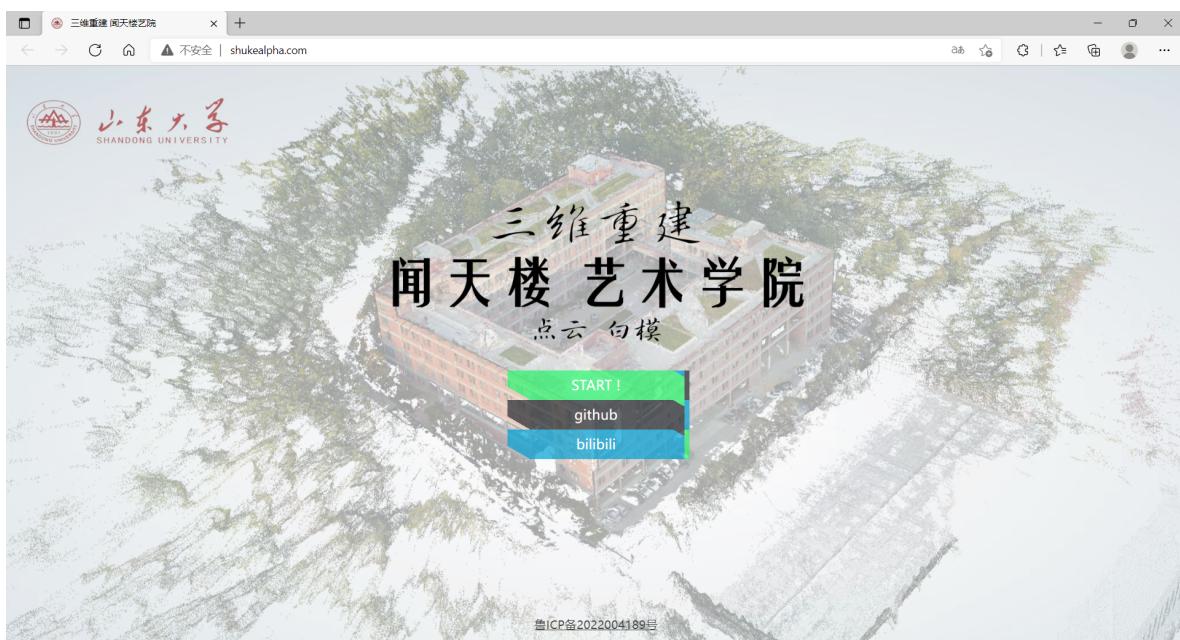


闻天楼和艺术学院的点云及白模网页端展示

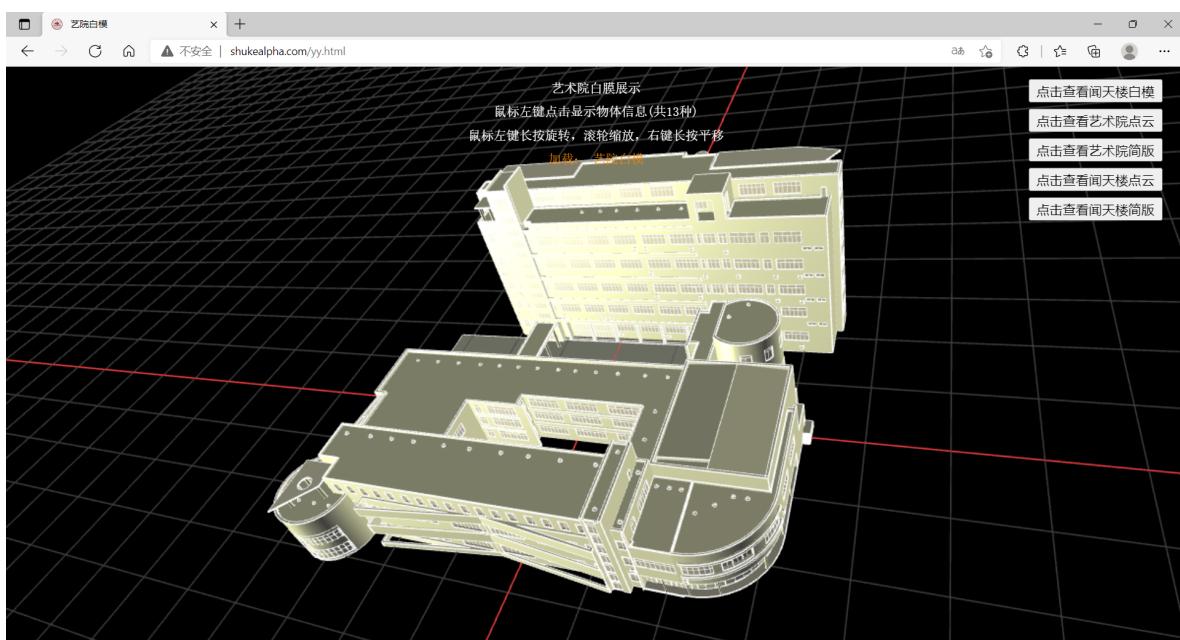
本文主要通过已有的山东大学威海校区闻天楼和艺术学院与科学实验楼的点云数据达到以下目标：

- 1.结合“计算机图形学和数字孪生”课程所学知识，使用 OpenGL 程序包（有 python 包和 webGL 可以用，也可以采用其他基于 OpenGL 的 python 和网页端包），用多个参数化的三维几何模型（如棱柱等）重画建筑物外表白模，还原程度越高分。
- 2.每组自行购买一个学生云主机，建立一个网站，把展示出来重建的三维点云模型（注意使用一些点云插件加快加载速度）和三维矢量白模，均可实现用鼠标随意移动翻转模型。
- 3.在网站上，鼠标可以点击某建筑白模任何位置，可以展示出第一部分识别的数据和几何信息。

下图为网站开始界面，分别进入模型展示、github文件及bilibili视频文件：



点击 START，即可进入模型展示界面（以艺术学院白模为例）：



1 点云

1.1 点云模型

之前的项目中我们使用SFM和MVS两种方法实现了点云模型的构建。其中SFM是MVS的上游。一个可以理解为稀疏重建，另一个可以理解为稠密重建。SFM给MVS算好了输入视角的位姿、内参、稀疏点以及它们的共视关系，MVS再利用这些信息以及彩色图来估计深度图以及做最后的操作。

在本次模型可视化展示中，我们使用了之前生成的稀疏重建点云模型，在网页上展示。在这里简要说明一下稀疏点云的生成。



1.1.1 特征提取

特征提取具体是在不同的尺度空间上查找关键点(特征点)，并计算出关键点的方向。所查找到的关键点是一些十分突出、不会因光照、仿射变换和噪音等因素而变化的点，如角点、边缘点、暗区的亮点及亮区的暗点等。

在一定阈值的条件下，在灰度图中提取出图中的关键点。这些关键点有如下特点，能代表图像的局部特征，具有平移旋转的不变性，信息量丰富，处理速度快等。特征点的提取有助于图像之间的特征向量之间的匹配。

1.1.2 特征匹配

在特征匹配中，业内常用的大致是四中方法：

Exhaustive：官方文档中说明Exhaustive耗时最长，图像两两匹配，理论上效果最好。但在实际发现这种方式耗时长而且效果未必好，对于场景位置的判断时不时出现失误。比如局部拍摄时无法确定某些图片的位置，场景相似时会出现误判。

Sequential：拍照时按照时间顺序，相邻的场景往往在一块，适合使用这种匹配方式。以相邻x张图像匹配，上面的问题就不会遇到。但在切换角度过大，或者拍摄开始和拍摄结束是同一个视角时会有误差。

Spatial：这种方式是利用了地理位置信息，也就是每张图像必须自带位置信息，例如gps。这种方式需要设置在多大范围内进行匹配，因此需要事先知道大概重建场景大小，以便选择合适参数。

Custom：自定义匹配方式，根据自身需求进行参数调整。本文结合实际运行效果、运行速度及自身情况。主要采用前两种匹配方法，以求速度与准确度相结合。

1.1.3 稀疏重建

采用增量式SfM（出自openMVG: open Multiple View Geometry library. Basis for 3D computer vision and Structure from Motion），逐步增加视角，并进行迭代优化重投影误差，计算不同视图的相机参数、得到场景的稀疏点云和确定不同视图与点云间的可视关系，最后可以得到场景的稀疏点云和各个视角的相机姿态。



1.1.4 稠密重建

通过去畸变、深度估计生成深度图等流程，将估计出来的深度图进行融合，得到稠密重建结果：

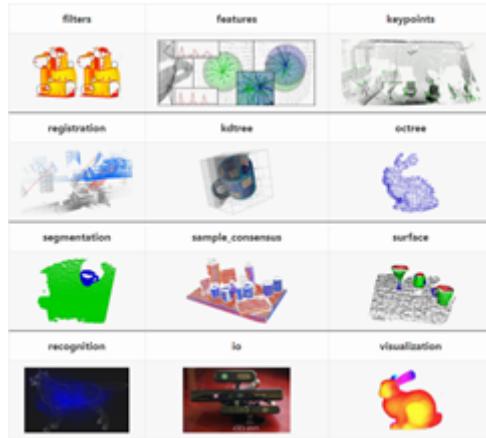


1.2.1 点云格式转换

PCL (Point Cloud Library) 是在吸收了前人点云相关研究基础上建立起来的大型跨平台开源编程库，它实现了大量点云相关的通用算法和高效数据结构，涉及到点云获取、滤波、分割、配准、检索、特征提取、识别、追踪、曲面重建、可视化等。支持多种操作系统平台，可在Windows、Linux、Android、Mac OS X、部分嵌入式实时系统上运行。如果说OpenCV是2D信息获取与处理的结晶，那么PCL就在3D信息获取与处理上具有同等地位。

PCL是指纳入了多种操作点云数据的三维处理算法，其中包括：过滤，特征估计，表面重建，模型拟合和分割，定位搜索等。每一套算法都是通过基类进行划分的，试图把贯穿整个流水线处理技术的所有常见功能整合在一起，从而保持了整个算法实现过程中的紧凑和结构清晰，提高代码的重用性、简洁可读。

在本次项目中，我们使用了pcl库对colmap处理出的模型进行格式转换。



1.2.2 点云格式

Ply格式：

在colmap中我们处理得到的点云模型为ply格式。PLY是一种电脑档案格式，全名为多边形档案（Polygon File Format）该格式主要用以储存立体扫描结果的三维数值，透过多边形片面的集合描述三维物体，与其他格式相较之下这是较为简单的方法。它可以储存的资讯包含颜色、透明度、表面法向量、材质座标与资料可信度，并能对多边形的正反两面设定不同的属性。每个PLY档都包含档头(header)，用以设定网格模型的“元素”与“属性”，以及在档头下方接着一连串的元素“数值资料”。一般而言，网格模型的“元素”就是顶点(vertices)、面(faces)，另外还可能包含有边(edges)、深度图样本(samples of range maps)与三角带(triangle strips)等元素。

Pcd格式：

Pcd格式同样储存三维物体的坐标数值，但由于很多格式在现有的文件结构因本身组成的原因不支持由PCL库引进n维点类型机制处理过程中的某些扩展，而pcd可以补足这个问题。因此在一些情况下，需要我们将其他格式转化为PCD格式。

Txt格式：

Txt也是点云中一种常见的数据保存格式。通常保存了各个点的三维坐标等信息。

我们在colmap中得到的ply的点云模型，但综合考虑了文件类型和后期操作方便，我们组用到了pcl库将点云转化为了pcd格式，最终能在网页中显示。

2 白模

2.1 mesh-posion

泊松重建是Michael Kazhdan等在2006年提出的网格重建方法，其文章题目为“Poisson Surface Reconstruction”。其输入是带有法向量属性的点云信息数据（也可以有RGB信息），输出是三角网格模型。

Posion重建是一个非常直观的方法。它的核心思想是点云代表了物体表面的位置，其法向量代表了内外的方向。通过隐式地拟合一个由物体派生的指示函数，可以给出一个平滑的物体表面的估计。

给定一个区域M及其边界 ∂M ，指示函数 χ_M 定义为

$$\chi_M(x) = \begin{cases} 1 & x \in M \\ 0 & x \notin M \end{cases}$$

这样，把重构 $S=\partial M$ 的问题转换为重构 χ_M 的问题。

简单列几点：

- Poisson在边缘处的锐度比VRIP要好。这是因为VRIP在大的边缘处TSDF的累加会有平滑效应，而Poisson依据的是法向量，不会引入额外的平滑。
- VRIP是局部方法，每次只更新当前深度图对应的TSDF。Poisson是全局方法。
- 从个人使用经验上看，Poisson对于噪声更加鲁棒一些。点云法向量估计的精度不能太差。
- 如果重建出奇怪的形状（分层、分块），请查看原始点云是否平滑，是否有噪声，调整生成网格的分辨率以适应点云。

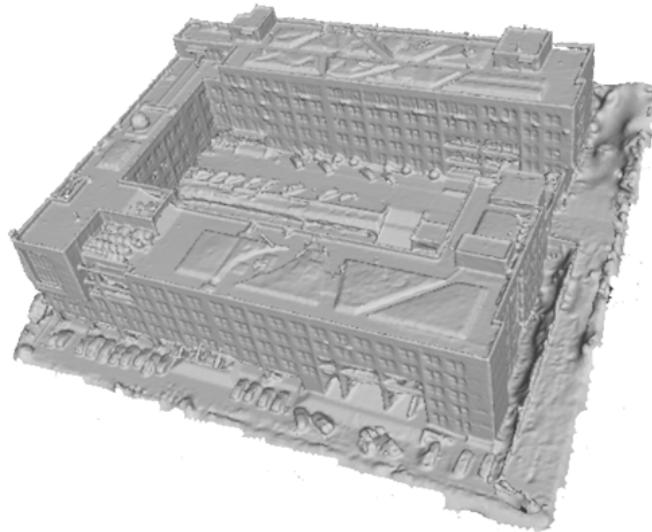
如下展示该算法下的三维模型：



2.2 mesh-delaunay

Delaunay 细化算法构建三角形或四面体（"元素"）的网格，适用于插值、渲染、地形数据库、地理信息系统等应用，以及要求最高的有限元方法偏微分方程的求解。Delaunay 优化算法通过维护 Delaunay 或受约束的 Delaunay 三角测量来运行，该三角测量通过插入其他顶点进行优化，直到网格满足对元素质量和大小的限制。这些算法提供了元素质量、边长和元素大小的空间分级的理论边界。复杂域的拓扑和几何保真度，包括具有内部边界的弯曲域；并在实践中真正令人满意的表现。

如下展示该算法下的三维模型：



2.3 结合相机照片信息添加纹理

在先前的两个方法中，我们输出的结果中有一个.out文件，存储着每个相机的位置及重建出的稀疏点云以及一组(个).ply文件，存储着由稀疏点云重建出的稠密点云。为了进一步消除死角和畸变，让模型更加真实，下面使用Meshlab，过一系列操作可创建出包含纹理的、干净的、高分辨率的网格，并自动计算UV映射及创建纹理图像。

其操作过程如下：

1. 打开bundle.rd.out 和list.txt文件

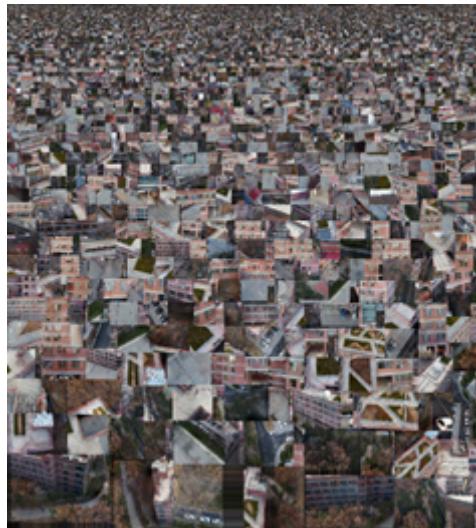
2. 生成稠密点云代替稀疏点云

3. 网格化，利用Poisson Surface Reconstruction算法由稠密点云生成多边形网格表面。Octree Depth控制着网格的细节，此值越大细节越丰富但占内存越大运行起来慢，一般设10，可慢慢调大。
(其中，在实际网格化过程中，Octree Depth=12)

4. 修复流形边缘，后续的纹理处理要求网格化的模型必须是流形(MANIFOLD)的，因此需删除非流形边。



5. 参数化 (Parameterization)，具体过程为：Filter-> Texture -> Parameterization from registered rasters。以艺术学院与科学实验楼为例，我们得到了UV映射如下：



6. 投影纹理，具体过程为：Filter-> Texture -> Project active rasters color to current mesh, filling the texture。该过程可以设置任意分辨率（512的2的二次方倍：512 / 1024 / 2048 / 4096 / 8192...）的纹理图。本文生成的分辨率是1024。

2.4 使用sketchup细化白模

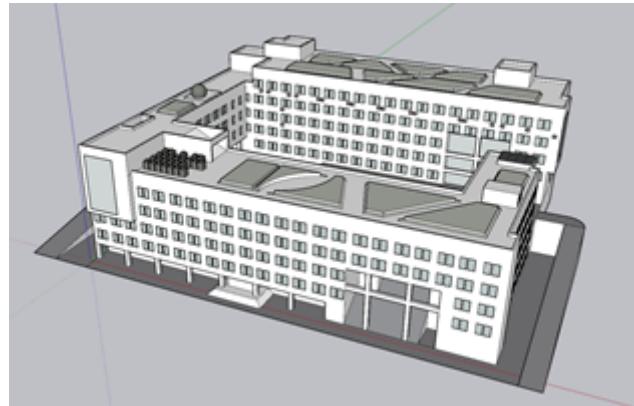
2.4.1 白模的不足

用之前方法构成的白模有一些不足，例如模型细节不明显，楼梯的每一级台阶不能很好的显示出来，又例如树木的遮挡，导致模型中会混入树的结构。因此建立的模型尽管能很好的反应建筑的比例与形状，但细节需要人为的改进。

2.4.2 sketchup

Sketchup是一款智能的3D模型绘制软件，通过简单的绘制直线，延伸面等操作绘制三维模型，并可以为模型渲染材质。

我们组根据已有点云模型，构建了建筑的坐标参考系，按照比例大小复原了建筑的主体。再根据照片等信息，找到点云不能体现的细节，逐一在模型中绘制。对于楼梯、窗台、花坛、空调外机等均得到复原。



最后添加材质，在模型中为绿化带、窗户、空调外机、道路等分别加上材质，并统计相关数据，以便在网页展示时显示相关物体的数据。

3 Web显示

3.1 Three.js

Three.js 是一款 WebGL 框架，由于其易用性被广泛应用。Three.js 在 WebGL 的 API 接口基础上，又进行的一层封装。它是由居住在西班牙巴塞罗那的程序员 Ricardo Cabbello Miguel 所开发，他更为人知的网名是 Mr.doob。

Three.js 以简单、直观的方式封装了 3D 图形编程中常用的对象。Three.js 在开发中使用了很多图形引擎的高级技巧，极大地提高了性能。另外，由于内置了很多常用对象和极易上手的工具，Three.js 的功能也非常强大。最后，Three.js 还是完全开源的，可以在 GitHub 上找到它的源代码，并且有很多人贡献代码，帮助 Mr.doob 一起维护这个框架。

The screenshot shows the official website for Three.js. On the left, there's a sidebar with links: 'three.js r137' (highlighted), 'Learn' (documentation, examples, editor), 'Community' (questions, discord, forum, slack, twitter), and 'Code' (github). To the right is a grid of 24 small images demonstrating various 3D projects made with Three.js, including a NASA logo, a futuristic interior, a planet, a soldier, a landscape, a person on a surfboard, a phone, a car, a tree, a person, a bicycle, a purple logo, a red flower, a white car, a white cube with a plant, a white board with text, a colorful abstract scene, and a dark geometric scene.

3.1.1 创建场景

在使用Three.js之前，需要在某个地方显示它。将以下 HTML 与 js/ 目录中的 three.js 的副本一起保存到计算机上的文件中，然后在浏览器中将其打开。

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My first three.js app</title>
    <style>
      body { margin: 0; }
    </style>
  </head>
  <body>
    <script src="js/three.js"></script>
    <script>
      // Our Javascript will go here.
    </script>
  </body>
</html>

```

就这样。下面的所有代码都进入空的标记。

为了能够真正用Three.js显示任何东西，我们需要三样东西：场景，相机和渲染器，以便我们可以用相机渲染场景。

```

const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera( 75, window.innerWidth /
window.innerHeight, 0.1, 1000 );

const renderer = new THREE.WebGLRenderer();
renderer.setSize( window.innerWidth, window.innerHeight );
document.body.appendChild( renderer.domElement );

```

现在，让我们使用**PerspectiveCamera**。

第一个属性是**视野**。FOV 是在任何给定时刻在显示屏上看到的场景范围。该值以度为单位。

第二个是**宽高比**。几乎总是希望使用元素的宽度除以高度，否则将获得与在宽屏电视上播放旧电影时相同的结果 - 图像看起来被挤压。

接下来的两个属性是**近剪裁平面**和**远剪裁平面**。这意味着，距离相机较远或距离较近的值较远的对象将不会被渲染。您现在不必担心这一点，但您可能希望在应用中使用其他值以获得更好的性能。

接下来是渲染器。除了我们在这里使用的WebGLRenderer之外，还有三.js附带了其他一些，通常用作使用较旧浏览器的用户或由于某种原因没有WebGL支持的用户的后备。

除了创建呈现器实例之外，我们还需要设置希望它呈现应用的大小。最好使用我们要用应用填充的区域的宽度和高度 - 在本例中为浏览器窗口的宽度和高度。对于性能密集型应用，还可以为 **setSize** 提供较小的值，如 **window.innerWidth/2** 和 **window.innerHeight/2**，这将使应用以四分之一大小呈现。

3.1.2 在本地运行内容

如果只使用程序化几何图形而不加载任何纹理，则网页应直接从文件系统工作，只需在文件管理器中双击HTML文件，它应该在浏览器中显示为工作（您将在地址栏中看到 `file:///yourFile.html`）。

许多编程语言都内置了简单的HTTP服务器。它们不像Apache或NGINX等生产服务器那样功能齐全，但是它们应该足以测试您的三.js应用程序。

一些代码编辑器具有插件，这些插件将按需生成一个简单的服务器。

- [Five Server](#) for Visual Studio Code.
- [Live Server](#) for Visual Studio Code.
- [Live Server](#) for Atom.

3.1.3 WEBGL 和 Three.js

WebGL 原生 API 是一种非常低级的接口，而且还需要一些数学和图形学的相关技术。对于没有相关基础的人来说，入门真的很难，Three.js 将入门的门槛降低了一大截，对 WebGL 进行封装，简化我们创建三维动画场景的过程。只要你有一定的 JavaScript 基础，有一定的前端经验，我坚信，用不了多长时间，三维制作会变得很简单。

用最简单的一句话概括：WebGL 和 Three.js 的关系，相当于 JavaScript 和 jQuery 的关系。

尽管这变得越来越不成问题，但某些设备或浏览器可能仍然不支持WebGL。以下方法允许您检查它是否受支持，如果不支持，则向用户显示一条消息。

将 <https://github.com/mrdoob/three.js/blob/master/examples/jsm/capabilities/WebGL.js> 添加到 javascript 中，并在尝试呈现任何内容之前运行以下内容。

```
if ( webGL.isWebGLAvailable() ) {  
  
    // Initiate function or other initializations here  
    animate();  
  
} else {  
  
    const warning = webGL.getWebGLErrorMessage();  
    document.getElementById( 'container' ).appendChild( warning );  
  
}
```

3.1.4 关于功能

Three.js 作为 WebGL 框架中的佼佼者，由于它的易用性和扩展性，使得它能够满足大部分的开发需求，Three.js 的具体功能如下：

Three.js 掩盖了 3D 渲染的细节：Three.js 将 WebGL 原生 API 的细节抽象化，将 3D 场景拆解为网格、材质和光源（即它内置了图形编程常用的一些对象种类）。

面向对象：开发者可以使用上层的 JavaScript 对象，而不是仅仅调用 JavaScript 函数。

功能非常丰富：Three.js 除封装了 WebGL 原始 API 之外，Three.js 还包含了许多实用的内置对象，可以方便地应用于游戏开发、动画制作、幻灯片制作、高分辨率模型和一些特殊的视觉效果制作。

速度很快：Three.js 采用了 3D 图形最佳实践来保证在不失可用性的前提下，保持极高的性能。

支持交互：WebGL 本身并不提供拾取（Picking）功能（即是否知道鼠标正处于某个物体上）。而 Three.js 则固化了拾取支持，这就使得你可以轻松为你的应用添加交互功能。

包含数学库：Three.js 拥有一个强大易用的数学库，你可以在其中进行矩阵、投影和矢量运算。

内置文件格式支持：你可以使用流行的 3D 建模软件导出文本格式的文件，然后使用 Three.js 加载，也可以使用 Three.js 自己的 JSON 格式或二进制格式。

扩展性很强：为 Three.js 添加新的特性或进行自定义优化是很容易的事情。如果你需要某个特殊的数据结构，那么只需要封装到 Three.js 即可。

支持HTML5 Canvas：Three.js 不但支持 WebGL，而且还支持使用 Canvas2D、Css3D 和 SVG 进行渲染。在未兼容 WebGL 的环境中可以回退到其它的解决方案。

虽然 Three.js 的优势很大，但是它也有它的不足之处：

1. 官网文档较为简短。
2. 国内的相关资源匮乏。
3. Three.js 所有的资料都是以英文格式存在，对国内的朋友来说又提高了门槛。
4. Three.js 不是游戏引擎，一些游戏相关的功能没有封装在里面，如果需要相关的功能需要进行二次开发。

3.1.5 Three.js 与其他库的对比

随着 WebGL 的迅速发展，相关的 WebGL 库也丰富起来，接下来介绍几个比较火的 WebGL 库。

与 Babylon.js 对比：

Babylon.JS 是最好的 JavaScript 3D 游戏引擎，它能创建专业级三维游戏。主要以游戏开发和易用性为主。与 Three.js 之间的对比：

Three.js 比较全面，而 Babylon.js 专注于游戏方面。

Babylon.js 提供了对碰撞检测、场景重力、面向游戏的照相机，Three.js 本身不自带，需要依靠引入插件实现。

对于 WebGL 的封装，双方做得各有千秋，Three.js 浅一些，好处是易于扩展，易于向更底层学习；Babylon.js 深一些，好处是易用扩展难度大一些。

Three.js 的发展依靠社区推动，出来的比较早，发展比较成熟，Babylon.js 由微软公司在2013推出，文档和社区都比较健全，国内还不怎么火。

与 PlayCanvas 对比：

PlayCanvas 是一个基于 WebGL 游戏引擎的企业级开源 JavaScript 框架，它有许多的开发工具能帮你快速创建 3D 游戏。与 Three.js 之间的对比：

PlayCanvas 的优势在于它有云端的在线可视化编辑工具。

PlayCanvas 的扩展性不如 Three.js。

最主要是 PlayCanvas 不完全开源，还商业付费。

与 Cesium 对比：

Cesium 是国外一个基于 JavaScript 编写的使用 WebGL 的地图引擎，支持 3D、2D、2.5D 形式的地图展示，可以自行绘制图形，高亮区域。与 Three.js 对比：

Cesium 是一个地图引擎，专注于 Gis，相关项目推荐使用它，其它项目还是算了。至于库的扩展，其它的配套插件，以及周边的资源都不及Three.js。

3.2 点云显示

3.2.1 threejs的点击事件

threeJS的点击，原理很简单。

将需要添加点击事件的对象放在一个数组里，生成点击事件。用户点击屏幕的时候，threejs会根据视角从触碰点会发射一条“激光”，激光扫到的所有记录在数组里的对象，都被会捕捉到。

```
var clickObjects=[];
clickObjects.push(obj1);
function initThreeClickEvent() {
    //点击射线
    var raycaster = new THREE.Raycaster();
    var mouse = new THREE.Vector2();
    document.getElementById("container").addEventListener('mousedown',
onDocumentMouseDown, false);
    function onDocumentMouseDown(event) {
        event.preventDefault();
        mouse.x = (event.clientX / renderer.domElement.clientWidth) * 2 - 1;
        mouse.y = -(event.clientY / renderer.domElement.clientHeight) * 2 + 1;

        raycaster.setFromCamera(mouse, camera);

        //总结一下，这里必须装网格，mesh，装入组是没有效果的
        //所以我们将所有的盒子的网格放入对象就可以了
        // 需要被监听的对象要存储在clickObjects中。
        var intersects = raycaster.intersectObjects(clickObjects);

        // console.log(intersects)
        if(intersects.length > 0) {
            // 在这里填写点击代码
            console.log("dianji");
            console.log(intersects[0].object)
            showDetailPage(intersects[0].object.name);

        }
    }
}
```

上面就是一个标准的点击事件的方法，这里需要注意几点：

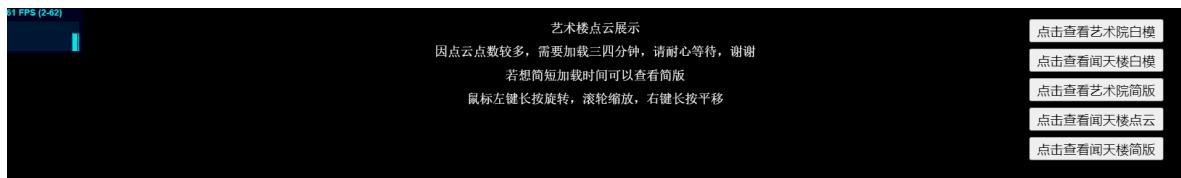
1. container是canvas的id名称,请换成自己的canvas。
2. clickObjects存储的都是mesh，对象实例，存储group是不会有反应的。
3. 在XXX被定义，生成的时候，调用scene.add(XXX)的时候，顺手一个
clickObjects.push(XXX)，就可以实现监听了。
4. 捕捉到的监听数量不唯一，intersects返回的是一个数组，被该激光打中的所有对象都会存储
在这个数组当中，选择你需要的提取出来下一步吧。
5. 即使点击空了，也会有点击回馈，intersects返回空数组。

3.2.2 点云展示

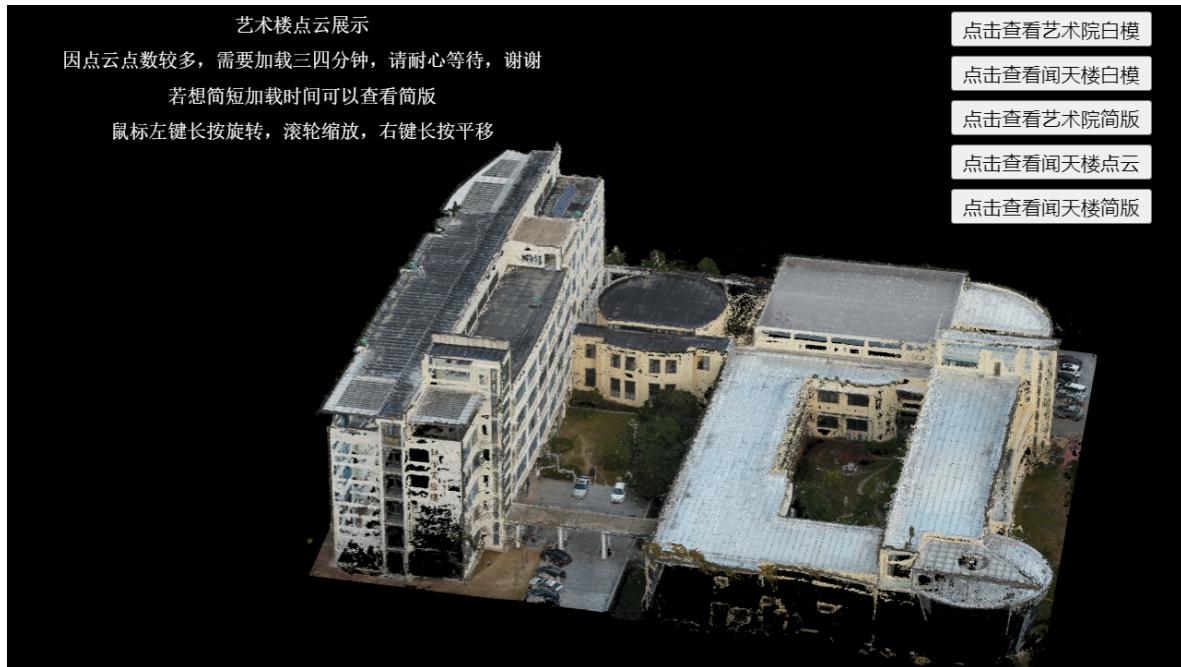
通过1得到目标建筑的pcd点云后，可以通过调试threejs实现web加载点云。

结果展示如下：

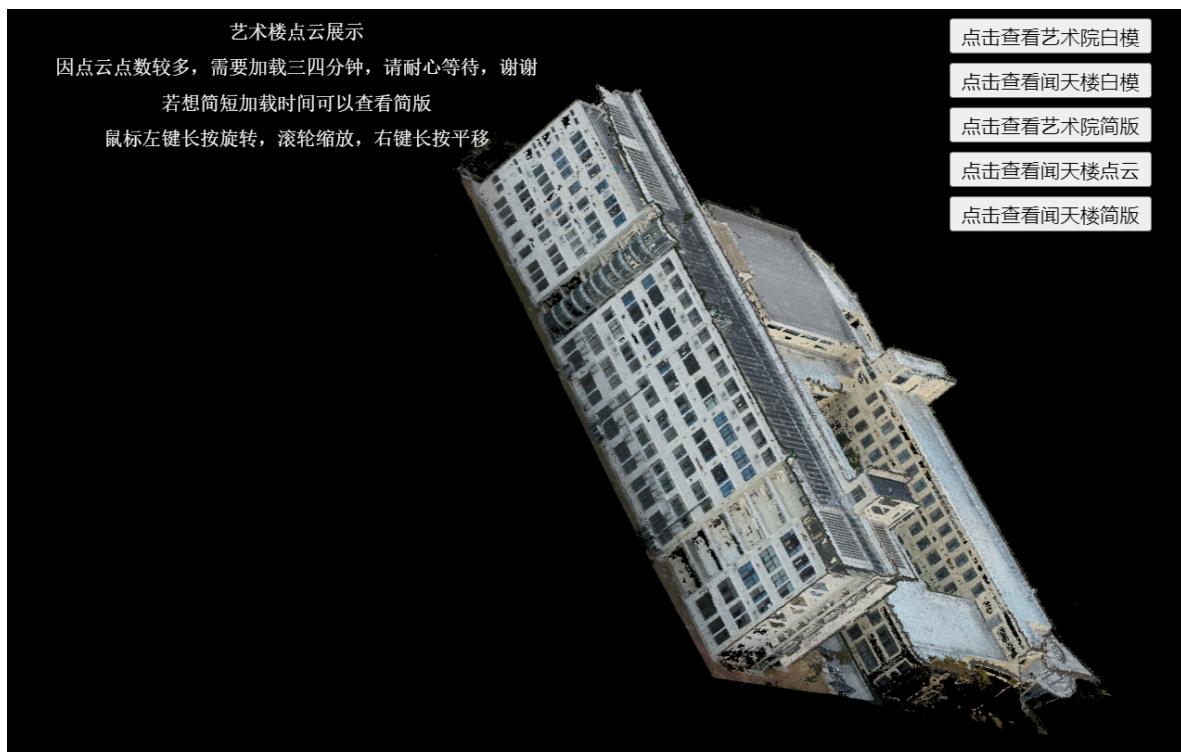
由于点云数据中点较多，我们通过筛选，我们将原点云中900余万个点删减至600余万个点。浏览网站时加载点云数据的总时间大概为三分钟。并且我们将点云数据再次删减，得到由200余万个点组成的简版点云数据，网页数据加载时间进一步缩短为1分钟左右。其加载过程如下：



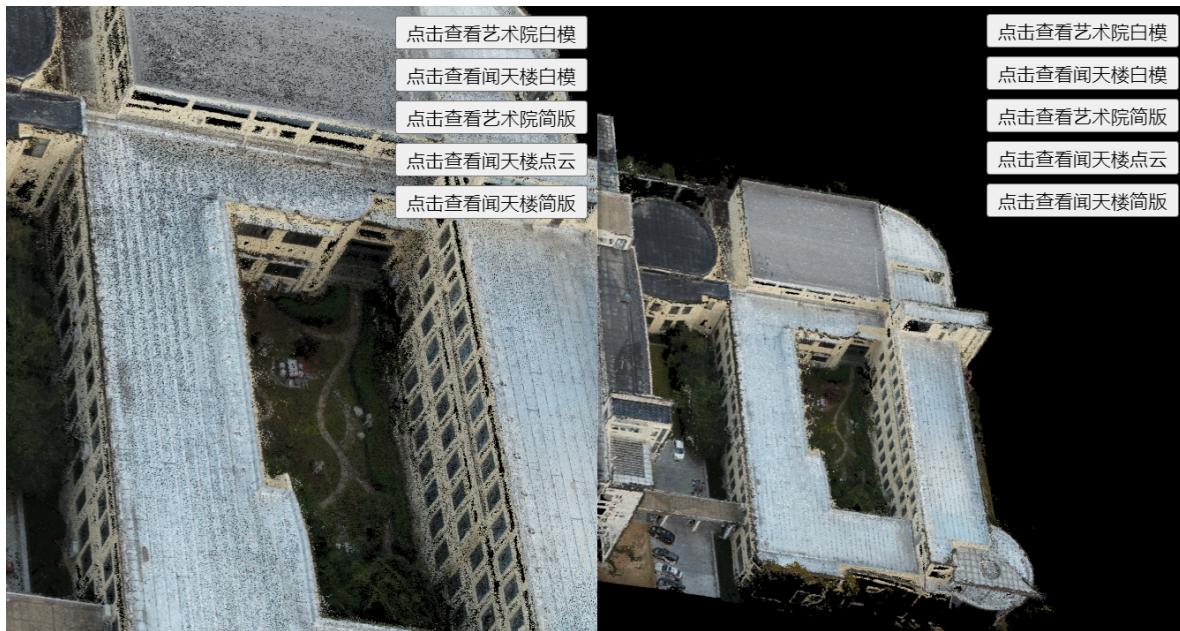
加载结束后，可以看见清晰的点云展示（以艺术学院与科学实验楼为例）：



通过鼠标左键可以实现点云图像的旋转：



并且滚轮可以实现点云图像的放大缩小：

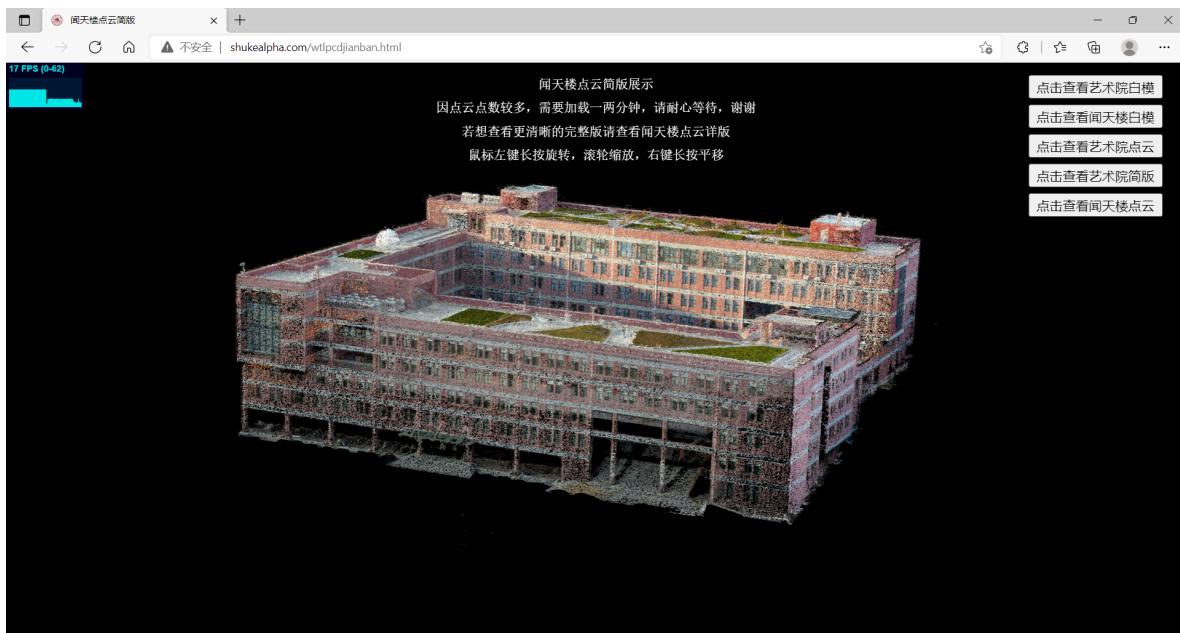


最后，比对一下简版点云数据与原点云数据网页效果，以艺术学院为例（右图为简化效果）：



可见，简化之后，点云模型依然有较好的展示效果。

艺术学院与科学实验楼展示如下（完整网页界面）：



3.3 白模的显示

这小节主要学习 ThreeJs 中的 demo oader/obj2，主要是分析一下 obj 是如何加载的，纹理以及材质是如何加载的，3d camera 以及 camera controller 这些是如何实现的等。

3.3.1 html部分

```
<div id="glFullscreen">
    <!-- 渲染 3D 场景的 canvas -->
    <canvas id="example"></canvas>
</div>
<!-- dat gui 的 div 占位-->
<div id="dat">

</div>
<!--three.js 的其他一些信息说明-->
<div id="info">
    <a href="http://threejs.org" target="_blank" rel="noopener">three.js</a> -
    OBJLoader2 direct loader test
    <div id="feedback"></div>
</div>
```

这一部分最重要的就是这个 `canvas` 标记的添加，也说明了 WebGL 的主要实现就去用这个 `canvas` 去绘制。这和 Android 端上的原生 API 很像。

3.3.2 script及OBJLoader2Example

script导入部分如下：

```
<!-- 导入 threejs 核心库 -->
<script src="../build/three.js"></script>
<!-- 导入 camera controller，用于响应鼠标/手指的拖动，放大，旋转等操作 -->
<script src="js/controls/TrackballControls.js"></script>
<!-- 材质加载 -->
<script src="js/loaders/MTLLoader.js"></script>
<!-- 三方库 dat gui 库的导入-->
<script src="js/libs/dat.gui.min.js"></script>
<!-- 三方库 stats 的导入-->
<script type="text/javascript" src="js/libs/stats.min.js"></script>
<!-- 构建 mesh,texture 等支持 -->
<script src="js/loaders/LoaderSupport.js"></script>
<!-- 加载 obj 的主要实现 -->
<script src="js/loaders/OBJLoader2.js"></script>
```

对于OBJLoader2Example，在JavaScript 中的函数与对象中了解到，JavaScript 中是通过原型 (prototype)来实现面向对象编程。这里先定义了函数 `OBJLoader2Example()`，然后再指定 `OBJLoader2Example` 的 `prototype` 的 `constructor` 为 `OBJLoader2Example()` 本身，这也就定义了一个“类” `OBJLoader2Example`，我们可以使用这个类来声明新的对象。

3.3.3 obj与mtl导入格式

obj数据格式如下：

```

# Blender v2.54 (sub 0) OBJ File: ''
# www.blender.org
# obj对应的材质文件
mtllib female02.mtl
# o 对象名称(object name)
o mesh1.002_mesh1-geometry
# 顶点
v 15.257854 104.640892 8.680023
.....
# 纹理坐标
vt 0.389887 0.679023
.....
# 顶点法线
vn 0.945372 0.300211 0.126926
.....
# group
g mesh1.002_mesh1-geometry_03_-_Default1noculli_03_-_Default1noculli
# 当前图元所用材质
usemtl _03_-_Default1noculli_03_-_Default1noculli
s off
# v1/vt1/vn1 (索引起始于1)
f 1/1/1 2/2/2 3/3/3
.....

```

以及mtl文件数据格式：

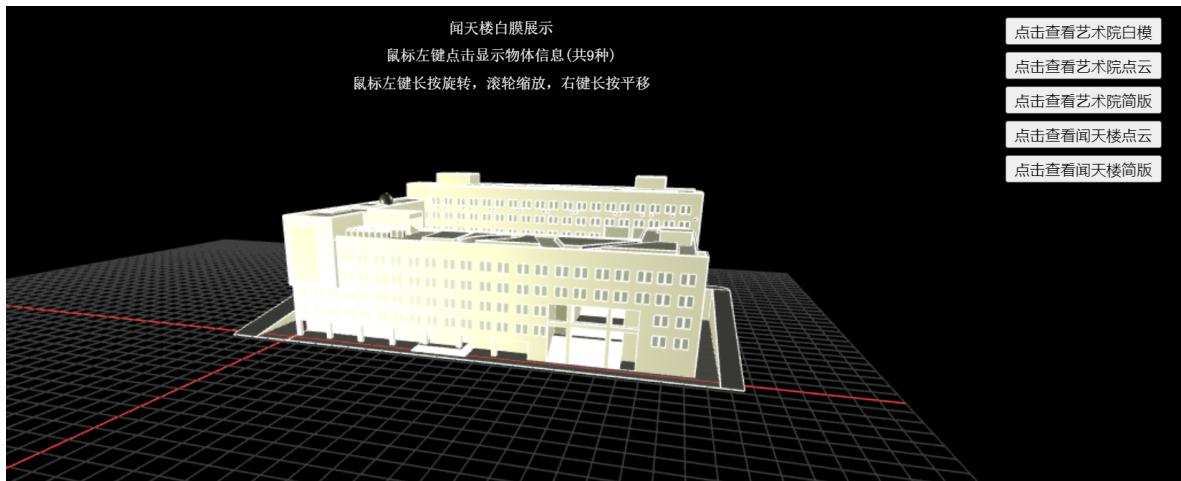
```

.....
# 定义一个名为 _03_-_Default1noculli_03_-_Default1noculli 的材质
newmtl _03_-_Default1noculli_03_-_Default1noculli
# 反射指数 定义了反射高光度。该值越高则高光越密集，一般取值范围在0~1000。
Ns 154.901961
# 材质的环境光 (ambient color)
Ka 0.000000
# 散射光 (diffuse color) 用Kd
Kd 0.640000
# 镜面光 (specular color) 用Ks
Ks 0.165000
# 折射值 可在0.001到10之间进行取值。若取值为1.0，光在通过物体的时候不发生弯曲。玻璃的折射率为1.5。
Ni 1.000000
# 渐隐指数描述 参数factor表示物体融入背景的数量，取值范围为0.0~1.0，取值为1.0表示完全不透明，取值为0.0时表示完全透明。
d 1.000000
# 指定材质的光照模型。illum后面可接0~10范围内的数字参数。各个参数代表不同的光照模型
illum 2
# 为漫反射指定颜色纹理文件
map_Kd 03_-_Default1noculling.JPG
.....
```

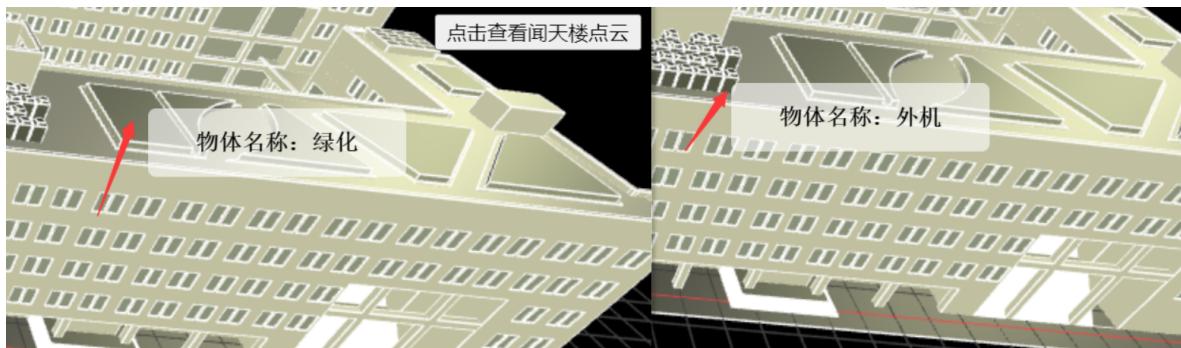
3.3.4 模型加载及结果展示

通过2得到较为完善的白模后，使用threejs加载白模，并通过threejs点击事件显示建筑物相关信息。

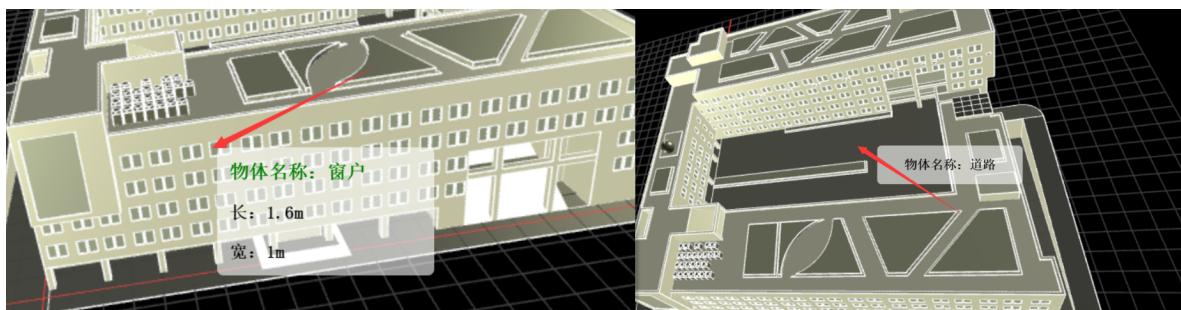
结果展示如下：



鼠标左键点击建筑物某处会显示相关物体信息 (以绿化及顶楼外机为例) :

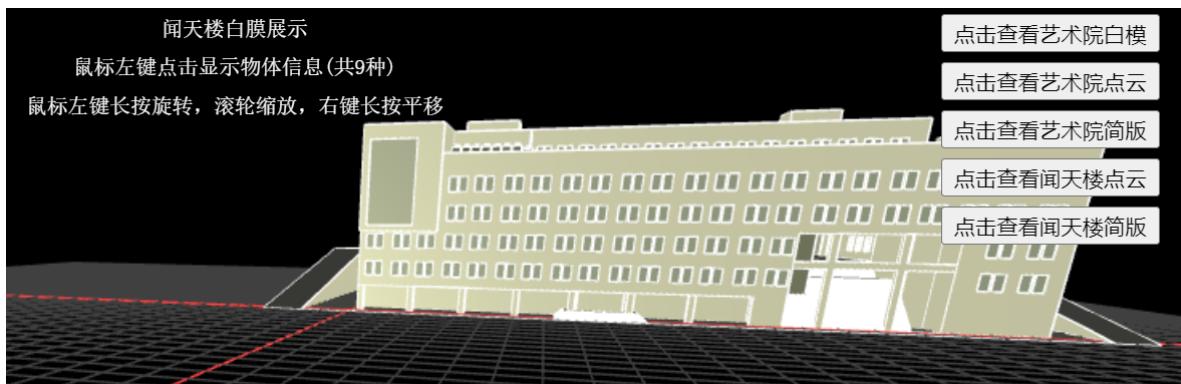


下面再展示下道路和窗户为例:

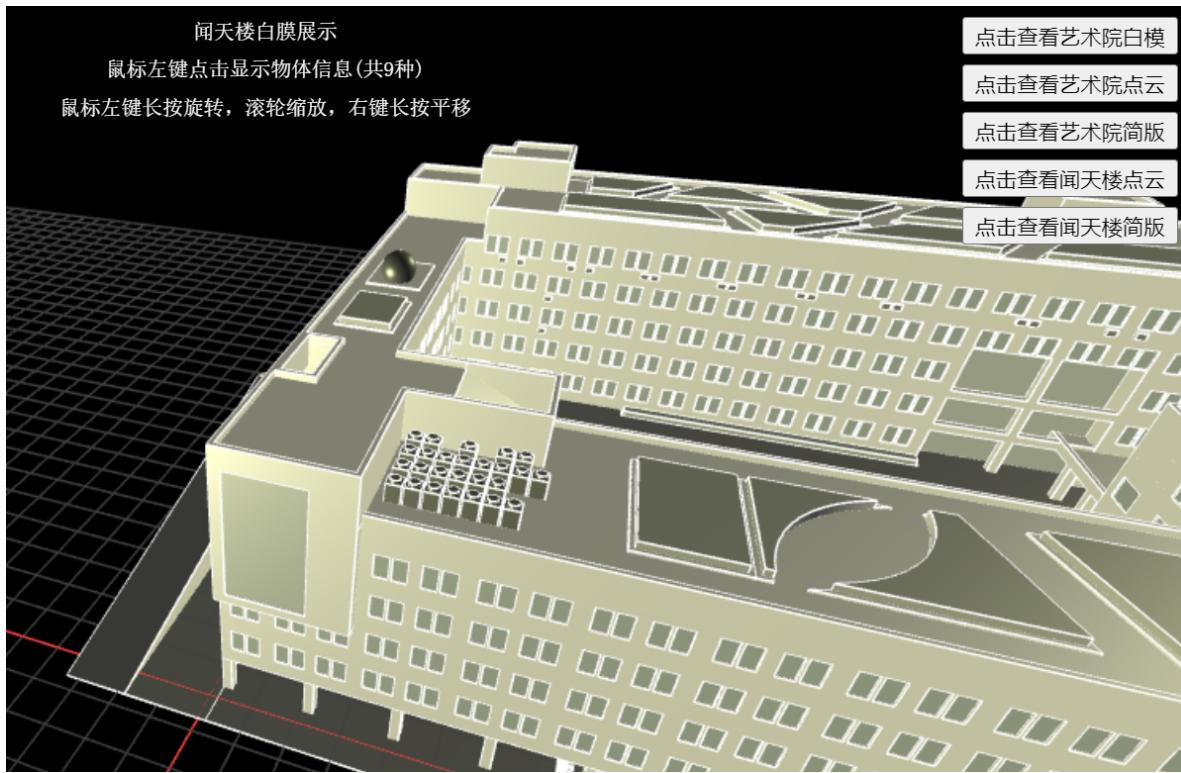


(其中闻天楼共空调外机、绿化、道路、太阳能板、外机、窗户 (不同大小不同识别) 、玻璃幕墙等9种标识)

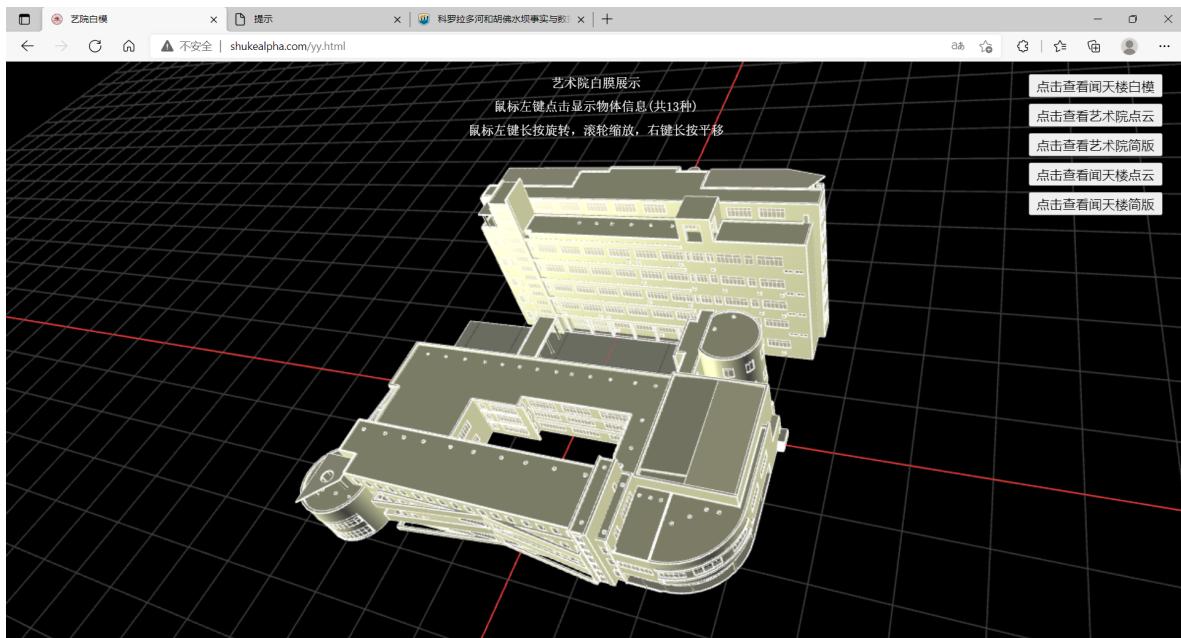
右键长按可实现平移:



滚轮缩放可缩放白模大小:



艺术学院及科学实验楼展示如下 (web端完整显示) :



(其中该白模共窗户、门、玻璃门、屋顶、白墙、管子、屋顶、棚顶、柱子、空调、地板、弧度玻璃等13种标识)

4 参考文献

- 1.PCL点云库<https://pointclouds.org>
- 2.PCL库学习 (1) ----带颜色纹理的点云格式转换 (txt格式转换为pcd格式) https://blog.csdn.net/Di_Wong/article/details/81334210?spm=1001.2014.3001.5502
- 3.PCL—— (3) PCD (点云数据) 文件格式简介<https://www.cnblogs.com/long5683/p/13274820.html>

4.OBJ 模型文件与MTL材质文件<https://blog.csdn.net/newchenxf/article/details/121394626?spm=1001.2014.3001.5502>

5.Three.js官网<https://threejs.org/>

6.Threejs入门<https://blog.csdn.net/u010588262/article/details/79570436?spm=1001.2014.3001.5502>

7.threejs视频教程<http://www.yanhuangxueyuan.com/threejs/docs/index.html>

8.Three.js 基础入门<https://blog.csdn.net/valada/article/details/80871701>

9.theejs的点击事件<https://www.jianshu.com/p/59797960292f>

10.ThreeJs学习笔记——ObjLoader加载以及渲染分析https://blog.csdn.net/weixin_34268610/article/details/88021125