to use this method :

Create the geometry with all our points, listed in 'positions' and their associated colors listed in 'colors' :

```
const geometry = new THREE.BufferGeometry();
geometry.addAttribute('position', new THREE.BufferAttribute(positions, 3));
geometry.addAttribute('color', new THREE.BufferAttribute(colors, 3));
```

Create a point cloud material :

```
const material = new THREE.PointCloudMaterial({ size: PointSize, vertexColors:
THREE.VertexColors });
```

Create point cloud with the geometry :

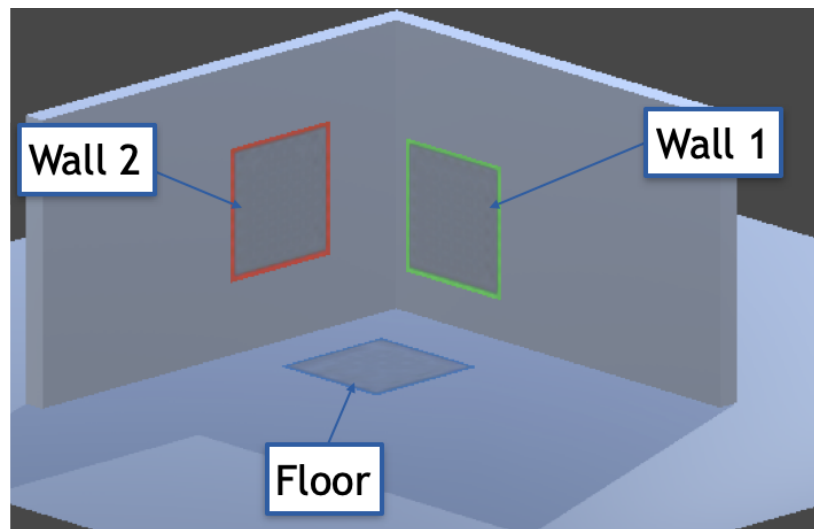
```
const points = new THREE.PointCloud(geometry, material);
```

A more detailed article explaining how to use this method can be found here :

<https://forge.autodesk.com/blog/basic-point-clouds-forge-viewer>

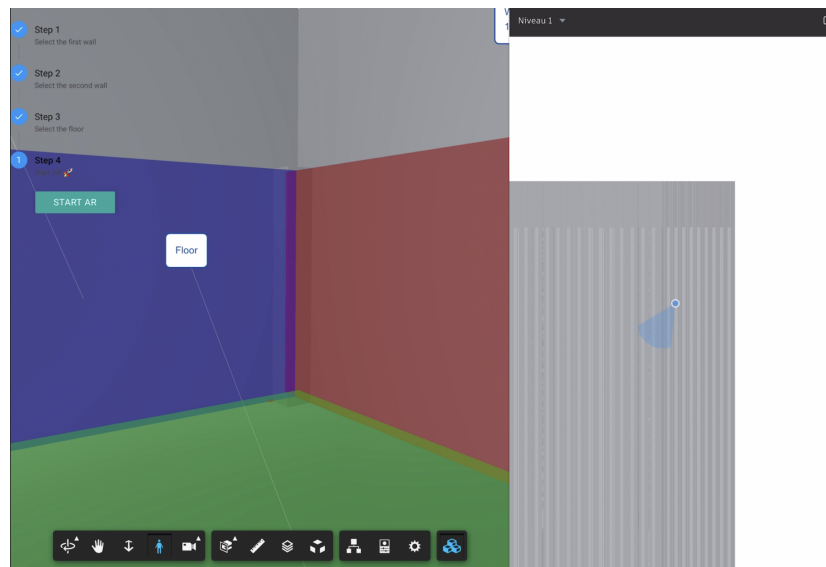
## Align with CAD

Points captured from reality are defined according to the ARKit scene origin. We need to find a common point between the 3D model and the ARKit scene. An easy method is to calculate the intersection between three perpendicular planes (two verticals and one horizontal) and get the vector between this point and the scene origin.



3 planes intersection

Repeat this process in the Forge Viewer, to calculate the vector between this point and the viewer origin (viewerOffset), and in augmented reality, to calculate the vector between the intersection in the real world and the ARKit scene origin (realOffset).



Planes selection in Forge Viewer



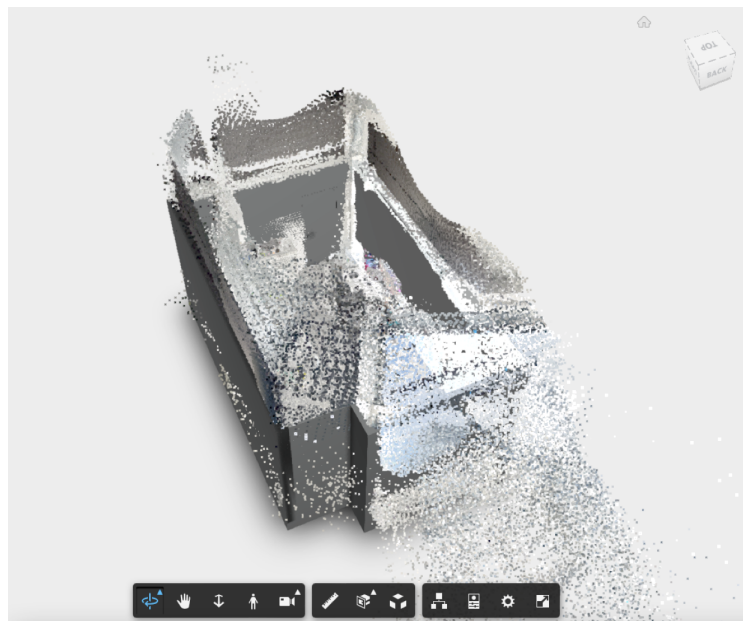
Planes selection in augmented reality

We use the first plane normal as reference to calculate an 'orientation' quaternion to apply to all points.

Using the Forge data, we can easily find the scale of the model and in augmented reality, we assume that values are in meters.

Point values in Forge Viewer can be found like that :

```
pointPosition = ((rawPointValue - realOffset) *
scaleFactor).applyQuaternion(quaternion)) + viewerOffset
```




Point cloud and 3D model in Forge Viewer

## Learn how to download and view 3D-Tiles in many 3D-Engine

To download and view 3D-Tiles, you need to use this script:


<https://gist.github.com/wallabyway/4c7de696d96edc6b57725e21d0dea374>

 **wallabyway** / **pull-tiles-offline.mjs** Edit Delete Unsubscribe Star C

Last active 2 months ago

<> Code Revisions 2 Embed <script src="https://" Download ZIP

handy batch download of 3d-tiles tileset.json from bim360 (using batchPromises for threads)

 **pull-tiles-offline.mjs** Raw

```
1 // PURPOSE: download 3d-tiles files from BIM360, to local drive
2 // INSTALL: npm install node-fetch
3 // RUN: add your BIM360 access-token and the base url, then type
4 // > node pull-tiles-offline.mjs myOutputFolder
5
6 import fetch from 'node-fetch';
7 import fs from 'fs';
8 import util from 'util';
9 import stream from 'stream';
10
11 const token = "eyJh...DV-OCgxNcJnIp-3bNk7Sc8ZE..ISKykb5Q";
12 const URN = "dXJuOm...yc2lrbj0x";
13 const url = 'https://cdn.derivative.autodesk.com/derivativeservice/v2/derivatives/urn:adsk.viewing:fs.file:${URN}/output/';
14 const NUMTHREADS = 64;
15
16 const uris = ["tileset.json"];
17
18 //-----UTILS
19 function recurseNode(node) {
20     uris.push(node.content.uri);
21     for (var n in node.children) {
22         recurseNode(node.children[n]);
23     };
24     return node;
25 }
26
27 // https://github.com/Nicktho/batch-promises/blob/3b73b218b165b974d116b2cc5a7720895af489db/index.js#L4
28 async function batchPromises(batchSize, collection, callback) {
29     const arr = await Promise.resolve(collection);
30     return arr
31         .map((_, i) => (i % batchSize ? [] : arr.slice(i, i + batchSize)))
32         .map((group) => (res) => Promise.all(group.map(callback)).then((r) => res.concat(r)))
33         .reduce((chain, work) => chain.then(work), Promise.resolve([]));
34 }
35
36 // download each tile
37 const streamPipeline = util.promisify(stream.pipeline);
38
39 //end----- UTILS
```



### PURPOSE:

This script (below), helps you download 3d-tiles files directly from BIM360, and saves to your local drive. From there you can host the 3d-tiles on your own server or local file-system. Just point a 3d-tiles viewer to these files.

### HOW TO INSTALL:

Install Node.js latest, and run the following command:

```
npm install node-fetch
```

### How to RUN:

1. Add your BIM360 access-token
2. the base url,
3. then type ...

```
node pull-tiles-offline.mjs myOutputFolder
```

This will then download the tileset.json file and the PNTS files into the myOutputFolder.

Here's the source code:

```
// PURPOSE: download 3d-tiles files from BIM360, to local drive
// INSTALL: npm install node-fetch
// RUN: add your BIM360 access-token and the base url, then type
// > node pull-tiles-offline.mjs myOutputFolder

import fetch from 'node-fetch';
import fs from 'fs';
import util from 'util';
import stream from 'stream';

const token = "eyJh...DV-OCgxNcjlIp-3bNk7Sc8ZE..ISKykb5Q";
const URN = "dXJuOm...yc2lrbj0x";
const url =
`https://cdn.derivative.autodesk.com/derivativeservice/v2/derivatives/urn:adsk.viewing
:fs.file:${URN}/output/`
const NUMTHREADS = 64;
```

```
const uris = ["tileset.json"];

//-----UTILS
function recurseNode(node) {
  uris.push(node.content.uri);
  for (var n in node.children) {
    recurseNode(node.children[n]);
  };
  return node;
}

//
https://github.com/Nicktho/batch-promises/blob/3b73b218b165b974d116b2cc5a7720895af489db/index.js#L4
async function batchPromises(batchSize, collection, callback) {
  const arr = await Promise.resolve(collection);
  return arr
    .map((_, i) => (i % batchSize ? [] : arr.slice(i, i + batchSize)))
    .map((group) => (res) => Promise.all(group.map(callback)).then((r) =>
res.concat(r)))
    .reduce((chain, work) => chain.then(work), Promise.resolve([]));
}

// download each tile
const streamPipeline = util.promisify(stream.pipeline);

//end----- UTILS

async function downloadFile (url, szFile, outfolder) {
  console.log(`downloading... ${szFile}`)
  const response = await fetch(url + szFile, { headers:{ 'Authorization':`Bearer ${token}`}});
  if (!response.ok) return; //skip it... throw new Error(`unexpected response ${response.statusText}`)
  await streamPipeline(response.body,
fs.createWriteStream(`${outfolder}/${szFile}`));
}

async function main (outfolder) {
```

```
fs.promises.mkdir(outfolder, { recursive: true }); // make destination folder, if
not exist

// download tileset.json file
const jsonTileset = await (await fetch(url + "tileset.json", { headers:{
'Authorization':`Bearer ${token}` } })).json();
// decode all it's uri's
recurveNode(jsonTileset.root);

// now wget those uri's, using 64 threads
console.log(`downloading ${uris.length} files`)
await batchPromises(NUMTHREADS, uris, (szFile) =>
    downloadFile(url, szFile, outfolder
))
});

await main(process.argv.slice(2)[0] || 'result');
console.log('done');
```

**Notes:**

- A quick way to get the access token, is to open the debug console, within BIM360, open a recap point-cloud file, then search for the word 'tileset.json'. That's the root file you need.
- Under header, copy the bearer token, and paste this as the 'access token' into the source code above. (See the video demonstration during the AU talk).

The path to the tileset.json file looks like this:

[`https://cdn.derivative.autodesk.com/derivativeservice/v2/derivatives/urn:adsk.viewing:fs.file:\\${URN}/output/`](https://cdn.derivative.autodesk.com/derivativeservice/v2/derivatives/urn:adsk.viewing:fs.file:${URN}/output/)

Copy the URN string into the source code as well.

Now run the script.

## Viewing in different viewers

### Here is Cesium

Here is some minimal cesium HTML file to load the offline tileset...

**NOTE:** change url: 'oaklandTrainStation/tileset.json', to your local myOutputFolder

```
<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Use correct character set. -->
  <meta charset="utf-8">
  <!-- Tell IE to use the latest, best version. -->
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <!-- Make the application on mobile take up the full browser screen and disable user
scaling. -->
  <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1,
minimum-scale=1, user-scalable=no">
  <title>Hello World!</title>
  <script
src="https://cesium.com/downloads/cesiumjs/releases/1.92/Build/Cesium/Cesium.js"></scr
ipt>
  <link
href="https://cesium.com/downloads/cesiumjs/releases/1.92/Build/Cesium/Widgets/widgets
.css" rel="stylesheet">
  <style>
    html, body, #cesiumContainer {
      width: 100%; height: 100%; margin: 0; padding: 0; overflow: hidden;
    }
    #debugger { margin: 20px; padding: 8px; background: rgba(255,255,255,0.5);
font-family: helvetica; position: absolute; top: 0px;}
  </style>
</head>
<body>
  <div id="cesiumContainer"></div>
  <div id="debugger"><input type="checkbox" onclick="tileset.debugShowBoundingVolume=
!tileset.debugShowBoundingVolume">Show Debug-Box</div>
```

```
<script>
  var viewer = new Cesium.Viewer('cesiumContainer', {timeline : false, animation :
false});

  Cesium.ExperimentalFeatures.enableModelExperimental = true;

  var tileset = viewer.scene.primitives.add(new Cesium.Cesium3DTileset({
    url : 'oaklandTrainStation/tileset.json',
    maximumScreenSpaceError: 800,
    debugShowBoundingVolume: false
  }));

  tileset.pointCloudShading.attenuation = true;
  tileset.pointCloudShading.geometricErrorScale = 1/256;

  viewer.zoomTo( tileset );

</script>
</body>
</html>
```

## Viewing in Mapbox

Download the repo: <https://github.com/wallabyway/tiles3d-next-mapbox>

Change the [oaklandtrainstation](#) reference to your own folder

Note that you will need to create a tileset.json file.

To position your point-cloud, somewhere on the planet, you will need to set the 'transform'.

For example, the oakland train station is positioned in Rotterdam (in the demo), with x and y values: 497663.2799987793, 6781475.25, found in the source code here:

<https://github.com/wallabyway/tiles3d-next-mapbox/blob/4888167c3a7ec6bdc4ce7d95a76e18d05271f9bf/oaklandtrainstation/draco/tileset.json#L3759-L3760>

You can convert to lat, long, using this function:

more details here: <https://gis.stackexchange.com/a/387677>



```
function latLongToEPSG3857(lat,lon) {  
    // https://gis.stackexchange.com/a/387677  
    const x = (Math.log(Math.tan((90 + lat) * Math.PI / 360)) / (Math.PI / 180)) *  
(20037508.34 / 180)  
    const y = (lon * 20037508.34 / 180)  
    return [x,y];  
}
```

## Viewing in Forge Viewer v8

Using latest beta forge-viewer (v8)

Create an index.html file:

```
<!DOCTYPE html>  
<html>  
  <head>  
    <title>Test</title>  
    <link rel="stylesheet" href="http://localhost:8000/build-tug/style.css"  
type="text/css">  
    <script src="http://localhost:8000/build-tug/viewer3D.js"></script>  
    <script src="http://localhost:8000/build-tug/init/generic-initialize.js"></script>  
  </head>  
  <body style="margin:0;height:100vh;font-family: Arial, Helvetica, sans-serif;">  
    <h1>3D Tiles POC</h1>  
    <div id="viewer3d" style="position: absolute; width: 100vw; height: calc(100vh -  
100px);"></div>  
  </body>  
  <script src="index.js"></script>  
</html>
```

and a index.js file:

```
import Tiles from "../tiles"

const urnStruct =
"dXJuOmFkc2sud2lwcHJvZDpmcy5maWx1OnZmLmRMVUxpaF9DU3AyWlEtYU15LU9zc2c_dmVyc2lvbj0y";

const
urnArch="dXJuOmFkc2sud2lwcHJvZDpmcy5maWx1OnZmLlZINHBjd1o5UWMYz084TUY0ZzZ0c3c_dmVyc2lvbj0y";

const
urnMEP="dXJuOmFkc2sud2lwcHJvZDpmcy5maWx1OnZmLnFFbmdYOE82VEdlldGFESGF3Z0I1YXc_dmVyc2lvbj0y"

const urn = `urn:${urnMEP}`;

//grab access token from BIM360, and use URL localhost:8000/?accessToken=eyJHbGC...etc

function initPage() {

  var av = Autodesk.Viewing;

  // See file: generic-initialize.js

  const viewer = initializeViewer(customizeOptions);

  // Support for viewer.restoreState()

  viewer.addEventListener(

    av.GEOMETRY_LOADED_EVENT,

    onViewReady.bind(null, viewer),

    { once: true }

  );

  // HACK: How to force a perspective camera from the start?

  viewer.addEventListener(

    av.CAMERA_CHANGE_EVENT, () => {

      if (!viewer.impl.camera.isPerspective) {

        viewer.navigation.toPerspective();

      }

    }

  );

}

function customizeOptions(options) {
```

```
var avp = Autodesk.Viewing.Private;

// LMV

options.isURN = true;

options.documentId = urn;

options.env = "AutodeskProduction";

options.shouldInitializeAuth = true;

options.getAccessToken = getAccessToken;

//options.accessToken = token;

options.api = avp.getParameterByName("api") || undefined;

options.useCookie = (options.api === 'fluent');

//options.useCredentials = options.isURN;

//options.shouldInitializeAuth = options.isURN;

options.isAEC = undefined;

options.config3d.isAEC = undefined;

options.headless = (avp.getParameterByName("headless") === "true");

delete options.config3d.globalOffset;
}

async function onViewReady(viewer) {

  // Initialize Tiles.

  const tiles = new Tiles(viewer);

  await tiles.load({

    url: 'https://dlnyndnhfe3te2.cloudfront.net/aecbuilding/tileset.json'

  });
}
```

```
// Is there a way to not render the tiles every frame?

const clock = new THREE.Clock();

const updateTiles = () => {

  tiles.update(clock.getDelta());

  requestAnimationFrame(updateTiles);

}

updateTiles(0);

}

initPage();
```

and a tiles.js file

```
import * as ThreeLoader3DTiles from "three-loader-3dtiles"

const av = Autodesk.Viewing;

const avp = Autodesk.Viewing.Private;

export default class Tiles {

  static #tilesId = 0;

  #id;

  #viewer;

  #scene;

  #model;

  #runtime;

  constructor(viewer) {

    this.#viewer = viewer;

    this.#id = Tiles.#tilesId++;

    // Add overlay scene.

    const sceneName = `Tiles-${this.#id}`;

    this.#viewer.overlays.addScene(sceneName);

    this.#scene = this.#viewer.impl.overlayScenes[sceneName].scene;

    this.#scene.name = sceneName;

  }

}
```

```
async load(options) {  
    const result = await ThreeLoader3DTiles.Loader3DTiles.load({  
        url: options.url,  
        renderer: this.#viewer.impl.glrenderer(),  
        options: {  
            dracoDecoderPath: 'https://unpkg.com/three@0.137.0/examples/js/libs/draco',  
            basisTranscoderPath: 'https://unpkg.com/three@0.137.0/examples/js/libs/basis',  
            pointCloudColoring: ThreeLoader3DTiles.PointCloudColoring.RGB,  
            maximumScreenSpaceError: 200  
        }  
    });  
  
    result.model.scale.set(4.0, 4.0, 4.0);  
    result.model.rotation.set(0, 0, 3.14/2);  
    result.model.position.set(70, 20, 20);  
    this.#model = result.model;  
    this.#runtime = result.runtime;  
    this.#scene.add(this.#model);  
}  
  
update(delta) {  
    const camera = this.#convertLMVCameraToThree(this.#viewer.impl.camera);  
    this.#runtime.update(delta, this.#viewer.impl.glrenderer(), camera);  
    this.#viewer.impl.invalidate(true, true, true);  
}  
  
#convertLMVCameraToThree(camera) {  
    const threeCamera = camera.isPerspective  
        ? new THREE.PerspectiveCamera()  
        : new THREE.OrthographicCamera();  
  
    threeCamera.near = camera.near;  
    threeCamera.far = camera.far;
```



```
if (camera.isPerspective) {  
    threeCamera.fov = camera.fov;  
    threeCamera.aspect = camera.aspect;  
} else {  
    threeCamera.left = camera.orthographicCamera.left;  
    threeCamera.right = camera.orthographicCamera.right;  
    threeCamera.top = camera.orthographicCamera.top;  
    threeCamera.bottom = camera.orthographicCamera.bottom;  
}  
  
threeCamera.position.copy(camera.position);  
threeCamera.up.copy(camera.up);  
  
threeCamera.lookAt(camera.target.x, camera.target.y, camera.target.z);  
  
threeCamera.updateProjectionMatrix();  
threeCamera.updateMatrixWorld();  
  
return threeCamera;  
}  
}
```

1. Specify the URN in the code
2. Then the URL parameter, add accessToken=[...]
3. Change the url for the path to the point-cloud
4. Align the cad / Scan by moving the transform
  - a. using three.js rotation/scale/position in the code above
  - b. changing the transform in the tileset.json file

## Learn how to merge points cloud in a Revit file by using Forge Design Automation

Our first approach to load point clouds into Revit was to import raw values (as we did in Forge Viewer) with a dedicated plugin.

Using *BasicPointCloudEngine*, it's possible to create a point cloud. But, with this method, nothing is persistent (if Revit is closed, you need to use the plugin again to load point cloud). Due to the fact that Revit (from version 2019) does not support raw point clouds import, we are very limited with this approach. However, Revit supports Recap Pro files.

### Converting our point cloud into Recap Pro format

Autodesk provides the Recap SDK which allows to create and update point clouds using RCP / RCS formats.

Assigning points coordinates and colors into RCPPointBuffer :

```
void assignPointAndColors(RCPointBuffer& pointBuffer, std::vector<std::wstring>&
plyPoints)
{
    pointBuffer.addAttribute(RCAttributeType::Position);
    for (auto y = 0u; y < plyPoints.size(); ++y)
    {
        std::vector<std::wstring> point;
        tokenize(plyPoints[y], ' ', point);
        const auto xv = std::stod(point[0]) * -1;
        const auto yv = std::stod(point[2]);
        const auto zv = std::stod(point[1]);
        pointBuffer.setPositionAt(y, RCVector3d(xv, yv, zv));
    }
    pointBuffer.addAttribute(RCAttributeType::Color);
    for (auto z = 0u; z < plyPoints.size(); ++z)
    {
        std::vector<std::wstring> point;
        tokenize(plyPoints[z], ' ', point);
        const int rv = std::stoi(point[3]);
        const int gv = std::stoi(point[4]);
        const int bv = std::stoi(point[5]);
        const int av = std::stoi(point[6]);
        pointBuffer.setColorAt(z, RCVector4ub(rv, gv, bv, av));
    }
}
```

Create RCP and RCS files from point buffer :

```
RCImportPointCloudSettings importSettings;  
importSettings.scanType = RCScanType::Stationary;  
importSettings.scanName = RCString(scanName.c_str());  
RCProjectImporter::createProjectFromPoints(pointBuffer, outputPath,  
importSettings, callbackPtr);
```

Compile this code as an executable and run it from serverless functions to create a microservice for file conversion.

### Point Cloud integration with Revit model

The Revit API allows use of *PointCloudEngine* and it supports natively RCP / RCS files.

Create a *PointCloudType* with Recap Pro scan (RCS) and create a *PointCloudInstance* :

```
var PCType = PointCloudType.Create(doc, "rcs", fileFullPath);  
var translation = Transform.CreateTranslation(new XYZ(0, 0, 0));  
var PCInstance = PointCloudInstance.Create(doc, PCType.Id, translation);
```

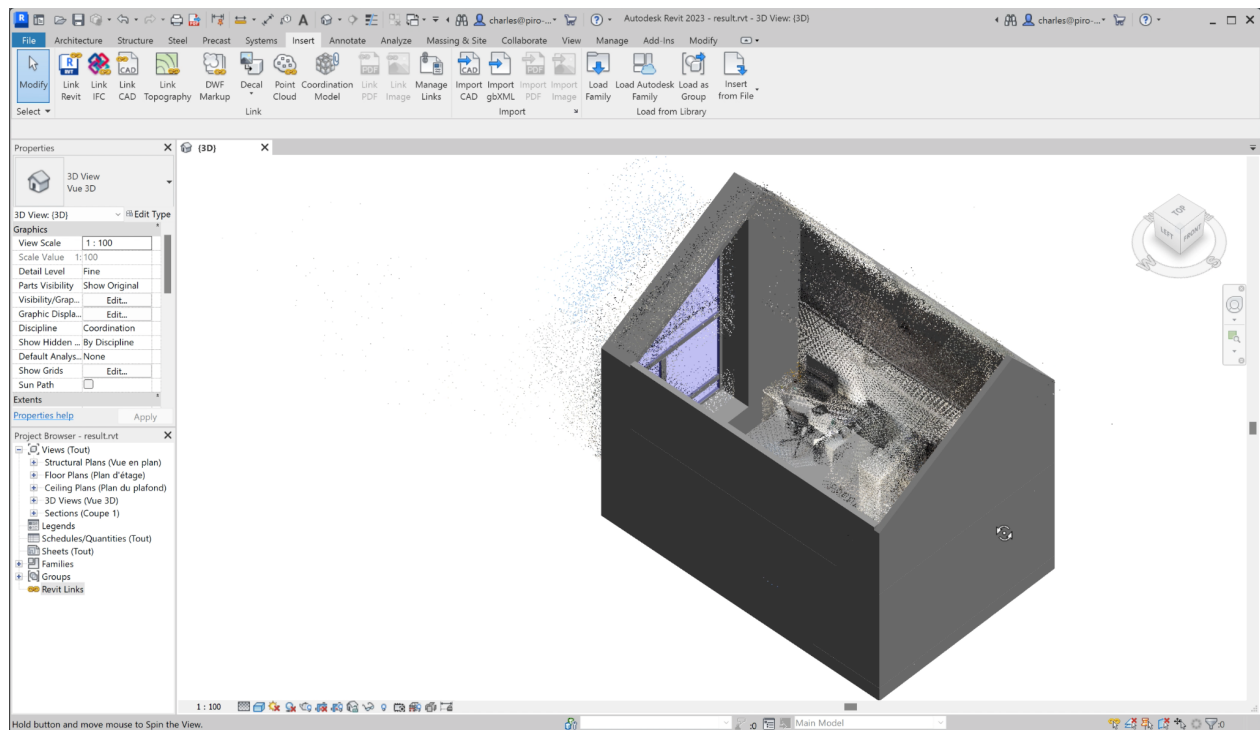
Alignment of point cloud with 3D model :

```
var ARCenter = data.DocumentOrigin.Subtract(data.RealOrigin);  
ElementTransformUtils.MoveElement(doc, _pointcloud.Id, ARCenter);  
var axis = Line.CreateUnbound(data.DocumentOrigin, Transform.Identity.BasisZ);  
var rotation = data.RealRef.Wall11.AngleTo(data.ARRefs[0].PlaneNormal);  
ElementTransformUtils.RotateElement(doc, _pointcloud.Id, axis, rotation);
```

## Use Design Automation for Revit to automate

With Forge Design Automation for Revit API, we have the ability to run a plugin on Forge servers to automate a process.

We used the plugin we created for Revit and made DesignAutomation adaptation. Then, after linking the requests together, we can get back our Revit model and our RCS scan, combined and aligned.



Point cloud and 3D model in Revit