Introduction to Digital and Computational Photography

Fredo Durand
MIT EECS
6.815/6.865
The unfinished revolution

• Traditional photography:
  - optics focuses optical array onto sensor
  - chemistry records final image

• Digital photography
  - optics focuses optical array onto sensor
  - digital sensor records final image
Computational Photography

- Arbitrary computation between the optical array and the final image
- Data recorded by sensor is not the final image
Computational Photography

- Arbitrary computation between optical array and final image (or final medium)
- Post-process after traditional imaging
  - a.k.a. image processing (maybe more interactive)
  - But also combine multiple images to overcome limits of traditional imaging (HDR, panorama)
- Design imaging architecture together with computation
  - Computational cameras, computational illumination, coded imaging, data-rich imaging
- Extract more than just 2D images
- New media (panorama, photo tourism)
Comp Photo
@MIT
Tone mapping

• One of your assignments!
Lattice focal lens

- Maximize depth of field
- All depths are blurry
- Computation to decode
- Co design of lens & computation
Input image
Hallucination at different time

Day

Blue hour

Golden hour

Night
Synthetic time-lapse
Revealing Invisible Changes In The World

Created for the NSF International Science & Engineering Visualization Challenge 2012
The class
Staff

- Instructor: Frédo Durand
- TAs: Neal Wadhwa, Adrian Dalca, Katie Bouman, Gaurav Chaurasia
- Secretary: Bryt Bradley
- Office hours
  - Fredo: Thursdays 4pm (tentative)
  - TAs:
    - Tuesday & Wednesday 5pm (tentative)
- Staff mailing list: 6815-staff@lists.csail.mit.edu
  - But use Piazza to ask pset questions
Practical details

- **Stellar**
  - slides, homework, etc.

- **Piazza**
  - https://piazza.com/class/i5o90nv8ogu4z
  - Won’t answer technical questions by email
  - Can be anonymous if you’re shy
  - Sign up!

- **No textbook**
Grading

• 95% of the grade: assignments
• 5%: participation in class
• 6.865: extra questions + 1 paper review
Assignments

• Every week
  - Except maybe two 2-week ones
  - Due Wednesdays at 9pm.
  - Deadline is absolute

• Mostly programming
  - C++

• Turn in code, images, readme

• Collaboration policy:
  - Chatting is encouraged
  - Code must be written alone
Policies

• Collaboration
  - You are welcome and encouraged to chat about assignments
  - Code must be written on your own

• Late
  - Homework must be submitted on time
    - I mean it: you get zero if you’re late
  - Modeled after the SIGGRAPH submission policy
    - If you’re even 5 minutes late, you’re late
    - If you upload the wrong file, too bad
  - A dean note is required to get an extension for medical reasons
  - Special circumstances: ask one week in advance
Assignments turn in

- New magical submission system
  - Put all in a zip file
- Readme (parsable)
  - How long it took
  - Potential issues with your solution and explanation of partial completion (for partial credit)
  - Collaboration acknowledgement (but again, you must write your own code)
  - What was most unclear/difficult
  - What was most exciting
- Source code (always!)
- Image results (sometimes)
Grading experiment

• Second iteration, first time with C++
• We will grade (semi) automatically
  - unit tests per question/function
  - cluster by output
  - all members of a cluster get the same score/comment
• The submission system will test that your code compiles
  - but submit early to avoid overloading the system.
  - remember: the deadline is absolute
• There will be an appeal process if you feel there is a mistake with the grading
Textbook (lack thereof)

• No textbook required
• Lots of resources on the net
• Siggraph course notes
  - http://web.media.mit.edu/~raskar/photo/
• Will post lectures slides
• Links to articles in slides
Camera

• You will need to take a photo or two
  - In particular for the panorama assignment
• No need for a fancy SLR
Questions?
Introductions

• Who are you?
• What do you know about photography?
• Why you want to take this class?
• Course VI?
• know C, C++, java
• 6.005, 6.004
• avid photographer
• wants to work in imaging
• has ever used a camera
• retouched images
• stitched a pano? time lapse?
• HDR
• know f number? change ISO?
What do you think you will learn
What the class is not about

- Little about art, photographers
- Little about EE & optics (sensors, A/D, etc)
- Not how to use Photoshop
  - But how its coolest tools work
- Not much about 3D imaging
- Not too much fundamentals of signal processing
- Not much computational imaging, no tomography, no radar, no microscopy
- Not much computer vision, computer graphics
- We avoided overlap with 6.837 and 6.801/6.866
What the class is about

• Software aspects of computational photography
  - but a bit of hardware as well, lens technology, new camera designs

• Basic tools
  - Linear & non-linear image processing, color

• Emphasis on applications
  - High-dynamic range photography, photomontage, panoramas, foreground extraction, inpainting, morphing

• Emphasis on recent research results
Skills you will acquire

• Implementation of basic tools
  - Color demosaicing
  - Bilateral filter, tone mapping
  - Gradient reconstruction
  - Features
  - Panorama stitching

• General approaches to computational photography

• Important problems in computational photography
Non-Photo motivation

• It's about any kind of data!
  - Speech, motion, geometry, etc.
  - Example:
    - Music
    - Motion graphs
    - Poisson mesh editing
    - BTF shop

• Lots of fundamental numerical tools
Questions?
Topics include 3D printing, solid modeling, geometry processing, simulation, 3D scanning, latest research (a lot of computer graphics)

AUS; satisfies Graphics and HCI concentration

Labs in the new Engineering Design Studio

Open-ended class projects

No final

TR 2:30-4pm, 32-144
Syllabus (tentative)
Syllabus

- Color and color perception
- Demosaicing
Syllabus

• High Dynamic Range Imaging
• Bilateral filtering and HDR display
• Matting
Syllabus

- Panoramic imaging
- Morphing
Syllabus

- Gradient image manipulation
Syllabus

- Future cameras
- Plenoptic function and light fields
Syllabus

- Fourier transform and deblurring
New this year: Halide & performance

- Make parallelization, vectorization, tiling, kernel fusion, etc. easy
- Decouple the algorithm from the optimization/schedule

Porting to new platforms does not change the algorithm code, only the schedule
Good pictures

http://fredodurand.net/
Assignments
Basics

• Brightness, contrast, black and white
• Color spaces, saturation
• Spanish Castle illusion  http://www.johnsadowski.com/big_spanish_castle.php
• Histograms
Resampling & Morphing

- Image rescaling & rotation
- Morphing one face into another one
convolution & denoising

- Blur, unsharp mask

- Denoising with the bilateral filter
Demosaicking
HDR imaging & tone mapping

Before

After
Panoramas

- Manual pano
- Automatic pano using features
Light fields & depth of field
Halide & performance

- Make parallelization, vectorization, tiling, kernel fusion, etc. easy
- Decouple the algorithm from the optimization/schedule

Porting to new platforms does not change the algorithm code, only the schedule

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Platform</th>
<th>Lines</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimized NEON ASM:</td>
<td>Nokia N900:</td>
<td>463</td>
<td>772 ms</td>
</tr>
<tr>
<td>C++, OpenMP+IPP:</td>
<td>Quad-core x86:</td>
<td>262</td>
<td>627 ms</td>
</tr>
<tr>
<td>Tuned C++:</td>
<td>Quad-core x86:</td>
<td>122</td>
<td>472 ms</td>
</tr>
<tr>
<td>Vectorized MATLAB:</td>
<td>Quad-core x86:</td>
<td>67</td>
<td>3800 ms</td>
</tr>
<tr>
<td>Halide algorithm:</td>
<td>Nokia N900:</td>
<td>145</td>
<td>741 ms</td>
</tr>
<tr>
<td>schedule:</td>
<td>Quad-core x86:</td>
<td>62</td>
<td>293 ms</td>
</tr>
<tr>
<td></td>
<td>Quad-core x86:</td>
<td>34</td>
<td>80 ms</td>
</tr>
<tr>
<td></td>
<td>Quad-core x86:</td>
<td>148</td>
<td>55 ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.75x shorter</td>
<td>5% faster than tuned assembly</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.7x shorter</td>
<td>2.1x faster</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3x shorter</td>
<td>5.9x faster</td>
</tr>
<tr>
<td></td>
<td></td>
<td>70x faster</td>
<td></td>
</tr>
</tbody>
</table>

Quad-core x86: 51 ms           CUDA GPU: 48 ms (13x)           CUDA GPU: 11 ms (42x)           CUDA GPU: 3 ms (1267x)
C++
Why C++?

- More efficient than java (because of compilation, memory)
  - ridiculously more efficient than python
- Still standard for many areas where performance matters (graphics, imaging)
- Allows us to use Halide (even more efficient)
- Good experience
Resources

• http://cs.brown.edu/courses/cs123/docs/java_to_cpp.shtml
• http://www.cprogramming.com/java/c-and-c+-for-java-programmers.html
• http://www.horstmann.com/ccj2/ccjapp3.html
• http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.132.6953&rep=rep1&type=pdf
C++ for java programmers

• Pretty similar.
  A little more clunky, less safe, but more efficient

- Fully compiled, 2 types of files, makefile
- Memory management
- Pointers, references, value
- Less class-centric, no interface, templates
- Minor syntactic differences
- Different standard libraries
Files & Compilation (easy)

• no VM, compiled directly to binary

• 2 types of source files:
  - header (.h) contains class and function declaration
  - program (.cpp) contains implementation
    - exception: “inlined” methods implemented in .h

• “#include” only header files (not “import”)
  - need for stupid #ifndef trick at the top of .h to avoid including files twice

• Use “make” on terminal to compile a project
  - makefile describes the build process
  - clunky and annoying
Life outside of classes

- global variables
- functions
Classes

• implementation of methods (.cpp) separate from declaration (.h)
• public and private *sections*
  - no need to say for each method
• constructor: class name
• destructor
  - called ~
  - always called for stack objects
  - but not heap/pointers, do it yourself
Syntax

• Variables shouldn’t start with _ or __
• Scope operator :: in implementation file to say which class you’re implementing:
  - MyClass::myFunction
  - because .cpp files could deal with multiple classes
• semicolon at the end of class definition
  - otherwise error in next file!
Value, reference, pointer (hardish)

- int myFunction(MyClass myObject)
  - value: copies the object (UNLIKE JAVA)
- int myFunction(MyClass *myObject)
  - pointer: just passes the memory address
- int myFunction(MyClass &myObject)
  - reference: modifiable version of the original object
  - not copied in memory
  - called the same as passing by value

- Pointers & references are mostly similar
  - But we will avoid pointers as much as possible
const keyword

• Declare a constant
  - const int size = 5

• For class method declaration, says that it won't mutate the object (e.g. getters)
  - int width() const;

• In function parameter:
  parameter won't be modified
  - int myFunction(const MyClass &myObject);
  - that's the way we pass most parameters
  - means the method will only be allowed to use const methods
Memory 1 (not too bad)

- Explicit management
  - `new, delete`, no garbage collector
  - but in this class, we won’t worry too much about memory leaks

- Stack object
  - created without `new`
  - `MyClass myObject();`
    unlike Java, this creates an object
    (on the stack, not heap)
Memory 2: unsafe *(very painful)*

- **Unsafe arrays**
  - `myArray[size+100]+=1;`
    will compile, run, and modify an arbitrary part of memory
  - will often result in a segmentation fault

- **Uninitialized pointer (or why we avoid them)**
  - `MyClass *myObject;`
  - `myObject->doSomething()`
  - will try to run the doSomething method on the arbitrary address pointed to by the initialized pointer

- In both will often **crash at a completely different place than where the problem is**
Image.h

```cpp
#ifndef __IMAGE_H
#define __IMAGE_H
#include "lodepng.h"
#include <iostream>
#include <vector>
#include <string>
#include "ImageException.h"

class Image {
public:

    Image(int x, int y = 0, int z = 0, const std::string &name="" );
    Image(const std::string &filename);
    // Image(const Image &im); //copy constructor
    ~Image();

    const std::string & name() const { return image_name; }
    int dimensions() const { return dims; }

    int stride(int dim) const;
    int width() const { return dim_values[0]; } //Extent of dimension 0
    int height() const { return dim_values[1]; } //Extent of dimension 1
    int channels() const {return dim_values[2];} //Extent of dimension 2
    int extent(int dim) const { return dim_values[dim]; }

    void write(const std::string &filename);

    // Accessors of the pixel values
    const float & operator()(int x) const;
    const float & operator()(int x, int y) const;
    const float & operator()(int x, int y, int z) const;

    float & operator()(int x);
    float & operator()(int x, int y);
    float & operator()(int x, int y, int z);

    long long number_of_pixels() const;

private:
    unsigned int dims;
    unsigned int dim_values[3];
    unsigned int stride_0, stride_1, stride_2;
    std::string image_name;
    std::vector<float> image_data;
    double gamma;
};
```

Image.cpp

```cpp
#include "Image.h"

// All functions students need to implement are here at the top

// We
const float & Image::operator()(int x) const {
    return the pixel at location x, where x spans all locations and color channels.
    // throw OutOfBoundsException if x is negative or larger than the number of pixels
}

// Remember to do bounds checking
const float & Image::operator()(int x, int y) const {
    // For this one, we return x,y in the first channel and throw out of bounds other channels.
}

const float & Image::operator()(int x, int y, int z) const {
    // For this one, we return x,y,c in the first channel and throw out of bounds other channels.
}

// The next three functions should have the same implementation as the previous three.
float & Image::operator()(int x) {
}

float & Image::operator()(int x, int y) {
}

float & Image::operator()(int x, int y, int z) {
}

// Return the total number of floats stored in the array.
// There are a number of ways to implement this.
// You can look at the size of image_data or use the extent function
long long Image::number_of_pixels() const {
}
```
recap: hard parts

- Errors not always thrown where the problem is:
  - forget ; at the end of class definition, error in next file
  - out of bound write or initialized pointer corrupt memory and could crash at any time.
  - When you get segmentation fault, look for array or pointer operations.
    (but hopefully we shouldn’t use pointers)
Development

• Use either the terminal or your favorite IDE
  - xcode, visual studio, eclipse
  - Just make sure things compile on our end!

• Write images to disk
  - view with your favorite viewer

• Use lots of print statements, assert, image write
  - Choose good debugging inputs

• Google is your friend, especially for compiler errors and C++ syntax
Images in C++
Digital Images

• Can be encoded as 2D arrays of RGB triplets
• Formally:
  - The domain is the 2D plane
  - The range is the RGB space
• Other color spaces are possible and will be used
• Values often encoded as 8 or 16 bit integers
  - But we will use floats in [0...1] to make life easier
Arrays

- **C++ vector**
  - dynamically sized
  - templatized by type, float in our case

- **Created with**
  ```
  data = std::vector<float>(size,initialValue);
  ```
  - for a float array with an initial value

- **References:**
1D to 3D

- vectors only have one 1D index
- turn 2D index into 1D one through strides
  - pixel at x, y, stored at y* width + x
  - width is called the “stride”

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

2D array

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
</table>

1D vector encoding

- For 3D index x, y, c where c is the color channel:
  - y* width* numChannels + x* numChannels + c
PSet 0
Pset 0

- Just a warm up
- familiarize yourself with C++
- compile
- change brightness & contrast of an image