CS448f: Image Processing For Photography and Vision

Lecture 1

Today:

- Introductions
- Course Format
- Course Flavor
  - Low-level basic image processing
  - High-level algorithms from Siggraph

Introductions and Course Format

http://cs448f.stanford.edu/

Some Background Qs

- Here are some things I hope everyone is familiar with
  - Pointer arithmetic
  - C++ inheritance, virtual methods
  - Matrix vector multiplication
  - Variance, mean, median

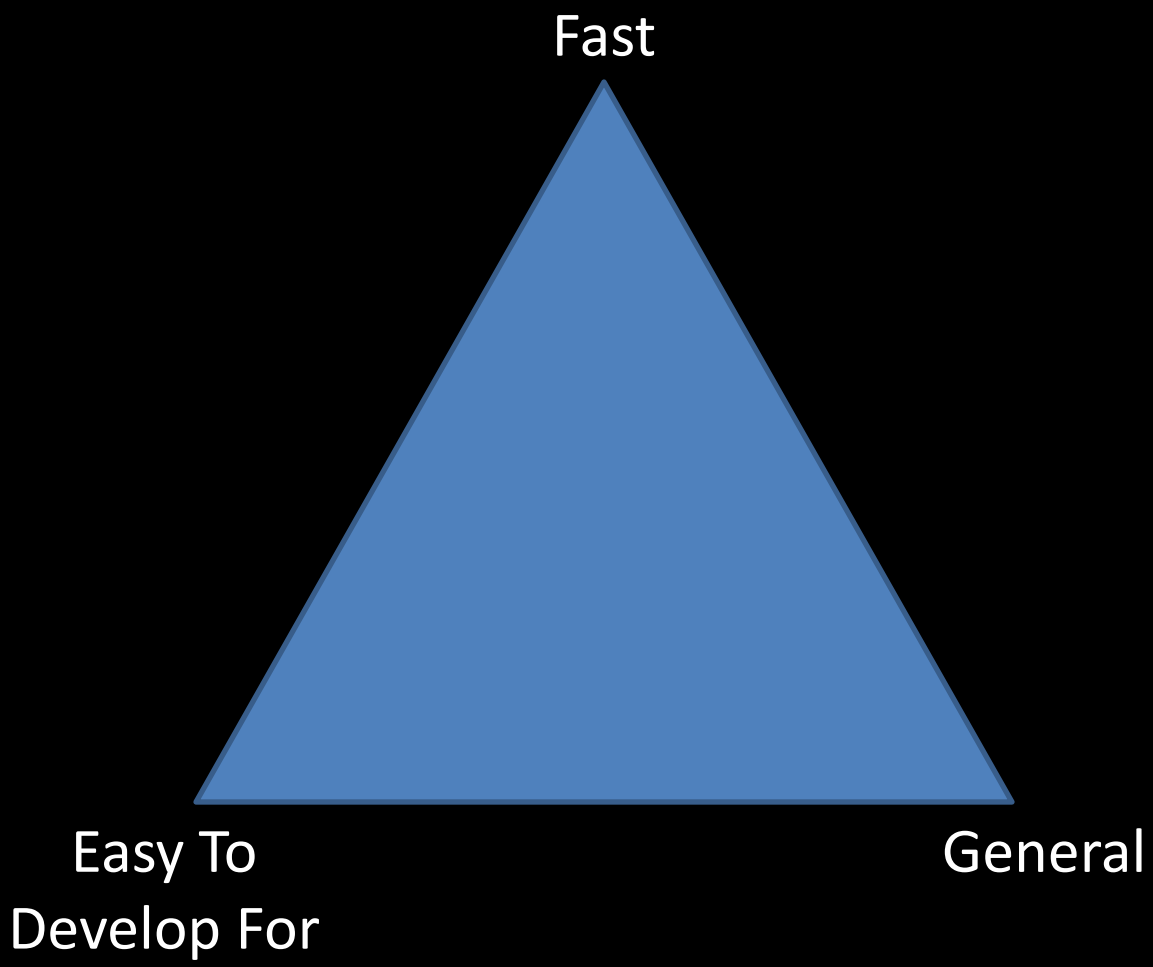
Some Background Qs

- Here are some things which I think some people will have seen and some people won't have:
  - Fourier Space stuff
  - Convolution
  - C++ using templates
  - Makefiles
  - Subversion (the version control system)

Background

- Make use of office hours – Jen and I enjoy explaining things.

What does image processing code look like?



Maximum Ease-of-Development

```
Image im = load(“foo.jpg”);

for (int x = 0; x < im.width; x++) {
  for (int y = 0; y < im.height; y++) {
    for (int c = 0; c < im.channels; c++) {
      im(x, y, c) *= 1.5;
    }
  }
}
```

Maximum Speed v0 (Cache Coherency)

```
Image im = load(“foo.jpg”);

for (int y = 0; y < im.height; y++) {
  for (int x = 0; x < im.width; x++) {
    for (int c = 0; c < im.channels; c++) {
      im(x, y, c) *= 1.5;
    }
  }
}
```

Maximum Speed v1 (Pointer Math)

```
Image im = load(“foo.jpg”);

for (float *imPtr = im->start();
     imPtr != im->end();
     imPtr++) {
  *imPtr *= 1.5;
}
```

Maximum Speed v2 (SSE)

```
Image im = load(“foo.jpg”);

assert(im.width*im.height*im.channels % 4 == 0);

__m128 scale = _mm_set_ps1(1.5);
for (float *imPtr = im->start();
     imPtr != im->end();
     imPtr += 4) {
  _mm_mul_ps(*((__m128 *)imPtr), scale);
}
```

Maximum Speed v3 (CUDA)

```
(…a bunch of code to initialize the GPU…)

Image im = load(“foo.jpg”);

(…a bunch of code to copy the image to the GPU…)

dim3 blockGrid((im.width-1)/8 + 1,
               (im.height-1)/8 + 1, 1);
dim3 threadBlock(8, 8, 3);

scale<<<blockGrid, threadBlock>>>(im->start(),
im.width(), im.height());

(…a bunch of code to copy the image back…)
```

Maximum Speed v3 (CUDA)

```
__global__ scale(float *im, int width, int
height, int channels) {
  const int x = blockIdx.x*8 + threadIdx.x;
  const int y = blockIdx.y*8 + threadIdx.y;
  const int c = threadIdx.z;
  if (x > width || y > height) return;
  im[(y*width + x)*channels + c] *= 1.5;
}
```

Clearly we should have stopped optimizing somewhere, probably before we reached this point.

Maximum Generality

```
Image im = load(“foo.jpg”);

for (int x = 0; x < im.width; x++) {
  for (int y = 0; y < im.height; y++) {
    for (int c = 0; c < im.channels; c++) {
      im(x, y, c) *= 1.5;
    }
  }
}
```

Maximum Generality v0
What about video?

```
Image im = load(“foo.avi”);

for (int t = 0; t < im.frames; t++) {
  for (int x = 0; x < im.width; x++) {
    for (int y = 0; y < im.height; y++) {
      for (int c = 0; c < im.channels; c++) {
        im(t, x, y, c) *= 1.5;
      }
    }
  }
}
```

Maximum Generality v1
What about multi-view video?

```
Image im = load("foo.strangeformat");

for (int view = 0; view < im.views; view++) {
  for (int t = 0; t < im.frames; t++) {
    for (int x = 0; x < im.width; x++) {
      for (int y = 0; y < im.height; y++) {
        for (int c = 0; c < im.channels; c++){
          …
        }
      }
    }
  }
}
```

Maximum Generality v2
Arbitrary-dimensional data

```
Image im = load(“foo.strangeformat”);

for (Image::iterator iter = im.start();
     iter != im.end();
     iter++) {
  *iter *= 1.5;
  // you can query the position within
  // the image using the array iter.position
  // which is length 2 for a grayscale image
  // length 3 for a color image
  // length 4 for a color video, etc
}
```

Maximum Generality v3
Lazy evaluation

```
Image im = load(“foo.strangeformat”);

// doesn’t actually do anything
im = rotate(im, PI/2);

for (Image::iterator iter = im.start();
     iter != im.end();
     iter++) {
  // samples the image at rotated locations
  *iter *= 1.5;
}
```

Maximum Generality v4 Streaming

```
Image im = load(“foo.reallybig”);
// foo.reallybig is 1 terabyte of data

// doesn’t actually do anything
im = rotate(im, PI/2);

for (Image::iterator iter = im.start();
     iter != im.end();
     iter++) {
  // the iterator class loads rotated chunks
  // of the image into RAM as necessary
  *iter *= 1.5;
}
```

Maximum Generality v5
Arbitrary Pixel Data Type

```
Image im<unsigned short> =
  load(“foo.reallybig”);
// foo.reallybig is 1 terabyte of data

// doesn’t actually do anything
im = rotate<unsigned short>(im, PI/2);

for (Image::iterator iter = im.start();
     iter != im.end();
     iter++) {
  // the iterator class loads rotated chunks
  // of the image into RAM as necessary
  *iter *= 3;
}
```

Maximum Generality v4
Streaming

```
Image im<unsigned short> =
  load("foo.reallybig");
// foo.reallybig is 1 terabyte of data

// doesn't actually do anything
im = rotate<unsigned short>(im, PI/2);

for (Image::iterator iter = im.start();
     iter != im.end();
     iter++) {
  // the iterator class loads rotated chunks
  // of the image into RAM as necessary
  *iter *= 3;
}
```

WAY TOO COMPLICATED

99% OF THE TIME

Speed vs Generality

```
Image im = load("foo.reallybig");
// foo.reallybig is 1 terabyte of data

// doesn't actually do anything
im = rotate(im, PI/2);

for (Image::iterator iter = im.start();
     iter != im.end();
     iter++) {
  // the iterator class loads rotated chunks
  // of the image into RAM as necessary
  *iter *= 1.5;
}
```

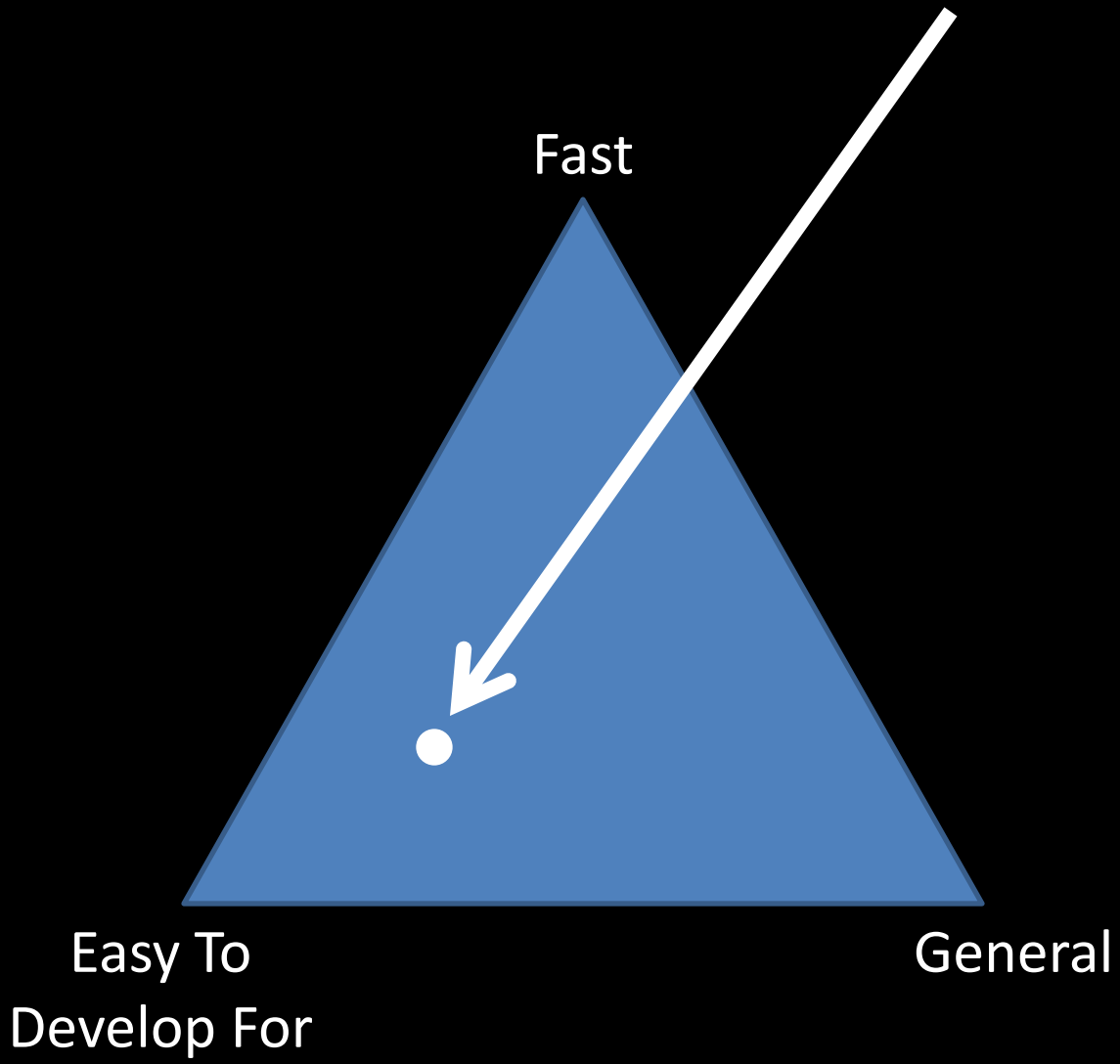
Incompatible

```
Image im = load("foo.jpg");

assert(im.width*im.height*im.channels % 4 == 0);

__m128 scale = _mm_set_ps1(1.5);
for (float *imPtr = im->start();
     imPtr != im->end();
     imPtr += 4) {
  _mm_mul_ps(*((__m128 *)imPtr), scale);
}
```

For this course: ImageStack



ImageStack

```
Image im = Load::apply("foo.jpg");

for (int t = 0; t < im.frames; t++) {
  for (int y = 0; y < im.height; y++) {
    for (int x = 0; x < im.width; x++) {
      for (int c = 0; c < im.channels; c++) {
        im(t, x, y)[c] *= 1.5;
      }
    }
  }
}
```

ImageStack
Concessions to Generality

```
Image im = Load::apply("foo.jpg");

for (int t = 0; t < im.frames; t++) {
  for (int y = 0; y < im.height; y++) {
    for (int x = 0; x < im.width; x++) {
      for (int c = 0; c < im.channels; c++) {
        im(t, x, y)[c] *= 1.5;
      }
    }
  }
}
```

Four dimensions is usually enough

ImageStack
Concessions to Generality

```
Image im = Load::apply("foo.jpg");

for (int t = 0; t < im.frames; t++) {
  for (int y = 0; y < im.height; y++) {
    for (int x = 0; x < im.width; x++) {
      for (int c = 0; c < im.channels; c++) {
        im(t, x, y)[c] *= 1.5;
      }
    }
  }
}
```

Floats are general enough

ImageStack
Concessions to Generality

```
Image im = Load::apply("foo.jpg");

Window left(im, 0, 0, 0,
  im.frames, im.width/2, im.height);

for (int t = 0; t < left.frames; t++)
  for (int y = 0; y < left.height; y++)
    for (int x = 0; x < left.width; x++)
      for (int c = 0; c < left.channels; c++)
        left(t, x, y)[c] *= 1.5;
```

Cropping can be done lazily, if you just want to process a sub-volume.

ImageStack Concessions to Speed

```
Image im = Load::apply("foo.jpg");

Window left(im, 0, 0, 0,
  im.frames, im.width/2, im.height);

for (int t = 0; t < left.frames; t++)
  for (int y = 0; y < left.height; y++)
    for (int x = 0; x < left.width; x++)
      for (int c = 0; c < left.channels; c++)
        left(t, x, y)[c] *= 1.5;
```

Cache-Coherency

ImageStack Concessions to Speed

```
Image im = Load::apply("foo.jpg");

Window left(im, 0, 0, 0,
  im.frames, im.width/2, im.height);

for (int t = 0; t < left.frames; t++)
  for (int y = 0; y < left.height; y++) {
    float *scanline = left(t, 0, y);
    for (int x = 0; x < left.width; x++)
      for (int c = 0; c < left.channels; c++)
        (*scanline++) *= 1.5;
  }
```

Each scanline guaranteed to be consecutive in memory, so pointer math is OK

ImageStack Concessions to Speed

```
Image im = Load::apply("foo.jpg");

Window left(im, 0, 0, 0,
  im.frames, im.width/2, im.height);

for (int t = 0; t < left.frames; t++)
  for (int y = 0; y < left.height; y++)
    for (int x = 0; x < left.width; x++)
      for (int c = 0; c < left.channels; c++)
        left(t, x, y)[c] *= 1.5;
```

This operator is defined in a header, so is inlined and fast, but can't be virtual (rules out streaming, lazy evaluation, other magic).

ImageStack
Ease of Development

```
Image im = Load::apply("foo.jpg");

im = Rotate::apply(im, M_PI/2);

im = Scale::apply(im, 1.5);

Save::apply(im, "out.jpg");
```

Each image operation is a class (not a function)

ImageStack
Ease of Development

```
Image im = Load::apply("foo.jpg");

im = Rotate::apply(im, M_PI/2);

im = Scale::apply(im, 1.5);

Save::apply(im, "out.jpg");
```

Images are reference-counted pointer classes. You can pass them around efficiently and don't need to worry about deleting them.

The following videos were then shown:

- Edge-Preserving Decompositions:
  - http://www.cs.huji.ac.il/~danix/epd/
- Animating Pictures:
  - http://grail.cs.washington.edu/projects/StochasticMotionTextures/
- Seam Carving:
  - http://swieskowski.net/carve/
  - http://www.faculty.idc.ac.il/arik/site/subject-seam-carve.asp
- Face Beautification:
  - http://leyvand.com/research/beautification/