ISBN 978-89-6211-397-6

로터 데이터 분석 및 가시화

(Analysis and Visualization of Rotor Data)

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

1. 서론 ····· 1

2. 로터 동역학 데이터 ····· 2
가. 듬성한 격자 로터 데이터 ····· 2
나. 조밀한 격자 로터 데이터 ····· 4
다. 동체 데이터 ····· 5
라. Vortex 분석 ····· 7
1) Vorticity ····· 7
2) Vortex 영역 ····· 7

3. VTK 구현 ····· 9
가. 데이터 구조 ····· 9
나. 데이터 가시화 ····· 10
1) 블레이드 표면에의 압력 분포 ····· 11
2) Vorticity 및 Q-Criteria Iso-surface 생성 ····· 12
3) 스트림라인 생성 ····· 15
4) 키메라 격자 구조 ····· 17
5) Transfer Function 생성 ····· 17

4. 구현 결과 ····· 21

5. 결론 ····· 23

6. 참고문헌 ····· 24

표 차례

[표 2-1] 로터 동역학 데이터 정보 ····· 2
[표 2-2] 듬성한 격자 블레이드 데이터 크기 정보 ····· 2
[표 2-3] 듬성한 격자 유동장 데이터 크기 정보 ····· 2
[표 2-4] 듬성한 격자 블레이드 데이터 변수 정보 ····· 3
[표 2-5] 듬성한 격자 유동장 데이터 변수 정보 ····· 3
[표 2-6] 조밀한 격자 블레이드 데이터 크기 정보 ····· 4
[표 2-7] 조밀한 격자 유동장 데이터 크기 정보 ····· 4
[표 2-8] 조밀한 격자 블레이드 데이터 변수 정보 ····· 4
[표 2-9] 조밀한 격자 유동장 데이터 변수 정보 ····· 4
[표 2-10] 동체 데이터의 벽면 데이터 크기 정보 ····· 5
[표 2-11] 동체 데이터의 유동장 데이터 크기 정보 ····· 6
[표 2-12] 동체 데이터의 벽면 데이터 변수 정보 ····· 6
[표 2-13] 동체 데이터의 유동 데이터 변수 정보 ····· 6

그림 차례

[그림 2-1] 듬성한 격자 로터 데이터 Outline ····· 3
[그림 2-2] 조밀한 격자 로터 데이터 Outline ····· 5
[그림 2-3] 동체 데이터 Outline ····· 6
[그림 3-2] 로터 블레이드 및 동체 데이터에 대한 압력 분포 가시화 ····· 12
[그림 3-3] 및 동체 데이터에 대한 Q-Criteria Iso-Surface 가시화 15
[그림 3-4] 듬성한 격자 및 조밀한 격자 로터 데이터에서의 스트림라인 생성 · 16
[그림 3-5] vtkScalarsToColors 클래스 계층구조 ····· 18
[그림 3-6] Transfer function 적용 결과 비교 ····· 19
[그림 4-1] 듬성한 격자 로터 데이터 가시화 결과 ····· 21
[그림 4-2] 조밀한 격자 로터 데이터 가시화 결과 ····· 22
[그림 4-3] 동체 데이터 가시화 결과 (Q-Criteria와 vorticity Iso-surface) · 22

수식 차례

[수식 2-1] Vorticity 계산 수식 ····· 7
[수식 2-2] Q-Criteria 계산 수식 ····· 8

소스 차례

[소스 3-1] vorticity와 Q-Criteria 값 계산 코드 ····· 14

1. 서론

가시화는 복잡한 과학 현상을 연구할 때 그 의미를 이해하기 위한 매우 중요한 과정 중 하나이다. 다양한 과학 분야에서 가시화 과정을 효율적으로 처리하기 위해 컴퓨터 그래픽스를 활용하는데, 유동 데이터 가시화 역시 여러 방법들이 발표되고 있다[1,2]. 한편, 많은 종류의 유동 데이터 중 로터 동역학 데이터는 최근 많은 연구가 계속되고 있는 분야이다. 로터 동역학은 헬리콥터나 풍력 발전기 개발에 있어서 없어서는 안 되는 중요한 연구인데, 그 거대한 크기로 인해 풍동이나 현장에서 실제로 실험하기 어려운 것이 현실이다. 최근 컴퓨팅 성능의 급격한 발전으로 인해 로터 블레이드 실험을 컴퓨터를 이용해서 수행하려는 시도가 계속되고 있지만 이를 통해 생성되는 거대한 데이터를 빠르게 가시화하기 위한 연구는 전무하다. 특히, 일반적으로 로터 동역학 데이터 자체는 매우 거대한 크기를 가지고 있지만 연구자가 이 데이터에서 보고자 하는 정보는 전체 데이터의 극히 일부일 뿐인데, 이들 정보를 표출하는 작업은 전체 가시화 시간의 대부분을 차지할 정도로 시간을 매우 많이 필요로 한다. 특히 거대한 로터 동역학 데이터의 경우 일반적인 PC에서는 전체 데이터를 메인 메모리에 올릴 수가 없기 때문에 데이터를 여러 부분으로 나누어 작업해야 해서 더욱 많은 시간을 필요로 한다. 본 기술문서에서는 고성능의 가시화 전용 컴퓨터를 이용해서 대용량의 로터 동역학 데이터를 빠르게 가시화하기 위한 방법에 대해 설명한다. 특히, 세가지 종류의 로터 동역학 데이터를 가시화하고 서로 비교하여 보다 거대한 데이터에서의 가시화 전략에 대해 살펴본다.

2. 로터 동역학 데이터

본 기술문서에서 다루는 로터 시뮬레이션 데이터는 총 3가지 종류이다. 이들 모두는 다중 블록(multi-block) 구조로 되어 있고 두 가지는 시변환(time-varian t) 데이터, 나머지 하나는 정적 데이터이다. 모두 로터 블레이드와 그 주변 유동장으로 구성되어 있으나 한 데이터는 동체(fuselage) 데이터도 함께 포함되어 있다. 여러개의 블록들 중 각 블록은 모두 구조화된 격자(structured grid)로 되어 있다. 각 데이터에 대한 정보는 표 2-1과 같다.

[표 2-1] 로터 동역학 데이터 정보

데이터	형태	격자 구조	비고
Coarse	시변환	구조화된 격자	로터 블레이드, 듬성한 격자구조
Fine	정적	구조화된 격자	로터 블레이드, 조밀한 격자구조
Fuselage	시변환	구조화된 격자	로터 블레이드 + 동체

가. 듬성한 격자 로터 데이터

듬성한 격자 (coarse grid) 로터 데이터는 16개의 블록으로 된 블레이드 데이터와 32개의 블록으로 된 유동장 데이터로 구성된 총 90 스텝의 시변환 데이터이다. 각 격자에는 격자 좌표 정보와 함께 여러 스칼라 및 벡터 정보가 저장되었고, 데이터 타입은 모두 배정도 부동소수점(double precision floating point)이다. 블레이드 데이터와 유동장 데이터의 크기 정보는 표 2-2, 2-3과 같고, 각 격자에 할당된 벡터와 스칼라 정보는 표 2-4, 2-5와 같다.

[표 2-2] 듬성한 격자 블레이드 데이터 크기 정보

블록 번호	블록 개수	크기	스텝	MBytes
홀수	8	52 X 43 X 1	90	30.70
짝수	8	52 X 23 X 1	90	16.42
합계	16	-	90	47.12

[표 2-3] 듬성한 격자 유동장 데이터 크기 정보

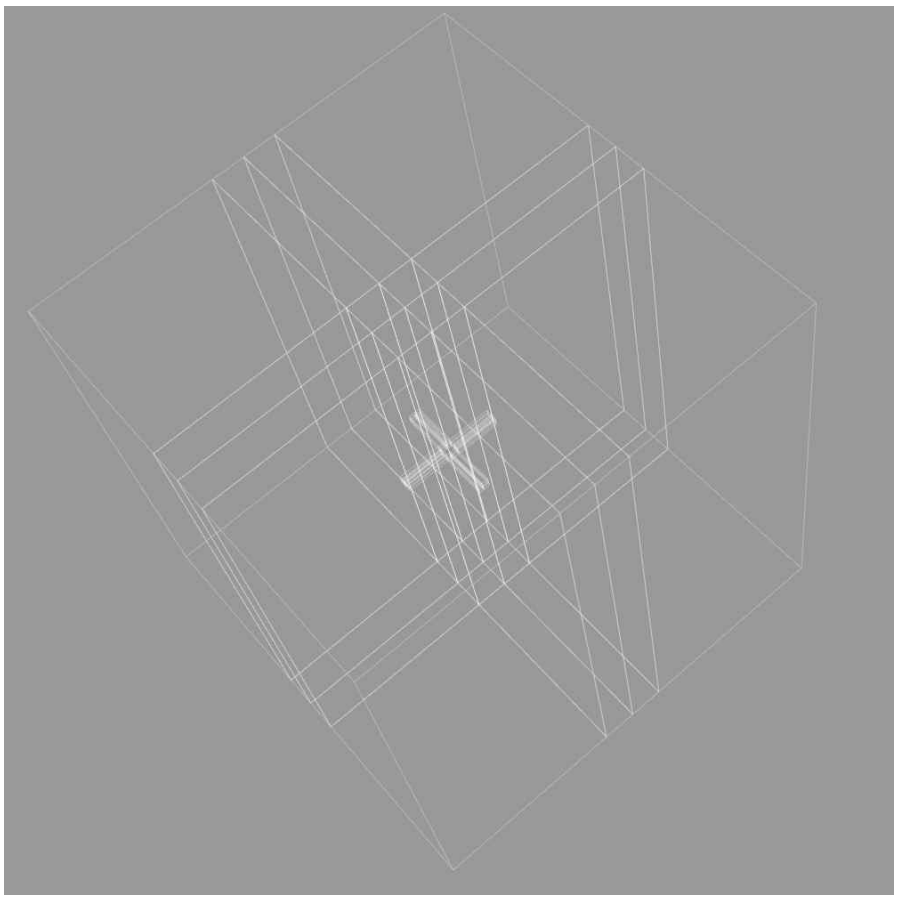
블록 번호	블록 개수	크기	스텝	GBytes
01-16	16	129 X 49 X 49	90	24.92
17-32	16	21 X 73 X 43	90	5.30
합계	32	-	90	30.22

[표 2-4] 듬성한 격자 블레이드 데이터 변수 정보

정보	형식	데이터 타입
Position	벡터 (x, y, z)	double
Blanking	스칼라	double
Pressure	스칼라	double

[표 2-5] 듬성한 격자 유동장 데이터 변수 정보

정보	형식	데이터 타입
Position	벡터 (x, y, z)	double
Blanking	스칼라	double
Density	스칼라	double
Pressure	스칼라	double
Velocity	벡터 (x, y, z)	double
Velocity Magnitude	스칼라	double
Vorticity	벡터 (x, y, z)	double
Vorticity Magnitude	스칼라	double
Q Criteria	스칼라	double



[그림 2-1] 듬성한 격자 로터 데이터 Outline

나. 조밀한 격자 로터 데이터

조밀한 격자 (fine grid) 로터 데이터는 8개의 블록으로 된 블레이드 데이터와 44개의 블록으로 된 유동장 데이터로 구성된 정적 (static) 데이터이다. 각 격자에는 격자 좌표 정보와 함께 여러 스칼라 및 벡터 정보가 저장되었고, 데이터 타입은 모두 배정도 부동소수점(double precision floating point)이다. 블레이드 데이터와 유동장 데이터의 크기 정보는 표 2-6, 2-7과 같고, 각 격자에 할당된 벡터와 스칼라 정보는 표 2-8, 2-9와 같다.

[표 2-6] 조밀한 격자 블레이드 데이터 크기 정보

블록 번호	블록 개수	크기	스텝	MBytes
45-52	8	68 X 65 X 1	1	0.67

[표 2-7] 조밀한 격자 유동장 데이터 크기 정보

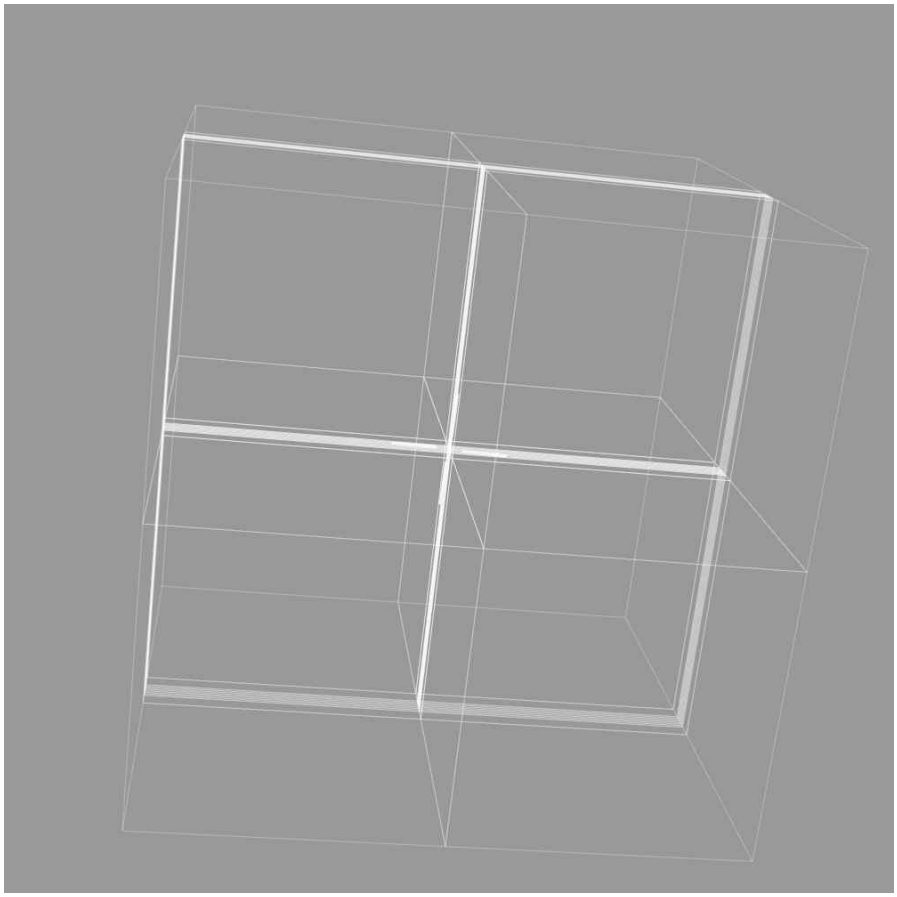
블록 번호	블록 개수	크기	스텝	GBytes
01-44	44	11 X 201 X 201	1	1.09

[표 2-8] 조밀한 격자 블레이드 데이터 변수 정보

정보	형식	데이터 타입
Position	벡터 (x, y, z)	double
Blanking	스칼라	double
Pressure	스칼라	double

[표 2-9] 조밀한 격자 유동장 데이터 변수 정보

정보	형식	데이터 타입
Position	벡터 (x, y, z)	double
Blanking	스칼라	double
Density	스칼라	double
Pressure	스칼라	double
Velocity	벡터 (x, y, z)	double
Velocity Magnitude	스칼라	double
Vorticity	벡터 (x, y, z)	double
Vorticity Magnitude	스칼라	double
Q Criteria	스칼라	double



[그림 2-2] 조밀한 격자 로터 데이터 outline

다. 동체 데이터

동체 데이터는 14개의 블록으로 된 블레이드 및 동체 데이터(이상 벽면 데이터)와 32개의 블록으로 된 유동장 데이터로 구성된 총 30 스텝의 시변환 데이터이다. 각 격자에는 격자 좌표 정보와 함께 여러 스칼라 및 벡터 정보가 저장되었고, 데이터 타입은 모두 배정도 부동소수점(double precision floating point)이다. 벽면 데이터와 유동장 데이터의 크기 정보는 표 2-10, 2-11과 같고, 각 격자에 할당된 벡터와 스칼라 정보는 표 2-12, 2-13과 같다.

[표 2-10] 동체 데이터의 벽면 데이터 크기 정보

블록 번호	블록 개수	크기	스텝	MBytes
73-80	8	67 X 65 X 1	30	19.94
81-82, 85-86	4	129 X 33 X 1	30	2.44
83-84	2	129 X 211 X 1	30	1.56
합계	14	-	30	23.94

[표 2-11] 동체 데이터의 유동장 데이터 크기 정보

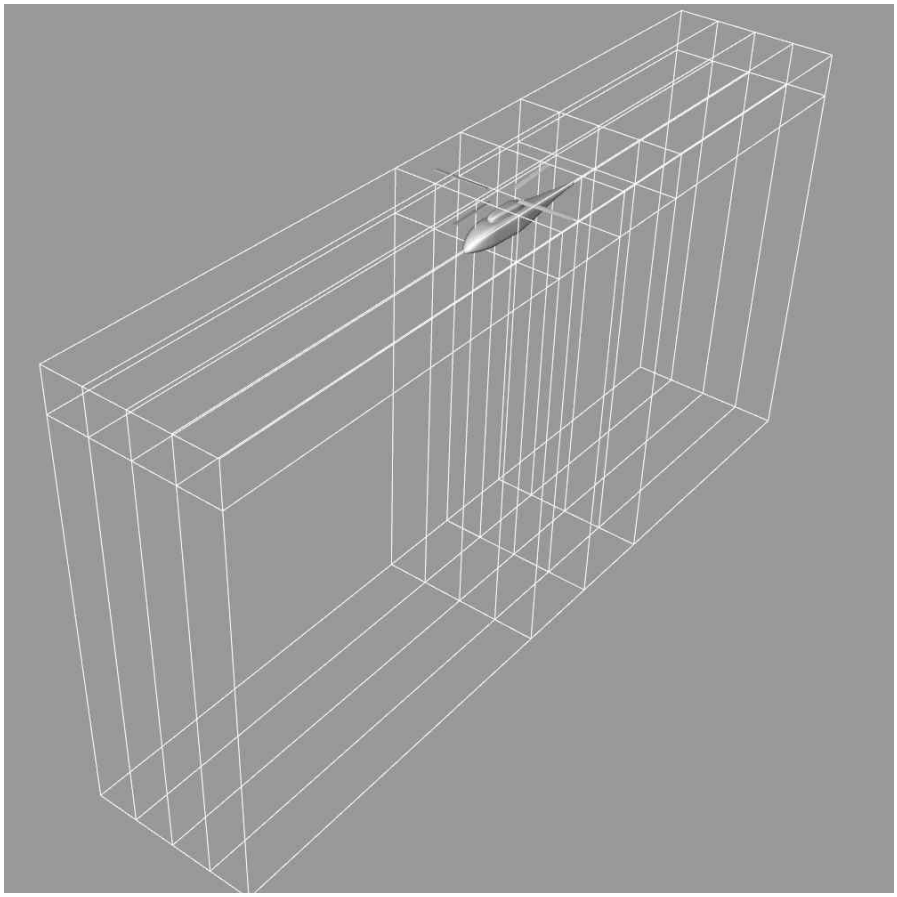
블록 번호	블록 개수	크기	스텝	GBytes
14-45	32	81 X 86 X 41	30	6.13

[표 2-12] 동체 데이터의 벽면 데이터 변수 정보

정보	형식	데이터 타입
Position	벡터 (x, y, z)	double
Blanking	스칼라	double
Pressure	스칼라	double

[표 2-13] 동체 데이터의 유동 데이터 변수 정보

정보	형식	데이터 타입
Position	벡터 (x, y, z)	double
Blanking	스칼라	double
Vorticity Magnitude	스칼라	double
Q Criteria	스칼라	double



[그림 2-3] 동체 데이터 Outline

라. Vortex 분석

로터 데이터에서 가장 중요하게 다루는 부분은 vortex 가시화이다. Vortex는 유동 데이터에서 유동이 급격하게 회전하는 부분을 말하며 회전하는 정도를 vorticity라고 한다. 일반적으로 vortex 영역은 vorticity가 크지만 vorticity가 크다고 전부 vortex인 것은 아니다. 많은 관련 연구자들이 vortex의 정의를 찾기 위해 노력하고 있지만 명확한 정의는 아직까지 만들어지지 않았다. 위에서 설명한 로터 동역학 데이터에는 vorticity와 vorticity magnitude, Q Criteria 등의 vortex 관련 변수들이 저장되어 있다. 그러나 본래 사용자로부터 받은 데이터에는 이러한 변수들이 존재하지 않았고 따로 후처리를 통해 생성했다. 이번 장에서는 이러한 vortex 관련 변수들의 의미와 생성 방법에 대해 설명한다.

1) Vorticity

Vorticity는 vector의 회전하려는 정도를 나타내는 값으로 수학적으로 curl로 표현된다. 이는 vector가 *(u, v, w)*라고 할 때, 다음과 같은 수식으로 표현할 수 있다.

$$vorticity = curl\,v = \nabla \times v = \begin{vmatrix} i & j & k \\ \frac{\partial}{\partial x} & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} \\ u & v & w \end{vmatrix} = \left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}\right)i + \left(\frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}\right)j + \left(\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}\right)k$$

[수식 2-1] Vorticity 계산 수식

여기서, vorticity vector는 $\left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}, \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}, \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}\right)$ 이고, vorticity magnitude는 $\sqrt{\left(\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}\right)^2 + \left(\frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}\right)^2}$ 이다.

2) Vortex 영역

Vortex의 영역을 찾는 가장 간단한 알고리즘은 vorticity의 크기가 큰 영역을 vortex로 정의하는 것이다. Vorticity는 velocity의 curl로 정의가 된다. 즉, $\nabla \times v$ 이다. 또, 이는 지역 유동의 회전을 표현하기도 한다. 물론 vortex 영역의 vorticity 크기는 항상 크다. 하지만 역은 성립하지 않는다. 즉, vorticity가 크다고 항상

vortex인 것은 아니다. 이 방법은 계산량이 적은 장점이 있지만 상대적으로 부정확하다. Vortex 영역을 찾는 또 다른 알고리즘으로는 vorticity 대신 helicity를 사용하는 것이다. Helicity란 vorticity vector를 velocity vector에 투영한 것이다. 즉, $(\nabla \times v) \cdot v$ 이다. 이 방법을 사용하면 velocity에 수직인 vorticity component는 모두 제거된다. 또 다른 Vortex 영역을 찾는 알고리즘으로 Jeong과 Hussain이 제안한 방법이 있다[4]. 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 2-2] Q-Criteria 계산 수식

이 방법은 Q-Criteria라는 이름으로 알려져 있다. 본 기술문서에서 다루는 로터 동역학 데이터들은 모두 Q-Criteria 방법을 통해 vortex 영역을 구분했다.

3. VTK 구현

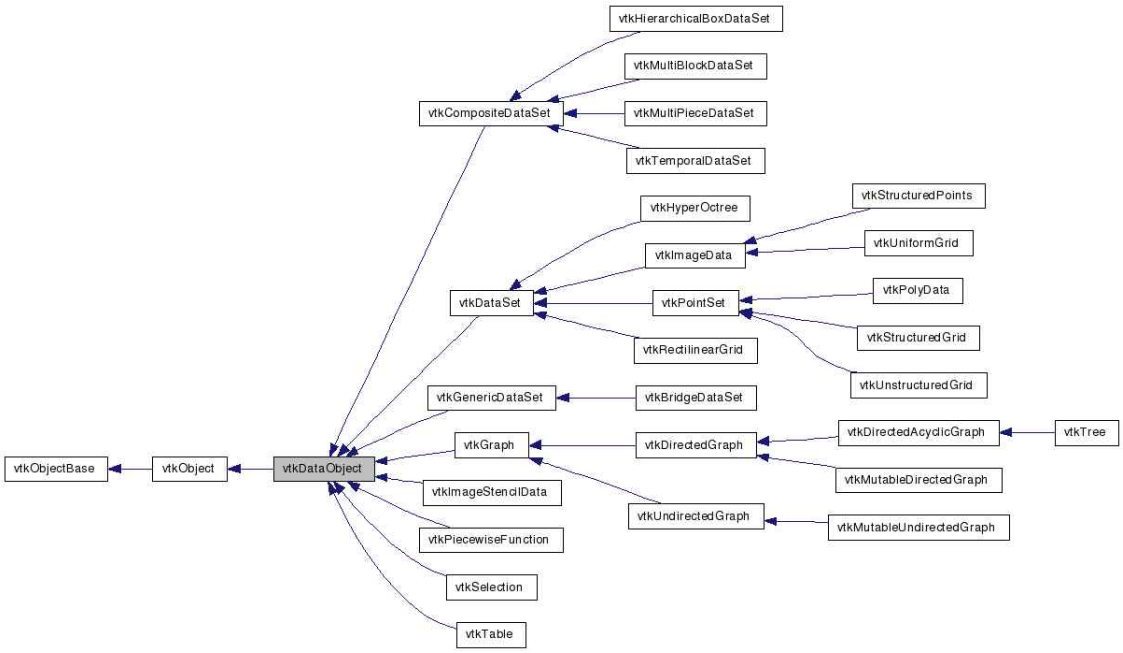
VTK[3]는 과학 데이터 가시화에 많이 사용되는 공개 소프트웨어 라이브러리로, 매우 다양한 자료구조와 가시화 관련 알고리즘 기능을 지원하며, 실제적으로 CFD, 의료, 화학, 구조역학 등 다양한 분야에서 유용하게 사용되고 있다. 이 라이브러리는 기본적으로 C++ 클래스 라이브러리로 구성되었지만 Tcl/Tk, Java, Python과 같은 다른 언어와의 인터페이스를 포함하고 있다. VTK에서 제공하는 가시화 알고리즘은 스칼라, 벡터, 텐서, 볼륨 등 다양한 종류의 데이터와 구조화된 격자, 비구조화된 격자, 정규격자 등 다양한 종류의 격자 구조를 지원한다. 이번 장에서는 VTK를 이용한 기본적인 로터 데이터 가시화에 대해 알아본다.

가. 데이터 구조

2장에서 설명했듯이 본 기술문서에서 다루는 로터 동역학 데이터는 다중 블록의 구조화된 격자구조이고 일부는 여기에 시변환 데이터이기도 하다. 이러한 데이터는 VTK의 혼합(composited) 데이터 구조를 이용하면 처리가 가능하다. 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 요소 별로 저장되는 값으로 벡터, 스칼라, 좌표 등의 값들이 저장된다. VTK의 데이터 구조는 모두 vtkDataObject로부터 파생된다. 이중, 기본 데이터 구조는 vtkDataSet으로부터, 혼합 데이터 구조는 vtkCompositeDataSet으로부터 파생된다.

혼합 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. 이러한 혼합 데이터 구조는 내부에 또 다른 혼합 데이터 구조를 포함할 수도 있고 vtkDataSet에서 파생되는 기본 데이터 구조를 포함할 수도 있다. VTK의 혼합 데이터 구조 중 중요한 두 가지 구조는 vtkTemporalDataSet과 vtkMultiBlockDataSet이다. vtkTemporalDataSet은 여러 타입 스텝으로 구성된 데이터 세트를 위한 데이터 구조이다. 이 데이터 세트는 각 타입 스텝들을 계층

트리의 서로 다른 노드에 저장한다. 또, 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.



[그림 3-1] vtkDataObject의 클래스 계층구조

위에서 설명했듯이 로터 동역학 데이터는 각 블록이 구조화된 격자로 된 다중 블록 시변환 데이터이므로 각 블록은 vtkStructuredGrid이고 각 타임 스텝은 vtkMultiBlockDataSet인 vtkTemporalDataSet을 이용하면 된다.

나. 데이터 가시화

로터 동역학 데이터는 그 특성상 로터 블레이드가 회전하면서 생기는 wake가 다른 블레이드들에 어떤 영향을 미치는지 파악해서 최대한 영향을 미치지 않도록 설계하는 것이 매우 중요하다. 또, 블레이드가 회전하면서 생기는 각종 힘에 견딜 수 있는지를 판단하는 것도 중요하다. 이를 판단하기 위해 로터 동역학 데이터에서 반드시 표출해야하는 데이터는 다음과 같다.

1. 블레이드 데이터에서의 압력 (Pressure) 분포
2. 유동장 데이터에서의 vorticity iso-surface
3. 유동장 데이터에서의 vortex 영역
4. 유동장 데이터에서의 스트림라인

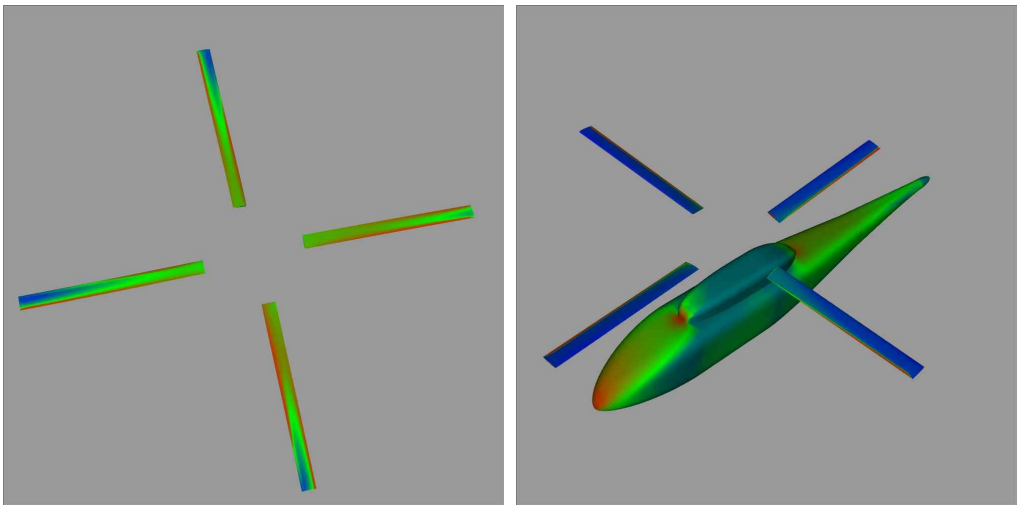
VTK는 이러한 데이터 표출을 원활하게 수행할 수 있도록 해준다. 이번 장에서는 VTK를 이용한 로터 동역학 데이터의 가시화 방법에 대해 설명한다.

1) 블레이드 표면에의 압력 분포

로터 블레이드 데이터를 가시화할 때 가장 중요하게 보아야 할 것은 블레이드 표면에서의 압력 분포이다. 로터 블레이드 데이터는 다중 블록의 구조화된 격자로 구성되어 있는데, 이러한 VTK 데이터는 그 자체로는 OpenGL 등에서 그래픽적으로 표현될 수 없다. 이와 같이 VTK 데이터를 그래픽 데이터로 표현하기 위해서는 이를 다각형 데이터로 변환해야 하는데, 이러한 역할을 해주는 필터가 vtkGeometryFilter이다. 이 필터를 이용해서 vtkPolyData를 생성하면 vtkPolyDataMapper를 이용해서 쉽게 그래픽 표현으로 바꾸는 것이 가능하다. 이때, 만약 표면을 추출하고 싶다면 vtkActor의 vtkProperty에서 SetRepresentationToSurface() 함수로 표면이 그려지도록 해야 하고 메쉬 구조를 그대로 추출하고 싶다면 SetRepresentationToWireframe()을 사용해야 한다. 또, 만약 블레이드 데이터에 단 하나의 스칼라 변수만 저장되어 있다면 문제가 없지만 여러 종류의 스칼라 변수가 저장되어 있는 경우에는 어떤 스칼라를 이용해서 디스플레이할 것인지를 결정해야 한다. 이는 vtkDataSet의 속성 데이터를 받은 뒤 SetActiveScalar() 함수를 호출하면 된다. 2장에서 소개한 로터 블레이드 데이터는 Blanking과 Pressure의 두 스칼라 변수를 포함하고 있으므로 여기에서는 SetActvieScalar("Presure")를 호출해야 할 것이다. 마지막으로 각 스칼라 값들을 어떤 색깔로 색칠할 것인지를 결정해야 하는데 이는 vtkColorTransferFunction 으로 가능하다. vtkColorTransferFunction 데이터를 생성한 뒤 이를 vtkMapper에 등록하면 된다.

한편, 로터 블레이드 데이터는 다중 블록(multi-block) 구조를 가지는데, VTK의 대부분의 필터는 여러 개의 블록을 동시에 처리하지 못하고 단일 블록 단위로만 처리하는 것이 가능하기 때문에 다중 블록 데이터는 여러 번에 걸쳐서 처리해야

한다. vtkGeometryFilter 역시도 단일 블록의 데이터만 처리할 수 있기 때문에 데이터가 다중 블록인 경우 블록의 개수만큼 여러 번에 걸쳐서 필터를 실행해야 하는데, 이럴 경우 여러 개의 vtkPolyData가 생성되게 된다. 이렇게 생성된 여러 개의 vtkPolyData는 하나의 vtkPolyData로 합쳐져야 하는데 이는 vtkAppendPolyData를 이용하면 가능하다.



[그림 3-2] 로터 블레이드 및 동체 데이터에 대한 압력 분포 가시화

2) Vorticity 및 Q-Criteria Iso-surface 생성

2장에서 설명한 로터 동역학 데이터 들 중 1, 2번의 경우에는 Vorticity나 Q-Criteria와 관련된 데이터가 존재하지 않았고 후처리를 통해 생성했다고 언급했다. Vorticity와 Q-Criteria를 계산하기 위한 수식을 보면 알 수 있지만 이러한 후처리 과정에는 로더 데이터의 각 격자에서의 편미분이 필요하다. 컴퓨터 수치해석을 이용한 편미분의 방법에는 여러 가지가 있지만 본 기술문서에서는 VTK를 이용해서 가시화를 하고 있는 만큼 편미분 역시 VTK를 이용하도록 한다. VTK에서는 셀 정보를 이용해서 편미분을 수행하도록 하는 함수를 제공한다. 이는 vtkCell 클래스 내부의 Derivative() 함수인데, 로터 동역학 데이터들을 구성하는 구조화된 격자(vtkStructuredGrid)가 포인트 기반 데이터 구조로써 모든 데이터가 vtkPoint를 기반으로 생성되었고 vtkCell 정보는 많지 않다는 것이 문제이다. 이를 해결하기 위해 먼저 격자구조 내부의 vtkPoint들을 이용해서 각 포인트가 할당되는 vtkCell 셀 정보를 생성해야 한다. 일반적으로 대부분의 격자구조에서 각 포인트들은 한 포인트가 여러 개의 셀에 할당되는데, vtkDataSet에 정의된 GetPointCells() 함수를 이용하면 각 포인트가 할당되는 셀 ID의 배열을 얻을 수 있

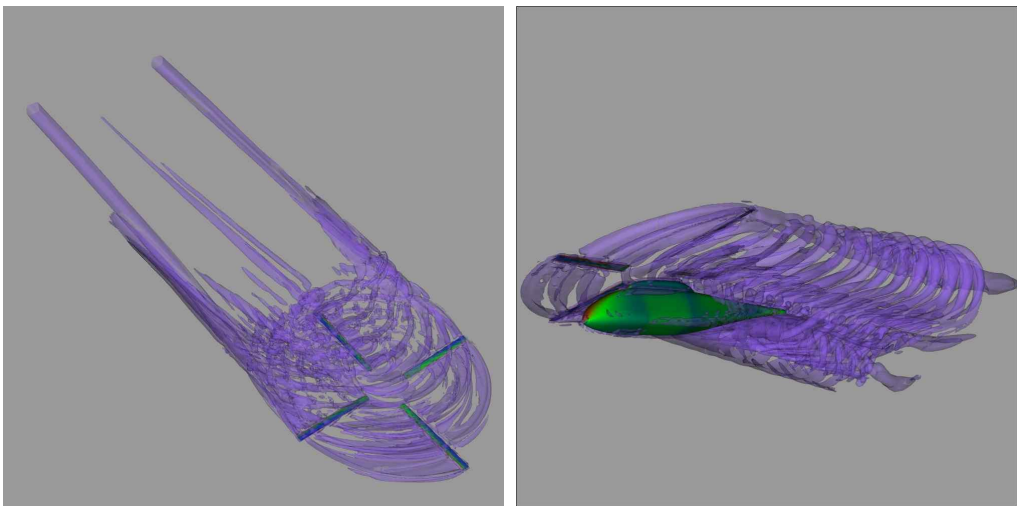
다. 여기서, 우리가 구하고자 하는 것은 각 포인트 위치에서의 편미분 값인데, 이는 그 포인트가 할당된 어떤 셀에서 계산하더라도 관계가 없으므로 GetPointCells() 함수로부터 구한 셀 ID 배열 중 가장 처음으로 저장된 셀을 사용하도록 한다. 이렇게 구한 셀 ID를 이용하면 실제 vtkCell을 구할 수 있는데, 이는 역시 vtkDataSet에 정의된 GetCell() 함수를 이용하면 가능하다. 이제 vtkCell의 Derivative() 함수를 호출하기 위해 그 셀에서의 해당 포인트의 매개변수 좌표(parametric coord.)를 구해야 하는데 이는 vtkCell의 GetParametricCoords() 함수를 이용하면 가능하다. 이렇게 구한 매개변수 좌표와 해당 포인트에서의 벡터 등을 이용해서 Derivative() 함수를 호출하면 편미분 행렬을 얻을 수 있고, 이를 이용하면 vorticity 및 Q-Criteria 값을 쉽게 계산할 수 있다.

이제 이렇게 생성한 vorticity와 Q-Criteria 값을 이용, Iso-surface를 생성하는 방법에 대해 살펴보자. Iso-surface란 특정 iso-value 값을 가지는 영역을 합쳐서 만든 표면(surface)를 말한다. 일반적으로 2차원의 공간에서는 같은 값을 가지는 영역을 모으면 선(line)이 되고 3차원의 공간에서는 같은 값을 가지는 영역을 모으면 면(surface)이 된다. VTK에서는 Iso-surface를 생성하기 위한 필터를 여러 개 제공하고 있다. 여러 종류의 Iso-surface 필터들 중, 본 기술문서에서는 vtkContourFilter를 사용한다. vtkContourFilter는 다양한 격자 구조 및 차원의 데이터를 지원할 뿐만 아니라 스칼라 트리(scalar tree)를 이용한 빠른 Iso-surface의 생성도 가능하다. 이 필터 역시 최종적으로 vtkPolyData 형태의 다각형 데이터를 생성해 내므로 vtkPolyDataMapper를 통해 VTK에서 그래픽화가 가능하다. 이때, 많은 경우 iso-surface의 색깔 및 투명도를 조절하는데, 이는 vtkColorTransferFunction을 생성한 뒤 vtkMapper에 등록하거나 vtkActor에서 vtkProperty를 받은 후 SetColor() 및 SetOpacity() 함수를 이용해서 변경하는 것이 가능하다.

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

```
 // for all points in the dataset
 for (k = 0; k < size; ++k)
 {
   vtkCell *cell = NULL;
   double p[3], pcoords[3];
   int pindex, subid = 0;

   vtkIdList *cellids = vtkIdList::New();

   // get the global coord. and cell ids for each point
   ds->GetPoint(k, p);
   ds->GetPointCells(k, cellids);

   // get the first cell attached in the point
   cell = ds->GetCell(cellids->GetId(0));

   // get the parametric coord. of the point and cell
   double *pc = cell->GetParametricCoords();
   pcoords[0] = pc[pindex*3];
   pcoords[1] = pc[pindex*3+1];
   pcoords[2] = pc[pindex*3+2];

   // get velocity vectors of each point in the cell
   velVector->GetTuples(cell->PointIds, cellVectors);

   // calculate derivative at the point
   cell->Derivatives(subid, pcoords, cellVectors->GetPointer(0), 3,
derivs);

   // calculate vorticity
   vorticity[k*3  ] = derivs[7] - derivs[5];
   vorticity[k*3+1] = derivs[2] - derivs[6];
   vorticity[k*3+2] = derivs[3] - derivs[1];

   // calculate q-criteria
   q[k] = -0.5 * (derivs[1]*derivs[3] + derivs[2]*derivs[6] +
                  derivs[5]*derivs[7]);

   cellids->Delete();
}
```

[소스 3-1] vorticity와 Q-Criteria 값 계산 코드



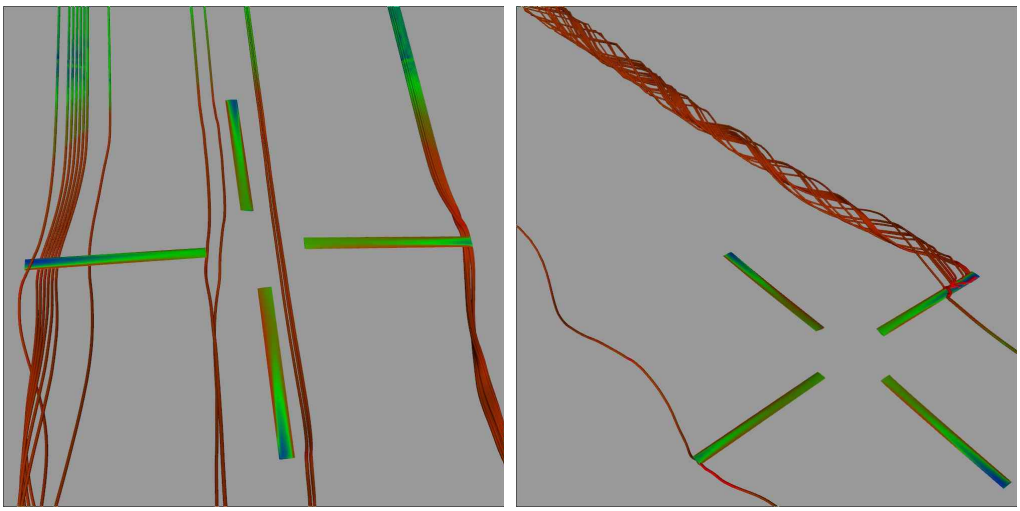
[그림 3-3] 블레이드 및 동체 데이터에 대한 Q-Criteria Iso-Surface 가시화

3) 스트림라인 생성

스트림라인이란 벡터 데이터 중 유동 데이터(flow data)를 가시화하는데 매우 중요하게 사용되는 표현 기법으로 운동하는 유체의 각 점에서 속도벡터의 방향이 접선이 되도록 그은 곡선을 의미한다. 로더 동역학 데이터 역시 스트림라인 가시화가 매우 중요한데, 이를 이용해서 vortex의 회전 정도를 한눈에 파악할 수 있기 때문이다. VTK에서는 여러 종류의 스트림라인을 제공하는데, 본 기술문서에서는 스트림라인과 스트림튜브(stream tube)를 사용한다. VTK에서 스트림라인을 생성하는 방법은 약간 복잡한데 이는 스트림라인을 생성하기 위해 선택해야 하는 옵션이 많기 때문이다. 먼저, VTK에서는 vtkStreamTracer, vtkStreamer, vtkStreamLine 등 여러 종류의 스트림라인 생성을 위한 클래스들이 있는데, 본 기술문서에서는 vtkStreamTracer를 사용한다. vtkStreamer이나 vtkStreamLine 등의 클래스는 모두 과거에 사용했던 클래스들로 최근에는 더 이상 업데이트 되지 않고 있어 시변환 데이터 지원이나 병렬처리 등이 전혀 구현되지 않고 있다. 물론, VTK 제작자들도 vtkStreamTracer를 사용할 것을 권장하고 있다. vtkStreamTracer를 사용하기 위해서는 먼저 선적분(integration) 방법을 결정해야 하는데, VTK에서는 오일러 기법은 물론 Runge-Kutta 기법도 제공한다. 본 기술문서에서는 여러 종류의 선적분 방법들 중 2차 Runge-Kutta 방법을 사용하는데, 이 방법은 4차나 5차 Runge-Kutta 방법에 비해 정밀도가 떨어지지만 훨씬 빠른 속도를 제공하기 때문이다. 실제로 2차와 4차 방법을 비교했을 때 두 방법의 오차는 극히 작았던 반면 속도는 1.5배 이상 2차가 빨랐다. 향후 병렬 처리 등을 통해 이러한 속도 문제가 해결된다면 4차 Runge-Kutta 방법을 사용할 계획이

다. VTK에서는 vtkRungeKutta2 클래스로 2차 Runge-Kutta 방법을 제공한다. 적분 방향은 전진(forward)와 후진(backword)를 모두 하도록 했고 (SetIntegrationDirectionToBoth() 함수), 스텝 수는 임의로 10000 스텝을 하도록 했다. 본래 스트림라인 생성 도중 필드 데이터 밖으로 나갈 경우 멈추도록 하고 싶었으나 VTK에서 이러한 모드를 제공하지 않아 임의로 10000 스텝을 처리하도록 했다. 물론, VTK는 스트림라인이 필드 밖으로 나가거나 특정 스텝 동안 계속 같은 셀 내에서만 있으면 스트림라인 생성을 멈추도록 하는 기능이 있으나 반드시 스텝 수를 지정하도록 되어 있는 문제가 있다. 따라서 충분히 큰 숫자로 생각되는 10000 스텝을 지정했으나 만약 데이터가 매우 큰 경우에는 스트림라인이 끊겨져 보이는 현상이 발생할 수 있다. 이러한 문제는 향후 반드시 개선해야 하는 문제이다. 물론, 실험 데이터에서는 이러한 문제가 발생하지 않았다. 또한 vtkStreamTracer 역시 단일 블록의 처리만이 가능하므로 블록의 개수만큼 여러 번 실행한 뒤 vtkAppendPolyData로 합치는 과정이 필요하다.

이와 같은 방법으로 스트림라인을 생성할 수 있는데, 여기서 전체 데이터의 어느 위치에서 스트림라인을 생성할지를 정하는 것은 매우 중요한 문제이다. 앞에서 설명했듯이 사용자가 로터 동역학 데이터에서 가장 중요하게 보는 부분은 Vortex 영역이다. 따라서 이 영역에 스트림라인을 생성해서 가시화한다면 실제로 유동이 어느 정도나 회전하는 지를 파악할 수 있다. 이를 위해 본 기술문서에서는 Vorticity나 Q-Criteria 값이 높은 부분에 스트림라인을 생성하도록 했다. 여기서 vorticity나 Q-Criteria 값이 높은 부분의 정의는 데이터마다 다를 수 밖에 없지만 평균적으로 상위 80% 이상이면 적당하다. 각 데이터 별로 이러한 영역에 seed point를 정의한 뒤 스트림라인을 생성한 그림은 그림 3-4와 같다.



[그림 3-4] 듬성한 격자 및 조밀한 격자 로터 데이터에서의 스트림라인 생성

4) 키메라 격자 구조

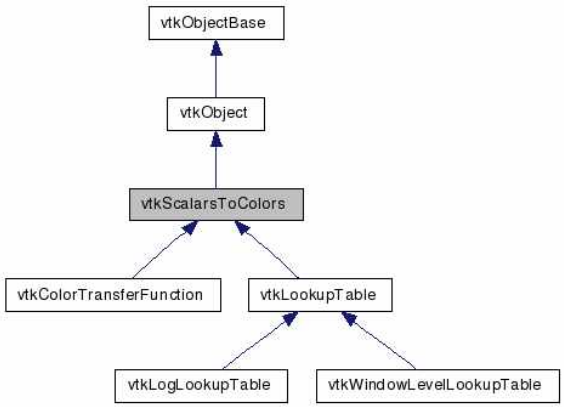
매우 복잡한 유동 문제를 풀 때 전체 필드를 단 하나의 연속된 격자계(구조화된 격자계 또는 비구조화된 격자계 등)로 표현하는 것은 매우 어렵다. 이러한 문제를 해결하기 위해 일반적으로 많이 사용하는 방법은 서로 다른 구조적 성질을 가지는 여러 종류의 격자계를 겹쳐서 구성하는 것이다. 이렇게 구성하는 방법을 키메라 격자구조(Chimera grid) 방법 또는 오버세트 격자구조(overset grid) 방법이라고 한다. 여러 격자가 겹쳐져 있으므로 각 셀 마다 겹치는 부분이 있게 되는데, 이러한 부분은 각 셀 또는 포인트의 가시성(visibility)을 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 달라지게 된다. 로터 데이터에서는 필드 데이터의 각 격자에 가시성 정보를 Blanking 필드에 저장하여 어떤 포인트가 구멍 포인트인지를 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.

5) Transfer Function 생성

본 기술문서에서 렌더링한 결과는 모두 vtkPolyDataSet으로 반환된다. 그런데 vtkPolyDataSet 내부에는 다각형 데이터의 기하학적인 정보만 저장될 뿐 색깔 등의 정보는 저장되지 않는다. 따라서 사용자가 따로 색깔 정보를 생성해야 하는데, 이때 데이터 내부의 스칼라 값에 따라 색깔을 변화시키고자하는 경우에는 transfer function을 함께 생성해야 한다. VTK에서는 vtkColorTransferFunction과 vtkLookupTable의 두 종류의 transfer function을 제공한다. 이들 클래스는 모두

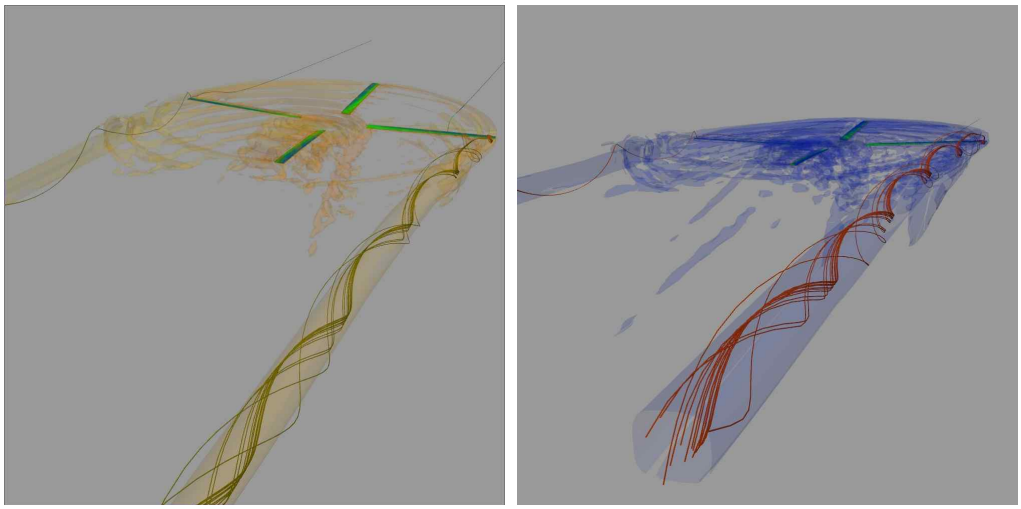
vtkScalarsToColors 클래스를 상속받는데, 이 클래스는 스칼라 값을 색깔로 변경하기 위한 일반적인 목적의 클래스이다. vtkColorTransferFunction은 piecewise hermite 함수를 이용해서 스칼라 값을 RGB 또는 HSV 값으로 매핑하는 클래스이다. vtkColorTransferFunction에 AddRGBPoint() 또는 AddHSVPoint() 함수로 중간 포인트들을 입력하면 vtkColorTransferFunction은 입력된 포인트들을 이용해서 piecewise hermite 함수를 생성한다. 이 때 각 포인트들은 linear또는 log scale의 함수로 연결된다. 반면에 vtkLookupTable은 table을 이용해서 스칼라 값을 RGBA 값으로 또는 거꾸로 RGBA 값을 스칼라 값으로 변경시키는 클래스이다. vtkLookupTable은 반드시 색깔 table이 존재해야 하는데, 이 table은 직접적으로 색깔 값을 입력하는 것으로도 생성이 가능하고 또는 SetRange() 등의 함수로 스칼라 영역을 입력한 뒤 SetRampToLinear() 등의 함수로 RGBA table을 생성하는 것도 가능하다.



[그림 3-5] vtkScalarsToColors 클래스 계층구조

본 기술문서에서는 VTK에서 지원하는 두 transfer function들 중 vtkColorTransferFunction을 사용한다. 이는 로터 동역학 데이터의 스칼라 변수에 대한 히스토그램(histogram)을 작성해보면 값의 분포가 균일하지 않고 특정 영역에 몰려있는데, vtkColorTransferFunction은 이러한 특정 영역을 강조해서 transfer function을 생성하는 데에 유리하기 때문이다. 본 기술문서에서는 전체 데이터를 분석해서 자동으로 transfer function을 생성하고자 하는데, 이 transfer function은 해당 스칼라의 최소값을 (0, 0, 1)의 파랑색, 중간값을 (0, 1, 0)의 초록색, 최대값을 (1, 0, 0)의 붉은색으로 하는 함수이다. 이때, 이 함수는 각 값들을 key로 다시 그 중간값들((0, 0,5, 0,5)나 (0.5, 0.5, 0) 등)을 생성하고 또 그 중간값을 key로 새로운 중간값들((0, 0.25, 0.75) 등)을 생성하는 등 재귀적(recursive)으

로 함수를 세밀화할 수 있다. 여기서 생성되는 중간값들이 많으면 많아질 수록 transfer function의 정확도는 높아질 것이다. 이때, 최소, 최대값은 명확히 정해져 있으나 중간값들을 무엇으로 할지가 문제이다. VTK 데이터 구조에서는 전체 데이터에서 최소, 최대값을 손쉽게 구할 수 있는 함수를 제공하므로 이는 쉽게 구하는 것이 가능하다. 이때, 데이터가 전체적으로 균일하게 분포되어 있다면 중간값들은 단순히 *(최소값 + 최대값) / 2*의 방식으로 계산이 될 것이다. 그러나 전체적으로 균일하지 못하고 한쪽으로 치우쳐져 있다면 전혀 다른 값이 될 가능성이 없지 않다. 이러한 이유로 여기에서는 전체 스칼라 값을 정렬한 뒤 실제 중간에 위치한 값이 무엇인지를 찾도록 했다. 단순히 (최소 + 최대) / 2로 한 경우와 정렬을 통해 중간값을 찾는 경우의 가시화 결과는 그림 3-6과 같다. 이 그림은 조밀한 격자 로터 데이터에서 Q-Criteria 값이 0.0001인 영역에 iso-surface를 생성하고 vorticity가 90% 이상인 영역에서 스트림라인을 생성한 것이다. 적용 transfer function 이외에는 모든 조건이 동일한데, 좌측은 단순히 (최소+최대)/2의 방식으로 transfer function을 생성한 결과이고, 우측은 스칼라 정렬을 한 뒤 transfer function을 생성한 결과이다.



[그림 3-6] Transfer function 적용 결과 비교

2장에서 소개한 로터 동역학 데이터들에는 시변환 데이터도 있는데 만약 transfer function이 각 타임 스텝마다 변경이 된다면 애니메이션 시 동일한 스칼라 값을 가지는 부분의 색깔이 계속적으로 변화가 되는 문제가 발생한다. 따라서 transfer function을 생성할 때 전체 타임 스텝에 대한 단일 transfer function을 생성해야만 한다. 이와 같이 전체 데이터에 대한 transfer function을 생성할 경우 전

체 데이터를 정렬해야 하는데 이것은 매우 많은 시간을 필요로 한다. 이를 해결할 수 있는 방법에는 다음과 같은 것들이 있다.

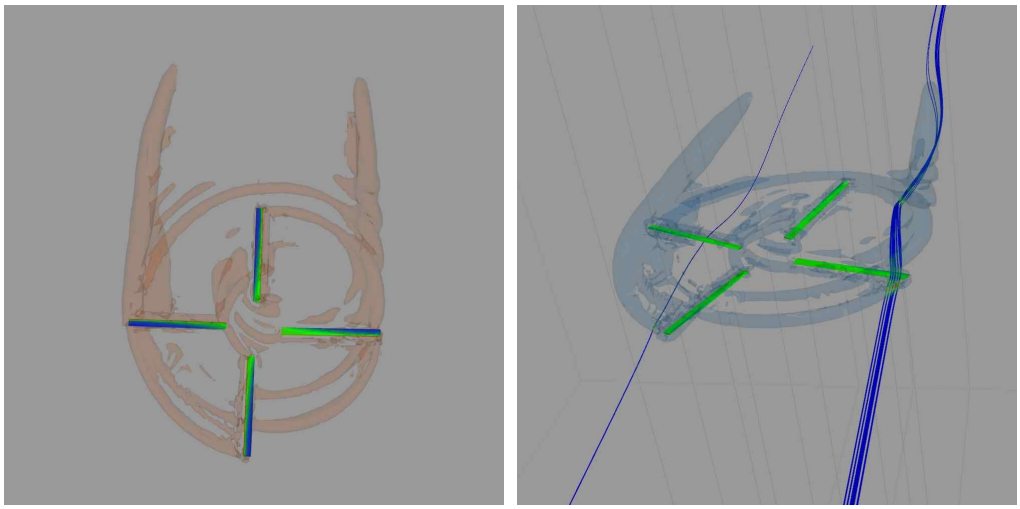
1. 정렬의 병렬화
2. 히스토그램을 이용한 영리한 transfer function 생성
3. 전처리

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

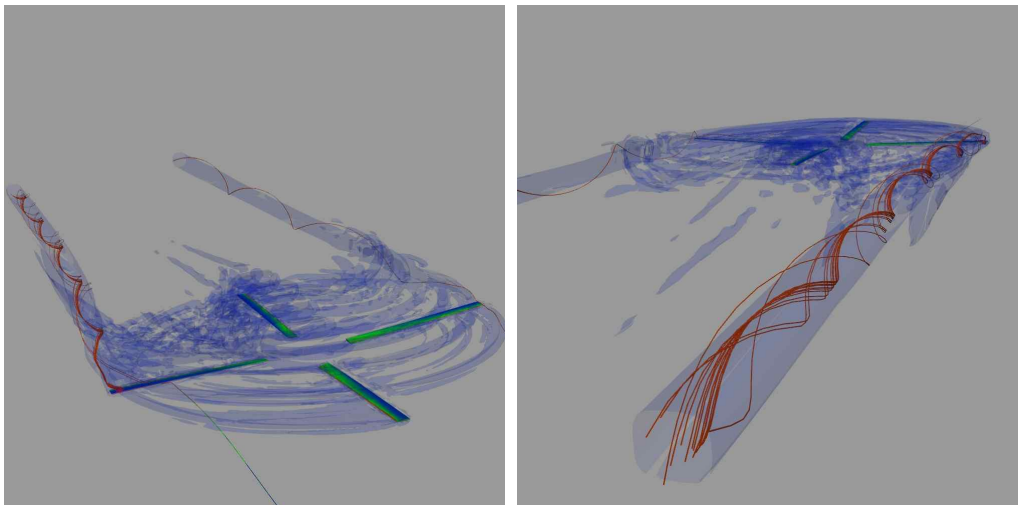
4. 구현 결과

VTK 이용한 로터 시뮬레이션의 가시화를 Intel Xeon X5450 가시화 컴퓨터를 이용해서 구현했다. 이 컴퓨터는 총 64GB의 메인 메모리를 가지고 있어 로터 시뮬레이션 데이터 전체와 VTK에서 생성하는 각종 데이터를 모두 저장하고 가시화할 수 있었다. 전체 데이터를 각 타입 스텝별로 가시화하면서 vorticity 및 pressure 분포의 변화와 블레이드 뒷부분의 wake 변화를 실시간으로 파악할 수 있었다. 또, 3장에서 설명한 가시화 기법들은 범용 데이터를 위한 가시화 기법으로, 일반 사용자도 쉽게 사용할 수 있도록 로터 시뮬레이션 데이터의 가시화를 위한 라이브러리를 새로 구축했다.

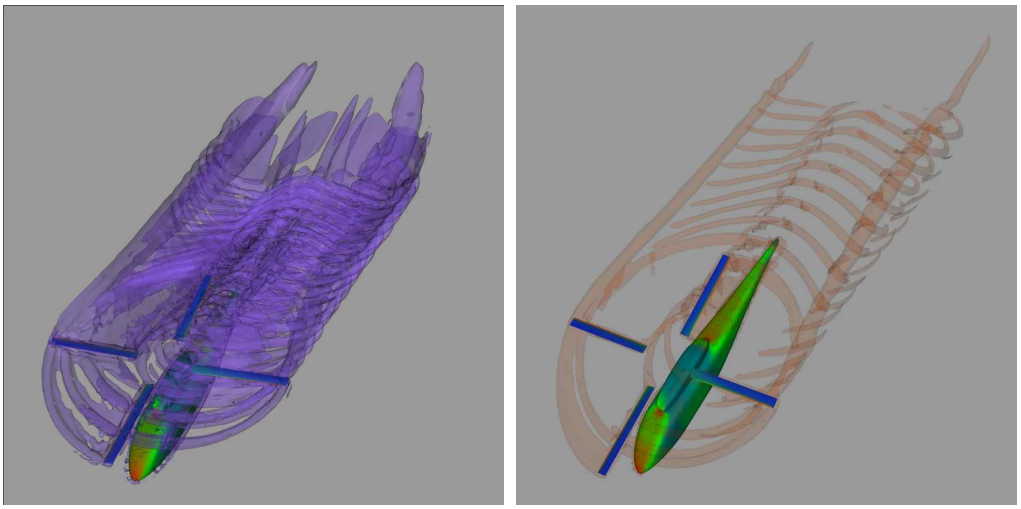
그림 4-1, 4-2, 4-3은 차례대로 듬성한 격자 로터 데이터, 조밀한 격자 로터 데이터 그리고 동체 데이터에 대한 가시화 결과이다. 세 데이터 모두 블레이드를 비롯한 벽면의 표면에는 압력 분포를 디스플레이했다. 듬성한 격자 로터 데이터와 조밀한 격자 로터 데이터의 필드 데이터에는 속도 벡터가 포함되어 있으므로 스트림라인과 Q-Criteria를 이용한 Iso-surface를 동시에 표현할 수 있었다. 이를 통해 Vortex 영역의 스트림라인이 급격하게 회전하는 것을 알 수 있었다. 동체 데이터의 경우 필드 데이터에 속도 벡터가 포함되어 있지 않아 스트림라인은 생성할 수 없었지만 동체 주변 보다는 로터 블레이드 주변의 유동장에 vortex가 많이 생성함을 확인할 수 있었다. 또한, 같은 iso-surface나 스트림라인에 서로 다른 transfer function을 적용함으로써 여러 가지 색깔을 입힐 수 있음을 확인했다.



[그림 4-1] 듬성한 격자 로터 데이터 가시화 결과



[그림 4-2] 조밀한 격자 로터 데이터 가시화 결과



[그림 4-3] 동체 데이터 가시화 결과 (Q-Criteria와 vorticity Iso-surface)

5. 결론

본 기술문서에서는 대용량 로터 시뮬레이션 데이터를 고성능 가시화 컴퓨터를 이용해서 실시간으로 가시화하는 방법에 대해 설명했다. 데이터를 이용해서 보여줄 수 있는 여러 정보들 중 사용자가 꼭 필요로 하는 정보들을 표출하는 방법과 이를 VTK를 이용해서 구현하는 방법에 대해 살펴보았다. 특히, 자동으로 높은 vorticity를 갖는 영역을 찾아 이 영역에서 스트림라인을 가시화하여 wake를 표출할 수 있도록 했다. 또, Q-Criteria 방법을 이용, vortex의 영역을 가시화하도록 했다. 비록 본 기술문서에서 대용량 데이터에 대한 가시화는 구현할 수 있었지만 스트림라인 생성과 같은 일부 기능에서는 데이터를 표출하는데 30초 이상의 많은 시간을 필요로 했다. 이것은 여러 대의 가시화 컴퓨터를 이용한 병렬 가시화나 GPU를 이용한 수치 계산 가속 기법을 통해 해결이 가능하다. 또, 본 기술문서에서는 약 600만개의 격자를 가지는 시변환 데이터를 다루었지만 향후 10배 이상의 거대 용량 데이터를 가시화할 계획이다.

6. 참고문헌

[1] H. Hauser, R. S. Laramee, H. Doleisch, "State-of-the-art report 2002 in flow visualization", Technical Report TR-VRVis-2002-003, 2002

[2] R. S. Laramee et al., "The state of the art in flow visualization: Dense and texture-based techniques. Computer Graphics Forum 23 (2004), 2004.

[3] VTK Homepage, http://www.vtk.org

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