# Indoor3D: A WebGL Based Open Source Framework for 3D Indoor Maps Visualization

Meng Gai*
Peking University

Guoping Wang
Beijing Eng Center of VSV, Peking University

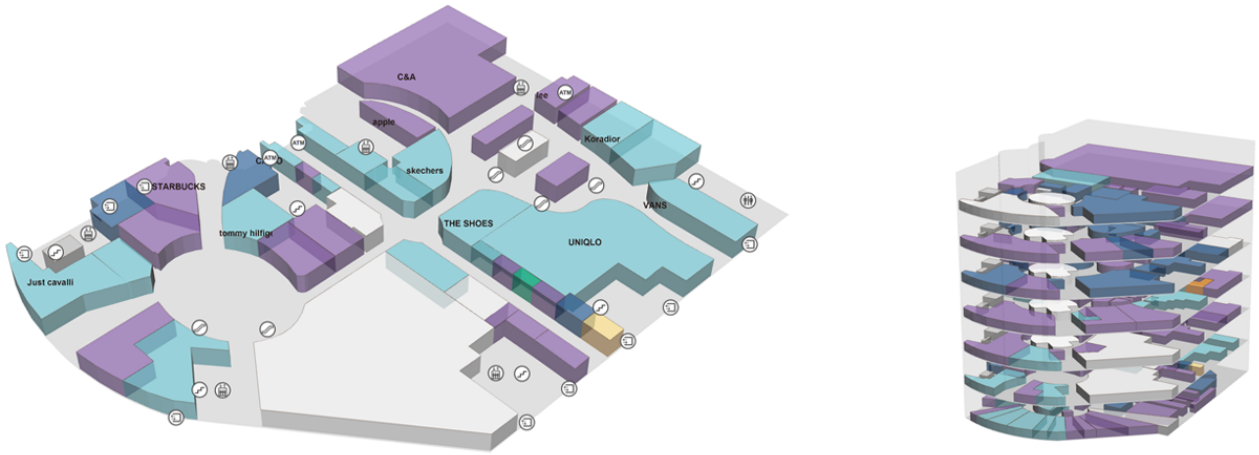



**Figure 1:** *A 3D indoor map visualized by Indoor3D*

## Abstract

In this paper, we present Indoor3D, an open source framework for 3D indoor maps visualization. It takes advantage of the WebGL technique of modern browsers, so it can work on any platform which supports the WebGL feature, including desktop computers and mobile devices. An extensible data structure is designed to describe the indoor scene. The proper default view is generated automatically according to the principal direction. A priority based algorithm is employed to control the visibility of the texts and icons. This library is well designed and easy to use. Developers can create an indoor map and customize its behaviour with only a few codes. Designers also benefit from this framework since they can change its visual style by providing a new theme. We believe our Indoor3D will be useful in many cases, such as airports, subway stations and shopping malls.

**Keywords:** visualization, webgl, web 3d, indoor map

## 1 Introduction

Indoor maps are widely used in airports, subway stations and shopping malls. We can find the indoor maps on boards at the most conspicuous positions in those buildings. And almost all the merchants manuals of shopping malls have several pages for indoor maps to show the retain ditribution. Instead of showing the details such as windows and doors, those maps usually only have a very brief and sketchy apperence, which using polygons and icons to present objects. Besides, most of those indoor maps are displayed in a pseudo-3D style to provide an intuitive impression for the readers and guide them to navigate.

With the development of computer graphics, 3D indoor maps becomes popular. Many organizations have built their own applications to provide more convenient experiences to customers. However there is no universal solution for it by now. Some applications such as Google Map provides their service of 3D indoor maps. But those indoor map modules are highly encapsulated. Developers have limited controls to customize the map. Besides, it is not a cross-platform solution to display uniform maps on both desktop computers and mobile devices.

The Web has come a new way to build cross-platform multimedia applications, especially as the HTML5 becoming more and more popular. Although there are many frameworks for web based visualization, most of them are for general usage. No framework focusing on 3D Indoor maps visualization has been proposed.

To address these concerns, we contribute Indoor3D, an open source framework for 3D indoor maps visualization. It provides a new way to build up an application integrated with 3D indoor maps. We make use of WebGL to make it usable on any platform with modern browsers. Meanwhile, to save and exchange the map data, we have designed a extensible file format based on the widely used JSON format.

## 2 Related Work

As far as we know, although there hasn't been any open source indoor map visualization framework proposed, some previous work provides fundamental techniques or has close relationship with our Indoor3D library.

*e-mail:gaimeng@pku.edu.cn


Web3D '15, June 18 - 21, 2015, HERAKLION, Greece
 ISBN 978-1-4503-3647-5/15/06…$15.00
DOI: http://dx.doi.org/10.1145/2775292.2775310

### 2.1 Graphics on the Web

There are many ways to show graphics objects on the web page. For more details, we refer the readers to a survey[Evans et al. 2014].

Scalable Vector Graphics(SVG) is popular since the graphics elements have their correspond HTML tags. Many popular visualization library[Bostock et al. 2011] is built based on SVG technique. But SVG can only handle 2D graphics. Similarly, HTML5 canvas only support 2D drawing as well, but it renders graphic elements in a imperactive way. Although it is possible to create a 3D rendering algorithm based on SVG or HTML5 canvas by handling the projection problems by developers own, the rendering efficiency remains a big problem.

There are some ways of showing 3D objects requiring plugins installed onto the browser. They are not convenient enough for the normal users. For example, Adobe Flash is one of the most popular ones.

WebGL enables the developer to access to graphics hardware by JavaScript. One of the biggest contribution is that it brings huge performance increase, allowing fast manipulation of thousands of vertices in 3D space.

ThreeJS[Cabello 2010] is one of the most famous libraries for 3D graphics based on WebGL. It provides high level APIs and well encapsulated functions. Moreover, it provides several different rendering engines(WebGL, Canvas and SVG), which makes it easier to deal with compatibility problems. For these reasons, we build our framework on top of the ThreeJS library.

### 2.2 Reconstruction from 2D Floor Plans

The 2D floor plan is a standard way to express the structures of a building in the architecture field. Several methods are developed to reconstruct 3D buildings from 2D floor plans. [Lewis and Séquin 1998] developed a semi-automatic system to create 3D building models from CAD plans. [Zhu et al. 2014] proposed an automatic algorithm to generate 3D buildings from 2D vector floor plans by recognizing the specific elements in the them.

Similarly, we also construct the 3D building models by extruding a 2D map. But we have different purposes. The results from those reconstruction methods are full of details, such as windows and doors. They are good for CAD (computer aided design), virtual reality and simulation purpose, but not good enough for the general public users.

We intend to visualize the indoor map with as less redundancy details as possible. All the rooms are expressed by simple polygons while all the windows and doors are omitted. Some public facility spots are simplified by a single point. As a result, it reduces the difficulty of the users' cognitive. More over, our file format benifits from the simplicity of the models. The file size can be smaller and better used in network transmission.

### 2.3 Indoor Maps Visualization

Although indoor maps are commonlly used in public buildings, bringing indoor maps into digital applications is a emerging direction.

Some commercial applications such as Google Map are integrated with the indoor map functions. But they are not friendly to developers to customize the maps. Although some of them provide a bunch of APIs, they are not easy to be used in cross-platform applications. Widitu[wid 2015] is one of the most professional applications which provide indoor maps services in China. It releases SDK on different platforms for developers. But it renders the maps only in 2D.

## 3 Architecture Design

Indoor3D is built beyond the ThreeJS library and designed under the rule of MVC(Model-View-Controller) pattern, which has been widely used to separate internal representations. After loading the map data represented by a json file, a Building object is created to hold the data and plays the role of *Model*. All the interactions of mouse and touch events are handled in a *Controller* class. The Indoor3D class contains the *Renderer* of the ThreeJS and cooperates with the model and controller.

```
{
  "Building": {
    "Shape": [
      [-202, -768, -208, -768, ...,  -202, -768]
    ],
    "Name": "Sample Building",
    "Address": "Sample Street Sample City",
    "Longitude": 116.436347485204,
    "Latitude": 39.9718578960126,
    ...// More building properties
  },
  "Floors": [
    {
      "Shape": [
        [566, -232, 566, -235, ..., 566, -232]
      ],
      "Name": "F1",
      "Area": 13985,
      "Height": 5,
      ...// More floor properties

      "Rooms": [
        {
          "Shape": [
            [-85, -545, -54, -747, ..., -85, -545]
          ],
          "Name": "Sample Shop",
          "Type": 102,
          "Center": [-137, -624],
          "Area": 261.94140625,
          ... // More room properties
        },
        ... //More Rooms
      ],

      "Points": [
        {
          "Center": [-399, -55],
          "Name": "Entrance",
          ... // More point properties
        },
        ... // More Points
      ]
    }
    ...//More floors
  ]
}
```

a: "Building" block (Shape, Name, Address, Longitude, Latitude, ...// More building properties)
b: "Floors" array
c: Floor properties block (Shape, Name, Area, Height, ...// More floor properties)
d: "Rooms" array
e: "Points" array
f: Room properties block (Shape, Name, Type, Center, Area, ... // More room properties)
g: Point properties block (Center, Name, ... // More point properties)

**Figure 2:** *The sample data. (a) The Building propterties block. (b) The Floors array. (c) The Floor propterties block. (d) The Rooms array. (e) The Points array. (f) The Room propterties block. (g) The Point propterties block*

### 3.1 Map Data Structure

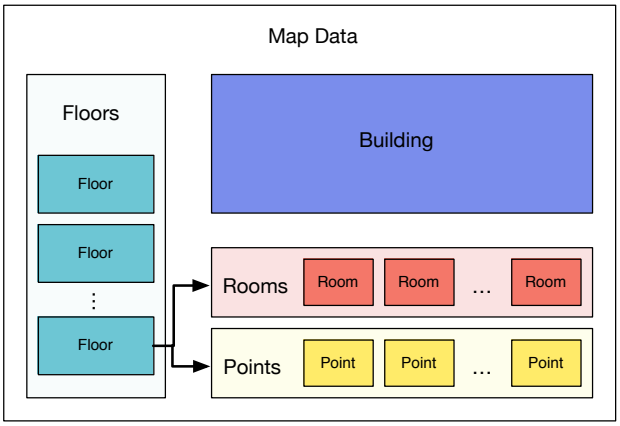



**Figure 3:** *The data structure of the JSON file.*

As far as we know, there has not been a standard file format for the indoor mpas. We designed a structure in JSON format to describe the information of an indoor map, which is widely used in Web development and data exchange. Although XML is another popular format with high readability, JSON format is more compact. The data structure is extensible, and designed independent from implementation, so that it can also be used or converted for other indoor maps applications easily, such as those displaying 2D indoor maps.

We design our file format according to the real structure of a building straightforwardly. A complete indoor map is composed of four parts: **Building**, **Floors**, **Rooms** and **Points**. All the geometry information are stored in 2D, and it will be extruded to 3D in the front end. **Building**, *Floors* and *Rooms* all contain a series of polygons (only one in most cases) to present their outlines, while *Points* stand for the facility spots whose positions are represented as single points. All of them have their own unique id to identify from each other. Many interfaces for the developers are using id as the parameters in Indoor3D.

The *Building* holds the information of the whole building itself, such as the building's name, its address, the latitude and the longitude. It also has a shape property which is a series of arrays. Each array is some numbers to represent a polygon. Each pair of the numbers represent a coordinate point. In most cases, only one array is used unless there are multi-polygons and holes.

The *Floors* structure is an array holding all the floors. Each element of the array describes a floor. The floor contains an array of *Rooms* and *Points*. The Floor's properties are also extensible to hold the floor propertie (the floor's name, area, height, etc.). The *Room* is a functional area in a floor, such as a shop in a mall or a closed area. The *Type* property of the *Room* is used for colourization. This is especially useful when using different colors to represent different functional areas. The *Floor* and *Room*'s geometry shapes are represented in the same way as the outline of the *Building*. And their center and area is pre-computed for later use. The *Point* is used to represent the facility spots, such as *Entrance*, *Staircases*, *Toilet* and so on. They will be shown as an round icon later.

A sample data of a map file is shown in Figure 2. And the diagram to sketch the hierarchical data structure is shown in Figure 3. Notice the information of the *Building* and the *Floors* are separated. Because in some cases, the building's information is not necessary. Moreover, some applications may take advantage of this file structure to support streaming transmission and display some of the floors as soon as they are loaded.

The original JSON file without trimming spaces is usually less than 1 MB. The file of a map with seven floors and more than 300 rooms in Figure 1 is less than 300KB. This file size is totally acceptable in web applications. Further compression techniques can make it even smaller, but need more time cost for uncompressing in the front browsers.

There are many ways to get the required map data. Since the data structure is clean and simple, the map data from other application can be converted to ours easily. Furthermore, we are developing an utility to let the users create the map interactively. But it still remains a tedious work for the users to create the map manually. So similar to the problem of building reconstruction from 2D floor plans, automatic recognition and converting approach need to be studied.

### 3.2 Architecture

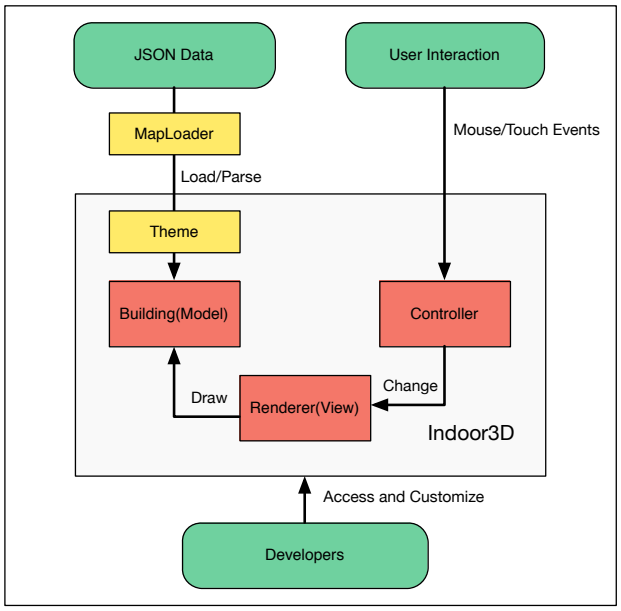



**Figure 4:** *The architecture of Indoor3D based on MVC pattern.*

As introduced before, the architecture of our framework is designed based on the MVC pattern. Several utility functions are defined to loose coupling and increase flexibility. The architecture is shown in Figure 4.

A *MapLoader* class is defined to request the JSON file and parse it to a *Building* object. The texts in the map is translated to 2D sprites by an extra HTML Canvas. The 3D Mesh models are all stylized according to the *Theme*. The 2D sprites of texts and icons are create in a lazy way to save memory. So if they are hidden, they won't be created. The properties of the *Theme* are easy to understand. They describe the color, opacity, stroke style, font, icons, etc. Even the designers without programming experience can modify it intuitively.

The *Building* class serves as a container of the 3D models and 2D sprites. When the user switch the floors, the *Building* is in charge of removing old objects from the scene temporarily and add the new objects in the current floor to the scene. If the 2D sprites in a floor have not been created yet, they will be create at the time when the

floor is switching to visible. Once they are created, they will be cashed in case of later use.

The *Renderer* is naturally the ones provided by ThreeJS. There are three kinds of render engines available: WebGL, HTML5 Canvas and SVG. The WebGLRenderer is first considered. And the other ones are alternative when WebGL is not supported by the users' browser.

The *Controller* class receives all the interaction events from the user and convert it to the transformation operations(pan, pivot and zoom) by manipulating the camera in Indoor3D. On desktop computers, the click event in the browser is dealed with. While on mobile devices, it handles the touch events. Although using WebGL can achieve high frame frequency, projecting the 2D sprites to the right position is a little time consuming. So we let the controller notice the *Renderer* to update the canvas only when the view is changed. This saves the computation resources significantly when the view is static, and is particularly useful when using the render engines other than WebGl Renderer.

The Indoor3D class serves as an whole wrapper of the model, the renderer and the controller and interacts with external calls. Besides the cameras are managed by it. We create two cameras. One is for rendering perspective 3D models, the other one is for the 2D sprites. This Indoor3D class provides the interfaces for the developers to access the data and specify the behaviour of the map. For example, the developers can set if the rooms in the current floor is selectable, if the 2D sprites should be shown, etc. Besides, this class has a function to generate UI for the users to switch between floors. The UI is a HTML element appended to the webpage, its style can be specified by the classical CSS.

### 3.3 Usage

The Indoor3D is really easy for the developers to use. In general, the only class that the developers need to interact is the *Indoor3D* class.

Two examples are shown in Figure 5, one is the simplest way and the other one is a little more complicated. In the simplest way, it creates an indoor map from a json file just in two rows of codes, and it will create an indoor map by default configuration. In the more complicated example, it shows some additional settings customized in the callback function when the map data is loaded successfully: The interactive selection is enabled. The room names are hidden. A callback function is added and it will be called when the selection event is triggered. At last, the UI of switching floors is added to the webpage.

Further more, the developer can specify an HTML ¡div¿ tag as the container of the indoor map by passing the parameter to Indoor3D. If the container is not provided, Indoor3D will generate a full screen canvas for map rendering by default.

## 4 Implementation

In this section, we will introduce some details about the algorithm in implementation. First we extrude the 3D models directly from the 2D coordinates stored in the map data. After that, we compute the principle axis of the map to provide the best view direction. Besides, we use an priority based algorithm to control the visibility of the 2D sprites to avoid occlusions.

### 4.1 3D Extrusion

The indoor map in the JSON file is described as 2D coordinates. To generate the 3D mesh model, we need extrude the polygon outlines

a

```
<script>
     var indoorMap = Indoor3D();
     indoorMap.load("sampledata.json");
</script>
```

b

```
<div id="indoor3d"></div>
<script>

    var params = {
       mapDiv:"indoor3d"
    }
    var indoorMap = Indoor3D(params);

    indoorMap.load("sampledata.json", function(){
        indoorMap.setSelectable(true);
        indoorMap.showRoomNames(false);
        indoorMap.setSelectionListener(callback);

        var ul = Indoor3D.getUI();
        document.body.appendChild(ul);
    });
</script>
```

**Figure 5:** *The sample data. (a) The simplest usage example. (b) A slightly more complicated example with some detailed settings.*

by the height of the floor. If the floor height data is missing, we set it to 5 meters by default.

The extrusion result depends greatly on the triangulation of the polygon. For complex polygons with holes, the well developed poly2tri[Contributors 2009-2014] library based on sweep-line algorithm [Domiter and Žalik 2008] is employed to fulfil this task.

When switched to the global view of the whole building to showing all the floors(as shown at right in Figure 1), the offsets between the floors are not the real distance to give a better visual experience for the users. The value of the offset is a multiple scalar operated on the floor's height. In our implementation, the multiple value is set to 4 and can be changed by developers.

### 4.2 Best View Generation

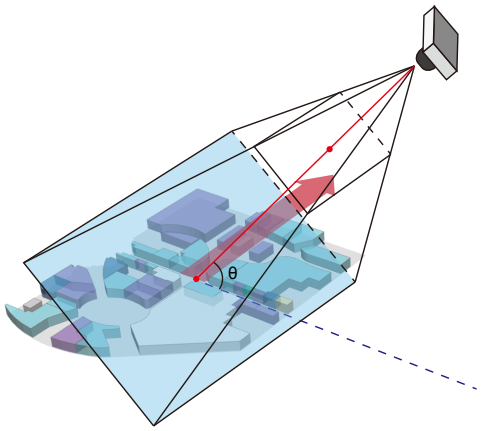



**Figure 6:** *The default view generation*

Displaying the indoor map in a proper default view is important for user experience, especially on those small screens of mobile devices.

Selecting the best views for 3D object has been well studied, especially in the CAD field. Previous work either maximized the visibility of interesting content using viewpoint entropy [Vázquez et al. 2003] or meaningful salient parts numbers[Mortara and Spagnuolo 2009]. [Fu et al. 2008] proposed a method to find the upright orientation of a model by several geometric atrributes, such as stability, symmetry, parallel and visibility. [Hu et al. 2011] applied a convex hull based method to find the upright orientation and chose the best view which owns the most visual features in the depth image from four candidate isometric views.

To implement the view selection method, we should balance the computation efficiency simultaneously. Since in our application, there is more prior knowledge about the model, This view selection problem can be treated much easier. Apparently, the supporting plane is ready-made, since the floors are always parallel to the horizontal plane. The solution space of the view point position is limited in the half upper space.

We first take all the polygons vertices of the floor's outline and use PCA (Principal components analysis) to find the first principal direction. This first principal direction can be treated as the width direction of its OBB (Oriented Bounding Box) and will be set parallel to the screen width direction later. The angle between the first principal direction and the original horizontal direction is stored in the JSON file so that it reduces computation in the front end.

To fit the 3D indoor map right in the view port, the proper distance between the camera and the center of the scene can be computed:

$$d = \frac{H}{2} cot(\frac{\theta_{fovy}}{2})$$

where $H$ is the height of the floor's OBB.

The last step is to decide at which side of the principal direction the camera should be positioned. Starting from the idea of helping the users to exhibit more informations, we choose the side with more rooms. This is computed by comparing the center of the floor's OBB and the average center point of all the rooms in this floor. The camera is positioned at the same side as the latter point.

The view is shown in Figure 6. The red arrow shows the first principal direction. The tilt angle $\theta$ is set to 75 degree in our system after researching on several existing pseudo-3D maps of some buildings, so that the users can see all the rooms and get good experience of 3D perspective.

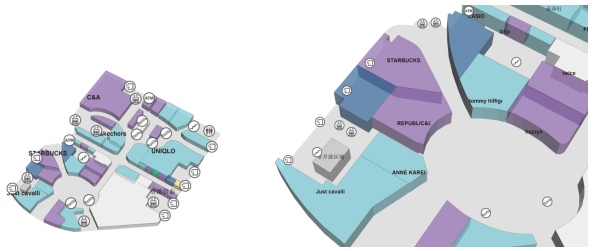
### 4.3 Progressive Visibility of Texts and Icons

Showing all the texts and icons at the same time will be a disaster. The overlap of these sprites will bring bad visual effects. To show the texts and icons progressively, we employed an priority based algorithm. When the indoor map is zooming out, the elements with high priority are more likely to be kept, others will become invisible.

For the text names, the default priority is set to the area of the room, which can be directly computed by the polygon outline based on Green's Theorem in a plane[Russ 2010]. Given points $(x_i, y_i)$, i = 0, ..., n, with $x_0 = x_n$ and $y_0 = y_n$, the area $S$ of a polygon in a plane can be rapidly calculated as:

$$S = \frac{1}{2} \sum_{i=0}^{n-1} a_i$$

where




(a) (b)

**Figure 7:** *Progressive visibility of the texts and icons. More detailed elements are shown when zooming in*

$$a_i = x_i y_{i+1} - x_{i+1} y_i$$

This strategy means the names of bigger rooms tend to be kept when zooming out. More advanced priorities can be designed if there is more information. For example, if the popularity index of every shop is available, the priority may take the popularity into account and defined as following:

$$P_i = w_1 \frac{S_i}{S_{max}} + w_2 \frac{Pop_i}{Pop_{max}}$$

where $P_i$, $S_i$ and $Pop_i$ are the priority, area and popularity index of the $ith$ shop respectively. $S_{max}$ and $Pop_{max}$ are the maximum shop area and popularity index of current floor. $w_1$ and $w_2$ are the weights to balance these two items.

For the icons of facility spots, the default priority descending order is set as Table 1. The priority is designed based on the idea that the icons showing navigation information are more important, and the entrances are the most important. This is coincident with the psychological expectation of the customers.

**Table 1:** *The priority of facility spots*

| point type | Entrance | Staircase | Escalator | elevator | Toilet | others |
|---|---|---|---|---|---|---|
| priority | 3 | 2 | 2 | 2 | 1 | 0 |

Notice that if the element is set to visible as soon as it turns not overlapping with other ones, there will be an undesired flicking effect. Because the elements may switch between visible and invisible frequently when the user is pivoting. So we add a margin area to the element's bounding rectangle when it is about becoming visible from invisible. Only when the gap between the bounding rectangles is beyond a threshold, it does turn to visible again. In our experiment, the margin value is set to 5 pixels to achieve good visual effects.

The algorithm is shown in Algorithm1.

### 4.4 User Interaction

Once the map is loaded, it's shown by the best view direction introduced in previous section. To display the 3D indoor map interactively by users, we use an orbit controller to control the movement of the camera. On a computer with the mouse, users can drag the left mouse button to pan the scene, right mouse button to pivot it and scroll the middle button to zoom in and out. While on a mobile device, single touch and multi-touch gestures are used to control the scene.

**Algorithm 1** decide the visibility of elements(texts and icons)

```
sort the element by descending priority order
for i = 1 to elements.length do
    visibility ← true
    margin ← 5
    for j = 0; j < i; j++ do
        recti ← elements(i).boundingRect
        rectj ← elements(j).boundingRect
        if elements(j).visible and recti.collide(rectj) then
            visibility ← false
            break
        end if
        recti.shrink(margin)
        rectj.shrink(margin)
        if elements(i).visible ≠ true and recti.collide(rectj)
        then
            visibility ← false
            break
        end if
    end for
    elements(i).visible ← visibility
end for
```

For interactive selection, we use the classical ray casting method to check which object is selected by users. When showing all the floors, the floors are treated as an integral whole object, while when showing a floor, the rooms are the objects to be checked. The default behaviour of the selected room is high lighted to light yellow.

Developers can set a selection listener to Indoor3D to customize the behaviour of the map when something is selected. There are 3 parameters passed to the callback listener: the id of the selected object, the projected 3D position of the click point, and the selected object. This is useful when the developer wants to create a detailed pop-up dialog or put a marker in the map when something is selected. Since the 3D object can be accessed by API, the developers can even create smooth animations with TweenJS [Gskinner 2015]. Some examples of customized interactions are shown in Figure 8.

### 4.5 Downward Compatibility

Based on the idea of progressive enhancement, we have many strategies for those browsers that do not support WebGL.

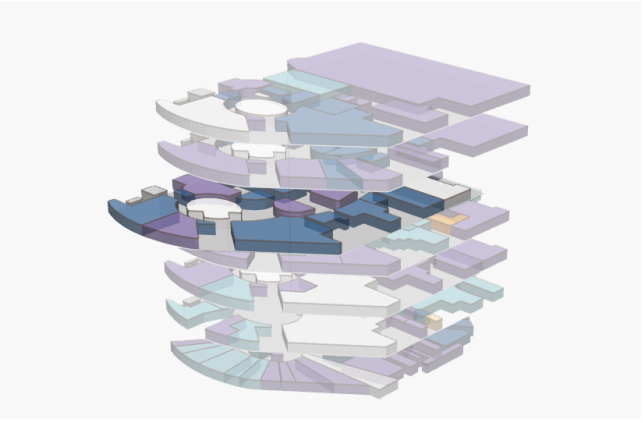
On a desktop computer with older browsers, Indoor3D use the CanvasRenderer provided by ThreeJS. But for some of the mobile devices, the computing resources are limited, so the CanvasRenderer is not efficient enough. We also provide a 2D version indoor map rendered by HTML5 canvas. The translation and rotation act directly on the canvas DOM element rather than redraw the canvas. In such a way, the 2D map can run smoothly on some older mobile devices.

For those older browsers which do not support HTML5 Canvas at all, the developers should consider to display a static image.

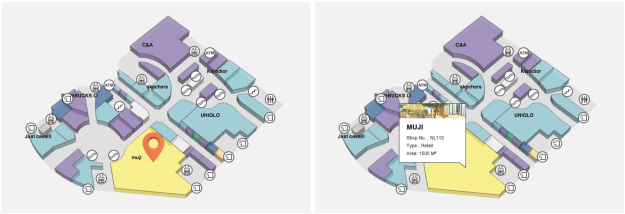
## 5 Result and Discussion

### 5.1 Results

We have tested our framework on several platforms, it can run on an real-time frame on every platform which supports WebGL. Because we redraw the scene only when the view is changed, it runs on a full frame rate when the view is static. When the user is interacting, the frame rate will drop down a little bit. The interacting frame rates



(a)



(b) (c)

**Figure 8:** *different customized interactions.(a) Floors selection (b) Add a marker on selected shop. (c) Add a pop-up dialog*

of showing the map in Figure 1 on 3 of the platforms are shown in Table 2.

**Table 2:** *The Frame rates of interacting on different platforms*

| Device | OS | CPU | GPU | Browser | *FPS* |
|---|---|---|---|---|---|
| Desktop PC | Windows 8 | i5 2.8GHz | NVIDIA GeForce GTX 550 Ti | Chrome 41 | 50 |
| Macbook Air | Mac OS X | i5 1.3 GHz | Intel HD Graphics 5000 | Chrome 41 | 60 |
| Macbook Air | Mac OS X | i5 1.3 GHz | Intel HD Graphics 5000 | Safari 8 | 40 |
| The new iPad | IOS 8 | A5X | A5X | Safari | 25 |

Figure 9 are some different rendering styles customized by changing the themes, which shows the flexibility and extensibility of our framework. The color, stroke and transparency are different between each other.

### 5.2 Future Work

This is the first step of our work, more future work is required for a complete indoor map framework. First, navigation function is usually necessary for a map application. When the users set the start point and end point in a map, the closest path will be generated. The shortest path is usually computed by A*[Hart et al. 1968] based searching algorithms. This function is especially useful when the indoor map is registered to the real building. Besides, more customization functions will bring more convenience to developers, such as adding different image layers onto the map. Moreover, acquiring the map data is an tough but important task. Creating an whole indoor map data manually remains tedious for the users. Developing the algorithm which can convert existing vector plans or

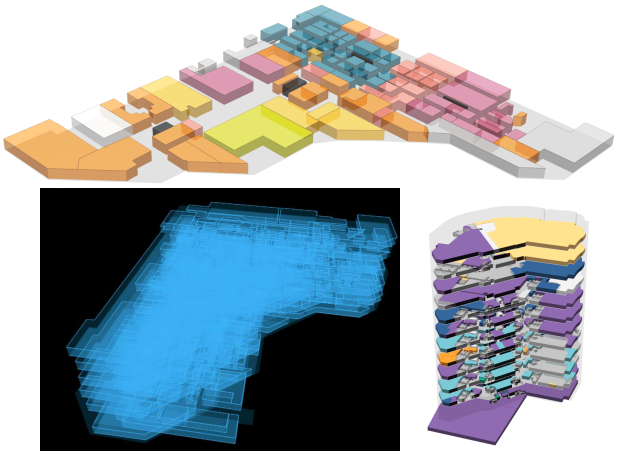


**Figure 9:** *different rendering styles*

raster images to our indoor maps data automatically will help a lot.

## 6 Conclusion

In this paper we presented our Indoor3D framework. It takes advantage of WebGL to render 3D indoor maps for cross-platform applications. We designed a JSON file format to store the map structure, and solves several problems sush as best view selection and progressive element visibility to achieve better user experience. The framework provides friendly interaction for users. And it is designed flexible to allow the developers and designers to customize it conveniently.

We believe many organizations will benefit from our Indoor3D framework. Airports, subway stations and shopping malls will provide better interactive services for guests and customers with our Indoor3D framework.

The source code and documentation are available from the website[Gai 2015].

## Acknowledgements

This research was supported by Grant No. 61421062, 61170205, 61232014, 61472010 from National Natural Science Foundation of China. Also was supported by Grant No. 2012AA011503 from The National Key Technology Research and Development Program of China.

## References

BOSTOCK, M., OGIEVETSKY, V., AND HEER, J. 2011. $D^3$ data-driven documents. *Visualization and Computer Graphics, IEEE Transactions on 17*, 12, 2301–2309.

CABELLO, R. 2010. Three. js. *URL: https://github. com/mrdoob/three. js*.

CONTRIBUTORS, P., 2009-2014. poly2tri.js. `http://code.google.com/p/poly2tri/`.

DOMITER, V., AND ŽALIK, B. 2008. Sweep-line algorithm for constrained delaunay triangulation. *International Journal of Geographical Information Science 22*, 4, 449–462.

EVANS, A., ROMEO, M., BAHREHMAND, A., AGENJO, J., AND BLAT, J. 2014. 3d graphics on the web: A survey. *Computers & Graphics 41*, 43–61.

FU, H., COHEN-OR, D., DROR, G., AND SHEFFER, A. 2008. Upright orientation of man-made objects. In *ACM transactions on graphics (TOG)*, vol. 27, ACM, 42.

GAI, M., 2015. Indoor3d. `https://github.com/wolfwind521/indoor3D`.

GSKINNER, 2015. Tweenjs. `http://www.createjs.com/TweenJS`.

HART, P. E., NILSSON, N. J., AND RAPHAEL, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on 4*, 2, 100–107.

HU, K., WANG, B., YUAN, B., AND YONG, J. 2011. Automatic generation of canonical views for cad models. In *Computer-Aided Design and Computer Graphics (CAD/Graphics), 2011 12th International Conference on*, IEEE, 17–24.

LEWIS, R., AND SÉQUIN, C. 1998. Generation of 3d building models from 2d architectural plans. *Computer-Aided Design 30*, 10, 765–779.

MORTARA, M., AND SPAGNUOLO, M. 2009. Semantics-driven best view of 3d shapes. *Computers & Graphics 33*, 3, 280–290.

RUSS, J. C. 2010. *The image processing handbook*. CRC press.

VÁZQUEZ, P.-P., FEIXAS, M., SBERT, M., AND HEIDRICH, W. 2003. Automatic view selection using viewpoint entropy and its application to image-based modelling. In *Computer Graphics Forum*, vol. 22, Wiley Online Library, 689–700.

2015. Widitu.com. `http://www.widitu.com`.

ZHU, J., ZHANG, H., AND WEN, Y. 2014. A new reconstruction method for 3d buildings from 2d vector floor plan. *Computer-Aided Design and Applications 11*, 6, 704–714.