Adaptive Real-Time 3D Acquisition and Contour Tracking within a Multiple Structured Light System

Andreas Griesser[1] Thomas P. Koninckx[2] Luc Van Gool[1,2]

[1]Swiss Federal Institute of Technology (ETH),
D-ITET / Computer Vision Laboratory,
Zürich, Switzerland
{griesser,vangool}@vision.ee.ethz.ch

[2]Katholieke Universiteit Leuven,
ESAT / VISICS,
Leuven, Belgium
{tkoninckx,luc.vangool}@esat.kuleuven.ac.be

Abstract

Active 3D acquisition setups currently capture 3D data from a single viewpoint. We propose a system that supports the effective use of multiple structured light modules placed around the object of interest. We show how oppositely positioned modules could work together and how we can avoid those parts of the background that unnecessarily take up computation time. Key to the setup is having the structured light patterns masked outside the outlines of the object to be captured. This is done very fast, so that the system can be used for motion capture. Only off-the-shelf hardware is required.

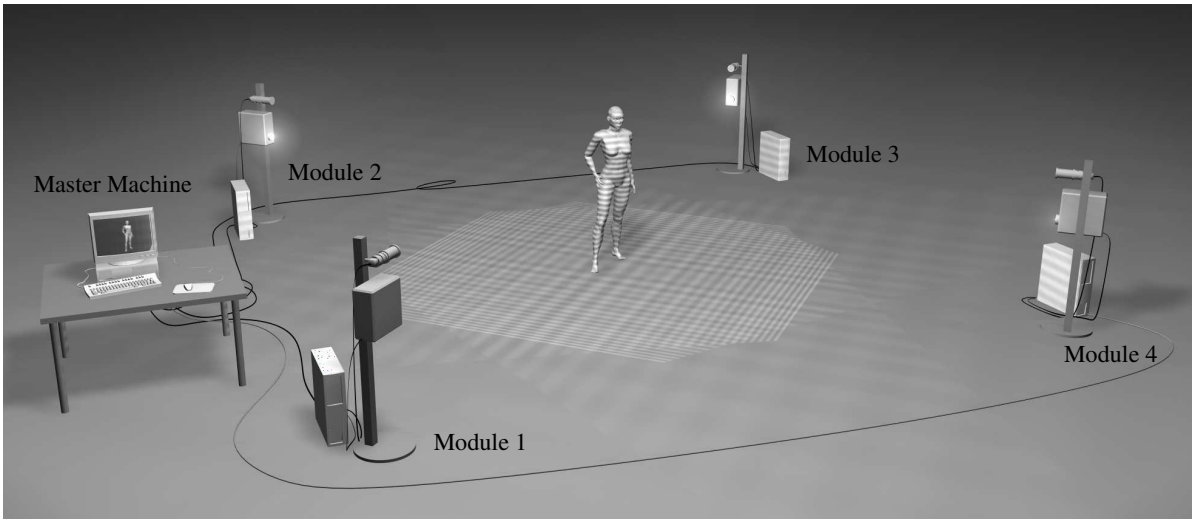
1. Introduction

Efficiently acquiring complete 3D shapes of objects still is a challenging problem. One can put several cameras around an object, but the quality of stereo based reconstructions tends to be low, especially in cases of weakly textured surfaces. Structured light techniques simplify the search for correspondences that is typical for stereo vision, but pose the problem that multiple such modules can hardly work simultaneously due to interference of the different structured light patterns. Therefore, active 3D acquisition setups currently capture 3D data from a single viewpoint. The system and object are then rotated with respect to each other in order to build up complete models. Here, we propose a setup that increases the efficiency of such model building. Not only do we build upon a state-of-the-art structured light module, we also increase the degree to which such modules can operate at the same time. Being able to quickly capture 360^o 3D is especially important for motion capture. Indeed, letting different modules capture their data one after the other would slow down the process to a point where the dynamics are no longer captured well. The results in this paper demonstrate how at least opposite modules can operate together, thereby already providing much denser 3D data than marker-based systems would, and this at 13 Hz.

Another important issue is that 3D acquisition systems - active or passive alike - will often capture irrelevant parts of the background. This not only means that one has to spend time on the calculation of these useless 3D data, but that one will also face the problem of having to remove such unwanted data from the final model of the desired object.

We propose work that attempts to remedy or at least reduce these problems. The authors of [1] and [8] already demonstrated the feasability of real-time structured light with a sequence of different, projected stripe patterns, but their method was restricted to relatively slow motion. We build on other, recent results in structured light technology, that have made it possible to get good 3D data from the projection of a single pattern (so-called one-shot 3D acquisition). These systems have the obvious advantage that they at least take only a very short time to acquire the necessary data, which may also come from an object that changes its shape during acquisition [7, 5, 6]. Most of these systems do not process the data on-line, however. A one-shot system that allows to also extract the 3D data while scanning has recently been proposed by Koninckx *et al.* [4, 3]. That method only requires off-the-shelf hardware (standard colour video camera, LCD projector, and PC). The structured light pattern used consists of vertical white stripes, and the stripes are identified via the detection of their intersection with diagonal colored stripes. Our setup combines several such modules. The structued light patterns used by these modules is adapted further, however.

Indeed, we mask the structured light pattern beyond a small margin around the desired object's outlines. The shape of the mask is dynamically updated as the object moves. The mask blocks the structured light from hitting extended parts of the background and focuses the system's computation time on the part of the scene that matters. As



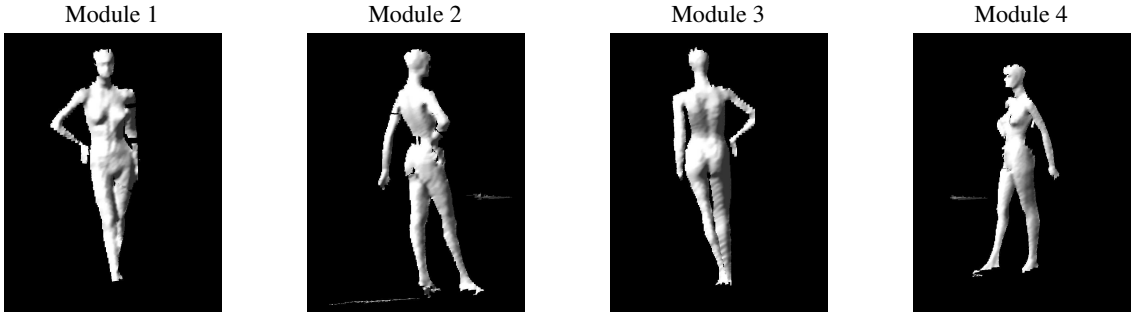


Figure 1. Multi-module structured light setup

a result, background data is eliminated from the start. Also, the other systems don't get to see sufficiently large parts of the other systems' light patterns in order for them to try and generate nonsensical data out of them. In addition to the masks, a spot crops out the structured light within the projection cone that would hit the camera of an opposite module, thereby preventing the opposite system being blinded in such cases. This is needed as soon as the object moves out of the axis between opposite modules. Figure 2 shows what a blinded camera's image looks like, when the scanned object is not blocking the line of sight between a projector and the opposite camera. Occurrence of this situation is also detected on the basis of the mask. This ensures that pairs of opposite modules can work concurrently at all times.

The paper is organized as follows. Section 2 describes the outline of the system, section 3 explains how the projection mask is calculated and how clipping of background items is done. In section 5 the tracker is described and section 4 shows how the tracked and masked pattern is projected using graphics hardware. Section‘6 shows some of the results. The performance is measured in section 7 and section 8 concludes the paper.

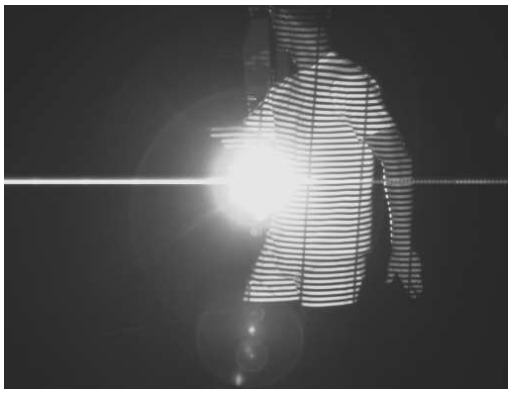


Figure 2. Blinded CCD-camera image

2. Overview

Our system is a combination of four scanning modules. Each module is composed of one digital camera, one LCD beamer, and one PC with framegrabber card. The PCs run the one-shot 3D acquisition application based on a real-time kernel [3]. It consists of two steps. The first detects

and indexes stripes in the camera image. The second transforms them into projector coordinates, performs depth clipping, and generates the mask delineating the person from the stripe endpoints on the body. After a short initial sequence of 3D snapshots a tracking algorithm further adapts the shape of the mask as the person moves.

2.1. Hardware Setup

As mentioned before, the setup is scalable in terms of the number of scanning modules - our current setup consists of four entities and is depicted in fig. 1. Each module's acquisition pipeline is initiated by an image capture, triggered by the master machine which is connected to the trigger input of the cameras through a digital I/O-board. A 100MBit ethernet network is used for transmitting the 3D-data to the master machine and to control the behaviour of the modules.

2.2. One-Shot Structured Light Kernel

The acquisition kernel projects a pattern of vertical, equidistant black and white stripes, covering a sparser set of oblique lines that assist in vertical stripe identification and that have scene adapted colors [4, 3]. The grabbed image is processed as follows:

- image preprocessing and stripe pattern detection
- finding the underlying code and labeling the stripes by assigning corresponding line numbers
- relaxation and subpixel-refinement
- reconstruction in 3D by triangulation

The result of the third step is a list of detected stripes in the camera image. For each the corresponding line number in the projector image is found through labeling. Fig. 3 shows part of an input image (top) and the resulting stripes (bottom).

2.3. Extensions to the Current Kernel

We extended the reconstruction pipeline as shown in fig. 4, where the original algorithm (right column of the figure) is described in earlier work [4, 3]. The stripe output is no longer directly provided to the 3D triangulation step, but is first used for creating the projection mask. Because this mask is presented in projector coordinates we transform all stripe-endpoints into the projector frame as described in section 3.1. Then clipping is applied based on the camera coordinates so that objects outside of the scanning area are not taken into account (see section 3.2). The tracking algorithm computes a stripe-specific placement prediction for the next projected frame (see section 5). The projection mask representing the object's silhouette and is generated from the predicted stripes or from the clipped stripes directly when tracking is disabled. Section 3.3 explains this

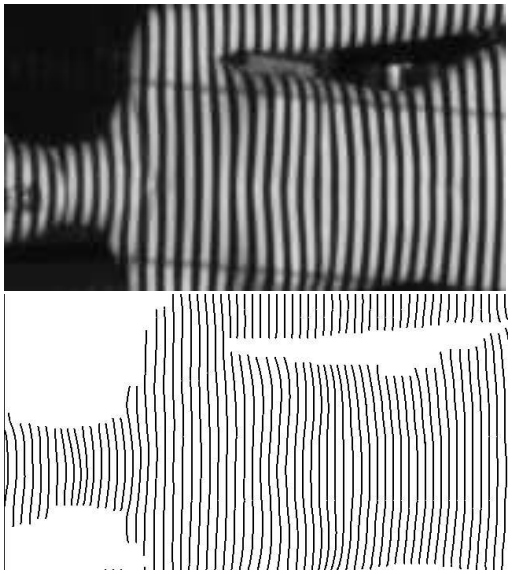


Figure 3. Grabbed input image (top), detected and labeled stripes (bottom)

algorithm in detail. The drawing of the masked projection pattern is completely done on the GPU. Section 4 describes the techniques of mask-offset and tesselation.

3. Mask Creation

The relaxation unit of the acquisition kernel outputs a list of detected and labeled stripes s_i, $i = 1..N$, in the camera image with N being the total number of stripes. Each item stores not only the two endpoints (top and bottom) of a stripe but also intermediate values with subpixel accuracy in horizontal direction.

3.1. Camera-Projector Transformation

The extraction of the projection mask delineating the person is based on where the stripes end on the body. Before connecting the endpoints, these have to be transformed into projector coordinates. This is done by simple epipolar geometry using the Fundamental matrix. From the calibration of the setup we retrieve both 3x4 projection matrices C_{cam} for the camera and C_{proj} for the projector and thus the epipolar geometry F_{c2p} between camera and projector. A point $x_c = [u_c, v_c, 1]^T$ in the camera image thus corresponds to the epipolar line

$$l' = [a, b, c]^T = F_{c2p} \cdot x_c = F_{c2p} \cdot [u_c, v_c, 1]^T \quad (1)$$

We assume we know the line number l_x of this point in the projected grid and the width w of each stripe. The point x_c represented in projector space, denoted as x_p, can now be calculated by

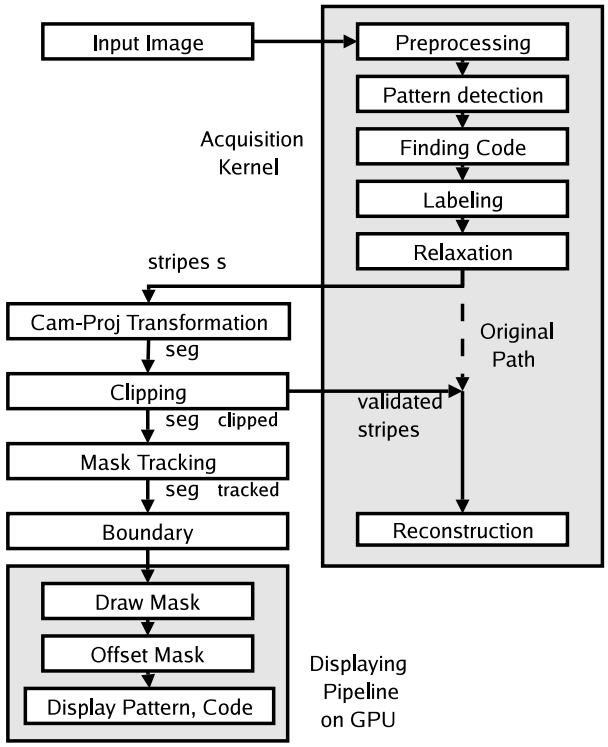


Figure 4. Overview of the acquisition-, masking-, and display-pipeline.

$$x_p = \begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix} = \begin{bmatrix} l_x \cdot w \\ -\frac{a \cdot l_x \cdot w + c}{b} \\ 1 \end{bmatrix} \quad (2)$$

The list of labeled stripes containing the top-endpoint $x_{c,t}$ and bottom endpoint $x_{c,b}$ is converted into a list of segments $seg_i = \langle u_{p,i}, v_{p,t,i}, v_{p,b,i} \rangle$, $i = 1..N$, by transfering both the top endpoint, indexed by t, and the bottom endpoint, indexed by b, into projector coordinates using the equation above.

3.2. Clipping

In order to free the acquired 3D-dataset from unwanted objects in the cluttered environment one may apply one of the following rules:

- Exclude unwanted objects from the segment list by marking them as invalid so that they will not be projected and detected from the camera and therefore not be reconstructed.
- Mark the unwanted objects as not visible within the original stripe list. Hence they will be visible in the next camera frame, but they won't be reconstructed by the acquisition kernel.

The first method has the advantage of not modifying the original stripe list but the effect on the 3D-dataset is delayed by one frame. We combine both methods and obtain a speedup in the overall process because instead of all the stripes only the valid ones (see further) need to be processed.

We assume that the camera-projector pairs are positioned around the scanned object. These modules define the overall scanning area and since all cameras and projectors are calibrated against a common world coordinate system the extrinsic parameters express the translation and rotation of their optical centers to this common origin.

Clipping occurs when an object has a distance to the camera-center greater than the defined scanning area. Also stripes projected onto the ground floor should be eliminated. Within the field of view of one projector we apply clipping planes slightly above the ground and for the maximal depth. Points lying behind one of these planes will be discarded and the corresponding segment will no longer be taken into account.

We could apply this rule to all pixels along a stripe corresponding to a segment seg_i. If one pixel fails the depth- or ground-test the whole segment would be discarded. In practice it suffices to check only the two endpoints of each stripe, reconstruct them in 3D, and if one of them fails the clipping test, the whole segment is deleted from the segment list and the corresponding stripe is marked as invalid.

Further details about the reconstruction of the two segment endpoints in 3D based on the camera coordinate system has already been described elsewhere [4] and will therefore not be explained here. Note that after the mask has been initialised based on the first clipping results, updates of the mask ('tracking') will reduce the amount of clipped data to a minimum from then on. The creation of the mask is discussed next.

3.3. Boundary creation

We now create a contour around the segments that survived clipping, following a connected component labeling type of procedure.

First, all segments $seg_{clipped,i}$ with $i = 1..N_{clipped}$ which passed the clipping test are sorted by line number (u-coordinate) and v-coordinate. Each computed boundary is represented by a polygon, stored as a linked list of vertices containing the pixel coordinates in projector space. During processing, nodes are added to this list n_k while sweeping a scanline in u-direction through the list of segments.

For each position of the scanline, denoted as *current line (cur_l)*, the affected segments have to be tested against the ones in the previous position of the scanline, denoted as *previous line ($prev_l$)*, for one of the following cases (see fig. 5):

1 If the current segment in cur_l does not overlap any segment in $prev_l$ then both endpoints are added to the list of nodes n_k and linked together (from top to bottom).

2 If a segment in $prev_l$ does not overlap any segment in cur_l then both endpoints are added to the list of nodes n_k and linked together (from bottom to top).

3 For all overlapping segments the following two actions must be taken:

3a: The top-most and bottom-most endpoints of a series of overlapping segments are added to the list of nodes n_k. The top points are linked together from the current line to the previous line, the bottom points are linked together in reverse direction. In fig. 5 the fourth column shows how two segments are connected to the one overlapping segment in the fifth column.

3b Two or more segments in cur_l which are overlapped by a single segment in $prev_l$ will be connected as shown in the fifth column of fig. 5: the top endpoint of the lower segment is linked to the bottom endpoint of the upper segment. In case one segments in cur_l overlaps two or more segments in $prev_l$ then these segments are reverse linked as shown in the third column of fig. 5.

Since each segment is only used once for processing the overall computing time is of $O(N)$. The algorithm computes all possible closed boundaries and deals well with occlusions. Fig. 5 includes one such occlusion with a different sense of orientation of the linked nodes than the outer contour itself.

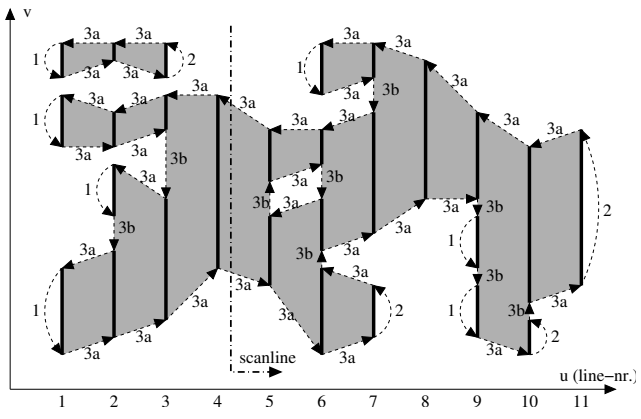


Figure 5. Contour finding: two boundaries with one of them having an occlusion can be found in $O(N)$ time. The dotted lines indicate the linkage between the nodes within the boundary list n_k.

4. Mask Offseting and Drawing

The object contour can be used to mask the projected stripe pattern. However, as the person will move, part of the body will have moved out of the masked region and part of this region will project outside of the body. Inevitably, the next time around the maks will have shrunk, as the intersection of the previous mask and the area covered by the body. In order to counteract this effect, the mask is enlarged. We achieve this through *polygon-offsetting*. The amount of offset is a tradeoff between minimizing either parts of the environment that are reconstructed (small offset) or the risk of losing part of the body out of sight (large offset). A good compromise could be found by keeping the offset narrow, but adding mask tracking.

Offsetting a 2D polygon can typically be done in two ways: offset all edges separately and then trim the resulting offset segments, or use the combined Voronoi diagram of all input contours. In [10] an algorithm using cone distance functions and Z-Buffering is described. This gives the opportunity to implement the offset on graphics hardware. An approximation of the offset pixel-error resulting from hardware pixel discretisation is given in [2] .

The basic idea behind discrete Voronoi diagrams is that the distance function for a line segment on the contour is composed of three parts: one for the segment itself and one for each endpoint. When representing distance as height above a point, for an endpoint the influence is formed by a downward cone and for the line segments it is represented by an inverse tent. The Depth-Buffer on the graphics hardware resolves intersections between the individual cones and tents. Each vertex and edge of the input line segment is a feature in the Voronoi diagram and in case they have different colors the rendered result is the discrete approximation of this *per-feature Voronoi diagram*.

If such a diagram is clipped at a depth d_{offs} we obtain the offset polygon. Since in our application we just need to know the orthographic projection of this isoline onto the plane of the segments, it suffices to consider the circles and rectangles resulting from this height clipping. This also reduces rendering time because we do not need the Depth-Buffer tests any more.

Figure (6) shows how a disk in OpenGL is drawn by slices which subdivide the disk around the z-axis. For a given offset-distance $R = d_{offs}$ and offset pixel-error ϵ the number of slices N_{slices} for one disk is computed by:

$$N_{slices} = \frac{2\pi}{\alpha} = \frac{\pi}{\arccos\left(1 - \frac{\epsilon}{d_{offs}}\right)} \qquad (3)$$

To confine stripe projection to the resulting mask, we use the ALPHA-test on the graphics hardware. The overall display pipeline is shown in figure (7) and consists of the following steps (see e.g. [9] for more details about OpenGL):

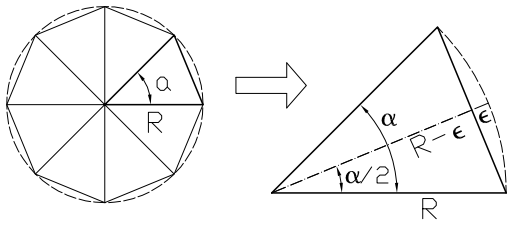


Figure 6. OpenGL: a disk is subdivided around the z-axis into a number of slices.

1. For all input polygons calculate the number of disk-slices for a given error ϵ and given offset-distance d_{offs}.
2. Clear the alpha buffer, disable ALPHA-testing and use RGBA=(0,0,0,1) as further drawing color.
3. Triangulate the polygons by using the OpenGL-Tesselator. The winding rule as described in [9] is set to NONZERO so that occlusions, i.e. polygons with reversed sense of orientation, will be handled in a correct manner.
4. Draw the offset contour by using two-dimensional discrete voronoi diagram.
5. The ALPHA buffer now is set to 1 where at least one of the previous functions has drawn something.
6. Finally paint the stripe-pattern with activated ALPHA-test. Only where ALPHA is set to 1 by one of the previous steps the stripes will be visible, the rest will remain unchanged, i.e. background color.

The impact of masking on the overall scanning process is depicted in fig. (8). A human-sized model is placed between two scanning modules and the background is cluttered with office and laboratory equipment. The projector-camera units are vertically oriented so that the stripes are now horizontally aligned. The top left image shows the fully projected stripe pattern (together with 7 green coding lines, see [3]). In the camera image one can also see the projected pattern in the background. The 3D reconstruction thus includes meshes from unwanted objects. Based on this result the detected and clipped stripes are projected into projector space and the boundary is created which serves as input for the mask-offsetting algorithm. An offset of $d_{offs} = 7$ pixels and $N_{slices} = 16$ is applied. Now that a mask is in place, the subsequent camera image shows no pattern on the background and the reconstruction only consists of the object of interest.

Offsetting the projection mask can solve the problem of regaining the object's silhouette but with the risk of blending the opposite camera in the scanning setup as soon as there is a direct line of sight between a projector and the opposite camera that is not covered by the scanned item (see

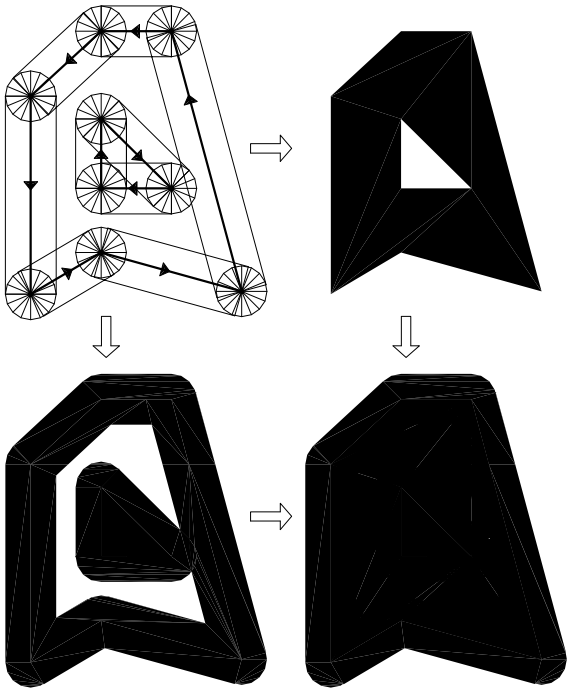


Figure 7. Display pipeline for the projection mask. Top-Left: input polygon with 5 vertices on the outer contour and 3 vertices on a nested polygon. Top-Right: Tesselation of the input polygon. Bottom-Left: offset-mask with $N_{slices} = 16$. Bottom-Right: Combined Mask

figure 2). From the system calibration we know the location of all cameras and projectors and thus all direct lines of sight. Therefore, we can black out the position of the opposite camera in the projector pattern, i.c. with a superimposed black circle. Nevertheless the projector can still be seen as a white spot, but as experiments have shown this doesn't affect the reconstruction pipeline at all. The black circle is supressed as soon as the scanned object blocks the line of sight by more than the mask offset, i.e. its place is always within the grown margin around the actual object contour. In the case of figure 7 this would be the part shown at the bottom left. Otherwise it would only unnecessarily suppress part of the stripe pattern, with a hole in the 3D reconstruction as a consequence. The effect of this procedure is visible in fig. 10 in the camera images of the left two columns. Even when the axis between the two opposite modules is not covered by the scanned person blinding is eliminated.

5. Mask Tracking

In order to let the projection mask follow the scanning object's movement we need to apply tracking to the

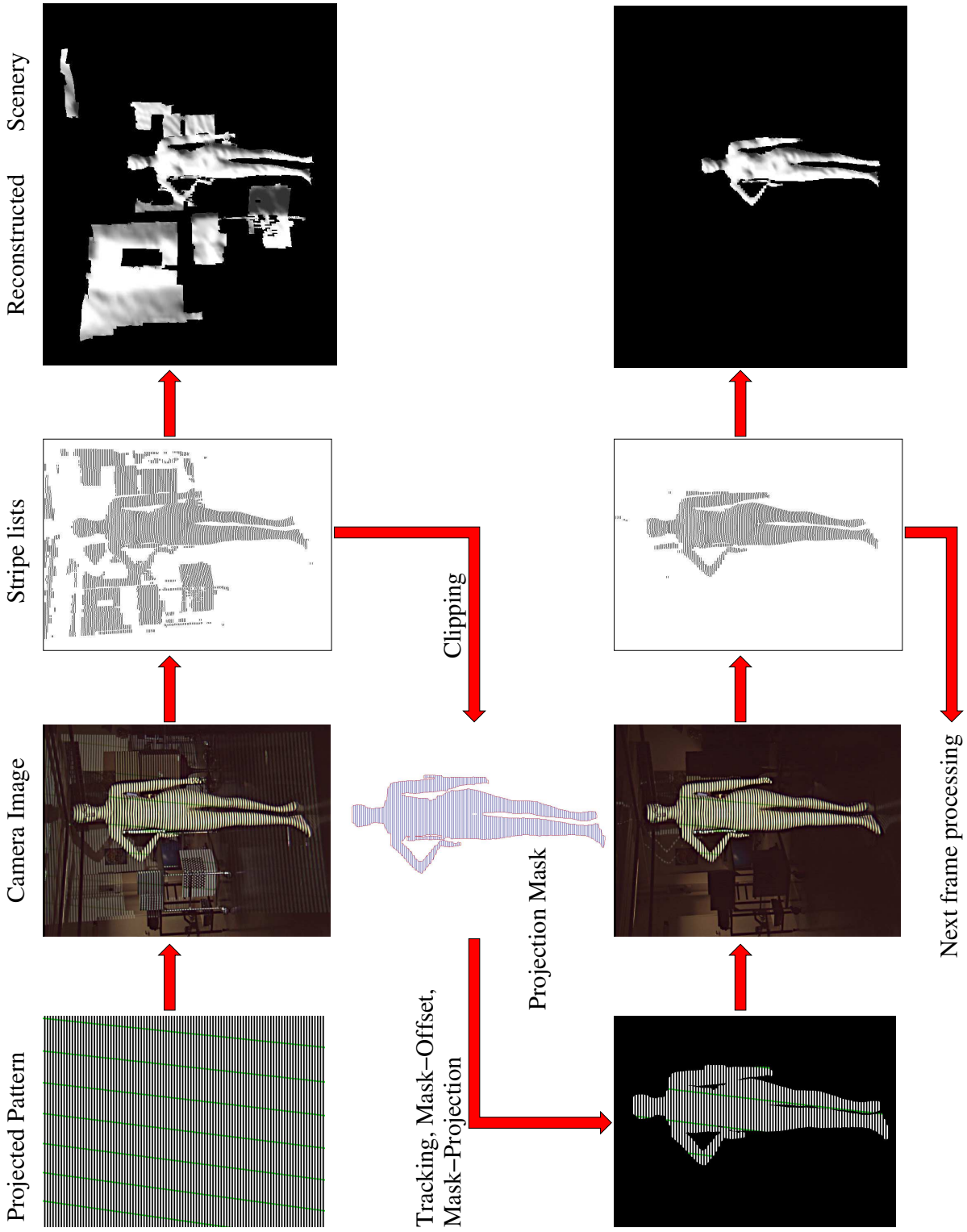


Figure 8. The effect of masking the projection pattern. Top Left: reconstruction without using mask. Top Right: masked projection pattern. Only the object of interest is reconstructed.

mask-polygon. The real-time constraint limits the options. It makes it difficult to use the full 3D-dataset of the object. An active contour based approach was also discarded because of this. We have implemented a simple, one-dimensional stripe-based technique.

¿From an input image at the time t_0 we calculate the silhouette of the scanned object as described in section 3. Then we apply clipping and retrieve segments $seg_{clipped,i,t_0}$ with $i = 1..N_{clipped}$. Assume we find, for a segment $seg_{clipped,i,t_0}$ at the current time t_0, a spatially overlapping segment $seg_{clipped,j,t_{-1}}$ in the segment list of the previously taken frame at the time t_{-1} (see fig. 9, a). Looking at the top endpoint of both segments we calculate the difference in u-direction $\Delta u_t = u_{i,t,t_0} - u_{j,t,t_{-1}}$. A straightforward prediction for the next frame would be to add this difference to the current endpoint. However, we introduce a factor α which controls the emphasis of the current frame with respect to the previous one:

$$\Delta u'_{p,t} = \alpha \cdot \Delta u_t \tag{4}$$

If $\alpha = 0$ no tracking occurs and always the current segment will be taken as prediction for the next frame. With $\alpha = -1$ the current frame will be ignored and therefore we do not trust the actual frame at all. Empirically we found $\alpha = 0.7$ as the best fitting value.

If we use the equation above and we have overlapping segments with big differences in u-direction, we possibly get too long or too short segments as a prediction. Hence a limitation of $\Delta u_{p,t}$ is applied, given by

$$\Delta u_{p,t} = min\left(\alpha \cdot \Delta u_t, \Delta u_{max}\right) \tag{5}$$

where Δu_{max} is a constant value. The tracking algorithm now iterates through all the segments at the time t_0 and checks for one of the following cases (see fig. 9):

- **a:** only one previous segment overlaps the current segment: apply equation (5) to both endpoints.
- **b:** no previous segment overlaps the current one: take the current segment. If no current segment overlaps a previous one, discard it.
- **c:** after applying equation (5) the predicted segment has reversed orientation: discard this segment.
- **d:** more than one previous segment overlaps the current one: find the nearest top and bottom endpoints and apply equation (5) to them. This step merges disconnected segments into continuous one.
- **e:** a previous segment is splitted into more than one current segment. If we apply equation (5) directly the result is in all likelihood wrong. A better solution is the next case
- **f:** for each detected gap in two consecutive segments at t_0 calculate the middle point $u_m = (u_{1,b} + u_{2,t})/2$. This middlepoint is projected into t_{-1} and virtually splits the previous segment so that again equation (5) can be applied.

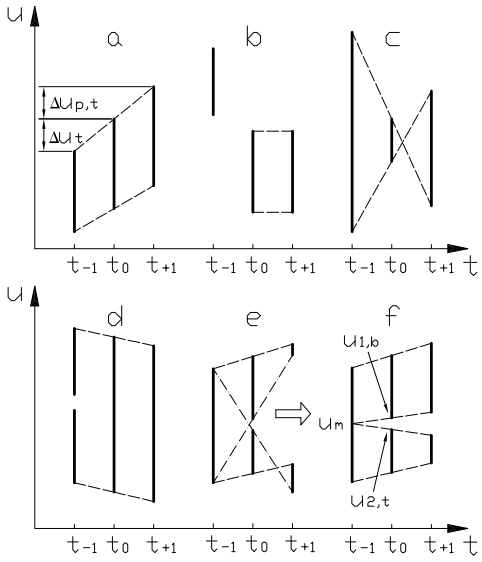


Figure 9. Possible cases for tracking of the pattern stripes.

As already mentioned in section 4, tracking supports fast recovery of the object's silhouette because a prediction for the next position of the object in space is computed. On the other hand, robustness is increased with a mask offset so that in case of mislabeling and therefore wrong tracking inputs, a pattern is still projected onto the object. Nevertheless, the offset should not be too big in order to eliminate the background from the reconstruction.

6. Results

In figure 10 the results of our masking and tracking algorithm can be seen on a person moving within the scanning area while scanning from two opposite sides. The two left columns show the input images from both cameras. Again the camera-projector units are vertically oriented. Even when the person is not exactly situated along the axis of two modules, the cameras are not blinded by the opposite projector. Indeed the projector itself is visible as a white spot but blinding effects, like in figure 2, are no longer visible and correct pattern detection is guaranteed.

The overall speed of the system is 13fps with two simultaneous scans. As can be seen in the fourth column of figure 10, the body model is not completely reconstructed due to little light along the borderline. Hence we positioned another two scanning modules perpendicular to the first pair. By having these units again work in parallel the missing parts of the body can be filled in and a 360^o 3D model can be generated with just two consecutive trigger signals.

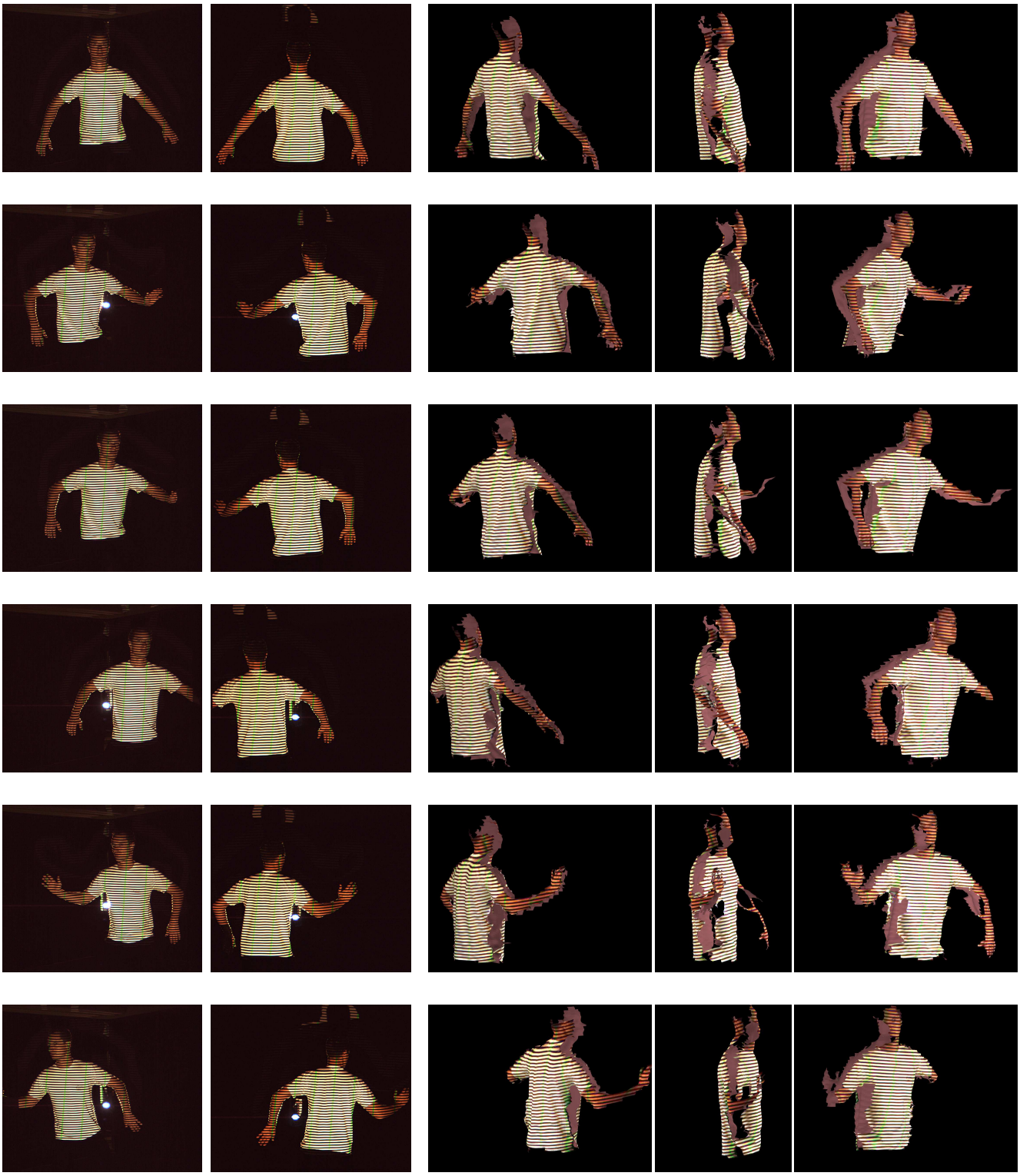


Figure 10. Result of Tracking and scanning from two opposite sides. The left two columns show the camera images of two oppositely positioned scanning modules.

7. Implementation and Performance

One 3D acquisition module consists of a single PC running Linux, a Framegrabber, an industrial camera with Base-CameraLink interface and 690x512 RGB resolution, and an off-the-shelf LCD projector with 1024x768 resolution. Four such modules have currently been installed whereas the baseline between camera and projector is about 50cm.

All cameras are externally triggered by a master-computer which also collects the 3D-data from the different modules. An active trigger from this master machine initiates the acquisition pipeline on the modules. Fig. 4 shows the different steps within this pipeline.

The scanning room has dimensions $6 \times 6m^2$. In order to test the clipping and masking, we put some office and laboratory equipment around and close enough to the object of interest so that the pattern projected on them would still be sharp and have sufficient contrast to allow for their 3D reconstruction.

The following time measurements on one of the scanning computers is based on a Pentium 4 CPU with 2.67 GHz, 1GB RAMBUS RAM and a nVidia Geforce4 Ti GPU graphics card. The scanned object is human-size doll, the projected stripe-width is 5 pixels and the number of detected stripes in the camera image after labeling is about 500, which leads to mask-polygons with about 900-1000 vertices and edges. The offset-distance for the projected mask is 5 pixels and the number of slices per drawn disk within the offset mask is 16.

Task	avg. time [ms]	description
Compute Mask	2.1	Cam2Proj-Transf., Clipping,Boundary
Mask Tracking	0.2	
Tesselate Mask	2.5	OpenGL-Transfer
Offset Mask	2.1	OpenGL-Transfer

The OpenGL-Transfer times are needed for sending the display commands to the GPU. They could be reduced by using faster AGP-busses.

8. Conclusions and Future Work

The work aims at the simultaneous and flexible use of multiple structured light systems. In particular, we have demonstrated how two oppositely placed, one-shot systems can be used to acquired dense motion capture data for a person. A masking stragety avoids unnecessarily spending time on background clutter. Avoiding directly projecting into the lens of the opposite camera also avoids that modules blend eachother. A simple tracking combined with rapid polygon offsetting strikes a good balance between big and small mask sizes.

Currently only one pair of oppositely positioned 3D acquisition modules are used, but a second one is available already, oriented orthogonally to the axis of this first pair. We want to start alternatiing between the two pairs in order to still better cover the complete body, as there are small holes where projection and viewing directions of a single pair are grazing, i.e. near the silhouettes in the images. In order to reduce the time difference between the trigger of the first pair and the second one, we can modify the timing in a modular way within the master machine. Immediately after the first pair has finished the camera exposure, the projectors can be switched off and the perpendicular pair can be activated. One pair can project a pattern while the modules of the other are computing 3D data and their next projection masks.

Acknowledgments

The authors gratefully acknowledge the support of Canon AG, Switzerland, and Tomas Svoboda for his contribution to the camera calibration. The authors also gratefully acknowledge support by KULeuven GOA project 'MARVEL' and SNF NCCR project 'IM2'.

References

[1] O. Hall-Holt and S. Rusinkiewicz. Stripe boundary codes for real-time tructured-light range scanning of moving objects. In *Int. Conf. on Computer Vision*, pages 359–366, 2001.

[2] K. E. Hoff, III, J. Keyser, M. Lin, D. Manocha, and T. Culver. Fast computation of generalized voronoi diagrams using graphics hardware. In *Proc. of the 26th conf. on comp. graph. and interactive techniques*, pages 277–286, 1999.

[3] T. Koninckx, A. Griesser, and L. Van Gool. Real-time range scanning of deformable surfaces by adaptively coded structured light. In S. Kawada, editor, *4. intern. conf. on 3-d dig. imaging and modeling*, pages 293–302, October 2003.

[4] T. Koninckx and L. Van Gool. High-speed active 3D acquisition based on a pattern specific mesh. In *Videometrics 7*, Jan. 2003.

[5] M. Maruyama and S. Abe. Range sensing by projecting multiple slits with random cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(6):647–651, 1993.

[6] M. Proesmans and L. Van Gool. One-shot active 3D image capture. In *Proc. SPIE*, volume 3023, pages 50–61, 1997.

[7] M. Proesmans, L. Van Gool, and A. Oosterlinck. Active acquisition of 3d shape for moving objects. In *Int. Conf. on Image Processing*, pages 647–650, 1996.

[8] S. Rusinkiewicz, O. Hall-Holt, and M. Levoy. Real-Time 3D Model Acquisition. In *Proc. of Siggraph*, 2002.

[9] D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGl Programming Guide*. Addison Wesley, 2003. Version 1.4.

[10] J. Smith. Voronoi Diagrams and Contour Offsets. www.cs.berkeley.edu/˜ jordans/research/classes/meshes.