potree-templates

Release 0.1

Federica Gaspari

Feb 24, 2023

CONTENTS:

CHAPTER

ONE

INTRODUZIONE

1.1 Credits

1.1.1 Subsection

Example of anchor to a section *Scopo*

POLITECNICO
MILANO 1863

POLITECNICO
MILANO 1863

Text goes here **Bold** *Italic* example^{superscript} example$_{subscript}$

1.2 Scopo

Note: Example of note box.

See also:

Example of seealso box.

Warning: Example of warning box.

Hint: Example of hint box.

Example of html iframe

Example of download, file must be in the repository folder `png`

This is a link to a web page https://labmgf.dica.polimi.it/.

This is a link to a web page.

```
<b><a href="https://labmgf.dica.polimi.it/">link</a></b> - Example of code-block.
```

1.3 Glossario

Example of bullet list:

- item 1
- item 2
- item 3
- etc.

Example of auto-enumerated list:

1. item 1
2. item 2
3. item 3

CHAPTER

TWO

POTREE

Potree is a *free open-source WebGL* based point cloud renderer for large point clouds, developed at the Institute of Computer Graphics and Algorithms, TU Wien, Austria. There are 3 ways for accessing Potree:

- **Potree Desktop**: Desktop version of Potree. Allows drag&drop of point clouds into the viewer (https://github.com/potree/PotreeDesktop/releases)
- **Potree Converter**: Convert your point cloud to the Potree format (https://github.com/potree/PotreeConverter/releases)
- **Potree Develop**: Edit and develop several potree examples (https://github.com/potree/potree/)

Project Credits

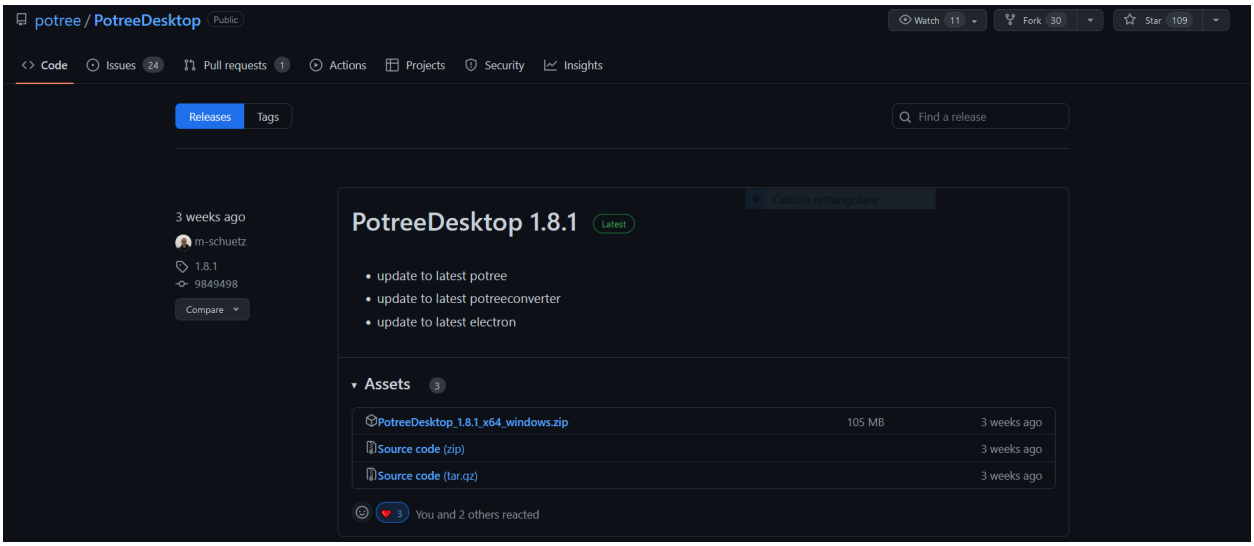
- The multi-res-octree algorithms used by this viewer were developed at the Vienna University of Technology by Michael Wimmer and Claus Scheiblauer as part of the Scanopy Project.
- Three.js, the WebGL 3D rendering library on which potree is built.
- plas.io point cloud viewer. LAS and LAZ support have been taken from the laslaz.js implementation of plas.io. Thanks to Uday Verma and Howard Butler for this!
- Harvest4D Potree currently runs as Master Thesis under the Harvest4D Project
- Christian Boucheny (EDL developer) and Daniel Girardeau-Montaut (CloudCompare). The EDL shader was adapted from the CloudCompare source code!
- Martin Isenburg, Georepublic, Veesus, Sigeom Sa, SITN, LBI ArchPro, Pix4D as well as all the contributers to potree and PotreeConverter and many more for their support.

2.1 Potree Desktop

A desktop/portable version of the web-based point cloud viewer Potree, thanks to Electron. This versione allows you to load converted point clouds from your hard disk or external drive. It's also portable, so you can put your models together with the viewer on a USB drive and open it wherever you go. It's only been tested on Windows at the moment. It may not work on other systems or you may only be able to use it on the same Operating System that you've initially built it on. You can also drag&drop cloud.js files into the window to add point clouds to the scene.

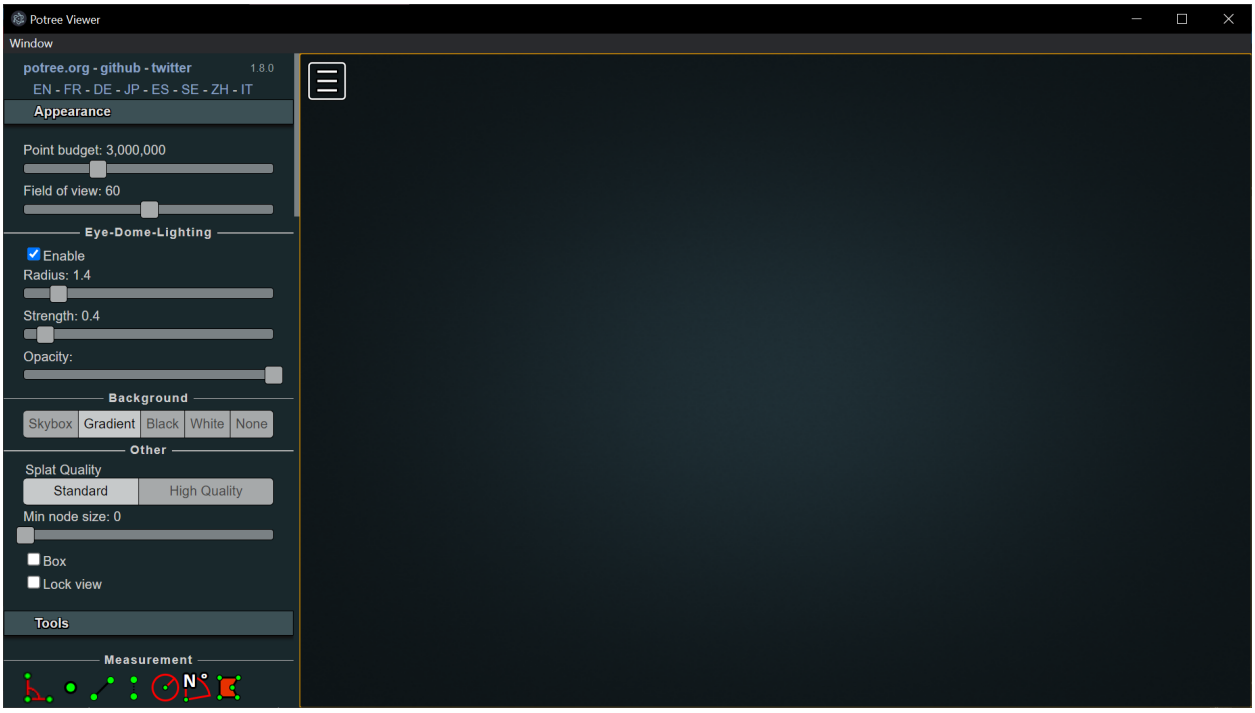
2.1.1 Getting started

Download the Potree ,zip files for Windows from this link: https://github.com/potree/PotreeDesktop/releases



2.1.2 Graphic User Interface

Once you downloaded the installer .zip, extract all the files and execute PotreeDesktop.bat. Then, a new window will appear with the main Graphic User Interface of Potree.



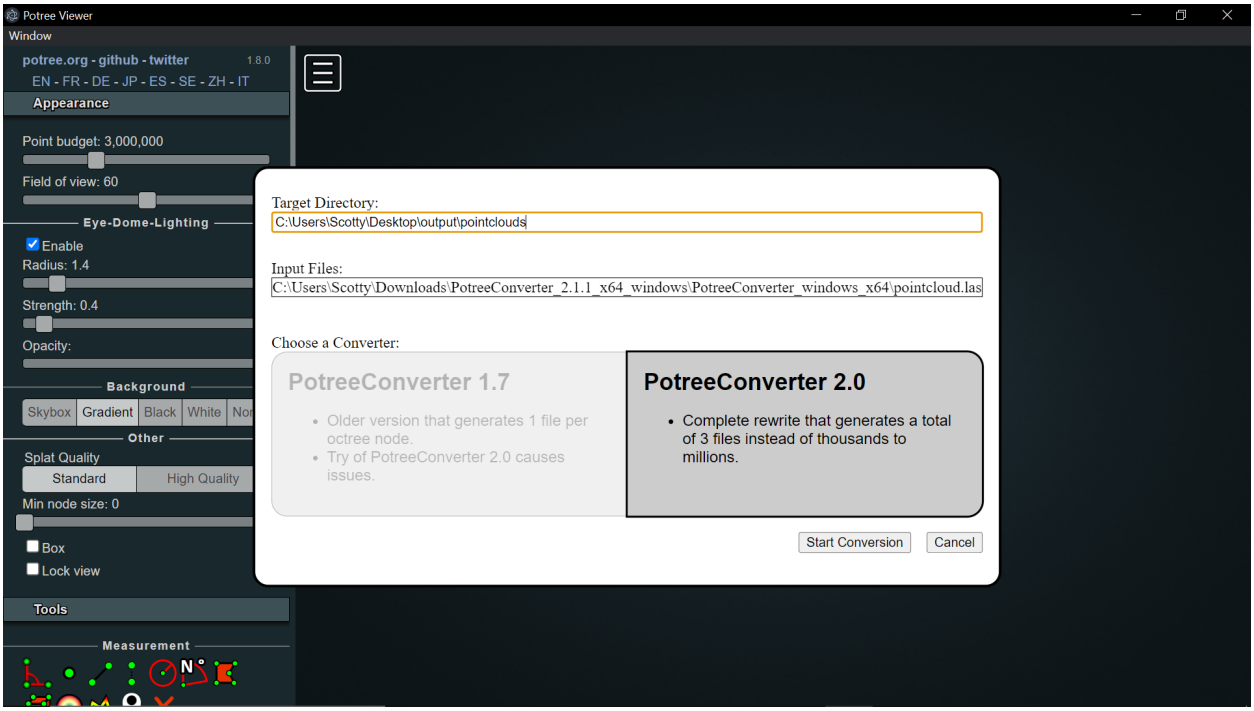
The Potree GUI is made of 2 components:

- **Sidebar**: on the left, it includes all the main features and tools for point-clouds elaborations in the Potree environment.

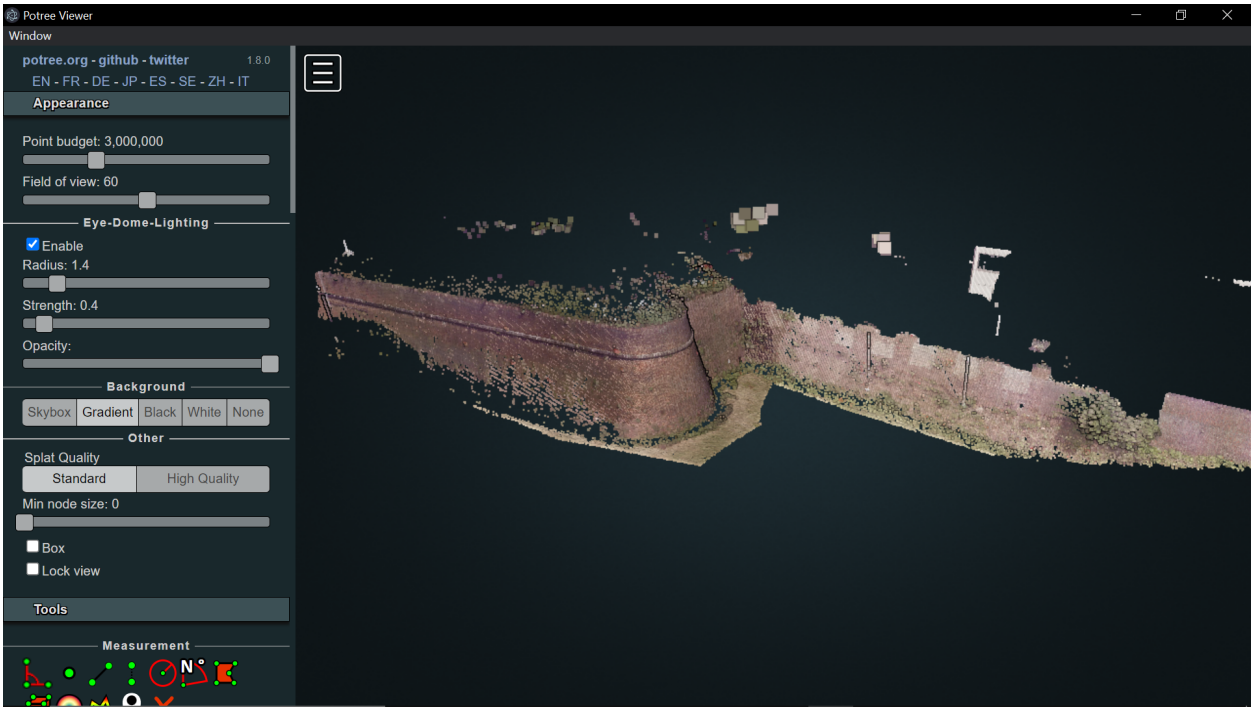
- **Viewer**: on the right, it is the actual space for visually exploring and navigating the point-clouds.

2.1.3 Pointcloud conversion

PotreeDesktop provides also a user-friendly interface for converting pointclouds in a Potree-compatible format. In order to do this, you can simply drag&drop the desired poincloud file (in a .las/.laz format) inside the viewer window. In a new window, after checking that the output target folder and the input files directory are defined as desired, it is required to select the PotreeConverter version to be adopted for the processing. Version 2.0 is the suggested one, generating only 3 files instead of thousands to millions. Click on the *Start Conversion* button to continue.



After the processing, the pointcloud is loaded in the viewer and the converted files are available in the previously defined output target directory.

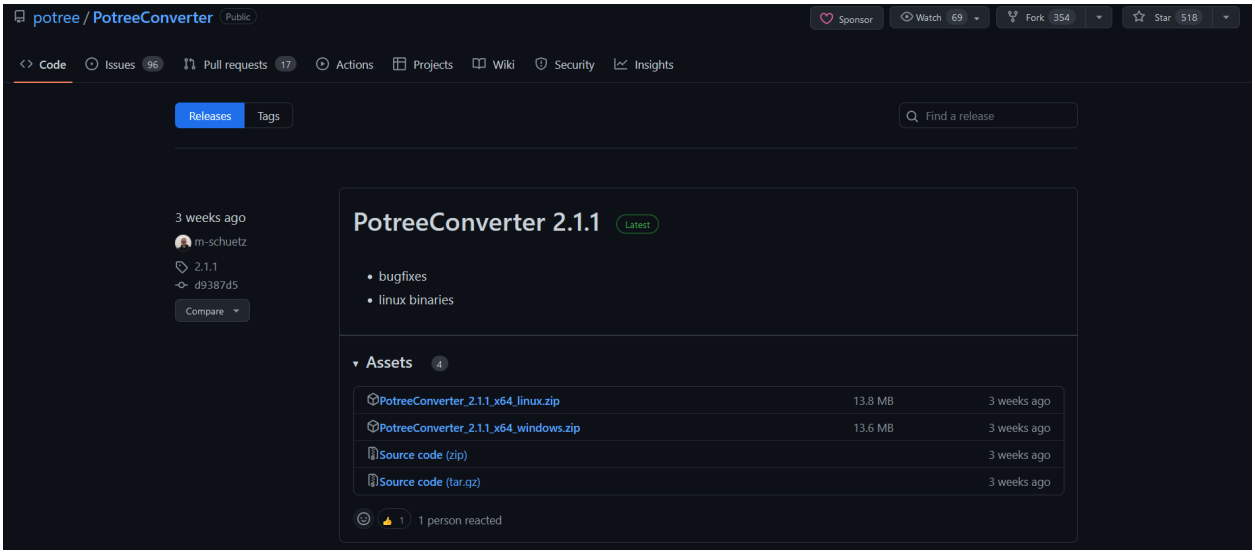


2.2 Potree Converter

PotreeConverter generates an octree LOD structure for streaming and real-time rendering of massive point clouds. The results can be viewed in web browser with Potree. The conversion operation produces a total of 3 files, with better support for standard LAS attributes and arbitrary extra attributes.

2.2.1 Getting started

Download and extract the PotreeConverter files for Windwos from this link: https://github.com/potree/PotreeConverter/releases



Be sure that the PotreeConverter.exe is present inside the unzipped folder.

Note: For making easier the conversion process, it is suggested to copy and paste in the unzipped PotreeConverter folder the pointcloud to be converted in a .las/.laz format.

For Windows users:

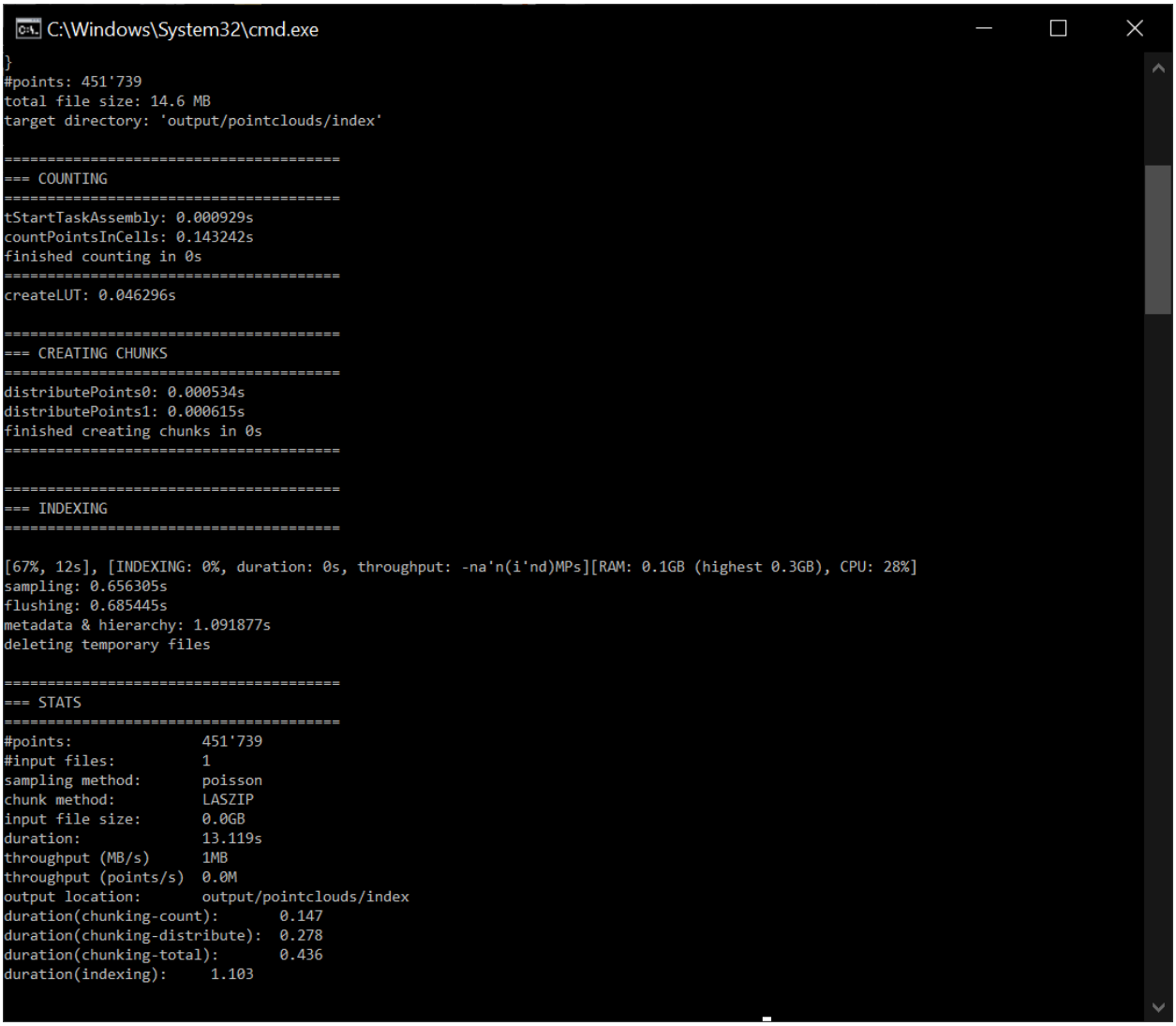
For starting the conversion, you can write "cmd" in File Explorer Address Bar and press enter. The cmd shell will be opened having as referenced directory the PotreeConverter folder. In order to trigger the conversion, modify the following code according to your needs and then paste it in the cmd shell:

```
.\PotreeConverter.exe pointcloud.las -o output -p index
```

Where:

- *PotreeConverter.exe* specify the executable file for the convertion;
- *pointcloud.las* select the point cloud (las or laz) to be converter. This format is valid in case the point cloud file is located in the same folder of the converter exe. Otherwise it is necessary to specify the complete path of the file;
- *output* is the name of the directory where to save the converted point cloud. In this case too, if the target folder is outside the converter one, it is necessary to specify the entire path.
- *index* is the default name to be given to the output file.

After launching, the conversion is executed and details about each step of the the processing are reported on the cmd shell window.



If successful, the conversion procedure gives as output a folder named as defined in the command (in this case *output*) containing 3 elements:

1. a folder named **libs** that contains all the required libraries and scripts to make Potree operative (e.g. *three.js*, *openlayers*...);
2. a folder named **pointclouds** containing an **index** directory that collects that 4 files resulting from the conversion of the original pointclouds. The *metadata.json* is the file called in the Potree environment when loading the pointcloud to the scene;
3. an html file called **index.html** that includes a basic structure for a web page with a Potree viewer.

2.2.2 Publications

- Potree: Rendering Large Point Clouds in Web Browsers.
- Fast Out-of-Core Octree Generation for Massive Point Clouds, Schütz M., Ohrhallinger S., Wimmer M.

2.3 Potree Develop

For more details about the codes and libraries on which Potree is built, it is recommended to check the official Github repository: https://github.com/potree/potree. Many examples on how to implement Potree functionalities and customize them are available on the example folder with formatted html files dedicated to each case.

CHAPTER

THREE

TEST ENVIRONMENT

This section will cover how to test locally a Potree web page before deploying it to a server.

Due to strict security policies in browsers, it is not possible to open potree html files directly on your pc because potree needs permission to load files. You have to put all necessary source files and the pointcloud on a webserver to view the result. You can, however, install a local webserver on your pc. XAMPP, which contains Apache Webserver as well as PHP and MySQL, is the suggested solution for testing locally potree pages.

After you've installed and started Apache/XAMPP, you can access files in your htdocs directory through a localhost URL. Assuming your htdocs directory is *C:xampphtdocs*, you can access it in your browser with:

```
http://localhost
```

CHAPTER

FOUR

GETTING STARTED WITH POTREE

This will cover how to create a simple template for navigating point cloud on a Potree environment.

Examples of potree templates:

- **Standard template**: basic structure of a Potree viewer with the possibilities to integrates functionalities for annotations, oriented images and more.
- **Cultural Heritage template**: example of an updated and custom Potree viewer that enrich the basic functionalities with tools for better exploring 3D products in the field of Cultural Heritage.

4.1 Viewer controls

- *Double Click* - Sets center of rotation and zooms into that point
- *Left Mouse Press* - Pan the view
- *Middle Scroll Wheel* - Zooms in and out
- *Right Mouse Press* - Rotate the view around the center point
- *Left Mouse Press (Pressing Ctrl)* - Rotate the view base on the camera position

4.2 User Interface

4.2.1 Appearance

Point Budget Slider

The point budget limits the number of points loaded and rendered at any given time, which helps to adapt performance requirements to the capabilities of different hardware. It gives you a complete version of point clouds.

Potree Viewer will save the previous selected point budget as the default point budget when the viewer is loaded next time.

Field of View

To control viewable range of point clouds elements to be included within the scene, the field of view can be adjusted. Default value is set to 55 degrees. The field of view will determine how much of the pointcloud you can see in your screen.

Eye Dome-lightning

The Potree Point Cloud viewer module can implement eye dome-lighting, a lighting model that highlights the shapes of objects.

Eye Dome-lighting group objects, shade their outlines and enhances depth perception in scientific visualization images.

It is useful for recognition and measurement of structures within pointcloud. It can be modified by adjusting Radius, Strength, and Opacity.Those are set to give you tight controls of movement with in the model.

By default, Eye Dome-Lighting is enabled on Potree viewer, but it can be disabled by clicking on the enable option.

Background

Potree viewer background can be modified with following available options:

- Skybox
- Gradient
- Black and white

Splat Quality

Standard/High quality are two different splat method which will give you the different viewing experience on your computer. Choose the one that works better for you.

- Min node size - impacts the point density of the nodes represented.
- Box - Displays the boxes of the nodes.
- Lock view - Lock the point cloud view, preventing to load or unload points to the model.

4.2.2 Measurements

Potree Viewer provides several tools for measurement. This tool set consist of 12 elements. It also has controls for showing or hiding the resulting measurement labels.

Measurements are performed by use of left clicking the mouse on the desired points and right clicking is needed to terminate the process.

Angle

This tool measures the tridimensional angle formed by the lines connecting three points. To start a measurement, click on the angle measurement icon, then left click on three desired position and the process will be automatically completed.The unit of angle measurement is degree.

All red points are draggable to adjust their positions.

Point

This tool set a point at desired position and display its XYZ coordinate. To start a measurement, click on the point measurement icon, then click on the desired position and the process will be completed. All red points are draggable to adjust their positions.

Distance

This tool measures the tridimensional distance of the lines connecting a series of points. To start a measurement, click on the distance measurement icon and start clicking on the desired position (two or more). Right click to complete the measurement process. The unit of distance measurement is meter.

Further information such as total length can also be obtained from selecting this under the scene section in the sidebar. All red points are draggable to adjust their positions.

Height

This tool measures the vertical distance between two points. To start a measurement, click on the height measurement icon and then click on the desired two positions. The process will be automatically completed. More information can also be obtained from selecting this element under the scene section in the sidebar.

All red points are draggable to adjust their positions.

Circle

This tool measures the radius of a circle formed by three points. To start a measurement, click on the circle measurement icon and then click on three desired positions.The process will be automatically completed. All red points are draggable to adjust their positions.

More information such as Circumference can also be obtained from selecting this element under the scene section in the sidebar.

Azimuth

This tool measures the azimuthal angle of a line. This line is formed by two points selected by the user, the angle is measured in degrees, clockwise from 0 to 360 and starting from the geographical north. To start a measurement, click on the azimuth icon and then click the first position for the center point, then click the second position to complete the measurement.

More information can also be obtained from selecting this element under the scene section.

Area

This tool measures the horizontal area formed by a polygon. To start a measurement, click on the area icon and start clicking on the positions forming the desired polygon (three or more).

Right click to finish measurement.

Further information can also be obtained from selecting this element under the scene section.

Volume (cube)

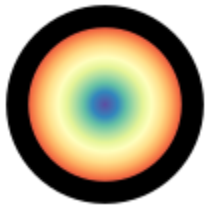
This tool measures the horizontal area formed by a cube. To start a meaurement, click on the volume(cube) icon and click on the desired position in pointcloud to place the cube.

It is possible to relocate, rescale and rotate the cube using the displayed handlers. click the eye icons on the cube can view cube from the angle normal to the plane.

Right click to finish the measurement.

More information can also be obtained from selecting this element under the scene section.

Volume (sphere)



This tool measures the volume formed by a sphere. To start a measurement, click on the volume (sphere) icon and click on the desired position in point cloud to place the sphere. It is possible to relocate, rescale and rotate the sphere using the displayed handlers.

Right click to finish measurement.

Further information can also be obtained from selecting this element under the scene section.

Height profile

The tool icon looks like a multicolored M.

This tool creates a height profile formed by a line on the point cloud. To start a measurement, click on the Height profile icon and then form a line on the pointcloud by clicking on the desired points (two or more).

When you mouse over the point cloud data, you should now see a red ball attached to your mouse cursor. This allows you to drop nodes and establish the location of your profile.

When you want to finalize your profile double click on the last node. Once a profile is finalized, you can still change its location by clicking on the red nodes and dragging them to a different location.

Further information and options, such as Show 2d Profile, can also been obtained from selecting this element under the scene section.

A profile of the lidar data should now be visible at the bottom of the screen. The profile will also update in real time if you move the profile throughout the data.

The save button in the upper right corner of the profile window will download a las file(3D) containing the points from the profile or a CSV file(2D) containing the metadata of points.

This is useful for extracting the data you want to use for further analysis.

Annotation

This tool creates an annotation label on a highlighted point on the pointcloud. To label a new annotation, click on the annotation icon and then click on the desired position.

To edit the annotation, select this element under the scene section, then edit Title and Description.

Clicking to the annotation can zoom to the annotation.

Remove all measurement

This tool removes all measurements on the pointcloud. To remove all measurement, click on the Remove all measurements icon or press Escape on the keyboard.

4.2.3 Clipping

Point cloud can be clipped by selecting an area. Clipping options include None / Highlight / Inside /outside.

To clip a point cloud, click on the volume clip icon, place the cube on the model and relocate, rescale, and rotate to contain the desired area. Highlight is set by default as the clipping method.

If only the points contained within the cube needs to be displayed, click on "Inside", otherwise click on "Outside".

To remove the clipping volume or polygons click on the Remove all clipping volumes icon.

4.2.4 Navigation

The default navigation mode is Earth control. Potree viewer will always save the previous selected navigation mode as the default navigation mode when the viewer is loaded next time.

Earth Control

Earth control navigated as anchored to the pointcloud. Mouse left button pans the pointcloud, mouse wheel controls zoom, and right button orbits the pointcloud.

Pressing "Ctrl", key and holding the left mouse button changes the direction of the view.

Fly Control

Fly control moves the view as in a bird eyes using the keyboard. Keys "W" and "S" moves forward and backwards, respectively and in the direction of the view, while "A" and "D" moves left and right respectively.

Also, the "R" and "F" keys moves the view up and down. The mouse left button changes the direction of the view, mouse wheel controls the speed for these movements and right button moves the view in the XYZ axis.

Helicopter Control

Helicopter control moves the view as in an aircraft using the keyboard. Keys "W" and "S" moves forward and backwards, respectively restricted in a horizontal plane, while "A" and "D" moves left and right respectively.

Also, the "R" and "F" keys moves the view up and down. The mouse left button changes the direction of the camera, mouse wheel controls the speed for these movements, and right button moves the model in the XY axis.

Orbit Control

Orbit Control is similar to earthcontrol. The mouse left button orbits the model, the wheel controls zoom, and the right button moves the model in the XYZ axis.

Full extent

Full extent button restores all pointcloud view.

Navigation cube

Navigation cube displays a wireframe cube from different direction. Click the icon to view pointcloud from the specific direction.

Compass

Compass button displays/hides a compass on the upper right corner.

4.2.5 Scene

The Scene section displays a file tree containing all the scene elements. Elements are arranged in six groups, which are Point clouds / Measurements / Annotations / Other / Vector / Images.

Each element within these groups have subtrees that can be selected to get more information and control its properties.

CHAPTER

FIVE

POTREE STANDARD TEMPLATES

This page will cover details on how to build a web page with a Potree viewer using already available functionalities.

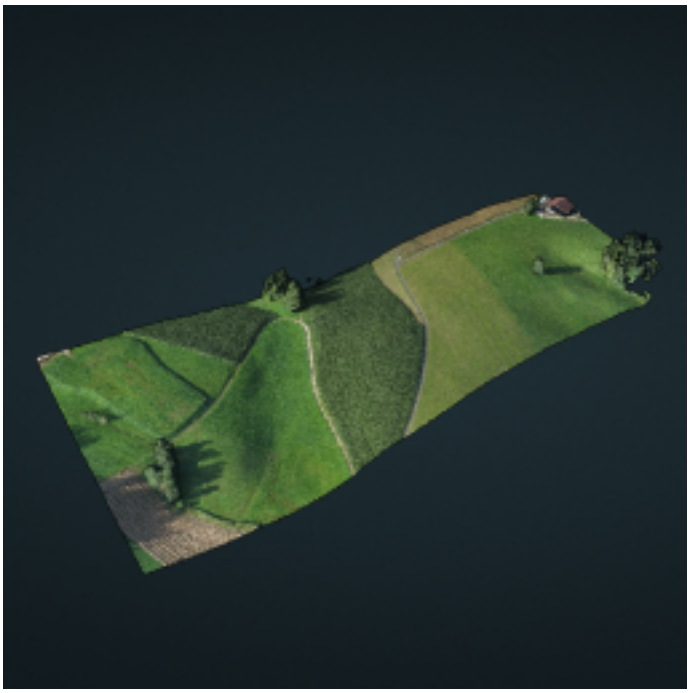
5.1 Examples

- *Basic Viewer*
- *CA13 (18 billion points)*
- *Retz (Potree + Cesium)*
- *Classifications*
- *Various Features*
- *Toolbar*
- *Load Project*
- *Matcap*
- *Virtual Reality*
- *Heidentor*
- *Lion*
- *Lion LAS*
- *Lion LAZ*
- *EPT*
- *EPT Binary*
- *EPT zstandard*
- *Clipping Volume*
- *Oriented Images*
- *Elevation Profile*
- *Measurements*
- *Meshes*
- *Multiple Point Clouds*
- *Camera Animation*
- *Features (CA13)*

- *Annotations*
- *Hierarchical Annotations*
- *Animation Paths*
- *Shapefiles*
- *Cesium CA13*
- *Geopackage*
- *Cesium Sorvilier*
- *Custom Sidebar Section*
- *Embedded iframe*
- *Gradient colors*

5.1.1 Basic Viewer

Working example



After cloning the Potree develop repository as suggested in section [reference], navigate to the *examples* folder and search for the viewer.html file. This file template includes the basic settings for a functional Potree Viewer and represents the basis for all the other examples too.

In the **head** section you can find all the stilesheets' references for each library needed in the basic viewer, defining the rendering of the Potree navigation area as well as the sidebar appearance. In this section, like every HTML page, it is possible to define the page title together with metadata information about the author and the content of the document.

```
<head>
        <meta charset="utf-8">
        <meta name="description" content="">
        <meta name="author" content="">
        <meta name="viewport" content="width=device-width, initial-scale=1.0, user-
↪scalable=no">
        <title>Potree Viewer</title>

        <link rel="stylesheet" type="text/css" href="../build/potree/potree.css">
        <link rel="stylesheet" type="text/css" href="../libs/jquery-ui/jquery-ui.min.css
↪">
        <link rel="stylesheet" type="text/css" href="../libs/openlayers3/ol.css">
        <link rel="stylesheet" type="text/css" href="../libs/spectrum/spectrum.css">
        <link rel="stylesheet" type="text/css" href="../libs/jstree/themes/mixed/style.
↪css">
</head>
```

Diving in the **body** section, first used JS libraries and dependencies are included.

```
<script src="../libs/jquery/jquery-3.1.1.min.js"></script>
<script src="../libs/spectrum/spectrum.js"></script>
<script src="../libs/jquery-ui/jquery-ui.min.js"></script>
<script src="../libs/other/BinaryHeap.js"></script>
<script src="../libs/tween/tween.min.js"></script>
<script src="../libs/d3/d3.js"></script>
<script src="../libs/proj4/proj4.js"></script>
<script src="../libs/openlayers3/ol.js"></script>
<script src="../libs/i18next/i18next.js"></script>
<script src="../libs/jstree/jstree.js"></script>
<script src="../build/potree/potree.js"></script>
<script src="../libs/plasio/js/laslaz.js"></script>
```

The Potree container class is then defined, settings also the renderer area and the sidebar elements too. By changing the *background.jpg* path in the renderer area, it is possible to change the background image that appears at the beginning of the rendering.

```
<div class="potree_container" style="position: absolute; width: 100%; height: 100%;␣
↪left: 0px; top: 0px; ">
        <div id="potree_render_area" style="background-image: url('../build/potree/
↪resources/images/background.jpg');"></div>
        <div id="potree_sidebar_container"> </div>
</div>
```

In the following lines, the script create a new viewer and define the scene settings. By calling different functions, it also defines appeareance options like:

- **Eye-Dome Lightning**: it could be enabled or disabled with *.setEDLEnabled()*;
- **Field of View**, defining the numerical value for the view angle with *.setFOV()*;
- **Point Budget** sets the default point population for point cloud rendering *.setPointBudget()*;

- **Size of Octree Cells** giving a numerical value as input to *.setMinNodeSize()*;
- **Background** appearance, choosing as parameter for *.setBackground()* one of the following 4 defaults options: skybox, gradient, black and white;
- **Description**, a text defined with *.setDescription()* that supports HTML and appears on the top of the renderer area;
- **Setting parameters**, loading with *.loadSettingsFromURL()* a file with the desired settings for appearance.

When applying *.loadGUI()*, it is possible to set the default style of the Potree sidebar by:

- Setting the display **language**: simply put the language code inside *.setLanguage()*;
- Choosing the **visibility of sidebar sections** at loading: this can be done by passing the section class name (e.g. #menu_tools) inside *$("#menu_tools").next().show()*. The other class name are: [UPDDATE];
- The **visibility of the entire sidebar**, set as true when including *.toggleSidebar()*.

```
<script>
        window.viewer = new Potree.Viewer(document.getElementById("potree_render_area"));
        iewer.setEDLEnabled(false);
        viewer.setFOV(60);
        viewer.setPointBudget(1_000_000);
        viewer.loadSettingsFromURL();
        viewer.setBackground("skybox");
        viewer.setDescription("Point cloud courtesy of <a target='_blank' href='https://
↪www.sigeom.ch/'>sigeom sa</a>");

        viewer.loadGUI(() => {
                viewer.setLanguage('en');
                $("#menu_tools").next().show();
                ("#menu_clipping").next().show();
                viewer.toggleSidebar();
        });
</script>
```

After setting the viewer and scene parameter, it's time to include the point cloud. This can be done through the *.loadPointCloud()* function, including in the parenthesis:

- the path to the file of the point cloud;
- the name of the point cloud that will appear in the scene section of the sidebar (e.g. "sigeom.sa");

Then, a series of parameters is set inside the loading function:

- **material.size** defines the size of dots used for the cloud rendering;
- **material.pointSizeType** indicates the point sizing view mode to be adopted in the render, choosing between the following options: *FIXED*, *ATTENUATED* or *ADAPTIVE*;
- **material.shape** sets the shape used for point shape rendering. It could be *SQUARE*, *CIRCLE* or *PARABOLOID*;
- **material.pointColorType** makes possible to select the attribute to be used for cloud coloring in the renderer area.

All these parameters corresponds to the properties accessible after clicking on the point cloud from the objects list in the scene sidebar section.

The *.addPointCloud()* is then applied to the scene to which the pointcloud should be added. Additionally, the view on the model is defined through the *.setView()* method that takes as input the coordinates of camera positions and targets.

```
<script>
        // Load and add point cloud to scene
        Potree.loadPointCloud("../pointclouds/vol_total/cloud.js", "sigeom.sa", e => {
                let scene = viewer.scene;
                let pointcloud = e.pointcloud;

                let material = pointcloud.material;
                material.size = 1;
                material.pointSizeType = Potree.PointSizeType.ADAPTIVE;
                material.shape = Potree.PointShape.SQUARE;
                material.pointColorType = Potree.PointColorType.RGB;

                scene.addPointCloud(pointcloud);

                viewer.fitToScreen();
                scene.view.setView(
                        [589974.341, 231698.397, 986.146],
                        [589851.587, 231428.213, 715.634],
                );
        });
</script>
```

Note: A useful trick for setting easily the view on the model is to click on Other/Camera in the Scene section of the Potree Sidebar. Once selected, this option will activate a new properties box with camera position and target coordinates dynamically changing as the user navigate through the model in the viewer. Those coordinates can easily be copied and pasted in the code.

5.1.2 CA13 (18 billion points)

Working example

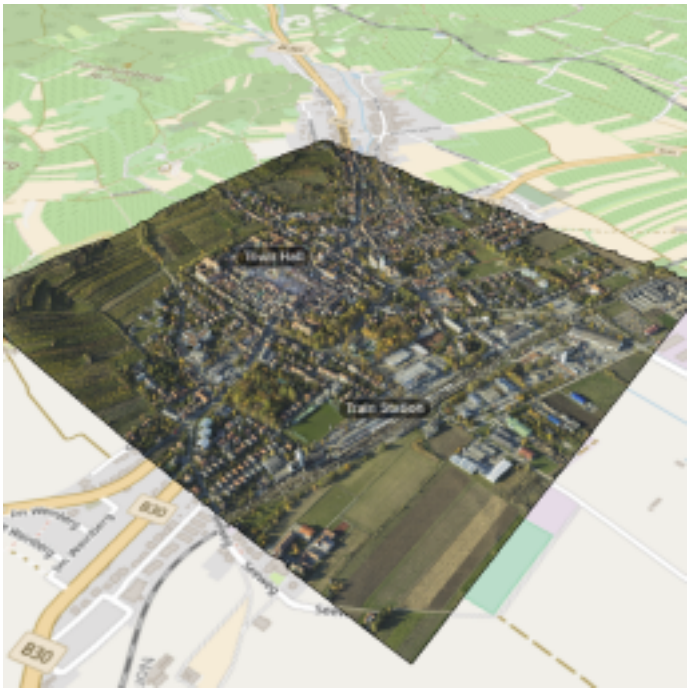
After cloning the Potree develop repository as suggested in section [reference], navigate to the *examples* folder and search for the ca13.html file. This file template includes the basic settings for a functional Potree Viewer that includes a point cloud with more than 18 billion points.

[TESTO]

```
<b><a href="https://labmgf.dica.polimi.it/">link</a></b> - Example of code-block.
```

5.1.3 Retz (Potree + Cesium)

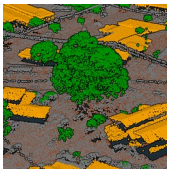
Working example



[TESTO]

5.1.4 Classifications

Working example



After cloning the Potree develop repository as suggested in section [reference], navigate to the *examples* folder and search for the classifications.html file. This file template includes the basic settings for a basic Potree Viewer (*Basic Viewer*) that includes a classified point cloud.

The main changes to the basic template consists in the definition of the type of visualisation for the pointcloud. In particular, this refers to the scalar field information to be considered when choosing the appearance of the point cloud. In this case, it is needed to show the classification of the point cloud, illustrating different classes of the territory like *ground*, *building*, *water*, *low vegetation* etc. To do so the active attribute for the point cloud material (*material.activeAttributeName*) should be set to the scalar field containing the required information to be shown (in this case, "classification".)

```
Potree.loadPointCloud("http://5.9.65.151/mschuetz/potree/resources/pointclouds/
↪opentopography/CA13_1.4/cloud.js", "CA13", function(e){
                    viewer.scene.addPointCloud(e.pointcloud);
                    e.pointcloud.position.z = 0;
                    let material = e.pointcloud.material;
                    material.size = 3;
```

(continues on next page)

(continued from previous page)

```
                material.pointSizeType = Potree.PointSizeType.FIXED;
                material.activeAttributeName = "classification";

                viewer.scene.view.position.set(694274.518, 3916261.987, 348.732);
                viewer.scene.view.lookAt(694683.097, 3916386.916, 30.879);
        });
```

Note: To be sure that the point cloud properties are correctly defined, before converting it, check if one of its scalar field is correctly named as "classification" (in software like CloudCompare..).

The working example additionally shows how to create button with functions for changing the classification scheme and color. This is achieved by first defining the html code for the buttons inside the potree container div element. Indeed, 3 buttons are defined through as many input tags:

1. **Default Scheme** (with onclick function *.setDefaultScheme()*) associated to the original classification scheme already defined in the potree.js source;
2. **Tree Scheme** (with onclick function *.setTreeScheme()*) linked to a new custom scheme highlighting only points belonging to one specific class;
3. **Random Scheme** (with onclick function *.setRandomScheme()*) with a function that randomly change colors of all classes.

```
<div class="potree_container" style="position: absolute; width: 100%; height: 100%;␣
↪left: 0px; top: 0px; ">
            <div id="potree_render_area" style="background-image: url('../build/potree/
↪resources/images/background.jpg');">

                <span style="position: absolute; bottom: 10px; left: 50%;␣
↪transform: translateX(-50%); z-index: 10000">
                    <input type="button" value="Default Scheme" onclick=
↪"setDefaultScheme()"/>
                    <input type="button" value="Tree Scheme" onclick=
↪"setTreeScheme()"/>
                    <input type="button" value="Random Scheme" onclick=
↪"setRandomScheme()"/>
                </span>

            </div>
            <div id="potree_sidebar_container"> </div>
    </div>
```

The set of operations executed by each function is later defined inside a script block.

```
import * as THREE from "../libs/three.js/build/three.module.js";

            window.setDefaultScheme = function(){
                viewer.setClassifications(Potree.ClassificationScheme.DEFAULT);
            }

            window.setTreeScheme = function(){
                viewer.setClassifications({
```

(continues on next page)

(continued from previous page)

```
                        5:       { visible: true, name: 'trees'         , color: [0.
↪0, 1.0, 0.0, 1.0] },
                        DEFAULT: { visible: false, name: 'other' , color: [0.0, 0.
↪0, 0.0, 1.0] },
                    });
            }

            window.setRandomScheme = function(){
                const scheme = {};

                for(let i = 0; i < 32; i++){
                    scheme[i] = { visible: true, name: `random:_${i}`, color:␣
↪[Math.random(), Math.random(), Math.random(), 1.0] };
                }

                viewer.setClassifications(scheme);
            }
```

First, *.setDefaultScheme()* is defined by giving as input to the *.setClassifications()* method the standard classification scheme defined by default in the potree.js source code as follows:

```
const ClassificationScheme = {

            DEFAULT: {
                0:       { visible: true, name: 'never classified'  , color: [0.5,␣
↪ 0.5,  0.5,  1.0] },
                1:       { visible: true, name: 'unclassified'      , color: [0.5,␣
↪ 0.5,  0.5,  1.0] },
                2:       { visible: true, name: 'ground'            , color: [0.63,
↪ 0.32, 0.18, 1.0] },
                3:       { visible: true, name: 'low vegetation'    , color: [0.0,␣
↪ 1.0,  0.0,  1.0] },
                4:       { visible: true, name: 'medium vegetation' , color: [0.0,␣
↪ 0.8,  0.0,  1.0] },
                5:       { visible: true, name: 'high vegetation'   , color: [0.0,␣
↪ 0.6,  0.0,  1.0] },
                6:       { visible: true, name: 'building'          , color: [1.0,␣
↪ 0.66, 0.0,  1.0] },
                7:       { visible: true, name: 'low point(noise)'  , color: [1.0,␣
↪ 0.0,  1.0,  1.0] },
                8:       { visible: true, name: 'key-point'         , color: [1.0,␣
↪ 0.0,  0.0,  1.0] },
                9:       { visible: true, name: 'water'             , color: [0.0,␣
↪ 0.0,  1.0,  1.0] },
                12:      { visible: true, name: 'overlap'           , color: [1.0,␣
↪ 1.0,  0.0,  1.0] },
                DEFAULT: { visible: true, name: 'default'           , color: [0.3,␣
↪ 0.6,  0.6,  0.5] },
            }
    };
```

In this way, for instance, each point containing the 0 value in the scalar field previously selected in the html code, will be considered as “never classified” and will be visible in the scene in the given RGBA color. If not explicitly stated in

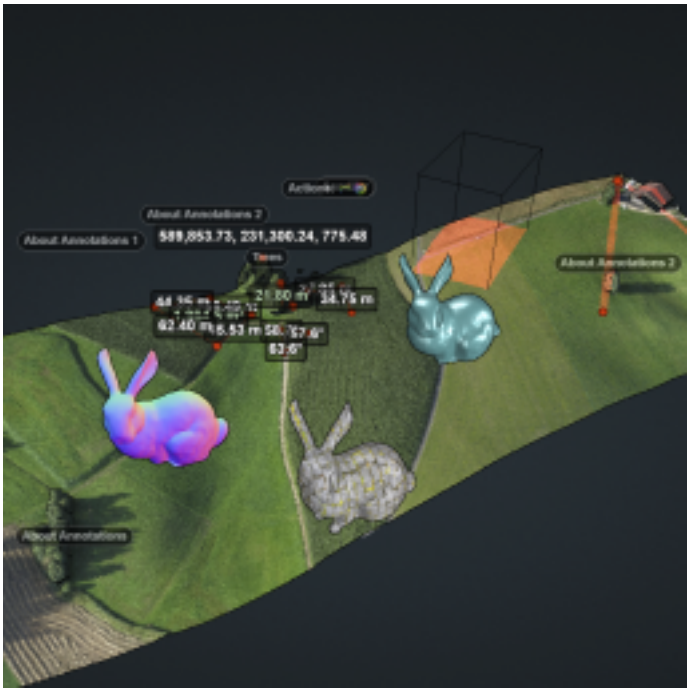
the scheme with a number, a point value will be assigned to the default class.

Such scheme is altered with the definition of the *.setTreeScheme()* in which all the points associated to the classification value 5 (originally "high vegetation") are assigned to a new class called "tree". All the other values are instead left in the other class, whose visibility is set as false (hidden).

Eventually, the *.setRandomScheme()* function, after defining a new empty scheme object, loop through 33 consecutive values and assign them a random triplet of RGBA colors.

5.1.5 Various Features

Working example



[TESTO]

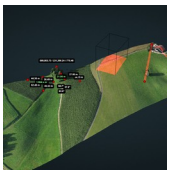
5.1.6 Toolbar

Working example

[TESTO]

5.1.7 Load Project

Working example



[TESTO]

5.1.8 Matcap

Working example

[TESTO]

5.1.9 Virtual Reality

Working example

[TESTO]

5.1.10 Heidentor

Working example

[TESTO]

5.1.11 Lion

Working example

After cloning the Potree develop repository as suggested in section [reference], navigate to the *examples* folder and search for the lion.html file. This file template includes the basic settings for a functional Potree Viewer (*Basic Viewer*) equipped with examples of custom sidebar.

This example show the rendering of the lion point cloud.

5.1.12 Lion LAS

Working example

After cloning the Potree develop repository as suggested in section [reference], navigate to the *examples* folder and search for the lion_las.html file. This file template includes the basic settings for a functional Potree Viewer (*Basic Viewer*) equipped with examples of custom sidebar.

This example show the rendering of the lion LAS point cloud in a basic viewer.

5.1.13 Lion LAZ

Working example

After cloning the Potree develop repository as suggested in section [reference], navigate to the *examples* folder and search for the lion_laz.html file. This file template includes the basic settings for a functional Potree Viewer (*Basic Viewer*) equipped with examples of custom sidebar.

This example show the rendering of the lion LAZ compressed point cloud in a basic viewer.

5.1.14 EPT

Working example

[TESTO]

5.1.15 EPT Binary

Working example

[TESTO]

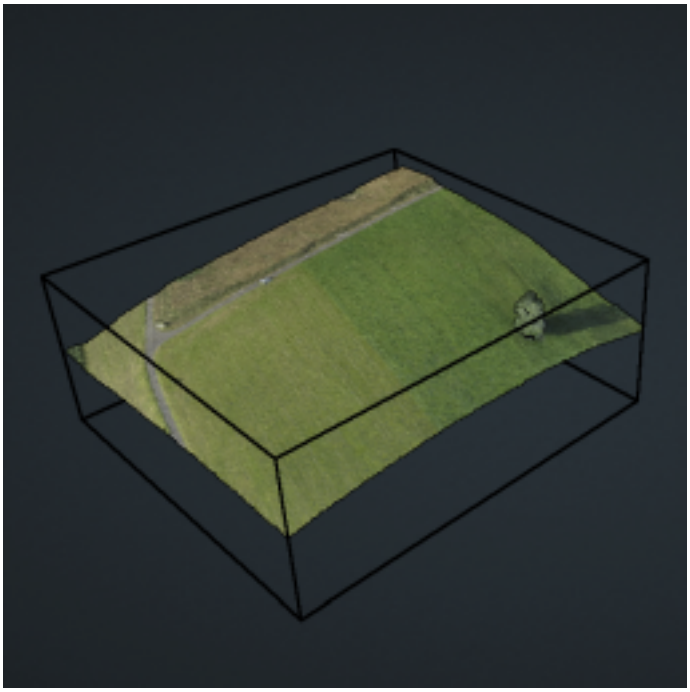
5.1.16 EPT zstandard

Working example

[TESTO]

5.1.17 Clipping Volume

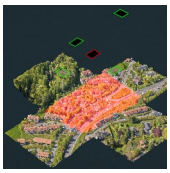
Working example



[TESTO]

5.1.18 Oriented Images

Working example



After cloning the Potree develop repository as suggested in section [reference], navigate to the *examples* folder and search for the oriented_images.html file. This file template includes the basic settings for a functional Potree Viewer (*Basic Viewer*) equipped with examples of custom sidebar.

In addition to the standard code for the simple basic viewer, the html page include an additional script element with the required functions to load the oriented images. Indeed, first the script declare 2 constants:

- **cameraParamsPath** that is defined with a file path to the .xml document that contains the calibration parameters (focal length, image width, length etc.) of the camera used to capture the oriented images.
- **imageParamsPath** which refers to the file path of the .txt document in which the oriented images parameters are listed together with the corresponding image name files.

The given constants are then passed together with the viewer element into the *Potree.OrientedImageLoader.load()* that then adds the images to the Potree viewer and scene.

```
<script type="module">
  import * as THREE from "../libs/three.js/build/three.module.js";
  const cameraParamsPath = "http://5.9.65.151/mschuetz/potree/resources/pointclouds/
↪helimap/epalinges/img_selected/IXM35_190522_nodistortion.xml";
  const imageParamsPath = "http://5.9.65.151/mschuetz/potree/resources/pointclouds/
↪helimap/epalinges/img_selected/Calib190522_MN95_NF2_cam_estim.txt";

  Potree.OrientedImageLoader.load(cameraParamsPath, imageParamsPath, viewer).then(␣
↪images => {
    viewer.scene.addOrientedImages(images);
              });

      </script>
```

In order to run smoothly the entire process, it is important that the parameters file are formatted correctly.

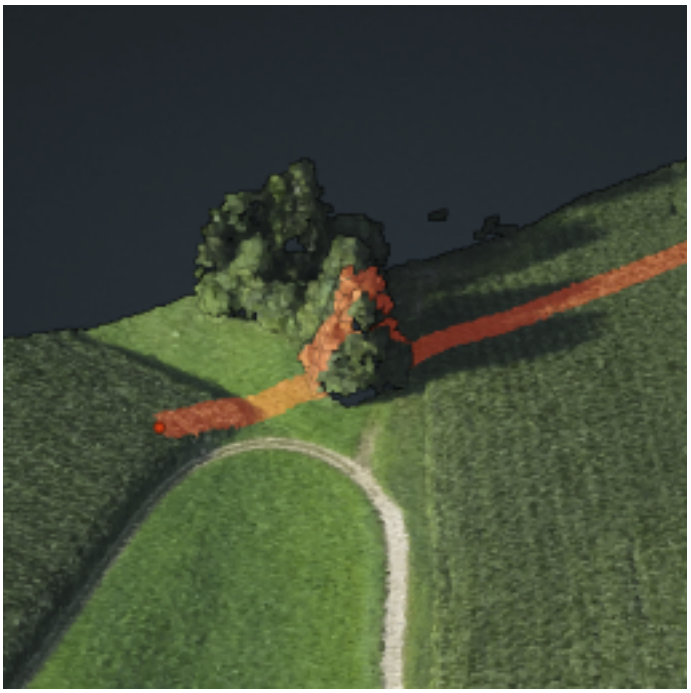
Note: In particular, the image parameters file should carefully respect the structure of the example below, with orientation angles indicated as Omega, Phi and Kappa.

```
# CoordinateSystem: PROJCS["CH1903+ / LV95",GEOGCS["CH1903+",DATUM["CH1903+",SPHEROID[
↪"Bessel 1841",6377397.155,299.1528128,AUTHORITY["EPSG","7004"]],TOWGS84[674.374,15.056,
↪405.346,0,0,0,0],AUTHORITY["EPSG","6150"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901
↪"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9102"]],AUTHORITY["EPSG","4150
↪"]],PROJECTION["Oblique_Mercator",AUTHORITY["EPSG","9815"]],PARAMETER["latitude_of_
↪center",46.95240555555561],PARAMETER["longitude_of_center",7.439583333333329],
↪PARAMETER["azimuth",90],PARAMETER["recitified_grid_angle",90],PARAMETER["scale_factor",
↪1],PARAMETER["false_easting",2600000],PARAMETER["false_northing",1200000],UNIT["metre",
↪1,AUTHORITY["EPSG","9001"]],AUTHORITY["EPSG","2056"]]
#Label        X/Easting       Y/Northing      Z/Altitude      Omega   Phi     Kappa   X_
↪est   Y_est   Z_est   Omega_est       Phi_est Kappa_est
47521.jpg     2541968.212303  1156043.192768  1004.707676     3.557724        3.273124  ␣
↪     -40.010726
47524.jpg     2542076.519349  1155923.975329  1009.489869     0.136003        4.313254  ␣
↪     -43.116678
47549.jpg     2542027.910785  1155984.936730  1013.691026     -2.697229       0.813820  ␣
↪     135.395195
47561.jpg     2542105.017595  1156065.259213  1014.761372     -0.735153       -2.127324 ␣
↪     -132.847907
47588.jpg     2541950.983809  1155914.133744  1100.203228     -0.849119       6.052296  ␣
↪     43.560809
```

[ADD DETAILS ON HOW TO EASILY CREATE THESE FILES WITH COMMERCIAL SOFTWARE (METASHAPE)]

5.1.19 Elevation Profile

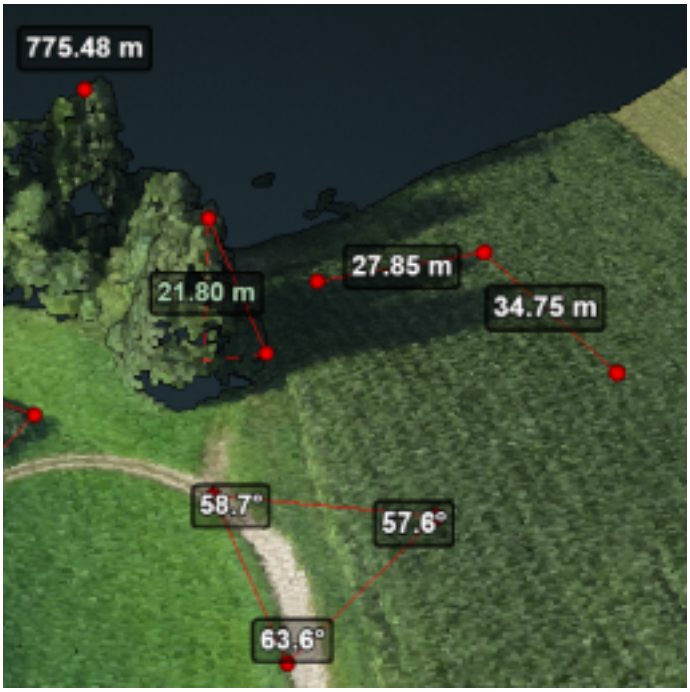
Working example



[TESTO]

5.1.20 Measurements

Working example



[TESTO]

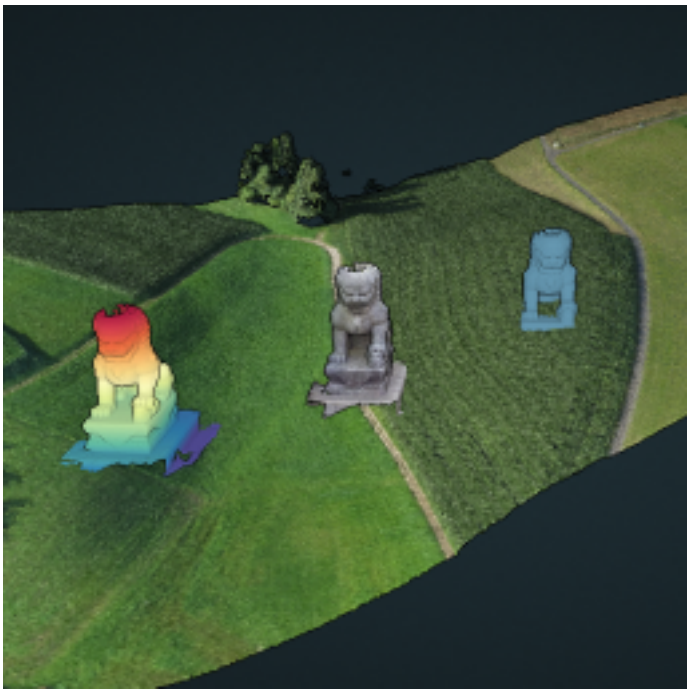
5.1.21 Meshes

Working example

[TESTO]

5.1.22 Multiple Point Clouds

Working example



[TESTO]

5.1.23 Camera Animation

Working example

After cloning the Potree develop repository as suggested in section [reference], navigate to the *examples* folder and search for the camera_animation.html file. This file template includes the basic settings for a functional Potree Viewer (*Basic Viewer*) equipped with examples of an implemented camera animation.

In order to include by default a camera animation to the Potree viewer, it is simply needed to include a script module in the html body script as follows:

```
<script type="module">

      import * as THREE from "../libs/three.js/build/three.module.js";

            const animation = new Potree.CameraAnimation(viewer);
```

(continues on next page)

(continued from previous page)

```
        const positions = [
                [590291.6145250637, 231565.3152460147, 888.181158774433],
                [590094.2454560432, 231235.32163877538, 870.7535717968211],
                [589675.8154371583, 231058.22066649256, 905.3068746322883],
                [589328.6700949036, 231385.37585641106, 813.9565903445384],
        ];

        const targets = [
                [589859.3465488373, 231456.18943956672, 758.2733646218901],
                [589846.4463098792, 231431.89813285187, 755.9090168440739],
                [589824.0843049305, 231444.72309070674, 760.3459659610106],
                [589799.7263767472, 231473.79043369304, 758.8332698380435],
        ];

        for(let i = 0; i < positions.length; i++){
                const cp = animation.createControlPoint();

                cp.position.set(...positions[i]);
                cp.target.set(...targets[i]);
        }

        viewer.scene.addCameraAnimation(animation);


</script>
```

This code, after defining coordinates for camera positions and targets - that can be achieved as suggested in the basic viewer note -, create the control point of the animation upon which the animation trajectory is built. Then, the animation is added to the scene with the *.addCameraAnimation()* method.

5.1.24 Features (CA13)

Working example

[TESTO]

5.1.25 Annotations

Working example

After cloning the Potree develop repository as suggested in section [reference], navigate to the *examples* folder and search for the annotations.html file. This file template includes the basic settings for a functional Potree Viewer (*Basic Viewer*) equipped with examples of interactive annotations.

In this example, first 2 distinct scenes are defined (**Potree.Scene()**). The first one (*sceneSG*) through the **.setScene()** method is set as the main one to be adopted when the viewer is loaded on the page. The other one will be later used as the target for an action embedded in an annotation.

```
let sceneSG = new Potree.Scene();
let sceneLion = new Potree.Scene();

viewer.setScene(sceneSG);
```

Then, the code includes the settings from the basic viewer for loading a pointcloud to the main scene previously named *sceneSG*. After that, a series of annotations is finally initialised. In particular the first one is defined as follow:

```
{
	let elTitle = $(`
		<span>
			About Annotations
			<img src="${Potree.resourcePath}/icons/goto.svg"
			name="action_set_scene"
			class="annotation-action-icon"
			style="filter: invert(1);" />
		</span>
	`);

	elTitle.find("img[name=action_set_scene]").click( (event) => {
		event.stopPropagation();
		viewer.setScene(sceneLion);
	});
	elTitle.toString = () => "About Annotations";
	let aAbout1 = new Potree.Annotation({
		position: [590043.63, 231490.79, 740.78],
		title: elTitle,
		cameraPosition: [590105.53, 231541.63, 782.05],
		cameraTarget: [590043.63, 231488.79, 740.78],
		description: `<ul><li>Click on the annotation label to move a predefined␣
↪view.</li><li>Click on the icon to execute the specified action.</li>In this case, the␣
↪action will bring you to another scene and point cloud.</ul>`
	});

	sceneSG.annotations.add(aAbout1);
}
```

In this example, first the title (*elTitle*) to be shown in the annotation label is defined. In this variable HTML is supported to set style and semantics of the title, enabling also the inclusion of images (with the img tag) linked to action. Indeed, by defining a name for the image element, later is possible to implement an action when the label is clicked by the user. This happens by defining a series of operations to be executed when the image named *action_set_scene* is clicked: all the desired operations should then be included with the final brackets of **elTitle.find("img[name=action_set_scene]").click()**. In this case, the action triggers the initiation of another scene (**viewer.setScene(sceneLion)**), calling the previously defined *sceneLion*.

Finally, a simple string name for the annotation is defined with **.toString** in order to have it implemented in the Scene tree in the Potree Sidebar.

Eventually, the **Potree.Annotation** object is finalised by declaring its *position*, *title* (linking to the previously declared variable), *cameraPosition*, *cameraTarget* and *description*. The desired camera position and camera target can be defined using the camera object in the scene list of the Potree Sidebar and are needed to define the view on the annotation once the label is clicked by the user.

Then, it's time to add the annotation to the main scene with **sceneSG.annotations.add(aAbout1)**.

[TO BE COMPLETED]

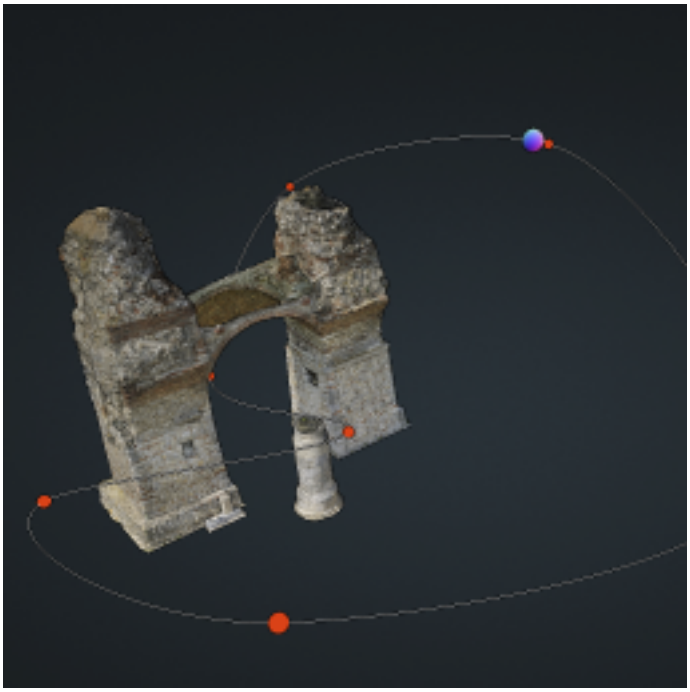
5.1.26 Hierarchical Annotations

Working example

[TESTO]

5.1.27 Animation Paths

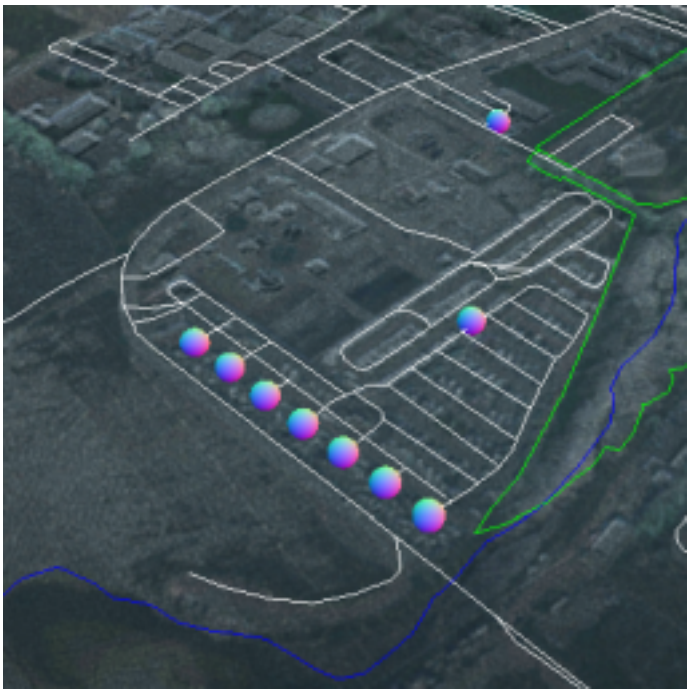
Working example



[TESTO]

5.1.28 Shapefiles

Working example



[TESTO]

5.1.29 Cesium CA13

Working example

[TESTO]

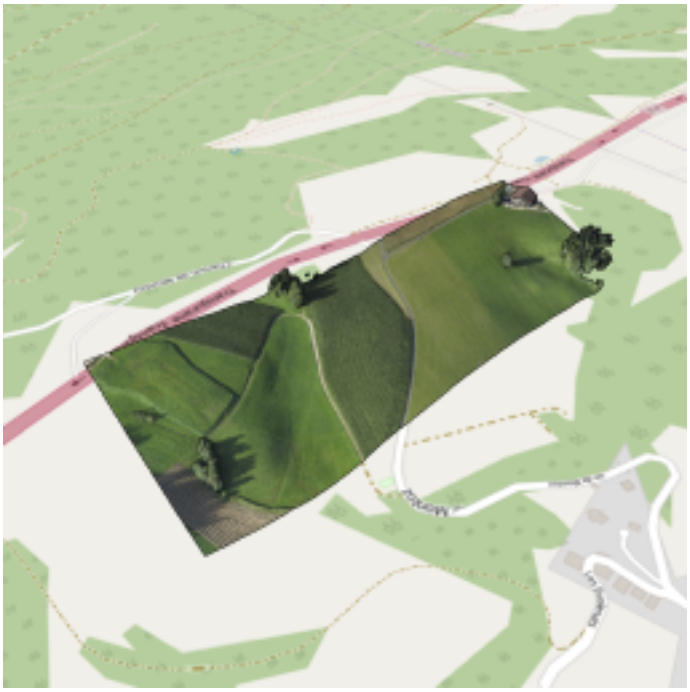
5.1.30 Geopackage

Working example

[TESTO]

5.1.31 Cesium Sorvilier

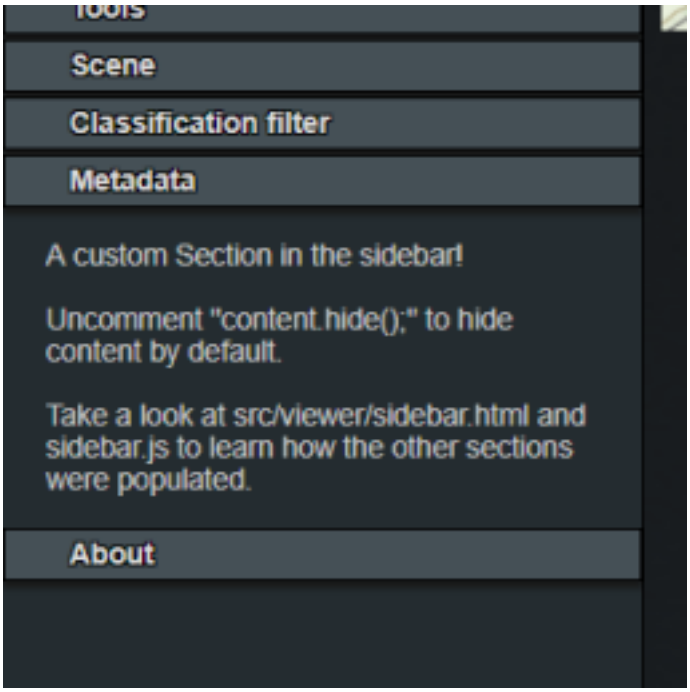
Working example



[TESTO]

5.1.32 Custom Sidebar Section

Working example



After cloning the Potree develop repository as suggested in section [reference], navigate to the *examples* folder and search for the custom_sidebar_section.html file. This file template includes the basic settings for a functional Potree Viewer (*Basic Viewer*) equipped with examples of custom sidebar.

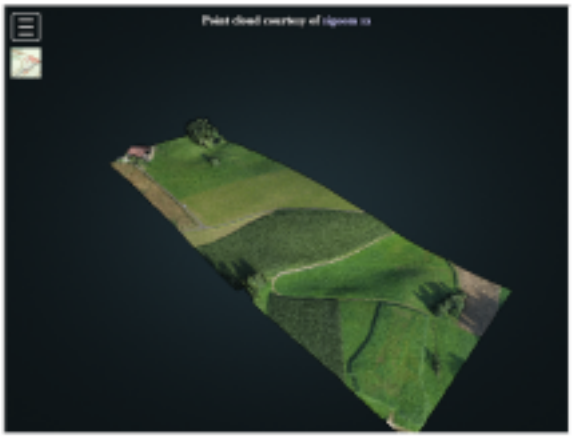
In the **body** section inside the script block, after defining the basic settings for initialising the viewer, the sidebar section are called inside the **.loadGUI()** function: First the new section name is declared with a variable (let section). To define its name and keeping the style coherent with the rest of the sidebar sections, simply substitute *Metadata* with the desired title. Advanced style settings can be performed as well accoring to CSS sintax. Then the title is attached to the content variable (let content). Finally the the section is populated with an HTML-compatible content (content.html()). This can include texts as well as other media. The new section is then inserted in the standard sidebar by first setting the toggling functionaly on click (*.slideToggle()*) and then indicating its order position in relation to the other sections (*.insertBefore()*). The visibility at first page loading is set by including the code the section variable name followed by *.hide()*.

```
<script type="module">
  viewer.loadGUI(() => {
                        viewer.toggleSidebar();

                        let section = $(`
                                <h3 id="menu_meta" class="accordion-header ui-widget">
↪<span>Metadata</span></h3>
                                <div class="accordion-content ui-widget pv-menu-list"></
↪div>
                        `);
                        let content = section.last();
                        content.html(`
                        <div class="pv-menu-list">
                                A custom Section in the sidebar!<br>
                                <br>
                                Uncomment "content.hide();" to hide content by default.<br>
                                <br>
                                Take a look at src/viewer/sidebar.html and sidebar.js to
                                learn how the other sections were populated.
                        </div>
                        `);
                        section.first().click(() => content.slideToggle());
                        section.insertBefore($('#menu_about'));

                });
</script>
```

[TESTO]

5.1.33 Embedded iframe

Working example



This example simply illustrates how to set up a webpage with a Potree Viewer embedded in iframe HTML element. The example code is available in the *examples* folder in the embedded_iframe.html file

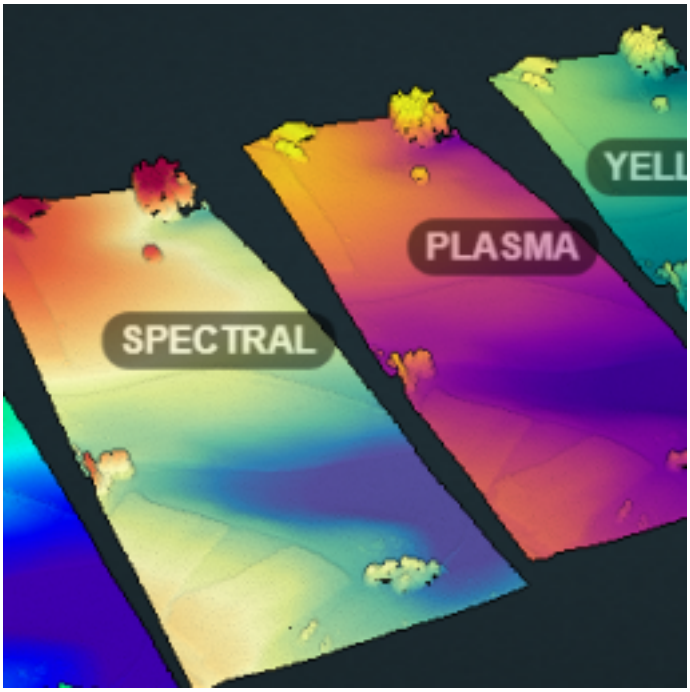
The iframe implementation is possible once a proper viewer is set up in a dedicated html file. In the case of the code example, the iframe is included in the **body** section in a **div** element and refers to the Potree viewer set in the *Basic Viewer* example. In order to change the viewer page to be embedded, subtitute *viewer.html* in the code below with the name of the html page in which you defined a Potree viewer.

```
<html>
  <head></head>
  <body>
    <div style="position: absolute; left: 20%; right: 20%; top: 20%; bottom: 20%">
      <iframe src="viewer.html" style="width: 100%; height: 100%"></iframe>
    </div>
  </body>
</html>
```

In the div element additional settings are defined, including the style and the position of the element within the page following CSS standards.

5.1.34 Gradient colors

Working example



[TESTO]

// UNDER CONSTRUCTION //

CHAPTER

SIX

POTREE CULTURAL HERITAGE TEMPLATE

This page will cover details on how to build a web page with a Potree viewer using a mix of already available functionalities and custom tools.

Repository with example: Tars4815/potree-chtemplate

6.1 About

This page illustrates custom features and their implementation for an updated template for Potree pages based on Vizcaya Museum project and inspired by potree-sfm project features.

A working example can be found on the LabMGF DICA PoliMi website.

6.1.1 Features

- Fullscreen HTML5 API
- Bootstrap integration
- Simplified navigation
- Accessible
- Informational overlays (custom buttons and annotation settings)
- Slowed annotation camera rotations
- Animations from annotations
- Oriented images on the model

6.1.2 License

For continuity sake, this project carries the same license as the original Potree project.

6.1.3 Prerequisites

Make sure you have the following installed and working in order to reproduce the project. You can view detailed installation guidelines within the following links:

- Xampp / Apache server
- Github to clone project

6.1.4 Performance

Examples work best and with Google Chrome and Firefox.

6.2 Folder structure

Testo da scrivere

[immagine schema]

6.2.1 Assets

Testo

6.2.2 Libs

Testo

6.2.3 Licenses

Testo

// UNDER CONSTRUCTION //

CHAPTER

SEVEN

USEFUL RESOURCES

// Under construction //

7.1 Video Tutorials

- Deploy Point Cloud with Potree by Klas Karlsson
- Potree Workflow Playlist by IQ Center
- Potree Converter/Desktop Playlist [ITA] by Paolo Corradeghini

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search