Abstract

Interfaces to three-dimensional (3D) virtual environments are restrictive in the way that they take advantage of the potential for 3D human-computer interaction. Restrictive because they are either too complex for users to easily learn, or the range of expressions they permit is too limited. Two application areas where this is a common problem is in computer aided design and gaming, where it is often necessary to manipulate 3D objects within virtual environments. The common approach is to map 2D movements onto 3D actions, but this does not take advantage of the object’s natural 3D interface. Object tracking using computer vision can solve this problem by providing a mapping between real and virtual objects. The proposal is to design and implement an efficient object tracking system based on computer vision techniques that will allow the 3D interface to be better exploited.
Video Tracking for Human-Computer Interface

1 Introduction

A human-computer interface is the way in which the human user and the computer interact with one another. To date, human to computer communication has mostly taken place by means of two-dimensional (2D) interfaces such as a keyboard, joystick or mouse. These devices are fine for most 2D application areas such as word processing but can be limiting when used in such areas as computer aided design (CAD) and gaming, where they are unable to take advantage of the higher dimensionality.

Object tracking is a technology that could potentially solve this problem by lowering the restrictions on user expression and providing virtual functionalities that could be configured online. One implementation of object tracking, known as model fitting, involves “snapping” a model of the tracked object to the video image of that object by aligning prominent features such as lines and edges.

The solution proposed in this paper is to employ the well structured visible features of a Rubik’s Cube [Figure 1] to simplify the model fitting approach of object tracking into an efficient implementation that can be run in the background of a 3D application without significantly degrading the applications performance.

This paper explores the current state of interfaces to 3D applications and then proposes a project that will implement the type of object tracker discussed above.

2 Background

2.1 Human-Computer Interfaces

Human-computer interfaces for interacting with virtual environments can be awkward to use due to there complexity or because they do not properly take advantage of the extra dimensionality. It has been found [1] that given a simple interface, users will usually become proficient quicker than if they were given a powerful interface, where the implications of the interface can be hard to learn. Thus an effective 3D interface must try to maximise power while minimising complexity.

2.2 6D Human-Computer Interfaces

The term “6D” is sometimes used to refer to an interface between a user and a 3D virtual environment, and it comes from the fact that there are six degrees of freedom in human movement: rotations, know as tilt, yaw and roll, and translations about the x, y and z axis.

Some different 6D approaches to the human-computer interface that have been implemented are the SpaceBall, 6D-joystick, 6D-mouse, trackball and DataGlove [1]. The DataGlove provides the most user friendly interface out of all of these as it allows the hand’s pose to be mapped directly into the virtual environment. However, such devices restrict the hand from performing other tasks such as typing and so are inappropriate for many existing 3D applications. Object tracking provides a possible solution to this problem.

By tracking a static objects position and interaction with the user, it may be possible to give that object virtual functions such as ‘click’ or ‘squeeze’. For example a cube could be converted into a virtual gun with the trigger consisting of a squeeze and the guns aim being implicit in the cube’s orientation and position. The directness of such a mapping from real to virtual object would make this interface intuitive and easy to use [1].

There is a useful interface discussed in [1] that could be applied to the idea of virtual objects. It uses a 3D joystick with an engagement button. When the engagement button is pressed, the current 3D position of the joystick is set to be the origin of movement. As the users hand moves away from the origin, the position is defined by the direction and magnitude of the vector spanning from the origin to the hand. As the hand moves further from the origin, the virtual movement accelerates. By repressing the button, the origin is reset to the hands current position. By extending this idea so that the button...
may be held for a short duration to permanently disengage the object tracking, the joystick could be “walked” across the 3D input space, just like a conventional mouse can be walked across its mat.

By applying the above model to the virtual object idea, what you would potentially have, is a flexible “3D mouse”, who’s virtual functions, such as clicks and squeezes, could be dynamically adjusted to match the immediate needs of the user, and the application.

2.3 Object Recognition

Object recognition is the process of locating and labelling elements of a 2D image of a scene that belong to objects within the scene [2]. It can be approached by either using the syntactic method, where only local features such as colour are considered, or the semantic method, in which global data such as object shape are used to classify features in the image. In general semantic methods are used for object recognition, as they are more robust in handling such things as partial object occlusion [3]. Object recognition can be useful in the calibration step of object tracking systems where it may be used to provide the system with the initial position and pose of an object.

2.3.1 The Hypothesis Approach

One of the semantic methods involves using models that are actually sub-models corresponding to various features of an object. In this method, features such as points and lines are extracted from the image and then matched to the feature models stored in a database. These matches are treat as hypotheses, which are then verified using other image information before being ranked [4].

One way of approaching this task is to let the hypotheses be either a pair of edges or a trihedral corner. Hypotheses are verified by noting that each corresponds to a part of the larger object model. Thus if two hypothesis about a single object feature are correct, they will both indicate that the feature is in the same pose. By assuming a particular hypothesis is correct, the other ‘uncertain’ hypotheses can be considered to either support or reject the correct hypothesis. From this, sets of hypotheses that support one another can be filtered out. The poses corresponding to the best sets can then be compared to the image using standard techniques, to find out which one is correct.

The approach to this method mentioned in [4], groups adjacent lines and trihedral corners separately. It then measures the edge lengths and the angle between for the adjacent lines, while it only measures the angles between the lines for trihedral corners. Only edges that were fully visible in an image would be used as hypotheses, while trihedral corners would not have that restriction. This characteristic makes the technique less susceptible to object occlusion since obstructed features will be ignored.

2.3.2 Feature Vector Classification Approach

Another useful semantic approach is known as feature vector classification [2]. This is an established model that relies on vectors of characteristic features of a model, which lie in the scenes feature space. For example colour, area, perimeter, number of features, etc. The steps to this approach involve selecting the appropriate features to measure, deciding how to measure them, and deciding on a policy for distinguishing the objects from others. One simple implementation of this method might take three different types of feature such as colour x, shape y, and size z, each of which specifies a range of possible values. Wherever the three are found together in an image with their values falling within the specified range, the pixels in that region are labelled as being part of the specified object. This method is less intelligent than the hypothesis approach and as a result it is less robust. If simplicity is a priority, feature vector classification may be a useful tool.

2.4 Object Tracking

Unlike object recognition, object tracking does not necessarily have to recognize the object within the scene. Some algorithms are focussed completely on finding the displacement of an object from its previous position and so require an initial calibration step, such as that discussed in section 2.3, in order
Video Tracking for Human-Computer Interface

to find the initial position of the object. It is this kind of tracker that will be the focus of the following section, and the main focus of the research.

One displacement tracking technique, described in [3], uses a 2D-3D approach in which the objects position is tracked in 2D, while its pose is tracked in 3D. The position tracking is based on a 2D affine motion model in which the objects movement is calculated statistically as a normal displacement from the objects contour. Because the predicted displacement is not always correct, a second step is carried out in which intensity gradients are used in an energy function to iteratively find the correct solution.

The affine motion model is performed in two stages. The first computes the normal displacements using the Moving Edges algorithm (ME), while the second uses this information in an extension of the robust multi-resolution estimation technique [3]. The pose finding technique uses a CAD polyhedral model and two techniques described in [3]. The technicalities of the method will not be explored in any more detail here. However, [3] states that this method can handle the situation where faces of the object become hidden and that it is also robust to partial occlusions. This tracking technique is highly appropriate for the research proposed in this paper.

In order to represent the tracked object by the virtual model there is the choice of using a rigid or a flexible model. A rigid model is generally in the shape of the tracked object and can be either fixed in size or parametric [2]. They are the most well suited models for situations in which the object model is exactly specified and the scene is complex, as they effectively constrain the search space and are tolerant to occlusions. When an initial guess for the models shape instance can be supplied and the search space size can be limited, the flexible models are a natural strategy. Like the rigid models, they can work even when the data is incomplete and the scene is complex. However, this strategy can be sensitive to certain types of data and the completeness of the model.

2.5 Summary

The background study has uncovered several key pieces of information relating to the implementation of a lightweight, robust visual object tracking system. Using this information, some implementation decisions, and ideas for future extensions of the proposed research can be drawn. These points are summarised in [Table 1] below, but more should be read about the proposals objectives before consulting the table.

3 Project Proposal

3.1 Requirements

Implementations of 3D object trackers such as the approach described in [3] are apparently robust and simple, being able to track objects in real time, in noisy situations. But these methods still require a certain amount of computational intelligence to make them work, and intelligence means more computational time. The question is: (1), could there be a simpler way to make a tracker work; and (2), by reducing the complexity, is the computational requirement decreased?

3.2 Objectives

The objective of the proposed research is to devise and implement a simple model based object tracking technique, based on the philosophy that after the initialisation stage in which the object’s model is first snapped (aligned) to the object’s image, by performing the re-snapping operation at extremely regular intervals, the search space and intelligence of the algorithm can be kept to an absolute minimum, thus reducing computational requirements.

The obvious question then is: will the increase in processing requirements resulting from the increase in sampling rate actually outweigh the smaller processing requirements of the smaller search space and

4
simpler algorithm? This trade off between sample rate and algorithm complexity will be explored given enough time.

### 3.3 Method

Where possible, the object tracker implementation will be built from the ground up using version 6.0 of the Microsoft Visual C++ programming environment, with the DirectShow features of DirectX 9 being utilized for video capture and display, and the TLIB tracking library for low level object tracking tools [Section 3.4.2]. This will allow the software to be flexibly modified in the future rather than having restriction imposed by having to modify complex open source software.

The implementation will be developed to track the Rubik’s Cube: a well known cube puzzle who’s sides are subdivided each into six small squares of varying colour [Figure 1]. The Rubik’s Cube was chosen both out of novelty and because it is a simple geometric object with well defined lines that could simplify the algorithms development.

The implementation will consist a graphical user interface that will display the live image feed from a video camera and provide the user with feedback about the tracking progress by overlaying an image of the object model over the top of the video image.

The run-time features will be as follows:
When the software is first executed, the user will be prompted to hold the Rubik’s Cube in a certain position in front of the video camera so that it aligns with the overlaid model on the video image. In the initial implementations, the same snap algorithm will be used for this initialisation stage as will be used for subsequent snapping. A more sophisticated initialisation method may be implemented later if time permits [Table 1].

Snapping will be continuously performed at short intervals, so that the object movement does not “outrun” that of the model. Snapping will be performed by searching at right angles from a line on the model for a line occurring close by on the actual cube. In the simple case, where only one side of the cube is visible and the cube is not rotating, by doing this for two virtual lines that are at right angles from one another, this would provide enough information to be able to snap to the actual cube. Once again, more sophisticated techniques will be investigated to handle the more advanced situations [Table 1]. Note that the unique colour configurations on each side of the cube could also be used to assist the tracking process.

<table>
<thead>
<tr>
<th>Decision</th>
<th>Refer to this section</th>
</tr>
</thead>
<tbody>
<tr>
<td>To use a rigid object model to begin with and then replace it with the flexible model if the need arises.</td>
<td>2.4</td>
</tr>
<tr>
<td>To implement a hypothesis approach to object tracking if a simpler technique fails to work.</td>
<td>2.4</td>
</tr>
<tr>
<td>To borrow some of the ideas of modern 2D mice, such as accelerated virtual movement as the object moves away from the origin, to allow more powerful control of virtual object.</td>
<td>2.2</td>
</tr>
<tr>
<td>To use a simple object recognition method to initialise the system by finding the object’s initial pose.</td>
<td>2.3</td>
</tr>
<tr>
<td>A possible extension of the software features would be the implementation of a ‘mouse button’ feature. Where the interaction of the user’s fingers with the 3D object is tracked and a simple gesture such as a tap, triggers a virtual mouse button.</td>
<td>2.2</td>
</tr>
</tbody>
</table>

[Table 1] Implementation decisions and ideas related to section 2.
3.4 Computer Vision Software Libraries

An image-processing library is an important tool in developing object-tracking software as it provides many low-level operation implementations and even some high level implementations that have already been optimised and are well tested and widely used. Although the aim of this research is partly to implement a tracking algorithm from the ground up, there may be aspects that are better implemented using these tools. There are a number of useful real-time image processing libraries around, each with their own merits. A careful choice must be made about which of these tools is appropriate. Below, the two main contenders: Open CV and TLIB are briefly compared.

3.4.1 Open CV

Open CV is Intel’s open source computer vision library. It has a large and powerful set of features, and supports object recognition and tracking. Its features [5] include:
- Fast pixel access macros.
- Contour processing.
- Line and ellipse fitting.
- Image statistics, including mean and Hu moments.
- Image pyramids, e.g. colour/texture segmentation.
- Matrix math.
- Drawing primitives for lines, arcs, etc.
- Utilities for template matching, scaling, and bi-linear interpolation.

Open CV is a large and complex library, with an equally as complex interface [6]. This complexity could inhibit productivity, especially for low level, lightweight, experimental programming. However, the advantages of using this library’s powerful features could outweigh the programming complexity.

3.4.2 TLIB

Developed by the Swiss Federal Institute of Technology (EPFL), TLIB, which stands for ‘tracking library’, is a more lightweight library that also supports real-time object recognition and tracking. Some of its features [6] include:
- C++ classes supporting pixel array processing, image processing and object tracking.
- Edge operators.
- Object extraction operators.
- Template matching operators.
- Format conversion operators.

Unlike Open CV, TLIB provides a simple library with a simple interface. This makes it particularly useful for the development of low level tracking applications.

3.4.3 Summary

For the purposes of this research, where the intension is to build tracking software from the ground up, TLIB appears to be the most appropriate library because it is simple and, presumably, would take a shorter time to learn. Open CV’s more powerful features are not required here, so it will be consulted only if TLIB turns out to be inadequate.

4 Conclusion

This paper has discussed existing human-computer interfaces, their inadequacies for interaction with 3D virtual environments, and some of the methods used to overcome this problem— in particular, computer vision for object tracking.

A method has been suggested by which it may be possible to tweak current object tracking implementation, and the possible advantages have been discussed.
If the objectives of the proposed research are achieved, the result will be an alternative piece of tracking software that does not require great amounts of processing power and that is appropriate for performing further research into the area of 3D object tracking.

5 Bibliography


[4] Shimshoni, I. and Ponce, J., Probabilistic 3D Object Recognition, Department of Computer Science and Beckman Institute, University of Illinois, Urbana, IL 61801, USA.
