ISBN 978-89-6211-285-6

VR 환경에서의 VTK의 사용

(Using VTK on VR environments)

허 영 주 (popea@kisti.re.kr)

한 국 과 학 기 술 정 보 연 구 원

Korea Institute of Science & Technology Information

목차

그림 차례

1. 서론

과학 시뮬레이션의 결과는 사람이 직관적으로 이해하기 어려운 숫자들의 나열인 경우가 대부분이다. 그러나 이런 수치 데이터를 가시화하면 보다 쉽게 이해할 수 있는 형태를 갖추게 된다. 즉, 과학 데이터 가시화 과정을 거치면 연구 과정에서 만들어진 방대한 양의 자료를 컴퓨터 그래픽스를 이용, 이미지를 생성해냄으로써 연구결과를 보다 포괄적으로 이해하여 문제를 빨리 해결할 수 있게 되는 것이다. 최근에는 VR 환경에서 과학 데이터를 가시화함으로써 사용자와 상호작용하면서 필요한 데이터를 가시화하는 기법을 많이 구현하고 있다. 그러나 VR 환경의 시스템적 한계 때문에 다양한 가시화 기법을 적용하는 데는 한계가 있다.

VTK는 3차원 컴퓨터 그래픽스, 이미지 처리 및 visualization에 주로 사용되는 공개 소스 소프트웨어 라이브러리다. VTK가 제공하는 기능은 매우 다양한데 스칼라, 벡터, 텐서, 텍스쳐 및 볼륨 데이터를 표현할 수 있는 자료 구조와 implicit modeling, polygon reduction, mesh smoothing, 컷팅(cutting), 컨투어링(contouring)과 같은 고급 모델링 기법을 제공하므로 과학 데이터 가시화에 매우 유용하다. 이런 다양한 기법을 제공하는 VTK를 VR 환경에서 사용할 수 있다면 VR 환경에서 보다 다양한 방식의 과학 데이터 이해가 가능해질 것이다. 그러나 유감스럽게도 VTK는 현재 Stereoscopic 기능만 지원할 뿐, 트래킹에 관련된 기능을 지원하거나 3차원 인터랙션같은 VR 기능을 지원하지 않는다. 이에, VTK의 방대한 기능을 VR 환경에서 이용하고자 하는 연구는 지속적으로 이뤄져 왔다.

VTK에 VR을 접목하는 방법은 크게 2가지로 나눠볼 수 있다. 우선 VTK의 actor를 VR 환경에서 주로 사용하는 씬 그래프(scene graph)로 변환하는 방식이 있고, VTK의 renderer를 VR 환경에 맞게 변경하는 방식이 있다. Actor 변환 방식은 렌더링이 복잡한 알고리즘에 의해 방해받지 않는다는 장점이 있는 대신, VTK 파이프라인과의 직접적인 인터랙션이 불가능하다는 한계를 갖게 되는데, 이 방식을 사용하는 대표적인 툴로는 vtkActorToPF나 VTK2CAVE를 들 수 있다. Renderer를 교체하는 방식은 VR 지원이 안되는 VTK의 renderer와 render window 클래스를 교체함으로써 렌더링을 VR 시스템이 수행하게 하는 방식으로 VTK의 거의 모든 기능을 그대로 사용할 수 있으며 구현하기도 그다지 어렵지 않다는 장점이 있다. vtkCave나 VR-VTK, ViSTA는 렌더러 교체 방식을 사용한다.

본 기술문서에서는 VTK의 다양한 기능을 VR 환경에서 이용하고자 하는 이런 다양한 연구에 대해 알아보고, 향후 적용 가능한 기법에 대해 알아보기로 한다.

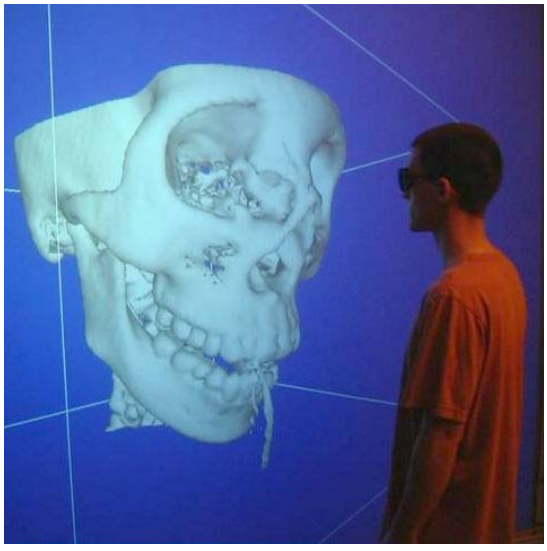
본 문서의 2장에서는 vtkActorToPF에 대해 설명하고 3장에서는 VTK2CAVE를 설명하겠다. 4장에서는 VR-VTK에 대해 설명한 다음, 5장에서는 ViSTA에 대해 알아보기로 한다.

2. vtkActorToPF

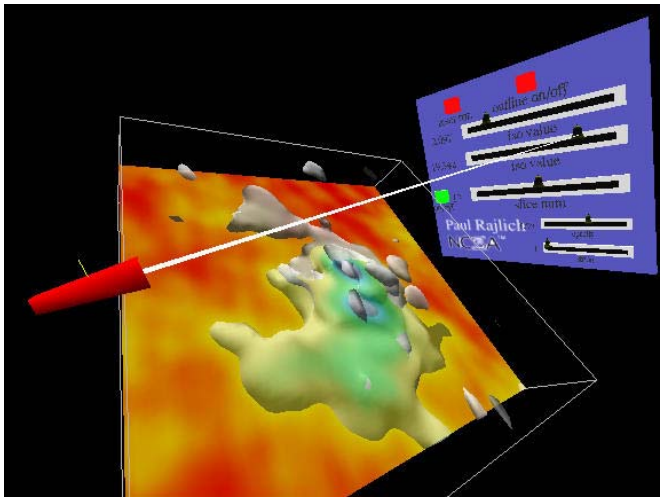
VTK는 기하정보를 actor라는 형태의 데이터 구조로 생성해서 렌더링하는데, vtk ActorToPF 함수는 이런 actor를 Performer의 씬 그래프(scene graph)에 포함된 pfGeode의 형태로 변환하는 역할을 수행한다. 즉, 사용자는 VTK를 이용해서 기하 정보를 추출해 내고 Iris Performer를 이용해서 그 정보를 렌더링하는 것이다. 이 때, 씬 그래프는 VTK와는 독립적으로 렌더링되며, CAVE같은 환경을 위한 다중 채널 렌더링이 가능해진다.

또 다른 방식은 vtkActorToPFTranslator 클래스를 사용하는 것이다. 이 클래스를 사용하면 특정 vtkActor와 pfGeode를 연결하는 변환 오브젝트를 생성할 수 있다. 이 경우에는 vtkActor에 변경 사항이 생길 때마다 pfGeode가 자동으로 변환된다.

따라서 사용자는 함수를 사용하거나 변환 오브젝트를 생성함으로써 vtkActor를 pfGeode로 변환할 수 있다. 이 때 vtkActorToPFTranslator 오브젝트를 사용하면 자동으로 Performer 콜백을 설정함으로써 vtkActor가 변경될 때마다 자동으로 pfGeode를 변환한다. 반면, vtkActorToPF 함수를 쓸 경우에는 vtkActor가 변경될 때마다 사용자가 pfGeode를 변환할 것을 명시해야 한다.



[그림 2-1] VisBox



[그림 2-2] CAVEvis

이 방식을 사용하면 복잡한 기하 정보 생성 때문에 렌더링이 방해받지 않는다는 장점이 있다. 그러나 사용자와의 인터랙션이 바로 VTK 파이프라인에 적용돼서 렌더링에 적용되지는 못한다는 한계를 갖게 된다.

vtkActorToPF를 사용한 사례는 매우 다양하다.

[그림 2-1]은 VisBox라는 애플리케이션을 이용해서 만든 간단한 isosurfacing 애플리케이션을 보여준다. [그림 2-2]에서는 과학 데이터를 가시화하는 CAVEvis 툴을 볼 수 있다. 이 툴은 천문학에서 주로 사용하는 데이터셋인 FITS와 HDF를 모두 지원하며, 반투명한 isosurface와 슬라이스를 모두 보여준다. 이외에도 VE-Suite, MPVR, BS-VE, LIMBO/VTK 등, 공학과 과학 분야의 다양한 애플리케이션과 툴을 지원한다.

vtkActorToPF는 IRIS Performer 인터페이스를 제공하는 모든 소프트웨어 툴킷에서 사용할 수 있는데, vtkActor만 지원할 뿐, vtkVolume은 지원하지 않는다.

3. VTK2CAVE

VTK2CAVE는 변환 프로세스의 말단에서 순수 OpenGL 명령어를 CAVE 라이브러리의 특정 프레임워크에 맞는 형태로 다시 생성한다. 따라서 이 툴을 사용하면 vtkActor를 VR 환경에서 디스플레이할 수 있다. VTK2CAVE는 매우 안정적이고 사용하기 쉬우며, 빌트-인된 공유 메모리를 사용한다.

VTK2CAVE를 사용하면 vtkActor는 새로운 vtkRenderer 클래스, 즉 cvtkCAVERenderer 클래스를 통해 직접적으로 변환되는데, 이 때 변환 프로세스는 cvtkCAVERenderer의 인스턴스에 vtkActor를 새로 추가함으로써 호출되게 된다. 이 vtkActor는 렌더링될 액터를 제어하는 리스트에 추가되며, 매번 프레임이 업데이트될 때마다 전체 리스트를 아이템별로 탐색해서 OpenGL 구문으로 변환하게 된다. cvtkCAVERenderer는 애플리케이션 프로그래밍 인터페이스 관점에서 봤을 때, vtkOpenGLRenderer를 변환한 것으로 볼 수 있다.

VTK2CAVE는 OpenGL 프레임워크를 따르기 때문에 OpenGL 인터페이스를 사용하는 모든 소프트웨어 툴킷에 적용할 수 있다. 그러나 VTK2CAVE 역시 vtkActor만 지원할 뿐, vtkVolume은 지원하지 않는다. 또, OpenGL 구문이 바뀔 때마다 매번 구문을 새로 추출해야 하는 번거로움이 있다. VTK2CAVE는 VTK Pipeline과 렌더링 프로세스를 공유 메모리를 사용하는 서로 분리된 프로세스로 구분해서 동작하기 때문에 성능이 매우 좋다는 장점이 있다.

4. VR-VTK

4장에서는 VR-VTK에 대해 알아보기로 한다.

VR-VTK는 VTK에 대한 multimodal VR 인터페이스로, 손으로는 위젯과 피커를 이용한 3차원 조작을, 발로는 클러칭을, 머리로는 카메라 조작을, 그리고 말로 시스템을 조절할 수 있게 해주는 인터페이스다.

가. 구조

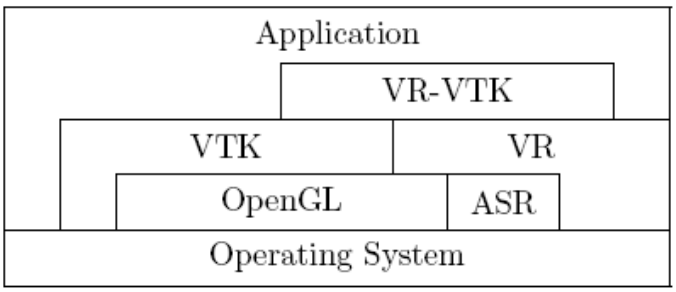
VR-VTK 레이어는 VTK에 VR 지원 기능을 더한 것으로, 몇가지 새로운 VR-VTK 클래스를 제공한다. 이 때 새로 추가된 VR-VTK 클래스는 인터랙션과 렌더링에 관련된 기능만 제공한다.

[그림 4-1]은 VR-VTK의 구조를 나타낸다.

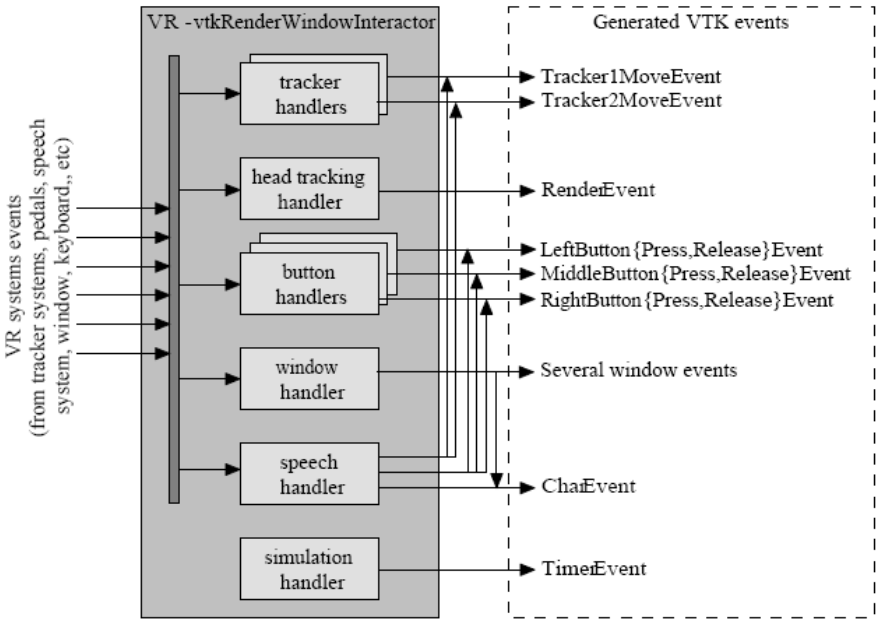
기본적으로 VR-VTK는 VTK의 렌더링 관련 클래스와 인터랙션 관련 클래스를 교체함으로써 VR 기능을 수행한다. 렌더링 관련 기능은 vtkRenderWindow와 vtkRenderer 클래스를 교체한 것으로, VR 시스템의 씬 렌더링(scene rendering) 과정을 제어한다. VR-vtkRenderWindow 클래스는 렌더링에 대해 쓰레드를 따로 생성함으로써 인터랙션과 렌더링을 분리한다. 헤드 트래커는 렌더링 쓰레드를 구동시키는 render 이벤트를 호출하며, Render 메소드는 render 이벤트를 호출한다.

인터랙션 관련 클래스는 VR-vtkRenderWindowInteractor 클래스로 vtkRenderWindowInteractor 클래스를 교체한 것이다. 이 클래스는 VR 시스템으로부터의 이벤트를 VTK 파이프라인이 해석할 수 있는 형태의 이벤트로 변환하는 역할을 담당한다. 여기에서 새로 생성한 이벤트 핸들러는 다음과 같다.

- 2개의 트래커 핸들러 (양손에 각각 하나씩): Tracker1MoveEvent, Tracker2



[그림 4-1] VR-VTK의 구조



[그림 4-2] VR-VTK 렌더 윈도우 인터랙터

MoveEvent

- 헤드 트래킹 핸들러
- 3개의 버튼 핸들러
- 윈도우 핸들러: 윈도우 시스템으로부터 이벤트를 받는 역할 담당 (키 누름, 윈도우 이벤트 등)
- 스피치 핸들러: 외부 ASR(Automatic Speech Recognition) 시스템으로부터 이벤트를 받아서 VTK 이벤트(문자 이벤트, 버튼 이벤트, 혹은 트래커 이벤트)로 변환하는 핸들러.
- 시뮬레이션 핸들러: 시뮬레이션이나 애니메이션을 위한 핸들러로, 사용자가 지정한 빈도로 VTK 타이머 이벤트를 생성하는 역할 수행.

vtkInteractorStyle 클래스를 대체하는 VR-vtkInteractorStyle 클래스는 picking 프로세스와 인터렉션 메소드를 제어하고 depth perception과 가시화를 향상시키는 기능을 제어한다.

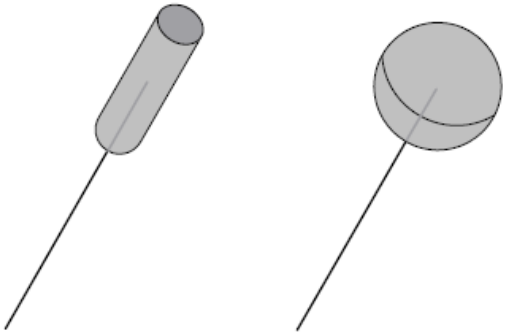
모든 VR-VTK 클래스는 플랫폼에 독립적이며, 대부분의 클래스는 VTK 이벤트에 의해서만 제어된다. VR-VTK 클래스는 PVR 라이브러리를 이용해서 구현했다.

나. 3D 인터랙션

VR-VTK은 여러 다양한 형태의 3차원 인터랙션을 제공한다.

1) 3D 커서 리젼(3D Cursor Region)

3D 커서 리젼은 3차원 공간에서 선택을 단순화하기 위한 공간으로, 리젼 안에 들어가는 오브젝트는 3D 커서 리젼에 의해 선택된 오브젝트다. 3D 커서 리젼은 구 형태와 원통 형태의 2가지가 있으며, 사용자의 손으로부터의 거리와 리젼 크기를 파라미터로 지정할 수 있다.

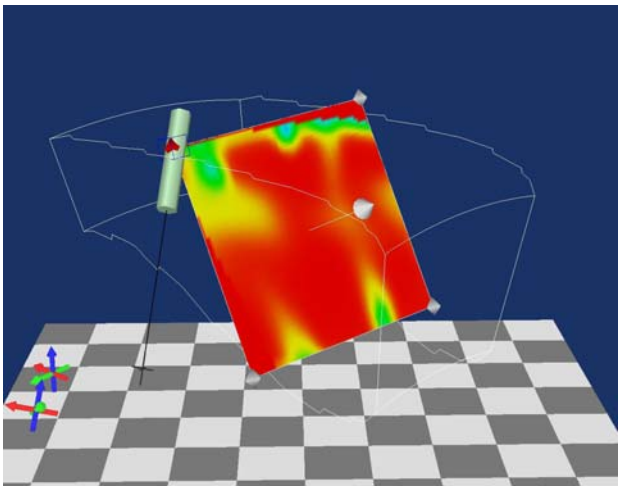


[그림 4-3] 3D 커서 리젼

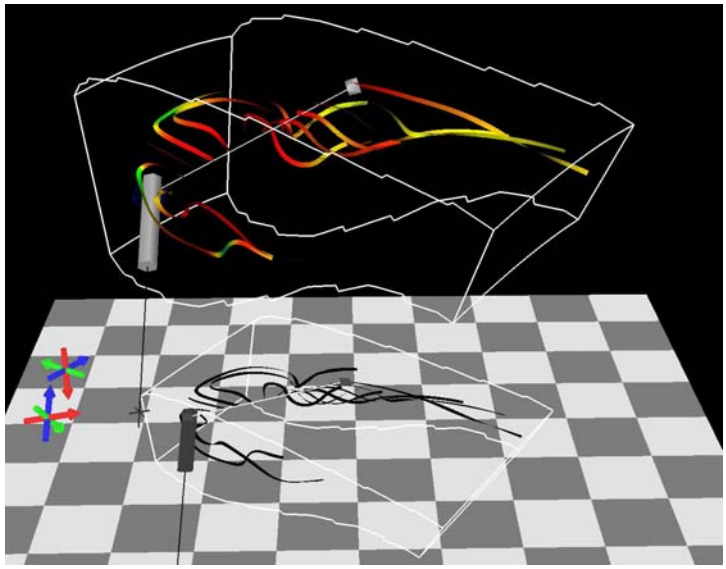
2) 3D 위젯

데이터 가시화와 관련된 모든 3D 인터랙션은 3D 위젯을 조작함으로써 이뤄진다. 위젯 조작은 VR-vtkWindowInteractor로부터의 이벤트와 커서 리젼의 위치에 의해 영향을 받는다.

위젯과 관련된 인터랙션은 다음과 같다.



[그림 4-4] 3D 위젯



[그림 4-5] VR-VTK의 스테이지와 그림자

- 위젯 기하 정보에 대한 3차원 조작: 트래커는 3차원 공간상에서 위젯의 핸들을 선택하고 움직이는데 사용되는 커서 리젼을 제어한다.
- 트래킹과 혼합된 스피치: 위젯의 기하 정보를 제어한다. 즉, 3D 트래커(커서 리전)은 위치를 지정하고, 스피치 명령으로는 핸들과 액션을 지정할 수 있다. (ex. "put that handle there")
- 전체 위젯을 제어하는 스피치: 스피치 명령으로 위젯에 대한 액션을 지정할 수 있다. (ex. "rotate that widget by...")

[그림 4-4]의 3D 위젯은 슬라이스 평면을 제어하는 평면 형태의 위젯이다. 이 위젯은 크기를 조절하는데 모서리에 붙어있는 핸들을 사용하고, 화살표 모양의 핸들을 사용해서 평면의 각도를 조절한다. 또, 평면을 잡고 조작하면 평면 자체를 움직일 수도 있다.

이외에도 VR-VTK에서는 World in hand, 양손 입력 등의 향상된 3차원 인터랙션과 여러 다양한 깊이 인식 정보를 제공한다. VR-VTK에서 제공하는 깊이 인식 관련 기능으로는 Motion parallax, 스테이지, 그림자 등이 있다.

다. 애플리케이션 제어

가시화 메소드를 사용 가능한 상태로 만들거나 파라미터 설정, 오브젝트를 위한 속성 설정, 컬러 테이블을 수정하는 등, 애플리케이션을 제어함으로써 사용자 작업을 수행해야 하는 경우는 매우 많이 발생한다. 여기에서는 VR-VTK에서 애플리케이션을 제어하는 방법에 대해 설명하겠다.

1) 스피치 입력

스피치 입력으로 제어할 수 있는 명령으로는 picking 컨트롤, 3차원 인터랙션 메소드의 설정, 덱스 큐의 설정 및 줌 기능 등을 들 수 있으며, 렌더링 프로세스를 제어하는 몇몇 명령을 포함한다. (입체 화면 제어 및 스크린 덤프 생성)

스피치 핸들러는 스피치 이벤트를 문자 이벤트로 변환하는 역할을 수행하며, 애플리케이션은 실제적으로는 문자 이벤트를 인식한다.

2) 2D 위젯

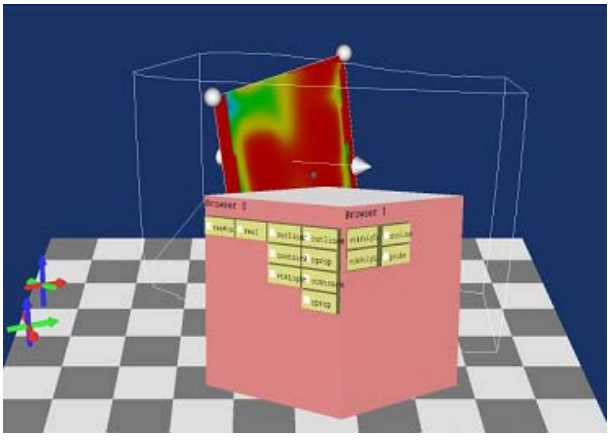
가) VR-VTK UI

VR-VTK UI 툴킷은 2D 위젯(버튼, 메뉴, 슬라이더 등)을 포함하는 가상 큐브와 선택용 디바이스인 가상 펜으로 구성된다. 가상 큐브와 가상 펜은 트래커로 제어할 수 있으며, 특정 버튼(e.g. 페달)을 누르거나 스피치 명령을 내림으로써 가시화 인터랙션에서 UI로 제어가 변환된다. 이 상태에서 특정 버튼이나 스피치 명령을 다시 내리면 위젯 인터페이스가 사라지고 데이터 가시화 인터랙션이 재개된다.

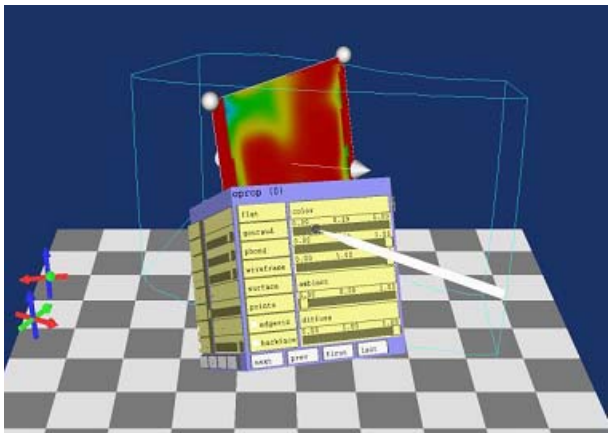
나) VTK 파이프라인 브라우저

VTK 파이프라인 브라우저는 VR-VTK UI 툴킷에 구현된 GUI 중 하나로, 사용자가 VTK 파이프라인 상의 모든 파라미터와 모든 오브젝트를 수정하거나 감시할 수 있게 해주는 툴이다.

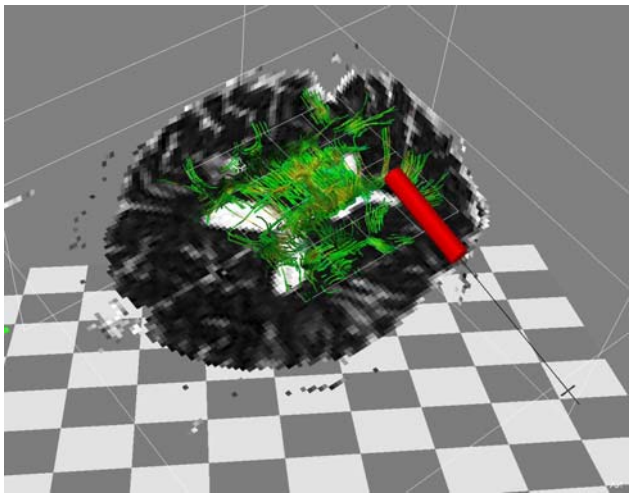
파이프라인 브라우저는 2개의 뷰(view)를 제공하는데, 한 뷰는 큐브의 하나 이상의 면에 연결된 그래프 형태를 보여주고(파이프라인 인터페이스), 다른 한 뷰



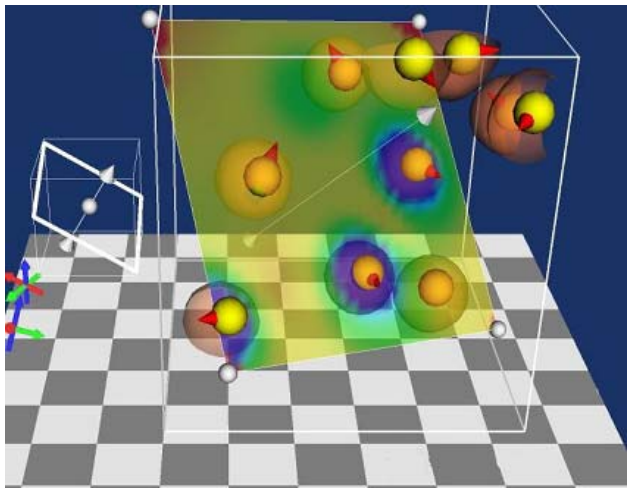
[그림 4-6] 파이프라인 인터페이스



[그림 4-7] 오브젝트 인터페이스



[그림 4-8] DTI: 슬라이스 플레인과 파이버 트래킹



[그림 4-9] 튀는 공에 대한 시뮬레이션

는 VTK 오브젝트에 대한 속성/메소드를 보여준다(오브젝트 인터페이스).

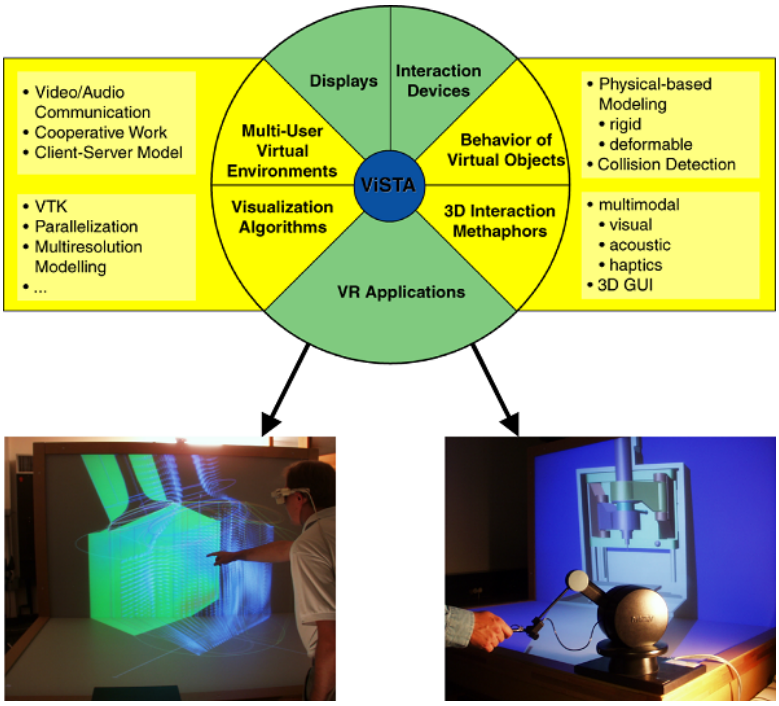
라. 실제 사례

여기에서는 VR-VTK를 이용한 몇 가지 애플리케이션을 소개한다.

[그림4-8]은 Diffusion Tensor Imaging을 나타내며, 스트림라인, 하이퍼스트림라인, 스트림튜브를 이용한 파이버 트래킹을 구현했다. 또, 스피치 입력을 받아들이며 VTK 파이프라인 브라우저 기능도 구현했다.

[그림 4-8]에서 볼 수 있는 사례는 일정 공간 안에서 튀는 공을 시뮬레이션 한 것으로, 사용자는 공을 선택해서 옮길 수도 있다. 이 애플리케이션은 공간내 한 지점에 가해지는 압력을 시뮬레이션 한 것으로, 슬라이스 플레인 위에 데이터셋에 가해진 압력을 컬러로 맵핑해 놓았다. 이 애플리케이션 역시 스피치 입력과 VTK 파이프라인 브라우저, VR-VTK UI 기능을 구현했다.

VTK-VR은 가시화와 VR 공간에서의 직접적인 데이터 조작이 가능하다는 것을 보여줬으며, 다양한 형태의 인터랙션을 VTK에 접목시켜 제공했다는 데서 그 의의를 찾아볼 수 있다.



[그림 5-1] Vista의 기능 및 사례. 왼쪽)엔진 실린더 내의 공기의 흐름을 VR-기반으로 가시화한 결과 오른쪽) 포스 피드백(force feedback)이 있는 로봇의 운동의 VR 프로토타입

5. ViSTA

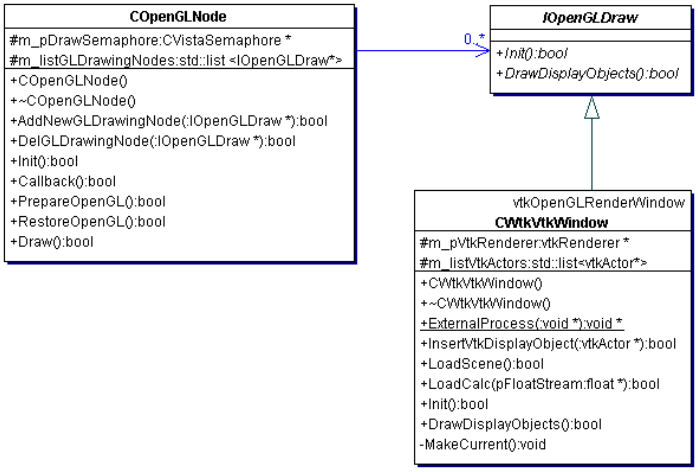
ViSTA는 Virtual Reality Software University of Technology Aachen의 약자로, 수치 시뮬레이션, 가상 프로토타이핑, 구조, 약학 및 심리학 분야에 대한 가시화를 주 타겟으로 한다.

ViSTA는 WTK(WorldToolKit)라는 상용 C 라이브러리를 사용하는데, 이 라이브러리는 SGI Irix, HP UX, Sun Solaris, LINUX 및 Win32 등의 다양한 플랫폼을 지원하며, 씬 그래프 렌더링이나 다양한 디스플레이 장치 및 인터랙션 디바이스에 대한 인터페이스를 제공한다.

[그림 5-1]은 ViSTA의 기능에 대해 보여주는데, ViSTA는 특히 multimodal 인터랙션의 구현에 개발의 초점이 맞춰져 있다.

가. VTK와 ViSTA

ViSTA는 vtkCave를 기반으로 VR 기능을 구현했다. vtkCave가 VTK2CAVE에 비해 구현이 간단하고 유연성이 높기 때문이다. vtkCave는 vtkCaveRenderer와 vtkCaveRenderWindow 클래스를 추가함으로써 VR 기능을 구현한 라이브러리다.



[그림 5-2] vtkActor의 변환에 대한 개념

VTK의 vtkOpenGLRenderWindow에서 발생하는 모든 윈도우 호출은 VR 기능을 탑재한 클래스로 대체됐으며, VTK의 카메라 기반 렌더링 기능은 제거됐다.

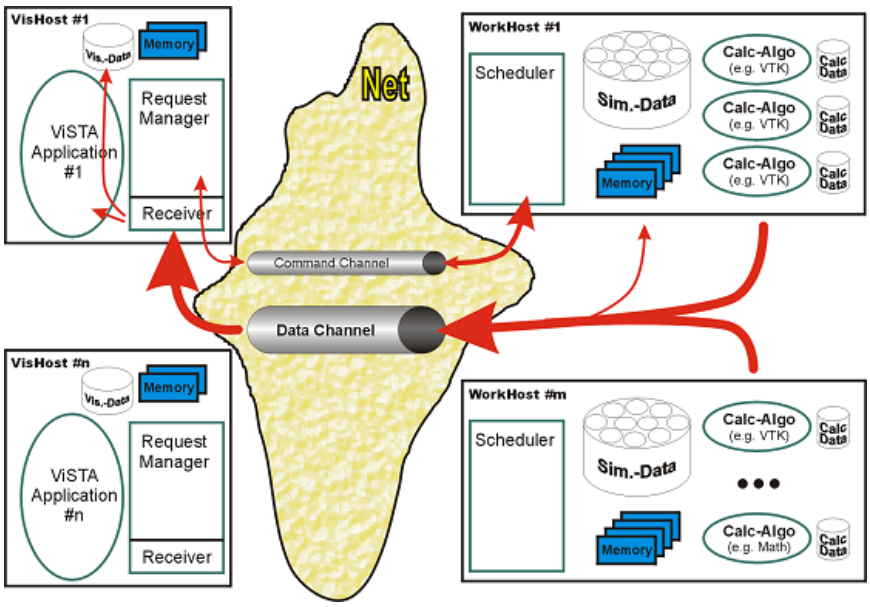
ViSTA에서는 vtkCave에 3개의 클래스가 추가됐는데, 이 추가된 클래스는 [그림 5-2]와 같다. COpenGLNode, IOpenGLDraw, CWtkVtkWindow, 이 세 클래스는 vtkActor 클래스를 대체하며, 각 클래스의 역할은 다음과 같다.

- IOpenGLDraw: 이 클래스는 다른 두 클래스에 대한 추상 클래스로 모든 actor에 대한 drawing 메소드인 DrawDisplayObjects() 메소드를 제공한다. 또, 이 클래스는 이 인터페이스로부터 파생된 모든 오브젝트(ex. OpenGL 오브젝트, VTK actor, etc.)를 추상화한다.
- COpenGLNode: 이 클래스는 이 인터페이스로부터 파생된 모든 오브젝트를 수집하는 역할을 하며, 특정 WTK OpenGL 노드를 WTK의 씬 그래프에 끼워 넣는다. 이 클래스는 내부 리스트를 탐색하는 콜백으로 구성돼 있으며, IOpenGLDraw 인터페이스를 소유하는 모든 오브젝트를 호출한다.
- CWtkVtkWindow

나. VTK 인터페이스의 병렬화

ViSTA는 병렬화된 기능을 제공하는데, work host와 visualization host를 써서 병렬 기능을 제공한다. work host 는 시간이 많이 걸리는 작업을 수행하는 병렬 기기를 가리키고 visualization host는 계산 결과를 가시화하는 기기를 가리킨다.

ViSTA의 병렬화 기능 및 그 과정은 [그림 5-3]에서 한눈에 볼 수 있다.

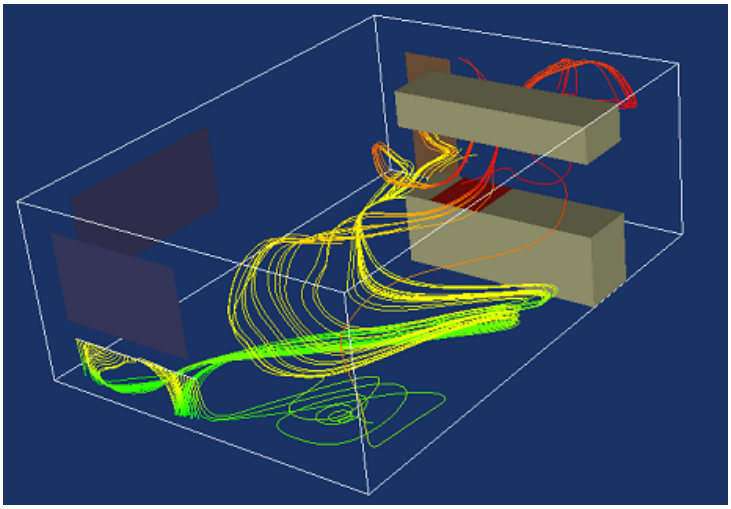


[그림 5-3] 병렬화된 데이터 흐름

각각의 visualization host는 똑같은 ViSTA 애플리케이션을 실행한다. 사용자 명령에 대한 응답으로 ViSTA 내부에서 request가 생성된다. 이 때 사용자 명령은 특정 지점에서 스트림라인을 계산해서 디스플레이하라는 식의 명령이 가능하다. 각 request에는 우선순위가 할당되는데, 이 우선순위는 request가 처리되는 순서를 결정짓는다. 그런 다음 이 request는 request manager로 전달되며, request manager는 이 request를 수행할 work host를 선정한다. request manager는 ViSTA의 일부이며, 서로 동기를 맞춤으로써 한 work host에 너무 많은 부하가 걸리지 않도록 조절하는 역할을 수행한다. request manager로부터 전송된 request는 work host의 scheduler가 받는다.

work host에 위치한 scheduler는 request를 처리하는 데 필요한 최소/최대 노드 개수를 산정한다. 노드 개수는 계산속도, 가용 메모리 및 네트워크 용량에 따라 달라진다. 이제 request는 scheduler의 작업 큐(queue)에 추가되며, 작업 큐 내에서는 request를 우선순위에 따라 순서를 배정한다. 가장 우선순위가 높은 request 처리에 필요한 만큼의 노드가 확보되면 scheduler는 실제 노드를 선택하고, 선택된 노드에서는 계산을 시작한다. 이 계산 결과는 request를 보낸 visualization host의 receiver로 전송된다. 이제 receiver는 애플리케이션에 결과를 넘겨줌으로써 request의 처리가 끝나게 된다.

이런 구조로 설계된 ViSTA 병렬 버전의 첫 프로토타입은 MPI를 이용해서 구현됐다. 이 프로토타입은 단일 visualization host와 work host만 지원한다. 이 프로



[그림 5-4] Kitchen 데모

토타입에는 스트림라인을 계산하는 간단한 병렬 함수가 구현돼서 사용됐다. 이 함수의 스트림라인 계산 과정은 다음과 같다.

- 병렬 work host의 각 노드에서 전체 데이터셋을 읽어들인다.
- 스트림라인의 계산은 사용가능한 노드에 똑같이 나뉘어서 수행되며, 각 노드에서는 다른 노드와는 독립적으로 결과를 계산한다.
- visualization host는 각 request에 대해 원칙적으로 하나의 결과만 기대하기 때문에 계산된 스트림라인은 한 노드에서 합쳐진 다음, visualization host로 전송된다.

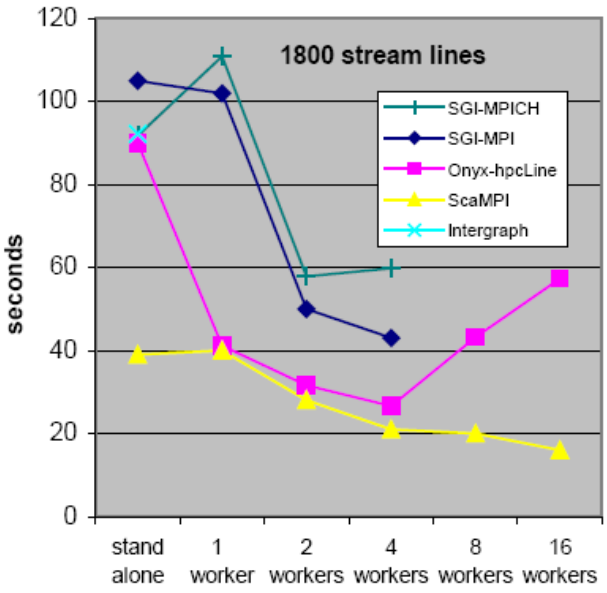
이 때, visualization host의 request manager와 receiver는 서로 다른 스레드를 사용해서 구현되는데, manager와 receiver는 공유 메모리를 통해 서로 데이터를 교환한다.

프로토타입의 데모에 사용한 애플리케이션은 VTK의 Kitchen 데모를 이용한 것으로, 이 데모 데이터는 부엌의 공기 흐름을 CFD로 분석한 것이다. 이 데이터는 크기가 그다지 크지 않은 structured grid 형태의 데이터로 28 x 24 x 17개의 그리드로 구성돼 있다. visualization 데이터셋은 해당 지점에서의 압력을 나타내는 스칼라 값을 포함한다.

이 프로토타입은 SGI Onyx-2 Infinite Reality 2를 가시화 엔진으로 하고 16대의 PC 노드(2 x Intel-PII 프로세서, 400MHz, 512KByte level-2 캐시, 512M Byte 메모리)를 갖춘 리눅스 클러스터를 컴퓨팅 시스템으로 테스트했다.

이런 시스템에서 여러 조건으로 테스트한 결과는 [그림 5-5]와 같다. 여기에서

의 시간은 visualization host에서 일어나는 초기화 이벤트에서부터 결과를 받을때까지 측정했으며, 결과적으로 Linux 클러스터가 SGI 보다 훨씬 빠르다는 것이 입증됐다. 물론, 네트워크의 속도도 실행 시간에 영향을 미치며, 이것은 hpcLine (계산 시스템)만 사용했을 때와 SGI/hpcLine을 엮어서 사용했을 때의 결과를 비교해보면 알 수 있다.



[그림 5-5] 실행 시간으로 본 결과

현재 ViSTA는 여전히 개발되면서 사용되고 있으며, 각종 가시화 관련 애플리케이션의 기반이 되는 라이브러리 개발에 이용되고 있다.

6. 결론

컴퓨팅 환경의 수준이 높아지고 사용자들의 요구가 높아짐에 따라 VR 환경에서 사용자 인터랙션을 받아서 바로 처리할 수 있게 하는 인터페이스의 개발이 늘어나고 있다. 과학 데이터를 가시화하는 분야에서도 이 현상은 예외는 아니어서, 여러 분야의 데이터를 VR 환경에서 가시화하고, 다양한 파라미터를 적용해서 그 결과를 VR 환경에서 바로 확인해 보는 등, 과학 데이터 가시화 분야에서도 VR 인터페이스에 대한 필요성은 높아지고 있다.

그러나 VR 환경은 시스템의 특성상 여러 제약사항을 갖고 있으며, 특히 과학 데이터를 가시화하기 위한 알고리즘이 잘 갖춰져 있는 라이브러리가 존재하지 않는다는 단점을 갖고 있다. 이런 단점을 극복할 수 있는 것이 VTK와의 접목이다. VTK는 다양한 과학 분야의 데이터를 다양한 방법으로 가시화할 수 있게 해주는 라이브러리로, 이미 여러 분야에서 안정적으로 사용해서 다양한 애플리케이션이 나와 있는 툴이다. 따라서 VTK와 VR 환경을 접목시키면 VTK의 방대한 알고리즘을 사용하면서 보다 편리하게 과학 데이터를 가시화할 수 있다.

이렇게 VTK와 VR을 접목시키려는 노력은 vtkActorToPF, VTK2CAVE, VR-VTK, ViSTA 등으로 이어져 왔으며, 각각의 방식은 VTK의 다양한 알고리즘을 VR 환경으로 끌어들이기 위한 노력으로 이어져 왔다. 각 방식은 각자 고유의 장단

점을 갖고 있는데, 구현에 있어서 중요한 것은 주로 사용자 인터페이스의 구현 방식이다. 사용자에게 보다 편리한 인터페이스를 제공하기 위해 다양한 형태의 multimodal 인터랙션을 제공하기도 하고, 3차원 공간상에서의 복잡한 인터랙션을 단순화해서 사용자에게 편리한 환경을 제공하려는 노력이 있어왔다. 이런 노력은 향후에도 계속 이어져나갈 것이며, 이런 기존의 알고리즘을 이용해서 사용자에게 더욱 발전된 형태로 보다 편리한 인터페이스를 제공하기 위한 연구는 앞으로도 계속 될 것이다.

7. 참고문서

[1] http://brighton.ncsa.uiuc.edu/prajlich/vtkActorToPF/

[2] Arjan J.F.Kok & Robert van Liere, "A Multimodal Virtual Reality Interface for 3D Interaction with VTK", Knowledge and Information Systems, vol 11(3), 2007.

[3] Thomas van Reimersdahl, Torsten Kuhelen, Andreas Gerndt, Jorg Henrichs, & Christian Bischof, "ViSTA: A Multimodal, Platform-independent VR-TOolkit based on WTK, VTK, and MPI", Proceedings of 4th International Immersive Projection Technology Workshop, 2000.