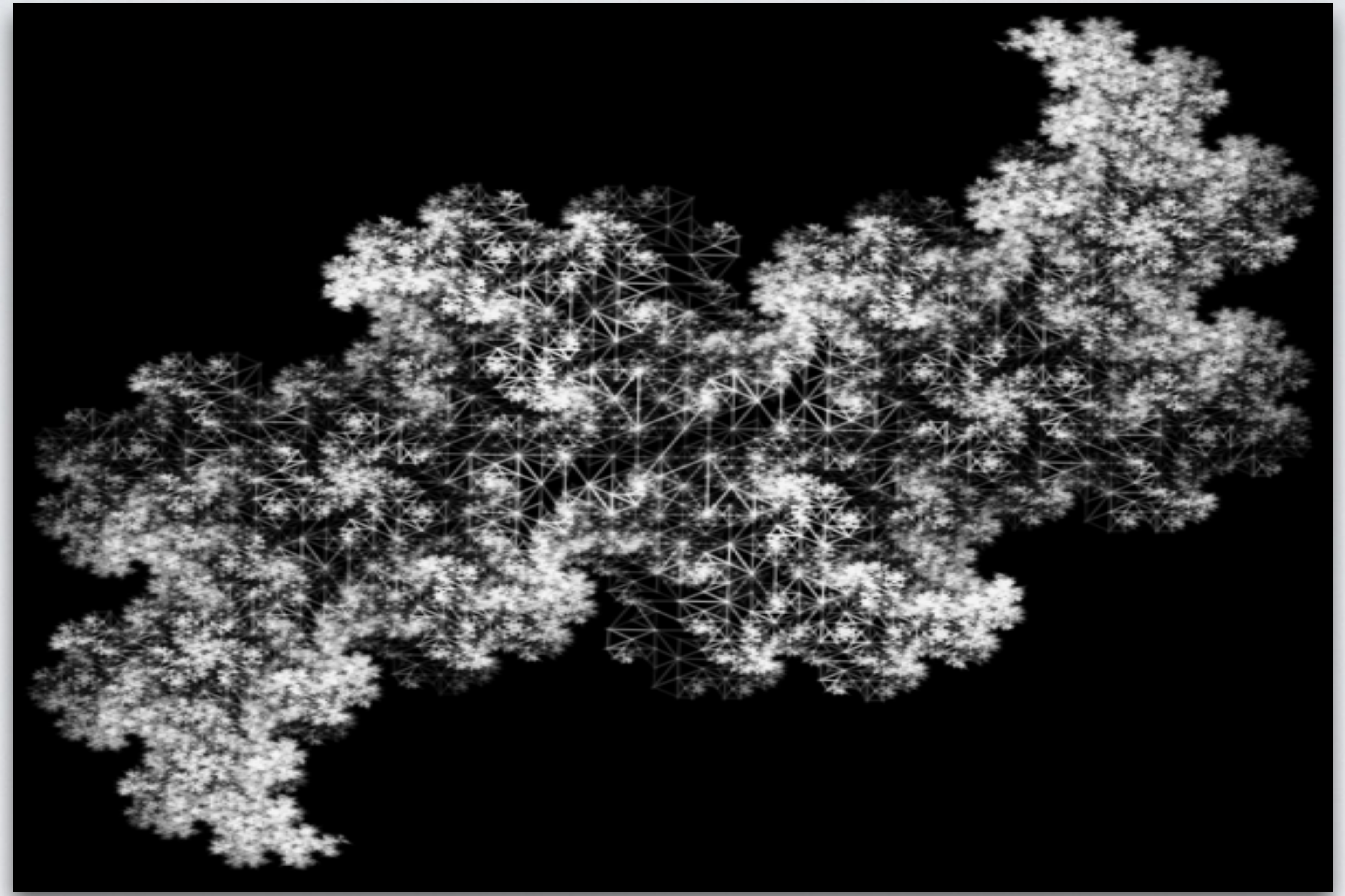
L-SYSTEMS

generating complex recursive systems



"erdos" (source)

University of Pennsylvania - CIS 700 Procedural Graphics

Rachel Hwang

DESCRIBING COMPLEXITY

drodd (source)

- Assertion: we can produce virtually anything procedurally.
- Growth functions and noise will get us far, but not enough to produce significant systemic/structural variation.
- Definitely possible! (demo from fabulous TA Austin Eng)
- So how do we describe complex systems?

BREAKING IT DOWN

2020site (source)

- Boils down to breaking complex systems into simple parts
- What are the basic components?
- In what context do they appear?
- Like programming! Or grammar!
- eg. this tree — essentially just a collection of needles and branches with regular patterning.
- Can generate fractal complexity!

BUT REALLY!

Jon Sullivan (source)

Romanesco "fractal" broccoli

FORMALIZING

L-SYSTEM GRAMMARS

String rewriting systems

variables : A B

constants : none

axiom : A

rules : (A → AB), (B → A)

which produces:

$n = 0$: A

$n = 1$: AB

$n = 2$: ABA

$n = 3$: ABAAB

$n = 4$: ABAABABA

$n = 5$: ABAABABAABAAB

- L-systems consist of:
    - An **alphabet of symbols**: our components (note that symbols can be abstract)
    - An **axiom**: initial configuration
    - A **grammar**: rules that determine what symbols appear in what contexts. Rules have
        - **Preconditions**
        - **Postconditions**

GRAPHICAL INTERPRETATION

We can interpret symbols as rendering instructions

- **F** = draw a line, moving forward
- **-** = rotate(30)
- **+** = rotate(-30)
- **[** = save position
- **]** = store position

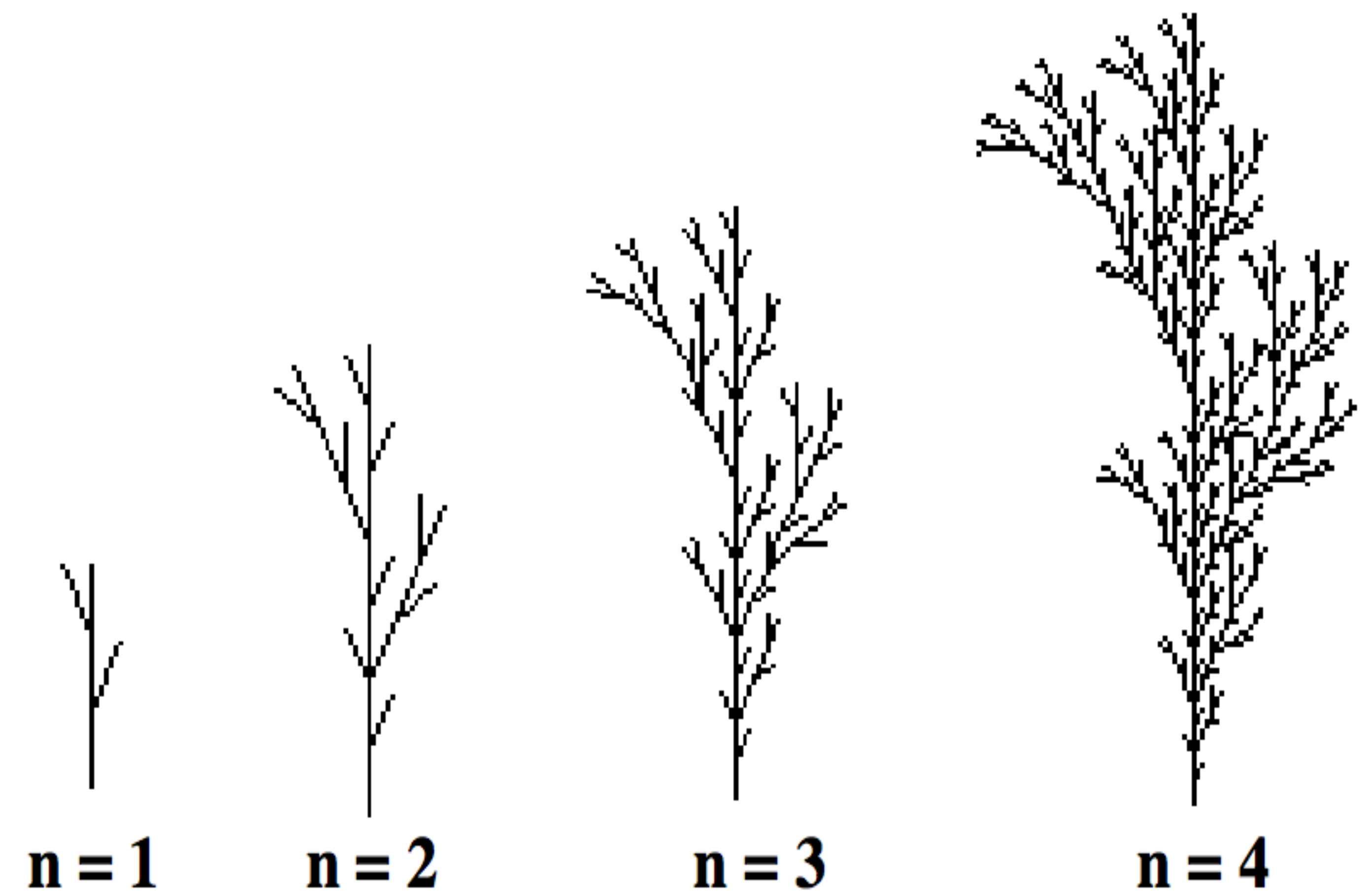
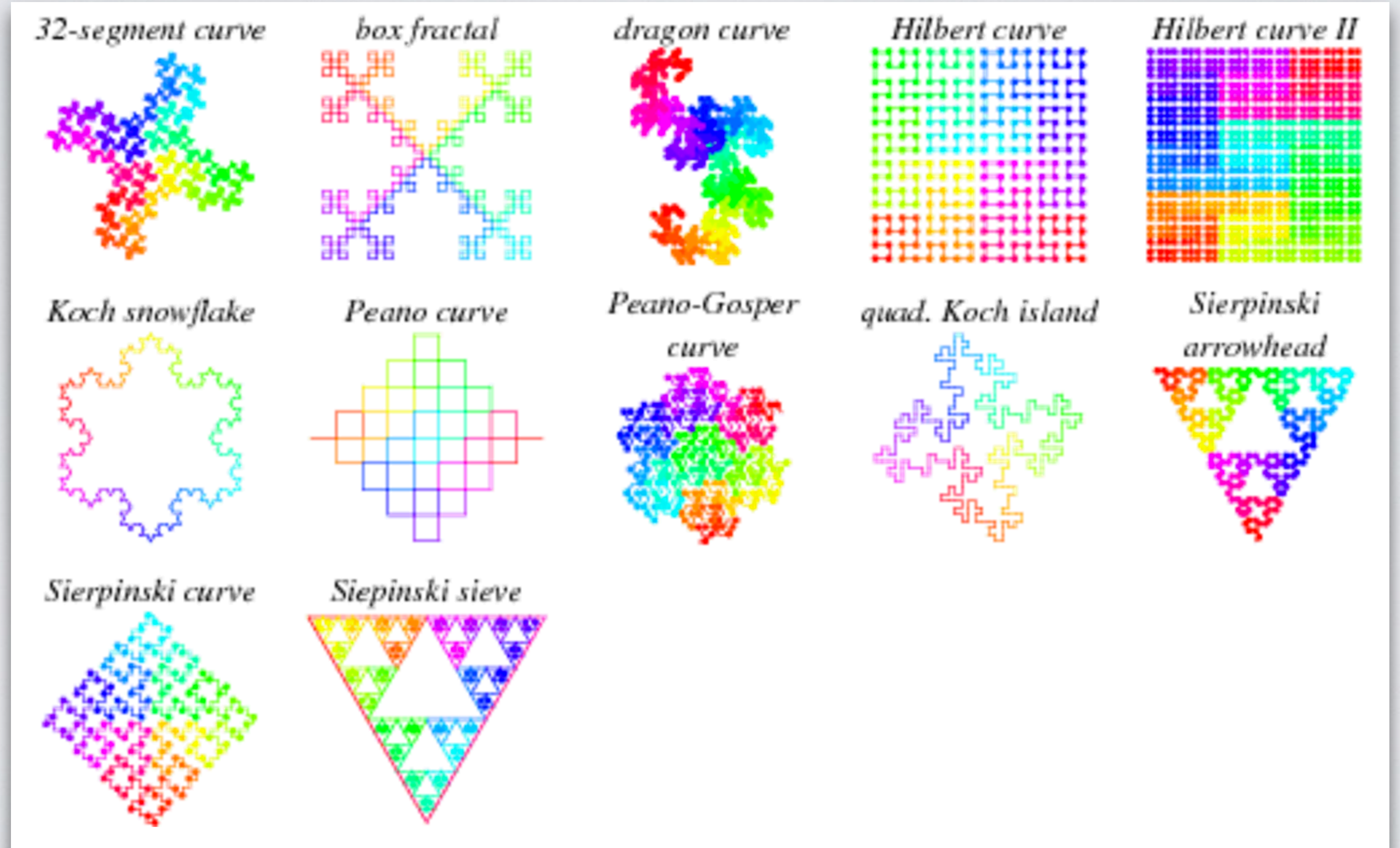


Fig. 5.3: Four rewrites of the bracketed L-system $F \rightarrow F[-F]F[+F][F]$

pcgbook (source)

GRAMMARS ARE POWERFUL!

Interactive demo here



Wolfram (source)

INJECTING VARIATION

So far, our systems have been deterministic, we can add variation in several ways:

1) Create multiple rules that apply in the same context with the same precondition

eg. {A → B, 50%}, {A → C, 50%}

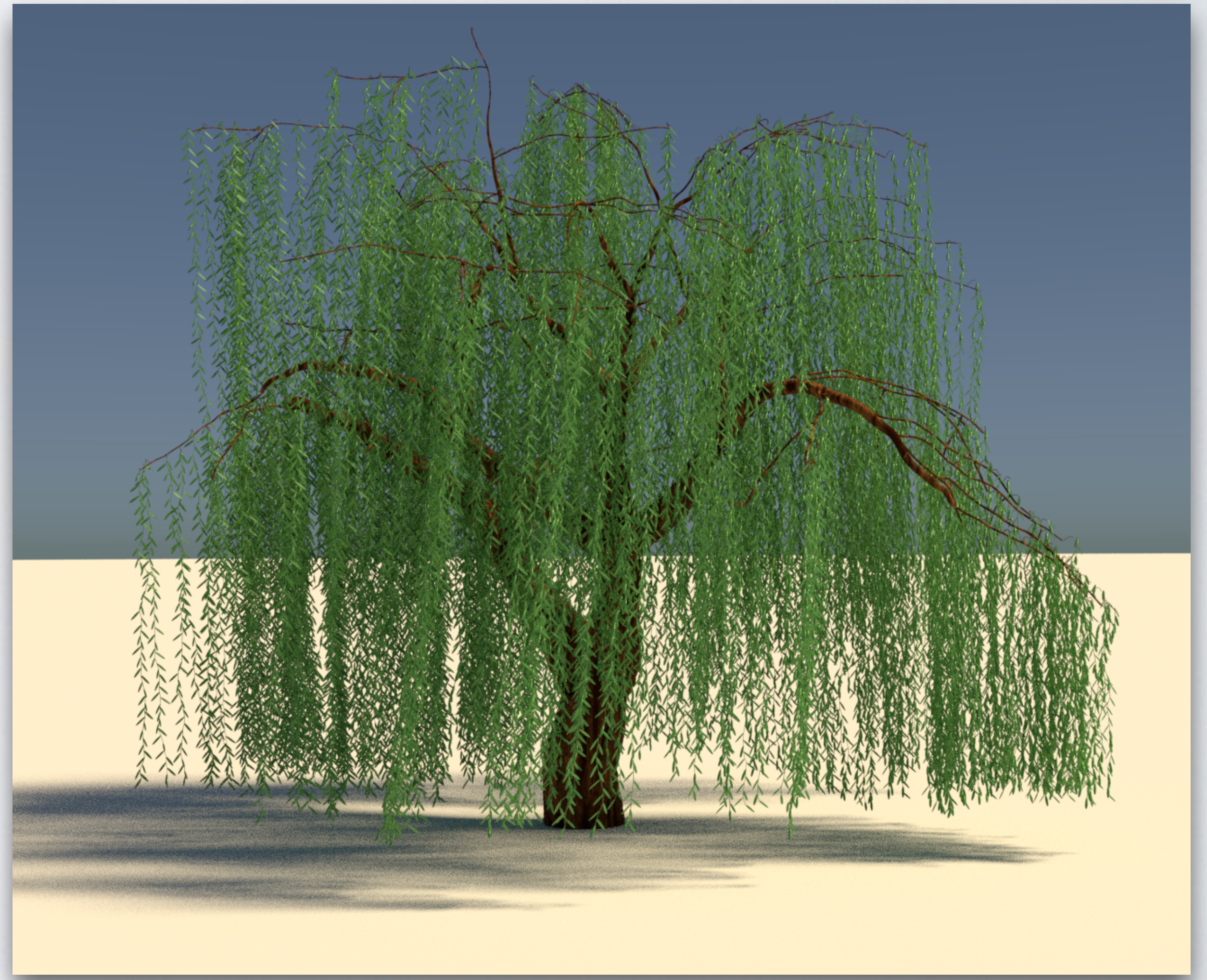
2) Have an element of randomness in rule interpretation

eg. A = rotateX(noise(n))

<u>For example</u>!

AN EXAMPLE

How do we generate a tree like this?



IMPLEMENTATION

Rough suggestion for the main parser function:

```
// Apply valid rules to each symbol in our list, for n iterations
SymbolList lsystemParse(int iterations, SymbolList axiom, RuleList grammar) {
  for (int i=0; i < iterations; ++i) {
    for (symbol old_sym : axiom) {
      // Find an applicable rule for old_sym (respect probabilty!)
      Symbol new_sym = applyRandomRule(old_sym, grammar);
      // Replace the old symbol in the axiom
      replace(old_sym, new_sym, axiom);
    }
  }
  return axiom;
}
```

IMPLEMENTATION

Rough suggestion for rule representation:

```
class Rule {
  symbol precondition;
  std::vector<Postcondition> postconditions;
};

class Postcondition {
  float probability;
  symbol new_symbol;
}
```

IMPLEMENTATION

Notes on the symbol representation:

- Symbols often represented as chars in a string
- However, constantly copying long strings per replacement, is inefficient
- We may also want to store additional information about a symbol, eg. the iteration it was added so we can apply a scale
- For all these reasons, linked lists are a nice solution.

IN SUMMARY

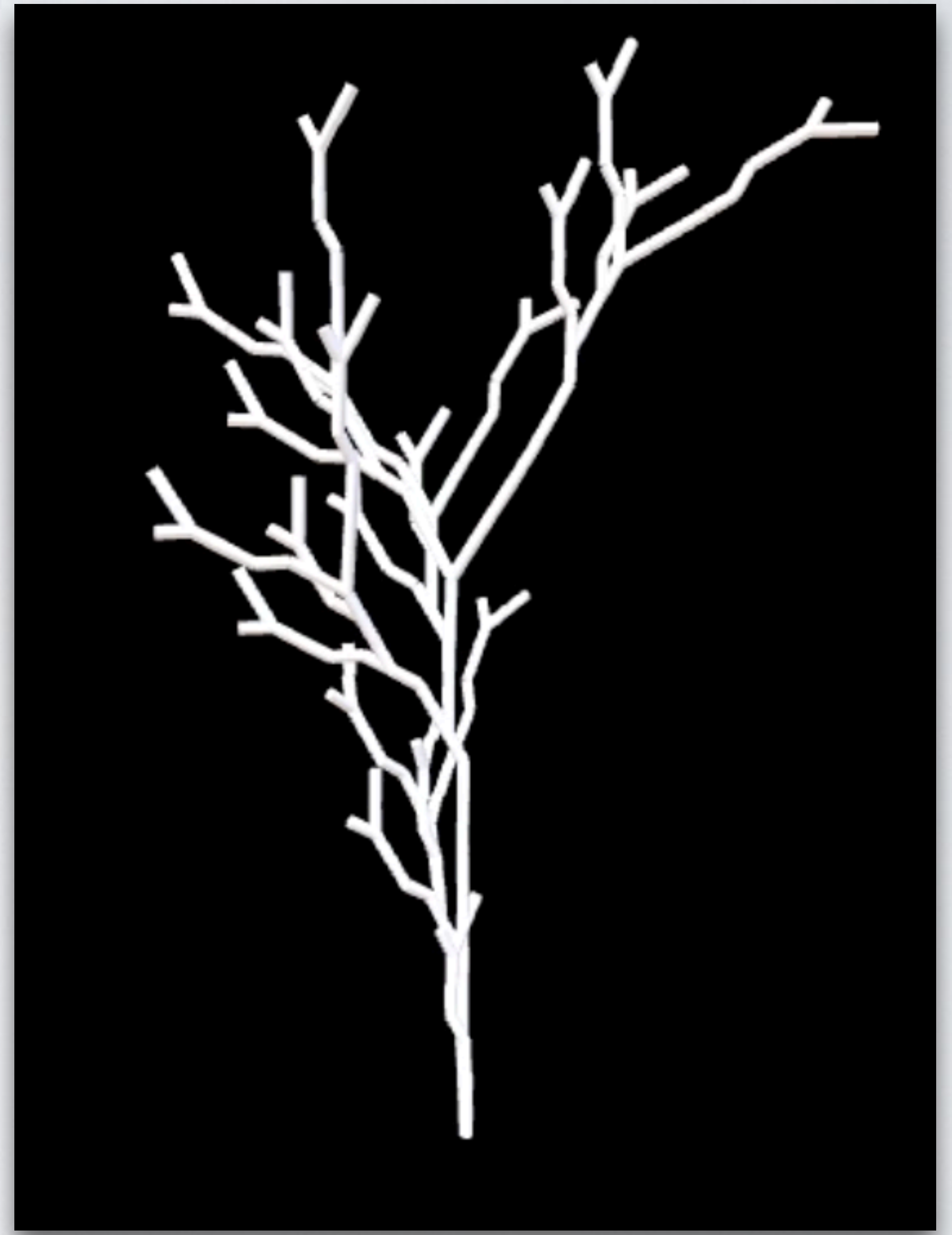
- Like programming, we can decompose complex systems into small logical units.
- We can formalize this as a simple grammar composed of rules for replacing symbols.
- We can add variation by giving rules probability, or adding random elements into our rendering rules
    - Symbols = {a, b, …. Z}
    - Rules of format = {A → B, 75%}
- Overall, lsystem assets are composed of two *separate* elements
    - The grammar
    - The rendering interpretation

REFERENCES

- Texts
  - Algorithmic Botany, textbook treatment of l-systems
  - Another text on grammars and lsystems
  - Houdini system reference
- Demos
  - Lsystem generator
  - Another Lsystem generator

ASSIGNMENT

- Create a linked-list structure to represent an alphabet symbol
- Create a rule format in which to encode grammar
- Create an lsystem parser to process symbols using a grammar
- Design an original grammar which generates plants with leaves or flowers of some kind.



Demo