3D City Reconstruction From Google Street View

Marco Cavallo
University Of Illinois At Chicago
Chicago, IL
mcaval4@uic.edu

ABSTRACT
Despite laser scan 3D point cloud acquisition has greatly improved over the next few years, the process of creating 3D large scale city models is still quite expensive and not straightforward. At the same time, nowadays services such as Google Street View provide a vast amount of geo-registered panoramic imagery, guaranteeing a decent resolution for dense locations at zero cost. Our idea is indeed to leverage this free information provided by Google Street View in order to obtain a cheap and automatizable 3D recontruction of an urban area, by extracting the depth information related to the great number of panoramic images available online.

ACM Classification Keywords
H.5.m. Graphical user interfaces; Computer Graphics; Visualization

Author Keywords
Google Street View; Point Clouds; 3D reconstruction

INTRODUCTION
Google Street View can be considered as an online browsable dataset consisting of billions of street-level panoramic images acquired all around the world. Since 2007, the company has continuously improved and updated its global image database, now collecting on average one panorama each 5-10m in urban environments [2]. Google provides public APIs for requesting virtual camera views of a given panorama from its GPS position or unique identifier. These views are rectilinear projection of the spherical panorama with a user selected field of view, orientation and elevation angle. These views can be considered as undistorted images taken from pinhole cameras [2], free of distortion, and need to be partially overlapped in order to reconstruct the 360° panorama.
Additionally, the speed with which Google's laser scanners get reflected by surfaces allows the company to calculate the distance from the camera to objects or buildings, allowing the creation of 3D models to be used in other applications such as Google Earth [1]. Google API indeed provides a way to obtain the depth map corresponding to a specific panorama, information that we will exploit in order to reconstruct the 3D scene. Once decoded and visualized in space, this depth data and the models generated from it may turn out to be useful for many different applications like navigation and augmented reality, but also virtual reality oriented videogames.

RELATED WORK
Numerous previous works tried to exploit the possibilities opened by Google Street View imagery, but not so many tried to make a significant use of the corresponding depth information. Some of the works which tried to leverage depth maps are oriented to the world of Robotics, where panoramic images are used for camera pose estimation. The sample work by *Agarwal et al.* [2] proposes a complementary method for robot geolocalization based on matching the features of a geotagged panoramic image with the input of a monocular camera, so that the robot can orientate in an environment without the usual need of pre-visiting a location. Some other approaches aimed instead at trying to use depth information to add virtual content to a single panorama, "augmenting" the traditional spherical representation provided by Google Street View. This could be meant for narrative or didactic purposes in order to increase interactiveness in geolocated environments. Among them we can cite projects like *ARLearn* [7] or the work proposed by *K. Hara* [6], aimed at identifying street-level accessibility problems; another interesting example tending to an even more interactive approach is a drive simulator [6] completely built with panoramic images and a real car. However, all of these works generally dealt with only one panorama at a time, exploiting depth information only for placing objects inside the spherical projection of the panorama. *Mikusic and Kosecka* [1] provide instead a method for combining more panoramic images with the goal of building a 3D textured representation of an urban environment. Our work apparently goes in their same direction, even though less oriented at photorealism and with a future idea of combining 3D reconstruction with augmentations, thus combining the previous approaches. In addition to that, we provide a one-to-one mapping from the real world coordinates to the virtual world coordinates, which could be an advantage for positioning virtual objects, eliminating the need of geometric computations when dealing with the usual spherical panoramic projection.

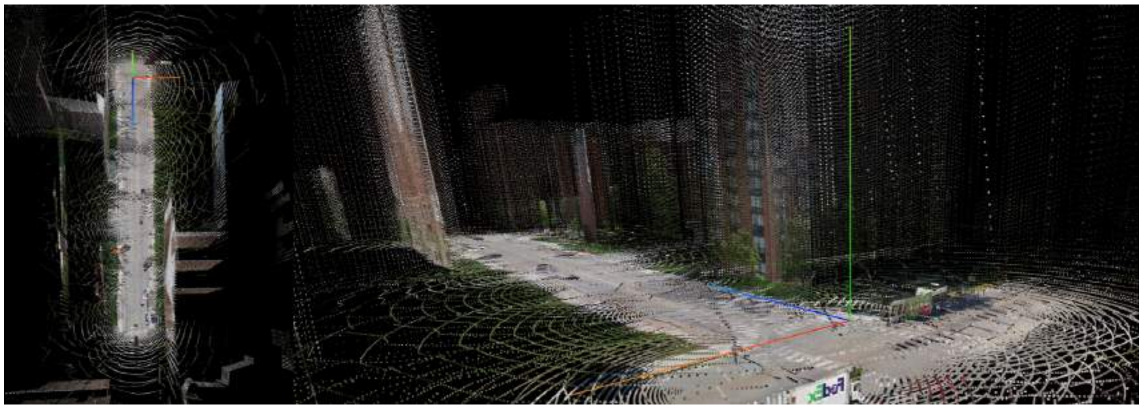
Figure 1. Sample point cloud reconstruction based on the output of merging 6 different panoramic images along a street.

METHOD

In this section we will present the main methods used to implement the software, which has been conceived as an interactive web application based on Three.js and WebGL.
The overall development process can be divided into three parts:

1. Composing a panorama
2. Computing the depth map
3. Creating the point cloud
4. Composing points clouds

The first two parts leverage the use of Google Street View API in order to gather and decode the data needed at the following stages. They also exploit two tiny open-source Javascript libraries, named GSVPano [13] and GSVPanoDepth [14], which helped fetching and composing Google's imagery.

Composing a panorama

The first part of the application has to deal with retrieving the panorama image closest to the specified initial position. By making a call to Google Maps REST API at the following address

```
https://maps.google.com/cbk?output=json&hl=x-local&ll=LAT,
LNG&cb_client=maps_sv&v=3
```

it is possible to obtain the unique identifier of the panorama. Google Street View Service Javascript API allows us to retrieve some information to be used in the next step: the panorama identifier, the available resolutions for the whole panorama, the resolution of the single tiles composing it, the world heading, the real coordinates and eventual neighboring panoramas.
Despite Google Street View may provide resolutions up to 13312x6656 pixels, we considered the resolution of 3328x1664 pixels more than sufficient for our purposes. In this particular case, the objective is to compose in a single image 7x4 tiles having a resolution of 512x512 pixels, since Google doesn't allow to download directly the panoramic image. So, we need to download each of the 28 tiles by using the REST API

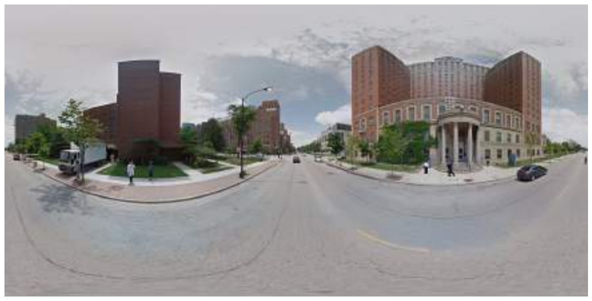
Figure 2. Sample panorama reconstructed from 7x4 tiles

```
https://cbks2.google.com/cbk?cb_client=maps_sv.tactile&
authuser=0&hl=en&panoid=PANORAMAID&output=tile&zoom=
QUALITY&x=XPOS&y=YPOS&TIMESTAMP
```

where in our case QUALITY is 3 and XPOS and YPOS correspond respectively to the position of the desired square tile in the 7x4 image grid. After having combined all of them in a single image by partially overlapping their borders, we obtain a 3328x1664 RGB image corresponding to the desired panorama.

Computing the depth map

Now that we have the panorama image, we need to retrieve its corresponding depth map. Google Maps REST API allows us to download a compressed JSON representation of the depth image from the url

```
http://maps.google.com/cbk?output=json&cb_client=maps_sv&
v=4&dm=1&pm=1&ph=1&hl=en&panoid=PANORAMAID
```

which contains the distance from the camera to the nearest surface at each pixel of the panorama. After having decoded from Base64 the data and having converted it to an array of unsigned 8-bit integers, we can fetch its header information obtaining useful values, like the number of referenced planes. In fact, each pixel in a grid of 512x256 pixels is corresponding to one of several planes, which are given by its normal vector and its distance to the camera. Therefore, in order to calculate the depth at a pixel, we have to determine the intersection point of a ray starting at the center of the camera and the plane

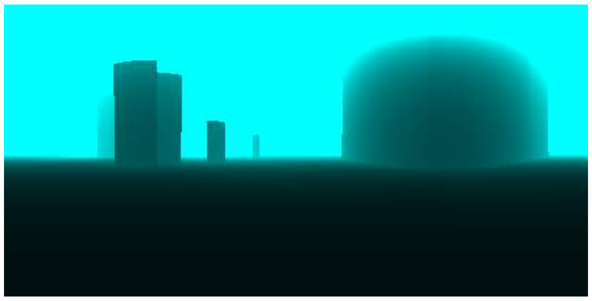
Figure 3. Depth map correponding to the panorama in Figure 1

corresponding to the pixel. Iterating for all the planes, we can then populate our depth map as 32-bit float array of 512x256 elements - which is much lower than the resolution of our RGB panorama image.
As for the computation, for each point we consider its associated plane and we compute its distance as

```
forall indeces x,y do
    $planeIndex \leftarrow indices[y * width + x]$
    $\phi \leftarrow \frac{w-x-1}{w-1} * 2\pi + \frac{\pi}{2}$
    $\theta \leftarrow \frac{h-y-1}{h-1} * \pi$
    $v \leftarrow [\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\phi]$
    if planeIndex > 0 then
        $plane \leftarrow planes[planeIndex]$
        $t = \frac{plane.d}{v \cdot plane.n}$
        $v \leftarrow v * t$
        $depthMap[y * w + (w - x - 1)] \leftarrow \sqrt{v \cdot v}$
    else
        $depthMap[y * w + (w - x - 1)] \leftarrow \infty$
    end
end
```
Algorithm 1: Depth map computation

where *indeces* is an array containing the plane associated to each pixel. In order to obtain a more intuitive image representation of the depth map monodimensional $w * h$ vector, for example we can simply create a colored canvas of width w and height h where each pixel is defined as

```
forall pixels x,y do
    $depth \leftarrow depthMap[y * w + x]$
    $image[4 * (y * w + x) + 0] = 0$
    $image[4 * (y * w + x) + 1] = \frac{depth*255}{50}$
    $image[4 * (y * w + x) + 2] = \frac{depth*255}{50}$
    $image[4 * (y * w + x) + 3] = 255$
end
```
Algorithm 2: Depth map visualization

which will lead to an image similar to the one in Figure 2.

Creating the point cloud

Now that we have the depth information for each pixel, we need to create our point cloud and map each point to its original

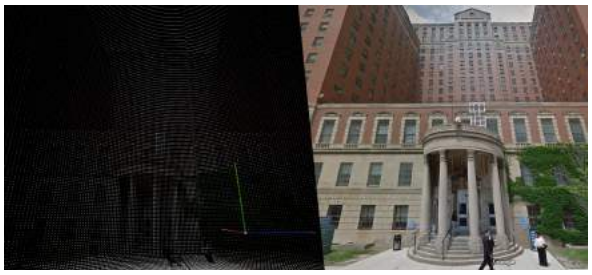
Figure 4. Comparison between a sample building reconstructed as a point cloud and the original displayed on Google Street View

color in the panorama obtained at step 1. Considering $n_{points} = w * h$ points, we define two $n_{points} * 3$ float arrays containing the 3D space position and the color of each point. Then we have to consider that the points of the panorama image originally belonged to a spherical image, so we have to reproject them in space by using the following algorithm:

```
for $y_{depth} \leftarrow 0$ to $h_{depth}$ do
    $lat \leftarrow \frac{y_{depth}*180}{h_{depth}} - 90$
    $r \leftarrow \cos\frac{lat*\pi}{180}$
    for $x_{depth} \leftarrow 0$ to $w_{depth}$ do
        $depth \leftarrow$
          $depthMap[y_{depth} * w_{depth} + (w_{depth} - x_{depth})]$
        $lng \leftarrow (1 - \frac{x_{depth}}{w_{depth}}) * 360 - 180$
        $x_{pos} \leftarrow -r * \cos\frac{lng*\pi}{180}$
        $y_{pos} \leftarrow \sin\frac{lat*\pi}{180}$
        $z_{pos} \leftarrow r * \sin\frac{lng*\pi}{180}$
        $pos_{point} \leftarrow [x_{pos}, y_{pos}, z_{pos}] * depth$
        $x_{norm} \leftarrow = \frac{1-x}{w}$
        $y_{norm} \leftarrow = \frac{y}{h}$
        $x_{color} \leftarrow int(x_{norm} * w_{color})$
        $y_{color} \leftarrow int(y_{norm} * w_{color})$
        $color_{index} \leftarrow y_{color} * w_{color} * 4 + x_{color} * 4$
        $color_point \leftarrow image_{color}[color_{index}]/255$
    end
end
```
Algorithm 3: Point cloud creation

Notice that, in addition to the reprojection, it was necessary to normalize the pixels positions in 2D in order to retrieve colors from the panoramic colored image, which has a different resolution.

Composing points clouds

By exploting the links between locations provided by Google Street View API, we know the identifier and geoposition of the neighboring panoramas. We can then apply iteratively all the previous steps to these new locations in order to populate the 3D scene with other panoramas in order to refine the one we have with more points or to reconstruct a real world path. In our case, we decided to rely on two parameters: the minumum distance between two panoramas and the depth of iteration, considering the first panorama loaded as the root of a tree to be explored in a breadth-first fashion.

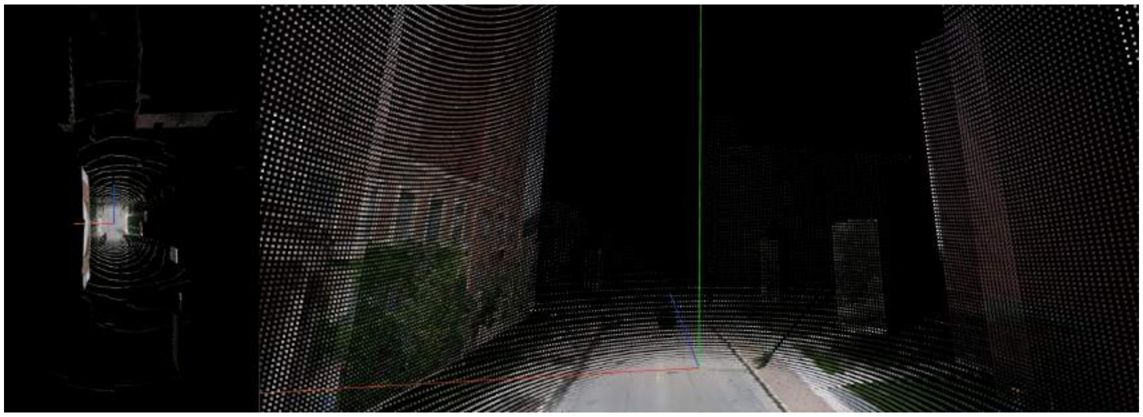
Figure 5. Sample point cloud reconstruction of the panorama in Figure 1 and Figure 2. On the left we can see the scene from a top view, while on the right we have the same scene from a different perspective. We can notice how points are projected onto planes (i.e. buildings and trees in this case) and how their density is much higher when they are closer to the origin of the point cloud.

On average, we noticed Google depth maps generally involve real world distances of 3-200m and we decided to try to build the scene with a scale of 1 virtual unit : 1 meter. So, we load the first point cloud with center in (0,0,0) and then we add the following data with an offset proportional to the real world distance from the first panorama location. By using some geometrical projection and a bit of approximation, we can write the following algorithm to compute the distance vector on the [x,z] plane:

```
Data: Latitute and longitude of a location
Result: [x,z] offset from the center of the scene
$x_{offset} \leftarrow \frac{lon*20037508.34}{180} - x_0$
$z_{offset} \leftarrow \log(\tan((90+lat)*\frac{\pi}{360}))/\frac{\pi}{180}*\frac{20037508.34}{180} - z_0$
if first panorama then
    $x_0 \leftarrow x_{offset}$
    $z_0 \leftarrow z_{offset}$
    return [0,0]
end
return $[x_{offset}, z_{offset}]$
```
Algorithm 4: Offset distance calculation

In addition to the translation, the point cloud needs to be rotated along the *y* vertical axis by the heading specified by the information retrieved from Google Street View. After combining multiple panoramas, we finally obtain a result similar to the one showed in Figure 1.

EVALUATION

Projection and position accuracy

Though we still have not defined any accuracy metric to be optimized, due to the correspondence between virtual units and meters it is possible to make some direct observations from a coarse-grained perspective. Despite on average the output of our method still produces non-photorealistic representations which convey a good idea of geometry, when it comes to bring the 3D reconstruction to a more texture and color representation we can easily notice some smaller issue. For example, in Figure 7 we can notice how the correspondence of depth map and RGB panorama is still not perfect: the border of the building, indeed, is characterized by the light blue of the sky next to it. In addition to this, the projection of the points on the planes often includes points relative to the sky that, in normal conditions, should have a distance tending to infinity (and so they should not be visible). One of the main differences with respect to Micusik's approach [1] is indeed outliers elimination, which in their case is accompained by much more sophisticated machine learning algorithms.

Another issue is represented by far points: when each panorama is loaded, the furthest points become more sparse and accuracy decreases. When combining multiple points clouds, some of them simply don't match and may just add unwanted noise to the reconstruction, as we can see from FIgure 8. A simple solution could be filtering them out basing on their distance when composing multiple panoramas.

Finally, while heading and latitude projection of single panoramas seem to be matching, longitudinal accuracy is still unprecise in the measure of few meters and so vertical planes may result shifted with a sort of "duplicate" effect when composing multiple panoramas. This could be possibly due to the approximation in the formula proposed at the fourth step of the *Method* section.

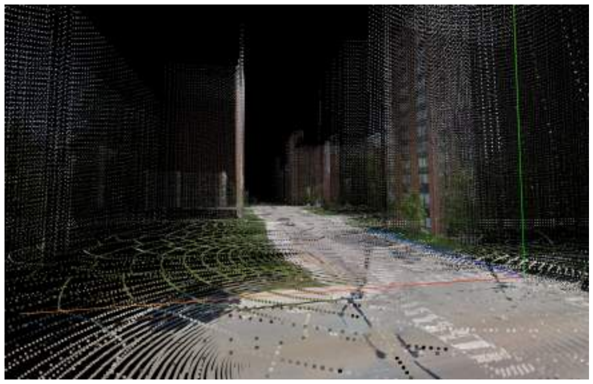
Figure 6. Combination of multiple panoramas along a street.

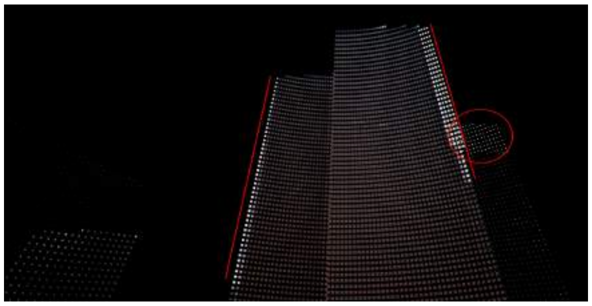

Figure 7. Still imperfect alignment with the panorama image and presence of sky unnecessary data.

Exploring the world

In order to make the evaluations listed above, we considered necessary to build a specific user interface to give the user the possibility to explore the world from different perspectives. In particular, we added tre cameras:

- A static top-down camera directed to the origin, useful to see step-by-step the reconstruction of the city while the algorithm loads consecutive panoramas. Its position is always updated to the offset of the most recently loaded point cloud and its rotation is never changed.
- A rotating camera following elliptic trajectories in order to give a broad perspective view of the whole reconstructed point cloud. Its movements for each frame are defined as

```
$lon \leftarrow lat + 0.15$
$lat \leftarrow max(-85, min(85, lat))$
$\phi \leftarrow (90 - lat) * \frac{\pi}{180}$
$\theta \leftarrow lon * \frac{\pi}{180}$
$pos \leftarrow 100 * [sin\phi \cos\theta, \cos\phi, \sin\theta]$
$lookAt \leftarrow$ last pointcloud center
```

Algorithm 5: Camera trajectory computation

- An interactive camera to allow the user complete freedom in the generated world. The camera moves along the $[x, z]$ plane according to the pressing of the directional arrows on the keyboard and changes orientation basing on the mouse movements.

In addition to this, a minimalistic menu allows changing the camera field of view and reprojecting the last panoramic image onto a sphere around the scene. This last feature is particularly useful for visually matching the point cloud with the panoramic image.

FUTURE WORK

As stated before, this is just an initial approach to 3D modeling from Google Street View imagery. A first future work is to increase the accuracy of the projection of the pixels, especially filtering out unwanted points and leaving only the ones corresponding to the actual buildings. This comprehends both realigning the depth map with the panorama colored image and removing sky and far points in general. In relation to this, we should test which is the optimal number of panoramic images (and points) needed to reconstruct an urban area and the suggested distance between each of them. Also, it would be interesting to reconstruct from the point cloud the meshes of the buildings and to project texures onto the corresponding vertical planes, in order to have more realistic building facades.

On top of that, until now the normal panoramic images has been reprojected onto a sphere for debug purposes, but a possible idea could be to combine it with the point cloud by finding some geometric correspondences, in order to allow the insertion of virtual additional content like in the projects Urban Jungle [10] and Hashima Island [11]. In particular, our work would differentiate from these previous applications thanks to the possibility of exploring a combined world made up of multiple panoramas, where there is a direct correspondence between real-world and virtual positions and distances. Other examples of augmenting Google Street View imagery can be found in [6], [7], [8], [9] and [12].

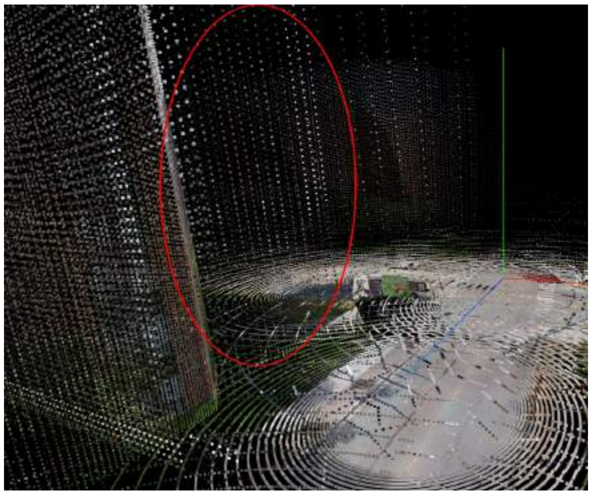

Figure 8. Noisy points of far panoramas sometimes get projected out of correct position. When connecting more relatively close panorama, they could simply be excluded during the computation of the point cloud.

CONCLUSION

In summary, we proposed an alternative approach to the normal use of panoramic images, aimed at visualizing through point clouds the depth information contained in these panoramas. By combining multiple point clouds, we demonstrated how it is possible to create a sparse-point reconstruction of an urban environment in a completely free and automatizable way, creating an explorable 3D world that could be used for many different purposes.

ACKNOWLEDGMENTS

This work was produced under the supervision of Professor Angus G. Forbes, assistant professor in the Department of Computer Science at University of Illinois at Chicago.

REFERENCES

1. Branislav Micusik, Jana Kosecka, *Piecewise Planar City 3D Modeling from Street View Panoramic Sequences*, IEEE Conference on Computer Vision and Pattern Recognition, 2009

2. Pratik Agarwal, Wolfram Burgard, Luciano Spinello, *Metric Localization using Google Street View*, IEEE Conference on Computer Vision and Pattern Recognition, 2015
3. Jay Bolter, Maria Engberg, Blair MacIntyre *Media Studies, Mobile Augmented Reality, and Interaction Design*, ACM Interactions, February 2013
4. Zeljko Medenica, Andrew L. Kun, Tim Paek, Oskar Palinko *Augmented Reality vs. Street Views: A Driving Simulator Study Comparing Two Emerging Navigation Aids*, MobileHCI '11 Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services, 2011
5. Dragomir Anguelov, Carole Dulong, Daniel Filip, Christian Frueh, Stephane Lafon, Richard Lyon, Abhijit Ogale, Luc Vincent, Josh Weaver, *Google Street View: Capturing the World at Street Level*, Computer, vol. 42, 2010
6. Kotaro Hara, Victoria Le, Jon E. Froehlich, *Combining Crowdsourcing and Google Street View to Identify Street-level Accessibility Problems*, CHI '13 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2013
7. Stefaan Ternier, Roland Klemke, Marco Kalz, Patricia, Ulzen, Marcus Specht, *ARLearn: Augmented Reality Meets Augmented Virtuality*, J-jucs journal vol. 18, August 2012
8. Mark Graham, Matthew Zook, Andrew Boulton, *Augmented reality in urban places: contested content and the duplicity of code*, Transactions of the Institute of British Geographers, vol. 38, July 2013
9. R. Diaconu, J. Keller, E. Triponez, *HybridEarth: Social Mixed Reality at Planet Scale*, Consumer Communications and Networking Conference (CCNC), 2014 IEEE
10. Urban Jungle, http://inear.se/urbanjungle
11. Hashima Island - The Forgotten World, http://hashima-island.co.uk
12. Floating Shiny Knot, http://www.clicktorelease.com/code/streetViewReflectionMapping
13. GSVPano Library, https://github.com/heganoo/GSVPano
14. GSVPanoDepth Library, https://github.com/proog128/GSVPanoDepth.js/tree/master