The skyline as a marker for augmented reality in urban context

Mehdi Ayadi, Leo Valque, Mihaela Scuturici, Serge Miguet, Chokri Ben Amar

The skyline as a marker for augmented reality in urban context

Mehdi Ayadi[1,2], Leo Valque[1], Mihaela Scuturici[1],
Serge Miguet[1], and Chokri Ben Amar[2]

[1] University of Lyon, CNRS
University Lyon 2, LIRIS, UMR5205
F-69676, France
[2] University of Sfax, ENIS
REGIM-Lab: REsearch Groups in Intelligent Machines
BP 1173, Sfax, 3038, Tunisia
{Mehdi.Ayadi,Mihaela.Scuturici,Serge.Miguet}@univ-lyon2.fr
leo.valque@ens-lyon.fr
chokri.benamar@ieee.org

Abstract. In recent years, augmented reality (AR) technologies have emerged as powerful tools to help visualize the future impacts of new constructions on cities. Many approaches that use costly sensors and height-end platforms to run AR in real-time have been developed. Little efforts have been made to embed AR on mobile phones. In this paper, we present a novel approach that uses the *Skyline* as a marker in an AR system. This lightweight feature enables real-time matching of virtual and real skyline on smartphones.
We use device's embedded instruments to estimate the user's pose. This approximation is used to insert a synthetic object in the live video stream. This first approach gives a very unrealistic impression of the viewed scene: the inserted objects appear to *hover* and *float* with the user's movements. In order to address this problem, we use the live video camera feed as additional source of information which provides a redundancy to the instruments estimation. We extract the Skyline (a set of pixels that defines the boundary between the building and the sky) as main visual feature. Our proposal is to use these *automatically extracted* points and track them throughout the video sequence, to allow synthetic objects to anchor these visual features, making it possible to simulate a a landscape from multiple viewpoints using a smartphone. We use images of the Lyon city (France) to illustrate our proposal.

Keywords: Mobile augmented reality · Image to geometry registration · Image comparison metric · 3D models · Skyline matching · urban landscape.

1 Introduction

Nowadays, preserving a good visual landscape, in terms of aesthetics, health, safety and build ability is an important criteria to enhance the quality of life in urban environments. With the development of Smart Cities around the world, policy makers and architects want to involve citizens in the process of city construction on an early stage. To increase this environmental awareness and assist users, new tools and methods are required. In this context, augmented reality (AR) is used as a landscape simulation system where 3D models are included in a live video stream. Most of these mobile AR systems are marker-based which limit their scope to small-scale environments and indoor objects. Outdoor systems, in the other hand, rely exclusively on noisy sensor data to calibrate the virtual view with the real one. This kind of approaches are generally inaccurate with a limited user experience. They fail to provide a meaningful augmentation.

In this paper, we present a system for a real-time video stream augmentation on mobile platforms. The proposed system combines many features: large-scale (3D city model), visual (skyline) and contextual (instruments) information to augment the live video with 3D models in the correct location and perspective.
The proposed system allows a user to navigate in the urban environment and visualize the impact of a construction projects on the urban landscape using their smartphones (or tablet). The goal is to precisely insert in the image flow the new building model with the most realistic experience for the user. We propose an hybrid approach where the user's pose is first estimated using the smartphone's embedded instruments and then corrected with the skyline, as explained in section 5. In other words, we cast the image/model matching problem into a curve matching problem which makes it a possible solution for real-time mobile application. Our AR system is running on a mobile platform (iPhone 6) with A8 processor.

The remainder of this paper is structured as follows:
In Section 2, we summarize the related previous works on video/GIS registration followed by literature on skyline matching methods. Next, section 3 gives an overview of the proposed system. Section 4 describes the synthetic image generation process followed by section 5 for our skyline matching methods and image comparison metrics. Results are presented in section 6 followed by the conclusions and future work.

2 Related work

One of the first mobile solutions for AR was *The Touring machine*. Other approaches, in [3] or [1], were introduced to overlay information on the camera image. These approaches are well suited to present illustrative informations but are not directly suited for visualization in urban environment or augmenting scenes with urban objects, as presented in this paper. In this context, Video/GIS registration technique is considered as a solution for large-scale augmented reality applications [2]. The problem consists in aligning pixels from the virtual world (3d city model, LiDAR scans, point clouds, ...) with the real image ones. We are in the case where the problem of image to point cloud registration is casted to an iterative process of image to image registration. To solve this problem, an implicit operation is done: the point cloud is turned into a synthetic picture. The real picture is then registered with the virtual one. In the following we present related works on video/GIS registration and skyline matching methods.

Existing approaches rely especially on 2D-features. The most explored methods in the literature use feature points such as SIFT or SURF points. The feature points are extracted from real and synthetic 2D images to be matched. This implies that the point clouds should have a texture, a color information or a reflectance value. In [4], a reflectance image is generated, from which SURF points are extracted. In [5], the image to image registration step relies on the RGB coloration of the point cloud using ASIFT descriptors. In this paper, we propose also to use a textured 3D city model, from which the rendering is a 3D textured object. However, the matching process is based only the on geometric features defined by the skyline.

For the real / virtual image registration step, [6] uses the Mutual Information between the two images. [7] proposes an improved approach, where a *Gradient Orientation Measure (GOM)* is used to compute the difference of the gradient orientation angle between the synthetic image and the real image. These methods are computationally expensive and memory consuming making them inappropriate for mobile real-time rendering. They also stay impractical for a large point clouds, as proposed in this paper.

[8] tries to improve camera orientations using multiple sensors, cameras and laser scans. This first approximation is then corrected with a skyline matching method based on an ICP algorithm. Other methods rely on specific 2D feature points, namely *the skyline*. This skyline matching step was also explored in the literature. In [9], a generic fisheye camera pointing upward and a 3D urban model were used to generate two an omni-directional real and virtual skylines. A matching step between the two images using a *chamfer*

matching algorithm and a shortest path algorithm gives a precise GPS localization. We introduce a novel image comparison metric respecting the real-time constraint on mobile devices.

In [10], panoramic 3D scans are acquired from which the extracted skyline is encoded as a string. The step of skylines matching is then casted to a string matching problem. [11] uses the overall Skyline feature to estimate yaw angle (camera's viewing direction) from the image while having zero pitch and roll. The panoramic skyline is devided into four joint images from which a Cross-Similarity function is calculated to find the azimuth. An extension of this work in [12] overcomes this limitations with calculating the pitch and toll angle using a vertical vanishing points detection algorithm. More recent work in [13] uses a wide-angle synthetic image generated from 3D geometry model and the its associated image. The skyline matching metric is based on Normalized Mutual Information (NMI) and Histogram or Oriented Gradients (HOG).

As mentioned in [14], a city's skyline is defined as *the unique fingerprint of a city*, and therefore characterizes the uniqueness of urban environments or images. It also represents the overall geometric feature of the image in the real and virtual world. As explained in section 5, we use it in our matching step. Our image/model matching problem is then transformed into a curve matching problem where the two skylines are matched with our similarity metric. This significantly reduce the computation complexity of our algorithm which makes it possible for testing on mobile platforms while respecting the real-time constraint.

3 Proposed system

Our method (Figure 1) takes as input a live video stream from the smartphone's camera and a point cloud of the neighborhood urban scene. We propose a two-step registration method to refine the estimated camera's pose, knowing the camera's intrinsic parameters: first, a live video-stream is acquired with the smartphone's camera from which a *real skyline (1)* is extracted using a skyline extraction algorithm [15]. Then, a camera pose is estimated in world coordinate system with its 6 degrees of freedom due to the combination of smartphone's embedded sensors, as explained in section 4.2 : magnetic compass, gyroscope, accelerometer and barometer. This first camera pose allows us to place a virtual camera in the 3D city model and so generate a synthetic image of what the user should sees at this position. From this virtual image, a *virtual skyline (2)* is extracted. Our skyline matching method, detailed in section 5, matches these *real* and *virtual* skylines and refines the camera's pose.

4 Synthetic image generation

We first generate the synthetic image at an estimated user's pose. The camera pose estimation process is explained in section 4.2. In the following, we detail our synthetic image generation process. We adopt a pinhole camera model, where the camera pose is expressed as a projection matrix P, giving the *2D pixel coordinate in image coordinate system* from the *3D vertex coordinates in the world coordinate system*. This projection matrix can be decomposed into three main components:

$$P = K * R * [I - t]$$

where K is the camera intrinsic parameter matrix ; R is the rotations matrix and t the translations vector.

We do not detail the camera calibration step. We perform it with an *iOS-SDK* and *OpenCV* framework. This calibration step gives us the camera's intrinsic parameters as well as the distortion parameters (3 radial and 2 tangential).

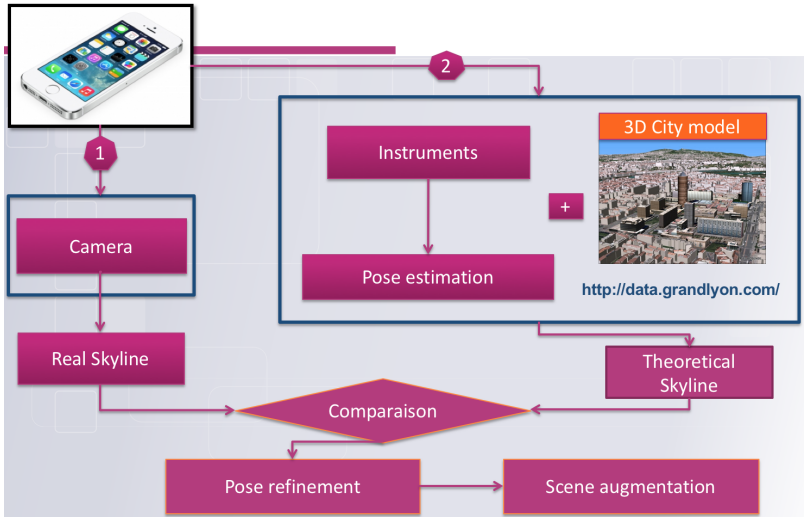


Fig. 1: System

4.1 Dataset

In this subsection, we present the data used in our experiments for the synthetic image generation step. First, we present the 3D city model used in here and associated pre-processing steps. Then our client-server architecture. Finally our image database associated to its meta-data.

3D city model: Many cities around the world own nowadays their virtual double. We mention New York (2016), Brussels (2014) or Lyon (2012) [3]. Producing these 3D geo-referenced data is feasible due to processes based on aerial or terrestrial acquisition campaigns. These data become more and more accurate and particulary open source. These models are available in several formats: 3DS, CityGML, KMZ, DirectX. In our case, we use CityGML format from (OGC). Storing 3D model on mobile device is too heavy, and reading them (XML) is computationally expensive. A preprocessing stage is then bedded.

Data used here is provided by the city of Lyon [4]. It represents about 550 square kilometers in *CityGML*.

Data preparation: Each district of the city is stored in a *CityGML* file. Data is converted and stored as text files. These *CityGML* files will beforehand have been automatically cut into fixed size tiles. All these operations can be batched server-side. In the case of data modification (other cities, districts, etc..), the tiled data may be recomputed easily and automatically. Textures are kept as their *png* original format. To do this, we used an open source tool [5].

We visualize in real-time the 3D city data of the user's neighborhood position. For this, we use a framework based on a mobile client / light server architecture, as described in figure 2. The server receives a request from the mobile device with its GPS position and sends back the corresponding tiled data in JSON format.

[3] https://www.citygml.org/3dcities, 2018
[4] http://data.grandlyon.com, 2018
[5] https://github.com/MEPP-team/3DUSE

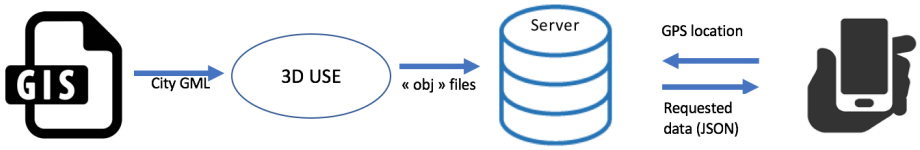


Fig. 2: data exchange pipeline

4.2 Pose estimation

We position our virtual camera in the 3D city model at the exact user's position, viewing direction and orientation.

The user's position (x, y, z) is estimated with a simple rough translation vector:

- (x, y) acquired directly by the smartphone's GPS (longitude, latitude);
- For the altitude (z), depending on smartphone, we retrieve data from:
  - Digital Elevation Model, giving an *absolute measure*;
  - Barometer, for most recent smartphones, giving a *relative altitude*;

Finally, the rotation matrix R is estimated by combining data from both smartphone's triaxial accelerometer and magnetic compass. The output of the accelerometer can be modeled as follows:

$$\overrightarrow{a_S} = \begin{pmatrix} a_{Sx} \\ a_{Sy} \\ a_{Sz} \end{pmatrix} = \overrightarrow{a_B} + R * \overrightarrow{g} \text{ where : } \begin{cases} \overrightarrow{a_S} & \text{Accelerometer output} \\ a_{Sx} & \text{Acc. along x-axis} \\ \overrightarrow{a_B} & \text{Acc. experienced by the moving body} \\ \overrightarrow{g} & \text{Gravity} \\ R & \text{Rotation matrix} \end{cases}$$

All of these vectors are oriented according to the North-East-Down convention. The rotation matrix R allows to project the gravitational vector $\overrightarrow{g}$ from the *terrestrial inertial referential* to the *moving object referential*.

To solve this equation, we must first take a look at the rotation matrices expressions, along x, y and z axis. Since we want to perform the three rotations successively, we must multiply the matrices between them. Or, matrices multiplication is not commutative. We must choose an arbitrary order for these multiplications. Comparing all possible results, we can notice that only R_{xyz} and R_{yxz} are exploitable solutions, allowing us to resolve the equation (depend only on θ and ϕ).

For $R = R_{xyz}$:

$$\overrightarrow{a_S} = R_x(\phi) * R_y(\theta) * R_z(\psi) * \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = ... = \begin{pmatrix} -sin(\theta) \\ sin(\phi)cos(\theta) \\ cos(\phi)cos(\theta) \end{pmatrix}$$

We obtain the following expressions: $\begin{cases} \phi = tan^{-1}(\frac{a_{Sy}}{a_{Sz}}) \\ \theta = tan^{-1}(\frac{-a_{Sx}}{\sqrt{a_{Sy}^2 + a_{Sz}^2}}) \end{cases}$

At this step, we found out the user's orientation along two axis: pitch and yaw. The roll is directly obtained from the magnetic compass, as detailed in figure 3.

We are now able to build the projection model : we place our virtual camera at this previous estimated pose. We illustrate in figures 4 and 5 both real and its corresponding virtual image at same time and location.

5 Skyline matching

Traditional methods use several textural features from the real and virtual images. These methods are quite computationally expensive and heavily depends on the quality of extracted features (SIFT, SURF, etc..), which is not suitable for our mobile AR context. We propose here to exploit scene's geometric features: the *skyline*. In fact, most images taken in urban environments contain a skyline which is defined as the border between sky and non-sky region. The proposed method assumes the existence of one significant sky regions inside the image. In [11], where images are horizontally rectified (zero tilt), we here authorize the six degrees of freedom.

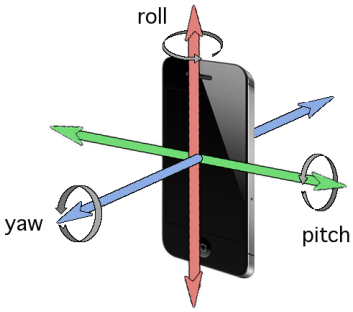


Fig. 3: Smartphone's axis

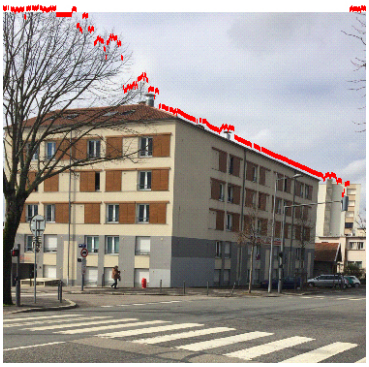


Fig. 4: real image with Skyline

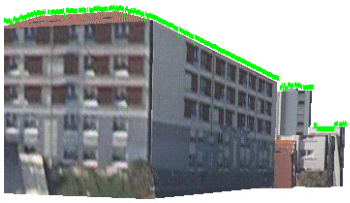


Fig. 5: virtual image withskyline

Both skylines (figures 4 and 5) are assumed to be extracted from the same camera position. In order to, we rely on the similarity of their shapes. This allows us to find the correction that should be applied to the user's pose.

The objective is then to move the virtual skyline in the image plane on the x , y and θ coordinates to match it with the real one. As an approximation, we consider that small pitch and yaw rotations are estimated as translations in the image plane. The problem is to minimize a distance according to the x, y and θ parameters. The minimization algorithm used here is a Gradient descent (simplex) from *OpenCV* optimization modules, *Downhill Solver*. The rendering and the matching steps are real time. We also define an image comparison metric to determine the distance between the skylines. It's clear that, if the camera pose corresponds to the pose that yields the real image, this comparison metric should be minimum. In our case, this metric should be resilient to multiple factors: noise due to errors in skyline extraction process, incomplete data, jagged skyline, too much vegetation, missing data (..).

Our skyline matching pipeline contains three steps : skyline simplification, polygonal approximation and downhill solver.

Skylines systematically contain many straight lines due to noise-related pixels derived from the skyline extraction process or due to their presence in the real world but not in the virtual one (trees, chimney, clouds, ...). Added to that, the original skyline contains a little more than 300 pixels (image's width). Starting the matching step using this vector size would compromise the real-time constraint. A simplification step is then needed to eliminate noise pixels. This step is divided into two steps: outliers removal followed by a *polygonal approximation* step. This reduces the skylines to a few dozens of points that clearly define the

shape of the scene.
Finally, to minimize our objective function, we launch a dynamic programming based on the Nelder-Mead method, namly : Downhill Solver. This allow us to find the global minimum while using a cost function based on one of our comparison metrics.

5.1 Skyline simplification

Algorithm 1 aberrant pixel removal

for $i = 1, i{+}{+}$, while $i < sizeof(input)$ **do**
  if $|input[i+1].y + input[i-1].y - 2 * input[i].y| < precision$ **then**
    $output \leftarrow input[i]$
  end if
end for

where "input" is the original skyline vector, "output" is the resulting vector and "precision" is a parameter reflecting the magnitude of vertical jump between two consecutive pixels.

We illustrate our results in our open source database for skyline extraction and simplification. The skyline is extracted from the real image. Then, outliers are removed based on algorithm 1, obtaining, on average, a skyline vector of 25 pixels. The polygonal approximation step comes after skyline simplification, which approximate the skyline curve with a new one with less vertices such that the distance between them is less than the precision parameter [6]. These pre-treatments (outliers removal + polygonal approximation) allow us to obtain , on average, skylines vectors of around 25 pixels.

5.2 Comparison metric

$\underline{\boldsymbol{L_1}:}$ We try to minimize a distance with the parameter x, y and θ. The first idea explored is to use distance L_1.

$$L_1(A,B) = \int_x \mid y_A(x) - y_B(x) \mid dx \tag{1}$$

This distance is simply the area between the two curves. This solution is also used as a cross-similarity function in [12] and works well for skylines that are very similar. It already provides quite satisfactory results. But the two input skylines often contain important differences:

- lack of precision of the 3D model (building are sometimes reduced to their simplest shape);
- some recent buildings are not yet represented in the 3D model;
- absence of street lights, lamp posts, electrical panels, power lines and other artifacts in the 3D model

When the real and theoretical (extracted from the 3D model) skylines present these differences, the matching step goes wrong. For this, we introduce a second comparison metric.

$\underline{\boldsymbol{L_{-1}}}$: The objective is to find a way to manage these differences while having the following features:

[6] http://en.wikipedia.org/wiki/Ramer-DouglasPeuckerAlgorithm

- close curves are strongly valued;
- distant curves are not too penalized: the curves must to be able to deviate in one point if it allows to be identical in others;
- distant lines and very distant ones have the same importance.

Among the tested metrics, one respects these criteria well. We define it as:

$$L_{-1}(A,B) = M - \int_x \frac{1}{\mid y_A(x) - y_B(x) \mid + c} dx \tag{2}$$

We denote A and B the real and virtual skylines. The constant "c" avoids divergences and allows the behavior to be adjusted: the smaller it will be, the more it will apply strongly the criteria at the risk of creating a lot of local minimums affecting the convergence of the minimization algorithms. On the contrary, the bigger it will be, the closer we will get to the behavior of the first metric 1.

This metric follows well the criteria defined above and is much better robust to the cases where skylines present important differences.

However, it fails when matching vertical lines and is far from *good matches* for some entries, including fairly simple entries where our first metric find better solutions. This is not quite the expected result and it is therefore necessary to adjust this metric to be able to use it.

$\boldsymbol{L_{final}}$ **:** The two previous metrics are combined to get advantages of each one: the first one works well for simple cases but not for complicated ones. The second works better for complicated cases but not for all simple. For this, we have to combine these two metrics. The typical way to combine two functions in an objective one is to use a weighted linear combination of the two, while adding a parameter $alpha$, e.g L_{final} = $alpha$ * L_1 + (1 - $alpha$) * L_{-1}. For this, we must make one of the following hypothesis :

- It exists more simple cases than complicated ones, and then, $alpha$ > 0.5 ;
- In contrary, more complicated than simple cases, and then, $alpha$ < 0.5.

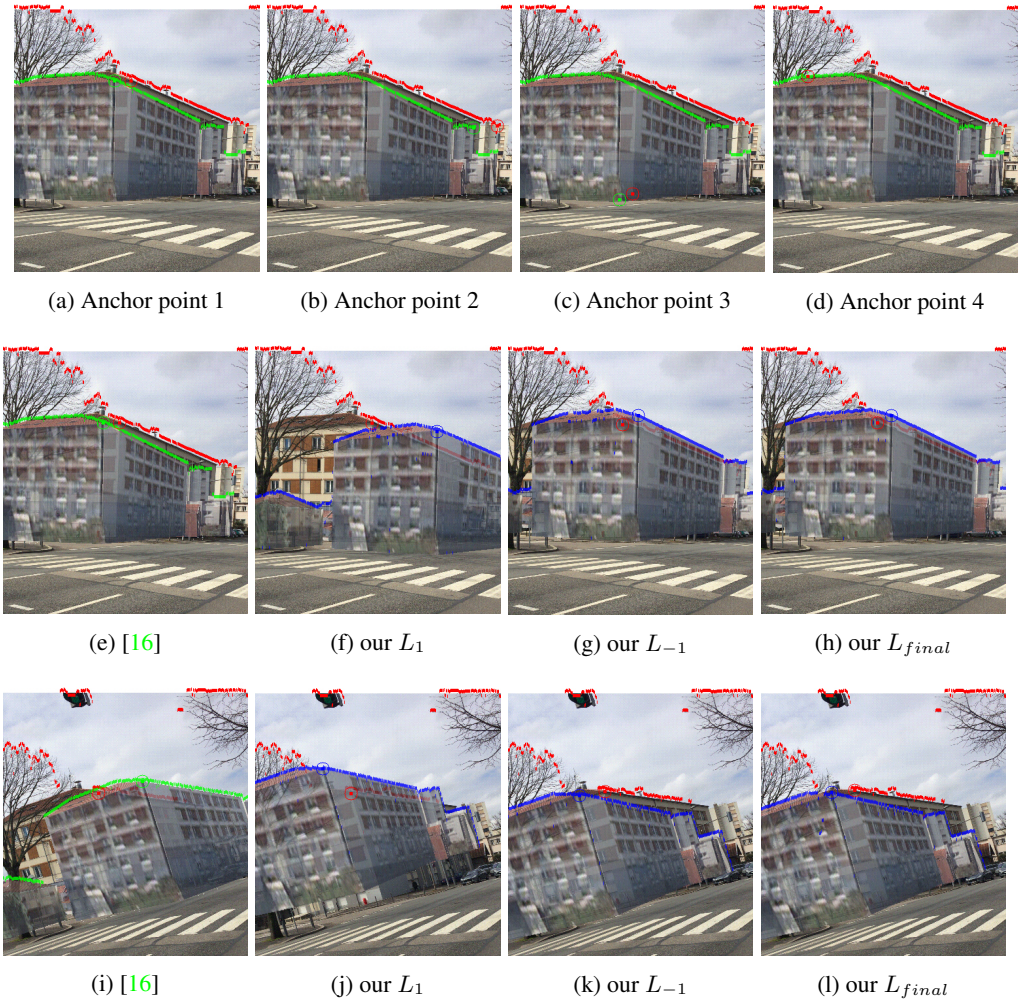
Due to the variability of the scenes (simple or complicated cases), we chose to not add such weighted combination and abstract our final metric of any hypothesis. Our final metric is then:

$$L_{final}(A,B) = L_1(A,B) * L_{-1}(A,B) \tag{3}$$

6 Results and discussion

In order to test the accuracy and robustness of our method, we conducted several experiments. Tests have been made on real 3D data provided by the city of Lyon (France) covering more than 500 km^2 of territory. Tests were made in different weather conditions and multiple types of areas. We take into account objects that can be found in real environment but not in 3D city model: along urban areas, mainly trees form the skyline of the foreground objects as can be found in our open source database images. Added to that, buildings and road furniture such as traffic lights, lamp posts, or street signs lead to wrong real skylines, that cannot be later matched with virtual one.

We conducted our experiments, and constructed a database of around 300 images. Global results are given in table 1 and examples are illustrated in figure 6. All image database are avaliable in [17]. For each image, we define multiple anchor points for residual error calculation, as the difference along the horizontal and vertical directions of those points. In figures 6a to 6d, we define these anchor points, that are essential in our evaluation protocol, and that have to be non-coplanar. For each image, the residual error is denoted



(a) Anchor point 1 (b) Anchor point 2 (c) Anchor point 3 (d) Anchor point 4

(e) [16] (f) our L_1 (g) our L_{-1} (h) our L_{final}

(i) [16] (j) our L_1 (k) our L_{-1} (l) our L_{final}

Fig. 6: Residual error calculation

$(\Delta x, \Delta y)$, in terms of the number of pixels, and is calculated as the average of all anchor points pixel error. Then, for each image with each of proposed comparison metrics, we choose, when possible, to use the same anchor points in the evaluation protocol, as illustrated in figures 6e to 6h. Then, as discussed previously, we authorize the six-degree of freedom. Figures 6i and 6l illustrate the same scene with two different perspectives (tilt angle). We notice in figures 6e to 6h, how this complicated case (presence of a tree on the left of the image) gives different results. With [16], error is acceptable ($(\Delta X, \Delta Y) = (28, 3)$). With our L_1 metric, we obtain very bad results ($(\Delta X, \Delta Y) = (99, 13)$). Then L_{-1} metric gives a little better result ($(\Delta X, \Delta Y) = (24, 14)$) and finally L_{-1} metric the best one ($(\Delta X, \Delta Y) = (4, 13)$). We compare our method to [16] with our different metrics. We calculate, for each image, the residual error. These results verify the high accuracy of our system for near and far objects.
Then, as a human-machine comparison step, we manually match the virtual skyline with the real one, and then generate a manually generated alignments. As the automatic pose estimation procedure, this step is also based only on translation in the image plane. We add rotations in the virtual environment and sees in real-time the virtual skyline moving in the image place. We denote this database: *manually alignment*. We compare our results to this manually alignments using the same residual error calculation method.

The developed system run on a smartphone with standard specifications (iPhone 6), including an Apple iOS 11.4 operating system and Open GL-ES 3.0. In table 1, we show results of our system : we compare our result (virtual images) to the real ones, then to the the manual generated alignments. We compare our results also to experiment 9 of [16]. We give in the table result for images shown in 6. The system runs in real-time on iPhone 6. Depending on the complexity of the rendered scene (number of buildings, complexity of the buildings, etc.), on average, the rendering process takes 12 ms and the matching step 11 ms.

image			comparison to real images				comparison to manual alignments		
			[16]	L_1	L_{-1}	L_{final}	L_1	L_{-1}	L_{final}
frame 58 : imgs 6e to 6h	Δ X		28,5	-98	15	-20	-106	-29	-30
	Δ Y		-3	-9	6	8	-10	9	8
frame 62 : imgs 6i to 6l	Δ X		73	56	32	49	45	41	46
	Δ Y		11	30	-12	-7	30	-8	-6
database	Δ X	Absolute average	10	3	10	3	4	9	2
		Std Dev	70	82	66	66	92	61	59
	Δ Y	Absolute average	15	20	14	14	24	17	18
		Std Dev	18	36	28	28	38	32	30

Table 1: residual error results

7 Conclusion and future work

We investigated the possibility of using skyline features as a marker for augmented reality. We proposed a system where the estimated pose with smartphone's instruments is corrected using the skyline. Our skyline matching method allows us to transform the image to model matching problem into a curve matching one. This reduced significantly the computation complexity (on an iPhone 6), allowing us to test our system in real-time video. The accuracy and robustness was tested on a set of 300 images, with synthetic and

real images. Our different comparison metrics gives promising results for simple and complicated cases. However, results suffers from cases, where occlusions are present (tall trees, etc..). On the other hand, our matching process here is in the image plane. We will try to add a matching process with six degree of freedom on the world coordinate referential.

References

1. Dieter Schmalstieg, Tobias Langlotz, and Mark Billinghurst. *Augmented Reality 2.0*, pages 13–37. Springer Vienna, Vienna, 2011.
2. Tobias Langlotz, Stefan Mooslechner, Stefanie Zollmann, Claus Degendorfer, Gerhard Reitmayr, and Dieter Schmalstieg. Sketching up the world: in situ authoring for mobile augmented reality. *Personal and Ubiquitous Computing*, 16(6):623–630, Aug 2012.
3. J. Benjamin Gotow, Krzysztof Zienkiewicz, Jules White, and Douglas C. Schmidt. Addressing challenges with augmented reality applications on smartphones. In Ying Cai, Thomas Magedanz, Minglu Li, Jinchun Xia, and Carlo Giannelli, editors, *Mobile Wireless Middleware, Operating Systems, and Applications*, pages 129–143, Berlin, Heidelberg, 2010.
4. M Wiggenhagen. Co-registration of terrestrial laser scans and close range digital images using scale invariant features. *Plastverarbeiter.De*, pages 208–212, 2010.
5. W. Moussa, M. Abdel-Wahab, and D. Fritsch. An automatic procedure for combining digital images and laser scanner data. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXIX-B5:229–234, 2012.
6. Zachary Taylor and Juan Nieto. Automatic calibration of lidar and camera images using normalized mutual information. page 8, 2012.
7. Z. Taylor, J. Nieto, and D. Johnson. Automatic calibration of multi-modal sensor systems using a gradient orientation measure. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1293–1300, Nov 2013.
8. S. Hofmann, D. Eggert, and C. Brenner. Skyline matching based camera orientation from images and mobile mapping point clouds. II-5:181–188, 2014.
9. S. Ramalingam, S. Bouaziz, P. Sturm, and M. Brand. Skyline2gps: Localization in urban canyons using omni-skylines. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3816–3823, Oct 2010.
10. Andreas Nüchter, Stanislav Gutev, Dorit Borrmann, and Jan Elseberg. Skyline-based registration of 3d laser scans. *Geo-spatial Information Science*, 14(2):85, May 2011.
11. Shupeng Zhu, Muriel Pressigout, Myriam Servières, Luce Morin, and Guillaume Moreau. Skyline Matching: A robust registration method between Video and GIS. In *Conference of the European COST Action TU0801 - Semantic Enrichment of 3D City Models for Sustainable Urban Development Location: Grad Sch Architecture*, pages 1–6, Nantes, France, October 2012.
12. S. Zhu, L. Morin, M. Pressigout, G. Moreau, and M. Servières. Video/gis registration system based on skyline matching method. In *2013 IEEE International Conference on Image Processing*, pages 3632–3636, Sept 2013.
13. Maximilien Guislain, Julie Digne, Raphaëlle Chaine, and Gilles Monnier. Fine scale image registration in large-scale urban lidar point sets. *Computer Vision and Image Understanding*, 157:90 – 102, 2017. Large-Scale 3D Modeling of Urban Indoor or Outdoor Scenes from Images and Range Scans.
14. Nurulhuda Abdul Hamid Yusoff, Anuar Mohd Noor, and Rosmadi Ghazali. City skyline conservation: Sustaining the premier image of kuala lumpur. 20:583–592, 2014.
15. Mehdi Ayadi, Loreta Suta, Mihaela Scuturici, Serge Miguet, and Chokri Ben Amar. A parametric algorithm for skyline extraction. In Jacques Blanc-Talon, Cosimo Distante, Wilfried Philips, Dan Popescu, and Paul Scheunders, editors, *Advanced Concepts for Intelligent Vision Systems*, pages 604–615, Cham, 2016. Springer International Publishing.
16. Fukuda2014 Fukuda, T., Zhang, T., Yabuki, N. : Improvement of registration accuracy of a handheld augmented reality system for urban landscape simulation. In Frontiers of Architectural Research, pages 386–397, 2014
17. Skyline database : $dionysos.univ - lyon2.fr/ \sim mayadi/ISVC'18/skylineDatabase$