Using software-configurable processors in biometric applications

By Philip Weaver and Fred Palma

Software-Configurable Processors (SCPs) are becoming an increasingly popular solution to challenging compute problems, spanning the gap between a general purpose processor and a dedicated digital signal processor. The benefits are especially clear for implementing biometric algorithms, which are complex and adaptive. This article describes an architecture for facial recognition, mapping C++ algorithms onto an SCP.

Humans detect and identify faces in a scene with little or no effort. However, building an automated system to accomplish this task has proved quite challenging due to the computational requirements of the algorithms. There has been a sharp increase in the public's interest for face recognition systems that are both reliable and non-intrusive. Such systems could make criminal identification, monitoring, and security access systems more effective. Unlike any other biometric technologies – such as fingerprint or iris scanning – face recognition systems can function in a real-time environment with little to no cooperation from the subject, who may not even be aware that the scanning is taking place.

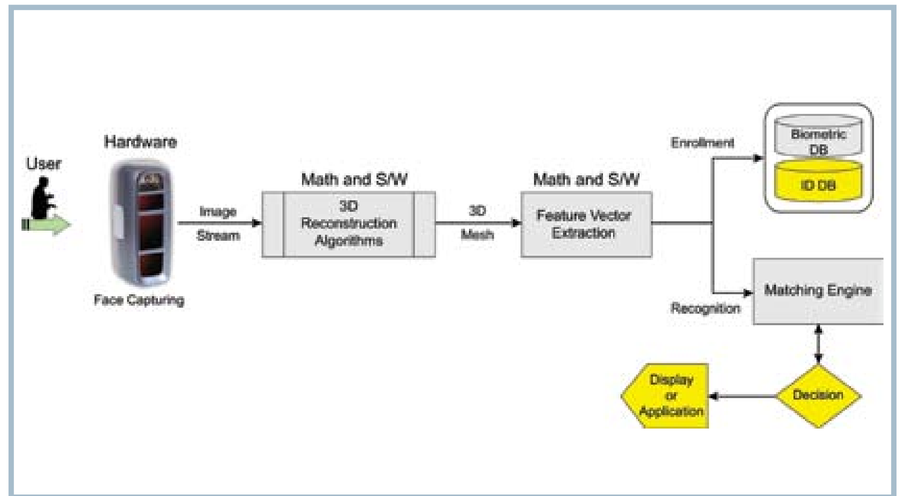

Figure 1

The core technology of a face recognition system consists of several steps outlined in Figure 1. First, a stream of 2D camera input is processed into 3D surface data. The 3D surface is then examined further to identify and measure facial features. These measurements are then distilled into a biometric template. This template is then compared with a database of previously enrolled templates. Faces are recognized based on how closely they match those already stored in the database. Each of these steps has their own set of data processing and computational requirements that challenge typical embedded processors.

Computing the 3D surface

A4Vision's proprietary hardware acquires surface data using structured light. A pattern of light is projected onto the subject's face while a camera observes

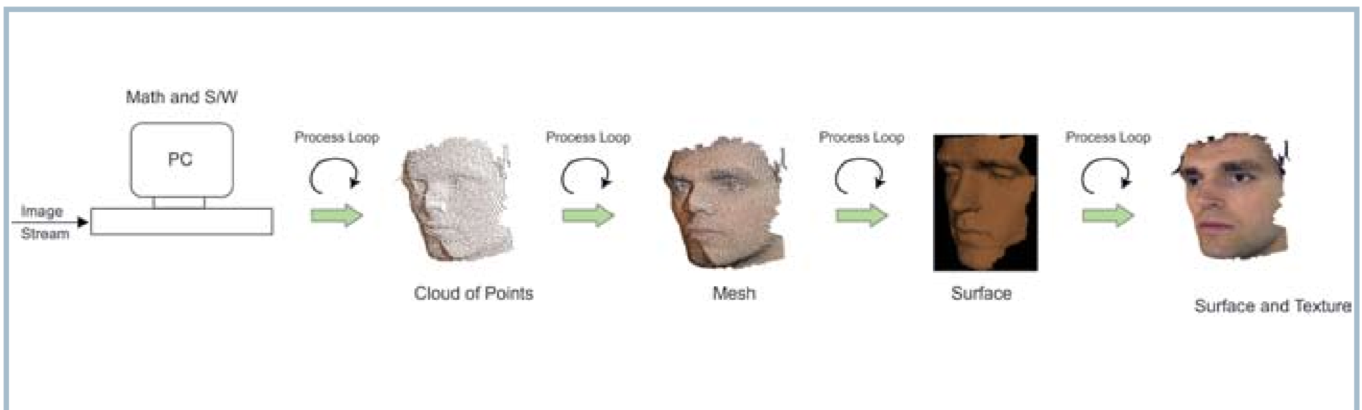

Figure 2

the pattern. The observed pattern is distorted, and this distortion depends on the individual geometry of the face. The pattern can also be projected using wavelengths that are outside the visible light spectrum, which allows the image to be acquired without the awareness of the subject.

Because the distortion of the structured light pattern depends on the individual geometry of the face, the 3D surface can be computed based on the image stream from the camera. First, image processing techniques are used to reduce noise and identify the projected pattern. Once the pattern has been identified, the x, y, and z coordinates of each point on the surface can be determined using trigonometry and linear algebra. The algorithm initially computes a cloud of points, and then links the points into a mesh. The reconstruction of the 3D surface is illustrated in Figure 2. Additional information about the surface and texture allows a realistic likeness to be recreated. Computing more points in the original cloud allows the system to produce a more accurate surface, but also requires more calculations.

Facial feature extraction and recognition

Further analysis of the facial surface allows features such as the forehead, eyes, and nose to be extracted. Based on the locations of these features, key points on the face are identified and the distances between them measured. The points chosen depend on rigid skeletal structures that typically do not change over time (as opposed to soft tissues that change as a person ages or fluctuates in body weight). These measurements are stored as a biometric template in an enrollment database. A face is identified in real-time by matching the biometric template against this database.

Meeting computational requirements

Developing an embedded system to perform real-time 3D face recognition is challenging. As algorithms continue to evolve, development using a high-level language such as C is desirable. General purpose processors such as the popular RISC-based cores widely used for embedded systems generally have high-quality C and C++ tools well suited to algorithm development, but are unable to perform the demanding computational tasks demanded by the algorithms. The fastest DSPs available may be able to support the computations, but their complex instruction sets typically require manual optimization of assembly code. The time required for this optimization dramatically slows down the development process and must be repeated when the algorithm changes.

Software-configurable processor

Stretch software-configurable processors deliver both flexibility for developing algorithms and computational performance to them in real time. The SCP consists of a RISC core and configurable logic working together seamlessly as a single processor. Its programming model is entirely C and C++ based, thus allowing for rapid development of new algorithms. The configurable logic also allows new custom instructions to replace functions that otherwise require tens or hundreds of operations to complete. This logic allows ordinary C or C++ code to be accelerated, without manually optimizing assembly code, to speeds that often exceed those attainable with standard DSPs.

The basis of SCP acceleration is instruction specialization, instruction pipelining, data bandwidth, and data parallelism. The SCP compiler tools create a single instruction from a C or C++ function containing multiple operations. The optimizing compiler then schedules the new instructions together with the standard instructions to minimize stalls in the processor.

The performance benefits of the custom instructions are substantial, and are multiplied when the instructions perform multiple calculations in parallel. The SCP architecture includes 128-bit wide register files that can hold multiple data objects to facilitate the parallel processing.

Because the tool chain used to program the SCP requires no assembly language programming or logic design, changes to the algorithm can be implemented by changing the C or C++ code that implements them. Algorithms can therefore be developed and refined in a high-level language and accelerated easily. This ability makes the SCP a unique platform for biometric algorithms.

Computing the 3D coordinates

To illustrate how the SCP architecture delivers high performance for face recognition, consider one portion of the algorithm that computes a 3D facial surface from a 2D video image using structured light. This part of the algorithm solves for the x, y, and z coordinates using three equations. The calculation is performed using Cramer's Rule:

Given the system of equation:

$$\begin{aligned} a_1x + b_1y + c_1z &= d_1 \\ a_2x + b_2y + c_2z &= d_2 \\ a_3x + b_3y + c_3z &= d_3 \end{aligned} \quad \text{where the determinant } D = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} \neq 0$$

Then

$$D_x = \begin{vmatrix} d_1 & b_1 & c_1 \\ d_2 & b_2 & c_2 \\ d_3 & b_3 & c_3 \end{vmatrix} \quad D_y = \begin{vmatrix} a_1 & d_1 & c_1 \\ a_2 & d_2 & c_2 \\ a_3 & d_3 & c_3 \end{vmatrix}$$

$$D_z = \begin{vmatrix} a_1 & b_1 & d_1 \\ a_2 & b_2 & d_2 \\ a_3 & b_3 & d_3 \end{vmatrix}$$

$$x = \frac{D_x}{D} \quad y = \frac{D_y}{D} \quad z = \frac{D_z}{D}$$

The determinant D is computed using expansion by minors:

$$\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} = a_1 \begin{vmatrix} b_2 & c_2 \\ b_3 & c_3 \end{vmatrix} - a_2 \begin{vmatrix} b_1 & c_1 \\ b_3 & c_3 \end{vmatrix} + a_3 \begin{vmatrix} b_1 & c_1 \\ b_2 & c_2 \end{vmatrix}$$

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

Representative C code is shown in Listing 1. A 3x4 matrix *m* holds the coefficients of the three equations. The *minors* array holds minor determinants used to compute *D*, *Dx*, *Dy*, and *Dz* as shown previously. From these values *x*, *y*, and *z* are determined and placed in the vector *X*.

```
// Non-accelerated version
// Compute partial determinants of
A needed for Cramer's rule.
minors[0][0] = (( m[1][1]*m[2][2]-
m[2][1]*m[1][2] )>>31);
minors[1][0] = (( m[2][1]*m[0][2]-
m[0][1]*m[2][2] )>>31);
minors[2][0] = (( m[0][1]*m[1][2]-
m[1][1]*m[0][2] )>>31);
minors[0][1] = (( m[1][2]*m[2][0]-
m[2][2]*m[1][0] )>>31);
minors[1][1] = (( m[2][2]*m[0][0]-
m[0][2]*m[2][0] )>>31);
minors[2][1] = (( m[0][2]*m[1][0]-
m[1][2]*m[0][0] )>>31);
minors[0][2] = (( m[1][0]*m[2][1]-
m[2][0]*m[1][1] )>>31);
minors[1][2] = (( m[2][0]*m[0][1]-
m[0][0]*m[2][1] )>>31);
minors[2][2] = (( m[0][0]*m[1][1]-
m[1][0]*m[0][1] )>>31);
// Compute D
D = ((m[0][0]* minors[0][0]+m[1][0]*
minors[1][0]+ m[2][0]* minors[2][0])>>26);
// Compute 1/D
invD = (1<<SCALEFACTOR)/D;
// Compute Dx and x
Dx = ((m[0][3]* minors[0][0]+m[1][3]*
minors[1][0]+m[2][3]* minors[2][0])>>26);
x = ((Dx * invD)>>SCALEFACTOR);
// Compute Dy and y
Dy = ( (m[0][3]* minors[0][1]+m[1][3]*
minors[1][1]+ m[2][3]* minors[2][1])>>26);
y = ((Dy*invD)>>SCALEFACTOR);
// Compute Dz and z
Dz = ((m[0][3]*minors[0][2]+m[1][3]*mino
rs[1][2]+ m[2][3]*minors[2][2])>>26);
z = ((Dz*invD)>>SCALEFACTOR);
// Store coordinate in vector.
X[0] = x;
X[1] = y;
X[2] = z;
```

Listing 1

In Listing 2, the matrix computations have been converted to use custom instructions. The custom instructions are written in Stretch C. Because Stretch C supports all of the data types and operators in C, the custom instructions are specified using code nearly identical to that shown in Listing 1. The code that calculates x, y, and z now uses special macros to load 128-bit Wide Registers (WRs), perform the various parts of the calculation, and store the results to the vector X. In the code listing, the WRGETxINIT() macros initialize streams of data to be loaded into the WRs, while the WRAGETxI each load a row of data from memory for processing. Custom instructions then compute the minors and use them to compute the determinants. Custom instructions are then used to perform the divide and multiplies that determine x, y, and z. Finally, the WRPUTINIT, WRPUT, and WRPUTFLUSH instructions write the resulting vector X to memory.

The combination of data streaming and wide data registers provides a high bandwidth path to the configurable logic that implements the instructions. Seven specialized instructions have been crafted to replace the large number of instructions a compiler would create to implement the code in Listing 1. For this computationally-intensive set of operations, the Stretch SCP processor reduced the number of compute cycles by a factor of nine.

```
// Accelerated version
// Load three rows of matrix data
WRGET0INIT(0, (void *)&m[0]);
WRGET1INIT(0, (void *)&m[1]);
WRGET2INIT(0, (void *)&m[2]);
WRAGET0I(&row1, 16);
WRAGET1I(&row2, 16);
WRAGET2I(&row3, 16);
// Compute intermediate values for determinants
EI_COMPUTE_MINORS_0(row1,
row2, row3, &minorsForDet_1,
&minorsForDetAndDet1_2);
EI_COMPUTE_ MINORS _1(row1, row2,
row3, &unused, &minorsForDet2_2);
EI_COMPUTE_ MINORS _2(row1,
row2, row3, &minorsForDet1to3_1,
&minorsForDet3_2);
// Compute determinants
EI_COMPUTE_DET(mc0,
minorsForDetAndDet1_2, &D);
EI_COMPUTE_DET(minorsDet1to3_1,
minorsForDetAndDet1_2, &Dx);
EI_COMPUTE_DET(minorsForDet1to3_
1, minorsForDet2_2, &Dy);
EI_COMPUTE_DET(minorsForDet1to3_
1, minorsForDet3_2, &Dz);
// Invert det for multiply
EI_INVERT(D);
EI_INPUT_DET123(Dx, Dy, Dz);
EI_COMPUTE_X(&xyz);
// Store result
WRPUTINIT(0, X);
WRPUTI(xyz, 12);
WRPUTFLUSH();
```

Listing 2

This example demonstrates the process of accelerating one part of the face recognition algorithm using the software-configurable processor. This process can be repeated throughout this or any other application. The resulting acceleration allows computationally-intensive algorithms to run at high speeds using only C / C++ software programming methods.

Conclusion

The ability to deliver a face recognition system unaffected by lighting conditions, background colors, facial hair, or makeup goes a long way toward meeting public demands for better monitoring systems, security systems, and criminal identification. 3D face recognition systems can meet this need, but one of the barriers to bringing such systems to larger markets is the requirement for a flexible and powerful embedded architecture. The software-configurable processor, however, has been demonstrated to enable this biometric application to move from desktop computer systems into an embedded platform. Furthermore, the SCP will continue to be used to accelerate the compute intensive section to enable different viewing angles for higher accuracy in real-life deployments as well as new algorithms for improved identification.

Philip A. Weaver *is a Member of Technical Staff, Applications Engineering, for Stretch Inc. He has vast technical experience in software, hardware, and mechanical systems. He was vice president of Engineering for Luidia, Inc., and has held senior engineering and management roles at Electronics for Imaging, Maxoptix Corporation and Quantum Corporation.*

To learn more, contact Philip at:
Stretch Inc.
777 E. Middlefield Road
Mountain View, CA 94043
Tel: 650-864-2707
E-mail: pweaver@stretchinc.com
Website: www.stretchinc.com

Fred Palma *is Vice President of Engineering for A4Vision. He has extensive senior management experience in leading development efforts for several commercial applications in card base identification, access control, and card personalization systems. He was a director of Engineering and Product Management for Diebold Card Systems, and vice president of Information Systems for Oberthur Card Systems, US.*

To learn more, contact Fred at:
A4Vision
840 W. California Avenue, Suite 200
Sunnyvale, CA 94086
Tel: 408-329-4566
E-mail: fred.palma@a4vision.com
Website: www.a4vision.com