Realtime projective multi-texturing of pointclouds and meshes for a realistic street-view web navigation

Alexandre Devaux, Mathieu Brédif

Realtime Projective Multi-Texturing of Pointclouds and Meshes for a Realistic Street-View Web Navigation

Alexandre Devaux, Mathieu Brédif*
IGN / SRIG - Laboratoire MATIS - Universite Paris Est - Saint-Mande, FRANCE

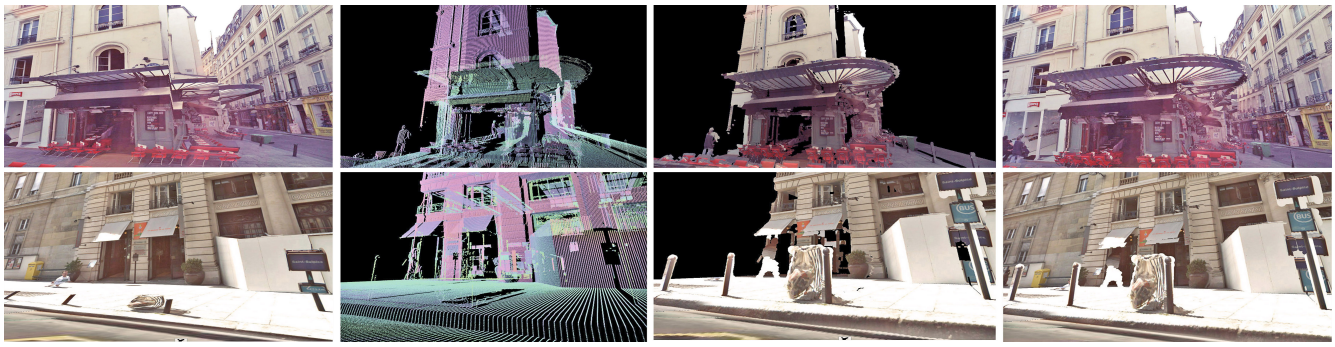

Figure 1: *Rendering using mesh and pointCloud. a) Projective texturing on simplified planar geometry (point of view one meter distant from original photography position) b) Original Point cloud c) Projective texturing on Point Cloud. d) Hybrid rendering.*

Abstract

Street-view web applications have now gained widespread popularity. Targeting the general public, they offer ease of use, but while they allow efficient navigation from a pedestrian level, the immersive quality of such renderings is still low. The user is usually stuck at specific positions and transitions bring out artefacts, in particular parallax and aliasing. We propose a method to enhance the realism of street view navigation systems using a hybrid rendering based on realtime projective texturing on meshes and pointclouds with occlusion handling, requiring extremely minimized pre-processing steps allowing fast data update, progressive streaming (mesh-based approximation, with point cloud details) and unaltered raw data precise visualization.

Keywords: Image Based Rendering, Projective Texturing, Street-view, WebGL, Point Based Rendering, GIS

Concepts: •**Information systems** → *Web applications;* •**Human-centered computing** → *Geographic visualization;* •**Computing methodologies** → *Image-based rendering; Point-based models;*

1 Introduction

City models visualization attained the last decade a high level of quality with realistic fly-throughs even on web platforms. However, displaying realistic virtual environements of real-world places at a pedestrian level is still a challenging task. On the one hand, accurate 3D geometry and calibrated images may be acquired and used, within an expensive process, to model a high quality environment with a sufficiently high level of details. On the other hand, with a more approximate geometry, different image-based rendering (IBR) techniques have coped to create satisfying renderings without requiring a detailed modelisation.

In the early days, Light Field Rendering or Unstructured Lumigraph Rendering [Buehler et al. 2001] allowed the real-time visualization of unstructured sets of cameras with or without relying on an approximate model (so-called geometry proxy). More recently, some impressive results [Chaurasia et al. 2013] have been achieved to provide plausible free-viewpoint navigation using local warps with few input calibrated images, which can cope with unreliable geometry. Google Street View™ uses a sparse set of panoramic images. Transitions between captured viewpoints employ cross-fading and geometry-aware warping to approximate the expected optical flow. Microsoft Photosynth™ displays an unstructured collection of photographs in the reconstructed spatial layout and applies image-space transformations and blending transitions. Despite their relative simplicity, these systems create reasonably compelling 3D experiences and they are quite suited for web platforms. The biggest remaining drawback is that such systems typically restrict viewers to be very close to one of the capture points. They also tend to remove small depth details such as poles or other light urban furnitures in the targeted street-view application.

The point-based representation is getting more popular since two decades, thanks to the progress of Structure From Motion, lidar technology, and light RGB-D system such as the kinect. The litterature on point-based rendering is therefore pretty dense. The most trivial way to render point clouds is to use one color per point and project it to the screen. It usually creates holes and low color resolution even for dense point clouds. The other classical approach is to mesh the point cloud using [Kazhdan et al. 2006] or 3D Delaunay reconstruction offering a hole-free model. Then all images are generally merged into a texture atlas to get a full-resolution unified texture. The usual drawback is to add artefacts to the original geometry and it lacks small details. Potree, a popular WebGL pointcloud renderer, uses an octree to spatialize the data for fast streaming and add compelling rendering algorithms such as Eye-Dome-Lighting (EDL) to improve depth perception without the need of normals on points. Combining point cloud with mesh is less studied. [Pagés et al. 2015] explored the mixing of meshes with splats creating hybrid renderings but the methodology requires heavy pre-processing such as Poisson reconstruction and geometry shape segmentation.

Some experiments on the perception of visual artifacts in image-based rendering have been conducted by [Vangorp et al. 2011] in urban areas and show for example that when cross-fading between panoramic images, shorter transition durations are preferred. Another side effect we discovered is that the eye is more distracted

*E-Mail for all authors:firstname.lastname@ign.fr

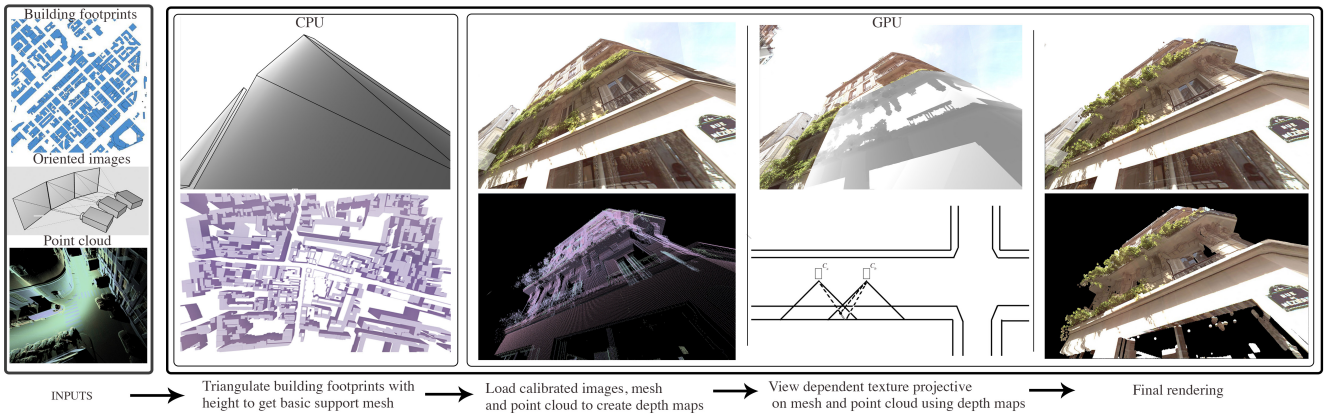

Figure 2: *Overall application pipeline*

by deformed textured meshes even though geometrically closer to reality than by very approximate planar geometries inducing more parallax effects (fig. 3). Creating a simplified mesh from the point-cloud using the previously discussed methods requires very refined algorithms working on high quality point clouds to get acceptable results inducing also heavier data to be transmitted and rendered. Another artefact often encountered while creating meshes is the loss of small details. In our case, we found it very hard to create meshes that could keep geometry such as vegetation, poles, balconies, and, more generally, urban furnitures. This is problematic as the user moves through the streets, as it means that the texture will be projected on an incorrect geometry, typically flattened on the ground or façades, inducing an unpleasant rendering (fig. 1.a).

Figure 3: *Comparaison of support for projective texturing. Poisson reconstructed mesh from Lidar (left) and simple planar geometry (right)*

For this reason, a view-dependant projection of oriented images on a simplified mesh (composed by few quads for the facade and the road such as LOD1 building model) is a very efficient solution as a base rendering. Its simplicity makes it extremely fast to stream and very light to render not to mention its increasing disponibility as OpenData. Another important feature is that it allows easy daily update as new data is acquired it is instantly visible without pre-processing. As in [Brédif 2013], we compute the mesh from 2D building footprints and a low resolution height-map (or Digital Terrain Model, DTM). We extend this method to the web environment by streaming the DTM and the 3D gutter linestrings only and reconstructing the simplified mesh online (in the client) using a constrained Delaunay triangulation.

Our main contributions are (i) to add geometric details and refine occlusion handling, by creating detailed depth maps using the point cloud (ii) to use realtime projective texturing on both the mesh and the point cloud to create a compelling view-dependent hybrid rendering that does not rely on any pointcloud or texture atlas preprocessing. The proposed method thus contributes on reducing parallax distortions, blurring, ghosting and popping errors that usually distort the scene appearance through IBR in a web-friendly manner as the lightweight base mesh and a low-res projective texture are streamed almost instantly, producing a reasonnable view (fig. 1.a), while local pointcloud chunks and higher resolution textures are streamed, updating continuously the visualization (fig. 1.d).

2 Overview of the method

Working with mobile-mapping acquired data, the application deals with oriented images taken through the vehicle trajectory, hence a small number of images usually see the same spot. This specificity makes projective texturing an adequate choice. The basic application pipeline is as follow (fig. 2):

1. Get the building footprints around the viewing position v and triangulate to create basic geometry proxy (mesh)
2. Get the projective textures $\{t_k\}$ (images and calibration) and lidar point cloud $\{c_l\}$ close to v.
3. Initialize the depth buffer of each texture t_k with a depth-pass on the base mesh and update it as new chunks c_l are received.
4. Hybrid (mesh + point cloud) projective texturing with texture distortion and occlusion handling support.

3 Point Cloud Visualization Optimization

Raw lidar data comes with a lot of information, such as 3D position, reflectance, number of echoes, etc but does not bring any color information. Offline colorization [Waechter et al. 2014] can provide good results using oriented images with method classically applied to meshes. Though it usually creates one colour per point or generate huge texture atlases.

Our method does not need any pre-processing to get very pleasing point clouds renderings. We rely on the same online projective texturing we use for the mesh hence mutualizing textures for fast web transmission. Another point to justify the whole methodology of projective texturing here is that the rendering is perfectly equal to

the original images when viewed from their viewpoint location, and degrades gracefully in-between viewpoint locations. Pre-processed textured meshes or point clouds are optimized to provide the best global representation of acquired data from any point of view which necessarily distort reality.

No matter the density of the point cloud, setting a unique color per point will always create a subsampled look compared to the original image. We experimented the creation of splats directly in the GPU in a single pass using barycentric coordinates. When receiving points from the server, we create for each point a triangle with co-located vertices but with differing barycentric coordinates so that splats may be created in the vertex shader (as geometry shaders are unavailable in WebGL) and change their projected geometry (as ellipses) in the fragment shader.

Alternatively to using splats, we propose a perspective-correct projective texture mapping of point sprites. Similarly, we approximate the point rendering by a set of planar patches, but instead of rasterizing one or two coplanar triangles per point, a point sprite (i.e. a screenspace square) is used, minimizing the primitive setup and rasterization workloads. The 3D planar support is left unchanged as the plane going through the 3D point P and orthogonal to its normal N. The main idea is to compute the homography H induced by this plane from the synthesized view to the input view [Hartley and Zisserman 2004]. Using projective geometry notations, we denote the respective input and synthesized view projections as $p = M(P-C)$ and $p' = M'(P-C')$ where C, C' are the viewpoints, M, M' are the view-projection 3x3 matrices and p, p' are the image coordinates in screen-pixel and texture coordinates respectively:

$$H = M' \left(I + \frac{(C - C')N^T}{N^T(P - C)} \right) M^{-1}$$

Implementation-wise, this homography may be computed in the vertex shader, passed as a $mat3$ varying and evaluated as $p'_{tex} = Hp_{screen}$ in the fragment shader. The vertex shader computation is performed as $H = \left(M' + \frac{E'N^T}{N^T(P-C)} \right) M^{-1}$ with:

- View uniforms M^{-1} and C
- Per-texture uniforms M' and $E' = M'(C - C')$
- Per-point attributes P and N ($N = P - C$ by default)

Figure 4 shows the large difference of rendering quality when computing one color per point versus texturing the whole point surface.

Figure 4: *Comparaison of point cloud rendering. (left) Basic point sprites, 1 color per point. (right) Projective textured point sprites*

Raw lidar point clouds do not generally come with normals. If unavailable, we could compute a screen-space normal map using a depth-prepass and a normal-map lookup, which would yield some side-benefits but would slow down the rendering on light GPU configurations with an extra pass. We propose a much simpler solution leaving the point sprites at their original behaviour: always facing the camera with an eye-vector normal approximation. Application-wise, this introduces a screen-space dilation of the point cloud support that fills holes and nicely convey the cylindricity on poles and other thin objects typically acquired by a single lidar scan line (where a normal has no real meaning). Note in figure 4 how high resolution projective textures handle well sparse point clouds.

4 Occlusions

A classic artefact when navigating through Street View applications are bad occlusions handling. Namely, texturing part of the geometry with an irrelevant texture. Examples are (i) disocclusions (e.g. at crossroads, fig. 6), where a disoccluded geometry is textured by a viewpoint from which it is not visible and (ii) unmodeled geometry (e.g. poles projected on the ground, fig. 1.a).

4.1 Per-texture Depth Map

We adapted the shadow mapping-like technique from mesh-only [Brédif 2013] to hybrid rendering (mesh+pointcloud). This is performed by computing an off-line rendering of the scene from each texture position, creating a depth texture that is pixel-wise aligned with the (undistorted) input color image (fig. 5). While rendering, a simple depth (shadow) test is performed between a texturing depthmap lookup and the depth relative to the sampled texture viewpoint. This technique is relatively cheap as it only costs a one-time depth-pass during the initialization phase and a comparison against a texture sample (i.e. a shadow test) in the rendering phase.

Figure 5: *Depth map for one projective texture (right) mixed with original imagery from other projective textures (left).*

4.2 Hybrid View-Dependent Multi-Texture Mapping

Compared to other View-Dependent Multi-Texture Mapping approaches, we propose two extensions. First, to address the web context, the projective texture calibrations (position, rotation and projection matrices) are stored in a PostgreSQL/PostGIS database with a Node.js frontend, enabling fast queries for nearest textures from the current view position and pre-fetching based on query regions. Second, the hybrid rendering of an approximate mesh with a more accurate but incomplete pointcloud renders well the added pointcloud protrusion details (e.g. balconies, poles...) but not the depression details where the approximate mesh was overestimated

(windows, doors, insets...). To cope with this issue, the mesh is rendered with an offset (e.g. 1m) that lets points belonging to small depressions pass the depth test.

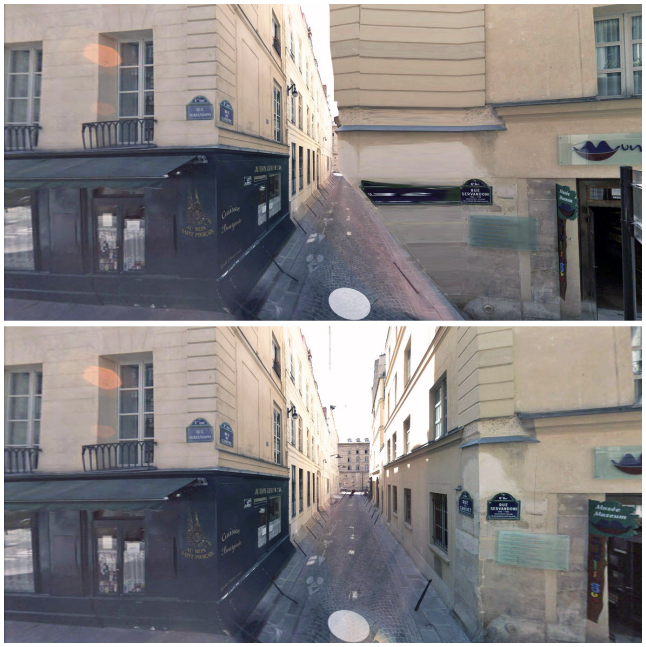

Figure 6: *View-dependent texturing with (bottom) and without (top) occlusion handling.*

5 Results and Limitations

The use of both a simplified mesh planar geometry and a point cloud offers a compelling visualization. As we said earlier they are combining in such a way to enhance realism. However, as we use simplified planar geometry computed from external databases (building footprints), its geometry can have an important shift with mobile mapping geometry. One way to deal with that is to bundle adjust the mesh geometry using trajectory information if available. As we load oriented images, they have absolute position informations and usually comes with height above ground information. Hence we can adjust the local altitude of the mesh with this information creating a local Digital Terrain Model (DTM), again, without preprocessing. Another way is to analyze the point cloud estimating the minimal height of the points gives already enough information to adjust the height of the mesh. Those are classical problematics when dealing with different sources of geospatial data.

6 Conclusion

The solution proposed in this paper contributes to web immersive navigation system enhancing its realism while keeping precise data representation. It can be improved in different ways. Even if there is a need to keep access to raw lidar point clouds for professional usage, pre-processing it for visualization aspect will lighten it for streaming as splats precomputing can save a lot of bandwith in streaming and in GPU memory. We are currently working on improving the streaming of the data, investigating with Shape Resource Container (SRC) [Limper et al. 2014] which offers great optimization for textured mesh level of detailed transmission and could also work for splats.

Acknowledgements

The implementation of this work and a sample dataset are available with an open source license at https://github.com/iTowns

References

BRÉDIF, M. 2013. Image-based rendering of lod1 3d city models for traffic-augmented immersive street-view navigation. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences 1*, 3, 7–11.

BUEHLER, C., BOSSE, M., MCMILLAN, L., GORTLER, S., AND COHEN, M. 2001. Unstructured lumigraph rendering. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 425–432.

CHAURASIA, G., DUCHENE, S., SORKINE-HORNUNG, O., AND DRETTAKIS, G. 2013. Depth synthesis and local warps for plausible image-based navigation. *ACM Trans. Graph. 32*, 3 (July), 30:1–30:12.

HARTLEY, R. I., AND ZISSERMAN, A. 2004. *Multiple View Geometry in Computer Vision*, second ed. Cambridge University Press, ISBN: 0521540518.

KAZHDAN, M., BOLITHO, M., AND HOPPE, H. 2006. Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SGP '06, 61–70.

LIMPER, M., THÖNER, M., BEHR, J., AND FELLNER, D. W. 2014. Src - a streamable format for generalized web-based 3d data transmission. In *Proceedings of the 19th International ACM Conference on 3D Web Technologies*, ACM, New York, NY, USA, Web3D '14, 35–43.

PAGÉS, R., GARCÍA, S., BERJÓN, D., AND MORÁN, F. 2015. Splash: A hybrid 3d modeling/rendering approach mixing splats and meshes. In *Proceedings of the 20th International Conference on 3D Web Technology*, ACM, New York, NY, USA, Web3D '15, 231–234.

VANGORP, P., CHAURASIA, G., LAFFONT, P.-Y., FLEMING, R., AND DRETTAKIS, G. 2011. Perception of visual artifacts in image-based rendering of façades. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering) 30*, 4 (07), 1241–1250.

WAECHTER, M., MOEHRLE, N., AND GOESELE, M. 2014. Let there be color! Large-scale texturing of 3D reconstructions. In *ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., vol. 8693 of *Lecture Notes in Computer Science*. Springer International Publishing, 836–850.