가상현실 가시화 시스템과
햅틱 디바이스의 실시간 연계

2014. 11

KiSTi www.kisti.re.kr 한국과학기술정보연구원

목 차

표 차례

그림 차례

1. 서론

일반적으로 가상현실 기반 visualization application은 데이터를 시각적으로 보여주는 데에 초점을 맞추고 있다. 물론 visualization 과정에는 입체영상 출력, lighting 계산 등 데이터를 시각적으로 이해하는 데에 도움을 주는 다양한 그래픽스 알고리즘이 포함되어 있고, 경우에 따라서는 global illumination, shadow 등 더 사실적인 이미지를 만드는 기법이 적용되기도 한다. 하지만 촉감과 같이 일반적인 visualization 과정에서는 잘 사용하지 않는 감각을 정보 전달 경로로 사용함으로써 사용자에게 데이터의 특성을 더 정확히 알려주는 기법에 대한 연구는 상대적으로 등한시되었다. 이 연구에서는 그 문제를 해결하기 위한 첫 번째 단계로 가상현실 기반 가시화 시스템과 햅틱 렌더링 서버의 실시간 연계 방안을 다룬다.

[그림 1-1] GIVI와 햅틱 서버를 동시에 사용하는 모습

이 보고서는 다음과 같이 구성되어 있다. 먼저 2장과 3장에서는 GIVI와 햅틱 서버의 원활한 통신을 구현하기 위한 통신채널과 프로토콜에 대해 설명한다. 그리고 4장은 햅틱 서버의 설계와 구현내용을 다룬다. 마지막 5장에서는 GIVI에 구현되어 있는 햅틱 입/출력 및 관련 정보 처리 방법에 대해 설명한다.

2. Communication channel

GIVI 마스터와 햅틱 서버 사이의 메시지 전송을 위한 communication channel은 두 개로 구분한다. 하나는 GIVI 마스터가 햅틱 서버로 명령을 전달하고 그 명령의 실행결과를 받기 위한 것이고, 다른 하나는 햅틱 서버에 연결되어 있는 햅틱 펜의 위치와 방향정보, 햅틱 렌더링 대상 mesh의 변경을 GIVI 마스터에게 알려주기 위해 운용한다. 이 주 첫 번째 채널은 command channel이라 부르고, 두 번째 채널은 information channel이라고 부른다.([표 2-1])

명 칭	포트 번호	TCP/UDP	메시지 종류	
			통신방향	
Command channel	19662	TCP	Command	Response
			GIVI → 햅틱 서버	햅틱 서버 → GIVI
Information channel	19663	TCP	Notification	Acknowledgement
			햅틱 서버 → GIVI	GIVI → 햅틱 서버

[표 2-1] GIVI - 햅틱 서버 간 통신채널 구성

나중에 설명하겠지만 햅틱 펜 정보(haptic pen info)와 같이 수시로 값이 변하는 정보는 UDP를 이용해서 보낼 수도 있겠으나 메시지의 크기가 충분히 작고, LAN 환경에서 실행되는 것을 전제하기 때문에 일단 모든 통신에 TCP를 이용하기로 했다. 그리고 GIVI는 여러 노드로 구성되어 있는 클러스터 환경에서 실행될 수도 있는데, 이 경우 마스터 역할을 담당하는 노드가 햅틱 서버와 통신을 하며, GIVI 마스터가 통신 내용을 GIVI 슬레이브에게 전파하는 역할을 담당한다.

3. GIVI ↔ Haptic 프로토콜 정의

이 장에서는 GIVI와 햅틱 서버 사이의 메시지 교환을 위한 네트워크 프로토콜을 정의한다. GIVI 마스터와 햅틱 서버 사이가 교환하는 메시지는 크게 command, response, notification, acknowledgement로 구분한다.

구분	용도
Command	GIVI가 햅틱 서버의 동작을 제어
Response	Command 실행결과를 GIVI 마스터에게 알려주기 위해 사용
Notification	햅틱 서버가 GIVI 마스터에게 현재 상태를 알려주기 위해 사용
Acknowledgement	햅틱 서버의 notification에 대한 ACK

[표 3-1] GIVI 마스터 - 햅틱 서버 간 메시지 종류

기본적으로 command는 GIVI에서 햅틱 서버로 명령을 전송할 때 사용하고, response는 햅틱 서버가 명령의 처리 결과를 보내줄 때 사용한다. 그리고 'mesh 교체'와 'haptic probe info' 등의 메시지는 notification으로 분류되며, command와는 달리 햅틱 서버가 GIVI에게 보내는 메시지가 여기에 포함된다.

가. Operation code의 정의

메시지의 operation code(이하 opcode)는 1byte unsigned char 형식을 갖고 있으며, 각 command와 response에 대응하는 opcode는 [표 3-2]에 정의되어 있다.

기능 / 명령	Opcode	통신방향
Calibration request	10	GIVI → 햅틱 서버
Calibration response	11	햅틱 서버 → GIVI
Mesh 추가 request	20	GIVI → 햅틱 서버
Mesh 추가 response	21	햅틱 서버 → GIVI
Mesh 교체 notification	30	햅틱 서버 → GIVI
Mesh 교체 acknowledgement	31	GIVI → 햅틱 서버
Mesh 삭제 request	40	GIVI → 햅틱 서버
Mesh 삭제 response	41	햅틱 서버 → GIVI
햅틱 시작 request	50	GIVI → 햅틱 서버
햅틱 시작 response	51	햅틱 서버 → GIVI
Haptic pen info	60	햅틱 서버 → GIVI
Haptic 성능 정보	61	햅틱 서버 → GIVI
햅틱 중지 request	70	GIVI → 햅틱 서버
햅틱 중지 response	71	햅틱 서버 → GIVI
Unknown error notification	255	햅틱 서버 → GIVI

[표 3-2] 햅틱 명령에 따른 opcode

나. Command error code

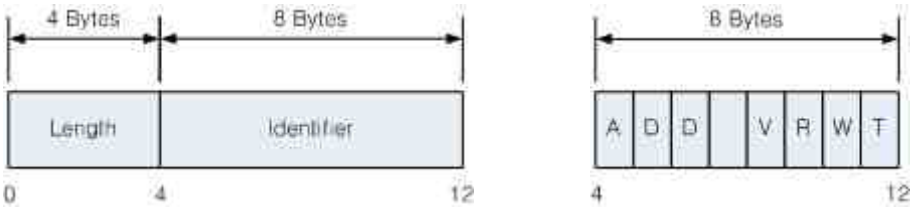
Error code	Message	In response to
0	Operation successful	All commands
1	Invalid command	
12	Failed to initialize haptic device	Calibration request
13	Calibration failed (other unknown errors)	
21	Duplicate mesh ID	Mesh 추가
22	Out of memory	
23	Invalid number of vertices / triangles	
24	Invalid number of data values	
51	No mesh with the specified ID	Mesh 교체
71	No mesh with the specified ID	Mesh 삭제
72	Failed to remove the target mesh	
111	Workspace undefined	햅틱 시작
112	Invalid workspace	
113	Invalid mesh ID	
114	Invalid matrix information	
211	Failed to stop haptic rendering	햅틱 종료

[표 3-3] 햅틱 명령에 대한 에러 코드

여기서 invalid command는 특정 작업이 필요한 시점에 부적절한 명령을 받을 경우, 이를 햅틱 서버가 GIVI에게 알려주기 위해 사용한다.

다. 메시지 헤더

헤더는 GIVI와 햅틱 서버가 주고받는 모든 메시지의 제일 앞부분에 위치해서 어플리케이션을 식별하는 데에 사용된다. 따라서 GIVI와 햅틱 서버는 네트워크로부터 메시지를 받을 때마다 첫 12바이트 중 마지막 8바이트의 내용이 특정 내용을 갖고 있는지를 확인하고, 그 결과에 따라서 메시지 처리 여부를 결정한다. 헤더의 실제 내용은 [그림 3-1]과 같다.



[그림 3-1] 헤더의 구성(왼쪽) 및 실제 내용(오른쪽)

엄밀하게 따지면 'Length' 부분이 없어도 어플리케이션의 작동에는 문제가 없지만 데이터의 무결성을 확보하기 위해 의도적으로 추가했다. 여기서 주의할 점은 'Length'는 전체 메시지에서 첫 4바이트(Length 부분)를 제외한 나머지의 길이를 의미하기 때문에 전체 메시지의 길이는 (Length + 4) 바이트가 된다는 사실이다.

Field name	Type	Bytes	Contents
Length	unsigned int	4	이후 따라오는 메시지의 길이 (bytes)
Identifier	char	1	A (대문자)
	char	1	D (대문자)
	char	1	D (대문자)
	char	1	공백문자(space, 0x20)
	char	1	V (대문자)
	char	1	R (대문자)
	char	1	W (대문자)
	char	1	T (대문자)

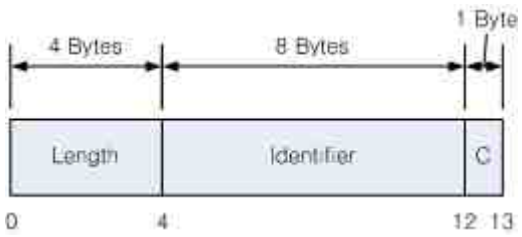
[표 3-4] 메시지 헤더의 실제 내용

라. Calibration

Calibration은 햅틱 장비를 사용하기 전에 원점을 맞추는 작업으로, 햅틱 장비를 처음 사용할 때 한 번만 실행하면 된다(물론 여러 번 실행해도 무방하다). 이 때 사용자가 햅틱 펜을 적절한 위치에 들고 있어야 하고, 경우에 따라서는 햅틱 렌더링을 하는 동안 오차가 발생해서 다시 calibration을 해야 하는 상황도 있기 때문에 사용자가 필요할 때마다 직접 calibration을 수행할 수 있도록 했다.

1) Calibration request (GIVI → Haptic)

Calibration request는 별도의 parameter를 필요로 하지 않기 때문에 request 메시지는 헤더와 command opcode의 13바이트만으로 구성되어 있다([그림 3-2]).



[그림 3-2] Calibration request

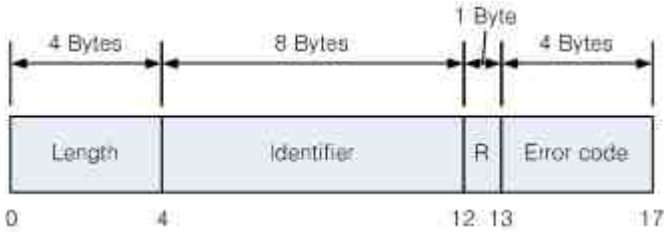
Calibration request의 실제 내용은 [표 3-5]와 같다.

Field name	Type	Bytes	Contents
Header	unsigned int	4	Message length
	char array	8	Identifier
Command	unsigned char	1	Calibration request, 10

[표 3-5] Calibration request sequence의 실제 내용

2) Calibration response (Haptic → GIVI)

Response는 calibration의 실행결과를 GIVI에게 알려주기 위해 사용한다. 정상적으로 calibration이 끝나면 error code에는 0이 저장되며, 그 외의 경우는 [표 3-3]의 에러 코드를 돌려준다.



[그림 3-3] Calibration response sequence

Calibration response의 실제 내용은 [표 3-6]과 같다.

Field name	Type	Bytes	Contents
Header	unsigned int	4	Message length
	char array	8	Identifier
Response code	unsigned char	1	Calibration response : 11
Error code	unsinged int	4	

[표 3-6] Calibration response sequence의 실제 내용

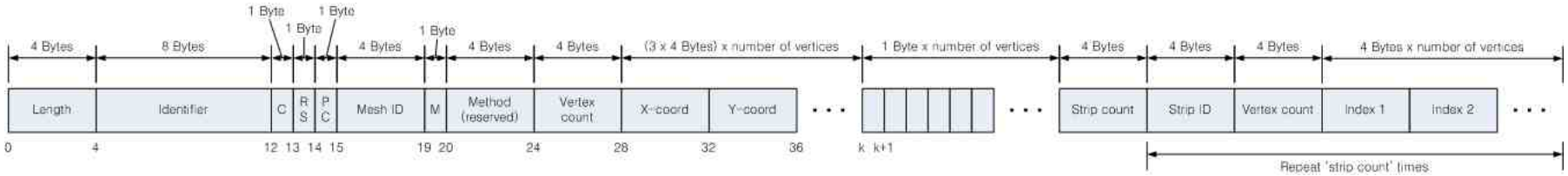
마. Mesh 추가

햅틱 서버에 mesh를 등록하기 위해 사용하는 메시지는 mesh의 종류에 따라서 달라진다. 우선 햅틱 서버가 다뤄야 하는 mesh의 종류를 살펴보자. [표 3-7]을 보면 2D plane과 general surface model은 내부에는 데이터가 존재하지 않으며, 표면에만 스칼라 또는 벡터 데이터가 존재한다. 그렇기 때문에 햅틱 렌더링도 표면에 정의된 데이터를 기반으로 한다. 하지만 streamline과 pathline은 조금 다르게 처리할 필요가 있다. GIVI에서 streamline과 pathline은 비록 이름에는 line이 들어가지만 control point와 방향벡터를 이용해서 구성한 tube로 표현된다. 그리고 햅틱 서버는 line이 아닌, tube 형태의 geometry를 받아서 햅틱 렌더링을 수행한다. 햅틱 렌더링은 tube 내부에 햅틱 포인터가 위치할 때 반력을 주는 방식으로 유동의 흐름을 느낄 수 있도록 해준다. 그렇기 때문에 tube의 경우에는 내부 데이터가 햅틱 렌더링의 소스가 된다.

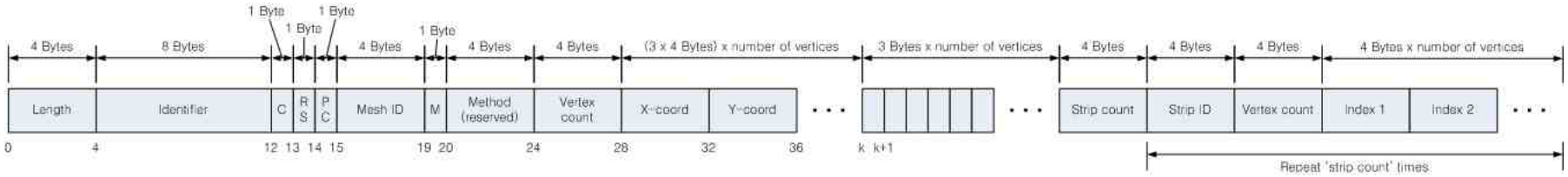
	표면 데이터	내부 데이터 (control point)	표면 햅틱 렌더링	내부 햅틱 렌더링
2D plane (in 3D space)	스칼라	없음	지원	미지원
	벡터		지원	미지원
	없음		미지원	미지원
General surface model	스칼라		지원	미지원
	벡터		지원	미지원
	없음		미지원	미지원
Stream/path line	스칼라	있음	미지원	미지원
	벡터		미지원	지원
	없음		미지원	미지원

[표 3-7] Mesh 형태별 데이터 구성 및 햅틱 렌더링 방법

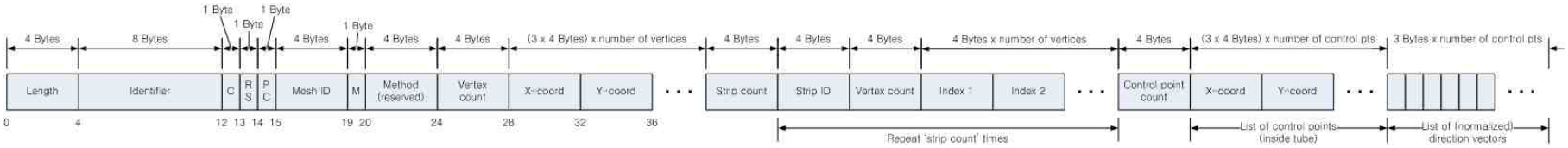
1) Mesh 추가 request (GIVI → Haptic)



[그림 3-4] Scalar 데이터가 정의된 surface mesh 추가 request



[그림 3-5] Vector 데이터가 정의된 surface mesh 추가 request



[그림 3-6] Streamline / pathline 추가 request

		Type	Bytes	Contents
Header		unsigned int	4	Message length
		char array	8	Identifier
OP code		unsigned char	1	Mesh 추가 request : 20
Rigidity		unsigned char	1	Rigid body : 'R', Soft body : 'S'
Workspace type		unsigned char	1	Plane : 'P', Cube : 'C'
Mesh ID		unsigned int	4	Mesh ID는 화면에 보이는 모든 visualization object에 발급되는 고유번호로, GIVI와 햅틱 서버에서 동일한 mesh는 같은 mesh ID를 갖고 있다.
Data organization		unsigned char	1	1 : Surface scalar 2 : Surface vector 3 : Stream / pathline
Haptic method		unsigned char	4	햅틱 렌더링 방법 (reserved)
Vertex count		unsigned int	4	Mesh를 구성하는 전체 vertex 개수
X-coord		float	4	Vertex 좌표. (X, Y, Z) tuple이 vertex count만큼 반복된다.
Y-coord		float	4	
Z-coord		float	4	
Data values		unsigned char	1	각 vertex에 bound 된 데이터를 0~255로 정규화(normalize) 한 값. 스칼라 데이터는 1byte, 벡터 데이터는 3bytes로 표현한다.
Strip count		unsigned int	4	Triangle strip 개수
반복	Strip ID	unsigned int	4	0 ~ (strip count - 1)
	Vertex count	unsigned int	4	해당 strip을 구성하는 vertex 개수
	Index 1	unsigned int	4	Vertex index
	...	unsigned int	4	

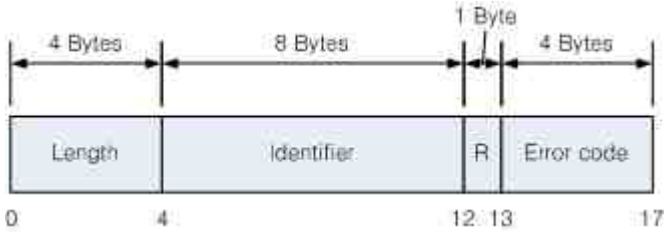
[표 3-8] Surface 추가 request의 세부내용

		Type	Bytes	Contents
Header		unsigned int	4	Message length
		char array	8	Identifier
OP code		unsigned char	1	Mesh 추가 request : 20
Rigidity		unsigned char	1	Rigid body : 'R', Soft body : 'S'
Workspace type		unsigned char	1	Cube : 'C'
Mesh ID		unsigned int	4	Mesh ID는 화면에 보이는 모든 visualization object에 발급되는 고유번호로, GIVI와 햅틱 서버에서 동일한 mesh는 같은 mesh ID를 갖고 있다.
Data organization		unsigned char	1	3 : Stream / pathline
Haptic method		unsigned char	4	햅틱 렌더링 방법 (reserved)
Vertex count		unsigned int	4	Mesh를 구성하는 전체 vertex 개수
X-coord		float	4	Vertex 좌표. (X, Y, Z) tuple이 vertex count만큼 반복된다.
Y-coord		float	4	
Z-coord		float	4	
Strip count		unsigned int	4	Triangle strip 개수
반복	Strip ID	unsigned int	4	0 ~ (strip count - 1)
	Vertex count	unsigned int	4	해당 strip을 구성하는 vertex 개수
	Index 1	unsigned int	4	Vertex index
	...	unsigned int	4	
Control point count		unsigned int	4	Tube 내에 존재하는 control point 개수
X-coord		float	4	Control point (X, Y, Z) tuple이 CP count만큼 반복된다.
Y-coord		float	4	
Z-coord		float	4	
X-coord		1	4	방향벡터 (X, Y, Z) tuple이 CP count만큼 반복된다.
Y-coord		1	4	
Z-coord		1	4	

[표 3-9] Surface 추가 request의 세부내용

2) Response (Haptic → GIVI)

Response는 명령 실행결과를 GIVI에게 알려주기 위해 사용한다. Mesh 추가작업을 정상적으로 수행하면 error code에는 0이 저장되며, 그 외의 경우는 [표 3-3]에서 정의한 에러코드를 돌려준다.



[그림 3-7] Calibration response sequence

Mesh 추가 response의 내용은 [표 3-10]과 같다. 문제가 발생하면 어떤 mesh를 추가하다가 문제가 발생했는지를 알려주기 위해 error code에 mesh ID가 포함되어야 하겠지만 GIVI와 햅틱 서버 사이의 command channel을 이용한 통신은 synchronous communication이기 때문에 에러가 바로 직전에 요청받은 mesh임을 암묵적으로 의미하는 것으로 간주하고 의도적으로 포함시키지 않았다.

Field name	Type	Bytes	Contents
Header	unsigned int	4	Message length
	char array	8	Identifier
Response code	unsigned char	1	Mesh 추가 response : 21
Error code	unsinged int	4	

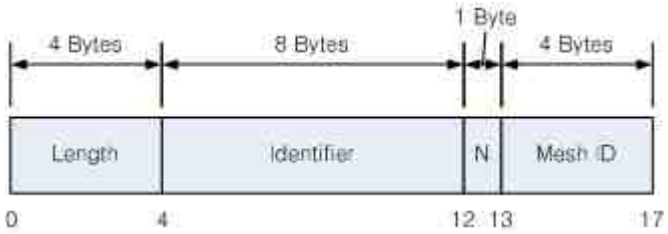
[표 3-10] Mesh 추가 response

바. Mesh 교체

햅틱 서버가 plane을 대상으로 햅틱 렌더링을 할 때와 3D 워크스페이스 내에 포함된 mesh를 대상으로 햅틱 렌더링을 할 때 약간의 차이가 있다. Plane을 대상으로 할 때에는 햅틱 렌더링을 종료할 때까지 다른 mesh를 선택할 수 없지만, 3D 워크스페이스 안에 다수의 mesh가 존재할 때에는 햅틱 펜(haptic pen)의 버튼을 누를 때마다 햅틱 렌더링 대상 mesh가 바뀐다. 그리고 햅틱 렌더링 대상이 바뀌었을 때 mesh 교체 notification을 GIVI에게 보내준다. Mesh 교체는 햅틱 서버가 GIVI에게 보내는 메시지이기 때문에 GIVI ↔ 햅틱 서버 사이의 통신채널 중 information channel로 메시지를 전송한다.

1) Notification (Haptic → GIVI)

워크스페이스에 포함되어 있는 mesh와 plane 모두 동일한 형태의 sequence를 사용한다. 이때 'Mesh ID' 필드는 Add mesh 명령을 통해 전달받은 고유번호를 이용한다.

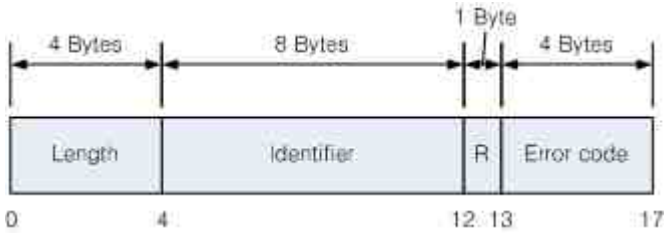


[그림 3-8] Mesh 교체 notification sequence

Field name	Type	Bytes	Contents
Header	unsigned int	4	Message length
	char array	8	Identifier
Command	unsigned char	1	Mesh 교체 notification : 30
Mesh ID	unsinged int	4	GIVI에서 지정한 ID

[표 3-11] Mesh 교체 notification의 세부내용

2) ACK (GIVI → Haptic)



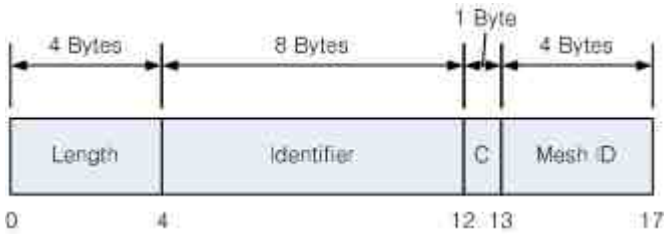
[그림 3-9] Change mesh ACK sequence

Field name	Type	Bytes	Contents
Header	unsigned int	4	Message length
	char array	8	Identifier
Response code	unsigned char	1	Mesh 교체 ACK : 31
Error code	unsinged int	4	

[표 3-12] Mesh 교체 acknowledge의 세부내용

사. Mesh 삭제

1) Request (GIVI → Haptic)



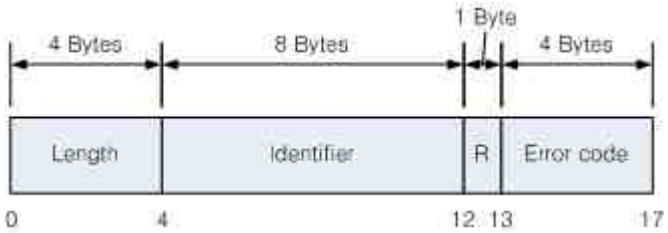
[그림 3-10] Mesh 삭제 command sequence

Mesh 삭제 request의 세부 내용은 [표 3-13]과 같다. 이 중 Mesh ID가 0xFFFFFFFF면 햅틱 서버는 현재 등록되어 있는 모든 mesh를 삭제하는 것을 의미한다.

Field name	Type	Bytes	Contents
Header	unsigned int	4	Message length
	char array	8	Identifier
Command	unsigned char	1	Mesh 삭제 : 40
Mesh ID	unsinged int	4	Mesh ID

[표 3-13] Mesh 삭제 request

2) Response (Haptic → GIVI)



[그림 3-11] Mesh 삭제 response sequence

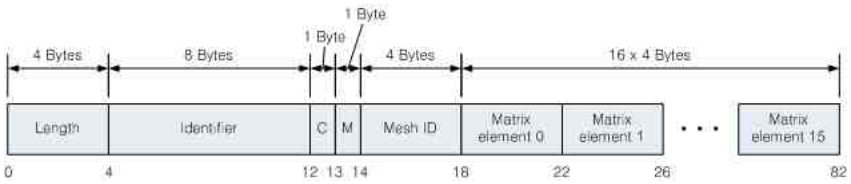
Field name	Type	Bytes	Contents
Header	unsigned int	4	Message length
	char array	8	Identifier
Response code	unsigned char	1	Mesh 삭제 response : 41
Error code	unsinged int	4	

[표 3-14] Mesh 삭제 response의 세부내용

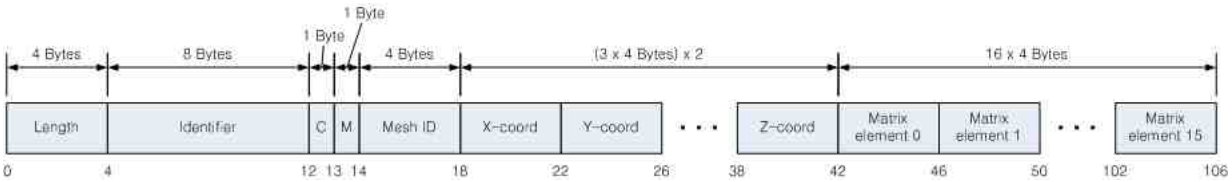
아. 햅틱 시작

1) Request (GIVI → Haptic)

햅틱 시작 request는 두 가지 메시지가 존재한다. 하나는 햅틱 렌더링 대상이 2차원 평면(clipping plane)인 경우를 위한 것이고, 다른 하나는 3차원 workspace가 지정되어 있을 때 사용한다.



[그림 3-12] Clipping plane에 대한 햅틱 시작 request



[그림 3-13] 3D workspace에 대한 햅틱 시작 request

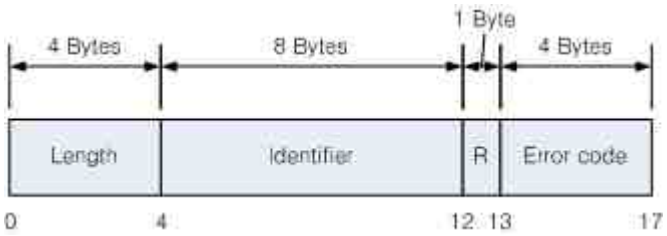
Clipping plane에 대한 햅틱 렌더링 시작 request에서 mesh ID는 특정 clipping plane을 지칭한다. 햅틱 렌더링 서버는 사용자의 입력에 관계없이 GIVI로부터 햅틱 종료 request를 받기 전까지는 해당 clipping plane에 대해서만 햅틱 렌더링을 실행한다. 반면 3D workspace에 대한 햅틱 시작 request는 최초 햅틱 렌더링을 시작하는 mesh ID를 알려주고, 사용자가 햅틱 펜의 버튼을 누를 때마다 순서대로 햅틱 렌더링 대상 ID를 바꿔간다. 단, 3D workspace를 대상으로 햅틱 렌더링을 할 때에는 clipping plane에 대한 햅틱 렌더링은 하지 않는다.

햅틱 시작 request의 세부내용은 [표 3-15]와 같다. 여기서 workspace의 lower(upper) left(right) corner 좌표는 GIVI의 screen space 상에서 AABB로 보이는 workspace를 world coordinate로 역변환 한 결과다. 그리고 matrix element는 OpenGL의 transformation matrix를 보내준다.

Field name	Type	Bytes	Contents
Header	unsigned int	4	Message length
	char array	8	Identifier
Command	unsigned char	1	햅틱 렌더링 시작 : 50
Mode	unsigned char	1	Clipping plane : 'P', Workspace : 'W'
Mesh ID	unsigned int	4	햅틱 렌더링 대상이 clipping plane이면 clipping plane의 ID를 가리키고, workspace인 경우에는 최초 햅틱 렌더링을 시작할 mesh ID를 지칭한다.
X-coord	float	4	Lower left corner of the workspace
Y-coord	float	4	
Z-coord	float	4	
X-coord	float	4	Upper right corner of the workspace
Y-coord	float	4	
Z-coord	float	4	
Matrix element 0	float	4	Transformation matrix for workspace OpenGL의 glGetMatrix 실행결과
...			
Matrix element 15	float	4	

[표 3-15] 햅틱 시작 request

2) Response (Haptic → GIVI)



[그림 3-14] 햅틱 시작 response sequence

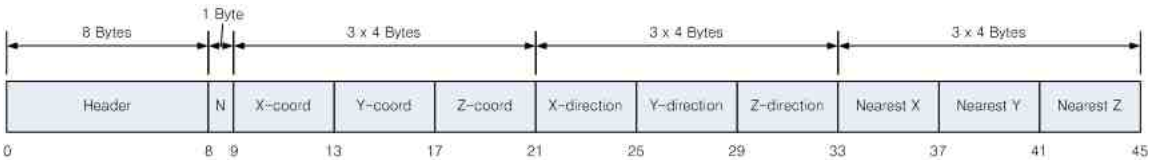
Field name	Type	Bytes	Contents
Header	unsigned int	4	Message length
	char array	8	Identifier
Response code	unsigned char	1	햅틱 렌더링 시작 response : 51
Error code	unsinged int	4	

[표 3-16] 햅틱 시작 response의 세부내용

자. Notification

1) Haptic pen (Haptic → GIVI)

이 메시지는 햅틱 펜(haptic pen)의 위치와 방향을 GIVI에게 알려주기 위해 사용한다. GIVI는 이 정보를 바탕으로 VR 스크린에 가상의 햅틱 펜을 그려서 실제 햅틱 렌더링이 진행되는 위치를 사용자에게 보여준다. 이 명령은 다른 명령들과 달리 response를 필요로 하지 않으며, 햅틱 렌더링이 진행 중인 동안 햅틱 서버는 쉬지 않고 GIVI에게 햅틱 펜의 위치를 알려줘야 한다.



[그림 3-15] Haptic pen notification sequence

햅틱 펜 정보 메시지는 [표 3-17]과 같다. 여기서 X, Y, Z 좌표는 햅틱 펜의 끝점을 의미한다.

Field name	Type	Bytes	Contents
Header	unsigned int	4	Message length
	char array	8	Identifier
Notification code	unsigned char	1	Haptic probe info : 60
X-coord	float	4	Start position of the haptic pen
Y-coord	float	4	
Z-coord	float	4	
X-direction	float	4	Direction of the haptic pen
Y-direction	float	4	
Z-direction	float	4	
Nearest X	float	4	Nearest point
Nearest Y	float	4	
Nearest Z	float	4	
Haptic rendering	char	1	햅틱 렌더링 여부 (true/false)

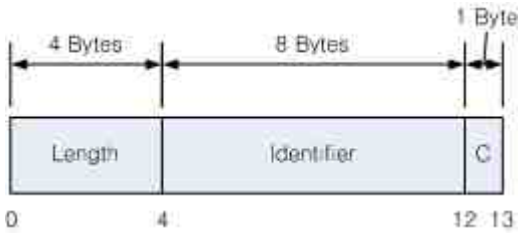
[표 3-17] 햅틱 펜 정보 메시지

차. 햅틱 중지

햅틱 렌더링이 수행 중일 때 이를 중지하기 위해 사용한다. 이 명령은 햅틱 렌더링이 진행 중일 때에만 유효하다.

1) Request (GIVI → Haptic)

햅틱 중지 명령은 별도의 추가 데이터를 필요로 하지 않기 때문에 전체 메시지는 헤더와 command opcode의 13바이트, 여기에 메시지 길이를 알려주는 추가 4바이트로 구성되어 있다 (전체 17바이트, [그림 3-16])



[그림 3-16] 햅틱 중지
request sequence

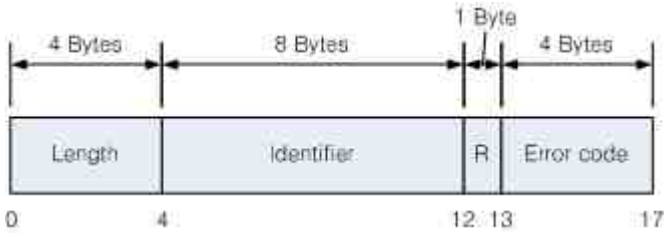
햅틱 중지 명령의 실제 내용은 [표 3-18]과 같다.

Field name	Type	Bytes	Contents
Header	unsigned int	4	Message length
	char array	8	Identifier
Command	unsigned char	1	햅틱 렌더링 중지 request : 70

[표 3-18] 햅틱 중지 command sequence의 실제 내용

2) Response (Haptic → GIVI)

햅틱 서버에서 햅틱 렌더링을 중단하고 그 결과를 돌려주기 위해 사용한다. 정상적으로 종료가 되면 error code에 0이 저장되고, 그 외의 경우에는 적절한 error code를 저장해서 돌려준다. 하지만 비정상 종료라고 해도 GIVI 입장에서는 햅틱 렌더링이 끝난 것으로 간주하고, 그 이후의 절차를 진행한다.



[그림 3-17] 햅틱 중지 response sequence

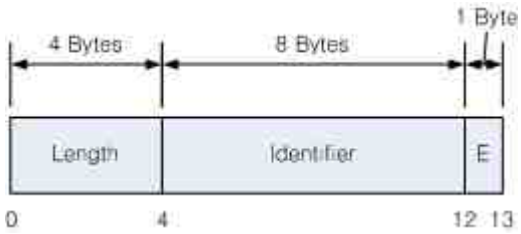
햅틱 중지 명령의 실제 내용은 [표 3-19]와 같다.

Field name	Type	Bytes	Contents
Header	unsigned int	4	Message length
	char array	8	Identifier
Response code	unsigned char	1	햅틱 렌더링 중지 response : 71
Error code	unsinged int	4	

[표 3-19] 햅틱 중지 response의 실제 내용

카. Unknown error

이 메시지는 햅틱 서버나 햅틱 디바이스에 문제가 발생해서 무조건 햅틱 렌더링을 종료하고 장비를 초기화시켜야 하는 상황이 발생했을 때 햅틱 서버가 GIVI에게 보낸다. 이 메시지를 받은 GIVI는 햅틱 렌더링 및 관련 기능을 모두 중단시킨 상태로 돌아가야 한다.



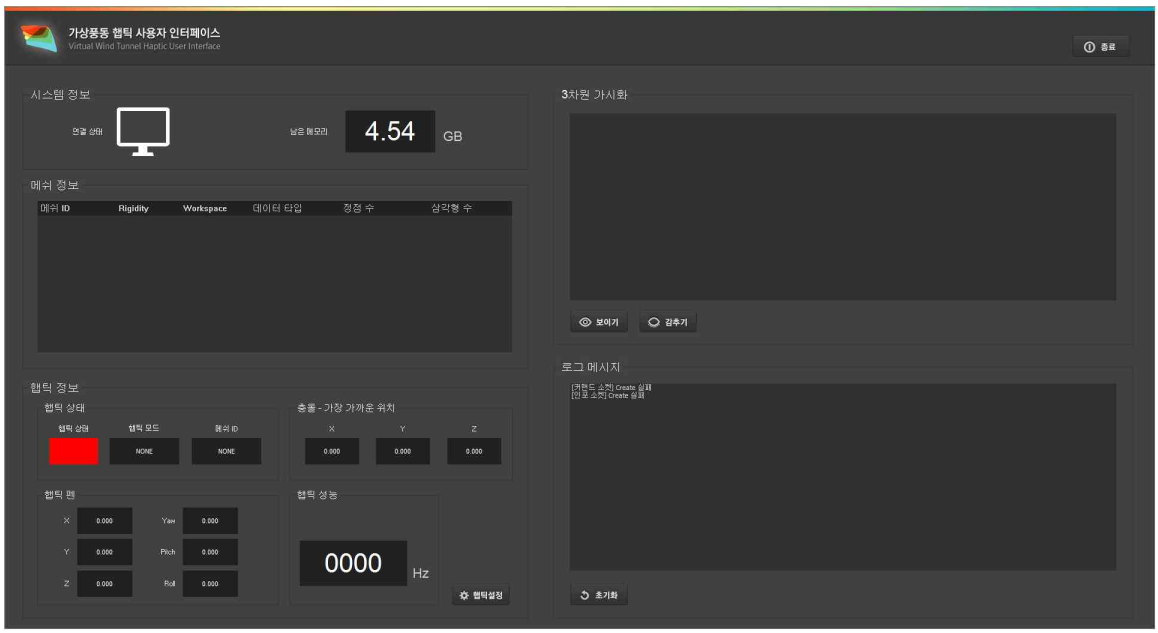
[그림 3-18] Unknown error sequence

Field name	Type	Bytes	Contents
Header	unsigned int	4	Message length
	char array	8	Identifier
Command	unsigned char	1	Unknown error : 255

[표 3-20] Unknown error sequence의 세부내용

4. Haptic 서버의 설계 및 구현

햅틱 서버는 GIVI로부터 geometry와 햅틱 데이터를 전송받아서 햅틱 렌더링을 수행하는 독립 어플리케이션이다. 햅틱 서버는 기본적으로 네트워크를 통해 GIVI와 관련정보를 주고받으며, 햅틱 장치를 제어하고 햅틱 렌더링에 필요한 자원을 관리한다.



[그림 4-1] 햅틱 서버 프로그램의 실행 모습

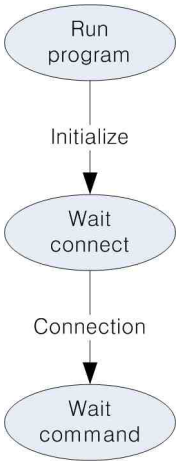
이 장에서는 GIVI의 명령에 대응하는 햅틱 렌더링 관련 기능의 실행흐름에 대해 설명한다.

가. 햅틱 명령의 처리

햅틱 서버의 거의 모든 기능은 GIVI의 명령에 따라서 작동한다. 그렇기 때문에 GIVI로부터 전달받은 명령에 따라서 어떻게 작동하는지를 명확하게 정의할 필요가 있다. 이 절에서는 햅틱 렌더링의 각 단계별로 햅틱 서버가 어떤 작업을 어떤 순서에 따라서 수행하는지 설명한다.

1) 실행 및 연결

이 단계는 햅틱 서버의 최초 실행 및 초기화와 관련이 있다. 사용자가 프로그램을 실행하면 햅틱 장치, 통신에 필요한 소켓 등을 초기화하고 GIVI로부터의 연결 요청을 기다린다.([그림 4-2])



[그림 4-2] 햅틱 시작 state transition diagram

GIVI와 햅틱 서버 사이의 모든 통신채널은 GIVI가 연결을 요청하고, 햅틱 서버가 승인하면 만들어진다. 통신채널이 만들어진 후 각종 메시지 전송이 가능하게 된다.

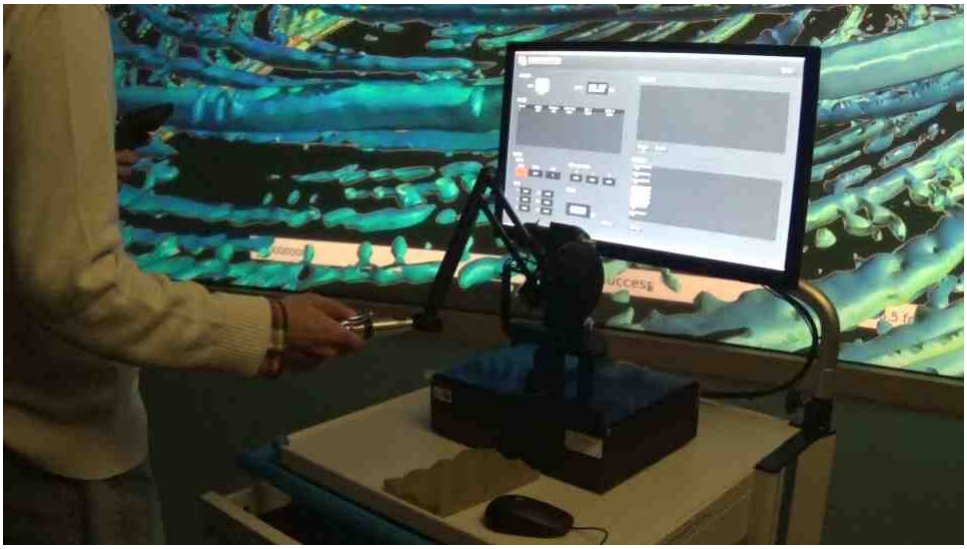
상태	설명
Run program	햅틱 장치 및 각종 파라미터 초기화
Wait connect	GIVI로부터의 연결 요청을 기다리는 상태
Wait command	GIVI로부터의 명령을 기다리는 상태

[표 4-1] 실행 및 연결 상태의 세부내용

[그림 4-2]의 state transition diagram을 보면 'Wait connect' 상태에서 'Wait command'로 바뀌는 이벤트(Connection)는 GIVI와 햅틱 서버가 정상적으로 연결됐을 때 발생한다.

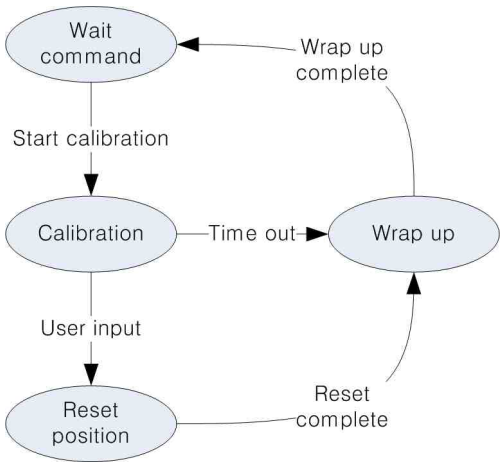
2) 햅틱 캘리브레이션

햅틱 장치의 위치를 보정하는 기능으로 햅틱 장치의 종류에 따라 방식이 다르다. 현재 사용중인 Phantom Premium 1.5 HF는 위치를 초기화하는 유형으로, 캘리브레이션 명령을 햅틱 장치로 전송하면 그 위치가 바로 영점(0, 0, 0)이 된다.([그림 4-3])



[그림 4-3] 햅틱 장치의 캘리브레이션을 위해 thumbpad의 버튼을 누른 모습

앞에서 설명한 '실행 및 연결'의 state transition과 마찬가지로 햅틱 캘리브레이션 역시 GIVI로부터의 명령과 사용자 입력이 이벤트로 작용한다. [그림 4-4]에서 'Wait command' → 'Calibration'으로의 상태전이는 GIVI로부터 전송된 calibrate 명령이 이벤트로 작용하기 때문이다. 그리고 'Calibration' → 'Reset position'으로의 상태전이는 사용자가 햅틱 펜의 thumbpPad를 눌렀기 때문에 발생한다.([그림 4-3])



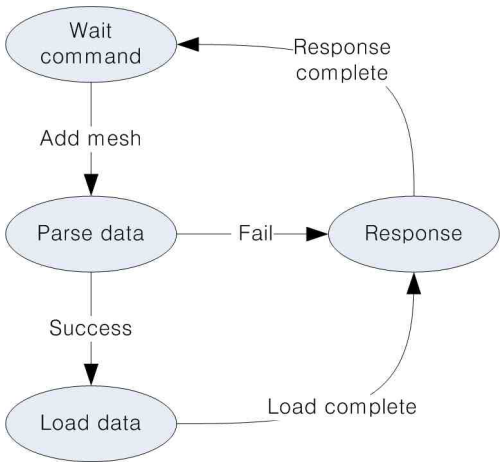
[그림 4-4] 햅틱 캘리브레이션 state transition diagram

노드	설명
Wait command	• GIVI로부터의 명령을 기다리고 있는 상태 • 햅틱 캘리브레이션 명령을 받으면 햅틱 장치에 캘리브레이션 서보루프를 등록하고 캘리브레이션을 시작한다.
Calibration	• 햅틱 장치의 thumbPad 입력을 기다리는 상태 • 사용자가 thumbPad를 누르면 캘리브레이션을 시작하고 서보루프를 종료 • 일정 시간이 지나도록 사용자 입력이 없으면 캘리브레이션을 하지 않고 서보루프 종료(time out)
Reset position	• 현재 햅틱 장치의 프로브 위치를 영점으로 지정
Wrap up	• 캘리브레이션의 결과를 클라이언트에 알리고 명령 대기 상태로 복귀

[표 4-2] 햅틱 캘리브레이션 내용

3) 메쉬 추가

햅틱 서버에 mesh를 등록하는 작업이다. 햅틱 서버가 사용하는 데이터(geometry 및 햅틱 데이터 일체)는 모두 GIVI로부터 전달받으며, 햅틱 렌더링을 수행하지 않을 때에만 새로운 데이터 등록이 가능하다.



[그림 4-5] 메쉬 추가 state transition diagram

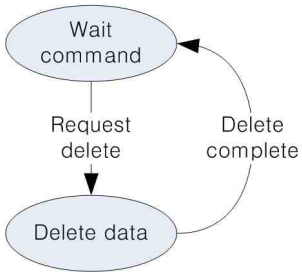
[그림 4-5]에서 Add mesh 이벤트는 GIVI가 보내는 명령으로, 이 때 mesh 데이터가 같이 전송된다. 그리고 'Success', 'Fail', 'Load complete' 이벤트는 햅틱 서버가 직접 발생시킨 이벤트다.

노드	설명
Wait command	• 클라이언트에서의 명령을 기다리고 있는 상태
Parse data	• 메쉬 추가 요청 명령과 함께 받은 메쉬 데이터 분석
Load data	• 메쉬 데이터의 분석을 성공적으로 완료하고 데이터에 이상이 없을 경우, 해당 메쉬 데이터를 햅틱 렌더링과 3D 가시화 등에서 사용할 수 있게 저장
Response	• 메쉬 추가의 결과를 클라이언트에 알리고 명령 대기 상태로 복귀

[표 4-3] 메쉬 추가 내용

4) 메쉬 삭제

클라이언트에서 메쉬 삭제 요청과 삭제해야 하는 메쉬 정보가 오면 햅틱 서버는 해당 메쉬 데이터를 찾아 삭제하고 그 결과를 클라이언트로 보낸다.



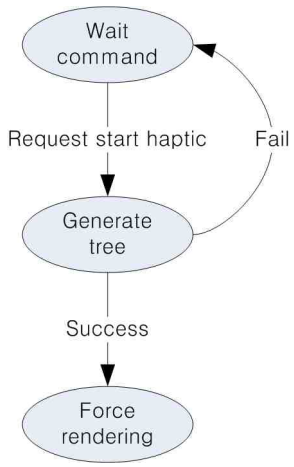
[그림 4-6] 메쉬 삭제 state transition diagram

노드	설명
Wait command	• 클라이언트에서의 명령을 기다리고 있는 상태
Delete data	• 요청된 메쉬 데이터를 삭제하고 그 결과를 클라이언트에 통지 후 wait command 상태로 복귀

[표 4-4] 메쉬 추가 내용

5) 햅틱 시작

GIVI로부터 햅틱 렌더링을 시작하는 명령을 받았을 때의 처리과정이다. 햅틱 서버는 1개 이상의 메쉬 및 햅틱 데이터를 가지고 있을 때 햅틱 렌더링을 실행할 수 있다. 햅틱 렌더링을 시작하기 위해서 클라이언트는 햅틱 서버로 햅틱 시작 요청과 함께 햅틱 렌더링이 이루어질 워크스페이스 혹은 대상에 대한 정보를 함께 준다.



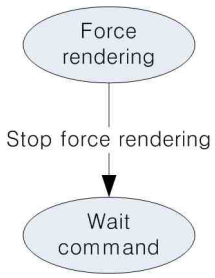
[그림 4-7] 햅틱 시작 state transition diagram

상태	설명
Wait command	• GIVI로부터의 명령을 기다리는 상태
Generate tree	• GIVI로부터 전달받은 geometry와 햅틱 데이터를 이용해서 데이터 트리 생성 (실시간 충돌검사용)
Force rendering	• 햅틱 장치에 포스 렌더링 서보루프를 등록하고 햅틱 렌더링 시작

[표 4-5] 햅틱 시작 내용

6) 햅틱 중지

햅틱 중지는 햅틱 서버가 햅틱 수행 상태일 때 실행 가능하다. 앞에서 설명한 '햅틱 시작'과 마찬가지로 GIVI가 햅틱 서버에게 햅틱 중지 명령을 보내면(이벤트) 햅틱 렌더링을 멈춘다.



[그림 4-8] 햅틱 중지 state transition diagram

상태	설명
Force rendering	• 햅틱 렌더링 진행 중
Wait command	• GIVI에게 햅틱 렌더링이 중지되었음을 통지

[표 4-6] 햅틱 중지 내용

나. 햅틱 서버의 구현

GIVI와의 통신은 MFC로 구현되었다. 그리고 햅틱 장치 제어 모듈은 OpenHaptics의 low-level API인 HD API로 구현했고, 충돌 검사에는 CGAL 라이브러리를 이용했다.

1) 햅틱 렌더링

① 충돌검사

충돌검사의 핵심 루틴은 CGAL 라이브러리로 구현했다. 특히 빠른 충돌검사를 위해 (3D) 데이터로부터 워크스페이스(workspace)를 추출하고, 이를 기반으로 AABB(axis aligned bounding box) 트리를 구성한다. 그리고 실제 충돌검사는 AABB 트리를 이용한다. 폴리곤과 포인트의 충돌을 감지하면 충돌지점에서 보간(interpolate) 된 scalar/vector 값을 햅틱 렌더링에 사용한다. 하지만 충돌지점을 직접 사용하기보다는 프로브와 가장 가까운 점(nearest point)을 찾고, 햅틱 프로브와의 거리에 따라서 폴리곤과 포인트의 거리에 따라서 서로 당기거나 밀어내는 힘이 추가되어 충돌보다는 찾아서 거리를 확인하는 방식을 사용한다.

② 햅틱 렌더링

햅틱 렌더링은 OpenHaptics 라이브러리의 low-level API인 HD로 구현되었다. 현재 햅틱 서버에 구현되어 있는 햅틱 렌더링 방식은 네 가지로 구분할 수 있다.

	세부내용
인력	평면(slice)과 스트림라인에 햅틱 프로브가 가까이 다가가면 프로브를 끌어당기는 힘
반력	특정 사물에 햅틱 프로브가 닿으면 프로브가 해당 메쉬를 뚫지 못하게 밀어내는 힘
-	평면 또는 메쉬와 가까운 위치에서 햅틱 프로브가 움직일 때 보간(interpolate) 된 scalar값에 비례하는 마찰력 / vector값의 방향과 크기를 가지는 힘
-	스트림 라인을 햅틱 프로브가 따라가게끔 유도하는 힘

[표 4-7] 햅틱 서버에 구현되어 있는 햅틱 렌더링 방법

2) 메시지 송수신

메시지 수신은 TCP 소켓으로 구현했으며, 송수신 도중 전체 시스템의 동작을 지연시키지 않도록 별도의 쓰레드가 소켓을 제어하도록 했다. 메시지 수신부는 [그림 4-10]에서 확인할 수 있듯이 먼저 메시지 길이(bytes)를 확인한 후 전체 메시지를 수신한다.

```
void
CCommandClntSocket::OnReceive (int nErrorCode)
{
CMainFrame *pFrame;

   pFrame = (CMainFrame*)AfxGetApp()->GetMainWnd();
   if (m_Flag == 1)
    {
      Receive(&m_nPacketSize, 4);
      m_Flag = 2;
      if (pFrame->m_CommandReceivedData)
       {
         delete pFrame->m_CommandReceivedData;
       }
      pFrame->m_CommandReceivedData = new byte[ m_nPacketSize];
      m_nCurrentReceivedSize = 0;
    }
   else
    {
     int nReceive;

      nReceive = m_nPacketSize - m_nCurrentReceivedSize;
      if (nReceive > 4096)
       {
          nReceive = 4096;
       }
      nReceive = Receive(m_buffer, nReceive);
      memcpy(&pFrame->m_CommandReceivedData[m_nCurrentReceivedSize],
             m_buffer, nReceive );
      m_nCurrentReceivedSize += nReceive;
      if( m_nCurrentReceivedSize >= m_nPacketSize )
       {
         SendMessage(m_hWnd, UM_COMMAND_MESSAGE_RECEIVED, 0,
                     (LPARAM) m_nPacketSize);
         m_Flag = 1;
       }
    }
   CSocket::OnReceive(nErrorCode);
}
```

[그림 4-9] 메시지 수신부의 구현

메시지 송신은 수신부와 마찬가지로 TCP 소켓으로 구현했다. [그림 4-11]에서 확인할 수 있듯이 먼저 메시지 길이를 송신한 후, 메시지를 4096 바이트 단위로 분할해서 보낸다.

```
void
CCommandClntSocket::SendData (byte *_data, int _size)
{
   Send(&_size, 4);

int totalBytes = 0;
int sentBytes;

   do
    {
     int leftBytes;

       sentBytes = 4096;
       leftBytes = _size - totalBytes;
       if (leftBytes < 4096)
        {
           sentBytes = leftBytes;
        }
       memcpy(m_buffer, &_data[totalBytes], sentBytes);
       sentBytes = Send(m_buffer, sentBytes);

       totalBytes += sentBytes;
       if (totalBytes >= _size)
        {
            break;
        }
    } while (sentBytes > 0);
}
```

[그림 4-10] 메시지 송신 구현

3) 캘리브레이션 서보루프

캘리브레이션 서보루프는 햅틱 장치의 프로브 위치와 사용자가 ThumbPad를 누르는지 확인하는 단순기능을 수행한다.

```
static HDCallbackCode
HDCALLBACK calibrationCallback (void *pUserData)
{
HDdouble pos[3] = get probe position;
HDdouble thumbpad = get thumb-pad value;

   pUserData.position = pos;
   pUserData.thumbpad = thumbpad;

   if thumbpad is down
    {
       Do calibration;
       finish;
    }

   return HD_CALLBACK_CONTINUE;
}
```

[그림 4-11] 캘리브레이션 서보루프 구현

4) 포스 렌더링 서보루프

포스 렌더링 서보루프는 힘을 계산하고 햅틱 장치로 렌더링하며 햅틱 장치의 프로브 위치와 사용자 입력을 전달한다.

```
static HDCallbackCode
HDCALLBACK forceRenderingCallback (void *pUserData)
{
   if tree is NULL
       return HD_CALLBACK_CONTINUE;

    rpos = convert pos to postion in rendering coordinate;
    ratt = convert att to attitude in rendering coordinate;
    bool forceOn = false;

   if tree.dataType is scalar
    {
       calculate distance between probe position and closest vertex position;
       if force rendering condition is OK
```

```
      {
         convert dist to magnet;
         convert probe velocity to friction;
         rendering magnet + friction;
         forceOn = true;
      }
   }
  else if tree.dataType is vector
   {
      calculate distance between probe position and closest vertex position;
      if distance is small enough
      {
         convert closest vertex's vector to force;
         rendering force;
         forceOn = true;
      }
   }
  else if tree.dataType is streamline
   {
      calculate distance between probe position and closest cp;
      if follow line is not started and distance is small enough
      {
         convert dist to magnet;
         rendering magnet;
         forceOn = true;
         if follow line start condition is OK
         {
            start follow line;
         }
      }
      else if follow line is started
      {
          double vec[3] = calculate current cp to next cp vector;
          double dist2 = calculate distance between vec and probe position;
          if dist2 is too big
          {
             end follow line;
          }
          else
          {
             convert dist2 to magnet;
             convert vec to force;
             rendering magnet + force;
             forceOn = true;
          }
      }
   }
```

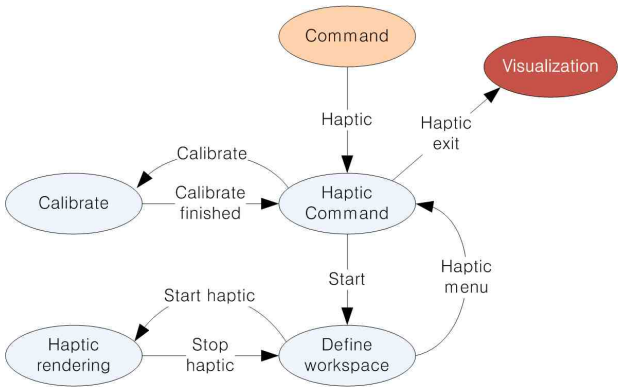
```
   send rpos, ratt, forceOn to Haptic Server Mainframe;
   (The Mainframe send rpos, ratt, forceOn to client)

   return HD_CALLBACK_CONTINUE;
}
```

[그림 4-12] 포스 렌더링 서보루프의 구현

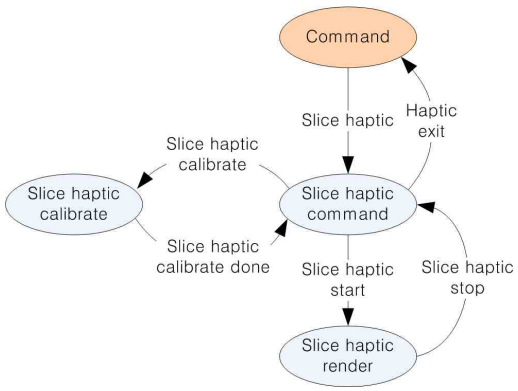
5. GIVI에서의 햅틱 정보 처리

가. 개요

GIVI에는 햅틱 렌더링을 위한 FSM state와 state transition 정보가 별도로 구현되어 있다. 특히 사용 및 구현상의 편의를 위해 햅틱 렌더링 영역(workspace)이 육면체(cube) 형태인 경우와 slice를 대상으로 햅틱 렌더링을 하는 경우를 분리했다. 아래의 [그림 5-1]과 [그림 5-2]는 이 두 가지 모드를 위한 FSM transition diagram을 각각 보여주며, [표 5-1]과 [표 5-2]는 두 diagram을 구성하는 FSM state의 세부내용을 설명한다.



[그림 5-1] HAPTIC state transition diagram



[그림 5-2] SLICE HAPTIC state transition diagram

여기서 주의 깊게 봐야 할 부분은 cube workspace를 사용하는 햅틱 렌더링은 workspace 지정을 위한 별도의 상태(Define workspace)를 필요로 하지만, slice에 대해 햅틱 렌더링을 수행할 때에는 대상 slice 자체가 workspace 역할을 맡기 때문에 workspace를 굳이 지정할 필요가 없다는 점이다.

State	세부내용
HAPTIC COMMAND	• 모든 haptic rendering 관련 명령이 시작하는 기본 대기상태
HAPTIC CALIBRATE	• Haptic device의 calibration 진행
HAPTIC WORKSPACE	• Haptic rendering 영역 지정
HAPTIC RENDER	• 햅틱 렌더링 진행

[표 5-1] HAPTIC 상태의 세부내용

State	세부내용
SLICE HAPTIC COMMAND	• 모든 slice haptic rendering 관련 명령이 시작하는 기본 대기 상태
SLICE HAPTIC CALIBRATE	• Haptic device의 calibration 진행
SLICE HAPTIC RENDER	• Slice에 대한 햅틱 렌더링 진행

[표 5-2] SLICE HAPTIC 상태의 세부내용

이 장에서는 위의 FSM을 구현할 때 필요한 루틴의 세부내용에 대해 설명한다. GIVI에서의 햅틱 정보 처리 루틴은 [표 5-3]과 같이 구분할 수 있다.

구분		세부내용
통신채널	command channel	햅틱 서버에 명령을 전달하고, 명령 실행결과를 확인하기 위한 통신 채널
	information channel	햅틱 렌더링이 진행 중일 때 프로브의 위치, 방향, 햅틱 렌더링 여부를 확인하기 위한 통신 채널
데이터 변환기		GIVI에 등록되어 있는 geometry를 햅틱 서버에 전달하기 위해 데이터를 변환하는 루틴의 집합
프로브 위치 표시		Information channel을 통해 전달받은 햅틱 프로브 정보를 이용해서 가상현실 공간에 가상의 햅틱 프로브를 시각적으로 보여주기 위한 루틴

[표 5-3] GIVI의 햅틱 관련 정보 처리 루틴의 분류

우선 통신채널은 command channel과 information channel로 구분할 수 있다. 이 중 command channel은 synchronous communication으로 충분하기 때문에 별도의 쓰레드를 구현하지 않아도 되지만 information channel은 햅틱 서버로부터 수시로 데이터를 받아야 하기 때문에 쓰레드를 따로 분리했다. 데이터 변환기는 화면에 보이는 geometry 중 햅틱 렌더링 데이터를 갖고 있는 것에 한해서 2장에서 설명한 프로토콜에 따라서 geometry 정보를 변환한다. 햅틱 서버는 이 정보를 전달받아서 독자적인 형태로 데이터를 재구성하고, 그 결과 데이터를 대상으로 햅틱 렌더링을 수행한다. 그리고 프로브 위치 표시 루틴은 햅틱 서버가 보내준 프로브 정보를 이용해서 현재 가상현실 공간의 어느 지점에 햅틱 프로브가 있으며 어느 지점의 데이터를 햅틱 렌더링하고 있는지를 시각적으로 보여준다.

나. Command channel

Command channel에서 사용하는 메시지의 송수신 루틴은 각각 givi::HapticIO 클래스의 Send와 Recv라는 멤버 함수로 구현되어 있다.

- givi::HapticIO::Send (haptic::Command &command)
- givi::HapticIO::Recv (haptic::Response &response)

앞에서도 언급했듯이 command channel은 synchronous communication을 사용하기 때문에 어플리케이션에서는 [그림 5-3]과 같이 사용한다. 그림에서 보듯이 어플리케이션은 Send 함수를 호출한 직후에 Recv 함수를 호출해서 바로 햅틱 서버로부터 응답을 기다린다.

```
// send command & geometry to the haptic server
// i->first : object ID,   i->second : pointer to the object
converter->Convert(hapticCommand, i->second);
gHapticIO->Send(hapticCommand);     // send geometry to haptic server
gHapticIO->Recv(response);          // get response from haptic serv
```

[그림 5-3] 햅틱 command channel의 실제 사용 예

GIVI가 햅틱 서버로 보내는 명령의 인코딩 루틴은 아래의 함수로 각각 분리 구현되어 있다. 아래 함수들은 command로 표현된 햅틱 관련 명령을 입력으로 받아서 octet이라는, byte stream으로 인코딩 한 결과를 돌려준다.

- givi::HapticIO::EncodeAddMesh (givi::Octet &octet, haptic::Command &command)
- givi::HapticIO::EncodeCalibration (givi::Octet &octet, haptic::Command &command)
- givi::HapticIO::EncodeScalarSurface (givi::Octet &octet, haptic::Command &command)
- givi::HapticIO::EncodeVectorSurface (givi::Octet &octet, haptic::Command &command)
- givi::HapticIO::EncodeLine (givi::Octet &octet, haptic::Command &command)
- givi::HapticIO::EncodeRemoveMesh (givi::Octet &octet, haptic::Command &command)
- givi::HapticIO::EncodeStartHaptic (givi::Octet &octet, haptic::Command &command)
- givi::HapticIO::EncodeStopHaptic (givi::Octet &octet, haptic::Command &command)

그리고 햅틱 서버로부터의 응답은 아래의 함수로 구현되어 있다. 이 함수들은 buffer, octet으로 표현된 데이터를 디코딩하고, 그 결과를 response로 돌려준다.

- givi::HapticIO::Decode (unsigned char *buffer, int length,
  haptic::Response &response)
- givi::HapticIO::Decode (givi::Octet &octet, haptic::Response &response)
- givi::HapticIO::DecodeResponse (givi::Octet &octet, haptic::Response &response)

다. Information channel

Information channel을 위한 쓰레드는 givi::HapticIO의 constructor에서 만들어진다.([그림 5-4])

```
givi::HapticIO::HapticIO ()
{

   중략

   // Start network receiver thread
   mInfoThread = new vpr::Thread(boost::bind(&HapticIO::InfoLoop, this));
   if (mInfoThread == NULL)
    {
       this->CloseCommSocket();
       this->CloseInfoSocket();
       LOG4CXX_ERROR(gvLogger, "Failed to create haptic IO thread");
       gContext->ResetHapticFlag();
    }
}
```

[그림 5-4] givi::HapticIO의 constructor 내에서 구현된 information thread의 생성

그리고 information thread는 [그림 5-5]와 같이 구현되어 있다. 큰 틀에서 보면 쓰레드가 종료될 때까지 아래의 세 작업을 반복수행하는 구조를 갖고 있다.

① 햅틱 메시지 수신
② 햅틱 메시지 디코드
③ 햅틱 메시지의 GIVI로의 전달

```
void
givi::HapticIO::InfoLoop (void)
{

   중략

   while (true)
    {
       usleep(1000);    // 과도한 CPU 부하 방지

       // 햅틱 메시지의 길이 확인 → 변수 'length'에 저장
       memset(buffer, 0, sizeof(unsigned char) * BUFSIZ);
       total  = recv(mInfoSocketFD, buffer, 4, 0);
       if (total <= 0)
        {
           continue;
        }
       memcpy(&length, buffer, 4);

       // 'length' 바이트의 데이터 수신 : 별도의 Recv 루틴을 필요로 하지 않는다.
       memset(buffer, 0, sizeof(unsigned char) * BUFSIZ);
       total = recv(mInfoSocketFD, buffer, length, 0);
       if (total <= 0)
        {
           continue;
        }

       // 메시지 헤더 체크 : 햅틱 프로토콜에 맞춰서 인코딩 된 메시지인지 판단
       mMutexCheckHeader.acquire();
       correctData = this->CheckHeader(buffer);
       mMutexCheckHeader.release();
       if (correctData == false)
        {
           continue;
        }

       this->Decode(buffer, total, info);
       this->Dispatch(info);
    }
}
```

[그림 5-5] Information thread의 구현

Information channel의 수신 루틴은 [그림 5]의 소스 내에 이미 구현되어 있으며, 메시지 송신 루틴은 givi::HapticIO 클래스의 Send로 별도로 구현되어 있다. Send 함수는 command channel의 Send 함수와 명칭은 동일하지만 parameter가 다르다.

- givi::HapticIO::Send (haptic::Acknowledgement &ack)

Information channel에서 사용하는 메시지의 encode 루틴은 아래와 같이 구현되어 있다. 이 중 상위 레벨 루틴은 givi::HapticIO::Encode 함수만 호출하고, Encode 함수 내에서 메시지 종류를 확인한 후 적절한 encode 루틴을 호출하도록 구현되어 있다. 그렇기 때문에 어렵지 않게 다른 메시지의 encode 루틴을 추가할 수 있다.

- givi::HapticIO::Encode (givi::Octet &octet, haptic::Acknowledgement &ack)
- givi::HapticIO::EncodeMeshChange (givi::Octet &octet,
  haptic::Acknowledgement &ack)

Information channel에서 사용하는 메시지의 decode 루틴은 아래와 같다. 앞에서 설명한 encode 루틴과 마찬가지로 상위 레벨 루틴이 호출하는 함수는 givi::HapticIO::Decode 하나로 통일되어 있으며, givi::HapticIO::Decoede 함수 내에서 메시지의 종류를 확인한 후 적절한 decode 루틴을 호출하도록 구현되어 있다.

- givi::HapticIO::Decode (unsigned char *buffer, int length, haptic::Information &info)
- givi::HapticIO::DecodeMeshChange (givi::Octet &octet, haptic::Information &info)
- givi::HapticIO::DecodeHapticPen (givi::Octet &octet, haptic::Information &info)
- givi::HapticIO::DecodeStatus (givi::Octet &octet, haptic::Information &info)

라. 햅틱 데이터 변환기

햅틱 데이터 변환기는 GIVI가 유지하고 있는 geometry 데이터를 햅틱 서버로 보내기 위해 octet stream으로 변환하는 루틴의 집합이다. 햅틱 데이터 변환기는 GIVI geometry를 크게 세 종류로 구분하고, 각 geometry 형태에 따라서 적절한 converter routine을 실행한다.

구분	대상 geometry	
Surface converter	givi::Surface givi::Isosurface	• Geometry 표면의 스칼라 또는 벡터 데이터를 normalize 한 후 햅틱 서버에 전송
Line converter	givi::Streamline givi::Pathline	• Streamline/pathline을 구성하는 control point와 direction vector를 normalize 한 후 햅틱 서버에 전송
Slice converter	givi::Slice	• Scalar distribution slice, contour slice, glyph slice 모두 포함

[표 5-4] 햅틱 데이터 변환기의 종류 및 세부내용

실제 데이터 변환은 프로토콜에 맞춰서 기계적으로 바꾸기 때문에 기술적으로 주목할 만한 내용은 없으며, 소스 코드를 보면 직관적으로 이해할 수 있다.

마. 햅틱 프로브 정보

햅틱 프로브 정보는 햅틱 서버가 GIVI에게 보내는 "Notification" 메시지에 포함되어 있다. 이 정보는 햅틱 렌더링이 진행 중인 동안에만 GIVI에게 전달되며, 메시지 전송 빈도는 초당 30회로 설정했다. 사실 햅틱 렌더링을 수행할 때 햅틱 디바이스의 데이터 샘플링 속도는 1000Hz에 육박하지만, GIVI의 frame rate가 60Hz 수준으로 고정되어 있고 사람은 frame rate가 30Hz를 넘어가면 사실상 real-time rendering이 되는 것으로 인지하기 때문에 전송 빈도를 대폭 낮췄다. "Notification" 메시지의 햅틱 프로브 정보는 [표 5-5]에서 확인할 수 있다.

구분	정보 형태	내용
햅틱 프로브 위치	(x, y, z)	• 햅틱 프로브의 world coordinate 상에서의 위치
햅틱 펜의 방향	(x, y, z)	• 햅틱 펜의 world coordinate 상에서의 방향 • VR 인터페이스에서 햅틱 펜을 특정한 geometry로 보여줄 때 rotation 정보로 사용
Nearest point	(x, y, z)	• 햅틱 프로브와 가장 가까운 햅틱 데이터의 위치
햅틱 렌더링 여부	true / false	• 햅틱 프로브가 실제로 데이터를 렌더링 할 때 true, 그렇지 않을 경우에는 false

[표 5-5] 햅틱 펜 정보 메시지

[그림 5-6] "Notification" 메시지의 정보를 이용해서 햅틱 프로브를 VR 화면에 출력한 모습
(좌측 상단 하얀색 sphere)

6. 결론

지금까지 가상현실 기반 visualization 어플리케이션과 햅틱 렌더링 서버의 연계를 위해 ① 네트워크 프로토콜 디자인 ② 햅틱 서버의 구현 ③ 가상현실 어플리케이션의 추가개발의 세 주제로 나눠서 설명했다. 실험결과를 보면 1Gbps 급의 느린 네트워크에서도 2~3백만 개의 triangle로 구성된 geometry는 햅틱 서버로의 전송, 햅틱 렌더링 등에 큰 무리가 되지 않는 수준의 성능을 보여줬다.

앞으로 추가연구가 필요한 분야는 다음과 같다. 현재 햅틱 서버와 가상현실 어플리케이션이 물리적으로 분리되어 있는데, 이를 하나의 어플리케이션으로 통합해서 데이터 전송과 가공 오버헤드를 제거하는 방법에 대한 고민이 필요하다. 두 번째 문제는 CFD 데이터의 특성을 반영한 햅틱 렌더링 방법론을 개발하는 것이다. 일반적으로 CFD 데이터에는 와류(vortex) 같이 유동이 심하게 소용돌이치는 영역이나 물리적 성질(속도, 점성 등)을 달리하는 유동이 혼합되는 영역이 포함되어 있는 경우가 매우 많다. 따라서 이러한 유동의 특성까지 고려한 햅틱 렌더링 방법을 구현할 수 있다면 유동의 특성정보를 사용자에게 보다 정확하게 전달하는 도구로서의 햅틱 렌더링 기술이 될 것이다.