COMP 6651

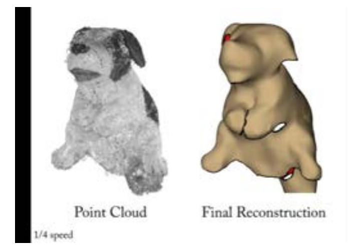
Algorithm Design Techniques

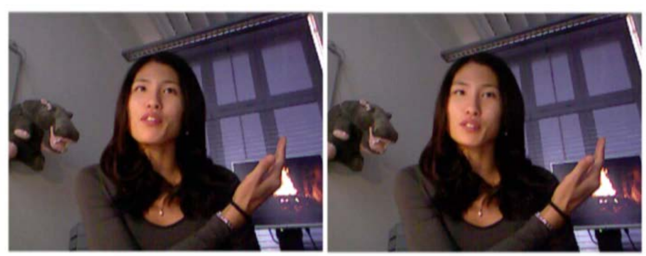
Week 1

Intro. Course Overview.

(some material is taken from web or other various sources with permission)

Computer Graphics





Instructor : Tiberiu Popa

- Office: EV 3.127
- Lectures: Fri17:45 - 20:15
- Office Hours: Wed 11:30 – 12:30 (EV 3.127)
- E-Mail: tiberiu.popa@concordia.ca
- Phone: (514)848-2424 ext. 4057
- TA: TBA
- Communication: Moodle
- Textbook – Mandatory!!!

Algorithm

- What is an algorithm?
- a procedure for solving a mathematical problem in a finite number of steps that frequently involves repetition of an operation
- a step-by-step procedure for solving a problem or accomplishing some end especially by a compute
- Non computer related examples?

Algorithm

- Non computer related algorithms?
- Surgery

Algorithm

- Non computer related algorithms?
- Dr Consultation

Algorithm

- Non computer related algorithms?
- Airplanes

Algorithm

- Non computer related algorithms?
- Automotive

Algorithm

- Non computer related algorithms?
- Automotive
  - **Loosen Lugs**
  - **Raise Car**
  - **Loosen Caliper**
  - …

Algorithm

- Non computer related algorithms?
- Euclid's algorithm for gcd (300 BC)
- Astronomical calendars (predicting the motion of the moon and planets (2000 BC)

Algorithm

- What is an algorithm?
- Modern definition:
  - Systematic method to solve problems
  - Often on a computer

Algorithm

- Example:
  - Go to the library and bring me a book
- Solution:
  - Check if the book is available and write down location
  - IF book not available msg me. Stop.
  - ELSE go to the physical location (building, floor shelf)
    - Pick it up
    - Check it out
    - Bring it to me

Algorithm

- Will this always work? What are the prerequisites?
  - Yes, internet connection
  - Books have to be organized (indexed)
- WHY?
- Data Structures!!!
  - Data organization is critical
  - To make algorithms more efficient

Pillers of Computer Science

Pillers of Computer Science

Algorithms

Data-structures

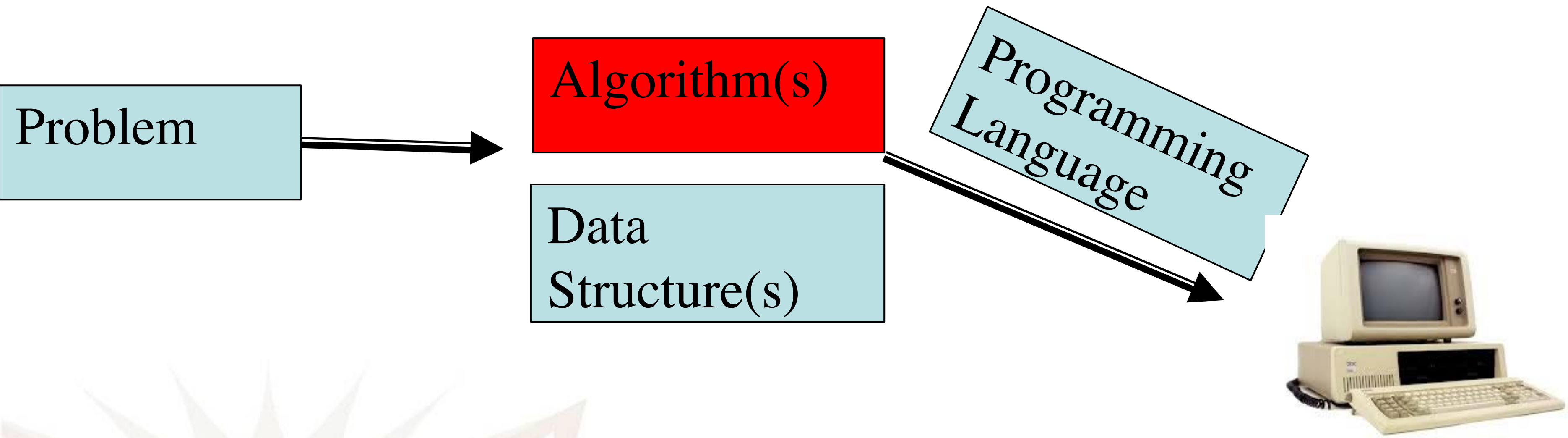
Programming language

Computer Science

Computer science: half Mathematics, half engineering

Composed of:

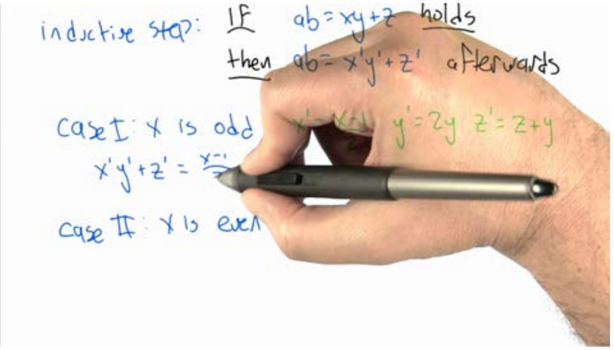
- Algorithms
- Data structures
- Programming languages



Algorithm Design

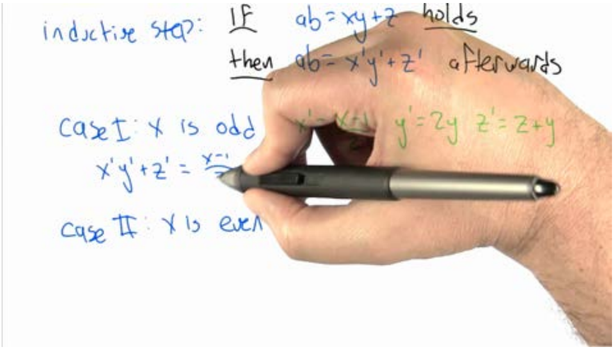
Ingredients:

1. Create the algorithm
   1. Pseudo-code
2. Analyze
   - Is it correct?
   - Is it efficient?
3. Implementation



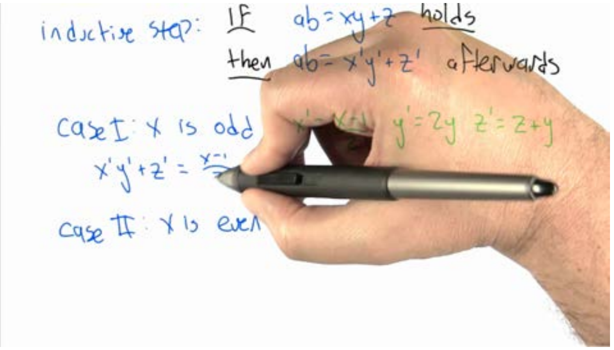
Algorithm Design

- Correctness
  - Theoretical (proof)
  - Empirical
- Efficiency
  - Theoretical (proof)
  - Empirical
- Implementation
  - C/C++

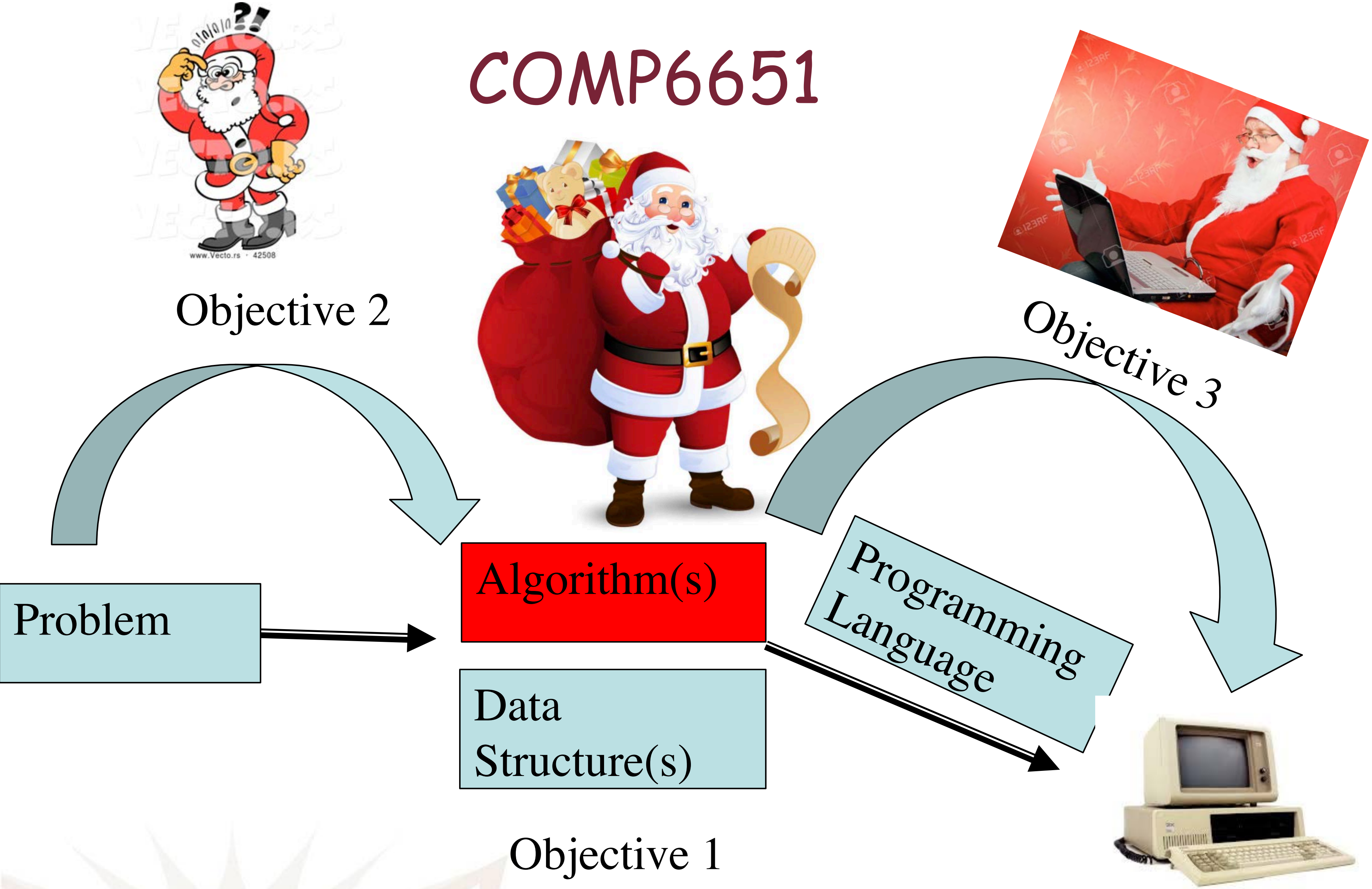


Algorithm Design

- Correctness
  - Theoretical (proof)
  - Empirical
- Efficiency
  - Theoretical (proof)
  - Empirical



COMP6651


Objective 2
Objective 3
Problem
Algorithm(s)
Data Structure(s)
Programming Language
Objective 1

Course logistics

- 13 lectures (see outline) - Theory
  - Algorithm paradigms (see outline)
  - Analysis
  - Problem solving

Course logistics

- 11 labs → Implementation
  - 1 Lab follows the topic of previous Fri
  - 5 instances / week (see schedule)
    - Same lab, several sessions
    - Mandatory!!!! –
    - No formal assignment, pick one first come first serve
  - Solve 1 sample problem of a given topic
  - Serve as an example for others
  - Critical if you want to do well

Course logistics

- Evaluation

Exercises (4)	12%
Midterm (1)	20%
Problems (2)	28%
Class Participation (bonus)	5%
Lab Attendance (bonus)	5%
Final Exam	40%

Course logistics

- Evaluation
  - Implementation
    - 4 exercises x 3%
      - Problems similar to the ones in the labs
    - 2 large problems x14%
      - Focus on problem solving

Course logistics

- Evaluation
  - Theory
    - 1 Midterm x20%
    - 1 Final exam x40%
      - Analysis and design (pseudocode)
      - Problem solving

Course logistics

- Evaluation
  - Lab Attendence (bonus)
    - >=10 labs = 5%
    - >=8 labs = 2%
    - Otherwise 0
  - Attendance is taken using your ID in the first 5 minutes
  - If you are late you miss it!!!!
  - If you leave early you miss it!!!!
  - TA are instructed to be VERY strict

Course logistics

- Evaluation
  - Class participation 5% (bonus)
    - All or nothing
    - At the discretion of the instructor

Course logistics

- Critical course
  - V large class
  - Difficult
  - Lots of work
  - Mandatory

Course logistics

- Critical course
  - Very large class
    - Cannot be helped
  - Mandatory
    - Important to know if you have a CS/SOFTENG degree
  - Difficult
    - If you do the work
    - We provide the support
    - 5 labs a week!!!
    - Lecture
    - Office hour

Course logistics

- Lots of work
- If you have a good undergraduate background
  - 75% of the material should be very familiar
  - **Everything I cover I learned in my undergraduate mandatory course**
  - 10—12 hours per week (standard)
- ELSE
  - You have to catch up!!!!
  - 15-20 hours per week

Course logistics

- I
  - Do my best to explain clearly the concepts
  - Labs – practice
- YOU
  - Due diligence
  - Do your labs
  - Do your homework
- We'll get along fine!!!!

Plagiarism

- Worst academic offence possible!!!
- Can have devastating consequences (up to and including removal from the program)
- In this class ALL your work must be original
- You are allow NO line of code from any other source except what is given in the lab and on moodle by the instructor
- Everything you submit is implicitly considered completely your own work

Plagiarism

- We are running automated checks with state of the art tools
- If a flag is raised you are called to explain
- If explanation is not satisfactory I will file a plagiarisms report with the dean
- It does not mater if you did not copy but you allowed someone else to copy after you!!!

Programming Environment

- Barebones C/C++
  - Why?
    - Simple and Easy
    - Show an example…
    - Was it easy?

Basics

- Algorithm evaluation
  - Efficiency
  - Correctness

Algorithm Efficiency

- A function of the size of input:
- Problem: Given an array of integers v[…], does it include an even number?
- Input has size N

```
for(int i=0;i<N;++i)
  if(cos(v[i])==0.7)
    return true;
return false;
```

- How many operations?

Algorithm Efficiency

- A function of the size of input:
- Problem: Compute maximum of a vector
- Input has size N
- How many operations?
  - Best case
  - Average case
  - Worst case
    - 2N-1
- Is it good?

```
for(int i=0;i<N;++i)
  if(v[i]%2==0)
    return true;
return false;
```

Algorithm Efficiency

- Typically look at worst case and average case
- Ex: quick_sort
  - Worst case – n^2
  - Average nlogn
  - In practice n

Algorithm Efficiency

- 2N-1
- Do we care about the constants?
- Intuitively:
  - If my input increases by 1, from N to N+1
  - How much many operations will it take
  - 2N-1 → 2N+1 → 2 extra operations
  - It scales fairly
  - aN+b → a extra operations
  - Constant may have some practical impact
  - Not dominant

Algorithm Efficiency

- N^2
  - N^2 → (N+1)^2 → 2N+1 extra operations
  - Does not scale so well
- N!
  - N! → (N+1)! → N*N! extra operations
  - With any extra input the algorithm explodes
  - Terrible
- Same for exponential
  - 2^N → 2^(N+1) → 2^N extra operations
  - doubles

Algorithm Efficiency

- We care about the asymptotic behavior
  - Either worst case or average case
  - Average case often difficult to analyze
- Ignore non-dominant elemengs (i.e. constants, lower ranked terms)
- Tools:
  - O notation(s)
    - $O(n)$ (1892)
    - o(n)
    - $\theta(n)$
    - $\omega(n)$
    - $\Omega(n)$

Why do we study this?

- Important in algorithm design
- If one understand this
  - Design much more efficient algorithms

Algorithm Efficiency

- $O(g(n)) = \{f(n) | \exists\, c > 0, n_o > 0\ s.t.\ 0 \le f(n) \le c\big(g(n)\big) \forall\, n > n_0\}$
- OK, but what is it? Looks like ancient greek
- O()→ is a SET of functions
  - Domain: space of functions
  - Input is a function
  - Co-domain/ output: set of function
- For example:
  - What is O(n) ?
  - (…. on the board ….)

Algorithm Efficiency

- $O(g(n)) = \{f(n) | \exists\, c > 0, n_o > 0 \; s.t.\, 0 \le f(n) \le c \cdot g(n). \forall\, n > n_0\}$
- Difficult to compute all functions belonging to this class
- More useful to ask if a function f(n) belongs to O(g(n))
- Examples:
  - Is $\mathrm{f}(n) = 2n + 1 \in O(n)$? Abuse of notation we say
    - IS $\mathrm{f}(n)$ $O(n)$?
  - Is $\mathrm{f}(n) = 2n + \log(n) \in O(n)$?
  - Is $\mathrm{f}(n) = n^2 + n \in O(n)$?
  - Is $\mathrm{f}(n) = n + 1 \in O(nlogn)$?
  - Is $O(n) == O(3n + 15)$?
  - Is $O(n) == O(an + b), a \neq 0$?
  - Is $O(n) == O(n^2)$?

Algorithm Efficiency

$O(g(n)) = \{f(n) | \exists\, c > 0, n_o > 0\ s.t.\ 0 \leq f(n) \leq c \cdot g(n). \forall\, n > n_0\}$

o$(g(n)) = \{f(n) | \forall c > 0, n_o > 0\ s.t.\ 0 \leq f(n) \leq c \cdot g(n). \forall\, n > n_0\}$

$\Omega\,(g(n)) = \{f(n) | \exists\, c > 0, n_o > 0\ s.t.\ f(n) \geq c \cdot g(n) \geq 0. \forall\, n > n_0\}$

$\omega\,(g(n)) = \{f(n) | \forall c > 0, n_o > 0\ s.t.\ f(n) \geq c \cdot g(n) \geq 0. \forall\, n > n_0\}$

$\theta\,(g(n)) = \{f(n) | \exists\, c_2, c_2 > 0, n_o > 0\ s.t.\ 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n). \forall\, n > n_0\}$

- Notes:
  - O and Ω – opposites
    - O – upper bound
    - Ω – lower bound
    - θ is both (sandwich)

Algorithm Efficiency

- In this class we use "mostly" O, Ω and θ
- O and Ω – opposites
    - O – upper bound
    - Ω – lower bound
    - θ is both (sandwich)

- Why do we care about the lower bound?
- Gives a theoretical limit
- Stop looking at faster algorithms
- Useful in reductions

Algorithm Efficiency

- Classical Example: sorting
  - You can prove that is O(nlogn) by providing an algorithm that is O(nlogn)
  - For instance: merge sort – O(nlogn)
  - If you can prove that is also Ω(nlogn)
    - nlogn → optimal algorithm

  - Exercise: prove that $f = O(g)$ and $f = \Omega(g)$ iff $f = \theta(g)$