석사학위논문

VTK.js의 확장을 통한 웹 기반 볼륨 가시화 연구

2022년

한 성 대 학 교 대 학 원

정보컴퓨터공학과

컴 퓨 터 공 학 전 공

이 윤 호

석 사 학 위 논 문
지도교수 계희원

VTK.js의 확장을 통한 웹 기반 볼륨 가시화 연구

Web-based volume visualization study through extension of VTK.js

2022년 6월 일

한 성 대 학 교 대 학 원

정 보 컴 퓨 터 공 학 과

컴 퓨 터 공 학 전 공

이 윤 호

석 사 학 위 논 문
지도교수 계희원

VTK.js의 확장을 통한 웹 기반 볼륨 가시화 연구

Web-based volume visualization study through extension of VTK.js

위 논문을 공학 석사학위 논문으로 제출함

2022년 6월 일

한 성 대 학 교 대 학 원

정 보 컴 퓨 터 공 학 과

컴 퓨 터 공 학 전 공

이 윤 호

이윤호의 공학 석사학위논문을 인준함

2022년 6월 일

심사위원장 김진모 (인)

심 사 위 원 계희원 (인)

심 사 위 원 이지은 (인)

국 문 초 록

VTK.js의 확장을 통한 웹 기반 볼륨 가시화 연구

한 성 대 학 교 대 학 원
정 보 컴 퓨 터 공 학 과
컴 퓨 터 공 학 전 공
이 윤 호

의료영상 데이터를 이용하여 볼륨 가시화를 할 때, 대부분의 볼륨 가시화를 수행하는 응용프로그램들은 운영체제 및 특정 컴퓨터 장치에 제약을 받는다. 이러한 문제점을 해결하고자 운영체제와 특정 컴퓨터 장치로부터 상대적으로 자유로운 웹 환경에서 볼륨 가시화 응용프로그램을 웹 기반 그래픽 프레임워크인 VTK.js를 이용하여 구현하였다. VTK.js에서는 가장 핵심적인 기능인 볼륨 가시화를 구현할 때 셰이더 프로그램을 이용한다. 본 논문에서는 셰이더 프로그램을 수정해 기존에 없던 기능을 확장하여 더 많은 기능을 사용자에게 제공할 수 있음을 보인다. 그 예시로 날카로운 모양의 전이함수를 이용해 볼륨 표면을 가시화할 때 발생하는 노이즈를 제거하는 기법인 전-적분(pre-integration)전이함수 기법과 인체조직의 표면을 복잡한 과정 없이 간단하게 볼륨 가시화할 수 있는 기울기 추정을 통한 표면 투명화 기법을 구현하였다. 또한 고품질의 음영을 생성하는 환경광 폐색 기법을 증분 알고리즘과 Web Assembly를 이용하여 구현하였다. 이 세가지의 기법을 VTK.js에서 제공하는 기능인 셰이더 프로그램에 통합하는 데 성공하였다.

【주요어】 볼륨 가시화, VTK.js, 전-적분 전이함수, 기울기추정을 통한 표면 투명화, 환경광 폐색, Web Assembly

목 차

표 목 차

그 림 목 차

Ⅰ. 서론

1.1 연구 배경 및 목적

의료영상 데이터를 가시화하는 방법으로는 볼륨 가시화 방법이 일반적이다. 기존의 볼륨 가시화 시스템은 개별 컴퓨터에 각각 설치되므로 소프트웨어 제조사 입장에서는 각각의 컴퓨터 프로그램에 대해 유지보수 비용이 발생한다. 또한, 응용프로그램이 다양한 하드웨어와 운영체제에서 동작하려면 각각 별도의 프로그램을 제작해야 하는 문제가 있다. 이를 해결하는 방법으로는 웹 기반의 응용프로그램이 주목받고 있다. 웹 응용프로그램은 대부분 운영체제의 영향을 받지 않으며 안정적인 실행환경을 제공한다. 설치와 같은 추가적인 환경 설정 없이 웹 브라우저를 사용하는 모든 장치는 이 응용프로그램을 사용할 수 있다. 의료영상 데이터를 가시화하기 위해서 VTK.js[Kitware, 2017]를 기반으로 응용프로그램을 개발하려 한다. C++로 제작된 그래픽 프레임워크인 VTK는 세계적으로 널리 사용되고 있으며, 이 VTK를 자바스크립트로 구현 언어만 변경한 웹 기반 그래픽 프레임워크가 VTK.js이다. 따라서 VTK.js는 의료영상 가시화에 사용되는 기능인 볼륨 가시화, 다중 평면 가시화 등 대부분의 기능을 지원하며 코드의 재사용성이 용이하고 생산성을 크게 높일 수 있다. 기존의 VTK.js는 GPU를 활용할 수 있는 WebGL[Parisi, T., 2012]을 이용하며 이는 셰이더 프로그래밍 기법[Fosner, R., 2003]을 지원한다. 이러한 방식으로 볼륨 가시화를 수행하면 일반적인 OpenGL을 이용하는 방식과 거의 동일한 성능으로 볼륨 가시화가 가능하다. 본 연구는 기존의 VTK.js에 별도의 기능을 추가하는 방법을 연구하였다. 셰이더 프로그램을 추가, 수정하여 사용자가 원하는 기능으로 시스템을 확장하고 Web Assembly[Haas, A., 2017]를 이용하여 CPU에서 동작하는 기능의 가속화를 보이는 것이 이 연구의 목표이다. 실제로 그 가능성을 보이기 위해 두 가지 기능인 전-적분(Pre-Integration) 전이함수[Roettger, S., 2003]를 사용한 볼륨 가시화 기능(이하 전-적분 기능)과 기울기 추정을 통한 표면 투명화 기능(Gradient Opacity 이하 표면 투명화 기능)[Lum, E, B., 2004]을 추가하였다. 또한, 환

경광 폐색(Ambient Occlusion)[Pharr, M., 2004] 기법을 적용하기 위해 증분 알고리즘과 Web Assembly를 이용하여 환경광 폐색에 필요한 사전작업인 마스크 볼륨 생성 단계를 가속화 하는 기능을 추가하였다. 전-적분 기능은 볼륨 가시화에 사용되는 색상 및 투명도 전이함수(transfer function)에 수치 적분을 적용하는 기법이다. 극단적으로 날카로운 형태를 가진 그래프를 가지는 전이함수를 이용하여 볼륨을 가시화하면 표면에 에일리어싱으로 인한 노이즈가 발생하는데, 전-적분 기능을 이용하면 이를 완화할 수 있다. 표면 투명화 기능은 볼륨 데이터에서 피부 및 뼈와 같은 조직의 경계를 찾아 경계의 표면을 강조하고 조직 내부를 투명하게 하여 관찰하는 기법이다. 각 복셀 들의 위치에서 기울기(Gradient)를 계산하여 기울기의 크기가 작은 경우, 해당 복셀은 균질한 조직의 내부에 있다고 판단할 수 있다. 이때 불투명도(alpha) 값을 감소시켜 투명하게 만들면 상대적으로 기울기가 큰 조직 경계 표면이 강조되어 피부, 또는 장기의 표면을 명확하게 가시화할 수 있다. 환경관 폐색 기법은 전이함수가 적용된 볼륨 가시화 결과물에서 움푹 파인 곳을 미리 탐색하여 마스크 볼륨에 기록하고, 마스크 볼륨을 기반으로 움푹 파인 곳의 음영을 강조하여 사실적인 그림자 표현이 가능해진다.

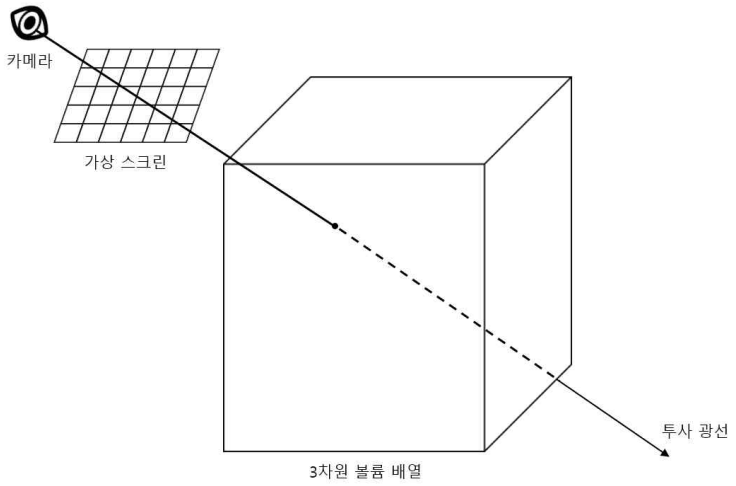
1.2 논문의 구성

본 논문의 구성은 다음과 같다. 2장에서는 볼륨 가시화의 기본적인 설명과 현재 웹 환경에서의 그래픽 가시화에 관한 연구 동향을 설명한다. 3장에서는 주로 VTK.js의 구조 및 VTK.js가 사용하는 셰이더 프로그램에 대한 설명이 뒤따른다. 4장에서는 본 논문에서 사용된 전-적분 전이함수 기능에 대하여 설명하고, 5장에서는 기울기 추정에 따른 표면 투명화에 관해 설명한다. 6장에서는 황경광 폐색을 이용한 고품질 음영 생성 방법에 관해 설명한다. 7장에서는 전-적분 기능, 표면 투명화, 환경광 폐색을 구현한 알고리즘과 실험 결과가 설명되어 있으며, 마지막으로 8장에서는 본 논문의 결론 대해서 기술한다.

II. 관련 연구

2.1 볼륨 가시화

볼륨 가시화[Drebin, R, A., 1988]는 삼차원 배열형식으로 되어있는 삼차원 의료영상 데이터를 이용하여 영상을 생성하는 가시화 방법이다. 예를 들어 의료영상 이미지를 촬영하는 대표적인 방식인 CT 촬영은 인체 내부의 X-선 흡수도, 즉 밀도 값을 여러 장의 이차원 영상으로 기록하게 된다. 뼈는 인체 조직 중 가장 단단한 조직이며 상대적으로 무른 조직 중 하나인 근육과 대비된다. 각각의 이미지 여러 장을 높이 방향으로 쌓아 올려 삼차원 배열로 치환하여 사용되며 이 삼차원 배열을 볼륨 데이터(volume data)라고 부르고, 볼륨을 이루는 삼차원 배열의 각각의 요소를 복셀(voxel)이라 부른다. 이렇게 생성된 볼륨을 가시화하는 방법은 여러 가지가 있는데 그중 가장 널리 사용되고 있는 기법은 광선 투사를 사용한 기법이다.



[그림 1] 광선 투사를 이용한 볼륨 가시화

카메라의 관찰 방향을 기준으로 수직인 가상의 스크린을 생성한 후 스크린을 구성하는 각각의 픽셀에서 관찰 방향으로 광선을 투사한다. 각각의 광선은 일정한 간격으로 투사를 진행하며 볼륨 데이터와 교차된 복셀의 값을 가져온 후 그 값을 전이함수에 대입하여 RGBA 값을 얻어온다. 전이함수란 밀도 값을 입력하면 RGBA값을 출력하는 함수로, 사용자가 직접 함수를 디자인하기 위해 그래프의 꼭짓점들을 입력하고 그 꼭짓점을 직선으로 이어서 전이함수를 생성하는 방식이 일반적이다. 이렇게 광선의 일정 간격마다 얻어진 RGBA 값을 알파 블렌딩을 통해 광선이 시작한 위치에 있는 픽셀에 누적시켜 완성된 스크린 영상을 화면에 출력하는 것이 바로 광선 투사를 사용한 볼륨 가시화 기법이다. 이러한 방식은 연산량이 많이 요구하므로 CPU의 성능으로는 실시간 가시화를 구현하는 데에는 많은 어려움이 따른다. 따라서 GPU를 활용한 병렬 프로세싱 기법으로 이 문제를 해결한다. 대표적인 병렬 프로세싱 기법은 OpenGL 등에서 지원하는 셰이더 프로그램을 활용하는 방식으로 본 논문에서 사용한 VTK.js또한 OpenGL의 웹 버전인 WebGL을 지원하는 셰이더 프로그램을 이용한다.

2.2 WebGL기반 가시화 기법

웹 기반의 다양한 볼륨 가시화 방법이 제안되었다. 대부분은 WebGL을 직접적으로 사용한 방법이다. 예를 들어 [Mobeen, M, M., 2012]의 경우 일반적인 다중 패스가 아닌 단일 패스 방식의 가시화 알고리즘을 제안하여 효율적인 가시화가 가능하도록 하였다. [Zhang, Q., 2019]의 경우 직접 Node.js 및 Socket.IO 라이브러리를 사용하여 WebGL기반의 가시화 시스템을 개발하였다. 그 이유는 웹에서는 서버와의 데이터 송신 지연시간이 중요하게 보았기 때문이다. [Mwalongo, F., 2018]의 경우 전송할 볼륨 데이터를 블록화시켜 저해상도 블록과 고해상도 블록을 따로 저장한 후 고해상도 블록이 전송되는 지연시간 동안 저해상도 블록을 전송시켜 낮은 해상도의 데이터를 시각화하고 고해상도 블록이 도착하면 즉시 렌더링 화면을 업데이트하는 시

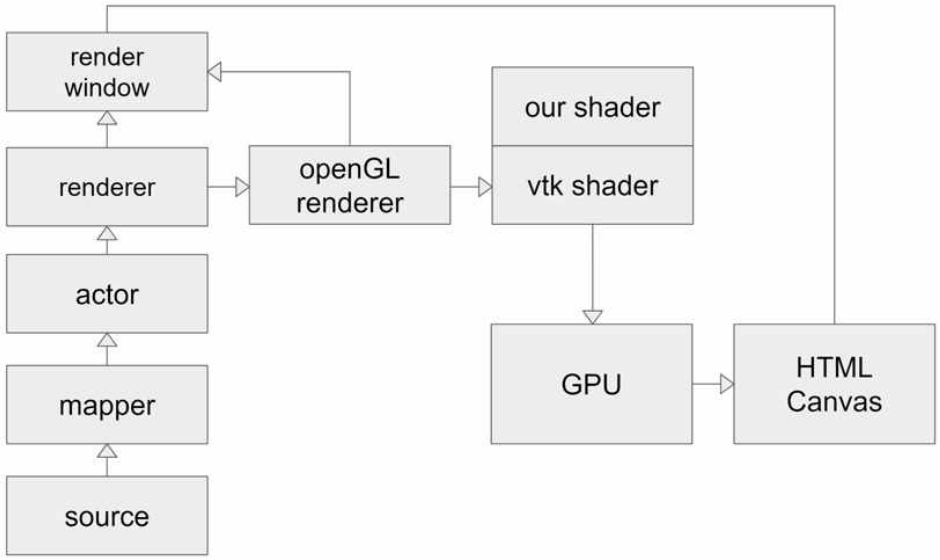
스템을 개발하였다. 또한, 인터넷 환경에서 많은 연구자들이 볼륨 가시화를 사용하는 시도는 과거에도 많이 있었다[Noguera, J, M., 2015]. 예를 들면 모바일 장치는 연산 능력이 제한적이므로, 모바일 장치는 대화명 명령을 받아 서버로 전송하고 서버는 명령을 처리해 가시화 결과물을 전송하는 방식이 있었고 [Lamberti, F., 2005], 네트워크로 가시화 영상을 효율적으로 전송하는 압축알고리즘 및 볼륨 데이터의 재구성을 위한 연구도 진행되었다[Zhou, H., 2006]. 본 연구는 WebGL을 직접적으로 사용하지 않고, VTK.js를 사용하는 방식으로 제작되었다. VTK.js는 대부분의 프레임워크가 이미 설계되어 있다. 이는 기존의 시각화 오픈소스인 VTK[Schroeder, W, J.,2000]를 자바스크립트의 언어로 이식한 결과이다. 그러나 VTK의 모든 기능이 자바스크립트로 이식되지 못하였기에 기존 VTK에서 제공하는 기능을 전부 사용하지 못한다. 또한, 자바스크립트의 언어적 특성으로 인해 기존 C++로 구현된 VTK의 수행성능을 그대로 얻지 못하였다. 하지만 VTK.js는 WebGL의 기술을 사용하여 OpenGL의 기능들을 구현하여 GPU 환경에서의 구동은 기존 VTK과 대등한 성능을 구현할 수 있다.

III. VTK.js를 이용한 웹 기반 볼륨 가시화 시스템

3.1 VTK.js의 구조

3.1.1 개요

VTK.js는 VTK의 구조와 동일하게 제작되었다. 대부분의 VTK 함수와 모듈이 VTK.js로 이식되었다. 다만 VTK가 C++과 OpenGL을 기반으로 동작하는 데 반해, VTK.js는 웹에서 동작하므로 자바스크립트와 WebGL을 기반으로 한다. VTK.js의 구조는 [그림 2]와 같이 actor, mapper, source, renderer, openGLRenderer, renderWindow 등으로 구성되어 있다.



[그림 2] VTK.js의 클래스 구성도

source는 가시화할 대상의 정보 데이터를 가지는 클래스이다. 표면 데이터라면 정점좌표, 표면 인덱스 등을, 볼륨 데이터라면 복셀 밀도 값 등을 갖는다. mapper는 source를 어떠한 방식으로 표현할지 정한다. 셰이더와 source 간의

인터페이스 역할을 수행하여 다양한 표현 방식을 설정할 수 있다. actor는 가시화 화면에 등장하는 개별 물체를 뜻한다. 경계상자(bound box)의 범위, 객체의 위치, 회전 값 등 물체를 표현하기 위한 정보를 담고 있다. renderer는 mapper와 연결된 actor를 가시화하는 3D 공간을 표현하는 개체이다. 카메라, 개체들이 renderer 안에 존재한다. openGLRenderer는 actor, mapper, renderer의 설정값들을 모아 셰이더에 전달하는 역할을 수행한다. 마지막으로 renderWindow는 renderer가 이미지를 그리는 그래픽 사용자 인터페이스의 창이다. 또한 여러 개의 renderer를 포함할 수 있어 한 화면에 다양한 가시화 결과를 표현할 수 있다.

3.1.2 Source

VTK.js는 의료데이터 가시화를 지원하기 위해서 볼륨 데이터를 ImageSource라는 형식으로 지원한다. ImageSource는 이차원뿐만 아니라 삼차원 영상도 지원한다. 이차원 단면 의료영상인 Dicom파일들을 적층하여 삼차원 형식으로 만들어 볼륨 데이터로 사용한다. 이 논문에서는 여러 개의 Dicom파일에서 CT 영상 데이터를 모두 추출 후 모든 이미지를 선형으로 연결하고, Dicom파일의 헤더에서 3차원 배열의 정보(해상도 및 픽셀 간격 등)를 추출 후 VTK.js가 사용할 수 있는 3D 배열 ImageSource로 만들어 사용하였다.

3.1.3 Mapper

Mapper는 셰이더 기능과의 인터페이스로써 역할을 수행한다. 본 논문에서는 볼륨 가시화를 수행하므로 볼륨 가시화를 표현할 수 있는 여러 가지 방식을 모드로 제공한다. Composite 모드는 가장 일반적인 모드로써 광선 투사 과정에서 알파 블렌딩의 누적과정의 결과물을 표현한다. Maximum intensity projection(이하 MIP), Minimum intensity projection(이하 MinIP) 모드는 광선 투사 중에 얻은 복셀 값 중 가장 작거나 큰 값만을 결과물로 표현한 방식

이다. Average intensity projection(이하 AvgIP) 모드는 마찬가지로 광선 투사 중 얻어진 복셀 값들의 평균값을 표현한다.

3.1.4 Actor

VTK.js는 렌더링할 대상 객체를 actor라고 칭한다. actor는 위치, 회전, 크기 등 물체를 표현하기 위한 기본적인 값들을 갖는다. actor는 property라는 물체의 속성 정보를 저장하는 객체를 포함한다. 이 논문에서 사용된 actor는 volume actor로 볼륨 가시화에 사용되는 속성 정보를 가지게 되는데, 대표적으로 전이함수가 있다. 전이함수는 볼륨 데이터에서 얻은 X-선 흡수도 값을 색상 값으로 변환하는 함수로 사용자가 임의로 지정하여 생성하게 된다.

3.1.5 Renderer

VTK.js에서 renderer는 일반적인 컴퓨터 그래픽 렌더링 파이프라인에서 말하는 월드 공간의 가시화를 담당한다. 이 월드 공간 안에 actor를 추가시켜 다양한 렌더링 결과물을 만들어 낸다. 본 논문에서는 volume actor를 renderer에 추가하여 웹 애플리케이션을 구축하였다. 또한, renderer는 카메라와 조명 객체를 포함하고 있어 카메라와 조명을 이용한 다양한 연출이 가능하다.

3.1.6 OpenGLRenderer

VTK.js에서 openGLRenderer는 source, mapper, actor, renderer의 정보를 통합하여 셰이더에 전달하는 역할을 수행한다. 본 논문에서는 mapper의 모드의 정보를 수정하여 Mapper에서 제공하는 기본 모드에 전-적분 전이함수모드, 기울기 표면 투명화 모드, 환경광 폐색 모드를 추가하였다. 이 추가된 모드들을 셰이더로 전달해야 하므로 OpenGLRenderer에서 또한 코드 수정을

수행해야 한다.

3.1.7 RenderWindow

VTK.js에서 RenderWindow는 생성된 Renderer, OpenGLRenderer등을 통합하여 사용자에게 다양한 기능을 제공하는 인터페이스 역할을 수행한다. VTK.js는 미리 설정된 사용자 입력 인터페이스를 여러 가지 옵션으로 제공하기 때문에 사용자는 이러한 옵션을 선택하기만 하면 쉽게 VTK.js를 조작할 수 있다. 예를 들어 ImageStyle 옵션을 선택하면 이미지의 Windowing 값을 마우스 드래그로 조정할 수 있고, 마우스 휠을 통하여 이미지의 확대 축소가 가능하다. 또한, RenderWindow는 여러 개의 Renderer를 포함할 수 있어 한 화면에 여러 개의 공간을 표시하는 것이 가능하다.

3.2 VTK.js의 셰이더 프로그램 분석 및 확장

3.2.1 셰이더 프로그램 개요

셰이더란 GPU에게 명령을 내리는 프로그래밍 언어 및 기능이다. 이 언어를 사용하여 그래픽 렌더링 처리를 병렬적으로 수행할 수 있다. 이전 CPU로 처리하던 조명연산, 좌표변환 연산 등을 GPU로 병렬 처리하여 연산을 가속화 하며, 이를 위해 다양한 연산자와 함수들을 제공한다. 셰이더는 크게 2가지로 구분하며, 정점 셰이더(Vertex Shader 이하 VS)와 프레그먼트 셰이더(Fragment Shader 이하 FS)로 구분할 수 있다. VS는 물체의 정점 정보를 병렬로 처리하며 그 결과물을 삼각형(primitive)으로 생성하여 FS에게 전달한다. 일반적으로 정점의 이동, 회전 같은 좌표변환 연산과 물리적인 파티클 구현에 사용된다. FS는 전달된 삼각형들의 면을 색상 픽셀값으로 채우는 역할을 담당한다. FS에서 가장 많이 담당하는 것은 주로 조명 및 텍스쳐 연산이며 그 외에 다양한 렌더링 효과를 구현하는 데 사용된다. VTK.js는 볼륨 가시화를

수행할 때 vtkVolumeVS.glsl, vtkVolumeFS.glsl 이라는 두 개의 파일을 셰이더로 사용한다. 파일명 말미의 VS, FS를 구분하여 VS 와 FS를 혼동하지 않게 방지하였다. 확장자인 glsl은 이 파일이 셰이더라는 것을 나타낸다. VTK.js에서 두 개의 VS, FS 파일에 데이터를 전달하기 위해서는 Uniform이란 방식을 사용하여 값을 전달하게 된다. Uniform은 WebGL에서 사용되는 전역 변수로써 외부에서 셰이더로 값을 전달할 때 사용되는 기능이다. VTK.js는 WebGL에서 제공하는 Uniform 함수를 이용하여 glsl 파일에 미리 작성해둔 Uniform 변수에 값을 전달하는 코드를 Mapper에서 작성하면, 완성된 glsl 파일이 컴파일될 때 값이 전달된다. Uniform은 변하지 않는 값이기 때문에 컴파일되는 시점에 값이 정해지고 이는 변경되지 않는다. 따라서 Mapper에서 사용된 Uniform 값을 변경되었을 때 변경된 값을 반영하려면 다시 컴파일하여야 한다. Uniform에서 지원하는 변수 타입은 일반적인 프로그래밍에서 사용되는 정수형 변수 타입 (Int, Short 등….)과 부동소수점 타입(Float, Double 등….) 등이 지원되며 GPU에서 작동되는 프로그램 언어이므로 행렬에 관련된 변수형도 지원한다. 대표적으로 Vector 타입(vec3, vec4 등….)을 지원하여 차원을 가지는 변수를 다룰 수 있으며 Matrix 타입 (mat3, mat4 등….)또한 지원하여 다양한 행렬 연산을 편하게 다룰 수 있어 단순한 좌표계산과 색상 계산을 하는 것부터 복잡한 물리적 수식까지 편리하게 코드 작성이 가능하다. 또한, Texture 기능을 지원 하므로 배열 데이터를 쉽게 셰이더로 전송할 수 있으며 전송된 데이터는 자동으로 보간이 이루어지므로 코드를 작성하는데 편리하고 적합한 데이터를 사용할 수 있다. 본 논문에서는 Texture 기능을 이용하여 전-적분 전이함수의 정보를 가지고 있는 배열을 셰이더로 전송하였으며, 환경광 폐색 기법에서는 3차원 마스크 볼륨 데이터 배열을 셰이더로 전송하는 데 사용되었다.

3.2.2 VTK.js에서의 셰이더 기능 확장

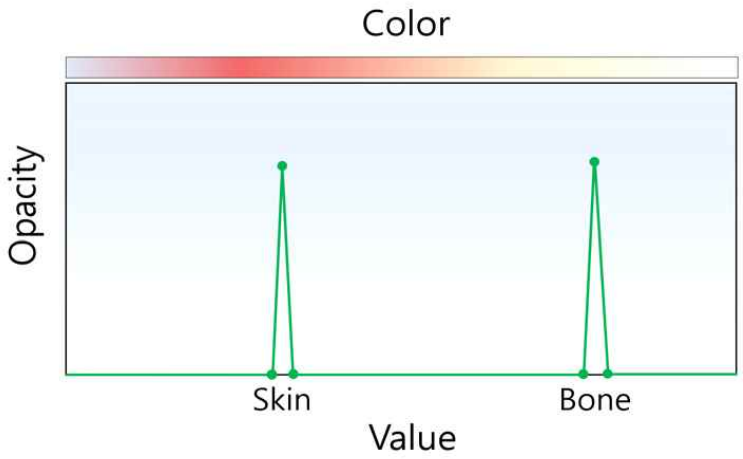
VTK.js에서 셰이더의 인터페이스 역할을 수행하는 Mapper는 볼륨 렌더링에 사용되는 다양한 알고리즘을 모드로 제공하며 기본적으로 제공하는 4가

지의 고정된 알고리즘을 선택할 수 있게 되어있다. 4가지의 모드는 Composite 모드, MIP 모드, MinIP 모드, AvgIP 모드가 있다. 기본 모드는 Composite 모드로 2.1에서 설명한 볼륨 가시화 기법을 구현한 모드이다. 다른 모든 모드들은 이 Composite 모드를 기반으로 약간의 변형을 가해 원하는 결과물을 만들어 낸다. MIP, MinIP는 Composite 과정과 비슷한 알고리즘으로 작동되나 광선 투사를 하여 얻은 샘플 간격당 얻은 복셀 값을 누적(alpha blending)하지 않고 복셀 값의 최대, 또는 최솟값을 갱신하면서 광선 투사가 끝나면 얻은 결과를 출력한다. 이때, VTK.js의 출력은 색상 표현이 가능하도록 전이함수에 대입하여 최종 RGB 값이 출력된다. AvgIP의 경우도 마찬가지로 샘플 간격당 얻은 복셀 값을 평균하여 최종 RGB 색상을 출력한다. 이 알고리즘에 추가로 3개의 방법을 추가 구현하였으며, 기존 시스템에 통합되도록, 번호를 부여하였다. 이 번호는 js의 Enum 문법으로 정의되어 있으므로, Mapper 클래스에서 사용하는 Constants 클래스(Enum)에 전-적분 전이함수 모드와 기울기 투명화 기능 모드, 환경광 폐색 모드를 의미하는 새로운 알고리즘 코드 번호를 추가하였다. 이 알고리즘 선택 코드는 가시화 과정에서 GPU 셰이더로 전달되어 본 논문에서 제작한 새로운 알고리즘을 수행하는 기준이 된다.

IV. 전-적분 전이함수

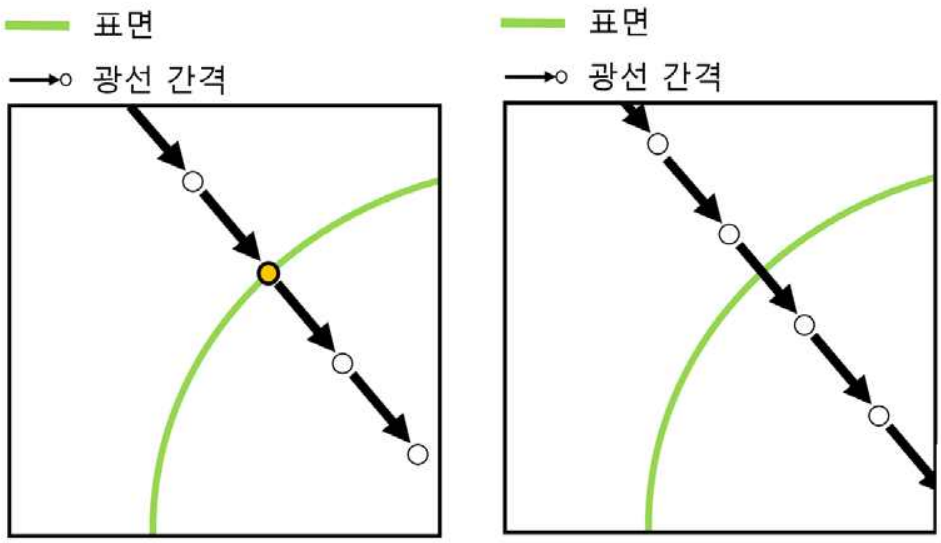
4.1 개요

의료영상을 볼륨 가시화 기법으로 표현할 경우 피부, 근육, 뼈 같은 인체 조직의 표면만을 따로 가시화를 해야 하는 경우가 있다. 이때, 표면에 해당하는 복셀 값을 포함하는 범위의 전이함수를 사용한다. 예를 들어 사용자가 피부와 뼈만 가시화하기를 원한다면 피부에 해당하는 복셀 값의 범위와 뼈에 해당하는 복셀 값의 범위를 찾아 전이함수를 해당 범위에서 반투명하도록 설정한다. 피부와 뼈를 제외한 나머지 범위는 모두 투명하게 설정하여 영상에 반영되지 않도록 한다. 이때 표현하려는 복셀 값의 범위가 매우 좁은 경우 [그림 3]과 같이 전이함수의 그래프는 날카로운 모양을 하게 된다. 이때, 특별한 고려 없이 이 전이함수를 가지고 가시화를 진행하면 가시화 영상에 노이즈가 발생한다. 노이즈가 발생하는 이유를 설명하기 위해서 다음과 같이 예시를 보인다.



[그림 3] 피부와 뼈의 표면만을 가시화하는 날카로운 모양의 전이함수

같은 성질의 복셀 들은 모두 같은 복셀 값을 가지지 않고 약간의 오차가 존재한다. 예를 들면 피부의 복셀 값이 100이라 가정한다면 주변 피부의 복셀 값은 오차범위 ±20 정도의 값 차이가 날 수 있다. 또한, 피부와 비슷한 밀도를 가진 근육의 경우 복셀 값이 150이고 오차범위가 ±40이라고 가정하면 110에서 120까지의 복셀 값은 피부로 파악될 수도 있고 근육으로 파악될 수도 있다. 사용자가 피부의 모든 범위를 포함하려 하면 해당 범위에 존재하는 근육도 포함하여 가시화가 진행되기 때문에 노이즈가 발생한다. 이를 피하고자 너무 좁은 영역의 전이함수로 가시화를 수행하면 피부에 해당하는 일부 복셀 들을 놓쳐서 또 다른 형태의 노이즈가 발생할 가능성이 생긴다. 또한, 표면을 얇은 범위의 전이함수를 사용하게 되면 다른 문제가 발생한다. 광선 투사 기반의 볼륨 가시화 기법은 광선이 투사되는 방향으로 한 간격씩 진행하며 복셀과 교차점을 검사한다. 이때 간격의 길이가 표면의 두께에 비해 너무 크다면 [그림 4 (b)]와 같이 표면을 교차하지 못하고 지나가게 된다. 표면의 두께는 전이함수의 범위에 따라 결정되므로 전이함수의 표현 범위가 얇다면 두께도 얇아지게 되어 교차 검사에 실패할 확률이 높아진다. 교차하지 못한 위치는 표면이 아니므로 투명한 공간으로 인식되고 해당하는 픽셀은 검은색 노이즈로 나타나는 결과로 이어진다. 이때 단순한 해결책으로 진행 간격의 길이를 줄여 표면에 교차하는 확률을 높이는 방법이 있다. 하지만 이 방법은 확률을 높이는 방식이기 때문에 노이즈를 모두 제거한다는 보장이 없으며 광선마다 진행 및 교차검증의 횟수가 반비례하여 증가하게 되므로 볼륨 가시화에 소모되는 시간이 크게 증가한다.



(a) (b)

[그림 4] (a) 표면과 광선이 교차한 경우
(b) 표면과 광선이 교차하지 못한 경우

이러한 문제를 해결하기 위해 본 논문은 전-적분 전이함수 기법을 도입하였다. 전-적분 전이함수 기법은 적분 평균을 이용하여 얇은 범위를 가지는 전이함수를 변형하는 기법으로, 표면의 두께가 얇아서 생기게 되는 광선에 교차하지 못한 복셀 들을 가시화 결과에 포함하여 노이즈를 제거한다.

4.2 전이함수의 수치 적분

적분 평균에 사용되는 전이함수는 배열형식으로 변환되어 저장된 함수이므로, 함수식을 직접 적분하지 않고 식 (1) 과같이 배열 값에 대한 수치 적분을 수행할 수 있다.

$$p(i) = \sum_{x=0}^{i} t(x) \tag{1}$$

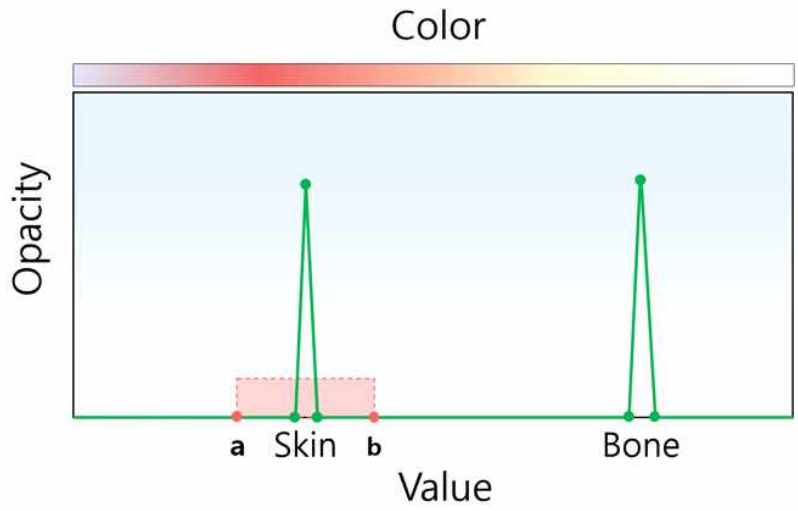
x의 범위는 배열의 색인의 범위, 즉 0 이상의 정수이며 함수 $t(x)$ 는 기존에 사용하던 전이함수를 나타내고, 함수 $p(i)$ 는 전-적분 기법에 사용될 누적 전이함수(이하 전-적분 전이함수)를 뜻한다. 전-적분 전이함수의 i 번째 요소의 값은 기존 전이함수 $t(x)$ 를 0부터 i 번째까지의 요소 값을 누적시킨 값과 같으며 이러한 전-적분 전이함수를 볼륨 가시화 이전에 전처리 작업으로 미리 생성하여 기존 전이함수를 대체하도록 한다. 이때 생성된 전-적분 전이함수는 식 (2) 와 같은 규칙이 성립한다.

$$\int_a^b t(x)dx = p(b) - p(a) \tag{2}$$

전이함수 $p(i)$ 는 $t(x)$ 의 부분합이므로 구간 a 부터 b 까지의 적분의 결과는 전-적분 전이함수 $p(i)$ 에 b 와 a 를 대입하여 뺀 차이 값이다. 이 식에 적분의 평균을 구하는 식을 적용하면 식 (3) 과 같다.

$$\frac{1}{b-a}\int_a^b t(x)dx = \frac{p(b)-p(a)}{b-a} \tag{3}$$

최종적으로 기존 전이함수 대신 얻어진 적분의 평균 기울기 값을 사용하여 볼륨 가시화를 진행하면 기존의 얇은 범위를 가진 전이함수가 상황에 따라 적절한 함수로 변형되어 이전에는 가시화 결과에 포함하지 못했던 복셀들도 가시화 결과에 포함되게 된다.



[그림 5] 전-적분 전이함수로 인해 a, b 구간이 변형된 전이함수

[그림 5]와 같이 a부터 b까지 적분 평균을 적용하면 기존 얇은 범위를 가진 전이함수가 넓은 범위를 가진 직사각형 모양의 전이함수로 변형된다. 볼륨 가시화에서 a의 의미는 광선의 현재 위치에서 얻은 밀도 값이고, b의 의미는 광선의 방향으로 한 간격만큼 이동한 위치의 밀도 값이다. 따라서 a가 기존 전이함수 범위에 포함되지 않더라도 적분으로 변형된 전-적분 전이함수를 사용하면 a와 b구간의 적분 평균값만큼의 투명도를 얻게 되어 얇은 표면을 놓치지 않고 가시화를 수행할 수 있게 된다. 한편 적분 평균을 이용하는 전-적분 기법에 넓은 범위를 가지는 일반적인 전이함수를 대입할 경우 기존 방법과 차이가 없게 된다. 전이함수 적분의 평균 기울기와 전이함수의 한 점에서의 높이는 차이가 작기 때문이다. 전-적분 기법을 기존 가시화 방법 대신에 대체해서 사용하였을 때, 얇은 범위의 전이함수에 대해서는 노이즈를 제거하고 넓은 범위의 전이함수에 대해서는 차이가 없는 우수한 영상을 얻을 수 있다.

V. 기울기 추정을 통한 표면 투명화

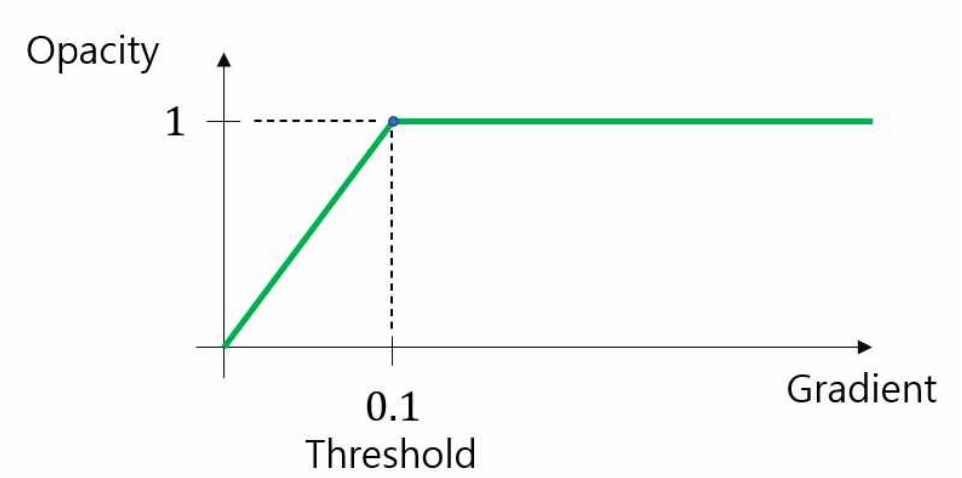
5.1 개요

기울기 추정을 통한 표면 투명화 기법은 전-적분 전이함수 기법과 마찬가지로 인체조직의 표면을 관찰할 때 사용하는 기법이다. 전-적분 전이함수 기법과는 달리 의료영상 데이터의 인체조직 간의 분리를 통한 표면 가시화가 목적이다. 전-적분 전이함수 기법은, 사용자가 표현하고자 하는 표면의 전이함수를 직접 제작하고 이를 누적해 전-적분 전이함수를 미리 생성해야 하지만 기울기 추정을 통한 표면 투명화는 그러한 과정이 불필요하다. 일반적인 전이함수를 그대로 사용할 수 있으므로 간편한 방식이다.

5.2 기울기 추정을 통한 표면 투명화

인체조직 간의 분리를 하기 위해서, 본 논문에서는 주변 복셀 값의 차이를 이용한다. 이 차이를 기하학적으로 해석하여 기울기 값으로 사용하며 이 기울기 값이 큰 차이를 나타낸다면 서로 다른 인체조직으로 인지하게 된다. 그 이유는 같은 인체조직끼리는 비슷한 밀도 값을 가지고 있기 때문이다. 이러한 방식은 영상처리의 기울기를 이용한 엣지 검출방식과 동일한 방식이다. 볼륨 가시화의 광선 투사 중 복셀과의 교차 시 교차된 복셀 위치에서 주변 복셀과 기울기를 계산한다. 기울기 벡터의 크기가 작다면 현재 위치는 주변 복셀과 동일한 인체조직의 내부로 판단하여 더 투명하게 처리한다. 반대로 기울기 크기가 크다면 해당 복셀은 서로 다른 조직이 서로 인접해있는 경계 표면으로 인식하여 해당 복셀을 그대로 가시화를 진행한다. 이러한 방식으로 볼륨 가시화가 완료되면 인체조직 내부의 서로 동일한 조직은 투명도가 올라가게 되어 볼륨 가시화 결과물에 반영되지 못하는 반면 표면에 해당하는 인체

조직의 복셀 들은 투명도가 낮게 적용되어 볼륨 가시화에서 결과물에 뚜렷이 나타나는 결과를 얻을 수 있다. [그림 6]은 기울기 값에 따른 투명도의 변화를 보인다.



[그림 6] 기울기 추정값의 크기에 따른 투명도 그래프

[그림 6]의 그래프는 기울기 값이 사용자가 정한 임곗값(threshold)보다 작으면 불투명도가 감소하는 것을 보인다. 기울기가 높은지 낮은지를 판별하는 임곗값은 값이 정해져 있지 않고 의료영상 데이터가 가지는 복셀 값의 표현 범위에 따라 달라지므로 사용자 경험에 의해서 조절된다. 위 그래프를 수식으로 표현하면 (식 4)와 같다.

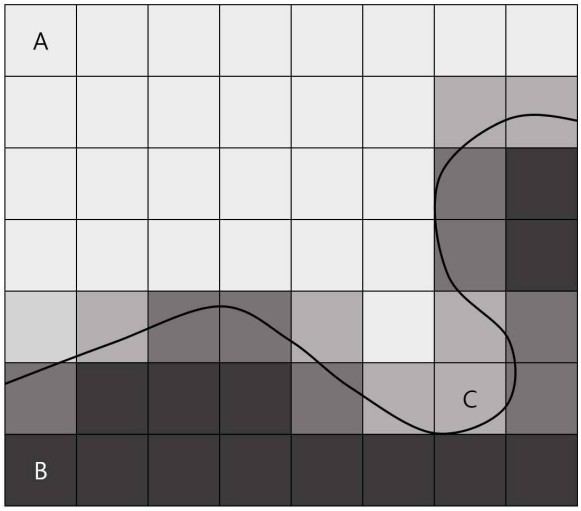
$$O = \begin{cases} O & G \geq T \\ O \times \dfrac{G}{T} & G < T \end{cases} \qquad (4)$$

각각 임곗값 T, 기울기 추정값 G, 투명도 O를 뜻하며 기울기와 임곗값의 크기 비교를 통해 분기된다.

VI. 환경광 폐색 기법

6.1 개요

환경광 폐색 기법(Ambient Occlusion)은 고품질의 음영을 생성하여 볼륨 가시화 및 일반 표면 가시화 품질을 향상하는 기법으로 많이 사용된다. 이 기법은 3차원 그래픽 환경에서 움푹 파여 주변 빛을 덜 받는 부분에 대해 환경광의 영향을 제한하여 더욱 어둡게 표현하는 방식이다. 환경광 폐색 기법을 사용하기 위해서는 각 지점에 대해 움푹 파인 정도에 대한 측정을 수행하고 그 결과를 저장해야 한다. 볼륨 가시화 환경에서는 모양이 정해진 표면 가시화와 달리 전이함수에 의해서 복셀의 투명도가 달라지므로 고정된 모양을 가지고 있지 않다. 따라서 볼륨 가시화 환경에서 환경광 폐색 기법을 사용하려면 전이함수가 변경될 때마다 움푹 파인 정도의 탐색을 다시 해야 한다. 볼륨 가시화에서의 움푹 파인 정도의 측정 방법은 다음과 같다. 각 복셀 지점에 대해 일정 커널 크기의 주변 영역을 설정하고 해당 영역의 불투명도 평균값을 계산하여 움푹 파인 정도를 측정한다.



[그림 7] 커널 영역 평균값에 따른 색 분포 그래프

[그림 7]은 3차원 볼륨 데이터에 전이함수를 적용하고 커널의 크기 K의 세제곱 크기만큼 구역을 나눈 후 불투명도에 따른 색 농도를 차등 적용시켜 2차원 이미지로 표현한 그림이다. [그림 7]의 곡선을 중심으로 위쪽 영역은 투명한 공간, 아래쪽 영역은 불투명한 공간을 표현한다. A의 영역은 불투명도가 낮은 공간이므로 일반적으로 공기와 같은 영역을 투명하게 표현할 경우가 이에 해당한다. B의 영역은 불투명도가 높은 공간이므로 전이함수가 불투명하게 표현하려는 물체가 이에 해당한다. A와 B의 영역은 서로 같은 물질끼리 모여있어 서로 복셀 값이 비슷하므로 영역 내의 복셀 값들의 편차가 크지 않고 서로 극단적인 평균값을 가지게 된다. 그러나 C의 영역의 경우 곡선을 중심으로 불투명한 부분과 불투명하지 않은 부분이 섞여 있으므로 A나 B의 영역에 비해 편차가 큰 편이며 평균값 또한 A나 B와는 차이가 나게 된다. 이러한 특성을 이용해 C 영역에 해당하는 부분을 움푹 파여진 곳으로 정의하여 C 영역에 해당하는 부분을 어둡게 표현하여 고품질의 음영을 생성한다.

6.2 증분 알고리즘을 통한 마스크 볼륨 생성

환경광 폐색 기법을 사용하기 위해서는 각 지점에 대해 움푹 파인 정도에

대한 측정을 수행하고 그 결과를 저장해야 한다. 본 연구에서는 이를 저장하기 위한 볼륨 데이터를 추가하여 마스크 볼륨이라고 정의한다. 마스크 볼륨은 각 복셀 지점에 대해 일정 커널 크기의 주변 영역을 설정하고 해당 영역의 불투명도 평균값을 계산하여 구하며 [그림 7]의 색 분포 그래프를 수치화 하여 3차원 배열에 저장하는 원리와 같다. 본 연구에서는 최초 실행할 때와 전이함수의 변경이 일어날 때만 마스크 볼륨 데이터를 변경하고 이후에는 생성된 데이터를 재사용한다. 또한 마스크 볼륨 데이터 생성 속도를 향상하기 위해 증분 알고리즘을 적용하였다. 마스크 볼륨 데이터를 생성하려면 볼륨 크기 n^3에 커널 크기 k^3를 곱한 만큼의 연산량 $O(n^3k^3)$이 소요된다. 증분 알고리즘을 이용하면 이를 $O(n^3)$으로 줄일 수 있다[남진현., (2015)].

Algorithm 1 : 증분 알고리즘이 적용되지 않은 마스크 볼륨 생성 알고리즘

input : 3차원 볼륨 데이터 V

```
1   For 볼륨 배열 s축에 대하여 Do
2       For 볼륨 배열 t축에 대하여 Do
3           For 볼륨 배열 u축에 대하여 Do
4               For x축에 대한 커널 범위 k에 대하여 Do
5                   T ← T + V[k][t][u]
6               End For
7               For y축에 대한 커널 범위 k에 대하여 Do
8                   T ← T + V[s][k][u]
9               End For
10              For z축에 대한 커널 범위 k에 대하여 Do
11                  T ← T + V[s][t][k]
12              End For
13              Mask[s][t][u] ← T
14          End For
15      End For
16  End For

17  Return Mask
```

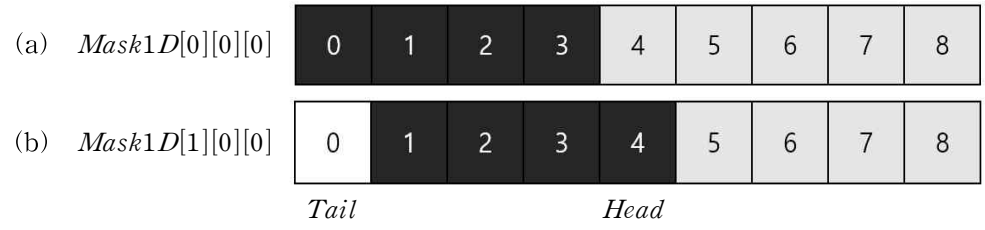
[표 1] 증분 알고리즘이 적용되지 않은 마스크 볼륨 생성 알고리즘

증분 알고리즘을 3차원 볼륨 데이터에 적용하려면 순차적인 과정이 필요하다. 먼저 1차원에 해당하는 x축에 증분 알고리즘을 적용하여 1차원 마스크

볼륨 데이터 $Mask1D$ 를 생성해야 한다. 그 다음 생성된 $Mask1D$ 를 기반으로 2차원, 3차원으로 확장하여 최종적인 마스크 볼륨 데이터를 완성한다.

$$Mask1D[s+1][t][y] := Mask1D[s][t][u] + Head1D - Tail1D$$
$$Head1D := V[s+1+(\frac{k}{2})][t][u] \quad (1)$$
$$Tail1D := V[s-(\frac{k}{2})][t][u]$$

수식의 s, t, u는 3차원 배열의 좌표를 의미하며 V는 볼륨 3차원 배열을 나타낸다. 수식 1. 로 생성된 $Mask1D[s][t][u]$는 1차원 x축에한하여 $V[s][t][u]$ 중심으로 커널 k의 크기만큼 주변의 값의 합산 값이다.



[그림 8] 커널 크기가 4인 $Mask1D$의 일부

[그림 8]은 (수식1)의 이해를 돕기 위한 그림이며 조건은 다음과 같다. (a)와 (b)의 0부터 8까지는 볼륨 3차원 배열의 x축 배열 일부를 표현하였으며 검은색 배경은 커널 크기를 4로 설정하였을 때 평균값을 구하기 위해 합산할 복셀들을 표시한 것이다. 따라서 (a)는 $Mask1D[0][0][0]$을 나타내며 0번 부터 3번 까지의 복셀들의 합이고, (b)는 $Mask1D[1][0][0]$을 나타내며 1번 부터 4번 까지의 복셀들의 합으로 표현된다. 이때 (a)와 (b)을 이루고 있는 복셀들은 1번 부터 3번 까지의 복셀들은 공유하며 0번과 4번에 해당하는 부분만을 차이점으로 두고 있다. 이러한 성질을 이용해서 $Mask1D$의 현재단계(b)를 구하려면, 이전단계 (a)의 앞부분에 해당하는 0번($Tail$)을 제거하고 4번($Head$)을 더하면 된다.

$$Mask2D[s][t+1][y] := Mask2D[s][t][u] + Head2D - Tail2D$$
$$Head2D := Mask1D[s][t+1+(\frac{k}{2})][u] \quad (1)$$
$$Tail2D := Mask1D[s][t-(\frac{k}{2})][u]$$

수식 1.에서 생성된 $Mask1D$는 3차원 볼륨 배열에 대하여 x축의 커널 크기만큼의 누적값을 가진 3차원 배열 데이터가 된다. 그런 다음 $Mask1D$를 이용하여 수식 2.를 적용시키면 x축 방향으로 누적된 값이 y축 방향으로 다시 누적되면서 한 복셀에 대하여 x축으로 k만큼, y축으로 k만큼의 크기를 가진 평면 커널 영역의 합 $Mask2D$를 구할 수 있게 된다.

$$Mask3D[s][t][u+1] := Mask3D[s][t][u] + Head3D - Tail3D$$
$$Head3D := Mask2D[s][t][u+1+(\frac{k}{2})] \quad (1)$$
$$Tail3D := Mask2D[s][t][u-(\frac{k}{2})]$$

또다시 수식2에서 완성된 $Mask2D$를 수식 3.과 같은 방식으로 3차원 z축에 적용하면, 한 복셀에 대하여 x축으로 k만큼, y축으로 k만큼, z축으로 k만큼의 크기를 가진 3차원 커널 영역의 합 $Mask3D$를 얻을수 있고, 모든 Mask 볼륨 데이터의 요소에 대해서 k^3 만큼 나눠면 평균값으로 이루어진 마스크 볼륨 데이터를 얻을 수 있다.

6.3 Web Assembly를 이용한 마스크 볼륨 생성 가속화

6.3.1 Web Assembly 개요

자바스크립트는 웹 환경에서 작동하는 언어로써 인터프리터 방식으로 동작된다. 코드를 미리 컴파일 하여 기계어로 번역하여 실행하는 컴파일 언어와 대비되는 특성이다. 인터프리터 언어의 장점은 컴파일을 미리 하지 않고 코드를 실시간으로 기계어 코드로 변경하여 실행하므로 미리 사전에 컴파일을 할 필요가 없어 컴파일 시간을 단축하여 코드를 실행 시킬수 있다는 장점이 있다. 그러나 이러한 장점은 단점으로도 작용한다. 실시간으로 코드를 기계어로 번역하기 때문에 단순히 기계어를 실행하는 컴파일 언어보다 실행속도가 떨어지는 현상이 나타난다. 기계어로 번역하는 과정중에 일어나는 최적화 작업 또한 인터프리터 언어에서는 코드의 번역과 동시에 일어나므로 컴파일 언어의 최적화와 비교하면 성능이 떨어지는 결과를 보여준다. 이러한 장단점으로 인해 단순하고 작은 일을 수행하는 웹 환경에 적합하여 많이 사용되어 왔다. 그러나 웹 환경의 발달로 인해 웹 환경에서도 복잡하고 높은 속도의 연산을 필요로 하는 경우가 많아졌다. 이러한 문제를 해결하기 위해 나온 것이 Web Assembly기능이다. Web Assembly는 C++으로 작성된 코드를 미리 컴파일하여 저수준의 어셈블리어로 변역하여 모듈화 한다. 이렇게 번역된 어셈블리어 모듈을 자바스크립트와 통신할 수 있는 인터페이스를 이용하여 자바스크립트 코드에서 실행할 수 있게 한다. 자바스크립트에서 실행된 Web Assembly로 제작된 모듈은 인터프리터의 실시간 컴파일 과정을 거치지 않고 바로 기계어로 실행되게 된다. 실행된 모듈은 네이티브 모듈과 거의 동일한 수준으로 작동되게 되므로 자바스크립트의 느린 실행속도를 보완해줄 수 있다. 따라서 사용자는 자바스크립트에서 높은 실행속도를 필요로 하는 부분을 Web Assembly 모듈로 제작하면 높은 효율성을 얻을수 있게 된다.

6.3.2 VTK.js에서의 Web Assembly 적용

ITK.js는 C++로 제작된 다양한 영상처리 모듈인 ITK를 Web Assembly로 변환하여 VTK.js에서 사용 할 수 있게 해주는 라이브러리이다. 이 라이브러리에서 제공하는 C++ 모듈을 Web Assembly로 변환하는 기능을 이용하면 ITK에서 제공하는 모듈 뿐만 아니라 사용자가 직접 작성한 모듈을 Web

Assembly로 쉽게 변환하는 것이 가능하다. 본 논문에서는 마스크 볼륨 생성을 위한 증분 알고리즘을 C++ 모듈로 제작하고 제작된 모듈을 Web Assembly 기능을 이용하여 VTK.js로 제작된 시스템에 통합하고자 한다. 다음은 C++로 작성된 증분 알고리즘이 적용된 마스크 볼륨 생성 알고리즘을 표현한 가상코드 이다.

Algorithm 2 : 증분 알고리즘이 적용된 마스크 볼륨 생성 알고리즘

input : 3차원 볼륨 데이터 V

1 **For** $Mask1D$의 s, t, u축에 대하여 **Do**
2 **For** 커널 크기 k **Do**
3 $Head \leftarrow V[s+1+\frac{k}{2}][t][u]$
4 $Tail \leftarrow V[s-\frac{k}{2}][t][u]$
5 $Mask1D[s][t][u] \leftarrow Mask1D[s-1][t][u] + Head - Tail$
6 **End For**
7 **End For**
8 **For** $Mask2D$의 s, t, u축에 대하여 **Do**
9 **For** 커널 크기 k **Do**
10 $Head \leftarrow Mask1D[s][t+1+\frac{k}{2}][u]$
11 $Tail \leftarrow Mask1D[s][t-\frac{k}{2}][u]$
12 $Mask2D[s][t][u] \leftarrow Mask2D[s][t-1][u] + Head - Tail$
13 **End For**
14 **End For**
15 **For** $Mask3D$의 s, t, u축에 대하여 **Do**
16 **For** 커널 크기 k **Do**
17 $Head \leftarrow Mask2D[s][t][u+1+\frac{k}{2}]$
18 $Tail \leftarrow Mask2D[s][t][u-\frac{k}{2}]$
19 $Mask3D[s][t][u] \leftarrow Mask3D[s][t][u-1] + Head - Tail$
20 **End For**
21 **End For**
22 **Return** $Mask3D$

[표 2]증분 알고리즘이 적용된 마스크 볼륨 생성 알고리즘

VII. 구현 및 실험 결과

7.1 실험 환경

이번 장에서는 구체적인 구현 방법과 실험 결과를 설명한다. 구현은 i5 CPU를 장착한 노트북 컴퓨터에서 수행되었으며, 실행은 데스크탑 및 노트북 등 크롬 웹 브라우저가 동작하는 다양한 환경에서 수행되었다. 본 연구는 웹 기반의 가시화를 수행하므로 출력 영상은 다양한 환경에서 모두 동일하며, 별도의 GPU가 설치되어 있는 일반적인 환경에서 실시간으로 가시화가 수행된다. GPU 기반의 광선 투사법은 다양한 효과를 부여하더라도 일반적으로 실시간으로 수행[강지선., (2019)]된다.

7.2 전-적분 전이함수 구현 및 실험 결과

다음은 전-적분 전이함수를 제작하는 방법을 설명한다. 전-적분 전이함수는 기존의 전이함수를 누적한 배열이다. VTK.js에서 기존의 전이함수는 1024개의 색인값(index)를 가지고 있으며 한 인덱스당 4(RGBA)Byte 크기를 가진 2차원 배열이므로 총 배열의 크기는 1021 × 4 × 2 Byte의 크기를 가진다. 제작하려는 전-적분 전이함수 또한 같은 크기로 메모리를 할당하였다. 1차원 배열인 전이함수를 2차원 배열로 만든 이유는 셰이더의 Texture 기능을 이용하기 위함이다. Texture 기능은 2차원 이미지 배열을 사용하기 때문에 이를 지원하기위해 전이함수를 2차원 배열로 확장하였다. 전이함수 배열은 복셀의 밀도값인 색인값에 대해 RGBA값이 저장되어 있으며 이때 RGBA의 범위는 0부터 255의 범위를 갖는다. 이 배열값을 누적해서 전-적분 전이함수 배열로 저장하려 한다. 여기서 주의해야 할 점은 최대 255의 값을 가진 배열을 1024번 누적될 경우 GPU에서 이 큰 값을 처리하지 못하고 오버플로 되는 현상이

관찰된다. 따라서 전-적분 전이함수를 만들 때는 각 값을 누적하기 전에 값을 255로 나누어 0에서 1 사이의 값으로 축소한 후 누적하여 오버플로 현상을 방지한다. GPU 영역의 셰이더에서 이 값을 사용할 때는 얻은 값에 다시 255를 곱하여 원본값으로 복구시키는 방식을 사용하였다. 전-적분 전이함수의 알고리즘은 전-적분 전이함수를 사용해 광선의 현재 위치와 다음 위치를 전-적분 전이함수에 넣어 결과값을 얻는다. 식 (3)과 같이 이 두 값의 차이를 구하고 광선의 길이(밀도값의 차이)로나누어 전이함수의 적분 평균값을 구하는 것이다. [표 3]은 알고리즘을 담은 가상코드이다.

Algorithm 3 : 전-적분 전이함수

input : 3차원 볼륨 데이터 V, 전-적분 전이함수 PITF, 광선 투사 시작 위치 RSP, 광선의 투사 방향 RD

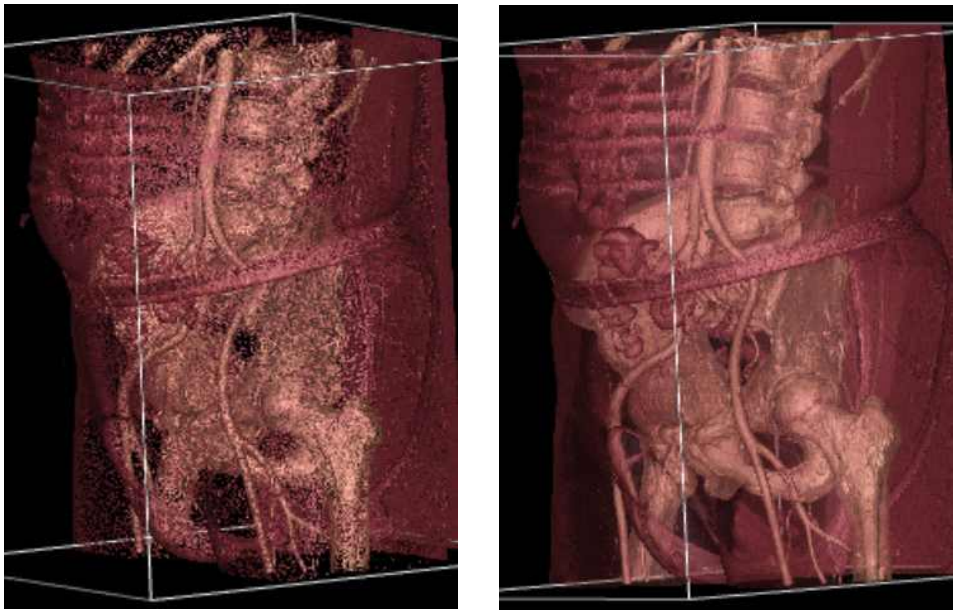
1 **For** 모든 RSP에 대하여 **Do**
2 $P \leftarrow RSP$
3 **For** 광선 투사 범위에 해당하는 모든 P에 대하여 **Do**
4 $A \leftarrow PTF(V[P])$
5 $B \leftarrow PTF(V[P+RD])$
6 $color,\ alpha \leftarrow \frac{B - A}{V[P+RD] - V[P]}$
7 $result \leftarrow Alpha\ Blending(color,\ alpha)$
8 **End For**
9 **End For**

10 **Return** *result*

[표 3] 전-적분 전이함수 알고리즘

[그림 9]는 기존의 VTK.js에서 제공하는 가시화 결과 영상과 본 연구에서 추

가한 전-적분 전이함수의 결과 영상의 비교이다. [그림 9 (a)]는 날카로운 모양의 전이함수를 사용한 결과로 좁은 표현 범위에 포함되지 못한 복셀과, 얇은 표피에 의해 광선과 교차되지 못한 복셀을 가시화하지 못하여 피부에 해당하는 표면과 뼈에 해당하는 표면에 구멍이 뚫려있는 것처럼 노이즈가 결과물에 생기게 된다. [그림 9 (b)]는 전-적분 전이함수를 사용함으로써 [그림 9 (a)]에서 생기는 노이즈를 제거한 모습을 보여준다.



(a) (b)

[그림 9] (a) 날카로운 모양의 전이함수를 사용한 결과물,
(b) 전-적분 전이함수를 적용한 결과물

7.3 기울기 추정을 통한 표면 투명화 구현 및 실험 결과

다음은 표면 투명화 기능의 구현 알고리즘에 관해 설명한다. 표면 투명화 기능에서 중요한 점은 복셀에서 기울기를 추정하는 방식을 고르는 법이다. 일반적으로 복셀은 3차원 데이터이기 때문에 인접한 복셀을 어떻게 정의하느냐의 따라서 기울기 값의 차이가 있을 수 있다. 일반적으로 조명 연산을 수행하기 위해 기울기가 활용되며, 복셀의 상, 하, 좌, 우, 앞, 뒤의 위치한 6개 샘

플값을 구한 후 대칭되는 방향의 값의 차이의 길이로 기울기를 결정한다(central difference). 그러나 이 방법은 한 번 기울기를 계산하기 위해 주변 샘플 6개를 전부 탐색해야 하므로 메모리 접근과 연산의 속도에서 비용이 발생한다. VTK.js에서는 이 비용을 줄이기 위해 상, 좌, 앞 3개의 샘플을 계산하여 중앙의 복셀과의 차이를 이용하는 방식을 이용한다(forward difference). 그 결과 약간의 정확도를 손실하지만 2배의 연산 비용을 절약할 수 있다. 따라서 본 연구는 VTK.js의 형식을 따라서 투명도 조절 시에도 같은 방법을 사용한다. 이렇게 얻게 된 기울기를 사용자가 정한 임의의 임계값과 비교하여 기울기가 임계값 보다 낮다면 투명하게, 높다면 현재 투명도를 그대로 유지하여 볼륨 가시화를 진행한다. [표 4]는 표면 투명화 기능의 알고리즘이다. 기본적으로 이 알고리즘은 볼륨 가시화 방식과 유사하게 동작한다. 광선을 투사하고 한 광선 투사 간격마다 얻을 수 있는 복셀의 값을 전이함수에 적용시켜 색상값과 알파값을 얻고 알파 블랜딩을 진행하는 과정까지는 동일하게 작동한다. 그다음 앞서 설명한 forward difference 방식으로 얻은 기울기값을 해당 복셀의 주변값으로부터 얻어와 사용자가 정한 임의의 임계값 T와 비교한다. 비교후 기울기 값이 임계값 보다 적다면 구해진 알파값의 [그림 6] 과같은 식를 적용하여 알파값의 투명도를 결정한다. 이과정을 모든 광선 투사의 한 간격마다 진행하여 표면 투명화를 완성한다.

Algorithm 4 : 기울기 추정을 통한 표면 투명화

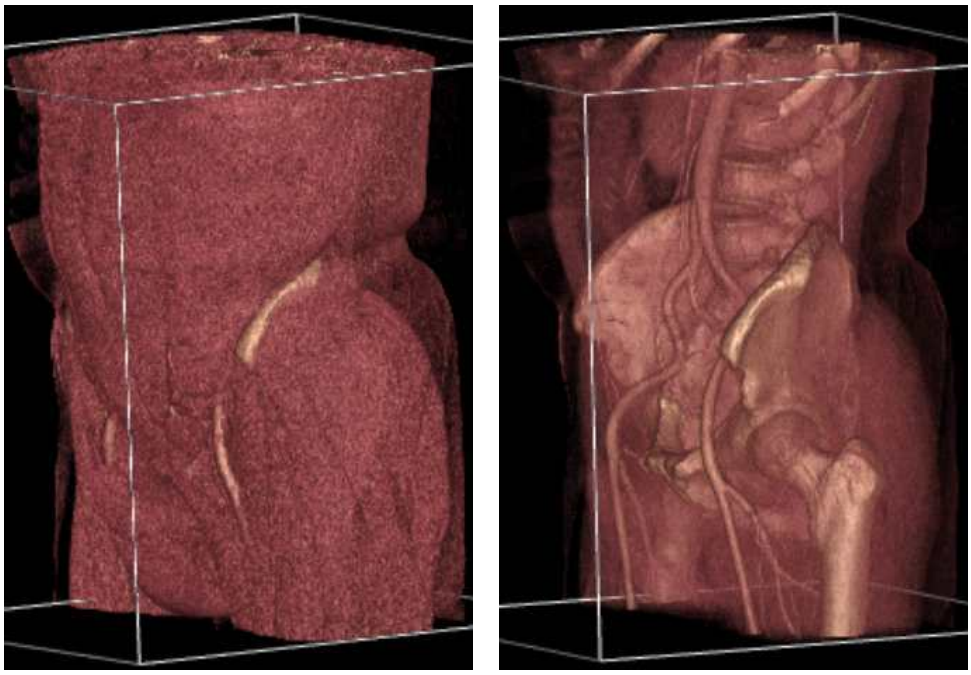
input : 3차원 볼륨 데이터 V, 전이함수 TF, 광선 투사 시작 위치 RSP, 광선의 투사 방향 RD, 임의의 임계값 T

For 모든 RSP에 대하여 **Do**
  $P \leftarrow RSP$
  For 광선 투사 범위에 해당하는 모든 P에 대하여 **Do**
    $color,\ alpha \leftarrow TF(V[P])$
    $color,\ alpha \leftarrow Alpha\ Blending(color,\ alpha)$
    $G \leftarrow Gradient(V[P])$
    If $G < T$ **Then**
      $alpha \leftarrow alpha \times \frac{G}{T}$
    End If
    $result \leftarrow color,\ alpha$
  End For
End For

Return *result*

[표 4] 기울기 추정을 통한 표면 투명화

[그림 10]은 기존의 VTK.js에서 제공하는 가시화 결과 영상과 본 연구에서 추가한 표면 투명화의 결과 영상의 비교이다. [그림 10 (a)]는 일반적인 전이함수를 사용한 볼륨 가시화 결과물이다. 근육과 뼈, 혈관을 포함하는 범위의 전이함수를 사용하였기 때문에 뼈와 혈관은 근육에 가려져서 보이지 않는다. 표면 투명화를 적용한 [그림 10 (b)]는 표면이 아닌 부분은 투명하게 만들어 근육, 뼈, 혈관의 표면을 명확하게 보여준다.



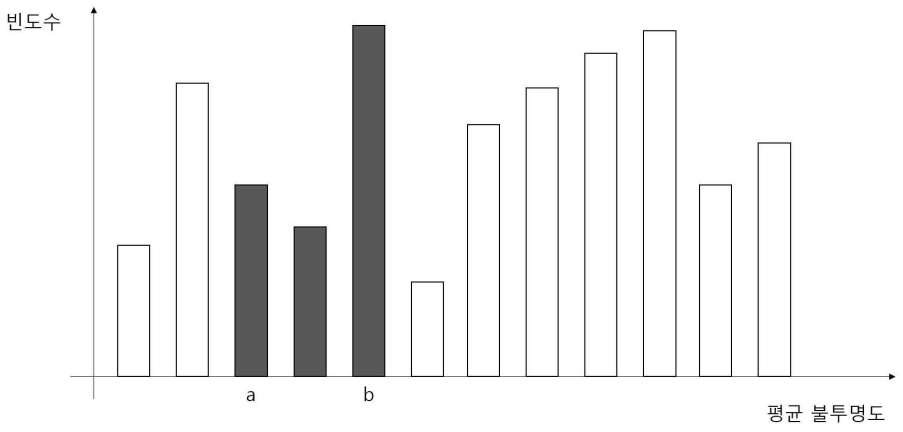
(a) (b)

[그림 10] (a) 표면 투명화가 적용되지 않은 볼륨 가시화 결과물, (b) 표면 투명화가 적용된 결과물

7.4 환경광 폐색 구현 및 실험 결과

다음은 환경광 폐색을 구현하는 알고리즘을 설명한다. 환경광 폐색을 적용하려면 6.2와 6.3에서 설명한 Web Assembly를 이용한 증분 알고리즘 모듈로 생성한 마스크 볼륨이 필요하다. 이 모듈은 Web Assembly와 증분 알고리즘에 의해 최적화 되어 빠른 속도로 마스크 볼륨을 생성하지만 실시간 가시화 환경에서 모든 프레임 마다 마스크 볼륨을 생성하는 것은 비효율적이다. 따라서 마스크 볼륨이 생성하는 시점은 프로그램이 시작되었을 때와 전이함수가 변경되었을 때로 제한한다.

마스크 볼륨의 히스토그램



[그림 11] (a) 마스크 볼륨의 히스토그램

[그림 11]은 마스크 볼륨의 히스토그램을 예시로 나타낸 그림이다. [그림 11]의 a부터 b 까지의 구간이 움푹 파인 부분이라고 가정한다면, 구간 a부터 b까지의 복셀값들의 색상을 더욱 어둡게 만들어 음영을 강조해야 환경광 폐색 기법이 적용된다. 따라서 사용자는 환경광 폐색 기법을 적용하려면 임의의 a, b 구간을 정하여 움푹 파인 구간을 지정해야 한다. 구간 a, b는 적용되는 전이함수와 사용자 판단하는 움푹 파인 정도에 따라 변하기 때문에 이를 정하는 방법은 전적으로 사용자의 경험에 의해 결정된다. 다음은 구간 a와 b가 주어졌을 때에 음영을 강조하기 위한 환경광 폐색 알고리즘 이다.

Algorithm 5 : 환경광 폐색 알고리즘

input : 3차원 볼륨 데이터 V, 전이함수 TF, 광선 투사 시작 위치 RSP, 광선의 투사 방향 RD, 임의의 값 a, b, 마스크 볼륨 Mask

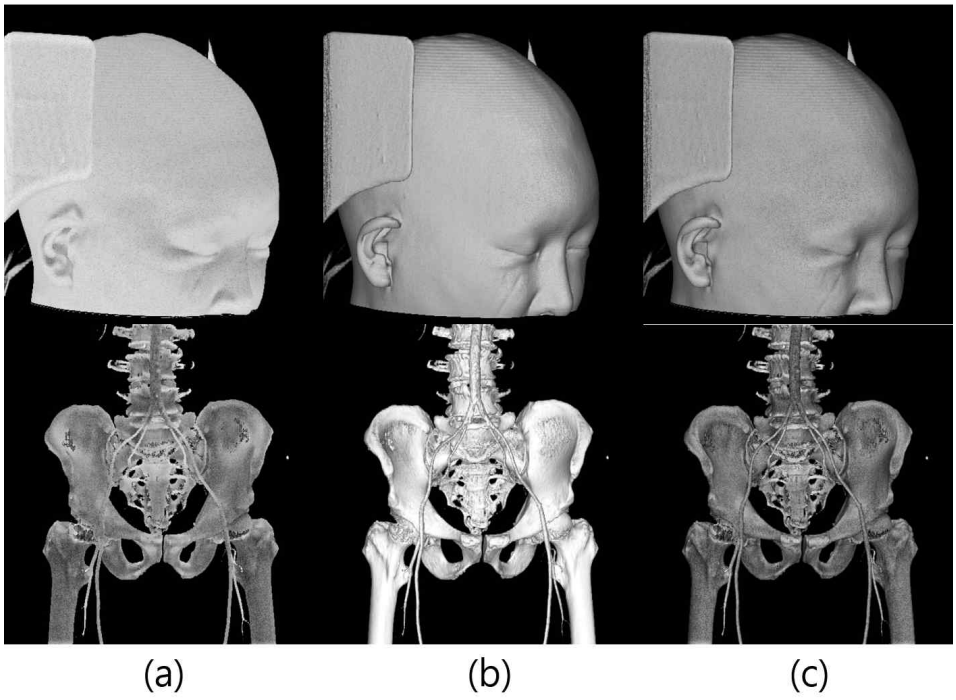
```
1   For 모든 RSP에 대하여 Do
2       P ← RSP
3       For 광선 투사 범위에 해당하는 모든 P에 대하여 Do
4           color, alpha ← TF(V[P])
5           color, alpha ← Alpha Blending(color, alpha)
6           M ← Mask[P]
7           If M < a Then
8               shade ← 1
7           Eles If a < M < b Then
8               shade ← 1 − (M − a)/(b − a)
7           Eles If M > b Then
8               shade ← 0
9           End If
10          result ← color × shade, alpha
11      End For
12  End For

13  Return result
```

[표 5] 환경광 폐색 알고리즘

[그림 12 (a)]는 불투명도 전이함수를 이용하여 생성된 마스크 볼륨 데이터를 가시화한 이미지이며 [그림 12 (b)]는 골반 데이터를 이용하여 일반적인 볼륨

가시화 결과 이미지이다. 실험의 최종 결과물인 [그림 12 (c)]는 생성된 마스크 볼륨 데이터를 이용하여 환경광 폐색 기법으로 볼륨 가시화를 수행하여 얻은 이미지이다. [그림 12 (c)]는 [그림 12 (b)]에서 표현되지 못한 음영을 강조하여 더욱 사실적이고 입체적인 이미지를 표현하고 있다.



[그림 12] 환경광 폐색과 일반 볼륨데이터의 결과 비교.
(a) 마스크 볼륨 데이터 영상,(b) 일반적인 볼륨 가시화 영상,(c) 환경광 폐색 영상

VIII. 결론

기존 VTK를 이용한 의료 영상의 볼륨 가시화는 유지보수가 어렵고 운영 체제에 종속되어 동작하는 단점을 가지고 있다. 이러한 문제점들을 해결하기 위해 VTK.js를 이용하여 웹 환경에서 의료 영상 볼륨 가시화 시스템을 제작하였다. 그러나 VTK.js는 기존 VTK의 모든 기능들을 지원하지 않고 있으며 사용자가 필요로 하는 고급 기능들을 지원하지 않는다. 본 논문에서는 VTK.js에서의 사용 중인 셰이더 프로그램을 분석 및 확장하여 기존 셰이더 프로그램에 없는 기능들을 확장할 수 있었고 이를 증명할 예로써 전-적분 전이함수를 이용한 노이즈 제거 기능, 기울기 추정을 통한 표면 투명화 기능, 환경광 폐색을 이용한 고품질 음영 생성 기능을 구현하는데 성공하였다. 기존 전이함수를 수치적분 하여 만들어진 전-적분 전이함수를 이용해 기존 전이함수의 문제점인 날카로운 그래프의 전이함수를 사용시 생기는 노이즈 문제를 해결하였다. 기울기 추정을 통한 표면 투명화 기능은 각 복셀의 기울기를 추정하여 인체조직간의 분리를 통한 경계면을 검출하여 조직 내부는 투명하게 하고 표면은 강조한 볼륨 가시화를 구현하였다. 고품질의 음영을 생성하는 환경광 폐색 기능을 활용하기 위해 필요한 마스크 볼륨은 증분 알고리즘을 통해 생성 알고리즘 연산을 최적화 하였으며, Web Assembly기술을 적용시켜 마스크 볼륨 생성 기능을 모듈화하여 기존 자바스크립트에서 동작하는 속도보다 더욱 빠른 속도로 동작하는데 성공하였다. 구현된 세가지의 기능은 VTK.js에서 제공하는 프레임워크에 성공적으로 통합되어 웹 브라우저 환경에서 GPU를 활용하여 실시간으로 볼륨 가시화가 가능하였으며, VTK.js에서 제공하는 UI 및 위젯을 활용하여 빠른 시간에 사용자에게 적합한 웹 서비스를 제작할 수 있었다. 기존에 제공하지 않는 다양한 셰이더 프로그램을 직접 제작하여 VTK.js의 기능을 확장할 수 있기 때문에 향후 더 많은 다양한 기능을 제작할 수 있다.

참 고 문 헌

1. 국내문헌

강지선, 이정진, 신영길, 김보형, (2019). 직접 볼륨 렌더링에서 사실적인 고속 피사계 심도 렌더링. 『한국차세대컴퓨팅학회 논문지』, 15(5), 75-83.

남진현, 계희원, (2015). 지역적 통계량을 이용한 고속 환경-광 가림 볼륨 가시화. 『멀티미디어학회 논문지』, 18(2). 158-167.

2. 국외문헌

Drebin, R, A. Carpenter, L. Hanrahan, P. (1988). Volume Rendering. ACM Siggraph Computer Graphics, 22(4), 65-74.

Fosner, R. (2003). Real-Time Shader Programming. Morgan Kaufmann.

Haas, A. Rossberg, A. Schuff, D. L. Titzer, B. L. Holman, M. Gohman, D. Bastien, J, F. (2017). Bringing the web up to speed with WebAssembly. ACM SIGPLAN Conference on Programming Language Design and Implementation, 38 , 185-200.

Kitware. (2017). Vtk.js. https://github.com/Kitware/vtk-js.

Lum, E, B. Ma, K, L. (2004). Lighting transfer functions using gradient

aligned sampling. IEEE Visualization 2004, 289-296

Lamberti, F. Sanna, A. (2005). A Solution forDisplaying Medical Data Models on Mobile Devices, SEPADS, 5, 1-7.

Mobeen, M, M. Feng, L. (2012). High-performance Volume Rendering on the Ubiquitous Webgl Platform. In 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems, 381-388.

Mwalongo, F. Krone, M. Reina, G. Ertl, T. (2018). Web-based Volume Rendering using Progressive Importance-Based Data Transfer. In VMV, 147-154.

Noguera, J, M. Jimenez, K, R. (2015). Mobile Volume Rendering: Past, Present and Future. IEEE Transactions on Visualization and Computer Graphics, 22(2), 1164-1178.

Parisi, T. (2012). WebGL: Up and Running. O'Reilly Media.

Pharr, M. Green, S. (2004). Ambient occlusion. GPU Gems, 1, 279-292.

Roettger, S. Guthe, S. Weiskopf, D. Ertl, T. Strasser, W. (2003). Smart hardware-accelerated volume rendering. In VisSym, 3, 231-238.

Schroeder, W, J. Avila, L, S. Hoffma, W. (2000). Visualizing with VTK: a Tutorial. IEEE Computergraphics and applications, 20(5), 20-27.

Zhang, Q. (2019). Medical Data Visual Synchroni zation and Information Interaction using Internet-Based Graphics Rendering and Message-Oriented Streaming. Informatics in Medicine Unlocked, 17(100253).

Zhou, H. Qu, H. Wu, Y. Chan, M, Y. (2006). Volume Visualization on Mobile Devices. In 14th Pacific conference on computer graphics and applications, 76-84.

ABSTRACT

Web-based volume visualization study through extension of VTK.js

Lee, Yun-Ho

Major in Information Computer

Dept. of Computer Engineering

The Graduate School

Hansung University

When performing volume rendering using medical image, most applications that perform volume rendering are limited by operating systems and specific computer devices. To solve this problem, a volume rendering application was implemented using VTK.js, a web-based graphic framework, in a web environment that is relatively free from the operating system and specific computer devices. VTK.js uses a shader program to implement volume rendering. In this paper, we prove that more functions can be provided to users by modifying the shader program to extend the functions that did not exist before. For example, the pre-integration transfer function technique, which is a technique that removes noise generated when rendering the volume surface using a sharp-shaped transition function, and the simple volume rendering of the surface of human tissue without a complicated process A surface

transparency technique was implemented by estimating the slope. In addition, an ambient occlusion technique that generates high-quality shadows was implemented using an incremental algorithm and Web Assembly. We succeeded in integrating these three techniques into the shader program, a function provided by VTK.js.

【Keyword】Volume Rendering, VTK.js, Pre-Integration Transfer Function, Gradient Opacity, Ambient Occlusion, Web Assembly