3D Turtle Geometry: Artwork, Theory, Program Equivalence and Symmetry

Tom Verhoeff

Faculty of Mathematics and Computer Science
Eindhoven University of Technology
Den Dolech 2
5612 AZ Eindhoven, Netherlands
Email: T.Verhoeff@tue.nl

January 2009, Revised February, June, December 2009
(Submitted to IJART)

Abstract

We define a 3D variant of turtle graphics and present the theoretical foundations of 3D turtle geometry. This theory enables one to reason about open and closed 3D polygonal paths by means of algebraic calculations. In particular, we introduce several equivalence relations on turtle programs and theorems that define corresponding standard forms. We also express the relationship between the symmetries of a 3D polygonal path and the symmetries of a generating turtle program in a suitable standard form. Finally, we discuss software tool support for 3D turtle geometry. Along the way, we present some artworks designed through 3D turtle graphics. These artworks have never been described in the literature before.

Keywords: 3D turtle geometry, turtle programs, program equivalence, symmetry, mathematical art, technology

"Mathematics is the tool specially suited for
dealing with abstract concepts of any kind and
there is no limit to its power in this field."
Paul Dirac

1 Introduction

Seymour **?** invented *turtle gaphics* to make computer programming attractive for children. Turtle graphics provides a simple way of producing graphical output by a robot turtle, and later by a virtual on-screen turtle, controlled through the *Logo programming language*. It is based on self-relative operations, rather than absolute co-ordinates. At any moment, the turtle has a position and a heading (orientation). The turtle walks in the plane and can be instructed to

activate (lower) or deactivate (raise) its pen; to move forward or back a given distance; to rotate left or right through a given angle, about its centre. This way,

```
Repeat 5 [ Forward 100 Left 72 ]
```

Figure 1: *2D turtle after drawing a regular pentagon (left) through an ACS Logo program (right)*

many figures can be described through easy programs that control the operation of the turtle. For instance, a regular pentagon is described without messy co-ordinate calculations, through the elegant little Logo program in Figure **??**.

In Section **??**, we define a 3D variant of turtle graphics. Many interesting 3D polygonal paths can be described concisely by 3D turtle programs. Koos Verhoeff has been using such programs as a convenient notation for designing, defining and analysing various of his artworks. We present some examples in Section **??**. Other applications for 3D turtle graphics are in the areas of computer numerical controlled (CNC) tools, 3D printers, and robot motion planning.

The mathematical study of turtle graphics is knows as **turtle geometry**. 2D turtle geometry has enjoyed a thorough analysis in (**?**), including the generalisation to curved surfaces. However, we are not aware of any published theory on 3D turtle geometry. This is a pity, because in some ways the 3D variant is more challenging than the 2D original.

We state some elementary theorems of 3D turtle geometry in Section **??**, including the reduction to a standard form. This provides the mathematics behind the analysis of *mitre and fold joints for closed polygonal paths* in (**?**), and of *regular polygonal paths of constant torsion* in (**?**). Section **??** addresses the question of determining whether two programs are equivalent (in various senses). In particular, this also concerns the ways in which a program is equivalent to itself, that is, how its symmetries relate to the symmetries of the generated polygonal path.

In Section **??**, we discuss software tool support for 3D turtle geometry. Section **??** concludes the article.

2 Definitions for 3D turtle graphics

Our 3D variant of turtle graphics involves a flying turtle whose **state** is defined by its **position** and **attitude** in 3D space. The turtle's attitude is determined by its **heading** (a vector) and its **normal** (a vector perpendicular to the heading, defining the relative up direction). The plane that contains the heading and that is perpendicular to the normal, is called the turtle's **base plane**. Similar definitions are provided in (**?**, § 3.4), but they develop no theory.

By default, the turtle starts in the origin, heading along the positive x-axis and with the positive z-axis as normal (see Figure **??**, left), so that the (x, y)-plane is its base plane. For our purposes, the pen is always active. The flying turtle obeys these **basic commands** (see (**?**) for an interactive demonstration):

- *Move*(d): turtle moves distance d in the direction of the current heading, leaving an ink trace (negative d moves backward);
- *Turn*(φ): turtle turns (yaws) clockwise by angle φ about the current normal, changing the heading but not the normal (negative φ rolls counterclockwise);
- *Roll*(ψ): turtle rolls clockwise by angle ψ about the current heading, changing the normal but not the heading (negative ψ turns counterclockwise).

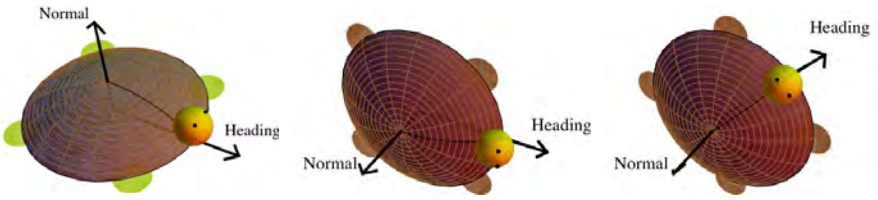


Figure 2: *3D flying turtle in initial state (left), after Roll*$(90°)$ *(middle), and after Roll*$(90°)$ *followed by Turn*$(45°)$ *(right)*

It is possible to include the command *Dive*(θ), to make the turtle dive (pitch) downward by angle θ, that is, turning the normal in the direction of the heading. However, this command is superfluous, because by suitably combining *Turn* and *Roll* commands, the turtle can head in any direction and produce any 3D path. In fact, we have

$$\mathit{Dive}(\theta) \quad \equiv \quad \mathit{Turn}(90°)\,;\,\mathit{Roll}(\theta)\,;\,\mathit{Turn}(-90°)\;. \tag{1}$$

Note that we use a semicolon ';' to indicate sequencing of commands, and $\equiv$ for equivalence of two programs. Program equivalence is defined formally in Section **??**.

For completeness' sake, we give an implementation of the three basic commands in terms of Cartesian co-ordinates. Let the turtle's state $s = \langle p, h, n\rangle$ consist of position $p = (p_x, p_y, p_z)$, heading $h = (h_x, h_y, h_z)$ and normal vector $n = (n_x, n_y, n_z)$, where heading and normal are unit vectors: $|h| = |n| = 1$. Define the turtle's *left unit vector* ℓ as the cross product of normal and heading:

$$\ell := n \times h = (n_y h_z - n_z h_y, n_z h_x - n_x h_z, n_x h_y - n_y h_x)\;. \tag{2}$$

The new state after each of the basic commands is given by

$$\begin{aligned}
\mathit{Move}(d)\langle p, h, n\rangle &= \langle p + d\,h, h, n\rangle\;, &(3)\\
\mathit{Turn}(\varphi)\langle p, h, n\rangle &= \langle p, h\cos\varphi + \ell\sin\varphi, n\rangle\;, &(4)\\
\mathit{Roll}(\psi)\langle p, h, n\rangle &= \langle p, h, n\cos\psi - \ell\sin\psi\rangle\;. &(5)
\end{aligned}$$

The commands *Move*, *Turn*, and *Roll* can be combined sequentially in many ways. A sequence of such commands describes a polygonal path in 3D space. For truly three-dimensional paths, all three commands are needed and it is useful to introduce an abbreviation for a common combination. For that purpose, we define $Segment(d, \psi, \varphi)$ by

$$Segment(d, \psi, \varphi) \quad := \quad Move(d)\,;\, Roll(\psi)\,;\, Turn(\varphi)\ , \tag{6}$$

where d, ψ, and φ are given parameter values. There are actually six permutations of these three commands. For closed paths, that is polygons, we can restrict ourselves to either of these six permutations as the single fundamental building block. Our choice for (**??**) is deliberate, but we will defer explanations to Section **??**. We finish this section with some informal definitions that apply to the artwork in the next section.

We call a turtle program, typically consisting of a sequence of *Segment* commands, **closed** when it returns the turtle to its initial position, and we call it **properly closed** when it returns the turtle to its initial *state*. That is, in order to be properly closed, not only should the turtle's final position equal its initial position, but also its final attitude must equal its initial attitude. Note that closed does not necessarily imply properly closed. Consider the program

$$Segment(1, 0, 120^\circ)\,;\, Segment(1, 0, 120^\circ)\,;\, Segment(1, 0, 120^\circ)\ . \tag{7}$$

Program (**??**) is properly closed and constructs an equilateral triangle in the (x, y)-plane. Now replace the last command $Segment(1, 0, 120^\circ)$ by $Segment(1, 90^\circ, 0)$. The program is then still closed, producing the same triangle. But it is no longer properly closed, because at the end the turtle's heading does not point along the x-axis and its normal does not point along the z-axis.

To avoid certain complications, we will often restrict ourselves to **simple** programs, where the turtle visits no point twice, except for ending in the initial position in the case of a closed program. A simple program generates a non-self-intersecting polygonal path.

An *internal edge* of a polygonal path is an edge that has both a predecessor and a successor. The **torsion angle** of an internal edge e in a directed polygonal path is defined as the directed dihedral angle between, on one hand, the plane spanned by edge e and its predecessor edge, and on the other hand, the plane spanned by e and its successor edge. Note that the torsion angle equals the roll angle ψ when the path is produced by a sequence of *Segment* commands. In the case of a closed path, this also holds for the first and last edge, provided that the program is properly closed.

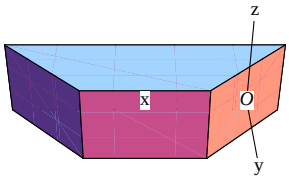
3 Artwork based on 3D turtle graphics

As a nice application of the definitions, we describe some artwork by Koos Verhoeff using 3D turtle graphics. One of the themes in his mathematical sculptures concerns non-self-intersecting 3D polygonal paths thickened to beams connected at the paths' vertices by mitre or fold joints (**?**). The cross section of the beam can be a line segment, in which case the beam is actually a strip, or it can be some polygon. The cross section is typically constant over the entire path. In this section we restrict ourselves to square and equi-triangular cross sections.

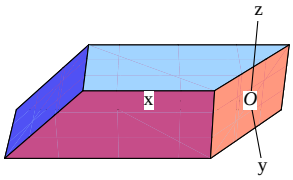
We use the following four abbreviations

$$\begin{aligned} T_d &:= \mathit{Segment}(d, 0, 90^\circ) \\ R_d &:= \mathit{Segment}(d, 90^\circ, 90^\circ) \\ L_d &:= \mathit{Segment}(d, -90^\circ, 90^\circ) \\ P_d &:= \mathit{Segment}(d, 180^\circ, 90^\circ) \end{aligned}$$

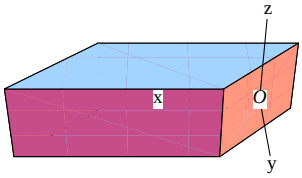
The names were chosen because they have a mnemonic value, when using bevelled square beams and mitre joints to construct the path (see Figure **??**).



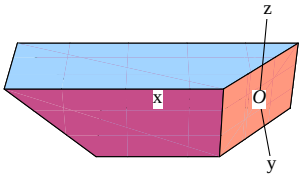
T_d has a trapezoid as projection



R_d makes right-handed spirals (mirror image of L_d)



P_d has a parallelogram as projection



L_d makes left-handed spirals (mirror image of R_d)

Figure 3: *Shapes of square beams bevelled for mitre joints*

The *Spiralosaurus*, on the left in Figure **??**, is generated by the turtle program

$$\left(T_4^2 \,;\, L_9^2 \,;\, T_4^2 \,;\, R_3^6 \right)^3 \quad , \tag{8}$$

where C^n stands for a sequence of n times command C. It consists of $(2 + 2 + 2 + 6) \times 3 = 36$ segements: a tight three-stranded right-hand inner spiral of 8 segments per strand $(T_4 \,;\, R_3^6 \,;\, T_4)$, enclosed by a loose half-pitch three-stranded left-hand outer spiral of 4 segments per strand $(T_4 \,;\, L_9^2 \,;\, T_4)$.

The *Braidwork*, on the right in Figure **??**, is generated by the turtle program

$$\left(L_1 \,;\, R_5 \,;\, R_6^2 \,;\, L_3 \,;\, R_1 \,;\, L_5 \,;\, L_6^2 \,;\, R_3 \right)^3 \tag{9}$$

and has 30 segments: a three-stranded right-hand spiral of 5 segments per strand $(L_1 \,;\, R_5 \,;\, R_6^2 \,;\, L_3)$, interwoven with a three-stranded left-hand spiral also of 5 segments per strand $(R_1 \,;\, L_5 \,;\, L_6^2 \,;\, R_3)$. It is congruent to its mirror image, that is, turning it upside-down is equivalent to reflecting it.

The triangular braiding in Figure **??** consists of three congruent links, each generated by the 18-segment turtle program

$$R_3 \,;\, P_3 \,;\, L_3 \,;\, P_3 \,;\, R_3 \,;\, L_3 \,;\, P_3^2 \,;\, R_3 \,;\, P_3^2 \,;\, L_3 \,;\, R_3 \,;\, P_3 \,;\, L_3 \,;\, P_3 \,;\, R_3 \,;\, L_3 \tag{10}$$

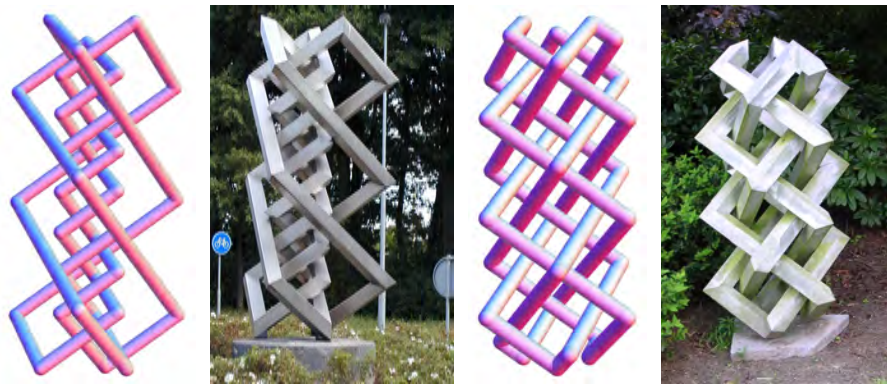


Figure 4: Spiralosaurus *(left: abstract design (1991) and stainless steel object in 1998) and* Braidwork *(right: abstract design (approx. 1988) and aluminium object (1990))*

Possibly not so easy to see, each link has a half-turn symmetry, because it can be written in the form

$$C\,;\,R_d\,;\,C'\,;\,L_d \tag{11}$$

where C' is the reverse of $C = R_3\,;\,P_3\,;\,L_3\,;\,P_3\,;\,R_3\,;\,L_3\,;\,P_3^2$. The half-turn axis connects the centres of the two explicit R- and L-segments of (**??**). The three links are braided together and cannot be taken apart without cutting. However, each pair of links is unlinked, as shown in the bottom row of Figure **??**. This is called a *Borromean* configuration.

The next artwork renders the figure-eight knot with 16 beams having an equilateral triangle as cross section (see Figure **??**). To describe it, we use the following two abbreviations

$$\begin{aligned} R'_d &:= \mathit{Segment}(d,\ \ 60^\circ, \varphi) \\ L'_d &:= \mathit{Segment}(d, -60^\circ, \varphi) \end{aligned}$$

where $\varphi = \arctan(2\sqrt{2}) \approx 70.5^\circ$. It is generated by the turtle program

$$(\ R'_2\,;\,L'_1\,;\,R'_1\,;\,L'_1\,;\,L'_2\,;\,R'_1\,;\,L'_1\,;\,R'_1\ )^2 \tag{12}$$

Programs (**??**), (**??**), (**??**), and (**??**) are all properly closed and simple. In the *Braidwork* (**??**) all torsion angles are $\pm 90^\circ$. The *Spiralosaurus* (**??**) also involves 0-torsion T-segments, whereas the triangular polylink (**??**) involves 180°-torsion P-segments. Hence, the latter two contain co-planar triples of consecutive segments. The first three artworks can be constructed from square beams using mitre joints that nicely match all the way round. In general, the latter property is non-trivial, which is addressed in (**?**). Note that in the actual artworks in Figure **??**, the beams have been rotated along their centre line by 45°. That way, the longitudinal faces of adjacent beams are not flush (co-planar) at the joint, adding a playful touch. Compare this to the triangular braided polylink in Figure **??**, where this was not done to allow the braid to be tightened, so that the beams of adjacent links can touch along their faces. The torsion angles in the *Figure Eight Knot* equal $\pm 60^\circ$. Again, the longitudinal edges of the beams match all the way round.

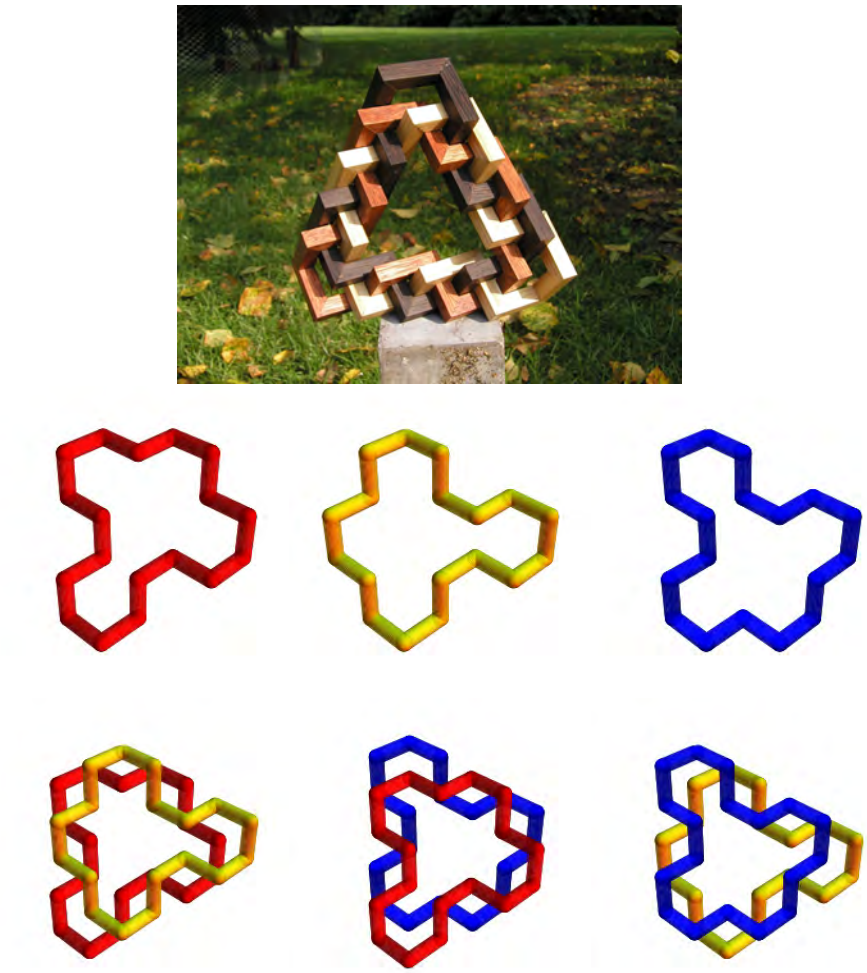


Figure 5: *Polylink (top, in three types of wood) consisting of three congruent links (middle) in Borromean configuration, also shown partly disentangled (bottom)*

4 Fundamentals of 3D turtle geometry

In this section, we present some basic 3D turtle geometry, that is, theorems about 3D turtle graphics (we are not aware of a good reference). Our aim is to have the ability to infer properties of a finite 3D polygonal path from a turtle program generating that path. For that purpose, we first define precisely what is meant by the path generated by a turtle program.

We start by defining the motion of a turtle program, relating time to position. Motion is not of primary interest, but facilitates other definitions (such as closed and simple). Given a turtle program p, the **motion** μ_p **of** p is a mapping

$$\mu_p : \mathbb{R} \to \mathbb{R}^3 \tag{13}$$

from time to space, such that $\mu_p(t)$ is the position of the turtle at time t under the assumptions that the turtle starts moving at $t = 0$ with a uniform speed

Figure 6: Figure Eight Knot *(abstract design, wooden object) from 16 beams with equilateral triangle as cross section*

of 1 distance unit per time unit for each *Move* command, and where *Turn* and *Roll* commands are executed instantaneously. Motion concerns the turtle's position only, because that is where ink is deposited, regardless of its attitude. By definition, all our programs start in the origin: $\mu_p(t) = (0,0,0)$ for $t \leq 0$. Note that the turtle's motion μ_p is a continuous function.

Let δ_p be the **total duration** of p's motion, that is,

$$\delta_p = \sum_{Move(d_i) \in p} |d_i| \tag{14}$$

Hence, $\mu_p(\delta_p)$ is the final position of the turtle under program p, and $\mu_p(t) = \mu_p(\delta_p)$ for $t \geq \delta_p$. Because of the unit speed, δ_p is also the total distance traveled by the turtle. The **final attitude of** p (heading, normal) is denoted by α_p.

Given a turtle program p, the **trace** τ_p **of** p (sometimes also referred to as path) is now defined as the set of points visited by p's motion, that is, the μ_p-image of the time interval $[0, \delta_p]$:

$$\tau_p = \mu_p[0, \delta_p] = \{\, \mu_p(t) \mid 0 \leq t \leq \delta_p \,\} \,. \tag{15}$$

The **empty program** is denoted by $\mathcal{I}$. It consists of no commands, leaving the turtle in its initial state. A turtle program p is called **trivial** when its trace consists of a single point: $|\tau_p| = 1$. The empty program is trivial. Since all programs start in the origin, p is trivial if and only if the turtle stays in the origin, that is, $\tau_p = \{\, (0,0,0) \,\}$. Note that the turtle could still wiggle around in the origin, changing its heading and normal.

We call program p **closed** when $\mu_p(0) = \mu_p(\delta_p)$. It is called **open** when it is not closed. Observe that every trivial program is closed, and hence every open program is non-trivial. However, there also exist closed non-trivial programs. A program is said to be **properly closed** when it is closed *and* its final attitude equals its initial attitude: $\mu_p(0) = \mu_p(\delta_p)$ and $\alpha_p = \alpha_{\mathcal{I}}$. The empty program is properly closed. Note that any closed program can be made properly closed by appending appropriate roll and turn commands.

Program p is called **simple** when $\mu_p(t_1) = \mu_p(t_2)$ for $0 \leq t_1 < t_2 \leq \delta_p$ implies $t_1 = 0$ and $t_2 = \delta_p$. That is, a simple program visits each point in its trace once, except when it is closed.

Requiring that μ_p is injective (one-one) on the interval $[0, \delta_p)$, allows 6-shaped traces (with initial position at the tip of the 6), and requiring it to be injective on $(0, \delta_p]$ allows 9-shaped traces (with final position at the tip of the 9). The 6-shape and 9-shape are not simple. There are two (non-exclusive) ways in which a program can fail to be simple: the program's trace contains **branching points** where three or more 'branches' meet (like in a figure 6, 8, or 9), or the program goes back and forth over the same piece two or more times. Both ways occur in the program

$$\textit{Turn}(90^\circ)\,;\,\textit{Move}(2)\,;\,\textit{Turn}(90^\circ)\,;\,\textit{Move}(1)\,;\;\;\textit{Turn}(180^\circ)\,;\,\textit{Move}(2) \tag{16}$$

that traces out a captical 'T', drawing the left-hand part of the cross bar twice.

We call a trace simple when there exists a simple program that generates it, and we call it closed when there exists a simple closed program that generates it. A non-simple trace has branching points. The following two theorems (without proof) motivate the relevance of simple programs.

> **Unique Motion Theorem for simple open programs** Let p and q be simple programs that define the same trace ($\tau_p = \tau_q$). If p is open, then these programs have the same motion as well ($\mu_p = \mu_q$), and q is also open.

This does not hold for closed programs, because a closed path can be traversed in two (opposite) directions, giving rise to two different motions. The following program is simple, closed, and has the same trace as (**??**), but not the same motion:

$$\textit{Turn}(60^\circ)\,;\,\textit{Segment}(1, 0, 60^\circ)\,;\,\textit{Segment}(-1, 0, 60^\circ)\,;\,\textit{Move}(1)\;. \tag{17}$$

For a program p, we define the **reverse motion** $\widetilde{\mu}_p$ **of** p as the motion μ given by

$$\widetilde{\mu}_p(t) = \mu_p(\delta_p - t) \tag{18}$$

Note that p is trivial, if and only if p's motion and its reverse are equal. We can now state the

> **Two-Motion Theorem for simple closed programs** Let p and q be simple programs that define the same trace ($\tau_p = \tau_q$). If p is closed, then q's motion is the same as either p's motion ($\mu_p = \mu_q$) or the reverse motion of p ($\widetilde{\mu}_p = \mu_q$), and q is also closed.

Non-simple programs defining the same trace can have vastly different motions.

4.1 Equivalence relations for turtle programs

Using these concepts we define five equivalence relations on turtle programs. These equivalence relations identify certain programs, thereby abstracting from specific features deemed irrelevant. Each equivalence can be viewed as a specific semantics for programs.

Two turtle programs p, q are called **motion equivalent**, denoted by $p \overset{\mu}{\equiv} q$, when their motions are equal, that is,

$$p \overset{\mu}{\equiv} q \quad \Longleftrightarrow \quad \mu_p = \mu_q\;. \tag{19}$$

Motion equivalence abstracts from how the sequence of program commands describes the motion. For instance, we have

$$\mathit{Move}(1)\,;\,\mathit{Roll}(90^\circ)\,;\,\mathit{Move}(1)\,;\,\mathit{Turn}(90^\circ) \;\overset{\mu}{\equiv}\; \mathit{Move}(2) \tag{20}$$

Note that motion-equivalent programs only need to have equal positions at corresponding times, but the attitudes need not equal. In fact, the attitude at time t is not a well-defined notion outside *Move* commands. In the preceding example (**??**), the attitudes start to diverge at $t = 1$. In particular, motion-equivalent programs have the same final position, but not necessarily the same final attitude.

Two programs p, q are called **final-attitude equivalent**, denoted by $p \overset{\alpha}{\equiv} q$ when their final attitudes are equal, that is,

$$p \overset{\alpha}{\equiv} q \quad \Longleftrightarrow \quad \alpha_p = \alpha_q \;. \tag{21}$$

Final-attitude equivalence abstracts from everything that happens between the initial and final state.

Turtle programs p, q are called **equivalent**, denoted by $p \overset{e}{\equiv} q$ or just $p \equiv q$, when they are motion equivalent *and* final-attitude equivalent. In particular, equivalent programs have the same final *state*. Note that equivalent implies motion equivalent, but not the other way round. This strongest form of equivalence is needed to reason about the sequential composition of programs. The result of $p\,;q$ not only depends on where p ends but also on the final attitude after p, because the final state of p serves as initial state of q.

The programs p, q are called **trace equivalent**, denoted by $p \overset{\tau}{\equiv} q$, when their traces are equal, that is,

$$p \overset{\tau}{\equiv} q \quad \Longleftrightarrow \quad \tau_p = \tau_q \;. \tag{22}$$

Trace equivalence abstracts from time, i.e., it does not matter when and how often a point is visited by the turtle. Note that motion equivalent implies trace equivalent, but in general not the other way round. The Unique Motion Theorem stated earlier can be rephrased as follows:

> For simple open programs, trace equivalence implies motion equivalence.

Note that all trivial programs are closed and simple, and they are all trace equivalent to each other, and hence also motion equivalent. The **empty program** consists of no commands, and we denote it by $\mathcal{I}$. It is trivial and properly closed, and acts as identity operation (unit element) for sequential composition. All properly closed trivial programs are equivalent to $\mathcal{I}$.

Finally, two turtle programs p, q are called **trace congruent**, or briefly **congruent**, denoted by $p \overset{c}{\equiv} q$, when their traces are congruent, that is, when the traces can be made to coincide by appropriately moving them in space through translation, rotation, reflection, or a combination of these operations. Congruence abstracts from all details of the program except the shape of the generated trace. Note that trace equivalent implies congruent, but not the other way round.

Each of the properties of being trivial, closed, properly closed, and simple is preserved by equivalence. That is, if programs p and q are equivalent ($p \equiv q$), and p has one of the aforementioned properties, then q has that property as well.

4.2 Sequential composition and equivalence

Since we want to reason about the equivalence of sequences of commands, it is good to know how sequential composition and equivalence are related. Observe that sequential composition is associative and has the empty program as unit (identity) element:

$$(p\,;q)\,;r \quad = \quad p\,;(q\,;r) \tag{23}$$

$$\mathcal{I}\,;p \quad = \quad p\,;\mathcal{I} \;=\; p \tag{24}$$

Consider four programs p, p', q, q'. Interesting relationships with equivalence have the form

$$\text{if } p \overset{x}{\equiv} p' \text{ and } q \overset{y}{\equiv} q' \quad \text{then} \quad p\,;q \overset{z}{\equiv} p'\,;q' \tag{25}$$

where x, y, z are one of μ, α, e, τ, c. Note that $q \overset{y}{\equiv} q'$ is based on execution starting in the default initial state (see Section **??**), viz. in the origin with heading along the positive x-axis and normal along the positive z-axis. On the other hand, in $p\,;q \overset{z}{\equiv} p'\,;q'$ the programs q, q' have as initial state the final state of p, p' respectively. The most relevant for us are the following three such relationships, and in particular the last one:

$$\text{if } p \overset{\alpha}{\equiv} p' \text{ and } q \overset{\alpha}{\equiv} q' \quad \text{then} \quad p\,;q \overset{\alpha}{\equiv} p'\,;q' \tag{26}$$

$$\text{if } p \overset{e}{\equiv} p' \text{ and } q \overset{\mu}{\equiv} q' \quad \text{then} \quad p\,;q \overset{\mu}{\equiv} p'\,;q' \tag{27}$$

$$\text{if } p \overset{e}{\equiv} p' \text{ and } q \overset{e}{\equiv} q' \quad \text{then} \quad p\,;q \overset{e}{\equiv} p'\,;q' \tag{28}$$

The latter implies that, if we replace a subsequence in a program by an equivalent subsequence, then we obtain a program equivalent to the original. Other combinations not subsumed in the preceding three do not work out, but we do have

$$\mathit{Turn}(\varphi) \quad \overset{\mu}{\equiv} \quad \mathcal{I} \tag{29}$$

$$\mathit{Roll}(\psi) \quad \overset{\mu}{\equiv} \quad \mathcal{I} \tag{30}$$

$$\mathit{Move}(d) \quad \overset{\alpha}{\equiv} \quad \mathcal{I} \tag{31}$$

Combining (**??**) and (**??**) with (**??**) yields

$$p\,;\mathit{Turn}(\varphi) \quad \overset{\mu}{\equiv} \quad p \tag{32}$$

$$p\,;\mathit{Roll}(\psi) \quad \overset{\mu}{\equiv} \quad p \tag{33}$$

that is, trailing roll and turn commands do not affect the motion. For leading turn and roll commands we have a weaker relationship:

$$\mathit{Turn}(\varphi)\,;p \quad \overset{c}{\equiv} \quad p \tag{34}$$

$$\mathit{Roll}(\psi)\,;p \quad \overset{c}{\equiv} \quad p \tag{35}$$

since the initial turn or roll only rotates the shape (along the normal and along the heading respectively).

4.3 Properties of basic commands

Every turtle program (obviously) defines a 3D polygonal path. Conversely, every 3D polygonal path starting in the origin can be generated by a suitable turtle program. The following properties of the basic commands pave the road for calculational reasoning. In particular, they will enable us to transform a given (simple) turtle program into an equivalent program in some standard form.

1. *Turn* and *Roll* are periodic with period 360°. That is, we have

$$\mathit{Turn}(\varphi_1) \;\equiv\; \mathit{Turn}(\varphi_2) \quad\Longleftrightarrow\quad \varphi_1 = \varphi_2 \pmod{360^\circ} \tag{36}$$
$$\mathit{Roll}(\psi_1) \;\equiv\; \mathit{Roll}(\psi_2) \quad\Longleftrightarrow\quad \psi_1 = \psi_2 \pmod{360^\circ} \tag{37}$$

   In particular, $\mathit{Turn}(-180^\circ) \equiv \mathit{Turn}(180^\circ)$ and $\mathit{Roll}(-180^\circ) \equiv \mathit{Roll}(180^\circ)$.

2. $\mathit{Move}(d)$ is equivalent to the empty program $\mathcal{I}$ if and only if $d = 0$. Similarly, $\mathit{Turn}(\alpha)$ and $\mathit{Roll}(\alpha)$ are equivalent to $\mathcal{I}$ if and only if $\alpha = 0 \pmod{360^\circ}$. In formulae:

$$\mathit{Move}(d) \equiv \mathcal{I} \quad\Longleftrightarrow\quad d = 0 \tag{38}$$
$$\mathit{Turn}(\varphi) \equiv \mathcal{I} \quad\Longleftrightarrow\quad \varphi = 0 \pmod{360^\circ} \tag{39}$$
$$\mathit{Roll}(\psi) \equiv \mathcal{I} \quad\Longleftrightarrow\quad \psi = 0 \pmod{360^\circ} \tag{40}$$

3. Adjacent commands of the same type can be merged:

$$\mathit{Move}(d_1)\,;\mathit{Move}(d_2) \;\equiv\; \mathit{Move}(d_1 + d_2) \quad \text{provided } d_1 d_2 \geq 0 \tag{41}$$
$$\mathit{Turn}(\varphi_1)\,;\mathit{Turn}(\varphi_2) \;\equiv\; \mathit{Turn}(\varphi_1 + \varphi_2) \tag{42}$$
$$\mathit{Roll}(\psi_1)\,;\mathit{Roll}(\psi_2) \;\equiv\; \mathit{Roll}(\psi_1 + \psi_2) \tag{43}$$

   Hence, on account of the preceding properties, $\mathit{Turn}(180^\circ)$ and $\mathit{Roll}(180^\circ)$ are their own inverse. Furthermore, $\mathit{Turn}(-\varphi)$ is the inverse of $\mathit{Turn}(\varphi)$, and similarly $\mathit{Roll}(-\psi)$ of $\mathit{Roll}(\psi)$. Note, however, that $\mathit{Move}(-d)$ is not the inverse of $\mathit{Move}(d)$ if $d \neq 0$, because a command does not erase points that have been drawn. This also explains the condition on merging two *Move* commands: this only works if either *Move* is equivalent to the empty program, or if both *Move* commands have the same sign. We do have final-attitude equivalence in case of arbitrary *Move*s:

$$\mathit{Move}(d_1)\,;\mathit{Move}(d_2) \;\overset{\alpha}{\equiv}\; \mathit{Move}(d_1 + d_2) \tag{44}$$

4. Since addition is commutative, Property **??** implies that adjacent commands of the same type commute:

$$\mathit{Move}(d_1)\,;\mathit{Move}(d_2) \;\equiv\; \mathit{Move}(d_2)\,;\mathit{Move}(d_1) \quad \text{provided } d_1 d_2 \geq 0 \tag{45}$$
$$\mathit{Turn}(\varphi_1)\,;\mathit{Turn}(\varphi_2) \;\equiv\; \mathit{Turn}(\varphi_2)\,;\mathit{Turn}(\varphi_1) \tag{46}$$
$$\mathit{Roll}(\psi_1)\,;\mathit{Roll}(\psi_2) \;\equiv\; \mathit{Roll}(\psi_2)\,;\mathit{Roll}(\psi_1) \tag{47}$$

   For *Move* this is again conditional on the signs of the parameters. And, again, we do have final-attitude equivalence in case of arbitrary *Move*s:

$$\mathit{Move}(d_1)\,;\mathit{Move}(d_2) \;\overset{\alpha}{\equiv}\; \mathit{Move}(d_2)\,;\mathit{Move}(d_1) \tag{48}$$

5. When the turtle rolls upside down (making a half-roll), its turning sense looks reflected:

$$\begin{aligned} \mathit{Roll}(180^\circ)\,;\,\mathit{Turn}(\varphi) &\equiv \mathit{Turn}(-\varphi)\,;\,\mathit{Roll}(180^\circ) \\ \mathit{Turn}(\varphi) &\equiv \mathit{Roll}(180^\circ)\,;\,\mathit{Turn}(-\varphi)\,;\,\mathit{Roll}(180^\circ) \end{aligned} \tag{49}$$

Similarly for half-turn and roll sense:

$$\begin{aligned} \mathit{Turn}(180^\circ)\,;\,\mathit{Roll}(\psi) &\equiv \mathit{Roll}(-\psi)\,;\,\mathit{Turn}(180^\circ) \\ \mathit{Roll}(\psi) &\equiv \mathit{Turn}(180^\circ)\,;\,\mathit{Roll}(-\psi)\,;\,\mathit{Turn}(180^\circ) \end{aligned} \tag{50}$$

And also for half-turn and move sense:

$$\begin{aligned} \mathit{Turn}(180^\circ)\,;\,\mathit{Move}(d) &\equiv \mathit{Move}(-d)\,;\,\mathit{Turn}(180^\circ) \\ \mathit{Move}(d) &\equiv \mathit{Turn}(180^\circ)\,;\,\mathit{Move}(-d)\,;\,\mathit{Turn}(180^\circ) \end{aligned} \tag{51}$$

6. Adjacent *Move* and *Roll* commands commute:

$$\mathit{Move}(d)\,;\,\mathit{Roll}(\psi) \equiv \mathit{Roll}(\psi)\,;\,\mathit{Move}(d) \tag{52}$$

7. The pairs *Turn*–*Move* and *Turn*–*Roll* do not commute, unless one of them is equivalent to the identity operation, or in the special case of

$$\mathit{Turn}(180^\circ)\,;\,\mathit{Roll}(180^\circ) \equiv \mathit{Roll}(180^\circ)\,;\,\mathit{Turn}(180^\circ) \tag{53}$$

The latter equivalence follows from (**??**) and (**??**). Its unicity is addressed after Property **??**. Consequently, the combination $\mathit{Turn}(180^\circ)\,;\mathit{Roll}(180^\circ)$ is its own inverse. This combination is so special that it deserves its own name. We call it $\mathcal{H}$, which derives from half-loop. This name is explained by its relation to *Dive*:

$$\begin{aligned} \mathit{Dive}(180^\circ) &\equiv \mathit{Turn}(90^\circ)\,;\,\mathit{Roll}(180^\circ)\,;\,\mathit{Turn}(-90^\circ) \\ &\equiv \mathit{Turn}(90^\circ)\,;\,\mathit{Turn}(90^\circ)\,;\,\mathit{Roll}(180^\circ) \\ &\equiv \mathit{Turn}(180^\circ)\,;\,\mathit{Roll}(180^\circ) \\ &\equiv \mathcal{H} \end{aligned}$$

$\mathcal{H}$ results in inverted and reverted flight.

8. Every program consisting of *Turn* and *Roll* commands only, i.e. without *Move* commands, has an equivalent program of the form

$$\mathit{Roll}(\psi)\,;\,\mathit{Turn}(\varphi)\,;\,\mathit{Roll}(\psi') \tag{54}$$

with $0 \le \varphi \le 180^\circ$ and $-180^\circ < \psi, \psi' \le 180^\circ$.

Proof: Choose ψ such that $\mathit{Roll}(\psi)$ makes the turtle's base plane contain the final heading in its left-hand half-plane. Choose φ such that $\mathit{Turn}(\varphi)$ aligns the heading with the final heading. Because the final heading is in the left-hand half-plane, we have $0 \le \varphi \le 180^\circ$. Choose ψ' such that $\mathit{Roll}(\psi')$ aligns the normal with the final normal. The range for ψ and ψ' follows from (**??**). □

We can refine Property **??** further. If $0 < \varphi < 180°$, then ψ and ψ' are uniquely determined. If $\varphi = 0 \pmod{180°}$, then it can be arranged that $\psi = 0$, in which case ψ' is uniquely determined, or it can be arranged that $\psi' = 0$, in which case ψ is uniquely determined.

As a corollary, there exists a rule to rewrite

$$\mathit{Roll}(\psi_1)\,;\,\mathit{Turn}(\varphi_1)\,;\,\mathit{Roll}(\psi_2)\,;\,\mathit{Turn}(\varphi_2)$$

into (**??**). However, in general, this rule involves some messy (inverse) trigonometry. Together with (**??**) and (**??**), this rule would suffice to prove Property **??** in a calculational way.

For completeness' sake, we point out that it would also be possible to restrict ψ in (**??**) to a 180° range, and φ and ψ' to a 360° range.

We now address the uniqueness of (**??**) stated in Property **??**, viz. that

$$\mathit{Turn}(\varphi)\,;\,\mathit{Roll}(\psi) \;\equiv\; \mathit{Roll}(\psi)\,;\,\mathit{Turn}(\varphi) \tag{55}$$

holds only if $\varphi = 0$, or $\psi = 0$ or $\varphi = \psi = 180°$ (all taken $\bmod 360°$). This follows from Property **??**, by rewriting (**??**) into

$$\mathit{Turn}(\varphi) \;\equiv\; \mathit{Roll}(\psi)\,;\,\mathit{Turn}(\varphi)\,;\,\mathit{Roll}(-\psi) \tag{56}$$

and exploiting the uniqueness of (**??**).

4.4 Reducing a turtle program to standard form

Intuitively, it seems reasonable to conjecture that every 3D polygonal path can be generated by a program in some standard form. However, there are a couple of subtleties to take into account. Therefore, we will carefully formulate the main standardisation theorem. To facilitate more complex expressions and calculations, we will use a simplified notation from now on:

$$\begin{array}{rcl} \mathcal{M}(d) & := & \mathit{Move}(d) \\ \mathcal{T}(\varphi) & := & \mathit{Turn}(\varphi) \\ \mathcal{R}(\psi) & := & \mathit{Roll}(\psi) \\ \mathcal{S}(d,\psi,\varphi) & := & \mathit{Segment}(d,\psi,\varphi) \end{array}$$

and we omit the semicolon, that is, sequencing will be denoted by juxtaposition. Thus, we have $\mathcal{S}(d,\psi,\varphi) \;=\; \mathcal{M}(d)\;\mathcal{R}(\psi)\;\mathcal{T}(\varphi)$.

Standardisation Theorem Every turtle program p is equivalent to exactly one turtle program of the form

$$\mathcal{R}(\psi_0)\;\mathcal{T}(\varphi_0)\;\mathcal{S}(d_1,\psi_1,\varphi_1)\;\ldots\;\mathcal{S}(d_n,\psi_n,\varphi_n)\;\mathcal{R}(\psi_{n+1}) \tag{57}$$

where $n \geq 0$ (if $n = 0$ then there are no $\mathcal{S}$ commands), and the parameters satisfy these constraints:

C1: $d_i > 0$ for $1 \leq i \leq n$,

C2: $-180° < \psi_i \leq 180°$ for $0 \leq i \leq n+1$,

C3: $0 \leq \varphi_i \leq 180^\circ$ for $1 \leq i \leq n$,

C4: $\varphi_i \neq 0$ for $1 \leq i < n$, i.e., between $\mathcal{M}$ commands, and

C5: if $\varphi_i = 0 \pmod{180^\circ}$ then $\psi_i = 0$ for $0 \leq i \leq n$.

Furthermore, if p is simple, then

C4a: $\varphi_i \neq 180^\circ$ for $1 \leq i < n$.

Observe that constraints C4 and C4a, can be combined into $0 < \varphi_i < 180^\circ$ for $1 \leq i < n$ in view of constraint C3.

We call (**??**) the **standard form**[1] **of** p, and denote it by $\sigma(p)$. We call it a **strict standard form** when constraint C4a holds. Thus, according to the Standardisation Theorem, simple programs have a strict standard form. However, a program in strict standard form is not necessarily simple, because self-overlapping can still occur for edges that are further apart. This is illustrated by the following program in strict standard form, generating an equilateral triangle whose first edge is drawn twice, so that it is not simple:

$$\mathcal{S}(1,0,120^\circ)\ \mathcal{S}(1,0,120^\circ)\ \mathcal{S}(1,0,120^\circ)\ \mathcal{S}(1,0,0)\ . \tag{58}$$

Usually, we will omit any of $\mathcal{R}(\psi_0)$, $\mathcal{T}(\phi_0)$, and $\mathcal{R}(\psi_{n+1})$ when their angle parameter equals zero. In general, the standard form consists of

- n move commands: $\mathcal{M}(d_1),\ \ldots,\ \mathcal{M}(d_n)$
- $n+1$ turn commands: $\mathcal{T}(\varphi_0),\ \ldots,\ \mathcal{T}(\varphi_n)$
- $n+2$ roll commands: $\mathcal{R}(\varphi_0),\ \ldots,\ \mathcal{R}(\varphi_{n+1})$

In view of Property **??**, it is not 'easy' to determine the standard form of an arbitrary turtle program. But it is easy to check that a given program is in (strict) standard form, because that concerns a local property of the program. Therefore, it is most convenient to write programs in standard form from the very beginning. The programs for the artwork in Section **??** are all in strict standard form.

Before proving the Standardisation Theorem, let us look at some corollaries. By dropping the trailing $\mathcal{R}$ command we infer, on account of (**??**), that every program p is *motion* equivalent to a program of the form

$$\mathcal{R}(\psi_0)\ \mathcal{T}(\varphi_0)\ \mathcal{S}(d_1,\psi_1,\varphi_1)\ \ldots\ \mathcal{S}(d_n,\psi_n,\varphi_n) \tag{59}$$

where d_i, ψ_i, and φ_i satisfy constraints C1 through C5. On account of (**??**) and (**??**), we now can even require $\psi_n = \varphi_n = 0$. By doing so, the representation (**??**) is unique. We call this the **μ-standard form of** p and denote it by $\sigma_\mu(p)$.

By dropping the leading $\mathcal{R}$ and $\mathcal{T}$ commands as well, using (**??**) and (**??**), it follows that every program p is *congruent* to a program in, what we call, **c-standard form** $\sigma_c(p)$:

$$\mathcal{S}(d_1,\psi_1,\varphi_1)\ \ldots\ \mathcal{S}(d_n,\psi_n,\varphi_n) \tag{60}$$

where the parameters again satisfy constraints C1 through C5, and additionally $\psi_1 = \psi_n = \varphi_n = 0$ when $n \geq 1$. Note that now the representation is

[1] We did not call it the *normal form* to avoid confusion with the turtle's normal vector.

no longer unique; even not for simple open programs. That is, two congruent simple open programs can have different c-standard forms. Consider, for example, $\mathcal{S}(1,0,90°)\ \mathcal{S}(2,0,0)$ and $\mathcal{S}(2,0,90°)\ \mathcal{S}(1,0,0)$. We settle this issue in Section **??**, because it involves (lack of) symmetry.

Here is a proof of the Standardisation Theorem. Consider turtle program p. We will first construct a standard form for p, then we address unicity and finally the case of simple p. We will give an inductive construction of a standard form for p. The empty program and the basic commands are the base cases, and for the inductive step, a longer program can be written as $p\ q$, where neither p nor q are the empty program.

$\mathcal{I}$**:** $\mathcal{R}(0)\ \mathcal{T}(0)\ \mathcal{R}(0)$

$\mathcal{M}(d)$**:** We distinguish three cases depending on the sign of d:

$d = 0$**:** See $\mathcal{I}$.

$d > 0$**:** $\mathcal{R}(0)\ \mathcal{T}(0)\ \mathcal{S}(d,0,0)\ \mathcal{R}(0)$

$d < 0$**:** Using (**??**): $\mathcal{R}(0)\ \mathcal{T}(180°)\ \mathcal{S}(-d,0,180°)\ \mathcal{R}(0)$, observing that $-d > 0$.

$\mathcal{T}(\varphi)$**:** On account of the periodicity of $\mathcal{T}$, we may assume $-180° \leq \varphi \leq 180°$. We distinguish two cases depending on the sign of φ:

$0 \leq \varphi \leq 180°$**:** $\mathcal{R}(0)\ \mathcal{T}(\varphi)\ \mathcal{R}(0)$

$-180° < \varphi < 0$**:** Using (**??**): $\mathcal{R}(180°)\ \mathcal{T}(-\varphi)\ \mathcal{R}(180°)$, observing that we have $0 < -\varphi < 180°$.

$\mathcal{R}(\psi)$**:** $\mathcal{R}(0)\ \mathcal{T}(0)\ \mathcal{R}(\psi)$, because we may assume $-180° < \psi \leq 180°$ on account of the periodicity of $\mathcal{R}$.

$p\ q$**:** We describe a way of stitching standard forms of p and q together to obtain a standard form for $p\ q$. Suppose we have standard forms for p and q with n and n' occurrences of $\mathcal{S}$ respectively:

$$\begin{aligned} \sigma(p) &= \ldots\ [\mathcal{M}(d_n)]\ \mathcal{R}(\psi_n)\ \mathcal{T}(\varphi_n)\ \mathcal{R}(\psi_{n+1}) \\ \sigma(q) &= \mathcal{R}(\psi'_0)\ \mathcal{T}(\varphi'_0)\ [\mathcal{M}(d'_1)]\ \mathcal{R}(\psi'_1)\ \ldots \end{aligned}$$

where the square brackets indicate that the move commands need not be present (viz. if $n = 0$ or $n' = 0$), and where the parameters satisfy constraints C1 through C5. Note that the $\ldots$ in $\sigma(p)$ is either empty (viz. if $n = 0$) or ends with $\mathcal{T}(\varphi_{n-1})$. Similarly, the $\ldots$ in $\sigma(q)$ is either empty (viz. if $n' = 0$) or starts with $\mathcal{T}(\varphi'_1)$.

We now rewrite $p\ q$, preserving equivalence, into standard form:

$p\ q$
  { by induction hypothesis: $p \equiv \sigma(p)$ and $q \equiv \sigma(q)$ }
$\sigma(p)\ \sigma(q)$
  { assumption about $\sigma(p)$ and $\sigma(q)$ }
$\ldots\ [\mathcal{M}(d_n)]\ \mathcal{R}(\psi_n)\ \mathcal{T}(\varphi_n)\ \mathcal{R}(\psi_{n+1})\ \mathcal{R}(\psi'_0)\ \mathcal{T}(\varphi'_0)\ [\mathcal{M}(d'_1)]\ \mathcal{R}(\psi'_1)\ \ldots$
  { $\mathcal{M}$ and $\mathcal{R}$ commute }
$\ldots\ [\mathcal{M}(d_n)]\ \mathcal{R}(\psi_n)\ \mathcal{T}(\varphi_n)\ \mathcal{R}(\psi_{n+1})\ \mathcal{R}(\psi'_0)\ \mathcal{T}(\varphi'_0)\ \mathcal{R}(\psi'_1)\ [\mathcal{M}(d'_1)]\ \ldots$
  { Property **??**, yielding $0 \leq \varphi \leq 180°$, and $\psi = 0$ if $\varphi = 0 \pmod{180°}$ }
$\ldots\ [\mathcal{M}(d_n)]\ \mathcal{R}(\psi)\ \mathcal{T}(\varphi)\ \mathcal{R}(\psi')\ [\mathcal{M}(d'_1)]\ \ldots$
  { $\mathcal{M}$ and $\mathcal{R}$ commute }
$\ldots\ [\mathcal{M}(d_n)]\ \mathcal{R}(\psi)\ \mathcal{T}(\varphi)\ [\mathcal{M}(d'_1)]\ \mathcal{R}(\psi')\ \ldots$

In case $n \neq 0$ and $n' \neq 0$, we need to do some more work if $\varphi = 0$, because it violates constraint C4 (in a standard form, $\mathcal{T}(0)$ is not permitted between $\mathcal{M}$ commands). In that case we can continue the rewriting as follows

$\ldots\ [\mathcal{M}(d_n)]\ \mathcal{R}(\psi)\ \mathcal{T}(\varphi)\ [\mathcal{M}(d'_1)]\ \mathcal{R}(\psi')\ \ldots$
  { $\varphi = 0$, thus $\mathcal{T}(\varphi) \equiv \mathcal{I}$ }
$\ldots\ \mathcal{M}(d_n)\ \mathcal{R}(\psi)\ \mathcal{M}(d'_1)\ \mathcal{R}(\psi')\ \ldots$
  { $\mathcal{M}$ and $\mathcal{R}$ commute }
$\ldots\ \mathcal{M}(d_n)\ \mathcal{M}(d'_1)\ \mathcal{R}(\psi)\ \mathcal{R}(\psi')\ \ldots$
  { merge adjacent $\mathcal{M}$ and $\mathcal{R}$, using $d_n, d'_1 > 0$ (C1) }
$\ldots\ \mathcal{M}(d_n + d'_1)\ \mathcal{R}(\psi + \psi')\ \ldots$

Note that $d_n + d'_1 > 0$, and that $\psi + \psi'$ can be reduced modulo 360° to the range $(-180°, 180°]$. If $n' = 0$ then the newly derived program satisfies constraint C5 as well. However, if $n' \neq 0$ then we may need to do some further rewriting, viz. if the roll angle before the trailing $\ldots$ (i.e., in $\mathcal{R}(\psi')$ or $\mathcal{R}(\psi + \psi')$) is non-zero and $\varphi'_1 = 0 \pmod{180°}$. In that case, the non-zero roll angle needs to be 'percolated' up the chain, using (**??**), (**??**), and (**??**), until reaching the end or a $\varphi'_i \neq 0 \pmod{180°}$. In view of constraints C4 and C4a, percolation takes at most one step for strict standard forms.

The parameters now satisfy constraints C1 through C5. Therefore, the resulting program is in standard form.

This proves the existence of the desired standard form (**??**). Its uniqueness follows from the observation that two programs in standard form (**??**) are equivalent if and only if they are equal. A detailed proof of this is rather tedious, since there are many cases to consider. The main idea is to focus on the earliest command where the two programs deviate and then argue that this causes either different motions or different final states. All parameter constraints on standard form (**??**) play a role; that is, weakening any parameter condition will destroy uniqueness. Finally, a program in standard form having $\varphi_i = 180°$ for

some i with $1 \leq i < n$ is not simple, because all d_i are positive. More precisely, $\mathcal{M}(d_i)$ and $\mathcal{M}(d_{i+1})$ would overlap if $\varphi_i = 180°$. □

It is possible to standardise programs in various other ways. For instance, instead of restricting roll angles ψ_i $(0 \leq i \leq n)$ to a 360° range (C2), these could be restricted to a 180° range, such as $-90° < \psi_i \leq 90°$ or $0 \leq \psi_i < 180°$. It would then be necessary to derestrict turn angles φ_i from their 180° range (C3) to a 360° range (excluding $\varphi_i \neq 0 \pmod{360°}$ for $1 \leq i < n$), such as $0 \leq \varphi_i < 360°$ or $-180° < \varphi \leq 180°$. Instead of C5, we could have taken

C5a: if $\varphi_i = 0 \pmod{180°}$ then $\psi_{i+1} = 0$ for $0 \leq i \leq n$,

In that case, the inductive construction would need to 'percolate' non-zero roll angles over $\varphi_i = 0 \pmod{180°}$ *down* the chain, instead of up.

These other forms can be constructed using the Standardisation Theorem and the basic properties, in particular, turn sign reversal (**??**). But for our purposes the standard form as defined above suffices.

4.5 Motivation for our definition of $Segment(d, \psi, \varphi)$

This is also a good moment to reflect on our definition of *Segment*. The standard form (**??**) written out in basic commands looks like this:

$$\mathcal{R}\ \mathcal{T}\ \mathcal{M}\ \mathcal{R}\ \mathcal{T}\ \mathcal{M}\ \mathcal{R}\ \mathcal{T}\ \ldots\ \mathcal{M}\ \mathcal{R}\ \mathcal{T}\ \mathcal{R} \tag{61}$$

The sequence $\mathcal{M}\ \mathcal{R}\ \mathcal{T}$ is not the only repetitive unit that can be discerned here. In fact, because $\mathcal{M}$ and $\mathcal{R}$ commute, each of the six permutations of the three basic commands can be used, as can be seen in Table **??**. Neither of these permutations matches perfectly, since each requires special leading and/or trailing commands as well.

$\mathcal{R}$	$\mathcal{T}$	$(\mathcal{M}$	$\mathcal{R}$	$\mathcal{T})$	$(\mathcal{M}$	$\mathcal{R}$	$\mathcal{T})$	$\ldots$	$(\mathcal{M}$	$\mathcal{R}$	$\mathcal{T})\ \mathcal{R}$
$\mathcal{R}$	$(\mathcal{T}$	$\mathcal{M}$	$\mathcal{R})$	$(\mathcal{T}$	$\mathcal{M}$	$\mathcal{R})$	$(\mathcal{T}$	$\ldots$	$\mathcal{M}$	$\mathcal{R})$	$\mathcal{T}\ \mathcal{R}$
$(\mathcal{R}$	$\mathcal{T}$	$\mathcal{M})$	$(\mathcal{R}$	$\mathcal{T}$	$\mathcal{M})$	$(\mathcal{R}$	$\mathcal{T}$	$\ldots$	$\mathcal{M})$	$\mathcal{R}$	$\mathcal{T}\ \mathcal{R}$
$\mathcal{R}$	$\mathcal{T}$	$\mathcal{R}$	$(\mathcal{M}$	$\mathcal{T}$	$\mathcal{R})$	$(\mathcal{M}$	$\mathcal{T}$	$\ldots$	$\mathcal{R})$	$(\mathcal{M}$	$\mathcal{T}\ \mathcal{R})$
$\mathcal{R}$	$\mathcal{T}$	$(\mathcal{R}$	$\mathcal{M}$	$\mathcal{T})$	$(\mathcal{R}$	$\mathcal{M}$	$\mathcal{T})$	$\ldots$	$(\mathcal{R}$	$\mathcal{M}$	$\mathcal{T})\ \mathcal{R}$
$\mathcal{R}$	$(\mathcal{T}$	$\mathcal{R}$	$\mathcal{M})$	$(\mathcal{T}$	$\mathcal{R}$	$\mathcal{M})$	$(\mathcal{T}$	$\ldots$	$\mathcal{R}$	$\mathcal{M})$	$\mathcal{T}\ \mathcal{R}$

Table 1: *All ways of finding repetitive units in the standard form*

Table **??** lists some characteristics of the six possible repetitive units when they would be used without special leading commands. The reason why we have chosen for the order $\mathcal{M}\ \mathcal{R}\ \mathcal{T}$ is that the roll angle ψ equals the torsion of the edge drawn by the move command, and the turn angle φ concerns the angle between that edge and the next one (if present). This is useful when constructing a path from beams connected by mitre joints. The parameters needed to construct the beam for an edge are the bevel angles at each end and the torsion angle[2]. These parameters are derivable from the program up to and including the command

[2]In general, also the longitudinal rotation angle of the cross section is needed. This is addressed in (**?**).

Unit	φ	ψ	**Without leading commands**
$\mathcal{M}\ \mathcal{R}\ \mathcal{T}$	self–succ	self	first edge along x^+-axis
$\mathcal{R}\ \mathcal{M}\ \mathcal{T}$	same as $\mathcal{M}\ \mathcal{R}\ \mathcal{T}$		
$\mathcal{T}\ \mathcal{M}\ \mathcal{R}$	pred–self	self	first edge in (x, y^+)-plane
$\mathcal{T}\ \mathcal{R}\ \mathcal{M}$	same as $\mathcal{T}\ \mathcal{M}\ \mathcal{R}$		
$\mathcal{R}\ \mathcal{T}\ \mathcal{M}$	pred–self	pred	none, provided $\varphi_0 = 0$ allowed
$\mathcal{M}\ \mathcal{T}\ \mathcal{R}$	self–succ	succ	first edge along x^+-axis, second edge in (x, y^+)-plane

φ concerns angle between indicated edges through $\mathcal{T}(\varphi)$
ψ is torsion angle for indicated edge through $\mathcal{R}(\psi)$, where
self = edge produced by M, pred = predecessor edge, succ = successor edge

Table 2: *Repetitive units and the role of angles*

$\mathcal{S}_i$ corresponding to that edge, viz. from φ_{i-1}, φ_i, and ψ_i. There is no need to look ahead.

For closed programs, however, the situation is somewhat more complicated. Consider the program (see Figure **??**, left, for its trace)

(62) $\mathcal{R}(90°)\ \mathcal{T}(45°)\ \mathcal{S}(d, -\psi, 120°)\ \mathcal{S}(d, 0, 120°)\ \mathcal{S}(d, \psi{-}90°, 90°)\ \mathcal{R}(-45°)$

where $d = \sqrt{2}$ and $\psi = \arctan(1/\sqrt{2}) \approx 54.74°$. It is in restricted standard form and properly closed. The program generates an equilateral triangle with as vertices the origin, $(1, 0, 1)$, and $(0, 1, 1)$. Two of the 60° vertex angles are explicit in the program, but not the one at the origin, where the triangle closes.

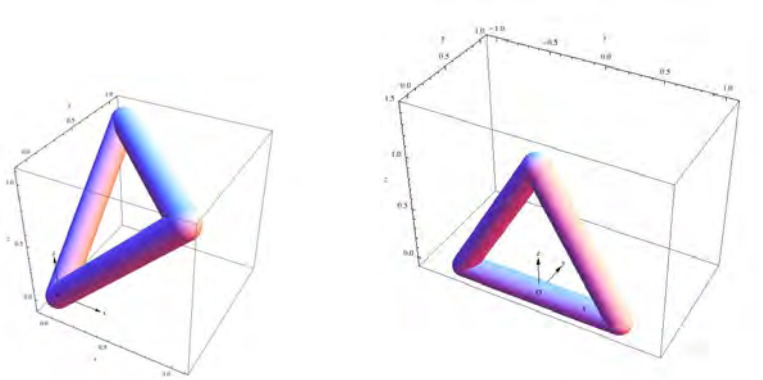


Figure 7: *Traces generated by programs (**??**) and (**??**)*

Furthermore, consider the program (see Figure **??**, right, for its trace)

(63) $\mathcal{S}(d/2, 90°, 120°)\ \mathcal{S}(d, 0, 120°)\ \mathcal{S}(d, 0, 120°)\ \mathcal{S}(d/2, -90°, 0)$

where again $d = \sqrt{2}$. It is also in restricted standard form and properly closed. The program generates another equilateral triangle with side d, now in the (x, z)-plane. Nevertheless, it contains *four* segment commands, because the

origin is at the centre of one of the edges, which thereby has been split into two separate but aligned segments. These two segments cannot be combined without displacing the entire path. The two programs (**??**) and (**??**) are congruent, but that is not easy to see from their c-standard forms. We will address these issues further when dealing with congruence of closed programs in Section **??**.

5 Determining program equivalences and symmetry

Given two turtle programs, how can one determine in what way(s) they are equivalent? We will answer this question for equivalence ($\equiv$) and motion equivalence ($\overset{\mu}{\equiv}$) in general. For trace equivalence ($\overset{\tau}{\equiv}$) and congruence ($\overset{c}{\equiv}$) we restrict ourselves to simple programs, treating the open and properly closed cases separately. When one is interested in the shape described by a turtle program, the congruence relationship is of primary importance. In particular, the ways in which a program is congruent to itself constitute the shape's symmetries. Symmetry is self-congruence.

On account of the Standardisation Theorem, we already know

$$p \equiv q \quad \Longleftrightarrow \quad \sigma(p) = \sigma(q) \ . \tag{64}$$

That is, to answer the question whether turtle programs p and q are equivalent we just compare their standard forms. This is particularly easy when p and q are already in standard form. Similarly, motion equivalence was handled through (**??**), that is, through the μ-standard form with $\psi_n = \varphi_n = \psi_{n+1} = 0$:

$$p \overset{\mu}{\equiv} q \quad \Longleftrightarrow \quad \sigma_\mu(p) = \sigma_\mu(q) \ . \tag{65}$$

Trace equivalence in general is not so easy to establish. For simple open programs we can make use of the Unique Motion Theorem, because in that case trace equivalence coincides with motion equivalence. For simple closed programs, we can apply the Two-Motion Theorem. This involves the reverse motion of p, which, unfortunately, is not directly available for comparison to the motion of q. However, we can define a reversal operator on turtle programs to obtain the reverse motion of p as the motion of the reversal of p, provided that p is *properly* closed.

5.1 Program reversal

For turtle program p, we define the **reversal of** p, denoted by $rev(p)$, as the program obtained by reversing the order of the constituting $\mathcal{M}$, $\mathcal{T}$, and $\mathcal{R}$ commands, *and* reversing the signs of the parameters of these commands. The reversal operator rev is thus characterized by the following properties:

$$\begin{aligned} rev(\mathcal{I}) &= \mathcal{I} \\ rev(\mathcal{M}(d)) &= \mathcal{M}(-d) \\ rev(\mathcal{T}(\varphi)) &= \mathcal{T}(-\varphi) \\ rev(\mathcal{R}(\psi)) &= \mathcal{R}(-\psi) \\ rev(p\ q) &= rev(q)\ rev(p) \end{aligned}$$

The trace of the reversal of p is obtained by imagining p's trace attached to the turtle in its final state and moving the turtle with attached trace to the initial state, thereby translating and rotating p's trace. Therefore, program reversal preserves shape, that is, a program is congruent to its reversal:

$$p \stackrel{c}{\equiv} rev(p) \tag{66}$$

It is its own inverse:

$$rev(rev(p)) = p \tag{67}$$

Program reversal preserves each of the properties of being trivial, closed, properly closed, and simple. If p is trivial or properly closed, then the reversal of p generates the same trace as p, that is, $p \stackrel{\tau}{\equiv} rev(p)$. However, there are other programs that are trace equivalent to their reversal, such as $\mathcal{M}(1)\ \mathcal{T}(180°)$:

$$\begin{aligned} rev(\mathcal{M}(1)\ \mathcal{T}(180°)) &\equiv \mathcal{T}(-180°)\ \mathcal{M}(-1) \\ &\equiv \mathcal{T}(180°)\ \mathcal{M}(-1) \\ &\equiv \mathcal{M}(1)\ \mathcal{T}(180°) \end{aligned}$$

Reversal also preserves equivalence:

$$\text{if } p \equiv q \text{ then } rev(p) \equiv rev(q) \tag{68}$$

We state without proof:

> **Reverse Motion Theorem** Let p be a properly closed turtle program. The reverse motion of p equals the motion of the reversal of p:
>
> $$\widetilde{\mu}_p \quad = \quad \mu_{rev(p)}\ .$$

Note that proper closure is a necessary assumption, because otherwise reversal rotates the trace.

We now have the ingredients to determine trace equivalence of simple properly closed programs p and q, by reducing it to motion equivalence:

$$p \stackrel{\tau}{\equiv} q \quad \Longleftrightarrow \quad p \stackrel{\mu}{\equiv} q \ \vee\ rev(p) \stackrel{\mu}{\equiv} q \tag{69}$$

that is, when p or the reversal of p is motion equivalent to q.

To apply this answer in practice, it would be useful to determine the standard form of the reversal of p from the standard form of p. In general, the reversal of a program in standard form need not be in standard form. First of all, the reversal of $\mathcal{S}(d, \psi, \varphi)$ is $\mathcal{T}(-\varphi)\ \mathcal{R}(-\psi)\ \mathcal{M}(-d)$, which is not a segment command, but this can be remedied by commuting adjacent $\mathcal{R}$ and $\mathcal{M}$ commands. More importantly, positive distances and turn angles in p will become negative in the reversal, and this is not allowed in the standard form by constraints C1 and C3. There is, however, a straightforward way to obtain $\sigma(rev(p))$ from $\sigma(p)$, using the program $\mathcal{H}$ defined earlier as $\mathcal{T}(180°)\ \mathcal{R}(180°)$.

First, we derive some further properties of $\mathcal{H}$, concerning how $\mathcal{H}$ behaves in combination with the basic commands:

$\mathcal{H}\ \mathcal{M}(d)$	$\mathcal{H}\ \mathcal{T}(\varphi)$	$\mathcal{H}\ \mathcal{R}(\psi)$
$\mathcal{T}(180°)\ \mathcal{R}(180°)\ \mathcal{M}(d)$	$\mathcal{T}(180°)\ \mathcal{R}(180°)\ \mathcal{T}(\varphi)$	$\mathcal{T}(180°)\ \mathcal{R}(180°)\ \mathcal{R}(\psi)$
$\mathcal{T}(180°)\ \mathcal{M}(d)\ \mathcal{R}(180°)$	$\mathcal{T}(180°)\ \mathcal{T}(-\varphi)\ \mathcal{R}(180°)$	$\mathcal{T}(180°)\ \mathcal{R}(\psi)\ \mathcal{R}(180°)$
$\mathcal{M}(-d)\ \mathcal{T}(180°)\ \mathcal{R}(180°)$	$\mathcal{T}(-\varphi)\ \mathcal{T}(180°)\ \mathcal{R}(180°)$	$\mathcal{R}(-\psi)\ \mathcal{T}(180°)\ \mathcal{R}(180°)$
$\mathcal{M}(-d)\ \mathcal{H}$	$\mathcal{T}(-\varphi)\ \mathcal{H}$	$\mathcal{R}(-\psi)\ \mathcal{H}$

In each column, the programs are equivalent, as can easily be verified from the basic properties. Thus, $\mathcal{H}$ can be used to reverse the signs in each of the basic commands.

We derive, in a sequence of equivalences, two further properties, which will turn out to be useful:

$\mathcal{H}\ \mathcal{R}(\psi)\ \mathcal{T}(\varphi)$	$\mathcal{H}\ \mathcal{R}(-\psi)\ \mathcal{T}(-\varphi)$
$\mathcal{T}(180°)\ \mathcal{R}(180°)\ \mathcal{R}(\psi)\ \mathcal{T}(\varphi)$	$\mathcal{T}(180°)\ \mathcal{R}(180°)\ \mathcal{R}(-\psi)\ \mathcal{T}(-\varphi)$
$\mathcal{T}(180°)\ \mathcal{R}(\psi)\ \mathcal{T}(-\varphi)\ \mathcal{R}(180°)$	$\mathcal{R}(180°)\ \mathcal{T}(180°)\ \mathcal{R}(-\psi)\ \mathcal{T}(-\varphi)$
$\mathcal{R}(-\psi)\ \mathcal{T}(180°)\ \mathcal{T}(-\varphi)\ \mathcal{R}(180°)$	$\mathcal{R}(180°)\ \mathcal{R}(\psi)\ \mathcal{T}(180°)\ \mathcal{T}(-\varphi)$
$\mathcal{R}(-\psi)\ \mathcal{T}(180° - \varphi)\ \mathcal{R}(180°)$	$\mathcal{R}(\psi + 180°)\ \mathcal{T}(180° - \varphi)$

Assuming we have the standard form for program p, say

$$\sigma(p) = \mathcal{R}(\psi_0)\ \mathcal{T}(\varphi_0)\ \mathcal{M}(d_1)\ \mathcal{R}(\psi_1)\ \mathcal{T}(\varphi_1)\ \ldots\ \mathcal{M}(d_n)\ \mathcal{R}(\psi_n)\ \mathcal{T}(\varphi_n)\ \mathcal{R}(\psi_{n+1})$$

with parameters satisfying constraints C1 through C5, we will rewrite $rev(p)$, while preserving equivalence, into standard form:

$rev(p)$
  { (**??**), using $p \equiv \sigma(p)$ }
$rev(\sigma(p))$
  { assumption about $\sigma(p)$ }
$rev(\mathcal{R}(\psi_0)\ \mathcal{T}(\varphi_0)\ \mathcal{M}(d_1)\ \mathcal{R}(\psi_1)\ \mathcal{T}(\varphi_1)\ \ldots\ \mathcal{M}(d_n)\ \mathcal{R}(\psi_n)\ \mathcal{T}(\varphi_n)\ \mathcal{R}(\psi_{n+1}))$
  { definition of rev }
$\mathcal{R}(-\psi_{n+1})\ \mathcal{T}(-\varphi_n)\ \mathcal{R}(-\psi_n)\ \mathcal{M}(-d_n)\ \ldots$
    $\ldots\ \mathcal{T}(-\varphi_1)\ \mathcal{R}(-\psi_1)\ \mathcal{M}(-d_1)\ \mathcal{T}(-\varphi_0)\ \mathcal{R}(-\psi_0)$
  { $\mathcal{R}$ and $\mathcal{M}$ commute, n occurrences }
$\mathcal{R}(-\psi_{n+1})\ \mathcal{T}(-\varphi_n)\ \mathcal{M}(-d_n)\ \mathcal{R}(-\psi_n)\ \ldots$
    $\ldots\ \mathcal{T}(-\varphi_1)\ \mathcal{M}(-d_1)\ \mathcal{R}(-\psi_1)\ \mathcal{T}(-\varphi_0)\ \mathcal{R}(-\psi_0)$
  { $\mathcal{H}$ is its own inverse }
$\mathcal{H}\ \mathcal{H}\ \mathcal{R}(-\psi_{n+1})\ \mathcal{T}(-\varphi_n)\ \mathcal{M}(-d_n)\ \mathcal{R}(-\psi_n)\ \ldots$
    $\ldots\ \mathcal{T}(-\varphi_1)\ \mathcal{M}(-d_1)\ \mathcal{R}(-\psi_1)\ \mathcal{T}(-\varphi_0)\ \mathcal{R}(-\psi_0)$
  { $\mathcal{H}$ changes sign of basic commands }
$\mathcal{H}\ \mathcal{R}(\psi_{n+1})\ \mathcal{T}(\varphi_n)\ \mathcal{M}(d_n)\ \mathcal{R}(\psi_n)\ \ldots\ \mathcal{T}(\varphi_1)\ \mathcal{M}(d_1)\ \mathcal{H}\ \mathcal{R}(-\psi_1)\ \mathcal{T}(-\varphi_0)\ \mathcal{R}(-\psi_0)$
  { the two further properties of $\mathcal{H}$ }
$\mathcal{R}(-\psi_{n+1})\ \mathcal{T}(180° - \varphi_n)\ \mathcal{R}(180°)\ \mathcal{M}(d_n)\ \mathcal{R}(\psi_n)\ \ldots$
    $\ldots\ \mathcal{T}(\varphi_1)\ \mathcal{M}(d_1)\ \mathcal{R}(\psi_1 + 180°)\ \mathcal{T}(180° - \varphi_0)\ \mathcal{R}(-\psi_0)$
  { $\mathcal{R}$ and $\mathcal{M}$ commute (one occurence); merge adjacent $\mathcal{R}$ }
$\mathcal{R}(-\psi_{n+1})\ \mathcal{T}(180° - \varphi_n)\ \mathcal{M}(d_n)\ \mathcal{R}(\psi_n + 180°)\ \ldots$
    $\ldots\ \mathcal{T}(\varphi_1)\ \mathcal{M}(d_1)\ \mathcal{R}(\psi_1 + 180°)\ \mathcal{T}(180° - \varphi_0)\ \mathcal{R}(-\psi_0)$
  { defining d'_i, ψ'_i, and φ'_i }
$\mathcal{R}(\psi'_0)\ \mathcal{T}(\varphi'_0)\ \mathcal{M}(d'_1)\ \mathcal{R}(\psi'_1)\ \ldots\ \mathcal{T}(\varphi'_{n-1})\ \mathcal{M}(d'_n)\ \mathcal{R}(\psi'_n)\ \mathcal{T}(\varphi'_n)\ \mathcal{R}(\psi'_{n+1})$
  { definition of $\mathcal{S}$ }
$\mathcal{R}(\psi'_0)\ \mathcal{T}(\varphi'_0)\ \mathcal{S}(d'_1, \psi'_1, \varphi'_1)\ \ldots\ \mathcal{S}(d'_n, \psi'_n, \varphi'_n)\ \mathcal{R}(\psi'_{n+1})$

where we have

$$\begin{aligned} d'_i &= d_{n+1-i} \\ \psi'_i &= \begin{cases} -\psi_{n+1-i} & \text{for } i = 0, n+1 \\ \psi_{n+1-i} + 180^\circ & \text{for } i = 1, n \\ \psi_{n+1-i} & \text{for } 1 < i < n \end{cases} \\ \varphi'_i &= \begin{cases} 180^\circ - \varphi_{n-i} & \text{for } i = 0, n \\ \varphi_{n-i} & \text{for } 1 \leq i < n \end{cases} \end{aligned}$$

Some cleaning up is still required:

- For C2, bring ψ'_i into the range $(-180^\circ, 180^\circ]$ for $i = 0, 1, n, n+1$;
- For C5 (if $\varphi'_i = 0 \pmod{180^\circ}$ then $\psi'_i = 0$, for $0 \leq i \leq n$), it may be necessary to 'percolate' non-zero ψ'_i up the chain over $\varphi'_i = 0 \pmod{180^\circ}$.

This finalises the construction.

For $1 < i < n$, segment $\mathcal{S}(d_i, \psi_i, \varphi_i)$ transformed into $\mathcal{S}(d_{n+1-i}, \psi_{n+1-i}, \varphi_{n-i})$. Note the shift by one in the index for the turn angles.

We now see that the c-standard form for $rev(p)$ is more easily obtained from the c-standard form for simple p:

$$\begin{aligned} \sigma_c(p) &= \mathcal{S}(d_1, \psi_1, \varphi_1) \ \ldots \ \mathcal{S}(d_n, \psi_n, \varphi_n) \\ \sigma_c(rev(p)) &= \mathcal{S}(d'_1, \psi'_1, \varphi'_1) \ \ldots \ \mathcal{S}(d'_n, \psi'_n, \varphi'_n) \end{aligned}$$

where we have

$$\begin{aligned} d'_i &= d_{n+1-i} \\ \psi'_i &= \psi_{n+1-i} \\ \varphi'_i &= \varphi_{n-i} \end{aligned}$$

Note that by definition we have $\psi_0 = \varphi_0 = \psi_1 = \psi_n = \varphi_n = \psi_{n+1} = 0$, and hence also $\psi'_1 = \psi'_n = \varphi'_n = 0$. If p is not simple, then $\varphi_i = 180^\circ$ can occur, and this will require 'percolation' of $\psi_i = 0$ in the above form.

As an example, consider the program

$$\mathcal{R}(0)\ \mathcal{T}(45^\circ)\ \mathcal{S}(1, 90^\circ, 120^\circ)\ \mathcal{S}(2, 0, 150^\circ)\ \mathcal{S}(\sqrt{3}, 45^\circ, 90^\circ)\ \mathcal{R}(-90^\circ) \tag{70}$$

It is properly closed and generates a $(30^\circ, 60^\circ, 90^\circ)$ triangle in the plane $x = y$ (see Figure **??**). The reversal of this program is

$$\mathcal{R}(90^\circ)\ \mathcal{T}(90^\circ)\ \mathcal{S}(\sqrt{3}, -135^\circ, 150^\circ)\ \mathcal{S}(2, 0, 120^\circ)\ \mathcal{S}(1, -90^\circ, 135^\circ)\ \mathcal{R}(0) \tag{71}$$

These programs are trace equivalent. Their c-standard forms are respectively:

$$\mathcal{S}(1, 0, 120^\circ)\ \mathcal{S}(2, 0, 150^\circ)\ \mathcal{S}(\sqrt{3}, 0, 0)$$

$$\mathcal{S}(\sqrt{3}, 0, 150^\circ)\ \mathcal{S}(2, 0, 120^\circ)\ \mathcal{S}(1, 0, 0)$$

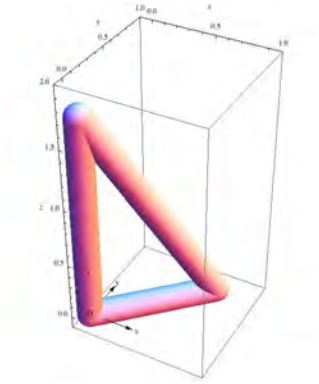


Figure 8: *The same trace generated by programs (??) and (??)*

5.2 Congruence of open simple programs

We now consider the congruence of two open simple programs p, q. Because the programs are simple and open, there are only two kinds of mappings possible between their traces:

1. the initial positions are paired and so are the final positions, or
2. the initial position of one trace is paired with the final position of the other trace and conversely.

The second case is reduced to the first case by comparing $rev(p)$ with q. To compare the traces of p and q when the initial positions are paired (the first case), we need to consider only rotations, reflections, and their combination. Rotations are taken care of by comparing their c-standard forms, since rotating p's trace is accomplished by an $\mathcal{R}\ \mathcal{T}\ \mathcal{R}$ prefix. Taking the c-standard forms ensures that the first two segments are aligned (if present).

What remains to be considered are reflections and their combination with rotation. For turtle program p, we define the **reflection of** p, denoted by $refl(p)$, as the program obtained by negating all roll angles. Reflection is thus characterized by the following properties

$$\begin{aligned} refl(\mathcal{I}) &= \mathcal{I} \\ refl(\mathcal{M}(d)) &= \mathcal{M}(d) \\ refl(\mathcal{T}(\varphi)) &= \mathcal{T}(\varphi) \\ refl(\mathcal{R}(\psi)) &= \mathcal{R}(-\psi) \\ refl(p\ q) &= refl(p)\ refl(q) \end{aligned}$$

The trace of the reflection of p is obtained by reflecting p's trace in the (x, y)-plane, thereby negating all z co-ordinates. If also all turn angles would be negated, then that would amount to reflection in the (x, z)-plane (negating y co-ordinates). But that would make it harder to calculate the standard form for the reflection.

Program reflection preserves shape, i.e., a program is congruent to its reflection:

$$p \overset{c}{\equiv} refl(p) \tag{72}$$

It is its own inverse:

$$refl(refl(p)) = p \tag{73}$$

Program reflection preserves each of the properties of being trivial, closed, properly closed, and simple. Reflection also preserves equivalence:

$$\text{if } p \equiv q \text{ then } refl(p) \equiv refl(q) \tag{74}$$

Assuming we have the standard form for program p, say

$$\sigma(p) = \mathcal{R}(\psi_0)\ \mathcal{T}(\varphi_0)\ \mathcal{S}(d_1, \psi_1, \varphi_1)\ \ldots\ \mathcal{S}(d_n, \psi_n, \varphi_n)\ \mathcal{R}(\psi_{n+1})$$

with parameters satisfying constraints C1 through C5, it is straightforward to see that $refl(p)$ has as standard form

$$\mathcal{R}(\psi'_0)\ \mathcal{T}(\varphi'_0)\ \mathcal{S}(d'_1, \psi'_1, \varphi'_1)\ \ldots\ \mathcal{S}(d'_n, \psi'_n, \varphi'_n)\ \mathcal{R}(\psi'_{n+1})$$

where

$$\begin{aligned} d'_i &= d_i \\ \psi'_i &= \begin{cases} -\psi_i & \text{for } \psi_i \neq 180^\circ \\ \psi_i & \text{for } \psi_i = 180^\circ \end{cases} \\ \varphi'_i &= \varphi_i \end{aligned}$$

The resulting form satisfies all constraints C1 through C5. Note that the case distinction for ψ' is needed to guarantee C2. This finalises the construction.

Similarly, the c-standard form for $refl(p)$ is obtained from the c-standard form for p as follows:

$$\begin{aligned} \sigma_c(p) &= \mathcal{S}(d_1, \psi_1, \varphi_1)\ \ldots\ \mathcal{S}(d_n, \psi_n, \varphi_n) \\ \sigma_c(refl(p)) &= \mathcal{S}(d_1, -\psi_1, \varphi_1)\ \ldots\ \mathcal{S}(d_n, -\psi_n, \varphi_n) \end{aligned}$$

where roll angle -180° is replaced by 180°. Note that by definition we have $\psi_1 = \psi_n = \varphi_n = 0$, and hence also $-\psi_1 = -\psi_n = 0$.

To summarize, we have obtained the following decision procedure. Open simple programs p, q are congruent if and only if at least one of the following four equalities holds:

$$\begin{aligned} \sigma_c(p) &= \sigma_c(q) \\ \sigma_c(rev(p)) &= \sigma_c(q) \\ \sigma_c(refl(p)) &= \sigma_c(q) \\ \sigma_c(refl(rev(p))) &= \sigma_c(q) \end{aligned}$$

Note that in case multiple equalities hold, this points at a symmetry in the trace (also see Section **??**).

5.3 Congruence of properly closed simple programs

For a (properly) closed simple program, the decision procedure for open simple programs can be applied but it does not suffice. It will not yield false positives, but false negatives are possible. The reason is that when looking for a mapping

between two closed paths, the initial position of one path need not be mapped to the initial or final position of the other path. In fact, the initial position need not even be a corner (where non-collinear segments meet), as illustrated by program (**??**). However, it is the case that each corner needs to be mapped to a corner, preserving their cyclic order. Such a cyclic shift corresponds to a rotation.

As noted in Section **??**, properly closed programs still have some issues around the common initial and final position, which hampers the detection of congruences. Furthermore, the c-standard form of a properly closed program is, in general, not properly closed. Therefore, we will introduce another standard form especially for properly closed programs.

The new definition will rely on the following theorem.

> **Cyclic Permutation Congruence Theorem** If program $p\ q$, that is, p followed by q, is properly closed, then so is $q\ p$, and we have
>
> $$p\ q \stackrel{c}{\equiv} q\ p \tag{75}$$

Program $q\ p$ is a cyclic permutation of program $p\ q$, and its trace is obtained from that of $p\ q$ by a translation (if $p \equiv \mathcal{M}(d)$), a rotation (about z-axis if $p \equiv \mathcal{T}(\varphi)$, or about x-axis if $p \equiv \mathcal{R}(\psi)$), or a combination of these. We will not provide a proof (hint: consider program $p\ q\ p$).

For properly closed program p, we define the ***cc*-standard form of** p, denoted by $\sigma_{cc}(p)$, as follows. Let the strict standard form of p be

$$\mathcal{R}(\psi_0)\ \mathcal{T}(\varphi_0)\ \mathcal{M}(d_1)\ \mathcal{R}(\psi_1)\ \mathcal{T}(\varphi_1)\ \ldots\ \mathcal{M}(d_n)\ \mathcal{R}(\psi_n)\ \mathcal{T}(\varphi_n)\ \mathcal{R}(\psi_{n+1})$$

with parameters satisfying constraints C1 through C5, and C4a. According to the Cyclic Permutation Congruence Theorem, the program

$$\mathcal{M}(d_1)\ \mathcal{R}(\psi_1)\ \mathcal{T}(\varphi_1)\ \ldots\ \mathcal{M}(d_n)\ \mathcal{R}(\psi_n)\ \mathcal{T}(\varphi_n)\ \mathcal{R}(\psi_{n+1})\ \mathcal{R}(\psi_0)\ \mathcal{T}(\varphi_0)$$

is also properly closed and congruent to p. Let $\mathcal{R}(\psi)\ \mathcal{T}(\varphi)\ \mathcal{R}(\psi')$ be equivalent to

$$\mathcal{R}(\psi_n)\ \mathcal{T}(\varphi_n)\ \mathcal{R}(\psi_{n+1})\ \mathcal{R}(\psi_0)\ \mathcal{T}(\varphi_0)$$

on account of Property **??**, with $0 \le \varphi \le 180^\circ$ and $\psi = 0$ if $\varphi = 0 \pmod{180^\circ}$. We then obtain

$$\begin{aligned}
p &\stackrel{c}{\equiv} \mathcal{M}(d_1)\ \mathcal{R}(\psi_1)\ \mathcal{T}(\varphi_1)\ \ldots\ \mathcal{M}(d_n)\ \mathcal{R}(\psi)\ \mathcal{T}(\varphi)\ \mathcal{R}(\psi') \\
&\qquad \{\ \text{CPC Theorem}\ \} \\
&\stackrel{c}{\equiv} \mathcal{R}(\psi')\ \mathcal{M}(d_1)\ \mathcal{R}(\psi_1)\ \mathcal{T}(\varphi_1)\ \ldots\ \mathcal{M}(d_n)\ \mathcal{R}(\psi)\ \mathcal{T}(\varphi) \\
&\qquad \{\ \mathcal{M} \text{ and } \mathcal{R} \text{ commute, merge adjacent } \mathcal{R}\ \} \\
&\stackrel{c}{\equiv} \mathcal{M}(d_1)\ \mathcal{R}(\psi' + \psi_1)\ \mathcal{T}(\varphi_1)\ \ldots\ \mathcal{M}(d_n)\ \mathcal{R}(\psi)\ \mathcal{T}(\varphi)
\end{aligned}$$

If $\varphi = 180^\circ$, then p is not simple. If $\varphi = 0$ (hence, also $\psi = 0$), then we can rewrite further

$$p \stackrel{c}{\equiv} \mathcal{M}(d_n + d_1)\ \mathcal{R}(\psi' + \psi_1)\ \mathcal{T}(\varphi_1)\ \ldots$$

This way, we obtain the *cc*-standard form of p:

$$\sigma_{cc}(p) \quad = \quad \mathcal{M}(d'_1)\ \mathcal{R}(\psi'_1)\ \mathcal{T}(\varphi'_1)\ \ldots\ \mathcal{M}(d'_n)\ \mathcal{R}(\psi'_n)\ \mathcal{T}(\varphi'_n) \qquad (76)$$

where the parameters satisfy these constraints ($1 \leq i \leq n$):

CC1: $d'_i > 0$,

CC2: $-180^\circ < \psi'_i \leq 180^\circ$,

CC3: $0 \leq \varphi'_i \leq 180^\circ$,

CC4: $\varphi'_i \neq 0$,

CC5: if $\varphi'_i = 180^\circ$ then $\psi'_i = 0$.

Furthermore, if p is simple, then

CC4a: $\varphi'_i \neq 180^\circ$.

Constraints CC3, CC4, and CC4a, can be combined into $0 < \varphi'_i < 180^\circ$.

Thus, a program in *cc*-standard form is a sequence of segment commands. Its first segment lies along the x^+-axis and its last segment lies in the (x, y^+)-half-plane. The latter can be understood by traversing the program in reverse, keeping in mind that it is properly closed. Consequently, its initial position (the origin) is a corner, whose angle is determined by φ'_n (viz., its supplement). The torsion of the last segment is given by ψ'_n. For example, the *cc*-standard form of both programs (**??**) and (**??**), which are properly closed, simple, and in strict standard form, is

$$\mathcal{S}(d, 0, 120^\circ)\ \mathcal{S}(d, 0, 120^\circ)\ \mathcal{S}(d, 0, 120^\circ) \qquad (77)$$

Note that a program of the form (**??**) satisfying constraints CC1 through CC5 need not be properly closed. However, if its last segment lies in the (x, y^+)-plane, then it can be made properly closed by appropriately tuning the last roll and turn angle.

From the definition of *refl*, we see that the reflection of a program in *cc*-standard form is itself almost in *cc*-standard form. We only need to replace roll angle -180° by 180°. This is similar to the reflection of the *c*-standard form.

For program reversal, it works slightly differently. The reversal of a simple program in *cc*-standard form can be put in *cc*-standard form as follows.

$$\begin{aligned} p \quad &= \quad \mathcal{S}(d_1, \psi_1, \varphi_1)\ \mathcal{S}(d_2, \psi_2, \varphi_2)\ \ldots\ \mathcal{S}(d_n, \psi_n, \varphi_n) \\ \sigma_{cc}(rev(p)) \quad &= \quad \mathcal{S}(d_n, \psi_n, \varphi_{n-1})\ \ldots\ \mathcal{S}(d_2, \psi_2, \varphi_1)\ \mathcal{S}(d_1, \psi_1, \varphi_n) \end{aligned}$$

If p is not simple, then $\varphi_i = 180^\circ$ will require 'percolation' of $\psi_i = 0$.

For properly closed turtle program p in *cc*-standard form, we define the **shift of** p by

$$\begin{aligned} p \quad &= \quad \mathcal{S}(d_1, \psi_1, \varphi_1)\ \mathcal{S}(d_2, \psi_2, \varphi_2)\ \ldots\ \mathcal{S}(d_n, \psi_n, \varphi_n) \\ \mathit{shift}(p) \quad &= \quad \mathcal{S}(d_2, \psi_2, \varphi_2)\ \ldots\ \mathcal{S}(d_n, \psi_n, \varphi_n)\ \mathcal{S}(d_1, \psi_1, \varphi_1) \end{aligned}$$

The shift of p is then also in *cc*-standard form, and it is congruent to p. When *shift* is applied k times, we denote this by shift^k.

Thus, we have established the following decision procedure. Properly closed simple programs p, q are congruent if and only if at least one of the following equalities holds:

$$\begin{aligned} \sigma_{cc}(\mathit{shift}^k(p)) &= \sigma_{cc}(q) \\ \sigma_{cc}(\mathit{rev}(\mathit{shift}^k(p))) &= \sigma_{cc}(q) \\ \sigma_{cc}(\mathit{refl}(\mathit{shift}^k(p))) &= \sigma_{cc}(q) \\ \sigma_{cc}(\mathit{refl}(\mathit{rev}(\mathit{shift}^k(p)))) &= \sigma_{cc}(q) \end{aligned}$$

where k ranges from 0 to $n-1$ with n the number of segments in $\sigma_{cc}(p)$. If $\sigma_{cc}(p)$ and $\sigma_{cc}(q)$ do not have the same number of segments, then they are certainly not congruent. This calculation requires at most $4n$ comparisons. Note that in case multiple equalities hold, this points at a symmetry in the trace (also see Section **??**).

5.4 Summary of Equivalence Determination

Table **??** summarizes how the various equivalences between turtle programs can be determined using appropriate standard forms and transformations.

Equivalence	**Condition on** p, q	**Criterion**
$p \equiv q$		$\sigma(p) = \sigma(q)$
$p \overset{\mu}{\equiv} q$		$\sigma_\mu(p) = \sigma_\mu(q)$
$p \overset{\tau}{\equiv} q$	simple, open	$p \overset{\mu}{\equiv} q$
$p \overset{\tau}{\equiv} q$	simple, properly closed	$p \overset{\mu}{\equiv} q \ \vee \ \mathit{rev}(p) \overset{\mu}{\equiv} q$
$p \overset{c}{\equiv} q$	simple, open	compare σ_c of p, $\mathit{rev}(p)$, $\mathit{refl}(p)$, $\mathit{refl}(\mathit{rev}(p))$, and q
$p \overset{c}{\equiv} q$	simple, properly closed	compare σ_{cc} of $\mathit{shift}^k(p)$, $\mathit{rev}(\mathit{shift}^k(p))$, $\mathit{refl}(\mathit{shift}^k(p))$, $\mathit{refl}(\mathit{rev}(\mathit{shift}^k(p)))$, and q

Table 3: How to determine the various equivalences among programs

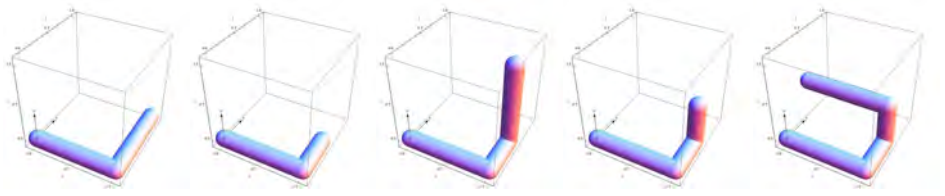
5.5 Symmetry

A symmetry of a trace is any translation, rotation, reflection, or combination of these that leaves the trace invariant (unchanged). In particular, the identity is a symmetry of every trace. In fact, the set of symmetries forms a group under the usual composition operator.

A single point (generated by a trivial program) and a single line segment (e.g. generated by a single move command) have an infinite number of symmetries, viz. every rotation whose axis contains the trace, and all combinations with a reflection. Other traces generated by turtle programs have a finite set of

symmetries. In case of simple traces, these symmetries can be determined by testing the generating program for congruence with itself, distinguishing open and properly closed programs. Every satisfied equality in the decision procedure is called a **symmetry of the program** and corresponds to a symmetry of the generated trace.

Figure **??** shows five open paths with their generating programs in *c*-standard form and five different symmetry groups. A *refl*-symmetry means that the path is its own mirror image without switching the endpoints. Consequently, a path has *refl* as symmetry if and only if it is planar. A *rev*-symmetry means that the path is rotation symmetric, switching the endpoints. A *refl rev*-symmetry does both, that is, it is a reflection that switches the endpoints.



$\mathcal{S}(1,0,90°)$ $\mathcal{S}(1,0,0)$	$\mathcal{S}(1,0,90°)$ $\mathcal{S}(1/2,0,0)$	$\mathcal{S}(1,0,90°)$ $\mathcal{S}(1/2,90°,90°)$ $\mathcal{S}(1,0,0)$	$\mathcal{S}(1,0,90°)$ $\mathcal{S}(1/2,90°,90°)$ $\mathcal{S}(1/2,0,0)$	$\mathcal{S}(1,0,90°)$ $\mathcal{S}(1/2,90°,90°)$ $\mathcal{S}(1/2,-90°,90°)$ $\mathcal{S}(1,0,0)$
I | *refl* *rev* | *refl rev*	I | *refl* |	I | *rev* |	I | |	I | | *refl rev*

Figure 9: *Five open paths having different symmetry groups (I is the identity)*

The symmetries of the artwork presented in Section **??** can now readily be determined. Since for these artworks the turn angles are constant over the path, it is particularly easy to determine the reversal: just reverse the description. For the reflection, one just has to replace every R_d by L_d and every L_d by R_d (similarly for $R'_d \leftrightarrow L'_d$ in the Figure Eight Knot). Table **??** lists the symmetries of the artworks.

Object	**Symmetries**	#
Spiralosaurus	$I, shift^{12}, shift^{24}, rev\ shift^{6}, rev\ shift^{18}, rev\ shift^{30}$	6
Braidwork	$I, shift^{10}, shift^{20}, refl\ shift^{5}, refl\ shift^{15}, refl\ shift^{25}$	6
Borromean Polylink	$I, rev\ shift^{-1}$	2
Figure-Eight Knot	$I, shift^{8}, refl\ shift^{4}, refl\ shift^{12}$	4

Table 4: *Symmetries of the artwork of Section* **??***; I is the identity*

6 Software tool support for 3D turtle geometry

Software for the execution of 3D turtle programs to produce graphical output is not hard to create; see for instance (**?**). There are many implementations of

2D turtle graphics freely available, but only a few for 3D, such as the Java library *Cheloniidae* (**?**), formerly known as *Terrapin*. The demonstration project (**?**) contains a small 3D turtle graphics library for *Mathematica*. **?** presents a 3D turtle graphics module for JavaScript based on the HTML5 `canvas` element. The execution of long turtle programs has the danger that numerical errors accumulate and invalidate output after a certain number of commands.

Software can also be used for simulation and analysis of 3D turtle programs. Simulation of a turtle program can produce (approximate) numerical results to determine such properties as the final state (and hence closure) and path length. The property of being (properly) closed is a so-called *local* property, which can be checked by considering the turtle's state or state change only. In some special cases, symbolic simulation of turtle programs using computer algebra systems (CAS), such as *Mathematica* or *Maple*, can provide exact results. However, in general, the trigonometry involved in symbolic simulation quickly becomes too hard for such systems to be of much use.

Determining whether a turtle program is simple is also not easy, in general. The property of being simple is a *global* property, which involves the entire history of the turtle's state. Checking whether a program is in some standard form and determining its symmetries is feasible through the theory of the preceding section.

Useful operations to implement in software are various turtle program transformations. Putting a program in standard form is, again, not easy in general when it needs to be done exactly. But numerically it is straightforward to convert programs into standard form. Determining equivalences between programs in standard form and determining program symmetries involves the program transformations *rev*, *refl*, and *shift*, which are straightforward to implement.

7 Conclusion

We have presented a 3D variant of turtle graphics. 3D turtle graphics is very suitable for the description of path-based structures. We have illustrated this on several mathematical artworks by Koos Verhoeff. Descriptions directly based on co-ordinates often are messy and not convenient for analysis, such as determining symmetries. The analysis of a path's properties is much simpler when done in terms of turtle programs, especially when these programs are in standard form.

We have also presented the foundations of 3D turtle geometry, that is, of theorems about turtle programs and equivalence relations among such programs. To the best of our knowledge, this is a new contribution. The analysis of a shape can now be done by simple algebraic manipulations.

Finally, we discussed software tool support for 3D turtle geometry. Software can be used for execution of turtle programs to produce graphical output, but also for simulation and analysis. For instance, it can check whether programs are in standard form and determine their symmetries. Currently, we are using 3D turtle geometry to investigate regular polygons of constant torsion as introduced in (**?**).

Acknowledgments

All artwork and photographs were made by Koos Verhoeff. The illustrations were made with ACS Logo and Mathematica. We thank Juraj Hromkovič for providing a wonderful environment to carry out part of our work. We would like to acknowledge the helpful feedback from the reviewer.

References

Abelson, H. and diSessa, A. (1981) *Turtle geometry: the computer as a medium for exploring mathematics*. MIT Press, 1981.

Tipping, S. (2009) *Cheloniidae, a Java library for 3D turtle graphics*. `http://spencertipping.com/#section=cheloniidae` (accessed 28 December 2009)

Papert, S. (1960s) *Turtle Graphics*, `http://en.wikipedia.org/wiki/Turtle_graphics` (accessed 13 January 2009)

Verhoeff, T. and Verhoeff, K. (2008) "The Mathematics of Mitering and Its Artful Application", *Bridges Leeuwarden: Mathematical Connections in Art, Music, and Science, Proceedings of the Eleventh Annual Bridges Conference, in The Netherlands*, pp. 225–234, July 2008.

Verhoeff, T. (2008) "Miter Joint and Fold Joint" from *The Wolfram Demonstrations Project* `http://demonstrations.wolfram.com/MiterJointAndFoldJoint/` (accessed 13 January 2009).

Verhoeff, T. (2008) "Mitering a Closed 3D Path" from *The Wolfram Demonstrations Project* `http://demonstrations.wolfram.com/MiteringAClosed3DPath/` (accessed 13 January 2009).

Verhoeff, T. and Verhoeff, K. (2009) "Regular 3D Polygonal Circuits of Constant Torsion", *Bridges Banff: Mathematics, Music, Art, Architecture, Culture, Proceedings of the Twelfth Annual Bridges Conference, in Canada*, to appear, July 2009.

Verhoeff, T. (2009) "3D Flying Pipe-laying Turtle" from *The Wolfram Demonstrations Project* `http://demonstrations.wolfram.com/3DFlyingPipeLayingTurtle/` (accessed 30 January 2009).

Verhoeff, T. (2009) *3D Turtle Graphics Module for JavaScript Using the* `canvas` *Element*. `http://www.win.tue.nl/~wstomv/edu/javascript/tg-machine.html` (accessed 28 December 2009).

A Glossary

$\equiv$	Short for $\overset{e}{\equiv}$
$\overset{\alpha}{\equiv}$	Final-attitude equivalence of programs
$\overset{c}{\equiv}$	Congruence of programs
$\overset{e}{\equiv}$	Equivalence of programs
$\overset{\mu}{\equiv}$	Motion equivalence of programs
$\overset{\tau}{\equiv}$	Trace equivalence of programs
attitude	Determined by turtle's heading vector and normal vector
base plane	Plane containing heading and perpendicular to normal
basic command	*Move*, *Turn*, and *Roll*
closed	Final position equals initial position
δ_p	Total duration of program p's motion
$\mathit{Dive}(\theta)$	$\mathit{Turn}(90°)\,;\,\mathit{Roll}(\theta)\,;\,\mathit{Turn}(-90°)$
empty program	Program consisting of no commands
$\mathcal{H}$ (half-loop)	$\mathit{Roll}(180°)\,;\,\mathit{Turn}(180°) \equiv \mathit{Dive}(180°)$
$\mathcal{I}$	Empty program
$\mathcal{M}(d)$	Short for $\mathit{Move}(d)$
$\mathit{Move}(d)$	Command to move turtle d units forward along heading
μ_p	Motion of program p
$\widetilde{\mu}_p$	Reverse motion of program p
open	Not closed
properly closed	Final state equals initial state
$\mathcal{R}(\psi)$	Short for $\mathit{Roll}(\psi)$
$\mathit{refl}(p)$	Reflection of program p in (x, y)-plane
$\mathit{rev}(p)$	Reversal of program p
$\mathit{Roll}(\psi)$	Command to roll turtle about heading over angle ψ
$\mathcal{S}(d, \psi, \varphi)$	Short for $\mathit{Segment}(d, \psi, \varphi)$
$\mathit{Segment}(d, \psi, \varphi)$	$\mathit{Move}(d)\,;\,\mathit{Roll}(\psi)\,;\,\mathit{Turn}(\varphi)$
simple	Visiting each position once, except when closed
$\mathit{shift}(p)$	Cyclic shift of program p in *cc*-standard form over one segment
state	Determined by turtle's position and attitude
$\sigma(p)$	Standard form of program p
$\sigma_c(p)$	*c*-Standard form of program p
$\sigma_{cc}(p)$	*cc*-Standard form of properly closed program p
τ_p	Trace (path) of program p
$\mathcal{T}(d)$	Short for $\mathit{Turn}(\varphi)$
torsion angle	Dihedral angle between adjacent corner-spanning planes
trivial	Having a single point as trace
$\mathit{Turn}(\varphi)$	Command to turn turtle about normal over angle φ