카메라 어레이를 위한 GPU 기반 이미지 병합

배도현, 이영준, 신희재, 뭉크바야르, 김민호*, 김진석
서울시립대학교 컴퓨터과학과
e-mail : ilsoo99@uos.ac.kr

GPU-Based Image Stitching for Camera Array

Do Hyun Bae, Young-joon Lee, Heejae Shin, Munkhbayar Bayartsogt, Minho Kim, Jin Suk Kim
School of Computer Science, University of Seoul

요 약

본 논문에서는 웹캠 카메라 어레이(camera array)로 얻은 여러 장의 이미지를 빠른 속도로 봉합(stitching)하여 고해상도 이미지를 얻기 위해 그래픽스 하드웨어를 이용하는 병렬 알고리즘을 제시한다. 고정된 레이아웃의 카메라 어레이를 이용하여 평면 혹은 원경을 촬영하는 경우, 기존에 널리 쓰이던 평면 사영 이미지 봉합(planar projective image stitching)과 선형 혼합(linear blending)을 통해 만족스런 결과를 얻을 수 있다. 본 논문에서는 이러한 연산을 그래픽스 하드웨어에서 병렬처리 함으로써 추후 실시간 고해상도 동영상 스트리밍 이미지 병합에 활용할 수 있을 정도로 빠른 속도로 처리하는 방법을 제시한다.

1 서론

최근 이미지 센서 및 디스플레이의 기능이 비약적으로 발전하면서 과거에는 얻을 수 없었던 고해상도 이미지를 얻을 수 있게 되었다. 그러나 이러한 고기능 센서의 경우 높은 가격으로 인해 아직까지 널리 보급되어 있지는 않은 실정이다. 이러한 점을 보완하고 또한 현재의 이미지 센서로는 얻을 수 없는 초고해상도 이미지를 얻기 위해서 여러 대의 카메라를 사용하여 얻은 이미지들을 봉합(stitching)하여 고해상도 이미지를 얻는 카메라 어레이에 대한 연구가 진행되고 있다. 본 논문에서는 실시간 고해상도 동영상 이미지 추출을 염두에 두고 GPU 를 이용한 고속 병렬 이미지 병합 알고리즘을 제시한다.

2 선행 연구

Yang et al.은 [1] 광각의 웹캠으로 구성된 카메라 어레이를 사용하여 촬영 후 이미지의 초점영역을 변환할 수 있는 광장(light field) 카메라를 개발하였다. Wilburn et al.은 [2] 망원렌즈 카메라 어레이를 이용한 고해상도 이미지 및 광각렌즈 카메라 어레이를 이용한 HDR(High Dynamic Range)이미지를 얻는 시스템을 개발하였다. 이미지 병합을 위해서는 AutoStitch [3] 패키지를 개량하여 사용하였다. Szeliski 는 [4] 평면 혹은 원경 이미지를 사영변환(projective transformation)에 의해 병합하는 평면 이미지 병합 방법을 제안하였다. 그 밖의 최신 이미지 병합 알고리즘은 Szeliski 의 책 [5]에 자세히 소개되어 있다.

- 본 논문은 한국과학기술정보연구원 '다중 웹캠어레이를 이용한 고해상도 이미지 추출 알고리즘' 연구과제의 지원으로 연구 되었습니다.
* 교신저자

3 배경 지식

3.1 사영변환 (Projective Transformation)

평면 이미지 병합을 위해서는 각 이미지가 사영변환에 의해 네 꼭지점 $(x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3)$ 에 의해 정의된 임의의 사각형 Q를 단위 정사각형으로 매핑하여야 한다. 이러한 사영변환은 3x3 균질 행렬(homogeneous matrix)에 의해 정의가 된다. 여기서는 편의상 그 역변환, 즉 단위 정사각형을 Q로 매핑하는 균질 행렬을 다음과 같이 정의한다.

$$P := \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

위 행렬은 균질(homogeneous)하기 때문에 실제로는 8개의 미지수로 구성되어 있음에 유의한다. 따라서 네 꼭지점 좌표의 매핑을 이용하여 8×8 선형 시스템의 해로 구할 수 있다. Heckbert 가 [6] 구한 해는 다음과 같다.

$$g := \frac{\begin{vmatrix} \sum x_i & x_3 - x_2 \\ \sum y_i & y_3 - y_2 \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & x_3 - x_2 \\ y_1 - y_2 & y_3 - y_2 \end{vmatrix}}$$

$$h := \frac{\begin{vmatrix} x_1 - x_2 & \sum x_i \\ y_1 - y_2 & \sum y_i \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & x_3 - x_2 \\ y_1 - y_2 & y_3 - y_2 \end{vmatrix}}$$

$$a := x_1 - x_0 + g x_1$$
$$b := x_3 - x_0 + h x_3$$
$$c := x_0$$
$$d := y_1 - y_0 + g y_1$$

$$e \coloneqq y_3 - y_0 + hy_3$$
$$h \coloneqq y_0$$
$$i \coloneqq 1$$

따라서 우리가 원하는 변환은 P^{-1}이다.

3.2 GPU 연산 (GPU Computing)

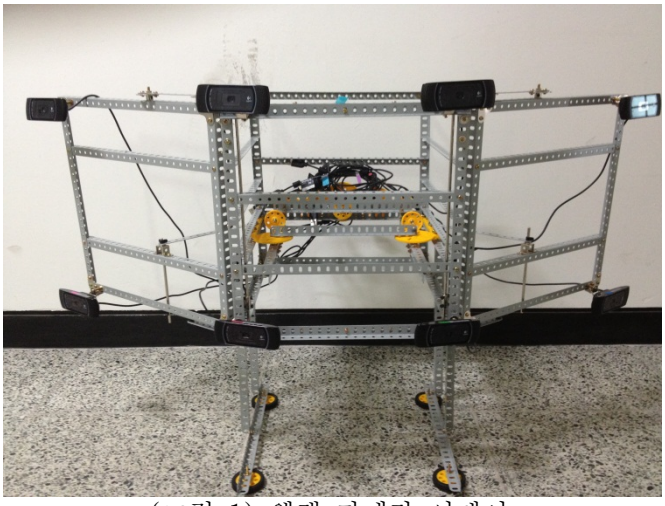
최근 게임산업의 급속한 팽창과 맞물려 그래픽스 하드웨어의 성능이 비약적으로 발전하였다. 특히 최신 그래픽스 하드웨어의 프로그래밍 기능을 활용하여 병렬연산을 매우 빠른 속도로 처리하는 것이 가능하게 되었다. 표준 그래픽스 라이브러리인 OpenGL 에서는 버전 2.0 이후에 GLSL(OpenGL Shading Language)를 통해 이러한 기능을 활용할 수 있도록 하였다.

4 GPU 기반 이미지 병합

여기서는 본 연구에서 이용하고 있는 웹캠 기반 카메라 어레이의 구성 및 이미지 추출, 그리고 GPU 기반 이미지 봉합 알고리즘을 소개한다.

4.1 웹캠 기반 카메라 어레이

본 시스템에서는 2×4의 웹캠으로 구성되어 있는 카메라 어레이를 이용하여 이미지 데이터를 얻는다. (그림 1) 웹캠은 Logitech 사의 HD 프로 웹캠 C910 모델을 사용하였고, 각 웹캠은 USB 2.0 포트를 통해 PC 와 연결되어 있다. 웹캠의 시야각이 상대적으로 넓기 때문에 인접한 웹캠의 시야각이 겹치는 영역을 줄이기 위해 그림처럼 각도를 벌여주었다. 양 끝의 카메라는 약 120 도의 각도로 벌어져 있다.



(그림 1) 웹캠 카메라 어레이

4.2 동시 촬영 과정

여러 대의 웹캠을 PC 에서 동시에 제어하기 위해서는 소프트웨어와 더불어 하드웨어적으로 해결해야 할 부분들이 존재한다. 일반적인 상용 PC 의 메인보드에서는 비디오 웹캠을 한꺼번에 받아들일 수 있는 대역폭이 2~3 대로 제한되어 있기 때문에 PCI-usb 카드를 별도 장착하여 8 대의 카메라를 동시에 촬영할 수 있는 추가 대역폭을 확보하였다. 소프트웨어적으로는 VideoInput 라이브러리로 웹캠을 제어하고 OpenCV [7] 라이브러리로 웹캠으로부터 jpeg 포맷의 이미지를 추출하였다.

4.3 전처리 과정 (Pre-processing stage)

본 시스템은 이미지 병합을 위해 평면 사영 이미지 병합 (planar projective image stitching)방법 [4]을 채택하였다. 이 방법은 병합 후 빠르게 렌더링할 수 있다는 장점이 있지만, 평면 혹은 원경의 이미지의 경우에만 적용될 수 있다는 단점이 있다. 본 시스템은 추후 실시간 원경 고해상도 동영상 추출을 염두에 두고 개발되었기 때문에 이 방법을 채택하였다.

전처리 과정에서는 레이아웃이 고정된 카메라 어레이를 통해 얻은 원경 이미지를 평면 사영 병합 방법을 통해 병합한다. 이는 카메라 어레이의 레이아웃에 따라 한 번만 수행이 되는 오프라인 과정으로, 기존의 최적화 알고리즘 혹은 수작업을 통해 수행된다. 이러한 전처리 과정을 통해 변형된 8 장의 이미지에 해당하는 사영변환 행렬을 구한다.

4.4 GPU 기반 이미지 스티칭

앞의 전처리 과정에서 얻은 변환 행렬은, 카메라의 레이아웃이 고정되어 있는 한 다른 원경 이미지를 병합할 때도 사용할 수 있다. 본 연구에서는 이러한 연산을 고속으로 처리하기 위해 GLSL 프로그램을 사용하여 그래픽스 하드웨어에서 병렬로 처리하였다. 렌더링 시에는 매 프레임마다 병합한 고해상도 이미지를 렌더링 할 FBO(Frame buffer object)의 전체 영역을 덮는 사각영역을 그래픽스 파이프라인으로 보낸다. 버텍스 쉐이더(Vertex shader)에서는 별다른 연산 없이, 단지 버텍스의 좌표를 선형보간(linear interpolation)되도록 프래그먼트 쉐이더(Fragment shader)로 보내는 역할 만을 수행한다. 프레그먼트 쉐이더에서는 (보간된) 각 프레그먼트의 좌표를 이용하여 각 사영변환 행렬을 이용하여 단위정사각형으로 매핑한다. 이렇게 변환된 좌표 값을 텍스처 좌표로 이용하여 텍스처로 저장되어 있는 각 이미지로부터 해당 색깔의 값 c_i를 얻는다. 이 때, 스티칭된 이미지들이 연결된 부분을 최소화 하기 위해 텍스처에 저장된 블렌딩 함수로부터 각 이미지의 가중치 w_i도 함께 얻는다. 이를 8 개 이미지 각각에 대하여 수행한 결과를 취합하여 다음과 같은 최종 프레그먼트 색깔을 얻게 된다.

$$\frac{\sum_{i=1}^{8} w_i c_i}{\sum_{i=1}^{8} w_i}$$

그림 2 와 3 은 각각 GLSL 버텍스 쉐이더와 프레그먼트 쉐이더를 보여준다. 그림 4 는 블렌딩 함수 텍스처를 보여준다. 선형 함수 외의 블렌딩 함수도 사용이 가능하지만, 각 이미지가 영역 밖에서 렌더링되는 것을 막기 위해서는 블렌딩 함수 텍스처의 가장자리 값을 모두 0 으로 설정하고 텍스처의 `GL_TEXTURE_WRAP_*` 값을 `GL_CLAMP` 로 설정해 주어야 한다.

```
varying  vec2     pos2d;
void main (void)
```

```
{
     gl Position = ftransform();
     pos2d = gl_Vertex.xy;
}
```

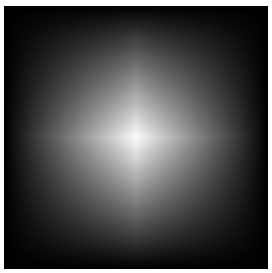
(그림 2) GLSL 버텍스 쉐이더

```
uniform sampler2D tex[8];
uniform mat3      M[8];
uniform sampler2D texw;
varying vec2      pos2d;

void main (void)
{
     vec3 tc;
     float         w;
     float         sum w = 0.0;
     gl_FragColor = vec4(0.0,0.0,0.0,0.0);

     for(int i=0 ; i<8 ; i++)
     {
          tc = M[i]*vec3(pos2d,1);
          tc.st /= tc.z;
          w = texture2D(tex w, tc.st);
          gl_FragColor += w*texture2D(tex[i],
          tc.st);
          sum w += w;
     }
     gl FragColor /= sum w;
}
```

(그림 3) GLSL 프레그먼트 쉐이더



(그림 4) 블렌딩 함수 텍스처

5 결과

그림 5 는 본 시스템을 이용하여 촬영한 원본 이미지들이다. 웹캠으로 촬영한 각 이미지의 해상도는 1920 × 1080이다. 그림 6 은 사영변환을 적용하여 이미지를 병합하여 얻은 고해상도 이미지이고, 그림 7 은 블렌딩 후 일부를 잘라낸 최종 이미지이다. GLSL 프로그램을 이용하여 6000 × 2000 해상도의 고정된 이미지를 FBO (Frame buffer Object)에 렌더링하는 경우 약 44fps 의 시간이 소요되었다. (시스템: Ubuntu 11.10 / Intel® Xeon® X5550 @2.67GHz / RAM 12GB / NVIDIA Quadro FX 1800) 추후 웹캠을 이용해 비디오 스트림을 병합하는 경우에도 충분히 실시간으로 병합이 가능할 정도의 속도라고 볼 수 있다.

(그림 5) 카메라 어레이 원본 이미지들

(그림 6) 병합된 이미지

(그림 7) 블렌딩된 이미지

6 결론 및 연구계획

현재 웹캠의 비디오 스트림을 입력 받아 실시간으로 이미지를 병합하여 고해상도 이미지를 얻는 연구를 진행 중에 있다. 고정된 변환값을 사용하여 이미지를 병합하는 경우 부분적으로 어긋나는 부분이 있어 실시간으로 이를 보정해주는 방법을 연구할 계획에 있다. 또한 평면 사영 변환으로는 이미지를 병합하는 데 한계가 있어 다른 병합 알고리즘을 이용하는 방법에 대해서도 연구 중에 있다.

참고문헌

[1] J. C. Yang, M. Everett, C. Buehler and L. McMillan, "A real-time distributed light field camera," in *Proceedings of 13th Eurographics workshop on Rendering*, 2002.

[2] B. Wilburn, N. Joshi, V. Vaish, E.-V. Talvala, E. Antunez, A. Barth, A. Adams, M. Levoy and M. Horowitz, "High Performance Imaging Using Large Camera Arrays," in *Proceedings of Siggraph '05*, 2005.

[3] M. Brown and D. Lowe, "Recognizing panoramas," in *Proceedings of ICCV '03*, 2003.

[4] R. Szeliski, "Video mosaics for virtual environments," *IEEE Computer Graphics and Applications,* vol. 16, no. 2, pp. 22-30, 1996.

[5] R. Szeliski, Computer Vision: Algorithms and Applications, Springer, 2010.

[6] P. Heckbert, "Fundamentals of Texture Mapping and Image Warping," *Master's Thesis,* 6 1989.

[7] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools,* 2000.