Generators, Patterns, and Oblique Strategies

Jonathan Bachrach

EECS UC Berkeley

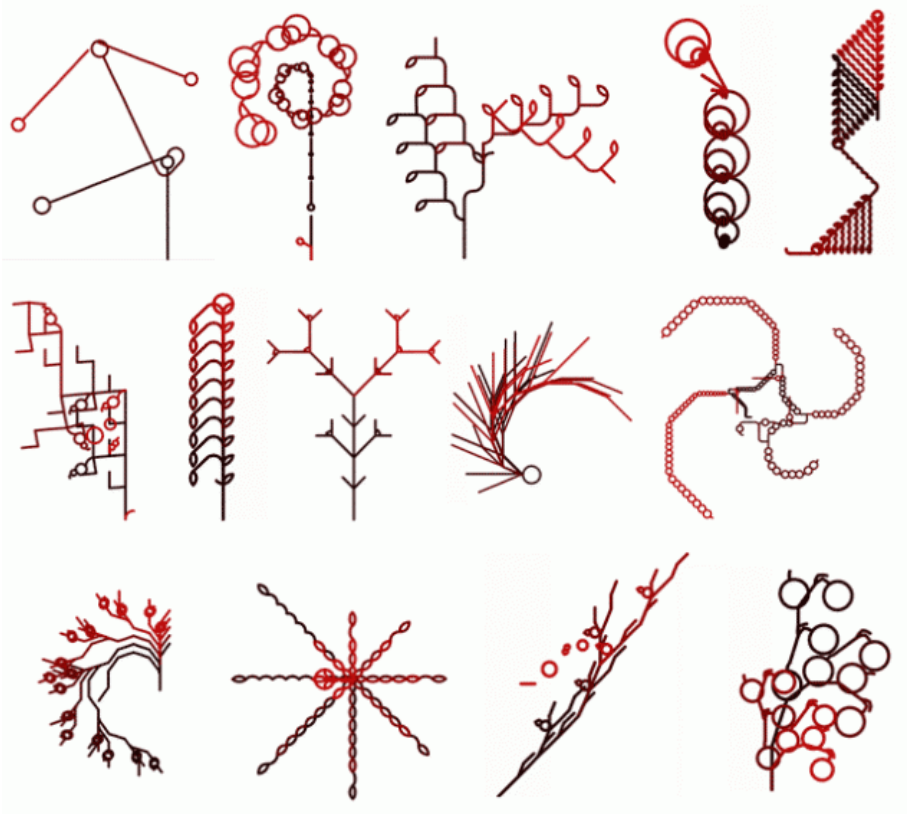
October 20, 2015

News

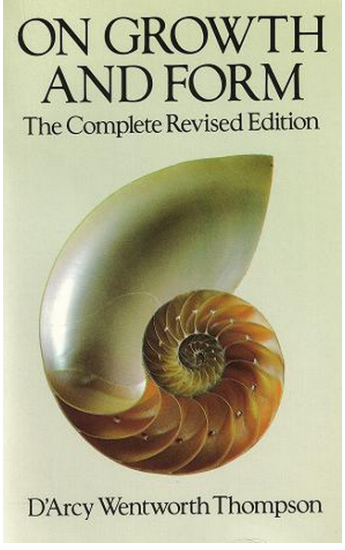
- last lab 6 due thursday
- one more lecture on thursday
- proposals/presentations due next week
- how many groups?

This Time: Generators and Patterns

- Generative Programming
- Generating Patterns - L-Systems
- Design Patterns
- Oblique Strategies



- create form using biological growth means
- biological process that causes an organism to develop shape
- bottom up goal seeking process
- digital form broadens notions and uses it within design
- also called *form finding*



Digital Morphogenesis Techniques

- cellular automata
- genetic algorithms
- l-systems
- soap bubbles

- string rewriting system
- invented by Aristid Lindenmayer 1968
- applied to graphics by Alvy Ray Smith 1984
- applied to plant growth by Przemyslaw Prusinkiewicz in 1990

- *grammar* on an *alphabet* of *symbols* like `F`, `+`, and `-`
- set of *productions* –
  - replacement of a nonterminals with string of zero or more symbols
  - `F -> F+F - - F+F`
  - every `F` in input string replaced by rhs string
- seeded by an axiom such as `F`

Rule Firings

- Rule: `F -> F+F - - F+F`
- Seed: `F`

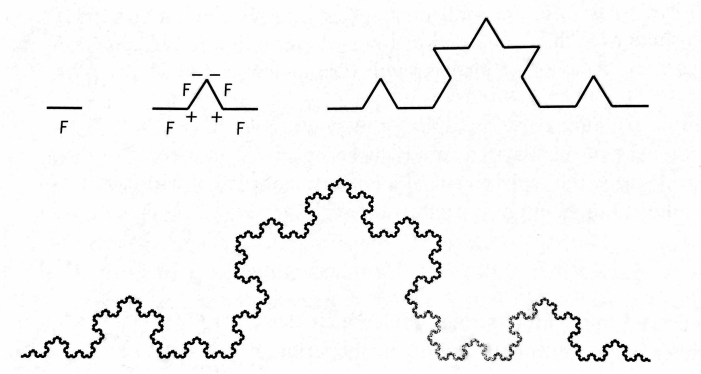
- productions are chosen in parallel
  - does not depend on the order the productions are applied
- deterministically chosen

1. `F`
2. `F+F - - F+F`
3. `F+F - - F+F + F+F - - F+F - - F+F - - F+F + F+F - - F+F`

Interpretation of Symbols

- often symbols are given geometric meanings
- think turtle graphics

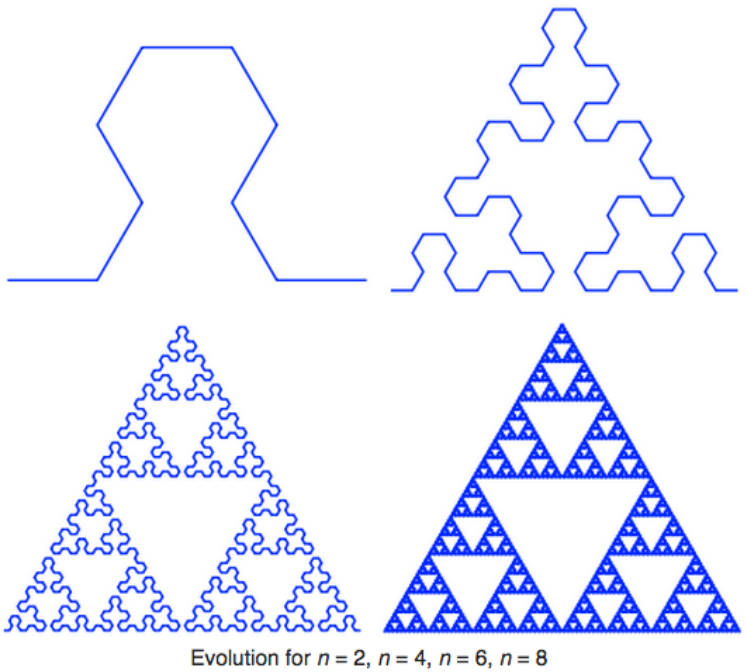
- F – forward leaving trail
- \+ – rotate counter clockwise
- \- – rotate clockwise



procedural synthesis of geometry by hart

Sierpinski Triangle

- **start:** `A`
- **rule1:** `A -> B-A-B`
- **rule2:** `B -> A+B+A`
- **angle:** 60 degrees

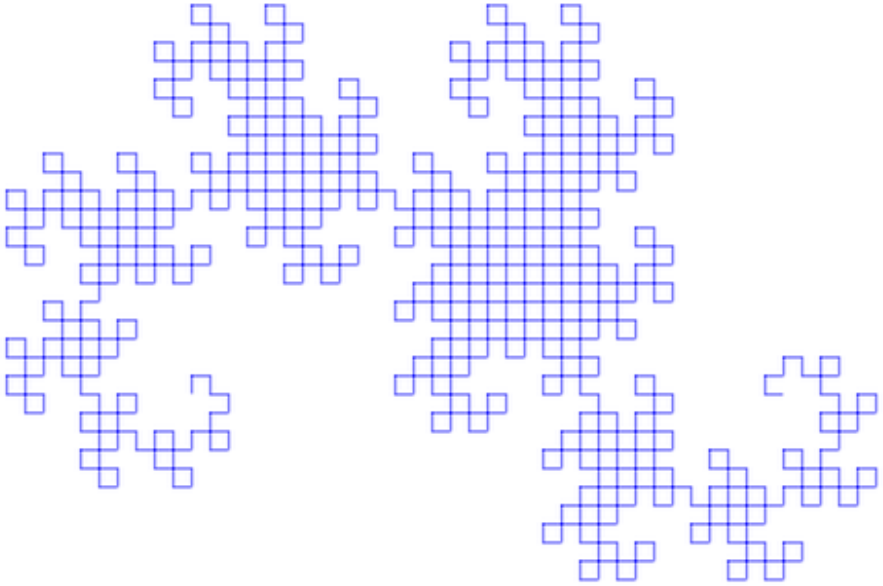


Evolution for $n = 2$, $n = 4$, $n = 6$, $n = 8$

wikipedia

Dragon Curve

- **start:** `FX`
- **rule1:** `X -> X+YF+`
- **rule2:** `Y -> -FX-Y`
- **angle:** 90 degrees



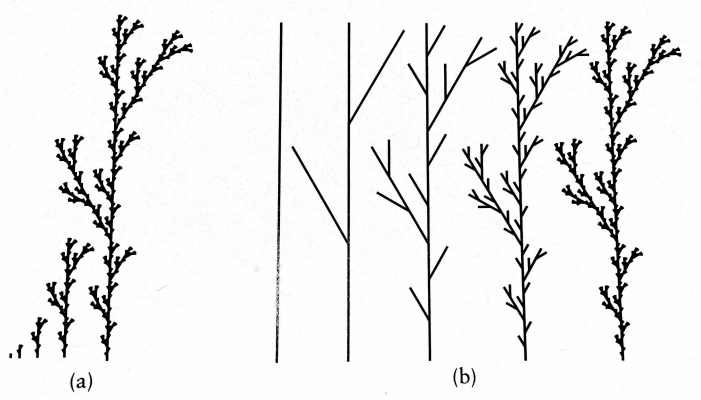
Dragon curve for $n = 10$

wikipedia

Branching

- could retrace back to base of branch or
- could save and restore state of turtle
- introduce stack and new symbols:
  - `[` – push state onto stack
  - `]` – pop recently state from stack
- where state is position and orientation of turtle

- `F -> F[+F]F[-F]F`



procedural synthesis of geometry by hart

Fractal Tree

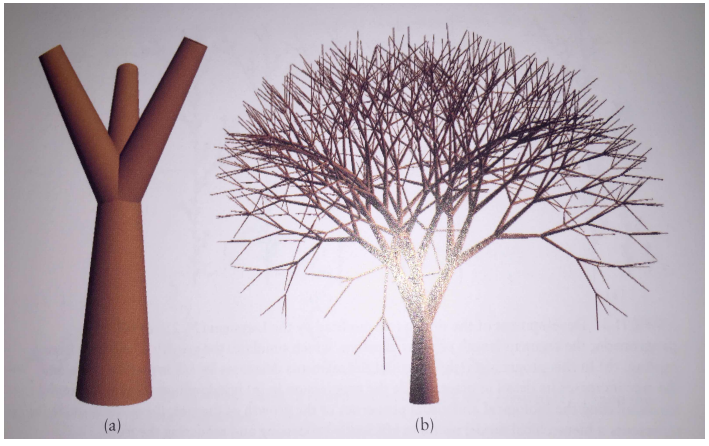
- **start:** `X`
- **rule1:** `X -> F-[[X]+X]+F[+FX]-X`
- **rule2:** `F -> FF`
- **angle:** 25 degrees

Fractal plant for $n = 6$

wikipedia

- `+` and `-` – change yaw (i.e., left right)
- `^` and `&` – change pitch (i.e., up and down)
- `\` and `/` – change roll (i.e., left and right)

- `F -> F[&F][&/F][&\F]`

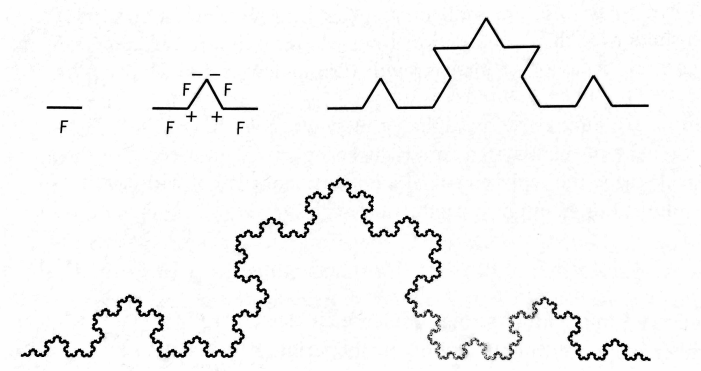


procedural synthesis of geometry by hart

Parameters

- want to specify different segment lengths and rotation angles
- elements can be parameterized using parentheses
- `F(30)` draws 50 unit segment
- `+(30)` rotates the turtle 30 degrees

- `F -> /(180)-(30)F+(60)F-(30)\(180)`



procedural synthesis of geometry by hart

- `!` – decreases the width of all segments
- `!(d)` – decreases the width by `d`
- saved and restored by `[` and `]` respectively

- `@0` – places sphere around current position

- `@Gs` – start curve
- `@Gc` – control point
- `@Ge` – end curve

Polygons

- `{` – start polygon
- `}` – end polygon
- `F` – describes edge
- `.` – defines vertex at current position
- `G` – move without defining a vertex

- also bicubic patches

- colors predefined in color map
- `;(f,b)` – frontside color to index f and the backside color to index b
- `[` – pushes color
- `]` – pops color

- say have brown, yellow, and green for branches, twigs, and stems, then
- `F[;+F[;+F][;-F]][;-F[+F][-F]]`
- will cycle through colors

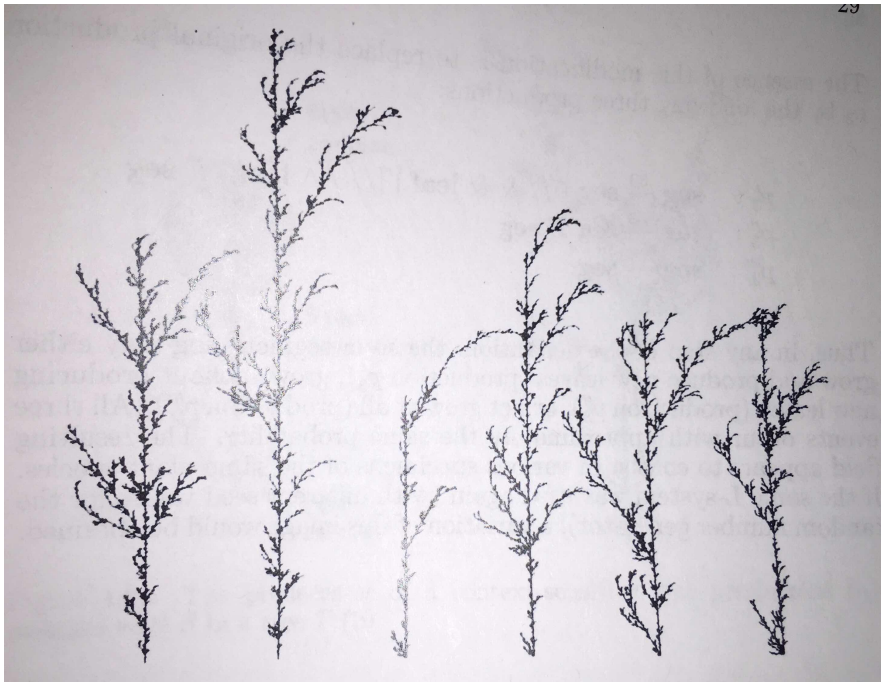
- `!` – decreases the width of all segments
- `!(d)` – decreases the width by `d`
- saved and restored by `[` and `]x` respectively

Stochastic L-systems

- probabilities on rules

example:

- `F ->(0.33) F[+F]F[-F]F`
- `F ->(0.33) F[+F]F`
- `F ->(0.34) F[-F]F`



l-system book

- rules may depend on predecessor's context
- used for simulating interactions between parts
- context sensitive rules have priority
- pre form `b < a -> b`
- pre/post form `b < a > c -> b`

example:

- `baaaaaaa`
- `b < a -> b`
- `b -> a`

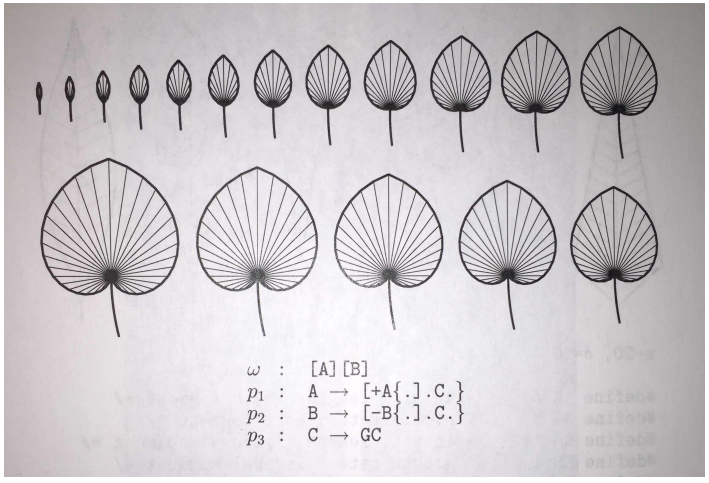
1. `baaaaaaa`
2. `abaaaaaa`
3. `aabaaaaa`
4. `aaabaaaa`
5. `aaaabaaa`
6. ...

- parameters on rules
- predicates that have to be true

example:

- `B(2)A(4,4)`
- `A(x,y) : y <= 3 -> A(x*2, x+y)`
- `A(x,y) : y > 3 -> B(x)A(x/y, 0)`
- `B(x) : x < 1 -> C`
- `B(x) : x >= 1 -> B(x-1)`

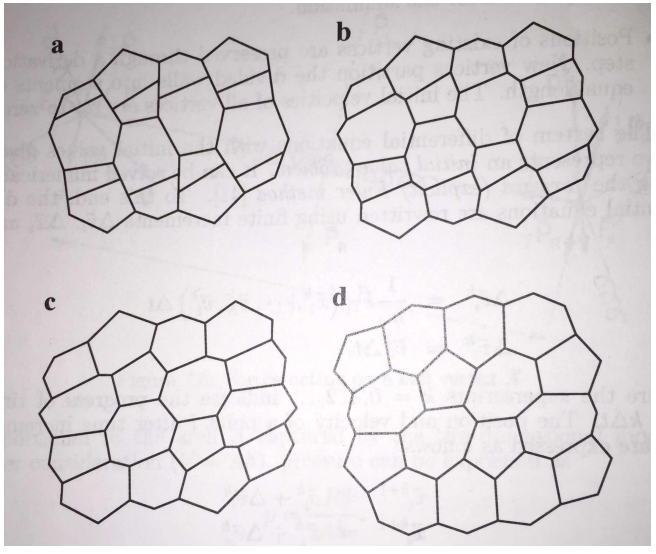
- parameters on rules
- predicates that have to be true



Algorithmic Beauty of Plants Book by Prusinkiewicz

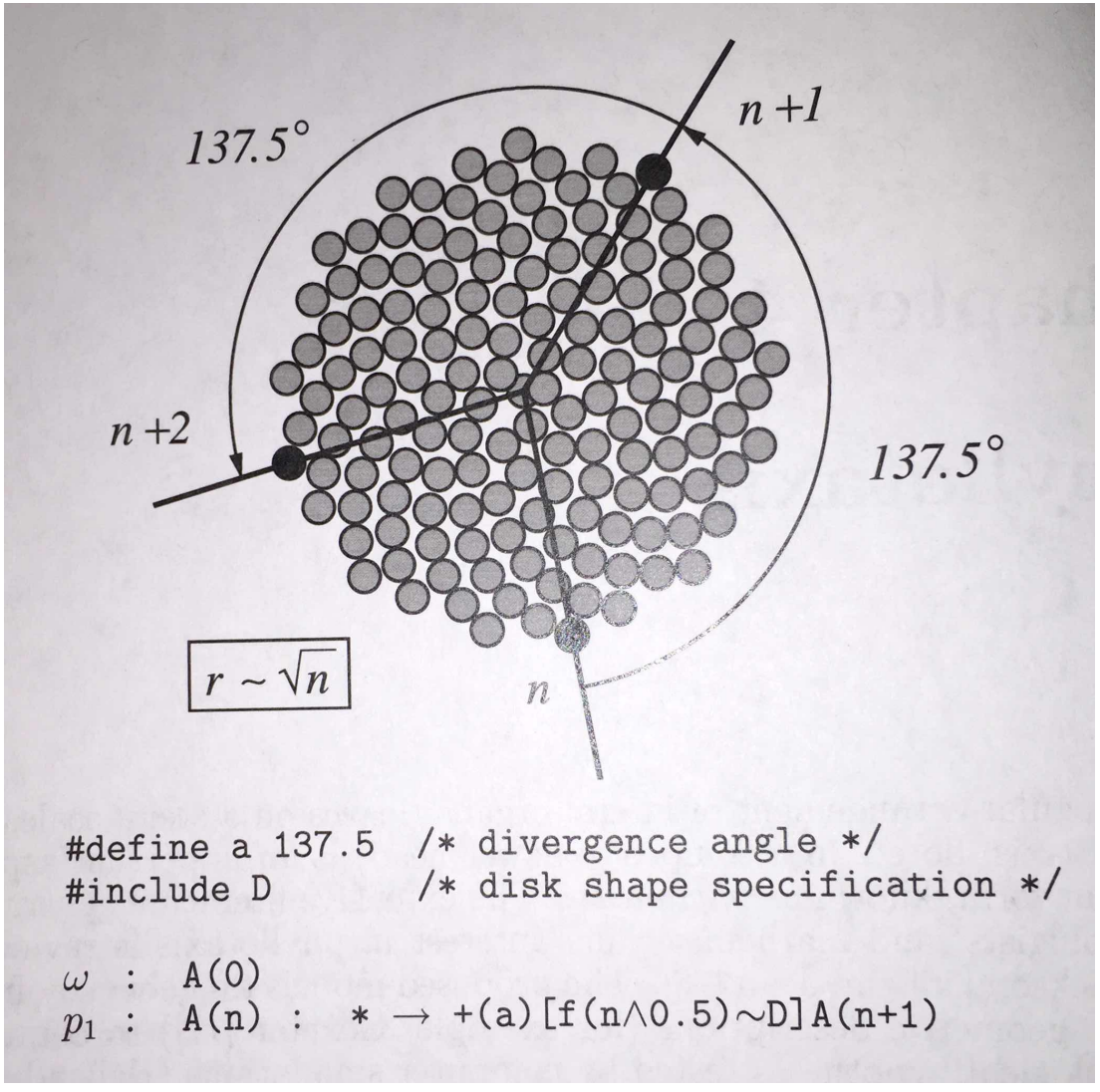
- finite set of regions
- each region is surrounded by boundary consisting of finite circular sequence of edges that meet at vertices
- each edge has one or two vertices associated with it
- every edge is a part of the boundary of a region
- the set of edges is connected (no islands)

- binary because regions split into two
- progagating in the sense that the edges cannot be erased
- regions (cells) cannot fuse or die
- markers specify the positions of inserted edges that split the regions



Algorithmic Beauty of Plants Book by Prusinkiewicz

Disk Packing




```
#define a 137.5  /* divergence angle */
#include D       /* disk shape specification */

ω  :  A(0)
p1 :  A(n) :  * → +(a)[f(n^0.5)~D]A(n+1)
```

Algorithmic Beauty of Plants Book by Prusinkiewicz

Organic Uses of L-Systems

Green

- parameterized functions
- randomly generated parameters
- recursive functions
- constraints
- emergence
- genetic algorithms

Deterministic Tree

```
defn gen_tree (depth, max_depth, from, dist, angle) :
  me = from + dist * [sin(angle), cos(angle)]
  draw(from, me)
  if depth < max_depth :
    inc = 2 * PI / depth
    gen_tree(depth + 1, max_depth, me, dist * 0.5, angle - inc)
    gen_tree(depth + 1, max_depth, me, dist * 0.5, angle + inc)

gen_tree(1, 4, [0.0, 0.0], 10, PI / 2.0)
```

- range – low to high
- distribution – uniform, gaussian
- perlin noise

```
defn gen_tree (depth, max_depth, from, dist, angle) :
  me = from + dist * [sin(angle), cos(angle)]
  draw(from, me)
  if depth < max_depth :
    inc_mean = 2 * PI / depth
    inc = inc_mean + 0.25 * rndf(-inc_mean,inc_mean)
    ndist = rndf(dist * 0.25, dist * 0.75)
    gen_tree(depth + 1, max_depth, me, ndist, angle - inc)
    gen_tree(depth + 1, max_depth, me, ndist, angle + inc)

gen_tree(1, 4, [0.0, 0.0], 10, PI / 2.0)
```

- tree in sun
- tree in room
- two trees in sun
- ivy on fence
- growing tree

Controlling Generative Systems

- you're the critic
- searching over runs
- filtering results
- searching over parameters
- optimization in process

Fit and Diverse: Set Evolution for Inspiring 3D Shape Galleries

Kai Xu*† Hao Zhang‡ Daniel Cohen-Or§ Baoquan Chen*
*Shenzhen VisuCA Key Lab/SIAT †National University of Defense Technology (NUDT)
‡Simon Fraser University §Tel-Aviv University

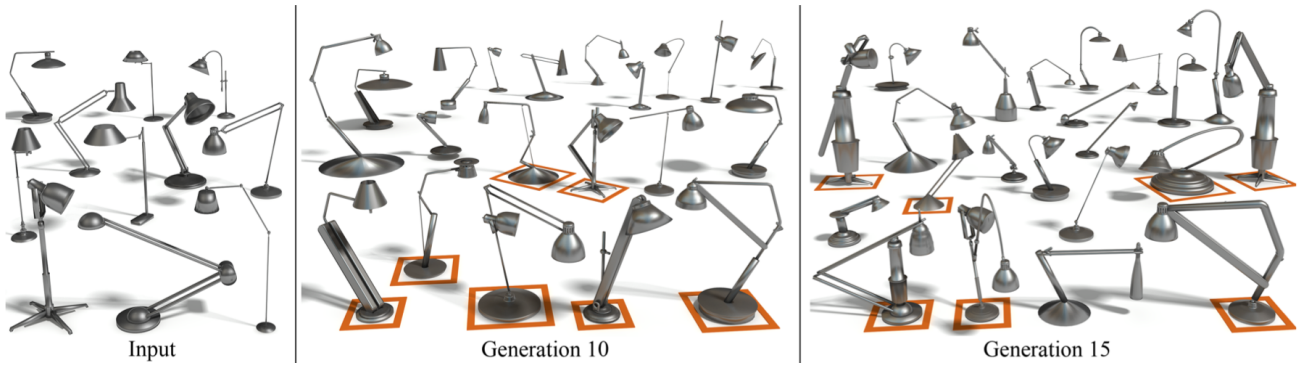


Figure 1: *Set evolution starting from a small input set of lamps (left). With the set evolution "fit and diverse", new generations of shapes are not only fit to be lamps but also exhibit significant and potentially inspiring variations.*

- general reusable solution to recurring problem within a given context
- not a finished design that can be transformed directly into usage
- description or template for how to solve a problem
- formalized from best practices

- architect
- taught at uc berkeley in CES
- wrote pattern language book in 1977
- 253 patterns in original book from large to small scale
- empower users of architecture
- encouraged broadening beyond architecture

- method for describing good design practices within a field of expertise
- attempt to express deeper wisdom through set of interconnected expressions
- document key ideas that make a good system different from a poor system
- syntax
- grammar
- network of patterns that call upon one another
- used in combination to create solutions
- decomposition into smaller problems
- eventually problems are small enough to be solved by builders

like a dictionary entry

- name
- descriptive entry
- cross references
- explain why that solution is good in the pattern's context

- **Name** – ChocolateChipRatio
- **Context** – You are baking chocolate chip chookies in small batches for family and friends
- **Consider First** – SugarRatio, FlourRatio, EggRatio
- **Problem** – Determine the optimum ratio of chocolate chips to cookie dough
- **Solution** – Observe that most people consider chocolate to be the best part of the chocolate chip cookie. also observe that too much chocolate may prevent the cookie from holding together, decreasing its appeal. Since you are cooking in small batches, cost is not a consideration. Therefore, use the maximum amount of chocolate chips that results in a really sturdy cookie.
- **Consider Next** – NutRatio or CookingTime or FreezingMethod

- need to get a reliable complete list of problems to be solved
- on-site improvisation by concerned empowered users
- create large scale initial solution
- hierarchical or iterative application
- same pattern can be used millions of times without ever doing it the same way twice

- `www.patternlanguage.com`
- These tools allow anyone, and any group of people, to create beautiful, functional, meaningful places.
- You can create a living world.

Kitchen Design Step 1

Making decisions in the right order when you lay out a kitchen controls the resulting comfort of the kitchen. In some cases the "right" order is highly unexpected.

If you get these steps in the right order, there is a good chance the kitchen will be comfortable. The windows and the table dominate everything.

KITCHEN

STEPS:

source www.patternlanguage.com

Kitchen Design Step 2

A PROCESS FOR MAKING A KITCHEN

1. STEP 1: MENTAL PREPARATION
2. STEP 2: THE WHOLE: ESTABLISH ROUGH PARAMETERS AND SOURCES OF LIGHT
3. STEP 3: THE FIRST CENTER: PUT THE TABLE IN THE BEST PLACE
4. STEP 4: THE NEXT CENTER: ESTABLISH POSITION AND SIZE OF YOUR MAIN WORK SPOT
5. STEP 5: PUT COUNTER AND THICK WALLS ALL AROUND THE ROOM
6. STEP 6: PLACE SUBSIDIARY CENTERS: STOVE, SINK, AND REFRIGERATOR
7. FINISH

STEPS:

source www.patternlanguage.com

- Relax and Be Truthful and Make Notes
- Your feelings are the most important. Just be unworried, relaxed about trial and error, and be very truthful with yourself about what makes you feel good.
- Spend a lot of time looking at photographs of kitchens that you really like and articulate to yourself what it is you like about them. Keep them in a folder or marked in books so that you can easily refer to them.
- Think a lot about yourself in the kitchen, the other people you live with and what kinds of activities are important. You may entertain and enjoy cooking or maybe elaborate meals are a rare event. It may be really important to you to be able to have young children play in the kitchen during meal preparation, or to be able to keep an eye on older children playing in the garden. There may be a student in the house who would like to work late at night at the kitchen table with coffee pot close at hand. You may have a dog or cat and they need to be included.
- YOU NOW HAVE YOUR NOTES ON WHAT SHOULD BE HAPPENING IN THE KITCHEN AND IDEAS FROM KITCHENS YOU LIKE.

Organize your notes into potential centers

- Kitchens like all other rooms are composed of centers. Typical centers will be either objects like windows or sink or activity areas such as the table and chairs (or benches) and the quiet bubble of air that surrounds it and working counter area. Each one, as it comes up in the sequence will receive your full attention as to its own organization and how it relates to the ones already in place. In the end they will all support each other in a collective strength which gives you, whether your kitchen is small or large, modest or high end, a comfortable place to work and just "be."
- Keep thinking of this room as a major place to "be" in... a place that is not a utility, but a real center for life in the house. Your choice of centers is personal. It could include or not a connection to the outside, a work spot for crafts, bar stools and high counter, a service window to a formal dining area, a place to grow herbs. It's YOUR kitchen.
- YOU NOW HAVE YOUR IDEAS ON WHAT CENTERS YOU WANT

establish rough parameters and sources of light

- Keep the Big Picture in Mind
- As you add or change elements in the design process, consider how each change impacts the kitchen as a whole. In the initial phase consider the room itself, its size and shape. Sometimes a small kitchen can be made bigger or have a more comfortable shape by moving walls, making half walls, or extending a window area. Sometimes we simply must make do with the space we've got.
- Work with a pad of tracing paper and make a basic outline of the room with planned or existing windows and doors and keep it as a template that you can quickly retrace again and again as you experiment. It can be a rough outline like in the short movie clip in the preceding page as long as the proportions are right.
- Does the room need more light? Kitchens do need generous daylight so you may now decide where a larger or an additional window should be by standing in the room. Mark the position on your drawing.
- Make a beautiful window which will bring life and air to the room. Have a look at patternlanguage.com for the sequences on window placement and window design.

Start by locating the kitchen table

- The table will be the first and most important center. It will be where you share meals, talk, work, and relax with a cup of coffee. Even if you can only have a small table, it should be in the best position, a place where you really want to “be” because of light, view, and a sheltered position in regard to traffic in the room.
- The table is the source of pleasure and of practical work together. A low hanging lamp provides a pool of light in the evening which will be more comfortable than homogeneous lighting.

If possible, place a second main center, on one edge of the room, in such a way that it adds life to the room.

- This second major center might be a fireplace near the table, and connected with it. Or the second major center which orients the room might be a real oil-burning stove like an Aga. Or it might be something you just plain like such as a deep windowsill that can hold flowering plants. So look back at the first step, where you identified activities or features that are important to you in a kitchen. Pick one of these as your second center.
- Place the second center in the kitchen which makes it a meaningful "place" for you and which, like the table, gives a focal point to the room. Reconsider the "whole" room and check that the placement of the table and this possible second center now give more feeling and structure to the whole space.
- YOU HAVE NOW PLACED TWO CENTERS

Now ask yourself where in the room you would like to stand while preparing food?

- Wherever it is, put a working counter there. Make sure it is going to be a truly comfortable place to work, deep enough, and long enough. Imagine yourself standing there, and make it so it is a pleasure to be there.
- The counter is the source of pleasure in work and of practical work together. This spot in the room which is comfortable for you needs to be determined BEFORE you place the appliances.
- YOU NOW HAVE THE MOST IMPORTANT WORKSPOT

Put counter and thick walls all around the room

- Now make an imaginary swath or band, about 2 feet thick, almost all around the room.
- The space will be used for counters, cupboards, and thick walls – wrapping around the table and making it comfortable. When you have done this, you may need to adjust the position of the table in the best place, taking into account this swath of counters and cupboards and trying to make sure the table is still as pleasant and comfortable as possible.
- YOU HAVE NOW DRAWN YOUR THICK WALLS AND MORE FINELY ADJUSTED YOUR TWO CENTERS
- The “thick wall and counter zone” is crucial to the comfort and effectiveness of the room as a kitchen. Thick walls are what identifies a house as yours.

Put counter and thick walls all around the room

- Make the counter continue around the room, as far as comfortable and useful.
- Some of this band may be storage, or cupboards; the rest of it can be open working top. Remember that deep cupboards are often not handy because what you want is always behind something else.
- Even cupboards and shelves help to make the room pleasant - because they are practical.

Three Counter Areas

Divide the counter part into three areas:

- the main central working top, you have already settled on,
- the stove, and
- the sink and draining board.

- Each of these should be made to feel like a center in its own right. However, do not exaggerate the importance of the positions of these last two; they should be pleasant and convenient and, typically, not more than ten feet between them, but it is important not to tie yourself in knots trying to make them meet special conditions. The kitchen will work alright anyway. Also remember that if you are remodeling an existing kitchen, the placement of the sink and stove can be changed; it is not necessarily a big deal to move them to where they really belong.
- The counter swath that runs around the room forms its boundary and its enclosure.

Do not let the refrigerator mess up the kitchen layout in any way.

- You will find that you can put it out of the way and your kitchen will still work well.
- Make the room beautiful and comfortable: this is what is important.
- YOU NOW HAVE PLACED THE SUBSIDIARY CENTERS
- You are now ready to proceed with implementation either by yourself or through a contractor.

Thoughts

- how do you use these patterns multiple ways?
- is it general enough?
- how do we execute it?
- how do we involve humans + computers?
- will this work?
- does it scale?

- name
- descriptive entry
- cross references
- explain why that solution is good in the pattern's context

Software Design Pattern Sections

- **Pattern Name and Classification:** A descriptive and unique name that helps in identifying and referring to the pattern.
- **Intent:** A description of the goal behind the pattern and the reason for using it.
- **Also Known As:** Other names for the pattern.
- **Motivation (Forces):** A scenario consisting of a problem and a context in which this pattern can be used.
- **Applicability:** Situations in which this pattern is usable; the context for the pattern.
- **Structure:** A graphical representation of the pattern. Class diagrams and Interaction diagrams may be used for this purpose.
- **Participants:** A listing of the classes and objects used in the pattern and their roles in the design.
- **Collaboration:** A description of how classes and objects used in the pattern interact with each other.
- **Consequences:** A description of the results, side effects, and trade offs caused by using the pattern.
- **Implementation:** A description of an implementation of the pattern; the solution part of the pattern.
- **Sample Code:** An illustration of how the pattern can be used in a programming language.
- **Known Uses:** Examples of real usages of the pattern.
- **Related Patterns:** Other patterns that have some relationship with the pattern; discussion of the differences between the pattern and similar patterns.

- provide an interface for creating families of related or dependent objects without specifying their concrete classes
- This insulates client code from object creation by having clients ask a factory object to create an object of the desired abstract type and to return an abstract pointer to the object.[4]
- see also Concrete class, Factory method pattern, Object creation

```
class MotifFactory ...;
factory = new MotifFactory;
...
CreateWindow(factory, x, y);
```

Criticism

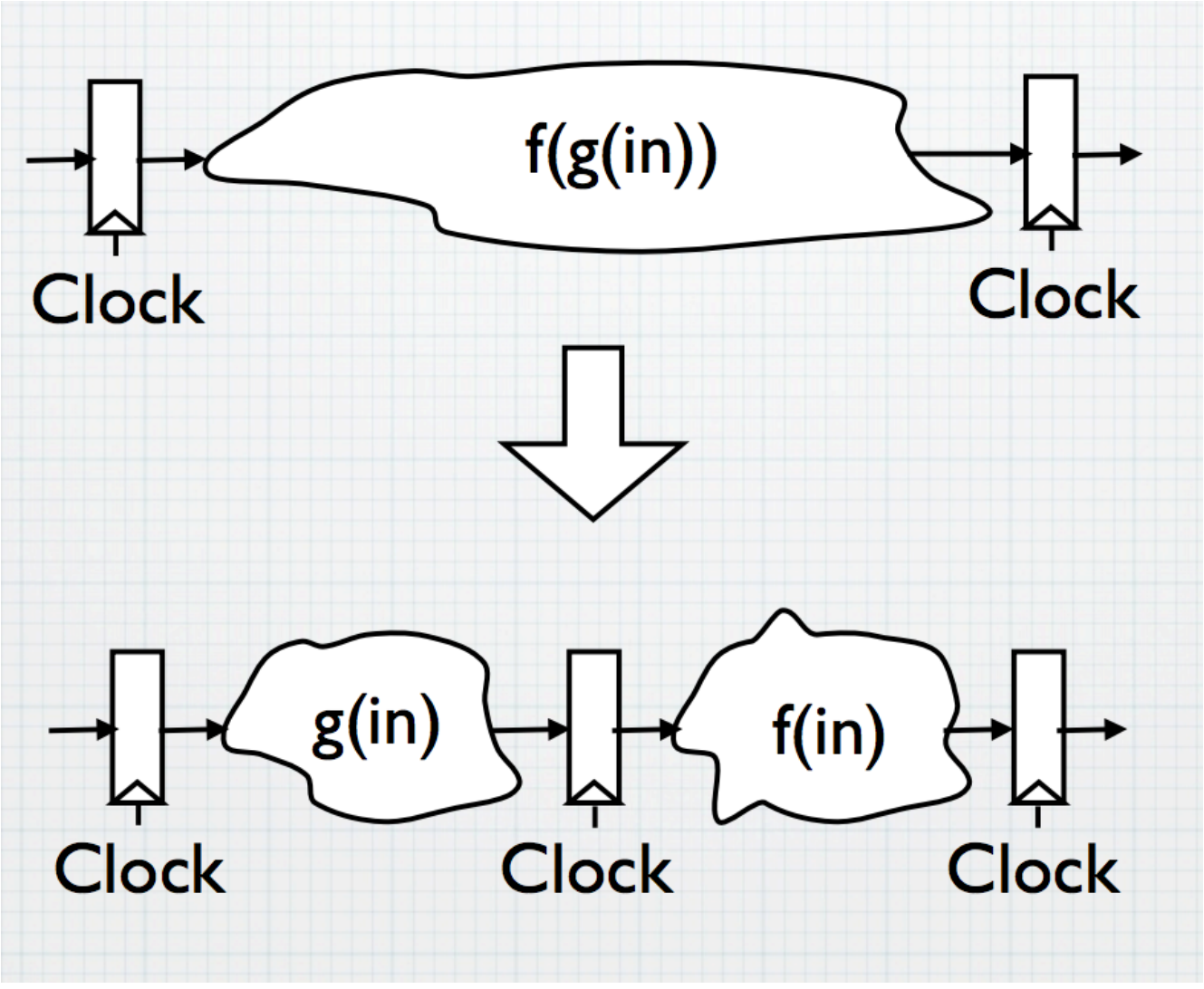
- shows signs of missing features in given programming language
- simplified or eliminated in more powerful languages
- inappropriate use may unnecessarily increase complexity
- informal – pattern in prose, but implemented from scratch each time

- programming language has this facility
- much easier to use and concrete and composable
- much more seamless implementation
- implement pattern within the language and instantiate/ call it for each use

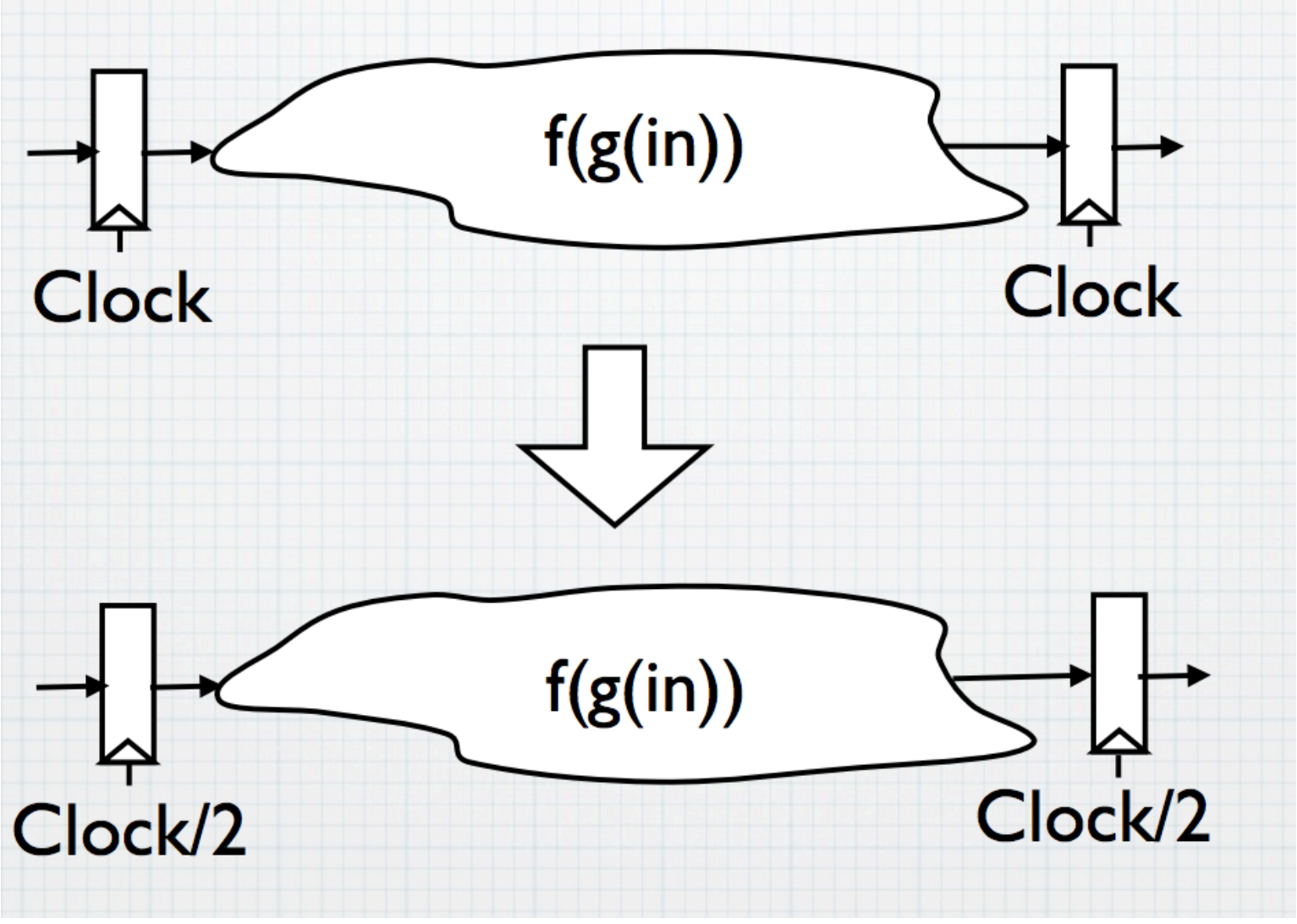
```
window-type := <motif-window>;
...
make(window-type, x, y);
```

- **Problem:** Describe the particular problem the pattern is meant to solve. Should include some context (small, high throughput), and also the layer of the pattern hierarchy where it fits.
- **Solution:** Describe the solution, which should be some hardware structure with a figure. Solution is usually the pattern name. Should not provide a family of widely varying solutions - these should be separate patterns, possibly grouped under a single more abstract parent pattern.
- **Applicability:** Longer discussion of where this particular solution would normally be used, or where it would not be used.
- **Consequences:** Issues that arise when using this pattern, but only for cases where it is appropriate to use (use Applicability to delineate cases where it is not appropriate to use). These might point at sub-problems for which there are sub-patterns. There might also be limitations on resulting functionality, or implications in design complexity, or CAD tool use etc.

- **Problem:** Combinational function of operator has long critical path that would reduce system clock frequency. High throughput of this function is required.
- **Solution:** Divide combinational function using pipeline registers such that logic in each stage has critical path below desired cycle time. Improve throughput by initiating new operation every clock cycle overlapped with propagation of earlier operations down pipeline.
- **Applicability:** Operators that require high throughput but where latency is not critical.
- **Consequences:** Latency of function increases due to propagation through pipeline registers, adds energy/op. Any associated controller might have to track execution of operation across multiple cycles.


f(g(in))
Clock
Clock
g(in)
f(in)
Clock
Clock
Clock

- **Problem:** Combinational function of operator has long critical path that would reduce system clock frequency. High throughput of this function is not required.
- **Solution:** Hold input registers stable for multiple clock cycles of main system, and capture output after combinational function has settled.
- **Applicability:** Operators where high throughput is not required, or if latency is critical (in which case, replicate to increase throughput).
- **Consequences:** Associated controller has to track execution of operation across multiple cycles. CAD tools might detect false critical path in block.


f(g(in))
Clock
Clock
f(g(in))
Clock/2
Clock/2

Automatic Rate Balancing

- identify bottlenecks and fatnecks
- **back pressure** – producer has data but consumer not ready
- **under utilization** – consumer ready for data but producer not valid
- adjust using pipelining and multicycle operators until even

Challenge of Pattern Usage

- amounts to automatic programming which has never really taken off
- program synthesis has which is more tractable
- another possibility is expression as cost function with optimization

(a) Clearance and reachability term excluded

(b) Alignment term excluded

(c) Emphasis term excluded

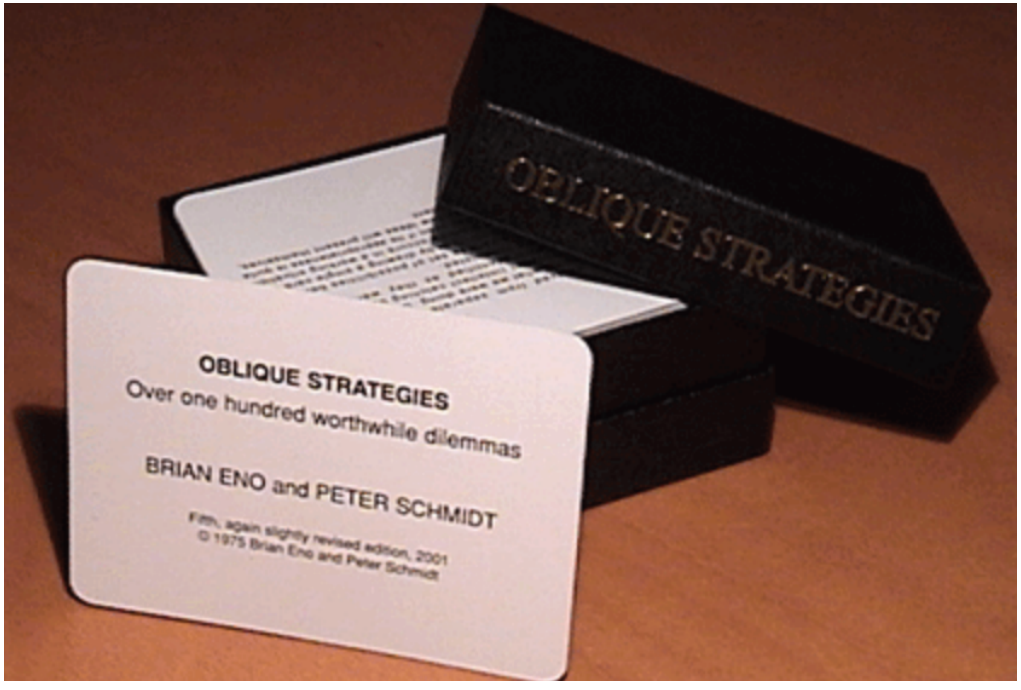
(d) Conversation and pairwise terms excluded

(e) All terms included

merrell + schkufza + li + agrawala + koltun

Oblique Strategies

- Brian Eno and Peter Schmidt
- Over 100 Aphorisms
- break creative blocks with lateral thinking



Example Oblique Strategies

- Use an old idea.
- State the problem in words as clearly as possible.
- Only one element of each kind.
- What would your closest friend do?
- What to increase? What to reduce?
- Are there sections? Consider transitions.
- Try faking it!
- Honour thy error as a hidden intention.
- Ask your body.
- Work at a different speed.

Next Time: Future Fabrication

- *Texturing and Modeling: A Procedural Approach* by Ebert + Musgrave + Peachey + Perlin + Worley
- *A Pattern Language* by Alexander
- *Design Patterns: Elements of Reusable Object-Oriented Software* by Gamma + Helm + Johnson + Vlissides
- *Algorithmic Beauty of Plants Book* by Prusinkiewicz
- *Digital Morphogenesis* by Neil Leach
- *Design Patterns in Dynamic Programming* by Peter Norvig