Interdisciplinary Information Sciences, Vol. 9, No. 1, pp. 183–187 (2003)

OPTi's Algorithm for Drawing the Limit Set

Masaaki WADA

Nara Women's University, Nara, Japan

Received July 1, 2002; final version accepted January 31, 2003

This note contains a short informal description of how OPTi draws the limit set, together with an introduction to programming for mathematicians.

KEYWORDS: once-punctured torus, limit set, mathematical programming

1. Recent Circumstances in Programming

This workshop "Topology and Computers" hosts people with diverse backgrounds. So, first I would like to write about recent situations in programming.

1.1 Programming language

Which one of the many programming languages should one choose? For the purpose of writing mathematical programs, the C language or the C++ language would be the most popular choice. Incidentally, the source code of OPTi is written in C++.

The C language is the most widely used programming language in science and technology fields today. It has a long history, and has an advantage of having a huge collection of program libraries. There are many good books on C, hence it is easy to learn.

The C++ language was newly developed around 1990. It is based on the C language and extends C so to allow the programmers to use the object-oriented programming style. Its specifications are upward compatible with those of C. Thus a program written in C can be run without alteration as a C++ program.

1.2 Programming in practice

To start programming, it is not enough to read textbooks. One, of course, needs a computer itself, and a package for writing programs, too. The first hurdle for the beginning programmer is not the content of the programming itself, but actually to set up the necessary environment. In this sense, to compile and run the tasteless program, which appears at the beginning of most textbooks and simply displays "Hello World!", is the first real obstacle.

About the programming itself, one will soon be able to write programs using conditional branches and loops (respectively `if` statements and `for` statements in the C language). A function in the C language is an arrangement of a series of procedures, so is quite different from a function in mathematics. The next hurdle in programming would be the efficient use of functions. In particular, it is important to understand the concept of pointers. If one can apply recursive calls of functions using pointers freely, he or she can write fairly advanced programs mathematically.

In additions to these, using the C++ language, one can adopt object-oriented programming. The paradigm of object-oriented programming is not a must for writing programs. But, it is rather a convenient framework in view of reuse of programs and management of the source code of large-scale programs.

1.3 Operating systems

Most computers used today fall into one of the three categories: Windows machine, Macintosh, and Unix machine. This classification is according to the operating system; the makers of the computer hardware vary. NEC, Fujitsu, IBM, Sony, and most other computer makers, except Apple, adopt Microsoft Windows (general term for Windows 2000, Windows NT, Windows XP, etc.) for their computer hardware. Windows machines, in total, have the largest share (90%?) in the market.

Apple sells Macintosh series of computers with their own operating system, Mac OS. They are popular in some fields like design, publication, and education. Macintosh is popular in universities and research institutes; the author also uses Macintosh usually. Thus OPTi is an application program for Macintosh computers.

Unix was first developed in the 1970s, and has been used ever since. Except for the case of some commercial products, its source code is generally open to the public, and many people have been revising it. In contrast to Windows and Mac OS, which were developed as operating system for personal computers, Unix has been developed as a multi-user operating system right from the beginning. It excels in network management, and is often used as operating systems for server computers. Unix has some variations like BSD, System V, and Solaris that is a commercial Unix. Recently, another variation of Unix called Linux has been popular among people around users of personal computers.

To use Unix on a personal computer, one needs to install Unix from, say, a CD to a Windows machine.

Apple had been developing its Mac OS up to version 9 as their original operating system. But, when it introduced the current version of Mac OS called Mac OS X (ten), Apple made a big change. Apple replaced the kernel (the core of the operating system which deals with tasks like memory management and process scheduling) of Mac OS with that of a BSD Unix called Mach kernel. It is the transition period right now, and programmers of Macintosh need to consider users of both Mac OS 9 and Mac OS X.

An application program designed for Mac OS 9 is called a Classic application, and that designed for Mac OS X is called a Carbon application. Both of them are carefully arranged so that they run on both Mac OS 9 and Mac OS X. Besides them, Apple supports yet another type of model for application development called Cocoa (Table 1). By the way, OPTi 3.30 is a Classic application, but the author is planning to change it to a Carbon application in the near future.

Table 1. Transition from Mac OS 9 to Mac OS X

	Classic application	Carbon application	Cocoa application
Mac OS 9	Yes	Yes #1	No
Mac OS X	Yes #2	Yes	Yes

#1 with CarbonLib
#2 with Classic environment

1.4 Developing programs

The process of program development varies with the operating system (Table 2). Assume using the C++ (or C) language. For developing programs on Windows, the Microsoft Visual C++ package is the de facto standard to use.

These days, programmers do not write the source code for an application program from scratch. Instead they use the prototypes of application programs provided as class libraries by the makers, etc. , and write only the additional parts of the program necessary for implementing the functionalities they want. On Windows, it is convenient to use a class library called MFC (Microsoft Foundation Classes).

Table 2. Programming Environments

OS	Windows	Mac OS	Unix
Graphic	(Win32 API)	QuickDraw	X Window
Coding Software	Visual C++	CodeWarrior	GNU tools, etc.
Application Framework	MFC	PowerPlant	OSF/Motif, etc.

To develop programs on Macintosh, one used to have programming environments provided by Symantec, and Apple itself, but currently, the only package usable in practice is Metrowerks CodeWarrior. This package is fairly complete, and the support from Metrowerks is very good, thus is quite convenient to use, although its fairly high price could be a drawback for students. To develop applications using CodeWarrior, it is convenient to use the class library provided by Metrowerks called PowerPlant.

The author does not have a detailed knowledge of developing application programs in the Unix environment. Using GNU tools with X Window and OSF/Motif package should be quite common.

On Unix, many developing environments in various levels are available, and may be obtained freely through the network. Their source codes are often available, too, thus can be modified if necessary. However, since these tools are usually maintained by individual people, it is not easy to find what tools are available. Also the quality of the support varies with the tool.

1.5 Graphics

Displaying a beautiful graphics using a computer is one of the real pleasures of computer programming. However, the programming environment for computer graphics is not well ordered.

One of the features of the C and C++ languages is that their language specifications do not specify the things that depend on the operating system or the computer hardware, but they rely on external libraries for these. In particular, C and C++ do not have any standard for graphics. Therefore, the method for producing graphics varies with programming environments.

On Windows, a collection of system calls called Win32 API has a set of functions dealing with graphics. A program uses these system calls to draw pictures on the computer display. Mac OS has, as a part of the operating system, a component called QuickDraw, which deals with drawing pictures. On Unix, the standard for drawing pictures on the display is the graphic library developed at MIT called X Window. Although these graphic libraries have conceptually many things in common, they are totally different in the code level. To develop graphic programs for different operating systems, one needs to rewrite the graphics part of the program completely.

This is totally inconvenient. The program development is inefficient. So, recently people are setting forward a standardization of graphic programming around the graphic library called OpenGL. It may become possible to write graphics programs independent of the underlying operating system in the near future.

2. Algorithm for Drawing the Limit Set

Now, let us move to the topic of drawing the limit set. Mathematically speaking, OPTi deals with quasi-conformal deformations of once-punctured torus groups, but we do not provide precise definitions and terms here. Interested readers are referred to [1–3].

To understand the following, the fact we need here is that the limit set of a quasi-Fuchsian once-punctured torus group is a Jordan curve on the sphere, which contains a dense subset of points called cusps.

2.1 Symbolic representation of the limit set

A once-punctured torus is a branched covering of index 2 of a $(2,2,2,\infty)$-orbifold. Therefore, the limit set of a once-punctured torus group is the same as that of the corresponding $(2,2,2,\infty)$-orbifold group. OPTi uses $(2,2,2,\infty)$-orbifold group for internal computations, for it is easier to deal with.

In the Poincaré disk D, consider a triangle T whose vertices are on the boundary S^1_∞ of D. From a point inside the triangle T, drop perpendiculars to the three edges of T and name the feet p, q, r. Let P, Q, R denote the π-rotations about these three points respectively. Then, they generate a Fuchsian $(2,2,2,\infty)$-orbifold group. (Fig. 1) Note that the limit set of a Fuchsian group is the circle at infinity S^1_∞ itself. In this situation, a cusp is a point on S^1_∞ which can be obtained from the vertices of T by applying P, Q, R a finite number of times.

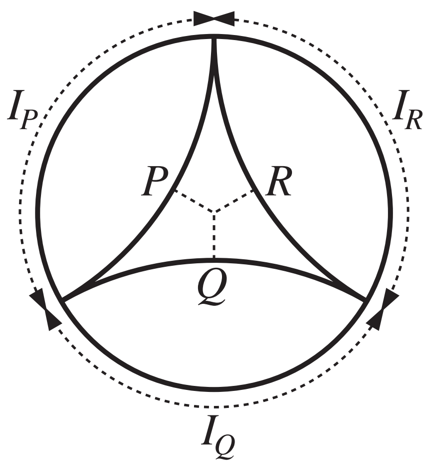

Fig. 1. P, Q, R generate a $(2,2,2,\infty)$-orbifold group

Divide S^1_∞ at the vertices of T into three arcs, and denote by I_P, I_Q, I_R, the arcs on the side of P, Q, R respectively. Then we can "represent" every point on S^1_∞ by an infinite sequence of P, Q, R as follows.

For each point x on S^1_∞, define the sequence of points $\{x_i\}$, and the sequence of symbols $\{S_i\}$ consisting of P, Q, R by the following

$$\begin{cases} x_0 = x, \\ x_i \in I_{S_i}, \\ x_{i+1} = S_i(x_i). \end{cases}$$

For any $x \in S^1_\infty$, the sequence of symbols

$$S_0 S_1 S_2 \ldots$$

satisfies the condition $S_i \neq S_{i+1} (\forall i)$, and may be considered as an infinite word in the $(2,2,2,\infty)$-orbifold group

$$\langle P, Q, R \mid P^2 = Q^2 = R^2 = 1 \rangle.$$

We call this sequence a symbolic representation of $x \in S^1_\infty$.

This correspondence is one to one, except for cusp points x, to which exactly two symbolic representations correspond. For example, if $x_n \in I_P \cap I_Q$, the corresponding symbolic representations are the following:

$$S_0 \ldots S_{n-1}PRQPRQ\ldots$$

$$S_0 \ldots S_{n-1}QRPQRP\ldots$$

This situation is similar to decimal expansion, where real numbers correspond to infinite sequences of digits and every finite decimal allows two decimal expansions, one ending in 0's and the other ending in 9's.

The above correspondence has the inverse defined as follows. First note that the mapping

$$P : I_P \longrightarrow I_Q \cup I_R$$

is expanding. Namely,

$$|P(x) - P(y)| \geq |x - y| \quad (\forall x, y \in I_P).$$

By definition, we have

$$x_n = S_{n-1}\ldots S_1S_0(x)$$

and hence

$$x = S_0S_1\ldots S_{n-1}(x_n), \quad x_n \in I_{S_n}.$$

Since each S_i on the right hand side is a shrinking map, the length of

$$S_0S_1\ldots S_{n-1}(I_{S_n})$$

decreases as n increases, and will become zero in the limit. (This is not a rigorous explanation.) Thus, every symbolic representation

$$S_0S_1\ldots S_n\ldots$$

determines a point x.

Let us identify the point x on S^1_∞ with its symbolic representation and write

$$x = S_0S_1\ldots S_n\ldots$$

Then for $S = P, Q, R$, we have

$$S(x) = SS_0S_1\ldots S_n\ldots$$

2.2 Drawing the limit set

In the previous section, we explained the correspondence between points on the limit set and their symbolic representations in the case of Fuchsian group. Actually, this correspondence also holds in the case of quasi-Fuchsian group. We can make use of this fact to draw the limit set of quasi-Fuchsian $(2, 2, 2, \infty)$-orbifold groups efficiently.

Computers do not deal with infinite sequences. So we must use approximations by finite sequences. Namely, we cannot draw the Jordan curve that is the limit set directly, so we approximate it by line segments joining the cusps, which are densely distributed in the limit set.

The algorithm for drawing the limit set may be arranged as follows. For instance, to process the interval corresponding to the sequence

$$S_0\ldots S_{n-1}P,$$

noting that the interval is the union of two subintervals corresponding to the sequences

$$S_0\ldots S_{n-1}PQ$$

$$S_0\ldots S_{n-1}PR$$

we can process the two subintervals separately. This way we may aptly apply a recursive function call. If the distance between the two end points of the interval to be processed is small enough (less than 1/10 of the pixel size, for instance), we stop processing subintervals and just join the two points by a straight line.

The basic idea for drawing the limit set has been said. Since OPTi starts from the upper half plane model, instead of the Poincaré disk, the algorithm must be modified accordingly. Moreover, the condition for terminating recursive calls does not depend simply on the comparison with the pixel size, but depends also on whether the processed part is inside of the displaying window or not, and are finely tuned. We omit the details here.

REFERENCES

[1] Akiyoshi, H., Sakuma, M., Wada, M., and Yamashita, Y., "Punctured torus groups and two-parabolic groups," Analysis and Geometry of Hyperbolic Spaces, RIMS Kokyuroku, No. 1065, 61–73 (1998).

[2] Akiyoshik, H., Sakuma, M., Wada, M., and Yamashita, Y., ''Ford domains of punctured torus groups and two-bridge knot groups,'' Hyperbolic Spaces and Related Topics II, RIMS Kokyuroku No. 1163, 67–77 (2000).
[3] Wada, M., ''OPTi's algorithm for discreteness determination,'' (Japanese), preprint.