

# **CS 349: Algorithms**

Michael Noukhovitch

Winter 2015, University of Waterloo

Notes written from Michael Terry's lectures.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Definitions . . . . .	5
<b>2</b>	<b>Events</b>	<b>5</b>
2.1	Event Loop . . . . .	5
2.2	Timer . . . . .	5
2.3	Interactor Tree . . . . .	5
2.4	Event Propagation . . . . .	5
<b>3</b>	<b>Model View Controller</b>	<b>6</b>
3.1	Idea . . . . .	6
3.2	Description . . . . .	6
<b>4</b>	<b>Layout</b>	<b>6</b>
4.1	Layout Manager . . . . .	6
4.1.1	Dynamic Layout . . . . .	7
4.1.2	Layout Strategies . . . . .	7
4.2	Responsive Design . . . . .	7
4.2.1	CSS . . . . .	7
4.2.2	Cascade . . . . .	7
<b>5</b>	<b>Visual Design</b>	<b>7</b>
5.1	Rules . . . . .	8
5.2	Gestalt Principles . . . . .	8
<b>6</b>	<b>Transformation</b>	<b>8</b>
6.1	Basics . . . . .	8
<b>7</b>	<b>Widgets</b>	<b>9</b>
7.1	Definition . . . . .	9
7.2	Types . . . . .	9
7.2.1	Simple Widgets . . . . .	9
7.2.2	Container Widgets . . . . .	9
7.2.3	Abstract Model Widgets . . . . .	9
7.3	Widget Toolkit . . . . .	10
<b>8</b>	<b>Responsiveness</b>	<b>10</b>
8.1	Human Deadline . . . . .	10
8.2	AJAX . . . . .	10
<b>9</b>	<b>Design Process</b>	<b>11</b>
9.1	Formal Language . . . . .	11
9.2	User Centered Design . . . . .	11
9.2.1	Understand the User . . . . .	11
9.2.2	Design the UI . . . . .	12
9.2.3	Refine the Design . . . . .	12

<b>10 Experimentation</b>	<b>12</b>
10.1 Definitions . . . . .	12
10.2 Experimental Process . . . . .	13
10.3 Study Designs . . . . .	13
10.4 Experimentation Ethics . . . . .	13
<b>11 History</b>	<b>14</b>
11.1 Batch Interface . . . . .	14
11.2 Conversational Interface . . . . .	14
11.3 Graphical Interface . . . . .	14
<b>12 Input</b>	<b>15</b>
12.1 Classification . . . . .	15
12.2 Text Input . . . . .	15
12.2.1 QWERTY vs Dvorak . . . . .	15
12.2.2 Keyboard Design . . . . .	15
12.3 Position Input . . . . .	16
12.4 Gestural Input . . . . .	16
<b>13 Input Performance</b>	<b>16</b>
13.1 Keystroke Level Model . . . . .	17
13.2 Fitt's Law . . . . .	17
13.3 Steering Law . . . . .	18
<b>14 Visual Perception</b>	<b>18</b>
14.1 Temporal Resolution . . . . .	18
14.2 Spatial Resolution . . . . .	18
14.3 Color Perception . . . . .	18
14.4 Peripheral Vision . . . . .	19
<b>15 Typography</b>	<b>19</b>
15.1 Defintions . . . . .	19
15.1.1 General . . . . .	19
15.1.2 Printing Terminology . . . . .	19
15.1.3 Font Terminology . . . . .	19
15.2 Computer Typography . . . . .	20
15.3 Design Rules . . . . .	20
<b>16 Accessibility</b>	<b>20</b>
16.1 Temporary Disability . . . . .	20
16.2 Inclusive Design . . . . .	20
<b>17 Touch Interfaces</b>	<b>21</b>
17.1 Display . . . . .	21
17.2 Input Challenges . . . . .	21
17.3 Interaction . . . . .	21
17.3.1 Principles . . . . .	21
17.3.2 Challenges . . . . .	22
17.4 Design . . . . .	22

17.5 Implementation . . . . .	22
<b>18 Touchless Interfaces</b>	<b>22</b>
18.1 Introduction . . . . .	22
18.2 In-Air Gestures . . . . .	22
18.3 Speech Interfaces . . . . .	23
<b>19 Information Visualization</b>	<b>23</b>
19.1 Why . . . . .	23
19.2 What . . . . .	23
19.2.1 Channel Types . . . . .	23
19.2.2 Channel Effectiveness . . . . .	24
19.3 How . . . . .	24

# 1 Introduction

## 1.1 Definitions

**Interface:** external presentation to user

- **controls:** manipulated to communicate intent
- **presentation:** what communicates response

**Interaction:** the actions a user must do to elicit corresponding response

- action and dialog
- unfolds over time

## 2 Events

### 2.1 Event Loop

```
while(true) {  
    if there is an event on queue:  
        dequeue it  
        dispatch it  
}
```

### 2.2 Timer

Some events are triggered by a timer, if that event's execution time is longer than the timer interval then by the end of the event execution, you should add another of your event to the queue!

### 2.3 Interactor Tree

We need a way to send information about what object is clicked

**interactor tree:** hierarchical tree-based organization of widgets

- each component's location is specified relative to parent
- we use **containers** whose sole purpose is to contain components
- events go **down** the tree to **capture** the target clicked
- event bubble **up** the tree to **handle** an event (e.g. `EventListener`)

### 2.4 Event Propagation

when an event happens:

1. calculate the parent node path
2. loop through it and execute capture phase handlers

3. execute DOM level 1 phase handler
4. execute bubble phase handlers
5. execute default browser behaviour

## 3 Model View Controller

### 3.1 Idea

We decouple presentation from data using the **observer** design pattern. This separation allots benefits:

- **change the UI**: easy to change how we interact with data
- **multiple view**: have different views of same data
- **code reuse**: different logic for same view etc..
- **testing**: data separation allows better logic testing

### 3.2 Description

**Model**: manages the data

- represent the data
- methods to manipulate data
- create and notify listeners

**View**: manages the presentation

- renders the data in a model
- references to the model
- is a listener to the model

**Controller**: manages user interaction

- between the model and view
- helps interpret input and model events

## 4 Layout

### 4.1 Layout Manager

**Layout Manager**: keeps the layout for components given their constraints and preferences

- uses composite and strategy design pattern

#### 4.1.1 Dynamic Layout

**Dynamic Layout:** maintain consistency with spatial layout

- reallocate space for widget
- adjust location and size
- change visibility, look, feel

#### 4.1.2 Layout Strategies

- **fixed layout**
- **intrinsic size:** find each item's preferred size and the container will grow to perfectly contain each item
- **variable intrinsic size:** layout determined in bottom-up and top-down phases
- **struts and spring:** items can either be fixed (strut) or variable (spring)

### 4.2 Responsive Design

**Responsive Design:** change layout to adapt to screen sizes of different devices

#### 4.2.1 CSS

**CSS:** specifying formatting

- consistency
- reduce size (cache CSS)
- code reuse
- separation of concerns

**CSS reset:** normalize appearance across browsers

#### 4.2.2 Cascade

Layout resolves CSS rules and renders following these rules:

1. find all declarations that match the element
2. sort declarations by **!important**
3. sort by origin (author > web browser)
4. sort by specificity of selector
5. sort by order (later rule wins)

## 5 Visual Design

Impose as little thinking as possible on the user

## 5.1 Rules

### Simplicity:

- facilitate recognition instead of recall
- use only the essentials

### Consistency:

- exploit perceptual patterns
- avoid ambiguous presentation
- present information consistent with user goals

### Organization and Structure:

- grouping
- hierarchy
- relationship

## 5.2 Gestalt Principles

Theories of visual perception that describe how people organize groups

- **proximity**: elements associated with nearby elements
- **similarity**: visual similarity
- **common fate**: moving together
- **continuity**: continuous forms are easy to perceive
- **closure**: see a complete figure even when info is missing
- **symmetry**
- **area**: visual field split into background and foreground
- **uniform connection**: connecting lines/regions
- **alignment**

## 6 Transformation

### 6.1 Basics

`translate` add scalar

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$



**scale**      multiply by scalar

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

**rotate**       $x' = x \cos \Theta - y \sin \Theta$   
                  $y' = x \sin \Theta + y \cos \Theta$

$$\begin{bmatrix} \cos \Theta & -\sin \Theta & 0 \\ \sin \Theta & \cos \Theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

## 7 Widgets

### 7.1 Definition

**Logical Input Device** : graphical component defined by its function/behaviour

**Widget** : part of an interface that has its own behaviour

- the **model** manipulated by the widget
- the **events** generated by the widget
- the widget **properties** which change behaviour and appearance

### 7.2 Types

#### 7.2.1 Simple Widgets

- labels and images
- button
- boolean *e.g. radio button*
- number *e.g. slider*
- text field

#### 7.2.2 Container Widgets

- pane
- tab
- menu
- list choice *e.g. dropdown menu*

#### 7.2.3 Abstract Model Widgets

...

## 7.3 Widget Toolkit

**Widget Toolkit:** software that defines a set of (event-driven) GUI components via API

- **complete:** GUI designers have all they need
- **consistent:** look, feel and usage paradigms are consistent
- **customizable:** devs can reasonably extend functionality

There are two common implementations of widgets:

- **Heavyweight widgets:** OS provides widgets and hierarchical windowing system *e.g. X, HTML*
- **Lightweight widgets:** OS provides top-level window, use toolkit for drawing and events *e.g. Java Swing*

## 8 Responsiveness

**Responsiveness:** the fulfillment of a user's real-time needs

### 8.1 Human Deadline

Make your program seem faster by using tricks to respond to a user's actions:

- busy indicator
- progress indicator
- rendering important information first
- fake heavyweight computations *e.g. scrolling*
- work ahead during periods of low load

### 8.2 AJAX

Classic web architecture relied on 'thin client, fat architecture', side step it using JS:

- AJAX issues call to web server (API)
- server handles request and returns data feed
- client updates UI with feed using JS

Advantages:

- minimize bandwidth
- increase speed
- avoid HTML headers

**Example 8.1.** AJAX backend using Bottle (Python)

```

from bottle import route, run, template, request

@route('/ajax', method='POST')
def ajax():
    # Retrieve the JSON
    json_data = request.json

    # upper-case and return the text
    upper = json_data['text'].upper()

    # Bottle will automatically send back
    # Python dictionaries as JSON objects
    return {'reply': upper}

```

## 9 Design Process

We need to have descriptions of UI to clarify intent and **define interaction**.

### 9.1 Formal Language

The design process can be strongly described with formal languages such as **Finite State Machines** allowing us to label states and transitions. But this rigid formality has troubles describing the rich informal world of the user:

- complexity of interfaces is too great
- can't clearly represent some ideas *e.g. timed events*
- transition time (to create/update model) is not negligible

### 9.2 User Centered Design

Design (and test) with real people in mind, using semi-formal languages

- understand user needs
- design UI first (then architecture)
- iterate
- use it yourself
- observe others using it

This can be put into a design process:

#### 9.2.1 Understand the User

- observe existing solutions
- list scenarios
- list functions required
- prioritize (freq and commonality)

### 9.2.2 Design the UI

- identify and design components
- design component distributions
  - storyboards
  - interaction sequences: macro structure
  - interface schematics: micro structure
- test the design with users
  - prototyping (high vs low fi, paper, Wizard of Oz)
  - user/usability studies
- iterate again!
- document the design
  - visual vocabulary

### 9.2.3 Refine the Design

- refine requirements
- add new scenarios
- walk through new scenarios
- adjust UI design

## 10 Experimentation

### 10.1 Definitions

**Hypothesis** : specific, falsifiable idea of how and which variables influence outcome

**Independent Variables** : variables manipulated by experimenter

**Dependent Variables** : variables that are measured

**Null Hypothesis** : there exists no relationship between independent and dependent variables

**Control Condition** : no experiment performed

**Experimental Condition** : experimental variable is manipulated

## 10.2 Experimental Process

1. formulate hypothesis
2. identify independent/dependent variables
3. design experiment
4. check for:

**validity** : are we measuring what we say we are

**reliability** : do we get same results reliably

**confounds** : are there influential variables we don't control for

**subject pool** : do the subjects represent intended users

5. select representative subjects
6. randomly assign to conditions
7. analyze results

## 10.3 Study Designs

**Between subjects:** each subject experiences one condition

- + no learning effects
- need more people
- problems with variation in subjects

**Within subjects:** each subject experiences every condition

- + order of conditions can counterbalance learning
- + need fewer subjects
- learning effects can persist

## 10.4 Experimentation Ethics

**Deception:** manipulating the truth by hiding or showing false information

1. **System Image Deceptions:** deceiving what the system is doing/can do  
(*e.g.* fluid progress bars)
2. **Behaviour Deceptions:** taking advantage of physical/sensory limits  
(*e.g.* “smooth” transition)
3. **Mental Mode Deceptions:** manipulating user's mental mode  
(*e.g.* fake static noise in Skype)

**Benevolent Deception:** deception that can help the user:

- close gap between expectation and reality (*e.g.* 15 minutes remaining)

- balance needs of individual vs group (*e.g.* noise in search results)
- protect user from themselves (*e.g.* ask whether user meant to delete something)
- satisfy design goals (*e.g.* placebo buttons)

**Malvolent Deception:** deceptions that benefit owner at user's expense

- using confusing language (*e.g.* using double negatives)
- hiding certain functionality (*e.g.* hide unsubscribe button)
- exploiting user mistakes (*e.g.* ads with arrow button)

## 11 History

### 11.1 Batch Interface

- punch cards prepared a priori
- response time of hours and days  $\Rightarrow$  no real interaction
- only highly trained users

### 11.2 Conversational Interface

- commands typed through keyboard
- wait for response (user cannot make changes during execution)
- heavily scripted interaction
- trained users

### 11.3 Graphical Interface

- input through keyboard, pointing device
- high resolution, refresh, graphics display
- user in control
- recognition memory over recall
- skeumorphism metaphors

Notable moments:

- *As We May Think*, Vannevar Bush
- Light Pen, Ivan Sutherland
- NLS Demo, Douglas Englebart
- Dynabook and Xerox Star, Alan Kay

## 12 Input

### 12.1 Classification

Sensing Method:

- mechanical (*e.g.* switch)
- motion (*e.g.* gyroscope)
- contact (*e.g.* capacitive touch)
- signal processing (*e.g.* computer vision)

Continuous v. Discrete

Degrees of freedom

### 12.2 Text Input

#### 12.2.1 QWERTY vs Dvorak

**Qwerty**

- designed to reduce typewriter jams and speed up typists
- not perfectly centered on home row

**Dvorak**

- letters typed with alternating hands, 70% on home row
- least common letters on bottom row
- right hand should do most of the typing
- **not faster** than QWERTY, no measurable difference

#### 12.2.2 Keyboard Design

**Mechanical Keyboards**

- fastest and most comfortable
- downsizing can pose difficult
- 80+ WPM

**Soft/Virtual Keyboards**

- ergonomic problems
- good when text input is limited
- 45 WPM

**Thumb/One-handed Keyboards**

- interesting but not mainstream
- “chording keyboards” are a thing
- 60 WPM

#### **Text Recognition and Gestures**

- Graffiti: map single strokes to enter letter (9 WPM)
- Natural handwriting recognition (33 WPM)

#### **Predictive Text**

- using characteristics of language to speed up typing
- T9: was the shit back in middle school
- 45 WPM for experts

#### **Gestural Text Input**

- “Swiping” across a keyboard to form words
- 30 WPM

### **12.3 Position Input**

- force vs displacement: joystick vs mouse
- position vs rate control
- absolute vs relative position: touchscreen vs mouse
- direct vs indirect contact: touchscreen vs mouse
  - Control-Display Gain:  $\frac{V_{pointer}}{V_{device}}$
- dimensions sensed: dial vs mouse vs wiimote

### **12.4 Gestural Input**

Gestures map movements to commands *e.g.* Myo gestures

## **13 Input Performance**

We want to measure which user interfaces are better using good metrics



### 13.1 Keystroke Level Model

Describe a task combining the length of time it takes for operators:

- keystroke: 0.08 - 1.2s
- pointing: 1.1s
- button press on mouse: 0.1s
- move hand mouse to/from keyboard: 0.4s
- mental preparation: 1.2s

Pros:

- easy to model
- can be done from just ideas and mockups

Cons:

- time estimates are inaccurate
- doesn't model learning, expertise
- doesn't model pointing well

### 13.2 Fitt's Law

A predictive model for pointing time based on device, distance, target size

$$MT = a + b \log_2\left(\frac{D}{W} + 1\right)$$

MT movement time

D distance from starting to middle of target

W constraining size of target, usually min(width, weight)

a,b characteristics of the pointing device

IP index of performance:  $1/b$

ID index of difficulty:  $\log_2\left(\frac{D}{W} + 1\right)$

making a cursor move slower over an object enlarges in in **motor space**, even though it looks the same size in screen space

### 13.3 Steering Law

An adaptation of Fitt's Law for steering between two goals:

$$ID = \log_2\left(\frac{A}{W} + 1\right)$$

A distance between goals

W width of goals

expanding this to make a tunnel:

$$ID = b \frac{A}{W}$$

## 14 Visual Perception

### 14.1 Temporal Resolution

People can only detect flickers up to 45 Hz after which it changes to continuous light. In the same way, pictures at 24 FPS become a movie

### 14.2 Spatial Resolution

High resolution only applies to 1% of photoreceptors in the eye. The distance from the screen necessary to distinguish a pixel  $d$ , can be stated as :

$$d = \frac{\text{size of a pixel}}{\tan(1/30)}$$

So if you are a distance of .4 - .7m away from the screen, pixels need to be 0.23 - 0.41cm apart to be distinguishable

### 14.3 Color Perception

**Color Models:** bases of colors used for displays

- RGB: used on displays
- HSB: hue, saturation, brightness
- YUV: optimized for human perception
- CMY(K): used in printing (subtracive)

**Display Monitors**

- each pixel is composed of 3 sub-pixels: red, green, blue
- vary intensity of each subpixel and use visual acuity restrictions to make a single color

## 14.4 Peripheral Vision

Low resolution vision in our periphery that helps

- guide focus
- detect motion
- see better in the dark

We can use this knowledge to direct user's attention, *e.g.*

- “wiggle” window
- reserve red for errors

## 15 Typography

Make reading as fast and clear as possible

### 15.1 Definitions

#### 15.1.1 General

**Typeface** : set of letters and numbers that make up a type design

**Font** : one width, weight, and style of typeface

**Glyph** : lowest divisible unit of type (a letter)

**Point** : 0.351mm

**Pixel** : size of one “dot” a screen

#### 15.1.2 Printing Terminology

**Baseline** : line on which most letters rest

**Point size** : total height of a typeface block

**Leading** : spacing between lines of text

**Line spacing** : leading + point size

**Ligature** : pair of letters replaced by a single printed unit (*e.g.* fi)

#### 15.1.3 Font Terminology

**x-height** : height of x

**em-space** : width of m

**Drop cap** : capital letter at start the descends multiple lines (like in manuscripts)

**Serif** : decorative stroke at end of main letter stroke

## 15.2 Computer Typography

**Bitmap font:** handcrafted font for bitmap display

- can't be properly scaled
- limited resolution

**Outline font:** vector outline of font, converted to bitmap prior to display

- compact storage
- bitmap representation can be inaccurate on small display

## 15.3 Design Rules

Consider different distances:

- far away
  - should look like a uniform display
  - headings should stand out
  - whitespace should guide
- nearby
  - distinct lines of text should be visible
  - short lines of text
- reading distance
  - words should be chunked
  - similar fonts, distinct fonts for different meanings

## 16 Accessibility

### 16.1 Temporary Disability

We all have “temporary disabilities”: distraction, injury, other focus. Just walking severely inhibits our ability to read, understand, and think about cognitive tasks.

### 16.2 Inclusive Design

1. Increases your audience:
  - 10 - 20% of population has a disability
2. Usually benefits everyone:
  - curb-cuts: made for wheelchairs to roll from curb to street but used by cyclists, strollers
  - cassette tapes: alternative for reel-to-reel tape so blind people could use it better

- closed captioning: useful for data mining videos and can help learn foreign languages
3. Required by law and is a basic precept for the internet:
    - use alt-text
    - use device-independent js events (`onSelect`)

## 17 Touch Interfaces

### 17.1 Display

- Resistive: two transparent layers that complete a circuit when pressed
- Capacitive: measures change in capacitance to find location of touch
  - single-touch: capacitors at four corners
  - multi-touch: grid of capacitors with a layer of driving lines to carry current, and sensing lines to detect it
- Inductive: use a magnetized stylus to induce EM field in back of display
- Optical: use motion-tracking cameras

### 17.2 Input Challenges

1. fat finger problem: finger occludes the object, can make object bigger or account for finger occlusion with touch offset
2. ambiguous feedback: touch interfaces can miss haptic feedback and confuse user if action unsuccessful
3. no hover state: no way to preview your actions before you do them
4. multi-touch capture: multiple fingers can lead to ambiguous input

### 17.3 Interaction

Since we have a touch screen, our interface is of **direct manipulation**:

- dragging a document to the trash
- dialing a phone number on virtual keys

#### 17.3.1 Principles

- visible and continuous representation of objects and actions
- all actions are valid, reversible, obvious
- interaction with object as opposed to interface

### 17.3.2 Challenges

- potentially not accessible
- analogies may not be clear

## 17.4 Design

Help users:

- enter information quickly
- know what action to take
- utilize real estate and avoid clutter

## 17.5 Implementation

Touch Event API:

- touchstart
- touchmove
- touchend

# 18 Touchless Interfaces

## 18.1 Introduction

Interfaces either through voice, air gestures, ... are extremely useful but suffer from errors due to being one-state input devices. The key idea is that the system should fail gracefully.

## 18.2 In-Air Gestures

**Live Mic Problem:** because touchless interfaces can be “always on”, we need to find a way to distinguish commands from unintentional gestures (*e.g.* scratching your head).

- reserved actions: making a particular set of actions always used for navigation or commands
- reserving a clutch: making a rare action be the indicator to start/stop a command
- multi-modal input: use a different mode (*e.g.* hardware button) to control universal params

## 18.3 Speech Interfaces

Voice recognition challenges:

- **Rejection error:** system has no hypothesis about what user has said
- **Substitution error:** system mistakes utterance
- **Insertion error:** recognize noise as legal utterance

Although voice recognition seems like a good metric, user satisfaction is better measured with:

- discourse segment pop: completing a sub-dialog and revealing context
- human conversability: understanding casual speech
- dialog boxes: asking for confirmation at right time

## 19 Information Visualization

### 19.1 Why

Good data visualization allows people to make better decisions and be more informed.

### 19.2 What

**Visual encoding** is the process of encoding the data into a visual format, consists of:

- graphical elements called **marks**
- **visual channels** which control the appearance of marks

#### 19.2.1 Channel Types

**Magnitude channels:** for organizing ordered attributes

- position on a scale
- area/volume
- color saturation

**Identity channels:** for organizing categorical attributes

- spatial region
- color hue
- shape

### 19.2.2 Channel Effectiveness

We can measure how effective a channel is with:

- accuracy
- discriminability
- separability
- popout

## 19.3 How

Use software structure diagrams to describe the models

- **Reference Model Pattern:** and separate them from the views to enable multiple views of a visualization
- **Data Column Pattern:** organize data into columns to provide flexible representations and schemes
- **Operator:** decompose visual data processing into composable operators
- **Renderer:** separate visual components from their rendering methods for dynamic rendering
- **Production Rule:** use if-then-else to dynamically determine visual properties