PARALLEL IMPLEMENTATION OF BIOLOGICALLY INSPIRED OBJECT RECOGNITION TECHNIQUES USING CUDA

CHINNAWAT DANAKEAW

ANON PURACHAKA

BACHELOR OF ENGINEERING PROGRAM IN SOFTWARE ENGINEERING
INTERNATIONAL COLLEGE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2013

Thesis – Academic Year 2013

B.Eng. in Software Engineering

International College

King Mongkut's Institute of Technology Ladkrabang

Title: Parallel Implementation of Biologically Inspired Object Recognition Techniques using CUDA

Authors:

1. Mr. Chinnawat Danakeaw Student ID : 53090007
2. Mr. Anon Purachaka Student ID : 53090027

Approved for submission

.....

(Dr. Natthapong Jungteerapanich)

Advisor

INTERNATIONAL COLLEGE KMITL

Date 29/8/2014

.....

(Dr. Ukrit Watchareeruetai)

Co-advisor

Date .....

Parallel Implementation of Biologically Inspired Object Recognition Techniques using CUDA

Mr. Chinnawat	Danakeaw	53090007
Mr. Anon	Purachaka	53090027
Dr. Natthapong	Jungteerapanich	Advisor
Dr. Ukrit	Watchareeruetai	Co-advisor

Academic Year 2013

ABSTRACT

This project studied a bio-inspired algorithm for object recognition based on human visual perception. In particular, we studied the HMAX algorithm, which simulates the working of the primary visual cortices in human brains and implemented the algorithm in parallel programming using CUDA. We experimented the speed-up of our parallel implementation of the HMAX algorithm over a normal sequential implementation. Depending on image sizes, the speed-up of over 500 times was achieved. To illustrate the application of the implemented algorithm, we developed a prototype of an image library application. The application facilitates the organization of the users' images and helps classifying the images by automatically suggesting tags for the added images.

Keywords: *Object recognition, Parallel programming, CUDA programming, GP-GPU, Gabor filter, Human Visual System.*

Acknowledgements

We would like to express our deepest appreciation to all those who provided us the possibility to complete this report. The appreciate guidance give to our fourth year project adviser, Dr. Natthapong Jungteerapanich and Dr. Ukrit Watchareeruetai, whose contribution in stimulating suggestions and encouragement, helped us to coordinate our project especially in ideal, teaching and writing this report. Furthermore, we would also like to acknowledge with much appreciation the crucial role of the staff of International College King Mongkut's Institute of Technology Ladkrabang, who gave the permission to use all required equipment and the necessary materials to complete our senior project "*Parallel Implementation of Biologically Inspired Object Recognition Techniques using CUDA*". Last but not least, many thanks go to the head of the project, Dr. Natthapong Jungteerapanich and Dr. Ukrit Watchareeruetai again whose have invested his full effort in guiding the team in achieving the goal. We have to appreciate the guidance given by other supervisor as well as the panels especially in our project presentation that has improved our presentation skills thanks to their comment and advices.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Most object recognition techniques developed so far are used for specific purposes or applications, such as face recognition and license plate recognition. Recently, there have been researches on object recognition techniques which are based on human visual perception. One prominent example is the hierarchical model by Riesenhuber and Poggio, which simulates the working of the primary visual cortices in human brains. This algorithm has been shown to successfully recognize any object in general. However, the sequential implementation of the algorithm performs very inefficiently. Consequently, the implemented algorithm is only practical for small images. Since human visual perception is highly parallel in nature, the algorithm can be optimized through the use of parallel programming. This motivates us to study and implement the algorithm as a parallel program.

In this project, we decide to adopt CUDA [8] in our implementation. CUDA is a platform introduced by NVIDIA for developing parallel programs by utilizing the numerous cores in graphic processing units. There are a number of reasons for choosing CUDA for our project. Firstly, CUDA does not require expensive hardware as it can be developed on any computer with an NVDIA GPU. Secondly, CUDA is widely used for general-purpose computation on GPUs, so that the tutorial books and documentations are broadly available. Finally, the syntax of CUDA is very similar to C language which makes it easy to use and understand.

1.2 Scope of Work

- Study the HMAX model and related models of human vision.
- Study classification techniques for object recognition.
- Develop object recognition program based on the studied model with the use of the CUDA library.
- Develop a program which enables the user to store photos, add tags to photos, and automatically suggests tags to the user based on the content of the photo.

1.3 Review of Related Works

- In the work proposed by *Riesenhuber, M. & Poggio, T* [5], visual processing in the visual cortex is modeled as a hierarchical representation, extended from the model of simple to complex cells of Hubel and Wiesel. They did little quantitative modeling to explore the biological feasibility to explain the aspects of high-level visual processing such as object recognition. The model is based on the MAX operation applied to inputs to certain cortical neurons that may have a general role in cortical function. The model is named HMAX by Mike Tarr [12]. This model is described in more detail in section 2.1.

- A paper [16] written by Jim Mutch and his colleagues talks about the framework for a fast simulation of cortical organization. CNS is a speedy development environment for cortical models. By utilizing GPUs, the simulation of cortical models in CNS can be faster 80-100x faster. Most viewpoints of a model are defined through MATLAB. For the purpose of CNS, "a cortical" model is defined to be a network model which consists of some number of N-dimensional layer of cell, where each layer encodes some N-dimension feature space. All the cells in a layer have to be of the same type, using the same algorithm. The license of CNS is under GNU General Public License.

Chapter 2

Background Knowledge

2.1 The Visual Physiology

2.1.1 The Visual Cortex

Visual cortex is at the lower rear of the brain. Projections that come from our eyes first go through a stopover point at the middle of the brain which is called the LGN and from there the signals are passed on to the visual cortex. The visual cortex composed of the number of areas. The first area to receive the single from the eyes which is called V1 and is also called the *primary visual cortex* or the *striate cortex*.

Signals from V1 will then be passed on to another area within the visual cortex. V2 up to V8 are called *extra-striate visual cortical areas*.

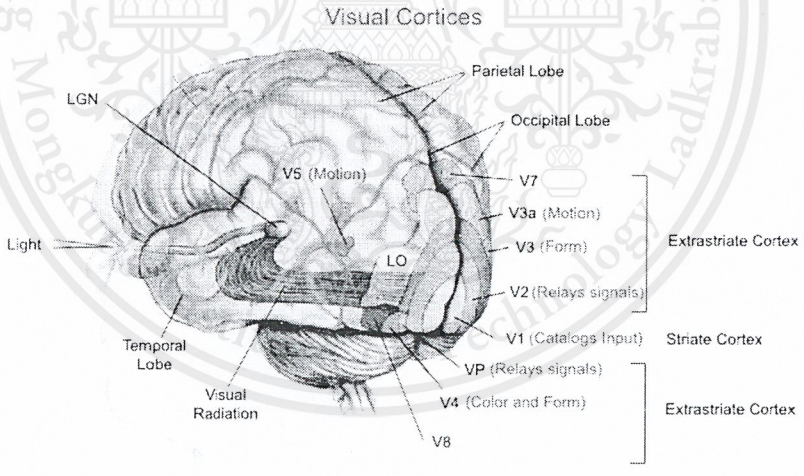

Figure 2.1 Brain's visual cortex[1]

2.1.2 Primary Visual Cortex (V1 Cells)

According to the discovery made by Hubel and Wiesel, there are 2 types of cells in the V1 cortex: simple cells and complex cells. In an experiment by Hubel and Weisel, simple cells responds to orientation and position of a line.

---

[1] *Image source:* Siry's Ecology Forum. http://myweb.rollins.edu/jsiry/Visual_Cortex.html. Accessed Oct 12, 2013

2.1.2.1 Simple Cells

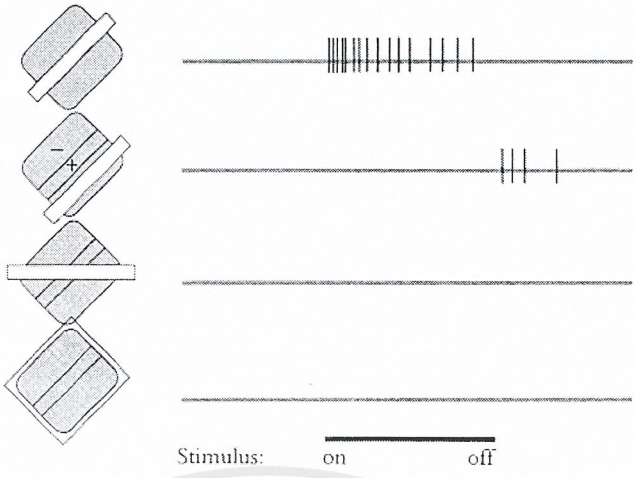

Figure 2.2 Stimulation Simple cells[2]

In an experiment by Hubel and Weisel, simple cells responds to orientation and position of a line. The most effective stimulus for this particular receptive field (Top) is one that puts a lot of light in the excitatory region, and only a little in the inhibitory region. It must have the right orientation, the right position, and the right size. Stimuli that are non-optimal in terms of position (middle), or orientation (middle), or size (bottom) are less effective.

2.1.2.2 Complex Cells

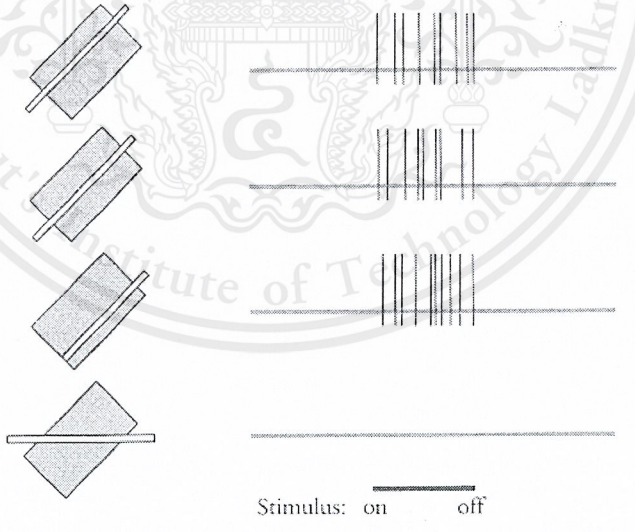

Figure 2.3 Stimulation Complex cells

---

[2] ***Image source:*** The simple and complex cells. http://fourier.eng.hmc.edu/e180/lectures/v1/node7.html. Accessed Oct 12th, 2013

Complex cells are the most numerous in V1. It looks like Simple cells, they respond only to appropriately oriented stimuli, but unlike Simple cells, Complex cells responds to orientation of a line regardless of their position.

2.1.2.3 Primary Visual Cortex (V1 Cells) Architecture

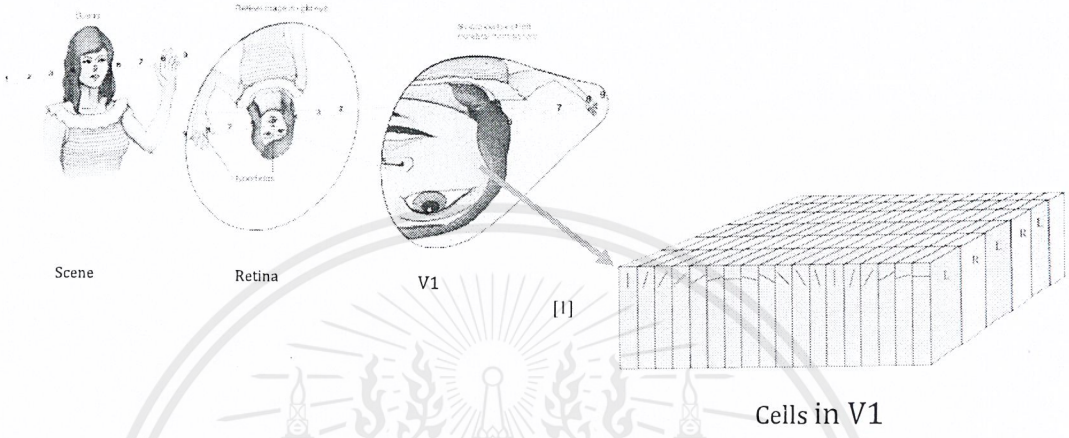

Figure 2.4 V1 Architecture[3]

In experiment by Hubel and Wiesel. Firstly, they discover that cell in V1 are arranged in a precise and ordered manner. It is called "Orientation columns". It detects orientation image and compare orientation in each V1 cell.

2.1.3 Bio-Inspired Model of Object Recognition System

Biological vision is another technique that has recently been a topic of interest for many researchers. This looks into the way the human visual system works and adopts it into computational system. Human visual system outperforms any *state-of-the-art systems*[4] in computer vision. Consequently, the researchers have studied the way the information is being processed in the visual system and tried to develop computational models.

In 1999, Riesenhuber and Poggio developed what is called as "the standard model of object recognition" based on Hubel and Wisiel theory of simple cells to complex cells at the visual system [5]". The model composed of hierarchical layers. Each layer has S units and C units. S units perform template matching of size and orientation. The result of the S unit is grouped and used as an input to C units that perform the MAX operation.

---

[3] **Image source:** Cortical Magnification. http://www.cns.nyu.edu/~david/courses/perception/lecturenotes/V1/lgn-V1.html/. Accessed Oct 13th, 2013

[4] **State-of-the-art systems** is the latest and advantage stage of technology, art, science.

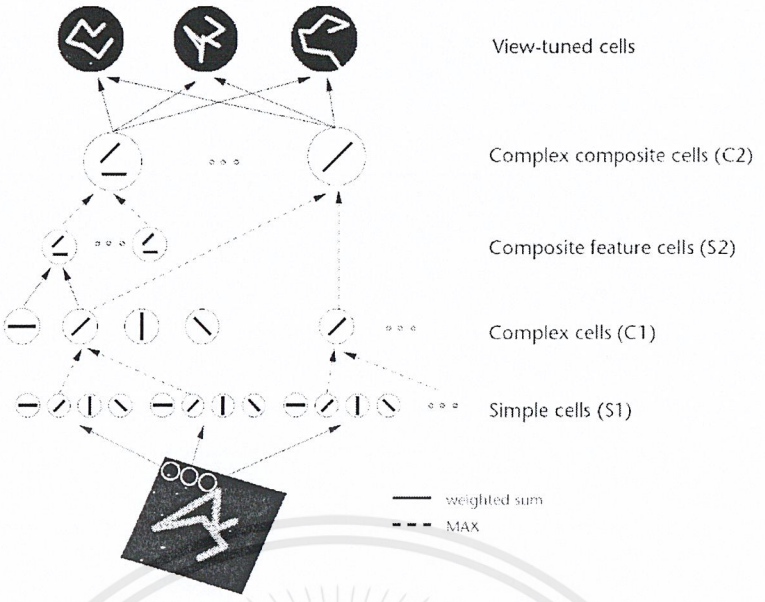

Figure 2.5 Schematic of Riesenhuber and Pogio's Hierarchical Model of Object Recognition[5]

2.1 HMAX Model

As a Bio-Inspired Model of Object Recognition System (In figure 3.6). The basic HMAX model consists of a hierarchy of five levels, from the S1 layer with simple-cell like response properties to the view-tuned unit that classifies feature vector of input image.

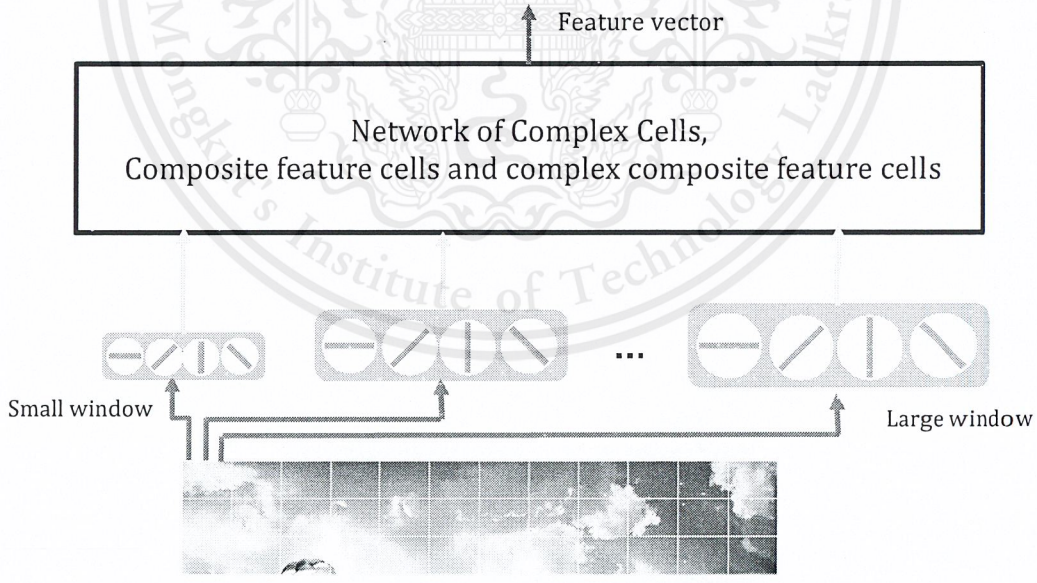

Figure 2.6 Riesenhuber and Pogio's Hierarchical Model

[5] ***Image source:*** Riesenhuber, M. & Poggio, T. (1999). *Hierarchical Models of Object Recognition in Cortex.* Nature Neuroscience **2**: P. 1021

2.1.1 Simple Cells (S1) Layer

In the HMAX model, S1 layer is responsible for orientation bar detecting, all pixels in input images are applied by Gabor filter of orientations 0°, 45°, 90°, and 135°, sizes from 7x7 to 29x29 pixels in odd steps. The result values are normalize to -1 and 1 interval and then take absolute to 0 – 1.The results of S1 layer called S1 units is fed further to next layer.

$$image:\ I[x,y] \quad \text{Where,}\quad x = 0,1,2,\dots,127,\ y = 0,1,2,\dots,127 \tag{1}$$

$$S1[d][s][x][y] = Gabor_{s,b}\,[I,x,y] \tag{2}$$

Where $s = 7\times 7, 9\times 9, \dots, 29\times 29$,

$d = 0°, 45°, 90°, 135°$

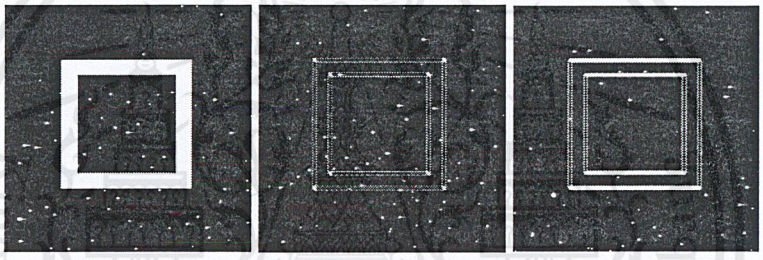

Figure 2.7 The result of S1 layer

1. Input Image. 2. Result from Gabor filter of 45° 3. Result from Gabor filter of 90°

2.1.2 Complex Cells (C1) Layer

This layer's responsibility is to provide translation-invariant for orientation bar in local area windows. Filter band is introduced a group of S1 units by the size of applied Gabor filter in previous layer 7x7 to 9x9 as small band, 11x11 to 15x15 and 17x17 and 21x21 as intermediate band, 23x23 to 29x29 as large band. The size of windows are specified by pooling range values from 4x4 for a small band, 6x6 and 9x9 for intermediate band, and 12x12 for large band and each adjacent window overlap 2 pixels to each other. C1 units are the result for MAX operation of all pooling range in all bands, means that C1 units determine by the local strongest input receives from S1 layer.

$$C1[b][d][x_{c1}][y_{c1}] = Max(S1[d][s][(p_b - o)(x_{c1} + i)][(p_b - o)(y_{c1} + j)]) \tag{3}$$

where, $i,j = 0,1,2,\dots,p_b - 1,\ o = 2$

$$s \in b \tag{4}$$

$$p_b = \begin{cases} 4\ if\ b = s \\ 6\ if\ b = m \\ 9\ if\ b = l \\ 12\ if\ b = xl \end{cases}$$

where, $s = \{7x7, 9x9\}$

$m = \{11x11, 13x13, 15x15\}$

$l = \{17x17, 19x19, 21x21\}$

$xl = \{32x32, 25x25, 27x27, 29x29\}$

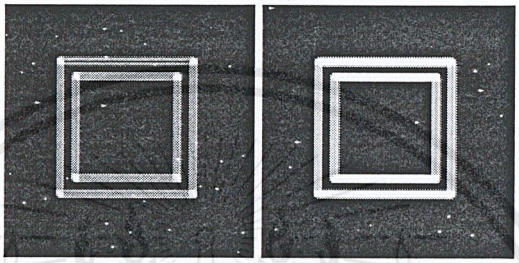

Figure 2.8 The result of C1 layer

2.1.3 Composite Feature (S2) Layer

For each filter band, non-overlapping windows of 2x2 C1 units are grouped to form up a composition of 4 orientation bars to detect and extract the structure of object in the image, so that there 256 different types of S2 units in each filter band from all possible 2x2 arrangements. The S2 unit value is a Gaussian function as follows.

$$S2[d_1][d_2][d_3][d_4][b][x_{s2}][y_{s2}] = e^{-\mathrm{P}} \quad (5)$$

$$P = SUM(expr_i)\ \ for\ all\ i = 0, \ldots, 3 \quad (6)$$

$$expr_0 = (C1[b][d_1][2x_{s2}][2y_{s2}] - 1)^2$$

$$expr_1 = (C1[b][d_2][2x_{s2} + 1][2y_{s2}] - 1)^2$$

$$expr_2 = (C1[b][d_3][2x_{s2}][2y_{s2} + 1] - 1)^2$$

$$expr_3 = (C1[b][d_3][2x_{s2} + 1][2y_{s2} + 1] - 1)^2$$

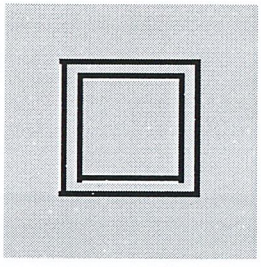

Figure 2.9 The result of C1 layer

2.1.4 Complex Composite Cells (C2) Layer

A C2 unit is calculated from MAX operation of a single arrangement of 4 orientation bars across all bands. So that C2 units are 256 elements of a feature vector produced by this model (HMAX), each element represents the existence of a certain composition in the whole area of an image.

$$C2[d_1][d_2][d_3][d_4] = MAX(\, S2[d_1][d_2][d_3][d_4][b][x_{s2}][y_{s2}]) \quad (7)$$

$$for\ all\ b, x_{s2}, y_{s2}$$

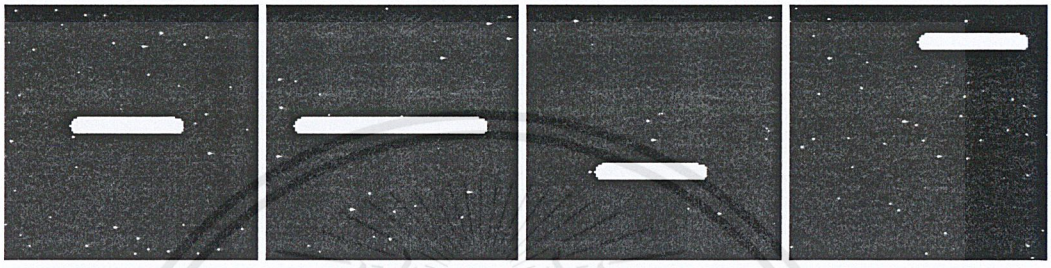

Figure 2.10 Input of HMAX model

2.2 Gabor Filter

Simple cells (S1) layers can be modeled by Gabor function to detect orientation in each window in the interested region. Image areas, which have the same orientation as applied Gabor filter, will be enhanced.

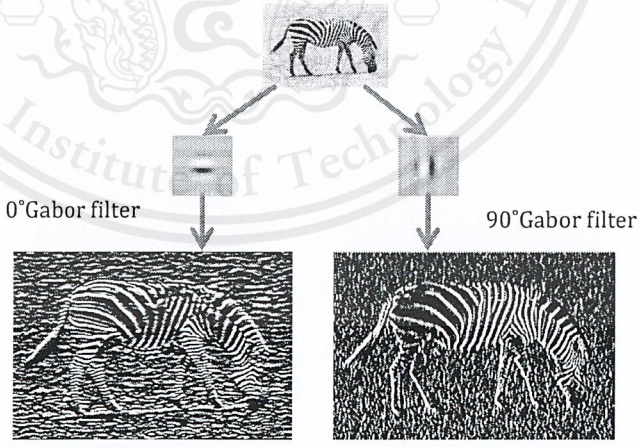

Figure 2.11 Gabor Filer

$$g(x,y) = exp\left\{-\frac{1}{2}\cdot\left[\frac{(x')^2}{\sigma_x^2}+\frac{(y')^2}{\sigma_y^2}\right]\right\}\cdot\cos(2\pi f_0 y') \quad (8)$$

$$x' = x\cdot\cos\theta + y\cdot\sin\theta \quad (9)$$

$$y' = -x \cdot \sin\theta + y \cdot \cos\theta \quad (10)$$

where, σ_x is the standard deviation of Gaussian equation in x-axis, σ_y is the standard deviation of Gaussian equation in y-axis, f_0 is a frequency of the image, θ is the local orientation estimation of the block (x, y).

2.3 k-Nearest Neighbors Algorithm (k-NN)

K-Nearest Neighbors Algorithm is a simple classifier algorithm that needs to be trained before being used mean (Supervised Classification) that the classifier predicts the type of sample data according to training data. A sample data is classified by finding k nearest training data in feature space (feature vector) and assigned to the most common number of the same type amongst its k closest neighbors. Euclidean distance is used to compute distance between vectors.

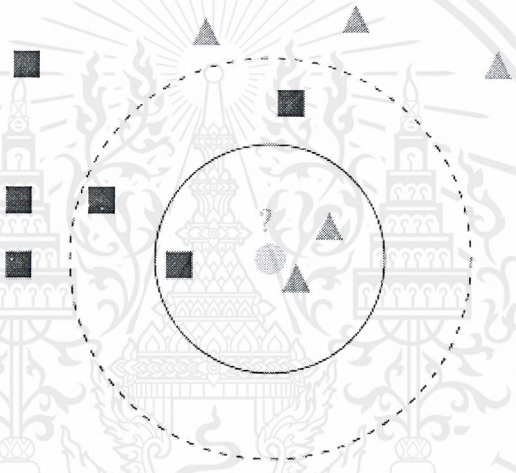

Figure 2.12 K-Nearest neighbors Classification[6]

2.4 NVIDIA GPU

2.4.1 Graphics Processing Units (GPU)

GPU-accelerated computing is the use of a graphics processing unit (GPU) together with a CPU to accelerate scientific, engineering, and enterprise applications. Discovered by NVIDIA, "*GPUs now power energy-efficient datacenters in government labs, universities, enterprises, and small-and-medium businesses around the world*" [10]

2.4.1.1 CPU Versus GPU

To compare CPU and GPU, we consider how they process tasks and their computational ability. A CPU consists of a few cores optimized for sequential serial processing that are able

---

[6] ***Image source*** Wikipedia the free encyclopedia. *http://en.wikipedia.org/wiki/File:KnnClassification.svg*. Example of k-nearest neighbour classificationnb. Accessed on Jan 20, 2014.

to perform computation more complex than GPU. While GPU consists of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously of lower level of complexity.

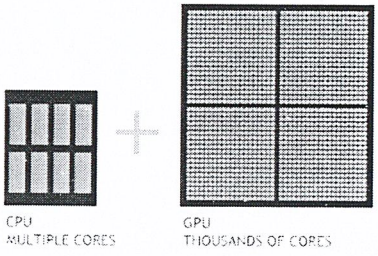

Figure 2.13 CPU versus GPU[7]

GPUs have thousands of cores to process parallel workloads efficiently

2.4.2 General-Purpose Computing on Graphics Processing Units

General-purpose computing on graphical processing units (GPGPU) is an emerging method for achieving high performance obtains on various computing problems such as database operations, neural networks, and physics-based simulations. The higher number of cores and the more efficient processing of complex mathematical calculations on GPUs in comparison with CPUs makes GPGPU an attracting new method for deploying algorithms. In an effort to reduce the runtime of protein-based search and retrieval algorithms.

GPGPU is the use of a GPU (graphics processing unit) together with a CPU to accelerate general-purpose scientific and engineering applications.

2.4.3 Architecture of NVIDIA GPU

Architecture of NVIDA GPU composed of Graphic card memory and GPU. There are 12 of Graphic card memories and a GPU

[7] ***Image source*** NVDIA about GPU computing. *http://www.nvidia.com/object/what-is-gpu-computing.html*. What is GPU accelerated computing. Accessed on Jan 18, 2014

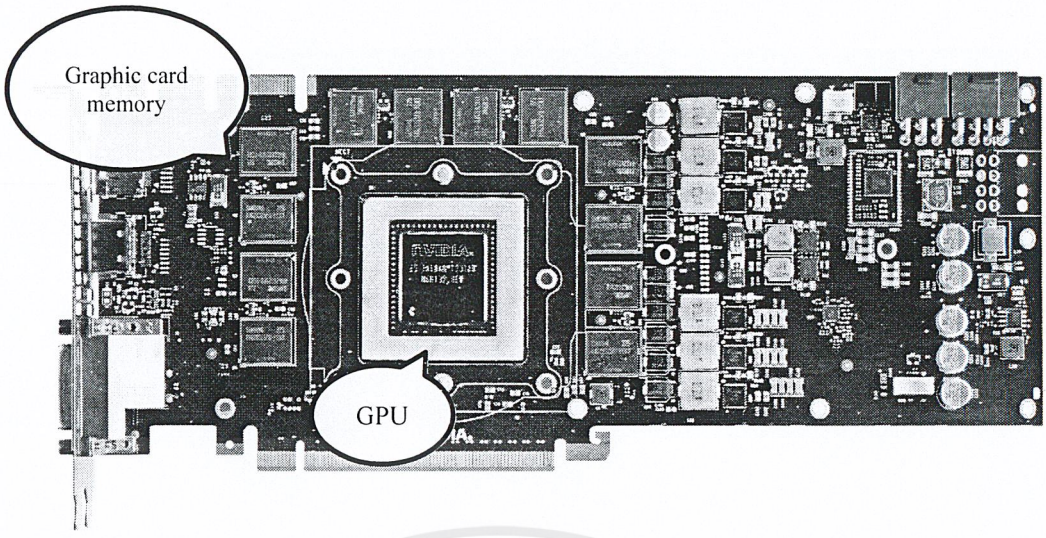

Figure 2.14 Architecture of NVIDIA GPU

2.4.3.1 The GK110 Kepler Architecture

Inside GPU, there is a full Kepler GK110 implementation includes 15 SMX units and six 64-bit memory controllers. Different products will use different configurations of GK110.

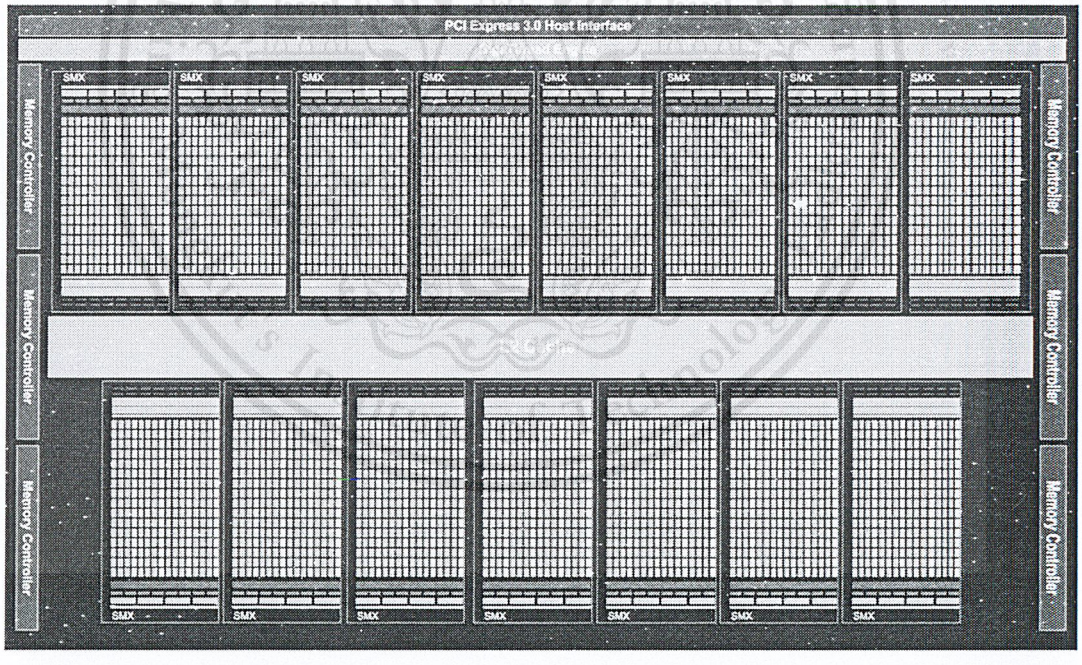

Figure 2.15 Kepler GK110 Full chip block diagram[8]

2.4.3.2 Streaming Multiprocessor (SMX) Architecture

The SMX unit mainly consists of two type of computing core. First, the single-precision cores are capable of computing mathematical variable up to floating point. Second the double-

[8] NVIDIA's Next Generation CUDATM Compute Architecture (2013). *Kepler GK110 the fastest, Most Efficient Architecture ever built v1.0.* GK110 Kepler Architecture: P. 6

precision core, which have fewer number that the single-precision, can computing up to double type number.

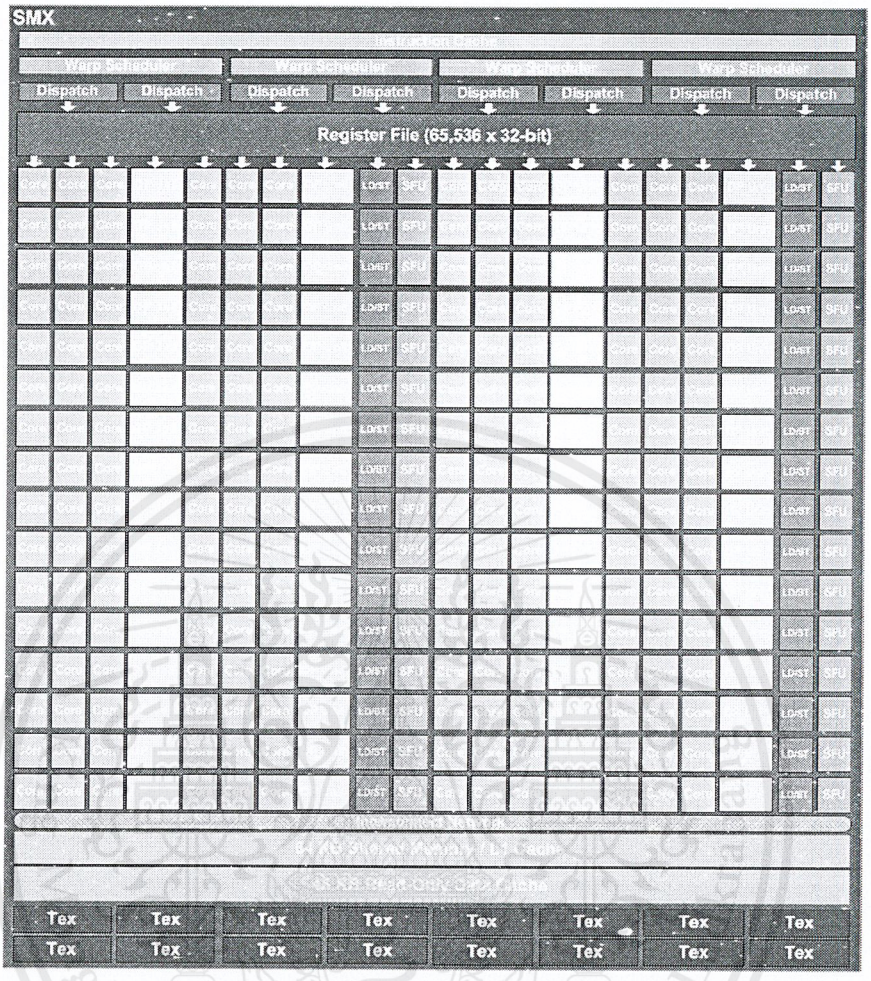

Figure 2.16 Streaming Multiprocessor (SMX)[9]

SMX: 192 single-precision CUDA cores, 64 double-precision units, 32 special function units (SFU), and 32 load/store units (LD/ST).

2.4.4 CUDA

CUDA (also known as Compute Unified Device Architecture) is a parallel computing platform and programming language which created by NVIDIA and implemented by the graphics processing units (GPUs). CUDA gives program developers direct access to the virtual instruction set and memory of the parallel computational elements in CUDA GPUs.

2.4.4.1 Heterogeneous Programing

CUDA is a serial program with parallel kernels, all in C/C++

---

[9] NVIDIA's Next Generation CUDATM Compute Architecture (2013). *Kepler GK110 the fastest, Most Efficient Architecture ever built v1.0.* Streaming Multiprocessor (SMX) Architecture: P. 8

- Serial programming language C/C++ code executes in a host thread such as CPU thread.
- Parallel kernels programming language C/C++ code executes in many device threads across multiple processing elements such as GPU threads.

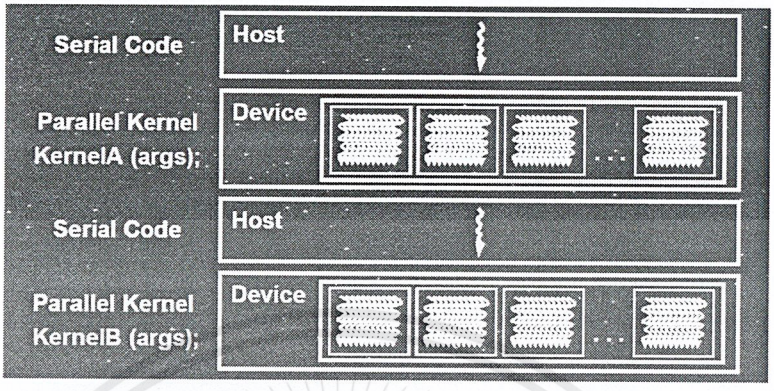

Figure 2.17 How Heterogeneous programing work [10]

2.4.4.2 CUDA Programming Model

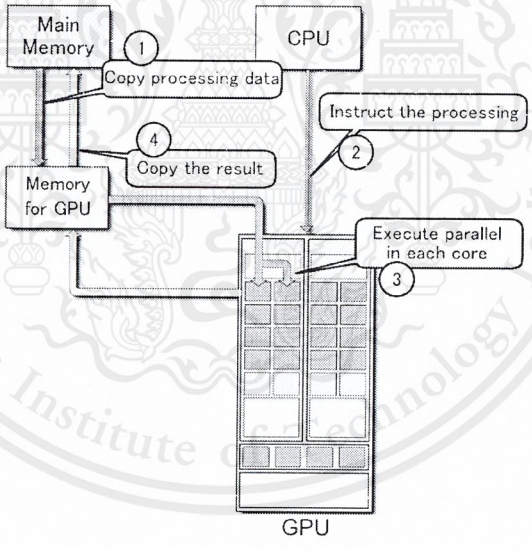

Figure 2.18 Example of CUDA processing flow[11]

Example of CUDA processing flow

The data is copied from main memory to GPU memory. CPU is going to instruct the process to GPU and then, GPU is going to execute parallel in each core. Finally, the result is copied from GPU memory to main memory.

---

[10] Cyril Zeller (2008). *NVIDIA Tutorial CUDA*. NVIDIA Developer Technology. Heterogeneous Programming: P. 17

[11] WIKIPEDIA The free Encyclopedia. CUDA processing flow. *http://en.wikipedia.org/wiki/File:CUDA_processing_flow.* Accessed on Fab 15, 2014.

2.4.4.3 Example of Code CUDA

CUP Program

```
void increment_cpu (float*a, float b, int N)
{
     for (int idx = 0; idx < N; idx++)
          a[idx] = a[idx] + b;
}

void main()
{
     .....
     increment_cpu(a, b, N);
}
```

CUDA Program

```
__global__ void increment_gpu (float*a, float b, int N)
{
     int idx = threadIdx.x;
     if (idx < N)
          a[idx] = a[idx] + b;
}

void main()
{
  ....
     increment_gpu<<< 1, N>>>(a, b, N);
}
```

[12]

Figure 2.19 Example syntax CUP program and CUDA program

[12] Cyril Zeller (2008). *NVIDIA Tutorial CUDA*. NVIDIA Developer Technology. Example, increment array element: P. 25

Chapter 3

Requirement Analysis

3.1 Requirements

3.1.1 Functional requirement

- The system allow the user store the image.
- The system allow the user to add tag and description to photo.
- The system can suggest tags for based on the content of the images.
- The user can manage tagging such as edit/remove tag with edit image feature.
- The system allow the user search for similar images to the given image.
- The system can search for images by providing keywords.
- The system can search for images with the given tag.
- The system allows the user to delete images.

3.1.2 Non-Functional requirement

- Easy-to-use graphic user interface (GUI).
- The system is implemented by object-oriented programming for extensibility and maintenance
- The system is always responsive.

3.2 Use Case Diagram

3.2.1 Use case diagram

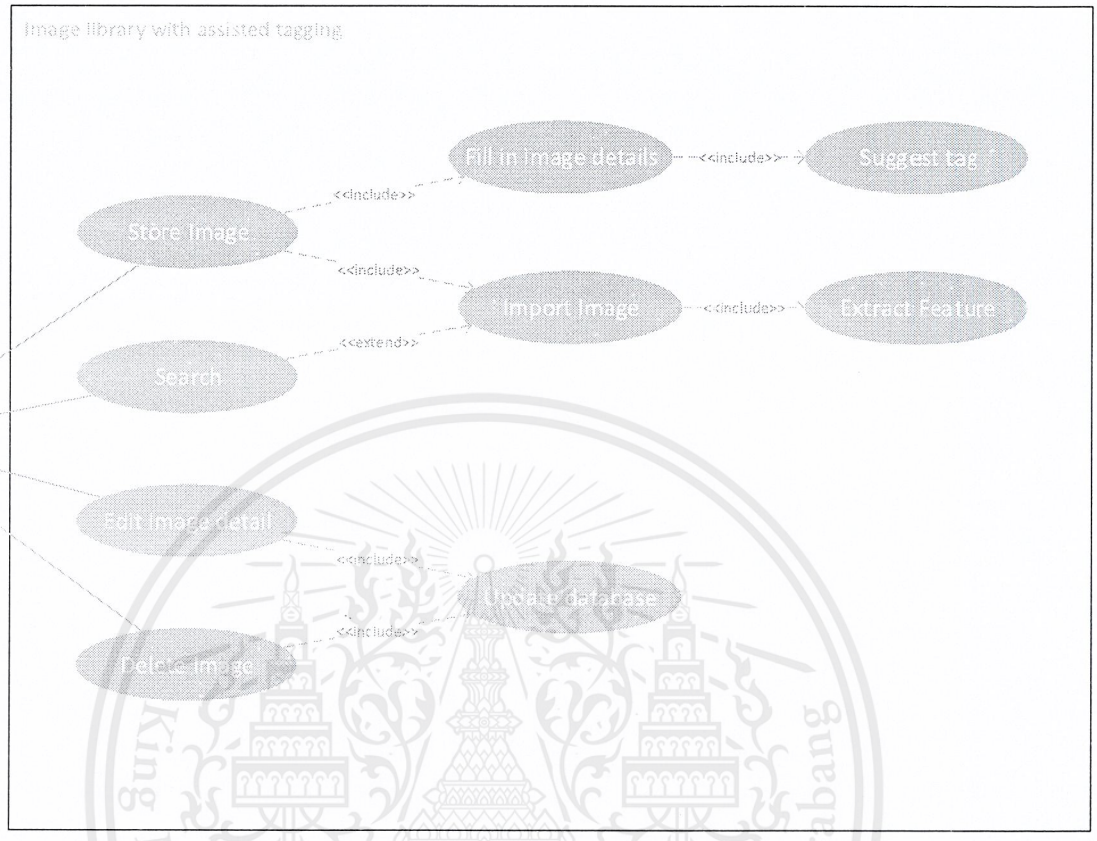

Figure 3.1 Use case diagram

3.2.2 User case description

Use case: Store a new image.

Actor: A user.

Goal: Store a new image (Information about tagging and description's image)

Overview: A user wants to add a new image into the program with "Import image" feature function which including; Image, tagging image, and description image. The system will suggest tag. The details are stored on separated database on the system.

Typical course of events:

Actor action		System response	
1	A user wants to add a new image.		
2	A user pushes "Import's button" in the program and select image. Then pushes open's button for being add.	3	The system shows new dialog for fill in information image, and preview a new image.
4	A user fills in tagging and description		

Actor action		System response	
	image in label box.		
5	A user pushes save's button for store correctly information and image into database.	6	The program is recording detail all image and image information to the data base and show new image with tagging image and description image on window home's display.

Alternative course:

Step 2 A user does not select image and pushes cancel's button, the program dialog for fill in information will not open.

Use case: Search

Actor: A user.

Goal: Search image by keyword and image

Overview: A user wants to search image by using "Search" feature function which including; searching by keyword, searching by image, and searching by tag.

Typical course of events:

Actor action		System response	
1	A user wants to search image.		
2	A user pushes "Search's button" in the program and select function for search.	3	The system shows new dialog search function which composed of three types for search such as search by keyword, search by image, and search by tag
4	Search by keyword. A user type keyword for search. Then, pushes search's button.	5	The program start search image by keyword which got from user. Then, show the results on window home's display.
6	Search by image. A user import image by using import's button. Then, pushes search's button.	7	The program start search image by image. Then, show the results on window home's display.
8	Search by tag. A user type tag for search. Then, pushes search's button.	9	The program start search image by tag. Then, show the results on window

Actor action		System response	
			home's display.

Alternative course:

Steps 5, 6, 8 The keyword/ image are not found. The program are going to show dialog massage to notice a user.

Use case: Edit image detail

Actor: A user.

Goal: Edit image detail such as tagging image, and description image.

Overview: A user wants to edit image detail by pushing edit's icon which is at bottom image. The program will show dialog for editing.

Typical course of events:

Actor action		System response	
1	A user wants to edit image detail.		
2	A user pushes "edit's icon" in the program.	3	The system shows dialog edit function for suppose editing.
4	A user revises image detail.		
5	A user pushes save's button for confirm image detail.	6	The program update image detail and back to window home.

Alternative course:

Step 5 A user pushes cancel' button, the program do not update and back to window home.

Use case: Delete image

Actor: A user.

Goal: Delete image and update information in database

Overview: A user wants to delete image by pushing delete's icon which is at bottom image. The program are going to show warming massage for confirmation.

Typical course of events:

Actor action		System response	
1	A user wants to delete image.		

Actor action		System response	
2	A user pushes "delete's icon" in the program.	3	The system shows dialog warming massage for confirmation.
4	A user pushes delete's button.	5	The program delete image and image detail and update database.

Alternative course:

Step 4 A user pushes cancel's button, the program do not delete image, image detail, and back to window home.

3.3 Activity Diagram

The activities diagram is represented how the program work. There are three diagrams for show activities in three functions in this program.

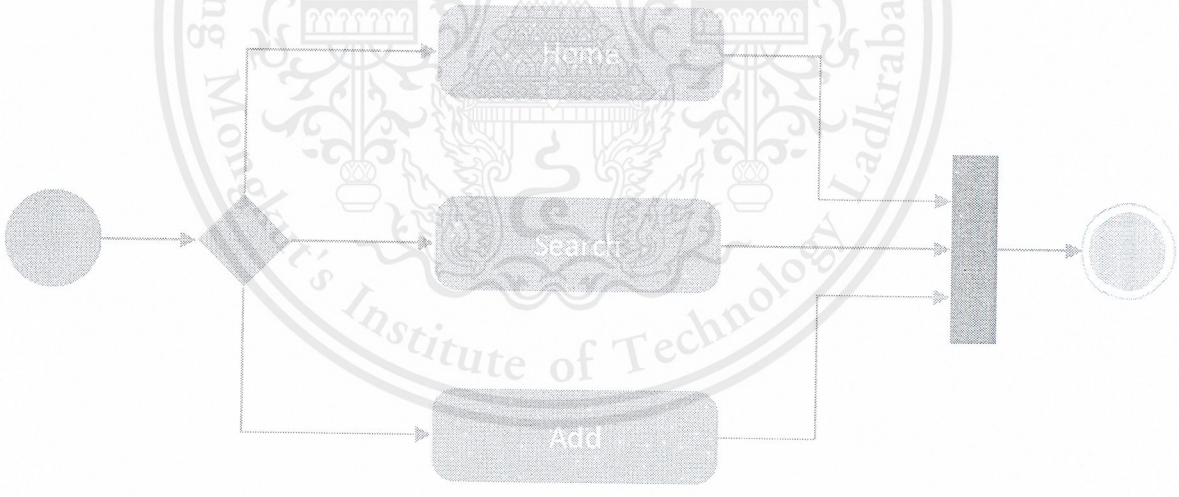

Figure 3.2 The activities diagram of the main function

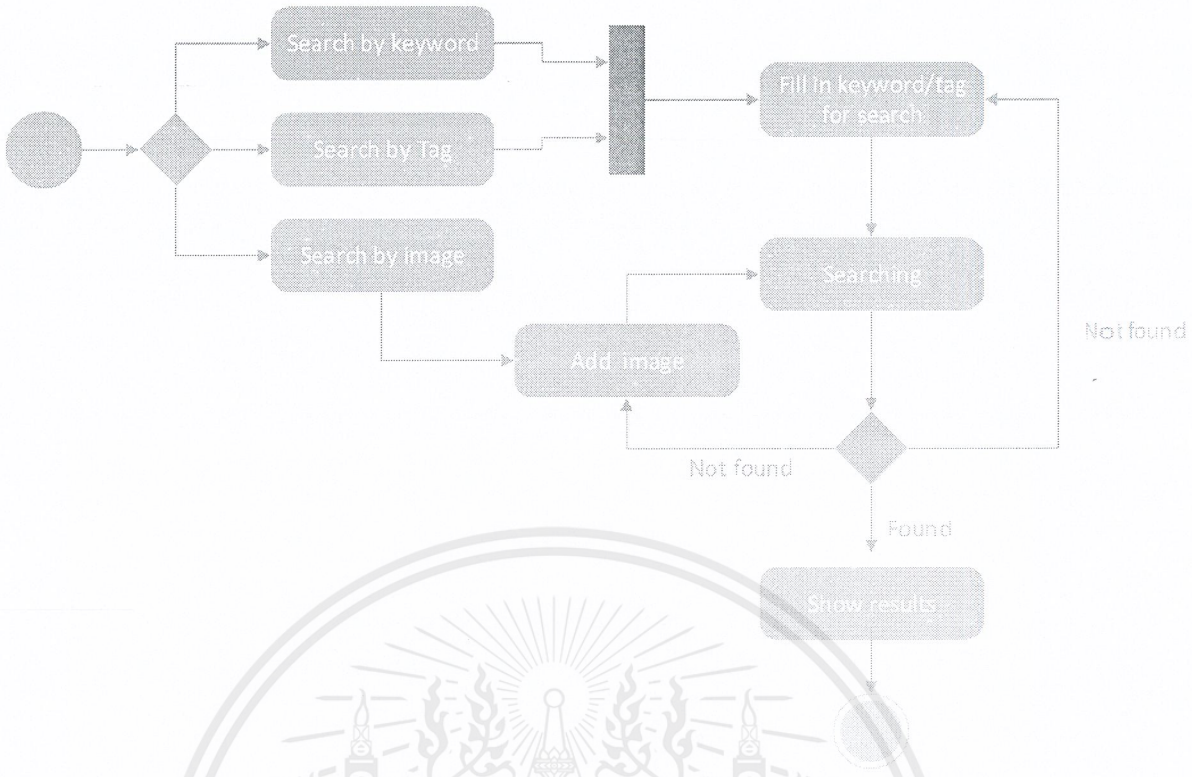

Figure 3.3 The activities diagram of the search function

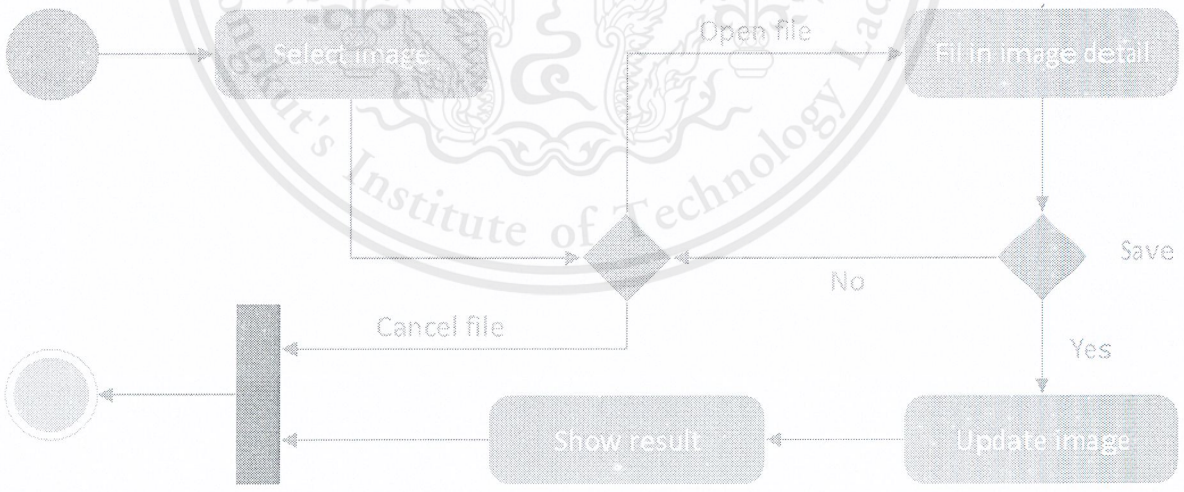

Figure 3.4 The activities diagram of the import image function

Chapter 4

Software Design

4.1 Overall Architecture

4.1.1 System architecture of HMAX model

The program are going to process as the figure 4.1. When the program input image from the user. The image is send to process in feature extractor. The HMAX algorithm is used in the feature and the simple cells (S1) which are implemented as Gabor filters can be executed in parallel and CUDA is implemented for develop of this model. The feature are going to send feature to classifiers node for classify vector feature. Finally, the identified object is result returned.

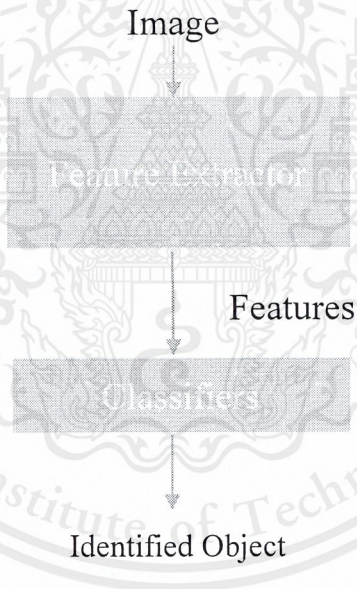

Figure 4.1 A general architecture of an object recognition system

4.1.2 System architecture of Parallel Implementation Bio-inspired object recognition.

There are two main software components: the application UI and the image library and application package. The user interacts with the program through the application UI. The UI then interacts with the image library and recognition package to perform image database functions and tag suggestion. This architecture of the software is shown in Figure 4.2

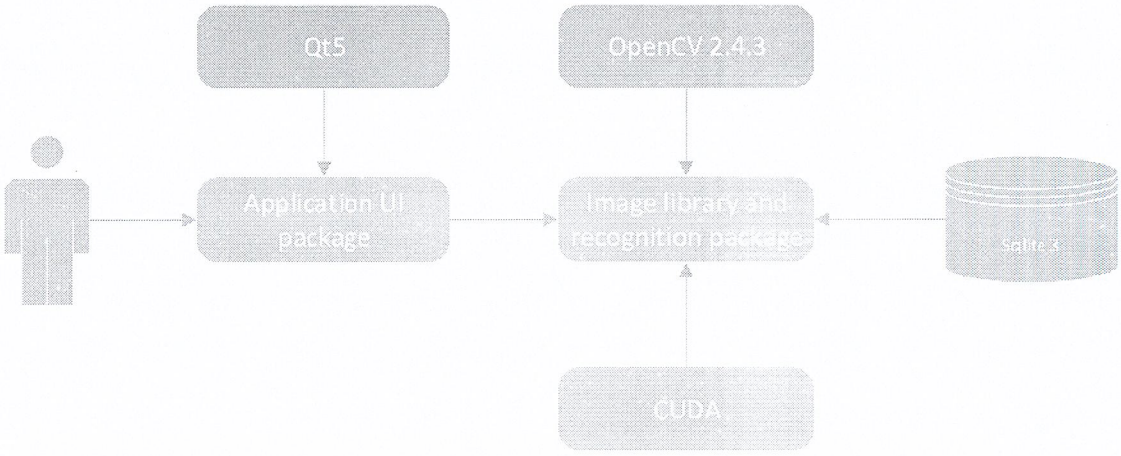

Figure 4.2 Software architecture

4.2 Class Diagram

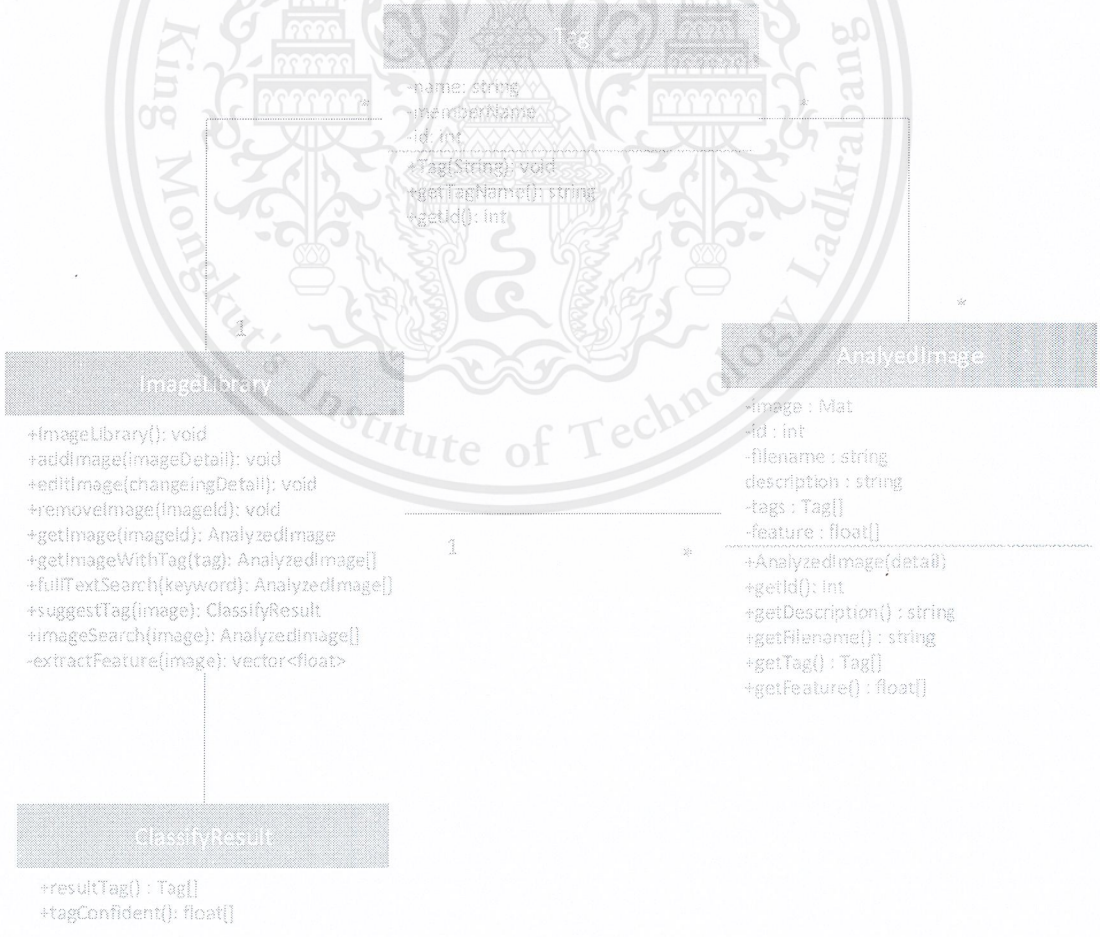

Figure 4.3 Class diagram

4.3 ER Diagram

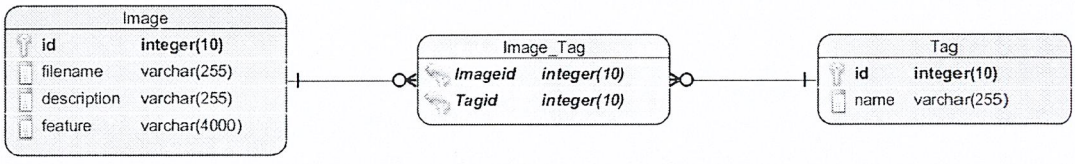

Figure 4.4 ER diagram

The database of our system obviously has only two main entities, Images and Tags. These two entities are many-to-many relationship, which must be broken down into a pair of one-to-many relationships. Because a relationship says an image can have several tags, and a tag can contain many images. If we do not eliminate the many-to-many relationship between Image and Tag, we cannot handle this relationship appropriately with two many-to-may SQL tables.

We eliminate a many-to-many relationship by creating an associative entity (cross-reference table) that is Image_Tag entity. In this case the logical primary key for Image_Tag is two foreign keys combined, one from Image's key and another from Tag's key. Now, for instance, the fact likes an image has three tags can be formed in this table by adding 3 rows with the same image's key and 3 different tag's keys, and likewise for a tag.

4.4 User Interface Design

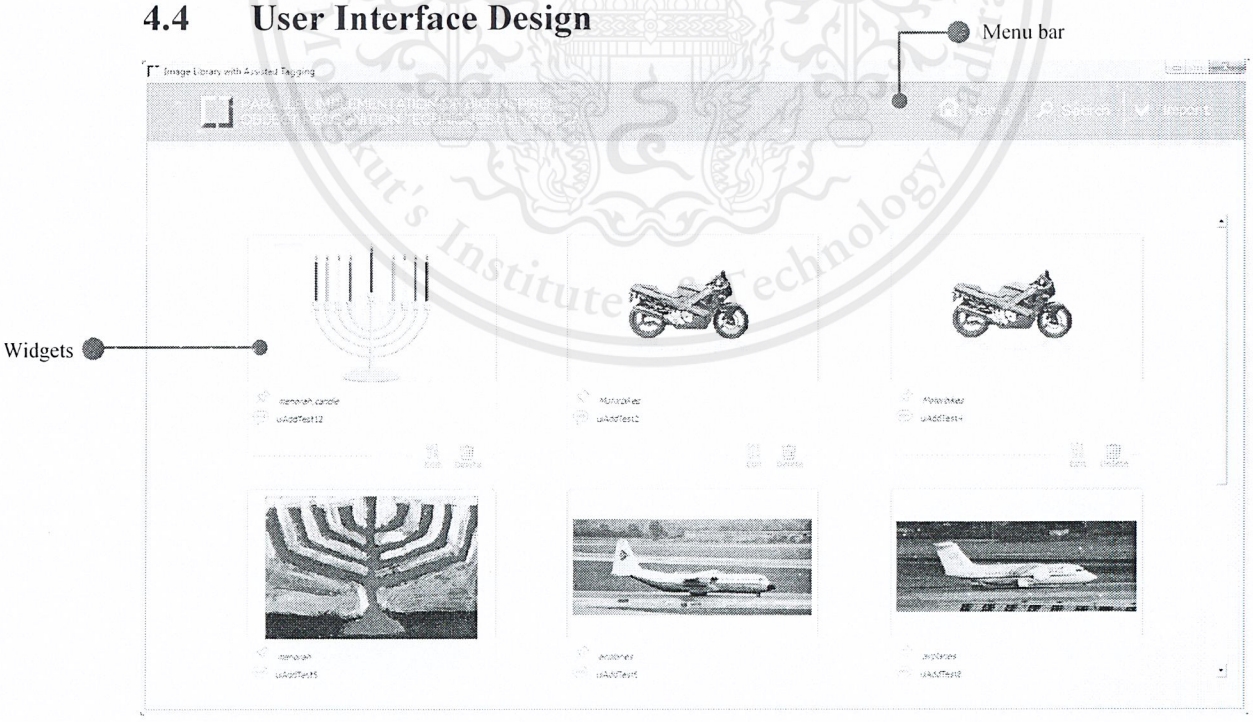

Figure 4.5 Main user interface

The main interface shows images with detail such as tag and description. There are seven functions including Home function, Search function, search by keyword, search by image, search by tag, import image function, edit function, and delete function.

Search Function composed of three functions such as search by keyword, search by image, and search by tag.

Import Function is function which support importation a new image into program. Moreover, a user can add image detail such as tag and description. Suggest tag feature is processed after a user chose image.

Home Function is function which refresh main user interface

Figure 4.6 Buttons in the main interface

4.4.1 Search Function

The user clicks search button for searching image. There are three sub-functions in this function which include search by keyword, search by image, and search by Tag

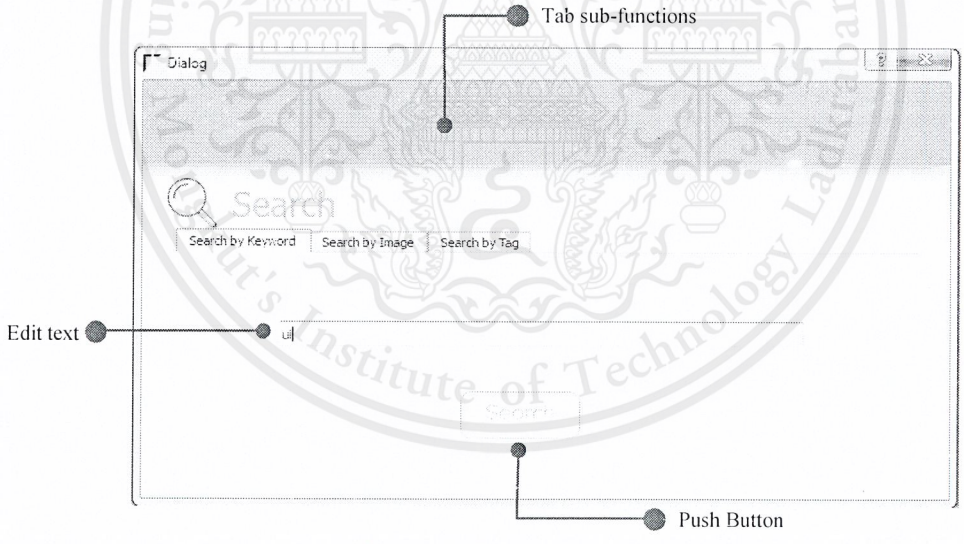

Figure 4.7 Search by Keyword

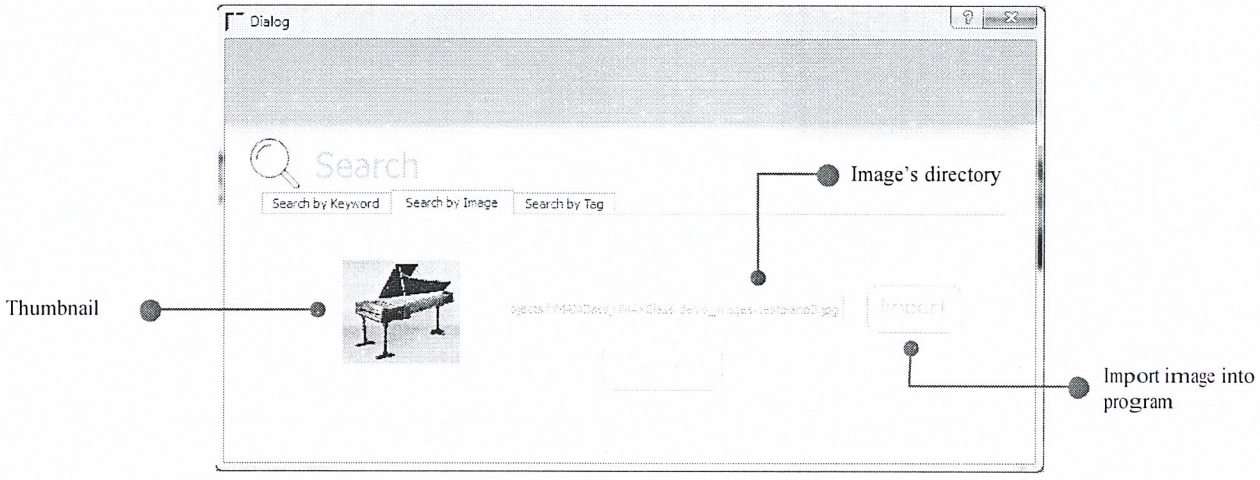

Figure 4.8 Search by Image

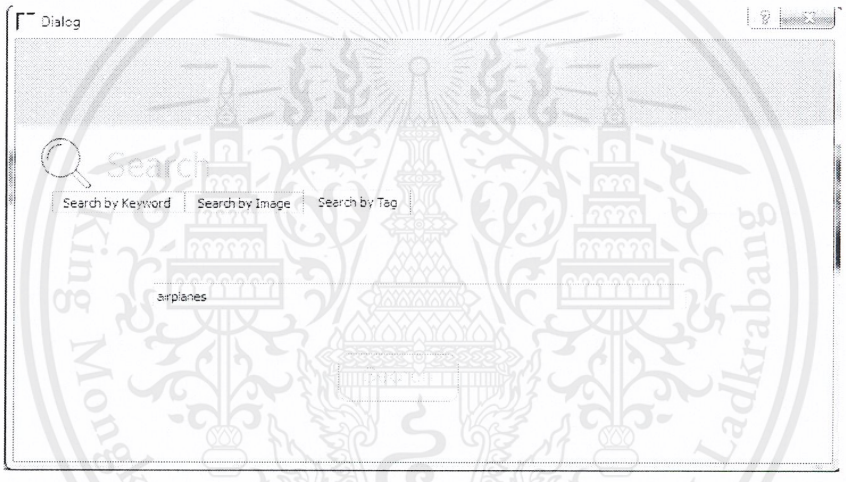

Figure 4.9 Search by Tag

4.4.2 Import Function

The user clicks import button for import image into the program then the program responds immediately by showing "open file" dialog. The image is selected by the user then clicks open's button (according to figure 4.10). Add description for importation a new image dialog after the user has already selected image. This dialog for the user inputs detail image which include tag, and description. Moreover, this function also has extra function such as Rotation image, and suggestion images (according to figure 4.11). The detail image have already saved into the system then the program responds by showing image in main menu. According to figure 4.12 the widget shows image, tag, and description and the user can edits or deletes image by click edit's button or delete's button.

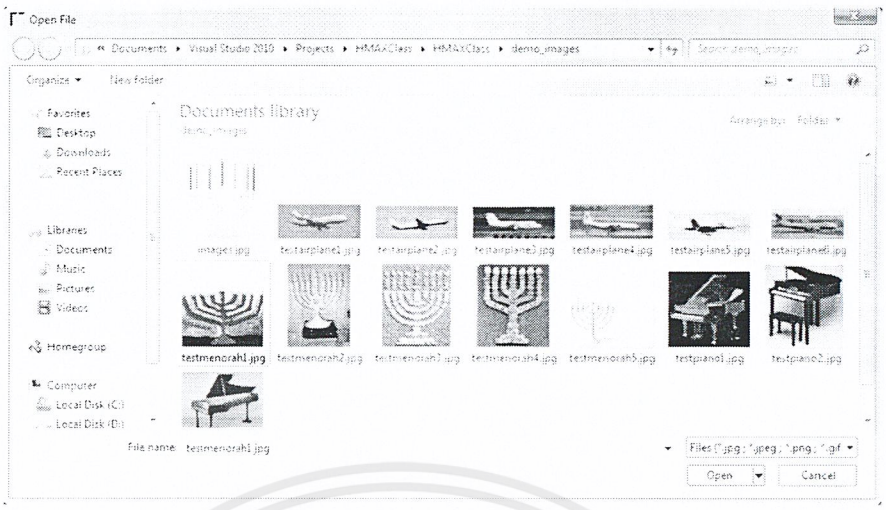

Figure 4.10 File dialog for choose image

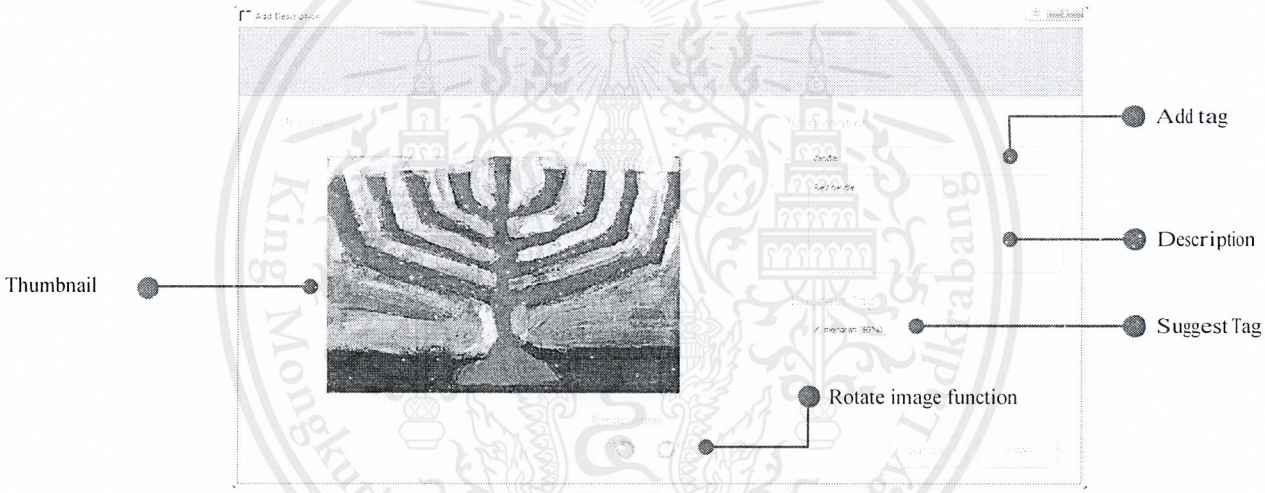

Figure 4.11 Add description for importation a new image

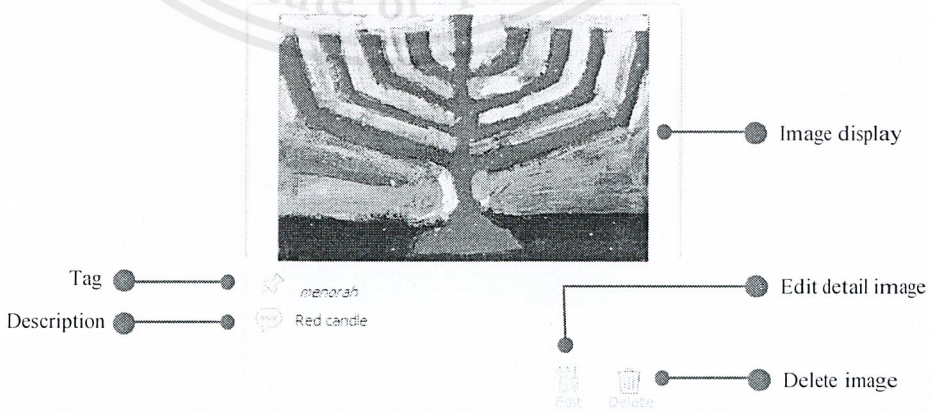

Figure 4.12 The widget shows image, tag, and description.

Chapter 5

Development

5.1 Software Development Process

Waterfall model is used to develop "Parallel Implementation of Biological inspired Object Recognition Techniques using CUDA". In a Waterfall model, every phase have to be clearly distinguished and after each phase is finished, it continues to do the next one.

5.2 Algorithms and Data Structure

5.2.1 Implementation of HMAX model with CUDA

Before an input image is taken in HMAX model to produce the feature vector, we need to transform it into grey-scale image because HMAX model considers only orientation in each region of image.

5.2.1.1 S1 Layer

First, we prepare Gabor filters of orientations 0°, 45°, 90°, and 135°, sizes from 7x7 to 29x29 pixels in odd steps (48 filters). These Filters are applied into an input image by convolution technique. As a Result, we get 48 images as outputs of this layer. Each output indicates peak values for each orientations and sizes according to Gabor filters.

Convolution technique is performed by aligning a center of filter to every pixel in the image and computes a result value by sum of product of corresponding pixels of filter and image.

Parallel Programming technique is used to perform convolution by assigning sum of product for one pixel to one core unit in CUDA. Therefore, all summing of product are compute simultaneously.

5.2.1.2 C1 Layer

In this step, the main work is to perform max operation of each type of orientation value across neighbor sizes (a band) in one pooling window (detailed described in section 2.1.2). All max resulted values in one band become an output image of C1 layer so that we get 16 images as result.

We assign max operation in one pooling window of a band to a CUDA's compute unit. Assigned units are executed at the same, then we get a resulted image of one band. We repeatedly run steps above for each band and each orientation.

5.2.1.3 S2 Layer

Non-overlapping window of 2x2 pixels of each C1 band images are groped to form a composition of 4 orientation bars. So, there are 256 result images of all possible 2x2 groupings.

When we call a parallel function for this layer, each core is given a task to compute a Gaussian function of a certain 2x2 composition and band simultaneously. Hence, we have to form a loop of 256 x 4 (bands) times to call that parallel function.

5.2.1.4 C2 Layer

C2 layer takes S2 result images of the same 4-orientation structure of all different bands into max operation to get the highest value of that structure, this value indicates whether the image has that structure or not, if so, how much.

Parallel algorithm of this layer runs partial max operation first by every core simultaneously then, takes those partial max values to do max operation again but sequentially by CPU.

From steps above, a result of C2 Layer is a 256-element feature vector of an input image.

5.2.2 Implementation of Quicksort with CUDA

Our Image Library application has a feature searching by image, the process is as followed.

1. Importing an input image.
2. Extract a feature vector of input image by HMAX model.
3. Compute Euclidean distance from input vector to all vectors of image in library.
4. Sort the list of all distances.
5. Return a group of nearest images.

Sort algorithm is essential in this process, so we decide to use Quicksort implemented in parallel fashion.

To implement Quicksort in parallel, we use a sophisticated feature called Dynamic Parallelism. This feature makes parallel function recursively call itself **more than once** at a time by open another computing streams to run recursive calls simultaneously. Our Parallel quicksort is perform by following steps.

1. Execute parallel quicksort by deploying one core.
2. Do partitioning by selecting pivot value and rearrange elements lower than pivot in the left side of pivot then greater than pivot in the right.
3. Recursively call parallel quicksort functions for the left list and right list by open one stream for each to run at the same time.
4. If the depth of recursive call is greater than defined depth threshold or the number elements to sort further lower than size threshold, then sort by linear sorting.

5.3 Implementation

5.3.1 User Interface tools

- Qt 5.0.2 [13]

  It is a cross-platform application framework which is used for implementation and developing application with a graphical user interface (GUI)

- Visual Studio Add-in 1.2.1 for Qt5

5.3.2 Implementation and Library

- OpenCV 2.4.5 [14]

  It is a library of programming functions which mainly purpose to real-time computer vision.

- Microsoft Visual Studio 2010 [15]

  Visual Studio is a program which provides us many tools and building many kinds of application because this program are using Qt5, OpenCV and CUDA to develop an application, C++ language program is one way needed to work with Qt5, OpenCV and CUDA which Visual Studio allows to write a program in C++ language program.

- CUDA

Chapter 6

Experiments and Discussion

6.1 Experiment 1: Speedup of CUDA

6.1.1 Objective

Determine the speedup of CUDA code for image normalization

6.1.2 Setup and Procedure

- The specification of this experiment as following:
  - CPU: Intel(R) Core™ i5-3470 CPU @ 3.20 GHz 3.20 GHz
  - GPU : NVIDIA GeForce GTX 460 with 336 cores
- Prepared the picture three sizes which including: 1024x1024, 4096x4096, and 8192x8192 and three threads for experiment.
- Run the serial program (CPU) and parallel program (GPU) with all sizes and all number of threads.

6.1.3 Result

The following table shows the experiment result.

Image Size (Pixel)	CPU Time	GPU Speedup					
		64	128	1024	4096	8192	16384
1024	0.0326	0.301852	0.427412	0.75463	0.829517	0.842377	0.842377
4096	0.4792	0.436589	0.739278	2.773148	3.870759	3.986689	4.067912
8192	1.901	0.451694	0.922099	2.971241	4.091692	4.216948	4.279604

Table 6.1 Shows the experiment result

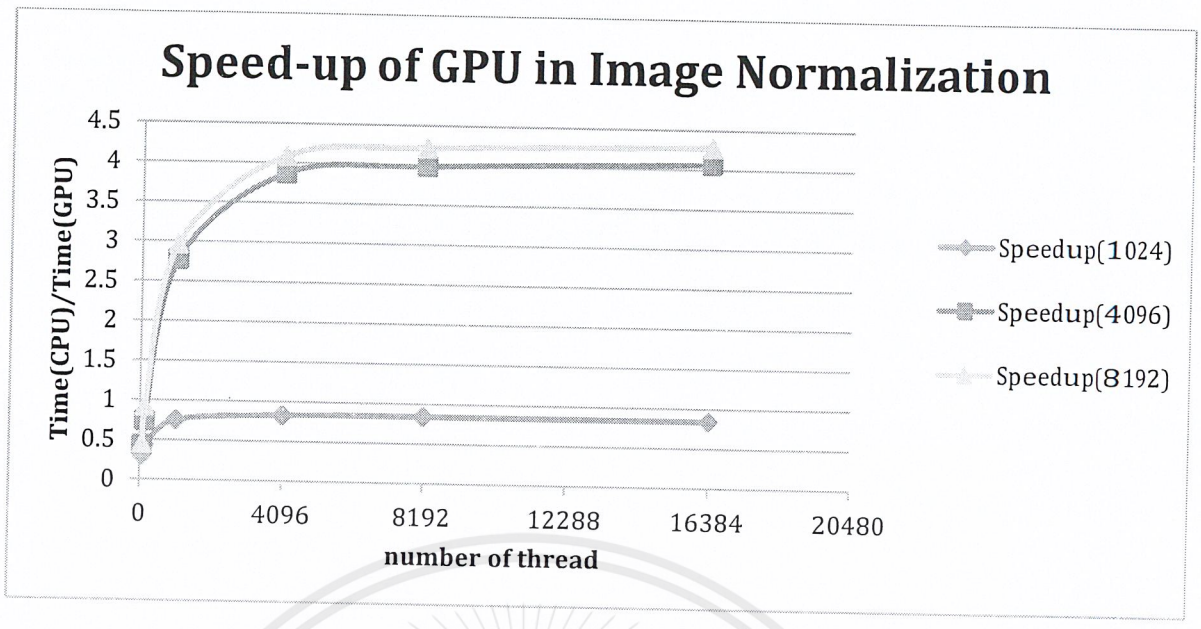

Figure 6.1 Graph show speedup of GPU with respect to CPU

6.1.4 Discussion

As the figure 6.1 graph shows speedup of GPU with respect to CPU. The less thread you have, the slower GPU processes and the faster CPU processes on the contrary, if the much thread you have, the faster GPU process and the slower CPU process.

6.2 Experiment 2: Speedup of CUDA for HMAX algorithm

6.2.1 Objective

Determine the speedup of CUDA code for HMAX algorithm

6.2.2 Setup and Procedure

- The specification of this experiment as following:
  - CPU: Intel(R) Core™ i5-3470 CPU @ 3.20 GHz 3.20 GHz
  - GPU : NVIDIA GeForce GTX TITAN
- Prepare 5 pictures of size, 64x64, 128x128, 256x256, 512x512, and 1024x1024 pixels.
- Run the serial program (CPU) and the parallel program (GPU) of HMAX algorithm with all prepares images of all sizes and collect time of execution.

6.2.3 Result

Image Size(Pixel)	Sequential (Sec)	Parallel (Sec)	Speedup (S/P)
64 x 64	20.142	0.389	51.77892031
128 x 128	86.956	0.465	187.0021505
256 x 256	366.876	1.017	360.7433628
512 x 512	1502.6	3.101	484.5533699
1024 x 1024	6065.496	11.315	536.0579761

Table 6.2 Shows the experiment result

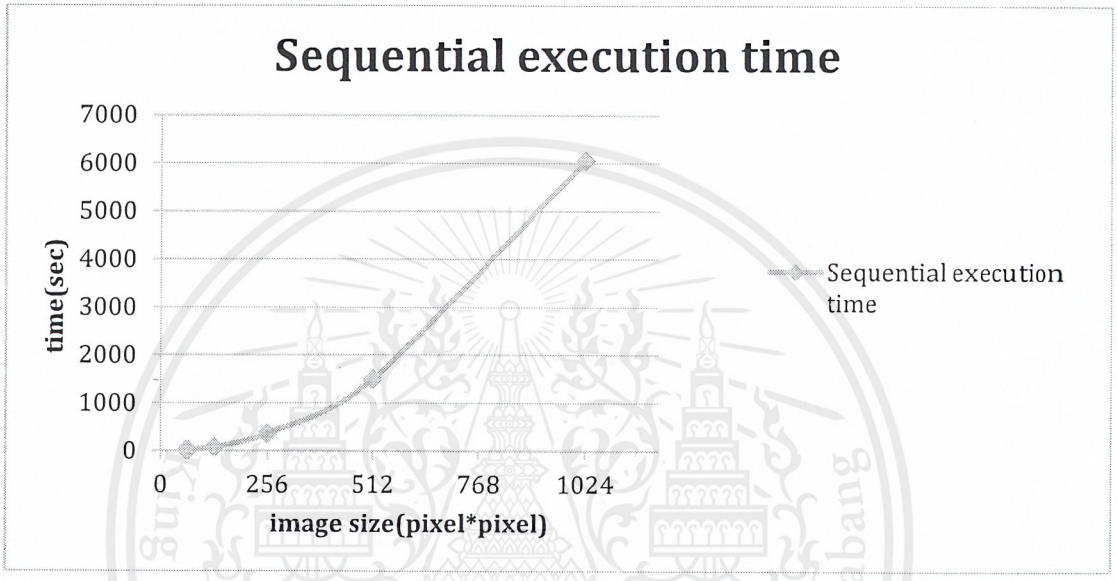

Figure 6.2 Sequential execution time

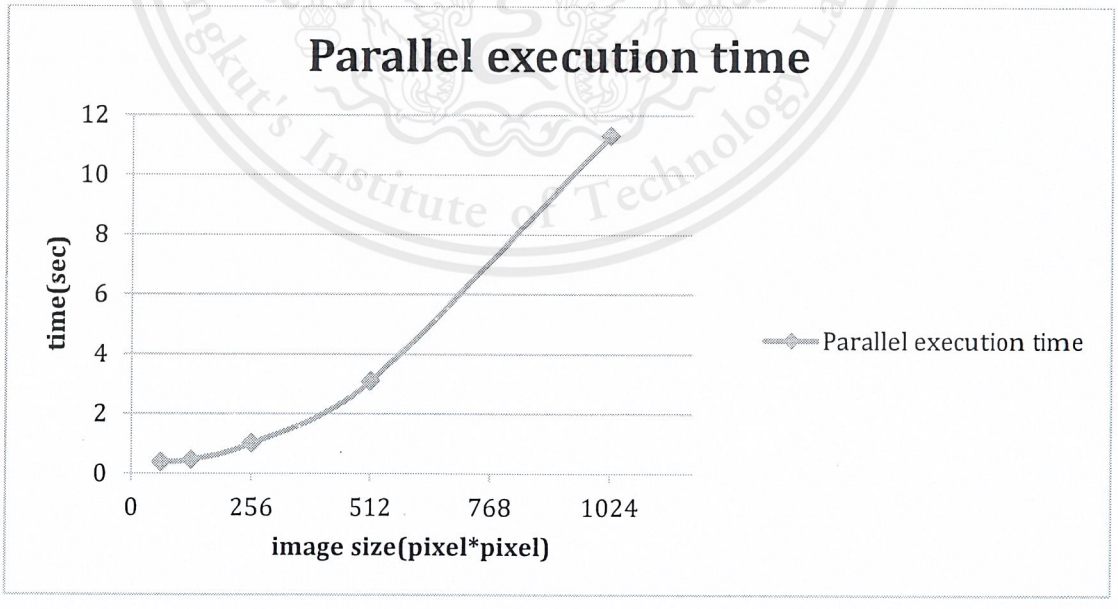

Figure 6.3 Parallel execution time

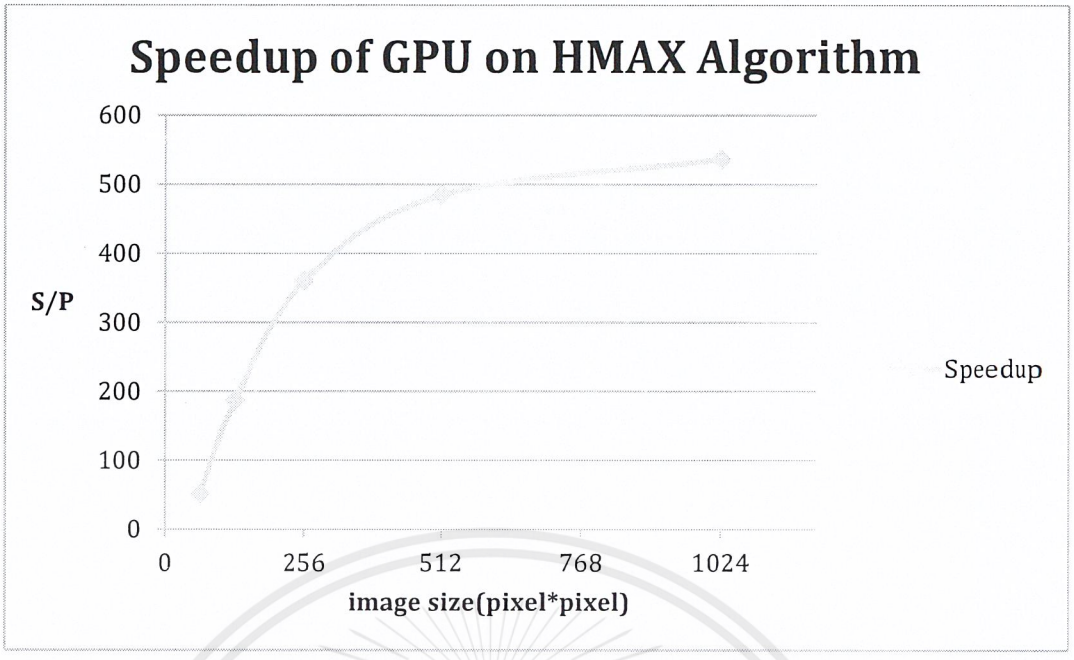

Figure 6.4 Speedup of GPU on HMAX Algorithm

6.2.4 Discussion

The result in 6.2.3 section indicates that the increased growth of sequential and parallel algorithm based on the size of images look almost the same but hugely different in the number of execution time, so that the speedup is very high. From a speedup graph, it shows that the speedup of parallel algorithm grows dramatically from 64x64 to 512x512 images and grows a little bit from this point to 1024x1024 image and eventually remains steady for much bigger images.

6.3 Experiment 3: Accuracy of k-NN version 1

6.3.1 Objective

Determine the accuracy of k-NN classification of HMAX vectors.

6.3.2 Setup and Procedure

- The specification of this experiment is the same as in 6.2.2 section.
- Prepare 5 types of images, 30 images per each type, for example, pianos, airplanes, and etc.
- Run the program that feeds all images to HMAX to get corresponding feature vectors, and picks 20 of each type randomly as training data set for k-NN classification and the rest are sample set to be classified with k value of 5 and 7.

- The program does previous step 10 times and collects the number of failure classification.

6.3.3 Result

k value	5	7
Max Accuracy	80%	76%
Min Accuracy	62%	62%
Avg. Accuracy	70%	68.4%

6.3.4 Discussion

The result table in 6.3.3 section indicates that k-NN with vectors generated by HMAX algorithm is low accurate classification with average results are greater than 50%. While the result looks accurate, it still cannot be trusted. Because we only use 5 types of object in images, a vector from HMAX is large with 256 element but we have only 150 images to do the experiment. If we consider that just one element in vector varying the vectors, we should have more than 256 images as a sample set. We need to another image source providing enough number of types and images to do more experiment for precise analysis of accuracy.

6.4 Experiment 4: Accuracy of k-NN version 2

6.4.1 Objective

Determine the accuracy of k-NN classification of HMAX vectors by adding more images.

6.4.2 Setup and Procedure

- The specification of this experiment is the same as in7.2.2 section.
- Prepare 14 types of images, 30 images per each type, for example, pianos, airplanes, and etc.
- Run the program that feeds all images to HMAX to get corresponding feature vectors, and picks 20 of each type randomly as training data set for k-NN classification and the rest are sample set to be classified with k value of 5, 7, and 15.
- The program does previous step 20 times and collects the number of failure classification.

6.4.3 Result

k value	5	7	15
Max Accuracy	40%	37.857%	37.857%
Min Accuracy	26.4286%	26.4286%	27.1428%
Avg. Accuracy	30.6%	31.536%	31.572%

6.4.4 Discussion

The result table in 6.4.3 section indicates that k-NN with vectors generated by HMAX algorithm is low accurate classification with average results are around 30%. A result appears under expected amount. We already checked the program if it works properly as it should be and we did not any false or bug in algorithm. The algorithm may improve by adding segmentation part to the process to remove background from interested objects. However, it might the limited ability of algorithm itself, because there are so many recognition algorithms in this field of research and some algorithms are not successful by the idea and process itself.

Chapter 7

Conclusion

7.1 Summary

We studied how biological vision works, the HMAX algorithm, and parallel programming in CUDA. We implemented the HMAX algorithm in CUDA and performed experiments to evaluate its efficiency and we implemented an image library application in Qt5 to demonstrate the usefulness of the implemented algorithm.

7.2 Problems and Obstacles

The HMAX algorithm works well only on an image which contains one object. The CUDA code, which was written in C++, was prone to errors and difficult to debug and combining OpenCV and CUDA in one project is quite complicated.

7.3 Further Work

- Improve the HMAX algorithm for images which contain multiple objects.
- Choose an alternative classification algorithm in place of the K-NN algorithm to classify images.
- Experiment the implementation more extensively.
- Speed up the performance of the algorithm by optimizing the use of threads in CUDA.
- Add more features to the image library application.

Bibliography

[1] Hubel, D., Wiesel, T.: **Receptive fields, binocular interaction and functional architecture in the cat's visual cortex**. J. Physiol. (Lond.) 160, 106–154 (1962)

[2] Serre, T., Wolf, L., Poggio, T.: **Object recognition with features inspired by visual Cortex**. In: Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). IEEE Computer Society Press, San Diego (2005).

[3] Aravind, I., Chandra, C., Guruprasad, M., Sarathi Dev, P., Samuel, R.D.S.: **Numerical approaches in principal component analysis for face recognition using eigenimages**. In: IEEE International Conference on Industrial Technology, IEEE ICIT 2002 (2002).

[4] Ji, Y., Chang, K.H., Hung., C.-c.: **Efficient edge detection and object segmentation using Gabor filters.** In: ACM Southeast Regional Conference, pp. 454–459 (2004).

[5] Riesenhuber, M. & Poggio, T. **Hierarchical Models of Object Recognition in Cortex**. *Nature Neuroscience* **2**: 1019-1025. (November 1999).

[6] Riesenhuber, M., Poggio, T.: **Models of object recognition**. Nature neuroscience supplement 3 (November 2000).

[7] Kreiman, G., Serre, T., Poggio, T.: **On the limits of feed-forward processing in visual object recognition**. VSS (May 2007), http://www.cosyne.org/c/images/6/65/Cosyne-poster-I-52.pdf

[8] Tanaka, K.: **Mechanisms of visual object recognition: monkey and human studies**. Curr. Opin. Neurobiol. 7, 523–529 (1997).

[9] Wikipedia, the free encyclopedia. **CUDA.** Available: http://en.wikipedia.org/wiki/

[10] NVIDIA. **What is GPU computing?** Available: http://www.nvidia.com/object/what-is-gpu-computing.html.

[11] Todd Sullivan et al. **General-purpose computing on graphics processing units: GPU processing of protein structure comparisons**. The University of Missouri-Columbia Medical Biological Digital Library Research Lab (MBDLRL) (2007).

[12] Maximilian Riesenhuber, et al. **HMAX.** The Laboratory for Computational Cognitive Neuroscience. Washington, DC 20007. Available: http://maxlab.neuro.georgetown.edu/hmax

[13] Jasmin B. and Mark S., **"C++ GUI programming with Qt 4"**, Prentice Hall, 2006.

[14] Robert Laganiere, **"OpenCV 2 computer vision application programming cookbook"**, Packt Publishing Ltd., 2011.

[15] Wikipedia, the free encyclopedia. **Microsoft Visual Studio.** Available: http://en.wikipedia.org/wiki/ Microsoft_Visual_Studio.

[16] Jim Mutch, Ulf Knoblich, and Tomaso Poggio. **CNS: a GPU-based framework for simulating cortically-organized networks**. MIT-CSAIL-TR-2010-013 / CBCL-286, Massachusetts Institute of Technology, Cambridge, MA, February 26, 2010.