

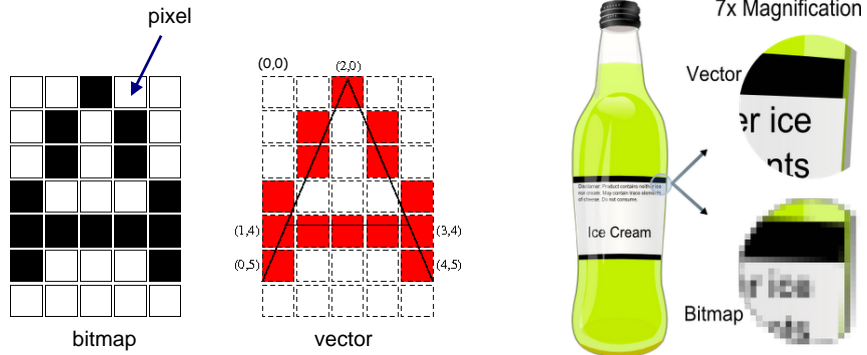
Lecture 1-1:

A Taste of Computer Science

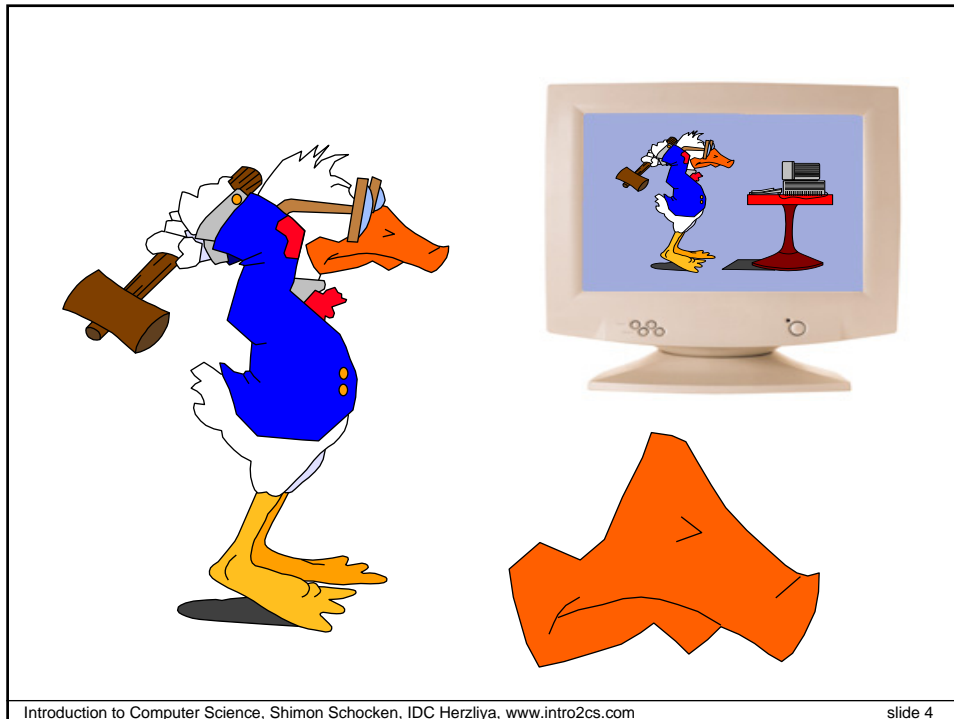
About this lecture

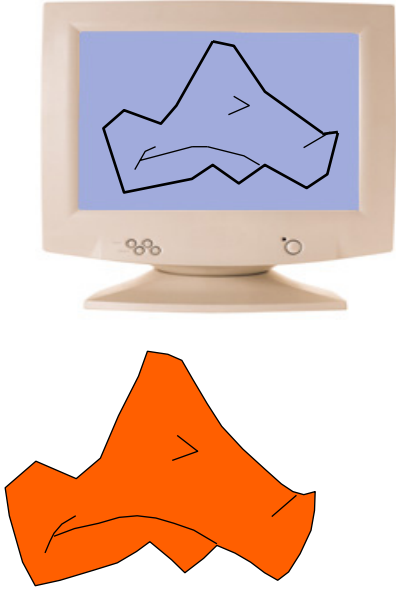
- In order to get a feel for CS, I will show some sophisticated stuff
- There is no need to understand the details of anything in this lecture!
- Objective: to give a taste of the spirit of CS

Computer graphics: bitmap versus vector



- **Bitmap file:** 00100, 01010, 01010, 10001, 11111, 10001, 00000, . . .
- **Vector graphics file:** drawLine(2, 0, 0, 5), drawLine(2, 0, 4, 5), drawLine(1, 4, 3, 4)
- In both methods, the only primitive operation that we have is **drawPixel(x, y)**
- Each method has its pros and cons
- In this lecture we will focus on **vector graphics**.



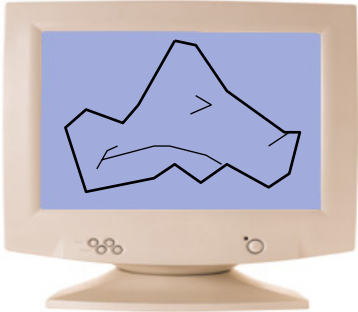


Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com slide 5

[It's all about line drawing](#)

- How to draw lines?
- How to draw them *fast*?

Remember: the only available operation is `drawPixel(x, y)`



[Simplify the problem](#)

Let's focus only on lines that go north-east.

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com slide 6

It's all about line drawing

- How to draw lines?
- How to draw them *fast*?

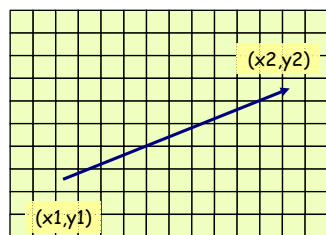
Remember: the only available operation is `drawPixel(x, y)`



Simplify the problem

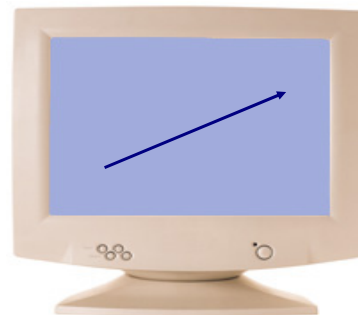
Let's focus only on lines that go north-east.

Primitive operation: `drawPixel(x,y)`



(0,0)

(Using a coordinate system with a conventional origin, to make things more intuitive)

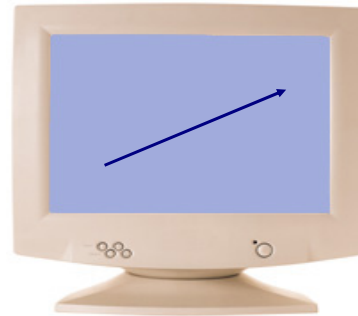
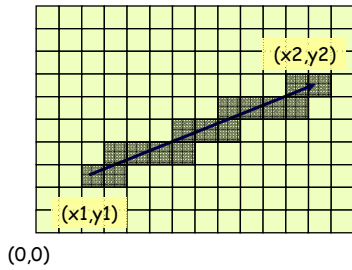


Simplify the problem

Let's focus only on lines that go north-east.

Task: implement `drawLine(x1, y1, x2, y2)` where $x2 > x1$ and $y2 > y1$, when the only available operation is `drawPixel(x, y)`

Primitive operation: drawPixel(x,y)



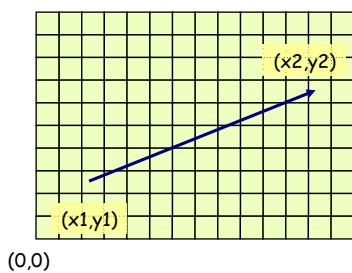
Simplify the problem

Let's focus only on lines that go north-east.

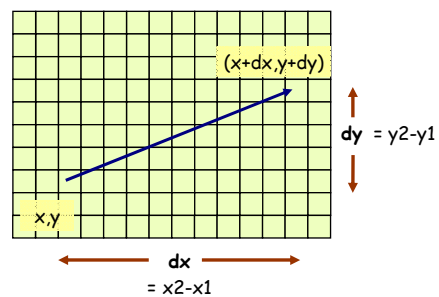
Task: implement drawLine(x1, y1, x2, y2) where $x_2 > x_1$ and $y_2 > y_1$, when the only available operation is drawPixel(x, y)

Insight: Because we always go north-east, in each step we either go north (up), or east (right).

drawLine(x1,y1,x2,y2)



drawLine(x,y,x+dx,y+dy)



Use a good notation

Elegance of expression is key.

Let variables **a** and **b** record how many pixels you went right and up, respectively

```

a=0, b=0    drawPixel(x+0,y+0)
a=1, b=0    drawPixel(x+1,y+0)
a=1, b=1    drawPixel(x+1,y+1)
a=2, b=1    drawPixel(x+2,y+1)
a=3, b=1    drawPixel(x+3,y+1) ...

```

In general:
a=a+1 or b=b+1 drawPixel(x+a,y+b)

drawLine(x,y,x+dx,y+dy)

```

set a=0, b=0
While there is more work to do:
  drawPixel(x+a,y+b)
  // set up for drawing the next pixel:
  to go right, set a=a+1
  to go up, set b=b+1

```

loop

Use a formal language

That's where programming enters the picture.

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com
slide 11

Let variables **a** and **b** record how many pixels you went right and up, respectively

```

a=0, b=0    drawPixel(x+0,y+0)
a=1, b=0    drawPixel(x+1,y+0)
a=1, b=1    drawPixel(x+1,y+1)
a=2, b=1    drawPixel(x+2,y+1)
a=3, b=1    drawPixel(x+3,y+1) ...

```

In general:
a=a+1 or b=b+1 drawPixel(x+a,y+b)

drawLine(x,y,x+dx,y+dy)

```

set a=0, b=0
While there is more work to do: ?
  drawPixel(x+a,y+b)
  // set up for drawing the next pixel:
  to go right, set a=a+1
  to go up, set b=b+1

```

```

a=0, b=0
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  // set up for the next pixel:
  to go right: a=a+1
  to go up: b=b+1

```

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com
slide 12

(x, y) $(x+a, y+b)$ $(x+dx, y+dy)$

dx dy

$\frac{dy}{dx}$

`drawLine(x,y,x+dx,y+dy)`

x, y $(x+dx, y+dy)$

$dx = x_2 - x_1$ $dy = y_2 - y_1$

Think abstractly

Look at the slopes.

```

a=0, b=0
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  // set up for the next pixel:
  to go right: a=a+1
  to go up:   b=b+1

```

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com
slide 13

$b/a > dy/dx$:
overshooting

(x, y) $(x+dx, y+dy)$

dx dy

$\frac{dy}{dx}$

$a = a + 1$

```

a=0, b=0
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  // set up for the next pixel:
  if b/a > dy/dx then a=a+1
  else b=b+1

```

```

a=0, b=0
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  // set up for the next pixel:
  to go right: a=a+1
  to go up:   b=b+1

```

Think abstractly

Look at the slopes.

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com
slide 14

Analyze:

The algorithm's run-time

Optimize:

Minimize run-time

$b/a > dy/dx$ implies $b*dx > a*dy$
Let: $diff = b*dx - a*dy$
Observations about diff:
 $a = b = 0$ implies $diff = 0$
 $b/a > dy/dx$ implies $diff > 0$
Therefore, instead of saying
if $b/a > dy/dx$ we can say if $diff > 0$
When we set $a = a + 1$,
we get $diff = diff - dy$
When we set $b = b + 1$,
we get $diff = diff + dx$

```
a=0, b=0
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  // set up for the next pixel:
  if b/a > dy/dx then a=a+1
  else b=b+1
```

Division! Oy Vey!

Perhaps we can get rid of the divisions



Analyze:

The algorithm's run-time

Optimize:

Minimize run-time

$b/a > dy/dx$ implies $b*dx > a*dy$
Let: $diff = b*dx - a*dy$
Observations about diff:
 $a = b = 0$ implies $diff = 0$
 $b/a > dy/dx$ implies $diff > 0$
Therefore, instead of saying
if $b/a > dy/dx$ we can say if $diff > 0$
When we set $a = a + 1$,
we get $diff = diff - dy$
When we set $b = b + 1$,
we get $diff = diff + dx$

```
a=0, b=0
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  // set up for the next pixel:
  if b/a > dy/dx then a=a+1
  else b=b+1
```



```
a=0, b=0
diff = b*dx - a*dy
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  if diff > 0 then a=a+1, diff = diff - dy
  else b=b+1, diff = diff + dx
```

Additions only!

How good is the algorithm?

Run-time analysis

- To draw a line consisting of n pixels, the algorithm performs n addition operations
- The fastest line drawing algorithm possible!

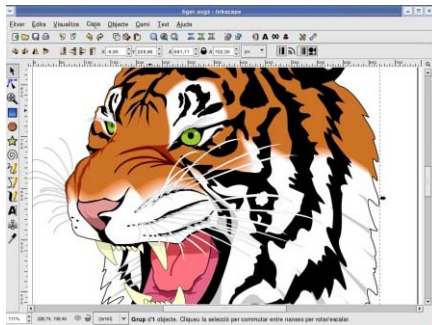


```
a=0, b=0
diff = b*dx - a*dy
while (a ≤ dx) and (b ≤ dy)
  drawPixel(x+a,y+b)
  if diff > 0 then a=a+1, diff = diff - dy
  else b=b+1, diff = diff + dx
```

Implications of this algorithm

Bottom up

- Now we can draw lines fast; therefore,
- We can draw polygons fast; therefore,
- We can draw images fast; therefore,
- We can draw 30 frames per second; therefore,
- We have Flash, animation, streaming, youTube, video on cellphones, . . .



Top down

- To display Flash files, you need vector graphics
- In vector graphics, you have to draw polygons
- To draw polygons, you need to draw lines
- To draw lines, you need to divide
- Division can be re-expressed as multiplication
- Multiplication can be reduced to addition
- Addition is easy.

Lessons for the beginning computer scientist

- **Abstraction:** when faced with a complex problem, ignore the details and focus on solving the essence
- **Simplicity:** Divide the problem into manageable sub-problems
- **Reduction:** Try to re-express the problem in terms of another problem that you know how to handle
- **Expression:** Invent and use a clean and sharp notation
- **Efficiency:** In many cases, the solution must be as efficient as possible
- **Elegance:** Whatever you do, try to keep it beautiful



Much of computer science is about abstraction, simplicity, reduction, expression, efficiency, and elegance.

About the Course:

Introduction to Computer Science
IDC Herzliya

Course resources

Course team:

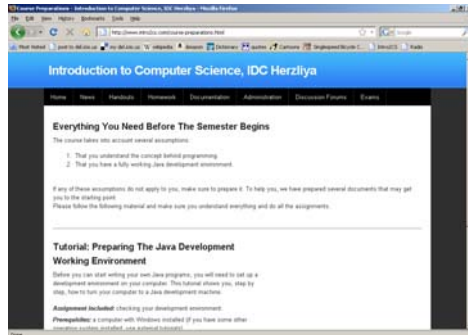
Instructor: Shimon Schocken

Teaching Assistant: Boaz Kantor

Chief Tutor: Idan Felix

Tutors team: Gil Moshayof, Iilit Raz, Steven Karas, Dan Sarig

Course web site:



www.intro2cs.com

Course textbook:



Java Software Solutions: Foundations of Program Design, by Lewis and Loftus, 6th edition (earlier editions are fine also), Publisher: Pearson / Addison Wesley

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com

slide 21

Course objectives

Upon completion of this course you will ...

- Become an informed Computer Science student
- Become a competent Java programmer
- Acquire a solid basis for subsequent CS courses

In addition, you will ...

- Sharpen your analytic skills
- Learn to think and plan clearly and elegantly
- Learn how to learn
- Develop a taste for beauty in science and engineering

Introduction to Computer Science, Shimon Schocken, IDC Herzliya, www.intro2cs.com

slide 22

Course rules

Requirements

- Attend / follow two weekly lectures
- Attend / follow one weekly recitation
- Submit a weekly homework assignment
- Visit the course web site at least once a day
- **Important:** the only formal and abiding channel of communications between the course team and the students is the course web site

How to ask questions and get help in this course

- Post your question on the course web site - you'll get an answer within 24 hours. The answer will be posted by the course team, or by another student
- Visit the lab in the days/times listed in the course web site. A tutor from the course team will be there to help.

Collaboration / cheating policy

- All homework assignments must be completed individually
- It is encouraged to talk to other students about the homework assignments
- But, this exchange should be restricted to talking only
- Once you start writing something together on a piece of paper or on the computer, you are crossing the line from collaboration to cheating
- **A very easy rule to remember:** any exchange of written notes about HW between students (hand-written, email, whatever) is cheating
- Cheaters will be prosecuted.