

REAL TIME 3D SURFACE FEATURE EXTRACTION ON FPGA

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ZAFER HAŞİM TELLİOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

JUNE 2010

Approval of the thesis:

REAL TIME 3D SURFACE FEATURE EXTRACTION ON FPGA

submitted by **ZAFER HAŞİM TELLİOĞLU** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. İsmet Erkmen
Head of Department, **Electrical and Electronics Engineering**

Assist. Prof. Dr. İlkey Ulusoy
Supervisor, **Electrical and Electronics Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering Dept., METU

Assist. Prof. Dr. İlkey Ulusoy
Electrical and Electronics Engineering Dept., METU

Assoc. Prof. Dr. Tolga Çiloğlu
Electrical and Electronics Engineering Dept., METU

Assist. Prof. Dr. Cüneyt Bazlamaçcı
Electrical and Electronics Engineering Dept., METU

M.Sc. Erdem Akagündüz
Image Processing Dep., ASELSAN Inc.

Date: 14.06.2010

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Zafer Haşim TELLİOĞLU

Signature:

ABSTRACT

REAL TIME 3D SURFACE FEATURE EXTRACTION ON FPGA

TELLİOĞLU, Zafer Haşim

M.Sc. Department of Electrical and Electronics Engineering

Supervisor: Assist. Prof. İlkay ULUSOY

June 2010, 108 pages

Three dimensional (3D) surface feature extractions based on mean (H) and Gaussian (K) curvature analysis of range maps, also known as depth maps, is an important tool for machine vision applications such as object detection, registration and recognition. Mean and Gaussian curvature calculation algorithms have already been implemented and examined as software. In this thesis, hardware based digital curvature processors are designed. Two types of real time surface feature extraction and classification hardware are developed which perform mean and Gaussian curvature analysis at different scale levels. The techniques use different gradient approximations. A fast square root algorithm using both LUT (look up table) and linear fitting technique is developed to calculate H and K values of the surface described by the 3D Range Map formed by fixed point numbers. The proposed methods are simulated in MatLab software and implemented on different FPGAs using VHDL hardware language. Calculation times, outputs and power analysis of these techniques are compared to CPU based 64 bit float data type calculations.

Keywords: FPGA, Scale Space, Gaussian curvature, Mean curvature, HK segmentation, hardware gradient approximation, feature, fast square root calculation

ÖZ

FPGA KULLANARAK 3B YÜZEYLERİN ÖZİNİTELİKLERİNİN GERÇEK ZAMANLI OLARAK ÇIKARTILMASI

TELLİOĞLU, Zafer Haşim

Yüksek Lisans, Elektrik Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Yrd. Doç. İlkey ULUSOY

Haziran 2010, 108 sayfa

Uzaklık haritaları ile ifade edilen üç boyutlu (3B) yüzeylerin, ortalama (H) ve Gaussian (K) eğrilik analizleri kullanılarak özneliklerinin çıkartılması, nesne algılamadan tanımaya kadar olan birçok bilgisayar görüntü işleme algoritması için önemli bir teknik olarak görülmektedir. Ortalama ve Gaussian eğrilik analizleri bilgisayarlar üzerinde yazılım tabanlı olarak geliştirilmiş ve uygulanmıştır. Bu çalışmada, FPGA üzerinde gerçek zamanlı çalışan sayısal eğrilik işlemcisi tasarlanmıştır. Hız bakımından yazılım tabanlı uygulamalara alternatif olarak, çoklu ölçekler üzerinde H ve K analizlerine göre yüzey sınıflandırması yapabilen, iki adet farklı sayısal eğrilik işlemcisi geliştirilmiştir. Bu iki işlemci farklı türev hesaplama yaklaşımları kullanmaktadır. İşlemciler içerisinde, analizler için gerekli olan gerçek zamanlı karekök hesaplama devresi tasarlanmıştır. Bu devre, giriş sayılarının olasılıklarını göz önünde tutarak, hatanın en az olması için gerekli bölgelerde RAM kullanan yada doğrusal tahmin yapan bir yapıdır. Analiz edilen yüzeyler sabit noktalı sayı temsilleri ile ifade edilmektedir. İşlemciler MatLab üzerinde tasarlanmış ve test edilmiş, VHDL donanım dili ile iki farklı FPGA üzerinde çalıştırılmıştır. Donanım, simülasyon ve yazılım sonuçları birbirleri ile kıyaslanmıştır. Ayrıca hız sonuçları ve güç tüketimleri de verilmiştir.

Anahtar Kelimeler: Ortalama eğrilik, Gaussian eğrilik, FPGA, hızlı karekök

To My Family

ACKNOWLEDGEMENTS

I would like to express my thanks to my supervisor Assist. Prof. İlkey Ulusoy for her guidance throughout the development of this thesis. I gratefully thank Erdem Akagündüz for his efforts and motivation during the thesis. His suggestions in several stages of the work provided important improvements in the thesis. I appreciate Burak Şekerlisoy for his assistance in hardware implementation. Also I would like to thank Sibel Şekerlisoy for her contribution in reviewing the thesis. Lastly, I would like to give my sincere thanks to my mother for her love and support all through my life.

TABLE OF CONTENTS

ABSTRACT	IV
ÖZ	V
ACKNOWLEDGEMENTS	VII
TABLE OF CONTENTS	VIII
LIST OF TABLES	XI
LIST OF FIGURES	XII
CHAPTERS	
1 INTRODUCTION	1
1.1 Motivation	1
1.2 Background and Literature	2
1.3 Thesis Objective and Outline	3
2 PRELIMINARIES	4
2.1 Range Map	4
2.1.1 Range Map Generation	4
2.1.2 Range Map Precision & Data Formats	5
2.1.3 Quantization Effect on Range Map.....	6
2.1.4 Valid and Invalid Points of Range Maps	7
2.1.5 3D Range Map.....	8
2.2 Scale Space	12
2.3 HK Segmentation	13
2.3.1 Curves, Tangent and Normal Vectors.....	13
2.3.2 Curvature	14
2.3.3 Surface Representation	15
2.3.4 Surface Tangent Plane and Normal Vector	16
2.3.5 Surface Curvature	18
2.3.6 Principle Curvatures	21
2.3.7 Mean and Gaussian Curvature Based Surface Classification	23
2.3.8 Derivation of K and H	24
2.4 Real Time Concept	28
2.4.1 Hardware & Software Partitioning	30
2.4.2 Real Time Criteria in HK Analysis.....	31
2.4.3 Real Time Hardware Platforms	31
2.5 FPGA Basics	31
2.5.1 Used FPGAs	33
2.6 System on Chip (SoC) Design	34

3	DIGITAL PROCESSING & HARDWARE IMPLEMENTATION	36
3.1	Mean and Gaussian Curvature Computation in Discrete Domain.....	36
3.1.1	Estimation Techniques.....	36
3.1.2	Noise on Surfaces and Definition of Classification Type Ranges	38
3.1.3	Filtering Range Data.....	40
3.1.4	Thresholds	41
3.2	Hardware Based Digital Curvature Processor Proposal	42
3.2.1	Real Time Requirements	42
3.2.2	Control of Parameters	42
3.2.3	Range Map Data Format and Quantization	43
3.2.4	Real Time Stream Signal Input and Parallelism	45
3.2.5	Pipeline Scale Space Analysis	48
3.2.6	Gradient Outputs.....	51
3.3	Non-Causality Problem in Range Map Processing	52
3.4	Digital Curvature Processor Type 1	54
3.4.1	Derivative Approximations in Hardware.....	55
3.4.2	Hardware Scale Space Technique.....	57
3.4.3	HK Calculation	59
3.5	Digital Curvature Processor Type 2	64
3.5.1	Gradient Approximations in Hardware.....	65
3.5.2	Hardware Scale Space Technique.....	66
3.5.3	HK Calculation	66
3.6	Surface Type Classification	67
3.7	Fast Square Root in Hardware	68
3.8	Thresholds.....	74
4	EMBEDDED COMPUTER & SOFTWARES.....	76
4.1	Computer Architecture.....	76
4.2	Software.....	78
4.2.1	Connected Component and Center of Mass Algorithm	78
5	RESULTS & COMPARISONS	82
5.1	Output Validation and Comparisons.....	82
5.1.1	Notation of Classification	82
5.1.2	Simulation and Hardware Outputs.....	82
5.1.3	Comparison of DCP Outputs	85
5.1.4	Fix Point Quantization Effect	87
5.1.5	Comparison of Software and Hardware Implementations	89
5.1.6	Rounding Operator Effect in DCP2.....	91
5.1.7	Rotation and Translation Invariance.....	92
5.1.8	Scale Space Level Outputs	92
5.2	Software Output	94
5.3	Time Measurements and Comparisons	96
5.3.1	Software Performance	96

5.3.2	Hardware Performance	97
5.3.3	Hardware and Software Speed Comparison	98
5.4	Embedded CPU Connected Component Algorithm Speed	99
5.5	Power Analysis.....	99
6	CONCLUSION AND FUTURE WORKS.....	101
	REFERENCES.....	103
	APPENDIX A. MEAN AND GAUSSIAN CURVATURE IN MATLAB.....	107

LIST OF TABLES

TABLES

Table 1 – Gaussian classification of surfaces.....	23
Table 2 – Available classifications in HK segmentation.	25
Table 3 – Sign representation in 2’s complement form	60
Table 4 – DCP Classification outputs	67
Table 5 – Effect of coefficient quantization.....	72
Table 6 – Selected node points for approximation in hardware.....	72
Table 7 – Colored representation of surface classification.	83
Table 8 – Time measurements of software implementations in MatLab.....	97
Table 9 – Possible maximum speeds of designed curvature processors.	98
Table 10 – Speed measurements of the processor realized on ml403 and ml505 boards.....	98

LIST OF FIGURES

FIGURES

Figure 1 - Intensity image and range map of a real scene [13] are shown in (a) and (b), respectively.....	5
Figure 2 - A peak is shown in the range map. The precision is enough in (a), but in (b) 1bit fraction length causes artificial edges and corners. The real values are shown in (c) and (d).	7
Figure 3 - Graphical visualization of a 3-D (2.5D) range map. The map has 4 2-D layers.	8
Figure 4 – Shifting in range data, in (b), causes damaging surface representation. In a good range data, in (a), mean error in each rows or columns should be zero.	10
Figure 5 – Bilinear transformation example of an erroneous range map.....	11
Figure 6 – A synthetic equally sampled range map.	11
Figure 7 – Scale space representations of a synthetic range map. 0 th scale space, the source data, is in (a). 1 st scale is in (b) and 2 nd is in (c).....	12
Figure 8 – A Curve and its tangent normal vectors.	14
Figure 9 – Tangent vector, Normal Vector and tangent plane of a surface [22]..	17
Figure 10 – Tangent vectors, tangent plane and normal vector of a parametric surface [22]	17
Figure 11 – Tangent vector of $\rho(t)$ and its angle with r_u [23].....	20
Figure 12 - Different curves on surface and principle curvature geodesics.....	21
Figure 13 – Changing of Curvature with θ [23].....	22
Figure 14 – FPGA design flow.	32
Figure 15 – Hardware software partitioning and System on Chip design flow in FPGAs.....	35
Figure 16 – Bending thresholds in 2-D.	38
Figure 17 – A closer look to a planar surface makes irregularities more visible.	39
Figure 18 – DCP and Range Map Generator connectivity	40
Figure 19 – Configuration of DCP by upper level user.	43

Figure 20 – 128x128 range map organization in a RAM.....	46
Figure 21 – Vector flow between Range map generator and DCP	46
Figure 22 – DCP application with range map generator (top) or a RAM (bottom)	47
Figure 23 – Timing diagram of interface signals	47
Figure 24 – Control signals for synchronization between DCP logics.	48
Figure 25 – Pipeline usage of DCP for scale space analysis.....	49
Figure 26 – Single DCP usage for scale space analysis.....	50
Figure 27 – Gradient output and address decoder supported DCP and IO signals	51
Figure 28 – Output signal flows of two different DCP, gradients available in top design.	52
Figure 29 – Scale space kernel.....	53
Figure 30 – Logic architecture of DCP type 1	54
Figure 31 – Input vectors used for derivatives.....	55
Figure 32 – Dv derivative and scale space block logic diagram.....	57
Figure 33 – Scale Space generation logic	58
Figure 34 – H and K sign extraction logic architecture.	62
Figure 35 – One instance usage example of HK Logic.	63
Figure 36 - Logic architecture of DCP type 2.....	64
Figure 37 – Required samples to calculate derivatives in DCP2.....	65
Figure 38 – Purposed square root logic.....	69
Figure 39 – Square root line and its discontinuous approximation.....	70
Figure 40 – Input distribution of square root function, Wl=22, Fl=6	71
Figure 41 – Overall approximation and error in the first interval.....	73
Figure 42 – H and K values and threshold effect.....	74
Figure 43 – Top level FPGA design	77
Figure 44 – Connected component label assignment.....	79
Figure 45 – Connected component algorithm output.....	79
Figure 46 – Artificial surfaces.....	84
Figure 47 – Simulation and real hardware outputs of the artificial data.....	84
Figure 48 – Range data of a scene and its 3D plot [13]	85

Figure 49 – Hardware outputs of the range data in Figure 48	86
Figure 50 – Software outputs of float64 and quantized fix-point range data.....	87
Figure 51 – Quantization effect.....	88
Figure 52 – Software and hardware outputs	89
Figure 53 – Outputs without rounding operation in DCP2.....	91
Figure 54 - Intensity image of the rotated object [12]	92
Figure 55 – Outputs of the rotated object	92
Figure 56 – Scale space levels and their analysis outputs for the artificial data..	93
Figure 57 – Scale space levels analysis outputs for the real data.....	94
Figure 58 – Connected component algorithm outputs.	95

CHAPTER 1

INTRODUCTION

1.1 Motivation

Processing image texture and range maps to extract features has been very popular approach for the past years. Usually, the extracted features are used by different applications, such as object recognition. Feature extraction enables faster processing for computers to evaluate desired algorithms. Without feature extraction, to have a valid output, for example face recognition, it is required to make the same calculations in the whole image again and again. Therefore extracting the key points, which are meaningful properties for the algorithm, is a more preferred technique. In range map processing, one of the commonly used feature type is surface classification. Different approaches are proposed for classification of surfaces until now. Edge and corner detection algorithms have been very popular in recent years. Texture based surface analysis is performed in [1]. In [2], Photometric stereo with frequency domain analysis has been proposed to extract surface texture information especially considering surface-rotation invariance. These algorithms are texture based and also applicable to range data. Behind these algorithms, curvature based approaches are started to be used widely. Curvature analysis does not only provide edge and corner information, but also classifies any discrete point (or pixel) in the real range data into a variety of categories. This categorization makes curvature analysis more powerful in digital image and video processing. However, complexity and computational load is very high in curvature based surface classification algorithms.

Although feature extraction provides less computational cost for higher level algorithms, processing the range data or an image to extract features requires

many computations. If both feature extraction and feature processing are implemented in software, the speed of whole algorithm can be worse than direct analyzing. Therefore software and hardware partitioning is used for system design. High computational task can be integrated into hardware while complex algorithms are implemented in software. Curvature extraction from range map using hardware will provide higher performance for surface classification based higher level algorithms.

1.2 Background and Literature

There are two different types of curvature analysis methods widely used in literature. First one is Mean, abbreviated as H , and Gaussian, abbreviated as K , curvature classification. This method is known as HK segmentation [3] and proposed by Besl in 1986. In this method, principle curvatures are used to calculate H and K values in each point in image or range data. According to signs of these values one of 8 possible classifications is assigned to the point. The second method is shape index, S , and curvature magnitude, curvedness, C , based classification method, which is proposed by Koenderink [4]. Again, principle curvatures determine S and C values. Both techniques are examined and compared previously [5, 6]. In [5], Cantzler and Fisher conclude that SC segmentation method is slightly better to noise than HK segmentation is. However, when multiple scale spaces are used, HK segmentation is more successful [6].

Another important analysis tool in digital image and video processing world is scale space construction and running the desired algorithm in each scale level. This is a very useful approach to convert scale dependent algorithms to scale invariant. Many algorithms, like SIFT [7], use scale space technique. Scale space is also used in curvature analysis previously in [8, 6]. Classification types applied to all points in the surface may make it hard to extract actual surface type. In most cases, analyzing next scale space makes large areas, i.e. relevant features, more visible. This study has been performed in [6].

HK based surface classification is researched and implemented in software by different authors for different purposes. Processing of laser points, range map generated by laser range device, is performed in [9]. Object recognition purposed classification is researched in [10,11]. An interesting study is performed in medical science in [12], aiming finding of airway lesions. None of these works are time critical applications.

1.3 Thesis Objective and Outline

Throughout this thesis, the main purpose is to develop a new hardware based real time digital curvature processor which is capable of classifying surface curvature by using HK segmentation technique in multiple scale space levels. Comparison of hardware based classification with the software ones is also aimed. The input of digital curvature processor, DCP, is real time serial 3D range map vectors. The algorithms use fixed point data format. We aim to use parallel processing and hardware acceleration. Two different design techniques are developed for DCPs. Both techniques are simulated in MatLab and implemented in two different XILINX FPGAs. The outputs of the two design approaches are compared with each other and both outputs are compared with the outputs of software based available techniques. Speed performance of the software and the developed hardware implementations are also compared.

The outline of the thesis is as follows; Chapter 2 includes a brief introduction to range maps, mathematical description of curves, surfaces, curvature, H and K . Also FPGA based design strategies and SoC design approach are explained in this chapter. In Chapter 3, digital curvature processing in hardware is developed. Derivative approximations, HK classifications and fast square root logic, considering input distribution and nonlinearity of square root line, are developed in this chapter. Chapter 4 includes embedded computer design and software. Detailed results and comparisons are given in Chapter 5. Chapter 6 is the conclusion part and presents possible future works.

CHAPTER 2

PRELIMINARIES

2.1 Range Map

2-D images can not express real surfaces very well, because they are intensity based and this limits expression of surface geometry. To overcome the problem, range maps are developed. Range maps include positions of all points with respect to a reference point. It can be thought as a real position image. A variety of range maps is available. A range map may include x, y and z values of points in the scene, or it might just contain z values. If only z values are available, range map is considered as a range image or z-buffering.

Range maps are also known as depth maps, depth images, range buffers and depth buffers.

2.1.1 Range Map Generation

Range map generation requires special techniques and devices. Different techniques are available in literature. Laser scanners are one of the mostly used types of devices. Stereo camera based approaches are also very common.

Human vision takes depth information by exploiting stereo vision approach, we have two eyes and this provides depth and approximate range information of the scene. Based on this idea, range maps are calculated by using stereo camera.

Laser scanner scans the scene using a laser beam and measures the distance. Laser distance measurement is very sensitive and may produce noisy maps. Therefore a post processing, smoothing filter, is required.

Range image construction has been researched in many universities. University of South Florida Computer Vision and Pattern Recognition Group

generated real range images using laser scanners and structured light method [13]. Ohio State University Signal Analysis and Machine Perception Laboratory group have many laser based range images. They also keep some other universities' range image data base [14]. Stanford University Computer Science department has real and synthetic range images [15]. Synthetic data is useful for testing the algorithms. Also, Stuttgart University synthetic range image database is very useful [16].

Besides software, range map construction has been implemented in hardware. Harlan Hile and Colin Zheng have generated range map by using stereo video processing in FPGAs [17]. Their data 512x480 pixels and representation is 5bit. Another work is performed by Ahmad Darabiha, Jonathan Rose and James MacLean [18]. This work produces 256x360 maps with 8bit resolution in FPGAs. The speed is 30 frames sec.

2.1.2 Range Map Precision & Data Formats

A range map is shown in Figure 1 below. The representation format of data is important especially for higher level algorithms. Data format defines precision of distance measurements, x, y and z. A high precision defines surface more accurately. On the other hand, high precision range map data requires more space since more bits used.

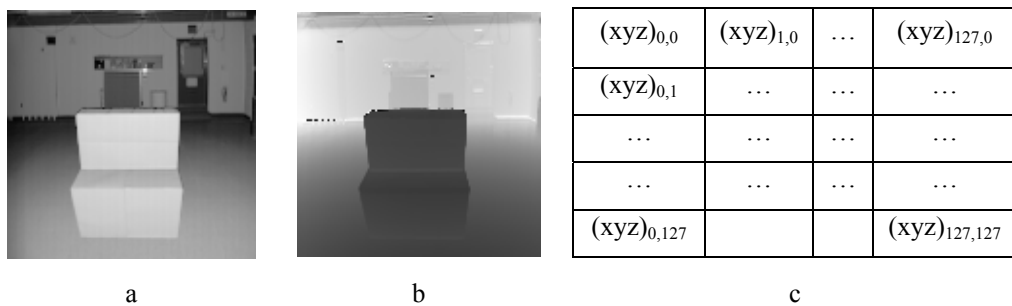


Figure 1 - Intensity image and range map of a real scene [13] are shown in (a) and (b), respectively. In (c) the map values are represented in matrix format. In each cell, x, y and z values are stored with desired representation format. The image and map are 128x128 pixel size.

In software based applications, 32bit floating point representation is usually used. Today's computer architectures are mostly 32bit and hard floating

point unit is available in many processors. Using 32bit range map data is preferred in both representation precision and computer architecture aspects. Others formats may also be used.

In hardware processing, floating point approach is not so preferred. Fixed point numbers are more useful, because multiplication, summation and subtraction operations are easier. However, more bits can be required to provide enough precision in fixed point data representation. Therefore, while range map is being generated, resolution of measurements should already be determined. According to the resolution, fixed point word and fraction lengths should also be selected accordingly.

2.1.3 Quantization Effect on Range Map

If fixed point fraction length is not enough to represent a scene, an excessive quantization error occurs. The quantization error in the range map may cause artificial edges, peaks or corners on the surface. An example is shown in Figure 2 below. In the figure, the range map includes a peak. Using enough precision, the peak is shown in (a). However, using the 1bit fraction length precision, the peak has some distortions as shown in (b). The real values are shown in (c) and (d). Obviously, 1bit precision is not enough for this range map.

Quantization of a surface data directly affects the quality of curvature analysis. As shown in Figure 2 (b), the corners of the peak are distorted. Curvature analysis of the corners on this data can not give actual description of the surface, the peak in the map. Most of the time, higher level algorithms put a smoothing filter to deal with surface roughness. If the roughness is due to quantization, the filtering can not solve the problem unless precision is increased at least in filtering stage. Therefore a valid range map must have enough precision to represent surface and scene in desired resolution.

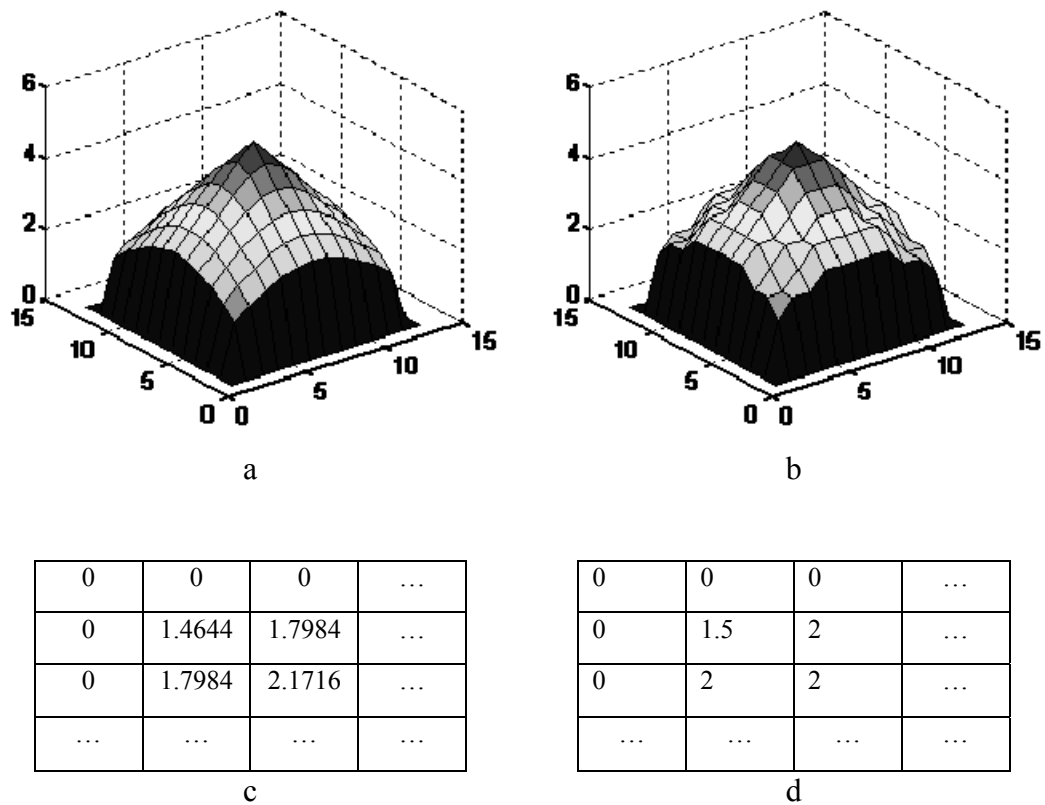


Figure 2 - A peak is shown in the range map. The precision is enough in (a), but in (b) 1bit fraction length causes artificial edges and corners. The real values are shown in (c) and (d).

2.1.4 Valid and Invalid Points of Range Maps

Range generating algorithms may not distinguish all points in the scene. In this case, the range map algorithm must sign these points. Usually another data in the range map is provided with x, y and z. The data is called by valid flags. This flag map explains which points in the range map are valid and invalid. Unfortunately, this technique increases the size of the map, even though 1bit is used for the flag. For a 128x128 map, extra 2K bytes are required. The problem is important especially for small embedded computers since they may not have enough memory. Another disadvantage of invalid points in curvature analysis may appear in surface classification. An invalid point causes misclassification of its neighbor points, even they are valid.

2.1.5 3D Range Map

As already mentioned, a range map does not only include depth, z , information. x and y values are also important to represent surface accurately. In differential geometry, a surface is represented by:

$$z = f(x, y)$$

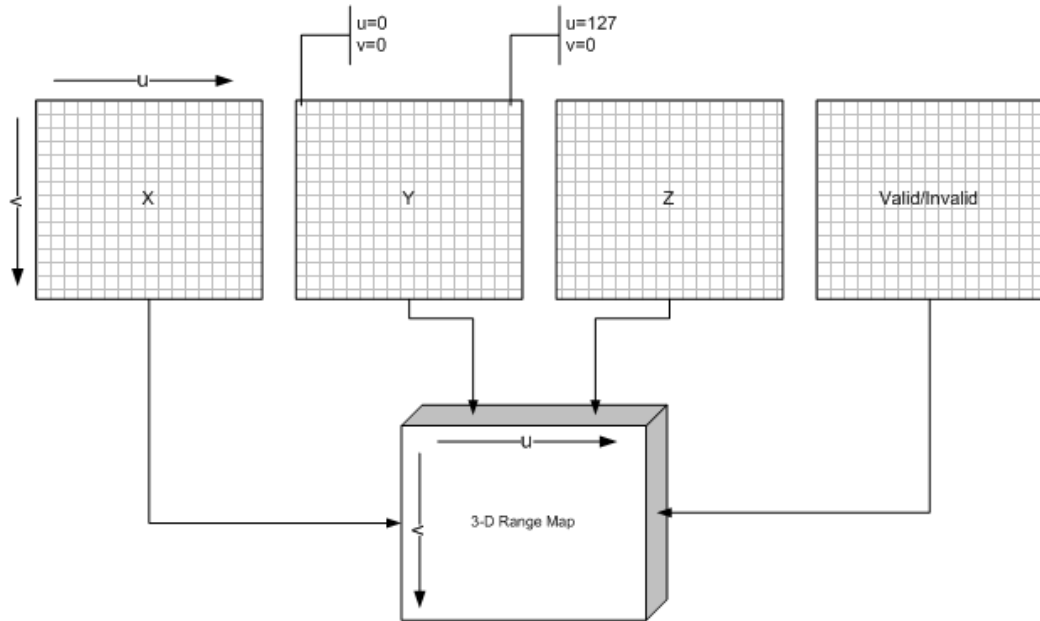


Figure 3 - Graphical visualization of a 3-D (2.5D) range map. The map has 4 2-D layers.

Therefore z is a function of x and y . Indeed, what a perfect range map is the function f . Another parameter of range map is its pixels. Conventionally u and v are used for range map pixel notation. In 2-D representation, i.e. matrix form, u refers to columns and v refers to rows of the range map. A complete range map contains 4 2-D maps. The fourth is valid/invalid map. In some resources, these types of range maps are called by 2.5D maps. Indeed, the map does not have all point information in 3D, it has only x , y and z values of a point of view. In this work, we prefer to use “3D map” term. A graphical representation is shown in Figure 3. Using new u and v variables, we may define the surface equation as;

$$\begin{aligned}
x &= X(u, v) \\
y &= Y(u, v) \\
z &= Z(u, v) = f(x, y) = f(X_{(u,v)}, Y_{(u,v)})
\end{aligned}$$

By these definitions, we can extract curvature structure of the range map. For a suitable range map, x and y values are monotone increasing with u and v values, respectively. A perfect range map has a relation between x and u, y and v values.

$$\begin{aligned}
X(u, v) &= u * \Delta_X \\
Y(u, v) &= v * \Delta_Y
\end{aligned}$$

Δ_X and Δ_Y values are desired horizontal and vertical resolutions of the range map. The precision of data type used in range map must cover these values. Most of the time, range maps, unfortunately, can not hold these relations exactly. Especially in hardware, providing a perfect range map is very difficult. Therefore we express the relations with using an error term;

$$\begin{aligned}
X(u, v) &= (u * \Delta_X) \pm \varepsilon_X \\
Y(u, v) &= (v * \Delta_Y) \pm \varepsilon_Y
\end{aligned}$$

In this case, the fix point resolution must be enough to express ε_X and ε_Y . These error terms depend on range map generation hardware, and they should be in a defined range. We may define the error ranges in worst case,

$$\begin{aligned}
0 &< \varepsilon_X < \Delta_X \\
0 &< \varepsilon_Y < \Delta_Y
\end{aligned}$$

Of course the range can be narrower, or zero, and range map generation algorithms should keep the errors in the ranges. If the errors excess defined ranges in a pixel, it is a definitely invalid point for upper level algorithm, curvature analysis in this case.

Also we may define instantaneous sampling errors as,

$$\begin{aligned}
\Delta_{X(u,v)} &= X_{(u,v)} - X_{(u-1,v)} \\
\Delta_{Y(u,v)} &= Y_{(u,v)} - Y_{(u,v-1)} \\
e_{x(u,v)} &= \Delta_X - [X_{(u,v)} - X_{(u-1,v)}] \leq \varepsilon_{X(u,v)} \\
e_{y(u,v)} &= \Delta_Y - [Y_{(u,v)} - Y_{(u,v-1)}] \leq \varepsilon_{Y(u,v)}
\end{aligned}$$

According to these definitions we may calculate mean sampling error in a single row;

$$\overline{e_{X(v)}} = \frac{1}{N} \sum_{u=1:N} [\Delta_x - (X_{(u,v)} - X_{(u-1,v)})] \cong 0 \quad v = [0..N-1] \quad (\text{Eq. 2.1})$$

N : width of the map

Here we define that the mean error for a single row must be zero. If this equation is not satisfied for a row, then we conclude that the row is shifted and the map is not valid. However, inverse of the assumption does not imply that the map is valid. Also, if the mean errors are equal for all rows, the assumption fails again because in that case the map has a phase shift and it does not affect the analysis. The assumption is also applicable to columns of range map.

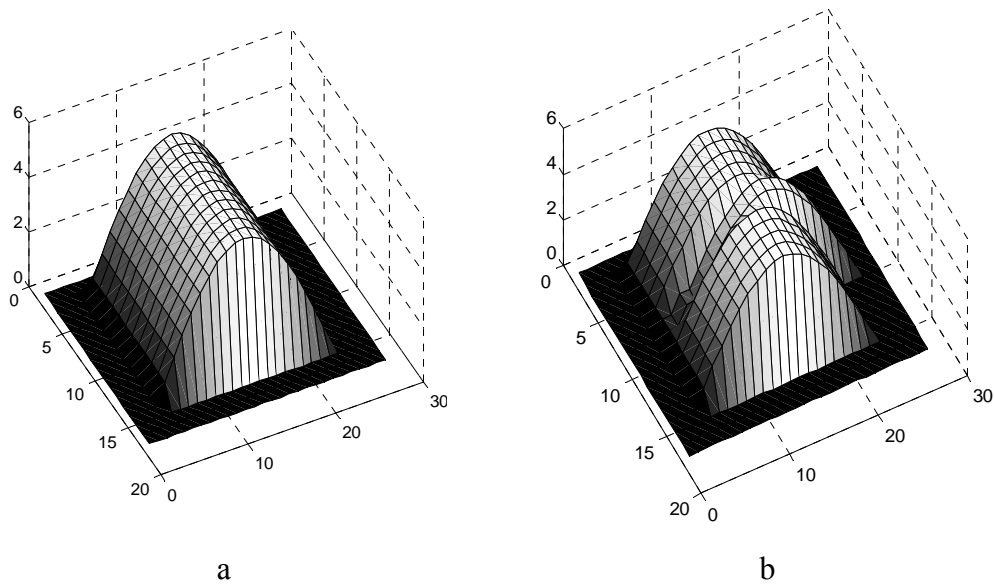


Figure 4 – Shifting in range data, in (b), causes damaging surface representation. In a good range data, in (a), mean error in each rows or columns should be zero.

A synthetic example is given in Figure 4. A cylindrical surface is shown in (a). One of the rows is shifted in x-direction and mean error for that row is greater than zero in (b). If all rows were shifted in the same amount, the cylindrical surface would be still valid.

An erroneous range map may be transformed into a zero erroneous map, i.e. $\varepsilon_x = \varepsilon_y = 0$. Basically, this is an interpolation process, bilinear or bicubic. In Figure 5, a noisy map and its zero error transform are shown. At top, some measurement errors in X and Y effect the Z values. The transform hinders measurement error in Z. Desired sampling distances are $\Delta_x = \Delta_y = 1$.

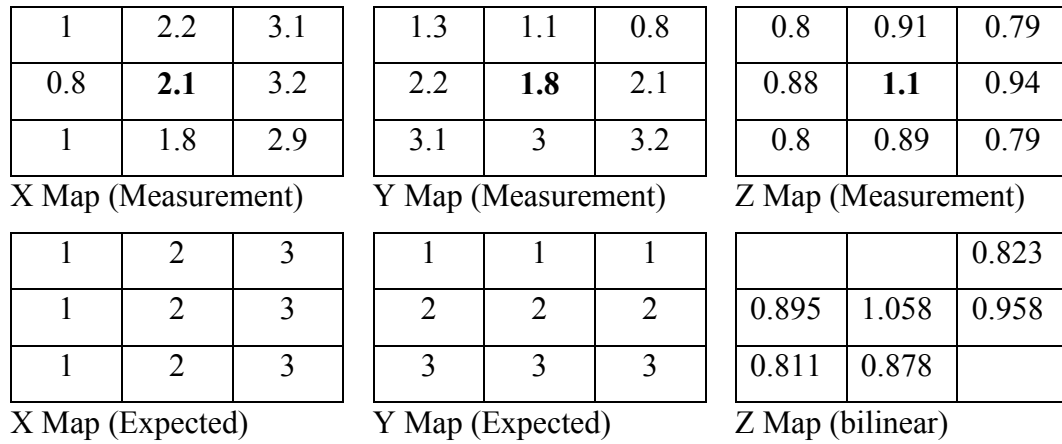


Figure 5 – Bilinear transformation example of an erroneous range map

In Figure 6, a synthetic perfect range map is shown.

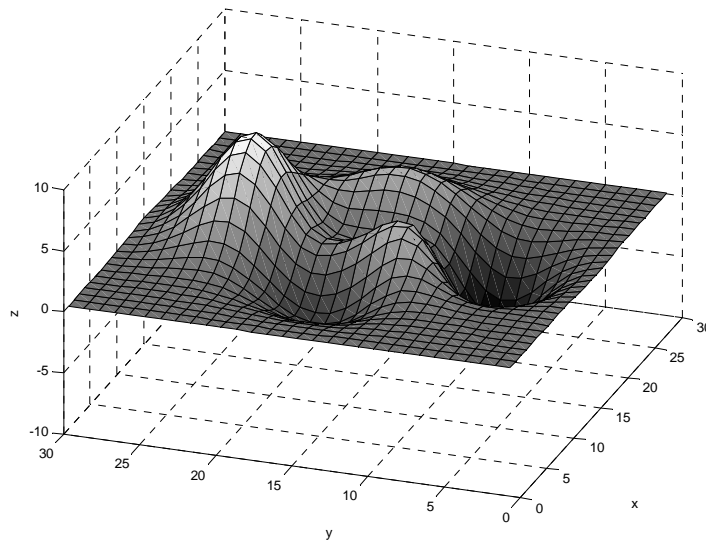


Figure 6 – A synthetic equally sampled range map.

2.2 Scale Space

Scale space provides scale invariant analysis for image processing. The concept has been analyzed by Tony Lindeberg [19]. Scale space is widely used in feature extraction, as well. In the same scene, there can be different size of features or a feature may contain many small features. Scale space technique is very helpful when the small features hide large features.

In curvature analysis, surface classification suffers from too detailed outputs. In a real range map data, surfaces are usually very rough, therefore outputs contain many different features, i.e. different classifications, on the same surface. Also, we may want to see only large surfaces and we may discard the small areas. Scale spaces of range maps provide these benefits in curvature analysis.

Scale space technique in curvature analysis is already used by many authors [6, 8].

In this work, we generate scale spaces by using Gaussian pyramids method, width and height of the map reduced by half in each scale using a 2-D Gaussian kernel. A scale space example of synthetic data is given in Figure 7. As going to next scale, details of surfaces disappear and only peaks stay visible.

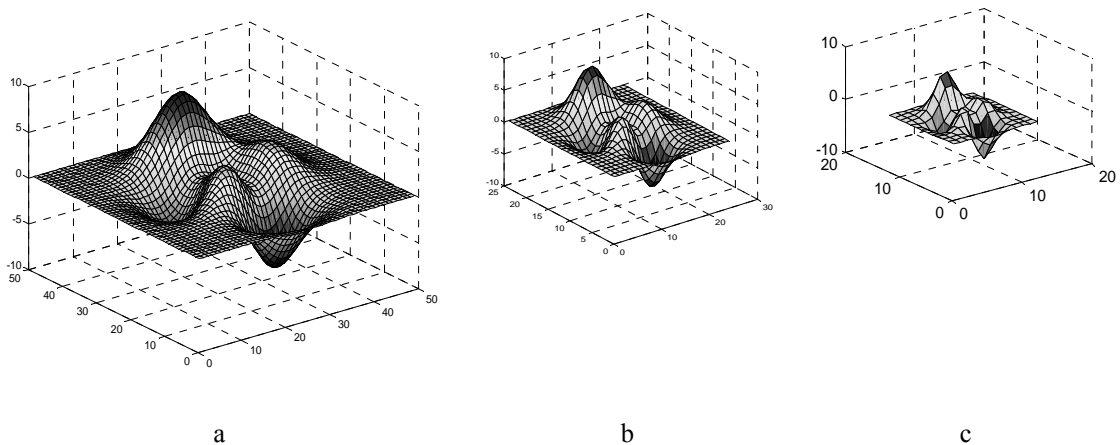


Figure 7 – Scale space representations of a synthetic range map. 0th scale space, the source data, is in (a). 1st scale is in (b) and 2nd is in (c).

2.3 HK Segmentation

Mean and Gaussian curvature based range map segmentation is basically a differential geometry operation. In this chapter, a mathematical description of HK analysis will be provided. The content can be found in an elementary differential geometry book.

2.3.1 Curves, Tangent and Normal Vectors

A differentiable curve in 3-D, α , is expressed by

$$\begin{aligned} I &= [a, b] \quad a, b \in R \\ \alpha &: I \rightarrow R^3 \\ \alpha(t) &= (x(t), y(t), z(t)) \quad t \in I \end{aligned}$$

The function above is called by plane curve in 3-D. If third component, z , is always zero, the curve is in 2-D and its representation;

$$\begin{aligned} I &= [a, b] \quad a, b \in R \\ \alpha &: I \rightarrow R^2 \\ \alpha(t) &= (x(t), y(t)) \quad t \in I \end{aligned}$$

Differentiable condition implies x , y and z functions to be differentiable. Therefore derivative of vector function α ;

$$\alpha'(t) = (x'(t), y'(t))$$

The tangent vector is also given by the formula above. Tangent vector contains direction of the curve at t instant. The unit tangent vector or direction vector, T , is calculated by;

$$T(t) = \frac{\alpha'(t)}{|\alpha'(t)|}$$

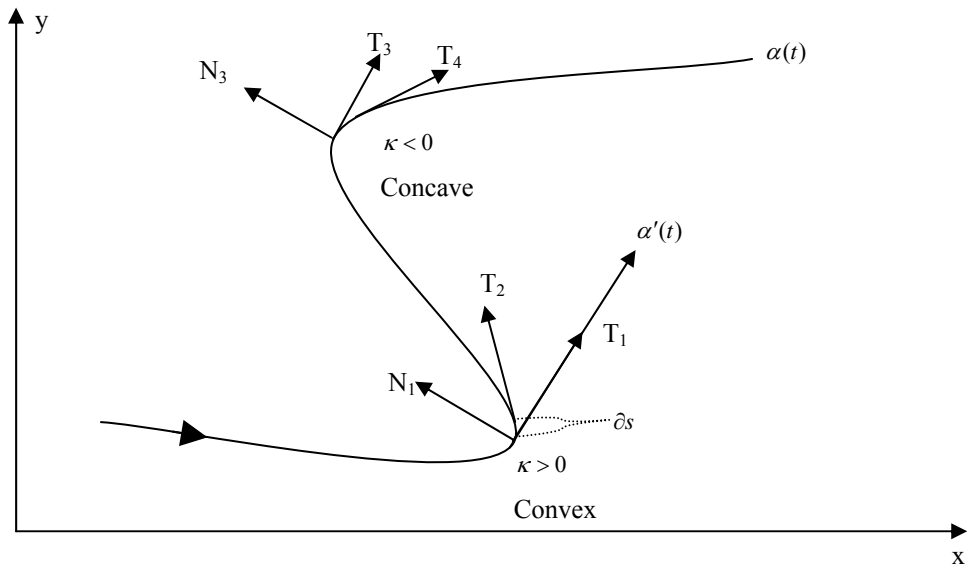


Figure 8 – A Curve and its tangent normal vectors.

Derivative of unit tangent vector is orthogonal to itself and it gives direction of normal vector. Unit normal vector, N , is expressed by;

$$\begin{aligned}
 T(t) \cdot T(t) &= 1 \\
 \frac{\partial(T(t) \cdot T(t))}{\partial t} &= 0 \\
 T'(t) \cdot T(t) + T(t) \cdot T'(t) &= 0 \\
 2 \cdot T(t) \cdot T'(t) &= 0 \Rightarrow T(t) \perp T'(t) \\
 N(t) &= \frac{T'(t)}{|T'(t)|}
 \end{aligned}
 \tag{Eq. 2.2}$$

The unit tangent and unit normal vectors are shown in Figure 8 for different t instants. The orthonormal vectors N and T construct a basis set for 2-D space.

2.3.2 Curvature

Curvature, κ , is a measurement of how much the curve is bended. Intuitively, it is a quantity of direction change while moving along the curve. For a straight line, direction, unit tangent vector, does not change while moving on the

line. Therefore, a normal vector has zero length and the curvature is also zero. Referring to Figure 8, the infinite small length of the curve, arch length, is represented by ∂s . We already define direction change by taking derivative of tangent vector. According to the definition, curvature;

$$\kappa = \left| \frac{\partial T}{\partial s} \right| = \left| \frac{\partial T}{\partial t} \cdot \frac{\partial t}{\partial s} \right|$$

The unit arch length is derivative of the vector function;

$$|\partial s| = |\alpha'(t)| \cdot |\partial t|$$

The curvature;

$$\frac{\partial T}{\partial t} \cdot \frac{\partial t}{\partial s} = \frac{\frac{\partial T}{\partial t}}{\frac{\partial s}{\partial t}} = \frac{1}{|\alpha'(t)|} \frac{\partial T}{\partial t} = \kappa N$$

Any regular parameterized curve can be re-parameterized by arch length [20]. Therefore arch length, s , is a parameterization parameter. The curvature for arch-length parameterized curves can be expressed by;

$$\kappa(s) = \frac{T'(s)}{N(s)}$$

Therefore, if T is the twisting counterclockwise, curvature is greater than zero, $\kappa > 0$ and the curve is convex. If T is twisting clockwise, curvature is negative and curve is concave [21]. In Figure 8, near $t=1$ point, the curve is convex and around $t=3$, it is concave.

2.3.3 Surface Representation

We may write a surface, S , in terms of its three components x , y and z ;

$$S \subset R^3$$

$$S = \{x, y, z\} \quad z = Z(x, y)$$

The range map, R , is a parametric representation of the components x , y , z .

$$x = X(u, v)$$

$$y = Y(u, v)$$

Considering our range map definitions;

$$X(u, v) = u * \Delta_X$$

$$Y(u, v) = v * \Delta_Y$$

There is a unique linear mapping between X and u values. Therefore we may use the following notation to represent surface S ,

$$U \subset I^2 \quad I : \text{integer numbers}$$

$$r : U \rightarrow S$$

$$r(U) = S$$

$$r(u, v) = (X_{(u,v)}, Y_{(u,v)}, Z(X_{(u,v)}, Y_{(u,v)})) \quad u, v \in U$$

$$r(u, v) = (u * \Delta_X, v * \Delta_Y, Z_{(u,v)}) \quad u, v \in U$$

In differential geometry, a defined one-to-one mapping between x and u components represented by;

$$z = f(x, y) = Z(u, v) = Z(x, y) \quad (\text{Eq. 2.3})$$

$$r(u, v) = (u, v, Z_{(u,v)})$$

This convention is known as graph surface representation and the surface is called Monge patch. The range map provides representation of surfaces with two variables, u and v , according to the assumptions. There exists a similarity between our range map and graph surface. In 3D range map, x and y values are depends on u and v values.

2.3.4 Surface Tangent Plane and Normal Vector

As in the curves, the surfaces also have normal vector for a given point. Directional derivatives, gradient, gives the normal vector of the surface at any point. For the surface U , unit normal vector [22], N , is;

$$\nabla U = \left(\frac{\partial U(x, y, x)}{\partial x}, \frac{\partial U(x, y, x)}{\partial y}, \frac{\partial U(x, y, x)}{\partial z} \right)$$

$$N = \frac{\nabla U}{|\nabla U|} \quad (\text{Eq. 2.4})$$

The gradient, i.e. unit normal vector, is orthogonal to tangent vector of all level curves of the surface at any point on the surface. On a continuous surface in 3-D, there are infinitely many curves passing through a point on the surface.

These curves are known as *geodesic* in differential geometry. In Figure 9 below, one of the curves, $r(t) = (x(t), y(t), z(t))$, gradient vector, ∇U , and the tangent vector of the curve, v , is shown.

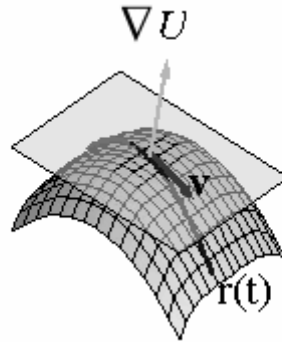


Figure 9 – Tangent vector, Normal Vector and tangent plane of a surface [22]

The tangent vector of any curves on the surface is orthogonal to normal vector of the surface;

$$\nabla U \bullet v = 0$$

A plane can be expressed by its normal and a point on it. The normal vector is known and taking a point on the surface, we may express tangent plane of the surface at that point. Tangent plane of the surface U is also shown in Figure 9.

For parametric surfaces, another possible normal vector calculation is cross product of tangent vectors along principle directions of parameterizations. A parameterized surface $r(u, v)$ has tangent vectors along u and v directions.

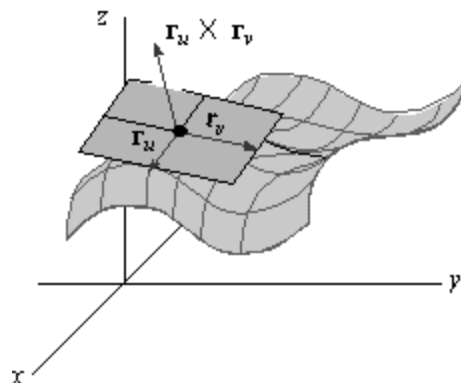


Figure 10 – Tangent vectors, tangent plane and normal vector of a parametric surface [22]

A parameterized surface \mathbf{r} is given by

$$\mathbf{r}(u, v) = (x(u, v), y(u, v), z(u, v)) \quad (\text{Eq. 2.5})$$

The tangent vectors along u and v directions;

$$\mathbf{r}_u = \left(\frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u} \right)$$

$$\mathbf{r}_v = \left(\frac{\partial x}{\partial v}, \frac{\partial y}{\partial v}, \frac{\partial z}{\partial v} \right)$$

The *unit surface normal vector* can be expressed by;

$$\mathbf{N} = \frac{\mathbf{r}_u \times \mathbf{r}_v}{|\mathbf{r}_u \times \mathbf{r}_v|} \quad (\text{Eq. 2.6})$$

\mathbf{r}_u and \mathbf{r}_v vectors construct a basis set for tangent plane of the surface. The normal vector, tangent plane and basis vectors are shown in Figure 10. The tangent vectors are orthogonal to both each other and normal vector.

2.3.5 Surface Curvature

Curvature is already defined in previous sections as direction change while moving on a curve within unit step. To achieve curvature of surface at a point, we should consider lines on the surface, i.e. geodesics. There are many different approaches in literature to define curvature of a surface. Shape operator, i.e. Weingarten Map, is commonly used in many resources. Here, mathematical approach is preferred and all content is based on works of Kevin Shirley and Jeff Knisley in [23]. More details can be found in this source.

The curvature of a curve is calculated using unit arch length function of the curve. Remember $\frac{\partial s}{\partial t}$ is a parameter for curvature. Assume $\mathbf{r}(u, v)$ is a surface and $\rho(t)$ is a geodesic on it. The arch length [23]

$$\left| \frac{\partial s}{\partial t} \right| = (\langle \rho', \rho' \rangle)^{1/2}$$

$$\rho' = r_u \cdot \frac{\partial u}{\partial t} + r_v \cdot \frac{\partial v}{\partial t}$$

$$\rho' \bullet \rho' = r_u \bullet r_u \left(\frac{\partial u}{\partial t} \right)^2 + 2 \cdot r_u \bullet r_v \cdot \frac{\partial u}{\partial t} \cdot \frac{\partial v}{\partial t} + r_v \bullet r_v \cdot \left(\frac{\partial v}{\partial t} \right)^2$$

Defining coefficients;

$$g_{11} = r_u \bullet r_u \quad g_{12} = r_u \bullet r_v \quad g_{22} = r_v \bullet r_v$$

Arch length;

$$\left(\frac{\partial s}{\partial t} \right)^2 = g_{11} \left(\frac{\partial u}{\partial t} \right)^2 + 2g_{12} \left(\frac{\partial u}{\partial t} \right) \left(\frac{\partial v}{\partial t} \right) + g_{22} \left(\frac{\partial v}{\partial t} \right)^2$$

Remember that $\rho'(t)$ is tangent vector of the curve and its normalized version gives direction vector \mathbf{T} . Considering $\rho'' = \kappa \mathbf{N}$; *normal curvature* κ_n in the direction of \mathbf{n} given by

$$\kappa_n = \rho'' \bullet \mathbf{n} \quad (\text{Eq. 2.7})$$

$$\mathbf{n} = \mathbf{r}_u \times \mathbf{r}_v$$

ρ'' ;

$$\rho'' = \frac{\partial r_u}{\partial t} \frac{\partial u}{\partial t} + r_u \frac{\partial^2 u}{\partial t^2} + \frac{\partial r_v}{\partial t} \frac{\partial v}{\partial t} + r_v \frac{\partial^2 v}{\partial t^2}$$

$$\rho'' = \left(\frac{\partial r_u}{\partial u} \frac{\partial u}{\partial t} + \frac{\partial r_u}{\partial v} \frac{\partial v}{\partial t} \right) \frac{\partial u}{\partial t} + r_u \frac{\partial^2 u}{\partial t^2} + \left(\frac{\partial r_v}{\partial u} \frac{\partial u}{\partial t} + \frac{\partial r_v}{\partial v} \frac{\partial v}{\partial t} \right) \frac{\partial v}{\partial t} + r_v \frac{\partial^2 v}{\partial t^2}$$

$$\rho'' = r_{uu} \left(\frac{\partial u}{\partial t} \right)^2 + r_{uv} \frac{\partial v}{\partial t} \frac{\partial u}{\partial t} + r_u \frac{\partial^2 u}{\partial t^2} + r_{vu} \frac{\partial u}{\partial t} \frac{\partial v}{\partial t} + r_{vv} \left(\frac{\partial v}{\partial t} \right)^2 + r_v \frac{\partial^2 v}{\partial t^2}$$

$$\rho'' = r_{uu} \left(\frac{\partial u}{\partial t} \right)^2 + 2r_{uv} \frac{\partial v}{\partial t} \frac{\partial u}{\partial t} + r_u \frac{\partial^2 u}{\partial t^2} + r_{vv} \left(\frac{\partial v}{\partial t} \right)^2 + r_v \frac{\partial^2 v}{\partial t^2}$$

Curvature along normal direction;

$$\kappa_n = \rho'' \bullet \mathbf{n} = \frac{\partial r_u}{\partial u} \bullet \mathbf{n} \frac{\partial u}{\partial t} \frac{\partial u}{\partial t} + r_u \bullet \mathbf{n} \frac{\partial^2 u}{\partial t^2} + \frac{\partial r_v}{\partial v} \bullet \mathbf{n} \frac{\partial v}{\partial t} \frac{\partial v}{\partial t} + r_v \bullet \mathbf{n} \frac{\partial^2 v}{\partial t^2}$$

$$\kappa_n = r_{uu} \bullet \mathbf{n} \left(\frac{\partial u}{\partial t} \right)^2 + 2r_{uv} \bullet \mathbf{n} \frac{\partial v}{\partial t} \frac{\partial u}{\partial t} + r_u \bullet \mathbf{n} \frac{\partial^2 u}{\partial t^2} + r_{vv} \bullet \mathbf{n} \left(\frac{\partial v}{\partial t} \right)^2 + r_v \bullet \mathbf{n} \frac{\partial^2 v}{\partial t^2}$$

\mathbf{r}_u and \mathbf{r}_v are orthogonal to \mathbf{n} ;

$$\begin{aligned}\kappa_n &= \rho'' \cdot \mathbf{n} = r_{uu} \cdot n \left(\frac{\partial u}{\partial t} \right)^2 + r_{vv} \cdot n \left(\frac{\partial v}{\partial t} \right)^2 + r_v \cdot n \frac{\partial^2 v}{\partial t^2} \\ \kappa_n &= r_{uu} \cdot n \left(\frac{\partial u}{\partial t} \right)^2 + 2r_{uv} \cdot n \frac{\partial v}{\partial t} \frac{\partial u}{\partial t} + r_{vv} \cdot n \left(\frac{\partial v}{\partial t} \right)^2\end{aligned}$$

Referring to Figure 11; $du/dt = \cos(\theta) |\mathbf{r}_u|^{-1}$ and $dv/dt = \sin(\theta) |\mathbf{r}_v|^{-1}$. Here, θ is the angle between \mathbf{r}_u and tangent vector, ρ' .

$$\rho' = \cos(\theta) \frac{\mathbf{r}_u}{|\mathbf{r}_u|} + \sin(\theta) \frac{\mathbf{r}_v}{|\mathbf{r}_v|}$$

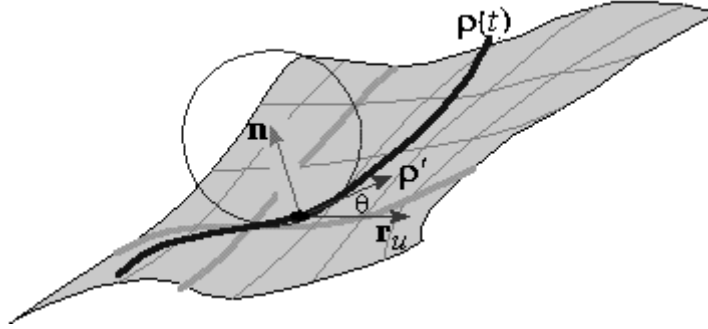


Figure 11 – Tangent vector of $\rho(t)$ and its angle with \mathbf{r}_u [23]

After substituting, the curvature along surface normal;

$$\kappa_n(\theta) = \frac{r_{uu} \cdot n}{|\mathbf{r}_u|^2} \cos^2(\theta) + \frac{r_{uv} \cdot n}{|\mathbf{r}_u| |\mathbf{r}_v|} \sin(2\theta) + \frac{r_{vv} \cdot n}{|\mathbf{r}_v|^2} \sin^2(\theta) \quad (\text{Eq. 2.8})$$

Obviously, curvature of a surface depends on the angle θ . This angle is between tangent vector of a geodesic and $\hat{\mathbf{u}}$ direction. In other words, angle of the curve to $\hat{\mathbf{u}}$ direction. For a continuous surface, there are infinitely many curves, geodesics, and the curvature changes continuously while θ is changing.

Meaning of changing θ is selecting another curve on the surface. Actually different curves cause different θ but thinking as changing the angle causes different curve selection is also meaningful approach to understand relationships

between the angle and the curvature. As we select different curve, i.e. different angle, we will have different curvature. In Figure 12 below, different selections of curves on the surface and their normal planes are shown. As mentioned, all of the curves have different curvature at the point on the surface.

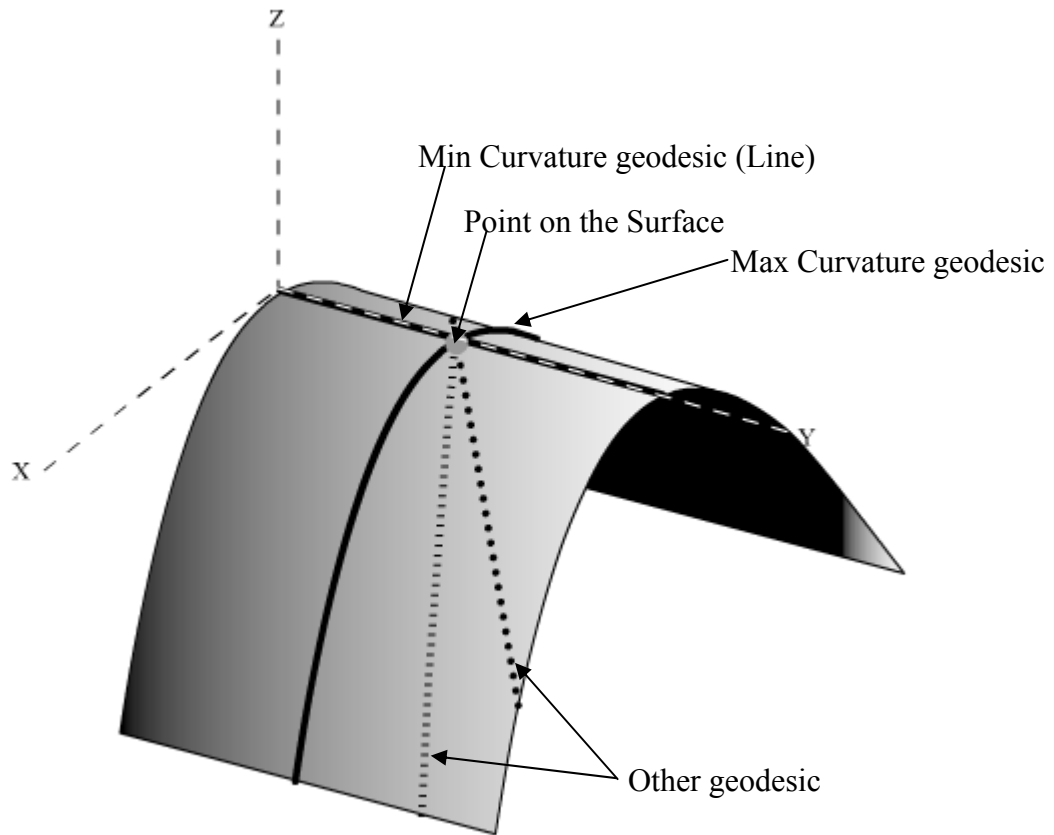


Figure 12 - Different curves on surface and principle curvature geodesics.

2.3.6 Principle Curvatures

Selecting different geodesic causes different curvatures. The angle is changing between $[0 \sim \pi]$. Observe that, curvature is periodic with $[0 \sim \pi]$. Let revise the formula [23];

$$\kappa_n(\theta) = \frac{r_{uu} \bullet n}{|r_u|^2} \cos^2(\theta) + \frac{r_{uv} \bullet n}{|r_u||r_v|} \sin(2\theta) + \frac{r_{vv} \bullet n}{|r_v|^2} \sin^2(\theta)$$

$$L = \frac{r_{uu} \bullet n}{|r_u|^2} \quad M = \frac{r_{uv} \bullet n}{|r_u||r_v|} \quad N = \frac{r_{vv} \bullet n}{|r_v|^2}$$

$$\kappa_n(\theta) = \frac{L-N}{2} \cos(2\theta) + M \sin(2\theta) + \frac{L+N}{2}$$

$$A_m = \sqrt{\left(\frac{L-N}{2}\right)^2 + M^2}$$

$$\bar{\kappa}_n = \frac{L+N}{2}$$

The curvature function is a sinusoidal wave with a phase shift. Its dc (mean) and ac components are $\bar{\kappa}_n$ and A_m , respectively.

If we plot the curvature values in the range;

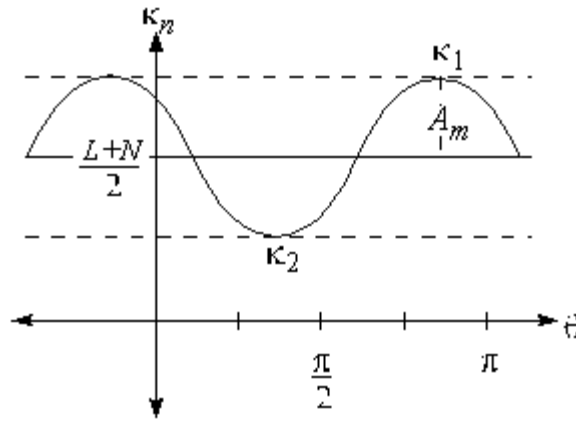


Figure 13 – Changing of Curvature with θ [23]

The peak values of the wave;

$$\kappa_{peak} = \bar{\kappa}_n \pm A_m$$

$$\kappa_1 = \bar{\kappa}_n + A_m \quad \kappa_2 = \bar{\kappa}_n - A_m$$

Two peaks of the wave, κ_1 and κ_2 , are known as *principle curvature* of the surface and, they are considered as an intrinsic property of surfaces in differential geometry. Indeed, if we consider Figure 12 again, it is shown that the one of the curve has maximum curvature, it is along x direction, and one of them is just a straight line, i.e. 0 curvature, it is along y direction. These two values provide an important clue about shape of the surfaces.


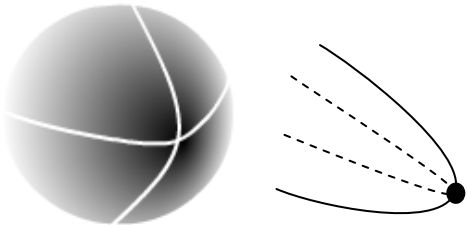
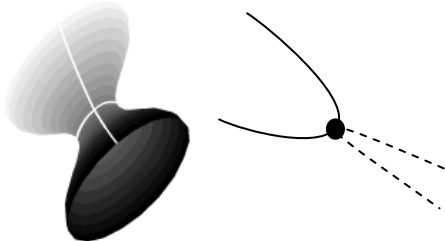
2.3.7 Mean and Gaussian Curvature Based Surface Classification

Johann Carl Friedrich Gauss (1777-1855) has proposed Gaussian curvature, K , to have information about a point on the surface just existence of knowledge κ_1 and κ_2 . The Gaussian curvature;

$$K = \kappa_1 \cdot \kappa_2 \quad (\text{Eq. 2.9})$$

Gauss classified the points on the surface as flat, elliptical and hyperbolic. If principle curvatures are zero, $K=0$, no change occurs near the point and obviously the plane shaped surface exists at the point. If both have the same sign, positive or negative, then $K>0$ and point is spherical. If $K<0$, i.e. different signed principle curvatures, the point is hyperbolic. The classifications are summarized in Table 1.

Table 1 – Gaussian classification of surfaces.

Plane	$K=0$	
Elliptical	$K>0$ Both principle curvatures have the same sign.	
Hyperbolic	$K<0$ Principle curvatures have opposite signs.	

Behind Gauss Classification, mean of the principle curvatures, H , provides some information about point on the patch. If mean curvature is positive, the

tangent vector turns clockwise and concave shape occurs at that point. Otherwise convex curvature is present.

$$H = \frac{\kappa_1 + \kappa_2}{2} \quad (\text{Eq. 2.10})$$

Using both H and K in surface classification is proposed by P.J. Besl [3]. This approach is better way for classification and it is widely used. For example, usage of only Gaussian curvature can not provide recognition of cylindrical surface as in Figure 12, because one of the curvatures, also K , is zero. Mean curvature together with Gaussian curvature provide more types for classification. The available types and their visualizations are shown in Table 2.

2.3.8 Derivation of K and H

For a surface, defined by $r(u, v)$, Gaussian curvature, K , value [23];

$$r(u, v) = (x_{(u,v)}, y_{(u,v)}, Z_{(u,v)})$$

$$\kappa_n(\theta) = \frac{r_{uu} \bullet n}{|r_u|^2} \cos^2(\theta) + \frac{r_{uv} \bullet n}{|r_u||r_v|} \sin(2\theta) + \frac{r_{vv} \bullet n}{|r_v|^2} \sin^2(\theta)$$

$$L = \frac{r_{uu} \bullet n}{|r_u|^2} \quad M = \frac{r_{uv} \bullet n}{|r_u||r_v|} \quad N = \frac{r_{vv} \bullet n}{|r_v|^2}$$

$$\kappa_n(\theta) = \frac{L - N}{2} \cos(2\theta) + M \sin(2\theta) + \frac{L + N}{2}$$

$$A_m = \sqrt{\left(\frac{L - N}{2}\right)^2 + M^2}$$

$$\bar{\kappa}_n = \frac{L + N}{2}$$


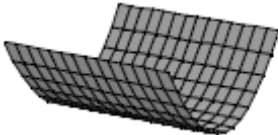
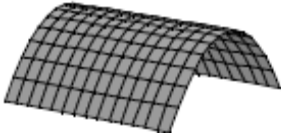

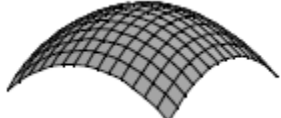
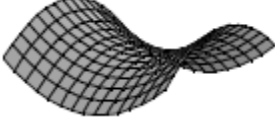
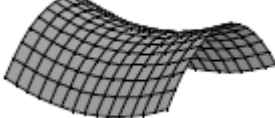

$$\kappa_1 = \bar{\kappa}_n + A_m \quad \kappa_2 = \bar{\kappa}_n - A_m$$

$$K = \kappa_1 \cdot \kappa_2$$

$$K = \left(\frac{L + N}{2}\right)^2 - (A_m)^2$$

$$K = \left(\frac{L + N}{2}\right)^2 - \left(\frac{L + N}{2}\right)^2 - M^2$$

Table 2 – Available classifications in HK segmentation.

Sign of H	Sign of K	Local (around the point) Surface Shape	Visualization [24]
0	0	Plane	
+	0	Concave Cylindrical (Valley)	
-	0	Convex Cylindrical (Ridge)	
+	+	Concave Elliptical (Pit)	
-	+	Convex Elliptical (Peak)	
0	-	Hyperbolic (Minimal Surface)	
-	-	Hyperbolic (Saddle Ridge)	
+	-	Hyperbolic (Saddle Valley)	

$$K = LN - M$$

$$K = \frac{r_{uu} \bullet n}{|r_u|^2} \cdot \frac{r_{vv} \bullet n}{|r_v|^2} - \left(\frac{r_{uv} \bullet n}{|r_u||r_v|} \right)^2$$

$$K = \frac{(r_{uu} \bullet n)(r_{vv} \bullet n) - (r_{uv} \bullet n)^2}{|r_u|^2 |r_v|^2 - (r_u \bullet r_v)^2}$$

Unit Normal vector, \mathbf{n} ;

$$U(u, v, z) = z - Z(u, v)$$

$$\nabla U = (-Z_u, -Z_v, 1)$$

$$\mathbf{n} = \frac{(-Z_u, -Z_v, 1)}{|(-Z_u, -Z_v, 1)|} = \frac{(-Z_u, -Z_v, 1)}{\sqrt{(Z_u^2 + Z_v^2 + 1)}}$$

The formulas above are extracted for any parametric surface. In our 3D range map definition, surface points are represented by;

$$\mathbf{r}(u, v) = (X_{(u,v)}, Y_{(u,v)}, Z(X_{(u,v)}, Y_{(u,v)}))$$

$$x(u, v) = u \cdot \Delta x$$

$$y(u, v) = v \cdot \Delta y$$

$$\mathbf{r}(u, v) = (u \cdot \Delta x, v \cdot \Delta y, Z_{(u,v)})$$

Note that, surfaces depth value, z , is actually a function of x and y . The parameterization by u and v defines the x , y and indirectly z values. For defined range map;

$$r_u = \frac{\partial \mathbf{r}(u, v)}{\partial u} = \left(\frac{\partial X(u, v)}{\partial u}, \frac{\partial Y(u, v)}{\partial u}, \frac{\partial Z(x, y)}{\partial u} \right)$$

$$r_v = \frac{\partial \mathbf{r}(u, v)}{\partial v} = \left(\frac{\partial X(u, v)}{\partial v}, \frac{\partial Y(u, v)}{\partial v}, \frac{\partial Z(x, y)}{\partial v} \right)$$

$$\frac{\partial X(u, v)}{\partial u} = \Delta x \quad \frac{\partial X(u, v)}{\partial v} = \Delta y$$

$$Z_u = \frac{\partial Z(x, y)}{\partial u} = \left(\frac{\partial Z(x, y)}{\partial x} \cdot \frac{\partial x}{\partial u} \right) + \left(\frac{\partial Z(x, y)}{\partial y} \cdot \frac{\partial y}{\partial u} \right) = Z_x \Delta x$$

$$Z_v = \frac{\partial Z(x, y)}{\partial v} = \left(\frac{\partial Z(x, y)}{\partial x} \cdot \frac{\partial x}{\partial v} \right) + \left(\frac{\partial Z(x, y)}{\partial y} \cdot \frac{\partial y}{\partial v} \right) = Z_y \Delta y$$

$$r_u = (\Delta x, 0, Z_x \Delta x) \quad r_v = (0, \Delta y, Z_y \Delta y)$$

Second derivatives;

$$r_{uu} = \frac{\partial r_u}{\partial x} \cdot \frac{\partial x}{\partial u} + \frac{\partial r_u}{\partial y} \cdot \frac{\partial y}{\partial u} = (0, 0, Z_{xx} \Delta x^2) \quad r_{vv} = (0, 0, Z_{yy} \Delta y^2) \quad r_{uv} = (0, 0, Z_{xy} \Delta y \Delta x)$$

Unit normal vector, \mathbf{n} , in our parameterization is $\mathbf{r}_u \times \mathbf{r}_v$;

$$r_u = (\Delta x, 0, Z_x \Delta x) \quad r_v = (0, \Delta y, Z_y \Delta y)$$

$$r_u \times r_v = \begin{bmatrix} \hat{x} & \hat{y} & \hat{z} & \hat{x} & \hat{y} \\ \Delta x & 0 & Z_x \Delta x & \Delta x & 0 \\ 0 & \Delta y & Z_y \Delta y & 0 & \Delta y \end{bmatrix} = (-Z_x \Delta x \Delta y, -Z_y \Delta x \Delta y, \Delta x \Delta y)$$

$$n = \frac{r_u \times r_v}{|r_u \times r_v|} = \frac{(-Z_x \Delta x \Delta y, -Z_y \Delta x \Delta y, \Delta x \Delta y)}{\sqrt{Z_x^2 \Delta x^2 \Delta y^2 + Z_y^2 \Delta x^2 \Delta y^2 + \Delta x^2 \Delta y^2}} = \frac{\Delta x \Delta y \cdot (-Z_x, -Z_y, 1)}{|\Delta x \Delta y| \sqrt{Z_x^2 + Z_y^2 + 1}}$$

$$= S \frac{(-Z_x, -Z_y, 1)}{\sqrt{Z_x^2 + Z_y^2 + 1}}, \quad S = \text{sign}(\Delta x \Delta y) = \pm 1$$

Then K ;

$$K = \frac{(r_{uu} \bullet n)(r_{vv} \bullet n) - (r_{uv} \bullet n)^2}{|r_u|^2 |r_v|^2 - (r_u \bullet r_v)^2}$$

$$r_{uu} \bullet n = (0, 0, Z_{xx} \Delta x^2) \bullet \left(S \frac{(-Z_x, -Z_y, 1)}{\sqrt{Z_x^2 + Z_y^2 + 1}} \right) = S \frac{Z_{xx} \Delta x^2}{\sqrt{Z_x^2 + Z_y^2 + 1}}$$

$$r_{vv} \bullet n = S \frac{Z_{yy} \Delta y^2}{\sqrt{Z_x^2 + Z_y^2 + 1}} \quad r_{uv} \bullet n = S \frac{Z_{xy} \Delta x \Delta y}{\sqrt{Z_x^2 + Z_y^2 + 1}}$$

$$(r_{uu} \bullet n)(r_{vv} \bullet n) - (r_{uv} \bullet n)^2 = S^2 \frac{Z_{xx} \Delta x^2 \cdot Z_{yy} \Delta y^2 - Z_{xy}^2 \Delta x^2 \Delta y^2}{|Z_x^2 + Z_y^2 + 1|}$$

$$S^2 = 1$$

$$(r_{uu} \bullet n)(r_{vv} \bullet n) - (r_{uv} \bullet n)^2 = \Delta x^2 \Delta y^2 \frac{Z_{xx} Z_{yy} - Z_{xy}^2}{|Z_x^2 + Z_y^2 + 1|}$$

$$\begin{aligned}
|r_u|^2 &= (\Delta x^2 + Z_x^2 \Delta x^2) & |r_v|^2 &= (\Delta y^2 + Z_y^2 \Delta y^2) \\
(r_u \bullet r_v)^2 &= Z_x^2 \Delta x^2 Z_y^2 \Delta y^2 \\
|r_u|^2 |r_v|^2 - (r_u \bullet r_v)^2 &= (\Delta x^2 + Z_x^2 \Delta x^2)(\Delta y^2 + Z_y^2 \Delta y^2) - Z_x^2 \Delta x^2 Z_y^2 \Delta y^2 \\
&= \Delta x^2 \Delta y^2 (Z_x^2 + Z_y^2 + 1) \\
K &= \frac{(r_{uu} \bullet n)(r_{vv} \bullet n) - (r_{uv} \bullet n)^2}{|r_u|^2 |r_v|^2 - (r_u \bullet r_v)^2} = \frac{\Delta x^2 \Delta y^2 \frac{Z_{xx} Z_{yy} - Z_{xy}^2}{|Z_x^2 + Z_y^2 + 1|}}{\Delta x^2 \Delta y^2 (Z_x^2 + Z_y^2 + 1)} \\
K &= \frac{Z_{xx} Z_{yy} - Z_{xy}^2}{(Z_x^2 + Z_y^2 + 1)^2}
\end{aligned}$$

Similarly H ;

$$H = \frac{(1 + Z_x^2) Z_{yy} - 2 Z_x Z_y Z_{xy} + (1 + Z_y^2) Z_{xx}}{2 \cdot (Z_x^2 + Z_y^2 + 1)^{3/2}}$$

These two formulas can be revisited by defining common variables;

$$\begin{aligned}
norm &= (Z_x^2 + Z_y^2 + 1) \\
E &= 1 + Z_x^2 & e &= Z_{xx} \\
F &= Z_x Z_y & f &= Z_{xy} \\
G &= 1 + Z_y^2 & g &= Z_{yy} \\
K &= \frac{e \cdot g - f^2}{norm \cdot (E \cdot G - F^2)} \\
H &= \frac{2 \cdot f \cdot F - e \cdot G - g \cdot E}{2 \cdot (E \cdot G - F^2) \sqrt{norm}} \tag{Eq. 2.11}
\end{aligned}$$

These notations will be very useful for implementation process.

2.4 Real Time Concept

Digital signal processing has been very popular for past years. The DSP algorithms are designed by mathematical tools in computers and usually implemented using software in computers again. Most of the case, output validity

is already known and it is the first verification test for both design and implementation time. This approach is meaningful because beginning of the project, the output accuracy is the most wanted requirement. If the implementation provides somehow necessary speed in software or in hardware, the design seems to meet all criteria. Indeed, algorithm designers usually do not consider implementation platform while designing. However, most of the time, especially in software implementations, the platform is very important, because computer architectures may differ from platforms to platforms. Speeds of RAMs, CPU frequency, main bus architecture, cache size etc... are very significant quantities for speed requirements. Also, in multi threaded environments, operating system and other running processes affect performance of algorithms. Therefore, before starting to implement an algorithm, it is very essential to know the speed requirements. If a real time running prerequisite exists, the algorithm implementation is changed according to the target requirements, both output validity and speed.

Basically, there is no exact definition of real time concept. The condition is given by the target algorithm. For example, a person tracker system in a garden may process at the rate of 1 frame/sec. On the other hand, for a vehicle tracker, this is very slow rate, since the vehicles move much faster than people. Therefore real time criteria and the expected outputs of system should be decided together at the beginning of the project.

The next step of real time design, after the requirements are decided, is algorithm development according to target platform. During design time, if the speed requirement is discarded and only providing a valid output is considered, it is very hard to achieve real time processing at the end of system realization. In this case, changing the target platform according to developed algorithm is usually used. Unfortunately, this approach does not work always. Finding very fast processors or hardware may be impossible or it can be very expensive. Therefore the best way of a system design is deciding both output and timing criteria altogether and then starting to development considering both criteria together.

2.4.1 Hardware & Software Partitioning

According to requirements, hardware/software partitioning should be determined. Software and hardware partitioning is a very meaningful approach for complex system design. Cost of developing all algorithms in software can be too much, or it can be impossible. One processor makes one computation at a time. The increasing of clock frequency may not enough to deal with timing requirement. Increasing processor count also increases both money and power costs. Also, parallel programming in software is not an easy approach. On the other hand, using only hardware to realize an algorithm would be a very heuristic try. Digital design is much harder than software and it is not useful way for algorithm developing. The algorithm should be already confirmed and it must be developed according to hardware facts. An untested algorithm can not be successful in hardware. Another hardware design struggle is required design time. Hardware designs take much more time than software designs.

The first key point of partitioning is parallelism. Trying to implement a sequential algorithm in hardware, even using pipelined architecture, may not be useful when considering design time and output accuracy. For these types of algorithms CPUs are more useful. Before starting hardware design, the parallel running stages should be constructed and these stages should be designed according to hardware facts. Another key point is recursive callings. A pure recursive algorithm should not be implemented in hardware. The recursive algorithms are fit to software; they are not so suitable for hardware design. If the hardware stages of the algorithm contain a recursive case, the stages should be revised and the algorithm should be changed. Thirdly, control of algorithms and systems should be kept in software. One of the main advantages of software is its flexibility. Changing software code is easy, so using state machines in hardware for algorithm control is not a good approach.

The partitioning approach has been researched and many algorithms have been implemented in this way by many researchers. An example of fixed point co-processor for floating point conversion has been performed in [25], and the researches continue in this science.

2.4.2 Real Time Criteria in HK Analysis

In feature extraction, time requirements depend on what purpose the features are used for. For an elementary object recognition, 100 msec processing time for a single frame is sufficient. Humans can not recognize object in 100 msec, therefore it is enough to have a valid output in curvature analysis for object recognition in this aspect. Of course, for different purposes, 100 msec. may not be adequate. In this thesis, we use 100msec/frame limit for real time curvature analysis. According to timing condition and speed measurements of software based curvature feature extraction, it is very useful to develop a hardware curvature processor. As already mentioned, real range maps are generated in hardware [17, 18]. The next step is processing the maps in hardware and providing features to software to decrease the processing load.

2.4.3 Real Time Hardware Platforms

Hardware design can be implemented in many platforms, such as ASICs, CPLDs, FPGAs etc... Today, the most preferred hardware design realization platform for testing purpose is reconfigurable FPGAs. Reconfigurable FPGAs are not one time programmable logic devices and their logics are designed by higher level hardware description languages, HDL. Also the designs can be converted to other platforms, supporting HDL. ASICs are application specific and they are not a programmable device. They are a custom logic device, but faster than FPGAs. Also ASIC design requires much more time than FPGA design. Therefore we prefer to use reconfigurable FPGAs to realize the hardware algorithms. The used HDL is VHDL.

2.5 FPGA Basics

Field programmable gate arrays, FPGAs, consist of configurable logic blocks (CLB), clock managers (PLL, DCM), on chip memory (BRAM), hard multipliers, summers and a matrix formed interconnection grids. Also modern

FPGAs have hard Ethernet MAC, PCI-E and hard processors etc... CLBs are configured according to desired application and logic elements of the design are constructed. These logics are connected via programmable interconnection lines. Therefore a high flexibility is achieved on a single device. Also FPGAs have dedicated memories and FIFO elements. Merely all logic designs require small memories or FIFOs. These elements are connected to designed logics (CLB) again via interconnection lines. Also, for signal processing application, FPGAs have hard DSP chips, providing fast multiply and accumulate logics. Moreover, the next generation FPGAs have hard processors. The processors and logic circuits in a single chip open System on Chip, SOC, era in electronics science. With these advantages, FPGAs are starting to be a very powerful hardware implementation platform.

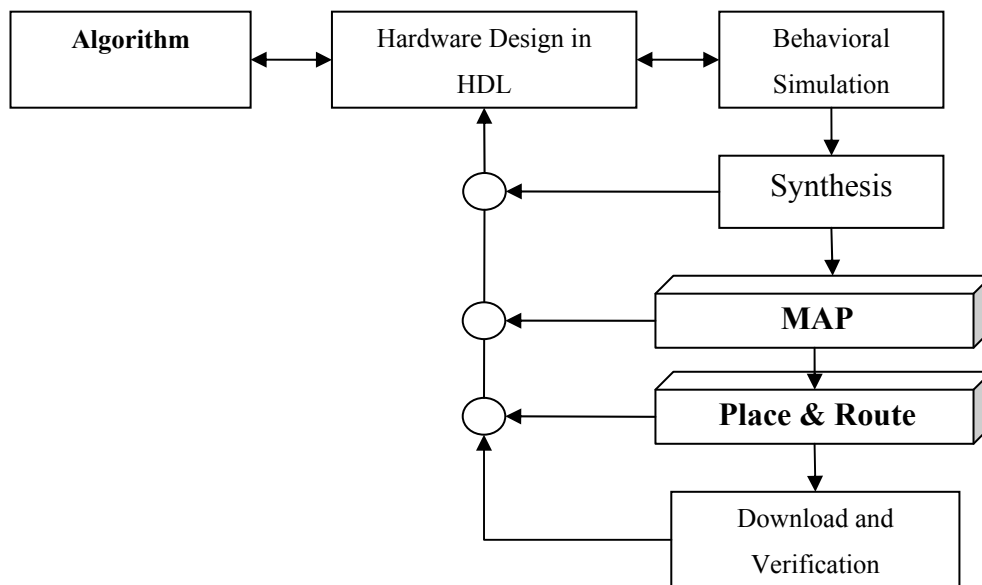


Figure 14 – FPGA design flow.

A typical FPGA design flow is shown in Figure 14. The algorithm is developed by mathematical tools, simulated in higher level software, MatLab for example. The algorithms should be in, of course, hardware sense. Then the HDL is prepared. Using schematic, Verilog or VHDL is possible. The next step is

behavioral simulation. In this step, the outputs of behavioral simulation of the HDL code should be match with outputs of the algorithm. Otherwise the revising is required. When the behavioral simulation is finished, hardware realization can be started. The first stage is synthesis. In this stage, the HDL code is converted into logic components, gates flip-flops etc... Sometimes HDL may contain non-synthesizable codes or synthesizing may not be possible for the codes. In this case HDL is revised again. If synthesis is finished successfully, the logic design can be transferred to the target FPGA. For this purpose, a mapping is started; elements of the logic circuit are mapped to target FPGA's elements. Unfortunately, if the design has too much elements for target FPGA, mapping fails. The design must be revised to have lower logic elements or the FPGA must be changed with a higher density chip. The last realization step is place and route. Here, the mapped logic elements are placed according to the required frequency of design and then they are routed, connected. The CLBs are static in the chip, therefore putting which logic circuits into which CLB to achieve desired frequency as the result of connections requires a brilliant algorithm and it takes time. Sometimes, designed circuits can not reach the required clock frequency and place and routing fail for target FPGA. Again, the failed connection has to be revised and the timing problem must be solved. After hardware realization is completed, the chip is programmed and output verification is made. The outputs may not match with behavioral simulation. Usually it is due to clock latency of used hard logic elements in FPGAs. Using internal signal browsers in the chip, the problem can be solved.

Obviously, any problem in any stage causes restart of the whole design flow, again. This time consuming process is one of the biggest disadvantage of the hardware implementation.

2.5.1 Used FPGAs

Implementation of digital curvature processor has been performed in two different FPGAs on two development boards, XILINX ml403 and ml505. Performance of the design is measured on these two boards. ml403 has DDR

SDRAM at 200MHz and ml505 has DDR2 SDRAM at 400MHz. ml403 has less on chip memory than ml505. The first version of DCP implemented on ml403 and the second version, which requires more on chip memory, is implemented on ml505.

2.6 System on Chip (SoC) Design

System on Chip design strategy has started to be a popular system building approach recently. Especially, both increasing importance of hardware software partitioning and improvements in FPGA technology boost SoC design strategy. In this approach, the software and hardware are integrated into a single chip. One or more processor implemented in FPGA provides software realization while hardware logics are running in the same chip. The FPGAs having dedicated hard processors provides faster processing speed in software. Also HDL based processors, known as soft-processors, are also very useful because there is no limitation of CPU count. Multi processor based single on chip designs in FPGAs has been researched by many scientists and many design have been proposed for different image and video applications [26, 27].

SoC design architecture adds another design flow into FPGA design steps. In Figure 15, a SoC design steps are shown. The new flow includes both hardware and software development phases. The generated software codes can be downloaded into memories in FPGA or the external memories, SRAM, DDR etc...

In digital curvature processor, SoC design has been used. In the following chapters, the design and its realization are explained.

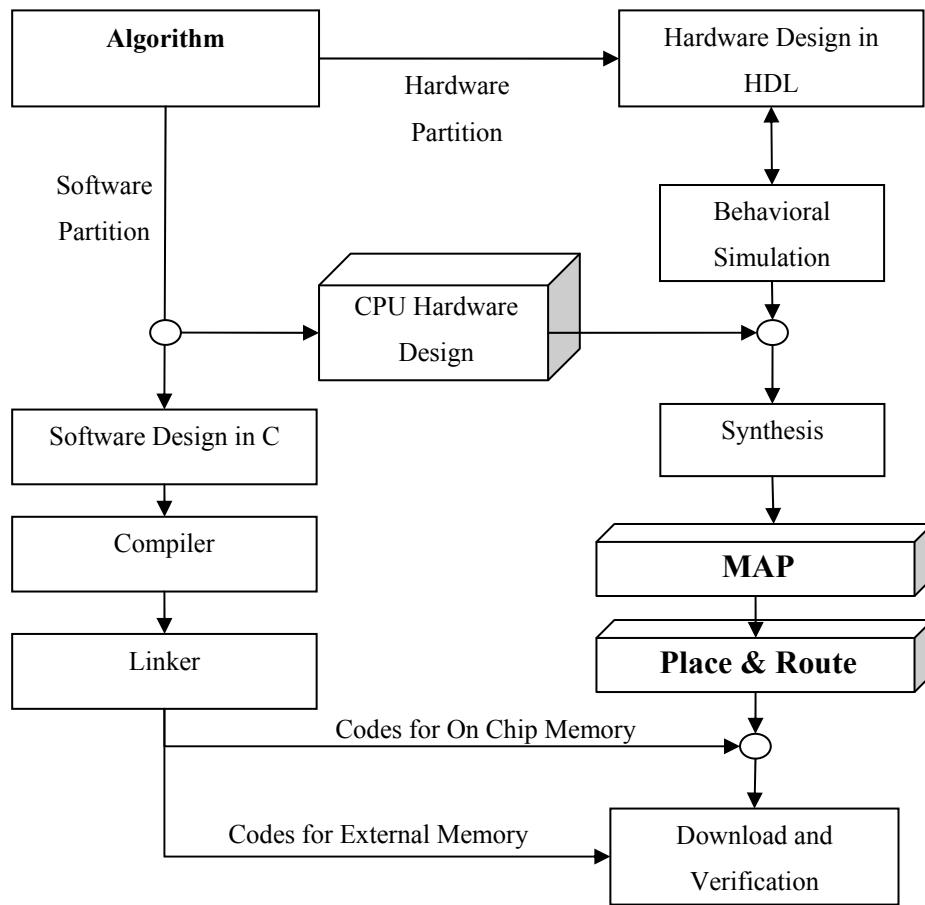


Figure 15 – Hardware software partitioning and System on Chip design flow in FPGAs.

CHAPTER 3

DIGITAL PROCESSING & HARDWARE IMPLEMENTATION

In this chapter, the requirements of hard curvature processor are decided and design strategies for these requirements are explained. The design steps are compared with software based approaches. Details of expected errors resulting from approximations are also given.

3.1 Mean and Gaussian Curvature Computation in Discrete Domain

In previous sections, continuous time principle curvature calculations are explained. Mean and Gaussian curvatures depend on principle curvatures and their derivation are already given. Digital processing, on the other hand, involves discrete samples and discrete computations. Therefore the derivations in continuous domain, for H and K , should be considered in discrete sense.

3.1.1 Estimation Techniques

In discrete domain, two basic approaches for calculation are available. The process may contain estimation of differential arguments in the H and K formulas, Eq. 2.11. **Hata! Başyuru kaynağı bulunamadı.** In this case a surface fitting approach, Spline based estimation or other techniques provide differential arguments. The second way is directly estimation of principle curvatures and then

calculating mean and Gaussian curvatures. Remember that Gaussian curvature is product of minimum and maximum curvatures, Eq. 2.9 and Eq. 2.10.

Flynn and Jain [28] have already worked on digital estimation of curvatures. Their research is very helpful for comparing the estimation techniques. In digital curvature calculation, the most preferred and good technique is surface fitting and it is also proven at that work. One usage of surface fitting is available in [6]. However these types of techniques, surface fitting or Splines etc... require too much computation and involve inverse matrix operations. These computations are not proper in hardware design. To overcome the disadvantage, we prefer numerical approximation techniques. Searching in different angles to determine principle curvatures can be used or directly approximation of first and second derivatives can be possible. The former, again, contains more computations than the latter. Also, in this work, we are using a Monga patch, graph surface range map, therefore we prefer to use directly estimation of differential arguments. H and K formulas in Eq. 2.11, derived in the previous chapter;

$$\begin{aligned}
 r(u, v) &= (x(u, v), y(u, v), Z(x, y)) \\
 x(u, v) &= u \cdot \Delta x \quad y(u, v) = v \cdot \Delta y \quad z = Z(x, y) \\
 norm &= (Z_x^2 + Z_y^2 + 1) \\
 E &= 1 + Z_x^2 \quad e = Z_{xx} \\
 F &= Z_x Z_y \quad f = Z_{xy} \\
 G &= 1 + Z_y^2 \quad g = Z_{yy} \\
 K &= \frac{e \cdot g - f^2}{norm \cdot (E \cdot G - F^2)} \\
 H &= \frac{2 \cdot f \cdot F - e \cdot G - g \cdot E}{2 \cdot (E \cdot G - F^2) \sqrt{norm}}
 \end{aligned}$$

Note that, the curvature depends on only differential arguments. The required estimations are;

$$Z_x \quad Z_{xx} \quad Z_y \quad Z_{yy} \quad Z_{xy}$$

3.1.2 Noise on Surfaces and Definition of Classification Type Ranges

As we deal with real surfaces, we have to face some noise on the patch. Even on a plane, we have some roughness. Actually, in real life, the plane definition disappears, because all surfaces, and planes, are constructed by other small surfaces. The definition of plane, in this context, depends on how much irregularities you accept for flatness.

More obvious explanation can be given by a straight line in 2-D domain. In Figure 16 below, a perfect line is shown in (a), and a noisy line is in (b). It is still a straight line because it is in the defined ranges, dashed thresholds. The ranges are bending limits, i.e. curvature, which is convexity or concavity information, for straightness. In (c) the curve exceeds the thresholds therefore it is not a line.

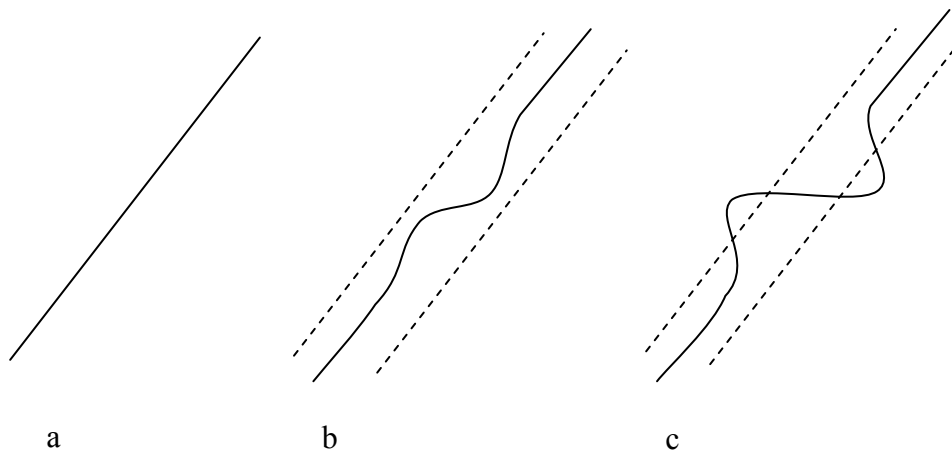
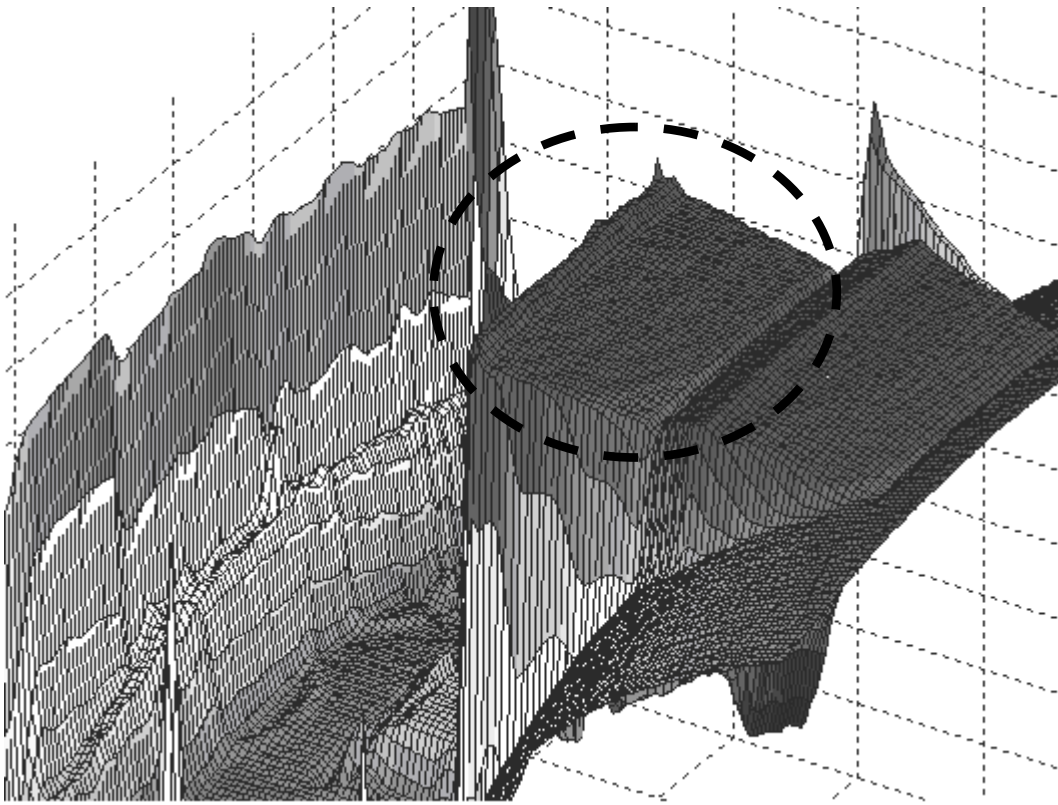
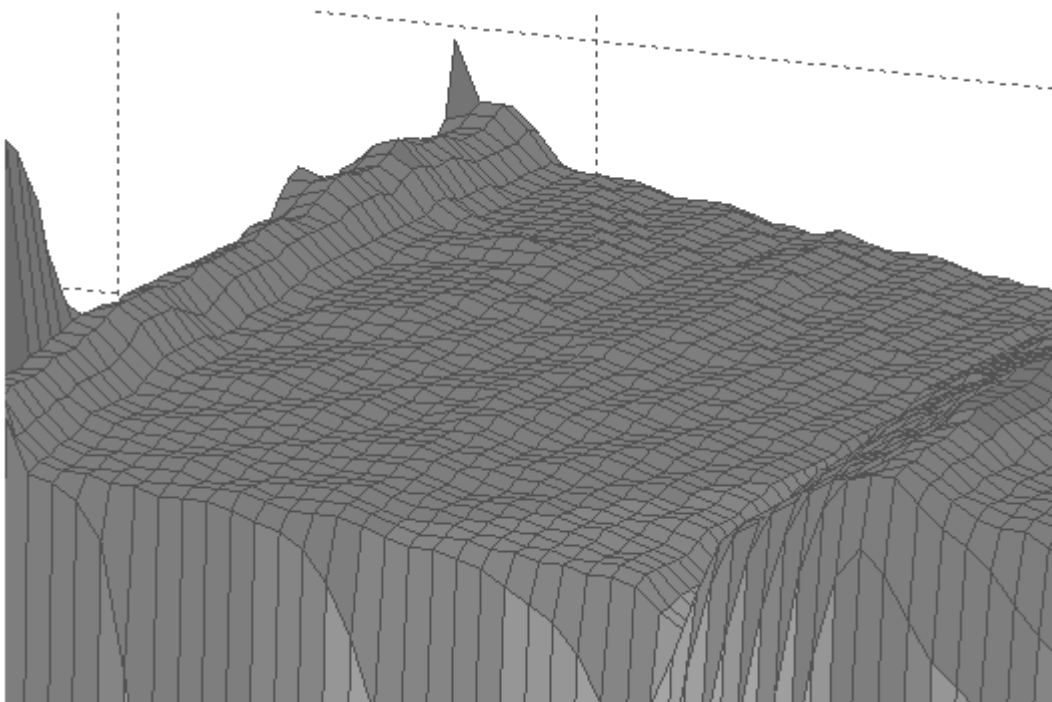


Figure 16 – Bending thresholds in 2-D.

The appearance of a real surface, in discrete domain, is shown in Figure 17. The closer appearance is also given in (b). Obviously, a close looking to the plane makes visible the roughness on the surface.



a



b

Figure 17 – A closer look to a planar surface makes irregularities more visible

Reason for roughness can be shape of the surface, i.e. the surface is not flat, or it might be due to measurement errors at range map generation step. To deal with these noises, usually a low pass filtering applied and some thresholds are defined.

3.1.3 Filtering Range Data

A low pass, smoothing, 2-D filter may help us to overcome large irregularities on the range map. However, filtering suppresses the details and smoother the edges. Also, filter size and coefficients depend on application and a general purpose filtering is not helpful in real curvature analysis. Therefore filtering hardware is outside of curvature processor and it is a separate logic, depending on application. The curvature processor accepts smooth enough range data as input signal. It applies thresholds to overcome the noises. The block diagram of the architecture is shown in Figure 18.

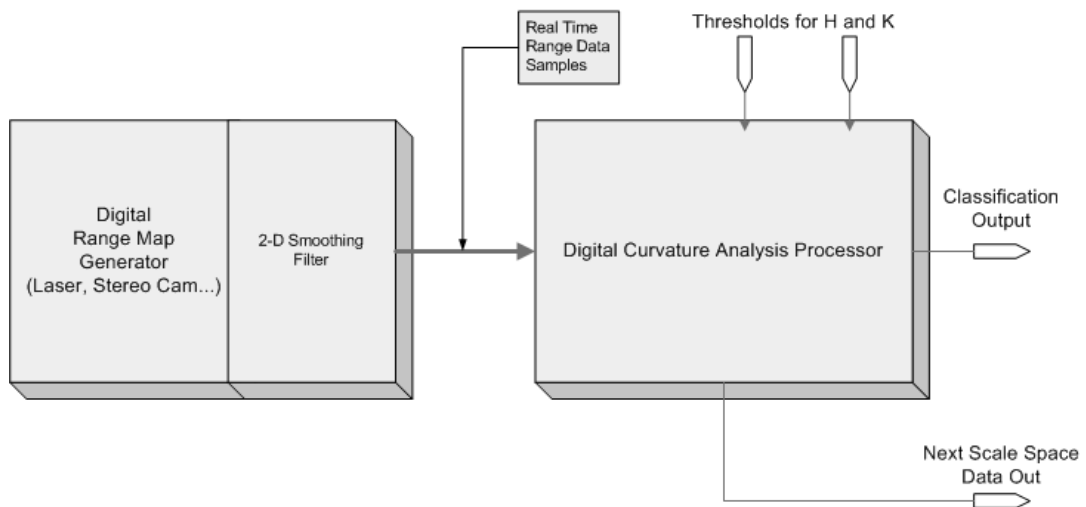


Figure 18 – DCP and Range Map Generator connectivity

3.1.4 Thresholds

Mean curvature, H , is used to decide concavity of the curve. Therefore a threshold for H determines straightness ranges. In Table 2, effects of H on surface classification can be easily seen. Defining thresholds and a new H , sign of H ;

$$H' = \begin{cases} 0 & -Th < H < Th \\ \text{sign}(H) & \text{otherwise} \end{cases} \quad (\text{Eq. 3.1})$$

Using 2 different thresholds, for positive and negative H , can be possible. In that case, convex and concave decision limits are independent. In this work, a single threshold is used.

As the same manner, Gaussian curvature decides whether surface is bended in two directions or not. The thresholds for K , therefore, acceptance limits of curvature along both principle directions. New K definition;

$$K' = \begin{cases} 0 & -Tk < K < Tk \\ \text{sign}(K) & \text{otherwise} \end{cases} \quad (\text{Eq. 3.2})$$

In different sources, some relations are found between Tk and Th values. The relation arises from the mathematical relationship between H and K values. However, using a static relation reduces flexibility of design. Therefore in curvature processor, two independent thresholds are available as it is seen in Figure 18. The upper level algorithm, user of the processor, can change the thresholds independently.

3.2 Hardware Based Digital Curvature Processor Proposal

3.2.1 Real Time Requirements

As in all designs, the first requirement is output validity. Digital curvature processor should provide a valid, expected, output. For the same inputs, hardware outputs of the algorithm should match with its simulation outputs.

The second requirement is speed. To provide a real time running for object recognition, 100msec/frame is enough, where analysis and generations of all scale spaces are included.

One of the important requirements is stability in timing. The DCP, digital curvature processor, should provide its outputs always in the same time spans. In other words it should have the same speed in each running, independent from input signals.

Additional hardware requirement is maximum running frequency in FPGA. In our design, two different DCPs are designed and their running frequencies are;

FPGAs→ DCP Type↓	xc4vfx12-10ff668 (XILINX ml403 board)	xc5vlx50t-1ff1136 (XILINX ml505 board)
DCP 1	100MHz	100MHz (Implemented) 110MHz Max.
DCP 2	-	75MHz (Implemented) 89MHz Max.

3.2.2 Control of Parameters

Another facility of digital hardware can be its flexibility, reconfiguration while running. The hardware should provide changing its parameters to its owner. A block representation is shown in Figure 19. Start and reset, of course, is one of the basic controls. The important parameters, thresholds and scale space count, are also provided as inputs.

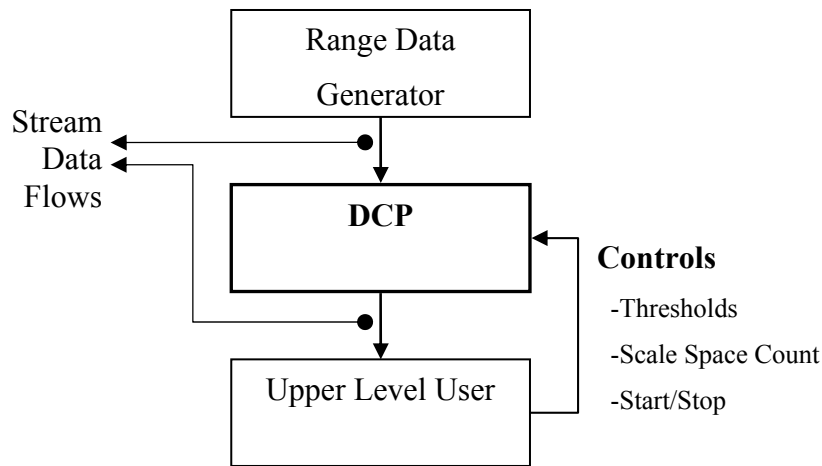


Figure 19 – Configuration of DCP by upper level user.

The thresholds are configurable while DCP is running. As we are proceeding with the chapters, we will provide details of parameters and signals.

3.2.3 Range Map Data Format and Quantization

Range map contains 4 fields, as shown in Figure 3. Three of them, x, y and z, are range information. In software applications, these fields are usually floating point, IEEE 754, representations. However using floating point units for all calculations utilizes too much logic unit in FPGA. Therefore it is preferable to use fixed-point representations. Multiplication and summation is much easy in fixed-point numbers. Also any standard VHDL synthesizers deal automatically with design of these arithmetic units. They use embedded DSP slices in the FPGA, so without worrying about too much detail, designers can realize the algorithms.

As already mentioned in the previous chapter, fixed-point number may cause quantization errors in range data. In this work, range data fixed-point word length is 10, fraction length is 3. The numbers are unsigned. An example representation of 65,125 is;

Word Length (Wl=10)									
1	0	0	0	0	0	1	0	0	1
							Fraction Length (Fl=3)		

The ranges of the representation is [0~127.875]. Note that in decimal numbers a 10 bit unsigned number can represent [0~1023].

The range map is a graph surface and it is;

$$\begin{aligned}x &= X(u, v) \\y &= Y(u, v) \\z &= Z(x, y) = Z(X_{(u,v)}, Y_{(u,v)})\end{aligned}$$

We already defined unit steps of x and y as Δ_x and Δ_y . The sampling errors are ε_x and ε_y .

$$\begin{aligned}X(u, v) &= (u * \Delta_x) \pm \varepsilon_x \\Y(u, v) &= (v * \Delta_y) \pm \varepsilon_y \\0 &< \varepsilon_x < \Delta_x \\0 &< \varepsilon_y < \Delta_y\end{aligned}$$

In our hardware implementation, unit steps of range map are defined as;

$$\Delta_x = \Delta_y = 1$$

In this thesis, we assume that $\varepsilon_x = 0$ condition is always ensured. In other words, we do not accept error in sampling. Assumption of $\Delta_x = \Delta_y = 1$ converts the range map to a graph surface, as in Eq. 2.3.

$$\begin{aligned}r(u, v) &= (x(u, v), y(u, v), Z(x, y)) \\x(u, v) &= u \cdot \Delta_x \quad y(u, v) = v \cdot \Delta_y \quad z = Z(x, y) \\x = u \quad y = v &\Rightarrow z = Z(u, v) \\r(u, v) &= (u, v, Z(u, v))\end{aligned}$$

Range map data cells, x, y, z and valid/invalid flags, are merged into a 32 bit big-endian word. The word format is;

32 bit big-endian				
Empty	Valid/Invalid	Z	Y	X
1 bit	1 bit	10 bit	10 bit	10 bit

By this way, in a 32bit CPU containing design, SoC design, the CPU and DCP may access the range data via multi user access memory. This approach is used in this design.

If graph surface condition is not satisfied in the range map, bilinear or bicubic interpolation techniques can be used to eliminate error in x and y data, as already mentioned in Figure 5 in Introduction chapter.

3.2.4 Real Time Stream Signal Input and Parallelism

Real time processing requires streaming analysis. Software based curvature algorithms use chunk of range map data, i.e. they need whole of range data. On the other hand, in hardware we prefer to use signal vectors, including consecutive samples. Using chunk of data in hardware is still possible, but in this case real time operation can not be possible. Implementation of mass data processor may be still faster than software approach, but it is not a desired way of hardware implementation at all.

Parallelism is the most important benefit of hardware design, and FPGAs. Therefore, digital signal processing in hardware, the algorithms, naturally, must be able to be partitioned into parallel sections. Also some sequential sections can be parallelized. The DCP processes 8 consecutive samples in parallel. We define input signal, vector, R , as;

$$\bar{R}_i = \begin{bmatrix} R_{v,u} \\ R_{v,u+1} \\ R_{v,u+2} \\ R_{v,u+3} \\ R_{v,u+4} \\ R_{v,u+5} \\ R_{v,u+6} \\ R_{v,u+7} \end{bmatrix} = \begin{bmatrix} z_{v,u} & y_{v,u} & x_{v,u} \\ z_{v,u+1} & y_{v,u+1} & x_{v,u+1} \\ z_{v,u+2} & y_{v,u+2} & x_{v,u+2} \\ z_{v,u+3} & y_{v,u+3} & x_{v,u+3} \\ z_{v,u+4} & y_{v,u+4} & x_{v,u+4} \\ z_{v,u+5} & y_{v,u+5} & x_{v,u+5} \\ z_{v,u+6} & y_{v,u+6} & x_{v,u+6} \\ z_{v,u+7} & y_{v,u+7} & x_{v,u+7} \end{bmatrix} \quad \begin{array}{l} v = i / 128 \quad u = i - (v \cdot 128) \\ u, v, i : \text{unsigned integer} \\ i \in [0, 8, 16, \dots, 112, 120] \\ v \in [0..127] \\ u \in [0..127] \end{array}$$

Graphical demonstration of the collection of all vectors is shown in Figure 20. This demonstration explains how a range map stays in a computer memory. The organization is an example illustration and it may change in different applications. Obviously, in Figure 20, all vectors, i.e. chunk of data, are available. Conversely, DCP use only one vector at a time. Size of the vector, R , is 256bit, 32x8. In Figure 21, input versus time graphic is given. DCP accepts input samples as stream vectors.

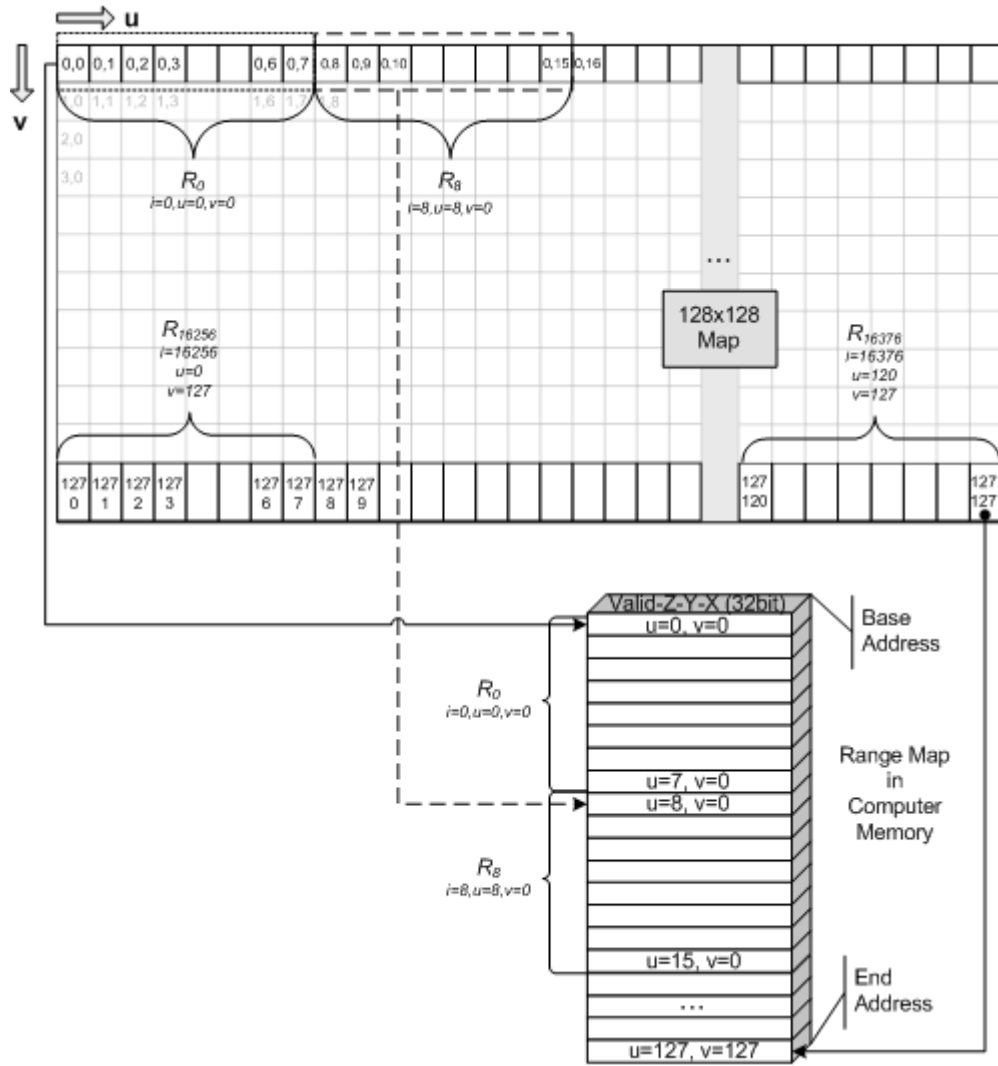


Figure 20 – 128x128 range map organization in a RAM

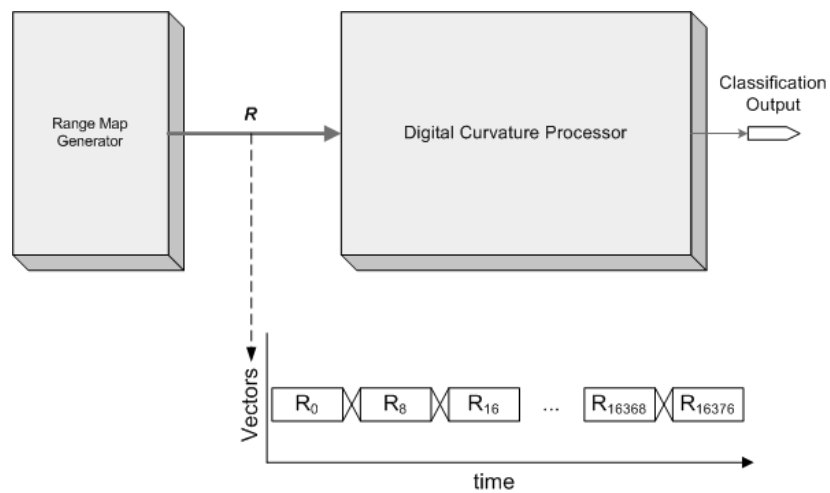


Figure 21 – Vector flow between Range map generator and DCP

Stream data input in DCP also forces range map generator to obey stream data timing scheme. Therefore a proper interface is required between range map generator and DCP. To keep synchronization in input, DCP provides a few control signals to range map generator. Also an address decoder is added into DCP. With the help of the addressing decoder and a RAM controller, DCP can be used on chunk of data in RAM. Two different possible usage of DCP is shown in Figure 22. In both applications, DCP takes sample vectors as shown in Figure 21, streaming vectors.

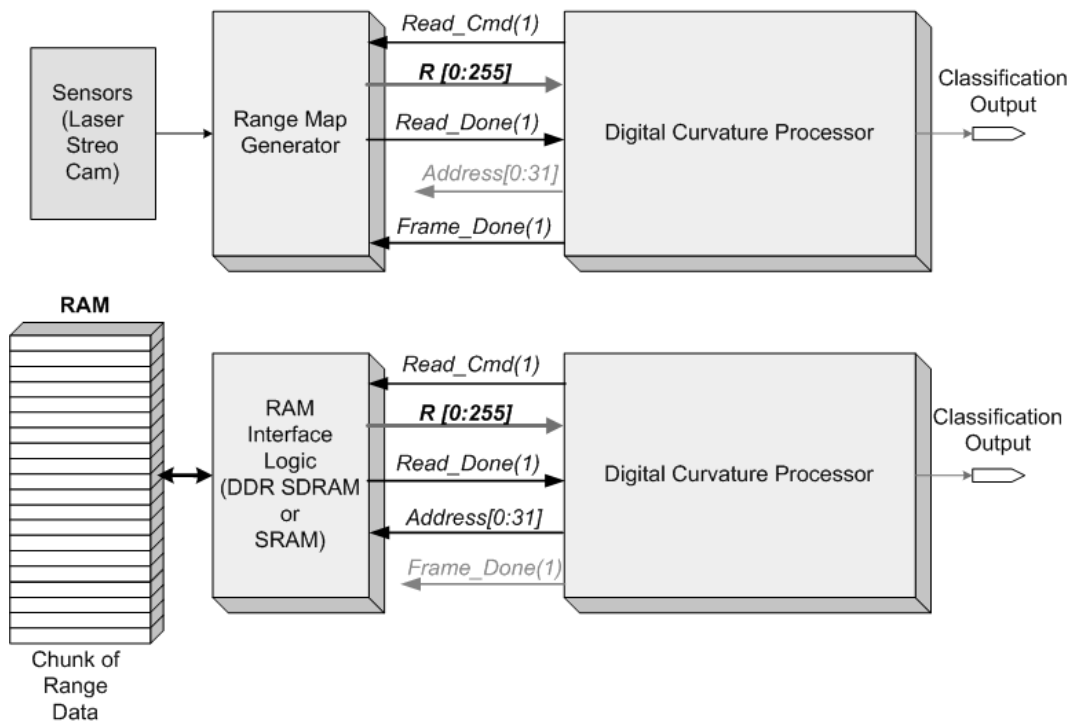


Figure 22 – DCP application with range map generator (top) or a RAM (bottom)

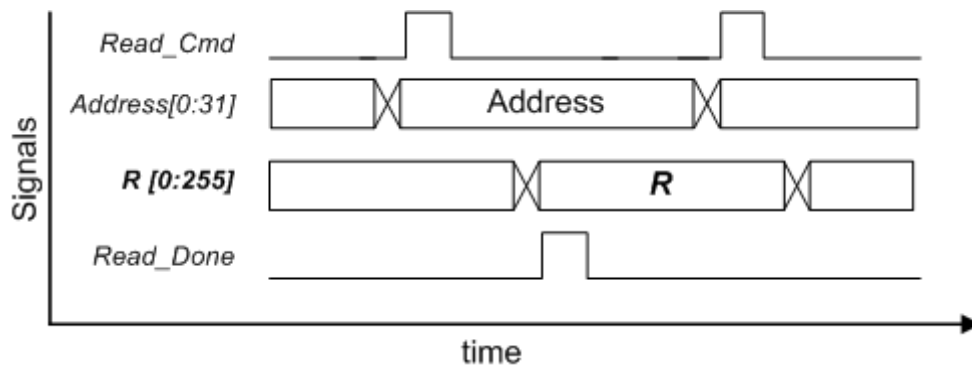


Figure 23 – Timing diagram of interface signals

Timing of synchronization signals are given in Figure 23. Once *Read_Cmd* signal is asserted, DCP is ready to accept *R* vector. A ram controller or a range generator puts *R* and asserts *Read_Done*. *Address* signal is for RAM controllers and it is synchronized with *Read_Cmd* signal. Also a *Frame_Done* signal is available for frame synchronization. All signals are active-high.

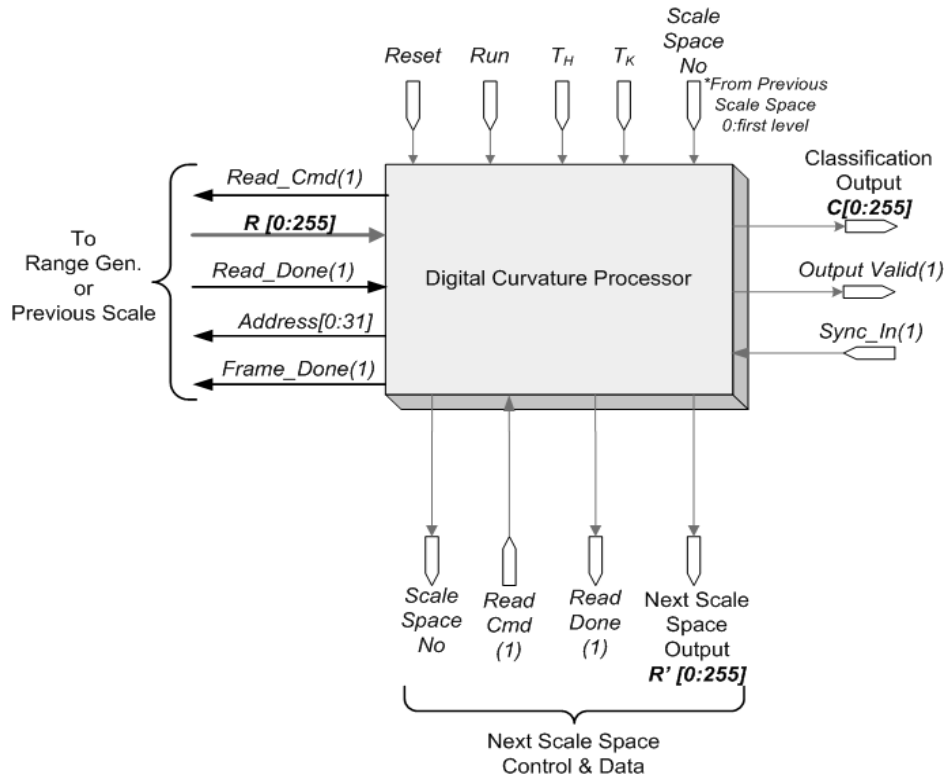


Figure 24 – Control signals for synchronization between DCP logics.

3.2.5 Pipeline Scale Space Analysis

It is desired to analyze curvatures of surface in multiple scale space levels in this project. Achievement of multiple scale space analysis in DCP has been designed as follows;

One instance of DCP analyzes only one scale space. The desired scale space level, i.e. scale space no, should be provided to DCP, as an input parameter. Also input vectors should be proper for the scale space level. Generation of the scale space range map vectors, therefore, must be generated by another DCP. As a result,

each DCP processes its scale spaces vectors and also it generates vectors of next scale space level *at the same time*. Obviously, this approach differs from usual scale space analysis techniques.

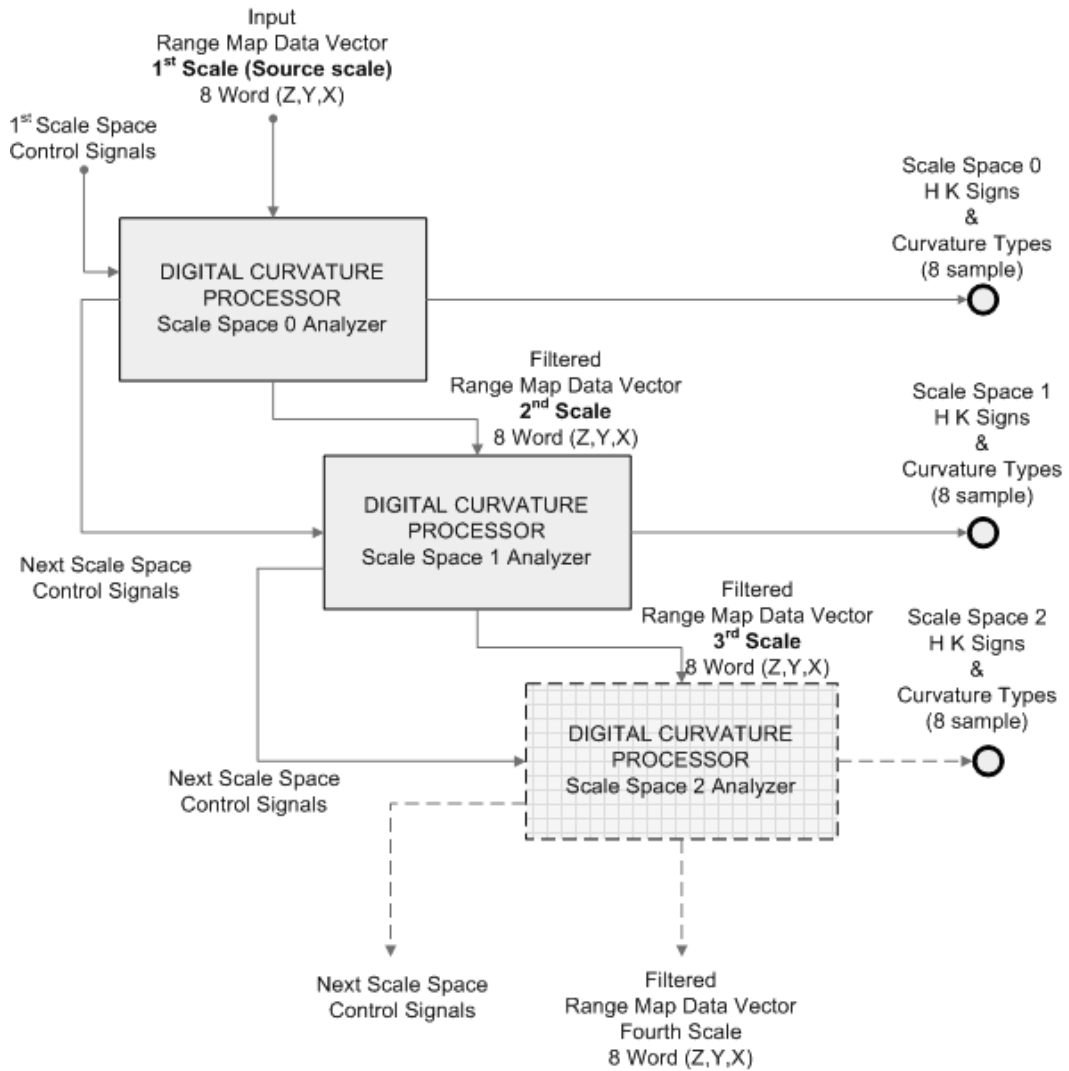


Figure 25 – Pipeline usage of DCP for scale space analysis

Almost all applications using scale space technique, like SIFT, SURF, first generates all scale spaces, and then processes them. Here a different approach has been proposed. Required new signals and new appearance of DCP is shown in Figure 24. Note that, the synchronization between DCP logics is also a new requirement. Synchronization is the same as in range map generator interface.

Example of multiple instance of DCP is shown in Figure 25. Number of DCP is scale space level count, depending on range map size. Each DCP generates its outputs independently with a small phase difference, which will be explained later.

Using only one instance of DCP can also be used to process all scale spaces. This approach requires a RAM Interface and a small controller, provides a multiplexer, run and scale space number signals. The usage example is given in Figure 26.

The reason for using only one DCP may be logic utilization in FPGA chip. If the chip is small, only one instance can be used. For large chips, using multiple DCP is more meaningful. In demo boards, FPGAs are small and therefore one instance is used.

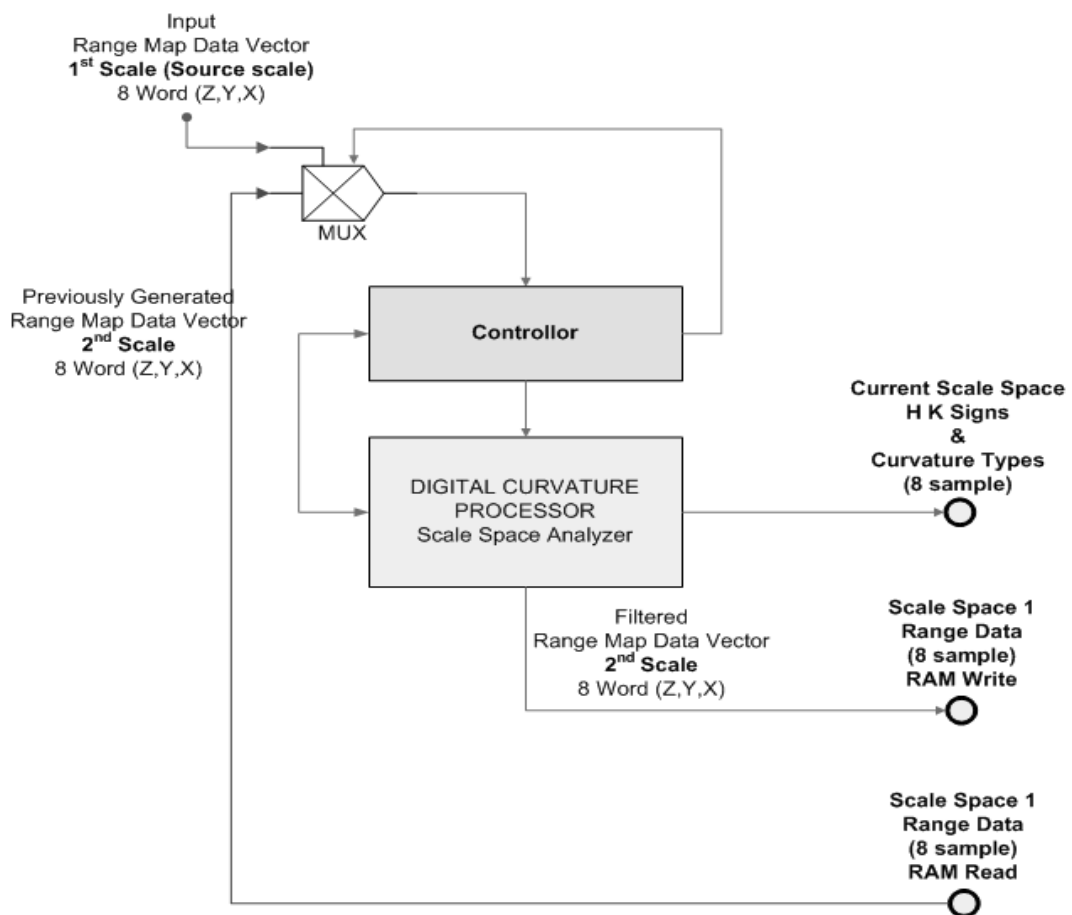


Figure 26 – Single DCP usage for scale space analysis

3.2.6 Gradient Outputs

DCP can also provide gradient outputs, if it is programmed. There are two reasons of serving gradient vectors at output. Gradient vectors can be used for debugging. While developing, seeing values of gradient vectors are very helpful. Another reason is that upper level algorithm can use these vectors whenever it needs.

If the gradient outputs are written into a RAM, the processing takes more time, of course. Also, providing these values at output uses more logics in FPGA. Therefore, two different DCP versions are available, gradients available or not. One more address logic is added into DCP for addressing of output vectors into RAM. Of course these signals are just for RAM controller. Static asserting of these signals converts DCP to stream running mode.

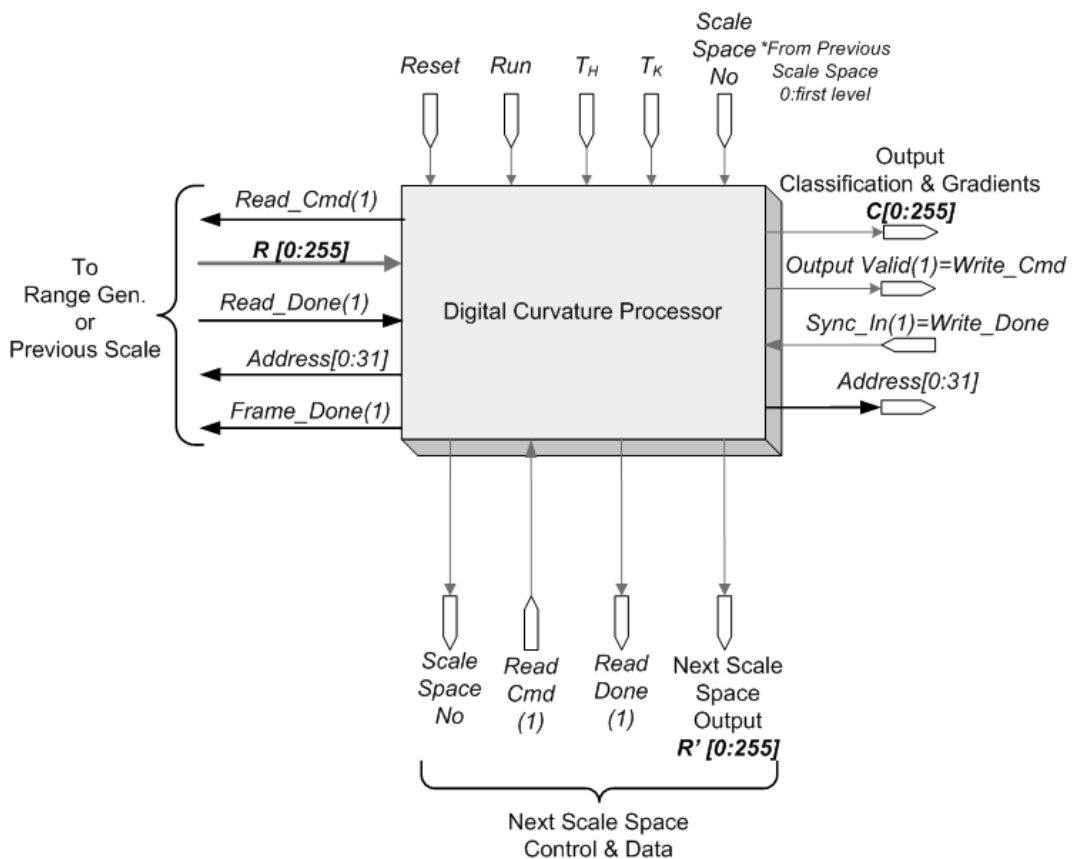


Figure 27 – Gradient output and address decoder supported DCP and IO signals

New address and control signals of outputs are shown in Figure 27. Timing diagrams of the signals are also given in Figure 28. Two different versions of DCP have different signals and timings at outputs. The gradient vector providing DCP is slower.

The overall IO signals of proposed DCP are shown in Figure 27. This logic, or IP Core, is used for analyze surfaces.

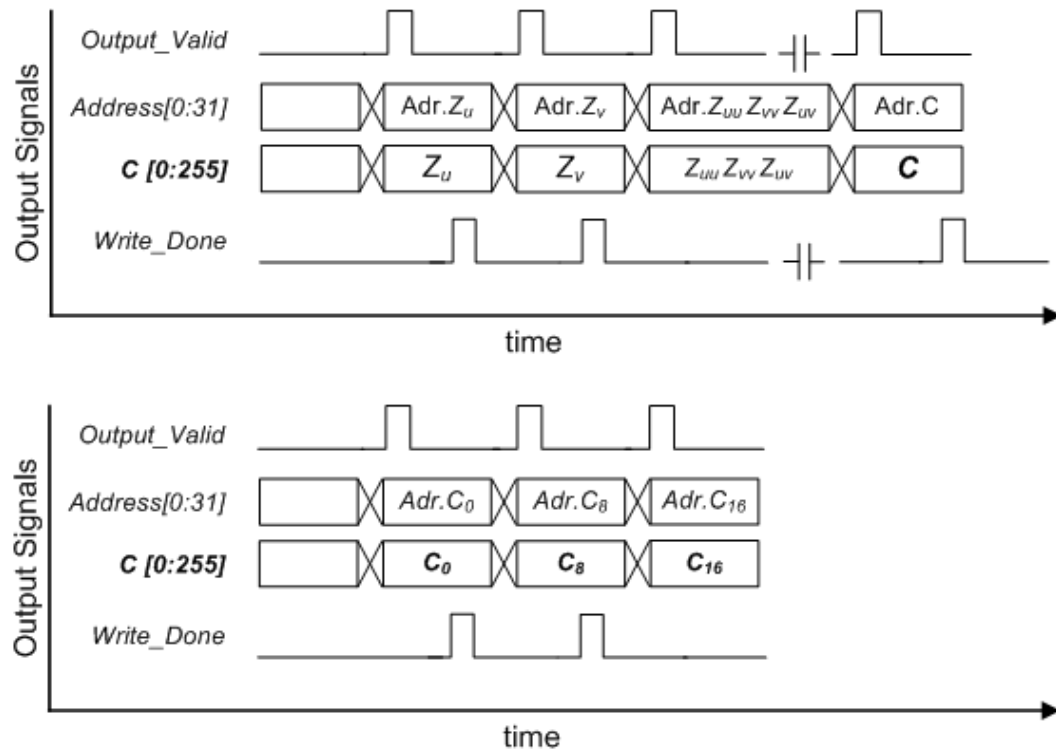


Figure 28 – Output signal flows of two different DCP, gradients available in top design.

3.3 Non-Causality Problem in Range Map Processing

Curvature analysis of a graph surface requires calculation of following quantities, in Eq. 2.11;

$$Z_x \quad Z_{xx} \quad Z_y \quad Z_{yy} \quad Z_{xy}$$

These values are derivatives of a discrete point in range map. Also generating scale space levels requires a 2-D kernel. For a 2D 2x2 kernel, the filter equation;

$$R_{v,u}^{n+1} = R_{2v,2u}^n \cdot k_{0,0} + R_{2v,2u+1}^n \cdot k_{0,1} + R_{2v+1,2u}^n \cdot k_{1,0} + R_{2v+1,2u+1}^n \cdot k_{1,1}$$

n : source scale space no, $n + 1$: next scale space no

for $n = 0$

$$R_{v,u}^1 = R_{2v,2u}^0 \cdot k_{0,0} + R_{2v,2u+1}^0 \cdot k_{0,1} + R_{2v+1,2u}^0 \cdot k_{1,0} + R_{2v+1,2u+1}^0 \cdot k_{1,1}$$

Whichever sample is current input, the filter will always require future sample of range map, because the samples come along \hat{u} direction. Therefore the filter equation is not causal. As shown in Figure 29, if samples come along v direction, we still have non-causal system. In that case u samples will be future samples.

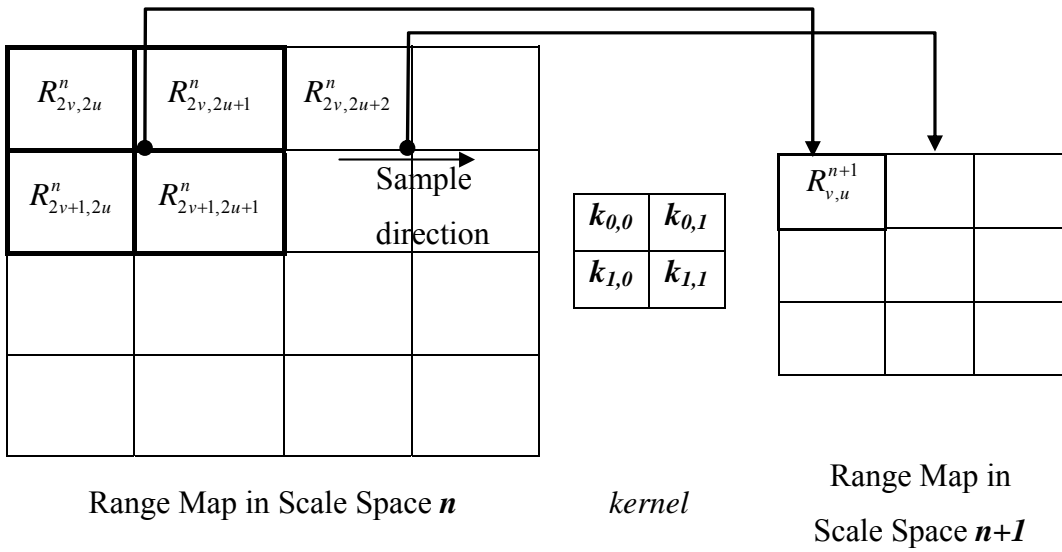


Figure 29 – Scale space kernel.

Derivative of a discrete function, f , at sample n can be approximated as follows;

$$f'[n] = \frac{f[n+1] - f[n-1]}{2}$$

Obviously, derivative functions also require future samples. Therefore they are non-causal.

For scale space generation, using delay lines helps us to deal with the non-causality. The detailed solutions of this problem have been proposed in following sections.

3.4 Digital Curvature Processor Type 1

As mentioned previously, two different digital curvature processors have been developed. In this section, the first type is explained.

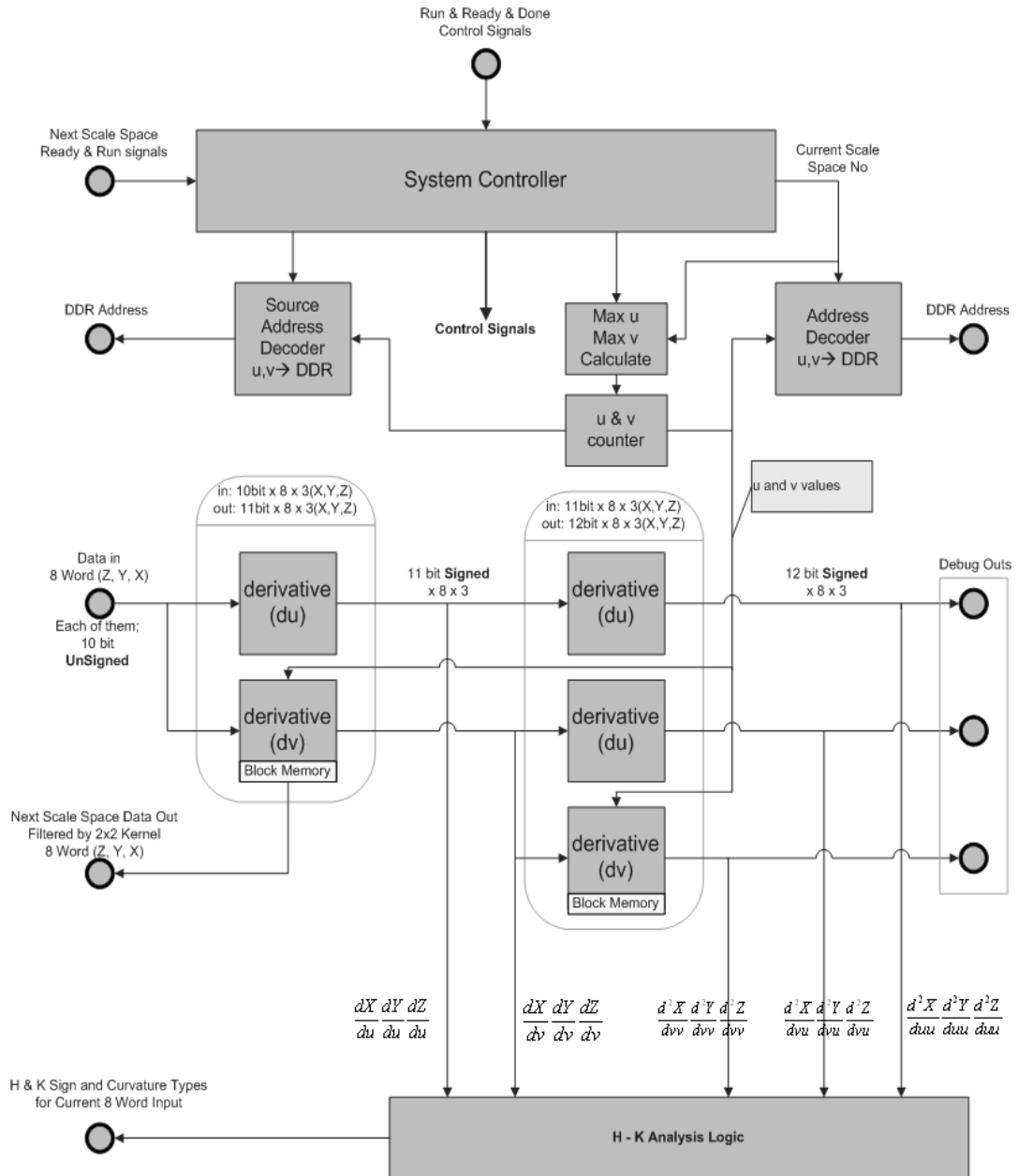


Figure 30 – Logic architecture of DCP type 1

In this DCP, less BRAM and logic utilization is aimed. Its implementation is easier and it requires less logic than DCP type 2. However, its output is less accurate than the other. Also, this DCP has no latency between its input and output.

The logic design of DCP type 1 is shown in Figure 30. The address decoders are for RAM controllers and they are not useful for Range Map generators. System controller checks input signals, run, reset etc... and generates synchronization signals for range generator or next scale space DCP. Debug ports are demonstrated as independent parallel outputs but they are serialized in the logic and served in a single port as already shown in Figure 28.

3.4.1 Derivative Approximations in Hardware

In this type of DCP, the first and second derivatives of a discrete point, say u and v , with respect to x and y , is approximated as follows;

$$\begin{aligned}
 Z_u[v, u] &= Z[v, u] - Z[v, u - 1] \\
 Z_v[v, u] &= Z[v, u] - Z[v - 1, u] \\
 Z_{uu}[v, u] &= Z_u[v, u] - Z_u[v, u - 1] \\
 Z_{vv}[v, u] &= Z_v[v, u] - Z_v[v - 1, u] \\
 Z_{vu}[v, u] &= Z_v[v, u] - Z_v[v, u - 1]
 \end{aligned}
 \tag{Eq. 3.3}$$

Remember, already define $x=u$, $y=v$. This approach is not an exact approximation of derivatives. Indeed, the first derivatives are a noisy sub-pixel derivative approximation. And then, second derivatives are much noisier. However, as seen, causality has been provided in these equations.

Let's visualize the derivatives in hardware;

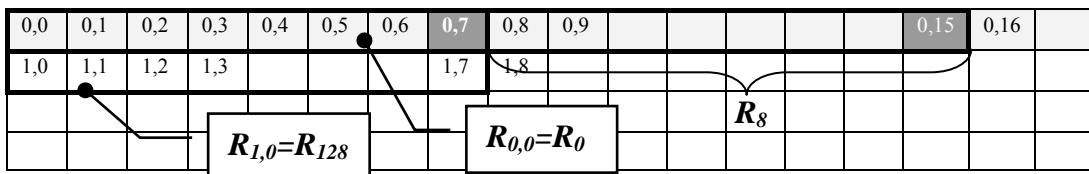


Figure 31 – Input vectors used for derivatives

DCP takes 8 consecutive samples in an \mathbf{R} vector. Therefore d_u calculation is easy. Only requirement in d_u is keeping last sample in \mathbf{R} . Because derivative of flowing vector requires the last sample of previous vector, the dark gray samples. Therefore the \mathbf{Du} logic in Figure 30 has a register to keep least sample. As new vectors coming, first derivative is calculated and then the register is updated. When $u=0$, the register is cleared. The first derivative, the left border, is always equal to input signal. The derivative of 8 samples is calculated at the same time, in parallel. Note that input vector is *unsigned* 10bit and output is *signed* 11bit.

Taking d_v is not easy because the equation need previous samples in the previous row. Therefore a delay line, light gray in Figure 31, is required. Registering, delaying, whole row is not a good approach. The row contains 256x128 bits. Therefore a Block Memory, BRAM, in FPGA is allocated in \mathbf{Dv} logic, as seen in Figure 30. Detailed diagram of \mathbf{Dv} logic is given in Figure 32. When a new vector arrived, the logic first calculates address of BRAM using u value, then reads the vector in the previous row. Under knowledge of current and previous rows, following calculation is made in *Parallel Subtractors*;

$$R_v[v, u] = R[v, u] - R[v-1, u] \quad , R[v-1, u]: \text{read from BRAM}$$

For example, following equation gives $R_v[1,0] = R_v[128]$ as shown in Figure 31;

$$R_v[128] = R[128] - R[0]$$

After the derivative calculated, the logic writes current vector into BRAM. Again input vector is *unsigned* 10bit and output is *signed* 11bit.

\mathbf{Dvy} , \mathbf{Duu} and \mathbf{Dvu} logic are designed as the same manner. The input and output signal lengths are increased by 1 bit.

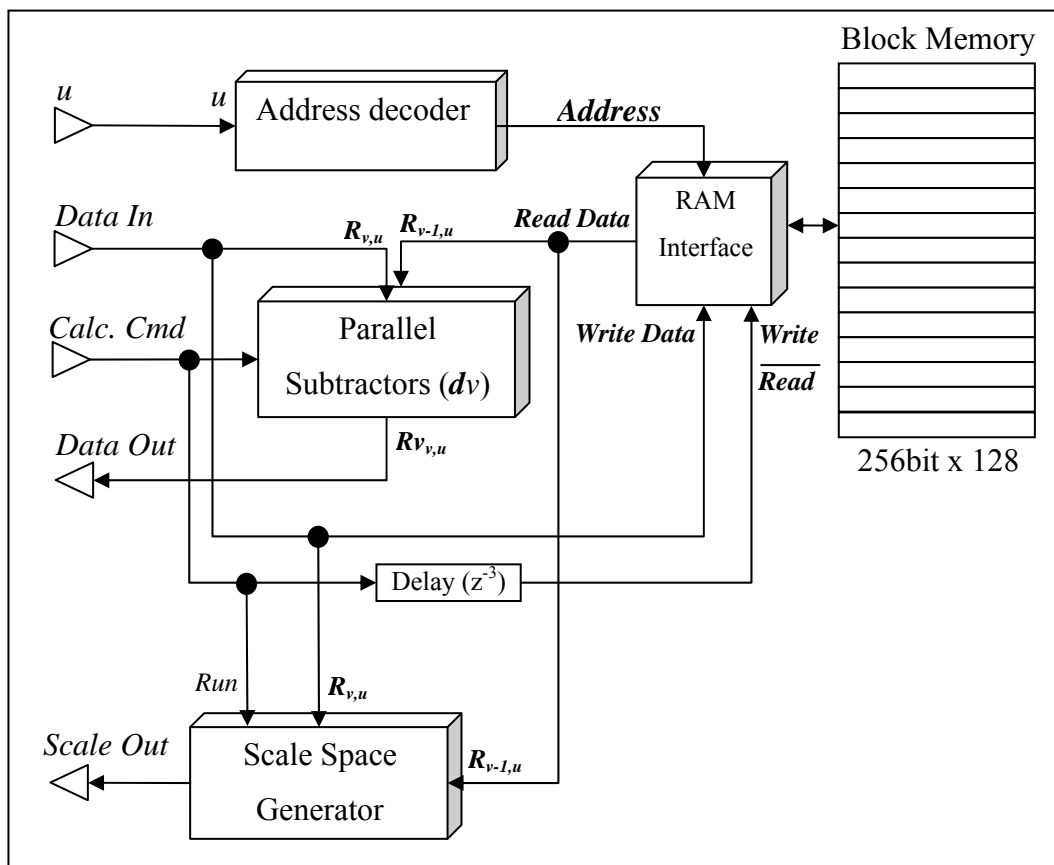


Figure 32 – Dv derivative and scale space block logic diagram

In this design only one row is saved into BRAM. Therefore BRAM utilization is not so much. Also we do not use any dividers or multipliers. We require only 8 subtractors in each of derivative logics.

As already mentioned, one disadvantage of this approach is the approximations cause noisy estimations, derivatives of sub-pixels. Another disadvantage is derivatives of first two rows and columns of range map, i.e. left and top borders, are not calculated.

Scale space calculation also requires the previous rows. Therefore scale space calculation core is embedded into **Dv** logic, as seen in Figure 32.

3.4.2 Hardware Scale Space Technique

DCP scale space algorithm uses a 2x2 kernel and decreases size of the input by half. Graphical view can be seen in Figure 29. The following filter equation is used for scale space generation;

$$R_{v,u}^{n+1} = R_{2v,2u}^n \cdot k_{0,0} + R_{2v,2u+1}^n \cdot k_{0,1} + R_{2v+1,2u}^n \cdot k_{1,0} + R_{2v+1,2u+1}^n \cdot k_{1,1}$$

All of the coefficients, k , is selected 0.25. Multiplication with this number is actually division by 4. An easy and fast implementation of this equation is designed as follows;

Convert the equation to;

$$R_{v,u}^{n+1} = (R_{2v,2u}^n + R_{2v,2u+1}^n + R_{2v+1,2u}^n + R_{2v+1,2u+1}^n) / 4$$

Summation of four 10 bit signals can be maximum 12 bit. The samples of scale space output are unsigned 10 bit. The logic diagram is available in Figure 33.

One vector along \hat{u} direction, contains 8 samples, provides 4 samples of the output. Therefore, the next vector is required to generate 8 samples at output. Second bit of u gives position of current 4 samples at output signal. The selection logic puts 4 samples to first or last 4 signals of output vector according to $u(2)$.

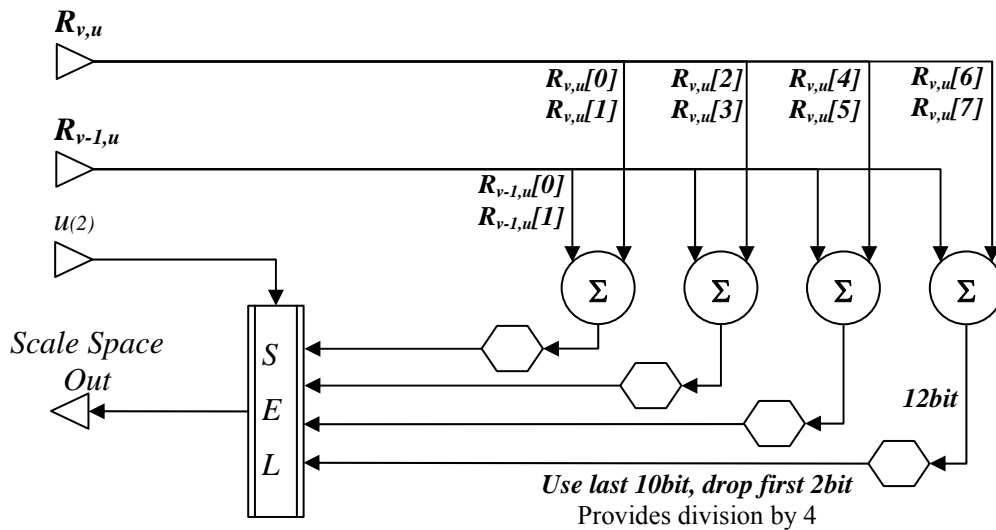


Figure 33 – Scale Space generation logic

Using the signal connection technique, ignoring first 2 bits of 12bit signal, division by 4 is achieved. By this way, no division logic is used. Also one clock cycle, just for summations, is enough to generate output for the next level DCP.

Scale space generator requires previous row vectors. Scale space generator is embedded into **Dv** logic because we already write the same row into BRAM in **Dv**. Using the integration, unnecessary BRAM utilization is avoided.

3.4.3 HK Calculation

After the derivatives are calculated, HK analysis logic generates desired surface classification types using the derivatives, as seen in Figure 30. Indeed the only requirement for curvature calculation is knowledge of partial derivatives, in Eq. 2.11;

$$Z_x \quad Z_{xx} \quad Z_y \quad Z_{yy} \quad Z_{xy}$$

Referring the Table 2, classification depends on only sign of H' and K' values. Remember that using threshold ranges, H is converted to H' and K is converted to K' . The summary of the equations is;

$$\begin{aligned} norm &= (Z_x^2 + Z_y^2 + 1) \\ E &= 1 + Z_x^2 & e &= Z_{xx} \\ F &= Z_x Z_y & f &= Z_{xy} \\ G &= 1 + Z_y^2 & g &= Z_{yy} \\ K &= \frac{e \cdot g - f^2}{norm \cdot (E \cdot G - F^2)} \\ H &= \frac{2 \cdot f \cdot F - e \cdot G - g \cdot E}{2 \cdot (E \cdot G - F^2) \sqrt{norm}} \\ H' &= \begin{cases} 0 & -Th < H < Th \\ sign(H) & otherwise \end{cases} \\ K' &= \begin{cases} 0 & -Tk < K < Tk \\ sign(K) & otherwise \end{cases} \end{aligned}$$

Using fixed-point numbers, here, causes a disadvantage. Unlike floating number representation, multiplying the fixed-point numbers increases output bit length. Length of the output is sum of bit length of multiplied numbers, without sign bit. In the formulas, both H and K calculation includes a division. If we

design divider logic, we utilize too much logic cells in FPGA due to too much divisor and dividend bit length. Also, providing a fast division speed is not easy. On the other hand, if we can use embedded DSP cells, including hard multipliers, in the FPGA, we can achieve more performance and we can utilize less logic components. For this purpose we should revise the formulas. What we need is, actually, sign of H and K . Considering Eq. 3.1 and Eq. 3.2, following formulas are used to get the signs;

$$\begin{aligned}
 K &= \frac{e \cdot g - f^2}{norm \cdot (E \cdot G - F^2)} \\
 K \cdot norm \cdot (E \cdot G - F^2) &= e \cdot g - f^2 \\
 Bk &= norm \cdot (E \cdot G - F^2) \\
 Pk &= e \cdot g - f^2 \\
 K' &= \begin{cases} 0 & (-Tk) \cdot |Bk| < Pk < Tk \cdot |Bk| \\ signum(Pk) \cdot signum(Bk) & otherwise \end{cases} \quad (Eq.3.4)
 \end{aligned}$$

$$\begin{aligned}
 H &= \frac{2 \cdot f \cdot F - e \cdot G - g \cdot E}{2 \cdot (E \cdot G - F^2) \sqrt{norm}} \\
 H \cdot 2 \cdot (E \cdot G - F^2) \sqrt{norm} &= 2 \cdot f \cdot F - e \cdot G - g \cdot E \\
 Bh &= 2 \cdot (E \cdot G - F^2) \sqrt{norm} \\
 Ph &= 2 \cdot f \cdot F - e \cdot G - g \cdot E \\
 H' &= \begin{cases} 0 & (-Th) \cdot |Bh| < Ph < Th \cdot |Bh| \\ signum(Ph) \cdot signum(Bh) & otherwise \end{cases} \quad (Eq.3.5)
 \end{aligned}$$

These equations are proper for hardware implementation. HK logic first generates e , g , f , E , G , F and $norm$ variables using DSP48 logics in FPGA according to fixed-point length and signs as seen in Figure 34. Then, pipelined architecture calculates other variables, defined the formula above.

Table 3 – Sign representation in 2's complement form

Sign	Number (2 bit)
0	00
+	10
-	11

In last step, signs are extracted. Signs are 2 bit numbers and they are given in Table 3. 2 bit signed number in 2-complemet format is used and, in 2-complement format, -1 is '11'.

In this design approach, H and K values are never calculated. Only signs are extracted and this can be seen as a disadvantage.

Thresholds are directly related with H and K values. To determine the required thresholds, we should know these H and K values. For this purpose, using a special range map, including test pattern, we can extract H and K map on computer software. We test and simulate all equations, considering fixed-point representation, in MatLab software. In MatLab, generating H and K maps are possible. By help of the simulations, the threshold values can be easily determined on the maps.

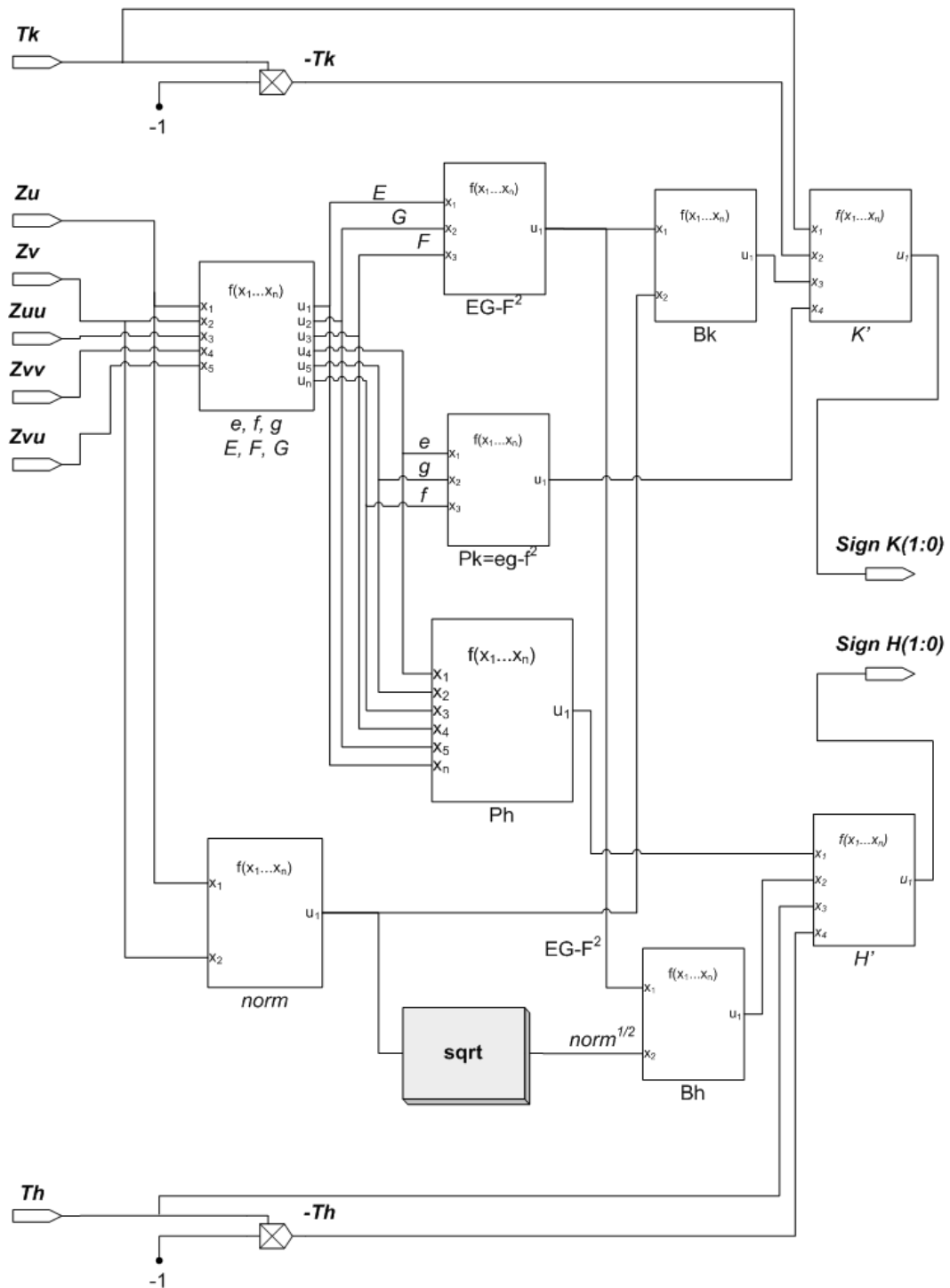


Figure 34 – H and K sign extraction logic architecture.

The HK logic core utilize significant amount of DSP48 and logic cells in the FPGA. Beside of processing 8 samples in parallel, we may also use just one instance of this logic, and we can process the vector input sequentially using a

serializer and de-serializer, as seen in Figure 35. This approach is proper when less logic utilization is required. On the other hand, speed decreases significantly, 8 of the previous case.

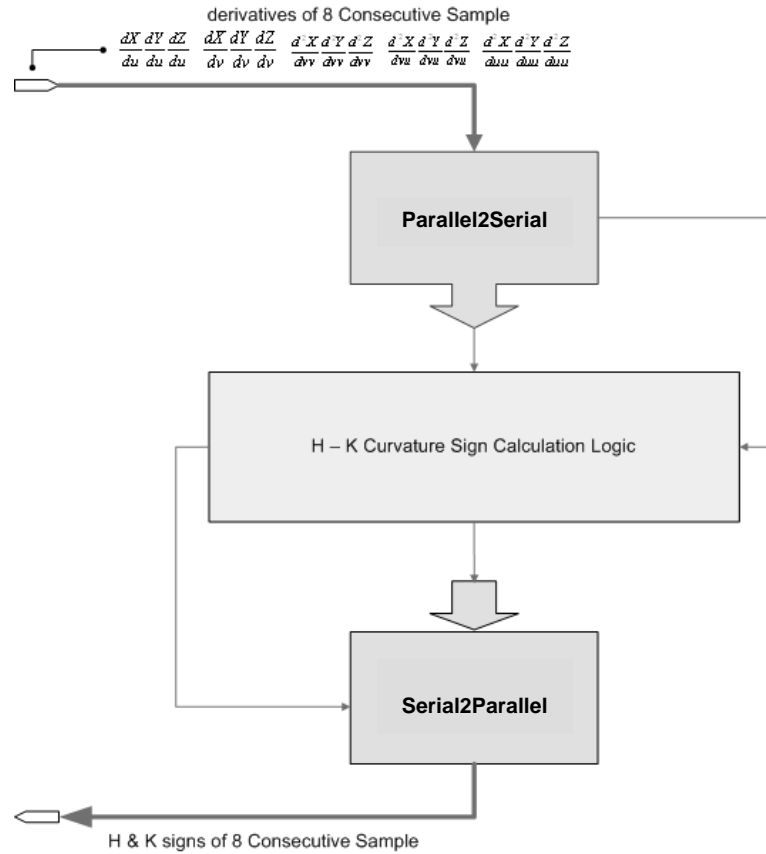


Figure 35 – One instance usage example of HK Logic.

3.5 Digital Curvature Processor Type 2

Second type of digital curvature processors, basically, deals with the problem of noisy derivative estimation. Here, another approach has been proposed for calculating the derivatives.

In this DCP, more BRAM and logic elements are required. On the other hand, the output is more accurate than output of DCP type 1.

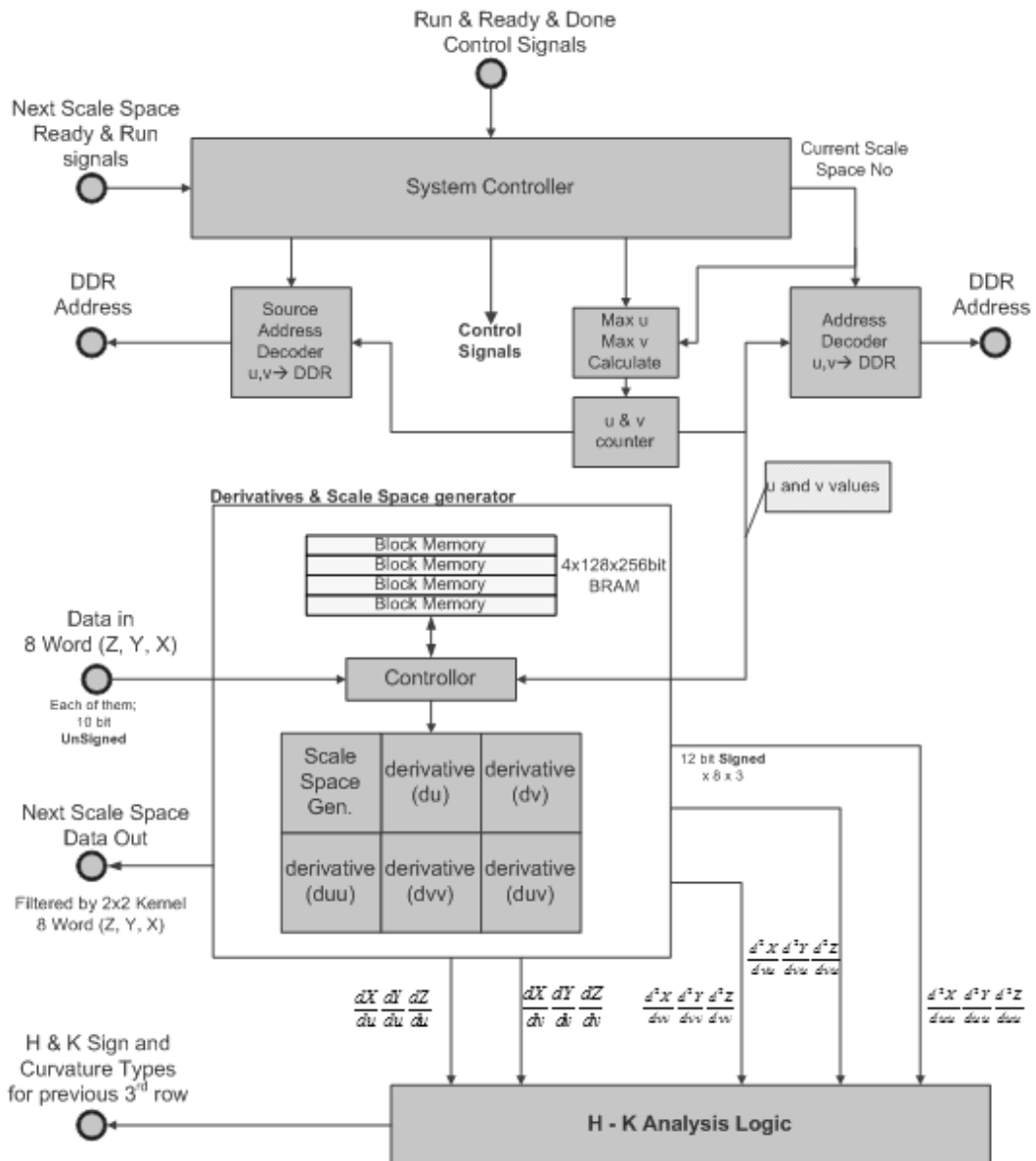


Figure 36 - Logic architecture of DCP type 2

The block diagram of DCP type 2 is shown in Figure 36. The differences are in derivative calculations and in scale space generation logics. Here, a controller reads and writes to on-chip memories and it provides required signals for derivative and scale space generation.

3.5.1 Gradient Approximations in Hardware

Following discrete derivative approximations are used in this DCP;

$$Z_u[v, u] = rnd\left(\frac{Z[v, u+1] - Z[v, u-1]}{2}\right) \quad (\text{Eq.3.6})$$

$$Z_v[v, u] = rnd\left(\frac{Z[v+1, u] - Z[v-1, u]}{2}\right)$$

$$Z_{uu}[v, u] = rnd\left(\frac{Z[v, u+2] - 2 \cdot Z[v, u] - Z[v, u-2]}{4}\right)$$

$$Z_{vv}[v, u] = rnd\left(\frac{Z[v+2, u] - 2 \cdot Z[v, u] - Z[v-2, u]}{4}\right)$$

$$Z_{vu}[v, u] = rnd\left(\frac{Z[v+1, u+1] - Z[v+1, u-1] - Z[v-1, u+1] + Z[v-1, u-1]}{4}\right)$$

$rnd()$ = rounding operation in fix point space

Obviously, to calculate current first and second derivatives, some samples in 2 rows later are required. Again, delay lines using registers are not proper for huge amount of data. Therefore we utilize some on-chip memory in the FPGA. To visualize situation, we may follow Figure 37. Required samples for derivatives of $R_{2,8}$ and $R_{2,15}$ are dark gray samples. In this application, the derivatives calculated belong to 2 rows before from current input R . Therefore a delay, about 2 rows, exists in this technique. Also, there is a latency of 5 rows. In other words, to get a valid output, first 4 rows must be stored and data of 5th row should be come. Another disadvantage is invalid approximations are generated for first and last two rows and columns.

0,0	0,1	0,2	0,3				0,7							0,15	0,16	
1,0	1,1	1,2	1,3													
2,0								2,8						2,15		
3,0																
4,0																

← Current Range Data Vector R →

Figure 37 – Required samples to calculate derivatives in DCP2

One advantage of this approach is accuracy of derivatives. They are reliable than the previous application. Also, note that, we generate the derivatives in a few clock and second derivatives are not dependent to first derivatives as in the previous approach, shown in Figure 30. Therefore, without pipelining, we generate derivatives at the same moment.

Another important point, here, is rounding operation in calculations. Discarding of beginning bits to divide the numbers by some powers of 2 causes floor operation in fixed-point number space. For example; 3.2 unsigned fix point number say '101'b is 5 in decimal and 1.25 in fixed point space. Let's divide it by 2;

$$\frac{1.25}{2} = \frac{'101'b}{2} = 010\lfloor 1 \rfloor_{drop} = '010'b = 0.5$$

The division is 0.5 but actual division is 0.75. Clearly bit dropping causes a floor operation. If we design the logic considers dropped bit;

$$\frac{1.25}{2} = \frac{'101'b}{2} + 00\lfloor FIRST BIT \rfloor = 010+'00'1 = '010'b = '001'b = '011'b = 0.75$$

Now, the output is more accurate, but a summer logic is required and the output has one clock latency. The rounding has a drastic effect at classification process. We have provided the outputs without rounding logic at “Rounding Operator Effect in DCP2” in “RESULTS & COMPARISONS” chapter.

3.5.2 Hardware Scale Space Technique

The scale space equation in this design is the same as in the previous design. The required samples are shown in Figure 37 at first 2x2 cells as heavy dark gray.

In this design, we do not integrate **Dv** and scale space logics. We directly generate scale space data and the controller provides required samples by reading from on-chip memory. Unlike derivative generation, there is no delay in scale space calculation, but one row latency exists.

3.5.3 HK Calculation

The HK calculation logic is the same in DCP type 1.

3.6 Surface Type Classification

Unlike software approach, in hardware, we do not need any computation to put classification labels, available in Table 2, to surface points. Defining 2 bit sign signals for H and K help us to extract surface type. Again, considering 32 bit architecture for upper level system, we use the following word notation for surface classification output, C ;

Empty	Empty	Sign H	Empty	Sign K
0xXXXX	00b	2bit	00b	2bit
2 Bytes	1 Byte		1 Byte	
32 bit, 4 bytes (Big Endian)				

As seen, first 2 bytes defines the classification and last bytes have no meaning in this context, written as X. Later we will use the last two bytes for different purpose. According to these definitions, the meaning of the output, C , is shown in Table 4.

Table 4 – DCP Classification outputs

Sign of H	Sign of K	C (32bit)	Local (around the point) Surface Shape
0 (00)	0 (00)	0xXX0000	Plane
+ (01)	0 (00)	0xXX0100	Concave Cylindrical (Valley)
- (11)	0 (00)	0xXX1100	Convex Cylindrical (Ridge)
+ (01)	+ (01)	0xXX0101	Concave Elliptical (Pit)
- (11)	+ (01)	0xXX1101	Convex Elliptical (Peak)
0 (00)	- (11)	0xXX0011	Hyperbolic
- (11)	- (11)	0xXX1111	Hyperbolic
+ (01)	- (11)	0xXX0111	Hyperbolic

3.7 Fast Square Root in Hardware

As seen in HK analysis logics, a square root operation is required. In hardware, calculating square root is still a popular topic. In this project, we need fixed-point square root logic.

Available square root algorithms are basically two types; iterative calculations and polynomial fitting approximations techniques.

Many FPGA synthesizers provide CORDIC based square root calculation logic. However, CORDIC is an iterative computation and it is not so fast [29]. Any other iterative approaches, like Newton-Raphson method, can also be considered as a slow method.

Polynomial fitting based solutions are also available. These kinds of solutions require many multiplier and adders, which abolish low resource requirements. To deal with this problem, multiply and sum method can be used, but this converts the algorithm to an iterative solution.

Look-Up table based solutions are faster but if source number space is large, this approach becomes unpractical in today's FPGA. Also, well known linear square root approximation formula, used in [30], is applicable only for small numbers.

Linear approximation of nonlinear functions allows us to utilize less hardware resources. There are some works on literature [31]. Lachowicz and Pfliederer have proposed linear approximation method (first order polynomial) to evaluate square root function in hardware. Their implementation mainly deals with speed. That implementation calculates square root in 1 clock cycle. They used variable step look up tables to construct linear approximation for minimizing error. Bajger and Omondi also have worked on square root function [32]. They consider relative error using piecewise linear approximation.

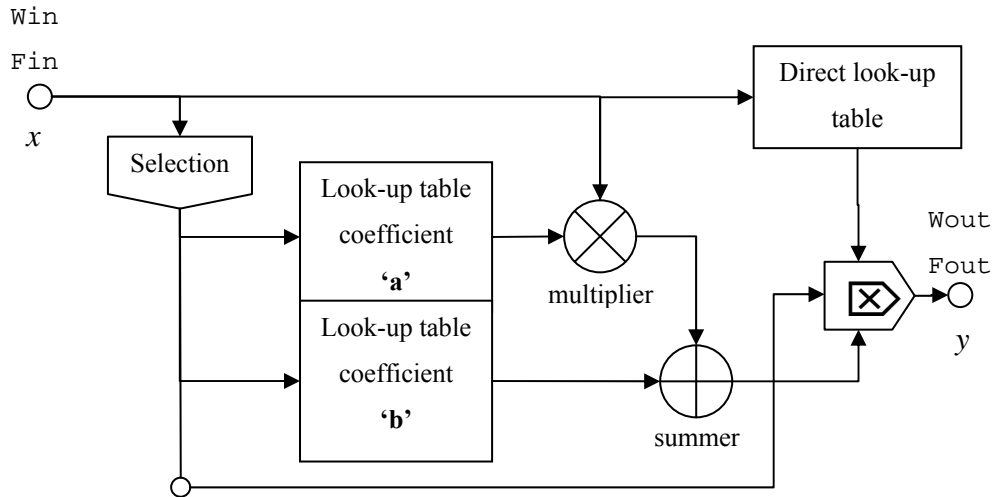


Figure 38 – Purposed square root logic

In this thesis, a special least square linear approximation method is used to calculate square root of a number. The nonlinear square root function is divided into a few non-equal length intervals. The intervals are called as segments and the boundaries of the segments are called as nodes. Linear fitting coefficients are changed at each node point to minimize error in each nonlinear interval. Position of nodes depends on probability distribution of the input of square root logic. In HK logic design, we prefer to use this technique. This approach can be called as a piecewise linear discontinued fitting in defined intervals of square root line. Discontinuities occur at node points due to changing of the fitting coefficients, in Eq. 3.7. The plot is shown in Figure 39 and the logic diagram is shown in Figure 38. Only one multiplier and summer is required for linear fitting and output latency is three clocks. Also, a look-up table is added for high accuracy required segments in fitting process.

In this circuit, the square root function is approximated by;

$$f_{(x)} = y = a_{i(x)}x + b_{i(x)} \quad i_{(x)} : \text{segment no} \quad (\text{Eq.3.7})$$

An example of discontinued approximation is shown in Figure 39. The discontinuities are not problem in HK calculation, because ' $norm^{1/2}$ ' is multiplied with a number and there is no summation in the formula, Eq.3.5.

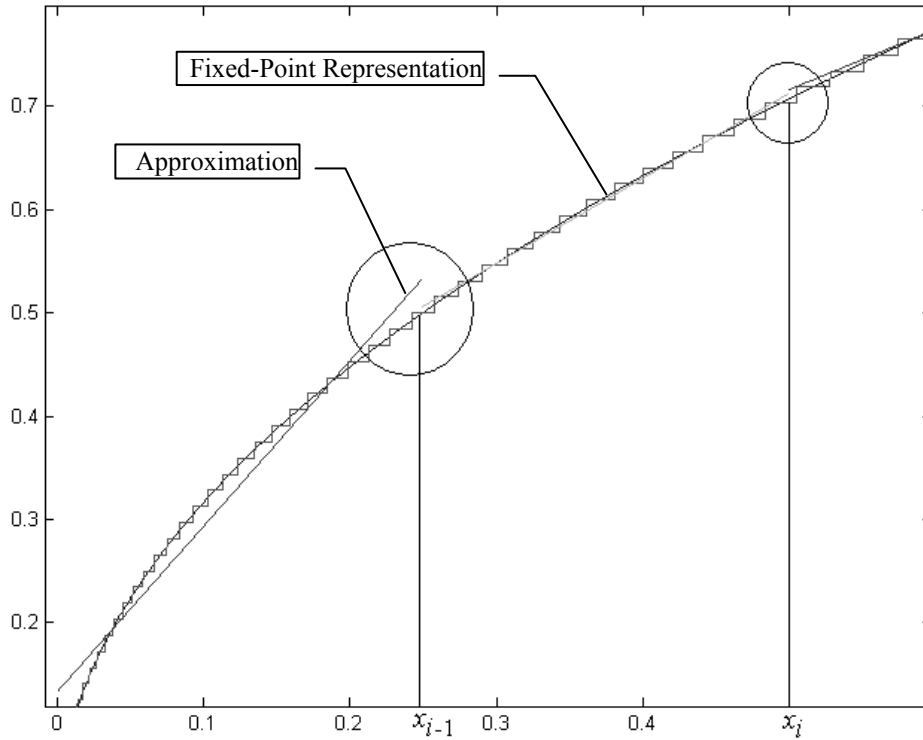


Figure 39 – Square root line and its discontinuous approximation

Remember ‘norm’ variable;

$$norm = (Z_x^2 + Z_y^2 + 1)$$

$$\sqrt{norm} = \sqrt{(Z_x^2 + Z_y^2 + 1)}$$

The input, *norm*, is a fixed point number, word length is 22 and fraction length is 6. The square root output is again fixed point, word length is 11 and fraction length is 3.

The probabilistic model is extracted by using simulations in MatLab software. A histogramming is performed on enough real range maps and the distribution is extracted. Range of 22.6 unsigned fixed-point data is in between [0...65535.984375]. In this range, the input distribution is shown in Figure 40. As seen, distribution is populated around 500, and the most likely value is 1. This is an expected result because for a planar surface *norm* stays around near 1. First derivatives are about zero for a flat surface. Also we have nearly no data after 8000. Smoothing of range map data makes smaller the first derivatives. Also, note that *norm* can not be less than 1.

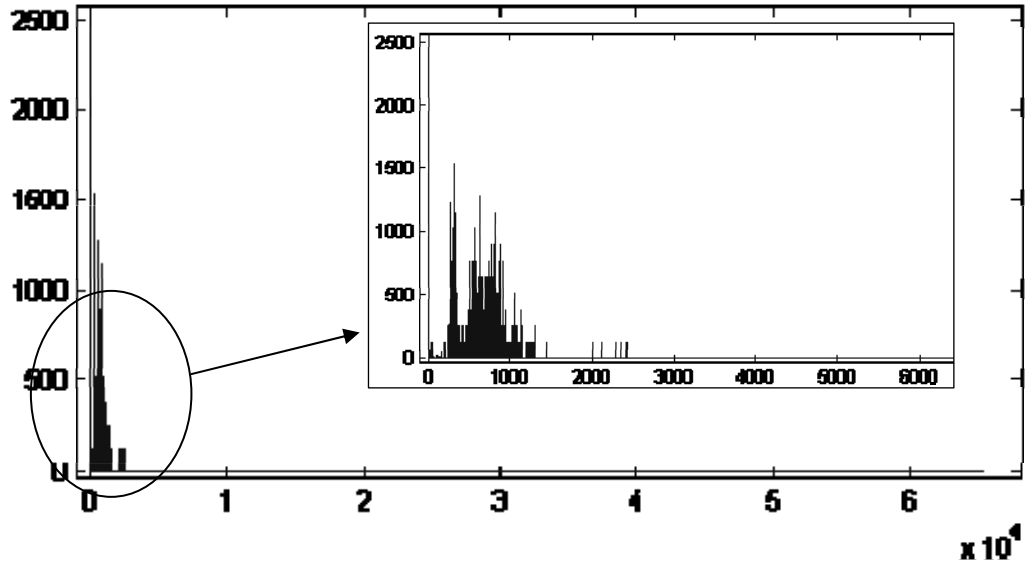


Figure 40 – Input distribution of square root function, Wl=22, Fl=6

According to the distribution, we should divide the range [1...1000] more segments. With considering distribution and linearity of square root function, we choose 22 node points for linear fitting in square root logic, shown in Table 6. Also, plot of node points on square root line is shown in Figure 41. Note that, in first segment, i.e. between [0...64), the error is very high. This segment is very important, because most of the first derivative values are small and so “norm” is small, too. Consider that surfaces in range maps are mostly planes. A high error in this range makes threshold limitation useless and causes misclassification. Therefore, this segment is written into a look-up table, ROM, as seen in Figure 38. When the input is in this range, the output is taken from the LUT. Reason of keeping range of this segment is utilizing less on-chip memory of FPGA.

Using fixed-point number causes a quantization error in representation. In Table 5, the quantization error is given for output, 0.0313 with respect to float-64 format. Also, approximation coefficients, a_i and b_i , are kept in a look up table in fixed point notation and their word and fraction length affect the output accuracy. In Table 5, there are measurement of mean error for different selection of word length and fraction length of the coefficients. According to the table, 25.18 is the most proper selection. Therefore we use this word and fraction length for coefficients in square root logic.

Table 5 – Effect of coefficient quantization

	Fixed-Point (Wl=10, Fl=3)	Coefficients Wl=27, Fl=20	Coefficients Wl=25, Fl=18	Coefficients Wl=22, Fl=15	Coefficients Wl=20, Fl=13
Mean Error	0.0313	0.0742	0.0855	0.2569	0.9807

Table 6 – Selected node points for approximation in hardware.

x	y
0	0
64	8
128	11.375
192	13.875
256	16
384	19.625
512	22.625
1024	32
2048	45.25
4096	64
6144	78.375
8192	90.5
12288	110.875
16384	128
20480	143.125
24576	156.75
32768	181
38912	197.25
45056	212.25
50176	224
55296	235.125
65535.9844	255.875

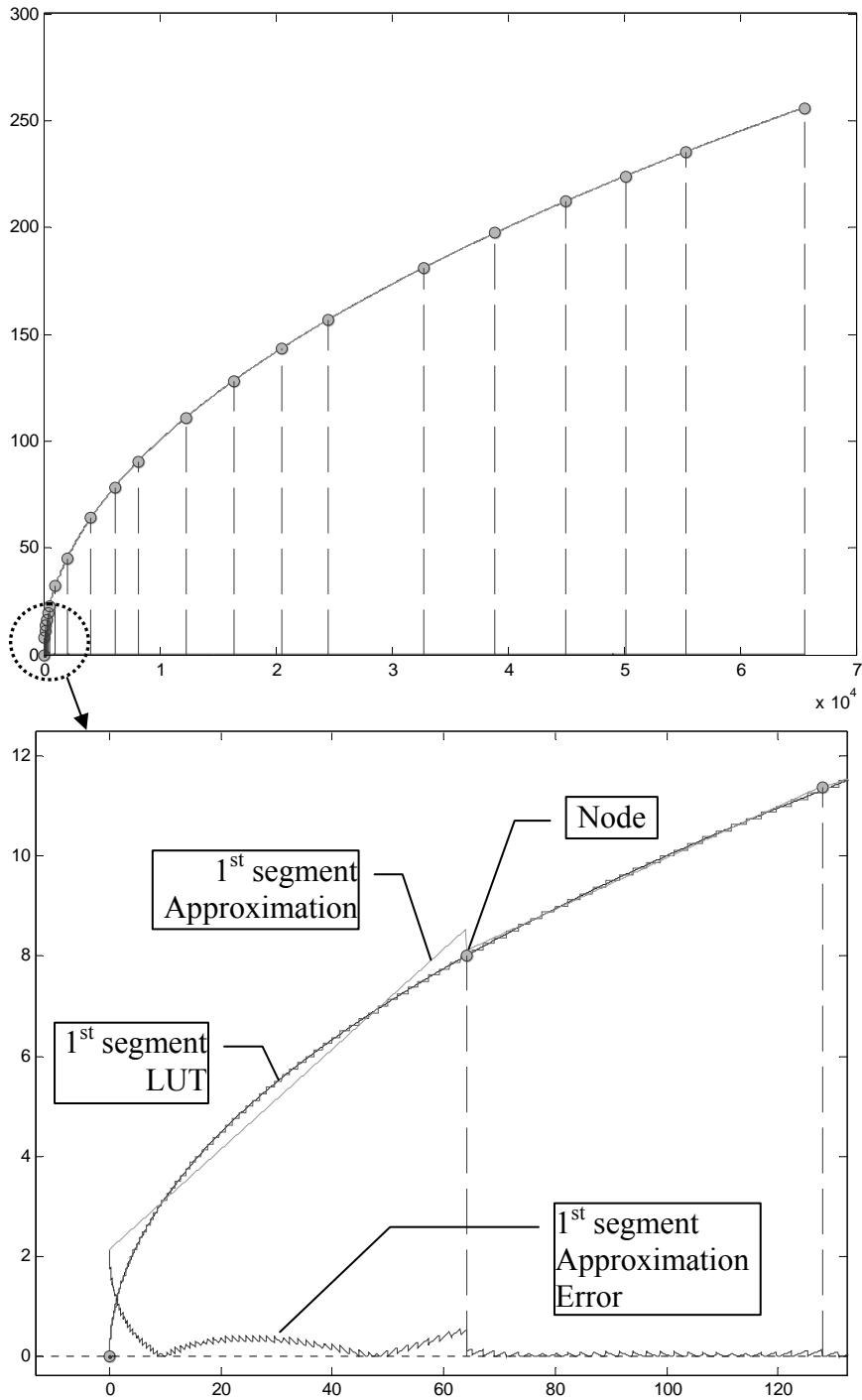


Figure 41 – Overall approximation and error in the first interval.

3.8 Thresholds

As already mentioned, we define thresholds for H and K values, Th and Tk respectively to deal with surface irregularities.

We determine the required Th and Tk by considering H and K values at desired point. An artificial surface and its H map are given in Figure 42. For the same points on a cylindrical surface, where $K=0$, in both plots, the H values are $H_1= 0.002695$ and $H_2=0.02462$. As we know, H represents concavity of the surface. A small threshold, i.e. less or equal to H_1 , provides more details about concavity but it may cause a noisy map. A large threshold, for example around H_2 , suppresses elliptical surface classification, as shown in Figure 42.

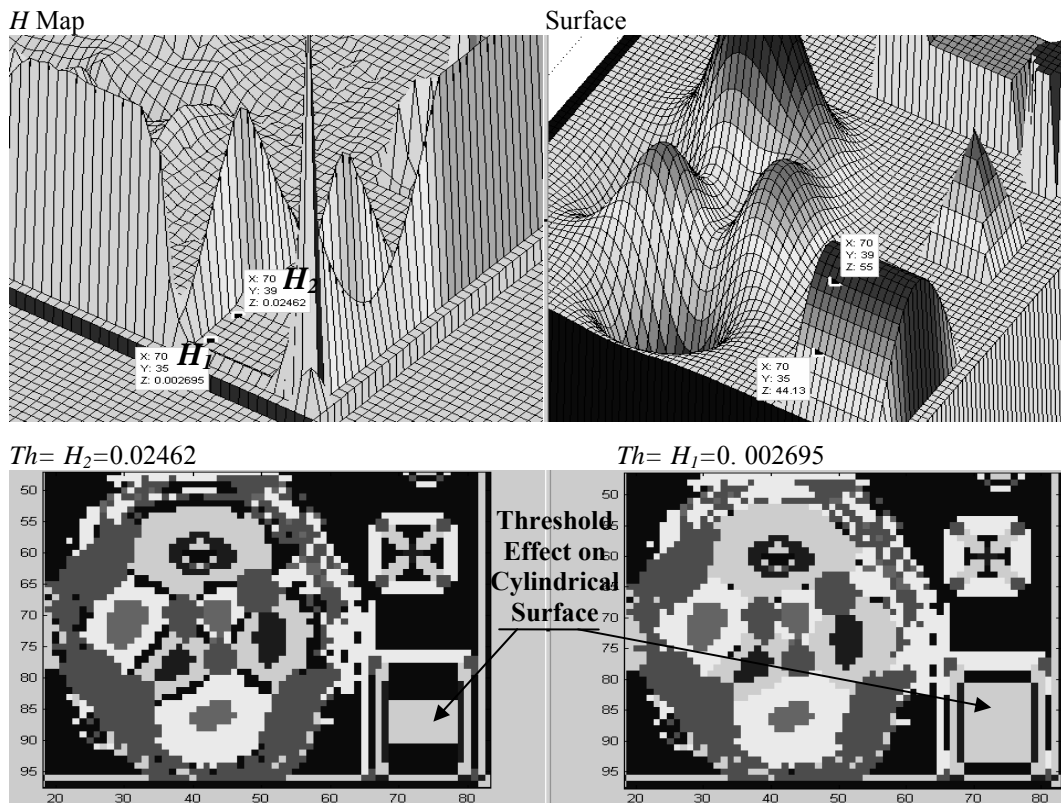


Figure 42 – H and K values and threshold effect

Resolution of Th and Tk are also important. Increasing bit length causes more logic utilization, but provides more accuracy in classification process. Bit length of Th and Tk is determined in MatLab simulations. After extracting of H

and K maps in simulation for real and artificial range maps, constructed by 10.3 fixed point numbers, we conclude that 11 bit fraction length is pretty enough for application. In DCP1, we aim to use low resources and DSP48, so threshold word length is 8 bit and fraction length is 6 bit, unsigned. The range and resolution of thresholds in DCP1;

$$resolution = \frac{1}{2^6} = 0.015625$$

$$Wl = 8 \quad Fl = 6 \quad \Rightarrow range : [0...3.984375]$$

In DCP2, thresholds are more sensitive, word length is 12 bit and fraction length is 11 bit, unsigned. The range and resolution of thresholds in DCP2;

$$resolution = \frac{1}{2^{11}} = 0.00048828125$$

$$Wl = 12 \quad Fl = 11 \quad \Rightarrow range : [0...3.9990234375]$$

CHAPTER 4

EMBEDDED COMPUTER & SOFTWARES

In this chapter, it is aimed to explain the designed small computer to control DCP logic. Behind controlling of DCP, the computer provides an important interface between DCP and PC. A personal computer helps us to see and interpret the outputs. Possibility of putting some processing software, for example *a connected component algorithm*, to the embedded computer is another advantage.

4.1 Computer Architecture

XILINX microprocessor, MicroBlaze, is one of the powerful soft-processor in FPGA world. It is a 32bit, big-endian, Harvard type, RISC processor. In this SoC design we use this processor. The system bus is IBM PLB bus, provided by XILINX. Using this bus, the CPU reaches the peripherals, its instructions and data sections.

In Figure 43, the system architecture is shown. The CPU has a bootloader in FPGA BRAM, initialized by FPGA bit stream. Required clocks are generated using a Digital Clock Manager. The CPU has a debug interface via JTAG connection. Another control interface is UART, 115200bps and connected to the system bus. We control the system via this UART module. Another UART is integrated in MDM JTAG module. We also use this UART as secondary control interface.

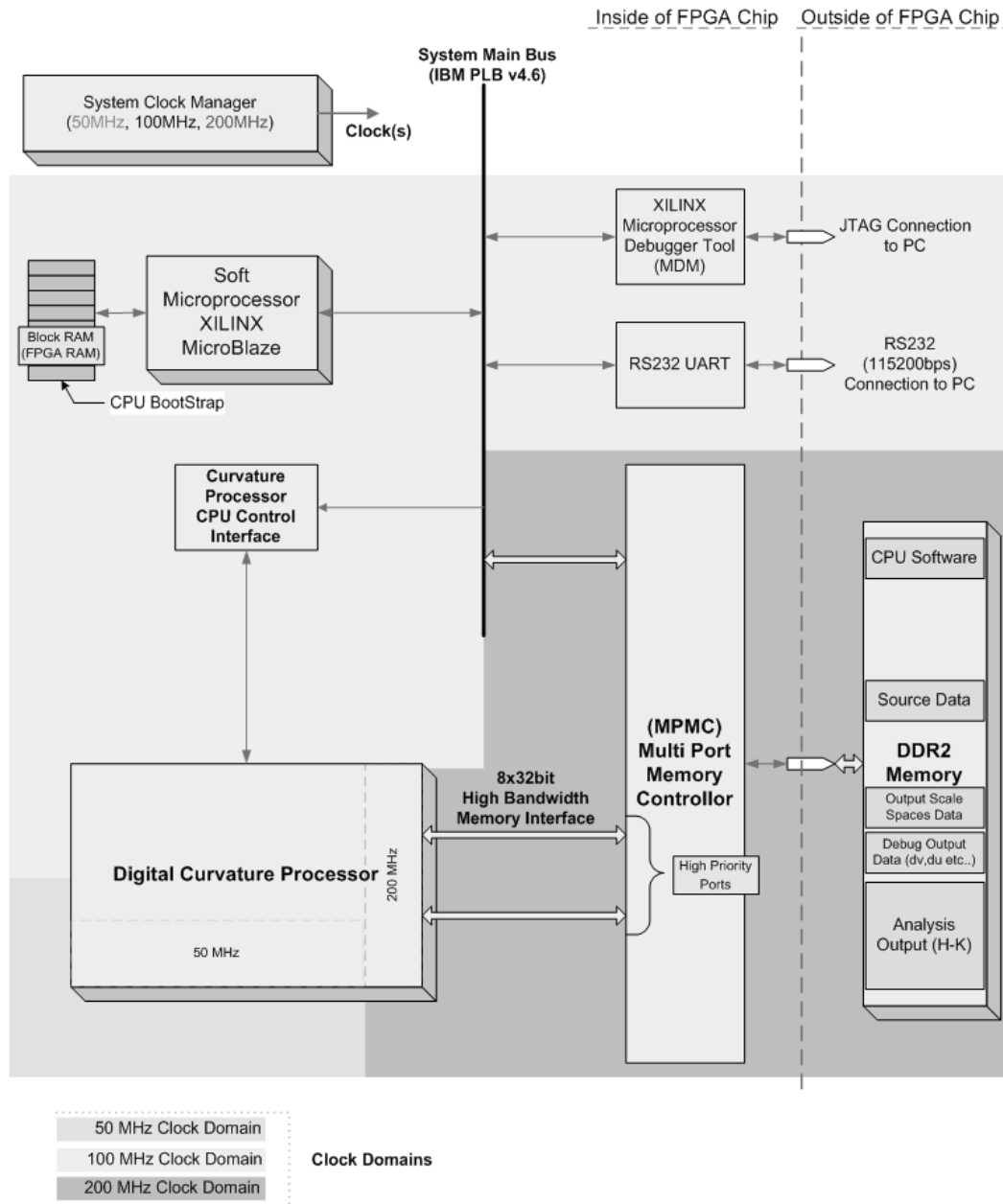


Figure 43 – Top level FPGA design

The curvature processor has no interface proper to IBM PLB architecture. Therefore, a PLB to DCP interface is required. This interface has an address map on the system bus and converts CPU signals to DCP parameters, such as run, thresholds etc... Also, it has a timer that measures the processing speed of DCP.

Another port of DCP is High Speed Memory Interfaces. We already design two addressing logics, for source and destination, in the DCP for RAM

controllers. We use them for DDR Memory in the computer design. The interface between DCP outputs and MPMC, XILINX multi port memory controller, not shown in the figure, provides reading and writing range map, scale space, derivatives and output data.

The missing part of this design is range map generator. Without range map generator, we can use DCP by help of this embedded computer design. We send a synthetic or real range data via embedded processor and we can download the DCP outputs into the PC. DCP can work alone, without CPU, if a range map flow is provided, such as using a range map generator in the same chip or via a high speed interface, 1Gbit Ethernet, PCI-E etc...

4.2 Software

Memory map is shown in Figure 43. Main software and source range data is in DDR memory. Also the scale space levels and outputs of each scale spaces are written into DDR by DCP via MPMC.

A control software is integrated into the embedded system. This software provides, memory read, memory write, memory download to PC. It also provides Threshold controls, run, reset and scale space count settings of DCP via UART modules.

4.2.1 Connected Component and Center of Mass Algorithm

Another software in the system is connected component algorithm. A basic connected component analysis is developed and integrated into the software. It gives labels to surface points. The labels are assigned to last two bytes of C vectors, as seen in Figure 44.

The connected component algorithm is given in Flow 1. The algorithm searches on Width*Height, area of scale space 0 surface type output, to find unlabeled pixels. If an unlabeled pixel is found, the 3D scale space connected component search is called. This function, firstly, learns surface type of sender

pixel. Then, it finds the same type of 3D neighbors of the sender pixel and calls itself recursively for these pixels. While putting the same label the connected pixels, also the area, the center x and y points are also calculated. Lastly, center scale, which can be a fractional number, is calculated by help of determined areas in each scale space.

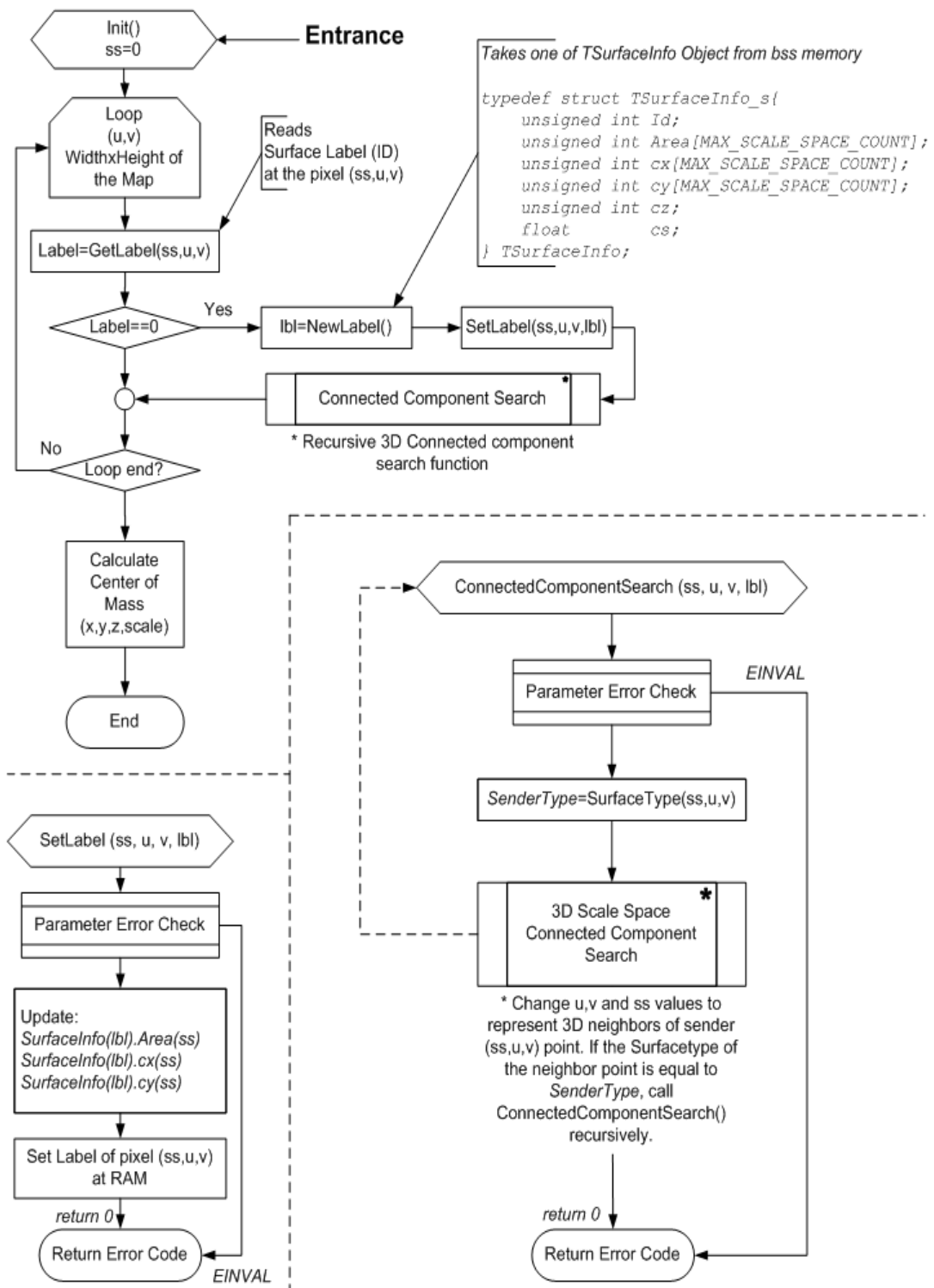
Labels	Empty	Sign <i>H</i>	Empty	Sign <i>K</i>
0xXXXX	00b	2bit	00b	2bit
2 Bytes	1 Byte		1 Byte	
32 bit, 4 bytes (Big Endian)				

Figure 44 – Connected component label assignment

A demonstration is given in Figure 45. The numbers in the cells are surface classification types, available in Table 4, and cell colors represent groups, connected cells.

4	4	3	0	0	<table border="1"> <tr><td></td><td>Surface 1</td></tr> <tr><td></td><td>Surface 2</td></tr> <tr><td></td><td>Surface 3</td></tr> <tr><td></td><td>Surface 4</td></tr> <tr><td></td><td>Surface 5</td></tr> <tr><td></td><td>Surface 6</td></tr> </table>		Surface 1		Surface 2		Surface 3		Surface 4		Surface 5		Surface 6
	Surface 1																
	Surface 2																
	Surface 3																
	Surface 4																
	Surface 5																
	Surface 6																
4	0	3	0	4													
0	0	0	4	4													
0	4	4	4	3													
0	4	3	3	3													
0	4	3	3	3													
0	3	4	4	4													
3	3	3	0	0													
3	3	0	0	0													

Figure 45 – Connected component algorithm output



Flow 1 – Connected component algorithm flow diagram.

<i>SurfaceInfo.Area(ss)</i>	<i>Number of connected the same SurfaceType in the scale space (ss).</i>
<i>SurfaceInfo.cx(ss)</i>	<i>sum of x values the same SurfaceType in the scale space(ss). (cx/Area is center of mass along x direction)</i>
<i>SurfaceInfo.cy(ss)</i>	<i>sum of y values the same SurfaceType in the scale space(ss). (cy/Area is center of mass along y direction)</i>
<i>SurfaceInfo.cs</i>	<i>Center of scale space levels. (calculated by areas in each scale)</i>

CHAPTER 5

RESULTS & COMPARISONS

5.1 Output Validation and Comparisons

In this section result of thesis and comparisons are supplied. Firstly, we will compare outputs of two DCP types with their MatLab simulation. In this way, we will prove the hardware of implemented algorithms.

Secondly, we will compare outputs of DCP type1 and DCP type 2. Effect of gradient approximations will be examined in both artificial and real range data.

Besides, we will see effect of quantization on range map, using different fixed-point representations. We will compare outputs of quantized range map with outputs of the software using float 64 data format.

Another comparison will be about rotation. We will see whether the hardware is rotation invariant or not.

DCP provides scale space analysis. Therefore, scale space outputs also should be considered. We will be interested in both scale space levels and their outputs.

5.1.1 Notation of Classification




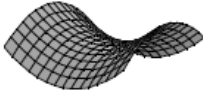

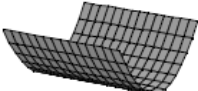

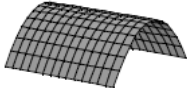



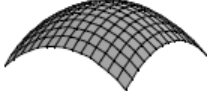
To visualize surface classifications, colorful plots are used. Meanings of colors and their visualizations are given in Table 7.

5.1.2 Simulation and Hardware Outputs

Simulations of DCPs are implemented in MatLab® software. First, we develop hardware algorithm in the software, upon the implementation of it, we

compare the outputs, gradient, scale space and classification outputs. Also we follow the signals in the FPGA using Chip Scope® signal browser tool. The tool provides us instantaneous signal debugging and comparison of values with simulation.

Table 7 – Colored representation of surface classification.

Color	Classification	Visualization [24]
	Plane (0)	
	Hyperbolic (Any) (1)	
	Concave Cylindrical (2)	
	Convex Cylindrical (3)	
	Concave Elliptical (4)	
	Convex Elliptical (5)	

An artificial range map data, shown in Figure 46, is used for testing. The map contains all type of surfaces. Simulation and FPGA outputs are shown in Figure 47, and they are exactly the same.

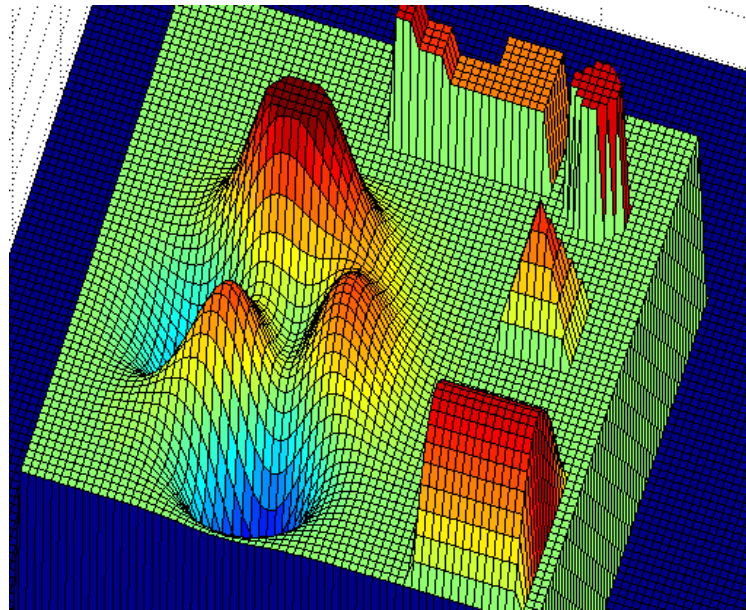


Figure 46 – Artificial surfaces

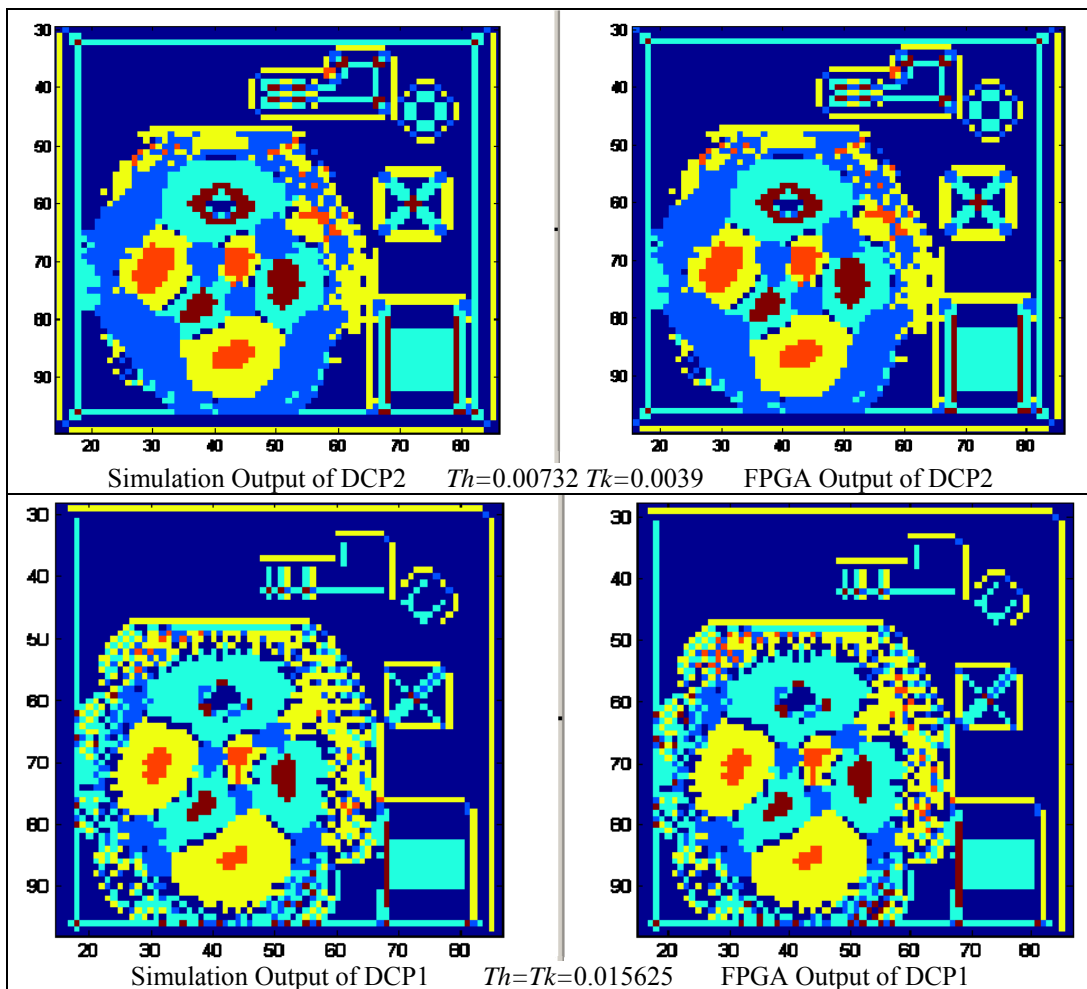


Figure 47 – Simulation and real hardware outputs of the artificial data

5.1.3 Comparison of DCP Outputs

In the previous section we give outputs of DCP for synthetic range map. A real range data taken from USF source [13] is shown in Figure 48.

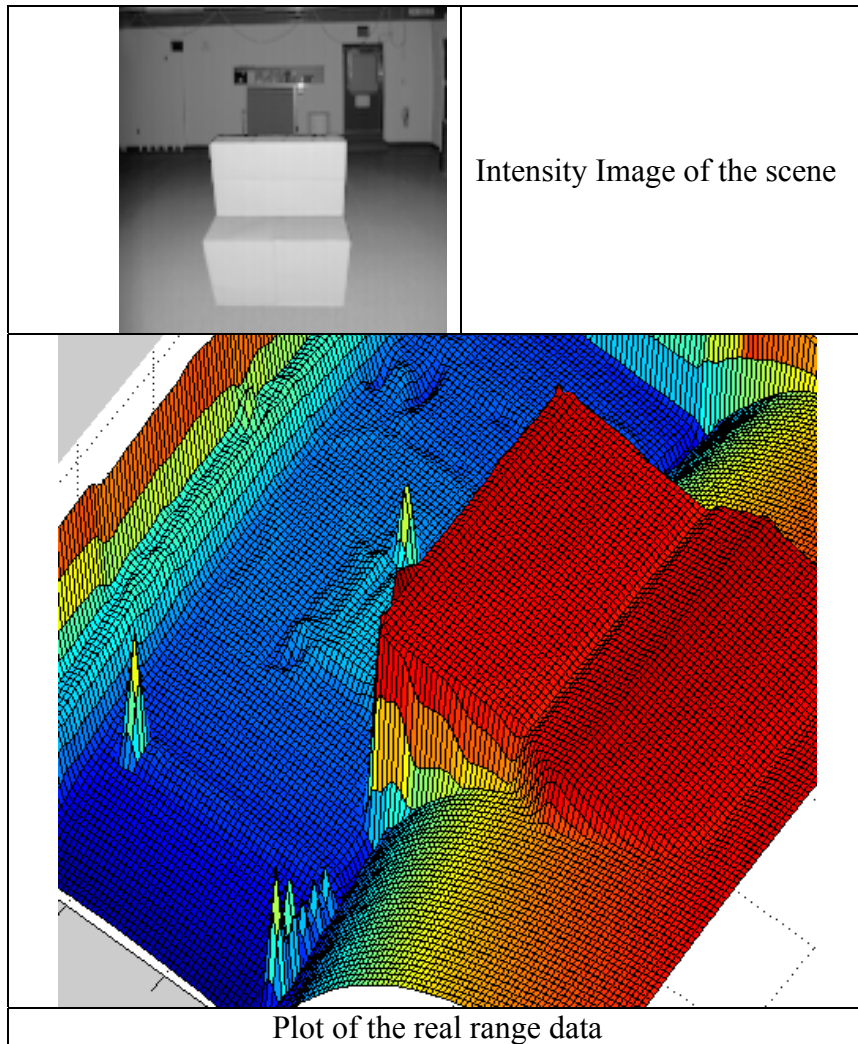


Figure 48 – Range data of a scene and its 3D plot [13]

To compare the curvature processors, we use both outputs for artificial and real range maps. Real range map outputs are shown in Figure 49 and outputs of artificial map are shown in Figure 47.

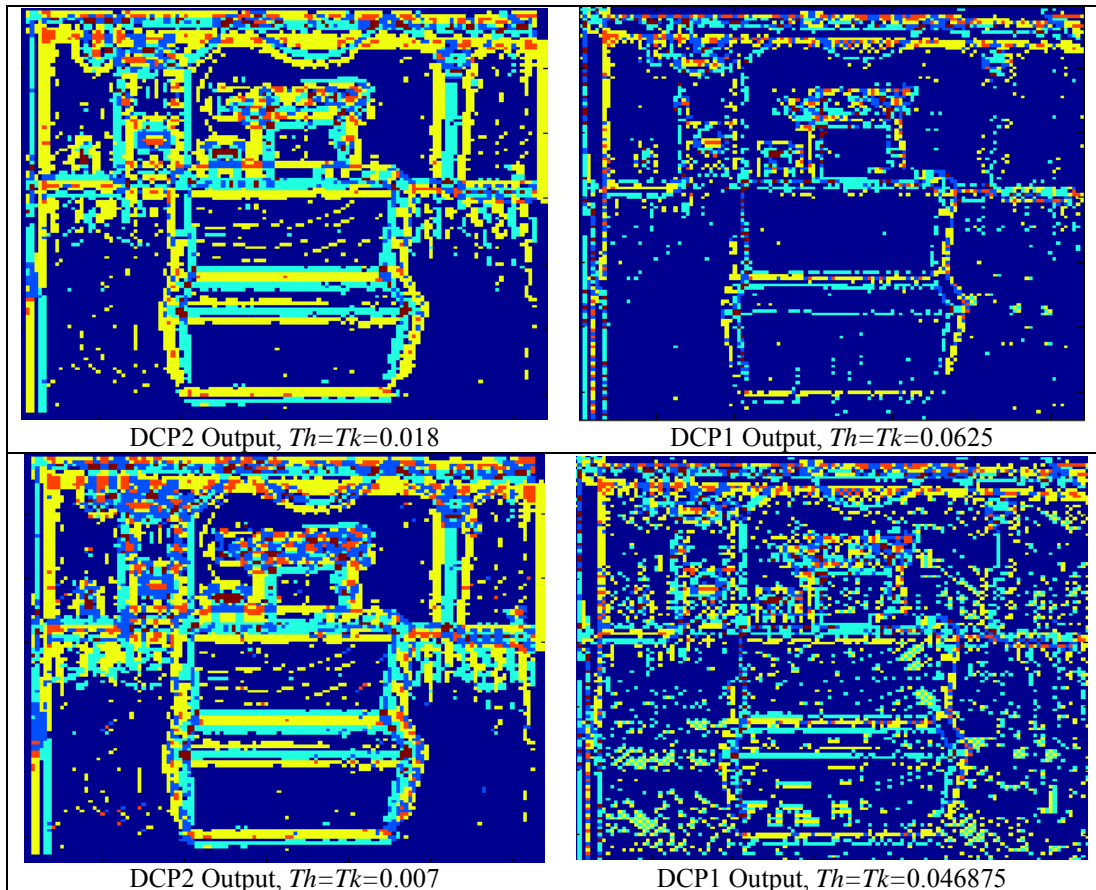


Figure 49 – Hardware outputs of the range data in Figure 48

As we see in Figure 47, DCP2 gives very nice output for artificial data. DCP1 also provides an acceptable output, but it has some errors especially on edges and hyperbolic points. Thin surfaces are also not so good. The noisy output on hyperbolic points shows us; DCP1 is more sensitive to noise on the surface. Indeed, considering real data outputs, in Figure 49, we see that DCP1 output is very noisy and DCP2 output is much better than it. We generate two outputs for two different thresholds to show disadvantage of DCP1. Selecting large threshold for DCP1, at top, disappear surfaces, i.e. plane classification is given for most of the points. However, DCP2 gives very accurate output. When we decrease threshold to see surfaces, we face with noisy output in DCP1, but DCP2 still provides a good output. Therefore, we conclude that establishing threshold can not solve problem of erroneous approximations, in Eq.3.3. Obviously, DCP2 is much better than DCP1 in real data analyzing.

5.1.4 Fix Point Quantization Effect

Digital curvature analysis in hardware requires fix point number representation. The range resolution has to be in desired fix point quantization and the map must be in the range of the selected fix point space. If these conditions are not met in the range map, the output can not be a deterministic result.

In this section, we compare the outputs of software based curvature analysis using float64 data type for range data. Then we quantize the data and process in again the software. The used data is the same in Figure 48.

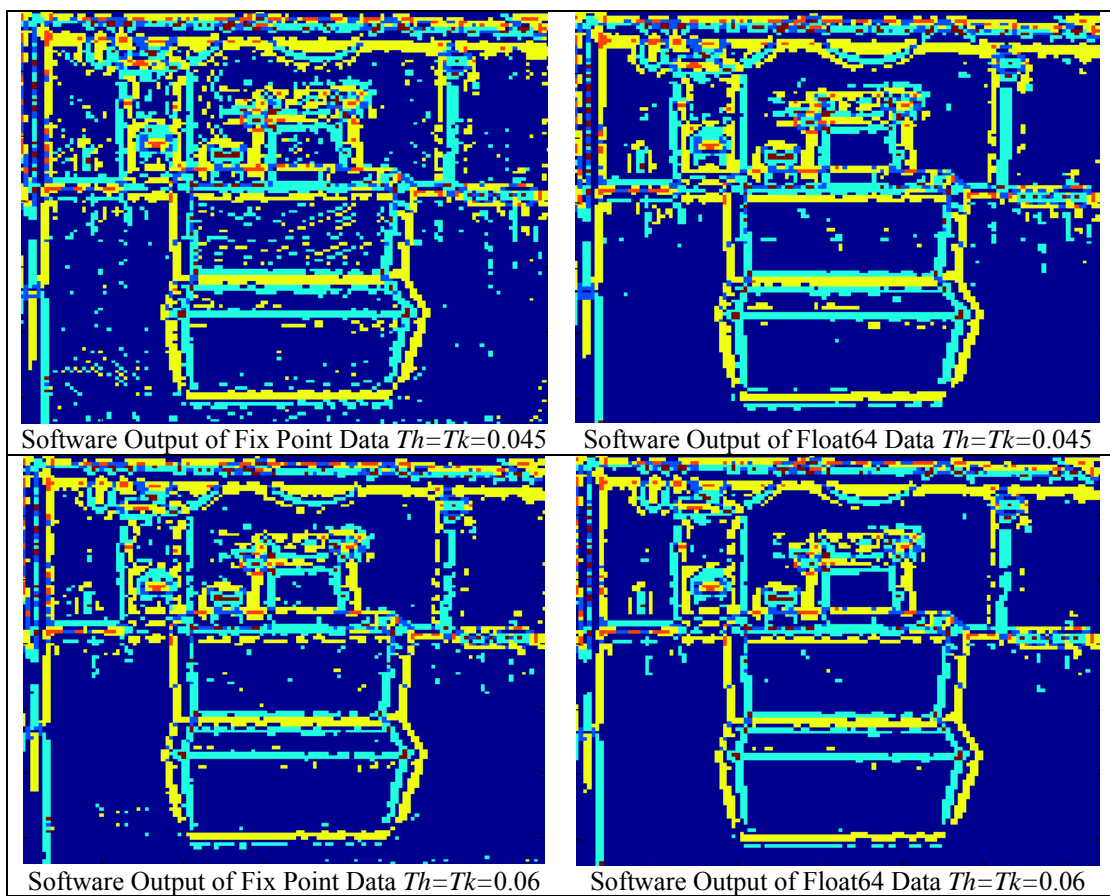


Figure 50 – Software outputs of float64 and quantized fix-point range data

In Figure 50, we see the outputs of software for different thresholds for quantized (10.3) and float data. Obviously, except small errors, preferred quantization is enough to represent surface. In software, we use full precision calculations, even if the input is fixed-point. Therefore we see only effect of range

map quantization. Comparison of software and hardware implementation is another topic.

If we decrease resolution of range map, the output will degenerate, although we use enough precision in DCP. An example is given in Figure 51. For the same thresholds, using DCP2, we use the same range data with different resolution. At left, quantization is 0.125 (10.3), the preferred resolution. At right, the range data quantized by 0.25 (10.2) and then converted to 10.3 fixed-point format again. The output shows that the resolution, 10.2, is not enough for this range data, especially in small surface patches. Note that establishing threshold may not be so helpful to deal with quantization error around small regions. The high thresholds hide both the quantization errors and details of surfaces, as shown at the bottom of the figure.

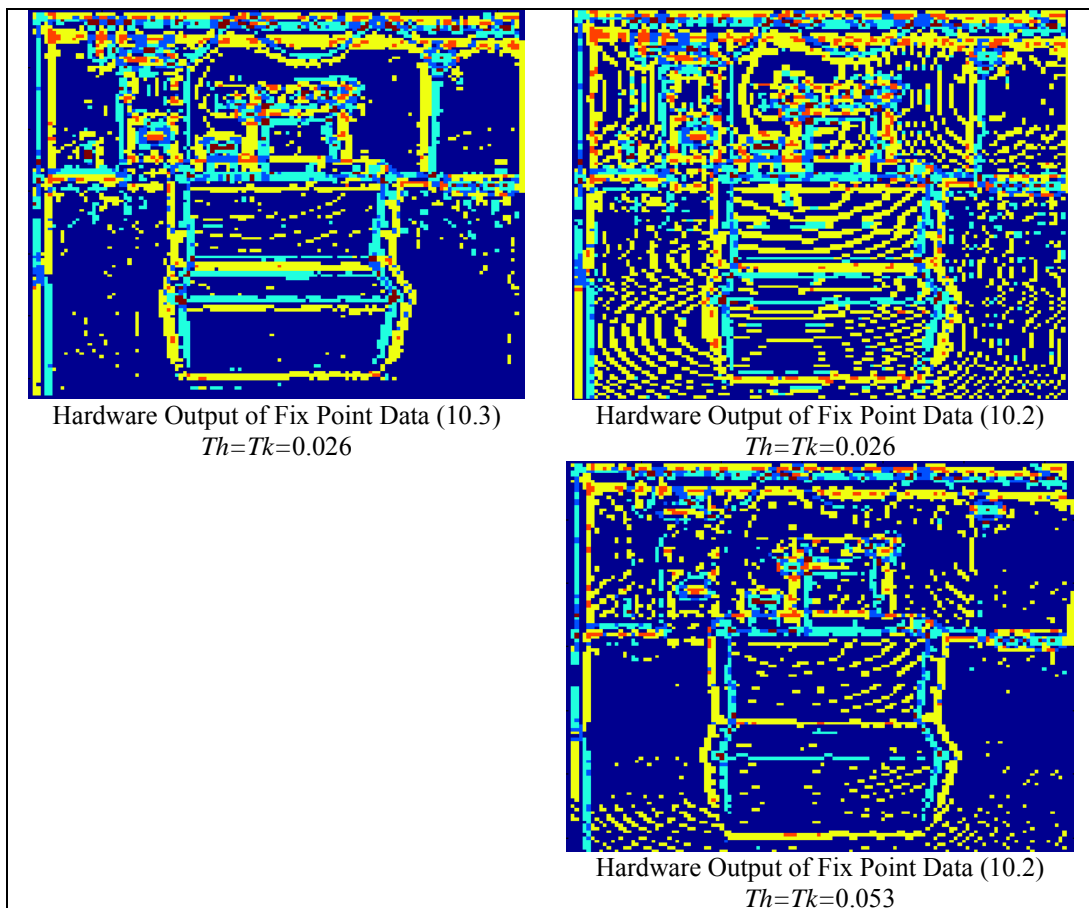


Figure 51 – Quantization effect

5.1.5 Comparison of Software and Hardware Implementations

In the previous topic, we examine range map quantization effect using software based curvature analysis. Here, we will use float data in software curvature analysis and quantized data in the hardware, DCP2, for comparisons.

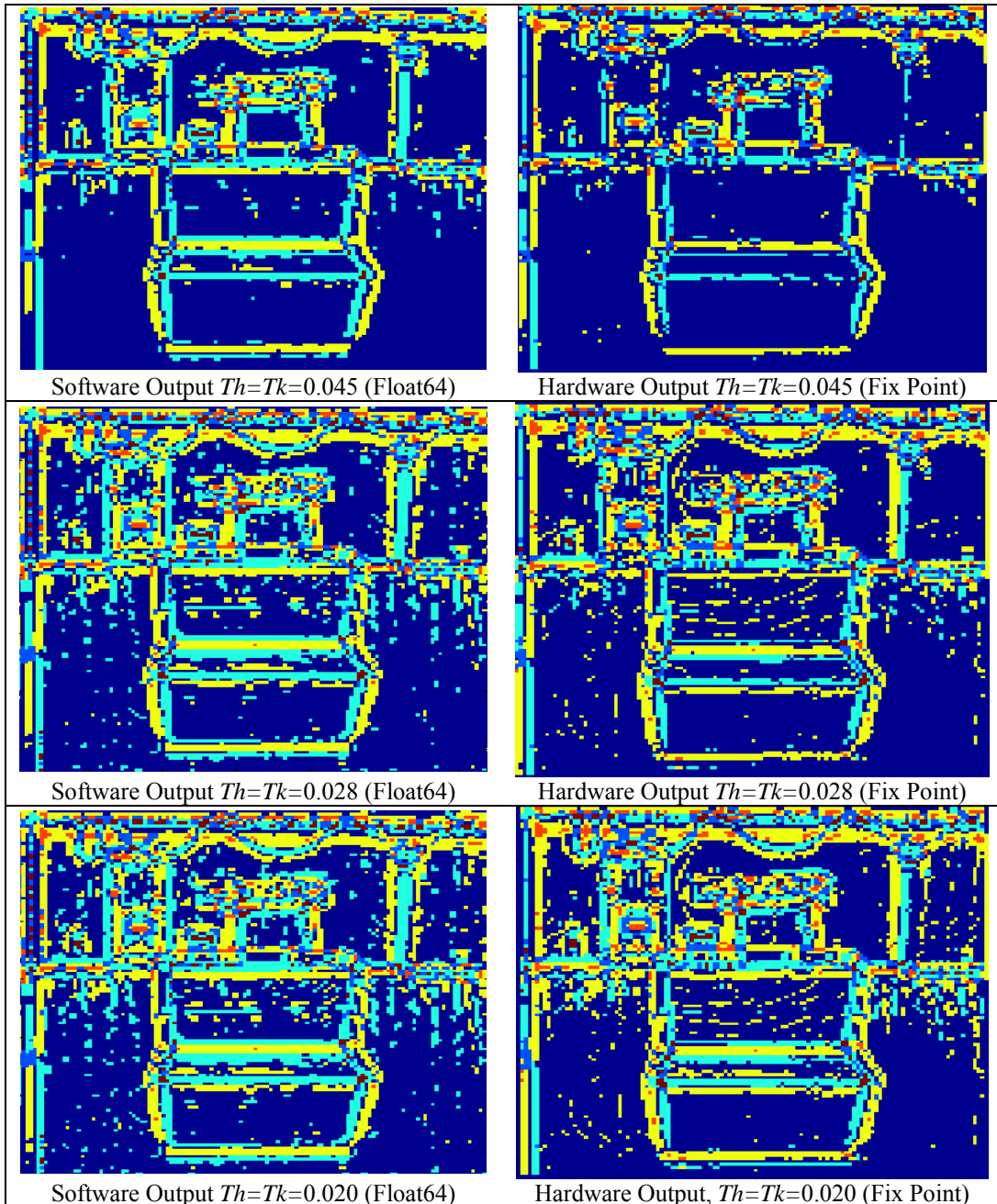


Figure 52 – Software and hardware outputs

Software application, using double data in calculations, has little better outputs than the hardware does, as we expected. The hardware uses fixed point in calculations and we also truncate some signals in HK analysis circuit to keep logic utilization low. The hardware, DCP2, has acceptable outputs in surface classification process. Discarding threshold effect, we now compare mean and Gaussian curvature values. Unfortunately, none of DCP implementation generates H and K maps, so we will get the values from simulations. Some statistical information using the real data is given in below;

	Software H	DCP2 H	DCP1 H	Software K	DCP2 K	DCP1 K
Min. Value	-2.04	-1.83	-2.85	-17.25	-17.8	-15.14
Max. Value	8.28	7.85	7.91	68.34	58.77	24.02
Mean	-0.0081	-0.0116	0.0001	0.0051	0.0047	-0.004
Variance	0.0471	0.0505	0.0885	0.4531	0.3914	0.1266

According to the mean and variance, DCP2 is closer to software algorithm in H and K map calculation. Remember that DCP2 has more reliable derivative calculation logic circuits. In H calculation, square root is used and error in the approximation causes dissimilar results, between software and hardware, more than in K calculation. Indeed, if we look at mean differences in H and K maps;

$\text{mean}(K_{software} - K_{DCP2})$	0.0059
$\text{mean}(K_{software} - K_{DCP1})$	0.0465
$\text{mean}(H_{software} - H_{DCP2})$	0.0326
$\text{mean}(H_{software} - H_{DCP1})$	0.0835

Obviously, DCP1 generates much noisy H and K values than DCP2 does. Also, H calculation is noisier than K calculation, due to linear square root approximation.

5.1.6 Rounding Operator Effect in DCP2

As we mentioned in the previous chapter, rounding logic in derivative calculation has a very important effect at output accuracy. Miscomputation of first and especially second derivatives causes wrong classification, because the derivatives are multiplied with each other and these operations increase the error drastically. The outputs without rounding logic are shown in Figure 53.

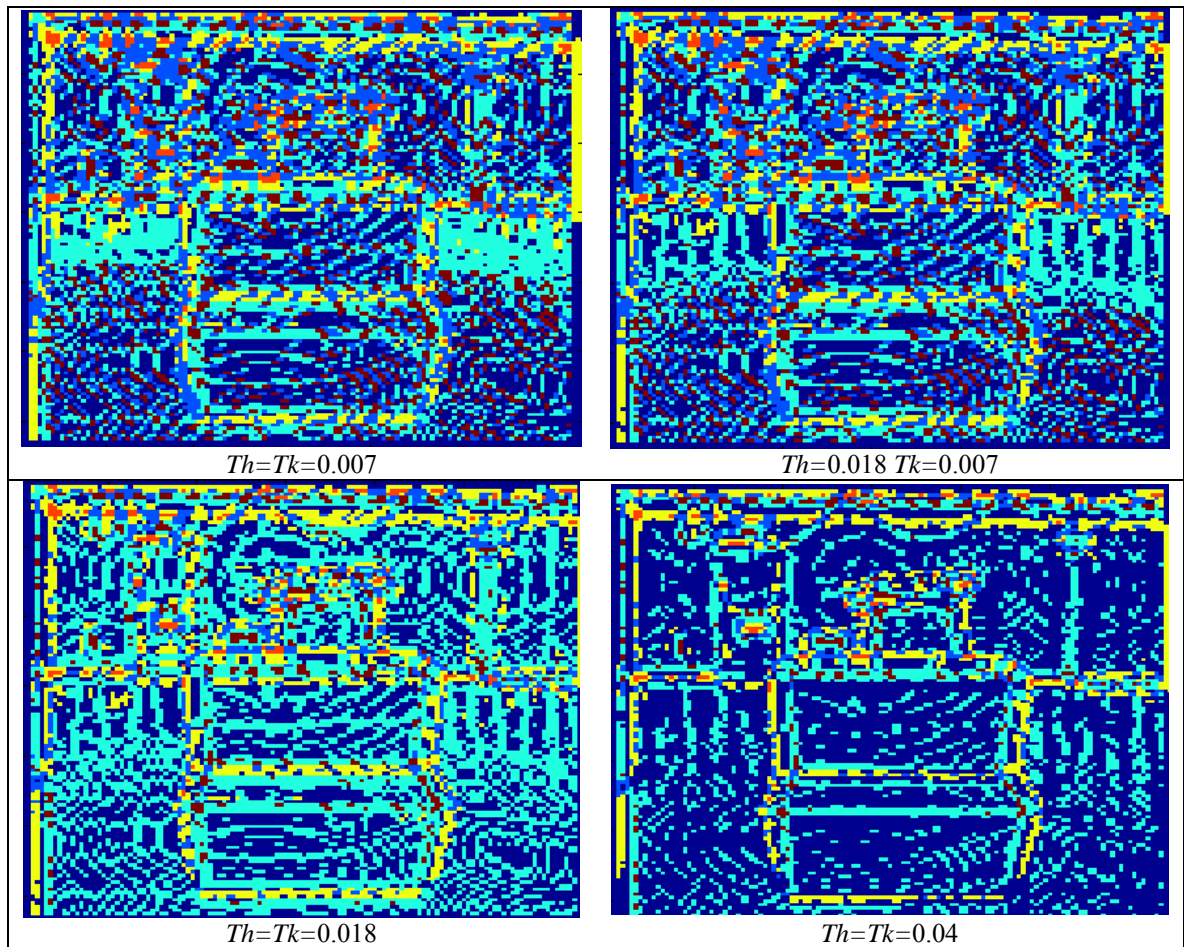


Figure 53 – Outputs without rounding operation in DCP2

If we compare the outputs with Figure 49, considering thresholds values, we see that rounding operation provides pretty better results. The output noises can not be eliminated, even if high thresholds are used, as seen in Figure 53.

5.1.7 Rotation and Translation Invariance

Gaussian and mean curvatures are rotation and translation invariant. In this section we will see whether the hardware provides this facility or not. Again, we will use the same object in the same scene as seen in Figure 54.

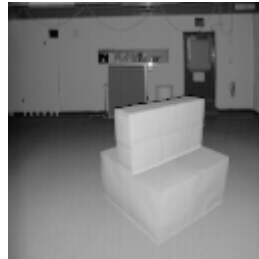


Figure 54 - Intensity image of the rotated object [12]

The output of DCP2 for the real range data is shown in Figure 55. The rotation invariance is achieved but it is not an excellent result. The software output is also similar to hardware result. The weakness results from using derivative approximation in calculations.

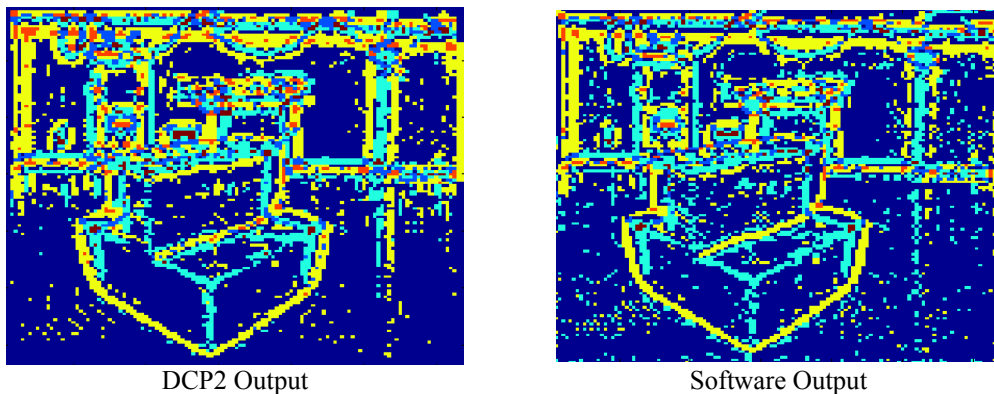


Figure 55 – Outputs of the rotated object

5.1.8 Scale Space Level Outputs

The curvature processors do not only generate surface type but also provide scale space levels. The scale spaces are constructed by using a 2x2 kernel and size of each level is half of its previous level. DCP implementations accept 8 consecutive samples in a vector. Therefore width of the map must be exact multiple of 8. For example, scale space levels of a 240x320 map can be 120x160,

60x80, but the 30x40 can not be a valid scale space because 30 is not a power of 8. For a 128x128 map, 64x64, 32x32 and 16x16 are valid scale space level sizes.

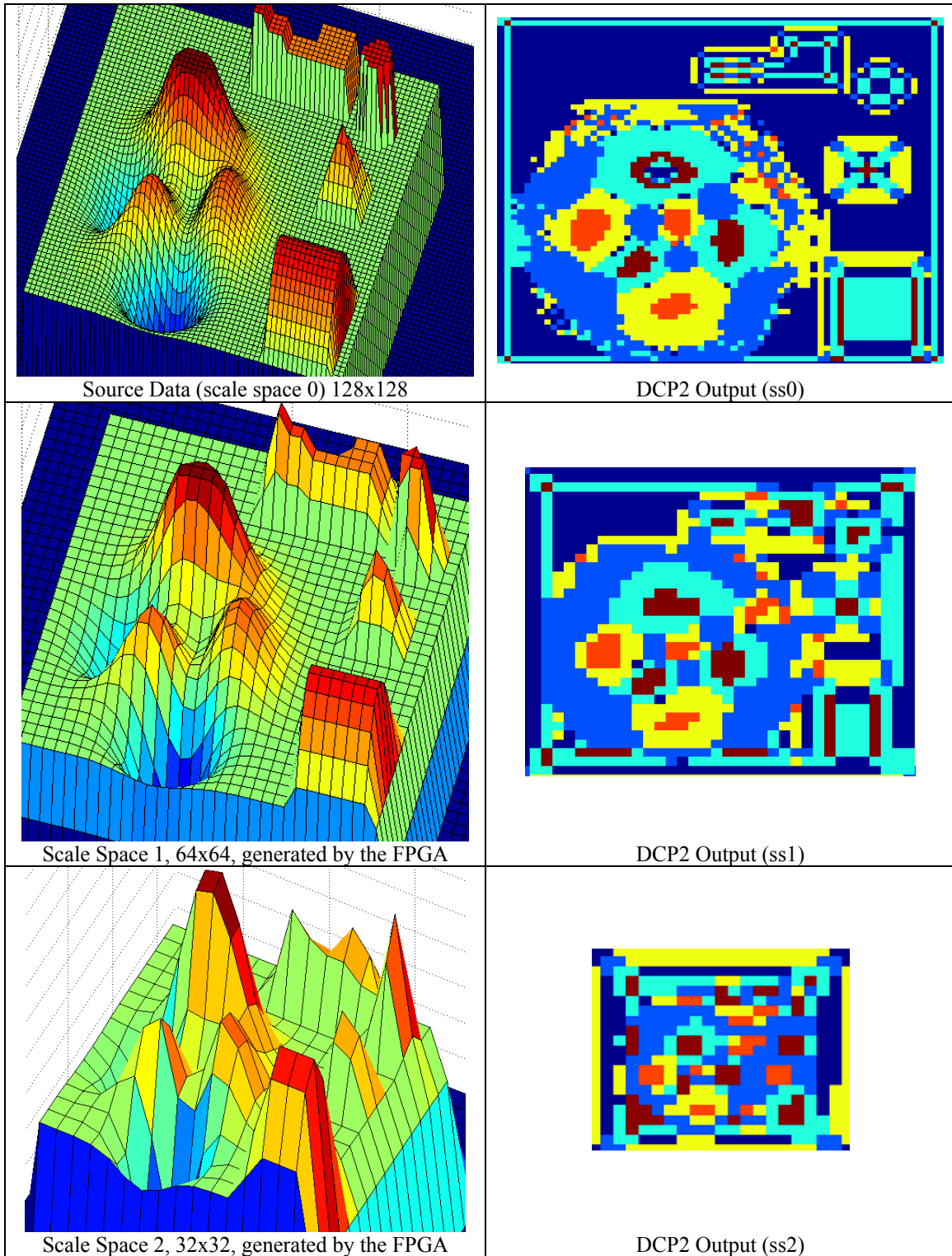


Figure 56 – Scale space levels and their analysis outputs for the artificial data.

In Figure 56, scale spaces constructed by hardware and their surface analysis results are given. Scale space analysis is a useful tool for excluding small regions in the map. For example, there is a small plane on one of the peaks. Its classification result is shown in scale space 0, ss0, analysis output. However, in the next scale space, ss1, the plane on the peak disappears and only peak classification is generated. Another clear effect is on hyperbolic regions, which is more obvious and noiseless in the ss1. Note that establishing thresholds can not provide this facility, and using high thresholds also influences other parts of map.

In Figure 57, scale space analysis outputs for real data are shown. Again small surfaces are not available the ss1 level and also some noise on the closest patch disappear. The upper level algorithm can use scale space level outputs to take more robust decisions.

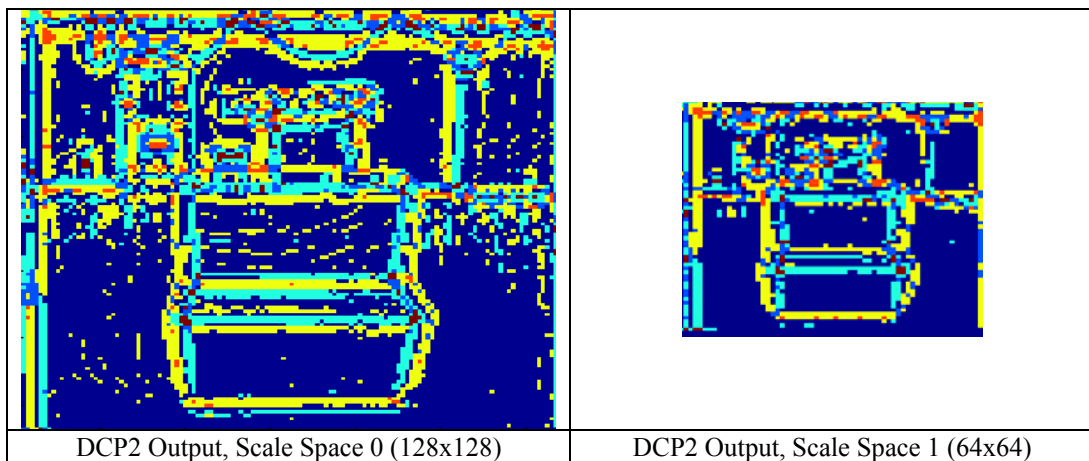


Figure 57 – Scale space levels analysis outputs for the real data.

5.2 Software Output

As we already mentioned, a connected component and center of mass algorithms are integrated into SoC design. In this section, the outputs of these algorithms are given.

In Figure 58, two scale space outputs are given. As talked about in the previous scale space output section, the small plane on the biggest peak in scale 0, $x=26$, $y=42$, is not visible in scale 1, $x=13$, $y=21$. In scale 0, 3D connected

component algorithm gives ID of '47' for the small plane. Around the plane, there is the surface '41', as seen in the figure. On the other hand, surface '47', the plane, is not available in scale 1. Surface '41' overlaps the surface '47', which affects center of mass and center scale.

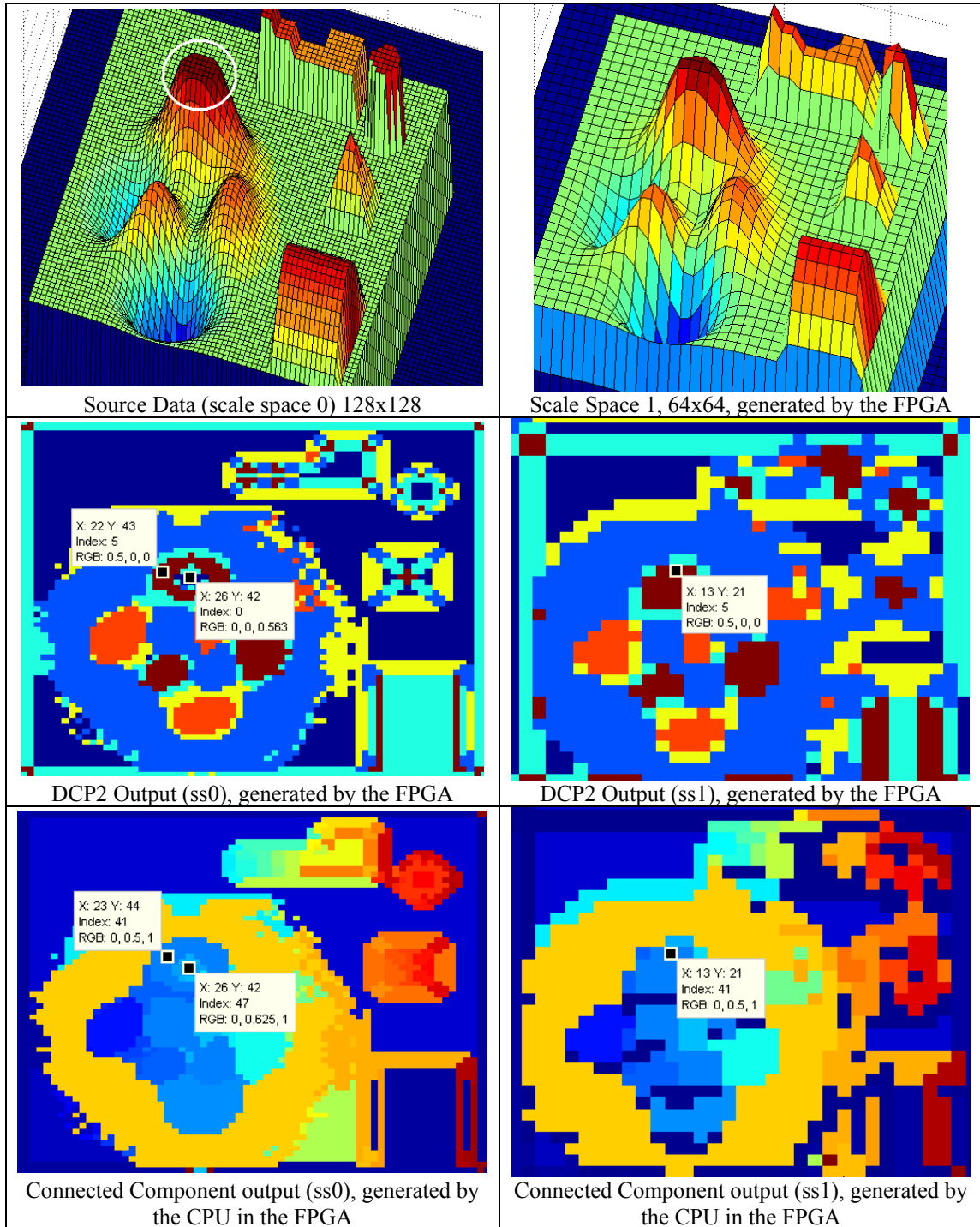


Figure 58 – Connected component algorithm outputs.

The outputs of center of mass algorithms for the examined surfaces are given in the following table.

Surface ID	Scale No	Area	Center X	Center Y
41	0	53	24	40
41	1	76	24	37
47	0	8	25	40
47	1	0	X	X

Surface 41 is available in scale 0 and scale 1. Surface 47 does not exist in scale 1 and surface 41 overlaps it. Therefore the area of surface 41 is larger in scale 1. This means that surface 41 is an important large surface and surface 47 is a small region on surface 41. Center scale of surface 41 is 0.6, using the areas (76/(76+53)). Center scale of surface 47 is obviously 0.

The connected component algorithm labels all connected surfaces, which are not given here.

5.3 Time Measurements and Comparisons

In this section, we will see the speed of proposed curvature processors and we will also make a comparison between software and hardware realizations. We start with software speeds.

5.3.1 Software Performance

The software implementation of curvature analysis is running on MatLab® software. Of course, MatLab is not a performance oriented platform, but the implementation in MathWorks, given in “appendix a” section, uses only matrix calculations and MatLab is very fast in matrix manipulations. Therefore the measurements can still give a clue about speed of software implementations. We also use quadratic surface fitting approach to calculate H and K in MatLab and we measure speed of this approach, too. To minimize effect of cache memory,

we measure the speed just after the computers are turned on, and we take two measurements to see effect of cache. Also we increase the priority of MatLab software in the Microsoft Windows, even we have no idea whether the operating system cares the priorities or not. The speeds are shown in Table 8, for two different computers.

Table 8 – Time measurements of software implementations in MatLab

	Intel® Centrino® T2400@1.83GHz x2Cores 2GB RAM Microsoft Windows XP	Intel® Xeon® X5452@3GHz x8 Cores 4GBytes RAM Microsoft Windows XP
128x128 Range Map 3 scale space levels (derivative approx.)	382ms (First Run) 166ms (Second Run)	263ms (First Run) 20ms (Second Run)
128x128 Range Map 3 scale space levels (3x3 quadratic fitting)	3181ms (First Run) 2402ms (Second Run)	2342ms (First Run) 1269ms (Second Run)

The high level CPU units, high speed memories and very impressive caches in today's computers provide very fast processing when discrete approximations are used to calculate the derivatives, as we can see in the table. The quadratic fitting, on the other hand, is very slow.

5.3.2 Hardware Performance

The implementation techniques of the hardware directly affect the performance of system. However, the technique does not affect the pixel speed of the processors, which are given in Table 9. The processing speed is also given in the table for a suggested implementation, as in Figure 25 and 8 HK Logics in each DCP. Note that, using scale space levels do not change the speed. The scale space outputs have one row latency, as previously explained. The speed in the table is the best performances the processors can achieve.

Table 9 – Possible maximum speeds of designed curvature processors.

	DCP1	DCP2
Speed of processing the input vector, (8 pixels in each vector)	22 clocks (220ns@100MHz)	32 clocks (640ns@50MHz)
Speed of processing 128x128 map (2048 vectors)	47104 clocks (471us@100MHz)	67584 clocks (1.3ms@50MHz)

We use development boards, ml403 and ml505, produced by XILINX, to realize the designs. The FPGAs on the boards are not high density chips. Therefore the processors are implemented as displayed in Figure 26 and Figure 35; only one DCP exists to analyze multiple scale spaces and only one HK logic is available to process 8 pixels, i.e. parallelism is vanished. The 8 parallel pixels are processed by sequentially as shown in Figure 35. Also scale spaces are not constructed by parallel; each scale space is generated by the logic and written into DDR RAM, than it is processed. Speed measurement of this realization including DDR RAM transactions, is given in Table 10.

Table 10 – Speed measurements of the processor realized on ml403 and ml505 boards.

	DCP1 (100MHz) on XILINX ml403 Board	DCP2 (50MHz) on XILINX ml505 Board
128x128 Range Map 1 scale space level	5.1 ms	11ms
128x128 Range Map 3 scale space levels	6 ms	17ms

Influence of RAM transactions slows down the processing speed; in each analysis, DCP reads source map, writes scale space and classifications data. Also, RAM frequency is another important parameter. In ml403, DDR is at 100MHz and in ml505, DDR2 is at 200MHz.

5.3.3 Hardware and Software Speed Comparison

Although the hardware realization technique is not the best approach in the demo boards, it is still much faster than the software. Also, if we consider the

running frequencies, the hardware is very impressive in speed performance, as we aimed in the thesis. In this thesis, real time criteria is determined as 100ms/frame for object recognition and the results show that purposed processor is much better than the requirement.

5.4 Embedded CPU Connected Component Algorithm Speed

The soft CPU, MicroBlaze, is running on 100MHz and the connected component is a very massive algorithm. The calculation times are given in following table.

	Calculation Time
Scale Space Count 1	5423ms
Scale Space Count 2	13749ms

Obviously connected component and center of mass calculations are very slow operations for this system. Optimizing the algorithms may help the timings, but providing real time operation is nearly impossible.

5.5 Power Analysis

Last analysis of DCP is about power consumption. The analysis results are taken from XILINX Power analysis tool.

Power analysis of DCP2 on ml505 is given in following table. The logic elements, explained in the previous chapters, are also given in the table. As seen, the power utilization of the logics is not so much.

Logic	Power(W)
DCP2 (Total)	0.064
Address Decoders	0.00027
Scale Space Calc.	0.00066
Derivative Logics	0.00285
BRAM Modules	0.03384
HK Surface Classification Logic	0.00978
<i>Square Root Logic (in HK Logic)</i>	<i>0.0003</i>

The system has also the CPU and some other interface components. The over all system power analysis on ml505 is given following table.

Logic	Power(W)
System Design (Total)	0.413
DCP2	0.064
CPU System	0.32
DCP CPU Interface	0.00089
DD2 High Band Width Interface	0.005

Power requirements of DCP, 64mW, is very good result for single use real time stand alone applications, without RAM interfaces and CPU. Please note that the power requirement of the logic increases if 8 HK logics are used for parallelism, which increases the speed drastically as shown in Table 9. Power consumption is approximately 0.132W when 8 HK logics are used. Also, an individual DCP instance is required for each scale spaces for real time pipeline operation. Otherwise a RAM controller must be used.

CHAPTER 5

CONCLUSION AND FUTURE WORKS

In this thesis, development of the digital hardware making surface classification on 3D range maps based on mean, K , and Gaussian, H , curvature is aimed. The thesis begins with introduction of range map description, mathematical definitions of curves, surfaces and curvature. Then FPGA design strategies and SoC design approaches are explained.

Two digital curvature analysis circuits are developed in FPGAs. The algorithms generate directly curvature signs and make classifications; H and K values are never calculated. Two implementations, using different derivative approximations, have different speed and output accuracies. Both processors use a fast square root circuit, designed in the thesis. Multiple scale space levels and their analysis outputs are also generated in parallel, unlike software techniques. Also, an extra connected component algorithm is added to the embedded software.

The outputs of both curvature processors are compared with traditional software implementations. The one of the proposed hardware, called DCP2 in the thesis, provides very similar results to software outputs. Its speed, on the other hand, is much faster than the software implementations. Other proposed technique, DCP1, is less accurate but it is extremely fast in processing of range maps.

Although the outputs of the processors are acceptable, they contain some noise. To deal with these irregularities, erosion or dilation based filters can be used at output of the processors. These operations are not implemented in the thesis and left as a future work.

An errorless 3D range map assumption is used in this thesis. A transform is required in range map generator hardware if its output is noisy. Derivation of curvature algorithms considering noisy range data is left as a future work. Although the errorless assumption is used for x and y , their values are integrated into input vector to allow the future implementations, considering erroneous values of x and y .

Also the connected component and center of mass algorithms in embedded software can be implemented on hardware, which is another future work.

This thesis gives important results about processing range maps. As we mentioned, range maps are already generated on hardware in video-rate. Processing the map in software is still using, but we have realized the processing in *real time sense* on hardware. We have seen that curvature analysis on hardware is possible and the outputs are very accurate. This work will open the way of real time object recognitions implementations for robotics or other standalone low power applications. If the connected component algorithm is implemented on hardware, leaved as a future work, the output will be a few surface features, which include center, area and type of the surface. Processing these features can be very easy for any kind of small processors, such as soft processors in FPGAs.

REFERENCES

- [1] Smriti H. Bhandari, and S. M. Deshpande. *Feature Extraction for Surface Classification*. World Academy of Science, Engineering and Technology 29 2007.
- [2] M. J. Chantler & J. Wu. *Rotation Invariant Classification of 3D Surface Textures using Photometric Stereo and Surface Magnitude Spectra*. 11th British Machine Vision Conference (BMVC2000), Vol.2, pp. 486 – 495, 2000
- [3] P.J. Besl and R.C. Jain. *Invariant surface characteristics for 3-d object recognition in range images*. Computer Vision Graphics Image Proc., 33:33–80, 1986
- [4] J. Koenderink and A. van Doorn. *Surface shape and curvature scales*. Image and vision computing, 10(8):557–565, 1992.
- [5] H. Cantzler and R. B. Fisher. *Comparison of HK and SC curvature description methods*. 3dim, pp.285, Third International Conference on 3-D Digital Imaging and Modeling (3DIM '01), 2001.
- [6] Erdem Akagündüz, Ömer Eskizara, İlkey Ulusoy. *Scale Space Approach for the Comparison of HK and SC Curvature Descriptions as Applied to Object Recognition*. 2009 IEEE International Conference on Image Processing, SA.PE.5a.
- [7] David G. Lowe. *Distinctive Image Features from Scale-Invariant Keypoints*. International Journal of Computer Vision, 2004
- [8] F. Leymarie and M. D. Levine. *Curvature Morphology*. December 1988. Proceedings of Vision Interface '89
- [9] Fabio Crosilla, Domenico Visintini, Francesco Sepic. *A Statistically Proven Automatic Curvature Based Classification Procedure of Laser Points*. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Vol. XXXVII. Part B5. Beijing 2008.
- [10] Jan Böhm, Claus Brenner. *Curvature based range image classification for object recognition*. Intelligent Robots and Computer Vision XIX: Algorithms, Techniques, and Active Vision, Volume 4197.

-
- [11] Soodamani Ramalingam, Zhi-Qiang Liu, and Dmitri Iourinski. *Curvature-Based Fuzzy Surface Classification*. Ieee Transactions On Fuzzy Systems, Vol. 14, No. 4, August 2006.
- [12] Ronald M. Summers, W. Scott Selbie, James D. Malley, Lynne M. Pusanik, Andrew J. Dwyer, Nikos A. Courcoutsakis, David J. Shaw, David E. Kleiner, Michael C. Sneller, Carol A. Langford, Steven M. Holland, James H. Shelhamer. *Polypoid Lesions of Airways: Early Experience with Computer-assisted Detection by Using Virtual Bronchoscopy and Surface Curvature*. Radiology, p331~p337, August 1998.
- [13] Computer Vision and Pattern Recognition Group in the University of South Florida, <http://marathon.csee.usf.edu/range/DataBase.html>
- [14] Ohio State University Signal Analysis and Machine Perception Laboratory database. <http://sampl.ece.ohio-state.edu/data/3DDB/RID/index.htm>
- [15] Stanford University Computer Science department.
<http://www.cs.cornell.edu/~asaxena/learningdepth/data.html>
- [16] University of Stuttgart Range Image Database.
<http://range.informatik.uni-stuttgart.de/htdocs/html/>
- [17] Harlan Hile and Colin Zheng. *Stereo Video Processing for Depth Map*. University of Washington.
- [18] Ahmad Darabiha. *Video-Rate Stereo Depth Measurement on Programmable Hardware*. Master Thesis. Graduate Department of Electrical and Computer Engineering, University of Toronto.
- [19] Tony Lindeberg. *Scale-space*. Encyclopedia of Computer Science and Engineering (Benjamin Wah, ed), John Wiley and Sons, Volume IV, pages 2495-2504.
- [20] Sebastian Montiel, Antonio Ros. *Curves and Surfaces*. American Mathematical Society. Page 8.
- [21] Theodore Shifrin. *Differential Geometry: A First Course in Curves and Surfaces*. University of Georgia.
- [22] Jeff Knisley. *Multivariable Calculus Online*; adapted from the textbook Calculus: A Modern Approach by Kevin Shirley and Jeff Knisley. Section: 3.6

Tangent Planes: <http://math.etsu.edu/Multicalc/Chap3/Chap3-6/index.htm>.

Department of Mathematics, East Tennessee State University.

[23] Jeff Knisley. *Multivariable Calculus Online*; adapted from the textbook *Calculus: A Modern Approach* by Kevin Shirley and Jeff Knisley. Section: 9.1 Curvature of a Surface: <http://math.etsu.edu/Multicalc/Chap3/Chap3-9/part1.htm>. Department of Mathematics, East Tennessee State University.

[24] Norbert Haalaa, Ralf Reulke, Michael Thies, Tobias Aschoff. *Combination of Terrestrial Laser Scanning With High Resolution Panoramic Images for Investigations in Forest Applications and Tree Species Recognition*.

[25] Lance Saldanha, Roman Lysecky. *Hardware/Software Partitioning of Floating Point Software Applications to Fixed-Pointed Coprocessor Circuits*. Embedded System Weeks CODES+ISSS Atlanta 2008.

[26] Sri Parameswaran, Haris Javaid. *Synthesis of Heterogeneous Pipelined Multiprocessor Systems using ILP : JPEG Case Study*. Embedded System Weeks CODES+ISSS Atlanta 2008.

[27] Eric Cheung, Harry Hsieh, Felice Balarin. *Software Optimization for MPSoC: A MPEG-2 Decoder Case Study*. Embedded System Weeks CODES+ISSS Atlanta 2008.

[28] Patrick J. Flynn, Anil K. Jain. *On Reliable Curvature Estimation*. In CVPR'88, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Ann Arbor, MI, June 5–9, 1988, pages 110–117.

[29] S. Lachowicz & Hans-Jörg Pflaiderer. *Fast Evaluation of the Square Root and Other Nonlinear Functions in FPGA*. pp.474-477, 4th IEEE International Symposium on Electronic Design, Test and Applications, 2008.

[30] İ. Özkan. *Traffic Sign Detection Using FPGA*. Master Thesis, Graduate Department of Electrical and Electronics Engineering, Middle East Technical University.

[31] Dimitris Bariamis, Dimitris K. Iakovidis. *A generalized architecture for efficient multi-segment logarithm approximation*. Department of Informatics and Telecommunications, University of Athens. The 7th International Conference on Computing, Communications and Control Technologies (CCCT), 2009.

[32] M.Bajger & Amos R. Omondi. *Implement of Square-Root and Exponential Functions for Large FPGAs*. 11th Asia-Pacific Conference, ACSAC 2006, Shanghai, China, September 6-8, 2006.

APPENDIX A

MEAN AND GAUSSIAN CURVATURE IN MATLAB

The code below calculates mean and Gaussian curvatures in MatLab software using matrix manipulations. Required derivatives are calculated by using discrete approximations, available in MatLab. The code is taken from MathWorks;

<http://www.mathworks.com/matlabcentral/forums/20208/1/HK.m>

```

% /*****
% Function Name : HK
% Author      Alireza Bossaghzadeh
% PURPOSE:    The Code calculate the Mean and Gaussian Curvature according to
the method described in
% Modern Differential Geometry of Curves and Surfaces with Mathematica.2nd
ed, 1997 (p. 377).

% The method Used in the Code:
% If  $x:U \rightarrow R^3$  is a regular patch, then the mean curvature is given by
%
%            $H = (eG - 2fF + gE) / (2(EG - F^2)),$ 
%            $G = (eg - f^2) / (EG - F^2)$ 
% where E, F, and G are coefficients of the first fundamental form and
% e, f, and g are coefficients of the second fundamental form
% For more information see Links Below
% http://mathworld.wolfram.com/MeanCurvature.html
% http://mathworld.wolfram.com/MongePatch.html
% http://mathworld.wolfram.com/GaussianCurvature.html

% Function Variables:
% Input          I    mesh contain depth values
% outputs        H    Contain Mean Curvature of surface
%                K    Contain Gaussian Curvature of surface
% Example        [H K]=HK(I);

% In the case of any problem you can call me by
% Email:Alibossagh@yahoo.co.uk

% Version:      1.00          Published: 2008 June 07

%This Code was written By Alireza Bossaghzadeh.
%In the case of any problem you can contact me By
%Email:Alibossagh@yahooc.o.uk
```

```

function [H K]=HK(Z)

% Calculate base parameters
Zx =gradient(Z);
Zxx =gradient(Zx);
Zy =gradient(Z')';
Zyy =gradient(Zy')';
Zxy =gradient(Zx')';

%Calculate First Fundamental Form coefficients
E=1+Zx.^2;
F=Zx.*Zy;
G=1+Zy.^2;
%Calculate First Fundamental Form coefficients
nom=sqrt(1+Zx.^2+Zy.^2);
e=Zxx./nom;
f=Zxy./nom;
g=Zyy./nom;

% Calculate Mean Curvature
H=-(e.*G-2*f.*F+g.*E)/(2*(E.*G-F.^2));

%This code also can be used
% H=(1+Zx.^2).*Zyy-2.*Zx.*Zy.*Zxy+(1+Zy.^2).*Zxx;
% H=-H./(2.*(1+Zx.^2+Zy.^2).^(3/2));

% Calculate Gaussian Curvature
K=(e.*g-f.^2)/(E.*G-F.^2);

```