Complexity

Big-*O*

Definition

- f is $O(g)$ means there are numbers k and C such that

  -- $f(n) \leq C\,g(n)$ for all $n > k$.

Meaning

- For *all* sufficiently large integers, $f(n)$ is less than a *constant* multiple of $g(n)$.
- $O(g)$ is only an Upper Bound on the size of f.

Observations

- k and C are not unique.
- g is not unique.
- g is usually chosen to be well-known function: $\log(n)$, n^p, 2^n.

Big-*O* Notation

Notation (Abuse of Notation)

- $f = O(g)$

Meaning

- $f \in O(g)$
- $O(g) = \{h \mid h(n) \leq C_h\, g(n) \text{ for all } n \geq k_h \text{ for some constant } C_h\}$

Diagrams

Examples

1. $c_n x^n + \cdots + c_1 x + c_0 = O(x^n)$

2. $$\frac{c_n x^n + \cdots + c_1 x + c_0}{d_n x^m + \cdots + d_1 x + d_0} = O(x^{n-m})$$

3. $$\sum_{k=1}^{n} k^p = O(n^{p+1})$$

4. $$\sum_{p=1}^{n} 2^p = O(2^n)$$

5. $n! = O(n^n)$

Most Common Orders

- $1, \log(n), n, n\log(n), n^2, n^p, 2^p, n!, n^n$

- Constant, Logarithmic, Linear, Polynomial, Exponential

Properties

0. $f = O(f)$

1. $f = O(g)$ and $g = O(h) \Rightarrow f = O(h)$

2. $f = O(g) \Rightarrow f + c = O(g)$

3. $f = O(g) \Rightarrow c\,f = O(g)$

4. $f_1 = O(g_1)$ and $f_2 = O(g_2) \Rightarrow f_1 + f_2 = O\big(\max(g_1, g_2)\big)$

5. $f_1 = O(g_1)$ and $f_2 = O(g_2) \Rightarrow f_1\,f_2 = O(g_1\,g_2)$

Properties

0. $f = O(f)$

1. $f = O(g)$ and $g = O(h) \Rightarrow f = O(h)$

2. $f = O(g) \Rightarrow f + c = O(g)$ FALSE (Try $f(n) = \frac{1}{n^2}$.)

3. $f = O(g) \Rightarrow c\,f = O(g)$

4. $f_1 = O(g_1)$ and $f_2 = O(g_2) \Rightarrow f_1 + f_2 = O\big(\max(g_1, g_2)\big)$

5. $f_1 = O(g_1)$ and $f_2 = O(g_2) \Rightarrow f_1\,f_2 = O(g_1\,g_2)$

Extra Credit

Prove or give a counterexample:

For all *positive* functions f and g, either $f = O(g)$ or $g = O(f)$.

Polynomial Algorithms

Polynomial Evaluation

- Horner's Method (Time)
- $O(n)$ vs. $O(n^2)$

Polynomial Interpolation

- Neville's Algorithm (Time and Space)
- $O(n^2)$ vs. $O(2^n)$

Polynomial Multiplication

- Fast Fourier Transform (FFT)
- $O\left(n\log(n)\right)$ vs. $O(n^2)$

Horner's Method

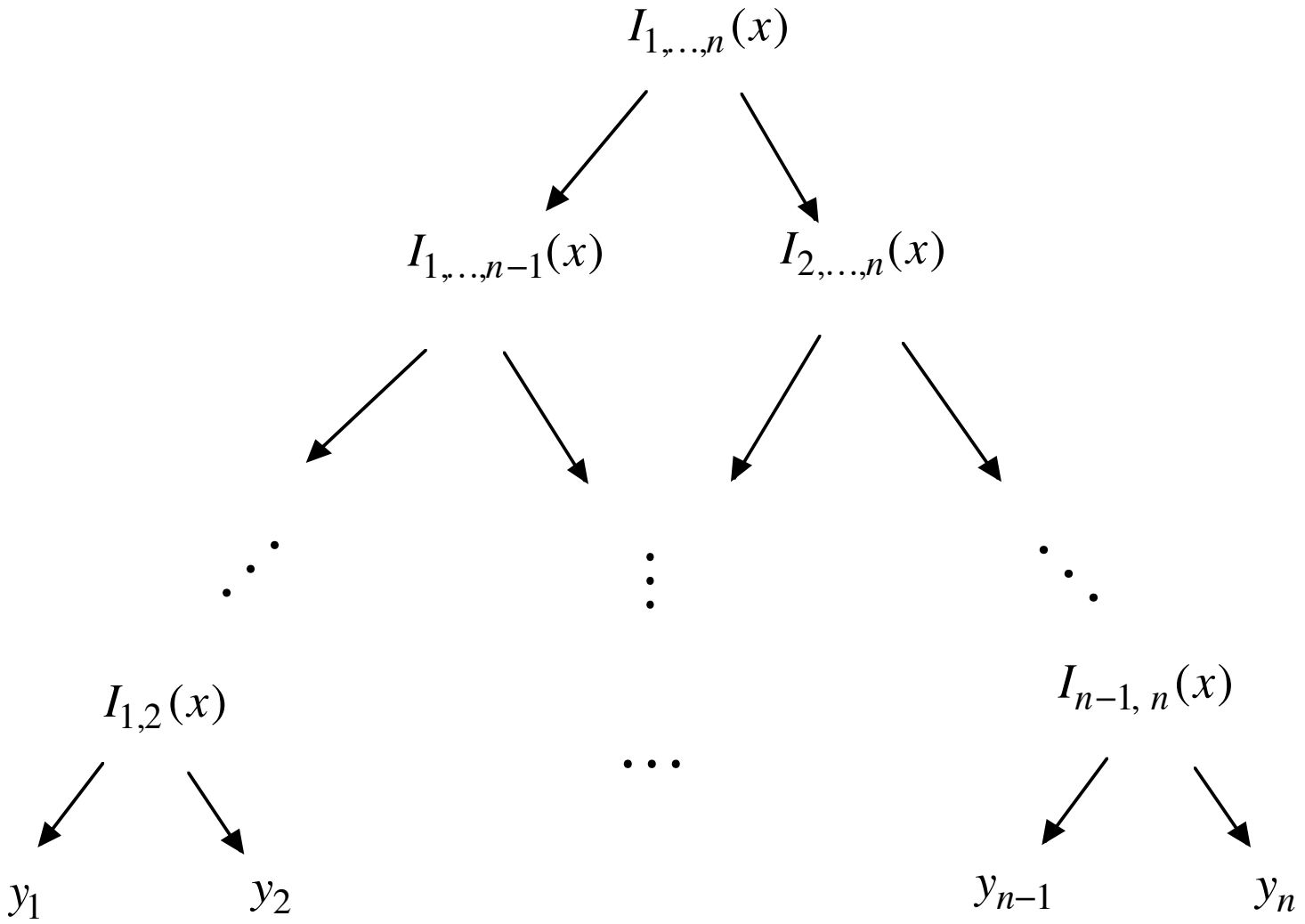
Example

- $7x^3 - 3x^2 + 11x - 5$ -- 6 multiplies
- $\big((7x-3)x+11\big)x-5$ -- 3 multiplies

General Case

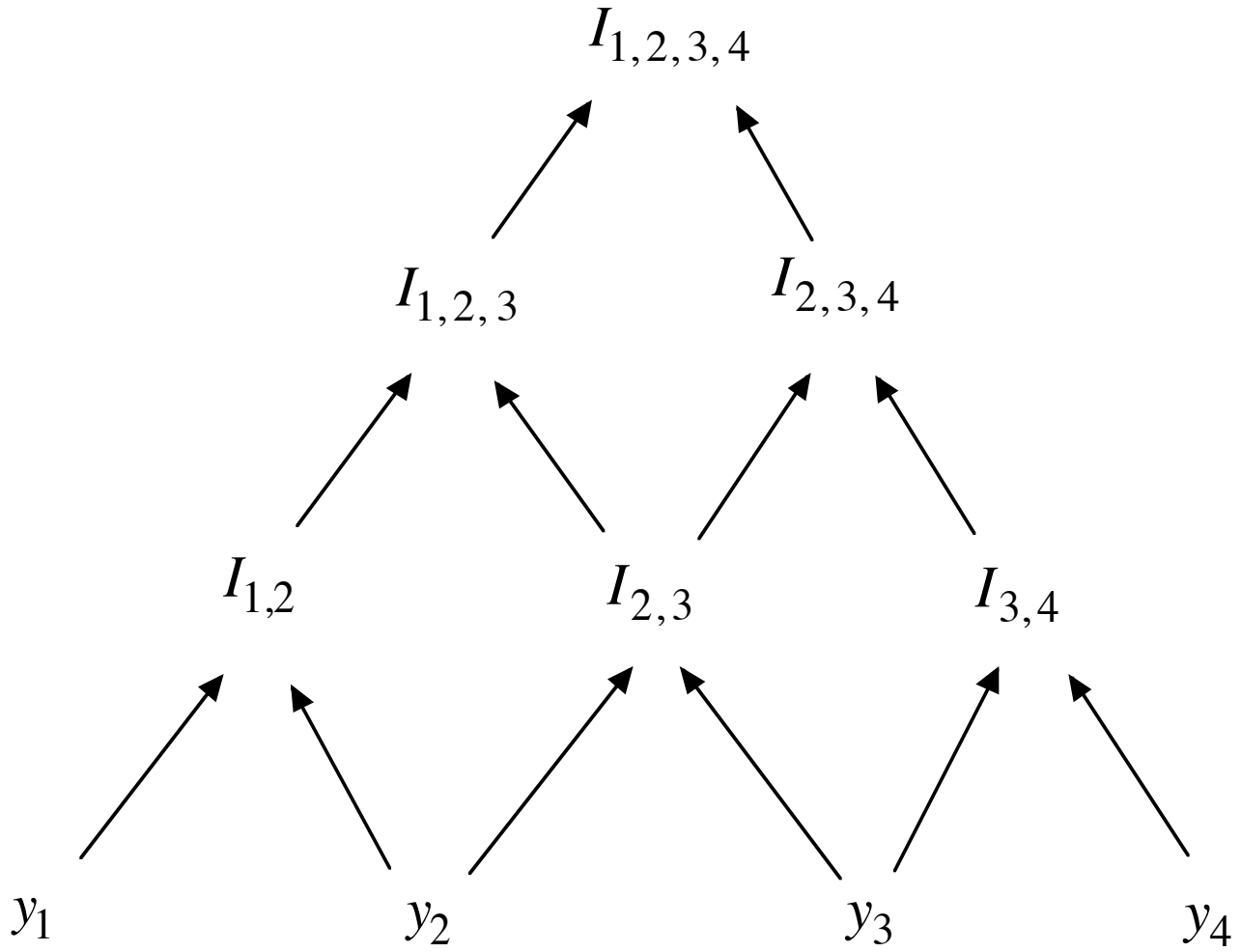
- $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ -- $\displaystyle\sum_{k=0}^{n} k = \frac{n(n+1)}{2} = O(n^2)$ multiplies
- $\big((a_n x + a_{n-1})x + \cdots + a_1\big)x + a_0$ -- $n = O(n)$ multiplies

Neville's Algorithm (Polynomial Interpolation) -- Recursive Implementation



$n-1$ *Levels* $\leftrightarrow$ $2^n - 2 = O(2^n)$ *multiplications*

Neville's Algorithm (Polynomial Interpolation) -- Iterative Implementation



$$n-1 \text{ Levels} \leftrightarrow 2\sum_{k=1}^{n-1} k = (n-1)n = O(n^2) \text{ multiplications}$$

P* and *NP

Definitions

P = problems that can be SOLVED in polynomial time

NP = problems that can be VERIFIED in polynomial time

Question

P = NP ?

Answer

Nobody knows!

NP

Examples (Later)

- SAT
- Hamiltonian Circuits

More Examples

- Factoring Integers
- Solving Polynomial Equations

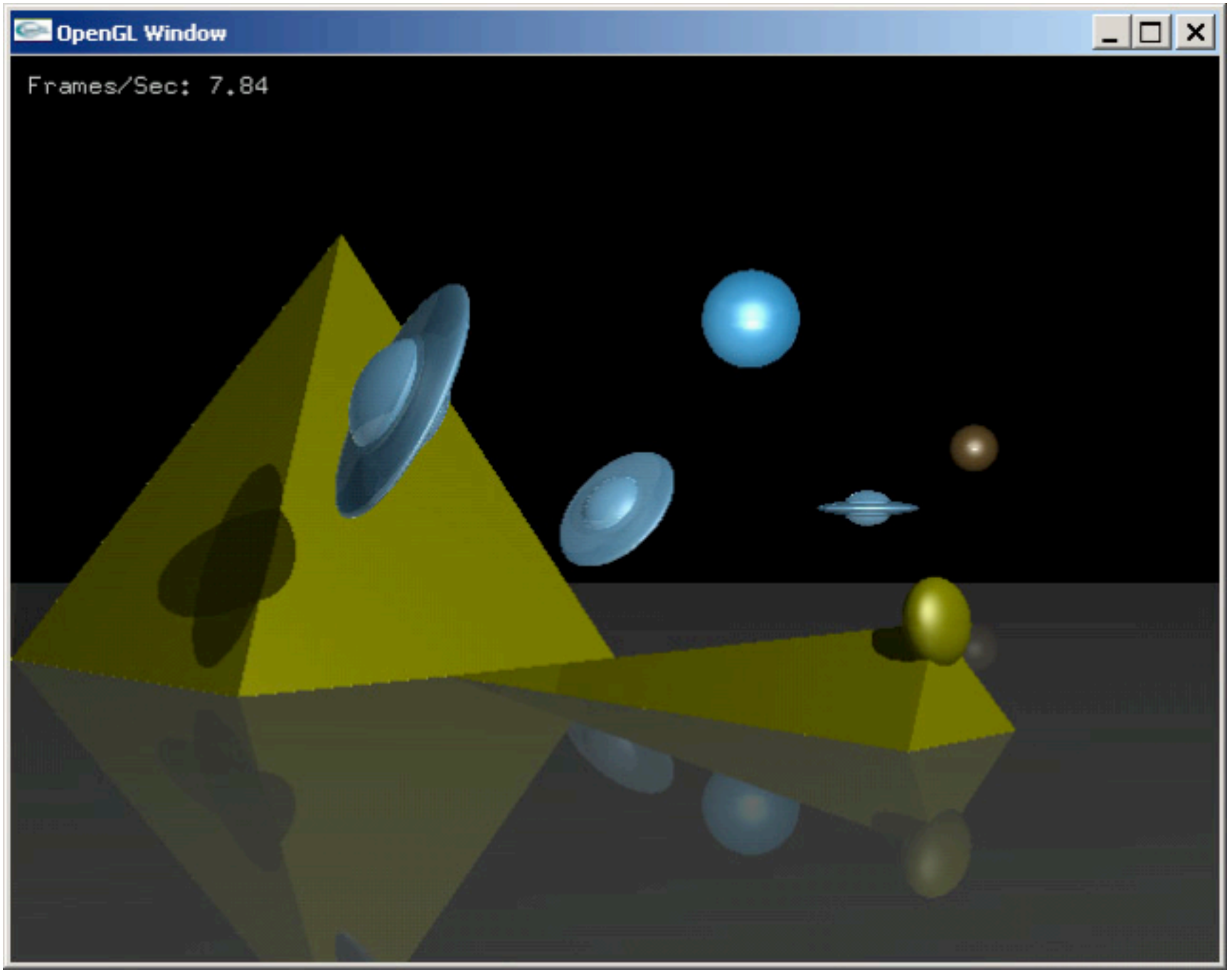
Searching Algorithms

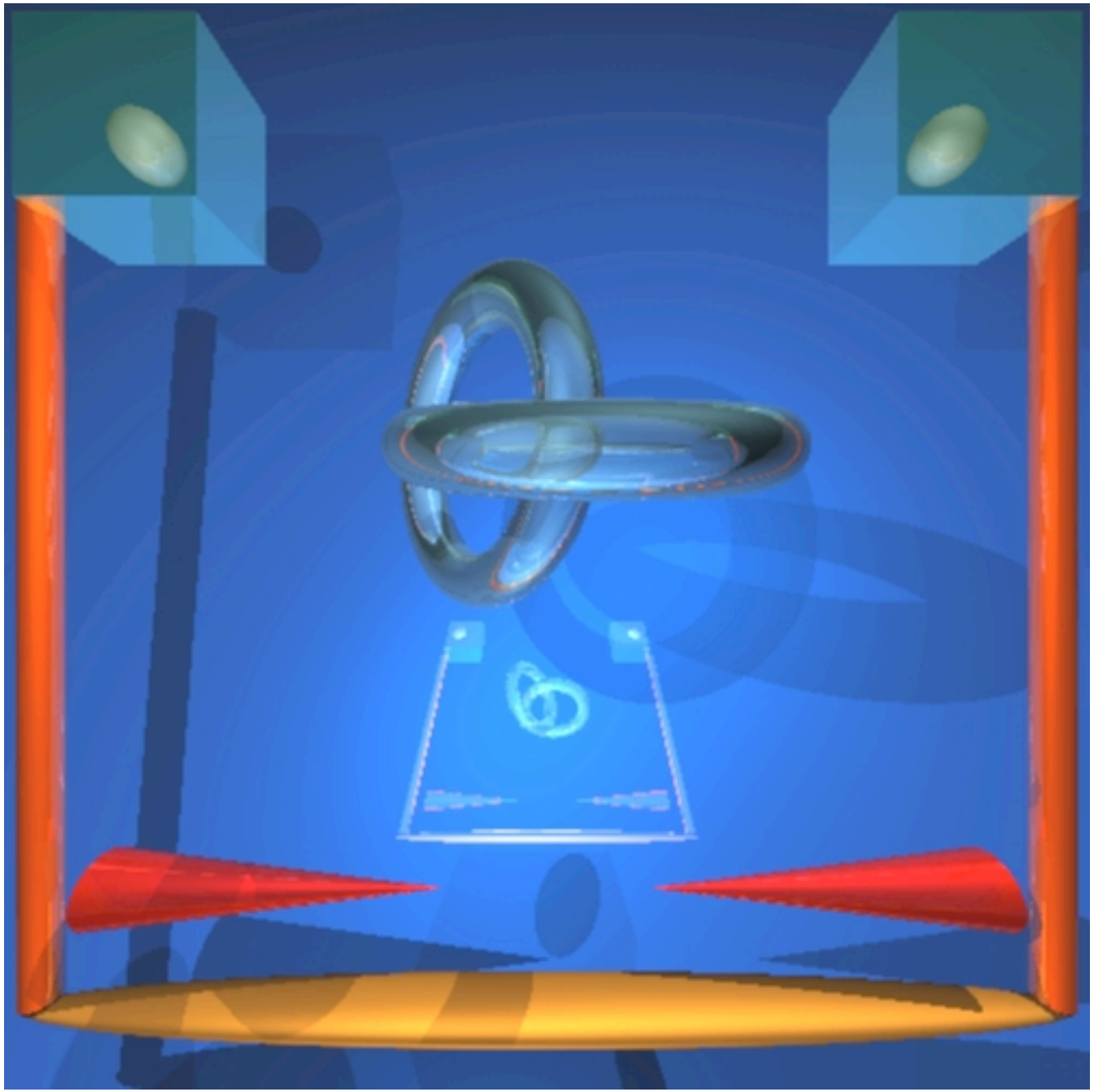
Linear Search

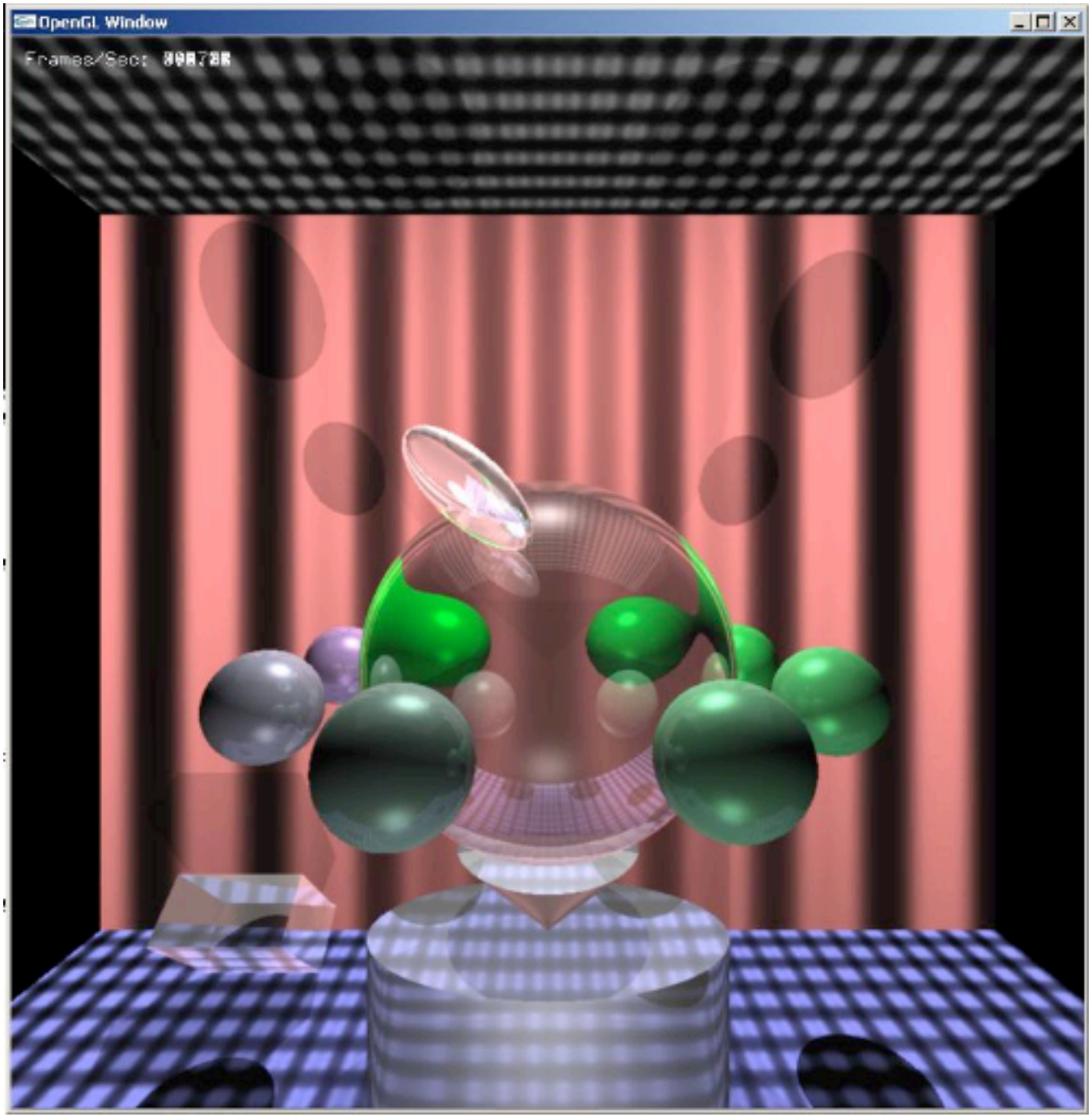
- Worst Case Analysis -- $O(n)$
- Average Case Analysis -- $O(n)$

Binary Search -- $O\big(Log(n)\big)$

- Root Finding (Mathematica Code)
- Ray Tracing


OpenGL Window
Frames/Sec: 7.84




OpenGL Window
Frames/Sec: 8987


Comp 360[Demo Application]
Frames/Sec: 59.76

Big-Ω

Definition

- f is $\Omega(g)$ means there are numbers k and C such that

  -- $f(n) \geq C\,g(n)$ for all $n > k$.

Meaning

- For *all* sufficiently large integers, $f(n)$ is greater than a *constant* multiple of $g(n)$.
- $\Omega(g)$ is only a Lower Bound on the size of f.

Observations

- k and C are not unique.
- g is not unique.

Big-Ω Notation

Notation (Abuse of Notation)

- $f = \Omega(g)$
- $f \in \Omega(g)$

Meaning

- $f \in \Omega(g)$
- $\Omega(g) = \{h \mid h(n) \geq C_h\, g(n) \text{ for all } n \geq k_h \text{ for some constant } C_h\}$

Diagrams

Big-Θ

Definition

- f is $\Theta(g)$ means there are numbers c, C and k such that

  -- $cg(n) \leq f(n) \leq Cg(n)$ for all $n > k$.

Meaning

- For *all* sufficiently large integers, $f(n)$ both $O(g)$ and $\Omega(g)$.
- f is $\Theta(g)$ is a Tight Bound on the size of f.

Observations

- k, c, C are not unique.
- g is not unique.

Big-Θ Notation

Notation

- $f = \Theta(g)$
- $f \in \Theta(g)$

Meaning

- $f \in \Theta(g)$
- $\Theta(g) = \{h \mid D\, g(h) \leq h(n) \leq C_h\, g(n)\ for\ all\ n \geq k_h\ \ for\ some\ constants\ C_h, D_h\}$
- $f \in \Theta(g) \Leftrightarrow f \in \Omega(g)\ and\ f \in O(g)$

Diagrams

Examples

1. $c_n x^n + \cdots + c_1 x + c_0 = \Theta(x^n)$

2. $$\frac{c_n x^n + \cdots + c_1 x + c_0}{d_n x^m + \cdots + d_1 x + d_0} = \Theta(x^{n-m})$$

3. $$\sum_{k=1}^{n} k^p = \Theta(n^{p+1})$$

4. $$\sum_{p=1}^{n} 2^p = \Theta(2^n)$$

Small o, ω

Definitions

- f is $o(g) \Leftrightarrow \lim_{x\to\infty} \dfrac{f(x)}{g(x)} = 0$ (much smaller)

- f is $\omega(g) \Leftrightarrow \lim_{x\to\infty} \dfrac{f(x)}{g(x)} = \infty$ (much bigger)

Observations

- f is $o(g) \Rightarrow f$ is $O(g)$ (much smaller)
- f is $\omega(g) \Rightarrow f$ is $\Omega(g)$ (much bigger)

Examples

- $\lim_{n\to\infty} \frac{\ln(n)}{n} = 0 \quad \Rightarrow \quad \ln(n) \ is \ o(n)$
- $\lim_{n\to\infty} \frac{n^p \ln(n)}{n^{p+1}} = 0 \quad \Rightarrow \quad n^p \ln(n) \ is \ o(n^{p+1})$
- $\lim_{n\to\infty} \frac{n^p}{2^n} = 0 \quad \Rightarrow \quad n^p \ is \ o(2^n)$
- $\lim_{n\to\infty} \frac{P(n)}{2^n} = 0 \quad \Rightarrow \quad P(n) \ is \ o(2^n), \ P(n) \ Polynomial$

Most Important

Big O is the most important complexity class, since we are most interested in bounding the time of an algorithm.

Additional Topics

P vs. NP

- Tractable vs. Intractable
  -- Factoring Integers and Cryptography
- Verifiable vs. Solvable
  -- SAT, Hamiltonian Circuits

Solvable vs. Unsolvable

-- Halting Problem

Other Models of Computation

- Parallel Computation
- Quantum Computing