MFC를 이용한 25가지 색상 양자화 모듈의 CUDA 구현

CUDA Implementation of 25 Color Quantization Module with MFC Dll Module

Ryu Dong-Sung
류동성

부산대학교 컴퓨터공학과
dsryu99@pusan.ac.kr

ABSTRACT

최근 많은 수의 사진이 촬영됨에 따라서, 색상 유사도와 시간을 이용한 사진 관리에 관한 연구가 많이 진행되고 있다. 이 때, 색상 유사도를 평가하기 위한 방법에는 각 영상의 내용이나 색상을 분석하는 과정이 필요하며, 이러한 대부분의 작업들은 많은 시간을 요구한다. 본 보고서에서는 두 영상의 25가지 색상 유사도를 비교할 때, 25가지 색상으로 양자화를 수행하는 모듈을 GPU를 이용한 병렬 처리 연산을 CUDA로 구현함으로써, 두 영상의 색상을 분석하고 비교하는 모듈의 수행 속도를 향상하였다. 일반적으로 GPU는 데이터 병렬 형의 연산에 특화된 프로세서 코어로 구성되어 있기 때문에, 병렬 형태로 프로그래밍 가능한 프로그램들은 처리 속도가 향상된다. 본 보고서에서는 이 연산을 MFC의 DLL 형태로 모듈화하여, CUDA 프로그래밍을 모르는 MFC 개발자들도 활용하기 편리하게 구성하였다.

Keywords CUDA, GPU, Color Similarity

1 Introduction

최근 들어 보급형 디지털 카메라의 확산과 메모리 기술의 발전으로 인한 사진 저장 용량의 확대는 일반 카메라 사용자들이 손쉽게 수백 장 혹은 수천 장의 사진을 촬영할 수 있게 하였다 [1]. 그러므로 이전의 필름 카메라를 사용하였던 시절과는 달리 일반 사람들이 일상생활에서 수백에서 수천 장의 사진을 관리해야 하는 일이 많아졌다 [2, 3]. 그러나 이 작업은 많은 수의 사진들을 일일이 살펴보고 사진의 종류를 판별해야 하기 때문에, 많은 시간이 소모되고 작업상의 불편함이 존재한다 [3].

대부분의 사진 관리 프로그램은 사진의 촬영 시각과 내용 그리고 촬영 장소를 기준으로 각 사진들을 관리하고 분류하는 방법을 사용한다. 여기서 촬영 시각이나 장소와 같은 분류 기준들은 데이터가 선형시간에 분석 가능하다. 그러나 영상의 내용과 같은 분류 기준은 각 사진의 내용을 분석하기 위해, 색상 분포를 분석하거나 형태를 근사화하는 등의 방법이 사용되며, 이들 대부분의 방법들은 2차원의 각 픽셀 데이터 색상의 스캔이나 마스크 기반의 연산을 수행하는 것과 같이 자체적으로 많은 시간이 소모된다. 본 보고서에서는 이러한 연산들 중의 하나인 25가지 색상의 양자화 모듈에 대한 속도를 개

선하는 방법에 대해 논의한다. 먼저 2차원 배열 구조의 연속된 색상값과 2차원 픽셀 기반의 색상값과 같은 구조는 병렬적으로 처리가 가능한 연산이 많은데, 이 연산들을 GPU를 통한 병열 연산으로 처리할 경우, 상당한 속도 개선 효과가 기대된다. 그러므로 본 보고서에서는 병렬 처리를 이용한 속도 개선의 한 예로써, 25가지 색상의 양자화된 영상과 색상 분포도를 분석하여, 두 영상의 색상 비교도를 분석하는 모듈을 CUDA로 구현하는 방법에 대해 논의한다. 또한 CUDA에서 수행되는 GPU 연산을 MS에서 제공하는 DLL(Dynamic Linking Library) 형태로 모듈화함으로써, 25가지 색상 양자화 모듈 연산을 편리하게 MFC 프로그램에서 활용하는 방법에 대해서도 논의할 것이다. 본 보고서는 다음과 같은 2가지 구현에 대해 초점을 맞추어 논의한다.

1. CUDA를 이용한 25 양자화 모듈 연산의 GPU 구현.
2. DLL을 이용한 CUDA연산의 모듈화.

2 문제 제기 - 25 색상의 양자화 모듈의 병목 부분

두 사진의 색상 유사도를 비교하기 위해서, 각 사진 이미지를 몇가지 색상으로 양자화 한 후, 비교를 수행한다. 사진 관리에 있어서 비교해야할 사진의 수가 많기 때무에 이러한 색상 단순화 작업을 수행하지 않는다면, 매 비교하는 사진의 쌍마다 색상 개수 만큼 많은 수의 색상 비교를 수행해야 한다. 예를 들어 k개의 사진이 있을 경우 각 사진들의 유사도를 비교할 경우 $k!$ 번의 사진 비교가 필요하며, 이 때마다 각 픽셀의 유사도를 비교하는 것은 비록 그 정확도 면에서는 뛰어나지만 엄청난 연산을 수행해야 한다. 그러므로 본 보고서에서는 두 사진의 색상 비교를 위해 각 쌍마다 25가지 색상 구성만 비교하는 방법을 사용한다. 즉 각자의 색상 등급의 차이를 이용하여 두 영상의 유사도를 비교한다. 알고리즘 1은 25가지 색상의 양자화 과정을 보여주는데, 가장 가까운 색상으로 양자화 하기 위해서, $15 \sim 25$ 행까지 $width \cdot height$ 와 같이 픽셀의 개수만큼 반복 수행된다. 25가지 색상의 양자화 모듈은 이미지의 각 픽셀마다 25가지 색상과의 RGB 색상의 유클리디안 거리를 계산하고 가장 가까운 색상을 선택해야 하기 때문에(R,G,B 채널은 각각이 콤포넌트로 구성된 독립적인 3차원 벡터), 이 곳에서 많은 시간이 소모된다.

일괄 명령 처리 방식인 CPU가 이 연산을 처리하기 위해선 이미지의 픽셀 개수 $O(m \cdot n)$에 25가지 색상 비교 연산이 적용되기 때문에 많은 시간이 소모된다. 본 보고서에서는 이를 병렬로 처리하기 위해서 CUDA를 이용한 GPU환경에서 양자화를 수행하는 모듈을 적용한다. CUDA (Compute Unified Device Architecture)는 그래픽 처리 장치에서 수행하는 (병렬 처리) 알고리즘을 C 프로그래밍 언어를 비롯한 산업 표준 언어를 사용하여 작성할 수 있도록 하는 GPGPU 기술이다. CUDA는 NVIDIA가 개발해오고 있으며 이 아키텍처를 사용하려면 NVIDIA GPU와 특별한 스트림 처리 드라이버가 필요하다[7]. CUDA는 G8X GPU로 구성된 지포스 8 시리즈급 이상의 모델에서 동작하지만 최근 NVIDIA는 GPGPU(General-Purpose computing on Graphics Processing Units) 즉 그래픽 처리 장치를 통한 일반 목적의 컴퓨팅을 이루고자 계속해서 CUDA를 개선하고 있는 실정이다.

CUDA를 이용하는 대부분의 연구들은 주로 병렬 연산을 통해 성능 향상을 위한 목적으로 사용된다. 또한 CUDA 프로그래밍은 C 기반의 언어로 비교적 손쉽게 구현할 수 있기 때문에 CUDA를 이용한

Algorithm 1 영상의 25가지 색상을 이용한 양자화 방법.

```
1: procedure QuantizationBy25Colors( Bitmap inputImage )
2:     Input
3:         inputImage: 양자화를 수행할 영상.
4:         color25s[25][3]: 25가지 색상 배열 각 원소는 RGB 3차원 벡터임.
5:     Output
6:         Bitmap QuantBitmap : 양자화된 결과 영상.
               ▷ 25가지 색상중에, 양자화될 인덱스와 최소 색상 차이를 비교하기 위한 임시 저장값
7:     minIndex = -1
8:     minColorDiff = 99999999                          ▷ 각 영상의 모든 픽셀을 순회한다.
9:     for x = 0 to width do
10:        for y = 0 to height do
11:            diff = 0                                 ▷ 각 픽셀의 RGB 값을 추출한다.
12:            RColor = inputImages[x][y][R]
13:            GColor = inputImages[x][y][G]
14:            BColor = inputImages[x][y][B]
15:            for i = 0 to 25 do
16:                diff += (color25s[i][R] - RColor)^2
17:                diff += (color25s[i][G] - GColor)^2
18:                diff += (color25s[i][B] - BColor)^2
19:                if diff ≤ minColorDiff then
20:                    minColorDiff = diff
21:                    minIndex = i
22:                end if
23:            end for                                  ▷ 15 ∼ 25 행까지 width · height 만큼 반복 수행됨
24:        end for
25:    end for
26: end procedure
```

병렬 처리 모듈 및 시스템은 계속 그 사용 빈도가 늘어가고 있는 실정이다. CUDA를 이용한 속도 개선 방법은 다양한 분야에서 활용되고 있지만, 본 보고서에서는 3차원 모델의 모델링에 관련된 연구 2가지와 실시간 시뮬레이션과 관련된 연구 1가지를 소개한다. 먼저, Sukhwani [5]는 생물학 분야에서 분자 간의 결합 과정을 모델링하였다. 이 때 각 분자 모델들은 충돌 검사와 같이 많은 계산량이 요구되는 연산 작업이 필요하다. Sukhwani는 이를 병렬 구조로 변환하여 CUDA에 적용하여 속도 개선을 이루었다. Jang [4]은 Neural Network 기반의 텍스트 인식 시스템을 CUDA로 구현하였다. 이를 통해 기존의 기법보다 15배 정도의 속도 향상을 가져왔다. CUDA 를 이용한 실시간 시뮬레이션과 관련된

연구로 Tobias [6] 는 의료분야에서 초음파 검색 결과를 시뮬레이션을 위해 CUDA 를 사용한 실시간 시뮬레이션 프로그램을 개발하였다. 이 시스템은 촬영된 3D CT 영상에서 발생하는 3D 및 2D 잡음을 제거하고 Acoustic impedance (음파중 특이 사항이 있는 곳을 추출) 및 반사 정도를 혼합하여 CT 영상의 실제 시뮬레이션 결과를 시각화한 연구이다.

3 CUDA와 DLL을 이용한 25 색상 양자화 모듈

Windows 계열의 운영체제에서는 운영체제 본래 기능으로 소프트웨어의 몇명 기능을 몇개의 모듈 파일로 나누어 디스크에 두고 필요한 것만을 실행 메모리에 실어서 사용하기 위한 동적 링킹 라이브러리 (Dynamic Linking Library)를 사용한다. DLL은 다수의 실행 파일에 공유될 수 있기 때문에 디스크 용량이나 메모리를 절약할 수 있지만 운영체제의 중요한 기능을 대체하기 때문에 윈도우즈의 해킹이나 보안상 문제가 되기도 한다. 본 보고서에서는 25가지 색상 유사도의 GPU 연산을 CUDA를 이용한 DLL 형태로 생성하여, 각 Windows 플랫폼에서 효율적으로 사용할 수 있다. 참고로 보고서에서 사용한 각 SDK 환경들의 버전은 Visual Studio 2008 그리고 CUDA 2.2 버전이며 CUDA를 이용한 DLL 파일 생성 방법은 다음과 같다.

1. Visual Studio에서 DLL을 위한 프로젝트 파일 (Project File: Win32, Templates: Console Application, Application Settings : DLL) 을 생성한다.
2. Make25Color_kernel.cu 파일을 프로젝트에 추가한 후, Make25Color_kernel.cu 파일의 컴파일 방법을 Visual Studio에게 다음과 같이 알려 준다.
   - Make25Color_kernel.cu 의 property >> Custom Build Step 에서 컴파일 방법은 다음과 같다
   - nvcc -I"$(CUDA_INC_PATH)" -c -o $(ConfigurationName)\Make25Color_kernel.cu Make25Color_kernel.cu
   - .cu 파일의 컴파일 결과를 저장할 Outputs 속성에 $(ConfigurationName) \Make25Color_kernel.cu 과 같이 입력한다.
3. 프로젝트 전체 속성에서 CUDA의 lib 파일들을 사용할 수 있게 하기 위해서, Linker 옵션의 Additional Dependencies 항목에 "cudart.lib" 와 "cuda.lib" 등을 연결한다.
4. CUDA관련 추가 라이브러리들을 MFC가 손쉽게 활용할 수 있도록 Additional Library Directory 에 $(CUDA_LIB_PATH) 를 입력한다.

여기서 nvcc는 NVIDA에서 CUDA를 컴파일하기 위해서 CUDA SDK에서 제공하는 CUDA 전용 컴파일러이며, $(CUDA_INC_PATH) 와 $(CUDA_LIB_PATH) 는 CUDA의 실행 환경등이 저장된 헤더 파일과 라이브러리 파일들이 존재하는 환경변수이다. 그 외 $(ConfigurationName) 은 VISUALSTUDIO에서 보통 "Debug"나 "Release"와 같이 실행 파일이 저장되는 폴더 경로를 의미하는 매크로이다.

여기까지가 CUDA를 이용하여 DLL을 만들기 위한 작업 과정은 구축된 상황이며, 25가지 색상 비교 모듈은 .cu 파일의 구성은 다음과 같다. 실제 GPU 프로그래밍은 블럭단위의 연산을 병렬적으로 처리하기 때문에, 프로그래머는 블럭단위의 연산하나만을 정의하면 나머지는 CUDA가 프로그래머가 정의한 각 블럭 연산들을 병렬적으로 수행한다. 다음 프로그램에서 $blockIdx$는 전체 영상의 한 블럭을 의미하며, 이 블럭에 대한 각 픽셀 연산(threadIdx)을 수행하면 된다. 25가지 색상의 양자화 모듈은 한 픽셀에 대해서 25가지 색상 중 가장 가까운 유클리디안 거리에 있는 색상을 찾는 과정이며 프로그램 1과 같다. 해당 프로그램에서 각 인자들 중 IN과 OUT은 $CUDA\_25ColorsQuantization$ 함수의 입력과 결과를 각각 의미한다.

프로그램 1. Make25Color_kernel.cu 파일 실제 GPU 프로그램

```
__global__ void CUDA_25ColorsQuantization(IN int *src,IN int sw,IN int sh,OUT int *dst,
    OUT int *h_DestIndexes,IN int colorNum){
    // src : 비트맵의 입력 데이터 BGRA 순서로 각각 int 배열로 들어온다.
    // sw, sh : 비트맵의 너비와 높이
    // dst : src 의 양자화된 비트맵 데이터 마찬가지로 BGRA순서로 구성된다.
    // h_DestIndexes : 양자화된 픽셀의 25가지 색상 인덱스
    // colorNum : 색상의 채널 개수 (RGBA 4가지)
    // 25가지 색상 테이블
    const int dominantColorTable[eCDOMCOLOR_COUNT][3] = {
            {0, 0, 0},      // Black        0
            {0, 182, 0},    // Sea Green    1
            ...
            {255, 146, 0},  // Orange       23
            {255, 255, 255} // White        24
    };
    /* 1. Index 설정 시작 */
    // Block Index 초기 위치 Cell 취급 (Thread의 집합)
    int bx = blockIdx.x;
    int by = blockIdx.y;
    int tx = threadIdx.x;
    int ty = threadIdx.y;
    int iX = bx *blockDim.x + tx;
    int jY = by *blockDim.y + ty;
    if (iX <0 || jY < 0 || iX >= sw|| jY >= sh )    return;
    /*2. 한번에 처리할 Block 설정
    CUDADS_BLOCKSIZE 내에 있는 픽셀들을 한꺼번에 처리한다.
    썸네일의 한 픽셀 (iX,jY)을 위해서, src의 grid 크기 만큼 처리한다.*/
```

```
float minData = 200000;
int minIndex = eCDOMCOLOR_COUNT +1;
float temp = 0.0;
int srcIdx = jY* sw + iX ;
int B = src[srcIdx*colorNum];
int G = src[srcIdx*colorNum +1];
int R = src[srcIdx*colorNum +2];
for (int i=0;i<eCDOMCOLOR_COUNT;i++){
    temp = (R-dominantColorTable[i][2])*(R-dominantColorTable[i][2])
        + (G-dominantColorTable[i][1])*(G-dominantColorTable[i][1])
        + (B-dominantColorTable[i][0])*(B-dominantColorTable[i][0]);
        if (minData > temp){
            minData = temp;
            minIndex = i;
        }
}
dst[srcIdx*colorNum] = dominantColorTable[minIndex][2];
dst[srcIdx*colorNum+1] = dominantColorTable[minIndex][1];
dst[srcIdx*colorNum+2] = dominantColorTable[minIndex][0];
dst[srcIdx*colorNum+3] = src[srcIdx*colorNum+3];
h_DestIndexes[srcIdx] = minIndex;
return;
```

생성된 프로젝트 파일을 컴파일하게 되면 그 결과 파일로써, lib 파일과 dll 파일이 생성된다. 이 두개의 파일을 이용하여 각 MFC 응용프로그램에서는 CUDA로 구현한 25가지 색상 양자화 GPU 모듈을 함수처럼 사용한다. 보고서에서는 $Make25Color\_kernel.cu$ 파일을 래핑(wrapping)한 함수의 구현은 생략하였는데 대략적으로 이함수가 하는 일은 Bitmap *파일의 색상 데이터를 추출하여 int * 데이터형태로 만들고 너비와 높이를 $Make25Color_kernel.cu$에 넘겨준 후, 계산된 양자화 결과를 다시 Bitmap* 결과 파일로 저장하는 단순한 인터페이스 역할을 한다. 그리고 래핑 함수가 하는 또 다른 중요 기능으로, 추출된 색상 데이터(int*)를 GPU(.cu 파일)에서 활용하기 위해서, CPU와 GPU의 메모리를 서로 접근하고 복사하는 등의 역할이 필요한데 이 역할은 $cudaMalloc$과 $cudaMemcpy$ 함수로 제어할 수 있다. 이 Wrapping 함수의 이름은 $CUDA\_25ColorsQuantizationDll$ 이며, dll 파일의 인터페이스 역할을 수행하기 위해서, 다음과 같이 선언하며 각 인자는 프로그램 1과 유사하다. 더 자세한 CUDA 관련 사항은 CUDA SDK의 Document 문서를 참조하면 된다.

- ___declspec(dllexport) void CUDA_25ColorsQuantizationDll (int *h_gpuSource,int sw,int sh, int *h_gpuDest,int *h_DestIndexes, int colorNum);

4 CUDA를 이용한 DLL의 사용

본 장에서는 앞서 CUDA로 생성한 DLL파일의 사용방법에 대해 논의한다. MFC 응용 프로그램에서의 dll 사용방법은 일반적으로 Window 프로그래머라면 쉽게 접근하고 알수 있는 방법이므로, 간단하게 다음과 같이 요약하였다.

1. dll에 정의된 앞 절의 인터페이스 함수를 CUDA를 사용할 곳에 같은 방식으로 선언한다.
2. $CUDA\_25ColorsQuantizationDll$ 함수를 호출해서 사용하면 된다.

이외에 실제 CUDA가 동작하는 환경을 만들기 위해서, CUDA 관련 라이브러리나 헤더 파일들을 MFC 응용 프로그램에 링크해줘야 한다. 그 과정은 첫번째로 $< cuda\_runtime\_api.h >$ 헤더 파일 선언을 수행하고 프로젝트의 속성에서 다음과 같이 설정한다.

1. Linker > input > Additional Dependencies : cuda.lib cudart.lib DLL 프로젝트의 결과 Library 파일.lib
2. Linker > General > Additional Library Directories: $(CUDA_LIB_PATH), DLL 프로젝트의 결과 Library 파일이 있는 곳
3. C/C++ > Additional Include Directories : $(CUDA_INC_PATH) 입력

5 결 론

표 1. CUDA를 이용한 사진의 로드 및 양자화 처리 모듈의 성능 개선 결과

사진 개수	해상도	CUDA 사용 전 (s)	CUDA 사용 후 (s)	개선 정도 (s)
100	1,600 × 1,200	99	17 ∼ 18	81
140	2,048 × 1,536	141	37	104

본 보고서에서는 사진 관리에서 각 사진들의 주요 분류 기준에서 계산 비용이 많이 소모되는 분류 기준 중 하나인 색상 유사도 비교 연산의 계산 속도를 개선하기 위해 GPU를 이용한 색상 양자화 모듈의 DLL 함수를 구현하였다. 그리고 그 결과 dll 파일을 본 저자가 구현하고 있는 사진 관리 프로그램에 적용하였는데, 각 사진을 메모리에 로드하고 분석하는 과정의 시간이 4.5 ∼ 6 배의 속도 개선이 이루어졌다. 향상된 속도에 대해 논의하기 위해서, 먼저 구현 중인 사진 관리 프로그램의 구체적인 시각 측정 구간 안에서 수행되는 작업들은 다음과 같다.

1. 폴더 내에 있는 원본 사진들을 메모리에 로드한다.
2. 각 사진의 대표 사진(360 × 240) R_i와 썸네일 (100 × 60)을 생성한다.
3. 대표 사진들 R_i의 25 양자화 영상을 계산하고 각 색상별 분포도를 조사한다.

4. OpenCV의 얼굴 찾기 알고리즘과 Surf 특징 추출 알고리즘을 대표 사진 R_i에서 수행한다.

본 보고서에서는 위의 3번째 과정을 GPU 연산으로 대체하였으며, 해상도 1,600 × 1,200 크기의 사진 100 장을 분석하는데, 총 99초 소모되던 계산 시간이 17초에서 18초 가량으로 단축되었으며, 해상도 2,048 × 1,536의 사진 140 장은 141초 소모되던 계산이 37 초 정도로 단축되었다. 모든 테스트 결과는 Windows 7 32비트 운영체제에서 실행되었으며, 측정한 컴퓨터 사양은 CPU Intel(R) Core(TM) i5 CPU 750 2.67Ghz 이며 설치된 메모리는 4GB (2.99GB 사용) 이다. 간단한 테스트 결과만으로도 대략적으로 4.5 ∼ 6 배 정도의 속도를 표1 과 같이 개선한 것을 알 수 있었다.

참고 문헌

1. B. B. Bederson, "PhotoMesa: a zoomable image browser using quantum treemaps and bubblemaps," in *Proc. of the 14th ACM UIST '01.* New York, NY, USA: ACM Press, pp. 71–80.
2. M. Cooper, J. Foote, A. Girgensohn, and L. Wilcox, "Temporal event clustering for digital photo collections," *ACM Transactions on Multimedia Computing, Communications and Applications*, vol. 1, no. 3, pp. 269–288, 2005.
3. A. Graham, H. Garcia-Molina, A. Paepcke, and T. Winograd, "Time as essence for photo browsing through personal digital libraries," in *Proc. of the 2nd ACM/IEEE-CS JCDL '02.* New York, NY, USA: ACM, 2002, pp. 326–335.
4. H. Jang, A. Park, and K. Jung, "Neural network implementation using cuda and openmp," in *Computing: Techniques and Applications, 2008. DICTA '08.Digital Image*, 2008, pp. 155–161. [Online]. Available: http://dx.doi.org/10.1109/DICTA.2008.82
5. B. Sukhwani and M. C. Herbordt, "Gpu acceleration of a production molecular docking code," in *GPGPU-2: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units.* New York, NY, USA: ACM, 2009.
6. J. P. Tobias Reichl and O. Acosta, "Ultrasound goes gpu: real-time simulation using cuda," http://campar.in.tum.de/pub/reichl2009spie/reichl2009spie.slides.pdf.
7. Wiki, "쿠다," http://enc.daum.net/dic100/contents.do?query1=10XX171957.