PERSPEX MACHINE III: CONTINUITY OVER THE TURING OPERATIONS

COPYRIGHT

Perspex Machine III: Continuity Over the Turing Operations

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

Abstract

The perspex machine is a continuous machine that performs perspective transformations. It is a super-Turing machine that contains the Turing machine at discrete locations in perspex space. We show that perspex spaces can be constructed so that all of the operations in a Turing program lie in a continuum of similar operations in the space, except for the Turing *halt* which is always a discontinuous operation. We then show how to convolve a Turing program to produce an isolinear program that it is robust to missing instructions and degrades gracefully when started incorrectly, sometimes even recovering in performance. We hypothesise that animal brains are similarly robust and graceful because animal neurons share the geometrical properties of the perspex machine. Furthermore, convolution of Turing programs makes possible the band-pass filtering and reconstruction of programs. Global processing can then be obtained by executing the broad bands before the finer ones. Hence, any existing computer program can be compiled on a perspex machine to make it global in operation, robust to damage, and degrade gracefully in the presence of error. The three "Holy Grails" of AI – globality, robustness, and graceful degradation – can be supplied by a compiler. They do not require specific programming in individual cases because they are geometrical properties of the perspex machine.

Keywords: isolinear program, perspex machine, Turing machine.

1. Introduction

The perspex machine performs perspective transformations and was introduced in[2] by unifying the Turing machine with projective geometry. This was done by showing how four, specific, perspective transformations can carry out the four operations of the Unlimited Register Machine (URM), and how these transformations can be laid out in space as the program and registers of the URM. The URM is equivalent to a Turing machine so the perspex machine can carry out all Turing operations. However, the perspex machine can be defined to operate in a continuous space, in which case it can perform operations on general real-numbers that are not accessible to a Turing machine. This establishes that the perspex machine is a super-Turing machine and opens up the possibility of convolving Turing programs and of filtering them.

Whilst[2] gave a clear description and examples of the perspex machine, the discussion of the halting condition was unsatisfactory. This was remedied in[3] where it was defined that the machine halts only when it executes the nullity perspex, H, with all elements $h_{ij} = \Phi$. The number nullity, $\Phi = 0/0$, is both a transrational[1] and a transreal[3] number, as is the number infinity, $\infty = 1/0$. The transreal numbers are the union of the strictly transreal numbers, $\{\Phi, \infty\}$, with the real numbers. Nullity lies off the real number line and infinity lies at its positive extreme. The strictly transreal numbers occur as co-ordinates in perspex space and, amongst other things, ensure that the Turing *halt* is a discontinuous operation, despite continuity over all other Turing operations. Various forms of the perspex are illustrated in[3], though we are concerned only with the matrix and computer instruction forms here.

The perspex, or perspective simplex, can be a 4×4 matrix with column vectors x, y, z, and t or it can be a computer instruction: $\vec{x}\vec{y} \to \vec{z}$; jump($\vec{z}_{11}, t$). The perspex machine operates in a 4D space, called "perspex" or "program" space[2], that contains perspexes at every point. The machine executes the perspex at a point as the instruction. This reads the perspexes at locations x and y, multiplies them together and writes the product, reduced to canonical form, into the location z. It then examines the top left element, $\vec{z}_{11}$, of the product and constructs a relative jump from the current location using the

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

components of t. If $\vec{z}_{11} < 0$ it jumps by t_1 along the x-axis, otherwise if $\vec{z}_{11} = 0$ it jumps by t_2 along the y-axis, otherwise if $\vec{z}_{11} > 0$ it jumps by t_3 along the z-axis. In every case it jumps by t_4 along the t axis. Thus, the machine starts at some point and control jumps from point to point until H is encountered.

In the next two sections we derive some elementary properties of the linear interpolation of perspexes and use these to define the interpolation of Turing programs. In the subsequent section, on filtering, we define the isolinear interpolation of Turing programs and give an example of convolving a Turing program that results in an isolinear program that is robust to missing instructions and degrades gracefully in the presence of erroneous starting conditions. In the subsequent section we discuss the properties of isolinear materials; the advantages of applying band-pass filtering to programs; the properties of perspex analogues of physical operations; and, finally, we give a geometrical explanation of why animal brains are global in operation, robust to damage, and degrade gracefully in the presence of error. All of these properties can be had by compiling any existing program on a perspex machine. The three "Holy Grails" of AI – globality, robustness, and graceful degradation – do not require specific programming in individual cases because they are geometrical properties of the perspex machine.

2. Interpolation of Perspexes

Lemma 1: The real linear blend of real perspexes stored at real locations is a real perspex stored at a real location.

Proof 1: Let there be two perspex matrices, A and B, stored, respectively, at locations L_A and L_B. Then, for a numerical parameter α, the linear blend of the locations is $L_P = (1-\alpha)L_A + \alpha L_B$ and the perspex, P, at L_P, is given by $P = (1-\alpha)A + \alpha B$. If L_A and L_B are real locations then, by the closure of real arithmetic, L_P is real for all real α. Similarly, if A and B are real matrices then, by the closure of real arithmetic, P is a real matrix for all real α.

Lemma 2: The real linear blend of locations and perspexes with some strictly transreal components has some strictly transreal components.

Proof 2: The definitions above hold. Without loss of generality consider the term $t = (1-\alpha)x_A + \alpha x_B$ where x_A and x_B are transreal numbers, being components of a location or perspex, and α is real.

Firstly, if either or both of x_A or x_B is Φ then t is the strictly transreal number Φ because:

$$t = (1-\alpha)\Phi + \alpha x_B = \Phi + \alpha x_B = \Phi \text{ and}$$

$$t = (1-\alpha)x_A + \alpha\Phi = (1-\alpha)x_A + \Phi = \Phi \text{ and}$$

$$t = (1-\alpha)\Phi + \alpha\Phi = \Phi + \Phi = \Phi.$$

Secondly, if either x_A or else x_B is ∞ then t is the strictly transreal number ∞ because:

$$t = (1-\alpha)\infty + \alpha x_B = \infty + \alpha x_B = \infty \text{ and}$$

$$t = (1-\alpha)x_A + \alpha\infty = (1-\alpha)x_A + \infty = \infty.$$

Thirdly, if x_A and x_B are both ∞ then t is the strictly transreal number Φ because:

$$t = (1-\alpha)\infty + \alpha\infty = \infty + \infty = \Phi.$$

Lemma 3: The strictly transreal blend of transreal perspexes at transreal locations is the nullity perspex, H, stored at the point at nullity, (Φ, Φ, Φ, Φ).

Proof 3: The definitions above hold. Let $\alpha = \Phi$ then the resultant perspex is stored at the point at nullity because:

$$L_P = (1-\alpha)L_A + \alpha L_B = (1-\alpha)\left[x_A\ y_A\ z_A\ t_A\right] + \alpha\left[x_B\ y_B\ z_B\ t_B\right] = (1-\Phi)\left[x_A\ y_A\ z_A\ t_A\right] + \Phi\left[x_B\ y_B\ z_B\ t_B\right] =$$
$$\Phi\left[x_A\ y_A\ z_A\ t_A\right] + \Phi\left[x_B\ y_B\ z_B\ t_B\right] = \left[\Phi\ \Phi\ \Phi\ \Phi\right] + \left[\Phi\ \Phi\ \Phi\ \Phi\right] = \left[\Phi\ \Phi\ \Phi\ \Phi\right].$$

Alternatively, let $\alpha = \infty$. When the locations are real the resultant perspex is stored at the point at nullity because:

$$L_P = (1-\infty)\left[x_A\ y_A\ z_A\ t_A\right] + \infty\left[x_B\ y_B\ z_B\ t_B\right] = \infty\left[x_A\ y_A\ z_A\ t_A\right] + \infty\left[x_B\ y_B\ z_B\ t_B\right] = \left[\infty\ \infty\ \infty\ \infty\right] + \left[\infty\ \infty\ \infty\ \infty\right] = \left[\Phi\ \Phi\ \Phi\ \Phi\right].$$

The cases where the locations have components Φ and/or ∞ are obtained similarly. (Compare with Lemma 2.)

The results just obtained for the vectors L_A and L_B generalise to the matrices A and B giving the resultant matrix H as required.

Lemma 4: The linear blend of any perspex P with H is H.

Proof 4: The proof may be obtained by a trivial application of the first branch of Lemma 2.

3. Interpolation of Turing Programs

In[2] it is shown that Turing programs can be implemented as real, specifically integral, perspexes with a Turing computable boot program at $(0, 0, 0, 1)$ leading into an arbitrary Turing program that begins at $(0, 0, 0, 2)$. The combined program operates on consecutive locations $(0, 0, 0, n)$ with $n = 1, 2, 3, \ldots, k$ until it halts or jumps to some non-consecutive location on the integral lattice with nodes (a_1, a_2, a_3, a_4) at integral a_i. It continues processing similarly from this location and either cycles indefinitely or else halts. Hence, we may consider a Turing program as being made up of perspexes stored at consecutive integral locations along infinite lines or finite line segments. It is then trivial to compute the interpolated perspex P_t at every distance t along a real numbered line segment with $1 \le t \le k$, for some k, combining line segments, if necessary, to produce an infinitely long program. Firstly, off the line we have $P_\Phi = H$. Secondly, at the end points of the line segment we have the given P_1 and P_k. Thirdly, in the interior of the line segment we have $P_t = (1-(t-\lfloor t \rfloor))P_{\lfloor t \rfloor} + (t-\lfloor t \rfloor)P_{\lfloor t+1 \rfloor}$ where $\lfloor t \rfloor$ is the integer floor of t. Fourthly, to the left of the line segment, with $t < 1$, we have $P_t = H$ because $P_t = (1-(t-\lfloor t \rfloor))P_{\lfloor t \rfloor} + (t-\lfloor t \rfloor)P_{\lfloor t+1 \rfloor} = (1-(t-\lfloor t \rfloor))H + (t-\lfloor t \rfloor)P_{\lfloor t+1 \rfloor} = H$ by Lemma 4. Similarly, $P_t = (1-(t-\lfloor t \rfloor))P_{\lfloor t \rfloor} + (t-\lfloor t \rfloor)P_{\lfloor t+1 \rfloor} = (1-(t-\lfloor t \rfloor))P_{\lfloor t \rfloor} + (t-\lfloor t \rfloor)H = H$ to the right of the line segment with $t > k$. Thus, by construction, the whole of the real numbered line segment contains perspexes where the perspex P_t is asymptotically close to equality with $P_{t+\varepsilon}$ for small ε such that $1 \le t, t+\varepsilon \le k$. This is what is meant by "continuity over the Turing operations." In other words, the perspexes that make up the operations of a Turing program can be laid out so that they lie at consecutive integral distances along line segments, but with the whole of a line segment filled with perspexes such that a perspex is asymptotically similar to the perspexes in a small neighbourhood about it on the line segment. This result generalises, by interpolation along every axis, to give continuity throughout a region of perspex space.

With perspexes interpolated linearly, as here, perspexes at nearby locations perform a nearly equal computation as can be seen by considering the following proof schema and filling out the detail of the error matrices $E^{(i)}$.

Let A be the perspex matrix at location L_t on a line segment then, with our usual notation, $A \equiv \vec{x}\vec{y} \to \vec{z};\ \text{jump}(\vec{z}_{11}, t)$ at L_t. Consider the perspex matrix B at location $L_{t+\varepsilon}$ on the same line segment. Then, for some error matrices $E^{(i)}$:

$$B = A + E^{(1)} \equiv \left(\overrightarrow{x + E_x^{(1)}}\right)\left(\overrightarrow{y + E_y^{(1)}}\right) \to \left(\overrightarrow{z + E_z^{(1)}}\right);\ \text{jump}\left(\left(\overrightarrow{z + E_z^{(1)}}\right)_{11}, t + E_t^{(1)}\right) \equiv$$
$$(X + E^{(2)})(Y + E^{(3)}) \to (Z + E^{(4)});\ \text{jump}(Z_{11} + E_{11}^{(4)}, t + E_t^{(1)}) \equiv$$
$$(XY + XE^{(3)} + YE^{(2)} + E^{(2)}E^{(3)}) \to (Z + E^{(4)});\ \text{jump}(Z_{11} + E_{11}^{(4)}, t + E_t^{(1)}).$$

Then, as ε tends to zero, all of the $E^{(i)}$ tend to the zero matrix giving, in the limit:

$$XY \to Z;\ \mathrm{jump}(Z_{11}, t) \equiv \vec{x}\vec{y} \to \vec{z};\ \mathrm{jump}(\vec{z}_{11}, t) \equiv A \quad .$$

This completes the proof that asymptotically close perspexes on a line segment perform an asymptotically similar computation.

Turing computable programs are of finite length so the region of perspex space containing a Turing computable program, laid out as above, is of finite size and is surrounded everywhere by H. Such a region of space contains real perspexes everywhere because the Turing perspexes are real[2] and the interpolants are real by Lemma 1. If execution is started at any point in the region it either remains in the region or else halts when it leaves the region. Turing computable programs all have the latter property that they start within the region and halt on leaving it.

Let us call any region of perspex space that is entirely filled with non-halting perspexes, but is surrounded by H, a "perspex material." Then the distribution of given perspexes within the perspex material that cause it to be filled by interpolation is analogous to the distribution of atoms in a common physical material. Both materials have bulk properties. In the perspex material the size and distribution of the given perspexes may vary, but the whole of the region has computational properties. In the physical material the size and distribution of atoms may vary, but the whole of the region has bulk properties such as thermal, sonic, and electrical conductivity. If we subject perspexes to the motions that atoms, or other particles, undergo then we immediately have a computational analogue of all physical processes, such as annealing and diffusion. Computational analogues of (simulated) annealing and (Brownian) diffusion appear in the AI and general optimisation literature, but the perspex machine now makes computational analogues of all physical things, or at least all atomic things, available. This supports the perspex thesis[3] to the extent that it shows that the perspex machine can model physical things and physical things can instantiate the perspex machine. This analogy might be made more realistic by defining that the perspex reads and writes to relative, not absolute, positions, but it is too early to assess the merits of relative and absolute addressing in the perspex machine and to give a definitive account of non-local effects in physics.

4. An Example of Filtering

A perspex space may contain arbitrary super-Turing perspexes and may be subject to arbitrary super-Turing filtering but, for practical purposes, we are interested in filtering Turing programs using filters that can be computed by a Turing machine to asymptotic or exact accuracy. One of the simpler such filters is obtained by convolution with a triangular kernel. This can be implemented efficiently as a linear blend. The section above describes how to form the linear blend of a Turing program. This is equivalent to a triangular convolution of the locations read from and jumped to by a Turing program, but it does not convolve the locations written to. This must be handled explicitly by forming the blend when a perspex is written to a location at a distance t_c along the time line $(0, 0, 0, t)$. Here t_c is the centre of the kernel. Taking t_0 as the integer floor of t_c and t_1 as the integer ceiling, it is sufficient to form the blend in the region $t_0 \le t \le t_1$. Thus, $P_t = (1 - t_c + t)P_c + (t_c - t)P_0$ for $t_0 \le t \le t_c$, and $P_t = (1 + t_c - t)P_c + (t - t_c)P_1$ for $t_c \le t \le t_1$.

The linear blend between consecutive integers has some useful properties. Firstly, it is local, so perspexes with strictly transreal components elsewhere in space do not force the components of a convolved Turing program to be strictly transreal. Secondly, the real blend can be efficiently approximated by a digital computer at discrete distances on a segment of the time line of unit length. Thirdly, duplicating the first and last perspexes at adjacent unit distances, respectively, before and after the program, produces, at little cost, a buffer between the end of the program and the surrounding halting instructions. Hence, the execution of an approximation that strays just beyond the ends of the given program executes the first or last instruction exactly and does not halt until execution jumps unequivocally, i.e. more than unit distance, beyond the end of the given program. If it is necessary to prevent a duplicate execution of the last instruction then writing H into the final integral location terminates the convolved program on a jump of exactly unit or more beyond the end of the given program. Let us call the buffer between parts of a program, created by duplicating a perspex, a "telomere." This name is suggested by analogy with chromosomal telomeres. Then, fourthly, placing a telomere at either end of consecutive reads and writes, respectively, from or into, one hyperplane forces all approximations of each of these reads and writes to read and write exactly from or into that hyperplane. In other words, telomeres guarantee that all input and output will take place in fixed hyperplanes even when an approximation of the program is executed.

The perspex program given in[1] is a fibre of perspex neurons that grows into a program that rotates a cube. This program is used in association with a boot program that enters the fibre at $(0, 0, 0, 12)$. We convolved the program as follows. Firstly, we initialised the time line with the perspex:

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

This fills a protofibre with a perspex that has an identity effect in the perspex space used. The perspex performs an identity write and causes control to jump to the next location. Its effect is to cause control to silently pass over missing Turing instructions in a fibre laid on top of the protofibre. In the absence of a protofibre the convolution given here would not be robust to missing Turing instructions.

Secondly, we inserted telomeres at the ends of the given program and convolved it as above. Our simulation of the perspex machine was modified to perform convolutions on writes, as just described. The convolved program was then started at various locations. The output of the program is shown in Figures 1 - 22.

Figure 1 shows the rotated cube as in[3]. This figure was obtained by entering the convolved fibre at the entry point $(0, 0, 0, 12)$ with a zero temporal offset dt of zero. Four perspexes were written into the hyperplane $t = 1$ which is used for output by the given program. The execution and output of the convolved program is identical to the execution and output obtained without convolution in[3]. However, the lighting has been adjusted by hand. All of the subsequent figures have been edited by hand to have the same viewpoint and lighting as Figure 1.

Figure 2 is the result of entering the program at a distance of 0.001 before the beginning of the program. Thus, control enters the fibre in the telomere which executes the first given instruction exactly, but continues executing instructions out of phase by a displacement of -0.001. The figure has lost one perspex, leaving just three in the output. Figures 3 and 4 show larger temporal offsets before the beginning of the program. As the offset increases the perspexes depart from the shape, size, orientation, and position of the exact solution in Figure 1. In Figure 5 the offset of -0.2 happened to produce a perspex that instructed a zero jump in control. In other words, the program became non-halting. However, the program had already grown part of the fibre that would have output perspexes were it to be called independently before it started to cycle. If the cycle were to be halted and re-started at a small displacement the convolved program would have continued in a way similar to the preceding Figure 4 or the subsequent Figure 6. In Figure 6 the program recovers to the extent that it outputs four perspexes.

The situation is rather better when the program is entered slightly after its entry point. Figure 7 shows the exact solution as in Figure 1. This time positive offsets are used. Figure 8 appears, to the naked eye, to be identical to the exact solution, but, in fact, there is a small displacement of the perspexes. The displacement increases in figures 9 to 12. A small “crack” is visible in 9 which widens in the subsequent figures. Figure 10 has lost its rotation, and figures 11 and 12 are markedly disorganised. Figures 13 to 15 each have four perspexes, but, as the end of the program approaches, Figure 16 has two perspexes which declines to one perspex in Figure 17. However, the program again recovers and Figure 18 has two perspexes. In Figure 19 there are no perspexes, but the program again recovers to give one perspex in Figure 20. Figures 21 and 22 again have no output, but they are in the convolution margin beyond the end of the given program.

It is important to appreciate how aggressive this test is. A perspex program was convolved and was started at erroneous positions near and far from the true entry point. This program executed in every case. In one case, in Figure 5, it became non-halting, but in all other cases control passed from the convolved program to the grown program. Except at the end of the program – in figures 19, 21, and 22 – the grown program produced an output that lies, at least partially, in the field of view of the exact solution. On several occasions the program recovered to give more outputs than it had done at a lesser offset to the true entry point. This is consistent with the Walnut Cake theorem[3]. Strikingly, all of the outputs have some perspexes that intersect the exact solution. This is an astonishing degree of robustness. By contrast, if a conventional program were started at the wrong location it would be liable to fail catastrophically on a divide by zero error or on accessing an uninstantiated memory location. Nor could a conventional program be started between instructions so only the two figures, 1 and 7, of the 18 non-empty figures given here would be computable.

Figure 1: Entry at $dt = 0$, 4 perspexes output.

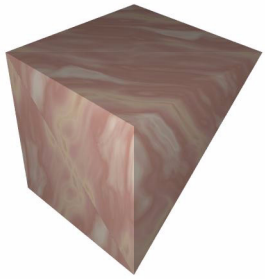

Figure 2: Entry at $dt = -0.001$, 3 perspexes output.

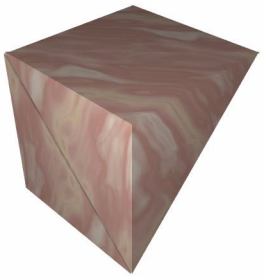

Figure 3: Entry at $dt = -0.01$, 3 perspexes output.

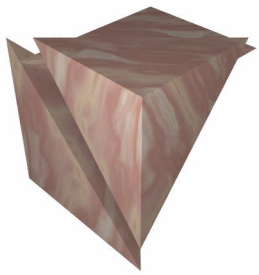

Figure 4: Entry at $dt = -0.1$, 3 perspexes output.

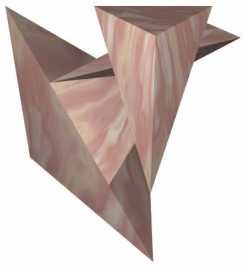

Figure 5: Entry at $dt = -0.2$, non-halting.

Figure 6: Entry at $dt = -0.3$, 4 perspexes output.

Figure 7: Entry at $dt = 0$, 4 perspexes output.

Figure 8: Entry at $dt = 0.001$, 4 perspexes output.

Figure 9: Entry at $dt = 0.01$, 4 perspexes output.

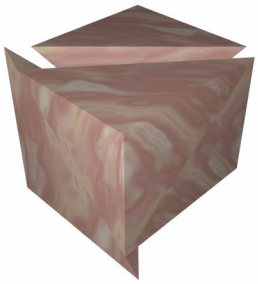

Figure 10: Entry at $dt = 0.1$, 4 perspexes output.

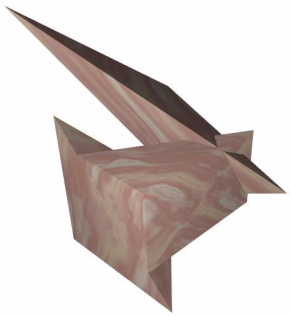

Figure 11: Entry at $dt = 0.2$, 4 perspexes output.

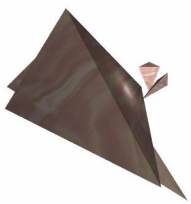

Figure 12: Entry at $dt = 0.3$, 4 perspexes output.

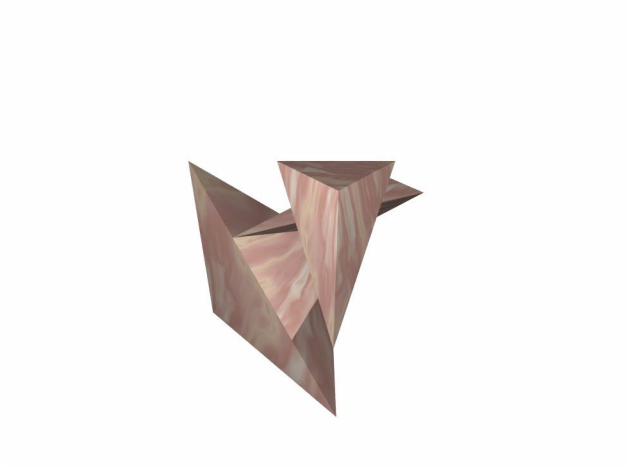

Figure 13: Entry at $dt = 1 - 0.3$, 4 perspexes output.

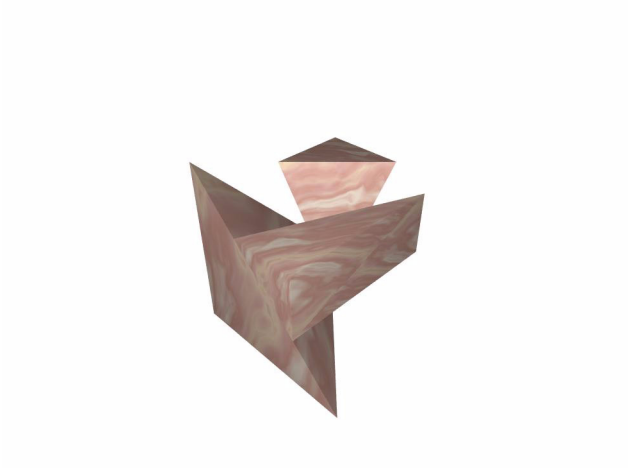

Figure 14: Entry at $dt = 1 + 0.3$, 4 perspexes output.

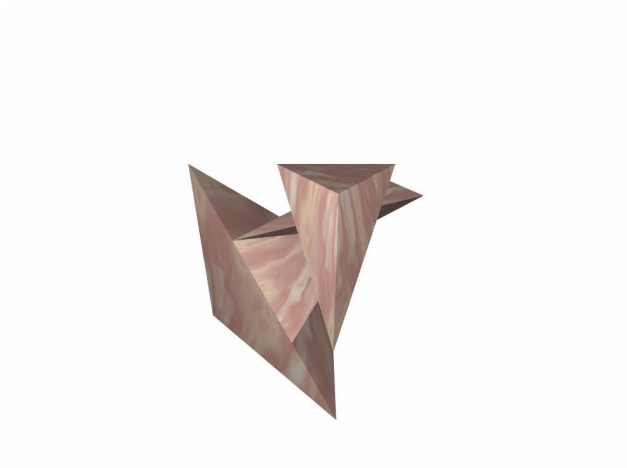

Figure 15: Entry at $dt = 2 - 0.3$, 4 perspexes output.

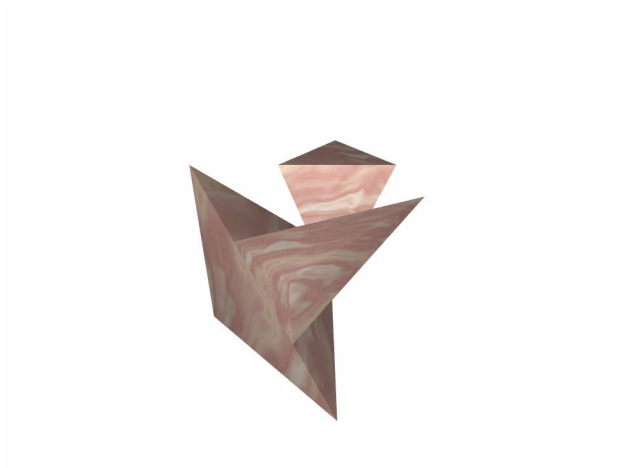

Figure 16: Entry at $dt = 2 + 0.3$, 3 perspexes output.

Figure 17: Entry at $dt = 3 - 0.3$, 1 perspex output.

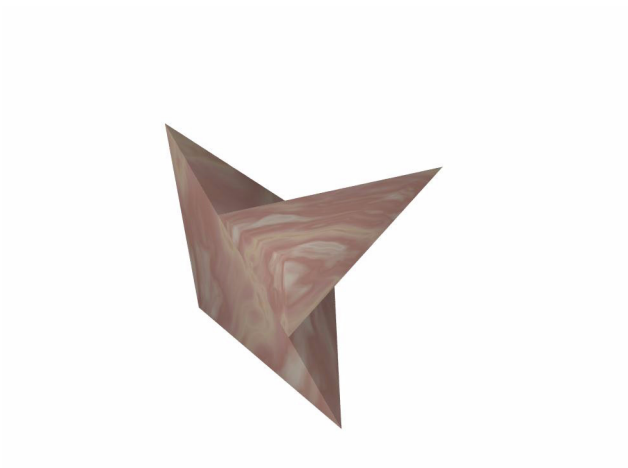

Figure 18: Entry at $dt = 3 + 0.3$, 2 perspexes output.

Figure 19: Entry at $dt = 4 - 0.3$, 0 perspexes output.

Figure 20: Entry at $dt = 4 + 0.3$, 1 perspex output.

Figure 21: Entry at $dt = 5 - 0.3$, 0 perspexes output.

Figure 22: Entry at $dt = 5 + 0.3$, 0 perspexes output.

Though it is only shown here in figures 8 and 9, starting close to any Turing instruction gives an output that is practically indistinguishable to starting at that instruction, as required by the continuity proof. Thus perspex programs are robust to small errors and, for many practical purposes, could be simulated in floating point arithmetic rather than the transrational arithmetic used here. This would lead to much faster program execution with a much reduced memory requirement.

The idea of convolving one Turing program can be extended to the simultaneous convolution of many Turing programs in one perspex material. Starting execution at any point in this material then executes a program that has the same component operations as the programs from which it is blended. In other words, the entire perspex material is a computational "isomer" of its components and is produced by a linear blend. Let us call such as perspex material an "isolinear" material[†]. Such materials are computationally robust to starting conditions, allow a contribution from all sub-programs, and provide an infinite variety of computational histories and outputs.

†. The word "isolinear" appears in the science fiction series Star Trek by Paramount Pictures. It describes a computational material that is physically robust and allows the simulation of real world events in the "holodeck" by combining the output of many subprograms to give a simulation with near infinite variety. Thus, science delivers one aspect of science fiction. But as happened in the case of space rockets and robots, the science of isolinear materials is more prosaic than the fiction.

5. Discussion

We begin the discussion by reviewing the properties of the perspex machine before considering its consequences for science, technology, and philosophy.

The perspex machine does not have any error states. It uses an exact, total arithmetic[1,3] so no arithmetical errors are possible. It also uses a total addressing scheme where any possible perspex accesses and jumps to points in perspex space that are always instantiated so no addressing errors are possible. The perspex machine does nothing other than accessing space, performing arithmetic, and jumping, none of which have any errors, so the machine is inherently robust. By contrast the Turing machine does have an error state. A Turing machine cannot progress beyond a non-deterministic selection of actions without the intervention of an external oracle.

The isolinear materials degrade gracefully in the presence of starting conditions containing a positional error with respect to the ideal starting point. Slight positioning errors do not cause catastrophic failure, but the degradation is non-monotonic. The Walnut Cake Theorem[3] shows that as a Turing machine attempts to improve its description of a continuous property, or a property at a finer resolution than its own symbols, it will go through paradigm shifts in which low resolution solutions are more accurate than high resolution ones. In the case of the isolinear material demonstrated here, in the section on filtering, low resolution solutions occur at values of dt approaching a phase of ± 0.5. We have seen that these phases can come closer to the exact solution of four perspexes than phases closer to 0. Thus, graceful degradation and recovery from error are geometrical properties of the isolinear perspex machine.

The protofibre provides a way of making an isolinear fibre robust to missing instructions. Protofibres operate by providing a skeleton of instructions that causes control to pass from one end of the protofibre to the other. If the isolinear fibre, grown on top of the protofibre, has a missing instruction then a convolution of the protoinstruction is executed instead and this passes control to any later instructions in the isolinear fibre. This certainly makes the isolinear fibre robust to missing instructions, but it requires prior knowledge of the length of the isolinear fibre, and its outgrowths, to be fully effective.

Isolinear programs are formed by convolution and describe a continuous fibre or a continuous volume of a higher dimensional space. This means that the programs themselves can be convolved and can be subject to band-pass filtering and reconstruction. The lowest possible band has just one perspex instruction in it that reads two convolved perspexes and writes one perspex which must be convolved when it is produced. The sole instruction then jumps to a halting instruction. Thus, an entire perspex program, of any finite length, can be approximated by a single instruction. As successively finer bands are executed the program successively approaches the exact solution. Whilst the norm of the error declines monotonically, as required by filtering theory, the actual error generally declines non-monotonically, as required by the Walnut Cake Theorem. In addition, the output values at the lower resolutions combine with any higher ones, each of which is convolved at the time of writing, so the absence of instructions, or entire bands, does not require prior knowledge of the distribution of any grown programs to continue execution. Just like an isolinear material, the precise location of execution is not critical so long as it falls within a band or close to it. Thus, reconstruction of any Turing program and its execution from filtered bands instantiates a global to local execution that is robust to missing instructions and data, and which degrades gracefully in the face of erroneous starting conditions. Globality, robustness, graceful degradation, and, for that matter, recovery via the Walnut Cake Theorem, are geometrical properties of band-pass filtered perspex programs.

In summary, every aspect of a perspex machine is robust. Isolinear and band-pass filtered programs are robust to missing instructions and data. They degrade gracefully, but non-monotonically. Band-pass filtered programs instantiate global processing. These are all geometrical properties of the perspex machine and do not need any special processing to achieve. Any Turing program can be compiled onto a perspex machine to yield a program that is robust, graceful, and global. We now consider some consequences of this for neurophysiology, psychology, philosophy, AI, and engineering.

It is widely held that recovery from brain injury proceeds by the brain returning to a normal physiological equilibrium and by it recruiting un-damaged parts to undertake important functions formerly undertaken by the injured parts. This is, undoubtedly, a valid account of the recovery, but another mechanism might also be in play. Recovery might be a consequence of slight re-positioning of synapses, and the like, which brings about an improvement in functioning via the Walnut Cake Theorem. If this mechanism is effective then it should be possible to demonstrate it in a healthy brain. Suppose that electrical stimulation of some part of the brain produces a recognisable and useful output, but that

stimulation of a nearby part does not. According to the Walnut Cake Theorem, as stimulation is moved from the effective to the non-effective part the performance will not-decline monotonically, but will go through oscillating periods of decline and recovery. If this effect can be established widely in the brain then it supports the view that biological neurons are subject to the same geometrical constraints as perspex neurons[3]. This would also explain non-monotonicity in learning.

Psychologists have demonstrated, in many learning experiments, that performance does not improve monotonically, but rather improves and declines repeatedly. This may well be caused by gross physiological effects that make parts of memory more or less accessible at different times, but the Walnut Cake Theorem might also be in play. If learning is effected by the positioning of some component, such as a synapse, then non-monotonic learning is to be expected.

AI researchers have attempted to simulate all manner of human performance, including the non-monotonic aspects of learning. For example, various researchers have developed computational models of the learning of arithmetic by children. Children exhibit systematic and non-monotonic errors which the researchers attempt to explain in terms of the computational representations a child might use. The perspex machine could be used to generate an interesting null hypothesis to these models. Implement a program in C, or in any computer language, that performs the arithmetical task set to the children. Compile the program onto a perspex machine and perturb the entry point. The perspex program will exhibit systematic, non-monotonic errors that are a function of the essential spacetime geometry of the perspex machine, not some side effect of symbolic representations. If the perspex errors explain the children's behaviour then no symbolic representations need be invoked. Alternatively, if errors in the symbolic representation can be distinguished from the perspex errors it might be possible to delineate the symbolic and non-symbolic parts of a child's mental arithmetic.

AI researchers have used computational analogues of the physical processes of annealing and diffusion, but the perspex machine makes analogues of all physical processes available. For example, if an isolinear program to control a robot arm does not put the arm in quite the right position then the program might be *bent, stretched, rotated,* or *translated* until it does position the arm correctly. As we have seen, isolinear programs can be placed in contact with each other so that control passes from one to the other, but they could be *welded* together by agitating the position of perspexes at the boundary and allowing some *mixing* before *cooling* the perspexes by reducing the agitation. Programs could be *poured* into *moulds* that define the boundaries of a computation and could be allowed to *set.* If necessary, the programs could be *polished* by introducing small perspexes to give a finer approximation to the boundary conditions. Programs could be *boiled* by agitating perspexes. Relating the boiling point, say, to the absolute determinant of the Cartesian part of a perspex would allow programs to be *distilled.* Distillates with a high boiling point would then have a gross effect on a computation, because they are large perspexes, and distillates with low boiling points would have a small effect, because they are small perspexes. New programs could be produced by mixing the distillates. Indeed, if the perspex thesis[3] is correct, any physical operation can give rise to a computational analogue in the perspex machine.

The isolinear machines become more disorganised as the phase of the entry point approaches the mid-point of two Turing instructions, but becomes organised as the entry point closely approaches a Turing instruction. Genetic algorithms could be implemented by using the entry point as the single gene in an isolinear material that contains many Turing programs. Alternatively, many starting points could be used to obtain an output from a collection of Turing programs embedded in an isolinear material. The Turing programs could be arbitrarily complex, but would be controlled by the simplest of genes that simply turn the programs on or off to different degrees.

Band-pass filtered programs could be used to control robots in hostile environments. For example, it might be vitally important that a Mars rover performs some actions despite damage to its programs. In this case no great harm will be done to Mars if the robot misbehaves as a result of a perspex correction to missing data or instructions, and the risk to the robot might be outweighed by the wasted time involved in ground controllers finding and correcting an error in a Turing program. If an error were discovered in a perspex program, ground controllers could upload the lowest damaged band first and then upload higher bands. Thus, the robot's performance would increase over time as the bands are received, perhaps over an extended period of time if only low bandwidth communication is available.

Band-pass filtered programs could be used to fit programs into smaller memory spaces by discarding the higher bands. The resultant program will generally perform a function similar to the original program. This might be a useful way of reducing large programs to run on small devices, or of running many parallel copies of a program at low resolution in a

fixed memory space to find approximate solutions before deleting the unpromising programs and replacing the promising ones with higher resolution copies.

In addition to its scientific and technological applications the perspex machine is also philosophically interesting. It is a super-Turing machine that cannot be entirely described in language or logic‡, yet philosophy is intimately concerned with analysing linguistic and logical descriptions of possible worlds. Several philosophers have promoted language and logic as the pinnacles of intellectual performance. This is clearly wrong. A perspex machine can do everything that a Turing machine can do, and more, so the non-verbal reasoning inherent in the perspex machine is more powerful than any verbal reasoning. Rather than language being the pinnacle of intellectual prowess, the ability to visualise things in a continuous space is a more competent form of reasoning.

The perspex machine also allows the precise formalisation of various mental phenomena[3] in a way that could be implemented and put to the test in a robot. Such precision is to be welcomed in technical philosophy. The treatment of randomness[3] in the perspex machine might also be of interest.

6. Conclusion

We have demonstrated that convolving a Turing program on a perspex machine produces a continuous machine that includes the Turing program at specific points and similar programs everywhere else in the convolved space. In particular, isolinear programs are robust to missing instructions and data, degrade gracefully in the presence of error in the starting conditions, and sometimes partially recover from such error. Isolinear materials can be used to supply computational analogues of all physical processes, not just the simulated annealing and Brownian diffusion described in the AI and optimisation literature. We argue that band-pass filtered programs are even more robust than isolinear programs and have the additional property of globality. An entire program can be approximated by a single low-resolution instruction. The norm of the accuracy of the approximation increases as higher bands are added, but the actual accuracy generally increases non-monotonically, as given by the Walnut Cake Theorem[3]. This explains non-monotonic learning in animals, and non-monotonic recovery from brain injury, because it is likely that the geometrical spacetime constraints on perspex neurons apply also to biological ones.

Whilst much AI research has been dedicated to developing robust, gracefully degrading, and global symbolic representations, and is no doubt extremely valuable, all of these properties can be had by the simple act of compiling a Turing program into a band-pass filtered perspex program. This might be useful in practical applications where it is desired to "harden" programs written in conventional computer languages, and/or to convert any existing program to time-critical execution by the simple act of recompiling it for a perspex machine.

At the time of publication all of the software described here will be available via the author's web sites[4].

References

1 J.A.D.W. Anderson, "Exact Numerical Computation of the Rational General Linear Transformations" in *Vision Geometry XI* Longin Jan Latecki, David M. Mount, Angela Y. Wu, Editors, Proceedings of the SPIE Vol. 4794, 22-28 (2002).

2 J.A.D.W. Anderson, "Perspex Machine" in *Vision Geometry XI* Longin Jan Latecki, David M. Mount, Angela Y. Wu, Editors, Proceedings of the SPIE Vol. 4794, 10-21 (2002).

3 J.A.D.W. Anderson, "Perspex Machine II: Visualisation" in *Vision Geometry XIII* Longin Jan Latecki, David M. Mount, Angela Y. Wu, Editors, Proceedings of the SPIE Vol. 5675, i.e., this volume.

4 The author's web sites are http://www.reading.ac.uk/~sssander and http://www.bookofparagon.btinternet.co.uk

‡. A derivation of propositional logic from the perspex continuum is known to the author. Thus, the perspex continuum provides operations at a more basic level than the existence and manipulation of the logical atoms *true* and *false*. It seems likely that these logically sub-atomic properties can be arranged to make the emergence of complex machines from a randomly populated perspex space highly probable. If so, physical systems can be designed to manifest complexity as they develop over time.