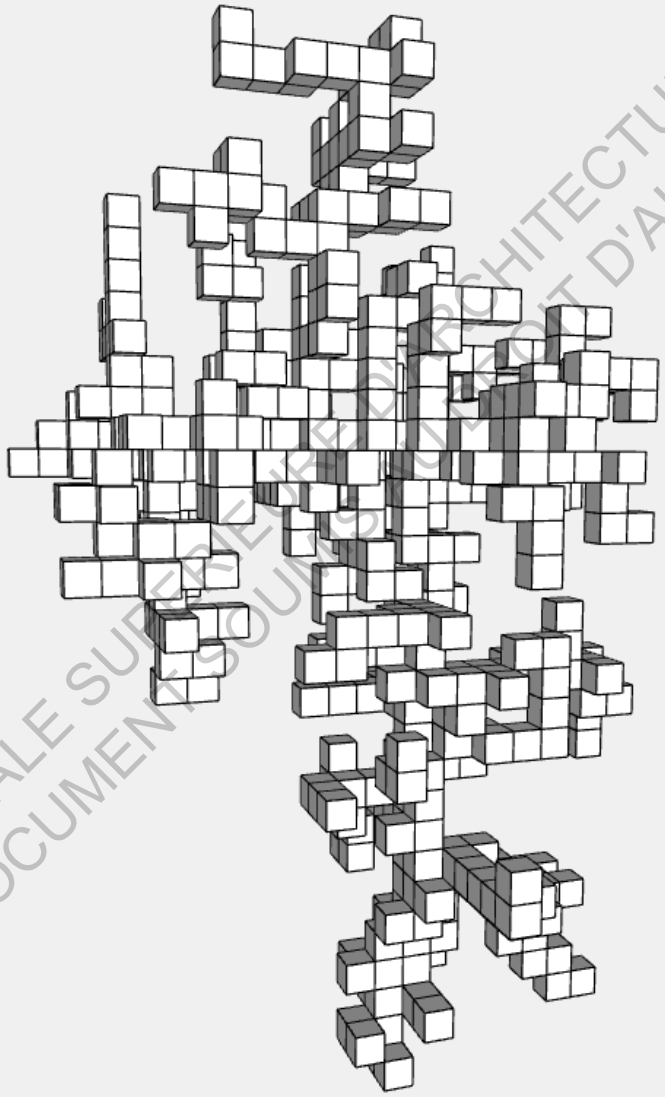
Architecture et code : programmer des systèmes de génération de formes

Éliot Marin-Cudraz

ARCHITECTURE ET CODE

Programmer des systèmes de génération de forme



Eliot Marin-Cudraz - *ENSA Nantes mémoire 2015*
Enseignants *Benoît Boris et Francis Miguet*

SOMMAIRE

1.1 Avant-propos

Ce mémoire s'appuit sur plusieurs expériences personnelles, notamment l'année de master 1 à la Dessau Institute of Architecture en Allemagne (2013-2014) durant laquelle mon intérêt pour le design algorithmique s'est développé.

De plus trois évènements viennent le nourir, directemement et indirectement, le premier « European Meetings on Cybernetics and System Research» ou EMCSR à Vienne en avril 2014, le second « IS4IS Summit Vienna 2015» en juin 2015, ainsi que « ISSS Berlin », en août 2015. Le principal apport de ces conférences réside dans les théories des systèmes, la pensée systémique et l'approche des système par la cybernétique.

1.2 Objectifs du mémoire

Les nouveaux outils de l'architecte ?

L'objectif de ce mémoire est de discuter les enjeux de conception sous-jacent à l'approche paramétrique et algorithmique de l'architecture. De ce fait, traiter du changement de paradigme par rapport à une approche traditionnelle représentative, c'est-à-dire par le dessin. Quelle nouvelle relation entretient l'architecte vis à vis de ces outils ? Comment cette démarche conceptuelle peut faire émerger de nouvelles façons de dessiner un projet ? Quelle pertinence y-a-t-il à adopter cette approche ?

En matière de conception architecturale, il n'existe pas de méthode universelle, ni d'outil indispensable. L'architecte reste un concepteur d'espace, architectural et urbain, dont la finalité est de produire des documents en vue de la construction du projet. Néanmoins nous devons noter que la technologie a toujours influencé l'architecte, soucieux de mettre en relation les problématiques de son époque, et de son projet, en lien avec les ressources qui sont à sa disposition. Par ressources nous entendons les matériaux, les moyens de fabrication, les outils de conception et de représentation ainsi que tout autre élément susceptible d'inspirer l'architecte.

Si a priori, il n'y a autant de méthode de conception que d'architecte, le rôle de l'architecte reste de transformer progressivement une demande, une problématique vers une solution, une réponse. Cela implique un dialogue, à chaque étape du projet avec des intervenants de différents domaines, de l'étude jusqu'à la réalisation. Et c'est pour cela que l'architecte doit faire avec les moyens et attentes de son temps, ne serait-ce que pour communiquer le projet. D'une part, il est totalement libre d'expérimenter ou d'améliorer des méthodes existantes, mais d'autre part il doit rester dans un cadre adapté à ses interlocuteurs, client comme partenaires.

La technologie et les sciences ont toujours fait progresser les outils à la disposition des architectes, sans pour autant changer leur but, concevoir des bâtiments. Chaque outil cependant implique deux choses, la première est une tendance à produire un certain résultat, comme un compas est plus utile pour tracer des cercles que des carrés, et la deuxième est un état d'esprit, c'est-à-dire qu'il induit un cadre et un vocabulaire dans lequel le concepteur doit penser. Pour reprendre l'exemple du cercle, le compas n'est qu'un outil parmi d'autre pour exécuter cette tache. Nous allons nous intéresser au principe de formation, c'est-à-dire au processus de génération de forme, plutôt qu'à la forme dans ce qu'elle représente. C'est pourquoi nous avons choisi de présenter une approche par la programmation, afin de mieux décrypter ces processus.

« By the time of the conference held by Boston Architectural Center in 1964, it had become clear that the electronic era would have a dramatic effect on building design. The aerospace industries were using computers to calculate complex warped surfaces and animated flight path simulations, which fascinated architects » AIA CC 2012

1.3 De nouvelles approches

Evolution technologique

Depuis toujours l'architecture entretient une relation particulière avec la technologie, au travers d'une approche à la fois esthétique et pratique. C'est le cas notamment avec l'impact de l'électricité sur les bâtiments, à la fois dans leur aspect, notre manière de les concevoir, et le scénario d'utilisation. Il faut bien distinguer les termes technique, et technologie. Le premier se définit par un but, comme faire cuire un aliment, et les technologies en relation définissent la méthode, par le feu, par l'électricité ou tout autre moyen développé à partir du même but. Aujourd'hui ce sont les outils paramétriques et algorithmiques, comme nouvelle technologie, qui transforment la conception, ainsi que l'esthétique de l'architecture. Cependant il faut noter que ces approches ne représentent qu'une seule parmi d'autres.

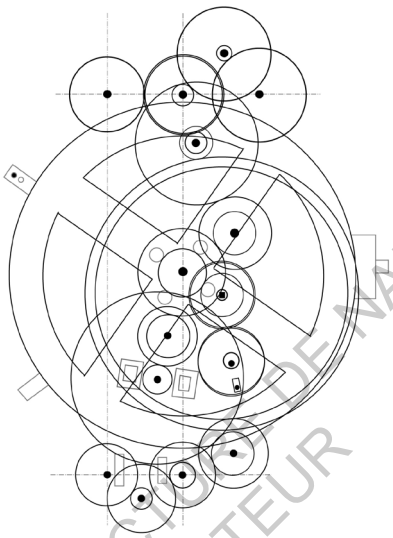
Après la seconde guerre mondiale commencait le temps de la reconstruction, cette période ainsi que les décénies qui ont suivis ont du traiter la problématique de la construction rapide et économique, mais surtout en masse. Cette approche a été l'occasion de mettre en pratique les théories héritées d'architectes modernes comme Le Corbusier, mais surtout de faire une table rase sur un contexte urbain dévasté.

Aujourd'hui, les villes ont eu le temps de se réadapter, mais elles ont du faire face à un patrimoine et des formes urbaines complexes. Ce contexte ainsi que les politiques de limitation de l'urbanisation tendent à fournir aux architectes un travail qui s'inscrit dans des géométries de parcelles complexe. Cela induit des réponses innovantes en terme de conception, ou du moins un défi en terme d'implantation et de morphologie du projet. On peut mettre cela en lien avec le développement des outils paramétriques, qui tendent à être performants sur des projets complexes, en terme de modularité, d'interface et de visualisation, pour reconstituer les causes de l'émergence de cette approche conceptuelle.

Nous verrons aussi que les modèles paramétriques posent souvent un problème de flexibilité, en effet l'adaptabilité requiert beaucoup plus de travail que ce qu'il n'y paraît.

HISTOIRE ET DÉFINITIONS

« There is no more greater error in science, than to believe that just because mathematical calculation has been completed, some aspect of Nature is certain » ***Alfred North Whitehead (1953)***



Reconsitution du méchanisme d'Antikythera

2.1 Un héritage lointain
Definitions et origines

Le mot algorithme vient du nom du mathématicien Al-khwarizmi traduit en latin par algoritmi. Vers -300 av. J.-C. Euclide formula un algorithme célèbre pour trouver le plus grand diviseur commun de deux nombres, en réduisant les étapes au maximum. Nous allons voir l'évolution plus en détail mais c'est dans les années 1930, mais ce sont les travaux de d'Alan Turing et Alonzo Church qui aident à préciser la notion d'algorithme. La machine de Turing, capable de lire et d'écrire en binaire, pose les bases de ce qui deviendra l'ordinateur et l'informatique d'aujourd'hui.

Deux définitions :

* Un algorithme est une méthode de résolution d'un problème en suivant un ensemble de règles. Les algorithmes sont essentiels pour l'informatique, et la plupart des appareils électroniques s'en servent.

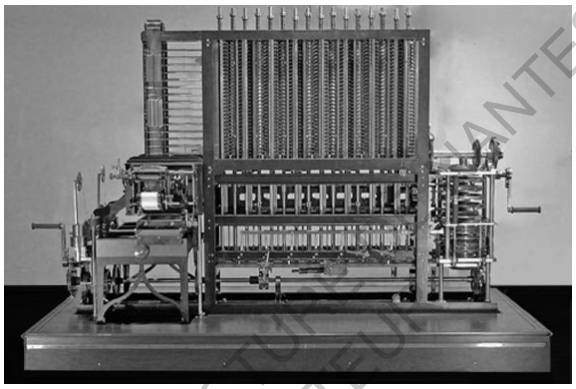
** Les algorithmes sont conçus comme des procédures théoriques pour mettre à exécution les tâches mathématiques. On les utilise constamment dans tous les ordinateurs du monde. Un algorithme n'est pas compliqué : il est juste une liste d'instructions mettant à exécution une tâche ou chaque étape est complètement sans ambiguité. Il peut être mis à exécution par un agent irréfléchi.

Le méchanisme d'Antikythera

Alors qu'Alan Turing est considéré comme un pionnier dans le développement des premiers ordinateurs, une récente découverte nous permet de dire que les Grecs avaient déjà formalisés des systèmes de calcul.

Le méchanisme d'Antikythera vieux de deux mille ans, trouvé lors d'une fouille archéologique en mer en 1901, est considéré comme le premier ordinateur créer par l'homme. Reconstitué seulement depuis quelques années, il s'agit d'un outil astronomique qui permet de visualiser la positions des astres du système solaire à un temps donné, et donc de prédire les éclipses à l'heure près.

* Mathématiques minutes, ed contre dires. ** 3 minutes de R.Brown



La machine de Babbage, construction posthume

2.2 Les origines de l'informatiques

1700 - 1960

Le développement de l'informatique joue un rôle important dans l'évolution de la pratique architecturale, notamment au niveau des outils de conception et de représentation. Nous allons voir quelles étapes clés, qui aboutirent ou non, ont permis à la fois de développer une pensée propre à l'informatique et de produire des outils et des machines qui fonctionnent sur son principe.

De 1700 à 1937

Le système binaire apparaît dans l'histoire dans différentes civilisations, égyptienne, chinoise et indienne, mais c'est surtout Leibniz au début du XVIIIe siècle qui favorisa sont développement en publiant une description du système binaire, dans lequel tous les nombres sont exprimés avec les unités 0 ou 1.

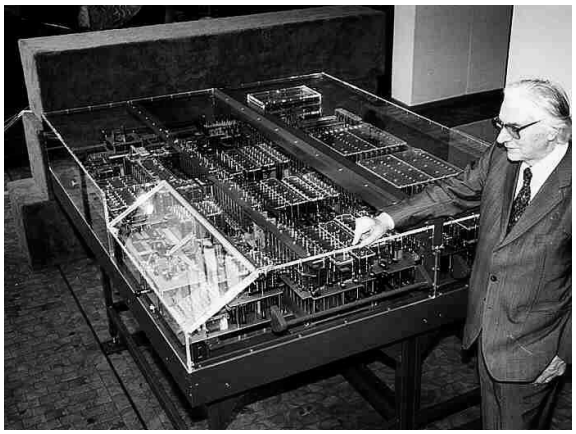
En 1833 Le mathématicien Charles Babbage imagine la "machine analytique" une machine de calcul entièrement automatique. Cependant son projet sera trop ambitieux et après 30 ans de travail il ne terminera jamais la machine, trop complexe pour l'époque à fabriquer.

Deux personnes à quelques années d'intervalle vont permettre de cristalliser des principes. D'abord en 1843 Ada Lovelace, mathématicienne, revisite le concept de Babbage. Elle écrit un algorithme relatif aux nombres de Bernoulli, qui est considéré comme le premier programme informatique. Le langage ADA, développé en 1980 lui fait référence, il est notamment utilisé dans les systèmes embarqués en aéronautique. Ensuite c'est George Boole qui en 1847 va établir des principes fondamentaux en programmation. Il met au point le " calcul propositionnel", et introduit les opérateurs logiques "and, or, not" qui permettent de déclarer des conditions. L'algèbre Booléenne est aujourd'hui au centre de la programmation informatique.

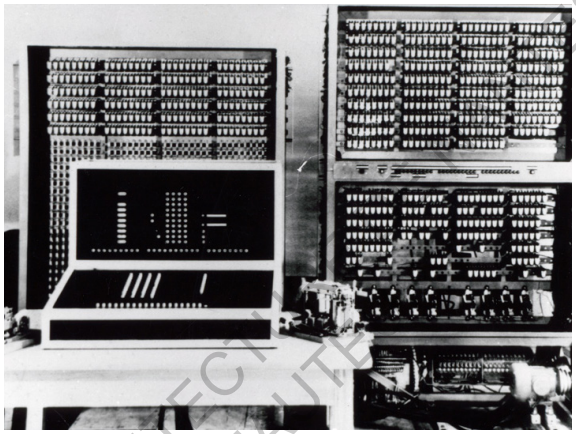
Herman Hollerith créer en 1886 une machine lisant les cartes perforées, elle sera utilisée pour faciliter le travail de recensement aux Etats-Unis. Grâce à l'automatisation de la machine, le traitement des données de population fut réduit de sept à deux ans. La technique des cartes perforées sera réinvestie plus tard dans les premiers ordinateurs, et utilisée dans les écoles comme support d'enseignement.

De 1937 à 1960

Ces travaux pour l'instant assez isolés vont trouver leur place dans les recherches d'Alan Turing et de Konrad Zuse. En 1937 Turing publie la théorie d'un ordinateur universel. Il prouve que les ordinateurs sont capables, en principe de résoudre n'importe quel problème mathématique qui peut



L'ordinateur Z1 et Konrad Zuse



L'ordinateur Z3

être exprimé de manière algorithmique. L'année d'après, en 1938 l'ingénieur Konrad Zuse achève le Z1, la première machine à calcul au monde à être programmable. Elle est entièrement basée sur le système binaire et inclut les nombres décimaux. La machine est cependant limitée à cause du manque de précision de certains composants. Trois ans plus tard Zuse termine le Z3, la nouvelle version de son ordinateur. Les circuits de la machine utilisent alors des transmissions électromagnétiques. Les composants de l'ordinateur moderne sont là, mais les programmes sont toujours externes à la machine.

Pendant la seconde guerre mondiale, les ingénieurs sont employés dans la recherche militaire, une course se met en marche entre les alliés et les troupes Allemandes notamment pour tenter de décoder les messages de l'autre camp. En 1944 les cryptographes Anglais mettent au point le Colossus, une machine qui leur permet de décrypter les messages de l'armée Allemande. Cette nouvelle machine peut traiter 5000 caractères par seconde. La même année à l'université de Harvard, Howard Aiken présente le Mark 1, une unité centrale de 17 mètres de long capable d'effectuer une addition en 0.3 seconde.La puissance de calcul des machines augmente progressivement, mais elles ne sont pas encore au point. En 1945 John von Neumann propose une théorie dans laquelle les programmes informatiques devraient être stockés dans la mémoire de l'ordinateur comme des données, ce qui permettrais à la machine d'être plus rapide et aux programmes facilement modifiables. Un an plus tard, les Etats-Unis mettent au point le ENIAC, le premier ordinateur entièrement électronique, il est 1000 fois plus rapide que les précédents modèles, cependant il faut plusieurs jours pour le reprogrammer.

C'est alors en Angleterre en 1948 qu'apparaissent les ordinateurs dont les programmes et fichiers sont stockés et éditables dans la mémoire interne de la machine. Au même moment Claude Shannon, un mathématicien Américain, est le premier à utiliser le terme « bit » (Binary digit) qui définit la plus petite unité d'information. Il explique que tout type d'information peut être exprimé en « bits »

A partir de là les innovations en matière d'informatique vont se multiplier, on peut noter l'arrivée du premier moniteur en 1951, afin de visualiser l'espace aérien Américain. On remplacera ensuite les tubes à vide par des transistors, permettant de rendre l'ordinateur plus petit. L'informatique devient ensuite modulaire à la fin des années 1950, grâce à l'ingénieur Jack Kilby, principe qui sera repris par la société IBM quelques années plus tard.

L'unité principale du Commodore C64

Ce sont donc les années 1960 qui marquent l'entrée de l'ordinateur dans les universités, dans les entreprises. Vingt ans plus tard le lancement du Commodore C64 permettra au grand public d'avoir accès à l'informatique, il sera vendu à 30 millions d'exemplaires.

Nous l'avons vu, la mise au point de machines, programmables et capables de calculer plus vite que le cerveau humain ne provient pas d'une démarche unique et linéaire mais s'est fait progressivement autour des notions de programmation, ou la définition d'une suite d'étapes précices comme un algorithme, et celle d'automatisation.

L'outil informatique est devenu un élément majeur et indispensable dans tous les domaines car il rend possible la création de nouveaux outils, les programmes. Nous allons donc voir dans quelles circonstances se sont développés les logiciels de CAO et l'interaction homme-machine.

« Sutherland's CAD innovation showed how design could be modified to meet the needs of contextual changes, which anticipated the possibilities within parametric architecture » ***R.Nandha***

Ivan Sutherland manipulant l'interface qui donnera naissance aux logiciels de CAO.

2.3 Evolution de la CAO

Vers plus d'interaction homme-machine

Depuis le développement de l'informatique, les logiciels de CAO ont progressivement remplacés les tables à dessin dans le travail des architectes. Plus récemment, deux domaines ont émergés, le design paramétrique et le design algorithmique, ouvrant de nouvelles voies dans la conception de bâtiments complexes.

Le premier outil CA0, a été développé en 1963 par Ivan Sutherland, au même moment D.Engelbart mettait au point l'interface « WIMP » qui nous sert toujours aujourd'hui. C'est-à-dire *windows, icon, menu* et *pointer.*

Ces outils ont d'abord vu le jour sous la forme de groupe de recherches au sein d'agences comme Foster & Partners, Gehry Partners, Arup et Zaha Hadid Architects, par le biais de groupes internes de recherche, respectivement : Specialist Modelling Group, Gehry Technologies, Advanced Geometry Unit et CODE.

Une grande confusion entoure les notions de design par ordinateur (computational design), design paramétrique et design algorithmique, des amalgames entre les méthodes et les logiciels sont souvent la source d'incompréhension. Il faut bien distinguer un outil de dessin numérique, avec lequel on vient représenter en 2D ou 3D des éléments, avec un logiciel paramétrique avec lequel on défini des emplacements, des caractéristiques et des relations entre des éléments.
Dans le premier cas on dessine par exemple un mur, dans le second on définit l'emplacement du mur et à côté on vient rentrer des paramètres d'épaisseur, de hauteur, de longueur, il se créer alors un lien entre l'élément et les paramètres. La notion la plus importante est qu'en associant les éléments entre eux, la modification d'un élément a une influence sur l'ensemble, par exemple un lien entre la hauteur du mur et la position des planchers d'étage.

Si le résultat peut être le même, la méthode est bien différente, ainsi que le temps passé si l'on considère la conception d'un bâtiment complexe. Une confusion existe aussi entre l'aspect ou l'esthétique, d'un projet et les outils mis en oeuvre. Nous verrons que l'usage d'algorithmes n'entraine pas nécessairement une forme de projet curviligne.

L'utilisation des logiciels et leurs avancées génèrent tout un vocabulaire de termes techniques qu'il est parfois difficile à comprendre en raison du manque de précision dans l'usage, alors qu'a priori ils ne tombent pas sous le sens.

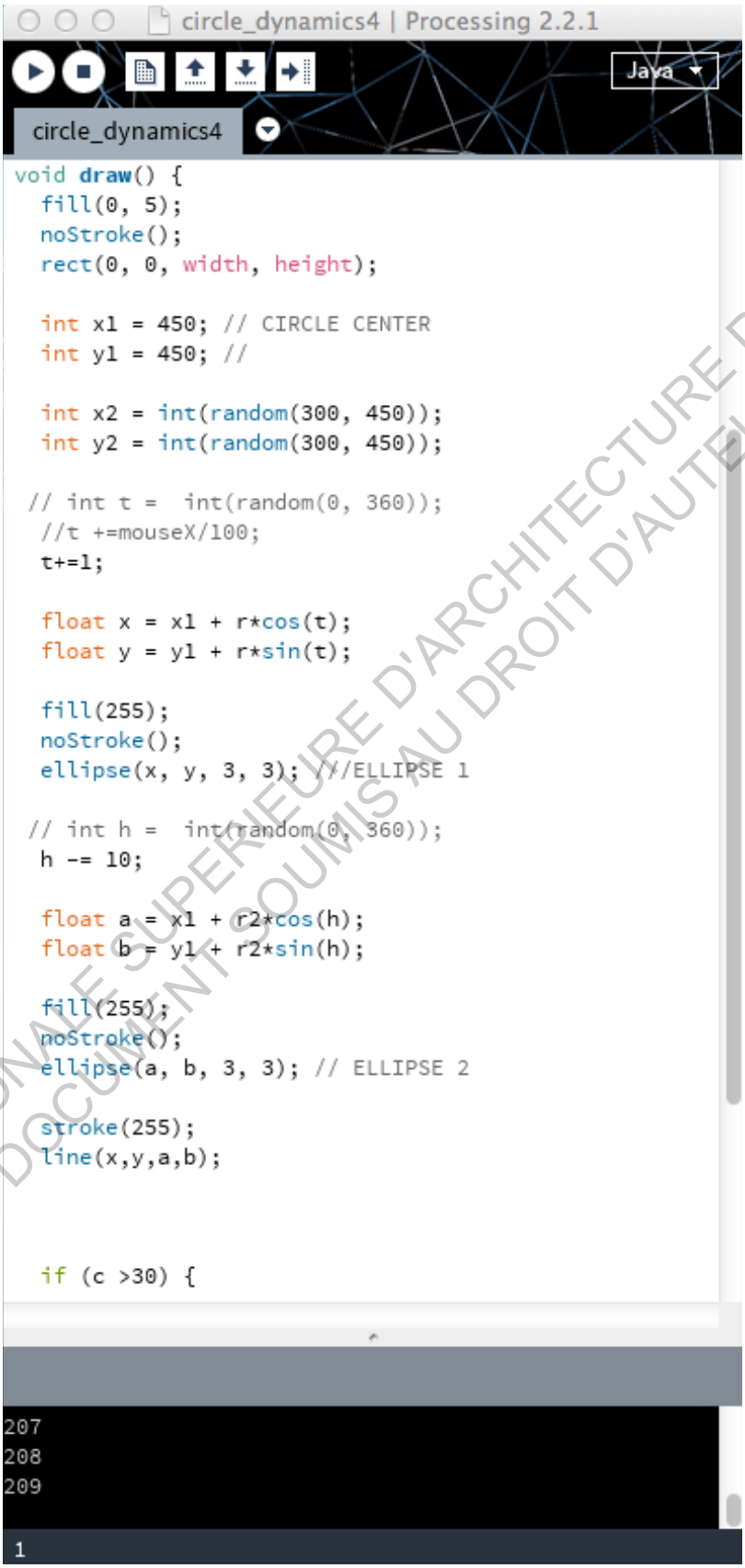
*« The computer didn't invent parametric design, nor did it redefine architecture or the profession ; it did provide a valuable tool that has since enabled architects to design and construct innovative buildings with more axacting qualitative anq quantitative conditions » **AIA CC***

2.4 Design paramétrique

Une notion ambigüe

Faire du design paramétrique signifie travailler avec des paramètres. Mais par extension cela fait référence à l'utilisation d'un logiciel paramétrique. Cet outil facilite notamment le travail avec des courbes en 3D, sans pour autant l'imposer, les principaux avantages sur une méthode traditionnelle ou manuelle sont la possibilité de modifier à l'infini, l'efficacité en terme de temps, ainsi que la précision, indispensable pour une fabrication. Alors que le terme désigne une méthode, ou process, il est largement employé à tort pour parler d'un style d'architecture.

Un algorithme, nous l'avons vu, suit des instructions, il peut s'inscrire dans un processus de design aussi bien traditionnel que numérique. Cependant l'ordinateur est capable de réaliser des instructions en boucle, qu'il est impensable de reproduire de manière analogue.Les logiciels Grasshopper, Processing, Rhino scripting ou Unity3D sont des outils algorithmiques et non paramétriques. Ils sont basés sur l'utilisation de code, de langage de programmation. Alors que les outils paramétriques sont basés sur la manipulation de formes. Malgré que ces outils soient différents, ils peuvent être utilisés ensemble, par exemple pour générer une forme de base par un algorithme, et ensuite la manipuler en tant que géométrie. Mais l'inverse est aussi possible, on peut affiner ou optimiser une géométrie existante à l'aide d'un algorithme. En effet on vient générer un nombre de viariantes d'une même forme afin de choisir la plus pertinente. L'approche de la conception par le design paramétrique permet entre autre de passer par une plateforme capable d'intégrer une grande variété de contraintes et de questions, que la simple représentation ignore. En effet, les bâtiments se complexifient en terme de structure, de fluides, de matériaux et d'intégration d'éléments techniques. A terme, c'est une approche BIM qui est envisagée pour la conception des projets, afin de partager la plateforme de conception et les outils d'analyse entre toutes les entreprises reliées au projet. Les architectes sont fascinés par la géométrie et les structures complexes inspirées de la nature, à l'échelle micro, ainsi que des espaces que cela produit à l'échelle macro. Ils partent du principe que si la nature se comporte d'une certaine manière, il y a alors des principes d'efficacité et de logique à comprendre et à réinvestir ailleurs.C'est à partir de 1982, avec le travail de Benoit Mandelbrot sur les fractals, et celui de K.J. Falconer en 1990 que l'ordinateur est devenu un outil adapté à la simulation de formes biologiques, dans un processus de morphogenèse. Cette recherche scientifique a inspiré beaucoup d'architectes et de designer dans leur démarche, dès les années 1980. Par exemple Greg Lynn a développé les typologies du « Blob » et du « Fold » dans l'architecture grâce aux systèmes et au code mis au point.



«Script rather than model»
P. Schumacher

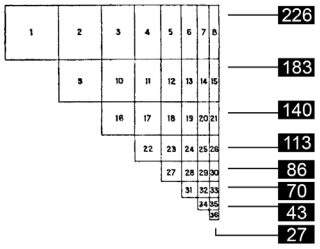
2.5 Pourquoi programmer?
A la frontière de l'architecture

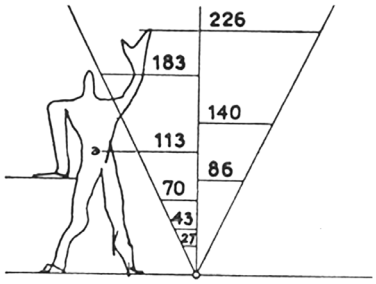
L'objectif est de mettre en avant l'intérêt d'une pratique nouvelle au sein d'un métier en perpétuelle re-questionnement, mais aussi d'apporter une critique sur la place que peuvent avoir ces outils dans un univers académique et professionnel.

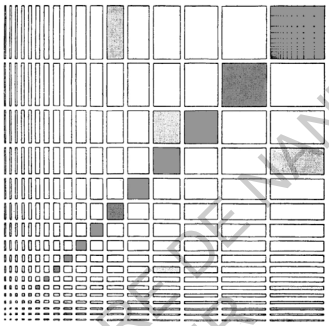
Apprendre un langage de programmation n'est pas un objectif de la formation initiale d'architecte, on imagine plus cette pratique dans des métiers de l'ingénierie du bâtiment. Que ce soit dans un bureau d'étude structure, fluide ou thermique, les algorithmes et la programmation trouvent leur place aisément dans un milieu de calculs et de mathématiques. Avant tout programmer c'est utiliser le langage d'une machine, d'un ordinateur, pour lui donner des ordres, afin décider d'actions à effectuer. De plus programmer c'est définir des variables, et les mettre en place dans un processus de calcul complexe qui serait impossible à reproduire à la main.

Le premier lien que l'on peut faire entre l'architecture et la programmation réside dans les manipulations géométriques. En effet il est possible de transposer n'importe quelle forme en code, en utilisant les propriétés de la géométrie classique. On peut traduire ainsi un cercle, un carré ou un triangle ou des éléments plus complexes comme une sphère ou un dodécaèdre, si l'on prend des figures régulières. Ces figures deviennent vraiment intéressante lorsqu'on les utilise comme éléments de base, ou en les exprimant grâce à des éléments plus simple qui sont les points, les arêtes et les faces qui vont servir de levier afin de transformer la géométrie.

Développer le projet en passant par de la programmation, du code, plutôt que de la modélisation directe, oblige le concepteur à exprimer ses intentions en tant que règles simples et ainsi permettre de filtrer toutes les questions de forme. La modélisation oblige à prendre des décisions assez rapidement, le code est donc, à l'instar du croquis, une manière d'exprimer des idées en se laissant un espace de liberté. Nous verrons plus loin comment l'établissement de règles définit les bases d'une méthode systémique.







2.6 L'unité d'habitation

Approche systémique de Le Corbusier

Nous allons voir comment la méthode mise en place par Le Corbusier dans son travail sur l'Unité d'habitation de Marseille, fait écho à des démarches que l'on entreprend aujourd'hui en design algorithmique. Le principe est simple, on cherche à faire émerger le projet en définissant des règles de base qui seront déclinées dans l'ensemble du bâtiment.

Le Corbusier développe l'idée du Modulor en cherchant les proportions parfaites en architecture. Il souhaite déterminer l'échelle des espaces en fonction du corps humain. Le nombre d'or, appellé Phi est environ égal à 1,618. Si l'on divise le plus grand côté d'un rectangle d'or par la longueur de son autre côté, on obtient Phi.

Le Corbusier applique ce rapport à une figure humaine idéale qu'il nomme Modulor, par exemple en divisant la hauteur du Modulor par Phi, on obtient la hauteur du nombril. On le voit en regardant le schéma ci dessous, 183 / 1,618 = 113,10.

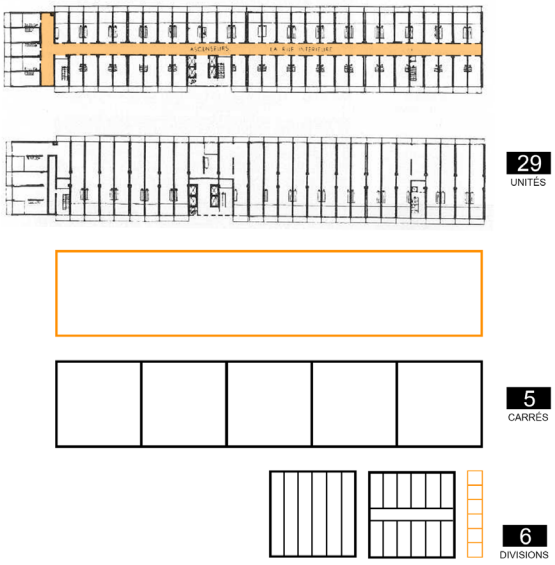
En établissant ce principe, Le Corbusier décline sa recherche sur le nombre d'or en grille, qui servira à proportionner les espaces de son Unité d'habitation. Il démarre par un carré de 226 cm de côté, étant la hauteur du Modulor le bras levé, qui est aussi le double de 113, la hauteur du nombril.

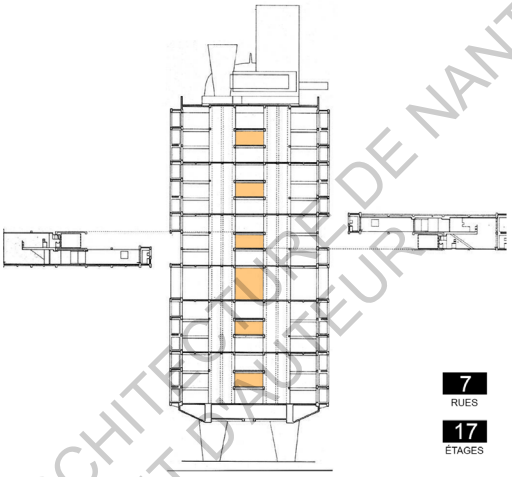
Le plan de l'Unité d'habitation se base sur la grille précédement évoquée pour définir la trame et proportions du bâtiment.

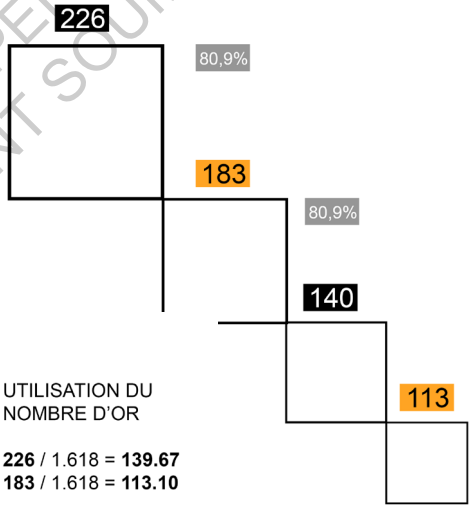
Cette approche démontre bien comment chaque élément du projet est connecté avec les autres par un système de répercusion et d'application de la grille. En effet, un changement de taille du modulor de quelques centimètres entrainerais un bouleversement du projet dans son ensemble.

On comprend bien que Le Corbusier ne s'intéressait pas vraiment au dimensionnement exacte de chaque élément de l'Unité, mais qu'il cherchait à mettre en place une démarche que l'on pourrait qualifier aujourd'hui de « paramétrique », ainsi il a cherché à d'abord concevoir un système qui dicterais les relations des éléments les uns par rapports au autres.

Le système qu'il met en place peut être qualifié de fermé, c'est-à--dire que rien n'est censé bouger ou être modifié sans quoi l'ensemble perdrait sa cohérence. Nous allons voir dans le prochain exemple comment Cedric Price a pensé l'inverse.

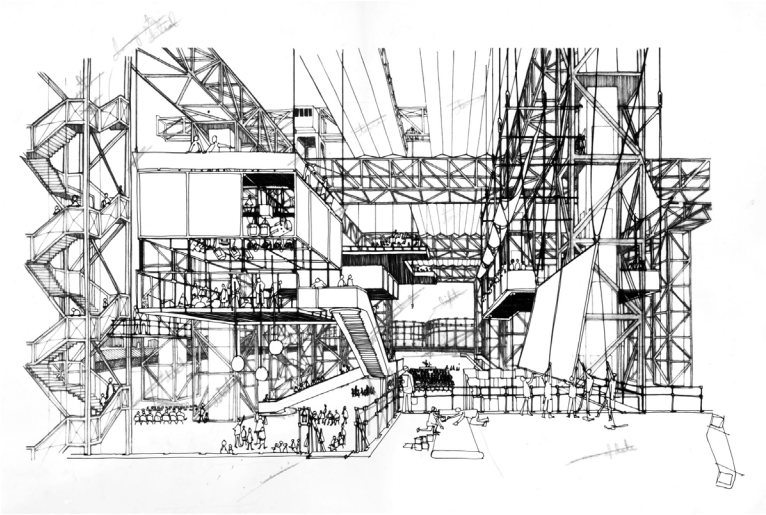






UTILISATION DU
NOMBRE D'OR

226 / 1.618 = **139.67**
183 / 1.618 = **113.10**



2.7 Le Fun Palace

Approche systémique de Cedric Price

Le Fun Palace de Cedric Price (1934-2003) est un projet des années 1960, qui ne sera pas construit, d'un théâtre dans lequel le public joue autant un rôle que les acteurs. Cela illustre une vision collaborative d'une architecture dans laquelle les murs et les planchers bougent comme un décor. Le projet est conçu comme un système, en effet pour Price, l'architecture est un système, composé de tuyaux, de ventilation, de machines, de cables. Il est donc considéré comme un « anti bâtiment » car son processus d'utilisation implique qu'il soit monté et démonté en permanence, suivant l'événement.

Le projet devait s'installer sur les rives de la Tamise, afin de réinvestir des zones industriels abandonnées pour les transformer en espaces culturels, vision qui, pour l'époque, était complètement nouvelle. Le Fun Palace est ambitieux, avec une capacité d'accueil de plus de 50 000 personnes, il se compose de 75 tours d'acier, exosquelette destiné à supporté les transformations du bâtiment.

Le projet intègre la notion de cycle à plusieurs niveaux, d'abord dans la reconfiguration de ses usages, mais aussi dans sa pérennité. En effet après 10 années d'exploitation l'objectif était de déconstruire (et non détruire) le bâtiment pour laisser la place à de nouveaux projets.

Le Fun Palace est dessiné comme un bâtiment en mouvement, c'est-à-dire qui se repense constamment dans son organisation. Price imagine des panneaux et des murs amovibles afin de pouvoir reconfigurer l'espace pour différentes occasions. L'idée derrière cela est que l'utilisateur et l'usage redéfinit à chaque fois le bâtiment, Price suggère donc à la fois une durée de vie très courte de l'architecture mais aussi une renaissance permanente. De ce fait l'habitant, l'usager fait partie du système généré par le Fun Palace.

Le projet est pensé comme un ensemble de systèmes dépendants des uns des autres, si l'un est défaillant, le reste ne peut plus fonctionner. Pour Cedric Price, un bâtiment doit correspondre aux besoins de ses usagers et être transformé ou démoli si ce n'est plus le cas. Alors que Le Corbusier suggérait un système fermé basé sur son modulor, Price cherche à développer une architecture qui soit un système ouvert à l'émergence et à la créativité des usagers.

Le projet a notamment inspiré Rodgers et Piano dans leur conception du Centre Pompidou (1977) à Paris.

« It is about designing tools that people themselves may use to construct – in the widest sense of the word – their environments and as a result build their own sense of agency. It is about developing ways in which people themselves can become more engaged with, and ultimately responsible for, the spaces they inhabit. It is about investing the production of architecture with the poetries of its inhabitants »

Gordon Pask - *collaborateur de Cedric Price*

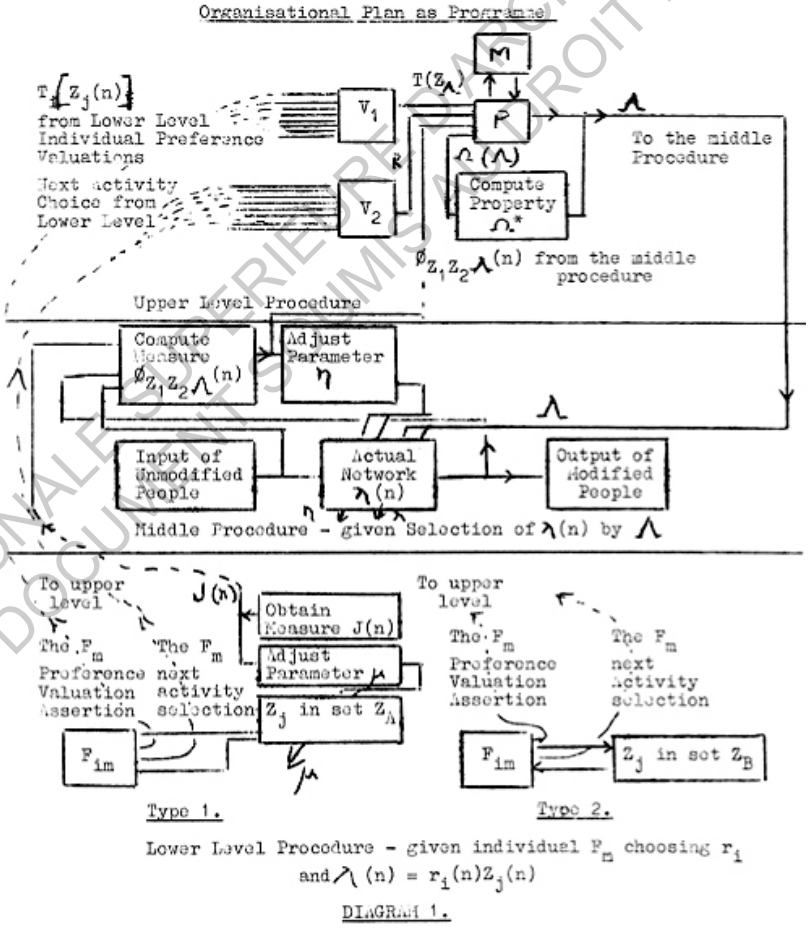
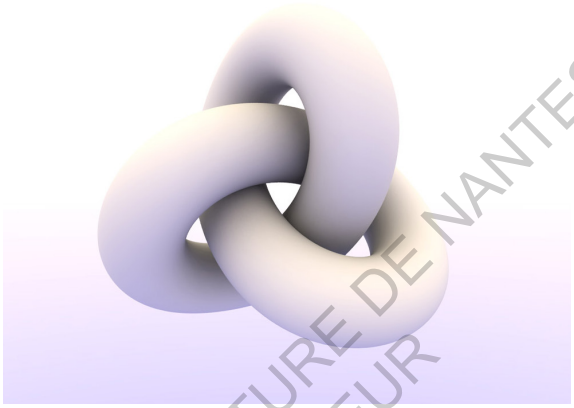


DIAGRAM 1.

Schéma pour le Fun Palace, ou l'architecture est pensée comme un système.



2.8 Musée Mercedes-Benz

L'approche formelle de UNStudio

Le projet du musée Mercedes-Benz à Stuttgart, livré en 2006, a été conçu suivant un principe de « noeud de trèfle » une sorte d'entrelac. Le musée s'organise comme un parcours composé de deux circuits continus qui s'enroulent autour d'un atrium

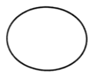
Ce projet va nous servir de contre exemple, en effet nous devons faire la distinction entre l'inspiration d'un motif géométrique, avec les valeurs d'estimes qu'il véhicule, ici en rapport avec le logo de la marque, et ce qui pourrait sembler être une application itérative de la fonction qui définit le noeud de trèfle. Les architectes ont établi ici un principe de rampe inspiré de ce motif, mais en aucun cas le processus de conception définit un système ou un ensemble d'éléments dépendant d'une fonction.

La figure ci-contre est extraite du livre Morphing* et traite justement du projet d'UNStudio. Elle présente une déclinaison de fonctions mathémétiques qui illustrent la transformation d'une ellipse en noeud de trèfle suivant différentes étapes. Bien que l'on puisse effectivement reconnaître les propriétés qui ont intéressées les architectes, c'est la forme qui a été mise la plus en valeur.

Le noeud de trèfle comme on le voit sur l'image ci-dessus se définit par des caractéristiques bien spéciales, tout d'abord à aucun moment il ne se touche lui même dans la boucle. Les plateformes du musée sont elles dessinées pour qu'il y ai des points de rencontre*. Puis à l'instar d'un anneau de Moebius, il définit une trajectoire unique en boucle de laquelle il est impossible de sortir. Ces propriétés sont intrinsèques, quelque soit le degré de courbure, l'épaiseur qu'on lui donne ou les variations formelles, car quelque soit l'interprêtation la fonction mathémétique ne change pas.

L'objectif ici n'est pas de dévaloriser le projet en lui même, mais de pointer le fait que trop souvent la forme ou l'apparence est au centre du discours quand en réalité ce sont les principes de formations et les caractéristiques de cette forme qui devraient attirer notre attention en tant que concepteurs. Cette notion sera développée dans la partie sur les expérimentations et le code.

**«The two main trajectories, (...) spiral downwards on the perimeter of the display platforms, intersecting with each other at several points allowing the visitor to change routes.»*
UNStudio

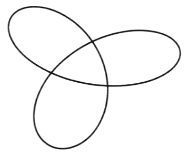


$\{ u \mid 0 \le u \ge 2\pi \}$

$x = 3(\cos(u))$
$y = 3(\sin(u))$

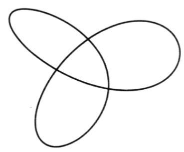
$\{ u \mid 0 \le u \ge 2\pi \}$

$x = 3((-3/2)\cos(2u)+\cos(u))$
$y = 3(\sin(u))$



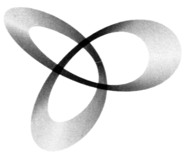
$\{ u \mid 0 \le u \ge 2\pi \}$

$x = 3((-3/2)\cos(2u)+\cos(u))$
$y = 3((3/2)\sin(2u)+\sin(u))$



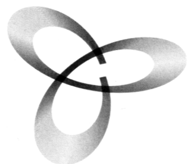
$\{ u \mid 0 \le u \ge 2\pi \}$

$x = 3((-3/2)\cos(2u)+\cos(u))$
$y = 3((3/2)\sin(2u)+\sin(u))$
$z = \sin(3u)$



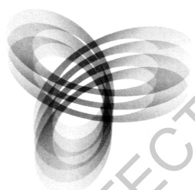
$\{ (u,v) \mid 0 \le u \ge 2\pi, 2\pi/3 \le v \ge \pi \}$

$x = v((-3/2)\cos(2u)+\cos(u))$
$y = v((3/2)\sin(2u)+\sin(u))$
$z = \sin(3u)$



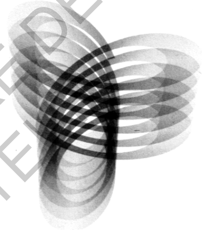
$\{ (u,v) \mid 0 \le u \ge 2\pi, 2\pi/3 \le v \ge \pi \}$

$x = v((-3/2)\cos(2u)+\cos(u))$
$y = v((3/2)\sin(2u)+\sin(u))$
$z = u/4$



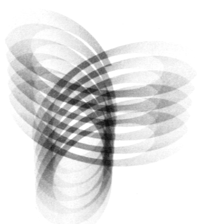
$\{ (u,v) \mid 0 \le u \ge 7\pi, 2\pi/3 \le v \ge \pi \}$

$x = v((-3/2)\cos(2u)+\cos(u))$
$y = v((3/2)\sin(2u)+\sin(u))$
$z = u/4$



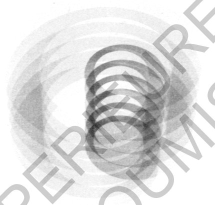
$\{ (u,v) \mid 0 \le u \ge 12\pi, 2\pi/3 \le v \ge \pi \}$

$x = v((-3/2)\cos(2u)+\cos(u))$
$y = v((3/2)\sin(2u)+\sin(u))$
$z = u/4$



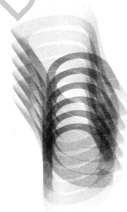
$\{ (u,v) \mid 0 \le u \ge 12\pi, 2\pi/3 \le v \ge \pi \}$

$x = v((-3/2)\cos(2u)+\cos(u))$
$y = v((3/2)\sin(2u)+\sin(u))$
$z = u/4$



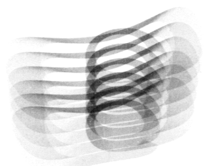
$\{ (u,v) \mid 0 \le u \ge 12\pi, 2\pi/3 \le v \ge \pi \}$

$x = v((-3/2)\cos(2u)+\cos(1-u))$
$y = v((3/2)\sin(2u)+\sin(1-u))$
$z = u/4$



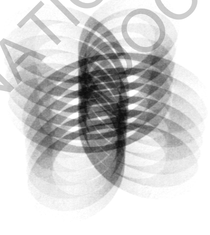
$\{ (u,v) \mid 0 \le u \ge 12\pi, 2\pi/3 \le v \ge \pi \}$

$x = v(\sin((-3/2)\cos(2u)+\cos(1-u)))$
$y = v((3/2)\sin(2u)+\sin(1-u))$
$z = u/4$



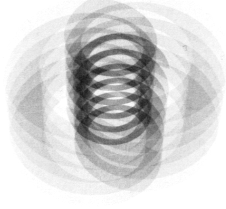
$\{ (u,v) \mid 0 \le u \ge 12\pi, 2\pi/3 \le v \ge \pi \}$

$x = v((-3/2)\cos(2u)+\cos(1-u))$
$y = v(\sin((3/2)\sin(2u)+\sin(1-u)))$
$z = u/4$



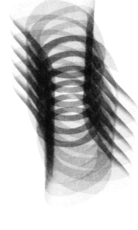
$\{ (u,v) \mid 0 \le u \ge 12\pi, 2\pi/3 \le v \ge \pi \}$

$x = v((-3/2)\cos(3u)+\cos(u))$
$y = v((3/2)\sin(3u)+\sin(u))$
$z = u/4$



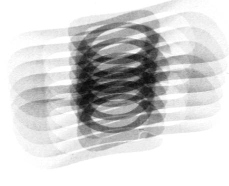
$\{ (u,v) \mid 0 \le u \ge 12\pi, 2\pi/3 \le v \ge \pi \}$

$x = v((-3/2)\cos(3u)+\cos(1-u))$
$y = v((3/2)\sin(3u)+\sin(1-u))$
$z = u/4$



$\{ (u,v) \mid 0 \le u \ge 12\pi, 2\pi/3 \le v \ge \pi \}$

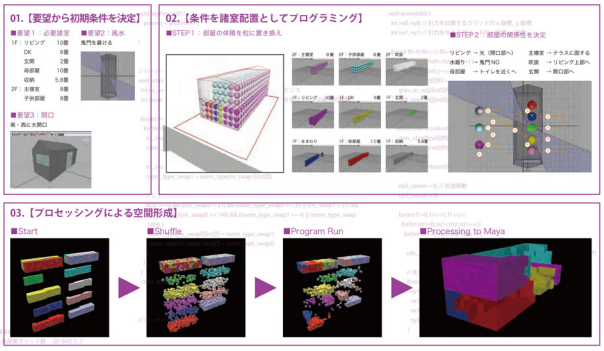
$x = v(\sin((-3/2)\cos(3u)+\cos(1-u)))$
$y = v((3/2)\sin(3u)+\sin(1-u))$
$z = u/4$



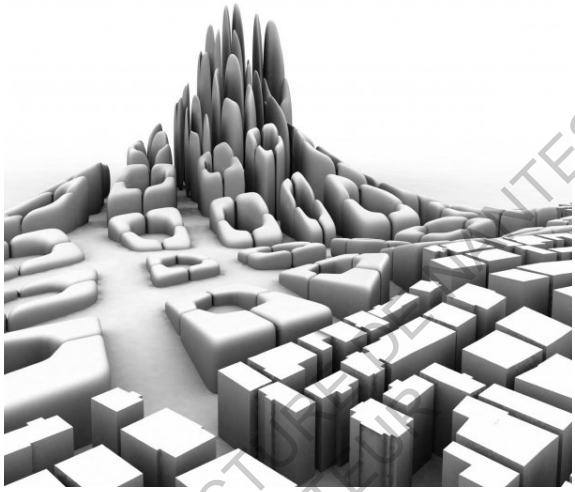
$\{ (u,v) \mid 0 \le u \ge 12\pi, 2\pi/3 \le v \ge \pi \}$

$x = v((-3/2)\cos(3u)+\cos(1-u))$
$y = v(\sin((3/2)\sin(3u)+\sin(1-u)))$
$z = u/4$

*Extrait du livre Morphing: A guide to mathematical transformations for architects and designers (Joseph Choma 2015)



1



2

2.9 Différentes échelles

Analyse d'exemples

A travers ces différents exemples, nous voulons montrer que la génération de modèles 3D à partir de règles et d'algorithmes se fait à toutes les échelles en intègrant plus ou moins de complexité et de paramètres.

La maison Est-ouest de Maeda Norisada (1)

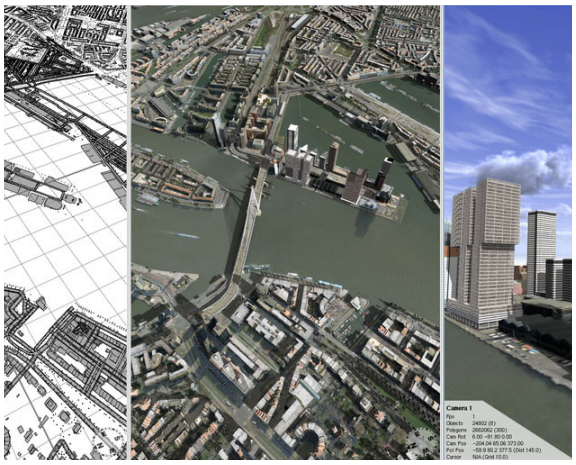
Ce projet de maison japonaise à la particularité d'être conçu de manière algorithmique. En effet l' architectes se sert de données liées à la parcelle et à ses environs, ainsi que des données de programme, comme le nombre de pièces, leur taille et orientation pour générer des volumes intérieurs et extérieurs. Cette démarche s'inscrit dans une démarche logique et systématique d'application de règles. L'établissement de ces règles permet de définir les relations souhaitées entre les différents espaces de la maison et nécessite aussi de l'architecte qu'il soit clair sur ses choix et priorités dans le travail qui vient en amont de la simulation.

Le projet d'Istanbul par Zaha Hadid (2)

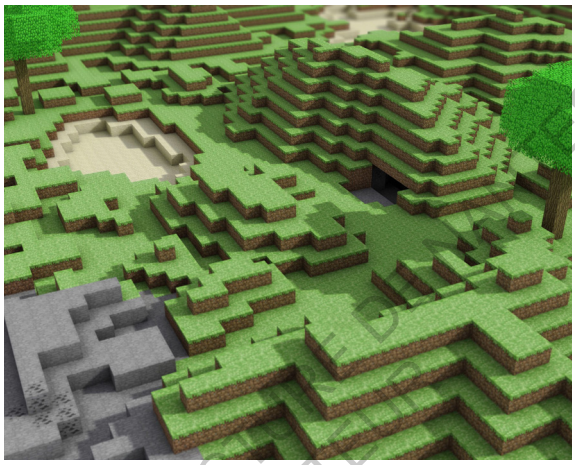
L'agence Zaha Hadid a travaillé sur la restructuration d'un quartier de la ville d'Istanbul. Cette phroposition revisite le principe de l'ensemble l'ilot urbain en venant arrondir la forme global, mais surout en développant des règles de relations entre les éléments. On remarque que plus les éléments sont hauts, plus le coeur d'ilot s'efface pour presque disparaître. Le travail part d'une grille urbaine paramétrée pour être dynamique, quand on modifie une partie du projet, une hauteur, une voie, les alentours s'actualisent pour répondre aux règles établies. L'existant est pris en compte, de manière avoir une progression douce entre l'ancien, et les parties en conception. Ce qui est important, notamment en urbanisme, c'est de comprendre comment une intervention à un endroit se répercute sur le réseau plus large d'infrastructures de la ville.

La ville virtuelle avec CityEngine (3)

CityEngine est un logiciel développé depuis 2008 qui transforme des informations cartographiques 2D en modélisation 3D. C'est un outil de visualisation et de conception destiné aux urbanistes, aux architectes mais aussi à l'industrie du film pour générer des villes imaginaires. Il a la particularité de fonctionner de manière paramétrique, ainsi il permet par exemple sur un ilot bâti de définir la taille, le nombre d'étage, l'épaisseur ou le type de bâtiments qui le compose. Par exemple on peux choisir s'il s'agit d'immeubles anciens, récent, de logements ou non, avec la possibilité de définir un rez-de-chaussé avec un commerce. Tout ces paramètre peuvent être actualisé en temps réel et le modèle s'adapte aux modifications. On le remarque assez vile, ce logiciel ne créer par vraiment une



3



4

ville, mais plutôt l'effet, l'impression d'une ville

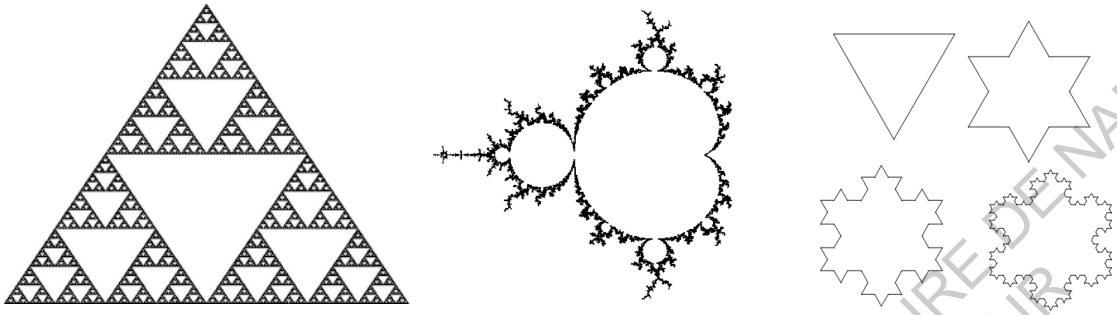
Le monde infini de Minecraft (4)

Les problématiques de génération de géométrie et d'espace ne sont pas seulement propres à la l'architecture. On peut faire un lien entre les jeux vidéos et l'architecture à deux niveau, le premier étant la modélisation en 3D d'un univers, qu'il soit urbain ou imaginaire, le second concerne les outils utilisés par le monde des jeux vidéos et les méthodes et logiciels employés. Ainsi il est peut être intéressant de faire un lien entre l'évolution des jeux, et une pratique de l'architecture qui se tourne de plus en plus au numérique, en matière de recherche notamment, sans forcément aboutir à une construction.

L'univers de Minecraft se développe autour d'un élément simple, le cube. Le jeu se caractérise par un monde ouvert composé de millions de cube formant des plaines, des montagnes, des îles et des galeries sous-terraines. Deux particularités sont à noter, la première étant que chaque cube peut être retirer de son emplacement d'origine et déplacer ailleurs, la seconde est qu'à mesure que le joueur se déplace, le jeu génère de nouveaux espaces. Cela suggère que le joueur peut marcher pendant des mois entiers dans une direction, sans atteindre de limite.

EXPÉRIMENTER

« Bien que toute invention technique puisse augmenter le champ de la liberté humaine, elle ne le fait que si les bénéficiaires humains sont libres de l'accepter, de la modifier ou de la rejeter, de l'utiliser quand et comme il convient à leurs intentions propres, en quantités conformes à leurs intensions » ***Lewis Mumford***



Le triangle de Sierpinski, l'ensemble de Mandelbrot, le flocon de Koch

3.1 Design et processus

L'inspiration de la nature

Tout d'abord le mot fractal renvoi à un type de géométrie auto-similaire, récurrente et homothétique.

On peut poser l'origine des fractales avec Apollonius, un disciple d'Archimède, il travail sur une forme géométrique qui s'appelle le baderne d'Apollonius, un triangle curviligne remplit de cercles de plus en plus petits. C'est trois siècles avant J-C qu'il pose la problématique des fractales sans que le mot n'existe encore.

Beaucoup plus tard vers 1500, Albrecht Dürer invente un objet fractal composé de d'une multitude de pentagones emboités les uns dans les autres. Cette figure amorce ce qui plus tard sera l'étude de la géométrie des flocons. Puis à la fin du XIXe siècle, Cantor exprime un concept de poussière, ou segment auquel on vient soustraire une portion de manière récursive.

En 1890 le mathématicien Peano travail sur une courbe qui se densifie au fur et à mesure de sa construction, produisant ce qui ressemble à un labyrinthe. Ce travail sera repris par von Koch lorsqu'il proposera une fonction connue comme le flocon de Koch, cette courbe récursive à la propriété d'avoir une surface finie, mais un périmètre infini. Puis en 1915 un autre mathématicien, Sierpinski expérimente différents algorithmes qui produisent deux figures emblématiques des fractales, le triangle, et l'éponge. Pour l'instant tout ces études sont faites à la main, comme nous l'avons vu aucune machine ne permet à cette époque de pouvoir calculer ce type de géométrie. Il faudra attendre les années 1970 pour que Benoît Mandelbrot s'intéresse aux fractales, il publiera plusieurs ouvrages sur le sujet.

Les travaux de Mandelbrot sur les fractales nous on permis de comprendre le processus de génération de géométrie complexe trouvé dans la nature, comme les montagnes, les nuages. Cela a été rendu possible par la mise en évidence d'algorithmes qui illustrent le processus itératif de ces formes.

On trouve dans la nature de beaux exemples de fractales, comme les arbres. Cependant il faut faire attention à ne pas confondre forme et formation, ou logique. On constate que l'arbre dans son ensemble est similaire à ses branches si l'on effectue un zoom. En revanche la forme n'est pas identique, c'est le processus de développement qui l'est.

« Mais non, la nature n'est pas analytique, lisse, dérivable, elle est fractale. Ces objets que Cantor, Peano, von Koch, Sierpinski, ont inventés – tirés de leur imagination, ou d'un libre jeu sur les définitions – en croyant s'émanciper de la nature, décrivent en fait mieux la nature que les fonctions analytiques des physiciens du XIXe siècle » **Mandelbrot**

3.2 Des thèmes récurrents

Cellular automata, fractals, branching

Plusieurs domaine d'étude sont récurrents dans les démarches algorithmiques, l'inspiration des concepteurs est souvent issue de l'observation et de l'analyse de phénomènes naturels. On peut citer comme exemples, les manières dont les plantes poussent, dont les cellules bougent, dont les flocons se forment, ou encore le phénomène d'évolution en génétique.

Nous allons tout d'abord voir dans l'expérimentation 1 comment programmer un système afin d'en ressortir différents modèles, générés suivant des règles précices.

Ensuite nous aborderons les cellular automata qui sont des systèmes dans lesquels des cellules possèdent différents états, comme allumées ou éteintes, qui évoluent suivant l'état des cellules adjacentes, ce thème sera illustré par l'expérimentation 2.

Puis l'expérimentations 3 traitera de ce que l'on appelle l'agent-based modeling, cela prendra la forme d'un système en recherche d'équilibre.

L'étape d'après servira à montrer comment la programmation permet de s'appuyer sur des bases de données pour une analyse ou une simulation par exemple, c'est ce qui sera traité dans l'expérimentatoin 4.

Dans l'expérimentation 5 nous verrons de la notion de branching, ou comment générer des formes de manière récursives, Ce sera aussi l'occasion d'adopter une posture critique sur cette démarche.

Pour finir, l'expérimentation 6 reposera sur un principe appellé DLA pour *diffusion limited aggregation.*

3.3 Expérimentation 1
Générer un modèle

Afin d'illustrer notre propos, nous présentons plusieurs travaux de programmation. L'objectif est de générer de la géométrie à partir de code et d'essayer de comprendre en quoi cela diffère d'une approche avec Revit, Rhino ou Autocad.

Le premier, effectué sous le logiciel Unity3D cherche à démontrer le degré de contrôle que l'on peut avoir pour réaliser une modélisation simple d'une «tour».

Nous avons commencé par l'élaboration d'un processus dans lequel un élément, ici un cube, se duplique en suivant une trajectoire à six possibilités, de manière aléatoire. Une fois le script lancé, à chaque étape un nouveau cube apparaît à une unité de distance du cube précédent (figure 1).

Dans le deuxième cas (figure 2) les même règles s'applique nt avec quelques changements, l'agent aura plus de chance de se dupliquer sur l'axe vertical que dans les autres sens, avec un plafond en hauteur, et des distances maximales sur les cotés, d'où l'effet visible s'agglomération en partie haute. Les figures 3 à 5 illustrent différentes déclinaisons de ce même principe, le code de la figure 5 étant reproduit en annexe.

Lors de l'élaboration d'un algorithme comme celui-ci, nous avons opté pour ce que l'on peut appeler un *système restrictif** c'est-à-dire que le développement du modèle dans différentes directions sont prédéterminés et intangibles, malgré le facteur aléatoire.

Dans le cas ou le modèle tiendrait compte de son environnement, on parlerait de *système permissif***, ce qui veux dire qu'à chaque étape, l'algorithme décide de développer le modèle 3D dans une direction, en vérifiant s'il n'y a aucun obstacle, comme un bâtiment mitoyen. Il en résulterait une génération de volume toujours sous un certain contrôle, mais en intégrant un autre paramètre à définir, soit la distance par rapport à un potentiel élément du système.

Le code de base de cet exemple est inspiré du script du « random walker » énoncé dans le livre « The nature of code » de D.Shiftman dans une version 2D pour Processing.

*, ** Cette distinction est notamment expliqué par Jacob Riiber dans sa thèse *Generative processes in architectural design* (2013) p 121, en citant un article de R.Glanville (2007)

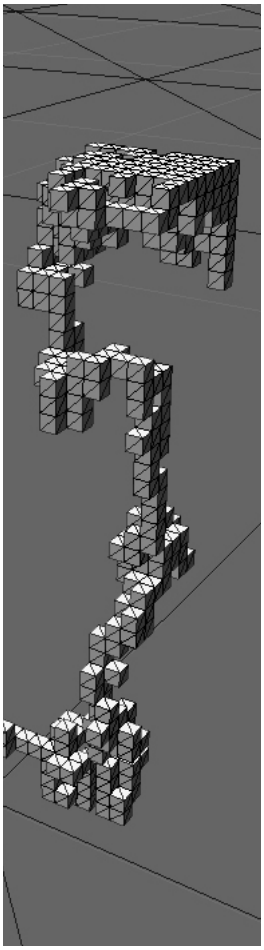
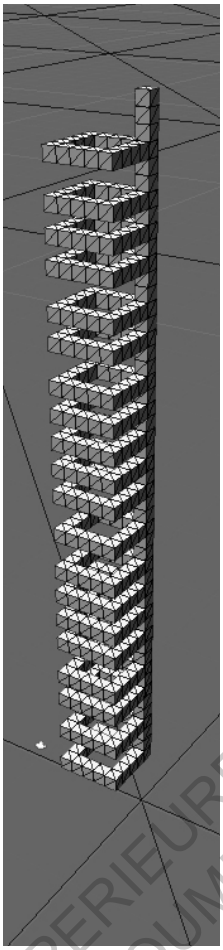
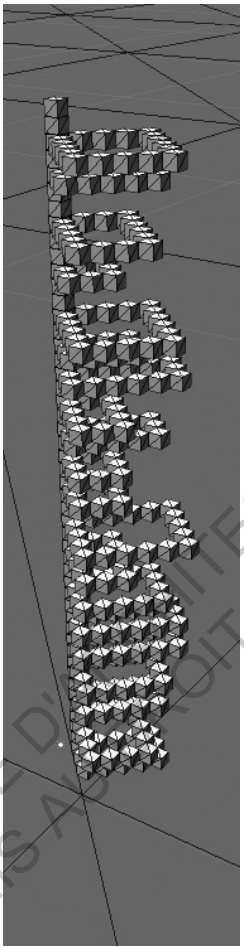
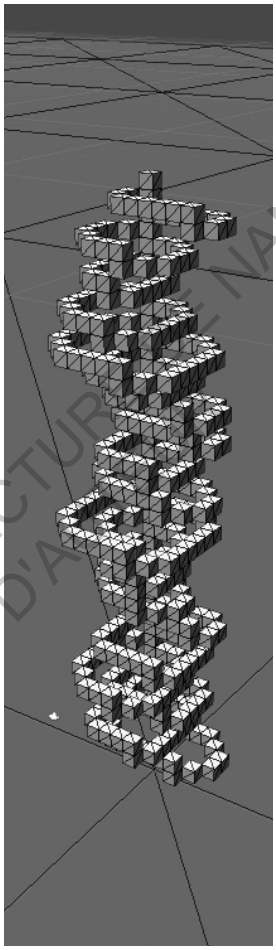


Figure 2







Figures 3, 4 et 5

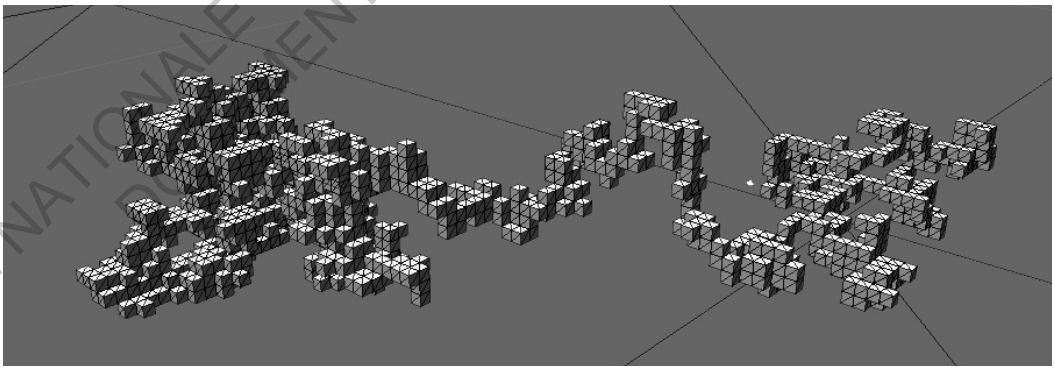


Figure 1

3.4 Processing
Plateforme d'expérimentation

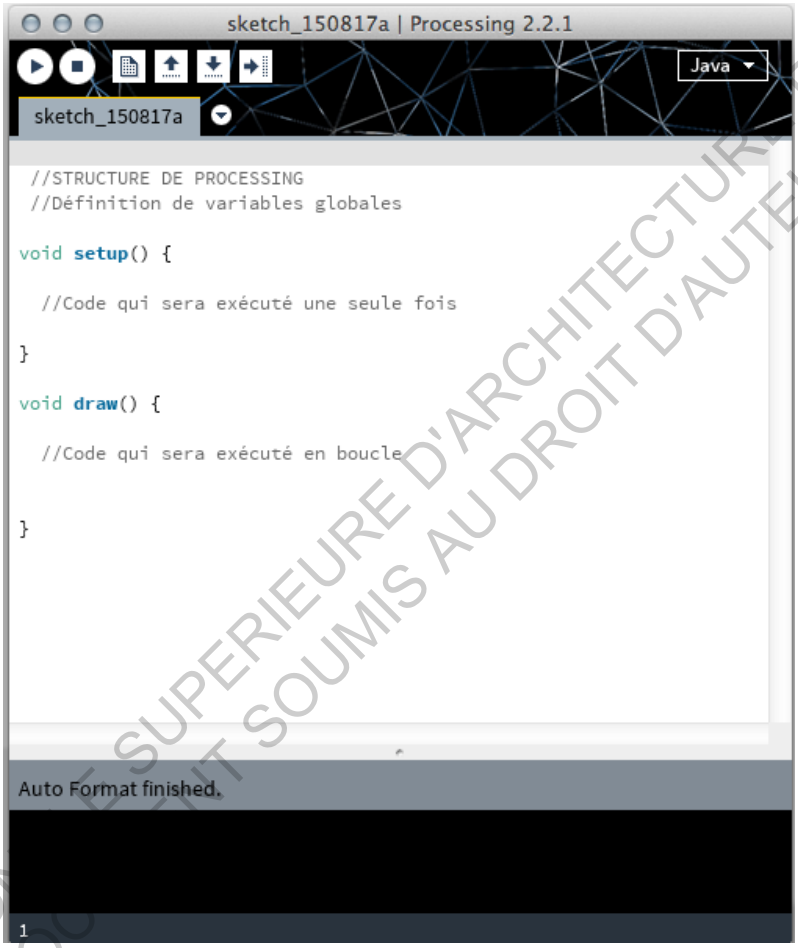
Pour le reste des expérimentaions nous avons choisi d'utiliser le logiciel Processing. C'est est un environnement de programmation qui utilise le langage Java. Il a été développé par Ben Fry et Casey Reas au sein du MIT. Comme beaucoup de logiciel il fonctione suivant la boucle « éditer, compiler, lancer ».Il est utilisé par une communauté de développeurs, d'artistes, de graphistes pour sa capacité à visualiser et traiter des données, générer des formes et produire un support d'analyse. Il est aussi pratiqué par des designers et des architectes qui cherchent à explorer la géométrie 2D ou 3D en ayant écrivant des fonctions, des procédures génératives et itératives, plutôt que de dessiner « à la main » avec des outils traditionnels ou informatiques.

Plus qu'une simple exploration, l'enjeu de cette démarche est de concevoir les outils nécessaires au projet, avant, ou en même temps que le projet lui-même, plutôt que d'adopter des outils existants. L'objectif est donc de d'abord concevoir la méthode et les outils de design, avant le design lui même. C'est-à-dire « designer le design ».

Cette approche non représentative pose la question suivante : comment peut-on engager un dialogue, une conversation, une collaboration avec la machine afin de concevoir un projet.

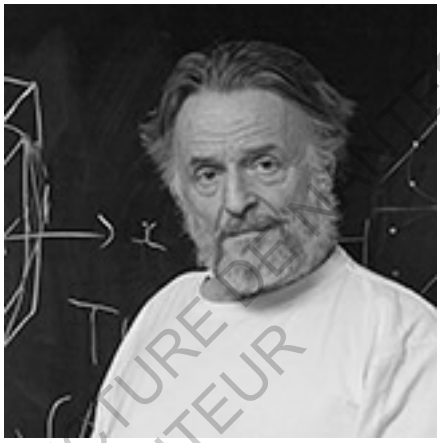
Les projets d'architecture sont de plus en plus complexes, en terme de fonctionnement ou de forme. De ce fait pour la conception ainsi que la représentation du projet il est indispensable de penser en trois dimensions, afin de comprendre l'imbrication des éléments, là ou une vision en plan ou en coupe ne permet pas d'appréhender la complexité du volume dans son ensemble.

Quel que soit le niveau de complexité en revanche, les éléments de départ, les règles sont toujours simple. Mais c'est l'itération de ces règles dans un système qui génère la complexité. Un simple changement dans les règles de départ a une influence qui se répercute dans tout le modèle ou la simulation.



Capture d'écran du logiciel

« CA simulate processes where local action generates global order, where global or centralized order « emerges » as a consequence of applying local or decentralized rules that in turn embody local processes » **Michael Batty**



John Conway inventeur en 1970 du *game of life*

3.5 Expérimentation 2
Cellular automata, Processing

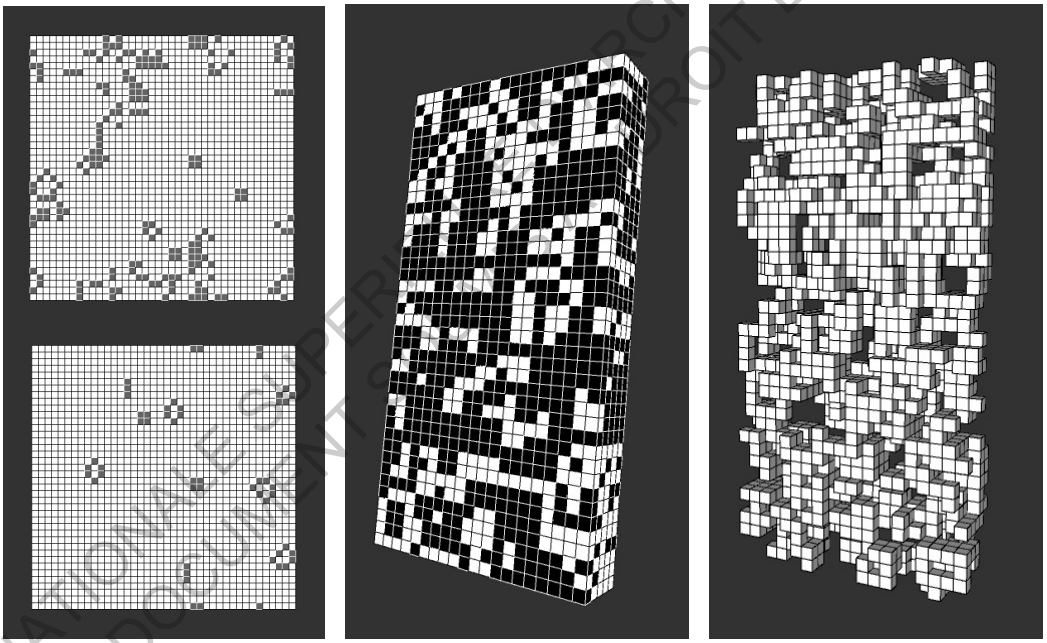
Les cellular automata, ou CA, sont un type de simulation qui s'appuie sur l'utilisation de cellules comme éléments de base. Inspiré par l'étude des organismes vivants au niveau cellulaire, les CA sont utilisés dans de nombreux domaines et c'est Von Neumann qui dans les années 1950 qui sera un pionier dans le domaine.

Dans ce deuxième exemple nous avons adapté le célèbre « game of life » sur le logiciel Processing, en 2D puis en 3D. Le processus est simple, une grille est dessinée, avec deux type de cases, noire ou blanche, « vivante » ou « morte ».

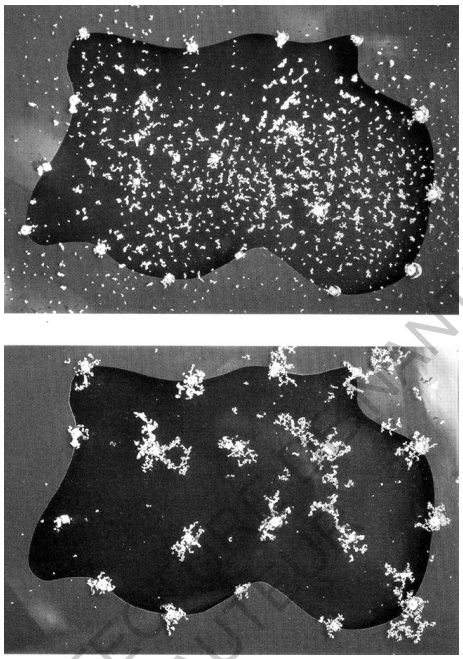
On démarre avec une distribution aléatoire des cases, dans une proportion à définir. Chaque case noire considère ses quatre voisins, si elle en a suffisament elle reste vivante, sinon elle meure et devient blanche. A l'inverse une case blanche qui a suffisament de voisins peut devenir noire, et ainsi de suite. La figure 7 montre deux états différents de la simulation, qui devient stable lorsqu'aucune case ne remplit plus les conditions évoquées.

Les figures 8 et 9 reprend le principe de la simulation précédente mais en 3D, la figure 9 ne montre que les « cases » blanches en faisant émerger une géométrie. L'objectif de cet exemple est de montrer comment des choses complexes peuvent émerger à partir d'éléments simple comme une grille. Si l'on compare cette simulation avec la précédente, on obtient cette fois des éléments ou «cellules» qui interagissent avec leur environnement proche. Tout le processus qui se met en place repose sur le fait que les cellules se transforment suivant l'état de leur plus proche voisine, ce qui génère une dynamique collective.

Le code de base du second exemple est inspiré du script du « game of life » présenté par J.Sanchez sur le site plethora-project.com dans une version 2D pour Processing.



Figures 7, 8 et 9



Distanciations et attractions simultanées, Frei Otto 1992

3.6 Expérimentation 3
Agent-based modeling

Ce que l'on appelle *agent-based modeling* correspond à un type de simulation informatique qui repose sur l'étude du comportement d'agents, au travers d'un algorithme donné, dans un système. Ce genre de simulation est utilisé notamment en biologie et dans les sciences sociales, par exemple dans l'étude du comportement des foules.

L'étude de différent phénomènes trouvés dans la nature a eu plusieurs répercusions, la découverte de structures, de motifs mais aussi de comportements. Des recherches sur les systèmes organisés de manière autonomes ou *self-organized systems* ont permis de développer une nouvelle approche de la modélisation 3D.

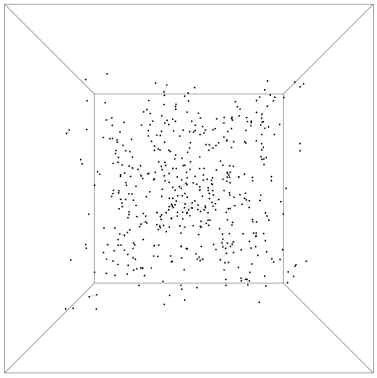
L'idée n'est plus de manipuler des solides réguliers pour créer une forme souhaitée, mais de définir un système d'agents, capables de recevoir des informations de leurs environnement ainsi que des autres agents. Cette méthode s'inspire largement du domaine de la cybernétique qui étudie les systèmes, vivant ou non vivant, qui intègrent un but.

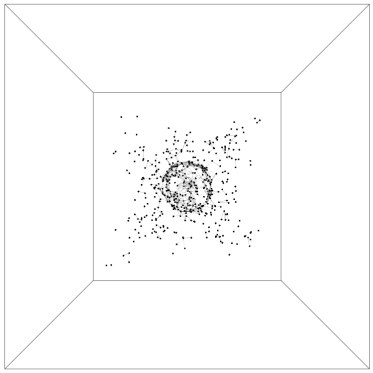
Afin de mieux appréhender la méthode, nous proposons l'étude illustré ci-contre. Un certain nombre d'agents , ici 500, sont générés dans un espace en trois dimensions. Un point dans l'espace, au centre, agit comme un aimant, qui attire les agents. Chaque agent calcule d'une part sa distance par rapport au centre, et d'autre par sa distance par rapport à tous les autres points. Si deux agents sont suffisament près l'un de l'autre, ils se connectent par une ligne. Enfin si un agent s'approche trop du centre, il est repoussé.

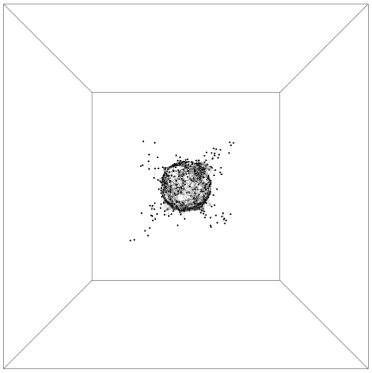
Le résultat est assez simple, d'un état de désordre, le système s'organise par le biais de différentes règles, et de ces règles émerge une organisation. Il faut noter qu'ici, à un moment de la simulation, tous les agents ont atteint leur limite de distance par rapport au centre, ils sont alors constament attiré et repoussé. Le système est donc organisé mais non stable, il s'adapte constament.

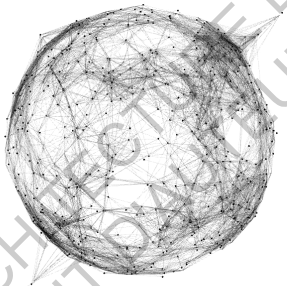
Une méthode similaire a été employée par Frei Otto en 1992, sans l'usage d'un ordinateur. Il utilisait des billes de polystyrène et des aimants plongés dans un bac d'eau pour réaliser son expérimentation de système en recherche d'équilibre. Grâce à ce travail il put définir la notion de surface minimale entre deux géométries.

L'idée d'une architecture comme un système qui se réorganise n'est pas nouvelle, dans les années 1960 c'est le groupe Archigram qui développera ces idées au travers de scénarios et d'utopie comme Walking city de Ron Herron, ou encore Plug-in city, de Peter Cook.









Evolution d'un système d'agents vers un etat d'organisation

Il est tentant d'évoquer le concept d'émergence, définit comme le produit de la rencontre d'une multitude d'idées, en regardant cette simulation. Cependant il faut être vigilant, l'ordinateur n'invente rien, il applique explicitement les différentes commandes qu'on lui donne. Par conséquent ça n'est pas en produisant des simulations complexes, qui seraient impossible à réaliser à la main, que l'émergence est possible. En revanche c'est bien en faisant intervenir un groupe d'individus au sein d'un système, comme le Fun palace évoqué précédement, qu'il est possible de faire émerger de nouvelles façons d'utiliser et de concevoir l'espace.

	A	B	C	D
1	NAME	LATITUDE	LONGITUDE	POP
2	Shanghai	31.22222	121.45806	14608512
3	Buenos Aires	-34.61315	-58.37723	13076300
4	Mumbai	19.07283	72.88261	12691836
5	Mexico City	19.42847	-99.12766	12294193
6	Karachi	24.9056	67.0822	11624219
7	?stanbul	41.01384	28.94966	11174257
8	Delhi	28.65381	77.22897	10927986
9	Manila	14.6042	120.9822	10444527
10	Moscow	55.75222	37.61556	10381222
11	Dhaka	23.7104	90.40744	10356500
12	Seoul	37.566	126.9784	10349312
13	São Paolo	-23.5475	-46.63611	10021295
14	Lagos	6.45306	3.39583	9000000
15	Jakarta	-6.21462	106.84513	8540121
16	Tokyo	35.6895	139.69171	8336599
17	Zhumadian	32.97944	114.02944	8263100
18	New York City	40.71427	-74.00597	8175133
19	Taipei	25.04776	121.53185	7871900
20	Kinshasa	-4.32758	15.31357	7785965

source: www.geodatasource.com/world-cities-database

3.7 Expérimentation 4
Travailler avec des données

Des logiciels tels que Processing sont performants quand il s'agit de travailler avec de larges bases de données, ou des données en temps réel. Pour le cas présent nous avons travaillé avec une base de données disponible en ligne qui fournit quatre indications sur les villes de plus de 5000 habitants dans le monde. Nous avons accès au nom, aux coordonées longitude et latitude de la ville, ainsi que le nombre d'habitants (hors agglomération).

Avec Processing nous pouvons importer ses données pour les visualiser de manière brut, comme le montre la première image, mais pas seulement. Chaque point représente une ville, mais est aussi considéré comme un agent, c'est à dire un élément faisant partie du système, capable d'interagir avec l'environnement développé. La seconde image nous propose une visualisation des mêmes villes, en intégrant la notion de taille, le diamètre des cercle dépend donc du nombre d'habitant répertorié dans la base. Seules, les données d'une ville ont peu d'importance pour avoir une vision globale, mais ensemble c'est un outil d'analyse puissant qui sort de l'échelle urbaine pour proposer une vision mondiale.

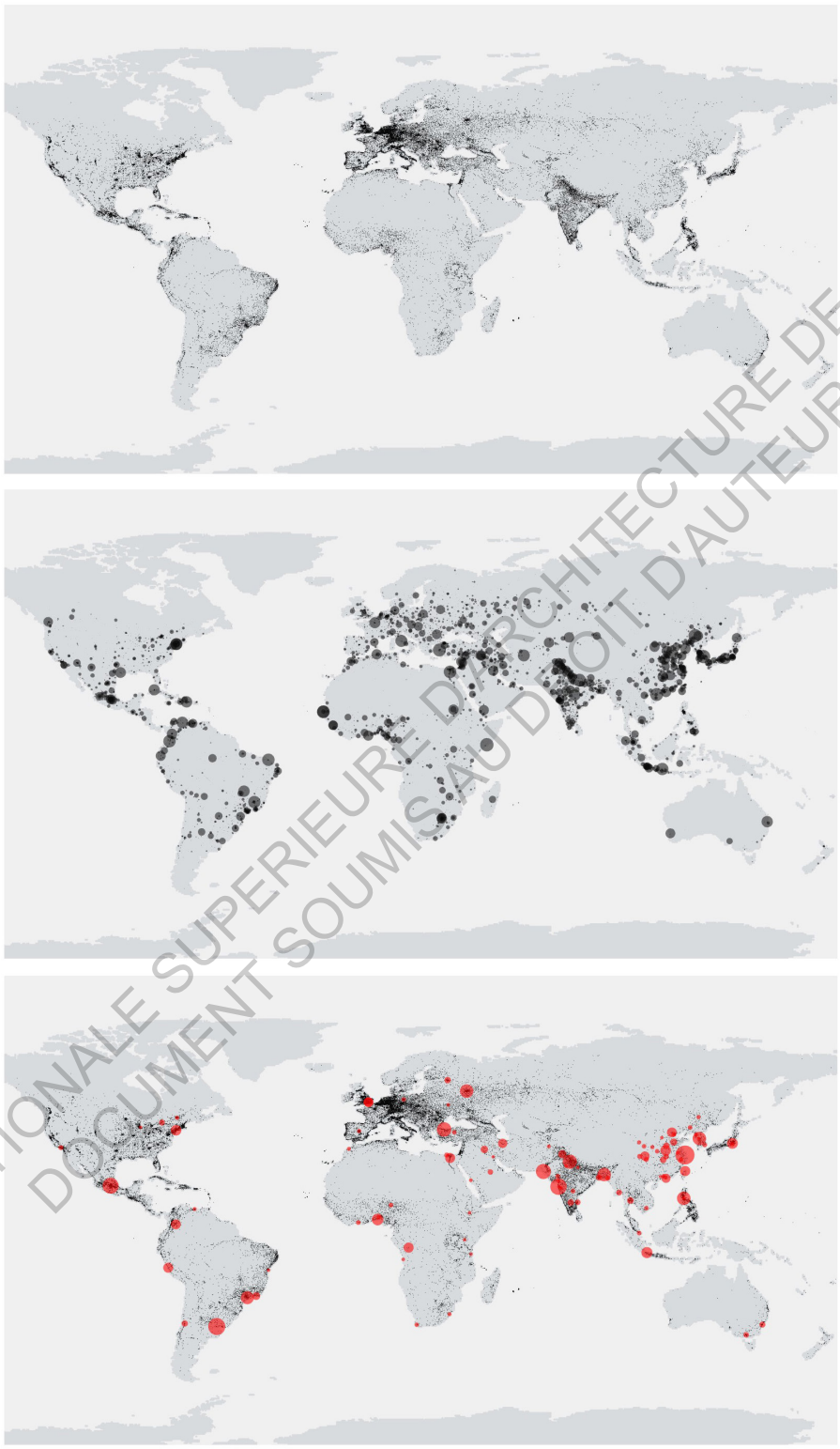
La troisième figure correspond à la première, avec l'ajout des 100 villes les plus peuplées, en rouge. L'intérêt de cette méthode n'est pas de s'arrêter à la visualisation, il faut voir cette carte comme uu outil de simulation. Que se passerait-il si les villes de taille moyenne commençait à migrer vers le point rouge le plus proche, comme attirés par un aimant ?

Cette carte invite à se poser des questions sur le long terme, comment l'urbanisation et la forte croissance démographique de pays comme l'Inde ou la Chine va-t-elle bouleverser notre manière d'appréhender l'espace habitable sur la planète ?

Le problème majeur en travaillant avec des données est de vérifier la pertinence des informations que l'on a collecté. De plus nous n'intégrons ici que quelques facteurs, en laissant de côté des notions comme la densité des villes ou le climat.

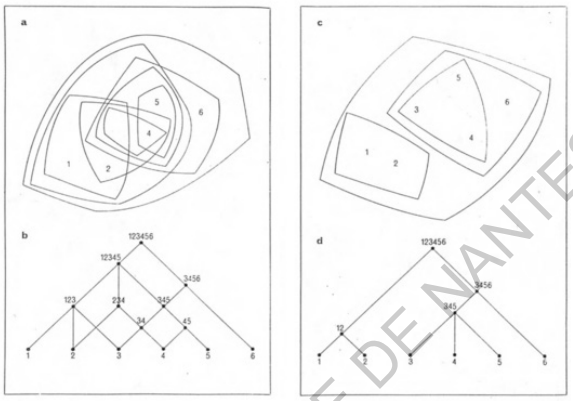
Le travail de l'architecte n'a jamais été de se focaliser sur une parcelle, mais de comprendre le rôle de ce terrain à une échelle plus large. Aujourd'hui les villes du monde entier se posent la question de l'extension urbain et de la densité. Aussi nous devons engager une réflexion sur les mouvements potentiels de population faisant face à des catastrophes climatique, et ce type d'outil peut aider, dans certaines limites, à l'appréhender.

Echelle mondiale



Haut: *Emplacements des villes de plus de 5000 habitants* ***Milieu:*** *Les mêmes villes avec le facteur de population* **Bas:** *En rouge, les 100 plus grandes villes du monde*

« When we think in terms of trees we are trading the humanity and richness of the living city for a conceptual simplicity which benefits only designers, planners, administrators and developers. Every time a piece of a city is torn out, and a tree made to replace the semilattice that was there before, the city takes a further step toward dissociation.» ***C. Alexander***



Branching: simulation de croissance

3.8 Expérimentation 5

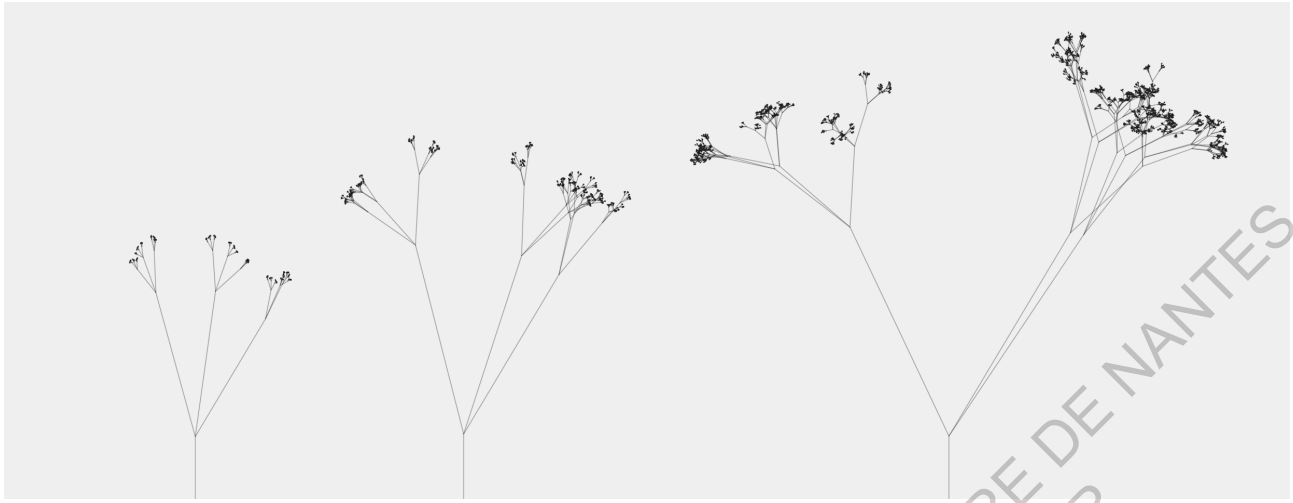
Fractal et branching, récursivité

Cette exemple montre comment par un processus récursif, il est possible de simuler le principe de croissance d'un arbre. Les règles sont les suivantes, à partir d'une branche principale trois nouvelles branches se développent, et à leur tour chacune va se diviser en trois et croitre de son côté.

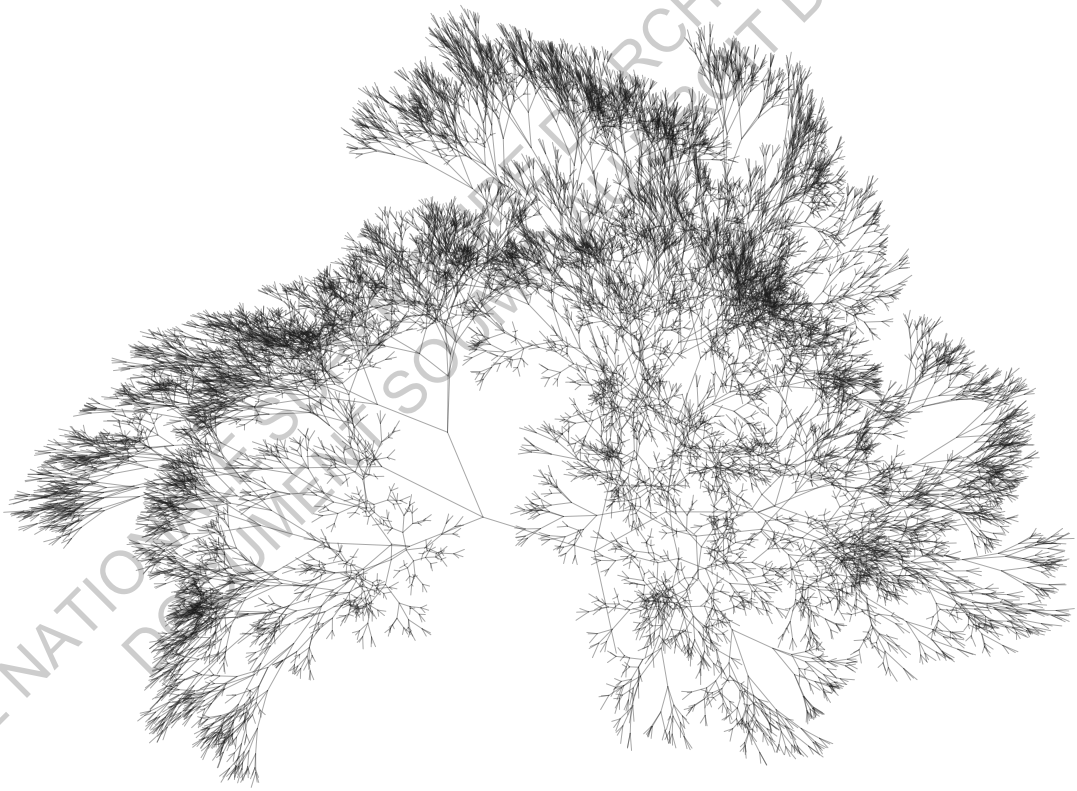
Si nous examinons les règles, on se rend compte qu'à aucun moment les trois branches de départ ne pourront se rejoindre au cours du développement de la structure. C'est ce phénomène de dissociation, propre à ce type de structure, dont parle Christopher Alexander dans son article « A city is not a tree ».

Il explique que cette solution de structure, favorisée par de nombreux architectes et urbanistes dans le dessin de nouvelles villes, ou dans des projets d'extension de zones urbaines existantes, ne produit en aucun cas un environnement favorable aux interactions dans la ville, et qu'au contraire ils sont appauvris.

Il est tentant de vouloir appliquer à une démarche architecturale des principes examiné dans la nature pour les programmer dans un espace virtuel, mais il faut être vigilant à ne pas se laisser entrainer par l'outil dans une direction avant de savoir ou non si cela a vraiment un sens.



Branching: simulation de croissance



Vue du modèle développé à un instant t
fragment d'animation.

DLA simulation de croissance

3.9 Expérimentation 6

Diffusion limited agregation (DLA)

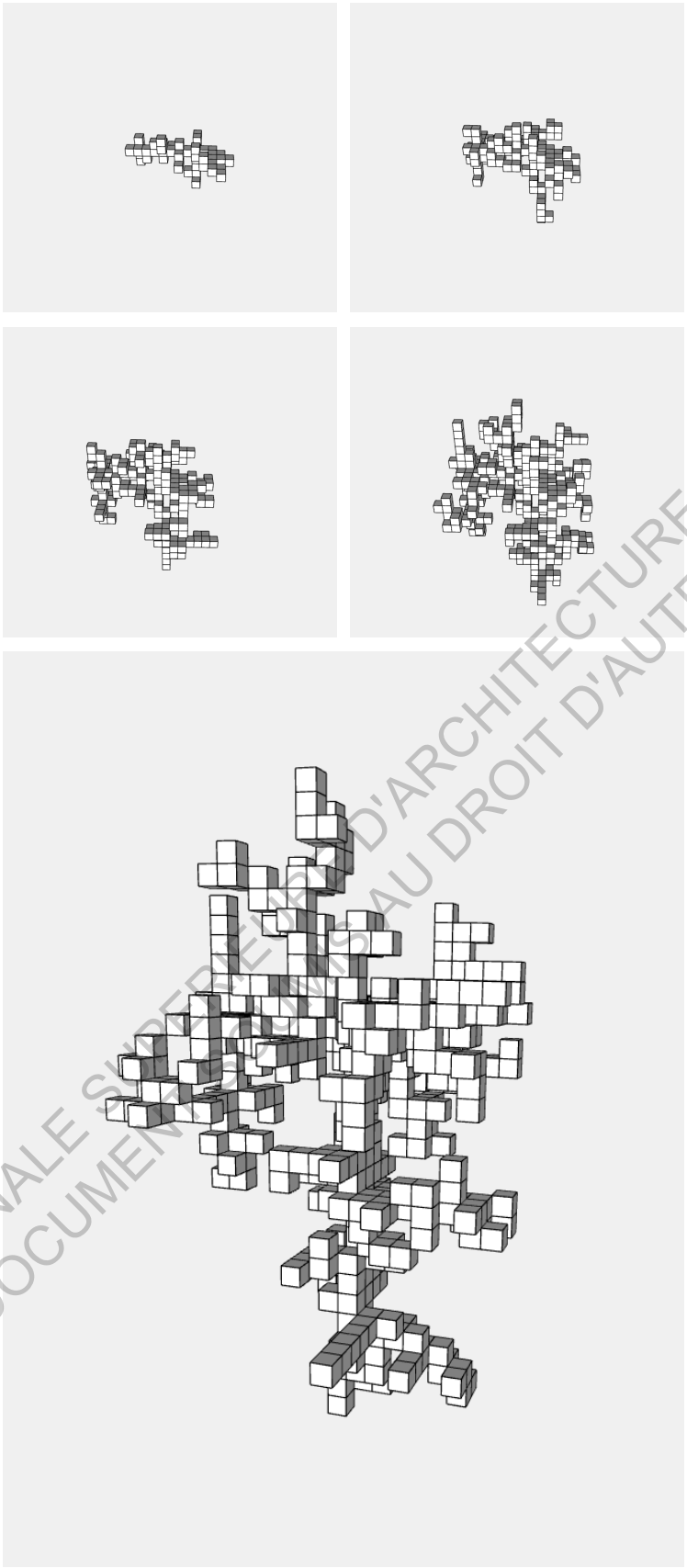
Le principe de diffusion de la DLA ressemble au déplacement aléatoire présenté dans l'exemple 1. Cependant au lieu de prendre comme point de référence l'élément généré précédement pour se développer dans une direction à partir de ce point, la diffusion ici peut se faire dans n'importe quelle direction, à partir du moment ou elle s'accroche à un élément existant.

Ce changement de règle produit un comportement et une géométrie complètement différente car les éléments sont alors obligé de se regrouper et de former des grappes, ou « cluster ».

Plus la croissance augmente, plus les chances que le développement soit en périphérie est élévé, c'est ce qui créer ce motif particulier.

Apprendre comment certaines règlent provoque une organisation spatiale spécifique, et essayer d'en tirer des principes, bon ou mauvais.

Ce type de simulation a été largement utilisé par des chercheurs en urbanisme, afin d'analyser différents modèles de croissance d'un sysème comme une ville ou un groupement de villes dans leur dynamique d'expansion urbaine.



Vue du modèle développé à un instant t
fragment d'animation.

«Since, as designers, we are concerned with the physical living city and its physical backbone, we must naturally restrict ourselves to considering sets which are collections of material elements such as people, blades of grass, cars, molecules, houses, gardens, water pipes, the water molecules in them etc. ***When the elements of a set belong together because they co-operate or work together somehow, we call the set of elements a system.****»*

Christopher Alexander 1965

3.10 Approche systémique

Synthèse

Tous les exemples dans cette partie reflètent un principe, c'est la conception de géométrie à partie de règles de base simples qui engendre un système complexe au travers du principe d'itération propre aux algorithmes.

Le technologie nous donne des moyens, des outils qui nous permettent d'étendre nos capacités à concevoir et traiter des problèmes complexes. En effet cette approche permet de comprendre plusieurs choses, d'abord que rien n'existe en soi, mais fait toujours parti d'un système plus large d'éléments connectés et dépendants, c'est la pensée systémique. Et ensuite qu'une simple variation dans l'énoncé de départ engendre des changements important dans le développement de ce système, c'est la théorie du chaos.

Les différentes expérimentations présentées ne servent en aucun cas d'exemple formel. C'est-à-dire que l'image ou l'apparence n'est en rien la finalité. Ce qui importe en revanche c'est de comprendre la manière dont les choses se forment, leur processus de génération. Car le coeur de ces simulations n'est pas la forme, on peut facilement remplacer l'élément « cube » utilisé à plusieurs reprises et décider qu'il soit en fait une sphère ou une banane. Mais l'élément majeur, le processus, lui ne dépend pas de la visualisation, la logique qui structure les simulations en revanche reste la même. Les architectes ne sont pas simplement des concepteurs d'espaces, architecturaux ou urbains, mais des concepteurs de systèmes, c'est pourquoi ils attachent tant d'importance au « contexte » élément majeur d'analyse dans la démarche de conception, à la fois local et global qui nourri le projet.

Les outils de programmation, textuels comme Processing ou visuel comme Grasshopper sont indispensables pour comprendre et pratiquer la complexité, cependant il faut rester vigilant et trouver des applications architecturales, urbaines qui d'une part s'émancipent de la forme pour se concentrer sur l'espace, la matière et les interactions humaines, sans quoi on reste bloqué dans un univers virtuel qui s'apparente à du jeu video ou du cinéma.

UN MONDE PROGRAMMÉ

«Technology, which has enabled architecture to expand its design boundaries, is the bridge between striking new conceptual possibilities on one hand, and the realities that govern construction and maintenance on the other» ***R.Nandha***

Projet MX3D, construction d'une passerele imprimé en acier à Amterdam en utilisant deux bras robotiques.

4.1 Fabrication programmée

Nouveau paradigme

Si les outils de conception ont évolué, c'est le cas aussi pour les procédés de fabrication. Pendant longtemps l'industrie se basait sur un principe de standardisation, on fabriquait des pièces avec des dimensions et des caractéristiques standard pour des raisons d'une part économiques et d'autre part pour un souci de compatibilité des éléments entre eux.

Aujourd'hui fabriquer 10 000 pièces identiques ou toutes différentes ne revient pas plus cher grâce aux machines de découpe laser et de prototypage 3D dont on dispose et qui se sont largement démocratisées. Cela signifie que les géométries de plus en plus complexes que nous dessinons dans le monde virtuel, peuvent naître dans la réalité. En effet, une modélisation complexe sera exporté en éléments simples dans un fichier destiné à la découpe pour ensuite être assemblés.

L'architecture fait partie d'un système global ou conception et construction sont lies par l'architecte, qui coordonne le projet et l'accompagne du début à la fin. De ce fait un changement dans cette "chaine de production" comme un nouvel outil pour l'architecte, de nouvelles façons de construire, ou une nouvelle règlementation résonne à tous les niveaux.

Le design paramétrique est donc légitimement un bouleversement puisque c'est une méthode de conception qui demande aux autres acteurs de l'architecture de s'adapter. L'adaptation est une forme de test pour savoir si un élément va durer dans le temps, ou s'il est voué à rester une tentative. On peut donc parler de l'outil comme faisant partie d'un système, et comme système à part entière.

Aujourd'hui l'industrie a quitté la période de la standardisation en développant des moyens de fabrication capables de faire des pièces identiques ou toutes différentes pour le même coût. D'un point de vue du consommateur, la beaucoup de produits sont maintenant personnalisables ou interchangeable en partie. C'est devenu une demande de pouvoir choisir la forme, la couleur, la matière d'un élément d'un produit, pour une fonctionnalité égale.

4.2 Matière programmée

Une évolution en marche

Depuis quelques années, grâce à l'évolution de techniques existantes comme les bras robots, l'impression 3D et l'apparition du hardware open-source comme Arduino ou Raspberry Pi on voit apparaître des recherches sur ce que l'on peut appeler la matière programmée. Ces recherches s'appuient sur différents thèmes que nous avons

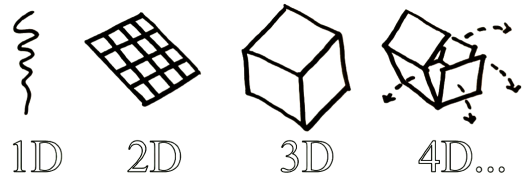


Schéma de présentation, *Self-Assembly Lab*

Octahèdre « programmé », *Self-Assembly Lab*

Les modules automatisés « Hypercell », *AADRL*

évoqué dans ce mémoire, comme les systèmes d'agents ou les cellular automata.

Tout d'abord à Londres au Design Research Lab de l'Architectural Association (AADRL), des étudiants ont mis au point de petites machines nommées « Hypercell » qui fonctionnent sur un principe inspiré du Game of life. Ils cherchent à reproduire mécaniquement des comportements d'agglomération/dissociation. L'objectif est de créer des structures urbaines dynamiques et adaptatives sur une base de polyèdres auto-structurés

Le chercheur Skylar Tibbits au sein d'un laboratoire du MIT explore depuis plusieurs années la notion d'auto-assemblage ou self-assembly, sous le nom « 4D Printing ». Dans une vidéo publiée en avril 2013 sur TedTalks il présente différents prototypes de géométrie, d'une à quatre dimensions, capables de passer d'un état désorganisé à un état organisé par interaction avec son environnement en utilisant une énergie passive, tel que la gravité, la chaleur ou l'énergie cinétique, ainsi ils revisitent certains principes déjà connus de matériaux à mémoire de forme en utilisant la technologie de l'impression 3D.

Enfin à Amsterdam la start-up MX3D a démarré sont projet de construire une passerelle en l'imprimant en 3D. Alors que la plupart des imprimantes fonctionnent à partir de matière plastique, leurs deux robots Kuka, bras articulés notamment utilisés dans les chaines de production automobile, sont capables d'imprimer de l'acier par dépôt de matière en fusion, ou FDM. Ce type de technologie qui permet de construire sur place et de manière automatisée est un véritable bouleversement qui a pour effet de repenser la relation entre concepteurs et ouvrage en s'affranchissant de la main d'œuvre traditionnelle.

CONCLUSION

« Désormais, le créateur peut être son propre outil. En effet, tout programme génératif est aussi simultanément un outil logiciel personnalisé, grâce auquel le concepteur peut ouvrivr de nouvelles voies » ***Design génératif***

« A designer might say I want to move and twist this wall, but you did not forsee that move and there is no parameter to accomodate the change. It then unravels your (parametric model). Many times you will have to start all over again » **Rick Smith**

4.3 Conclusion

Une pratique centrée sur l'outil

Nous avons chercher à comprendre ce que pouvait nous apporter la programmation dans une démarche architecturale. Nous avons évoqué notamment la problématique du temps ou comment les bons outils sont capables de nous aider à produire un travail qu'il serait impensable de faire à la main.

Daniel Davis dans sa thèse* montre que les modèles paramétriques manquent cependant souvent de flexibilité, et qu'au lieu de pouvoir changer un simple paramètre afin de mettre à jour le modèle, il faut en fait recommencer à zéro.

Un autre problème récurrent est mis en évidence de manière provoquante par l'architecte Mark Gage (voir citation ci-contre), il déplore que l'outil soit au centre du discours des concepteurs, des étudiants, au détriment du projet lui-même et des questions auquelles il est censé répondre. Ainsi quand bien même on peut être fasciné par les possibilités offerte par un outil informatique, il est nécessaire d'éviter de se focaliser sur le *comment*, pour se recentrer sur le *pourquoi*. L'outil n'est en effet jamais une fin en soit et la vague de popularisation des imprimantes 3D ces dernières années en est aussi un bon exemple, la valeur véhiculée est trop souvent du fait qu'un objet soit imprimé sans vraiment se soucier du pourquoi de l'objet et sa véritable fonction.

Les outils numériques ont trop souvent incité les architectes à produire de la forme pour la forme. La complexité formelle étant de plus en plus facile à produire, nous avons assisté depuis les années 2000 à une production architecturale écléctique centrée sur la notion d'icone où chaque bâtiment est une déclinaison formelle de plus, plus sculpture qu'architecture, plus façade qu'espace. En effet quand l'outil n'est pas mis en valeur, c'est la forme qui est le sujet d'intérêt, ce qui entraine une confusion générale dans l'esprit des gens qui ne sont pas du domaine architectural.

Le point positif de cet enthousiasme vers l'outil est qu'il emporte avec lui une résurgence des thèmes que nous avons abordé, fractales, cellular automata et avec eux une pensée héritée du XXe siècle que l'on redécouvre ainsi qu'une approche systémique qui nous le pensons peut jouer un rôle majeur dans notre capacité à gérer les évolutions de notre société et la croissante complexité de nos espaces de vie.

*Modelled on Software Engineering : Flexible Parametric Models in the Practice of Architecture (2013)

"… use computation, but stop fucking talking about it. Your project isn't any better because you told me it was scripted from the secret code found in the lost book of the Bible handed to you by your Merovingian great grandmother. Nor because you spent a semester producing the most intricate parametric network ever seen by man, & still ended up with three crumpled potatoes in glossy grey." **Mark Gage**

4.4 Quel avenir ?

Notre époque est communément appellée révolution technologique ou révolution numérique qui se situe sur une courbe exponentielle au travers «d'avancées» comme la miniaturisation de l'informatique. Si il est vrai chaque jour apporte son lot de nouveautés, il est indispensable de prendre du recul vis-à-vis de ce que l'on peut appeller «Neomania».

Nous sommes entrée dans une ère que l'on peut nommer «hyperhistory» (L.Floridi) ce qui signifie que nous sommes maintenant dépendants des technologies d'information et de communication, ou ICTs. Il nous est désormais impossible de penser en dehors du système que nous avons bâti.

ANNEXE

Code source des différentes expérimentations

Expérimentation 1

Différentes générations de géométrie

```
var emiter : Transform;
var bloc : GameObject;

function Start () {
for( var a:int=0; a<20;a++){
tower1();
}
}
function tower1 () {
var stepX:int = Random.Range(1,6);
var stepZ:int = Random.Range(1,6);
var stepD:int = Random.Range(1,3);
var stepY:int = Random.Range(1,2);

for( var i:int =0 ; i<stepZ; i++){
emiter.transform.position.z += 1;
Instantiate(bloc,emiter.transform.position,
emiter.transform.rotation);
}

for( var j:int =0 ; j<stepD; j++){
emiter.transform.position.x += 1;
emiter.transform.position.z += 1;
Instantiate(bloc,emiter.transform.position,
emiter.transform.rotation);
}

for( var l:int =0 ; l<stepX; l++){
emiter.transform.position.x += 1;
Instantiate(bloc,emiter.transform.position,
emiter.transform.rotation);
}

for( var a:int =0 ; a<stepD; a++){
emiter.transform.position.z -= 1;
emiter.transform.position.x += 1;
Instantiate(bloc,emiter.transform.position,
emiter.transform.rotation);
}

for( var n:int =0 ; n<stepY; n++){
emiter.transform.position.y += 1;
Instantiate(bloc,emiter.transform.position,
emiter.transform.rotation);
}

for( var k:int =0 ; k<stepZ; k++){
emiter.transform.position.z -= 1;
Instantiate(bloc,emiter.transform.position,
emiter.transform.rotation);
}

for( var b:int =0 ; b<stepD; b++){
emiter.transform.position.z -= 1;
emiter.transform.position.x -= 1;
Instantiate(bloc,emiter.transform.position,
emiter.transform.rotation);
}

for( var c:int =0 ; c<stepX; c++){
emiter.transform.position.x -= 1;
Instantiate(bloc,emiter.transform.position,
emiter.transform.rotation);
}

for( var d:int =0 ; d<stepD; d++){
emiter.transform.position.z += 1;
emiter.transform.position.x -= 1;
Instantiate(bloc,emiter.transform.position,
emiter.transform.rotation);
}

for( var m:int =0 ; m<stepY; m++){
emiter.transform.position.y += 1;
Instantiate(bloc,     emiter.transform.position,
emiter.transform.rotation);
 }
}
```

Expérimentation 2

Game of life 3D

```
import peasy.*;
import toxi.geom.*;
PeasyCam cam;

int size = 20;
int step = 5;

int cols = size;
int rows = int(random(80, 100));
int depth = 5;

CA grid [][][] = new CA[cols][rows][depth];

void setup() {
  size(900, 900, P3D);
  frameRate(50);
  cam = new PeasyCam(this, 500);

  for (int i=0; i<cols; i++) {
    for (int j=0; j<rows; j++) {
      for (int k=0; k<depth; k++) {
         Vec3D place = new Vec3D(i*step, j*step,
k*step);
        grid[i][j][k] = new CA(place, i, j, k);
      }
    }
  }
}

void draw() {
  background(50);
  stroke(255);
  fill(255, 20);

  for (int i=0; i<cols; i++) {
    for (int j=0; j<rows; j++) {
      for (int k=0; k<depth; k++) {
        grid[i][j][k].run();
      }
    }
  }
  for (int i=0; i<cols; i++) {
    for (int j=0; j<rows; j++) {
      for (int k=0; k<depth; k++) {
        grid[i][j][k].updateType();
      }
    }
  }
}
```

```
class CA {
  Vec3D loc = new Vec3D(0, 0, 0);

  int x;
  int y;
  int z;

  int type = 0;
  int futType = 0;
  int neigh = 0;

  CA(Vec3D _loc, int _x, int _y, int _z) {

    loc = _loc;
    x = _x;
    y = _y;
    z = _z;

    float rdm = random(100);
    if (rdm <60) {
      type = 1;
    }
  }

  void run() {
    display();
    calc();
  }

  void updateType() {
    type=futType;
  }

  void calc() {
    int count =0;

        if  (grid[(x+cols)%cols][(y+rows+1)%rows]
[(z+depth)%depth].type == 1) count++;

        if  (grid[(x+cols+1)%cols][(y+rows)%rows]
[(z+depth)%depth].type == 1) count++;

         if  (grid[(x+cols)%cols][(y+rows)%rows]
[(z+depth+1)%depth].type == 1) count++;

        if  (grid[(x+cols)%cols][(y+rows-1)%rows]
[(z+depth)%depth].type == 1) count++;

        if  (grid[(x+cols-1)%cols][(y+rows)%rows]
[(z+depth)%depth].type == 1) count++;

         if  (grid[(x+cols)%cols][(y+rows)%rows]
[(z+depth-1)%depth].type == 1) count++;

    if (type == 1 && count <2) {
      futType = 0;
    }
    if (type == 1 && count <=5 && count>=2) {
      futType = 1;
    }
    if (type == 1 && count > 4) {
      futType = 0;
    }

    if (type == 0 && count ==4) {
      futType = 1;
    }
  }

  void display() {

    if (type ==1) {

      pushMatrix();
      translate (loc.x, loc.y, loc.z);
      stroke(255);
      fill(255, 0, 0);
      box(5);
      popMatrix();

    }
  }
}
```

Expérimentation 3

Agents en recherche d'équilibre

```
import toxi.geom.*;
import peasy.*;

PeasyCam cam;
ArrayList group;
int f = frameCount;
int i;

void setup() {
  size(displayWidth, displayHeight, P3D);
  smooth();

  cam = new PeasyCam(this, 500);
  group = new ArrayList();

  for (i=0; i< 500; i++) {
   Vec3D startLoc = new Vec3D (random(-250, 250),
    random(-250, 250), random(-250, 250));

    Agent a = new Agent(startLoc);
    group.add(a);
  }
}

void draw() {
  background(240);

  stroke(20, 50);
  strokeWeight(1);
  noFill();
  box(500);

  for (i=0; i< group.size (); i++) {

    Agent nn = (Agent)group.get(i);
    nn.run();
  }
}
```

```
class Agent {

  float r = 3;
  float maxDist = 20;

  Vec3D loc = new Vec3D(0, 0, 0);
  Vec3D vel = new Vec3D(0, 0, 0);
  Vec3D acc = new Vec3D(0, 0, 0);

  Agent(Vec3D _loc) {
    loc = _loc;
  }

  void run() {
    display();
    update();
    target();
    findClosest();
  }

  void display() {

    noStroke();
    stroke(0);
    strokeWeight(r);
    point(loc.x, loc.y, loc.z);
  }
  void update() {

    vel.addSelf(acc);
    vel.limit(1);
    loc.addSelf(vel);

    acc.clear();
  }
  void target() {

    Vec3D target = new Vec3D(0, 0, 0);
    Vec3D dif = target.sub(loc);

    dif.normalize();
    dif.scaleSelf(10);

    acc.addSelf(dif);

    for (int i = 0; i< group.size (); i++) {
      float distance = loc.distanceTo(target);

      if (distance < 50) {
        loc.addSelf(dif);
      }
    }
  }

  void findClosest() {

    Vec3D close =new Vec3D();
    int count =0;

    for (int i=0; i<group.size (); i++) {
      Agent other = (Agent)group.get(i);

      float distance = loc.distanceTo(other.loc);

      if (distance < maxDist) {
        count++;

        stroke(0, 50);
        strokeWeight(0.3);
line(loc.x, loc.y, loc.z, other.loc.x, other.
loc.y, other.loc.z);

      }
    }
  }

  void cohesion(float magnitude) {

    Vec3D sum = new Vec3D();
    int count = 0;

    for (int i=0; i<group.size (); i++) {
      Agent other = (Agent)group.get(i);

      float distance = loc.distanceTo(other.loc);
      if (distance > 0 && distance < 100) {
        sum.addSelf(other.loc);
        count++;
      }
    }
    if (count > 0) {
      sum.scaleSelf(1.0/count);
    }
    if (sum.magnitude()>0) {
      Vec3D steer = sum.sub(loc);
      steer.scaleSelf(magnitude);
      acc.addSelf(steer);
    }
  }

  void separate(float magnitude) {

    Vec3D steer = new Vec3D();
    int count = 0;

    for (int i=0; i<group.size (); i++) {
      Agent other = (Agent)group.get(i);

      float distance = loc.distanceTo(other.loc);
      if (distance > 0 && distance < 100) {

        Vec3D diff = loc.sub(other.loc);
        diff.normalizeTo(1.0/distance);
        steer.addSelf(diff);
        count++;       }
    }
    if (count > 0) {
      steer.scaleSelf(1.0/count);
    }

    steer.scaleSelf(magnitude);
    acc.addSelf(steer);
  }
}
```

Expérimentation 4

Visualisation des données mondiales

```
import toxi.geom.*;
boolean record;
PImage img;

int Cols = 213;
int Rows = Cols;

Point grid [][] = new Point [Cols][Rows];
ArrayList ballCollection;
String [] data;

void setup() {
  size(1600, 900, P3D);
  img = loadImage(«base_map3.png»);
  import1();

}

void draw() {
  background(240);
  tint(255, 50);
  image(img, 0, 80);

  scale(5);
  translate(140, 100);
  rotate(radians(-90));

  for (int i = 0; i<Cols; i++) {
    for (int j = 0; j<Rows; j++) {
      grid[i][j].run();
    }
  }

  for (int i=0; i<ballCollection.size (); i++ )
{
    Point mb = (Point) ballCollection.get(i);
    mb.run();

  }
}

void import1() {

 data = loadStrings(«data/cities_5000_pop.csv»);

  int countX = 0;
  int countY = 0;
  ballCollection = new ArrayList();

  for (int i = 0; i<data.length; i++) {
       String [] fields = split(data[i], ‘;’);

    float x = float(fields[1]);
    float y = float(fields[2]);
    float pop = float(fields[3]);

Vec3D pointLoc = new Vec3D(x, y, 0);
grid[countX][countY] = new Point(pointLoc, pop);
ballCollection.add(grid[countX][countY]);

    countX++;
    if (countX == Cols) {
      countY++;
      countX = 0;
    }
  }

}
```

```
class Point {

  Vec3D loc = new Vec3D(0, 0, 0);
  Vec3D speed = new Vec3D(0, 0, 0);
  Vec3D acc = new Vec3D(0, 0, 0 );

  float pop;
  float fac = 500000;


  Point(Vec3D _loc, float size) {
    loc = _loc;
    pop = size;
  }

  void run () {
    display();
  }

  void display() {
    noStroke();
    fill(10);
    ellipse(loc.x, loc.y, 0.2, 0.2);
  }

}
```

Version courte n'incluant ni les grandes villes comme attracteurs, ni le comportement des points, le code étant trop long pour figurer sur une seule page.

Expérimentation 5
Branching

```
import peasy.*;
PeasyCam cam;

float theta;
float rot = 0;
float grow = 0.1;

void setup() {

  size(1600, 900, OPENGL);
  smooth();

  cam = new PeasyCam(this, 800);
}


void draw() {

  background(240);
  theta = PI/6;

  translate(width/2, height);
  stroke(30, 100);

  tree(400, 0);
}

void tree(float len, int level) {

  strokeWeight(0.5);
  line(0, 0, 0, -len);
  translate(0, -len);

  grow += 0.0000001;
  if (grow > 0.66) {

    grow = 0.66;
  }

  len *=grow;
  level++;
  rot += 0.000001;

  if (level<10) {

    pushMatrix();
    rotate(theta);
    rotateY(30);

    shearX(6);
    tree(len, level);
    popMatrix();

    pushMatrix();
    rotate(-theta);
    rotateY(-30+rot);

    shearX(-6);
    tree(len, level);
    popMatrix();


    pushMatrix();
    rotate(theta*0.8);
    rotateY(-45+rot);
    shearX(-6);
    tree(len, level);
    popMatrix();
  }
}
```


Réalisé d'après une introduction d'Alexander Kalachev

Expérimentation 6

Diffusion limited aggregation

```
import peasy.*;
PeasyCam cam;

float cc = 12;
float scaleFactor = cc/4;

int extentX = 50;
int extentY = 50;
int extentZ = 20;
boolean start=false;

Box [][][] lattice = new Box  [extentX][extentY]
[extentZ];

Walker walker = new Walker (2,2,2);

void setup(){
 size(1600,900,P3D);
 cam = new PeasyCam(this,500);

 for (int i=0; i<extentX; i++) {
    for (int j=0; j<extentY; j++) {
      for (int k=0; k<extentZ; k++) {
        lattice[i][j][k] = new Box (i,j,k);
      }
    }
  }
}

 void draw(){
      background(240);
      lights();

  if(start){
       if (walker.complete == true) {
    walker= new Walker (random(2,extentX-2),ran-
dom(2,extentY-2),random(3,extentZ-3));
  }
    while (walker.complete == false) {
    walker.step();
    walker.check();
  }

 fill(255);
   for (int i=0; i<extentX; i++) {
    for (int j=0; j<extentY; j++) {
      for (int k=0; k<extentZ; k++) {
        lattice[i][j][k].display();
      }
    }
  }
 }
}
```

```
class Box {

  float posX;
  float posY;
  float posZ;

  boolean on = false;
  Box (float x, float y, float z) {

    posX = x*cc;
    posY = y*cc;
    posZ = z*cc; //cc/2

   if (x == int(extentX/2) && y == int(extentY/2)
&& z == int(extentZ/2)) {
      on=true;
    }
  }

  void display() {

    pushMatrix();
    translate (posX, posY, posZ);

  if (on == true) {
      scale(cc);
      stroke(0);
      strokeWeight(0.1);
      fill(255);
      box(1);
    }
    popMatrix();
  }
}
```

```
class Walker {
  int posX;
  int posY;
  int posZ;

  boolean complete = false;

    Walker (float x, float y, float z) {
    posX = int(x);
    posY = int(y);
    posZ = int(z);
  }

  void step() {
    boolean moved = false;

    while (moved == false) {
      float dir = random(6);

      if (dir < 1  && posZ!=extentZ-3 ) {
        posZ = posZ + 1;
        moved = true;
      } else if (dir < 2 && posZ!=2 ) {
        posZ = posZ - 1;
        moved = true;
      } else if (dir < 3 && posX!=extentX-2) {
        posX = posX + 1;
        moved = true;
      } else if (dir < 4 && posX!=1 ) {
        posX = posX - 1;
        moved = true;
      } else if (dir < 5 && posY!=extentY-2) {
        posY = posY + 1;
        moved = true;
      } else if (dir < 6 && posY!=1) {
        posY = posY - 1;
        moved = true;
      }
    }
  }

  void check() {

    if (

   lattice [posX  +1  ][posY] [posZ].on == true ||
   lattice [posX  -1  ][posY][posZ].on == true ||
   lattice [posX][posY  +1 ] [posZ].on == true ||
   lattice [posX][posY  -1] [posZ]  .on == true ||
   lattice [posX][posY][posZ  -1 ]  .on == true ||
  lattice [posX][posY][posZ  +1 ]  .on == true )

    {
      lattice[posX][posY][posZ].on = true;
      complete = true;
    }
  }
}
```

Inspiré du code de David Andreen, qui lui utilise des octahèdres tronqués dans sa simulation. http://www.openprocessing.org/sketch/22793

BIBLIOGRAPHIE ET SOURCES

Livres

Alexander, Christopher - A city is not a tree (Extrait pdf) 1965
Batty, Michael - Cities and Complexity - 2005
Boesiger, W - Le Corbusier Oeuvre complète 1946-1952 - 1955
Brown, Richard - 3 minutes pour comprendre les 50 plus grandes théories mathématiques
Carpo, Mario – The alphabet and the Algorithm - 2011
Choma, Joseph - Morphing A guide to mathematical transformations for architects and designers- 2015
Ellul, Jacques - Le système technicien - 1977
Glendinning, Paul - Mathématiques minutes - 2014
Leach, Neil & Yuan, Philip – Fabricating the Future - 2012
Leach, Neil & Yuan, Philip – Scripting the Future - 2012
Rushkoff, Douglas – Program or be programed – 2010
Rushkoff, Douglas – Present Shock - 2013
Smith, Leonard - Chaos a very short introduction - 2007
Wassim Jabi - Parametric Design for Architecure - 2013
Werner, Liss C. – EnCoding Architecture - 2014

Revues

Dosya n°29 – 2012 – Computational design
AD août 2011 - Mathematics of space
AD juillet 2013 – System city
Tangent hors series n°18 – 2004

Travaux de recherche

Davis, Daniel - Modelled on Software Engineering : Flexible Parametric Models in the Practice of Architecture - 2013
Nandha, Rohit - Past present and future of parametric architecture - 2015
Riiber, Jacob - Generative processes in architectural design - 2011
Rosenberg, Daniel - Designing for uncertainty - 2009
Sumi, Kavi - Parametric design, a new paradigm in architecture - 2014

Web

Sur Cedric Price

http://jilliancrandall.net/theatricality-of-performance-systems/
www.joh.cam.ac.uk/anti-building-future-world-cedric-price
www.cca.qc.ca/fr/collection/283-cedric-price-fun-palace
http://complexitys.com/francais/fun-palace/

www.aiacc.org/2012/06/25/parametric-design-a-brief-history/
www.architectural-review.com/essays/architecture-and-the-computer-a-contested-history/8678167.article
www.makery.info/2015/08/03/un-design-lab-revisite-le-jeu-de-la-vie-en-architecture/
http://mx3d.com/projects/bridge/

www.pangaro.com/definition-cybernetics.html
www.selfassemblylab.net/
www.unstudio.com

Programmation

www.natureofcode.com
www.plethora-project.com
www.processing.org
www.openprocessing.org

Conférences

EMCSR 2014
IS4IS Summit Vienna 2015
ISSS Berlin 2015

Videos

TedTalk - Self-assembly labhttps://www.youtube.com/watch?v=0gMCZFHv9v8
AADRL Design studio - https://www.youtube.com/watch?v=nof22Sl3DGc

Autre

Musée des technologies de Berlin

ARCHITECTURE ET CODE

Programmer des systèmes de génération de forme

L'objectif de ce mémoire est de discuter les enjeux de conception sous-jacent à l'approche paramétrique et algorithmique de l'architecture. De ce fait, traiter du changement de paradigme par rapport à une approche traditionnelle représentative, c'est-à-dire par le dessin.

Quelle nouvelle relation entretient l'architecte vis à vis de ces outils ? Comment cette démarche conceptuelle peut faire émerger de nouvelles façons de dessiner un projet ? Quelle pertinence y-a-t-il à adopter cette approche ?

Eliot Marin-Cudraz