Ecole Nationale Supérieure d'Architecture Paris-Malaquais

SEMINAIRE DE MASTER "La Connaissance Computationnelle"

Sylvain Usai

Architecture scriptée

L'application de systèmes multi-agents à la conception architecturale

Mémoire de Master sous la direction de Christian Girard / département *Digital Knowlege* / Février 2014

Ce que l'on désigne par *Scripting* en architecture qualifie en fait une multitude d'outils différents. Qu'est-ce que l'algorithmique, le paramétrique, le code, le génératif en architecture ? Un style ? Un Processus de conception ? Le lexique de l'architecture computationnelle est soit trop (quantitativement) soit trop peu (qualitativement) défini et la compréhension de termes clés peut différer d'un auteur à l'autre. David Rutten, concepteur et principal développeur du logiciel largement considéré *paramétrique* Grasshopper3d[1] a donné sa version du lexique computationnel à l'occasion d'une question ouverte[2]. Le paramétrique y est défini par «*Make shapes with the mouse or the keyboard. Processes are recorded and can be tweaked, results are recomputed*» (Rutten, 2013) il ajoute, «Parametric has been hijacked, it clearly doesn›t mean ‹parametric› in the mathematical sense any more. In effect, everything a computer does is parametric». Grasshopper3d tiendrais plutôt de l'*Algorithmique :* «*Quite similar to Parametric* (and a lot of overlap with *Computational* too). However the process is more central. Rather than the computer recording (and replaying) what you do with the mouse, you directly create the algorithm yourself. I would say this is the best description of what Grasshopper is»[3] (Rutten, 2013). (le lexique mentionné comporte la définition des termes *Manual/CAD, Parametrique, Algorithmique, Génératif et Computationnel,* lien en note).

Ce lexique, bien que de première main, n'est pas forcément le plus partagé. Notons d'ailleurs que la définition d'*algorithmique,* censée qualifier un logiciel considéré *paramétrique* reprend les grandes lignes du design *génératif* par Hartmut Bohnacker: « *dessiner un processus plutôt qu'un objet*[4]» (Bohnacker, 2009)

Pour notre part, nous retiendrons deux termes : *paramétrique* (souvent couplé avec *associatif*) parce qu'il incarne la vitrine de l'architecture dite 'digitale' et *génératif* (assimilé aux termes *itératif* et émergence), parfois posé en réaction aux pratiques paramétriques comme une architecture digitale de deuxième génération. Ces deux approches nous intéressent dans la mesure où elles n'agissent pas comme un transfert des procédures de conception depuis l'analogique vers le numérique mais participent pleinement à la conception.

Dans le but d'étudier les relations entre paramétrique et génératif, nous nous intéresserons à leurs support commun, le Scripting, et plus particulièrement la programmation orientée objet. Enfin, on s'intéressera à l'usage de la simulation et de la fabrication digitale comme stratégies de passage d'un espace virtuel à un espace réel et matériel.

Le but de la recherche est de définir une pratique de l'architecture algorithmique contemporaine entre abstraction du design en tant qu'intention architecturale et exactitude computationnelle.

---

1 Grasshopper3d, plug-in pour Rhinoceros édité par McNeel.

2 '*Generative vs Paramétric Modeling'* http://www.grasshopper3d.com/forum/topics/generative-vs-parametric-modeling le 6 septembre 2013. Notons qu'il s'agit d'une réponse « à chaud » à une question directe et non un article ou un essai théorique.

3 **Rutten David**, '*Generative vs Paramétric Modeling', id.* « Toutes production dessinée à la souris ou au clavier, le processus est enregistré et peut être modifié à posteriori, le resultat est recalculé à chaque modifications […] Le terme paramétrique à été détourné et ne signifie plus 'paramétrique ' au sens mathématique. Tout ce que fait un ordinateur est paramétrique […] [l'Algorithmique est] assez similaire au paramétrique (et se recoupe avec computationnel), toutefois, l'intérêt est plus porté sur le processus. Plutôt que d'enregistrer et de rejouer les actions de la souris, l'algorithme est créé par l'utilisateur. Je pense que c'est le terme qui défini le mieux Grasshopper » (notre traduction).

4 **Laub Julia, Bohnacker Hartmut, Groß Benedikt, Lazzeroni Claudius**, '*Design Génératif : Concevoir Programmer Visualiser'*, éditions PYRAMID, Paris, 2010,. p.5. titre original *'Generative gestaltung*', Hermann Schmidt Mainz, 2009.

I. Architecture générative

i. genèse

En introduction, nous avons défini deux approches de l'architecture « digitale » : paramétrique et génératif.

Le paramétrique, tel que nous l'entendons, fait référence au domaine de l'animation et permet, par variation et différenciation d'une même matrice de paramètres, la déclinaison d'objets d'une même famille : c'est *l'objectile* de Bernard Cache[5]. L'instanciation d'un détail constructif sur une grille tridimensionnelle non orthogonale en est un exemple. Les objets géométriques ne sont plus définit de manière absolue par des longueurs fixes mais relativement entre eux si bien que la transformation d'un objet entraine la transformation de tous les objets auquel il est lié par une relation. Le paramétrique peut être considéré comme une technique de CAD (le 'D' valant ici pour 'Drafting' plus que 'Design') avancée puisqu'il s'agit dans un premier temps d'une automatisation du dessin et de ses mises à jour, on parlera dans ce cas de modèles paramétriques 'clos' (en opposition aux modèles paramétriques 'ouverts' qu'on définira ultérieurement). Le potentiel de production de variation des modèles paramétriques est exploité en tant que processus de conception à la fois à une échelle globale pour une exploration formelle et à une échelle locale pour la différentiation d'éléments de détails. Ce double usage 'design/dessin' est symptomatique de l'usage de l'ordinateur en architecture[6] : une application à des sous-disciplines techniques de l'architecture (dessin, structure, etc.) peu à peu transformée et intégrée en tant qu'intention architecturale.

Le génératif appartient au domaine de la simulation. Il permet notamment de travailler au sein de systèmes non linéaires – c'est à dire ne pouvant pas être soumis à une résolution analytique. L'interaction locale d'éléments avec leur voisinage suivant un nombre limité de règles simples conduit à l'émergence de comportements globaux. Le terme émergence a été popularisé par John Henry Holland avec la théorie des systèmes complexes dans son ouvrage « *emergence* : *from chaos to order* » en 1998[7] et présente des applications en biologies, sociologie, physique… En architecture, art et design, les termes génératifs et émergence sont intimement liés et souvent confondus. Il faut néanmoins les distinguer des approches aléatoires ou utilisant le perlin noise[8] : c'est bien la complexité du système à échelle macro qui rend imprévisible son comportement pour l'œil humain et non le recours à une fonction aléatoire, la répétabilité n'est donc pas exclue.

Son utilisation par les simulations multi-agents, même très simples comme la relaxation dynamique ou le sphère packing, permet la distribution de la complexité au sein du système.

Cette tension entre associatif et itératif ou paramétrique et génératif est récurrente (même si, on le verra, les deux ne sont pas mutuellement exclusifs), surtout parmi les adeptes de la seconde qui tentent de faire entendre leurs voix dans le monde de l'architecture digitale post- Schumacher : « *In practice, when we look at computation, there have been two schools of thought. Both have emerged from a fundamental difference of control – parametric and generative – and they reinforce the top-down/bottom-up discussion in the most primitive form*[9] » (Spyropoulos, 2013).

L'environnement digital, envisagé comme une écologie *de l'information*[10], permet de considérer le projet comme un système interconnecté à l'image du modèle du treillis imaginé par *Christopher Alexander*[11]. Une prise de décision à l'intérieur de ce système (l'attribution d'une valeur à une variable) peut être motivée par le système lui-même et les informations qu'il fournit par associativité. L'image de ce système interconnecté est au cœur de la notion de contrôle évoquée par Théodore Spyropoulos et renvoie à la prise de décision, à

5 **Cache Bernard**, '*Terre Meuble'*, editions HYX, Orléans, 1997.

6 **Burry Mark**, *contextual summury of computing, scripting and speculative design* in *'Scripting Cultures',* Wiley Press, Londres, 2011. pp 13-27, notamment la p.17.

7 Holland John, *'emergence : from chaos to order',* première edition par Addison-Wesley, Redwood City, 1998.

8 Perlin Noise sous Processing 2.0 : http://processing.org/reference/noise_.html

9 **Spyropoulos Theodore**, *John Henry Holland and Theodore Spyropoulos in conversation* in *'Constructing Adaptive Ecologies : Correlated systems of living'*, Architectural Association Publisher, Londres, juin 2013
«En pratique, deux écoles de pensée coexistent au sein de la computation, toutes deux issues d'une différence fondamentale vis-à-vis du contrôle – paramétrique et génératif- et elles entretiennent le débat top-down/bottom-up » (notre traduction).

10 **Andrasek Alisa**, '*Biothing, matérialité vitale'*, HYX, Orléans, 2009.

11 **Alexander Christopher**, *A city is not a tree* publication originale in '*Architectural Forum',* vol122 n°01 Avril 1965 pp.58-62 pour la partie 1 et vol122 n°02 Mai 1965 pp.58-62 pour la partie 2. Disponible en ligne : http://www.abc.polimi.it/fileadmin/docenti/TEPAC/2012/FONTANA/A_City_is_not_a_Tree.pdf (consulté le 6déc2013).

l'évaluation ou à l'optimisation de ce système.

La principale critique des travaux paramétriques tiens dans leurs échec à exploiter de tels systèmes et s'en tiennent à une prise de décision arbitraire soutenue par la variation :

«*Due to the inherent specificity of computational complexity or the desire for visual unifying consistency, parametric design typically reduces the number of formal variables, but maximizes their variability through transformational affects which are engendered via quantity* [12]» (Meredith, 2009).

Du fait de cette réduction de la complexité, la variété produite par un modèle paramétrique clos n'est souvent qu'une variété anticipée :

«Parametric models used in practice have a propensity for inflexibility, which sometimes leads the model to break […] In the worst case, the designer is dissuaded from making the change and ends up with a design that was not so much created in a parametric model as it was created for the limitations of a parametric model[13]» (Davis, 2013).

Ou, formulé différemment par Roland Snooks :

« *Parametric models are structured hierarchically, however, having direct cascading, causal relationships – an obvious impediment to this description of generative design [designing process rather than artifact]. The parameters within these models -the ubiquitous sliders in software programs epitomize- confine the model to a known set of limits. So while parametric models enable a distribution of difference, this is not the difference that emerges from intensive processes, but rather a directly described, top-down, smooth gradient operating within a predefined range. Here, all possibility is already given within the starting condition*[14] » (Snooks, 2012).

Plutôt qu'une distribution de la différence s'impose alors une distribution de la complexité :

«*To the extent the profession has utilized parametric today; there is very little instigating complexity other than a mind-numbing image of complexity, falling far short of its rich potential to correlate multivalent processes or typological transformations, parallel meanings, complex functional requirements, site-specific problems or collaborative networks*[15]» (Meredith, 2009).

On retrouve dans la mise en place de cette distribution de la complexité la dualité 'dessin/design' ou 'optimisation/spéculation', réinterprétée dans le contexte de l'architecture générative et exemplifiée par les travaux suivants :

- La recherche de performances multivalentes, obtenue par une sortie de l'auto référencement du modèle paramétrique avec une associativité ouverte aux 'paramètres extérieurs' (contraintes matérielles, environnementales). Cette *paramétrisation ouverte* intégrant à la fois la prise de décision et l'évaluation dans une logique de feedback rétroactif. Les travaux les plus représentatifs sont ceux

12 **Meredith Michael**, *Never enough (transform, repeat ad nausea)* in *'From Control to Design, parametric/algorithmic architecture'* Actar-D, Barcelone, 2008, p. 6.
«En raison de la précision inhérente à la complexité computationnelle ou par désir d'homogénéité visuelle, le design paramétrique réduit souvent le nombre de variables décisionnaires mais maximise leurs applications, l'esthétique étant produite par la variation et la quantité » (notre traduction)

13 **Davis Daniel**, *Problems with Flexibility* in '*Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture'*, thèse soutenue au RMIT School of Architecture and design, Melbourne, Février 2013 p.6.
«Les modèles paramétriques utilisés en pratique sont peu robustes (inflexibles), ce qui peut conduire à une interruption du modèle […] Dans le pire des cas, le manque de robustesse dissuade le designer d'opérer les changements voulus et le projet n'est plus tant crée par le modèle paramétrique que par ses limites » (notre traduction)

14 **Snooks Roland**, *Volatile Formation* in *'Reclaim Resi[lience]stance // ……..R²'* Log°25, AnyCorporation, été 2012. P.56.
« Les modèles paramétrique sont structurés hiérarchiquement, avec une relation directe de cause à effet -incompatible avec cette description du design génératif [designer le processus plutôt que l'objet]. Les paramètres de ces modèles -incarné par l'omniprésent 'slider'- confine le modèle à un domain connu. Alors que les modèles paramétriques permettent une distribution de la différence, cette différence n'émérge pas du processus lui-même mais plutôt d'un gradient explicite, top-down, opérant dans un domaine pré-défini. Toutes les possibilitéés d'évolution sont données dans les conditions initiales » (notre traduction).

15 **Meredith Michael**, *Never enough (transform, repeat ad nausea)* in *'From Control to Design, parametric/algorithmic architecture'* Actar-D, Barcelone, 2008, p. 7.
« A ce jour, l'utilisation du paramétrisme par la profession architecturale est très souvent limité à une image illusoire de la complexité, bien loin d'exploiter leur potentiel à traiter en tant qu'ensemble des processus polyvalents, des transformations topologiques, des notions parallèles, des besoins complexes, des problèmes inhérents au site ou des réseaux collaboratifs » (notre traduction)

menés sur le '*material system*[16]' par Achim Menges.

- L'utilisation d'une instabilité générative propre aux systèmes émergents, c'est-à-dire l'encodage d'intentions architecturales à une échelle locale et l'observation de ce système à une échelle globale. Les travaux de Roland Snooks et Robert Stuart-Smith au sein du studio de recherche *Kokkugia* en sont un exemple.

La majeure partie de la production architecturale computationnelle contemporaine se situe entre ces deux extrêmes, composant librement avec ceux-ci. La question du contrôle est abordée différemment par les deux pôles. Si une approche 'material system' est fondamentalement accès sur la performance, et donc la sélection de la variation, le choix des paramètres impliqués reste le fait du designer. La question de l'imitation exhaustive de la référence (le réel) ou de la simplification se pose alors :

« *When we code, we are implementing a possible universe, which can only resemble our own to the degree that we are able to describe it*[17]» (Collins&Hasegawa, 2009).

Les données reçues et le processus qui les traite doivent être d'une résolution et d'une précision suffisamment accrue. Dans le cas contraire, que penser de « *la pluri sensibilité au contexte et la circonspection qui caractérisent ces agents* [ici, les objets étudiés par la simulation] *et non l'approche brutale et phénoménologique par essai aléatoire et erreur corrigée*[18]» (Varenne, 2013) quand la définition de cette pluri sensibilité et de cette circonspection sont le fruit d'essais aléatoires et d'erreurs corrigées par le designer invalidant ainsi leur capacité à « *faire montre de vitalité, de biomorphisme, plus qu'une forme à l'allure globalement biomimétique*[19] » (Varenne, 2013).

Les travaux plus marqués par 'l'instabilité générative' sont moins atteint par la question de l'exhaustivité par rapport à une référence puisque leurs modèle est celui de la métaphore (débridée depuis Archilab 2013[20]) qualifiée par *'l'abstraction créative*[21]*'.* Il ne s'agit pas pour kokkugia d'imiter les formations naturelles mais d'explorer leur possible utilisation pour l'architecture, on est alors plus proche d'une approche de l'émergence par l'art génératif[22].

ii. De l'objet à l'agent

Franck Varenne décrit la simulation comme « *any computer treatement of either a mathematical model without analytical solution or a rules based inference motor – like cellular automata, multi-agent systems or object oriented modeling. The emphasis here is on the discretization and the step by step resolution*[23] ».

L'absence de solution analytique implique qu'une simulation n'est pas une résolution paramétrique (au sens mathématique). Les problèmes d'optimisation, notamment par algorithmes génétiques sont un exemple connu de système non linéaire utilisés en architecture, la résolution de ces systèmes se fait par itération successive et sélection du meilleur résultat rencontré (nous ne nous attarderons pas sur les algorithmes génétiques, le billet de David Rutten '*Evolutionary Principles applied to Problem Solving'*[24] et son article '*galapagos, on the logic and limitation of generic solvers'*[25] constituent une bonne entrée dans le sujet). Ce

16 **Hensel Michael & Menges Achim**, *'Morpho-Ecologies, toward heterogeneous space in architectural design'*, Architectural Association Publisher, Londres, Février 2007.

17 **Collins Mark & Hasegawa Toru**, rapporté par **Burry Mark** in '*Scripting Cultures'*, op.cit. p.60.
« le code permet la création d'un nouvel univers qui ne peux ressembler au notre que si on le décrit avec suffisamment de précision » (notre traduction).

18 **Varenne Frank**, *Le parti pris des choses computationnelles* in *'Naturaliser l'Architectur, Archilab'*, editions HYX, Orléans, 2013, p.102

19 **Varrenne Frank**, *Le parti pris des choses computationnelles, id.*

20 *Archilab 2013 : Naturaliser l'Architecture*, exposition aux Turbulences du Frac Centre, commissaires d'exposition : **Marie Ange Brayer & Frédéric Migayrou**, Orléans, 2013.

21 **Holland John,** *chap 8 metaphors* in *'emergence : from chaos to order',* Oxford University Press Oxford, 1998, page 375

22 **Pearson Matt**, *Generative Art*, Manning, New York, 2011.

23 **Varenne Frank**, *What does a computer simulation prove ?* in '*Simulation in Industry*' Proc of the 13th European Simulation Symposium, Marseille, France, October 18-20th, 2001, edité par N. Giambiasi et C. Frydman, SCS Europe Bvba, Ghent, 2001, p.550

«[simulation :] Tout traitement par ordinateur, soit d'un modèle mathématique sans solutions analytiques, soit d'un moteur décisionnaire régit par des règles –comme le sont les automates cellulaires, les systèmes multi-agents ou la modélisation orientée objet. Cette définition insiste sur la discrétisation du modèle et la résolution itérative » (notre traduction)

24 **Rutten David**, '*Evolutionary Principles applied to Problem Solving',* http://ieatbugsforbreakfast.wordpress.com/2011/03/04/epatps01/, visité le 6 janvier 2014

25 **Rutten David**, *galapagos, on the logic and limitation of generic solvers* in '*Computation Work, The Building Of An Algorithmique thought*' Architectural Design, Profile n°222, mars/avril 2013, Wiley Press, London. pp. 132-135

'meilleur résultat' est un optimum prédéfini qui peut se heurter aux mêmes limitations et critiques qu'un modèle paramétrique fermé précédemment évoqué (càd, confiner le modèle à un domaine connu et prédéfini) qui, si elles sont parfaitement acceptables pour un problème de pure optimisation (structurelle par exemple), peuvent être un obstacle à une recherche formelle.

Dans le cas des systèmes multi-agents, la notion d'optimum n'as pas lieu d'être puisque toutes les solutions obtenues sont des optimums vis-à-vis du système défini en cela qu'elles respectent et sont un produit des règles établies.

Le terme '*multi-agents*' défini un système constitués d'objets '*conscients*' de leurs environnements[26]. Ils peuvent interagir avec cet environnement et le modifier pour atteindre un but.

La paternité des algorithmes de types swarms (une sous famille d'algorithmes multi-agents) tels qu'ils sont utilisés par une population d'architectes, artistes et designers est souvent attribué à Craig Reynolds et à son Bird Like Droid en 1986. Notons également l'existence des travaux de Douglas Hofstadter sur les colonies de fourmis publiés en 1979 et plus souvent cité comme origine des systèmes swarms par un public scientifique. La différence entre ces deux travaux tient en ce qu'Hofstadter cherchais à expliquer biologiquement le déplacement des fourmis quand Reynolds était plus intéressé par la modélisation graphique de nuées d'oiseau.

Le Boid constitue en effet un bon exemple d'utilisation d'intelligence collective. Le mouvement de chaque Boid est conditionné par trois règles simples : éviter les chocs, aller dans la même direction que ses voisins proches et rester groupés. Le champ de vision d'un objet ne lui permet pas d'avoir connaissance du mouvement de la totalité du groupe, seulement de ses voisins proches, les règles sont donc appliquées à un niveau local. Ce n'est pas la sophistication de l'objet lui-même qui crée la complexité du comportement observé mais le nombre d'objets et leurs interactions : « *les mouvements de groupe ne requièrent pas de vision globale et n'ont pas besoin d'origines comportementales complexes, il suffit simplement de suivre ses voisins.*[27] »

La définition mathématique des règles de comportement donne à la simulation un caractère ascalaire : la structure d'un petit groupe de boids est la même que celle d'un grand groupe, les deux se comporteront de la même manière.

Si l'on isole la première règle du Boid de Reynolds (éviter les chocs), on obtient la définition d'un système d'auto organisation de type packing. La définition des relations de voisinage tient dans les lignes suivantes (en pseudo code) :

« Si la distance entre l'objet considéré et n'importe quel autre objet est inférieure au rayon de l'objet considéré ajouté à celui du rayon de l'objet en contact, alors, le mouvement de l'objet considéré est inversé »

Cette simple règle suffit virtuellement à donner une matérialité et à le rendre impénétrable par le reste de l'environnement. Pour des corps non circulaires, il conviendra d'appliquer le test suivant :

« Pour un contour fermé et un point, si la ligne partant définie par ce point et un autre point quelconque intersecte le contour un nombre impaire de fois, le point est à l'intérieur de l'objet et doit se déplacer »

Les règles qui régissent cette simulation sont toutes d'ordre géométrique (une distance avec les objets voisins ou un nombre d'intersections), cette traduction en terme d'opérations logiques, mathématiques et géométriques est nécessaire à l'encodage d'une intention.

De plus, on a précédemment évoqué la possibilité pour un agent de suivre un objectif. Du fait des interactions avec l'environnement et les autres agents, cet objectif local peut différer d'un objectif global (l'intention du designer). Cette négociation entre bottom-up et top-down est une des explorations du design génératif.

Cette négociation peut prendre la forme soit d'une transformation des règles locales (on pourra parler d'intention '*hardcoded*'[28]) soit d'une interaction dynamique avec le concepteur pendant la simulation, ce que Roland Snooks décrit comme un « *feedback of algorithmic systems and explicit decision – a form of messy computation.*[29] ». De manière plus pragmatique cette interaction homme/machine présente également l'im-

26 Encore une fois, les objets n'ont conscience de leurs environnement que dans la mesure où on les en informe et où l'on encode le processus de prise d'information, cf. note de bas de page n°17.

27 **Ball Phillip**, *Follow your neighbours* , in '*Flows: Nature's Patterns: A Tapestry in Three Parts*' - chapter 5, Oxford University Press, 2011. pp. 127-159 (notre traduction)

28 *'hardcoded'* par opposition à '*softcoded'*, se dit d'une variable qui n'a pas d'interface dynamique avec l'utilisateur

29 **Snooks Roland**, *Volatile Formation*, op. cit., p.55

mense avantage de ne pas trop s'attarder sur les *exceptions* durant l'encodage de la simulation, celles-ci pouvant être résolues manuellement durant la simulation. Cette gestion manuelle des exceptions s'avère assez efficace en termes d'économie de code. En effet, s'il suffit parfois de quelques dizaines de lignes pour mettre en place une simulation, le double voire le triple est souvent nécessaire pour gérer les quelques cas particuliers qui peuvent se présenter. Néanmoins, lorsque le nombre d'éléments devient plus important (la probabilité de voir apparaitre une exception croissant proportionnellement) et que les règles sont plus complexes, ou encore qu'on veuille simplement exploiter l'apparition de ces exceptions, il peut être impossible de mettre en place cette interaction, l'utilisateur s'en remet alors au caractère heuristique des simulations multi-agents.

II. Programmation

La programmation orientée objet désigne une programmation par modules de codes appelés objets destinés à être réutilisé de manière partielle ou totale afin de construire d'autres modules de codes. Elle peut être déployée aussi bien dans une approche générative que paramétrique. On émet ici l'hypothèse qu'un contrôle de l'outil (le scripting par programmation orientée objet) permet un contrôle du processus de conception.

i. Programmation orientée objet

La programmation orientée objet désigne une procédure de programmation informatique élaborée dans les années 1960 par Ole-Johan Dahl et Krysten Nygard puis poursuivie dix ans plus tard par les travaux d'Alan Kay. Cette approche propose d'envisager un programme comme un assemblage de modules de codes dédiés chacun à une tache spécifique, on parle alors d'un programme modulaire. Pour définir un objet, on s'intéresse à ses caractéristiques génériques (le type de données nécessaires à sa création) exprimées sous la forme de variables et à son comportement (le traitement de ces données) exprimé sous la forme de fonctions. Variable et fonction sont regroupées au sein d'une classe. Une classe n'est pas un objet mais une définition potentielle de plusieurs objets. Un objet est une instance, un individu crée à partir d'une classe avec des variables propres.

« *Variables and functions are the building blocks of software. Several functions will often be used together to work on a set of related variables. Object Oriented Programming was developed to make this process more explicit. Object oriented programming uses objects and classes as building blocks. A class defines a group of methods (functions) and fields (variables). An object is a single instance of class. The field within objects is typically accessible only via its own methods, allowing an object to hide its complexity from other parts of a program.*[30] »

Dans un logiciel de CAO, l'utilisateur n'a pas un accès direct à la programmation orientée objet. Néanmoins, lorsque l'on crée une ligne en donnant deux points dans l'espace de travail, on fait en fait appel à une classe :

Entrer la commande ou cliquer sur l'icône correspondant appelle la classe « ligne » qui demande alors deux points à l'utilisateur. Les deux points sont deux variables de cette classe (d'autres variables peuvent intervenir, la longueur de la ligne, l'épaisseur du trait, etc.). Une classe peut donc être utilisée pour créer plusieurs objets suivant des variables différentes ou être intégrée dans une autre classe afin de créer un objet de hiérarchie supérieure. La classe « ligne » fait d'ailleurs ici appel à une autre classe « point », définie par trois coordonnées x, y, z et un repère. La ligne et le point de cet exemple sont des datatypes composites issus de l'assemblage de datatypes primitifs (nombres entiers, décimaux et booléens). Ainsi, au plus petit niveau de l'algorithme, on trouvera toujours un nombre. Créer une classe revient en fait à créer un nouveau datatype.

« un aller-retour entre un système algorithmique et une décision explicite – une forme de computation désordonnée » (notre traduction)

30 **Casey Reas & Ben Fry**, *Objects* in *'Processing : a programming handbook for visual designers and artists'* 2007, The MIT Press Cambridge, Massachusetts, London, England p.395

« Variables et fonctions sont les modules (building blocs) du logiciel. Plusieurs fonctions sont souvent utilisées ensemble sur un ensemble de variables. La programmation orientée objet à été developpée pour rendre ce procedé plus explicite. La programmation orientée objet utilise des objets et des classes comme modules. Une classe définie un ensemble de méthodes (function) et un champ d'application (variables). Un objet est un une instance individuelle de classe. Les variables d'un objet sont définies localement, permettant de distribuer la complexité au sein du programme » (notre traduction).

Si l'opération de créer une ligne entre deux points ne brille pas par sa complexité, on peut imaginer des assemblages beaucoup plus riches.

La même logique opère lorsque l'on programme dans un logiciel de CAO. Cette relation est illustrée par l'entête d'une page de code où l'on « importe » les librairies utilisées par la suite. Ces librairies constituent un ensemble de classes prédéfinies : si l'on écrit « import rhinoscriptsyntax » dans un éditeur Python ou « import processing.lib » dans un éditeur Javascript, on informe l'ordinateur que l'on va utiliser respectivement les méthodes issues des librairies de Rhinocéros ou de Processing. Ainsi, la création d'une ligne dans python est définie par la commande « rhinoscriptsyntax.AddLine (start, end) » (start et end étant deux points ou deux listes de points). L'utilisateur peut également importer des librairies de sa création ou mise à disposition par d'autres.

Le système de classe et la réutilisation de modules de codes qu'il implique est une des clés de la programmation orienté objet :

"*Objects should be built for reuse. After difficult programming object is solved and encoded inside an object, that code can be use later as a tool for building new code*".[31]

ii. Meta données

La continuité entre les différentes étapes de création d'un processus génératif et/ou est maintenue par l'utilisation de métadonnées. Une métadonnée est une donnée en accompagnant une autre, dans le cadre de la conception architecturale, on considéra comme métadonnée tout ce qui n'est pas visible. Si la continuité de données depuis les premières étapes de conception jusqu'à la fabrication se prête bien à un contexte académique ou de recherche impliquant un nombre réduit de personnes partageant les mêmes compétences vis-à-vis de la programmation, elle ne fait pas règle dans le monde de l'architecture construite[32,33].

En effet, face à la multitude d'acteurs et à la nécessaire exhaustivité d'un projet architectural complexe, il est impossible (et pourtant souhaitable) de condenser la totalité d'un projet dans une seule définition.

L'utilisation des outils digitaux est alors fragmentaire, afin de permettre à plusieurs utilisateurs de travailler sur un même projet en même temps. La continuité paramétrique est assurée par des métadonnées (non géométriques) et par leurs organisations : c'est l'arborescence de Digital Project (Gehry Tech), le dataTree de Grasshopper (McNeel Associates) ou les procédures de nomenclatures en Python, Visual Basic, etc. Le principe de classe, support de la « programmation modulaire » précédemment évoquée en programmation orientée objet, trouve son équivalent dans tous les logiciels paramétriques ne traitant pas directement avec du code comme la PowerCopy de DigitalProject ou les Clusters et UserObjects dans Grasshopper.

L'utilisation de métadonnées dans une interface de scripting et plus explicite, elles sont traitées au même titre que la géométrie et l'organisent : il peut s'agir par exemple de la place d'un élément dans une liste, du nom d'un objet ou d'une note qui l'accompagne.

iii. Influence sur la production architecturale

Dans son Reader '*Scripting Culture'*, Mark Burry dresse une liste non exhaustive des outils (softwares et langages) disponibles :« *There is a very long list of scripting tools available. The list in the table here is distilled from my 34 correspondents: Adobe ActionScript | C, C#, C++ | Generative Component Script | Html | Iron Python |Java, JavaScript | LUA | Mathematica | MATLAB | MaxScript | MayaScript |Maya (Maya Embedded Language (MEL) and Python | Objective-C |Perl | PHP | Processing (Java) | Python | Rhinoscript (VB, Grasshopper, and Python) | Rhl | VB, VBA* [34]» (Burry, 2013).

---

31 **Casey Reas & Ben Fry,** *Objects* in *'Processing : a programming handbook for visual designers and artists'* op. cit. p.403 « Les objets devrait être crées dans le but d'être réutilisés. Après la programmation d'un objet complexe, ce code peut être stocké dans cet objet et utilisé plus tard pour la construction de nouveau code » (notre traduction)

32 **Hanno Stehling, Fabian Scheurer, Jean Roulier,** *Bridging the gap from CAD to CAM* paper submitted for the Fabricate conference,Zurich, October 2013, not published yet

33 *Very Large Gasshopper files, Collaboration, UI ideas*, Sujet de discussion sur le forum grasshopper, 3 septembre 2013 : http://www.grasshopper3d.com/forum/topics/very-large-gasshopper-files-collaboration-ui-ideas

34 **Burry Mark**, *Cultural Defense* in '*Scripting Cultures'*, op.cit. pp.36-37

Avec l'utilisation d'un logiciel de CAD, le concepteur s'en remet à l'algorithme attenant à la commande utilisée (par exemple, la création d'une surface entre deux lignes). Ces algorithmes conditionnent le langage (formel) utilisé et le processus est opaque. Ce constat pousse une génération d'architectes et de designer à s'emparer de la création de logiciel pour répondre de manière plus précise et contrôlée à leurs besoins. Processing est un bon exemple de cette appropriation : crée par Ben Fry et Casey Reas dans le cadre de leurs recherches au laboratoire Aesthetics + Computation Group (ACG) du MIT Media Lab en 2001, Processing est une interface de scripting basée sur le langage Java à destination des designers. Le langage est simplifié pour permettre une utilisation plus intuitive et explicitement orienté pour la création graphique : Processing ne contient 'que' 250 fonctions pour sa version 2.1 listées sous les catégories Structure, Environment, Data, Control, Shape, Input, Output, Transform, Lights&Camera, Color, Image, Rendering, Typography et Math. L'utilisation de librairies externes permet d'étendre le nombre de fonctions disponibles et d'y intégrer des notions de géométrie plus avancées, de physique, etc.

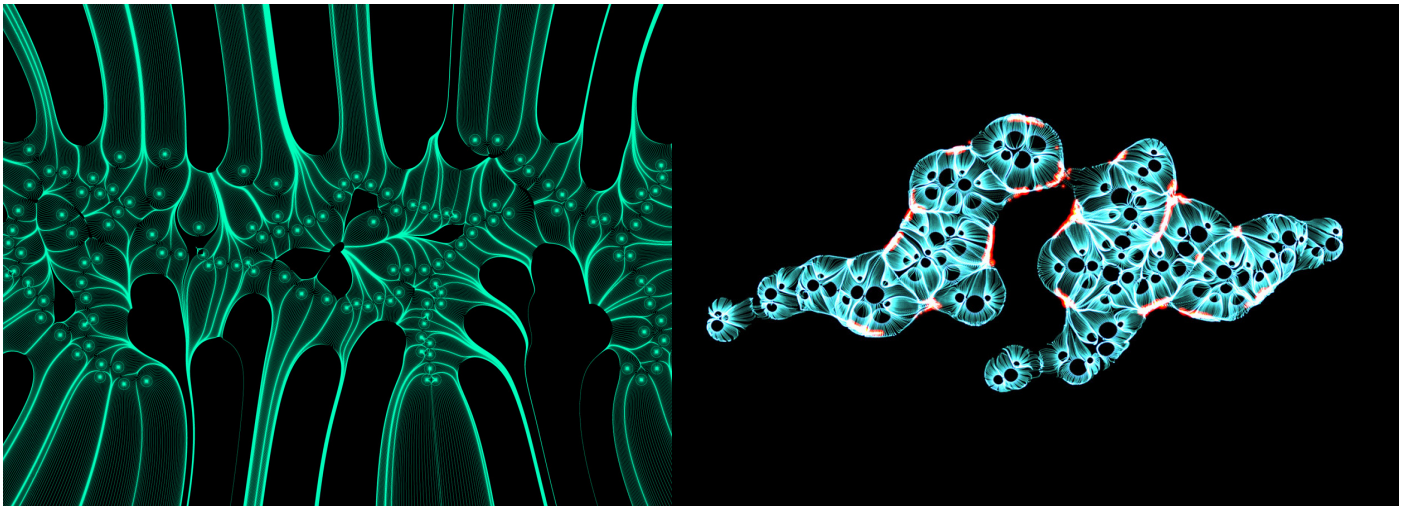
Notons que le canvas de Processing est un espace 2D et que le travail en 3 dimension (indispensable à la création d'espace) n'est permis que par un détournement de l'outil. Si l'interface et la syntaxe sont assez intuitifs (bien qu'on trouve des langages de plus 'haut niveau', c'est-à-dire dont la syntaxe se rapproche plus du langage humain que du langage machine), la gestion de métadonnées s'avère plus compliquée.

Précédemment, on a établi le double usage du scripting par les architectes : 'optimisation' et 'spéculation'. Il est intéressant de mettre face à face l'interface et la production. On note alors que les travaux d'optimisation plus souvent écrit dans un langage offrant des possibilités d'export et de traitement de métadonnées simples (designtoproduction travaille actuellement sous Iron Python pour Rhino) quand les travaux plus spéculatifs privilégient l'interface simple de processing.

iv. Formation à la programmation et styles

La tentative d'identification d'un '*style'* ou d'un *'ordre'* paramétrique permettant l'élaboration d'une architecture plus *'parlante'* par Michael Schummarer dans son ouvrage '*The Autopoiesis of Architecture: A New Framework for Architecture*' a été très largement décriée, le contre argument le plus répandu étant que la conception computionelle offrant une capacité de production de forme illimitée, la réduction de cette dernière à un style est de fait impossible. Néanmoins, force est de constater l'émergence de 'tendances' digitales : application d'un diagramme de voronoi pour la panélisation d'une surface, de champs éléctro-magnétiques, etc.

La raison de l'usage récurrent d'un algorithme particulier se trouve dans les modalités de formation à la programmation des architectes. Contrairement aux informaticiens, introduits à la programmation d'un point de vue technique et académique, l'apprentissage du scripting par les architectes se fait souvent de manière autonome, par la répétition de code existant (tutoriaux, ressources en ligne, etc.), et, si cette formation a lieu dans un contexte académique, c'est souvent par répétition de la pratique du professeur (avec un peu de recul, n'est ce par le modus operandi de toutes formations architecturales et pas seulement computationnelle ?) : on assiste alors à l'émérgence d'une 'zone de confort' constituée d'algorithmes simples ou suffisamment répétés pour le devenir.



à droite, OrbitaSeries de Biothing (2006), à gauche, softCastConfiguration de Minimaforms (2013)

III. Etat des lieux : quelle place pour les systèmes non linéaires dans l'équation architecturale.

L'application d'un modèle paramétrique ouvert, d'un système génératif multi-agent ou d'un hybride entre les deux peut prendre plusieurs formes selon la destination du modèle.

i. Agents simples & Packing :

Conseil d'Education d'Abu Dhabi, Aedas R&D

Le programme de R&D au sein de l'agence AEDAS dirigé par Christian Derix à développer de nombreuses utilisations de systèmes d'auto organisation aussi bien pour la programmation urbanistiques qu'architecturale. Aedas R&D est présenté comme une alternative à la recherche dans le domaine paramétrique : « *Instead of another 'advanced geometry' or 'modelling' group, Aedas decided to form a computational design group that would explore the application of artificial life and intelligence based on self-organization, bottom up processes and distributed representation to architectural design projects.* [35]».

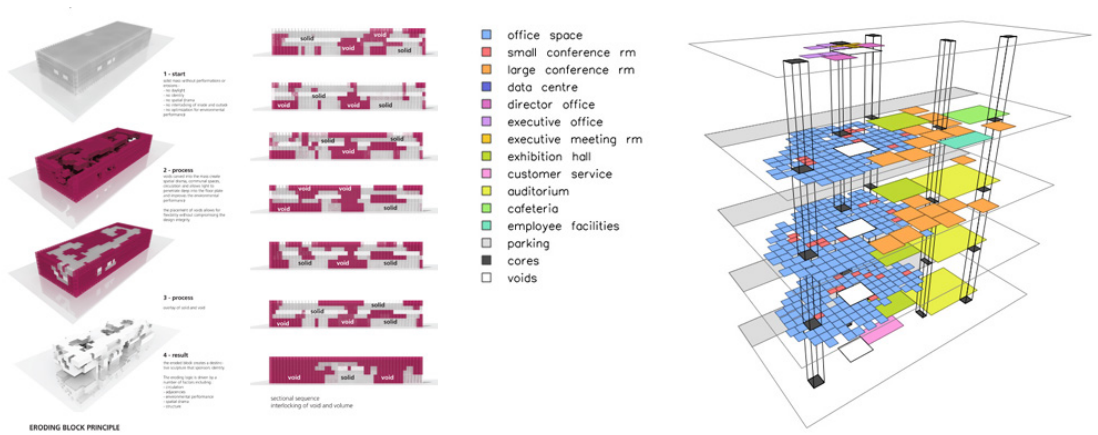


figure 1 et 2 : Conseil d'Education d'Abu Dhabi, diagrammes d'organisation et packing

Les systèmes multi agents sont ici utilisés pour simuler l'occupation d'un bâtiment, les règles appliquées ont pour but de gérer la distribution et les relations entre différents types d'espace, la mise en place de circulation ou de connections visuelles… Le but n'est pas ici de créer une géométrie architecturale mais plus un outil de conception, souvent diagrammatique, permettant de traduire une intention architecturale en termes d'algorithmes. Ainsi, à propos du *Conseil d'Education d'Abu Dhabi : « The interactive design tool developed by the Computational Design and Research team allowed the architectural team, and the client, to easily test and evaluate a number of different programmatic scenarios. Areas are loaded from the accommodation schedule, and these try to arrange themselves automatically according to an adjacency matrix. It is possible to also pull the boxes around manually to obtain good solutions to the layout.*[36] ».

L'algorithme utilisé est un système de type packing : les objets ne peuvent pas se recouvrir et s'attirent ou

35 **Derix Christian** et **Izaki Asmund**, *Spatial computing for the new organic* in '*Computation Work, The Building Of An Algorithmique thought*' Architectural Design, Profile n°222, mars/avril 2013, Wiley Press, London. p. 43
« Plutôt qu'un autre groupe de 'géométrie avancée' ou de 'modélisation', Aedas a décidé de former un groupe de design computationnel explorant les applications de la vie artificielle et de l'intelligence basée sur l'auto organisation, les processus bottom up et la représentation distribuée à des projets d'architecture » (notre traduction)

36 Aedas website : http://aedasresearch.com visité le 8 septembre 2013
« L'outil interactif developpé par l'équipe de recherche&design computationel a permis aux architectes et au client de tester et d'évaluer facilement différents scénarios programmatiques. Les surfaces sont importés à partir du 'tableau des surfaces' et s'organisent spatialement selon les contraintes définies. Il est également possible d'ajuster manuellement la position des objets afin d'obtenir une disposition satisfaisante par niveaux » (notre traduction)

se repoussent suivant leurs propriétés. Concrètement, on peut imaginer un espace bruyant comme une salle de réunion qui tendra à repousser des espaces calmes comme une cellule de travail.
Notons la dernière phrase de la citation sur l'interaction entre le modèle et son utilisateur, l'interaction est ici d'autant plus justifiée par le fait que l'utilisateur du modèle ne soit pas son concepteur.

ii. Agents complexes : *Biothing & Kokkugia*

Les recherches de l'agence Biothing - dirigée par Alisa Andrasek – et de Kokkugia – Rob Stuart Smith & Roland Snooks - sont une référence dans l'utilisation de systèmes multi-agents en architecture.

L'usage d'agents intelligents est présenté comme une alternative à la prise de décision arbitraire au sein du design paramétrique : « *Les décisions liées à la conception sont prises à l'intérieur de cet espace de recherches [la simulation], et non pas au dehors, ce qui change la politique et la manière d'envisager l'écologie de la conception.*[37] ». Cet usage est justifié par la volonté de création d'objets continus : « *Ce magma de données primordiales s'instancie en plusieurs facettes qui sont autant d'états émergents pré-géométriques ; leurs résistance et leurs souplesse d'utilisation permettent ainsi de redonner une seule texture aux expressions structurelles, matérielles et organisationnelles.*[38] », émergents, et en relation avec un environnement modélisé par un nombre croissant de données disponibles. Notons que la production est qualifié de « pré-géométrique » et ne peut donc pas être soumise telle quelle à la fabrication.

On retrouve la même préoccupation pour la prise de décision non explicite, l'émergence et la continuité chez Roland Snooks : « *a subjective design sensibility emerges from orchestrating the interaction of many micro decisions and intention.*[39] ». Dans le texte *Volatile Formation,* ce n'est pas seulement le design paramétrique qui est remis en cause mais également les procédures d'optimisations et de form-finding (pourtant également issus de l'intelligence collective et d'une interaction locale entre éléments d'un système global) au profit d'un contrôle beaucoup moins direct : « *the argument for volatility is more than a theoretical concern* ; *it is a fundamental concern for the importance of subjectivity and the nature of risk within design.*[40] ».

Pavillon Seroussi (2007) – Biothing[41]

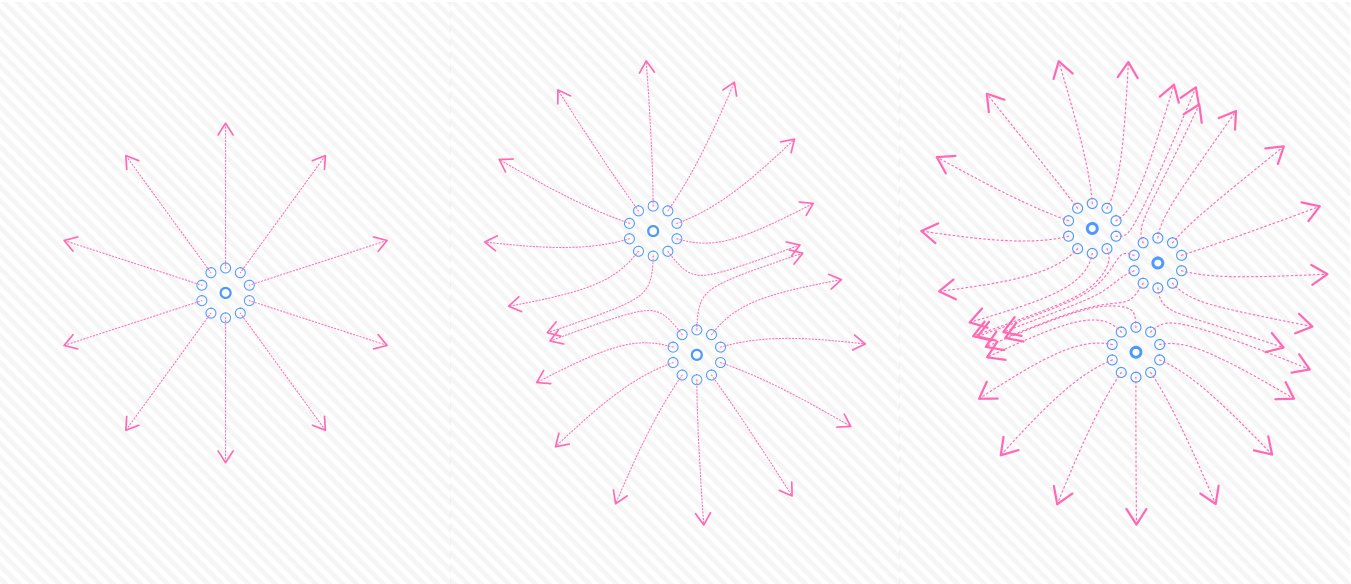


figure 3: comportement des EMFs avec 1, 2 et 3 naissances

37 **Andrasek Alisa**, *Biothing, une conception à intentionnalité variable* in '*Biothing*', editions HYX Orléans, 2009. p.54.
38 **Andrasek Alisa**, *Biothing, une conception à intentionnalité variable.* op.cit. p.55.
39 **Snooks Roland**, *Volatile Formation*, op. cit., p.60 « Une sensibilité subjective émerge de l'orchestration des interactions de multiples micro décisions et intensions » (notre traduction)
40 **Snooks Roland**, *Volatile Formation*, id. p.62 « l'interêt pour la volatilité est plus n'est pas une question théorique ; c'est un questionnement fondamental par rapport à l'importance de la subjectivité et la nature du risque dans les processus de design » (notre traduction)
41 Andrasek Alisa : principal designer, FlowerPower custom written plug-in: Kyle Steinfeld with Alisa Andrasek, Design team: Ezio Blasetti / Che Wei Wang / Fabian Evers / Lakhena Raingsan / Jin Pyo Eun / Mark Bearak

La proposition de biothing pour le pavillon Seroussi est un des premiers travaux de l'agence. Le travail algorithmique se décompose en deux parties.
La première consiste à dessiner la toiture en deux dimensions en utilisant une population d'agents se déployant suivant des règles similaires à celles d'un champ magnétiques à partir de naissances définies à la souris par l'utilisateur. Les agents servent ici à mettre en place un cloisonnement de l'espace : tous les éléments du système (naissances et particules) agissent comme des répulseurs, quand deux particules se rencontrent, leurs trajectoires s'inclinent. Un grand nombre de particules tend décrire des trajectoires groupées lors de la fuite, se dessine alors une répartition de l'espace à la manière d'un diagramme de Voronoi.
La toiture plane obtenue est ensuite levée en suivant les variations d'une fonction sinus, cette étape se rapproche plus d'un traitement paramétrique que multi agent.
Dans ce processus, la relation entre l'agent et son environnement est limité à sa perception de la population dans laquelle il évolue, le reste (le placement des naissances) est laissé au choix du concepteur.

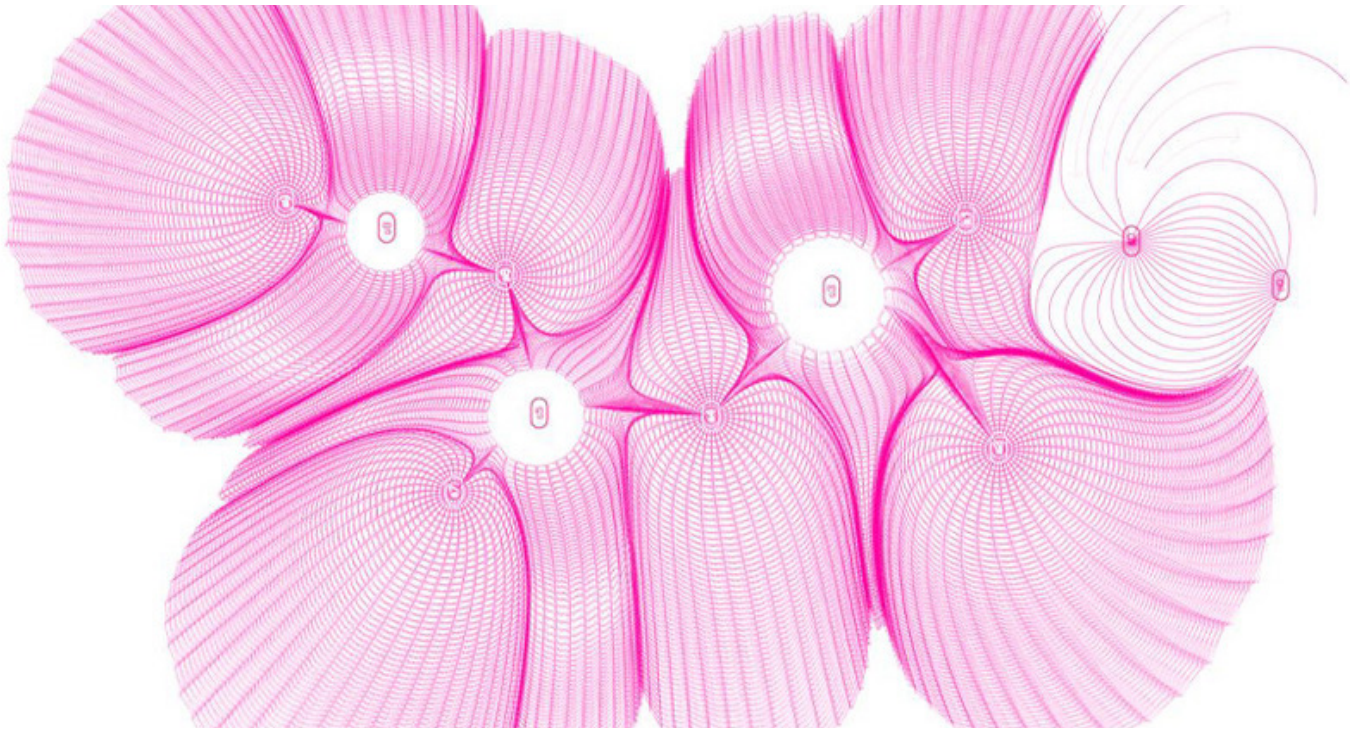


figure 4 : Pavillon Seroussi

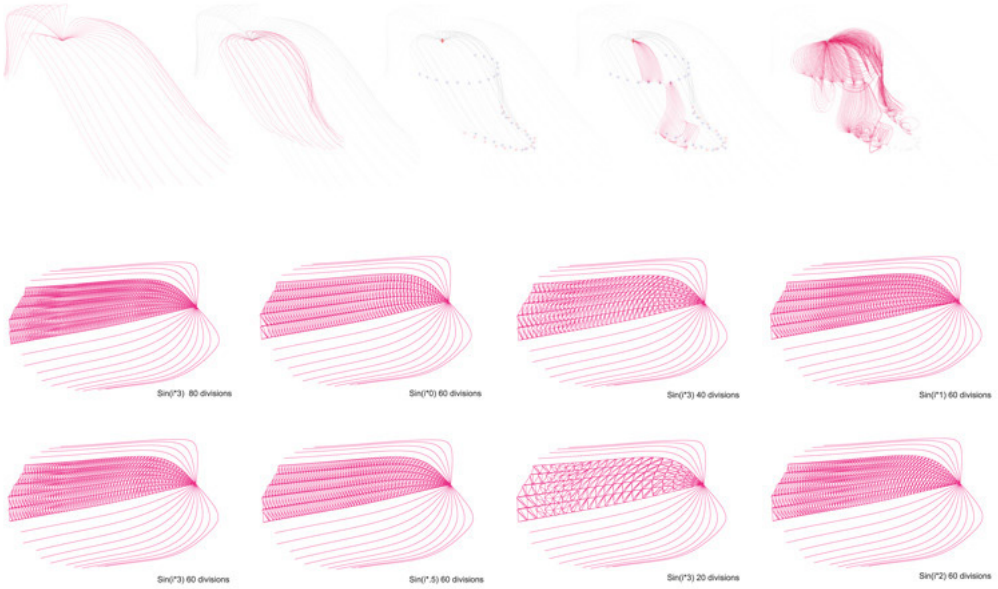


figure 5 : Pavillon Seroussi, 'panelisation'

FissurePort (2010) – Biothing[42]

figure 6 : FissurePort

Le travail algorithmique de FissurePort est principalement concentré sur la façade du terminal Kaohsiung de Taiwan. Des agents descendent sur un mesh vertical prédéfini et le creuse à la manière d'une falaise sous l'érosion. En fait, ces particules sélectionnent les vertices du mesh les plus proches et les déplacent vers l'intérieur, plus la particule est lente, plus le déplacement du vertice est profond, le passage successif de plusieurs agents creuse la façade. L'évaluation du résultat est vraisemblablement subjective, l'architecte indique que les failles agissent comme des puits de lumière sur la façade mais il n'y a pas l'air d'y avoir un feedback entre une mesure de la luminosité intérieure et le parcours des agents. Le mesh de base, les naissances des agents et le nombre de passage semble être le fruit d'une décision explicite de l'architecte. Comme pour le pavillon Seroussi, l'architecte émet une intention globale (une répartition de l'espace ou le percement d'un puits de lumière sur la façade) et anticipe un résultat formel mais le dessin est automatisé.

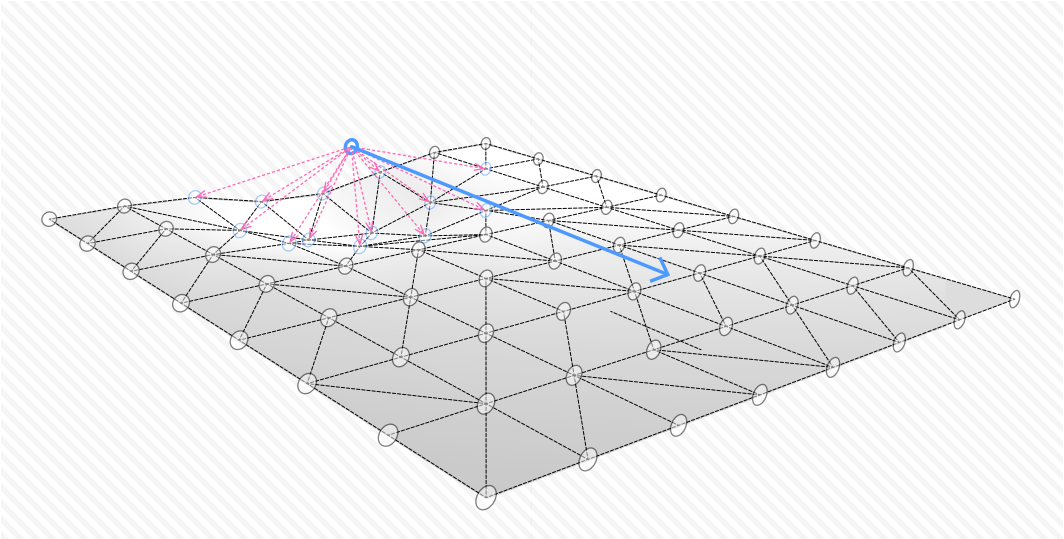


figure 7 : FissurePort algorithme

Swarm Matter (2009) – Kokkugia[43]

L'analyse du kAgent[44] pour le workshop de Strelka[45] laisse à penser que l'algorithmie sous-jacente aux recherche sur le swarm de kokkugia est une version élaborée du Boid de Reynolds : reprise des fonctions *séparation (separation)*, *alignement (align)* et *cohésion (cohesion)*, ajout des fonctions *errance (wander)*, *guidage (steer)* et de relations *ressorts (springs)*. Ce projet cristallise les intentions de Roland Snooks à propos de *formation volatile* évoquées précédemment. L'algorithme à l'œuvre s'apparente à celui d'un *ant trail*, ce n'est pas la particule en elle-même qui constitue l'objet mais la trace qu'elle laisse durant son déplacement. Un second algorithme vient fusionner ces sillages lui donnant son épaisseur. Le résultat est un objet continu composé d'éléments unis sans distinction hiérarchique.

42 Alisa Andrasek & Jose Sanchez : principal designers, design team: Knut Brunier – Gabriel Morales – Denis Lacej

43 Design Director: Roland Snooks, Project Team: Pablo Kohan, Alan Song-Ching Tai (software development)

44 kAgent Strelka Workshop, **Robert Stuart Smith** (2013) basé sur kAgent, Roland Snooks (2007) : https://github.com/a1anme/indicator/blob/318cff3c6273054485301313b77ff94acbe41e90/kAgent.pde (visité le 20 décembre 2013)

45 **Robert Stuart Smith**, *FORMATION AND POLYVALENCE: SELF-ORGANISATION OF ARCHITECTURAL MATTER* in '*Open Closed city Branchpoint' Workshop,* Strelka Institute, 4 aout 2013 : http://issuu.com/katyalarina_ulab_spb/docs/open_closed_city_workshp_strelka_20 (visité le 20 décembre 2013)

Deux applications sont alors possibles : une augmentation de l'échelle ou une augmentation du nombre d'agents à l'œuvre. Le plus souvent, la population d'agents agit sur ou à l'intérieur d'une surface prédéfinie. Le problème de la génération de la forme n'est donc que partiellement résolu par le swarm car une autre étape (générative ou non) est nécessaire en amont. Le travail des agents s'apparente alors plus une panélisation locale qu'à de la réelle génération simultanée remplie un des objectifs de la *formation volatile* :

De plus, le passage d'un état pré-géométrique à un état géométrique constructif et constructible n'est pas réalisé. Le discours d'une adéquation entre la conception et les modes de fabrication (actuels) peut être remis en cause dans la mesure où cette fabrication ne semble permise que par des procédés additifs peu viables pour une géométrie complexe à une échelle architecturale.

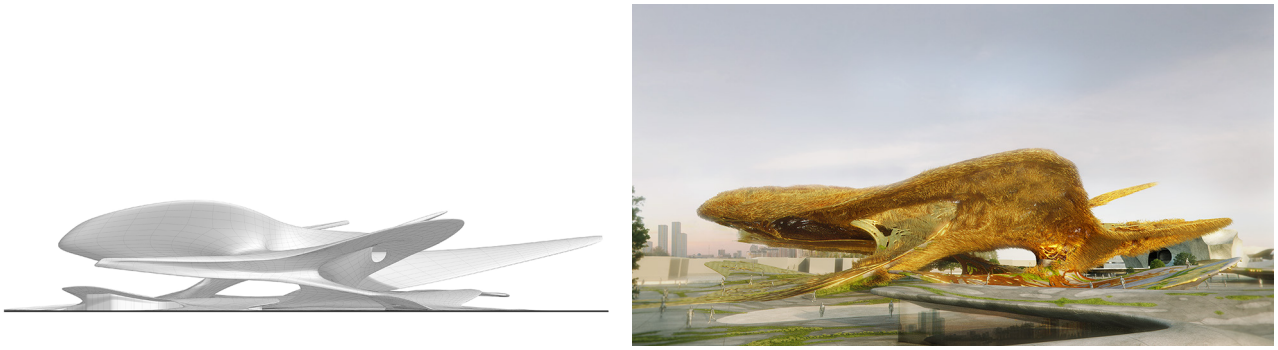


figure 8 : Surface de référence et Population de surface

Protohouse (2012) – SoftkillDesign

Cette utilisation des procédés additifs est illustrée par le projet *protohouse* de softkill, développé par des étudiants de Robert Stuart Smith dans un post graduate à l'Architectural Association de Londres. Ce projet présente une conception algorithmique plus complexe que le swarm matter de Kokkugia puisqu'il mêle travail d'agent et optimisation structurelle. L'optimisation structurelle permet de définir un premier volume composé de voxels présentant des pleins et des vides qui sont ensuite populé par une nué d'agent afin de former les fibres destinées à être imprimées en 3D à grande échelle. Le « réalisme » de la proposition architecturale puisqu'on y reconnaît des éléments de programme (un bureau, un escalier, une mezzanine).

figure 9 : protohouse, détail

iii. Agents comme support d'un travail paramétrique

Le Groningen Stadbalkon (2003) – designtoproduction

La même distance vis-à-vis des « décisions explicites » peut être trouvée dans les textes de Fabian Scheurer sur l'auto-organisation en architecture : « *When it comes to actual construction of a complex building, the question arise : What is a reasonable quantity of explicit information for a specific design.*[46] ». On retrouve également la reconnaissance et l'utilisation d'un état pré-géométrique dans la simulation : « *the complexity of this model [the simulation] is delibertatly 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.*[47] ».

46 **Fabian Scheurer**, *Getting complexity organised Using self organisation in architectural construction* in '*Automation in Construction n°16',* Elsevier, 2007, p.78

"lorsque l'on parle de la construction d'un edifice complexe, on peut se poser la question suivante : quel est le nombre raisonnable de décisions explicite pour un projet spécifique » (notre traduction)

47 **Fabian Scheurer,** *A simulation toolbox for self organisation in architectural design* in '*Innovation in Architecture, Engi-*

Le Groningen Stadsbalkon (Kees Christiaanes Architects an Planners, 2003) utilise un système d'agents similaires à celui présenté précédemment pour le *circle packing* afin de distribuer les colonnes porteuses de la couverture d'une allée piétonne.

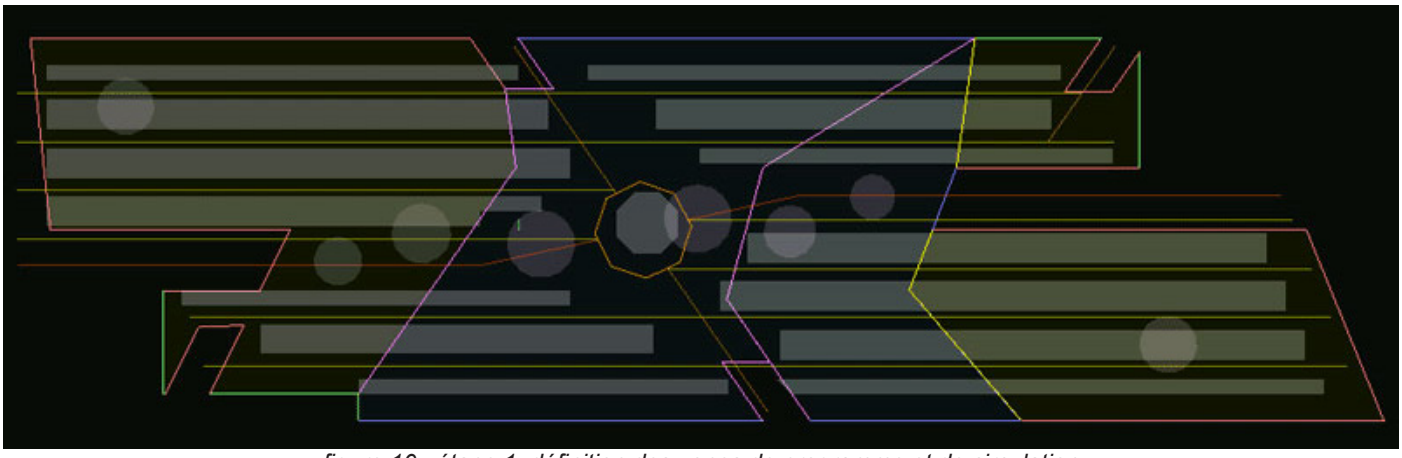


figure 10 : étape 1, définition des zones de programme et de circulation

Une colonne est définie par deux points respectivement sur une surface basse (le sol) et une surface haute (la couverture). Chaque point est le centre de deux cercles, le premier défini le diamètre d'un poteau (en relation avec la charge reçue) et le deuxième défini sa zone d'influence, la ligne reliant le centre bas et le centre haut est contrainte par un angle maximal par rapport au sol. A chaque itération, les cercles évoluent proportionnellement selon les règles suivantes causant la répartition des points sur la surface :

- Mouvement : si deux zones d'influence se chevauchent, les cercles se déplacent dans la direction opposée
- Croissance : un poteau croît si son diamètre n'est pas suffisant pour la charge qu'il reçoit
- Décroissance : un poteau décroît si son diamètre est supérieur à celui nécessaire pour la charge qu'il reçoit
- Division : si un poteau atteint la taille maximale, il se divise en deux poteaux de taille minimale
- Elimination : si un poteau atteint la taille minimale, il est supprimé
- Environnement : Des courbes définissant les allées ont un effet de répulsion sur les poteaux

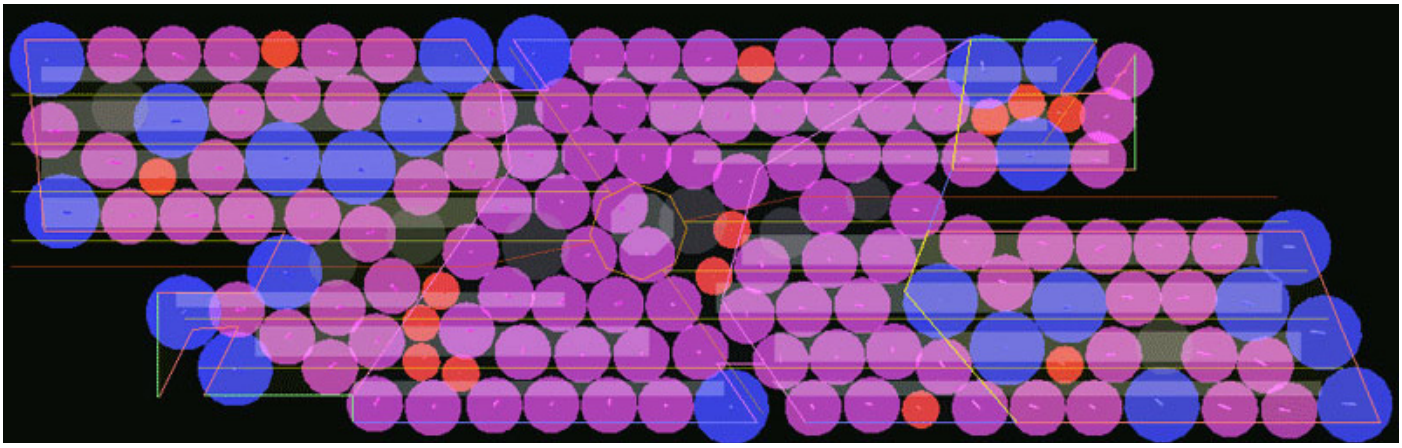


figure 11 : étape 2, auto organisation

Toutes ces règles sont directement reliées à des critères numériques (la charge qu'un poteau peut ou ne peut pas porter) ou géométriques (la présence à l'intérieur ou à l'extérieur d'une zone de répulsion) facilement évaluable. La notion de « messy computation » (l'interaction entre un système algorithmique et décisions

neering and Computing (AEC)', ed; Sevil Sariyildiz et Bige Tuncer, Rotterdam, NL : Delft University of Technology, Faculty of Architecture, 2005. p.534.

"La complexité de ce modèle [la simulation] est délibérement plus faible que celle du projet construit, cette simplification permet une interaction en temps réel avec l'utilisateur qui peut alors ajuster les paramètres ou influencer localement la position d'un composant » (notre traduction)

explicites) est présente, l'utilisateur pouvant influencer la simulation en déplaçant des points pendant son exécution. Ce système de règles reliées à des critères numériques ou géométrique et l'interaction avec l'utilisateur permet de mette en place une évaluation qualitative de la production à la fois selon des critères numériques (la réponse ou non à un problème) et subjectifs. Cette évaluation qualitative est une forme d'optimisation.

La complexité de la simulation étant inférieure à celui de la réalité construite (n'y figure que des représentations simplifiées des éléments construits), elle constitue la base d'un travail paramétrique qui permet la transformation de cet état proto géométrique vers un état géométrique et constructible.

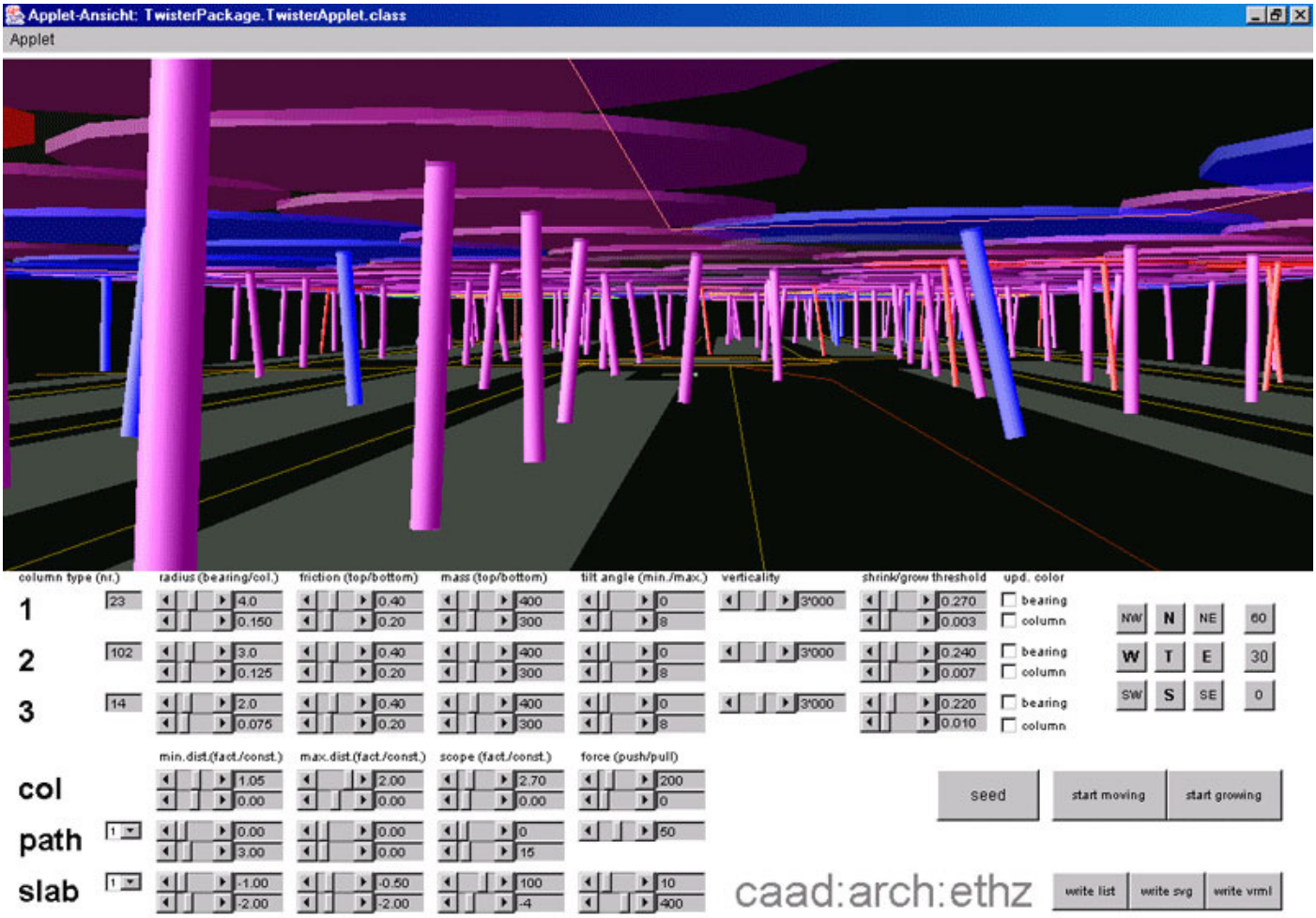


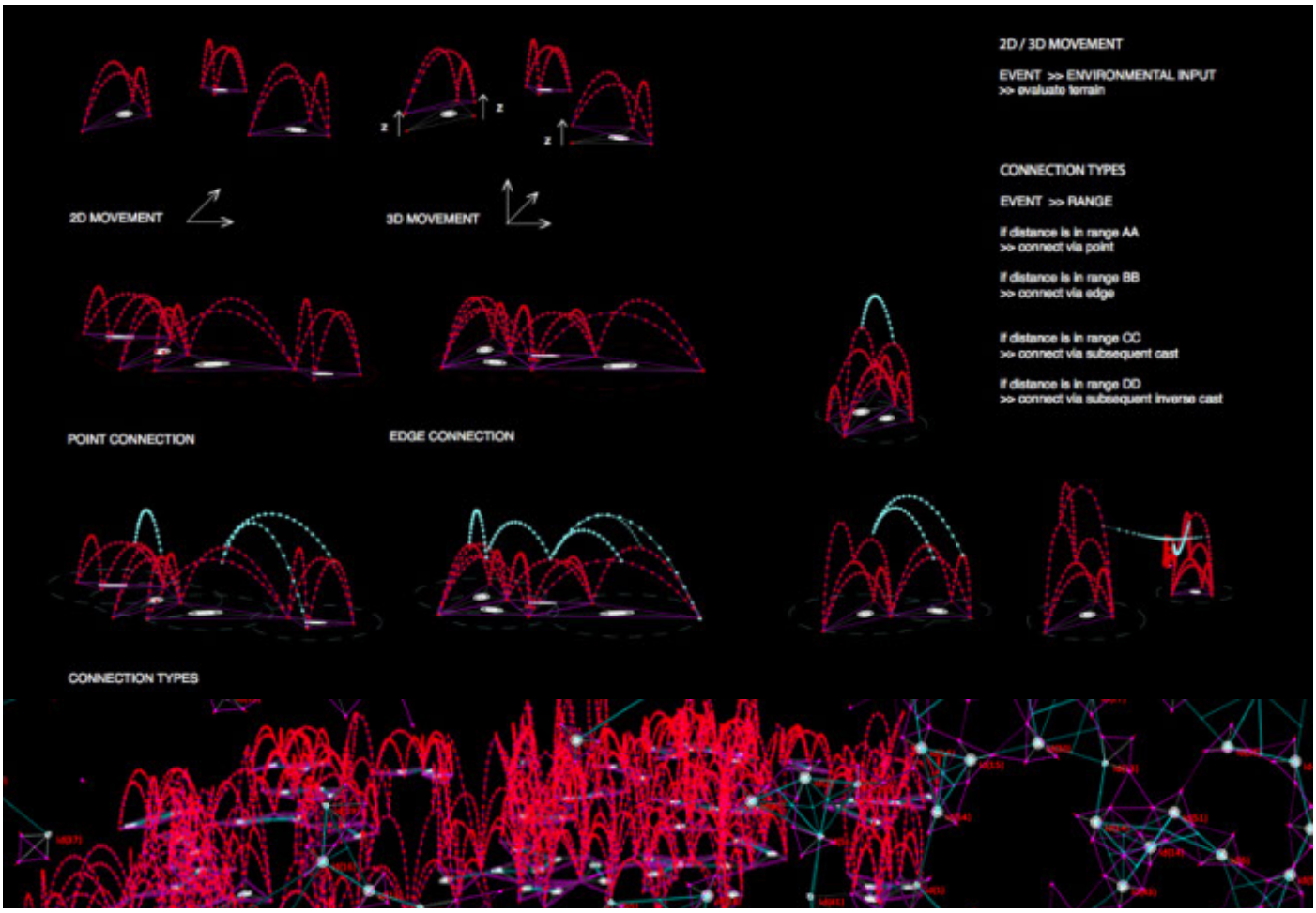
figure 12 et 13 : vue intérieures dans la simulation et le batiment construit

La simulation est donc ici considérée comme la base d'un travail paramétrique. Ce travail paramétrique est relativement simple dans le cas du Groningen Stadsbalkon - l'instanciation d'un poteau sur la ligne qui le symbolise – mais devient beaucoup plus complexe, par exemple dans le cas de la relaxation dynamique d'un réseau de poutre sur une surface.

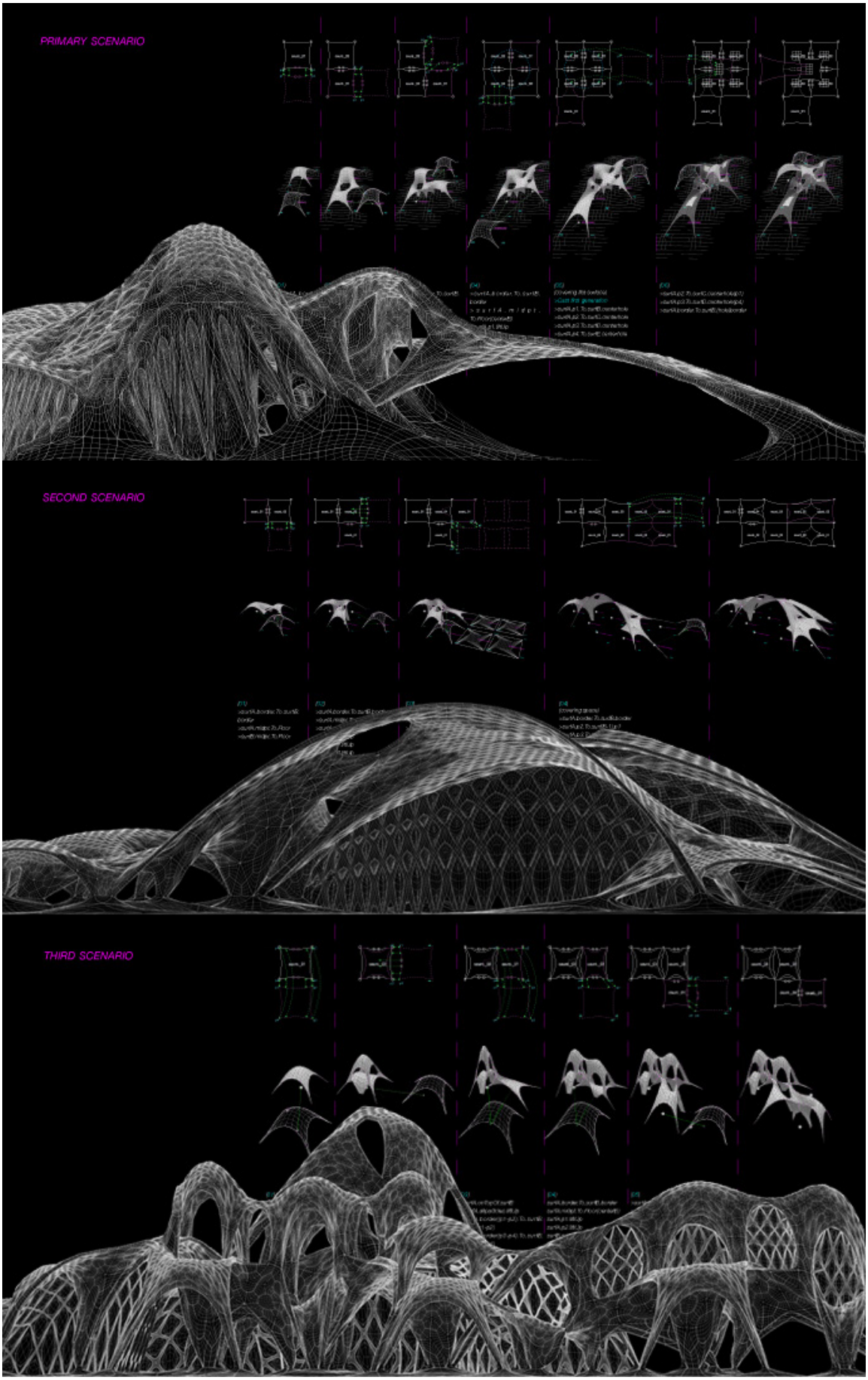
iv. l'agent comme simulation de la matière

Digital Plaster (2009-2013) – Manuel Garcia Jimenez

Le projet Digital Plaster a débuté en 2009 dans le cadre du post-graduate de Theodore Spyropoulos[48] par la simulation du comportement du béton coulé dans un coffrage en tissus. Cette étude matérielle préliminaire est cruciale dans la définition du projet puisqu'elle fixe le cadre de travail. Contrairement aux projets de Biothing ou de Kokkugia précédement évoqués, Digital Plaster pose les contrainte de fabrication et de mise de œuvre a cœur du projet architectural : ce n'est pas la matière qui se 'plie' à la géométrie (ou à la 'proto-géométrie) mais bien l'inverse. La génération du projet se déroule en deux phases. Dans un premier temps, l'agglomération et l'adaptation rétroactive de modules de couvertures dirigée vers des coordonnées prédéfinies. C'est durant cette phase qu'est généré l'espace, les modules sont représentés par des surfaces sans épaisseur pouvant se connecter soit par des points ponctuels ou par leurs arrêtes formant des assemblages plus ou moins denses et nombreux selon les règles définies. La deuxième consiste en la mise en volume des surfaces précédemment générées, les points de contrôles de ces surfaces se doublent et se dilatent pour simuler l'intrusion du béton (en pratique, un *spring* liant deux points de contrôles homogène se raccourci ou s'étend selon la pression des *springs* adjacents). Durant ces deuxièmes étapes, les emplacements pour la coulée du béton sont placés, ces derniers influant également sur la forme. La production matérielle n'en est encore qu'au stade de prototype au 1/10e mais des recherches sont en cours pour la réalisation à échelle architecturale. La totalité du bâtiment serait alors générée par une population d'agents.



48 Digital Plaster, Architectural Association, 2009. Design Team : Manuel Jimenez, Roberto Garcia, Stella Dourtme, Claudia Ernst. Tutor : Theodore Spyropoulos. Tech tutors : Shajay Bhooshan, Maustafa El Sayed.



page de gauche : figures 14 et 15, règles de connections entre modules et simulation globale
ci dessus : figures 16, 17, 18, adaptation des règles de la simulation à l'intention architecturale

v. Relation à la fabrication

Si la représentation classique en plans, coupes, élévation est appropriée pour une architecture orthogonale, elle échoue à transmettre la volumétrie (les longueurs n'étant pas maintenues dans une simple projection 2D) et la complexité d'un projet non standard :

« *Unlike most man-made structure, the architectures of these structures are not embedded in a blueprint, but rather are correlated operations governed through emerging collective interaction.*[49] ».

La procédure de création d'un objet ayant changé avec le paramétrique et le design génératif, on peut imaginer que la représentation change également. Par représentation, nous n'entendons pas ici la création d'une image du projet mais bien la communication des éléments nécessaires à sa matérialisation. En effet, la précision d'un modèle de fabrication doit être en accord avec celle des outils utilisés pur cette fabrication, vraisemblablement robotisée au vue de la complexité des composants dessinés. La libre interprétation par l'entrepreneur de l'intention de l'architecte à partir de documents papiers n'est plus envisageable dans le paradigme d'une architecture computationnelle et robotisée.

Nous l'avons vu, le paramétrique et le design génératif ne relève plus d'une description absolue (tant au niveau de la conception que de la représentation) mais d'une description relative des objets par rapport à leur environnement proche. Le contour d'un objet (ce qui est visible dans une interface CAO) n'est qu'une partie de ce qu'est réellement l'objet. Une courbe n'est pas seulement un élément tracé mais une liste de points de contrôle, une relation entre les deux éléments qu'elle connecte, un nom, la variation d'une courbe de référence, etc. Toutes ces informations existent au sein du modèle paramétrique et peuvent être exploité par une « planification topologique », c'est-à-dire, la représentation relative des éléments et l'intégration de métadonnées.

La transformation opérée pour rendre un fichier de CAO lisible pour la Fabrication Assistée par Ordinateur en est un exemple frappant. La plupart des logiciels de FAO intègrent des algorithmes de reconnaissance d'opération machines simples (slots, pocket, drilling hole, etc.) mais leur fiabilité est faible. Trois solutions se présentent alors :

- La création par le concepteur du code machine (auquel cas on se passe d'un logiciel externe de FAO), mais cette création nécessite de connaitre préalablement la machine utilisée. Si c'est souvent le cas dans un projet académique, ça ne l'est pas toujours pour une construction « réelle » où ce choix est laissé à l'entrepreneur.
- l'assignation manuelle de chaque opération, mais elle peut s'avérer très fastidieuse face à un nombre important d'objets et d'opérations.
- La définition et l'importation de ces opérations depuis un modèle paramétrique.

Cette représentation topologique a deux buts, communiquer la complexité d'un projet à d'autres personnes impliquées dans la conception et faciliter la transition de cette conception à la fabrication et la construction.

49 **Theodore Spyropoulos**, *Constructing Adaptive Ecologies : notes on a computational urbanism*. 2013. p.11

L'étude de l'utilisation de simulations multi agent en conception architecturale permet de mettre en avant différentes applications :

1. Une utilisation pour la production diagrammatique (point i)
2. Une recherche formelle libre sans anticipation ou ambition de production (point ii)
3. Une étape du processus de conception antérieure à un travail paramétrique (point iii)
4. La génération entière et autonome d'un projet architectural depuis la conception à la fabrication (point iv)

La différence entre les deux premières et les deux dernières étapes tient dans la prise en compte des contraintes de fabrication dans le processus de conception.

En effet, si l'approche 3 est utilisée de manière courante et pas forcément visible pour l'optimisation en vue de la fabrication (relaxation dynamique, simulation d'éléments finis, etc.), elle est plus souvent appliquée par des équipes d'ingénieurs en reprise du processus de conception que par les architectes dès les premières étapes de celles-ci. La prise en compte de disciplines externes à l'architecture par l'application de systèmes multi-agents (l'écologie de l'information) était pourtant l'une des motivations premières de son utilisation.

Si l'utilisation pour la production diagrammatique atteint ce but, son impact sur la conception est réduit puisqu'elle en est un résultat plus qu'une origine. De même, la recherche formelle porte un impact réduit sur la fabrication puisqu'elle s'abstrait des contraintes du réel.

Néanmoins, l'approche globale et autonome décrite en 4 semble être la piste la plus prometteuse puisqu'elle conjugue une étude sur la matière et sa mise en œuvre avec une intention architecturale. On assiste alors à une redéfinition du statut de l'architecte et une réappropriation de toutes les étapes du processus du projet par une meilleure maitrise des outils qui le conçoivent.

Sources images

page 10 à gauche : **Biothing** (Alisa Andrasek), *orbita series 2006*, http://www.biothing.org/

page 10 à droite : **Minimaforms** (Théodore Spyropoulos), *soft cast 2013*, http://minimaforms.com/

page 11 : **Aedas R&D** (Christian Derix), *Abu Dhabi Education Council*, http://aedasresearch.com/

page 12 : illustrations par l'auteur

page 13 : **Biothing** (Ezio Blasetti), *Seroussi Pavillon 2007*, http://portfolio.ezioblasetti.net/

page 14 : figure 6 : **Biothing** (Alisa Andrasek), *Fissure Port 2010*, http://www.biothing.org/

page 14 : figure 7 : illustration par l'auteur

page 15 : figure 8 : **Kokkugia** (Robert Stuart Smith & Roland Snooks), *kazakhstan symbol 2013*, http://www.rolandsnooks.com/

page 15 : figure 9 : **SofKill Design**, *protohouse*, http://protohouse.tumblr.com/

page 16&17 : **designtoproduction** (Fabian Scheurer), *Groningen Stadtbalkon 2009*, http://www.designtoproduction.ch/

page 18&19 : **M(a)dm** (Manuel Jimenez), *Digital Plaster*, http://www.madmdesign.com/

Bibliographie

Scripting : éducation et analyse

- Casey Reas & Ben Fry.- Objects in 'Processing : a programming handbook for visual designers and artists' 2007, The MIT Press Cambridge, Massachusetts, London.
- Matt Pearson, *Generative Art*, 2011, Manning Publications
- Meredith Michael, *Never enough (transform, repeat ad nausea)* in *'From Control to Design, parametric/algorithmic architecture'* Actar-D, Barcelone, 2008, pp. 6-9.

Emergence : vulgarisation scientifique

- Philip Ball.- *Follow your neighbours* , in '*Flows: Nature's Patterns: A Tapestry in Three Parts*' - chapter 5, 2011, Oxford University Press.
- Franck Varenne.- *What does a computer simulation prove ?* in '*Simulation in Industry*' Proc of the 13th European Simulation Symposium, Marseille, France, October 18-20th, 2001, edité par N. Giambiasi et C. Frydman, SCS Europe Bvba, Ghent,
- Stephen Wolfram, *How Do Simple Programs Behaves* ? in *'Programming Cultures, Architecture, Art and Science in the Age of Software Development'*, Architectural Design Volume 76, n°4, july/august 2006, Wiley Press, London.

Usage des Algorithmes à des fins spéculatives

- Alisa Andrasek, '*Biothing'*, editions HYX Orléans, 2009.
- Matthias Kohler.- *Aerial Architecture* in '*Reclaim Resi[lience]stance // ……R²* Log°25, op. cit.
- Mike Silver, *Building Without Drawings : Automason Ver 1.0* in '*Programming Cultures, Architecture,*

Art and Science in the Age of Software Development', Architectural Design Volume 76, n°4, july/august 2006, Wiley Press, London.

- Roland Snooks.- *Volatile Formation*, in '*Reclaim Resi[lience]stance // ……R²*' Log°25, summer 2012, edité par AnyCorporation and MIT Press,

- Theodore Spyropoulos, *Constructing Adaptive Ecologies : notes on a computational urbanism*. 2013. Not yet published

Usage des Algorithmes à des fins d'optimisation

- Christian Derix et Asmund Izaki, *Spatial computing for the new organic* in '*Computation Work, The Building Of An Algorithmique thought*' Architectural Design, Profile n°222, mars/avril 2013, Wiley Press, London.

- Michael Hensel, Achim Menges .- '*Morpho-ecologies'*, Architectural Association Press, Londres, 2006

- Michael Hensel, Achim Menges, Michael Weinstock.- *Morphogenesis and Emergence (2004-2006*) in *'The Digital Turn in Architecture 1992-2012* ' édité par Mario Carpo pour Wiley Press, 2012, Paris.

- Neil Leach .- *Swarm urbanism*, in '*Digital Cities*' Architectural Design Vol°79, n°4, july/august 2009, Wiley Press London

- Alexander Schifner, Mathias Höbinger, Johannes Wallner, Helmut Pottman.- Packing Circle and Sphere on Surfaces, à paraitre dans le cycle de conférence ACM SIGGRAPH. www.geometrie.tugraz.at/wallner/packing.pdf (consulté le 12 décembre 2012)

- Fabian Scheurer.- Getting complexity organised Using self organisation in architectural construction in 'Automation in Construction n°16', Elsevier, 2007

- Fabian Scheurer.- *A simulation toolbox for self organisation in architectural design* in '*Innovation in Architecture, Engineering and Computing (AEC)*', ed; Sevil Sariyildiz et Bige Tuncer, Rotterdam, NL : Delft University of Technology, Faculty of Architecture, 2005.

- Fabian Scheurer.- *Signal to Noise* in T. Valena, T. Avermaete, G. Vrachliotis (ed.), *'Structuralism Reloadad: Rule Based Design in Architecture and Urbanism'*, p. 269-274, Edition Axel Menges, Stuttgart/London 2011

- Hanno Stehling & Fabian Scheurer.- *Lost in parameter*, in '*Mathematic of space'*, Architectural Design Volume 81, n°4, july/august 2011, Wiley Press, London.

Fabrication

- Karola Dierichs, Achim Menges.- *Aggregate Structure* in '*Material Computation'*, Architectural Design Profile n° 216, March/April 2012, Wiley Press, London.

- Karola Dierichs, *Material Computation* in '*Acadia 2010, Life in : formation, proceedings of the 30th Annual Conference of Association for Computer Aided Design in Architecture'*, Aaron Sprecher, New York 2010.

- Jelle Feringa.- *Implicit Fabrication*, Acadia 2012

- Matthias Kohler.- *Aerial Architecture* in '*Reclaim Resi[lience]stance // ……R²*' Log°25, op. cit.

- Gramazio & Kholeer et Raffaello d'Andrea.- *Flight Assembly Architecture*, editions HYX, Orléans, 2012.

- Hanno Stehling, Fabian Scheurer & Jean Roulier.- *Bridging the gap from CAD to CAM* paper submitted for the Fabricate conference, October 2013, not published yet

- Skylar Tibbits, *Design To Self-Assembly*, in '*Material Computation'* Architectural Design Profile n° 216, March/April 2012, Wiley Press, London.