Rapid rendering of apparent contours of implicit surfaces for real-time tracking

Edward Rosten and Tom Drummond
{er258|twd20}@eng.cam.ac.uk
Department of Engineering
University of Cambridge
Cambridge, CB1 2BZ, UK

Abstract

This paper addresses the problem of real-time visual tracking of structures with curved surfaces by localising the apparent contour in each frame of an image sequence. A scheme is presented for rapidly rendering the apparent contour from a predicted pose. Errors between this contour and the observed contour are then used to update the pose estimate for tracking. The rendering algorithm makes use of two contributions. Firstly, a differential equation is derived which traces out the contour generators on an iso-surface of a scalar field. Secondly a set of rules for determining the visibility of each part of the apparent contour is presented. These techniques are used to render structures of moderate complexity in under 30ms which permits real-time tracking at video frame rate.

1 Introduction

A significant proportion of model-based computer vision is concerned with the use of polyhedral models. However, there is a large class of important structures (including humans) which are non-polyhedral in nature. For such structures the principal feature is the apparent contour which is the image of those parts of the surface that are tangent to the viewing ray. This feature is interesting because it moves relative to the structure in a viewpoint dependent manner which can cause problems for some algorithms.

This paper addresses the problem of real-time visual tracking of curved surfaces using the apparent contour. This is an important technology which has applications including gesture tracking for user interfaces. In order to perform these tasks, it is necessary to achieve rapid computation of the apparent contour for a predicted view. This is then compared to the observed contour in order to estimate the error between the predicted and true viewpoints. This paper presents two key contributions which enable rapid rendering. Firstly, a differential equation is derived which is used to trace out the apparent contour generator (being those points on the surface which project onto the apparent contour). Secondly a fast method is presented for breaking the contour generator up into segments and computing the visibility of each segment. Finally, it is shown how this contour may be used with standard edge-based tracking techniques to obtain real-time performance.

1.1 Background

Apparent contours are important when structure and motion problems are applied to non-polyhedral objects. Deformations of the apparent contour have been used to determine camera motion [9] and also the structure of the object under observation[10].

Apparent contours are also important in model-based tracking. One method of tracking involves rendering the apparent contour and then comparing it to the image. [8, 4] use the rendered contour to initiate local searches for image edges, for real-time tracking of polyhedral objects and to some extent curved objects[3]. Another technique is to converge an active contour on moving edges in the image and compare this to the rendered apparent contour[2].

Fua and Plänkers[12, 13] use dense stereo to find the pose and structural parameters of the object. They also note the importance of silhouette (apparent contour) information and integrate this with the dense stereo information by initiating searches based on the predicted position of the object.

Model-based tracking requires a model which must describe the curved surfaces present. The description used affects the accuracy of the model and the speed at which it can be processed. Some of the more popular ways are

1. Collections of curved primitives, such as truncated quadrics[3] and tapered superquadrics[6]. This technique is appealing since these shapes can be dealt with in a computationally efficient manner, which is necessary for real-time tracking. Unfortunately, there are relatively few things in the real world which are accurately modelled by collections of these shapes.

2. Polygonal meshes. Another aproach is to approximate the surface of the object with a polygonal mesh [11] and then treat the object as a polyhedral model[8, 4]. Very large meshes are required in order to accurately capture the shape of complex objects and these can become computationally inefficient.

3. Implicit surfaces (used in this paper). These are defined as the isosurface of a scalar function in $\mathbb{R}^3$. Primitive implicit shapes can be smoothly combined by summing their functions. There are several common kinds or primitives. Radial basis functions are widely used and are attractive from a modelling point of view since they can model smooth surfaces of arbitrary complexity and techniques exist[7] for fitting them to known 3-D data. Metaballs[12] (Gaussians in $\mathbb{R}^3$) are another commonly used primitive. These are essentially a type of non-isotropic radial basis functions. Since the surfaces are not defined explicitly, it can be computationally slow to locate them.

A thorough mathematical analysis of the properties of curved surfaces is given in [1] and [5]. An analysis of apparent contours of implicit shapes under orthographic projection is given in [14].

The current techniques can be split in to two groups, relatively inaccurate models which can be processed efficiently enough to track them in real time and more accurate models which require more processing than can be done in real-time. This paper presents a method for rapidly rendering the apparent contours of implicit shapes so that the accurate models afforded by these can be tracked in real-time. Section 2 derives a differential equation which can be used to rapidly plot an apparent contour in three dimensions. Section 3 describes a set of rules that allow the visibility of apparent contours to be determined by looking at intersections between them, thus reducing the amount of searching needed. Section 4 introduces a set of algorithms which allow rapid searches to be performed on chains of connected line segments, allowing intersections to be calculated very rapidly. Finally, section 5 show how these objects can be tracked.

2 Calculating the Occluding Contour

A 3D shape, S can be defined by a scalar function f of $\mathbb{R}^3$. A point $\mathbf{x}$ lies inside S if $f(\mathbf{x}) > 0$ and on the suface, U, of S if

$$f(\mathbf{x}) = 0. \tag{1}$$

If S is viewed from a camera centred at $\mathbf{c}$, one of the most notable features is the outline, or *visible apparent contour* of the shape. For a point $\mathbf{p}$ on this contour, there is a corresponding point $\mathbf{x}$ on U which projects to $\mathbf{p}$. A ray back projected from $\mathbf{p}$ will be at a tangent to U at $\mathbf{x}$, so

$$(\mathbf{x}-\mathbf{c})\cdot\nabla f(\mathbf{x}) = 0 \tag{2}$$

(since $\nabla f(\mathbf{x})$ is normal to the surface). The points on U satisfying this equation define the *contour generators*, which are curves in $\mathbb{R}^3$. The projection of this curve by the camera at $\mathbf{c}$ is the *apparent contour*. Some parts of it are occluded by S and it is the visible parts which make up the *visible apparent contour*.

Equations 1 and 2 each provide one constraint. By combining these we get a one dimensional set of solutions which describe the contour generators. To solve the equation for $\mathbf{x}$, we start by parameterizing the contour generator with the variable t, such that $\mathbf{x} = \mathbf{x}(t)$. Differentiating Equation 1 with respect to t gives

$$\dot{\mathbf{x}}\cdot\nabla f(\mathbf{x}) = 0. \tag{3}$$

Differentiating Equation 2 with respect to t gives:

$$\dot{\mathbf{x}}\cdot\nabla f(\mathbf{x}) + (\mathbf{x}-\mathbf{c})\cdot\left(\mathscr{H}[f(\mathbf{x})]\,\dot{\mathbf{x}}\right) = 0 \tag{4}$$

where $\mathscr{H}$ is the Hessian operator. Substituting in Equation 3 and writing $H(\mathbf{x}) = \mathscr{H}[f(\mathbf{x})]$ gives

$$(\mathbf{x}-\mathbf{c})\cdot(H(\mathbf{x})\,\dot{\mathbf{x}}) = 0. \tag{5}$$

By transposing, we get

$$\dot{\mathbf{x}}\cdot(H(\mathbf{x})\,(\mathbf{x}-\mathbf{c})) = 0 \tag{6}$$

since the Hessian matrix is symmetric. Equations 3 and 6 give orthogonality constraints on $\dot{\mathbf{x}}$, so $\dot{\mathbf{x}}$ can be defined as

$$\dot{\mathbf{x}} = \alpha\left(H(\mathbf{x})\,(\mathbf{x}-\mathbf{c})\right)\times\nabla f(\mathbf{x}), \tag{7}$$

with α giving an arbitrary scale.

The contour generator is now described by a starting point, $\mathbf{x}(0)$ and a first order differential equation. This can be solved using standard ODE integration techniques and an example is given in Figure 1. The visible apparent contour in B looks quite simple, but the contour generator corresponding to this is quite elaborate (D).

Equation 7 constrains the contour generator to never leave U and so providing it does not become singular[14] and the surface has finite complexity (so the contour is of finite length), it defines closed contours on U.

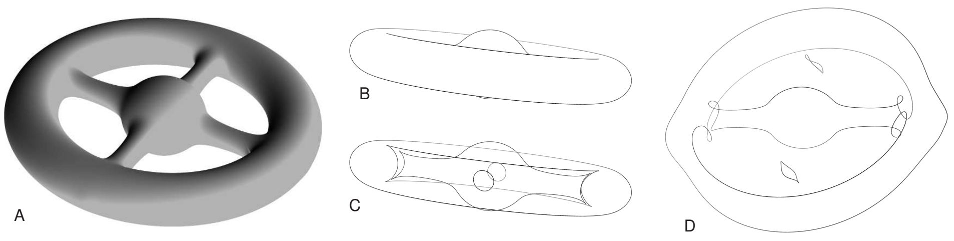

Figure 1: (A) A spoked wheel. (B) The visible apparent contour. (C) The apparent contour. (D) The contour generator from C, viewed slightly from above.

2.1 Calculating the Contour Generator

Equation 7 defines the contour generator for any implicit surface. In this paper, a sum of Gaussians in $\mathbb{R}^3$ are used to define the shape. The implicit function used has the form:

$$g_i(\mathbf{x}) = \beta_i e^{(\mathbf{x}-\mu_i)^\top \mathbf{C}_i(\mathbf{x}-\mu_i)} \tag{8}$$

$$f(\mathbf{x}) = \sum_i g_i(\mathbf{x}) - \frac{1}{2} \tag{9}$$

where $\mathbf{C}_i$ is the inverse covariance matrix for Gaussian i and is a real symmetric matrix. The gradient and Hessian are given by:

$$\nabla f(\mathbf{x}) = -\sum_i \mathbf{C}_i (\mathbf{x}-\mu_i)\, g_i(\mathbf{x}) \tag{10}$$

$$H(\mathbf{x}) = \sum_i \left(2\mathbf{C}_i (\mathbf{x}-\mu_i)^\top (\mathbf{x}-\mu_i)\, \mathbf{C}_i - \mathbf{C}_i\right) g_i(\mathbf{x}). \tag{11}$$

Since $\dot{\mathbf{x}}$ is only defined up to an arbitrary scale, it has been set to be of unit length. The curve was then integrated with a fixed step size fourth order Runge-Kutta solver.

Equation 7 defines any given contour, provided that a suitable starting point has been found. Real-time tracking requires that suitable starting points are found rapidly. The process to achieve this is split in to three stages. The first is an off-line stage which scatters points randomly over the surface of the shape. This is done by scattering points randomly over the surface of each of the individual ellipsoid and then moving the point to the surface of the shape using Newton's method. At run time, points on the surface of the shape close to the contour generator are selected by selecting points where the surface is nearly parallel the viewing ray. An iteration scheme involving a Newton-Raphson step along the ray (to satisfy Equation 2) followed by a step towards the surface is used to move the points to the contour generator. The third step rejects points close to an existing contour using the rapid search techniques described in section 4.

3 Determining the visibility of the apparent contour

Equation 7 yields a set of closed contours. Not all parts of these contours will necessarily be visible to the camera due to occlusion by S. A naïve solution to this would be to

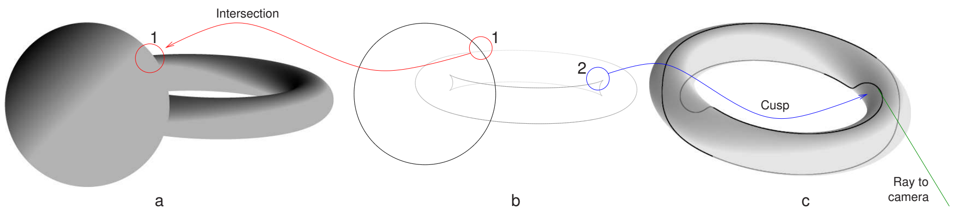

Figure 2: A torus partially occluded by a sphere. (a) rendered image. (b) Apparent contour. (c) Contour generator of the torus rendered from a different angle.

search along the ray from the camera to each point on each contour generator, testing for $f(\mathbf{x}) > 0$, but this is very inefficient.

In this section a technique is presented which allows the visibility to be determined more rapidly. This technique is based on the *occlusion depth* of a point on a contour generator which is defined to be the number of times that a ray from the point to the camera intersects U. This is a powerful approach because the occlusion depth can only change at easily identified points as each contour is traversed. The occlusion depth and hence the visibility of the contour between these points is constant. The visibility of the least occluded part of each contour can then be determined by propagating information between the contours.

Figure 2a shows a sphere occluding part of a torus and the image of the contours generated by Equation 7 is shown in Figure 2b. The occlusion depth can only change in two ways. The most obvious occurs when the contour becomes occluded by another part of S, for example the point labelled 1 in Figure 2b. This can easily be detected since it implies that the contours intersect. A less obvious change in occlusion depth occurs due to a cusp in the apparent contour, Figure 2b, labelled 2. This happens when the viewing ray is coincident with the contour generator. This is shown in Figure 2c. Note that the the contour generator is always continuous even when there is a cusp in the apparent contour.

3.1 Intersections

When two contours intersect in the image, the one with the generator furthest from the camera becomes occluded by part of S. When one part of the contour generator becomes occluded by another part of the shape, a ray to the occluded part must pass in and out of the shape again, i.e. it intersects U *twice*. At an intersection, therefore, the depth of the contour is either incremented by two or decremented by two, depending on the shape of the surface at the foremost contour generator. In order to determine this, the number of intersections between the ray and the foremost surface is calculated as the ray is moved along the rearmost contour. To do this, the ray through the point $\mathbf{x}_1 + \lambda\dot{\mathbf{x}}_2$ can be considered (where $\mathbf{x}_1$ is the point on the foremost surface and $\dot{\mathbf{x}}_2$ is the tangent of the rearmost contour). Intersections between this ray and the foremost surface can then be represented as:

$$f\big((1+\alpha)\left(\mathbf{x}_1 + \lambda\dot{\mathbf{x}}_2\right)\big) = 0, \tag{12}$$

where α parameterizes the position along the ray. Equation 12 can be solved locally at $\mathbf{x}_1$ by performing a Taylor expansion up to second order terms. Since the solution is local, α

and λ are small, so terms in $\alpha\lambda$ can be ignored, leaving a quadratic equation in α:

$$\alpha^2 \mathbf{x}_1{}^\mathsf{T} H \mathbf{x}_1 + 2\alpha\lambda \dot{\mathbf{x}}_2{}^\mathsf{T} H \mathbf{x}_1 + \lambda^2 \dot{\mathbf{x}}_2{}^\mathsf{T} H \dot{\mathbf{x}}_2 + 2\lambda \dot{\mathbf{x}}_2 \cdot \nabla f(\mathbf{x}) = 0, \tag{13}$$

where $H = H(\mathbf{x}_1)$. The number of solutions for α is given by the sign of the discriminant, D, of this equation. At $\lambda = 0$, $D = 0$, confirming that there is a repeated solution, i.e. the ray just brushes U at $\mathbf{x}_1$. Differentiating D with respect to λ gives:

$$d = \left.\frac{\partial D}{\partial \lambda}\right|_{\lambda=0} = -\left(\mathbf{x}_1{}^\mathsf{T} H \mathbf{x}_1\right) \dot{\mathbf{x}}_2 \cdot \nabla f(\mathbf{x}) \,. \tag{14}$$

If this derivative is positive, then going in the direction of $\dot{\mathbf{x}}_2$, D becomes positive, so the ray intersects U twice near $\mathbf{x}_1$. That is if the derivative is positive, then the depth of the rearmost contour increases by two otherwise, the depth decreases by two.

3.2 Cusps

A point, $\mathbf{x}$, on the contour generator can be classified as belonging to *inner surface* or *outer surface*. If at $\mathbf{x}$ the ray to the camera dives in to S then $\mathbf{x}$ is on an inner surface. If the ray moves away from S then $\mathbf{x}$ is on an outer surface. A ray from a point on an inner surface to the camera has an odd number of intersections with U, so the point has an odd occlusion depth is occluded invisible.

When a contour generator goes from being on an outer surface to being on an inner surface, its depth changes by one. At the point where this happens, the contour generator tangent is parallel to the ray to $\mathbf{x}$ as shown in Figure 2c. Motion along the generator at this point results in no motion along the apparent contour in the image, so this appears as a cusp of the apparent contour. At a cusp, if the tangent of the contour generator is pointing away from the camera, the occlusion depth increases by one, otherwise it decreases by one. Essentially, moving away from the camera, the occlusion depth can only increase, and moving towards the camera can only cause a decrease.

3.3 Propagating depth information between contours

Tests for intersections give the relative change in depth of the contour, but cannot determine the minimum overall depth (i.e. whether the least occluded part of the contour can be seen). This can be resolved by propagating information between contours. Rule 1 is an invariant property of the occlusion depths at intersections.

Rule 1 *If contour A intersects contour B such that B becomes occluded, the depth of B just before the occlusion must be greater than or equal to the depth of A at the occlusion.*

Application of Rule 1 does not necessarily completely determine the visibility of all contours. If after application of this rule a contour contains a segment at depth 0 (and hence is potentially visible) but is completely contained within another contour then it may still be occluded. In this case a search must be performed to find its depth. The camera can be searched to find *if* the ray intersects U, rather than to find the *number* of intersections since this is sufficient for determining visibility. The second of these provides more information, but is slower. Rule 1 must be reapplied if the segment is found to be occluded, since it is an invariant property of the system. After all segments requiring a search have been tested, then the visibility of every part of the apparent contour is known.

Algorithm 1 Fast proximity detection.

```
bool Proximity(Tree node T, Point p, Distance d)
    if(p is within d if the bounding box of T)
        if(T is a leaf node AND p is within d of the line segment in T)
            return true
        else
            return Proximity(T.left, p, d) OR Proximity(T.right, p, d)
    return false
end
```

Algorithm 2 Fast piecewise linear contour intersection.

```
form an empty list of pairs of line segments, L

Intersect(Tree Node m, Tree Node n)
    if(bounding boxes of m and n intersect)
        if(m and n are leaf nodes)
            if(the line segments in m and n intersect)
                add (m, n) to L
        else
            call Intersect on all pairs of children
end
```

4 Fast Contour Techniques

In the process of rendering the visible contour, there are three operations which need to be performed rapidly on the contours and contour generators. These are testing the proximity of a point to a contour generator, determining if a point is inside a contour and finding all intersections between contours. The two and three dimensional contours consist of a loop of connected line segments. The ends of the line segments are at the points generated by integrating Equation 7. If the step size needed for accurate integration is small, then the contours can consist of a large number of line segments. As a result, a simplistic implementation of the tests runs very slowly.

A fast method of performing these tests has been developed which makes use of a balanced binary tree. The leaves of this tree contain a line segment and a bounding box. For each node, a bounding box is computed that exactly contains the bounding boxes of the child nodes. To determine if a point is near a contour, Algorithm 1 is used. The worst case execution time is $O(N)$, but the typical running time is $O(\log N)$. Algorithm 2 is used to calculate intersections between contours rapidly. The worst case execution time is $O(N^2)$, since there can be at most $O(N^2)$ intersections, but the typical running time is $O(\log N)$. If a point is inside a polygon, then an infinite ray from that point will cross an odd number of edge segments of that polygon. A similar algorithm is used to perform this test in approximately logarithmic time.

5 Tracking

In order to track the object from an image, the derivatives of visible boundary points (in the image) with respect to the motion parameters must be calculated. As the camera

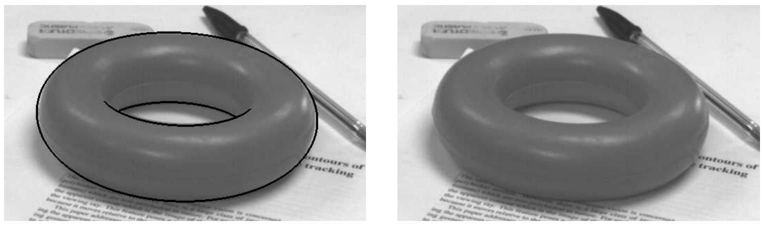

Figure 3: Cusps are very clear in the rendered outline of the torus (left), but in practice, they are very hard to localise in the image (right).

moves, the boundary moves in the image. One component of this motion is due to the motion of the contour generator relative to the camera. The shape of the contour depends on the position of the camera, i.e. as the camera moves, the contour slips across U. This is the second component of the image motion.

The contour generator is by definition at a point where the surface is at a tangent to the viewing ray. Any small motion across the surface will therefore be along the viewing ray and will cause no image motion. More formally, $\mathbf{x}_0$ is a point on the contour generator and $\mathbf{x}_1$ is the point on U which corresponds to $\mathbf{x}_0$ after a small camera motion. A correspondence plane is described by the camera centre, $\mathbf{c}$, the initial point, $\mathbf{x}_0$ and the surface normal, $\nabla f(\mathbf{x}_0)$. The position of $\mathbf{x}_1$ can then be written as

$$\mathbf{x}_1 = \mathbf{x}_0 + a\left(\mathbf{x}_0 - \mathbf{c}\right) + b\left(\nabla f(\mathbf{x}_0)\right). \tag{15}$$

We also know that $\mathbf{x}_1$must lie on U, i.e.

$$f\left(\mathbf{x}_0 + a\left(\mathbf{x}_0 - \mathbf{c}\right) + b\left(\nabla f(\mathbf{x}_0)\right)\right) = 0. \tag{16}$$

Performing a Taylor series expansion of this up to first order terms gives

$$f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)\left(a\left(\mathbf{x}_0 - \mathbf{c}\right) + b\left(\nabla f(\mathbf{x}_0)\right)\right) = 0. \tag{17}$$

Substituting Equation 1 and Equation 2 gives

$$b\left\|\nabla f(\mathbf{x}_0)\right\| = 0 \tag{18}$$

and therefore $b = 0$. Using this correspondence, then, $\mathbf{x}_0$ moves to

$$\mathbf{x}_1 = \mathbf{x}_0 + a(\mathbf{x}_0 - \mathbf{c}). \tag{19}$$

This proves that motion of $\mathbf{x}_0$ is only along a ray to a camera, and therefore produces no image motion of $\mathbf{p}$. For small motions, the visual motion of the contour generator is equivalent to the visual motion of a rigid wire frame which can be tracked using a standard technique [4].

When there is a cusp in the image, it is possible for the motion of the cusp to lie outside the correspondence plane. When this happens, the approximation of the motion is inaccurate, which would make tracking of cusps inaccurate. This point has not been addressed because cusps are very weak image features, as can be seen in Figure 3. Since they are so weak, they are not particularly useful features to track.

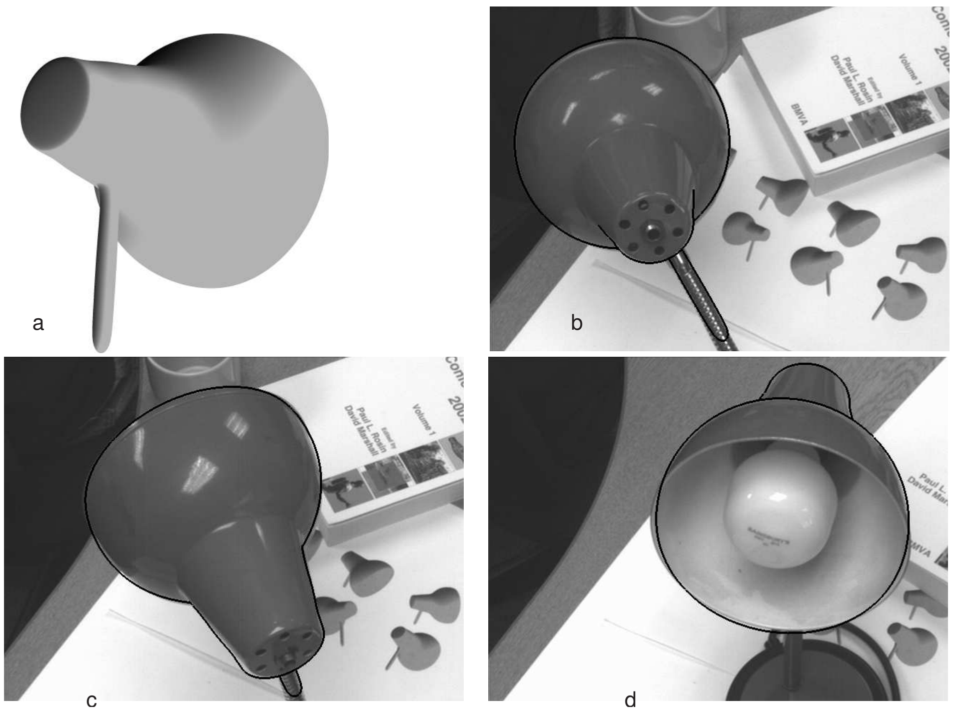

Figure 4: (a) the model of the lamp. (b)–(d) the lamp being tracked in various poses.

5.1 Results

The system was tested by tracking a simple object (a torus) and a more complex object (a desk lamp). The torus can be seen in Figure 3. Figure 4a is of a rendered image of the lamp model. Figure 4b–d show the lamp being tracked in various orientations. The system was fast enough to track the incoming video stream in real-time (25 frames per second). The model of the lamp contains 13 primitives (143 parameters), and approximately 0.025 is required to calculate the visible apparent contour.

6 Conclusion and future work

A method has been presented that allows real time tracking of moderately complex objects modelled with an implicit surface. The system is capable of tracking despite some inaccuracies in the model for example Figure 4d. It is possible to produce very accurate models of shapes using implicit surfaces, but such models can be difficult to construct by hand. Future work will examine the automatic refinement of approximate models.

In calculating the occluding contour, the bulk of the time is spent evaluating the implicit function and its derivatives, since exponentials are computationally expensive. Increasing the complexity causes the computation time to increase for two reasons. First, with more primitives, the function takes longer to calculate at each point. Fortunately, Gaussians also fall away very quickly towards zero and so only have a short range affect.

Future work will reduce the computation required by building a hierarchical model to minimize the number of evaluations needed for each point. Secondly, a complex models have a range of primitives with different sizes and scales. The larger the range of primitives, the larger the range of step sizes required for accurate ODE integration on different parts of the contour generator. Future work can improve efficiency by using more advanced (including variable step) integration techniques, to reduce the number of function evaluations required. These future improvements will allow much more complex models (including even articulated models) to be tracked in real-time.

References

[1] Roberto Cipolla and Peter Giblin. *Visual Motion of Curves and Surfaces*. Cambridge University Press, 2000.

[2] Quentin Delamarre and Olivier D. Faugeras. 3d articulated models and multi-view tracking with silhouettes. In *ICCV (2)*, pages 716–721, 1999.

[3] Tom Drummond and Roberto Cipolla. Real-time tracking of highly articulated structures in the presence of noisy measurements. In *ECCV*, pages 315–320, 2001.

[4] Tom Drummond and Roberto Cipolla. Real-time visual tracking of complex structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):932–946, 2002.

[5] R. B. Fisher, editor. *Design and Application of Curves and Surfaces*. Number 5 in Mathematics of Surfaces. Clarendon Press, 1994.

[6] D.M. Gavrila and L. Davis. 3d model-based tracking of humans in action: A multi-view approach. In *IEEE Computer Vision and Pattern Recognition*, 1996.

[7] Federico Girosi, Michael Jones, and Tomaso Poggio. Priors stabilizers and basis functions: From regularization to radial, tensor and additive splines. Technical Report AIM-1430, 1993.

[8] Chris Harris and Carl Stennett. RAPID, a video rate object tracker. In *Proceedings of British Machine Vision Conference*, pages 73–77, 1990.

[9] Fredrik Kahl and Kalle Åström. Motion estimation in image sequences using the deformation of apparent contours. *IEEE Transactons on Pattern Analysis and Machine Inteligence*, 27(2):114–127, 1999.

[10] Kenneth Wong Kawan Yee and Roberto Cipolla. Structure and motion from silhouettes. In *ICCV*, pages 217–222, 2001.

[11] David G. Lowe. Fitting parameterized 3-d models to images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:441–450, 1991.

[12] Ralf Plaenkers and Pascal Fua. Model-based silhouette extraction for accurate people tracking. *ECCV*, pages 325–339, 2002.

[13] Ralf Plänkers and Pascal Fua. Articulated doft objects for video-based body modeling. In *Proceedings* 8th *ICCV*, 2001.

[14] Gert Vegter and Malgosia Szafraniec. Apparent contours of implicit surfaces. Technical Report ECG-TR-124102-03, ECG, 2002.