ISBN 978-89-6211-395-2

GLOVE 렌더링 엔진 설계 및 구현

(Design and Implementation of GLOVE Rendering Engine)

이 중 연 (jylee@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

컴퓨터 시뮬레이션은 물리, 화학, 생물, 기계공학 등 자연과학·공학뿐만 아니라 인문·사회과학 분야에서도 많이 적용되고 있는 연구방법으로 계산(computation)과 가시화(visualization)라는 두 가지 기술에 의해 발전되고 있다. 고성능 컴퓨터(HPC: High Performance Computer)를 포함한 계산 기술의 빠른 발전은 기존에 적용하기 어려웠던 문제를 빠른 시간 내에 해결하는 것을 가능하게 하였으며 새로운 현상의 발견도 가능하게 함으로써 컴퓨터를 이용한 시뮬레이션이 가속화되고 있다. 시뮬레이션 결과는 연구자가 내용을 그대로 보면서 해석하는 것이 매우 어렵기 때문에 연구자가 쉽게 알아볼 수 있도록 데이터에 그래픽처리를 하여 보여주는 기술이 과학적 가시화 기술이다. 그러나 기하급수적으로 발전하는 계산 기술에 비하여 가시화 기술의 발전은 이를 따라가지 못하고 있으며 시뮬레이션 연구자가 워크스테이션이나 데스크탑을 이용한 기존의 방식으로는 대용량의 복잡한 결과 데이터를 이해하기가 어려워지고 있다. 이에 사용자들은 평면 모니터 상에서 수행하는 단순한 인터페이스보다 더 발전된 방식을 요구하고 있으며 이러한 사용자 요구에 대한 해결방법으로 가상현실(virtual reality) 기술이 적용되고 있다[1, 2]. 한편, 시뮬레이션의 결과 데이터를 가시화하는 방법은 매우 다양하며, 때로는 복잡한 가시화 알고리즘을 필요로 하기도 한다. 이런 복잡한 알고리즘을 지원하는 툴은 매우 다양한데, 이중 특히 VTK[3,4]는 과학 데이터 가시화에 많이 사용되는 공개 소프트웨어 라이브러리로, 매우 다양한 자료구조와 가시화 관련 알고리즘 기능을 지원하며, 실제적으로 CFD, 의료, 화학, 구조역학 등 다양한 분야에서 유용하게 사용되고 있다. 하지만 VTK는 고해상도 디스플레이 장치 출력이나 VR에 관련된 기능은 전혀 지원하지 않는다. VTK가 이런 환경을 지원하게 한다면 VTK의 다양한 알고리즘을 이용해서 복잡한 대용량 데이터를 고해상도 디스플레이에서 가시화하면서 적절한 가시화 인터페이스를 제공하는 것도 가능할 것이다. GLOVE[5,6]는 이런 점에 착안, VTK와 VR과의 접목을 통해 로터 동역학 분야의 대용량 데이터를 고성능 컴퓨팅 환경에서 렌더링해서 고해상도의 이미지를 생성하고, VR 환경을 기반으로 하는 통합 인터페이스 환경을 통해 사용자가 가시화 과정, 더 나아가서는 시뮬레이션 과정을 직접 제어하는 프레임워크를 구축하는 프로젝트이다. 본 기술문서에서는 VTK를 기반으로 설계, 구현된 GLOVE의 렌더링 엔진에 대해 살펴본다.

2. 기본 렌더러

GLOVE의 렌더링 기능은 기본적으로 VTK의 렌더링 기능을 기반으로 개발되었다. VTK는 다양한 분야의 가시화를 지원하기 위해 많은 가시화 필터를 제공하는데, GLOVE는 기본적으로 로터 동역학 분야에서 가장 많이 사용되는 iso-surface, 스트림라인 등의 필터를 사용한다. GLOVE에서 지원해야하는 렌더링 기능은 다음과 같다.

1. 스칼라 값 디스플레이 : 벽면 데이터 표면에 스칼라값을 디스플레이
2. Iso-surface : 필드 데이터에 특정 iso-value로 iso-surface 생성
3. 스트림라인 : 필드 데이터의 속도 벡터(velocity vector)를 이용해서 스트림라인, 스트림튜브 등을 생성
4. 외곽선 : 벽면 또는 필드 데이터의 외곽선 추출
5. 메쉬 : 벽면 또는 필드 데이터의 메쉬를 디스플레이
6. Glyph : 필드 데이터의 각 격자에 할당된 벡터를 화살표나 삼각뿔 등을 이용해서 디스플레이
7. Probe by plane (cutting plane): plane을 이용해서 데이터의 단면을 잘라서 디스플레이
8. 패스라인(pathline) : 시변환 데이터에 대해 시간축으로 데이터를 변경하면서 스트림라인을 생성

여기에 나열한 렌더링 기능들은 모두 VTK에서 지원가능하며 로터 동역학 데이터를 가시화하는데 필수적으로 필요한 기능들이다. 그러나 2009년 11월 현재 1~5번까지의 기능만이 구현이 완료되었고 6~8번 기능들은 향후 계획으로 남겨놓았다. 이 장에서는 위에서 나열한 렌더링 기능들을 VTK를 이용해서 구현하는 방법과 이를 위한 렌더러 클래스 설계 및 GIVI와의 인터페이스 등에 대해 설명한다.

가. VTK 구현

1) 스칼라 디스플레이 및 메쉬 추출

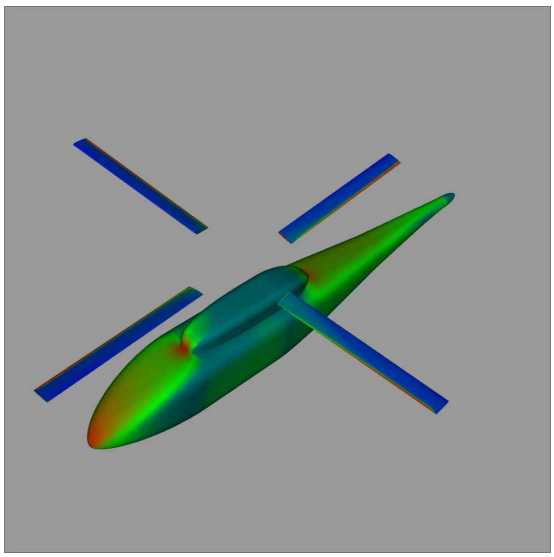
스칼라 디스플레이는 벽면데이터의 표면을 지정한 스칼라 변수들과 transfer function을 이용해서 색칠하는 것을 말한다. 여기서 중요한 것은 벽면데이터에서 표면을 뽑아내는 것인데, 벽면 데이터가 2차원으로 구성되어 있는 경우에는 큰 어려움이 없다. 하지만 3차원으로 구성되어 있는 경우에는 여기서 무엇이 표면인지를 찾아야 하는 문제가 발생한다. 로터 동역학를 비롯한 CFD나 구조해석 분야의 데이터는 대부분 구조화된 격자구조(structured grid) 또는 비구조화된 격자구조(unstructured grid)로 이루어져 있다. 이때, 구조화된 격자구조로 되어 있는 경우에는 격자구조의 위상이 규칙적이기 때문에 표면을 구하기가 비교적 쉽다. 그러나 비구조화된 격자일 경우에는 데이터를 생성한 사람이 어디서부터 어디까지가 표면인지를 일일이 지정해 주어야 한다. 비구조화된 격자의 경우에는 규칙성이 전혀 없기 때문에 본 기술문서에서 다루기에는 무리가 있고, 로터 동역학 분야의 경우 많은 경우에 데이터가 구조화된 격자로 되어 있기 때문에 본 기술문서에서는 데이터가 구조화된 격자구조라고 가정하기로 한다.

3차원으로 구성된 구조화된 격자구조의 경우 각 격자를 (i, j, k)로 인덱싱이 가능하다. 이때, 격자의 표면은 대부분 i, j, k 중 하나가 0인 경우인데, 이중 k가 0인 경우가 많다. 그러나 이는 이런 경우가 많다는 것이지 모든 구조화된 격자구조의 데이터가 다 그런 것은 아니다. 정확한 표면 정보는 데이터 생성자에게 문의하는 것이 좋다.

이렇게 생성된 벽면의 표면 데이터는 그 자체로는 여전히 로우(raw) 데이터이기 때문에 그래픽으로 생성하는 것이 어렵다. 이를 해결하는 것은 VTK에서 제공하는 vtkGeometryFilter를 사용해서 메쉬를 추출하면 된다. vtkGeometryFilter는 다양한 격자구조의 데이터를 vtkPolyData 형식의 다각형 데이터로 변경하는 필터이다. 이 필터를 이용해서 vtkPolyData를 생성하면 vtkPolyDataMapper를 이용해서 쉽게 그래픽 표현으로 바꾸는 것이 가능하다. 이때, 만약 표면을 추출하고 싶다면 vtkActor의 vtkProperty에서 SetRepresentationToSurface() 함수로 표면이 그려지도록 해야 하고 메쉬 구조를 그대로 추출하고 싶다면 SetRepresentationToWireframe()을 사용해야 한다. 또, 만약 벽면 데이터에 단 하나의 스칼라 변수만 저장되어 있다면 문제가 없지만 여러 종류의 스칼라 변수가 저장되어 있는 경우에는 어떤 스칼라를 이용해서 디스플레이할 것인지를 결정해야 한다. 이는 v

tkDataSet의 속성 데이터를 받은 뒤 SetActiveScalar() 함수를 호출하면 된다. 마지막으로 각 스칼라 값들을 어떤 색깔로 색칠할 것인지를 결정해야 하는데 이는 vtkColorTransferFunction 으로 가능하다. vtkColorTransferFunction 데이터를 생성한 뒤 이를 vtkMapper에 등록하면 되는데, 자세한 내용은 5장에서 다룬다.

한편, 벽면 데이터는 많은 경우에 다중 블록(multi-block) 구조를 가진다. VTK의 대부분의 필터는 여러 개의 블록을 동시에 처리하지 못하고 단일 블록 단위로만 처리하는 것이 가능하기 때문에 다중 블록 데이터는 여러 번에 걸쳐서 처리해야 한다. vtkGeometryFilter 역시도 단일 블록의 데이터만 처리할 수 있기 때문에 데이터가 다중 블록인 경우 블록의 개수만큼 여러 번에 걸쳐서 필터를 실행해야 하는데, 이럴 경우 여러 개의 vtkPolyData가 생성되게 된다. 이렇게 생성된 여러 개의 vtkPolyData는 하나의 vtkPolyData로 합쳐져야 하는데 이는 vtkAppendPolyData를 이용하면 가능하다.



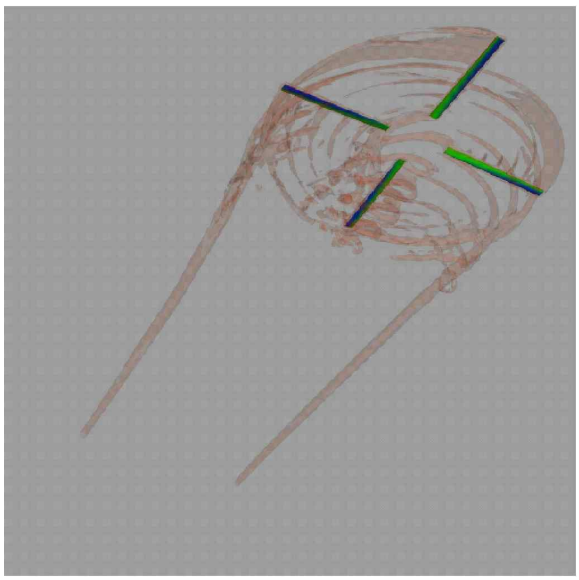
[그림 2-1] 동체 데이터 스칼라 디스플레이

2) Iso-Surface

Iso-surface란 특정 iso-value 값을 가지는 영역을 합쳐서 만든 표면(surface)를 말한다. 일반적으로 2차원의 공간에서는 같은 값을 가지는 영역을 모으면 선(line)이 되고 3차원의 공간에서는 같은 값을 가지는 영역을 모으면 면(surface)이 된다. VTK에서는 Iso-surface를 생성하기 위한 필터를 여러 개 제공하고 있다. 여러 종류의 Iso-surface 필터들 중, GLOVE에서는 vtkContourFilter를 사용

한다. vtkContourFilter는 다양한 격자 구조 및 차원의 데이터를 지원할 뿐만 아니라 스칼라 트리(scalar tree)를 이용한 빠른 Iso-surface의 생성도 가능하다. 이 필터 역시 최종적으로 vtkPolyData 형태의 다각형 데이터를 생성해 내므로 vtkPolyDataMapper를 통해 VTK에서 그래픽화가 가능하다. 이때, 많은 경우 iso-surface의 색깔 및 투명도를 조절하는데, 이는 vtkColorTransferFunction을 생성한 뒤 vtkMapper에 등록하거나 vtkActor에서 vtkProperty를 받은 후 SetColor() 및 SetOpacity() 함수를 이용해서 변경하는 것이 가능하다.

한편, vtkContourFilter 역시 다중 블록 데이터는 처리하지 못한다. 따라서 스칼라 디스플레이의 경우와 마찬가지로 여러 블록을 개별적으로 처리해야 하고 이렇게 생성된 vtkPolyData는 vtkAppendPolyData로 합쳐줘야 한다.

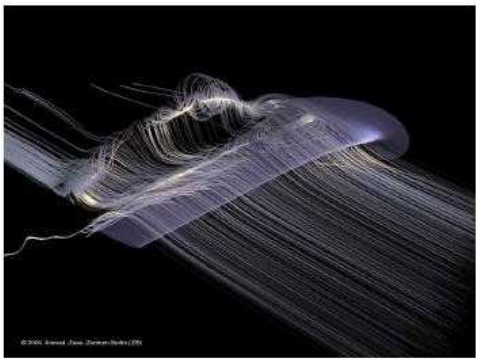


[그림 2-2] 로터 데이터 Vorticity Iso-Surface

3) 스트림라인

스트림라인이란 벡터 데이터 중 유동 데이터(flow data)를 가시화하는데 매우 중요하게 사용되는 표현 기법으로 운동하는 유체의 각 점에서 속도벡터의 방향이 접선이 되도록 그은 곡선을 의미한다. 그림 2-3은 스트림라인 가시화의 실제 예이다. 로더 동역학 데이터 역시 스트림라인 가시화가 매우 중요한데, 이를 이용해서 vortex의 회전 정도를 한눈에 파악할 수 있기 때문이다. VTK에서는 여러 종류의 스트림라인을 제공하는데, GLOVE에서는 스트림라인과 스트림튜브(stream tube)를 제공하고 향후 스트림리본(stream ribbon)과 스트림 파티클(stream

particl)도 제공할 계획이다. VTK에서 스트림라인을 생성하는 방법은 약간 복잡한데 이는 스트림라인을 생성하기 위해 선택해야 하는 옵션이 많기 때문이다.



[그림 2-3] 스트림라인

먼저, VTK에서는 vtkStreamTracer, vtkStreamer, vtkStreamLine 등 여러 종류의 스트림라인 생성을 위한 클래스들이 있는데, GLOVE에서는 vtkStreamTracer를 사용한다. vtkStreamer이나 vtkStreamLine 등의 클래스는 모두 과거에 사용했던 클래스들로 최근에는 더 이상 업데이트 되지 않고 있어 시변환 데이터 지원이나 병렬처리 등이 전혀 구현되지 않고 있다. 물론, VTK 제작자들도 vtkStreamTracer를 사용할 것을 권장하고 있다. vtkStreamTracer를 사용하기 위해서는 먼저 선적분(integration) 방법을 결정해야 하는데, VTK에서는 오일러 기법은 물론 Runge-Kutta 기법도 제공한다. GLOVE에서는 여러 종류의 선적분 방법들 중 2차 Runge-Kutta 방법을 사용하는데, 이 방법은 4차나 5차 Runge-Kutta 방법에 비해 정밀도가 떨어지지만 훨씬 빠른 속도를 제공하기 때문이다. 실제로 2차와 4차 방법을 비교했을 때 두 방법의 오차는 극히 작았던 반면 속도는 1.5배 이상 2차가 빨랐다. 향후 병렬 처리 등을 통해 이러한 속도 문제가 해결된다면 4차 Runge-Kutta 방법을 사용할 계획이다. VTK에서는 vtkRungeKutta2 클래스로 2차 Runge-Kutta 방법을 제공한다. 적분 방향은 전진(forward)와 후진(backword)를 모두 하도록 했고 (SetIntegrationDirectionToBoth() 함수), 스텝 수는 임의로 10000 스텝을 하도록 했다. 본래 스트림라인 생성 도중 필드 데이터 밖으로 나갈 경우 멈추도록 하고 싶었으나 VTK에서 이러한 모드를 제공하지 않아 임의로 10000 스텝을 처리하도록 했다. 물론, VTK는 스트림라인이 필드 밖으로 나가거나 특정 스텝 동안 계속 같은 셀 내에서만 있으면 스트림라인 생성을 멈추도록 하는 기능이 있으나 반드시 스텝 수를 지정하도록 되어 있는 문제가 있다. 따라서 충분히 큰 숫자로 생각되는 10000 스텝을 지정했으나 만약 데이터가 매우 큰 경우에는 스트림라인이 끊겨져 보이는 현상이 발생할 수 있다. 이러한

문제는 향후 반드시 개선해야 하는 문제이다. 물론, 실험 데이터에서는 이러한 문제가 발생하지 않았다.

마지막으로 vtkStreamTracer 역시 단일 블록의 처리만이 가능하므로 블록의 개수만큼 여러 번 실행한 뒤 vtkAppendPolyData로 합치는 과정이 필요하다.

4) 외곽선 추출

VTK에서 외곽선을 추출하기 위해서는 vtkOutlineFilter를 사용하면 된다. 이때, vtkOutlineFilter 역시 단일 블록의 처리만이 가능하므로 블록의 개수만큼 여러 번 실행한 뒤 vtkAppendPolyData로 합치는 과정이 필요하다.

5) vtkRenderer

위에서 설명한 vtkPolyData는 VTK의 데이터 구조로서 이를 실제 화면에 디스플레이하기 위해서는 vtkRenderer에 이를 등록해야 한다. vtkRenderer는 실제 VTK 데이터를 렌더링 하는 클래스인데, 이 클래스에 등록하기 위해서는 다음과 같은 VTK 클래스들이 필요하다.

1. vtkMapper
2. vtkActor
3. vtkColorTransferFunction

여기서 vtkMapper는 VTK 데이터를 OpenGL 등을 위한 그래픽 표현(geometry)으로 맵핑시켜주는 클래스이고 vtkActor는 이러한 그래픽 표현과 별도의 색깔이나 투명도, 물질정보(ambient color, diffuse color 등)을 지정하는 속성(property) 그리고 텍스쳐맵(texture map) 등을 저장, 관리를 위한 클래스이다. vtkActor는 vtkRenderer의 입장에서는 일종의 object로 관리된다. 각 vtkMapper 들은 vtkPolyData를 비롯한 vtkDataSet을 가지고 있으면서 여기에 저장된 활성 스칼라(active scalar)의 값에 따라 색깔을 변경할 수 있는데, 이를 위해서는 vtkColorTransferFunction이 필요하므로 이를 등록해야 한다. 경우에 따라 vtkMapper 내부에 등록된 vtkDataSet의 내용이 변경되는 경우가 있는데, 이럴 때에는 이를 참조

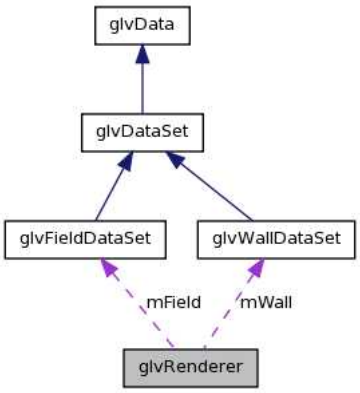
하는 vtkMapper를 찾아서 Update()를 실행시켜야 변경된 내용이 전체 파이프라인에 문제없이 반영된다.

나. GLOVE 렌더러 클래스 설계

GLOVE에서는 위에서 설명한 VTK의 렌더링 기법을 이용해서 렌더링 클래스를 생성했다. 이 클래스의 이름은 glvRenderer이고 GLOVE에서 제공하는 모든 렌더링 기능들을 실제로 구현한다. glvRenderer의 역할은 로우 데이터로부터 위에서 설명한 각종 렌더링 기능들을 실행하고 그 결과를 vtkPolyData와 vtkColorTransferFunction으로 돌려주는 것으로 끝난다. 즉, vtkActor, vtkMapper, vtkRenderer 등은 관리하지 않는 것이다. 이는 이러한 클래스들은 프로그램의 운영체제나 그래픽스 카드, 운영 환경의 종류에 따라 달라질 수 있기 때문이다.

1) 데이터 지원

glvRenderer는 GLOVE의 렌더링 클래스이므로 GLOVE의 데이터 구조를 직접 지원한다. 따라서 사용자는 GLOVE에 GLOVE의 벽면 및 필드 데이터를 직접 등록해야 하고 glvRenderer는 이를 내부 protected 영역에 저장해 놓아 외부에서 접근이 불가능하도록 했다. 이를 위해 glvRenderer는 RegisterWallDataSet(), RegisterFieldDataSet() 등의 함수를 제공한다. 이때, 이미 등록된 데이터가 있을 경우 등록된 데이터를 제거하고 새로운 데이터를 등록하도록 한다. 또, glvRenderer는 이미 등록된 데이터를 제거하는 함수를 명시적으로 지원하는데, RemoveWallDataSet()과 RemoveFieldDataSet()이 그것이다. 등록된 벽면 및 필드 데이터를 얻기 위해서는 GetWallDataSet()과 GetFieldDataSet() 함수를 이용하면 된다. GLOVE 데이터 구조는 내부적으로 VTK 데이터 구조를 포함하고 있는데 GLOVE 렌더링 클래스는 이러한 GLOVE 데이터 내부에 저장된 VTK 데이터를 받아서 VTK 렌더링 기법을 이용, 다각형 데이터를 생성하도록 한다. VTK 데이터는 전적으로 GLOVE 렌더러 내부에서 처리하는 데이터로써 클래스 바깥에서는 접근이 불가능한 데이터이다.



[그림 2-4] glvRenderer 협동 다이어그램

GLOVE 또는 VTK 데이터 이외에도 glvRenderer에서 관리하는 데이터가 있다. 그것은 vtkColorTransferFunction이다. glvRenderer는 내부에 mTF라는 이름의 vtkColorTransferFunction 형식의 2차원 배열을 저장하고 있다. 이 배열에서 첫 번째 행 (mTF[0])는 벽면 데이터의 스칼라 변수들을 위한 transfer function을 저장하고 두 번째 행(mTF[1])은 필드 데이터의 스칼라 변수들을 위한 transfer function을 저장한다. Transfer function에 대한 자세한 설명은 5장에서 하도록 한다.

2) 벽면 데이터 처리

GLOVE의 두 종류의 데이터 중 벽면 데이터에 대한 렌더링 기능은 그리 많지 않다. 이는 GLOVE에서 목표로 하는 로터 동역학의 데이터 중 벽면 데이터에는 많은 정보가 들어있지 않은 까닭이다. 벽면 데이터를 위한 렌더링 기능 중 가장 중요한 것은 벽면의 표면을 스칼라 값에 따라 다른 색으로 디스플레이 하는 것이다. 이는 RenderScalarDisplay() 함수를 통해 가능하다. RenderScalarDisplay() 함수에는 여러 버전이 있는데, 파라미터가 하나도 들어가지 않은 함수는 벽면 데이터의 활성 스칼라(active scalar)를 이용해서 색을 디스플레이 한다. 만약 파라미터로 스칼라의 ID나 이름을 넣으면 해당되는 스칼라를 이용하게 된다. 또 지정한 스칼라 변수가 존재하지 않는 경우에는 GLV_FALSE를 돌려주고 현재 활성 스칼라를 이용해서 색을 디스플레이한다.

벽면 데이터를 위한 또 다른 렌더링 기능으로 데이터의 외곽선을 추출하는 기능이 있다. 이 기능은 RenderWallBoundary() 함수로 가능하다. 이때, 외곽선의 색

은 무조건 흰색이 되도록 했는데, 이는 VTK 파이프라인의 vtkActor에서 수정이 가능한 부분이다. 물론, RenderWallBoundary() 함수 내에서도 색깔 지정이 되도록 수정할 필요도 있을 것이다. 이 함수와 쌍둥이 함수로 필드 데이터를 위한 RenderFieldBoundary() 함수도 있다. 또, 등록된 벽면 및 필드 데이터 이외의 데이터를 위한 RenderBoundary(glvDataSet *) 함수도 존재한다.

3) Iso-surface 처리

glvRenderer는 등록된 두 데이터 중 필드 데이터에 대해 iso-surface를 생성한다. 이를 위한 함수는 두 종류가 있는데, RenderIsoSurface()와 RenderIsoSurfaces() 이다. RenderIsoSurface()는 지정한 하나의 스칼라 값(iso-value)을 이용, iso-surface를 생성하고 RenderIsoSurfaces()는 여러 개의 스칼라 값들을 이용, 여러 종류의 iso-surface들을 생성한다. 이때, 스칼라 값들은 double 배열로 넘겨져야 하고 크기가 함께 따라와야 한다. VTK에서는 단순한 스칼라 값(들)을 이용한 iso-surface 뿐만 아니라 iso-surface의 개수나 각 iso-value의 간격을 지정해서 일정한 간격의 iso-surface를 생성하도록 할 수도 있지만 GLOVE에서는 현재까지 이러한 기능은 지원하지 않고 있다. 이러한 기능은 향후 추가될 예정이다.

4) 스트림라인 처리

스트림라인은 로터 동역학 뿐만 아니라 대부분의 CFD 분야에서 매우 중요하게 다루는 부분이다. glvRenderer에서는 VTK의 여러 종류의 스트림라인 관련 기능들 중 스트림라인 생성과 스트림튜브 생성 기능을 구현했다. 스트림라인은 RenderStreamline()과 RenderStreamlines()를 통해서, 스트림튜브는 RenderStreamTube()와 RenderStreamTubes()를 통해서 처리가 가능하다. 각 함수들은 씨드포인트(seed point)가 (x, y, z) 형태로 지정되어야 한다. 이때, 좌표계는 VTK의 글로벌 좌표계여야 한다. 각 함수들 중 뒤에 s가 붙은 함수들은 물론 여러 개의 스트림라인(또는 튜브)를 동시에 생성하기 위한 함수들이다. 이러한 함수들에는 씨드 포인트가 단 한 개만 들어가지 않고 씨드 포인트들을 위한 배열이 들어가며 배열의 크기도 함께 지정되어야 한다. 스트림튜브를 위한 함수에는 스트림라인을 위한 함수와는 달리 튜브의 반지름도 함께 지정되어야 한다.

다. GIVI와의 인터페이스

1) GIVI

GIVI는 GLOVE를 위한 통합 가시화 인터페이스로 GLOVE Integrated Visualization Interface의 역자이다[5,6,7]. GIVI는 vjVTK를 기반으로 개발되었는데, vjVTK는 VTK와 VRJuggler가 vtkRenderer, vtkMapper, vtkActor 등을 사용한다. 따라서 vtkMapper와 vtkActor에 등록이 가능한 VTK object를 돌려줘야 한다. 여기서는 vtkPolyData와 vtkColorTransferFunction으로 한다. 이를 위해 glvRenderingObject를 만들었다.

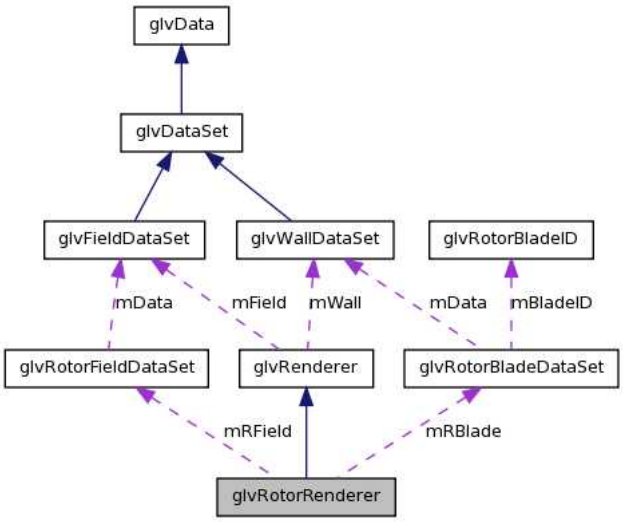
2) glvRenderingObject

glvRenderingObject는 glvRenderer와 GIVI가 렌더링 결과인 vtkPolyData와 vtkColorTransferFunction을 효율적으로 교환하도록 하기 위한 클래스이다. glvRenderingObject는 내부에 vtkPolyData와 vtkColorTransferFunction 뿐만 아니라 전체 데이터의 색깔을 결정하도록 double 형식의 color 배열을 포함하고 있다. glvRenderingObject는 반드시 GLV_RO_TF와 GLV_RO_COLOR 중 한 형식이어야 한다. 여기서 GLV_RO_TF는 내부에 transfer function을 포함하고 있는 데이터이고 GLV_RO_COLOR은 color 정보를 포함하고 있는 데이터이다. glvRenderingObject는 이들 형식을 직접적으로 지정하는 함수는 제공하지 않고 SetTransferFunction()이나 SetColor() 등의 함수에서 암시적으로 처리한다. GIVI의 렌더링 모듈에서는 glvRenderingObject를 실제 렌더링할 때 데이터 형식을 파악한 후 GLV_RO_TF이면 vtkMapper에 등록된 vtkColorTransferFunction을 등록하고 GLV_RO_COLOR이면 vtkActor의 vtkProperty에서 색깔 및 투명도를 등록한다. 또, glvRenderingObject가 GLV_RO_TF 형식이라 하더라도 투명도는 따로 등록이 가능한데, 이는 SetOpacity() 함수를 통해 가능하다. GIVI에서도 glvRenderingObject가 어떤 형식이던 관계없이 GetOpacity() 함수를 통해 투명도를 가져온 후 vtkProperty에 등록시켜야 한다. 만약 SetOpacity() 함수로 따로 등록하지 않을 경우에 기본 투명도는 1.0 이다. glvRenderingObject의 SetColor() 또는 SetOpacity() 함수의 R, G, B, A 값은 모두 0.0 ~ 1.0 사이의 값을 가져야 한다. 만약 이러한 값을 가지지 않으면 자동으로 이 사이의 값으로 clamping 된다.

3. 로터 데이터 렌더러

2장에서 설명한 기본 렌더러는 일반적인 CFD나 구조해석 분야의 데이터를 위한 렌더링 클래스이다. 이 장에서는 로터 데이터만을 위해 최적화된 렌더링 클래스에 대해 설명한다.

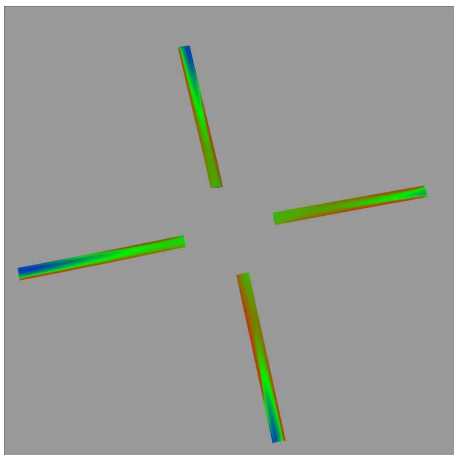
GLOVE에서는 로터 동역학 분야의 데이터만을 위해 최적화된 렌더링 클래스로 glvRotorRenderer를 제공한다. 이 클래스는 기본적으로 glvRenderer로부터 파생된 클래스로 glvRenderer의 기능들을 모두 사용할 수 있을 뿐만 아니라 로터 동역학 데이터를 위한 사용자 친화적인 기능들을 제공한다. 여기에는 주로 속도(velocity) 벡터와 vorticity, Q-criteria 값 등을 이용한 도메인 관점에서의 함수들이 있다. 로터 동역학에서는 로터 블레이드의 표면에서의 압력 분포와 함께 로터 주변의 필드에서의 vortex 정보가 매우 중요하게 다루어진다. 특히, 시변환 (time-variant) 로터 데이터의 경우 앞부분의 로터 블레이드가 발생한 vortex를 뒤에서 따라오는 로터 블레이드에 맞는지를 보는 것이 중요한데, 이러한 도메인 종속적인 분석을 위해 vortex 영역을 가시화 하는 것도 필요하다. 이러한 도메인 종속적인 분석을 위해 glvRotorRenderer에서는 glvRenderer와는 달리 glvRotorBladeDataSet과 glvRotorFieldDataSet을 등록받아 처리한다. 따라서 glvRotorBladeDataSet과 glvRotorFieldDataSet에서 제공하는 로터 데이터 종속적인 함수들을 사용하는 것이 가능하다. 이러한 로터 데이터 등록은 RegisterRotorBladeDataSet() 함수와 RegisterRotorFieldDataSet() 함수로 가능하다.



[그림 3-1] glvRotorRenderer 협동 다이어그램

가. 압력 분포 가시화

로터 데이터의 두 데이터 형식 (glvRotorBladeDataSet, glvRotorFieldDataSet)에는 모두 압력(pressure) 정보가 포함되어 있다. 각 데이터 마다 압력을 가시화하는 방법이 다른데, 블레이드 데이터에서는 블레이드 표면에 압력 분포를 디스플레이 하는 방법으로 가시화하고, 필드 데이터에서는 전체 필드에서 특정 압력값에 대한 iso-surface를 생성하는 방법으로 가시화한다. 이때, 블레이드 데이터와 필드 데이터는 서로 분리된 데이터이므로 압력 값이 서로 연동되어 있지 않다. 따라서 서로 같은 압력 값을 가짐에도 불구하고 서로 다른 색깔로 가시화될 수가 있다. 이러한 경우를 방지하기 위해 GLOVE에서는 등록된 블레이드 데이터와 필드 데이터에서 서로 같은 이름을 가지는 변수들은 같은 transfer function을 사용하도록 했다. transfer function에 대한 자세한 내용은 5장에서 찾아볼 수 있다. glvRotorRenderer에서 등록된 로터 블레이드 표면에 압력 분포를 가시화하기 위해서는 RenderPressureDisplay() 함수를 사용하면 된다. 이 함수를 호출하면 등록된 로터 블레이드 데이터의 현재 타임스텝에서의 표면 데이터에 압력 분포를 색칠하여 가시화한다. 만약, 로터 블레이드 데이터가 등록되어 있지 않으면 NULL을 돌려준다. 로터 필드 데이터에 압력 iso-surface를 생성하기 위해서는 RenderPressureIsoSurface() 함수를 호출하면 된다. 이때, 어떤 압력 값으로 iso-surface를 생성할지를 결정해야 하는데, glvRotorRenderer에서는 이에 대한 어떠한 가이드라인도 제공하지 않고 전적으로 사용자에게 맡긴다. 다만, glvRotorRenderer에 등록된 glvRotorFieldDataSet에서는 GetPressureRange() 함수를 제공하여 전체적인 압력 분포에 대한 정보를 제공하므로 이를 이용해서 적절한 압력 값을 정할 수 있다.



[그림 3-2] 로터 블레이드에서의 압력 분포 가시화

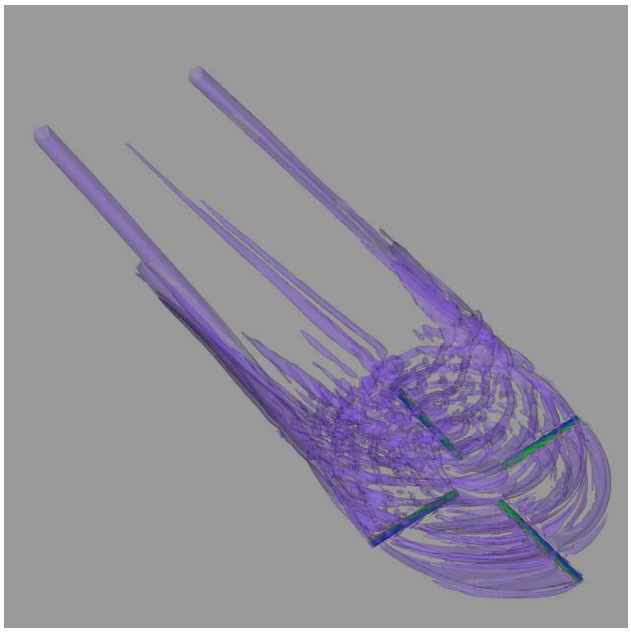
나. Vortex 가시화

로터 데이터에서 가장 중요하게 다루는 부분은 vortex 가시화이다. Vortex는 유동 데이터에서 유동이 급격하게 회전하는 부분을 말하며 회전하는 정도를 vorticity라고 한다. 일반적으로 vortex 영역은 vorticity가 크지만 vorticity가 크다고 전부 vortex인 것은 아니다. 많은 관련 연구자들이 vortex의 정의를 찾기 위해 노력하고 있지만 명확한 정의는 아직까지 만들어지지 않았다. GLOVE에서는 vortex 영역을 찾기 위해 [7]에서 정의한 Q-Criteria 방법을 사용한다. 이를 위한 Q값은 glvRotorFieldDataSet에 반드시 등록되어 있어야 하며 glvRotorFieldDataSet이 생성될 때 자동으로 계산되어 저장된다. 즉, 정상적인 glvRotorFieldDataSet은 모두 Q-Criteria 변수를 포함하고 있다. glvRotorRenderer에서는 이와 같이 glvRotorFieldDataSet에 미리 등록된 Q 값을 사용해서 iso-surface를 생성하는 것으로 vortex 영역을 가시화한다. 이렇게 Q 값을 이용해서 vortex를 가시화하기 위해서는 RenderQCriteriaIsoSurface() 함수를 사용하면 된다. 이때, 어떤 Q 값을 사용할지는 데이터마다 다를 수 있으므로 전적으로 사용자에게 맡기는데, 일반적으로 10^{-4} 이나 10^{-5}을 많이 사용한다. 한편, Q-Criteria를 이용해서 iso-surface를 생성하면서 색깔은 Q-Criteria 값이 아닌 vorticity와 같은 다른 값을 이용해야할 필요가 있다. 이는, Q-Criteria 값으로 색을 칠하면 iso-surface의 모든 표면이 같은 색으로 칠해지는 반면 vorticity나 velocity 등을 이용하면 서로 다른 색깔로 칠해지면서 같은 Q 값을 가지는 영역에서의 vorticity나 velocity의 변화를 함께 확인할 수 있기 때문이다. 이를 위해 glvRotorRenderer에서는 RenderQCriteriaIsoSurfaceByVorticity()와 RenderQCriteriaIsoSurfaceByVelocity() 함수를 제공한다.

시변환 로터 데이터의 경우 선행하는 로터 블레이드에서 생성된 vortex가 후행하는 로터 블레이드에 닿는지를 파악하는 것이 매우 중요하다. GLOVE에서는 이를 파악할 수 있도록 시간 흐름에 따른 블레이드와 vortex의 변화를 애니메이션으로 가시화해준다. 이는 glvRotorRenderer에 등록된 glvRotorFieldDataSet과 glvRotorBladeDataSet의 현재 시간을 동기화한 뒤 점진적으로 증가/감소시키고 새롭게 렌더링함으로써 가능해진다. 이를 통해 연구자는 vortex 영역이 로터 블레이드와 닿는지 아닌지를 가시적으로 알 수 있게 된다.

마지막으로 Q-Criteria를 이용한 votex 영역 추출이 아닌 순수한 vorticity 값의 iso-surface를 보고 싶은 경우가 있는데, 이를 위해 glvRotorRenderer에서는 RenderVorticityIsoSurface() 함수를 제공한다. 이 함수는 glvRotorRenderer에 등록

된 glvRotorFieldDataSet에서 지정한 vorticity 값을 가지는 영역에 대한 iso-surface를 생성한다.

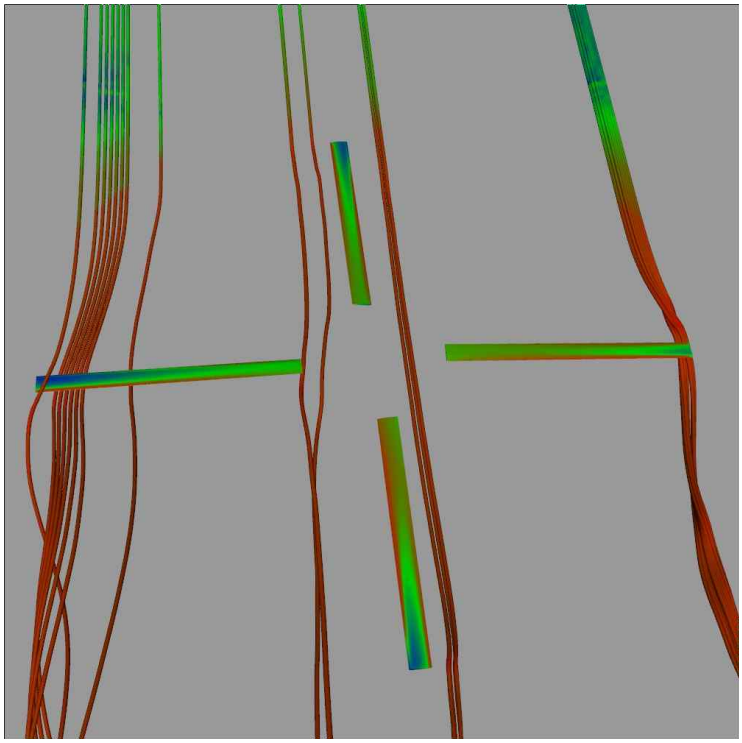


[그림 3-3] 로터 블레이드에서의 Q-Criteria를 이용한 Vortex 가시화

다. 스트림라인 생성

스트림라인은 로터 동역학 데이터에서도 매우 중요하게 다루어진다. Vorticity나 Q-Criteria를 통해 vortex 영역을 파악할 수도 있지만 실제 해당 영역에서의 스트림라인을 그려봄으로써 유동이 실제로 회전하는지 아닌지를 알 수 있기 때문이다. glvRotorRenderer에서는 RenderVelocityStreamline() 함수를 이용해서 스트림라인을 생성할 수 있다. 또, RenderVelocityStreamTube() 함수를 이용하면 스트림 튜브를 생성할 수 있다. 이들 함수는 glvRenderer에서와 마찬가지로 여러 스트림라인들을 동시에 생성할 수 있도록 하는 RenderVelocityStreamlines()와 RenderVelocityStreamTubes() 등의 변종도 제공한다. 그러나 이들 함수들은 모두 정확한 seed point의 위치를 VTK의 글로벌 좌표계 값으로 지정해야 한다는 점에서 문제가 발생한다. 일반적으로 사용자들은 중요한 위치에서 스트림라인이 자동으로 생성되기를 바랄 것이다. 이를 위해 glvRotorRenderer에서는 속도(velocity)나 vorticity가 큰 영역에서 스트림라인을 자동으로 생성하도록 하는 함수를 제공한다. 이러한 함수에는 RenderVelocityStreamlinesByVelocity()와 RenderVelocityStreamlinesByVorticity()가 있다. 각 함수에는 인자로 0.0~100.0 사이의

값을 받는데, 이는 전체 유동 필드에서의 velocity와 vorticity의 크기 정도를 백분율로 나타낸 것이다. 즉, 인자로 90.0이 들어간다면 전체 필드에서 90% 크기를 나타내는 것이고 이는 90% 이상의 velocity(또는 vorticity)를 가지는 모든 영역에서 스트림라인을 생성하도록 한다.



[그림 3-4] 로터 블레이드 주변에서의 스트림라인 가시화

4. 병렬 처리

GLOVE의 렌더링 엔진은 모든 렌더링을 매번 실시간으로 처리한다. 즉, vtkPoly DataSet과 같은 다각형 데이터를 미리 생성하고 디스플레이하는 것이 아니라 매번 장면이 변경될 때마다 렌더링 루틴을 새롭게 실행하는 것이다. 그런데, 데이터가 매우 크거나 생성해야 하는 다각형이 많을 경우 iso-surface 생성이나 스트림라인의 생성의 경우 많은 시간을 필요로 한다. 이러한 점을 해결하기 위해 GLOVE에서는 iso-surface 생성과 스트림라인 생성에 병렬처리를 지원한다. 이번 장에서는 이러한 병렬처리의 방법에 대해 살펴본다.

가. Iso-Surface

Iso-surface 생성은 GLOVE에서 가장 빈번하게 실행되는 가시화 방법이다. GLOVE에서는 기본적으로 압력(pressure), 밀도(density), 속도(velocity), vorticity, Q-Criteria에 대한 iso-surface 생성을 지원한다.

glvRenderer 설명에서 설명했듯이 GLOVE의 데이터 구조가 멀티 블록일 경우 glvRenderer는 각 블록별로 iso-surface를 생성한다고 했다. 즉, 작업을 블록 단위로 나누어 차례대로 수행한 뒤 작업 결과를 하나로 모으는 것이다. 그런데, 각 블록별 iso-surface 생성 작업은 서로 영향을 미치지 않기 때문에 병렬로 동시에 실행이 가능하다. 따라서, GLOVE에서는 데이터 구조가 멀티 블록일 경우 OpenMP를 이용해서 병렬로 iso-surface를 생성하도록 했다. 만약 n개의 블록으로 나뉘어져 있고 m개의 프로세서의 사용이 가능하다면 각 프로세서는 n/m 개의 블록을 할당받아 iso-surface를 생성하게 된다. 이때, vtkAppendPolyData는 여러 프로세서에서 함께 공유하게 되는데, 여러 프로세서에서 이 데이터에 동시에 쓰려고 할 경우 문제가 발생할 수 있다. 이를 방지하기 위해 vtkAppendPolyData에 작업 결과를 쓸 때는 serialize가 되도록 #pragma omp ciritical 을 선언했다. 현재 iso-surface 생성의 병렬화는 여러 가지 문제점이 있는데, 이는 다음과 같다.

1. 작업 밸런싱 문제
2. 작업 결과를 합칠 때 serialize 문제
3. 공유메모리 구조가 아닌 병렬 컴퓨터에서의 구동 문제

먼저, 작업 밸런싱 문제는 각 프로세서에 할당하는 블록의 크기가 서로 다를 수 있다는 문제이다. 즉, 멀티 블록을 구성하는 각 블록의 크기가 서로 다를 수 있는데, 블록의 크기는 무시하고 무조건 프로세서마다 같은 개수의 블록을 할당하는 것이다. 이를 해결하기 위해서는 각 블록의 크기를 고려해서 할당하는 알고리즘이 추가되거나 빨리 끝난 프로세서에 남은 블록을 할당하는 동적 할당 기법을 적용해야 한다. 또, 경우에 따라서는 구성된 멀티 블록과 관계없이 이미 생성된 블록을 나누고 여러 블록을 하나로 합치는 등의 과정을 통해 새로운 멀티 블록을 생성해야 할 필요도 생긴다. 두 번째로 작업 결과의 serialize 문제는 각 프로세서마다 vtkAppendPolyData를 만들어서 각자 관리하고 모든 프로세서가 작업을 끝낸 뒤 각자 가지고 있는 다각형 데이터를 하나로 합치는 방법으로 처리가 가능하다. 다만, 이 방법은 모든 프로세서가 vtkAppendPolyData를 생성하고 관리해야 하기 때문에 메모리 사용량이 늘어난다는 단점이 발생한다. 마지막으로 병렬 컴퓨터에서의 구동 문제는 현재 OpenMP로 작성되어 있는 코드를 모두 MPI나 기타 분산메모리 구조를 위한 병렬화 라이브러리를 사용해서 바꾸어야 하는 문제가 있다. 이러한 문제점들은 모두 향후 해결해야할 과제로 남겨둔다.

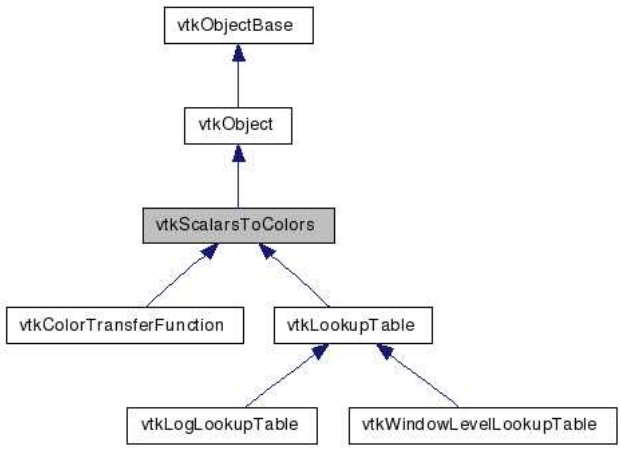
나. 스트림라인

스트림라인 생성은 GLOVE에서 제공하는 렌더링 기능들 중 가장 많은 시간을 필요로 하는 기능이다. 일반적으로 하나의 스트림라인을 생성하기 위해서 여러 데이터 블록을 접근해야 하기 때문에 이 경우에는 데이터 블록 단위로 프로세스를 나누는 것은 문제가 있다. 즉, 만약 데이터 블록 단위로 프로세스를 나눈다면 스트림라인의 정의상 한 프로세스(블록)에서의 스트림라인 처리가 끝나야지만 다음 프로세스에서 스트림라인 처리를 할 수 있기 때문에 작업이 serialize되게 된다. 이러한 문제를 방지하기 위해 스트림라인 생성에서는 작업을 seed point 단위로 나누어서 처리했다. 즉, 데이터 블록은 모든 프로세서에서 공유하고 각 프로세서마다 seed point를 균등하게 할당, 처리하도록 한 것이다.

5. Transfer Function

GLOVE의 렌더링 엔진에서 렌더링한 결과는 모두 vtkPolyDataSet으로 반환된다. 그런데 vtkPolyDataSet 내부에는 다각형 데이터의 기하학적인 정보만 저장될 뿐 색깔 등의 정보는 저장되지 않는다. 따라서 이를 위해 GLOVE에서는 glvRenderingObject에 transfer function이나 색깔 정보 자체를 저장한 뒤 glvRenderingObject 단위로 반환하게 된다. 이때, 데이터 내부의 스칼라 값에 따라 색깔을 변화시키고자하는 경우에 transfer function을 함께 반환하는데, 이 transfer function은 VTK에서 제공하는 vtkColorTransferFunction 형태로 반환되어야 한다. 그런데, 이러한 vtkColorTransferFunction을 각 데이터마다 사용자가 직접 생성하기는 어려움이 많다. 이러한 점을 해결하기 위해 glvRenderer와 glvRotorRenderer에서는 자동으로 transfer function을 생성시키는 함수를 제공하는데, 이는 GenAutomaticTF() 함수이다. 다만, 이 함수는 protected 함수로써 클래스 외부에서는 호출이 불가능하고 전적으로 스칼라 변수를 등록할 때 내부에서 자동으로 호출하게 된다. 이번 장에서는 이러한 자동 transfer function 기능에 대해 살펴본다.

GLOVE에서 모든 transfer function은 vtkColorTransferFunction의 형태로 전달된다. vtkColorTransferFunction은 VTK에서 제공하는 transfer function 중 하나로 vtkScalarsToColors 클래스를 상속받는다. vtkScalarsToColors는 스칼라 값을 색깔로 변경하기 위한 일반적인 목적의 클래스로 vtkColorTransferFunction 뿐만 아니라 vtkLookupTable 클래스도 자식 클래스로 가지고 있다. vtkColorTransferFunction은 piecewise hermite 함수를 이용해서 스칼라 값을 RGB 또는 HSV 값으로 매핑한다. vtkColorTransferFunction에 AddRGBPoint() 또는 AddHSVPoint() 함수로 중간 포인트들을 입력하면 vtkColorTransferFunction은 입력된 포인트들을 이용해서 piecewise hermite 함수를 생성한다. 이 때 각 포인트들은 linear 또는 log scale의 함수로 연결된다. 반면에 vtkLookupTable은 table을 이용해서 스칼라 값을 RGBA 값으로 또는 거꾸로 RGBA 값을 스칼라 값으로 변경시키는 클래스이다. vtkLookupTable은 반드시 색깔 table이 존재해야 하는데, 이 table은 직접적으로 색깔 값을 입력하는 것으로도 생성이 가능하고 또는 SetRange() 등의 함수로 스칼라 영역을 입력한 뒤 SetRampToLinear() 등의 함수로 RGBA table을 생성하는 것도 가능하다.



[그림 5-1] vtkScalarsToColors 클래스 계층구조

GLOVE에서는 VTK에서 지원하는 두 transfer function들 중 vtkColorTransferFunction을 사용하는데, 이는 GLOVE의 스칼라 데이터에 대한 히스토그램(histogram)을 작성해보면 값의 분포가 균일하지 않고 특정 영역에 몰려있는데, vtkColorTransferFunction은 이러한 특정 영역을 강조해서 transfer function을 생성하는데에 유리하기 때문이다. 물론 향후에는 vtkColorTransferFunction 뿐만 아닌 vtkLookupTable도 지원하도록 할 예정이다.

GLOVE 렌더링 엔진에서 자동으로 생성하는 transfer function은 해당 스칼라의 최소값을 (0, 0, 1)의 파랑색, 중간값을 (0, 1, 0)의 초록색, 최대값을 (1, 0, 0)의 붉은색으로 하는 함수이다. 이때, 이 함수는 각 값들을 key로 다시 그 중간값들((0, 0,5, 0,5)나 (0.5, 0.5, 0) 등)을 생성하고 또 그 중간값을 key로 새로운 중간값들((0, 0.25, 0.75) 등)을 생성하는 등 재귀적(recursive)으로 함수를 세밀화할 수 있다. 여기서 생성되는 중간값들이 많으면 많아질 수록 transfer function의 정확도는 높아질 것이다. 이때, 최소, 최대값은 명확히 정해져 있으나 중간값들을 무엇으로 할지가 문제이다. VTK 데이터 구조에서는 전체 데이터에서 최소, 최대값을 손쉽게 구할 수 있는 함수를 제공하므로 이는 쉽게 구하는 것이 가능하다. 이때, 데이터가 전체적으로 균일하게 분포되어 있다면 중간값들은 단순히 *(최소값 + 최대값) / 2*의 방식으로 계산이 될 것이다. 그러나 전체적으로 균일하지 못하고 한쪽으로 치우쳐져 있다면 전혀 다른 값이 될 가능성이 없지 않다. 이러한 이유로 GLOVE 렌더링 엔진에서는 전체 스칼라 값을 정렬한 뒤 실제 중간에 위치한 값이 무엇인지를 찾도록 했다. 단순히 (최소 + 최대) / 2로 한 경우와 정렬을 통해 중간값을 찾는 경우의 가시화 결과는 그림과 같다.

GLOVE에서 다루는 로터 데이터는 기본적으로 시변환 데이터임을 전제로 하고 있는데 만약 transfer function을 각 타임 스텝마다 변경이 된다면 애니메이션 시

동일한 스칼라 값을 가지는 부분의 색깔이 계속적으로 변화가 되는 문제가 발생한다. 따라서 transfer function을 생성할 때 전체 타임 스텝에 대한 단일 transfer function을 생성해야만 한다. 이와 같이 전체 데이터에 대한 transfer function을 생성할 경우 전체 데이터를 정렬해야 하는데 이것은 매우 많은 시간을 필요로 한다. 이를 해결할 수 있는 방법에는 다음과 같은 것들이 있다.

1. 정렬의 병렬화
2. 히스토그램을 이용한 영리한 transfer function 생성
3. 전처리

여기서 정렬의 병렬화는 여러 종류의 병렬 정렬 알고리즘이 있으므로 이를 이용하면 구현이 어렵지 않다. 또, 히스토그램을 생성하면 전체 데이터의 분포를 알 수 있는데, 이 정보를 이용하면 중요한 정보를 놓치지 않는 transfer function의 생성이 가능할 것으로 판단된다. 여기서 정렬 알고리즘은 보통 $O(nlogn)$의 복잡도를 갖는 반면 히스토그램 생성은 $O(n)$으로 충분하기 때문에 훨씬 빠른 transfer function 생성이 가능하다. 마지막으로 데이터가 실시간으로 생성되는 경우가 아닌 미리 생성된 데이터를 이용해서 가시화하는 경우에는 데이터 정렬을 미리 처리해놓고 사용할 수 있다.

6. 결론

본 기술문서에서는 GLOVE를 위한 렌더링 엔진에 대해 살펴보았다. GLOVE의 렌더링 엔진은 기본적으로 VTK의 렌더링 기능을 기반으로 하였지만 로터 동역학 분야에 최적화해서 해당 분야 연구자들이 쉽게 사용할 수 있도록 했다. 우선 로터 동역학 분야 뿐만 아니라 기타 다른 CFD나 구조해석 분야의 데이터도 처리할 수 있도록 glvRenderer를 설계, 구현했고 이를 바탕으로 로터 데이터에 최적화된 glvRotorRenderer를 개발했다. 이렇게 개발한 GLOVE 렌더링 엔진은 다각형 데이터와 함께 색깔, transfer function 들을 GIVI에 돌려주는데, 이를 효과적으로 처리하기 위한 glvRenderingObject 클래스도 소개했다. 로터 동역학 분야에서 중요하게 다루는 가시화 방법으로 iso-surface와 스트림라인이 있는데 이러한 기능들을 구현함으로써 vortex 영역과 이 영역에서의 유동의 흐름을 가시화하는데 성공했고 이를 사용자가 쉽게 사용할 수 있도록 glvRotorRenderer에서 함수 형태로 구현했다. 그런데 이러한 가시화 방법은 그 알고리즘이 매우 복잡해서 데이터가 거대할 경우 많은 시간을 필요로 한다. 이를 해결하기 위해 병렬 처리를 도입했으며 OpenMP를 이용한 빠른 렌더링을 구현했다. 물론 이는 공유 메모리 구조에서만 사용이 가능하지만 향후에는 분산 메모리 구조에서도 사용이 가능하도록 확장할 계획이다.

그러나 현재의 GLOVE 렌더링 엔진은 속도가 빠르지 못한 점과 영상 품질이 높지 못하다는 문제가 있다. 여기서 속도 문제는 앞서 설명했던 병렬화의 효율을 보다 향상시키고 기본 가시화 알고리즘의 성능도 보다 향상시키는 방법으로 해결이 가능하다. 그러나 영상 품질을 높이는 방법은 GLOVE 렌더링 엔진이 VTK를 기반으로 하는 한 풀기 어려운 문제이다. 특히 볼륨 렌더링 등을 이용해서 가시화를 하고자 할 경우 vjVTK를 기반으로 하는 GIVI와의 인터페이스에서 많은 문제가 발생할 수 있다. 향후에는 이러한 문제를 해결하기 위해 VTKEdge나 기타 다른 쉐이더 사용 볼륨 렌더링 알고리즘을 적용할 계획이다.

7. 참고문헌

[1] S.Bryson, "Virtual reality in scientific visualization", Communications of the ACM, 1996.

[2] A.van Dam, A.S.Forsberg, D.H.Laidlaw, J.J.LaViola, R.M.Simpson, "Immersive VR for Scientific Visualization: A Progress Report", IEEE Computer Graphics and Applications, 2000.

[3] W.Schroeder, K.Martin, B.Lorensen, "The Visualization Toolkit, an Object-Oriented Approach to 3D Graphics, 4th Edition", Kitware, 2006.

[4] VTK, http://www.vtk.org

[5] Bokhee Keum, Youngju Hur, Geebum Koo, Joongyoun Lee, "A Global High-Performance VIrtual Environment for Collaborative Immersive Interaction", HPC Asia & APAN 2009

[6] GLOVE, http://www.vce.kr/svwiki/GLOVE

[7] J. Jeong and F. Hussain, "On the Identification of a Vortex", Journal of Fluid Mechanics, 285, pp.69-94, 1995

Appendix. GLOVE Renderer 레퍼런스 라이브러리

Contents

Chapter 1

Class Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Class Documentation

3.1 glvRenderer Class Reference

glvRenderer (p. 5) is a class for GLOVE rendering

`#include <glvRenderer.h>`

Inherited by **glvRotorRenderer**.

Public Member Functions

- int **RegisterWallDataSet** (glvWallDataSet *wall)
- int **RegisterFieldDataSet** (glvFieldDataSet *field)
- vtkColorTransferFunction * **GetTransferFunction** (int ID)
- int **RemoveWallDataSet** ()
- int **RemoveFieldDataSet** ()
- int **RemoveTransferFunction** (int ID)
- glvWallDataSet * **GetWallDataSet** ()
- glvFieldDataSet * **GetFieldDataSet** ()
- **glvRenderingObject** * **RenderWallBoundary** ()
- **glvRenderingObject** * **RenderFieldBoundary** ()
- **glvRenderingObject** * **RenderBoundary** (glvDataSet *data)
- **glvRenderingObject** * **RenderWallMesh** ()
- **glvRenderingObject** * **RenderFieldMesh** ()
- **glvRenderingObject** * **RenderMesh** (glvDataSet *data)
- **glvRenderingObject** * **RenderScalarDisplay** ()
- **glvRenderingObject** * **RenderScalarDisplay** (int ID)
- **glvRenderingObject** * **RenderScalarDisplay** (char *name)
- **glvRenderingObject** * **RenderScalarDisplay** (glvDataSet *data)
- **glvRenderingObject** * **RenderIsoSurface** (double value)
- **glvRenderingObject** * **RenderIsoSurface** (int ID, double value)
- **glvRenderingObject** * **RenderIsoSurface** (char *name, double value)
- **glvRenderingObject** * **RenderIsoSurfaces** (double *values, int size)

- **glvRenderingObject** * **RenderIsoSurfaces** (int ID, double *values, int size)
- **glvRenderingObject** * **RenderIsoSurfaces** (char *name, double *value, int size)
- **glvRenderingObject** * **RenderStreamline** (double x, double y, double z)
- **glvRenderingObject** * **RenderStreamline** (char *name, double x, double y, double z)
- **glvRenderingObject** * **RenderStreamline** (double point[3])
- **glvRenderingObject** * **RenderStreamline** (char *name, double point[3])
- **glvRenderingObject** * **RenderStreamlines** (double *points[3], int size)
- **glvRenderingObject** * **RenderStreamlines** (char *name, double *points[3], int size)
- **glvRenderingObject** * **RenderStreamTube** (double x, double y, double z, double radius)
- **glvRenderingObject** * **RenderStreamTube** (char *name, double x, double y, double z, double radius)
- **glvRenderingObject** * **RenderStreamTube** (double point[3], double radius)
- **glvRenderingObject** * **RenderStreamTube** (char *name, double point[3], double radius)
- **glvRenderingObject** * **RenderStreamTubes** (double *points[3], int size, double radius)
- **glvRenderingObject** * **RenderStreamTubes** (char *name, double *points[3], int size, double radius)

Static Public Member Functions

- static **glvRenderer** * **New** ()

  *Create new **glvRenderer** (p. 5) object.*

Protected Member Functions

- vtkSmartPointer< vtkPolyData > **CreateMesh** (glvDataSet *data)
- vtkColorTransferFunction * **GenAutomaticTF** (glvDataSet *data, int id)

Protected Attributes

- vtkColorTransferFunction * **mTF** [2][GLV_MAX_TRANSFER_FUNCTION]
- vtkColorTransferFunction * **mCurrentTF**

  pointer to current active transfer function

- glvWallDataSet * **mWall**
  Registered wall dataset.
- glvFieldDataSet * **mField**
  Registered field dataset.

3.1.1 Detailed Description

glvRenderer (p. 5) is a class for GLOVE rendering

3.1.2 Member Function Documentation

3.1.2.1 vtkSmartPointer< vtkPolyData > glvRenderer::CreateMesh (glvDataSet * *data*) [protected]

Create mesh polygons from the input glvDataSet. If input dataset is time-variant, dataset of current time step is rendered.

Parameters:

data glvDataSet object

Returns:

NULL if fails

3.1.2.2 vtkColorTransferFunction * glvRenderer::GenAutomaticTF (glvDataSet * *data*, int *id*) [protected]

Automatically generate transfer function of given scalar in the given dataset.

Parameters:

data glvDataSet object
id scalar ID

Returns:

NULL if fails

3.1.2.3 glvFieldDataSet * glvRenderer::GetFieldDataSet ()

Get registered field dataset

Returns:

NULL if fails

3.1.2.4 vtkColorTransferFunction * glvRenderer::GetTransferFunction (int *ID*)

Return transfer function of given ID

Parameters:

transfer function ID

Returns:

NULL if fails

3.1.2.5 glvWallDataSet * glvRenderer::GetWallDataSet ()

Get registered wall dataset

Returns:

NULL if fails

3.1.2.6 int glvRenderer::RegisterFieldDataSet (glvFieldDataSet * *field*)

Register field dataset. Only one field dataset can be registered.

Parameters:

field field dataset to register

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.1.2.7 int glvRenderer::RegisterWallDataSet (glvWallDataSet * *wall*)

Register wall dataset. Only one wall dataset can be registered.

Parameters:

wall wall dataset to register

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.1.2.8 int glvRenderer::RemoveFieldDataSet ()

Remove registered field dataset

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.1.2.9 int glvRenderer::RemoveTransferFunction (int *ID*)

Remove registered transfer function by ID

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.1.2.10 int glvRenderer::RemoveWallDataSet ()

Remove registered wall dataset

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.1.2.11 glvRenderingObject * glvRenderer::RenderBoundary (glvDataSet * *data*)

Render boundary of given glvDataSet

Parameters:

data glvDataSet or its subclass

Returns:

NULL if fails

3.1.2.12 glvRenderingObject * glvRenderer::RenderFieldBoundary ()

Render boundary of registered field dataset

Returns:

NULL if fails

3.1.2.13 glvRenderingObject * glvRenderer::RenderFieldMesh ()

Render mesh of registered field dataset

Returns:

NULL if fails

3.1.2.14 glvRenderingObject * glvRenderer::RenderIsoSurface (char * *name*, double *value*)

Render Iso surface of registered field dataset by given scalar

Parameters:

value iso value

Returns:

NULL if fails

3.1.2.15 glvRenderingObject * glvRenderer::RenderIsoSurface (int *ID*, double *value*)

Render Iso surface of registered field dataset by given scalar

Parameters:

value iso value

Returns:

NULL if fails

3.1.2.16 glvRenderingObject * glvRenderer::RenderIsoSurface (double *value*)

Render Iso surface of registered field dataset by it's active scalar

Parameters:

value iso value

Returns:

NULL if fails

3.1.2.17 glvRenderingObject * glvRenderer::RenderIsoSurfaces (char * *name*, double * *value*, int *size*)

Render Iso surfaces of registered field dataset by givn scalar using given iso-values

Parameters:

name scalar name

values array of iso values

size size of iso value array

Returns:

NULL if fails

3.1.2.18 glvRenderingObject * glvRenderer::RenderIsoSurfaces (int *ID*, double * *values*, int *size*)

Render Iso surfaces of registered field dataset by givn scalar using given iso-values

Parameters:

ID scalar ID

values array of iso values

size size of iso value array

Returns:

NULL if fails

3.1.2.19 glvRenderingObject * glvRenderer::RenderIsoSurfaces (double * *values*, int *size*)

Render Iso surfaces of registered field dataset by it's active scalar using given iso-values

Parameters:

values array of iso values

size size of iso value array

Returns:

NULL if fails

3.1.2.20 glvRenderingObject * glvRenderer::RenderMesh (glvDataSet * *data*)

Render mesh of given glvDataSet

Returns:

NULL if fails

3.1.2.21 glvRenderingObject * glvRenderer::RenderScalarDisplay (glvDataSet * *data*)

Render given wall dataset by active scalar of the dataset

Returns:

NULL if fails

3.1.2.22 glvRenderingObject * glvRenderer::RenderScalarDisplay (char * *name*)

Render registered wall dataset by givn scalar

Parameters:

name scalar name

Returns:

NULL if fails

3.1.2.23 glvRenderingObject * glvRenderer::RenderScalarDisplay (int *ID*)

Render registered wall dataset by givn scalar

Parameters:

ID scalar ID

Returns:

NULL if fails

3.1.2.24 glvRenderingObject * glvRenderer::RenderScalarDisplay ()

Render registered wall dataset by active scalar

Returns:

NULL if fails

3.1.2.25 glvRenderingObject * glvRenderer::RenderStreamline (char * *name*, double *point*[3])

Render streamline of registered field dataset from the given seed point location

Parameters:

name vector name

point seed point location

Returns:

NULL if fails

3.1.2.26 glvRenderingObject * glvRenderer::RenderStreamline (double *point*[3])

Render streamline of registered field dataset from the given seed point location

Parameters:

point seed point location

Returns:

NULL if fails

3.1.2.27 glvRenderingObject * glvRenderer::RenderStreamline (char * *name*, double *x*, double *y*, double *z*)

Render streamline of registered field dataset from the given seed point location

Parameters:

vector name

x X coordinate of seed point

y Y coordinate of seed point

z Z coordinate of seed point

Returns:

NULL if fails

3.1.2.28 glvRenderingObject * glvRenderer::RenderStreamline (double *x*, double *y*, double *z*)

Render streamline of registered field dataset from the given seed point location

Parameters:

x X coordinate of seed point

y Y coordinate of seed point

z Z coordinate of seed point

Returns:

NULL if fails

3.1.2.29 glvRenderingObject* glvRenderer::RenderStreamlines (char * *name*, double * *points*[3], int *size*)

Render multiple streamlines of registered field dataset from the given seed point locations

Parameters:

name vector name

points array of seed points number of seed points

Returns:

NULL if fails

3.1.2.30 glvRenderingObject * glvRenderer::RenderStreamlines (double * *points*[3], int *size*)

Render multiple streamlines of registered field dataset from the given seed point locations

Parameters:

points array of seed points number of seed points

Returns:

NULL if fails

3.1.2.31 glvRenderingObject* glvRenderer::RenderStreamTube (char * *name*, double *point*[3], double *radius*)

Render stream tube of registered field dataset from the given seed point location

Parameters:

name vector name

point seed point location

radius radius of stream tube

Returns:

NULL if fails

3.1.2.32 glvRenderingObject * glvRenderer::RenderStreamTube (double *point*[3], double *radius*)

Render stream tube of registered field dataset from the given seed point location

Parameters:

point seed point location

radius radius of stream tube

Returns:

NULL if fails

3.1.2.33 glvRenderingObject* glvRenderer::RenderStreamTube (char * *name*, double *x*, double *y*, double *z*, double *radius*)

Render stream tube of registered field dataset from the given seed point location

Parameters:

name vector name

x X coordinate of seed point

y Y coordinate of seed point

z Z coordinate of seed point

radius radius of stream tube

Returns:

NULL if fails

3.1.2.34 glvRenderingObject * glvRenderer::RenderStreamTube (double *x*, double *y*, double *z*, double *radius*)

Render stream tube of registered field dataset from the given seed point location

Parameters:

x X coordinate of seed point

y Y coordinate of seed point

z Z coordinate of seed point

radius radius of stream tube

Returns:

NULL if fails

3.1.2.35 glvRenderingObject* glvRenderer::RenderStreamTubes (char * *name*, double * *points*[3], int *size*, double *radius*)

Render multiple stream tubes of registered field dataset from the given seed point locations

Parameters:

name vector name

points array of seed points number of seed points

radius radius of stream tube

Returns:

NULL if fails

3.1.2.36 glvRenderingObject * glvRenderer::RenderStreamTubes (double * *points*[3], int *size*, double *radius*)

Render multiple stream tubes of registered field dataset from the given seed point locations

Parameters:

points array of seed points number of seed points

radius radius of stream tube

Returns:

NULL if fails

3.1.2.37 glvRenderingObject * glvRenderer::RenderWallBoundary ()

Render boundary of registered wall dataset

Returns:

NULL if fails

3.1.2.38 glvRenderingObject * glvRenderer::RenderWallMesh ()

Render mesh of registered wall dataset

Returns:

NULL if fails

3.1.3 Member Data Documentation

3.1.3.1 vtkColorTransferFunction* glvRenderer::mTF[2][GLV_MAX_TRANSFER_FUNCTION] [protected]

Registered transfer functions. 1st entry is for default TF of wall dataset. 2nd entry is for default TF for field dataset.

The documentation for this class was generated from the following files:

- /home/osung/tmp/glove/trunk/include/renderer/glvRenderer.h
- /home/osung/tmp/glove/trunk/src/renderer/glvRenderer.cc

3.2 glvRenderingObject Class Reference

glvRenderingObject (p. 18) is set of rendering objects. All member functions and variables are public.

`#include <glvRenderingObject.h>`

Public Member Functions

- void **Delete** ()

  *Delete **glvRenderingObject** (p. 18) object.*

- void **SetPolyData** (vtkSmartPointer< vtkPolyData > polydata)
- void **SetTransferFunction** (vtkColorTransferFunction *tf)
- void **SetColor** (double r, double g, double b, double a)
- void **SetColor** (double r, double g, double b)
- void **SetColor** (double color[4])
- void **SetOpacity** (double a)
- vtkSmartPointer< vtkPolyData > **GetPolyData** ()
- vtkColorTransferFunction * **GetTransferFunction** ()
- double * **GetColor** ()
- double **GetOpacity** ()

  Return opacity.

- int **GetType** ()

  Return type of the rendering object.

Static Public Member Functions

- static **glvRenderingObject** * **New** ()

  *Create new **glvRenderingObject** (p. 18) object.*

Private Attributes

- vtkSmartPointer< vtkPolyData > **_mPolyData**

  internal polygonal dataset

- vtkColorTransferFunction * **_mTF**

  internal transfer function

- double **_mColor** [4]

  internal color. RGBA

- int **_mType**

type of the rendering object

3.2.1 Detailed Description

glvRenderingObject (p. 18) is set of rendering objects. All member functions and variables are public.

3.2.2 Member Function Documentation

3.2.2.1 double * glvRenderingObject::GetColor ()

Return color

Returns:

NULL if fails

3.2.2.2 vtkSmartPointer< vtkPolyData > glvRenderingObject::GetPolyData ()

Return registered polygonal data

Returns:

NULL if fails

3.2.2.3 vtkColorTransferFunction * glvRenderingObject::GetTransferFunction ()

Return registered transfer function

Returns:

NULL if fails

3.2.2.4 void glvRenderingObject::SetColor (double *color*[4])

Set color of this rendering object. Set type to GLV_RO_COLOR.

Parameters:

color RGBA format color

3.2.2.5 void glvRenderingObject::SetColor (double *r*, double *g*, double *b*)

Set color of this rendering object. Set type to GLV_RO_COLOR.

Parameters:

r red

g green

b blue

3.2.2.6 void glvRenderingObject::SetColor (double *r*, double *g*, double *b*, double *a*)

Set color of this rendering object. Set type to GLV_RO_COLOR.

Parameters:

r red

g green

b blue

a alpha (opacity)

3.2.2.7 void glvRenderingObject::SetOpacity (double *a*)

Set opacity of this rendering object

Parameters:

a alpha (opacity)

3.2.2.8 void glvRenderingObject::SetPolyData (vtkSmartPointer< vtkPolyData > *polydata*)

Register polygonal data. Replace if already registered

Parameters:

polydata polygonal dataset to register

3.2.2.9 void glvRenderingObject::SetTransferFunction (vtkColorTransferFunction * *tf*)

Register transfer function. Replace if already registered

Parameters:

tf transfer function to register

The documentation for this class was generated from the following files:

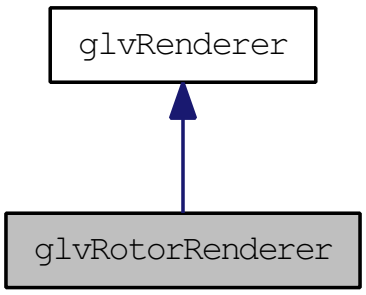
- /home/osung/tmp/glove/trunk/include/renderer/glvRenderingObject.h
- /home/osung/tmp/glove/trunk/src/renderer/glvRenderingObject.cc

3.3 glvRotorRenderer Class Reference

glvRotorRenderer (p. 22) is a subclass of **glvRenderer** (p. 5). It is specalized class for rendering of rotor dataset

`#include <glvRotorRenderer.h>`

Inherits **glvRenderer**.Collaboration diagram for glvRotorRenderer:



Public Member Functions

- int **RegisterRotorBladeDataSet** (glvRotorBladeDataSet *blade)
- int **RegisterRotorFieldDataSet** (glvRotorFieldDataSet *field)
- glvRotorBladeDataSet * **GetRotorBladeDataSet** ()

  Get registered rotor blade dataset.

- glvRotorFieldDataSet * **GetRotorFieldDataSet** ()

  Get registered rotor field dataset.

- **glvRenderingObject** * **RenderRotorBladeBoundary** ()

  Render boundary of rotor blade dataset.

- **glvRenderingObject** * **RenderRotorBladeBoundary** (int baldeID)

  Render boundary of rotor blade dataset.

- **glvRenderingObject** * **RenderRotorFieldBoundary** ()

  Render boundary of rotor field dataset.

- **glvRenderingObject** * **RenderRotorBladeMesh** ()

  Render mesh of rotor blade dataset.

- **glvRenderingObject** * **RenderRotorBladeMesh** (int bladeID)

  Render mesh of rotor blade dataset.

- **glvRenderingObject** * **RenderRotorFieldMesh** ()

  Render mesh of rotor field dataset.

- **glvRenderingObject** * **RenderPressureDisplay** ()
  Render wall dataset with pressure.
- **glvRenderingObject** * **RenderPressureDisplay** (int bladeID)
  Render wall dataset with pressure.
- **glvRenderingObject** * **RenderPressureIsoSurface** (double value)
  Render Iso-surface with pressure.
- **glvRenderingObject** * **RenderDensityIsoSurface** (double value)
  Render Iso-surface with density.
- **glvRenderingObject** * **RenderVelocityIsoSurface** (double value)
  Render Iso-surface with velocity magnitude.
- **glvRenderingObject** * **RenderVorticityIsoSurface** (double value)
  Render Iso-surface with vorticity magnitude.
- **glvRenderingObject** * **RenderQCriteriaIsoSurface** (double value)
  Render Q-Criteria Iso Surface.
- **glvRenderingObject** * **RenderQCriteriaIsoSurfaceByVorticity** (double value)
  Render Q-Criteria Iso Surface and color using vorticity TF.
- **glvRenderingObject** * **RenderPressureIsoSurfaces** (double *values, int size)
  Render multiple Iso-surfaces with pressure.
- **glvRenderingObject** * **RenderDensityIsoSurfaces** (double *values, int size)
  Render multiple Iso-surfaces with density.
- **glvRenderingObject** * **RenderVelocityIsoSurfaces** (double *values, int size)
  Render multiple Iso-surfaces with velocity magnitude.
- **glvRenderingObject** * **RenderVorticityIsoSurfaces** (double *values, int size)
  Render multiple Iso-surfaces with vorticity magnitude.
- **glvRenderingObject** * **RenderVelocityStreamline** (double point[3])
  Render a streamline of velocity.
- **glvRenderingObject** * **RenderVelocityStreamline** (double x, double y, double z)

Render a streamline of velocity.

- **glvRenderingObject** * **RenderVelocityStreamlines** (double *points[3], int size)

  Render multiple streamlines of velocity.

- **glvRenderingObject** * **RenderVelocityStreamlinesByVelocity** (double percent)

  Render multiple streamlines of velocity.

- **glvRenderingObject** * **RenderVelocityStreamlinesByVorticity** (double percent)

  Render multiple streamlines of velocity.

- **glvRenderingObject** * **RenderVelocityStreamTube** (double point[3], double radius)

  Render a stream tube of velocity.

- **glvRenderingObject** * **RenderVelocityStreamTube** (double x, double y, double z, double radius)

  Render a stream tube of velocity.

- **glvRenderingObject** * **RenderVelocityStreamTubes** (double *points[3], int size, double radius)

  Render multiple stream tubes of velocity.

- **glvRenderingObject** * **RenderVelocityStreamTubesByVelocity** (double percent, double radius)

  Render multiple stream tubes of velocity.

- **glvRenderingObject** * **RenderVelocityStreamTubesByVorticity** (double percent, double radius)

  Render multiple stream tubes of velocity.

- **glvRenderingObject** * **RenderVelocityStreamTubesByQCriteria** (double percent, double radius)

  Render multiple stream tubes of velocity.

Static Public Member Functions

- static **glvRotorRenderer** * **New** ()

  *Create new **glvRotorRenderer** (p. 22).*

Protected Attributes

- glvRotorBladeDataSet * **mRBlade**

  Registered rotor blade dataset.

3.3.1 Detailed Description

glvRotorRenderer (p. 22) is a subclass of **glvRenderer** (p. 5). It is specalized class for rendering of rotor dataset

3.3.2 Member Function Documentation

3.3.2.1 int glvRotorRenderer::RegisterRotorBladeDataSet (glvRotorBladeDataSet * *blade*)

Register rotor blade dataset. Only one rotor blade dataset can be registered.

3.3.2.2 int glvRotorRenderer::RegisterRotorFieldDataSet (glvRotorFieldDataSet * *field*)

Register rotor field dataset. Only one rotor field dataset can be registered.

The documentation for this class was generated from the following files:

- /home/osung/tmp/glove/trunk/include/rotor/glvRotorRenderer.h
- /home/osung/tmp/glove/trunk/src/rotor/glvRotorRenderer.cc

Index