The VIPA project - some notes on the pedagogical approach to design education using active 3d worlds

Paul Coates, Robert Thum. Christian Derix

p.s.coates@uel.ac.uk;r.p.thum@uel.ac.uk;c.derix@uel.ac.uk

University of East London School of Architecture and the Visual Arts,

http://uel.ac.uk/ceca

Abstract. This paper should be seen as complementary to the other papers by Mullins et al where the EU VIPA project is described. It is the intention of this paper to deal solely with the educational aspects of using computers in architectural education, which of course is the raison d'etre of both ECAADE and the VIPA project, but on this occasion to look at the earliest models of this, to revisit the pioneers in case we have forgotten something, and to see in what way their original aims were achieved, and how their self proclaimed task can be transferred into the current situation.

It is the intention of this paper to argue that Papert and Kay's use of the computer in education still has many useful things to say in the context of vipa.

The general approach is fundamental to the VIPA project and will inform the ongoing design of the scripting and modelling platform, based on Smalltalk / Open Croquet, and Blender / Python.

.

Keywords. LOGO, Smalltalk, emergence, computer based education.

Introduction

This paper proposes that the best way to encourage design students to think about space and form as dynamic in nature is to embed the learner in a 3d simulation which they can easily design and change through the medium of scripting.

By reiterating the principles of the founding fathers of computing and education in particular Papert and Kay, we hope to work towards a reapplication of the principle notions of how the computer can make a difference in the process of learning when the computer is not used only as a drafting or modelling tool but as a **'tool to think with**' in the Papertian Constructionist way (see paper by M. Mullins for this conference for a discussion of Constructionism and other educational ideas in the context of VIPA) .

This approach to design leads to an active engagement with the machine. The difference between active and passive use is that of representing architectural designs as code scripts generating emergent outcomes, and on the other hand representing them as static geometry.

The paper will first look at the origins of the use of computers as creative tools for learning in order understand the original motivation of the pioneers. These could perhaps be summarised as: the need to encourage abstract thought rather than learn the standard procedures, and the aspect of the unexplored side of computer literacy, that it is not only necessary to be able to read the new media, but also to write them too.

In the second part of the paper an example will illustrate the thesis with one example; that of using agents to define spatial relations.

Origins

Seymour Papert (1928 -) was, with Marvin Minsky, one of the people in at the birth of the MIT Artificial Intelligence project in the late '50s. It was a mixture of computing science, philosophy and mathematics, and initially gave birth to LISP, an elegant formalism of an artificial language upon whose foundations many other languages and projects were based. Mostly AI was making the assumption that intelligence was best seen as the application of formal systems of logic such as first order predicate calculus. This emphasis on well defined syntactically consistent artificial languages is a leitmotif in the writings of Papert and others (especially McCarthy of course who developed the language[1]). It may be that Chomsky's "syntactic structures" (1957)[2], which showed how to define a human language as a recursively defined structure of functions gave an impetus to the development of LISP, and it was certainly seen as a conceptual leap above BASIC, FORTRAN and all the engineering based languages, which were ill structured, inconsistent and clumsy. Papert reserves many words for abuse of BASIC in talking about using computers to teach children.

Papert also worked with Piaget (1896 – 1980) in the '60s at his "international centre for genetic epistemology" (1955 – 80) and it is this experience which he credits for turning him towards the study of learning and the development of strategies for learning, "making children into epistemologists" as he says.

In his development of the Turtle / computer as an object to think with, Papert describes it as

"an object in which there is an intersection of cultural presence, embedded knowledge and the possibility for personal identification"

(Quotes here and below are all taken from Papert (1980))

Thus bridges can be made between the turtle geometry and the body geometry of the child. Added to this was the aim of providing an environment where the learner is "embedded in a learning environment as one is as an infant learning to speak". The computer, if sufficiently flexible and friendly could provide this environment and "as in a good art class the child is learning technical knowledge as a means to get to a creative and personally defined end." This is summed up by his assertion that "there will be a product".

Papert is of course mostly talking about new approaches to making children understand the fundamentals of mathematics s and geometry, which might be thought a far cry from teaching adults about architecture. Papert's assertion that "Our culture is relatively poor in models of systematic procedures, there is no word for nested loops or bug or debugging" may seem irrelevant in the architectural context, but it is often illuminating for students to look again at the age old conundrums of space and spatial organisation using a somewhat recalcitrant (but always listening) computer to tease out the convolutions in some idea about space and form.

He is particularly keen on pointing out that Logo hides the mechanics (the plumbing) of the interaction with the computer, In Papert's case this is described by comparing logo with basic which he describes as engineering based unsophisticated

[1] History of Lisp Stanford U 1979. "My desire for an algebraic list processing language for artificial intelligence work on the IBM 704 computer arose in the summer of 1956 during the Dartmouth Summer Research Project on Artificial Intelligence which was the first organized study of AI. During this meeting, Newell, Shaw and Simon described IPL 2, a list processing language for Rand Corporation's JOHN-NIAC computer in which they implemented their Logic Theorist program"

[2] Chomsky is said to have been influenced by Jehoshua Bar-Hillel at Harvard who defined "the first generative grammar" : the history of phrase structure grammar *Proceedings of the ESSLLI 2000 Workshop on Linguistic Theory and Grammar Implementation, Birmingham, Great Britain,2000.*

and only originally justifiable with reference to the tiny memories of the original educational computers ("there will soon be machines more powerful than the Tandy radio shack computer"" a fervid hope that came true and that we all shared in 1979).

"LOGO is more powerful, more general and more intelligible", he finally asserts. As an example of this, consider the idea of recursion. First of all, why should an architectural student want to explore this technique? Partly it is because after the simple <u>loop</u> has been understood it provides a way of thinking about more subtle and interesting things that for instance, encapsulate the ideas of self similarity and yet difference across scales. The idea simplifies and explains seemingly complex objects, and it is a clear way to thinking about many branching and subdivision type systems which are often needed in Architecture. The idea that something should be made out of just itself is an odd one, and hard to grasp. There seems to be something missing, the idea lacks the **subject > object** relation of the observable world. Using a recursively defined language where a function can be arranged to call itself as its own argument is a clear demonstration that it is both possible and easy to understand. The mechanics are set out and LO! A cathedral emerges! Many researchers have shown that particular architectural artefacts such those of Palladio, Wright or Gaudi can be better understood as recursive e.g. Mitchell(1990) , but although it is useful to read these texts, a real understanding and development of the ideas can only come from working with the ideas as expressed in scripting.

Once this has been observed and digested the student can go on to modify the code and try to match the mechanics to the architectural problem at hand. This contrasts with the imposition of arbitrary geometrical patterns onto an architectural idea where the actual structure of the generator is unexplored and unmatched in any way to the problem at hand. Another advantage of working this way is that students are encouraged to be explicit about their design intentions, and then translate their intentions into algorithmic relations, producing a multiplicity of relevant but often counter intuitive outcomes which challenging their learnt preconceptions.

Alan Kay (1940 -)

Papert himself recognised Kay as one of the people he saw as working towards his own goals. Throughout the '70s at the Xerox research institute and also at MIT and latterly Apple computers Kay worked constantly to design a machine that would create the interactive self learning experience that Papert was also looking for. (Variously dubbed the Dynabook or Flex Machine) Essentially Kay was a computer programmer, and his essay "the early history of Smalltalk" (Kay 1993) is a fundamental look at the way the design of a language can derive from a philosophy of education. Kay is more likely to provide a good technical quote than Papert; in the new media reader's collection of essays Wardrip-Fruin & Montfort (2003) Kay's "personal dynamic media" is introduced with a reference to Papert, followed by his remark:

"I was possessed by the analogy between print literacy and LOGO. While designing the FLEX machine I had believed that end users needed to be able to program before the computer could become truly theirs—but here was a real demonstration, and with children! The ability to 'read' a medium means you can *access* materials and tools generated by others. The ability to 'write' in a medium means you can *generate* materials and tools for others. You must have both to be literate. In print writing, the tools you generate are rhetorical; they demonstrate and convince. In computer writing, the tools you generate are processes; they simulate and decide." ("User Interface: A Personal View," 1993)

In his categorisation of programming languages he shares Papert’s scorn for basic et al:

“Programming languages can be categorized in a number of ways: imperative, applicative, logic-based, problem-oriented, etc. But they all seem to be either an "agglutination of features" or a "crystallization of style." COBOL, PL/1, Ada, etc., belong to the first kind; LISP, APL-- and Smalltalk--are the second kind. It is probably not an accident that the agglutinative languages all seem to have been instigated by committees, and the crystallization languages by a single person.”

The distinction between the committee language and the personal creation is quite nice, but one can also to this as a distinction between a computer language as an engineering project (FORTRAN, BASIC etc) and as a linguistic one. Kay begins his history of “Object Oriented Programming” (a term now widely used that he invented 30 years ago)[3] with some notes on the early designers of simulation systems where both data and the operations needed to describe them were held together in a self describing way.
What Kay helped to invent was the essence of the new paradigm of computing where the representation of data and program is held together. Kay is scathing about the clunky way that the agglomerative languages have developed with a strict separation between the coding of the algorithms and the data they need to work on. Kay’s eventual description of his vision of computing was

“In computer terms, Smalltalk is a recursion on the notion of computer itself. Instead of dividing "computer stuff" into things each less strong than the whole--like data structures, procedures, and functions which are the usual paraphernalia of programming languages--each Smalltalk object is a recursion on the entire possibilities of the computer. Thus its semantics are a bit like having thousands and thousands of computer all hooked together by a very fast network. Questions of concrete representation can thus be postponed almost indefinitely because we are mainly concerned that the computers behave appropriately, and are interested in particular strategies only if the results are off or come back too slowly.”

This notion of a programming language being a recursion on the notion of a computer is relevant because any coding scheme tends to encapsulate a model of what this particular machine is. With poorly designed languages the computer is conceptualized as a giant calculator, or maybe a huge database, or of course a flexible typewriter or drawing board. What Kay wants us to consider is a way of programming that refuses to conceptualise the machine in any terms other than its essential self. This approach is a key insight into how to ensure the utmost flexibility and freedom for the students, and to constrain them as little as possible by the design of the language.

Reading and writing and computer literacy

The main lessons to be learnt from this discussion are that while the use of the computer in architectural education can begin and end with just using the medium as presented by the programmers of the application, a much more powerful set of experiences can be encouraged if the s^{t}udent can enter a dialogue with the machine at the level of designing the medium itself, by being able to recast the design intentions and goals in terms that extend the capabilities of the software in front of them. Of

[3] This and later extracts are taken from (Kay 1993) ACM SIGPLAN Notices Volume 28 No 3 March 1993

course these goals can only be achieved if this interaction is transparent, and self evident, with a clearly defined way of writing to the machine that is not cumbersome but also is not toy-like.

In the following section a few examples are presented which hope to show how using scripting allows the development of *more powerful abstractions of the design task* than students can achieve by just drawing. It is often useful for students to use metaphors such as “pressures” on the site, actual physical activities like human movement and so on, but rather than but just illustrating them, with a few days of scripting and a little technical skill and application it is possible to have the exhilarating experience of actually creating and designing simulations with the forces themselves.

The implications for the VIPA platform

What then can we learn from the last 30 years of computer based learning research briefly outlined above?

Firstly it would seem that the task for the designer of an embedded computer based teaching system is to provide useful tools that mirror the operational functions of the world to be explored/understood. [4]

Papert’s turtle is a representation of the learner’s physical life, with axial symmetry and locomotion built in at the fundamental level of existence. Because of this it becomes natural to “take a line for a walk” as Paul Klee said of his drawings Klee 1925), using turtle commands, and as an easy extension it becomes natural to develop small collections of moves (Microworlds as Papert calls them) to encapsulate these ideas into a repertoire of useful things to do. In particular (following Klee) there are aspects of responsiveness to the moves one makes, with particular feedback loops that can be developed leading to unconsidered outcomes (happy accidents, emergent complexes and so on)

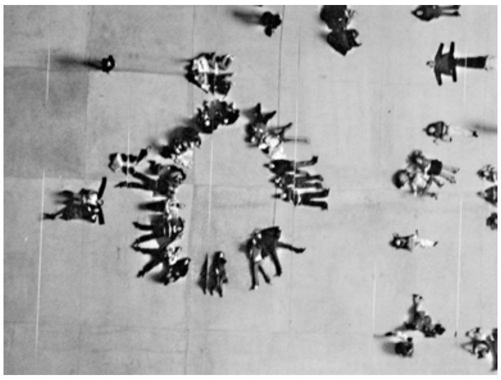
Secondly, following Kay, we should try to present the learning environment not as a broken up collection of algorithms and data, with elaborate protocols and development environments to overcome, but as a seamless web of useful little computers, each one fully functional and capable, and all of which can be used in parallel to develop thought provoking simulations and explorations.

Architecture deals with space and its human occupation, so to develop an exploration of architecture we need to come up with themes that reflect the world type operations of building, manipulating and inhabiting space.

Overarching this division however is the choice of representation; how will the topology and geometry be encoded in the machine? In the following illustrations the representation takes the form of animated particles and fields of forces and emergent 3d networks, which allow the synthesis and embodiment of spatial organisation and envelopes of form. Other representations have been explored, but for reasons of space are not covered. (See Coates & Thum(2000,2001), Coates & Derix (2004).

[4] for instance people have a certain size, they have bodies that occupy space and need certain conditions to be met, the stuff you use to make architecture is big and heavy and eventually come to occupy some particular place on the planet etc.

Illustrations of consensus



The following examples are based on the Papert paradigm of allowing the geometry to emerge from the algorithm rather than being imposed from outside. In this case the geometry is based on the circle, which is then extended to cover more complex geometries such as the Voronoi (emergent tessellation). These are “illustrations of consensus” because the bit you can see (the figures below) is the emergent result of all the components of the system (turtles mostly) finally reaching some agreement about where to be. The phrase begs the question as to what the turtles are being asked to agree about, and what architectural idea might be involved. Generally the task is to distribute themselves with respect to two conflicting pressures – that of the group based on some higher order pattern, and that of the individual.

‘Other ways of drawing circles’

Papert points out that the equations

```
Xcirc = originX + R cos (angle)
Ycirc = originY + R sin (angle)
```

Do not capture any useful information about circles, whereas

```
To circle
      Repeat 36
            Forward 1
            Turn Left 10
            End repeat
End circle
```

requires only English and a familiarity with walking.

As Resnic points out (1995) with parallel computation we can propose another implementation of the circle using not just one turtle but many of them. The algorithm is based on the characterisation of a circle as being an array of points all at the same distance from another common point To do this with turtles we :

1. create a lot of turtles at random
2. get each turtle to turn towards the centre of the circle
3. get each turtle to measure the distance between itself and this centre point
4. if this distance is less than the desired radius then take a step back (too near)
5. if it greater then take a step forward (too far away)
6. Go on doing this for ever.

This procedure can be written in netlogo as so:

```
to attract
      ask turtles
      [
               set heading towardsxy 0 0
               ifelse ((distancexy 0 0 ) < radius)
                      [bk 1]
                      [fd 1]
        ]
end
```

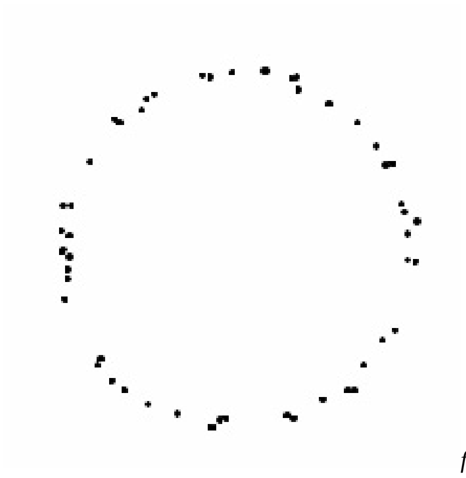


figure1 turtles form a circle

Notice that nowhere in the procedure is it given where the turtles are to walk to, they just walk back and forth. In fact the "circle" is only apparent to the human observer, and while we look at it, it shimmers into being rather than being constructed carefully.

The result is a ring of turtles defining a circle. In fact there is one more thing to do because just using this process will result in an uneven circle with gaps in because the turtles start off randomly and gather in random spacings around the circumference. How can we get the turtles to spread themselves out – the answer is to do roughly the same thing as in attract, but instead of using the global point 0 0 (the centre of netlogo's universe) we use the nearest turtle's position, and back away from that

```
to repel
     ask turtles
      [
       set closest-turtle min-one-of turtles with [self != myself] [distance
myself]
       set heading towards closest-turtle
       bk repel-strength
       ]
end
```

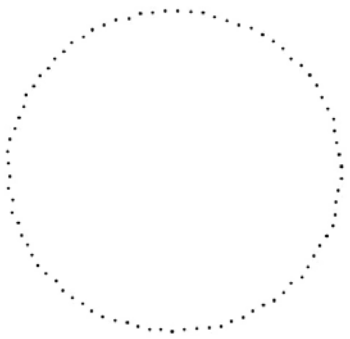


figure 2 turtles smooth out the circle

These two procedures; **repel** and **attract** form a useful test bed for experiments. Given the high level of abstraction we can begin to model more complex shapes and spatial organisations than individual geometric objects without having to do much extra coding.

Extending the model

A more complex outcome that we can achieve with only small modifications is the emergent voronoi diagram (dirichelet tessellations). Voronoi diagrams are conventionally calculated using computational geometry. But using turtles the only extra needed is just one more type of turtle "target turtles" and the others will organise themselves. There are othe appropriate simulations based on reaction diffusion processes and cellular automata which are also easy to model reported e.g. in (Adamatsky 2001)(Coates 2004) but for reasons of space have been omitted here.

Generating spatial organisation

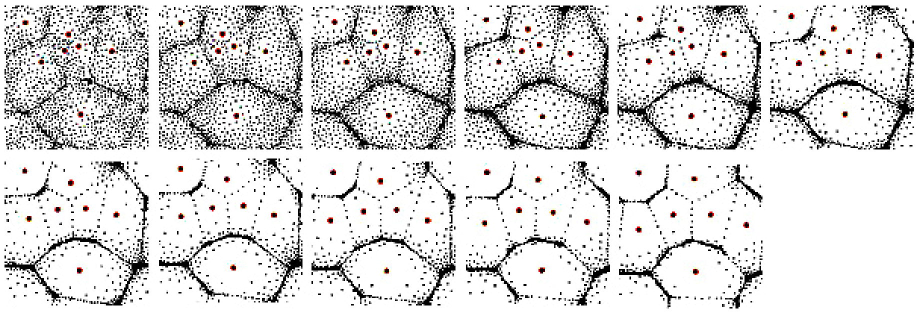


Fig3 The 11 generations of the emergent Voronoi (top left to bottom right). Note that both the voronoi generators (big spots) are self organising with respect to each other, as well as the population of small dots.

So we can make two kinds of turtles – normal ones and targets. Both the normal turtles and the target turtles obey the repel rule, but the attract rule only applies to normal turtles, who try to stay at a particular radius from the target turtles

```
to attract
   locals [targets]
       ask turtles
       [
          set targets turtles with [target = true]
          set closest-turtle min-one-of targets with [self != myself][distance
myself]

          set heading  towards closest-turtle
          ifelse ((distance closest-turtle ) < radius) [bk 1] [fd 1]
       ]
```

end

Some experiments with repel and attract

By now the model has developed into a method for exploring some useful relations in the development of spatial organisation, with the notion of varieties of functional needs and competing self organising pressures that construct emergent patterns. With simple tweaking of the rules new and useful design ideas can be introduced and experimented with.

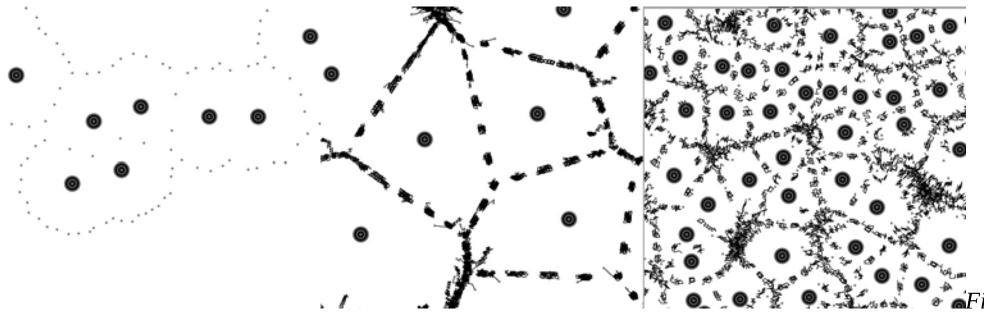


Figure 4 a b c emergent voronoi organisation using netlogo turtles

With only a tiny adjustment to the code (adding a “pitch” direction to the “heading”) the self organising tessellations could become properly 3d. Currently the data from the model is exported to an external modeller for post processing, but the VIPA implementation will be able to work natively in PYTHON and the post processing can be done seamlessly in BLENDER

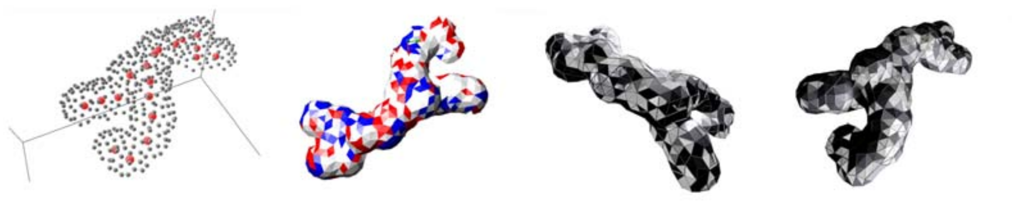


Fig 5 a b c emergent spheroids using the same attract repel algorithm in 3d (skinning done in AutoCAD vba via spreadsheet transfer)

Once the basic idea has been grasped it is easy to further elaborate the system with other varieties of turtles, so as to generate different outcomes, such as towers, rings and so on. In some of the illustrations the networks shown are in fact a turtle represented by a line, which also works in parallel with the spherical types, with its own simple rules.

In these experiments the target balls are controlled interactively so as to have a greater degree of control over the emergent morphology.

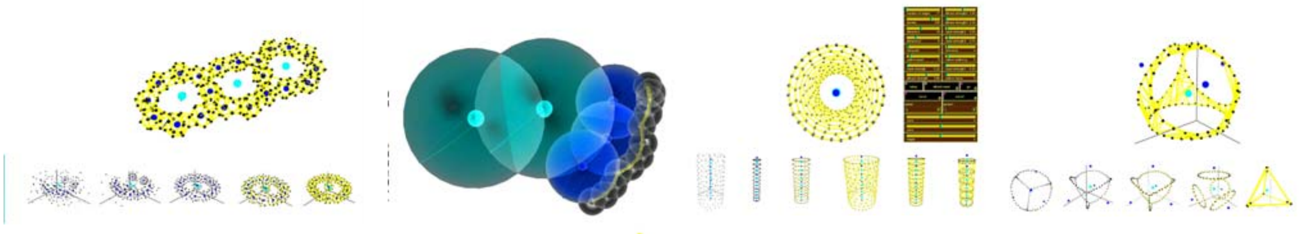


Fig 6 a b c three and four agent simulations using interactively manipulated targets to develop complex geometries

Conclusion

These examples attempt to illustrate how students can develop models of self organisation that are derived from parallel processes. The aim is to allow a dynamic perspective of the design problem to feed directly into a form / space making process. In this way we can exploit the computer in ways outlined by Papert & Kay as a new way of thinking about some domain, in this case 3d design. It is to be hoped that this will be seen as a valuable addition to the armoury of the virtual campus we are building with our VIPA partners.

References

Adamatzky, Andrew : 2001 *Computing in Nonlinear Media and Automata Collectives* Institute of physics Philadelphia

Coates & Thum 2000: Agent based modelling Greenwich 2000 University of Greenwich London 2000

Coates Thum & Derix 2001: *Dust Plates & Blob*s Generative Art Conference Milan 2001

Coates,P: 2004 Fractal decomposition in architecture 1st Fractarch congress Madrid infiniart Madrid 2004

McCarthy, J *:* 1979 *History of Lisp* Stanford U

Mitchell,W 1990: *The logic of architecture* MIT press

Kay,A & Goldberg A:2003 *Personal Dynamic Media in* Wardrip-Fruin, N & Montfort *The New Media Reader* MIT

Kay ,A 1993 : 1993 *History of Smalltalk* ACM SIGPLAN Notices Volume 28 No 3 March

Klee.P 1925 :pedagogical sketchbook

Resnic, M Turtles *termites and traffic jams*. MIT 1995

Papert , S: 1980 *Mindstorms , children computers and powerful ideas* Harvester Press UK