OpenBR – Open Source Biometric Recognition

and Beyond

Josh Klontz

www.openbiometrics.org

February 17, 2013

Why Open Source?

Why Open Source?

Reproducible Research

Support a common set of file formats and tools for algorithm design, development, and evaluation.

Why Open Source?

Reproducible Research

Support a common set of file formats and tools for algorithm design, development, and evaluation.

Decrease Time to Market

Provide a well-engineered and consistent framework for deploying new algorithms.

Why Open Source?

Reproducible Research

Support a common set of file formats and tools for algorithm design, development, and evaluation.

Decrease Time to Market

Provide a well-engineered and consistent framework for deploying new algorithms.

Reduce Duplication

Supply state-of-the-art baseline components for algorithm design.

Why Open Source?

Reproducible Research

Support a common set of file formats and tools for algorithm design, development, and evaluation.

Decrease Time to Market

Provide a well-engineered and consistent framework for deploying new algorithms.

Reduce Duplication

Supply state-of-the-art baseline components for algorithm design.

Improve Collaboration

Help foster a community where collaboration takes place at the source code level.

What's in it?

Off-the-shelf algorithms

- Face Recognition
- Gender Classification
- Age Estimation
- Commercial Wrappers

What's in it?

Off-the-shelf algorithms

- Face Recognition
- Gender Classification
- Age Estimation
- Commercial Wrappers

Tools for algorithm evaluation

- Standardized set of file formats
- Automatic plot generation
- Command line interface supporting common biometrics tasks

What's in it?

Off-the-shelf algorithms

- Face Recognition
- Gender Classification
- Age Estimation
- Commercial Wrappers

Tools for algorithm evaluation

- Standardized set of file formats
- Automatic plot generation
- Command line interface supporting common biometrics tasks

Software framework for algorithm development

- C++ plugin API for implementing new algorithms
- Grammar for image processing
- Automatic testing, packaging and deployment

Software Architecture

Qt

Cross-platform application and UI framework

OpenCV

Image processing library

Eigen

Linear algebra library

CMake

Cross-platform build system

Software Architecture

Qt

Cross-platform application and UI framework

OpenCV

Image processing library

Eigen

Linear algebra library

CMake

Cross-platform build system

br

Command line application for running algorithms and evaluating results.

C API

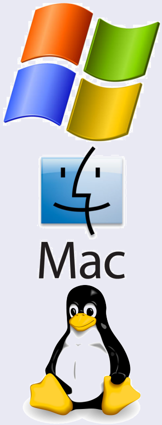
High-level interface for other programming languages.

C++ Plugin API

Core interface for using and developing algorithms.

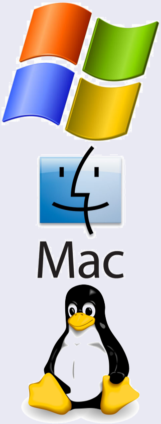
Supported Platforms

Now

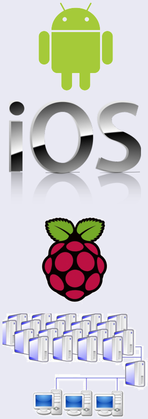


Supported Platforms

Now



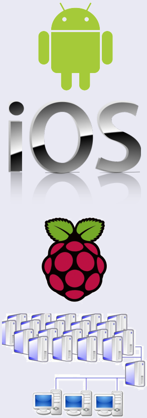
Soon



Supported Platforms

Now

Soon



Future

intel inside
Xeon Phi
NVIDIA
ACCELERATED
A·SERIES
PROCESSOR
AMD

Algorithm Evaluation

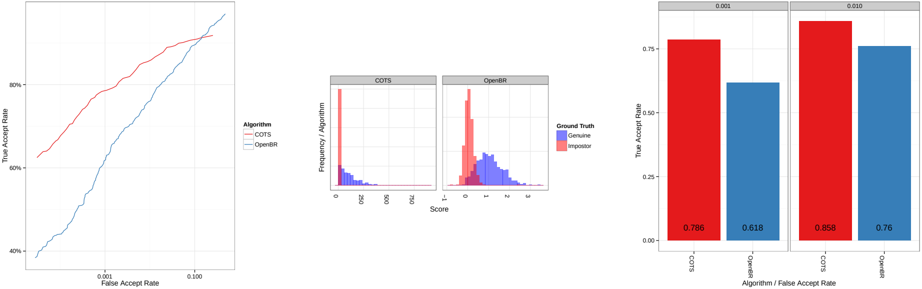


Figure: OpenBR vs COTS face recognition on *MEDS* mugshot database.

Algorithm Evaluation

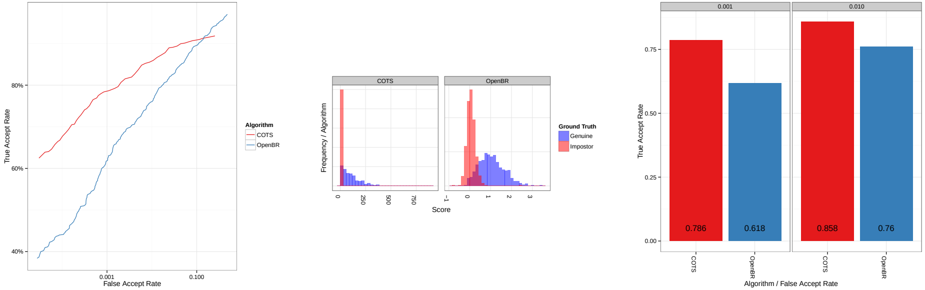
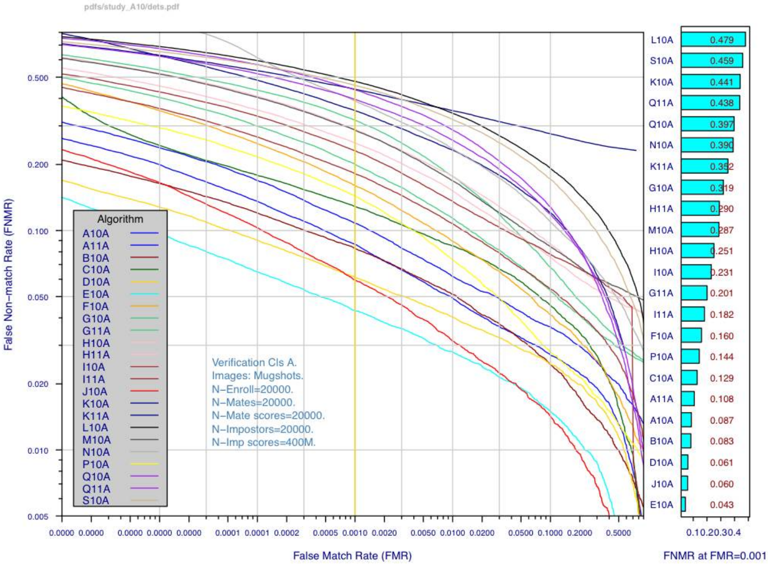


Figure: OpenBR vs COTS face recognition on *MEDS* mugshot database.

	OpenBR	COTS-A	COTS-B	COTS-C	COTS-D
TAR @ FAR = 0.01	0.77	0.93	0.96	0.86	0.80
Template Size (kB)	0.75	2.8	5.0	36	74
Enrollment Speed	10	N/A	N/A	1.3	1.2
Comparison Speed	3,800,000	N/A	110,000	19,000	2,000

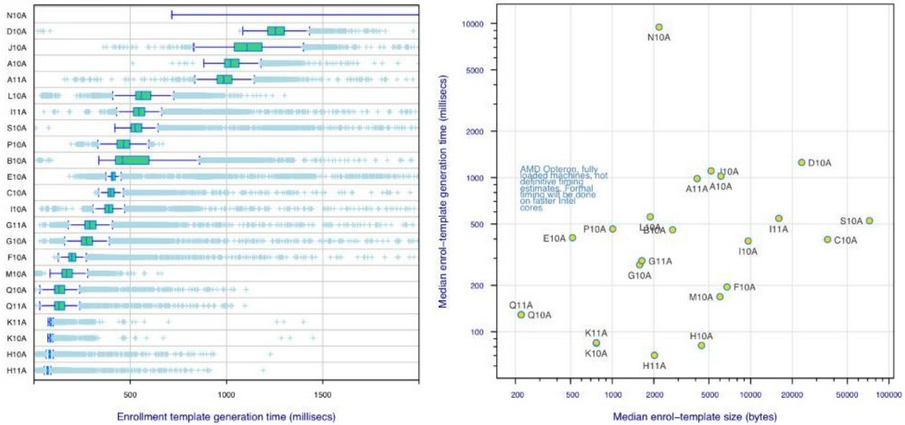
FRVT 2012 (OpenBR = 'K')



Verification Cls A.
Images: Mugshots.
N−Enroll=20000.
N−Mates=20000.
N−Mate scores=20000.
N−Impostors=20000.
N−Imp scores=400M.

FRVT 2012 (OpenBR = 'K')

pdfs/study_A10/ttime_and_tsize.pdf



AMD Opteron, fully loaded machines, not definitive timing estimates. Formal timing will be done on faster Intel cores.

Algorithm Example: Face Recognition

```
$ br -algorithm FaceRecognition -compare me.jpg you.jpg
```

Algorithm Example: Face Recognition

```
$ br -algorithm FaceRecognition -compare me.jpg you.jpg
```

FaceRecognition

FaceDetection!<FaceRegistration>!<FaceExtraction>+<FaceEmbedding>+<FaceQuantization>:UCharL1

Algorithm Example: Face Recognition

```
$ br -algorithm FaceRecognition -compare me.jpg you.jpg
```

FaceRecognition

FaceDetection!<FaceRegistration>!<FaceExtraction>+<FaceEmbedding>+<FaceQuantization>:UCharL1

FaceDetection

Open+Cvt(Gray)+Cascade(FrontalFace)

Algorithm Example: Face Recognition

```
$ br -algorithm FaceRecognition -compare me.jpg you.jpg
```

FaceRecognition

FaceDetection!<FaceRegistration>!<FaceExtraction>+
<FaceEmbedding>+<FaceQuantization>:UCharL1

FaceDetection

Open+Cvt(Gray)+Cascade(FrontalFace)

FaceRegistration

ASEFEyes+Affine(88,88,0.25,0.35)+FTE(DFFS)

Algorithm Example: Face Recognition

```
$ br -algorithm FaceRecognition -compare me.jpg you.jpg
```

FaceRecognition

FaceDetection!<FaceRegistration>!<FaceExtraction>+<FaceEmbedding>+<FaceQuantization>:UCharL1

FaceDetection

Open+Cvt(Gray)+Cascade(FrontalFace)

FaceRegistration

ASEFEyes+Affine(88,88,0.25,0.35)+FTE(DFFS)

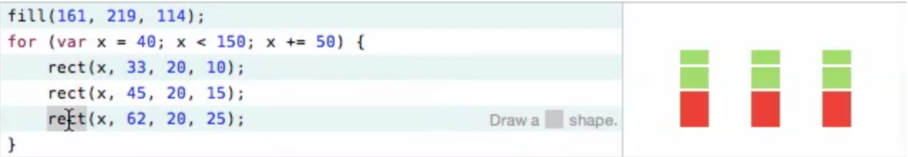
...

FaceEmbedding

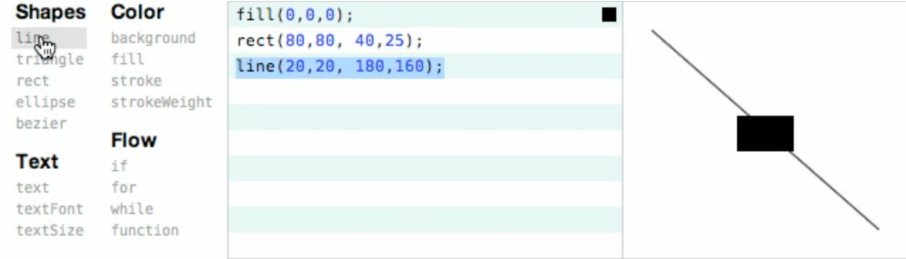
Dup(12)+RndSubspace(0.05,1)+LDA(0.98)+Cat+PCA(768)

Live Coding

Live Coding



```
fill(161, 219, 114);
for (var x = 40; x < 150; x += 50) {
    rect(x, 33, 20, 10);
    rect(x, 45, 20, 15);
    rect(x, 62, 20, 25);
}
```



```
fill(0,0,0);
rect(80,80, 40,25);
line(20,20, 180,160);
```

Inventing on Principle

http://www.youtube.com/watch?v=PUv66718DII

CPU Scaling

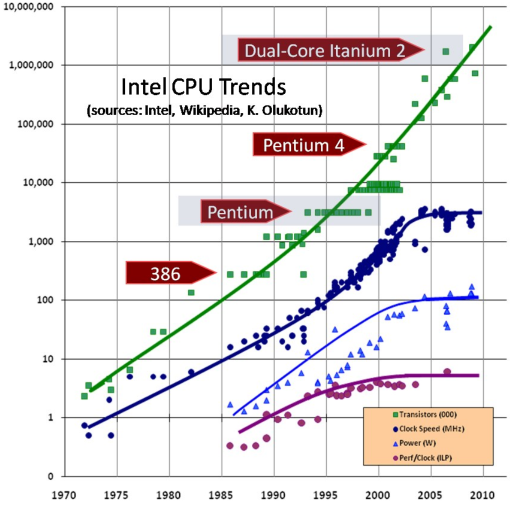


Figure: http://www.extremetech.com/computing/116561-the-death-of-cpu-scaling-from-one-core-to-many-and-why-were-still-stuck

Evolution of Hardware and Software

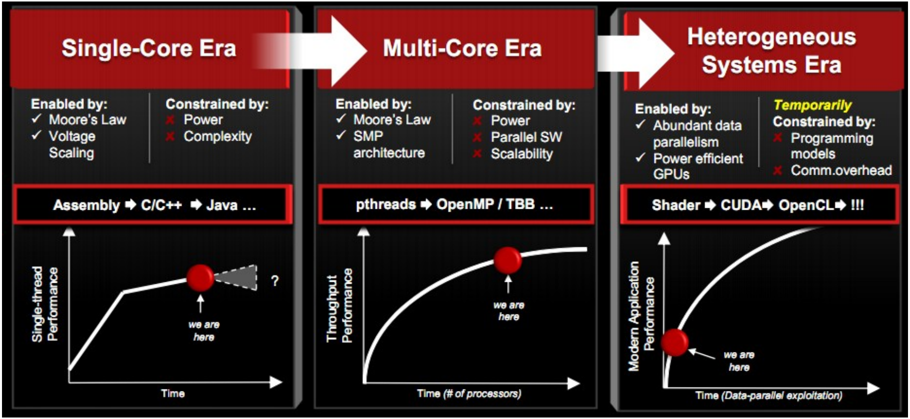


Figure: http://www.extremetech.com/computing/116561-the-death-of-cpu-scaling-from-one-core-to-many-and-why-were-still-stuck

Hardware Realities

Figure: i7 3930k

Figure: GTX 680

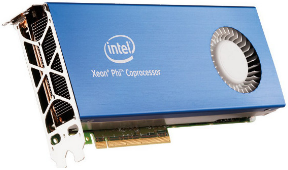


Figure: Xeon Phi 5110p

Hardware Realities

Figure: i7 3930k

$570.00

Figure: GTX 680

$568.50

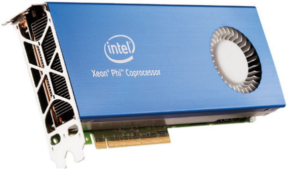


Figure: Xeon Phi 5110p

$2,649

Hardware Realities

Figure: i7 3930k

$570.00

76.8 GFLOPS

Figure: GTX 680

$568.50

1665 GFLOPS

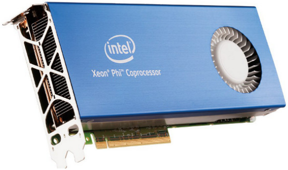


Figure: Xeon Phi 5110p

$2,649

1011 GFLOPS

Hardware Realities

Figure: i7 3930k

$570.00

76.8 GFLOPS

Figure: GTX 680

$568.50

1665 GFLOPS

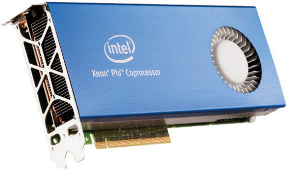


Figure: Xeon Phi 5110p

$2,649

1011 GFLOPS

Gotcha: Memory Bandwidth

12.8 GFLOPS **48.0 GFLOPS** **80 GFLOPS**

Hardware Realities

Figure: i7 3930k

$570.00

76.8 GFLOPS

Figure: GTX 680

$568.50

1665 GFLOPS

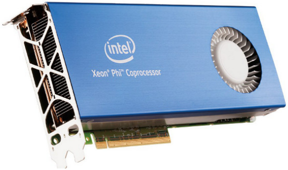


Figure: Xeon Phi 5110p

$2,649

1011 GFLOPS

Gotcha: Memory Bandwidth

12.8 GFLOPS **48.0 GFLOPS** **80 GFLOPS**

Gotcha: Code Duplication

Need a separate code base for optimized performance on each device!

Requirements

What we want

- Write once and run everywhere
- Automatically utilize all available hardware
- Run faster on future hardware

Requirements

What we want

- Write once and run everywhere
- Automatically utilize all available hardware
- Run faster on future hardware

What we need

- Virtual machine or just-in-time compiler
- Express computations using induction variables or “kernels”:

```
void example_kernel(int *a, int *b, int i) { a[i] += b[i]; }
```

Requirements

What we want

- Write once and run everywhere
- Automatically utilize all available hardware
- Run faster on future hardware

What we need

- Virtual machine or just-in-time compiler
- Express computations using induction variables or “kernels”:

```
void example_kernel(int *a, int *b, int i) { a[i] += b[i]; }
```

What we’re proposing

- LLVM IR and JIT compiler
- Designing for OpenCL 2.0 standard

Goals

Perfectly Composable Image Processing Primitives

A grammar for building algorithms from orthogonal primitive kernels with typeless semantics and optimized execution.

Goals

Perfectly Composable Image Processing Primitives

A grammar for building algorithms from orthogonal primitive kernels with typeless semantics and optimized execution.

When we say...

```
Transform *lbpu2 = Transform::make("LBP(1)+U2");
```

Goals

Perfectly Composable Image Processing Primitives

A grammar for building algorithms from orthogonal primitive kernels with typeless semantics and optimized execution.

When we say...

```
Transform *lbpu2 = Transform::make("LBP(1)+U2");
```

...we mean

Give me a pointer to a function that computes $LBP_{8,1}^{u2}$ on an image, minimizes main memory transactions by combining kernels, and is optimized for parallel execution on the hardware available.

Goals

Perfectly Composable Image Processing Primitives

A grammar for building algorithms from orthogonal primitive kernels with typeless semantics and optimized execution.

When we say...

```
Transform *lbpu2 = Transform::make("LBP(1)+U2");
```

...we mean

Give me a pointer to a function that computes $LBP_{8,1}^{u2}$ on an image, minimizes main memory transactions by combining kernels, and is optimized for parallel execution on the hardware available.

Take-Home Message

$Compilation = Source\ Code + Available\ Hardware + First\ Image$

The End

Slides

openbiometerics.org/slides.pdf

Source

github.com/biometrics/openbr

E-mail

openbr-dev@googlegroups.com

Thank You!

Welcome to the Parallel Jungle!

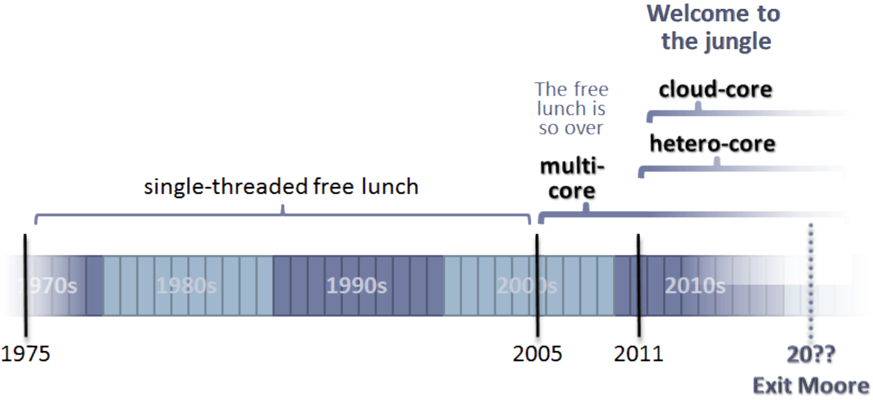


Figure: http://www.drdobbs.com/parallel/welcome-to-the-parallel-jungle/232400273

Plugin Example: Local Binary Patterns

Plugin Example: Local Binary Patterns

```
#include <openbr_plugin.h>
```

Plugin Example: Local Binary Patterns

```
#include <openbr_plugin.h>
class LBP : public Transform {
```

Plugin Example: Local Binary Patterns

```
#include <openbr_plugin.h>
class LBP : public Transform {
  BR_PROPERTY(int, radius, 1)
```

Plugin Example: Local Binary Patterns

```
#include <openbr_plugin.h>
class LBP : public Transform {
 BR_PROPERTY(int, radius, 1)
 void project(const Matrix &src, Matrix &dst) const {
```

Plugin Example: Local Binary Patterns

```
#include <openbr_plugin.h>
class LBP : public Transform {
  BR_PROPERTY(int, radius, 1)
  void project(const Matrix &src, Matrix &dst) const {
    for (int r=radius; r<src.rows-radius; r++)
      for (int c=radius; c<src.cols-radius; c++) {
        float cval = p[r*src.cols+c];
        dst(r, c) =
          (p[(r-radius)*src.cols+c-radius] >= cval ? 128 : 0) |
          (p[(r-radius)*src.cols+c] >= cval ? 64 : 0) |
          ...;
      }
```

Plugin Example: Local Binary Patterns

```
#include <openbr_plugin.h>
class LBP : public Transform {
  BR_PROPERTY(int, radius, 1)
  void project(const Matrix &src, Matrix &dst) const {
    for (int r=radius; r<src.rows-radius; r++)
      for (int c=radius; c<src.cols-radius; c++) {
        float cval = p[r*src.cols+c];
        dst(r, c) =
          (p[(r-radius)*src.cols+c-radius] >= cval ? 128 : 0) |
          (p[(r-radius)*src.cols+c] >= cval ? 64 : 0) |
          ...;
      }
  }
};
```

Plugin Example: Local Binary Patterns

```
#include <openbr_plugin.h>
class LBP : public Transform {
  BR_PROPERTY(int, radius, 1)
  void project(const Matrix &src, Matrix &dst) const {
    for (int r=radius; r<src.rows-radius; r++)
      for (int c=radius; c<src.cols-radius; c++) {
        float cval = p[r*src.cols+c];
        dst(r, c) =
          (p[(r-radius)*src.cols+c-radius] >= cval ? 128 : 0) |
          (p[(r-radius)*src.cols+c] >= cval ? 64 : 0) |
          ...;
      }
  }
};
BR_REGISTER(Transform, LBP)
```

Plugin Example: Local Binary Patterns

```
#include <openbr_plugin.h>
class LBP : public Transform {
  BR_PROPERTY(int, radius, 1)
  void project(const Matrix &src, Matrix &dst) const {
    for (int r=radius; r<src.rows-radius; r++)
      for (int c=radius; c<src.cols-radius; c++) {
        float cval = p[r*src.cols+c];
        dst(r, c) =
          (p[(r-radius)*src.cols+c-radius] >= cval ? 128 : 0) |
          (p[(r-radius)*src.cols+c] >= cval ? 64 : 0) |
          ...;
      }
  }
};
BR_REGISTER(Transform, LBP)
...
Transform *lbp = Transform::make(”LBP(1)”);
```

Plugin Example: Local Binary Patterns

```
#include <openbr_plugin.h>
class LBP : public Transform {
  BR_PROPERTY(int, radius, 1)
  void project(const Matrix &src, Matrix &dst) const {
    for (int r=radius; r<src.rows-radius; r++)
      for (int c=radius; c<src.cols-radius; c++) {
        float cval = p[r*src.cols+c];
        dst(r, c) =
          (p[(r-radius)*src.cols+c-radius] >= cval ? 128 : 0) |
          (p[(r-radius)*src.cols+c] >= cval ? 64 : 0) |
          ...;
      }
  }
};
BR_REGISTER(Transform, LBP)
...
Transform *lbp = Transform::make(”LBP(1)”);
Transform *lbpu2 = Transform::make(”LBP(1)+U2”);
```