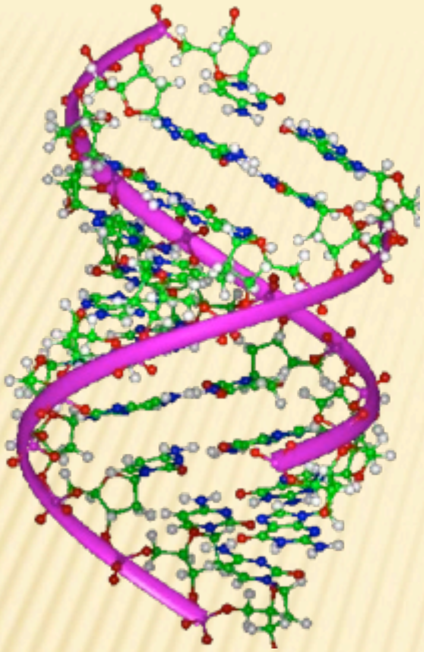


CSE-111 Great Ideas in Computer Science
Albert Y. C. Chen
University at Buffalo, SUNY

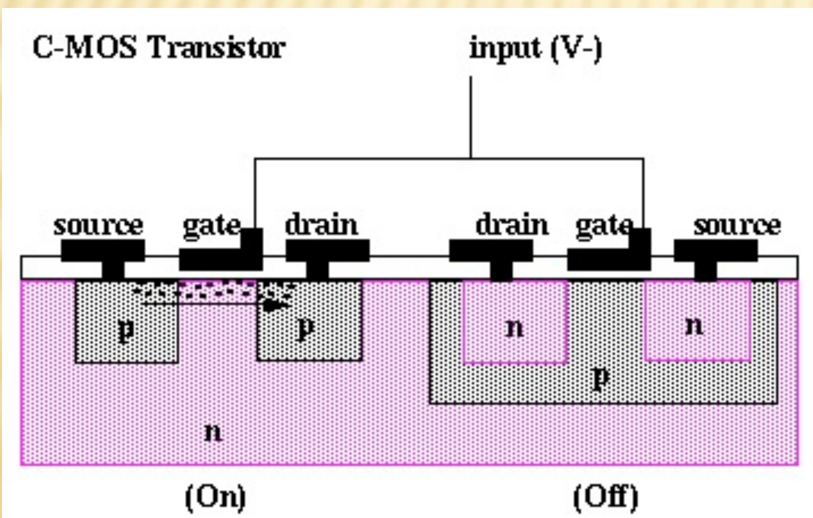
THINKING IN THE OBJECT ORIENTED WAY + A BRIEF INTRO TO CS RESEARCH

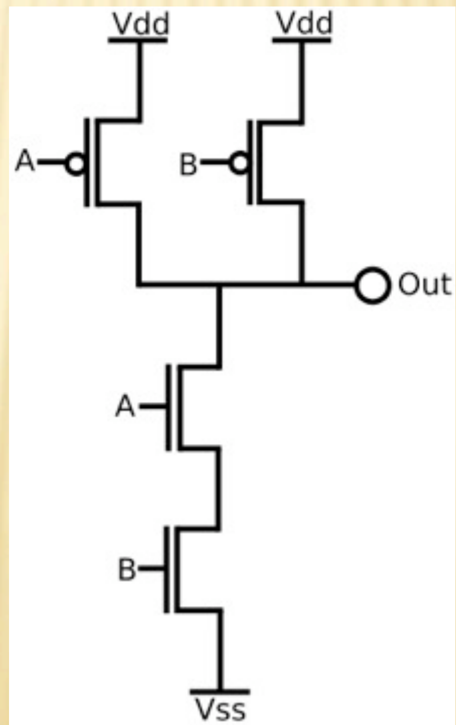
OVERVIEW

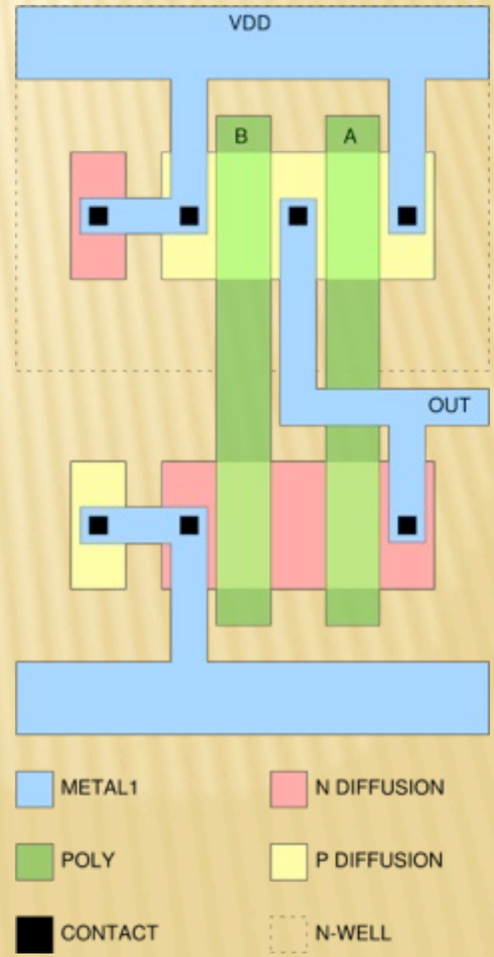
- Finally, putting everything all together. Also, we'll learn how complicated things are simplified by thinking in the object oriented way.
  - From logic gates to making CPUs!
  - From machine language to Karel-the-robot to 3D dragon drawing using the concepts of OOP!!
- A gentle introduction to the fields of research in computer science.
  - What can we do with all the powerful tools we have developed in the past few decades?

LOGIC GATES

- How are logic gates implemented on "semi-conductors"?
- e.g. NAND gate using CMOS:

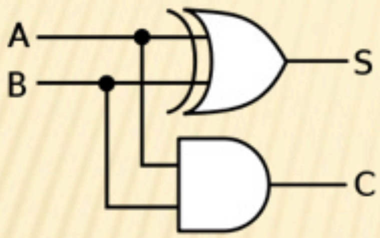




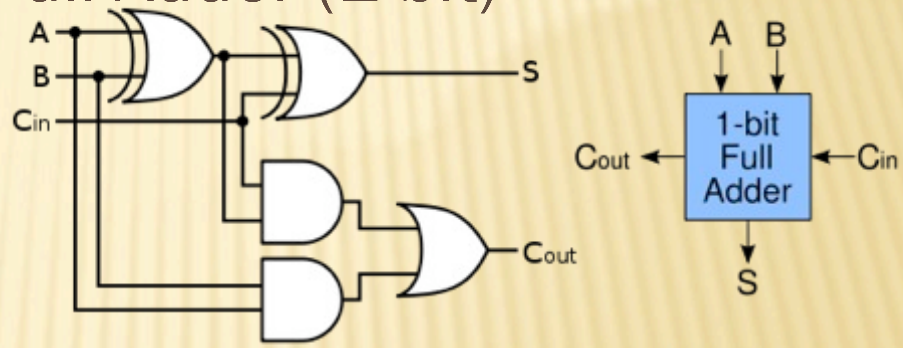


LOGIC GATES → BASIC ARITHMETIC UNITS

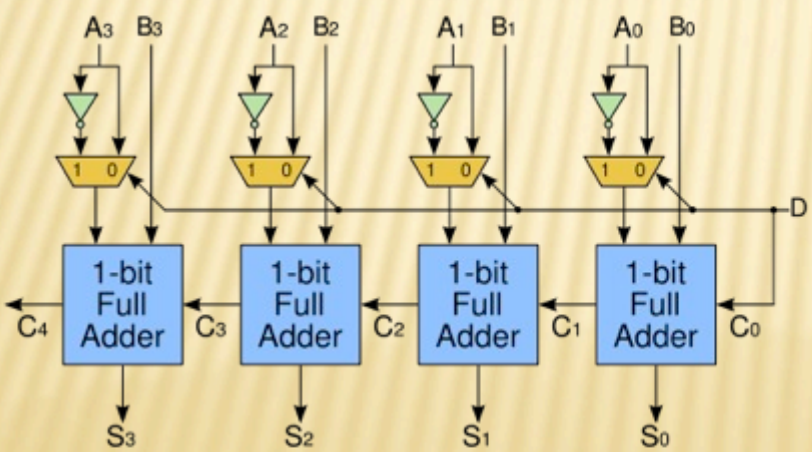
- Half Adder (1-bit)



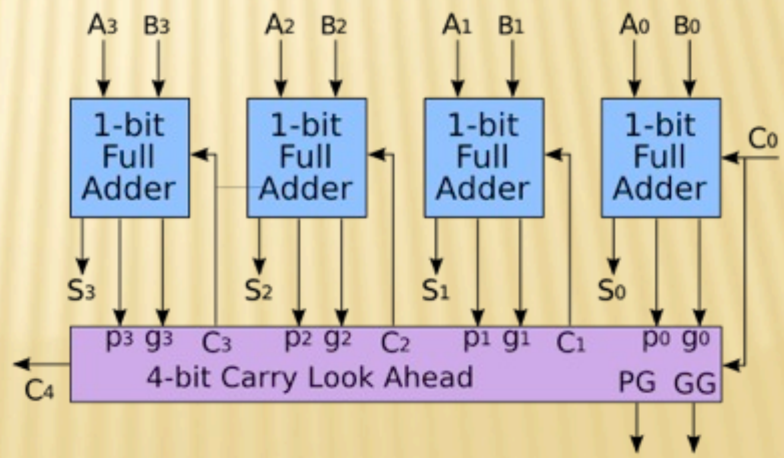
Full Adder (1-bit)



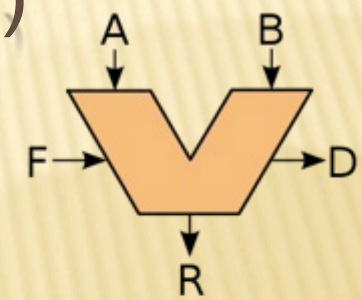
- Ripple Carry Adder



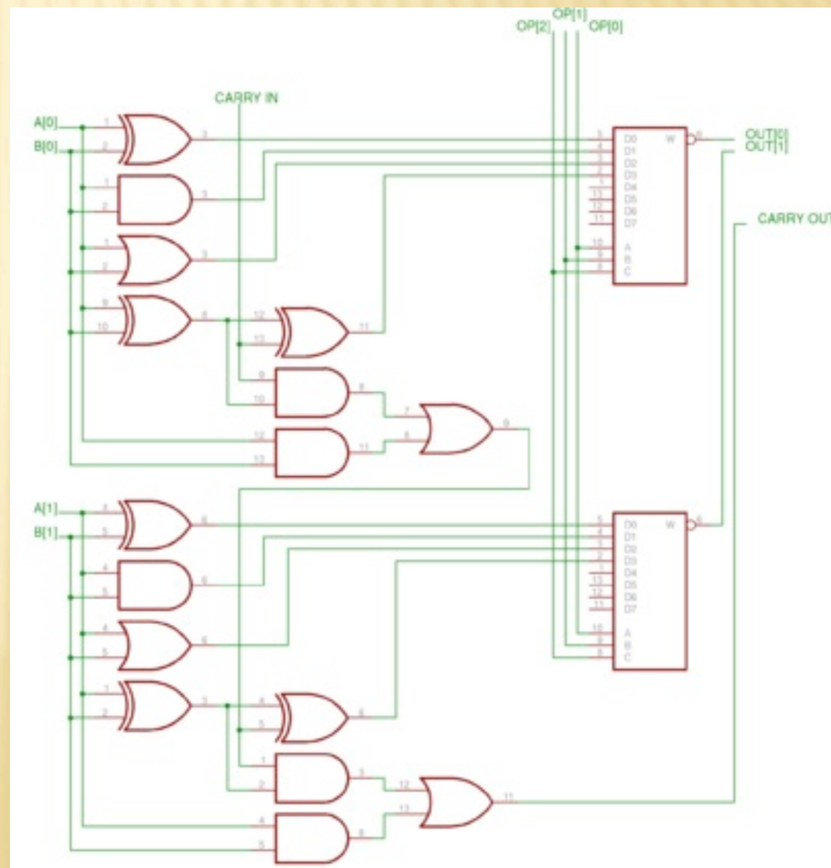
Carry look-ahead Adder



BASIC ARITHMETIC UNITS
→ ARITHMETIC LOGIC UNITS (ALU)

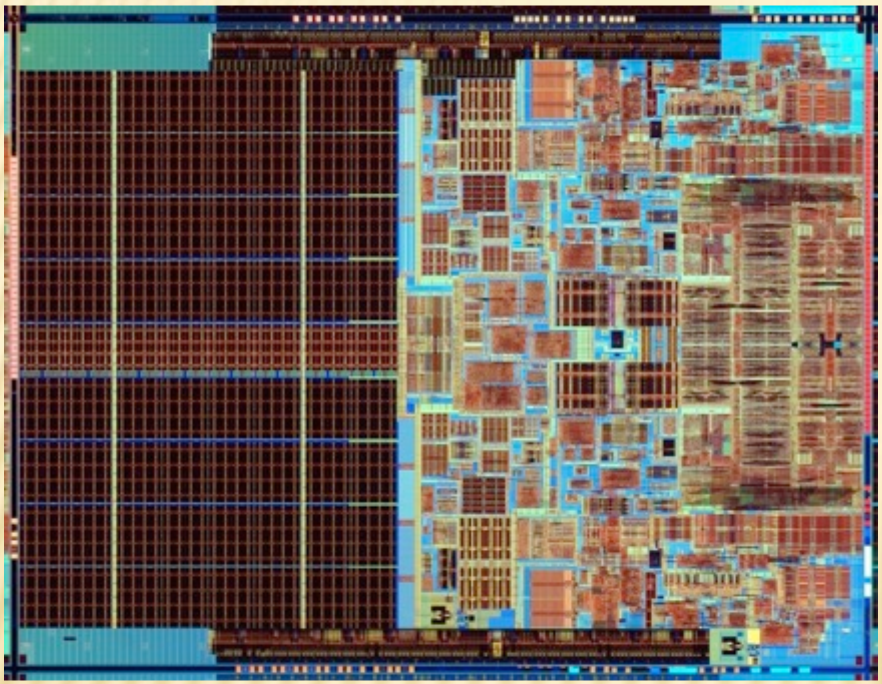


- ALU: Basic building block for computer's CPU (Central Processing Unit)
  - Takes input **A**, **B**, then perform the operation according to the given instruction **F.**
    - Addition, subtraction
    - Multiplication, division (optional)
    - Bitwise Logic op. (AND, OR, NOT)
    - Bit shifting operations
  - E.g. (right) 2 bit ALU that does XOR, AND, OR, addition.



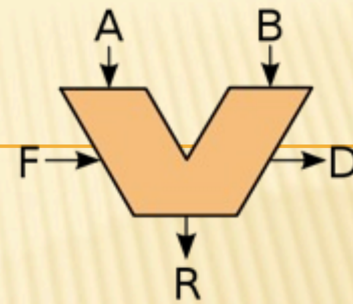
ARITHMETIC LOGIC UNITS (ALU) → CENTRAL PROCESSING UNIT (CPU)

- The “die” of Intel’s Core 2 Duo (Conroe) CPU



  - With machines so complicated and powerful, how are we going to send instructions (the control signal “F” in the ALU example) to the ALU?

INSTRUCTIONS FOR THE CPU?



- Starting with the 2 bit ALU.
  - We can have 00, 01, 10, 11 four different kind of control signals, i.e. we can integrate 4 different operations into this ALU (e.g. F=00 XOR; F=01 AND; F=10 OR; F=11 addition).
  - Examples
    - Suppose we want to perform A+B, where A = 00 and B = 10, and suppose the instructions sent to the ALU are 6 Boolean digits of the order “2 bit command” “2 bit A” “2 bit B”.
    - We’ll look up the command for “+”, which is “11”, then write “110010” in our program as the instruction for the ALU.
    - When the ALU receives 110010, it will decode it and know that you want it to set F=11 (which is to perform addition”, and the inputs are A = 00, B = 10

INSTRUCTIONS FOR THE CPU?

- Are these four instructions sufficiently enough?
  - Not actually, we'll also need instructions to:
    - Control flow instructions such as "if", "else".
    - Data control instructions to move things around in the memory.
- Should all commands be implemented in hardware?
  - CISC (Complex Instruction Set Computer):
    - Powerful commands, yet takes longer for each command to be executed.
  - RISC (Reduced Instruction Set Computer):
    - Simpler commands, each command runs faster, but may takes more commands to do a job.
  - For example, a ALU only capable of doing "addition" can calculate "2x5" by doing "2+2+2+2+2".

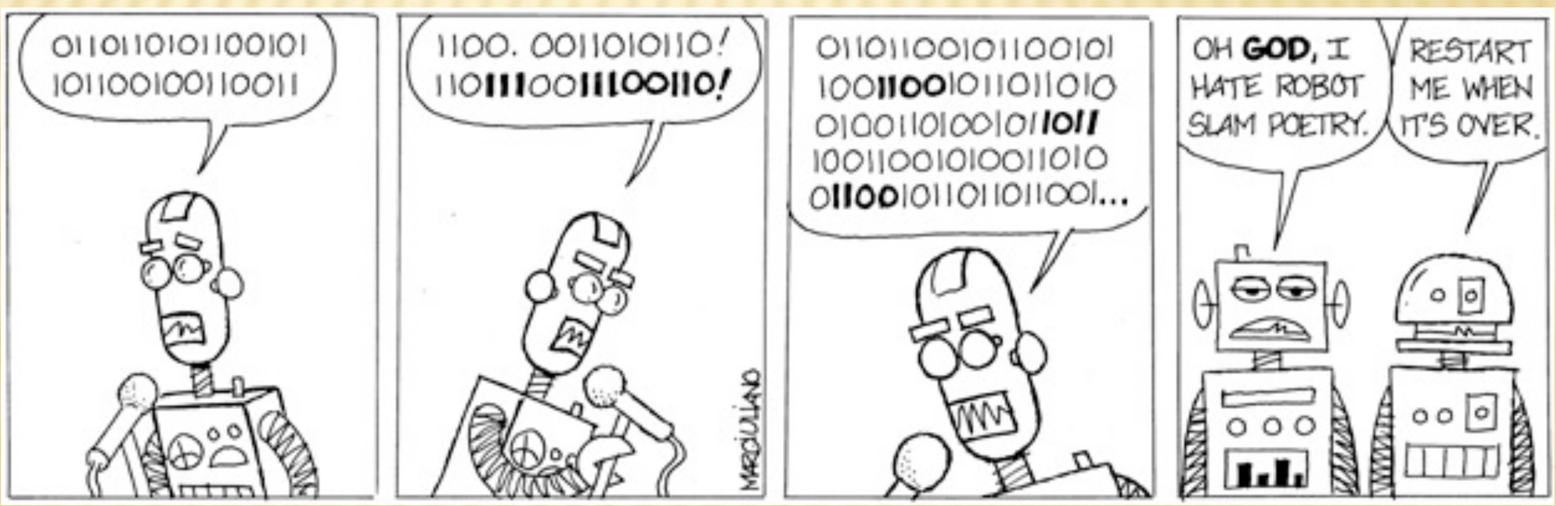
1ST GENERATION PROGRAMMING LANGUAGE

- Writing a program directly in CPU instructions:

  01010010 11001011 10100101 000100100 10011010 …

  - Ugh … Can we write these in decimal or hexadecimal?
  - OK, here's the 32-bit x86 machine code (1st generation programming language) to calculate the nth Fibonacci number (i.e. 0,1,1,2,3,5,8,13,21,34,55,89,…)

    8B542408 83FA0077 06B80000 0000C383 FA027706 B8010000 00C353BB 01000000 B9010000 008D0419 83FA0376 078BD98B C84AEBF1 5BC3



2ND GENERATION PROGRAMMING LANGUAGE

- Instructions that we can finally remember (sort of…)
- Still "Machine Dependent", thus still "low level".
  - i.e. code would need to be completely re-written whenever Intel or AMD introduces a new CPU…
- Fibonacci number calculator in MASM assembly language:

```
fib:
    mov edx, [esp+8]
    cmp edx, 0
    ja @f
    mov eax, 0
    ret
 @@:
    cmp edx, 2
    ja @f
    mov eax, 1
    ret
@@:
push ebx
mov ebx, 1
mov ecx, 1

@@:
    lea eax, [ebx+ecx]
    cmp edx, 3
    jbe @f
    mov ebx, ecx
    mov ecx, eax
dec edx
   jmp @b

   @@:
   pop ebx
   ret
```

High-Level Programming Languages (3rd Gen. and Beyond)

- Provides a higher level of abstraction from details of the computer (e.g. CPU commands, registers, ...etc.)
  - Our Karel-the-robot language is high level.
    - So are Basic, C, C++, Java, Python, ... almost any programming language you can think of.
    - The term “high” doesn’t mean it’s superior to low level languages, its that it provides a higher level of abstraction.
  - So, what’s the difference?
    - Instead of sending “11010101” or “mov edx [esp+8]”, we use commands that are very similar to plain English (if, else; while {...})
    - Instead of specifying which “CPU register” or “memory location” we’re going to access, we use “Variables” to store data.

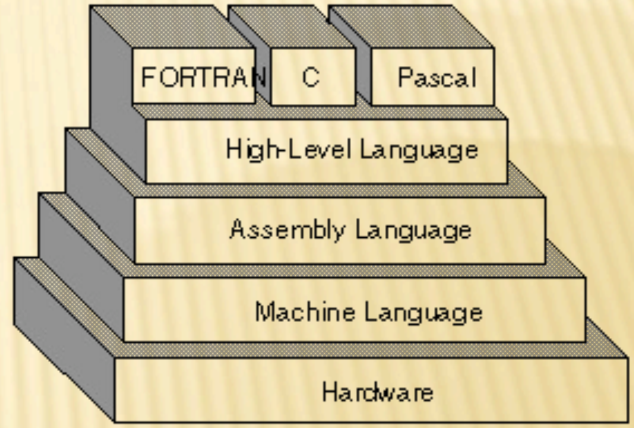
HIGH-LEVEL PROGRAMMING LANGUAGES – FIBONACCI NUMBERS REVISITED

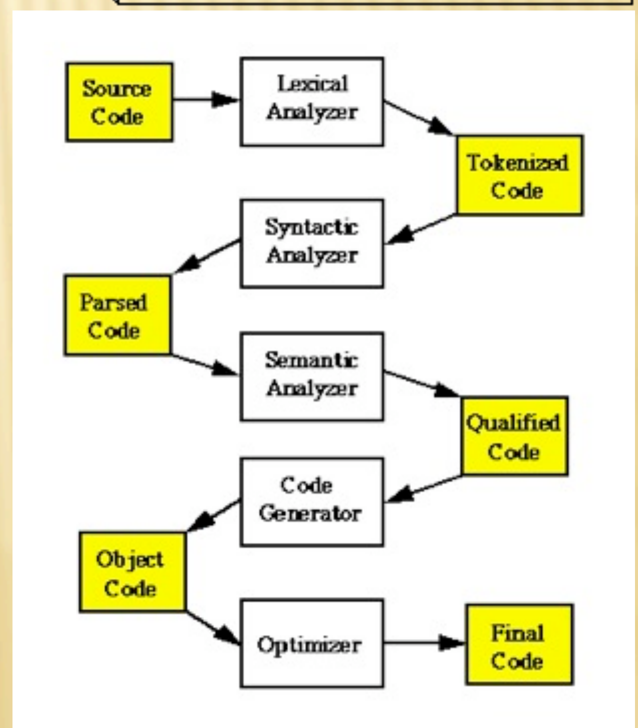
- Fibonacci Numbers: 0,1,1,2,3,5,8,13,21,34,55,89,…
  - The N^{th} number is the sum of the $(n-1)^{th}$ and $(n-2)^{th}$ number
- Using Karel's parental language "Pascal":

```
program fibonacci
var
    NumOfFibs, i, prevNumOne, prevNumTwo, currentNumber : integer;
begin
    NumOfFibs := 10; i:=0; prevNumOne := 1; prevNumTwo := 0;
    writeln(prevNumTwo); writeln(prevNumOne);
    while i<NumOfFibs-2 do begin
        currentNumber := prevNumOne + prevNumTwo;
        writeln(currentNumber);
        i := i +1;
        prevNumTwo := prevNumOne;
        prevNumOne := currentNumber;
    end;
end;
```

COMPILERS

- Compilers do all the hard work of translating programming languages to machine-specific instructions.
- When we write in Karel and hit the button “compile”, our code is examined by the compiler in the following order:
  - Lexicons (the vocabulary we used, e.g. “if”)
  - Syntax (e.g. missing “;”, “end;”)
  - Semantic…
- A translator of this sort will be needed for every two layers in the programming language pyramid shown in the upper right figure.





OBJECT ORIENTED PROGRAMMING (OOP)

- Why OOP?
  - Hardware and software became increasingly complex.
  - To assure “quality” and “code re-usability”
    - It’s impossible for a 3D game developer nowadays to write codes for each pixel he/she wants to display on the screen – It will not only be slow, but also prone to error.

Spacewar! (1961)

Ultima1 (1980)

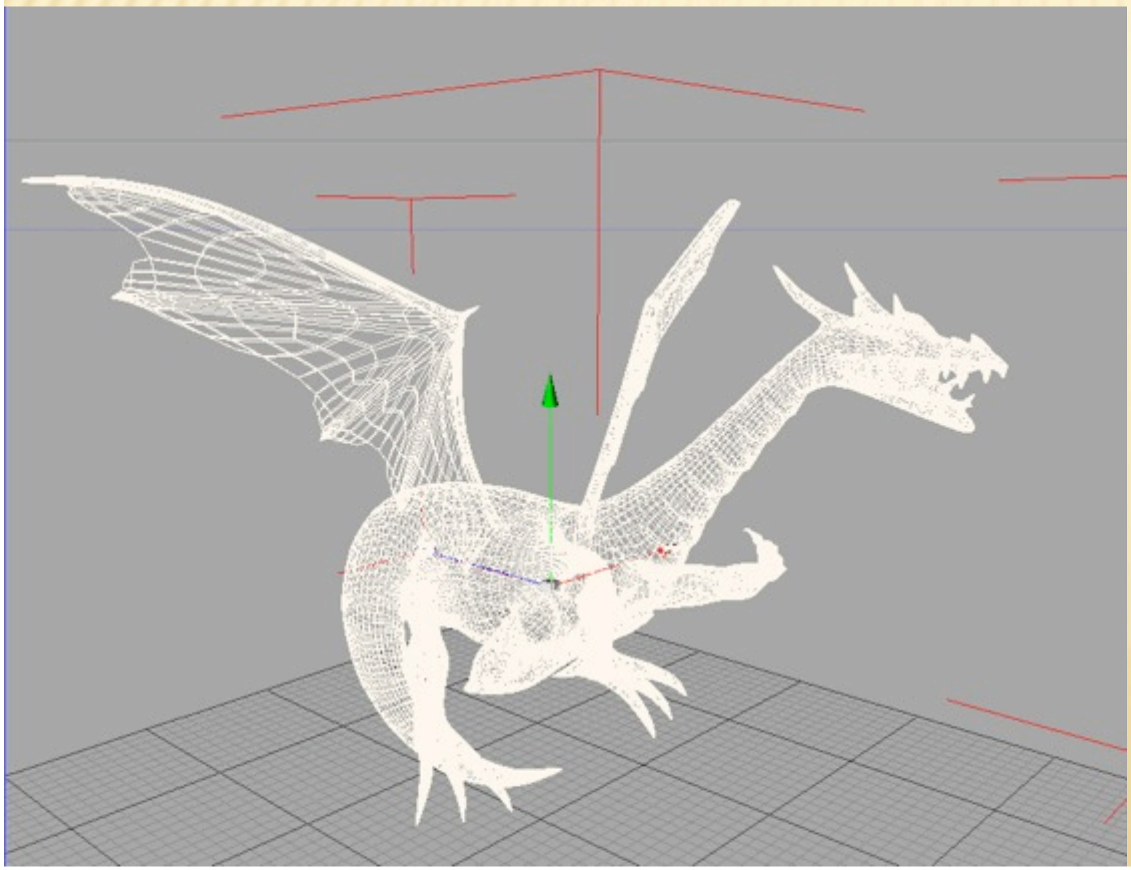
Doom (1993)

World of Warcraft (2004)

OBJECT ORIENTED PROGRAMMING (OOP)

- Yet another way of thinking in/teaching OOP.
  - Today, if we are "dragon behavior/motion specialists" and hired by our favorite gaming company to develop the latest game.
    - Wait… I don't know how to code in 3D yet!? (don't panic, be cool…)
    - Let's ask the programmers to do us a favor: give us a black box, we'll input the (x,y,z) location of the dragon's major joints, and the box will draw the dragon.
    - The programmers would go home and think: I guess the dragon's head, torso, tail, two wings and four legs will move separately. Maybe I'll need 9 smaller not-so-black boxes to implement the movements of these parts.

OBJECT ORIENTED PROGRAMMING (OOP)



- (Continuing with the “dragon coders” in the last slide)
  - Each smaller box will need at least the following components: a muscle object, and a skin object. The smaller not-so-black box will draw the head/tail according to the muscle object and skin object.
  - Suppose the “skin objects” are composed of 10 different kinds of scales, which we call it the “scale object”. Skin is drawn according to the scale objects provided.
  - Finally, each scale is composed of multiple “polygons” (basic drawing unit of 3D objects)

THEN WHAT?

- What have we learned from the hardware and software sides of the story?
  - When things gets huge and complicated, we better break down the problem nicely, work as a team, and each be in charge of developing a reliable part (object).
  - When we fit the parts (objects) together (or using those developed by our predecessors), we can create some really nice stuff.
- What tools besides “the concept of OOP” can we rely on to tackle other problems?
  - Powerful hardware, a nice operating system, high-level programming languages, and compilers that takes care of most low level interactions with the hardware.

A GENTLE INTRODUCTION TO THE FIELDS OF RESEARCH IN COMPUTER SCIENCE

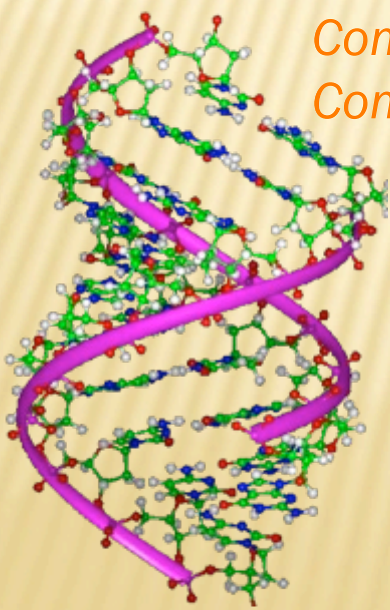
- Instead of starting with a boring long list of research topics, let's ask ourselves, what do we want our computers to do for us?
- Basic problems that trouble us:
  - "Run faster you stupid computer!!! "
    - If you want to devote yourself to fixing this problem, please refer to: *Digital Logic*, *VLSI*, *Computer Architecture*, *High Speed Computing* (including distributed and parallel processing)
  - "OS crashed!!! @#$%&*)#@% !!!"
    - *Operating System*, *Embedded Systems*
  - "Run/Compile you stupid Karel!!!"
    - *Programming Languages*, *Compilers*

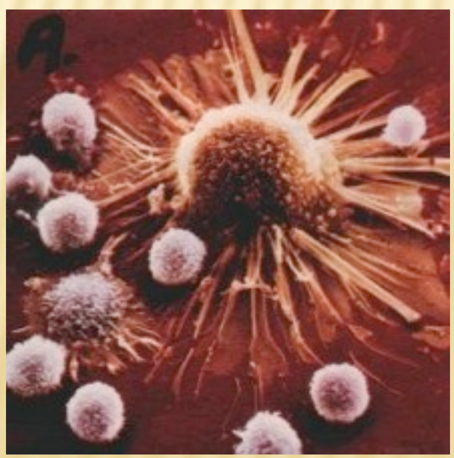
A GENTLE INTRODUCTION TO THE FIELDS OF RESEARCH IN COMPUTER SCIENCE

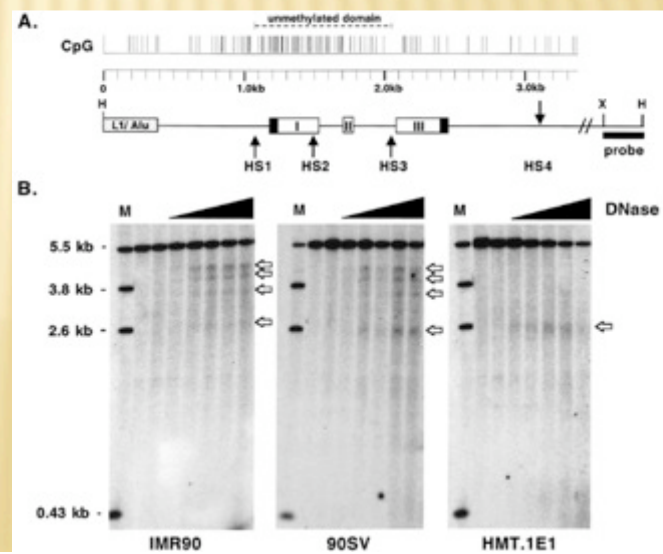
- Intermediate problems that trouble us:
  - "Why is my (wireless) internet so slow?"
    - *Network Design and Analysis*, *Wireless and Sensor Networks*, *Security*, probably even some *Information Theory and Coding*, *Graph Theories*, *Operating Systems*
  - "My SPAM filter doesn't work… Can I design a better one?"
    - *Machine Learning*
  - "I told my cell to call Monica, but it called mom"
    - *Signal Processing* and *Machine Learning*, better off with knowledge in Linear Algebra, Numerical Analysis, Probability and Statistics.

A GENTLE INTRODUCTION TO THE FIELDS OF RESEARCH IN COMPUTER SCIENCE

- Advanced problems that might occasionally trouble us:
  - “I want to study science, and help create a world without cancer” (from RPCI’s commercials)
    - There’s still a long way to go, but here’s what computer scientists can help other disciplines of research analyze their data:
    - *Bioinformatics* (including gene analysis) , *Cognitive Science*, *Computational Chemistry*, *Computational Neuroscience*, *Computational Physics*.

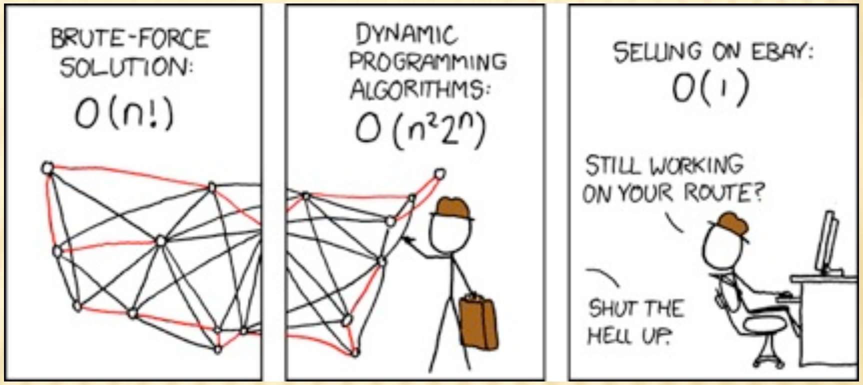






A GENTLE INTRODUCTION TO THE FIELDS OF RESEARCH IN COMPUTER SCIENCE

- Advanced problems that might never trouble us:
  - Problems such as the Traveling Salesman Problem (TSP):



    - Given a number of cities and the costs of travelling from any city to any other city, what is the least-cost round-trip route that visits each city exactly once and then returns to the starting city?
    - *Algorithms*, *Analysis of Algorithms*, *Computational Complexity Theory*, *Computability Theory*, *Graph Theory*, … etc.

A GENTLE INTRODUCTION TO THE FIELDS OF RESEARCH IN COMPUTER SCIENCE

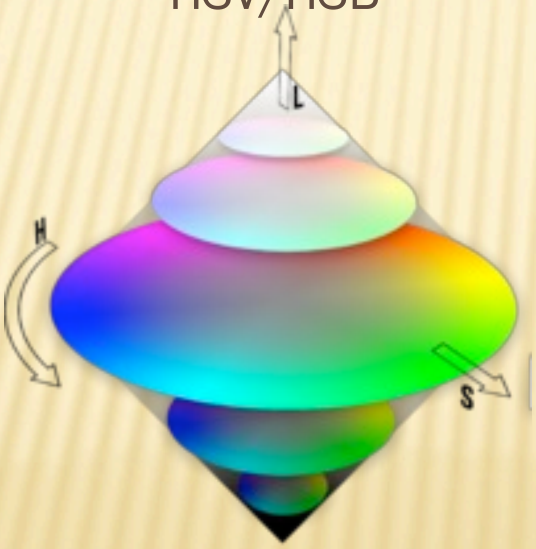
- Problems that only trouble us in our daydreams:
  - "Can't my computer just do everything for me? (study, take tests, drive the car, feed me…)"
    - *Artificial Intelligence*, *Computer Vision*, *Machine Learning*, *Pattern Recognition*, *Robotics*.
  - How hard is it to teach a computer to tell between leaves and grass, sky and ocean, roads and buildings?
    - Very very hard…
    - Even the most advanced systems could only achieve around 75%-80% accuracy.

BRIEF INTRODUCTION TO COMPUTER VISION

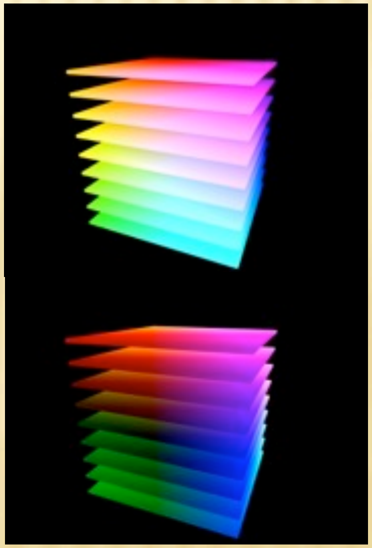
- Problems, problems, problems:
  - Given an image, what color space should one use? Or should one use infrared, ultraviolet sensors instead of visible light?

RGB

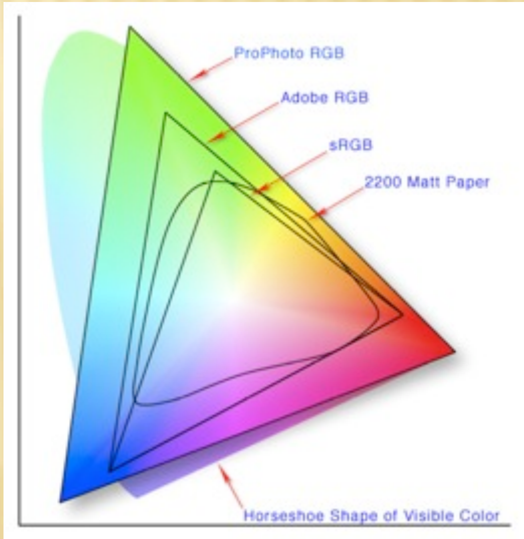
HSV/HSB



YUV



RGB v.s. visible colors



BRIEF INTRODUCTION TO COMPUTER VISION

- Take leaves for example, should we learn by their color? texture? shape? size!!?

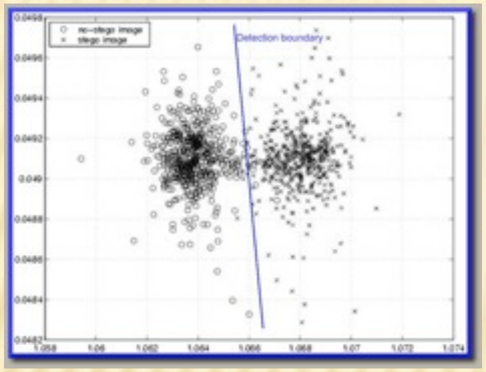


- Since there is no "universal solution", we usually take MANY of the features into account.
- How are we going to weigh/normalize different features? How are we going to calculate the similarity between features?
- *Pattern Recognition* will shed some light.

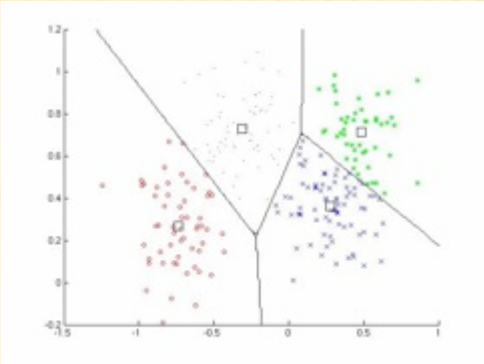
BRIEF INTRODUCTION TO COMPUTER VISION

- Pattern Recognition will try to learn/discriminate the distribution of different groups of objects.

LDA



K-means



SVM

  - These are just the "easier to illustrate" ones.
  - Supervised v.s. unsupervised?
  - Over learning v.s. generality?
  - Parametric v.s. Non-parametric?
  - Generative Models v.s. Discriminative methods?
- This is just the beginning of all the struggles ☺