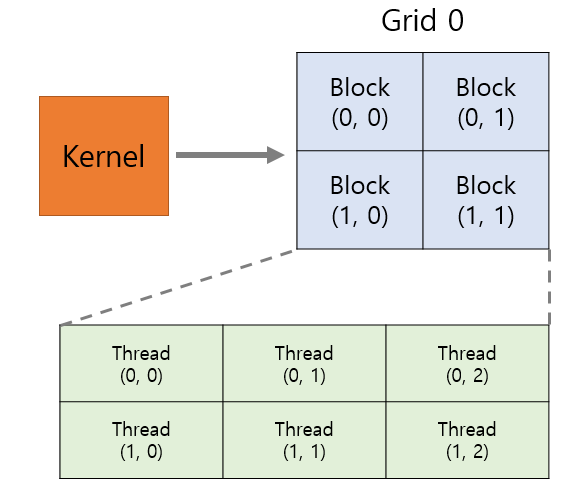
|  |
| --- |
| **OpenGL을 사용한 CUDA 기반 실시간 광원 효과**  **구현 및 성능 분석** |
| 석혜경 장승민  서강대학교 컴퓨터공학과  hkseok@sogang.ac.kr |
| CUDA-based Real-Time Light Effects Using OpenGLImplementation and performance analysis |
| Hyekyoung Seok Seungmin Jang  Department of Computer Science and Engineering, Sogang University |
| **요 약**  한국정보과학회는 정보과학에 관한 기술을 발전, 보급시키고 회원상호간의 친목을 도모하기 위하여 1973년 3월 3일에 설립되었으며, 정보통신부에 '사단법인 한국정보과학회'로 등록되었다. 학회의 주요 활동은 1) 컴퓨터 기술 및 이론에 관한 새로운 연구결과를 발표하는 기회를 제공하고, 2) 국내의 컴퓨터 관련 기술 개발에 참여하며, 3) 국제적 학술 교류 및 협력 증진을 도모하고, 4) 회원 상호간의 친목을 증진시키는 것이다. | | |

**1. 서론**

GPU(Graphics Processing Unit)는 매우 높은 부동 소수점 처리 성능, 거대한 메모리 대역폭 및 비교적 저렴한 비용으로 일반 목적의 계산을 위한 매우 유용한 하드웨어 플랫폼으로 발전하였다. GPU의 급속한 발전으로 인해 그래픽 처리의 주요 목적을 넘어서 성능, 아키텍처 및 프로그래밍 가능성에서 다양한 분야에 응용할 수 있는 GPGPU(General-Purpose computing on Graphic Processing Unit) 기술이 활발히 사용되어진다.

 GPGPU는 병렬 처리 능력과 속도 개선을 목적으로 다방면에 걸쳐 사용되고 있는데, 대표적인 GPGPU기술로는 NVIDIA사의 CUDA(Compute Unified Device Architecture)[1]가 있다. CUDA는 여타 프레임워크에 비해 높은 성능과 편리한 툴킷을 제공하여 널리 사용되고 있다[2].

<그림 1> 커널, 그리드, 블록, 쓰레드

본 논문에서는 OpenGL과 CUDA를 활용한 실시간 광원 기법을 구현하고, 커널 사이즈에 따른 성능 변화를 분석한다. 구현한 프로그램에서는 각 스레드마다 디스플레이의 픽셀들에 대한 연산을 병렬적으로 수행하여, 연산 속도를 향상시킨다. 연산의 병렬화로 인해 빠른 시간 내에 픽셀 렌더링이 가능하다. 또한 연산 수에 따른 커널의 실행 단위 구성을 달리했을 때 어떤 성능 변화가 발생하는지, 또 그 원인이 무엇인지 추정한다. 커널 실행 단위로 구성한 다섯 가지 경우는 블록의 개수와 쓰레드 개수의 조절로 구성하였다. 또한 경우에 따른 프레임 한 장당 렌더링 시간 및 1초당 렌더링 되는 프레임 수를 각각 구한다. 이를 통해 효과적인 CUDA 커널의 구성을 제안한다.

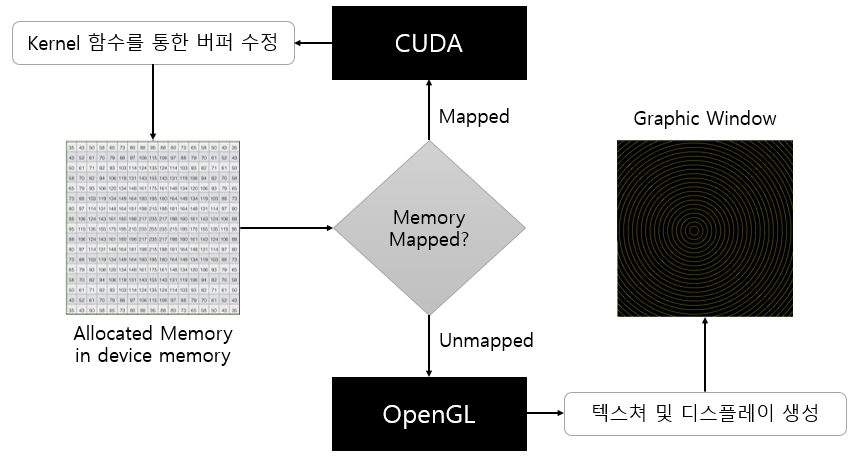
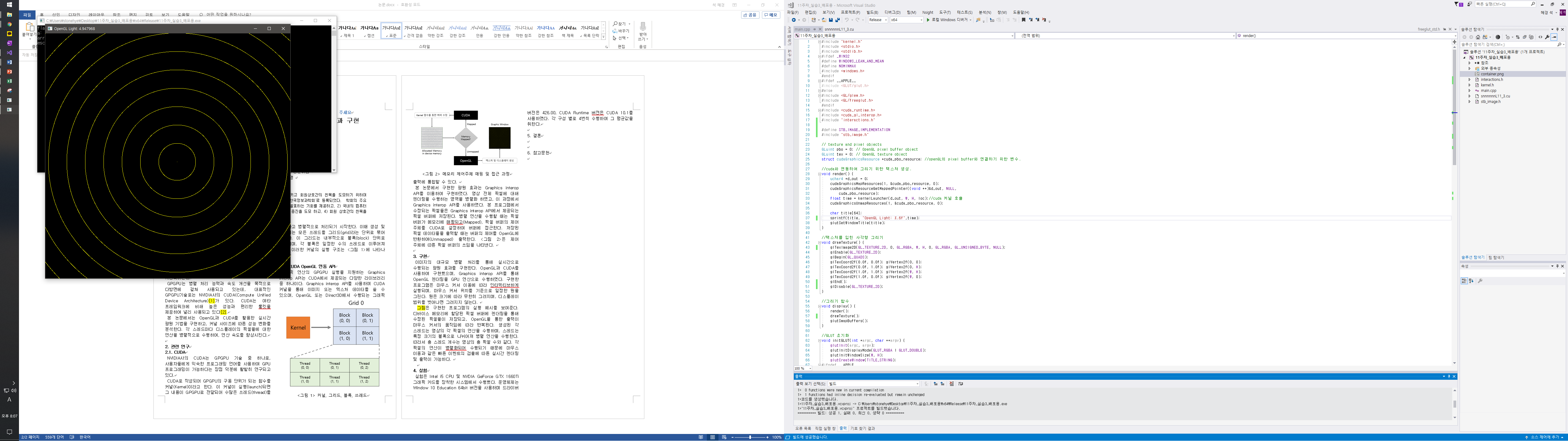
**2. 관련 연구**

**2.1. CUDA**

NVIDIA사의 CUDA는 GPGPU 기술 중 하나로, 사용자들에게 익숙한 프로그래밍 언어를 사용하여 GPU 프로그래밍이 가능하다는 장점 덕분에 활발히 연구되고 있다.

CUDA로 작성되어 GPGPU의 구동 단위가 되는 함수를 커널(Kernel)이라고 한다. 이 커널이 실행(launch)되면 그 내용이 GPGPU로 전달되어 수많은 쓰레드(thread)를 생성하고 병렬적으로 처리되기 시작한다. 이때 생성 및 수행되는 모든 쓰레드를 그리드(grid)라는 단위로 묶어 칭한다. 이 그리드는 내부적으로 블록(block) 단위로 구성되며, 각 블록은 일정한 수의 쓰레드로 이루어져 있다. 이러한 커널의 실행 구조는 <그림 1>에 나타나 있다.

**2.2. CUDA OpenGL 연동 API**

그래픽 연산의 GPGPU 실행을 지원하는 Graphics interop API는 CUDA에서 제공되는 다양한 라이브러리 중 하나이다. Graphics interop API를 사용하여 CUDA 커널을 통해 이미지 또는 텍스처 데이터를 쓸 수 있으며, OpenGL 또는 Direct3D에서 수행되는 그래픽 출력에 통합할 수 있다.

<그림 2> 메모리 제어주체 매핑 및 접근 과정

본 논문에서 구현한 광원 효과는 Graphics interop API를 이용하여 구현하였다. 영상 전체 픽셀에 대해 렌더링을 수행하는 영역을 병렬화 하였고, 이 과정에서 Graphics interop API를 사용하였다. 본 프로그램에서 수정되는 픽셀들은 Graphics interop API에서 제공되는 픽셀 버퍼에 저장된다. 병렬 연산을 수행할 때는 픽셀 버퍼가 메모리에 매핑되고(Mapped), 픽셀 버퍼의 제어 주체를 CUDA로 설정하여 버퍼에 접근한다. 저장된 픽셀 데이터들을 출력할 때는 버퍼의 제어를 OpenGL에 반환하여(Unmapped) 출력한다. <그림 2>은 제어 주체에 따른 픽셀 버퍼의 쓰임을 나타낸다.

**3. 구현**

이미지의 대규모 병렬 처리를 통해 실시간으로 수행되는 광원 효과를 구현한다. OpenGL과 CUDA를 사용하여 구현했으며, Graphics interop API를 통해 OpenGL 렌더링을 GPU 연산으로 수행하였다. 구현한 프로그램은 마우스 커서 이동에 따라 인터렉티브하게 실행되며, 마우스 커서 위치를 기준으로 일정한 원을 그린다. 원은 크기에 따라 무한히 그려지며, 디스플레이 범위를 벗어나면 그려지지 않는다.

<그림 3>은 구현한 프로그램의 실행 예시를 보여준다. 디바이스 메모리에 할당된 픽셀 버퍼에 렌더링을 통해 수정된 픽셀들이 저장되고, OpenGL을 통한 출력이 마우스 커서의 움직임에 따라 반복된다. 생성된 각 스레드는 영상의 각 픽셀의 연산을 수행하며, 스레드는 특정 크기의 블록으로 나뉘어져 병렬 연산을 수행한다. 따라서 총 스레드 개수는 영상의 총 픽셀 수와 같다. 각 픽셀의 연산이 병렬화 되어 수행되기 때문에 마우스 이동과 같은 빠른 이벤트의 검출에 따른 실시간 렌더링 및 출력이 가능하다.

**4. 실험**

실험은 Intel i5 CPU 및 NVDIA GeForce GTX 1660Ti 그래픽 카드를 장착한 시스템에서 수행했다. 운영체제는 Window 10 Education 64bit 버전을 사용하며 드라이버 버전은 426.00, CUDA Runtime 버젼은 CUDA 10.1를 사용하였다. 각 구성 별로 4번씩 수행하여 그 평균값을 취한다.

실험 시 800×800 크기의 디스플레이를 구성하였고, 실험에 사용된 커널은 한 개의 픽셀 당 double형 계산을 수행하여 총 640,000번을 수행한다. 실행 단위의 구성은 총 다섯 경우이다.

<그림 3> 구현 프로그램 실행 예시

1) 블록 25×25개, 각 블록의 쓰레드 32×32개

2) 블록 25×50개, 각 블록의 쓰레드 32×16개

2) 블록 50×50개, 각 블록의 쓰레드 16×16개

2) 블록 13×50개, 각 블록의 쓰레드 64×16개

2) 블록 800×800개, 각 블록의 쓰레드 1×1개

<표 1>은 다섯 경우에 대해 프레임 한 장당 렌더링 시간을 나타내며 단위는 초(sec)이다. <표 2>는 <표 1>의 값에 역수를 취해준 값으로, 다섯 경우에 대한 1초당 렌더링 되는 프레임 수이며 단위는 fps이다.

<표 1> 프레임 한 장당 렌더링 시간 비교 (단위: 초)

<표 2> 1초당 렌더링 되는 프레임 수 비교 (단위: fps)



다섯 경우 중 블록 당 쓰레드 개수보다는 블록의 개수를 더 크게 구성했을 때(블록 25×50개, 각 블록의 쓰레드 32×16개 / 블록 50×50개, 각 블록의 쓰레드 16×16개)의 성능이 가장 뛰어났다. 반면 블록과 쓰레드 어느 쪽으로든 편중되게 구성했을 경우 상대적으로 성능이 떨어졌다. 기본적으로 블록은 하나의 스트리밍 멀티 프로세서(Streaming Multiprocessor)에 할당되어 GPGPU의 자원을 할당 받는 단위가 되는데, 이러한 이유로 블록의 수가 전체 커널의 수행 성능에 영향을 미치는 것으로 보인다. 다만 800×800개의 블록을 구성했을 때 굉장히 낮은 성능을 보였는데, 이는 극단적으로 한 블록 당 하나의 쓰레드를 구성해 비효율적인 GPU 프로그래밍을 수행하여 이 같은 결과를 도출한 것으로 보인다.

GPU 스케줄러는 블록을 단위로 각 SM에 일을 배정한다. 이때, 블록 사이즈가 너무 크면 SM에 과도하게 일이 몰릴 수 있다. 반면에 블록 사이즈가 너무 작으면 SM이 자주 GPU스케줄러에 의해서 일을 받아와야 함으로서 성능의 저하를 가져온다. 추가적으로 하나의 SM은 일을 처리할 때 wrap(32)단위로 일을 수행한다. 따라서 블록 사이즈는 32의 배수로 설정하는 것이 성능을 최적화하는데 도움이 된다.

**5. 결론**

본 논문에서는 CUDA를 활용한 실시간 렌더링 기법을 구현했다. 구현한 커널의 픽셀 연산 수에 맞춰 커널의 실행 단위 구성을 달리할 때의 소요 시간 변화를 측정하고, 이때 발생하는 성능 변동의 원인을 추정했다.

꾸준히 항상 되고 있는 그래픽 하드웨어의 성능아래, 최대한의 성능을 발휘하는 프로그램을 만들기 위해서는, GPU의 구조와 GPGPU 기능에 대한 이해가 필수적이라고 할 수 있다. 이를 토대로 향후 CUDA GPGPU 실행 단위 구성의 최적화된 방법이 제시되기를 기대한다.

**6. 참고문헌**

1. Harris, M. (2005). Gpgpu: General-purpose computation on gpus. SIGGRAPH 2005 GPGPU COURSE, 1-51.
2. Kirk, D. (2007, October). NVIDIA CUDA software and GPU parallel computing architecture. In ISMM (Vol. 7, pp. 103-104).