How to Write Fast Code

18-645, spring 2008
16th Lecture, Mar. 17th

Instructor: Markus Püschel

TAs: Srinivas Chellappa (Vas) and Frédéric de Mesmay (Fred)

Today

- **Guide to benchmarking and making nice plots**
- **Starting on transforms**

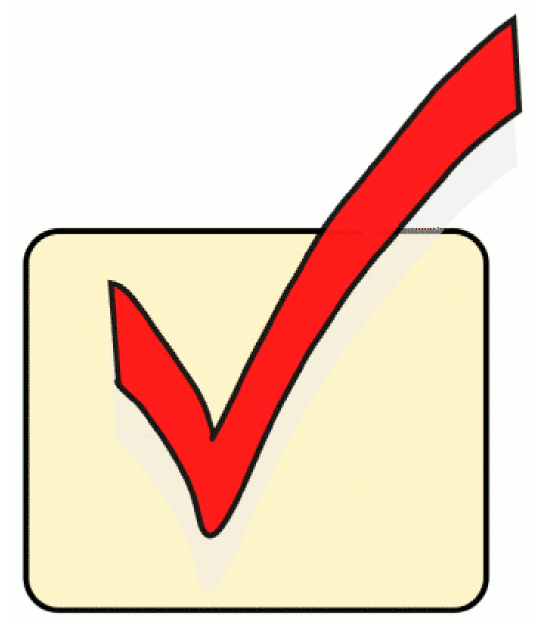
- **Rough plan for the next lectures**
  - Next “homework” is working on project
  - Transforms and filters (same as: correlation, interpolation, stencil, polynomial multiplication)
  - Another round of one-on-one meetings
  - Shared memory parallelization, other functionality, advanced topics
  - Discuss project presentations

Benchmarking

- **Before you start**
- **Type 1: Evaluation of the performance of your code**
  (no external competitor)
- **Type 2: Comparisons against other code**
  (you want to show your code is better)
- **Presenting your results (plots)**
  - In writing
  - Talking
  - Making nice plots

Before You Start

- **Verify your code!**
  - And that very carefully
  - It is utterly embarrassing to publish or present meaningless results



Evaluating Your Own Code

- **Measure**
  - Runtime
  - Performance (floating point cost by analysis or instrumenting your code)
  - Percentage of peak

- **Make sure you use your compiler properly**
  - Optimization flags (e.g., try -O2, -O3, specify platform if possible)
  - For compiler vectorization and written vector code see vector lecture

Comparison Against Other Code

- **Be fair!** 
  - Make sure the comparison is apples to apples
    - Your code computes exactly the same
    - Same interface (e.g., order of input array, data structures)
  - Compile other code properly (maybe specific flags are specified)
  - Use the same timing method
  - Always do a sanity check: compare to published results etc.
  - Apply obvious, easy optimizations also to the competitor code! (but say so when you report)
- **Compare against the fastest available code**
- **Report performance if possible**
  - But use same op count for computing (so it's inverse runtime)
  - Shows efficiency of code besides who is better
  - Yields higher is better plots (psychologically more intuitive)

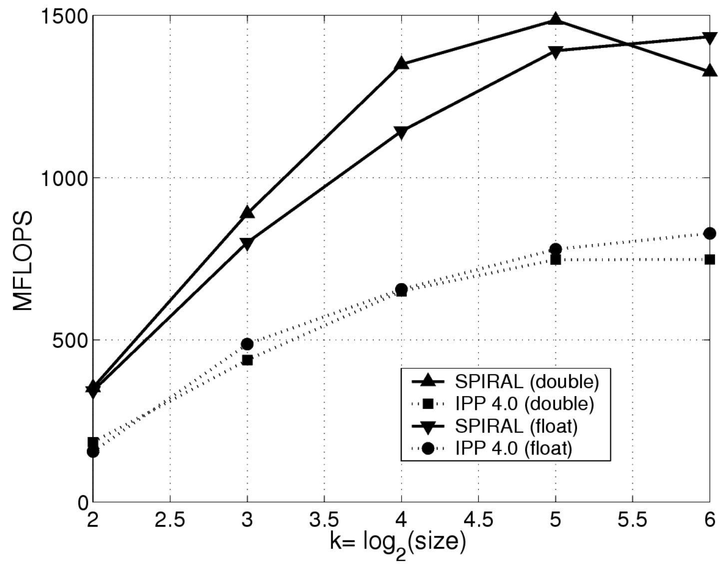
How to Present Results in Writing

- **Specify machine**
  - processor type, frequency
  - relevant caches and their sizes
  - operating system
- **Specify compilation**
  - compiler incl. version
  - flags
- **Explain timing method**
- **Plot**
  - Has to be very readable (colors, lines, fonts, etc.)
  - **Discuss interesting aspects of plots and extract a main message**
  - Choose proper type of plot: message as visible as possible

How to Present Results Talking

- Briefly explain the experiment
- Explain x- and y-axis
- Say, e.g., “higher is better” if appropriate
- Give an example: this line/point means that ….
- Discuss plot and extract a message in the end

Performance of the discrete cosine transform:



Message:
- Spiral code is 2x faster
- reaches up to 50% of peak

Platform:
P4 (HT), 3GHz, 8KB L1, 512KB L2, WinXP

Compiler:
icc 8.0

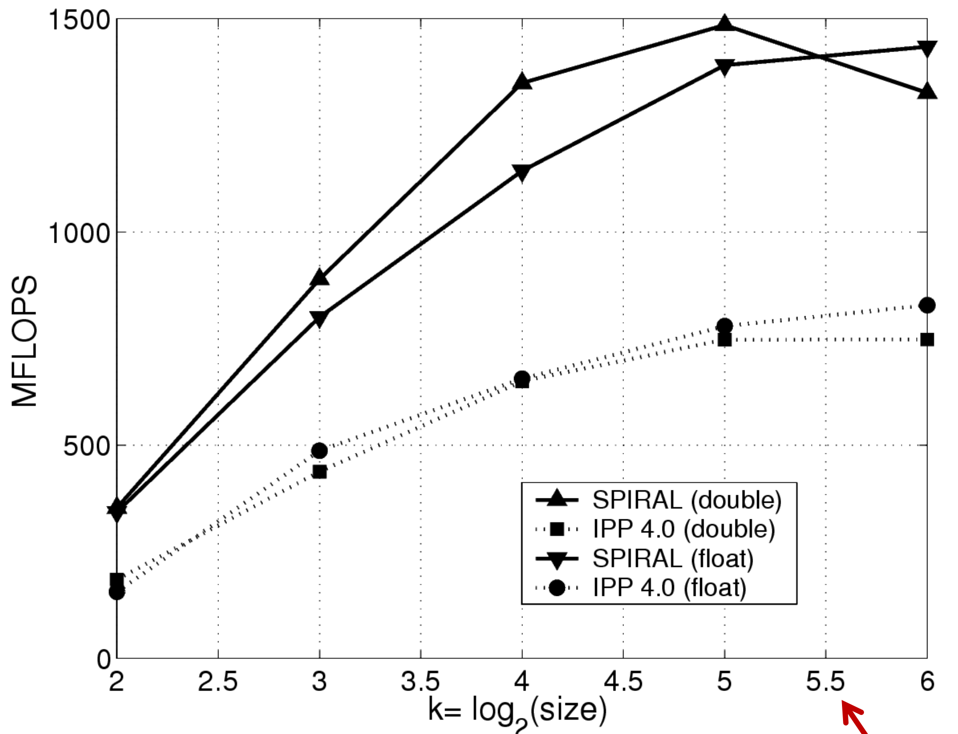
Compiler flags:
/QxKW /G7 /O3

Plots: The Basics

- **Very readable**
  - Title, x-label, y-label need to be there
  - Fonts large enough
  - Enough contrast line to background (e.g., no yellow on white please)
  - Enough difference between lines
  - Proper number format (where appropriate)
    - **No:** 13.254687; **yes:** 13.25
    - **No:** 2.0345e-05 s; **yes:** 20.3 µs
    - **No:** 100000 B; **maybe:** 100,000 B; **yes:** 100 KB
- **Clearly shows the message**
  - Proper type of plot (line, bars, properly ordered)
  - All the above
  - Check it: you know the message; does it jump in your face?
- **Beautiful**
  - Tough, but all the above makes it more beautiful, more later

Example: Mediocre Plot

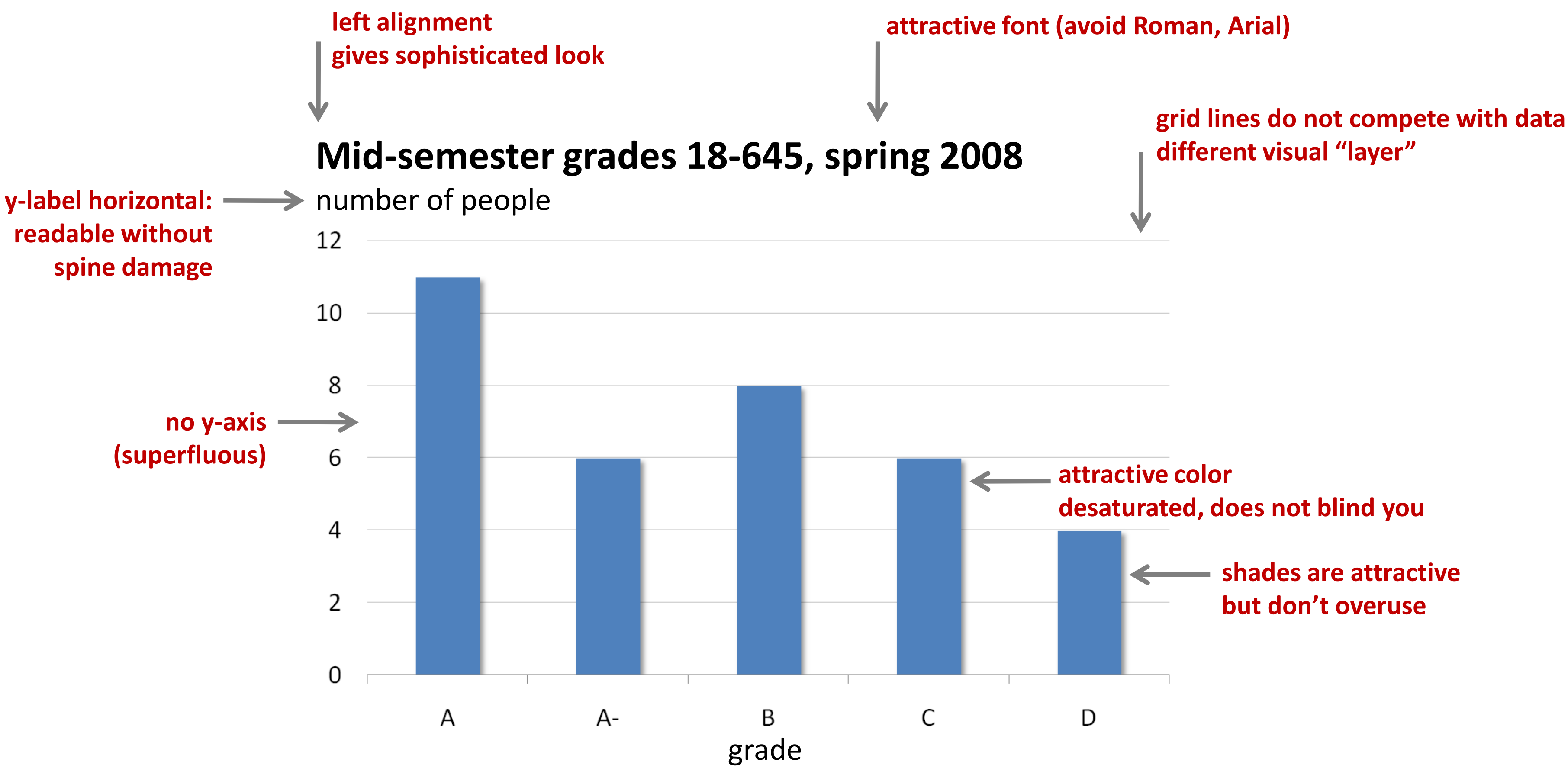
Well, 3 years ago I thought it is a good one ☺



Except for this

How do we make it better?

Example I: Good Plot

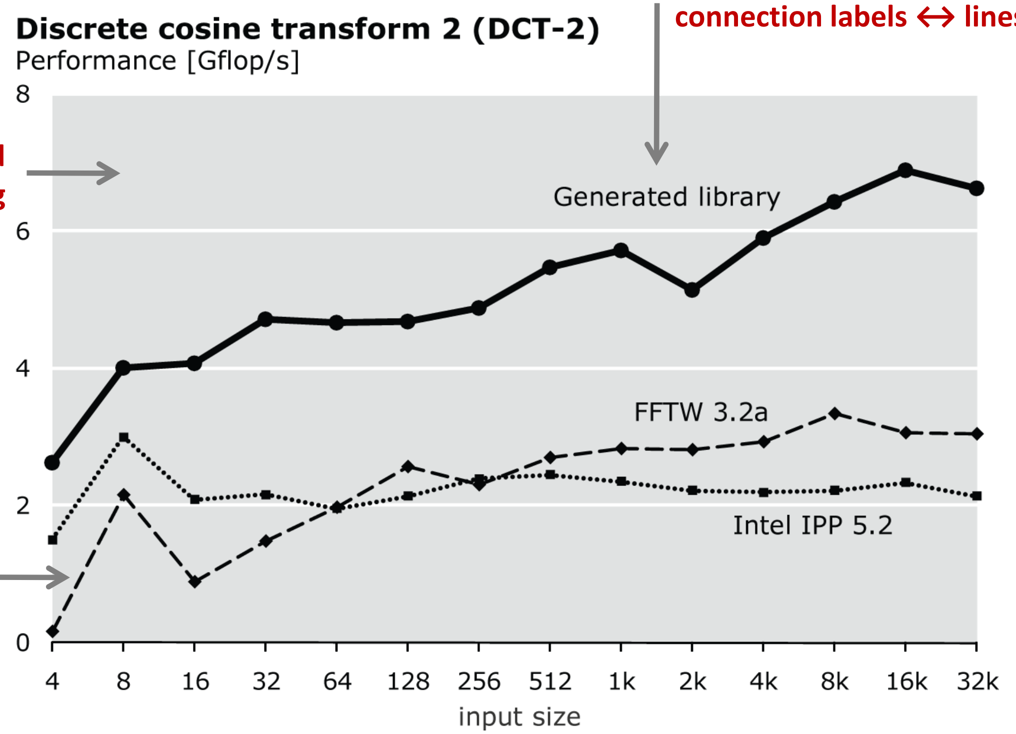


Example II: Good Plot

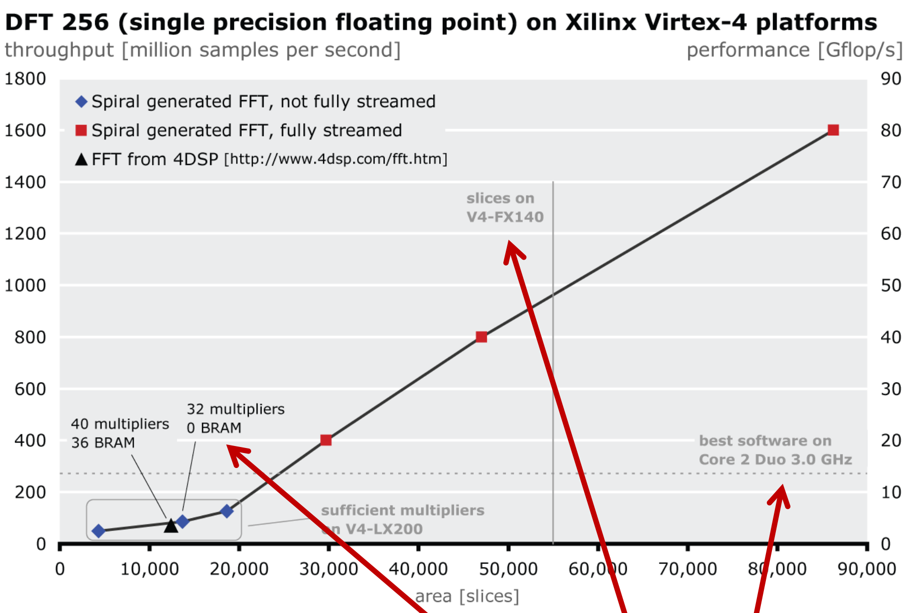
no legend
connection labels ↔ lines immediate

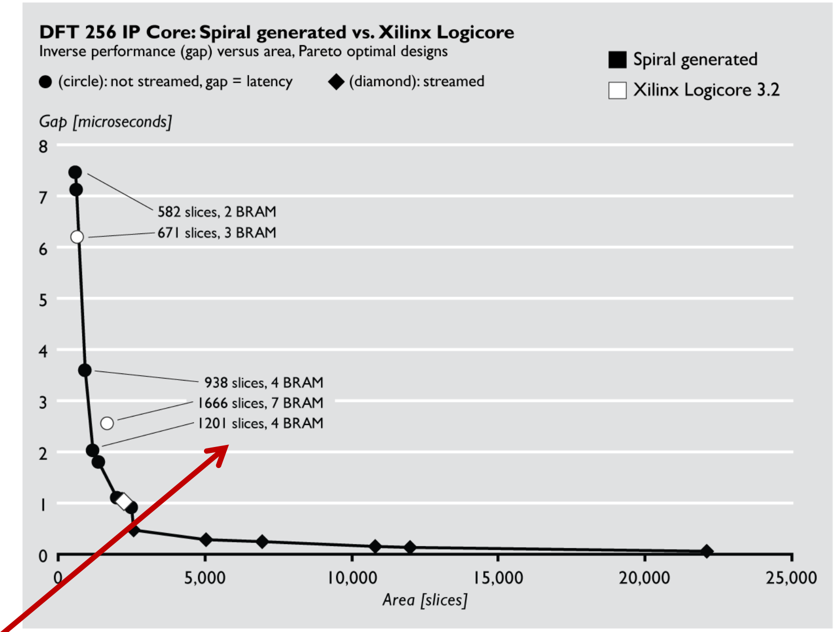
background/grid lines inverted
another way of layering

consistent line styles
to distinguish
(color is much better though)



Example III: Good Plots





- additional information can be packed into a plot
- but use different visual layers
- and make sure it is readable
- good for print publications or web (reader has time to study)

Good Plots: Advanced Principles

- **No Roman or other serif font, avoid Arial if possible**
  - Calibri (Office 2007)
  - Myriad
  - Verdana
  - Gill Sans

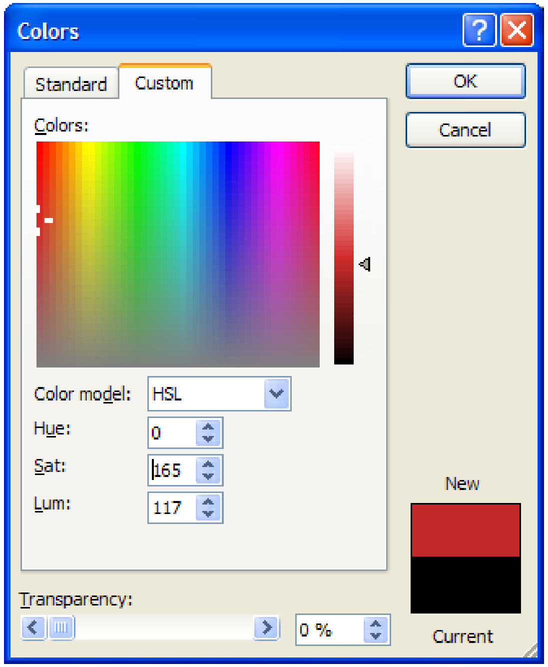
M serif

M sans serif

- **Layering**
  - Grid lines, axes, etc. should not compete with data lines for attention
  - More care necessary when more information is packed into plot
  - Good example for layering: maps
- **Alignment**
  - Title, horizontal y-label: left (general design principle)
  - x-label, vertical y-label: center

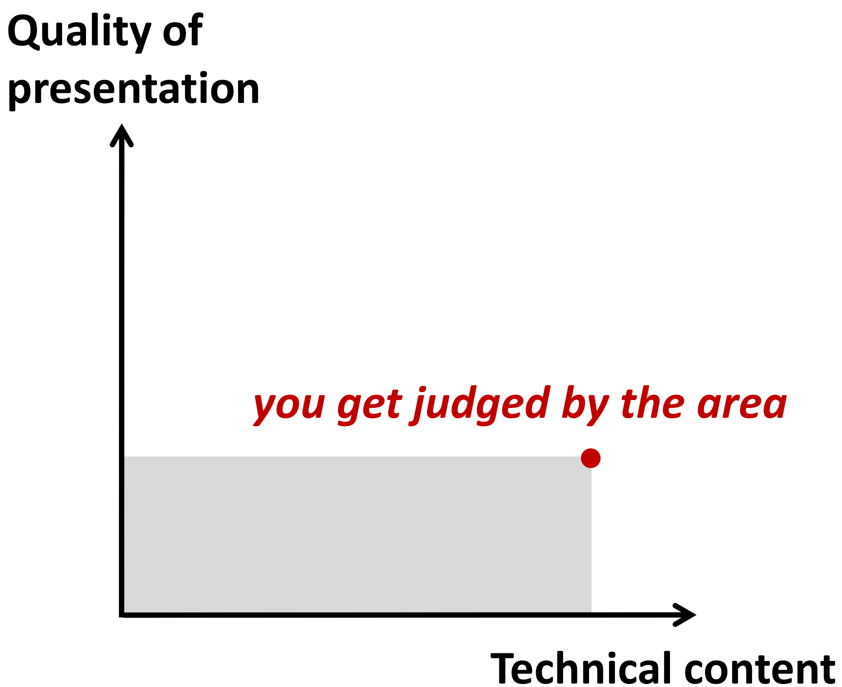
Good Plots: Advanced Principles

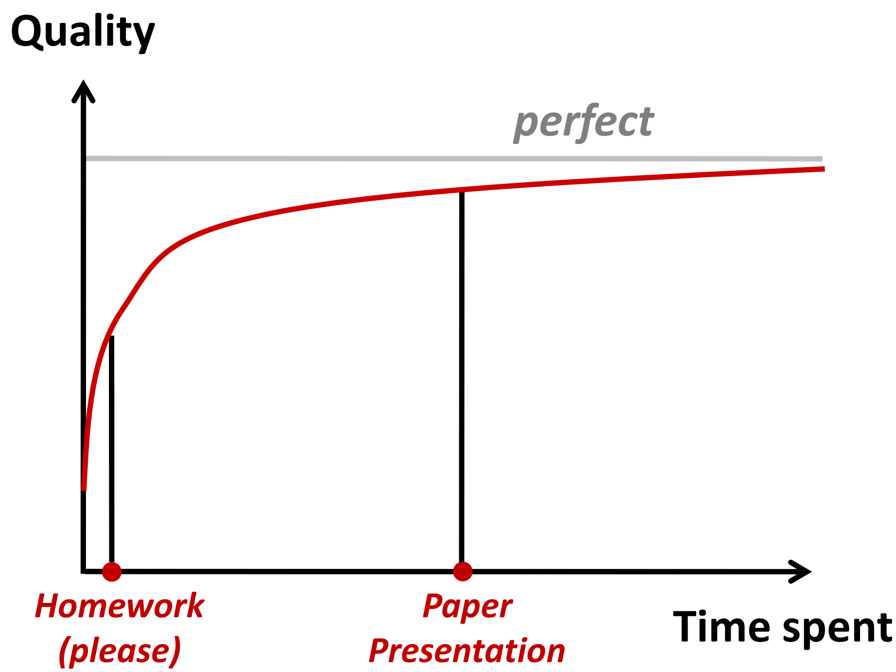
- **Colors**
  - Use them, except for most print publication
  - Don't use **fully saturated colors**
  - Use **somewhat desaturated colors**



- **Get rid of chart junk**
  - Maximize:
    (ink used on data)/(ink used on the rest)

Keep in Mind





Left plot based on a discussion with Jim Bain

Tools and More Information

- **Software for making plots**
  - Matlab (plots by default ugly, but totally configurable, scriptable)
  - Excel (2003: by default ugly but a little clicking …., get Office 2007!)
  - Gnuplot (totally configurable, scriptable, only for linux really)
  - For highest quality I use: Excel to get it roughly right, then copy-paste into Adobe Illustrator for fine-tuning (everything editable)

- **How to learn more**
  - Look how good magazines do it (Economist, National Geographic, NY Times, …)
  - Edward Tufte:
    - Visual display of quantitative information
    - Beautiful evidence
  - See also: Guide to making nice tables

Transforms

The Protagonists: Linear Transforms

- **Mathematically: Change of basis**
  Two "schools" of representation

$$y = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} \qquad y_k = \sum_{\ell=0}^{n-1} t_{k,\ell} x_\ell \qquad x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}$$

Summation formula

$$y = Tx \qquad T = [t_{k,\ell}]$$

Matrix-vector product

- **Used in signal processing, scientific computing, …**
- **Example: Discrete Fourier transform (DFT)**

$$\mathrm{DFT}_n = [e^{-2k\ell\pi i/n}]_{0\le k,\ell<n}$$

Transforms: Examples

- More than 30 transforms in the literature

$$\mathrm{DFT}_n = [e^{-2k\ell\pi i/n}]_{0\le k,\ell<n}$$

$$\mathrm{RDFT}_n = [r_{k\ell}]_{0\le k,\ell<n}, \quad r_{k\ell} = \begin{cases} \cos\frac{2\pi k\ell}{n}, & k \le \lfloor\frac{n}{2}\rfloor \\ -\sin\frac{2\pi k\ell}{n}, & k > \lfloor\frac{n}{2}\rfloor \end{cases}$$

universal tool

$$\mathrm{DHT} = \left[\cos(2k\ell\pi/n) + \sin(2k\ell\pi/n)\right]_{0\le k,\ell<n}$$

$$\mathrm{WHT}_n = \begin{bmatrix} \mathrm{WHT}_{n/2} & \mathrm{WHT}_{n/2} \\ \mathrm{WHT}_{n/2} & -\mathrm{WHT}_{n/2} \end{bmatrix}, \quad \mathrm{WHT}_2 = \mathrm{DFT}_2$$

$$\mathrm{IMDCT}_n = \left[\cos((2k+1)(2\ell+1+n)\pi/4n)\right]_{0\le k<2n, 0\le\ell<n}$$

MPEG

$$\text{DCT-2}_n = \left[\cos(k(2\ell+1)\pi/2n)\right]_{0\le k,\ell<n}$$

JPEG

$$\text{DCT-3}_n = \text{DCT-2}_n^T \quad \text{(transpose)}$$

$$\text{DCT-4}_n = \left[\cos((2k+1)(2\ell+1)\pi/4n)\right]_{0\le k,\ell<n}$$

Fast Transform Algorithms

- **Reduce runtime from $O(n^2)$ to $O(n \log(n))$**
- **> 200 publications on transform algorithms**
- **Example: Cooley-Tukey fast Fourier transform (FFT)**

Again two schools:

$$y_{n_2 j_1 + j_2} = \sum_{k_1=0}^{n_1-1} \left(\omega_n^{j_2 k_1}\right) \left(\sum_{k_2=0}^{n_2-1} x_{n_1 k_2 + k_1} \omega_{n_2}^{j_2 k_2}\right) \omega_{n_1}^{j_1 k_1}$$

sequence of summations

$$\mathrm{DFT}_n = L_{n_2}^{n} (I_{n_1} \otimes \mathrm{DFT}_{n_2}) T_{n_1}^{n} (\mathrm{DFT}_{n_1} \otimes I_{n_2})$$

matrix factorization

DCT, type III

Algorithm derivation

Fast implementation of this algorithm: next homework

Just kidding ☺

Typical derivation (> 200 such papers)

Recent research at CMU:

- **Simplifies derivation**
- **Explains origin**

G. Bi "Fast Algorithms for the Type-III DCT of Composite Sequence Lengths" IEEE Trans. SP 47(7) 1999

Discrete Fourier Transform (DFT)

- Blackboard