Misi

The Challenge of Motion Analysis

Synchronizing OpenGL and OpenCV

in Real-time to Allow Processing of High Resolution Stream

By Ela Shapira and Tamir Fridman - Managing Directors

www.misi-tech.com

Misi Tech INC. – White paper

December 2016

Table of content

Introduction	**3**
Purpose	3
Role of Data Synchronization	3
Challenges of real-time computer vision	4
Scope of OpenCV vs. OpenGL	**5**
What is object analysis?	5
Understanding GPU Architecture	5
The Logic of CPU	6
Now Let’s Synchronize	6
Using GPU Surface for Data Stream Parallel processing	**8**
Streaming multi-resolution	8
Managing the Graphical Object	8
Integrating OpenCV with OpenGL	9
Now Let’s Detect	9
Code Example	10
Summary	**12**

Introduction

Purpose

The purpose of this paper is to describe new ideas in parallel processing of GPU and CPU as a key element for High-resolution object and motion analysis. These ideas are based upon review of the physics of the problem and an analysis of applicable technological approaches.

Role of Data Synchronization

Modern computing architecture defines parallel processing as a key element for graphics operation systems, the O/S uses parallel processors (GPU) to display windows, images, videos, animations and complex 3D games. With the traditional serial processing (CPU) which runs all the software logic, the modern computer has two processing units running side-by-side with different clock rates.

Data synchronization is a must-take approach when trying process high-resolution video stream in real-time. Synchronizing the logic process (CPU) with the graphical process (GPU) will allow real-time object and motion analysis for “heavy duty” streams (4K/60FPS) when defining the synchronization pipeline to be efficient and simple enough to run at such rates.

Introduction

Challenges of real-time computer vision

Computer Vision (CV) is about understanding what is in a given image (frame) or a stream (set of frames) by applying image filters (GPU shaders) and image processing algorithms (CPU programs - shape detection and motion detection).

Today’s availability of High Resolution video camera in every handheld device, hinders real-time object detection technologies projects, while the limited memory creates the need to reduce as much as possible the amount of data sent to the CPU for analysis and actually extract from the frame using GPU shading techniques the relevant data for the CPU to analyze.

Scope of OpenCV vs. OpenGL

What is object analysis?

Object analysis is an algorithm which allows detection of objects (geometric shapes) and movements (moving objects) in a high-resolution video stream while implementing logic into the stream. Finding objects in a frame and comparing it to the shapes from the previous frames allows us to detect motion and analyze the objects movement (direction and velocity).

Different shapes are uses to detect familiar objects like faces (circles and lines abiding to certain logic), hands, gestures and so on.

Understanding GPU architecture

The GPU or Graphical Processing Unit, is a processor designed for parallel implementation of graphical definitions in high-resolution. Logic used in GPU is completely different from the “classical” logic of a sequential code. The GPU runs the program on every pixel concurrently to achieve parallel mathematical manipulation for each pixel. For example, the program “**set transparency to 0.5**” runs on all the image pixels at the same time and the image becomes half transparent in a single process [O(1)], just for the comparison the CPU would required a loop to run on all the pixels [O(n)] and a 4K frame would require 8,294,400 iterations to process the frame!

Scope of OpenCV vs. OpenGL

The logic of CPU

CPU understands logic, and logic is built from flows. In every flow we can define data processing, computations and logical statements that can activate other flows. A flow efficiency is being measured by the number of the CPU iterations that will be required to complete the flow. a complex flow would cause the CPU to slow down all the system and can cause unexpected results.

Now let’s synchronize

Our main goal is to synchronize a high-speed parallel graphic processor with serial process CPU, the main process is to prepare the frame using the GPU, send it to CPU for analysis and send the result back to the GPU for rendering the result.

Scope of OpenCV vs. OpenGL

For example the following timeline define the SYNC flow for 30 FPS stream:

```
          GPU                      CPU
———————————————— | ——————————————
 1. Prepare Frame |
                       <== SYNC ==>
2. Process Output |  2. Analyze Frame
                             <== SYNC ==>
3. Render Output |
```

Since every frame maximum process time is ~33ms for the entire flow and the flow defines two SYNC operations each SYNC payload should be very small in order to assure maximum process time.

Using GPU surface for data stream parallel processing

Streaming multi-resolution

In order to overcome the main speed issue we must stream different resolution and outputs to CPU and GPU, while the GPU is processing high-resolution image and implementing all graphic manipulations, the CPU gets in parallel a filtered low-resolution image witch enables both processors to perform simultaneously.

Managing the graphical object

Managing a graphical object on the screen (i.e. games) projects need to follow the behavior of an object when moving between frames. For example if we run a shape detection algorithm to find square shapes and to draw virtual objects on the square shape, it requires to "move" the virtual object (change its X,Y position) to the exact position of the square in the real image for every frame. As soon as the square disappears from the image, the virtual object should also disappear. If the image contains more than one square, a virtual object should be attached to every square. This makes the algorithm more complex, since every square should be compared to the previous frame squares to find the one that most likely was the same one.

Using GPU surface for data stream parallel processing

Integrating OpenCV with OpenGL

The integration of OpenCV and OpenGL requires building a stable regulated pipe-line between the 2 projects that can transfer stable filtered data to the CPU and return graphical instructions to the GPU. In the last example (detecting squares and attaching virtual objects to the squares) we've been required to run shape detection algorithm. This algorithm uses contours and key-points detection by analyzing the contour path to detect if it is a square. If the given image will be unfiltered, too many key-points and contour will be detected and the process time will be very long. To ease this process, the GPU prepares the image using several filters (like Sobel (X&Y), Canny, thresholding, blur etc.') thus ensuring that image for the CPU will be "clean" and ready to process.

Now let's detect

Now lets detect a table and place a vase on it, in a way that it could be viewed through the camera from every angle. To do this, we need a image filtering to emphasize edges and a shape detection algorithm to determine if the given shape is a table (by the simple definition that each table has a big closed shape (square/round/oval) on 3 or more vertical lines (the table legs)). The algorithm should find the right angle of the shape and should return to the GPU a single point (X,Y) and rotation angle to render the virtual vase.

Using GPU surface for data stream parallel processing

Code Example

```
//pseudo code

MisiGL()
{
        on (GL Frame)
        {
                get frame from input
                create filtered image for MisiCV
                SYNC
                crearte filtered output
                SYNC
                render virtual objects on MisiCV points
        }
}
```

Using GPU surface for data stream parallel processing

Code Example

```
MisiCV()
{
    while (GL on)
    {
        SYNC
        find shapes in MisiGL image
        find table in shapes
        if(has table)
        {
            compare with previous frame table
            if(match)
                move vase object to new position
            else
                add new vase in position
        }
        SYNC
    }
}
```

Summary

The paper presented a rationalization for synchronization of parallel (GPU) and serial (CPU) image processing, in order to enable complex processing of high resolution image in real-time, when process includes both image manipulation and object detection.

Thank you

Misi Tech Inc.

MisiCam video-camera and gaming console, download:

Download on the App Store

ANDROID APP ON Google play

Watch our demo on YouTube

Misi Tech INC.
340 S LEMON AVE #2396N , WALNUT, CA, 91789
www.misi-tech.com info@misi-tech.com

Misi