ISBN 978-89-6211-601-4

VRJuggler Configuration을 이용한 가상현실 환경의 설정

허 영 주 (popea@kisti.re.kr)

한 국 과 학 기 술 정 보 연 구 원
Korea Institute of Science & Technology Information

목차

1. 서론 ····· 1

2. VRJuggler 설치방법 ····· 2

가. VRJuggler 설치에 필요한 패키지 ····· 2

1) Doozer ····· 2

2) Boost ····· 2

3) CppDom ····· 2

4) Scons ····· 3

5) GMTL (Generic Math Template Library) ····· 3

나. VRJuggler 설치 ····· 4

다. 환경 변수 설정 ····· 4

라. 설치시 발생할 수 있는 문제 ····· 5

3. VRJuggler Configuration ····· 7

가. 개요 ····· 7

나. 입력 디바이스 설정 ····· 8

1) Input manager ····· 8

2) 입력 디바이스 ····· 9

3) 디바이스 프록시 ····· 9

4) 포지션 필터 ····· 11

다. 디스플레이 설정 ····· 16

1) 디스플레이 매니저 ····· 17

2) 사용자(User) ····· 17

3) 디스플레이 윈도우 ····· 17
4) OpenGL 프레임 버퍼 ····· 17
5) 그래픽스 윈도우 입력 ····· 18
6) 뷰포트(Viewport) ····· 18
라. 클러스터 컴퓨터의 설정 ····· 20
마. Sonix Sound Manager ····· 21
1) sound API ····· 22
2) Listener position ····· 23
3) file search path ····· 23
4) sound object ····· 23
바. VRJuggler 설정방법 ····· 24

4. VRJuggler와 VTK의 좌표계 변환 ····· 37
가. VRJuggler의 행렬 변환 ····· 37
나. GMTL 행렬의 VTK 행렬로의 변환 ····· 38

5. VRJuggler와 CaveLib ····· 43
가. VR 라이브러리가 지원하는 시스템 ····· 43
나. CaveLib 환경과 VRJuggler 환경 ····· 44
1) CaveLib 프로그램과 VRJuggler 프로그램 ····· 44
2) Initialize, Draw, 그리고 Frame ····· 46
3) 디바이스로부터의 입력 ····· 47
4) 환경 설정 ····· 48

6. 결론 ····· 49

표 차례

[표 3-1] Intersense IS-900의 디바이스 프록시 ····· 10
[표 5-1] 각 라이브러리가 지원하는 플랫폼 ····· 43
[표 5-2] 각 라이브러리가 지원하는 디스플레이 디바이스 ····· 44

그림 차례

[그림 3-1] vrjconfig 실행 화면 ····· 7
[그림 3-2] 트래킹 시스템 설정시의 기준 좌표계 ····· 12
[그림 3-3] Keyboard/Mouse Input Handler 추가 ····· 24
[그림 3-4] Keyboard/Mouse Input Handler 추가 뒤의 모습 ····· 24
[그림 3-5] Keyboard/Mouse Proxy의 추가 ····· 26
[그림 3-6] Exit에 사용할 Digital Device 추가 ····· 26
[그림 3-7] Exit에 사용할 Digital Proxy 추가 ····· 28
[그림 3-8] Head 설정에 사용할 Positional Device의 추가 ····· 28
[그림 3-9] Head 설정에 사용할 Positional Device의 추가 ····· 28
[그림 3-10] Head 설정에 사용할 Positional Proxy의 추가 ····· 30
[그림 3-11] Head 설정에 사용할 Proxy Alias의 설정 ····· 30
[그림 3-12] Camera 설정에 사용할 Positional Device의 추가 ····· 30
[그림 3-13] Camera 설정에 사용할 Position Proxy의 추가 ····· 31
[그림 3-14] Simulator Device의 추가 ····· 33
[그림 3-15] Simulator Device에 대한 Proxy의 추가 ····· 33
[그림 3-16] User 추가 ····· 33
[그림 3-17] Display System의 추가 ····· 34
[그림 3-18] Display Window의 생성 ····· 35
[그림 3-19] Simulator Viewport의 설정 ····· 35
[그림 3-20] Simulator Plugin의 설정 ····· 35
[그림 4-1] Trackd Sensor Device 단위 ····· 40
[그림 4-2] Wand Proxy의 입력 단위 ····· 41
[그림 4-3] Head Proxy의 입력 단위 ····· 42

소스 차례

[소스 2-1] CppDom 설치시 발생할 수 있는 에러 ····· 2
[소스 2-2] Sconstruct에서 주석처리할 부분 ····· 3
[소스 2-3] GMTL 설치시 발생할 수 있는 에러 ····· 3
[소스 2-4] 에러를 발생시키는 try 구문 ····· 4
[소스 2-5] 환경변수 설정예 ····· 5
[소스 4-1] gmtl::Matrix44f를 이용한 translation 행렬 생성 ····· 37
[소스 4-2] transform 위치로 바로 옮겨가게 하는 코드 ····· 37
[소스 4-3] scale 변환을 수행하는 행렬 생성 ····· 37
[소스 4-4] 변환 정보를 알아내는 코드 ····· 38
[소스 4-5] gmtl 행렬을 vtkTransform 행렬로 변환하는 코드 ····· 39
[소스 4-6] 애플리케이션의 기본 단위로 위치 데이터를 요청하는 방법 … 40
[소스 5-1] CaveLib 프로그램의 형태 ····· 45
[소스 5-2] VRJuggler 프로그램의 형태 ····· 46

1. 서론

VRJuggler는 PC같은 단순한 데스크탑 시스템에서부터 클러스터나 워크스테이션, 혹은 슈퍼컴퓨터 시스템에 탑재되는 복잡한 멀티-스크린 시스템에 이르기까지 확장이 가능한 가상현실 애플리케이션 구축을 위한 플랫폼을 제공하는 통합 환경 라이브러리다. 즉, VRJuggler는 다른 Juggler 요소들간의 "접착제" 역할을 수행하는 것으로, PC같은 단일 데스크탑 시스템에서부터 하이-엔드 워크스테이션이나 슈퍼컴퓨터와 맞물려 돌아가는 복잡한 멀티-스크린 시스템에 이르기까지, 다양한 시스템에 적용할 수 있다. VRJuggler가 수행하는 기능은 매우 다양하며, 여러 다양한 툴을 기반으로 다음과 같은 기능을 지원한다.

- VRJuggler 가상 플랫폼
- Gadgeteer라는 디바이스 관리 시스템 (I/O 디바이스에 대한 로컬, 혹은 원격 접속)
- 수학 템플릿 라이브러리
- 크로스-플랫폼 쓰레드, 소켓 및 직렬 포트 프리미티브를 제공하는 런타임
- Sonix라는 사운드 매니저
- Tweek이라는 분산 모델-뷰 컨트롤러
- XML을 기반으로 하는 설정 시스템

VRJuggler는 데스크탑 VR, HMD, CAVE, 그리고 Powerwall에 이르기까지 매우 다양한 환경을 지원할 수 있기 때문에 유연성이 높으며, 다양한 OS를 지원할뿐만 아니라 프로그래밍 언어간 이식성도 높기 때문에 가상현실 환경을 구축하는데 있어 매우 유용한 툴이라 할 수 있다.

그러나 이런 다양한 환경을 지원하다보니 여러 가지 툴을 복합해서 사용해야 할 뿐만 아니라 구축하고자 하는 VR 환경에 따라 그 설정 방법이 달라지기 때문에 설정 방식 자체가 복잡해질 수밖에 없다. 본 문서에서는 다양한 VR 환경을 구축하는데 필요한 VRJuggler의 환경 설정방법과 KISTI에서 보유하고 있는 Picasso 시스템에서의 VRJuggler 설정 방법에 대해 다룰 것이다. 또, VRJuggler의 설치시 발생할 수 있는 문제점과 해결책 및 VTK 애플리케이션 개발시 필요한 애플리케이션간 좌표 변환방법, 그리고 CaveLib과 VRJuggler에 대한 비교분석을 다루기로 한다.

2. VRJuggler 설치방법

가. VRJuggler 설치에 필요한 패키지

VRJuggler를 설치하기에 앞서 설치돼 있어야 하는 프로그램 패키지는 다음과 같다.

1) Doozer

VRJuggler의 샘플 애플리케이션을 컴파일할 때 필요한 크로스-플랫폼 빌드 시스템으로, 주로 VRAC 워크스테이션과 PC에서 VRJuggler를 포함한 3D 그래픽스 애플리케이션을 컴파일하고 링크하는데 사용된다.

2) Boost

이식성 있는 C++ 소스 라이브러리를 제공하며, Fedora Core의 경우에는 기본으로 주어지는 yum 디렉토리에서 바로 설치할 수 있다.

3) CppDom

DOM 표현방식을 사용하는 C++ 기반의 XML 로더다. 매우 가볍고 성능이 좋기 때문에 많이 사용되고 있으며, JDOM과 유사한 API와 기능을 제공하는, XML 프로그래밍을 위한 C++ 인터페이스를 목표로 한다.CppDom은 scons, flagpoll, doxygen, graphviz등을 추가로 필요로 하는데, CppDom을 컴파일할 때 다음과 같은 에러가 발생할 수 있다.

```
AttributeError: 'module' object has no attribute 'is_valid_construction_var':
  File "/usr/src/redhat/BUILD/cppdom-0.7.10/SConstruct", line 83:
    opts.AddOption(sca_opts.SeparatorOption("\nPackage Options"))
  File "deps/scons-addons/src/SConsAddons/Options/Options.py", line 586:
    if not SCons.Util.is_valid_construction_var(k):
error: Bad exit status from /var/tmp/rpm-tmp.93417 (%build)


RPM build errors:
    Bad exit status from /var/tmp/rpm-tmp.93417 (%build)
```

[소스 2-1] CppDom 설치시 발생할 수 있는 에러

이런 에러가 발생하는 이유는 CppDom 패키지에 포함돼 있는 Scons 추가 모듈(Scons addon module)이 이상하기 때문인데, 시스템에 독자적으로 추가 모듈을

설치한 후 패키지의 SConstruct를 수정하면 된다.

4) Scons

Scons는 오픈 소스로 구성된 소프트웨어 구축용 툴이다. VRJuggler는 scons-addons도 필요로 하는데, CppDom의 source rpm을 가져오면 추가 모듈도 포함돼 있기는 하지만, Fedora Core 9에 패키지로 들어가 있는 추가 모듈로는 정상적인 컴파일이 불가능하다. 따라서 rpmbuild 실행 과정에서 source tarball을 풀고난 뒤, 컴파일에 들어가기 직전에 SConstruct의 다음 부분(line 10~11)을 주석처리함으로써 시스템이 자체 추가 모듈이 아닌, 시스템에 설치된 추가 모듈을 이용하게 해야 한다.

```
sys.path.insert(0, os.path.join('deps', 'scons-addons', 'src'))
print "NOTE: The build is currently in development.
 It needs the SVN trunk version of scons-addons"
```

[소스 2-2] Sconstruct에서 주석처리할 부분

5) GMTL (Generic Math Template Library)

rpmbuile로 GMTL을 설치할 때 다음과 같은 에러가 발생할 수 있다.

```
scons: Reading SConscript files ...
AttributeError: 'module' object has no attribute 'options':
  File "/usr/src/redhat/BUILD/gmtl-0.5.4/SConstruct", line 30:
    except AttributeError: has_help_flag = SCons.Script.options.help_msg
error: Bad exit status from /var/tmp/rpm-tmp.22161 (%build)

RPM build errors:
    Bad exit status from /var/tmp/rpm-tmp.22161 (%build)
```

[소스 2-3] GMTL 설치시 발생할 수 있는 에러

에러의 원인은 GMTL의 SConstruct에 있는 try 구문 때문인데, 이 구문을 넘어가도록 주석처리하고 설치한다.

```
try: has_help_flag = SCons.Script.Main.options.help_msg
except AttributeError: has_help_flag = SCons.Script.options.help_msg
--------
```

```
has_help_flag =''
#try: has_help_flag = SCons.Script.Main.options.help_msg
#except AttributeError: has_help_flag = SCons.Script.options.help_msg
```

[소스 2-4] 에러를 발생시키는 try 구문

나. VRJuggler 설치

VRJuggler를 설치하는 순서는 다음과 같다.

1. Juggler 소스 디렉토리에서 ./autogen.sh를 실행한다.
2. 소스 디렉토리에 build 디렉토리를 생성한다.
3. build 디렉토리에서 다음 명령을 실행해서 configure.pl을 실행한다.
   - ../configure.pl --prefix=$HOME/vrjuggler --with-java-orb=JDK
   - 이 때, boost 디렉토리를 --with-boost 옵션으로 지정할 수도 있다.
     (../configure.pl --with-boost=/home/user1/pkga/boost)
   - Mondrian에서는 다음과 같이 실행했다.
     (./configure/pl --prefix=/usr/local/VRJuggler --with-boost-includes =/usr/include/)
4. make install을 실행한다.
5. make build를 실행한다. 이 때, Fedora Core 9에서는 다음과 같은 2가지 에러가 발생할 수 있다.
   - memcpy와 유사한 함수가 정의되지 않았다는 에러가 발생할 수 있는데, 이는 소스코드 내에 string.h가 포함돼지 않았기 때문으로, 에러가 발생한 소스 파일의 첫 부분에 #include <string.h>를 추가하면 해결할 수 있다.
   - stropts.h를 찾지 못한다는 에러가 중간에 발생하는데 단순히 주석처리하면 넘어갈 수 있다.

다. 환경 변수 설정

VRJuggler 설치시 설정해야 하는 환경 변수는 다음과 같다.

- VJ_BASE_DIR (ex. VJ_BASE_DIR=/home/software/vrjuggler)
- VJ_DATA_DIR (ex. VJ_DATA_DIR=$VJ_BASE_DIR/share/vrjuggler-2.2)

- LD_LIBRARY_PATH에 $VJ_BASE_DIR/lib 를 추가한다.
- FLAGPOLL_PATH를 설정한다. (*.fpc 파일이 있는 디렉토리로, 일반적으로는 /usr/lib/flagpoll, /usr/lig64/flagpoll, /usr/share/flagpoll, /usr/lib/pkgconfig, /usr/lib64/pkconfig, /usr/share/pkconfig 순으로 검색된다.)
- 다음은 환경변수 설정 예다.

```
export VJ_BASE_DIR=/usr/local/pkg/vrjuggler/current
export PATH=${PATH}:${VJ_BASE_DIR}/bin
export DZR_BASE_DIR=${VJ_BASE_DIR}/share/Doozer
export JCCL_BASE_DIR=${VJ_BASE_DIR}
export VJ_CONFIG_FILES=${VJ_BASE_DIR}/share/vrjuggler/data/configFiles
export VJ_CFG_PATH=${VJ_BASE_DIR}/share/vrjuggler/data/configFiles
export VPR_DEBUG_NOTIFY_LEVEL=2
export VPR_DEBUG_ENABLE=1
export JDK_HOME=/opt/java/jdk1.5.0_05
juggler_libs="${VJ_BASE_DIR}/lib:/usr/local/lib64:/usr/local/lib"
if [ -z "$LD_LIBRARY_PATH" ]; then
   export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:$juggler_libs
else
   export LD_LIBRARY_PATH=$juggler_libs
fi
unset juggler_libs

   In addition, a symbolic link is normally created from the user's home d
  irectory to the directory containing runtime configuration files. An exa
  mple of how this is done is listed below.
   ln -s $VJ_BASE_DIR/share/vrjuggler/data/configFiles/ ~/.vjconfig
```

[소스 2-5] 환경변수 설정예

라. 설치시 발생할 수 있는 문제

VRJuggler 설치시 발생할 수 있는 에러와 그 해결책이다.

- memset was not declared in this scope
  - 문제파일: modules/vapor/vpr/IO/BlockIO.h, modules/vapor/vpr/IO/BufferObjectReader.h
  - 해결방법: #include <string.h>를 파일 앞부분에 추가한다.
- strerror was not declared in this scope
  - 문제파일: modules/vapor/펙/md/common/InetAddrHelpers.cpp
  - 해결방법: #include <string.h>를 파일 앞부분에 추가한다.
- memcpy was not declared in this scope

- 문제파일: modules/sonix/plugins/OpenAL/OPenALSoundImplementation.cpp
- 해결방법: #include <string.h>를 파일 앞부분에 추가한다.

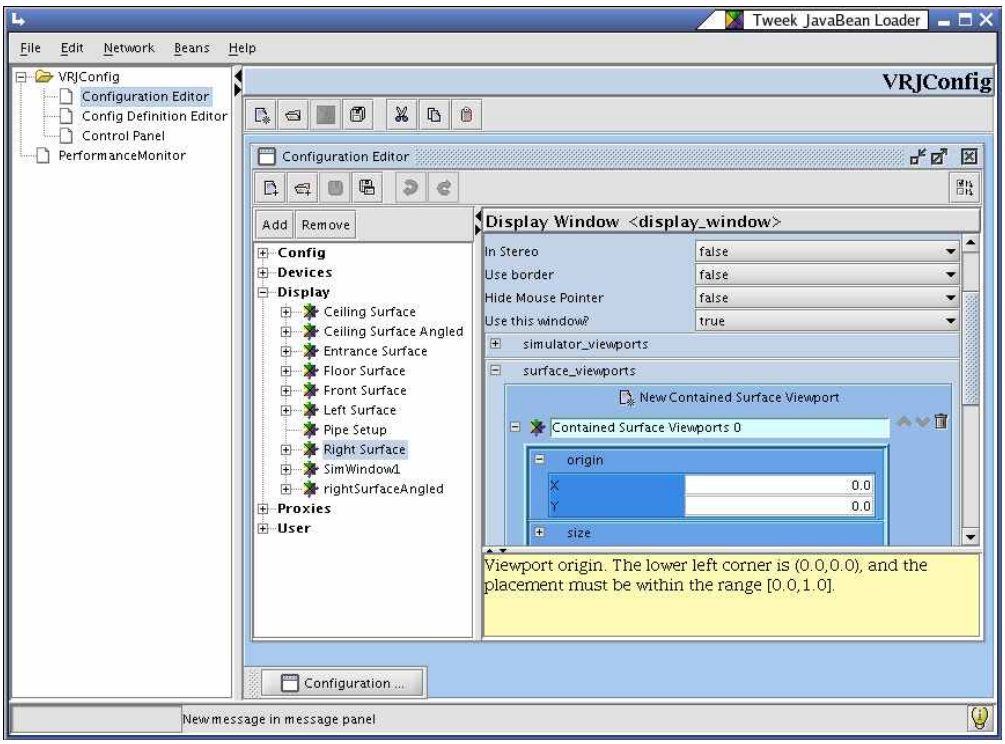
- stropts.h: No such file or directory
  - 문제파일: modules/gadgeteer/drivers/3Dconnection/SpaceBall/puck_device.cpp
  - 해결방법: #include <stropts.h>를 #include <sys/ioctl.h>로 변경한다.
- /boost/version.hpp is not readable
  - 원인: boost의 헤더 파일이 설치된 디렉토리를 찾지 못하기 때문에 발생하는 에러로, 리눅스 배포판에 따라 에러 발생 여부가 달라진다.
  - 해결방법: configure.pl 스크립트를 실행할 때 --with-boost-includes=/usr/include를 지정한다.

3. VRJuggler Configuration

가. 개요

VRJuggler는 vrjconfig를 이용해서 환경을 설정한다. vrjconfig는 VRJuggler의 bin 디렉토리에 설치되는 응용 프로그램으로, VRJuggler의 configuration을 편집하는 데 사용되는 JAVA 기반의 GUI 툴로써, XML로 구성된 .jconfig 파일을 생성한다.

[그림 3-1]은 vrjconfig의 설정 윈도우를 나타낸다.



[그림 3-1] vrjconfig 실행 화면

VRJuggler는 윈도우의 wizard와 비슷한 개념인 컨트롤 패널(control panel)을 제공하긴 하지만, 이것은 기존의 설정 파일을 편집하는 데만 사용할 수 있을 뿐, 설정 파일을 새로 생성하는 데는 사용할 수 없다.

VRJuggler의 설정 파일에서 가장 중요한 부분은 인풋(input)과 디스플레이(display), 이 2부분이다. 이 2가지만 설정하고 나면 대부분의 프로그램을 실행해서 볼 수 있다. 일반적으로 설정 파일을 작성할 때는 (physical한 형태건 simulation 형태건 간에) 일단 인풋과 관련된 디바이스와 프록시들을 설정하고 카메라와 관련

된 디바이스를 설정한 뒤, 디스플레이 부분을 생성하는 순서로 진행하는 것이 가장 간편하다. 그리고, 어떤 형태의 입력이건 간에 키보드부터 등록하고 나서 작성하는 것이 좋다. 즉, (device->proxy)->user->display(혹은 cluster), 이 순서로 설정 파일을 작성하되, 입력과 관련된 부분을 가장 먼저 작성하는 것이 가장 직관적인 방법이라 할 수 있겠다. (device와 proxy는 서로 엮이므로 순서가 바뀔 수도 있다.)

나. 입력 디바이스 설정

입력 디바이스는 Input device(physical 혹은 simulator)를 설정하고 실제 디바이스를 나타내는 프록시(proxy)를 설정한 뒤, 프록시 앨리어스(proxy alias)를 설정하는 순서로 설정 과정이 진행된다.

1) Input manager

Input manager는 디바이스 드라이버를 로딩하는데 사용되는 툴로, 사용자는 VRJuggler 내에서 필요에 따라 마음대로 디바이스 드라이버를 로딩해서 사용할 수 있다. 이 때 config element를 사용해서 input manager를 설정하며, config element는 검색 경로, 드라이버 DLL 이르미, 그리고 로딩할 드라이버 스캔을 위한 디렉토리로 구성된다.

- 드라이버 검색 경로 (driver search path)
  - 드라이버 검색 경로는 드라이버 DLL을 검색할 수 있는 디렉토리 목록을 나타낸다.
  - 리눅스용 기본값은 $VJ_BASE_DIR/lib/gadgeteer/drivers다.
  - Gadgeteer와 함께 설치된 기본 디바이스 드라이버들은 기본 검색경로에서 찾아볼 수 있으므로 일반적으로는 드라이버 검색 경로를 설정할 필요는 없다. 단, VRJuggler 밖에 위치한 디바이스 드라이버를 위한 검색 경로를 제공하기 위해 이 속정을 설정하는 것이다. 사용자가 검색 경로를 추가할 경우, 기본값은 항상 디렉토리 목록의 맨 마지막에 위치하게 되며, 기본 디렉토리는 항상 맨 마지막에 검색된다.
- 드라이버 DLL 이름 (driver DLL name)
  - 특정 디바이스 드라이버를 로딩하는데 사용되는 속성이다.

- 드라이버 스캔 경로 (driver scan path)
  - 해당 디렉토리 내에 위치한 모든 디바이스 드라이버를 로딩하는데 사용한다.
  - 드라이버를 개별적으로 로딩할 필요가 없을 때 사용하는 속성이다.
  - Input manager의 메모리 오버헤드가 증가한다는 단점은 있지만, 드라이버 로딩과 관련된 설정 에러가 발생할 가능성은 상당히 줄일 수 있다.

2) 입력 디바이스

입력 디바이스는 크게 physical 디바이스와 simulator 디바이스로 나눌 수 있다. physical 디바이스란 기본적으로 몰입형 입력을 받아들이는 디바이스로, 여러 방향에 대한 DOF(Degree Of Freedom)를 갖는 디바이스나 사용자의 행동에 대한 특정 정보를 제공하는 디바이스라 할 수 있다.

Gadgeteer input manager 관점에서 봤을 때, physical 디바이스는 드라이버 플러그-인을 필요로 한다. 하드웨어와의 직접적인 커뮤니케이션 프로토콜을 구축하고 VRJuggler 애플리케이션으로 넘기기에 앞서 디바이스로부터 읽어들인 데이터를 해석하는 것은 드라이버의 역할이다. 따라서 각 physical 디바이스에는 고유한 디바이스 플러그-인이 존재한다. 결국 physical 디바이스를 설정한다는 것은 디바이스 드라이버를 설정한다는 것을 의미한다. 드라이버가 제대로 설정되지 않으면 input과 관련된 동작이 제대로 동작할 수 없다. 다시 말해서 드라이버가 input 디바이스로부터 데이터를 제대로 읽어들이지 못하면 VRJuggler로 데이터를 제대로 넘겨주지 못한다는 것이다. 디바이스 드라이버 플러그-인이 모든 physical 디바이스에 대해 존재하는 것은 아니다. 따라서 input manager를 제대로 설정하는 것이 절대적으로 중요하다. 디바이스 드라이버에서 사용하는 configuration element들은 드라이버에서 자체적으로 처리하며, input manager가 필요한 디바이스 드라이버를 로딩하지 못하면 config element는 처리할 수 없다. 그렇기 때문에 input manager를 설정하는 것은 physical device를 설정하는 작업의 일환으로도 볼 수 있다.

Simulator 디바이스는 키보드와 마우스로부터 입력을 받아들여서 input manager가 지원하는 형태의 디바이스 입력으로 바꿔주는 역할을 수행한다. simulator 디바이스를 설정하려면 키보드/마우스 입력 핸들러(Keyboard/Mouse Input Handler), 입력 윈도우 및 실제 시뮬레이터 디바이스를 설정해야 한다.

3) 디바이스 프록시

디바이스 프록시는 VR Juggler 애플리케이션과 특정 디바이스가 tightly-coupled되지 않도록 보장하는 추상화된 레이어로써, 디바이스 프록시가 존재함으로써 하드웨어 디바이스로의 직접적인 접근이 차단된다. 즉, 디바이스 프록시란 실제 디바이스로부터 나오는 데이터 소스에 대한 포인터라 할 수 있으며, 디바이스 프록시의 종류는 디바이스의 종류와 일치한다.

디바이스 프록시의 종류는 다음과 같다.

- Analog proxy
- Command proxy
- Digital proxy
- Digital glove proxy
- Gesture proxy
- Keyboard/mouse proxy
- Positional proxy
- String proxy

각각의 입력 소스에 대해서는 프록시가 하나씩 마련돼 있어야 하며, 여러개의 프록시가 하나의 입력 디바이스를 가리킬 수도 있다. 입력 디바이스는 자신의 입력 소스 타입에 대해 서로 다른 큐를 사용하며, 각 프록시에 대해 unit identifier라는 속성을 적절하게 설정해서 사용한다. 이 때 unit identifier는 다양한 입력 소스에 대해 zero-based index로 작용한다.

[표 3-1]은 InterSense IS-900(TM)에 대한 디바이스 프록시를 나타낸다.

Input Source	Proxy Type	Proxy Name	Unit Identifier
Head Tracker	Position	Head Proxy	0
Wand Tracker	Position	Wand Proxy	1
Wand Trigger Button	Digital	Button 0 Proxy	0
Wand Red Button	Digital	Button 1 Proxy	1
Wand Yellow Button	Digital	Button 2 Proxy	2
Wand Green Button	Digital	Button 3 Proxy	3

Wand Blue Button	Digital	Button 4 Proxy	4
Wand Joystick X-Axis	Analog	Joystick X-Axis Proxy	0
Wand Joystick Y-Axis	Analog	Joystick Y-Axis Proxy	1

[표 3-1] Intersens IS-900의 디바이스 프록시

4) 포지션 필터

트래커의 센서 정보를 처리할 때, 트래킹 시스템으로부터 나온 위치 정보를 바로 이용할 수는 있으나, 대체로 트래커와 그 센서로부터 받은 정보는 Real World와는 차이를 보인다. 이 때문에 트래킹 데이터는 common coordinate frame으로의 transform 과정을 거쳐야 한다. 즉, 트래커 시스템이 사용하는 좌표계는 VR Juggler 애플리케이션이 기대하는 좌표계와는 다르기 때문에 transform 과정이 필요하다는 것이다. 이는 VR Juggler가 특정 좌표계를 사용하기 EOans이 아니라, 애플리케이션 자체에서 몰입형 시스템이 표준 좌표계를 사용할 것을 기대하기 때문이다.

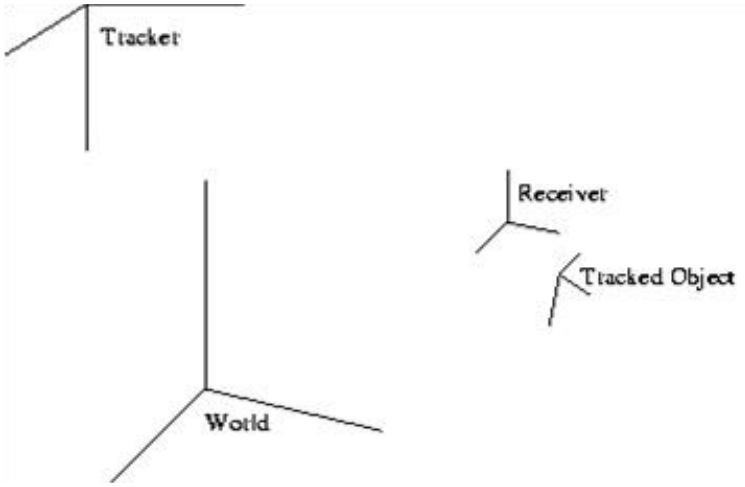
일단 여기에서 사용할 변환 행렬의 표기법은 다음과 같다.

- ${}_{frame1}M_{frame2}$: frame 1에서 frame 2로의 행렬 변환
- $P_{frame1} = {}_{frame1}M_{frame2} * P_{frame2}$: frame 2의 포인트들을 frame 1상의 포인트로 옮기는 행렬 변환
- ${}_{frame1}M_{frame2}$, ${}_{frame3}M_{frame2}$:
  - 이런 행렬들이 있을 때, ${}_{frame1}M_{frame3} = {}_{frame1}M_{frame2} * {}_{frame2}M_{frame3}$
  - 이 때, ${}_{frame2}M_{frame3} = inv\ ({}_{frame3}M_{frame2})$

변환 행렬에 대한 표기가 이와 같은 경우, 트래커 좌표 변환에 대해 알아보기로 한다.

트래킹 시스템 설정시의 기준 좌표계는 [그림 3-2]와 같으며, 각 좌표계에 대한 설명은 다음과 같다.

- World Coordinate Frame
  - 실제 world 시스템의 기본좌표계.
  - 즉, 몰입형 시스템의 물리적 공간에 대한 좌표계
  - VR Juggler의 좌표계는 일반적으로 X축이 오른쪽으로, Y축이 위로, 그리



[그림 3-2] 트래킹 시스템 설정시의 기준 좌표계

고 Z축은 화면에서 앞으로 튀어나오는 방향이다.(즉, 오른손 좌표계)

- Tracker
  - 트래킹 시스템에 대한 기본 프레임
  - 일반적으로 트래킹 시스템의 “base unit"에 의해 정의된다.
  - 예를 들어, Ascension Flock of Birds 트래킹 시스템의 경우, 이 coordina te frame은 transmitter의 위치에 의해 정의된다.
- Receiver
  - 물리적인 센서(physical sensor)에 대한 coordinate frame.
  - 일반적으로는 receiver에는 독립적인 좌표계가 존재하는게 아니라 트래킹 시스템에 대한 상대 위치가 존재하는 것으로 간주된다. 일견 이것은 사실이긴 하지만, 이 변환은 receiver에 대한 또다른 로컬 좌표계를 정의하는 것이기도 하다.
- Tracked Object
  - 실제로 트래킹하고자 하는 물체의 위치.
  - 경우에 따라서는 Receiver와 일치할 수도 있다.
  - Tracked object에 대한 가장 흔한 예는 사용자의 눈의 위치를 들 수 있다. 즉, 입체 안경에 부착된 센서의 위치, 혹은 사용자가 TM고 있는 헬멧에 부착된 오브젝트의 위치 등을 들 수 있다. 이 경우, tracked object에 대한 coordinate frame은 눈의 위치다.

이 때, 각 좌표계 사이의 변환 행렬은 다음과 같다.

- ${}_{world}M_{tracker}$
  - 실제 world에서 tracker base system의 위치를 잡는 행렬
- ${}_{tracker}M_{receiver}$
  - receiver에 대한 변환 행렬
  - 애플리케이션이 실행됨에 따라 변하는 유일한 행렬이며, 이것은 트래킹 시스템이 리턴하는 값이기도 하다.
- ${}_{receiver}M_{tracked-object}$
  - 트래킹 시스템이 리포트한 위치를 트래킹하고자하는 실제 위치로 변환하는 행렬
- 궁극적으로 얻고자 하는 행렬: ${}_{tracker}M_{tracked-object}$
  - 즉, real world에서의 tracked object의 위치
  - 최종 변환식: ${}_{world}M_{tracked\_object} = {}_{world}M_{tracker} \times {}_{tracker}M_{receiver} \times {}_{receiver}M_{tracked\_object}$
  - tracking coordinate system에서의 트래커의 위치(i.e.${}_{tracker}M_{receiver}$)는 실행 시간에 변화하는 유일한 값이며, 다른 두 값은 tracking system configuration element에서 값을 조정해야 한다.

이 때, 위치 정보를 리턴하는 각각의 오브젝트(즉, position device와 position device proxy)에는 포지션 필터(position filter) 속성이 존재한다. 포지션 필터는 positional object의 configuration element 내에 포함된 일련의 config element이며, 각각의 physical position device에는 이 속성이 존재한다. 포지션 필터에는 position transform filter, 이 한가지 속성이 있으며, 이 필터는 VR Juggler configuration 내에서 positional object에 transformation을 추가하는데 사용된다. 이 필터는 사용자가 정의한 transformation matrix를 pre/post multiflying한다.

포지션 필터를 설정하는 순서는 다음과 같다.

- 트래킹 장비를 설정하려면 트래킹 시스템으로부터 나온 값(${}_{tracker}M_{receiver}$)을 가상현실상에서의 트래킹 시스템의 위치(${}_{world}M_{tracker}$)에 의해 조정(pre-transform)해야 한다.
  - 이 때, 이 조정은 트래킹 디바이스의 config element에 대한 “position filter" 속성에 position transform filter를 추가하고, pre-translation과 pre-rotation 값을 설정함으로써 이뤄진다.

- 이 때, pre- 및 post-translation 속성의 물리적인 단위는 모두 미터로 설정해야 한다. (예외 없음.)

- pre-translation과 -rotation이 설정되고 나면 device unit을 결정해야 한다.
  - position tranform filter의 config element에는 단위를 tracking 단위에서 미터로 변환하는 속성이 존재한다.
  - 이 설정값은 트래커 하드웨어에 따라 달라지지만, VR Juggler 애플리케이션으로부터 받는 위치 데이터에 의심이 갈 경우에는 feet to meter conversion 속성을 사용해서 설정한다.
- tracked objec 위치를 설정하려면 tracked object에 대한 position proxy(ex. user head proxy)에 적용되는 transformation을 설정해야 한다.
  - 여기에 필요한 것은 값에 post-transform을 추가하는 것으로, positional device proxy에 대한 config 속성에서 “position filter" 속성에 position transform filter를 추가하고, 오브젝트를 트래킹하는데 사용되는 센서에 대한 tracked object의 상대적인 위치에 post-translation 및 post-rotation을 추가한다.
  - position 디바이스 프록시에 대해 position transform filter를 설정할 EO는 항상 unit conversion factor를 meter(1.0)로 남겨둬야 한다. 트래커 데이터가 프록시에 도달하는 시점에서는 이미 미터로 설정돼 있기 때문에 이 값을 다른 값으로 바꾸면 VR Juggler 애플리케이션 단에서 문제를 야기할 수도 있다.

이제 실질적으로 포지션 필터를 설정하는 방법에 대해 알아보자.

제일 먼저 할 일은, 가상 세계의 원점을 정의하는 것이다. 가상세계의 원점을 정의하는 방법은 다음과 같다.

- 가상 세계의 원점은 실제 공간(VR 시스템 공간)에서의 한 지점을 가리키며, [그림 3-2]에서 World 좌표계에서의 원점을 가리킨다.
- 원점을 정하는 방법
  - 바닥이 있는 프로젝션 시스템의 경우, 바닥의 중앙을 가상 세계의 원점으로 설정하는 것이 좋다. 이렇게 원점을 설정하면 사용자의 몰입감을 증가시킬 수 있다.

- 바닥이 없는 프로젝션 시스템의 경우에는 다소 복잡해진다. 원점을 화면에서 너무 먼 곳으로 정하면 가상현실상의 오브젝트가 뷰 밖으로 벗어날 확률이 높고, 원점이 화면 너머에 있으면 몰입감이 감소될 수 있다. 만약 원점이 바닥보다 위쪽에 있다면 사용자는 world를 낮은 곳에서 항상 올려다보게 될 것이다.
- Head-mouted display의 경우, 원점은 트래킹 시스템의 transmitter(혹은 emittor)에 상대적으로 정해야 한다. 예를 들어 transmitter가 천장에 매달려 있다면 원점을 transmitter 바로 아래로 정하는 것이 좋다. 만약 transmitter가 바닥에 놓여 있다면 원점을 transmitter의 바로 앞으로 정하는 것이 좋다.
- 원점은 반드시 tracked area 내에 존재해야 한다. 원점이 tracking되는 영역 내에 존재하지 않는 이유로 사용자가 애플리케이션 내에서 원점에 물리적으로 도달할 수 없다면 애플리케이션의 기능성이 떨어질 수 있다.

다음으로 할 일은 transmitter의 위치와 방향을 설정하는 것이다. transmitter의 위치와 방향을 설정하는 방법은 다음과 같다.

- 가상 세계의 원점이 정해지면 트래커 디바이스의 포지션 필터에 pre-translation 정보를 입력한다. 이는 트래커로부터의 정보를 가상 세계의 좌표계로 변환하기 위함이다.
- pre-translation 값의 계산
  - transmitter의 coordinate frame을 정한다. 일반적으로 이 값은 하드웨어에 의해 정해진다.
  - VR Juggler의 coordinate frame과 일치시키기 위한 회전값을 계산한다. 일단 VR Juggler는 내부적으로 Y가 위로 가는 오른손 좌표계를 사용한다.
  - 트래커의 coordinate frame을 하드웨어상에서 조정하는 것이 가능하다면 트래커를 조정하는 편이 쉽다.
- transmitter의 좌표계를 설정할 때 주의할 점은 다음과 같다.
  - 서로 다른 그래픽스 API는 서로 다른 coordinate frame을 사용한다. 하지만 VR Juggler를 사용할 때는 이런 점을 무시할 수 있다. VR Juggler의 coordinate frame을 그래픽스 API의 coordinate frame으로 변환하기 위

해 고민할 필요는 없다.

- 현재 포지션 필터에서는 왼손 좌표계를 오른손 좌표계로 바꾸는 방법은 제공하지 않는다. 따라서 트래커가 왼손 좌표계를 사용한다면 이를 표현할 방법은 VR Juggler에는 없다. 하지만 지금까지는 왼손 좌표계만 사용하도록 허용하는 트래커가 없기 때문에 큰 문제가 되지는 않을 것이다.

마지막으로 할 일은 트래킹된 포인트들을 정의하는 것이다. 포인트를 정의하는 방법은 다음과 같다.

- 여기에서 트래킹된 포인트라는 것은 트래킹 시스템의 센서를 가리킨다. 일반적으로는 헤드 센서와 wand 타입의 디바이스에 대한 센서가 존재한다.
- 예를 들어, 트래커가 입체 안경에 장착돼 있다고 할 경우, 트래킹된 포인트는 사용자의 눈 사이의 한 점이 될 것이다. 이 값은 센서에 대한 프록시에서의 post-transformation을 필요로 한다.
- 포지션 프록시에 대한 post-transformation
  - 우선 센서의 coordinate frame을 정의한다.
  - 그런 다음 센서에서 트래킹된 vhdlsxmRK지의 거리를 측정한다.
  - 센서에 대해 트래킹된 포인트가 가지는 상대적인 X, Y, Z값(미터 단위)이 바로 프록시에 대한 포지션 필터의 post-translation 값이다.
  - post-rotation 값을 얻으려면 VR Juggler coordinate frame(Y가 위로 향하는 오른손 좌표계)로부터의 rotation 값을 측정한다. 이 값을 degree 단위로 post-rotation값으로 입력하면 된다.
  - post-translation값을 측정할 때 주의해야 할 점은, 트래킹 시스템 transmitter에서의 측정방식과는 정 반대 방식으로 측정해야 한다는 것이다. transmitter의 경우에는 사용자가 정의한 원점에서부터 트래킹 시스템의 원점까지의 거리를 쟀고, rotation 값으로는 transmitter의 좌표축을 회전시켜서 VR Juggler 좌표축과 일치시키는데 필요한 값을 측정했었다.

다. 디스플레이 설정

디스플레이는 VR Juggler 설정 요소중 가장 중요한 부분이라 할 수 있는 부분으로, 이 속성의 설정만으로도 화면 디스플레이가 가능하다. 여기에서는 디스플레

이 설정에 필요한 여러 속성에 대해 알아보기로 한다.

1) 디스플레이 매니저

디스플레이 매니저는 디스플레이 설정에 가장 필수적인 요소로 “Display System " 속성에 의해 설정 가능하다. 클러스터 설정은 예외이지만, 그 외의 경우에는 D isplay System 설정 요소가 반드시 존재해야 한다. 여기에서 설정하는 속성은 특정 윈도우 시스템에 의해서 사용되는데, 디스플레이 매니저에서는 ”number of pi pes"와 “X11 displays", 이 2가지 속성을 설정할 수 있다.

2) 사용자(User)

VR Juggler에서는 사용자의 특성을 설정할 수 있게 돼있다. 대표적인 사용자 특성으로는 사용자의 눈 사이의 거리(interocular distance)를 들 수 있다. 또, 사용자의 head에 대한 디바이스 프록시의 위치도 설정할 수 있다. VR Juggler의 설정에는 반드시 하나 이상의 사용자가 필요한데, 이 때 사용자는 디스플레이 윈도우 내의 뷰포트(viewport)를 정의하게 된다. VR Juggler에서는 사용자를 여러 개 설정해서 동시에 여러 개의 디스플레이 윈도우를 제어하는 것도 가능하다.

3) 디스플레이 윈도우

디스플레이 윈도우는 씬을 렌더링하는데 사용되는 속성이다. VR Juggler에서는 모든 디스플레이 윈도우가 OpenGL 그래픽스를 디스플레이할 수 있으며, OpenG L을 다루는 윈도우를 열어서 관리하는데 필요한 platform-specific API를 사용해서 씬을 렌더링할 수 있다. platform-specific 속성에 필요한 기능은 디스플레이 윈도우의 추상화 개념 뒤에 숨어있기 때문에, 모든 플랫폼에 대해 동일한 설정 과정을 거칠 수 있는 것이다. 디스플레이 윈도우 요소에서 설정하는 속성으로는 origin, size, pipe, in stereo, full screen, always on top, active가 있다.

4) OpenGL 프레임 버퍼

모든 디스플레이 윈도우에는 고유의 OpenGL 프레임 버퍼 설정이 존재한다. 이 중첩 가능한 설정 요소는 platform-specific한 OpenGL 윈도우 API와 그 방식이 일치하는 면이 있기 때문에 프레임 버퍼 설정에 있어서 WGL, GLX, 혹은 AGL에 대한 지식이 유용할 수도 있다.

프레임 버퍼 설정은 그다지 힘든 일이 아니지만, 제대로 설정하려면 조금 신경을

써야만 한다. 속성에 사용되는 용어는 WGL, GLX 및 AGL에서 사용되는 용어와 비슷한 면이 많다.

color depth 속성과 depth buffer 설정은 기존 방식과는 반대로 최소 요구 사양으로 설정되는 경향이 있다. 예를 들어 red channel size가 1비트로 설정되면, 윈도우 시스템은 OpenGL 비주얼에 red 색상에 요구되는 비트수를 1 이상으로 요청한다. 따라서 윈도우 시스템은 red channel size를 1에서 8 비트 사이의 숫자 중에서 선택할 수 있으며, 가능하면 최대값으로 선택하게 된다. 만약 red channel size가 8로 설정되는 경우에는 윈도우 시스템은 red channel의 비트수로 8만 선택할 수 있다. 만약 이런 비주얼이 존재하지 않는다면, 윈도우는 열리지 않는다. 만약 red channel size가 0으로 설정돼 있는 경우에는 사용 가능한 비주얼 중 가장 작은 비트값이 선택된다. 이 방식은 모든 bit size 설정에서 동일하게 사용된다.

5) 그래픽스 윈도우 입력

VR Juggler 1.0에서는 디스플레이 윈도우는 키보드와 마우스로부터 입력을 받을 수 없게 돼 있었다. 따라서 별도의 윈도우(VR Juggler 1.0 용어로는 “keyboard window")를 키보드/마우스 입력의 용도로 사용했다. VR Juggler의 개발의 초기 단계에서는 이렇게 할만한 충분한 이유가 있었지만, 시간이 지나면서 사용성에 있어서 점점 더 큰 문제가 야기되기 시작했다. 이런 이유로 VR Juggler 1.1 이후 버전에서는 키보드와 마우스 입력을 그래픽스 윈도우에서 직접 받아들이는 것이 가능해졌다. 이렇게 하기 위해서는 키보드/마우스 입력 핸들러를 통해 키보드와 마우스 입력 이벤트를 받아들이게끔 하는 설정이 필요하다. 이 개념이 바로 Keyboard/Mouse Input Handler다. 기본적으로 디스플레이 윈도우는 키보드/마우스 입력 핸들러로부터 이벤트를 받지 못한다. 그래픽스 윈도우 입력 요소에는 4개의 속성이 존재하며, 그 속성들은 keyboard/mouse input handler name, allow mouse locking, lock key, start locked, sleep time, 이 4가지다.

6) 뷰포트(Viewport)

뷰포트에는 시뮬레이터 뷰포트(simulator viewport)와 서피스 뷰포트(surface viewport), 이 2가지가 있다. 이 2가지 뷰포트를 적절히 사용함으로써 여러 개의 뷰포트를 유지하는 것이 가능하다.

- 시뮬레이터 뷰포트
  - VR Juggler의 몰입 시스템 시뮬레이터 인터페이스를 렌더링하는데 사용

되는 뷰포트로, 기본적으로 head, wand 및 detached/movable camera의 이 3가지 구성 요소가 있으며, 실제 세계의 원점과 좌표계를 나타내는 좌표축도 렌더링된다. 시뮬레이터 뷰포트의 입력은 시뮬레이터 디바이스를 통해 이뤄지며, 뷰포트 자체의 설정은 간단한 설정을 통해 쉽게 할 수 있다.

- vertical field of view라는 속성은 field of view의 각도를 나타내는 (floa t-point) 값을 받아들여서 뷰를 설정한다. 이 속성의 기본값은 80.0이다.
- 시뮬레이터 뷰포트 설정에만 사용되는 독특한 속성으로 simulator plug0in을 들 수 있다. 이 속성은 VR Juggler 2.0 Alpha 2 버전에서 시뮬레이터 인터페이스가 플러그-인 구조를 통해 정의된 데서 비롯되었다. VR Juggler에는 2개의 기본적인 시뮬레이터 플러그-인이 존재한다. 하나는 Open GL Draw Manager에 대한 것이고, 다른 하나는 OPenGL Perfarmer Draw Manager에 대한 것이다. 시뮬레이터 뷰포트를 사용하려면 디폴트 시뮬레이터 플러그-인이 설정돼 있어야 하는데, 그 설정 속성으로는 camera position, wand position, draw projections, surface, color, head model, wand model을 들 수 있다.

- 서피스 뷰포트
  - 서피스 뷰포트는 CAVE나 immersive desk의 각 스크린이나 HDM의 스크린을 설정하는데 필요한 것이다. 이 뷰포트에서는 사용자의 눈과 카메라가 영구적으로 매달리는 형상디 a되며, 그렇기 때문에 렌더링된 뷰가 사용자의 위치에 따라 달라지게 된다. 뷰가 계산되는 방식은 서피스가 고정돼 있느냐, 혹은 움직이느냐에 따라 달라진다. 고정된 서피스는 일반적으로 CAVE wall의 경우이고, 뷰가 움직이는 경우는 HMD같이 트래커의 움직임이 심한 경우다. 예를 들어 사용자의 머리가 움직일 때마다 HMD의 눈도 함께 움직이게 되고, immersive desk의 서피스 역시 피봇 포인트(pi vot point)의 각도에 따라 달라질 수 있다. VR Juggler의 서피스 뷰포트는 이런 모든 경우에 다 대처할 수 있다.
  - 서피스 뷰포트의 설정 속성으로는 corners, is tracked, tracker proxy를 들 수 있다.
- 단일 윈도우상에서의 다중 뷰포트
  - 단일 윈도우상에 여러 개의 뷰포트를 설정하는 것도 가능하며, 디스플레이 윈도우 내에서 각 뷰포트의 위치와 속성을 설정하는 것만으로 여러개

의 뷰포트를 설정할 수 있다. 여기에 필요한 속성은 origin과 size, 이 2가지다.

라. 클러스터 컴퓨터의 설정

여러 대의 클러스터를 이용해서 디스플레이 장치를 구축한 경우에는 각각의 클러스터 노드에 대해 설정을 해야 한다. 이 때, 가장 기본이 되는 것은 각각의 클러스터 노드에 대한 설정과 클러스터 매니저의 설정이다.

클러스터 노드에서는 각 노드의 디스플레이 시스템에 대한 정보와 VRJuggler에서 클러스터 연결에 사용할 포트번호, 호스트 네임과 같은 정보를 설정함으로써 클러스터 노드를 서로 연결해서 사용할 수 있다. 클러스터 매니저에서는 클러스터에 대해 사용할 여러 클러스터 플러그-인에 대한 정보를 설정한다. 클러스터 설정시 많이 사용하는 플러그-인은 다음과 같다.

- Remote Input Manager
  - 플러그-인: RIMPlugin
  - 입력 디바이스로부터 들어오는 데이터를 클러스터 노드간에 공유하는데 필요한 플러그-인
  - 별도로 설정할 필요는 없다.
  - 사용자의 움직임이 트래킹 가능한 몰입형 가시화 시스템의 경우에는 헤드 트래커로부터 입력되는 정보가 씬의 형성에 매우 중요한 영향을 미친다. 따라서 그런 입력 정보를 클러스터 노드간에 공유하고 있어야 각각의 렌더링 노드에서 적절한 씬을 렌더링할 수 있다.
  - VRJuggler의 클러스터 설정 부분에서는 Remote Input Manager 플러그-인을 통해서 이 정보를 공유하며, 이 플러그-인이 없으면 입력된 디바이스 데이터는 디바이스가 실제로 연결된 노드에서만 사용할 수 있다.
  - 애플리케이션이 디바이스로부터 입력을 받아들이려면 이 플러그-인이 반드시 로딩돼야 하는데, 이 플러그-인에는 별도의 configuration element가 없으므로 cluster manager에 이 플러그-인을 추가하기만 하면 RIMPlugin에 대한 설정은 끝난다.
- Start Barrier
  - 플러그-인: StartBarrierPlugin

- 모든 노드가 동일한 입력 데이터로부터 시작될 것을 보장하는 플러그-인
- start barrier "master"와 다른 클러스터 노드로부터의 입력 커넥션을 위해 네트워크 포트를 설정해야 한다.
- RIM 플러그-인이 로딩되면 반드시 start barrier 플러그-인도 로딩돼야 한다. 이 플러그-인은 클러스터에 묶인 모든 노드가 동일한 프레임에 실행을 시작할 것을 보장한다. 즉, cluster manager에서 설정된 모든 노드들이 연결되기 전까지는 각 프레임 루프의 실행을 “master" 노드가 차단하는 역할을 하는 것이다.
- Start barrier에 사용할 포트는 클러스터 노드의 일반 통신에 사용할 포트와는 다른 것을 선택해야 한다.

- Application Data Manager
- TCP Swap Lock
- Parallel Port(Wired) Swap Lock

마. Sonix Sound Manager

Sonix Sound Manager를 ld용하면 Juggler의 Sonix 모듈을 손쉽게 이용해서 사운드를 제어할 수 있다. Sonix는 VRJuggler 애플리케이션 오브젝트에서 코딩을 통해 직접 제어할 수도 있지만, Sonix Sound Manager를 이용하면 Sonix를 사용할 때마다 코드를 작성해야 하는 번거로움을 피할 수 있다. 예를 들어, Sonix S Ound Manager를 이용하면 오디오 API의 선택, 오디오 “프레임”의 업데이트, Sonix에서 사용하는 사운드 오브젝트 등록 등의 일을 처리할 수 있다. 이런 일을 VRJuggler 애플리케이션에서 직접 처리하는 것도 가능하긴 하지만, 이렇게 되면 애플리케이션 오브젝트 코드의 복잡도만 증가될 뿐이다. 결국 Sonix Sound Manager를 이용하는 가장 큰 장점은 VRJuggler 애플리케이션 오브젝트를 작성하는 프로그래머가 (상대적으로 간단하게) Sonix 사운드 핸들 오브젝트를 사용할 수 있게 하는 데 있다.

Sonix Sound Manager를 설정하는 데는 다음 4가지 요소가 필요하다.

- 사운드 사용할 API
- 사운드를 듣게 될 사용자의 3차원 위치 좌표
- 오디오 파일을 찾는데 필요한 검색 경로

- VRJuggler 애플리케이션 오브젝트가 접근할 사운드 오브젝트 목록

이 4가지 요소는 Sonix Sound Manager의 설정 속성(config element)과도 일치한다. 각 속성은 다음과 같다.

1) sound API

이 속성은 Sonix에서 오디오 데이터를 렌더링하는데 사용할 오디오 백엔드(back end)를 나타낸다. 현재 Sonix에서는 stub, OpenAl, Audiere, AudioWorks 및 Subsynth, 이 5가지 오디오 백엔드를 사용할 수 있다. 오디오 백엔드는 런타임에 Sonix가 로딩할 수 있는 컴포넌트를 가리키며, 다시 말해서 Sonix가 사용하는 플러그-인이라 할 수 있다. 어떤 플러그-인을 사용할 지는 플러그-인의 호환성이나 유효성, 그리고 런타임 환경에 따라 달라지게 된다. 다음은 Sonix에서 사용하는 플러그-인에 대한 설명이다.

- OpenAL: OpenAL은 크로스-플랫폼 사운드 프로그래밍 인터페이스를 제공하는 API로, OpenAL 1.0 스펙에 의하면 마이크로소프트 윈도우즈, 리눅스, BSD, MacOS X, 그리고 IRIX를 지원한다. 하지만 IRIX에서 OpenAL을 사용하면 POSIX 쓰레드를 사용하므로, VRJuggler Portable Runtime의 SPROC 서브시스템과의 호환성에 문제가 발생할 수 있다.
- Audiere: 다양한 오디오 파일 포맷을 재생할 수 있는 간단한 크로스-플랫폼 프로그래밍 인터페이스로 C나 C++ 프로그래머들에게 유용한 툴이다. 마이크로소프트 윈도우즈, 리눅스, 그리고 IRIX를 지원한다. Audiere가 마지막으로 업데이트된 것은 2003년 9월(버전 1.9.3)로, 이 프로젝트가 지속될 지는 불분명하다. OPenAL만큼 이식성이 높지는 않지만 플랫폼간에 일관성 있는 오디오 플레이백을 제공한다는 면에서는 OpenAL보다 신뢰도가 높으며, OpenAL과 마찬가지로 POSIX 쓰레드를 사용하기 때문에 VRJuggler의 SPROC 버전과는 호환되지 않는다.
- AudioWorks: AudioWorks2는 3D 오디오 플레이백에 사용되는 프로그래밍 인터페이스로, Sonix AudioWorks 백엔드는 IRIS에서만 테스트가 완료된 상태다. OpenAL이나 Audiere와는 반대로 VRJuggler의 SPROC 버전과만 호환된다. 이 플러그-인은 계속 Multigen-Paradigm사의 상품으로 유지될 수 있을지 불투명하다.
- Subsynth: Subsynth는 Sonix를 만든 Kevin Meiner과 석사 과정중에 수행한

프로젝트다. 로우-레벨 오디오 툴로써 Subsynth의 역할은 OpenAL이나 Audiere와는 엄연히 다르며, Subsynth 플러그-인 프로젝트 역시 종료된 지 오래다. 이런 이유로 Sonix에서 Subsynth 백엔드를 사용하는 것은 그다지 추천할 만한 사항은 아니다.

- Stub: Stub은 엄격히 따지면 플러그-인이 아니라 다른 오디오 API가 없을 경우에 대비한 빌트-인 컴포넌트다. 이 컴포넌트는 사운드를 재생할 수 없는 경우에도 애플리케이션이 Sonix에 접근할 수 있게 하므로 서로 다른 VR 시스템에서 코드가 이식성을 갖는지 여부를 판정하는데 유용하다. 물론 애플리케이션의 기능을 유지하려면 사운드 기능이 필요하긴 하지만, 사운드 기능을 사용할 수 없을 경우에도 stub 플러그-인을 사용하면 코드를 따로 수정하지 않고도 처리할 수 있다. 일반적으로 stub 플러그-인은 사운드 API로는 사용되지 않는다.

2) listener position

가상 현실의 world coordinate 공간 상에서의 사용자의 위치를 3차원 좌표(x, y, z)로 나타낸 것으로, 일반적으로 VRJuggler origin 값으로 설정한다.

3) file search path

sound object 속성에서 설정한 오디오 파일을 검색할 디렉토리로 0개 이상의 디렉토리를 설정할 수 있다. 현재는 Sonix Sound Manager에서 이 속성을 사용하지 않는 상태이지만, 향후에는 이 속성을 반영할 예정이다.

4) sound object

런타임에 Sonix에 등록될 사운드 오브젝트 목록을 가리킨다. Sonix Sound Manager는 0개 이상의 사운드 오브젝트를 포함하며, VRJuggler 애플리케이션 오브젝트를 작성하는 사람은 Sonix 사운드 핸들을 통해 사운드 오브젝트에 접근할 수 있다. Sonix Sound Manager는 사운드 데이터를 메모리에 로딩하는 작업과 사운드 오브젝트를 설정하는 작업의 세부사항을 처리하므로, 애플리케이션 작성자는 Sonix 사운드 핸들을 선언하고 초기화하기만 하면 사운드를 사용할 수 있다.

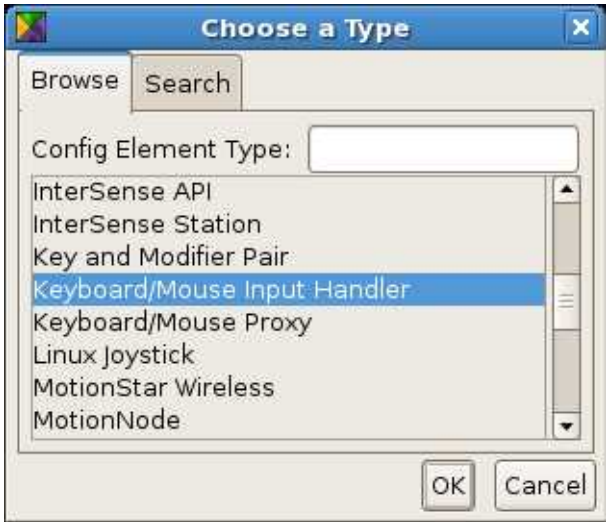
사운드 오브젝트의 이름은 VRJuggler 애플리케이션 오브젝트에 의해 Sonix로부터 요청되는 오브젝트에 대한 이름이므로 Sonix Sound Manager에 주어지는 각각의 sound object config element에는 서로 구별할 수 있는 이름이 주어져야 한다. 사운드 오브젝트에서 설정할 수 있는 속성은 다음과 같다.

- filename: 로딩할 사운드 파일의 경로
- ambient: 사운드의 ambient 여부를 설정하는 boolean flag
- re-triggerable: 사운드가 재생되는 동안 다시 시작할지 여부를 나타내는 boolean flag
- streaming: 사운드가 스트리밍되고 있는지 여부를 나타내는 boolean flag으로, Audiere의 경우, 루핑되는 사운드는 반드시 이 속성이 true로 설정돼 있어야 한다. Audiere에서 이 속성이 true가 아닌 경우에는 한 번만 재생되는 사운드 효과로 처리된다.
- loop: 사운드가 재생될 때 몇 번이나 재생될 지를 나타내는 정수값. -1을 끝없이 계속 반복되는 사운드를 나타낸다. Audiere의 경우에는 사운드가 반복재생되려면 streaming 값도 true로 설정돼 있어야 한다. 기본값은 1이다.
- pitch bend: 사운드의 pitch를 변경할 수 있는 값으로, [0, n) 범위의 값을 주로 사용한다. 1.0은 normal pitch를, 2.0은 double pitch를, 그리고 0.5는 half pitch를 나타낸다. 기본값은 1.0이다.
- cutoff: 사운드에 대한 로우-패스 필터 컷오프(low-pass filter cutoff)를 나타내는 소수값. 이 값은 [0.0, 1.0] 범위 내의 값이어야 하며, 기본값은 1.0이다.
- volume: 사운드의 볼륨을 나타내는 소수값으로 [0.0, 1.0] 범위의 값이다. 기본값은 1.0(full volume)이다.
- position: 가상 현실의 world coordinate상에서 사운드의 위치를 나타내는 위치값으로 (x, y, z)로 나타낸다.

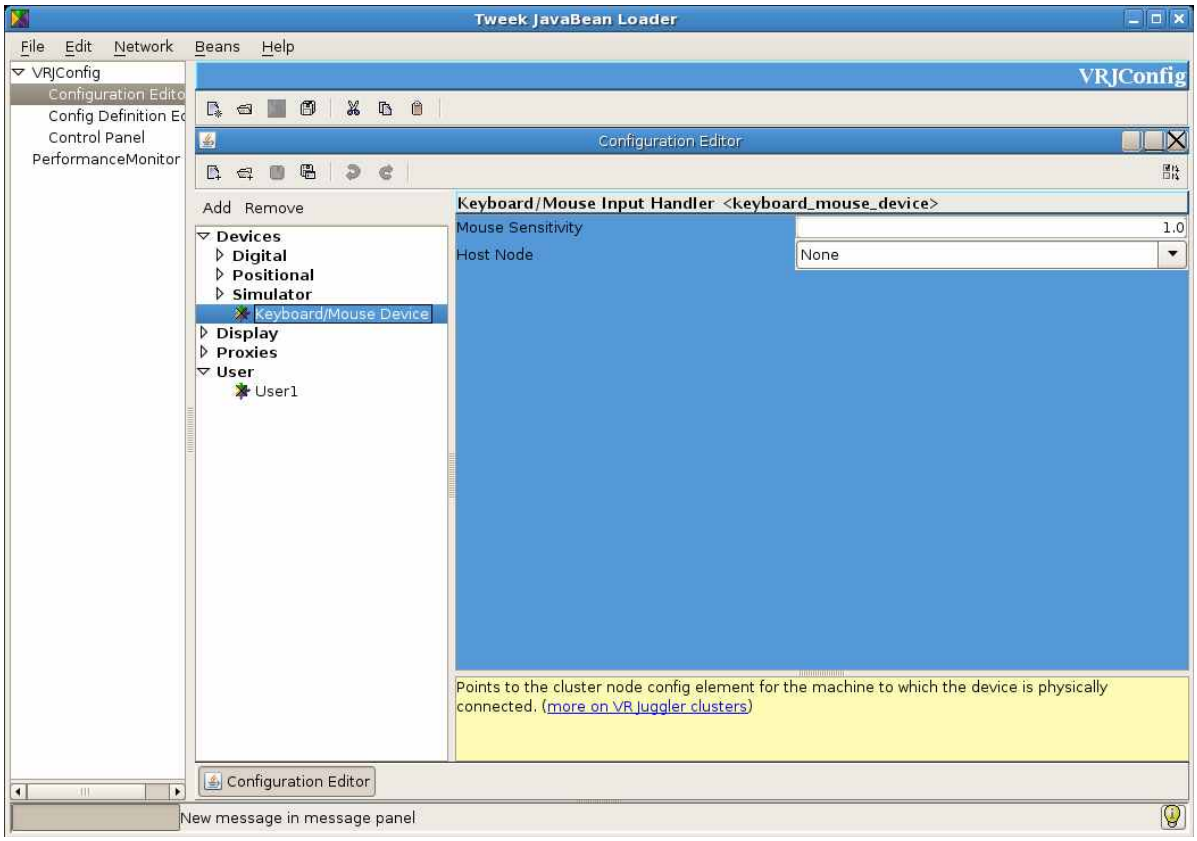
바. VRJuggler 설정 방법

간단하게 독립 적으로 실행 가능한 VRJ 설정 파일을 만들어 보자.

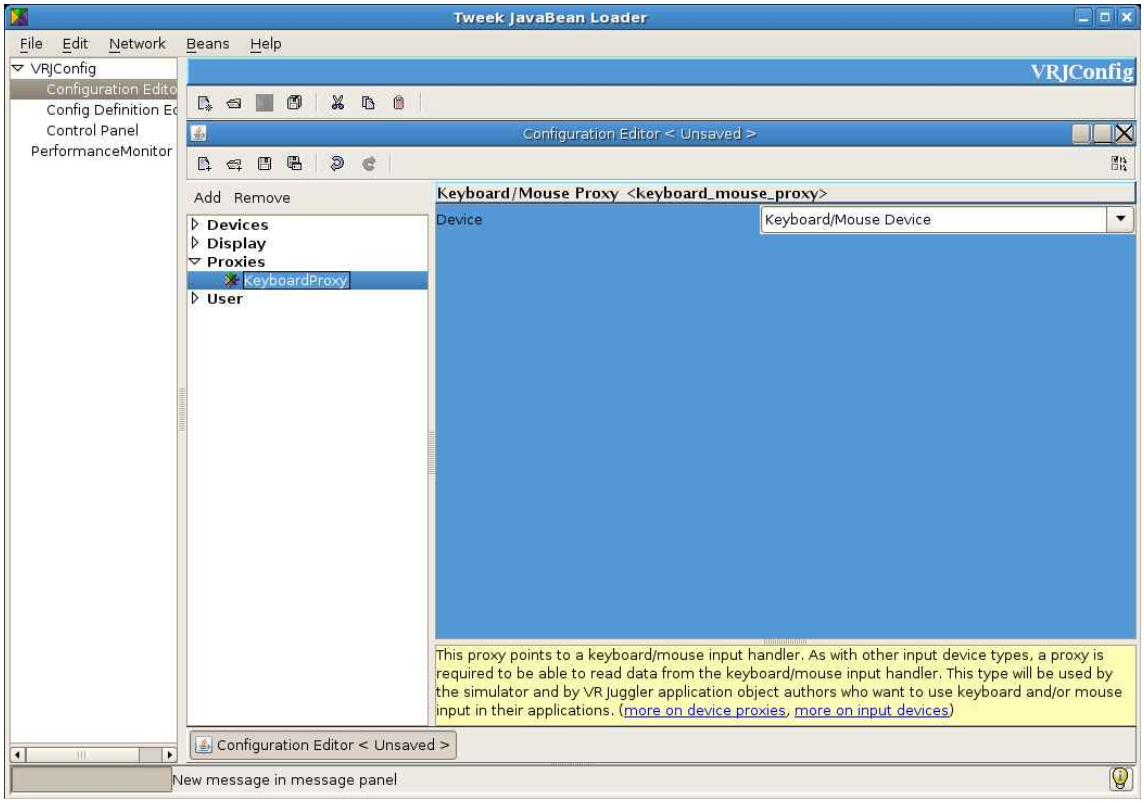
1. VRJConfig 윈도우를 실행한다.
2. 우선 가장 기본적인 Keyboard/Mouse Device를 추가한다.
   - VRJConfig 윈도우 패널 상단의 Add 버튼을 누르고 Keyboard/Mouse Input Handler를 선택해서 추가한다. ([그림 3-3], [그림 3-4])



[그림 3-3] Keyboard/Mouse Input Handler의 추가

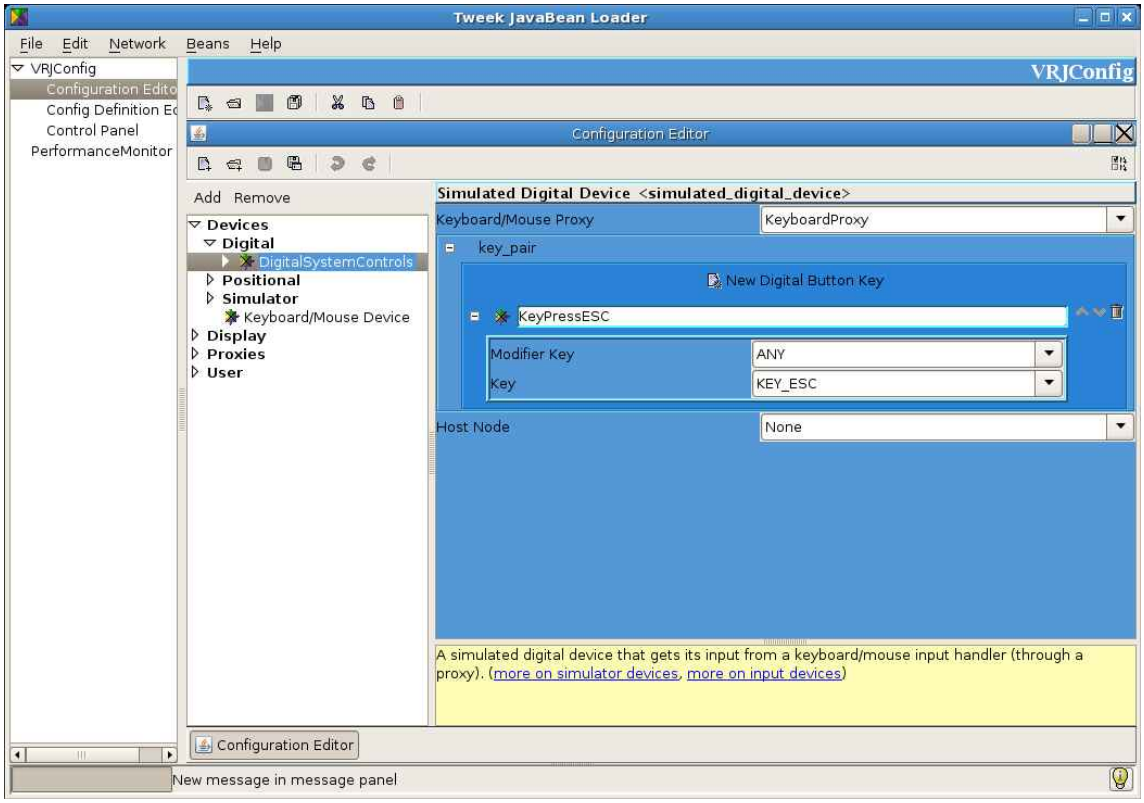


[그림 3-4] Keyboard/Mouse Input Handler 추가 뒤의 모습

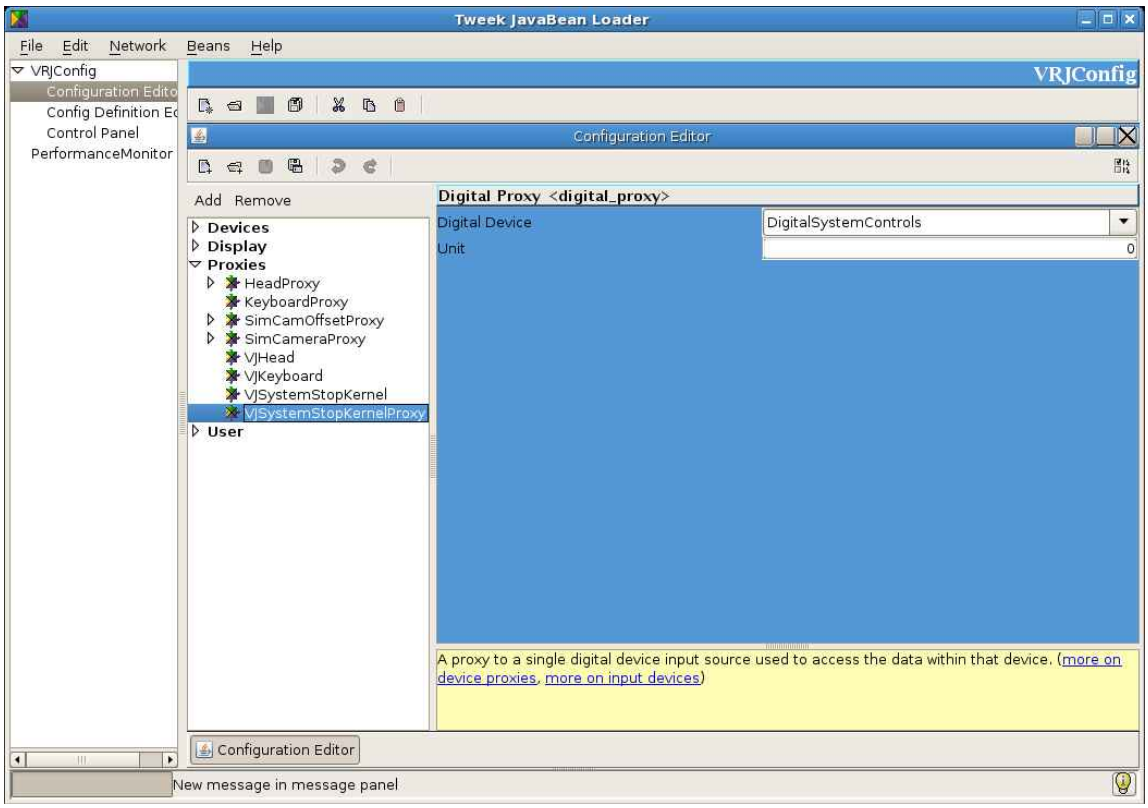


[그림 3-5] Keyboard/Mouse Proxy의 추가

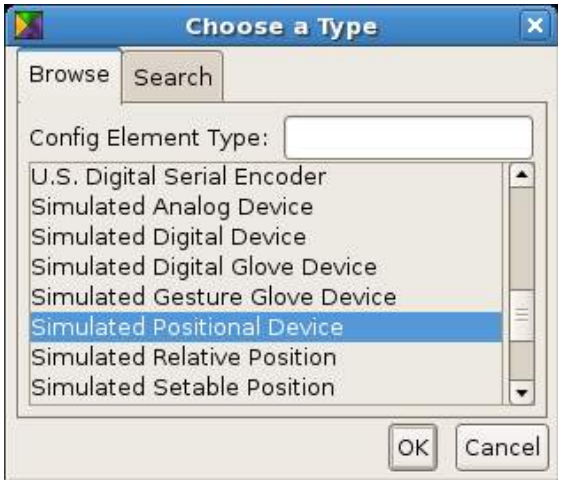
3. Keyboard/Mouse Input Handler에 대한 Proxy를 설정한다.
   - VRJConfig 윈도우 패널 상단의 Add 버튼을 누르고 Keyboard/Mouse Proxy를 선택해서 추가한다. ([그림 3-5])
   - 디바이스 속성을 위에서 추가한 Keyboard/Mouse Device로 설정하고, 이름을 Keyboard Proxy로 변경한다.
4. 이제 Viewing을 제어할 디바이스 장치들(ex. Head, Camera)을 구체적으로 설정할 차례다. 우선, 키보드로 Exit을 실행할 수 있는 Digital Device를 추가한다. ([그림 3-6], [그림 3-7])
   - VRJConfig 윈도우 패널 상단의 Add 버튼을 누르고 Simulated Digital Device를 선택해서 추가한다.
   - Keyboard/Mouse Proxy는 위에서 생성한 Keyboard Proxy를 선택한다.
   - key_pair를 하나 생성하고 이름을 설정한 뒤(여기에서는 KeyPressESC), Modifier Key는 Any, Key는 KEY_ESC로 설정한다.
   - VRJConfig 윈도우 왼쪽 패널 상단의 Add 버튼을 누르고 Digital Proxy를 추가한다.



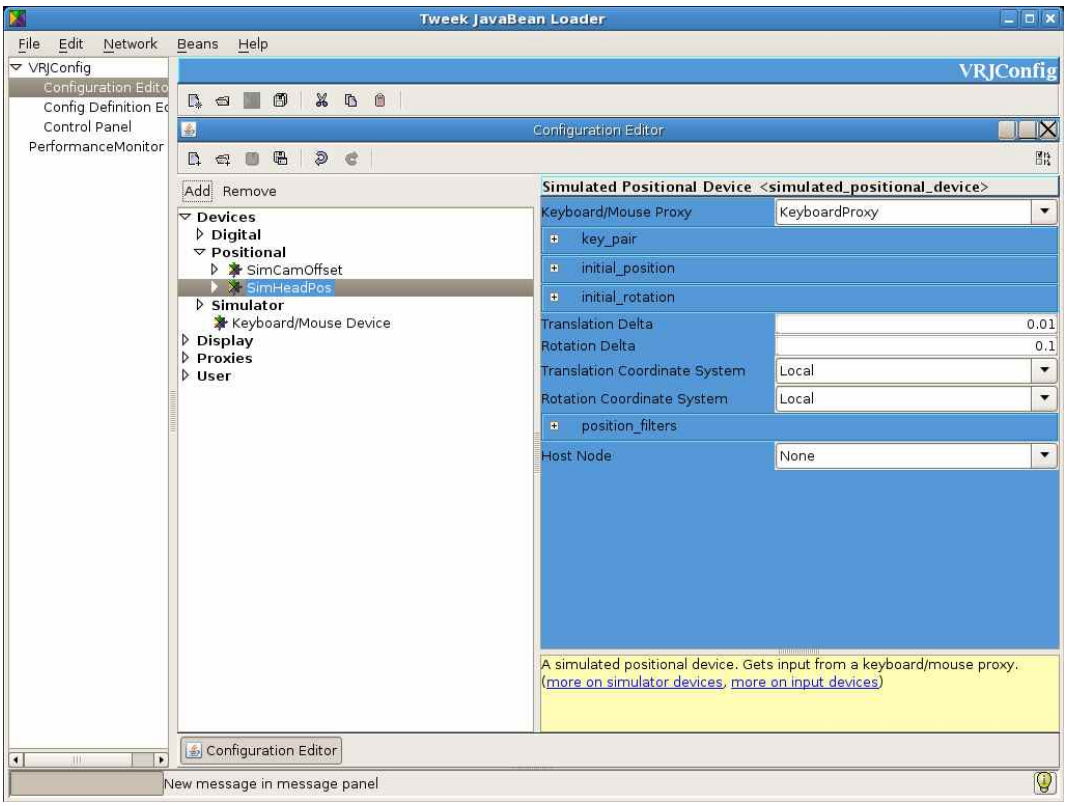
[그림 3-6] Exit에 사용할 Digital Device 추가



[그림 3-7] Exit에 사용할 Digital Proxy 추가



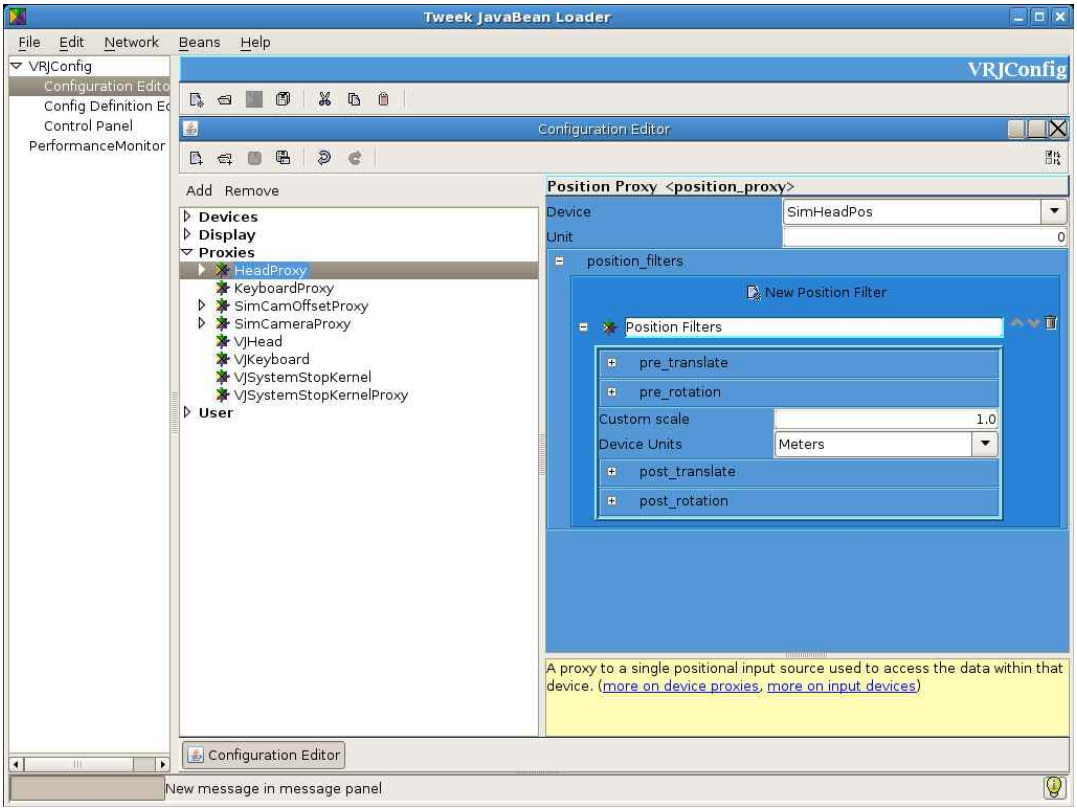
[그림 3-8] Head 설정에 사용할 Positional Device의 추가



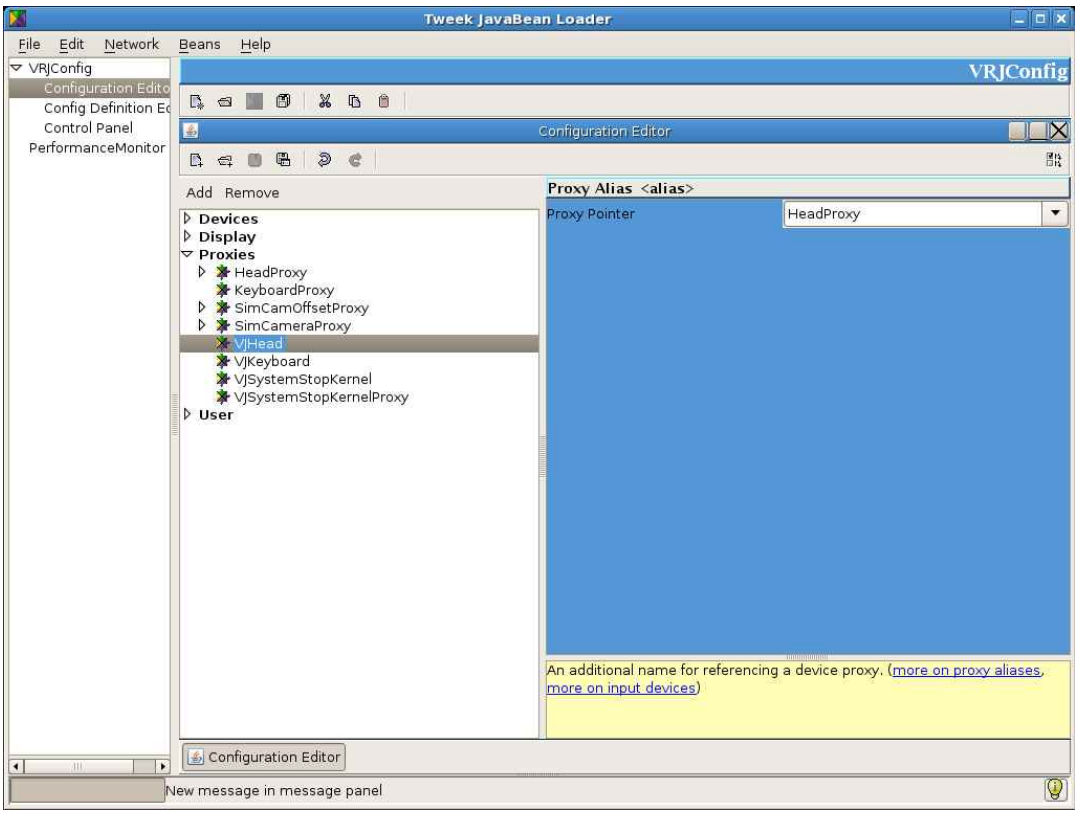
[그림 3-9] Head 설정에 사용할 Positional Device의 추가

- 이 Digital Proxy의 이름을 VJSystemStopKernelProxy로 변경하고, Digital Device를 앞에서 추가한 DigitalSystemControls로 선택, 설정한다.

5. 다음으로, Head의 위치를 제어할 Positional Device와 Proxy를 생성한다. ([그림 3-8], [그림 3-9], [그림 3-10], [그림 3-11])



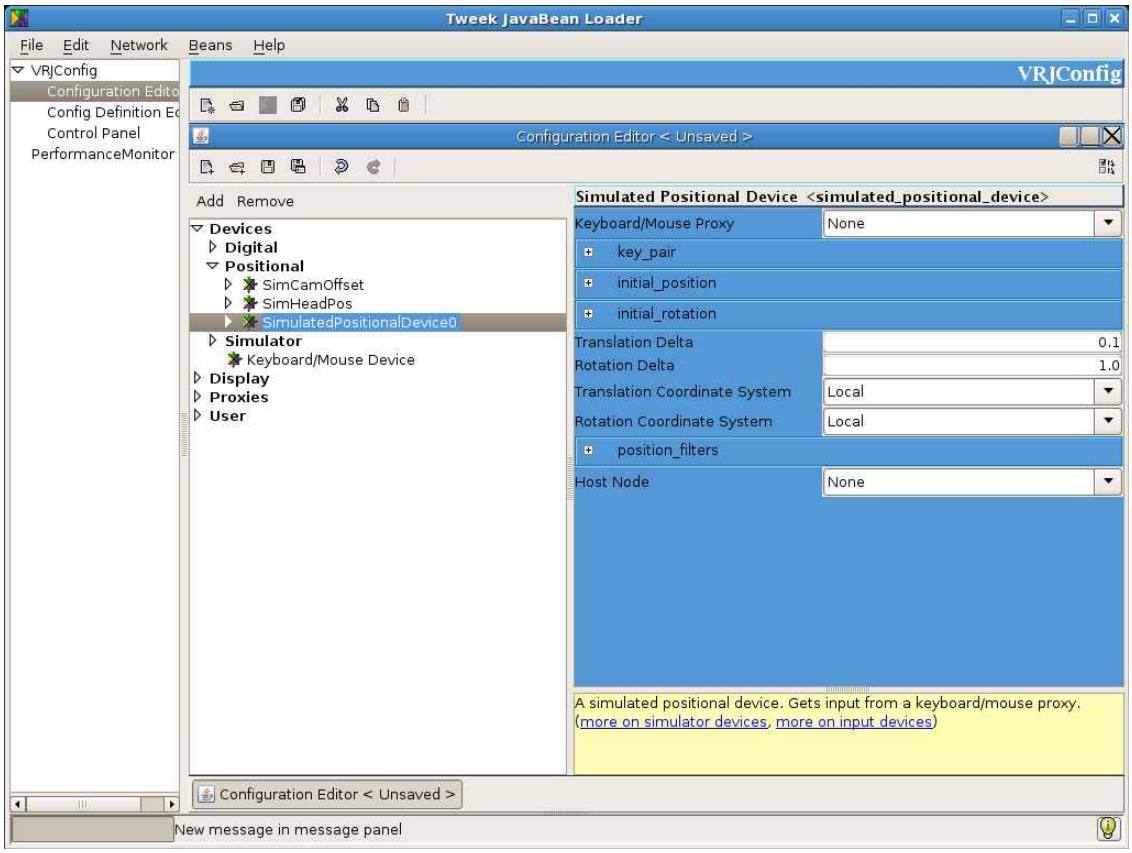
[그림 3-10] Head 설정에 사용할 Position Proxy의 추가



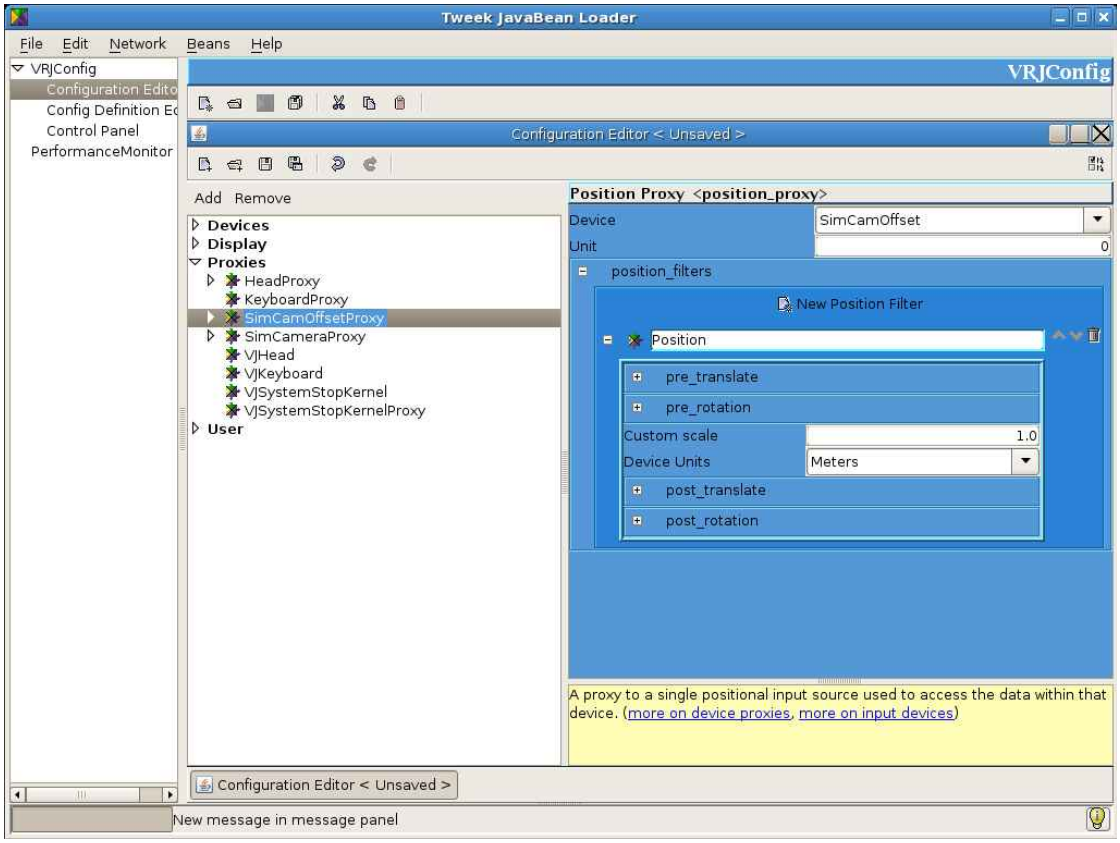
[그림 3-11] Head 설정에 사용할 Proxy Alias의 설정

- VRJConfig 윈도우 패널 상단의 Add 버튼을 누르고 Simulated Positional Device를 생성한 뒤, 이름을 SimHeadPos로 변경한다.
- 디바이스 속성에서 Keyboard/Mouse Proxy를 위에서 생성한 Keyboard/Mouse Proxy로 설정하고, 그 외 다른 속성을 설정한다.
- VRJConfig 윈도우 왼쪽 패널 상단의 Add 버튼을 눌러서 Position Proxy를 하나 생성한 뒤, 이름을 HeadProxy로 변경한다.
- Device 속성을 앞에서 생성한 SimHeadPos로 설정하고, 그 외 다른 속성을 설정한다.
- Proxy Alias를 하나 생성해서 이름을 VJHead로 설정한 뒤, Proxy Pointer를 앞에서 설정한 HeadProxy로 설정한다.

6. 이번에는 Camera의 위치를 제어할 Positional Device와 Proxy를 생성할 차례다. ([그림 3-12], [그림 3-13])
   - VRJConfig 윈도우 패널 상단의 Add 버튼을 누르고 Simulated Positional Device를 생성한 뒤, 이름을 SimCamOffset으로 변경한다.
   - 디바이스 속성에서 Keyboard/Mouse Proxy를 앞에서 생성한 KeyboardPro



[그림 3-12] Camera 설정에 사용할 Positional Device의 추가

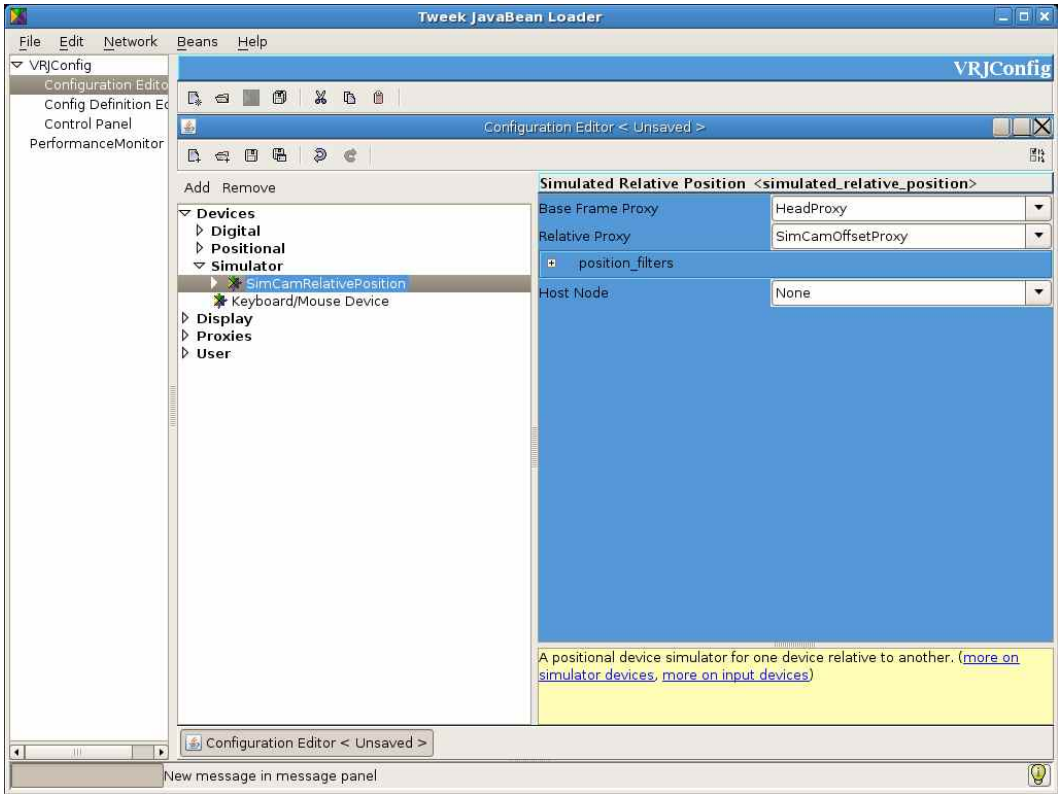


[그림 3-13] Camera 설정에 사용할 Position Proxy의 추가

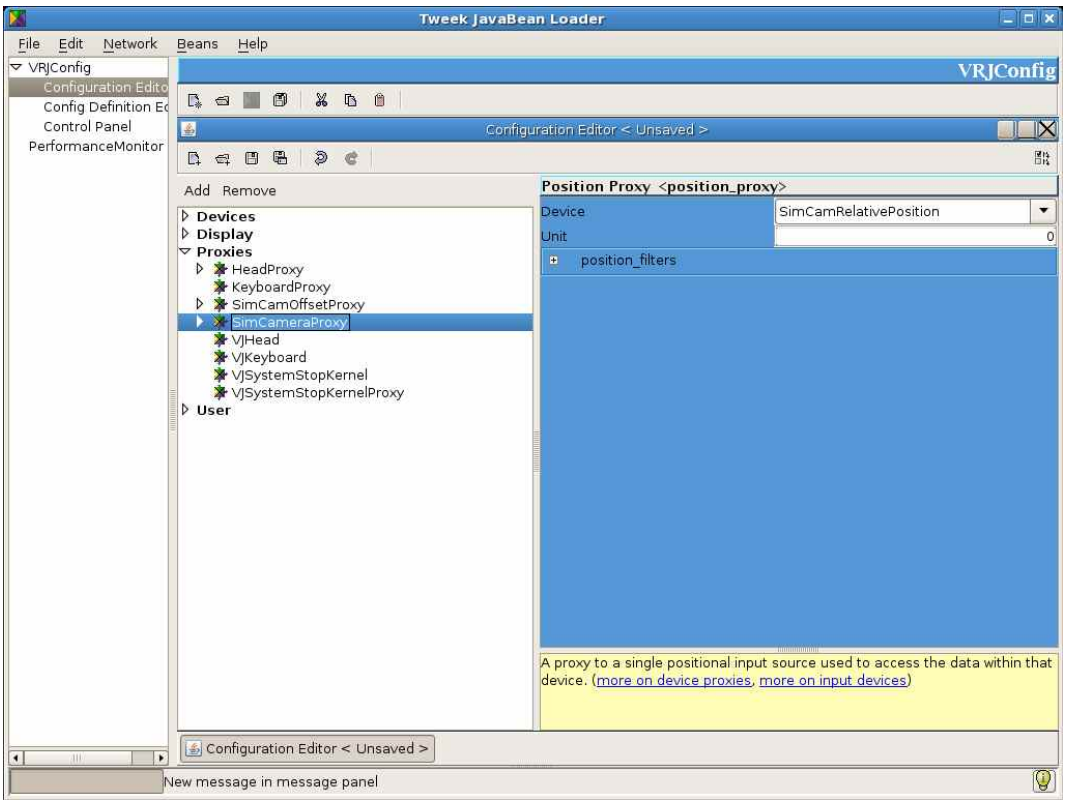
xy로 설정하고, 그 key_pair 등의 다른 속성을 설정한다. 특히 key_pair는 실제로 카메라 움직임을 제어할 키 속성을 설정해야 한다.

- VRJConfig 윈도우 왼쪽 패널 상단의 Add 버튼을 누르고 Position Proxy를 하나 생성한 뒤, 이름을 SimCamOffsetProxy로 변경한다.
- Device 속성을 앞에서 생성한 SimCamOffset으로 설정하고, 그 외 다른 속성을 설정한다.

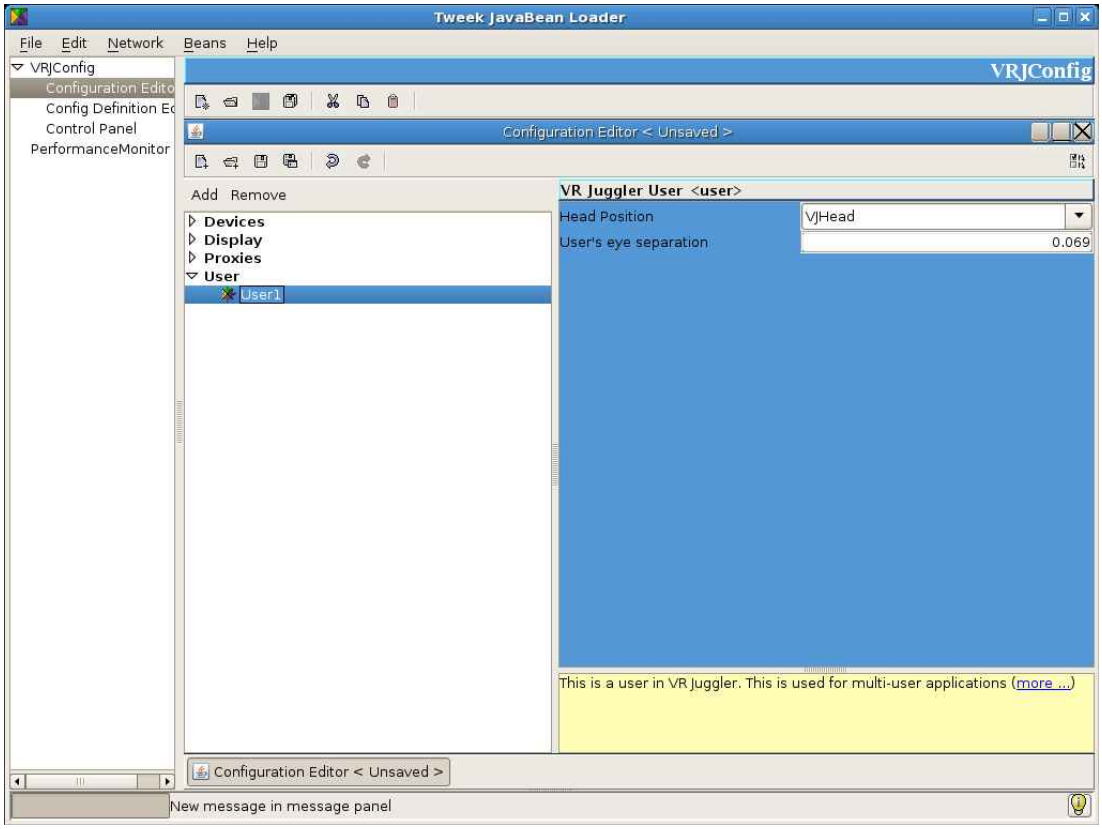
7. 다음으로, Simulator Device와 Proxy를 생성한다. ([그림 3-14], [그림 3-15])
   - VRJConfig 윈도우 패널 상단의 Add 버튼을 누르고 Simulated Relative Position Device를 생성한 뒤, 이름을 SimCamRelativeOffset으로 변경한다.
   - Base Frame Proxy를 앞서 설정한 Head Proxy로, Relative Proxy를 SimCamOffsetProxy로 설정한다.
   - 이 Simulated Relative Position Device는 Head와 Camera의 움직임을 연동시켜준다.



[그림 3-14] Simulator Device의 추가



[그림 3-15] Simulator Device에 대한 Proxy 추가



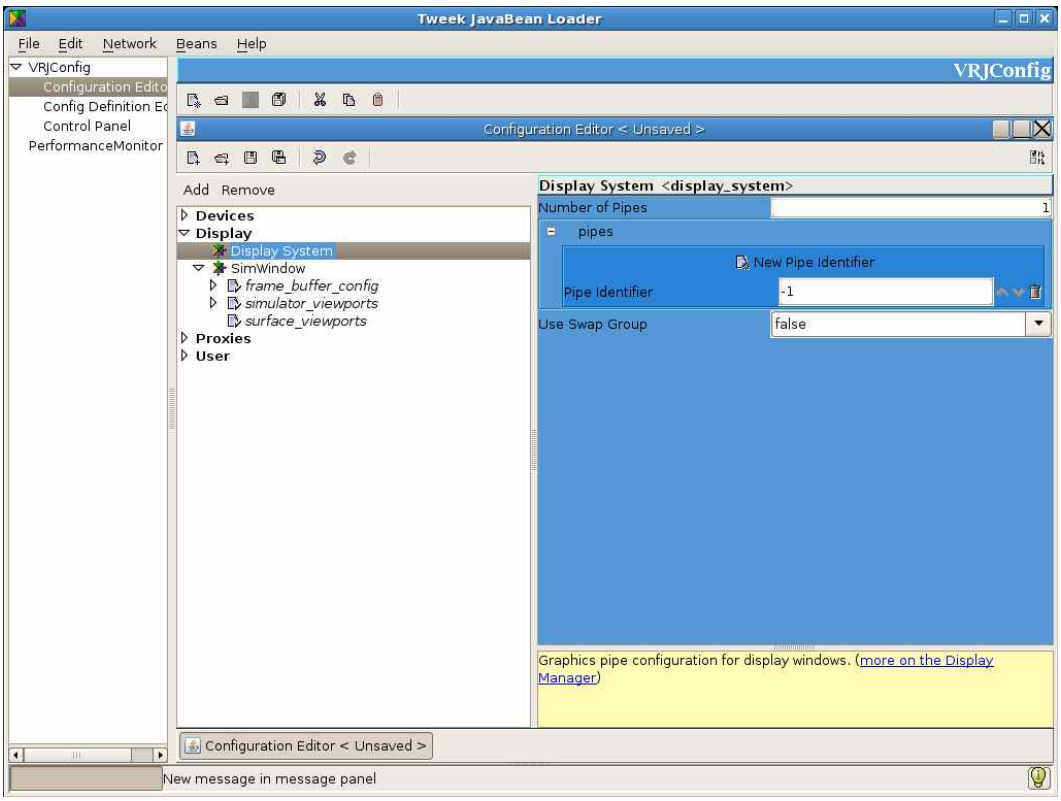
[그림 3-16] User 추가

- VRJConfig 윈도우 왼쪽 패널 상단의 Add 버튼을 눌러서 Position Proxy를 하나 생성한 뒤, 이름을 SimCameraProxy로 변경한다.
- SimCameraProxy에서 Device 속성을 SimCamRelativePosition으로 설정한다.

8. User를 추가한다. ([그림 3-16])
   - VRJConfig 윈도우 패널 상단의 Add 버튼을 누르고 VRJuggler User를 하나 생성한 뒤, 이름을 VjUser로 변경한다.
   - User는 사용자의 뷰포인트(viewpoint)를 잡아서 디스플레이 윈도우와 연동하는 역할을 수행한다.
   - Head Position 속성을 앞서 생성한 VjHead로 설정한다.

9. Display System을 생성한다. ([그림 3-17])

- VRJConfig 윈도우 패널 상단의 Add 버튼을 누르고 Display System을 하나 생성한다.
- Display System에서는 디스플레이 윈도우에서 사용할 그래픽스 파이프라인을 설정한다.

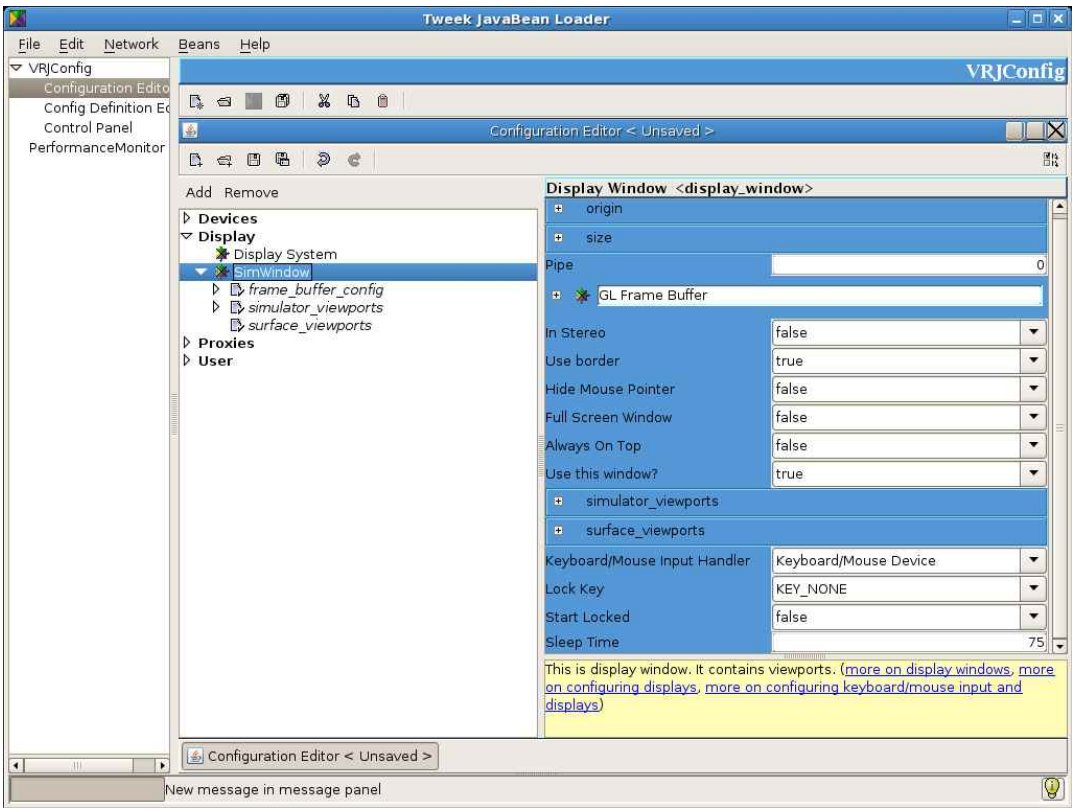


[그림 3-17] Display System의 추가

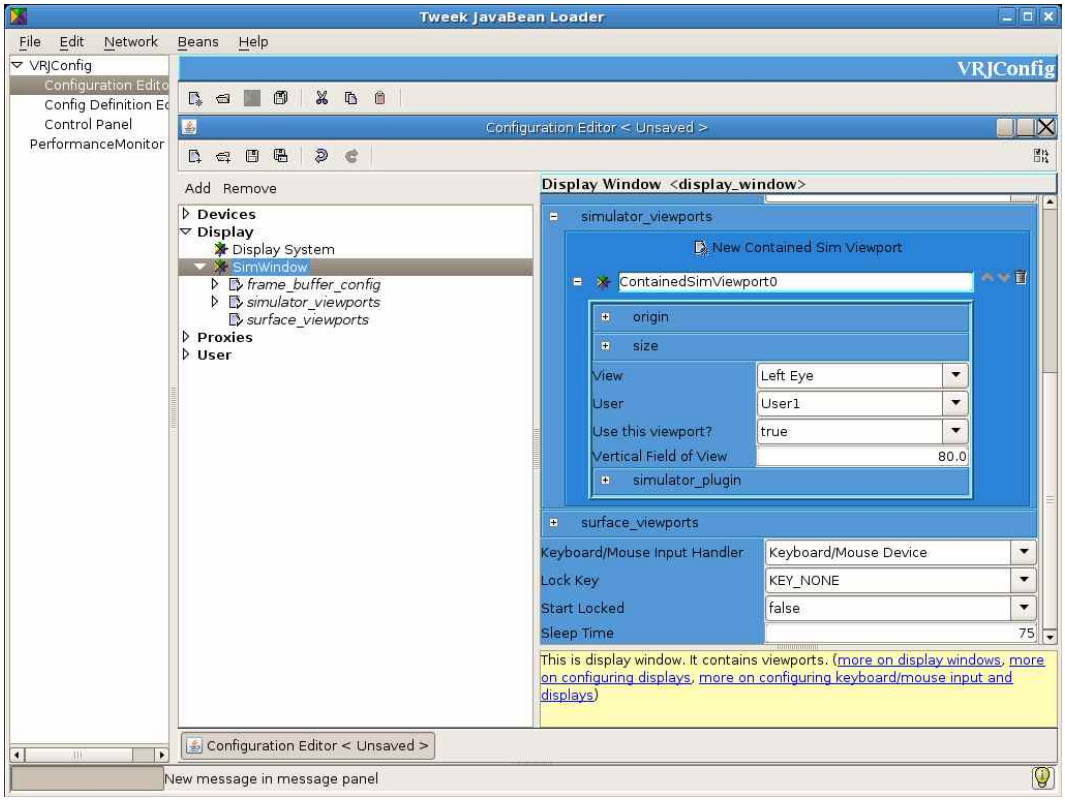
10. Display Window를 생성하고 설정한다. ([그림 3-18], [그림 3-19], [그림 3-20])

- VRJConfig 윈도우 패널 상단의 Add 버튼을 누르고 Display Window를 생성한 뒤, SimWindow로 이름을 변경한다.
- 해당 속성을 설정하고, Keyboard/Mouse Input Handler를 앞서 생성한 Keyboard/Mouse Device로 설정한다.
- Viewport를 생성한다. 본 예제에서는 Simulator Viewport를 생성해서 속성값을 설정하기로 한다.
- Simulator Viewport를 하나 생성해서 origin, size, View, User, Vertical Field of View 등의 속성을 생성하고, Simulator Plugin을 생성한다.
- Simulator Plugin에서 Camera Position을 앞서 설정한 SimCameraProxy로 설정해서 연결하고, 그 외 Wand Position, Draw Projection 등의 속성을 설정한다.

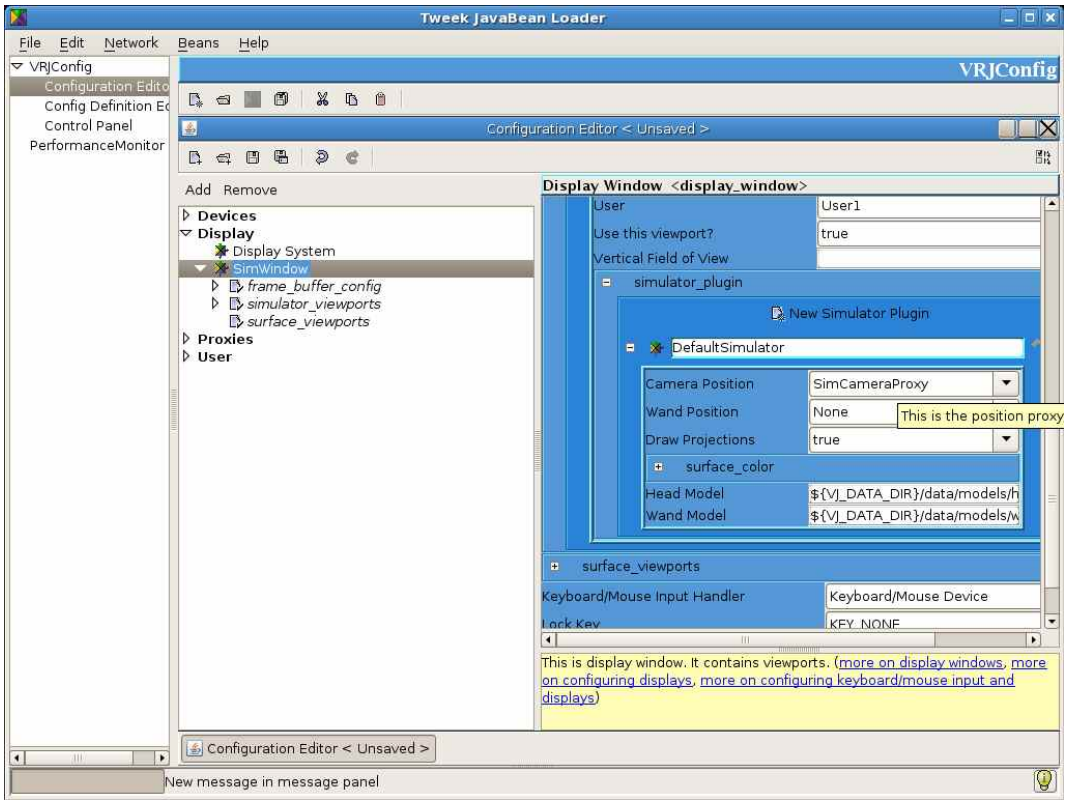
이와 같은 과정을 거쳐 VRJuggler 설정 파일을 생성하면, 커널에서 간단한 키보드/마우스 조작으로 제어 가능한 가상현실 환경을 구축할 수 있다.



[그림 3-18] Display Window의 생성



[그림 3-19] Simulator Viewport의 설정



[그림 3-20] Simulator Plugin의 설정

4. VRJuggler와 VTK의 좌표계 변환

가. VRJuggler의 행렬 변환

VRJuggler에서는 gmtl의 Vec<S, T> 클래스나 gmtl::Matrix44f 클래스를 통해 오브젝트의 위치를 나타내고 설정할 수 있다.

gmtl::Mtrix44f 클래스의 특성은 다음과 같다.

- single-precision floating-point값으로 구성된 4 x 4 행렬
- column-major 순서로 구성된 행렬
- 기본적인 행렬 연산을 제공한다. (행렬 할당, 일치 비교, 전치행렬, 역행렬, 더하기, 빼기, 곱하기, 스칼라 값으로 나누기, 기준 행렬 만들기, 0행렬 만들기, XYZ, ZYX, 혹은 ZYX 오일러 변환 행렬 만들기, 등)
- gmtl:Matrix를 이용해서 translation 변환 행렬을 생성하는 방법은 [소스 4-1]에, 특정 좌표를 바로 transform 위치로 옮겨가게 하는 방법은 [소스 4-2]에 나와 있다.

```
gmtl::Matrix44f mat;
2       gmtl::Vec3f trans(4.0, -4.231, 1.0);
3       mat = gmtl::makeTrans<gmtl::Matrix44f>(trans);
```

[소스 4-1] gmtl::Matrix44f를 이용한 translation 행렬 생성

```
 #transform 위치로 바로 옮겨가게 하려면 다음과 같이 사용한다.
2       gmtl::Vec3f trans(4.0, -4.231, 1.0);
3       mat = gmtl::setTrans(mat, trans);
```

[소스 4-2] transform 위치로 바로 옮겨가게 하는 코드

```
1       gmtl::Vec3f scale(1.5, 1.5, 1.5);
2       mat = gmtl::makeScale<Matrix44f>(scale);
```

[소스 4-3] scale 변환을 수행하는 행렬 생성

- [소스 4-3]은 scale을 수행하는 행렬을 생성하는 방법이다.

```
1      # translation 정보
2         gmtl::Vec3f trans;
3       gmtl::setTrans(trans, mat);
4      # z축에 대한 rotation 정보
5          float z_rot = (gmtl::makeRot<gmtl::EulerAngleXYZf>(mat))[2];  /
/radian으로 return됨.
6      # x축에 대한 rotation 정보
7          float x_rot = (gmtl::makeRot<gmtl::EulerAngleXYZf>(mat))[0];  /
/radian으로 return됨
```

[소스 4-4] 변환 정보를 알아내는 코드

- [소스 4-4]는 특정 변환에 대한 정보를 얻는 방법을 나타낸다.
- gmtl의 행렬은 OpenGL과 호환성이 있기 때문에 OpenGL에서 바로 사용하는 것이 가능하다. (ex. glMultMatrix(mat.getData())

나. GMTL 행렬의 VTK 행렬로의 변환

VTK는 vtkTransform 클래스를 통해 오브젝트의 위치를 나타내거나 설정한다. gmtl 행렬을 VTK에 적용해서 오브젝트의 위치를 설정하기 위해서는 gmtl::Matrix44f 클래스를 vtkTransform 클래스의 형태로 변환해서 적용해야 한다. 이 때, gmtl 행렬을 VTK 행렬로 변환하는 과정(즉, column major 특성을 갖는 행렬을 row major 특성을 갖는 행렬로 변환하는 과정)은 [소스 4-5]와 같다.

```
1      void gmtlMatrixToVTKMatrix(vtkTransform *vtkxform, const gmtl::Matr
ix44f& vjmat)
2      {
3         const float *fmat = vjmat.getData();
4         double dmat[16];
5         int i, j;
6         for (i = 0; i < 4; i++)
7         {
8            for (j = 0; j < 4; j++)
9            {
10             dmat[i + 4 * j] = fmat(j + 4 * i];
11           }
12        }
13        vtkxform->Identity();
14        vtkxform->Concatenate(dmat);
```

```
15      }
```

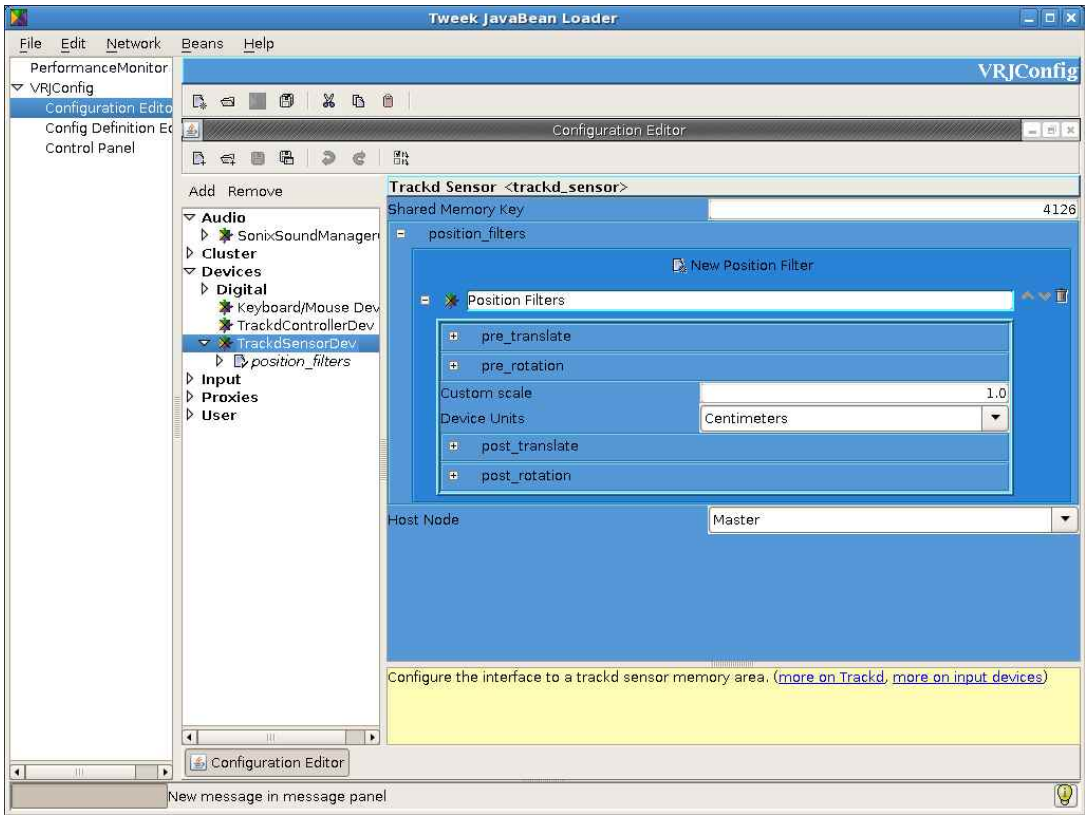
[소스 4-5] gmtl 행렬을 vtkTransform 행렬로 변환하는 코드

이렇게 gmtl 행렬을 VTK 행렬로 변환할 때, 몇가지 고려해야 할 사항이 있다. 제일 먼저 고려해야 할 것은 VRJuggler와 VTK 애플리케이션간의 단위다. 이런 단위에 대한 보정 작업은 간단한 함수 호출을 통해 해결할 수 있는데, gadget::PositionProxy::getData() 함수는 인자 없이 호출됐을 경우, 기본 단위의 데이터를 리턴한다. 다음과 같이 getDrawScaleFactor() 함수를 이용해서 getData() 함수를 인자를 입력하게 되면, 위치 좌표를 원하는 단위로 구할 수 있다. 이 때, 인자가 ConvertToMeters인 경우에는 getData() 함수를 통해 리턴된 좌표값에 1이, ConvertToCentimeters인 경우에는 100이 ConvertToFeet인 경우에는 3.28이 각각 곱해진다.

```
1       // 애플리케이션의 기본 단위로 Positional Data를 요청하는 방법
2        #include <gmtl/Matrix.h>
3       #include <gadget/Type/Position/PositionUnitConversion.h>
4       #include <gadget/Type/PositionInterface.h>
5       #include <vrj/Draw/OGL/GlApp.h>
6       class MyApp : public vrj::GlApp()
7       {
8         public:
9           MyApp() : vrj::GlApp ()
10          {
11              // Do nothing
12          }
13          virtual ~MyApp()
14          {
15              // Do nothing
16          }
17          void init()
18          {
19              mWand.init("VJWand");
20          }
21          // Use meters for the application units
22          float getDrawScaleFactor()
23          {
24             return gadget::PositionUnitConversion::ConvertToMeters;
25          }
26          void preFrame()
27          {
28             // Request the current wand transformation matrix in
29             // application units (meters).
```

```
30            const float units = getDrawScaleFactor();
31            gmtl::Matrix44f wand_mat (mWand->getData(units));
32            // Do something with the wand transformation...
33         }
34         void Draw()
35         {
36            // Draw something ...
37         }
38       private:
39          gadget::PositionInterface mWand;
40     };
```

[소스 4-6] 애플리케이션의 기본 단위로 위치 데이터를 요청하는 방법



[그림 4-1] Trackd Sensor Device 단위

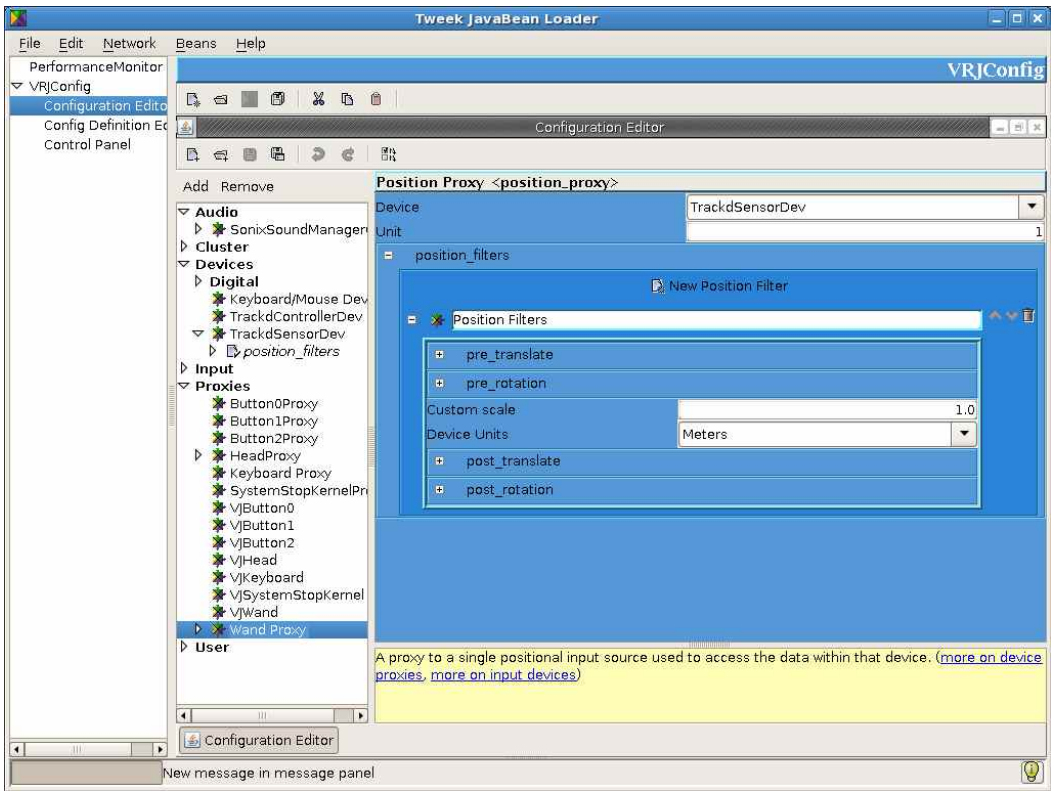
2번째로 체크해야 할 점은 VRJuggler Configuration에서 입력 디바이스의 좌표값을 받아들이는 단위다. VRJuggler Configuration에서 입력 디바이스를 설정할 때, 입력 디바이스에서 사용하는 좌표값의 단위를 설정할 수 있다. [그림 4-1]의 경우에는 trackd sensor에 대한 설정을 나타내는데, 포지션 필터를 추가하면서 여기에서 device unit값을 설정함으로써 단위 설정이 가능하다. 디바이스의 device unit에서 사용하는 단위는 실제 물리적으로 입력 디바이스에서 사용하는 단위와

일치해야 한다.

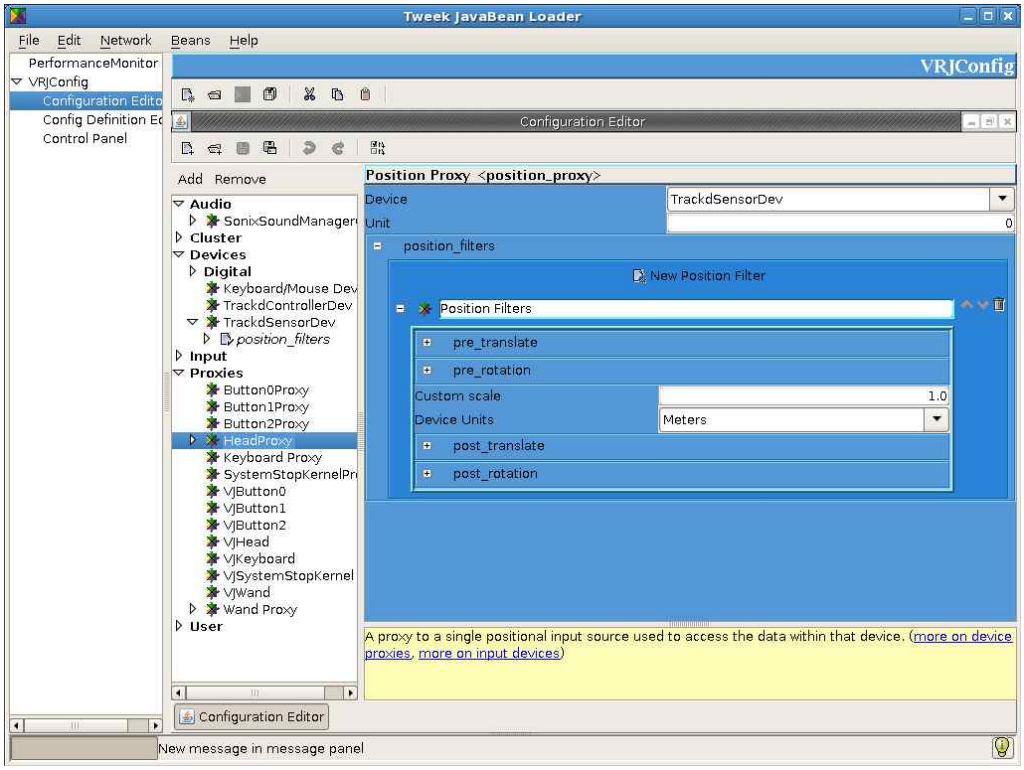
입력 디바이스에 대한 물리적 좌표 단위를 설정한 다음에는 실제 애플리케이션에서 사용하는 프록시 디바이스에서 사용하는 좌표 단위를 설정함으로써 애플리케이션에서 실제적으로 사용하는 좌표 단위를 설정할 수 있다. Configuration에서 애플리케이션에서 사용하는 단위를 설정할 수 있기 EOans에 실제 코드의 수정 없이 configuration의 설정만으로 애플리케이션에서 실제로 사용하는 단위를 변경, 전환하는 것이 가능하다.

Picasso 시스템에서는 trackd를 사용하므로 [그림 4-1]과 같이 trackd sensor의 device unit값을 실제 picasso 시스템의 trackd에 설정돼 있는 물리단위인 centi meters로 설정했다. 그리고 이 trackd sensor의 값을 받아서 사용하는 2개의 프록시, 즉 wand에 관한 좌표를 나타내는 Wand Proxy와 헤드 위치에 관한 좌표를 나타내는 Head Proxy의 Device Unit값을 각각 [그림 4-2], [그림 4-3]과 같이 Meter로 설정함으로써 애플리케이션에서 사용하는 단위로 맞췄다.

이 때, Wand의 위치를 나타내는 좌표 단위가 제대로 설정돼 있지 않으면 wand가 picking하는 위치가 엉뚱한 곳으로 인식되기도 하고, wand pointer를 wand 위치에 맞춰서 그릴 수도 없다.



[그림 4-2] Wand Proxy의 입력 단위



[그림 4-3] Head Proxy의 입력 단위

Head의 위치를 나타내는 좌표 단위는 전체 씬(Scene)에 대한 뷰에 영향을 준다. 따라서 이 값이 제대로 설정돼 있지 않으면 엉뚱한 위치에 그림이 그려지거나 wand를 보는 위치가 달라지는, 이상한 현상이 발생할 수도 있다.

이와같이 VRJuggler를 이용해서 애플리케이션을 구현할 때는, VRJuggler와 Application을 구성하는 라이브러리의 사이의 단위 조정이 중요하며, 특히 VRJuggler configuration에서 입력 디바이스의 좌표값을 받아들이는 단위를 잘 맞춰줘야만 무리없이 애플리케이션을 구현할 수 있다.

5. VRJuggler와 CaveLib

VRJuggler와 CaveLib은 가장 많이 사용되는 가상환경 구축용 툴이다. 4장에서는 VRJuggler와 CaveLib, 그리고 또다른 가상환경 구축용 툴인 FreeVR의 기능을 서로 비교해 보고, CaveLib 애플리케이션을 VRJuggler 환경으로 옮기는 과정에 대해 설명하겠다.

가. VR 라이브러리가 지원하는 시스템

VRJuggler와 CaveLib, 그리고 FreeVR이 지원하는 플랫폼 환경은 [표 5-1]과 같다.

VRJugger는 HP-UX를 제외한 대부분의 플랫폼을 지원하지만, CaveLib은 MacOS X, FreeBSD 및 Solaris 환경을 지원하지 않는 것을 알 수 있다.

[표 5-1] 각 라이브러리가 지원하는 플랫폼

	CaveLib	VRJuggler	FreeVR
IRIX	Y	Y	Y
Linux	Y	Y	Y
Win32	Y	Y	cygwin on XP
MacOS X	N	Y	Y
FreeBSD	N	Y	Y
HP-UX	Y	N	Y
Solaris	N	Y	Y

[표 5-2]에서는 각각의 VR 라이브러리가 지원하는 디스플레이 디바이스의 형태를 볼 수 있다.

[표 5-2] 각 라이브러리가 지원하는 디스플레이 디바이스

	CaveLib	VRJuggler	FreeVR
CAVE-like	Y	Y	Y
ImmersaDesk	Y	Y	Y
Powerwall-like	Y	Y	Y
HMD	Y	Y	Y
BOOM	Y	?	Y

VR 라이브러리 대부분이 현존하는 다양한 형태의 디스플레이 디바이스를 지원하는 것을 알 수 있다. 그러나 CaveLib은 설정에 대한 유연성이 떨어지며, 특히 안정성 부분에서 VRJuggler보다 신뢰도가 낮은 측면이 고려돼야 할 것이다.

나. CaveLib 환경과 VRJuggler 환경

여기에서는 실제 프로그래밍 작업에 적용할 수 있는 CaveLib 환경과 VRJuggler 환경의 차이에 대해 알아보기로 한다.

1) CaveLib 프로그램과 VRJuggler 프로그램

CaveLib 애플리케이션을 VRJuggler 애플리케이션으로 옮기려면 여러 요소를 고려해서 포팅 작업을 진행해야 한다. 우선 VRJuggler와 CaveLib의 소스코드 형태를 비교해보자.

```
void app_shared_init();
void app_compute_init();
void app_init_gl();
void app_draw();
void app_compute();

void
main (int argc, char **argv)
{
   CAVEConfigure(&argc, argv, NULL);
```

```
   app_shared_init(argc, argv);
   CAVEInit();
   CAVEInitApplication(app_init_gl, 0);
   CAVEDisplay (app_draw, 0);
   app_compute_init(argc, argv);
   while (!getbutton(ESCKEY))
     {
       app_compute();
     }
   CAVEExit();
}
```

[소스 5-1] CaveLib 프로그램의 형태

```
class MyApplication : public vrj::GlAPp
{
   public:
        // Data callbacks
        virtual void init();
        virtual void preFrame();
        virtual void intraFrame();
        virtual void postFrame();

        // OpenGL callbacks(put only OpenGL code here)
        virtual void contextInit();
        virtual void draw();
}

init
main(int argc, char* argv[])
{
vrj::Kernel* kernel;

    // configure kernel with *.config files
    kernel = vrj::Kernel::instance();  //Get the kernel
    for (int i = 1 ; i < argc ; i++)
      {
         //loading config file passed on command line...
         kernel->loadCOnfigFile(argv[i]);
      }

    // start the kernel
    kernel->start();

    //set the application for the kernel to run
MyApplication* application;

    application = new MyApplication();
    kernel->setApplication(application);
```

```
    // block until the kernel exits
    kernel->waitForKernelStop();

    return 0;
}
```

[소스 5-2] VRJuggler 프로그램의 형태

[소스코드 5-1]과 소스코드 [5-2]에서는 가장 기본적인 형태의 CaveLib 프로그램과 VRJuggler 프로그램을 보여주고 있다.

가장 큰 차이점은 CaveLib 프로그램은 프로그램에 대한 초기화 작업이 끝나고 난 뒤, Exit 신호가 올 때까지 main은 루프를 돌며 기다린다는 것이다. 반면, VR Juggler는 커널을 구동시키고 애플리케이션을 초기화한 뒤, 커널에서 Stop 신호를 받아서 프로그램을 종료한다. 그리고, CaveLib 프로그램은 애플케이션을 함수의 형태로 인식하지만, VRJuggler는 애플리케이션을 클래스의 형태로 인식한다는 점을 가장 큰 차이로 들 수 있겠다. 이런 차이점을 인식하고 있다면 CaveLib 애플리케이션의 VRJuggler 환경으로의 포팅 작업은 그다지 어렵지 않게 진행될 수 있을 것이다.

2) Initialize, Draw, 그리고 Frame

CaveLib은 C 함수 포인터로 구현돼 있는 callback을 이용해서 frame을 정의하고 초기화와 draw 작업을 수행한다. 반면, VRJuggler는 애플리케이션 오브젝트를 이용한 루틴이 “콜백(callback)"되는 것이로 볼 수 있다. 여기에서 애플리케이션 오브젝트는 애플리케이션의 기능을 캡슐화한 메소드가 정의돼 있는 C++ 클래스를 가리킨다.

CaveLib에서 Daw 루틴은 CAVEDisplay()에 함수 포인터를 전달함으로써 디스플레이 콜백 함수가 정의된다. 프레임 함수는 CAVEFrameFunction()에 정의돼 있으며, 초기화 작업은 CAVEInitApplication()으로 정의된 콜백 함수를 통해 이뤄진다.

VRJuggler에서는 C 함수 포인터가 아니라 애플리케이션 오브젝트에 대한 포인터가 커널에 주어져야 한다. 애플리케이션의 디스플레이 콜백 함수는 파생된 클레스에서 draw()라는 멤버 함수를 지정함으로써 정의된다. 이 draw() 함수에는 glBegin(), glVertex()같은 OpenGL 렌더링 명령이 나올 수 있다.

내비게이션, 충돌, 물리, 인공지능 등과 관련된 계산은 프레임 함수에 위치하게

되는데, VRJuggler의 프레임 함수는 3개의 멤버 함수로 나뉜다.

- MyApplication::preFrame(): draw() 함수에 앞서 호출된다.
- MyApplication::intraFrame(): draw() 함수가 실행되는 동안 호출된다.
- MyApplication::postFrame(): draw() 함수 뒤에 호출된다.

VRJuggler에서 초기화 과정은 데이터를 위한 초기화 멤버 함수와 디스플레이 리스트나 텍스쳐 오브젝트같은 context-specific한 데이터를 생성하기 위한 초기화 멤버 함수로 나눌 수 있다. 후자는 시스템에서 각각의 디스플레이 컨텍스트에 대해 호출된다.

- MyApplication::init(): 애플리케이션이 시작될 때 한번만 호출된다.
- MyApplication::contextInit(): 디스플레이 컨텍스트가 생성될 때마다 호출된다.

3) 디바이스로부터의 입력

CAVELib이 트래킹 정보를 얻는데 사용하는 함수는 다음과 같다.

- CAVEGetPosition(id, pos)
- CAVEGetOrientation(id, orient)
- CAVEGetVector(id, vec)
- CAVEGetSensorPosition(sensor, coords, pos)
- CAVEGetSensorOrientation(sensor, coords, orient)
- CAVEGetSensorVector(sensor, id, vec)

CAVELib이 버튼 입력에 사용하는 매크로는 다음과 같다.

- CAVEBUTTON1, CAVEBUTTON2, CAVEBUTTON3, CAVEBUTTON4, CAVE_JOYSTICK_X, CAVE_JOYSTICK_Y
- CAVEButtonChange()

반면, VRJuggler는 디바이스의 종류를 Positional, Digital, Analog 등의 인터페이스로 분류하는데, Gadgeteer에서 사용하는 입력 디바이스의 종류는 다음과 같다.

- Analog

- 연속 구간에 속한 입력값을 뜻하며, 디지털 컴퓨터에서 아날로그 값은 시뮬레이션을 통해 계산한다.
- Gadgeteer에서는 이 시뮬레이션을 floating-point 형태로 수행한다.

- Digital
  - 주로 “On"과 ”Off" 상태를 갖는 버튼 디바이스에 적합한 개념이지만, 이 2가지 값 외에도 몇 가지 다른 값을 가질 수 있다.
  - On/Off/Toggle On(이전 프레임에서 Off상태였다가 On으로 바뀜)/Toggle Off(이전 프레임에서 On 상태였다가 Off 상태로 바뀜), 이 4가지 형태의 값을 가지는 디바이스
- Position
  - Polhemus Fastrak이나 Ascension MotionStar같이 6DOF 트래커로부터 정보를 얻는 디바이스
  - 일반적으로 리턴값은 특정 트래커의 위치와 방향을 나타내는 4x 4 transformation 행렬이 된다.
- Simulator
  - 6DOF 트래커가 없는 환경에서 VR 애플리케이션을 사용할 경우, 마우스와 키보드로 6DOF 를 시뮬레이트할 수 있게 해주는 디바이스 모드
- Command
- Glove

이런 디바이스는 Gadgeteer의 DeviceInterface를 사용해서 인식할 수 있다. Gadgeteer에서 사용하는 DeviceInterface는 다음과 같다.

- gadget::PositionInterface: 트래커와 다른 position device에 사용
- gadget::DegitalInterface: 버튼 및 다른 on/off 디바이스
- gadget::AnalogInterface: potentiometer와 다른 multi-range 데이터 디바이스에 사용
- gadget::KeyboardMouseInterface: 키보드와 마우스 입력에 사용

4) 환경 설정

CAVELib은 모든 설정 변수를 .caverc 파일에 저장해서 사용하며, VRJuggler는 매우 복잡한 형태의 설정 파일을 사용한다.

6. 결론

VRJuggler는 가상현실 환경 구축을 위한 플랫폼을 제공하는 통합 환경 라이브러리로, 다양한 플랫폼과 다양한 형태의 디바이스를 지원할뿐만 아니라 애플리케이션을 구현한 프로그래밍 언어간 이식성도 높다. 하지만 이런 다양성을 보장하기 때문에 설치부터 설정에 이르기까지, 모든 과정은 복합적인 툴을 통해 이뤄지며, 그 과정 자체도 복잡하고 어려울 수밖에 없다. 그럼에도 불구하고 CAVELib보다 안정성이 높고 워낙 다양한 형태의 디스플레이를 지원한다는 장점을 가지고 있기 때문에 가장 널리 사용되고 있으며, 가상현실 환경을 구축하는데 최적의 환경을 제공하는 툴이라는 평가를 받고 있다.

본 문서에서는 이런 VRJuggler를 설치하고, 사용자의 환경에 맞게 설정하는 데 필요한 기법들을 다뤘다. 그리고 애플리케이션을 구현하는데 있어 필수적인 요소의 좌표계간 변환방법을 다루고, 그 예로 VTK와의 좌표계 변환 방법에 대해 다뤘다. 또, CAVELib과의 비교를 통해 기존의 CAVELib 애플리케이션을 VRJuggler 환경으로 이식하는 데 필요한 내용도 설명했다.

이렇게 다양한 기능을 지원하는 VRJuggler는 현재도 지속적인 개발 활동을 통해 프로그래밍 기술과 디바이스에 대한 진입장벽과 트래킹 디바이스에 대한 지원을 향상시키기 위해 노력하고 있다. 현존하는 VR 환경 구축용 라이브러리 중, 개발이 지속되고 있는 라이브러리는 VRJuggler밖에 없으며, 향후에는 더 많은 기능 개선과 다양한 지원을 통해 가상현실 환경을 구축하는 데 있어 최상의 환경을 제공하는, 유용한 툴이 될 것이다.