PERSPEX MACHINE VI: A GRAPHICAL USER INTERFACE TO THE PERSPEX MACHINE

Perspex Machine VI:
A Graphical User Interface to the Perspex Machine

Christopher J.A. Kershaw & James A.D.W. Anderson*
Computer Science, The University of Reading, England

Abstract

The perspex machine is a continuous, super-Turing machine which, in previous work, was simulated programatically on a digital computer in the AI language Pop11. Here we present a C^{++} simulation of the perspex machine, along with a graphical user interface, that can be used to implement, edit, visualise, instrument, and run perspex programs interactively. The interface uses a number of different projections to make 4D perspex-space more accessible to the user.

We also present a new proof of the Walnut Cake Theorem that has much weaker conditions than the previous proof and is, therefore, much more widely applicable. It predicts non-monotonicities in numerical algorithms with sub-quadratic convergence.

Keywords: perspex machine, graphical user interface, GUI, Walnut Cake Theorem.

1. Introduction

The perspex machine is a continuous machine[4] that performs perspective transformations. It was introduced in[2] by unifying the Turing machine with projective geometry. The halting condition was re-specified in[3] and a data-structure, called the "access column," was presented that allows the arbitrary accessing of elements within a perspex. A different approach to accessing the individual elements of a perspex is taken in the universal perspex machine.[6]

The perspex can be a 4×4 matrix with column vectors x, y, z, and t or it can be a computer instruction: $\vec{x}\vec{y} \rightarrow \vec{z}$; $\text{jump}(\vec{z}_{11}, t)$. 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,[2,6] reduced to canonical form,[2,3] 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 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 the halting perspex, H, is encountered.

The elements of a perspex are transreal numbers.[1,3] 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.[4] Various forms of the perspex are illustrated in.[3] For example, the perspex can be a tetrahedron, a perspective transformation, an artificial neuron, and a computer instruction.

The original simulation[3] of the perspex machine was implemented in Pop11. Pop11's text editor was used to code, edit, instrument, and execute programs. PovRay[12] was used off-line to render images of perspexes. Here we describe a C^{++}

* Corresponding author. author@bookofparagon.com, http://www.bookofparagon.com
Computer Science, The University of Reading, Reading, Berkshire, England, RG6 6AY.

simulation of the perspex machine that has a Graphical User Interface (GUI) to facilitate the implementation, editing, instrumentation, and execution of perspex programs. This allows the interactive development and visualisation of perspex programs. The GUI was tested using perspex programs implemented by hand and those generated by a C to perspex compiler.[5] The results of the GUI simulation were verified against the earlier system.[3]

We now describe the C^{++} simulation and the interactivity supplied by the GUI. The paper concludes with suggestions for future work. Finally, the appendix presents a new proof of the Walnut Cake Theorem that has much weaker conditions than the previous proof and is, therefore, much more widely applicable. It predicts non-monotonicities in numerical algorithms with sub-quadratic convergence.

2. Simulation of the Perspex Machine

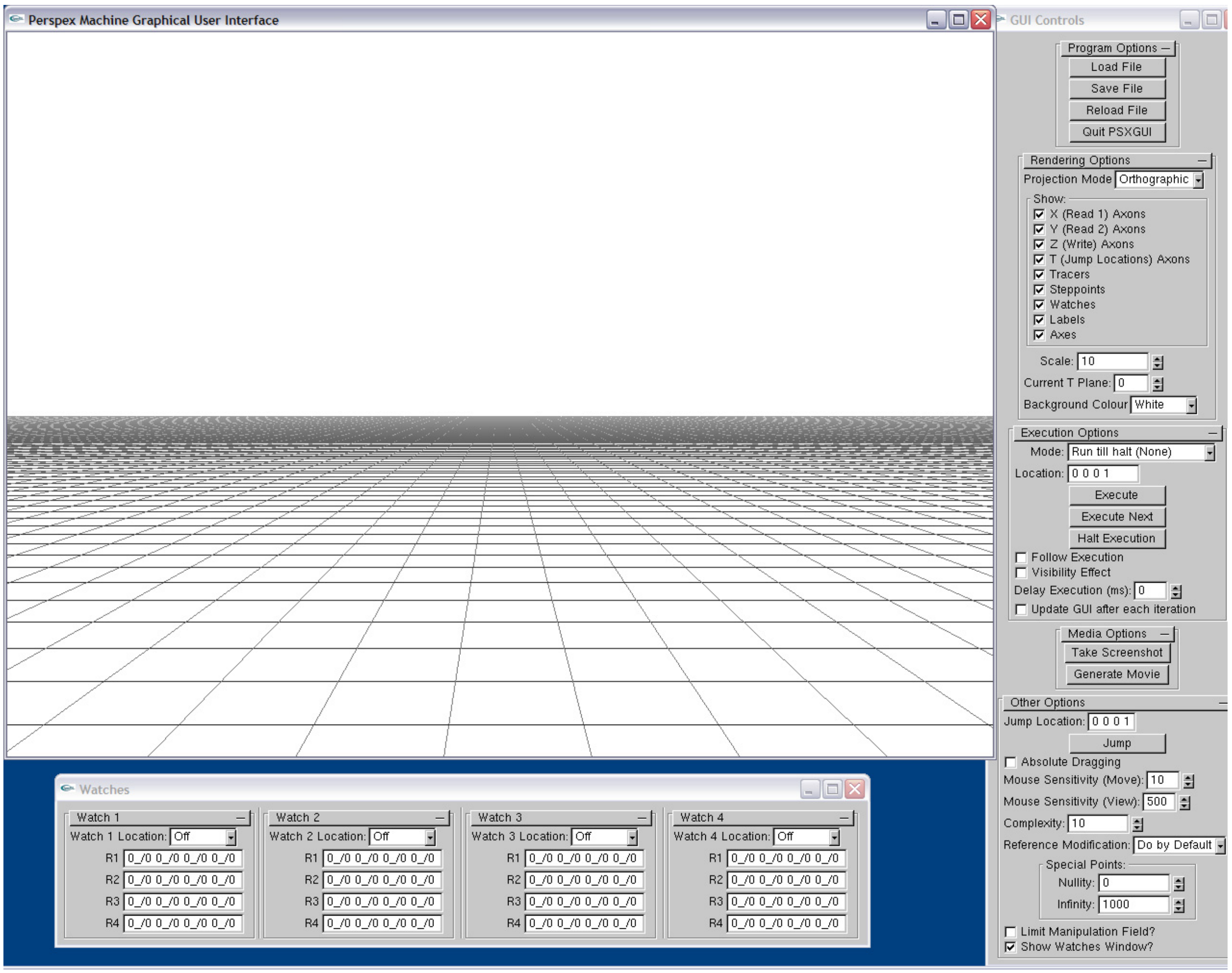
Practical implementations of the perspex machine use transrational arithmetic.[1] A transrational arithmetic class was implemented in C^{++} which traps fractions with zero denominator and calculates the transrational result, but otherwise uses the rational arithmetic package, LEDA,[10] to handle strictly rational numbers. The new class supports the addition, subtraction, multiplication, and division of transrational numbers, as well as C^{++}'s relational operators. Perspexes are implemented as 4×4 matrices of transrational numbers.

The original implementation of the perspex machine,[3] in Pop11, uses a sparse array implemented with hash tables to model perspex space, but the C^{++} version developed here uses nested lists. This is a retrograde development. Hash tables provide faster execution. The fastest execution would be obtained with standard arrays, but this approach would be suitable only for perspex machines of a fixed size and resolution. Nonetheless, the list approach adopted here is adequate to demonstrate user interaction. It has been used on a program with over 600 perspexes that implements Dijkstra's solution to the travelling salesman problem. See Figure 4.

3. Visualisation

The C^{++} implementation of the perspex machine uses OpenGL,[11] along with the GLUT[9] and GLUI[8] libraries, to provide a GUI. Perspexes are drawn in their neuronal form,[3] because this gives a clear indication of where reads, writes, and jumps occur. The C^{++} implementation largely maintains the default colour conventions of the original visualisation software.[3] The read locations, $\vec{x}$ and $\vec{y}$, are rendered in red, the neuron body is rendered in orange, the write location, $\vec{z}$, is rendered in green. This maintains the traffic light analogy of accessing going from red, through orange, to green. Jumps are rendered in blue. The original version of the software uses PovRay[12] and renders axons in marbled shades of cream and pink. The C^{++} version uses white on a black background, but can use black on a white background. Black backgrounds are generally preferred in movies, whereas white backgrounds are generally preferred in printed documents. A neuron's body and synapses are rendered as spheres, the axons as cylinders. The original software[3] rendered synapses as sphere-capped cylinders of larger diameter than the axon, and of a length greater than the radius of a neuron's body, so that when a synapse is placed at the centre of a neuron's body it forms a roughly hemispherical button on the surface of the body.

Figure 1 shows the GUI. The large *view window* shows a grid drawn on the ground plane. This helps orient the user to the spatial axes. Several *control panels* interfacing to *GUI controls* are shown to the right of the view window. The control panels below the view window show where the user has placed a *watch* on perspex space to raise an alarm and begin visualising the space when control passes into a watched region. The GUI controls are divided into several types. *Program Options* deal with loading and saving files, as well as exiting the GUI. *Rendering Options* select the projection method, determine the scale relating perspex-space units to view-window units, and determine whether axes and labels should be drawn on neuron bodies. *Execution Options* control whether execution is to be visualised all the time, or only when it passes into a watched region, or into or out of a trace point, as well as controlling the form of visual feedback that is to be given. *Media Options* allow screen shots and movies to be saved. *Other Options* deal with a miscellany of controls that determine whether dragging a neuron is absolute or relative, what *complexity,* or resolution, to draw the image at, and where to place icons denoting the points at nullity and infinity. A relative drag moves the position of a neuron's body, but leaves its synapses at fixed locations, whereas an absolute drag moves the whole of the perspex.



Projection Mode: Orthographic

Show:

- [✓] X (Read 1) Axons
- [✓] Y (Read 2) Axons
- [✓] Z (Write) Axons
- [✓] T (Jump Locations) Axons
- [✓] Tracers
- [✓] Steppoints
- [✓] Watches
- [✓] Labels
- [✓] Axes

Scale: 10
Current T Plane: 0
Background Colour: White

Mode: Run till halt (None)
Location: 0 0 0 1

- [ ] Follow Execution
- [ ] Visibility Effect

Delay Execution (ms): 0

- [ ] Update GUI after each iteration

Jump Location: 0 0 0 1

- [ ] Absolute Dragging

Mouse Sensitivity (Move): 10
Mouse Sensitivity (View): 500
Complexity: 10
Reference Modification: Do by Default

Special Points:
Nullity: 0
Infinity: 1000

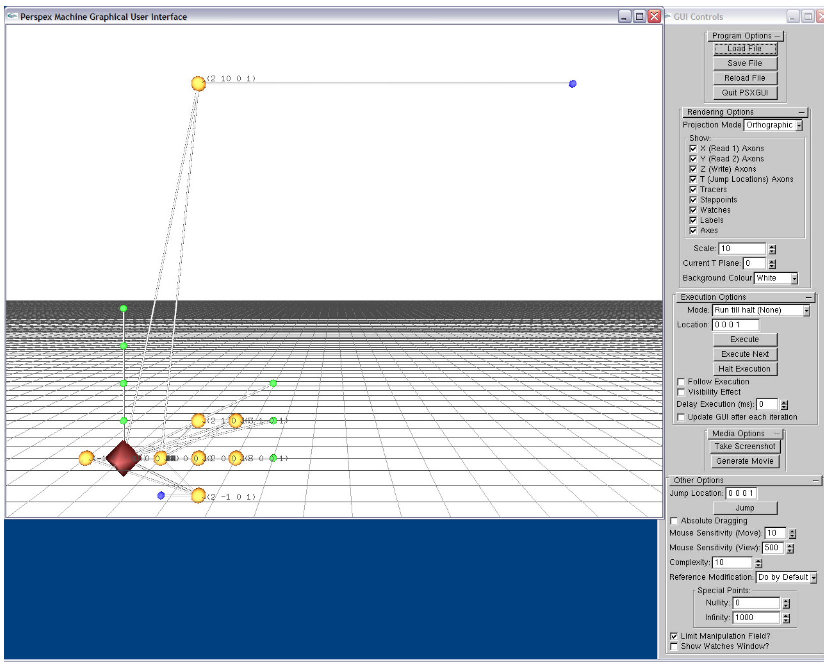
- [ ] Limit Manipulation Field?
- [✓] Show Watches Window?

	Watch 1	Watch 2	Watch 3	Watch 4
Location	Off	Off	Off	Off
R1	0_/0 0_/0 0_/0 0_/0	0_/0 0_/0 0_/0 0_/0	0_/0 0_/0 0_/0 0_/0	0_/0 0_/0 0_/0 0_/0
R2	0_/0 0_/0 0_/0 0_/0	0_/0 0_/0 0_/0 0_/0	0_/0 0_/0 0_/0 0_/0	0_/0 0_/0 0_/0 0_/0
R3	0_/0 0_/0 0_/0 0_/0	0_/0 0_/0 0_/0 0_/0	0_/0 0_/0 0_/0 0_/0	0_/0 0_/0 0_/0 0_/0
R4	0_/0 0_/0 0_/0 0_/0	0_/0 0_/0 0_/0 0_/0	0_/0 0_/0 0_/0 0_/0	0_/0 0_/0 0_/0 0_/0

Figure 1: C^{++} Perspex GUI

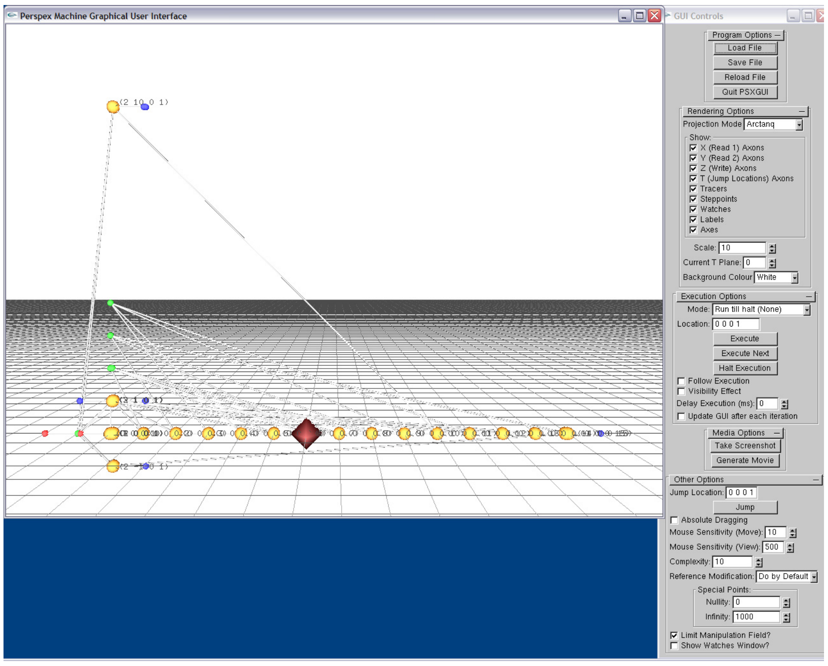
The user can navigate through perspex space by changing the position and orientation of the *camera* that visualises the space. A middle mouse-click toggles the mouse focus from moving in the desktop to moving the camera. By default, mouse movements change the position of the camera in the *x*- and *z*-axes, and rotation of the mouse wheel changes the position in the *y*-axis. Thus, the mouse is used as a 3D positioning device with simultaneous manipulation of each axis of space. By contrast, many GUIs only support 2D manipulations via the mouse. Clicking the left mouse button toggles between changing the position of the camera and changing its orientation. Movement of the mouse and its wheel then controls rotation about the camera's orthogonal *Up, Right,* and *View* axes. The rotation is carried out using quaternions so that rotations may be added, giving an incremental look and feel to rotations of the camera as well as to translations of its position. Keyboard versions of all mouse controls are also supported.

All of the rendering of 3D space is handled by OpenGL, but the user must select a projection method that projects 4D perspex-space into 3D before OpenGL is called. The simplest method, Figure 2, is an orthogonal projection in the time axis. This has the benefit of showing the spatial layout of the perspexes. The position of the current perspex being executed can then be animated. However, it has a serious disadvantage when viewed as a still image. Perspexes at different times can project onto the same position in space. This problem is eased by *frame projection* (called "plane" projection), that shows the contents of perspex space, at a fixed time, in orthogonal projection. Axons connecting to earlier times are shown projecting to a point to the left of the frame and axons connecting to a later time are shown projecting to a point to the right of the frame. This maintains the conventional relationship of time increasing to the right. The user can scroll through the frames as in conventional movie-editing software.



Projection Mode: Orthographic
Show:
☑ X (Read 1) Axons
☑ Y (Read 2) Axons
☑ Z (Write) Axons
☑ T (Jump Locations) Axons
☑ Tracers
☑ Steppoints
☑ Watches
☑ Labels
☑ Axes
Scale: 10
Current T Plane: 0
Background Colour: White
Mode: Run till halt (None)
Location: 0 0 0 1
☐ Follow Execution
☐ Visibility Effect
Delay Execution (ms): 0
☐ Update GUI after each iteration
Jump Location: 0 0 0 1
☐ Absolute Dragging
Mouse Sensitivity (Move): 10
Mouse Sensitivity (View): 500
Complexity: 10
Reference Modification: Do by Default
Special Points:
Nullity: 0
Infinity: 1000
☑ Limit Manipulation Field?
☐ Show Watches Window?

Figure 2: Orthographic projection of a perspex program that computes Fibonacci numbers.



Projection Mode: Arctanq
Show:
☑ X (Read 1) Axons
☑ Y (Read 2) Axons
☑ Z (Write) Axons
☑ T (Jump Locations) Axons
☑ Tracers
☑ Steppoints
☑ Watches
☑ Labels
☑ Axes
Scale: 10
Current T Plane: 0
Background Colour: White
Mode: Run till halt (None)
Location: 0 0 0 1
☐ Follow Execution
☐ Visibility Effect
Delay Execution (ms): 0
☐ Update GUI after each iteration
Jump Location: 0 0 0 1
☐ Absolute Dragging
Mouse Sensitivity (Move): 10
Mouse Sensitivity (View): 500
Complexity: 10
Reference Modification: Do by Default
Special Points:
Nullity: 0
Infinity: 1000
☑ Limit Manipulation Field?
☐ Show Watches Window?

Figure 3: Arctanq projection of the perspex program that computes Fibonacci numbers.

The user can also select an *arctanq* projection,[3] Figure 3, that maps the whole of space at a fixed time onto a unit cube and then lays out the cubes from left to right on the *x*-axis in temporal order. Like frame projection, this maintains the conventional association of earlier time lying to the left and later time lying to the right. As many hand written[3] and compiled[5] perspex programs use a jump of unity in time this projection creates an analogue of the time line running along the *x*-axis.

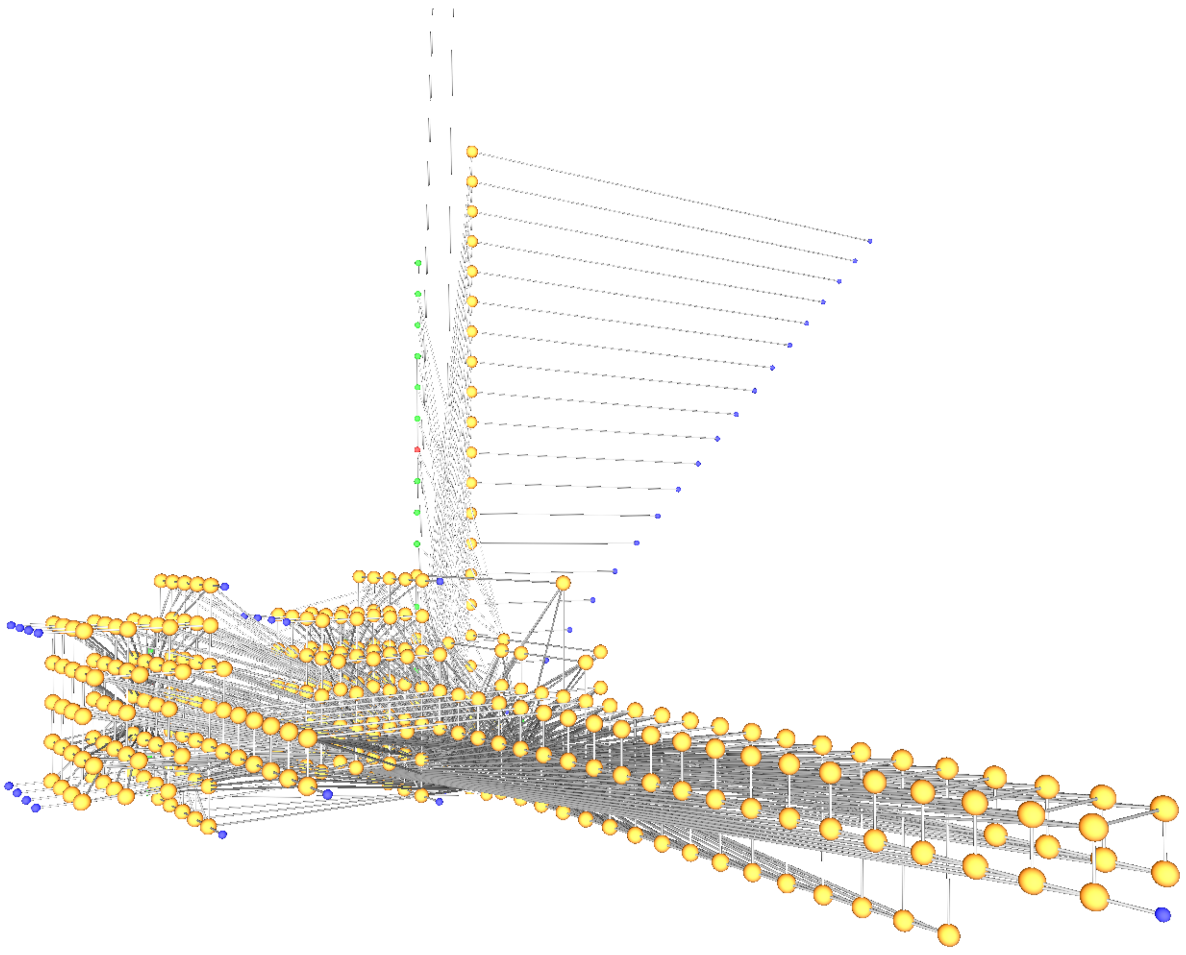


Figure 4: Dijkstra's algorithm for solving the travelling salesman problem, compiled from C into perspexes.[5]

4. Editing, Execution, and Instrumentation

The C to perspex compiler[5] generates code in a perspex-interchange-language that the present GUI can read and write. The interchange language uses only Turing symbols, but this is sufficient to communicate between the compiler and the GUI, because they are Turing computable. A class of theoretical, super-Turing, perspex spaces can be described by the language if it is given access to super-symbols, that is Turing incomputable, irrational numbers. However, the language can describe only countably many perspexes, because it has only countably many place holders for symbols of any kind. Therefore, general perspex spaces cannot be described in the language. However, this is not a practical constraint when computer simulations of the perspex machine are used, as here.

The advantage of having an interchange language is that a C programmer can give compiler directives, "pragmas," that turn on or off various parts of the GUI and its instrumentation of a perspex program. This facilitates the debugging of programs and, in future, might support a tighter integration between the compiler and the GUI within one Human Computer Interface (HCI).

The user can instruct the GUI to load files of perspex programs implemented in the interchange language. Alternatively, the user can enter perspexes by filling out forms in the interface. However a program is obtained it can be edited by dragging and dropping parts of a perspex neuron, or by selecting the neuron and overwriting its values in a pop-up form. Selecting a neuron with a left mouse-click gives visual feedback on the neuron selected. A right-mouse click additionally pops up a menu that offers editing and instrumentation options. Keyboard input is also available as an alternative to all mouse interactions and has the advantage that movements and changes are under exact control of keyboard clicks, not analogue movements of the mouse.

The selection of parts of a program, and visualisation of its operation, can be simplified by turning on or off the rendering of parts of a neuron. The user can individually select the types of axons that are to be shown, as well as selecting visual feedback modes. The user can select single stepping of a perspex program, and can turn single stepping on or off at trace points. When execution passes through a trace point it takes on the colour of the trace point so the user can follow the behaviour of a program between trace points. The user can also set generic trace conditions that turn tracing on when a perspex being read, written, executed, or jumped to has certain values. These trace conditions can also be assigned a unique colour so that the user can visually trace execution back to the point where the condition was met, or can trace forward to see how the program handles the condition. This is a particularly effective way of tracing how "erroneous" perspexes are produced and handled. However, this raises the problem that several trace conditions can be satisfied at a single perspex. This ambiguity is resolved by the user specifying the conditions in a priority list so that only the highest priority condition changes the colour of the trace.

The user can also set watches on a region of space so that the visual display is turned on whenever execution is within the watched region. A watch can also display the values of a perspex at a given location. Watches are particularly helpful for identifying where a perspex program accesses peripheral devices or internal structures that are built into perspex space at fixed locations.

Perhaps the most useful feedback mode is a "slip stream" mode in which the most recently executed perspexes are shown brightest, with intensity decaying over the sequence of perspexes visited. This shows a slip stream, or tail, of the active regions of a program. It is a very intuitive way of following the flow of control through perspex space.[7]

The ultimately verbose form of instrumentation is to print out the current location, execution, read, and write perspexes, and to display a thumbnail view of perspex space. This is akin to using a binary debugger in a conventional programming language.

5. Conclusion

We describe a GUI that can be used to create perspex programs or read perspex programs generated by a C to perspex compiler. The GUI can be used to edit programs and to instrument their execution. As should be expected of a geometrical programming system, such as the perspex machine, the GUI is more direct and more visual than GUIs for conventional computing languages. The GUI can also be controlled by pragmas in the C source that is compiled into perspexes. In future, this might allow the tighter integration of the GUI and the compiler into a single interface. It would certainly allow the compiler to generate more sophisticated and fuller instrumentation than can be achieved by a human user. For example, the GUI could be instructed to enforce analogue interpretations of perspexes arranged in columns[2,3,5] so that drag and drop changes a program analogically. As analogue representations seem to arise naturally in both hand coded[3] and compiled[5] programs, this would seem to be a critical aspect for future development.

The GUI is presently used by a C to perspex compiler,[5] but it is equally useful to any compiler that generates descriptions of perspexes in the interchange language. We intend to develop this language as perspex research progresses. It might be sensible to develop a high-level, perspex language, but, as a first step, the decision to compile C into perspexes provides a route to compile almost all scientifically and commercially significant languages into perspexes by using cross-compilers to C. The results can then be visualised and edited in the GUI developed here.

The appendix sets out a new, more general, proof of the Walnut Cake Theorem. The theorem makes testable predictions about all manner of paradigm shifts, including predicting non-monotonicities in numerical algorithms with sub-quadratic convergence.

6. Appendix

Turing machines are inherently symbolic, but this is not to say that they are devoid of spatial properties.[6] It is well accepted that a Turing machine's tape is a one dimensional, oriented object with distinct directions "left" and "right." [14,15] Further, in a footnote on page 249, Turing defines[14] that symbols are compact spaces. It follows that Turing's "m-configuration" [14] is a lattice of compact spaces. Hence, the whole of Turing's machine is a lattice of compact spaces. Furthermore, this lattice is embedded in a continuous space, as claimed by Turing[16] in the final paragraph of page 439. Specifically, Turing machines exist on lattices embedded in a continuous, perspex space.[2] Thus, any symbolic computation is subject to spatial constraints which, if not exploited by the machine itself, may be exploited by a meta-machine emulating or simulating it. (A perspex machine may *emulate* a Turing machine exactly or else *simulate* it approximately by exploiting geometrical properties not present within the symbols of the Turing machine at the nodes of the lattice, but present in the perspex space in which the Turing machine's lattice is embedded.) One important, and very general, spatial constraint affects how a discrete system describes a finer discrete or continuous system. We now present a proof of the Walnut Cake Theorem that has much weaker conditions than the proof in[3] and is, therefore, much more widely applicable. We begin by reciting the introduction to the proof without change.[3]

Without loss of generality, consider the bounded segment of the real number line $[0, 1]$. This is shown in Figure 5 as the circumference of a circle drawn clockwise from 0 to 1. (As rational bounds on this segment are symmetrical about $1/2$ this circular figure has the advantage of illustrating the symmetry.)

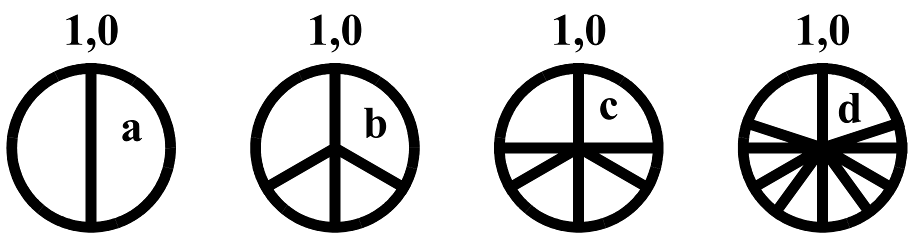


Figure 5: Successive rational bounds (walnut cake).

We suppose that measurements are made by making a cut from the centre of the circle to the circumference. In the first generation 2 cuts are made, dividing the circle into 2 equal parts of size (a). In the second generation 3 cuts are made, dividing the circle into 3 equal parts of size (b), in addition to some parts surviving from earlier cuts. There are just 4 segments (sectors) in the circle (b), not $2+3 = 5$, because of the common cuts at the position $2/2 = 3/3 = 1$. In the third generation 4 cuts are made, but there are just 6 segments, not $2+3+4 = 9$, because of the additional common cuts at $4/4 = 1$ and $2/4 = 1/2$. This process continues without limit. At each stage the potential number of cuts is given by the arithmetical sequence $2, 3, 4, \ldots, d$, but all of the repeated cuts are removed from the sequence.

We now introduce new material which removes the requirement that a complete, arithmetic sequence is present. A *normal* measurement is defined to be a measurement, at a higher precision than a sequence of measurements, that does not tighten the bounds on the true value established by the sequence, whereas a *revolutionary* measurement is at a higher precision and does tighten the bounds. We define, further, that a *paradigm shift* is a revolutionary measurement that occurs after a normal measurement. We now derive an upper bound on the precision at which a revolutionary measurement takes place, this bound is the same as in,[3] and we derive a lower bound on the number of cuts in a sequence of rational measurements, excluding repetitions of a cut. This lower bound is looser than the bound in,[3] though that bound depended on having a complete arithmetic sequence. The bounds are used to establish the Walnut Cake Theorem which proves, under certain weak conditions, that, firstly, a sequence of rational measurements goes through alternating sequences of normal and revolutionary measurements and, secondly, that the accuracy of a normal measurement is, in general, strictly less than its preceding revolutionary measurement. That is, under the given weak conditions, rational measurements go through paradigm shifts.

Definition 1: A measurement sequence is denoted by a list, m_d, of increasing integers selected from 2 to d inclusive, but always containing the number d as its largest element. These numbers are the denominators of rational measurements shown in Figure 5. The sequence denoted by $m_d = 2$, with $d = 2$, is the trivial sequence. A non-trivial sequence has $d > 2$. For example, $m_4 = 2, 4$, $m_4 = 2, 3, 4$, $m_4 = 3, 4$, and $m_4 = 4$ all denote well formed measurement sequences. If it is desired to distinguish measurement sequences with the same greatest term d, this may be done using a second, arbitrary, subscript i in $m_{d,i}$. For example, to rehearse the sequence just given, we might state that $m_{4,1} = 2, 4$, $m_{4,2} = 2, 3, 4$, $m_{4,3} = 3, 4$, and $m_{4,4} = 4$. Thus, we have defined the terms used in the proof and defined some background terms that may help the reader put the proof in context.

Lemma 1: Lower bound. The number of cuts, $c(m_d)$, in the measurement sequence denoted by m_d, excluding repeated cuts, is given by $c(m_d) \geq 1 + \sum_i (r_i - 1)$, where the r_i are the largest, relatively prime terms in the measurement sequence. For example, in $m_4 = 2, 3, 4$ we have $r_1 = 4$ and $r_2 = 3$, with no other relatively prime terms, giving $c(m_d) \geq 1 + (4-1) + (3-1) = 1 + 3 + 2 = 6$, which is in agreement with Figure 5 (c).

Proof 1: There are exactly d cuts denoted by $m_d = d$. In this case $r_1 = d$ and $c(m_d) \geq 1 + \sum_i (r_i - 1) = 1 + (d-1) = d$, as required. In all cases, the largest available term r_2 that is relatively prime with r_1, adds r_2 cuts, with one repeated cut at $r_2/r_2 = 1$, in other words it adds $r_2 - 1$ unrepeated cuts. In the case of at least two terms, $c(m_d) \geq 1 + \sum_i (r_i - 1) = 1 + (d-1) + (r_2 - 1) + \ldots = d + (r_2 - 1) + \ldots$. Similarly, each of the largest available $r_{i>1}$, that are relatively prime with all of the preceding $r_1, r_2, \ldots, r_{i-1}$, adds $r_i - 1$ cuts, giving $c(m_d) \geq 1 + \sum_i (r_i - 1)$. However, large terms that are not relatively prime to the $r_1, r_2, \ldots, r_i$ can contribute some unrepeated cuts so the inequality is required. For example, with d even and $d > 4$ the term $d - 2$ is not relatively prime with $r_1 = d$, because it has a common factor 2, but contributes at least one cut at $1/(d-2)$. This establishes the lower bound on the number of unrepeated cuts.

Lemma 2: Upper bound. If all segments with denominators $2, 3, \ldots, d$ have been constructed, then a measurement at the single precision $1/d^2$ subdivides every segment, giving rise to an increase in precision everywhere.[3]

Proof 2: A segment arises from exactly two bounds. The smallest constructed segment has a numerator of unity and denominator as large as possible. That is, the smallest segment has size:

$$1/(d-1) - 1/d = 1/(d^2 - d).$$

Therefore, segmenting at the single precision $1/d^2$ subdivides every segment constructed up to precision $1/d$. The lemma can be stated in other words: given a sequence of measurements up to precision $1/d$, a revolutionary measurement occurs on or before a measurement at precision $1/d^2$.

Lemma 3: If $r_2 > 2$ in m_d then a measurement at the single precision $1/(d+1)$ cannot subdivide every segment denoted by m_d so it cannot increase precision everywhere.

Proof 3: In m_d, $r_1 = d$. Suppose that r_2 takes on the least value greater than 2, i.e. $r_2 = 3$, then $c(m_d) \geq 1 + \sum_i (r_i - 1) = 1 + (d-1) + (3-1) = d + 2$. The single measurement at precision $1/(d+1)$ has exactly $d+1$ cuts, which is insufficient to subdivide all of the $d+2$ segments denoted by m_d. This result obtains similarly if $r_2 > 3$.

Walnut Cake Theorem: In general, a sequence of rational measurements, denoted by m_d, has many terms, giving rise to many relatively prime factors r_i and, in particular, $r_2 \gg 2$. Therefore, by lemma 3, there is a low probability that a single measurement at precision $1/(d+1)$ will tighten the bounds on the true value established by m_d. This is to say that there is a low probability that the measurement is revolutionary and a complementarily high probability that the measurement is normal. Consequently, there is a vanishingly small probability that all successive measurements at precisions $1/(d+i)$, with $i = 1, 2, \ldots$, will be revolutionary, which is to say that the probability of some subsequent measurement, $1/(d+i) = 1/k$, being normal tends to one. But then, by lemma 2, m_k is subject to a revolutionary measurement at or before a measurement at precision $1/k^2$. This establishes that, in general, a sequence of rational measurements goes through one or more normal measurements followed by at least one revolutionary measurement, which is to say that it goes through a paradigm shift. By recursion of the argument just given, the sequence is subject to many paradigm shifts.

Corollary: In general, a normal measurement is strictly less accurate than its preceding revolutionary measurement.

Proof: A normal measurement is at a higher precision than its preceding, revolutionary measurement, but it does not tighten the bounds on the true value. Therefore, neither the upper nor lower, normal bound lies between the revolutionary upper and lower bounds, which is to say that the normal measurement is no more accurate than its preceding revolutionary one.

If either revolutionary bound is a factor of the normal bound then this one bound is common. In this case the accuracy of the normal measurement is equal to its preceding revolutionary measurement. However, in a general sequence of measurements, the probability that a revolutionary bound is a factor of a normal bound is low so, in general, a normal measurement is strictly less accurate than its preceding, revolutionary measurement.

The theorem and corollary above deal with computable[13] measurements that have two bounds: an upper and a lower bound. An extension of the theorem and corollary that deal with the probability of increasing the accuracy of semi-computable[13] measurements, that is measurements with only one bound, being either an upper or else a lower bound, is known to the author, but is beyond the scope of this paper.

A consequence of the Walnut Cake Theorem is that, in general, a Turing machine that describes a finer discrete or continuous system operates non-monotonically, going through repeated cycles of normal and revolutionary measurements. In other words it goes through paradigm shifts. This is a consequence of the metrical properties of space. A consequence for Artificial Intelligence is that when a Turing machine is instantiated in a physical computer with real-world sensors and effectors, it is subject to paradigm shifts. A consequence for biology is that, on the assumption that a genotype is a discrete approximation of a finer discrete or continuous phenotype, then evolution proceeds through punctuated equilibria, that is, biological evolution is subject to paradigm shifts.

The above non-monotonic behaviour is quite general, but it is possible to avoid it in special cases. If the sequences of measurements is restricted to binary numbers, not the more dense rational numbers, there is no non-monotonicity. However, this is of little practical help because practical computer-programs process real-world signals in floating-point numbers and these are subject to the above non-monotonicity. Consequently, floating-point, numerical algorithms with sub-quadratic convergence are subject to the non-monotonicity just described, despite proofs of absolute convergence from real analysis. The point at issue being that floating-point numbers are a proper sub-set of the real numbers so the Walnut Cake Theorem holds regardless of differential calculus.

Monotonic logics are, of course, possible. One ascribes a non-binary, world measurement to a predicate with the binary values True or False. Logical operations on the predicates, but not the measurements themselves, are then immune from the above non-monotonicity because they are expressed in binary arithmetic.

A wider, but less useful, class of exceptions is the sequences of measurements where the successive denominators are sufficiently widely spaced to prevent paradigm shifts. In practical cases, one abstains from making measurements until the measurement procedure has improved sufficiently to guarantee that no non-monotonicity will be measured. But this is wasteful and impractical. For example, one might wish to abstain from engineering genetic improvements until one is sure that the engineering procedure will deliver an improvement, but it would be impractical to come by this certainty without conducting experiments that are prone to be less successful. A scientific procedure of accepting gains and reverses has the practical advantage that it allows the continual testing of hypotheses, rather than outlawing testing of certain hypotheses until one is certain that an experiment will improve on all of its predecessors.

A very narrow, but welcome, class of exceptions is the sequences of close measurements where successive denominators do improve the accuracy of the measurement. Such accidental runs of good luck are to be accepted wherever they occur, but they cannot be relied on.

In conclusion, the Walnut Cake Theorem explains how paradigm shifts arise from discrete approximations to finer discrete or continuous systems. If the universe is discrete, as some readings of quantum mechanics would have it, then continuous systems cannot exist. Therefore, all approximations are discrete approximations to a finer discrete system. In this case, the Walnut Cake Theorem applies to everything. In particular, the theorem explains why biological evolution proceeds by punctuated equilibria, explains the folk psychology of having good days and bad days, and justifies the aphorism, “If it ain’t broke don’t fix it.” Any attempt to make a small improvement to anything is likely to fail, but sometimes the attempt is wildly successful. And, very occasionally, one can have a run of extraordinary success. All of this arises from a geometrical property of space.

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, 100-111 (2005).

4 J.A.D.W. Anderson, "Perspex Machine III: Continuity Over the Turing Operations" in *Vision Geometry XIII* Longin Jan Latecki, David M. Mount, Angela Y. Wu, Editors, Proceedings of the SPIE Vol. 5675, 112-123 (2005).

5 M. Spanner & J.A.D.W. Anderson, "Perspex Machine V: Compilation of C Programs" in *Vision Geometry XIV* Longin Jan Latecki, David M. Mount, Angela Y. Wu, Editors, Proceedings of the SPIE Vol. 6066 (2006).

6 J.A.D.W. Anderson, "Perspex Machine VII: The Universal Perspex Machine" in *Vision Geometry XIV* Longin Jan Latecki, David M. Mount, Angela Y. Wu, Editors, Proceedings of the SPIE Vol. 6066 (2006).

7 The corresponding author's web site gives examples of C programs that are compiled into perspexes and movies of their execution: www.bookofparagon.com was verified on 21 June 2005.

8 GLUI Is a GLUT-based C++ user interface libraries. This free software is supported on many computer architectures and is available at sourceforge.net/projects/glui as verified on 21 June 2005.

9 GLUT is the OpenGL Utility Toolkit. This free software is supported on many computer architectures and is available at www.opengl.org/resources/libraries/glut.html as verified on 21 June 2005.

10 LEDA Is a commercial, rational arithmetic, topology, and geometry package. It is available at www.algorithmic-solutions.com/enleda.htm as verified on 21 June 2005.

11 OpenGL The Open Graphics Library is an Applications Program Interface for a graphics engine. It is supported on many computer architectures. A description of it is available at www.opengl.org as verified on 21 June 2005.

12 PovRay The Persistence of Vision Ray Tracer is free software. It is supported on many computer architectures and is available at www.povray.org as verified on 21 June 2005.

13 Pour-El, M.B. & Richards, I.J. *Computability in Analysis and Physics,* Springer Verlag (1989).

14 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).

15 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).

16 Turing, A.M. "Computing Machinery and Intelligence" in *Mind,* vol. LIX, no. 236, (1950).