Author

Dr. John H. Reif
Dept. Comp. Sci., Duke Univ., Durham, USA and Adj. Fac. of Comp., KAU, Jeddah, SA, Durham, USA

Editor

Dr. Robert A. Meyers
RAMTECH LIMITED, Larkspur, USA

This article represents the version submitted by the author.
It is currently undergoing peer review prior to full publication.

Mechanical Computing: The Computational Complexity of Physical Devices

Glossary

- *Mechanism:* A machine or part of a machine that performs a particular task computation: the use of a computer for calculation.

- *Computable*: Capable of being worked out by calculation, especially using a computer.

- *Simulation:* Used to denote both the modeling of a physical system by a computer as well as the modeling of the operation of a computer by a mechanical system; the difference will be clear from the context.

Definition of the Subject

Mechanical devices for computation appear to be largely displaced by the widespread use of microprocessor-based computers that are pervading almost all aspects of our lives. Nevertheless, mechanical devices for computation are of interest for at least three reasons:

(a) ***Historical***: The use of mechanical devices for computation is of central importance in the historical study of technologies, with a history dating back thousands of years and with surprising applications even in relatively recent times.

(b) ***Technical & Practical***: The use of mechanical devices for computation persists and has not yet been completely displaced by widespread use of microprocessor-based computers. Mechanical computers have found applications in various emerging technologies at the micro-scale that combine mechanical functions with computational and control functions not feasible by purely electronic processing. Mechanical computers also have been demonstrated at the molecular scale, and may also provide unique capabilities at that scale. The physical designs for these modern micro and molecular-scale mechanical computers may be based on the prior designs of the large-scale mechanical computers constructed in the past.

(c) ***Impact of Physical Assumptions on Complexity of Motion Planning, Design, and Simulation:*** The study of computation done by mechanical devices is also of central importance in providing lower bounds on the computational resources such as time and/or space required to simulate a mechanical system observing given physical laws. In particular, the problem of simulating the mechanical system can be shown to be *computationally hard* if a hard computational problem can be simulated by the mechanical system. A similar approach can be used to provide lower bounds on the computational resources required to solve various motion planning tasks that arise in the field of robotics. Typically, a robotic motion planning task is specified by a geometric description of the robot (or collection of robots) to be moved, its initial and final positions, the obstacles it is to avoid, as well as a model for the type of feasible motion and physical laws for the movement. The problem of planning, such as the robotic motion-planning task, can be shown to be computationally hard if a hard computational problem can be simulated by the robotic motion-planning task.

Introduction

Abstract Computing Machine Models

To gauge the computational power of a family of mechanical computers, we will use a widely known abstract computational model known as the Turing machine, defined in this section.

The Turing Machine

The *Turing machine* model formulated by Alan Turing [*1*] was the first complete mathematical model of an abstract computing machine that possessed universal computing power. The machine model has (i) a finite state transition control for logical control of the machine processing, (ii) a tape with a sequence of storage cells containing symbolic values, and (iii) a tape scanner for reading and writing values to and from the tape cells, which could be made to move (left and right) along the tape cells.

A machine model is *abstract* if the description of the machine transition mechanism or memory mechanism does not provide specification of the mechanical apparatus used to implement them in practice. Since Turing's description did not include any specification of the mechanical mechanism for executing the finite state transitions, it can't be viewed as a concrete mechanical computing machine, but instead is an abstract machine. Still it is valuable computational model, due to it simplicity and very widespread use in computational theory.

A *universal* Turing machine simulates any other Turing machine; it takes its input a pair consisting of a string providing a symbolic description of a Turing machine M and the input string x, and simulates M on input x. Because of its simplicity and elegance, the Turing machine has come to be the standard computing model used for most theoretical works in computer science. Informally, the Church-Turing hypothesis states that a Turing machine model can simulate a computation by any "reasonable" computational model (we will discuss some other reasonable computational models below).

Computational Problems

A *computational problem* is: given an input string specified by a string over a finite alphabet, determine the Boolean answer: 1 is the answer is YES, and otherwise 0. For simplicity, we generally will restrict the input alphabet to be the binary alphabet {0,1}. The *input size* of a computational problem is the number of input symbols; which is the number of bits of the binary specification of the input. (Note: It is more common to make these definitions in terms of language acceptance. A *language* is a set of strings over a given finite alphabet of symbols. A computational problem can be identified with the language consisting of all strings over the input alphabet where the answer is 1. For simplicity, we defined each complexity class as the corresponding class of problems.)

Recursively Computable Problems and Undecidable Problems

There is a large class of problems, known as *recursively computable* problems, that Turing machines compute in finite computations, that is, always halting in finite time with the answer. There are certain problems that are not recursively computable; these are called *undecidable* problems. The *Halting Problem* is: given a Turing Machine description and an input, output 1 if the Turing machine ever halts, and else output 0. Turing proved the halting problem is undecidable. His proof used a method known as a diagonalization method; it considered an enumeration of all Turing machines and inputs, and showed a contradiction occurs when a universal Turing machine attempts to solve the Halting problem for each Turing machine and each possible input.

Computational Complexity Classes

Computational complexity (see [*2*]) is the amount of computational resources required to solve a given computational problem. A *complexity class* is a family of problems, generally defined in terms of limitations on the resources of the computational model. The complexity classes of interest here will be associated with restrictions on the time (number of steps until the machine halts) and/or space (the number of tape cells used in the computation) of Turing machines. There are a number of notable complexity classes:

P is the complexity class associated with efficient computations, and is formally defined to be the set of problems solved by Turing machine computations running in time polynomial to the input size (typically, this is the number of bits of the binary specification of the input).

NP is the complexity class associated with combinatorial optimization problems which, if solved, can be easily determined to have correct solutions. It is formally defined to be the set of problems solved by Turing machine computations using nondeterministic choice running in polynomial time.

PSPACE is the complexity class defined as the set of problems solved by Turing machines running in space polynomial to the input size.

EXPTIME is the complexity class defined as the set of problems solved by Turing machine computations running in time exponential to the input size.

NP and **PSPACE** are widely considered to have instances that are not solvable in **P**, and it has been proved that **EXPTIME** has problems that are not in **P**.

Polynomial Time Reductions

A *polynomial time reduction* from a problem Q' to a problem Q is a polynomial time Turing machine computation that transforms any instance of the problem Q' into an instance of the problem Q which has an answer YES if and only if the problem Q' has an answer YES. Informally, this implies that problem Q can be used to efficiently solve the problem Q'. A problem Q is *hard* for a family F of problems if for every problem Q' in F, there is a polynomial time reduction from Q' to Q. Informally, this implies that problem Q can be used to efficiently solve *any* problem in F. A problem Q is *complete* for a family F of problems if Q

is in *C* and also hard for *F*.

Hardness Proofs for Mechanical Problems

We will later consider various mechanical problems and characterize their computation power:

- *Undecidable mechanical problem*; this was typically proven by a computable reduction from the halting problem for a universal Turing machine problem to an instance of the mechanical problem; this is equivalent to showing the mechanical problem can be viewed as a computational machine that can simulate a universal Turing machine computation.
- *Mechanical problems that are hard for **NP, PSPACE, or EXPTIME***; typically this was proved by a polynomial time reduction from the problems in the appropriate complexity class to an instance of the mechanical problem; again, this is equivalent to showing the mechanical problem can be viewed as a computational machine that can simulate a Turing machine computation in the appropriate complexity class.

The simulation proofs in either case often provide insight into the intrinsic computational power of the mechanical problem or mechanical machine.

Other Abstract Computing Machine Models

There are a number of abstract computing models discussed in this Chapter, that are equivalent, or nearly equivalent, to conventional deterministic Turing machines.

- **Reversible Turing machines** A computing device is *(logically) reversible* if each transition of its computation can be executed both in the forward direction as well as in the reverse direction, without loss of information. Landauer [*3*] showed that irreversible computations must generate heat in the computing process, and that reversible computations have the property that if executed slowly enough, can (in the limit) consume no energy in an adiabatic computation. A *reversible Turing machine* model allows the scan head to observe 3 consecutive tape symbols and to execute transitions both in the forward as well as in the reverse direction. Bennett [*4*] showed that any computing machine (e. g., an abstract machine such as a Turing machine) can be transformed to do only reversible computations, which implied that reversible computing devices are capable of universal computation. Bennett's reversibility construction required extra space to store information to insure reversibility, but this extra space can be reduced by increasing the time. Vitanyi [*5*] gave trade-offs between time and space in the resulting reversible machine. Lewis and Papadimitriou [*95*] showed that reversible Turing machines are equivalent in computational power to conventional Turing machines when the computations are bounded by polynomial time, and Crescenzi and Papadimitriou [*6*] proved a similar result when the computations are bounded by polynomial space. This implies that the definitions of the complexity classes **P** and **PSPACE** do not depend on the Turing machines being reversible or not. Reversible Turing machines are used in many of the computational complexity proofs to be mentioned involving simulations by mechanical computing machines.
- **Cellular automata** These are sets of finite state machines that are typically connected together by a grid network. There are known efficient simulations of the Turing machine by cellular automata (e. g., see Wolfram [*7*] for some known universal simulations). A number of the particle-based mechanical machines to be described are known to simulate cellular automata.
- **Randomized Turing machines** The machine can make random choices in its computation. While the use of randomized choice can be very useful in many efficient algorithms, there is evidence that randomization only provides limited additional computational power above conventional deterministic Turing machines (In particular, there are a variety of pseudo-random number generation methods proposed for producing long pseudo-random sequences from short truly random seeds, which are widely conjectured to be indistinguishable from truly random sequences by polynomial time Turning machines.) A number of the mechanical machines to be described using Brownian-motion have natural sources of random numbers.

There are also a number of abstract computing machine models that appear to be more powerful than conventional deterministic Turing machines.

- **Real-valued Turing machines** According to Blum et al. [*8*], each storage cell or register in these machines can store any real value (that may be transcendental). Operations are extended to allow infinite precision arithmetic operations on real numbers. To our knowledge, none of the analog computers that we will describe in this chapter have this power.
- **Quantum computers** A *quantum superposition* is a linear superposition of basis states; it is defined by a vector of complex amplitudes whose absolute magnitudes sum to 1. In a quantum computer, the quantum superposition of basis states is transformed in each step by a unitary transformation (this is a linear mapping that is reversible and always preserves the value of the sum of the absolute magnitudes of its inputs). The outputs of a quantum computation are read by observations that that project the quantum superposition to classical values; a given state is chosen with probability defined by the magnitude of the amplitude of that state in the quantum superposition. Feynman [*9*] and Benioff [*10*] were the first to suggest the use of quantum mechanical principles for doing computation, and Deutsch [*11*] was the first to formulate an

abstract model for quantum computing and show it was universal. Since then, there is a large body of work in quantum computing (see Gruska [*12*] and Nielsen [*13*]) and quantum information theory (see Jaeger [*14*] and Reif [*15*]). Some of the particle-based methods for mechanical computing described below make use of quantum phenomena, but generally are not considered to have the full power of quantum computers.

The Computational Complexity of Motion Planning and Simulation of Mechanical Devices

Complexity of Motion Planning for Mechanical Devices with Articulated Joints

The first known computational complexity result involving mechanical motion or robotic motion planning was in 1979 by Reif [*16*]. He considered a class of mechanical systems consisting of a finite set of connected polygons with articulated joints, which were required to be moved between two configurations in three dimensional space avoiding a finite set of fixed polygonal obstacles. To specify the movement problem (as well as the other movement problems described below unless otherwise stated), the object to be moved, as well as its initial and final positions, and the obstacles are all defined by linear inequalities with rational coefficients with a finite number of bits. He showed that this class of motion planning problems is hard for PSPACE. Since it is widely conjectured that PSPACE contains problems which are not solvable in polynomial time, this result provided the first evidence that these robotic motion planning problems were not solvable in time polynomial in *n* if the number of degrees of freedom grew with *n*. His proof involved simulating a reversible Turing machine with *n* tape cells by a mechanical device with *n* articulated polygonal arms that had to be maneuvered through a set of fixed polygonal obstacles similar to the channels in Swiss-cheese. These obstacles where devised to force the mechanical device to simulate transitions of the reversible Turing machine to be simulated, where the positions of the arms encoded the tape cell contents, and tape read/write operations were simulated by channels of the obstacles which forced the arms to be reconfigured appropriately. This class of movement problems can be solved by reduction to the problem of finding a path in a *O(n)* dimensional space avoiding a fixed set of polynomial obstacle surfaces, which can be solved by a PSPACE algorithm from Canny [*17*]. Hence this class of movement problems are PSPACE complete. (In the case where the object to be moved consists of only one rigid polygon, the problem is known as the piano mover's problem and has a polynomial time solution by Schwartz and Sharir [*18*].)

Other PSPACE Completeness Results for Mechanical Devices

There were many subsequent PSPACE completeness results for mechanical devices (two of which we mention below), which generally involved *multiple degrees of freedom*:

- **The Warehouseman's Problem** In 1984 Schwartz and Sharir [*19*] showed that moving a set of *n* disconnected polygons in two dimensions from an initial position to a final position among a finite set of fixed polygonal obstacles is PSPACE hard.

There are two classes of mechanical dynamic systems, the Ballistic machines and the Browning Machines described below, that can be shown to provide simulations of polynomial space Turing machine computations.

Ballistic Collision-Based Computing Machines and PSPACE

A *ballistic computer* (see Bennett [*20*,*21*]) is a conservative dynamical system that follows a mechanical trajectory isomorphic to the desired computation. It has the following properties:

- Trajectories of distinct ballistic computers can't be merged,
- All operations of a computational must be reversible,
- Computations, when executed at constant velocity, require no consumption of energy,
- Computations must be executed without error, and need to be isolated from external noise and heat sources.

Collision-based computing [*22*] is computation by a set of particles, where each particle holds a finite state value, and state transformations are executed at the time of collisions between particles. Since collisions between distinct pairs of particles can be simultaneous, the model allows for parallel computation. In some cases the particles can be configured to execute cellular automata computations [*23*]. Most proposed methods for *Collision-based computing* are ballistic computers as defined above. Examples of concrete physical systems for collision-based computing are:

- **The billiard ball computers** Fredkin and Toffoli [*24*] considered a mechanical computing model, the *billiard ball computer*, consisting of spherical billiard balls with polygonal obstacles, where the billiard balls were assumed to have perfect elastic collisions with no friction. They showed in 1982 that a Billiard Ball Computer, with an unbounded number of billiard balls, could simulate a reversible computing machine model that used reversible Boolean logical gates known as Toffoli gates. When restricted to a finite set of *n* spherical billiard balls, their construction provides a simulation of a polynomial space-reversible Turing machine.
- **Particle-like waves in excitable medium** Certain classes of excitable medium have discrete models that can exhibit particle-like waves which propagate through the media [*25*]. Using this phenomena, Adamatzky [*26*] gave a simulation of a universal Turing

Machine. If restricted to *n* particle-waves, his simulation provides a simulation of a polynomial space Turing Machine.

- **Soliton computers** A soliton is a wave packet that maintains a self-reinforcing shape as it travels at constant speed through a nonlinear dispersive media. A soliton computer [*27*,*28*] makes use of optical solitons to hold state, and state transformations are made by colliding solitons.

Brownian Machines and PSPACE

In a mechanical system exhibiting *fully Brownian motion*, the parts move freely and independently, up to the constraints that either link the parts together or forces the parts exert on each other. In a fully Brownian motion, the movement is entirely due to heat and there is no other source of energy driving the movement of the system. An example of a mechanical system with fully Brownian motion is a set of particles exhibiting Browning motion, as with electrostatic interaction. The rate of movement *a* of mechanical system with fully Brownian motion is determined entirely by the drift rate in the random walk of their configurations.

In other mechanical systems, known as *driven Brownian motion*, the system's movement is only partly due to heat; in addition, there is a source of energy driving the movement of the system. Examples of driven Brownian motion systems are:

- Feynman's Ratchet and Pawl [*29*], which is a mechanical ratchet system that has a driving force but that can operate reversibly.
- Polymerase enzyme, which uses ATP as fuel to drive their average movement forward, but also can operate reversibly.

There is no energy consumed by fully Brownian motion devices, whereas driven Brownian motion devices require power that grows as a quadratic function of the drive rate in which operations are executed (see Bennett [*21*]).

Bennett [*20*] provides two examples of Brownian computing machines:

- An *enzymatic machine* This is a hypothetical biochemical device that simulates a Turing machine, using polymers to store symbolic values in a manner to similar to Turing machine tapes, and uses hypothetical enzymatic reactions to execute state transitions and read/write operations into the polymer memory. Shapiro [*30*] also describes a mechanical Turing machine whose transitions are executed by hypothetical enzymatic reactions.
- A *clockwork computer* This is a mechanism with linked articulated joints, with a Swiss-cheese like set of obstacles, which force the device to simulate a Turing machine. In the case where the mechanism of Bennett's clockwork computer is restricted to have a linear number of parts, it can be used to provide a simulation of PSPACE similar that of [*16*].

Hardness Results for Mechanical Devices with a Constant Number of Degrees of Freedom

There were also additional computation complexity hardness results for mechanical devices, which only involved a *constant number of* degrees of freedom. These results exploited special properties of the mechanical systems to do the simulation.

- **Motion planning with moving obstacles** Reif and Sharir [*31*] considered the problem of planning the motion of a rigid object (the robot) between two locations, while avoiding a set of obstacles, some of which are rotating. They showed this problem is PSPACE hard. This result was perhaps surprising, since the number of degrees of freedom of movement of the object to be moved was constant. However, the simulation used the rotational movement of obstacles to force the robot to be moved only to a position that encoded all the tape cells of *M*. The simulation of a Turing machine *M* was made by forcing the object between such locations (that encoded the entire n tape cell contents of *M*) at particular times, and further forced that object to move between these locations over time in a way that simulated state transitions of *M*.

NP Hardness Results for Path Problems in Two and Three Dimensions

Shortest path problems in fixed dimensions involve only a constant number of degrees of freedom. Nevertheless, there are a number of NP hardness results for such problems. These results also led to proofs that certain physical simulations (in particular, simulation of multi-body molecular and celestial simulations) are NP hard, and therefore not likely efficiently computable with high precision.

- **Finding shortest paths in three dimensions** Consider the problem of finding a shortest path of a point in three dimensions (where distance is measured in the Euclidean metric) avoiding fixed polyhedral obstacles whose coordinates are described by rational numbers with a finite number of bits. This shortest path problem can be solved in PSPACE [*17*], but the precise complexity of the problem is an open problem. Canny and Reif [*32*] were the first to provide a hardness complexity result for this problem; they showed the problem is NP hard. Their proof used novel techniques called *free path encoding* that used 2^n homotopy equivalence classes of shortest paths. Using these techniques, they constructed exponentially many shortest path classes (with distinct homotopy) in single-source multiple-destination problems involving *O(n)* polygonal obstacles. They used each of these paths to encode a possible configuration of the nondeterministic Turing machine with *n* binary storage cells. They also provided a technique for simulating each step of the Turing machine by the use of polygonal obstacles whose edges forced a permutation of these paths that encoded the modified configuration of the Turing machine. These encodings allowed them to prove that the single-source single-destination problem in three dimensions is NP-hard. Similar free path encoding techniques were used for a number of other complexity hardness results for the mechanical simulations described below.

- **Kinodynamic planning** *Kinodynamic planning* is the task of motion planning while subject to simultaneous kinematic and dynamic constraints. The algorithms for various classes of kinodynamic planning problems were first developed in [*33*]. Canny and Reif [*32*] also used free path encoding techniques to show that two dimensional kinodynamic motion planning with a bounded velocity is NP-hard.
- **Shortest curvature-constrained path planning in two dimensions** We now consider *curvature-constrained shortest path problems* which involve finding a shortest path by a point among polygonal obstacles, where the there is an upper bound on the path curvature. A class of curvature-constrained shortest path problems in two dimensions were shown to be NP hard by Reif and Wang [*34*] by devising a set of obstacles that forced the shortest curvature-constrained path to simulate a given nondeterministic Turing machine.

PSPACE Hard Physical Simulation Problems

- **Ray Tracing with a Rational Placement and Geometry.** Ray tracing is given an optical system and the position and direction of an initial light ray, determine if the light ray reaches some given final position. This problem of determining the path of light ray through an optical system was first formulated by Newton in his book on Optics. Ray tracing has been used for designing and analyzing optical systems. It is also used extensively in computer graphics to render scenes with complex curved objects under global illumination. Reif, Tygar, and Yoshida [35] first showed in 1990 the problem of ray tracing in various three dimensional optical systems, where the optical devices either consist of reflective objects defined by quadratic equations, or refractive objects defined by linear equations, but in either case the coefficients are restricted to be rational. They showed this ray tracing problems are PSPACE hard. Their proof used free path encoding techniques for simulating a nondeterministic linear space Turing machine, where the position of the ray as it enters a reflective or refractive optical object (such as a mirror or prism face) encodes the entire memory of the Turing machine to be simulated, and further steps of the Turing machine are simulated by optically inducing appropriate modifications in the position of the ray as it enters other reflective or refractive optical objects. This result implies that the apparently simple task of highly precise ray tracing through complex optical systems is not likely to be efficiently executed by a polynomial time computer. This results for ray tracing are another example of the use of a physical system to do powerful computations. A number of subsequent papers showed the NP-hardness (recall NP is a subset of PSPACE, so NP-hardness is a weaker type of hardness result PSPACE-hardness) of various optical ray problems, such as the problem of determining if a light ray can reach a given position within a given time duration [36,37,38,39,40], optical masks [41], and ray tracing with multiple optical frequencies [42,43] (See [44] for a survey of these and related results in optical computing). A further PSPACE-hardness result for an optics problem is given in a recent paper [45] concerning ray tracing with multiple optical frequencies, with an additional concentration operation.
- **Molecular and gravitational mechanical systems.** The work of Tate and Reif [46] on the complexity of n-body simulation provides an interesting example of the use of natural physical systems to do computation. They showed that the problem of n-body simulation is PSPACE hard, and therefore not likely efficiently computable with high precision. In particular, they considered multi-body systems in three dimensions with n particles and inverse polynomial force laws between each pair of particles (e.g., molecular systems with Columbic force laws or celestial simulations with gravitational force laws). It is quite surprising that such systems can be configured to do computation. Their hardness proof made use of free path encoding techniques similar to their proof [35] of the PSPACE-hardness of ray tracing. A single particle, which we will call the *memory-encoding particle*, is distinguished. The position of a memory-encoding particle as it crosses a plane encodes the entire memory of the given Turing machine to be simulated, and further steps of the Turing machine are simulated by inducing modifications in the trajectory of the memory-encoding particle. The modifications in the trajectory of the memory-encoding particle are made by use of other particles that have trajectories that induce force fields that essentially act like force-mirrors, causing reflection-like changes in the trajectory of the memory-encoding particle. Hence highly precise n-body molecular simulation is not likely to be efficiently executed by a polynomial time computer.

A Provably Intractable Mechanical Simulation Problem: Compliant Motion Planning with Uncertainty in Control

Next, we consider compliant motion planning with uncertainty in control. Specifically, we consider a point in 3 dimensions which is commanded to move in a straight line, but whose actual motion may differ from the commanded motion, possibly involving sliding against

Share

obstacles. Given that the point initially lies in some start region, the problem is to find a sequence of commanded velocities that is guaranteed to move the point to the goal. This problem was shown by Canny and Reif [*32*] to be non-deterministic EXPTIME hard, making it the first provably intractable problem in robotics. Their proof used free path encoding techniques that exploited the uncertainty of position to encode exponential number of memory bits in a Turing machine simulation.

Undecidable Mechanical Simulation Problems

- **Motion planning with friction** Consider a class of mechanical systems whose parts consist of a finite number of rigid objects defined by linear or quadratic surface patches connected by frictional contact linkages between the surfaces. (Note: this class of mechanisms is similar to the analytical engine developed by Babbage described in the next sections, except that there are smooth frictional surfaces rather than toothed gears). Reif and Sun [*47*] proved that an arbitrary Turing machine could be simulated by a (universal) frictional mechanical system in this class consisting of a finite number of parts. The entire memory of a universal Turing machine was encoded in the rotational position of a rod. In each step, the mechanism used a construct similar to Babbage's machine to execute a state transition. The key idea in their construction is to utilize frictional clamping to allow for setting arbitrary high gear transmission. This allowed the mechanism to make state transitions for arbitrary number of steps. Simulation of a universal Turing machine implied that the movement problem is undecidable when there are frictional linkages. (A problem is *undecidable* if there is no Turing machine that solves the problem for all inputs in finite time.) It also implied that a mechanical computer could be constructed with only a constant number of parts that has the power of an unconstrained Turing machine.
- **Ray tracing with non-rational postitioning** Consider again the problem of ray tracing in a three dimensional optical systems, where the optical devices again may either consist of reflective objects defined by quadratic equations, or refractive objects defined by linear equations. Reif et al. [*35*] also proved that in the case where the coefficients of the defining equations are not restricted to be rational and include at least one irrational coefficient, then the resulting ray tracing problem could simulate a universal Turing machine, and so is undecidable. This ray tracing problem for reflective objects is equivalent to the problem of tracing the trajectory of a single particle bouncing between quadratic surfaces, which is also undecidable by this same result of [*35*]. An independent result of Moore [*48*] also showed that the problem of tracing the trajectory of a single particle bouncing between quadratic surfaces is undecidable.
- **Dynamics and Nonlinear Mappings.** Moore [49], Ditto [50] and Munakata et al [51] have also given universal Turing machine simulations of various dynamical systems with nonlinear mappings.

Concrete Mechanical Computing Devices

Mechanical computers have a very extensive history; some surveys given in Knott [52], Hartree [53],Engineering Research Associates [54], Chase [55], Martin [56], Davis [57]. Norman [58] gave an overview of the literature of mechanical calculators and other historical computers, summarizing the contributions of notable manuscripts and publications on this topic.

Mechanical Devices for Storage and Sums of Numbers

Mechanical methods such as notches on stones and bones, or knots and piles of pebbles, have been used since the Neolithic period for storing and summing integer values. One example of such a device, the abacus , which may have been developed in Babylonia approximately 5000 years ago, makes use of beads sliding on cylindrical rods to facilitate addition and subtraction calculations.

Analog Mechanical Computing Devices

Computing devices will considered here to be *analog*(as opposed to digital) if they don’t provide a method for restoring calculated values to discrete values, whereas digital devices provide restoration of calculated values to discrete values. (Note that both analog and digital computers uses some kind of physical quantity to represent values that are stored and computed, so the use of physical encoding of computational values is not necessarily the distinguishing characteristic of analog computing.) Descriptions of early analog computers are given by Horsburgh [59], Turck[60], Svoboda [61], Hartree [53],Engineering Research Associates [54] and Soroka [62]. There are a wide variety of mechanical devices used for analog computing:

- **Mechanical Devices for Astronomical and Celestial Calculation.** While we have not sufficient space in this article to fully discuss this rich history, we note that various mechanisms for predicting lunar and solar eclipses using optical illumination of configurations of stones and monoliths (for example, Stonehenge) appear to date to the Neolithic period. The Hellenistic civilization in the classical period of ancient historyseems of developed a number of analog calculating devices for astronomy calculations. A *planisphere*, which appears to have been used in the *Tetrabiblos*by Ptolemy in the 2nd century,is a simple analog calculator for determining for any given date and time the visible potion of a star chart, and consists of two disks rotating on the same pivot. *Astrolabes*are a family of analog calculators used for solving problems in spherical astronomy, often consisting of a combination of a planisphere and a dioptra (a sighting tube). An early form of an astrolabe is attributed to Hipparchus in the mid 2nd century. Other more complex mechanical mechanisms for predicting lunar and solar eclipses seems to have been have been developed in Hellenistic period. The most impressive and sophisticated known example of an ancient gear-based mechanical device from the Hellenistic periodis the *Antikythera Mechanism*, and recent research [63] provides evidence it may have been used to predict celestial events such as lunar and solar eclipses by the analog calculation of arithmetic-progression cycles. Like many other intellectual heritages, some elements of the design of such sophisticated gear-based mechanical devices may have been preserved by the Arabs at the end of that Hellenistic period, and then transmitted to the Europeans in the middle ages.

- **Planimeters.** There is a considerable history of mechanical devices that integrate curves. A *planimeter*is a mechanical device that integrates the area of the region enclosed by a two dimensional closed curve, where the curve is presented as a function of the angle from some fixed interior point within the region. One of the first known planimeters was developed by J.A. Hermann in 1814 and improved (as the polar planimeter) by J.A. Hermann in 1856. This led to a wide variety of mechanical integrators known as wheel-and-disk integrators, whose input is the angular rotation of a rotating disk and whose output, provided by a tracking wheel, is the integral of a given function of that angle of rotation. More general mechanical integrators known as ball-and-disk integrators, who’s input provided 2 degrees of freedom (the phase and amplitude of a complex function), were developed by James Thomson in 1886. There are also devices, such as the Integraph of Abdank Abakanoviez(1878) and C.V. Boys(1882), which integrate a one-variable real function of x presented as a curve y=f(x) on the Cartesian plane. Mechanical integrators were later widely used in WWI and WWII military analog computers for solution of ballistics equations, artillery calculations and target tracking. Various other integrators are described in Morin [64].

· **Harmonic Analyzers.**A *Harmonic Analyzer*is a mechanical device that calculates the coefficients of the Fourier Transform of a complex function of time such as a sound wave. Early harmonic analyzers were developed by Thomson[65] and Henrici [66] usingmultiple pulleys and spheres, known as ball-and-disk integrators.

· **Harmonic Synthesizers.**A *Harmonic Synthesizer*is a mechanical device that interpolates a function given the Fourier coefficients. Thomson (then known as Lord Kelvin) in 1886 developed [67] the first known Harmonic Analyzer that used an array of James Thomson's (his brother) ball-and-disk integrators. Kelvin's Harmonic Synthesizer made use of these Fourier coefficients to reverse this process and interpolate function values, by using a wire wrapped over the wheels of the array to form a weighted sum of their angular rotations. Kelvin demonstrated the use of these analog devices predict the tide heights of a port: first his Harmonic Analyzer calculated the amplitude and phase of the Fourier harmonics of solar and lunar tidal movements, and then his Harmonic Synthesizer formed their weighted sum, to predict the tide heights over time. Many other Harmonic Analyzers were later developed, including one by Michelson and Stratton (1898) that performed Fourier analysis, using an array of springs. Miller [68] gives a survey of these early Harmonic Analyzers. Fisher [69] made improvements to the tide predictor and later Doodson and Légé increase the scale of this design to a 42-wheel version that was used up to the early 1960s.

· **Analog Equation Solvers.**There are various mechanical devices for calculating the solution of sets of equations. Kelvin also developed one of the first known mechanical mechanisms for equation solving, involving the motion of pulleys and tilting plate that solved sets of simultaneous linear equations specified by the physical parameters of the ropes and plates.John Wilbur in the 1930s increased the scale of Kelvin’sdesign to solve nine simultaneous linear algebraic equations. Leonardo Torres Quevedo constructed various rotational mechanical devices, for determining real and complex roots of a polynomial. Svoboda [61] describes the state of art in the 1940s of mechanical analog computing devices using linkages.

· **Differential Analyzers.**A *Differential Analyzer*is a mechanical analog device using linkages for solving ordinary differential equations. Vannevar Bush [70] developed in 1931 the first Differential Analyzer at MIT that used a torque amplifier to link multiple mechanical integrators. Although it was considered a general-purpose mechanical analog computer, this device required a physical reconfiguration of the mechanical connections to specify a given mechanical problem to be solved. In subsequent Differential Analyzers, the reconfiguration of the mechanical connections was made automatic by resetting electronic relay connections. In addition to the military applications already mentioned above, analog mechanical computers incorporating differential analyzers have been widely used for flight simulations and for industrial control systems.