A SIMULATION TOOLBOX FOR SELF-ORGANISATION IN ARCHITECTURAL DESIGN

Fabian Scheurer

scheurer@hbt.arch.ethz.ch

Chair for Computer Aided Architectural Design (CAAD)
Federal Institute of Technology (ETH) Zurich, Switzerland

ABSTRACT

This paper describes an ongoing research project on the use of dynamically self-organizing structures in design, mainly influenced by the increasing number of irregular spatial constructions in current architecture. The focus of this work however, is not on the generation of form, but on the organisation of constructive components within a given form and their optimisation according to functional and structural requirements. To explain the approach a real-world example is presented whose constructive design has been finished in the year 2003 with the help of a self-organising process and which is about to be realized in the Netherlands. The very encouraging outcomes of this test case led to further examinations of the principles and finally to the implementation of a programming toolbox that enables rapid development of dynamic simulation software. The concepts of this toolbox are described and exemplified through the presentation of its first application, the generation of an adaptive quad-mesh that defines the structure for a trade-fair pavilion.

Keywords: dynamic simulation, emergent design, interactive tool

INTRODUCTION

There seems to be an increasing number of irregular spatial constructions in current architecture, which are readily explained by the help of metaphors such as “a forest of columns”, “a crystalline façade”, “a bird's nest” or “bubbles in a foam bath”. Though those images – all of them quoted from descriptions of recently built or planned architecture (ARUP 2004; Balmond et al. 2002; Bull 2004) – give a powerful picture of the designers’ intentions, they leave a long and thorny way to formally defining, detailing and finally building the structure.

The proposition of my research is to take those images literally and look at the processes that lead to the referred phenomena in nature: self-organizing adaptive systems and their ability to generate complex configurations without central control. If it were possible to turn the design process “downside up” those bottom-up methods could be used to solve the complex problems arising when trying to define the “formal” configuration of what Balmond calls “informal” design. The examinations are based on the wide range of research in the field of emergence over the past years, which give surprising insights on how nature is able to generate highly efficient structures, be it the growth of crystals, the organization of cells or the formation of swarms (Holland 1998; Johnson 2002; Kennedy et al. 2001).

Since there is no way (yet) to grow real buildings, experimenting with such concepts is only possible by means of computer simulation. The metaphoric idea of the designer has to be translated into the computer model of a complex adaptive system that simulates interacting agents in a virtual environment. The simulation is not meant to be physically correct in terms of structural analysis, but it should behave as if the components were physical implementations of the metaphor, making it intuitively understandable for the user. The complexity of this model is deliberately much lower than that of the intended construction, allowing to present the results of the simulation to the user in real time, enabling him or her to directly interact with the emerging structures by adjusting parameters or influencing the position of single components. However it is possible to implement simple "rules of thumb" to ensure structural and functional integrity up to a certain degree. The simulation is used as a tool to explore the vast space of possible solutions spanned by the rules of the model.

The approach was tested successfully on a real-life project, the "Groningen Twister" (Scheurer 2003), which is described in the next section to clarify the basic principles. The knowledge gained from this test case led to the development of a software toolbox, which allows rapid development of further software simulations for other projects. The concepts of this toolbox will be explained in detail by example of a second real-life application, a trade fair pavilion.

A SWARM OF COLUMNS: THE GRONINGEN TWISTER

In the year 2003 the office of Kees Christiaanse Architects & Planners (KCAP) together with Ove Arup & Partners in Amsterdam and the chair for CAAD at the ETH experimented with self-organizing structures for the Groningen Stadsbalkon to create "a forest of columns", some 150 slender pillars with differing diameters, randomly dispersed and slightly inclined in different directions that hold up a large concrete slab. The structural implications of this design had been roughly calculated and approved by the engineers, but the mind-twisting task was to define the exact location, diameter and inclination of every single column so that it would not obstruct the predefined walking or cycling paths, the distance to neighbouring columns was compatible with the spanning and cantilevering capacity of the slab and its diameter was sufficient for the portion of the slab it was holding up. Consequently every local change in one column would propagate through the whole structure because it affected the parameters for the placement of its neighbours. And, of course, the total number of columns was not predefined but should be minimised to save costs. Since the placement was only controlled by local rules, it seemed to suggest itself to use the power of self organisation and leave the optimisation to the columns.

Agent based simulation

To simulate a process of self-organization, the above definition was translated into a computer simulation model where the columns resemble *agents* that are "living" in a *habitat*. They can adapt to their environment by moving, tilting and changing their diameter (which will increase or decrease the bearing capacity) within certain limits.

If an agent reaches the maximum strength and still can’t cope with the load, it may split into two agents of the smallest size, thus creating a new column in the habitat. If an agent reaches the minimum size it may eventually die, removing itself from the habitat (Fig. 2). The computer simulation was based on particle dynamics, a simple but powerful way of simulating the behaviour of objects that have a mass but no spatial extent (Witkin 1997). By connecting particles with damped springs, complex non-rigid structures can be easily simulated, as has been used for example in real time structural analysis (Martini 2001). The structure of the simulation model is shown in Fig. 1: The top and bottom ends of a column are particles connected by a spring, which tries to align them vertically (this spring resembles the actual column rendered in the 3D view). A circular area around the top marks the bearing capacity of the column, which is dependent on the column diameter. The tops are repelling each other by force fields which try to keep the distance so that the circles of neighbouring columns just touch. The edges of the slab and the expansion joints are linear repellers, pushing the column heads away, the centrelines of the paths do the same for the column feet. The openings in the slab are defined as repeller points.

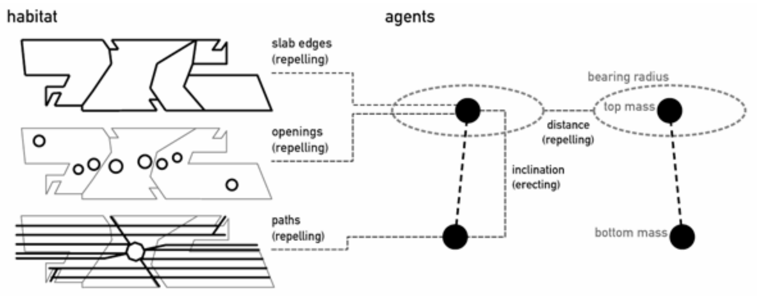


Fig. 1: Relations between habitat and agents in the Groningen Twister

The growing, shrinking, dying and splitting of an agent is triggered when the pressure it experiences exceeds or falls short of defined threshold values. The pressure is simply accounted by adding up the pushing forces applied to the top end of the agent by neighbours and repellers.

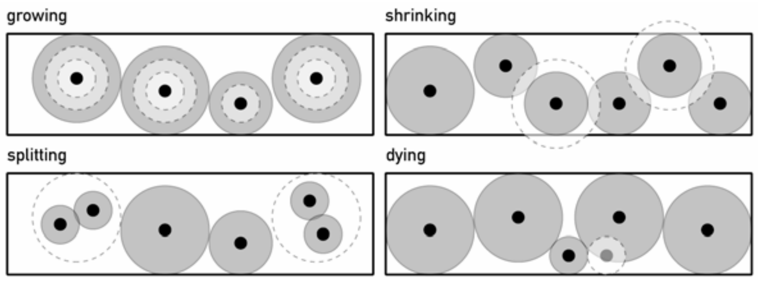
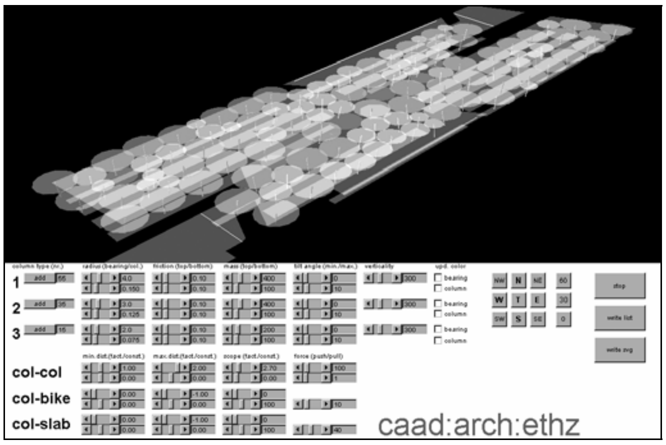


Fig. 2a-d: Growth and decline of the column agents within the habitat

The simulation was programmed in Java with the Java 3D API to visualize the results in three dimensions. The model can be influenced interactively by changing the various parameters. It is also possible to pick single columns and drag them to desired locations while the simulation is running. When the model reaches a stable state, the results can be exported in various formats, ranging from a list of the column coordinates to a VRML-model.



	column type (nr.)	radius (bearing/col.)	friction (top/bottom)	mass (top/bottom)	tilt angle (min./max.)	verticality	upd. color
1	add 66	4.0 / 0.150	0.10 / 0.10	400 / 100	0 / 10	300	bearing / column
2	add 35	3.0 / 0.125	0.10 / 0.10	400 / 100	0 / 10	300	bearing / column
3	add 15	2.0 / 0.075	0.10 / 0.10	200 / 100	0 / 10	300	bearing / column

	min. dist.(fact./const.)	max. dist.(fact./const.)	scope (fact./const.)	force (push/pull)
col-col	1.00 / 0.00	2.00 / 0.00	2.70 / 0.00	100 / 1
col-bike	0.00 / 0.00	-1.00 / 0.00	0 / 100	10
col-slab	0.00 / 0.00	-1.00 / 0.00	0 / 100	40

Fig. 3: Screenshot of the running simulation

The simulation model proved to be very successful and enabled the architects to develop a number of possible solutions that were complying with the rules in very short time. But programming the software took a considerable effort and due to the short timeframe many additional ideas that were developed could not be realized.

ZORA: A TOOLBOX FOR SELF-ORGANISATION

The experience gained with the Groningen project led to the development of a software tool-box that helps building similar simulation models at much lower effort. The intention was to have a basic set of components which can be quickly put together to form complex models without having to care too much about property editors, 3D rendering, export and synchronization issues. This tool-box – named ZORA – is a library of Java classes that enable rapid development of customized simulation models. Functionality for easy GUI building, real-time 3D-rendering and XML-import/export are included in the basic object classes. This section will explain the system architecture. The next section displays the first application of the toolbox.

Agents: The Simulation Model

The ZORA simulation model is a hierarchical tree of interacting components. The nodes in the model tree can be directly derived from the following set of basic component classes:

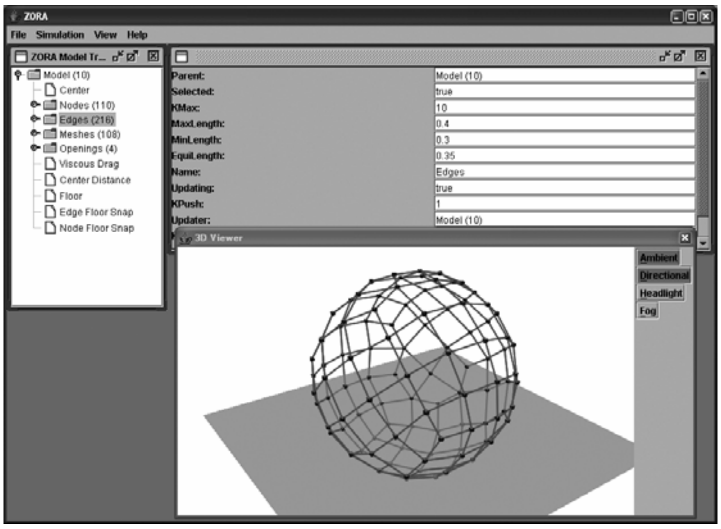
- The **model** is a special group component at the root of the model tree. It is controlling the time of the whole simulation model by sending update events to all the components at regular intervals. When receiving an update event, every component updates its state according to the given time step.
- **Particles** are geometric objects defined by their location and mass without physical extent. They can be accelerated by forces and will subsequently change their location according to Newton’s laws upon receiving an update event.
- **Rigid bodies** are geometric objects which have – in addition to particles – a physical extent and thus an orientation in space. Up to now, two types of rigid bodies have been implemented: the plane and the block (cuboid) type.

- **Local Relations** are connecting two components and influence them upon receiving an update event. Two basic binary relation types are implemented up to now: springs and magnets. Spring relations connect two particles or rigid bodies and apply a force to both of them, directly proportional to the extension or compression from the idle length. Magnetic relations do the same, but the applied forces are inversely proportional to the square of the components' distance.
- **Global relations** are connecting only one component with a global property, for instance the gravity or a viscous drag that slows down fast moving particles.
- **Groups** are themselves components and can be used to organize components hierarchically. Members can be added to and removed from groups at runtime, enabling dynamically growing structures. If a group is part of a relation it can apply the relation to all its group members, thus making it easier to define complex relations. For example the viscous drag can be applied to a particle group and influences all the group members.

A small but efficient physics engine, which simulates the behaviour of particles and rigid bodies very much like the game engines used in arcade games to show realistic physical behaviour of 3D objects, is used to evaluate the model trees built from those components. A parameter binding mechanism based on the design patterns for Java Beans allows to define dependencies between properties of different components, so that for example all the members are notified of specific property changes in their parent group. By using the inheritance feature of the Java programming language, powerful specialized classes can be programmed at ease.

Interaction: The Graphical User Interface

The graphical user interface (GUI) gives direct access to the properties of the model and shows the results of the simulation in real time. Every component is displayed in the model tree (Fig. 4, top left) and its properties can be managed with a property editor (top right).



Parent:	Model (10)
Selected:	true
KMax:	10
MaxLength:	0.4
MinLength:	0.3
EquiLength:	0.35
Name:	Edges
Updating:	true
KPush:	1
Updater:	Model (10)

Fig. 4: Screenshot of the ZORA application QUADMESHER

By consequently applying the design patterns for Java Beans, it was possible to create generic interfaces for all components. Thus, a standard property editor is always available even for derived components, which speeds up the development of complex models enormously.

Display: 3D real-time rendering

Since the aim is to generate three-dimensional structures, it would hardly be possible to judge the results without a 3D view of the model. The Java 3D API is used to render the components in real time. The 3D model has the same hierarchical structure as the simulation model but is completely separated. Every component of the simulation model is represented by a renderer-object in the 3D model, which takes care of displaying its component. Communication between model and representation is again based on the Java event model, so that different views on the simulation model can be generated by adding different renderers (or even no renderer at all to speed up simulation in a non interactive mode). Since the Java 3D API is rather complex to program, a default renderer for every basic component class is provided (e.g. a cube renderer for particles, a line renderer for springs etc.), so that a simple 3D view of the simulation model can be developed without difficulty. The default renderer classes implement basic interactions like picking and dragging components.

Persistence: XML Import and Export

To save the current state of a simulation model, the model tree can be exported to an XML file containing all properties of all components. In reverse, the XML file can be imported to restore the model and continue the simulation. Generic methods for reading and writing of XML data are implemented in the basic component classes – similar to the generic property editors – so that derived components can inherit the functionality and liberate the programmer from developing import-/export-routines. The universality of XML is especially beneficial for further processing the results in other programs capable of reading XML. In the following example, an XSL transformation is used to strip the output of the ZORA simulation from unnecessary information and convert it to a format that can be imported by a CAD program, which then converts it into the control code for a CNC mill – a continuous process from generation and optimization to production and the first application of the ZORA libraries.

SQUARING THE SPHERE: QUADMESHER

To show the potential of the “digital chain”, a continuously digital process from Computer Aided Design (CAD) to Computer Aided Manufacturing (CAM), the chair for CAAD developed a small pavilion for a trade fair in Basel. The pavilion has the form of a sphere with two metres radius and reaches a height of three metres. It is assembled from quadrilateral wooden frames, very much like a coffered dome. Every frame consists of four wooden boards standing perpendicular on the surface of the sphere. But while in a traditional coffered dome openings have to be aligned with the regular structure, here the structure was required to react on the deliberately asymmetric placement of the windows, deforming the quadrilateral mesh that defines the geometry of the frames.

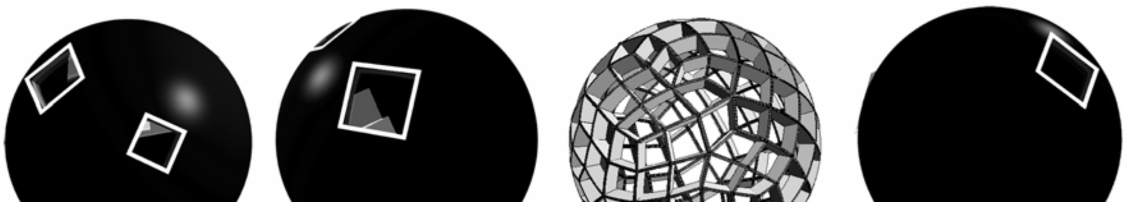


Fig. 5: Renderings of the pavilion's design and structure

To generate a quad-mesh is a well known task for everyone familiar with finite element systems, and there are a number of algorithms and programs for this problem. However, in this case there are some special boundary conditions: The mesh has to adjust to one large and four smaller openings so that the edges of the mesh are aligned with the edges of the openings. Also, the lower edges of the base frames have to align with the floor plane. The final structure can then be built by simply leaving away the frames below the floor level and inside the openings. The agent based simulation for this adaptive quad mesh is the first application based on the ZORA toolbox.

A growing mesh

To simulate an adapting mesh, again an agent based system on the base of particle dynamics was created. It consists of the following elements, derived from the basic component classes (see Fig. 6):

- The **centre** of the sphere is defined by its coordinates.
- **Nodes** are the basic components of the mesh. Each node is a particle that is connected to the centre of the sphere with a spring-relation that defines the sphere's radius. All the nodes of the model are collected in a node group.
- **Edges** are spring relations connecting two nodes. The idle length of the spring is set to the optimal length of the edge, so all edges try to achieve the same length. All the edges of the model are collected in an edge group.
- **Meshes** are group components containing a list of the four edges, two diagonals and a diagonal connection. All the meshes of the model are collected in a mesh group.
- **Diagonals** are spring relations connecting two opposite nodes of a mesh. Every mesh group contains two diagonals that try to make the mesh rectangular by adjusting their idle length according to the lengths of the mesh's edges.
- **Diagonal connections** are spring relations connecting the centre points of a mesh's diagonals. The diagonal connection assures that the mesh stays convex and as planar as possible.

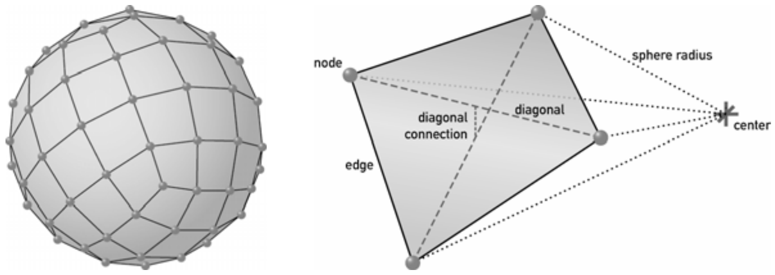


Fig. 6a, b: The mesh and its components

The resulting mesh is flexible to a certain degree, allowing it to adapt not only to changes in the sphere’s diameter but also to external forces applied to it. This is used to align the nodes and edges to the floor and the openings by introducing “magnetic” planes. The floor is a horizontal plane cutting through the sphere one metre below its centre. Each opening defines a four-sided pyramid from its corners to the centre of the sphere. By attracting the nodes while at the same time repelling the centres of the meshes, the planes align the edges in their vicinity (Fig. 7).

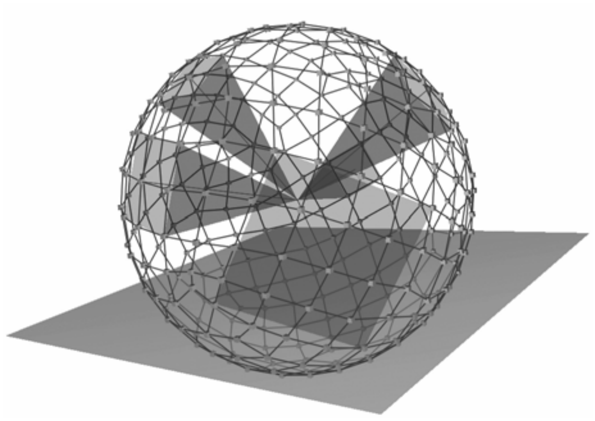


Fig. 7: Mesh aligning with floor and openings

While adjusting to the radius of the sphere and the openings, the topology of the mesh can be altered by the following three transformations, which define a Shape Grammar (Stiny and Gips 1972) that assures that only quadrilateral meshes are created. They are triggered by internal checks the meshes perform in every update cycle:

- **Splitting a node:** If the pulling forces that are applied to a node by its adjacent edges are exceeding a certain threshold, the node may split and create a new node and a new mesh for every adjacent mesh (Fig. 8a).
- **Joining two nodes:** If a mesh is deformed beyond a certain threshold (e.g. the angles are too far from being rectangular or the edges differ in length) the mesh may collapse by joining two opposite nodes into one (Fig. 8b).
- **Deleting a node**: If (by joining two nodes) two meshes are sharing three nodes, the middle node and its adjacent edges are removed and the two meshes are joined (Fig. 8c).

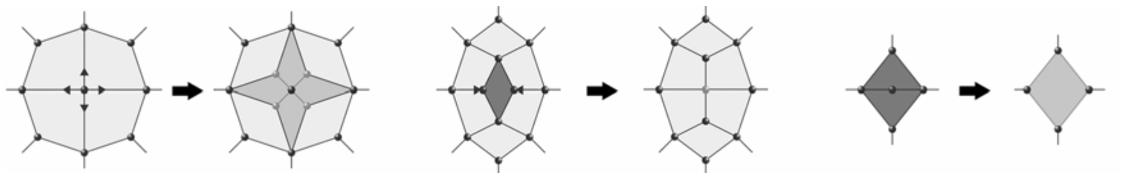


Fig. 8a-c: Mesh transformations: splitting a node, joining two nodes and removing a node

From simulation to the real world

The generation starts with a rhombic dodecahedron, a regular polyhedron composed of 12 rhombic meshes (Fig. 9a). Since the edges of the starting configuration are too long, the nodes immediately begin to split and create new meshes until the strain on the surface is bearable (Fig. 9b). When the centre distance is slowly increased, more meshes are added until the sphere reaches its target size (Fig. 9c). Switching on the "magnetic planes" of the openings and the floor plane aligns the meshes with the desired lines.

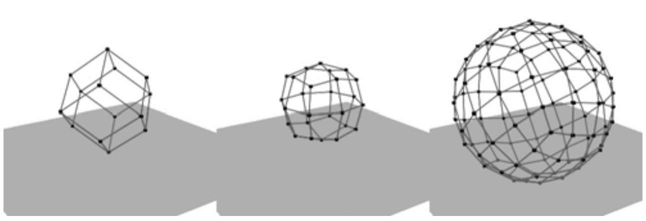


Fig. 9a-c: Starting configuration and growing mesh

The resulting mesh can be exported to an XML file, which is further processed by an XSL transformation to the import format of second software. This software, programmed in the scripting language of a CAD package, generates the volumes of the wooden boards based on the coordinates of the mesh. After checking the structure within the CAD program, another script automatically creates the CNC code to produce all the boards on a computerized five-axis mill.

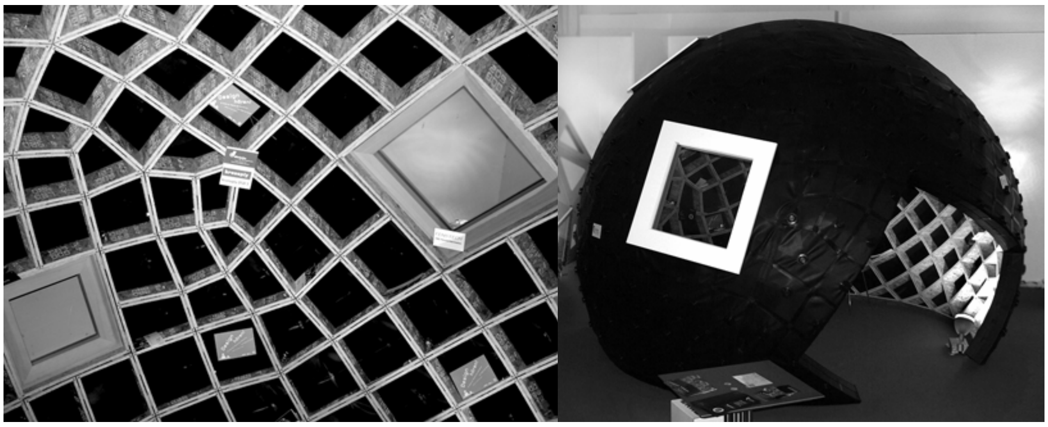


Fig. 10a, b: Inside and outside view of the finished pavilion

CONCLUSIONS

It has been shown that self-organization in agent based simulation can be used to organise and optimise complex spatial constructions in order to comply with requirements of form, function and structure. In other words, emergence can help architects and engineers to create exact and formal configurations from metaphorical or informal ideas. However, getting an agent based simulation to create the desired result is a process of trial and error, since there is not yet a coherent methodology of "designing for emergence". Unfortunately this does not only mean to program customised simulation software for each project. It also must be possible to interactively examine different settings of the model, which requires real-time presentation of the results and a user interface – both difficult features to program. The presented toolbox is intended to facilitate the development of such simulation systems, thus speeding up the loop of design, implementation and testing. Based on the ZORA toolbox it is planned to closely examine various further applications of self-organising structures.

CREDITS

I would like to express my gratitude to the following people without whom this work would not have been possible: Prof. Dr. Ludger Hovestadt (head of the chair for CAAD) for supporting my studies. Kees Christiaanse and Andy Woodcock (KCAP, Rotterdam), Joop Paul and Arjan Harbraken (Ove Arup, Amsterdam) for their collaboration and support in the Groningen project. And last but not least the Swissbau-pavilion team, especially my colleagues Christoph Schindler and Markus Braach for their collaboration and the permission to show their work.

REFERENCES

ARUP. (2004). "Beijing National Stadium, Olympic Green, China." Ove Arup & Partner. Corporate Website, available at http://www.arup.com/project.cfm?pageid=2184.

Balmond, C., Smith, J., and Brensing, C. (2002). Informal, Prestel, Munich.

Bull, S., Downing, Steve. (2004). "Beijing Water Cube - the IT challenge." The Structural Engineer, 23-26.

Holland, J. H. (1998). Emergence - from chaos to order, Addison-Wesley, Reading, Mass.

Johnson, S. (2002). Emergence - the connected lives of ants, brains, cities and software, Penguin Books, London.

Kennedy, J., Eberhart, R. C., and Shi, Y. (2001). Swarm intelligence, Morgan Kaufmann, San Francisco.

Martini, K. (2001). "Non-linear Structural Analysis as Real-Time Animation - Borrowing from the Arcade." CAAD Futures 2001 - Proceedings of the ninth international conference held at the Eindhoven University of Technology, Kluwer Academic Publishers, Dordrecht, Boston, London.

Scheurer, F. (2003). "The Groningen Twister - an experiment in applied generative design." Generative Art 2003. Proceedings of the 4th international conference, C. Soddu, ed., Milan.

Stiny, G., and Gips, J. (1972). "Shape Grammars and the Generative Specification of Painting and Sculpture." Proceedings of IFIP Congress71, C. V. Freiman, ed., North-Holland, Amsterdam, 1460-1465.

Witkin, A. P. (1997). "Particle System Dynamics." SIGGRAPH 1997 lecture notes.