Reconfigurable Computing for Augmented Reality

W. Luk, T.K. Lee and J.R. Rice
Department of Computing
Imperial College
180 Queen's Gate
London SW7 2BZ, England

P.Y.K. Cheung
Department of Electrical Engineering
Imperial College
Exhibition Road
London SW7 2BT, England

Abstract

Augmented reality involves combining three-dimensional real and synthetic objects for real-time user interaction. We describe a framework for supporting augmented reality applications by appropriate hardware and software. The benefits of reconfigurable computing, which allows optimised video analysis and synthesis to adapt to environmental changes, are explained using this framework. Our approach is illustrated by video mixing, image extraction, and object tracking. Our designs have been implemented using an FPGA-based platform and run at video rate for image size up to 640 by 480 pixels.

1 Introduction

Augmented reality is a technology for enhancing environmental perception and interaction by combining real and synthetic images in real time [3]. The enhancement may consist of virtual artifacts superposed on a real environment (Figure 4), real objects overlaying on a synthetic background (Figure 6), or a display of non-geometric information about objects in the scene (Figure 9). Applications of augmented reality include surgical planning and medical image visualisation, guidance for manufacturing and repair, path planning in tele-robotics, and special effects for entertainment purposes [3].

Several features of augmented reality motivate the use of reconfigurable hardware. First, augmented reality requires intensive processing since synthetic objects have to be blended with life-video of real objects. In contrast, applications such as virtual reality involve only generating synthetic images. Second, it is well-known that image processing [1] and computer graphics [9] are fertile areas for reconfigurable hardware acceleration; advances made in these two fields should directly benefit a technology requiring their combination. Third, augmented reality applications frequently involve real-time user interaction or adaptation to environmental variations, so the speed and flexibility of run-time reconfigurable processors would have an advantage over fixed-function devices. Finally, augmented reality is developing rapidly and should benefit from reconfigurable platforms on which algorithms, architectures and user interfaces can be explored.

This paper describes a framework for augmented reality research based on a combination of hardware and software. Our framework has been used to support three basic functions for augmented reality applications: video mixing, image extraction, and object tracking. We explain and illustrate the benefits of reconfigurable computing, which enables optimised video analysis of real objects and their combination with synthetic objects, and the adaptation of the system to environmental changes. Prototype designs have been implemented using a low-cost platform based on a Xilinx 6216 FPGA [4]. Previous implementations of augmented reality applications involve workstations [3] or multiple digital signal processors such as the TMS320C40 [7]; we are not aware of comparable work based on reconfigurable computing.

2 Framework

A framework supporting the production of augmented reality applications should be able to handle efficiently high-level representations of real and synthetic objects, and their combination. It should facilitate the investigation of promising techniques, such as run-time reconfigurability, to improve design flexibility and adaptability. The framework should offer a variety of implementations involving both hardware and software, since augmented reality designs often require a range of algorithms and data representations.

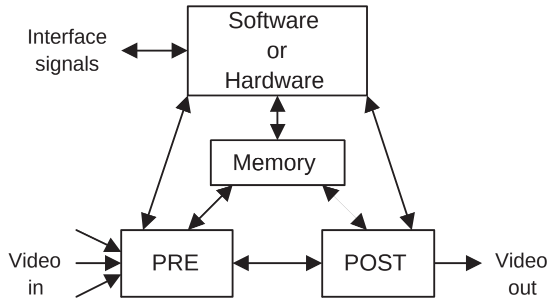

Figure 1 A framework for structuring augmented reality designs, where PRE and POST are hardware elements. Typically PRE performs data analysis such as feature extraction, while POST performs synthetic image generation and mixing of real and synthetic objects.

Our framework provides a guide for structuring augmented reality platforms. It is an extension of a system for video processing [4], and is shown in Figure 1. The hardware elements PRE and POST are for pre-processing and post-processing high-speed data streams, some dedicated to real objects while others contain synthetic scenes. The PRE element accepts one or more video streams as input; its typical task is to carry out low-level, high-speed processing to analyse the input data. PRE has two types of output. The first type of output consists of one or more video streams passed, possibly altered or annotated, to the POST element which generates the appropriate synthetic images and combines them with real objects. The POST element may also further processes the resulting video to achieve the desired effect, such as improved realism, in the output video. Another purpose of PRE is to apply data reduction techniques to information extracted from the input, and to arrange it in an appropriate format for other software or hardware elements. Usually software is used when complex data structures, floating-point or data-dependent computations are involved. Such computations may determine how the system components can be reconfigured to adapt to environmental changes. Other hardware elements may be used to interface to sensors or actuators.

The computational elements described above may contain local memories not shown in Figure 1. They may also have access to a shared memory containing information such as knowledge about real and synthetic objects, models of how they interact, and hardware and software data for system reconfiguration.

This framework covers several possible situations. It covers the case when real objects are captured on video, while synthetic objects are generated within the system; an example is our current platform (Figure 2), described later. The framework also covers the cases when synthetic objects are included in external video streams.

We have identified several opportunities in augmented reality computations that can benefit from run-time reconfiguration. The purpose is to use optimal data representations and operations for various real and synthetic objects in multiple evolving environments. The first two opportunities for reconfiguration involve mainly PRE, while the remaining three involving mainly POST; both may also be supported by software. We explain how these opportunities can be exploited in the following sections. The particular areas that we have identified are given below.

- Analysis of the image sequences of real objects to obtain information such as their size, colour, shape, location and motion. Reconfiguration can be used to provide different analysis procedures at run time to produce optimal results efficiently (Sections 4 and 5).

- Calibration of cameras and other sensors, and tracking of real objects of interest. The change of background scene, noise characteristics, and object shape or location may warrant different video analysis or synthesis procedures which can be installed by reconfiguration (Section 5).
- Generation of different synthetic images or textures. Reconfiguration enables the use of optimal generators depending on the input video, the system state and the application (Section 3).
- Mixing of real and synthetic objects. Different object representations may require different ways of combining them to produce effects such as occlusion (Section 3).
- Techniques for improving realism of the output video, such as texture mapping or production of reflections and shadows. Reconfiguration can be used to provide the appropriate modules to obtain the optimal effect (Section 3).

A further opportunity for reconfiguration is to replace circuits no longer needed by other useful circuits [1]. An example will be given in Section 6.

The results reported in this paper are obtained using a low-cost PCI platform, which has been extended with a video decoder and a video encoder to deal with real-time video [4] (Figure 2). The PRE and POST elements are implemented on the XC6216 FPGA which supports partial run-time reconfiguration. The XC4013 FPGA contains system control circuits and the PCI interface to the PC host. Our hardware designs are developed using the Pebble system [5], and reconfigurable implementations are produced using the *Configdiff* tool [6]. Techniques used for facilitating the development of augmented reality applications will be presented in the next section.

3 Video mixing

A critical requirement for augmented reality is to mix three-dimensional real and synthetic objects at video rate to achieve effects such as occlusion and deformation. We aim to meet this requirement by adopting an object-oriented approach for representing various real and synthetic objects in a uniform and efficient way. This approach improves the flexibility, extensibility and portability of the system by hiding the low-level details. It promotes re-usability by enabling new components to inherit properties and operations of existing components. The image objects can be processed by hardware or software, or a combination of both.

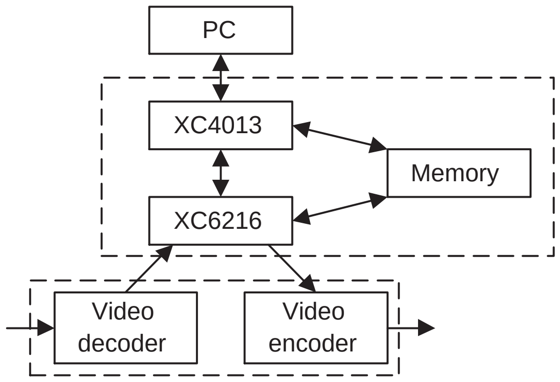

Figure 2 Our FPGA-based platform. The upper dotted box is a commercially available, low-cost FPGA board; the lower dotted box contains a video decoder and a video encoder interfaced to the user-programmable FPGA, the XC6216.

The attributes for image objects include their size, colour, shape, degree of transparency, orientation, location and motion parameters. They are used in defining various image object types, each with a specific set of capabilities. For instance, objects containing images of the background do not require motion attributes which specify how objects move relative to the background. Also since the background image is always opaque, the transparency description is not required.

Object depth can be represented in several ways. One possibility is to assign a depth value to every pixel; this technique has been used in some video mixing methods such as Z-keying [2]. Although relatively simple, this technique requires a

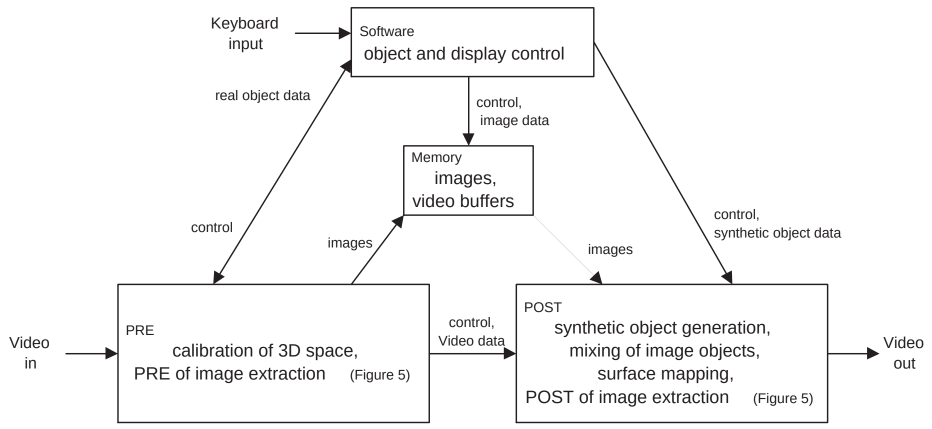
Keyboard input
Software
object and display control
real object data
control, image data
Memory
images, video buffers
control
images
images
control, synthetic object data
Video in
PRE
calibration of 3D space, PRE of image extraction (Figure 5)
control, Video data
POST
synthetic object generation, mixing of image objects, surface mapping, POST of image extraction (Figure 5)
Video out

Figure 3 System organisation for video mixing. Note that PRE passes information about real objects extracted from the input video to software for further processing, while software sends control information, such as reconfiguration data, to both PRE and POST.

large store for the depth map and a large number of computations to determine, for instance, object occlusion. Our current implementation follows an alternative approach, in which an image object is given a single depth value. Composite object descriptions can be used to represent objects with varying thickness.

A typical system organisation for video mixing is shown in Figure 3. Three major tasks are involved. The first is to identify and characterise, in a calibrated three-dimensional space, real objects of interest from the input video; this is performed in the PRE element which contains hardware to implement image extraction, shown as another PRE element in Figure 5. The second task, performed in the POST component, is to generate synthetic objects and surfaces. The third task, also performed in the POST component, is to combine the real and synthetic objects and to perform surface mapping if required.

Our current implementation contains several hardware pipelines for functions such as address generation in controlling object movement, and pattern generation in producing textures and surfaces. These circuits can be reconfigured so that one design can be replaced by another which has a different function or has different trade-offs in speed and area. As an example, a circuit for generating a black-and-white image of a simple, resizable face image requires around 200 cells, or 5% of an XC6216. Producing basic three-dimensional geometric shapes require around 600 cells (15% of an XC6216); these include control for size, motion and clipping. Complex objects can be built by composing, replicating and transforming the basic building blocks.

Software dynamically adjusts object attributes, such as object depth, which are used in occlusion calculations. Since the computational resources on the XC6216 FPGA are memory mapped, object attributes can be stored on registers implemented in the FPGA to facilitate hardware access.

The memory element shown in Figure 3 contains video buffers for synthetic objects and data for surface mapping. It may also contain real or synthetic images of the background used in the image extraction process; this will be explained in Section 4.

Figure 4 shows an example of video mixing. A frame of the input video is shown on the left. The picture on the right shows how this frame is augmented by synthetic images of a couple and a square pattern. Note that image extraction tech-

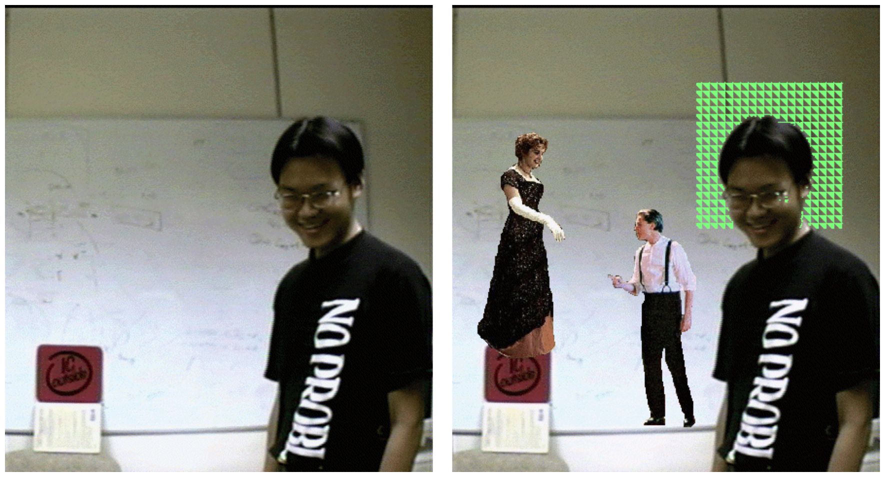

Figure 4 An example showing the effect of mixing real and synthetic objects. A frame of the input video is shown on the left. The picture on the right shows how this frame can be augmented by three synthetic objects: a man, a woman and a square pattern. Note that image extraction techniques are used to distinguish the spectacled person from the background, so that a square pattern can be placed behind him.

niques described in Section 4 are used to distinguish the spectacled person from the background, so that the pattern can be placed behind him.

4 Image extraction

The purpose of image extraction is to produce information about real objects of interest from a video sequence, so that they can be mixed and interacted with synthetic objects. Various image features can be used to guide the extraction process, including size, colour, location, shape, and motion. A number of feature extraction circuits have been developed for our FPGA-based platform [4]: colour detectors and edge detectors are two types of building blocks that can be used in image extraction. Depth detectors can also be used: a reconfigurable engine for stereo vision [8] is one such example.

This section describes a simple method for image extraction which can be used in replacing a real background by a synthetic image. Background replacement is commonly used in news and weather reporting, although in such cases the background is usually provided with a particular colour to facilitate the extraction process: a technique known as chroma-keying.

Our image extractor is based on image differencing with noise compensation. Figure 5 shows the organisation of the image extractor. The PRE element captures the reference image containing the background scene and stores it into the memory. This reference image is then used in the POST component to compare against an input image: if the difference between the corresponding pixels in the incoming image and the reference image is smaller than a given threshold, the incoming pixel is regarded to be the background and will be replaced by the corresponding pixel of another background image. Different threshold values can be used in different parts of the image depending on a noise model implemented in software.

Figure 6 shows the effect of image extraction, coupled with surface mapping and background replacement. A frame of the input video is shown on the left. Two real objects are extracted: a person and a mouse pad to his right labelled 'IC

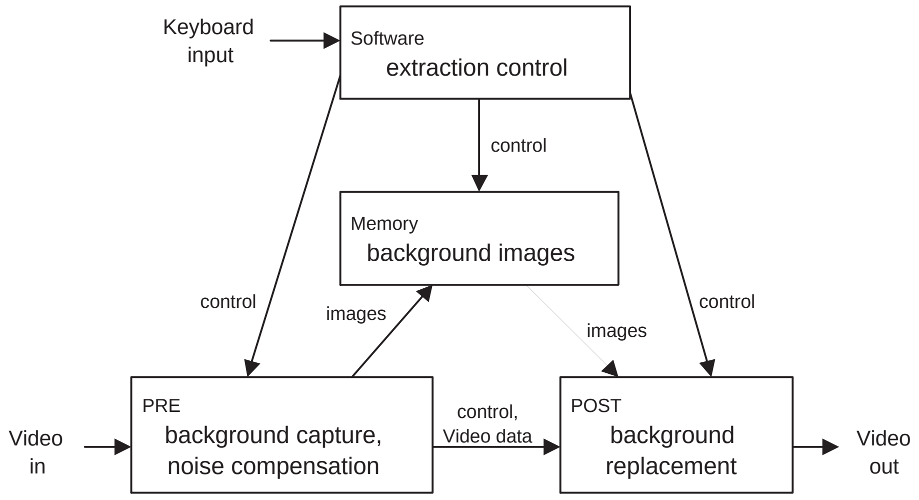

Figure 5 System organisation for image extraction.

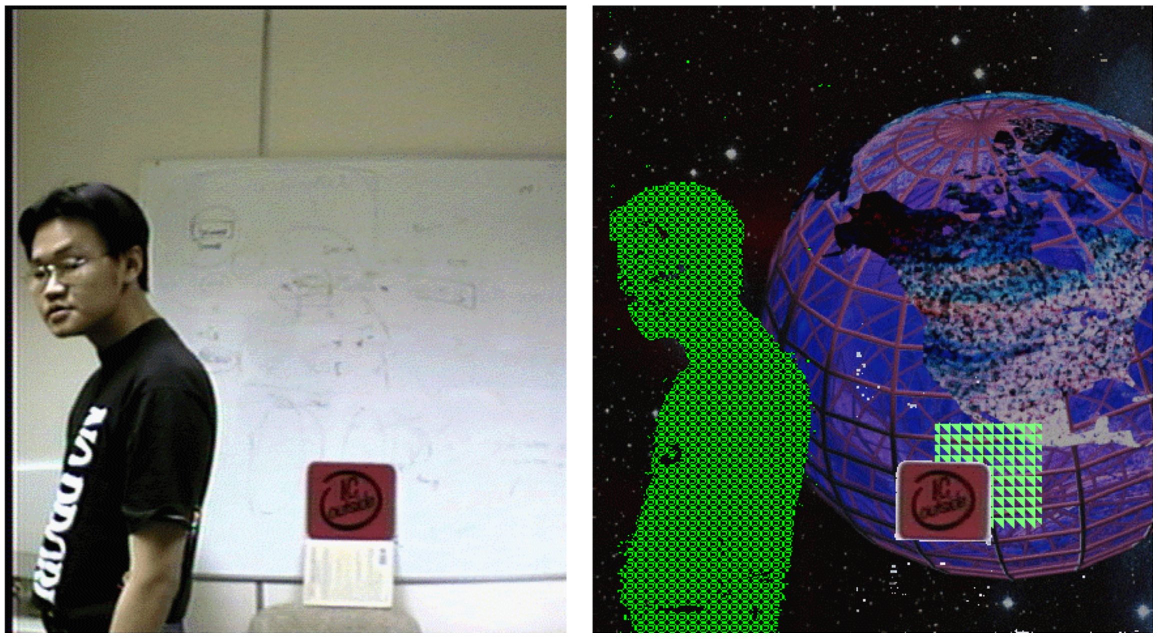

Figure 6 An example showing the effect of image extraction, coupled with surface mapping and background replacement. A frame of the input video is shown on the left. Two real objects are extracted: a person and a mouse pad to his right labelled 'IC outside'. In the corresponding output frame on the right, the real background is replaced by a synthetic image. A surface map is applied to the person, while a synthetic pattern is placed behind the mouse pad.

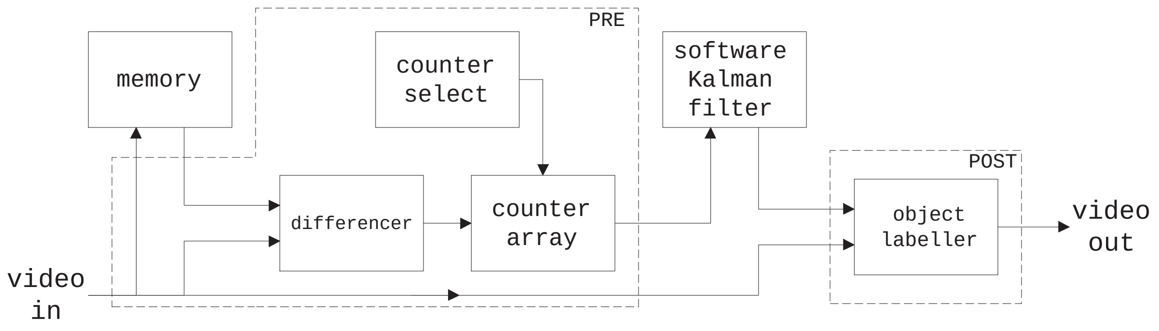

Figure 7 A design for object tracking based on motion detection.

outside'. In the corresponding output frame on the right, the real background is replaced by a synthetic image. A surface map is applied to the person, while a synthetic pattern is placed behind the mouse pad.

5 Object tracking

This section outlines the structure of an object tracker [4] and its use in calibration and in motion-guided effects. The object tracker described here is based on motion detection, and is implemented partly in hardware and partly in software. The PRE component of the hardware part consists of a differencer and a counter array; the POST component labels the detected moving objects in the output video stream. The software part contains a Kalman filter to minimise the effect of noise on tracker performance (Figure 7).

A simple way of identifying motion is to compute the difference between corresponding pixels in two consecutive video frames. The result is thresholded to give a binary value for each pixel. The differenced frame is divided into blocks corresponding to different regions of the image. The amount of motion in each block can then be recorded by a counter; for instance an array of 64 counters will be required for a design with 8 by 8 blocks. A counter selector keeps a record of the current position of the incoming pixels on the screen, and enables the appropriate counter to increment accordingly.

The main function of the PC is to calculate the statistically 'best' estimate of the position, velocity and size of moving objects using a Kalman filter. Our Kalman filter involves mainly floating-point operations and hence is best implemented in software. The post-processing on the FPGA consists of hardware for labelling the detected objects in the video stream, the locations of which are supplied by the PC.

Our object tracker can be used in various ways. It can be used in generating effects related to motion, a topic which will be explained in more detail below. It can also be used to detect camera movement and used in camera calibration, if a reference feature designated to be part of the background is detected to be in motion.

Motion-guided effects are based on the idea that pre-defined motion sequences can be used to trigger corresponding events with observable effects. Motion can be characterised by the following attributes: initial and final locations and the corresponding time stamps, absolute or relative velocity and acceleration, and actions resulting in changes of the environment. A gesture can be decomposed into a series of movements.

Motion-guided effects involve generating effects guided by the movement of objects. There are four major tasks: (a) search for moving objects within each video frame; (b) characterise the moving objects; (c) catalogue each of the identified moving object; (d) perform corresponding operations for effect generation.

A system organisation for producing motion-guided effects using our reconfigurable platform is given in Figure 8. The PRE element has access to the definition of motion and events in a software database. It detects object motion using the object tracker and links movements the corresponding events in the database. The memory stores synthetic image data for the POST element, which contains synthetic image generators and display control circuits. The POST element executes the operations associated with the detected events in generating the output video. Software is used in coordinating the PRE and POST processes as well as implementing the Kalman filter used in the tracker.

Figure 9 left shows a synthetic image whose size, position and pattern are controlled by the motion of the hand. The picture on the right shows an instant when the hand movement triggers an event. The corresponding operation is to surface map a given pattern on all images that are detected black in colour: these include the monitor screen and the object at the bottom right-hand corner.

When prior knowledge about possible observations is available, optimised designs can be used for more effective detection and identification of objects and motion. Such designs, if implemented in hardware, can be placed in the FPGA at the appropriate time using run-time reconfiguration. An example of using reconfiguration to overcome hardware size limitation will be described next.

6 Run-time reconfiguration example

Consider the case when an augmented reality application is required to track a real object, whose motion will be used in parametrising the generation of synthetic images. However, the FPGA that we use is not large enough to accommodate the object tracker and the synthetic object generators at the same time. Run-time reconfiguration is used to overcome this limitation: when the tracking process is completed, the object tracker is reconfigured to become the appropriate synthetic object generators. The process is repeated to emulate the behaviour of an FPGA large enough to accommodate all required circuits.

Figure 10 shows the FPGA floorplans for object tracking (left) and video mixing with on-chip synthetic object generation (right). The shaded components in Figure 10 are common to both designs, and do not need to be reconfigured. They include buffers close to the left and right edges, an image extractor and a pattern generator. The incremental configuration data are produced by the *Configdiff* tool [6].

Our design is driven by a pixel clock of 9.2MHz. It takes $600\mu s$ to reconfigure the object tracker to become the synthetic image generators, and $710\mu s$ to reconfigure the synthetic image generators back to the object tracker. The critical path for the object tracker is 49ns while that for the synthetic image generators is 69ns. The combined design should therefore run at 14.5MHz, and should be capable of processing real-time video of 30 frames per second up to a resolution of 800 by 600 pixels.

7 Summary

This paper presents the use of hardware and reconfigurable computing techniques for augmented reality applications. Let us summarise the main novel aspects of our work. The first aspect is a framework for augmented reality design using a combination of hardware and software. The second aspect is the elucidation of the benefits of reconfigurable computing, particularly run-time reconfiguration, for augmented reality applications. The third aspect is the illustration of our approach using three basic functions: video mixing, image extraction and object tracking. Our current reconfigurable implementations of these functions, using a small FPGA running at 9.2MHz, produce images of resolution up to 640 by 480 pixels at 30 frames per second. In contrast, a 300MHz Pentium II PC can only support a few frames per second.

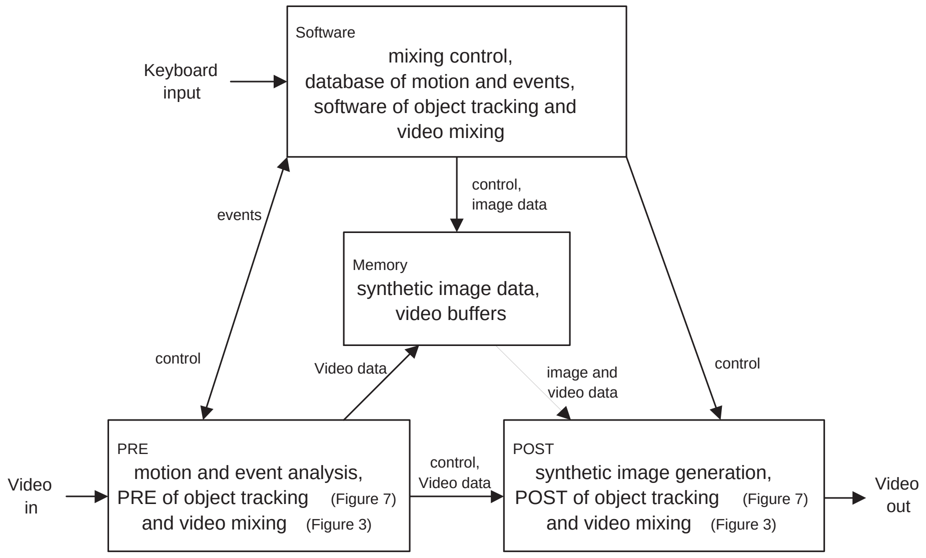
Keyboard input
Software
mixing control, database of motion and events, software of object tracking and video mixing
events
control, image data
Memory
synthetic image data, video buffers
control
Video data
image and video data
control
Video in
PRE
motion and event analysis, PRE of object tracking (Figure 7) and video mixing (Figure 3)
control, Video data
POST
synthetic image generation, POST of object tracking (Figure 7) and video mixing (Figure 3)
Video out

Figure 8 System organisation for motion-guided effects.

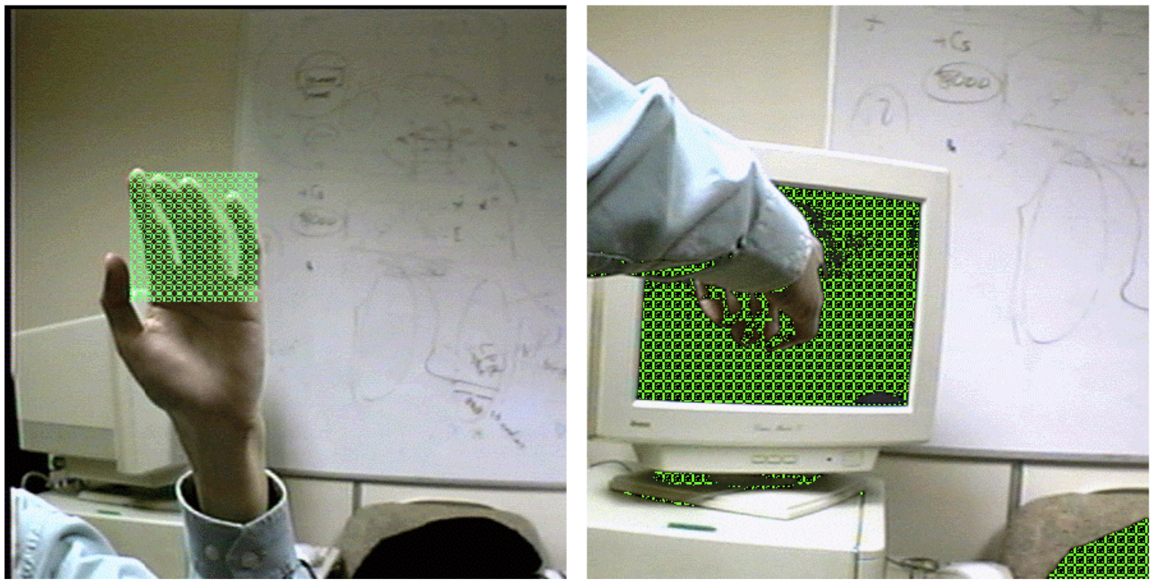

Figure 9 An example of motion-guided effects. The picture on the left shows a synthetic image whose size, position and pattern are controlled by the motion of the hand. The picture on the right shows an instant when the hand movement triggers an event. The corresponding operation is to surface map a given pattern on all images that are detected black in colour: these include the monitor screen and the object at the bottom right-hand corner.

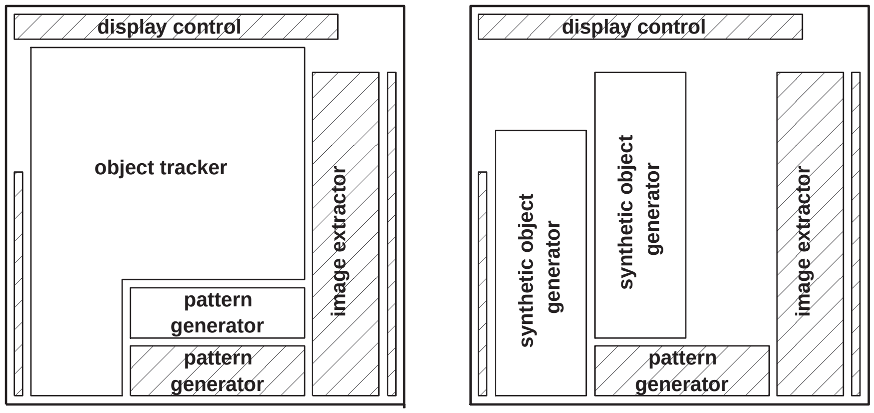

Figure 10 Schematics showing the FPGA floorplans for object tracking (left) and video mixing with on-chip synthetic object generation (right). The shaded components are common to both designs and do not need to be reconfigured. They include buffers close to the left and right edges, an image extractor and a pattern generator.

Current and future work includes refining the tools and applications, and exploring the adaptation of our framework for interactive development and run-time synthesis of augmented reality designs. The incorporation of powerful computer vision techniques, such as the use of stereo vision for depth measurement [8], is also of interest.

Acknowledgements

Many thanks to Simon Haynes and Nabeel Shirazi for their comments and discussions. The support of the Croucher Foundation, the UK Overseas Research Student Award Scheme, the UK Engineering and Physical Sciences Research Council (Grant number GR/24366, GR/54356 and GR/59658), and Xilinx Development Corporation is gratefully acknowledged.

References

[1] R.D. Hudson, D.I. Lehn and P.M. Athanas, "A run-time reconfigurable engine for image interpolation", *Proc. FCCM98*, IEEE Computer Society Press, 1998, pp. 88–95.

[2] T. Kanade et. al., "Video-rate Z keying: a new method for merging images", Technical Report CMU-RI-TR-95-38, The Robotics Institute, Carnegie Mellon University, December 1995.

[3] G.J. Klinker et. al., "Confluence of computer vision and interactive graphics for augmented reality", *Presence: Teleoperators and Virtual Environments*, 6(4), 1997, pp. 433–451.

[4] W. Luk et. al., "A reconfigurable engine for real-time video processing", in *Field-Programmable Logic and Applications*, LNCS 1482, Springer, 1998, pp. 169–178.

[5] W. Luk and S. McKeever, "Pebble: a language for parametrised and reconfigurable hardware design", in *Field-Programmable Logic and Applications*, LNCS 1482, Springer, 1998, pp. 9–18.

[6] W. Luk, N. Shirazi and P.Y.K. Cheung, "Compilation tools for run-time reconfigurable designs", *Proc. FCCM97*, IEEE Computer Society Press, 1997, pp. 56–65.

[7] M. Uenohara and T. Kanade, "Vision-based object registration for real-time image overlay", *Proc. CVRMed95*, 1995, pp. 13–22.

[8] J. Woodfill and B. Von Herzen, "Real-time stereo vision on the PARTS reconfigurable computer", *Proc. FCCM97*, IEEE Computer Society Press, 1997, pp. 201–210.

[9] A.G. Ye and D.M. Lewis, "Procedural texture mapping on FPGAs", *Proc. FPGA99*, ACM Press, 1999.