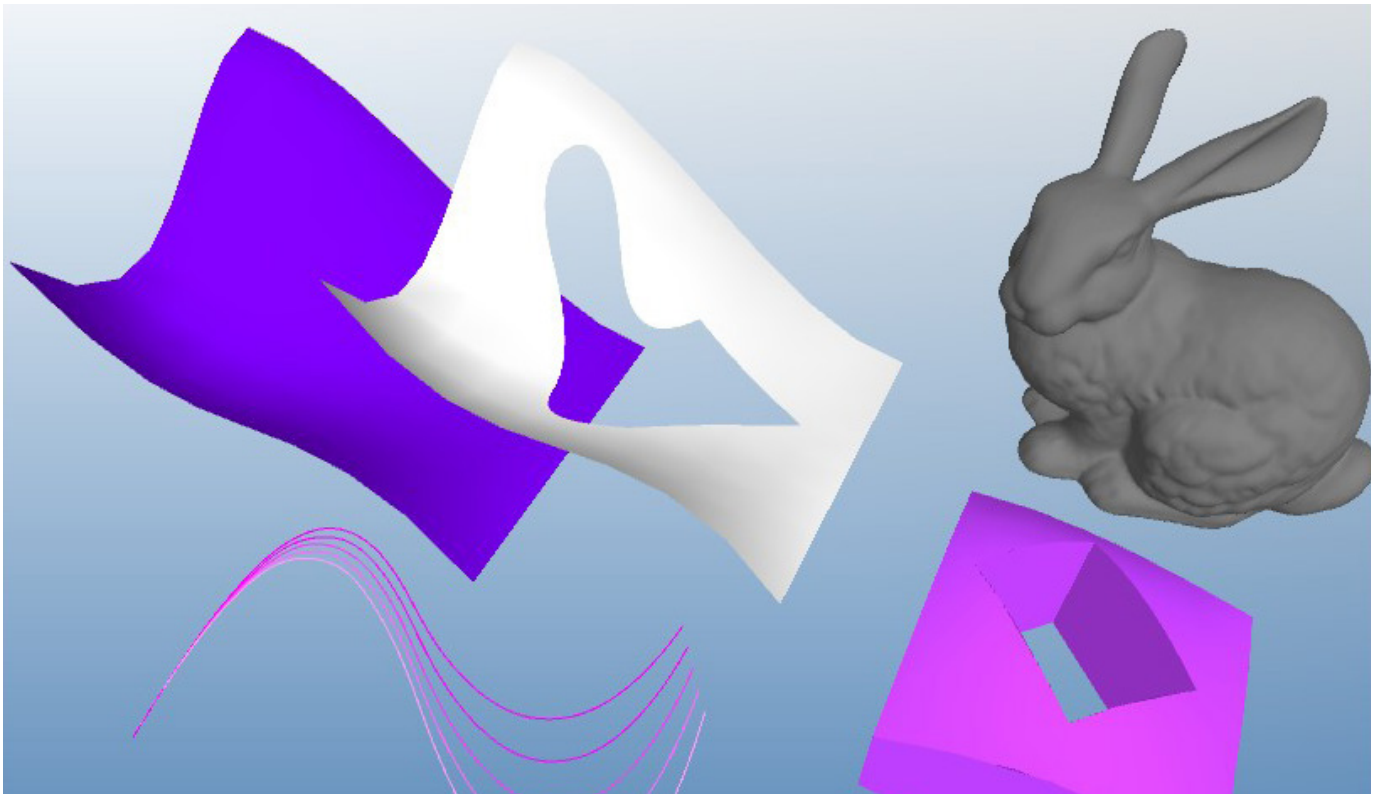
IGEO

ALGORITHM DEVELOPMENT ENVIRONMENT FOR COMPUTATIONAL DESIGN CODERS WITH INTEGRATION OF NURBS GEOMETRY MODELING AND AGENT-BASED MODELING

Satoru Sugihara Southern California Institute of Architecture



1 Types of geometries supported in iGeo: point, NURBS curve, NURBS surface, BREP, and polygon mesh

ABSTRACT

Writing codes of computational design algorithms to perform agent-based modeling and generate complex geometry is powerful but not simple. To help computational design coders write algorithms for complex geometry and agent-based modeling, an algorithm development environment named iGeo has been developed as an open source library on Processing. This paper describes the design of this library for efficient and effective algorithm development integrating NURBS geometry modeling and agent-based modeling. It also describes computational design algorithms developed on this environment for various projects as case studies to evaluate how the environment helps algorithm development and the computational design process. The paper discusses advantages and disadvantages observed in the case studies and in comparison to existing libraries.

INTRODUCTION

Use of agent-based algorithms is getting popular in computational design. They are known to be able to generate complex self-organizing geometry but difficult to control (Sugihara 2011a). To develop algorithms with agents' behaviors for 3D geometry modeling, comprehensive control of space and time is important. Many existing 3D modeling software provide a wide range of effective geometry operations. However, their supports on geometry modeling by iterative behavioral rules are usually limited. It is also difficult for these software to execute indeterminate iterative algorithms including agent-based algorithms, which have no predefined end of execution. On the other hand, when an algorithm is developed as a piece of independent software, indeterminate algorithms can be executed, but a designer needs to manage all codes to handle the iteration process through time, 3D geometry data and their visualization, and graphical user interfaces. The primary objective of the development of iGeo is to help computational design coders explore design possibilities in agent-based algorithms by providing a comprehensive environment to develop agent-based algorithms and 3D geometry modeling simultaneously. The secondary objective is to develop an effective and practical geometry modeling API for architectural design practice and provide an efficient coding environment to minimize the coding effort.

THE DESIGN OF THE LIBRARY

For these objectives, the iGeo library is designed to have the following features: Use of Processing as coding environment, automatic data management of geometry and agents, and method chaining for coding efficiency (described in sections Processing as a Host Coding Environment, Geometry and Agent Object Management by an Internal Server, and Concise Coding and Method Chaining); NURBS geometry modeling, agent-based modeling framework and predefined agent classes for effective integration of geometry and agent-based modeling (section NURBS, Geometry Modeling, Agent-Based Modeling and Integration with Geometry Modeling, 2.4); and Non-geometric information and file I/O for practicality in architectural design practice (section Non-Geometric Information and File I/O for Practical Use of Modeling Data).

PROCESSING AS A HOST CODING ENVIRONMENT

To run indeterminate iterative algorithms, iGeo is built as a library on the existing concise coding environment, Processing (Reas and Fry 2007). Processing provides an integrated coding environment with a simple code execution. On Processing, iGeo provides 3D visual representation in multi-views rendered by JOGL (Java binding for OpenGL) and 3D mouse navigation so users can focus on algorithms to generate 3D geometry rather than on visualization on a 2D screen.

NURBS GEOMETRY MODELING

iGeo provides NURBS geometry modeling functions for efficient and continuous geometry control via a UV parameter space together with a 3D vector math library for precise geometric operations. The NURBS surface class also handles inner and outer trim curves. A BREP class is experimentally implemented as a logical set of multiple trimmed NURBS surfaces (Figure 1).

The implementation of NURBS geometry follows the mathematical definition with a set of control points, degrees, knot vectors, and Bernstein polynomial basis functions. The polynomials are implemented as symbolic formulas with symbolic manipulation methods including differentiation. This computer algebra system approach allows us to calculate tangent and normal vectors more precisely and robustly than a numerical analysis approach by sampling points at infinitesimal neighborhoods on the geometry.

AGENT-BASED MODELING AND INTEGRATION WITH GEOMETRY MODELING

For agent-based modeling, iGeo provides a simple agent template and predefined agents forming an agent-modeling framework. The agent template consists of only two methods: the update method and the interact method. A user defines an agent by writing a subclass of the template class (IAgent) with these two methods (Algorithm 1). The update method describes behaviors to update the agent's internal states in each time frame. The interact method describes the behaviors to interact with other agents influencing their internal states. The interact method has one input of a list containing all existing agents at the time frame. To define an interaction behavior, a user iterates through the list and picks specific types of agents to interact with. There is also another simpler template for the interact method (Algorithm 2). This template takes one agent instance as its input. Whereas the interact method with a list input is invoked once in each time frame, the simpler one is invoked for each existing agents in each time frame. The simpler template can be used to define a behavior described only by two agents, such as drawing a line when two agents are within a certain distance. On the other hand, if a behavior definition involves multiple other agents, such as transforming geometry when it finds 30 other agents within a certain distance, the interact method with a list input is suitable. The interact methods are invoked for all agents before invoking update methods in each time frame, so agents can gather information from others before updating own internal states.

Algorithm 1: Example code of agent definition

```
class UsersAgent extends IAgent{
    void interact(ArrayList<IDynamics> agents){
        // behaviors to interact with all other agents.
    }
    void update(){
        // behaviors to update own internal states.
    }
}
```

Algorithm 2: Example code of agent definition with simpler interaction method

```
class UsersAgent2 extends IAgent{
    void interact(IDynamics agent){
        // behaviors to interact with each agent.
    }
    void update(){
        // behaviors of update own internal states.
    }
}
```

For integration of geometry modeling and agent-based design, iGeo provides two ways to establish relationships between geometry and agents. One way is for a geometric object to own agents inside to update the geometry itself dynamically. Another way is for an agent to exist by itself in case it has no association with geometry, such as a gravity agent, or it controls multiple geometries as the agent's subordinate data.

PREDEFINED AGENT CLASSES FOR EFFICIENT AGENT ALGORITHM DEVELOPMENT

iGeo contains various predefined agent classes following the agent framework (see Appendix). The base agent class IAgent handles iterations and updates through time, but it does not contain any geometry. Predefined agent classes inherit IAgent, add geometry, and define interaction and update behaviors to achieve known types of agents. There are four major types of predefined agents in iGeo: particle, swarm, tension, and force field. To perform physical simulation under the agent framework, properties such as velocity and force are added to those agents.

A particle agent contains a point, velocity, force, and weight and simulates physical movement under applied forces. This particle class (IParticle) defines a method called "push" to take a force vector. Every time frame, a particle agent accumulates all force vectors applied through the push method in the interaction phase, and it updates the point location by the force and velocity in the update phase. A swarm class (IBoid) is provided as a subclass of the particle class adding the known swarm behaviors with the three rules of cohesion, separation, and alignment. Those force-generating rules are written in the interact method, and forces are applied via the push method.

A tension agent connects two particle agents with a tensile force. A tensile vector is generated and given to the particle's push method in the tension agent's interact method. A force field agent also gives a force vector to the push methods of all existing particles in the force field's interact method. iGeo provides various types of force fields formed by a vector (e.g., gravity), by a point (e.g., attractor), by a NURBS curve (e.g., curve tangent force field), and by a NURBS surface (e.g., surface normal field) (See Appendix).

Because of the force-based particle interaction mechanism with the push method, any agents inheriting particle class can receive forces from any force field agents automatically without writing codes to specify interaction rules, thus minimizing the amount of coding. Force field agents also have a mechanism to apply forces only to specified agent classes by registering the classes at a target method.

GEOMETRY AND AGENT OBJECT MANAGEMENT BY AN INTERNAL SERVER

iGeo manages instances of geometric objects and agents automatically by an internal data server. All geometry classes and agent classes are subclasses of iGeo's object class, IObject (see Appendix). An instance of IObject subclasses finds an internal server and registers itself to the server at the IObject's constructor. An instance of the internal server is accessed th rough a static method using the singleton design pattern (Gamma, Helm, Johnson, and Vlissides 1995). The purpose of this automatic registration mechanism is to minimize the amount of coding. For this purpose iGeo's internal server also manages objects to display on 3D multiviews in different shading modes and to execute agent rules by invoking interact and update methods every time frame. It also imports and exports geometry data from/to external files. Otherwise, a user has to write codes to prepare variables to store objects and manage them to perform the functions listed above.

CONCISE CODING AND METHOD CHAINING

To minimize the amount of coding, the API of the iGeo library is designed to have a concise class, method, property name, and also to enable method chaining in many methods, especially in transformation methods in geometry classes. Method chaining is enabled by letting the method return the object, which the method belongs to. By chaining these methods, the code can be reduced by removing intermediate variables and the statement punctuation (Algorithm 3). However, overuse of method chaining is known to make codes unorganized and difficult to read. iGeo has a policy to provide as many flexible options as possible and to trust users to choose the best option. Due to this policy, the access control of most classes and members in iGeo is set to "public" even when a user can destroy internal data consistency by careless modification.

Algorithm 3: Example of the longer and shorter way of coding with method chaining in *iGeo*

```
// longer way of coding.
// for vector transformation
IVec vec = new IVec(x,y,z);
vec.add(vec2); // vector add
vec.rot(axis,angle); // rotation
vec.mul(factor); // scaling

// for surface creation
ISurface srf =
new ISurface(points);
srf.mv(vec3); // translating
srf.rot(axis,angle); // rotation
srf.clr(r,g,b); // setting color

// for particle update by force
frc.mul(frameSec); // force
frc.div(mass);
vel.add(frc); // velocity
vel.mul(1-fric);
IVec tmpVel = vel.cp();
tmpVel.mul(frameSec);

pos.add(tmpVel); //position   // shorter way of coding with method chaining

// for vector transformation with static IG method
IVec vec = IG.v(x,y,z).add(vec2).rot(axis,angle).mul(factor);

// for surface creation with static IG method
ISurface srf = IG.srf(points).mv(vec3).rot(axis,angle).clr(r,g,b);

// for particle update by force with the use of scale-add
pos.add(vel.add(frc,frameSec/mass).mul(1-fric),frameSec));
```

NON-GEOMETRIC INFORMATION AND FILE I/O FOR PRACTICAL USE OF MODELING DATA

To be practical for geometry analysis and visualization in construction and digital fabrication, non-geometrical information can be generated in iGeo as text objects, layers, colors, and material attributes, and those can be exported into external files in Rhinoceros file format version 4 (.3dm) or Adobe Illustrator file format version 7 (.ai). A user can write an algorithm to measure, analyze, count geometry, add text labels in 3D spaces, color-code geometric objects, and categorize them into layers (Figure 2). An algorithm to produce a diagram with unfolded shapes and a count of identical shapes can be developed with these information and transformation operations, which will be shown later (Figure 6).

IMPLEMENTATION OF THE LIBRARY

The internal geometry data organization of iGeo was originally inspired by the file specification of Wavefront OBJ version 3 (Alias Wavefront 1997). The development and implementation of file import and export functions from/to an OBJ file also follows the specification. The Rhino 3DM file version 4 file import and export functions are ported and modified from parts of the C++ implementation in the open source project openNURBS (McNeel and Assoc. 2011). For the Adobe Illustrator file format, iGeo's implementation follows the file format specification in version 7 (Adobe Systems Inc. 1998). The solar path simulation algorithm is ported and modified from parts of the open source codes of NASA's IDL Astronomy User's Library (NASA 1987). The implemented library and the source codes are published at the iGeo project webpage (Sugihara 2011b).

ALGORITHM RESULTS DEVELOPED WITH IGEO

The results of iGeo as an algorithm development environment are described and evaluated by algorithms developed with the iGeo library for various projects in the following sections.

FOLDING ALGORITHM FOR THE COURTYARD FACADE AT EMERSON COLLEGE

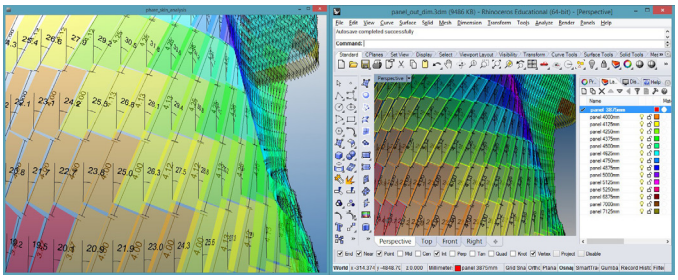
A folding algorithm with constraints in an unfolded shape and spanning distance was developed to design pre-rationalized facade panels for the courtyard of Emerson College Los Angeles (Figure 3). By designing the folding algorithm, it can generate a continuous range of 3D folded forms that can be produced in sheet metal fabrication with the construction constraints satisfied. This helps a designer explore geometric design possibilities to find better panel shapes. The continuity guarantees the consistency in the facade pattern when variations of panel shapes and folding are limited and discretized for rational panel modules.

PANEL FORMATION ALGORITHM BY FORCE FIELDS FOR THE COURTYARD FACADE AT EMERSON COLLAGE LOS ANGELES CENTER

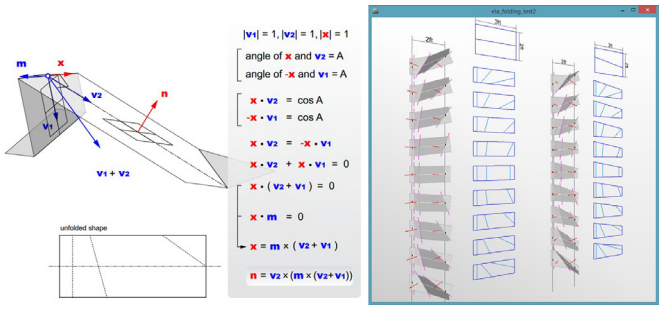
A panel formation algorithm was developed by particle and force field agents to have flexible control over the formation of pre-rationalized folded panels (Figure 4). The panel formation can be controlled by this flexible algorithm that can accept multiple soft constraints to satisfy constraints of view transparency in different areas, density limits for smoke evacuation, and geometric field response to the central building.

TENSILE NETWORK ALGORITHM FOR MESH RELAXATION OF DIAGRID STRUCTURE IN PHARE TOWER

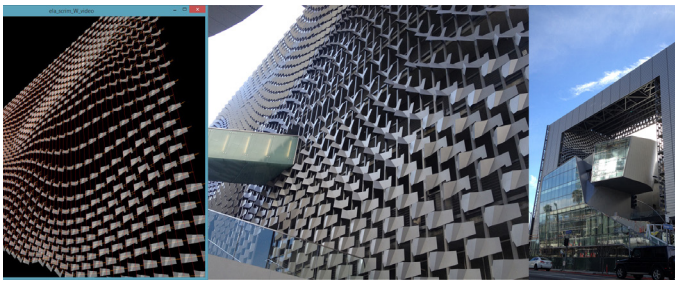
A particle agent algorithm responding to tension and straightening force agents with a constraint of staying on each floor edge curve was developed to design continuous and smooth diagrid structure on irregular boundaries (Figure 5). Through iterations in time, these agents can minimize kink angles at each node, maximize smoothness in a diagrid connection and satisfy the irregular boundary conditions simultaneously (Sugihara and Mayne 2013).



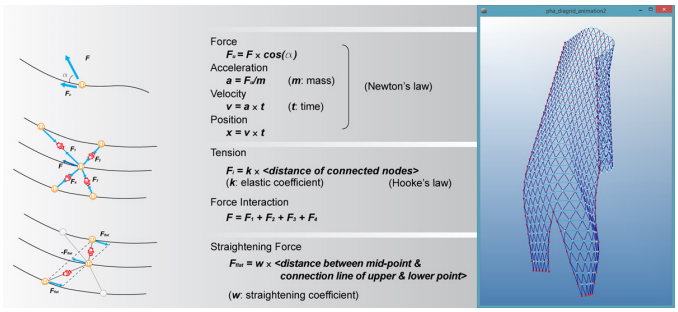
2 Example of text labeling, color-coding, and layer categorization executed in iGeo and exported to Rhinoceros via 3dm file



3 Folding algorithm producing a range of 3D shapes from rational 2D unfolded shapes



4 Formation of folded panels at Emerson College Los Angeles Center. (The architectural design copyright belongs to Morphosis. Computational design performed by the author)



5 Mesh relaxation algorithm by tension for diagrid. (The architectural design copyright belongs to Morphosis. Computational design is performed by the author.

GEOMETRY ANALYSIS AND VISUALIZATION ALGORITHM FOR GLAZING PANEL RATIONALIZATION IN PHARE TOWER

A geometry analysis algorithm was developed to categorize identical panel shapes and to calculate panel type counts (Figure 6). The rationalization process is performed manually, but the algorithm can show the panel types and counts immediately with color-coding and layer organization, and it gives feedback to the designer about the current rationality. An unfolded diagram is also generated by the algorithm and helps communication with fabricators and contractors so they have a better understanding of the geometry to help in finding potential issues and cost implications for the construction.

SOLAR OPTIMIZATION ALGORITHM AND VISUALIZATION FOR SOLAR SHADING SKINS OF PHARE TOWER

The solar optimization algorithm takes two inputs: a solar angle at a specific orientation calculated by a solar simulator in *iGeo* and the panel axis. Then it generates a panel geometry to maximize the cast shadow area on each panel (Figure 7). By optimizing each panel individually, it minimizes unnecessary closure of the skin to maximize the unblocked view area to the outside without sacrificing performance. A solar optimization diagram is also generated showing what date and time each panel is optimized for.

INTERACTIVE LIGHTING AGENT ALGORITHM FOR PHARE TOWER

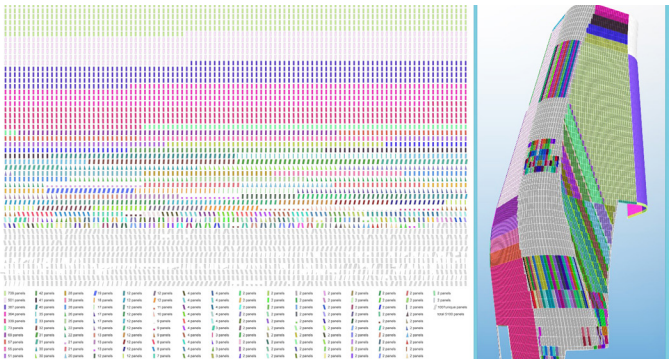
A panel geometry was activated as an agent making connection with neighboring panels, and each agent passes a light signal to the next panel. Agents on the edge monitor a microphone input, and they emit light signals according to the sound level (Figure 8). Due to the integration of geometry modeling and agent design, the lighting design process can happen simultaneously with facade panel design process, which results in the strong design consistency in space and time. Processing's extensibility also makes it easy to take real-time inputs for the interaction.

PANELIZATION AND DYNAMIC LIGHTING SIMULATION AGENT ALGORITHM FOR CEILING DESIGN

A panelization algorithm was developed to generate rationalized panel geometry together with a lighting agent on each panel that simulates dynamic lighting patterns (Figure 9). With this integrated algorithm development, the design of ceiling and dynamic LED color lighting can be developed simultaneously to give constant feedback to both design processes.

PRE-RATIONALIZED PANEL FITTING ALGORITHM FOR CLYDE FRAZIER'S WINE AND DINE

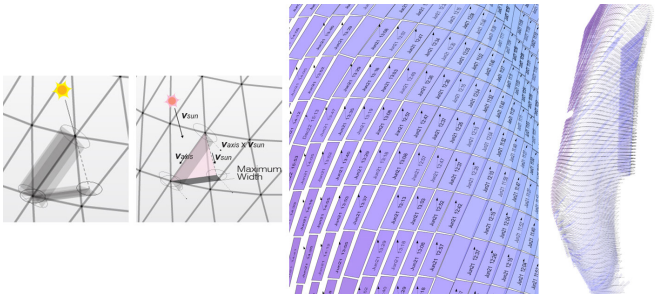
A panel fitting algorithm was developed to fit pre-rationalized panel modules on an organic ceiling form. Each panel installation boundary is unique and irregular, and the algorithm searches for the best fitting panel type, position, and orientation (Figure 10). Once the fitting process is established, the fitting result gives feedback to revise the further pre-rationalization. The repetition of this feedback loop makes the design more cost effective without sacrificing final architectural quality.



6 Geometry analysis and unfolded diagram of Phare Tower glazing panels with color-coding and panel counts

TENSILE NETWORK ALGORITHM FOR MEMBRANE DESIGN IN LOBBY DESIGN COMPETITION

A tensile network algorithm was developed to design the geometry of tensile membrane materials supported by steel rings (Figure 11). Once the tensile network algorithm is set to generate membrane shape automatically from the ring structure, the rings become the control geometry for the design. This makes it easier to adjust the location of rings to coordinate with other building structure, and the designer can focus on the topological connection and relationship to other architectural components in the lobby.



7 Diagrams of solar optimization algorithm for Phare tower facade

SWARM ALGORITHM FOR SELF-ORGANIZED FORMATION

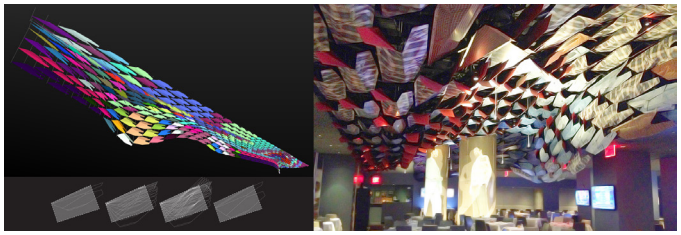
Swarm behaviors to create a self-organized spatial formation were explored in the workshop at Woodbury University in 2012. Starting from a conventional swarm behavior, various interaction rules and geometry generation rules were added (Figure 12). By using swarm agents in *iGeo*, workshop participants focused on the development of geometry generation including proximity-based line generation and different types of interactions such as the one between current agents or between current and past agents.

8 Dynamic lighting simulation for Phare tower

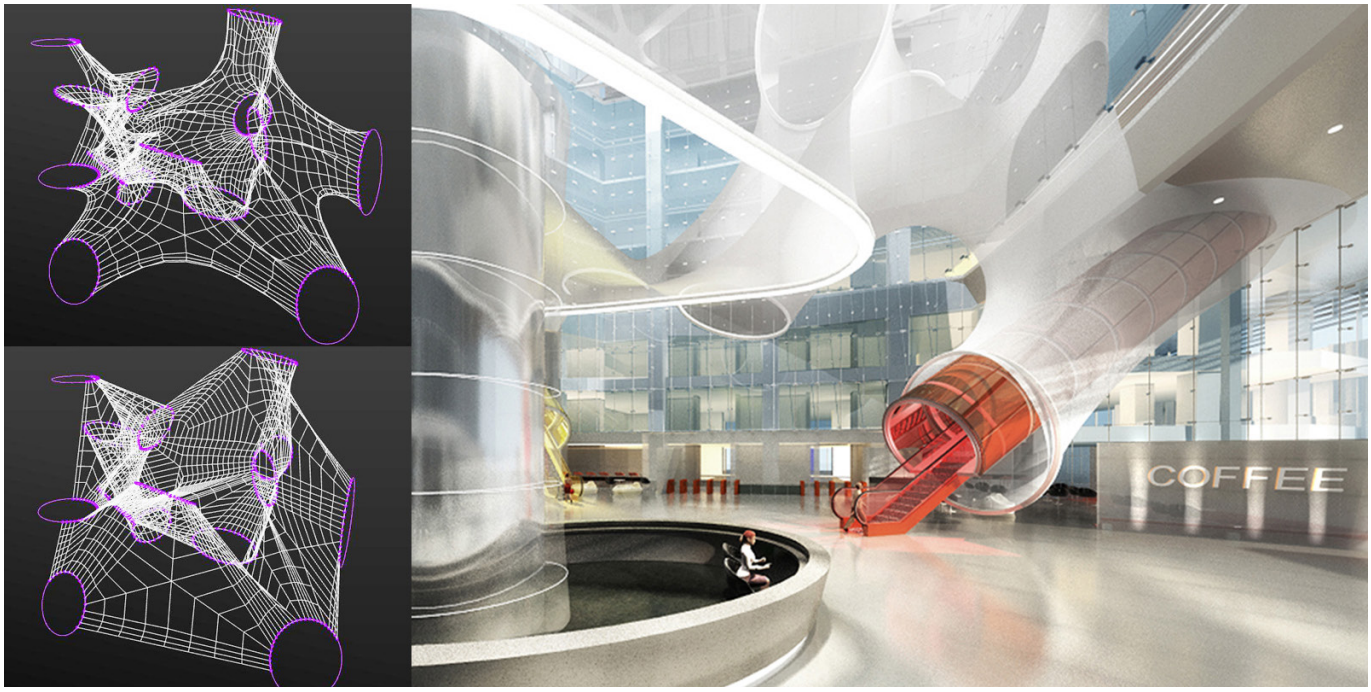
BIOLOGICAL FORM GROWTH SIMULATION BY COMBINING PREDEFINED AGENT ALGORITHMS

Swarm, branching, and gravity algorithms are combined to simulate the growth of mushroom gills (Figure 13). Swarm agents are emitted radially and fill the surfaces between adjacent agents. The control points of the surfaces are replaced by particle agents, and they sag from gravity. Swarm agents create a branching child agent when the gap between adjacent agents is too large. This shows that some simple agents such as swarm, particle, tension, and force fields can become building blocks to build complex agent behaviors. A new agent algorithm can be designed by assembling and connecting those predefined agents in *iGeo* and by adding rules to generate geometry through time.

9 Integrated panelization and lighting design for the International Medical Clinic, Shanghai. (Designed by collaboration with Kane AUD and ATLV. Color lighting consultation by Tsutomu Mutoh)



10 Panel fitting algorithm for Clyde Frazier's Wine and Dine. (Architectural design copyright belongs to Morphosis. Computational design performed by the author)



11 Membrane surfaces generated by the tensile network algorithm (designed by collaboration of Kane AUD and ATLV)

STRUCTURE SELF-OPTIMIZING ALGORITHM BY COMBINATION OF SWARM, TENSION, AND FORCE FIELD ALGORITHMS

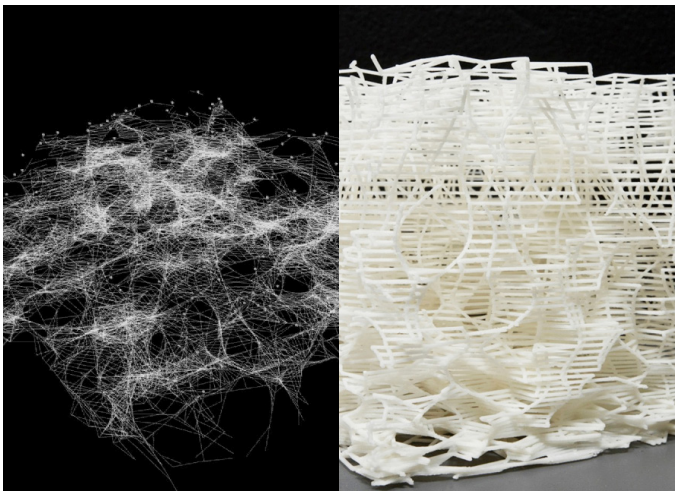
To build a self-organized formation with physical materials, an algorithm was developed by combining form-generating swarm, tension, compression, and gravity agents. In this algorithm, each agent optimize the structure and minimize the deformation and torsion under gravity (Figure 14). The algorithmically generated geometry was fabricated with fishing wire and acrylic sheets and assembled as a gallery installation "A(g)ntense". It proved the agent-generated structure worked in real gravity as well as it did in the simulation.

DISCUSSION

The algorithms above show that *iGeo* can help a designer develop a wide range of computational design algorithms efficiently by providing integrated tools in geometry and agent modeling and also practically by non-geometric information and its file I/O. To develop codes for rationalization with determinant algorithms, *iGeo* helps a designer with efficient coding with concise geometry API if the designer has a clear idea on the algorithm. Although *iGeo* provides low-level framework for geometry modeling with vector math and *NURBS* geometry, it does not provide generic determinant rationalization algorithms.

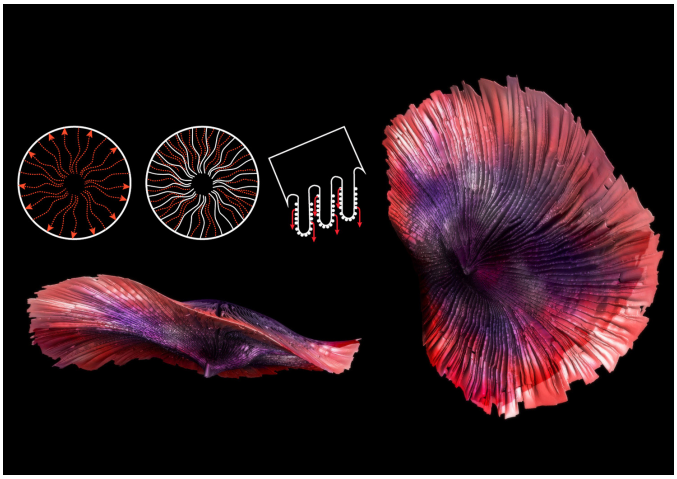
A designer has to write the whole algorithm on the lower-level geometry framework, because determinant rationalization algorithms tend to require precise understanding of geometric issues specific to a design project and to be difficult to be generalized. The case studies described in section *Folding Algorithm for the Courtyard Facade at Emerson Collage Los Angeles Center and Geometry Analysis and Visualization Algorithm for Glazing Panel Rationalization in Phare Tower* showed this issue and the algorithms for folding, unfolding and unit categorization had to be implemented from scratch to match with the specific design logic in the projects. Despite the difficulty, predefined algorithms for geometry rationalization are desired as a future work.

The library allows a designer to build a new agent-based algorithm by assembling predefined algorithms as building blocks. Currently, *iGeo* provides a simple agent framework and some predefined agents (particle, swarm, tension, and force fields). They are general enough to support development of a wide range of algorithms, but it is also true that there can be more specific algorithmic components as building blocks to make algorithm development more efficient. For example NetLogo (Tisue and Wilensky 2004) has an agent framework with four distinct types of agents: turtle (an autonomous agent with locations), patches (a pixelized or voxelized spatial environment), links (a connection agent of two turtles), and observer (a global agent without location).
These agents can be developed under *iGeo*'s general agent framework, but it would be beneficial to provide more specific types of agents as predefined building blocks.

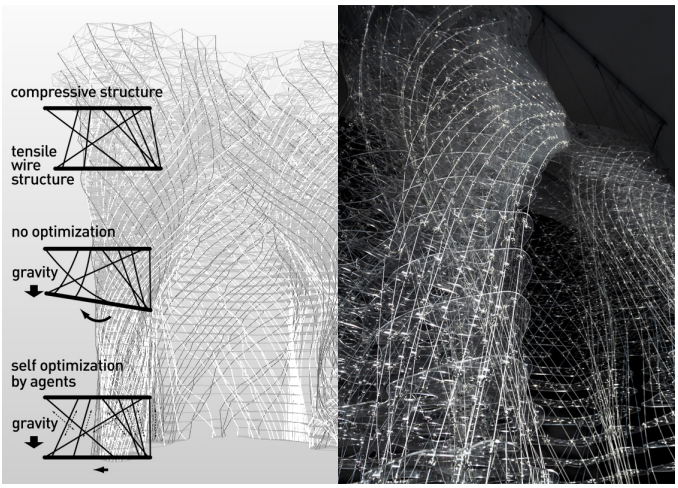


12 Swarm formation algorithm and the 3D printed result (student work at Woodbury University by Conner Macphee and Anali Gharakhani, photo by Taiyo Watanabe)

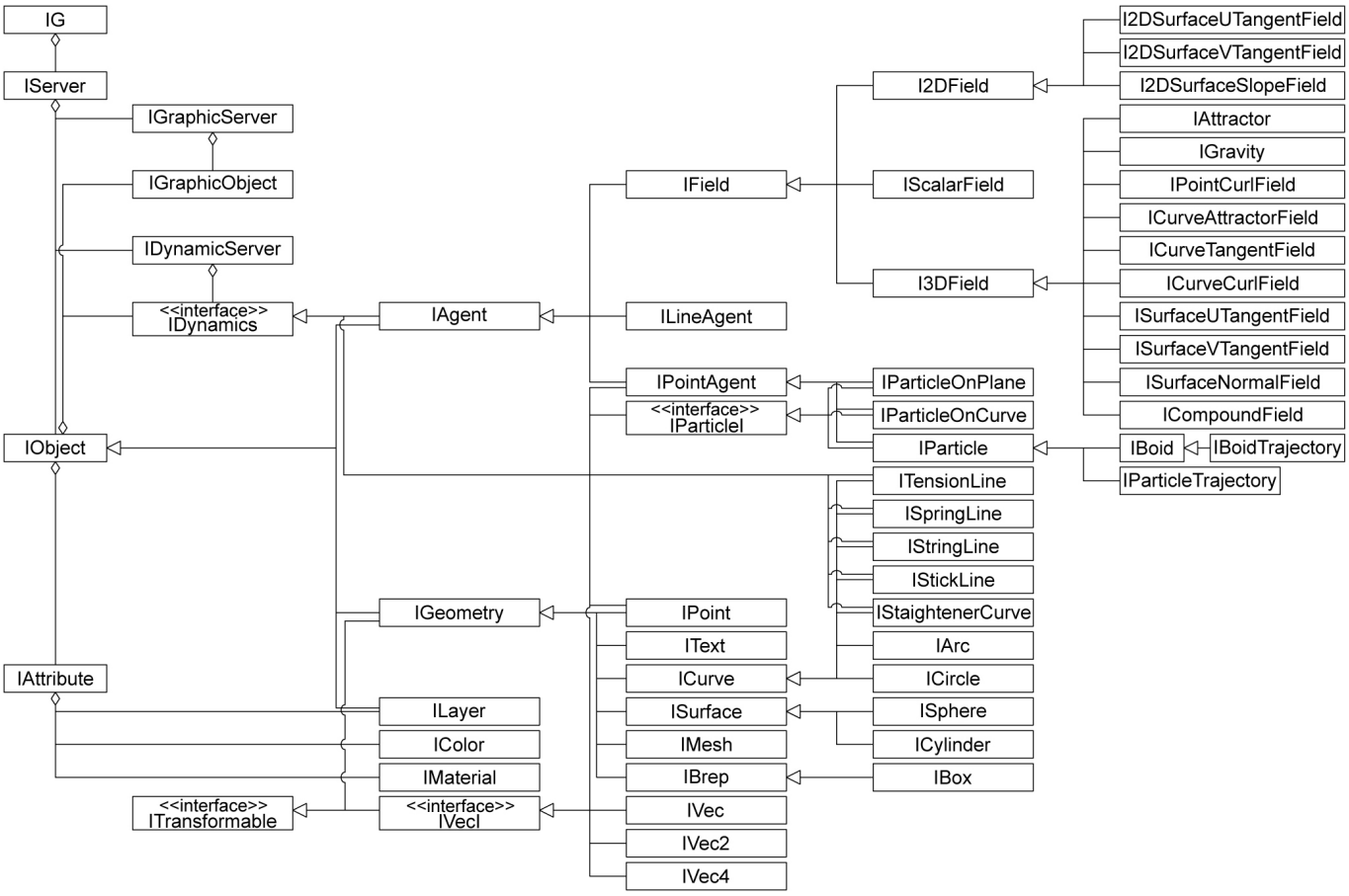
In comparison to existing geometry libraries in Processing, *iGeo* shows some advantages and disadvantages. Toxiclibs is a popular geometry library in Processing (Schmidt 2011). In geometry modeling, toxiclibs has a more powerful capability in polygon mesh modeling including subdivision operations whereas *iGeo* has a focus on *NURBS* geometry. For agent-based algorithms, toxiclibs provides a particle-based physics engine with a mechanism to customize particle behaviors by constraints whereas *iGeo* realizes physics simulation within the broader agent framework through a distributed agent simulation. Whereas the centralized physics engine provides more precise and powerful physics simulation including fluid dynamics and cloth simulations, the distributed agents with hierarchical class structure under the agent framework (see Appendix) provide more interoperability and extensibility. Features equivalent to the data management mechanism described in section *Geometry and Agent Object Management by an Internal Server and 3D navigation described in section Processing as a Host Coding Environment* are not provided by toxiclibs. Instead, it relies on Processing's original capabilities and convention. These features in *iGeo* make coding easier for coding beginners, although, coders who already have experience in Processing could get confused with the different convention. The geometry library ANAR+ provides geometry modeling and agent-based modeling capability in Processing (Labelle et al. 2010). Geometry modeling in ANAR+ has a focus on parametric relationship, which can be adjusted during the execution of codes, whereas *iGeo* does not generate parametric relationship automatically to keep data light and to achieve higher complexity. ANAR+ provides agent-based modeling capability by the similar API with NetLogo whereas *iGeo* has its own way to integrate agent types under the agent framework and provides various predefined agent classes. ANAR+ has an advantage in file export by writing codes for other modeling software including Rhino VB script, AutoLisp, MEL script and SketchUp Ruby.



13 Mushroom growth simulation algorithm by swarm, branching, and gravity force field algorithms (Student work at SCI-Arc by Koho Lin)



14 Structure self-optimizing algorithm and completed installation A(g)ntense for Blindspot initiatives group exhibition in 2014 (photo by Taiyo Watanabe)



Appendix UML Class Inheritance and Aggregation Diagram of Major Classes in iGeo

This can be powerful when it can contain parametric relationship. These advantages of other libraries including more powerful polygon mesh modeling, physics simulation and file I/O are desired to be implemented in *iGeo* and left as future works.

CONCLUSIONS AND FUTURE WORK

To help computational designers develop algorithm codes, *iGeo* is designed as an algorithm development environment with *NURBS* geometry and agent-based modeling capability. The paper describes the algorithms developed on *iGeo* for various projects as case studies and they show the efficiency and the effectiveness of the environment. They also show the efficiency in the development of custom agent-based algorithms achieved by predefined agent classes as building blocks. We can say the foundation of the algorithm development environment is set, and predefined algorithms help designers develop new algorithms efficiently and effectively. Further development of predefined algorithmic building blocks for agents and rationalization, more powerful physics simulation, polygon mesh modeling, and file I/O remains as a future challenge for *iGeo* to provide higher efficiency and effectiveness for computational designers.

REFERENCES

Adobe Systems Inc. 1998. Adobe Illustrator File Format Specification. http://partners.adobe.com/public/developer/en/illustrator/sdk/AI7FileFormat.pdf

Alias Wavefront. 1997. Advanced Visualizer Object Format Specification. http://www.martinreddy.net/gfx/3d/OBJ.spec

Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1995. Design Patterns. Boston: Addison-Wesley Publishing Company, Inc.

Labelle, G., J. Nembrini, and J. Huang. 2010. Geometric Programming Framework: ANAR+: Geometry library for Processing. In Future cities: 28th eCAADe Conference Proceeding, 403-410. Zurich, Switzerland: ETH Zurich.

McNeel and Associates. 2010. openNURBS SDK. http://www.rhino3d.com/openNURBS

NASA. 1997. IDL Astronomy User's Library. http://idlastro.gsfc.nasa.gov/contents.html

Reas, C. and B. Fry. 2007. Processing: A Programming Handbook for Visual Designers and Artists. Cambridge: MIT Press.

Schmidt, K. 2011. Toxiclibs. http://toxiclibs.org/

Sugihara, S. 2011a. Comparison between Top-Down and Bottom-Up Algorithms in Computational Design Practice. In Proceeding of the International Symposium on Algorithmic Design for Architecture and Urban Design. Tokyo, Japan: ALGODE 2011

Sugihara, S. 2011b. iGeo: Computational Design Library. http://iGeo.jp

Sugihara, S. and T. Mayne. 2013. Irregularity and Rationality Mediated by Agents: Modeling process of Phare tower. In Architecture in Formation, ed. P. Lorenzo-Eiroa and A. Sprecher, 254-257. New York: Routledge.

Tisue, S. and U. Wilensky. 2004. NetLogo: Design and implementation of a multi-agent modeling environment. In Proceedings of Agent 2004 Conference, Chicago: Agent 2004.

IMAGES CREDITS

Figure 1. Image created by the author.

Figure 2. Ibid.

Figure 3. Morphosis Architects (2014) Emerson College Los Angeles. Los Angeles. Image by the author.

Figure 4. Morphosis Architects (2014) Emerson College Los Angeles. Los Angeles. Right photo by Shanna Yates, others by the author.

Figure 5. Morphosis Architects (2010) Phare Tower. Paris, France. Image by the author.

Figure 6. Ibid.

Figure 7. Ibid.

Figure 8. Ibid.

Figure 9. Kane AUD and ATLV (2012) Lobby design for the International Medical Clinic. Shanghai, China.

Figure 10. Morphosis Architects (2012) Clyde Frazier's Wine and Dine. New York. Images by the author.

Figure 11. Kane AUD and ATLV (2012) A proposeal for a lobby design competition. Shenzhen, China.

Figure 12. Macphee, Conner and Gharakhani, Anali (2012), Swarm work for iGeo workshop. Woodbury University Gallery, Los Angeles. Photo by Taiyo Watanabe.

Figure 13. Lin, Koho (2013) Research for Coding Form Seminar. SCI-Arc, Los Angeles.

Figure 14. Sugihara, Satoru (2014) A(g)ntense. Keystone Gallery, Los Angeles. Photo by Taiyo Watanabe.

SATORU SUGIHARA is a principal and founder at the computational design firm ATLV founded in 2012. Prior to his firm, he worked at Morphosis as a computational designer, at DR_D and Greg Lynn FORM as an architectural designer, and at IMRF in Tokyo as a researcher in media art. He is a faculty at SCI-Arc teaching computational design seminars. He taught computational design at the University of British Columbia, Paris-Malaquais, Woodbury University and Tokyo University of the Arts. Satoru Sugihara received his M.S. in computer science from Tokyo Institute of Technology in 2001 and his M.Arch. from UCLA in 2006.