PERSPEX MACHINE

COPYRIGHT

Perspex Machine

James A.D.W. Anderson*
Department of Computer Science, The University of Reading, England

Abstract

We introduce the perspex machine which unifies projective geometry and the Turing machine, resulting in a supra-Turing machine. Specifically, we show that a Universal Register Machine (URM) can be implemented as a conditional series of whole numbered projective transformations. This leads naturally to a suggestion that it might be possible to construct a perspex machine as a series of pin-holes and stops. A rough calculation shows that an ultraviolet perspex machine might operate up to the petahertz range of operations per second. Surprisingly, we find that perspex space is irreversible in time, which might make it a candidate for an anisotropic spacetime geometry in physical theories. We make a bold hypothesis that the apparent irresversiblity of physical time is due to the random nature of quantum events, but suggest that a sum over histories might be achieved by sampling fluctuations in the direction of time flow. We propose an experiment, based on the Casimir apparatus, that should measure fluctuations of time flow with respect to time duration - if such fluctuations exist.

Keywords: perspex machine, Turing machine, anisotropic spacetime, Casimir effect.

1. Introduction

The *perspex machine* unifies projective geometry[6] and the Turing machine[4,10,11], hence all Turing computations, including computations in projective geometry, can be performed geometrically. The perspex machine operates in a *program space* of homogeneous co-ordinates which is identical to homogeneous models of perspective space[6,7,8] and is analogous to the machine states and data tape in a Turing machine. The perspex machine operates on simplexes of homogeneous co-ordinates called *perspexes*[1]. The name "perspex" is a portmanteau of "perspective" and "simplex". The perspex performs the two roles of being an operation and a symbol in a Turing Machine. Correspondingly, in projective geometry, the perspex performs the two roles of being a perspective transformation and a figure. The standard perspective dualities of perspexes are meaningful in program space, as is a substitution which interchanges register *locations* in program space with perspexes stored at those locations. The unification of projective geometry and the Turing machine is carried out in these geometrical terms. A unification at the level of the axioms of projective geometry may properly await the further development of the perspex machine.

In the standard homogeneous models of projective geometry[6,7,8] the homogeneous origin, which exists and is denoted by the zero vector, is punctured from perspective space[7]. This vector is called the *point at nullity*[1], Φ. Analogously, Φ exists in program space and, by default, holds the zero perspex. The machine may *read* a perspex from Φ, but any attempt to *write* a perspex to Φ, or to have control *jump* to Φ, or to have control jump by a relative step of Φ halts the machine. Thus the perspex at nullity is a read-only variable that can never be corrupted by a perspex program. The zero perspex specifies a read from, write to, and a relative jump by Φ, so it operates as a *universal halting instruction*, H, that halts the machine at any location in program space. There are also infinitely many halting instructions that halt the machine by instructing it to jump from each location in program space to Φ. By default all locations in program space contain H. However, in all non-trivial cases the program space is *instantiated* with some non-halting perspexes. The machine starts by executing the perspex at the Euclidean origin and halts when it encounters H or any other halting instruction. This halting occurs part way through the instruction, so a null machine halts without completing any instructions. The null machine is, therefore, different from the identity machine which repeatedly executes an identity instruction.

* J.A.D.W.Anderson@reading.ac.uk, http://www.cs.reading.ac.uk
Department of Computer Science, The University of Reading, Reading, Berkshire, England, RG6 6AY.

Practical implementations of the perspex machine use rational co-ordinates and implement the program space as a hash table. As usual, the default value, here H, is stored only once by the hashing algorithm and is returned when an uninstantiated cell of the hash table is read. Writing H into an instantiated cell has the side effect of freeing the memory associated with the perspex stored there. Thus all computations can be performed exactly and in finite time, because rational co-ordinates are used for everything. In addition, the computer's memory can be used efficiently by implementing a memory management policy that writes H into redundant locations.

In the early sections we define the perspex machine and review the properties of an *Unlimited Register Machine* (*URM*)[4] which is known to be equivalent to a Turing machine. In later sections we show that the rational perspex machine can simulate a URM, so it can simulate a Turing machine. The operations of rational matrix algebra are known to be computable[4], hence the rational perspex machine is equivalent to a Turing machine. However, a real perspex machine exceeds the computational abilities of a Turing machine. If it is physically possible to construct a real perspex machine it will falsify the Church-Turing thesis and will be a more powerful computer than is currently, theoretically possible.

It is also shown that the perspex machine is reversible in the Euclidean dimensions of space, but irreversible across natural numbered steps in the augmenting dimension of perspective space. Thus the perspex machine provides a geometrical model of spacetime in which instructions in space are reversible, but instructions in time are irreversible. We identify one physical phenomenon, randomness, which provides the required general, natural numbered exclusion of time reversal.

However, we also hypothesise that the sum over histories of quantum physics can be obtained by sampling fluctuations in the direction of time flow. On this assumption, we propose an experiment involving a modification of the classical Casimir apparatus[3] that should measure fluctuations in time flow with respect to time duration - if any such fluctuations exist.

2. Perspex Machine

Projective geometry is usually carried out in a real or complex space[8]. If a theoretical perspex machine were defined to operate in such a space it would be able to do more than a Turing machine. For example, it would be able to decide if any given irrational number is equal to zero. Whereas a Turing machine faced with the decimal expansion of a small number with indefinitely many leading zeros after the decimal point and before the significant figures is not be able to decide this case[10,11]. It is interesting to note that such a simple condition as implementing a perspex machine in a real space would falsify the Church-Turing thesis. However, all contemporary, practical perspex machines are defined in a rational space.

A perspex machine can be defined in a space of any natural numbered dimension, but we are particularly interested in applying the perspex machine to practical robot vision, so we define the machine in a rational, 4D, homogeneous space, corresponding to a 3D Euclidean world. Thus the program space P is defined as:

$$P = Q \times Q \times Q \times Q \qquad \text{(Eqn 1)}$$

Classical projective geometry[6] adopts the axiom $kA \equiv A$, for $k \neq 0$ and k a number of the same type as is used to define the perspective space, and with A a transformation or vector. This constraint is too strong to support a practical vision machine in a real numbered space, because, in particular, $-A \equiv A$ and such a machine cannot distinguish between left and right, forward and backward, up and down, or inside and outside. A practical vision machine requires an orientable projective geometry, such as arises naturally in a complex space, or by construction in a projective geometry such as that popularised by Stolfi[8].

We define a weaker equivalence in program space. We will see later that the perspex machine can compute anything that a Turing machine can, so it can compute the equivalences required by classical projective geometry[6].

We begin by defining that A is a 4×4 rational matrix, called a *perspex transformation*. We then define the following equivalence. Hence the left hand side, kA, defines the canonical form of A. This form also defines an unorientable projective geometry, but it leads to a convenient method for distinguishing position and direction vectors. Following one popular interpretation in computer graphics[5] a vector t with $t_4 = 1$ is considered to be a position vector in 3D Euclidean space and a vector with $t_4 = 0$ is considered to be a direction vector in 3D Euclidean space, analogous to a point at infinity. Thus the 3D hyperplane $t_4 = 1$ corresponds to all of the positions in 3D Euclidean space and the 3D hyperplane $t_4 = 0$ corresponds to all of the directions in 3D Euclidean space. The unoritentable geometry defined by the equivalence below can be made orientable by the computer graphics technique of clipping[5] which relies on the affine fixing of perspective space to the Euclidean origin as adopted here.

$$kA \equiv A \text{ with } k = \begin{cases} 1/a_{44}, a_{44} \neq 0 \\ 1, \text{otherwise} \end{cases} \qquad \text{(Eqn 2)}$$

Unless otherwise noted, all transformations are premultiplying, active transformations of co-ordinates. It is then convenient to call successive column vectors of A the x-, y-, z-, and t-vectors. Thus:

$$A = \begin{bmatrix} x_1 & y_1 & z_1 & t_1 \\ x_2 & y_2 & z_2 & t_2 \\ x_3 & y_3 & z_3 & t_3 \\ x_4 & y_4 & z_4 & t_4 \end{bmatrix} \qquad \text{(Eqn 3)}$$

An arrow written as a diacritical mark above a symbol denotes indirection. Thus x is the vector $\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T$, but $\vec{x}$ is whatever is stored at the location $\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T$ in program space. By default, every location of program space holds the zero perspex H, which is a universal halting instruction in the perspex machine. However a location of program space can be instantiated with any non-zero perspex and can be over written with any perspex. Thus any location of program space can hold any perspex.

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{(Eqn 4)}$$

An arrow written between symbols denotes writing, so $\vec{x} \to \vec{y}$, means that the contents of the location $\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T$ are written into the contents of the location $\begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix}^T$.

The product always implies reduction to canonical form by (Eqn 2). Thus $\vec{x}\vec{y}$ denotes the matrix product of the perspexes stored at $\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T$ and $\begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix}^T$, followed by reduction to canonical form. This product provides exactly the transformations required by projective geometry[7]. In general, the canonical result is written to a location, $\vec{z}$, as denoted by $\vec{x}\vec{y} \to \vec{z}$, but if $\vec{z} = \Phi$ the machine halts without writing to $\vec{z}$.

The requirement that the product always denotes canonicalisation is a very strong constraint that has profound effects on the perspex machine. In the discussion of the identity perspex machine below, and in the section on irreversibility, we see that this constraint imposes a temporal ordering on the machine. This ordering can be exploited to modularise perspex programs and to reduce the risk of entering infinite loops.

The perspex machine has one, compound instruction:

$$\vec{x}\vec{y} \rightarrow \vec{z}\,;\ \text{jump}(\vec{z}_{11}, t) \qquad \text{(Eqn 5)}$$

This instruction is isomorphic to a perspex that has column vectors x, y, z, and t, so it is called a *perspex instruction*. Perspex transformations and instructions are interchangeable in the perspex machine, because all perspexes are valid transformations and instructions. We have just seen how the first part of this instruction works. The second part of the instruction examines the resultant element $\vec{z}_{11}$ and instructs control to jump from the current location in program space by a relative step j as controlled by the vector t. The machine halts if $j = \Phi$ or j instructs a jump to Φ from the current location.

$$j_1 = \begin{cases} t_1, \vec{z}_{11} < 0 \\ 0, \text{otherwise} \end{cases}$$

$$j_2 = \begin{cases} t_2, \vec{z}_{11} = 0 \\ 0, \text{otherwise} \end{cases}$$

$$j_3 = \begin{cases} t_3, \vec{z}_{11} > 0 \\ 0, \text{otherwise} \end{cases}$$

$$j_4 = t_4 \qquad \text{(Eqn 6)}$$

When simulating a Turing machine it is convenient to have the perspex machine start by attempting to execute the instruction at the Euclidean origin $\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T$. The machine halts when it encounters any halting instruction, such as the universal halting instruction H. However, the perspex machine is defined for use in a robot[2], so it is convenient to re-start the perspex machine after a halt, so that the robot can continue to respond to its environment. This does not cause any difficulty when simulating a Turing machine, because the simulation can arrange to set a flag to indicate that the simulation has halted, so that the simulation is not re-run when the machine re-starts. Thus a perspex program can do something with the results of a Turing execution of a computable algorithm. When simulating parallel machines it is convenient to allow several perspex machines to operate in the same program space, but to start them at general locations and times. Parallel perspex machines face the difficulty of *non-deterministic write contentions* when two or more perspexes are written simultaneously into the same location, but this can be handled deterministically by averaging the perspexes that are written into the location.

3. Null Perspex Machine

The null perspex machine has a halting instruction at the Euclidean origin. With the universal halting instruction H at the Euclidean origin the general perspex instruction $\vec{x}\vec{y} \rightarrow \vec{z}\,;\ \text{jump}(\vec{z}_{11}, t)$ becomes:

$$\overrightarrow{\left[0\ 0\ 0\ 0\right]}^{T}\overrightarrow{\left[0\ 0\ 0\ 0\right]}^{T} \to \overrightarrow{\left[0\ 0\ 0\ 0\right]}^{T}$$

$$\text{jump}\left(\overrightarrow{\left[0\ 0\ 0\ 0\right]}^{T}_{11}, \left[0\ 0\ 0\ 0\right]^{T}\right)$$

Thus the perspex machine reads the perspex at nullity twice and forms its canonical product. That is, it computes:

$$\overrightarrow{\left[0\ 0\ 0\ 0\right]}^{T}\overrightarrow{\left[0\ 0\ 0\ 0\right]}^{T} = HH = H$$

The machine is instructed to write the result into the location $\left[0\ 0\ 0\ 0\right]^{T} = \Phi$, but this is an illegal operation and halts the machine. Thus the machine halts mid-way through the instruction. That is, the null perspex machine halts without completing any instructions.

4. Identity Perspex Machine

There are many ways to construct an identity perspex machine that repeatedly executes an identity instruction. One way to do this is in terms of the identity perspex I.

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(Eqn 7)}$$

We instantiate the program space with I at each of the x, y, z, and t locations of I. That is:

$$I \to \overrightarrow{\left[1\ 0\ 0\ 0\right]}^{T}$$

$$I \to \overrightarrow{\left[0\ 1\ 0\ 0\right]}^{T}$$

$$I \to \overrightarrow{\left[0\ 0\ 1\ 0\right]}^{T}$$

$$I \to \overrightarrow{\left[0\ 0\ 0\ 1\right]}^{T}$$

If we want the machine to stop after one execution of an identity instruction we need not instantiate any more locations of program space. However, if we want a natural number n of executions of an identity instruction we make the, possibly further, instantiations:

$$I \to \overrightarrow{\left[0\ 0\ 0\ i\right]}^{T} \text{ for } i \text{ from 1 to } n$$

With the identity perspex I at the Euclidean origin the general perspex instruction $\vec{x}\vec{y} \to \vec{z}$; jump$(\vec{z}_{11}, t)$ becomes:

$$\overrightarrow{\left[1\ 0\ 0\ 0\right]}^{T}\overrightarrow{\left[0\ 1\ 0\ 0\right]}^{T} \to \overrightarrow{\left[0\ 0\ 1\ 0\right]}^{T}$$

$$\text{jump}\left(\overrightarrow{\left[0\ 0\ 1\ 0\right]}^{T}{}_{11}, \left[0\ 0\ 0\ 1\right]^{T}\right)$$

Thus the perspex machine reads the identity perspex twice and forms its canonical product. That is, it computes:

$$\overrightarrow{\left[1\ 0\ 0\ 0\right]}^{T}\overrightarrow{\left[0\ 1\ 0\ 0\right]}^{T} = II = I$$

It then performs the legal write $I \to \overrightarrow{\left[0\ 0\ 1\ 0\right]}^{T}$, but this does not change the contents of the location $\left[0\ 0\ 1\ 0\right]^{T}$ because it already contains I. Thus none of the contents of program space are changed.

The machine then examines the resultant element $\vec{z}_{11} = I_{11} = 1$ and constructs the relative jump $j = \left[0\ 0\ 0\ 1\right]$ as follows. Firstly, $j_1 = j_2 = 0$, because the corresponding otherwise-clauses of (Eqn 6) apply. Secondly, $j_3 = 0$, because $t_3 = 0$. Finally, $j_4 = 1$ because $t_4 = 1$.

The relative jump $j = \left[0\ 0\ 0\ 1\right]$ is legal, so control jumps from the Euclidean origin, $\left[0\ 0\ 0\ 1\right]$, to $\left[0\ 0\ 0\ 2\right]$. Thus the machine has completed one identity instruction. In the case that $\left[0\ 0\ 0\ 2\right]$ is uninstantiated the machine halts, but if it is instantiated with I then the identity instruction is performed a second time and the machine jumps to $\left[0\ 0\ 0\ 3\right]$. This continues until the jump is to a location containing a halting instruction, that is, until the machine jumps to an uninstantiated location.

Thus the identity perspex machine executes its first instruction stored in the 3D hyperplane $t_4 = 1$. It then progresses sequentially through the 3D hyperplanes $t_4 = 2, 3, 4\ldots$ until it halts. This is a general property of perspex machines and is a consequence of the canonicalisation, defined in (Eqn 2), of the product in (Eqn 5).

5. Irreversibility

The canonicalisation, defined in (Eqn 2), of the product in (Eqn 5), forces the element $t_4 = 0, 1$. The vector t is used to construct a relative jump, so control either stays in the current 3D hyperplane n when $t_4 = 0$, or else control jumps to the successor hyperplane $n+1$ when $t_4 = 1$. The perspex machine starts execution in the $t_4 = 1$ hyperplane, so it visits all natural numbered hyperplanes, in sequence, until it halts. The machine may execute an arbitrary number of instructions in each hyperplane, thus spending time in that hyperplane, but once it jumps to the next hyperplane it cannot jump back. Thus canonicalisation enforces an irreversible temporal order on the hyperplanes visited by a perspex machine. By contrast, if

the perspex machine were defined using affine, not perspective transformations, then the canonicalisation defined in (Eqn 2) would not be needed and the machine could have arbitrary values of t_4, thus permitting it to jump arbitrarily amongst hyperplanes. Thus, in general, perspective perspex machines are not time reversible, whereas affine perspex machines are time reversible.

The constraint of visiting hyperplanes in sequence is not too onerous. If it is desired to make control jump regardless of the sign of $\hat{z}_{11}$ then t_1, t_2, and t_3 are all set to some non-zero value and the program's instructions are triplicated in the x-, y-, and z-axes. This keeps a perspective perspex machine time reversible within a hyperplane and gives the machine a partial fault-tolerance - in so far as the corruption of the program on an axis will leave the remaining copies, if any, intact and able to operate on some data. Thus a perspex machine degrades gracefully when damaged.

However, time irreversibility is a useful phenomenon. When a sub-program is finished the perspex machine can jump to the next hyperplane, preventing control from re-entering the finished sub-program. This allows a degree of modularisation in perspex programs. Thus hyperplanes tend to have dedicated functions.

Time irreversibility can also be used to prevent a perspex machine entering an infinite loop. If, instead of triplicating code in the current hyperplane, the machine always jumps to a successor hyperplane, then it can still compute arbitrary functions up to some iteration limit, but will halt when it jumps to an un-instantiated hyperplane. Such a machine might copy programs into the successor hyperplane, thus instructing an infinite loop, but memory limitations in a practical machine will cause it to halt.

When a perspex machine is used to control a robot[2], the machine is re-started after a halt, so the machine can get back to some pre-computed stage of a computation, unless real-time input to the robot prevents it. Thus a perspex machine can use the re-start to control iterations and halt them after some iteration count and before a memory allocation fault causes a halt. A perspex machine can potentially recover from the allocation of all of its memory by writing H into redundant locations in program space, so it is not essential that the iteration count prevents a memory fault. Hence irreversible perspex machines can enter an infinite loop only if the loop causes a memory fault on every re-start.

Thus time irreversibility can be put to good use in a perspex machine to modularise programs and to almost always prevent infinite loops.

It is extremely interesting to note that time irreversibility is a consequence of projective geometry, but not of affine geometry. Though it is, of course, possible to retain (Eqn 2) in an affine geometry, in which case the perspex machine's instructions are reversible in the spatial dimensions x, y, and z, but are irreversible across natural numbered steps in the time dimension t. A natural numbered time exclusion principle is also available in physics. Quantum fluctuations are said to be random, but the outcome of a random process does not depend on the outcomes at earlier or later times, so a physically random process does not reverse when time reverses, instead the reversal of time re-presents an opportunity for a random process to produce an outcome, but the outcome it produces is chosen randomly. In the continuous, that is, real numbered, period between quantum measurement events the universe may be reversible, but once a discrete, that is, natural numbered, measurement event takes place the universe is generally irreversible. Thus the physical universe is generally irreversible and quantum entropy, if not macro entropy, continues to increase even if time flows backwards. But we now see that the reversal of time does not have the effects it is commonly said to have. In common parlance, the randomness of quantum physics excludes general time reversals.

The quantum entropy of the universe might be the vacuum energy. If so, the effect of time flow fluctuations in the partial absence of virtual particles should be measured by a variant of the Casimir apparatus[3], Figure 1.

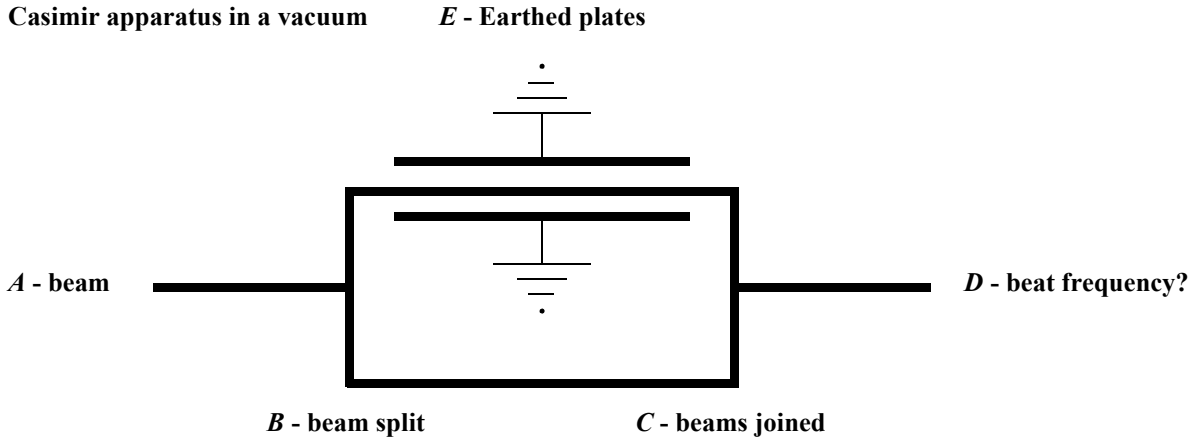

Fig.1: Modified Casimir apparatus to measure putative fluctuations in time flow with respect to time duration.

The Earthed plates at *E* in the Casimir apparatus[3] partially exclude the formation of virtual particles between the plates, creating a differential pressure with respect to the virtual particles in the vacuum of the apparatus. This pressure provides a small force driving the plates together. Now suppose that a coherent or entangled beam emanates from *A* and is split at *B*, such that one beam passes between the plates and the other passes outside the plates. The beams are joined at *C*. The beam passing between the plates is exposed to fewer random, quantum events (the creation and annihilation of virtual particles) than the beam passing outside the plates. So, if the direction of time flow fluctuates at a shorter wavelength than the path of light between the plates, the beam passing between the plates should be perturbed by this fluctuation and show a, possibly complex, beat pattern at *D*. Here fluctuations in time flow are measured in terms of the number of cycles of time reversal with respect to elapsed time in the vacuum.

It would be profoundly interesting to know if this latter day Michelson-Morley type experiment will reveal a positive or negative result. If the result is positive then one might broadcast signals forward and backward in time by the simple expedient of modulating the charge or separation of the plates. This would make it possible to construct a computer that takes no elapsed time to compute the domain and range of any function that fits inside the computer. If negative, it rules out some models of the internal structure of time.

If the result is positive, then it provides a simple explanation of the sum over histories in quantum physics. Thus, a particle moves in a straight line, but is perturbed by Brownian motion in continuous spacetime. When a natural numbered measurement process takes place the particle is found to be on the shortest path, or at a distance from it governed by the statistics of the Brownian process. Such a Brownian process also explains electron tunnelling and the uncertainty principle. Surely this provides reason enough to consider conducting the experiment?

6. Unlimited Register Machine

The *Unlimited Register Machine (URM)*[4] has an indefinitely long strip of registers indexed by natural numbers that each hold a whole number. The registers are analogous to the data tape of a Turing machine[4,10,11]. A URM program is composed of a finite list of instructions. The URM starts a program by executing the first instruction in the list and stops when it is instructed to advance control beyond the end of the list. The URM has four instructions called *Z*, *S*, *T*, and *J* in[4], but which are renamed here, respectively, as: the instruction $G(n)$ which *grounds* register n by setting it to zero; the instruction $C(n)$ which *counts* by incrementing the register n by one; the instruction $T(m,n)$ which *transcribes* a value by writing the contents of register m into register n; and the operation $A(m,n,p)$ which compares m and n then jumps to instruction p

if $m = n$, but otherwise *advances* to the next instruction in the list. In its operation the URM executes successive instructions in the list and can jump to a non-adjacent instruction only if the instruction A is executed. When the URM halts the result of the program is left somewhere in the strip of registers.

6.1 Perspex URM

We now show that a perspex machine can perform all of the operations of a URM, but first we show how a URM program and registers can be represented and how they can be kept distinct. We begin with the identity perspex machine that executes one identity instruction, thereby passing control to the origin of the $t_4 = 2$ hyperplane. This leaves the $t_4 = 1$ hyperplane unmodified and we identify the first n natural numbered locations $\begin{bmatrix} i & i & i & 1 \end{bmatrix}^T$, for i from 1 to n, with the registers of the URM. Thus the registers are represented and are distinct from the boot perspex program, I, in the $t_4 = 1$ hyperplane and from the URM program in the $t_4 = 2$ hyperplane. The URM program is a collection of perspexes that perform the URM operations, explained below, and is laid out arbitrarily, apart from the first URM instruction which must be, or begin, at the location $\begin{bmatrix} 0 & 0 & 0 & 2 \end{bmatrix}$. All locations in the $t_4 = 2$ hyperplane that are not instantiated with URM instructions contain H, so advancing control outside the URM program causes the perspex machine to halt, analogous to a URM halt. The explanation of how a perspex program computes the four URM instructions so as to effect control, and affect the element x_{11} of a perspex exactly like the contents of a URM register, now completes the proof that a perspex machine can compute anything that a URM can. The URM is known to be equivalent to a Turing machine[4], so the perspex machine can compute anything that a Turing machine can.

6.2 Perspex G

$$G = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Writing G into the location z zeros the element z_{11} as required by the URM instruction $G(z)$. Hence it is sufficient for the perspex to be of the form $GI \to \overset{\flat}{z}$; jump$(\overset{\flat}{z}_{11}, t)$.

6.3 Perspex C

$$C = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The URM instruction $C(z)$ increments the contents of register z. The perspex transformation C has the same effect, as can be proved by induction on the following series.

$$Z^{(0)} = G = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \Rightarrow {z^{(0)}}_{11} = 0$$

$$Z^{(1)} = CG = \begin{bmatrix} 1&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix} \begin{bmatrix} 0&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \\ 1&0&0&0 \end{bmatrix} = \begin{bmatrix} 1&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \\ 1&0&0&0 \end{bmatrix} \Rightarrow z^{(1)}{}_{11} = 1$$

$$Z^{(2)} = CCG = \begin{bmatrix} 1&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix} \begin{bmatrix} 1&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \\ 1&0&0&0 \end{bmatrix} = \begin{bmatrix} 2&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \\ 1&0&0&0 \end{bmatrix} \Rightarrow z^{(2)}{}_{11} = 2$$

6.4 Perspex T

The URM instruction $T(m, n)$ writes the contents of register m into register n. A sufficient perspex is of the form $\vec{m}I \to \vec{n}$; $\text{jump}(\vec{n}_{11}, t)$.

6.5 Perspex A

The URM instruction $A(m, n, p)$ compares registers m and n then jumps to instruction p if $m = n$, but otherwise *advances* to the next instruction q in the list. It is sufficient for a perspex to jump depending on the value of $m_{11} - n_{11}$. This can be done by using a subtraction operation S to compute an intermediate perspex R with the element $r_{11} = m_{11} - n_{11}$.

$$S = \begin{bmatrix} 0&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \\ -1&0&0&0 \end{bmatrix}$$

$$R = \begin{bmatrix} m_{11}&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \\ 1&0&0&0 \end{bmatrix} \begin{bmatrix} 0&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \\ -1&0&0&0 \end{bmatrix} \begin{bmatrix} n_{11}&0&0&1 \\ 0&1&0&0 \\ 0&0&1&0 \\ 1&0&0&0 \end{bmatrix} = \begin{bmatrix} m_{11}-n_{11}&0&0&-1 \\ 0&1&0&0 \\ 0&0&1&0 \\ 1&0&0&0 \end{bmatrix}$$

Hence a sufficient perspex program computes R and then performs the jump $RI \to \vec{z}$; $\text{jump}\left(\vec{z}_{11}, \begin{bmatrix} q' & p' & q' & 0 \end{bmatrix}^T\right)$. Here p' and q' are, respectively, the relative jumps from the current location to p and q.

7. Perspex Computability

We show above that the perspex machine can compute anything that a Turing machine can compute, but can a perspex machine compute more than a Turing machine? If the perspex machine operates on rational numbers then all of the matrix multiplications are rational and these operations are known to be Turing computable[4]. The perspex jump is Turing computable, because it can be implemented as change of state in a Turing machine. So the rational perspex machine is equivalent to a Turing machine. However, if the perspex machine operates on real numbers, including irrational ones, then it can decide the equality of irrational numbers, which is known not to be Turing computable[10,11]. Hence a real perspex machine would exceed the computational power of any, currently, theoretically possible computer.

It is conceivable that a real perspex machine could be built using pin-holes to provide the perspective transformation in the perspex instruction and stops, mirrors, or wave guides, to provide the jump. Using ultraviolet light it might be feasible to have of the order of 10^7 computing elements per metre. Light travels at a speed of order $10^8\ ms^{-1}$, so an ultraviolet perspex machine might operate at a speed up to $10^7 \times 10^8 = 10^{15}$ operations per second. This is in the petahertz range and is at least one order of magnitude faster than the current theoretical limit for the speed of a silicon, semiconducting computer.

An optical perspex machine need not be very large. The optical element could be a block of material with mirrors on either side. The computation could then be arranged like the generation of a beam in a laser, with the computation passing between the mirrors until the result is computed. This would make a very inefficient laser, but an extremely fast computer. Higher speeds might be achieved in a medium using shorter wavelength parts of the electromagnetic spectrum. But the highest possible speed might be achieved following a positive result in the experiment proposed above.

8. Dualities and Substitutions

The perspex describes the vertices of a, possibly degenerate, tetrahedron, so any duality, or any operation whatsoever, that generates vertices generates a valid perspex. Thus the dualities of projective geometry are trivially valid in program space.

A non-trivial substitution is to record all of the locations jumped to by a perspex program and group them, in order, in fours, with possible repetitions, so that the groups form perspexes. Placing the perspex thus formed at the location first visited by the group over writes some of the locations in program space, but produces a valid perspex program, because there is a perspex at the origin and all of the vectors in all of the perspexes reference locations that contain either perspexes present in the previous generation of the program or newly constructed ones. Thus this substitution supports deterministic genetic algorithms which do not require random or pseudo random numbers or arithmetical error or any form of error at all to produce mutants. That is, this perspex substitution provides a deterministic form of computational evolution.

9. Conclusion

We describe an original machine, the perspex machine, that unifies the projective geometry of tetrahedral figures and Turing computation. Hence all possible Turing computations can be performed geometrically. Specifically, all possible computations in projective geometry can be performed using Tarski's algorithm[9]. In theory the perspex machine is a supra-Turing machine in that it can perform exact, irrational computations. It might be possible to construct a practical perspex machine as an optical machine, by exploiting the perspective transformations inherent in the perspex machine. If so, an ultraviolet perspex machine might operate up to the petahertz range of operations per second - at least one order of magnitude faster than is currently, theoretically possible for a silicon, semiconducting computer.

The dualities of projective geometry hold trivially in perspex program space. A non-trivial substitution of the locations jumped to by a perspex program produces a deterministic form of computational evolution that does not use any source of randomness or error to produce mutants.

The perspex machine has the surprising property that instructions are reversible in the Euclidean dimensions, but are irreversible in the augmenting, perspective dimension. Thus perspex program space provides a spacetime geometry in which instructions in space are reversible, but instructions in time are not. This insight allows us to identify one physical phenomenon, randomness, which excludes general time travel. However, it is conceivable that fluctuations in the direction of time flow are a natural, physical phenomenon that is masked by random processes. If so, the speed of time flow fluctuation with respect to time duration in the vacuum might be measured by a proposed Casimir apparatus. It would be profoundly interesting to know if this experiment would produce a positive or negative result. If positive, then it should be possible to broadcast signals forward and backward in time and, thereby, arrange that computers can compute any function that fits inside the machine in no elapsed time in the vacuum. A negative result would rule out some conceivable models of the internal structure of time.

References

1 Anderson, J.A.D.W. "Representing Geometrical Knowledge" *Phil. Trans. Roy. Soc. Lond.* series B, vol. 352, no. 1358, pp. 1129-1139, Aug. 1997.

2 Anderson, J.A.D.W. "Robot Free Will" to appear in van Harmelen, F. (ed) *ECAI 2002. Proceedings of the 15th European Conference on Artificial Intelligence* IOS Press, Amsterdam, 2002.

3 Casimir, H.B.G. *Proc. Kon. Ned. Akad. Wetensch.* B51, 793, 1948.

4 Cutland, N.J. *Computability* Cambridge University Press, 1980.

5 Foley, J.D., van Dam, A., Feiner, S.K. & Hughes, J.F. *Computer Graphics: Principles and Practice* 2nd edn., Addison-Wesley, 1990.

6 Greenberg, M.J. *Euclidean and Non-Euclidean Geometries*, 3rd edition, W.H. Freeman, 1993.

7 Riesenfeld, R.F. "Homogeneous Coordinates and Projective Planes in Computer Graphics" *IEEE CG&A* pp. 50-55, 1981.

8 Stolfi, J. *Oriented Projective Geometry* Academic Press, 1991.

9 Tarski, A. *A Decision Method for Elementary Algebra and Geometry* University of California Press, 1951.

10 Turing, A.M. "On Computable Numbers, with an Application to the Entscheidungs Problem" *Proc. Lond. Math. Soc.* vol. 2, no. 42, pp. 230-265, 1937.

11 Turing, A.M. "On Computable Numbers, with an Application to the Entscheidungs Problem. A correction." *Proc. Lond. Math. Soc.* vol. 2, no. 43, pp. 544-546, 1937.