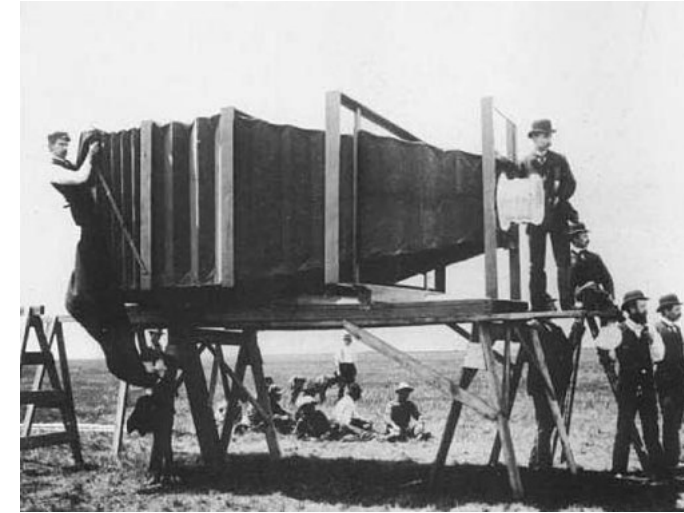


Digital and Computational Photography

Frédo Durand
MIT EECS
6.8370/1



- **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

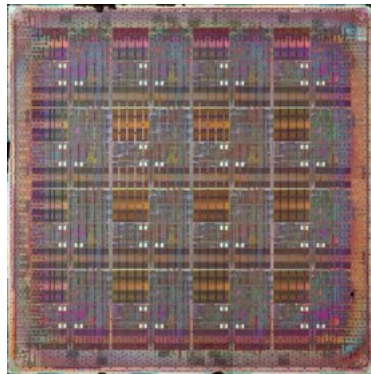


Computational Photography

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



Generalized imaging



Lots of computation



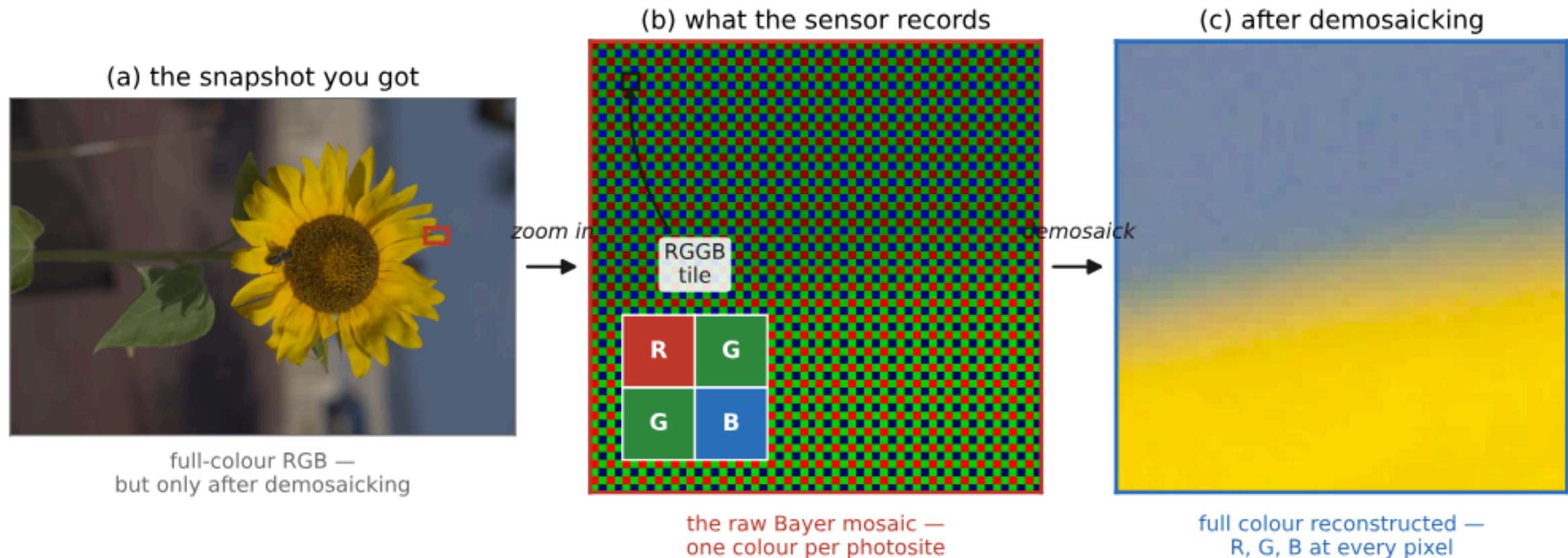
Final image

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)

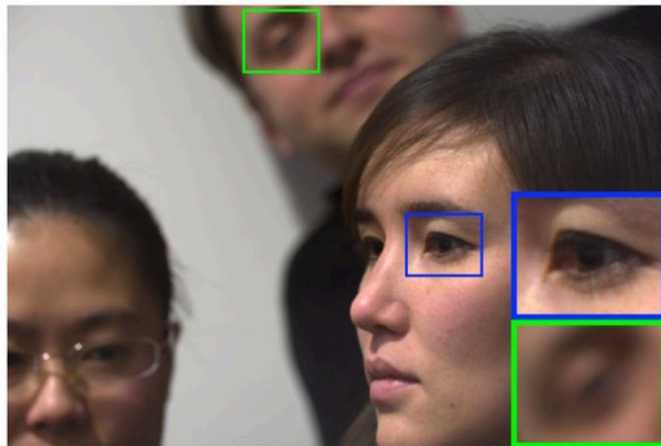
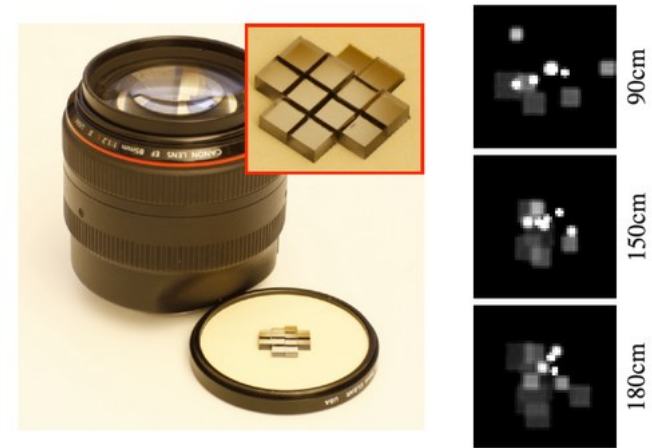
Computation was involved since the first day

Even a plain snapshot needs computation: the sensor records one colour per pixel; the photo exists only after demosaicking

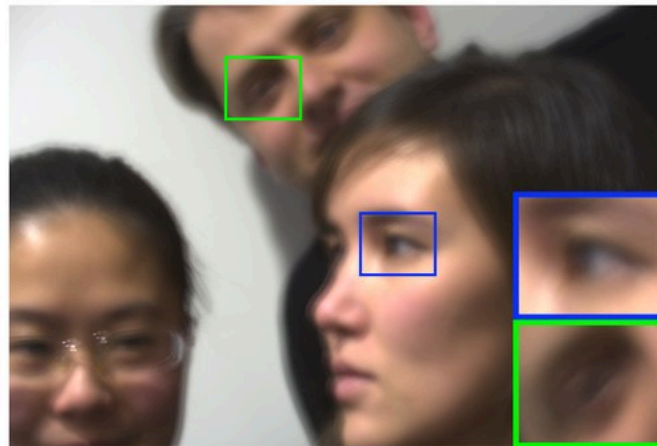


Lattice focal lens

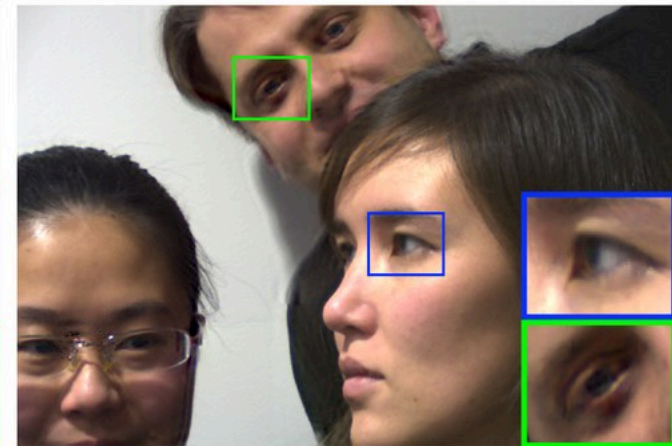
- Maximize depth of field
- All depths are blurry
- Computation to decode
- Co design of lens & computation



Standard lens image



Our lattice-focal lens: input



Lattice-focal lens: all-focused output

Goals of computational photography

- **Prettier pictures:** straightforwardly better-looking results: cleaner, sharper, better-toned, more pleasing color.
- **Alleviate physical limitations:** beat the constraints the hardware imposes: too little light, too little dynamic range, a lens too small or too shaky.
- **Record more information:** capture what an ordinary photograph discards: depth, scene structure, separate layers of the scene.
- **More control after capture:** give the photographer more control after capture rather than locking in every decision at the moment of the shot.
- **New visual experiences:** make images that simply have no single-shot equivalent, like a refocusable photograph or a navigable panorama.
- **Reveal the invisible:** surface what the eye cannot see on its own, from tiny motions and color changes to information outside the visible band.

Goal: prettier pictures

Most of the distance between a raw file and a photograph is rendering, not taste

The raw, rendered flat



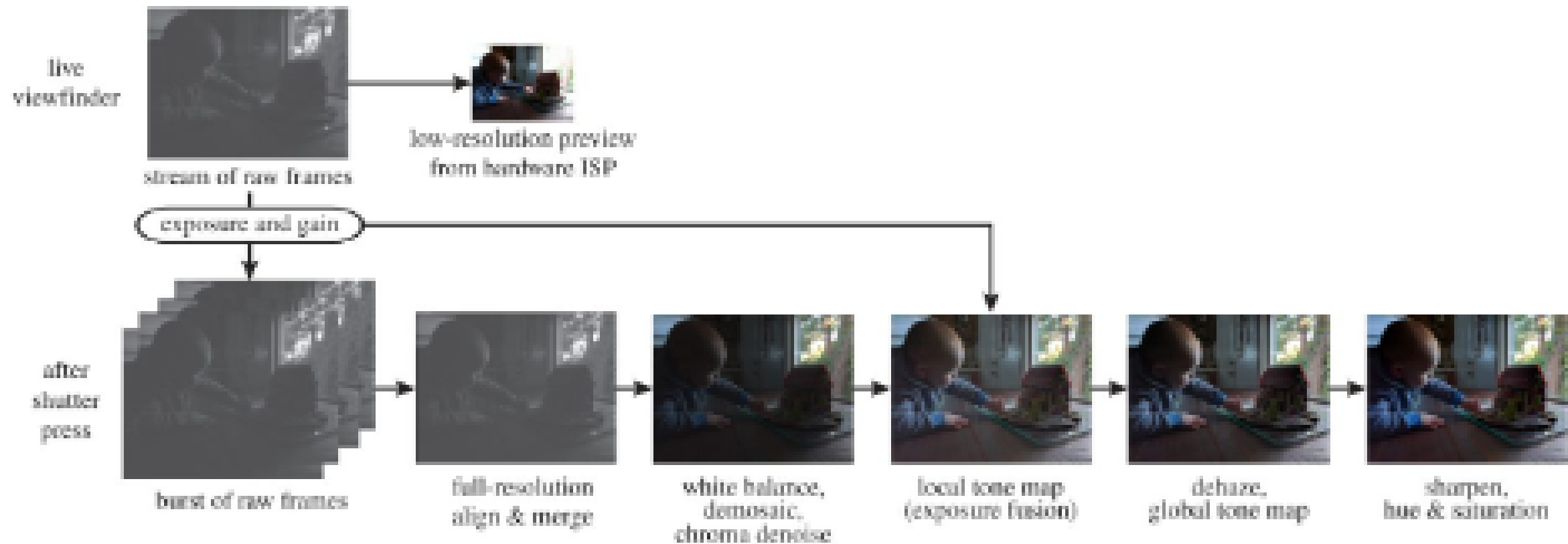
demosaic, camera white balance, matrix, encode — nothing else

The same frame, edited



the rendering every camera applies, plus the photographer's choices

Goal: alleviate physical limitations



Reproduced from Hasinoff et al. (2016), with permission of S. Hasinoff.

Goal: alleviate physical limitations

Panorama

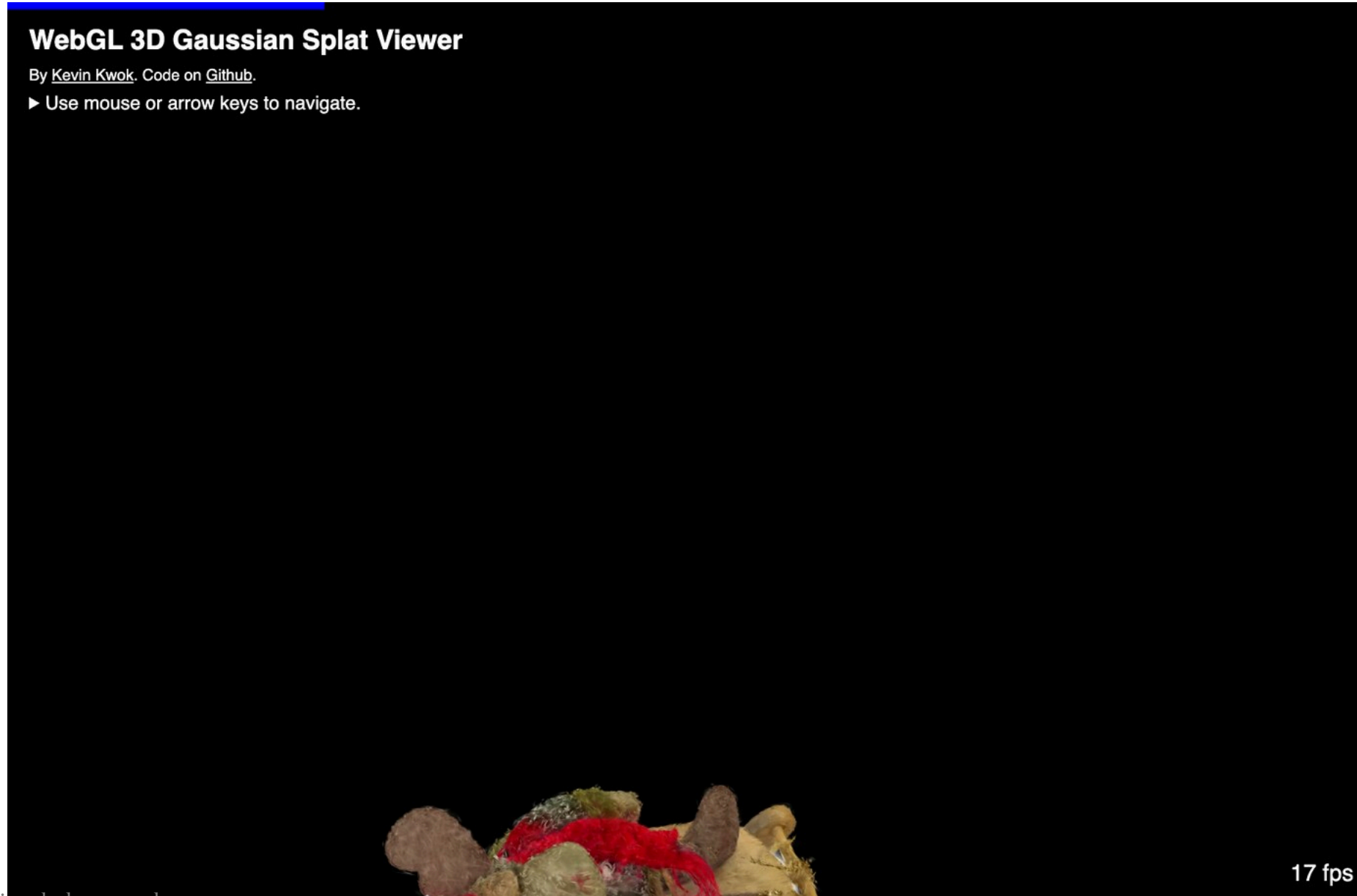


input frames



stitched panorama

Goal: record more information



notes: The goals of computational photography [notes](#) · [book](#)

Goal: more control after capture

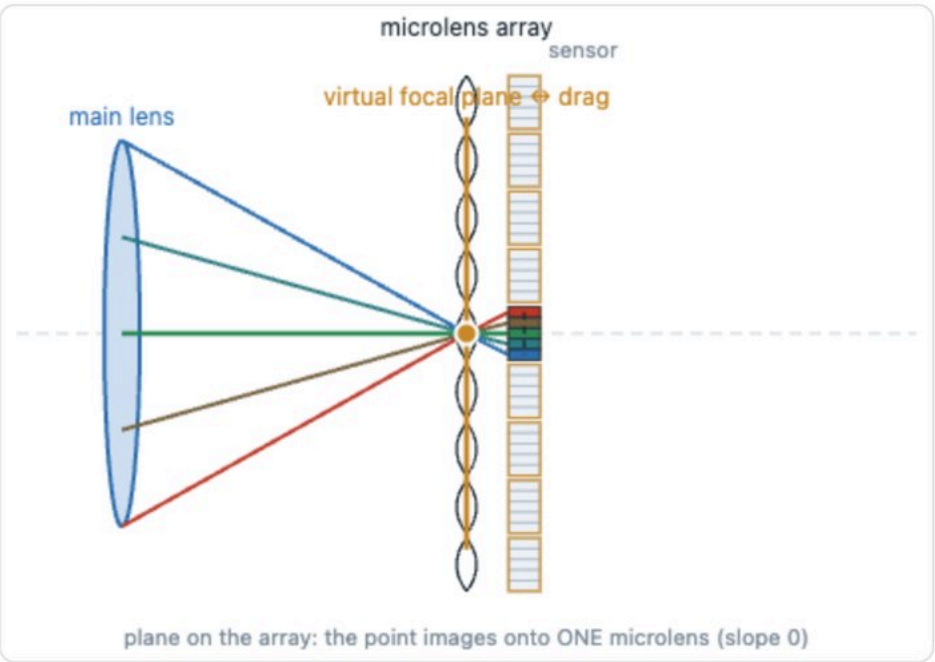
soundtrack: [Everything Is Awesome](#) (The LEGO Movie)

Refocusing is choosing the depth of the virtual focal plane

A plenoptic camera records the whole light field — every ray's **aperture position** (which sub-pixel behind a microlens) and **image position** (which microlens). To refocus *after* the shutter you pick a **virtual focal plane** at some depth and re-sum the rays as if the sensor sat there. On the **left**, the optical configuration: drag the **virtual focal plane** (or use the slider) and watch the rays that reconstruct one point tilt across the microlenses — that tilt is the shift-and-add slope. On the **right**, the **actual** result: the same slider shift-and-adds a real light field — Stanford's wide-baseline "chess" gantry capture — refocusing the photograph from near to far.

virtual focal-plane depth near far slope -0.50
synthetic aperture pinhole wide (shallow DoF)

optical configuration — main lens · microlens array · sub-pixels, with the virtual focal plane



the same slider refocuses a real light field actual demo · chess light field



9×9 sub-aperture views · slide the virtual focal plane

Goal: new visual experiences

Look around a 360° panorama

A stitched panorama isn't a flat picture — it's the whole sphere of directions around one viewpoint, wrapped onto an **equirectangular** image (longitude across, latitude up). Map that image back onto a sphere, put the camera at the centre, and you can **look anywhere**: **drag** to turn, **scroll** / **pinch** to zoom (change the field of view). The straight quay bends because a wide field of view flattens a curved slice of the sphere onto your screen — the same projection trade-off as any ultra-wide lens.

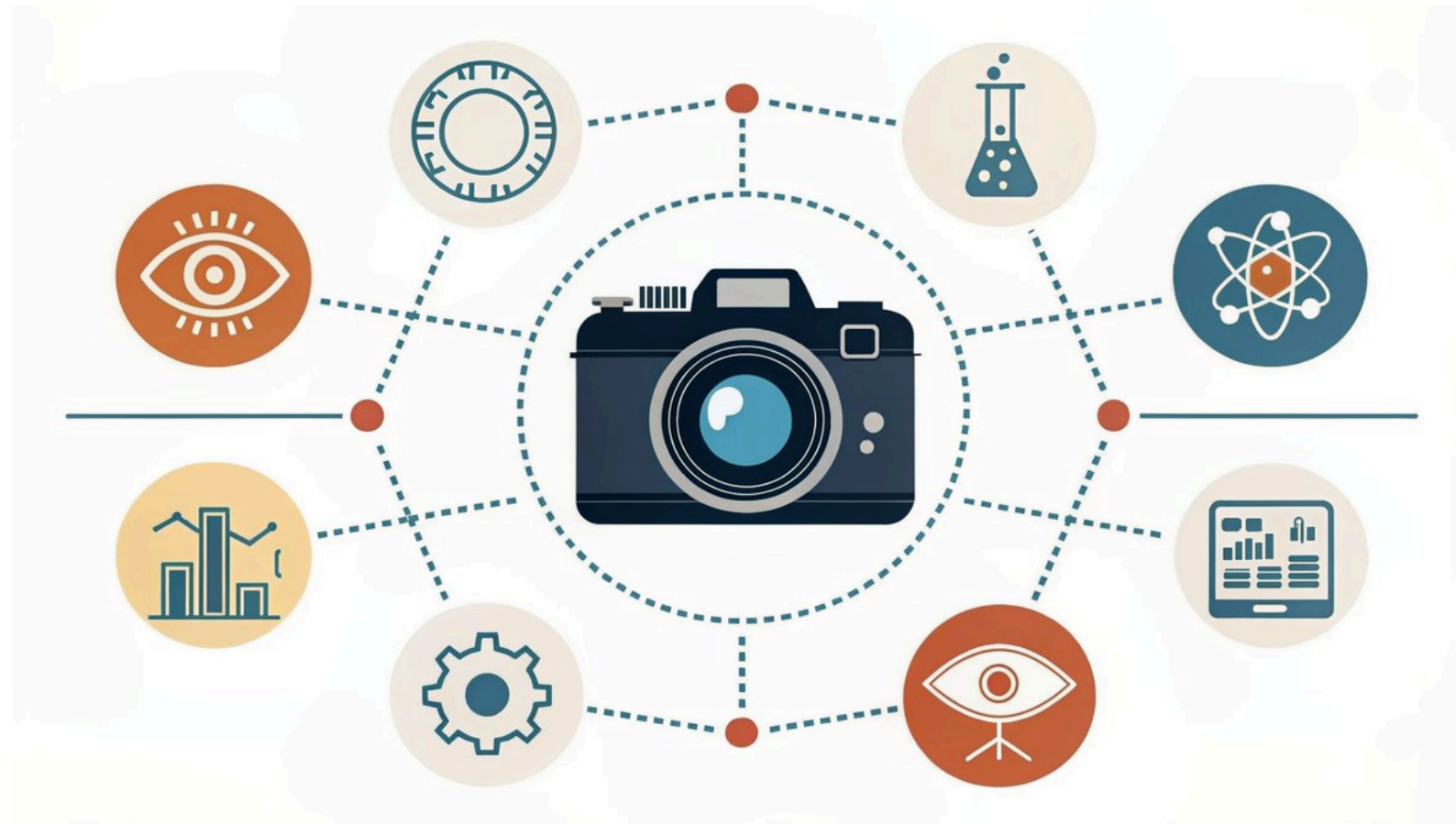


Goal: reveal the invisible



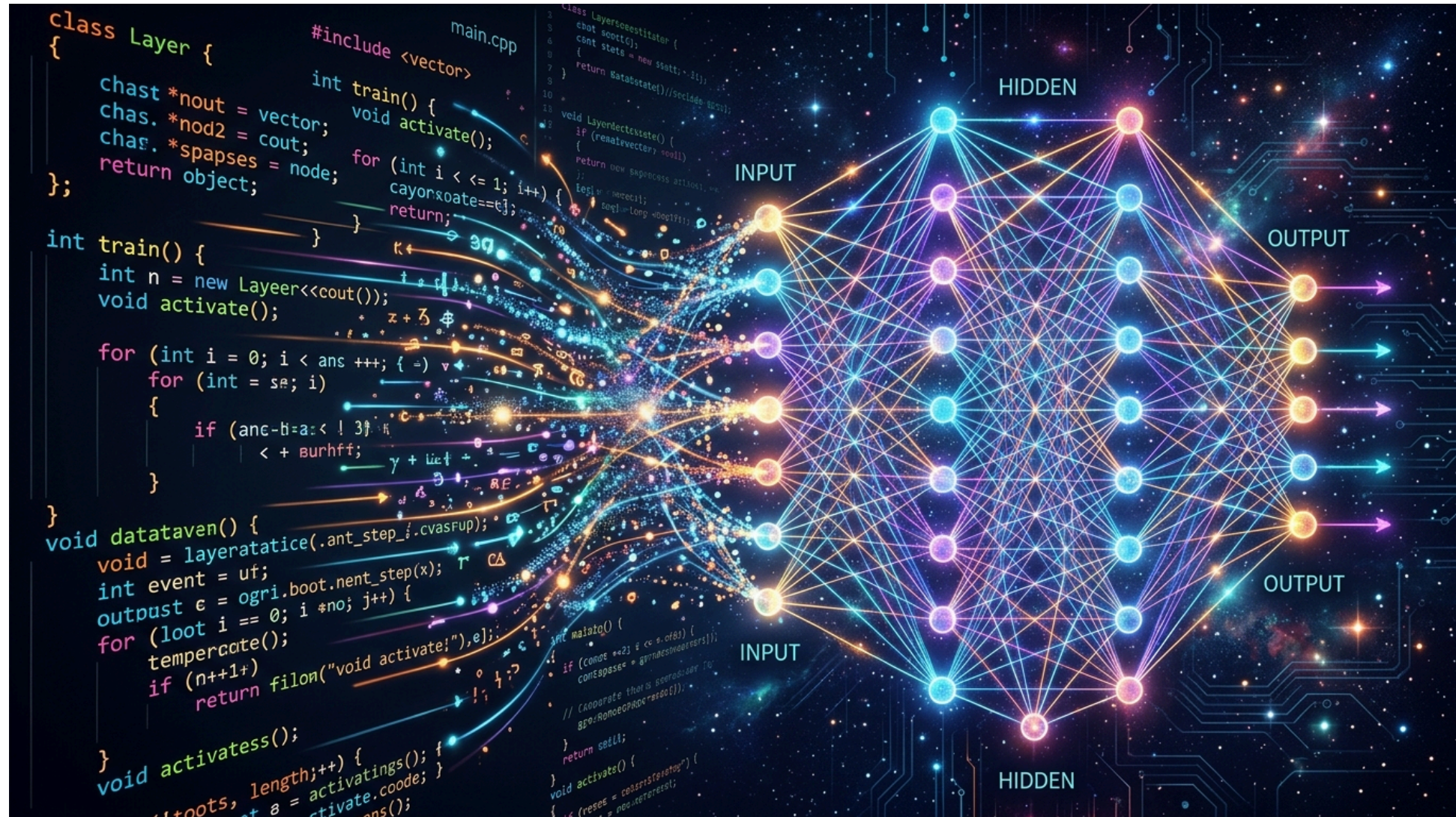
An interdisciplinary field

Optics, Math, Algorithms, AI and machine learning, Computer vision, Computer graphics, Systems, Electrical engineering, Mechanical engineering, Chemistry, Perception, Art, Human-computer interaction, Ethics



A field in transition

From manual to deep learning and generative AI





The class



Staff

- **Instructor: Frédo**
- **TAs:, Camille Biscarrat , Milena Sanchez**
- **Office hours:**
 - **TBD**
- **Use Piazza for any communication**

Practical details

- **Piazza: questions**
 - <https://piazza.com/mit/fall2026/683701>
 - Won't answer technical questions by email
 - Can be anonymous if you're shy
 - Sign up!
- **Magical pset system**
 - <https://compphoto-portal.fredopdurand.workers.dev/>
 - You will get accounts soon
- **Magical textbook and AI assistant**
 - <https://comp-photo-book.pages.dev/>

Grading

- 100% of the grade: assignments
- 6.8370 (grad): extra questions + 1 paper review
- Standard cutoffs:
 - $>90\% \Rightarrow A$
 - $>80\% \Rightarrow B$
 - $>70\% \Rightarrow C$
 - $>60\% \Rightarrow D$

Pset weights

- 0 has weight 0
- Auto pano + blending has weight 2
- NPR has weight 1.0
- Choose your own has weight 1.5
- Ethics writeup has weight 0.25
- All others have weight 1 (including morphing)
- Normalize by sum of weights, of course.

Assignments

- **Every week, then every two weeks**
 - See schedule
 - Due Wednesdays at 9pm.
- **Mostly programming**
 - C++
- **Turn in code, images, readme**
- **Collaboration policy:**
 - Chatting is encouraged
 - Code must be written alone
 - Only our AI assistant is allowed. It can be consulted but cannot write your code

Pre-reqs

- **You are not expected to know C++**
 - The first few p-sets will ramp up your skills
- **You are expected to know programming**
 - Pre-req: 6.1010
 - Including object-oriented programming
- **Basic knowledge of math (18.02)**
 - Basic linear algebra, derivatives of functions of multiple variables

Policies

- **Collaboration**
 - You are welcome and encouraged to chat about assignments
 - Code must be written on your own
- **Late**
 - 3 late days for the semester
 - No fractional late days (i.e. 1.01 late days = 2 late days)
 - A dean note is required to get an extra extension for medical reasons
 - More generous for last problem set

Assignments turn in

- **Magical submission system**
 - Put all in a zip file
- **Form**
 - 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)**

Computer-Aided Grading

- **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.
- **There will be an appeal process if you feel there is a mistake with the grading**
- **You'll get accounts soon**

Textbook: NEW and experimental

- <https://comp-photo-book.pages.dev/>
 - Co-written with Claude AI
 - Very rough around the edges, enormous, not complete, most of it never read by any human, will change throughout semester.
 - Not organized like the class
 - Can hide more optional stuff (three level)
- 3 areas: Book, Course notes (subset of book per lecture), and slides (here!)
- Links to articles in slides, notes and books
- NOT required reading

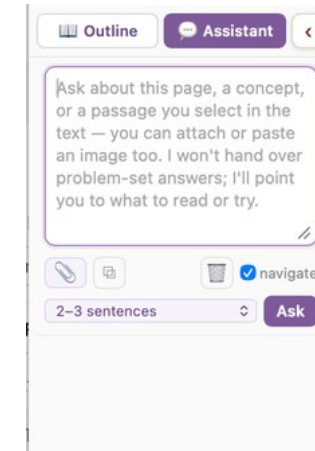
Work in progress

Email me about bugs and feature requests

Content will change/improve over the semester

AI policy

- You are only allowed to use the class's AI assistant at <https://comp-photo-book.pages.dev/> embedded in the book, notes, slides, and problem set
 - it is built to help you think, not to hand you the answer
 - it knows our material: chapters, notes, demos, problem sets
- Fine: brainstorming, explanations, editing, debugging, learning support
- Not fine: pasting problem statements into other (general) AI tools, asking for solutions to assigned problems, or submitting AI-written text, code, equations or analysis as your own
- Keep the line visible: distinguish your own work from AI-assisted work, and mention how the AI helped you (or not)
- Why: learning needs productive struggle. Remove the struggle and you stop growing as an independent, critical thinker



More on the AI assistant

- It can navigate to the notes/slides/book
- You can indicate what length of answer you want
- Currently based on Claude Sonnet 5
- It will try not to give away code
- VERY EXPERIMENTAL
 - Give me feedback and I will try to fix it
- It keeps anonymized logs of all interactions for me to debug and understand how AI helps you

Questions?

Getting to know you

- Course VI?
- know typescript, C, C++, swift
- 6.101, 6.102
- avid photographer
- wants to work in imaging
- has ever used a (real) camera
- retouched images
- stitched a pano? time lapse?
- HDR
- know f number? change ISO?

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 6.4400 and 6.8300
- Not much deep learning / generative AI

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

Non-Photo motivation

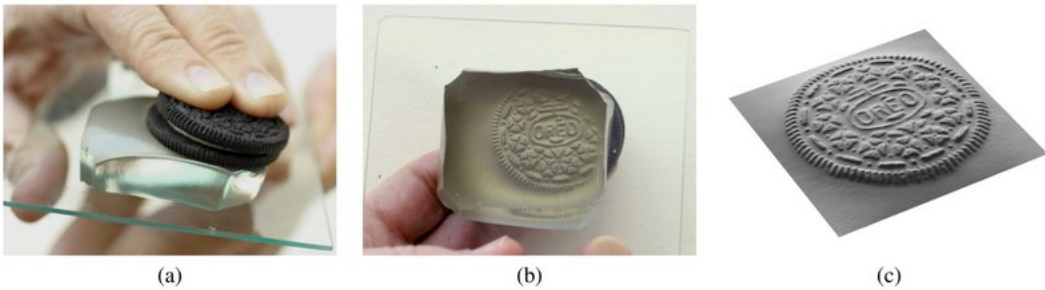
- It's about any kind of data !
 - Speech, motion, geometry, etc.
 - Example:
 - Music
 - Motion graphs
 - Poisson mesh editing
 - BTF shop
- Lots of fundamental numerical tools

GelSight: turning a camera into a sense of touch

The sensor is a tiny camera behind a soft gel: an assembled GelSight finger and its exploded view

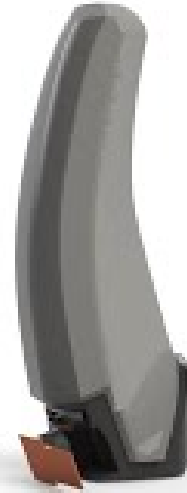


Press an object and the gel conforms; photometric stereo recovers its 3-D shape (here, a cookie)



So a robot can feel its grasp — a LEGO brick, a screwdriver, a flashlight — from contact geometry

Task	Pinch Grasp LEGO Brick		Lateral Pinch Grasp Screw Driver		Three-finger Power Grasp Flash Light		
Mono Diff. Image							
Est. $\begin{bmatrix} \tau_{bend} \\ \tau_{twist} \end{bmatrix}$ ($N \cdot mm$)	$\begin{bmatrix} 64.5 \\ 20.2 \end{bmatrix}$	$\begin{bmatrix} 59.8 \\ 28.3 \end{bmatrix}$	$\begin{bmatrix} 43.0 \\ 27.2 \end{bmatrix}$	$\begin{bmatrix} 69.0 \\ 32.2 \end{bmatrix}$	$\begin{bmatrix} 40.9 \\ 18.4 \end{bmatrix}$	$\begin{bmatrix} 65.5 \\ 20.7 \end{bmatrix}$	$\begin{bmatrix} 45.3 \\ 29.4 \end{bmatrix}$



GelSight Svelte

A tactile robot finger that is curved, compact, shaped like a human finger with proprioceptive + tactile sensing

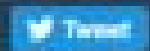
Jialiang (Alan) Zhao and Edward H. Adelson

Massachusetts Institute of Technology

Supported by Toyota Research Institute

<https://gelsight-svelte.alanz.info>

Calculating pathways through the song ...



Play

Tune

Short URL

Auto Canonizer Version

Questions?

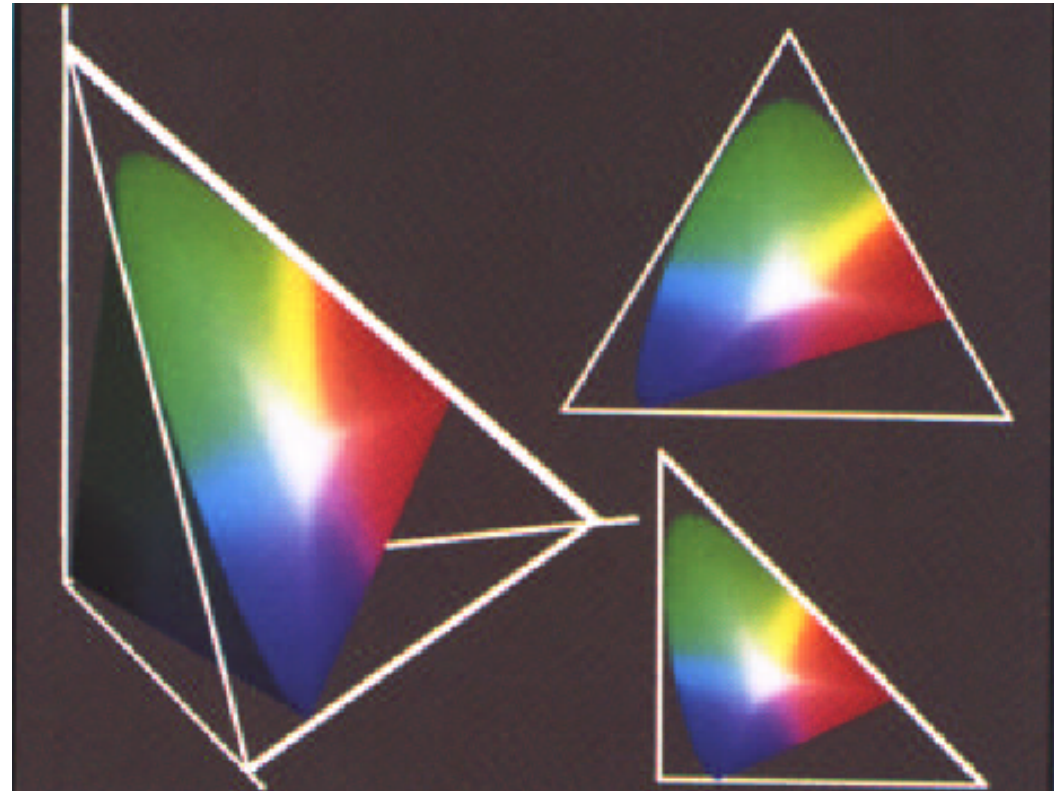
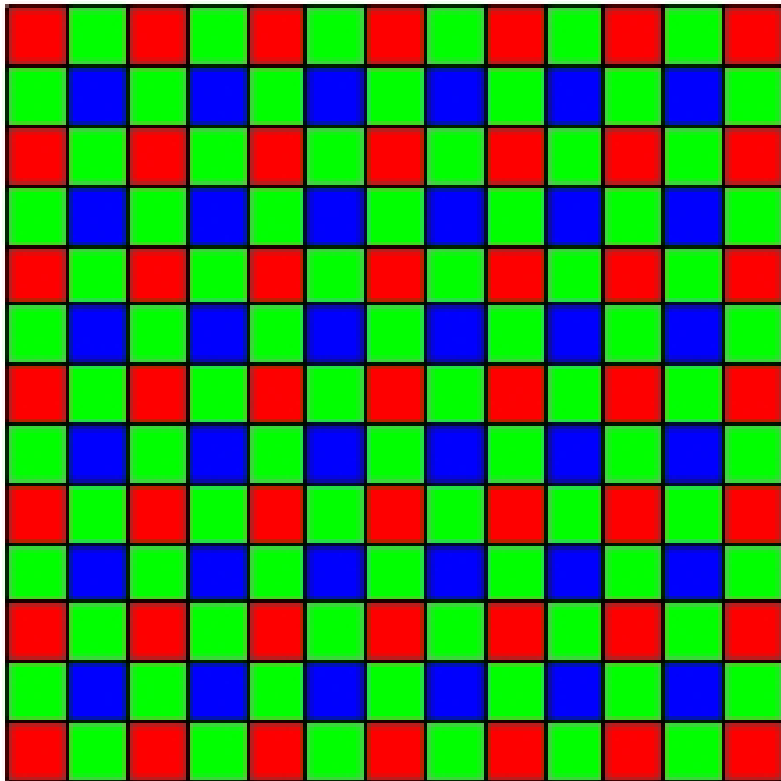


Syllabus (tentative)



Syllabus

- Color and color perception
- Demosaicing



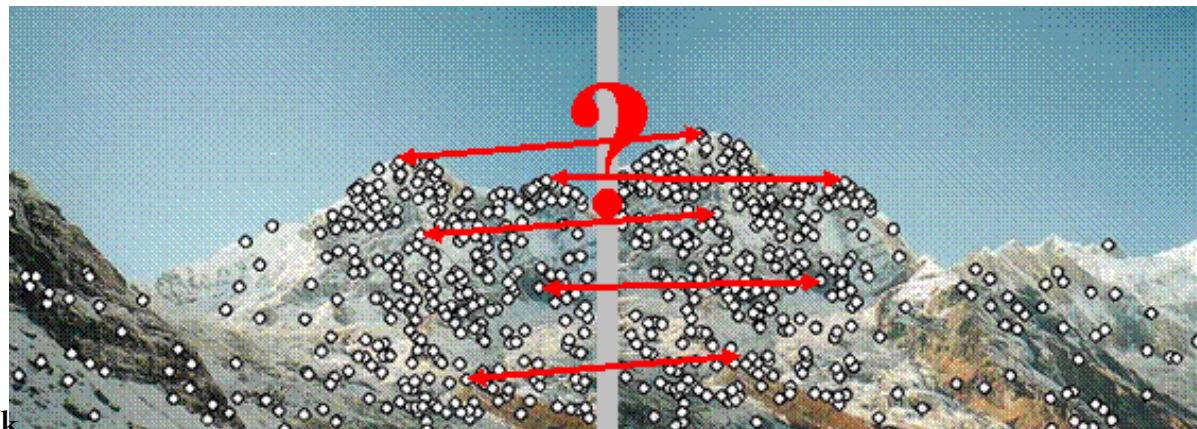
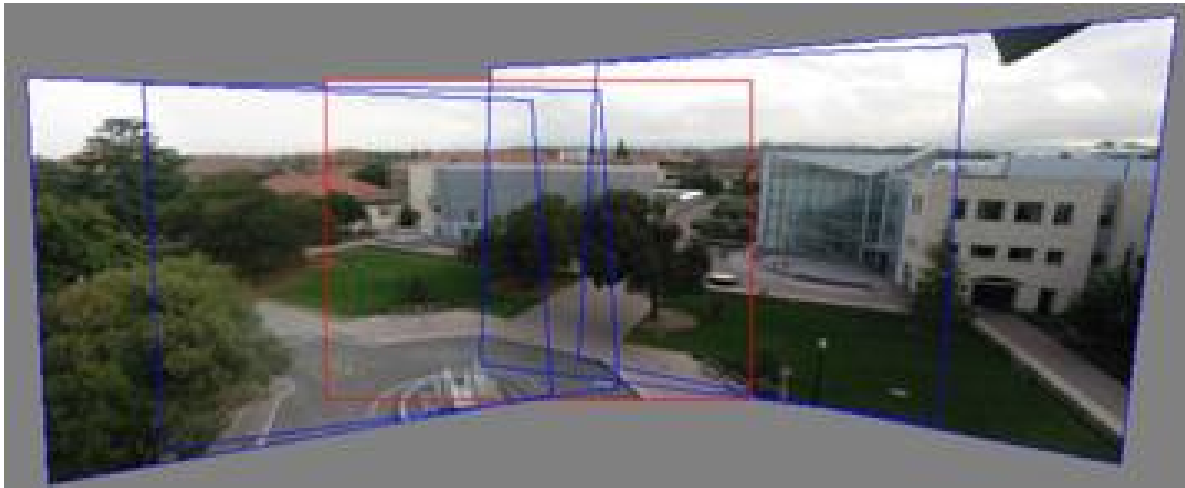
Syllabus

- High Dynamic Range Imaging
- Bilateral filtering and HDR display
- Matting



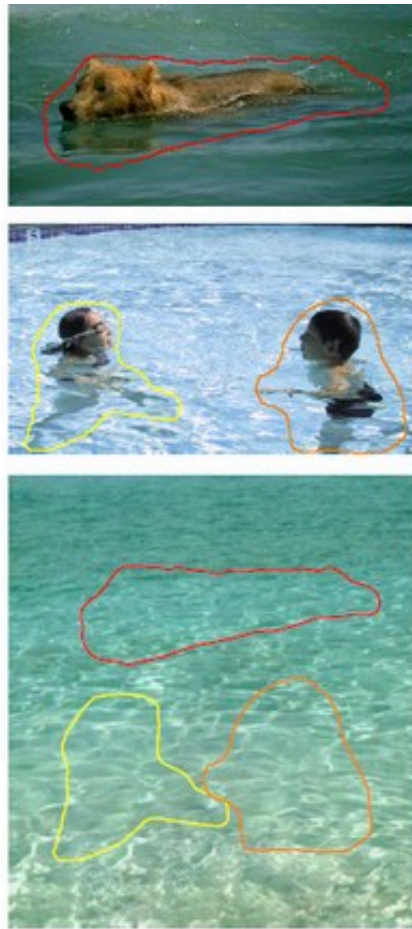
Syllabus

- Panoramic imaging
- Morphing



Syllabus

- Gradient image manipulation



sources/destinations



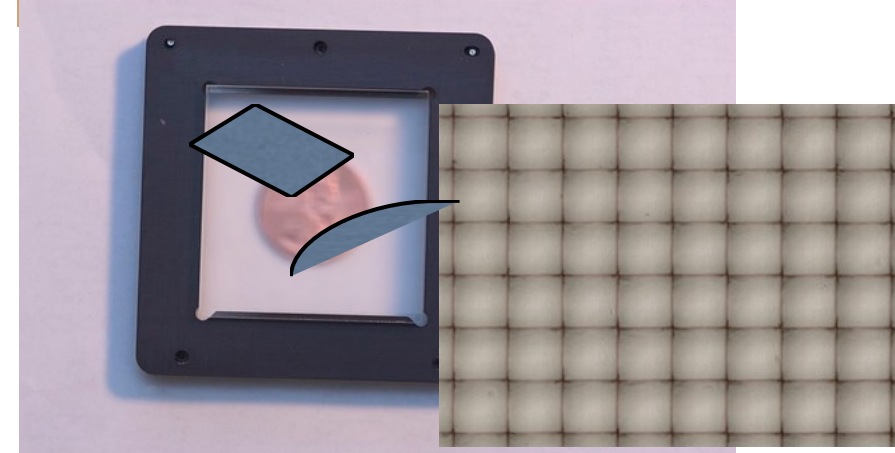
cloning



seamless cloning

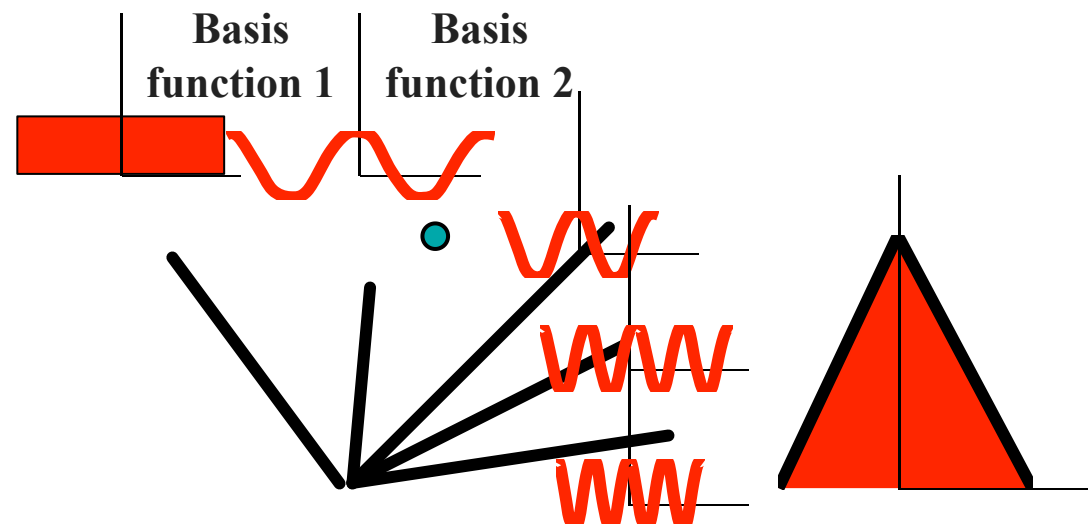
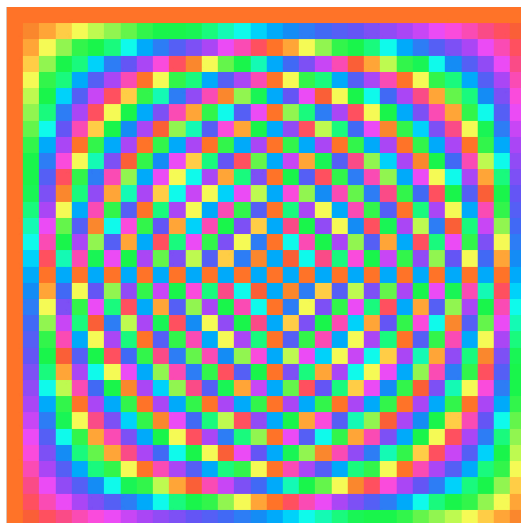
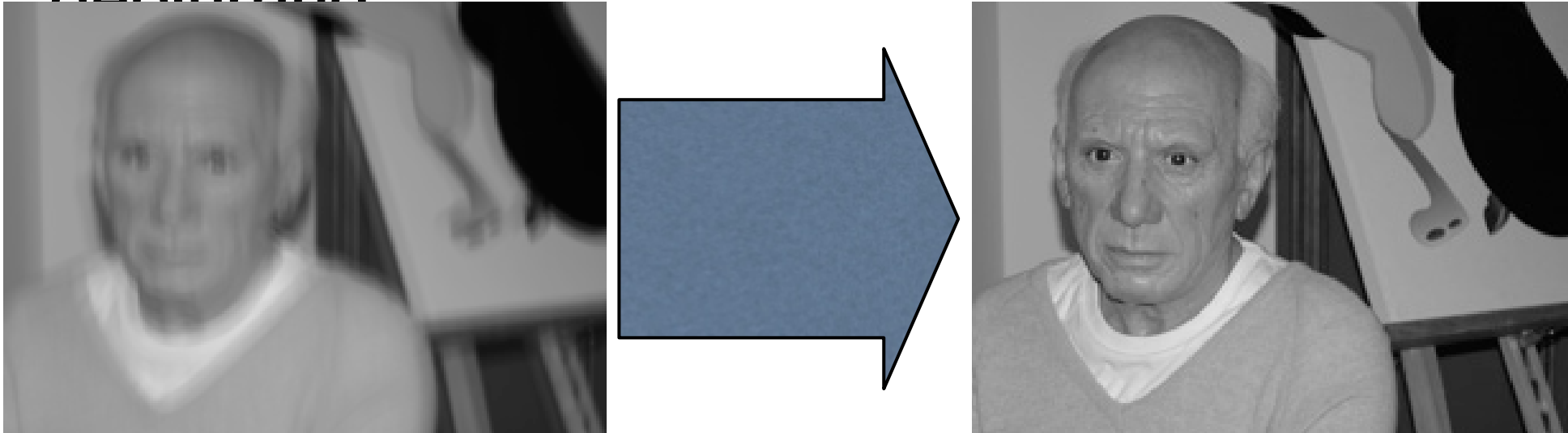
Syllabus

- Future cameras
- Plenoptic function and light fields



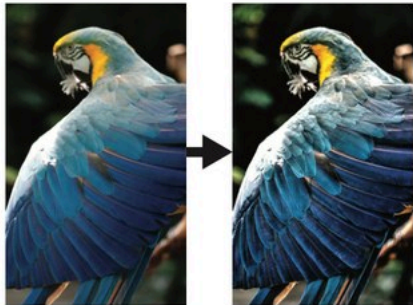
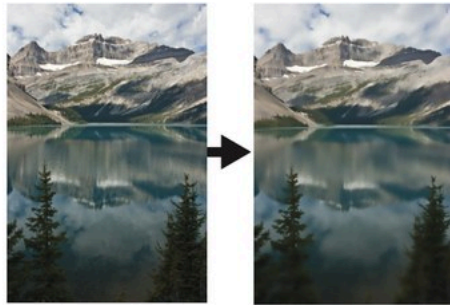
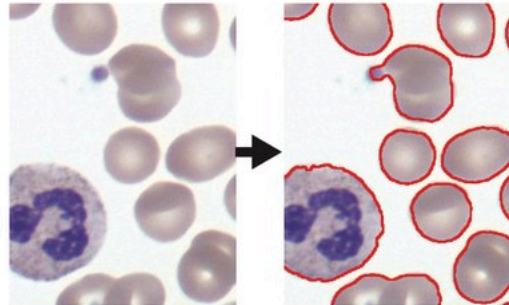
Syllabus

- Fourier transform and deblurring

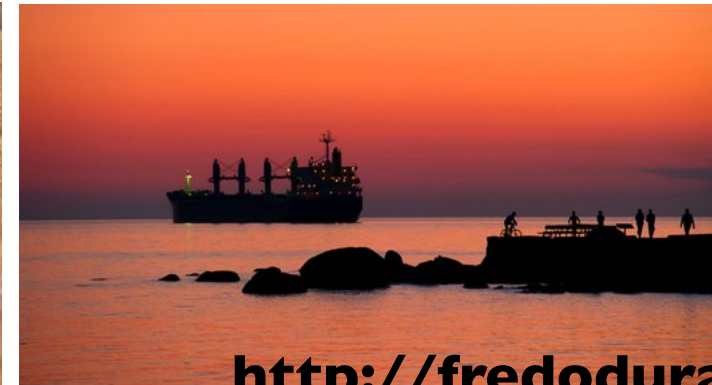
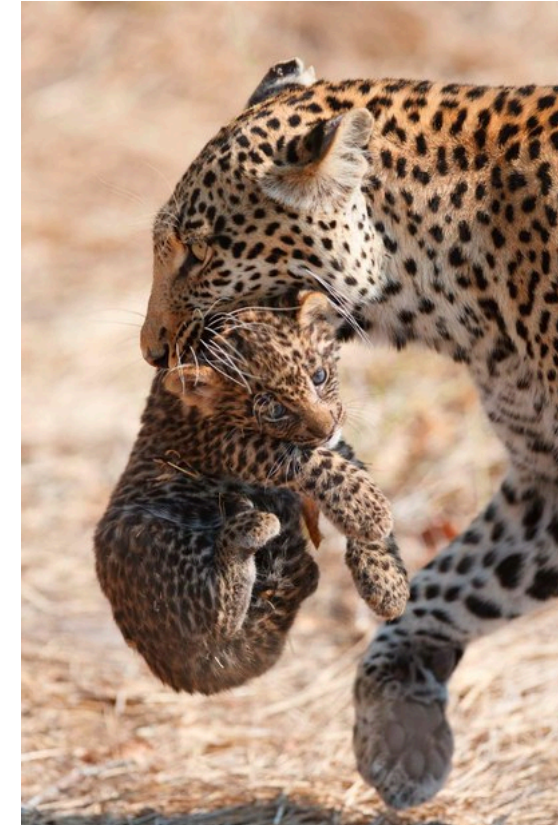


Halide & performance

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

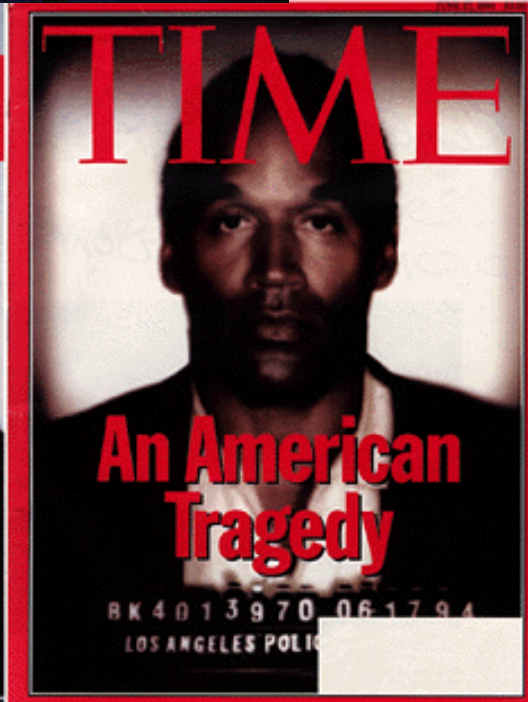
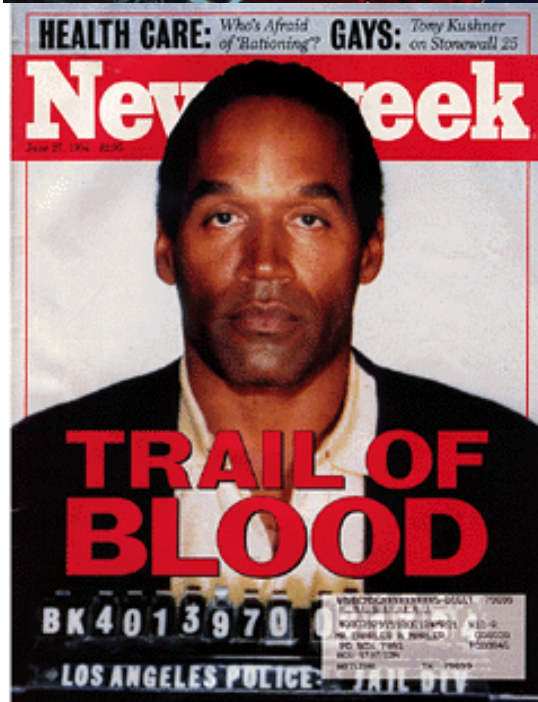
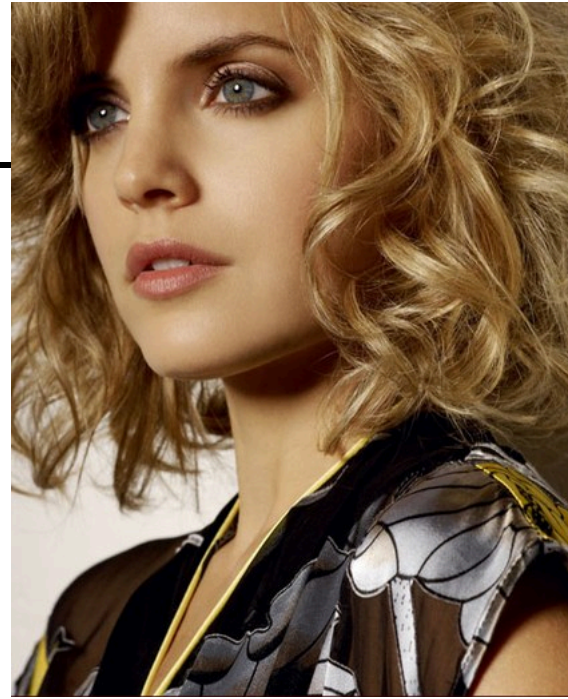
			
Camera Raw Pipeline	Local Laplacian Filter	Bilateral Grid	Snake Image Segmentation
Optimized NEON ASM: 463 lines Nokia N900: 772 ms	C++, OpenMP+IPP: 262 lines Quad-core x86: 627 ms	Tuned C++: 122 lines Quad-core x86: 472ms	Vectorized MATLAB: 67 lines Quad-core x86: 3800 ms
Halide algorithm: 145 lines schedule: 23 lines Nokia N900: 741 ms	Halide algorithm: 62 lines schedule: 7 lines Quad-core x86: 293 ms	Halide algorithm: 34 lines schedule: 6 lines Quad-core x86: 80 ms	Halide algorithm: 148 lines schedule: 7 lines Quad-core x86: 55 ms
2.75x shorter 5% faster than tuned assembly	3.7x shorter 2.1x faster	3x shorter 5.9x faster	70x faster
Porting to new platforms does not change the algorithm code, only the schedule			
Quad-core x86: 51 ms	CUDA GPU: 48 ms (13x)	CUDA GPU: 11 ms (42x)	CUDA GPU: 3 ms (1267x)

Good pictures



[http://fredoduran
d.net/](http://fredoduran.d.net/)

Ethics





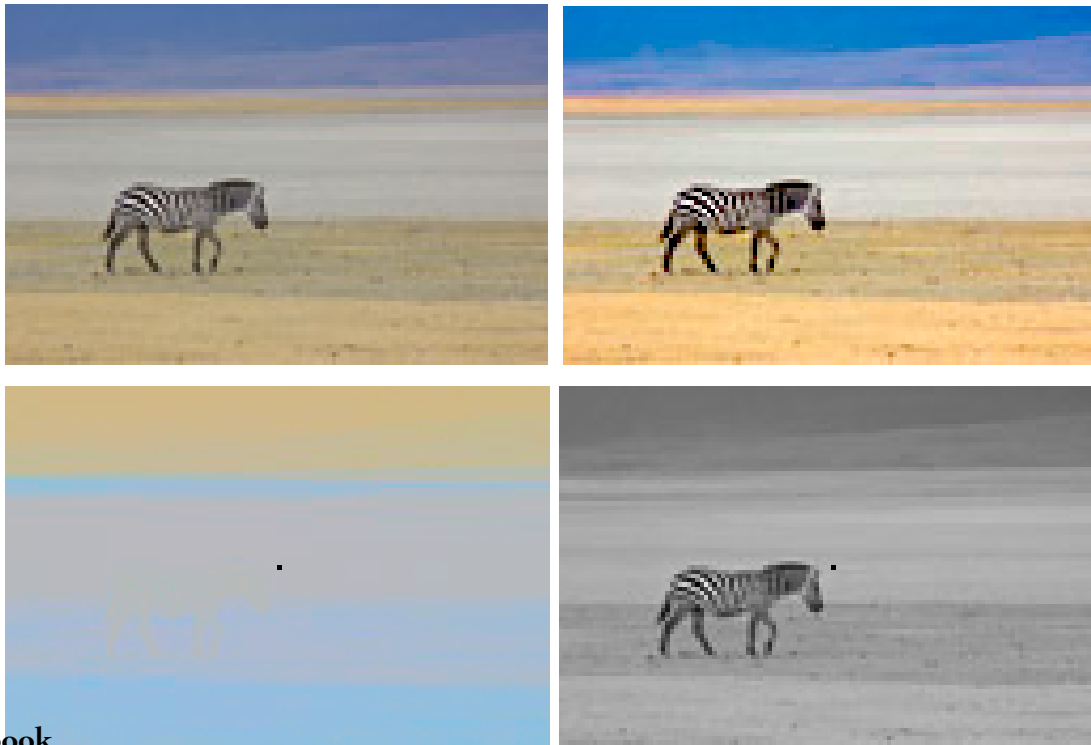
Assignments



Basics

- Brightness, contrast, black and white
- Color spaces, saturation
- Spanish Castle illusion <http://www.demonstrations.wolfram.com/SpanishCastleIllusion/>
- Histograms

http://www.johnsadowski.com/big_spanish_castle.php

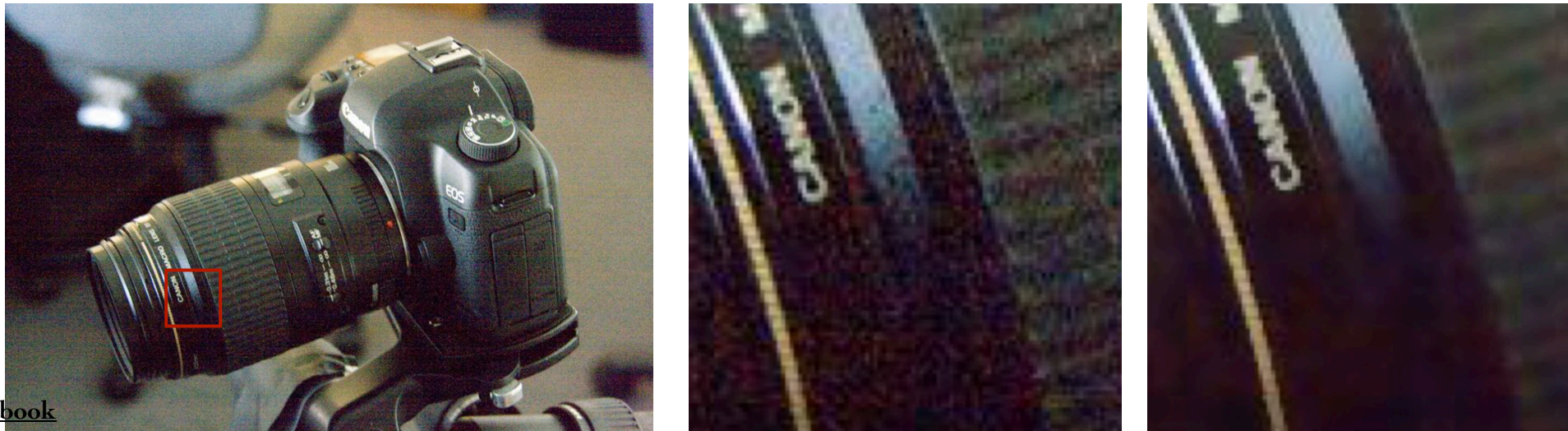
[illegible]

convolution & denoising

- Blur, unsharp mask



- Denoising with the bilateral filter

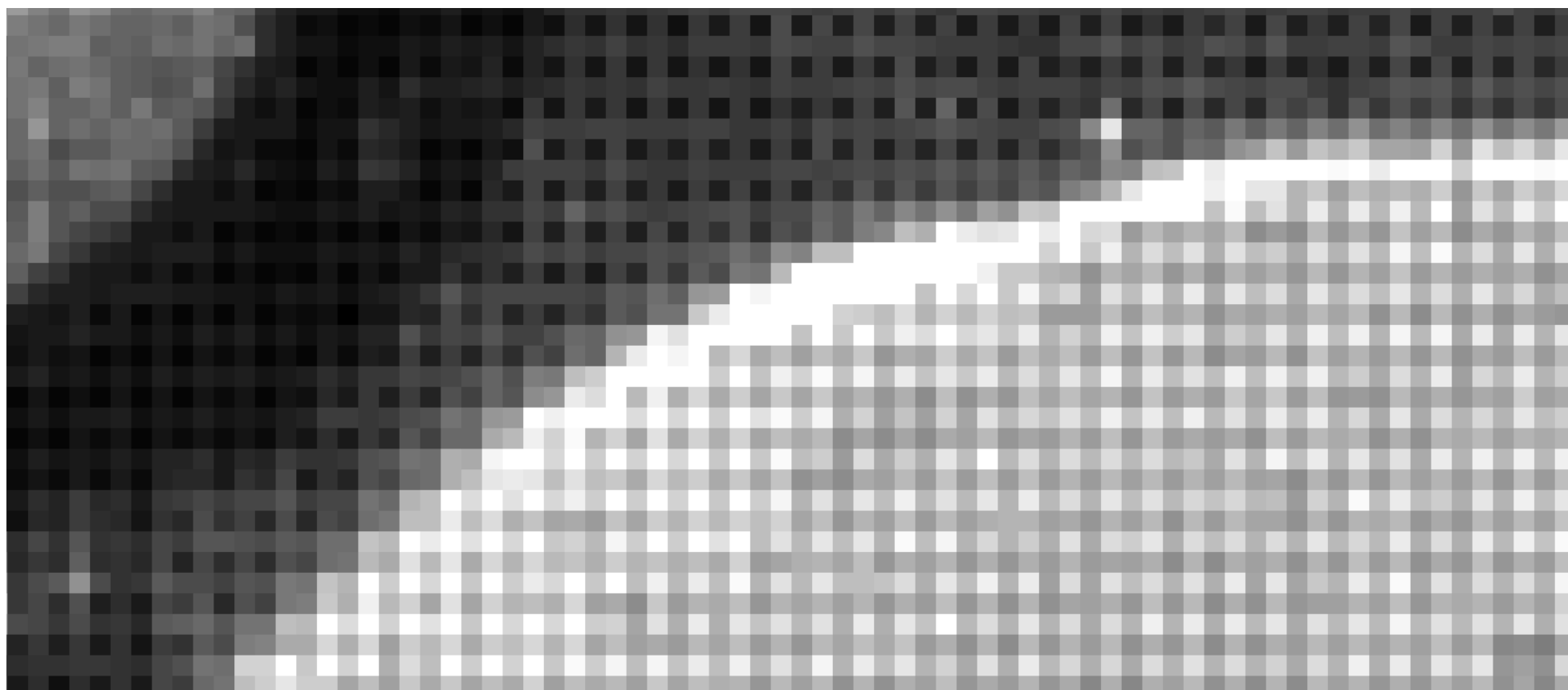
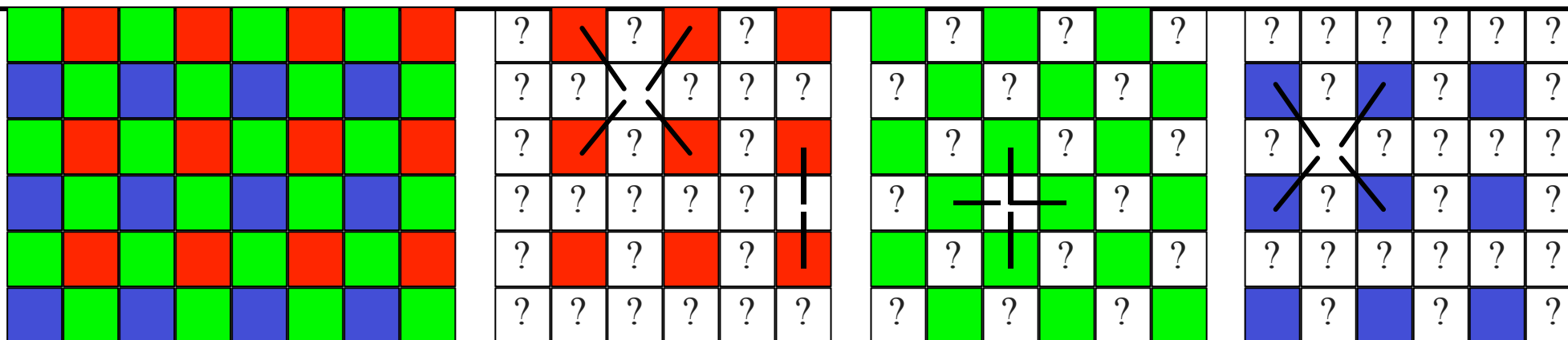


Resampling & Morphing

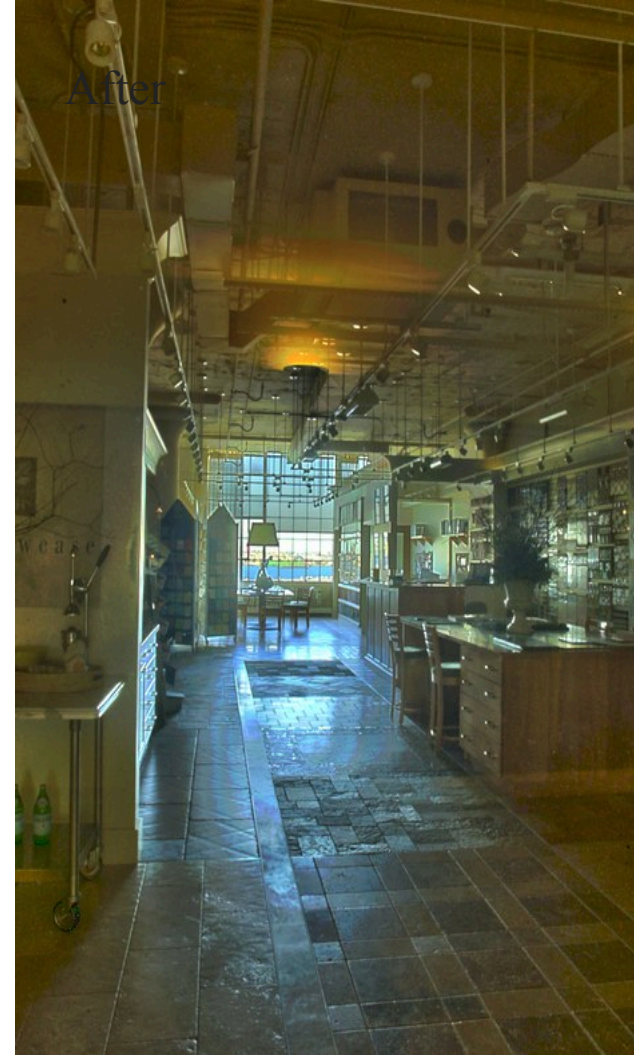
- Image rescaling & rotation
- Morphing one face into another one



Demosaicking

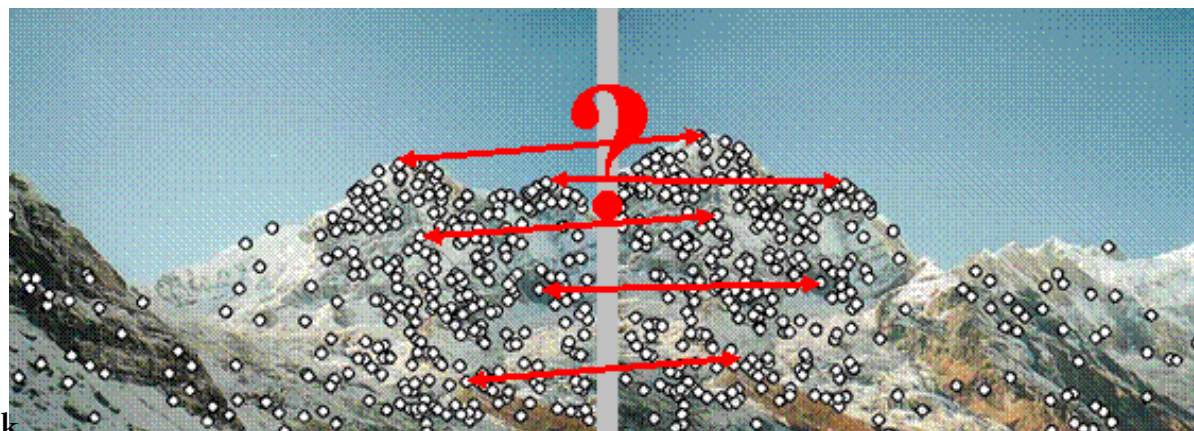
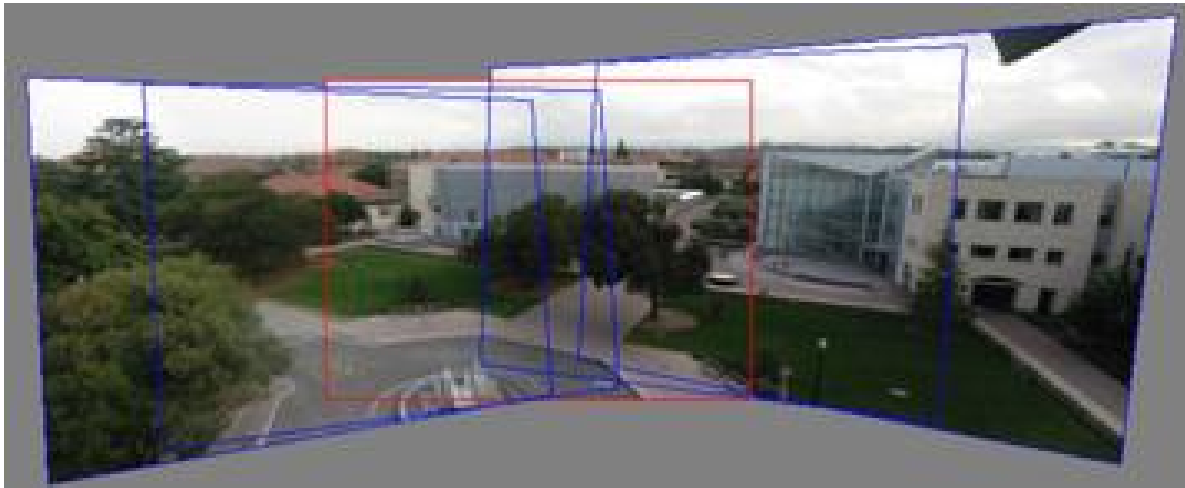


HDR imaging & tone mapping



Panoramas

- Manual pano
- Automatic pano using features

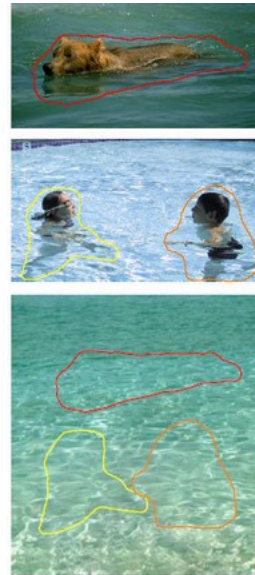
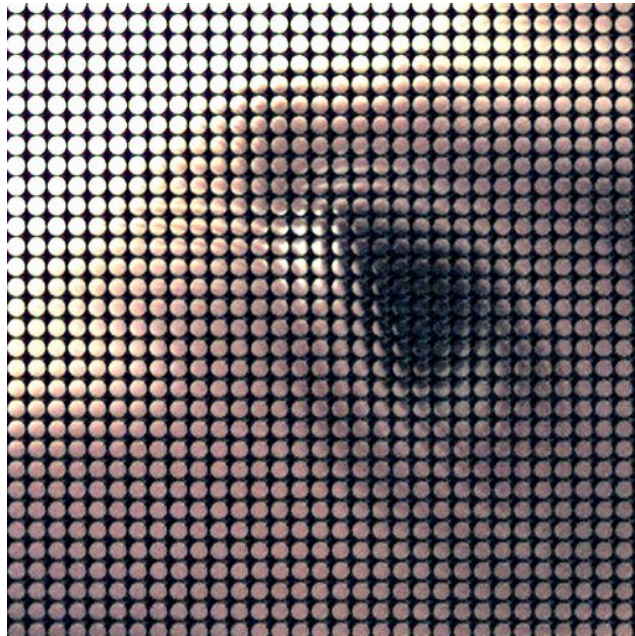
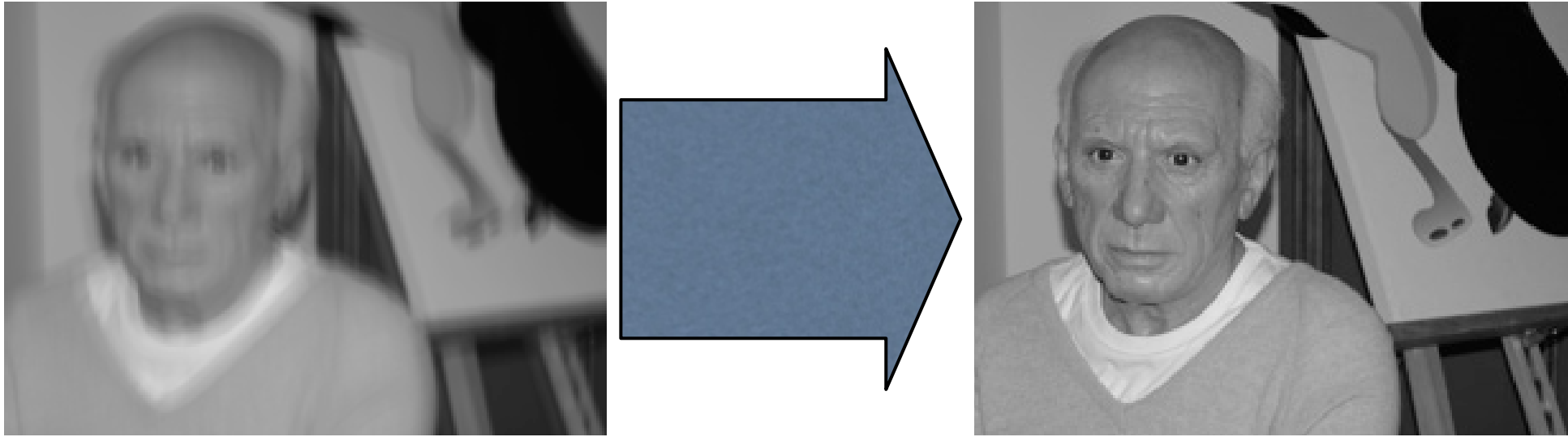




Non-Photorealistic Rendering



Choose your own p-set (or project)



sources/destinations



cloning



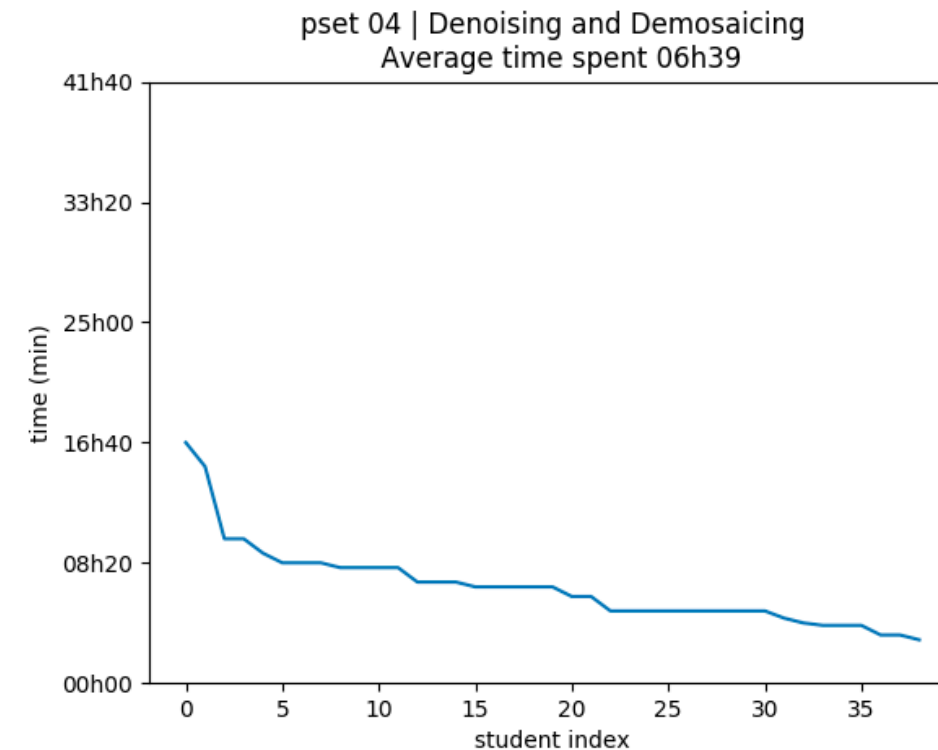
seamless cloning

Schedule (tentative)

- See book syllabus area <https://comp-photo-book.pages.dev/class/>

Average completion time

- pset 01 | Introduction to C++
 - 00h05 (00h10) | 39 answers
- pset 02 | Basic Image Processing
 - 04h40 (01h55) | 40 answers
- pset 03 | Convolution and the Bilateral Filter
 - 07h35 (03h29) | 39 answers
- pset 04 | Denoising and Demosaicing
 - 06h39 (02h48) | 39 answers
- pset 05 | HDR and tonemapping
 - 06h16 (02h34) | 39 answers
- pset 06 | Resampling, Warping and Morphing
 - 07h35 (02h20) | 39 answers
- pset 07 | Homographies and Manual Panoramas
 - 06h35 (02h45) | 38 answers
- pset 08 | Harris Corners, Features and Automatic P
 - 13h31 (07h30) | 38 answers
- pset 09 | Introduction to Halide
 - 09h30 (04h15) | 36 answers



Questions?

C++

Why C++ ?

- More efficient than java (because of compilation, memory)
 - ridiculously more efficient than python
 - sometimes 1,000x faster!
- Still standard for many areas where performance matters (graphics, imaging)
- Allows us to use Halide (even more efficient)
- Good experience

C++ for Python programmers

- More clunky, less safe, wayyyy more efficiency
- Compiled, not interpreted
 - translate all files into machine code, then execute from command line
- Separation of header and implementation files

C++ for Python programmers

- More clunky, less safe, way more efficiency
- Compiled, not interpreted
 - translate all files into machine code, then execute from command line
- Separation of header and implementation files
- Slightly different syntax, types first when declaring variables, no let
- Fewer built-in high-order functions
- Different I/O
- Explicit memory management

C++ for Python programmers

Python

```
1 def gcd(u, v):
2     # we will use Euclid's algorithm
3     # for computing the GCD
4     while v != 0:
5         r = u % v    # compute remainder
6         u = v
7         v = r
8     return u
9
10 if __name__ == '__main__':
11     a = int(raw_input('First value: '))
12     b = int(raw_input('Second value: '))
13     print 'gcd:', gcd(a,b)
```

C++

```
1 #include <iostream>
2 using namespace std;
3
4 int gcd(int u, int v) {
5     /* We will use Euclid's algorithm
6        for computing the GCD */
7     int r;      static type, declare variables
8     while (v != 0) {    parentheses, curly brackets
9         r = u % v;    // compute remainder
10        u = v;        semicolon at the end of lines
11        v = r;
12    }    curly bracket
13    return u;
14 }
15
16 int main( ) {
17     int a, b;
18     cout << "First value: ";
19     cin >> a;
20     cout << "Second value: ";
21     cin >> b;
22     cout << "gcd: " << gcd(a,b) << endl;
23     return 0;
24 }
```

Figure 1: Programs for computing a greatest common divisor, as written in Python and C++.

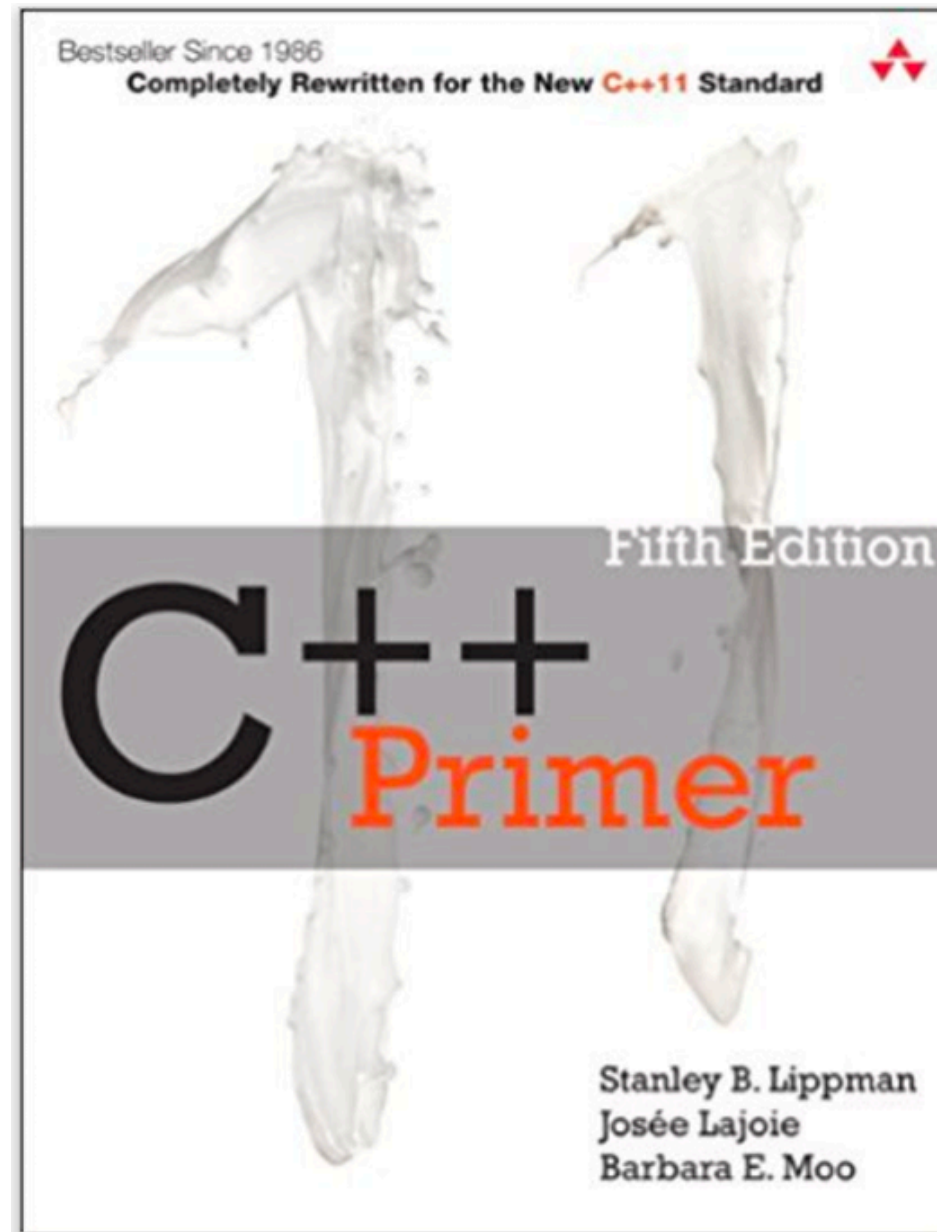
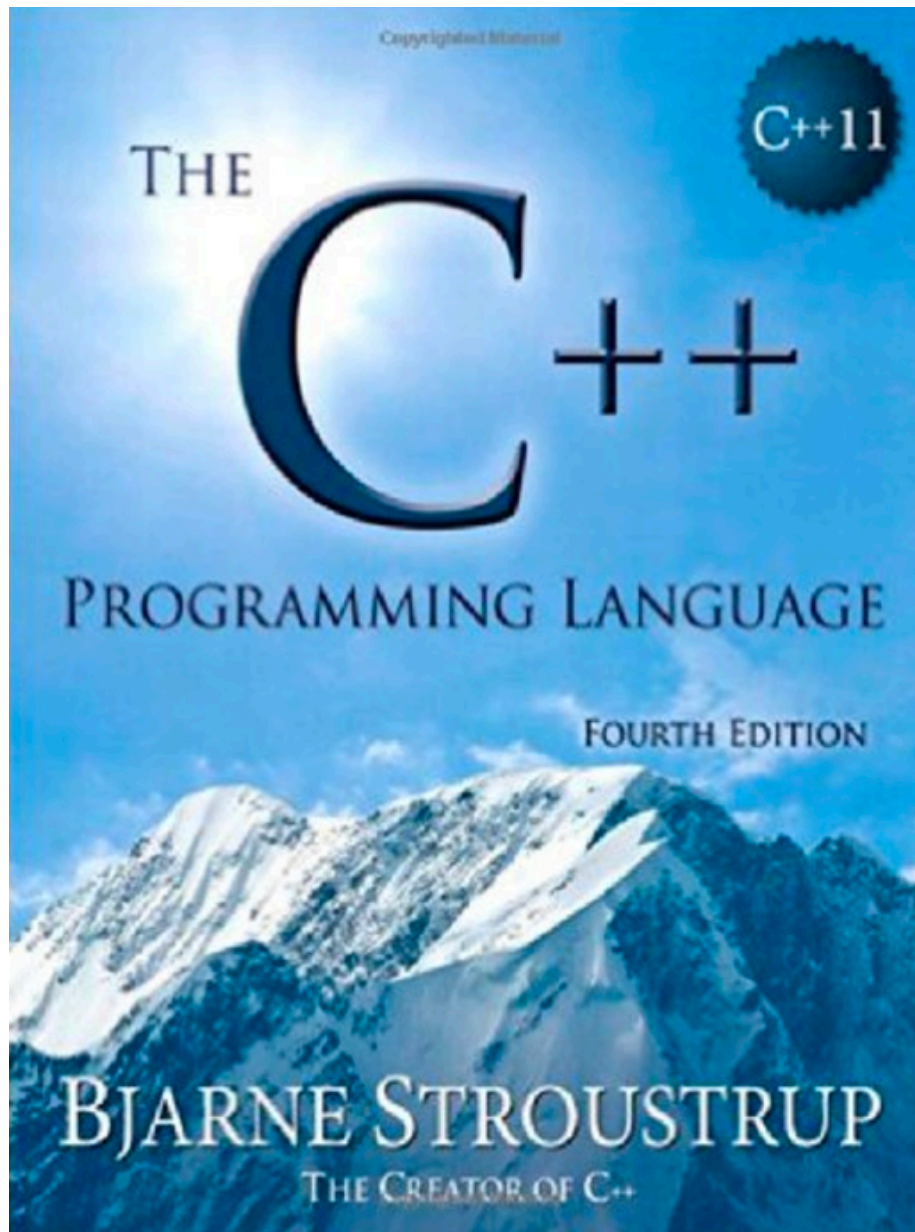
<http://cs.slu.edu/~goldwamh/publications/py>

[thon2cpp.pdf](#)

Resources for python folks

- <http://users.aims.ac.za/~nneoma/cpp4py/cppptut.html>
- <http://cs.slu.edu/~goldwamh/publications/python2cpp.pdf>

Books



notes: How to read this book [notes](#) · [book](#)

Files & Compilation (easy)

- no virtual machine, not interpreter, compiled to binary
- **2 types of source files:**
 - header (.h) contains class and function declaration
 - program (.cpp) contains implementation
 - exception: “inlined” methods implemented in .h
 - Originally for compiler efficiency purposes
 - Also nice for abstraction and separation of interface and implementation
- **Use “make” on terminal to compile a project**
 - makefile describes the build process
 - clunky and annoying

Image.h

```
#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

```
#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 otherwise
}

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 otherwise
}

// The next three functions should have the same implementation as the previous four
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 {
}

}
```

Files & Compilation (easy)

- “#include” only header files (not “import”)
 - It needs a silly #ifndef trick at the top of .h to avoid including files twice
 - because compilers are not smart enough to figure out that a file was already included
 - so you leverage the so-called preprocessor, a kind of interpreted meta-language that gets run before compilation
 - Put each header file in a big #ifndef statement that tests if a preprocessor variable has already been defined. Then at the end of it you define the variable
 - Will be done for you in template files.

Image.h

```
/* -----  
 * File:    Image.h  
 * Created: 2015-08-29  
 * -----  
 *  
 * The 6.815/6.865 Image class  
 * -----*/  
  
#ifndef __IMAGE_H  
#define __IMAGE_H  
  
#include <iostream>  
#include <vector>  
#include <string>  
#include <sstream>  
#include <cfloat>  
#include <cmath>  
  
#include "ImageException.h"  
#include "lodepng.h"  
  
class Image {  
public:  
    // Constructor to initialize an image of size width_*height_*channels_  
    // If height_ and channels_ are zero, the image will be one dimensional  
    // If channels_ is zero, the image will be two dimensional  
    Image(int width_, int height_ = 0, int channels_ = 0, const std::string &name="");  
  
    // Constructor to create an image from a file. The file needs to be in the PNG format  
    Image(const std::string &filename);  
};
```

...

```
Image* ReadJpeg ( char *path );  
  
#endif
```

Life outside of classes

- global variables
- functions

Memory

- A computer's memory can be seen as a big 1D array
- **Pointers are addresses to that memory**
 - In C++ you can manipulate addresses/pointers directly (dangerous!)
- **Memory for your program can be split into**
 - local to a function (on the **stack**)
 - global (**heap**)
- **We'll mostly ignore these tricky issues**

Classes

- implementation of methods (.cpp)
separate from declaration (.h)
- **public and private *sections***
 - Private methods/field are accessible only to the class
 - no need to say for each method
- **constructor: class name**
- **destructor**
 - called ~
 - always called for stack objects
 - but not heap/pointers, do it yourself
 - Not a huge concern for this class.

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)

- How do we pass inputs to functions/methods?
- Does the function/method work on a copy or on the original ?
- **Two considerations:**
 - Do we want to modify the original ?
 - Efficiency

Value, reference, pointer (hardish)

- **int myFunction(MyClass myObject)**
 - value: copies the object (UNLIKE JAVA)
 - Can be costly
- **int myFunction(MyClass *myObject)**
 - pointer: just passes the address. We won't use
- **int myFunction(MyClass &myObject)**
 - reference: modifiable version of the original object
 - not copied in memory, faster
 - Invoked the same way as passing by value
- **Pointers & references are mostly similar**
 - But we will avoid pointers as much as possible

const: important but overloaded

- **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);`
 - Passed by reference for efficiency reasons
 - that's the way we pass most parameters
 - means the method will only be allowed to use const methods

(semi advanced) C++ const

- A class can have two functions:

```
int myFunction(MyClass &myObject) const  
int myFunction(MyClass &myObject)
```

- How does the compiler know which to use?
- Think of it like a type
 - just like you can have the same function name with an int or float argument
 - int f(int n) as well as int f(float x)
 - when you call f(0.5) vs f(1) the compiler knows which version to call
 - The type system then chooses methods accordingly
- But why? Because const function can only call func functions for safeguarding

Memory 1

- (not too bad, not important for this class)
- **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]=0;`
will compile, run, and modify an arbitrary part of memory
- will often result in a segmentation fault
- or will corrupt a completely unrelated variable

Memory 2: unsafe (very painful)

- Unsafe arrays
 - `myArray[size+100]=0;`
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 difference place from where the problem is***

Image.h

```
#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

```
#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 otherwise
}

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 otherwise
}

// The next three functions should have the same implementation as the previous four
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 writes or uninitialized pointers corrupt memory and could crash at any time.
 - When you get segmentation fault, look for array or pointer operations.
(but hopefully we won't use pointers)

Development

- **Use either the terminal or your favorite IDE**
 - xcode, 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 and stack overflow are your friends, especially for compiler errors and C++ syntax**
 - You can try AI too but acknowledge it

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
 - templated by type, float in our case
- **Created with**
- `data = std::vector<float>(size, initialValue);`
 - for a float array with an initial value
- **References:**
 - http://www.codeguru.com/cpp/cpp/cpp_mfc/stl/article.php/c4027/C-Tutorial-A-Beginners-Guide-to-stdvector-Part-1.htm
 - <http://www.cplusplus.com/reference/vector/vector/>

1D to 3D

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

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

2D array

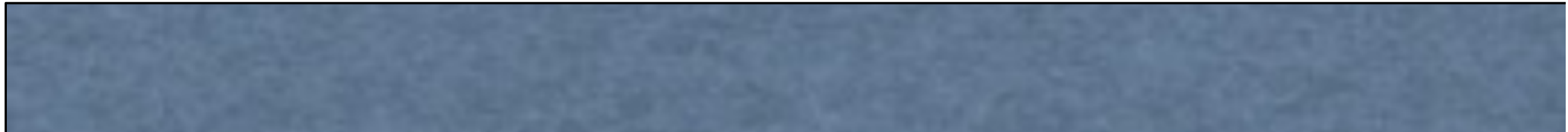
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

1D vector encoding

- For 3D index x, y, c where c is the color channel:
 - $y * \text{width} * \text{numChannels} + x * \text{numChannels} + c$



PSet 0



Pset 0

Speaker notes

- **Just a warm up**
- **familiarize yourself with C++**
- **compile**
- **write minimal code**
- **use vectors**
- **Basic brightness/contrast**