Introduction to McBilliards

W. Patrick Hooper and Richard Evan Schwartz *

September 28, 2004

Abstract

The purpose of this paper is to describe a computer program we wrote, called McBilliards, which effectively searches for periodic billiard paths in triangles.

1 Introduction

Let T be a triangle−more precisely, a triangular region in the plane−with the shortest edge labelled 1, the next shortest edge labelled 2, and the longest edge labelled 3. A *billiard path* in T is an infinite polygonal path $\{s_i\} \subset T$, composed of line segments, such that each vertex $s_i \cap s_{i+1}$ lies in the interior of some edge of T, say the n_ith edge, and the angles that s_i and s_{i+1} make with this edge are complementary. The infinite sequence $\{n_i\}$ is the *orbit type*. A billiard path models the trajectory taken by a frictionless and infinitely small billiard ball as it rolls on a billiard table shaped like T.

In 1775 Fagnano showed that every acute triangle admits a periodic billiard path. In the early 1990s, Galperin-Stepin-Vorobets [**GSV**] and independently Holt [**H**] proved that every right triangle admits a periodic billiard path. Compare [**CPK**] and [**T**]. Masur [**M**] proved that a triangle−and in fact a general polygon−has a periodic billiard path provided that its angles are rational multiples of π. Little is known about the case of irrational obtuse triangles. So far the results along these lines have to do with characterizing the orbit types which can appear as periodic billiard paths. See [**HH**].

* Supported by N.S.F. Grant DMS-0305047 and also by a Guggenheim Fellowship

Recently, we wrote a computer program, McBilliards, which searches for periodic billiard paths in triangles. Using this program we can obtain various results. For example, a triangle has a periodic billiard path provided all its angles are less than 100 degrees [**S1**]. There is nothing special about the number 100; it is simply a convenient place to stop computing. We hope that our program will eventually settle the basic question about existence of periodic billiard paths in triangles. [1]

Our program has a graphical user interface, written in tcl/tk [**O**], which makes the organization of data gleaned from the search convenient and efficient. The computational guts of the program are written in C. The graphical user interface and the initial search program were written by Rich. Pat subsequently redesigned the search algorithm, making is much faster and more efficient. (Pat's ideas improved Rich's initial search algorithm as well.)

The reader can download McBilliards from URL

www.math.umd.edu/∼res/Billiards/index.html.

McBilliards requires a C compiler and a tcl/tk interpreter, items which can be downloaded free from the internet. We hope to have a Java version available one of these days. Also, we plan to update McBilliards periodically, adding new features as we think of them.

The purpose of this article is to describe the basic operation of McBilliards. The paper is organized as follows: In §2 we will describe the overall structure and operation of McBilliards. In §3 we will explain the original search algorithm employed by McBilliards. In §4 we will explain the improved search algorithm. Actually, the current version of McBilliards uses two of Pat's search algorithms, and only one of them is documented here.

Rich would like to thank Curt McMullen for many discussions about McBilliards and it's potential uses. He would also like to thank the Max Planck Institute in Bonn for providing a very stimulating environment in which some of McBilliards was developed.

[1]There is a second computer program which searches for periodic billiard paths in triangles written, inpdendently from us, by Marek Rychlik and Justin Theissen from the University of Arizona. However, as of this writing, the Rychlik-Theissen program is not publicly available and we have not been able to find out how the program works or what results have been obtained from it.

2 The Graphical User Interface

2.1 Overview

As we mentioned in the introduction, the graphical user interface portion of McBilliards—the part with which the user actually interacts—is written in tcl/tk. When activated, the program pops up as a big rectangular window. Figure 2.1 shows a schematic picture of the graphical user interface of McBilliards.

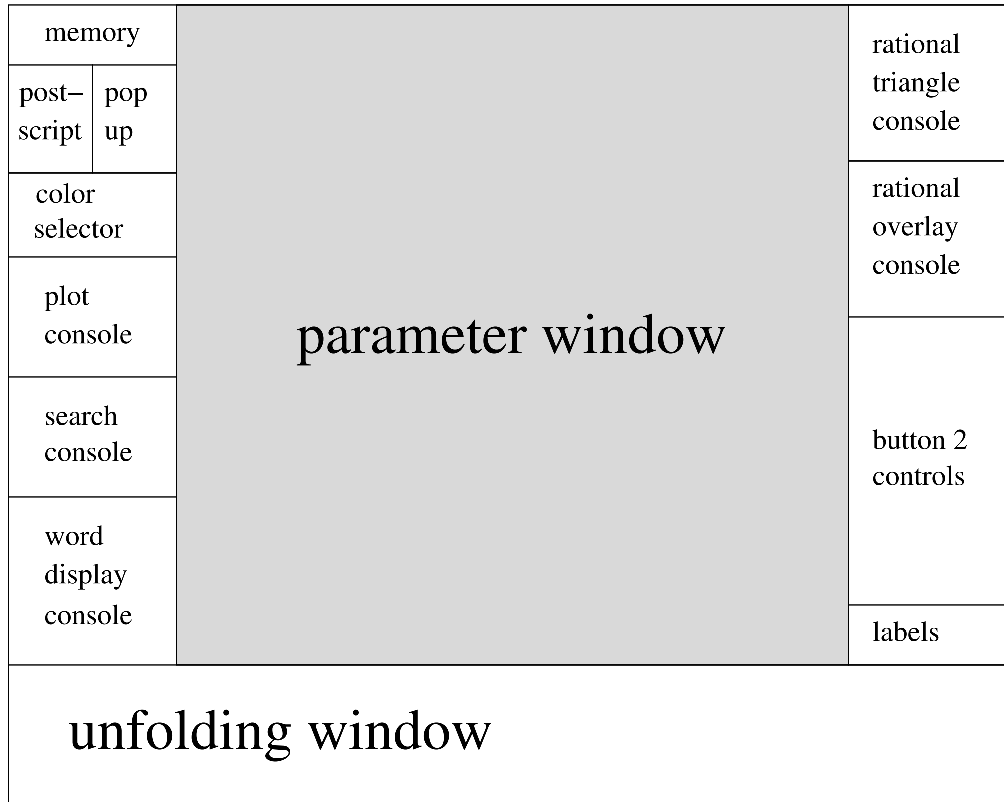

Figure 2.1

In the next sections we will describe the functions of each of these components.

2.2 Parameter Window

The parameter window represents the parameter space of triangles. The point (x, y) in the parameter window corresponds to the triangle $T(x, y)$, two of whose angles are $\pi x/2$ and $\pi y/2$. The line of right triangles divides parameter space into the set of acute triangles and the set of obtuse triangles. The latter set is shaded in Figure 2.2.

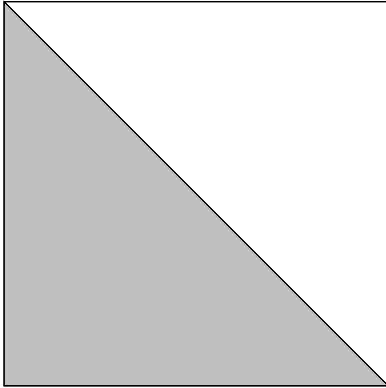

Figure 2.2

Clicking the middle mouse button on the point (x, y) selects the triangle $T(x, y)$ for study. Clicking on the left or right mouse buttons scale the parameter window, so that the user can zoom in to, or out from, the computer plots.

The main objects plotted in the parameter window are *orbit tiles*. We now explain what these objects are. Given a fixed orbit type W, the set Tile(W) is the set of points (x, y) such that $T(x, y)$ has a periodic billiard path of type W. We call Tile(W) the orbit tile associated to W. During a typical session with the program, the parameter window is partially filled with a finite list of orbit tiles−the resulting picture looks line an atlas of an alien world, with each orbit tile being a country and the empty spaces being the ocean. §3.1 has more info on orbit tiles.

One might choose the word W in some arbitrary way and then try to plot Tile(W). However, typically Tile(W) is empty. The words W which yield nonempty orbit tiles are rather special, and need to be found using a search. We will explain this in the next section.

2.3 The Search Console

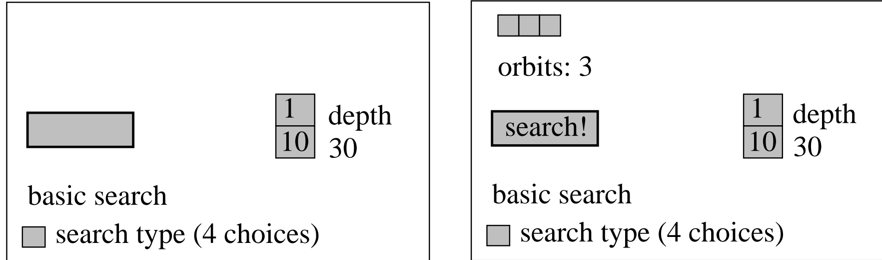

Figure 2.3

Figure 2.3 shows a picture of the search console. The left half shows what the console looks like before a search is done and the right half shows what the picture looks like when (an example) search is done.

Before searching, one selects a triangle $T(x, y)$. The search is done with respect to this selected triangle. The two buttons at right control the depth of the search. For instance, when the depth is 30, McBilliards finds all periodic billiard paths for $T(x, y)$ whose orbit types have even period less than or equal to 30. (For the measure search the depth has a different but vaguely related meaning.)

The *search!* button initiates the search. When the search is done, the found orbits are "stored" in little boxes which we call *orbit boxes*. (There are 3 of these in a row.) In the program these boxes are yellow. When one clicks on a yellow box, the orbit type W corresponding to that word is brought to the attention of McBilliards, and McBilliards does 3 things:

- Displays W in the word display console.
- Displays the *unfolding* of $T(x, y)$ in the unfolding window.
- Gives the user the option of plotting the orbit tile associated to W.

We will explain each of these features below. The *search type* button lets the user toggle between

- the basic search, described in §3
- the slalom search, described in §4
- the measure search, as yet undocumented

- the right angled search, which is a variant of the basic search.

The right angled search, which is a variant of the basic search. The right angled search looks for periodic billiard paths in right angled triangles. In this case $T(x, y)$ is replaced by the right triangle $T(x', y')$, where

$$x' = x + a/2; \quad y + a/2; \qquad a = 1 - (x + y). \tag{1}$$

At the end of a right angled search the found orbit types are "stored" in green rather than yellow boxes. Otherwise the state of McBilliards is the same.

There is one subtle point about clicking on the orbit boxes. When one uses the left button, the word is replaced by it's left rotation. When one uses the right button, the word is replaced by it's right rotation. Here, the left and right rotations amount to cycling the word one unit clockwise or counterclockwise. For instance the left rotation of *orange* is *rangeo*. When the middle button is used, the word is not altered at all. We have added this feature because the McBilliards search considers two words equivalent if they are rotations of each other. This, strictly speaking, McBilliards finds not every periodic orbit type, but just every representative of an equivalence class of periodic orbit types.

2.4 The Word Display Console

Figure 2.4 shows a the word display console when $W = 31323213132321$. The triangle $T(x, y)$ is also displayed, as the lightly shaded triangle.

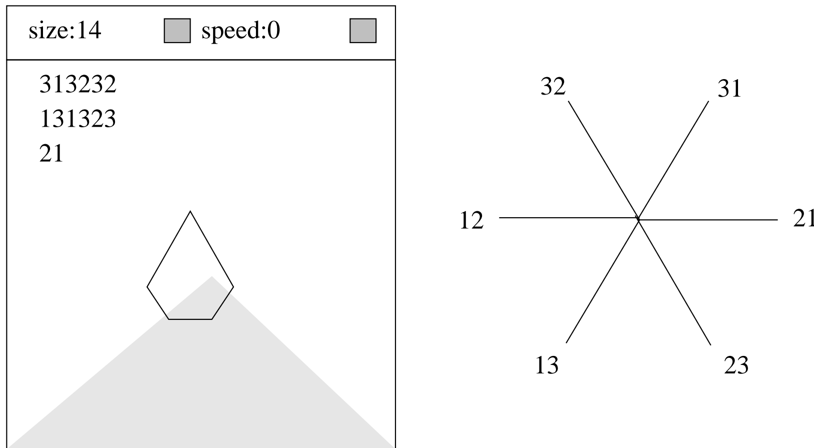

Figure 2.4

The polygonal curve drawn in the window is a graphical representation of W, defined as follows. We break W up into couplets, as

$$W = 31.32.42.13.13.23.21$$

We assign to each couplet a 6th root of unity, according to the scheme shown in the right hand side of Figure 2.4. This gives us 7 complex numbers $z_1, ..., z_7$. Our curve has vertices $z_1, z_1 + z_2, z_1 + z_2 + z_3, ...$ The fact that the curve is closed is equivalent to the fact that W is a *stable* periodic orbit. This is to say that Tile(W) is an open set of parameter space. We will explain the stability condition in detail in §3. For us, the polygonal curve is simply another way of representing the word W. When W is long, the polygonal representation has many advantages over a long string of digits.

The *speed* button lets you change the rate at which the polygonal path is drawn. The higher the number, the slower the drawing. This feature is useful when the polygonal path is not embedded.

The button in the upper right hand corner lets you scroll the word. This feature is useful when the word is too long to be completely displayed in the window.

2.5 The Plotting Console and Color Selector

The left side of Figure 2.5 shows a picture of the plotting console when McBilliards is ready to plot an orbit tile.

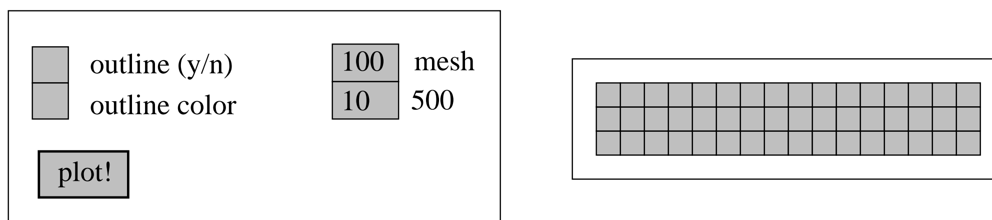

Figure 2.5

The right side of Figure 2.5 shows the color selector. This is a cluster of 3×16 squares. The color selector has its own window, at the very top left of McBilliards. One selects the red/green/blue values of the color and then the color selector window changes into the selected color. The color selector is also used to select colors for other parts of McBilliards−e.g. the labels one can add to the plots.)

When McBilliards is ready to plot, it has two pieces of information: the word W and a point $(x,y) \in \text{Tile}(W)$. McBilliards then keeps track of an angle θ, beginning with $\theta = 0$. McBilliards has 4 different plot options:

- The basic plot: McBilliards finds the intersection of the ray R_θ with the boundary of Tile(W). This intersection point is plotted and then θ is incremented. Here R_θ is the angle that the ray makes with the positive x-axis. If the number of data points is N then θ is incremented by $2\pi/N$ and a total of N points are used to plot the tile. The numbered buttons let the user change the value of N. In the example, $N = 500$. The higher the value of N, the sharper the plots.

- newton plot: McBilliards finds the vertices of the orbit tile using Newton's method, and then plots the edges between the vertices, again using Newton's method. This plot option is more precise than the basic plot, but still potentially has some bugs in it.

- convex hull: McBilliards finds the vertices by Newton's method and then takes the polygon spanned by these vertices. In practice this polygon is always convex, and hence coincides with the convex hull of the vertices. The convex hull of the vertices usually contains the tile as a proper subset.

- inner hull: After computing the polygon spanned by the (say) N vertices, McBilliards inserts a new vertex near the center of each edge, producing a $2N$-gon which seems always to be a proper subset of the tile.

Typically Tile(W) is nearly a convex polygon, and all the methods above produce nearly identical shapes.

The plotting console also has a button which lets the user decide whether or not the plotted tile has an outline or not. Finally, the plotting console has a second button which lets the user decide on the outline color.

2.6 The Unfolding Window

Given a triangle $T = T(x,y)$ and an orbit type $W = (a_1, ..., a_{2k})$ we define a sequence $T_1, ..., T_{2k}$ of triangles, as follows: We set $T_0 = T$ and then inductively define T_{j+1} so that T_{j+1} and T_j are related by a label-respecting reflection through the a_jth edge of T_j. When we are done we discard T_0.

We set $U(W,T) = \{T_j\}$ and call $U(W,T)$ the *unfolding* of the pair (W,T). Figure 2.6 shows an example corresponding to the word W from §2.3.

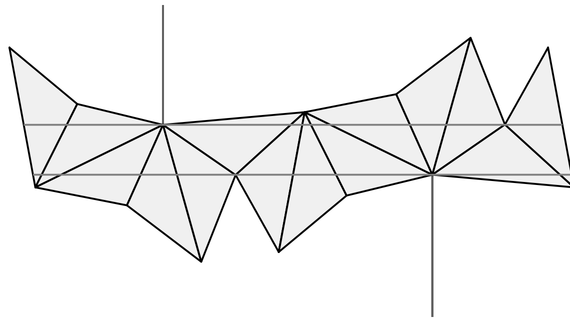

Figure 2.6

The unfolding window plots the unfolding $U(W,T)$ whenever W and T are given. Figure 2.6 shows an example corresponding to the word W from §2.3. For instance, if the user clicks the middle mouse button on a point of Tile(W) then the unfolding $U(W,T')$ is drawn, where T' is the triangle corresponding to the clicked point.

We shall only be interested in pairs (W,T) such that the first and last edges of $U(W,T)$ are parallel. The words W which have this property for all triangles are precisely the stable words discussed in §2.3. In the stable situation we rotate the picture so that the axis of the translation carrying the first side to the last side is horizontal, as in Figure 2.6.

In Figure 2.6 there is a horizontal strip S which separates the "top" vertices of $U(W,T)$ from the the "bottom" vertices. The boundary of S contains some of the top vertices and some of the bottom vertices. These vertices are pointed out by the vertical line segments.

There is a unique label-preserving isometry I_j from T_j to the original triangle T. The union

$$\bigcup_{j=1}^{2k} I_j(L' \cap T_j) \tag{2}$$

is a periodic billiard path in T. The path closes up because the first and last sides are parallel and the isometries match up the points where L' cuts these edges.

As for the plotting window (and the word display window) the graphical objects in the window can be scaled using the left and right mouse buttons.

The color scheme for the unfolding—e.g. the background color or the triangle color—can be changed using the controls in the small window to the right of the unfolding window. We will explain this below, in detail.

2.7 Button 2 Controls

The *Button 2 Controls* determine the behavior of the middle mouse button when it is clicked on parts of McBilliards. Three of the buttons have to do with scaling and scrolling. The remaining buttons have to do with the modofication of the plotted orbit tiles.

Once a group of tiles is drawn, they can be modified in various ways. Each tile can be

- simply recognized. In the "normal" mode of function, clicking a tile simply focuses McBilliards' attention on this tile. For instance, the word corresponding to the tile is drawn and the unfolding relative to the word and the selected triangle is drawn.
- deleted;
- recolored;
- raised relative to the other tiles;
- lowered relative to the other tiles;
- cycled—that is, the unfolding for the word is replaced by the unfolding for the left rotation of the word.
- recentered. When the tile is initially drawn, some center point (x, y) is used. This center point can be changed.

Each of these options is executed by clicking the middle button on a point of the tile. The tile console window lets the user set the action of the middle mouse button—i.e. select which option obtains when the middle mouse button is clicked.

There is one additional option we have included, which have called *slop*. This option tells McBilliards to draw the unfolding $U(W, T)$ even when the selected triangle T does not correspond to a point of Tile(W). On other words, the unfolding feature is allowed to "slop over" the edges of the tile. The slop feature is useful for presenting a more global view of the unfoldings associated to a word.

2.8 The Memory Console

Figure 2.8 shows the memory console.

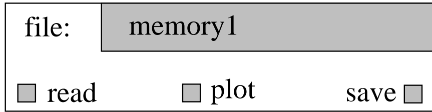

Figure 2.8

When the user clicks on the strip at top right, McBilliards focuses the keyboard output to this strip. The user can then select a filename. Many but not all of the keyboard symbols are available.

A memory file contains a list of triples (W_j, p_j, C_j), where $j = 1, ..., n$. Here

- W_j is an orbit type of a tile.
- $p_j \in \text{Tile}(W_j)$ is a distinguished point.
- C_j is the color of the tile.

When the *read* button is clicked, the list of triples is "stored" in the little boxes from the search console, just as if the orbits were found by a search. The stored tiles can then be plotted just as above.

When the *plot* button is clicked, the list of triples is stored and plotted. Unfortunately, McBilliards cannot be interrupted while it is plotting. So, if the memory file is big and the computer is slow, the user can be in for a long wait. (We hope to add an "interrupt" feature soon.) It is usually a good practice to read the file first, to see how many triples it contains.

When the *save* button is clicked, the tiles currently plotted in the parameter window are saved into the file. By this we mean that each tile is assigned a triple, which captures the (word,center,color) of the tile. McBilliards will write over an existing file, but will save the original file as *file.bak*. However, if the save button is clicked twice, then all information about the original file is lost. In other words, the *save* button should be used with care.

2.9 Pop Up

The region marked Pop Up in Figure 2.1 has two buttons in it:

- The *object console* button calls up a pop-up window which lets the user recolor, raise, or lower various markings in the parameter window, the unfolding window, and the word display window.

- The *welcome* button calls a pop up window which gives instructions to McBilliards.

Here are the objects in the parameter window which can be modified using the object console:

- The background color.

- A grid, which indicates the points in parameter space corresponding to triangles, one of whose angles has the form $\pi/2n$, where $n = 2, 3, 4....$

- A diagonal line which comprises the points in the parameter space corresponding to right triangles.

- A small crosshairs, which is moved around by rational imcrements from the rational overlay console.

- A small square cursor which indicates the point clicked by the user.

Here are the objects in the unfolding window which can be modified using the object console:

- The background color.

- The triangles in the unfolding.

- The edges of the triangles.

- The horizontal lines which comprise the corridor for the strip.

- The vertical lines which mark out the important vertices.

Here are the objects in the word display window which can be modified using the object console:

- The background color

- The graphical display of the word

- The digits of the word
- The big triangle which indicates the selected point in parameter space.
- The edges of the big triangle.

The object console has one last button which let's you raise, lower, and recolor the labels you can produce using the labelling window.

2.10 Postscript Window

The postscript window lets the user save pictures of each of the three main windows into postscript files:

- The parameter window.
- The unfolding window
- The plotting window.

The user can select the bounding box for the postscript file, as well as the scale factor.

2.11 The Labelling Console

The small window marked *labels* lets you add labels to the parameter or unfolding windows and then delete them. When you click on the window, McBilliards focuses the attention of the keyboard to the strip. The user can then enter a label. As for the entering of file names, many but not all keys are available.

2.12 The Rational Overlay Console

Figure 2.12 shows a picture of the rational overlay console. This console allows the user to locate specific rational points in the parameter window, by moving a crosshairs around in rational jumps. The cluster of buttons on in the upper right hand corner allows the user to move a crosshairs around the parameter window. The button in the northwest corner moves the cross in the northwest direction, and so forth. The cross moves by rational jumps.

The *step* button controls the size of the jumps. When the *dyadic* option is selected, the jumps go by way of dyadic fractions. In this case, a stepsize of N causes the jumps to be of size 2^{-N}.

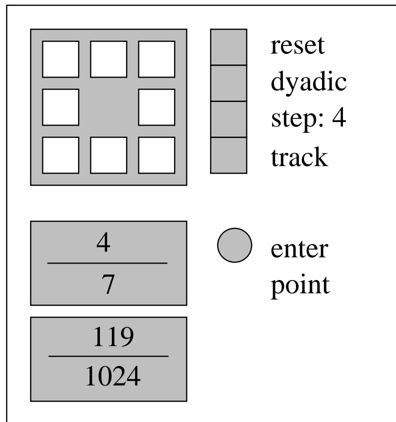

Figure 2.12

When the *Farey* option is selected, the jumps move from one level-N Farey fraction to the next one. The first few levels of Farey fractions are:

$$\frac{0}{1}, \quad \frac{1}{1}$$

$$\frac{0}{1}, \quad \frac{1}{2}, \quad \frac{1}{1}.$$

$$\frac{0}{1}, \quad \frac{1}{3}, \quad \frac{2}{3} \quad \frac{1}{1}.$$

$$\frac{0}{1}, \quad \frac{1}{4}, \quad \frac{1}{2}, \quad \frac{3}{4}, \quad \frac{1}{1}.$$

In general, level N is obtained from level $N-1$ by using the Farey addition law on consecutive entries, similar to Pascal's triangle. Every rational number appears at some level. The level of p/q is the number of steps in the Euclidean algorithm applied to (p, q). The *Farey* option is an efficient way of accessing all rational numbers.

When activated, the *track* button lets the user control the location of the crosshairs by directly clicking on the parameter window. McBilliards then picks the best rational approximation to the selected point, according to the stepsize.

The two rectangles at lower left display the coordinates of the cross. The *enter point* button lets the user select precisely this point on the parameter window.

2.13 The Triangle Entry Console

Figure 2.13 shows a picture of the triangle entry console. This console lets the user triangule portions of the parameter window using triangles of rational coordinates.

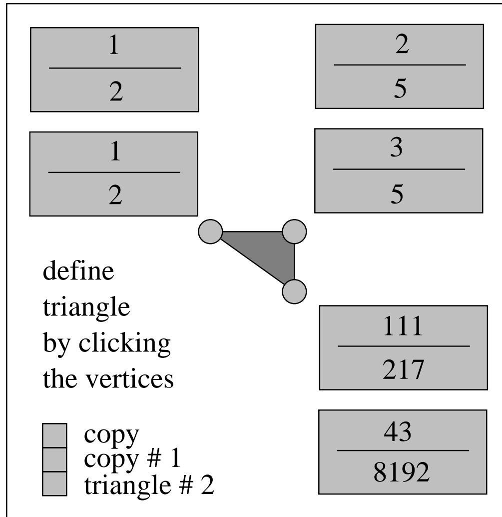

Figure 2.13

The rational coordinates in question are first selected using the rational overlay console. Once the user has selected a rational coordinate, they can click on one of the three vertices of the central triangle in the triangle entry console. Doing so causes McBilliards to assign this rational number to the relevent vertex of a triangle. When all three vertices have been determined, the triangle is plotted.

The *triangle* button lets the user tell McBilliards which triangle to determine with the current mouse clicks. In this way the user can define a triangulation of the parameter window by triangular regions with rational vertices. The *copy* button lets the user copy one triangle to another. This feature is useful in creating triangulations, since adjacent triangles in the triangulation will have 2 vertices in common. The *copy #* button tells McBilliards which triangle to copy onto the current triangle.

As for the rational overlay console, the big rectangles display the coordinates of the vertices.

Once a triangle is plotted, it behaves somewhat like a tile. The user can recolor, raise and lower the triangle by setting the options in the tile console and then clicking on the triangle with the middle mouse button.

3 The Basic Search Algorithm

The purpose of this chapter is to describe the initial search algorithm used by McBilliards. In §4 we will describe the improved version.

3.1 Balanced Words and Orbit Tiles

We remind the reader that unfoldings were discussed in §2.6. As in §2.6, the object $U(W,T)$ refers to the unfolding of a triangle T with respect to a word W. (Here *word* and *orbit type* are used synonymously.)

We say that the word W is *balanced* if the first and last sides of $U(W,T)$ are parallel for every triangle T. To understand balancing combinatorially, we break W into couplets: $W = w_1w_2, w_3w_4, w_5w_6, ...$ We let N_{ij} denote the number of occurances of the couplet ij. It is an easy exercise to show that W is balanced if and only if $N_{i,i+i} - N_{i,i-1}$ is independent of i. This definition is the same as saying that the polygonal path of §2.4 is closed. For the remainder of this section we assume that W is balanced.

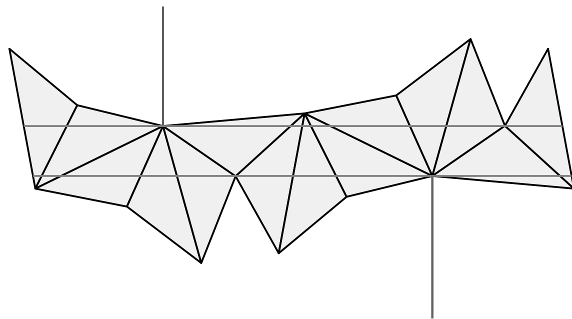

Figure 2.6

When W is balanced it never happens in our program that the first and last edges of $U(W,T)$ lie on the same line. See the end of §3.3. Henceforth, we ignore degeneracy. We say that W has a *lane* if we can arrange W as in Figure 2.6, so that some nonempty strip separates the a vertices from the b vertices. As we saw above, T has a stable periodic orbit of type $[W]$ provided that W is balanced and $U(W,T)$ has a lane. The orbit is stable because $U(W,T')$ also has a lane for T' sufficiently close to T.

If $U(W,T)$ has a lane we define Tile(W) to be the set of all triangles T' such that $U(W,T')$ has a lane. This definition coincides with the definition given in §2.2. Then Tile(W) is an open subset of the parameter space of triangles.

3.2 The Weak Test

Figure 3.2 shows a (roughly) drawn picture of an unfolding $U(W,T)$ where $W = 2313213$ and T is some triangle.

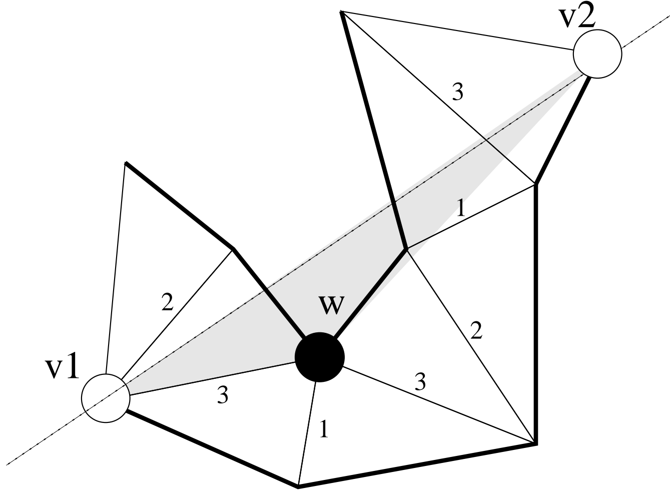

Figure 3.2

The word W is not balanced, and so it does not make sense to talk about Tile(W). However, there certainly are balanced words $\widehat{W}$ which contain W as a subword. Note that $U(\widehat{W},T)$ can never contain a lane, because the vertex w lies on the wrong side of the line $\overline{v_1,v_2}$. Thus Tile($\widehat{W},T$) is empty for any word $\widehat{W}$ which contains W. Computationally we test this condition by computing the sign of the signed area of the triangle $\Delta(v_1,v_2,w)$.

Note that $U(W,T)$ has two special paths, which have have drawn in bold. When $U(W,T)$ is embedded, these paths are the boundary components of $U(W,T)$. The *weak test* is a computational test we apply to the pair (W,T). It goes as follows:

- Let v_1 and v_2 be the two endpoints of one of the two special paths.
- Check the sign of the signed area of the triangle $\Delta(v_1,v_2,w)$ for every vertex w on the other path.
- Reverse the roles of the two paths and repeat.

If we ever get the wrong sign, we way that (W,T) fails the weak test. Thus, if (W,T) fails the weak test then Tile($\widehat{W},T$) $= \phi$ for any balanced word $\widehat{W}$ which contains W as a subword.

3.3 The Strong Test

The strong test is what we use to decide W represents an orbit type of a periodic billiard path on T. First of all, the strong test checks that W is balanced. If W is not balanced then W fails the strong test. Henceforth assume that W is balanced.

We distinguish one of the two special paths of $U(T,W)$ and let v_1, v_2 be the two endpoints of this path. Let $v'_1, ..., v'_n$ be the remaining vertices on this path. We define

$$m' = \max \text{Area}(\Delta(v_1, v_2, v'_j)) \tag{3}$$

Next, we let $w_1, ..., w_m$ denote the vertices on the other special path of $U(T,W)$. We define

$$m = \min \text{Area}(\Delta(v_1, v_2, w_j)) \tag{4}$$

If we rotate the picture so that the translation taking the first edge of $U(W,T)$ to the last edge is horizontal, then the areas we have computed are all proportional to the y-coordinates of the vertices. The constant of proportionality is independent of vertex. Thus, $U(W,T)$ has a lane iff $m' < m$. The strong test computes all the areas and then tests if $m' < m$. Thus (W,T) passes the strong test iff W is the orbit type of a periodic billiard path on T.

Remark: If $U(W,T)$ was such that the first and last edges were collinear, then we could get "division by zero" errors in our program and it would halt. In practice this never happens, so we don't have to worry about this wierd degenerate situation.

3.4 The Lexi Test

The lexi-test is a test which is applied to words W. A word W fails the lexi-text iff W contains a subword W' which comes before W in the lexicographic ordering. For instance $W = 131213$ fails the lexi-test because $W' = 1213$ comes before W in the lexicographic order. Our search algorithm throws out words which fail the lexi-test because they are redundant.

By throwing out such words, McBilliards does not find all possible (even length) periodic orbits up to certain depth. Rather, McBilliards finds all possible equivalence classes of orbit types, where two types are equivalent if they are rotations of each other.

3.5 The Algorithm

The input to the search is an integer D, which defines the depth of the search, and a triangle T. The search algorithm begins with a list of words called CONTENDERS and a second list of words called WINNERS. Initially CONTENDERS consists of the single word 12 and WINNERS is empty. The algorithm proceeds until CONTENDERS is the empty list, then halts. At this point, WINNERS is the list of even length balanced words of length less or equal to D which are orbit types of periodic billiar paths in T.

1. If CONTENDERS $\neq \emptyset$ let W be the first word on CONTENDERS.
2. If W fails the lexi test or the weak test, delete W from CONTENDERS and return to Step 1.
3. If W passes the strong test append W to WINNERS.
4. Let $L = \text{Length}(W)$. If $L \leq D-2$ then delete W from CONTENDERS and prepend to CONTENDERS the 4 words W_1, W_2, W_3, W_4 which have length $L+2$ and contain W as its initial word. Go to Step 1.

As an example for the last step: if $W = 12$ then the 4 words are 1212, 1213, 1231, and 1232.

Our algorithm implements a depth first search through the tree of words, pruning off any branches whose initial node fails the weak test or lexi test.

Remark: The right angled search works just as the balanced search, except that the balance condition is different. Here we weaken it to the condition that the difference $N_{j3} - N_{3j}$ is independent of $j = 1, 2$ and $N_{12} - N_{21}$ is even.

4 The Slalom Search Algorithm

Our algorithm has the same overall structure as the one described in §3, but uses a different test, the *slalom test*, in place of the weak test. Our slalom test is actually a variant on Graham's scan for convex hulls of finite point sets in the plane. To orient the reader, we will first discuss the situation for planar convex hulls.

4.1 Planar Convex Hulls

Here we will discuss an algorithm for finding the convex hull of a finite list of points in the plane. The algorithm we describe is related to Graham's scan, and works by induction. Here we will explain the induction step. Let H_k denote the convex hull of the first k points, $p_1, ..., p_k$. We would like to find H_{k+1}. We assume that the points $p_1, ..., p_k$ come in cyclic order on H_k. In our example, shown in Figure 4.1, we have $k = 11$.

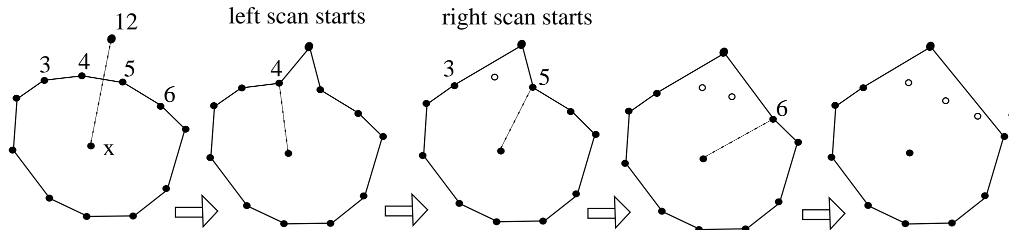

Figure 4.1

If $p_{k+1} \in H_k$ then $H_{k+1} = H_k$. Suppose that $p_{k+1} \notin H_k$. Let x be a generic point in the interior of H_k. Then there is some index i such that the line $\overline{xp}$ crosses the edge $\overline{p_i p_{i+1}}$ in the interior point. We have $i = 4$ for our example. We create a provisional version of H_{k+1} as follows: We replace e_i by the two edges $\overline{p_i p_{k+1}}$ and $\overline{p_i p_{k+1}}$.

If H_{k+1} is convex we are done. Otherwise, we proceed as follows. If H_{k+1} is not locally convex at p_i we modify H_{k+1} by replacing $\overline{p_{i-1}p_i} \cup p_i p_{i+1}$ with the edge $\overline{p_{i-1}p_{i+1}}$. (This has the effect of deleting p_i from the list of relevant vertices.) Next we move left to p_{i-1} and repeat the procedure. We stop "scanning to the left" when we reach a vertex at which H_{k+1} is locally convex. In our example, the vertex is p_3. Following this, we "scan to the right", starting with p_{i+1}, performing the same procedure. The right scan stops at p_7 in our example. When we are finished modifying H_{k+1}, it is the convex hull of the first $k + 1$ points.

4.2 The Slalom Test and Convex Hulls

Now we explain the connection between convex hulls and the slalom test. Like the weak test, the slalom test decides if there is a line segment which starts in the first triangle of an unfolding, then hits every edge we unfold along, and ends in the the last triangle of an unfolding. Figure 4.1 displays an unfolding $U(W,T)$, for some triangle T and for the word $W = 13123$. This unfolding should pass the weak test because of the existence of the line segment L.

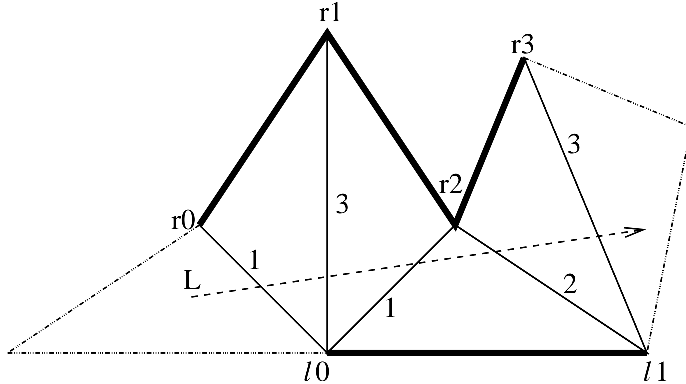

Figure 4.2

As discussed in section 3.2, there are two special paths which our line segment must avoid. In our picture they are darkened and labeled l_0l_1 and $r_0r_1r_2r_3$. We call our test the slalom test because it tests to see if there is a line segment L which "runs the slalom course" set up by these vertices, passing to the left of the vertices l_0l_1 and to the right of the vertices $r_0r_1r_2r_3$. We call the vertices l_0l_1 left vertices and the vertices $r_0r_1r_2r_3$ right vertices.

We say a line L *runs the slalom course* given by $U(W,T)$, if it enters the first triangle of our unfolding, then passes through edges as given by W, and finally exits through the last triangle of our unfolding.

Now, take an unfolding $U(W,T)$ for which there is an line L which runs the slalom course as above. Thus the set of all lines S which run the slalom course given by $U(W,T)$ is non-empty. Define the set

$$S_P = \bigcup_{L \in S} L \subset \boldsymbol{R}^2$$

The compliment of S_P is convex in a sense we describe below. Figure 4.3 displays the region S_P for the unfolding shown in figure 4.2.

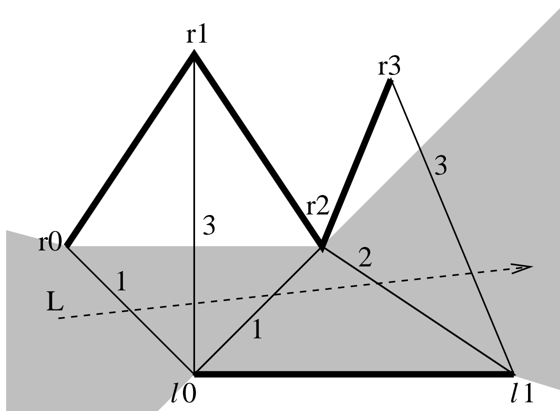

Figure 4.3

Consider the usual embedding of Euclidean plane inside the projective plane, $\boldsymbol{R}^2 \subset \boldsymbol{R}P^2$, as an affine patch. S_P naturally a subset of $\boldsymbol{R}P^2$. We say that a subset $C \subset \boldsymbol{R}P^2$ is *convex relative to the line* L if it is a convex subset of the affine plane $\boldsymbol{R}P^2 \setminus L$. We will call $C \subset \boldsymbol{R}P^2$ relatively convex if there is a choice of a line L for which C is convex relative to L.

Here is the connection between “running the slalom course” and convex hulls:

Proposition 4.1 *If $L \in S$, then $\boldsymbol{R}P^2 - S_P$ is convex relative to L.*

The boundary of ∂S_P is a polygon in $\boldsymbol{R}P^2$ whose vertex set is a subset of the vertices in our unfolding. The vertices in ∂S_P have a natural cyclic ordering and we define $C(W,T)$ to be the cyclically ordered set of these vertices. In our example we have $C(W,T) = \{l_0 l_1 r_0 r_2\}$.

We want to know when such a cyclic list of points determines a polygon which is relatively convex. It is helpful to consider first the analogous question for convex polyhons. For $v_1, v_2, v_3 \in \boldsymbol{R}^2$ let $SA(v_1, v_2, v_3) \in \{-1, 0, 1\}$ to be the sign of the area of the triangle v_1, v_2, v_3.

Proposition 4.2 *A cyclically ordered set of points $v_1, \ldots, v_n$ defines a convex polygon in the Euclidean plane if and only if*

1. *The polygon with successive vertices $v_1, \ldots, v_n$ is embedded.*

2. *$SA(v_i, v_{i+1}, v_{i+2})$ is independent of i.*

Here (and below) we take indices mod n.

We want a similar fact for our case. One problem is that a cyclically ordered set of points in $\boldsymbol{R}P^2$ does not determine a unique polygon, because there are two choices for an edge joining consecutive vertices. However, we have additional information about our particular collection of vertices; they come from an unfolding. We use this information to define a polygon $(v_1, ..., v_n)$ whose vertex set is $v_1, ..., v_n$. First, we define the function

$$L : \{v_1, \dots, v_n\} \to \{-1, 1\} : \begin{cases} -1 & \text{if } v_i \text{ is a left vertex} \\ 1 & \text{if } v_i \text{ is a right vertex} \end{cases}$$

Our points $v_1, \dots, v_n$ lie in a Euclidean plane embedded into the projective plane. Given two vertices, a line segment drawn between them is called finite if it is entirely contained in our Euclidean plane and infinite otherwise. We define our polygon $(v_1, \dots, v_n)$ by joining consecutive vertices by an edge which is finite iff $L(v_i) = L(v_{i+1})$. Thus our cyclically ordered set $C(W,T)$ and our labeling L determine the polygon uniquely. The polygon $(v_1, \dots v_n)$ in turn determines S, the set of lines which run the slalom course given by $U(W,T)$.

Proposition 4.3 $(v_1, ..., v_n)$ *is convex relative to some line iff*

1. $(v_1, ..., v_n)$ *is embedded.*
2. $SA(v_{i-1}, v_i, v_{i+1}) = L(v_{i-1})L(v_i)L(v_{i+1})$ *for all* i.

4.3 The Algorithm

Our algorithm proceeds inductively. The base case is a word of length 2. A word of length 2 should always pass the weak/slalom test. Here the picture always looks the same, regardless of the triangle. Our cyclically ordered set $C(W,T)$ consists of three points, and our labeling L is such that exactly two labels match. See figure 4.4.

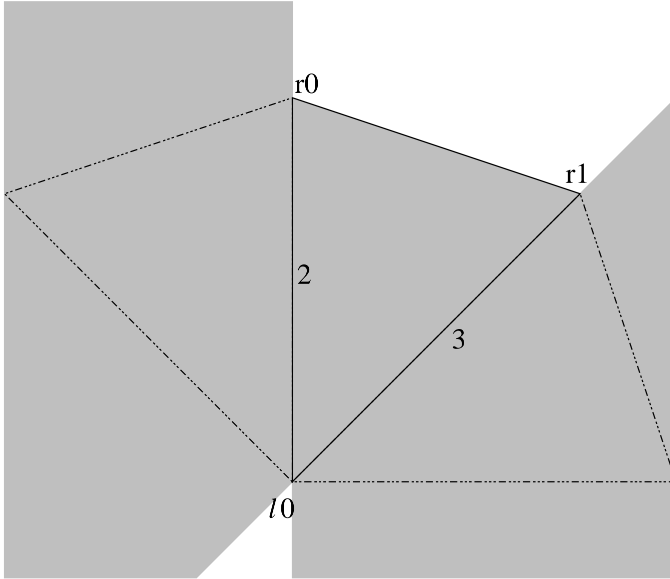

Figure 4.4

Assuming that there is a line which runs the slalom course set by $U(T, W)$ we pass to our algorithm the following information:

- T and W;
- the cyclically ordered set $C(W, T)$;
- the labeling $L : C(W, T) \to \{-1, 1\}$;
- an additional letter $l \in \{1, 2, 3\}$.

Our algorithm will tell us if there is a line which runs the slalom course given by $U(W', T)$, where $W' = Wl$ is the word W with l appended. Further, if there is such a line, we return with $C(W', T)$ and the appropriate labeling L'. If there is no such line, the unfolding $U(W', T)$ fails the slalom test.

We will continue the example set out in figures 4.2 and 4.3. In this case we have $W = 13123$ and $C(W, T) = \{l_0 l_1 r_0 r_2\}$. We will consider the case $l = 1$, so that $W' = 131231$. We look at the unfolding $U(W', T)$, depicted below in figure 4.5. Note that the shaded polygon in Figure 4.5 is the set of lines which run the slalom course for $U(W, T)$–i.e. $C(W, T)$.

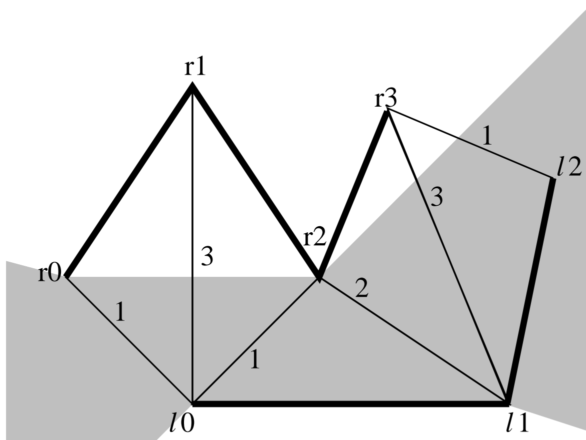

Figure 4.5

In general, one of the special paths of $U(W',T)$ will have an additional vertex v which extends the same special path for $U(W,T)$. Call this vertex the *extending vertex*. In the example at hand, the extending vertex is $v = l_2$. A line runs the slalom course for $U(W',T)$ iff it runs the slalom course for $U(W,T)$ and passes to the left of $v = l_2$. (Were v to extend the right special path, "left" would be replaced by "right" in the previous sentence.) Computing $C(W',T)$ from $C(W,T)$ and v is analogous to Graham's scan, discussed in §4.1. We will present our algorithm below.

For the algorithm, we define a vertex v_k in a labeled cyclically ordered set of points to be *good* if it satisfies the second criterion of proposition 4.3−i.e. $SA(v_{k-1}, v_k, v_{k+1}) = L(v_{k-1})L(v_k)L(v_{k+1})$. We let v be the extending vertex and then run the following algorithm.

- If $L(v) = -1$ (respectively $L(v) = 1$) insert v after the last left (respectively right) vertex in $C = C(W,T)$.
- If v is not good, then $U(W',T)$ passes the slalom test, v is irrelevant, and $C(W',T) = C(W,T)$. Otherwise...
- Repeat the following until told to BREAK:
  - Set w be the vertex which comes immediately after v in the cyclic ordering.
  - If $L(v) = L(w)$ then $U(W',T)$ fails the slalom test. (*Note:* In this situation, we have eliminated all left vertices or all right vertices, because v was the last left or the last right vertex.)

  - if w is good then BREAK, otherwise delete w and continue.

- Repeat the following until told to BREAK:
  - Let w be the vertex which comes immediately before v in the cyclic ordering.
  - if w is good then BREAK, otherwise delete w and continue.
- $U(W', T)$ passes the slalom test and our cyclic set of points satisfies proposition 4.3. We return with our modified cyclic set and its labeling.

Now we step through our algorithm for example.

- We add l_2 to our cyclic set of points. Now, $C = \{l_0 l_1 l_2 r_0 r_2\}$.
- l_2 is good. This is because $L(l_1)L(l_2)L(r_0) = (-1) \cdot (-1) \cdot 1 = 1$. Also, $SA(l_1, l_2, r_0) = 1$. Another way to say this is that if we walk from l_1 toward l_2 in the Euclidean plane, then turn and walk toward r_0 we turn left. This left turn is equivalent to positive signed area.
- We enter the first loop.
  - Set $w = r_0$.
  - $L(w) = 1$ while $L(v) = -1$, so we continue on.
  - $w = r_0$ is not good. This is because $L(l_2)L(r_0)L(r_2) = (-1) \cdot 1 \cdot 1 = -1$ and $SA(l_2, r_0, r_2) = 1$. We turn left as we walk from l_2 toward r_0 and then toward r_2. We delete r_0 from our cyclic list of points, so that now $C = \{l_0 l_1 l_2 r_2\}$. We continue the loop.
  - Set $w = r_2$.
  - $L(w) = 1$ while $L(v) = -1$, so we continue on.
  - $w = r_2$ is good. $L(l_2)L(r_2)L(l_0) = (-1) \cdot 1 \cdot (-1) = 1$ and $SA(l_2, r_2, l_0) = 1$. We turn left in our t rip from l_2 to r_2 to l_0. Thus we break this loop.
- We enter the second loop.
  - Set $w = l_1$.

- $w = l_1$ is not good. $L(l_0)L(l_1)L(l_2) = -1$ and $SA(l_0, l_1, l_2) = 1$. We delete $w = l_1$ from our cyclic list, obtaining $C = \{l_0 l_2 r_2\}$. We continue the loop.
- Set $w = l_0$.
- $w = l_0$ is good. $L(r_0)L(l_0)L(l_2) = 1$ and $SA(r_0, l_0, l_2) = 1$. We break the loop.

- $U(W', T)$ passes the slalom test, and $C(W', T) = \{l_0 l_2 r_2\}$.

The data returned by our algorithm is displayed in figure 4.6.

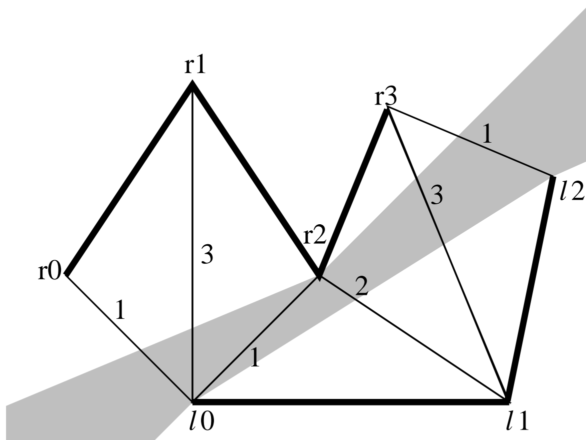

Figure 4.6

As another example, consider the unfolding for the triangle shown in Figure 4.6 and the word $W = 312$. (This unfolding is a subset of the one shown in Figure 4.6.) Assume our algorithm is given $C(W, T) = \{l_0 l_1 r_1\}$ with the usual labeling. We wish to apply the slalom test with additional letter $l = 1$. Thus we consider the unfolding $U(W', T)$ where $W' = Wl = 3121$. We introduce the new point l_2, as shown in the right side of figure 4.7. We note l_2 should be a left point, and so $L(l_2) = -1$. We pass this information to the algorithm, and then run it:

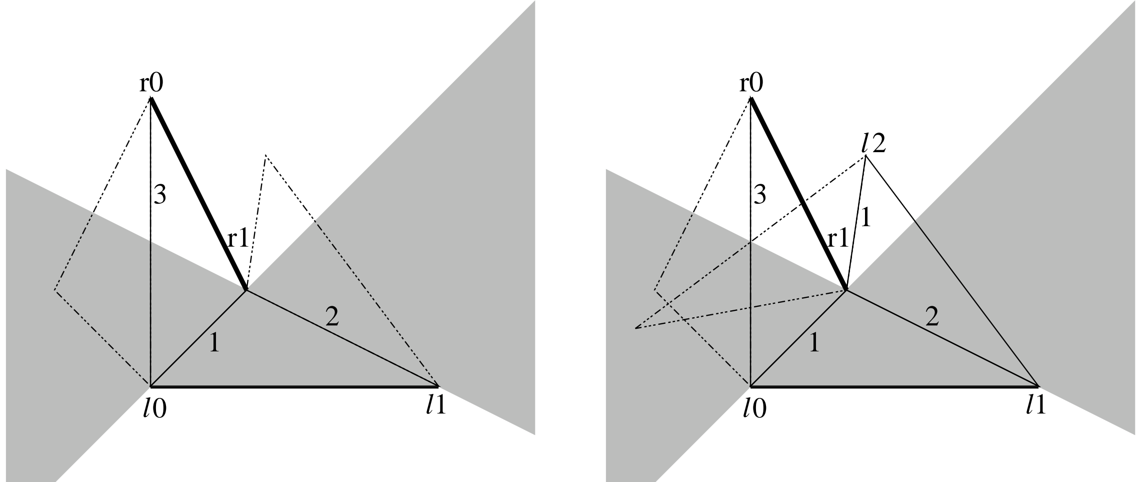

Figure 4.7

- We add l_2 to our cyclicset of points. Now, $C = \{l_0 l_1 l_2 r_1\}$.
- l_2 is good, as $L(l_1)L(l_2)L(r_1) = (-1) \cdot (-1) \cdot 1 = 1 = SA(l_1, l_2, r_1)$.
- We enter the first loop.
  - Set $w = r_1$.
  - $L(w) = 1$ while $L(v) = -1$, so we continue on.
  - $w = r_1$ is not good, as $L(l_2)L(r_1)L(l_0) = 1$ and $SA(l_2, r_1, r_2) = -1$. Therefore we delete r_1 from our cyclic list of points giving us the list $C = \{l_0 l_1 l_2\}$. We continue with the loop.
  - Set $w = l_0$. (Recall our list is cyclic, so l_0 now comes after l_2)
  - Now $L(w) = L(l_0) = -1$ and $L(v) = L(l_2) = -1$. Thus this example fails the slalom test. No line runs the slalom course $U(W', T)$.

4.4 Speed

Our algorithm should run very fast. Suppose we wish to run our algorithm on a triangle T and a word W of length n. Set W_k to be the word consisting of the initial k letters of W. To apply the slalom test to W we need to run the algorithm first on our base case $U(W_3, T)$, record the result and run the algorithm on $U(W_4, T)$, etc., all the way to $U(W_{n-1}, T)$. Only then can we apply the slalom test to $U(W, T)$. But, all of this can be done in $O(n)$

time. The computational time of an algorithm is generally accumulated in loops. But, each step of our loops results either in deleting a vertex from our cyclically ordered set C or terminating the loop. We now break our program into pieces. In each application of the slalom test, we do some initial constant time setup including adding a vertex to our cyclically ordered set C, we delete some vertices from the cyclically ordered set, and we terminate the two loops. The setup and the loop terminations are certainly constant time. Finally, in all the cumulative applications of the algorithm on $U(W_3, T) \ldots U(W_n, T)$, we only add approximately n vertices to our cyclic set. Thus, we can delete at most n vertices as well. Thus, this whole sequence of applications of the slalom test runs in $O(n)$ time.

Remarks 4.4 *Our methods should slightly improve the strong test from §3 as well. We can run the same strong test, but vertices in our cyclically ordered set $C(W, T)$ are the only vertices that need to be checked for the strong test. Typically $C(W, T)$ is much smaller than the set of all vertices of an unfolding, so this should speed up the strong test as well.*

4.5 Signs of Areas of Triangles

In our original implementation of the slalom test, we encountered bizarre errors on some long searches. These errors stemmed from numerical errors in computing the signs of the areas of triangles. When we employ the usual computational methods−e.g. computing with doubles and rounding off arithmetic operations in the standard way−we find that the sign errors are sometimes inevitable. We need to use special computing methods to avoid the sign errors.

One solution is to use an arbitrary precision arithmetic package. Our code does not do this because these tools are generally not standard across platforms. We instead compute the sign of areas of triangles explicitly inside the program. Essentially, computation of the area of a triangle amounts to computing the determinant of a three by three matrix. Thus we must compute the sign of the sum of six terms, where each term is a product of (in our case) two doubles. We compute these terms explicitly without rounding. Then, we successively take the most significant pieces from the terms and add them up. If the sum has absolute value so big that adding the less significant pieces to it can not change its sign, then we are done. Otherwise, we continue adding less significant pieces.

5 References

[**CHK**] B. Cipra, R. Hanson, A. Kolan, *Periodic Trajectories in Right Angled Billiards*, Physical Review E **52** (1995) pp 2066-2071

[**GSV**], G.A Galperin, A. M. Stepin, Y. B. Vorobets, *Periodic Billiard Trajectories in Polygons*, Russian Math Surveys **46** (1991) pp. 204-205.

[**H**] F. Holt, *Periodic Reflecting Paths in Right Triangles*, Geometriae Dedicata **46** (1993) pp. 70-93

[**HH**], L Halbeisen and N Hungerbuhler, *On Periodic Billiard Trajectories in Obtuse Triangles*, SIAM Review **42.4** (2000) 657-670

[**M**] H. Masur, *Closed Trajectories of a Quadratic Differential and Applications to Billiards*, Duke Math Journal **53**, (1986) 307-313

[**O**] J. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wiley Professional Computing Series, 1994

[**S**] R. E. Schwartz, *Slightly Obtuse Triangles have Periodic Billiard Paths*, in preparation

[**T**] S. Troubetzkoy, *Periodic Billiard Orbits in Right Triangles*, preprint 2004.