Digitising the complex form

Zane Egginton, Nikolay Popov, Brett Orams

Unitec, Auckland, New Zealand

Abstract: The demands of Landscape Architects and Architects use of organic and complex forms at various scales heavily fuel this project. In a typical project a designer is often faced with three challenges, how to create a digital model, how to enhance their design and how to visualise the final product. However due to the limitations of typical CAD programs used in the industry as a stand alone solution for digital representation the designer may end up representing complex form by illustrating either by photographic means or hand rendering. While these two methods have their place as representational tools it limits the techniques available to the designer to manipulate and interrogate these forms as they would with a typical CAD model. The difficulty to produce digital models of complex forms limits the methods to visualise the design and often mutes the detail presented. In this paper we will look at digitising and manipulating the complex form in various ways.

Introduction

Traditionally design has been represented as drawings, be it technical plans or rendered perspectives. When the form becomes too difficult to communicate or relate to scaled physical models are employed which allow the viewer to engage with the forms three dimensions while at the same time allowing an alternative way of interpreting it. This basic approach is still used although the tools the designer can employ appear vastly different to those of the past. Computers have introduced many new modelling and visualisation techniques, however the success of this is very much dependent on the designers technical skill and the form being created.

Organic and complex forms have always been popular in all form of design and are commonly inspired by natural processes. However in nature the form of an object is due to very complex interactions with many influencing factors. These forms can be created via three dimensional modelling techniques, programmed algorithms or a combination of both but as the form becomes more complex the representation and manipulation of it becomes correspondingly more complex. A simple box for example can be easily created and manipulated however vegetation, rocks, fabrics, or landscapes may prove to be more difficult. How then do we deal with three dimensional objects that are not easily represented by basic primitives and what opportunities does it open up to the design process.

Creating Complex Form.

There is a plethora of tools and techniques available to create 3D form digitally however these can be organised into three major categories;

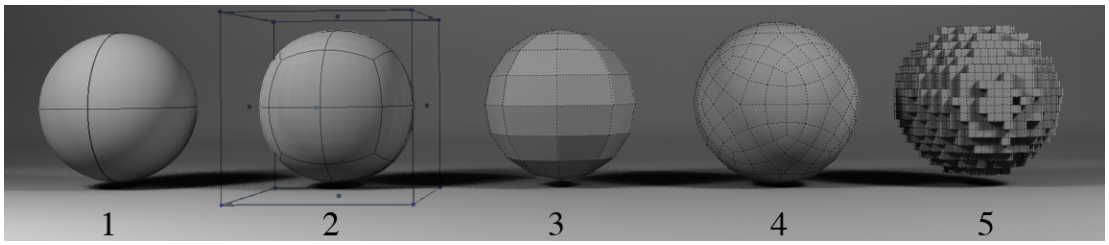
- Capture: In which the physical form is digitised.
- Model: Created within a software application by direct manipulation.
- Programme. Form derived via a set of variables and algorithms.

Any one or a combination of these could be used to create an object. For example I could 3D scan a marble sculpture modify it in a 3D modelling application then run it through a script that distorts its surface (see Fig. 1). With almost all digital work flows each of these processes are in some way reliant on at least one of the others. 3D scanners almost always rely on the user removing or manipulating the form, 3D modelling will in most cases employ procedural tools to enhance the form. However each of these three categories of creating form presents limitations and opportunities to the design process which will be described hereafter.

Fig. 1: Manipulated 3D scan of a small statue

Surfaces

To understand three dimensional digital representation we must first understand the types of surfaces and how they can be used. The surface type is the most significant element of a digital model as this is what defines the object in a numerical space. The renderable form and the interior of the solid is defined by it's surfaces and the shaders/textures attached to it. It also will dictate how the user will manipulate the form and what type of procedural modifiers can be used with it. The three main types of 3D objects that are currently available in 3D applications are; Voxels, NURBS, and Polygons. Subdivision and smoothed surfaces (seen in Fig. 2)



1. NURBS sphere, 2. Subdivision sphere (from a cube), 3. Polygon sphere, 4. Smoothed dodecahedron, 5. Voxel sphere (simulated)

Fig. 2: Types of surfaces.

Voxels

Voxels can be thought of as a three dimensional raster images. A raster image (like those taken with a digital camera) is made up of many pixels that form an image. Voxels are very similar except instead of two dimensional pixels Voxels are three dimensional, like a building blocks. The surface of a voxel is normally smoothed however the image in Fig. 2 shows what a voxel sphere would look like if it was not. The big advantage of voxel sculpting over other techniques is way a surface can be reshaped. if you pull on a polygon surface stretching of the faces occurs which will require more polygons to be created before further sclupting can take place. However pulling a voxel surface creates new voxels as it does therefore the experience is more tactile. Very few applications allow for direct Voxel sculpting, in fact all the big applications (Maya, 3D Max, Rhino, Houdini, etc) at the time of writing did not have Voxel sculpting tools, however some do use them internally (Houdini and Maya's fluid simulation uses Voxels). There are a handfull of voxel modelling applications, one being online and allows for very crude voxle scuplting http://kyucon.com/qblock. The simple models produced can be manipulated in realtime on the website and take on a very 80's computer art feel. A far more advanced tool is 3D Coat and has many tools to scuplt the surface. Compared to other scupting applications it feels more like work with clay, adding and removing volume from the surface. It has the ability to import geometry from polygon data coverted to voxels, scuplted and then exported back as a poygon surface and furthur manipulated/rendered in another application. 3D coat also has very good polygon editing, resurfacing and texturing tools that can be used for geometry created. Some game engines have used voxels for terrains instead of height maps (an image that defines z heights for a grid of polygons) as it allows for caves, overhangs, and arches.

NURBS

In the early 90's a new way of creating surfaces was appearing in 3D software called *Non-uniform rational basis splines* or "NURBS". The technique uses a system similar to Bézier curves (also known as B-Splines) originally developed in the 50's. These surfaces are defined by control points that influence surface rather than a collection fixed points connected in space (polygon surface). These control points can be on the surface of the object or as a weighted force, like an adjustable rubber band pulling on the surface. As a NURBS surface is mathematically generated it theoretically has an infinite resolution resulting in very smooth surfaces (dependent on the software and it's settings). As the surface is interpreted by the software the level of detail (LoD) can be adjusted, whereas a traditional polygon surface can not. NURBS for a long time where all the rage especially for smooth surfaces and most modern 3D applications support them however they do have limitations; they're difficult to texture, inflexible (esp when being used in different programs) and are difficult to use when modelling complex objects.

Polygon modelling

The traditional and most common form of 3D modelling is to use a collection of polygons each of which are defined in a 3D space. A polygon surface is made up vertices (a point in space) connected by edges to form a network of faces (typically triangular or quadrangular faces). These surfaces are simple to process (compared to NURBS), very portable, easy to work with and may also store additional information about the object for example the 'normal' direction of each face and a 'UV map' to permit the fixing of textures to the surface.

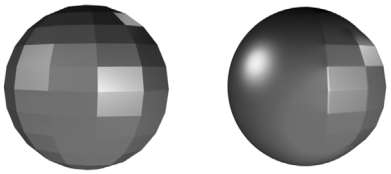


Fig. 3: Maya's Sub division proxy. The polygon sphere on the left controls the shape of the subdivided sphere on the right.

In recent years polygon modelling has seen a resurgence with simple modelling tools like the popular (and free) Google SketchUp. However high-end modelling has also seen a similar gain in popularity with the introduction of subdivision surfaces and other surface modifiers (eg Maya's smooth polygon feature). One incredibly powerful feature in Maya is it's subdivision proxy mesh tool (Fig. 3). This gives you the ability to model using simple polygon tools which in real time creates a separate more complex subdivision surface. This gives you the ability to define both the general shape plus control the crease weighting of the edges. The Image seen in Fig. 4 was modelled in Maya from a deformed polygon cube that was then cut up into many faces. Each of the edges where used to dictate the final geometry while the faces were removed to produce holes. The resulting mesh was smoothed and further manipulated to clean up the surface then rendered into a HDR scene (high dynamic range images contain more information about the light than a standard raster image therefore producing realistic lighting on the rendered model).

Fig. 4: Complex cube created in Maya and rendered into a HDR scene.

3D scanning

An obvious way to digitise a form is to capture the object itself. This technology has been around for a long time however the ability to create accurate models has only recently been accessible to the general public. The 3D model seen in Fig. 1 was a scan of a small (approx 40cm high) marble statue and took about 20 hours to capture and process using a desktop 3D scanner from NextEngine. The NextEngine scanner uses a techniques that fires a laser at the object while taking images of where the laser hits the surface resulting in a accurate surface to approximately 0.15 mm accuracy. Although 3D scanning seems like a simple process, in practice it seldom is. May issues may arise and are dependent on the technology and the object being scanned. In this case the surface (marble) can absorb and scatter the laser light resulting in a rough finish however the use of chalk dust can reduce this. Multiple scans from many different angles are required to capture the entire surface especially as some areas of the model are difficult to get the laser light into while also 'seeing' it from the offset camera resulting in holes or inaccurate surfaces.

Fig. 5: 3D Scan of a Lyonel Grant Bronze Casting.

Parametric/Procedural modelling

Parametric and procedural modelling in CAD programmes makes use of 'intelligent' objects and tools that can create an object (and sometime manipulate other relevant objects) by use of parameters, programmed algorithms and linked geometry. For example In many applications it's possible to create a wall by defining variables such as height, length, width, material, structure, etc. It is then possible to place a window object into this wall in a similar fashion however the wall will then need a hole cut in it, this is where the 'intelligence' comes in, almost magically and without user interaction a hole appears in the wall to accommodate the window. From here we can create timber framing, foundations, wall accessories (eg architraves, skirting, sills, etc), and produce a bill of materials, all without defining any geometry. Although this is a rather simple example there are far more complex tools, and this is an area in the CAD world that is vastly expanding.

In 3D modelling applications parametric/procedural objects and tools are both common and indispensable, however they do differ. At the most simple level we can create primitives, spheres, cubes, pyramids, etc without the need to define a complex set of polygons. At the other end of the scale we can simulate physical dynamics; gravity, wind, chaos, force fields, materiality, as well as scripted procedural manipulations which allow for the creation of complex systems and ultimately form.

Procedural polygon modelling of Haeckel Radiolaria in 3D Studio Max

Ernst Haeckel was an eminent 19th century German biologist who, among other things, coined the term 'Ecology' and is the first person to have been known to use the phrase 'First World War'. He also prolifically produced highly detailed drawings of life forms and microscopic organisms. His most famous work, 'Kunstformen der Natur' (Art Forms in Nature) was published in 1904. Using selected renderings from plates in 'Kunstformen' as both inspiration and reference Brett Orams has developed various techniques of procedural polygon modelling in Autodesk's '3D Studio Max' software to create three dimensional representations of a number of the microscopic marine organisms known as Radiolaria and Diatomea (Fig. 6 & Fig. 7).

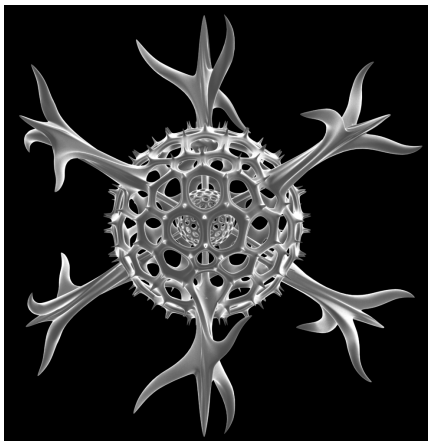


Fig. 6: Radiolarian 3D model. Hexancistra quadricuspis

The modelling typically begins with a 'primitive' object - a geosphere for instance - and evolves the model, increasing detail and complexity with 'modifiers' like 'edit poly', 'displace', 'shell' and 'mesh smooth'. These procedural modifiers consequently provide a 'build history' of the model which can be sequentially turned on or off to illustrate the stages of construction of the final virtual object. Much time is spent manipulating the geometry, texture and bump mapping, lighting, and render settings to achieve the ideal emulation and 3D enhancement of von Haeckel's detailed 19th century 2D drawings.

Complex 3D models can be made by using 2D images of real-world 3D objects of any scale as a starting point, and then developing techniques and using imagination to visualise what the original 3D object looked like from any angle. The translation back to a physical form from the virtual file has recently been made possible with the advent of 3D printers.

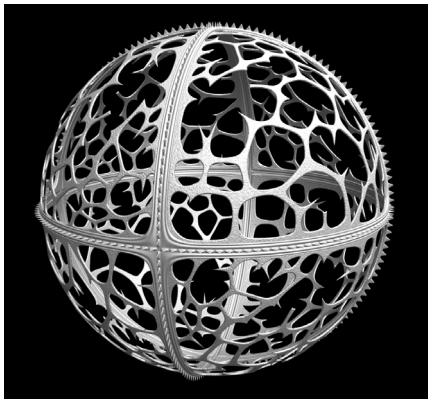


Fig. 7: Radiolarian Trissocyclus sphaeridium

Modelling with Dynamics

Creating form via dynamics is truly inspiring. Most 3D applications have some form of dynamics, this is where real world physics are simulated within the software rather that manual keyframe animation. This could

be gravity (eg an object falling down stairs), cloth simulation, fluid simulation, particle dynamics (eg smoke, fire, clouds, mist, etc). Most dynamic simulations will animate not only the position of a given object but will manipulate the object itself in response. For example with Maya's nCloth applied to a sphere we can define the objects, weight, internal pressure, stretch resistance, rigidity, and more. Many of these variables can be set on a per vertex basis, ie the base of the sphere could be more rigid than the top.

Modelling with Dynamic Particles.

This is where things get very interesting. Particles have many configurable attributes and characteristics, for example particles can be effected by wind and gravity, they can also be attracted or repelled from each other or another object. These attributes allow us to set up many different and dynamic situations however these particles are only points in space and are meaningless untill we attach geometry. This can be done in two ways; by replacing the point with a copy (instance) of an object or creating a mesh that is directly influenced by the point cloud (particles).

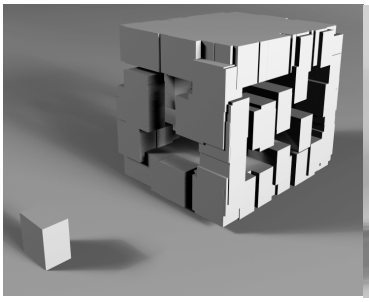


Fig. 8: Particle driven form on a cube

Fig. 9: Same model with rotated instances

By representing each particle with a copy of some geometry we can effectively create form that complies to the same rules as the particles. This is very useful if we what our particles to be seen as an instance of geometry. In Fig. 8 we see a simple model of where particles represented by an instance of geometry (seen to the bottom left of the frame) that have been dropped on and stuck to a cube. Each instance of geometry however can have many of it's characteristics (eg position, scale, rotation, etc) mapped to a variable generated either directly from the particle engine or a scripted variable created by the user. In the above simple example all of the characteristics of the instanced geometry are identical with the exception of position which is being derived by the position of the particle. However lets now lets effect the rotation of the instance by its position in the world. We now see in Fig. 9 that as the pieces fall towards the ground plane they rotate around our object.

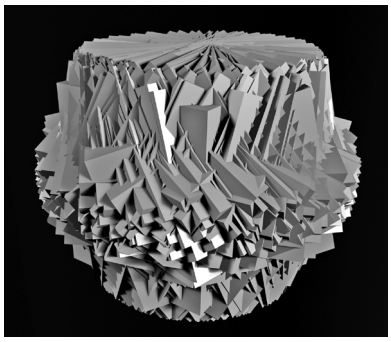


Fig. 10: Particle driven form

Things get more interesting when we use the particles as a driver for our instances rather than a direct translation. Instead of the position of the instanced cubes being derived from the particle how we now set it to fixed location and then drive the rotation by the age and position of the particles. We now have a very different model (Fig. 10). The model created is becoming more complex and the outcome each time is less predictable, in fact each time the simulation is run we get a different form.

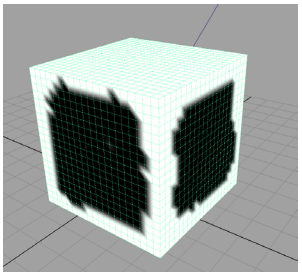


Fig. 11: Field magnitude weighting on a cube

As mentioned earlier another way to generate form from particles is to create a polygon mesh from the points themselfs. There is many variables and thresholds that can control how the mesh is created and to what detail. As an experiment I've taken a cube and divided the faces into a 20 x 20 grid. The vertices (intersecting points on the face) have then had weightings applied for stickiness, friction and a field magnitude that will effect the general settings of the cube. The most significant attribute in this case is the *field magnitude* which has a general setting of -25 making it very attractive to particles, however each vertex is set with a low weighting value of -.5 and a high value of 1. It is then possible paint on values where black = 12.5, which will repel particles, and white = -25, which will attract particles (see Fig. 11).

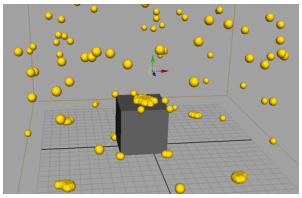


Fig. 12: Particle Simulation being run in Maya

We now place a particle emitter above the scene that for the first 100 frames creates particles at a rate of 50/sec for a total of 200 particles. These particles also have some key attributes, the major ones being mass, self collisions (so that they bounce of each other), stickiness, and a self repelling factor. This sets up a behaviour that causes them to avoid each other if possible however they will stick to each other if they do manage to touch. In Fig. 12 we can see the particle simulation being run inside of Maya. This simulation proved to be very interesting as can be seen in Fig. 13. The particles begin to attach to the edges at the start however as they begin to compete for space they occupy regions of cube that they should be repled from, the particles in these areas begin to move very rapidly until for some unknown reason they explode from the faces and the form reaches a stable state.

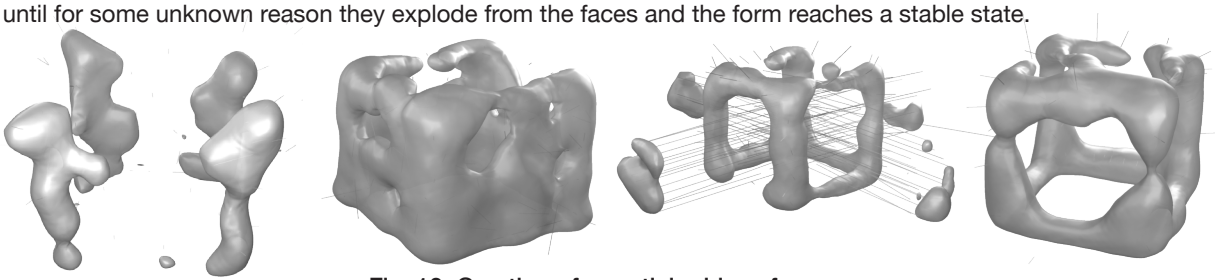


Fig. 13: Creation of a particle driven form.

Cloth Dynamics

Another form of dynamic simulation is that of cloth. In Maya this is essentially very similar to a particle simulation however the particles are arranged along the vertices of a polygon mesh and have the ability to deform the attached mesh. Again the cloth like the particle simulations have many configurative attributes however there are the added attributes that control the connection between the vertices and the object as a whole; stretch, bend, compressibility, bend resistance, internal pressure, etc The cloth will also react to external forces like wind and gravity and has the ability to be pined to geometry, and can be set up to tear when a threshold is exceeded.

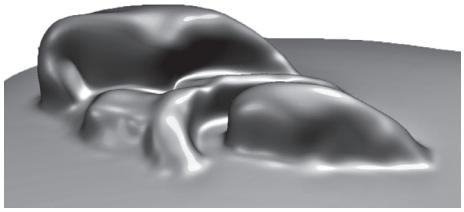


Fig. 14: nCloth with large self collision setting

To test cloth dynamics a simple scene containing some `three cubic forms on a plane with a cloth above that is effected by gravity which also has a large self collsion set to increase the distance of the folds (this can be seen in Fig. 14). After the cloth settles a wind is applied and the cloth moves according to the constraints configured.

In a second test the same basic configuration exists however a far more complicated mesh is dropped, this time the cloth acts in a similar way but has a tendency to get caught on the corners of the cubes. It also has less tendency to bend on the edges as can bee seen in the top left of Fig. 15.

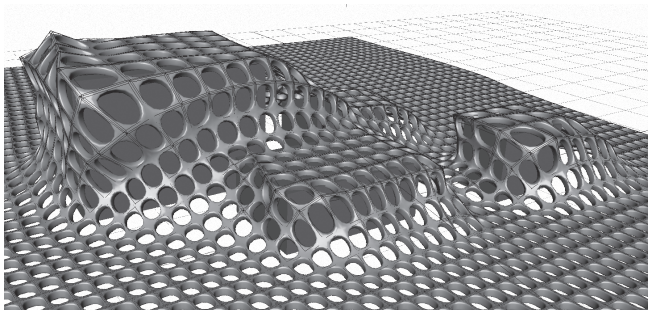


Fig. 15: Complex nCloth in Maya

Gestalt Space and Form

Complexity theory provides us with new ways of seeing and knowing space and form. Temporal and spatial organizations can be modelled, utilising simple algorithms, as 'self-organizing morphologies' defined as part of a process. This approach redefines architecture and landscape architecture as an emergent outcome of dynamic relations of simple elements bounded together by multiple feedback loops (Paul Coates, 2010). The fundamental idea of this chapter is that the initial design representation of a complex form or spatial organization is best done by using algorithms as texts written in language rather than some graphics. As Coates(2010) explains:

> Fundamentally the proposition is that Chomsky's dictum – that finite syntax and lexicon can nevertheless generate an infinite number of useful (well-formed) structures – can be applied to artificial languages, and that texts can be written in those languages to generate architectural objects, taken to mean well-formed configurations of space and form. This is the generative algorithm and the idea is that a generative algorithm is a description of the object as much as the measurement and analysis of the object, the illustration of the object and the fact of its embodiment in the world.

The examples in this chapter are very simple and the aim is to illustrate the principle rather than attempting to generate complete architectural spatial organizations. They are proof of concept. They are written in artificial or computer language, called Netlogo (Wilensky, 1999). Netlogo itself is embodied in two of the most common computational frameworks for designing such models. They are known as Cellular Automata (CA) and Agent Based Modelling (ABM).

Cellular Automata

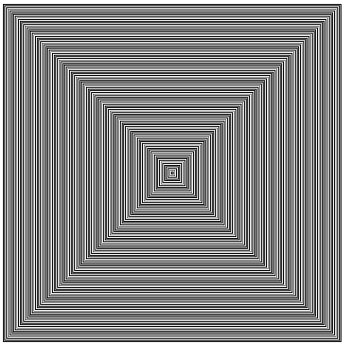
Cellular automata are discrete and dynamic computational systems. Space in CA is tessellated in a regular lattice. Each point of this lattice is called cell that can have finite number of states. Time is also discrete in CA and proceeds in iterative steps, i.e. t, t+1, t+2, t+4, etc. The states of the cells are updated in a parallel manner according to a local rule, commonly concerning just the cell in question and the neighbouring cells, at every time step, i.e. the state of a cell at time (t), depends on the state of the cell and the states of its neighbours at time (t-1). All of the cells are updated synchronously and the state of the entire lattice advances in discrete time steps. Cellular automata have been used as a simulation technique in the study of an impressively wide range of urban phenomena, including regional growth, urban sprawl, gentrification, residential growth, population dynamics, economic activity and employment, historical urbanization, land use evolution, and polycentricity to name but a few. The architectural interpretations of CA are studied by Devetacovic, et al. (2009), Coates, et al. (1996), Krawczyk(2002) and many others.

The first example is a version of Game of Life and is probably the best-known example of cellular automaton. It was devised by the British mathematician John Conway in 1970. In Game of Life the world is a two dimensional grid of cells. Every cell has one of two possible states – alive (white) or black (death). At every time step, each cell interacts with its eight neighbours - the cells vertically, horizontally, or diagonally adjacent to it. Described in English, the rules of interaction are:

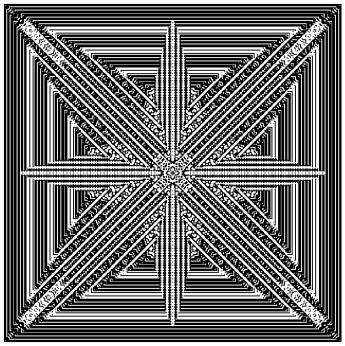
- If I am black (death) and have at least one neighbour that is white (alive), I become white (alive).
- If I am white (alive) and have more than desired number (threshold variable that can be adjusted by the person who is running the model) of white (alive) neighbours, I turn black (death).

In Netlogo the cells are called ‘patches’, cell’s colour is called ‘pcolor’, and the algorithm expressed as text in artificial language, i.e. Netlogo looks like that:

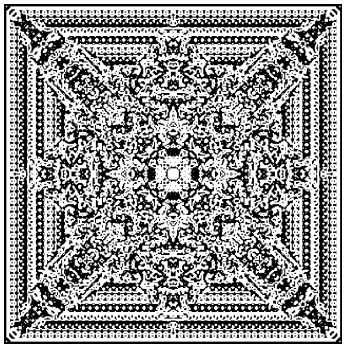
```
ask patches
[ifelse pcolor = black
[if count neighbors with [ pcolor = white] > 0 [set pcolor white]]
[if count neighbors with [ pcolor = white] > threshold [set pcolor black]]]
end
```



Threshold = 2, Iterations = 200



Threshold = 3, Iterations = 200



Threshold = 6, Iterations = 200

Fig. 16: 2D Game of life.

At first, what could be expected by such an algorithm is randomly distributed array of black and white cells. The truth is that the outcomes are rather symmetrical patterns recognizable from a synoptic viewpoint. (see Fig. 16) The point to note is that the pattern is visible only to the observer, the person who is running the model, the cells are not ‘aware’ of it; they just ‘know’ themselves and their eight neighbours. This is probably the most canonical example of emergence. The overall description of the pattern is NOT in the algorithm. Without any additional programming the Game of Life can be played in 3D. The only difference is that every cell has 26 neighbours (Fig. 17)

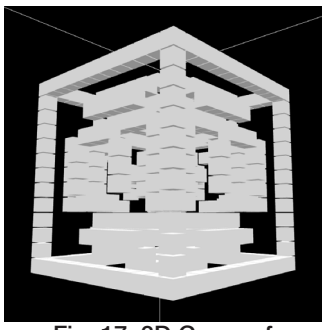


Fig. 17: 3D Game of life Iterations = 8

The second example is a cellular automaton that generates Voronoi tessellations of space i.e., minimum energy pathways between a set of points. Voronoi patterns occur spontaneously (bottom up) in nature at variety of scales. They resemble biological cells, forest canopies, territories of animals, fur and shell patterns, crystal growth and grain growth, cracks in dried mud, etc. The traditional, top-down analytical method to draw such tessellations looks rather clumsy and doesn’t seem to capture the underlying dynamic of the phenomenon. It is explained by Aranda and Lasch (2006):

1. Take a set of points.
2. Construct a bisector between one point and all the others.
3. The Voronoi cell is bounded by the intersection of these bisectors.
4. Repeat for each point in the set.

Fig. 18: 2D Voronoi Tessellation

Much more ‘natural’ is the algorithm using the parallel computation of CA. The world again is regular, two dimensional grid of cells (patches) overlaid with set of points. The algorithm is:

- Take set of points and colour them differently.
- Every cell colours itself to the colour of the nearest point.

Once again the Voronoi pattern is not in the algorithm, it emerges just by following the local rules described above. In Netlogo language the algorithm is described in the following way:

```
to recolor
ask patches
[set pcolor [color] of min-one-of points [distance myself]]
end
```

The Fig. 18 and Fig. 19 show variations of Voronoi tessellations in 2D. CA models outputs are symbolic or topological descriptions of space, i.e. networks, grids, etc and should not be interpreted as metric partitions of space. The authors mentioned at the beginning of this section experimented

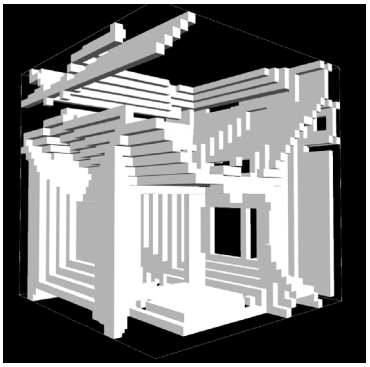


Fig. 19: Boundaries of a 3D Voronoi model

with translating the outputs of CA models into geometric spatial organizations but the format of this chapter doesn't allow us to elaborate on that. The interested reader can refer to the articles listed in the references.

Agent Based Modelling (ABM)

This type of modelling describes time and space using agents – autonomous little computers. In Netlogo agents are called turtles. Every agent has a strategy, usually defined in parametric space, and is aware of part of its surroundings. All of the agents try to implement their strategies synchronously and can alter their positions in metric space at discrete time steps. The actions of the agents are undertaken in Euclidian space and therefore distances, angles, and other metrics are significant. Agent Based Models are much more flexible, mobile and dynamic than the restricted topological tessellations of Cellular Automata. ABM are extensively studied in ecology, social sciences, and urban planning. Attempts to translate them in architectural context have been made by Coates(2010), Mottram (2005), Turner(2005) to name just a few.

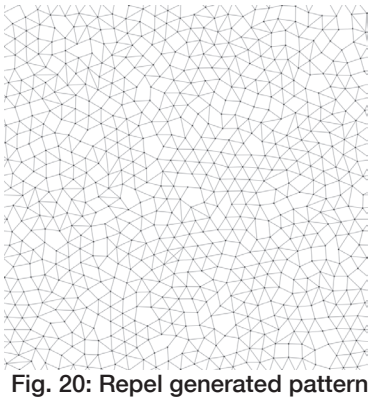


Fig. 20: Repel generated pattern

The first two examples create simple geometric shapes like triangles and circles without any knowledge of geometry and exemplify once again the concept of emergence using parallel computation. We will begin with the triangle example. The algorithmic text (in English) is:

- Create a set of points at random.
- Each point faces the closest point to itself.
- Each point takes a step backwards.
- Repeat this many times.

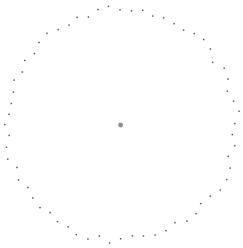


Fig. 21: Attract generated pattern

Opposite to what might be expected from such algorithm after number of iterations the points settle down on a triangular lattice. This happens because when ever they move away from this grid they are in unstable situation and will fall back into the triangular pattern. (see Fig. 20) This model provides nice example of how an algorithm itself can have an epistemic independence of the structural output of the model. In Netlogo the text is expressed like that:

```
to repel
ask turtles
[ set closest-turtle min-one-of other turtles [distance myself]
 face closest-turtle
back 1]
end
```

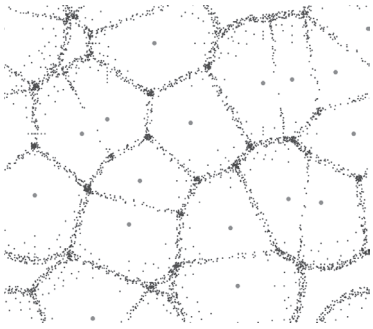


Fig. 22: Attract & repel generated pattern

The second example is how a circle can emerge from bottom up interactions without any top-down geometry. The idea is to do exactly the opposite from the first example and was proposed by Resnick (1994), i.e. `

- Create one point, called 'Centre'
- Create set of points at random.
- Each point faces the centre
- If it is too close make a step backwards, if it is too far take a step back.
- Repeat 2 - 4

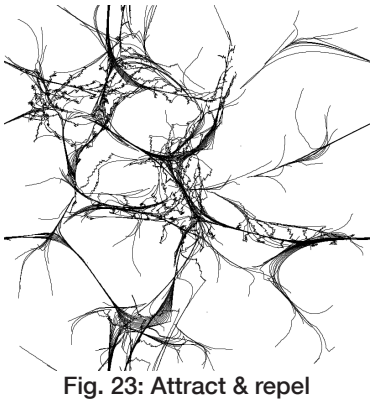


Fig. 23: Attract & repel to a moving object

The result is an uneven circle made of points that shimmers into being rather than being constructed deliberately. (seeFig. 21)

The Netlogo code is:

```
to attract
ask turtles
[ face centre
 ifelse distance centre < radius
[bk 1]
[fd 1]]
end
```

If both attract and repel algorithms are run at the same time the points will spread themselves along the circumference of the circle. To achieve more complex results we can make two types of agents – attractors and points. Attractors can be distributed randomly or can form some pattern. The points are attracted to the closest attractor and repelled from each other. The result is again emergent – Voronoi diagram, as described in Cellular Automata section. (see Fig. 22) Different emergent pattern can be constructed with the same algorithms if the attractors start moving. The

results will be branching structures that can resemble hydrological patterns, animal and pedestrian trails, etc. (see Fig. 23)

This chapter illustrates how complex spatial and temporal patterns can be modelled as dynamic collective aggregations of simple units bounded together by multiple feedback loops. Parallel computing allows the algorithm to unfold in unpredictable ways and to provide us with emergent spatial patterns that can be observed synoptically. Most importantly the global pattern is gestalt constructed by the reader and is not inherent in the model. Coates (2010) stresses that:

> "So if the output is larger than the rule, then one is justified in claiming that the whole system is complex rather than complicated".

In order models (like those described above) to be of any use, it is crucial not to over specify the problem, this is to say, the algorithm should not contain the answer to the problem. The outcome should emerge from interactions of simple more general rules. The art of these models is parsimony. Simulations should have 'epistemic autonomy' in order to generate 'structural autonomy'(Pask, 1962). The algorithm describes the form as a process, usually a simple process. It resembles to great extent the relationship between genotype and phenotype – the DNA of an organism doesn't know what the organism will look like. "The distributed representation means that the pattern is not coded anywhere but everywhere"(Paul Coates, 2010).

Conclusion

This paper has outlined many different approaches to digitise complex form and even though many of the models have been relatively simple they could easily be developed further to interrogate a design and perhaps even drive a design. As a process that creates complex form can not have a predicted outcome the creation within a digital work flow must either take an algorithmic approach or replicate one that has been created within natural processes. As the use of generic CAD tools seldom allow for this we must take an alternative approach to creating form outside of these applications in new and innovative ways. Evolutionary design is not just a computer search in some type of parametric space, but it creates new space that accommodates the design.

References

Arand, B., & Lasch, c. (2006). Pamphlet Architecture 27: Tooling. New York: Princeton Architectural Press.

Coates, P. (2010). Programming.Architecture. London: Routledge.

Coates, P., Healy, N., Lamb, C., & Voon, W. L. (1996). The use of Cellular Automat to explore bottom up architectonic rules. London, UK: Imperial College London. Retrieved from http://uelceca.net/research/other/eurographics1996.pdf

Devetacovic, M., Petrusevsky, L., Dabic, M., & Mitrovic, B. (2009). Les Folies Cellulaires - An Exploration in Architectural Design Using Cellular automata. Milan, Italy. Retrieved from http://www.generativeart.com/

Haekel, E. (1914). 1st World War, Indianapolis Times.

Haeckel E. (1904). Kunstformen der Natur

Krawczyk, R. J. (2002). Architectural Interpretation of Cellular Automata,. Milan, Italy. Retrieved from http://www.iit.edu/~krawczyk/rjkga02.pdf

Mottram, C. P. (2005). Evolving Street Plans from Shops and Shoppers. Milan, Italy

Pask, G. (1962). A proposed evolutionary model. In H. v. Foester & G. W. Zapf (Eds.), Principles of selforganization. New York: Pergamon Press.

Resnick, M. (1994). Turtles, Termites, and Traffic Jams: Explorations

in Massively Parallel Microworlds.: MIT Press.

Turner, A. (2005). An Ecomorphic Theatre as a Case Study for Embodied Design.

Wilensky, U. (1999). Netlogo. Evanston, IL, USA: Center for Connected Learning and Computer-Based Modelling, Northwestern University.