

# Holistic system for mathematical computation: a report

Henry Cejtin      Igor Rivin

## 1 Introduction

This is a report on the exploratory project entitled *A Holistic system for mathematical computation*, conducted by Igor Rivin and Henry Cejtin under the auspices of the Geometry Center.

The project has had the following goals:

- To investigate the present state of the art in programming languages and tools available for development of mathematical software.
- To investigate the present state of available tools for computation in geometry and geometric modelling.
- To develop tools for computation in geometry, both for their own sake, and as a way to test the available tools.

Since the mathematical research interests of one of us (Igor Rivin) tend to hyperbolic geometry, and since we could get a much clearer impression of the utility of the tools previously available and those we had developed if we could use them in our own research, there has been a certain bias towards hyperbolic geometry and related fields in this project. An incomplete list of the various tools we developed can be found in section 2.

Computationally, our system is largely centered on the `Scheme` programming language (see [17, 4, 3]), although many of the projects described below have been implemented using whatever seemed like the right system at the time, consistently with the “right tool for the job” philosophy underlying the project. Our `Scheme` experience is described in section 3.

The purpose of the Scheme substrate, in addition to that of being the programming language of choice for much of our work has been to serve as glue, unifying various “servers” of various kinds. For example, a lot of our graphics has been ultimately rendered in GL – the Silicon Graphics Inc graphics language, and PostScript. We have used the X and NextStep window systems to interface with the user, and we used a variety of programs as “compute servers” (Allan Wilks’ `dt` for computational geometry has been the most useful, though it does have shortcomings). We describe our (so far quite simple) communication strategies in section 4.

Finally, in the Appendix we give a summary of various papers published during the granting period, and of the talks the authors have given and conferences they had attended.

## 2 The tools

We plan to make the tools described below available by anonymous `ftp` from the geometry center.

### 2.1 Basic Utilities

These are the implementations of the fundamental data structures (eg, heaps, stacks, trees, etc) and algorithms (sorting, merging, selection) of computer science. Some of these are implemented as `Scheme` procedures, and some as `C` libraries. This collection of utilities needs a lot more work to make it truly comprehensive.

### 2.2 Mathematical expression parser

This takes a mathematical expression such as  $f(x) = x^2 + \sin(y)$ , typed at the `Scheme` listener and produces a parse-tree for it. The parse-tree is actually a `Scheme` expression, which then can be given to any of the tools described later in this section.

**Implementation note.** This somewhat  $LL(1)$ -based parser was not implemented as a `yacc` grammar, but done directly in `Scheme`

## 2.3 Adaptive function plotter

This takes as input a function of one variable, the range, and some option, and produces a plot, which is, at present, a `PostScript` program. The plotter is adaptive, in that it detects the regions in which the graph is particularly wiggly, and samples the abscissas more frequently there. In this way, it is not dissimilar to the *Mathematica* function `Plot[]`, although it is, on one hand, much cruder (does not have a Turing-complete tickmark placement language, and, as a matter of fact, does not produce tickmarks), and on the other hand, employs a superior adaptive algorithm.

**Notes.** This was produced largely as a proof of concept, since the plotting process incorporates within itself many identifiable separate components: the function evaluator (the function need not be an algebraic expression, but may be any kind of black box, which produces the  $y$ -values when given the  $x$ -values), the adaptive plotter (which needs to communicate with the evaluator when it needs more points), and the renderer(s) (since the user may be interested in, *eg*, previewing the plot on the screen, and then printing it on a laser printer).

The adaption strategy is as follows: First, a number of equally spaced points are computed, and successive points are joined by straight lines. A data structure (a heap) is constructed, where the entries are points with weights equal to, in essence, the angle between the two line segments incident to the point. The top of the heap is examined, and if the angle is sufficiently close to  $\pi$ , the adaption halts, otherwise, the two segments incident to the top of the heap are subdivided, the five new or changed points are reinserted into the heap, and the process continues.

**Future directions.** It would be nice to produce accurate drawings of implicitly defined curves in the plane (or in space), and there has been considerable demand for this functionality, especially since commercial packages do not seem to provide it. This would involve such components as adaptive solving of ODE, for which there exists sophisticated `FORTRAN` codes, which could perhaps be integrated in their entirety.

## 2.4 Polyhedral surface plotter

This takes as input a parametrized surface, of the form

$$(x, y, z) = (x(s, t), y(s, t), z(s, t))$$

and renders it to the specified resolution. This plotter is *not* adaptive. The rendering is presently done in `GL`. As before, axes, tickmarks, and other *Mathematica*-type niceties are non-existent, but in many other ways the output is superior (see notes below).

**Notes.** Just as the curve plotter described above, this need not take a symbolically defined quantity, but can deal with any black box, which takes parameter values as inputs, and produces points in  $\mathbf{R}^3$  as outputs. This is important, since by composing these black boxes, the surfaces can be deformed in useful and entertaining ways (see 2.6 below).

The surface plotter produces a triangular mesh of polygons (which are convenient for both `GL` and `PostScript` rendering), with the extra refinement that it can either produce or be given normals at the vertices of the polygons. These can, in turn, be used to embellish the surface, using one of the many smooth shading, texture-mapping, etc, algorithms. This is important for a number of reasons: the decorations can communicate mathematical information; they are aesthetically pleasing, and they can be used to decrease the resolution of the sampling, without compromising the quality of the image. That is the main reason why `GL` is the renderer preferred at the moment.

**Future directions.** The primary goal is improving the surface plotter to be adaptive. Another is dealing with implicitly given surfaces, perhaps using the algorithm of Dobkin, Levy, Thurston, Wilks.

In addition, the current reliance on `GL` is viewed by us as unhealthy, since it severely limits the choice of hardware platforms (to one – SGI). Unfortunately, SGI machines are still quite expensive and not available even to the majority of the mathematical community, never mind college and high school students. One solution would be to develop a high quality software renderer, which would remove the reliance on a particular hardware vendor, and would also allow the production of high quality images on a printed page (as opposed to CRT-resolution images now possible with SGI technology). Such a project has already been undertaken at the Geometry center (by Daeron Meyer and Tim Rowley), as part of producing a version of `Geomview` that could work on generic hardware running the `X` window system.

## 2.5 Linear Algebra

We have implemented a collection of linear algebra utilities in arbitrary dimensions in `Scheme`. These are somewhat skewed towards geometric applications, so that operations such as finding dot products, finding the intersection of linear subspaces, finding equidistants, projective transformations, matrix inverses, etc, can be performed relatively efficiently.

**Notes** Originally, we considered using some of the widely available, efficient, and robust linear algebra libraries, implemented in `FORTRAN` and available from `netlib`, and for particularly sophisticated algorithms we may still do this. On the other hand, for a lot of the applications we are interested in, the computations must be performed either with rational or algebraic numbers, or with integers or real numbers of arbitrary precision, and for this, the available libraries are completely unsuitable.

## 2.6 Hyperbolic Geometry

1. **Model transformation.** We can transform procedurally defined geometric objects defined in the conformal (Poincaré) model of  $\mathbf{H}^n$  into objects in the projective (Klein) model, and *vice versa*.
2. **Transformation transformation.** Isometries of  $\mathbf{H}^2$  or  $\mathbf{H}^3$  given as elements of  $\text{SO}(n, 1)$ , ( $n = 2, 3$ ) can be transformed into linear-fractional transformations in  $\text{SL}(2, \mathbf{R})$  and  $\text{SL}(2, \mathbf{C})$  respectively.
3. **Discrete groups.** The above foundations, together with the linear algebra capabilities described in section 2.5 can be used in a fairly elegant way to compute Dirichlet domains of discrete groups acting on  $\mathbf{H}^n$ .

**Implementation Note.** To produce a Dirichlet region given the (matrix) generators of a group, we proceed as follows: First, we compute successive “generations” of the group – the  $n$ -th generation is the set of words of length no greater than  $n$  in the generators supplied, so the 0-th generation is the set consisting of the identity matrix, while the first generation is just the generating set, together with the identity. Given the  $n$ -th generation, we produce the  $n + 1$ -st, by taking all products (on the left) of the generating set and the  $n$ -th generation,

and then prune the resulting set by deleting any elements that appears more than once. Since the computations are now performed in floating point, all *approximate* duplicates must be deleted. The precise meaning of *approximate* is not so easy to determine *a priori*.

Now, given the  $n$ -th generation of the group, for sufficiently large value of  $n$ , which is hard (or impossible) to determine algorithmically, but is usually easy to guess in specific instances, we compute the orbit of a point. Using the fact that the Poincaré model preserves spheres, it is possible to compute the Delaunay triangulation of this orbit, using Euclidean computational geometry (we used Allan Wilks' program `dt`). Now, given the Delaunay triangulation, the Voronoi cell (= Dirichlet region) of the base point can be computed using simple linear algebra in the Klein model.

It should be noted, that, while we achieved some success with this method, the existing computational geometry software (both `dt`) and the new convex hull/Delaunay triangulation program of B. Barber *et al* is rather ill suited for this problem, since the orbit is very frequently **not** in general position. Small perturbations of the points may lead to a change in combinatorics, which is, needless to say, an un-sustainable situation. Recently, the group directed by Herbert Edelsbrunner at University of Illinois (Champaign-Urbana) has implemented a collection of tools which are of the precise type that would be required ([12]). Unfortunately, this came too late for us to have had time for extensive experiments.

## 2.7 Lattice Reduction

In the seminal paper [9], the authors introduced an algorithm for lattice reduction, and used it give a polynomial-time algorithm to factor polynomials over the field of rational numbers. Since then, the lattice reduction (or LLL, or it has become known) algorithm has been used for a variety of purposes. We have become interested in it as a tool for recognizing algebraic numbers (given an inexact number, the LLL algorithm can be used to guess a polynomial of small degree and with small coefficients of which the number is a root). This turns out to be quite useful in computing arithmetic invariants (for example, the invariant trace field) of hyperbolic 3-manifolds.

We decided to implement a version of the LLL described by Schnorr and Euchner in [15]. This uses floating point arithmetic as much as possible, and only when pressed goes to exact integer arithmetic. While this version cannot be rigorously proved to run in time polynomial in the size of input, its performance in practice is much better than that of the ordinary LLL. We implemented the algorithm in C, using the GNU Multiprecision library `gmp` (see [8]) to deal with the exact integers. While the program works very well, it was noted that using a library like `gmp` was only reasonable for projects where the use of multiprecision arithmetic is tightly controlled (as it was here) and not pervasive. The reason for this is that using mantras such as `mpz_mul(&tmp, &foo, &bar)` to multiply `foo` and `bar` and to put the result in `tmp` quickly becomes burdensome. Hiding the burden by overloading the `*` operator (as can be done in C++) is possible, but the resulting code is extremely inefficient. For any larger project there is really no substitute for a language that has a first-class `integer` (as opposed to a 32-bit `int`) type.

## 2.8 Geometric Optimization.

One of us (Igor Rivin) has theoretical and algorithmic results on complexes made out of ideal hyperbolic simplices (see [14]). These results give efficient algorithms for producing optimal triangulations of surfaces, constructing hyperbolic polyhedra of prescribed shape, finding complete hyperbolic structures on manifolds equipped with a given (topological) ideal triangulation, and computing symmetry groups of such manifolds. The algorithms entail first setting up a system of *linear* constraints (equations and inequalities) for the angles of a triangulation, and then optimizing a concave functional on the feasible region (if non-empty) of the resulting linear program. To do this part, we first co-opted the public domain package `lp_solve` written by Michel Berkelaar (now at IBM Thomas Watson Research Center), which solves mixed integer/real linear programs (the integer part was not of interest in the current project). This package communicates with the rest of the suite purely by means of unix pipes, through some filters that do the appropriate conversions. Then, with the help of the singular value code from Numerical Recipe (see [13]) we implemented a version of the conjugate gradient algorithm, to optimize the convex functional (the hyperbolic volume of a certain polyhedral complex). Now it was necessary to interface this optimizer with some appropriate geometry software. The first such linkage was done with

Jeff Weeks' `Snappea` program – `Snappea`'s output format is a little opaque, but Jeff Weeks was very helpful in constructing the appropriate conversion routines. As a result, we have a suite of programs that very efficiently finds the hyperbolic structure on a given triangulated 3-manifold. Extensions to this should be able to be effectively used to study the Dehn surgery space.

A second partner for the geometric optimizer was `GraphTool` (see [10]). Again, with the help of one of the principals (Michael Dillencourt) we were able to construct the appropriate conversion filters. The resulting suite of programs have already been used to find optimal embeddings of planar graphs – the picture below is for the icosahedral graph (one of the vertices is placed at infinity).

## 2.9 Geometry of surfaces

Together with Greg McShane (see [11]) we have been studying the geometry of simple geodesics on the punctured torus (equipped with a hyperbolic met-

ric of finite volume) . It turns out that there is at most one such geodesic in each homology class, and the valuation giving for a  $(m, n) \in \mathbf{Z} \times \mathbf{Z}$ , with  $m$  relatively prime to  $n$ , the length of the simple geodesic in the homology class  $(m, n)$  can be extended to a *norm*  $\mathcal{L}$  on the homology with real coefficients. We have been studying (among other things) the asymptotics of the number of simple geodesics of bounded length (if the word “simple” was omitted, this would be given by the Selberg trace formula). The picture below of the unit ball of  $\mathcal{L}$  for the modular torus was produced by an interaction between **Scheme** and *Mathematica*. The picture was produced by recursively applying the Farey process, which is a very natural algorithm to encode in **Scheme** and not so natural in **C** (which also suffers from lack of extended precision arithmetic).

## 2.10 User interface

We have conducted experiments with both `NextStep` and `X` window systems. While `NextStep` provides an unquestionably superior development environment, it seems that `NeXT` is not on the way to being a major player (although there is still some hope for their `NextStep-486` effort), so, regrettably, `X`, `Windows`, and the Macintosh window system are the remaining window system choices. The last two of those, while by far the most popular window system presently in use, run in conjunction with extremely primitive operating systems, which are not at all hospitable to our multi-process, distributed view of the world.

In the two years since our project started, there development environment for for programs using `X`-windows has improved considerably, due in large part to the development the `tc1` (stands for “toolkit construction language”) and `tk` (stands for “toolkit”) by John Ousterhout, of the University of California, Berkeley. These allow one to construct quite professional-looking user interfaces with a minimum of programming effort. While it is impractical to write large programs in `tc1`, the language is designed to make constructing a “front end” for a program a relatively painless task. We have tested it by combining Charles Sims’ strong generating sets algorithm with a simple `tk` front end to produce a graphical  $N^2 - 1$ -puzzle (for  $N = 4$  this is Sam Loyd’s popular 15 puzzle). The group theoretic part of this is implemented in `Scheme`, and communicates with the front end by passing around closures.

## 3 Programming environments

An important facet of our endeavors has been the search for programming environments suitable for our tasks. While we started the project with few preconceptions (but with a considerable amount of experience in programming in all kinds of environments from assembler to *Mathematica*), it was clear that while `C` was adequate for some of our needs, we needed more flexible and powerful languages for the more sophisticated projects. Our first choice was `Scheme`, a language invented by Guy Steele and G. J. Sussman at MIT some twenty years ago. `Scheme` is small, clean, and powerful, and thus would make an ideal “assembly language” for a mathematical computer. `Scheme`’s main competitors in the symbolic computation world have recently

been `Common Lisp` and `ML`. There are a number of decent `Common Lisp` implementations, mostly commercial, but to us, the huge size and complexity of the `Common Lisp` system was daunting, and certainly went a long way towards discouraging any use of `Common Lisp` as a substrate. `ML` has been (and is still being) developed at the AT&T Bell Laboratories and the Princeton University Computer Science department, and has some things to recommend it (a nice semantic specification and a high quality compiler), but it has a couple of drawbacks, chiefly a strong type system (which we found unnecessarily restrictive) and a somewhat baroque syntax.

The `Scheme` language was first specified almost twenty years ago, but, sadly, (perhaps due to the dominance of `C` on one hand, and the defence department advocacy of `Common Lisp` and `Ada` on the other), there does not exist any truly satisfying `Scheme` programming environment at this time. Here is a brief synopsis of what we have found:

1. **The interpreters.** There are a number of free, and reasonably good `Scheme` interpreters available (usually free). Some of these are quite venerable (`MacScheme` ([2], `PC-Scheme` ([5])), some newer (`scm`, from Project GNU, `vscm` from Princeton, etc), but none have performance within a factor of 10 of `C`, and hence are not really suitable as bases on which to build large systems. However, the performance for basic programming tasks is much higher than that of systems like `Maple` or `Mathematica`. A lot of our more recent projects have used `vscm`.
2. **The compilers.** The only commercial `Scheme` development environment currently supported is `ChezScheme` (see, eg, [7, 6]), from Cadence Research, in Bloomington, IN. `ChezScheme` does come with a compiler of sorts, which is optimized for the speed of compilation, rather than the speed of generated code, and does not support creating standalone executables (which form much of the basis of our system). In addition, it is not cheap. Another, widely distributed (and free) `Scheme` system is `MIT-Scheme` (see [1]). This does have a compiler, and a rather extensive development environment. It does, however, suffer from numerous drawbacks. Among them are the following: the compiler only generates code for a small number of target architectures; the generated code is of rather poor quality, when it exists; there is no capability of generating standalone executables; the system is extremely large,

and usable only on extremely well-configured machines. All these features conspire to rule it out from serious consideration. Yet another relatively widely available system is **T** (see [16]), developed at Yale in the mid-eighties. The system was originally quite well regarded in the **Scheme** community, but is now definitely showing its age – the compiler, once regarded as an advance, is now mediocre (partly because the original target architecture was the DEC Vax, which does not map well to modern RISC processors), the development environment is abysmal, the ability to generate standalone executables is rumored to exist, but was never actually demonstrated to our satisfaction, and, again, the system has only been ported to a relatively limited number of machines. A rather good more recent compiler is **Gambit**, from Brandeis University, which has most of the features we desire, but only runs on Motorola 680x0 processors. Since there are no UNIX 68k-based machines manufactured at this time, this does not seem to be a feasible choice.

In many of our projects we have used **SchemeToC**. **SchemeToC**, as the name suggests, compiles **Scheme** programs into **C**, and then lets the **C** compiler do the rest. This has some advantages – certain programs achieve performance equal to (or greater than) that of corresponding **C** programs, and, perhaps more importantly, it is quite easy to implement a foreign function interface, and thus to communicate with other **C** programs, the UNIX operating system, and various utilities among them. **SchemeToC** also makes it easy to create standalone programs. The drawbacks of **SchemeToC** are less obvious, but still rather significant. The main is that certain very important compiler optimizations cannot be performed if the “target machine” has **C** as the machine language, and hence various important classes of programs run fairly slowly. In addition, **SchemeToC** has no development environment or debugging tools to speak of, besides the ability to run the program interpreted (this last is extremely important, however). Furthermore, **SchemeToC** does not come with a bignum implementation, which makes it hard to use for exact or algebraic computations, or computations which require extended precision floating point computations. The bignum capability could be added (there are some portable bignum package around, notably **BigNum**, from INRIA), but this would require considerable effort.

In conclusion, `SchemeToC` has proved a somewhat satisfactory experimental platform, but it is clear that a serious implementation would have to be undertaken, if this work is to be taken much further in certain directions.

More recently there has been (in essence) a somewhat more modern implementation of `SchemeToC`, at INRIA in France, called `Bigloo`. This is somewhat more efficient, but suffers from a lot of the same shortcomings.

Recently, we became aware of a DARPA-funded project at Carnegie-Mellon University to develop a high performance `Dylan` system. `Dylan` is a sort of a hybrid of `Scheme` and `Common Lisp`, designed at Apple Computer, and although we have considerable reservations about some of the technical aspects of its design, it could conceivably provide a serviceable compromise. Another strike against `Dylan`, however, is that it is copyrighted by Apple Computer (which has been quite aggressive in the past about defending what it perceives as its intellectual property).

## 4 Communicating with the external world

Since communication is at the heart of our system, a few words should be said about what we have been doing so far, even though much of it has been quite traditional.

1. **Foreign function interface.** We have used this when the other end of the conversation was a particularly primitive `C` or `FORTRAN` library (eg, `GL`). `SchemeToC` provides an only mildly painful way to do this, and one can (and we did) use the greater expressivity of `Scheme` to streamline the original library interface (an example: `GL` has many versions of each library routine, depending on whether the arguments are of type `int`, `float`, or `double`. This can be made completely invisible at the `Scheme` level.
2. **UNIX `stdio` library.** A lot of our external communication tools are constructed on top of the standard UNIX system calls and libraries, which have been repackaged and streamlined, to be easily accessible

from the **Scheme** world, and to be used in creating abstraction barriers as necessary. For example, in our interactive plotting system, the **PostScript** interface is only about a dozen lines of code, which could be easily modified to speak to **GL** or any other renderer.

3. **Scheme closures.** This is what we use internally to arrange communication between various modules of our program, and it has pretty much fulfilled our expectations, as the method of choice. There are still a number of syntactic and performance issues to settle, however.
4. **Files.** Certain other programs, most particularly **Geomview**, is most easily communicated with by way of files, and we have facilities to read and write files in **Geomview** format.

## 5 A few words about performance

This has been a serious concern when we started this project, both for us, and for many people at the Geometry Center and elsewhere, with whom we have discussed our ideas. Rather gratifyingly, performance has not, by and large, been a problem, even though we did not strive to write our programs in a particularly efficient way. The speed bottleneck for most of our experiments has been rendering speed, which, on SGI workstations, is a hardware function, and hence out of our control. It is true, however, that some of our programs have been rather memory intensive (sometimes using as much as 50 megabytes for the more complicated discrete group computations and rendering). We know ways to reduce this memory usage by a large factor, but as mentioned above, this has not been a top priority.

## 6 Summary

We have developed a number of tools, and conducted a number of experiments, which convince us that the path we have chosen is a very promising one. We very much hope that we will be able to pursue our ideas further, and we expect that a larger scale effort will succeed in unifying the various tools we are developing into a hoped-for coherent whole. The need for such a system is, if anything, become more acute in the last two years. With the

continuing drop in the price of hardware, and with the continuing increase in availability of high quality free operating and systems software (most importantly, the GNU software suite and the Linux operating systems), and the great explosion in the use of Internet, for the first time secondary schools and people with home computers actually have the opportunity to use the software previously available only to academic and corporate research community. This greatly enhances the need to make such software actually useable. In addition, as the number of users and of creators of such software explodes, it will no longer be as practical for people to track down the authors in order to figure out how to talk to a particular tool – it is thus becoming increasingly important to put effort into the design of proper communication protocols.

## A Publications

The following papers appeared or were started during the course of the grant period; a number of others are in preparation:

- Igor Rivin, On the geometry of ideal polyhedra in hyperbolic 3-space. *Topology*, v. 32, n. 1, January 1993. refereed research article, 8 pages.
- M. M. J. Treacy, Satish Rao and Igor Rivin, A topological method for determining new zeolite frameworks. In proceedings of *Ninth International Zeolite Conference*, Montreal, Canada, July 1992. refereed, 6 pages.
- C. D. Hodgson, Igor Rivin and Warren D. Smith, A characterization of ideal polyhedra and polyhedra inscribed in the sphere. *Bulletin of the AMS*, **27**(2), October 1992, refereed research announcement, 6 pages.
- Yuh-Dauh Lyuu and Igor Rivin, A note on tight bounds for transition to perfect generalization in perceptrons. *Neural Computation*, **4**(6), November 1992. Refereed research article, 10 pages, contribution level.
- Igor Rivin and C. D. Hodgson, A characterization of compact convex polyhedra in hyperbolic 3-space. *Inventiones Mathematicae*, v. 111, f. 1, January 1993 (corrigendum, v. 117(1994), p. 359). refereed research article, 34 pages.

- Igor Rivin, Intrinsic geometry of convex ideal polyhedra in hyperbolic 3-space. In: Analysis, Algebra and Computers in Mathematics – Proceedings of the 21st Nordic Congress of Mathematicians, in the series Lecture Notes in Pure and Applied Mathematics, Marcel Dekker, New York-Basel-Hong Kong, 1994. refereed, 17 pages.
- Igor Rivin, Counting simple geodesics on tori. Institut Fourier preprint, June 1994, accepted by publication by *International Mathematics Research Notices*, refereed, 8 pages.
- Greg McShane and Igor Rivin, Geometry of geodesics and a norm on homology. Preprint, 10 pages.
- Igor Rivin, A characterization of ideal polyhedra in hyperbolic 3-space. Submitted for publication.
- Igor Rivin, Euclidean structures on simplicial surfaces and hyperbolic volume. *Annals of Mathematics*, **139**, May 1994, refereed research article, 28 pages.
- Igor Rivin, On the simplex. Submitted for publication.
- Igor Rivin, Ilan Vardi and Paul Zimmerman, The  $N$ -queens problem. *American Mathematical Monthly*, August-September 1994. refereed research and survey article, 14 pages.
- Igor Rivin and Srimat Chakradhar, Discrete test generation by continuous methods. Proceedings of the IEEE VLSI Testing Symposium, Atlantic City, April 1994, accepted for publication in *Electronic Testing*. Eight pages, refereed (both conference proceedings and journal) research article.
- Henry Cejtin, Igor Rivin, Warren Smith, and Frederic Tudor, An analysis of the game of Kalah. NEC Research Institute preprint, 1993. Six pages.

## B Travel, Meetings and Talks

During the grant period, Igor Rivin attended a number of meetings spoke at a number of venues. Among these were:

- An invited talk at the special session on hyperbolic geometry, AMS regional meeting at the University of Southern California, Los Angeles, CA. (November 1992).
- A Karcher colloquium at the University of Oklahoma (Norman) (November 1992).
- An invited talk at Warwick University in June 1993 (in conjunction with the special year in hyperbolic geometry)
- An invited address at the conference on geometric and analytic aspects of hyperbolic space, Durham, UK (July 1993)
- An invited talk at New York University, November 1993.
- An invited talk in the special session on discrete geometry, AMS regional meeting at Brooklyn, NY (April 1994)
- Two invited talks at Institut Fourier, Grenoble, France (June 1994)
- An invited talk at ENS, Lyon, France (June 1994)
- A colloquium at the computer science department, University of Chicago (July 1994).
- Two invited seminars at California Institute of Technology (September 1994)
- An invited seminar at University of Southern California (September 1994).

In addition, Igor Rivin has attended the MSRI conference on mathematical visualization in fall of 1992, numerous DIMACS conferences on mathematics and computation, the Geometry Center workshop on computational group theory (January 1994), and made several visits to the Geometry Center.

He has also worked and had discussions with people too numerous to mention. The discussions Jeff Weeks, Michael Dillencourt, Herbert Edelsbrunner, Allan Wilks, Rico Tudor, and many members of the Geometry Center staff particularly stand out.

## References

- [1] *Mit Scheme Manual, Seventh Edition*. Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Mass., September 1984.
- [2] *Macscheme Reference Manual*. Semantic Microsystems, Sausalito, California, 1985.
- [3] Revised<sup>3</sup> report on the algorithmic language scheme. *ACM Sigplan Notices*, 21(12), December 1986.
- [4] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, Mass., 1985.
- [5] David H. Bartley and John C. Jensen. The implementation of pc scheme. In *Proceedings of the 1986 ACM Conference on Lisp and Functional Programming*, pages 86–93, 1986.
- [6] R. Kent Dybvig. *The Scheme Programming Language*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1987.
- [7] R. Kent Dybvig and Bruce T. Smith. *Chez Scheme Reference Manual Version 1.0*. Cadence Research Systems, Bloomington, Indiana, May 1985.
- [8] T. Granlund. *The GNU Multiple Precision Arithmetic Library*. The Free Software Foundation, Cambridge, Massachusetts, May 1993. Version 1.3.2.
- [9] A. K. Lenstra, H. K. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.
- [10] V. J. Leung, M. B. Dillencourt, and A. L. Bliss. **GraphTool**: A tool for interactive design and manipulation of graphs and graph algorithms. In N. Dean and G. E. Shannon, editors, *Computational Support for Discrete Mathematics: DIMACS Workshop*, New Brunswick, NJ, March 1992. American Mathematical Society, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 15.

- [11] Gregory McShane and Igor Rivin. Geometry of geodesics and a norm on homology. Preprint, September 1994.
- [12] Ernst Mücke. *Shapes and Implementations in Three-dimensional Geometry*. PhD thesis, University of Illinois, Urbana-Champaign, 1993.
- [13] W. Press and S. Teukolsky. *Numerical Recipes in C*. Cambridge University Press, 1989.
- [14] Igor Rivin. Euclidean structures on simplicial surfaces and hyperbolic volume. *Annals of Mathematics*, 139, May 1994.
- [15] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In L. Budach, editor, *Fundamentals of Computation theory (FCT'91)*, Gosen, Germany, September 1991. Springer Verlag. Springer Lecture Notes in Computer Science, 529.
- [16] Stephen Slade. *The T Programming Language*. Prentice-Hall Inc., Englewood Cliffs, N.J., 1987.
- [17] Gerald Jay Sussman and Guy Lewis Steele Jr. Scheme: an interpreter for extended lambda calculus. MIT AI Memo 349, Massachusetts Institute of Technology, Cambridge, Mass., December 1975.