

Edge Tracking of Textured Objects with a Recursive Particle Filter

Thomas Mörwald*
Vienna University of Technology

Michael Zillich†
Vienna University of Technology

Markus Vincze‡
Vienna University of Technology

Abstract

This paper proposes a new approach of model-based 3D object tracking in real-time. The developed algorithm uses edges as features to track, which are easy and robust to detect. It exploits the functionality of modern highly parallel graphics boards by performing hidden face removal, image processing, texturing and particle filtering. Using a standard 3D -model the tracker requires neither memory and time -consuming training nor other pre-calculations. In contrast to other approaches this tracker also works on objects where geometry edges are barely visible because of low contrast.

Keywords: Object tracking, edge detection, 3D tracking, real-time, image processing, feature matching

Best to read in color.

1 Introduction

Tracking the pose of a three dimensional object from a single camera is a well known task in computer vision. What seems to be simple for humans turns out to be significantly more complicated for computers. While humans are able to perform highly parallel image processing, even modern *central processing units (CPUs)* have problems calculating the pose of an object with sufficient accuracy, robustness and speed. This leads to the idea of using modern graphics cards, which also work in parallel, to solve this problem. This approach exploits the parallel power of *graphics processing units (GPUs)* by comparing edge features of camera images and a 3D -model.

Graphics boards are designed to render virtual scenes as realistically as possible. The main idea is to compare those virtual scenes with an image captured from reality. Texturing is a common method of simulating realistic surfaces. In this paper, the edges of those textures are used for comparison. Fast progress in computer science will soon allow the inclusion of more and more optical effects like shadows, reflections, shading, occlusions or even smoke, fire, water or fog.

1.1 Related work

One of the first successful approaches of tracking objects by their edges was RAPiD [Harris 1992]. It uses points on edges and searches for correspondence to its surroundings along the edge gradient. However this method lacks robustness and several improvements were applied to overcome this problem as in [Drummond and Cipolla 1999; Philipp Michel 2008; Luca Vacchetti and Fua 2004; Klein and Drummond 2003].

Another approach is to globally match model primitives with those from the camera image [Lowe 1992; Gennery 1992; D. Koller and Nagel 1993; Kosaka and Nakazawa 1995; A. Ruf and Nagel 1997]. This method has been used for robot and car tracking applications, but was later replaced by improved versions based on RAPiD.

[Lucie Masson and Jurie 2004] also uses edges and textures for tracking. They extract point features from the texture and use them together with the edges to calculate the pose. This turns out to perform very fast and robust against occlusion. Our approach not only uses patches but the whole texture, which usually lets the pose converge very quickly to the accurate pose. Since the algorithm runs on the GPU, it is as fast as the method in [Lucie Masson and Jurie 2004].

The work presented in [M. Vincze and Zillich 2001] uses edge features to track but does not take into account texture information. This makes it less robust against occlusion. Since the search area in that approach is very small, it is also less robust against fast movement and getting caught in local minima.

The work presented in this paper is based on [Klein and Murray 2006] where they take advantage of graphics processing by projecting a wireframe model into the camera image. Then a particle filter with a Gaussian noise model is used to evaluate the confidence level with respect to the pose.



Figure 1: Edges from geometry vs. edges from texture

Our approach not only uses geometry edges but also edge features from textures which extends the class of trackable models by those that have curved surfaces as illustrated on the right of Figure 1. This is because in a standard 3D -model curvature is approximated by triangles and quadrangles which would produce virtual edges which do not correspond to the actual edges as shown on the left of Figure 1. The particle filter is extended by using it in a recursive design that evaluates a single pose estimate given by the most likely particles.

1.2 Overview

The idea of this approach is to

- extract the edges from the incoming camera image,
- extract the edges from the textured 3D -model,
- generate hundreds of slightly different views of the model relative to a pose estimate,
- calculate the most likely pose of the model by matching the edges of the camera image and the 3D -model.

The algorithm developed is separated into two parts. The *preprocessing* in Section 2, where all possible pre-calculations are made

*e-mail: moerwald@acin.tuwien.ac.at

†e-mail: zillich@acin.tuwien.ac.at

‡e-mail: vincze@acin.tuwien.ac.at

and the *recursive particle filtering* in Section 3, which generates several hundred poses and for each of them evaluates the confidence level. Therefore the second part is very time -crucial. A *linear Kalman filter* described in Section 4 is applied for smoothing the resulting trajectory. Section 5 gives some hints on how to implement the proposed methods. The *results* in Section 6 and the *conclusion* in Section 7 summarize the advantages and strengths of the presented tracker.

2 Preprocessing

In the preprocessing stage of the algorithm the edge image I_C^e of the incoming camera image I_C is extracted and stored for comparison later. The color surface S of the object to track is *projected* into the incoming image I_C and the edges are calculated again. This edge image of the object I_S^e is then used to *re-project* to the geometry in world space which results in the corresponding edge map of the original surface S^e .

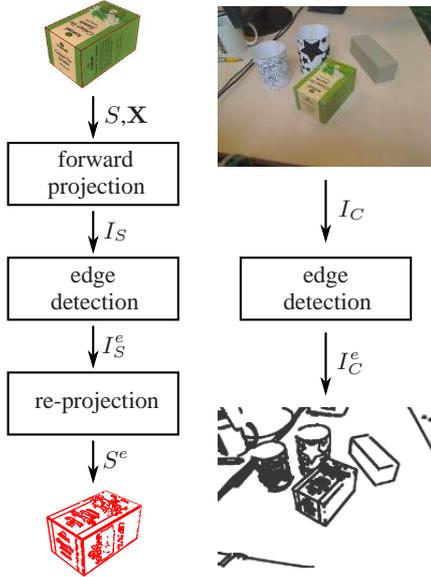


Figure 2: Flow chart of preprocessing

2.1 Edge detection

The edges I^e are found by convolving the original image I with a Gaussian smoothing \mathbf{G} and the two Sobel kernels $\mathbf{H}_{s,x}$ and $\mathbf{H}_{s,y}$ as described in [Burger and Burge 2008].

$$I^e = \begin{pmatrix} I_x^e \\ I_y^e \end{pmatrix} = \begin{pmatrix} \mathbf{H}_x * \mathbf{G} * I \\ \mathbf{H}_y * \mathbf{G} * I \end{pmatrix} \quad (1)$$

Furthermore, the result is improved by applying thinning and spreading algorithms. Note that for the gradient calculation in Section 3.2 the x and y values are stored separately.

Figure 3 shows the different results of edge detection where the x - and y -components of the gradient are stored in the red and green color channel. The detection tolerance can be influenced by applying spreading, which broadens the edges by a specific number of pixels according to the tolerance level. This means that instead of searching for edge pixels close to each other, the line width of the edge is raised as shown in Figure 3, which broadens the matching area.

2.2 Forward projection

The 3D -model is projected into the camera image I_C as defined by Equation (2). Using the camera image I_C takes into account that edges are not visible when there are similar light and color conditions in the background. Then the edges of the image are extracted using Equation (1).

The transformation of the model from world space to image space is performed by the following matrix operations:

$$\mathbf{u}_S = \mathbf{T}_p \mathbf{X} \mathbf{v}_S \quad (2)$$

$$I_S(u, v) = \begin{cases} S(\mathbf{v}_S) & \text{if } (u, v) \in \mathcal{U} \\ I_C(u, v) & \text{else} \end{cases}$$

where I_S is the camera image with the projected model. \mathcal{U} defines the geometry of the object in image space with

$$\mathbf{u}_S = [u_S, v_S] \in \mathcal{U}$$

\mathbf{T}_p denotes the projection- and \mathbf{X} the model view or world transformation matrix which defines the pose of an object to track with a rotational and translational term \mathbf{R} and \mathbf{t} .

$$\mathbf{X} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$$

The geometry of the object in world space \mathcal{V} is represented by its vectors

$$\mathbf{v}_S = [x_S, y_S, z_S] \in \mathcal{V}$$

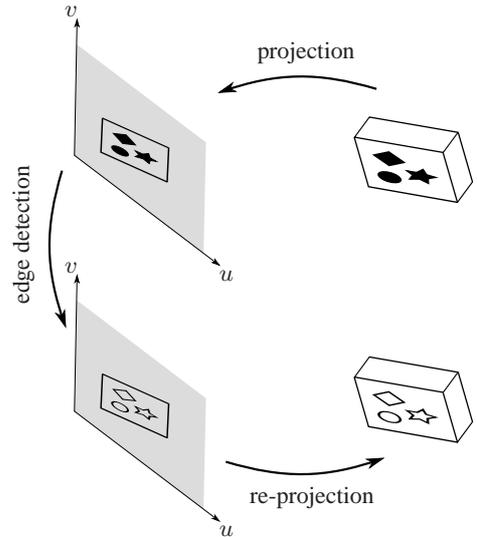


Figure 4: Forward projection and re-projection

2.3 Re-projection

The idea of re-projection is to replace the color surface of the 3D -model with the corresponding edge map. Note that it is not possible to do the image processing on the color surface S of the object directly, as the edge features get distorted and thinned out when they are projected to image space and therefore wrongly fail the edge matching test. Comparing the edges of the model with the camera image requires the same methods applied to the same point of view and also the same scaling of the edge width.

Using a particle filter requires drawing the model several hundred times at different poses \mathbf{X}_i with $i = 1 \dots N$. Replacing the surface

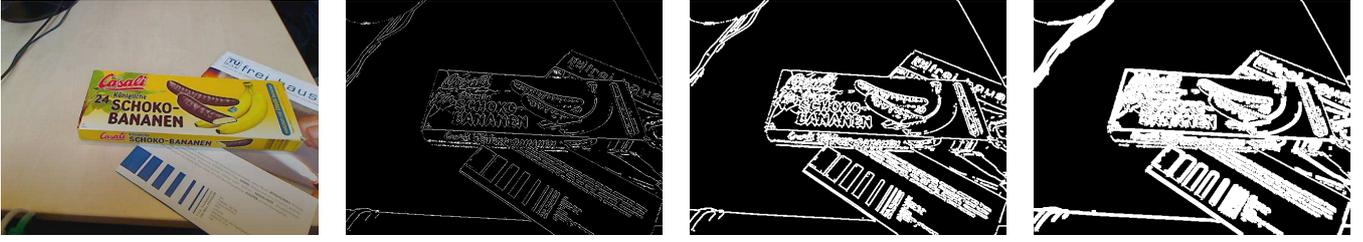


Figure 3: Edge detection results from left to right: original, no spreading, one time spreading, three times spreading

and running the edge detection algorithm for each particle would cause the tracker to be far away from real-time capability. For this reason, the surface of all particles is replaced by only one edge map I_S^e , calculated by using the prior tracking result \mathbf{X}^+ . This, in principle causes the same problems as mentioned above, but assuming that the motion of the particles \mathbf{X}_i remains small within one tracking pass, the distortions and thinning out can be disregarded.

3 Recursive particle filtering

For each tracking pass the recursive particle filtering executes the methods shown in Figure 5. First the particles $i = [1 \dots N]$, representing the pose of the object, are generated using Gaussian noise. Then the confidence level of each particle i is evaluated by matching it against the edge image of the camera I_C^e . If there is still processing time t_f remaining, then a further recursion step of particle generation and evaluation of the confidence levels is performed with different parameters as described in Section 3.4. Otherwise the maximum likely particle is passed to the next step. The linear Kalman filter, including a physical motion model, is attached to the outcome of the recursive particle filter to fine tune the result and remove remaining jitter. As this additional filter is not part of the recursion it is explained in Section 4.

The reason for this setup is to benefit from the robustness and speed of a particle filter. For higher accuracy, the standard deviation of the noise is reduced in each recursion. The linear Kalman filter is attached just for fine tuning as mentioned above.

3.1 Particle generation

The prior pose \mathbf{X}_i^- of each particle is calculated by perturbing the posterior \mathbf{X}^+ with Gaussian noise $\mathbf{n}(\sigma^2)$ with a standard deviation scaled by the prior confidence level of the pose w_m , a scaling factor for motion effect \mathbf{f}_m and a scaling factor \mathbf{f}_c set by each recursion step:

$$\begin{aligned} \mathbf{X}_i^- &= \mathbf{X}^+ + \mathbf{n}(\sigma^2(w_m, \mathbf{f}_m, \mathbf{f}_c)) \\ i &= [1 \dots N] \end{aligned} \quad (3)$$

The standard deviation is evaluated by

$$\sigma = \sigma_I \mathbf{f}_m \mathbf{f}_c \cdot (1 - w_m) \quad (4)$$

where the confidence level of the prior pose w_m is multiplied to the initial standard deviation σ_I so that the particle distribution narrows with higher confidence. The motion effect \mathbf{f}_m takes into account that motion in world space along the camera viewing axis causes less change in image space than the same motion orthogonally to the viewing axis. \mathbf{f}_c becomes smaller with each recursion step in the particle filter. σ_I is implemented as a parameter to be set by the user, but should be evaluated automatically in the future regarding the tracking conditions.

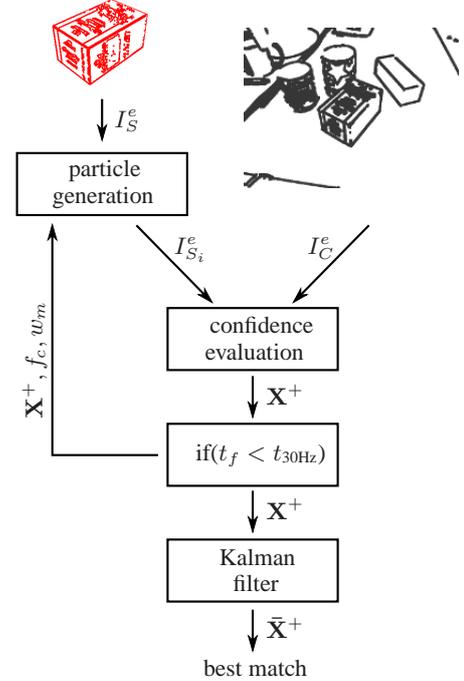


Figure 5: Block scheme of motion with recursive particle filter and Kalman filter

3.2 Confidence level

Each particle is tested against the camera image and a confidence level is calculated. Therefore the correlation between the gradients of the edges $\mathbf{g}_{S_i}(u, v)$ and $\mathbf{g}_C(u, v)$ is evaluated by comparing the direction of the edges at each image point (u, v) .

$$\begin{aligned} \mathbf{g}_{S_i}(u, v) &= \begin{pmatrix} I_{S_i, x}^e(u, v) \\ I_{S_i, y}^e(u, v) \end{pmatrix} \\ \mathbf{g}_C(u, v) &= \begin{pmatrix} I_{C, x}^e(u, v) \\ I_{C, y}^e(u, v) \end{pmatrix} \end{aligned}$$

The angles between those vectors are calculated, producing the edge correlation image Φ_i :

$$\begin{aligned} \phi &= \arccos(\mathbf{g}_{S_i} \cdot \mathbf{g}_C) \\ \Phi_i(u, v) &= \begin{cases} 1 - \frac{2\phi}{\pi} & \text{if } \phi < \pi/2 \\ 1 - \frac{2|\phi - \pi|}{\pi} & \text{if } \phi > \pi/2 \\ 0 & \text{if } (u, v) \notin \mathbf{v}'_S \end{cases} \quad (5) \end{aligned}$$

Note that it is assumed that the result of the $\arccos()$ function lies within 0 and π . The image Φ_i now contains the degree of correlation between the pose suggested by the particle i and the camera

image. The angular deviation of the edge angles Φ_i is scaled to the range of 0 to 1.

The confidence level w_i is evaluated as follows:

$$w_i = \left(\frac{m_i}{n_i} + \frac{m_i}{n_{max}} \right) \frac{1}{W} \quad (6)$$

with

$$\begin{aligned} m_i &= \int_u \int_v \Phi_i(u, v) dv du \\ n_i &= \int_u \int_v |I_{S_i}^e(u, v)| dv du \\ W &= \sum_{i=1}^N w_i \\ n_{max} &\propto \max(n_i) \end{aligned}$$

The first term within the brackets is the percentage of matching edge pixels m_i with respect to the total visible edge pixels n_i . Calculating the confidence level only with this term would cause the tracker to lock when only one side of the 3D object is visible. When at this special pose the object is rotated slightly, another side of the object becomes visible. The particle matching this rotation would be equal or most often less than the prior front facing particle. The reason for this is that the number of matching pixels m_i grow less than the total visible pixels n_i when rotating the object out of the front side view. This effect amplifies when taking into account that edge detection for strongly tilted faces is very faulty.

The second term allocates more weight to the total number of matching pixels m_i which is intrinsically higher for the rotated particle. n_{max} which are the maximum visible edge pixels in the actual area scales the pixels to the proper range. As this summation would lead to confidence levels higher than 1, it is divided by the sum of confidence levels W .

This differs from the function used in [Klein and Murray 2006]

$$\text{Likelihood}(\mathbf{X}_i^-) \propto \exp\left(k \frac{d_i}{v_i}\right)$$

which we experienced to lock very fast at the local minima mentioned above. Here d_i denotes the number of matching edge pixel, v_i the total edge pixels of the wireframe model, where the hidden edges are removed and k is a constant for distributing the likelihood.

3.3 Determining the pose

As explained in the sections 2.2 and 2.3 for projection and re-projection of the model, a single pose \mathbf{X}^+ has to be defined. This is where the approach suggested in this paper differs from usual, straight forward particle filters, where the whole propability density function

$$\begin{aligned} w_i &= P(\mathbf{X}_i^-) \\ i &= [1 \dots N] \end{aligned}$$

of the previous timestep, is carried over into the next estimation step.

The pose \mathbf{X}^+ is evaluated using the mean of the top most likely particles M . \mathbf{X}_k^- in Equation (7) denotes the particles sorted by the confidence level w_k in descending order.

$$\mathbf{X}^+ = \frac{1}{w_m} \sum_{k=1}^M \mathbf{X}_k^- \cdot w_k \quad (7)$$

with

$$w_m = \frac{1}{M} \sum_{k=1}^M w_k$$

Experiments have shown that increasing the number of most likely particles M to consider in the mean confidence, while lowering the standard deviation in Equation (4) by f_c and the edge width (see Figure 3) for each further recursion obtains good results.

3.4 Recursion

The methods described in Section 3.1 and 3.2 are performed for each of the hundreds of particles. The idea of recursion is to take advantage of the information gain when calculating. This means that the N particles are divided into subranges R_1, R_2, R_3, \dots and so forth. For every range R_k , the pose estimate \mathbf{X}^+ and confidence level w_m of the previous particle filtering R_{k-1} is used. The standard deviation for the particle generation is reduced by the scaling factor f_c , which narrows the *search area* of the filter. Therefore Equation (3) becomes

$$\begin{aligned} \mathbf{X}_i^- (R_k) &= \mathbf{X}^+ (R_{k-1}) + \mathbf{n}(\sigma^2) \\ i &= [1 \dots N_k] \end{aligned}$$

with

$$\sigma = \sigma_I f_m f_c(R_{k-1}) \cdot (1 - w_m(R_{k-1}))$$

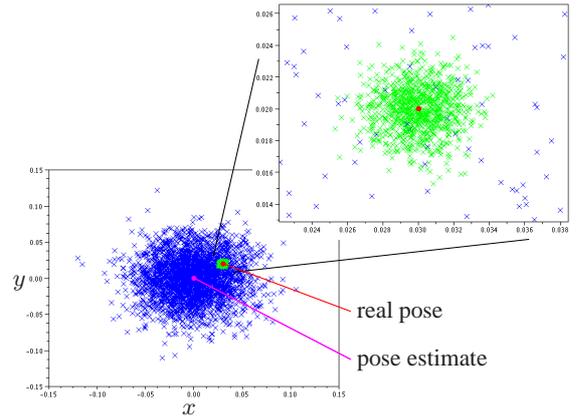


Figure 6: Recursive particle filter

Figure 6 shows the principal idea of recursive particle filtering in 2D. In the lower left graph the particles are perturbed using a high standard deviation of the Gaussian noise of the particles, covering a large area around the prior pose estimate. The mean of the top most likely particles is used to evaluate the rough pose of the real object. The upper left graph shows particles with lower standard deviation, where this time the most likely particles measure the real pose much more accurately.

The example in Figure 6 is drawn with 1500 particles with high and 500 with low standard deviation. This method allows the tracker to respond both quickly and accurately without increasing the tracking time, because it does not require any more particles than before.

4 Linear Kalman filtering

The discrete Kalman filter implemented for this approach uses a motion model for all six degrees of freedom of the object. The

reason why the motion model is not applied in the particle filter is because this would reduce the speed and accuracy of the tracker, as modelling the velocity of the object would rise the degree of freedom from 6 to 12. This would mean that the particles have to cover 6 more dimensions. Therefore the Kalman filter is attached only to smooth the resulting trajectory and remove remaining jitter.

Time Update - "Predict"

$$\begin{aligned}\mathbf{x}_k^- &= \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_k \\ \mathbf{P}_k^- &= \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^T + \mathbf{Q}\end{aligned}$$

Measurement Update - "Correct"

$$\begin{aligned}\mathbf{K}_k &= \mathbf{P}_k^- \mathbf{H}^T \left(\mathbf{H}\mathbf{P}_k^- \mathbf{H}^T + \mathbf{R}(w_m) \right)^{-1} \quad (8) \\ \mathbf{x}_k &= \mathbf{x}_k^- + \mathbf{K}_k (\mathbf{z}_k - \mathbf{H}\mathbf{x}_k^-) \\ \mathbf{P}_k &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}) \mathbf{P}_k^-\end{aligned}$$

where

$$\mathbf{x}_k = [x_k, y_k, z_k, \alpha_k, \beta_k, \gamma_k, \dot{x}_k, \dot{y}_k, \dot{z}_k, \dot{\alpha}_k, \dot{\beta}_k, \dot{\gamma}_k]$$

denotes the state of the Kalman filter containing the pose and velocity of the six degrees of freedom.

$$\mathbf{A} = \begin{bmatrix} \mathbf{I} & \text{diag}(\Delta t) \\ \mathbf{0} & \mathbf{I} \end{bmatrix}$$

is the *state matrix*,

$$\mathbf{B} = [\text{diag}(0)]$$

the *input matrix*,

$$\mathbf{H} = [\mathbf{I}, \text{diag}(0)]$$

the *output matrix*, \mathbf{P}_k the *estimate error covariance*, \mathbf{Q} the *process noise covariance*, \mathbf{R} the *measurement noise covariance*, \mathbf{I} the unity matrix and \mathbf{K}_k the *Kalman gain*. Please refer to [Welch and Bishop 2004] for more details on Kalman filtering.

The process covariance matrix \mathbf{Q} defines the noise of the physical model, where no information is available. However, in contrast to \mathbf{R} , this matrix is set to a fixed value.

The measurement covariance matrix \mathbf{R} depends on the confidence level of the pose as follows:

$$\mathbf{R}(w_m) = \begin{bmatrix} \text{diag}((w_{t-3})^3) & 0 \\ 0 & \text{diag}((w_{t-3})^3) \end{bmatrix}$$

As shown in Equation (8), this means that \mathbf{R} rises proportionally to the delayed confidence level of the measurement from the particle filter. Therefore, the Kalman gain drops, giving less weight to the measurement \mathbf{z}_k and more weight to the motion model, which smooths the result. This means that jitter is removed when the confidence is high, for example when the object to track does not move.

On the other hand, the lag behind the real object, caused by the motion model during acceleration, is decreased when the confidence falls, which usually happens when the object to track moves fast. In this case, the Kalman gain increases the weight of the measurement, increasing the speed but also allowing jitter which is barely visible when the object is moving anyway.

Experimentants have shown that a cubic function yields to a smooth tracking behaviour. The delay of the confidence level w_{t-3} removes overshooting of the motion model when the object suddenly stops moving.

5 Implementation notes

The implementation of the algorithm requires discretization which is denoted by bold letters for images in this section.

5.1 Notes for preprocessing

The preprocessing is done once per tracking pass and is not as time critical as the recursive particle filtering in the following section. However, the overall performance of the tracker needs to be as fast as possible, so this part is also implemented using the graphics processing unit. Therefore, the image received by the camera is sent to the graphics board where it is stored as texture. The convolution with the Gaussian and Sobel kernels are applied by shaders as well as thinning and spreading. The edge image is again stored as *RGB* texture where the *R*- and *G*-channels are used for the *x*- and *y*-components of the image gradient.

As the channels only allow values between 0 and 1, the normalized gradients ranging from -1 to 1 need to be adjusted.

The surface of the object to track is made up of vertices which form triangles and quadrangles, also called primitives. The projections described in Equation (2) and (9) are performed for these vertices only, because the surface points within a primitive can be determined by linear interpolation, which is optimized by the graphics adapter.

\mathbf{T}_p and \mathbf{X} denote the projection and model view matrix, which can be queried from OpenGL. The six degrees of freedom are stored in the vector

$$\mathbf{x}_i = [x_i, y_i, z_i, \alpha_i, \beta_i, \gamma_i]$$

Instead of transforming the edge map I_S^e back to world space by solving Equation (2) with respect to S , the coordinates for the edge map in image space are evaluated. This has to be done only once, since those coordinates do not change for the other particles.

$$\begin{aligned}\mathbf{u}_S^+ &= \mathbf{T}_p \mathbf{X}^+ \mathbf{v}_S \quad (9) \\ I_{S_i}^e(u, v) &= I_S^e(\mathbf{u}_S^+)\end{aligned}$$

The particles i are represented by the pose matrices \mathbf{X}_i^- . The vectors \mathbf{u}_{S_i} that are used to find the corresponding point in the camera image I_C^e are calculated with

$$\begin{aligned}\mathbf{u}_{S_i} &= \mathbf{T}_p \mathbf{X}_i^- \mathbf{v}_S \\ I_{C_i}^e(u, v) &= I_C^e(\mathbf{u}_{S_i})\end{aligned}$$

At this point for each particle i the model $I_{S_i}^e$ and camera image $I_{C_i}^e$ are ready for comparison which is described in Section 3.2.

5.2 Notes for recursive particle filtering

The calculations described in this section are very critical with respect to optimized programming, since every single line of code is called several hundreds of times. In particular evaluating the confidence level using the *NVIDIA Occlusion Query* is definitely a bottle-neck in the algorithm. This OpenGL extension is responsible for counting the pixels of the whole edge map. The matching pixels m_i and total edge pixels n_i are evaluated by summing up the correlation image Φ_i and the edge map $\mathbf{I}_{S_i}^e$:

$$\begin{aligned}m_i &= \sum_{u,v} \Phi_i(u, v) \\ n_i &= \sum_{u,v} \mathbf{I}_{S_i}^e(u, v)\end{aligned}$$

As this extension only supports counting pixels disregarding the value, they only can be marked to be rendered or not. Therefore, the displacement image is thresholded by the angle ϵ :

$$\begin{aligned} \phi &= \arccos(\mathbf{g}_{S_i}(u,v) \cdot \mathbf{g}_C(u,v)) \\ \Phi_i &= \begin{cases} 0 & \text{if } \phi < \epsilon \\ 1 & \text{if } \phi \geq \epsilon \end{cases} \end{aligned} \quad (10)$$

where ϵ denotes the angular threshold. For the match evaluation shader 0 means that the pixel is discarded when rendering and does not increase the counter, whereas 1 means the pixel is drawn and therefore increases m_i .

Figure 7b) shows the pixels m_i successfully passing the match evaluation. 7c) are the pixels which fail the match test of Equation (10) and 7d) is the total number of pixels n_i of the edge image of the object. Note the missing pixels on the very upper edge, as a result of the background having the same color (yellow) as the object. The mismatch of the edges on the front side of the box is caused by the inaccuracy of the placement of the texture on the geometry of the model which is produced manually by a 3D modeling tool.

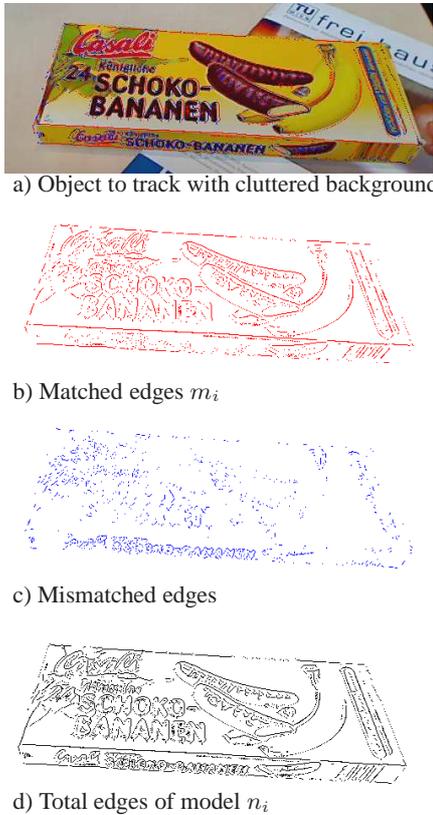


Figure 7: Edge matching

6 Results

As the tracker requires fast parallel calculations, the focus of the system is on the graphics board where it is implemented. It has been tested on an NVIDIA GeForce GTX 285 with a fill rate of 50 billion pixels per second, and an Intel Core2 Quad CPU Q6600 with 2.4 GHz. To fulfill a minimum frame rate of 30 FPS (frames per second), the time for one tracking pass is 33 ms. Within this limits it is possible to draw 1500 particles for a box (6 faces textured with a 900x730 pixel image) like in Figure 7a). With the cylinder

model shown in Figure 1 with 16 faces and a 900x400 pixel texture image, the tracker achieves 580 particles within the same time and 100 particles with the cylinder with 64 faces.

Figure 8 shows some results of a video sequence. The first row demonstrates the robustness of the tracker. The whole top surface and the edges of the geometry are covered by the hand and there are reflections of the checkerboard pattern on the front face and a cluttered background, but the pose of the box still can be estimated. Of course at some degree of occlusion, the accuracy of the pose drops until it cannot be determined at all.

The second row shows the fast and reliable convergence. Although the deviation of the estimated from the actual pose is very high, the tracker finds the correct alignment within a second. The third row illustrates the concept of recursive particle filtering. Especially in the second image from the left, where the speed of the moving object is high, the benefit is clearly visible (compare with Figure 6).

7 Conclusion

We presented a method for fast and robust object tracking. It converges fast to the correct pose and is able to handle relative large deviations, for example when initializing. Partial occlusion, reflections, light changes, shadows and cluttered background are no problem for the tracker, as long as enough features are visible to determine the pose. Exploiting the power of a graphics processing unit with a particle filter in a recursive design allows high tracking speed with sufficient accuracy.

However, there are several improvements possible. Firstly the mismatch of the 3D -model to the real object, as described in Section 5.2, can be learned and corrected with constraints that prevent the model from strong distortion. Secondly corner detection, color matching and so forth can be implemented which would most likely further improve the accuracy and robustness of the tracker.

The correlation map Φ_i in Equation (5) is simplified to Equation (10), because the *NVIDIA Occlusion Query* only counts visible pixels disregarding the information about the angular displacement stored within. A future work would be to implement precise evaluation of the confidence level as described in Equations (5) and (6).

The bottle-neck, with respect to the frame time of this approach, is definitely the particle filter with its evaluation of the confidence level using the OpenGL extension. The tracking error e directly correlates with the standard deviation σ and number of particles N as follows:

$$e \propto \frac{\sigma}{N}$$

with

$$N \propto t = t_{33ms}$$

This means that the tracking error e can be significantly reduced by lowering the standard deviation σ for each degree of freedom independently. When for example the z position of an object to track is known, because it lies on a table the accuracy can be increased by lowering the standard deviation for this degree of freedom.

There are several points of the algorithm where further investigation needs to be done, like finding the optimal boundary conditions and functions for the recursive particle filter and designing a better function for particle generation. Or more specifically, designing better functions for calculating the standard deviation of the Gaussian noise in Equation (4).

A further problem that needs to be solved is, that the tracker cannot supply information if tracking fails when it locks into a local min-

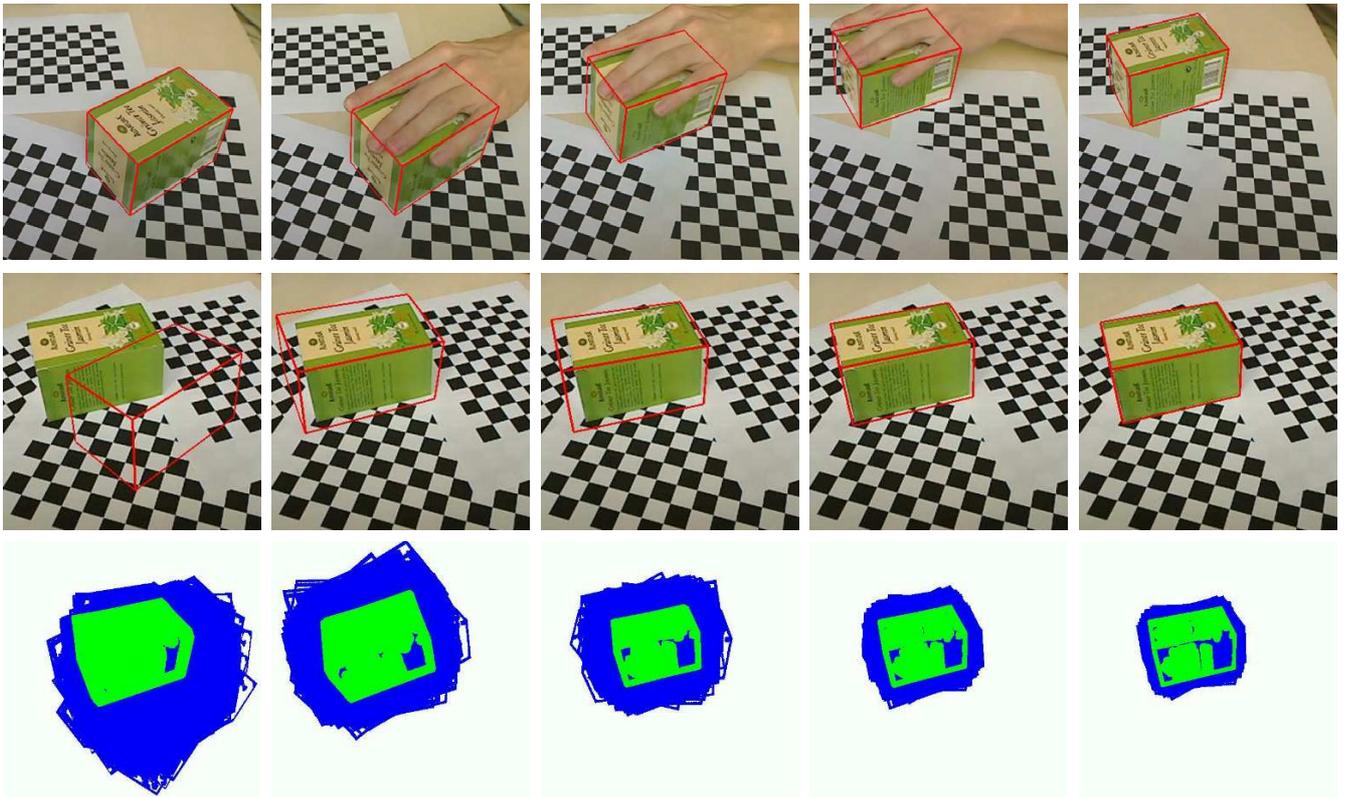


Figure 8: First row: robustness against occlusion, reflections and background clutter; Second row: fast and robust convergence; Third row: particle distribution with three recursions

imum. There, the confidence evaluation returns sometimes values as high as at the correct tracking pose.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement No. 215181, CogX.

References

- A. RUF, M. TONKO, R. H., AND NAGEL, H.-H. 1997. Visual tracking by adaptive kinematic prediction. *Proceedings of International Conference on Intelligent Robots and Systems*.
- BURGER, W., AND BURGE, M. J. 2008. *Digital Image Processing, An Algorithmic Introduction Using Java*. Springer.
- D. KOLLER, K. D., AND NAGEL, H.-H. 1993. Model-based object tracking in monocular image sequences of road traffic scenes. *International Journal of Computer Vision*.
- DRUMMOND, T., AND CIPOLLA, R. 1999. Real-time tracking of complex structures with on-line camera calibration. 574–583.
- GENNERY, D. 1992. Visual tracking of known three-dimensional object. *International Journal of Computer Vision*.
- HARRIS, C. 1992. Tracking with rigid objects. *MIT Press*.
- KESSENICH, J. 2008. *The OpenGL Shading Language, Version 1.30*.
- KLEIN, G., AND DRUMMOND, T. 2003. Robust visual tracking for non-instrumented augmented reality.
- KLEIN, G., AND MURRAY, D. 2006. Full-3d edge tracking with a particle filter. *British Machine Vision Conference Proc 17th*.
- KLEIN, G., AND MURRAY, D. 2007. Parallel tracking and mapping for small ar workspaces. *Proc International Symposium on Mixed and Augmented Reality (ISMAR)*.
- KOSAKA, A., AND NAKAZAWA, G. 1995. Vision-based motion tracking of rigid objects using prediction of uncertainties. *International Conference on Robotics and Automation*.
- L. VACCHETTI, V. L., AND FUA, P. 2004. Stable real-time 3d tracking using online and offline information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- LOWE, D. G. 1992. Robust model-based motion tracking through the integration of search and estimation. *International Journal of Computer Vision*.
- LUCA VACCHETTI, V. L., AND FUA, P. 2004. Combining edge and texture information for real-time accurate 3d camera tracking.
- LUCIE MASSON, M. D., AND JURIE, F. 2004. Robust real time tracking of 3d objects.
- M. VINCZE, M. AYROMLOU, W. P., AND ZILLICH, M. 2001. Edge-projected integration of image and model cues for robust model-based object tracking. *The International Journal of Robotics Research*.

- MUSTAFA ÖZUYSAL, MICHAEL CALONDER, V. L., AND FUA, P. 2009. Fast keypoint recognition using random ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- P.A. SMITH, I. R., AND DAVISON, A. 2006. Real-time monocular slam with straight lines. *Proc 17th British Machine Vision Conference* (sept).
- PHILIPP MICHEL, J. C. E. A. 2008. Gpu-accelerated real-time 3d tracking for humanoid autonomy.
- R. KOCH, K. KOESER, B. S., AND EVERS-SENNE, J.-F. 2005. Markerless image-based 3d tracking for real-time augmented reality applications. *WIAMIS* (april).
- ROST, R. J. 2006. *OpenGL Shading Language*, vol. Second Edition. Addison-Wesley.
- SEGAL, M., AND AKELEY, K. 2008. *The OpenGL Graphics System: A Specification, Version 3.0*.
- SUN-KYOO HWANG, M. B., AND KIM, W.-Y. 2008. Local descriptor by zernike moments for real-time keypoint matching. *IEEE Congress on Image and Signal Processing*.
- VINCENT LEPETIT, JULIEN PILET, P. F. 2004. Point matching as a classification problem for fast and robust object pose estimation. *Conference on Computer Vision and Pattern Recognition* (june).
- WELCH, G., AND BISHOP, G. 2004. An introduction to the kalman filter.