An interview with TOM AIRD

Conducted by Thomas Haigh
On
3 and 4 June, 2004
Reno, Nevada

Interview conducted by the Society for Industrial and Applied Mathematics, as part of grant (DE-FG02-01ER25547DE-FG02-01ER25547) awarded by the
US Department of Energy.
.

ABSTRACT:

Aird describes his career in mathematical software, focusing particularly on his involvement from 1972 to 1993 as a senior member of mathematical software library vendor IMSL. Aird entered Olivet College in 1958, and was first exposed to programming as an intern with Burroughs in Detroit. He went on to graduate school in mathematics at the University of Michigan, earning a masters degree. In 1965, he moved to Houston where he provided support to programmers working at the Manned Space Center, under a contract held first by Wolfe Research and Development and then by Lockheed Electronics. After two years, he returned to an academic environment, working as manager of User Services in Purdue University's computer center while pursuing a Ph.D. with John Rice in its computer science department. Aird also mentions the contributions of Carl deBoor and John Lynch. These early experiences gave Aird first hand exposure to the needs and behaviors of scientific computer users, and to early mathematical software libraries. In 1972, he was recruited by Ed Battiste to join IMSL, a small and young company dedicated to the sale of a packaged library of mathematical routines. Aird discusses the origin of IMSL and the roles of co-founder Charles Johnson and early employees Olin Johnson and Walt Gregory. Aird served as Director of Mathematics and was responsible for the development of numerical routines. As the firm grew he came to supervise a large team, and was responsible for the development of software tools to manage the source code base for portability to exploit the interactive capabilities of minicomputers. Aird contrasts the capabilities and design of the IMSL library with competitors including IBM's SSP and NAG and discusses its relationship with public domain projects such as EISPACK and LINPACK. He also explains IMSL's sales strategies and discusses its relationship with customers. During the 1980s, the IMSL library was rewritten in FORTRAN 77, which Aird considers a very important development, and adapted for various microcomputer and workstation platforms. The firm experimented with other projects, including John Rice's PROTRAN, a high level preprocessor, a graphics library, and a C subset of the main library. Aird left IMSL in 1993 following a change in the senior management team after which the company deemphasized the mathematical library business, merged with Precision Visuals and aimed (without success) at an initial public offering. IMSL eventually became Visual Numerics. After leaving IMSL, Aird worked as a partner in Windward Technologies, a very small firm selling optimization software. He was active for many years as part of the IFIP 2.5 Working Group on mathematical software.

THOMAS HAIGH: Thomas J. Aird interviewed by Thomas Haigh, Tape 1, Side 1. This is an oral history interview conducted as part of the SIAM project to investigate the early history of software for scientific computing and numerical analysis. This interview is being conducted in Reno, Nevada, at Dr. Aird's house, on the third of June 2004. This is the first of two projected sessions.

Dr. Aird, good afternoon. Thank you for taking part in the interview. I wonder if you could begin by talking in general terms about your early life background and education, particularly inasmuch as it led you towards mathematics and other areas of applied science.

TOM AIRD: Sure. Well I grew up in Detroit, and I went to the Detroit public schools, Carleton Elementary and Denby High. Of course there were really no computers, this was in the early fifties to 1958, but we did study mathematics. I was not a particularly good student in those days mainly because I had interests outside of school and did about as little as I could to get through high school. But then in 1958, I went to Olivet College in Olivet, Michigan, and I was a math major. I studied general mathematical background, linear algebra, differential equations, and four years of math, and did an honors program in number theory, which was not much related to computing. In those days we heard about computing, but we didn't do any computing.

After graduating from Olivet College I had a job at Burroughs Corporation in Detroit, Michigan. I would say that was my first exposure to computing. Part of the job as a summer intern at Burroughs was working on a machine called a card sorter. Everything was punch cards in those days, and this was IBM equipment. There was an IBM collating machine and the company kept a lot of records on punch cards. The accounting people, and other people that had these records wanted them sorted. By today's standards, this was very low level stuff, but at the time that was kind of the state of the art.

Across the hall from this card sorting and collating operation that I mainly worked at, was a Burroughs 205 computer (I believe that was actually bought from Datatron). That computer had a drum memory with something like 7000 words of storage, so by today's standards this was a pretty small amount, and it was organized with what they called "high speed loops." The loop of 4000, 5000, 6000 addresses, everything was repeated twenty times, so as the drum was spinning around it would be twenty times faster access than the low level loops. The programming language was actually ALGOL. In those days, Burroughs was very big into ALGOL. This was my first exposure to computing, and I found it very interesting.

At the end of the summer of 1962 I went on to the University of Michigan, to do a graduate degree in Mathematics. I took a course from Bernie Galler on computing. Bernie was the designer of a language named MAD, and it was quite a good language, it was certainly better than the prevailing language which was FORTRAN. Bernie made it a pretty exciting course and that really got me interested in computing. The University had an IBM 7090 or 7094 computer. Everything was still on punch cards so you'd write a program in MAD, you'd write it out on a piece of paper and then you'd go and sign up for time at the computing center using a card punch, and you'd punch your program onto punch cards and then you'd submit it to the computing center. Sometimes you would get your output back within a couple of hours, which was a printed listing, and then you'd find your compiler error, and you'd go back to the key punch and correct it, submit it again, and if you were lucky the second or third time you'd get things running. I would say that was the turning point for me, there was no turning back at that point.

HAIGH: Well let's move back then and go through some of those points in more detail.

AIRD: Okay.

HAIGH: So one thing you had mentioned was that you weren't particularly motivated as a high school student. But you also then went to college and majored in mathematics. So what drew you towards mathematics?

AIRD: I don't think there was any one single event that drew me to mathematics it was just kind of a natural inclination. Perhaps my parents said, "maybe you would be good at math," something like that, and it happened.

HAIGH: Was it something that you'd been doing relatively better at than other subjects in high school?

AIRD: Actually no. I think I was okay in math, but it was more or less the amount of time that I was willing to spend on it in high school, I don't think it was the ability it was just social events and friends and things that we did outside of the academic side.

HAIGH: And did you have any science or technology related hobbies?

AIRD: None.

HAIGH: So you weren't the kind of person who liked to tinker with machines or electronics?

AIRD: I had a car and I tinkered with the car, you know, things like that, but nothing else.

HAIGH: You'd also mentioned the work with punch cards. What kinds of jobs were being run with those machines?

AIRD: Well it was mainly personnel records and accounting records and things like that, there wasn't anything very exciting by today's standards. Let's say you had a name and address and a phone number and information like that, kind of a data base if you would by today's standards, on punch cards. They might have a few several thousand cards in the file and they'd say we want this sorted by phone number, or salary, or some field like that. Very, very elementary applications by today's standards.

HAIGH: Entirely data processing rather than scientific computing?

AIRD: Entire data processing.

HAIGH: And were you working as a machine operator?

AIRD: In the summer of 1962, I was a summer intern at Burroughs and so I, along with one or two other people, operated the sorting machine and the collating machine. You could change the way these things did certain things by wiring, there were boards that you would pull out and you could put wires in different configurations and cause them to perform different operations.

HAIGH: The plug boards.

AIRD: The plug boards, yes.

HAIGH: And you were picking up stacks of cards and moving them between machines?

AIRD: Yes.

HAIGH: So did you have any say in designing the procedures or the wiring, or were you just following instructions?

AIRD: Just following instructions, I was just a summer intern at that point. Pretty exciting, but the card sorter, oh, it was so fast, cards would just zip through.

HAIGH: And do you think that experience was something that made you more interested in computing technology?

AIRD: Oh definitely. Especially when I moved across the hall at times to play with the Burroughs 205. As summer interns we didn't have specific duties. The duties were to sort the cards and things, but when things weren't busy there was this computer across the hall and the interns got to play with it. You could write a program, punch it on cards, feed it into the machine and the compiler would compile it and you could stand by the printer, you could watch lines of code come out, cachunk, cachunk, so you'd have about one line compiled every second so if you had a couple hundred lines of code it would take three minutes to compile.

HAIGH: And were you able to do any work with the computer or did you just look at it?

AIRD: No we wrote programs to do simple minded things, you know, simple minded numerical formulas, interpolation, or things like that, nothing substantial.

HAIGH: Okay. Now after you got your undergraduate degree you then decided to go on to graduate school. So by this point had you become much more interested in academic matters?

AIRD: Well I was definitely interested in mathematics at that point. Until this summer internship and the first semester at Michigan, I probably didn't know how much I would be interested in the actual process of computer programming. So that was an evolution from mathematics, studying things that would ultimately lead to numerical analysis. I would say the real two turning points, one was the internship at Burroughs that got the initial interest in computers and computing, and probably the most significant event was the course at the University of Michigan taught by Bernie Galler on the MAD language, because that was real computing, solving differential equations and real problems. It was extremely interesting, even though the process in those days was rather slow and tedious and frustrating at times with the punch cards and the job submittal process. I mean I can't imagine by today's standards what one would think if every time you wrote a program you had to submit it and go away for two hours and come back to get your output.

HAIGH: And by that point were many of the masters students in mathematics using computers as part of their research or course work?

AIRD: I would say yes. I didn't have a lot of direct contact other than the course that I took at Michigan, but you could see that there were many many people just using this computer all the time there, the scientists and engineers were starting to write programs of their own in various fields. I did not get involved in those things but I could see that they were going on at that time.

HAIGH: And what kinds of topics were covered in Bernie Galler's course? You'd mentioned the MAD language itself, but were there other things?

AIRD: Well there were various projects. I remember one that involved trajectories in solving differential equations. We were given the algorithms for solving the differential equations and the idea was that you would fire this missile, or whatever, and then you would get a solution and if you overshot your target then you would adjust certain parameters to play around and finally your program would actually set all the parameters correct so that the trajectory would lead from your origin point to the target and then your program was finished when you did that.

HAIGH: And were there any other courses or topics that you were exposed to in your graduate and masters degrees that you think had a particular effect?

AIRD: Well at the University of Michigan there were other mathematics courses, there was a complex variables and there was linear programming, I think the professor was Robert Thrall, that taught a course in linear programming which was very relevant too.

HAIGH: Now when you graduated with your masters degree how did you decide what you wanted to do next?

AIRD: Well at that point I felt I knew a lot about computing, though I probably didn't know very much. I decided that I had gone to school long enough and I had a job offer to go back to Burroughs.

HAIGH: And that was a follow up from your earlier internship?

AIRD: From the internship, yes. The same group within Burroughs. From 1963, that was the point I graduated from the University of Michigan, until about December 1965, I was an associate mathematician, with a group that was developing, I believe, what was known as the E101, although it's pretty obscure in the history of computers, but it was in fact a little accounting type computer. Burroughs had developed a lot of banking equipment machines. I was with a group of people: there were two or three Ph.D.s in the group and there were maybe five, programmers. We had various assignments. One of the assignments was the automatic circuit layout. In those days they were just starting to use circuit boards and there were more and more components, and they wanted a computer program that would decide how to run a wire from Component A to Component B through this maze of other components, and I did that with Dr. Milnes, Harold Willis, and that paper was published in the Journal of the Industrial Mathematics Society. [Miles, H. W. and Aird, T. J., -A program for Automating Circuit Layout, Journal of Industrial Mathematics Society, Vol 14, Part 2, 1964]. Probably pretty obscure again in circuit layout, and by today's standards fairly irrelevant, but in those days that was the state of the art.

HAIGH: That would seem to be an example where another group within Burroughs had a need and you were helping them with the problem on a consulting kind of basis?

AIRD: Yes. It was an engineering group that was developing the E101 machine and we were a mathematical and programming support service to that group.

HAIGH: So the programs that you wrote were used internally?

AIRD: Yes, only internally.

HAIGH: Now do you know if there was a separate group within Burroughs that would have been writing mathematical routines for use by customers?

AIRD: I don't think so, no. Maybe a relevant fact was that Burroughs had developed the B5000 computer, and that computer was designed around the ALGOL language so it was very, very efficient on compiling and executing ALGOL, it was a multi processing, multi tasking system, I thought far in advance of anything else on the market. So it was a very exciting time in my career to program in ALGOL and use this machine.

HAIGH: So was your group, internally, by that point using a Burroughs 5000?

AIRD: Yes.

HAIGH: Can you remember any other kinds of problems that people would work on?

AIRD: Well there were various mathematical tasks that we had to do but we were never part of the overall design so those were just kind of a typical thing a mathematical software support group might do, and nothing that I would say where we had input into the design of the machine or the specific mission. It was just a support activity.

HAIGH: And how about your personal feeling towards programming. Was it something that you found yourself getting more and more into as you did more of it?

AIRD: Definitely. I would say that's where my real interest lies, the programming. I studied a lot of mathematics and then I studied numerical analysis, but, I would say my real interests have always been in programming, the art of programming, writing good programs, making them error free and making subroutines that other users could use, that were easy to use.

HAIGH: So you've already mentioned the fact that you would write the program on punch cards and wouldn't receive the output for some time later. Is there anything else that stands out in your mind just about the actual activity of programming in those days, what it was like?

AIRD: Well it seemed to be moving along fairly rapidly as far as the development of the machines from the time of my first job at Burroughs in the summer of 1962, where this 205, Burroughs 205, was just compiling maybe one line a second, maybe sixty lines a minute of ALGOL. The Burroughs 5000 was certainly many, many times faster, though it was still punch card oriented.

HAIGH: So did that make a big difference to the speed at which you got your jobs back?

AIRD: Well sure by today's standards this would be almost unworkable, but if you look back we were not being as productive as you can be today. I mean today you can sit down and you can hack out a code and you can pull together pieces and you have editors and you can find things, in those days you had to have a listing so if you had more than a few hundred lines of code it became unworkable. You had to look through that, find your error or write it out, and then go get the punch cards, find the card, repunch it, reinsert it.

HAIGH: Did you punch your own cards?

AIRD: Definitely, I don't think I ever had a key punch service.

HAIGH: And were you allowed to use the computer directly, or was it a separate bunch of operators who did that?

AIRD: Well, again, depends on where we're talking about. At the University of Michigan the computing center ran the computer we didn't have anything but a window, we gave them a deck of cards and they took it from there. Now at Burroughs we were hands on, when we came into the room our group took over the machine and we were the operators.

HAIGH: So different groups would be signing up for time on the machine?

AIRD: Yes.

HAIGH: And did you finish up getting a lot of time in the middle of the night?

AIRD: We actually did. I can remember many days being told okay you have computer time at midnight, so we would go home and rest up and come back at midnight and maybe work for four or five hours until our time slot was up. But I was young in those days and it didn't seem to bother me much.

HAIGH: Yes. And did the group create subroutine libraries for its own use?

AIRD: Not at Burroughs, no, this subroutine business came a little bit later in my career.

HAIGH: So other than the stuff that was built into ALGOL you'd be writing every program from scratch?

AIRD: Well there were sources for algorithms and things. There were a lot of algorithms in those days, so you would look up an algorithm in a journal like *Numerische Mathematik* or places like that, and then you would actually write the code to implement that algorithm.

HAIGH: So in that sense you'd be using things but it wouldn't be in the form of a standard library that you'd just link to. You'd actually be going to a printed source and re-implementing the algorithm every time?

AIRD: Absolutely. I don't remember in those days actually having a library, everything was in your deck of cards and if you wanted to use something you put that piece, that procedure, if it was ALGOL it was a procedure, in your deck of cards.

HAIGH: Right. Just one more question about the Burroughs years then. I wonder could you say something about how large this group was and if you can remember anything about the backgrounds of the other people on it?

AIRD: Well there was two parts to the group. Our group was programmers and there were, I remember maybe four people, two of whom had masters degrees, and maybe two with bachelors degrees, not a lot of programming experience but they were becoming programmers. The other part of the group, Harold Mills was in. He's a Ph.D. mathematician, and there were maybe two other Ph.D.s in that group, and they did not program but they were the mathematical side of the equation.

HAIGH: So the theory was then that the Ph.D.s would be doing the mathematical side and people like you would be doing the implementation?

AIRD: Yes.

HAIGH: Is that how it worked in practice?

AIRD: That's how it worked in practice.

HAIGH: Okay. So unless you have other things to say about the time at Burroughs then perhaps we can move on to your next job.

AIRD: Okay, I think that's enough about Burroughs. There was one person in our group, a programmer who had retired from the Navy. He had a contact, Bob Witte, he knew at Wolfe Research and this was in Houston at what was then was known as the Manned Space Center. He said they were hiring people in Houston and they had pretty exciting jobs down there, they had very good computing equipment and they had, of course, a mission to send a man to the moon, ultimately. I went down there for an interview. I'd never been to Houston before. I arrived on, I believe, a December day in 1965, it was a beautiful sunny day in Houston, and had a wonderful interview and decided this would be the place where I'd like to move. I joined Wolfe Research. Wolfe provided programming services, and the space center had a group that was called Theory and Analysis, which had several Ph.D.s, let's see, I remember Henry Decell and Gene Davis, and there were a few others. These people had backgrounds either in mathematics or in mechanics or something else. There were several computers in that group, Control Data, IBM and others, so we had lots of computing time and various assignments to support this group. Actually, part of that job was to develop subroutines, although it was on a small scale for various mathematical applications.

HAIGH: So Wolfe Research and Development was the company that you worked for and this was a subcontractor to NASA?

AIRD: That's correct.

HAIGH: And their work was part of the Apollo program?

AIRD: Yes.

HAIGH: So did they have a big contract or two, or was it miscellaneous kinds of work?

AIRD: Well Wolfe had one big contract and I remember there were, maybe, I'd have to guess, but three or four hundred on that contract, so it was large and growing.

HAIGH: What was the contract, what were they doing?

AIRD: The contract was to support the Theory and Analysis group, so instead of becoming a Government employee they were subcontracting this to Wolfe, and Wolfe was hiring programmers and moving programmers in from all over the world and housing them and giving them office space.

HAIGH: Alright. So there's one big contract but in practice that contract was to support, so it would break down into lots and lots of smaller jobs?

AIRD: Yes.

HAIGH: Okay.

AIRD: And then as you see on my resumé here, I also worked for Lockheed Electronics. Well what happened was that contract went up for bid and Lockheed won over Wolfe, and so it was really the same job but it was with Lockheed Electronics.

HAIGH: So in practice you were basically doing that job from 1965 through 1967?

AIRD: Yes.

HAIGH: Was it still basically the same kind of thing that you were doing, where was a special purpose mathematical routine that needed to be implemented and you would work with a Ph.D. mathematician and you would do the programming, or was the character of the work different?

AIRD: No, it was really very similar to what was being done at Burroughs. That is, the people in the Theory and Analysis office were the Ph.D.s. Some of them were mathematicians but some of them were either engineers or mechanics and things like that, so they were taking projects from other parts of NASA and doing the design work and then telling our group, "Okay we need something here to solve a system of linear equations or we need something to solve differential equations," and so on and so forth. We would implement those routines, and even though this was a short period of time, we gathered several interesting subroutines and maybe you could say the beginnings of a subroutine library. Again, it was all punch cards, same old deal.

HAIGH: Were you still programming ALGOL or was it in FORTRAN?

AIRD: Actually that's a good question because the Burroughs machines were not very popular, unfortunately, and the machines that were popular were all being programmed in FORTRAN. I remember the Control Data 3600, and it seemed like it was IBM maybe 7094, or something like that, that were the two main computers and it was all FORTRAN.

HAIGH: So was that a hard adjustment to make as a programmer?

AIRD: Actually it was. ALGOL had things like dynamic storage allocation that made writing programs very, very much easier and it was a beautiful language. FORTRAN was very rudimentary and I think I did have some trouble, but finally I really got into it.

HAIGH: Now you'd mentioned the subroutine libraries and said that you were still using punch cards, so at that point would the idea be that you would have a FORTRAN procedure that was represented by a small stack of cards and then when you fed your job into the compiler that you would include the cards for the new program and then you would also pull from the shelf some additional cards with more generalized subroutine on them and feed them in as well?

AIRD: Absolutely. There were card cabinets full of these subroutines and that was the main archive for those, I think that there were some disk files in those days also, but disk storage was kind of at a premium so you didn't keep a lot of these things on disks. You would tend to go to the card cabinet and you might duplicate a deck of cards. If the subroutine was fifty cards you might take that out and make a copy of it and put that in with yours and then continue to run that, instead of just linking to a library, very mechanical.

HAIGH: And was this idea of reusing subroutines something that the management had come up with or was the initiative coming from individual programmers?

AIRD: I can't honestly remember the management. The management seemed to be more business management rather than technical management, so these were ideas that were becoming popular at the time. It seemed obvious that you didn't want to reinvent the wheel, so if you had a nice subroutine for solving systems of linear equations you wanted to preserve that and reuse it.

HAIGH: In practice, then, it would be more that you knew that one of your colleagues had come up with a nice routine and you would go and duplicate some cards and then use them for your own purposes?

AIRD: Yes.

HAIGH: And can you remember any attempts, either with the routines that you wrote or with ones that you borrowed from other people, to make the routines more general so as to increase their reusability?

AIRD: Not so much in those days. It wasn't like the IMSL library where you were serving thousands of scientists and engineers. You were sharing your routines with a few people and, where they might have a few comments here or there, generally they accepted what you did and then used it. So there wasn't a great deal of emphasis put on really elegant easy to use user interfaces for these subroutines. It was mainly, we have a function, we want to solve this particular mathematical problem and we want to do it in a reliable manner. Also, those early routines lack any kind of error handling so if you didn't get it right it would blow up and then it would be your job to fix it and to make sure that you got everything input right. By today's standards documentation was very rudimentary.

HAIGH: And do you remember any examples of the kinds of jobs that you programmed there?

AIRD: Well I remember there was a lot of emphasis on solving differential equations and so there was a lot of work on what is known as stability theory and I did some of this work with one of the NASA people, his name was Milo Keathley and that was published as a NASA technical note in 1967. [Keathley, S. M. and Aird, T. J., -Stability Theory of Multistep Methods, NASA Technical Note TND-3976, May, 1967]. They were solving differential equations over large intervals, and if the numerical methods weren't stable then their solutions would diverge and they didn't have the properties they want, so we spent a lot of time on stability theory.

HAIGH: Was there a sense that the Apollo program was requiring cutting edge work in this kind of area?

AIRD: Actually from my recollection, I don't recall any really direct application, it seemed like it was indirect. The Theory and Analysis office was interacting with the NASA engineers, and then we would interact with the Theory and Analysis office personnel, so you never really had a sense that you were directly helping the mission, but in the overall scheme of things I think we were.

HAIGH: Yes. If this group was three or four hundred people that must have been one of the very largest collections of mathematical programmers around at that point.

AIRD: It probably was, it was a very diverse group and supporting lots of different functions at NASA. I didn't know, other than the work at the Theory and Analysis office, what exactly the other groups were doing or what the other parts of the group were doing.

HAIGH: Did you need any kind of security clearance for that?

AIRD: I did get a security clearance, and to be honest with you, I can't remember what kind. It certainly wasn't one of the top top levels, but there was secret clearance on some level where there had to be an investigation of my background.

HAIGH: Now then in 1967 you left this job, and went to Purdue. What prompted that?

AIRD: Well, again, it was this interaction that I was working with a group of Ph.D.s, and I could see that now there was more and more for me to learn, and I was more and more interested. It seemed the best way to achieve that was to probably go back to school and work on a Ph.D. program myself. In the investigation that I did at various schools, Purdue came up on the radar screen many times. In particular there were people there like John Rice, and Sam Conte was the head of the department at that time. It was a numerical analysis group within the computer science department. Carl deBoor was there, and Bob Lynch, and it was a very outstanding group of professors in a department that seemed very much oriented to the subject matter than I wanted to learn about, namely, numerical analysis and computing. So I applied there, and it turned out that they had a job in the computing center and so I was hired as the Manager of User Services for the Purdue University computing center. Actually, at the time I started Sam Conte was both head of the Computer Sciences Department, and they had a big computing center, so I worked in the job as Manager of User Services and became a full time graduate student at Purdue University in West Lafayette, Indiana.

HAIGH: So what did being manager of User Services involve?

AIRD: The year that I arrived, 1967, Purdue bought a Control Data (6500, I believe was the number of it). The one interesting thing I remember is that they built the mathematical sciences building on the campus and they had designed it based on bringing an IBM machine, and the IBM machine had a smaller dimension than the Control Date machine. For some reason IBM couldn't deliver, this happened before I arrived, and so they changed the order to order the machine from Control Data, and this was a, I don't know, million dollar machine, so it was a very large expense. The Control Data machine was much larger, so and they had to dig a big hole in the basement of the Math Sciences building and jackknife through eighteen inches of reinforced concrete to build an entryway for this machine.

By the time I arrived, which I believe was August 1967, that machine was already in the Math Sciences building. The User Services Group consisted of a consulting function, staffed with graduate students. Everything was punch cards, so that scientists and engineers on campus that used the computer would come over, they would submit their job in a window, come back, and they would have an error, maybe, in the compiling (this again was mainly FORTRAN) and rather than spend time on it they could go across the hall at the computing center and sit down with a graduate student that was on duty and the graduate student would help solve the programming problem. It might just be as simple as a syntax error. Also, I remember in those days, you actually had to specify how much memory you needed for your job but that actual specification had to be an octal (base 8) number. Quite often this was kind of a joke, the user would specify that they would need 90 thousand words and they'd say 9 is not an octal number. The consulting service was run through User Services, so this was actually a point in time when we developed a lot of subroutines and tried to develop very good documentation and have subroutines available on disks that users could then use directly, just like we think of a subroutine library today. So I would say this was the first full scale attempt I was involved in to develop real libraries of real mathematical software.

[End of Tape 1 Side A] [Start of Tape 2 Side B]

So you'd begun to describe the work at the computer center at Purdue. Now it sounds like your job as Manager of User Services was in some ways the natural continuation of the work that you'd been doing at Lockheed and Burroughs. Were there differences between the kind of support that you'd be giving in a university center and the work as a consultant on these mathematical problems that you'd been doing with Lockheed and Burroughs?

AIRD: Yeah there were some differences and some similarities. I would say the difference was that we were providing computing services to the university which meant professors. There was a Professor Michael Rossman, Professor of Biological Science, who's a well known X-ray crystallographer at Purdue University, who used all the computing time that he could get and wrote huge programs. Then there were graduate students in various departments that were doing work for their thesis and in those days there weren't many experts around. In fact, the User Services consulting staff quite often had very little computing experience. If someone knew how to use a key punch, that was good news for someone getting into that group. If you had somebody who had experience in computing that was just outstanding, but nonetheless, these people were put in charge and there were various levels of help and there were other people on the staff of User Services. So we tried to first answer the question for the users at the consultant level but if they couldn't then we would ratchet it up to another staff level, and quite often I would get directly involved in helping users. So I would say providing that consulting service was maybe the most important function of User Services.

HAIGH: So the users would try and write their own routines and then when they didn't work they would come to the consulting services and say why doesn't my program work?

AIRD: Yes. Sometimes it was a simple syntax error and sometimes it was a very complicated algorithmic thing, where the program was crashing somewhere down the line. There really weren't any wonderful debugging tools in those days, so the consultants were the first line of defense, and then the staff of User Services was the second line of defense. In those days there was a much bigger demand for experts than we could fulfill, so you had to force people to go to the consultants first even though they thought they might not get the answer. But I think overall this helped to move things along. Without that service people would have been lost because everybody was writing their own programs. All the students in science and engineering were all writing FORTRAN programs in those days, that was the only way of getting things done. There were a few packages coming along like the BMD statistical analysis programs, and then there was a package called SPSS that came along, and so the computing center tried to acquire those packages and provide support for them. One of the packages that came along a little later on was the IMSL library. Purdue became a subscriber to the IMSL library in maybe 1971 or somewhere around that time. We had built up subroutines of our own but when you acquired the IMSL library this gave a tremendous repository of things that these users could use, and so we had to make documentation available and tell users how to use the IMSL software.

HAIGH: So your first exposure to the IMSL library was actually during your time at Purdue in the computer center.

AIRD: Yes, I was a customer, and of course then when we started using it there were a few problems/bugs, like you would have with any software, and so I was the one that was reporting those problems back to IMSL and working with the people at IMSL.

HAIGH: And did CDC itself supply a library with the computer?

AIRD: I don't remember a CDC library. Certainly there were the functions that came with FORTRAN, the built in functions, the trigonometric functions and absolute value functions and things like that, but I don't remember much of a CDC library. There was the IBM Scientific Subroutine Package that came along and that was also part of what we offered at the center.

HAIGH: So IBM SSP could be compiled to run on the CDC?

AIRD: Yes, we had the source code. What I remember is that that package was not of very good quality, there were just a whole bunch of problems with it and it was not supported, so the bugs were not getting fixed. That was really the reason that IMSL was formed, to take over where the IBM SSP left off. I don't want to get ahead of myself here, but the IBM Scientific Subroutine Package developer group kind of split off and formed IMSL. So while we had that scientific subroutine package it was difficult to support, so it was really kind of a breath of fresh air when the IMSL library came along. It was much higher quality, and easier for the users to run.

HAIGH: It's interesting that IBM would allow its bundled software to be used on competing machines.

AIRD: I'm trying to remember, recall the exact nature, but it seemed to me that the IBM Scientific Subroutine Package was almost a public domain thing. It didn't seem there were any issues. You didn't become a subscriber. You just could get it and you had the source code and therefore you could compile it. There was a slight difference in the compiler between IBM and CDC. You could make some changes, but I don't remember any subscription or anything like that.

HAIGH: So at that point then you didn't think of mathematical software as something that would be paid for. That is, until the IMSL library, you just assumed that it could be copied and distributed?

AIRD: I think that's right, I think it was a little bit of a shock, "Gee we have to pay for this stuff", the cost was about $800 a year. It was supplied on a subscription basis from IMSL, and that was the University price, I believe, at that time. And it seemed like we talked about it and decided okay. I think I made a pitch that this would really be worth it because it looks like it's much better than what we have and it would really help the users. I think it was good, but it was part of an evolutionary thing of how you pay for software.

HAIGH: And had there been any kind of CDC user group exchanging programs?

AIRD: I don't remember any user group at Purdue, CDC or other. It seemed like everything was either developed by the User Services group or acquired by the User Services group.

HAIGH: And can you say something about the routines that the User Services group developed?

AIRD: Well there was software developed by various graduate students, some of whom worked for the User Services group, some of whom were graduate students studying various areas. I know there were routines for solving linear equations, that was kind of a popular problem at that time, and ideas like using iterative refinement and things. We had a very nice way of solving linear equations and detecting singularity and things like that. I'd have to do some looking at my notes but, you know, I'm sure there were other areas, like solving differential equations, there were subroutines for solving nonlinear equations. These things were sort of gathered together, documented according to standards that the User Services group developed, and then documentation was stored in filing cabinets. It wasn't online things like today's standards, so everything was run through a filing cabinet, you made copies and then a user could come down and hunt through a master list, and say, "okay I want a subroutine for solving differential equations, I want to use the Runga Kutta method, and here it is in the filing cabinet," and take a copy of the document and go off and then go from there.

HAIGH: And was the same thing true with the IMSL library when that arrived, that it was supplied on punch cards and the users would just pick up the procedure they needed?

AIRD: The IMSL library was on the hard drives. IMSL provided a loose leaf format manual with the documentation, so I think users could take the documentation and perhaps make copies of it but I don't think you had punch cards versions at Purdue, we had it on disk. Disk space was still at a premium, but some of these packages that were developed were valuable enough to actually use the disk space for that storage, and then there were library concepts that people could get a subroutine from the disk and load it into their program.

HAIGH: Now one important feature of this job in terms of your later career must have been that you were working with users who didn't necessarily have any background in programming. The experience had given you quite an exposure to the kinds of programs that ordinary scientific users were trying to write.

AIRD: I would say it was another enlightening part of my career as far as the element of being easy to use. For years all we talked about is the easy to use version. Users would come in and say "I've got an error here and it says that some element is small, but I don't know what that means in terms of my problem, why don't you translate that into terms that I can understand." We would think about that, and then there were maybe arguments to the subroutine that were somewhat obscure. So it was clearly a period where we were putting more and more emphasis on making the subroutine's arguments meaningful and error messages meaningful in terms of what a general user would know about the problem they were solving, and not necessarily about the algorithm they were using to solve the problem.

HAIGH: And were users in general keen to use library routines rather than write their own?

AIRD: I would say yes. Some of the people were like Michael Rossman, I think he wanted write all of his own stuff, but I would say by and large the users at the University appreciated having available good quality software. And I think the key was that much software was buggy in those days. There were algorithms that just failed miserably and people didn't want to have anything to do with that. But once they saw a good quality library of routines and were getting good feedback from other users then I think it was successful.

HAIGH: And did you have problems with users not understanding the mathematics in the subroutine that they were attempting to use?

AIRD: Well yes, and I think that was part of the problem. The early routines were often written with error messages that conveyed things that were part of the algorithm and not part of the user's problem. Part of the overall education was to learn how to translate error messages from the algorithm into meaningful messages of what does it mean in terms of the problem. We always expected the user to understand that they were solving, let's say, a system of linear equations, $AX = B$, they had to understand they had a matrix A and a vector B, and they were getting a solution X and there were times when the matrix was singular. But they didn't necessarily understand what a small pivot element would mean. But maybe even to a greater extent with other algorithms as far as having subroutine arguments that were meaningful to the user and having error messages that were meaningful to the user in terms of the problem the user understood. That was, I think it was a key breakthrough, or if you will, concept in software in that era. Today I think it's sort of taken for granted, but in those days you went from not having any algorithms available to having algorithms that occasionally had an obscure error message to finally getting algorithms that were very clear and easy to use.

HAIGH: So moving then on from the computer center side of your time at Purdue to the graduate program that you were doing simultaneously. You had mentioned that you chose Purdue because a large number of faculty with relevant research interests were there.

AIRD: Yes, absolutely. John Rice was one of the key people, and so I asked him to be my major professor and undertook a program in numerical analysis. My specialty then was solving nonlinear least squares problems in a global sense. John was famous for developing polyalgorithms, so that meant instead of just one algorithm for a specific problem you could write a program that would have some logic or some elements of knowledge of the problem area, and put together one of several algorithms. So you combined several algorithms in nonlinear optimization, and we combined a viable search strategy into that, so part of the thesis was to document this process and to write a program, which we did.

HAIGH: So that was your thesis topic?

AIRD: That was my thesis topic. Global solution of nonlinear least squares problems.

HAIGH: So in a sense that sounds like artificial intelligence kind of technique, as the program is trying to figure out for itself which is the best technique to use.

AIRD: Yeah, in a very rudimentary form. Not by today's standards of artificial intelligence, but it was a start in that direction. And it was successful, I think. Many problems could be solved by the polyalgorithm that could not be solved by a single algorithm, and so rather than a user having to try lots of different things you were taking advantage of the power of the computer to try different things and find something that worked.

HAIGH: Is there anything you remember from the course work for the Ph.D. that had a major effect on you later?

AIRD: Well I took several courses from John Rice, and I can remember in numerical analysis learning about things like backward error analysis that at first seemed a little bit strange but then seemed to be a very elegant and practical way of looking at what you can actually expect from an algorithm. In other words, you can't always expect that you're going to get a perfect solution to some problem that you think you're solving, when you really have the elements of finite arithmetic and rounding errors. Even the input from the user is sometimes very inaccurate, so the concept of backward error analysis was fairly impressive to me.

HAIGH: And was that based on work that had been done since your masters degree?

AIRD: Well that was, you could say, that was part of the course work at Purdue for the Ph.D. program.

HAIGH: No, I mean backward error analysis is associated with Wilkinson, am I reading that right?

AIRD: Well, yeah, what you do is you look at the math, you have a problem and you have a solution and you try to look at the solution as the exact solution of a specific problem. Then you try to see how far that problem is from the problem that you originally tried to solve, and you look at the differences there and if the differences are within your experimental error, the accuracy of your data, then you say that's about as good as you can get.

HAIGH: So what I meant with the question was that this wasn't an area that they taught you in Michigan in 1962, 1963 –

AIRD: No, I didn't learn that concept until Purdue.

HAIGH: But there had been important advances in the field itself between those two points, so that there were some techniques and methods that were better developed in the late 1960's than had been in the early 1960's? So the algorithm was building on new developments in numerical analysis?

AIRD: Yes. Maybe what I should comment on was the work that I did with Bob Lynch. There was a time when John Rice was on sabbatical and I was still a student and the concept of errors was kicking around. Lynch had some ideas and I worked with him and we wrote a paper that was published in ACM TOMS in 1975, on Computable Accurate Upper and Lower Error Bounds for Approximate Solutions of Linear Algebraic Equations. [Aird, T. J. and Lynch, R. E., -Computable Accurate Upper and Lower Error Bounds for Approximate Solutions of Linear Algebraic Systems, ACM TOMS, Vol. 1, No. 3, September 1975, pp. 217-231]. This was maybe the forward error analysis, so now you're looking at intervals for your input data. You have a system of linear equations, let's say, and you know that your coefficient is somewhere between 1.99 and 2.01, and another coefficient has a certain range. We developed algorithms that gave rather practical estimates on the bounds on the solution for that problem. Now that work I would classify as forward error analysis. I think it was very, very interesting technically and from a numerical analysis point of view but the lesson that I learned from that was that very few users are interested in that concept. Even to this day, I think, users are willing to accept an approximate solution to a problem rather than delving into all of the rigors of finite mathematics and bounds on things. We wrote a program and made it available. Honestly I don't know of anyone at Purdue that ever used it other than a few numerical analysts that were fooling around with these things. Even thirty or forty years later there still seems to be very, very little interest in this topic.

HAIGH: So that's an area where the numerical analysis community hasn't been able to persuade the users to take a great interest?

AIRD: I would say yes. There are people who spent their lives at this are of interval arithmetic, and have done an excellent job. The work that I did with Lynch was only a few months but from my point of view the lesson learned was that, as a practical matter, scientists and engineers don't want to get into that level of detail of numerical analysis.

HAIGH: And was Lynch a member of the mathematics faculty?

AIRD: Lynch was part of the computer sciences department, he was a professor.

HAIGH: So actually your return to Purdue was at the point where your affiliation in terms of department shifted from mathematics to computer science?

AIRD: Yes. At Olivet College there was nothing like computer science. I was a mathematics major at the University of Michigan, I don't think they had a computer science department at that time, they had a mathematics department. I believe Purdue claims to be the original computer science department offering a Ph.D. in computer science, and it started a few years before I got I there. I arrived there in 1967 and they were already a fully fledged Ph.D. program.

HAIGH: John Rice just published a revised version of an article on the history of that program, so some of the people there have written about the history. I was thinking more on a personal level though. You didn't go back and do a Ph.D. in mathematics, you decided to do one in computer science.

AIRD: Oh, yeah, it was very clear to me at that point in my career that my main interest was programming, numerical analysis and algorithms.

HAIGH: And the inspiration to do the Ph.D. was it that people with Ph.D.s had been doing the analysis on the problem itself and then you had been doing the programming. Was the motivation that, you know, that you wanted to do the things that they were doing and that to do that that you needed to get a Ph.D.?

AIRD: Maybe a little bit of that but I would say it was more of just wanting to learn more about numerical analysis and more about programming. I had no desire to go off and be an engineer or work in an application area, my interests have always clearly been in what I might call the application independent area of numerical analysis and programming, and Purdue seemed to offer exactly that program.

HAIGH: So you've mentioned Lynch and Rice. Were there any other faculty there that you worked with closely or were particularly impressed by?

AIRD: I was always impressed by Carl deBoor. Carl developed a very good subroutine for numerical integration - CADRE. Before his work was done this was an area of difficulty for the computing center, and he did such a bang up job I know there's lots of papers published on that work. He made it easy to use, it had error handling and it solved just about anything that you would throw at it, so it was a wonderful piece of work.

HAIGH: So in that sense do you think that the kinds of problems that people were experiencing in the computer center were having an influence on the work that the faculty were doing?

AIRD: The faculty of, you mean in other parts of the University?

HAIGH: Within the computer science department.

AIRD: Run that by me one more time.

HAIGH: It sounded like what you just gave me was an example where certainly the program done by member of the faculty was very useful –

AIRD: To the general users -

HAIGH: For the general users, via the computer center –

AIRD: Yes it was –

HAIGH: And that once the program was produced it was used and appreciated. Now I was wondering if you could go beyond that and say that the process was going the other way and that the computer science faculty's ideas of what problems needed to be solved or what would be a worthwhile project to undertake might be influenced by knowledge of the problems that real users were having. Maybe it was a two way process?

AIRD: Well, I don't know if I could say I saw that relation. I would see Carl's work as a kind of a milestone in the development of really high quality excellent easy to use robust numerical software to solve standard problems in mathematics. That subroutine could be used in all areas of science and engineering throughout the university, which people were using for other applications for which we never really got into the detail. They would have to integrate a function over a certain interval and with this subroutine they could do that. So you could say, hey you know this worked, because even though Carl spent a lot of time and a lot of energy it had a big payoff and it helped a lot of users and it really set a new, set the bar you could say at a higher level for what you would expect from a subroutine in numerical analysis.

HAIGH: Right. And it did this by being more robust and easier to use, not by just being more efficient or better mathematically. So in that sense at least, were some of the faculty taking note of those kind of questions of robustness and ease of use? Not just coming up with a method that was exciting in mathematical terms but trying to present in a form that would actually be useful to people with real problems to solve?

AIRD: I would say yes, certainly Carl deBoor had that attitude, and John Rice had that attitude.

HAIGH: Yeah. And you've mentioned your thesis topic. Were there any other pieces of software or projects that you remember from your time in the Ph.D. program that you worked on?

AIRD: There were various subroutines developed before I did my thesis work. Again, there was some work in linear equations and some work in just nonlinear equations and some subroutines, but nothing really of note I would say.

HAIGH: Okay. Anything else that stands out from your time at Purdue?

AIRD: Right now I think we've covered that topic.

HAIGH: Alright. Now did you go and work for IMSL directly after graduating from Purdue?

AIRD: Yes.

HAIGH: How did that come about?

AIRD: I was pretty happy working for the computing center at Purdue. Of course I had this contact with the IMSL people, because I was a customer of theirs through Purdue University computing center. I had met Ed Battiste who was the president of IMSL and he called me up one day. I was close to finishing the work on the Ph.D. program but had really no intention of leaving Purdue at that time. I hadn't really thought about it, kind of liked Lafayette as a town and liked Purdue and liked the University. We had lived in Houston before, and my wife really liked the warm weather (Lafayette had cold winters) so she suggested to me that I consider this offer seriously. So rather than telling Ed, "Oh I don't think I'm interested", I said "well, let's explore it." I would say a big influence was the family's desire to go back to the Houston environment. Of course at that time IMSL was in the business that I was very interested in, it was mathematical subroutine development, so it seemed like a very good fit. So after some discussions, and after finishing my work at Purdue, we moved back to Houston to join IMSL.

HAIGH: And were there any other specific options that you considered?

AIRD: No, because at the time I had not thought about moving and IMSL made me an offer, and the offer was almost a perfect offer in the sense that it was a company doing work that I was very very interested in, and located in an area where the family had a high desire to move. So it had these two converging factors, and I said we're moving back to Houston.

HAIGH: Had you considered looking for a faculty post or more pure research kind of job?

AIRD: No I think all through my career to that point my interest lay mainly in programming, and developing programs -- in fact exactly what IMSL was doing. Even going to work was interesting at the computing center developing subroutines, the idea of actually building a business around this was also of interest, so I had you could say some business interest as well as the technical interest. I never really considered anything else.

HAIGH: What was Battiste like?

AIRD: That's an interesting question. He was a strong advocate for his company - IMSL. I believe IMSL actually formed in about 1970, so this happened before my time, and Ed Battiste was with IBM at that time. His research center was located in Houston, and was doing work on the scientific subroutine library. IBM was sort of de-emphasizing that work, and Ed felt that it could be a successful business so he got together with Chuck Johnson. Chuck Johnson was the financial end of the equation, and Ed Battiste was the technical end, and they put a company together and hired some people, and I would say in almost record time developed the initial product. Ed was a very dynamic person, he had a lot of enthusiasm for the business, had a lot of emotion for the business. Sometimes that emotion could be a good thing and sometimes it could be just emotion, but he had a passion for developing that company.

HAIGH: And what kind of impression did he make on you when you first met him?

AIRD: Well in a social setting Ed was just really a loveable character and you just had to have a good time with him, joking around. Then there was a technical side of his desires for the business that was consistent with mine, so it was an attractive relationship in that regard.

HAIGH: And what kind of condition was IMSL in when you joined it?

AIRD: Well I was rather young and I guess somewhat naive in the business sense and so I didn't delve into the financial well being of the company. I believe there was an original investment of about a million dollars from the Johnson family, and the company was still a long way from making money. They were still losing money, they were getting new customers but it was still a business whose viability was yet to be proved when I joined. It took a few years really of hard work to get there but gradually ISML added more and more customers, the concept grew, and I'd have to see some financial records, but it seemed that it was maybe in 1976, or maybe three and a half years after I joined IMSL, to where it was actually at the break even point.

HAIGH: There's some figures in your article in the Cowell book about, I think, a number of customers, a number of libraries, those kind of things. [Aird, T.J., The IMSL Library, Sources and Development of Mathematical Software, Edited by Wayne Cowell, Prentice-Hall, 1984]. I'm wondering also what was the firm like on a more subjective kind of level. When you joined it how many employees were there, what was the office like, what was it like to work there, those kinds of things?

AIRD: IMSL had maybe seven or eight employees at that time, they had had other people that maybe had been hired and left. The first person that had the job that I took was Olin Johnson. Olin left ISML to go to the University of Houston to be a professor, so that was what created the opening for me. My position was called Director of Mathematics, so I was the Ph.D. who was directing all the development work for mathematics. There was another person, Walt Gregory, who was the Director of Statistics, and then Ed Battiste who was the president of the company. And then there were programming support people that worked for either Walt or me, or sometimes both, and they were to do the programming. The Director of Math and Director of Stat were supposed to design routines, find algorithms, decide what work had to be done, and programmers were provided to support that function, and there were, I guess, three or four or five. It seemed like some programmers were doing non-programming work for other parts of the company, so you could say a small company somebody was maybe handling shipping and receiving at the same time that they were doing programming, so as the company grew those people perhaps migrated to the other position.

IMSL was located on Hillcroft in Houston in an office building, and had a remote card reader that could transmit programming jobs. Everything was still punch cards, you'd read the punch cards in, they were transmitted to Rice University to the IBM computer there. It was the same kind of thing that you had to write a test program, you had to write your subroutine, you had, although as I recall, we stored certain things on disk at Rice in those days. It was still kind of slow. IMSL was supporting, I believe three platforms at that time, they were supporting IBM, they were supporting CDC which was the computer at Purdue, and the Univac, and so you had portability issues because each of these machines had different sizes of floating point numbers, everything from 60 bits on a Control Data to the IBM that had 32 bits, you had to have either single precision or double precision. It was pretty clear right away that the trend was to suport more and more computers, there was no standard for floating point arithmetic, and each version of FORTRAN had subtle idiosyncrasies.

HAIGH: You mention in the paper that it was 1973 that the terminal was installed. So for the first three years, I guess before you were there, they actually had to get their stacks of cards physically picked up by a local service bureau, so it probably made a big difference when they first got access to the terminal.

AIRD: Okay. Yes, I'm sure that in the earlier days they had a courier deliver things back to Rice University, so it was even slower. So having that service was a big improvement.

HAIGH: And it confirms what you just mentioned, the first three versions were released in 1971 and 1972 for the IBM, the Univac and the CDC, and then the next version was in 1975 for Honeywell. Now at the time you joined was there anyone devoted to marketing or was that another thing that people would do in addition to their other roles?

AIRD: No. Looking back that was clearly a weakness in the business plan at IMSL, I guess there was this concept that if you build a better mousetrap the world will beat a path to your door. In the early days, I believe, IMSL thought they were developing these subroutine libraries for smaller companies because they felt that the bigger companies would want to develop their own. It turned out that that wasn't true, so they probably, looking back, didn't really know who their target customers were. And everyone in the company did a certain amount of, you could say, marketing and sales work, and sales were very, very slow. I think Ed Battiste did a lot of sales work himself. He was a good salesman but I did some sales work and I was not a good salesman, and I don't think my heart was in the activity. But at some point IMSL hired its first actual sales person dedicated to the process, and I don't know the exact date of this, but it seemed like it might have been around 1974. It still wasn't a marketing function as such but there was a salesman. It was found out that a salesman, this was a big breakthrough, actually can be good at selling a product even though they don't know all the technical details of the product. As I recall, and I'm probably overstating, it caught on like wildfire: they found that if they had two sales people they could almost sell twice as much as if they had one, and break even. I'd have to look up in my records, but it seemed like it was about 1976 when that concept was in full motion and the company was actually at a break even point.

HAIGH: That was a great moment in the history of the company then?

AIRD: Absolutely, yes, very exciting.

HAIGH: So presumably the person that you would have to sell to would be the computer center manager, is that true?

AIRD: Yes, in those days the big companies and the universities had computing centers and if you were in sales then you were talking to the computing center personnel.

HAIGH: Can you remember what kind of arguments you would use to persuade them to buy it?

AIRD: Well I think the argument was that this was good quality and it would save you money because for a few hundred dollars a year you wouldn't have to be developing your own. If you developed your own you would not have the same expertise, we had world expertise, we had an advisory board of world experts in these areas, and full testing and so it was worth the investment. On the other side I think that there was a concept of, probably because of the Scientific Subroutine Package, that software was in some sense free. So as I recall, in the early days that was the competition, people say well we have SSP so we don't have to pay anything, but there was article after article after article about how bad the SSP package was and there was no support for it. Yet it still took a lot of effort to overcome that.

[End of Tape 1, Side A] [Start of Tape 2, Side B]

HAIGH: So to pick up on what you were saying. You had suggested that the biggest selling point was that it would be cheaper than trying to produce the software in-house, and you had mentioned that the biggest source of resistance was the idea that software was something that you got free and you shouldn't have to pay for?

AIRD: Yes, I would say those were the arguments. Here was going to be a better, more reliable, solution to your problems for a reasonable price. Then the competition was, well we have this free software, so we had to overcome that.

HAIGH: And do you remember any specific methods that were used to overcome that?

AIRD: Well I think as more and more studies were done we could show users, hey here is a subroutine in SSP and it does lots of things but it just doesn't solve your problem. It's not reliable, and I think we pushed that as much as we could, and the sales department, when it came about, would push that. In the early days, it was a difficult sell and then it became easier and easier. Over maybe the course of maybe three or four years, it just became that no one would even argue with you,.

HAIGH: So would the early users of IMSL themselves tell their friends about it, was that an important process?

AIRD: Well, I think in some sense there was a reputation developing for the IMSL library as a high quality product. It's hard to say exactly what role that played, but I'm sure that it played a big role because the customers were really scientists and engineers and maybe technical business people that were writing their own FORTRAN programs, so they had access to common documentation, common journals, and things. So anything that promoted the high quality of the IMSL library helped.

HAIGH: Yeah. Actually I was interested in your reference to studies saying that SSP was inadequate. Where would those kinds of studies be appearing?

AIRD: That's a good question. You know, I can't think of a specific reference at this point, but I know there were people that were solving problems and showing us…. You know, it might have been that we didn't have a formal publication, that we just had a collection of cases where the SSP didn't do very well.

HAIGH: So that would be evidence that IMSL itself would be collecting and passing on to potential customers?

AIRD: Yes.

HAIGH: So I think that gives us a good sense of what the company was like when you first joined it in 1973. Now other than the addition of a real sales team how did it progress over the next few years?

AIRD: Well there was clearly an issue of portability, it was the fundamental focus in the business problem that had to be solved when they had an IBM version and a CDC version and a Univac version. They were actually three different source code files, and this clearly would become unworkable. We didn't know it at the time how many platforms we would support but looking back it was just an enormous number of platforms. Each one had a slightly different floating point base and maybe slightly different variation in FORTRAN.

Let me step back. At Ed Battiste's prompting, we applied for a National Science Foundation scholarship to study portability, mainly because it was an interesting issue in the community at large, but it was also something that IMSL needed to do for its business. So that study then developed techniques in software for handling portability, much like the today's preprocessors, except more tailored to the exact problems that came up. What were the problems? Well for each different environment, each different platform, you had to know certain numerical characteristics: you had to know the precision of the machine, so a constant known as machine EPS, the value was ten to the minus 16 for IBM single precision and ten to the minus 32 for IBM double precision, ten to the minus 28 for the CDC 60 bit environment, and something else for Univac. So in your algorithms you couldn't try to find twenty digits when you were working in 16 digit arithmetic, you had to have your algorithm rely on those numerical constants. All of that stuff was built into the preprocessor that was developed to solve the portability problem, the largest floating point number, the smallest floating point number, etc. Machine EPS was probably the single most useful, but there were lots of other parameters that describe the environment and were built into the preprocessor. So if you wanted to include machine precision, you had a code that you put into your source code, and the preprocessor would come along and would say okay you're running on CDC we'll go get the number out of that table and bingo we'd plug it in.

HAIGH: So essentially you would define some constants and then the preprocessor would include the appropriate pieces of codes on a conditional basis when the program was compiled?

AIRD: Yes. So what the big advantage was that instead of having separate source files, which becomes almost unworkable when you get to four of five separate versions of your algorithm that are almost the same, we had one basis deck (today you would say basis file, deck was used at that time since most files were decks of cards) and almost all of the lines of code were the same. If we had something that we had to do that was specific for Univac we could write "C$ IF Univac THEN do this," but we tried to avoid that. We tried to program in what we called the portable subset of FORTRAN. But then there were other things like machine EPS, and largest number, smallest number, and so on, that would just be plugged into in your algorithm. Ninety-five percent of the lines would be just common FORTRAN, you could read it almost like FORTRAN, and then you could preprocess it and produce any one of the numerous versions.

HAIGH: Towards the beginning of the 1984 article I think –

AIRD: A count here, we have twelve different platforms at this point, and it seems to me there were probably even more, and so clearly solving a portability problem was really essential for the business to work.

HAIGH: Yes. And at the point at which you joined, had the code base for the three different platforms drifted substantially apart?

AIRD: A little bit. They were still much the same, but they had to maintain three separate decks of cards, one for Univac, one for CDC, and one for IBM.

HAIGH: Right. And this system you've described that would be what you called in your article the IMSL FORTRAN Converter? [This system is also the topic of Aird, T. J., -The IMSL Fortran Converter: An Approach to Solving Portability Problems, Lecture Notes in Computer Science, Portability of Numerical Software, Edited by Wayne Cowell, Springer-Verlag, New York, 1977].

AIRD: Yes. That was the result of the National Science Foundation funded portability study. Ed Battiste was the head of IMSL, but I was certainly instrumental in developing the converter itself.

HAIGH: And was it unusual for a for-profit company to receive an NSF grant of that kind?

AIRD: It seemed to me it took a lot of convincing and it wasn't impossible but it was less likely than a research organization. IMSL was successful and it probably was a key event, I would say. I don't know, looking back, what would have happened if IMSL had tried to fund that work on their own. Maybe they would have had to do it, but this was a wonderful thing and I think at that time the whole math software community benefited by all these portability studies that were going on.

HAIGH: Yes. Can you remember how big the award was?

AIRD: It was about $60,000 over a two year period.

HAIGH: Sure. Now from what you've said there it also seems that IMSL perceived itself as being very much a part of this community of mathematical software producers. Can you talk about that side of it?

AIRD: Well there were a lot of sources of algorithms, again, during any particular year it might have been the ACM *Communications* page on algorithms, or *TOMS* algorithms, or Numerische Mathematik. IMSL did a lot of work to maintain professional relationships with the people associated with these activities. We had a good relationship with John Rice, and had an advisory board, kind of a who's who in numerical analysis. In some sense that was maybe the marketing part of the company, to promote itself through the experts that were the advisory board, to use those advisors to gain insight into the best algorithms that they could find. Of course you had to put together various pieces, an advisor might have a good algorithm, but maybe the advisor wasn't exactly the best in programming details, so we took kind of the best of these various experts and tried to put them together into a cohesive business climate.

HAIGH: Sure. So I think you implied there that the role of the advisory board was partly in marketing –

AIRD: Well, it was supposed to be a technical advisory board but I would say that if you looked at what its actual function was it served a marketing function, because it gave this company that people hadn't heard of some credibility. Here's our advisory board of ten or twelve experts in the whole world, if they heard ofone of them then that would give IMSL credibility.

HAIGH: Yeah. So what motivated the experts to sit on the advisory board?

AIRD: Well they were given a monthly stipend which was pretty small, probably not enough to make anybody do anything that they didn't want to do. So I would say most of them were enthusiastic about IMSL and about the process of developing good mathematical software, and therefore their technical interest probably was the convincing point.

HAIGH: And you'd also suggested that they would actually be suggesting mathematical methods and other things that should be incorporated into the software.

AIRD: Yes, I would say from time to time and, again, there were some advisors like John Rice, who were very, very active and others who you didn't hear from very often unless you contacted them. The idea was that the advice could come in at any level there was no preconceived idea. An advisor could simply write a note and say I think you should improve this or there's an algorithm in this journal that I think you should implement.

HAIGH: And how active were you and the IMSL people in attending conferences, presenting papers, that kind of thing?

AIRD: Reasonably active, not overly active. There was a math software conference at Purdue that I know I attended and talked about various things, there were conferences here and there.

HAIGH: So, reasonably active in numerical analysis and software?

AIRD: Well maybe more active in conferences that would attract the users, rather than specialist numerical analysts. I know we attended IBM user group meetings and tried to give presentations at that meeting. There was, let's see, I think it was a Digital Equipment user group meeting. There were various user group meetings and our point wasn't to go out and impress the numerical analysis community, but to kind of convince the user community that this was a fine product and well worth their investment. And also to get feedback, of course, from the user group.

HAIGH: We'll talk more about the relationship to users later.

AIRD: Okay.

HAIGH: That's an important topic. How about the relationship to the projects that were coming along at that point, such as EISPACK and LINPACK?

AIRD: Well we were interested, and we did some attending of meetings, but I don't think we were ever the central force. But we were supportive of those activities and attended various meetings, and we benefited by having those high quality EISPACK and LINPACK subroutines available.

HAIGH: So were those subroutines incorporated into the library?

AIRD: Yes. They were definitely the highest quality. They were public domain algorithms and so IMSL used them, I think referenced them, acknowledged the support from those various groups. But we put our own front end, and built it into the standards of the IMSL library.

HAIGH: So in practical terms, what would be the benefits that a user would get from using the versions in the IMSL library as opposed to the free standing ones?

AIRD: Well the benefit would be the uniformly documented, consistent presentation. Let's say you were already a IMSL customer then you have the library, it's on your system, it's compiled for your system. If all they wanted was one EISPACK routine and they didn’t have a budget, well then they could go and they could get it, but the cost of doing that was almost as much as you’d pay for having the IMSL library for a year.

HAIGH: So the big benefit would be this integration and support?

AIRD: Yes. It would be guaranteed to be compiled on your machine, integrated with other routines that you were used to, you’d know where to find it, you’d know that it was tested in your particular environment.

HAIGH: And would the authors of routines that were incorporated into the library receive any payment?

AIRD: No, there was no direct payments that I recall. I know that at one time Cleve Moler was an advisor to IMSL and so he was getting the standard advisory fee, but there was nothing that I can recall in any of those areas of specific payments for specific routines.

HAIGH: So the only people who would be paid directly would be programmers actually working for IMSL itself?

AIRD: Yes. I don’t believe there was any outsourcing in those days. We’d get the raw materials in and hone them to fit into the IMSL library, and it was all done in-house by IMSL employees.

HAIGH: And do you remember any cases where authors might have been reluctant to have their work redistributed?

AIRD: I can’t think of anything until much, much later. I don’t know how far in the history you want to go, or how obscure. There was one case where IMSL had contracted with an author and made a payment that was a one time deal, so we can use this any way we want. Then there was some objection later on maybe, but that was fairly obscure and it’s the only case that I can remember in the history. Certainly there was nothing that I can recall with regard to anything in LINPACK or EISPACK, or of the public domain stuff.

HAIGH: Okay. So through this whole period the academic and national lab people who were involved in producing these kinds of public domain software had no objection to its commercialization?

AIRD: Not that I recall.

HAIGH: That’s interesting. Now returning to that question of IMSL’s relationship to the mathematical community. You’d mentioned that EISPACK and LINPACK were incorporated because they were the best implementations of software in their respective areas. Were there cases where researchers would come up with a new and improved mathematical method and then IMSL programmers would need to incorporate that into the library?

AIRD: You mean specifically in the areas covered by LINPACK or EISPACK?

HAIGH: No, more generally.

AIRD: More generally. I'm sure that that happened, there were new additions maybe every year to year and a half, and new additions would include new routines and improvements to old routines. At this moment I can't think of a specific case but, you know, at ISML part of the process along with the portability was to develop better and better testing. So we had a system, because if you ran this on ten different machines then you'd have all these listings, so we developed a concept of tests, and minimal tests. We were actually taking the output, the answers, and doing checksums and things so there was a kind of automatic pass or fail. There were some obvious numerical considerations and there was this number within this range, so there was better and better testing and as we developed more and more tests for algorithms that were already in the library problems were discovered and improvements were made. So it was a constant ongoing activity to find and improve the best algorithm and improve it.

HAIGH: Right. So it sounds from what you're saying that the majority of the improvements and additions to the library would be in terms of improving the software itself, not because there had been some substantial improvement in the understanding of the underlying mathematics?

AIRD: I can't recall specific instances now, but before EISPACK came along obviously there were algorithms for eigenvalue routines and EISPACK was a study in both the mathematics, the numerical analysis, and in the programming, and it came out with these better algorithm systems. This process was going on in other parts of the library at the same time, so whether it was a user calling and saying "Here's my problem and this algorithm of yours doesn't do a very good job of solving it," or whether it was a publication in *Numerische Mathematik*, all of those things were coming together within IMSL and going into the programs.

HAIGH: Okay. Let's fall back from that and talk about the library itself then. You'd mentioned that when you joined there were three versions and, I think, you have said that there were about four programmers.

AIRD: Roughly speaking, yes.

HAIGH: So where had the programmers come from, what was their background?

AIRD: I think that at least three of the four programmers were people that had worked for IBM, for the center that Ed Battiste was involved with. There was Toby Yeager and Nancy Boston, both of those people were programmers that worked with Ed Battiste at IBM. There was Larry Williams, I think he might have been with IBM, but I can't be sure. And, Eunice Chow, she may have just been hired from Houston to work for IMSL as a programmer

HAIGH: So those would be some of the same programmers who had worked on SSP?

AIRD: Yes, certainly Nancy Boston and Toby Yeager, certainly those two I know for sure, and maybe Larry Williams.

HAIGH: You had mentioned that later on LINPACK and EISPACK were incorporated. Now with the early version of the library would the, would all or most of the routines have been written in-house by this group of programmers?

AIRD: I would say yes, the majority of this came from perhaps algorithms in Communications of ACM or Numerische Mathematick, prior to my joining IMSL. So there were already a good number of routines. In 1971 was 200, 1972 it was 240, 1973 it was 280, by 1980 it was up to 500, so those first 200 to 240 algorithms were all completely programmed by the IMSL people.

HAIGH: Right. And these routines would have been based on published pseudo code or ALGOL versions in the sources that you mentioned?

AIRD: Yes.

HAIGH: So the focus was really on just having improved portable and efficient implementations of these same kinds of algorithms, which would already have been available from other sources?

AIRD: Yes, I think the value added by IMSL was the fact that these algorithms were documented in a consistent way and tested thoroughly. Tested and reliable at a reasonable cost. I don't want to get too far off course here, but in the history of IMSL there was always a question about whether a customer really needed 250 routines or do they need 500 routines. Whereas logic would say "well, probably not," the marketing forces and things that worked turned out that that was what customers really wanted. They just wanted to have the entire library and even though they weren't going to use a whole big bunch of these routines, if they needed something it was there, so that's what they wanted to buy at a reasonable cost.

HAIGH: One of the things that was interesting to me from your article is that the library, at least to that point, was only licensed as a whole.

AIRD: Yes.

HAIGH: Did you ever think of breaking it into pieces and selling subsets, for example, the mathematics and statistics software separately?

AIRD: We talked about that all the time. I believe those discussions probably started the day I arrived at IMSL and they were still going the day that I left. For whatever reason, I don't think the company could ever find a way that would be satisfactory in the distribution and customer needs. Even after they separated math functions from stat functions, there was a deal that you could pay so much and get both parts, and that's what people tended to buy. So I don't think they ever solved the problem, and probably they haven't today, of selling anything less than just this big library. If you have 500 or 600 routines, and you need two of them, what you've paid for the library probably makes it worthwhile. If you need ten of them then, you know, you've got a windfall profit.

HAIGH: And when did the separation of mathematics and statistical happen?

AIRD: Well there was a major revision, and major redo, of the library that was released in about 1987. At that time there was a math library and a stat library, but still sales continued to sell both parts and maybe gave too much of a discount if you bought both parts, so nobody just wanted math or stat, there were very few sales like that. There was some suggestion that "gee we're getting so many routines, we've got so much here, and nobody can use this, there's got to be a way to divide this up and do a better job of getting customers what they really want."

HAIGH: Okay. So other than that separation, which would really have been quite late in the history, through your whole time at IMSL the library was just sold as one thing. Then eventually as two things, but most people bought it as one anyway?

AIRD: That's right.

HAIGH: And the other thing that struck me that could have gone another way, is that the software was leased on an annual basis rather than being sold outright. Do you know if thought was ever given to selling it or providing it on a different kind of basis?

AIRD: The idea was that you lease but a new edition was coming and therefore you were supported. As opposed to just sending something out there and then a year and a half later you got fifty new routines and you've improved another fifty of them and then you have to go sell the new release. The idea of a lease worked very, very well. IMSL was compounding growth that occurred during the later 1970's. It was really a wonderful business model and seemed to serve the interests of the user community because they were automatically supported, you leased this product and therefore when the next edition came along you were going to get more. I'd have to look through some more detailed notes because it seemed to me that let's say up to 1987, that was pretty much the way it was. Then the PC model came along and so obviously today there's paid up licenses.

HAIGH: Yes, I understand that leasing was the way that mainframe software was pretty much invariably supplied. Perhaps because that's what IBM itself did it when it unbundled their packages –

AIRD: Customers were used to that concept.

HAIGH: When historians study the history of the independent software industry, they tended to point to IBM's unbundling of hardware and software as an important event. Was that something that was in people's minds as creating a new opportunity at that point?

AIRD: Well, I'm trying to remember the sequence of events. It seemed to me that the only piece of the IBM offering was the scientific subroutine package that affected IMSL, and it seemed to me that, at least at some point, became a non supported, freely available library, and that was significant. I don't honestly remember any other financial events that really had any impact on IMSL.

HAIGH: So it sounds that SSP really was fading out in the 1970's. Did IBM come up with any new or more competitive products, or did they just let the scientific market go to independent firms?

AIRD: No, I think they let it go. SSP was unbundled and unsupported, and it just wasn't of the quality that you might expect from mathematical software that had been developed in the early 1970's. There were obviously pieces of it that worked fairly well, but there were many, many pieces of it that didn't, and without support it was just a danger for any user because you didn't know if you were going to be using something that didn't work. There was not going to be an improved algorithm, so as the years went by it became easier and easier. I think this happened from 1973 to maybe the late 70's, and at some point it was kind of a slam dunk that there was no more discussion about SSP, they just faded off the charts, but during the early years it was definitely a competitor.

HAIGH: Okay. So back to the library. Now you have mentioned that there were something over 200 routines in the library when you came, and you talked about the sources of them. I was wondering, did any of the users in computer centers ever spontaneously offer either improvements to the IMSL code or new routines that they thought would be a good addition to the library?

AIRD: Well I know that when I was the customer of IMSL, that I offered suggestions for certain routines, to improve certain routines in the library. I probably even made suggestions of areas that they could add to, so I would say in general yes.

HAIGH: So it would be on the level of a suggestion?

AIRD: Yes. I don't recall anyone actually saying here's working wonderful code, but, you know, users would say, "Hey you need more in nonlinear optimization, or you need something that's very special and very fast for symmetric systems", and suggestions like that.

HAIGH: So there wasn't anything like with today's open source software where users would make changes to actual code and feed it back into the system?

AIRD: Not to my knowledge. The IMSL library, as we maybe noted here, was always distributed as source code, so users always had the source code, they could always look at it. We encouraged them to not make changes and suggested that if you modified it that it would be very difficult for us to support. So I don't remember a lot of users doing much more than, even though they had the source code. They used it, and reported a problem when either it didn't give a meaningful error message or it gave an error message when it should have returned an answer. There was always a very close relationship between the users and the designers and the programmers of each subroutine, so that information went right to them, right to the source.

HAIGH: And was it technologically impossible at that point to provide routines in a way that wouldn't have required giving users the source code?

AIRD: Well I think at some point it became technically feasible. There were binary libraries, and such, but there seemed to be a customer demand for source code so I think the company was reluctant to not supply it. There was always a discussion of, "okay the source code is out there and somebody could get it and they could do this and do that," but to my recollection, during my time with the company, none of the bad things that could happen ever happened. The customers appreciated having the source code out there, and felt more comfortable and possibly gave them more flexibility as far as which compilers and things like that. Even though we had many environments we didn't necessarily support every variation of every compiler.

HAIGH: Can you remember at what point it would have become technically feasible?

AIRD: Well not an exact date, but it seemed to me that by the late 70's it would have been technically possible to have a binary library on most of those machines, but for these other reasons that we mentioned, it was deemed not doable.

HAIGH: Sure. Do you have an impression of what proportion of users might have ever actually read the source code?

AIRD: Well an impression would be very few. They appreciated having it, it gave them some confidence, but very few of them got into reading the code.

HAIGH: Okay. And to loop back to the questions I raised a little bit earlier, in terms of sources of routines. You've mentioned that the initial batch of routines were produced exclusively in-house, and you've also that mentioned some big projects like LINPACK and EISPACK, were incorporated into the library. Were there any other sources of routines or was that it?

AIRD: Yes, in-house didn't mean devising algorithms but getting algorithms from sources like *Numerische Mathematik* or *CACM*, and later on *TOMS*, and then kind of fine tuning the code or taking the algorithm into code. Of course when EISPACK and LINPACK and these things came along this was much closer to what you would call the end product, so all that IMSL had to do was really put a front end on it and there'd be some error handling.

HAIGH: Alright. Were there any other external libraries that were incorporated apart from LINPACK and EISPACK?

AIRD: Not to my recollection. In fact, most of the work in statistics was even more in-house development than the mathematics, because there weren't the same level of sources of these statistical algorithms. So that work, maybe even the algorithmic work, was done in-house although a lot of the statistical algorithms use underlying mathematical algorithms.

HAIGH: So was there much overlap between the work of the two groups? Did you have to coordinate things closely?

AIRD: The two groups being mathematics and statistics?

HAIGH: Yeah.

AIRD: If you start in 1973, we had Walt Gregory, Director of Statistics, and Tom Aird Director of Mathematics. We had 4 or 5 programmers that we shared, and so there wasn't really very much to coordinate at all. IMSL had certain conventions, naming conventions, argument conventions, documentation conventions, and so we followed those. There wasn't anything to work closely on with either group, work went on separately but with these standards being enforced. Later on there were much bigger groups of statisticians and mathematicians and if anything the statisticians made use of some of the underlying mathematical algorithms in their work. But then for some of these things, like random number generators they would get an algorithm from a publication or a friend or a colleague, or devise something themselves, and do the basic programming. So, again, I would not say much of an issue of coordination between mathematics and statistics.

HAIGH: So the two groups were largely separate?

AIRD: Yes.

HAIGH: And how about the development of the library over time? I mean, you had mentioned that the number of routines was constantly growing, and that that was used to sell the library. In the article you wrote that the library was divided up into "chapters" and that each chapter would contain routines that would cover some more kind of ground. So what kind of new areas were covered over time?

AIRD: Well, I can look at the chapter headings of the manuals and things. One thing we might want to discuss first, there was sort of an early organization of the library into chapters that started with certain letters, L for linear equations, S for statistics or something. There were certain conventions and FORTRAN was really FORTRAN 66, and then as over the years it became clear that FORTRAN was progressing and that users were expecting more from error handling, they were expecting more from the interface, and they were expecting more from storage allocation, and so there was a major rewrite of the library that was released in 1987. And so the chapters were changed. There was a math library and a stat library, major enhancements, I would say, to the usability especially with regard to passing work storage, and with regard to error handling.

HAIGH: So that was the thing that was the single biggest change in the history of the library?

AIRD: Absolutely. It was a major, major undertaking, and you know, major enhancement made the library from something that had a 1970 historical feel to it, to something that was really more of a modern language. FORTRAN still lacked in its entirety dynamic storage allocation and such, but the library that was released in 1987 addressed that issue pretty well I think.

[End of Tape 2, Side A] [Start of Tape 2, Side B]

HAIGH: And between the regular releases of the library, would it be common that there would be rewriting of individual routines or they would mostly stay how they'd been written, and the change with the new release would just be the addition of further routines?

AIRD: Well I would say there was a constant level of communication between the designers and programmers for any given routine and the customer. So that information was fed in, and we always had a concept that there was a change manual and you go in and you could make changes, or if it was a subroutine change then you could actually make an enhancement and you might send it to that user specifically, but then put it in the queue for release in the next edition. We were always trying to improve this product to make it as good as it could be - for documentation, for numerical accuracy, and for ease of use.

HAIGH: So a lot of the changes would then be motivated by the needs of individual users?

AIRD: Yes, I would say user feedback was a real big part of it. As hard as one would try, a user might say, "this error message just doesn't mean anything to me, could you explain it to me." Okay if you explained it to them then they might say, "oh yeah I could change the wording in the manual and make it clear to everybody else" so you had the manual on the shelf at the office and it was getting all these red lines in it. When you had a new edition those would all go back and be reviewed and put in the new edition. If it was a coding change then similar things would happen.

HAIGH: Okay. So then you would be making constant streams of little updates and improvements to the existing routines?

AIRD: Yes.

HAIGH: And how about the new routines. You mentioned, I think, that the number had gone up from 200 and something to around 500.

AIRD: Yes, by 1980 if was 500.

HAIGH: So it doubled in size. What were all the new routines doing?

AIRD: We had a lot of new special functions added during that time, that were a lot more specialized. In other words, instead of having just a linear equation solver you might have a liner equation solver for full matrix, you might have a linear equation solver for symmetric matrices, for positive definite matrices, and other special categories in linear algebra that apply. So we were filling in, because in each area if you have a special property of a matrix like positive definiteness you can exploit that. There were better algorithms, so you can do a faster or better job numerically. We were filling in those special areas. Then there were areas of optimization where we really never had enough coverage. You could have one nonlinear equation and one unknown, you can have a system of equations, you can have nonlinear least squares, you can have constrained nonlinear optimization, and so there was an attempt to fill in the needs based on users calling in and saying "I need this," so we would put it on the list.

HAIGH: And would the number of programmers working for you have changed over the 1970's?

AIRD: Well I think the company started to grow fairly substantially in the late 1970's. You can take 1987 when the new release came out and I can make a guess that there were ten Ph.D. level people, mathematicians, statisticians, and so on, working as designers, and there were probably twenty programmers that were working in development to support them.

HAIGH: That's interesting. So all the way through to the 1980's you maintained this type of split between the Ph.D. mathematicians and the coders?

AIRD: Yes. It was designers, Ph.D.'s were designers, the developers that we called were generally not Ph.D's. Not that there was anything good or bad, it's just that most of the designers were interested in designing algorithms in this higher level work and weren't so much interested in a detail level that has to go into writing the programs and adhering to all the standards and doing the error handling, and writing the test programs. The designer would say, for example, "okay here's ten tests that you need to run then the programmer would write test programs for those." Like I say, we not only had the system for running the test and printing out answers, but we would actually feed the output into a processor that would do a checksum, and in turn produce either a pass or fail message. So we could run hundreds of tests on a given machine and immediately we could see either pass or fail, we would run some specific tests but at least that would give us the initial answer. So there were those tasks that were inherently strictly programming, and those tasks that were inherent strictly design, and there was a lot of crossover. So how a designer/programmer team worked together depended on the designer or programmer. As I said, my personal interests was strongly in programming, so when I worked with a developer, I maybe tended to do more programming than some other designers, and there was no hard fast rule. I can't say there weren't conflicts, but I think all in all it worked well, it worked smoothly.

HAIGH: You had mentioned that choosing suitable test cases was something that the designer would do. What other things would the designer do?

AIRD: Well the designer would be responsible, for, say, a chapter and having adequate coverage of the topics that one would expect if one was going to buy a library. They would be responsible for the design, here's the subroutine, here's the routine, here's what it does, here's the arguments, here's kind of the typical error situation that can come up in terms of the user's problem –

HAIGH: So in that sense, they were defining the user interface?

AIRD: Yes.

HAIGH: So they would be responsible for testing and user interface design?

AIRD: Well they were responsible for gathering or figuring out what test cases needed to be run, yes.

HAIGH: And designing the user interface, were there any other main areas of responsibility for designers?

AIRD: Documentation.

HAIGH: They would actually write the documentation?

AIRD: Yes. Between the designer or programmer, they were both responsible for getting this done, but that was primarily a designer task.

HAIGH: So there wasn't a separate group of technical writers for the documentation?

AIRD: Well later on we did have some technical writers but, by and large, the documentation was done directly by the designer with help from the programmer, and under the guidelines of the documentation conventions.

HAIGH: Was that because it would be hard for writers without a strong mathematical background to understand it well enough to do the documentation?

AIRD: I would say that was part of it, yes. The designer and programmer were a team that supported this thing directly. The users called right in to the designer or the programmer, and if something came in for that routine then they were responsible for it. With the standards and guidelines for documentation, I think we were getting better documentation than we would get by outsourcing it to a technical writer.

HAIGH: Yes. So switching to the programmer half of the team, I imagine that what they would receive from the designer would be this specification for the interface, the error messages, and so on. They would receive, probably from some published source, the description of the algorithm itself, since you had said that those were mostly coming from things like *TOMS*.

AIRD: Yes, various sources. *TOMS, Numerische Mathematik*.

HAIGH: And so they would take this algorithmic description and the specifications for the routine and go away and write their FORTRAN code?

AIRD: Yes, I would say that would be one scenario. Another might be that the designer actually got part of EISPACK or LINPACK and said okay we're going to implement these, and I want this one implemented by this name, and this is the user interface and here's the running code. So anywhere in that extreme from the programmer getting an algorithm written out in some pseudo language to getting one in code, and then tuning that code to be FORTRAN, adhering to the portability constraints, the preprocessor, knowing that you had to port this to many different environments, knowing that you had to have test programs, you have to have, what we called, minimal tests, and exhaustive tests.

HAIGH: And would you yourself have designed or written some of the routines?

AIRD: Yes.

HAIGH: Did that change over time?

AIRD: Oh definitely. As time moved on, between 1973, and let's say 1987, by 1987 I wasn't doing any direct programming at all. I had a very large group to manage, my position was mainly managerial. While I missed the programming activity, there were at one time a couple hundred people reporting to me. There was the development group, and then there was the computer services group, and so it was strictly managerial at that point.

HAIGH: And did you enjoy managerial responsibilities?

AIRD: Yes, I did. I would say that I enjoy the actual process of writing programs and doing that work more than some of the managerial stuff. Management can be frustrating. Of course some of the coding stuff can be frustrating too, but I would say that if I had to pick number one I would say writing programs would be my first love. Managerial activities were important and I enjoyed them, and I especially enjoyed the success (of the product) that was coming with it. Now being a manager in a very unsuccessful organization I think maybe I couldn't stand that, but being a manager in a successful organization, where you're primarily hiring and you're giving raises, and you're moving to newer and better quarters, and you're getting better and better equipment, that's fairly enjoyable. You know there are certain unpleasantnesses that go along with it. Every once in a while as a manager you have an employee that isn't performing well or is doing something wrong, there's discipline and things like that that have to be done that aren't fun. But overall I would say it was very enjoyable in the late 1970's into the 80's, up to 1987 to the release of the revised library.

HAIGH: Do you think there's anything special about managing programmers?

AIRD: Well I always felt that the people that we had were very good and very motivated, and you didn't have to stand over them and crack a whip to get them to do their job. Once in a while you had to say, "look why don't you just take an afternoon off and take a break from it." So you had to provide them with the equipment and the environment, the tools needed to do their job, and then they would do it very well.

HAIGH: And where were the recruits coming from?

AIRD: During the height of the employment era at IMSL, we were recruiting all over the country. We were bringing people in for interviews all the time. I know at one time we had actually recruited many, many people from Michigan. I'm from Michigan originally as you know, and therefore I had lots of contacts, and the Detroit, Michigan area was falling on hard times and there were a lot of very good people looking for work, so we hired a lot of people from Michigan. It wasn't hundreds of people but there were ten or twelve that came out of Michigan.

HAIGH: And among the programmers as opposed to the designers, what kind of background or experience would you look for?

AIRD: Either a bachelor's or master's degree in math, stat, or computer science.

HAIGH: Would you expect them to have any particular kind of direct experience with writing mathematical routines, as opposed to other kinds of software?

AIRD: Well that would certainly be a plus, but I would say we weren't looking for that. When you were hiring some people directly out of school, they were getting their bachelor's degree or master's degree, they generally had the kind of experience that you would get in school, and so they had the fundamental knowledge. Then at IMSL we had lots of other people who were doing the same things and we had the standards. So it wasn't like just okay go document this, here's a bunch of subroutines, here's the manual, and here's the coding conventions. Most of them did a very good job.

HAIGH: So what proportion of the programmers do you think would have been hired straight out of school?

AIRD: That's a good question. I could guess that maybe fifty percent of them, there was a pretty high percentage of fresh graduates.

HAIGH: Okay. What I would suggest now is that we go back and talk a little bit more about the coding practices and the portability things that we've already discussed, and then probably leave the other issues for the second session. Would that work?

AIRD: Okay.

HAIGH: So you wrote in your article that in 1977 a Data General Eclipse was installed, and at that point IMSL had its own in-house interactive development system. What kind of a difference did that make to the way that people worked?

AIRD: Well I would say that was a very, very big breakthrough. I loved it, I thought it was just fantastic because you now were out of the punch card era, you had files on disk, you could sit there with an editor and create a program and run it, and run through this portability converter and things like that. It vastly improved the whole process. I remember that there was a point when IMSL had a whole room full of card cabinets and we kind of had a little joke going that when we moved things around that we could move the card cabinets, but they were always to be moved closer to the front door, and finally there was a day when they were moved out the front door. Acquisition of the Data General and this terminal driven computing, with editors and such, was clearly a big step in that direction.

HAIGH: And would that have been the first time at which people began using video terminals as opposed to teletype type systems?

AIRD: Well as I recall before that it was punch cards and then you were either driving punch cards someplace or reading them into a terminal, so it was definitely a terminal driven system. It started off just in one room with the terminal and then it ended up that people had terminals in their offices. Of course, we did suffer the general problem of centralized computing, that when we had one mini computer and we had ten people on terminals running them then we could, all of a sudden people come out of their office and say what's happening this thing has slowed down, but, I think, overall it was just a wonderful step for the company, a milestone in our computing capability.

HAIGH: And in the article you also wrote about the IMSL development system that was constructed to support the developers. Did the Data General come with the kinds of routines and compilers and utilities that you needed, or did a lot of that have to be developed in-house?

AIRD: I guess I'm going to have to review my own article.

HAIGH: Alright let's defer that question for the second session then.

AIRD: Okay.

HAIGH: We'll talk a little bit more later about that.

AIRD: Let me make a note here then. Development System.

HAIGH: And that's chronologically a little later, so we'll get to address it later. You've also referred during the interview to some of the coding practices and standards that the programmers were required to use. Had those been defined by the time that you joined the company?

AIRD: Yes. IMSL, as far as I know, always had coding practices. The source code had to be very, very clean. You weren't allowed just to have stuff all over. First it was done by hand, you indented and spaced things properly, you had certain headings at the front of the code. The on-line, or computer readable part of the documentation was with the subroutine and there were standards for how the heading had to be. If you looked at any IMSL manual, you could expect certain things. Therefore, a user becomes familiar with that and can use multiple routines with one learning curve. You're expected to have arguments ordered, input arguments, followed by output arguments. Sometimes in earlier versions there were work space arguments, and everything had an error parameter. So I think that was an important part of making the library easy to use and user friendly.

HAIGH: So those coding standards would have covered things like indentation, spacing, headings and comments, and the arguments that were provided to the function?

AIRD: Yes. All of the above.

HAIGH: Do you think that having such detailed coding standards was an unusual thing in the scientific software area at this point in the early 70's, or was it something that was widely understood?

AIRD: If I was working at the Purdue University computing center and I was developing a subroutine, and a colleague of mine was developing another subroutine, almost anything would go. We just had our own ideas for these things. I guess I thought that it gave a uniform appearance in nature to a collection like IMSL even though there were times when programmers said, “well I don’t really like this for my routine.” Well, let’s try to stick to the standards. If you looked at published algorithms they were kind of all over the place, but for a commercial product that was meant to serve scientists and engineers, in a broad area, it was a very good thing.

HAIGH: So, for example in your previous programming positions there hadn’t been any real controls over those kind of things, or source code indentation, or anything like that?

AIRD: Now certainly in the earlier employment like at Burroughs or NASA and stuff, there no coding standards. At Purdue there was a program called Clean, that would take your source code, your grungy source code, and do the indentation and make other changes that we were trying to apply. So that was in some sense part of it, but nothing as far as standards for including comments at the beginning that documented the code.

HAIGH: And I’m also aware that this period of the early 70’s was a time when structured programming became a buzz word, people started worrying about GOTO statements and elegance and readability in code. Did any ideas like that from outside the company have an influence on how you thought about what was good programming?

AIRD: Well I guess that you say in the early, early years we didn’t talk about that very much. Obviously a code that was written with a lot of GOTOs was hard to read and therefore it was frowned upon. Maybe by the programmer, or maybe if it was a programmer/designer team the designer had to read this code so there was some common sense about this stuff. But FORTRAN didn’t really have all the facilities to support structured programming at that time. You had a hard time not having some GOTOs and so when we talked about it I don’t think that had a big impact. I think the overall standards and the common sense approach of trying to make this code very readable and very workable were more of the driving force than any kind of structured programming approach.

HAIGH: Yes. Actually another idea that I know got some attention in those years was the idea of having a code walk through. Did you have anything like that where a programmer would have to talk people through and defend their coding decisions?

AIRD: Well there was always a review process. The coder would give the code, there was a designer/program routine, and then there was another designer/program routine, and I don’t remember exactly when that started, or whether it was always there. But certainly there was that concept of a review, and so another designer and another programmer would actually receive the code and documentation in its final form, or at least beta test form, and be asked to go through and look at it and give comments.

HAIGH: And in answering the question on elegance and structure, you said, “Well in the early years we didn’t think so much about that”. Was there a time later on at which it did become more of a concern?

AIRD: Well I think as FORTRAN progressed from FORTRAN 66 to the FORTRAN 77 era, the main emphasis was on ease of use. One of the big things on ease of use was not having to pass workspace to routines, being able to take advantage of dynamic storage allocation or some other allocation mechanism that we had. At one time there was a workspace and you provided this common block with work space… it was sort of like dynamic storage allocation but not exactly. So ease of use, arguments, storage allocation, error handling, I would say were more dominant in our discussions than these more theoretical structured programming arguments.

HAIGH: Alright so just to finish up with these topics. Now you had already talked about the IMSL FORTRAN converter, and what it was doing. I was wondering did, I know that there were projects at other places, including the TAMPR system from Argonne, originally used with LINPACK, and work at Bell Labs on portability and libraries. I was wondering did, how would you compare what you were doing with these projects, and were there any ideas that came in that you incorporated from these other attempts to produce more portable software?

AIRD: Well I can remember being at portability meetings and discussing these various systems and it seemed like the work that was done at IMSL was providing certain functionality, like the functionality of the conditional compilation. And then we had certain conventions with dollar signs and things so that certain code could actually compile even though it was called a basis deck, or basis file. At the time, if you didn’t run it through the converter you could actually compile it on this system because these other lines that were for the converter were comments. Without recalling specific details, it seemed to be that these other approaches provided similar facilities but in a slightly different manner and therefore we were sort of along our own way. Not to say it was any better, but similar functionality from a different approach. So we went our way and they went their way. I can’t cite a specific thing we gathered from them. It seemed like it was a fairly straightforward process to say “well, in portability here are the differences, these machine constants, these condition statements, and a few other things, that the converter would do for you.” And therefore we implemented that. I’m making it sound simple, it was, I believe, an eighteen month study, so there was a little more to it than that but I can condense it down to that process. It seemed right from the beginning that what had to be done was intuitive.

HAIGH: Right. And were there any particular problems or unexpected challenges that you faced in realizing this idea?

AIRD: Not that I recall. I could say there was the tedious task of gathering all the constants for a given environment if you were hitting a new platform, but I think there were no real challenging technical intellectual problems in this process.

HAIGH: Okay. One area of work that, it struck me might have been directly relevant, is interest in defining this portable subset of FORTRAN and verifying programs against it. I was wondering in the early days of the library, did you feel that you already feel that you already had a good handle on the best way of writing code in FORTRAN to make it more portable, or was that something that you developed more expertise with over a number of years?

AIRD: There was something called PFORT, developed a Bell Labs, that you could run against a code and it would tell you if your code deviated from the portable subset of FORTRAN. I'm sure that we used PFORT at times. But I think through the course of developing all this software over all these environments it was sort of in-house expertise that said "avoid this" and it was passed from programmer to programmer. I think we benefited by the PFORT work, but probably didn't revise anything because of it. And in the rare case where somebody wanted to use certain statements on a machine that weren't fully portable, they had this conditional compilation. You could always have IF IBM… IF CDC… IF anything else, that kind of thing. We obviously tried to minimize that, just from the readability of the code. I don't know if that answers your question or not.

HAIGH: Yes, what I understand you're saying is that by the time that the PFORT system appeared, IMSL had enough expertise in doing this that they didn't need it as much. Presumably for someone who didn't specialize in writing portable software it would have been much more useful.

AIRD: Yeah. In some sense the business forced everyone to try to write code in the most portable subset and we had probably more platforms that most other people in the world, and, therefore, had the knowledge of what worked and what didn't work.

HAIGH: Yes. Okay, well I think that would probably be a good point to wrap up the first session then.

AIRD: Okay.

[End of Tape 2, Side B] [Start of Tape 3, Side A]

Tom Aird interviewed by Thomas Haigh. This is the start of Session 2, which is taking place, again, at Dr. Aird's house in Reno, Nevada, or just outside it. It's the fourth of June 2004. We're continuing where we left off the last time.

So, I think at the end of last session we'd just begun to talk about the Eclipse minicomputer based development system that was produced within IMSL during the mid- to late-1970's.

AIRD: Yes. The development system was a big transition in the history of the development group at IMSL and it really ushered in the era of using file systems and mini computer technology to replace the punch card technology. The payoff was huge gains in productivity. In fact, the work that IMSL did in the late 70's, and through the 80's, probably couldn't have been done without this technology.

So a system was devised that consisted of, first of all, a file system with naming conventions, much like the naming conventions that one uses today with PC's to denote different types of files. The IMSL converter was a central piece of this, and the programmers and designers worked with source code that had statements in it that the converter read. Therefore, programmers could work with one file and through this technology could produce source code versions of subroutines, and test programs for the other compiler platforms that were supported. There were other pieces of the development system that are pretty common place right now: there was some analysis of codes, like a branch analysis that would tell you where your code was going and give you an idea of the inner relationships of the pieces of the code; there was a code to compare versions, often one wants to compare two subroutines for differences; obviously there were editors, you didn't just have to sit down at a key punch like the olden days and find the card and replace it, you had pretty good editors.

There was a PFORT verifier that one could run code through that would tell you if you were using any non-portable constructions. Most of the time, as I recall, we were attempting to program in portable FORTRAN, but not completely, because the converter allowed us to use non-portable statements if they were important. If it was a non-important feature then of course we stuck to the portable FORTRAN. Rounding out the tools there was a program that was called Polish that reformatted codes so that you didn't have to worry about indentations and such, and gave the programmers a productivity enhancement. Instead of them spending time moving statements to and fro, the Polish program would format them. And there were tools related to the IMSL coding conventions we used. If you put a subroutine through the review process, it would tell you whether you had deviated from any of the coding conventions, checking that you had a purpose statement and argument statement and all those pieces that we required in the code. On top of that then one could, from the minicomputer environment sitting at a terminal, actually produce test programs and such, and submit them to be run on any one of the platforms that was supported.

Some of the support was done out of house and some of it was done by remote job entry through, I guess, it was the Data 100 at that time. But from the programmer's point of view the, programmer had to just simply say "I'm submitting this job to be run on a Univac or CDC computer." It would go into a queue, and the queue might then at some point be moved to a magnetic tape and the tape would be mounted on the remote job entry, and the remote job entry would send that to the external test site, the program would be run, output would come back. It wasn't as fast a turnaround as it would be today if you're using a PC on your desk, but in those days it was fairly efficient and effective for getting the job done.

HAIGH: And were all of these software tools that you mentioned developed within IMSL, or did some pieces of the system come from Data General?

AIRD: Well some of the pieces were developed in-house, like the review process and the remote job entry, those were done through programming and through the Data General command language. I don't remember the author of Polish, but it was certainly not developed in-house. PFORT was developed at Bell Labs, the portable FORTRAN verifier. So we used a combination of in-house development and tools that were developed externally.

HAIGH: And do you have a sense of whether this kind of system would have been put together by other computer software firms during the period, or do you think that IMSL was ahead of the curve with this technology?

AIRD: Well I don’t know the specifics, but NAG was in the same business and was doing the same things, and I assume they had similar software. I don’t recall ever doing any kind of a comparison with them. There weren’t very many organizations that were supporting the multiple platforms that IMSL and NAG were supporting, so they probably didn’t have quite a need for it. Various projects would develop FORTRAN using the PFORT verifier rather than this tool set, but they would send out strictly portable FORTRAN, whereas, IMSL had a harder requirement to meet in getting these codes tested on so many platforms.

HAIGH: We already discussed the three platforms that the IMSL library originally supported in the early 70’s. Now your article in the Cowell edited volume, *Sources and Development of Mathematical Computer Software*, gives a list of additional platforms that were supported through 1980. So I wonder if you could talk a little bit about the process of bringing it onto extra platforms, particularly the transition that seems to start in the late 70’s towards versions for different mini computer platforms. And also perhaps talk about platforms that were added after 1980.

AIRD: Yes, during that era there were many, many different manufacturers, new ones coming up and old ones developing new computers. So there was a demand. Scientists and engineers that were used to using the IMSL library would acquire one of these new computers and therefore they would ask IMSL to support that platform. So there were certain financial considerations given to that process that, namely how many computers of that type were out there, did it actually have good compilers, was it feasible to port, and if we did the porting then would it be a profitable venture. Although I don’t think that was a hard and fast rule, I don’t remember turning down too many platforms, and the good news was that through the automation and the development system the incremental costs of supporting a new platform was only a small fraction of the cost of the original development. So it made it feasible to just continue to add platforms. There were various floating point modes, that is, in the 32 bit environments we had both single precision and double precision, and then there were environments that had 36 bit arithmetic, maybe one or more that had 48 bit, and Control Data had the 60 bit, so outside of the 32 bit it was pretty much the single precision library. The process was fairly straightforward of acquiring a test site, and then doing some preliminary work to run tests, first of all, to encode the characteristics of the new environment into the converter, that is, constants like machine precision and largest floating point numbers, and then let the converter produce a sampling of test programs and take them to the new environment. If it passed them, and passed the business test, the company would launch a porting (IMSL used the term mapping in placed of the more common porting term) of the library to that new environment. There were mainframe computers, like the IBM and the CDC, Univac, Honeywell, and then came along some of the minicomputers like Data General, and there was the DEC, the PDP series and the DEX VAX environment, there was Hewlett-Packard 3000, there was Prime computer, and so forth, and they just seemed to keep coming.

HAIGH: Were there any particular challenges involved in squeezing the library into a relatively small platform such as the Eclipse or the PDP?

AIRD: As long as they had a good compiler, not really. I can remember struggling with the first instance of the IBM PC in that regard, but when we were porting to minicomputers it seemed like as long as the compiler could be made to compile code correctly then the library would run it correctly. During the porting activity to these new environments, when an error occurred it was actually more likely that it was a compiler bug than a bug in the library product itself. We were, you could say, instrumental in finding and helping to fix bugs in FORTRAN compilers during that era.

HAIGH: Well tell me about porting to microcomputers then.

AIRD: Okay. I'll have to recall the exact year that this happened, but it was probably in the mid 1980's, and the first IBM PC computer came along and it was sadly lacking in compiler technology and space and things. I'm trying to recall the exact nature, but it was just a bear of a problem to get code running, and it took a long time before that happened, and it seemed to me it was perhaps not until later revisions of the PC, it had more memory and better compilers, that we actually got that running.

HAIGH: I would imagine that the memory segmentation might have been a problem, you could only address one 64k chunk at the time I believe.

AIRD: Yes, there were very serious size limitations on the problem you could run, and it seemed at first that it really was not a good platform for the IMSL library.

HAIGH: But a successful PC version was eventually produced?

AIRD: It was eventually produced, and it was a fairly successful product offering. I know that there were more considerations then for the pricing of it, because it was a fairly expensive product and marketing and sales had to really work to set the right price, and to get sales going. The price, rather than the need for the product, I would say was the main challenge at that point. It was successful and, I'm trying to remember the year, maybe in the late 1980's, there was probably a million dollars in sales on the PC platform.

HAIGH: I notice in your article in the Cowell volume that by that point it had already been established that the versions for the minicomputers would cost less than the mainframe versions. Did a similar thing happen so that the PC versions that cost substantially less than the minicomputer version?

AIRD: It did. The price was set at $2000 for a paid-up license which is how most PC libraries were sold. It was very price sensitive.

HAIGH: Did that policy ever produce any friction with customers that you recall?

AIRD: From what I recall, initially the prices were not lowered very much and that did produce a lot of friction because the personal computer didn't have anywhere near the capacity of the mainframe, so the mini's pricing had to be worked out.

HAIGH: Sure. And did it remain the case that the software was licensed rather than purchased?

AIRD: In the early years, the library was only available through a lease. Later, there was a paid-up license for every platform class and there were 5 classes. Class I was the super computer class with a paid-up license costing $30,000 in 1990. The PC was Class V with a paid-up license costing $2000.

HAIGH: The other question would be: did it remain that it was on a per-machine basis rather than on a site license or a number of users?

AIRD: Yes, I think that was the whole issue of pricing on the PC. A PC was generally considered to be for one user, therefore customers wanted a much lower price. As I recall, some customers negotiated for site license, but the vast majority of PC libraries were sold via a paid-up license.

HAIGH: And were any attempts made to produce versions of the library for the newer generation of supercomputers with multi processors and vector capabilities, such as the famous Cray machines?

AIRD: Yes, there was a successful mapping to Cray and a couple of other vector machines. I think the issue probably was that the library did not necessarily fully take advantage of all of the hardware features. On the other hand it did provide the functionality and did run very fast when the compiler was able to automatically use those features.

HAIGH: Yes. And from the user point of view, would they essentially call the library in exactly the same way, with the same parameters, and use the same techniques whether it was running on a mini computer or a mainframe or a super computer?

AIRD: Yes. That was certainly a design objective, and we achieved that across this whole spectrum of environments. One could have tests, or application programs imported back and forth using the IMSL library, and at least the IMSL part did not need to be changed.

HAIGH: As to these new kinds of machines, which opened up scientific computing to new kinds of users, and that people who worked in new ways, did that change anything in your relationship with the users? Dealing with individuals instead of dealing with relatively small number of mainframe computer centers?

AIRD: Well the user base for the IMSL library was always scientists, engineers, and what we called technical business people who were more or less doing there own programming. So they were FORTRAN programmers, either by choice or by necessity to get their own work done. Even though scientists come in different flavors, and engineers come in different flavors, they all had the goal of solving one of their own problems by developing an application, and that application had pieces that involved the solution of the mathematical problems that the IMSL library provided. So in some sense it was standard across the user base.

Now I think as the user base grew we probably moved towards people who were less and less sophisticated with mathematical software and therefore wanted more and more examples. Our examples in the manual were always just straight forward, here set up a matrix, set up a vector, call a linear equation solver, and you're done. That wasn't the way the engineer or scientist was working, it was some piece of the problem and then they wanted to see more how to use the library to solve their particular problem, whether it be automotive design or analysis of some process. We put effort into that but, it proved to be a difficult task. In the long run, we did develop some application examples but I don't think we ever succeeded in solving that problem while I was at IMSL. On the other hand, the set of examples was sufficient to make it possible for most of those users to use the product.

HAIGH: And it's also occurring to me that when you move into the microcomputer area it would be more likely that the end user themselves would be ordering the product, compared to the earlier era where it would be the computer center manager that you would sell to.

AIRD: Yes. Since I was in the development group I didn't get too directly involved, but the sales people had to, at each stage in this, find the particular person responsible for placing an order. In the early days it was always the head of the computing center, so it was pretty obvious you went someplace and you sold to an organization. Whereas later, when the computers got smaller and smaller, you had to find a group, or even one scientist, and so it was a tougher sales task.

HAIGH: Now I think you'd said a little bit before about the methods that were used to test the programs, that you would have a suite of tests that they would be compiled on different platforms to make sure that you got the right answers. I was wondering if user sites were involved with testing versions of the software and getting feedback on problems?

AIRD: There were two levels of tests: the exhaustive tests that we ran in-house to make sure that the algorithm and everything else was sound, and then we had what we called a minimal test. The minimal test was at least the example that was in the manual, maybe there were other small examples. Those were set up as a complete package so that if a customer was installing the library on their own minicomputer then as part of this process, once they installed the files, they could actually run the minimal tests. The minimal tests would produce a summary of either pass or fail messages, so it was very easy. One didn't have to look at any numbers and say is this the right answer or the wrong answer. If something failed then there was either some problem with the IMSL code or there was some problem with the setup, or maybe there was some problem with the particular compiler. More often than not those problems on the individual environments turned out to be problems with compilers. So if the user encountered a problem like that maybe it was an upgrade in the compiler and then once everything passed the user had some assurance that the library was properly installed and running on their environment.

HAIGH: So you had mentioned that the IMSL technical support must have spent quite a lot of time dealing with those kinds of compiler quirks and configuration issues.

AIRD: I would say it was a substantial part of the porting process. If you found an error in your program that you could quickly fix it was one thing, but if you found a particular statement wasn't working then you had to go to the manufacturer. Or you had to code around it in some way. It wasn't overwhelming, but it was a common occurrence.

HAIGH: And was there a beta testing program so that users could try out new versions of the library with their own problems?

AIRD: I don't think we had a beta test program as one knows it today. If we were working on a bug for a particular user then we would send that user the code and try it out, but if we were trying new versions we pretty much did the testing ourselves in-house, and when we were satisfied that it was ready then we simply sent it to customers.

HAIGH: And how was the software distributed to users?

AIRD: Mainly on magnetic tapes as I recall.

HAIGH: Was there a capability within IMSL to duplicate tapes or was that outsourced somewhere?

AIRD: IMSL had tape duplicating equipment in-house. All of that work was done in-house.

HAIGH: Now how about attempts to get feedback from users. You've already talked about things in general terms. I think you mentioned in the article that there was a quarterly user news publication, and in passing you'd also mentioned some user group meetings. So I wonder if you could talk some more about those topics.

AIRD: I'm trying to remember the year, maybe 1984 or something like that, there was actually an IMSL user group formed. Prior to that, probably more often than not, people from the development group would attend other user group meetings. IBM had their SHARE meeting, Digital Equipment had a user group meeting. Any computer manufacturer that had a user group meeting, IMSL would try to participate in that and set up a session to talk to users, so that was small scale. But then during the 80's an actual IMSL user group was formed. I would say there was much more dialogue between the active users and the development group because the programmer/developer teams were always the ones that were contacted. Anyone that was willing to pay the price of a long distance call, or send a letter (much of this was before e-mail) could have direct contact with the designer or the programmer. A great deal of emphasis was put on that kind of relationship with the customers, to ask them what they liked, what they didn't like, and get feedback to either correct a bug or to improve the quality or readability of the code.

Most often those user interactions involved documentation or, perhaps once in a while, a bug. It was rarely a user telling us about a new algorithm.

HAIGH: Yes. In fact you'd mentioned that you yourself had been a user of the library before you joined IMSL. Did many of the staff that you recruited have previous experience working with the library?

AIRD: I don't recall that as an issue during recruiting. As time moved on, obviously in the early 70's, there were very few people who knew about the library, in the later 80's it may have happened, but it wasn't a usual occurrence.

HAIGH: Do you have anything to say about the user news newsletter? What kinds of things would have been in it? Anything you remember about how successful it was?

AIRD: Well it was a means for the company to tell the users directly what new things were planned for a new version, or what bugs were planned to be fixed, so it was a meaningful communication method.

HAIGH: We spoke a little bit already about the changing level of sophistication of some of the end users. I wonder were there any other changes apparent from the 70's through the 80's, to the early 90's, in terms of the kinds of application area in which the library was being used?

AIRD: Well the library was always this generic mathematical capability that could be applied to the various areas. I think that if you look through the evolution, when we started off it was a linear equation solver for full matrices, and then there was a linear equation solver for positive definite matrices, and then there was a linear equation solver for special symmetric matrices, and things like that. So those things I'm sure came about because of the nature of the problems that people were solving and the fact that they needed more efficient solutions to the specific problems. So I would say that was how the evolving set of applications resulted in changes in the IMSL library.

HAIGH: So because the mathematics was so decoupled from the kinds of real world domains that the problems were being solved, in you're not aware of any significant shifts in the kinds of organizations that were using the library at the time?

AIRD: No. I would say that the library always remained generic mathematics applicable to any scientific, any engineering, any technical business problem.

HAIGH: Actually a question I realize I didn't ask a minute ago. I know that the IEEE floating point standards were an attempt to standardize hardware capabilities, and so eliminate a lot of these portability issues that you'd been facing in terms of precision and differences in the architecture. Did those have a significant practical effect during the time that you were with IMSL?

AIRD: We had a strong interest in that development, and of course had a desire not to have this proliferation of multiple floating point units, and all that came with it as far as trying to decide whether single precision was good enough, or double precision was good enough, or things like that. It was a wonderful thing when the IEEE standard came along and since then the reduction of the number of machines, and the number of environments. But the library product that was developed in this time frame would have worked just equally well in the IEEE floating point environment without any changes.

HAIGH: Were the standards rapidly adopted by hardware manufacturers?

AIRD: My recollection is that the hardware manufacturers that didn't adapt sort of faded away, and as you can see most of the ones on the list are no longer around, and the ones that are around have adapted. Now that happened over a period of many years, but nonetheless it happened, and so there was sort of a peak in the number of environments that IMSL supported, and once the IEEE standard took over, the number of environments dropped, and FORTRAN became more and more standardized (with FORTRAN 77 and later versions of FORTRAN) the portability issues, in some sense, faded from the forefront.

HAIGH: Yeah. Well let's follow up on the FORTRAN 77 question then. Yesterday you had spoken about the major new release of the library in the mid 1980's, and I think you had mentioned that being rewritten in FORTRAN 77 was one of its main advantages. So can you talk for a moment about what the practical advantages of the new version of FORTRAN were for mathematical software?

AIRD: Well I say there were two main issues with what we would call the early versions of the library. One was that those users always had to provide workspace, it was just another programming detail that the user had to worry about. At the same time there was an issue with error handling, as routines became more and more sophisticated, trying very hard to get back a meaningful message to the user through some error handling system. Having gone through from 1970 to the new release in 1987, all of the collective knowledge that we had was put into the redesign, using FORTRAN 77, using all of the technology that existed at that time, using everything that we knew about how the users wanted error handling, and using everything that we knew about how the users did not want to provide workspace. Those fundamental ideas were collected and put together, giving maybe the biggest project that IMSL had ever undertaken to produce this library.

As the library grew from 200 routines to, I think, more than 800 routines, there was a feeling that it was too big. Therefore, in that this rewrite, the library was split into a math library and into a stat library. So documentation was revised to give users better description of the algorithms and the highest level routine that the user called did not require any workspace. This was snot automatic or dynamic allocation because FORTRAN 77 did not have that facility, but it was handled through a common block that the user could set up. And then there was a second level routine that actually was, in some sense, for fall back so that if all else failed the user could provide work space directly, but if the user wanted this more easy-to-use version then the user set up a common block and set a certain size and then all the routines in some sense shared that. This was not a multi-tasking library, it was a single tasking activity, so that worked, and I think that was fairly successful because that's pretty much the way the product is today.

HAIGH: And were there any changes in FORTRAN itself with the 77 revision that made those changes in the library easier to produce or was the change of language largely independent from those improvements?

AIRD: FORTRAN 77 was a more fully developed language and therefore made it easier for the programmers to write the code and made the code more readable. But from the user point of view, I suppose that similar things could have been added along the way, even in FORTRAN 66.

HAIGH: Now I imagine that as the company grew larger and the number of people working there and reporting direct to you increased, then there must have been some further developments in the way that the work was organized. In your article in the Cowell volume you discuss the breakdown of the company. At the point you wrote it, there were a number of groups including the research and design group, the development, the product systems group, and the special products group. I believe, you mentioned in yesterday's interview that you became personally responsible for the computer services group as well as the development area.

AIRD: Yes, well as the company grew and the task grew, of course, we hired more and more people and they had to be properly organized. What you've just described then is the way things sort of seemed to fit together. I think it was a successful organization at the time. We still stuck with the designer/programmer team, with the designer being primarily responsible for algorithm selection and documentation, and programmer being responsible for programming. Many of the issues with various environments were handled by what we called the external data processing people, and of course there was an internal data processing group that provided the computing services and stuff. So instead the initial stage of the when company everybody was doing everything, you had a more specialized task with a narrower range and perhaps less complication. And then we had specialists who were out there learning about all the different mappings, we had the designers worrying about the higher level algorithms and user documentation, and the external data processing people worrying about all these environments that we were reporting to and the difficulties finding test sites and all that kind of stuff. So I think the organization fit the needs of the company at that time.

HAIGH: And what did the computer services group do?

AIRD: Well, at that time we were running a Data General system with terminals to each one of the offices. In later years we switched to Sun systems, so the data processing people were really in charge of acquiring that hardware and setting it up, and making sure that it worked. Of course, if you have a mini computer and you have a couple of people using it its response is very good, but as your group grows and grows, and you have ten people using it then it slows down, so you have to order a bigger or faster machine. So the data processing folks were really responsible for making sure that the development team had computing facilities adequate that could do the job.

HAIGH: So "computer services" is another name for the data processing team that you'd already mentioned.

AIRD: Yes.

HAIGH: Okay. And the special products group?

AIRD: We were kind of thinking about how would IMSL get out of this one product family and, therefore, we set up this special products group that had some freedom. It was a small group, as I recall, and could explore various ideas and things, so perhaps an outshoot of that was the PROTRAN product that we talked about, an investigation of C products, and such. So would you like to go into detail on some of those specifics now, or?

HAIGH: Sure, let's move to the other products.

[End of Tape 3, Side A] [Start of Tape 3, Side B]

AIRD: So as time marched on IMSL was really a one product company, becoming more and more successful. Things were happening in the world of scientific computing, it was probably more apparent on the statistical side that users really did not want to be FORTRAN programmers, and if they didn't have to be FORTRAN programmers to get their job done then they chose that route. SPSS was fairly successful in those days, and SAS, which I think ultimately became even more successful. Those were referred to at that time as fourth generation languages, that is, the user could specify a task to be performed more or less at a higher level without getting into nitty gritty details like allocating storage and do loops and things of that nature.

So IMSL felt that this was an important area, at the same time our advisor, John Rice, had been thinking about ideas in this regard and came up with some specific design specifications for a language that became known as PROTRAN. PROTRAN was designed initially as a FORTRAN preprocessor, but if you looked at it just as a language it was a language in the spirit of fourth generation languages. MATLAB was a mathematical language that was available, starting to become more and more popular, and so PROTRAN, at a language level, had constructions of the same nature as MATLAB. So one could just simply declare a matrix and write out the elements, one could say I want to solve A times X equals B, no programming details or things like that.

HAIGH: And then the preprocessor would expand that into blocks of FORTRAN which would then be compiled by the FORTRAN compiler?

AIRD: Yes. PROTRAN in fact was implemented as a preprocessor. Initially PROTRAN was not a full rich language. We designed it initially as a preprocessor, so that a user who was writing applications could take advantage of its facilities to make usage of the IMSL library, which was the ultimate goal, much easier than it would have been through writing calls to the subroutines. John Rice wrote a book, "Numerical Methods, Software and Analysis", that was published in 1983 that really defined the PROTRAN language, and there was a lot of discussion about where computing was going. [J.R. Rice. Numerical Methods, Software and Analysis. McGraw-Hill, (1983), 483 pages; IMSL-reference edition, (1983), 661 pages].

In fact, there's a quote in the manual that I think was very insightful where John Rice writes, "The PROTRAN system is an extension of FORTRAN which brings many of the IMSL library programs into the language in a simple natural way. It provides much shorter programs for various examples, and two, it is an example of the higher level languages that will spread throughout numerical and scientific computation in the coming years". So it seemed compelling to people who thought about these things that that was a natural progression of how programming would be done, and in order for IMSL to be in that environment it spent its time and resources in developing PROTRAN as a product.

HAIGH: Had any thought be given to developing more interactive systems, such as MATLAB?

AIRD: PROTRAN was initially received with mild enthusiasm, I would say, and the big issue was that it was not a complete language and it was not interactive. Going through a compiler you could generate statements and then you could have compiler errors and things. That problem could have been solved and the design group, the special products group was working on that. However in the short life of PROTRAN, one of the things that happened is the company was so successful at selling libraries that it was very, very difficult to convince the marketing and sales people that they should in any way sacrifice to sell the PROTRAN product. By that I mean a sales person selling libraries could easily sell a half a million dollars worth in a year, or you could have a million in revenue a year, some good sales people even more than that. A good PROTRAN sales person would be lucky if they had a few thousand dollars a year. Nonetheless, if you looked into the future…. This was, I believe, a standard business scenario where you have to initially have some vision to invest, and you have to have some persistence to push through the problems. So IMSL probably was a victim in that regard of its own success. PROTRAN was never a successful product and was not sold after 1989.

HAIGH: Can you remember when it was introduced?

AIRD: It was introduced in the early 80's, before the book came out, so it might have been 1981 or 1982, probably 1982.

HAIGH: And this was a stand alone product, or did it require purchase of the main library?

AIRD: It did require the library, in fact it was a preprocessor for the library, so one had to pay extra to get PROTRAN, and one had to have the library in order to make PROTRAN work.

HAIGH: Oh. So it would be targeted to companies that already had the library?

AIRD: Yes. I think that the people who were interested were always library customers and wanted ease of use.

HAIGH: And from what you said earlier it sounded that you considered a lack of internal enthusiasm for promoting the product to be the primary reason it wasn't more successful. Is that true?

AIRD: You could have a debate on this whether it was a good product or a bad product. I think hindsight tells us that we were heading in the right direction and if the company…. Look at MathWorks. MathWorks started with MATLAB, and MATLAB was not always as successful as it is today, it struggled. MathWorks had one product that was either do or die, that they didn't have anything to fall back on, so they didn't have any choice to make. IMSL had a choice to make, but the library sales, in this period, were just growing enormously. The company, I believe, in the late 1980's reached revenues as high as eighteen million dollars, and PROTRAN was not any part of that. With a reasonable amount of sales work, a reasonable sales person could generate a half million, sometimes even a million dollars, in revenue. PROTRAN was a struggle and so really the issue was could we, in development make it interactive? Yes, we had worked out plans to do that but I would say it was the financial people within marketing and sales that who were not very enthusiastic about it. Then it was a company decision where there's no point in investing in the development if the sales and marketing group is not going to sell it. So it had a short life. It was a product that was in the right direction, though whether or not it would have ever become a big success is anyone's guess.

HAIGH: Now you've already spoken about SSP as a competitor in the early period, and MATLAB emerging as a competitor in the 80's. Now it occurs to me that NAG, the British numerical analysis group, must have become a significant competitor at some point in the library business.

AIRD: Well NAG was a competitor, but as far as I could tell IMSL was growing about as fast as it could grow through the 80's, so it wasn't hurting IMSL's growth. We certainly ran into NAG when we tried to sell the IMSL library in the United Kingdom, but we even had some success there. IMSL had success in various parts of Europe, Germany and France, where we set up offices, so from the historical perspective I would say NAG was actually growing in size but IMSL was growing in size faster. At least in the United States, NAG was not really seriously cutting into what IMSL could expect in revenue.

HAIGH: So at least on the development side you weren't particularly worried by any library competitors, or feeling that there were any changes that you might need to make to the product to address any challenges?

AIRD: No we could always look at their product, and we always liked what we had done ourselves more than we liked NAG. We always felt that they had a good quality product and we had a good quality product and, therefore, to some extent it would come down to sales and marketing power. I think perhaps IMSL won the sales and marketing war more than it won the development war.

HAIGH: Were there any strengths or weaknesses that the two libraries, compared to each other, that stand out in your memory?

AIRD: Well I think there weren't many. Maybe in the later 80's, NAG had done a better job in nonlinear optimization than IMSL had, probably that was the number one area. So we put more emphasis into that, I don't think we ever really caught NAG in nonlinear optimization. But in the overall library it was very comparable.

HAIGH: Were there any things you think that IMSL was much better for?

AIRD: Well I think we had better user interface and better documentation. I think it was easier to use, but that's a subjective matter, and NAG was a good product and IMSL was a good product.

HAIGH: So through the 1980's, roughly what proportion of the worldwide market for mathematical software libraries do you think that IMSL would have had?

AIRD: Well I don't have a good recollection of NAG's revenue, I don't remember, but roughly speaking I would say that IMSL was probably at least twice the revenue of NAG during that period, up to 1990.

HAIGH: And on the mathematical side you don't recall any significant commercial competitors other than NAG?

AIRD: On the mathematical side?

HAIGH: Yes, as opposed to the statistical side.

AIRD: Well, again, you have to look at the specific time period. I think MATLAB was clearly becoming very successful in the late 1980's. It was very clear that the scientists and engineers were moving more and more to MATLAB. Mathematica also came out at that time, but I don't think that it had as big a market share as MATLAB. But I would say that MATLAB was definitely the new guy on the block. It was the fourth generation language, it was what scientists and engineers wanted, and the same time in the business world users were migrating more and more to spreadsheets. And so it seems to me, now looking back from 2004, that in fact that migration did occur, it occurred over maybe a period of fifteen or twenty years, but the scientists and engineers who wanted to write their own algorithms migrated to MATLAB. There are other languages like IDL, but MATLAB seems to be the dominant force in that side, and today it's Microsoft Excel as the spreadsheet. These programs in Excel have enormous capable.

HAIGH: But those people weren't just migrating away from IMSL, they were migrating away from the whole idea of a library attached to a standard high level language.

AIRD: Yes, they were migrating away from a third generation language, namely FORTRAN, or, as we might mention later on, C, where they had to write in this third generation language and compile and worry about all the nitty gritty details of writing programs. The higher level made it much easier for the user to express the problem that he or she wanted to solve.

HAIGH: Right. But within the third generation library business for mathematical software, after SSP faded, NAG was the only significant competitor?

AIRD: Yes.

HAIGH: Let's talk about the C version of the library then.

AIRD: Okay. This was a similar process. We talked about what was happening in the computing world and at the same time one was noticing if the users that were programming in FORTRAN wanted things to be easier and easier to use and, therefore, there was this migration to fourth generation languages.

There were also a group of people that developing specific applications for the scientists and engineers. So instead of the scientists and engineers writing their own program there were either companies dedicated to writing special applications or there were companies, or groups of programmers within an organization, who supported the scientists and engineers. Those individuals were migrating more and more to the C language, as Unix became a dominant force in operating systems, C became more and more of a dominant language over FORTRAN. C always had dynamic storage allocation, which was a big thing, and once programmers got into using the C language they really wanted to use C. It was difficult or impossible to call FORTRAN subroutines from C and, therefore, IMSL put effort into developing a C math library. I'm going to have to look up the date of the release of these but the manual I have is dated 1992, so we came out with the C math library and a C stat library, and those were subsets of the FORTRAN library.

HAIGH: So the full library wasn't ported to C?

AIRD: No, and I don't believe it ever has been.

HAIGH: Other than the dynamic storage allocation you mentioned, what were the strengths and weaknesses of C as a language for scientific application development compared to FORTRAN?

AIRD: Well I think perhaps the main issue was this storage allocation issue. Whether you like C or not is subjective and it depends on your background. Systems programmers were doing things with C, they knew C. C is, I would say, a much more cryptic language than FORTRAN, so if just an ordinary scientist or engineer were to look at it they might prefer FORTRAN. A systems programmer looks at it they prefer C just because it compiles fast, it has the dynamic storage allocation, they can integrate it with other things that they're doing. So it was really that issue of language preference. It was a market segment that said we're not going to use FORTRAN, so if you don't have a C product we're not going to buy your library. If you have a C product then we'll use it, and buy it, and use it to develop the application.

HAIGH: And did C have the same kinds of issues as FORTRAN in terms of standardization of compilers?

AIRD: C was much, much better in that the compilers were pretty standard and much better than the FORTRAN compilers. In the big picture, I would say the FORTRAN compilers were always fairly buggy, and the C compilers were always very much more reliable.

HAIGH: So did that make portability easier to achieve with the C version?

AIRD: Definitely, it did. The portability issues just went away.

HAIGH: Now there were a couple of other products that you mentioned on the resume that you sent me. One of those was FORTRAN Exponent Graphics. What was that?

AIRD: During the development and growth of IMSL, from time to time users, in addition to asking for more mathematical subroutines and more statistical subroutines, asked for graphic capabilities. We were growing very fast, lots of new business, lots of customers asking for this, so we had to make a business decision: should we either acquire a package outside or should we develop our own? Based on the information available and the business model, we decided that it would be better for IMSL to have its own package. So we hired a graphics software person, Mike West was his name, and he joined IMSL in about 1986, and perhaps in 1987 we announced Exponent Graphics.

HAIGH: What did it do?

AIRD: It did the kind of plots and charting applications that would fit in with the IMSL library. So obviously if you were doing some kind of interpolation or approximation you could plot your function, you could plot your data. It had other things like bar charts, but mainly it was geared to support the applications that the scientists and engineers were doing that were using the IMSL library.

HAIGH: Was graphical output a hard thing to accomplish from FORTRAN?

AIRD: Exponent Graphics was written in C.

HAIGH: But could it be called from the FORTRAN version of the library or was it purely for the C version of the library?

AIRD: Yes, there was a FORTRAN interface to the C product.

HAIGH: Actually what you gave me it mentioned two products. FORTRAN Exponent Graphics which presumably was FORTRAN –

AIRD: Yes, so the FORTRAN Exponent Graphics was the interface for the C product, it was impossible to develop all of the data structures and everything, and the designer, Mike West, had done all this stuff in C, so Exponent Graphics itself was written in C, and then the interface was developed for FORTRAN.

HAIGH: I imagine that adding graphical capabilities must make it less portable than the library itself?

AIRD: Well since the underlying code was developed in C it was actually easier portability. The difficulty was really getting a product developed, but anything that was written in C really did not have the portability issues of FORTRAN. By the late 1980's many of those portability issues were waning. I don't remember the exact set of environments for which we were supporting Exponent Graphics but it certainly was a small subset, it was probably Sun work stations and things like that that had strong Unix environments and C compilers.

HAIGH: Graphical interfaces seemed to vary a lot between systems in those days, so I imagine that would have made portability harder. I suppose in that days X windows would have emerged, or be about to emerge, as the standard on Unix systems?

AIRD: I'm a little bit rusty on those details.

HAIGH: Sure. Okay, so you mentioned PROTRAN, you've mentioned the C version of the library and you've mentioned the Exponent Graphics systems. Were there any other products produced by IMSL?

AIRD: There was a special functions package that was developed. Wayne Fullerton joined the company and did a very extensive job of developing lots of special functions, and that was a FORTRAN product.

HAIGH: And that was the only other product?

AIRD: That was the only other one.

HAIGH: So the business stayed really quite focused on the library itself and a couple of essentially add-on products?

AIRD: Yes. If you're going through the 1980's where IMSL was growing by leaps and bounds, really the revenue was generated by FORTRAN subroutine library sales. These other things were attempts to get IMSL out of this one product company status, but in fact the libraries were totally dominant.

HAIGH: Now yesterday you'd talked about the role of Battiste and Johnson as founders, and someone on the statistics side, whose name I can't remember (Walt Gregory). Now through the late 70's and 80's were there any other hires that came to play important roles within the company?

AIRD: Well Jim Gentle was really the head statistician at IMSL for many years in the 1980's, into maybe early 1990's. But are you talking now about the business side or the development side of the business?

HAIGH: Well I was just thinking about the human side. As the company grew a lot more people were coming on board, were there any important changes in responsibility, were there any colleagues who joined after 1973 who played important roles, were there any people within your part of the organization that you worked particularly close with?

AIRD: Well the development team grew steadily through the 70's, but the big growth came in the 80's when there was just very high demand for the IMSL FORTRAN libraries and sales were just booming, and every year was at least ten percent, maybe twenty percent, more revenue. So the company was still enthusiastic, but there was more work to be done and so several people were hired, I mentioned Jim Gentle as heading up the statistics group during that time. There were other people like Richard Hanson that joined to work on the mathematics side, and –

HAIGH: So what were his responsibilities?

AIRD: Richard joined in the 80's, and he was an expert in vector and parallel computing and as time moved on he was learning more and more about FORTRAN 90. In, I believe, 1991 we released the FORTRAN 90 MP library. That was a subset of the math library aimed at taking advantage of the FORTRAN 90 structure and perhaps being more efficient on multi-processing systems,.

HAIGH: Was that product successful?

AIRD: Not that I know of. I left IMSL in 1993, and even in 1993 the only thing that was successful was the FORTRAN library.

HAIGH: Anybody else?

AIRD: Well I think we should talk a little bit about the change that came abut in 1990, when the top management of the company was changed.

HAIGH: And was that the first significant change in top management in the company's history?

AIRD: Well it was the second, the first change was when Ed Battiste stepped down as president, which was 1976, and Walt Gregory became president. Walt Gregory was president from 1976 on to 1999, and during Walt's presidency the company was growing steadily - with growing sales and profits. One thing that happens when you're in a small company at a certain point you start to attract the attention of venture capital people and investment bankers. This happened to IMSL, maybe as early as the mid 1980's when revenue reached about twelve, thirteen million. There were a period of a few years where a few specific investment banking people were talking to the Chuck Johnson and saying, "Now you have a wonderful company and you're doing very well and I know you're growing at twenty percent a year, but you could do much better at this stage you could really turn this into a fifty million dollar company. But in order to do that you have to make changes in the top management, because your top management has to be a team that's experienced and knows the venture capital market and is willing to work with the venture capital people so they can make investments and then you can purchase other companies and grow by leaps and bounds." So by about 1990 the owners of IMSL, the Johnson family, decided to make that change.

HAIGH: And they owned all of the equity in the company?

AIRD: Yes, it was privately owned. Employees had stock and some stock options, but the Johnson family controlled about ninety or ninety-five percent of the stock, so they were clearly in control. And so their decision was to go ahead based on advice of various investment banking companies, I think one was a guy named Gary Hromadko from Merrill Pickard. They were convincing in the sense they said you really need to make these changes in management. However, the Johnson's decided to make the change in two steps. They hired Richard Couch, from Diablo Management, a California company that was kind of a crisis management company, to come in take over as president, with the expectation of being temporary. They also had a plan to do a search through the country, find just the right top level management team that could handle this fast growth and things, and bring that them in and make a transition. Several candidates were interviewed but no one was deemed to be suitable, and so Richard Couch and his helpers from Diablo Management Company, with other management consultants, continued to run the company.

At that point in 1990, although IMSL was a one product company, it was growing by leaps and bounds. You have to understand that the, internally the company was really focused on the product and doing everything to develop the best product, and serving the users and understanding the scientific community. Richard Couch and the Diablo management team really didn't have any experience at this mathematical software area and didn't have a full appreciation for what the people who developed the product, or for the people who used the product. Nonetheless, they made some substantial efforts aimed at taking IMSL from this point, which might have been nineteen million in revenue, to becoming a fifty million dollar company and then going public. That was a somewhat painful process because there was definitely a de-emphasis on the product and the product development. There were layoffs of people in order to move expenditures from product development into management and sales and marketing efforts during that time,. It was discouraging, you could say, after you spent years building this wonderful development group to see it disappear slowly.

They had a plan to acquire other products and go forward through a series of business efforts. They acquired Precision Visuals, a Boulder, Colorado company, Precision Visuals had a product named PV Wave that was based on IDL, which was a product from Research Systems also located in Boulder. Precision Visuals had full rights and this product which was also a fourth generation language, although Precision Visuals was not a mathematical or statistical orientated language, it was primarily a signal processing product. The idea was to merge the IMSL library product with the Precision Visuals product, and in some sense go on into this fourth generation language product.

HAIGH: So the idea was that this other product would serve in part as an interactive front end for the existing library call routines?

AIRD: Yes. It would in some sense be the competitor to MATLAB. Although I don't know that that was specifically targeted at that time, if you look in hindsight it would tell you yes. So now you would have this interactive fourth generation language.

Precision Visuals' revenue, as I recall, was about sixteen million dollars, in 1991 when this happened. IMSL's was maybe nineteen million so the combined revenue of the company was about thirty-five million. So this was a big step in revenue, but as often I guess is the case when you try to merge two companies, you have culture shock. There was a product produced that did integrate many of the IMSL mathematical and statistical routines into PV-Wave, but PV-Wave had this market of signal processing users and IMSL had this market of scientists and engineers that were using FORTRAN. Although IMSL was successful in integrating the products, I don't think it was successful in the market place.

I left in 1993. There were a couple of layoffs during this time, so that the development group was really being seriously downsized and there was very little emphasis on developing any new mathematical or statistical subroutines or any activity on the library. Fortunately the release of the library that was made in 1987 was in pretty good shape and would stand the test of time. I believe their sales of the combined group continued to decline, and all I can say is that ten years later they're still selling the same product but with much less revenue, probably in the neighborhood of ten million dollars a year. From my observation very little new work being done on any new subroutines or any new development in that regard.

HAIGH: So do you believe that the NAG library would now have assumed a dominant position in what's left of the library business?

AIRD: Well I think the NAG library is perhaps suffering the same fate, in that fewer and fewer people are using FORTRAN subroutines because fewer and fewer people are programming in FORTRAN. I'm even less aware of what NAG is doing as far as a competitor to a product like MATLAB. As far as I can see MATLAB is really dominant in scientific and engineering computing. Other than the remaining few programmers who are programming in FORTRAN, it would be very difficult to imagine any upshot in revenue unless one has a strong competitor to MATLAB, or on the business side to Excel, or one of the statistical packages, probably SAS.

HAIGH: A question that comes to mind then on the statistical side. You had mentioned earlier that products like SAS and SPSS were eroding the market. Had there been a significant competitor in terms of a library based statistical system?

AIRD: No, I don't remember any of that. NAG had some statistics but I think that the IMSL statistic library was the dominant force on the statistics side. However, if you look at what drove sales of the library, the library was bundled, mathematics and statistics. I think that the mathematics part was driving the sales, because even when the libraries were separated people still wanted the full library. But statistical users who were doing standard things in statistics were migrating more and more to SPSS or SAS, though some of the people doing experimental things were still writing FORTRAN programs. But I think that over time that set is becoming smaller and smaller, and today is probably very small.

HAIGH: I would imagine so. Now you mentioned a culture shock between the two groups. So how was the culture of the PV-Wave people different from the culture of the IMSL people?

AIRD: They differed in how they were structured, how they were organized, how they dealt with their users. The focus of their sales for the product was such that they were not used to selling mathematics and statistics. So the Boulder group had very little interest in mathematics and statistics. These companies were put together at a management level and the employees weren't asked, "is this something that you'd really like to do, or you think it would be good." At the senior level of the companies it was decided this would be a good thing, but when push came to shove they said, "well you've got to go out, now you've got to sell this product." The PV-Wave people were used to cultivating prospects that were interested in image processing and signal processing, and those things, and not so much in the mathematics, so it was a tough line for both sides of the organization. And they were really being pushed for revenue. Revenue was now the focus of the company, instead of saying "We want to have the best mathematics routines in the world, and the easiest to use, and we love scientists and engineers," the company wanted to drive revenue.

HAIGH: Were the two groups physically merged?

AIRD: No, the PV-Wave group was located in Boulder, Colorado, stayed in Boulder, Colorado. It may still be there. And the IMSL group was in Houston, and still there is a small group in Houston that maintains the library product, but I don't know many of the details there. I think most of the management of Visual Numerics is now in California.

HAIGH: So in this period from '90 to '93, immediately before you left the company, did your role remain pretty much the same?

AIRD: At some point around 1993, I was moved out of development and into a business development role, and this was sort of my phasing out of the company. It was a trying time, I would say, because this fine group that had evolved during the 80's was really disappearing, this was becoming a completely different organization as far as the focus and what they had to do.

HAIGH: And were you responsible for making layoffs within the group?

AIRD: Layoffs occurred and I guess I had some input into that, but pretty much that was dictated. I had to do the dirty work.

HAIGH: And was Johnson's role throughout as chairman? Was he a part of the hands on day to day management?

AIRD: No, he was definitely not hands on. In the early years he was very interested in the company and came, made many visits to the IMSL people and enjoyed meeting with them. Especially with the development people, and talking about computing and mathematics, as he is an engineer by training. As this transition was made, through 1991 on, my feeling is that he became even less and less involved and had less and less direct contact with the employees. You could say some employees were happy but there was a lot of unhappiness, because there were lots of layoffs and we were in a period, where people asked "am I next?" So everybody was a little bit on edge, and some people just chose to leave the company during that time. It was a seriously big downsizing of the development group.

[End of Tape 3, Side B] [Start of Tape 4, Side A]

HAIGH: So from what you said earlier, I got the impression that the technical merging of the two products was reasonably successful and you feel that the main reason that it didn't succeed was the problems involved in selling to a different kind of user base.

AIRD: Yes, if I tried to analyze it I would say we took a product that was in this signal image processing area based on IDL. Now you have to understand that RSI, the company that originally had developed IDL was located in Boulder, and the original, president of that company, David Stern, was very active in developing the product. So there was a competitor for this IDL PV-Wave signal processing part and that company still had the focus of the development group and an appreciation for making the product better. That group has subsequently become very, very successful. In fact it was sold to Kodak recently, and I believe it achieved the goal of becoming a fifty million dollar company. So PV-Wave faced, in its own product base, some strong competition from the original developer. Then on the mathematical side you merge this thing and you have a company that had a culture that's supposed to sell this product but they don't have the interest, the training, the motivation, to sell to the mathematics community. At the same time you had MATLAB, a wonderful product that was growing by leaps and bounds, so MATLAB was winning the sales that would have gone to that product on the strictly mathematical side for the scientists and engineers, and in this more specialized area IDL was winning, so on both sides very strong competition.

HAIGH: And what was the new product called?

AIRD: Well the product from the company in Boulder that IMSL purchased was called PV-Wave.

HAIGH: And this was targeted in the signal processing area but the idea was to merge in parts of the library and make it into a more of a general purpose.

AIRD: Yes, you could say make it into a MATLAB like product. So you had the same underlying technology that you needed with this fourth generation interpretive language.

HAIGH: So IMSL didn't acquire the whole thing, they acquired this specific product?

AIRD: No, IMSL, see this is maybe a little confusing because there arethree companies involved. Precision Visuals was the name of the company in Boulder, their product name was PV-Wave. IMSL purchased the company, Precision Visuals, and then PV-Wave integrated the mathematics and statistics into it.

HAIGH: And the integrated product was still called PV-Wave?

AIRD: PV-Wave, yes. But the original product from which PV-Wave was derived was named IDL, and the company that developed it was known as RSI, or Research Systems, Inc.

HAIGH: Okay. So Precision Visuals had licensed the IDL code and built PV-Wave on top of it?

AIRD: Yes, and kind of ended up with what we would call a "paid up license." In other words, Research Systems no longer had to support and they received no more money. Precision Visuals could do anything with the product, including sell it, and they didn't have to pay any royalties.

HAIGH: Okay. And when IMSL acquired Precision Visuals they also acquired that license?

AIRD: Yes.

HAIGH: Okay. I think that clarifies that part of the story. Is there anything else you have to say about this?

AIRD: No, I think that's it.

HAIGH: So let's switch into the post IMSL years then. When you left the company did you have a particular idea of what it was that you wanted to do next?

AIRD: Well there seemed to be more and more demand for nonlinear optimization, nonlinear optimization was where I did my thesis work and I had a lot of interest in that area.

HAIGH: Had you been active in that area during your time at IMSL?

AIRD: Well, again, in the later years at IMSL my role was strictly management so I did very little development, and certainly no mathematical research, but I did have contacts and one of the contacts I had was a professor named Leon Lasdon, at the University of Texas. Leon had developed a very good nonlinear constrained optimization package, though the licensing arrangement that could never be worked out at IMSL because of the need to pay royalties. However, upon leaving Visual Numerics, I, along with Phil Smith and Mike West, formed a company named Windward Technologies. Windward Technologies was going to develop and market optimization software. So Windward Technologies signed an agreement with Leon Lasdon and started selling the code that he developed. Windward developed its own front end interface. Windward is a very small company and not a significant player in this market. Windward is still Phil Smith and myself, Mike West went on to some other position. Phil is also the Director of High Performance Computing at the Texas Tech University in Lubbock, Texas. So Phil is still only mildly active in Windward. We have a few customers, we get royalties every now and then, and every now and then make a new sale, but it's a very small business, just a few thousand dollars a year.

HAIGH: And what kind of customers had you been targeting?

AIRD: Well Windward's idea wasn't to sell to a mass audience but to find a few people who wanted to develop an application program that needed optimization capabilities, and they wanted to build it in. It had to be very flexible for licensing. There were some reasonably good customers that came along and still paying royalties. So it's a nice business but it's a very small business, it's not enough to support two full time people.

HAIGH: Right. Are there other companies that have been more successful in this niche or is it just an extremely small niche?

AIRD: Well what I see is that there’s an awful lot of mathematical optimization software out there, so there’s a lot of competition. Stanford University business office has packages like SNOPT, and there’s another group at Northwestern University that has a package named KNITRO, it’s a newer interior point algorithm. There are others that are out there, so there’s lots of competition for optimization software and they’ve continued the research. So for certain problems, you know, KNITRO might be faster, for certain linear problems SNOPT is better. I don’t know of anyone whose been making a big success in this niche, but some people are augmenting their income through sales in that area.

HAIGH: And something that we haven’t talked about is your personal involvement in the IFIP 2.5 Working Group. How did that come about?

AIRD: Well John Rice, fro many years, has been a member of that group which specializes in numerical mathematics. It’s a collection of people from around the world, and it was meeting in various places around the world once a year, to get together with people that were doing similar things. I became inactive about 4 years ago, and I guess you could say there are two reasons for that. One is that in the early days of that group there was a focus on developing exactly what IMSL was doing, so it was very directly related to my personal interests and business interests, and a lot of it was FORTRAN. Now, that group is somewhat fragmented and I don’t know if there is anybody left who’s actually doing that kind of development. The other side is that I don’t have a corporate sponsor, so traveling to the meetings becomes very expensive and I’m sort of phasing into retirement.

HAIGH: During the time that you were active on the group, were there any things that you accomplished that you think are of particular significance?

AIRD: Well in the early days that group was very significantly involved in helping with the various projects like EISPACK and LINPACK, and also very much involved in helping to develop the IEEE floating point standard, and debating that kind of thing. I would say those were the highlights of the group.

HAIGH: And are there any particular personal contributions that you can recall making, any debates within the group that you took sides on, those kinds of things?

AIRD: Most of the members were on the academic side, so I represented the business community and tried to bring that to the group.

HAIGH: And were there any areas where your perspective, coming from business, would be significantly different from the academic?

AIRD: Not really, only as a matter of emphasis that they needed to consider. I’d have views on the business side and what it really meant, how important certain things were versus other things, so it was very general, very high level. I wouldn’t say there was any debate or any arguments, just general discussion, it was always interesting.

HAIGH: Now with Windward Technologies did you return to more of a hands-on programming role?

AIRD: Well I'm the only employee, so, I do all the programming, all the sales, all the marketing. It's very inactive, but we have a web site, www.maxthis.com, and so people find us on the web, or through a contact. Windward has only developed a small amount of software, like Leon Lasdon's nonlinear optimization. The programming was mainly in making the interface a little more user friendly, so it wasn't the significant kind of development that you would get into to develop the heart of the algorithm.

HAIGH: And returning to programming did you find that a lot has changed since your more hands on work in the 1970's, you know, in terms of modern tools and platforms? Other than the obvious issue of being able to get a much quicker feedback when you make changes in the code?

AIRD: Now I don't use punch cards at all. Well my environment is PC, right now I'm using a Intel Pentium 4 hyper threaded 2.6 gigahertz system that has the Microsoft Visual Studio. Visual Studio is really a C compiler within a framework with the editor, the debugging. The FORTRAN compiler that's integrated in there has a history: it was originally developed at Digital Equipment, and then Compaq bought Digital Equipment and it became the Compaq FORTRAN compiler, and now it's migrating to Intel. Nonetheless, the version of the FORTRAN compiler that I have in that environment is still from Compaq, which is now part of Hewlett-Packard. So I have three languages you could say that I use: FORTRAN, C, and Visual Basic. The one program that I've developed is a stock market analysis program that uses an efficient Fourier calculation, so it has kind of a spreadsheet interface for data, and it has a graphical interface. So Visual Basic is really a very nice environment for doing that. The underlying optimization that's used is a C program, but the whole program is written in Visual Basic, we use a software part that plugs in to Visual Basic to do the spreadsheet part of it, and we have another part that plugs in to do the graphics part of it, and you sort of have to wire these things together. So it makes it very very effective for doing original program development.

On the other side almost all the optimization software is now written in C, and I have a strong preference for using the C language. I've done that migration myself, but still some of the work is done in FORTRAN. So the tools are wonderful as far as debugging, the editors can compare things, you can search for things, you can start the debugger and you can go to certain statements and see what variables are and change them. So if you look from 1973 to today it's not comparable at all.

HAIGH: Interesting. So other than the change from FORTRAN to C, the actual code that reflects the actual numerical algorithms would look the same, but the kinds of larger software systems in which you can now embed it would make the individual programmer much more productive?

AIRD: Yes. One example is that some of the work I do is for a company in Incline Village named Frontline Systems. Frontline Systems offers optimization technology to the Microsoft Excel community, and that's a fairly substantial example of integration of mathematical software with Excel, so you have all these interface issues. It's very doable in FORTRAN but so much of the other work is done in C that they want to have the numerical algorithms in C. But from a user point of view, the user can set up the mathematical models for optimization in Excel and really never look at any of the programming details. A user would not be able to tell whether it's in FORTRAN or C, but when you have several people that are working on a project and most of them are doing their work in C, if FORTRAN comes along it's sort of the odd man out. If you had an algorithm that was never converted to C then you work with it in FORTRAN, but if you have a choice you use the C version.

HAIGH: So moving to conclude the interview. If you looked back on your career, what you think would be the single achievement of which you're most proud?

AIRD: Well I think the work at IMSL was clearly the biggest achievement in my career. The putting together of a very, very talented development group of programmers and designers to build a library that had a world class reputation. The culmination of the release of the library in 1987 was a very, very fine product that had the attributes of being easy to use and being reliable and trustworthy.

HAIGH: So if you had to pick one thing it would be that '87 release of the library?

AIRD: Absolutely.

HAIGH: And the flip side of that. What do you think, in terms of your own responsibility, would be your biggest regret?

AIRD: Well the transition that occurred at IMSL, which became known as Visual Numerics, starting in from 1990, 1991. I went through a period of about three years where I saw all of the structure, all of the group, that had been built up through the 80's sort of disappear into layoffs, and it was a very unpleasant part of my career. I could have bailed out earlier and avoided the pain, but in some sense I guess it was an experience and maybe in some sense I made it slightly easier for the people that were involved, to the extent that I could, even though the end result was inevitable. So I guess I've gained from that experience.

HAIGH: Okay. Well that concludes all the areas that I had planned to discuss. If you have anything else that you would like to say at this point?

AIRD: I believe we've covered it all.

HAIGH: Okay, great. Well thank you very much for agreeing to participate in this interview. [End of Tape 4, Side A]

Appendix: Aird's Resume as of 2003

Thomas J. Aird
30 Snowball Ct.
Reno, NV 89511
Phone: 775-852-4685
Fax: 775-201-0034
E-mail: TomAird@aol.com
Web site: www.MaxThis.com

Professional Interests

Mathematical software with special interest in nonlinear optimization; graphical user interfaces; software parts technology, OLE and ActiveX controls; object oriented software design with Visual Basic;

Education

Ph. D., Computer Science, Purdue University, Lafayette, IN, 1973

Thesis: Computational Solution of Global Nonlinear Least Squares Problems, under Professor John R. Rice

Awarded the Distinguished Alumnus Award, Purdue University School of Science, April 1993

M. A., Mathematics, University of Michigan, Ann Arbor, MI, 1963.

B. A. with Honors, Mathematics, Olivet College, Olivet, MI, 1962.

Employment

1994 to present,	Windward Technologies, Inc., Reno, NV President
1973 to 1993,	Visual Numerics, Inc. (Formerly IMSL) Houston, TX Senior Vice President, Libraries Business Development Other positions held:

Senior Vice President, Product Development
Vice President, Research and Development

Achievements

A major part of my career has centered on the development of the IMSL Library -- a collection of 1000 Fortran 77 subroutines in mathematics and statistics. I contributed in several ways to the building of this library into a world leading product used by scientists, engineers, and professional business people throughout the world. I designed and programmed many subroutines including linear and nonlinear equation solvers. I managed the development group with responsibility for planning, design, programming, documentation, product support and the delivery of software product masters and documentation masters. Today, the IMSL Library is known for its great wealth of mathematical and statistical functionality, ease-of-use, reliability and excellent product support. The product is used in most of the major corporations, universities, and research centers throughout the world -- (E. I. DuPont, General Motors, MIT, Purdue University, NASA, ...).

I also contributed to the building of a very professional and effective development organization of mathematical, statistical, and graphics specialists.

I list the total redesign of the IMSL Library as another major accomplishment. This project took about four years, culminating in the release of a new product in 1987. The new product, while retaining all of the functionality of the old product, was better designed, had standard argument list conventions, performed workspace allocation, and included good error handling. It was quickly accepted by the customer base.

Other major product development projects that I managed:

Fortran Exponent Graphics -- a Fortran 77 graphics library companion to the math and stat library. It was released in 1990 and includes a wide range of 2D and 3D graphics functionality such as function plots, contour plots, and surface plots.

C/Stat, and Exponent Graphics for C -- library products that include much of the functionality of the IMSL Fortran libraries, but packaged for the professional C programmer. Exponent Graphics for C is much like its Fortran counterpart, but provides built-in interactivity in an object oriented framework. These products were released in 1991 and 1992.

Fortran 90 MP Library -- a small subset of routines implemented in Fortran 90 for the DEC Mpp computer (a massively parallel computer manufactured by MasPar for Digital Equipment Corporation) and optimized for parallel processing on systems with 1024 to 16384 processors. This product was released in June, 1993.

I contributed to the financial success of the company as a member of the executive management team under which revenues grew to $22M in 1992.

Other Work Experience

Year	Job Title	Company	Location
1976 - 1996	Board of Directors	Visual Numerics, Inc.	Houston, TX
1967 - 1973	Manager, User Services	Purdue University	Lafayette, IN
1965 - 1967	Scientific Systems Supervisor	Lockheed Electronics Company	Houston, TX
1965	Scientific Programmer Analyst	Wolf Research and Development	Houston, TX
1963 - 1965	Associate Mathematician	Burroughs Corporation	Detroit, MI

Memberships

1984 to 2002, Member IFIP WG 2.5, an international group dedicated to the improvement of the field of numerical software.

Publications

Miles, H. W. and Aird, T. J., -A program for Automating Circuit Layout, Journal of Industrial Mathematics Society, Vol 14, Part 2, 1964.

Aird, T. J., -Mutual Primal-Dual Method, Communications of the ACM: Algorithms, May 1966.

Keathley, S. M. and Aird, T. J., -Stability Theory of Multistep Methods, NASA Technical Note TND-3976, May, 1967.

Aird, T. J. and Lynch, R. E., -Computable Accurate Upper and Lower Error Bounds for Approximate Solutions of Linear Algebraic Systems, ACM TOMS, Vol. 1, No. 3, September 1975, pp. 217-231.

Aird, T. J. and Rice, J. R., -Systematic Search in High Dimensional Sets, SIAM Journal on Numerical Analysis, 14(2), 1977, pp. 296-312.

Aird, T. J., Battiste, E. L., and Gregory, W. C., -Portability of Mathematical Software Coded in Fortran, ACM TOMS, Vol. 3, No. 2, June 1977, pp. 113-127.

Aird, T. J., -The IMSL Fortran Converter: An Approach to Solving Portability Problems, Lecture Notes in Computer Science, Portability of Numerical Software, Edited by Wayne Cowell, Springer-Verlag, New York, 1977.