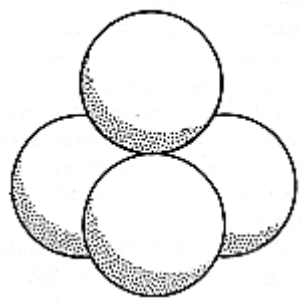
Pythonic Mathematics

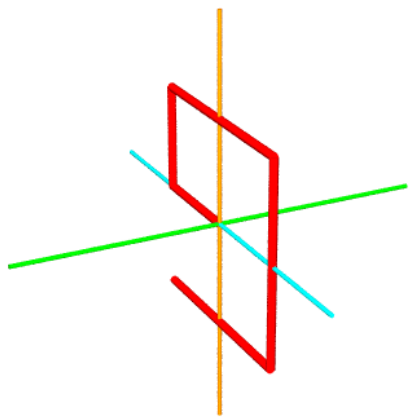
Kirby Urner

EuroPython 2005
Göteborg, Sweden

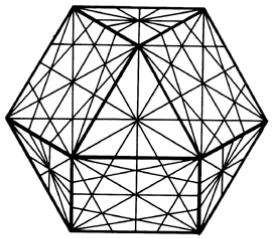


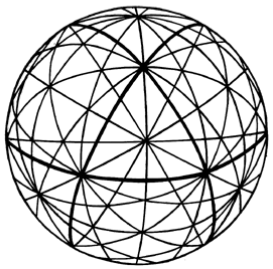
Principal Themes

- Math Through Programming
- Math Through Storytelling
- Beyond Flatland
- Curriculum as Network

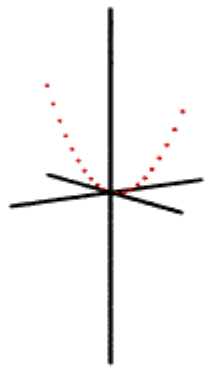


Math Through Programming



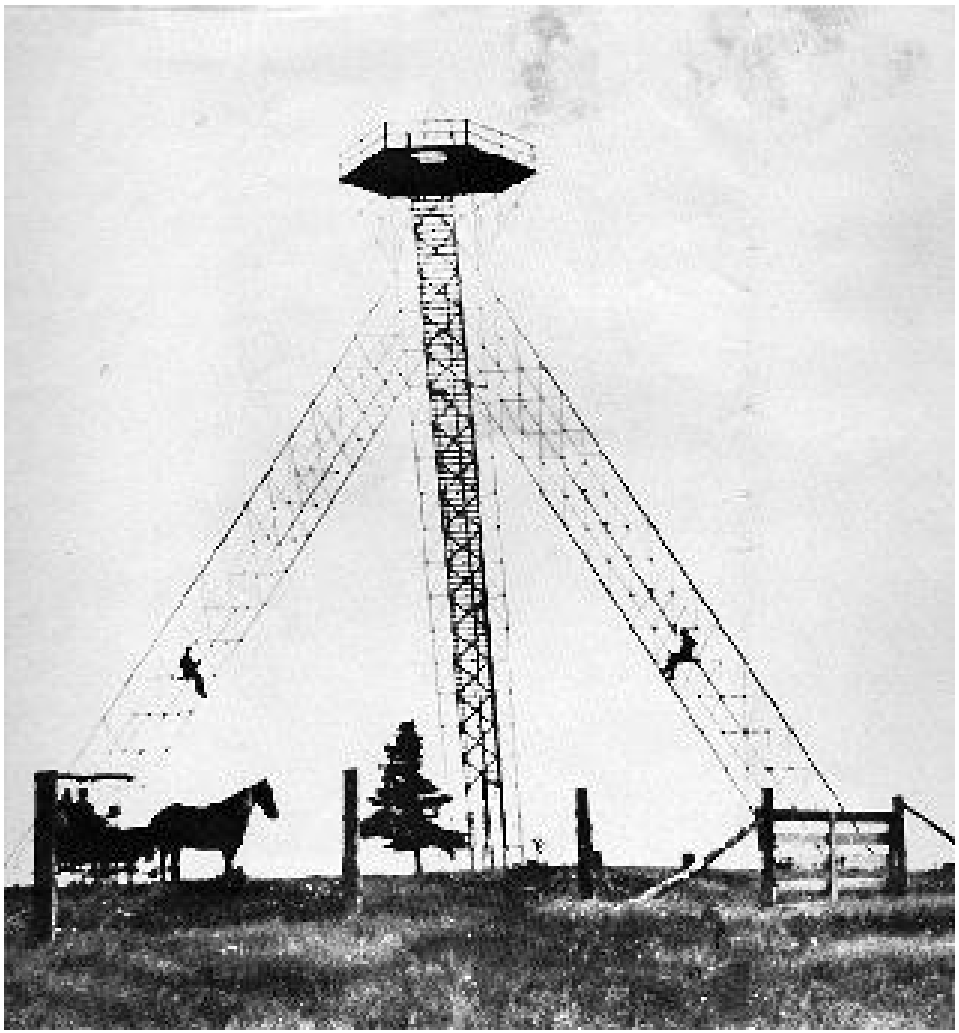


- Common goal: Literacy
- Levels of fluency
- Reading (Recognition)
- Writing (Recall)
  - Planning
  - Designing
- Testing and Correcting (Debugging)



Math Through Storytelling

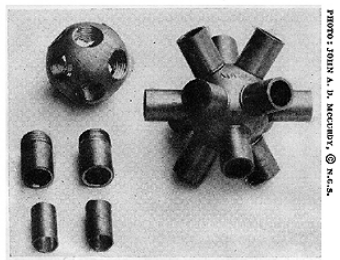
Beyond Flatland



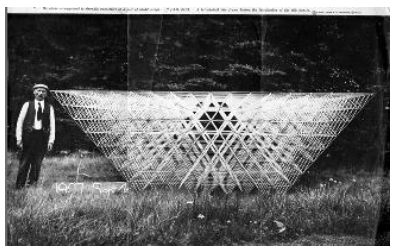
A. Graham Bell
Has New Idea
In Architecture

Opening of the Tetrahedral Tower, Seventy Feet High on Beinn Bhreagh.

It May Become an Important Factor in Building of the Future.

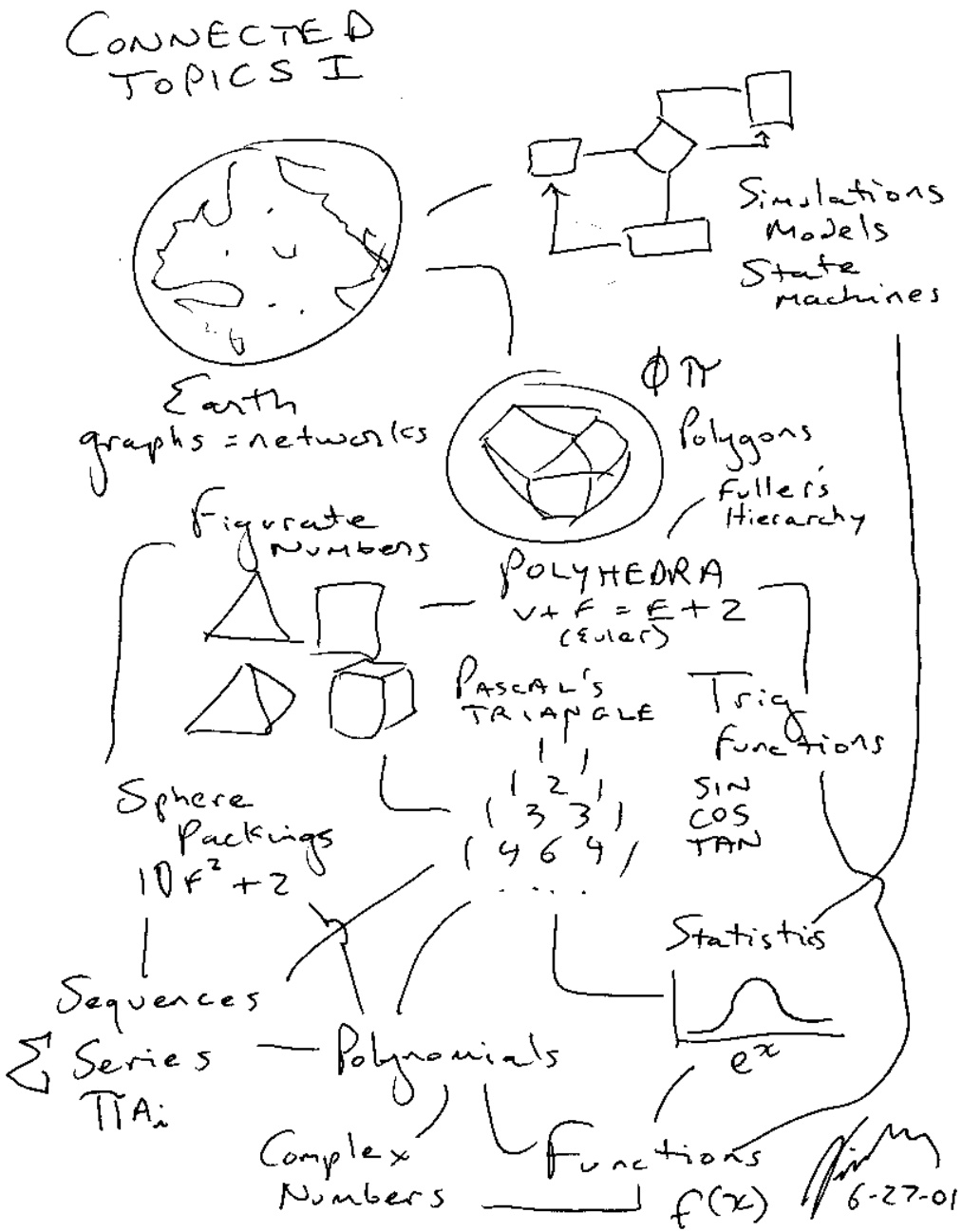


One of Dr. Bell's several connecting systems.



Curriculum as Network:

Typical Urner-style graph of inter-connecting topics



OO, Functions, More OO

We need a big picture of the OO design at first, to make sense of dot notation around core objects such as lists i.e. “these are types, and this is how we think of them, use them.” Use lots of analogies, metaphors.

Write your own classes a little later, after you’ve defined and saved functions in modules. Get more specific about syntax at this point.

ABOUT CLASSES

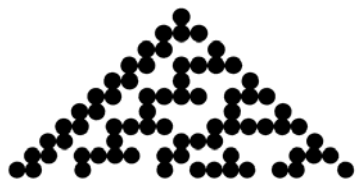
```
class Dog:

   def __init__(self, name):
      # constructor
      self.name = name

   def  bark(self, loud):
      if loud == 1: print 'woof'
      elif loud == 2: print 'Bark'
      elif loud == 3: print 'BARK!'
      else: print 'Huh?'

   def __repr__(self):
      # representation
      return 'Dog named %s' % self.name
```

A class consists of methods and other attributes. An object is a specific instance of a class. You may create many instances of the same class, and yet each is an individual place in memory with its own information. Every instance has a *self*.

From a class for home schoolers, taught by me @ Free Geek in PDX
See: http://www.4dsolutions.net/ocn/pygeom.html

OO & Data Structures

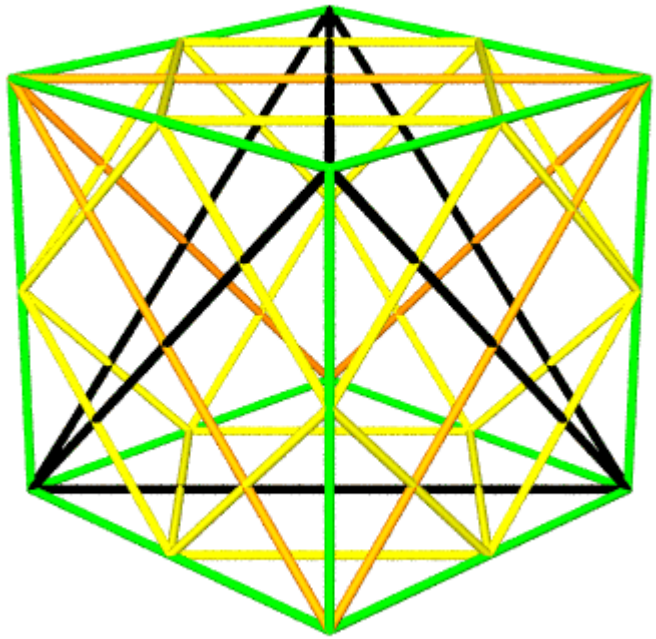
- Aristotle and the Kingdoms (family trees)
  - The Tree (big in classical thinking)
- Sets (big in mid 20th century “new math”)
  - Data Structures: list, array, vector, matrix, table, n-tuple, record, dictionary, hash, stack, queue, network, tree, linked list, graph…
- Python: a good language for show and tell plus plays well with others



New Hybrid: CS + Mathematics

- Mathematics as extensible type system
- A focus on algorithms that use objects
- Mix traditional notation w/ OO's dot notation
- Math Objects: polynomials, integers, integers modulo N, permutations, vectors, matrices, quaternions, polyhedra …
- Python's operator overloading: a gateway to new levels of abstraction

Python plays well with others



The prospect of getting to do computer graphics is a motivational incentive to tackle “the hard stuff.”

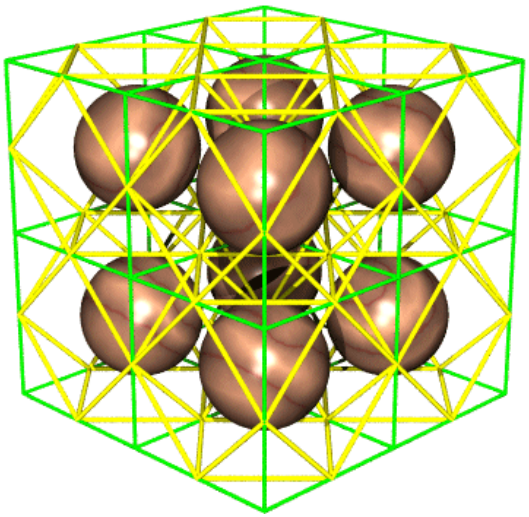
The promise: we’ll take you somewhere fun.

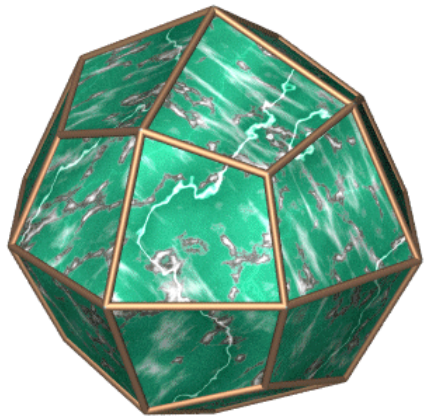
Types of Graphics with sample apps & libraries

	Still	**Moving**
Flat	PIL	Tk wxWidgets PyGame
Spatial	POV-Ray	POV-Ray VPython PyOpenGL

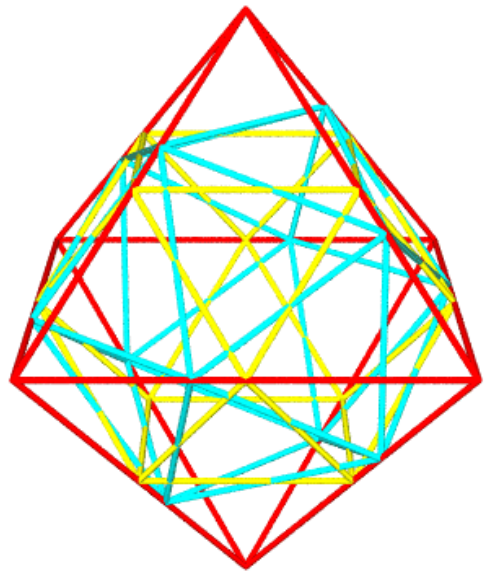
From presentation given at OSCON in 2004. See: http://www.4dsolutions.net/oscon2004/

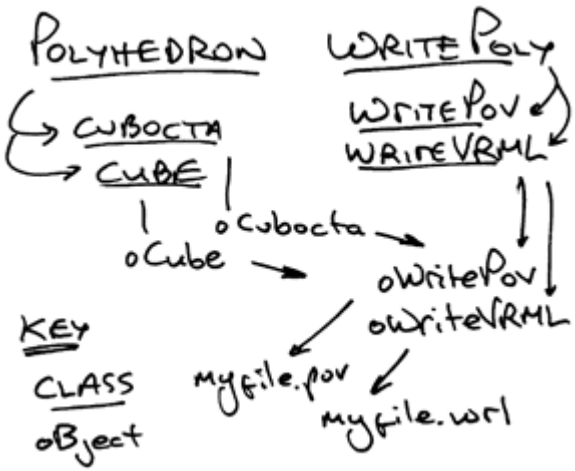
Python + POV-Ray



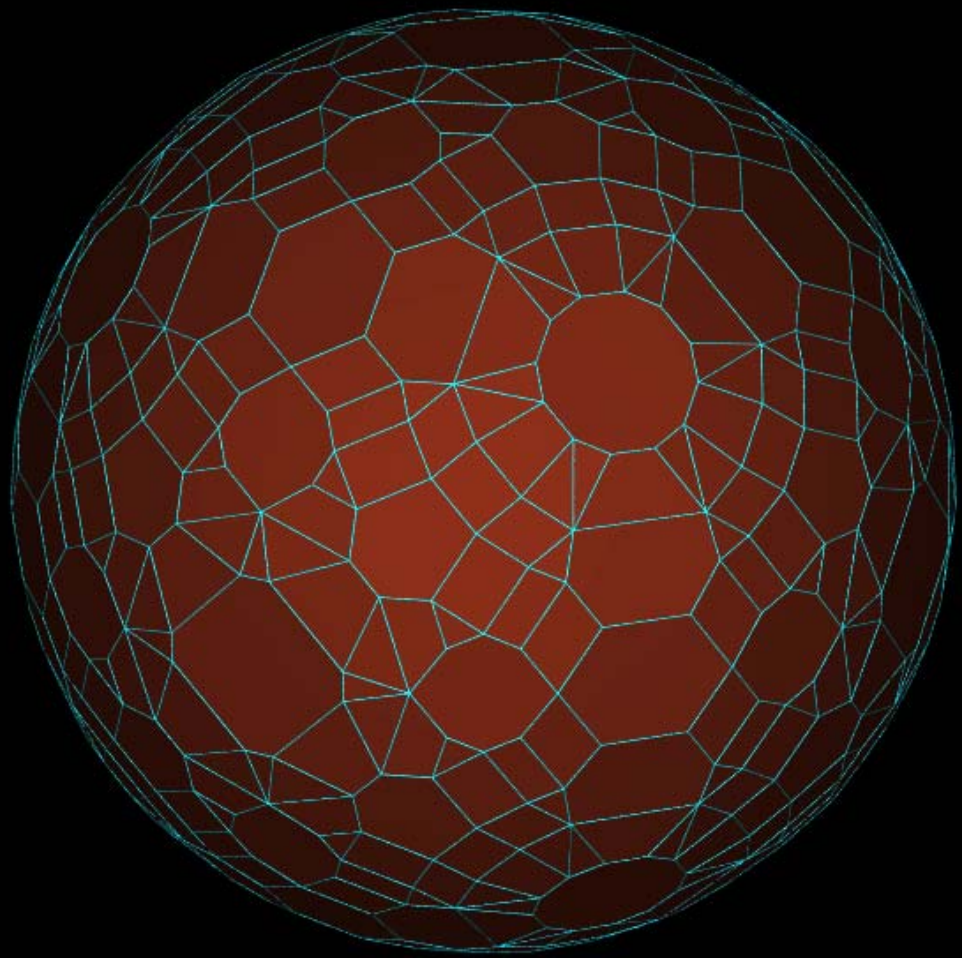


Trapezoidal Icositetrahedron

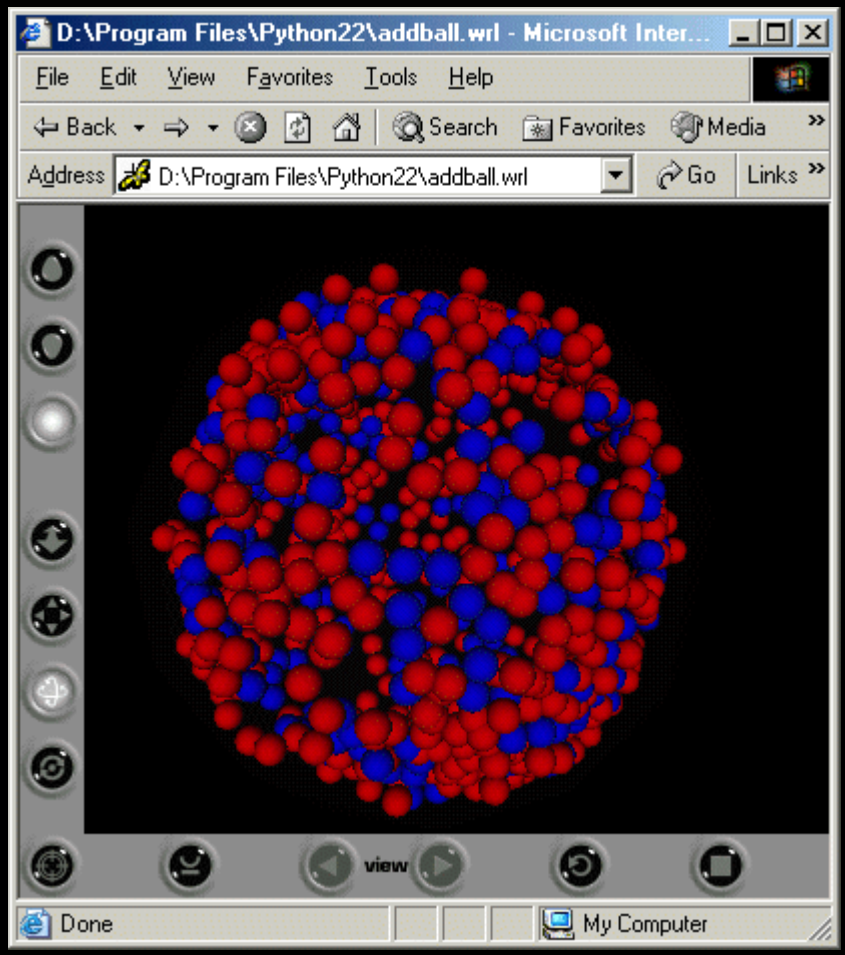


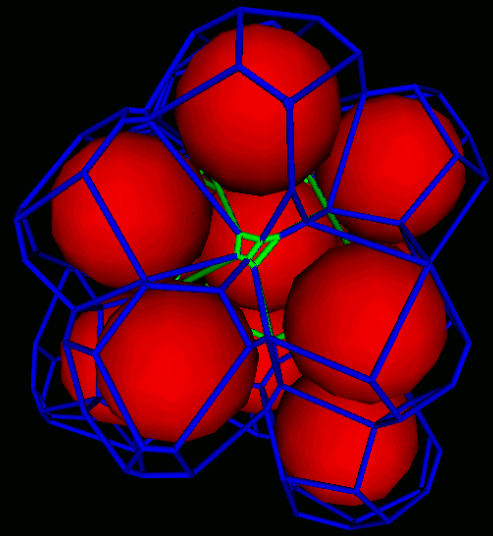


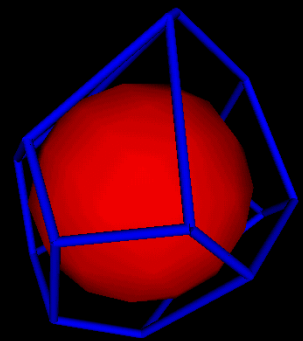
Jython + POV-Ray + QuickHull3D



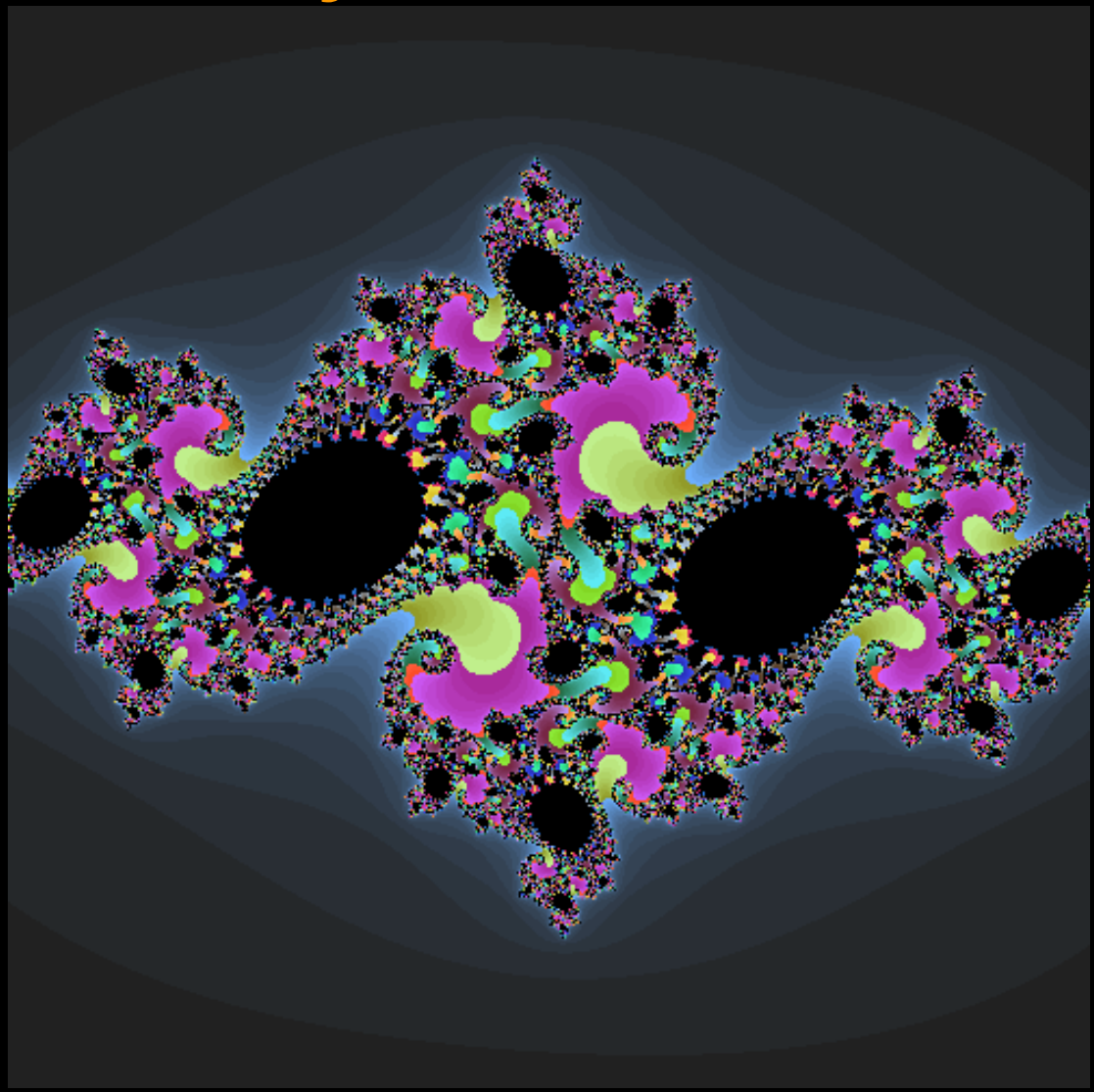
Python + VRML + Qhull





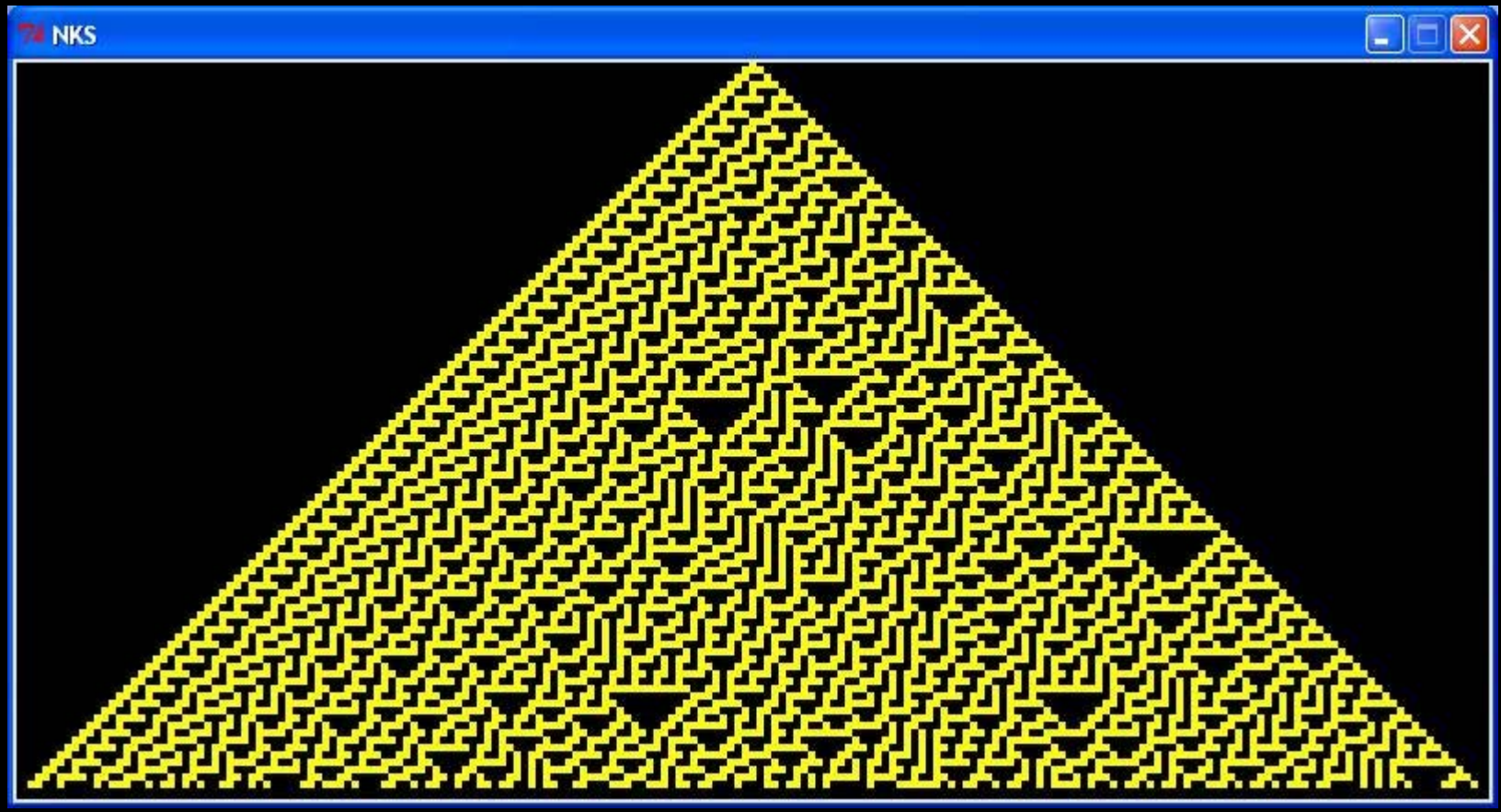


Python + PIL



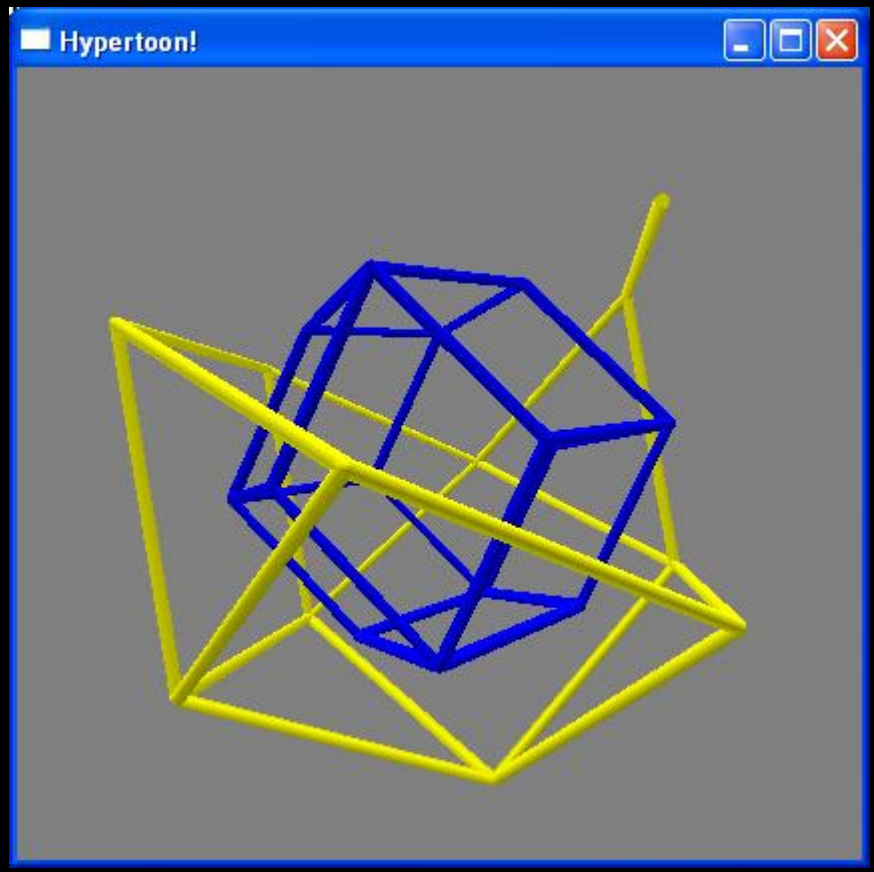
http://www.4dsolutions.net/ocn/fractals.html

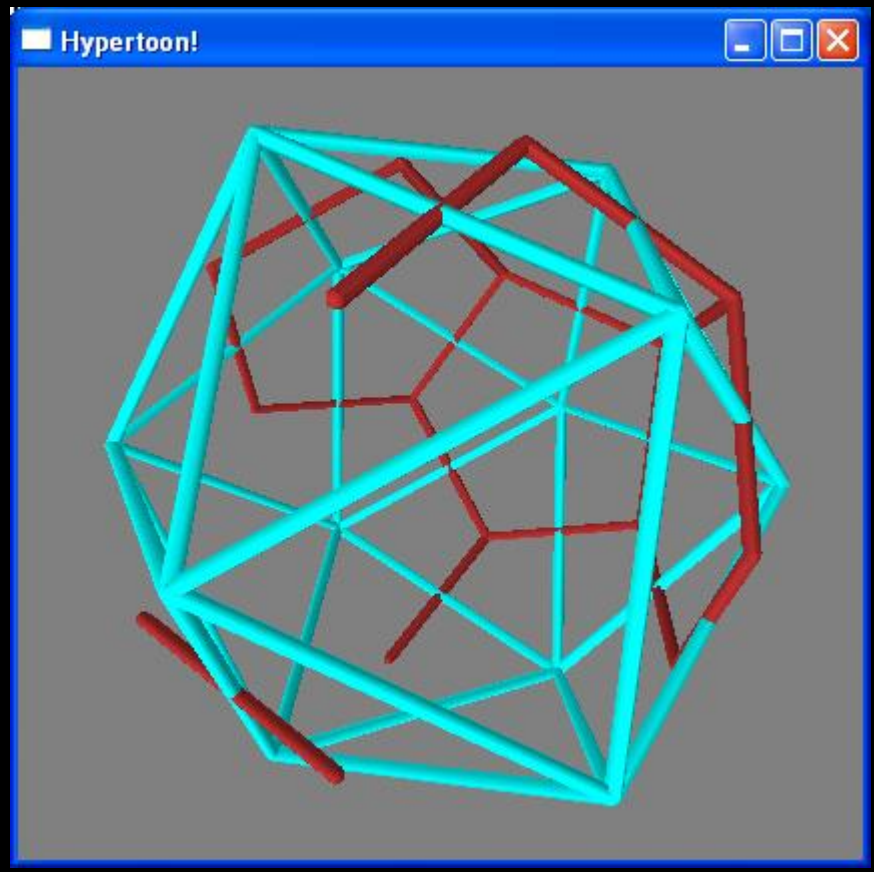
Python + Tk



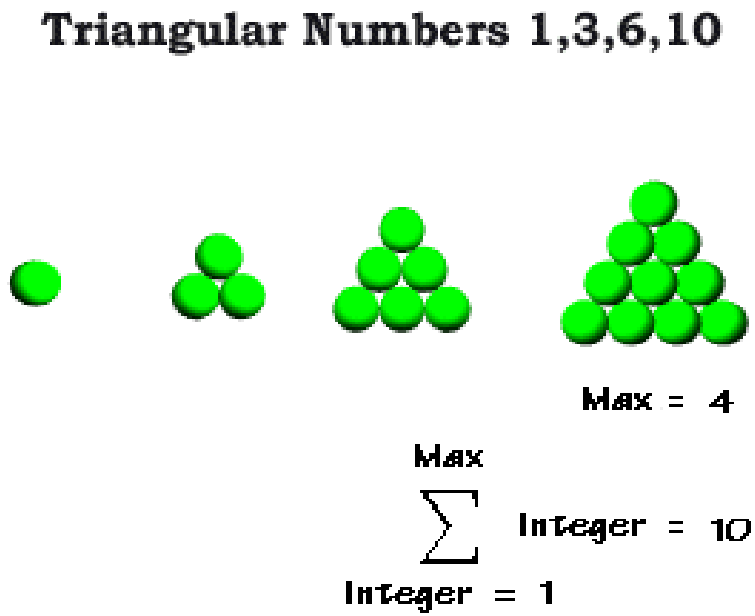
Cellular Automata ala Wolfram generated using graphics.py by John Zelle

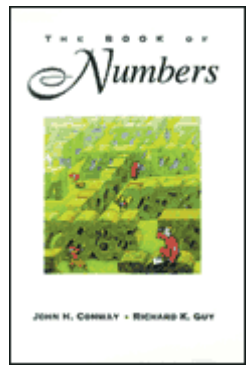
Python + VPython





Functions and Figurate Numbers



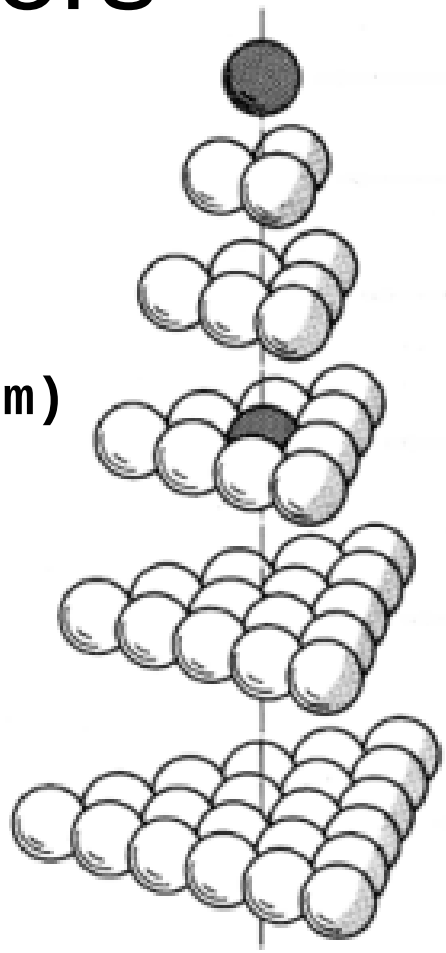


Front cover:
The Book of Numbers
by Conway & Guy

```
def tri(n):
    return n * (n + 1) // 2

>>> [tri(x) for x in range(1, 10)]
[1, 3, 6, 10, 15, 21, 28, 36, 45]
```

Sequence Generators

```
>>> def tritet():
        term = trinum = tetranum = 1
        while True:
            yield (term, trinum, tetranum)
            term += 1
            trinum += term
            tetranum += trinum


>>> gen = tritet()
>>> [gen.next() for i in range(6)]
[(1, 1, 1), (2, 3, 4), (3, 6, 10),
(4, 10, 20), (5, 15, 35), (6, 21, 56)]
```



from *Synergetics*
by RBF w/ EJA

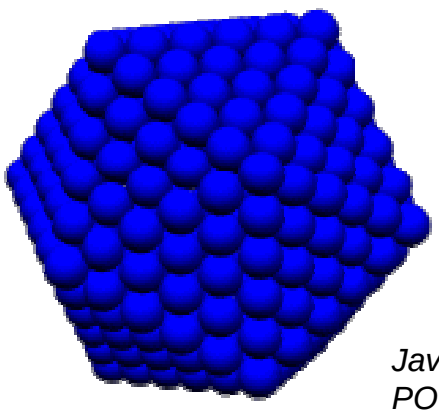
Polyhedral Numbers

Animation: growing cuboctahedron

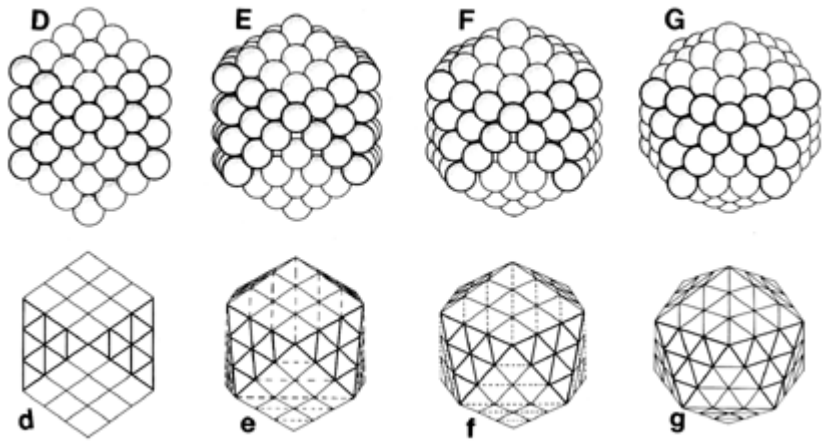
```
>>> gen = cubocta()
>>> [gen.next() for i in range(6)]
[(1, 1, 1), (2, 12, 13), (3, 42, 55),
(4, 92, 147), (5, 162, 309), (6, 252, 561)]
```

Cubocta Shell = Icosa Shell

```
def cubocta():
    freq = shell = cubonum = 1
    while True:
        yield (freq, shell, cubonum)
        shell = 10 * freq**2 + 2
        cubonum += shell
        freq += 1
```



Java + POV-Ray



From *Synergetics*

Pascal's Triangle

```
def pascal():
      row = [1]
      while True:
          yield row
          this = [0] + row
          next = row + [0]
          row = [a+b for a,b in zip(this, next)]
```

```
>>> gen = pascal()
>>> gen.next()
[1]
>>> gen.next()
[1, 1]
>>> gen.next()
[1, 2, 1]
```

Fermat's Little Theorem

```
def gcd(a,b):
    """Euclidean Algorithm"""
    while b:
        a, b = b, a % b
    return abs(a)


if isprime(p) and gcd(b,p) == 1:
    try:
        assert pow(b, p - 1, p) == 1
    except:
        raise \
        Exception, 'Houston, we've got a problem.'
```

Euler’s Theorem

```
def tots(n):
    return [i for i in range(1,n)
            if gcd(i, n)==1]


def phi(n): return len(tots(n))


if gcd(b,n) == 1:
    try:
        assert pow(b, phi(n), n) == 1
    except:
        raise \
        Exception, 'Houston, we’ve got a problem.'
```

RSA

```
def demo():
    """ Abbreviated from more complete version at:
    http://www.4dsolutions.net/satacad/sa6299/rsa.py"""
    plaintext = "hello world"
    m = txt2n(plaintext)
    p,q = getf(20), getf(20) # two big primes
    N = p*q
    phiN = (p-1)*(q-1)
    e = 3
    s,t,g = eea(e, phiN) # Extended Euclidean Algorithm
    d = s % phiN
    c = encrypt(m,N) # pow(m, e, N) w/ booster
    newm = decrypt(c,d,N) # pow(c, e*d, N)
    plaintext = n2txt(newm)
    return plaintext
```

Math Objects:

grab from the library
and/or build your own

```
>>> mypoly = Poly([ (7,0), (2,1), (3,3), (-4,10) ] )
>>> mypoly
(7) + (2*x) + (3*x**3) + (-4*x**10)

>>> int1, int2 = M(3, 12), M(5, 12) # used in crypto
>>> int1 * int2  # operator overloading
3
>>> int1 – int2
10

>>> tetra = rbf.Tetra()
>>> bigtetra = tetra * 3  # volume increases 27-fold
>>> bigtetra.render()
```

Example: Build A Rational Number Type

From http://www.4dsolutions.net/ocn/python/mathteach.py

```
def cf2(terms):
    """
    Recursive approach to continued fractions, returns Rat object

    >>> cf2([1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1])
    (4181/2584)

    >>> cf2([1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2])
    (665857/470832)

    >>> (665857./470832)
    1.414213562373899
    """
    if len(terms)==1:
        return terms.pop()
    else:
        return Rat(terms.pop(0),1) + Rat(1, cf2(terms))
```

Conclusions

- “Programming to learn” is as valid an activity as “learning to program.” My proposal is about programming in Python in order to build a stronger understanding of mathematics.
- Mastering specialized “learning languages” or even “math languages” doesn’t offer the same payoff as mastering a full-featured generic computer language.
- This approach and/or curriculum is not for everyone.
- A wide variety of approaches exist even *within* the broad brush strokes vision sketched out here.

Thank You!

And now…

HyperToon demo + Q&A

Presentations repository:
http://www.4dsolutions.net/presentations/