Preface

Lin Chao
Editor
Intel Technology Journal

This Q2'98 issue of the *Intel Technology Journal* focuses on failure analysis testing methodologies used at Intel and software-based research in computer vision, floating-point operations, and H.323 set of standards for multimedia communications.

Failure analysis is one of the key competencies in Intel and is essential to Intel's steep product ramp and high-volume manufacturing. The first paper in this issue describes failure analysis techniques used to detect and localize flaws in the silicon manufacturing process.

Intel actively pursues technologies that expand computing and telecommunication capabilities in PCs. The other papers in this issue describe new software-based research that contributes to this expansion.

The second paper is on computer vision. Helping computers to "see" is just one aspect of broader research into a "perceptual interface" for PCs where computers can "speak", "sense" touch, and in the case of this paper, "see." This paper describes the development of a 4-degree of freedom color object tracker. Included with this paper are three video clips showing the abilities of the head-tracking software.

Floating-point divide, remainder, and square root are three important operations performed by computers today. The third paper describes research into software alternatives to the hardware implementation of floating-point operations. This paper describes some of the general properties of floating-point computations, and proves the IEEE correctness of iterative algorithms that calculate the square root of a floating-point number.

The fourth and fifth papers describe an important industry standard, H.323, for multimedia communication over packet-based networks. The fourth paper, co-authored by the Chair of the H.323 Interoperability Group, presents an overview of the H.323 core components and functionality. IP telephony is an important industry trend, and the role of H.323 procedures in deploying IP telephony are explained.

The fifth paper presents a characterization of video and audio traffic transported over the Internet by video conferencing applications following the H.323 standards. This paper looks at the multimedia traffic sources of a H.323 terminal. Issues such as packet format and multiplexing of audio and video frames at the host are studied.

An Overview of Advanced Failure Analysis Techniques for Pentium® and Pentium® Pro Microprocessors

Yeoh Eng Hong, Intel Penang Microprocessor Failure Analysis Department, Malaysia
Lim Seong Leong, Intel Penang Microprocessor Failure Analysis Department, Malaysia
Wong Yik Choong, Intel Penang Microprocessor Failure Analysis Department, Malaysia
Lock Choon Hou, Intel Penang Microprocessor Failure Analysis Department, Malaysia
Mahmud Adnan, Intel Penang Microprocessor Failure Analysis Department, Malaysia

Index words: failure analysis, DFT, DFFA, e-beam, DPM, RTL

Abstract

Failure analysis (FA) is one of the key competencies in Intel. It enables very rapid achievement of world class manufacturing standards, resulting in excellent microprocessor time-to-market performance. This paper discusses the evolution of FA techniques from one generation of microprocessors to another.

According to Moore's law, transistor count doubles as transistor dimensions are reduced in half every 18 months, allowing for more complex microprocessor architecture designs. For example the Intel486DX™ microprocessor had 1.2 million transistors while the Pentium® microprocessor contains 3.1 million transistors. With rapid technological advances such as more complex microprocessor architecture, an increasing number of interconnect layers, and flip-chip packaging technology for products like the Pentium® and Pentium® II microprocessors, conventional FA techniques, in use since the Intel386DX™ processor generation, are no longer effective. These conventional techniques require in-depth knowledge of the processor's architecture, and they involve exhaustive e-beam probing work, which typically results in very long FA throughput times. Other traditional defect localization techniques, such as emission microscopy from the frontside of the die, are also becoming less successful due to the increased number of metal interconnect layers that obscure local circuitry.

This paper provides insight into FA techniques that have been adopted at Intel. It discusses the evolution of software fault isolation techniques based on Design For Testability (DFT) features, and other special FA techniques. In this paper, we will discuss these techniques and show how they are effectively used to produce fast FA support turnaround for both silicon debug and manufacturing. We will also review their technical merits and return on investment, as well as the cost of each technique to Intel. The main focus of this paper is electrical fault isolation techniques, as opposed to physical defect localization techniques such as liquid crystal analysis and emission microscopy.

In the context of this paper, Fault Localization and Fault Isolation (FI) are synonymous. These are defined as the task of electrically isolating the location of a defect in logical space. Another approach, termed Defect Localization, refers to the task of isolating the location of a defect in physical space.

Introduction

Failure analysis plays a very important role in the semiconductor industry in enabling timely product time-to-market and world-class manufacturing standards (greater than 95% manufacturing yields, lower than 100 defects per million, or DPM). At Intel, the situation is even more compelling: quick failure analysis turnaround is necessary to support Intel's steep product ramp and high-volume manufacturing, where annual microprocessor production volume is in the range of tens of millions of units.

However, the increasing complexity of microprocessors in the form of more complex architecture designs, shrinking transistor feature sizes, and new packaging technologies have significantly increased the failure analysis challenges on Intel's Pentium and Pentium Pro family of microprocessors. A roadmap recently published by the Semiconductor Industry Association (SIA) predicts that by the year 2000, microprocessor transistor counts will exceed 21 million transistors. The same industry roadmap also predicts that by that time, microprocessors will utilize full flip-chip technology, instead of current wirebond assembly and packaging technology. Traditional fault isolation techniques using intensive e-beam probing and assembly code minimization as well as die frontside defect localization with liquid crystals or emission microscopy are no longer sufficient even for today's complex microprocessors like the Pentium and Pentium Pro microprocessors.

Newer FA techniques based on design-for-testability (DFT) and design-for-failure-analysis (DFFA) features have proven to be highly successful for the Pentium and Pentium Pro microprocessors, as evidenced by high analysis success rates (>90%) and short analysis throughput time. Without these new FA techniques that pinpoint the exact failing location, detection of sub-micron defects such as silicon dislocation, as shown in Figure 1 below, is very difficult if not impossible.

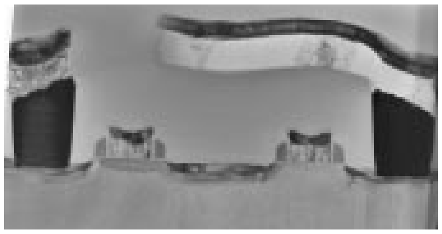

Figure 1: A TEM micrograph of silicon dislocation

This paper provides insight into the various FA techniques based on the DFT and DFFA features that have been successfully developed for the Pentium and Pentium Pro microprocessors. Two other advanced FA techniques and tools will also be discussed. These two techniques are the low-cost personal computer (PC) system-level tester solution and the processor cartridge-level (SECC) FA technique.

Failure Analysis Overview

Figure 2 illustrates a typical FA flow used at Intel. The first step is to verify the failure on a functional tester in the failure analysis lab to ensure that the failure's electrical signature observed on the production tester can be duplicated on the lab's functional tester. Fault localization or defect localization is the next step. Its function is to narrow down the fault to a small block of circuitry. In this case, a small block of circuitry could just be a via, a transistor, a gate, or even a functional unit block (FUB). Reverse engineering, or physical FA as it is better known, is the next step. Individual interconnect layers are selectively removed either mechanically or chemically and examined for defects. Occasionally, defect characterization is introduced to gain more insight into the defect's behavior over time and under stress. This is crucial in order to develop the best possible defect screen to maintain a high quality product. A case is considered closed when a root cause is determined and corrective action is put in place. A corrective action could be a minor change such as the addition of new test screens, or it could also involve major re-engineering via process and design fixes.

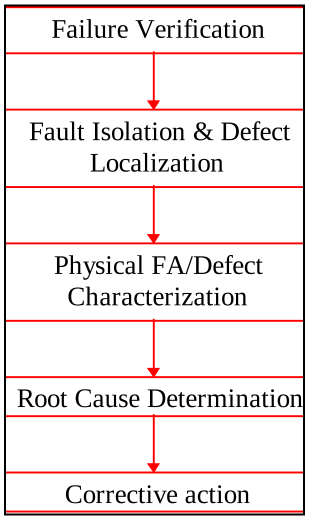
Failure Verification
Fault Isolation & Defect Localization
Physical FA/Defect Characterization
Root Cause Determination
Corrective action

Figure 2: A typical failure analysis flow

From the flow in Figure 2, it is clear that the key to determining the overall FA success rate actually lies in the fault localization and defect localization steps. These steps provide the spatial information on where the defect is, and they control the rest of the steps.

Fault Localization

As the name implies, fault localization is the technique of localizing a fault in a failing circuit functionally and logically. Traditionally, this is a two-step process that involves coarse-level isolation using assembly code minimization followed by fine-level isolation using e-beam probing. With increasing device architecture complexity, coarse-level isolation using assembly code minimization becomes very time consuming and ineffective. Figure 3 below depicts the evolution of the two-step fault localization flows for different generations of Intel's microprocessors.

DFT and DFFA features built into newer microprocessors to accelerate the silicon debug and production testing processes can also be employed as coarse-level fault-localization tools. The use of DFT and DFFA features as FA tools in conjunction with e-beam probing is proven to be very successful on the Intel486DX and Pentium processors where the internal signals of interest are still accessible for e-beam probing. However, the increasing number of interconnect layers (Intel's Pentium® 90/100 MHz processor has four metal layers, and the Pentium® II

333 MHz processor has five metal layers) and changed packaging technologies from frontside wirebond technology to full flip-chip technology makes the e-beam probing process even more challenging. The use of Register Transfer Level (RTL) simulation as a fine-level isolation tool addresses this challenge.

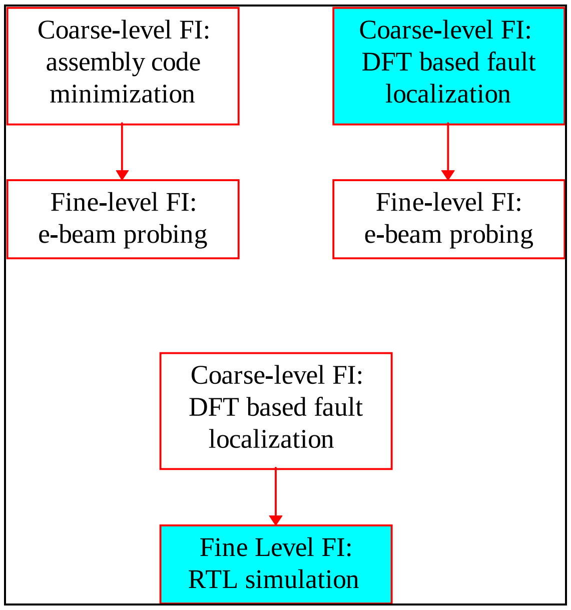
Coarse-level FI: assembly code minimization
Fine-level FI: e-beam probing
Coarse-level FI: DFT based fault localization
Fine-level FI: e-beam probing
Coarse-level FI: DFT based fault localization
Fine Level FI: RTL simulation

Figure 3: An overview of various fault-isolation approaches for the Intel386DX™ processor (top left), the Intel486DX™ and Pentium® processors (top right), and the Pentium® Pro and Pentium II® processors (bottom)

DFT-Based Fault Isolation Techniques

The following sections elaborate on the new fault localization techniques adopted from DFT features currently available on the Pentium and Pentium Pro microprocessors.

Micropatching

Microcodes are integrated instructions that dictate the processor's internal operation. In traditional architecture design, access to built-in microcodes is restricted. However, an ability to create new microcodes and control their flow would be very useful for fault isolation. With such a tool, the failure analyst would be able to create customized microcode subroutines and control microcode flow. This capability is now realized on the Pentium Pro microprocessor family through the introduction of the Micropatching DFT feature. This capability has caused significant breakthroughs in debugging processors with microcode failures. For example, with micropatching, it is now possible to systematically perform fault isolation on failures in the processor's reset subroutine, which is implemented in microcode.

The Micropatching DFT feature consists of two key elements: the microcode patch RAM and several pairs of Match and Destination registers. The microcode patch RAM stores externally programmed microcodes. (From here on, programmed microcodes that reside in the microcode RAM will be called *external* microcodes and built-in microcodes that reside in the microcode ROM will be called *internal* microcodes.)

The Match and Destination registers are used for controlling the microcode flow. Whenever a microcode address in Microcode Instruction Pointer (UIP) matches the content of a Match register, the UIP will be reloaded with a new address from the Destination register.

In order to control the microcode flow, the Match registers are loaded with the UIP that the user intends to jump from. Similarly, the Destination register must be set to the UIP that the user wishes to jump to.

Besides its usefulness in electrical fault isolation, the Micropatching FA tool can also be used in conjunction with liquid crystal analysis and emission microscopy. Without Micropatching, the processor may need to execute many other instructions in the test pattern completely unrelated to the failure before reaching the instruction that causes the failure. By then, the CPU would have generated a lot of heat and made the liquid crystal analysis less sensitive to the heat generated by the failing instruction (i.e., the heat generated by the defective circuitry). With Micropatching, the UIP for the reset subroutine can be set in the Match register to point to the failing UIP, thereby bypassing the reset subroutine altogether. This minimizes the generation of surrounding heat that could result in genuine hot spots, caused by the defect, to be hidden.

Array Dump

Memory arrays are important elements in a processor. Memory arrays are usually used for storing data, control signals, processor status, etc. Visibility into these arrays helps a failure analyst understand the internal operation of the device thereby speeding up the fault isolation process and reducing e-beam probing time. It is almost impossible to analyze a dynamic execution machine such as the Pentium Pro processor without knowing the contents of the arrays. With the Array Dump tool, the contents of many important arrays can be dumped out to produce snapshots of the arrays at any core clock. In order to save data analysis time and avoid human error, a post-processing program was developed to compare the acquired data with that from RTL simulation. Mismatches are automatically highlighted.

The Array Dump FA tool is developed by applying two new DFT features in the Pentium Pro processor family. The first DFT feature, called Micro Breakpoint, is one of the built-in debug trigger-response mechanisms that allows the processor to execute certain tasks in response to preprogrammed trigger events. An external debug breakpoint pin is set up to trigger a processor "micro-breakpoint" event when the pin is asserted. In response to this trigger event, the processor executes a preprogrammed debug command called "All Array Freeze" which is the second key DFT feature that makes Array Dump possible. The "All Array Freeze" feature causes all major memory arrays in the device to freeze their normal execution. The Array Dump feature is capable of acquiring array data at any core cycle relative to a processor's external bus cycle.

Scanout

Accessibility to internal control and datapath signals also greatly enhances FA capability. Scanout is the FA tool developed at Intel for monitoring internal control signals. Scanout has already been successfully and regularly used in Pentium microprocessor failure analysis. One of the Pentium processor's external pins is used to trigger a Test Access Port (TAP) instruction that causes the data from selected control and datapath signals to be latched into Scanout data buffers. The data captured and stored in these buffers are then shifted out serially through the TAP controller's Test Data Output (TDO) pin.

The Scanout FA tool's implementation on the Pentium Pro microprocessor is triggered differently from the Pentium processor's method. This is necessitated by the introduction of odd bus-to-core clock ratios. To overcome this problem, an innovative approach that uses the Micro Breakpoint DFT feature was developed similar to the one in the Array Dump tool. This Micro Breakpoint trigger-response mechanism makes it possible to capture Scanout data on every core cycle of the processor's execution.

A total of around 2000 internal nodes are available for observation. This coverage is much wider than what was available on the original Pentium processor (about 200 Scanout nodes) and newer versions of the Pentium processor family (about 400 Scanout nodes). A program was also written to compare the Scanout data with RTL simulation to aid in interpreting the data. The tool automatically highlights Scanout mismatches.

Memory DAT and LYA Mode

Direct Access Test (DAT) is a special test mode that is specifically intended for manufacturing use. Newer implementations of the Pentium Pro processors contain DAT capability to enhance the testability of most of the major core memory structures. Once DAT mode is enabled, the processor will behave like a pipelined SRAM. It accepts DAT commands every bus clock and returns data for a DAT memory read command to the external output bus a few clocks later.

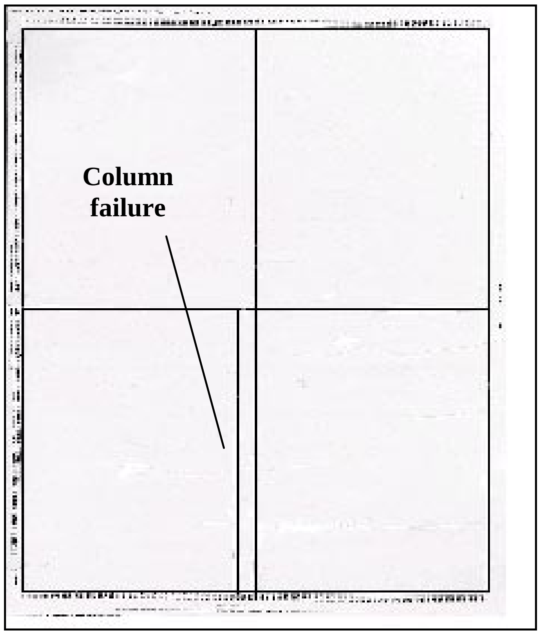

Figure 4: Example of a cache raster bitmap display showing a cache column failure

In DAT mode, the processor only understands DAT instructions. These instructions incorporate the memory array's address information, input data, and array access commands to specify a particular cache operation. The information is directly fed to the array under test. This direct access bypasses the decoding circuitry used in normal operation. Thus, it reduces the amount of effort required to determine whether a cache failure is caused by a problem in the memory array itself or a problem in the supporting circuitry such as the address decoders.

Because only a subset of the external pins are needed in DAT mode, it is very easy to develop a small but effective cache-testing pattern to raster the entire cache line within one memory array. Also, the entire set of advanced cache-testing algorithms used in production testing can be easily written into a small test pattern. If a failure occurs in production, the rastering program can easily provide the failing set and the way and bit information in a bitmap form for failure analysis purposes. Figure 4 above shows an example of the cache rastering program's bitmap display. (In this figure, bits (columns) are arranged horizontally, and cache lines (rows) are arranged vertically.)

However, although DAT mode testing provides the failing set and way and bit information, it provides little information about the failure mechanism. In the latest version of the Pentium Pro processor, a Low Yield Analysis (LYA) mode was added as an enhancement to DAT mode for the largest cache structure.

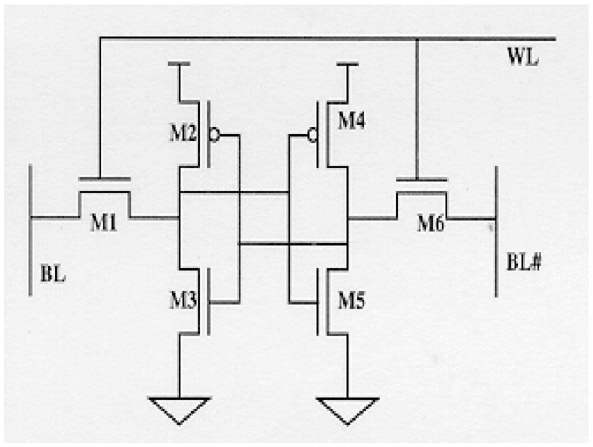

Figure 5: Schematic diagram of a 6-transistor SRAM cell for LYA mode illustration

When LYA mode is enabled, the bitline (BL) and bitline# (BL#) signals of a memory cell are connected to external pins. By having this direct access through the external pins, DC measurements on the memory cell can be carried out easily. LYA mode testing produces cell transistor current readings, as well as cell trip points and other DC measurements. From these LYA "signatures" acquired from a failing device, the failure can be quickly categorized into a few different failure mechanisms, such as open transistor, missing contact, or shorted transistor. With this information, the defect location can be more accurately located, and the physical FA can focus on that location. This results in shorter analysis throughput time and improved success rates in handling cache failures.

Figure 5 above shows a 6-transistor memory cell. When LYA mode is enabled, the BL and BL# signals of this memory cell are directly accessible from the external pins. Enabling the word line select (WL) signal activates this memory cell. In this case, both transistors M1 and M6 are turned on to allow various DC measurements to be performed on this memory cell.

PBIST

Programmable Built-In Self-Test (PBIST) is a memory DFT feature that incorporates all the required test systems into the chip itself. The test systems implemented on-chip are as follows:

- algorithmic address generator
- algorithmic data generator
- program storage unit
- loop control mechanisms

PBIST was originally adopted by large memory chips that have high pin counts and operate at high frequencies, thereby exceeding the capability of production testers. The purpose of PBIST is to avoid developing and buying more sophisticated and very expensive testers.

The interface between PBIST, which is internal to the processor, and the external tester environment is through the standard TAP controller pins. Algorithms and controls are fed into the chip through the TAP controller's Test Data Input (TDI) pin. The final result of the PBIST test is read out through the TDO pin.

PBIST supports the entire algorithmic memory testing requirements imposed by the production testing methodology. In order to support all of the required test algorithms, PBIST must have the capability to store the required programs locally in the device. It must also be able to perform different address generation schemes, different test data pattern generation, looping schemes, and data comparisons.

The program storage structure is used to store the test algorithm. The algorithm includes the types of operations to be performed (e.g., memory read, memory write), the types of address generation modes, the data to be written into and read out from the memory array, and the types of looping schemes.

The address generator is responsible for generating the memory address where the next data are read from or written into. Correct address generation is very important because the physical mapping of the array is always different from its logical mapping. In order to achieve the required test coverage, the address generator needs to be able to generate addresses in different fashions in order to accommodate different kinds of addressing flows such as March-C, Galloping patterns, Address Complements, Fast X, and Fast Y.

The data generator plays a very similar role to the address generator. In order to get the inverse data for each physically adjacent cell, data has to be generated based on the logical-to-physical mapping of the memory array and the data background that is required, such as checkerboard, reverse checkerboard, column stripe, row stripe, and diagonal.

The loop control system is the major sequencing logic in PBIST that allows testing of the entire memory array using only a few program steps. Without the looping control mechanism, test programs of thousands of lines need to be written in order to test an entire array. With

PBIST, testing of the large on-chip memory arrays becomes a lot simpler. Also, PBIST can be used for Test During Burn-In (TDBI) where the processor is tested in the burn-in oven. Other cache-testing methods cannot be used because only the TAP controller pins and a few other control pins are active in the burn-in environment while the rest of the pins are tristated.

RTL Simulation For FA

The motivation for doing fault isolation based on RTL simulation is driven by the need for detailed information about the internal workings of the processor. This information is readily available from the RTL model. Furthermore, as described in the previous sections, the implementation of more sophisticated DFT and debug features in the Pentium, Pentium Pro, and Pentium II generations of microprocessors have helped to promote this technique by providing better observability of the internal signals, thus resulting in less dependency on e-beam probing.

In previous generations of Intel's microprocessors, RTL simulation was used in the failure analysis flow primarily for test pattern generation, test modification, and signal tracing. But the fault isolation process largely depended on extensive e-beam waveform probing work to trace the failure from an architectural starting point, moving upstream until the signals acquired at the inputs of a logic block were correct, while the output was faulty.

Because of the lack of observability of the internal nodes, e-beam probing became the necessary means to gathering the detailed internal signal information of a microprocessor. However, relying on e-beam probing alone has become increasingly difficult or even futile on current generations of microprocessors as more metal interconnect layers are used (obscuring most local signals that run only in Metal1 and Metal2 layers). Additionally, while products are beginning to move into C4 packaging, next-generation waveform probers that allow probing of internal signals through the backside of the die are still not widely available.

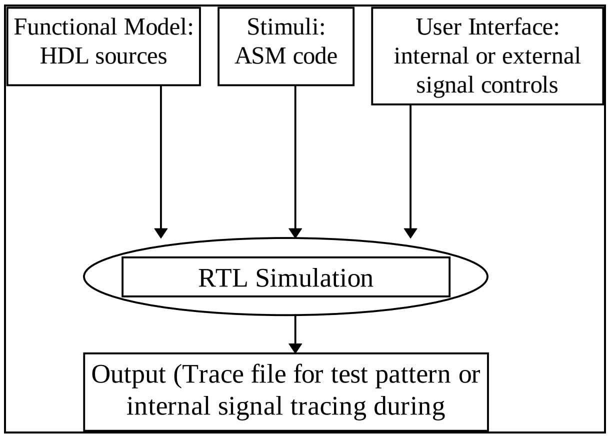
Functional Model: HDL sources
Stimuli: ASM code
User Interface: internal or external signal controls
RTL Simulation
Output (Trace file for test pattern or internal signal tracing during

Figure 6: A high-level block diagram of the RTL model simulation environment

Advances and improvements in today's RTL simulation tools used for Intel microprocessors result in a better and more efficient environment for performing fault isolation than that of the previous generation of RTL simulation tools. More sophisticated capabilities and user-friendly features have been added to the RTL simulation environment, such as more extensive node coverage; the capability to use application programming interface (API) programs; an interactive simulation mode; and reliable simulation state, save, and restore functions. Employing other design tools such as schematic viewers, layout databases, and circuit-level simulation to complement the RTL simulation analysis further helps in the fault isolation process. Figure 6 provides an overview of the key components of the RTL model simulation environment.

The simulation tool's API enables application programs to access signal information from the full chip RTL model of the microprocessor during the model's execution. The application program runs as a separate process and uses routines provided in the API library to obtain signal information from the model. The primary purpose of the API is to isolate the model from user events, enabling the designer (as well as failure analyst) to quickly modify application programs without having to relink the model. The advantage of such an API is its reusability. It is platform independent and can run with several models. For the average user, the API is easier to support than a user event, and it is simpler to write and debug.

A very useful RTL simulation API is the *p6watch* tool, which enables visualization of large amounts of RTL signal values by displaying them in a human format. For example, instead of displaying a set of control signals for a finite state machine (FSM) in binary or hexadecimal format, the user can use p6watch to convert the signals to strings that represent the individual states of the FSM. Another example would be to use p6watch to display the field contents of an array in terms of their decoded or functional meanings.

In this method of performing fault isolation, the failure location is deduced based on extensive analysis of the full-chip RTL model. The failure analyst is also required to have an understanding of the processor’s microarchitecture to aid in interpreting the HDL information and the RTL simulation results, and comparing these data with the behavior observed in the failing device. This fault isolation method is very similar to RTL debugging during the design phase.

The RTL simulator provides crucial information on the expected behavior of the processor. By running the tests that cause the DUT to fail using the RTL simulator, the failure analyst can observe the processor’s pipe stages and see how instructions and data are being transacted and propagated. It is also useful to run passing tests to compare the differences in control and data handling. Among the things to look for at the high level is the flow of data along the datapath blocks and the control signals that arbitrate the machine pipeline. The gathered information will help to determine the logic block that is most likely to have caused the failure.

When the logic block has been isolated, more detailed RTL signal analysis is performed within that logic block. By isolating the failure to a small logic block, the analyst can generate more exhaustive and more focused tests for that block of logic. These new focused tests are then run on the failing device to see how it responds to each test case. For example, if a failure is isolated to a multi-ported memory block, then the focused tests are targeted at all of the available ports to see if there is any specific dependency of the failure on a particular port. Other test cases would involve testing the address generation logic.

Moreover, when running a test it is possible to introduce a “defect” into the model and simulate its effects in terms of the processor’s response to do a “what-if” type of analyses.

E-beam-Less Fault Isolation

E-beam-less fault isolation can produce results faster and reduce analysis by focusing on test data collection, data analysis, and comparison with simulation results. The cost of the analysis is reduced by avoiding the e-beam probing step in the fault isolation process. E-beam probing work involves many hours of sample preparation time (to perform depassivation and prepare probe holes using the FIB) as well as many more hours of waveform acquisition time. Avoiding the e-beam probing step can save a significant amount of analysis time. However, in order to skip e-beam probing work, an alternative method is required to collect valuable information from the failed device. This alternative approach is described in the following paragraphs in this section.

First of all, the failing signature needs to be identified in order to understand why the unit is failing from the standpoint of the processor architecture. Then, DFT tools are used to collect internal node information to further understand the failure mode. Later, RTL simulation is performed to verify the hypothesis made based on the collected data and to identify the failing node. As RTL simulation is usually good enough to anticipate the processor’s internal operation and build a hypothesis about the failure location, probing is not necessary. When compared to the traditional approach where probing is needed to confirm the failing node, which takes an average of 31 days, this e-beam-less approach takes an average of only 15 days throughput time. This new approach has been shown to produce a very accurate estimation of the defect location with greater than 95% success rate on the Pentium and Pentium Pro microprocessors. The high success rate can be attributed to the avoidance of the high-risk steps involved in e-beam probing, especially during sample preparation. This method readily and efficiently supports Intel’s virtual factory concept where fault isolation work is easily shared among all participating factories. By using the FA tools, data collection can be done at the site where the failing device is located. These data are then sent to the site where the product FI expertise resides for in-depth analysis. The predicted defect location is then sent back to the original site for physical FA.

Advanced FI Techniques

The next sections of this paper present the advanced fault isolation techniques used on the Pentium and Pentium Pro microprocessor families. Specifically, these techniques involve the PC FA tool and the SECC FA tool, both of which were developed to address issues that will be discussed in the following sections, and to overcome the challenges and roadblocks described in the introduction.

PC-based FA Tester Platform

With the increasing complexity of Intel’s microprocessor designs, failure analysis TPT would be longer without the adoption of DFT and DFFA features in fault isolation techniques. However, most of the DFT-based fault isolation tools and techniques have been developed for use on expensive functional testers. To reduce the dependency on these expensive lab testers, a new approach has been developed based on the personal computer (PC) platform. The key elements of the PC FA tester platform are shown in Figure 7.

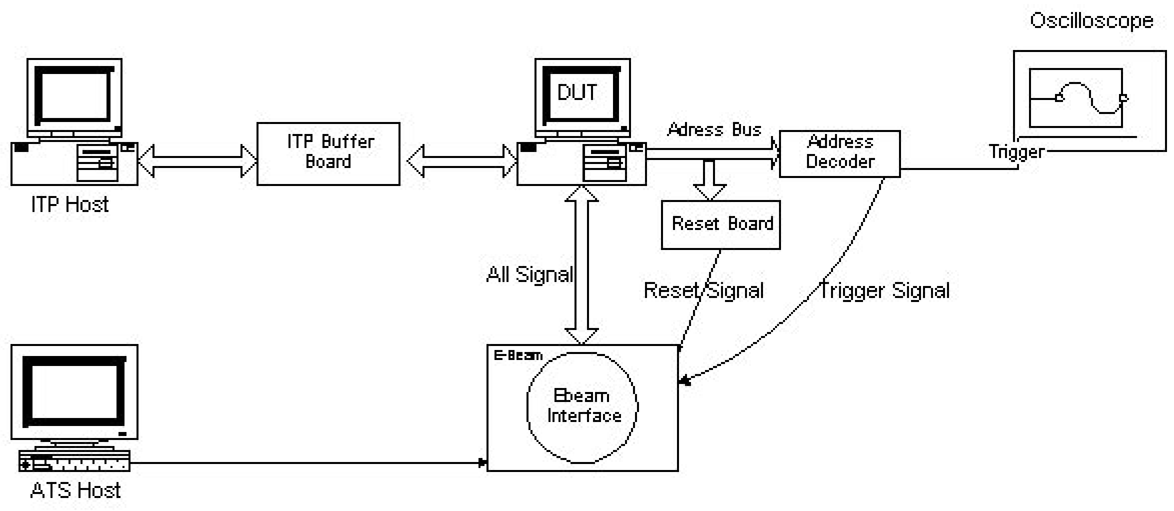

Figure 7: A simplistic block diagram of the PC-based FA platform configuration

The In-Target Probe (ITP) tool has been widely used for system-level debug and validation. On this test platform, a host PC is used to control a target PC where the device under test (DUT) resides. The host PC is used to control the DUT and to upload the test program that the DUT will execute. The ITP is developed around the processor’s *Probe Mode* debug feature and is able to access all programmer-visible and non-visible registers in the DUT. This gives the ITP the necessary control mechanisms to halt the DUT, modify its internal state, and resume normal program execution at any chosen instruction boundary.

In this new approach, the ITP is used as the controller of the target PC where the failing pattern can be uploaded to the target PC through the TAP controller interface. The host-target PC configuration acts like a tester where it drives the necessary data to the input pins of the microprocessor. However, the output pins of the microprocessor are not strobed to compare with expected data. Rather, the output data is monitored by a hardware board that sends out a signal when certain conditions are met. This signal can be used as a trigger signal, such as the e-beam trigger signal, to enable hooking up the DUT to the e-beam, although this has only been proven experimentally.

In order to do e-beam probing, the DUT needs to execute the test program in a loop. To achieve this, another hardware board is used to monitor the activities of the DUT. When a preset condition is met, the hardware board sends out a signal that is used to reset the DUT. Once reset, the DUT restarts execution from the normal boot-up address and reruns the entire test pattern.

FA tools based on the processor’s DFT features have also been implemented on the PC test platform where the host PC is used as an interface to the DUT. Commands to invoke the DFT features are fed into the DUT through the TAP controller interface on the ITP board. The results of the test are dumped to the host PC through the same TAP controller interface.

FA tools that employ the DFT features such as Scanout, Array Dump, and cache rastering have been developed for the PC test platform. To use tools such as Scanout and Array Dump, the DUT is initially halted. The required register for stopping the DUT from execution is programmed through the ITP. The trigger condition for the breakpoint is set, and the desired processor response to the breakpoint is also programmed. Next, the original architectural state is restored, and the DUT is allowed to resume execution from the point where it was halted. When the preprogrammed breakpoint condition is met, the DUT will stop, and as a result of the response to the breakpoint, the data are shifted out through the ITP to the host PC. These data provide information on the internal state of the DUT to enable further fault isolation.

To perform cache rastering on the DUT, a data background for the processor’s internal cache is preset by writing into each memory location. Examples of the patterns typically used for cache rastering include checkerboard and reverse checkerboard patterns. Next, the data in the memory array are read out and compared with the data that were originally written. If there is any discrepancy between the acquired and expected data, the failing memory cell is identified. The PC is a perfect platform for data-retention type of memory testing because all commands and data are shifted serially through the TAP controller. This method takes longer to perform cache rastering as compared to lab functional testers, but it is cheaper, and it frees up the utilization time of the lab testers.

By providing capabilities similar to a lab tester such as providing stimuli to drive the DUT, enabling interface to the e-beam, and supporting various FA tools, the PC test platform can be used for functional FA work, a task which had been performed predominantly by lab testers. The complete test platform is significantly cheaper and more compact than the traditional workstation and tester combination.

A few FA cases have been successfully resolved using this PC test platform. E-beam waveform of a clock signal has been acquired indicating the feasibility of using the PC to replace a tester.

SECC Form-Factor Testing

The concept of processor *bus fractions* was introduced to make it possible to increase the processor's core clock frequency while at the same time maintaining the external bus clock frequency at a lower speed. To support this idea, a frequency multiplier is designed into the processor to multiply the external bus clock to produce a higher frequency clock in the processor core. The term "1:2 bus fraction" refers to a clock configuration in which the internal processor frequency is twice as fast as the speed of the external bus. For example, for an external processor bus clock of 66 MHz, the internal clock frequency is 133 MHz. In this situation, there are two clock domains: one running at 66 MHz and another running at 133 MHz. Extra care is required when developing test programs for products that support bus fractions: input and output data that cross the clock boundaries must be correctly aligned. The Pentium processor only has one bus clock domain and one core clock domain. These two different clock domains are isolated from each other by the processor's input and output buffers. In this case, data alignment is handled by the processor itself, thus reducing the amount of complexity associated with aligning the data in the test pattern.

The Pentium II processor introduces a new bus fraction concept, which increases the complexity of the test pattern development and testing. The Pentium II processor implements two separate sets of bus frequencies: a frontside and a backside. The frontside bus is connected to external components such as chipsets and other peripheral interface devices. Alternatively, the backside bus is a local bus for the Pentium II processor that is connected only to the level-2 (L2) memory. These two buses run at different clock speeds. The backside bus frequency is always equal to or faster than the frontside bus frequency. Together with the internal core clock, there are a total of three different clock domains in this implementation. In this case, the processor's bus fraction configuration is described in terms of all three clock domains relative to each other. The term "1:4:2 bus fraction" refers to a processor with a backside bus running 2x faster than the frontside bus, and a core frequency that is 4x faster than the frontside bus or 2x faster than the backside bus. The data alignment between the frontside bus and the core logic is handled by the frontside input and output buffers. Meanwhile, the data alignment between the backside bus and the core logic is handled by the backside input and output buffers. The user, or the test program developer, is responsible for managing the data alignment between the frontside and backside buses.

For a round numbered (or even) bus fraction configuration between the frontside bus and the backside bus, the data alignment can be easily handled by the test program. This is illustrated in Figures 8 and 9 below.

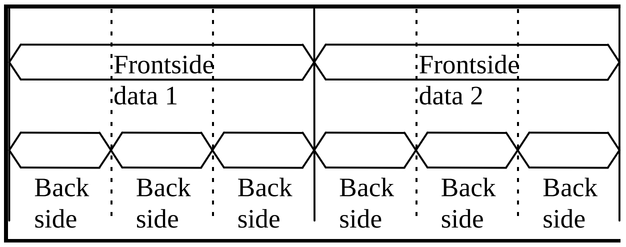

Figure 8: A processor's frontside and backside data switching and alignment for a 1:3 (frontside-to-backside) bus fraction

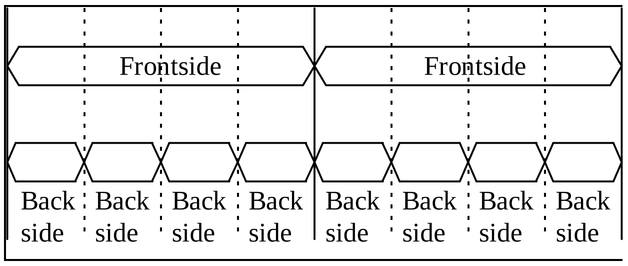

Figure 9: A processor's frontside and backside data switching and alignment for a 1:4 (frontside-to-backside) bus fraction

In the examples shown in both of these figures, frontside data switches at the same time as backside data. This boundary can be used as the alignment point for both the frontside and backside data. However, in a fractional (or odd) bus fraction configuration, data switching and data alignment become more complicated. An example of a fractional frontside-to-backside bus fraction is a 4:7 (frontside to backside) configuration.

In the 4:7 example shown in Figure 10 below, when the frontside data FD1 switches to FD2, the data are not aligned with backside data switching. The same situation occurs during the transitions from FD2 to FD3 and FD3 to FD4.

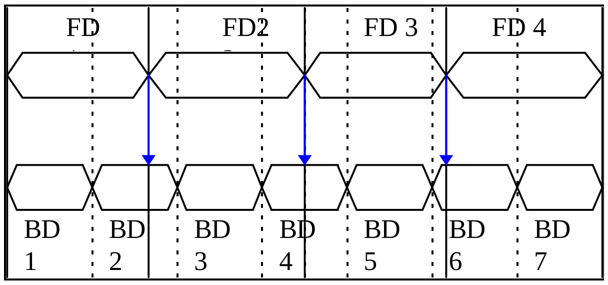

Figure 10: The die frontside and backside data switching and alignment for 4:7 (frontside-to-backside) bus fraction

The functional tester needs to simultaneously drive input data to the processor and strobe for the output data on both the frontside and the backside bus. Advanced production testers are capable of handling these situations because they offer very flexible timing capabilities where the rising and failing edges of the data can be placed anywhere within a tester period. This allows the edge placement of the first cycle to be different from the edge placement of the second cycle. Unfortunately, the failure analysis test platform employed by the FA labs does not support advanced flexible timing capabilities. Therefore, functional testing of a Pentium II processor in a lab environment becomes very difficult.

The fractional bus testing problem is addressed by mounting the Pentium II processor onto a Single Edge Contact Cartridge (SECC) card. The backside bus, which used to be accessible externally, now only communicates with the on-board L2 cache. Only the frontside bus is accessible to the external tester through the edge connector.

In the SECC card form factor testing, the FA tester only needs to handle the data transactions occurring on the frontside bus; the backside bus data transactions are transparent to the test platform. Hence, the complex *frontside:core:backside* bus fraction is reduced to a simple *frontside:core* bus fraction. This setup enables the normal FA functional tester to execute test vectors that use complex bus fractions.

To support SECC testing on a normal FA tester, a pattern conversion tool is needed to convert a pattern from component format into SECC format. This includes extracting the frontside bus data and realigning it into SECC format. This concept has been proven to be very successful in analyzing the latest Pentium II microprocessors.

Future Directions In Failure Analysis Techniques

The advent of more sophisticated DFT and debug features, coupled with faster RTL simulators and better computer-aided fault debugging tools increases the usage of the fault isolation methods described in this paper. The trend towards more extensive use of computer based analysis tools is continuing. With the introduction of new DFT features which provide high degrees of controllability in addition to observability in next generation Intel processors, the concept of performing computer-aided fault diagnosis at the logic or RTL level is taken to the next abstraction level of the design, where fault diagnosis is done at the gate level using intelligent computer-aided fault diagnostic tools. By having both observability and controllability of many important internal signals, the concept of virtual probing will be realized.

Conclusion

As device complexity and interconnect layers increase, new fault isolation techniques need to be continuously developed in order to maintain a high analysis success rate and short throughput time. As discussed in this paper, performing FA using traditional methods has been shown to lower FA success rates. By quickly isolating the defect location and identifying the failure mechanism that is causing low manufacturing yields or abnormal failure rates, problems can be fixed in a short amount of time to improve manufacturing yields. In addition, the corrective actions developed as a result of these FA techniques are documented and proliferated to newer products in the form of design rules. This will prevent the problems from recurring, which in turn enables even a steeper volume ramp. The innovative FA techniques developed for the Pentium and Pentium Pro processors have been one of the key elements in the continued exceptional performance of Intel's product time-to-market and high-volume manufacturing.

References

[1] Yeoh Eng Hong, Martin Tay, "The Application of Novel Failure Analysis Techniques for Advanced Multi-Layered CMOS Devices," *International Test Conference*, 1997.

[2] Adrian Carbine, Derek Feltham, "Pentium ® Pro Processor Design for Test and Debug," *International Test Conference*, 1997.

Authors' Biographies

Yeoh Eng Hong graduated from Monash University in 1992. He joined Intel as a microprocessor failure analysis

engineer. He is now the microprocessor product FA manager in Intel Penang. His main technical interests are developing new FA/FI techniques and studying new failure mechanisms. His e-mail address is Yeoh_Eng_Hong@ccm.ipn.intel.com.

Seong Leong, Lim graduated from University Science Malaysia in 1992. He joined Intel as a microprocessor failure analysis engineer. He is now leading the Pentium Pro microprocessor Fault Isolation group in Intel Penang. His main technical interests are microprocessor architecture and fault isolation. His e-mail address is Seong_Leong_Lim@ccm.ipn.intel.com.

Wong Yik Choong graduated from the University of Malaya in 1994. He joined Intel as a microprocessor failure analysis engineer. He is now the lead Pentium II processor FA engineer in the Penang Microprocessor FA department. His main technical interests are computer architecture and networking, and software development. His e-mail address is Yik_Choong_Wong@ccm.ipn.intel.com

Lock Choon Hou graduated from the University of Malaya in 1996. He joined Intel as a microprocessor product failure analysis engineer. His e-mail address is Choon_Hou_Lock@ccm.sc.intel.com

Mahmud Adnan received a BSEE from Bradley University, Peoria, Illinois in 1990. He joined Intel Penang in 1991 as a microprocessor failure analysis engineer. He is currently working on Merced™ processor's DFT design validation and Merced debug and FA tools development. His e-mail address is Mahmud_Adnan@ccm.sc.intel.com.

Computer Vision Face Tracking For Use in a Perceptual User Interface

Gary R. Bradski, Microcomputer Research Lab, Santa Clara, CA, Intel Corporation

Index words: computer vision, face tracking, mean shift algorithm, perceptual user interface, 3D graphics interface

Abstract

As a first step towards a perceptual user interface, a computer vision color tracking algorithm is developed and applied towards tracking human faces. Computer vision algorithms that are intended to form part of a perceptual user interface must be fast and efficient. They must be able to track in real time yet not absorb a major share of computational resources: other tasks must be able to run while the visual interface is being used. The new algorithm developed here is based on a robust non-parametric technique for climbing density gradients to find the mode (peak) of probability distributions called the mean shift algorithm. In our case, we want to find the mode of a color distribution within a video scene. Therefore, the mean shift algorithm is modified to deal with dynamically changing color probability distributions derived from video frame sequences. The modified algorithm is called the Continuously Adaptive Mean Shift (CAMSHIFT) algorithm. CAMSHIFT's tracking accuracy is compared against a Polhemus tracker. Tolerance to noise, distractors and performance is studied.

CAMSHIFT is then used as a computer interface for controlling commercial computer games and for exploring immersive 3D graphic worlds.

Introduction

This paper is part of a program to develop a Perceptual User Interface for computers. Perceptual interfaces are ones in which the computer is given the ability to sense and produce analogs of the human senses, such as allowing computers to perceive and produce localized sound and speech, giving computers a sense of touch and force feedback, and in our case, giving computers an ability to see. The work described in this paper is part of a larger effort aimed at giving computers the ability to segment, track, and understand the pose, gestures, and emotional expressions of humans and the tools they might be using in front of a computer or settop box. In this paper we describe the development of the first core module in this effort: a 4-degree of freedom color object tracker and its application to flesh-tone-based face tracking.

Computer vision face tracking is an active and developing field, yet the face trackers that have been developed are not sufficient for our needs. Elaborate methods such as tracking contours with snakes [[10][12][13]], using Eigenspace matching techniques [14], maintaining large sets of statistical hypotheses [15], or convolving images with feature detectors [16] are far too computationally expensive. We want a tracker that will track a given face in the presence of noise, other faces, and hand movements. Moreover, it must run fast and efficiently so that objects may be tracked in real time (30 frames per second) while consuming as few system resources as possible. In other words, this tracker should be able to serve as *part* of a user interface that is in turn *part* of the computational tasks that a computer might routinely be expected to carry out. This tracker also needs to run on inexpensive consumer cameras and not require calibrated lenses.

In order, therefore, to find a fast, simple algorithm for basic tracking, we have focused on color-based tracking [[7][8][9][10][11]], yet even these simpler algorithms are too computationally complex (and therefore slower at any given CPU speed) due to their use of color correlation, blob and region growing, Kalman filter smoothing and prediction, and contour considerations. The complexity of the these algorithms derives from their attempts to deal with irregular object motion due to perspective (near objects to the camera seem to move faster than distal objects); image noise; distractors, such as other faces in the scene; facial occlusion by hands or other objects; and lighting variations. We want a fast, computationally efficient algorithm that handles these problems in the course of its operation, i.e., an algorithm that mitigates the above problems "for free."

To develop such an algorithm, we drew on ideas from robust statistics and probability distributions. Robust statistics are those that tend to ignore outliers in the data (points far away from the region of interest). Thus, robust algorithms help compensate for noise and distractors in the vision data. We therefore chose to use a robust non-parametric technique for climbing density gradients to find the mode of probability distributions called the mean shift algorithm [2]. (The mean shift algorithm was never intended to be used as a tracking algorithm, but it is quite effective in this role.)

The mean shift algorithm operates on probability distributions. To track colored objects in video frame sequences, the color image data has to be represented as a probability distribution [1]; we use color histograms to accomplish this. Color distributions derived from video image sequences change over time, so the mean shift algorithm has to be modified to adapt dynamically to the probability distribution it is tracking. The new algorithm that meets all these requirements is called CAMSHIFT.

For face tracking, CAMSHIFT tracks the X, Y, and Area of the flesh color probability distribution representing a face. Area is proportional to Z, the distance from the camera. Head roll is also tracked as a further degree of freedom. We then use the X, Y, Z, and Roll derived from CAMSHIFT face tracking as a perceptual user interface for controlling commercial computer games and for exploring 3D graphic virtual worlds.

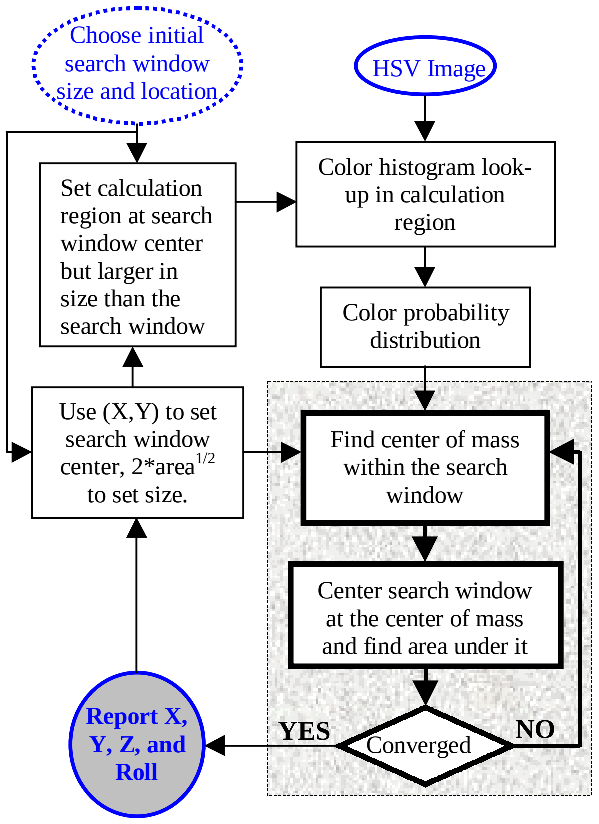
Choose initial search window size and location
HSV Image
Set calculation region at search window center but larger in size than the search window
Color histogram look-up in calculation region
Color probability distribution
Use (X,Y) to set search window center, 2*area^1/2 to set size.
Find center of mass within the search window
Center search window at the center of mass and find area under it
Report X, Y, Z, and Roll
YES
Converged
NO

Figure 1: Block diagram of color object tracking

Figure 1 summarizes the algorithm described below. For each video frame, the raw image is converted to a color probability distribution image via a color histogram model of the color being tracked (flesh for face tracking). The center and size of the color object are found via the CAMSHIFT algorithm operating on the color probability image (the gray box is the mean shift algorithm). The current size and location of the tracked object are reported and used to set the size and location of the search window in the next video image. The process is then repeated for continuous tracking.

Video Demonstrations

The following three videos demonstrate CAMSHIFT in action.

1. FaceTrack_Fast.avi
2. FaceTrack_Distractors.avi
3. FaceTrack_HandOcclusion.avi

The first video shows CAMSHIFT tracking rapid face movements. The second video shows CAMSHIFT tracking a face with other faces moving in the scene. The third video shows CAMSHIFT tracking a face through hand occlusions. These videos are available from this paper on the Web in the *Intel Technology Journal Q2'98* under the site http://developer.intel.com/technology/itj.

Color Probability Distributions

In order to use CAMSHIFT to track colored objects in a video scene, a probability distribution image of the desired color (flesh color in the case of face tracking) in the video scene must be created. In order to do this, we first create a model of the desired hue using a color histogram. We use the Hue Saturation Value (HSV) color system [5][6] that corresponds to projecting standard Red, Green, Blue (RGB) color space along its principle diagonal from white to black (see arrow in Figure 2). This results in the hexcone in Figure 3. Descending the V axis in Figure 3 gives us smaller hexcones corresponding to smaller (darker) RGB subcubes in Figure 2.

HSV space separates out hue (color) from saturation (how concentrated the color is) and from brightness. We create our color models by taking 1D histograms from the H (hue) channel in HSV space.

For face tracking via a flesh color model, flesh areas from the user are sampled by prompting users to center their face in an onscreen box, or by using motion cues to find flesh areas from which to sample colors. The hues derived from flesh pixels in the image are sampled from the H channel and binned into an 1D histogram. When sampling is complete, the histogram is saved for future use. More robust histograms may be made by sampling flesh hues from multiple people. Even simple flesh histograms tend to work well with a wide variety of people without having to be updated. A common misconception is that different color models are needed for different races of people, for example, for blacks and whites. This is not true. Except for albinos, humans are all the same color (hue). Dark-skinned people simply have greater flesh color saturation than light-skinned people, and this is separated out in the HSV color system and ignored in our flesh-tracking color model.

During operation, the stored flesh color histogram is used as a model, or lookup table, to convert incoming video pixels to a corresponding probability of flesh image as can be seen in the right-hand image of Figure 6. This is done for each video frame. Using this method, probabilities range in discrete steps from zero (probability 0.0) to the maximum probability pixel value (probability 1.0). For 8-bit hues, this range is between 0 and 255. We then track using CAMSHIFT on this probability of flesh image.

When using real cameras with discrete pixel values, a problem can occur when using HSV space as can be seen in Figure 3. When brightness is low (V near 0), saturation is also low (S near 0). Hue then becomes quite noisy, since in such a small hexcone, the small number of discrete hue pixels cannot adequately represent slight changes in RGB. This then leads to wild swings in hue values. To overcome this problem, we simply ignore hue pixels that have very low corresponding brightness values. This means that for very dim scenes, the camera must auto-adjust or be adjusted for more brightness or else it simply cannot track. With sunlight, bright white colors can take on a flesh hue so we also use an upper threshold to ignore flesh hue pixels with corresponding high brightness. At very low saturation, hue is not defined so we also ignore hue pixels that have very low corresponding saturation (see Implementation Details section below).

Originally, we used a 2D color histogram built from normalized red green (r,g) space (r = R/(R+G+B), g = G/(R+G+B)). However, we found that such color models are much more sensitive to lighting changes since saturation (which is influenced by lighting) is not separated out of that model.

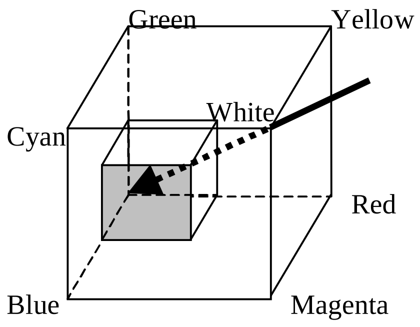

Figure 2: RGB color cube

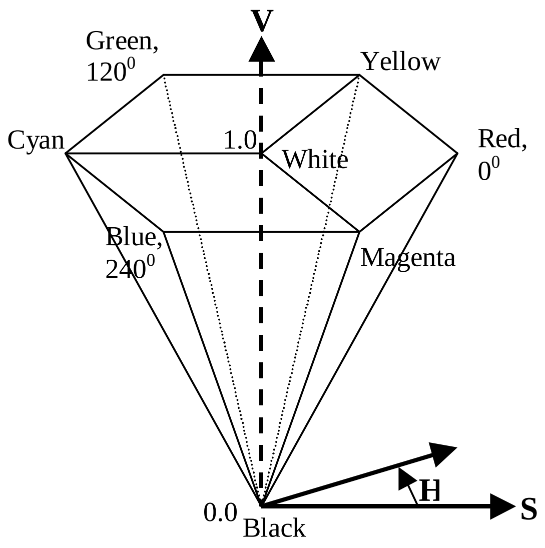

Figure 3: HSV color system

CAMSHIFT Derivation

The closest existing algorithm to CAMSHIFT is known as the mean shift algorithm [2][18]. The mean shift algorithm is a non-parametric technique that climbs the gradient of a probability distribution to find the nearest dominant mode (peak).

How to Calculate the Mean Shift Algorithm

1. Choose a search window size.
2. Choose the initial location of the search window.
3. Compute the mean location in the search window.
4. Center the search window at the mean location computed in Step 3.
5. Repeat Steps 3 and 4 until convergence (or until the mean location moves less than a preset threshold).

Proof of Convergence [18]

Assuming a Euclidean distribution space containing distribution f, the proof is as follows reflecting the steps above:

1. A window *W* is chosen at size *s*.

2. The initial search window is centered at data point p_k
3. Compute the mean position within the search window

$$\hat{\bar{p}}_k(W) = \frac{1}{|W|} \sum_{j \in W} p_j .$$

The mean shift climbs the gradient of *f(p)*

$$\hat{\bar{p}}_k(W) - p_k \approx \frac{f'(p_k)}{f(p_k)} .$$

4. Center the window at point

$$\hat{\bar{p}}_k(W).$$

5. Repeat Steps 3 and 4 until convergence.

Near the mode $f'(p) \cong 0,$ so the mean shift algorithm converges there.

For discrete 2D image probability distributions, the mean location (the centroid) within the search window (Steps 3 and 4 above) is found as follows:

Find the zeroth moment

$$M_{00} = \sum_x \sum_y I(x,y).$$

Find the first moment for *x* and *y*

$$M_{10} = \sum_x \sum_y xI(x,y); \; M_{01} = \sum_x \sum_y yI(x,y).$$

Then the mean search window location (the centroid) is

$$x_c = \frac{M_{10}}{M_{00}}; \quad y_c = \frac{M_{01}}{M_{00}};$$

where *I(x,y)* is the pixel (probability) value at position *(x,y)* in the image, and *x* and *y* range over the search window.

Unlike the Mean Shift algorithm, which is designed for static distributions, CAMSHIFT is designed for dynamically changing distributions. These occur when objects in video sequences are being tracked and the object moves so that the size and location of the probability distribution changes in time. The CAMSHIFT algorithm adjusts the search window size in the course of its operation. Initial window size can be set at any reasonable value. For discrete distributions (digital data), the minimum window size is three as explained in the Implementation Details section. Instead of a set or externally adapted window size, CAMSHIFT relies on the zeroth moment information, extracted as part of the internal workings of the algorithm, to continuously adapt its window size within or over each video frame. One can think of the zeroth moment as the distribution "area" found under the search window. Thus, window radius, or height and width, is set to a function of the the zeroth moment found during search. The CAMSHIFT algorithm is then calculated using any initial non-zero window size (greater or equal to three if the distribution is discrete).

How to Calculate the Continuously Adaptive Mean Shift Algorithm

1. Choose the initial location of the search window.
2. Mean Shift as above (one or many iterations); store the zeroth moment.
3. Set the search window size equal to a function of the zeroth moment found in Step 2.
4. Repeat Steps 2 and 3 until convergence (mean location moves less than a preset threshold).

In Figure 4 below, CAMSHIFT is shown beginning the search process at the top left step by step down the left then right columns until convergence at bottom right. In this figure, the red graph is a 1D cross-section of an actual sub-sampled flesh color probability distribution of an image of a face and a nearby hand. In this figure, yellow is the CAMSHIFT search window, and purple is the mean shift point. The ordinate is the distribution value, and the abscissa is the horizontal spatial position within the original image. The window is initialized at size three and converges to cover the tracked face but not the hand in six iterations. In this sub-sampled image, the maximum distribution pixel value is 206 so we set the width of the search window to be *2*M$_0$/206* (see discussion of window size in the Implementation Details section below). In this process, CAMSHIFT exhibits typical behavior: it finds the center of the nearest connected distribution region (the face), but ignores nearby distractors (the hand).

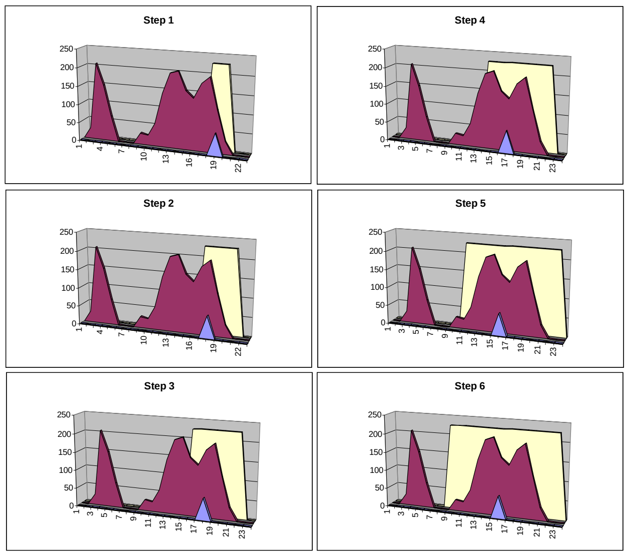

Figure 4: CAMSHIFT in operation down the left then right columns

Figure 4 shows CAMSHIFT at startup. Figure 5 below shows frame to frame tracking. In this figure, the red color probability distribution has shifted left and changed form. At the left in Figure 5, the search window starts at its previous location from the bottom right in Figure 4. In one iteration it converges to the new face center.

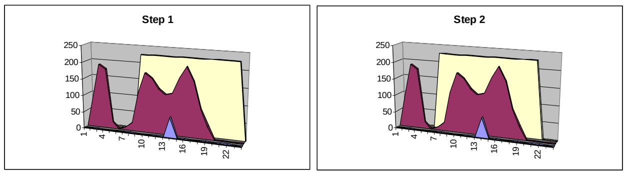

Figure 5: Example of CAMSHIFT tracking starting from the converged search location in Figure 4 bottom right

Mean Shift Alone Does Not Work

The mean shift algorithm alone would fail as a tracker. A window size that works at one distribution scale is not suitable for another scale as the color object moves towards and away from the camera. Small fixed-sized windows may get lost entirely for large object translation in the scene. Large fixed-sized windows may include distractors (other people or hands) and too much noise.

CAMSHIFT for Video Sequences

When tracking a colored object, CAMSHIFT operates on a color probability distribution image derived from color histograms. CAMSHIFT calculates the centroid of the 2D color probability distribution within its 2D window of calculation, re-centers the window, then calculates the area for the next window size. Thus, we needn't calculate the color probability distribution over the whole image, but can instead restrict the calculation of the distribution to a smaller image region surrounding the current CAMSHIFT window. This tends to result in large computational savings when flesh color does not dominate the image. We refer to this feedback of calculation region size as the Coupled CAMSHIFT algorithm.

How to Calculate the Coupled CAMSHIFT Algorithm

1. First, set the calculation region of the probability distribution to the whole image.
2. Choose the initial location of the 2D mean shift search window.
3. Calculate the color probability distribution in the 2D region centered at the search window location in an area slightly larger than the mean shift window size.
4. Mean shift to convergence or for a set number of iterations. Store the zeroth moment (area or size) and mean location.
5. For the next video frame, center the search window at the mean location stored in Step 4 and set the window size to a function of the zeroth moment found there. Go to Step 3.

For each frame, the mean shift algorithm will tend to converge to the mode of the distribution. Therefore, CAMSHIFT for video will tend to track the center (mode) of color objects moving in a video scene. Figure 6 shows CAMSHIFT locked onto the mode of a flesh color probability distribution (mode center and area are marked on the original video image). In this figure, CAMSHIFT marks the face centroid with a cross and displays its search window with a box.

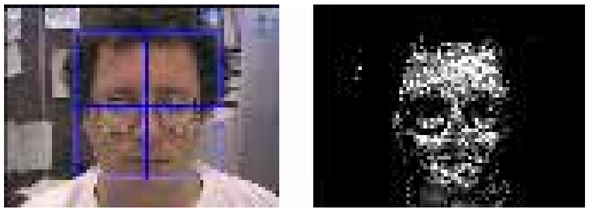

Figure 6: A video image and its flesh probability image

Calculation of Head Roll

The 2D orientation of the probability distribution is also easy to obtain by using the second moments during the course of CAMSHIFT's operation where (x,y) range over the search window, and I(x,y) is the pixel (probability) value at (x,y):

Second moments are

$$M_{20} = \sum_x \sum_y x^2 I(x,y); \quad M_{20} = \sum_x \sum_y x^2 I(x,y).$$

Then the object orientation (major axis) is

$$\theta = \frac{\arctan\left(\dfrac{2\left(\dfrac{M_{11}}{M_{00}} - x_c y_c \right)}{\left(\dfrac{M_{20}}{M_{00}} - x_c^2 \right) - \left(\dfrac{M_{02}}{M_{00}} - y_c^2 \right)} \right)}{2}$$

The first two Eigenvalues (major length and width) of the probability distribution "blob" found by CAMSHIFT may be calculated in closed form as follows [4]. Let

$$a = \frac{M_{20}}{M_{00}} - x_c^2,$$

$$b = 2\left(\frac{M_{11}}{M_{00}} - x_c y_c \right),$$

and

$$c = \frac{M_{02}}{M_{00}} - y_c^2,$$

Then length l and width w from the distribution centroid are

$$l = \sqrt{\frac{(a+c) + \sqrt{b^2 + (a-c)^2}}{2}},$$

$$w = \sqrt{\frac{(a+c) - \sqrt{b^2 + (a-c)^2}}{2}}.$$

When used in face tracking, the above equations give us head roll, length, and width as marked in Figure 7.

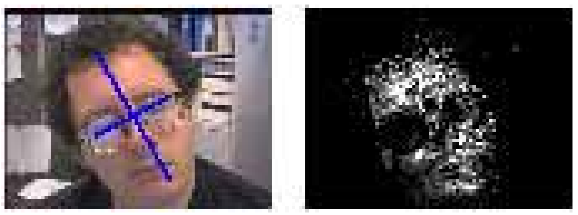

Figure 7: Orientation of the flesh probability distribution marked on the source video image

CAMSHIFT thus gives us a computationally efficient, simple to implement algorithm that tracks four degrees of freedom (see Figure 8).

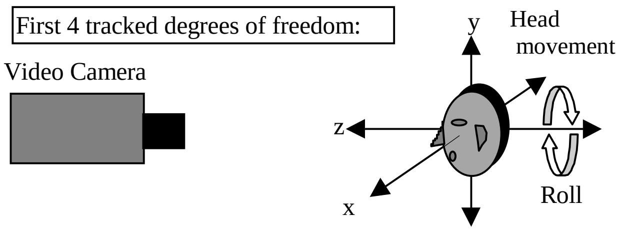

Figure 8: First four head tracked degrees of freedom: X, Y, Z location, and head roll

How CAMSHIFT Deals with Image Problems

When tracking color objects, CAMSHIFT deals with the image problems mentioned previously of irregular object motion due to perspective, image noise, distractors, and facial occlusion as described below.

CAMSHIFT continuously re-scales itself in a way that naturally fits the structure of the data. A colored object’s potential velocity and acceleration scale with its distance to the camera, which in turn, scales the size of its color distribution in the image plane. Thus, when objects are close, they can move rapidly in the image plane, but their probability distribution also occupies a large area. In this situation, CAMSHIFT’s window size is also large and so can catch large movements. When objects are distant, the color distribution is small so CAMSHIFT’s window size is small, but distal objects are slower to traverse the video scene. This natural adaptation to distribution scale and translation allows us to do without predictive filters or variables–a further computational saving–and serves as an in-built antidote to the problem of erratic object motion.

CAMSHIFT’s windowed distribution gradient climbing causes it to ignore distribution outliers. Therefore, CAMSHIFT produces very little jitter in noise and, as a result, tracking variables do not have to be smoothed or filtered. This gives us robust noise tolerance.

CAMSHIFT’s robust ability to ignore outliers also allows it to be robust against distractors. Once CAMSHIFT is locked onto the mode of a color distribution, it will tend to ignore other nearby but non-connected color distributions. Thus, when CAMSHIFT is tracking a face, the presence of other faces or hand movements in the scene will not cause CAMSHIFT to loose the original face unless the other faces or hand movements substantially occlude the original face.

CAMSHIFT’s provable convergence to the mode of probability distributions helps it ignore partial occlusions of the colored object. CAMSHIFT will tend to stick to the mode of the color distribution that remains.

Moreover, when CAMSHIFT’s window size is set somewhat greater than the root of the distribution area under its window, CAMSHIFT tends to grow to encompass the connected area of the distribution that is being tracked (see Figure 4). This is just what is desired for tracking whole objects such as faces, hands, and colored tools. This property enables CAMSHIFT to not get stuck tracking, for example, the nose of a face, but instead to track the whole face.

Implementation Details

Initial Window Size and Placement

In practice, we work with digital video images so our distributions are discrete. Since CAMSHIFT is an algorithm that climbs the gradient of a distribution, the minimum search window size must be greater than one in order to detect a gradient. Also, in order to center the window, it should be of odd size. Thus for discrete distributions, the minimum window size is set at three. For this reason too, as CAMSHIFT adapts its search window size, the size of the search window is rounded up to the current or next greatest odd number. In practice, at start up, we calculate the color probability of the whole scene and use the zeroth moment to set the window size (see subsection below) and the centroid to set the window center.

Setting Adaptive Window Size Function

Deciding what function of the zeroth moment to set the search window size to in Step 3 of the CAMSHIFT algorithm depends on an understanding of the distribution that one wants to track and the goal that one wants to achieve. The first consideration is to translate the zeroth moment information into units that make sense for setting window size. Thus, in Figure 4, the maximum distribution value per discrete cell is 206, so we divide the zeroth moment by 206 to convert the calculated area under the search window to units of number of cells. Our goal is then to track the whole color object so we need an expansive window. Thus, we further multiply the result by two so that the window grows to encompass the connected distribution area. We then round to the next greatest odd search window size so that the window has a center.

For 2D color probability distributions where the maximum pixel value is 255, we set window size *s* to

$$s = 2 * \sqrt{\frac{M_{00}}{256}}.$$

We divide by 256 for the same reason stated above, but to convert the resulting 2D region to a 1D length, we need to take the square root. In practice, for tracking faces, we set window width to *s* and window length to *1.2s* since faces are somewhat elliptical.

Comments on Software Calibration

Much of CAMSHIFT's robustness to noise, transient occlusions, and distractors depends on the search window matching the size of the object being tracked—it is better to err on the side of the search window being a little too small. The search window size depends on the function of the zeroth moment M_{00} chosen above. To indirectly control the search window size, we adjust the color histogram up or down by a constant, truncating at zero or saturating at the maximum pixel value. This adjustment affects the pixel values in the color probability distribution image which affects M_{00} and hence window size. For 8-bit hue, we adjust the histogram down by 20 to 80 (out of a maximum of 255), which tends to shrink the CAMSHIFT window to just within the object being tracked and also reduces image noise.

HSV brightness and saturation thresholds are employed since hue is not well defined for very low or high brightness or low saturation. Low and high thresholds are set off 10% of the maximum pixel value.

Comments on Hardware Calibration

To use CAMSHIFT as a video color object tracker, the camera's field of view (zoom) must be set so that it covers the space that one intends to track in. Turn off automatic white balance if possible to avoid sudden color shifts. Try to set (or auto-adjust) AGC, shutter speed, iris or CCD integration time so that image brightness is neither too dim nor saturating. The camera need not be in focus to track colors. CAMSHIFT will work well with cheap cameras and does not need calibrated lenses.

CAMSHIFT'S Use as a Perceptual Interface

Treatment of CAMSHIFT Tracking Variables for Use in a Perceptual User Interface

Figure 8 above shows the variables X, Y, Z, and Roll returned by the CAMSHIFT face tracker. For game and graphics control, X, Y, Z, and Roll often require a "neutral" position; that is, a position relative to which further face movement is measured. For example, if the captured video image has dimensions (Y, X) of 120x160, a typical neutral position might be Y=60, X=80. Then if X < 80, the user has moved 80-X left; if X > 80, the user has moved X-80 right and so on for each variable.

Piecewise Linear Transformation of Control Variables

To obtain differential control at various positions including a jitter-damping neutral movement region, each variable's relative movement (above or below the neutral position) is scaled in "N" different ranges.

In the X variable example above, if X is in range #1, X would be scaled by "X scale 1"; if X is in range #2, X would be scaled by "X scale 2" and so on.

The formula for mapping captured video head position P to a screen movement factor F is

$F = \min(b_1,P)s_1 + [\min(b_2-b_1, P-b_1)]^+s_2 + \ldots + [\min(b_{(i+1)}-b_i, P-b_i)]^+s_{(i+1)} + \ldots + [P-b_{(N-1)}]^+s_{N},$ **(control equation 1)**

where $[\#]^+$ equals "#" if # > 0, and zero otherwise; min(A, B) returns the minimum of A or B; b_1-b_N represents the bounds of the ranges, s_1-s_N are the corresponding scale factors for each range, and P is the absolute value of the difference of the variable's location from neutral.

This allows for stability close to the neutral position, with growing acceleration of movement as the distance away from neutral increases, in a piecewise linear manner as shown in Figure 9.

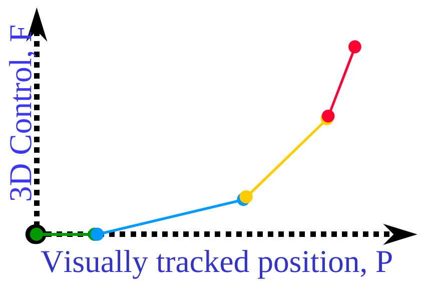

Figure 9: Piecewise linear transformation of CAMSHIFT position P interface control variable F

Frame Rate Adjustment

If the graphic's rendering rate R can be determined, the final screen movement S is

$S = F/R$ **(control equation 2)**

Computer graphics and game movement commands S can be issued on each rendered graphic's frame. Thus, it is best for the movement amount S to be sensitive to frame rate. In computer-rendered graphic or game scenes, simple views (for example, looking up at blue sky) are rendered much faster than complex views (for example, texture mapped city skylines). The final rate of movement should not depend on the complexity of the 3D view if one wants to achieve satisfying motion when immersed in a 3D scene.

Y Variable Special Case for Seated User

If a user sits facing the camera (neutral X,Y) and then leans left or right (X movement) by pivoting on his/her chair, Y will decrease as shown in Figure 10 below.

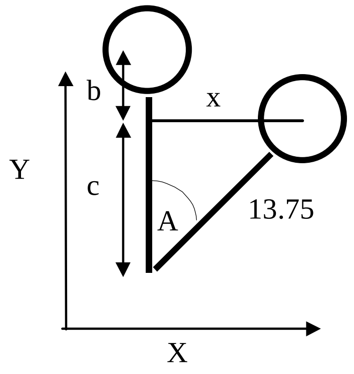

Figure 10: Lean changes Y and head roll

In order to overcome this, we use an empirical observation that the 2nd Eigenvalue (face half width) of the local face flesh color distribution length is proportional to face size (see Figure 7), which is, on average, often proportional to body size. Empirically, the ratio of the 2nd Eigenvector to torso length from face centroid is

1 to 13.75 (2 inches to 27.5 inches).**(control equation 3)**

Given lean distance x (in 2nd Eigenvector units), and seated size of 13.75, as in Figure 10 so that $\sin(A) = x/13.75$. Then,

$A = \sin^{-1}(x/13.75)$, **(control equation 4)**

so $c = 13.75\cos(A)$, and

$b = 13.75(1 - \cos(A))$ **(control equation 5)**

in units of 2nd Eigenvectors. This is the Y distance to correct for (add back) when leaning.

Roll Considerations

As can be seen from Figure 10, for seated users lean also induces a change in head roll by the angle A. Thus, for control that relies on head roll, this lean-induced roll should be corrected for. Correction can be accomplished in two ways:

- Make the first range boundary b1 in control equation 1 large enough to contain the changes in face orientation that result from leaning. Then use a scale value s1 = 0 so that leaning causes no roll.
- Subtract the measured roll from the lean-induced roll, A, calculated in control Equation 4 above.

Another possible problem can result when the user looks down too much as shown in Figure 11. In this case, the user is looking down at the keyboard. Looking down too much causes the forehead to dominate the view which in turn causes the face flesh color "blob" to look like it is oriented horizontally.

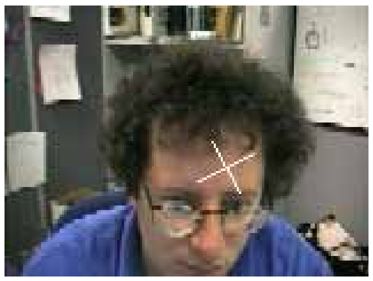

Figure 11: Extreme down head pitch causes a corrupted head roll value

To correct for such problems, we define a new variable, Q called "Roll Quality." Q is the ratio of the first two Eigenvalues, length l and width w, of the distribution color "blob" in the CAMSHIFT search window:

$Q = l/w.$ **(control equation 6)**

For problem views of the face such as in Figure 11, we observe that Roll Quality is nearly 1.0. So, for face tracking, roll should be ignored (treated as vertical) for quality measures less than 1.25. Roll should also be ignored for very high quality scores greater than 2.0 since such distributions are un-facelike and likely to have resulted from noise or occlusions.

CAMSHIFT's Actual Use as an Interface

CAMSHIFT is being used as a face tracker to control games and 3D graphics. By inserting face control variables into the mouse queue, we can control unmodified commercial games such as Quake 2 shown in Figure 12. We used left and right head movements to slide a user left and right in the game, back and forth head movements to move the user backwards and forwards, up or down movements to let the user shoot (as if ducking or getting jolted by the gun), and roll left or right to turn the user left or right in the game. This methodology has been used extensively in a series of demos with over 30 different users.

Head tracking via CAMSHIFT has also been used to experiment with immersive 3D graphics control in which natural head movements are translated to moving the corresponding 3D graphics camera viewpoint. This has been extensively tested using a 3D graphics model of the Forbidden City in China as well as in exploring a 3D graphics model of the big island of Hawaii as shown in Figure 13. Most users find it an enjoyable experience in which they naturally pick up how to control the graphics viewpoint movement.

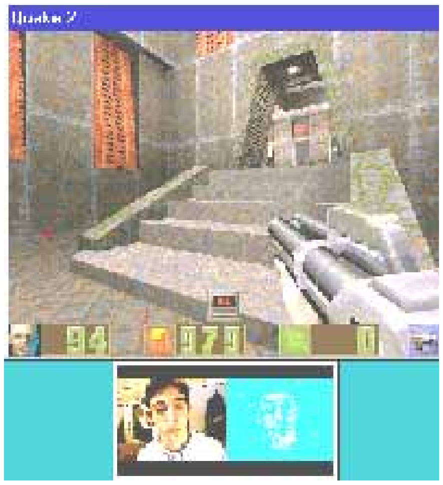

Figure 12: CAMSHIFT-based face tracker used to play Quake 2 hands free by inserting control variables into the mouse queue

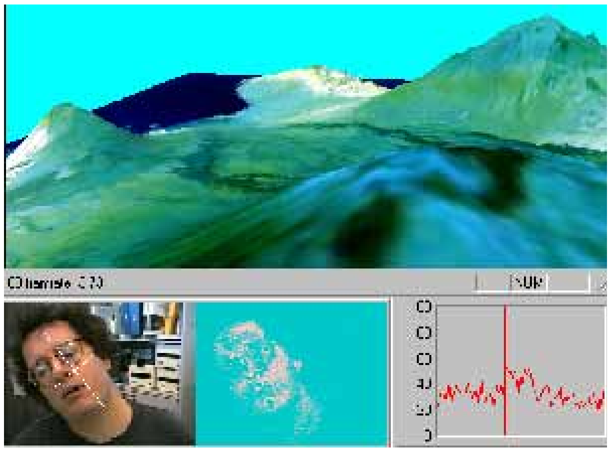

Figure 13: CAMSHIFT-based face tracker used to "fly" over a 3D graphic's model of Hawaii

CAMSHIFT Analysis

Comparison to Polhemus

In order to assess the tracking accuracy of CAMSHIFT, we compared its accuracy against a Polhemus tracker. Polhemus is a magnetic sensor connected to a system that measures six degrees of spatial freedom and thus can be used for object tracking when tethered to an object. The observed accuracy of Polhemus is +/- 1.5cm in spatial location and about 2.5^o in orientation within 30 inches of the Polhemus antenna. We compared Polhemus tracking to CAMSHIFT color object tracking using a 320x240 image size (see Figure 14a-d). The coordinate systems of Polhemus and the camera were carefully aligned prior to testing. The object tracked was pulled on a cart in a set trajectory away from the Polhemus origin. The comparison between CAMSHIFT and Polhemus in each of X, Y, Z, and Roll yielded the results shown Table 1.

Tracking Variable	X	Y	Z	Roll
Standard Deviation of Difference	0.27cm	0.58cm	3.4cm	2.4^o

Table 1: Standard deviation of Polhemus vs. CAMSHIFT tracking differences

Z exhibited the worst difference because CAMSHIFT determines Z by measuring color area, which is inherently noisy. X, Y, and Roll are well within Polhemus's observed tracking error and therefore indistinguishable. Z is about 2cm off. Except for Z, these results are as good or better than much more elaborate vision tracking systems [17], although CAMSHIFT does not yet track pitch and yaw.

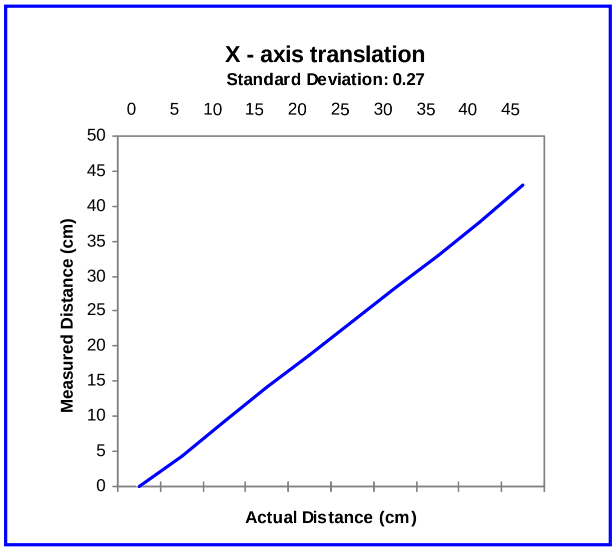

Figure 14a: Comparision of X tracking accuracy

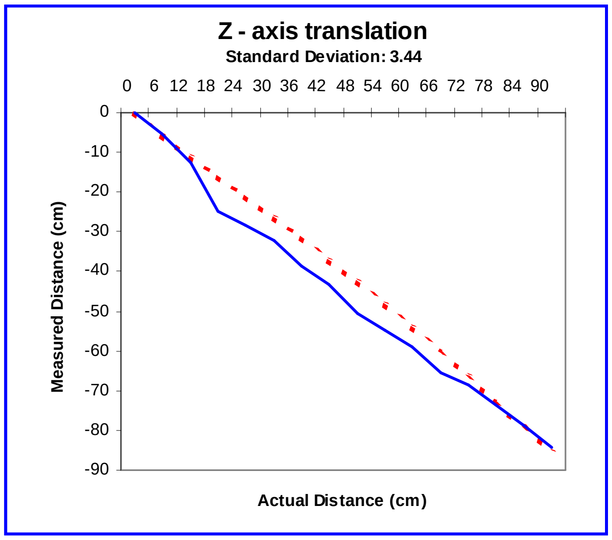

Figure 14c: Comparision of Z tracking accuracy

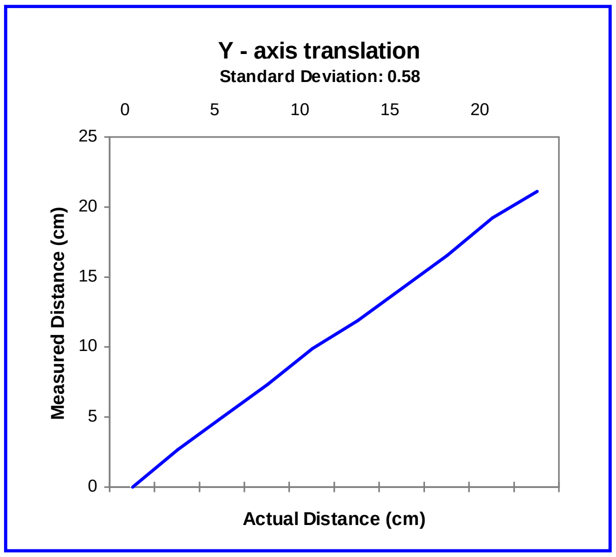

Figure 14b: Comparision of Y tracking accuracy

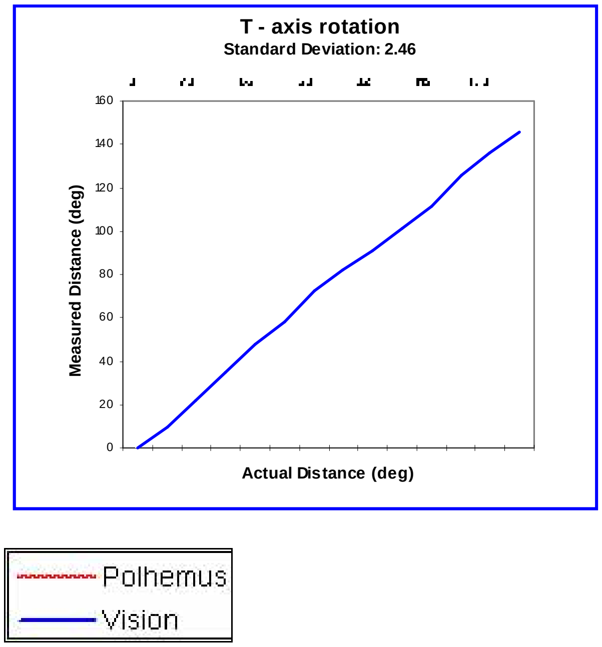

Figure 14d: Accuracy comparison of Polhemus and CAMSHIFT tracking for roll.

Tracking in Noise

CAMSHIFT's robust ability to find and track the mode of a dynamically changing probability distribution also gives it good tracking behavior in noise. We videotaped a head movement sequence and then played it back adding 0, 10, 30, and 50% uniform noise. Figure 15 shows 50% noise added to the raw image on the left, and the resulting color probability distribution on the right. Note that the use of a color model greatly cuts down the random noise since color noise has a low probability of being flesh color.

Nevertheless, the flesh color model is highly degraded and there are many spurious flesh pixels in the color probability distribution image. But CAMSHIFT is still able to track X, Y, and Roll quite well in up to 30% white noise as shown in Figure 16a-d. Z is more of a problem because CAMSHIFT measures Z by tracking distribution area under its search window, and one can see in Figure 15 that area is highly effected by noise. Y shows an upward shift simply because the narrower chin region exhibits more degradation in noise than the wider forehead. Roll tracks well until noise is such that the length and width of the face color distribution are obscured. Thus, CAMSHIFT handles noise well without the need for extra filtering or adaptive smoothing.

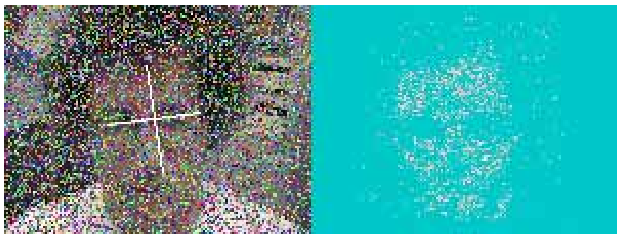

Figure 15: Tracking in 50% uniform noise

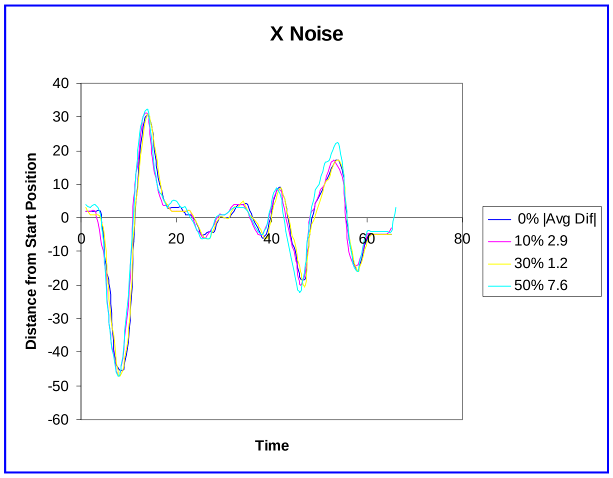

Figure 16a: X accuracy in 0-50% uniform noise

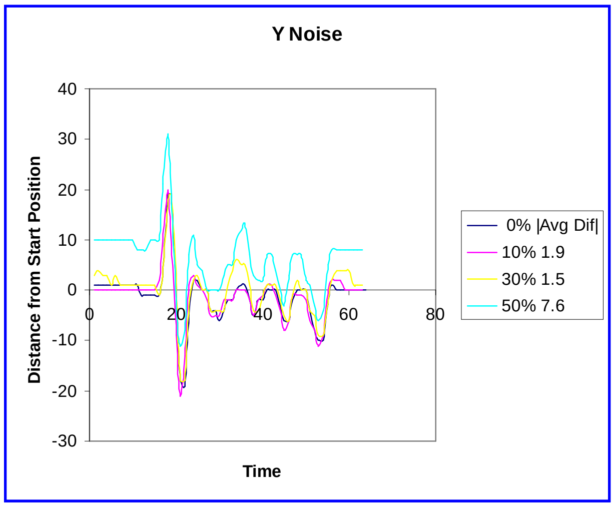

Figure 16b: Y accuracy in 0-50% uniform noise

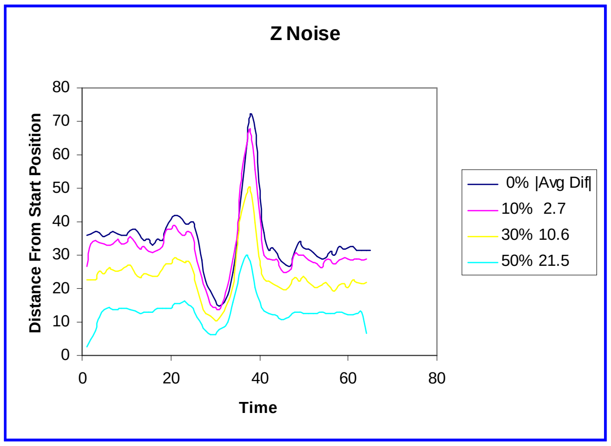

Figure 16c: Z accuracy in 0-50% uniform noise

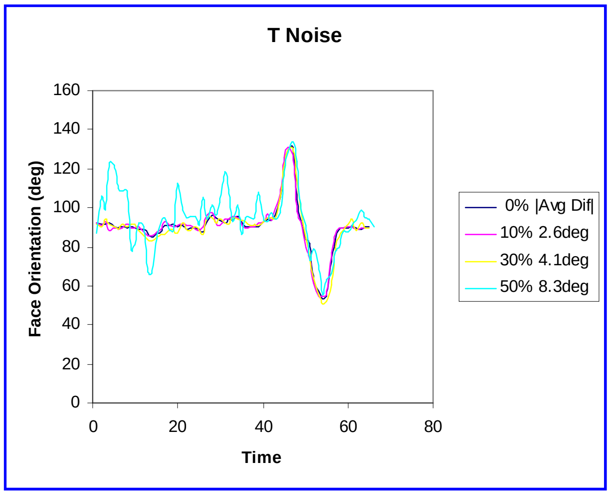

Figure 16d: Roll accuracy in 0-50% uniform noise

Tracking in the Presence of Distractions

CAMSHIFT's search window converges to span the nearest dominant connected probability distribution. If we adjust the nature of the probability distribution by properly setting the HSV brightness and saturation threshold (see Implementation Details section above), the search window will tend to stay just within the object being tracked as shown in the marked image at top left in Figure 17. In such cases, CAMSHIFT is robust against distracting (nearby) distributions and transient occlusions. This robustness occurs for distractors because the search window rarely contains the distractor as shown sequentially down the left, then right, columns of Figure 17.

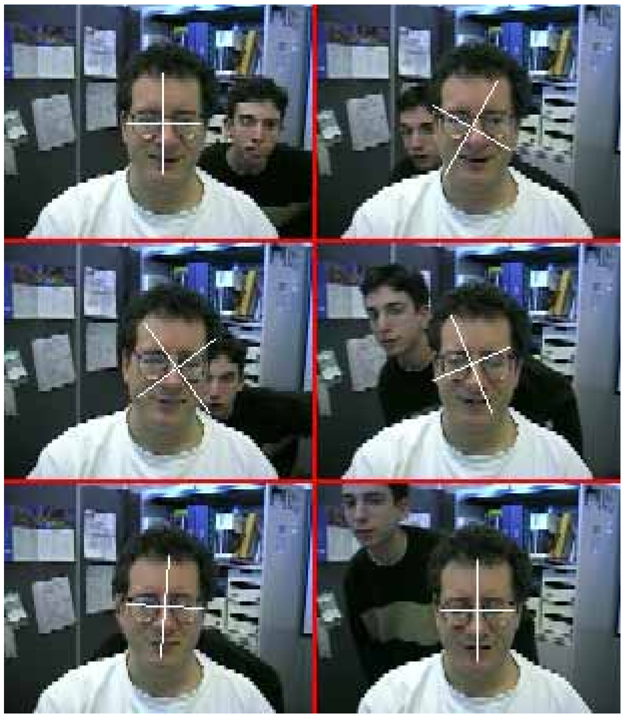

Figure 17: Tracking a face with background distractor faces (sequence: down left then right columns)

Table 2 shows the results collected from 44 sample point on five tracking runs with active background face distraction such as that shown in Figure 17. Since the distracting face rarely intersects much of CAMSHIFT's search window, the X, Y, and Z tracking variables are perturbed very little. Roll is more strongly affected since even a small intersection of a distractor in CAMSHIFT's search window can change the effective orientation of the flesh pixels as measured by CAMSHIFT.

Tracked Variable	Average Std. Deviation	Maximum Std. Deviation
X (pixels)	0.42	2.00
Y(pixels)	0.53	1.79
Z(pixels)	0.54	1.50
Roll (degrees)	5.18	46.80

Table 2: Perturbation of CAMSHIFT tracking variables by face distractors

CAMSHIFT tends to be robust against transient occlusion because the search window will tend to first absorb the occlusion and then stick with the dominant distribution mode when the occlusion passes. Figure 18 demonstrates robustness to hand occlusion in sequential steps down the left, then right columns.

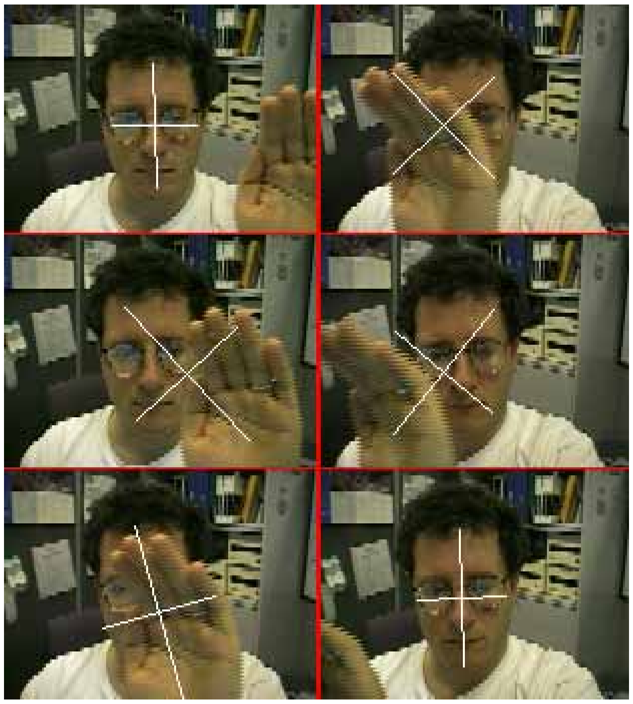

Figure 18: Tracking a face in the presence of passing hand occlusions (sequence: down left then right columns)

Table 3 shows the results collected from 43 sample points on five tracking runs with active transient hand occlusion of the face. Average perturbation is less than three pixels for X, Y, and Z. Roll is more strongly effected due to the arbitrary orientation of the hand as it passes through the search window.

Tracked Variable	Average Std. Deviation	Maximum Std. Deviation
X (pixels)	2.35	7.17
Y(pixels)	2.81	6.29
Z(pixels)	2.10	4.65
Roll (degrees)	14.64	34.40

Table 3: Perturbation of CAMSHIFT tracking variables by passing hand occlusion

We see from the above table that CAMSHIFT gives us wide tolerance for distraction and occlusion "for free" due to the statistically robust workings of the algorithm.

Performance

The order of complexity of CAMSHIFT is $O(\alpha N^2)$ where α is some constant, and the image is taken to be NxN. α is most influenced by the moment calculations and the average number of mean shift iterations until convergence. The biggest computational savings come through scaling the region of calculation to an area around the search window size as previously discussed.

CAMSHIFT was run on a 300 MHz Pentium® II processor, using an image size of 160x120 at 30 frames per second (see Figure 19). CAMSHIFT's performance scales with tracked object size. Figure 19 shows the CPU load from the entire computer vision thread including image acquisition, conversion to color probability distribution, and CAMSHIFT tracking. In Figure 19 when the tracked face is far from the camera, the CAMSHIFT thread consumes only 10% of the CPU cycles. When the face fills the frame, CAMSHIFT consumes 55% of the CPU.

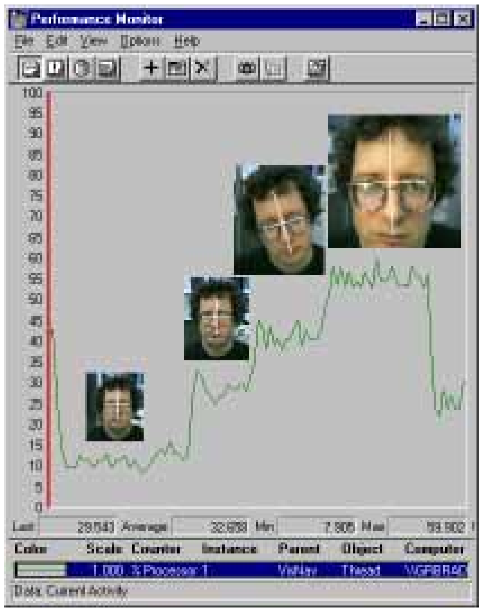

Figure 19: Performance scales inversely with tracked object size (ordinate is percent of CPU used)

Figure 20 traces computer vision thread's performance in an actual control task of "flying" over a 3D model of Hawaii using head movements. In this case, the average CPU usage was 29%. VTUNE™ analysis showed that the actual CAMSHIFT operation (excluding image capture, color conversion or image copying) consumed under 12% of the CPU. CAMSHIFT relies on Intel's MMX™ technology optimized Image Processing Library available on the Web [3] to do RGB-to-HSV image conversion and image allocation. MMX technology optimized image moments calculation has recently been added to the Image Processing Library, but not in time for publication. The use of such optimized moment calculations will boost performance noticeably since this forms part of the inner mean shift calculation loop. Even without this improvement, CAMSHIFT's current average actual use efficiency of 29% allows it to be used as a visual user interface.

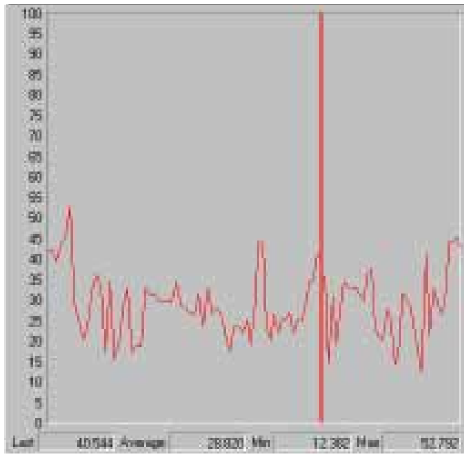

Figure 20: In actual use, CAMSHIFT consumed an average of 29% of one 300 MHz Pentium® II CPU when used to control a 3D graphic's Hawaii fly through

Discussion

This paper discussed a core tracking module that is part of a larger effort to allow computers to track and understand human motion, pose, and tool use. As such, the module was designed to be simple and computationally efficient. Yet, this core module must still handle the basic computer-vision problems outlined in this paper. We've seen that CAMSHIFT handles these problems as follows:

- **Irregular object motion**: CAMSHIFT scales its search window to object size thus naturally handling perspective-induced motion irregularities.
- **Image noise**: The color model eliminates much of the noise, and CAMSHIFT tends to ignore the remaining outliers.
- **Distractors:** CAMSHIFT ignores objects outside its search window so objects such as nearby faces and hands do not affect CAMSHIFT's tracking.
- **Occlusion:** As long as occlusion isn't 100%, CAMSHIFT will still tend to follow what is left of the objects' probability distribution.
- **Lighting variation:** Using only hue from the HSV color space and ignoring pixels with high or low brightness gives CAMSHIFT wide lighting tolerance.

CAMSHIFT's simplicity does cause limitations however. Since CAMSHIFT derives Z from object area estimates, Z is subject to noise and spurious values. The effects of noise are evident in Figure 16c. That CAMSHIFT can get spurious area values is evident in Figure 11.

Since CAMSHIFT relies on color distributions alone, errors in color (colored lighting, dim illumination, too much illumination) will cause errors in tracking. More sophisticated trackers use multiple modes such as feature tracking and motion analysis to compensate for this, but more complexity would undermine the original design criterion for CAMSHIFT.

CAMSHIFT also only detects four (X, Y, Z, and Roll) of the six modes of freedom (above plus pitch and yaw). Unfortunately, of the six degrees of head movement possible, Roll is the least useful control variable since it is the least "natural" head movement and is therefore fatiguing for the user to use constantly.

Conclusion

CAMSHIFT is a simple, computationally efficient face and colored object tracker. While acknowledging the limitation imposed by its simplicity, we can still see that CAMSHIFT tracks virtually as well as more expensive tethered trackers (Polhemus) or much more sophisticated, computationally expensive vision systems [17], and it tracks well in noisy environments. Thus, as we have shown, even though CAMSHIFT was conceived as a simple part of a larger tracking system, it has many uses right now in game and 3D graphics' control.

Adding perceptual interfaces can make computers more natural to use, more fun for games and graphics, and a better medium of communication. These new features consume more MIPs and so will take advantage of more MIPs available with future Intel® CPUs.

In this project, we designed a highly efficient face tracking algorithm rather than a more complex, higher MIPs usage algorithm. This was done because we want to be able to demonstrate compelling applications and interfaces on today's systems in order to prepare the way for the future use of computer vision on PCs. CAMSHIFT is usable as a visual interface now, yet designed to be part of a more robust, larger tracking system in the future. CAMSHIFT will be incorporated into larger, more complex, higher MIPs-demanding modules that provide more robust tracking, posture understanding, gesture and face recognition, and object understanding. In this way, the functionality of the computer vision interface will increase with increasing Intel CPU speeds. A user will thus be able to upgrade their computer vision interface by upgrading to higher speed Intel CPUs in the future.

Acknowledgments

Many thanks to Mark Holler for countless discussions and feedback leading to the development of this work. Special thanks to Ryan Boller for rapid implementation of the CAMSHIFT game control demo and for major help in implementing the testing routines for CAMSHIFT.

References

[1] D. Comaniciu and P. Meer, "Robust Analysis of Feature Spaces: Color Image Segmentation," CVPR'97, pp. 750-755.

[2] K. Fukunaga, "Introduction to Statistical Pattern Recognition," Academic Press, Boston, 1990.

[3] MMX^{TM} technology optimized libraries in image, signal processing, pattern recognition and matrix math can be downloaded from http://developer.intel.com/design/perftool/perflibst/index.htm)

[4] W.T. Freeman, K. Tanaka, J.Ohta, and K. Kyuma, "Computer Vision for Computer Games," Int. Conf. On Automatic Face and Gesture Recognition, pp. 100-105, 1996.

[5] A.R. Smith, "Color Gamut Transform Pairs," SIGGRAPH 78, pp. 12-19, 1978.

[6] J.D. Foley, A. van Dam, S. K. Feiner and J.F. Hughes, "Computer graphics principles and practice," Addison-Wesley, pp. 590-591.

[7] P. Fieguth and D. Terzopoulos, "Color-based tracking of heads and other mobile objects at video frame rates," In Proc. Of IEEE CVPR, pp. 21-27, 1997.

[8] C. Wren, A. Azarbayejani, T. Darrell, A.Pentland, "Pfinder: Real-Time Tracking of the Human Body," SPIE Vol. 2615, 1995.

[9] M. Hunke and A. Waibel, "Face locating and tracking for human-computer interaction," Proc. Of the 28th Asilomar Conf. On Signals, Sys. and Comp., pp. 1277-1281, 1994.

[10] K. Sobottka and I. Pitas, "Segmentation and tracking of faces in color images," Proc. Of the Second Intl. Conf. On Auto. Face and Gesture Recognition, pp. 236-241, 1996.

[11] M. Swain and D. Ballard, "Color indexing," Intl. J. of Computer Vision, 7(1) pp. 11-32, 1991.

[12] M. Kass, A. Witkin D.Terzopoulos, "Snakes: Active contour Models," Int. J. o f Computer Vision (1) #4, pp. 321-331, 1988.

[13] C. Vieren, F. Cabestaing, J. Postaire, "Catching moving objects with snakes for motion tracking," Pattern Recognition Letters (16) #7, pp. 679-685, 1995.

[14] A. Pentland, B. Moghaddam, T. Starner, "View-based and Modular Eigenspaces for face recognition," CVPR'94, pp. 84-91, 1994.

[15] M. Isard, A. Blake, "Contour tracking by stochastic propagation of conditional density," Proc. 4th European Conf. On Computer Vision, Cambridge, UK, April 1996.

[16] T. Maurer, and C. von der Malsburg, "Tracking and learning graphs and pose on image sequence of faces," Proc. Of the Second Intl. Conf. On Auto. Face and Gesture Recognition, pp. 176-181, 1996.

[17] A. Azabayejani, T. Starner, B. Horowitz, and A. Pentland, "Visually Controlled Graphics," IEEE Tran. Pat. Anal. and Mach. Intel. pp. 602-605, Vol. 15, No. 6, June 1993.

[18] Y. Cheng, "Mean shift, mode seeking, and clustering," IEEE Trans. Pattern Anal. Machine Intell., 17:790-799, 1995.

Author's Biography

Dr. Gary R. Bradski received a Ph.D. in pattern recognition and computer vision from Boston University. He works in computer vision research in the Microcomputer Research Labs at Intel's Mission College campus. His interests include segmenting and tracking people in visual scenes; perceptual computer interfaces; applying visual 3D structure to 3D graphics; pattern recognition; biological perception and self-organization. His e-mail is gary.bradski@intel.com.

Proving the IEEE Correctness of Iterative Floating-Point Square Root, Divide, and Remainder Algorithms

Marius Cornea-Hasegan, Microprocessor Products Group, Hillsboro, OR, Intel Corp.

Index words: floating-point, IEEE correctness, divide, square root, remainder

Abstract

The work presented in this paper was initiated as part of a study on software alternatives to the hardware implementations of floating-point operations such as divide and square root. The results of the study proved the viability of software implementations, and showed that certain proposed algorithms are comparable in performance to current hardware implementations. This paper discusses two components of that study:

(1) A methodology for proving the IEEE correctness of the result of iterative algorithms that implement the floating-point square root, divide, or remainder operation.

(2) Identification of operands for the floating-point divide and square root operations that lead to results representing difficult cases for IEEE rounding.

Some general properties of floating-point computations are presented first. The IEEE correctness of the floating-point square root operation is discussed next. We show how operands for the floating-point square root that lead to difficult cases for rounding can be generated, and how to use this knowledge in proving the IEEE correctness of the result of iterative algorithms that calculate the square root of a floating-point number. Similar aspects are analyzed for the floating-point divide operation, and we present a method for generating difficult cases for rounding. In the case of the floating-point divide operation, however, it is more difficult to use this information in proving the IEEE correctness of the result of an iterative algorithm than it is for the floating-point square root operation. We examine the restrictions on the method used for square root. Finally, we present possible limitations due to the finite exponent range.

Introduction

Floating-point divide, remainder, and square root are three important operations performed by computing systems today. The IEEE-754 Standard for Binary Floating-Point Arithmetic [1] requires that the result of a divide or square root operation be calculated as if in infinite precision, and then rounded to one of the two nearest floating-point numbers of the specified precision that surround the infinitely precise result (the remainder will always be exact).

Most processor implementations to date have used hardware-based implementations for divide, remainder, and square root. Recent research has led to software-based iterative algorithms for these operations that are expected to always generate the IEEE-correct result. Several advantages can be envisioned. Firstly, software-based algorithms lead to a possibly higher throughput than before because they are capable of being pipelined; secondly, they are easier to modify; and finally, they reduce the actual chip size because they do not have to be implemented in hardware.

Different algorithms are used depending on the target precision of the result, or on the particular architecture of the processor. In either case, performance and IEEE correctness have to be ensured. The expected levels of performance are possible due to improvements in the floating-point architecture of most modern processors. Proving the IEEE correctness of the results generated by the divide, remainder, and square root operations, which is of utmost importance for modern processors, is the object of this paper. A new methodology for achieving this goal is presented. Operands that lead to "difficult" cases for rounding are also identified, allowing better testing of the implementations for these operations. The method presented was verified through a mix of mathematical proofs, sustained by Mathematica [2], C language, and assembly language programs.

Correctness proofs following methods similar to those presented in this paper, in conjunction with careful and thorough verification and testing of the actual implementation, should ensure flawless floating-point divide, remainder, and square root operations on modern processors that adopt iterative, software-based algorithms.

Some of the mathematical notations used in the following sections are explained at the end of this paper.

General Properties of Floating-Point Computations and IEEE Correctness

Floating-point numbers are represented as a concatenation of a sign bit, an M-bit exponent field, and an N-bit significand field. Mathematically

$$f = \sigma \cdot s \cdot 2^e$$

where $\sigma = \pm 1$, $s \in [1,2)$, $e \in [e_{min}, e_{max}] \cap \mathbf{Z}$, $s = 1 + k / 2^{N-1}$, $k \in \{0,1,2,\ldots,2^{N-1}-1\}$, $e_{min} = -2^{M-1}+2$, and $e_{max} = 2^{M-1}-1$. Let $\mathbf{F}_N$ be the set of floating-point numbers with N-bit significands and unlimited exponent range, and let $\mathbf{F}_{M,N}$ be the set of floating-point numbers with M-bit exponents and N-bit significands (no special values included). Note that $\mathbf{F}_{M,N} \subset \mathbf{F}_N \subset \mathbf{Q}^*$.

The IEEE-754 Standard for Binary Floating-Point Arithmetic allows several formats, but the most common are single precision (M=8, N=24), double precision (M=11, N=53), and double-extended precision (M=15, N=64). Even though there is only a finite number of real values that can be represented as floating-point numbers (which constitute a finite subset of the rational numbers), the total of $2 \cdot 2^M \cdot 2^N$ floating-point numbers or special values for a given precision can be huge. Verifying correctness of a binary floating-point operation for double-extended precision by exhaustive testing on a state-of-the-art workstation could easily take 2^{50} years. The only operation that lends itself to such testing, from among those considered herein, is the single precision square root.

The variable density of the floating-point numbers on the real axis implies that it would be difficult to have a requirement regarding the absolute error when approximating real values by floating-point numbers. Instead, most iterative algorithms are analyzed by trying to limit the relative error of the result being generated.

The IEEE-754 standard requires that the result of a divide or square root operation be calculated as if in infinite precision, and then rounded to one of the two nearest floating-point numbers of the specified precision that surround the infinitely precise result. Special cases occur when this is outside the supported range. The IEEE standard specifies four rounding modes: to nearest (*rn*), to negative infinity (*rm*), to positive infinity (*rp*), and toward zero (*rz*).

In order to determine whether an iterative algorithm for an IEEE operation yields the correctly rounded result (in any rounding mode), we have to evaluate the error that occurs due to rounding in each computational step. Two measures are commonly used for this purpose. The first is the error of an approximation with respect to the exact result, expressed in fractions of an *ulp*, or unit in the last place. For the floating-point number $f = \sigma \cdot s \cdot 2^e \in \mathbf{F}_N$ given above, one *ulp* has the magnitude

$$1\ ulp = 2^{e-N+1}$$

An alternative is to use the relative error. If the real number x is approximated by the floating-point number a, then the relative error ε is determined by

$$a = x \cdot (1+\varepsilon)$$

The non-linear relationship between *ulp*s and the corresponding relative error is illustrated in Figure 1 (where the outer envelopes mark the minimum and maximum values of the relative error), by Theorem 1, and also by Corollaries 1a and 1b of this theorem. For example, if $x > 1.000$ in Figure 4, but x is much closer to 1.000 than to 1.001, then if we approximate x by $a=1.001$, the relative error is close to the largest possible (when staying within 1 *ulp* of the approximation), i.e., close to $\varepsilon = 1/8$ ($1/8 = 2^{-N+1}$ for N = 4).

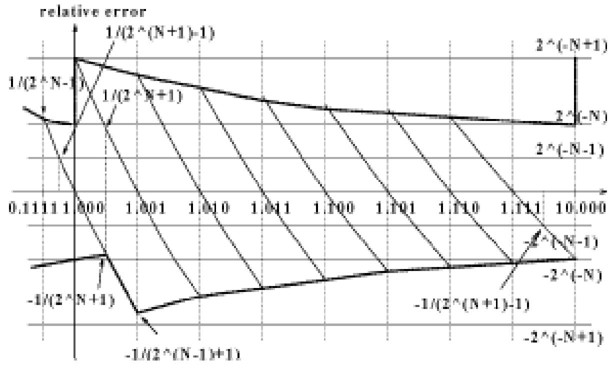

Figure 1: Relative error when approximating within 1 *ulp*, positive real numbers by floating-point numbers in $\mathbf{F}_4$

Theorem 1 (*ulp*-s versus relative error) Let $x \in \mathbf{R}^*$ be a real number, and $a \in \mathbf{F}_N$, $a = \sigma \cdot s \cdot 2^e$, $|\sigma| = 1$, $s = 1+k/2^{N-1}$, $k \in \mathbf{Z}$, $0 \le k \le 2^{N-1}-1$, $e \in \mathbf{Z}$ and $m \in \mathbf{R}$, $0 < m \le 1$. Then

(a) For $k \neq 0$: $a = \sigma \cdot (1+k/2^{N-1}) \cdot 2^e \cong x$ within m *ulp* of x $\Leftrightarrow$
$a = x \cdot (1+\varepsilon)$, $\varepsilon \in (-m/(2^{N-1}+k+m), m/(2^{N-1}+k-m))$

(b) For k=0: $a = \sigma \cdot 2^e \cong x$ within m *ulp* of x $\Leftrightarrow$
$a = x \cdot (1+\varepsilon)$, $\varepsilon \in (-m/(2^{N-1}+m), m/(2^N-m))$

(For the sake of brevity, no proofs are included in this paper.)

Several corollaries can be derived from this property, but the following two are often useful.

Corollary 1a. Let $x \in R^*$, and $a \in \mathbf{F}_N^*$. If $a \cong x$, within 1 *ulp* of x, then $a = x \cdot (1+\varepsilon)$, with $|\varepsilon| < 1/2^{N-1}$.

Corollary 1b. Let $x \in R^*$, and $a \in \mathbf{F}_N^*$. If $a = x \cdot (1+\varepsilon)$, with $|\varepsilon| < 1/2^N$, then $a \cong x$, within 1 *ulp* of x.

Some properties used in IEEE correctness proofs are formulated in terms of errors expressed in *ulp*-s. However, it is easier to evaluate the errors introduced by each computational step in an iterative algorithm as relative errors. The two properties above, together with other similar ones, are used in going back and forth from one form to the other, as needed.

In order to show that the final result generated by a floating-point algorithm represents the correctly rounded value of the infinitely precise result x, one has to show that the exact result and the final result of the algorithm *before rounding*, y, lie within such an interval that, through rounding, they end up at the same floating-point number. Figure 2 illustrates these intervals over a binade (an interval delimited by consecutive integer powers of the base 2) when (a) only rounding to nearest, and (b) any IEEE rounding mode is acceptable. In this figure, the floating-point numbers are marked on the bottom real axis.

First we will discuss the square root, and then we will examine the divide and remainder operations.

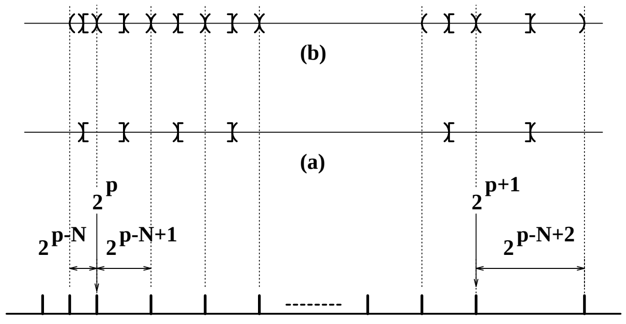

Figure 2: Intervals that condition (a) $x_{rn} = y_{rn}$, and (b) $x_{rnd} = y_{rnd}$

IEEE Correctness Proofs for Iterative Floating-Point Square Root Algorithms

We will first present the means to determine difficult cases for rounding in floating-point square root operations, and then we will show how to use this knowledge in proving the IEEE correctness of the result of iterative algorithms implementing the floating-point square root operation.

Iterative Algorithms for Floating-Point Square Root

Among the most common iterative algorithms for calculating the value of the square root of a floating-point number are those based on the Newton-Raphson method. Such algorithms are suggested in [3]. Starting with an initial guess of $1/\sqrt{a}$, a very good approximation of $1/\sqrt{a}$ is derived first in a number of iterations. From this, the value of $\sqrt{a}$ can be calculated. For example, an algorithm that starts by determining the value of the root of $f(x) = 1/x^2 - a$, could have the following iteration:

$$e_i = (1/2 - 1/2 \cdot a \cdot y_i^2)_{rn}$$
$$y_{i+1} = (y_i + e_i \cdot y_i)_{rn}$$

where e_i is the error term, y_{i+1} is the next better approximation, and the subscript *rn* denotes the IEEE rounding to nearest mode.

No matter what iterative algorithm is being used, one can easily evaluate the relative errors introduced in each computational step. We will show that if the relative error that we can derive is small enough, then the IEEE correctness of the generated result is proven. In practice though, the relative error of the final result is more often than not larger than required. We can compensate for the difference between what we can determine in terms of relative error and what is needed, by knowing the values of input arguments that lead to the most difficult cases for rounding. The following subsections will show how to achieve this.

Difficult Cases for Rounding

The lemma shown below allows difficult cases for rounding to be determined. For rounding to nearest, these are represented by values of the argument a of the square root function such that $\sqrt{a}$ is different from, but very close to, a midpoint between two consecutive floating-point numbers. For rounding toward negative infinity, positive infinity, or zero, $\sqrt{a}$ has to be different from, but very close to, a floating-point number. We will show next how to determine the most difficult cases in each category.

Lemma 1. (a) Let $a \in \mathbf{F}_N$, $a > 0$, $a = \sigma \cdot s \cdot 2^e$. Then its value can be written as

$$a = A \cdot 2^{e-2N+2} \quad \text{for } e = 2 \cdot u,\ u \in \mathbf{Z}, \text{ or}$$
$$a = A \cdot 2^{e-2N+1} \quad \text{for } e = 2 \cdot u+1,\ u \in \mathbf{Z}$$

where $A \in [2^{2N-2}, 2^{2N-1})$, $A \equiv 0 \pmod{2^{N-1}}$ for $e=2\cdot u$, $u \in \mathbf{Z}$, and $A \in [2^{2N-1}, 2^{2N})$, $A \equiv 0 \pmod{2^N}$ for $e=2\cdot u+1$, $u \in \mathbf{Z}$.

(b) $\sqrt{a} \in \mathbf{F}_N$ if and only if $\sqrt{A} \in [2^{N-1}, 2^{N-1/2}) \cap \mathbf{Z}$ for $e=2\cdot u$, $u \in \mathbf{Z}$, and $\sqrt{a} \in \mathbf{F}_N$ if and only if $\sqrt{A} \in [2^{N-1/2}, 2^N) \cap \mathbf{Z}$ for $e=2\cdot u+1$, $u \in \mathbf{Z}$ (note that $\sqrt{A}$ has to be an integer in both cases).

(c) $\sqrt{a}$ is a midpoint between two consecutive floating-point numbers in $\mathbf{F}_N$ if and only if $\sqrt{A}$ is an integer + 1/2, $\sqrt{A} \in [2^{N-1}, 2^N)$.

(d) Let $a \in \mathbf{F}_N$. If $\sqrt{a} \notin \mathbf{F}_N$, then $\sqrt{a} \notin \mathbf{F}_{N+1}$ (this is to say that $\sqrt{a}$ cannot be a midpoint between two consecutive floating-point numbers in $\mathbf{F}_N$; the observation is necessary, as $\mathbf{F}_N \subset \mathbf{F}_{N+1}$).

The properties above will help us determine the difficult cases for rounding. Consider first the cases of rounding toward negative infinity, positive infinity, or zero. If $\sqrt{a} \notin \mathbf{F}_N$ (it is not representable as a floating-point number with N bits in the significand), then how close can it be to a floating-point number $f \in \mathbf{F}_N$? Because the values of $\sqrt{A}$ fall in $[2^{N-1}, 2^N)$, where floating-point numbers with N-bit significands are integers and 1 *ulp* = 1, this can be re-formulated as "how close can $\sqrt{A}$ be to an N-bit integer number F?" The answer is given in Figure 3.

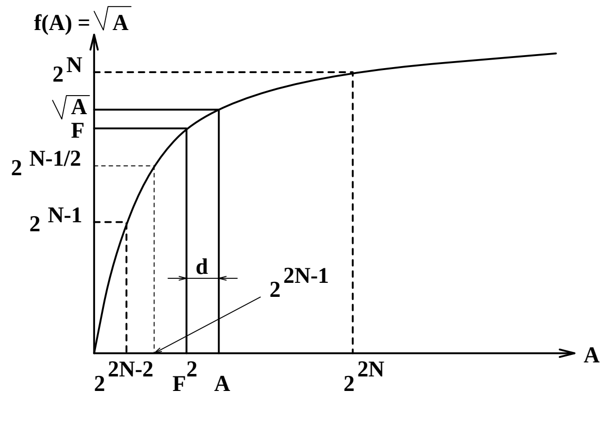

Figure 3: The distance d of A to the square of an integer $F \in [2^{N-1}, 2^N)$

This figure shows that our question is directly related to "how close can A be to the square of an N-bit number in $[2^{N-1}, 2^N)$?" To find out, we have to solve the equation

$$(2^{N-1} + k)^2 = A + \delta$$

for increasing values of $\delta \in \mathbf{Z}^*$, and for $0 \le k \le 2^{N-1}-1$, $k \in \mathbf{Z}$. This leads to two cases, depending on whether the exponent e of a is even or odd.

If e is even, according to Lemma 1, the equation above becomes

$$(2^{N-1} + k)^2 = 2^{2N-2} + m \cdot 2^{N-1} + \delta$$

for $0 \le m \le 2^{N-1}-1$, $m \in \mathbf{Z}$.

If e is odd, according also to Lemma 1, the equation becomes

$$(2^{N-1} + k)^2 = 2^{2N-1} + m \cdot 2^N + \delta$$

for $0 \le m \le 2^{N-1}-1$, $m \in \mathbf{Z}$.

Solving the diophantine equations above (equations in integer numbers) for $\delta = 1,-1,2,-2,3,-3,\ldots$ we find at most two acceptable solutions for each value of δ. For example, for e even and $\delta = 1$, the only solution is $A = 2^{2N-2}+2\cdot 2^{N-1}$, which is at a distance $d = \delta = 1$ of $(2^{N-1}+1)^2$, the square of $F = 2^{N-1}+1$. The value of a that corresponds to this A (a obtained as shown in Lemma 1), has a significand of $1.00\ldots010_B$, and it constitutes a difficult case for rounding toward negative infinity, positive infinity, or zero, as $\sqrt{a}$ is different from, but very close to, a floating-point number in $\mathbf{F}_N$ (it is in fact the "most difficult" such case). For example, for N = 24 and e = 0

$$\sqrt{1.0000000000000000000000010_B} = 1.00000000000000000000000111111\ldots_B$$

is very close to, but less than,

$1.00000000000000000000001_B$.

Consider next the case of rounding to nearest. If $\sqrt{a} \notin \mathbf{F}_N$ (it is not representable as a floating-point number with N bits in the significand), then how close can it be to a midpoint between two consecutive floating-point numbers in $\mathbf{F}_N$? This can be re-formulated as "how close can $\sqrt{A}$ be to an N-bit integer number plus 1/2?" The answer is given in Figure 4. This figure shows that our question is directly related to "how close can A be to the square of a midpoint between two consecutive N-bit integer numbers in $\mathbf{F}_N \cap [2^{N-1}, 2^N)$?" To find out, we have to solve the equation

$$(2^{N-1} + k + 1/2)^2 = A + 1/4 + \delta$$

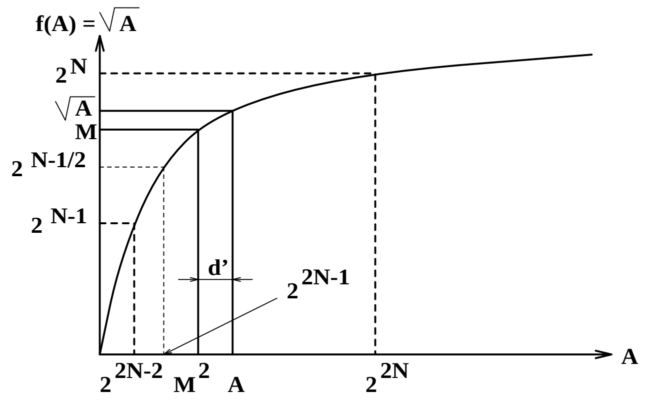

Figure 4: The distance d' of A to the square of M, M = integer +1/2$\in [2^{N-1}, 2^N)$

for increasing values (in absolute value) of $\delta \in \mathbf{Z}$, and for $0 \le k \le 2^{N-1}-1$, $k \in \mathbf{Z}$. This leads again to two cases, depending on whether the exponent e of a is even or odd.

If e is even, according to Lemma 1, the equation above becomes

$$(2^{N-1} + k + 1/2)^2 = 2^{2N-2} + m \cdot 2^{N-1} + 1/4 + \delta$$

for $0 \le m \le 2^{N-1}-1$, $m \in \mathbf{Z}$.

If e is odd, according also to Lemma 1, the equation becomes

$$(2^{N-1} + k + 1/2)^2 = 2^{2N-1} + m \cdot 2^{N} + 1/4 + \delta$$

for $0 \le m \le 2^{N-1}-1$, $m \in \mathbf{Z}$.

Solving the diophantine equations above for δ = 0,1,-1,2, -2,3,-3, … we find again at most two acceptable solutions for each value of δ. For example, for e even and δ = 0, the only solution is $A = 2^{2N-2}+2^{N-1}$, which is at a distance $d = \delta+1/4 = 1/4$ of $(2^{N-1}+1/2)^2$, the square of $M = 2^{N-1}+1/2$. The value of a that corresponds to this A (a obtained as shown in Lemma 1), has a significand of $1.00\ldots001_B$, and it constitutes a difficult case for rounding to nearest, as √a is different from, but very close to, a midpoint between two consecutive floating-point numbers in $\mathbf{F}_N$ (it is in fact the "most difficult" such case). For example, for N = 24 and e = 0

$\sqrt{1.00000000000000000000001_B}$ =
$1.000000000000000000000000111111\ldots_B$

is very close to, but less than,

$1.00000000000000000000000_B$ + 1/2 *ulp*.

Not all solutions to the equations above can be represented by simple formulas. For example, for e even and δ = -2, we obtain

A = $7e08a5000000_H$ and a = $1.f82294_H \cdot 2^e$ if N = 24 (√a very close to $1.673f4a_H \cdot 2^{e/2}$+1/2 *ulp*)

A = $1d407bb3641da50000000000000_H$ and a = $1.d407bb3641da5_H \cdot 2^e$ if N = 53 (√a very close to $1.5a24e31b39fa5_H \cdot 2^{e/2}$+1/2 *ulp*)

A = $4d7f90be2ec18ed98000000000000000_H$ and a = $1.35fe42f8bb063b66_H \cdot 2^e$ if N = 64 (√a very close to $1.19b4bb639c98c0b4_H \cdot 2^{e/2}$ + 1/2 *ulp*)

This method of determining values of the argument a that lead to values of √a that are difficult to round is also useful in generating good quality test vectors for any implementation of the floating-point square root operation, not just for iterative algorithms (these values can be added to others chosen by different criteria). In addition, this method can help in proving the IEEE correctness of the result of iterative algorithms that calculate the square root of a floating-point number, as shown below.

General Properties

Two main properties, Theorems 2 and 3, are used in the proposed method of proving the IEEE correctness of the result of iterative floating-point square root algorithms.

Theorem 2. Let $a \in \mathbf{F}_N$, a>0, $a = \sigma \cdot s \cdot 2^e$. If $\sqrt{a} \notin \mathbf{F}_N$, and A is determined by scaling a as specified in Lemma 1, then for any integer $F \in [2^{N-1}, 2^N)$, and for any $f \in \mathbf{F}_N$

(a) The distance $w_{\sqrt{A}}$ between √A and F satisfies
$w_{\sqrt{A}} = | \sqrt{A} - F | > 1/2^{N+1}$

(b) The distance $w_{\sqrt{a}}$ between √a and f satisfies
$w_{\sqrt{a}} = | \sqrt{a} - f | > 2^{e/2-2N}$, if e = 2·u, and
$w_{\sqrt{a}} = | \sqrt{a} - f | > 2^{e/2-2N-1/2}$, if e = 2·u+1, $u \in \mathbf{Z}$

This theorem states that if √a is not representable as a floating-point number with an N-bit significand, then there are exclusion zones of known minimal width around any floating-point number, within which √a cannot exist. The minimum distance between √a and f, or equivalently, between √A and F, was determined by examining instead the distance between A and F^2, as shown in Figure 3. The minimal width of the exclusion zones can be extended by gradually eliminating the difficult cases for rounding toward negative infinity, positive infinity, or zero (starting with the most difficult cases). For example, if we solve the diophantine equations that determine these cases (as shown above) just for δ = 1,-1,2,-2,3, and -3, two solutions are found for any N, and statements (a) and (b) in Theorem 2 are modified to reflect exclusion zones that are four times wider (in (b), the two inequalities were replaced by the more restrictive one)

(a) The distance $w_{\sqrt{A}}$ between $\sqrt{A}$ and F satisfies
$w_{\sqrt{A}} = | \sqrt{A} - F | > 1/2^{N-1}$, except for
$A_1 = 2^{2N-2} + 2 \cdot 2^{N-1}$ if $e = 2 \cdot u$, $u \in \mathbf{Z}$, and
$A_2 = 2^{2N} - 2 \cdot 2^{N}$ if $e = 2 \cdot u+1$, $u \in \mathbf{Z}$

(b) The distance $w_{\sqrt{a}}$ between $\sqrt{a}$ and f satisfies
$w_{\sqrt{a}} = | \sqrt{a} - f | > 2^{e/2-2N+3/2}$, except for
$a_1 = (1+2^{-N+2}) \cdot 2^{e}$ if $e = 2 \cdot u$, $u \in \mathbf{Z}$, and
$a_2 = (2-2^{-N+2}) \cdot 2^{e}$ if $e = 2 \cdot u+1$, $u \in \mathbf{Z}$

This property is used in conjunction with the one from Theorem 3.

Theorem 3. Let $a \in \mathbf{F}_N$, $a > 0$, $a = \sigma \cdot s \cdot 2^e$. If $\sqrt{a} \notin \mathbf{F}_N$, and A is determined by scaling a as specified in Lemma 1, then for any midpoint m between two consecutive numbers in $\mathbf{F}_N$, and for any midpoint M between two consecutive integers in $[2^{N-1}, 2^N)$, $M = k+1/2 \in [2^{N-1}, 2^N)$, $k \in \mathbf{Z}$

(a) The distance $w_{\sqrt{A}}'$ between $\sqrt{A}$ and M satisfies
$w_{\sqrt{A}}' = | \sqrt{A} - M | > 1/2^{N+3}$

(b) The distance $w_{\sqrt{a}}'$ between $\sqrt{a}$ and m satisfies
$w_{\sqrt{a}}' = | \sqrt{a} - m | > 2^{e/2-2N-2}$, if $e = 2 \cdot u$, and
$w_{\sqrt{a}}' = | \sqrt{a} - m | > 2^{e/2-2 \cdot N-5/2}$, if $e = 2 \cdot u+1$, $u \in \mathbf{Z}$

This theorem states that if $\sqrt{a}$ is not representable as a floating-point number with an N-bit significand, then there are exclusion zones of known minimal width around any midpoint between two consecutive floating-point numbers, within which $\sqrt{a}$ cannot exist. The minimum distance between $\sqrt{a}$ and m, or equivalently, between $\sqrt{A}$ and M, could be determined by examining instead the distance between A and M^2, as shown in Figure 4. Just as in the case of Theorem 2, the minimal width of the exclusion zones can be extended by gradually eliminating the difficult cases for rounding toward nearest (starting with the most difficult cases). For example, if we solve the diophantine equations that determine these cases (as shown above) just for δ = 0,1,-1,2,-2,3,-3, and –4, seven solutions are found for N=24, N=53, and N=64 (but not always from the same equation for all the three values of N), and statements (a) and (b) in Theorem 3 are modified to reflect exclusion zones that are 17 times wider (in (b), the two inequalities were replaced by the more restrictive one):

(a) The distance $w_{\sqrt{A}}'$ between $\sqrt{A}$ and M satisfies
$w_{\sqrt{A}}' = | \sqrt{A} - M | > 17/2^{N+3}$

except for a set of known values A_1' through A_7'.

(b) The distance $w_{\sqrt{a}}'$ between $\sqrt{a}$ and m satisfies
$w_{\sqrt{a}}' = | \sqrt{a} - m | > 17 \cdot 2^{e/2-2N-5/2}$

except for a set of known values a_1' through a_7'.

Some of the actual values of a_i' and of the corresponding A_i' (not shown here) were determined directly from mathematical expressions, but others were determined by C programs, using recursion. Note that it is not necessary to have the same number of solutions for different values of N.

This property is used, as explained below, in conjunction with that in the preceding Theorem 2.

IEEE Correctness of Iterative Floating-Point Square Root Algorithms

In order to prove that an iterative algorithm for calculating the square root of a floating-point number a generates a result that is IEEE correct, we have to show that the result R* before rounding and the exact value of $\sqrt{a}$ lead, through rounding, to the same floating-point number. If the result R* before rounding and the exact $\sqrt{a}$ are closer to each other than half of the minimum width of any exclusion zone determined as shown in Theorems 2 and 3, then $(R^*)_{rnd} = (\sqrt{a})_{rnd}$ for any IEEE rounding mode *rnd*. This is illustrated in Figure 2 (b) above and in Figure 5.

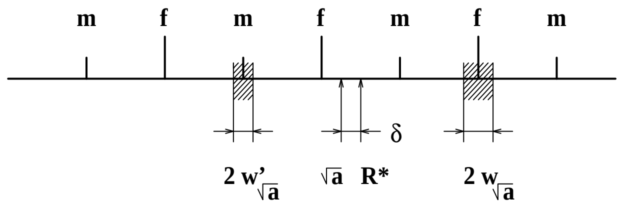

Figure 5: Exclusion zones around floating-point numbers f and around midpoints m between consecutive floating-point numbers

As in Theorem 2 (a), $1/2^{N+1} = 1/2^{N+1}$ *ulp* in $\mathbf{F}_N$ = 1 *ulp* in $\mathbf{F}_{2N+1}$, and in Theorem 3 (a), $1/2^{N+3} = 1/2^{N+3}$ *ulp* in $\mathbf{F}_N$ = 1 *ulp* in $\mathbf{F}_{2N+3}$, the above can be expressed in the following corollaries of these two theorems:

Corollary 2. If $a \in \mathbf{F}_N$, $a > 0$, $R^* \in \mathbf{R}$ and $|R^* - \sqrt{a}| \le 1$ *ulp* in $\mathbf{F}_{2N+1}$, then $(R^*)_{rnd} = (\sqrt{a})_{rnd}$ for any rounding mode $rnd \in \{rm, rp, rz\}$

Corollary 3. If $a \in \mathbf{F}_N$, $a > 0$, $R^* \in \mathbf{R}$ and $|R^* - \sqrt{a}| \le 1$ *ulp* in $\mathbf{F}_{2N+3}$, then $(R^*)_{rn} = (\sqrt{a})_{rn}$

In practice, the inequalities to be verified are

$|R^* - \sqrt{a}| \leq (w_{\sqrt{a}})_{min}$

$|R^* - \sqrt{a}| \leq (w_{\sqrt{a}}')_{min}$

The method of proving the IEEE correctness of the result of an iterative algorithm that calculates the square root of a floating-point number can then be summarized as a sequence of steps:

Step 1. Evaluate the relative error ε of the final result before rounding

$R^* = \sqrt{a} \cdot (1+\varepsilon)$

where an upper bound for $|\varepsilon|$ is known as $|\varepsilon| \leq 2^{-p}$, for some given $p \in \mathbf{R}$, $p > 0$.

Step 2. Evaluate $|R^* - \sqrt{a}| = |\sqrt{a} \cdot \varepsilon| \leq \sqrt{a} \cdot 2^{-p}$ and determine the minimum widths $(w_{\sqrt{a}})_{min}$ and $(w_{\sqrt{a}}')_{min}$ for which the following hold

$|R^* - \sqrt{a}| \leq (w_{\sqrt{a}})_{min}$

$|R^* - \sqrt{a}| \leq (w_{\sqrt{a}}')_{min}$

Step 3. For Theorems 2 and 3, determine a sufficient number of difficult cases for rounding, to allow augmenting the widths of the exclusion zones to the values determined in Step 2 (the smaller this new minimum width, the fewer the points we have to determine). Except possibly for these values, the iterative algorithm for calculating $\sqrt{a}$ generates a result that is IEEE correct. Alternatively, instead of steps 1, 2, 3 above, we could use Theorem 1 to verify that the conditions in Corollaries 2 and 3 are satisfied.

Step 4. Verify directly that the algorithm generates IEEE correct results for the special points determined in Step 3. Once this is done, the algorithm is proven to generate IEEE correct results for all the possible input values of the argument.

The method presented above was applied to several software-based iterative algorithms for calculating the square root of a floating-point number. It proved to be of good practical value, as the number of special points to be verified directly was never larger than 20. It therefore represents a viable option for verifying correctness of this class of algorithms.

IEEE Correctness Proofs for Iterative Floating-Point Divide and Remainder Algorithms

As we did for square root, we will first present a method for determining difficult cases for rounding in floating-point divide operations. We will show, however, that a perfect parallel with the square root is not possible.

The IEEE correctness of the floating-point remainder operation (which is always exact, and therefore not affected by the rounding mode) is a direct consequence of the IEEE correctness of the result of the floating-point divide operation, as

$\text{rem}(a,b) = a - b \cdot \textit{near}(a/b)$

where *near*(x) is the nearest integer to the real number x. Two problems arise. First, we must verify that *near*(a/b) fits in an integer ([4] explains this). Second, we must counter the possibility of a double rounding error because what we calculate is actually

$\text{rem}(a,b) = a - b \cdot \textit{near}((a/b)_{rn})$

To do this, we just need to compare $(a/b)_{rn}$ and $(a/b)_{rz}$, and apply a correction if needed. Note that this might occur only in the tie cases for the rounding to nearest performed in $\textit{near}((a/b)_{rn})$.

It is thus straightforward to prove the IEEE correctness of the floating-point remainder operation, once we have proved it for the floating-point divide. Therefore, from this point on, we will focus on the floating-point divide operation.

Iterative Algorithms for Floating-Point Divide

Just as for square root, among the most common iterative algorithms for calculating the value of the quotient of two floating-point numbers, a/b, are those based on the Newton-Raphson method. Such algorithms are suggested also in [3]. Starting with an initial guess of 1/b, a very good approximation of 1/b is derived first in a number of iterations. From this, the value of a/b can be calculated. For example, an algorithm that starts by determining the value of the root of f (x) = b - 1/x, could have the following iteration:

$e_i = (1 - b \cdot y_i)_{rn}$

$y_{i+1} = (y_i + e_i \cdot y_i)_{rn}$

where e_i is the error term, and y_{i+1} is the next better approximation.

Regardless of the particular iterative algorithm being used, one can evaluate the relative errors introduced in each computational step. We will show that if the relative error that we can derive is small enough, then the IEEE correctness of the generated result is easily proven. In practice though, the relative error of the final result is larger than desired. In the case of the divide (unlike for the square root), we cannot easily compensate for the difference between what we can determine in terms of relative error and what is needed. In such cases, alternative methods have to be used, such as those presented in [3].

Difficult Cases for Rounding

For rounding to nearest, the difficult cases are represented by values of the arguments a and b of the divide operation such that a/b is different from, but very close to, a midpoint between two consecutive floating-point numbers. For rounding toward negative infinity, positive infinity, or zero, a/b has to be different from, but very close to, a floating-point number. We will show next how to determine the most difficult cases in each category.

The following lemma shows how to scale a and b to A and B respectively in order to make reasoning easier (a and b are assumed to be positive here).

Lemma 2. Let $N\in\mathbf{Z}$, $N \geq 3$, $a,b\in\mathbf{F_N}^*$, and *rnd* any IEEE rounding mode. Then

(a) a/b can be expressed as $a/b = A/B\cdot 2^E$, where $A\in\mathbf{Z}\cap\mathbf{F_N}$, $A \equiv 0 \pmod{2^{N-1}}$, $B\in [2^{N-1},2^N)\cap\mathbf{Z}$ and $(A/B)_{rnd}\in [2^{N-1},2^N)\cap\mathbf{Z}$. Let $A = 2^k\cdot A_1$, such that $A_1\in[2^{N-1}, 2^N)\cap\mathbf{Z}$. If $A_1<B$ then $k = N$, and if $A_1>B$, then $k = N-1$.

(b) $a/b\in\mathbf{F}_N$ if and only if $A/B\in[2^{N-1},2^N)\cap\mathbf{Z}$.

(c) a/b is a midpoint between two consecutive floating-point numbers in $\mathbf{F}_N$ if and only if A/B is an integer + 1/2 in $[2^{N-1},2^N)$.

(d) If $a/b\notin\mathbf{F}_N$, then $a/b\notin\mathbf{F}_{N+1}$ (this is to say that a/b cannot be a midpoint between two consecutive floating-point numbers in $\mathbf{F}_N$; the observation is necessary, as $\mathbf{F}_N\subset\mathbf{F}_{N+1}$).

Note that above, $A_1/B = s_a/s_b$ (ratio of the significands).

We can now determine the most difficult cases for rounding.

Consider first the cases of rounding toward negative infinity, positive infinity, or zero. If $a/b\notin\mathbf{F}_N$ (it is not representable as a floating-point number with N bits in the significand), then how close can a/b be to a floating-point number $f\in\mathbf{F}_N$? Because the values of A/B fall in $[2^{N-1},2^N)$, where floating-point numbers with N-bit significands are integers, and 1 *ulp* = 1, this can be re-formulated as "how close can A/B be to an N-bit integer number F?" To answer this, we have to solve the equation

$$A/B = q + \delta/B$$

for increasing values of $\delta\in\mathbf{Z}^*$, and for $q\in[2^{N-1},2^N)\cap\mathbf{Z}$. This leads to two cases, depending on whether $A_1<B$, or $A_1>B$ (see Lemma 2 (a) above).

If $A_1<B$, according to Lemma 2, the equation above becomes

$$2^N\cdot A_1 = B\cdot q + \delta$$

If $A_1>B$, according also to Lemma 2, the equation becomes

$$2^{N-1}\cdot A_1 = B\cdot q + \delta$$

In both cases, $q\in[2^{N-1}, 2^N)\cap\mathbf{Z}$.

We can solve the diophantine equations above for δ = 1, -1,2,-2,3,-3, … and B fixed. The method we applied was to use Euclid's algorithm to determine the greatest common divisor of 2^k and B, gcd (2^k, B) (where k=N or k=N-1), and to express it as a linear combination of B and 2^k. This yields a base solution, from which all the admissible solutions can be derived (see [5] for using Euclid's algorithm to determine above solutions). The most difficult cases are obtained for δ=1 and δ=-1. The number of solutions is very large (unlike for the square root), which makes it impractical to verify them all for a given divide algorithm. For example, for N=24, $A_1<B$, and δ=1, the number of solutions is 1289234. One example is

$$0.a49d25_H \;/\; 0.ff7e75_H = .a49e2300000100018b\ldots_H$$

which is very close to, but greater than, $.a49e23_H$.

Consider next the case of rounding to nearest. If $a/b \notin \mathbf{F}_N$ (it is not representable as a floating-point number with N bits in the significand), then how close can it be to a midpoint between two consecutive floating-point numbers in $\mathbf{F}_N$? This can be re-formulated as "how close can A/B be to an N-bit integer number plus 1/2?" To find out, we have to solve the equation

$$A/B = q + 1/2 + \delta/B$$

for increasing values (in absolute value) of δ, $2\cdot\delta\in\mathbf{Z}^*$, and for $q\in[2^{N-1},2^N)\cap\mathbf{Z}$. We can solve the diophantine equations above for δ = 1/2,-1/2,1,-1,3/2,-3/2, … and B fixed. The method applied was similar to that described for the other three rounding modes. The most difficult cases for rounding are obtained for δ = 1/2 and δ = -1/2, when B is odd. The number of solutions is again very large. For example, for N = 24, $A_1<B$, and δ = -1/2, the number of solutions is 1285649. One example is

$$0.c8227b_H \;/\; 0.e73317_H = .dd9a537fffff724506a\ldots_H$$

which is very close to, but less than, $.dd9a53_H$ + 1/2 *ulp*.

For N=24, $A_1 < B$, and $\delta=1/2$, the number of solutions is 1287219. An example is

$$0.ac1228_H \; / \; 0.b461d1_H = .f43467800000b5a8\ldots_H$$

which is very close to, but greater than, $.f43467_H+1/2$ *ulp*.

The method of determining values of the arguments a and b that generate values of a/b that are difficult to round is also useful in generating good quality test vectors for any implementation of the floating-point divide operation (not just for iterative algorithms), that can be added to others chosen by different criteria. In addition, this method can help in proving IEEE correctness of certain iterative algorithms for calculating the result of the floating-point divide operation, as shown below.

General Properties

Two main properties, Theorems 4 and 5, are used in the proposed method of proving the IEEE correctness of the result generated by iterative floating-point divide algorithms. They are expressed in terms of the scaled values A and B of the arguments a and b, as explained by Lemma 2 above.

Theorem 4. Let $a,b\in \mathbf{F}_N$, $a=\sigma_a\cdot s_a\cdot 2^{e\_a}$, $b=\sigma_b\cdot s_b\cdot 2^{e\_b}$. If $a/b\notin \mathbf{F}_N$, and A and B are determined by scaling a and b respectively as specified in Lemma 2, then for any $f\in \mathbf{F}_N$, and for any integer $F\in [2^{N-1},2^N)$

(a) The distance $w_{A/B}$ between A/B and F satisfies
$$w_{A/B} = | A/B - F | > 1/2^N$$

(b) The distance $w_{a/b}$ between a/b and f satisfies
$$w_{a/b} = | a/b - f | > 2^{e\_a-e\_b-2N}, \text{ if } s_a < s_b, \text{ and}$$
$$w_{a/b} = | a/b - f | > 2^{e\_a-e\_b-2N+1}, \text{ if } s_a > s_b$$

This theorem states that if a/b is not representable as a floating-point number with an N-bit significand, then there are exclusion zones of known minimal width around any floating-point number, within which a/b cannot exist. The minimal width of the exclusion zones cannot be easily extended in this case. If we try to gradually eliminate the difficult cases for rounding toward negative infinity, positive infinity, or zero (starting with the most difficult cases), we find too many cases to make it practical to verify directly.

Theorem 5. Let $a,b\in \mathbf{F}_N$, $a=\sigma_a\cdot s_a\cdot 2^{e\_a}$, $b=\sigma_b\cdot s_b\cdot 2^{e\_b}$. If $a/b\notin \mathbf{F}_N$, and A and B are determined by scaling a and b respectively as specified in Lemma 2, then for any $m\in \mathbf{F}_{N+1}-\mathbf{F}_N$ (midpoint between two consecutive floating-point numbers in $\mathbf{F}_N$), and for any M, integer+1/2$\in [2^{N-1},2^N)$

(a) The distance $w_{A/B}'$ between A/B and M satisfies
$$w_{A/B}' = | A/B - M | > 1/2^{N+1}$$

(b) The distance $w_{a/b}'$ between a/b and m satisfies
$$w_{a/b}' = | a/b - m | > 2^{e\_a-e\_b-2N-1}, \text{ if } s_a < s_b, \text{ and}$$
$$w_{a/b}' = | a/b - m | > 2^{e\_a-e\_b-2N}, \text{ if } s_a > s_b$$

This theorem states that if a/b is not representable as a floating-point number with an N-bit significand, then there are exclusion zones of known minimal width around any midpoint between two consecutive floating-point numbers, within which a/b cannot exist. Just as in the case of Theorem 4, the minimal width of the exclusion zones cannot be easily extended.

This property is used, as explained below, in conjunction with that in the preceding Theorem 4.

IEEE Correctness of Iterative Floating-Point Divide Algorithms

In order to prove that the result of an iterative algorithm for calculating the quotient a/b is IEEE correct, we have to show that the result R* before rounding and the exact value of a/b lead, through rounding, to the same floating-point number. If the result R* before rounding and the exact a/b are closer to each other than half of the minimum width of any exclusion zone determined as shown in Theorems 4 and 5, then $(R^*)_{rnd} = (a/b)_{rnd}$ for any IEEE rounding mode *rnd*. Figure 5 above, which illustrates this idea for the square root, is applicable in this case too.

As in Theorem 4 (a), $1/2^N = 1/2^N$ *ulp* in $\mathbf{F}_N = 1$ *ulp* in $\mathbf{F}_{2N}$, and in Theorem 5 (a), $1/2^{N+1} = 1/2^{N+1}$ *ulp* in $\mathbf{F}_N = 1$ *ulp* in $\mathbf{F}_{2N+1}$, the above can be expressed in the following corollaries of these two theorems:

Corollary 4. If $a,b\in \mathbf{F}_N$, $R^*\in \mathbf{R}$ and $|R^*-a/b| \le 1$ *ulp* in $\mathbf{F}_{2N}$, then $(R^*)_{rnd} = (a/b)_{rnd}$ for any rounding mode $rnd \in \{rm, rp, rz\}$

Corollary 5. If $a,b\in \mathbf{F}_N$, $R^*\in \mathbf{R}$ and $|R^*-a/b| \le 1$ *ulp* in $\mathbf{F}_{2N+1}$, then $(R^*)_{rn} = (a/b)_{rn}$

In practice, the inequalities to be verified are

$$|R^* - a/b| \le (w_{a/b})_{min}$$
$$|R^* - a/b| \le (w_{a/b}')_{min}$$

The method of proving the IEEE correctness of the result of an iterative algorithm that calculates the quotient of two floating-point numbers can then be summarized as a sequence of steps:

Step 1. Evaluate the relative error ε of the final result before rounding:

$$R^* = a/b \cdot (1+\varepsilon)$$

where an upper bound for $|\varepsilon|$ is known as $|\varepsilon| \leq 2^{-p}$, for some given $p \in \mathbf{R}$, $p > 0$.

Step 2. Evaluate $|R^* - a/b| = |(a/b) \cdot \varepsilon| \leq |a/b| \cdot 2^{-p}$. If the following hold

$$|a/b| \cdot 2^{-p} \leq (w_{a/b})_{min}$$

$$|a/b| \cdot 2^{-p} \leq (w_{a/b}')_{min}$$

then the algorithm generates IEEE-correct results for any input values a and b.

The method presented above was applied to software-based iterative algorithms for calculating the quotient of two floating-point numbers, when sufficient extra precision existed for the intermediate computational steps to allow the conditions above to be satisfied. Whenever this is not possible, alternative methods, such as the one presented in [3], have to be used.

It is worth mentioning that Corollaries 4 and 5 above were also obtained independently, starting from the following Lemma:

Lemma 3. Let $a,b \in \mathbf{F}_N$, $b \neq 0$. Then

(a) Either $a/b \in \mathbf{F}_N$ (the exact a/b is representable with N bits in the significand), or a/b is periodic.

(b) If $a/b \notin \mathbf{F}_N$ (it is periodic), then its infinite representation in binary cannot contain more than N-1 consecutive zeroes past the first binary one, and it cannot contain more than N-1 consecutive ones.

The Effect of the Limited Exponent Range

A note should be made here regarding the effect of the limited exponent range in iterative algorithms. If we analyze the algorithm by assuming valid inputs in a given floating-point format, and a certain precision (significand size) is ensured for all the computational steps, we may draw the conclusion (possibly using methods such as those presented above) that the final result is IEEE correct. It is possible though for an intermediate computational step to overflow, underflow (due to the limited exponent range), or to lose precision (due to the limited exponent range and to the limited significand size). Such cases have to be identified by carefully examining every computation step and, if they exist, the input operands that can lead to them have to be singled out. For such operands, alternate software algorithms are expected to be used in order to generate the IEEE-correct results, most likely by appropriate scaling of the input values and corresponding scaling back of the results.

Results

There are two main results of the work described in this paper.

First, a new method of proving IEEE correctness of the result of iterative algorithms that calculate the square root of a floating-point number was devised. This included establishing theoretical properties that also allowed determination of difficult cases for any of the four IEEE rounding modes, and for any desired precision. The method proposed for proving the IEEE correctness was successfully used for several algorithms that calculate the square root of single precision, double precision, and double-extended precision floating-point numbers.

Second, a parallel was drawn for iterative algorithms that calculate the quotient of two floating-point numbers. Again, the theoretical study allowed determination of difficult cases for rounding for any precision (single, double, or double-extended). The method proposed for proving the IEEE correctness of the result was successfully used this time only for algorithms that calculate the quotient of two single precision floating-point numbers. The reason for this limitation was clearly identified.

Overall, the study helped to demonstrate that efficient, IEEE-correct, software-based algorithms can compete with hardware-based implementations for floating-point square root and divide.

Discussion

The methods proposed are general and can be applied to any iterative algorithm that implements the floating-point square root or divide operation. Their usefulness was verified for software-based algorithms.

The method proposed for proving the IEEE correctness of the result for the floating-point square root operation should work for any iterative algorithm. Based on this method, it is possible to completely automate the verification process, and to build an IEEE correctness checker that would take as input the algorithm to be verified, together with the precision and rounding mode for every step. The method proposed for proving the IEEE correctness of the result for the floating-point divide operation will only work if there is sufficient extra precision in the intermediate calculations with respect to the precision of the final result. This will hold mostly for single-precision computations, but might also be true for higher precisions, depending on the particular implementation.

Methods to generate cases of difficult operands for rounding were also described, which are independent of any particular algorithm. They can be used to generate test vectors for any desired precision.

Conclusion

The two goals stated in the beginning of this paper were reached: a methodology for proving the IEEE correctness of the results for iterative algorithms that implement the square root, divide, and remainder operations was established, and operands that lead to results that constitute difficult cases for rounding were identified. This work is important to Intel Corporation because it proves the existence and viability of alternatives to the hardware implementations of these operations.

Notation

Some of the mathematical notations used in this paper are given here for reference:

N	the set of natural numbers
Z	the set of integer numbers
Q	the set of rational numbers
R	the set of real numbers
$\mathbf{F}_N$	the set of floating-point numbers with N-bit significands and unlimited exponent range
$\mathbf{F}_{M,N}$	the set of floating-point numbers with M-bit exponents and N-bit significands
$a \in \mathbf{S}$	a is a member of the set **S**
$\mathbf{S} \cap \mathbf{T}$	the intersection of sets **S** and **T**
$\mathbf{S} \cup \mathbf{T}$	the union of sets **S** and **T**
$\mathbf{S} \subset \mathbf{T}$	the set **S** is a subset of set **T**
$\mathbf{S} - \mathbf{T}$	the difference of sets **S** and **T**
$[l,h]$	the set of real values x, $l \le x \le h$
(l,h)	the set of real values x, $l < x < h$
$x \cong y$	the real number x is approximately equal to the real number y
$p \Leftrightarrow q$	statement p is equivalent to statement q
$m \equiv n \pmod p$	integers m and n yield the same remainder when divided by the positive integer p

In all the cases of sets, an asterisk (*****) indicates the absence of the value 0, e.g., $\mathbf{R}^* = \mathbf{R} \setminus \{0\}$.

Acknowledgments

The author thanks Roger Golliver from Intel Corporation, Peter Markstein from Hewlett-Packard Company, and Marius Dadarlat from Purdue University for their support, ideas, and/or feedback regarding parts of the work presented in this paper.

References

[1] ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic, IEEE, New York, 1985.

[2] Wolfram, S., Mathematica – A System for Doing Mathematics by Computer, Adison-Wesley Publishing Company, 1993.

[3] Markstein, P., Computation of Elementary Functions on the IBM RISC System/6000 Processor, *IBM Journal*, 1990.

[4] *Pentium® Pro Family Developer's Manual*, Intel Corporation, 1996, pp. 11-152 to 11-154.

[5] Stark, H. M., *An Introduction to Number Theory*, The MIT Press, Cambridge MA, 1995, pp. 16-50.

Author's Biography

Marius Cornea-Hasegan is a staff software engineer with Microcomputer Software Labs at Intel Corporation in Hillsboro, OR. He holds an M.Sc. in Electrical Engineering from the Polytechnic Institute of Cluj, Romania, and a Ph.D. in Computer Sciences from Purdue University, in West Lafayette, IN. His interests include mainly floating-point architecture and algorithms, individually or as parts of a computing system, and formal verification. His e-mail address is marius.cornea@intel.com.

Demystifying Multimedia Conferencing Over the Internet Using the H.323 Set of Standards

James Toga, Emerging Products Division, Intel Architecture Labs, Intel Corporation
Hani ElGebaly, Emerging Products Division, Intel Architecture Labs, Intel Corporation

Index words: H.323, conferencing, Internet, multimedia, gatekeeper

Abstract

The Telecommunication Sector of the International Telecommunication Union (ITU-T) has developed a set of standards for multimedia conferencing over packet-based networks. These standards are aggregated under a standard umbrella termed Recommendation H.323. Recommendation H.323 describes terminals, equipment, and services for multimedia communication over networks such as the Internet.

H.323 terminals and equipment carry real-time voice, video, and data, in any combination thereof. Terminals signal calls using Q.931-derived procedures defined in Recommendation H.225.0. After the call signaling phase, terminals proceed to the call control phase where they exchange capabilities and logical channel information using the H.245 protocol defined in Recommendation H.245. Once the call has been established, audio and video (if supported) are initiated. Both media types use the Real Time Protocol (RTP) defined by the Internet Engineering Task Force (IETF) as their Transport Protocol. Procedures for audio and video packet format and transport are described in Recommendation H.225.0. Recommendation H.323 allows for the use of a variety of video codecs (e.g., H.261, H.263, H.263+) and audio codecs (e.g., G.711, G.723.1). Data collaboration is also allowed using Protocol T.120. We provide an overview of these protocols and explain the H.323 call scenario.

Other entities such as gatekeepers, Multipoint Control Units (MCUs), and gateways are also addressed in Recommendation H.323. These entities allow network management, centralized multipoint, and interoperability with other conferencing standards. We explain each of these entities briefly and provide some scenarios of their interaction with the H.323 terminals.

IP telephony has become an important driver for packet-based communications. We address the role of H.323 procedures in deploying IP telephony, and describe how new H.323 features such as supplementary services and security facilitate this purpose.

Introduction

Recommendation H.323 [8] describes the procedures for point-to-point and multipoint audio and video conferencing over packet-switched networks. In addition to video conferencing terminals, Specification H.323 describes other H.323 entities including gateways, gatekeepers, and MCUs. Gateways allow interoperation of H.323 systems with other audio/video conferencing systems on integrated services digital networks (ISDN), plain old telephone systems (POTS), asynchronous transfer mode (ATM), and other transports. Gatekeepers provide admission control and address translation to H.323 endpoints. MCUs can engage more than two H.323 endpoints in a centralized multipoint conference.

Recommendation H.323 comprises a number of related documents that describe terminals, equipment, services, and interactions. Examples of other core standards that are referred to in Recommendation H.323 are H.225.0 [5] (procedures for call signaling, media packet format, and synchronization); H.245 [7] (procedures for capability exchange, channel negotiation, and flow control); H.450.x [11] (procedures for supplementary services); H.246 [8] (procedures for terminals' interoperability through gateways); and H.235 (security and encryption procedures). Other referenced standards include media codecs for audio (such as, G.711, G.723.1, G.729, and G.722) and video (such as, H.261 and H.263).

The purpose of this paper is to present an overview of the H.323 core components and functionality and the current industry trends with respect to H.323, such as IP telephony and corporate conferencing. Towards this goal, we briefly describe the H.323 architecture, the responsibilities of H.323 entities, and basic call scenarios. The main obstacles to the adoption and deployment of the H.323 standard in industry are also explained. Finally, H.323 zone management, multipoint operation, telephony services features, and security are highlighted.

H.323 Architecture

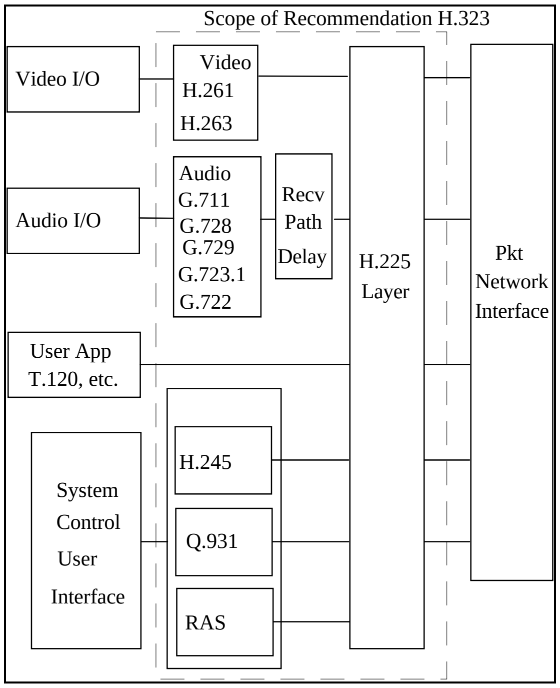

Figure 1: H.323 architecture

The architecture of an H.323 terminal is shown in **Figure 1**. The scope of Recommendation H.323 is limited to the definition of the media compression standard, packet format, signaling, and flow control. Media capturing such as video capturing schemes, audio recording, or user data applications are outside the scope of Recommendation H.323. Initial H.323 implementations targeted IP networks; however, Recommendation H.323 supports alternative transports such as IPX. The Transport layer carries control and media packets over the network; there are no specific requirements for the underlying transport except for support for reliable and unreliable packet modes. An example of a well-known transport is User Datagram Protocol (UDP) over Internet Protocol (IP).

The video codec (e.g., H.261, etc.) encodes the video from the video source (i.e., camera) for transmission and decodes the received video code that is output to a video display. The mandatory video codec for an H.323 terminal is H.261 [3] with quarter common intermediate format (QCIF) resolution. (QCIF is a video picture size.) Other codecs such as H.263 may be supported. H.263 [4] has better picture quality and more options. A terminal can also support other picture sizes such as CIF and SQCIF. During the terminal capability exchange phase, the video codecs, resolution, bitrate, and algorithm options are exchanged between terminals, using the H.245 protocol. Terminals can open channels only with parameters and options chosen from the intersection of the capability sets. In general, the receiver always specifies what the transmitter may send.

The audio codec (G.711, etc.) encodes the audio signal from the microphone for transmission and decodes the received audio code that is output to the loudspeaker. G.711 0 is the mandatory codec for an H.323 terminal. A terminal may be capable of optionally encoding and decoding speech using Recommendations G.722, G.728, G.729, MPEG1 audio, and G.723.1. Since G.711 is a high-bitrate codec (64Kb/s or 56Kb/s), it cannot be carried over low-bitrate (< 56 kbps) links. G.723.1 [2] is the preferred codec in this situation because of its reasonably low rate (5.3Kb/s and 6.4 Kb/s). H.323 terminals open logical channels using a common capability that is supported by all entities and exchanged during the H.245 capability exchange phase.

The Data Channel supports telematic applications such as electronic whiteboards, still image transfer, file exchange, database access, audiographics conferencing, etc. The standardized data application for real-time audiographics conferencing is T.120. Other applications and protocols may also be used via H.245 negotiation such as chatting and fax.

The System Control Unit (H.245, H.225.0) provides signaling and flow control for proper operation of the H.323 terminal. H.245 [7] is the media control protocol that allows capability exchange, channel negotiation, switching of media modes, and other miscellaneous commands and indications. The H.225 standard describes the Call-Signaling Protocol used for admission control and for establishing connection between two or more terminals. The Connection Establishment Protocol is derived from the Q.931 specification [12]. The H.225.0 [5] Layer also formats the transmitted video, audio, data, and control streams into messages for output to the network interface, and it retrieves the received video, audio, data, and control streams from messages that have been input from the network interface, using the Real Time Transport Protocol (RTP) and its companion, the Real Time Control Protocol (RTCP). The RTP performs logical framing, sequence numbering, timestamping, payload distinction, source identification, and occasionally, error detection and correction as appropriate to each media type. The RTCP provides reporting and status that may be used by both senders and receivers to correlate performance on the media streams.

H.323 Entities and Responsibilities

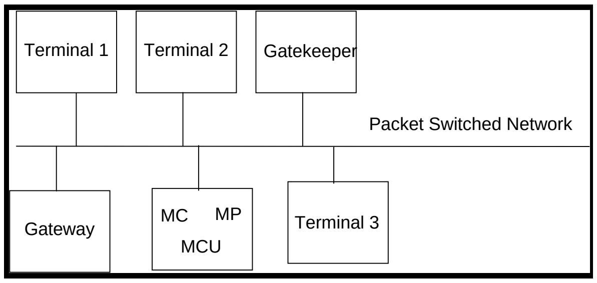

Figure 2: H.323 entities

Figure 2 shows the different entities that can be involved in an H.323 conference. A gatekeeper is an optional element in an H.323 conference that provides call control services to the H.323 endpoints such as address translation, admission control, bandwidth control, and zone management. The gatekeeper may also provide other optional functions such as call authorization and call accounting information. A gatekeeper cannot be specified as a destination in a call. Address translation is the method by which an alias address (e.g., e-mail address) is translated to a transport address. Admission control is a way of limiting H.323 access to a network, or to a particular conference using Request, Admission, and Status (RAS) messages defined in Recommendation H.225.0. A gatekeeper also manages bandwidth allocation to H.323 endpoints using RAS messages. Zone management defines the scope of entities over which a gatekeeper has control. These include endpoints, gateways, and MCUs. In addition, a gatekeeper can allow secure access to the conference using various authentication mechanisms. Q.931 and H.245 messages can be routed through the gatekeeper, and statistical information about the calls in progress can be collected. Gatekeepers may also perform telephony service operations such as call forwarding and call transfer.

H.323 endpoints can interact with each other directly in a point-to-point or multipoint conference if no gatekeeper is present. When a gatekeeper is present, all endpoints have to register with it.

A gateway operates as an endpoint on the network that provides real-time, two-way communication between H.323 terminals on the packet-based network and other ITU terminals on a switched-circuit network, or to another H.323 gateway. Other ITU terminals include those complying with Recommendations H.310 (B-ISDN), H.320 (ISDN), H.321 (ATM), H.322 (GQoS-LAN), H.324 (GSTN), H.324M (Mobile), and POTS. The gateway provides the appropriate translation between transmission formats (for example, H.225.0 of an H.323 endpoint to/from H.221 of an H.320 endpoint) and between communication procedures (for example, H.245 of an H.323 endpoint to/from H.242 of an H.320 endpoint). This translation is specified in Recommendation H.246. The gateway also performs call setup and clearing on both the network side and the Switched-Circuit Network (SCN) side. Translation between video, audio, and data formats may also be performed in the gateway. In general, the purpose of the gateway is to complete the call in both directions between the network endpoint and the SCN endpoint in a transparent fashion.

The MCU is an endpoint on the network, which provides the capability for three or more terminals or gateways to participate in a multipoint conference. It may also connect two terminals in a point-to-point conference, which may later develop into a multipoint conference. The MCU consists of two parts: a mandatory Multipoint Controller (MC) and optional Multipoint Processors (MP). The MC provides the capability for call control to negotiate with all terminals to achieve common levels of communication. It is this element that is required for all multipoint conferences. The MP allows mixing, switching, or other processing of media streams under the control of the MC. The MP may process a single media stream or multiple media streams depending on the type of conference supported. In the simplest case, an MCU may consist only of an MC with no MPs.

The following section provides an overview of the basic operation of an H.323 endpoint in a point-to-point conference without a gatekeeper and then with a gatekeeper.

H.323 General Operation

Figure 3 shows call establishment and tear down steps between two H.323 endpoints without a gatekeeper. All of the mandatory Q.931 and H.245 messages exchanged are listed. Note that some of these messages may be overlapped for increased performance. Each message has an assigned sequence number at the originating endpoint. Endpoint A starts by sending a *Setup* message (1) to endpoint B containing the destination address. Endpoint B responds by sending a Q.931 *Alerting* message (2) followed by a C*onnect* message (3) if the call is accepted. At this point, the call establishment signaling is complete, and the H.245 negotiation phase is initiated. Both terminals will send their terminal capabilities (4) in the *terminalCapabilitySet* message. The terminal capabilities include media types, codec choices, and multiplex information. Each terminal will respond with a *terminalCapabilitySetAck* (5) message. The terminals' capabilities may be resent at any time during the call. The Master/Slave determination procedure (6-8) is then started. The H.245 Master/Slave determination procedure is used to resolve conflicts between two endpoints that can both be the MC for a conference, or between two endpoints that are attempting to open bi-directional channels at the same time. In this procedure, two endpoints exchange random numbers in the H.245 *masterSlaveDetermination* message to determine the master and slave endpoints. H.323 endpoints are capable of operating in both master and slave modes. Following master/slave determination procedures, both terminals proceed to open logical channels (9-10). Both video and audio channels are unidirectional while data is bi-directional. Terminals may open as many channels as they want. Only one logical channel is shown in **Figure 3**, yet the same procedure applies if more channels are opened.

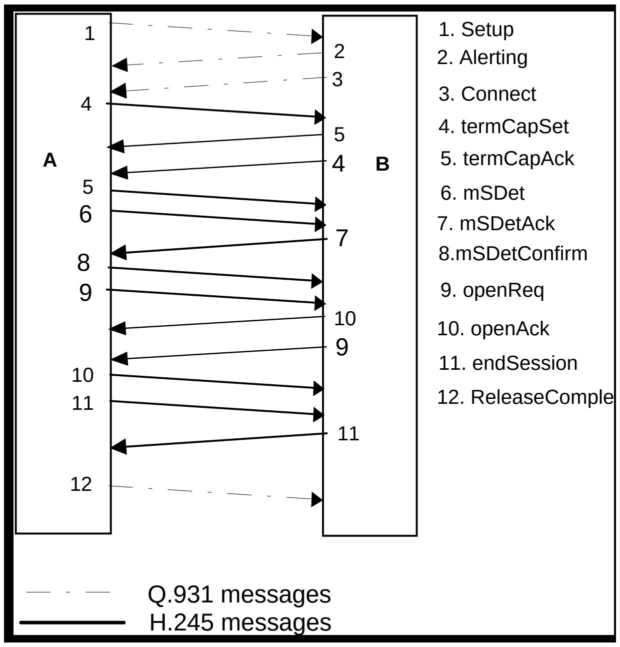

Figure 3: Q.931 and H.245 messages exchanged between two H.323 endpoints

Other H.245 control messages may be exchanged between the endpoints to change media format, request video key frames, change the bitrate, etc.

Termination of a call is initiated by one endpoint sending an *endSession* message (11). Endpoint B, on receiving the *endSession* command, will respond with another *endSession* message (11) to Endpoint A. Endpoint A will finally send a Q.931 *ReleaseComplete* message (12), and the call is terminated.

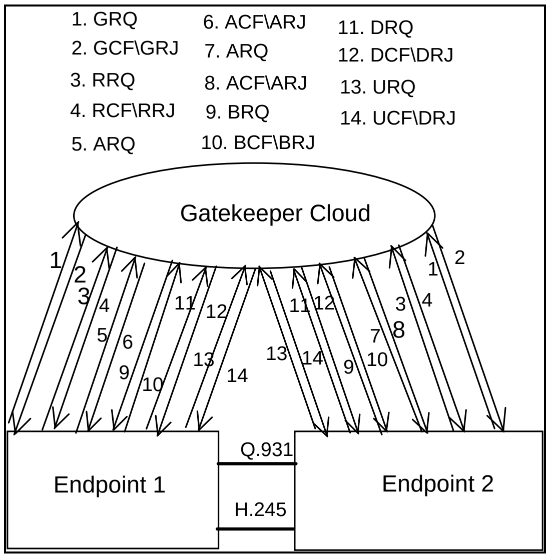

Figure 4: Gatekeeper interaction in an H.323 call

Figure 4 shows messages exchanged between a gatekeeper and an H.323 endpoint. Before the conference starts, both endpoints look for a gatekeeper by multicasting a *GatekeeperDiscovery* (GRQ). Request. The gatekeeper will reply either with a *GatekeeperConfirm* (GCF) message or with a *GatekeeperReject* (GRJ) message. Both endpoints will then register their alias names with the gatekeeper using the *RegistrationRequest* (RRQ) message. The gatekeeper will acknowledge by sending a *RegistrationConfirm* (RCF) message or will deny it using a *RegistrationReject* (RRJ) message. Registering alias names with the Gatekeeper allows endpoints to call each other using user-friendly addresses such as e-mail, etc., rather than the transport address. The discovery and registration procedure is valid until the gatekeeper indicates otherwise.

An endpoint or gatekeeper can request the location of another endpoint using its alias name by using a *LocationRequest* (LRQ) message, and the gatekeeper replies with a *LocationConfirm* (LCF) message containing the resolved address for the alias name.

When a user places a call from an endpoint, the endpoint starts by requesting admission from the gatekeeper using an *AdmissionRequest* (ARQ) message. The gatekeeper can accept (ACF) or deny the request (ARJ). If the call is accepted, the endpoint sends a Q.931 *Setup* message to the remote destination. The recipient of the *Setup* message in turn requests admission from its gatekeeper by sending an ARQ. When the call is accepted, the Q.931 call signaling sequence is completed followed by the H.245 message negotiation. The *AdmissionRequest* (ARQ) message carries the initial bandwidth the endpoint requires for the duration of the conference. If during H.245 logical channel negotiation, an endpoint requires more bandwidth, it issues a *BandwidthRequest* (BRQ) message to the gatekeeper. If the request is accepted, the gatekeeper replies with a *BandwidthConfirm* (BCF) message; otherwise, it replies with a *BandwidthReject* (BRJ) message.

When the call is terminated, both endpoints send a *DisengageRequest* (DRQ) message to inform the gatekeeper that a call is being terminated. The gatekeeper replies with a *confirm* (DCF) or *reject* (DRJ) message. Alternatively, endpoints may unregister from the gatekeeper by sending an *UnregisterRequest* (URQ) message. The gatekeeper replies with an *UnregisterConfirm* (UCF) message or an *UnregisterReject* (URJ) message.

H.323 Deployment Obstacles

In order to achieve H.323 deployment in real networks, limitations at both the network level and the client platform level must be resolved at the H.323 client. The client should scale performance based on the available bandwidth. Given the inconsistencies of networks with best effort traffic, (i.e., no guaranteed Quality of Sevice (QoS), the Internet), it is extremely important to provide mechanisms for fault tolerance and error resiliency at the client platform, if it is to be used on an unmanaged network such as the Internet. At the network level, broad connectivity, policy management, and security are considered the major issues in the deployment of a new technology such as H.323. Switched Circuit Network connectivity is achieved in the H.323 context by using gateways for H.320, H.324, H.323, POTS, and other endpoints on other networks. Policy management is achieved using gatekeepers to provide call admission, authentication, and zone management. Deployment of QoS protocols such as RSVP can also help with policy management. The security of media streams is another important factor in the success of H.323 deployment, especially in unsecured environments such as the Internet.

The platform consists of two main components: the operating system and the CPU. Many media compression algorithms are currently limited by the machine speed. (We expect this issue to improve as more powerful processors hit the market.) Moreover, popular desktop operating systems do not supply consistent real-time services. Multitasking can adversely affect the quality of audio and video in an H.323 conference.

H.323 Applications

A number of applications can take advantage of H.323 technology both in the corporate environment and in the home-user environment. One obvious application is video conferencing between two or more users on the network. In this application, the user is expecting the same services as those provided by a telephone. The quality of service should be equal to or better than POTS. The Internet does not appear to be readily addressing these issues; however, there is work in progress at the corporate Intranet level to improve the quality of multimedia communication.

Multimedia call centers are another application for H.323. An H.323 call center provides a well-integrated environment for Web access and other data/voice business situations. The call centers are used by banks for customer service, shops for extra retail outlets, etc. The call center can just be an H.323 terminal or an MCU, or it can be a full featured endpoint with a gatekeeper, a gateway, and MCU capability. The front end of the legacy call center may be a gateway, which allows installed systems to operate with minimal disruption.

Another compelling application for H.323 is telecommuting. Telecommuters can attend meetings at their companies, check their mail, or talk with someone at the company while on the road or at home.

Finally, IP telephony is another area in which H.323 has found a significant role; this will be covered in a later section.

H.323 Zone Management

Gatekeepers fulfill a required set of operational responsibilities and may offer a number of optional functions to entities within their *zone.* Before we describe how zones are managed, we will review some of these responsibilities and functions.

A gatekeeper acts as a monitor of all H.323 calls within its zone on the network. It has two main responsibilities: call approval and address resolution. An H.323 client that wants to place a call can do so with the assistance of the gatekeeper. The gatekeeper provides the address resolution to the destination client. (This division of work is due to alias name registration procedures.) During this address resolution phase, the gatekeeper may also make permissioning decisions based upon available bandwidth. The gatekeeper can act as an administrative point on the network for IT/IS managers to control H.323 traffic on and off the network.

Strictly speaking, a gatekeeper zone is defined by what it contains: it is defined by all of the endpoints, gateways, and MC(U)s that are or will be registered with a gatekeeper. Another way to describe a gatekeeper zone is to call it an "administrative domain," although the formal ITU-T recommendation text uses zone. An example of gatekeeper zones is given in **Figure 5**.

Some important aspects of zone coverage are summarized as follows:

- Zones are defined by all H.323 devices registered to a single gatekeeper.
- Zone design may be independent of physical topology.
- Each zone has only one gatekeeper.
- Zone definition is implementation-specific.
- Gatekeeper zones are logical in nature.
- Network topology and administrative scope are both factors in zone design.
- Resources such as gateways and proxies may affect the partitioning of zones.

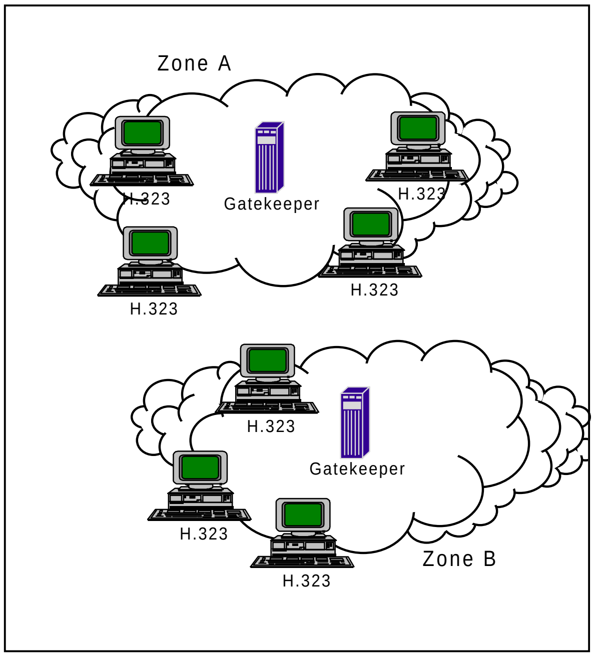

Figure 5: Gatekeeper zones

Multipoint Conferences: Centralized Versus Decentralized

Multipoint conferences are defined as calls between three or more parties. During these conferences, call control and media operations can become considerably more complex than during a simple point-to-point conference. The coordination and notification of participants entering and leaving a conference along with the marshalling of the media streams require the presence of at least a Multipoint Control (MC). In other situations, an MCU is required; we will explain why in the next section.

The two broad models of multipoint, centralized and decentralized, differ in their handling of real-time media streams (audio and video). The centralized model operates in the same fashion as other circuit-based conferencing (e.g., H.320 or telephony "bridges"). In this model, all of the audio and video is transmitted to a central MCU that mixes the multiple audio streams, selects the corresponding video stream, and re-transmits the result to all of the participants. **Figure 6** illustrates this procedure.

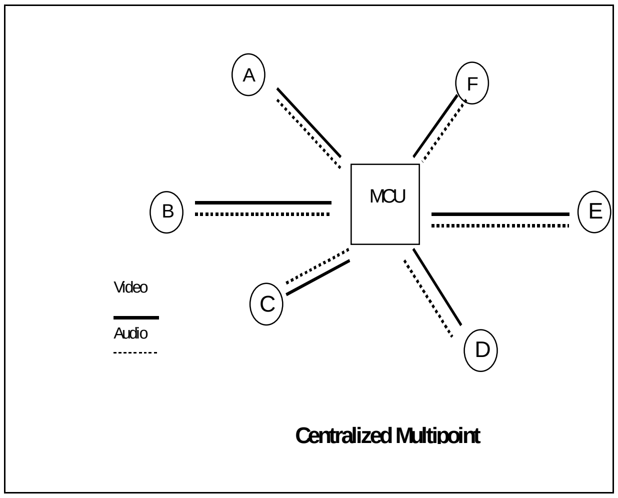

Figure 6: Centralized model

Note that the MCU is only 'logically' at the center of the conference configuration; the physical topology may be very different. The operational model of the conference is such that each endpoint is exchanging media control signaling (H.245), audio, and video directly with the MCU. In order to prevent echo, the MCU must provide the current speaker with a custom audio mix that does not contain the speaker's own audio. The remaining participants may all receive the same audio and video media. In some implementations, the MCU may actually decode the video streams and combine them in a mixing operation to provide continuous presence of all participants on the endpoint displays. The amount of processor speed required for this operation increases significantly.

The decentralized model shares common control characteristics with the centralized model, but the media streams are handled differently. One of the participating entities must be an MC, but it will typically be co-located with one of the endpoints. All of the H.245 connections will have one end terminating at the MC just as with the

MCU in the centralized model. Whereas the MCU does the media processing in the centralized model, the media streams are sent and received by all participating entities on a peer-to-peer basis in the decentralized model. As **Figure 7** demonstrates, the logical configuration for this is a bus. There is no MCU to process the multiple streams; each entity is responsible for its own audio mixing and video selection. The media may be sent between all entities utilizing either multicast, or a multiple uni-cast if the underlying network does not support multicast.

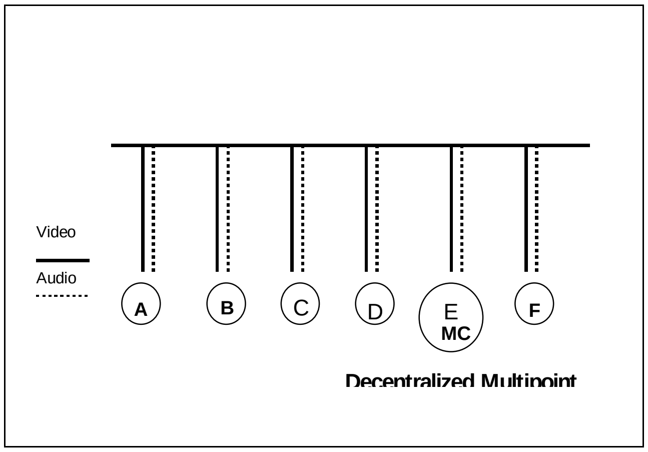

Figure 7: Decentralized model

There are a number of advantages and disadvantages to both the centralized and the decentralized models. In the centralized model, the endpoints that participate in this type of conference are not required to be as powerful as in the decentralized model. Each endpoint has only to encode its locally produced media streams and decode the set sent by the MCU. The MCU may provide specialized, or higher performance conferencing. The MCU's ability to provide custom-mixed media streams can allow otherwise constrained endpoints to participate in conferences. For example, if an endpoint is connected to the conference by a low-bitrate connection, it may only have the capacity to transmit and receive one media stream. If this same endpoint were connected to a decentralized conference, it would have no ability to detect which speaker was the focus when the focus changed. In most situations, the MCU will choose the largest common set of attributes within which to operate the conference. This may lead to a conference that is operated in a least common denominator mode, such as QCIF, rather than CIF or lower video resolution.

By definition, the decentralized multipoint model does not require the presence of an MCU, a potentially expensive and limited resource. It allows for disproportionate processing at the endpoints with each running at its own level. The decentralized multipoint model does require that one of the participating entities contain an MC. If the endpoint that has the MC leaves the conference, the MC must stay active or the conference is terminated. In order for this model to be bandwidth efficient, the underlying network should support multicast. If it doesn't, each endpoint can send multiple uni-cast streams to all others, but this becomes increasingly inefficient with more than four entities participating in a conference.

H.323 Features for IP Telephony

The initial environment envisioned for H.323 was the corporate network environment, primarily local area networks. Wide Area Network (WAN) access was to be gained by using gateways to H.320/ISDN. During the implementation of Revision 1 of H.323, it became clear that IP telephony was gaining popularity and relevance as various infrastructure elements were improved upon. A number of proprietary IP-based telephones were creating many small islands that could not communicate with one another. Recommendation H.323 provided a good basis for establishing a universal IP voice and multimedia communication in larger, connected networks. With Revision 2 of the Recommendation, new additions and further extensions were added specifically to make it more suitable for IP telephony. These changes will be described in a later section.

By using Q.931 as its basis for establishing a connection, H.323 allows for relatively easy bridging to the public switched telephone networks (PSTN) and circuit-based phones. The required voice codec of G.711 also allows for easy connections to the legacy networks of telephones. The uncompressed 64kb/sec stream can easily be translated between digital and analog media. One of the addressing formats provided in Recommendation H.323 is the E.164 address. This is another ITU Recommendation that specifies standard telephone numbers (e.g., the digits 0-9, * and #). These addresses, which ultimately map onto the IP addresses for the H.323 endpoints, allow regular telephones to 'dial' them. Gatekeepers provide the final important element for IP telephony. Gatekeepers supply the ability to have integrated directory and routing functions within the course of the call setup. These operations are important for real-time voice or video when resources must be balanced, and points of connectivity are highly dynamic. The gatekeeper functions, which provide call permissioning and bandwidth control, enable load monitoring, provisioning, and ultimately, commercial-grade IP telephony service.

Interoperability

Recommendation H.323 comprises a number of interrelated documents and sub-recommendations.

Therefore, customers are faced with a number of options, which they may or may not implement. These design choices, in conjunction with differing interpretations of the required elements, can lead to implementations that cannot interoperate. Stated more simply, compliance with the recommendations does not always imply interoperability. The complexity and flexibility of H.323 essentially requires that implementations be tested together to ensure interoperability.

To this end, the International Multimedia Technical Consortium (IMTC) has focussed largely on this area of interoperability. The testing events sponsored by this consortium have provided a venue for more than thirty companies to participate in "bake-offs" to test the interoperability of their product. The methodology that is followed provides for a gradual testing of component protocol stacks that then leads to higher level end-end scenario testing. **Figure 8** illustrates the grouping of the components. Each horizontal level provides increasing H.323 functionality, and each vertical group indicates a specific stack function that is required.

Much of the mass interoperability testing has occurred in face-to-face events. In theory, the H.323 protocol should allow for testing from remote sites across the Internet; in practice, however, remote testing has not turned out to be useful. The lack of a controlled environment makes distinguishing interoperability issues from simple network problems extremely difficult.

Examples for initial interoperability problems encountered during some of the testing events include misaligned bit field and byte-ordering issues. These problems are common in the early development stage of protocols. For example, there are a number of adaptations that are specified in the Q.931 protocol used by H.323, which make the Protocol Data Units (PDUs) slightly different from the protocol messages used in the circuit world (such as ISDN).

Porting of the existing Q.931 code to the packet environment provided mis-matches. During the initial course of development, there were a small number of defects in the Recommendation that were discovered by the implementers. These defects were recorded in a document called the *Implementer's Guide* and eventually corrected in the revision of the base Recommendation. In addition to the low-level control protocol interoperability, the media stream is another potentially problematic area. Recommendation H.245 allows the receiver to specify the maximum bitrate that may be sent, but currently there is no way in which to specify the maximum Real Time Protocol (RTP) packet size. This can lead to interoperability problems if a receiver implementation makes assumptions about the buffering required and then cannot decode the packet. Although H.323 specifies G.711 and H.261 as its baseline codecs (audio and video respectively), a large number of initial H.323 implementations were targeted for low-bitrate connectivity where these codecs could not operate. The result of this was that the audio codec, G.723.1, was chosen due to its low (5.3-6.4kb/s) data rate. This low bitrate allowed H.263 video to operate in an acceptable fashion across a 33.6kb connection.

Implementation choices that are made as a result of operating environments elevate the interoperability issues to the next level. At an operational level, endpoint implementations may select to support an asymmetrical media model for optimal performance; other implementations may not. The ability to enter extended information at the user interface may determine whether an endpoint can operate with certain gateways or H.323 proxies. Gatekeepers may or may not provide intelligent zone control, which allows them to operate in a network connected to other activated gatekeepers. With the addition of a range of security options to H.323, strict policy constraints may prevent H.323 entities from interoperating while still complying with all the recommendations.

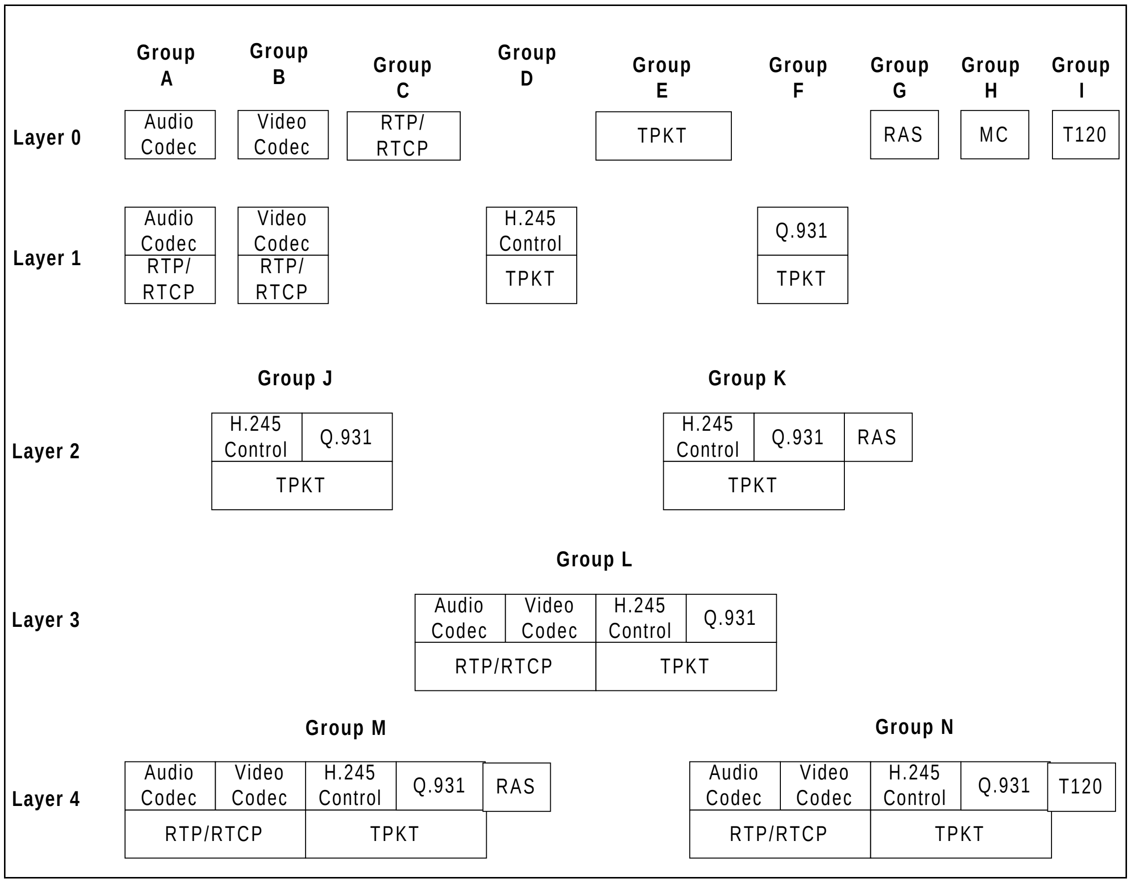

Figure 8: Testing matrix[1]

New Features

Revision 2 (1998) of Recommendation H.323 contains a number of improvements for IP Telephony, among other areas. A new Recommendation (H.235) was developed to provide a full security framework for H.323 and other multimedia systems. It may provide the services of Authentication (which can be used for authorization), Privacy, and Integrity. The system can utilize underlying security protocols such as Internet Protocol Security (IPSEC) or Transport Layer Security (TLS) as established in the IETF. A number of new features and options were introduced regarding interactions with gatekeepers. During the process of discovering gatekeepers, endpoints may actually receive a number of gatekeeper addresses to utilize. This redundancy in the protocol will eliminate the single point of failure if a gatekeeper becomes inoperative. When endpoints register their current address with a gatekeeper, they may specify multiple addresses; this allows a 'line hunting' mode of operation. Keep alive types of messages called Request In Progress (RIP) can stop premature retries on lengthy operations.

The call setup has a new method called FastStart that establishes bi-directional media in one round trip time of messages (discounting the establishment of the actual TCP connection). **Figure 9** shows FastStart messages exchanged. The *OpenLogicalChannels* messages that occur after H.245 is established in the regular case are piggy-backed on the Setup-Connect exchange. This facility allows instant audio connection that resembles the regular phone call model as opposed to the standard lengthy H.323 startup procedures.

[1] In this figure TPKT refers to the layer that segments a TCP stream into separate message 'packets' to be delivered to the H.323 application.

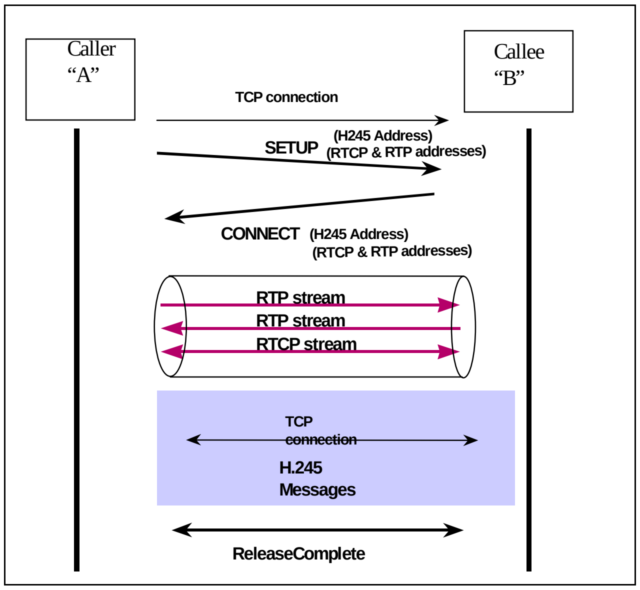

Figure 9: FastStart

Many new explicit address types have been added. including RFC822 (e-mail) formatted names and URLs, which network-based users find more familiar and easier to remember. Each endpoint may register under a number of these aliases. New unique call identifiers were added to better track calls as they traverse multiple network nodes and topologies. A number of new features were added to H.245 also, which allow for a much richer set of media possibilities. In addition to layered codecs, which divide up the video stream into additive layers of detail, Quality of Service parameters (e.g., RSVP) may now be signaled when opening up media streams. Finally, the ability to pass the media on other transports such as ATM (while keeping the rest of call and media control on IP) have been added.

In addition, a new family of recommendations has been developed to set up a framework for supplementary services. The H.450.1, H.450.2, and H.450.3 are loosely based on the Q-signals (QSIG) protocol, which is used by PBX and switch vendors. The initial services that have been defined are call transfer and call forward. Both of these features are a peer-peer option requiring no specific help from a centralized network entity (such as the PBX in the circuit world). A number of standardized supplementary services are expected to be developed in the future.

Along with the new features, the latest development of H.323 also includes an expanded topology model in the H.332 document. Recommendation H.332 allows an H.323 conference model to expand to literally thousands of participants. The tightly controlled H.323 'panel' is surrounded by a very large number of 'listeners.' Members may participate in the conference by joining the panel. They may also leave as they wish. This H.323 panel is analogous to a panel on a stage in a large auditorium. Occasionally participants might get up from the audience and join the panel and others might leave the panel and join the audience. This movement of participants occurs using the standard Q.931 Invite and Join signaling as described in the H.225.0 document.

Conclusion

The H.323 system and its associated recommendations provide a useful and flexible system for multimedia communication. The factors that allow the protocols to easily bridge data and voice networks also make H.323 scalable. The ability of most, if not all of the system, to operate on a general-use platform such as a personal computer allows the H.323 system to scale with underlying processing power. As the processor's speed budget increases, the H.323 system can provide a better end-user experience. The dynamic exchange of capabilities allows the communications to change if needed during a call and adapt to any environmental or endpoint constraints.

Recommendation H.323 has become the single standards-based solution for a complete array of communication systems from simple point-to-point telephony to a rich multimedia conference with data sharing. Through continued effort by the ITU-T study group, Recommendation H.323 continues to evolve and adapt to new situations. Many of the real world difficulties in utilizing H.323 come about from infrastructure issues or other problems that are being resolved. Globally coordinated addressing and consistent QoS are two areas where we expect to see great improvements in the future. Some of these improvements will be facilitated by expected higher level communications between gatekeepers and gateways as their interaction is standardized.

References

[1] ITU-T Recommendation G.711 (1988) "Pulse Code Modulation (PCM) of Voice Frequencies."

[2] ITU-T Recommendation G.723.1 (1996) "Dual Rate Speech Coders for Multimedia Communication Transmitting at 5.3 & 6.3 kb/s."

[3] ITU-T Recommendation H.261 (1993) "Video Codec for Audiovisual Services at p X 64 kb/s."

[4] ITU-T Recommendation H.263 (1996) "Video Coding for Low Bit-rate Communication."

[5] ITU-T Recommendation H.323 (1998) "Packet Based Multimedia Communications Systems."

[6] ITU-T Recommendation H.225.0 (1998) "Call Signaling Protocols and Media Stream Packetization for Packet Based Multimedia Communications Systems."

[7] ITU-T Recommendation H.245 (1998) "Control Protocol for Multimedia Communication."

[8] ITU-T Recommendation H.246 (1998) "Interworking of H-Series Multimedia Terminals."

[9] ITU-T Recommendation H.235 (1998) "Security and Encryption of H series (H.323 and other H.245 based) Multimedia Terminals."

[10] ITU-T Recommendation H.332 (1998) "Loosely Coupled H.323 Conferencing."

[11] ITU-T Recommendation H.450.1 (1998) "Generic Functional Protocol."

[12] ITU-T Recommendation Q.931 (1993) "Digital Subscriber Signaling System No. 1 (DSS 1)-ISDN User-Network Interface Layer 3 Specification for Basic Call Control."

Authors' Biographies

Jim Toga holds a B.Sc in Chemistry from Tufts University and a M.Sc in Computer Science from Northeastern University. Before joining Intel, he was the principal engineer on StreetTalk[*] Directory with Banyan Systems where he designed and developed the Yellow Pages service. Presently, he is a senior staff software architect for the Standards and Architecture Group in the Intel Architecture Labs. He coordinates product groups giving guidance on architecture and standards. His primary tasks are H.323/Internet Telephony, Directory, and real-time security issues.

Outside of Intel, Mr. Toga develops standards and standards-based products within ITU-T, IETF, and IMTC. He is also involved in the following related activities:

Editor of ITU-T H.323 Implementors Guide
H.235 Security Standard
Chair of IMTC "Packet Networking Activity Group"
Chair of H.323 Interoperability Group
He has also written numerous articles for trade magazines. His e-mail is jim.toga@intel.com.

Hani ElGebaly is the project technical lead for the H.323/H.324 protocols development team within the Conferencing Products Division of Intel Architecture Labs. He received an M.Sc. in Computer Science from the University of Saskatchewan, Canada, and a B.Sc. in Electrical Engineering from Cairo University, Egypt. Mr. ElGebaly is currently pursuing a Ph.D. with the University of Victoria, Canada. His primary areas of interest include multimedia conferencing protocols, embedded programming, fault tolerant systems, and computer architecture. His e-mail is hani.el-gebaly@intel.com.

[*] All other trademarks are the property of their respective owners.

Characterization of Multimedia Streams of an H.323 Terminal

Hani ElGebaly, Emerging Products Development, Intel Architecture Labs, Intel Corporation

Index words: H.323, Conferencing, Internet, G.723.1, H.263

Abstract

A study of multimedia performance over the Internet requires accurate representation of the workload model. Measurements of multimedia conferencing applications provide a snapshot of real workloads. This paper presents a characterization for video and audio traffic transported over the Internet by multimedia conferencing applications following the H.323 set of standards defined by the International Telecommunication Union (ITU-T).

Our goal is to investigate multimedia traffic multiplexing issues at the host. We are not concerned with the Internet infrastructure or the store and forward technology deployed. The traces collected in this study are done at the multimedia source host. They provide information about packet length, inter-arrival time, jitter, overhead, and burstiness.

These traces help in understanding local-induced effects on the multimedia traffic mix. Many of these effects are primarily due to limitations of operating systems, bandwidth sharing, or the choice of traffic parameters. These local effects contribute significantly to traffic delay or the abuse of network bandwidth and congestion.

A few tradeoffs are involved in the choice of multimedia traffic parameters. For example, a tradeoff exists between the network protocol overhead and the local latency of multimedia packets. Another tradeoff exists between the video packet size and the burstiness of the video packets. In this paper, the various tradeoffs involved in generating audio and video packets are discussed. We focus on the audio and video codecs defined in Recommendations G.723.1 and H.263 of the ITU, respectively. Both codecs are extensively utilized by H.323 developers because of their efficiency, popularity, and suitability for transmission over low bandwidth pipes. We derive an optimal operating point for both audio and video traffic for better bandwidth efficiency and acceptable latency.

We also discuss the interaction of conferencing media components as they are multiplexed on the host to be transmitted to a peer terminal. Lack of appropriate multiplexing algorithms can lead to one or more media components oversubscribing to the shared bandwidth and penalizing other participants. A new performance qualifier is introduced in this paper. This qualifier is the number of interleaved video bytes scheduled for transmission between audio packets. This number turned out to be a good local indicator for audio jitter and latency.

Introduction

Advances in computer technology, such as faster processors, faster modems, Intel MMX™ technology, and better data compression schemes have made it possible to integrate audio and video data into the computing environment. A new type of video conferencing is now possible: desktop video conferencing. Desktop video conferencing applications include telecommuting, corporate meetings to cut travel costs, family gatherings, Call Centers, etc. Banks, shopping centers, retail centers, and so on can be more efficient and cut a lot of overhead by providing video-conferencing customer service centers for customers to dial into.

One of the most prominent enabling technologies for desktop video conferencing is the H.323 standard developed by the ITU-T. Recommendation H.323.[6] describes terminals, equipment, and services for multimedia communication over networks such as the Internet.

This paper addresses issues of the multimedia traffic sources of an H.323 terminal. Issues such as packet format and multiplexing of audio and video frames at the host are studied. The traces collected in this study were done at the network edge. These traces provide information about the packet length, inter-arrival time, jitter, protocol overhead, and burstiness. Local-induced effects on the conferencing traffic can occur due to limitations of the operating system, bandwidth sharing, and lack of an accurate performance qualifier for choice of traffic parameters. The collected traces help to understand these effects and eliminate them.

Measurement Methodology

This paper studies the packet format parameters and multiplexing issues of audio and video traffic at the transport layer before reaching the network. Traffic measurements capture packet overhead imposed by the Transport, Network, and Datalink layers' protocols. It also provides information on the local inter-arrival time, latency, and jitter.

We focused our study on audio and video codecs defined in the G.723.1 specification [3] and the H.263 specification [5], respectively. Although the mandatory codecs for the H.323 standard are G.711 [2] and H.261 [4], most of the H.323 terminals that connect to the Internet via modems through Internet service providers use G.723.1 and H.263 codecs as their preferred conferencing codecs. G.723.1 has lower bit rate (5.3/6.4 kb/s) than G.711 (64 kb/s). Hence, it is more suitable for transmission over low bandwidth links (e.g., 28.8 kb/s and 33.6 kb/s modems). However, for video conferencing, H.263 has better quality than H.261.

It is important to capture the worst-case behavior of the network such that the solutions proposed can have wide applicability. It is also important to observe the most congested time of the day over the Internet. We used a variety of comparable Internet service providers for establishing conferencing sessions.

Media Trace Format

Audio Format

G.723.1 **[3]** is a dual-rate speech-coding standard, which operates in low bit rate while maintaining high perceptual quality. It is recommended as the preferred speech codec for the ITU H.323 **[6]** conferencing standard over the Internet when the access link to the Internet has limited bandwidth (e.g., an ISP connection using a modem). The G.723.1 codec has two bit rates associated with it. These rates are 5.3 and 6.4 kb/s, the latter being of better quality. Both rates are a mandatory part of the encoder and decoder.

The collected measurements are applied to a traffic scheduler. The scheduler accepts audio and video packets and schedules them for transmission using the scheduling algorithm of interest. During the simulation run, measurements of packet sizes, inter-arrival time, latency, and jitter are computed. The scheduler is depicted in Figure 1.

The scheduler is a multithreaded application. Each traffic source has its own thread. The scheduler also has its own separate thread. Additional traffic sources such as T.120 data or other video or audio codecs can be easily hooked to the scheduler.

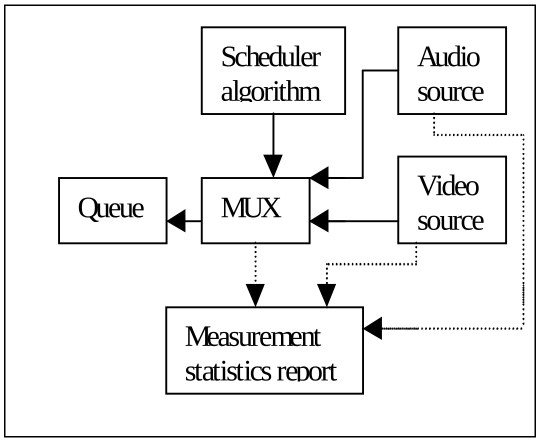

Figure 1: Architecture of the traffic scheduler

The G.723.1 encoder provides one frame of audio every 30 milliseconds. The audio frame size is 20 bytes of pulse-coded modulation (PCM) samples for the low rate (5.3 kb/s) and 24 bytes for the high rate (6.4 kb/s). Each application is set for a particular number of frames per audio packet before the conference starts. These values are negotiated when the call is established, and the least number prevails. Both applications are forced to use this number as the maximum number of frames incorporated in an audio packet.

Increasing the number of frames per audio packet improves the bandwidth utilization and decreases network packet overhead. However, it also introduces additional delay to the audio playback since a packet has to wait for all the audio frames to be accumulated before sending it across. This host-induced delay can be even more harmful especially with the timeliness requirement of real-time audio telephony.

Each audio packet has a Real-time Protocol (RTP) [7] header (12 bytes) that carries time-stamps, sequence numbers, a payload type, and a synchronization source identifier. In addition, a User Datagram Protocol (UDP) header is needed to carry UDP information about the packet (8 bytes). Then an Internet Protocol (IP) header of 20 bytes is also needed to carry routing information. Finally, since we are strictly constrained in bandwidth as we are expected to connect to the Internet via Internet service providers, an additional 5 bytes of point-to-point protocol (PPP) header is also required.

Video Format

The H.263 codec [5] is designed for a wide range of bit rates (10kb/s - 2 Mb/s). The codec supports five different resolutions. In addition to Common Intermediate Format

(CIF) and Quarter Common Intermediate Format (QCIF) that were supported by H.261, there is sub QCIF (SQCIF), four times CIF (4CIF), and sixteen times CIF (16CIF). SQCIF is approximately half the resolution of QCIF. 4CIF and 16CIF are four and sixteen times the resolution of CIF, respectively. Support for 4CIF and 16CIF allows the H.263 codec to compete with other higher bit-rate video coding standards such as the MPEG standards. H.263 is a hierarchical codec, i.e., some performance and error recovery options can be sacrificed for lower bit rate.

It is important to understand how the video source generates video data in order to analyze video-generated traffic. Video pictures are fragmented into multiple video fragments. Each fragment forms a video packet. Each packet is sent across the network after RTP, UDP, IP, and PPP headers have been added. Video is different from audio in the sense that the generated bit rate can be variable while the audio bit rate is usually constant. The ITU-T standard left this issue up to the application as to whether to use a fixed or variable bit rate for video [6]. Another important difference between video and audio media types is fragmentation. Video can be fragmented because a video-generated frame can be quite large; audio is not fragmented. Fragmentation means a video frame is divided into multiple fragments. Each fragment forms a video packet that is sent across the network. There is an upper limit on the maximum fragment size used by the fragmentation process.

A third major difference between audio and video is the burstiness of the video source due to fragmentation, which causes multiple fragments to cluster in a small interval of time. This burst of packets can lead to packet loss, latency, or at least some amount of unfavorable jitter.

In this section, we focus on the characterization of generated video traffic after fragmentation and the implied tradeoffs when enforcing different maximum fragment sizes for video packets.

Smaller fragment size results in shorter inter-arrival time yet bandwidth efficiency decreases because of larger packet overhead. However, larger fragment size results in longer inter-arrival time yet maintains good utilization of bandwidth. As the maximum fragment size limit increases, larger video packets will occur more often. These packets will require more time to be transmitted and hence the video latency increases.

An experiment was conducted to study the video packet sizes (number of bytes per packet) for different maximum fragment limits as generated by our video source. We set the maximum fragment size to four different values (128, 256, 512, and 750 bytes). We ran four experiments, each with a different maximum fragment limit and collected the corresponding video packet sizes generated after the fragmentation module.

Figure 2 shows how video packet sizes are distributed for different maximum fragment sizes. The figure shows that by using a maximum fragment size of 750 bytes, 70% of the video packets had a size in the range of 50-250 bytes. Less than 10% of the total generated video packets were more than 512 bytes. For a fragment size of 512 bytes, 75% of the packets had sizes between 50-250 bytes. By decreasing the fragment size further to 256 bytes, more that 60% of the packets were in the range of 150-250 bytes. For a 128 maximum fragment size, the video packet sizes were equally distributed in the range of 50-100 bytes, and 100-150 bytes. Generally, the majority of the packets for all maximum fragment sizes were less than 256 bytes. This can mean that the maximum fragment size choice should be in the range of 256-512 bytes.

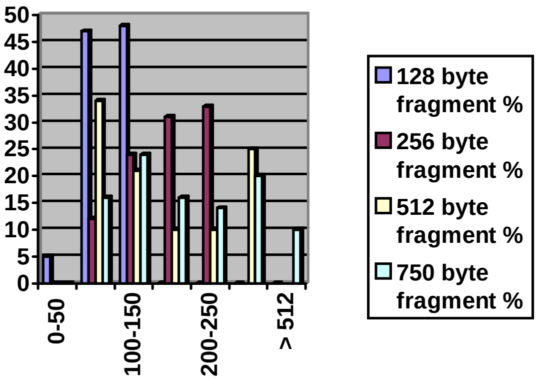

Figure 2: Byte distribution for video fragment sizes

Determining Traffic Parameters

Audio Traffic

It has been demonstrated that the choice of the number of frames per packet affects both local latency and protocol overhead (i.e., bandwidth utilization). We are primarily concerned with the latency induced by the audio packet preparation manager that buffers the captured audio frames and synthesizes them into audio packets. This latency is computed before the packet is actually sent on the network, and it only accounts for local capture and packet preparation delay. The choice of the number of frames per audio packet should minimize both local latency and packet overhead.

Results for Uncompressed Audio Packet Header

Figure 3 shows the percentage packet overhead for both the high and low bit rate G.723.1 audio codec. The packet overhead decreases as the number of frames per packet increases. Intuitively, high bit-rate audio has slightly less overhead when compared to low bit-rate packets.

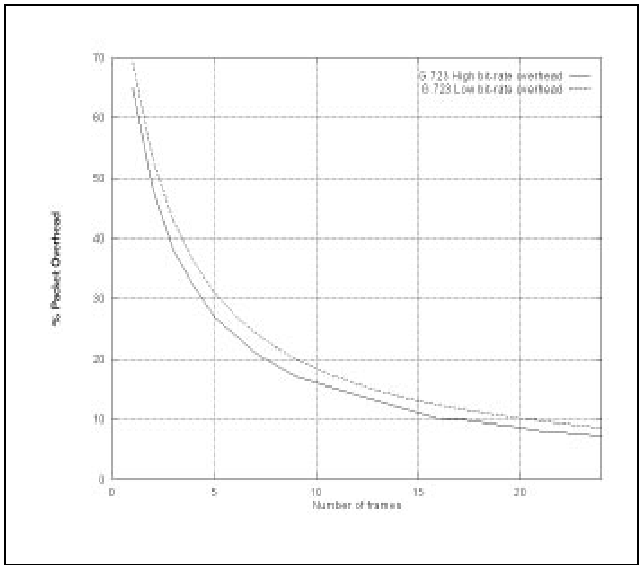

Figure 3: **Packet overhead for low and high G.723.1 audio**

In Figure 4, the number of audio frames per packet is plotted for both rates against normalized packet overhead and latency. The latency is normalized against the maximum latency value exhibited by the largest audio packet (24 audio frames per packet) used in our experiment. The packet overhead is normalized against the maximum overhead exhibited by an audio packet. An audio packet with only one audio frame has the maximum packet overhead as given in Figure 3. As the number of audio frames per packet increases, the packet overhead decreases. The overhead decrease is explained by more information bytes (audio frames) being included in a packet with a fixed header (RTP+UDP+PPP+IP).

The latency increases as the number of frames per packet increases. This is expected since more audio frames require more time to be captured and buffered. Hence, a tradeoff exists between packet overhead and local latency for audio packet transfer. The intersection of the latency and the overhead curves provides the choice of the number of frames that has the minimum latency and least overhead.

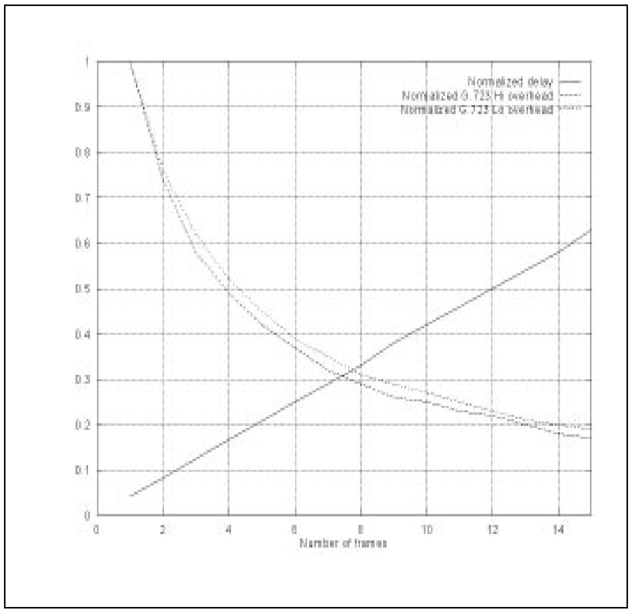

Figure 4: Packet overhead versus latency

The optimal point for the number of frames based on this plot is between seven and eight for both G.723.1 audio bit rates. This value should be used by terminals that target packet efficiency as well as low latency for audio packets.

Sometimes, the audio local latency is reduced at the expense of packet overhead, in order to achieve latencies acceptable to users. In fact, a few applications are willing to sacrifice some protocol overhead for achieving better audio latency. Some H.323 terminals use a value of 3 or 4 for the number of frames per audio packet in order to reduce this latency.

Results for Compressed Audio Packet Header

Protocol header compression has been an active research area for the past couple of years especially after the maturity of the protocols and standards that drive audio and video streaming over the Internet. Besides IP and UDP, researchers have also investigated RTP header compression. Jacobson and Casner [1] proposed an approach for compressing RTP, UDP, and IP headers to be used over low-speed serial connections to the Internet. Examples of these links include low speed (e.g., 28.8 kb/s) modem connections to the Internet through Internet service providers.

This approach seems ideally suited to better bandwidth utilization of the media streams since the protocol overhead is significantly reduced. A few companies are currently working on the deployment of this approach inside the network interface infrastructure in order to improve multimedia conferencing over the Internet for terminals connected via low speed links.

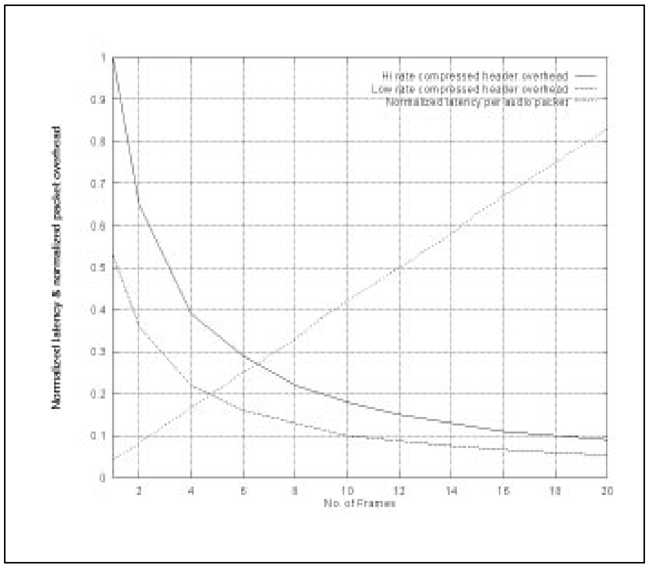

Figure 5: Packet overhead versus latency with compressed IP/UDP

Figure 5 shows a plot of the packet overhead versus latency after applying IP/UDP header compression. The curve suggests that the optimal number of audio frames is almost five for low bit-rate audio and six for high bit-rate audio.

Video Traffic

Another major decision in video source parameters is the choice of an appropriate maximum fragment size. The maximum fragment size contributes to the latency, network overhead, and the amount of generated traffic. As the size increases, larger video packets are generated. Large packets lead to increased latency and less protocol overhead. Reducing the maximum fragment size will lead directly to more packets being generated for the same number of video frames.

Video Packet Length Effect

Histograms of the video traffic are shown in figures 6-8. The choice of a maximum fragment size should capture most of the data clustering and at the same time should not penalize audio packets by generating large video packets that induce huge delays.

We chose 10 kb/s for the video bit rate, which is suitable for ISP-based Internet video conferencing terminals. We used the Quarter Common Intermediate Format (QCIF) for video streaming. Each trace lasted 15 minutes. The trace shows the video fragments generated from the H.263 codec after the RTP header is added. Measurements are taken before the addition of any UDP or IP packet header.

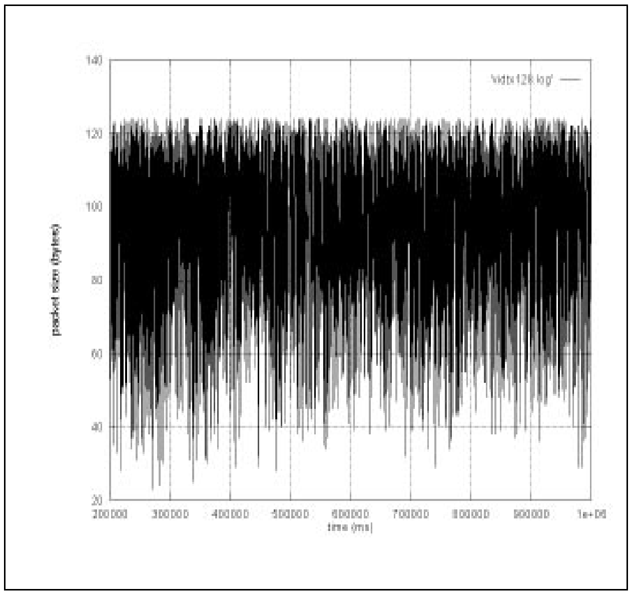

Figure 6: Video histogram for a maximum fragment size of 128 bytes

Figure 6 shows a plot of the video packet sizes against time with the maximum fragment size set to 128 bytes. The packets varied in size from as small as 25 bytes to the maximum value allowed, which is 128 bytes. The plot seemed too crowded as more packets are generated to make up for the small value of the maximum fragment size.

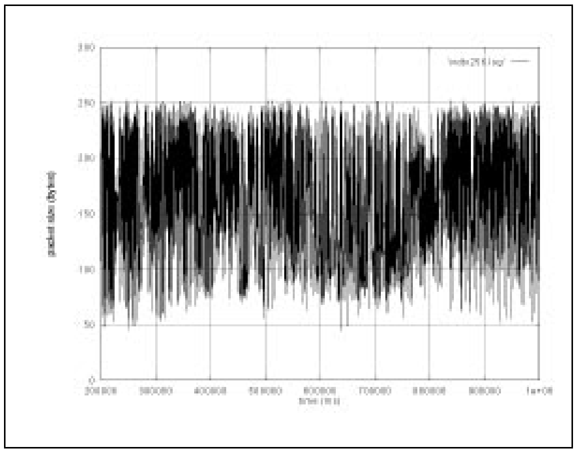

Figure 7: Video histogram for a maximum fragment size of 256 bytes

Figure 7 shows a plot of the video packet size against time where the maximum fragment size is set to 256 bytes. Packets varied in size from as high as 256 bytes to as low as 50 bytes. Figure 7 appears to be less crowded than Figure 6.

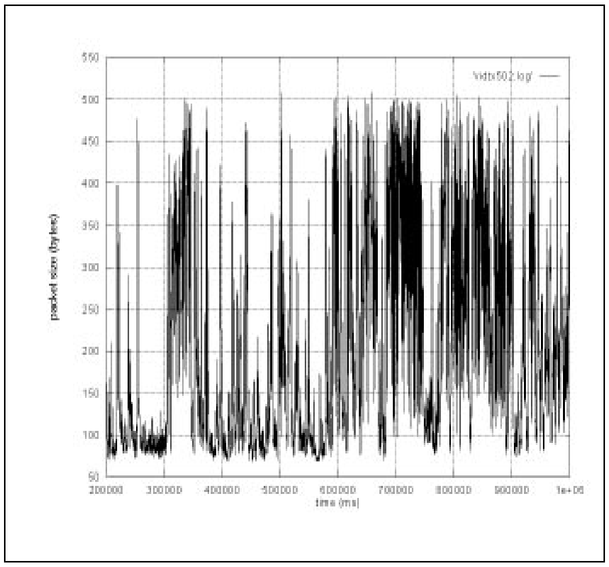

Figure 8: Video histogram for a maximum fragment size of 512 bytes

Figure 8 shows a plot of the video packet size against time with the maximum fragment size set to 512 bytes. The packet sizes ranged from 75 bytes to 512 bytes. The larger swing in the packet size range allowed for a less dense histogram compared to those in Figure 6 and Figure 7.

Table 1 shows the maximum fragment size and the corresponding mean, standard deviation, and number of generated fragments of the video traffic source.

Max Frag	Mean size	Standard Dev	Mean arrival	Frag no.
500	191.64	127.2	275.027	3680
256	169.38	51.358	224.40	4464
128	94.581	21.46	132.568	6382

Table 1: Mean and standard deviation for different fragment sizes

In Table 1, the smaller the maximum fragment size, the smaller the mean size of generated fragments and consequently the smaller the delay (smaller inter-arrival time). However, by examining the number of fragments generated for each maximum fragment size in Table 1, we note that as the maximum fragment size decreases, the number of generated fragments increases. This is expected, however, since the number of video frames to be fragmented and formed into packets is almost fixed for all maximum fragment sizes with which we experimented. In addition, as the maximum fragment size decreases, the packet overhead increases, and the bandwidth utilization decreases. Normally, it doesn't take any longer to send and receive fewer but larger video packets than it does to send and receive more but smaller video packets, if the same amount of video data is played. In fact, the extra packet handling overhead may cause the reverse. However, larger video packets can cause audio latency as the bandwidth is shared, and this is the major concern.

Video Packet Inter-Arrival Time and Jitter Effect

Another important measure we can deduce from the data provided by these graphs is the mean inter-arrival time. This measure provides an estimate for video packet latency induced locally at the host. We compute the rate of change of the inter-arrival time, which is a measure of video packet jitter. As the maximum fragment size decreases, it is expected that the number of packets generated within the same interval of time will increase. The measurements for inter-arrival time and inter-arrival jitter for maximum fragment sizes of 128, 256, and 512 are shown in figures 9-11.

Figure 9 shows a plot of the inter-arrival time and jitter of video packets against time where the maximum fragment size is set to 128 bytes. Note that many packets are generated with variant inter-arrival time and hence the jitter between packets is significant. As the maximum fragment size is increased to 256 bytes (as in Figure 10), the jitter is less significant as a smaller number of packets with less delay variations is generated. Increasing the maximum fragment size further to 512 bytes decreases the jitter significantly as shown in Figure 11.

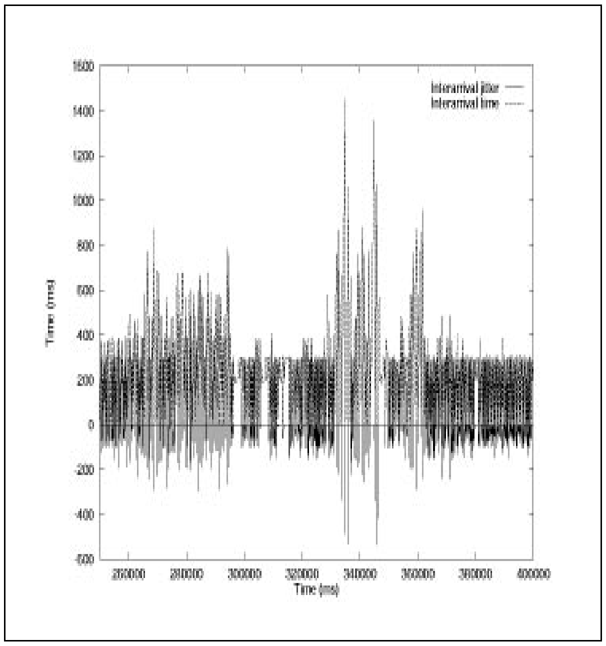

Figure 9: Inter-arrival time and jitter for video maximum fragment size of 128 bytes

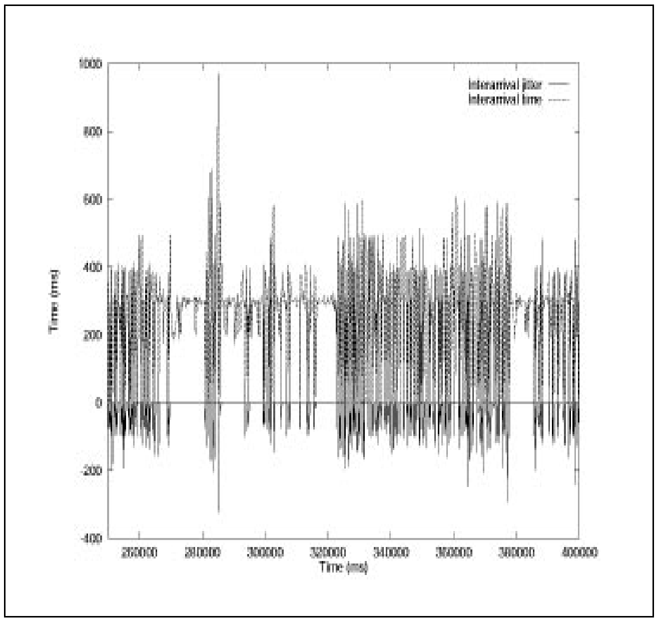

Figure 10: Inter-arrival time and jitter for video maximum fragment size of 256 bytes

In general, local jitter increases as the maximum fragment size decreases. This phenomenon is explained by the fact that more packets are generated for the smaller maximum fragment size during the same time interval. More packets with a variety of inter-arrival times result in a greater likelihood of inter-arrival time variation and hence, jitter.

Theoretically, as the maximum fragment size decreases, the sizes of the packets are normalized and jitter should be reduced. However, the increase in jitter in this case is also due to packet handling overhead in the transport stack from the increased number of packets.

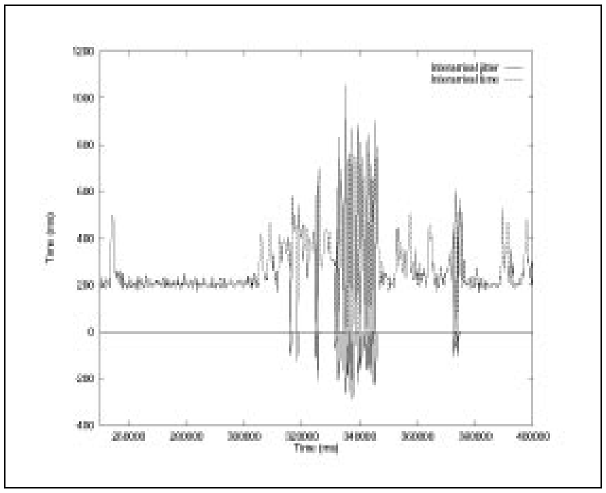

Figure 11: Inter-arrival time and jitter for video maximum fragment size of 500 bytes

Multiplexing Video and Audio Packets

Audio and video packets are generated by their respective sources and then funneled through a shared communication medium. Each media component should subscribe to a sufficient amount of bandwidth to meet quality standards. Lack of appropriate multiplexing algorithms can lead to one or more media components oversubscribing to the shared bandwidth and penalizing other participants.

A typical example of this multiplexing problem is when the scheduler dispatches all media types through a First Come First Served (FCFS) queue. All packets are serviced in the order they arrive without any attention being paid to their priority or timeliness. In this example, if the video data is bursty and a train of video packets is generated during an audio silent (inter-arrival) period, audio packets will be blocked for the duration of the video packet train. This duration may be sufficiently long to disturb the continuity of the audio speech and make it incomprehensible. To illustrate this example, we fed audio and video traffic to the scheduler shown in Figure 1. We chose a worst- case audio source that continuously generates audio packets of four frames each every 120 milliseconds. Video is of variable bit rate with maximum fragment size set to 256 and 512, respectively. We are interested in the number of video bytes that reside between audio packets during periodic silence intervals. If too many video bytes accumulate between two consecutive audio packets, severe delay (and jitter) may be experienced for the blocked audio packet.

We computed the mean number of video bytes interleaving audio packets from the plot. We depicted the host's extra inter-arrival time penalty to audio packets as time incurred by interleaving video bytes. Further, by knowing the maximum number of video interleaved bytes, we can show the maximum penalty incurred on the inter-arrival time of audio packets that will show at the receiver side due to the local multiplexing effect. We can also deduce the mean local audio jitter by computing the rate of change of the interlaced video packets over the experiment interval.

Figure 12 plots the number of video bytes residing between consecutive audio packets against time using an H.263 video source of maximum fragment size 256. Although it is commonly believed that using a smaller maximum video fragment size will lead to better audio performance, our experience proved otherwise. It is observed that the mean interleaved video bytes between audio packets for a video maximum fragment size of 256 is larger than the mean of the corresponding trace for a maximum fragment size of 512 as shown in Figure 13. Hence, reducing the video fragment size alone will not help the audio performance. On the contrary, it may lead to irregular spacing in time between audio packets, which renders audio playback unintelligible.

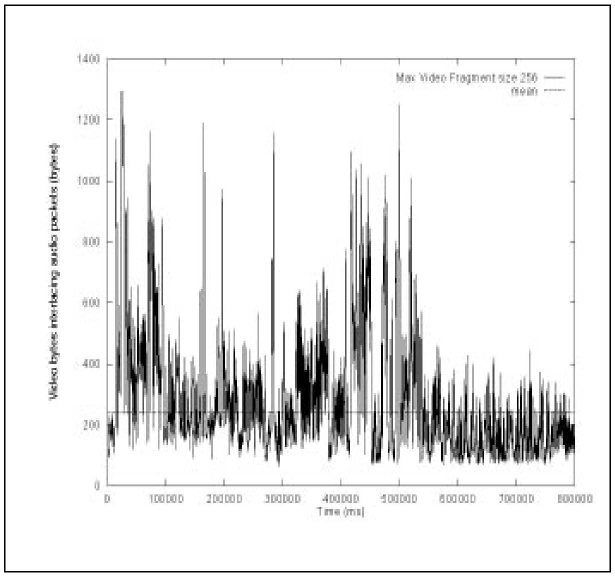

Figure 12: Interleaved video bytes between audio packets (maximum video fragment size 256)

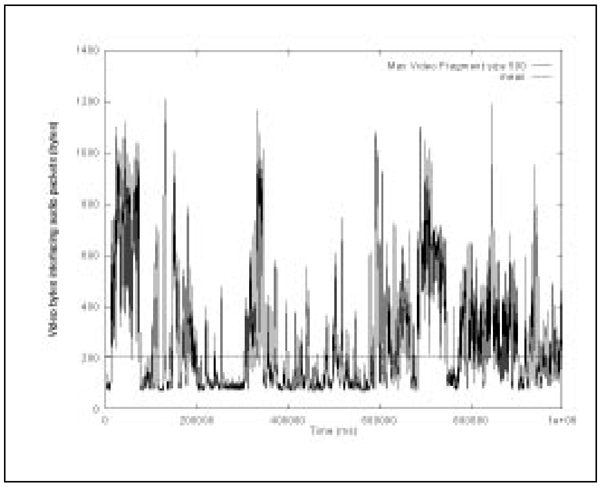

Figure 13: Interleaved video bytes between audio packets (maximum video fragment size 500)

The Problem with Media Multiplexing on the Host

Multiplexing audio and video packets over a shared bandwidth on the local host has its drawbacks. One major drawback we observed was where video oversubscribes its bandwidth. This leads to significant variable gaps between audio packets that causes audio inter-arrival delay and jitter.

There are two reasons for this problem. The first reason is directly related to the variable size of the video packet or fragment. This can be controlled by fixing the size of the video packets, which can be inefficient. The second reason for the problem is related to the burstiness of the video source. Burstiness can be caused by the encoder, the packet fragmenter, or both. The remedy for the burstiness problem is a traffic regulator that controls the flow of video packets and maintains quality of service for all media types on the network. Other solutions include a recovery mechanism at the remote end for audio jitter effects, careful traffic monitoring and remote control of the violating traffic source, and use of priority schemes for audio at the network level and below. These solutions are beyond the scope of this paper.

Conclusion

Real video traffic workload is collected and analyzed in this paper. Worst-case G.723.1 audio traffic is used as the audio source input. The measurements collected during traffic analysis are packet length, inter-arrival time, jitter, and packet overhead.

Analysis of audio traffic revealed a tradeoff between the audio latency and the network bandwidth utilization. Increasing the number of frames decreases the packet overhead and increases the network utilization. However, increasing the number of frames per packet also increases audio local-induced latency by the time taken to buffer the frames before assembling them into packets. The optimal number of frames for minimum latency and minimum overhead is approximately seven per packet for an uncompressed header. This number will add latency overhead to the audio packet of 210 ms. Analysis of audio packets with compressed UDP and IP headers revealed the optimal number of frames to be five and six for low and high bit-rate audio, respectively. These numbers induce a local delay on the audio packet of 150 ms for low and 180 ms for high bit rates.

Analysis of the video traffic revealed that the mean packet size is less than 250 bytes for all fragment size limits. The standard deviation of packet size for all fragment sizes of choice is less than 180 bytes. The majority of the video packet sizes lay in the range of 50-250 bytes. This information suggests that a maximum fragment size of between 256 and 512 bytes is an appropriate choice.

It was also shown that as the maximum fragment size increases, the mean inter-arrival time between video packets increases, the number of generated fragments decreases, and the inter-arrival jitter decreases. In theory jitter should be reduced as the fragment size decreases. However, the increase in jitter in this case is also due to the packet-handling overhead (from the increased number of packets) in the transport stack by the packet assembler.

The local audio latency and jitter experienced by the audio-video scheduler are due to variations in video fragment size and burstiness of the video source. A new performance measurement was introduced. This measurement is the number of interleaved video bytes scheduled between audio packets. This number is a good indicator for audio latency and jitter at the host.

These results help assess H.323 terminal traffic locally at the host during a video conference. They provide useful information for tuning media traffic parameters. The results also provide a better understanding of the audio/video multiplexing problem over a shared bandwidth medium, and they enable the development of effective solutions.

Acknowledgments

The author would like to thank all his colleagues at Intel Architecture Labs. Special thanks and acknowledgment are extended to Steve Ing and Jose Puthenkulam of IAL for productive discussions during the various stages of this work.

References

[1] Casner S., Jacobson V., “Compressing IP/UDP/RTP Headers for Low-Speed Serial Links.” Internet Draft, draft-ietf-avt-crtp-03.txt, July 1997.

[2] ITU-T, Recommendation G.711 (1988)–Pulse Code Modulation (PCM) of Voice Frequencies.

[3] ITU-T Recommendation G.723.1 (1996)–Dual Rate Speech Coders for Multimedia Communication Transmitting at 5.3 & 6.3 kb/s.

[4] ITU-T, Recommendation H.261 (1993)–Video Codec for Audiovisual Services at p X 64 kb/s.

[5] ITU-T, Recommendation H.263 (1996 –Video Coding for Low Bit-rate Communication.

[6] ITU-T Recommendation H.323 (1996 –Terminal for Low Bit-rate Multimedia Communication over Non-Guaranteed Bandwidth Networks.

[7] Schulzrinne H., Casner S., “RTP: A Transport Protocol for Real-Time Applications,” draft-ietf-avt-rtp-04.txt, October 1993.

Author’s Biography

Hani ElGebaly is the project technical lead for the H.32x protocols development within the conferencing products division at Intel Architecture Labs. He received an M.Sc. in Computer Science from the University of Saskatchewan, Canada, and a B.Sc. in Electrical Engineering from Cairo University, Egypt. Mr. ElGebaly is currently pursuing a Ph.D degree with the University of Victoria, Canada. His primary areas of interest include multimedia conferencing protocols, embedded programming, fault tolerant systems, and computer architecture. His e-mail address is hani.el-gebaly@intel.com