PROJECTIVE MULTITEXTURING OF CURRENT 3D CITY MODELS AND POINT CLOUDS WITH MANY HISTORICAL IMAGES

Maria Scarlleth Gomes de Castro [1,2], Mathieu Brédif [2,*]

[1] Ecole Polytechnique, IP Paris, 91120 France
[2] LASTIG, Univ Gustave Eiffel, IGN, ENSG, 94165 France
mathieu.bredif@ign.fr

Commission IV, WG IV/9

KEY WORDS: Projective texturing, 3D City Model, Point cloud, Cultural Heritage, Image-Based Rendering.

ABSTRACT:

Iconographic image collections are a cultural heritage that could reach a larger audience by proposing their immersive presentation in a 3D web application. Proposing a historical street view application, based on these historical images, raises issues such as the unavailability of historical 3D models of the scene and the heterogeneity and sparsity of these photographs. We propose to use the 3D city and terrain models of the current scene, as well as a 3D point cloud if available, to simultaneously reproject and blend many historical images using an image-based rendering approach. Our contributions raise significantly the number of projective textures blended per rendering pass (typically from 8 to 40) on triangular meshes (of the 3D city and terrain models) and on point clouds. As a first step to tackle diachrony artifacts, we also propose a simple point cloud classification to filter in the shader the points corresponding to building or terrain details from the points corresponding to transient objects.

Figure 1. Sample photographs/postcards from the image collection "Fonds Charles Gros". ©Nicéphore Niépce Museum.

1. INTRODUCTION

1.1 Context

Galleries, Libraries, Archives, and Museums (GLAM) are collecting massive image collections (Figure 1), which are reflections of the past that could benefit from an immersive presentation in a 3D virtual world. This study focuses on the case of image collections that have already been georeferenced (*i.e.* all camera parameters are known: position, orientation, focal length...), but for which the photogrammetric derivation of a 3D model of the scene is not tractable, because of image quality, moving objects or image sparsity preventing multi-view reconstruction. In other words, we would like to propose a street view navigation through historical images from GLAM providers despite the absence of historical geometric models, by leveraging the current geometry of 3D city models and point clouds (Figure 2). This 3D immersive experience could help valorize the iconographic cultural heritage like photographic collections to better understand and analyze the past, with applications in humanities and tourism for instance.

To maximize outreach and replicability, the proposed approach should leverage web technologies, which means rendering using a WebGL framework to enable visualization in any recent web browser and access to geospatial datasets using standard OGC web services. If required, any joint analysis of the different datasets should occur on the client at rendering time, rather than at preprocessing time, so that the image and geospatial dataset providers may expose images, 3D city models (terrain and buildings), and point cloud datasets through existing standard web services such as IIIF, WMTS, WFS, and 3D tiles, with no application-dependent preprocessing and no coupling between the different datasets.

1.2 Related Work

The georeferencing of such iconographic cultural heritage collections is still an active topic of research, which may be addressed by an interactive process, using user selection of point correspondences between the historic image and the current city model, point cloud or images (Blettery et al., 2021).

Presentation of historic image collections has been addressed in a 3D web context by providing interactive web geovisualization applications (Bruschke et al., 2017, Paiz-Reyes et al., 2021) that help the user understand the coverage of the dataset and discover photographs of interest through interactive navigation. These are however only reprojecting a few image textures at a time.

Due to the heterogeneity of the photographs in terms of color, details, and time, the precomputation of a texture by merging all photographs into a single texture (Waechter et al., 2014) used to give a historical appearance to the current city model is likely to give poor results. This context thus calls for an image-based

* Corresponding author

rendering approach that can provide view-dependent texturing by virtually replacing the cameras with projectors that project their images back to the scene, with blending weights depending on the view camera parameters (Brédif, 2013).

Using densely sampled image collections and heavy joint precomputations on all images and the scene geometry, very high quality and real-time immersive navigation is achievable (Hedman et al., 2016). However, even if such datasets were available, it would require some duplication of the scene geometry for each texturing image and it would introduce coupling between the scene geometry and the image datasets.

In contrast, we propose to render the geometry of the current 3D model without any pre-processing, by reprojecting all available images of the collection on the first surface of the current 3D model, using a multi-texturing shadow-mapping approach (Brédif, 2013). This approach presents however bottlenecks that limit the number of simultaneously projected image textures.

This paper identifies and proposes workarounds for the 3 main bottlenecks of this approach that limit the maximum number of projected image textures per rendering pass : i) the occupancy of texture slots, ii) the occupancy of `varying`s when rendering point clouds and iii) the depth map updates for visibility estimation from texturing cameras. We also propose a simple estimation of the relevance of point clouds for texturing based on their distance from the 3D city model, as a first attempt to address diachrony artifacts.

2. METHODOLOGY

2.1 Texture 2D arrays

The trivial implementation in WebGL of projective multi-texturing with shadow mapping uses two texture slots per projective texture: one slot for the color texture and one for the floating-point depth texture. However, there is a hardware limit on the maximum number of usable texture slots in vertex and fragment shaders. We propose here to lift this linear requirement in the number of texture slots by using only two `texture2DArray`s. A `texture2DArray` is a texture type that groups in a single texture slot many textures as 2D slices of a 3D texture. As in all shadow mapping approaches, whenever the geometry of the scene changes, the depth map of the scene maintained for each texture must be updated by rendering the scene from the perspective of the texturing camera and saving the depth to a depth map (the color is discarded). A nice feature is that `texture2DArray` slices may be defined as render targets so that the implementation of the render to texture of the depth map for texturing cameras only requires to target a `texture2DArray` slice rather than a `texture2D` target.

The main induced constraint is that all slices must have the same width and height. This is the case when images have been scanned with homogeneous settings from similar physical photographs. Thus, a texture array with maximal width and height may be allocated to store all images in its slices, either using limited upscaling or padding. Note that, if padding is used, the projection matrix should be updated to account for the padded image. Furthermore, if texturing images have heterogeneous dimensions, then texture virtualization could be used to split images in chunks of equal sizes (Mittring and Crytek, 2008). An alternative solution would have been to pack the textures in a texture atlas by juxtaposing them in a very large 2D texture. Indeed these approaches may be combined by using an array of texture atlases or using a `texture2DArray` for storing the virtual texture tiles on the GPU. This virtual texturing approach is left for future work.

In practice, the proposed approach takes advantage of the possibility to have depth texture dimensions that do not match the dimensions (or even aspect ratios!) of the projected image, so that all depth maps have the same resolution and, thus, can be all stacked together in a single depth `texture2DArray`. The shared resolution of the depth map slices is a trade-off parameter between i) GPU memory usage and depth rendering time and ii) accuracy of the visibility estimate to project the image only on the first surface as seen by the texturing camera.

The use of only two `texture2DArray`s, one for images and one for all depth maps limits the use of texture slots to a constant number, instead of 2 per projective textures. This removes an upper bound on projective textures : $\lfloor \texttt{MAX\_TEXTURE\_IMAGE\_UNITS}/2 \rfloor$ (typically: $\lfloor 16/2 \rfloor = 8$ simultaneous textures).

2.2 Progressive Depth Map Updates

Even when the scene is static, the massive data volumes of the scene geometry (terrain, buildings, point clouds) call for level of detail approaches that only stream to the client the part of the dataset that is currently visible. Therefore, the rendered 3D scene is continuously updated as different levels of detail are selected or deselected for rendering. Thus, the depth map approach, used for projecting images only on the first scene surface as seen from the texturing camera, requires the depth map of the scene from a texturing camera to be updated whenever any visible part of the scene is loaded or unloaded. This workload is linear in the number of projective textures, thus, when this becomes the limiting factor, these depth map updates directly limit the rendering frame rate.

Following a progressive approach similar to (Schütz et al., 2020), we propose to limit the number $D_{\max}$ of depth map updates to a fixed budget per frame, while prioritizing the depth map updates of textures that have the most impact on the current view. The determination of the update priority is for now simplistically the distance between the camera centers. This update is progressive in the sense that if a scene with T projective textures is not changing, then all depth maps will be updated after $\frac{T}{D_{\max}}$ frames. This number $D_{\max}$ may be dynamically tuned to raise the frame rate (at the cost of a convergence after more frames) or instead lower the frame rate (to accelerate the convergence, measured in frames).

2.3 Point cloud rendering

Trivial projective texturing of point clouds uses the same texture coordinates for all pixels corresponding to a 3D point. Whenever the screen size of a point is larger than 1 pixel, this produces a constant color region. This aliasing may be removed by considering the local tangent plane of the point when computing the texture coordinates.

Let us consider the view camera, and the texture camera i, of respective projection equations $p = M(P - C)$ and $p_i = M_i(P - C_i)$, where p is the pixel coordinate in screen space, p_i is the corresponding point in texture space, P is the 3D point of the scene to be textured, C is the viewer position, C_i are

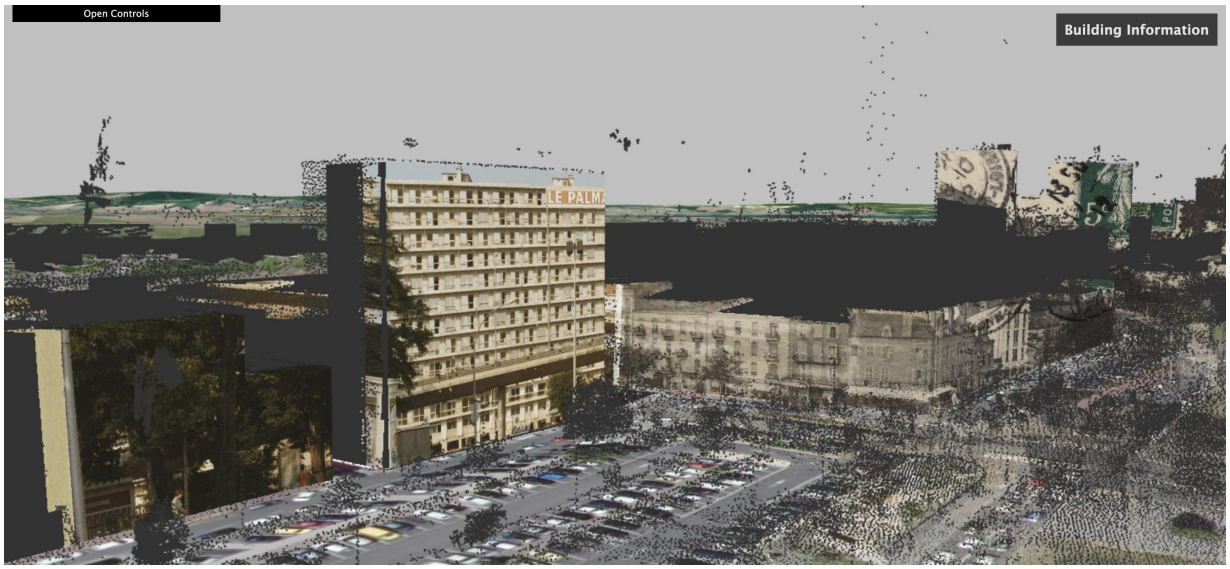

Figure 2. Current 3D city and terrain model and mobile mapping point cloud textured on the fly in real-time using historical images.

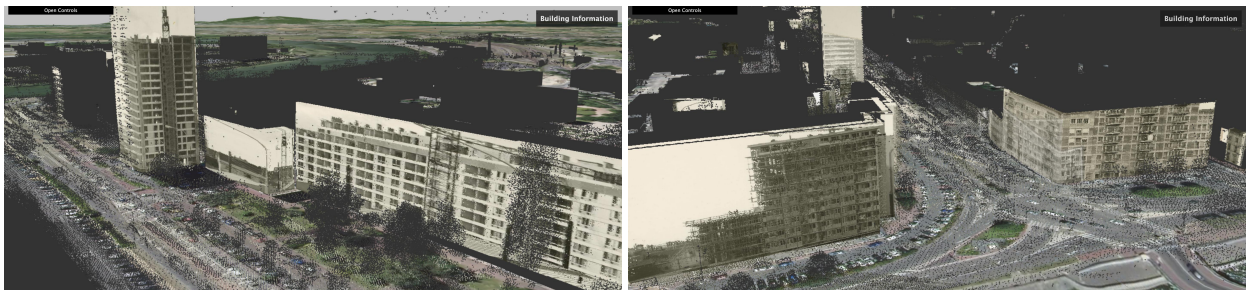

Figure 3. (a) The photograph is texturing the fully built tower, although the building was still in construction when the photograph was taken. (b) On the left, the bright area corresponds to sky pixels texturing a building that was not built yet when the photograph was taken.

the texture camera center points and M, M_i are 3x3 projective matrices. If the surface normal is known at P, then (P, N) denotes its tangent plane, which induces a homography transform H_i from the screen p to each texture coordinates p_i (Hartley and Zisserman, 2004):

$$H_i = M_i \left(I + \frac{(C - C_i)N^T}{N^T(P - C)} \right) M^{-1} \quad (1)$$

$$= \left(M_i + \frac{E_i N^T}{N^T(P - C)} \right) M^{-1} \quad (2)$$

where $E_i = M_i(C - C_i)$ are known as the epipoles in the texture cameras, considering each stereo pair (view camera, texture camera i). If the point normal is unknown, then the front-facing guess $(P - C)$ is used to estimate a best-effort homography in the neighborhood of p.

Considering a 3D point cloud with normals, each point defines a tangent plane on which reprojection homographies H_i may be computed. (Devaux and Brédif, 2016) proposed to factor the computation of H_i in the vertex shader and pass it to the fragment shader as a `varying mat3`. This enables very efficient computation of the texture coordinates as $p_i = H_i p$ in the fragment shader for each fragment and each texture. This however uses 3x3=9 `varying` components per texture, which effectively limits the number of texture reprojections per rendering pass, as there is a system-dependent maximum number of `varying` components. Assuming no other `varying`s, the limit on the number of textures is then : $\lfloor$`MAX_VARYING_COMPONENTS`$/9\rfloor$, a typical value being $\lfloor 124/9 \rfloor = 12$.

	(Devaux and Brédif, 2016)	(Proposed)
JS	M^{-1}, C, E_i, M_i	$M^{-1}, C, E_i, H'_i{=}M_i M^{-1}$
VS	$H_i{=}\left(M_i{+}\frac{E_i N^T}{N^T(P-C)}\right)M^{-1}$	$N'{=}\frac{M^{-T}N}{N^T(P-C)}$
FS	$p_i{=}H_i p$	$p_i{=}H'_i p{+}(N'{\cdot}p)E_i$
`varying`	H_i: 9 floats per texture	N': 3 floats in total

Table 1. Computation of the reprojection p_i in texture i of the screen-space point p provided by `FragCoord`. The first line (JS) contains the uniforms computed on the CPU in javascript, (VS) shows the `varying`s evaluated in the vertex shader, and (FS) provides the evaluation of the texture coordinate p_i in the fragment shader. P and N are the position and normal point attributes, N defaulting to $(P - C)$ if unavailable.

Instead of exhausting the available `varying` components by computing all H_i homographies in the vertex shader, we propose to compute a single `vec3` $N' = \frac{M^{-T}N}{N^T(P-C)}$ in the vertex shader, at the cost of a slight increase in computation in the fragment shader (Table 1). This effectively lifts the bottleneck on the number of `varying` components.

Figure 4. Point Filtering: (top left) On the front, the point cloud contains tree samples which are textured by the historical photograph, while it causes on the back shadows (untextured regions) on the buildings. (top right) Points away from the terrain and buildings remain black (they could also be discarded), while points next to buildings are textured, increasing their geometric details. (bottom) Other views showing that the online point cloud classification prevents shadowing artifacts on the buildings and texturing of the trees with building pixels.

2.4 Texture Visibility Estimate

The proposed approach does not require any joint preprocessing of the various datasets (3D city models, terrain, point clouds, images...). This prevents the computation costs of preprocessing and ensures flexibility. However, there might be inconsistencies due to diachrony and semantics across datasets. For instance, a building may be non-existent, or even in construction, on a photograph while its current geometry is present in the 3D city model (Figure 3). Also, a mobile mapping point cloud samples points on the ground and on buildings, but also on transient objects like vehicles, pedestrians, and trees.

If a semantic classification of the point cloud is available, we propose to only consider non-transient classes for texturing and depth map estimation. This will prevent respectively texturing a transient object with photographs (which are very unlikely to contain the color of a coherent transient object at the same location), and casting a shadow on further non-transient objects.

If points have no explicit classification providing readily a segmentation between transient and non-transient points, then we propose to only keep 3D points that have a depth within a given distance ϵ of the 3D city buildings and terrain models, considering that, in this case, these points correspond to geometric details on the 3D model. This can be easily implemented in the shadow testing for visibility estimation in texture space. First, the depth maps in texture space are calculated only considering the 3D model (buildings, terrain...) but not the 3D point cloud. Second, instead of keeping all the 3D points whose depth in texture camera space is lower than the texture depth map (corresponding to the 3D city model), we only keep points whose depth is within a distance ϵ of the texture depth map (Figure 4).

3. RESULTS

The proposed approach has been implemented using iTowns, a Three.js-based framework written in Javascript/WebGL for visualizing 3D geospatial data. The geospatial datasets are accessed using standard web services. The application connects to a WMTS server to retrieve elevation and orthophotographic tiles for the terrain. The LOD 1 buildings are accessed by WFS tiles of 2DZ polygons (and extruded in 3D in the web client) and the point clouds are served a 3DTiles dataset. For now, the image files are served over HTTP as the resolution is moderate, but any image server could be implemented (IIIF, IIP...). Image parameters are available as a WFS service where the feature is the 3D point location of the image center and the other extrinsic and intrinsic parameters are available as feature properties.

Figure 5 shows some example views where the viewer is placed at the location where a photograph was taken. This enables a view that focuses on a given photograph but that benefits from the context of the current, possibly textured, datasets (city, terrain, and point cloud). This mode of visualization enables one to look at an image while keeping the 3D immersion provided by the embedding 3D scene. Figure 6 presents screenshots of the application as the user navigates in the scene. The frame rate was capped at 60 frames per second on a laptop with a GeForce GTX 980 GPU, using 5 depth map updates per frame and 40 projective textures. Thus the rendering is converging after 40/5=8 frames or 8/60=0.13s after the last scene visibility update.

A strength of this image-based rendering approach is that it can cope well with imprecise geometries of the scene or inaccurate georeferencing of the photographs. These imprecisions only introduce slight misalignment artifacts at the silhouettes of objects (figure 2). These misalignments are even not visible any-

more when the camera is placed at the position of the texture (figure 5), as the texture reprojection homography is no longer dependent on the scene for colocated view and texturing cameras.

Figure 5. Focus and context configuration: In these 4 cases, the camera of the synthesized view is set to a zoomed-out version of one of the cameras that acquired a photograph of the dataset. Thus, this photograph is shown at the center of the view, with some context around it and in front of it (where the point cloud and the terrain are occluding the photograph).

4. CONCLUSION

The proposed approach significantly raises the maximum number of projective textures that may be applied at interactive frame rates in a single rendering pass, by addressing the 3 most limiting factors: the occupancy of texture slots, the occupancy of `varyings` for point cloud rendering, and the required depth map updates for visibility estimation from texturing cameras. Precisely, this raises from 8 to 40 the number of simultaneous projective textures that may be blended in a single rendering pass.

Figure 6. Snapshots of an animation: the user is moving and turning to its right, showcasing how the reprojected photographs are blended together to give some feeling of immersion in a 3D virtual world.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-V-4-2022-213-2022 | © Author(s) 2022. CC BY 4.0 License.

To go further on the same approach, the next limiting factors are (i) the occupancy of uniforms, which may be addressed by packing them into textures, at the cost of more texture sampling operations, and (ii) the absence of culling on the GPU. Better culling could be implemented on the CPU by testing the bounding boxes of scene geometries against the frustums of texturing cameras, similar to (Hedman et al., 2016) by leveraging the existing bounding volume hierarchies of geospatial data access services (*e.g.* quad-trees or octrees). This culling at the object level on the CPU could then be refined at the fragment level on the GPU using an acceleration structure that would prevent from iterating over all the textures selected by the CPU, but only to those who have the 3D position of the current fragment in their frustum.

The handling of diachrony between datasets definitely deserves further research. An interesting approach is the modeling of spatio-temporal city models (Schindler and Dellaert, 2012), which models not only a temporal snapshot of the city but which compactly encodes and is able to deliver the geometry of the city at any date. Using such a 4D city model in our context is a subject for future work. Likewise, if all pixels of images and all points of point clouds were classified as transient or non-transient as a preprocess, the rendering artifacts could be reduced by leveraging this classification. Even further, if each and every pixel and point had a time interval of validity, similar to the lifespan of objects in a spatio-temporal city model, the rendering could make more informed decisions before texturing a city object or a point cloud with a pixel by comparing their valid time intervals.

REFERENCES

Blettery, E., Fernandes, N., Gouet-Brunet, V., 2021. How to Spatialize Geographical Iconographic Heritage. *Proceedings of the 3rd Workshop on Structuring and Understanding of Multimedia heritAge Contents (SUMAC 2021 @ ACM Multimedia 2021)*, Chengdu, China.

Brédif, M., 2013. Image-Based Rendering of LOD1 3D City Models for traffic-augmented Immersive Street-view Navigation. *City Models, Roads and Traffic workshop (CMRT13)*, International Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences (ISPRS Annals), II-3/W3, Antalya, Turkey, 7–11.

Bruschke, J., Niebling, F., Maiwald, F., Friedrichs, K., Wacker, M., Latoschik, M. E., 2017. Towards browsing repositories of spatially oriented historic photographic images in 3d web environments. *Proceedings of the 22nd International Conference on 3D Web Technology - Web3D '17*, ACM Press, 1–6.

Devaux, A., Brédif, M., 2016. Realtime projective multitexturing of pointclouds and meshes for a realistic street-view web navigation. *21st International Conference on Web3D Technology*, Web3D '16 Proceedings of the 21st International Conference on Web3D Technology, ACM Press, Anaheim, United States, 105–108.

Hartley, R., Zisserman, A., 2004. *Multiple View Geometry in Computer Vision*. 2 edn, Cambridge University Press.

Hedman, P., Ritschel, T., Drettakis, G., Brostow, G., 2016. Scalable Inside-Out Image-Based Rendering. *ACM Trans. Graph.*, 35(6), 231:1–231:11.

Mittring, M., Crytek, 2008. Advanced virtual texture topics. *SIGGRAPH '08*.

Paiz-Reyes, E., Brédif, M., Christophe, S., 2021. Cluttering Reduction for Interactive Navigation and Visualization of Historical Images. *ICC 2021 - 30th International Cartographic Conference*, Proc. Int. Cartogr. Assoc., 4, Florence, Italy, 81. short.

Schindler, G., Dellaert, F., 2012. 4D Cities: Analyzing, Visualizing, and Interacting with Historical Urban Photo Collections. *Journal of Multimedia*, 7.

Schütz, M., Mandlburger, G., Otepka, J., Wimmer, M., 2020. Progressive Real-Time Rendering of One Billion Points Without Hierarchical Acceleration Structures. *Computer Graphics Forum*.

Waechter, M., Moehrle, N., Goesele, M., 2014. Let There Be Color! Large-Scale Texturing of 3D Reconstructions. *ECCV (5)*, Zürich, Switzerland, 836–850.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-V-4-2022-213-2022 | © Author(s) 2022. CC BY 4.0 License.