특집논문 (Special Paper)
방송공학회논문지 제25권 제5호, 2020년 9월 (JBE Vol. 25, No. 5, September 2020)
https://doi.org/10.5909/JBE.2020.25.5.655
ISSN 2287-9137 (Online) ISSN 1226-7953 (Print)

실시간 렌더링을 위한 MPEG-I RVS 가속화 기법

안 희 준[a)‡], 이 명 진[b)]

MPEG-I RVS Software Speed-up for Real-time Application

Heejune Ahn[a)‡] and Myeong-jin Lee[b)]

요 약

자유시점 영상합성기술은 MPEG-I(Immersive) 표준에서 중요한 기술 중 하나이다. 현재 MPEG-I에서 개발하여 사용하는 RVS (Reference View Synthesizer) 프로그램은 다수의 시점의 컬러영상과 깊이영상을 바탕으로 임의시점의 영상을 생성하는 DIBR (Depth Information-Based Rendering) 프로그램이다. RVS는 기존의 DIBR이 갖는 깊이정보 전달의 문제를 컴퓨터 그래픽스의 메쉬 표면 방식으로 접근하여 이전 화소방식에 비하여 2.5dB 이상의 성능향상을 보이며 OpenGL을 사용하면 CPU에서 동작하는 코드보다 10배 이상의 속도를 보인다. 그러나 여전히 2개의 2k 해상도 입력 영상에서 0.75fps 정도의 비실시간 처리속도를 보인다. 본 논문에서는 현 RVS의 내부 구현을 분석하고 이를 바탕으로 1) OpenGL 버퍼와 텍스쳐 객체의 재사용 2) 파일 입출력과 OpenGL 실행의 병렬화 3) GPU 셰이더 프로그램과 버퍼 데이터 전송의 병렬화를 적용하였다. 그 결과 두 개의 2k 해상도 입력 영상의 처리속도를 34배 이상 가속하여 22-28fps의 실시간 성능을 확보하였다.

Abstract

Free viewpoint image synthesis technology is one of the important technologies in the MPEG-I (Immersive) standard. RVS (Reference View Synthesizer) developed by MPEG-I and in use in MPEG group is a DIBR (Depth Information-Based Rendering) program that generates an image at a virtual (intermediate) viewpoint from multiple viewpoints' inputs. RVS uses the mesh surface method based on computer graphics, and outperforms the pixel-based ones by 2.5dB or more compared to the previous pixel method. Even though its OpenGL version provides 10 times speed up over the non OpenGL based one, it still shows a non-real-time processing speed, i.e., 0.75 fps on the two 2k resolution input images. In this paper, we analyze the internal of RVS implementation and modify its structure, achieving 34 times speed up, therefore, real-time performance (22-26 fps), through the 3 key improvements: 1) the reuse of OpenGL buffers and texture objects 2) the parallelization of file I/O and OpenGL execution 3) the parallelization of GPU shader program and buffer transfer.

Keyword : MPEG-I, RVS (Reference View Synthesizer), OpenGL, Real-time Optimization, Free viewpoint.

a) 서울과학기술대학교 전기정보공학과(Dept. of Electrical and Information Engineering, Seoul National University of Science and Technology)
b) 한국항공대학교 항공전자정보공학과(Dept. of Electronics and Information Engineering, Korea Aerospace University)
‡ Corresponding Author : 안희준(Heejune Ahn)
E-mail: heejune@seoultech.ac.kr
Tel: +82-2-970-6543
ORCID: http://orcid.org/0000-0003-1271-9998
※ 본 연구는 서울과학기술대학교 2019년도 연구교수 지원 사업의 지원을 받았음.(This work was supported by SeoulTech Research Professor Program (2019)).
· Manuscript received July 6, 2020; Revised August 9, 2020; Accepted September 2, 2020.

Ⅰ. 서 론

MPEG-I (Immersive)는 실감 영상 포맷을 표준화하는 MPEG의 하부조직으로 최근 3DOF, 3DOF+, 6DOF 영상 압축과 포맷 등에 대하여 표준화를 하고 있다. 사용자에게 자유 시점 영상을 통해 끊어짐이 없는 뷰(view)를 제공하기 위해서는 아무리 많은 수의 카메라로 촬영하더라도 모든 시점의 영상을 제공하지는 못하므로, 사용자의 움직임에 실시간으로 뷰포트(viewport)를 렌더링하는 것이 중요하다[1][2][3]. 현재 MPEG-I의 가상 시점 영상을 생성하는 기법은 DIBR (depth image based rendering)[4]에 기반을 두고 있다. MPEG-I의 주 관심은 영상 압축 알고리즘과 포맷에 대한 것이지만, 깊이 기반 영상을 다루기 위하여 촬영 영상으로부터 깊이 정보를 추출하는 DERS (Depth Estimation Reference Software)[5]와 RVS (Reference View Synthesizer)[6] 등의 도구들을 개발하여 사용하고 개선하여 사용하고 있다.

RVS는 2018년도에 채택된 MPEG-I의 가상 시점 합성 표준 소프트웨어로, RVS는 메쉬표면방식의 기법을 사용하여. 기존의 화소기반의 방식에 비하여 성능이 높다. RVS는 깊이 정보와 화소정보를 표면 (surface)로 모델링하고 이를 바탕으로 그래픽스의 기하 변환 방식을 사용하는 알고리즘으로 기존의 방식보다 화질 면에서 2.5dB 이상 높은 것으로 보고되고 있다[7]. 또한 일반적으로 참고 코드 속도는 가장 중요한 요구사항은 아니지만, OpenGL 방식을 사용하였을 때 C++/CPU 구현에 비하여 10배 이상 빠른 처리속도를 보인다[3][4]. 그럼에도 불구하고 현재 수행 속도는 1.0fps 이하로 실시간 응용을 위해서는 절대적으로 수행 속도 개선이 필요하며, 이를 사용하여 연구를 하거나 시연 등을 하기 위해서는 실시간 성능을 보이는 시스템의 개발이 필수적이다.

본 논문의 최종목표는 소프트웨어적으로 6DOF가 지원되는 클라이언트 시스템을 구축하는 것으로 우선 RVS를 사용하여 2k영상의 실시간 가상시점 영상 재생을 목표로 하였다.이를 위하여 우선 RVS 구조를 분석하고 또한 각 단계별로 프로파일링을 수행하였다. 이를 바탕으로 OpenGL을 활용함에 있어서 비효율적인 요소들을 검출하고 이를 해결하기 위한 방안을 도출하고 구현하였다. 이렇게 구현된 결과는 MPEG-I 표준 테스트 영상에서 평가를 하였으며, 그 결과 2k 해상도의 ERP 영상에 대하여 약 22-28 fps의 실시간 성능을 확인하였다.

본 논문은 다음과 같이 구성되었다. 제 II절에서는 RVS 소프트웨어의 구조를 설명하고 프로파일링 결과를 제시한다. 이를 바탕으로 제 III 절에서 병목 요소와 원인을 분석하고 이를 해결하기위한 방안을 OpenGL 시스템과 응용을 고려하여 구체적으로 제시한다. 제 IV절에서 제안된 방식으로 바뀐 성능에 대하여 MPEG-I 표준 데이터 영상을 통하여 실험한 결과를 제시한다. 제 V절에서 논문의 결론과 추후 연구 방향에 대하여 기술한다.

Ⅱ. RVS 구조와 성능 분석

1. RVS 소프트웨어 구조

RVS[6][7][8]는 MPEG-I에서 2018년에 개발한 DIBR기반의 가상시점 합성 소프트웨어로 현재 평면 프로젝션 영상과 360도영상 모두의 가상시점 영상 생성을 제공한다. 소프트웨어 구조와 사용법에 대해서는 [6]에 제시되어 있고, 영상화질특성에 대해서는 참고 문헌 [7]과 [8]에 상세히 다루고 있다. 그 결과 이전 버전인 화소기반 방식인 VSRS[9]에 비하여 2.5dB 이상의 화질 향상이 있다고 보고 되었다[7]. 하지만 그 어디에도 수행속도에 대한 분석 내용은 보고되고 있지 않는다. 서론에서 언급한 바와 같이 MPEG-I 소프트웨어의 목적은 화질과 호환성에 초점이 맞추어져 있기 때문에 수행속도는 우선적인 관심 범위가 아니라고 할 수 있다. 하지만, 이를 사용하여 연구를 하거나 시연 등을 하기 위해서는 실시간 성능을 보이는 시스템의 개발이 필수적이다.

수행 속도를 개선을 위해서는 본래 시스템의 구조와 이해가 필수적이다. 그렇지 않고 가속화 요소를 기계적으로 찾는 것은 어려울 뿐 아니라 가속화 이후에 원래 소프트웨어가 갖는 바람직한 특성이 없어질 수 있기 때문이다. 그림 1은 기존 화소기반 DIBR방식과 RVS의 메쉬표면기반 방식을 비교하여 보여준다. 기존의 화소기반 방식은 생성하려는 시점에서의 깊이정보를 생성하는 단계에서는 전방향 매핑을 수행한 후에 화소값들을 얻어오는 과정에서 후방향 매핑을 사용하게 된다. 즉, 단계 1)에서 회소위치 (u,v)에서

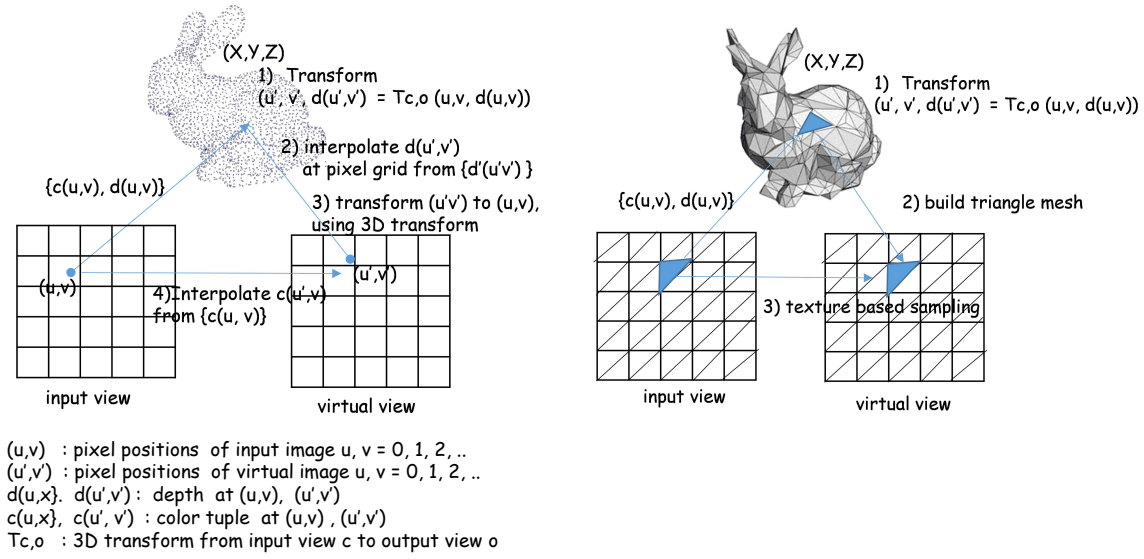

그림 1. 기존 화소기반 생성 방식(상)과 RVS에서 사용한 메시(표면)기반 방식
Fig. 1. Previous pixel-based virtual view synthesizer methods vs. Triangle mesh surface based method in RVS

의 좌표값과 깊이정보 d(u,v)를 바탕으로 카메라 파라미터를 사용하여 가상뷰에서의 위치 (u',v') 깊이값 d(u',v')을 계산한다. 단계 2)에서 (u',v')은 일반적으로 정수 값이 아니므로, 가상뷰에서 정수 값을 갖는 화소 위치의 깊이 값은 인터폴레이션을 통하여 얻어진다. 이렇게 얻어진 깊이값을 통하여 단계 3에는 대응되는 원영상에서의 화소 위치를 계산하고, 단계 4에서 인터폴레이션에 의하여 해당 색상 값을 가져올 수 있다. 이때 가려진 영역에 대한 처리가 복잡해지며, 두 차례의 전이 과정에서 계산상의 오차가 생기기도 한다. 반면 삼각형 메쉬로 구성한 표면을 넘기는 방식에서는 단계1)에서 전방향 화소점을 이동한 후 단계 2)에서 대응되는 삼각형을 계산하여 단계 3)에서 삼각형면마다의 특성을 이용하여 화소를 생성하는 방법을 사용함으로써 보다 효과적인 처리가 가능해진다.

현재 RVS는 리눅스와 윈도우즈 모든 환경에서 사용이 가능하며 현재 OpenGL을 사용하는 옵션과 사용하지 않는 옵션이 존재한다. 본 논문에서는 OpenGL[9]을 사용하여 가속화 하는 것을 다루므로, 해당 옵션이 선택된 경우에 대해서만 설명한다. OpenGL은 그래픽스 프로세서 구동을 위한 프로그래밍 환경과 API 표준으로 관련 산학전문가들이 표준화하고 있고, 실제로 거의 모든 GPU에서 제공하고 있는 표준이다. OpenGL은 물체를 메쉬 표면으로 모델링하고 이를 바탕으로 2차원으로 렌더링하는 것이 주목적이다. 따라서 6DOF의 기본 취지와 잘 맞는다고 볼 수 있다.

그림 2는 RVS의 동작의 중요한 단계를 순서도로 나타낸

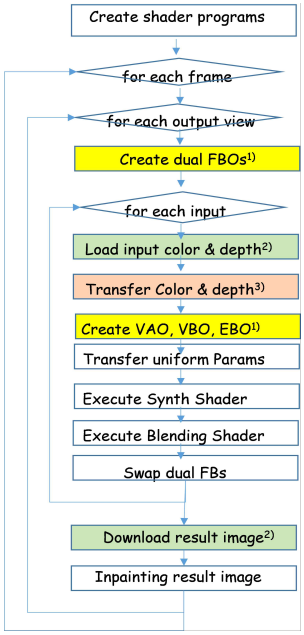

그림 2. RVS 프로그램[7]의 수행 순서도 (1,2,3는 III 절에서 제시한 문제 점 및 가속화 방법)
Fig. 2. RVS software[7] execution flows (1, 2, and 3 denote the issue and acceleration method in section III respectively)

것이다. RVS는 CPU단의 작업과 GPU 단의 작업으로 구분되는데, CPU에서는 입력 컬러데이터와 깊이 데이터를 읽고 카메라들 설정정보를 읽으며 이를 GPU측의 셰이더 프로그램에 전달하고 셰이더 프로그램을 설정하고 실행한다. 합성된 최종 결과는 받아서 in-painting 등을 수행하고 이를 저장한다. OpenGL은 vertex 셰이더에서 입력화면의 모든 화소들을 메쉬의 vertex 위치로하여 (VAO: vertex array object, VBO: vertex buffer object, EBO: element buffer object) 출력 뷰에서의 vertex 위치를 계산하고, geometric 셰이더 에서 삼각형 요소들에 대한 변형 정도를 통하여 출력 삼각형 요소의 화질을 예측하고, 최종적으로 입력이미지를 텍스춰로 사용하여 렌더링을 fragment 셰이더에서 frame buffer (FBO: frame buffer object)로 렌더링을 통하여 뷰를 변환한다. 그림에서 각 1), 2), 3)으로 표시한부분은 다음 절에서 가속화시에 가속화 방식을 의미한다.

그림 3은 두 개의 셰이더 프로그램의 구성 셰이더와 입출력을 보여주고 있고, 각 셰이더에서 사용하는 버퍼의 종류 및 크기와 처리 요소수를 표 1에 표시하였다. GPU측에서 처리되는 내용은 실제 합성의 핵심내용으로 Synth(합성) 셰이더와 Blending 셰이더의 두 개의 셰이더 프로그램으로 구성되어 있다. 이 과정은 입력 별로 차례로 수행된다. 합성 셰이더 프로그램은 입력 컬러와 깊이영상을 사용하여 화소단위의 3차원 메쉬를 구성하고, 이를 시점 변화를 통하여 목적 시점에서의 표면으로 변환하는 작업을 수행한다. 이를 위하여 버텍스(vertex), 기하(geometric), 프래그먼트(fragment) 셰이더를 구현한다. 또한 블렌딩 프로그램은 버텍스 와 프래그먼트 셰이더 만을 사용하며, 여러 입력을 거리와 변형 기울기 정도 (quality)에 바탕으로 가중치를 주어 누적하는 작업을 수행한다. 이 과정이 가상 시점을 구성하는 입력에 대하여 각기 수행되므로 계산양은 입력의 개수에 비례한다. 버텍스 셰이더는 입력 영상의 각 화소점을 입력 카메라 T_c와 출력 카메라 T_o의 3차원 좌표 변환을 사용하여 $T_{c,o} = T_o T_c^{-1}$변환을 수행하여 가상화면에서의 버텍스로 $\{v_i^r, i = 1, N\}$ 이동한다. 여기서 N 은 모든 화소가 되므로 입력영상의 해상도와 같다.

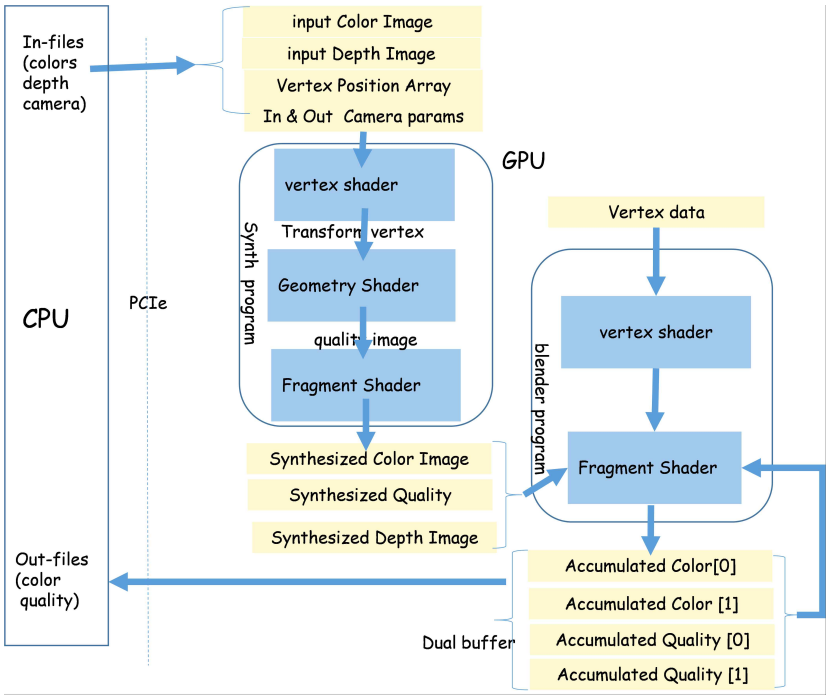

그림 3. RVS 프로그램의 구조 (셰이더 부분 중심)
Fig. 3. RVS software structure focusing on the OpenGL shader processing

표 1. RVS 세이더의 자원과 계산 단위
Table 1. RVS shaders resource and computation elements

	Relevant shader	Resource used	# of computation elements
Synth	vertex	VBO: num of input pixels EBO : 3*2*# of input pixels	# of input pixels
	geometry		2 * # of pixels
	fragment	Input 2*3*# of pixels (image) 2*# of pixels (depth) FBO (dual) 2*3*# of pixels (image) 2*# of pixels (quality)	~ # of output pixels
Blending	vertex	VBO: 4* vec4	4 (quad)
	fragments	(use the same FBO of Synth	~ # of output pixels

$$v_i^{'} = T_{c,o} v_i \quad (1)$$

기하 세이더는 이 삼각형의 변형정도를 바탕으로 출력 삼각형의 화소 화질을 계산한다. 간략히 설명하자면 입력 화면의 메쉬를 구성하는 직각 삼각형이 가상화면으로 매핑되었을 때, 그 크기가 커지거나 변형이 크게 일어나게 되면 해당된 영역의 생성 화질이 감소하며, 그 반대의 경우는 화질이 높게 유지된다. 자세한 설명은 해당 참고 논문 [6]을 참고 하기 바란다. 입력 삼각형과 버텍스 세이더에 의하여 에서 얻어진 가상뷰의 삼각형 $t_k = \{v_i, v_{i+1}, v_{i+2}\}$과 $t_k^{'} = \{v_i^{'}, v_{i+1}^{'}, v_{i+2}^{'}\}$의 변환화질은 다음과 같이 추정된다.

$$q(t_{k,c}^{'}) = f(t_k^{'}, t_k) \quad (2)$$

함수 f( )는 두 개의 대응되는 삼각형 사이의 모양과 크기 변화로부터 해당 영역의 화질을 계산하는 함수로 [6]에 정의된다.

프레그먼트 세이더에서는 입력 영상을 텍스쳐 정보로 사용하여 합성된 영상을 생성한다. 그 결과 확보된 정보는 출력 시점에서의 합성영상, 깊이영상, 그리고 화질 영상이 얻어진다. 블렌더 세이더 프로그램은 버텍스 세이더의 역할은 사실상 없고 프레그먼트 세이더에서 깊이 정보와 화질 정보를 이용하여 가중치를 계산하여 다음 식 (3)과 같이 누적 블랜딩을 수행한다.

$$c(u^{'}, v^{'}) = \sum_{c=1}^{c} W(q(u_c^{'}, v_c^{'}), d(u_c^{'}, v_c^{'})) c(u_c^{'}, v_c^{'}) \quad (3)$$

여기서 $c(u^{'}, v^{'})$과 $c(u_c^{'}, v_c^{'})$는 출력과 입력영상 c의 화소 값이고 $W(q(u_c^{'}, v_c^{'}), d(u_c^{'}, v_c^{'}))$는 각 입력별 가중치로 거리와 앞서 계산된 화질변화값에 따라 정해진다[6].

2. 실행 속도 성능 평가

본 논문에서는 OpenGL을 사용한 구성에서 프로파일링을 수행하였다. Synth 세이더와 Blending 세이더는 매 입력 마다 실행되므로 실행시간은 입력의 개수에 선형적으로 비례한다. 가장 실용적인 상황을 고려하여 생성하는 위치에 가까운 2개의 입력만을 사용하는 경우에 대하여 실험하였다. 구체적으로 실험에 사용한 환경은 GTX1070 그래픽 카드와 Ubuntu Linux를 사용한 Dell T630 에서 MPEG-I Classroom ERP 영상을 (8bit 2k 해상도로 변환) 사용한 결과를 표 2에 제시하였다. 그 결과는 Synth 세이더 프로그램 전체의 실행시간의 절반 정도를 차지하며, 데이터 입출력 및 후처리 시간이 나머지를 차지하는 것으로 확인된다. 4k 원영상에서도 실험하였지만 이러한 경향은 동일하였다. 다음 절에서 구체적으로 다루겠지만 각 세이더에서 사용하는 버퍼들과 자료를 매번 새로 세팅하고 있고, 파일 입출력 부분이 동일 스레드로 처리되는 점 등 실행 속도에 관련된 고려가 되어 있지 않음을 확인 할 수 있다.

표 2. RVS OpenGL 기반 수행 시의 프로파일링 결과 (두 개의 입력을 사용한다고 가정하므로 셰이더는 두 번씩 실행 됨)
Table 2. Execution time for each steps (each shader program is executed twice because two input views are assumed)

Processing step	exe. time (ms)	proportion (%)
1. file loading and format conversion	101.2	8%
2. synthesis shader	374.6	46%
1) texture buffer	12.5	
2) texture transfer	124.4	
3) create VAO/VBO/EBO	147.2	
4) run synthesis shader	11.8	
3. blender shader	0.1	1%
1) setup blending VAO/VBO	0.01	
2) run blending shader program	0.09	
4. result	523.7	32%
1) download results/ rendering	39.9	
2) in-painting	290.3	
3) save file	196.6	
Total	1376.3	

III. RVS 실행 속도 개선 방법

일반적으로 실행시간 최적화를 할 경우에는 알고리즘을 변화시키는 방법과 시스템/구현 레벨 최적화를 하는 방법이 있다. 일반적으로 OpenGL[10] 셰이더 프로그램에서 계산양은 주로 버텍스 셰이더의 실행시간은 버텍스 수에 비례하고, 프레그먼트 셰이더는 출력 해상도에 비례한다. 그러나, 이 두 가지 요소를 변화 시키는 것은 현재 RVS에서는 알고리즘 자체를 변화시키는 것이므로, 출력 영상의 화질이 달라지게 된다. 따라서 본 논문에서는 시도하지 않았으나 향후 연구가 필요한 요소이다. 본 논문에서는 앞 절의 RVS 구조 분석과 프로파일링 결과 그리고 OpenGL 시스템의 특성을 고려하여 시스템레벨의 최적화만을 시행하였다. 구체적으로 앞 절의 분석내용을 바탕으로 다음과 같은 개선요소를 파악하였고 이에 대응되는 해결책을 제시한다.

첫째, 현재 RVS는 셰이더에 사용하는 자원들의 생성을 매 프레임마다 새롭게 생성하고 있다. 첫째로 Synth 셰이더 프로그램의 VAO (vertex array object), VBO (vertex buffer object), EBO (element buffer object)를 생성하고 관련 데이터 전송을 수행한다. 특히 Synth 셰이더의 버텍스는 모든 화소가 되기 때문에 해상도의 2배만큼 정보양이 사용 되는데, 좌표값을 셰이더에서 계산에 의하여 생성하지만, EBO를 전달하여주는 부분이 여전히 필요하다. 또한, Blending 셰이더의 VAO, VBO를 매 화면마다 새롭게 생성하고 전송한다. 또한 Synth 셰이더의 결과와 Blending 셰이더의 결과의 누적치를 저장하는 프레임 버퍼 오브젝트를 매 화면마다 생성한다. 이렇게 중복적으로 보이는 생성과정을 유지하는 것은 매 화면마다 다른 해상도나 카메라 변수를 사용하는 경우에 대응하기 용이하게 해준다. 하지만, 일반적으로는 입/출력 영상의 해상도가 바뀌는 경우는 매우 드문 경우이나 그러한 응용이 있을 수 있다고 가정하였다. 따라서 본 논문에서는 그림 4의 의사코드처럼 이러한 경우도 처리가 가능하도록 이 부분을 입/출력의 크기가 변하는 경우에만 재설정하도록 하여 문제를 해결하였다. 특히 Synth 셰이더의 EBO세팅은 상당한 시간이 소모되므로 수정 시 효과가 매우 크다.

Before optimization	After optimization
reconfigure new OpenGL objects execute the shader program	if parameter changed release OpenGL objects (VAOs, EBOs) reconfigure new OpenGL objects execute the shader program

그림 4. 세이더 오브젝트 재사용을 위한 수정
Fig. 4. Shader Object reuse mechanism

둘째, CPU 측에서 파일에서 입력을 로딩하는데 걸리는 작업과 GPU작업이 병렬화되어 있지 못하다. (로딩 시간 20% 이상). 이는 멀티 스레드와 동기화 기법을 사용하여 GPU 관리 스레드와 데이터 로딩 스레드를 구분하여 처리하였다. 병렬화 후의 실행은 그림 5에 PBO (Pixel Buffer Object) 적용과 함께 제시하였다. 이 병렬화에의하여 CPU 셰이더 실행시간 이외의 추가적인 CPU 실행 시간을 거의 필요치 않게 되었다.

셋째, 이미지와 깊이 정보를 텍스쳐로 전송 시에 사용하는 방식이 비동기적인 전송방식을 따르고 있지 못하다. 동기 전송의 경우는 CPU측 OpenGL의 동작과 셰이더의 실행

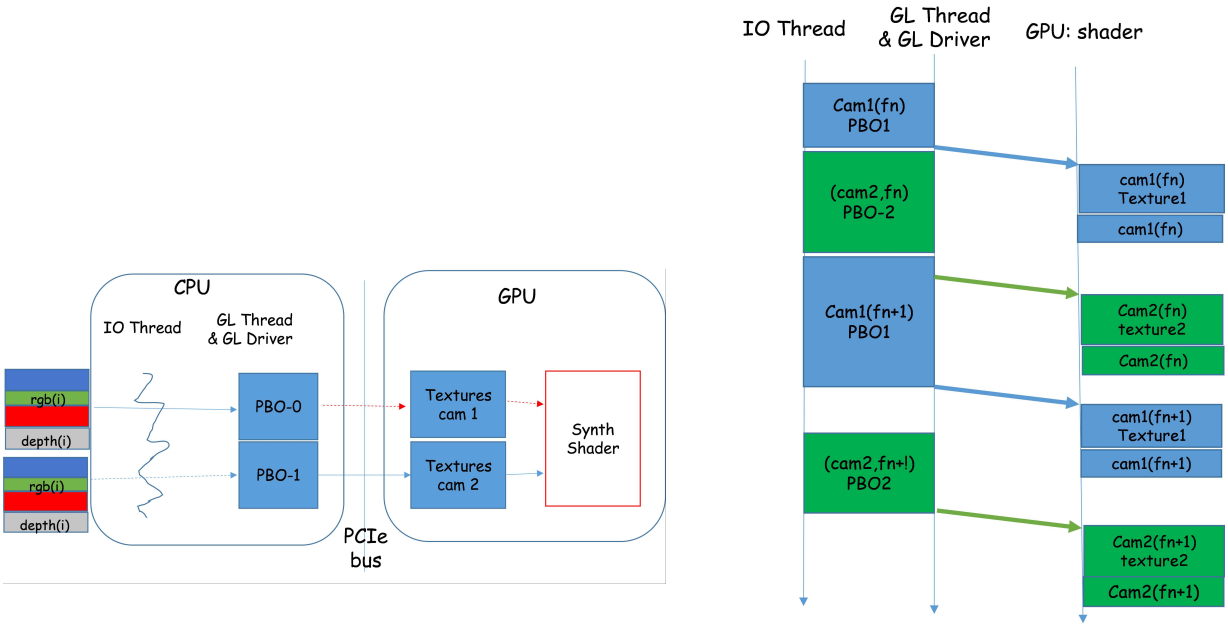

그림 5. PBO(pixel buffer object 또는 page locked) 메모리를 사용한 메모리 복사 절약 [6] 및 병렬화
Fig. 5. PBO(pixel buffer object or page locked) memory based memory copy reduction [6] and parallelization

이 병렬화 돼지 못한다는 점이다. 기존 자료들 [10, 11]에 따르면 PBO (Pixel Buffer Object)를 Dual로 사용하여 IO 작업과 텍스쳐 전송 및 텍스쳐 데이터 사용 즉, 셰이더 실행을 병렬화 한다. 본 논문의 시스템에서는 다중 입력을 사용하므로 각 입력 별로 이중 버퍼를 사용하는 효율 향상은 크지 않다. 따라서 각 입력 별로 PBO를 하나씩 생성하고 또한 카메라 별로 텍스쳐 버퍼를 생성한다. 이를 통하여 GPU에서의 셰이더 실행과 전송을 병렬화 할 수 있다. PBO를 사용하는 또 한 가지 이득은 사용자메모리에서 드라이버 메모리로 복사하는 과정을 IO 스레드에서 사용가능하여 복사 횟수를 줄일 수 있다는 점이다.

넷째, 다중 GPU 환경에 대하여 확장성이 고려되어 있지 못하다. 최근 까지는 OpenGL은 하나의 GPU에서만 실행하거나 NVIDIA의 경우 RTI 방식으로 그래픽 카드를 연결하여야 가능한 것으로 알려져 있다. 최근 다중 GPU를 활용하는 OpenGL 방식에 대한 지원이 시도되고 있는데 현재는 NVIDIA QUADRO에서만 일부 적용이 가능한 것으로 알려져 있다[11]. 본 논문에서 현실적으로 적용할 환경이 제공되고 있지 않았고, 일반적으로 연구자나 개발자들도 이러한 환경을 구축하기가 어려울 것으로 판단되어 다중 GPU 활용은 본 연구에서 적용하지 못하였다.

이외에 영상 결과를 파일로만 저장하도록 구현된 것을 바로 화면에 출력이 가능하도록 수정하였다. 이때 입력이 YUV 포맷이기 때문에 이를 RGB로 변경하는 셰이더 프로그램을 추가로 작성하였다. 이 부분은 추가 실행 시간이 수 millisecond 정도의 추가 실행시간이 필요하다. 현 구현에는 가림 영역을 채우기 위한 인패인팅은 구현 되어 있지 않고 향후 이를 추가할 계획이다. 실험결과 시차가 크지 않은 경우에 인패인팅이 필요한 영역은 매우 작은 것으로 확인 되었다.

Ⅳ. 실험 결과

본 실험은 혹시 ERP(Equi-rectangular projection) 영상과 평면투영 영상에서의 차이가 있는지를 확인하기 위하여, MPEG-I에서 제공하는 Classroom[12] ERP 영상과 Technicolor-Painter 평면투영 영상을 사용하였다 (Fig. 6). 앞서 프로파일링 했을 때와 같이 Classroom영상은 4k (4096x2048) 10bit 영상을 2k (2048x1024) 8bit영상으로 변환하고,

그림 6. 실험 결과 영상 (Classroom영상 입력카메라 v0, v2, 출력 v1 영상 (12 번째 화면), Painter 영상 입력 카메라 v0, v2, 출력 v1 영상 (61 번째 화면))
Fig. 6. The synthesized output sample using the proposed method (12th frame of v1 view from ClassRoom v0 and v2 input view sequence & 61st frame of v1 view from Painter v0 and v2 input view sequence)

TechnoPainter 영상은 2048x1088 10bit영상을 2048x1024 8bit으로 변환하여 해상도를 동일하게 하여 실험하였다. Classroom 과 TechnocolorPainter 영상은 각기 200 프레임과 300프레임으로 구성되어있으며, 입력카메라는 v0부터 v14까지 총 15개로 구성되어있다. 실험은 다양한 시점에서 수행하였으나 결과는 차이가 없기 때문에 v0와 v1 시점을 사용하여 v1시점을 생성하는 실험에 대한 결과를 대표적으로 제시하였다. 두 개의 시점을 사용하는 경우는 실험적으로 인패인팅(inpainting)이 필요한 가림 영역은 거의 없었다. 합성된 v0 시점과 원v0 입력과 거의 동일한 결과를 얻었으나, 본 연구는 렌더링 합성의 성능에 대한 것이 아니므로 화질에 대한 분석은 참고문헌 [8]을 참고하기 바란다.

Table3에서 각 단계별 시간 감소를 원 프로그램과 비교하여 제시하였다. 특히 각 단계별 앞서 제안한 가속화 방식을 매칭하여 표시하였다. Classroom ERP 영상과 TechnocolorPaninter 평명투영영상의 경우가 거의 동일한 결과를 보임을 알 수 있다. 버텍스 셰이더에서 ERP 의 경우 좌표변환을 위하여 삼각함수를 사용하고 있으나 이에 따른 셰이더의 계산시간의 차이는 크지 않은 것으로 확인할 수 있었다. 따라서 입력 영상의 해상도 이외에는 크게 계산 시간

표 3. 최적화를 통한 실행 시간 변화 (ms)
Table 3. Execution time change after optimization (ms)

processing step	Classroom		TechnocolorPainter		note	accel. method
	baseline	Optimized	baseline	Optimized.		
1. file loading and format conversion	101.2	< 0.01	101.2	< 0.01	parallel	2
2. synthesis shader	374.6	16.5	373.8	16.4	# of in views	
1) texture buffer	12.5	< 0.01	12.5	< 0.01		1
2) texture transfer	124.4	15.4	124.4	15.4	*	1
3) create vertex array	147.2	(4.1)	147.2	(4.1)	once	1
4) run shader	11.8	0.20	11.0	0.19		3
3. blender shader	0.1	0.09	0.1	0.09	# of in views	
1) buffer setup	0.01	<0.01	0.01	<0.01		1
2) run shader	0.09	0.09	0.09	0.09		1
4. result	523.7	0.01	523.7	0.01		
1) download results/ rendering	39.9	0.01	39.9	0.01		none
2) in-painting	290.3	<0.01	290.3	<0.01		None
3) save file	196.6	<0.01	196.6	<0.01		None
Total (w. 2 inputs)	1376.3	~38	1375.5	~37	~26 fps	

에 영향을 주는 요소는 없는 것은 볼 수 있다. GTX 1070 환경 Ubuntu에서 기존과 최적화 후에 현재는 26 fps정도의 성능을 보인다.

Table 4는 본 실험에서 2k 입력영상시의 GPU부하는 GPU 부하가 각기 약 20W에서 54W (최대 230W)로 증가함을 확인하였다. 현재 사용량은 40%에 못 미치는 상황으로 아직 가속화 여지가 있다. 일반적으로 셰이더 프로그램은 자원이 확보되어야 실행이 가능하므로 셰이더 입력 데이터의 의존성관계로 인한 비명시적 동기화현상을 확인해야한다. 마지막으로 CPU에서만 구현되어 있는 인페인팅(in-painting)부분은 생략하였다. 향후 이 부분도 구현상 보완이 되어야 할 요소이다. 참고로, 두 개의 비교적 근접한 위치의 카메라 입력을 사용하는 경우 실험적으로 가림(occlusion)이 발생하지 않기 때문에 인페인팅이 필요한 영역은 거의 무시할 정도로 발생하였다.

표 4. 'nvidia-smi' 명령으로 본 GPU사용량 변화
Table 4. GPU resource utilization before and after Optimization (using 'nvidia-smi' tool)

Resource	Classroom		TechnocolorPainter	
	baseline	Opt	baseline	opt.
Power (Watt) (Utilization %)	23/230 1	54/230 39	23/230 1	54/230 39
GPU memory (MiB)	839/8118	787/8114	839/8118	787/8114

V. 결 론

RVS 코드 구조 분석과 프로파일링을 통하여 본 논문에서 제안한 병렬화와 버퍼재활용 기법으로 RVS 프로그램은 2개의 2k 입력 영상을 사용하는 경우에 실시간 동작을 하도록 최적화 되었다. 좀더 자세한 OpenGL 프로파일링을 통해 세부적인 최적화를 하면 좀 더 높은 성능도 가능하리라 예상된다. 하지만 완전한 클라이언트 렌더링 시스템이 되기 위하여는 몇 가지 보완해야 할 사항들이 있다. 첫째로 현재 입력데이터는 YUV 포맷의 원본영상을 사용한다. 일반적으로는 동영상 압축된 영상을 사용하는 것이 실용적이며, 이를 위해서는 소프트웨어 코덱으로 여러 개의 동영상을 디코딩하는 것은 자원 소모가 많으므로 가급적 PC 환경에서는 NVIDIA 비디오 디코더를 사용하는 것이 바람직해 보인다. 디코딩 결과를 효율적으로 사용하기 위해서는 디코딩된 컬러와 깊이 정보를 셰이더에 입력 버퍼로 바로 전달하는 방식을 확보할 필요가 있다. 또한 같은 실험을 마이크로소프트 윈도우즈 OS상에서 도 실행하였을 때의 결과는 현재 는 리눅스보다 3배에서 5배가량 느리다. 윈도우 OS의 경우 DirectX기술과의 경쟁으로 일반적으로 드라이버의 성능에 문제가 있다는 것이 관련 전문가 그룹의 의견이나 구체적인 확인이 필요한 상황이다.

참 고 문 헌 (References)

[1] G. Lafruit, D. Bonatto, C. Tulvan, M. Preda and L. Yu, "Understanding MPEG-I Coding Standardization in Immersive VR/AR Applications," in SMPTE Motion Imaging Journal, vol. 128, no. 10, pp. 33-39, Nov.-Dec. 2019.

[2] G. Park, "Trend of free-view point video and A case study of extensible view point selection," Information & communications magazine, Vol. 36, No. 12, pp. 3-9, 2019.

[3] J. Yang, M. Song, G. Park, "Implementation of Integrated Player System based on Free-Viewpoint Video Service according to User Selection," J. of Broadcasting Engineering, Vol. 25, No. 2, 2020.

[4] C. Fehn, "Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV," Stereoscopic Displays and Virtual Reality Systems XI. Vol. 5291. International Society for Optics and Photonics, 2004.

[5] S. Rogge, D Bonatto, J. Sancho, R. Salvador, E. Juarez, A. Munteanu, and G. Lafruit, "MPEG-I Depth Estimation Reference Software," In IEEE 2019 International Conference on 3D Immersion (IC3D) pp. 1-6, December, 2019

[6] S. Fachada, D. Bonatto, A. Schenkel and G. Lafruit, "Depth image based view synthesis with multiple reference views for virtual reality," (3DTV-CON), Helsinki, 2018.

[7] B. Kroon, G. Lafruit, "Proposed update of the RVS manual," ISO/IEC JTC1/SC29/WG11, w18068, Oct. 2018.

[8] H.-H. Kim, J.-G. Kim, "Performance Analysis on View Synthesis of 360 Videos for Omnidirectional 6DoF in MPEG-I" J. of Broadcasting Engineering, Vol. 24, No. 2, 2019.

[9] T. Senoh, K. Yamamoto, N. Tetsutni, H. Yasuda, and K. Wenger, "View synthesis reference software (VSRS) 4.2 with improved in-painting and hole filling, " ISO/IEC JTC1/SC29/WG11, M40657, Apr. 2017.

[10] OpenGL Programming Guide, 9th Edition. ISBN 978-0-134-49549-1.

[11] Cozzi, P., & Riccio, C. (Eds.). (2012). OpenGL insights. CRC press.

[12] B. Kroon, "ClassroomImage: A frame of ClassroomVideo with less noise and more views," ISO/IEC JTC1/SC29/WG11, m44762, Oct. 2018.

저 자 소 개

안 희 준

- 2000년 2월 : KAIST 전자전산학부 박사
- 1997년 2월 ~ 2002년 7월 : LG전자 차세대 단말연구소 선임연구원
- 2002년 8월 ~ 2004년 1월 : TmaxSoft 연구소 책임연구원
- 2004년 2월 ~ 현재 : 서울과학기술대학교 정보통신대학 교수
- ORCID : http://orcid.org/0000-0003-1271-9998
- 주관심분야 : 컴퓨터비젼, 딥러닝, 컴퓨터통신

이 명 진

- 2001년 8월 : KAIST 전자전산학부 박사
- 2001년 3월 ~ 2004년 2월 : 삼성전자 System LSI 사업부 책임
- 2004년 3월 ~ 2007년 2월 : 경성대학교 전기전자공학전공 조교수
- 2007년 3월 ~ 현재 : 한국항공대학교 항공전자정보공학부 교수
- ORCID : https://orcid.org/0000-0002-3136-2819
- 주관심분야 : 영상통신, 영상처리, 임베디드시스템