Methods for Indexing Stripes in Uncoded Structured Light Scanning Systems

Alan Robinson
Sheffield Hallam University
Howard Street
Sheffield S1 1BW
United Kingdom
a.robinson@shu.ac.uk

Lyuba Alboul
Sheffield Hallam University
Howard Street
Sheffield S1 1BW
United Kingdom
l.alboul@shu.ac.uk

Marcos Rodrigues
Sheffield Hallam University
Howard Street
Sheffield S1 1BW
United Kingdom
m.rodrigues@shu.ac.uk

ABSTRACT

This paper presents robust methods for determining the order of a sequence of stripes captured in an *uncoded structured light* scanning system, i.e. where all the stripes are projected with uniform colour, width and spacing. A single bitmap image shows a pattern of vertical stripes from a projected source, which are deformed by the surface of the target object. If a correspondence can be determined between the projected stripes and those captured in the bitmap, a spatial measurement of the surface can be derived using standard rangefinding methods. Previous work has uniquely encoded each stripe, such as by colour or width, in order to avoid ambiguous stripe identification. However, colour coding suffers due to uneven colour reflection, and a variable width code reduces the measured resolution. To avoid these problems, we simplify the projection as a uniform stripe pattern, and devise novel methods for correctly indexing the stripes, including a new *common inclination constraint* and *occlusion classification*. We give definitions of patches and the continuity of stripes, and measure the success of these methods. Thus we eliminate the need for coding, and reduce the accuracy required of the projected pattern; and, by dealing with stripe continuity and occlusions in a new manner, provide general methods which have relevance to many structured light problems.

Keywords

Stripe indexing, structured light, 3d scanner, surface occlusions.

1. INTRODUCTION

The goal of structured light techniques is to measure the shape of three dimensional objects using automatic non-contact techniques. Early systems used a single stripe or spot of laser light to measure a small part of the object in each scan. Now the availability of controlled light output from LCD projectors allows the projection of a more complex pattern of light to increase the area measured in a single instantaneous scan.

The classic single stripe scanning system [Ber92a] provides the profile of one "slice" through the target object. In order to build a model of the complete surface a number of spatially related profiles must be scanned. To achieve this a sequence of scans is captured. For each scan, the target object is moved in relation to the scanner, or the projected stripe moves in relation to the object, the movement being controlled to the same resolution as required by the scanning system. A system may require an accuracy of 1:20000 [Lev00a].

To avoid the need for accurate mechanisms and in order to speed up the acquisition process, a number of stripes can be projected at the same time and captured as a sequence of stripes in a single frame. However, it may be difficult to determine which captured stripe corresponds to which projected stripe, when we attempt to index the captured sequence in the same order as the projected sequence. We call this the ***stripe indexing problem.*** For this reason methods have been devised to uniquely mark each stripe, by colour [Roc01a], stripe width [Dal98a] and by a combination of both [Zha02a].

These and other works state the disadvantages of coded structured light: with colour indexing there may be weak or ambiguous reflections from surfaces of a particular colour, and with stripe width variations the resolution is less than for a uniform narrow stripe. This last problem can be addressed by projecting and capturing a succession of overlapping patterns of

Journal of WSCG, Vol.12, No.1-3, ISSN 1213-6972
WSCG'2004, February 2-6, 2003, Plzen, Czech Republic.

differing width [Hal01a] but this means that it is not possible to measure the surface in a single frame. Single frame or "one-shot" capture is desirable because it speeds up the acquisition process, and leads to the possibility of capturing moving surfaces.

Moreover, because of the limits of any colour coding scheme, ambiguities will still exist; and stripe width coding is likely to increase the difficulty of interpreting shape correctly, such as when occlusions occur.

In this work we examine how far the correspondence problem can be solved with uncoded stripes, where the correct order of stripes in the captured image is determined by original algorithmic methods. Each captured frame will therefore provide a complete data model for a patch on the surface of the target object.

In part 2 we describe the system, showing the dependence between the stripe index and the measurement of a surface point, and we define the *common inclination constraint*. In part 3 we define the continuity of stripes and the boundaries between continuous patches, and two algorithms are designed each of which indexes a sequence of corresponding stripes. In part 4 we classify occlusions [Cas02a], to improve the validity of the boundaries, and describe a further connectivity algorithm. In part 5, the index is compared to a template sequence, created by hand using prior knowledge to ensure an exact correspondence with the projected sequence. The results of implementing both algorithms, with and without dealing with occlusions, are presented.

2. DESCRIPTION OF THE SYSTEM

The scanning apparatus is shown in Figure 1, viewed "from above". Evenly spaced, vertical (i.e. parallel to the Y axes) stripes are projected which intersect the X_O axis, and the distance between intersections is W. A point $s = (x,y,z)$ on the surface of the target object reflects a beam (shown dotted) in stripe n through the camera lens at O_L and onto the image plane. This light is sensed at (h,v) in the camera CCD array, measured to an accuracy better than one pixel by the subpixel estimator defined later in this section. The relation between the CCD pixel array and the spatial image plane is given by $x_B = hCF$, $y_B = vCF$, where F is the focal length of the camera and CF is the spatial size of one square pixel. P and D are the distances from the origin $X_0Y_0Z_0$ to the centres of the projector and camera lens respectively. θ is the angle between the Z axes of the camera and projector.

The parameters of s are given by the scanning function $scan(h,v,n) = (x,y,z)$

$$x = Wn - z\frac{Wn}{P} \tag{2.1}$$

$$y = vC(z\cos\theta - D) \tag{2.2}$$

$$z = \frac{DhC + Wn(\cos\theta + hC\sin\theta)}{hC\cos\theta - \sin\theta + \frac{Wn}{P}(\cos\theta + hC\sin\theta)} \tag{2.3}$$

where h and v give the position of the sensing pixel and n is the index of the stripe containing the sensed beam. The constants W, P, D, C and θ are found by calibration. Note that, by construction, P is never equal to zero.

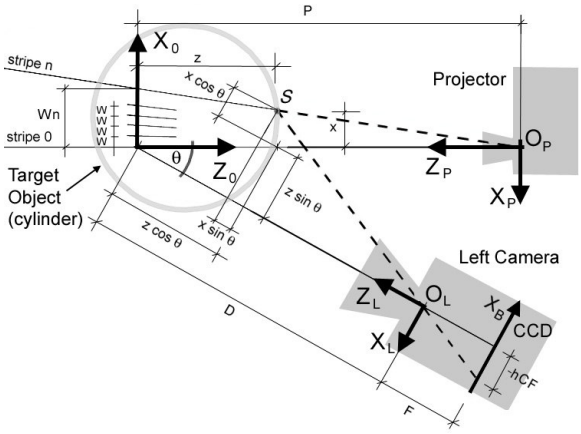

Figure 1. Scanner viewed down the *Y* axes

Hence it can be seen that successful measuring of the target surface is dependent upon correctly determining the index n of each stripe appearing in the image.

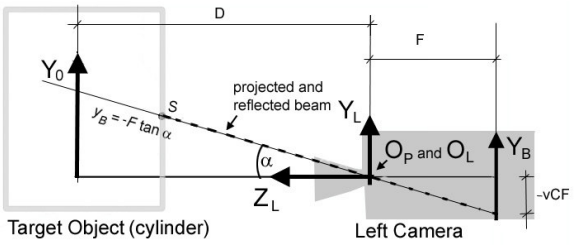

Figure 2. Scanner viewed down the X_L axis

Figure 2 shows the system viewed "sideways", i.e. down the X_L axis. In order that parametric equations 2.1, 2.2 and 2.3 are correctly formulated, the projector and camera are aligned "horizontally", so that axes Z_O, Z_P, Z_L, X_O, X_P and X_L are in the same plane.

The Common Inclination Constraint

The system is further constrained by positioning the projector so that its origin O_P lies on the X_L axis of the camera. Figure 2 shows that O_P and O_L will now both lie on a plane $y_B = -F\tan\alpha$, inclined at angle α to Z_L. A beam projected in this plane to surface point s will be reflected back in the same plane and onto the image plane at "row" $-F\tan\alpha$. If it is assumed that any beam can only be reflected at one surface point, and that any stripe can only contain one beam at a specific inclination, it follows that ***each stripe can be sensed at only one position in any "row" in the image plane***. This is only true when the system is constrained so that the projected and reflected line of a beam always shares a "common inclination", hence the ***common inclination constraint.***

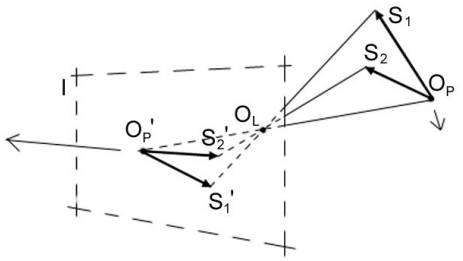

Figure 3. The Common Inclination Constraint

Another view (Figure 3) of this constraint is to consider beams from O_P striking the surface at S_1 and S_2. Lines O_P to S_1 and O_P to S_2 are projected through the lens at O_L and onto the image plane I as lines O_P' to S_1' and O_P' to S_2'. If O_P follows the arrow to where line O_P to O_L is parallel to image plane I, its "image" O_P' will move towards infinity, and all lines O_P' to S_n' will be parallel. Therefore, if all stripes are considered as a bundle of beams emanating from the unique projection point O_P, then it follows that ***every beam path will appear to be parallel when viewed from the image plane***.

Processing the Image Data

The current system uses a standard monochrome video camera synchronised to the PAL standard, at a bandwidth of 5.5 MHz. The data is presented to the system processors as a C x R array A_B of discrete brightness levels where $a_B \in [0, 255]$, and the array is situated in the sensing plane of the camera.

Referring to Figure 4, from the bitmap array A_B where $f(c,r) = a_B$, a peak array A_P is created of local horizontal maxima, i.e. peaks at the centre of each stripe, where

$$a_P = \begin{cases} \text{TRUE if } f(c\text{-}1, r) \le f(c,r) < f(c+1,r) \\ \text{FALSE otherwise} \end{cases} \qquad (2.4)$$

The indexing algorithms then use the peaks array A_P to label each peak with a stripe index array $A_I = C$ x R x N where n is either a stripe index or "NULL" (shown blank). To provide h, v and n for the *scan*() function, a further data type $I = H$ x V x N is created using a subpixel estimator $spe(c,r) = (h,v)$ with a design similar to those used by [Fis96a].

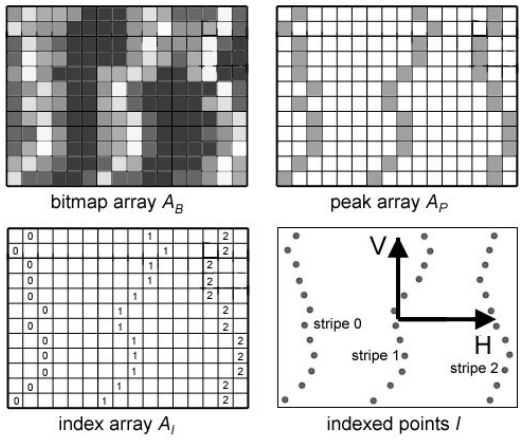

Figure 4. Processing the pixel array

3. STRIPE INDEXING

The discrete bitmap image on the sensing plane is a square tessellation of pixels, each pixel representing a square with four corners and four sides.

Definition 1 *Pixels P and Q are called direct neighbours (d-neighbours) if they share a side, and indirect neighbours (i-neighbours) if they share one and only one corner.*

As all pixels in the captured image are situated in a two dimensional Euclidean plane, then each pixel may have maximally eight neighbours (four direct and four indirect) as seen in Figure 5, at position 4. [Pav82a].

We now refer to the peaks array A_P in which pixels are marked as TRUE if they are peaks (shown grey in Figure 5).

Definition 2 *A northern neighbour (n-neighbour) of a current peak is defined as a d- or i- neighbour in the row incremented by one. A southern neighbour (s-neighbour) of a current peak as a d- or i- neighbour in the row decremented by one.*

Definition 3 *A western neighbour (w-neighbour) is defined as the nearest peak in the same row which has a lesser column index. An eastern neighbour (e-neighbour) is defined as the nearest peak in the same row which has a greater column index.*

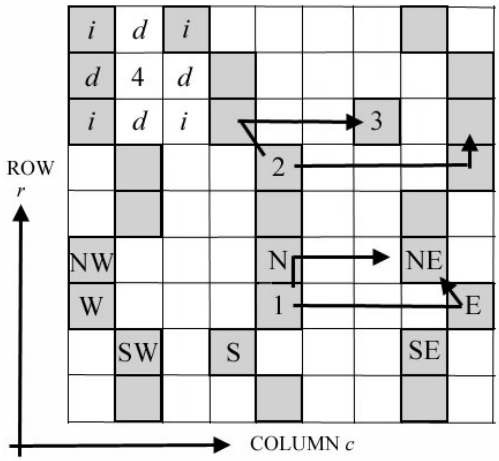

Figure 5: Moving around the peaks data.

The maximum distance to the "nearest" peak will be defined by an estimation of the greatest likely separation of two consecutive stripes. The definition of i-neighbours assumes that the stripe cannot move laterally by more than one pixel as it moves up or down, i.e. that the "slope" of the stripe is never less than 45°. This will not be true in practice but, in common with our general approach it will create extra boundaries rather than wrongly connected stripes. Figure 5 shows, relative to peak 1, its n-neighbour *N,* its s-neighbour *S*, its w-neighbour *W* and its e-neighbour *E.*

Definition 4 *A continuous stripe is a sequence of distinct peaks $\{P_1, P_2, \ldots, P_n\}$ such that each peak in the sequence $\{P_1, P_3, \ldots, P_{n-1}\}$ has one and only one northern neighbour.*

Consequently sequence $\{P_2, P_3, \ldots, P_n\}$ has one and only one southern neighbour.

Definition 5 *A patch is a subset of A_P, maximally possible, i.e. its peaks can be arranged in a maximal number of successive continuous stripes.*

Boundary peaks are created by a set of conditions which are tested at each position in the peaks data. The test assumes that a valid position must have valid adjacent positions. In Figure 5, starting at position 1, position NE can be found by moving north and east, or by moving east and north. Similar tests are performed in the south-east, north-west and south-west directions. From position 2 moving north and east arrives at a different position from moving east and north. Position 2 is therefore marked as a boundary, caused by the disconnected peak at position 3.

In order to index the stripe peaks and mark boundaries, two algorithms have been designed: the *stripe tracer* and the *flood filler*.

The sequence of operations for the stripe tracer, shown in Figure 6 is:

1. Find a start position in the peaks data space. Set the stripe index to zero. The start point can be found by hand or automatically.

2. Trace the current stripe northwards until a boundary condition is met.

3. Return to start and repeat 2 moving southwards.

4. Increase the stripe index, move to the next peak eastwards and repeat 2 and 3.

5. Repeat 4 until boundary condition is met.

6. Return to start and repeat 4 and 5 moving westwards, decreasing the stripe index each time.

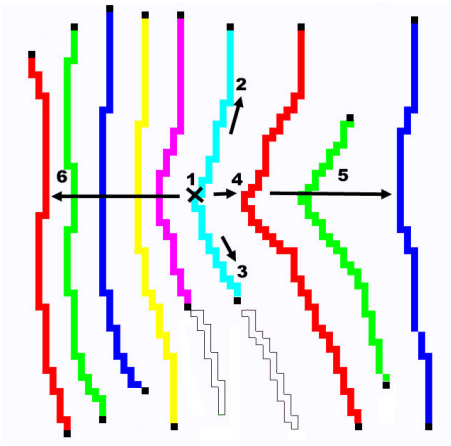

Figure 6: The stripe tracer algorithm.

The disadvantage of this process is that once a boundary has been reached, the stripe will not be "picked up" again, as can be seen from the lower part of the two middle stripes. Therefore some valid stripes in the continuous surface will be lost. To address this problem a second algorithm has been devised to perform a more thorough search for valid stripes: the *flood filler*.

The flood fill recursive function is a classic algorithm used in graphics software, e.g. by [Hil90a]. It has been extensively adapted in this work to pass parameters of `stripe, row, heading` and `index`:

```
flood(stripe, row, heading, index) {
  if(heading==NORTH) goNorth();
  if(heading==SOUTH) goSouth();
  if(heading==EAST) goEast(); index++;
  if(heading==WEST) goWest(); index--;

  if(boundary || alreadyIndexed) return;

  indices[stripe][row] = index;

  flood(stripe, row+1, NORTH, index);
  flood(stripe, row-1, SOUTH, index);
  flood(stripe+1, row, EAST,  index);
  flood(stripe-1, row, WEST,  index);
}
```

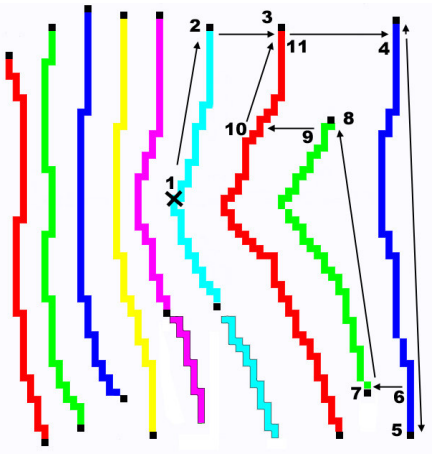

Figure 7. The flood filler algorithm

The *flood filler* will move north if boundary conditions allow, otherwise east, otherwise south, or finally west. If it cannot move in any direction, the current *flood*() function is taken from the stack revealing the parameters of the previous position and this is repeated until a previous position is returned to which has a valid move. In Figure 7 when the algorithm arrives at 5 it cannot move, and peels back until it arrives at previous position 6, whence it can move west to 7. This algorithm will index the stripes missed in Figure 6, *the stripe tracer*.

4. DEALING WITH OCCLUSIONS

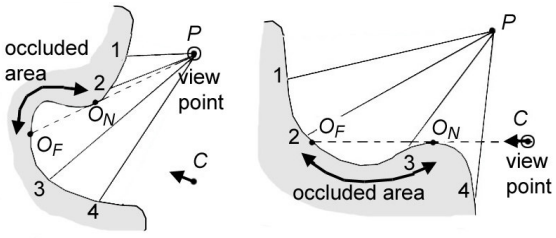

Figure 8. Projector and Camera Occlusions

The tests in Section 5 show that connectivity algorithms produce some indexing errors, typically by connecting peaks which are from different stripes. These errors are often caused by *occlusions*. An occlusion is an obstruction, and in our scanning system it can produce two types: a *projector occlusion* and a *camera occlusion.*

In Figure 8 (left), four stripes radiate from the projector origin at *P,* and illuminate the surface at positions *1, 2, 3* and *4.* In the Figure we draw a dashed line from *P* which grazes the surface at point O_N, and continues to point O_F, where the line intersects the surface at a second point. This line we call a *supporting line,* which for a smooth surface will be tangential. The part of the surface between O_N and O_F is an *occluded area,* shaded from the projection point *P*. We call O_N a *near occlusion point* and O_F a *far occlusion point.* Note that an actual beam can only strike the surface at or close to one of these two points.

Figure 8 (right) again shows four stripes cast onto a surface, and a supporting line from the camera origin *C* to *near occlusion point* O_N and *far occlusion point* O_F. Here the occluded area is that part of the surface which is hidden from the camera view. **Therefore the viewpoint for the camera occlusion is the camera origin, and the viewpoint for the projector occlusion is the projector origin.**

Definition 6. A *near occlusion point* is a point on the surface of the target object, such that a straight line from that point to the viewpoint is supporting to the surface at the *near occlusion point*.

Definition 7. A *far occlusion point* is a point on the surface of the target object, such that a straight line from that point to the viewpoint will support the surface at a *near occlusion point.*

Occlusion boundaries

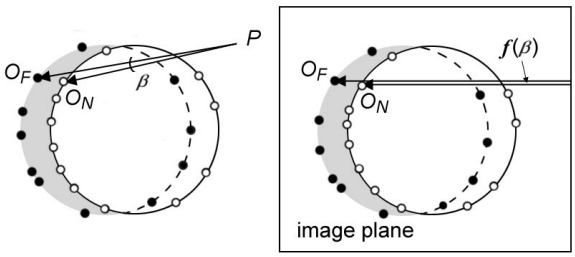

Figure 9. Occluded areas

Figure 9 (left) shows a target object, a circular button protruding from a flat surface, lit by two light beams in the same stripe plane, where the angle β between the beams is very small. One beam strikes the surface at a near occlusion point O_N, denoted by a ring, the other at a far occlusion point O_F, denoted by a black disc. We see an occluded area shaded grey which is defined by a set of near and far occlusion points (rings and disks) lying on the *occlusion boundary.* This shaded area is the result of a projector occlusion, but there is a second occluded area to the right of the dotted line, caused by a camera occlusion, and this hides some of the projected light from the projector view.

Figure 9 (right) shows the same target object, viewed in the image plane of the camera, when the system is set up with the ***Common Inclination Constraint***. We recall that with this constraint the beams appear to run in parallel directions, which means that O_F will be translated laterally from O_N but will be vertically very close (a function of the angle β).

Occlusions of Projected Stripe Patterns

We can extend these observations to look at stripe patterns on a target surface. Figure 10 shows our target object of Figure 10, this time lit by a stripe pattern, indexed from 1 to 7. Position c repeats the situation in Figure 9, where we can assume that two adjacent beams from stripe 2 strike the surface close to an occlusion and are translated laterally. Similar situations occur at b, d and e.

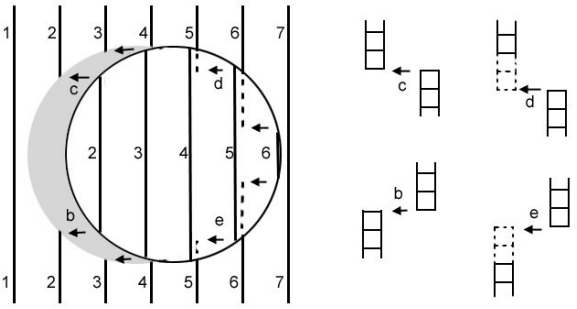

Figure 10. Classifying occlusions

In Figure 10 (right) parts of the object are magnified to pixel level, and four typical situations are classified: at (b) a *low projector occlusion* (l.p.o.), at (c) a *high projector occlusion* (h.p.o.), at (d) a *high camera occlusion* (h.c.o.) and at (e) a *low camera occlusion* (l.c.o.).

Extending Connectivity to Deal with Occlusions

We can now add further rules to our connectivity algorithms, using the cases derived from Figure 10. Firstly, a new data set is created from the peak array A_P, called the *occlusion array,* A_O with elements:

$$a_o = \begin{cases} \text{N if } \neg(p(c,r)) \\ \text{D if } \neg(p(c-1,r-1) \vee p(c,r-1) \vee p(c+1,r-1)) \\ \text{U if } \neg(p(c-1,r+1) \vee p(c,r+1) \vee p(c+1,r+1)) \\ \text{C otherwise} \end{cases} \quad (2.5)$$

Thereby peaks, i.e. TRUE pixels, are labelled as disconnected going up (U), disconnected going down (D), or connected (C). Non peaks are denoted as N.

Figure 11 (left) shows a practical example of connectivity, using A_O and the cases shown in Figure 10. From the D peak at position 1 we look for U peak complying with case b, c, d or e. Note that we can cross fully connected stripes in our search.

In Figure 11 (right) we have found our U peak at position 2, corresponding to a high camera occlusion (case d). The occlusion boundary is drawn as a dotted line connecting the D peak with the U peak. This occlusion line now crosses a seemingly connected stripe, and we therefore know that this is a falsely connected stripe, and that there must be a break approximately in the region of the occlusion line. We therefore mark U and D peaks at positions 3 and 4.

We can see intuitively that a correct indexing should now give us three stripes, where the stripe ending at 1 resumes at 3, and the stripe broken at 4 resumes at 2. The indexing algorithms should find these connections now that the new occlusion boundaries are known, and preliminary results show some success.

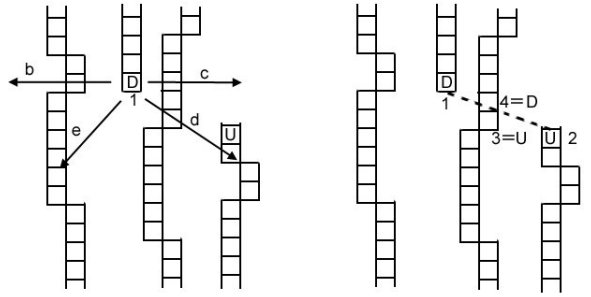

Figure 11. Connecting occlusion points

5. EXPERIMENTAL RESULTS

In this section we present the results of implementation of both algorithms. As a target object we use a sculpted head, 200mm. high. We project 200 black and 200 white stripes into the camera viewing volume, which equates to a spacing of 6 pixels between white peaks. The theoretical limit is 2 pixels, i.e. one white pixel and one black pixel, although this would result in unresolvable spacing at surfaces which are oblique to the camera. The results use a maximal test set within this 6 pixel spacing, and future work will reduce the spacing minimum.

The above procedures will produce patches of surface considered by the system to be valid. To measure the success of the two algorithms we compare a produced data set with a previously created template. To create the template, a copy of the bitmap image is marked by hand so that the system will provide a data set *which is judged to be correct by eye using* ***prior knowledge*** *of the target surface*.

It can be seen from Figure 13 that the *flood filler* (tests 3 and 4) covers a larger patch than the *stripe tracer* (tests 1 and 2). This observation is evaluated in Table 1. The tests show that errors occur due to miscorresponding and noncorresponding peaks.

We recall from Section 2 the array A_I of index values (either a stripe index or "NULL") for a pixel at position (c,r) in the bitmap image. Function $t(c,r)$ gives the index of pixel (c,r) in the template, and $a(c,r)$ the index of the pixel in the same position in the index array produced by the automated algorithms. We then produce the following sets:

- C, "corresponding indexed peaks", i.e. the number of elements where the peaks have the same index. Peak $(c,r) \in C$ such that

$$(t(c,r) = a(c,r)) \wedge (a(c,r) \neq NULL)$$

- M, "miscorresponding indexed peaks", i.e. the number of elements where the peaks have different indices. Peak $(c,r) \in M$ such that

$$(t(c,r) \neq a(c,r)) \wedge (a(c,r) \neq NULL) \wedge (t(c,r) \neq NULL)$$

- X, "noncorresponding indexed peaks", i.e. the number of elements where a false peak is found. Peak $(c,r) \in X$ such that

$$(a(c,r) \neq NULL) \wedge (t(c,r) = NULL)$$

These comparisons are presented in Table 1. It can be seen that the stripe tracer (test 1) gives a smaller total patch (21347 peaks) but with much greater correspondence to the template than for the flood filler (test 3) which covers a greater area (45641 peaks) but with many more differences from the template (5188 + 14333 peaks).

These tests are then repeated (tests 2 and 4) when the occlusion boundaries are added to the algorithm conditions. A common connectivity error is seen in Figure 12 (left), where stripes are shown as connected at A, B and C when, with prior knowledge, we know that a boundary exists, caused by an occlusion. These errors are corrected using occlusion detection whose results are shown in Figure 12 (right) with the "correct" indexing at A, B and C. The "correctness" is measured in the tests tabulated in Table 1.

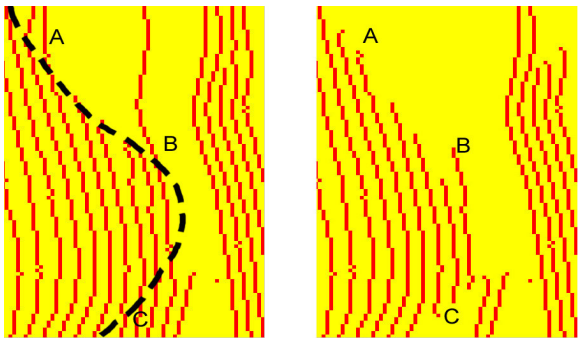

Figure 12. Connectivity errors at A, B and C, corrected at right using occlusion detection.

It can be seen from Table 1 that for the stripe tracer the **mis- and noncorresponding peaks are reduced to zero when occlusion boundaries are included**. In addition, the occlusion boundaries prevent the more pervasive flood fill algorithm from finding noncorresponding peaks (14333 peaks reduced to 31).

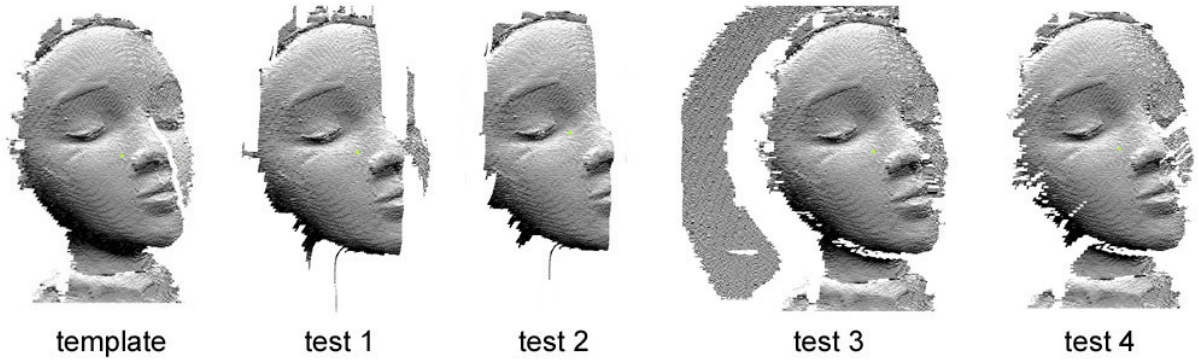

Figure 13. Patches of template and four tests (see Table 1).

Test no:	Occlusion detection	Algorithm type	Corresponding peaks *C*	Miscorresponding peaks *M*	Noncorrespond-ing peaks *X*	Total Peaks *C + M + X*
1	**NO**	**stripe tracer**	**18681**	**2304**	**362**	**21347**
2	*YES*	*stripe tracer*	*17332*	*0*	*0*	*17332*
3	**NO**	**flood filler**	**26120**	**5188**	**14333**	**45641**
4	*YES*	*flood filler*	*25668*	*2175*	*31*	*27874*

Table 1: Numerical analysis of correspondence between template and automatic models, with and without the occlusion boundaries. The starting point is at (0,0).

6. CONCLUSIONS

In this paper we have shown that bounded patches can be created which accurately model the surface of part of a target object, using a uniform stripe scanning system. To achieve this we have defined system constraints such as the *common inclination* constraint, to simplify the indexing algorithms. We have then defined stripe continuity, patches and boundaries on the target surface. We have shown, in the tests against a template, that each of these factors contributes to the successful scanning of the target object.

An open issue is the complexity of the algorithms. The estimated upper bound of the *stripe indexer* algorithm is $O(n^2)$ where n is the maximum number of rows in the bitmap image. This value can be improved. The complexity of the *flood filler* algorithm is under investigation.

A comparison of the flood filler and stripe tracer algorithms shows that greater correspondence can be achieved with an algorithm which has more constrained boundary tests and therefore creates a smaller patch. Our current algorithms follow the principle of ***"greater constraint and smaller patches"***.

The addition of boundaries deduced from the likely position of occlusions has been added to the indexing algorithms, which have further increased the contribution of correctly corresponding peaks, while reducing the overall patch size. These results are important for solving the indexing problem and provide a robust and significant contribution to the creation of accurate patches.

7. REFERENCES

[Ber92a] J.-A. Beraldin, M. Rioux, F. Blais, G. Godin, R. Baribeau, (1992) *Model-based calibration of a range camera*, proceedings of the 11th International Conference on Pattern Recognition: 163-167. The Hague, The Netherlands. August 30-September 3, 1992.

[Cas02a] U. Castellani, S. Livatino, R. B. Fisher, *Improving Environment Modelling by Edge Occlusion Surface Completion*, Proc. Int. Symp. on 3D Data Processing Visualization and Transmission (3DPVT), Padova, Italy, pp 672-675, June 2002.

[Dal98a] Raymond C. Daley and Laurence G. Hassebrook, (1998) Channel capacity model of binary encoded structured light-stripe illumination, in Applied Optics, Vol.37, No 17, 10 June 1998.

[Fis96a] R. B. Fisher and D. K. Naidu (1996) A Comparison of Algorithms for Subpixel Peak Detection, in Sanz (ed.) Advances in Image Processing, Multimedia and Machine Vision, Springer-Verlag, Heidelberg.

[Hal01a] Olaf Hall-Holt and Szymon Rusinkiewicz, (2001) Stripe Boundary Codes for Real-Time Structured-Light Range Scanning of Moving Objects, proceedings of the Eighth International Conference on Computer Vision (ICCV 2001), July 2001.

[Hil90a] F. S. Hill Jr., *Computer Graphics using OpenGL,* Prentice Hall, New Jersey, 1990.

[Lev00a] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk, (2000) *The Digital Michelangelo Project: 3D scanning of large statues* in Computer Graphics (SIGGRAPH 2000 Proceedings).

[Pav82a] Theo Pavlidis, *Algorithms for Graphics and Image Processing,* Springer-Verlag, Berlin-Heidelberg, 1982.

[Roc01a] C. Rocchini, P. Cignoni, C. Montani, P. Pingi and R. Scopigno, (2001) *A low cost 3D scanner based on structured light,* Computer Graphics Forum (Eurographics 2001 Conference Proc.), vol. 20 (3), 2001, pp. 299-308, Manchester, 4-7 September 2001.

[Zha02a] Li Zhang, Brian Curless and Stephen M. Seitz, (2002) *Rapid Shape Acquisition Using Color Structured Light and Multi-pass Dynamic Programming,* 1st International Symposium on 3D data processing, visualization and transmission, Padova, Italy, June 19-22, 2002.