ISBN 978-89-6211-396-9

GLOVE 데이터 구조 설계 및 구현

(Design and Implementation of GLOVE Dataset)

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

수식 차례

1. 서론

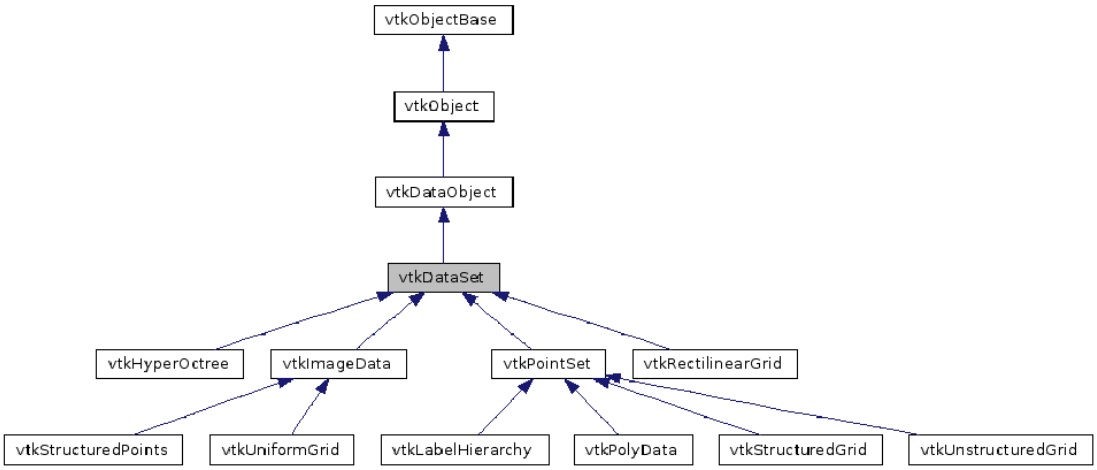
컴퓨터 시뮬레이션은 물리, 화학, 생물, 기계공학 등 자연과학·공학뿐만 아니라 인문·사회과학 분야에서도 많이 적용되고 있는 연구방법으로 계산(computation)과 가시화(visualization)라는 두 가지 기술에 의해 발전되고 있다. 고성능 컴퓨터(HPC: High Performance Computer)를 포함한 계산 기술의 빠른 발전은 기존에 적용하기 어려웠던 문제를 빠른 시간 내에 해결하는 것을 가능하게 하였으며 새로운 현상의 발견도 가능하게 함으로써 컴퓨터를 이용한 시뮬레이션이 가속화되고 있다. 시뮬레이션 결과는 연구자가 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]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[4,5]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. GLOVE에서는 VTK 데이터 형식을 기반으로 로터 동역학에 최적화된 데이터 형식을 사용하고 있는데, 본 기술문서에서는 이러한 GLVOE의 데이터 형식에 대해 살펴본다.

2. VTK 데이터 구조

VTK 데이터는 데이터 구조와 데이터 속성으로 나뉘는데 데이터 구조는 3차원(또는 2차원) 필드에서 각 데이터 요소(element)의 topological, geometrical 위치(즉, 격자 구조)를 정의한다. 데이터 속성은 정의된 데이터 구조에서 각 요소별로 저장되는 값으로 vector, scalar, coordinates 등의 값들이 저장된다.

가. 기본 데이터 구조

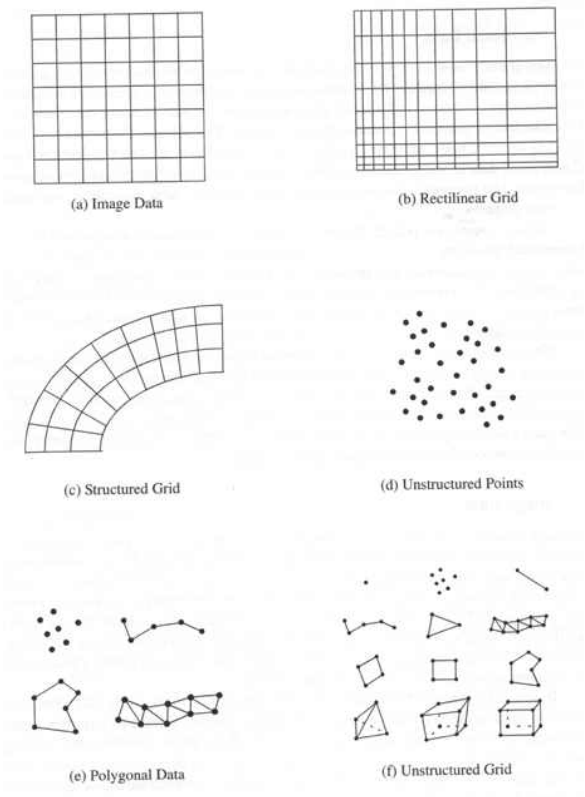
대부분의 VTK 데이터 세트는 vtkDataSet 클래스에서 파생된다. 여기에서는 우선 vtkDataSet 클래스의 슈퍼클래스인 vtkObject부터 차례로 아래로 내려가면서 설명한다.



[그림 2-1] vtkDataSet 클래스 계층구조

1) vtkDataSet

vtkDataSet 클래스는 VTK 데이터세트 객체들에 대해 공통된 인터페이스를 제공하기 위한 추상 클래스로 vtkDataObject 클래스에서 파생되었으며 모든 VTK 데이터 세트 클래스들은 이 클래스에서 파생된다. 또한 이 클래스는 각 데이터 타입에 대해 중심점, 경계상자, 대표 길이 등의 각종 정보를 전달하기 위한 공통된 메소드를 제공한다. 이 클래스에서 파생되는 대표적인 데이터 세트 클래스에는 vtkImageData, vtkRectilinearGrid, vtkStructuredGrid, vtkUnstructuredGrid 등이 있다. 각 데이터 타입별 구조는 아래 그림과 같다.



[그림 2-2] 데이터 타입별 격자 구조

2) 슈퍼클래스들

가) vtkObject

vtkObject 클래스는 대부분의 VTK 데이터 클래스의 기초가 되는 추상 클래스로 거의 모든 VTK 데이터 클래스들이 이 클래스 또는 이 클래스의 서브 클래스들로 부터 파생되었다. 이 클래스에는 VTK 데이터 클래스들에서 기본으로 사용되는 클래스 생성, 소멸, 디버그, 프린트, reference count 등에 대한 멤버 및 메소드가 정의되어 있다. 여기서 reference count랑 현재 해당 object를 참조하는 object가 얼마나 있는가를 count하는 역할을 한다. 즉, 두개의 object가 참조를 한다면 이 reference count는 2이다. 만약 어떤 object도 참조하지 않아 count가 0이 되면 해당 object는 소멸된다.

나) vtkDataObject

vtkDataObject 클래스는 VTK에서 사용하는 모든 가시화 데이터 클래스의 슈퍼클래스로 가시화 표현을 위한 일반화된 정보를 담고 있다.

3) 정규격자 데이터 구조

가) vtkImageData

vtkImageData는 uniform rectilinear grid를 위한 데이터 세트 클래스로 1D 배열, 2D 이미지, 3D 볼륨을 표현한다. 이 데이터 세트 클래스는 geometry와 topology 양쪽에 있어 규칙적으로 dimension, width, height, depth, origin (lower-left corner), interpoint space 등이 정의되면 실제 element의 위치가 암시적으로 결정된다. 만약 이 데이터 세트가 2차원이면 이미지로 간주되고 vtkPixel 셀 타입으로 구성된다. 만약 3차원으로 정의되면 이 데이터 세트는 볼륨으로 간주되고 vtkVoxel 셀 타입으로 구성된다.

나) vtkUniformGrid

vtkUniformGrid는 vtkImageData의 서브 클래스로 blanking 기능을 추가한 데이터 세트 타입이다. 이 클래스는 정의된 각 mesh point에 blanking 기능을 할당하여 이 point를 사용할 것인지 안사용할 것인지를 끄고 켜는 기능을 제공한다. 이는 Chimera grid 등을 사용할 때 유용할 것으로 보인다.

다) vtkStructuredPoints

vtkStructuredPoints 역시 vtkImageData의 서브 클래스로 실제 데이터의 범위(extent)가 업데이트 범위 (update extent)와 반드시 일치해야만 하는 클래스이다. vtkImageData에서는 실제 데이터 범위가 업데이트 범위보다 커도 문제가 없다. 또 vtkImageData는 origin이 (0, 0, 0) 포인트로 정의되는 반면 이 클래스에서는 첫번째 포인트가 origin으로 정의되는 차이점이 있다.

라) vtkRectilinearGrid

vtkRectilinearGrid는 topologically 규칙적이고 geometrically 반규칙적인(semi-regular) 구조로 각 포인트의 좌표는 X, Y, Z 축을 따라 불규칙적인 간격으로 저장되는 3개의 벡터 배열로 정의된다. 따라서 각 포인트는 항상 좌표축에 평행하고 배치된다. 이 벡터 배열은 XCoordinates, YCoordinates, ZCoordinates 배열에 저장된다.

4) 포인트 기반 데이터 구조

가) vtkPointSet

vtkPointSet는 격자 구조의 포인트 좌표를 명시적으로 정의하는 데이터 클래스들을 위한 추상 클래스이다.

나) vtkStructuredGrid

vtkStructuredGrid는 topology는 규칙적이지만 geometry는 불규칙적인 포인트의 배열이다. 이 데이터의 셀은 3차원의 경우 vtkHexahedron으로 구성되고 2D의 경우에는 vtkQuad로 구성된다.

다) vtkPolyData

vtkPolyData는 꼭짓점, 선, 다각형, 삼각형 스트립 등을 정의하기 위한 데이터 클래스이다. 이 클래스 역시 각 꼭짓점에 scalar나 vector 같은 데이터 속성이 붙을 수 있다. 이 클래스의 실제 셀 타입은 vtkVertex, vtkPolyVertex, vtkLine, vtkPolyLine, vtkTriangle, vtkQuad, vtkPolygon 그리고 vtkTriangleStrip이 될 수 있는데, 3차원 셀(vtkCell3D)은 셀 타입으로 지정할 수 없다.

라) vtkUnstructuredGrid

vtkUnstructuredGrid는 완전히 불규칙적으로 구성된 데이터 세트를 위한 데이터 타입으로 데이터 내의 포인트는 물론 셀 정보와 각 셀의 연결관계까지도 명시적으로 지정해야 한다. 우선 포인트는 이 클래스의 슈퍼클래스인 vtkPointSet으로 표현되고 셀 정보는 vtkCellArray, vtkCellTypes, vtkCellLinks 등의 조합으로 구성된다. 이 클래스는 많은 부분에서 vtkPolyData와 유사한데, vtkPolyData는 셀이 모두 2차원으로 정의되어야 하는 반면 vtkUnstructuredGrid는 3차원(vtkCell3D)으로도 정의될 수 있다는 차이점이 있다.

5) 트리 기반 데이터 구조

가) vtkHyperOctree

vtkHyperOctree는 전체 데이터를 octree로 관리하는 데이터 구조이다. 이 클래스를 사용하기 위해서는 데이터 세트의 element 개수가 2n 이어야 한다. 이 데

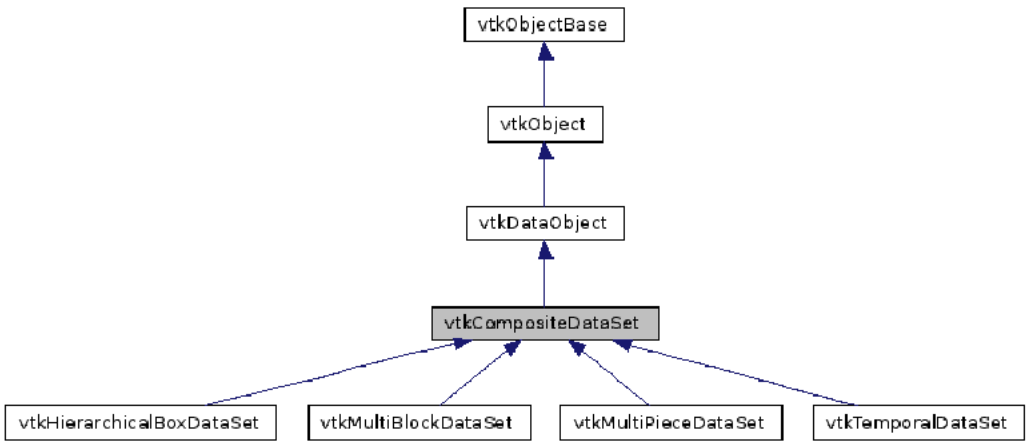
이터 세트에서 각 노드는 포인트가 아닌 셀이 저장된다. 이것은 VTK의 다른 데이터 구조와 차별화되는 점이다. Geometry는 반드시 hypercube이어야 할 필요는 없고 직사각형 모양이어도 된다. 속성은 트리의 단말노드에 저장이 되고 LOD 목적으로 자식 노드들의 속성값의 평균값을 이용해서 중간노드의 사용이 가능하다. 이 데이터 구조는 geometry가 sparse한 경우에 메모리를 효율적으로 사용할 수 있게 해준다. 또한, LOD 기능을 사용하면 전체 데이터 세트 중 일부분을 빠르게 cull할 수 있다.

나) vtkLabelHierarchy

vtkLabelHierarchy는 label의 octree를 저장한 클래스로 VTK 버전 5.4에서 부터 구현되었다. 이 클래스는 각 포인트들을 label로 관리하면서 계층트리를 생성하는데, 중요한 부분은 빠르게 렌더링되도록 상위 노드에 배치하고 덜 중요한 부분은 하위 노드에 배치한다.

나. 혼합 (composited) 데이터 구조

데이터 구조는 VTK 5.0에서 처음으로 도입된 데이터 구조로써 다른 VTK 데이터 구조들의 조합으로 이루어지는 데이터 구조를 의미한다. 이러한 구조는 복잡한 데이터 구조를 정의할 때 유용한데, 여러 개의 단순한 데이터 구조들의 조합으로 하나의 복잡한 데이터 구조를 만드는 것이다. 이러한 데이터 구조에는 멀티 블록 (multi-block) 데이터 구조와 AMR(adaptive mesh refinement) 구조를 들 수 있다. AMR이란 수치 시뮬레이션을 통해 특정 영역의 물리적 도메인을 자동으로 상세하게 나타내는 기술을 말한다[6]. 혼합 데이터 구조의 클래스 계층구조는 그림과 같다.



[그림 2-3] vtkCompositeDataSet 클래스 계층 구조

1) vtkCompositeDataSet

vtkCompositeDataSet는 멀티 블록이나 AMR과 같은 혼합 데이터 구조를 위한 추상 클래스이다. 이 클래스는 vtkCompositeDataIterator iterator를 통해 데이터 세트에 접근할 수 있도록 하는 공통된 인터페이스를 제공한다. 또한 이 클래스는 데이터를 트리 구조로 저장할 수 있도록 하는 데이터 구조를 제공하는데, 이 클래스를 상속받는 서브 클래스들에서는 이를 기반으로 각 클래스에 적합한 트리를 생성한다. 여기서 트리 구조 자체는 vtkCompositeDataSetInternals 타입을 이용한 linked list로 저장되고 vtkCompositeDataIterator를 통해 트리에 대한 탐색이 이루어진다. 일반적인 VTK 필터들은 이 클래스와 서브클래스들을 처리하지 못하므로 vtkCompositeDataSet를 위해 특별히 설계된 클래스들을 사용해야 한다. 이러한 클래스에는 vtkCompositeDataGeomertyFilter와 vtkCompositeDataPipeline 등이 있다.

2) vtkMultiBlockDataSet

vtkMultiBlockDataSet 클래스는 여러 개의 블록으로 이루어진 데이터 세트를 관리하기 위한 클래스이다. 각 블록들은 vtkDataSet나 vtkCompositeDateSet의 서브 클래스들이어야 한다. 특히 vtkMultiBlockDataSet를 블록으로 하는 경우에는 각 블록 안에 또 다른 멀티 블록이 저장되는 계층 구조의 구현이 가능해지고 이를 통해서 AMR의 구현 역시 가능해진다. 각 서브 블록들은 여러 프로세서에 분산되어 병렬로 처리하는 것도 가능하다.

3) vtkMultiPieceDataSet

vtkMultiPieceDataSet 클래스는 여러 조각으로 구성된 데이터를 하나의 데이터로 합쳐서 관리하기 위한 클래스이다. 이 클래스는 vtkMultiBlockDataSet 클래스와 매우 유사한데, 차이점은 vtkMultiBlockDataSet는 여러 블록으로 된 데이터를 계층적으로 관리하기 위한 클래스이고 각 블록이 여러 프로세서에 각각 할당되어 처리될 수 있는 반면에, vtkMultiPieceDataSet는 여러 조각(혹은 블록)으로 된 데이터를 하나의 데이터로 합치기 위한 클래스이고 각 조각들이 여러 프로세서로 분산되어 처리되기 보다는 전체 데이터가 하나의 프로세서에 할당되어 처리된다. 즉, 여러 조각으로 나뉘어 하나의 프로세서에서 한 알고리즘으로 처리되

기 어려운 데이터를 하나의 데이터 구조로 합쳐주는 역할을 한다. 또, vtkMultiBlockDataSet는 혼합 데이터 구조를 블록으로 받을 수 있는 반면 vtkMultiPieceDataSet는 이를 블록으로 받을 수 없다는 점도 다르다. VTK나 paraview에서는 이러한 정보를 이용해서 각 조각들을 합친 데이터의 전체 영역이나 크기 등의 정보를 표출해주고 ghost level 등도 생성해준다. vtkMultiPieceDataSet는 그 자체만으로도 사용이 가능하겠지만 vtkMultiBlockDataSet나 vtkHierarchicalBoxDataSet의 한 노드로 함께 사용하는 것이 더욱 좋다.

4) vtkHierarchicalBoxDataSet

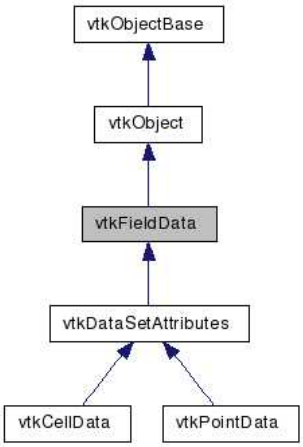
vtkHierarchicalBoxDataSet 클래스는 uniform grids의 계층 구조 데이터 세트이다. 이 클래스는 AMR(Adoptive mesh refinement)를 위해 디자인되었다. 이 데이터 구조는 여러 레벨로 구성되는데 각 레벨 마다 데이터 세트가 저장된다. 여기서 데이터 세트는 vtkUniformGrid이어야만 한다. 각 레벨의 데이터 세트에는 vtkAMRBox가 저장되는데, 여기에는 각 레벨을 구성하는 데이터 세트의 영역이 경계상자 형태로 저장된다.

5) vtkTemporalDataSet

vtkTemporalDataSet는 여러 타입 스텝으로 구성된 데이터 세트를 위한 데이터 구조이다. 이 데이터 세트는 각 타입 스텝들을 계층트리의 서로 다른 노드에 저장한다.

다. 데이터 속성

VTK에서 데이터 속성은 위에서 정의한 각 데이터 구조의 포인트나 셀 등에 할당되는 스칼라, 또는 벡터 값 등을 저장한다. 이러한 데이터 속성은 추상적인 형태의 데이터 배열들을 저장하는 vtkFieldData를 부모로, 보다 실제적인 데이터 속성을 저장하는 vtkDataSetAttributes와 이의 서브 클래스들인 vtkCellData, vtkPointData로 구성된다. 클래스 계층구조는 그림 2-4와 같다.



[그림 2-4] VTK 데이터 속성 클래스 계층구조

1) vtkFieldData

vtkFieldData는 여러 종류의 데이터 배열들을 저장하고 관리하기 위한 클래스이다. 이 클래스에서는 필드를 m×n의 행렬을 이용해서 관리하는데, 여기서 m은 튜플(tuple)의 개수이고 n은 콤포넌트(component)의 개수이다. 필드는 한 개 또는 여러 개의 데이터 배열의 집합으로 간주하는데, 여기서 각 배열들은 서로 다른 형식으로 저장될 수 있다. (int, float, double, char 등등) 또 각 배열들을 서로 다른 개수의 콤포넌트로 이루어질 수 있다. 각 데이터 배열은 튜플의 개수인 m의 길이를 갖는데, 이는 일반적으로 데이터 구조 안의 포인트나 셀의 개수이다. 또한, 각 배열은 반드시 character string 형식의 이름을 가지고 있어야 한다.

2) vtkDataSetAttributes

vtkDataSetAttributes는 데이터 속성을 저장하기 위한 클래스이다. 여기서 속성에는 스칼라, 벡터, 노말 벡터, 텍스쳐 좌표, 텐서 등등이 있다. 이러한 속성 정보는 vtkFieldData에 각 데이터 배열의 형태(속성)이나 현재 활성화된 배열이 무엇인지 등의 정보를 더해준다. 다시 말해서 현재 활성화된 스칼라 배열 또는 벡터 배열이 무엇인지 알 수 있고 그 배열을 사용할 수 있도록 해준다. 덧붙여서 vtkDataSetAttributes는 각 배열들에서 값들을 가져오는 방법을 설정해줄 수 있도록 하는데, 여기에는 전체 배열을 통째로 넘기기, 일부분만은 복사해서 넘기기, 그리고 선택된 튜플들을 보간(interpolation)해서 넘기기 등이 있다.

3) vtkCellData

vtkCellData는 셀로 구성된 데이터 구조의 속성을 저장하고 관리하기 위한 클래스이다. 여기서 속성은 스칼라, 벡터, 노말 벡터 등을 말한다. vtkDataSetAttributes를 상속받아 하나의 셀에서 데이터를 복사하거나 셀의 보간 가중치를 이용해서 값을 보간하여 가져오기 등을 지원한다.

4) vtkPointData

vtkPointData는 포인트로 구성된 데이터 구조의 속성을 저장하고 관리하기 위한 클래스이다. 여기서 속성은 스칼라, 벡터, 노말 벡터 등을 말한다. 대부분의 기능은 vtkDataSetAttributes에서 처리된다.

3. 필드 데이터

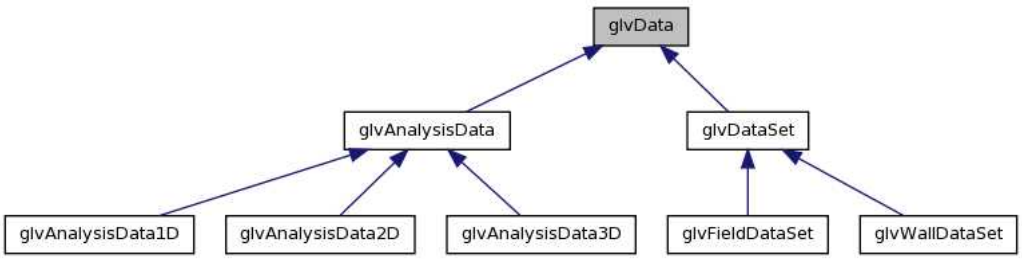
GLOVE에서 다루는 데이터는 크게 두 가지 종류로 나뉜다. 하나는 구조화된 격자나 비구조화된 격자 등 격자 구조 기반에서 정의되는 필드(field) 데이터이고 다른 하나는 일반 1차원, 2차원 등의 배열로 정의되는 분석(analysis) 데이터이다. 이 두 데이터 종류는 서로 다른 특성을 지니고 있기 때문에 따로 생각해서 디자인해야 한다. GLOVE의 필드 데이터는 기본적으로 VTK 데이터 구조를 기반으로 로터 데이터 처리에 최적화하여 디자인해야 한다. 일반적인 로터 데이터의 특징을 보면 GLOVE의 필드 데이터는 아래의 특징을 만족해야 한다.

1. 멀티 블록 데이터 구조를 지원해야 한다.
2. 시변환(time-varying) 데이터를 지원해야 한다.
3. 벡터 및 스칼라 변수를 지원해야 한다.
4. Chimera Grid 지원

여기서 3번은 VTK에서 기본적으로 지원하는 특징이고 4번 역시 구조화된 격자에 한해서 VTK에서 지원하는 부분이다. 1번과 2번은 VTK의 기본 데이터 구조에서는 지원하지 못하지만 혼합 데이터 구조에서는 지원한다. 따라서 GLOVE에서는 VTK의 혼합 데이터 구조를 기반으로 설계, 구현하며 데이터가 구조화된 격자인 경우에 한해서 Chimera Grid를 지원한다.

가. 기본 데이터 구조

GLOVE의 데이터 구조는 아래와 같은 클래스 계층 구조를 가지고 있다.



[그림 3-1] GLOVE 데이터 클래스 계층 구조

여기서 최상위 클래스인 glvData는 현재는 little endian과 big endian을 구분하기 위한 용도일 뿐 다른 용도는 없다. 이는 향후 모든 GLOVE 데이터에서 공통적으로 필요한 기능이 있을 때 구연하기 위해 예약해둔 클래스이기 때문이다. GLOVE 데이터의 실질적인 최상위 클래스는 glvDataSet이다. glvDataSet은 glvFieldDataSet과 glvWallDataSet에서 필요로 하는 대부분의 기능들이 구현되어 있다. 이는 필드 데이터(field data)나 벽면 데이터(wall data)나 모두 똑같이 VTK 데이터를 기반으로 하고 있기 때문에 공통되는 변수 및 함수가 많기 때문이다. glvWallDataSet은 CFD나 구조해석 분야에서 중요하게 다루는 물체의 표면을 저장하는 데이터 구조이다. 이 구조는 필드를 가시화할 때는 필요없는 표면을 디스플레이할 때 함께 가시화할 변수를 지정할 수 있도록 했다. 이를 위해 SetDisplayVariable() 함수를 제공한다. glvFieldDataSet은 물체의 표면을 둘러싸는 필드를 위한 데이터 구조이다. 일반적으로 벽면 데이터는 타임 스텝의 변화에 따라 계속 변화하는 경우가 많은데 필드 데이터는 변화하지 않고 고정되어 있는 경우가 많다. 이때, 움직이는 벽면 데이터의 격자와 필드 데이터의 격자가 겹치는 경우가 많은데, 이렇게 겹치는 격자의 경우 필드 데이터에서 격자를 무효화해야 한다. 이를 위해 필드 데이터 구조에서는 각 격자마다 무효화할지 안할지를 결정하는 배열을 저장하도록 했고 이를 SetBlankField() 함수로 지정할 수 있도록 했다. 또, EnableBlanking()과 DisableBlanking() 함수를 통해 이를 활성화/비활성화 할 수 있도록 했다. 이러한 격자 구조를 키메라 격자구조(Chimera Grid)라고 한다.

1) 기반 VTK 데이터 구조

GLOVE의 데이터 구조는 VTK 데이터 구조를 기반으로 한다. 그런데, 앞장에서 살펴보았듯이 VTK 데이터 구조는 매우 복잡하다. 이를 일반적인 사용자가 모두 이해하고 사용하도록 하는 것은 무리가 있을 것이다. 따라서 GLOVE에서는 VTK 데이터 구조를 사용하되, 이를 추상화하여 사용자는 VTK의 어떤 데이터 구조를 사용하는지 몰라도 GLOVE를 사용할 수 있도록 했다. GLOVE에서는 VTK와 달리 기본적으로 멀티 블록 구조 및 시변환 데이터를 지원한다. 이를 위해 GLOVE의 기본 데이터 구조인 glvDataSet은 vtkTemporalDataSet을 포함한다. vtkTemporalDataSet은 VTK에서 시변환 데이터를 지원하기 위해 만든 데이터 구조로 정적(steady) 데이터인 경우는 내부 변수인 numberOfTimeSteps를 1로 하면 되기 때문에 이 데이터 형식을 이용한다고 해도 정적 데이터를 처리하는 데에 아무런 문제가 없다. vtkTemporalDataSet은 각 타임 스텝별로 데이터를 저장하는데 이

는 GetTimeStep() 함수로 접근이 가능하다. 또, 새로운 타입 스템을 저장할 때는 SetTimeStep() 함수를 이용하면 된다. 이때, 각 타임 스텝은 vtkDataSet과 vtkCompositeDataSet의 상위 클래스인 vtkDataObject 형식이므로 VTK에서 지원하는 모든 데이터 형식이 다 들어갈 수 있다. 그러나 GLOVE에서는 멀티 블록 구조를 기본으로 제공해야 하고 혹시 데이터가 멀티 블록 구조가 아니라고 하더라도 블록 개수가 1인 멀티 블록으로 저장하면 되기 때문에 각 타임 스텝은 vtkMultiBlockDataSet으로 저장되어야만 되도록 했다. 즉, GLOVE의 기본 데이터 구조는 각 타임 스텝이 vtkMultiBlockDataSet인 vtkTemproalDataSet인 것이다. GLOVE의 데이터 구조를 실제 데이터 구조에 최적화된 VTK 데이터 구조를 사용하지 않고 무조건 이와 같이 시변환 데이터나 멀티 블록 데이터 구조를 사용하는 것은 최대한 GLOVE의 데이터 구조를 단순화하기 위함이다.

2) 수치 데이터 저장

수치 데이터란 데이터 구조의 각 요소(격자 또는 셀) 마다 할당되는 수치값으로 일반적으로 스칼라나 벡터로 표현되는 데이터를 말한다. GLOVE에서는 VTK 데이터 구조를 사용하는 만큼 VTK의 속성 데이터를 이용해서 이러한 수치 데이터를 저장한다. 따라서 데이터 구조의 각 요소마다 다양한 종류, 개수의 수치 데이터가 유연하게 저장될 수 있다. 여기서 문제가 발생할 수 있는데 한 데이터에 여러 개의 벡터 또는 스칼라값이 존재할 때 많은 변수들 중 어느 변수를 이용해서 각종 가시화 알고리즘(iso-surface, streamline 등)을 수행할 것인지를 정하는 문제이다. VTK에서는 이 문제를 해결하기 위해 vtkDataSetAttributes 클래스에 SetActiveScalars 또는 SetActiveVectors 등의 함수를 제공해서 여러 스칼라 또는 벡터들 중 한 변수만을 활성화하도록 했다. 즉, 가시화 알고리즘을 실행하면서 저장된 여러 스칼라 또는 벡터 변수 중 하나를 명확하게 지정하는 방법이 아닌 데이터 자체에서 변수를 활성화하는 방법을 택했다. 이는 VTK의 정책상 알고리즘 클래스로의 데이터 접근을 제한했기 때문인 것으로 보인다. GLOVE의 데이터 구조도 역시 기본적으로는 VTK의 정책을 따른다. 즉, 데이터에서 활성화한 변수를 이용해서 렌더러(renderer)가 렌더링 알고리즘을 수행하도록 하는 방법으로 데이터와 렌더러 분리하는 것이다. 이를 위해 GLOVE 데이터 클래스에서는 SetActiveScalar(), SetActiveVector() 등의 함수를 지원한다. 특이할만한 점은 VTK에서는 현재 활성화된 스칼라나 벡터가 무엇인지를 알 수 있는 방법이 없기 때문에 사용에 불편함이 많았는데, GLOVE에서는 GetActiveScalarName()이나

GetActiveVectorName() 등의 함수를 지원하여 현재 활성화된 스칼라 또는 벡터를 쉽게 알 수 있도록 했다. 이와 같이 GLOVE는 VTK의 데이터 접근 정책을 따르지만 사용자 편의를 위해 렌더러 등 외부에서 직접 데이터를 접근하는 것도 가능하도록 했다. 이를 위해 GLOVE의 데이터 클래스는 현재 데이터 내부의 스칼라 및 벡터 변수들의 목록 등의 정보를 가져올 수 있도록 GetNumberOfVariables(), GetNubmerOfVectors(), GetNumberOfScalars(), GetVectorNames(), GetScalarNames() 등의 함수를 지원한다.

VTK에서 수치 데이터를 저장하는 vtkDataSetAttributes 클래스는 수치 데이터를 스칼라, 벡터, 텐서 등의 타입으로 구분하지만 vtkDataSetAttributes의 부모 클래스인 vtkFieldData는 이러한 타입을 구분하지 못한다. 그런데 문제는 실제 데이터를 저장하는 것은 vtkFieldData이지 vtkDataSetAttributes가 아니라는 것이다. 즉, VTK에서 실제로 수치 데이터를 저장할 때는 스칼라나 벡터 등을 구분하지 않고 그냥 vtkDataArray 형식으로 저장한 뒤 vtkFieldData에서 추가한다. 그리고 실제로 사용할 때는 각 vtkDataArray의 콤포넌트 개수를 파악해서 1개면 스칼라, 2개 이상이면 벡터 이런 식으로 사용하는 것이다. 따라서 vtkDataSetAttributes는 등록된 전체 vtkDataArray 중에서 어느 것이 스칼라이고 어느 것이 벡터인지 알지 못한다. 이는 각 배열의 콤포넌트 개수를 통해 유저가 직접 파악해야 하는 것이다. 이를 원활히 관리하도록 하기 위해서 vtkDataSetAttributes에는 SetActiveScalars()류의 함수를 만들었다. 따라서 glvDataSet에서도 이를 관리하기 위해 mAttributeTypes라는 배열을 두고 등록된 각 변수 마다 속성 형식이 무엇인지 (스칼라 또는 벡터) 저장하도록 했다. 이 자료구조를 통해 전체적인 변수 데이터의 타입 관리가 가능해진다. 또 이 자료구조를 이용해서 관련된 여러 함수들 (GetScalarName(), GetNumberOfScalars() 등)의 생성이 가능했다.

나. 로터 데이터

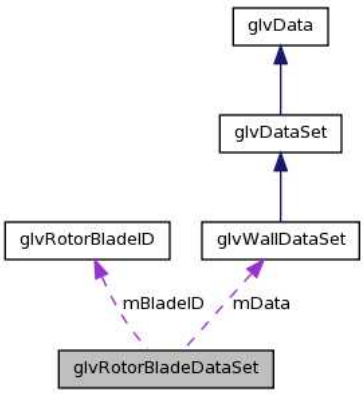
1) 로터 데이터 구조

GLOVE의 기본 데이터 구조는 일반적인 CFD나 구조해석 분야의 데이터는 모두 처리할 수 있도록 했다. 그러나 GLOVE의 주요 응용분야인 로터 동역학 분야에서 이를 이용하기 위해서는 사용자가 많은 부분을 이해해야만 하기 때문에 여전히 사용에 어려움이 많다. 이러한 이유로 GLOVE에서는 로터 동역학 데이터에 최적화된 로터 데이터 구조로 glvRotorBladeDataSet과 glvRotorFieldDataSet을

제공한다. glvRotorBladeDataSet은 glvWallDataSet을 기반으로 생성된 클래스이고 glvRotorFieldDataSet은 glvFieldDataSet을 기반으로 생성된 클래스이다.

가) glvRotorBladeDataSet

glvRotorBladeDataSet은 로터 블레이드를 저장하고 관리하기 위한 클래스로 glvWallDataSet을 기반으로 생성되었다. glvWallDataSet은 모든 벽면 정보를 통합해서 하나로 관리하지만 glvRotorBladeDataSet은 내부 변수로 가지는 glvWallDataSet을 블레이드 단위로 관리한다. 즉, 내부에 현재 glvRotorBladeDataSet이 가지고 있는 블레이드의 개수, 한 블레이드가 가지는 glvWallDataSet의 블록 개수 등의 정보를 저장해서 glvRotorBladeDataSet에서는 블레이드의 ID를 이용해서 블레이드 단위로 데이터를 사용할 수 있도록 한 것이다. 이때, 각 블레이드는 모두 같은 수의 glvWallDataSet 블록 및 타임 스텝으로 이루어져야 하며 격자의 개수나 각 격자에 할당된 속성의 종류, 개수 등이 모두 일치해야 한다. 또한 glvRotorBladeDataSet은 로터 블레이드의 특성 상 블레이드의 가장 중요한 정보인 압력(pressure) 정보를 직접 접근할 수 있는 함수들을 제공한다. 이때, 압력 정보는 로터 블레이드의 ID 및 블록 번호, (i, j, k)로 표시되는 블록 내부의 격자 정보 등으로도 접근이 가능하고 글로벌 좌표계에서의 (x, y, z)의 좌표값으로도 접근이 가능하다. 이러한 접근 방법은 블레이드 내의 압력 분포 그래프를 생성할 때 유용하게 사용될 수 있다.



[그림 3-2] glvRotorBladeDataSet 협동 다이어그램

나) glvRotorFieldDataSet

glvRotorFieldDataSet은 로터 블레이드를 둘러싸는 필드를 저장하고 관리하기 위

한 클래스로 glvFieldDataSet을 기반으로 생성되었다. 이 클래스에는 로터 동역학에서 매우 핵심적인 역할을 하는 여러 변수들을 반드시 포함해야 하는데, 여기에는 압력(pressure), 밀도(density) 등의 스칼라 변수와 속도(velocity) 벡터가 있다. 로터 동역학의 후처리에는 이외에도 속도 크기(velocity magnitude), vorticity 크기(vorticity magnitude), Q-criteria[7]를 위한 Q 값 등의 정보가 필요한데, 이러한 정보들은 모두 속도 벡터를 통해 유도가 가능하다. 따라서 glvRotorFieldDataSet은 내부의 glvFieldDataSet에 이러한 정보가 있는지를 파악하고 만약 없으면 속도 벡터를 통해 이러한 변수들을 실시간으로 생성해낸다. 물론, 가장 중요한 속도 벡터가 없으면 경고 메시지를 출력하고 추가적인 정보 생산을 중단한다. 속도(velocity) 벡터를 (u, v, w)라고 할 때 vorticity 크기는 다음과 같다.

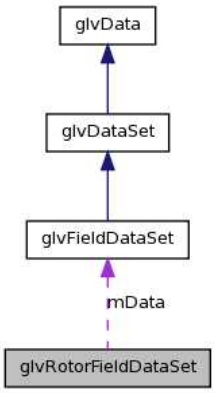
$$|\Omega| = \sqrt{\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial z} - \frac{\partial w}{\partial y}\right)^2}$$

[수식 3-1] Vorticity 계산 식

이를 위해 각 격자에서의 속도 벡터에 대한 편미분이 계산되어야 하는데, 이는 vtkCell 클래스의 Derivative() 함수를 이용해서 계산이 가능하다. Q-Criteria는 vortex의 영역을 정의하는 방법 중 하나로 [7]에서 처음 제안한 방법이다. 이 방법은 전체 유동장에서 vorticity가 strain보다 큰 부분을 vortex라고 정의한다. 계산식은 다음과 같다.

$$Q = \Omega^2 - S^2 = -\frac{1}{2}\left(\frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + \frac{\partial u}{\partial z}\frac{\partial w}{\partial x} + \frac{\partial v}{\partial z}\frac{\partial w}{\partial y}\right)$$

[수식 3-2] Q-Criteria 계산 식



[그림 3-3] glvRotorFieldDataSet 협동 다이어그램

이 값 역시 vorticity를 계산할 때 Derivative() 함수를 이용해서 함께 계산할 수 있다.

2) 키메라 격자 구조

매우 복잡한 유동 문제를 풀 때 전체 필드를 단 하나의 연속된 격자계(구조화된 격자계 또는 비구조화된 격자계 등)로 표현하는 것은 매우 어렵다. 이러한 문제를 해결하기 위해 일반적으로 많이 사용하는 방법은 서로 다른 구조적 성질을 가지는 여러 종류의 격자계를 겹쳐서 구성하는 것이다. 이렇게 구성하는 방법을 키메라 격자구조(Chimera grid) 방법 또는 오버세트 격자구조(overset grid) 방법이라고 한다. 여러 격자가 겹쳐져 있으므로 각 셀 마다 겹치는 부분이 있게 되는데, 이러한 부분은 각 셀 또는 포인트의 가시성(visibility)을 on/off 하는 방법으로 해결한다. 이때, off 되어 사용하지 않는 포인트를 구멍 포인트(hole point)라고 한다. 또, 각 격자가 서로 연결되는 경계 부분의 값을 얻을 때는 서로 인접한 블록의 정보를 가지고 있으면서 인접한 셀의 값을 인접한 블록과 교환하고 그 값들을 보간하는 방법을 사용해야 한다.

일반적으로 로터 데이터에서도 이러한 키메라 격자구조를 많이 사용하는데, 전체 필드 데이터는 고정된 멀티 블록 데이터 구조로 구성되고 블레이드 데이터는 움직이는 (즉, 시변환) 멀티 블록 데이터 구조로 구성된다. 이때, 필드 데이터는 고정되고 블레이드 데이터는 움직이기 때문에 매 프레임마다 블레이드 데이터와 필드 데이터의 겹치는 부분이 달라지게된다. 로터 데이터에서는 필드 데이터의 각 격자에 가시성 정보를 Blanking 필드에 저장하여 어떤 포인트가 구멍 포인트

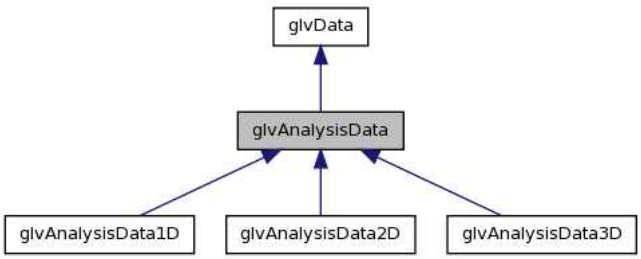
인지를 알 수 있도록 했다. VTK에서는 vtkStructuredGrid와 vtkUniformGrid의 두 격자구조에서 Blanking 포인트 (또는 셀)을 지원하므로 키메라 격자의 구현이 어렵지 않다. 그러나 만약 구조화된 격자나 균등 격자(uniform grid)가 아닌 다른 종류의 격자구조에서 키메라 격자 구조의 구현이 필요하다면 Blanking 포인트 기능을 직접 구현해야 할 필요가 생길 수도 있다.

4. 분석 데이터

분석 데이터는 GLOVE에서 그래프로 표현할 수 있는 일종의 통계 데이터를 말한다. 이러한 데이터는 그 특성 상 1~3차원의 배열로 구성된다. 로터 데이터를 위한 분석 데이터는 크게 두 종류가 있다. 하나는 곡선(curve) 그래프로 표현되는 데이터이고 나머지 하나는 원반(disk) 형태로 표현되는 데이터이다. 곡선 그래프로 표현되는 데이터는 1차원 배열로 해결이 가능하고 원반 형태로 표현되는 데이터는 2차원 배열로 해결이 가능하다. 이 장에서는 먼저 일반적인 분석 데이터의 구조에 대해 설명하고 그 뒤에 로터 동역학 데이터에 최적화된 분석 데이터 설계 및 구조에 대해 설명한다.

가. 기본 구조

기본적으로 분석 데이터는 단순한 1~3차원의 배열이다. 그러나 분석 데이터를 단순한 다차원 배열로 생성하면 사용하기에 불편한 점이 많다. 따라서 GLOVE에서는 glvAnalysisData 및 glvAnalysisData[1~3]D의 새로운 클래스를 생성했다. glvAnalysisData[1~3]D는 모두 glvAnalysisData의 서브 클래스이다. 대부분의 함수는 glvAnalysisData에서 구현되었고 특정 차원을 위한 클래스들은 각 차원별로 특별히 구현되어야 하는 함수만이 구현되어 있다. 편의상 모든 내부 변수의 형식은 double로 하였다.



[그림 4-1] glvAnalysisData 클래스 계층구조

분석 데이터가 배열의 형태를 가지고는 있지만 실질적으로는 구조화된 격자이다. 즉, 2차원 배열일 경우 x축과 y축의 단위 및 격자의 실제 간격이 서로 다른 것이다. 따라서 glvAnalysisData에서는 각 축의 인덱스 및 이름을 저장할 수 있도록 했다. 이때, 인덱스의 개수는 실제 데이터의 해당 축의 개수와 같아야 한다.

즉, x축의 인덱스 개수는 너비(width)와 같아야 하고 y축의 인덱스 개수는 높이(height)와 같아야 한다. 각 축의 이름은 char string으로 저장되고 만약 이름이 없을 경우에는 NULL이 저장된다. glvAnalysisData는 이러한 각 축의 인덱스를 하나씩 받을 수 있도록 했다. 이는 GetXIndex(), GetYIndex(), GetZIndex() 함수로 가능하다. 실제 분석 데이터를 얻기 위해서는 GetData() 함수를 사용하면 된다. 이 경우 GetData() 함수는 width × height × depth 크기의 1차원 배열을 돌려주게 되고 사용자가 적절히 3차원 배열처럼 사용해야 한다.

모든 glvAnalysisData의 서브 클래스들은 mData[4]에 각 축의 인덱스와 분석 데이터를 저장한다. 이때, 1차원 데이터의 경우에는 mData[0]에 x축 정보를 저장하고 mData[1]에 분석 데이터를 저장한다. 2차원 데이터의 경우에는 mData[0]에 x축 정보를, mData[1]에 y축 정보를 저장하고 분석 데이터는 mData[2]로 1만큼 밀려서 저장된다. 즉, 분석 데이터는 데이터의 차원에 따라 저장되는 위치가 달라지는 것이다. 이러한 문제를 해결하기 위해 glvAnalysisData는 내부적으로 GetDataIndex() 함수를 제공한다. 이 함수는 실제 분석 데이터가 저장된 인덱스가 무엇인지를 돌려주는 함수로 도든 glvAnalysisData의 서브 클래스들에 각각 구현되어있다.

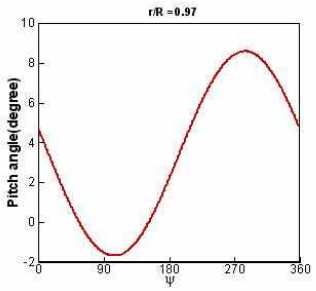
나. 로터 데이터를 위한 분석 데이터 구조

로터 데이터의 분석 데이터를 그래프에 표시되는 내용에 따라 분류하면 크게 pitch angle, sectional data, 압력(pressure)의 세 가지로 분류할 수 있다. 이 중 pitch angle과 sectional data는 사용자가 제공한 포트란 프로그램에서 생성이 가능하다. 이들 데이터를 포트란 프로그램으로 미리 생성한 뒤 하드 디스크에 저장해서 사용하면 좋겠지만, 많은 경우에 각 데이터는 보고자 하는 블레이드의 span 위치에 따라 값이 달라지고 이 span 위치는 매우 다양하기 때문에 미리 생성해서 가지고 있는 것은 문제가 될 수 있다. 따라서 이들 데이터는 사용자가 제공한 포트란 프로그램을 이용해서 실시간으로 계산해야 한다. 이 프로그램들과는 IPC Pipe를 이용해서 통신할 수 있다. 압력 그래프의 경우에는 glvRotorBladeDataSet에서부터 GetPressure() 함수를 이용해서 가져올 수 있으므로 크게 문제가 되지 않는다.

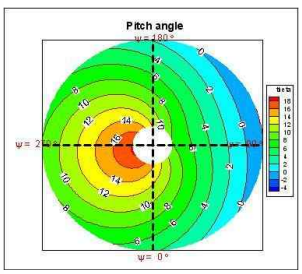
이와 같은 로터 데이터의 분석 데이터 처리를 위해 glvRotorAnalysisData 클래스를 생성했다. 이 클래스에는 CreatePitchAngleData(), CreateSectionalData(), Cr

eatePressureData() 등의 함수를 제공해서 span 위치에 따른 곡선 그래프를 생성할 수 있도록 했고, 또한 CreatePitchAngleDistData(), CreateSectionalDistData() 등의 함수로 원반 형태의 분포 그래프를 생성할 수 있도록 했다. 이들 함수들은 glvAnalysisData의 형태로 데이터를 돌려주게 된다.

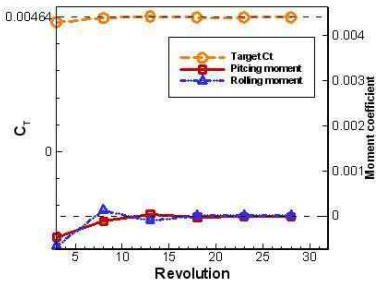
각 분석 데이터 생성 함수들은 사용자가 제공한 포트란 프로그램을 이용해서 실시간으로 생성된다. 사용자의 포트란 프로그램은 pitch angle을 생성하는 프로그램과 sectional data를 생성하는 프로그램으로 크게 두 종류가 있다. 원래 사용자 프로그램은 특정 span 위치에 따른 곡선 그래프를 위한 데이터와 원반 형태의 분포 데이터를 함께 생성했는데, 이를 수정하여 두 데이터를 생성하는 프로그램을 분리하였다. 즉, 총 4개의 프로그램이 있는 것이다. 또한 sectional data를 생성하는 프로그램의 경우 사용자가 제공한 프로그램은 총 6개의 sectional data를 모두 생성하도록 했지만 이것 역시 수정하여 미리 지정한 sectional data 만을 생성하도록 했다. 이들 포트란 프로그램들이 생성하는 분석 데이터는 전진속도나 방위각 증가값, 참조 속도(reference velocity)가 무엇인지, 또 비행의 종류가 전진비행인지 제자리 비행인지 등에 따라 모두 다른 값을 생성하게 된다. 현재 각 포트란 프로그램들은 이들 값이 모두 상수로 미리 지정되어 있는데, 이들 값을 함수 인자로 받아서 처리하도록 변경해야할 필요가 있다.



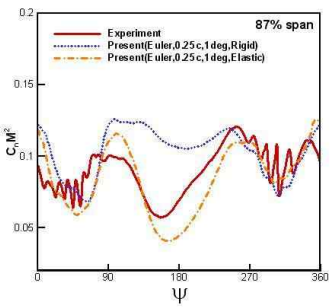
Pitch Angle



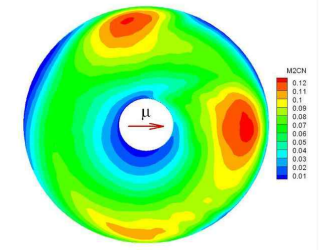
Pitch Angle Distribution



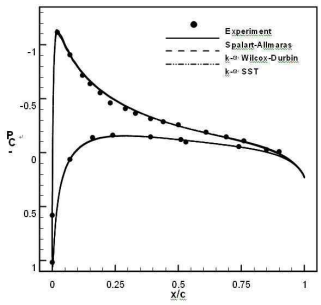
AverageV



Sectional Normal Force



Sectional Normal Force Distribution



Pressure

[그림 4-2] 로터 데이터 분석 그래프들

5. 결론

본 기술문서에서는 GLOVE를 위한 데이터 구조에 대해 살펴보았다. GLOVE의 데이터 구조는 기본적으로 VTK의 데이터 구조를 기반으로 하였지만 로터 동역학 데이터에 최적화해서 로터 동역학 분야 연구자들이 쉽게 사용할 수 있도록 했다. 우선 로터 동역학 분야뿐만 아니라 기타 다른 CFD나 구조해석 분야의 데이터도 처리할 수 있도록 필드 데이터 구조와 벽면 데이터 구조를 설계하였고, 특별히 로터 데이터의 처리를 최적화화기 위해 전체 데이터를 블레이드와 필드의 두 종류로 나누어 서로 다른 형식으로 저장하도록 했다. 또, 분석 데이터를 위한 데이터 클래스도 따로 만들어서 pitch angle이나 sectional thrust와 같이 로터 블레이드 시뮬레이션에서 생성되는 여러 종류의 분석 데이터를 손쉽게 처리하도록 했다. 여기서 설계하고 구현한 데이터 구조들은 로터 데이터의 후처리를 원활하게 하기 위한 것으로 여러 개의 블레이드들 중 특정 블레이드만을 뽑아서 후처리한다거나 대용량의 필드 데이터에서 vortex 영역만을 뽑아서 가시화한다던가 하는 것을 쉽고 빠르게 처리할 수 있도록 한다.

그러나 GLOVE의 데이터 구조는 전적으로 VTK를 기반으로 하였기 때문에 VTK에서 단점으로 지적되는 부분이 그래도 GLOVE의 단점이 된다는 문제가 있다. 대표적으로 GLOVE에서는 데이터에 접근할 때 kd-tree나 octree와 같은 트리 구조를 전혀 사용하지 않기 때문에 거대 데이터를 사용할 때에 속도가 느리다는 단점이 있다. 이는 특히 비구조화된 격자일 경우에 더욱 문제가 될 수 있다. 또 키메라 그리드의 경우에도 VTK에서는 구조화된 격자와 정규 격자에서만 지원을 하기 때문에 다른 형식의 격자구조를 가지는 데이터에 대해서는 지원이 불가능한 상황이다. 이러한 문제점을 해결하기 위해서는 VTK의 구조를 수정하거나 VTK에서 지원하지 않는 부분에 한해서 GLOVE에서 따로 구현을 해야만 한다. 또, 향후 GLOVE에서 병렬 처리를 지원할 경우 데이터를 분산해서 나누어야할 필요가 있는데, VTK의 데이터 구조는 병렬 처리에 취약하다. 사용자가 일일이 데이터를 분배해야 하는데, 데이터 구조에 따라서는 데이터를 일정하게 나누는 것이 매우 어려운 경우도 많다. 향후에는 VTK 데이터 구조를 기반으로 하되 특별히 병렬 및 분산 처리에 효과적인 데이터 구조를 만들어야 할 것이다.

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

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

[6] "Adaptive Mesh Refinement - Theory and Applications", Springer, 2005

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

Appendix. GLOVE Dataset 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

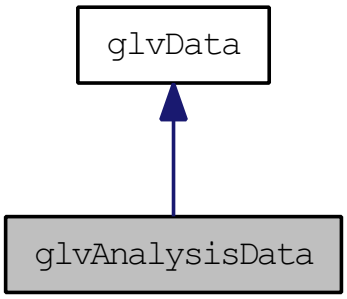
3.1 glvAnalysisData Class Reference

glvAnalysisData (p. 5) is a class for analysis data. This class is an abstract class of **glvAnalysisData1D** (p. 10), **glvAnalysisData2D** (p. 13) and **glvAnalysisData3D** (p. 15).

`#include <glvAnalysisData.h>`

Inherits **glvData**.

Inherited by **glvAnalysisData1D**, **glvAnalysisData2D**, and **glvAnalysisData3D**.Collaboration diagram for glvAnalysisData:



Public Member Functions

- virtual int **RegisterXIndex** (double *index, int size)
- virtual int **RegisterYIndex** (double *index, int size)
- virtual int **RegisterZIndex** (double *index, int size)
- int **RegisterData** (double *data, int size)
- unsigned int **GetWidth** ()
- unsigned int **GetHeight** ()

- unsigned int **GetDepth** ()
- unsigned int **GetSize** ()
- double * **GetXIndex** ()
- double * **GetYIndex** ()
- double * **GetZIndex** ()
- double * **GetData** ()
- int **SetWidth** (unsigned int width)
- int **SetHeight** (unsigned int height)
- int **SetDepth** (unsigned int depth)

3.1.1 Detailed Description

glvAnalysisData (p. 5) is a class for analysis data. This class is an abstract class of **glvAnalysisData1D** (p. 10), **glvAnalysisData2D** (p. 13) and **glvAnalysisData3D** (p. 15).

3.1.2 Member Function Documentation

3.1.2.1 double * glvAnalysisData::GetData ()

Get array of anlysis data

Returns:

NULL if fails or not exists

3.1.2.2 unsigned int glvAnalysisData::GetDepth ()

Get depth of analysis data array. Note that the depth is a size of Z index. Depth must be 1 if dimension of analysis data is less than 3.

3.1.2.3 unsigned int glvAnalysisData::GetHeight ()

Get height of analysis data array. Note that the heihgt is a size of Y index. Height must be 1 if dimension of analysis data is less than 2.

3.1.2.4 unsigned int glvAnalysisData::GetSize ()

Get size of analysis data array. Note than the size is width * height depth

3.1.2.5 unsigned int glvAnalysisData::GetWidth ()

Get width of analysis data array. Note that the width is a size of X index.

3.1.2.6 double * glvAnalysisData::GetXIndex ()

Get array of X index

Returns:

NULL if fails or not exists

3.1.2.7 double * glvAnalysisData::GetYIndex ()

Get array of Y index

Returns:

NULL if fails or not exists

3.1.2.8 double * glvAnalysisData::GetZIndex ()

Get array of Z index

Returns:

NULL if fails or not exists

3.1.2.9 int glvAnalysisData::RegisterData (double * *data*, int *size*)

Register analysis data of given dimension.

Parameters:

data analysis data to register

size size of registering analysis data

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.1.2.10 int glvAnalysisData::RegisterXIndex (double * *index*, int *size*) [virtual]

Register index array of X axis

Parameters:

index index array to register

size size of registering index array

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

Reimplemented in **glvAnalysisData1D** (p. 11).

3.1.2.11 int glvAnalysisData::RegisterYIndex (double * *index*, int *size*) [virtual]

Register index array of Y axis

Parameters:

index index array to register

size size of registering index array

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

Reimplemented in **glvAnalysisData1D** (p. 11).

3.1.2.12 int glvAnalysisData::RegisterZIndex (double * *index*, int *size*) [virtual]

Register index array of Z axis

Parameters:

index index array to register

size size of registering index array

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

Reimplemented in **glvAnalysisData1D** (p. 12), and **glvAnalysisData2D** (p. 14).

3.1.2.13 int glvAnalysisData::SetDepth (unsigned int *depth*)

Set depth of analysis data. Note that the depth is a size of Z index.

Parameters:

depth Depth of analysis data

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.1.2.14 int glvAnalysisData::SetHeight (unsigned int *height*)

Set height of analysis data. Note that the heihgt is a size of Y index.

Parameters:

height Height of analysis data

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.1.2.15 int glvAnalysisData::SetWidth (unsigned int *width*)

Set width of analysis data. Note that the width is a size of X index.

Parameters:

width Width of analysis data

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

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

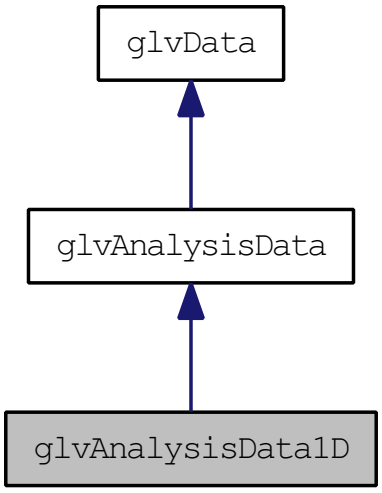
- /home/osung/tmp/glove/trunk/include/common/glvAnalysisData.h
- /home/osung/tmp/glove/trunk/src/common/glvAnalysisData.cc

3.2 glvAnalysisData1D Class Reference

glvAnalysisData1D (p. 10) is a class for 1D analysis data. This class is a concrete implementation of **glvAnalysisData** (p. 5).

`#include <glvAnalysisData1D.h>`

Inherits **glvAnalysisData**.Collaboration diagram for glvAnalysisData1D:



Public Member Functions

- virtual void **Delete** ()
- int **RegisterIndex** (double *index, int size)
- int **RegisterXIndex** (double *index, int size)
- int **RegisterYIndex** (double *index, int size)
- int **RegisterZIndex** (double *index, int size)

Static Public Member Functions

- static **glvAnalysisData1D** * **New** ()

  Create a 1D analysis data object.

3.2.1 Detailed Description

glvAnalysisData1D (p. 10) is a class for 1D analysis data. This class is a concrete implementation of **glvAnalysisData** (p. 5).

3.2.2 Member Function Documentation

3.2.2.1 void glvAnalysisData1D::Delete () [virtual]

Delete an object. A pure virtual function.

Implements **glvData** (p. 17).

3.2.2.2 int glvAnalysisData1D::RegisterIndex (double * *index*, int *size*)

Register index array of X axis. There is only x index in 1D array.

Parameters:

index index array to register

size size of registering index array

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.2.2.3 int glvAnalysisData1D::RegisterXIndex (double * *index*, int *size*) [virtual]

Register index array of X axis

Parameters:

index index array to register

size size of registering index array

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

Reimplemented from **glvAnalysisData** (p. 7).

3.2.2.4 int glvAnalysisData1D::RegisterYIndex (double * *index*, int *size*) [virtual]

Register index array of Y axis

Parameters:

index index array to register

size size of registering index array

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

Reimplemented from **glvAnalysisData** (p. 8).

3.2.2.5 int glvAnalysisData1D::RegisterZIndex (double * *index*, int *size*) [virtual]

Register index array of Z axis

Parameters:

index index array to register

size size of registering index array

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

Reimplemented from **glvAnalysisData** (p. 8).

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

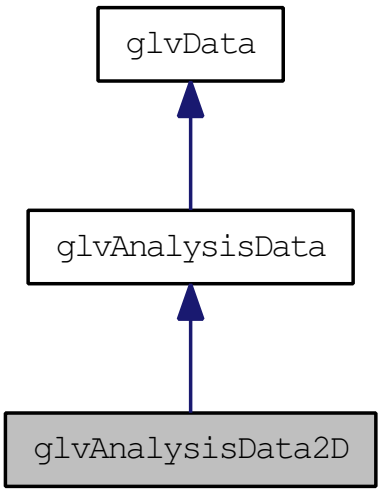
- /home/osung/tmp/glove/trunk/include/common/glvAnalysisData1D.h
- /home/osung/tmp/glove/trunk/src/common/glvAnalysisData1D.cc

3.3 glvAnalysisData2D Class Reference

glvAnalysisData2D (p. 13) is a class for 2D analysis data. This class is a concrete implementation of **glvAnalysisData** (p. 5).

`#include <glvAnalysisData2D.h>`

Inherits **glvAnalysisData**.Collaboration diagram for glvAnalysisData2D:



Public Member Functions

- virtual void **Delete** ()
- int **RegisterZIndex** (double *index, int size)

Static Public Member Functions

- static **glvAnalysisData2D** * **New** ()

  Create a 2D analysis data object.

3.3.1 Detailed Description

glvAnalysisData2D (p. 13) is a class for 2D analysis data. This class is a concrete implementation of **glvAnalysisData** (p. 5).

3.3.2 Member Function Documentation

3.3.2.1 void glvAnalysisData2D::Delete () [virtual]

Delete an object. A pure virtual function.

Implements **glvData** (p. 17).

3.3.2.2 int glvAnalysisData2D::RegisterZIndex (double * *index*, int *size*) [virtual]

Register index array of Z axis

Parameters:

index index array to register

size size of registering index array

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

Reimplemented from **glvAnalysisData** (p. 8).

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

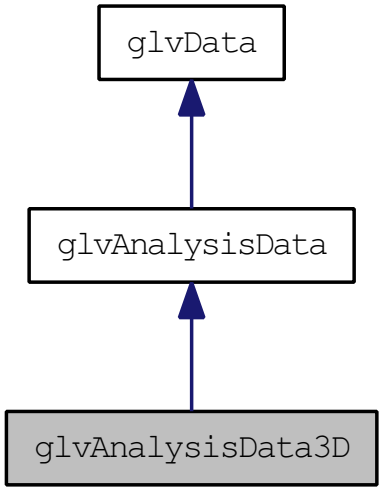
- /home/osung/tmp/glove/trunk/include/common/glvAnalysisData2D.h
- /home/osung/tmp/glove/trunk/src/common/glvAnalysisData2D.cc

3.4 glvAnalysisData3D Class Reference

glvAnalysisData3D (p. 15) is a class for 3D analysis data. This class is a concrete implementation of **glvAnalysisData** (p. 5).

`#include <glvAnalysisData3D.h>`

Inherits **glvAnalysisData**.Collaboration diagram for glvAnalysisData3D:



Public Member Functions

- virtual void **Delete** ()

Static Public Member Functions

- static **glvAnalysisData3D** * **New** ()

  Create a 3D analysis data object.

3.4.1 Detailed Description

glvAnalysisData3D (p. 15) is a class for 3D analysis data. This class is a concrete implementation of **glvAnalysisData** (p. 5).

3.4.2 Member Function Documentation

3.4.2.1 void glvAnalysisData3D::Delete () [virtual]

Delete an object. A pure virtual function.

Implements **glvData** (p. 17).

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

- /home/osung/tmp/glove/trunk/include/common/glvAnalysisData3D.h
- /home/osung/tmp/glove/trunk/src/common/glvAnalysisData3D.cc

3.5 glvData Class Reference

glvData (p. 17) is an abstract class of **glvDataSet** (p. 19)

`#include <glvData.h>`

Inherited by **glvAnalysisData**, and **glvDataSet**.

Public Member Functions

- virtual void **Delete** ()=0
- int **GetEndian** ()
- int **SetEndian** (int endian)

Protected Member Functions

- int **ConvertToLittleEndian** ()

  Convert internal byte order to little endian.

- int **ConvertToBigEndian** ()

  Convert internal byte order to big endian.

Protected Attributes

- int **mEndian**

3.5.1 Detailed Description

glvData (p. 17) is an abstract class of **glvDataSet** (p. 19)

3.5.2 Member Function Documentation

3.5.2.1 virtual void glvData::Delete () [pure virtual]

Delete an object. A pure virtual function.

Implemented in **glvAnalysisData1D** (p. 11), **glvAnalysisData2D** (p. 13), **glvAnalysisData3D** (p. 15), **glvDataSet** (p. 19), **glvFieldDataSet** (p. 38), and **glvWallDataSet** (p. 78).

3.5.2.2 int glvData::GetEndian ()

Get byte order.

See also:

SetEndian() (p. 18)

Returns:

either of GLV_LITTLE_ENDIAN or GLV_BIG_ENDIAN

3.5.2.3 int glvData::SetEndian (int *endian*)

Set byte order.

Parameters:

endian should be either of GLV_LITTLE_ENDIAN or GLV_BIG_-ENDIAN

See also:

GetEndian() (p. 17)

Returns:

Return non-zero if succeeds; Return GLV_FALSE if fails

3.5.3 Member Data Documentation

3.5.3.1 int glvData::mEndian [protected]

Byte order. Either of GLV_LITTLE_ENDIAN or GLV_BIG_ENDIAN

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

- /home/osung/tmp/glove/trunk/include/common/glvData.h
- /home/osung/tmp/glove/trunk/src/common/glvData.cc

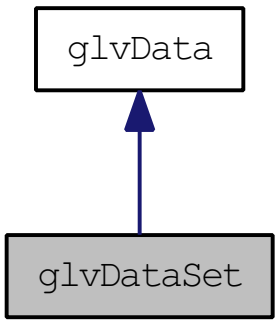
3.6 glvDataSet Class Reference

glvDataSet (p. 19) is an abstract calss of **glvFieldDataSet** (p. 38) and **glvWallDataSet** (p. 78). This dataset can be a multi-block (or single block) of single (or multiple) time steps Every data block in this dataset should have identical variables.

`#include <glvDataSet.h>`

Inherits **glvData**.

Inherited by **glvFieldDataSet**, and **glvWallDataSet**.Collaboration diagram for glvDataSet:



Public Member Functions

- int **RegisterVTKObject** (vtkDataObject *data)
- int **RegisterVTKTimeStep** (vtkDataObject *data, int timeStep)
- int **MergeInternalVTKObjects** (vtkTemporalDataSet *blocks)
- int **AddVector** (char *name, int step, double *vector, int ncomp, int size)
- int **AddVector** (char *name, int step, int block, double *vector, int ncomp, int size)
- int **AddScalar** (char *name, int step, double *scalar, int size)
- int **AddScalar** (char *name, int step, int block, double *scalar, int size)
- int **AddVectorMagnitude** ()
- int **AddVectorMagnitude** (int ID)
- int **AddVectorMagnitude** (char *name)
- virtual void **Delete** ()=0

  Delete an object.

- void **ReleaseData** ()

  Release associated VTK data object.

- int **RemoveTimeStep** (int timeStep)
- int **GetNumberOfBlocks** (int timeStep)
- int **GetNumberOfTimeSteps** ()

- int **GetNumberOfPoints** ()
- int **GetNumberOfVariables** ()
- int **GetNumberOfScalars** ()
- int **GetNumberOfVectors** ()
- int **GetCurrentTimeStep** ()
- const char * **GetVariableName** (int id)
- int **GetVariableID** (char *name)
- const char * **GetScalarName** (int id)
- int **GetScalarID** (char *name)
- char ** **GetScalarNames** (int *size)
- const char * **GetActiveScalarName** ()
- const char * **GetActiveVectorName** ()
- int **GetActiveScalarID** ()
- int **GetActiveVectorID** ()
- const char * **GetVectorName** (int id)
- int **GetVectorID** (char *name)
- char ** **GetVectorNames** (int *size)
- int **SetActiveScalar** (char *name)
- int **SetActiveScalar** (int id)
- int **SetActiveVector** (char *name)
- int **SetActiveVector** (int id)
- int **SetCurrentTimeStep** (int timeStep)
- int **SetNextTimeStep** ()
- int **SetPrevTimeStep** ()
- double * **Get** (double x, double y, double z, char *variableName)
- double * **Get** (double point[3], char *variableName)
- double * **GetScalarRange** (char *variableName)
- double * **GetScalarRange** (int id)
- double * **GetScalarRange** ()
- int **GetScalarRange** (char *variableName, double range[2])
- int **GetScalarRange** (int id, double range[2])
- int **GetScalarRange** (double range[2])
- int **GetSingleStepScalarRange** (int timeStep, double range[2])
- int **GetSingleStepScalarRange** (double range[2])
- int **GetSingleStepScalarRange** (char *variableName, int timeStep, double range[2])
- int **GetSingleStepScalarRange** (char *variableName, double range[2])
- int **HasScalar** (char *name)
- int **HasVector** (char *name)
- virtual **glvDataSet** * **GetBlock** (int block, int timeStep)=0
- virtual **glvDataSet** * **GetSingleTimeStep** (int timeStep)=0
- virtual **glvDataSet** * **GetTemporalBlock** (int block)=0
- vtkTemporalDataSet * **GetVTKObject** ()
- vtkDataSet * **GetVTKDataSet** (int block, int timeStep)
- vtkMultiBlockDataSet * **GetVTKDataBlock** (int timeStep)
- vtkMultiBlockDataSet * **GetVTKDataBlock** ()
- vtkTemporalDataSet * **GetVTKTemporalBlockObject** (int block)

Protected Member Functions

- void **SetNumberOfTimeSteps** (int steps)
- int **CheckTimeStep** (int step)
- void **SetAttributeTypes** ()

  Update attribute types (scalar or vector) in registered VTK dataset.

- int **GetScalarArrayIndex** (int idx)

  Get actual array index of scalar index.

- int **GetVectorArrayIndex** (int idx)

  Get actual array index of vector index.

- vtkMultiBlockDataSet * **ConvertToMultiBlock** (vtkDataSet *dataset)
- vtkTemporalDataSet * **ConvertToTemporalDataSet** (vtkMultiBlock-DataSet *mb)

Protected Attributes

- vtkTemporalDataSet * **mData**

  internal VTK data object

- int **mNumberOfTimeSteps**

  number of time steps

- int **mCurrentTimeStep**

  current time step

- int * **mAttributeTypes**
- int **mActiveScalar**

  ID of current active scalar.

- int **mActiveVector**

  ID of current active vector.

3.6.1 Detailed Description

glvDataSet (p. 19) is an abstract calss of **glvFieldDataSet** (p. 38) and **glvWallDataSet** (p. 78). This dataset can be a multi-block (or single block) of single (or multiple) time steps Every data block in this dataset should have identical variables.

3.6.2 Member Function Documentation

3.6.2.1 int glvDataSet::AddScalar (char * *name*, int *step*, int *block*, double * *scalar*, int *size*)

Add scalar by data block unit

Parameters:

name name of adding scalar

step target time step to add. It must be less than total number of time steps

block number of block. It must be less than total number of blocks

scalar array of adding scalar

size number of tuples

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.6.2.2 int glvDataSet::AddScalar (char * *name*, int *step*, double * *scalar*, int *size*)

Add scalar by time step unit

Parameters:

name name of adding scalar

step target time step to add. It must be less than total number of time steps

scalar array of adding scalar

size number of tuples

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.6.2.3 int glvDataSet::AddVector (char * *name*, int *step*, int *block*, double * *vector*, int *ncomp*, int *size*)

Add vector by data block unit.

Parameters:

name name of adding vector

step target time step to add. It must be less than total number of time steps

block number of block. It must be less than total number of blocks

vector array of adding vector

ncomp number of components

size number of tuples

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.6.2.4 int glvDataSet::AddVector (char * *name*, int *step*, double * *vector*, int *ncomp*, int *size*)

Add vector by time step unit

Parameters:

name name of adding vector

step target time step to add. It must be less than total number of time steps

vector array of adding vector

ncomp number of components

size number of tuples

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.6.2.5 int glvDataSet::AddVectorMagnitude (char * *name*)

Add scalar field for the magnitude of given registered vector

Parameters:

name name of vector

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.6.2.6 int glvDataSet::AddVectorMagnitude (int *ID*)

Add scalar field for the magnitude of given registered vector

Parameters:

ID vector ID

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.6.2.7 int glvDataSet::AddVectorMagnitude ()

Add scalar field for the magnitude of current active vector

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.6.2.8 int glvDataSet::CheckTimeStep (int *step*) [protected]

Check input step is valid or not

Parameters:

step time step to check

Returns:

GLV_TRUE if valid, GLV_FALSE if not valid

3.6.2.9 vtkMultiBlockDataSet * glvDataSet::ConvertToMultiBlock (vtkDataSet * *dataset*) [protected]

convert single block dataset to multi-block format contians single block

3.6.2.10 vtkTemporalDataSet * glvDataSet::ConvertToTemporalDataSet (vtkMultiBlockDataSet * *mb*) [protected]

convert steady multi-block dataset to the temporal format contians single time step

3.6.2.11 double* glvDataSet::Get (double *point*[3], char * *variableName*)

Return data of input point location

Parameters:

point point location

variableName name of variable to access

Returns:

array of double. It returns data as an array pointer due to the vector

3.6.2.12 double* glvDataSet::Get (double *x*, double *y*, double *z*, char * *variableName*)

Return data of input point location

Parameters:

x X coordinate of the point location

y Y coordinate of the point location

z Z coordinate of the point location

variableName name of variable to access

Returns:

array of double. It returns data as an array pointer due to the vector

3.6.2.13 int glvDataSet::GetActiveScalarID ()

Get ID of active scalar

Returns:

ID of current active scalar

3.6.2.14 const char * glvDataSet::GetActiveScalarName ()

Get name of active scalar

Returns:

char string for name of currently active scalar

3.6.2.15 int glvDataSet::GetActiveVectorID ()

Get ID of active vector

Returns:

ID of current active vector

3.6.2.16 const char * glvDataSet::GetActiveVectorName ()

Get name of active vector

Returns:

char string for name of currently active vector

3.6.2.17 virtual glvDataSet* glvDataSet::GetBlock (int *block*, int *timeStep*) [pure virtual]

Return data block of the given time step and block ID

Parameters:

block block ID
timeStep time step

Returns:

NULL if fails

Implemented in **glvFieldDataSet** (p. 39), and **glvWallDataSet** (p. 79).

3.6.2.18 int glvDataSet::GetCurrentTimeStep ()

Get current time step

Returns:

current time step

3.6.2.19 int glvDataSet::GetNumberOfBlocks (int *timeStep*)

Get nubmer of blocks of single time step

Parameters:

timeStep time step

Returns:

number of blocks

3.6.2.20 int glvDataSet::GetNumberOfPoints ()

Get number of points in the whole data blocks

Returns:

number of points

3.6.2.21 int glvDataSet::GetNumberOfScalars ()

Get number of scalars

Returns:

number of scalars

3.6.2.22 int glvDataSet::GetNumberOfTimeSteps ()

Get number of time steps

Returns:

number of time steps

3.6.2.23 int glvDataSet::GetNumberOfVariables ()

Get number of variables (scalars + vectors)

Returns:

number of variables

3.6.2.24 int glvDataSet::GetNumberOfVectors ()

Get number of vectors

Returns:

number of vectors

3.6.2.25 int glvDataSet::GetScalarID (char * *name*)

Get scalar ID

Parameters:

name scalar name

Returns:

scalar ID of given name

3.6.2.26 const char * glvDataSet::GetScalarName (int *id*)

Get name of scalar

Parameters:

id scalar ID

Returns:

scalar name of given ID

3.6.2.27 char ** glvDataSet::GetScalarNames (int * *size*)

Get name of scalars

Parameters:

size container to return number of scalars

Returns:

2D array of scalar names

3.6.2.28 int glvDataSet::GetScalarRange (double *range*[2])

Get scalar range of the current active scalar variable.

Parameters:

range container to return scalar range

Returns:

Return GL_FALSE if the variable does not exist or is not scalar variable. Return GL_TRUE if succeeds.

3.6.2.29 int glvDataSet::GetScalarRange (int *id*, double *range*[2])

Get scalar range of the variable.

Parameters:

id scalar ID

range container to return scalar range

Returns:

Return GL_FALSE if the variable does not exist. Return GL_TRUE if succeeds.

3.6.2.30 int glvDataSet::GetScalarRange (char * *variableName*, double *range*[2])

Get scalar range of the variable.

Parameters:

variableName scalar name

range container to return scalar range

Returns:

Return GL_FALSE if the variable does not exist or is not scalar variable. Return GL_TRUE if succeeds.

3.6.2.31 double * glvDataSet::GetScalarRange ()

Return scalar range of the current active scalar variable. Return NULL if the variable does not exist or is not scalar variable.

Returns:

double [2]

3.6.2.32 double * glvDataSet::GetScalarRange (int *id*)

Return scalar range of the variable. Return NULL if the variable does not exist.

Parameters:

id ID of scalar to access

Returns:

double [2]

3.6.2.33 double * glvDataSet::GetScalarRange (char * *variableName*)

Return scalar range of the variable. Return NULL if the variable does not exist or is not scalar variable.

Parameters:

variableName name of scalar to access

Returns:

double [2]

3.6.2.34 int glvDataSet::GetSingleStepScalarRange (char * *variableName*, double *range*[2])

Get range of the given scalar value of the current time step

Parameters:

variableName name of scalar

range container to return scalar range

Returns:

Return GL_FALSE if the variable does not exist or is not scalar variable. Return GL_TRUE if succeeds.

3.6.2.35 int glvDataSet::GetSingleStepScalarRange (char * *variableName*, int *timeStep*, double *range*[2])

Get range of the given scalar value of the given time step

Parameters:

variableName name of scalar

timeStep time step

range container to return scalar range

Returns:

Return GL_FALSE if the variable does not exist or is not scalar variable. Return GL_TRUE if succeeds.

3.6.2.36 int glvDataSet::GetSingleStepScalarRange (double *range*[2])

Get range of active scalar value of the current time step

Parameters:

range container to return scalar range

Returns:

Return GL_FALSE if the variable does not exist or is not scalar variable. Return GL_TRUE if succeeds.

3.6.2.37 int glvDataSet::GetSingleStepScalarRange (int *timeStep*, double *range*[2])

Get range of active scalar value of the given time step

Parameters:

timeStep time step

range container to return scalar range

Returns:

Return GL_FALSE if the variable does not exist or is not scalar variable. Return GL_TRUE if succeeds.

3.6.2.38 virtual glvDataSet* glvDataSet::GetSingleTimeStep (int *timeStep*) [pure virtual]

Return dataset of the given time step

Parameters:

timeStep time step

Returns:

NULL if fails

Implemented in **glvFieldDataSet** (p. 40), and **glvWallDataSet** (p. 80).

3.6.2.39 virtual glvDataSet* glvDataSet::GetTemporalBlock (int *block*) [pure virtual]

Return temporal block of the given block ID

Parameters:

block block ID

Returns:

NULL if fails

Implemented in **glvFieldDataSet** (p. 40), and **glvWallDataSet** (p. 80).

3.6.2.40 int glvDataSet::GetVariableID (char * *name*)

Get variable ID

Parameters:

name variable name

Returns:

variable ID of given name

3.6.2.41 const char * glvDataSet::GetVariableName (int *id*)

Get variable name

Parameters:

id variable ID

Returns:

variable name of given ID

3.6.2.42 int glvDataSet::GetVectorID (char * *name*)

Get vector ID

Parameters:

name vector name

Returns:

vector ID of given name

3.6.2.43 const char * glvDataSet::GetVectorName (int *id*)

Get name of vector

Parameters:

id vector ID

Returns:

vector name of given ID

3.6.2.44 char ** glvDataSet::GetVectorNames (int * *size*)

Get name of vectors

Parameters:

size container to return number of vectors

Returns:

2D array of vector names

3.6.2.45 vtkMultiBlockDataSet * glvDataSet::GetVTKDataBlock ()

Return internal vtk data object of current time step as a vtkMultiBlockDataSet

Returns:

NULL if fails

3.6.2.46 vtkMultiBlockDataSet * glvDataSet::GetVTKDataBlock (int *timeStep*)

Return internal vtk data object of given time step as a vtkMultiBlockDataSet

Parameters:

timeStep time step to extract

Returns:

NULL if fails

3.6.2.47 vtkDataSet * glvDataSet::GetVTKDataSet (int *block*, int *timeStep*)

Return internal vtk data object as a vtkDataSet

Parameters:

block block ID

timeStep time step to extract

Returns:

NULL if fails

3.6.2.48 vtkTemporalDataSet * glvDataSet::GetVTKObject ()

Return internal vtk data object

Returns:

NULL if no internal vtk data is registered

3.6.2.49 vtkTemporalDataSet* glvDataSet::GetVTKTemporalBlockObject (int *block*)

Return internal vtk data object as a vtkTemporalDataSet

Parameters:

block block ID

Returns:

NULL if fails

3.6.2.50 int glvDataSet::HasScalar (char * *name*)

Parameters:

name name of scalar to search

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.6.2.51 int glvDataSet::HasVector (char * *name*)

Parameters:

name name of vector to search

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.6.2.52 int glvDataSet::MergeInternalVTKObjects (vtkTemporalDataSet * *blocks*)

Merge two internal VTK objects.

Parameters:

blocks should have same number of time steps to already registered object

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.6.2.53 int glvDataSet::RegisterVTKObject (vtkDataObject * *data*)

Register a VTK dataset object. The object should be either of vtkDataSet and subclasses or vtkCompositeDataSet and subclasses. If vtk object is already registered, this function replaces it.

Parameters:

data should be either of vtkDataSet and subclasses or vtkCompositeDataSet and subclasses.

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

See also:

RegisterTimeStepVTKObject()

3.6.2.54 int glvDataSet::RegisterVTKTimeStep (vtkDataObject * *data*, int *timeStep*)

Register a single time step vtk object into the specified temporal location. If already registered, replace it. Registered object should be identical data type and have exactly same variables.

Parameters:

data should be either of vtkDataSet and subclasses or vtkCompositeDataSet and subclasses.

timeStep time step to register

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

See also:

RegisterVTKObject() (p. 34)

3.6.2.55 int glvDataSet::RemoveTimeStep (int *timeStep*)

Remove specified time step

Parameters:

timeStep time step to delete

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.6.2.56 int glvDataSet::SetActiveScalar (int *id*)

Set active scalar variable by integer ID

Parameters:

id ID of scalar to activate

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.6.2.57 int glvDataSet::SetActiveScalar (char * *name*)

Set active scalar variable by char string

Parameters:

name name of scalar to activate

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.6.2.58 int glvDataSet::SetActiveVector (int *id*)

Set active vector variable by integer ID

Parameters:

id ID of vector to activate

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.6.2.59 int glvDataSet::SetActiveVector (char * *name*)

Set active vector variable by char string

Parameters:

name name of vector to activate

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.6.2.60 int glvDataSet::SetCurrentTimeStep (int *timeStep*)

Set current time step

Parameters:

timeStep time step to be set to current

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.6.2.61 int glvDataSet::SetNextTimeStep ()

Increase currnet time step

Returns:

GLV_TRUE if succeeds, GLV_FALSE if already last step

3.6.2.62 void glvDataSet::SetNumberOfTimeSteps (int *steps*) [protected]

Set number of time steps

Parameters:

steps number of time steps

3.6.2.63 int glvDataSet::SetPrevTimeStep ()

Decrease currnet time step

Returns:

GLV_TRUE if succeeds, GLV_FALSE if already first step

3.6.3 Member Data Documentation

3.6.3.1 int* glvDataSet::mAttributeTypes [protected]

array of attribute types of variables. either of scalar(0) or vector(1)

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

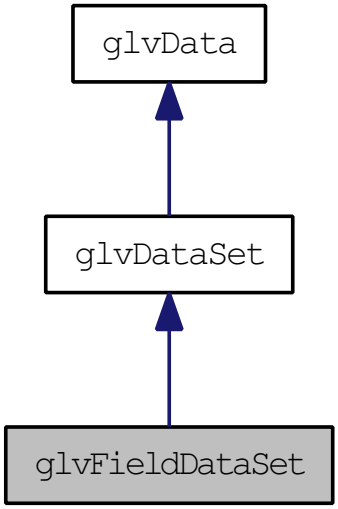
- /home/osung/tmp/glove/trunk/include/common/glvDataSet.h
- /home/osung/tmp/glove/trunk/src/common/glvDataSet.cc

3.7 glvFieldDataSet Class Reference

glvFieldDataSet (p. 38) is a class for field dataset. This class is a concrete implementation of **glvDataSet** (p. 19).

`#include <glvFieldDataSet.h>`

Inherits **glvDataSet**.Collaboration diagram for glvFieldDataSet:



Public Member Functions

- void **Delete** ()

  Remove dataset.

- int **RegisterTimeStep** (**glvFieldDataSet** *dataset, int timeStep)
- int **AddTimeStep** (**glvFieldDataSet** *data)
- **glvFieldDataSet** * **GetBlock** (int block, int timeStep)
- **glvFieldDataSet** * **GetSingleTimeStep** (int timeStep)
- **glvFieldDataSet** * **GetTemporalBlock** (int block)
- int **SetBlankField** (int ID)
- int **SetBlankField** (char *name)
- int **EnableBlanking** ()
- int **DisableBlanking** ()

Static Public Member Functions

- static **glvFieldDataSet** * **New** ()
- static **glvFieldDataSet** * **New** (int timeStep)
- static **glvFieldDataSet** * **New** (vtkDataObject *data)

3.7.1 Detailed Description

glvFieldDataSet (p. 38) is a class for field dataset. This class is a concrete implementation of **glvDataSet** (p. 19).

3.7.2 Member Function Documentation

3.7.2.1 int glvFieldDataSet::AddTimeStep (glvFieldDataSet * *data*)

Add a single time step dataset at the end of the time step

Parameters:

data registered dataset

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

Reimplemented from **glvDataSet** (p. 19).

3.7.2.2 int glvFieldDataSet::DisableBlanking ()

Disable blanking

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.7.2.3 int glvFieldDataSet::EnableBlanking ()

Enalbe blaning if blanking field is set

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.7.2.4 glvFieldDataSet * glvFieldDataSet::GetBlock (int *block*, int *timeStep*) [virtual]

Return data block of given time step and block ID

Parameters:

block block ID

timeStep time step

Returns:

NULL if fails

Implements **glvDataSet** (p. 26).

3.7.2.5 glvFieldDataSet * glvFieldDataSet::GetSingleTimeStep (int *timeStep*) [virtual]

Return data set of the given time step

Parameters:

timeStep time step

Returns:

NULL if fails

Implements **glvDataSet** (p. 31).

3.7.2.6 glvFieldDataSet * glvFieldDataSet::GetTemporalBlock (int *block*) [virtual]

Return temporal block of the given block ID

Parameters:

block block ID

Returns:

NULL if fails

Implements **glvDataSet** (p. 31).

3.7.2.7 glvFieldDataSet * glvFieldDataSet::New (vtkDataObject * *data*) [static]

Create an dataset object from the input vtk object

Parameters:

data initial internal vtk object

See also:

New() (p. 41)

3.7.2.8 glvFieldDataSet * glvFieldDataSet::New (int *timeStep*) [static]

Create an dataset of speficed blocks and time steps

Parameters:

timeStep number of time steps

See also:

New() (p. 41)

3.7.2.9 glvFieldDataSet * glvFieldDataSet::New () [static]

Create an empty object

See also:

New() (p. 41)

3.7.2.10 int glvFieldDataSet::RegisterTimeStep (glvFieldDataSet * *dataset*, int *timeStep*)

Register a single time step dataset into the specified temporal location; If alread registered, replace it.

Parameters:

dataset registered dataset

timeStep insert temporal position

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

Reimplemented from **glvDataSet** (p. 19).

3.7.2.11 int glvFieldDataSet::SetBlankField (char * *name*)

Set blank field

Parameters:

name scalar name. name must have scalar property

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.7.2.12 int glvFieldDataSet::SetBlankField (int *ID*)

Set blank field

Parameters:

ID scalar ID. ID must be scalar

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

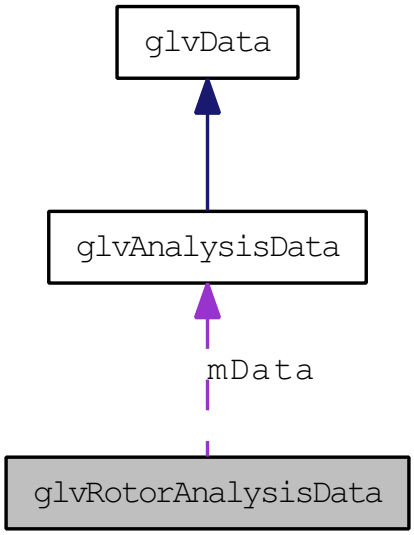
The documentation for this class was generated from the following files:

- /home/osung/tmp/glove/trunk/include/common/glvFieldDataSet.h
- /home/osung/tmp/glove/trunk/src/common/glvFieldDataSet.cc

3.8 glvRotorAnalysisData Class Reference

glvRotorAnalysisData (p. 43) is a class for rotor analysis data. Default width of all anlaysis data is 361. And Default height is 21.

`#include <glvRotorAnalysisData.h>`Collaboration diagram for glvRotorAnalysisData:



Public Member Functions

- int **CreatePitchAngleData** (double spanPosition)
- int **CreatePitchAngleDistData** ()
- int **CreateSectionalData** (ROTOR_ANALYSIS_TYPE t, ROTOR_FLIGHT_TYPE f, double sPos, int start, int end)
- int **CreateSectionalData** (ROTOR_ANALYSIS_TYPE t, ROTOR_FLIGHT_TYPE f, double sPos)
- int **CreateSteadySectionalData** (ROTOR_ANALYSIS_TYPE t, ROTOR_FLIGHT_TYPE f, int frame, int resol)
- int **CreateSectionalDistData** (ROTOR_ANALYSIS_TYPE t, ROTOR_FLIGHT_TYPE f, int start, int end)
- int **CreateSectionalDistData** (ROTOR_ANALYSIS_TYPE t, ROTOR_FLIGHT_TYPE f)
- int **CreatePressureData** (**glvRotorBladeDataSet** *blades, int step, ROTOR_BLADE_FACE face, int size)
- int **CreateAverageVData** (ROTOR_TRIM_TYPE t)
- int **Delete** ()

  Delete the object.

- int **ReleaseData** ()

- ROTOR_ANALYSIS_TYPE **GetType** ()
- **glvAnalysisData** * **Get** ()

Static Public Member Functions

- static **glvRotorAnalysisData** * **New** ()

  Create empty data object.

Protected Attributes

- **glvAnalysisData** * **mData**

  internal analysis data

- ROTOR_ANALYSIS_TYPE **mDataType**

  type of rotor analysis data

3.8.1 Detailed Description

glvRotorAnalysisData (p. 43) is a class for rotor analysis data. Default width of all anlaysis data is 361. And Default height is 21.

3.8.2 Member Function Documentation

3.8.2.1 int glvRotorAnalysisData::CreateAverageVData (ROTOR_TRIM_TYPE *t*)

Create AverageV data

Parameters:

t rotor trim type. One of ROTOR_CT, ROTOR_ROLL_MOMENT, or ROTOR_PITCH_MOMENT,

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.8.2.2 int glvRotorAnalysisData::CreatePitchAngleData (double *spanPosition*)

Create pitch angle data at the given span position

Parameters:

spanPosition span position. Must be >= 0.0 and <= 1.0

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.8.2.3 int glvRotorAnalysisData::CreatePitchAngleDistData ()

Create pitch angle distribution data.

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.8.2.4 int glvRotorAnalysisData::CreatePressureData (glvRotorBladeDataSet * *blades*, int *step*, ROTOR_BLADE_FACE *face*, int *size*)

Create pressure data of given rotor blade

Parameters:

blades rotor blade data

step time step

face front or back face

size size of 1D array (= width)

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.8.2.5 int glvRotorAnalysisData::CreateSectionalData (ROTOR_ANALYSIS_TYPE *t*, ROTOR_FLIGHT_TYPE *f*, double *sPos*)

Create sectional data at the given span position. Default start and end parameter is used.

Parameters:

t rotor analysis type. One of ROTOR_SECT_NORM_FORCE, ROTOR_SECT_SPAN_FORCE, ROTOR_SECT_CHORD_FORCE, ROTOR_SECT_X_MOMENT, ROTOR_SECT_Y_MOMENT, ROTOR_SECT_Z_MOMENT

f flight type either of ROTOR_FLIGHT_HOVER and ROTOR_FLIGHT_FORWARD

sPos span position

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.8.2.6 int glvRotorAnalysisData::CreateSectionalData (ROTOR_ANALYSIS_TYPE *t*, ROTOR_FLIGHT_TYPE *f*, double *sPos*, int *start*, int *end*)

Create sectional data at the given span position

Parameters:

t rotor analysis type. One of ROTOR_SECT_NORM_FORCE, ROTOR_SECT_SPAN_FORCE, ROTOR_SECT_CHORD_FORCE, ROTOR_SECT_X_MOMENT, ROTOR_SECT_Y_MOMENT, ROTOR_SECT_Z_MOMENT

f flight type either of ROTOR_FLIGHT_HOVER and ROTOR_FLIGHT_FORWARD

sPos span position

start start time step

end end time step

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.8.2.7 int glvRotorAnalysisData::CreateSectionalDistData (ROTOR_ANALYSIS_TYPE *t*, ROTOR_FLIGHT_TYPE *f*)

Create distribution of sectional data

Parameters:

t rotor analysis type. One of ROTOR_SECT_NORM_FORCE, ROTOR_SECT_SPAN_FORCE, ROTOR_SECT_CHORD_FORCE, ROTOR_SECT_X_MOMENT, ROTOR_SECT_Y_MOMENT, ROTOR_SECT_Z_MOMENT

f flight type either of ROTOR_FLIGHT_HOVER and

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.8.2.8 int glvRotorAnalysisData::CreateSectionalDistData (ROTOR_ANALYSIS_TYPE *t*, ROTOR_FLIGHT_TYPE *f*, int *start*, int *end*)

Create distribution of sectional data

Parameters:

t rotor analysis type. One of ROTOR_SECT_NORM_FORCE, ROTOR_SECT_SPAN_FORCE, ROTOR_SECT_CHORD_FORCE, ROTOR_SECT_X_MOMENT, ROTOR_SECT_Y_MOMENT, ROTOR_SECT_Z_MOMENT

f flight type either of ROTOR_FLIGHT_HOVER and

start start time step

end end time step

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.8.2.9 int glvRotorAnalysisData::CreateSteadySectionalData (ROTOR_ANALYSIS_TYPE *t*, ROTOR_FLIGHT_TYPE *f*, int *frame*, int *resol*)

Create sectional data at the given time step.

Parameters:

t rotor analysis type. One of ROTOR_SECT_NORM_FORCE, ROTOR_SECT_SPAN_FORCE, ROTOR_SECT_CHORD_FORCE, ROTOR_SECT_X_MOMENT, ROTOR_SECT_Y_MOMENT, ROTOR_SECT_Z_MOMENT

f flight type either of ROTOR_FLIGHT_HOVER and

frame time step

resol resolution (default is 51)

3.8.2.10 glvAnalysisData * glvRotorAnalysisData::Get ()

Return internal analysis data

Returns:

NULL if fails

3.8.2.11 ROTOR_ANALYSIS_TYPE glvRotorAnalysisData::GetType ()

Return rotor analysis type

Returns:

rotor analysis type. One of ROTOR_SECT_NORM_FORCE, ROTOR_SECT_SPAN_FORCE, ROTOR_SECT_CHORD_FORCE, ROTOR_SECT_X_MOMENT, ROTOR_SECT_Y_MOMENT, ROTOR_SECT_Z_MOMENT

3.8.2.12 int glvRotorAnalysisData::ReleaseData ()

Release internal data

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

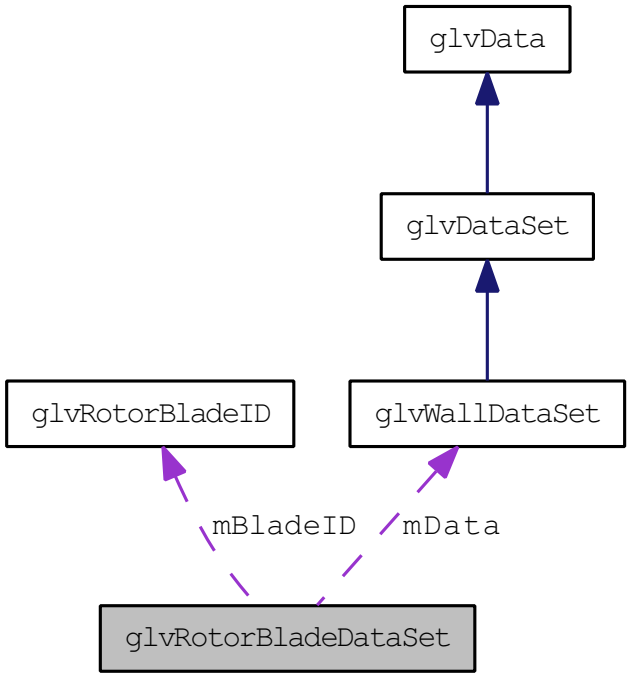
The documentation for this class was generated from the following files:

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

3.9 glvRotorBladeDataSet Class Reference

glvRotorBladeDataSet (p. 49) is a class for rotor blade dataset

`#include <glvRotorBladeDataSet.h>`Collaboration diagram for glvRotorBladeDataSet:



Public Member Functions

- int **Register** (**glvWallDataSet** *wall)
- int **Register** (**glvWallDataSet** *wall, int num)
- int **Register** (**glvRotorBladeDataSet** *dataset)
- int **RegisterBlade** (**glvWallDataSet** *wall)
- int **RegisterBlade** (**glvRotorBladeDataSet** *data)
- virtual int **Delete** ()

  Delete rotor blade dataset.

- int **RemoveRotorBlade** (int Id)
- int **RemoveAllRotorBlades** ()
- int **SetNumberOfBladeBlocks** (int number)
- int **GetNumberOfBlades** ()

  Get number of blades.

- int **GetNumberOfBladeBlocks** ()
- int **GetNumberOfTimeSteps** (int Id)

- int **GetNumberOfTimeSteps** ()

  Get number of time steps.

- int **GetCurrentTimeStep** ()

  Get current time step.

- int **SetCurrentTimeStep** (int step)
- int **SetNextTimeStep** ()
- int **SetPrevTimeStep** ()
- **glvWallDataSet** ∗ **GetWallDataSet** ()
- **glvWallDataSet** ∗ **GetWallDataSet** (int step)
- **glvWallDataSet** ∗ **GetBladeWallDataSet** (int Id)
- **glvWallDataSet** ∗ **GetBladeWallDataSet** (int Id, int step)
- **glvRotorBladeDataSet** ∗ **GetBlade** (int Id)
- **glvRotorBladeDataSet** ∗ **GetBlade** (int Id, int step)
- double **GetPressure** (int step, int Id, double spanPosition, int face)
- double **GetPressure** (int step, double x, double y, double z)
- double **GetPressure** (int step, double point[3])
- int **GetPressureRange** (int step, double range[2])
- int **GetPressureRangeWholeTimeStep** (double range[2])
- int **GetBladePressureRange** (int step, int Id, double range[2])
- int **GetBladePressureRangeWholeTimeStep** (int Id, double range[2])

Static Public Member Functions

- static **glvRotorBladeDataSet** ∗ **New** ()

  Create empty dataset object.

- static **glvRotorBladeDataSet** ∗ **New** (**glvWallDataSet** ∗wall)
- static **glvRotorBladeDataSet** ∗ **New** (**glvWallDataSet** ∗wall, int num)

Private Member Functions

- int **CalcNumberOfBlades** ()

Private Attributes

- int **mNumberOfBlades**

  number of blades

- **glvRotorBladeID** ∗ **mBladeID**

  ID of rotor blades.

- **glvWallDataSet * mData**

  Internal wall dataset.

3.9.1 Detailed Description

glvRotorBladeDataSet (p. 49) is a class for rotor blade dataset

3.9.2 Member Function Documentation

3.9.2.1 int glvRotorBladeDataSet::CalcNumberOfBlades () [private]

Calculate number of blades using number of registered wall dataset and number of blocks in blade.

3.9.2.2 glvRotorBladeDataSet* glvRotorBladeDataSet::GetBlade (int *Id*, int *step*)

Return a blade of given ID and time step as **glvRotorBladeDataSet** (p. 49)

Parameters:

Id blade ID

step time step

Returns:

NULL if fails

3.9.2.3 glvRotorBladeDataSet* glvRotorBladeDataSet::GetBlade (int *Id*)

Return a blade of given ID as **glvRotorBladeDataSet** (p. 49)

Parameters:

Id blade ID

Returns:

NULL if fails

3.9.2.4 int glvRotorBladeDataSet::GetBladePressureRange (int *step*, int *Id*, double *range*[2])

Return range of pressure variable on the given blade at the given time step

Parameters:

step time step

Id blade ID

range array to store range

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.9.2.5 int glvRotorBlade-DataSet::GetBladePressureRangeWholeTimeStep (int *Id*, double *range*[2])

Return range of pressure variable on the given blade across entire time steps

Parameters:

Id blade ID

range array to store range

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.9.2.6 glvWallDataSet * glvRotorBlade-DataSet::GetBladeWallDataSet (int *Id*, int *step*)

Return a blade of given ID and time step as **glvWallDataSet** (p. 78)

Parameters:

Id blade ID

step time step

Returns:

NULL if fails

3.9.2.7 glvWallDataSet * glvRotorBlade-DataSet::GetBladeWallDataSet (int *Id*)

Return a blade of given ID as **glvWallDataSet** (p. 78)

Parameters:

Id blade ID

Returns:

NULL if fails

3.9.2.8 int glvRotorBladeDataSet::GetNumberOfBladeBlocks ()

Get number of blocks which contain a single blade

Returns:

GLV_FALSE if number of blocks in blade is not same.

3.9.2.9 int glvRotorBladeDataSet::GetNumberOfTimeSteps (int *Id*)

Get number of time steps of given blade

Parameters:

Id blade ID

3.9.2.10 double glvRotorBladeDataSet::GetPressure (int *step*, double *point*[3])

Return pressure at given temporal and spatial location

Parameters:

step time step

point position on VTK global coordinates

Returns:

-1 if fails

3.9.2.11 double glvRotorBladeDataSet::GetPressure (int *step*, double *x*, double *y*, double *z*)

Return pressure at given temporal and spatial location

Parameters:

step time step

x X coordinate of the point location

y Y coordinate of the point location

z Z coordinate of the point location

Returns:

-1 if fails

3.9.2.12 double glvRotorBladeDataSet::GetPressure (int *step*, int *Id*, double *spanPosition*, int *face*)

Return pressure of given span position on the given blade. If multiple grid points which store pressure exist, add them. It is a kind of integration.

Parameters:

step time step

Id blade ID

spanPosition span position

face front or back face

Returns:

-1 if fails

3.9.2.13 int glvRotorBladeDataSet::GetPressureRange (int *step*, double *range*[2])

Return range of pressure variable at given time step

Parameters:

step time step

range array to store range

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.9.2.14 int glvRotorBlade-DataSet::GetPressureRangeWholeTimeStep (double *range*[2])

Return range of pressure variable across entire time steps

Parameters:

range array to store range

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.9.2.15 glvWallDataSet* glvRotorBladeDataSet::GetWallDataSet (int *step*)

Return blade dataset of given time step as **glvWallDataSet** (p. 78)

Parameters:

step time step

Returns:

NULL if fails

3.9.2.16 glvWallDataSet * glvRotorBladeDataSet::GetWallDataSet ()

Return entire blade dataset registered as **glvWallDataSet** (p. 78)

Returns:

NULL if fails

3.9.2.17 glvRotorBladeDataSet * glvRotorBladeDataSet::New (glvWallDataSet * *wall*, int *num*) [static]

Create empty dataset object using given **glvWallDataSet** (p. 78)

Parameters:

wall wall data to register

num number of wall dataset consits a single balde

3.9.2.18 glvRotorBladeDataSet * glvRotorBladeDataSet::New (glvWallDataSet * *wall*) [static]

Create rotor blade dataset object using given **glvWallDataSet** (p. 78) Each wall dataset consists a single blade structure

Parameters:

wall wall data to register

3.9.2.19 int glvRotorBladeDataSet::Register (glvRotorBladeDataSet * *dataset*)

Register rotor blade dataset. Replace if alreay registered.

Parameters:

dataset **glvRotorBladeDataSet** (p. 49) which have **glvWallDataSet** (p. 78) to register

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.9.2.20 int glvRotorBladeDataSet::Register (glvWallDataSet * *wall*, int *num*)

Register rotor blade dataset. Replace if alreay registered.

Parameters:

wall wall data to register

num number of wall dataset consits a single balde

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.9.2.21 int glvRotorBladeDataSet::Register (glvWallDataSet * *wall*)

Register rotor blade dataset. Replace if alreay registered.

Parameters:

wall wall data to register

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.9.2.22 int glvRotorBladeDataSet::RegisterBlade (glvRotorBladeDataSet * *data*)

Register a single rotor blade. Returns blade ID

Parameters:

data **glvRotorBladeDataSet** (p. 49) which have **glvWallDataSet** (p. 78) to register. Number of time steps and data blocks must be same with already registered one. The data must be a single blade.

Returns:

blade ID if succeeds, GLV_FALSE if fails

3.9.2.23 int glvRotorBladeDataSet::RegisterBlade (glvWallDataSet * *wall*)

Register a single rotor blade. Returns blade ID

Parameters:

wall wall data to register as a blade. Number of time steps and data blocks must be same with already registered one. The data must be a single blade.

Returns:

blade ID if succeeds, GLV_FALSE if fails

3.9.2.24 int glvRotorBladeDataSet::RemoveAllRotorBlades ()

Remove all registered rotor blades

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.9.2.25 int glvRotorBladeDataSet::RemoveRotorBlade (int *Id*)

Remove registred rotor blade

Parameters:

Id blade ID to remove

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.9.2.26 int glvRotorBladeDataSet::SetCurrentTimeStep (int *step*)

Set current time step

Parameters:

step time step

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.9.2.27 int glvRotorBladeDataSet::SetNextTimeStep ()

Proceed to next time step

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.9.2.28 int glvRotorBladeDataSet::SetNumberOfBladeBlocks (int *number*)

Set number of blocks which constists a single blade. Number of blocks in every blade are changed with given number.

Parameters:

number number of blocks in a single blade

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.9.2.29 int glvRotorBladeDataSet::SetPrevTimeStep ()

Back to previous time step

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

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

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

3.10 glvRotorBladeID Class Reference

glvRotorBladeID (p. 59) is a class of ID of rotor blade

`#include <glvRotorBladeDataSet.h>`

Public Member Functions

- int **GetNewID** (int block)
- int **GetBladeID** (int block)
- int **GetBlockID** (int blade)
- int **RemoveID** (int ID)

Private Member Functions

- int **SearchID** (int ID)
- int **GetFirstEmptyID** ()

Private Attributes

- int **mNumberOfIDs**

  Number of blade IDs.

- int **mID** [GLV_MAX_ID]

3.10.1 Detailed Description

glvRotorBladeID (p. 59) is a class of ID of rotor blade

3.10.2 Member Function Documentation

3.10.2.1 int glvRotorBladeID::GetBladeID (int *block*)

Return blade ID of given data block

Parameters:

block block ID

Returns:

-1 if fails

3.10.2.2 int glvRotorBladeID::GetBlockID (int *blade*)

Return block ID of given blade

Parameters:

blade blade ID

Returns:

-1 if fails

3.10.2.3 int glvRotorBladeID::GetFirstEmptyID () [private]

Return first located unused ID

Returns:

-1 if fails

3.10.2.4 int glvRotorBladeID::GetNewID (int *block*)

Create new ID;

Parameters:

block block ID

Returns:

-1 if fails

3.10.2.5 int glvRotorBladeID::RemoveID (int *ID*)

Remove given blade ID

Parameters:

ID blade ID

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.10.2.6 int glvRotorBladeID::SearchID (int *ID*) [private]

Search blade ID using block ID

Parameters:

ID block ID

Returns:

-1 if fails, blade ID if succeeds

3.10.3 Member Data Documentation

3.10.3.1 int glvRotorBladeID::mID[GLV_MAX_ID] [private]

internal block IDs of corresponding blade IDs. Blade ID is an index of an array

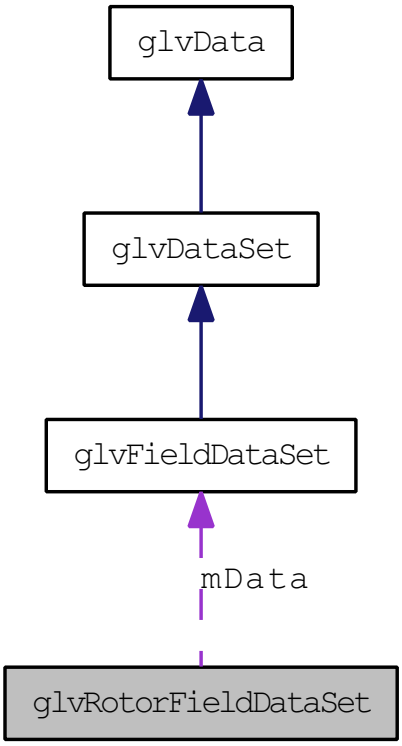
The documentation for this class was generated from the following files:

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

3.11 glvRotorFieldDataSet Class Reference

glvRotorFieldDataSet (p. 62) is a class for rotor field dataset

`#include <glvRotorFieldDataSet.h>`Collaboration diagram for glvRotorFieldDataSet:



Public Member Functions

- int **Register** (**glvRotorFieldDataSet** *data)
- int **Register** (**glvRotorFieldDataSet** *data, int step)
- int **Register** (**glvFieldDataSet** *data)
- int **Register** (**glvFieldDataSet** *data, int step)
- int **CreateVorticity** ()
- int **CreateQCriteria** ()
- virtual int **Delete** ()

  Delete rotor field dataset.

- int **GenVelocityMagnitude** ()
- int **GenVorticityMagnitude** ()
- int **GetNumberOfTimeSteps** ()

  Get number of time steps.

- int **GetCurrentTimeStep** ()

  Get current time step.

- int **SetCurrentTimeStep** (int step)
- int **SetNextTimeStep** ()
- int **SetPrevTimeStep** ()
- **glvFieldDataSet** * **GetFieldDataSet** ()
- **glvFieldDataSet** * **GetFieldDataSet** (int step)
- **glvRotorFieldDataSet** * **GetSingleTimeStep** (int step)
- **glvRotorFieldDataSet** * **GetBlock** (int block, int step)
- double **GetPressure** (int step, double x, double y, double z)
- double **GetPressure** (int step, double point[3])
- double * **GetPressureRange** ()
- int **GetPressureRange** (double range[2])
- double * **GetPressureRange** (int step)
- int **GetPressureRange** (int step, double range[2])
- double **GetDensity** (int step, double x, double y, double z)
- double **GetDensity** (int step, double point[3])
- double * **GetDensityRange** ()
- int **GetDensityRange** (double range[2])
- double * **GetDensityRange** (int step)
- int **GetDensityRange** (int step, double range[2])
- double * **GetVelocityVector** (int step, double x, double y, double z)
- double * **GetVelocityVector** (int step, double point[3])
- double **GetVelocityMagnitude** (int step, double x, double y, double z)
- double **GetVelocityMagnitude** (int step, double point[3])
- double * **GetVelocityMagnitudeRange** ()
- int **GetVelocityMagnitudeRange** (double range[2])
- double * **GetVelocityMagnitudeRange** (int step)
- int **GetVelocityMagnitudeRange** (int step, double range[2])
- double * **GetVorticityVector** (int step, double x, double y, double z)
- double * **GetVorticityVector** (int step, double point[3])
- double **GetVorticityMagnitude** (int step, double x, double y, double z)
- double **GetVorticityMagnitude** (int step, double point[3])
- double * **GetVorticityMagnitudeRange** ()
- int **GetVorticityMagnitudeRange** (double range[2])
- double * **GetVorticityMagnitudeRange** (int step)
- int **GetVorticityMagnitudeRange** (int step, double range[2])
- double **GetMachNumber** (int step, double x, double y, double z)
- double **GetMachNumber** (int step, double point[3])
- double ** **GetPointsByVelocity** (double percent, int *size)
- double ** **GetPointsByVorticity** (double percent, int *size)
- double ** **GetPointsByQCriteria** (double percent, int *size)

Static Public Member Functions

- static **glvRotorFieldDataSet** * **New** ()

  Create empty dataset object.

Protected Attributes

- **glvFieldDataSet** * **mData**

  internal ***glvFieldDataSet*** *(p. 38)*

3.11.1 Detailed Description

glvRotorFieldDataSet (p. 62) is a class for rotor field dataset

3.11.2 Member Function Documentation

3.11.2.1 int glvRotorFieldDataSet::CreateQCriteria ()

Calculate and add Q-Criteria into the registered field dataset. The field dataset must have "Vorticity" vectors. If "Q-Criteria" is already registered, replace it.

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails.

3.11.2.2 int glvRotorFieldDataSet::CreateVorticity ()

Calculate and add vorticity and Q-Criteria into the registered field dataset. The field dataset must have "Velocity" vectors. If "Vorticity" or "Q-Criteria" is already registered, replace it.

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails.

3.11.2.3 int glvRotorFieldDataSet::GenVelocityMagnitude ()

Calculate and add velocity magnitude into the registered field dataset. The field dataset must have "Velocity" vectors. If already registered, replace it.

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails.

3.11.2.4 int glvRotorFieldDataSet::GenVorticityMagnitude ()

Calculate and add vorticity magnitude into the registered field dataset. The field dataset must have "Vorticity" vectors. If already registered, replace it.

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails.

3.11.2.5 glvRotorFieldDataSet* glvRotorFieldDataSet::GetBlock (int *block*, int *step*)

Return a single block of field dataset of given time step as **glvRotorFieldDataSet** (p. 62)

Parameters:

block block ID

step time step

Returns:

NULL if fails

3.11.2.6 double glvRotorFieldDataSet::GetDensity (int *step*, double *point*[3])

Return density at given temporal and spatial location

Parameters:

step time step

point position on VTK global coorinates

Returns:

-1 if fails.

3.11.2.7 double glvRotorFieldDataSet::GetDensity (int *step*, double *x*, double *y*, double *z*)

Return density at given temporal and spatial location

Parameters:

step time step

x X cooridnate of the point location

y Y cooridnate of the point location

z Z cooridnate of the point location

Returns:

-1 if fails.

3.11.2.8 int glvRotorFieldDataSet::GetDensityRange (int *step*, double *range*[2])

Return range of density in the field dataset of given time step

Parameters:

step time step

range array to store range

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.11.2.9 double* glvRotorFieldDataSet::GetDensityRange (int *step*)

Return range of density in the field dataset of given time step

Parameters:

step time step

Returns:

NULL if fails.

3.11.2.10 int glvRotorFieldDataSet::GetDensityRange (double *range*[2])

Return range of density in entire field dataset

Parameters:

range storage for pressure range to be returned

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.11.2.11 double * glvRotorFieldDataSet::GetDensityRange ()

Return range of density in entire field dataset

Returns:

double array of size 2 is allocated and returned. NULL if fails.

3.11.2.12 glvFieldDataSet* glvRotorFieldDataSet::GetFieldDataSet (int *step*)

Return field dataset of given time step as **glvFieldDataSet** (p. 38)

Parameters:

step time step

Returns:

NULL if fails

3.11.2.13 glvFieldDataSet * glvRotorFieldDataSet::GetFieldDataSet ()

Return entire field dataset as **glvFieldDataSet** (p. 38)

Returns:

NULL if fails

3.11.2.14 double glvRotorFieldDataSet::GetMachNumber (int *step*, double *point*[3])

Return velocity magnitude at given temporal and spatial location as a mach number

Parameters:

step time step

point position on VTK global coorinates

Returns:

-1 if fails.

3.11.2.15 double glvRotorFieldDataSet::GetMachNumber (int *step*, double *x*, double *y*, double *z*)

Return velocity magnitude at given temporal and spatial location as a mach number

Parameters:

step time step

x X cooridnate of the point location

y Y cooridnate of the point location

z Z cooridnate of the point location

Returns:

-1 if fails.

3.11.2.16 double ** glvRotorFieldDataSet::GetPointsByQCriteria (double *percent*, int * *size*)

Create set of points where Q-Criteria is bigger than given percent

Parameters:

percent percent

size pointer to store size of point array to return

Returns:

2D arrray of points. Size of array is *size * 3. NULL if fails

3.11.2.17 double ** glvRotorFieldDataSet::GetPointsByVelocity (double *percent*, int * *size*)

Create set of points where velocity is bigger than given percent

Parameters:

percent percent

size pointer to store size of point array to return

Returns:

2D arrray of points. Size of array is *size * 3. NULL if fails

3.11.2.18 double ** glvRotorFieldDataSet::GetPointsByVorticity (double *percent*, int * *size*)

Create set of points where vorticity is bigger than given percent

Parameters:

percent percent

size pointer to store size of point array to return

Returns:

2D arrray of points. Size of array is *size * 3. NULL if fails

3.11.2.19 double glvRotorFieldDataSet::GetPressure (int *step*, double *point*[3])

Return pressure at given temporal and spatial location

Parameters:

step time step

point position on VTK global coorinates

Returns:

-1 if fails.

3.11.2.20 double glvRotorFieldDataSet::GetPressure (int *step*, double *x*, double *y*, double *z*)

Return pressure at given temporal and spatial location

Parameters:

step time step

x X cooridnate of the point location

y Y cooridnate of the point location

z Z cooridnate of the point location

Returns:

-1 if fails.

3.11.2.21 int glvRotorFieldDataSet::GetPressureRange (int *step*, double *range*[2])

Return range of pressure in the field dataset of given time step

Parameters:

step time step

range array to store range

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.11.2.22 double* glvRotorFieldDataSet::GetPressureRange (int *step*)

Return range of pressure in the field dataset of given time step

Parameters:

step time step

Returns:

NULL if fails.

3.11.2.23 int glvRotorFieldDataSet::GetPressureRange (double *range*[2])

Return range of pressure in entire field dataset

Parameters:

range storage for pressure range to be returned

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.11.2.24 double * glvRotorFieldDataSet::GetPressureRange ()

Return range of pressure in entire field dataset

Returns:

double array of size 2 is allocated and returned. NULL if fails.

3.11.2.25 glvRotorFieldDataSet* glvRotorFieldDataSet::GetSingleTimeStep (int *step*)

Return field dataset of given time step as **glvRotorFieldDataSet** (p. 62)

Parameters:

step time step

Returns:

NULL if fails

3.11.2.26 double glvRotorFieldDataSet::GetVelocityMagnitude (int *step*, double *point*[3])

Return velocity magnitude at given temporal and spatial location

Parameters:

step time step

point position on VTK global coorinates

Returns:

-1 if fails.

3.11.2.27 double glvRotorFieldDataSet::GetVelocityMagnitude (int *step*, double *x*, double *y*, double *z*)

Return velocity magnitude at given temporal and spatial location

Parameters:

step time step

x X cooridnate of the point location

y Y cooridnate of the point location

z Z cooridnate of the point location

Returns:

-1 if fails.

3.11.2.28 int glvRotorFieldDataSet::GetVelocityMagnitudeRange (int *step*, double *range*[2])

Return range of velocity magnitude in the field dataset of given time step

Parameters:

step time step

range array to store range

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.11.2.29 double* glvRotorFieldDataSet::GetVelocityMagnitudeRange (int *step*)

Return range of velocity magnitude in the field dataset of given time step

Parameters:

step time step

Returns:

NULL if fails.

3.11.2.30 int glvRotorFieldDataSet::GetVelocityMagnitudeRange (double *range*[2])

Return range of velocity magnitude in entire field dataset

Parameters:

range storage for pressure range to be returned

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.11.2.31 double * glvRotorFieldDataSet::GetVelocityMagnitudeRange ()

Return range of velocity magnitude in entire field dataset

Returns:

double array of size 2 is allocated and returned. NULL if fails.

3.11.2.32 double* glvRotorFieldDataSet::GetVelocityVector (int *step*, double *point*[3])

Return velocity vector at given temporal and spatial location

Parameters:

step time step

point position on VTK global coorinates

Returns:

double array[3], NULL if fails.

3.11.2.33 double* glvRotorFieldDataSet::GetVelocityVector (int *step*, double *x*, double *y*, double *z*)

Return velocity vector at given temporal and spatial location

Parameters:

step time step

x X cooridnate of the point location

y Y cooridnate of the point location

z Z cooridnate of the point location

Returns:

double array[3], NULL if fails.

3.11.2.34 double glvRotorFieldDataSet::GetVorticityMagnitude (int *step*, double *point*[3])

Return vorticity magnitude at given temporal and spatial location

Parameters:

step time step

point position on VTK global coorinates

Returns:

-1 if fails.

3.11.2.35 double glvRotorFieldDataSet::GetVorticityMagnitude (int *step*, double *x*, double *y*, double *z*)

Return vorticity magnitude at given temporal and spatial location

Parameters:

step time step

x X cooridnate of the point location

y Y cooridnate of the point location

z Z cooridnate of the point location

Returns:

-1 if fails.

3.11.2.36 int glvRotorFieldDataSet::GetVorticityMagnitudeRange (int *step*, double *range*[2])

Return range of vorticity magnitude in the field dataset of given time step

Parameters:

step time step

range array to store range

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.11.2.37 double* glvRotorFieldDataSet::GetVorticityMagnitudeRange (int *step*)

Return range of vorticity magnitude in the field dataset of given time step

Parameters:

step time step

Returns:

NULL if fails.

3.11.2.38 int glvRotorFieldDataSet::GetVorticityMagnitudeRange (double *range*[2])

Return range of vorticity magnitude in entire field dataset

Parameters:

range storage for pressure range to be returned

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.11.2.39 double* glvRotorFieldDataSet::GetVorticityMagnitudeRange ()

Return range of vorticity magnitude in entire field dataset

Returns:

double array of size 2 is allocated and returned. NULL if fails.

3.11.2.40 double* glvRotorFieldDataSet::GetVorticityVector (int *step*, double *point*[3])

Return vorticity vector at given temporal and spatial location

Parameters:

step time step

point position on VTK global coorinates

Returns:

double array[3], NULL if fails.

3.11.2.41 double* glvRotorFieldDataSet::GetVorticityVector (int *step*, double *x*, double *y*, double *z*)

Return vorticity vector at given temporal and spatial location

Parameters:

step time step

x X cooridnate of the point location

y Y cooridnate of the point location

z Z cooridnate of the point location

Returns:

double array[3], NULL if fails.

3.11.2.42 int glvRotorFieldDataSet::Register (glvFieldDataSet * *data*, int *step*)

Register rotor field dataset at given temporal location.

Parameters:

data **glvFieldDataSet** (p. 38) to register

step time step

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails.

3.11.2.43 int glvRotorFieldDataSet::Register (glvFieldDataSet * *data*)

Register rotor field dataset. Replace if already registered.

Parameters:

data **glvFieldDataSet** (p. 38) to register

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails.

3.11.2.44 int glvRotorFieldDataSet::Register (glvRotorFieldDataSet * *data*, int *step*)

Register rotor field dataset at given temporal location. Replace if already registered.

Parameters:

data **glvRotorFieldDataSet** (p. 62) contains **glvFieldDataSet** (p. 38) to register.

step time step

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails.

3.11.2.45 int glvRotorFieldDataSet::Register (glvRotorFieldDataSet * *data*)

Register rotor field dataset. Replace if already registered.

Parameters:

data **glvRotorFieldDataSet** (p. 62) contains **glvFieldDataSet** (p. 38) to register.

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails.

3.11.2.46 int glvRotorFieldDataSet::SetCurrentTimeStep (int *step*)

Set current time step

Parameters:

step time step

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails.

3.11.2.47 int glvRotorFieldDataSet::SetNextTimeStep ()

Proceed to next time step

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails.

3.11.2.48 int glvRotorFieldDataSet::SetPrevTimeStep ()

Back to the previous time step

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails.

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

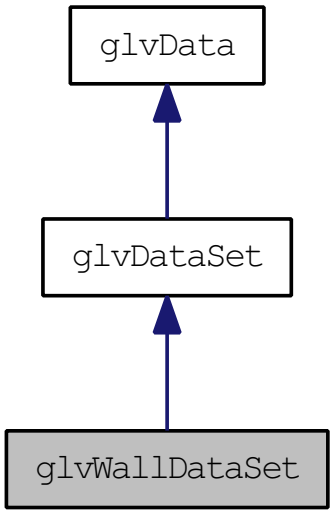
- /home/osung/tmp/glove/trunk/include/rotor/glvRotorFieldDataSet.h
- /home/osung/tmp/glove/trunk/src/rotor/glvRotorFieldDataSet.cc

3.12 glvWallDataSet Class Reference

glvWallDataSet (p. 78) is a class for wall dataset. This class is a concrete implementation of **glvDataSet** (p. 19)

`#include <glvWallDataSet.h>`

Inherits **glvDataSet**.Collaboration diagram for glvWallDataSet:



Public Member Functions

- void **Delete** ()
  Delete an object.
- int **RegisterTimeStep** (**glvWallDataSet** *dataset, int timeStep)
- int **MergeWallDataSet** (**glvWallDataSet** *dataset)
- int **AddTimeStep** (**glvWallDataSet** *data)
- int **SetDisplayVariable** (char *variableName)
- int **SetDisplayVariable** (int id)
- char * **GetDisplayFieldAsName** ()
- int **GetDisplayFieldAsID** ()
- virtual **glvWallDataSet** * **GetBlock** (int block, int step)
- virtual **glvWallDataSet** * **GetSingleTimeStep** (int step)
- virtual **glvWallDataSet** * **GetTemporalBlock** (int block)

Static Public Member Functions

- static **glvWallDataSet** * **New** ()
- static **glvWallDataSet** * **New** (int steps)
- static **glvWallDataSet** * **New** (vtkDataObject *data)

Protected Attributes

- char **mDisplayVariable** [256]
  Display variable. Stored as a character string.

3.12.1 Detailed Description

glvWallDataSet (p. 78) is a class for wall dataset. This class is a concrete implementation of **glvDataSet** (p. 19)

3.12.2 Member Function Documentation

3.12.2.1 int glvWallDataSet::AddTimeStep (glvWallDataSet * *data*)

Add a single time step dataset at the end of the time step

Parameters:

data wall dataset to add

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

Reimplemented from **glvDataSet** (p. 19).

3.12.2.2 glvWallDataSet * glvWallDataSet::GetBlock (int *block*, int *step*) [virtual]

Return data block of the given time step and block ID

Parameters:

block ID of block

step time step

Returns:

NULL if fails

Implements **glvDataSet** (p. 26).

3.12.2.3 int glvWallDataSet::GetDisplayFieldAsID ()

Get display field as a integer ID

Returns:

-1 if fails

3.12.2.4 char* glvWallDataSet::GetDisplayFieldAsName ()

Get display field as a name

Returns:

NULL if fails

3.12.2.5 glvWallDataSet * glvWallDataSet::GetSingleTimeStep (int *step*) [virtual]

Return dataset of the given time step

Parameters:

step time step

Returns:

NULL if fails

Implements **glvDataSet** (p. 31).

3.12.2.6 glvWallDataSet * glvWallDataSet::GetTemporalBlock (int *block*) [virtual]

Return temporal block of the given block ID

Parameters:

block ID of block

Returns:

NULL if fails

Implements **glvDataSet** (p. 31).

3.12.2.7 int glvWallDataSet::MergeWallDataSet (glvWallDataSet * *dataset*)

Merge input **glvWallDataSet** (p. 78) Every single vtk dataset of same time step and same block ID are merged

Parameters:

dataset wall dataset to merge with internal vtk dataset

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.12.2.8 glvWallDataSet * glvWallDataSet::New (vtkDataObject * *data*) [static]

Create an dataset object from the input vtk object

Parameters:

data internal vtk object

See also:

New() (p. 81)
New(int) (p. 81)

3.12.2.9 glvWallDataSet * glvWallDataSet::New (int *steps*) [static]

Create an dataset of specified time steps

Parameters:

steps number of time steps

See also:

New() (p. 81)
New(vtkDataObject *) (p. 81)

3.12.2.10 glvWallDataSet * glvWallDataSet::New () [static]

Create an empty object;

See also:

New(int) (p. 81)
New(vtkDataObject *) (p. 81)

3.12.2.11 int glvWallDataSet::RegisterTimeStep (glvWallDataSet * *dataset*, int *timeStep*)

Register a single time step data set into the specified temporal location. If already registered, replace it.

Parameters:

dataset wall dataset to register

timeStep time step to be registered

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

Reimplemented from **glvDataSet** (p. 19).

3.12.2.12 int glvWallDataSet::SetDisplayVariable (int *id*)

Set display field as an integer ID. Default is the first scalar variable.

Parameters:

id ID of scalar to display at the surface of wall

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

3.12.2.13 int glvWallDataSet::SetDisplayVariable (char * *variableName*)

Set display field as a char string. Default is the first scalar variable.

Parameters:

variableName name of scalar to display at the surface of wall

Returns:

GLV_TRUE if succeeds, GLV_FALSE if fails

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

- /home/osung/tmp/glove/trunk/include/common/glvWallDataSet.h
- /home/osung/tmp/glove/trunk/src/common/glvWallDataSet.cc

Index