ISBN 978-89-6211-283-2

Grid Visualization

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. 머리말

그리드 컴퓨팅은 곳곳에 산재해 있는 컴퓨팅 리소스를 동적으로 묶어서 분산 컴퓨팅을 수행하는 모델로, 규모가 크거나 처리 시간이 긴 작업을 해결하는데 유용한 모델이다.

과학 데이터를 가시화한다는 것은 이론상으로 믿을만한 예측을 하기에 너무 복잡한 현상, 또는 실험실에서 수행하기에는 너무 위험하거나 비용이 많이 드는 현상 등을 연구하는 과정에서 만들어진 방대한 양의 데이터를 컴퓨터 그래픽스 기술을 이용, 이미지를 생성함으로써 계산 분야에 종사하는 과학자/공학자가 연구 결과를 보다 신속하게, 포괄적으로 파악할 수 있게 해주는 것이다.

이렇게 과학 데이터를 가시화하는 기존 방식은 배치 프로세스의 결과로 나온 계산 결과를 사용자의 워크스테이션으로 다운로드한 뒤, 로컬 워크스테이션에서 가시화하는 것이었다. 이런 방식에서는 시뮬레이션과 가시화 간의 연결이 정적인 엔드-포인트와 가시화 툴킷 사이에 맺어지기 마련이다. 그러나 최근에는 데이터의 용량이 커짐에 따라 경우에 따라서는 사용자의 로컬 워크스테이션만으로는 가시화를 수행할 수 없는 상황도 발생할 수 있다는 단점을 드러낸다. 게다가 대용량 데이터를 가시화하는 데는 처리 시간의 기하급수적인 증가도 감수해야 하는 것이다. 이런 문제를 해결하기 위해 제안된 방법이 그리드 환경에서의 가시화다. 이런 그리드 환경에서의 가시화 기법은 그리드 기술의 발전과 더불어 하나의 독립적인 프로젝트로 자리잡아 가고 있다.

본 문서에서는 그리드 환경에서의 가시화에서 고려해야 할 몇 가지 개념을 소개한 뒤, 관련 프로젝트, 특히 GVK(Grid Visualization Kernel)에 대해 자세히 소개하기로 한다.

2. 그리드 환경과 가시화

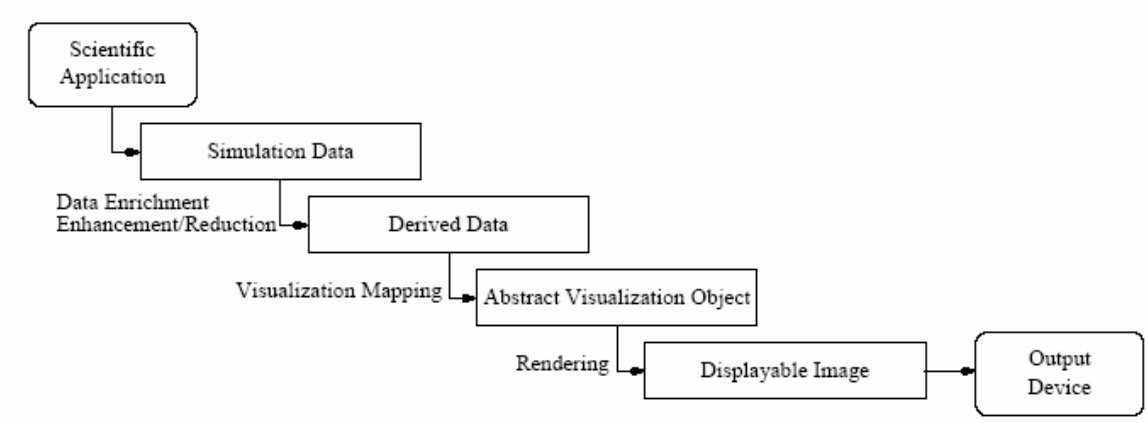
가. 개요

그리드 컴퓨팅은 계산과학이나 공학 분야에서 컴퓨팅 파워를 필요로 하는 애플리케이션을 위해 대규모 분산 컴퓨팅 환경을 열어 주기 위한 한 방편이다. 그리드 컴퓨팅은 넓은 지역에 걸쳐 퍼져 있는 다양한 계산 리소스 및 정보를 통합해서 사용하는 과정을 단순화하는 것을 그 목적으로 하며, 고성능 컴퓨팅 기능에 대한 자유로운 접속을 제공한다. 따라서 그리드 컴퓨팅이 기저로 삼는 하드웨어의 복잡성은 미들웨어 레이어에 의해 숨겨지며, 미들웨어 레이어는 물리적인 하드웨어상에서 그리드 애플리케이션의 실행을 투명하게 관리한다. 그리드 툴킷은 통신, 동기화, 오류 감지, 보안, 리소스 및 데이터 관리, 이식 등과 같은 기능을 위한 그리드 인프라 서비스를 제공하는데, 이런 그리드 툴킷으로는 글로버스(Globus), Legion, UNICORE 등을 들 수 있다.

전통적으로 원격 가시화는 2가지 방식으로 이뤄져 왔다. 그중 하나는 모든 가시화 과정을 서버에서 수행하고 이미지 데이터만 클라이언트로 전송하는 방식으로, 이 방식은 대용량 데이터를 가시화하는데 유리한 방식이다. 또다른 방식은 대용량 데이터의 일부를 서버에서 클라이언트 워크스테이션으로 전송한 다음, 이 일부 데이터를 클라이언트의 데스크탑에서 가시화하는 방식이다. 이 방식은 사용자와의 상호작용이 중요시되는 애플리케이션에서 주로 사용된다. 최근에는 이런 원격 가시화에 그리드 미들웨어를 이용, 데이터 생성, 저장에서 가시화에 이르는 일련의 단계를 수행하는 기술이 많이 연구되고 있으며, 일부는 실용화 단계에 와있는 것도 있다.

나. 가시화 파이프라인 구성

일반적으로 가시화 파이프라인은 [그림 2-1]과 같이 구성된다. 계산과학 애플리케이션의 시뮬레이션 과정을 거쳐 생성된 데이터는 데이터 조작 과정을 거쳐서 가시화에 적합한 형태로 바뀌게 되며, 이렇게



[그림 2-1] 가시화 파이프라인

가시화할 데이터는 맵핑 과정을 거쳐서 추상 가시화 오브젝트로 변환된다. 이 가시화 오브젝트를 렌더링하면 최종 이미지가 생성되는 것이다.

애플리케이션의 실행 결과로 나온 시뮬레이션 데이터를 실제로 가시화할 데이터로 가공하는 과정을 필터링(filtering)이라 한다. 이 과정에는 데이터 강화/축소 과정이 포함된다. 필터링 과정은 raw 데이터에 적용되며, 이 과정을 거치면 데이터 추출, 삽입, 간략화 등의 과정을 거쳐 실제로 가시화에 필요한 데이터만 남게 된다.

가시화에 필요한 데이터를 추상 가시화 오브젝트(AVO, abstract visualization object)로 변환하는 과정을 가시화 매핑(visualization mapping)이라 한다. 이 과정은 추상 가시화 오브젝트를 조성하는 과정으로 가시화할 데이터를 지형, 시간, 색상, 투명도, 광도, 텍스쳐와 같은 추상 가시화 오브젝트 속성에 대응시키는 과정이다.

이렇게 생성된 추상 가시화 오브젝트를 가지고 최종 이미지를 생성하는 과정이 바로 렌더링(rendering) 과정이다. 이 과정에서는 (뷰 전환, 은면 제거, 쉐이딩 등과 같은) 컴퓨터 그래픽스나 이미지 프로세싱 기술을 통해 추상 가시화 오브젝트로부터 이미지를 생성한다.

이렇게 가시화 파이프라인은 크게 필터링, 가시화, 그리고 렌더링 과정으로 구분할 수 있다. 이 가시화 파이프라인을 최적화하는 방안은 여러 가지가 있는데, 그중에는 일부 프로세서를 병렬로 처리하는 방식이나 전처리 과정을 컴퓨팅 파워가 뛰어난 서버에서 실행함으로써 성능을 최적화하는 방식이 있다. 여기에서는 가시화 파이프라인을 재

설계함으로써 가시화 프로세스의 일부를 클라이언트에서 시뮬레이션 서버로 이관하는 방식에 대해 설명하겠다.

일반적으로 많이 사용되는 파이프라인 구성 방식은 다음과 같다.

- 방법 1. 클라이언트: 필터링 + 가시화 + 렌더링
- 방법 2. 서버: 필터링

  클라이언트: 가시화 + 렌더링
- 방법 3. 서버: 필터링 + 가시화

  클라이언트: 렌더링
- 방법 4. 서버: 필터링 + 가시화 + 렌더링

클라이언트에서 모든 파이프라인 과정을 수행하는 방법 1은 기존에 행해지던 방식과 다르지 않다. 이 방식은 입력 데이터를 원격지에 있는 서버로부터 전송 받아서 모든 파이프라인 프로세스를 클라이언트에서 수행한다.

방법 2는 서버에서 필터링 과정을, 그리고 클라이언트에서 가시화와 렌더링 과정을 수행하는 방식이다. 가시화 클라이언트로 전송되는 데이터는 데이터 감축 과정을 거치며, 결국 클라이언트에는 정말 가시화에 필요한 일부 데이터만 전송되게 된다. 이 방식은 간단한 데이터 구조에 의존하는 가시화 기술에 의해 제한을 받게 된다. 예를 들어 벡터 데이터로 구성된 유체 데이터를 리본 형태로 가시화한다고 했을 때, 전체 데이터가 아니라 벡터의 위치, 방향 및 크기만 클라이언트로 전송되는 것이다.

방법 3은 서버에서 필터링과 가시화 과정을, 클라이언트에서 렌더링을 수행하는 방식으로 서버는 모든 폴리곤 오브젝트가 생성될 때까지의 모든 파이프라인 단계를 구성해서 실행하게 된다. 이렇게 생성된 폴리곤 데이터는 가시화 사이트에 일종의 디스플레이-목록의 형태로 저장되며, 클라이언트에는 이 폴리곤 데이터가 전송된다. 따라서 뷰잉 파라미터만 변경되는 경우에는 클라이언트로 다시 폴리곤 데이터를 전송할 필요가 없다. 그러나 폴리곤 데이터의 크기는 다소 큰 경향이 있기 때문에 이런 지형 데이터를 전송하려면 네트워크의 대역폭이 커야한다는 단점이 있다.

방법 4에서는 서버에서 모든 파이프라인 과정을 수행하고 클라이언트에는 최종 완성된 이미지만 전송한다. 이 방식은 대용량 폴리곤 데이터 셋을 가시화 사이트에 전송해야 하는 상황에 적합한 방식인데, 뷰포인트의 변경에 대한 지연시간이 길다는 단점이 있다.

다. 그리드 기반 가시화 시스템에서 고려할 점

그리드를 기반으로 하는 가시화 시스템을 설계할 때 특히 고려해야 할 사항은 다음 3가지다.

(1) 시스템 이종성(heterogeneity)의 극복

(2) 시스템의 처리량(throughput) 최대화

(3) 지연 시간의 최소화

시스템의 이종성에 대한 해결은 그리드 인프라에서 발생할 수 있는 본질적인 문제다. 이 문제를 해결하려면 다양한 종류의 하드웨어 및 소프트웨어 요소를 처리하는 능력을 갖춰야 하는데, 그렇게 하기 위해서는 내부에 깔려 있는 복잡한 구조를 사용자로부터 감추는 인터페이스를 잘 정의해야 한다. 이런 추상화 과정은 사용자 애플리케이션과 시스템을 잇는 미들웨어 단에서 수행해야 한다.

시스템 처리량은 데이터가 시뮬레이션 과정을 통해 생성되는 동안 시간에 따라 변하는 데이터를 시각적인 형태로 재현하는데 매우 중요한 요소다. 시스템 처리량을 최대화하는 목적은 사용 가능한 네트워크 용량을 가능하면 효율적으로 사용하는 데 있다. 시스템 처리량을 최대화하는데 중요한 요소로는 데이터 선정, 디스플레이 레이트, 데이터 생산기기와 렌더링 기기간의 거리 및 데이터 압축을 들 수 있다. 이 중 데이터 선정(data selection)은 목적 데이터 셋과 그 데이터 셋의 구조뿐만 아니라 사용자의 뷰포인트에 의해 결정된다. 디스플레이 레이트는 사용자 인터랙션의 종류 및 속도, 가시화 디바이스에서의 뷰에 의해 정해진다. 데이터 생산기기와 렌더링 기기간의 거리는 시뮬레이션 기계와 출력 디바이스간의 네트워크 연결에 의해 주어지는 것

으로 시간에 따라 계속 변할 수 있다. 데이터 압축은 네트워크를 통해 전송되는 절대적인 데이터량을 감소시키려는 노력으로, 출발지에는 데이터를 인코딩하는 루틴이, 그리고 목적지에는 데이터를 디코딩하는 루틴이 필요하다.

지연시간은 가시화와 시뮬레이션 간의 인터랙션과 피드백이 성립된 경우에 매우 중요하게 생각되는 요소다. 가시화에서의 지연시간은 대화식 명령과 시스템의 응답 사이에 소요되는 지연시간을 의미한다. 만약 가시화를 수행하는 기기에 전체 데이터가 저장돼 있는 상태라면 지연 시간은 렌더링 하드웨어의 성능에 좌우된다. 상호 작용과 데이터 탐색이 로컬 컴퓨터에서 이뤄지기 때문에 네트워크의 영향을 받지 않기 때문이다. 그러나 대부분의 시뮬레이션에서 데이터셋은 시간에 따라 변하기 마련이며, 따라서 지속적인 네트워크 전송이 필요하다.

3. 그리드 기반 가시화

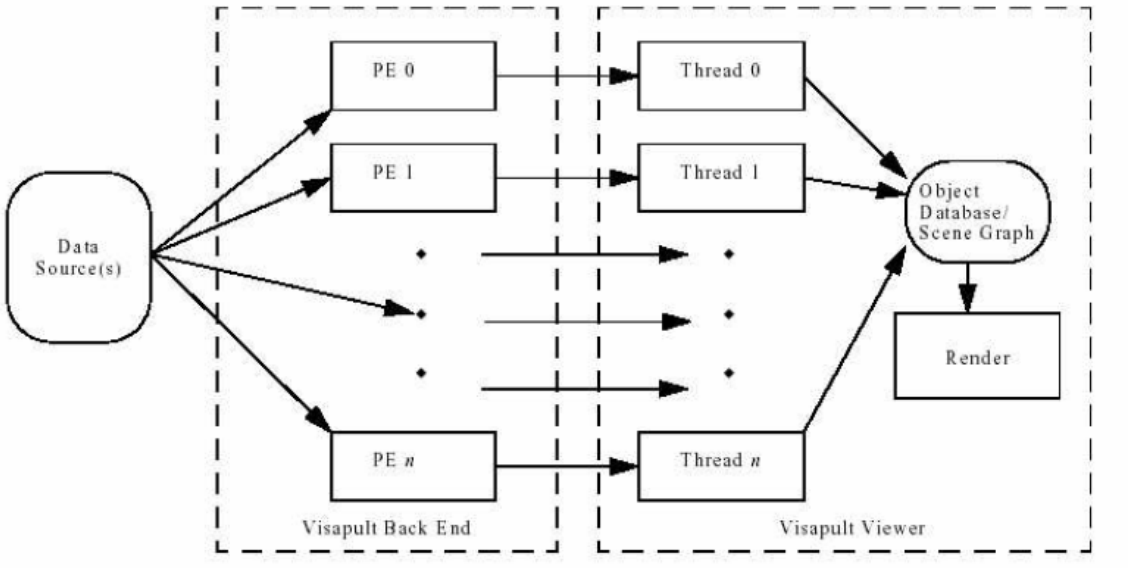
3장에서는 그리드 환경에서 가시화를 수행하는 여러 다양한 애플리케이션과 프로젝트에 대해 소개하기로 한다.

가. Visapult

Visapult는 볼륨 렌더링을 위한 병렬 이미지 기반 시스템으로, IBRAVR 가시화 알고리즘에 대한 고성능 가시화 기능을 구현한다. Visapult는 원시 데이터 소스, 뷰어, MPI-기반의 백-엔드, 이 3가지 요소로 구성되며, 다음과 같은 특징을 가진다.

- MPP 컴퓨팅 파워를 사용, 부분 렌더링을 수행한다.
- 가격이 저렴한 클라이언트 워크스테이션에서도 사용 가능한 하드웨어 기반 렌더링 기법을 이용한다.
- 데스크탑에서의 상호작용과 데이터를 네트워크를 통해 전송할 때 발생하는 지연 시간을 분리한다.

Visapult의 시스템 구조는 [그림 3-1]과 같다. 데이터 소스는 여러

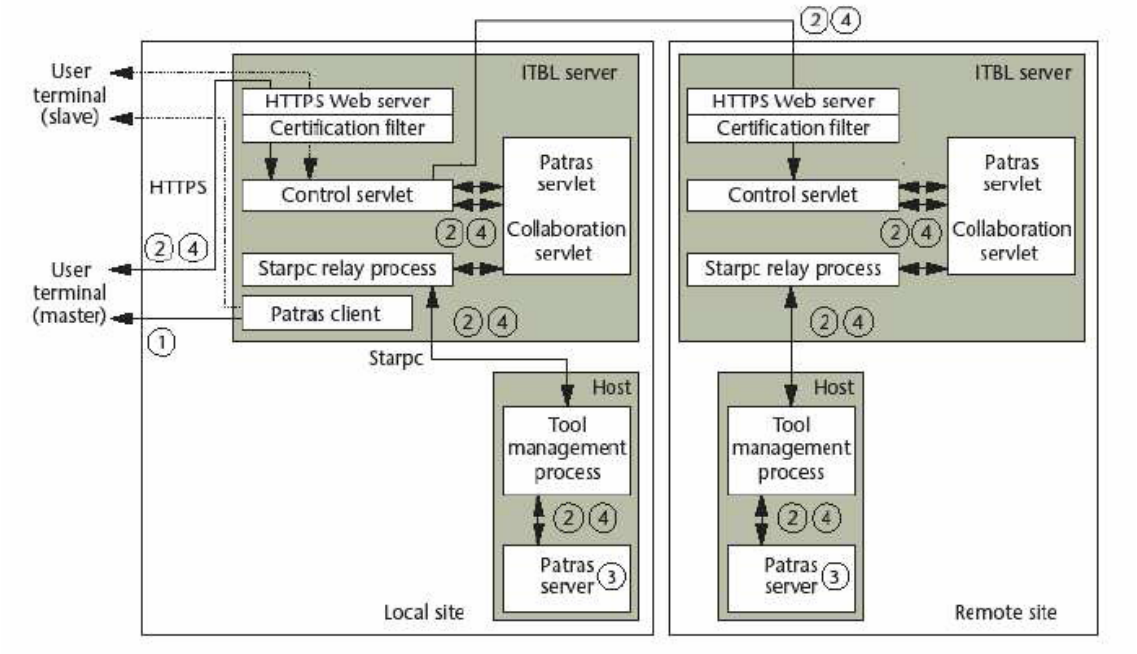


[그림 3-1] Visapult의 시스템 구조

개의 프로세스로 나뉘어 들어가서, Visapult 뷰어에서 씬 그래프(Scene Graph)의 형태로 합쳐져서 렌더링된다. 우선 데이터를 나누는 단계에서는 데이터는 각각의 프로세서에 의해 병렬로 읽혀진다. 그런 다음 각 프로세서는 자신이 가진 일부 데이터를 가지고 소프트웨어 볼륨 렌더링을 수행한다. 이렇게 렌더링된 결과 이미지는 네트워크를 통해 Visapult 뷰어로 전송된다. 결과 이미지는 뷰어 내에서 2D 텍스쳐의 형태로 씬 그래프에 추가되며, 사용자는 이 씬 그래프를 렌더링한 최종 결과 이미지를 얻게 된다.

나. Patras/ITBL

Patras는 2003년 Suzuki가 제안한 방식으로, 클라이언트-서버 모델에 기반을 둘 원격 가시화 시스템이다. 이 시스템은 시뮬레이션 프로그램의 실행과 가시화를 동시에 수행하는 기능을 제공하며, 초대용량 데이터셋의 가시화도 지원한다. Patras는 슈퍼컴퓨터(서버)의 각 프로세서에서 가시화 프로세스를 수행하고, 사용자 터미널(클라이언트)에 이미지를 디스플레이한다. 사용자는 시뮬레이션 파라미터를 조정할 수 있으며, 따라서 시뮬레이션 과정을 제어할 수 있다. 실제로 Patras 가시화 라이브러리를 사용하려면 사용자는 시뮬레이션 프로그램 내에 Patras 가시화 라이브러리에 대한 호출 루틴을 추가해야 한다.



[그림 3-2] Patras/ITBL 가시화 시스템의 내부 구조

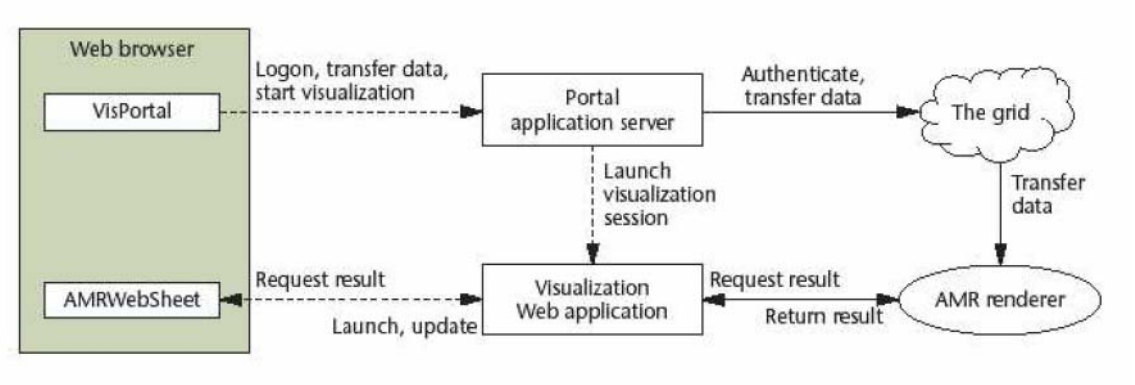
Patras/ITBL 가시화 시스템의 구조는 [그림 3-2]에서 볼 수 있다.

Patras 라이브러리는 데이터-병렬 구조에서 실행되게 설계돼 있으며, domain decomposition 방식에 기반을 두고 있다. 각각의 프로세서는 최종 이미지의 일부분을 Z-버퍼값과 함께 생성하며, 따로 지정된 프로세서가 이렇게 생성된 부분 이미지를 통합, 최종 이미지를 생성한다. Patras는 매 시뮬레이션 타임 스텝마다 Z-버퍼 값을 사용해서 부분 이미지를 하나의 이미지로 통합하며, MPI를 이용, 이미지 데이터를 최종적으로 통합한다.

다. VisPortal/AMRWebSheet

그리드를 이용해서 가시화를 제어하는 데는 2가지 방식이 있는데, 그 중 하나는 웹을 기반으로 하는 가시화 시스템을 구축하는 것이고 다른 하나는 그리드같은 환경(ex. 포털)에서 분산 가시화 시스템을 구축하는 것이다. VisPortal/AMRWebSheet은 이중 그리드 환경을 사용해서 분산 가시화 시스템을 구축한 것이다.

그리드 리소스를 사용자가 효율적으로 사용하게 하려면 중앙 접속 지점(central access point)이 있어서 리소스를 관리하고 데이터 탐색을



[그림 3-3] VisPortal/AMRWebSheet의 시스템 구조

위한 시각적인 인터페이스를 제공하거나 향후 탐색을 위한 경로 저장 기능을 제공해야 한다. VisPortal은 University of California, Davis와 Lawrence Berkeley National Laboratory(LBNL)가 2003년 공동으로 개발한 중앙 접속 지점이다. 이렇게 중앙집중화된 시스템은 그리드 가시화 시스템에 대한 포털의 역할을 수행하게 된다.

이 시스템의 3가지 주요 요소는 다음과 같다.

- 그리드 기반 가시화 서비스에 대한 웹-기반 사용자 인터페이스
- 가시화 결과 탐색에 사용되는 가시화 웹 애플리케이션
- 그리드 리소스 사용에 대한 인증을 관리, 조정하는 포털 애플리케이션 서버

이 시스템의 구조는 [그림 3-3]과 같다. 그림으로 알 수 있듯이, 이 시스템은 AMR(Adaptive Mesh Refinement) 방식으로 표현된 볼륨 데이터셋을 렌더링하기 위한 볼륨 렌더러를 둘러싸는 식으로 구성돼 있다.

AMRWebSheet 인터페이스는 Python으로 구현돼 있으며, 가시화 세션과 이 세션에 대한 인터페이스의 상호작용을 관리하는 일련의 오브젝트로 구성돼 있다. 이 시스템은 가시화 세션을 관리하는 웹 애플리케이션 서브렛을 Webware 애플리케이션 환경을 사용해서 Python으로 구현했다. 또, 웹 페이지 서비스에는 리눅스 환경에서 돌아가는 Apache를 사용했다. 일단의 서브렛은 프로세스를 생성하고 세션을 저장하는데 사용된다.

클라이언트가 연결되면 애플리케이션은 임시 쿠키에 의해 식별된 새로운 세션과 서브렛 오브젝트를 생성한다. 사용자가 HTML 인터페이스와 상호작용을 할 때마다 애플리케이션은 AMRWebSheet HTTP 요청을 인터페이스 서브렛에 전달하며, 이 요청은 순차적으로 가시화 세션 상태를 수정하게 된다. 클라이언트가 업데이트를 필요로 하면 서버는 refresh를 수행함으로써 새로운 정보를 디스플레이한다. 세션이 종료되거나 타임 오버에 도달하면 애플리케이션은 세션 결과를 XML 문서로 인코딩해서 애플리케이션 서버에 저장하게 된다.

애플리케이션 서버는 AMR 볼륨 렌더러와 AMRWebSheet 사이의 모든 커뮤니케이션을 처리한다. AMRWebSheet이 결과를 요청하면 웹 애플리케이션에 존재하는 visualization transformation 클래스 인스턴스가 볼륨-렌더링 서비스로부터 해당 결과를 요청하게 된다. 볼륨 렌더러가 해당 결과를 리턴하면 웹 애플리케이션은 가시화 세션 결과와 함께 해당 결과를 저장한다. 그런 다음, 애플리케이션은 refresh 명령을 내림으로써 클라이언트의 웹-브라우저에 결과를 디스플레이한다.

라. RealityGrid

RealityGrid는 영국 정부에서 e-Science 사업을 추진하면서 엮어진 대학과 연구기관간의 컨소시엄이다. 이 컨소시엄의 목적은 복잡한 나노 단위의 집적 구조체에 대한 사실적인 모델링 및 시뮬레이션과 신소재 개발을 그리드 환경에서 수행코자 하는 것으로, 그리드 기반 과학, 의학, 상업적인 행위를 위한 일반 기술을 제공하는 데 있다. RealityGrid에서는 그리드를 통해 Virtual Reality 센터의 개념을 확장할 것을 제안하고 있으며, 그리드 기술을 사용, 고성능 컴퓨팅 센터에 있는 계산 자원이나 실험 장비와 연결함으로써 처리율이 높은 실험 환경과 가시화 환경을 엮으려는 시도를 계속하고 있다.

이 시스템은 OGSI(Open Grid Services Infrastructure)에 기반을 두고 있으며, 미들웨어 그리드 인프라에 의해 연결된 기존의 애플리케이션을 사용한다.

마. GEMSS

GEMSS는 Grid-Enabled Medical Simulation Service의 약자로, 유럽연합(EU)에서 수행하고 있는 연구 프로젝트다. 이 프로젝트는 분산된 온-디맨드 슈퍼컴퓨팅을 위한 보안 서비스 중심의 인프라에서 의료 그리드 서비스에 대한 프로토타입을 개발하고자 하는 것이다. 이 프로젝트는 그리드를 기반으로 하지 않는 애플리케이션과 그리드 서비스간에 인터페이스를 담당하는 미들웨어 개발을 목적으로 한다. 이 프로젝트에서는 새로운 애플리케이션 개발이 필요치 않으며, 단지 각 애플리케이션간에 인터페이스를 담당하는 중간 단계의 미들웨어 애플리케이션 개발만 필요할 뿐이다. 그러나 개발하고자 하는 애플리케이션의 개수에 따라서는 미들웨어 애플리케이션이 매우 복잡해질 수 있다는 단점이 있으며, 가시화에 대한 유연성이 떨어진다는 단점도 존재한다.

바. UniGrids

UniGrids는 OGSA(Open Grid Service Architecture)와 호환되는 그리드 서비스 인프라를 구축하는 프로젝트로, 독일의 UNICORE 및 UNICORE Plus 프로젝트에서 개발된 UNICORE 그리드 소프트웨어에 기반을 둔다.

이 프로젝트에서 구현한 패키지는 VISIT 툴킷에 대한 통합 패키지로 구성돼 있다. (VISIT은 실시간 시뮬레이션의 개발을 지원하고 시뮬레이션과 가시화 및 데이터 교환 과정의 연결에 필요한 함수를 제공하는 라이브러리다.)

사. TeraGrid

TeraGrid는 열린 과학 연구를 위해 규모가 크고 빠른 분산 인프라를 구축하고자 하는 프로젝트로 5개 사이트에 흩어져 있는 20테라 플롭스의 컴퓨팅 파워와 거의 1 페타바이트의 데이터를 관리, 저장할 수 있는 장비, 고해상도 가시화 환경 및 그리드 컴퓨팅에 필요한 툴킷을

50기가비트/초의 용량을 제공하는 네트워크를 사용해서 통합하려는 목적을 가진 프로젝트다.

이 프로젝트는 하드웨어와 미들웨어의 통합에 관심을 두고 있으며, 소프트웨어 레이어에서는 기존의 툴를 갖고 가시화 과정을 수행하는 데 초점을 맞추고 있다.

아. GVK

GVK는 Grid Visualization Kernel의 약자로, 시뮬레이션 서버와 가시화 클라이언트에 대한 포털을 제공한다. 가시화 파이프라인의 실제 프로세스는 그리드 리소스상에서 투명하게 수행된다.

GVK는 그리드 환경에서 실행 가능한 가시화 소스와 출력 디바이스 간의 연결점을 제공하며, 기존 의 가시화 프로세스를 사용함으로써 가시화 과정을 단순화하는 역할을 수행한다. GVK의 목적은 그리드 애플리케이션에서의 가시화를 위한 미들웨어 레이어 확장판을 제공하는 데 있다.

1) GVK의 특성

가) 투명성

GVK 미들웨어 인터페이스는 가장 필요한 기능만 제공한다. 따라서 하위 레벨의 서비스는 가능하면 자동으로 실행된다.

나) 사용상의 편이

GVK에서는 OpenDX, AVS 및 VTK같은 기존의 가시화 패키지를 사용한다. 따라서 기존의 가시화 함수를 그대로 재사용할 수 있을 뿐만 아니라 기존 패키지를 사용하던 사용자들은 별도로 가시화 기능을 익힐 필요 없이 바로 자신의 애플리케이션을 그리드 환경으로 확장할 수 있다.

다) 다양한 연결성

GVK는 다중 데이터 소스와 다중 가시화 목적지간의 연결을 제공한다. 소스 애플리케이션은 사용 가능한, 그리고 필요한 컴퓨팅 리소스에 따라 그리드 환경 내에서 이곳저곳으로 옮겨다닐 수 있으며, 가시화 클라이언트는 사용자가 사용할 수 있는, 네트워크 연결이 가능한 디바이스라면 어디에서건 호출할 수 있다. 또, 암호화나 사용자 인증 같은 기능은 기존의 미들웨어 기능을 이용하면 충분하다.

라) 최적화 기법의 사용

GVK는 시스템 처리량이나 지연시간 같은 네트워크 문제를 해결하기 위해 최적화 기법을 적용한다. 네트워크 문제 해결을 위한 기법에는 표준 압축 기법도 포함되며, 가시화 데이터와 그래픽 표현의 저장 공간을 최소화하기 위한 기법도 포함된다. 이런 기법으로는 LOD(Levels-of-detail), occlusion-culling, image-warping 등을 들 수 있다.

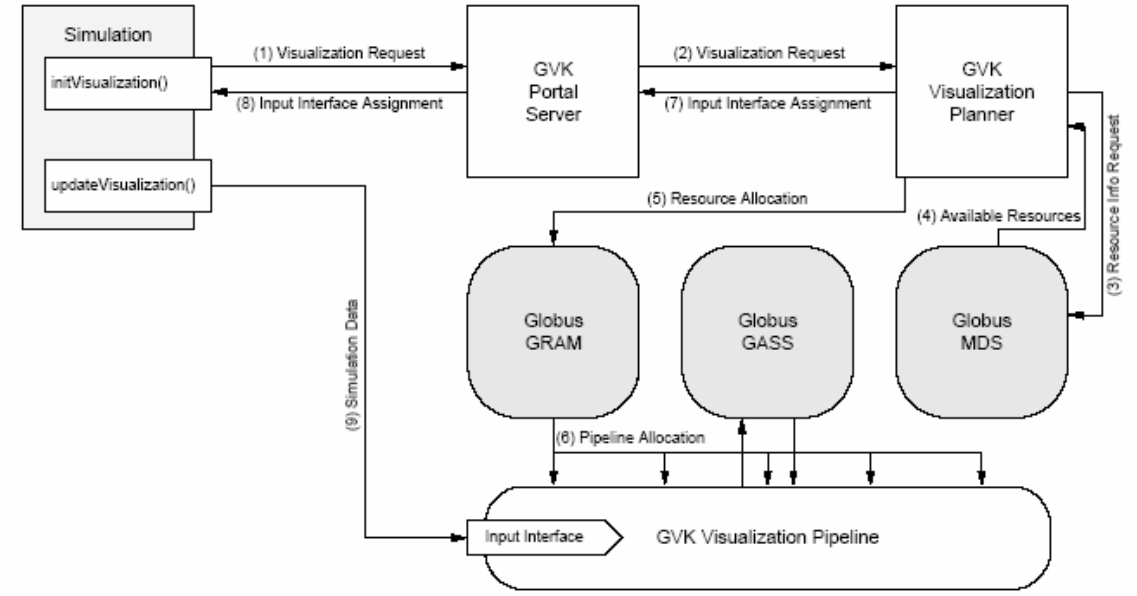
2) GVK 가시화 파이프라인

GVK는 시뮬레이션과 가시화간의 인터페이스를 제공하면서 가시화 파이프라인을 구현한다. GVK에 대한 인터페이스로는 OpenDX(Open Data eXplorer)를 사용하는데, 사용자는 시뮬레이션 인터페이스로부터 디스플레이 가능한 이미지로의 데이터-흐름도를 작성함으로써 가시화 파이프라인을 명시할 수 있다. 이 때, 데이터 흐름도의 몇몇 OpenDX 모듈은 GVK 모듈로 대체되기도 한다. 가시화 파이프라인은 어떤 시점에서건 나뉘어 수행될 수 있으며, 처리 모듈은 그리드 환경에서 분산 처리된다.

GVK의 가시화 파이프라인은 [그림 3-4]와 같다. 다음은 GVK 가시화 파이프라인의 각 부분에 대한 상세한 설명이다.

(1) 초기화 단계에서 시뮬레이션 서버는 GVK 포털 서버로 가시화 요청 메시지를 전송한다. 이 요청 메시지의 내용은 가시화 클라이언트에 대한 최소 요구사항이다.

(2) 포털 서버는 시뮬레이션 서버를 확인하고 가시화 요청 메시지를



[그림 3-4] 가시화 파이프라인

GVK 가시화 플래너로 전송한다.

(3) 가시화 플래너는 리소스 정보 요청 메시지를 글로버스 MDS로 전송한다. 이는 가시화 파이프라인이 사용할 수 있는 리소스를 파악하기 위한 것이다.

(4) 글로버스 MDS는 사용 가능한 리소스에 대한 정보를 GVK 가시화 플래너에 리턴한다.

(5) 가시화 플래너는 GRAM(Globus Resource Allocation Manager)에게 리소스 할당(resource allocation) 요청을 전송한다.

(6) 글로버스 GRAM은 가시화 파이프라인을 할당하고 가시화 플래너에게서 요청받은대로 모든 변환 모듈(transformation module)을 초기화한다.

(7) 가시화 플래너는 GVK 가시화 파이프라인에서 새로 초기화된 입력 인터페이스에 대해 입력 인터페이스 할당(input inteface assignmen) 내역을 GVK 포털 서버에 리턴한다.

(8) 포털 서버는 향후 요청에 대해 새로운 가시화 서비스를 저장하고 입력 인터페이스에 대한 할당 내역을 시뮬레이션 소스에 전송한다.

(9) 시뮬레이션은 데이터 처리를 계속 실행한 뒤, 시뮬레이션 데이터

를 GVK 가시화 파이프라인의 입력 인터페이스로 전송함으로써 업데이트를 수행한다.

가시화 클라이언트에 대한 등록은 가시화 요청 메시지와 함께 초기화된다. 이 때, 가시화 클라이언트의 최대 성능이 주어진다. 이 정보를 가지고 GVK 포털 서버는 시뮬레이션 서버와 가시화 클라이언트를 대응, 연결할 수 있다. GVK는 이렇게 등록된 가시화 클라이언트를 가지고 위의 과정을 거쳐 파이프라인을 연결하고 그리드 환경에서 가시화 작업을 수행한다.

3) GVK의 최적화 기법

GVK에서는 네트워크의 처리 속도와 지연 시간의 향상을 위해 몇가지 최적화 기법을 사용한다. 이중에는 표준 압축 기법을 사용하는 방식도 있으며, 컴퓨터 그래픽스의 하드웨어 및 소프트웨어 가속 기법을 응용한 방식도 있다.

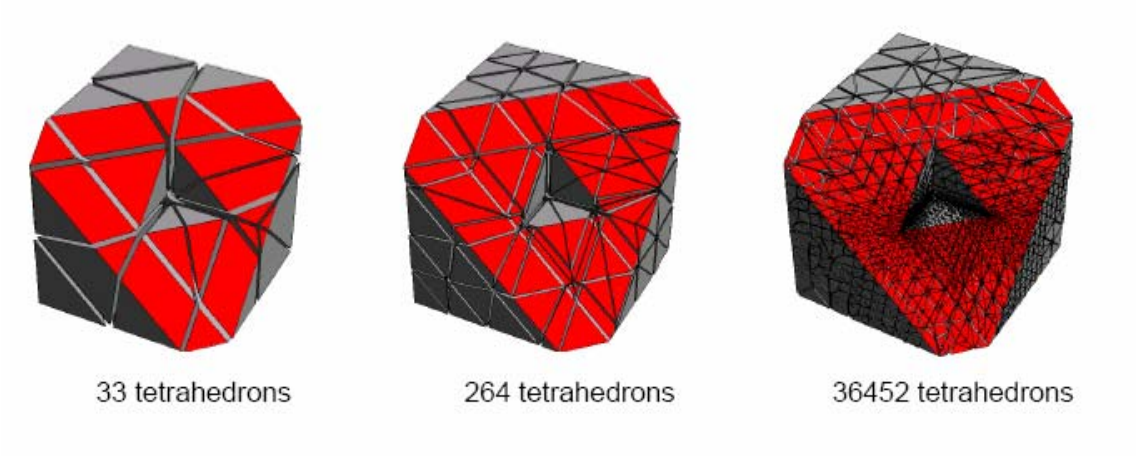
가) Level-of-detail 필터링

Level-of-detail은 서로 다른 폴리곤 단계의 계층적 구조를 이용해서 렌더링을 수행하는 기법으로, 사용자가 처음에 성긴 폴리곤으로 데이터셋의 대략적인 모습을 분석하는 동안 상세 내용을 담은 폴리곤 데이터를 네트워크를 통해 전송함으로써 사용자의 렌더링 요청에 대한 응답 시간을 최대한 줄이는 기법이다.

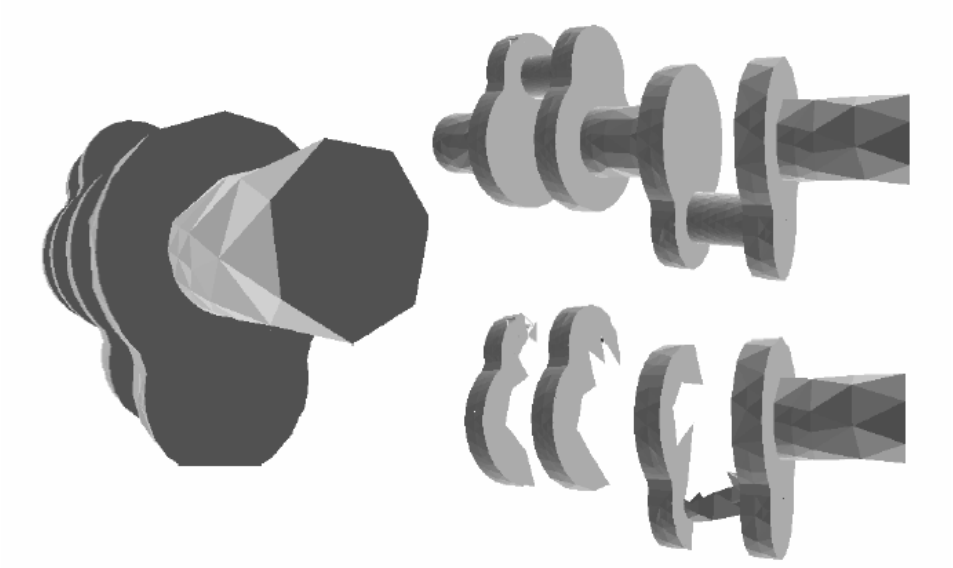
Level-of-detail에 대한 예는 [그림 3-5]에서 볼 수 있다. [그림 3-5]에서처럼 처음에는 33개의 사면체로 구성된 폴리곤 데이터를 전송하고 사용자가 그 결과를 관찰하는 동안, 네트워크로 지속적으로 세부 폴리곤을 전송하는 것이다. 결국 사용자의 렌더링 요청에 대한 초기 응답시간은 크게 단축될 수 있다.

나) Occlusion-culling

GVK에서 이용하는 Occlusion-culling의 개념은 보이는 오브젝트와



[그림 3-5] Level-of-detail



[그림 3-6] 크랭크축에 대한 occlusion-culling

보이지 않는 오브젝트를 구분함으로써 전송해야 하는 폴리곤의 수를 줄이겠다는 것이다. 즉, 폴리곤을 사용자에게 바로 보이는 폴리곤과 그렇지 않은 폴리곤으로 구분을 하는데, 이 때 사용자에게 보이는 폴리곤은 가시화 클라이언트로 즉시 전송돼서 첫 이미지를 렌더링하게 된다. 사용자에게 보이는 폴리곤만 렌더링하므로 Level-of-detail과 마찬가지로 이 때의 사용자 응답 시간도 매우 빨라진다. 사용자가 이 폴리곤을 관찰하는 동안, 나머지 폴리곤도 서버로부터 전송이 됨으로써 향후 사용자 뷰포인트 변화에 대비한다.

[그림 3-6]은 크랭크축에 대한 occlusion-culling을 나타낸다. 전체 크랭크축 모델은 66,223개의 사면체로 구성돼 있다. 그러나 occlusio

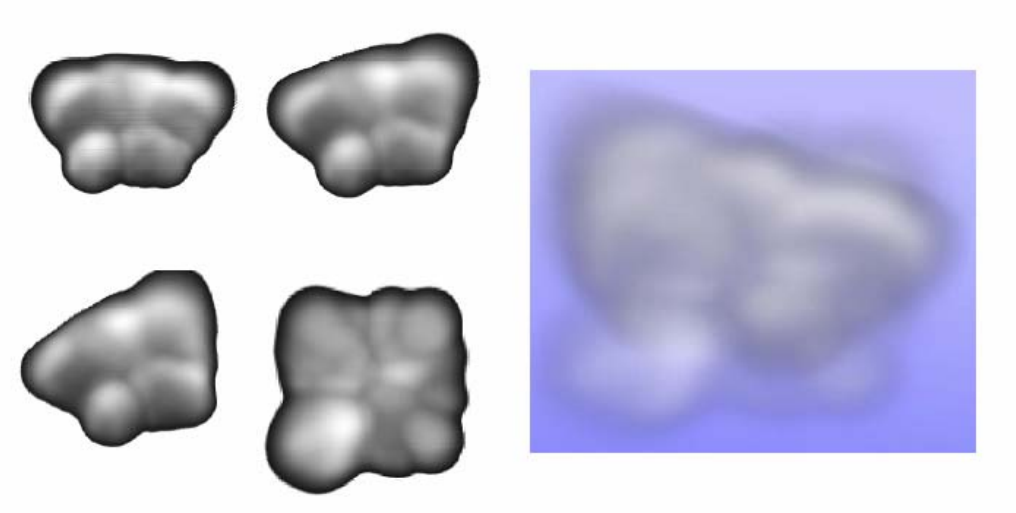
n-culling을 수행하면 초기에는 18,511개의 사면체만 전송되면 된다. [그림 3-6]에서 왼쪽에 있는 그림은 크랭크축에 대한 렌더링 결과를 나타내고 오른쪽 아래에 있는 그림은 이 이미지를 그리기 위해 전송된 폴리곤을 나타낸다.

다) 레퍼런스 렌더링(reference rendering)과 IBR(Image-Based Rendering)

레퍼런스 렌더링은 시뮬레이션 서버에서 수행되며, 일련의 '레퍼런스 이미지'라는 것을 생성해 낸다. 레퍼런스 이미지는 가시화할 오브젝트를 여러 뷰포인트에서 렌더링한 이미지를 나타내기도 하고, 이런 이미지를 조합해서 만들어낸 일련의 이미지를 가리키기도 한다. 이런 레퍼런스 이미지를 서버에서 생성함으로써 짧은 시간 내에 화질 좋은 이미지를 생성할 수 있는 것이다.

이렇게 생성된 레퍼런스 이미지는 가시화 클라이언트로 전송되며, 가시화 클라이언트는 여러 다양한 이미지-기반 렌더링(IBR, Image-Based Rendering) 기법을 사용함으로써 최종 이미지를 생성하게 된다.

레퍼런스 렌더링과 IBR을 이용한 사례는 [그림 3-7]에서 찾아볼 수 있다. [그림 3-7]에서 4개의 레퍼런스 이미지는 그리드 상에서 생성되며, 최종 이미지는 사용자의 디바이스로 출력된다. 4개의 레퍼런스 이미지는 서로 다른 카메라 위치상에서 렌더링되며, 최종 이미지는



[그림 3-7] 레퍼런스 렌더링과 IBR 기법으로 그린 구름 이미지

이 4개의 입력 이미지를 뷰포인트에 대한 알파 블렌딩 기법을 사용해서 합침으로써 생성된다. 이런 렌더링 기법은 구름, 안개나 연기같이 뚜렷한 모양이 없이 밀도로만 나타낼 수 있는 현상을 나타내는데 특히 적합하다.

이런 최적화 기법은 언제나 렌더링하고자 하는 컨텐트에 따라 적용 방법이 달라지며, 사용자는 원하는 가시화 작업과 데이터 구조를 위해 파이프라인 구성을 선택해야 한다.

4. 결론

컴퓨팅 기술은 나날이 발전하고 있다. 그러나 현재의 계산과학에 요구되는 컴퓨팅 파워는 기술의 발전 속도보다 한층 더 빠른 속도로 늘어나고 있으며, 이에 의해 계산 과학자들은 보다 많은 계산 리소스를 확보하기 위해 노력하고 있다. 이러한 노력의 일환으로 발전한 기술 중 하나가 그리드 컴퓨팅 기술이다. 이 기술은 특히 규모가 크거나 처리 시간이 긴 작업을 해결하는데 유용한 것으로, 다양한 분야에 응용할 수 있는데, 이런 응용 분야 중에는 계산 데이터를 가시화하는 분야도 해당된다.

전통적으로 가시화 작업은 많은 리소스를 필요로 하는, 복잡하고 처리 시간이 긴 작업에 속한다. 따라서 컴퓨팅 파워가 큰 원격지 서버에서 렌더링을 수행하고 가시화 작업의 결과물만 전송하는 원격 가시화 기법에 대한 연구는 기존에도 많이 이뤄져 왔다. 혹은 대용량 데이터의 일부를 서버에서 클라이언트 워크스테이션으로 전송한 후, 이 일부데이터만 클라이언트에서 가시화하는 방식도 많이 사용되는 기법 중 하나다. 그러나 최근에는 그리드 미들웨어를 이용, 데이터 생성, 저장 및 가시화에 이르는 일련의 단계를 모두 그리드 환경에서 수행하는 기술에 대한 연구가 활발하게 이뤄지고 있으며, 여기에 대한 다양한 사례 연구를 찾아볼 수 있다. 향후에는 이런 연구 사례에 힘입어 다양한 분야에서 사용할 수 있는 본격적인 애플리케이션이 개발돼서 사용되게 될 것이다.

5. 참고 문헌

[1] Dieter Kranzlmuller, Paul Heinzlreiter, Jens Volkert, "Grid-Enabled Visualization with GVK", Proceedings of 1st European Access Grids Conference, Santiago de Compostela, Spain, February 2003

[2] Dieter Kranzlmuller, G.Kurka, P.Heinzlreiter, J.Volkert, "Grid Middleware Extension for Scientific Visualization", Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Vol.2, 2002

[3] Dieter Kranzlmuller, Gerhard Kurka, Paul Heinzlreiter, Jens Volkert, "Optimizations in the Grid Visualization Kernel", Proceedings of the 16th International Parallel and Distributed Processing Symposium, 2002.

[4] Pascal Kleijer, Eiichi Nakano, Toshifumi Takei, Hiroshi Takahara, Arihiro Yoshida, "API for Grid Based Visualization Systems", GGF 12, September 20, 2004

[5] E.W.Bethel, J.Shalf, "Grid-distributed visualization using connectionless protocols", IEEE computer Graphics Application, 2003.

[6] Y.Suzuki, K.Sai, N.Matsumoto, O.Hazama, "Visualization systems on the information-technology-based laboratory", IEEE Computer Graphics Application, 2003

[7] T.J.Jankun-Kelly, O.Kreylos, K.Ma, B.Hamann, K.I.Joy, J.Shalf, E.W.Bethel, "Deploying web-based visual exploration tools on the grid", IEEE COmputer Graphics Application, 2003

[8] T.J.Jankun-Kelly, K.Ma, "Visualization exploration and encapsulation via a spreadsheet-like interface", IEEE Transactions on Visualization and Computer Graphics, 2001.