# Interactive Scene Navigation using the GPU

Shanat Kolhatkar
School of Information Technology and Engineering (SITE)
800 King Edward
Ottawa, Ontario, Canada, K1N 6N5
Email: shanat.kolhatkar@gmail.com

Robert Laganière
School of Information Technology and Engineering (SITE)
800 King Edward, P.O. Box 450, Stn A
Ottawa, Ontario, Canada, K1N 6N5
Email: laganier@site.uottawa.ca

***Abstract*—In this paper we present a component of the NAVIRE Viewer, which is an application to allow a user to virtually navigate in a remote scene in real-time from a collection of pre-captured panoramic views. The work presented in this paper concerns viewpoint interpolation which is essential to achieve smooth and realistic viewpoint transition while the user is moving from one panorama to another. Our viewer achieves real-time interpolation between panoramas by using a precalculate flow field between a pair of images, and by applying a view morphing algorithm to generate virtual viewpoints in-between existing views. A GPU implementation of the viewpoint interpolation algorithm has been developped allowing to generate virtual viewpoints at a rate of up to 300 panoramas per seconds.**

## I. Introduction

This work was developped within the scope of the NAVIRE project (virtual NAVigation in Image-based representations of Real world Environment). The objective is to achieve real-time navigation in image-based renditions of real environments. The project covers all aspects of the image-based environment representation and navigation problem. There are multiple aspects to this project such as the acquisition and storage of high quality panoramas, the estimation of the pose between panoramas, the interpolation of views and the rendering and navigation tools to name a few. Our goal is to provide the user with an intuitive software to navigate any given environment in real-time, and provide the highest degree of realism and immersion possible.

Image-based modeling and rendering methods [2] has attracted a lot of attention from both the computer vision and the computer graphics communities, and in recent years, we have seen the appearance of applications using collections of images in order to allow users to remotely explore a given existing environment. Such systems include the Google Street View system for city landsapes and the Furukawa et al. [13] for indoor spaces.

A multitude of improvements can be added to these applications, aimed especially at enhancing the quality of the immersive experience. One can, for instance, improve the quality of the viewing conditions by increasing the quality of the images by using very large display (e.g. CAVE) or head mounted device if available. In most image-based navigation applications, the movement of the user is limited to a linear path. This could be improved by allowing the user to move freely inside of the environment; this would require the availability of a multitude of points of view distributed over the environment. Also, when virtually visiting a scene, the user has to hop from one panorama to another; these unnatural saccadic displacements often confuse the user who may lose his orientation. This could be alleviated by generating realistic transitions between panoramas to let the user see the path taken. However this is a difficult task because most of the time the 3D geometry of the scene is not known and the movement generated between images depends on the distance between the view points and scene elements.

In this paper we interpolate views between $360^\circ$ panoramic images to obtain realistic virtual views and use an optical flow algorithm to produce realistic view morphing during the transitions. An important feature of navigation softwares is interactivity, and thus we have designed our algorithm to achieve the view interpolation process at frame rate. Our method allows users to realistically transit from one panorama to any adjacent one. The scene is assumed to be static (i.e. does not contain moving objects) and the pre-captured panoramas are taken at a relatively short distance from each other. No 3D information about the scene or the motion is used.

Section 2 presents the previous work relevant to our method. Section 3 presents an overview of the method used to achieve the interpolation of view as well as the necessary elements to achieve real-time navigation. Section 4 presents some results and Section 5 is a conclusion.

## II. Related Work

There are a few different approaches to texture morphing that have been developed over the last few years. The work in [9] depends heavily on user input, and only works for textures that are composed of repeated similar patterns (for example cells of the interior of a bee hive). The method presented in [3] represents, to the authors' knowledge, the state of the art in the field of texture morphing, and combines linear color interpolation and motion compensation to generate the composite texture. This paper is at the basis of our interpolation algorithm. It has the advantages of not depending much on user input and of working with a wide variety of textures while still producing high quality results. However, this method needs to have features of similar sizes in both origin and target textures. Also, this method cannot morph smoothly between highly different textures.

One of the fundamental paper in the field of view interpolation is [5]. This paper focuses on creating the transition views to move between pairs of images, which are not necessarily parallel. To create the transition images, the method prewarps two input images to fit them on parallel planes. It then applies a morphing procedure to obtain an intermediate image and postwarps that intermediate image to obtain the resulting transition image. Both prewarping and postwarping transformations are done using projective transformations. A downpoint of this approach is that it cannot handle complex scenes. It was designed to handle pairs of images of single objects taken in different poses. This makes this approach difficult to generalize to arbitrary viewpoints. Other papers regarding view morphing also attack the subject of view interpolation, such as [6], [8]. Both approaches triangulate the images using a certain set of points in the images, which can be chosen manually or automatically using for example [1]. Once the triangulation is done, the triangles of the images are warped and the colors are interpolated to get a transition image. Another paper related to the topic of view interpolation is [7], which uses a ray-tracing approach to synthesize new views from an input pair of cubic panoramas. A downside of this approach is that it is time consuming and the resulting interpolated images present many artifacts.

In our approach, fluid view transitions are generated on the fly for the whole panorama. The addition of new panoramas requires only the computation of new optical flow fields. Many approaches to evaluate the optical flow between pairs of images have been proposed. Some classic methods are [10], [11], [12]. Another more recent approach to calculating the optical flow is [4], in which all possible combinations of displacement values are tested by the algorithm in order to find the best displacement vector for each pixel. For each of these shifts the algorithm computes a goodness fonction for each pixel of the image. This algorithm also handles slanted surfaces and uses a contrast-invariant matching measure which is perfect for real environments where building facades are frequently observed and where the lighting condition can be different between viewpoints.

## III. Approach to Interpolating Panoramas in Real-Time

In this section we will describe the approach that was taken to achieve the navigation of an environment in real-time. We will be using the optical flow algorithm [4] for which an implementation is publicly available[1].

Our approach would work on standard imagery, but to enhance the sense of immersion of the user and the realism of the scene rendering, we have chosen to use $360°$ panoramas. Different representations are available to manipulate the panoramas: cubic, spherical or cylindrical panoramas; in our application we have chosen to use the cubic representation. (Figure 1) The main advantage of the cubic representation is that it allows to decompose the global $360°$ panorama into 6 regular limited field-of-view cameras, where each face is a planar image. Since classical optical flow estimation methods were designed for this type of images, these can be applied on each face of the cubic representation. However each face cannot be considered independently from the others because pixels at the border of a face will end up on an adjacent face. To solve this problem, we create an extended cubic panorama representation, which simply extend each face in every direction. Displacements going from one face to another can then be captured while considering each face independently during the optical flow calculation. Note that, when proceeding this way, it happens that some displacement vectors are computed twice, because they appear on each extended portion of two adjacent faces To ensure consistency of the results across the faces, we need to check which one gives the best result by comparing the color neighborhood in both images according to the L2-norm. The one with the greatest distance is then replaced by the other one.

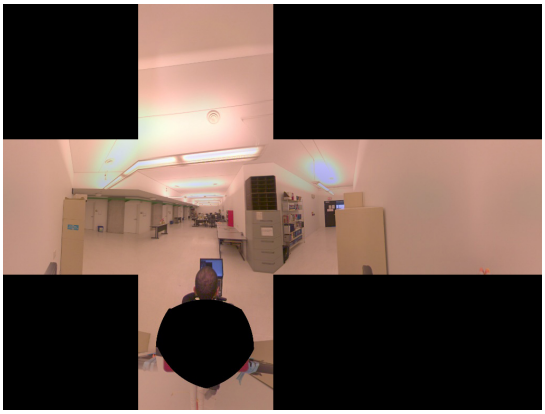

Fig. 1. Example of a cubic panoramas, displayed in an unfolded form

### A. View Interpolation

Our view interpolation is based on the algorithm developed in [3] which we adapted to cubic panoramas of real environments. The computed optical flow algorithms can fairly accurately evaluate the dispacement between images. Our algorithm works on a per face basis. Our panoramas are taken close to each other, in a given city/environment. To generate transition viewpoints, the straightforward approach would be to use linear blending. However, this blending, even though easy to implement, produces blurry images with a lot of artifacts, especially because this approach doesn't take into consideration the geometry of the scene. We create more realistic transitions by including the dense displacement maps into our morphing algorithm. Following the approach in [3], the morphing between pairs of images is given by:

$$\hat{I}(x,y) = (1-w)I_0(wW_{01}(x,y)^{-1}) + wI_1((1-w)W_{10}(x,y)^{-1}) \quad (1)$$

with the following definition:

- $I_0$, $I_1$ and $\hat{I}$ are the origin, destination and transition images respectively.

[1] http://www.cs.umd.edu/ ogale/download/code.html

- $W_{ij}$ is the dense displacement field from image $I_i$ to $I_j$. According to the optical flow algorithm that we use, $W_{ij}(x, y) = -W_{ji}(x, y)$.
- $w$ is the weight applied to the color and the displacement.

### B. Real-Time Implementation

Real-time navigation is based on four elements: preprocessing of the optical flow, data buffering, multi-threading and the implementation of the interpolation algorithm on the GPU.

The computation of the optical flow is the most time consuming step but as it can be precalculated, its estimation time does not affect the rendering performance. During navigation, the reference images and the corresponding computed optical flow field are loaded into memory.

During the navigation of a scene, hard drive accesses might occur hundreds of time. Thus we need to access this data smoothly and continuously. To do so, our application needs to be multi-threaded: we will need to have at least one thread to continuously load the data and another to deal with the display of our scene and the interpolation. Next this data needs to be stored in a small buffer, which will contain the panoramas that are the most likely to be accessed next by our application. Thinking of our scene as a graph where the panoramas are nodes and the possible movement are represented as edges, we determine which panorama to put in the buffer using a breath first approach.

The GPU implementation is the last part of our processing chain. Since we have already precalculated the flow, the interpolation of each pixel value is independent from the rest of the image which makes it a perfect candidate for parallel implementation on the GPU. The GPUs on the other hand are much faster than CPUs and are designed to handle intensive graphics calculations, and allows us to do these calculations while the CPU is handling other kinds of operations (graphics calls and hard drive calls in our case). In addition to the panoramas and the displacement fields needed by the interpolation, we also need to pass the optical flow window boundaries that we used in the optical flow calculation algorithm as well as the texture sizes in order to be able to calculate the origin and target pixel coordinates of each of the interpolated image pixels directly on the GPU accurately.

## IV. Results

We captured our scenes using a Ladybug camera, which can capture 6 images of resolution 1024x768 to create panoramic images in a single shot (5 on the sides, and 1 at the top). We are using extended cube faces of size 320x320 pixels each, each extended cube created in about 3 minutes. The precalculation and correction of the optical flow and correction took up to an hour on a 320x320 image with a [-40; 40] interval on both x and y, on an Athlon 64 X2 6000+/3GHz. We were able to synthesize 20 images per second on the CPU using our interpolation algorithm, and went up to synthesizing 333 images per second on a Radeon Mobility X700 GPU.

According to our results, the quality of the interpolated images' geometry is very similar to the corresponding reference panorama with few artifacts visible. Comparing our interpolated images with few real imags captured at the same location with obtained a RMS error of 15.2 in average. These results show that our interpolated images are much similar to the actual image that the linearly interpolated image.

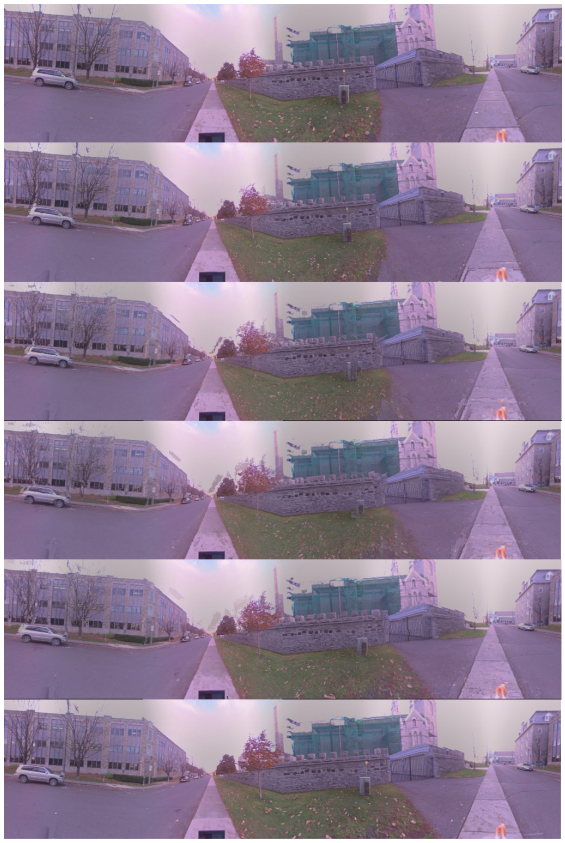

Fig. 2. An outside sequence. Top and bottom images are the captured origin and destination. The other images are (from top to bottom) are interpolated images with $w$=0.2, 0.4, 0.6 and 0.8.

One important thing to remember is that since this interpolation is built for real-time naviguation and the user will not be stopping on an interpolation image, such that many artifacts will go unnoticed. Figure 2 presents some of our results on a pair of panoramas taken outdoors. Even though some artifacts are noticeable, the quality of the results are very realistic.

## V. Conclusion and Future Work

This paper presented a new way of interactively navigating an environment by computing the transition images on the GPU. Our contributions concern especially the ability to interpolate intermediate viewpoints of a scene in real-time. Using panoramas that were taken at reasonable distances from each other, we achieved good quality results, except for a few artifacts.

Some future works possibilities include improving the quality of the dense flow field and enhancing its evaluation speed. It would also be interesting to increase the flexibility and accessibility of our approach by generating panoramas from images acquired from any kind of handheld camera.

## ACKNOWLEDGMENT

This work is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under the NAVIRE Strategic Project Grant.

## REFERENCES

[1] J. Shi and C. Tomasi, Good features to track., CVPR (1994)
[2] H-Y Shum, SB Kang, A Review of Image-based Rendering Techniques, Proceedings of IEEE/SPIE, 4067, 11 (2000)
[3] Wojciech Matusik and Matthias Zwicker and Fredo Durant, Texture Design Using a Simplicial Complex of Morphable Textures, ACM Transactions on Graphics , 24, 787 - 794 (2005)
[4] A. S. Ogale and Y. Aloimonos, A roadmap to the integration of early visual modules, International Journal of Computer Vision, 72, 9–25 (2007)
[5] Steven M. Seitz and Charles R. Dyer, View Morphing, SIGGRAPH (1996)
[6] Maxime Lhuilliier and Long Quan, Image Interpolation by Joint View Triangulation, CVPR (1999)
[7] Feng Shi and Robert Laganiere and Eric Dubois and Frederic Labrosse, On the use of Ray-tracing for Viewpoint Interpolation in Panoramic Imagery, CRV (2009)
[8] Xiaoyong Sun and Eric Dubois, View morphing and interpolation through triangulation, IVCP(2005)
[9] Ziqiang Liu and Ce Liu and Heung-Yeung Shum and Yizhou Yu, Pattern-based Texture Metamorphosis, PG'02: Proceedings of the 10th Pacific Conference on Computer Graphics and Applications, 184–191 (2002)
[10] Berthold K.P. Horn and Brian G. Schunck, Determining Optical Flow, Artificial Intelligence, 17, 185–203 (1981)
[11] Lucas, B., and Kanade, T., An Iterative Image Registration Technique with an Application to Stereo Vision, Proc. of 7th International Joint Conference on Artificial Intelligence, 674–679 (1981)
[12] Jean-Yves Bouguet, Pyramidal Implementation of the Lucas Kanade Feature Tracker (2002)
[13] Yasutaka Furukawa, Brian Curless, Steven M. Seitz and Richard Szeliski, Reconstructing Building Interiors from Images, ICCV (2009)