GPU Accelerated Sheet Forming Grid Measurement

Michael Kinsner[1], Allan Spence[2] and David Capson[3]

[1]McMaster University, kinsnemh@mcmaster.ca
[2]McMaster University, adspence@mcmaster.ca
[3]McMaster University, capson@mcmaster.ca

ABSTRACT

Sheet forming processes frequently require prediction, measurement, and analyses of part dimensional and strain values. Although FEA can be used to guide initial designs, measurement and analyses of actual forming results is also required – particularly for validation when new materials are proposed, or for urgent corrective action when material property variations occur during production. For greatest convenience in an industrial environment, manual manipulation of the computer vision camera based measurement system is preferred, requiring the associated data processing algorithms to operate at camera video frame rates, and remain robust in the presence of poorly focused image data. This paper presents a practical approach for high speed extraction of grid measurement data from camera video image sequences. To overcome the speed limitations of conventional microprocessors, and to ensure that no video frames are skipped, Graphics Processing Unit (GPU) hardware is used to accelerate the algorithm. Results are presented demonstrating the reliability and throughput obtained.

Keywords: strain analysis, grid extraction, graphics processing unit (GPU).
DOI: 10.3722/cadaps.2010.675-684

1 INTRODUCTION

During sheet forming processes, a flat sheet is stretched over a die to create the desired part shape. Excessive (positive strain) stretching will tear the material. Negative strain will result in buckling. When a new material is proposed, a sample is formed into a standard dome shape, and its strain properties are experimentally measured and entered into a Finite Element Analysis (FEA) database. Design stage FEA methods are used to predict final part dimensions and strain values, and to establish the nominal shape of the forming die. Prior to initial manufacturing, the die is "tried out" and adjusted as needed to obtain in tolerance part dimensions and strain values. During volume manufacturing, material variation or related problems may require urgent "courtesy" measurement and analysis, so that immediate corrective action can be applied.

Although industrial touch probe and laser scanner based dimensional measurement methods are now well established, predominant strain measurement techniques remain manual, with automation restricted to handling a very small part area, or only a test dome specimen [13]. This is insufficient to confidently establish material properties, and too slow for production line use. As a result,

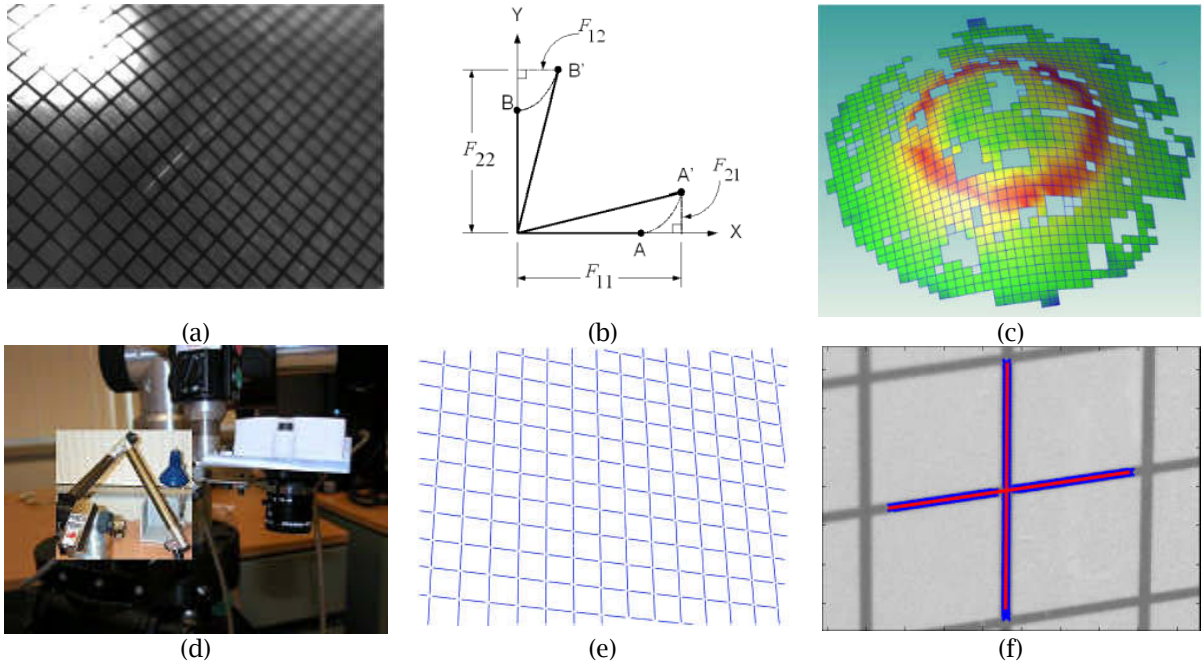


Fig. 1: (a) Stamped sheet metal dome with deformed square grid, (b) Grid distortion mathematics, (c) Example dome strain analysis display showing dimensional reconstruction and overlaid thickness strain [13], (d) Camera head mounted on FARO portable arm CMM, (e) Detected grid lines, (f) Parabola fitting (red lines) and sub-pixel intersection detection.

uncertainty in design stage FEA prediction remains, and urgent production problems are solved in an expensive and wasteful ad hoc fashion. As is detailed below, the primary automation impediment is the time consuming grid measurement data processing step. Previously, this impediment made processing at video frame rate speeds impractical.

To accelerate data processing to interactive speeds, a Graphical Processing Unit (GPU) solution is developed in the remainder of this paper. Section 2 briefly reviews sheet forming dimensional and strain measurement. Section 3 describes the computationally intensive grid intersection detection algorithm. Section 4 introduces the NVIDIA GPU architecture. Section 5 describes mapping of the scale-space algorithm to GPU hardware. Section 6 presents results from real data acquisitions. The paper concludes in Section 7.

2 SHEET FORMING DIMENSIONAL AND STRAIN MEASUREMENT

Sheet forming measurement begins with printing of a square grid on the flat, undeformed part specimen. To obtain necessary dimensional resolution, a grid intersection spacing of 0.1 inches (2.54 mm) is common. A standard technique for computation of surface strain values is based on deformation of the corner points (Fig. 1(a)) [12]. Fig. 1(b) illustrates distortion of corner points $A=\left(X_1,Y_1\right), B=\left(X_2,Y_2\right)$ to $A'=\left(x_1,y_1\right), B'=\left(x_2,y_2\right)$ during deformation. Maintaining the lower left vertex at the origin and applying a constant second order tensor $\mathbf{F}_{2\times2}$ results in the linear systems

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ Y_1 \end{bmatrix}; \quad \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix} \cdot \begin{bmatrix} X_2 \\ Y_2 \end{bmatrix} \tag{2.1}$$

Rearranging,

$$\begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} X_1 & Y_1 & 0 & 0 \\ X_2 & Y_2 & 0 & 0 \\ 0 & 0 & X_1 & Y_1 \\ 0 & 0 & X_2 & Y_2 \end{bmatrix} \cdot \begin{bmatrix} F_{11} \\ F_{12} \\ F_{21} \\ F_{22} \end{bmatrix} \tag{2.2}$$

can be solved for $\mathbf{F}$. The surface true strains $\varepsilon_{1,2}$ satisfy the relationship $\varepsilon_{1,2} = \ln\left(\lambda_{1,2}\right)$, where $\lambda_{1,2}$ are the eigenvalues of $\mathbf{F}$. The thickness (third) strain ε_3 is calculated using $\varepsilon_3 = -\varepsilon_1 - \varepsilon_2$. Fig. 1(c) shows the thickness strain from previous work [13], in which a deformed sheet metal dome was analyzed.

To compete with manual systems, strain resolution to one percent is required. This implies a digitizing resolution of 0.025 mm. A standard 1024 x 768 pixel camera will therefore capture approximately one square inch per image, operating at a range of 25 mm above the part surface. The authors have previously reported a pre-programmed bridge style Coordinate Measuring Machine (CMM) based solution [14], but such a design is not portable or conveniently manipulated. A better system for industrial deployment is a FARO or equivalent portable arm CMM with a camera mounted to the tool point (Fig. 1(d)). The system operates in either initial calibration or measurement mode. The major data processing workflow steps are to:

- detect the grid intersections (Fig. 1(e-f));
- follow the manual FARO arm motion (interframe motion tracking)
- calibrate the camera and FARO/camera parameters
- triangulate the location of the intersections into 3-D space
- compute the strain values, and display 3-D results.

Calibration mode includes determination of the intrinsic camera [5][17][19] and FARO/camera [2][15][16] geometric parameters, and will be discussed in a separate paper. In either mode, computing the grid intersections is the most computationally intensive step, and must occur at 15 video frame per second rates. Achieving this throughput is the focus of the remainder of the paper.

3 GRID INTERSECTION DETECTION

Because of the need to measure small strain values, accurate detection of the grid corner points is essential. To maintain sufficient resolution, the FARO arm mounted camera must be held close to the part surface. The close range provides the needed magnification, but the corresponding narrow depth-of-field / depth of focus will result in many frames being poorly focused.

To address these requirements, a novel scale-space based approach was implemented. The method consists of three primary steps:

1. Generation of a scale-space representation for each video image frame
2. Detection of the ridge points for all potential grid corner intersections within the frame
3. Parabolic fitting based computation of grid line intersection locations to sub-pixel accuracy

In the following subsections, the proposed algorithm is described together with an analysis of its computational requirements.

3.1 Scale-Space

Multi-scale techniques are used extensively in computer vision applications, particularly when the size / scale of interesting features (in this case, corner grid intersections) is not known in advance. Scale-space techniques provide a framework in which image metrics may be applied to detect features at various scales [18]. A basic scale-space representation of an image $f\left(x,y\right)$ may be defined as $L(x,y;t) = g(x,y;t) * f\left(x,y\right)$ where $g\left(x,y;t\right) = \frac{1}{\sqrt{2\pi t}} e^{-x^2/2t} \cdot \frac{1}{\sqrt{2\pi t}} e^{-y^2/2t}$ represents a Gaussian kernel with

Computation	Add/Subtracts	Multiplications	Square Roots	Powers
Convolution	70	82	0	0
L_q	5	8	3	0
L_{qq}	2	7	0	0
$\delta_t(\varepsilon_{norm}L(x,y;t))$	8	9	0	2
$\delta_{tt}(\varepsilon_{norm}L(x,y;t))$	22	21	0	1
Total/(pixel*scale*frame)	107	127	3	3

Tab. 1: For each frame, this table shows the number of arithmetic operations required per pixel at each scale level. For a 1024 by 768 frame, at 8 scale levels, this equates to 673,185,792 adds/subtracts and 799,014,912 multiplications (1.5 gigaFLOPS)

variance (smoothing parameter) t [1], and $*$ is the convolution operator. Derivatives of this scale-space are then defined as $L_{x^\alpha y^\beta}(x,y;t) = \partial_{x^\alpha y^\beta} L(x,y;t) = g_{x^\alpha y^\beta}(x,y;t) * f\left(x,y\right)$ where α and β define the order of differentiation with respect to the x and y image axes respectively [8].

The scale-space can be viewed as the solution to the diffusion equation $\partial_t L = 0.5\nabla^2 L$ with initial condition $L\left(x,y;0\right) = f\left(x,y\right)$. A semi-discretized diffusion equation solution kernel is defined as

$$L\left(x,y;t\right) = \sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} T\left(m,t\right)T\left(n,t\right)f\left(x-m,y-n\right) \tag{3.1}$$

where $T\left(n;t\right) = e^{-t}I_n\left(t\right)$ [8] and I_n are the modified Bessel functions of integer order. Implementation may use an alternative solution exhibiting minimal rotational asymmetry, and defined as a fixed kernel [7]. The resulting convolution three-kernel, with a smoothing parameter step size of $\Delta t = 1/3$, is

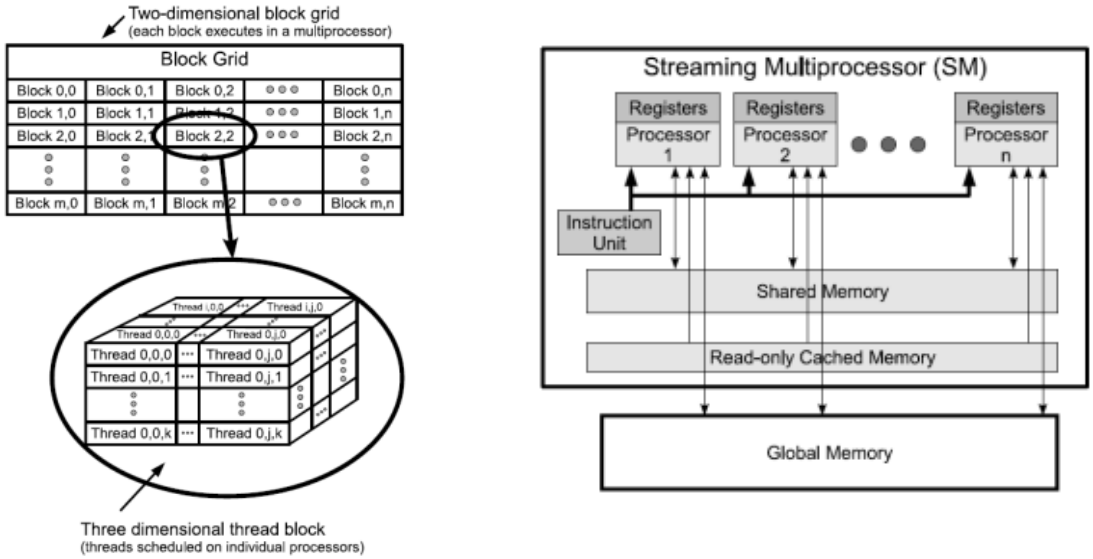
$$k = \left[\begin{array}{ccc} \frac{\Delta t}{2} = \frac{1}{6} & 1-\Delta t = \frac{2}{3} & \frac{\Delta t}{2} = \frac{1}{6} \end{array}\right] \tag{3.2}$$

This scale-space kernel is used to generate all scale-spaces in this work. Eight filter variances are used, resulting in eight scale levels at which features may be detected. When variance steps greater than $\Delta t = 1/3$ are required, repeated self-convolutions of Eqn. (3.2) are used to create a composite kernel. Convolution is implemented in the spatial domain, as shown by Florack [4] to be most accurate for large spatial scales compared with multiplication in the frequency domain.

3.2 Ridge Detection

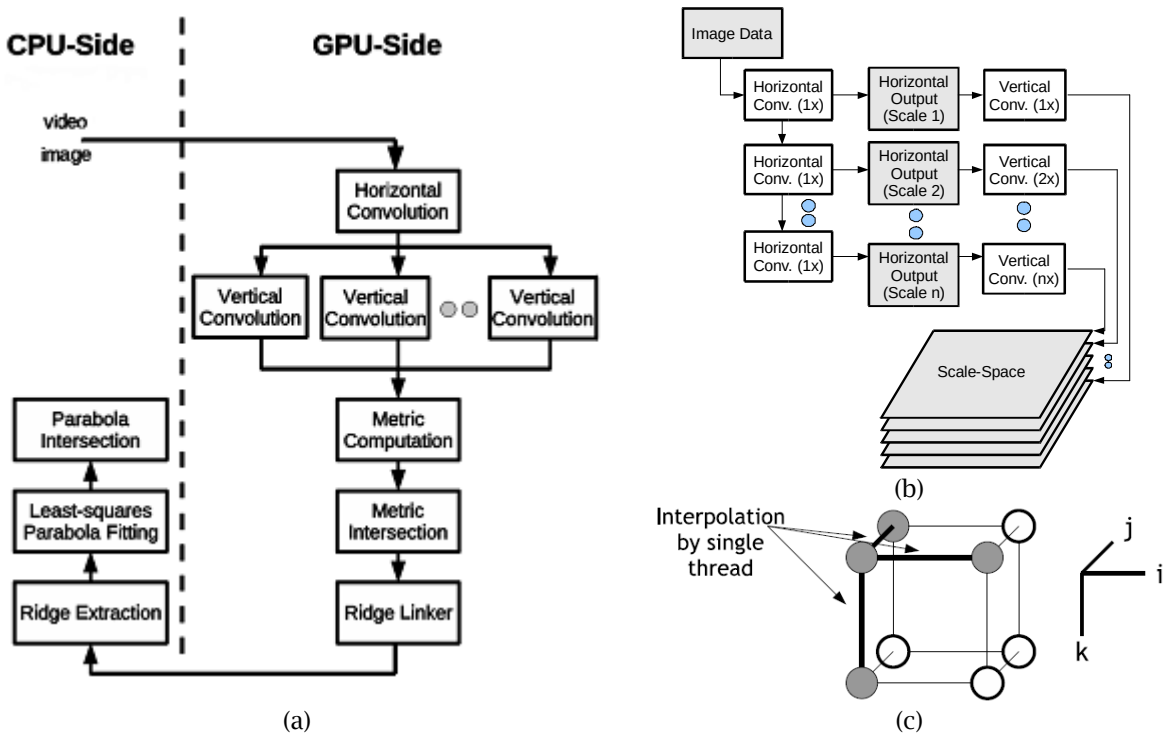
Once a scale-space has been generated from an image, feature detectors can be defined that operate within that space. A multi-scale representation is produced, so that geometric feature detectors may locate the same feature at multiple scales. A method is therefore required to choose the scale at which the feature is best spatially localized. Lindeberg [9] proposed a formulation in which automatic scale selection is incorporated into the feature detection process. This process locates features at the intersections of metric isosurfaces defined in the three dimensional scale-space as formed, for example, by the expression for a dark-line ridge:

$$L_q = 0; \quad L_{qq} < 0; \quad \partial_t\left(\varepsilon_{norm}L\left(x,y;t\right)\right) = 0; \quad \partial_{tt}\left(\varepsilon_{norm}L\left(x,y;t\right)\right) < 0 \tag{3.3}$$



(a) (b)

Fig. 2: GPU Organization: (a) Hierarchical thread organization, (b) Streaming multiprocessor.



(a) (b) (c)

Fig. 3: (a) Division of work between CPU and GPU, (b) Convolution work flow (shaded is stored data), (c) Metric zero interpolation by a single thread.

Kernel Name	GPU Time [mS]	Percent Total Time
Horizontal Convolution	0.9	0.9 %
Vertical Convolution	17.2	17.8 %
Metric Computation	20.9	21.6 %
Metric Intersection	18.4	19.0 %
Ridge Linker	5.5	5.7 %
Memory Transfer	33.9	35.0 %
Total (excluding memory transfer)	62.9	-

Tab. 2: GPU kernel execution times (single frame).

Kernel Name	Registers	Shared Memory	Occupancy	Divergent Branches	Instructions	Serialized Warps
Horizontal Convolution	14	4248 bytes	75%	26	163962	0
Vertical Convolution	9	3236 bytes	100%	3684	877078	0
Metric Computation	15	4968 bytes	100%	6576	2984181	34410
Metric Intersection	17	6844 bytes	88%	58694	2386735	94947
Ridge Linking	13	2268 bytes	88%	32795	660192	47292

Tab. 3: GPU resource utilization and kernel efficiency.

where ε_{norm} is a scale-normalized ridge strength metric, and a *(p,q)* coordinate system is defined at each pixel with axes parallel to the principal curvature of the image intensity at that pixel. The L_q and L_{qq} terms of Eqn. (3.3) define ridge features which may be detected at many or all scales, and the remaining terms define the scale at which a ridge is best localized spatially. By interpolating isosurface intersections of the ridge location and the best localization scale within small volume units (corners defined by neighboring pixels), ridge segments are located and marked. Fig. 1(e) shows an example of ridge segments extracted from one frame of a grid video sequence.

3.3 Subpixel Grid Line Intersection Location

Given the set of grid lines generated through ridge detection, the four line segments surrounding each grid intersection are located and classified into pairs according to their angle relative to the image axes. A parabola is then fit to the pairs of line segments in a least squares sense, and the parabolas are intersected to produce a subpixel estimate of the grid line intersection point. Fig. 1(f) shows an example of the result of the parabola fitting to the four ridge segments surrounding an intersection point. The set of intersections are used to form three-dimensional reconstructions of the scanned part surface through multi-view triangulation [5].

3.4 Computational Effort Required

By analyzing the equations used for creating the scale-space and the metric computations within that space, the data presented in Tab. 1 were obtained. This table shows the number of arithmetic operations required for each pixel, at each scale level. For a representative 1024 by 768 image size at eight scale levels approximately 1.5 gigaFLOPS (1.5 billion Floating Point Operations per Second) will be required. Conventional microprocessors will therefore achieve approximately four frames per second –

well short of the required 15 frame per second rate. To achieve a practical solution, supplementary computational acceleration is therefore required.

4 GPU ARCHITECTURE

Although early research work using GPUs for general purpose computing required algorithm mapping to the conventional graphics processing pipeline, more recent commercial GPU product offerings expose the graphics pipeline in a more general form, greatly simplifying the task of using the computing hardware for parallel algorithms. NVIDIA provides access to the GPU hardware through a C-based interface known as the Compute Unified Device Architecture (CUDA) [11], that is employed within this work.

This GPU is organized in a hierarchical structure, with the most fundamental unit being a single program thread that runs in parallel with other threads. The threads are grouped into a "thread block", and then further grouped into a "block grid", as shown by Fig. 2(a). A thread block is mapped into a Streaming Multiprocessor (SM), the structure of which is shown in Fig. 2(b). Our work used an NVIDIA GTX280 GPU that incorporates 240 processor cores organized into 30 SMs. A minimum of 32 threads run within a streaming multiprocessor in parallel. All share the same instruction codes, and so simultaneously execute the same instruction (but on different pieces of data). This processing model is referred to as Single Instruction Multiple Thread (SIMT).

5 SCALE-SPACE ACCELERATION USING GPU HARDWARE

5.1 Mapping the New Algorithm to the GPU and CPU

Although the scale-space ridge detection algorithm requires sequential ordering of the major processing steps, parallel computing may be used within each step. Recognizing memory access requirements, and the SIMT GPU processing nature, algorithm steps were judiciously assigned to execute either serially on the CPU, or in parallel on the GPU (Fig. 3(a)). The input to this process is a single frame of raw image data from the video sequence, and the output is a set of detected grid line intersection coordinates.

The convolution process data flow is shown in Fig. 3(b), with the shaded boxes representing data that is stored to GPU memory. The separable kernel is prepared on the CPU, and multi-scale horizontal convolutions are then performed on the GPU. Each resulting horizontally convolved image plane is vertically convolved with the relevant kernel also on the GPU, followed by computationally intensive calculation of the ridge metric values defined by Eqn. (3.3). All derivatives are computed through central difference approximation, and custom data formats and conventions were implemented to allow for more efficient processing and storage given the intrinsic single-precision floating point format of the GPU.

After generation of the ridge metric values from the convolved image data, metric iso-surface intersections are computed for each volume unit of the scale space (one volume unit is defined with vertices formed by immediately neighbouring pixels). The work is divided between parallel threads through the approach shown in Fig. 3(c), where a single thread performs interpolation on three edges of a volume unit cube. The remaining edges of a cube are interpolated by neighbouring parallel threads. Built-in GPU functions are used to synchronize before using the intersection results. From the metric iso-surface interpolation across the volume unit edges, intersection points on the cube faces are then linearly interpolated. A bit-field data format is used to efficiently store the results of this process, with individual bits in the data field flagging ridge events that have occurred locally. Results for ambiguous or malformed data are discarded. The bit-field data format is complete for the associated volume unit cube, so it can be used by later processing stages to rapidly determine whether a well formed ridge exists within the cube.

The final stage of processing on the GPU involves linking of individual ridge segments (within individual volume units) into runs of segments that form longer portions of a continuous ridge. The linking process within a multiprocessor operates on a portion of the data that is locally available in memory, and a set of summary data is then produced around the borders of that data. The summary

data follows a custom data convention for fast GPU manipulation, and is later used by the CPU to efficiently link segments of a ridge into a single continuous structure upon which a parabola can be fit.

Final processing stages on the CPU produce completed ridges. By using the summary data output from the GPU, the CPU can rapidly generate ridges, filter short noise ridges out of the data, and then fit parabolas to matching pairs of ridge segments that surround a grid intersection. The sub-pixel locations of parabola intersections are taken as the true grid line intersection locations.

5.2 Memory Transfers

A fundamental bottleneck in the GPU is the transfer of data from the multiple memories into the processor cores. Large latencies, on the order of 300 clock cycles, are experienced when accessing some memories. A significant portion of GPU software development time was required to structure the algorithm so that processing tasks may be performed while a GPU core is waiting for a transfer from memory. Through careful algorithm design, it is possible to hide most memory access times behind processing of data that is already available. The need to hide memory transfers implies that algorithms containing multiple arithmetic operations for each data load operation are the most suitable for GPU implementation. Algorithms not possessing this arithmetic intensity attribute are unlikely to benefit from a parallel GPU implementation.

In addition to the transfer of data within a GPU system, data must also be transferred from the CPU system to the GPU. Current GPU designs allow this data to be asynchronously streamed from the CPU system into mass memory on the graphics board while the GPU is processing data. Given this capability, the GPU can be kept busy with processing almost continuously provided that new frames of image data are being transferred simultaneously with computation of older frames.

6 RESULTS

6.1 Test System

The test system employed was an Intel Core2 Duo (E8400) processor at 3GHz with an NVIDIA GeForce GTX 280 graphics card, running CUDA 2.0 under Windows XP. The test data was composed of 8-bit grayscale video sequences, 1024 by 768 pixels in size. Within the scale-space generation and processing, eight scale levels $t = 1,4,7,10,13,16,19,22$ were used.

6.2 GPU Timing

The actual execution times of the GPU-based computations are shown in Tab. 2, including the memory transfer times between the CPU-system and graphics card over the PCIe bus. Because the memory transfer time is less than the aggregate sum of the remaining task execution times, the transfer of data between the host CPU system and graphics card can indeed be performed in parallel with execution of data on the GPU. With a 62.9 mS frame processing period, the GPU processing rate achieved is better than the 15 video frame per second requirement.

6.3 GPU Resource Utilization and Performance

Memory resource usage on the GPU is presented in the left columns of Tab. 3. Registers are single-cycle local memories available to the processor, and shared memory is a larger memory also with small access times, but requiring very specific access patterns by parallel threads [11]. Occupancy represents the efficiency of the kernel in terms of GPU hardware use, with 100% occupancy being very difficult or impossible to achieve for many real algorithms. Divergent branches occur where a group of parallel threads take multiple paths in a branch, such as some threads executing an “if” statement, while others do not. Instructions are the number of instructions performed within a GPU thread group, and serialized warps relate to the number of times that parallel threads must be processed serially. Smaller numbers of divergent branches and serialized warps result in higher GPU performance.

Convolution requires many more arithmetic operations than the metric computation phase of processing (Tab. 1), but is processed faster as shown in Tab. 2. This performance is caused by the high

(a) (b)

Fig. 4: Programmable Ink Jet Grid Creation [3]: (a) CMM Ink Jet System; (b) Sample Grid applied to Aluminum Sheet.

arithmetic intensity of convolution, combined with the relative simplicity of the memory access patterns allowing serialized warps to be eliminated. Efficient use of the GPU computing resources for these computations allows for high data throughput, thereby reducing the processing time.

6.4 Remaining Steps

The CPU post-processing involves extraction of the GPU-generated ridge data and formation of final ridges in CPU system memory. The overall time required for production of the variable length output ridges in system memory is dependent upon the number of ridges that must be formed, and for images in this application is better than 15 frames per second. Since the post-processing frame rate is faster than the associated GPU frame rate, it does not introduce a bottleneck into the system. Multiple view triangulation, and dimensional/strain calculations can be approximated during data capture to maintain interactive updating of the operator display. Based on past experience, exact final calculation using output from all video frames is not expected to take significant processor time.

For brevity, we have not included details on the camera / FARO arm intrinsic parameter calibration, or on the interframe motion tracking tasks. As noted above, the camara / FARO arm calibration is based on [16][19]. Interframe motion tracking can be difficult when using a regular grid. To overcome this, random fiducial marks were manually added for calibration with a flat plate. For production measurement use, our intention is to substitute ink jet printer dots for the regular square grid (Fig. 4) [3]. Applying these dots at a planned density of 1/96 inch will better support measurement of high strain gradients. Programmatically omitting a dot is equivalent to adding the fiducials that are required for interframe tracking.

7 CONCLUSION

This paper has presented a practical method for sheet forming dimensional and strain measurement at an interactive processing speed. A square grid pattern is printed onto the test part and imaged using a manually manipulated portable arm CMM mounted camera. Adequate resolution requires close-up positioning that must be robustly tracked from frame to frame. Grid lines that are partially out of focus were reliably extracted using scale-space ridge detection methods. Using GPU hardware to accelerate data processing, throughput at 15 frames per second – better than 20 gigaFLOPS – was achieved. The implementation is scalable to newer GPU hardware, and provides an effective solution for similar problems in which arithmetic intensity dominates.

ACKNOWLEDGEMENTS

The authors thank SHARCNET for providing access to GPU hardware and for scholarship funding for M. Kinsner. Discovery Grant funding from the National Sciences and Engineering Research Council of Canada (NSERC) (for A. Spence and D. Capson) is gratefully acknowledged.

REFERENCES

[1] Babaud, J.; Witkin, A. P.; Baudin, M.; Duda, R. O.: Uniqueness of the Gaussian kernel for scale-space filtering, IEEE Transactions of Pattern Analysis and Machine Intelligence, 8(1), 1986, 26–33.

[2] Dornaika, F.; Horaud, R.: Simultaneous Robot-World and Hand-Eye Calibration, IEEE Transactions on Robotics and Automation, 14(4), 1998, 617-622.

[3] El Sahi, S.; Jiang, Y., Spence, A. D.: Oriented 3-D Ink Jet Printing, Computer Aided Design & Applications, 6(3), 2009, 399-406.

[4] Florack, L.: A Spatio-Frequency Trade-Off Scale for Scale-Space Filtering, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(9), 2000, 1050–1055.

[5] Hartley, R.; Zisserman, A.: Multiple View Geometry in Computer Vision, Cambridge University Press, New York, New York, 2000.

[6] Kurfess, T.; T. Tucker, K. A.; Meghashyam, P.M.: GPU for CAD, Computer Aided Design and Applications, 4(6), 2007, 853– 862.

[7] Lim, J.Y.; Stiehl, H.: A Generalized Discrete Scale-Space Formulation for 2-D and 3-D Signals, Scale Space Methods in Computer Vision, 2003, 132–147. Springer Berlin / Heidelberg.

[8] Lindeberg, T.: Scale-Space for Discrete Signals, IEEE Transactions of Pattern Analysis and Machine Intelligence, 12(3), 1990, 234–254.

[9] Lindeberg, T.: Discrete Derivative Approximations with Scale-Space Properties: A Basis for Low-Level Feature Extraction, Journal of Mathematical Imaging and Vision, 3(4), 1993, 349–376.

[10] Lindeberg, T.: Edge Detection and Ridge Detection with Automatic Scale Selection, Proceedings of Computer Vision and Pattern Recognition, 1996, 465–470.

[11] Kirk, D.B.; Hwu, W.-M.W., Programming Massively Parallel Processors, Morgan Kaufmann, Burlington, MA, 2010.

[12] Sowerby, R.; Duncan, J. L.; and Chu, E.: The Modeling of Sheet Metal Stampings, Int. J. Mechanical Sciences, 1986, A28(7), 415-430.

[13] Spence, A. D.; Capson, D. W.; Sklad, M. P; Chan, H.-L.; Mitchell, J. P.: Simultaneous Large Scale Sheet Metal Geometry and Strain Measurement, Trans. ASME, J. Manufac. Sci. and Eng. 130, 2008.

[14] Spence, A. D.; Chan, H. L.; Mitchell, J. P.; and Capson, D. W.: Automotive Sheet Metal and Grid Digitizing Solutions, Computer-Aided Design & Applications, 2, 2005, 135-144.

[15] Strobl, K. H.; Hirzinger, G.: Optimal Hand-Eye Calibration, IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006, 4647-4653.

[16] Strobl, K. H.; Hirzinger, G.; More Accurate Camera and Hand-Eye Calibrations with Unknown Grid Pattern Dimensions, ICRA 2008 - IEEE International Conference on Robotics and Automation, 2008. 1398-1405.

[17] Tsai, R.: A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology using Off-The-Shelf TV Cameras and Lenses, IEEE Journal of Robotics and Automation, 3(4), 1987, 323-344.

[18] Witkin, A. P.: Scale-Space Filtering, Proceedings of the 8th International Joint Conference on Artificial Intelligence, 1983, 1019–1022.

[19] Zhang, Z.: A Flexible New Technique for Camera Calibration, IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(11), 2000, 1330-1334.