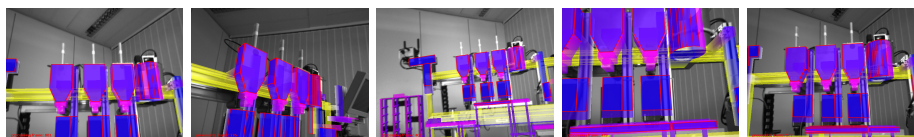


# A real-time learning-free texture-based tracking algorithm

Matthias Grundmann and Selim Benhimane  
{grundman,selim.benhimane}@cs.tum.edu

December 5, 2006



## 1 Introduction

In this project we implemented a new real-time learning-free texture-based tracking algorithm. The problem statement we addressed is, supposing one have a calibrated camera mounted on a HMD or on a mobile robot, to track a piecewise-planar object in an image sequence in real-time to estimate the six parameters of the camera pose. With this information, one can augment the acquired images or estimate the the robot displacement.

Standard methods can be separated into two approaches. Feature based and intensity base. Feature based methods require a preliminary step to detect features, such as points, lines or edges. Then they are using a interframe matching or tracking algorithm to update the pose. While they are robust to deal with partial occlusions they are often highly sensitive to the quality of feature extraction, which can be significantly affected by motion blur or zooming. One the other hand intensity-based methods try to iteratively minimize an error defined by image patches, like the SSD of the current and the reference image. The error parameterizations range from simple 2D transformations to more complex nonlinear parametrization of the 3D camera pose with six degree of freedom. While these methods do not need to extract features, the convergence radius is limited to small inter-frame displacements. So a vital goal is to extend the convergence radius by using second order approximation. While the computation of Hessian in classical approaches is prohibitive for realtime applications the ESM tracking algorithm we used archives second-order optimization while being equivalent to first order algorithms in complexity.

## 2 Our Tracking Application

Our work consists of serveral sections, which can be used separately and each is encapsulated in a DLL. We will describe each sections and how they work together below.

## 2.1 The ESM tracking algorithm

We implemented the ESM tracking algorithm described in ?? in C++ using the Intel Performance Library and the Intel Math Kernel Library. The tracking algorithm is able to track arbitrary polygonal regions simultaneously and output the current camera pose for each frame with respect to the first frame. The input needed to specify are the normal of each region in the model and the calibrated camera. The algorithm also determines the quality of the tracking by correlating the warped reference frame and the current frame. A good track is assumed when the correlation is above 0.9. The tracking algorithm works in real-time and has a complexity of  $O(n^2)$ , where  $n$  are the number of pixels of the tracked image patches. To improve the performance, every data we are working on is aligned to 32 byte boundaries and an intelligent memory layout allows us, to reuse big chunks of data without copying it. We are using masking operations to discard pixels which are outside of the current frame or outside the polygon. To improve speed the Jacobian are packed with respect to the mask, so that unnecessary computations are avoided. To solve the normal equations in each minimization step, we are decomposing the left symmetric matrix with Cholesky decomposition and solve for the solution using a specialized LAPACK routine. We also implemented a gaussian pyramid to first extend the convergence radius and second speed up the tracking significantly.

## 2.2 Tracking 3D Objects

The algorithm described in the previous section was extended to extract the faces of the object automatically. For this approach one need the images, the camera pose in the first reference frame, and a 3D model, created in any 3D modeling software and saved in Alias OBJ-file format, with z-axis pointing upward. The triangulated model is then simplified by our application, so that the largest connected faces are obtained. This reduces the amount of faces to work on by around 75% in our examples. We also construct a 3D BSP-Tree we use for trackable surface determination. This is an offline preprocessing step, that can be done very fast. The result is saved in a so-called simplified model file, which can be read from disk next time. From the absolute position of model in the first frame the trackable faces can be determined automatically. To achieve this the BSP-Tree is used to determine the order of all faces of the model with respect to the z-direction of the camera. Back-faces and too small faces are discarded. Cause the BSP-Tree is static this can be done in just  $O(n)$ , where  $n$  is the number of faces in the model. For details about BSP-Tree please refer to [1] or [3]. From the z-ordered sequence of faces, we obtain a list of those, which are not occluded by others. Once a face is determined to be occluded, it has never to be checked again, so even if the worst case of this occlusion culling is  $O(n^2)$ , the amortized cost are supposed to be around  $O(n)$ , although this was not proven. To deal with partial occlusions a face is defined to be occluded, if more than 5% of the face's area is occluded by other faces. This calculation is done by 2-D polygon intersection test. For this we use 2D BSP-Trees, which make it possible to determine the intersection is just  $O(n+m)$ , where  $n$  and  $m$  are the number of edges of the two faces to intersect. The extracted faces also provide the normal from the model, so one can start tracking immediately. The output of the algorithm is the absolute position of the camera with respect to model in the OpenGL coordinate system, i.e. the z-axis is pointing out of the screen.

## 2.3 Automatic update of trackable faces

The automatic extraction of trackable faces leads to the question, whether this can be done during the tracking procedure to update the faces and increase sequence

of images that can be tracked from the reference images without re-calibration. The naive approach to re-extract the faces leads to error cumulation and drifting of the model. To prevent this we used a two step approach. First when a face  $I$  is first recognized it will be stored in a Database. For each face exists two templates. The one in the Database (called reference template), referring to the first time of extraction  $t_{I^*}$  and the current one, referring to the last time of extraction  $t_I$ . The face is first tracked with the current template which leads to a camera pose  $x$ . Then it is tracked with the reference template starting at  $x$  which leads to camera pose  $x^*$ . The second step is to examine the difference in camera motion  $\|x^* - x\|$ . If it is below a given threshold the reference template in the Database is updated by the current one. This approach is described in more detail in [2] With the automatic update it is possible to track the object over a long sequence from just one key-frame, even when the viewpoint changes dramatically. In our experiments we were able to track a 580 images long sequence, where rotation around 90 degrees and zooming between factor 0.5 and 2.0 occurred.

## 2.4 GUI and augmented reality application

To demonstrate the tracking results, we implemented a GUI to specify the input images, the camera calibration matrix, the position of the camera in the first frame and the 3D model. One also can adjust various parameters of the tracking. We use OpenGL to overlay the model on the image to visually check the tracking results. To achieve this the projection matrix has to match with one from the camera. This was not too easy, cause many known approaches suffer from clipping problems occurring when the principal point is not in center of the image. Furthermore detailed information about the tracking quality in each frame and the iterations used are displayed.

## 3 Results and future work

We developed a complete framework to reliable track 3D-objects in long video sequences in real-time. Each part of the framework can be used separately and incorporated into different projects. A GUI was provided to examine the results visually and adjust the parameters. A re-initialization to recover from situations, where tracking is lost is for the most part implemented but not finished yet.

Future work could be addressing the question about good texture. If the extracted trackable faces have no or very less texture at all, the tracking algorithm theoretically could run into false minima. A long-time work could be to initialize the tracking from structure from motion, so that no 3D model and no key-frame are needed, although accurate structure from motion is not known to done in real-time.

## References

- [1] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer graphics: principles and practice (2nd ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.
- [2] I. Matthews, T. Ishikawa, and S. Baker. The template update problem. In *Proceedings of the British Machine Vision Conference*, September 2003.
- [3] P. J. Schneider and D. Eberly. *Geometric Tools for Computer Graphics*. Elsevier Science Inc., New York, NY, USA, 2002.