Image::Scale

High performance image resizing

Andy Grundman
ActiveState

David Maxim Micic - *ECO*

The Problem

- Image resizing libraries are designed for desktop-class systems
- Logitech Media Server needed to run on much slower systems such as these —->
- The Touch may also have to play perfect 24/96 audio at the same time the image cache is being built!

Netgear ReadyNAS Duo
240MHz Sparc clone, no FPU

Marvell SheevaPlug
1.2 GHz ARM9, no FPU

Squeezebox Touch
SD & USB storage support
Freescale i.MX35 533Mhz ARM11, slow FPU

Sparc?!

- Resizing a typical 1500x1500 JPEG album cover on this guy takes…

libgd	34s
ImageMagick	30s

To make matters worse…

- The server needs to cache thumbnails for every album in several different sizes:

40x40 64x64	On-device art
50x50 100x100	Web UI
75x75	iOS/Android

Arcane - *Known/Learned*

libgd & ImageMagick

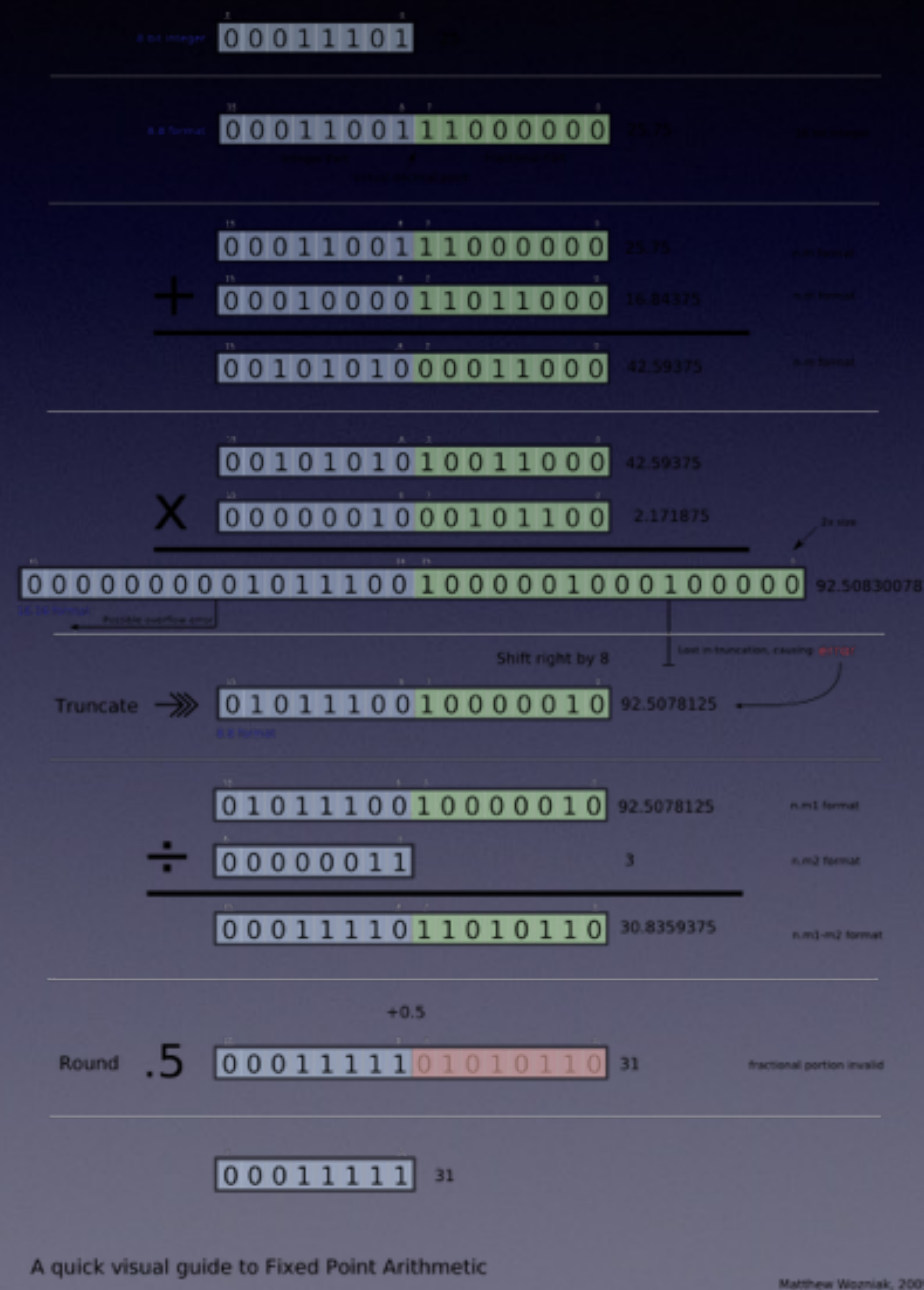
Pros	Cons
High Quality	Floating-point math
IM has over 14 different algorithms to choose from	Inefficient memory use (Touch LMS gets 64MB!)
	No safeguards against loading massive images
	Very difficult to build, especially on Windows

The Solution

- Port the best quality resizing algorithms from both libgd and ImageMagick into fixed-point math.

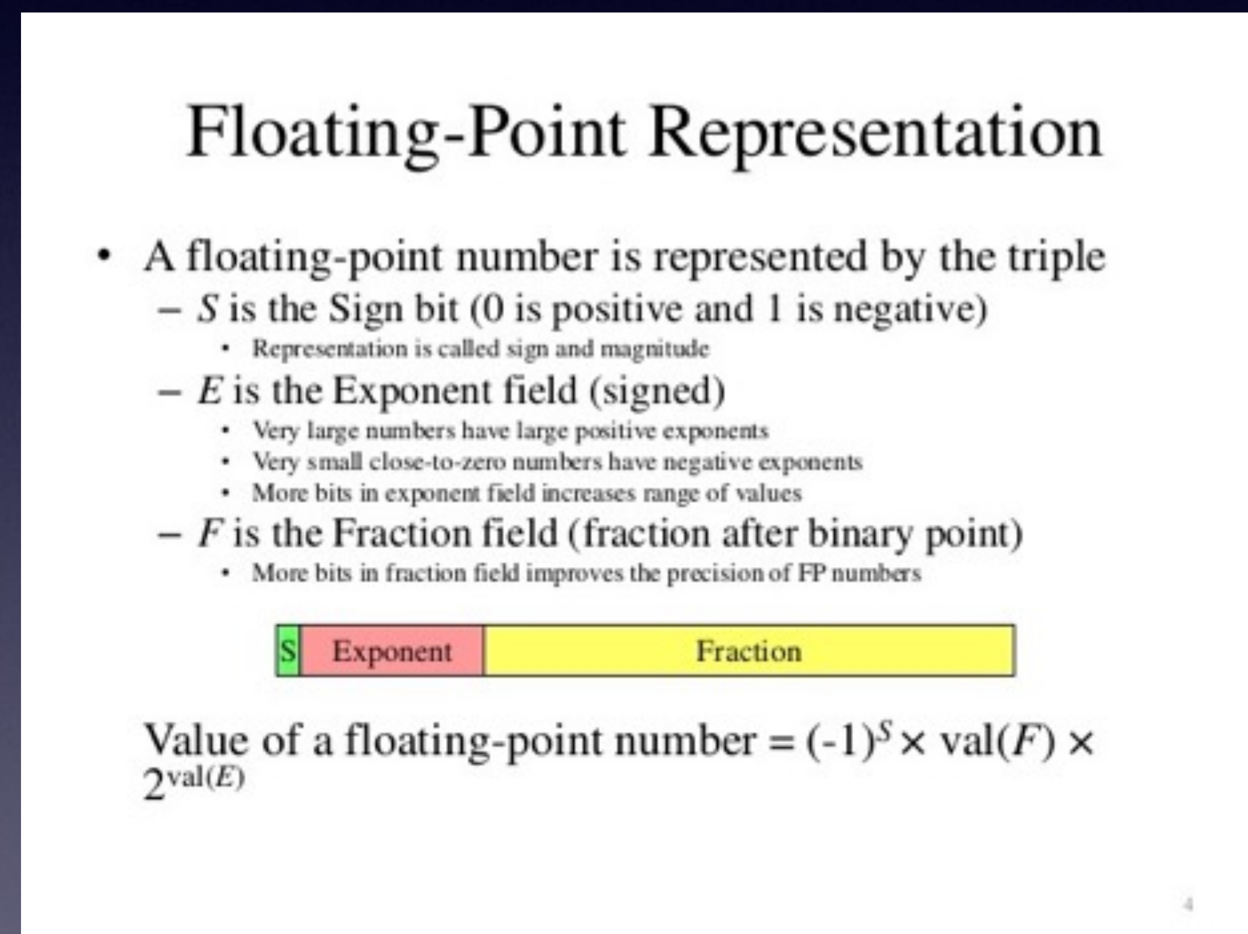
- Support JPEG, PNG, GIF, and BMP.

- my $img = Image::Scale->new('image.jpg'); $img->resize_gd_fixed_point({ width => 150 }); $img->save_jpeg('resized.jpg'); # or save_png()



Fixed-Point Basics

- Stored in a 32-bit signed integer using 19.12 format
- First 19 bits are the integer part, and the remaining 12 are the fractional part. The remaining bit is the sign.
- Range of numbers that can be represented with 19.12: 0.000244140625 to 524287.999755859375
- Math operations retain ~4 decimal places of accuracy, resulting in essentially identical image quality.



Converting to/from fixed

```
#define FRAC_BITS 12

typedef int32_t fixed_t;

static inline fixed_t int_to_fixed(int32_t x) {
  return x << FRAC_BITS;
}

static inline int32_t fixed_to_int(fixed_t x) {
  return x >> FRAC_BITS;
}

static inline fixed_t float_to_fixed(float x) {
  return ((fixed_t)((x) * (float)(1L << FRAC_BITS) + 0.5));
}

static inline float fixed_to_float(fixed_t x) {
  return ((float)((x) / (float)(1L << FRAC_BITS)));
}
```

Math Operations

```c
// Note: gcc

#define FRAC_BITS 12

typedef int32_t fixed_t;

static inline fixed_t fixed_mul(fixed_t x, fixed_t y) {
  return (fixed_t)(((int64_t)x * y) >> FRAC_BITS);
}

static inline fixed_t fixed_div(fixed_t x, fixed_t y) {
  return (fixed_t)(((int64_t)x << FRAC_BITS) / y);
}
```

Mix in some ASM for even more performance!

```c
// This improves fixed-point performance about 15-20% on x86
static inline fixed_t fixed_mul(fixed_t x, fixed_t y) {
  fixed_t __hi, __lo;
  __asm__ __volatile__(
    "imull %3\n"
    "shrdl %4, %1, %0"
    : "=a"(__lo), "=d"(__hi)
    : "%a"(x), "rm"(y), "I"(FRAC_BITS)
    : "cc"
  );
  return __lo;
}

// ARM
static inline fixed_t fixed_mul(fixed_t x, fixed_t y) {
  fixed_t __hi, __lo, __result;
  __asm__ __volatile__(
    "smull %0, %1, %3, %4\n\t"
    "movs %0, %0, lsr %5\n\t"
    "adc %2, %0, %1, lsl %6"
    : "=&r" (__lo), "=&r" (__hi), "=r" (__result)
    : "%r" (x), "r" (y), "M" (FRAC_BITS), "M" (32 - (FRAC_BITS))
    : "cc"
  );
  return __result;
}
```

Other cool features

- JPEG, PNG, and GIF support via libjpeg, libpng, and giflib. libjpeg-turbo (2-4x faster ASM-optimized version) is recommended.
- BMP is also supported, but I think only because I wanted to learn how BMP worked. :)
- Supports JPEG IDCT scaling, which efficiently returns a pre-shrunk version at certain fixed ratios such as 1/2, 1/4, 1/8, etc. Extremely important memory savings.
- Auto-rotates JPEG images that contain EXIF orientation metadata, e.g. iPhone photos.
- The floating-point versions of each resizer are included to make benchmarking easy.

Benchmarks (1/3)

GD copyResampled	1x
resize_gd	3.16x
resize_gd_fixed_point	3.1x

2.4 GHz MacBook Pro (2009 model)
1425x1425 JPEG -> 200x200
libjpeg-turbo v8 with scaling

- On a fast CPU, the original floating-point code edges out the fixed-point version.
- However, in this module it is still over 3x faster than libgd!

Benchmarks (2/3)

GD copyResampled	1x
resize_gd	2x
resize_gd_fixed_point	7.4x

Marvell SheevaPlug 1.2GHz ARM9
1425x1425 JPEG -> 200x200
libjpeg-turbo 6b with scaling

- On a system with no FPU, fixed-point shines

Benchmarks (3/3)

GD copyResampled	1x
resize_gd	1.1x
resize_gd_fixed_point	66x

240MHz Netgear ReadyNAS Duo (Sparc)
JPEG 1425x1425 -> 200x200
libjpeg 6b with scaling

- Not a typo

Image Quality

200x200, enlarged 2x

Float

Fixed

Where?

- http://search.cpan.org/dist/Image-Scale/
- https://github.com/andygrundman/Image-Scale

David Maxim Micic - *EGO*