Standalone Edge-Based Markerless Tracking of Fully 3-Dimensional Objects for Handheld Augmented Reality

João P. Lima[1*] Veronica Teichrieb[1†] Judith Kelner[1‡] Robert W. Lindeman[2§]
[1]Virtual Reality and Multimedia Research Group
Informatics Center - Federal University of Pernambuco, Brazil
[2]Department of Computer Science
Worcester Polytechnic Institute, USA

Abstract

This paper presents a markerless tracking technique targeted to the Windows Mobile Pocket PC platform. The primary aim of this work is to allow the development of standalone augmented reality applications for handheld devices based on natural feature tracking of fully 3-Dimensional objects. In order to achieve this goal, a model-based tracking approach that relies on edge information was adopted. Since it does not require high processing power, it is suitable for constrained devices such as handhelds. The OpenGL ES graphics library was used to detect the visible edges in a given frame, taking advantage of graphics hardware acceleration when available. In addition, a subset of two computer vision libraries was ported to the Pocket PC platform in order to provide some required algorithms to the markerless mobile solution. They were also adapted to use fixed-point math, with the purpose of improving the overall performance of the routines. The port of these libraries opens up the possibility of having other computer-vision tasks being executed on mobile platforms. An augmented reality application was created using the implemented technique and evaluations were done regarding tracking performance, accuracy and robustness. In most of the tests, the frame rates obtained are suitable for handheld augmented reality and a reasonable estimation of the object pose was provided.

CR Categories: I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Tracking H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, Augmented, and Virtual Realities;

Keywords: markerless tracking, augmented reality, computer vision, handheld, mobile

1 Introduction

Two topics have been gaining more attention from Augmented Reality (AR) researchers over the last few years: handheld-device support and markerless tracking. Mobility and compactness requirements of some application domains favor AR projects that focus on handheld platforms. In addition, the presence of markers in an environment can be considered intrusive, justifying the need for natural-feature tracking.

This paper proposes a standalone markerless AR solution based on tracking of fully 3-Dimensional objects, running on the Microsoft Windows Mobile Pocket PC handheld platform, which promotes the development of fully mobile AR applications. An edge-based markerless tracking algorithm [Wuest et al. 2005] was adapted to run under the constrained processing and graphics capabilities of a handheld device. Some modifications were performed in the visible-edge-detection and control-points-matching steps, and all math operations were converted to fixed-point math. As a byproduct, portions of the Vision-something-Libraries (VXL) [VXL 2008] and the Visual Servoing Platform (ViSP) [Marchand et al. 2005] were ported to the Pocket PC platform, providing infrastructure for the development of other computer-vision solutions targeted at mobile devices. A method for the edge visibility checking step, which makes use of OpenGL ES, was implemented. It consists of projecting all edges using hidden line removal, and associating one color to every model edge. After that, the algorithm looks for sampled points in the image and compares if the color is the same as its edge to decide if that point is visible. It is suitable for the reduced functionality of the given mobile graphics library, and can also exploit the graphics hardware acceleration available on some handheld devices.

There have been some attempts to perform markerless tracking on mobile devices. Mozzies is a First Person Shooting (FPS) game where camera movement is detected based on the captured image in order to allow the player to aim at the enemies [Siemens 2008]. Kick Real consists of a penalty shootout competition where the player kicks the virtual ball with his own foot [Reimann 2005]. However, the previously mentioned games only perform 2D tracking of the environment. Virtual Video uses a mobile phone equipped with a Global Positioning System (GPS), accelerometers and a magnetometer to determine the position and orientation of the camera in relation to the real world [Kahari and Murphy 2006]. Nevertheless, the tracking is not very accurate and provides only approximations of object locations. AR-PDA [Gausemeier et al. 2003] and AR Phone [Woodward 2006] are able to perform precise 3D natural feature tracking. However, they are distributed, since all computer vision processing is done by a server. The ULTRA system performs autonomous 3D markerless calibration, although not in real-time [Riess and Stricker 2006]. Wagner et al. [2008] developed a standalone natural feature tracking solution for mobile phones that runs at interactive frame rates, but it is only able to detect planar objects. The solution proposed in this work is capable of performing markerless tracking of fully 3-Dimensional objects in a standalone manner, using a handheld device.

This paper is organized as follows. Section 2 details the edge-based tracking algorithm pipeline used as the basis for this work. Section 3 explains the steps needed to implement the markerless-tracking feature on the Pocket PC platform. The results obtained and evaluations performed are presented in Section 4. Section 5 draws some final considerations and discusses possible future work.

2 Edge Based Tracking

The work described in this paper implements a variation of the Wuest et al. [2005] single-hypothesis, edge-based tracking tech-

*e-mail: jpsml@cin.ufpe.br
†e-mail: vt@cin.ufpe.br
‡e-mail: jk@cin.ufpe.br
§e-mail: gogo@wpi.edu

VRST 2009, Kyoto, Japan, November 18 – 20, 2009.
© 2009 ACM 978-1-60558-869-8/09/0011 $10.00

nique . This technique was chosen because it utilizes consolidated methods in the markerless tracking area and also due to its low CPU load, making it suitable for execution on mobile devices in an autonomous way.

The first step of edge-based tracking is to determine what the visible parts of the edges are when the 3D model of the object is projected onto the image plane using previous pose information. The visibility checking used by Wuest et al. could not be used on the handheld platform, since it makes use of an OpenGL extension that is not available for mobile devices. Therefore, a different approach was developed, as further explained in Section 3.2.

A sampling of the projected visible edges is done, obtaining control points. This is performed in image space in order to produce evenly spaced points. The number of sampled points per edge n is calculated using the following formula:

$$n = \text{edge length}/\text{sampling step}. \quad (1)$$

The sampling step value, which determines the density of control points, was empirically set to 10 pixels in the current implementation.

Next, for each control point, a corresponding point in the image gradient is determined. This search is done in a perpendicular direction relative to the edge. Differently from Wuest et al., who use a simple filter mask, the Moving Edges (ME) algorithm was utilized for finding the correspondences, which is a more refined way of retrieving the edge matches [Bouthemy 1989]. Since the ME algorithm works with gray scale images, the input colored image must first be converted. When the correspondences between points and edges are known, the Levenberg-Marquardt method is used to calculate the pose minimizing the reprojection error, defined as:

$$err = \sum_i |(q_i - p_i) \cdot (n_i)|, \quad (2)$$

where p_i is the projected control point, q_i is the corresponding point in the image and n_i is the normal of the projected edge. The pose is parameterized using six variables: three for rotation and three for translation. Instead of considering all the nine elements of a 3x3 matrix, the rotation is represented using an exponential map [Lepetit and Fua 2005]. This reduces the number of variables to be minimized and avoids the introduction of unnecessary constraints to ensure that the rotation matrix is orthonormal. In the exponential map formulation, the three parameters define a vector, which represents the rotation axis. The vector magnitude is the rotation angle around the axis.

3 Handheld Implementation

In order to run the edge-based tracking solution on a Pocket PC, existing computer vision libraries that implement some required features needed to be available on the platform. Therefore, part of the VXL and ViSP libraries were ported to the Pocket PC. VXL provided all the required math support, including the Levenberg-Marquardt method. ViSP contains an implementation of the ME algorithm. After that, most of the math code was modified to use fixed point, aiming for performance improvements. Finally, edge visibility was handled using the OpenGL ES graphics library. The next subsections describe how the fixed-point math and visible-edge detection steps were accomplished.

3.1 Fixed-Point Math

Since the target processor does not have a FPU, all floating-point operations are emulated in software, which incurs a significant performance penalty. Therefore, instead of using floating-point types to perform real-number calculations, a fixed-point type had to be utilized. In order to accomplish this, a type for real numbers was created. Depending on the platform being used, it is mapped to a fixed- or floating-point type. In addition, the code of the supporting libraries and the application was modified to use the real type instead of the floating-point type directly.

The fixed-point type provides an implementation for basic operations, comparison, absolute value, square root, rounding, and trigonometric functions. The 64-bit fixed-point math format initially used was S47.16 (1 bit for sign, 47 bits for the integer part, and 16 bits for the decimal part). A large number of bits was used to represent the integer part in order to avoid overflows.

One of the main drawbacks of using a fixed-point representation is the lack of precision in the decimal part. As a consequence, the fixed-point version of the Levenberg-Marquardt algorithm needed a larger number of iterations to converge than its floating-point counterpart, resulting in a performance decrease. In addition, the quality of the minimization also decreased. Therefore, the number of bits used to represent the decimal part was raised, and the S40.23 format was adopted. As a result, it avoids overflows and the numeric optimization routines' outputs were satisfactory.

3.2 Visible-Edge Detection

The approach adopted by Wuest et al. for determining the visible parts of the edges at a given frame makes use of the OpenGL extension GL_OCCLUSION_TEST_HP. However, neither this extension nor the equivalent standard extension GL_ARB_OCCLUSION_QUERY is available for OpenGL ES. In addition, reading from the depth buffer is not allowed by the mobile graphics library.

An alternative method was developed to perform visibility testing on the handheld platform. It is inspired by the Facet-ID method [Vacchetti et al. 2004]. Since its goal is to identify edges, it is called the Edge-ID method. In Facet-ID, the index of each polygon is encoded in its color value, and after the model is rendered, it is possible to discover the facet that generated a given pixel when projected. Edge-ID exploits the same idea for edges, but for a different purpose: while Facet-ID is used for finding the 3D back-projection of a pixel and its normal at the model, Edge-ID aims to determine if a control point sampled from an edge is visible or not. Another difference between the methods is that in Facet-ID the model is drawn with filled faces, while in Edge-ID a wireframe model with hidden line removal is rendered. This way, only the visible model edges will have a color value different from the background color. It is then possible to find out if a control point $p(x, y)$ is visible by comparing the index of its edge with the index decoded from the color stored at the position (x, y) in the color buffer. The use of unique IDs for each edge is justified by the fact that points from different edges can be projected to the same position in image space. If no ID checking is performed, a hidden control point could be considered visible. Figure 1 illustrates the proposed visibility testing approach.

In summary, the outline of the Edge-ID method is as follows:

1. Map the color value of each model edge to its index
2. Render the model edges with hidden line removal
3. For each model edge i
   (a) Sample the edge, obtaining control points
   (b) For each sampled point $p(x, y)$
      i. If ID$(x, y) = i$, then the point is visible

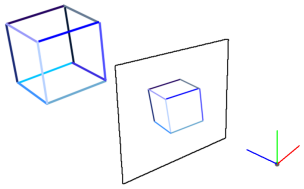

Figure 1: *Edge-ID method.*

Initially, the coding scheme adopted for mapping the IDs to RGB color components was rather simple. The color black ($R = 0$, $G = 0$, $B = 0$) is reserved for representing the background. Then, each edge index is incremented by one and, considering its 24-bit binary representation, the most significant byte is stored at the red channel, the next byte is stored at the green channel and the least significant byte is stored at the blue channel. The inverse process is done for decoding. With this representation, the maximum number of model edges is $2^{24} - 1 = 16,777,215$. The average edge count of the models commonly used for tracking does not even approach this value. Using this coding scheme on the handheld platform presented some problems related to OpenGL ES. In the available graphics library implementation, the primitives are not rendered with the exact color value specified for it. Instead, an approximation is done and a color value close to the original one is used. This leads to confusion between different edges that are drawn with the same color. The solution found to this problem was to choose uniformly spaced values in the color domain for representing the edges. A spacing of 8 levels between consecutive component values was shown to be sufficient in order to prevent confusion between colors. A 15-bit representation was adopted, using only the upper 5 bits of each channel. This way, an edge index i is encoded using the following equations:

$$R = ((i+1)/128) \mod 256, \tag{3}$$

$$G = ((i+1)/4) \mod 256, \tag{4}$$

$$B = ((i+1) \cdot 8) \mod 256. \tag{5}$$

The edge index can then be decoded by:

$$i = 32,768 \cdot R + 1,024 \cdot G + 32 \cdot B - 1. \tag{6}$$

This coding scheme is capable of representing at most $2^{15} - 1 = 32,767$ edges, which is sufficient for 3D tracking applications.

4 Results

A simple standalone application that tracks a 3D cube object and displays a solid cone model registered with it was developed for the handheld platform. It applies the described edge-based tracking algorithm and the computer vision infra-structure. The mobile device used in the tests was a Personal Digital Assistant (PDA) Dell Axim x51v. It has a 624 MHz Intel PXA270 XScale ARMV5 processor, 256 MB of ROM, 64 MB of RAM, and a VGA LCD display with 16-bit color depth. This PDA also has an Intel 2700G multimedia accelerator with 16 MB of video memory. The operating system is Microsoft Windows Mobile 5.0 Pocket PC. The camera used on the mobile device was the Spectec SD (Secure Digital) Camera SDC-001A, with QVGA resolution (320 x 240) and a frame rate of 15 fps (frames per second). The development tool employed to implement the project was Microsoft Visual Studio .NET 2005 Professional Edition. Hardware accelerated OpenGL ES version 1.0 was utilized for 3D graphics rendering, and the GLUT-ES library was in charge of the GUI, application loop, and input handling.

Once working, the application was evaluated taking into account frame rate, accuracy, and robustness metrics. Initially, synthetic QVGA images were used as input. Figure 2 shows some pose estimation results obtained on the handheld platform.

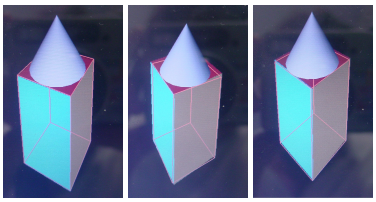

Figure 2: *Tracking results for frames 0 (left), 35 (middle), and 45 (right) of the synthetic sequence.*

Table 1 presents the percentage of time required by each step of the tracking algorithm running on the handheld device and using the cube sequence mentioned above as input. The average total time spent for tracking a frame is 64 ms, which results in a 15.625 fps rate. Around 60 points are tracked during the sequence. Figure 3 shows the total time spent for tracking each of the first 60 synthetic frames. The obtained frame rate is adequate for AR applications, especially those targeted for handhelds.

Table 1: *Percentages of computation time for each step of the tracking algorithm on the handheld platform.*

STEP	TIME (%)
Visible-edge detection	28
Image gray scaling	19
ME	37
Pose calculation	16

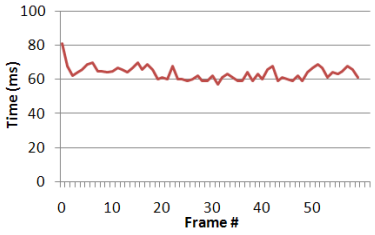

Figure 3: *Total computation time for each of the first 60 frames of the synthetic sequence.*

The camera positions calculated by the tracking algorithm for the synthetic sequence in the x, y and z axes are presented in Figure 4, together with the corresponding ground-truth values.

The average-error values were the following: 4.10 mm in the x axis, 1.79 mm in the y axis and 2.73 mm in the z axis. The average distance between the calculated camera position and the ground-truth was 6.12 mm. The distance between the tracked object and the camera was about 150 mm. The side length of the cube was 60 mm. Since visual perception is very important in AR applications, it is reasonable to say that the obtained error rates are acceptable. While the pose estimation error is not very visually perceptible, tracking accuracy should still be improved.

After the first tests with synthetic data, the handheld, edge-based tracker was evaluated using images of the real world captured by a camera. Figure 5 depicts some augmentation results. The tracker proved to be robust up to a certain level of occlusion of the tracked object. The cube model has 12 contour edges and was tracked at 15

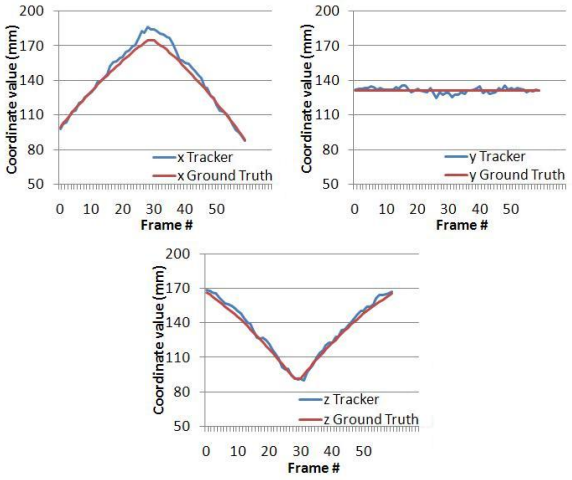

Figure 4: *Estimation accuracy of camera position for the synthetic sequence in the x (top left), y (top right), and z (bottom) axes.*

fps. The wood toy model has 30 contour edges and was tracked at 10 fps. As can be seen, the tracking on the mobile device is highly dependent on the edge count, currently being suitable only for non-complex objects. It can achieve interactive frame rates (4-5 fps) when the object has at most a hundred edges.

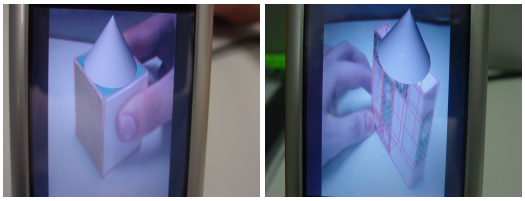

Figure 5: *Handheld augmented reality results using the developed markerless 3D tracker.*

5 Conclusions and Future Work

An edge-based tracking solution for the Pocket PC platform has been developed, which makes handheld based standalone markerless AR applications feasible. The Edge-ID algorithm for edge-visibility checking was developed, exploiting graphics resources available on the mobile platform. We also built a computer vision infrastructure for the Pocket PC platform through a partial port of the VXL and ViSP libraries. Fixed-point math was also used in most calculations to increase performance.

The system provides a reasonable estimation of object pose, visually speaking. However, tracking accuracy can be improved. The frame rate obtained with the test application is suitable for handheld AR when the target object model has a limited number of edges. We believe that with the release of more powerfull PDAs and cell phone architectures, such as the Nvidia Tegra [Rayfield 2008], markerless tracking algorithms such as the one described in this work will be able to handle complex objects. For example, it will be possible to exploit GPU programming and image processing units in these mobile devices.

As future work, robust estimators could be used to improve tracking robustness and accuracy. Also planned for the handheld platform is the implementation of an automatic tracker initialization method [Shahrokni et al. 2002], since this is currently being done manually.

References

BOUTHEMY, P. 1989. A maximum likelihood framework for determining moving edges. *IEEE Transactions on Pattern Analysis and Machine Intelligence 11*, 5 (May), 499–511.

GAUSEMEIER, J., FRUEND, J., MATYSZCOK, C., BRUEDERLIN, B., AND BEIER, D. 2003. Development of a real time image based object recognition method for mobile ar-devices. In *Proceedings of the International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa (AFRIGRAPH)*, ACM, 133–139.

KAHARI, M., AND MURPHY, D. 2006. Mara - sensor based augmented reality system for mobile imaging. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, IEEE, 1 p.

LEPETIT, V., AND FUA, P. 2005. Monocular model-based 3d tracking of rigid objects. *Foundations and Trends in Computer Graphics and Vision 1*, 1, 1–89.

MARCHAND, E., SPINDLER, F., AND CHAUMETTE, F. 2005. Visp for visual servoing: a generic software platform with a wide class of robot control skills. *IEEE Robotics and Automation Magazine 12*, 4 (Dec.), 40–52.

RAYFIELD, M., 2008. Welcome to the next pc revolution. http://www.nvidia.com/content/nvision2008/tech_presentations/Technology_Keynotes/NVISION08-Tech_Keynote-Mobile.pdf.

REIMANN, C. 2005. Kick-real, a mobile mixed reality game. In *Proceedings of the International Conference on Advances in Computer Entertainment Technology (ACE)*, ACM, 387–387.

RIESS, P., AND STRICKER, D. 2006. Ar on-demand: a practicable solution for augmented reality on low-end handheld devices. In *Proceedings of the AR/VR Workshop of the German Computer Science Society*, GI, pp. 119–130.

SHAHROKNI, A., VACCHETTI, L., LEPETIT, V., AND FUA, P. 2002. Polyhedral object detection and pose estimation for augmented reality applications. In *Proceedings of Computer Animation (CA)*, IEEE, 65.

SIEMENS, 2008. Siemens sx1. http://en.wikipedia.org/wiki/Siemens_SX1.

VACCHETTI, L., LEPETIT, V., AND FUA, P. 2004. Stable real-time 3d tracking using online and offline information. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 10 (Oct.), 1385–1391.

VXL, 2008. Vxl - c++ libraries for computer vision. http://vxl.sourceforge.net.

WAGNER, D., REITMAYR, G., MULLONI, A., DRUMMOND, T., AND SCHMALSTIEG, D. 2008. Pose tracking from natural features on mobile phones. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, IEEE, 125–134.

WOODWARD, C., 2006. Augmented reality, feature detection - applications on camera phones. http://cic.vtt.fi/projects/vbe-net/Interactive_3D/Woodward_English.pdf.

WUEST, H., VIAL, F., AND STRIEKER, D. 2005. Adaptive line tracking with multiple hypotheses for augmented reality. In *Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR)*, IEEE, 62–69.