A Trans-disciplinary Program for Biomimetic Computing and Architectural Design

Sebastian von Mammen, Joshua M. Taron
University of Calgary, Canada

In this article, we present our trans-disciplinary approach to teaching biomimetic computing and architectural design to graduate students in Architecture. In particular, we present our selection of topics, their initial, conceptual presentation to the students, their appropriation by the students through programming and examples of the students' material implementations in architectural design projects.

Introduction

Multi-agent systems promote modeling of complex processes by researchers and designers without the need for a profound background in Mathematics. Conceptual models can be directly translated into programming code and the consequences of a previously theorized model can visually unfold, undergo rigorous analysis, and experience iterative improvement. Agent-based modeling also empowers designers to apply a paradigm of self-organizing systems: swarms of reactive software agents engaging in complex interactions, potentially even reproducing constructive processes. Experiencing and investigating complex systems in nature is another important aspect that promotes the outlined approach to design. Developmental processes in organisms, evolution, self-organizing formations in cell populations or animal societies all serve as an invaluable source for inspiration and for comprehending the ways decentralized, self-organizing, emergent multi-agent models can carry fruits for research and design.

In this work, we present our approach to teaching and training the ideas of agent-based modeling and related topics around complex biological systems. We have established a trans-disciplinary course setup between computer science and architecture. A computer science course on biomimetic computation provides the theoretical foundation and programming know-how for developing agent-based software simulations with a focus on developmental, generative, and interactive processes. Architecture students concurrently enroll in an Architecture research studio in which they have the opportunity to apply and evolve their agent-based models developed in the computer science course. An inspiring feedback cycle emerges from the trans-disciplinary, theoretically founded and practically applied tandem of project-driven courses.

The contents and the coursework are closely attuned to maximize the opportunity for mutual synergetic fertilization of skills and ideas. In the Computer Science course, students are first familiarized with the basic concepts of computational processes and algorithms using Processing. Units on coding basics culminate in live programming demos that apply the gathered knowledge about basic data structures and process flow. Simple yet colorful simulation examples are crafted from scratch in front of the class, thoroughly discussed and made available online for future reference. Subsequent lecture units present biological examples of concepts like developmental growth, self-organization and evolutionary processes. Corresponding programming codes are presented in class. Finally, students' projects commence, maturing from the initial proposals over prototype implementations into original architectural design works (supported through the Architecture research studio).

The Architecture research studio component of the endeavor provides an outlet through which these computational processes can be tested at a variety of architectural scales and formations. By focusing on complex processes in heterogeneous urban environments, multi-agent systems serve as both a tool for mapping cities as well as for the production of architectural design techniques. In our case, students are assigned a partially completed skyscraper in downtown Calgary as their site with the task of intervening in the typical procedural construction/assembly processes necessary to complete the tower. The exercise challenges students to use agent-based models to graft into an already ongoing procedural process thereby augmenting its formal, visual and programmatic performance. The results of the studio are series of new tower iterations using agent-based techniques developed and supported through the Computer Science course.

These agent-based designs provide the material results of the trans-disciplinary exercise, which are evaluated for the purposes of improving the next iteration of the experiment. Methods are discussed that might allow students to improve upon previous years' achievements and thus increase the intensity and intelligence of the models themselves over the course of time.

Programming Nature

Computation happens through manipulating data. Traditionally, sequences of instructions that determine how certain data are manipulated are subsumed under high-level commands, generally referred to as procedures, functions, macros, or methods. Methods can be associated with specific data objects that combine various kinds of information, e.g. symbolic strings, numeric data, or other data objects. A repeatedly occurring example would be a *Person* object, for instance *person1* with the attributes *name* = "Susan", *age* = 32 and *gender = female*. The execution of a method in respect to a *Person* object could, for instance, update Susan's age to 33.

Similar to subsuming instruction sequences, objects and their associated methods can be inherited by other object classes. An *Employee* class, for example, could expand the attributes and methods of the *Person* object class. This *object-oriented programming* approach represents the state-of-the-art programming paradigm in software engineering. It is of great value because the programmer can immediately understand and work with complex code objects and use them for creating his own software. A comprehensive introduction to object-oriented programming is provided by [1].

Agent-based Programming

Agent-based programming is an extension of the object-oriented approach. It turns passive data objects into active *agents* that act in accordance with their *behavior*, their *situation* and their available *data* [2]. One speaks of *Multi-Agent Systems (MAS),* if there are multiple agents at work. The programmer endows the agents with behaviors and properties in such a way that they work efficiently together and accomplish computationally challenging tasks. Potential benefits of MAS can be high robustness as failures in parts of the system can be compensated by intact agents or high efficiency as tasks can be performed in parallel and be assigned with respect to the involved agents' specializations.

MAS lend themselves naturally for designing biomimetic computational models, in which systems of molecules, cells, organs, organisms, or societies are retraced. Individuals in these systems act based on their own agenda and contribute to the *emergence* of high-level processes or designs [3]. The structural properties and the behaviors of living organisms have evolved to yield streamlined, adaptive metabolic processes to occupy and exploit ecological niches. The agent-based modeling approach allows the designer to directly map physiological properties and biological behaviors to computational representations. The only limitations are the knowledge and creativity of the designer on the one hand, and computational power on the other hand.

Swarms

MAS can be designed in many ways. One can, for instance, implement a centralized controller agent that oversees the ongoing processes and concerts the activities of the remaining agents as it sees fit. Inspired by biological systems, one can alternatively attempt to configure the agents in such a way that a centralized control is not required. A system of such *decentralized* agents brings a number of advantages: (1) It is generally more robust against failure as there is no crucial, central part that can go missing. (2) The computational cost for coordinating the agents is reduced by the agents making locally informed decisions. (3) If the task at hand can be divided into independent subtasks, they can be accomplished faster as there are no holdups. Besides such computationally intriguing properties of decentralized systems, there are other aspects that reach even further. For instance, they support the idea of simulating biological *self-organization*, where a system can reach *self-maintaining* states independent of its initial configuration [4]. In general, one can say that a decentralized system is a system whose agents can act freely, whereas any kind of control infrastructure introduces varying degrees of limitations in respect to the possible interaction processes. Of course, depending on the system, a rigid control infrastructure might actually be vital, like the coordination of our motor-sensory activity through the central nervous system. Along these lines, we would like to underline that control infrastructures are the results of self-organizing processes themselves. Therefore, decentralized MAS, or *swarms*, seem to be the least biased, most direct, and thus, most profound approach to computational modeling.

Development

Ultimately, living organisms are biochemical structures that drive their own development, maintenance, and reproduction. These seemingly distinct objectives can all be reduced to systematic metabolic processes, that is the construction and destruction of products ranging from simple molecules to large molecular chains to cells and complex tissues [2]. From this perspective, processes describe the flow of state changes, whereas structures refer to materializations that persist for a perceivably long period of time. Even this careful attempt to distinguish processes from structures emphasizes the role of the observer and it forces us to accept that structure and process are two closely interwoven aspects of life.

Computational swarms can retrace developmental processes, if their interactions yield persistent structures. Swarm agents can create structures in numerous ways. They can, for instance, become part of a larger structure like simulated molecules in Artificial Chemistries [5]. They can deposit building blocks when building their nests like wasps [6], or hollow out tunnels and chambers like ants [7].

Due to the swarm agents' degrees of freedom, it is a challenge to assign them behaviors and properties that make them interact in a productive, coordinated fashion [8]. The structures built by a swarm, however, provide meaningful evidence of the swarm's productivity [9], which can serve to find swarm configurations that yield desirable designs, for instance by means of evolutionary computation [10].

Hands On Code

We can only assume that a small group of students has been exposed to programming or 3D modeling before enrolling in our trans-disciplinary program. As a result, we have to guide them through the very first steps of a programming curriculum to arrive at the point where they are empowered to read and manipulate programming code or to be motivated to design and implement programs from scratch.

Visual programming environments like Grasshopper/Rhino, Quartz Composer, or Max/MSP provide high-level interfaces that make it easy to compose intriguing programs by hiding implementation details that are usually unimportant for the designer. These environments can offer simple interfaces because they constrain the way designers think, i.e. by forcing them to follow a functional programming paradigm.

In addition to the advantages of these environments, we teach the students in Processing [11], an environment that empowers them with the expressiveness of the established, object-oriented Java programming language. Understanding programming on the level of algorithmic instruction sequences and memory manipulation in terms of a generic programming language allows one to naturally understand other languages and high-level interfaces as well. Furthermore, it enables the programmer to break out of an imposed programming paradigm, and in the case of our trans-disciplinary discourse, create the programming infrastructure for agent-based models and simulations.

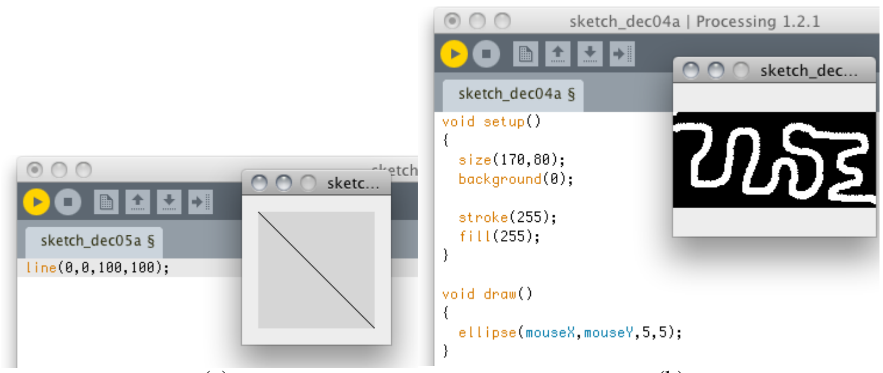
Surface as Architectural and Mathematical Territory

Architectural form serves as a common territory where both visual and agent-based programming techniques can be deployed. While students are being introduced to algorithmic approaches in the Computer Science course, the Architecture studio runs through a series of NURBS modeling exercises enabling students to tackle architectural problems of scale, massing and circulation while allowing students to become familiar with non-linear geometric relationships within those visual environments such as points, curves, surfaces and manifold spaces. The results are designed not just for producing spaces for human inhabitation, but more so for the purpose of defining explicit mathematical territories for the students' yet-to-be-developed biomimetic code to inhabit and further articulate.

The exercises continue to evolve in complexity by employing tactics of object instancing, duplication and formal reproduction. This is particularly useful in partially previewing problems and opportunities afforded by MAS. Principles of transformation, gradient change and morphological part-to-part behaviors in these architectural explorations establish programmatic strategies and aesthetic sensibilities that influence and inform students' Computer Science projects.

Getting Started with Swarm-programming

Processing is widely used in architecture and art [12-14]. Writing a program, or *sketch* in Processing lingo, can be as simple as typing a drawing command such as *line(0,0,100,100);* into its editor window and clicking the play button (Fig. 1(a)). Comprehensive documentation and references are accessible through Processing's menu. Fig. 1(b) shows a simple interactive Processing sketch that, when started, changes the simulation window size to 170 by 80 pixels, sets its background color to black (color value: 0) and sets the paint color to white (value: 255). For as long as this sketch is running, a circle of radius 5 will be drawn where the mouse pointer hovers over the simulation window—resulting in a squiggly line in the given example.



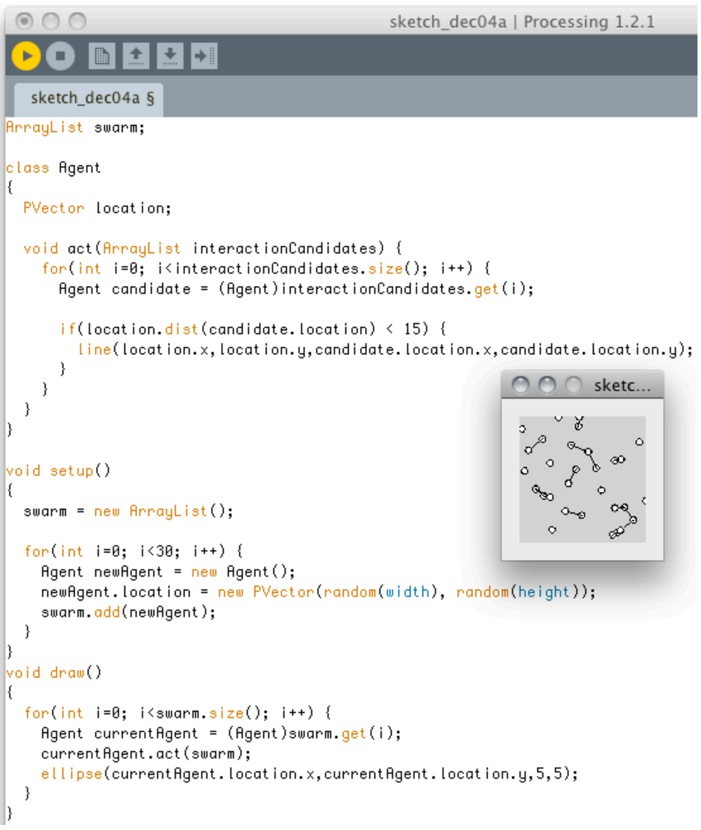
```
line(0,0,100,100);
```

```
void setup()
{
  size(170,80);
  background(0);

  stroke(255);
  fill(255);
}

void draw()
{
  ellipse(mouseX,mouseY,5,5);
}
```

Fig. 1 (a) Processing offers an easy-to-use editor and various predefined drawing commands such as *line()*. (b) Code inside the *setup()* method is executed when the simulation is started. *draw()* is executed repeatedly until the simulation is stopped.

Fig. 2 shows a basic swarm-programming infrastructure in Processing code. In its *setup()* method, new *Agent* objects are created. Their *locations* are set to somewhere on the canvas (dimensions: *width x height*). The newly created *Agent* objects are stored in a list called *swarm*. The *draw()* method iterates through this *swarm* list, executes each swarm agent's *act()* method and renders it as a circle on the canvas. In the given example, the whole swarm may inform each agent's actions. As a result, the *swarm* list is used as the *interactionCandidates* parameter of the agents' *act()* method. Instead of interacting, i.e. changing its or its partner's state, the agents in the given example only indicate with whom they would interact by drawing a line to each other. A distance lower than 15 between two agents is the criterion for interaction in the given case.

The example shown in Fig. 2 has two purposes. First, it shows how a very generic swarm-programming infrastructure can be created. Second, it indicates the potential interactions by drawing lines between subsets of agents. Extensions to the *Agent* class in respect to its attributes and its *act()* method infuse the model with meaning. The designer/programmer, therefore, has to decide what the agents represent, what their relationships are, which control instances and constraints should be applied, and how emerging processes and structures inform each other.



```
ArrayList swarm;

class Agent
{
  PVector location;

  void act(ArrayList interactionCandidates) {
    for(int i=0; i<interactionCandidates.size(); i++) {
      Agent candidate = (Agent)interactionCandidates.get(i);

      if(location.dist(candidate.location) < 15) {
        line(location.x,location.y,candidate.location.x,candidate.location.y);
      }
    }
  }
}

void setup()
{
  swarm = new ArrayList();

  for(int i=0; i<30; i++) {
    Agent newAgent = new Agent();
    newAgent.location = new PVector(random(width), random(height));
    swarm.add(newAgent);
  }
}
void draw()
{
  for(int i=0; i<swarm.size(); i++) {
    Agent currentAgent = (Agent)swarm.get(i);
    currentAgent.act(swarm);
    ellipse(currentAgent.location.x,currentAgent.location.y,5,5);
  }
}
```

Fig. 2 A basic swarm-programming infrastructure in Processing code. The *draw()* method executes the *act()* method of a list of Agent objects.

Explorations of Biomimetic Design

The students in our program are asked to develop a sense for dynamic swarm systems and explore how they can impact the creation of architecture. In this section, we outline several examples of student projects that cover a range of conceptual and programmatic ideas.

Sentient Surfaces

Jared Brookes and Michael Scantland worked on an extension of the previous programming example. Agents serve as the vertices of a mesh and their neighbor relations translate into the mesh topology. Movements of agents can thus dynamically reconfigure the surface. Fig. 3 shows a basic setup of an according simulation of *sentient surfaces*. Fig. 4 depicts exploration states of the emerging mesh dynamics.

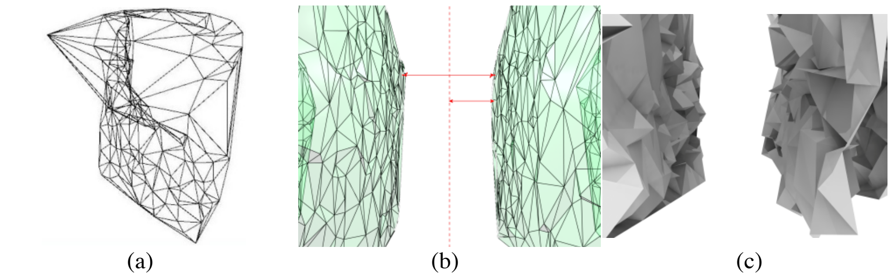


Fig.3 In *sentient surfaces*, agents serve as mesh vertices and their movements reconfigure the structures. (a) A single agent drags its neighbors out of the mesh. (b) Conceptual illustration of perception thresholds between two sentient surfaces. (c) Attracting and repelling forces among the agents result in rough surface configurations.

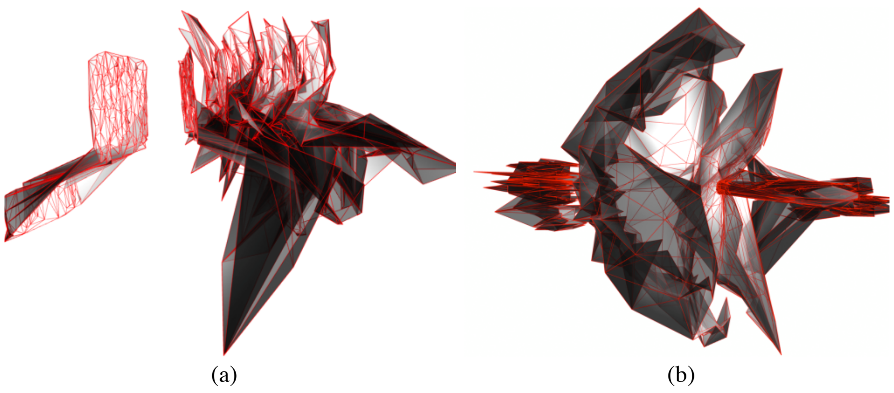
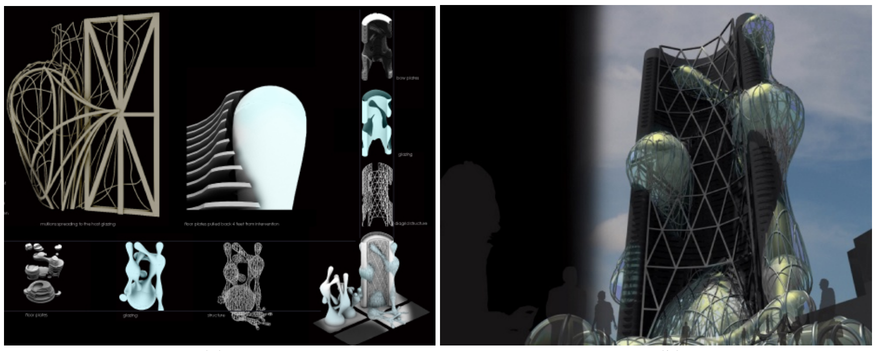


Fig. 4 Behavior studies of sets of dynamic swarm surfaces. (a) Repelling field conditions between a pair of swarm surfaces yield erratic, frayed structures. (b) One agent shoots through several swarm surfaces.

Scantland's studio project with Julie Brache paralleled the sentient surface investigation through the deployment of a nodal network distributed throughout the tower site. While addressing a different frequency and scale of modulation in the studio project, displacement of interior spaces and replacement of exterior structure formed an integrated relationship between differential programs. Fig. 5(a) diagrams the responsive relationship between exterior structure, interior space and building envelope generated by grafting the two systems together. Fig.

5(b) illustrates the exterior view of the tower as the nodal network weaves through the building.

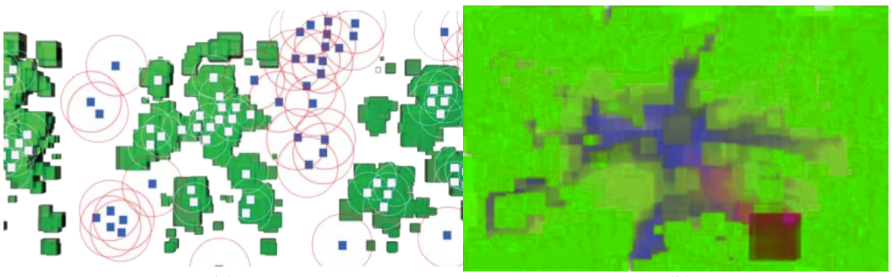


(a) (b)

Fig.5 (a) Grafting Strategy (b) Exterior Perspective (authors: Scantland + Brache)

Creating Space through Diversity

Ryan Palibroda approached the organization of land occupation from a 2D perspective. Agents keep pushing each other in accordance with their preferences until a steady state is reached (Fig. 6).



(a) (b)

Fig. 6 (a) Agents of a specific type form clusters as they push agents of other types away. (b) Opposing forces between different agent types result in organically shaped high-density areas.

Ryan Trefz also experimented with different agent types (Fig. 7). In addition to repelling forces, Trefz relied on the whole array of *boid urges* to inform his agents' flight: *alignment, cohesion, separation* [15]. Differently configured agents can be distinguished through size and hue.

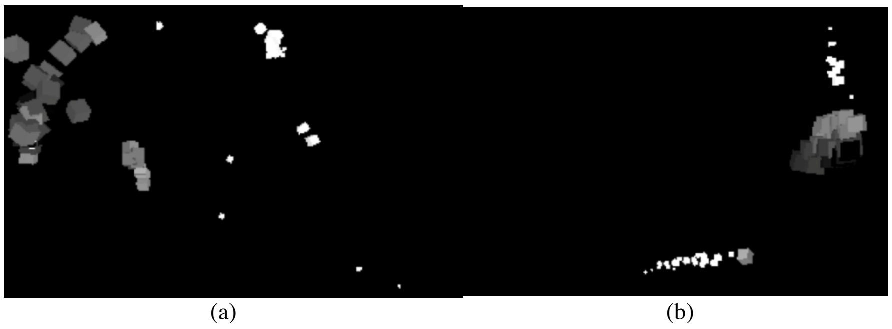


Fig. 7 Cluttering and clustering flocking formations to inform a dynamic architecture inspired by Craig Reynolds' *boids* [15] and Nicholas Reeves' Mascarillons [16].

Trefz and Palibroda collaborated in studio to produce a tower whereby the building exterior operated like a solar landscape. While the exterior borrowed tactics from Palibroda's displaced fields (Fig. 8(a)), interior spaces were formed by tracing flocking positions into structural networks (Fig. 8(b)).

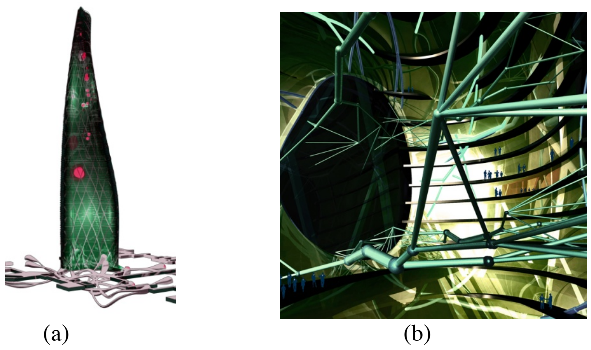


Fig. 8 (a) Hotspots embedded within the building facade operate as attractors for (b) interior conditions that trace the position of flocking particles through space.

Carving Structures

Chris Vander Hoek explored subtractive generation of architectural spaces. In his project, commuting swarms (Fig. 9(a)) carve out cubic volumes that recursively decompose into eight smaller cubes on collision with a swarm individual (Fig. 9).

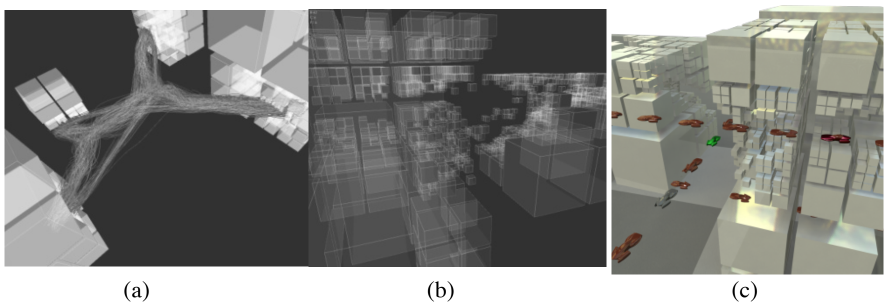


Fig. 9 (a & b) Commuting swarms carve out cubic volumes. Upon collision between a swarm individual and a volume, it recursively decomposes into eight cubes until it completely disappears. (c) Future city optimized for mid-air traffic flow.

His studio project with Arthur Coudeville used the same particle swarm to generate a 3D voronoi extrusion from the building base in order to extend the interiority of the tower into the adjacent plaza (Fig. 10(a)). Physical explorations focused on fabrication and assembly techniques that subtract from adjacent spaces.

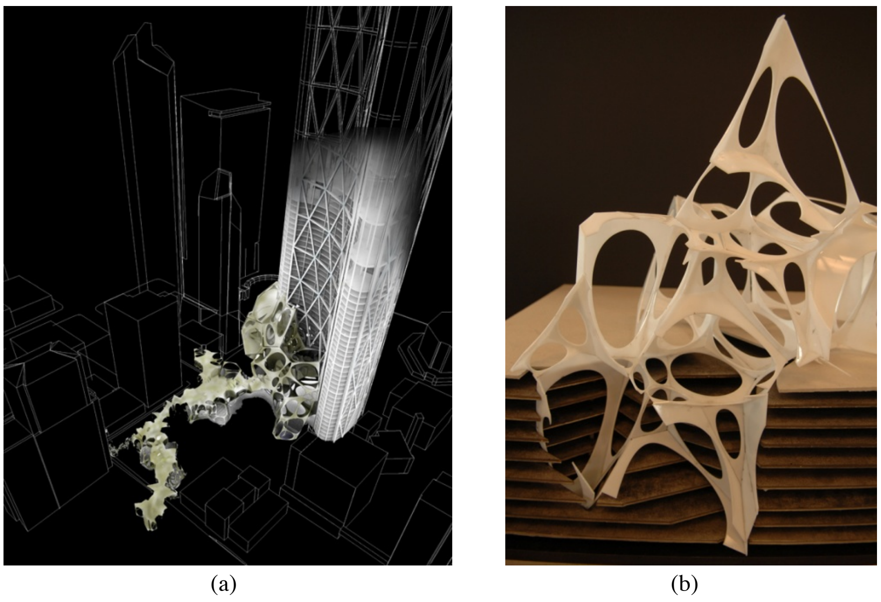


Fig. 10 (a) Aerial perspective (b) Sectional study model carves away from the subterranean parking lot below the site.

Procreating Particles

Jonathan Choo and Fadilah Hamid applied their knowledge of agent-based modeling in a simulation written in MEL, the scripting language of the Maya rendering software. A predefined space is populated with agents (Fig. 9(a)) that attract and repel each other (Fig. 9(b)) and procreate on collision. The inter-agent relations translate into a smooth surface with hollow spaces (Fig. 9(c) and Fig. 10).

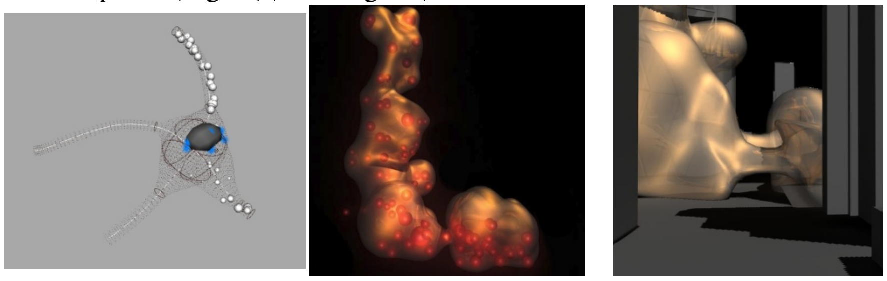

(a) (b) (c)

Fig. 9 (a) Slowly a predefined volume is populated with agents (represented as spheres). (b) Attracting agents are colored in bright red. (c) A smooth mesh encloses the interacting agents.

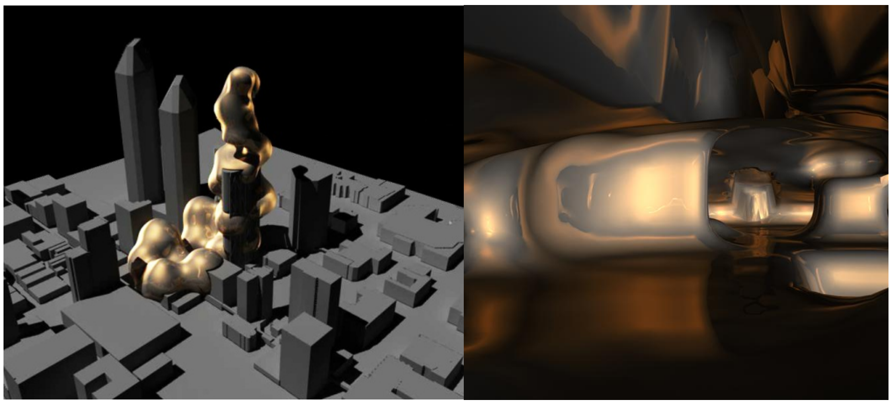

(a) (b)

Fig. 10 (a) An architectural site is redefined by interacting particles. (b) The interior space of the resulting space.

Program Evaluation & Future Work

We learned much in this first attempt at a trans-disciplinary program. Success manifested through a combination of algorithmic and biological foundations offered through the Computer Science course while the Architecture research studio provided a space for exploration and application of those techniques in the context of the built environment. Problems developed in the studio were in turn framed as means for evolving the projects in the Computer Science course. We found that enthusiasm was renewed on both fronts with the constant unfolding of new problems, innovative solutions and range of applications.

The Computer Science projects did suffer from starting only after a number of weeks of introductory coding exercises were completed. As such, group projects did not have the full term to evolve and develop. Future course iterations will employ instructor-generated podcasts of basic lessons that students will make use of while initiating their own projects early in the term in order to mitigate the issue.

The Architecture research studio did provide an appropriate moment in the curriculum to engage this kind of material (in the sixth term of a six-term graduate program) given a developed knowledge during their first two and a half years of graduate education. However, the Computer Science course is seen as introductory and would certainly serve as a valuable skill to have earlier in a graduate curriculum (Architecture, Computer Science or otherwise). A looser but perhaps more profound connection might exist between the Computer Science course and an Architecture studio positioned earlier in a graduate program, thus giving students more opportunities to experiment with and develop their coding skills as they continue through school.

We view the premise of decentralized control in both Computer Science and Architecture as fundamental to the advancement of our own research and in both disciplines at large. By producing a pedagogical framework whereby swarms, natural systems and Architecture operate within an interchangeable space, each can inform the others in unique and useful ways; envisioning biomimetic code as Architecture, Architecture as nature, and nature as codified milieu. While the courses reinforce one another by structuring the exchange of information between one another, less resolved are the structures that might produce continuity and evolution from one year to the next. By archiving code packages developed in previous course iterations, incoming students could develop those definitions further, hybridize multiple definitions together or if nothing is attractive to them, start something from scratch, thus broadening the “gene pool.” This gives code developed in a given term the chance to “go on living” after a course iteration has ended.

Another line of continuation might make a selection of previously developed definitions available to an Architecture research studio with the charge that they design with/make use of them in an architectural capacity. This is already a model in use whereby swarm code developed by Taron is released to Architecture students for use as a generative design tool.

References

1. Booch G, Maksimchuk R, Engle M, Young B, Conallen J, Houston K (2008) Object-oriented analysis and design with applications, 3rd edition, Addison-Wesley Professional
2. Wooldrige MJ (2009) An Introduction to MultiAgent Systems, 2nd edition, John Wiley & Sons Ltd.
3. Camazine S, Deneubourg JL, Franks NR, Sneyd J, Theraulaz G, Bonabeau, E (2003) Self-Organization in Biological Systems, Princeton Studies in Complexity, Princeton University Press
4. Banzhaf W (2004) Artificial Chemistries – Towards Constructive Dynamical Systems in Solid State Phenomena 97-98: 43-50
5. Dittrich P, Ziegler J, Banzhaf W (2001) Artificial Chemistries – A Review in Artificial Life 7:225-275
6. Karsai I, Penzes Z (1993) Comb Building in Social Wasps: Self-organization and Stigmergic Script in Theoretical Biology 161,4:505-525
7. Hölldobler B, Wilson EO (1990) The Ants, Springer Berlin-Heidelberg
8. Bonabeau E, Dorigo M, Theraulaz G (1999) Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press
9. Jacob C, von Mammen S (2007) Swarm Grammars: Growing Dynamic Structures in 3D Agent Spaces in Digitial Creativity 18:54-64
10. von Mammen S, Jacob C (2008) Evolutionary Swarm Design of Architectural Idea Models, Genetic and Evolutionary Computation Conference (CECCO), 143-150, ACM Press
11. Aesthetics and Computation Group, MIT Media Lab (2010) Processing online at: http://processing.org
12. Erdman D, Lee C (2010) online http://davidclovers.com/
13. Kokkugia (2010) online http://kokkugia.com/
14. Reas CEB (2010) online http://reas.com/category.php?section=works
15. Reynolds C (1987) Flocks, Herds, and Schools: A Distributed Behavioral Model, SIGGRAPH Conference Proceedings 4:25-34, ACM Press
16. Nembrini J, Reeves N, Poncet E, Martionli A, Winfield A (2005) Mascarillons: Flying Swarm Intelligence for Architectural Research, Swarm Intelligence Symposium, 225-232, IEEE Press