GLOVE 시스템 관리자 가이드

2017년

KiSTi www.kisti.re.kr
한국과학기술정보연구원
Korea Institute of Science and Technology Information

목 차

표 차례

그림 차례

표 차례

그림 차례

1. 서론

GLOVE는 고성능 컴퓨팅 환경을 위한 대용량 데이터 가시화 어플리케이션이다. GLOVE는 여러 오픈 소스 소프트웨어를 활용하기 때문에 원활한 실행을 보장하기 위해 추가로 설치해야 할 패키지가 상당히 많은 편이다. 특히 리눅스 기본 배포판이나 EPEL[1]에서 구할 수 없는 소프트웨어도 필요하고, 일부 소프트웨어는 KISTI가 독자적으로 패치를 진행했기 때문에 시스템 관리자와 개발자 입장에서 보면 소프트웨어 설치와 패키지 제작, 유지보수에 대한 부담이 다소 늘어날 수밖에 없는 것이 사실이다.

이 문서는 GLOVE를 설치/운영할 때 필요한 소프트웨어 관리방법에 대해 설명한다. 여기에는 GLOVE를 설치할 때 추가로 설치해야 하는 소프트웨어의 목록, rpm으로 구할 수 없는 소프트웨어를 rpm으로 제작하는 방법, rpm을 만들 때 주의해야 할 사항 등이 포함된다. 여기서 설명하는 모든 내용은 CentOS7과 EPEL repository를 사용하는 시스템을 기준으로 하지만 다른 rpm 기반 리눅스(CentOS6, Fedora 등)를 사용할 때에도 거의 동일하게 적용할 수 있다.

이 문서는 다음과 같이 구성되어 있다. 2장에서는 GLOVE 설치에 필요한 소프트웨어를 분류하고, 각 소프트웨어 사이의 의존성 관계에 대해 설명한다. 3장은 EPEL repository를 등록하고, EPEL repository로부터 소프트웨어를 설치하는 방법에 대해 설명한다. 4장~8장은 GLOVE가 사용하는 소프트웨어 패키지 중 가장 설치방법이 복잡한 핵심 프로그램(OpenSceneGraph, VR Juggler, Global Arrays toolkit, VTK)의 설치방법에 대해 구체적인 명령어와 함께 제시한다. 마지막으로 9장에서는 GLOVE의 설치 및 설정방법에 대해 설명한다.

1) Extra Packages for Enterprise Linux – 3장에서 별도로 설명한다.

2. 패키지 구성

앞에서 언급했듯이 GLOVE는 다양한 오픈소스 소프트웨어를 사용한다. 따라서 GLOVE를 설치하거나 GLOVE rpm을 제작하려면 먼저 GLOVE가 필요로 하는 소프트웨어를 모두 설치해야 한다. 문제는 일부 소프트웨어는 OS가 기본적으로 제공하지만, 몇몇은 홈페이지로부터 직접 소스코드를 받아와서 별도로 설치해야 하는 경우도 있다는 점이다. 이 장에서는 소스코드로부터 GLOVE를 컴파일하거나 GLOVE rpm을 제작할 때 필요한 소프트웨어를 분류하고, 각각의 소프트웨어를 확보하는 방법에 대해 설명한다.

가. 패키지 목록

GLOVE 설치에 필요한 소프트웨어는 표 1과 같이 구분할 수 있다. 표에서 말하는 Vault는 새로운 CentOS 배포판이 발표될 때마다 그 버전에 해당하는 스냅샷을 모아놓은 repository로 이해하면 된다. 그리고 EPEL은 대중적으로 잘 알려진 프로그램들을 대표적인 리눅스 배포판 전용 rpm으로 만들어서 제공하는 repository로 보면 무리가 없다(3장 참고).

구분	세부내역
OS 기본 패키지	• OS를 설치할 때 같이 설치되거나, OS 설치 후 DVD 또는 Vault로부터 바로 추가설치가 가능한 패키지
EPEL 패키지	• EPEL repository 추가 후 설치가 가능한 패키지
외부 패키지	• Vault, EPEL에서 제공하지 않기 때문에 직접 사이트로부터 소스코드를 받아서 설치해야 하는 패키지
KISTI 패치버전	• OS 기본 패키지, EPEL 패키지, 외부 패키지 중 KISTI에서 제작한 패치를 적용해서 업그레이드/대체하는 패키지
GLOVE	• GLOVE rpm

표 1. GLOVE 관련 소프트웨어 패키지의 구분

표 1의 구분에 따라서 실제로 설치가 필요한 소프트웨어의 구체적인 목록은 표 2에서 확인할 수 있다. 이 중 OS 기본 패키지와 EPEL 패키지는 yum[2]으로 설치하면서 의존성 문제까지 해결할 수 있다. 예를 들어서 yum으로 OpenMPI나 mvapich를 설치하면 인피니밴드 관련 라이브러리가, Global Arrays를 설치할 때에는 scalarpack, atlas 등 Global Arrays가 필요로 하는 선형대수학 문제해결 라이브러리가 자동으로 더 설치된다. 이와 같이 yum이 자동으로 의존성 문제를 해결해주는 프로그램들은 표 2에 명시하지 않았다. 하지만 Vault와 EPEL이 모두 제공하지 않기 때문에 시스템 관리자가 직접 의존성 문제를 해결해야 하는 패키지(특히 VR Juggler 관련 패키지)는 의도적으로 표 2에 포함시켰다.

2) Yellowdog Updater, Modified의 약자. 레드햇 계열 리눅스(RedHat, CentOS)에서 사용하는 패키지 관리 유틸리티

구분	이름	버전		특이사항
		CentOS6	CentOS7	
OS 기본 패키지	boost	1.41	1.53	
	gcc	4.4.7	4.8.5	
	log4cxx	미포함	0.10.0	아래 KISTI 패치버전 참고
	mpich	3.1	3.2	CentOS7은 3.0.4도 포함
	mvapich	1.2.1	2.2	
	OpenMPI	1.10	1.10.3	인피니밴드 지원
	SCons	2.0.1	2.5.1	
EPEL 패키지	boost			
	OmniORB	4.1.6	4.2.0	
	Global Arrays	5.3b	5.3b	아래 KISTI 패치버전 참고
	VTK	5.8.0	6.1.0	아래 KISTI 패치버전 참고
	math libraries			Global Arrays가 필요로 하는 패키지 세부목록 생략
외부 패키지	Oracle JAVA	8u131	8u131	Oracle로부터 별도 설치
	cppdom	1.2.0	1.2.0	
	doozer	2.1.6	2.1.6	
	flagpoll	0.9.4	0.9.4	
	GMTL	0.6.1	0.6.1	
	OpenSceneGraph	3.0.1	3.0.1	
	VR Juggler	3.1.8	3.1.8	
KISTI 패치버전	log4cxx	0.10.0	0.10.0	KISTI 패치 적용
	Global Arrays	5.5	5.5	5.3b를 5.5로 업그레이드 하면서 KISTI 패치 적용
	VTK	6.2.0	6.2.0	6.1을 6.2로 업그레이드
GLOVE	glove	2.0	2.0	glore, givi 포함

표 2. GLOVE를 설치하는 데에 필요한 외부 소프트웨어 목록

나. 패키지 의존성 다이어그램

그림 1과 그림 2는 GLOVE를 설치할 때 필요한 소프트웨어 사이의 의존성 관계를 나타낸 것이다. 시스템 관리자는 두 그림을 참고로 소프트웨어 설치순서를 확인하고, 그에 맞춰서 설치작업을 진행하면 된다.

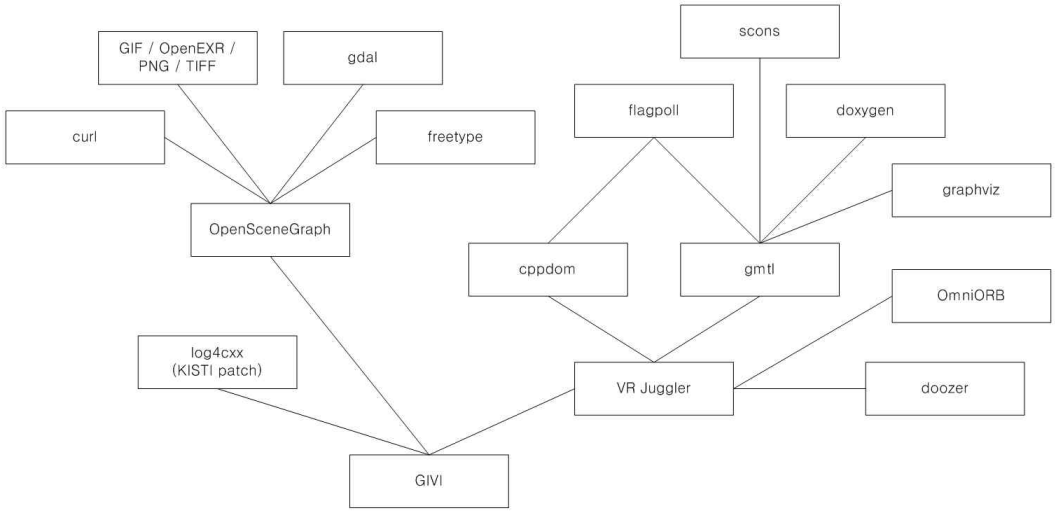

그림 1. GIVI 패키지 의존성 다이어그램

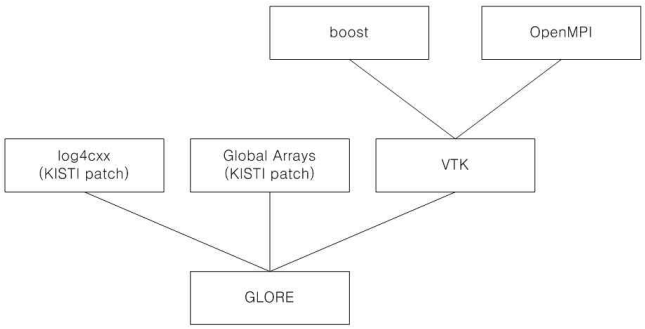

그림 2. GLORE 패키지 의존성 다이어그램

3. EPEL repository 등록

EPEL(Extra Packages for Enterprise Linux)은 리눅스의 기본 배포판에 포함되어 있지는 않지만 대중적으로 잘 알려진 소프트웨어를 rpm 형태로 공급하는 프로젝트다. EPEL은 레드햇, CentOS, Scientific Linux(SL), Oracle Linux(OL) 등 여러 리눅스 배포판에 대응하도록 구성되어 있으며, 기본 배포판에 포함되어 있는 rpm과의 충돌 문제가 거의 발생하지 않기 때문에 관리자 입장에서는 리눅스 기본 배포판을 설치할 때 EPEL을 같이 설치하거나 미리 받아놓는 것이 이후의 시스템 운용에 유리한 경우가 많다.

GLOVE 역시 EPEL이 제공하는 오픈소스 라이브러리를 사용하기 때문에 GLOVE를 설치하기 전에 EPEL 패키지를 설치할 수 있도록 준비할 필요가 있다. EPEL repository의 등록은 그림 3과 같이 할 수 있다. EPEL release rpm은 수시로 업데이트되기 때문에 버전은 달라질 수 있다.

```
# wget http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-9.noarch.rpm
# yum install epel-release
```

그림 3. CentOS7에서의 EPEL repository 활성화

EPEL repository가 성공적으로 등록된 후에는 /etc/yum.repos.d 디렉터리에 epel.repo와 epel-testing.repo 두 개의 파일이 추가된 것을 볼 수 있다.

```
# ls -l /etc/yum.repos.d
total 52K
-rw-r--r-- 1 root root 1664 Nov 30  2016 CentOS-Base.repo
-rw-r--r-- 1 root root 1309 Nov 30  2016 CentOS-CR.repo
-rw-r--r-- 1 root root  649 Nov 30  2016 CentOS-Debuginfo.repo
-rw-r--r-- 1 root root  314 Nov 30  2016 CentOS-fasttrack.repo
-rw-r--r-- 1 root root  630 Nov 30  2016 CentOS-Media.repo
-rw-r--r-- 1 root root 1331 Nov 30  2016 CentOS-Sources.repo
-rw-r--r-- 1 root root 2893 Nov 30  2016 CentOS-Vault.repo
-rw-r--r-- 1 root root 2150 May 21  2014 elrepo.repo
-rw-r--r-- 1 root root  957 Dec 28  2016 epel.repo
-rw-r--r-- 1 root root 1056 Dec 28  2016 epel-testing.repo
-rw-r--r-- 1 root root  173 Jul 19  2016 google-chrome.repo
```

그림 4. EPEL repository의 정상적인 등록여부 확인

이후 yum을 이용해서 패키지를 검색/설치할 때에는 EPEL repository도 대상에 들어가 있음을 확인할 수 있다.

```
Loaded plugins: auto-update-debuginfo, fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: data.nicehosting.co.kr
 * elrepo: repos.lax-noc.com
 * epel: mirror.premi.st
 * epel-debuginfo: mirror.premi.st
 * extras: centos.mirror.cdnetworks.com
 * updates: data.nicehosting.co.kr
```

그림 5. yum 명령을 실행할 때 EPEL repository가 등록되어 있을 때의 메시지

4. OpenSceneGraph

4장부터 7장까지는 GLOVE가 사용하는 오픈소스 소프트웨어 중 작업량이 상대적으로 많은 프로그램 – OpenSceneGraph, Global Arrays, VTK – 의 설치와 rpm 제작 방법에 대해 설명한다. 세 개의 프로그램은 각각 조금씩 다른 방법으로 설치를 해야 하므로 주의해야 한다. 우선 4장에서는 OpenSceneGraph의 설치 및 rpm 제작방법에 대해 설명한다.

VR Juggler는 렌더링을 위한 그래픽 라이브러리로 OpenGL, OpenSG, OpenGL Performer , OpenSceneGraph를 지원한다. 다만 VR Juggler는 최신 버전의 OSG(버전 3.4.x)를 제대로 지원하지 않는 한계가 있다. 2017년 9월 현재 VR Juggler와 가장 통합이 잘 되는 OSG 버전은 3.0.1이며 GIVI도 이를 반영해서 핵심 렌더링 루틴이 모두 OSG v3.0.1으로 구현되어 있다. 따라서 이 장에서 설명하는 모든 내용은 OSG v3.0.1을 기준으로 한다.

가. 사전 패키지 설치

OSG 역시 오픈소스 프로젝트고, 다양한 렌더링 루틴을 구현하기 위해 자체적으로 다른 오픈소스 소프트웨어를 사용한다. 다만, OSG가 Vault나 EPEL에 등록되어 있지는 않기 때문에 의존성이 걸린 모든 소프트웨어를 직접 설치해야 하는 문제가 있다. 작업량을 최소로 줄이기 위해 FFMPEG이나 COLLADA와 같이 GLOVE의 실행과 직접적인 관련이 없는 라이브러리의 설치는 생략했다.

1) Jasper 라이브러리

Jasper 라이브러리는 JPEG-2000 이미지를 만들어서 저장할 수 있는 라이브러리다. 이 프로그램은 CentOS7 Vault에 등록되어 있으며, OSG의 컴파일을 위해 devel 패키지도 추가로 설치해야 한다.

```
# yum install jasper jasper-devel
```

그림 6. jasper 라이브러리 설치

2) FreeType

FreeType은 벡터 폰트와 비트맵 폰트를 다양한 플랫폼에서 렌더링 할 수 있도록 지원하는 라이브러리다. 리눅스 배포판에 포함된 많은 소프트웨어들이 이 라이브러리를 사용하기 때문에 OS를 설치할 때 실행 라이브러리 정도는 기본적으로 설치될 가능성이 높다. 하지만 소프트웨어 개발에 필요한 파일(헤더파일 등)은 설치되지 않을 수도 있기 때문에 devel 패키지 설치여부를 확인할 필요가 있다.

```
# rpm -qi freetype-devel
```

그림 7. freetype-devel 패키지의 설치여부 확인

```
# yum install freetype freetype-devel
```

그림 8. freetype 라이브러리 설치

3) GDAL

GDAL(Geospatial Data Abstraction Library)은 지형 데이터 변환 라이브러리로 EPEL에 포함되어 있다. 따라서 GDAL을 설치하기 전에 EPEL repository를 미리 등록해야 한다.

```
# yum install gdal gdal-devel gdal-libs
```

그림 9. GDAL의 설치

4) GIF/OpenEXR/PNG/TIFF

OSG에서 특정 파일 포맷으로 이미지를 저장할 때 사용하는 라이브러리들이다. OS를 설치할 때 PNG와 TIFF는 실행 라이브러리만 설치되고, GIF와 OpenEXR은 관련 rpm이 아예 설치되지 않을 가능성이 높기 때문에 이 부분을 정확히 확인해서 소프트웨어 개발 라이브러리(*-devel)까지 설치해야 한다.

```
# yum install giflib giflib-devel
# yum install OpenEXR OpenEXR-devel OpenEXR-libs
# yum install libpng libpng-devel
# yum install libtiff libtiff-devel
```

그림 10. OpenEXR / PNG / TIFF 라이브러리 설치

나. OSG rpm 제작

앞에서 설명한 패키지의 설치가 모두 끝난 후 OSG를 컴파일/설치한다. OSG는 Vault와 EPEL 모두 rpm을 제공하지 않기 때문에 홈페이지(www.openscenegraph.org)로부터 소스코드를 받아야 한다.

```
# svn co http://svn.openscenegraph.org/osg/OpenSceneGraph/tags/OpenSceneGraph-3.0.1
```

그림 11. OSG 3.0.1 checkout

OSG의 컴파일은 cmake로 진행한다.

```
# cd OpenSceneGraph-3.0.1
# ccmake .
```

그림 12. cmake의 실행

cmake에서의 주요항목의 설정은 표 3과 같이 한다. FFMPEG을 사용하게 지정할 수도 있으나, OSG v3.0.1은 최신버전의 FFMPEG을 제대로 사용하지 못하는 문제가 있어서 의도적으로 사용하지 않았다. 표 3의 설정을 위해 필요한 라이브러리(jasper, freetype, gdal 등)는 모두 리눅스 배포판이나 EPEL에서 제공하기 때문에 별도로 설치할 필요는 없다. 다만, OSG 패키지를 만들기 전에 해당 라이브러리를 설치해야 한다. 그리고 cmake 내에서 config를 실행하면 대부분의 라이브러리는 자동으로 검색이 되니 일일이 지정할 필요는 없다.

구분	세부항목	내용
Freetype	FREETYPE_INCLUDE_DIR_freetype2	/usr/include/freetype2
	FREETYPE_INCLUDE_DIR_ft2build	/usr/include
	FREETYPE_LIBRARY	/usr/lib64/libfreetype.so
GDAL	GDAL_INCLUDE_DIR	/usr/include/gdal
	GDAL_LIBRARY	/usr/lib64/libgdal.so
GIF	GIFLIB_INCLUDE_DIR	/usr/include
	GIFLIB_LIBRARY	/usr/lib64/libgif.so
OpenEXR	OPENEXR_Half_LIBRARY	/usr/lib64/libHalf.so
	OPENEXR_INCLUDE_DIR	/usr/include
	OPENEXR_Iex_LIBRARY	/usr/lib64/libIex.so
	OPENEXR_IlmIlf_LIBRARY	/usr/lib64/libIlmImf.so
	OPENEXR_IlmThread_LIBRARY	/usr/lib64/libIlmThread.so
PNG	PNG_LIBRARY_DEBUG	미지정
	PNG_LIBRARY_RELEASE	/usr/lib64/libpng.so
	PNG_PNG_INCLUDE_DIR	/usr/include
TIFF	TIFF_INCLUDE_DIR	/usr/include
	TIFF_LIBRARY	/usr/lib64/libtiff.so

표 3. cmake로 OpenSceneGraph를 컴파일할 때의 설정내용

cmake 설정이 모두 끝난 후에는 make를 실행한다.

```
# make
```

그림 13. OSG 3.0.1 컴파일

OSG의 RPM은 별도의 spec 파일 없이, cmake를 이용해서 제작한다.

```
# cpack -G RPM --config <OSG_BUILD_DIR>/CPackConfig-openscenegraph-doc.cmake
```

그림 14. cmake를 이용한 OpenSceneGraph의 rpm 만들기

그림 14에서 OSG_BUILD_DIR은 OpenSceneGraph를 컴파일하기 위해 cmake를 실행했던 디렉터리로, cmake 실행 후 특별히 디렉터리를 바꾸지 않았다면 생략해도 된다. 위와 같이 하면 여러 개의 rpm 패키지가 만들어지는데, 그 중 openscenegraph-all이라는 패키지는 모든 파일을 하나의 rpm에 집약한 것이므로, 실무에서는 이 파일 하나만 설치하면 된다. 자세한 내용은 OpenSceneGraph 홈페이지[3]에서 확인할 수 있다.

3) http://trac.openscenegraph.org/projects/osg/wiki/Community/Packaging

5. VR Juggler

VR Juggler는 가상현실 어플리케이션 개발을 위한 프레임워크로 다양한 가상현실 입/출력장치 지원, 멀티 플랫폼 지원, 런타임 설정변경 등 풍부한 기능을 제공하며, 완성도가 높고 비교적 안정적인 작동을 보장한다. 지원하는 입/출력 장치의 종류가 상당히 많고, 오디오 출력이나 렌더링도 한 가지 방법으로 제한하지 않는 등 자유도가 높기 때문에 외부 라이브러리를 많이 사용하는 특징이 있다.

다행히 몇몇 소프트웨어는 Vault와 EPEL에서 바로 설치할 수 있는데, 여기에는 scons, boost, OmniORB, SDL, portaudio 등이 포함된다. 이 프로그램들은 yum을 이용하면 의존성 문제까지 해결하면서 기계적으로 설치할 수 있기 때문에 별도의 설명은 생략한다.

```
# yum search omniorb
중략
omniORB-devel.x86_64 : Development files for omniORB
omniORB-doc.x86_64 : Documentation files for omniORB
omniORB-servers.x86_64 : OmniORB naming service
omniORB-utils.x86_64 : Development files for omniORB
omniORB.x86_64 : A robust high performance CORBA ORB for C++ and Python
중략
# yum install omniORB-devel omniORB-servers omniORB-utils omniORB
```

그림 15. yum을 이용한 OmniORB의 설치

가. 사전 패키지 설치

앞에서 언급하지 않은 프로그램은 직접 홈페이지나 SourceForge, GitHub 등으로부터 다운받아서 설치해야 한다. 여기서는 이후의 관리 부담을 줄이기 위해 rpm 형태로 제작하는 방법까지 설명한다.

1) Oracle JAVA

Oracle JAVA는 vrjconfig과 같이 VR Juggler에 포함되어 있으면서 독립적으로 실행 가능한 일부 프로그램이 사용한다. java.oracle.com에서 RPM을 받을 수 있으며 프로그램 컴파일이 필요하기 때문에 JRE가 아닌, 소프트웨어 개발이 가능한 Java SE를 받아서 설치해야 한다.

```
# rpm -i jdk-8u131-linux-x64.rpm
Unpacking JAR files...
        tools.jar...
        plugin.jar...
        javaws.jar...
        deploy.jar...
        rt.jar...
        jsse.jar...
        charsets.jar...
        localedata.jar...
```

그림 16. Oracle JAVA 설치

정상적으로 설치되었는지의 여부는 그림 17과 같이 확인한다.

```
# rpm -qi jdk1.8.0_131
Name        : jdk1.8.0_131
Epoch       : 2000
Version     : 1.8.0_131

후략
```

그림 17. Oracle JAVA의 정상적인 설치여부 확인

2) flagpoll

Flagpoll은 소프트웨어 컴파일에 필요한 메타정보를 관리하기 위한 프로그램으로 VR Juggler 패키지를 제작할 때 필요하다. 이 프로그램은 CentOS Vault나 EPEL 모두 제공하지 않으며, GitHub에서 소스코드를 받아서 rpm을 만들어야 한다.

```
# git clone https://github.com/mccdo/flagpoll.git
```

그림 18. flagpoll 소스코드를 다운받는 방법

이 프로그램은 rpm을 만드는 방법이 조금 특이해서 소스코드 디렉터리 안에서 python 스크립트를 실행하면 바로 rpm이 만들어진다. 자세한 설명은 소스코드 안에 포함된 README 디렉터리에서 확인할 수 있다.

```
# python setup.py bdist_rpm
```

그림 19. flagpoll rpm의 제작

3) GMTL

GMTL은 수학연산 루틴을 템플릿 형태로 구현한 라이브러리다. 이 프로그램 역시 CentOS Vault나 EPEL 모두 제공하지 않기 때문에 http://ggt.sourceforge.net에서 소스를 가져와서 rpm으로 만들어야 한다. 소스코드 안에 gmtl.spec(C++ 라이브러리 rpm 제작)과 pygmtl.spec(Python 라이브러리 rpm 제작)이 포함되어 있는데, 이 중 gmtl.spec만 이용한다. 작업순서는 다음과 같다.

① 소스파일에서 cppdom.spec을 추출해서 ${HOME}/rpmbuild/SPECS 디렉터리에 복사한다.

② 소스파일을 압축된 상태 그대로 ${HOME}/rpmbuild/SOURCES 디렉터리에 복사한다.

③ ${HOME}/rpmbuild/SPECS 디렉터리에서 rpmbuild를 실행한다. 이 때 gmtl.spec 파일에서 Source: 가 SOURCES 디렉터리에 복사한 파일과 일치하는지 확인해야 한다.

```
# rpmbuild -ba gmtl.spec
```

그림 20. rpmbuild의 실행

④ rpmbuild가 정상적으로 진행되면 ${HOME}/rpmbuild/RPMS/noarch 밑에서 rpm 파일을 확인할 수 있다. 여기서 noarch는 CPU를 가리지 않고 설치가 가능한 패키지라는 뜻이다. src.rpm은 ${HOME}/rpmbuild/SRPMS 디렉터리에 저장되어 있다.

4) CppDom

CppDom은 XML 파일을 읽고 쓸 때 사용하는 라이브러리다. 이 프로그램도 CentOS Vault나 EPEL에서 제공하지 않기 때문에 https://sourceforge.net/projects/xml-cppdom/에서 소스를 가져와서 rpm으로 만들어야 한다. 소스코드 안에 cppdom.spec이 있기 때문에 이 파일을 이용하면 바로 rpm을 제작할 수 있다. 작업순서는 다음과 같다.

① 소스파일에서 cppdom.spec을 추출해서 ${HOME}/rpmbuild/SPECS 디렉터리에 복사한다.

② 소스파일을 압축된 상태 그대로 ${HOME}/rpmbuild/SOURCES 디렉터리에 복사한다.

③ ${HOME}/rpmbuild/SPECS 디렉터리에서 rpmbuild를 실행한다. 이 때 cppdom.spec 파일에서 Source: 가 SOURCES 디렉터리에 복사한 파일과 일치하는지 확인해야 한다.

```
# rpmbuild -ba cppdom.spec
```

그림 21. rpmbuild의 실행

④ rpmbuild가 정상적으로 진행되면 ${HOME}/rpmbuild/RPMS/x86_64 밑에서 rpm 파일을 확인할 수 있다. src.rpm은 ${HOME}/rpmbuild/SRPMS 디렉터리에 저장되어 있다.

5) Doozer

Doozer는 소프트웨어 컴파일 과정을 단순화하기 위해 사용하는 툴이다. 이 프로그램은 CentOS Vault나 EPEL 모두 제공하지 않기 때문에 https://sourceforge.net/projects/doozer/에서 소스를 가져와서 rpm으로 만들어야 한다. 소스코드 안에 doozer.spec이 있기 때문에 이 파일을 이용하면 바로 rpm을 제작할 수 있다. 작업순서는 다음과 같다.

① 소스파일에서 doozer.spec을 추출해서 ${HOME}/rpmbuild/SPECS 디렉터리에 복사한다.

② 소스파일을 압축된 상태 그대로 ${HOME}/rpmbuild/SOURCES 디렉터리에 복사한다.

③ ${HOME}/rpmbuild/SPECS 디렉터리에서 rpmbuild를 실행한다. 이 때 doozer.spec 파일에서 Source: 가 SOURCES 디렉터리에 복사한 파일과 일치하는지 확인해야 한다.

```
# rpmbuild -ba doozer.spec
```

그림 22. rpmbuild의 실행

④ rpmbuild가 정상적으로 진행되면 ${HOME}/rpmbuild/RPMS/noarch 밑에서 rpm 파일을 확인할 수 있다. 여기서 noarch는 CPU를 가리지 않고 설치가 가능한 패키지라는 뜻이다. src.rpm은 ${HOME}/rpmbuild/SRPMS 디렉터리에 저장되어 있다.

나. VR Juggler 설치

앞에서 설명한 프로그램의 설치가 모두 끝나면 VR juggler를 컴파일하고 rpm을 제작할 수 있게 된다. 가장 먼저 해야 할 일은 JAVA 관련 환경변수를 지정하는 것이다.

```
export JDK_HOME=/usr/java/default
```

그림 23. 환경변수 지정

vrjuggler.spec 파일은 release 디렉터리에 존재한다. 이 파일은 ${HOME}/rpmbuild/SPEC, 압축된 소스파일 원본은 ${HOME}/rpmbuild/SOURCES 디렉터리에 각각 복사한다.

VR Juggler가 사용하는 대부분의 라이브러리는 EPEL과 CentOS7 기본 패키지로 모두 설치가 가능하지만 일부 옵션 패키지는 정식 rpm으로 구하기 어렵다. 이때에는 VR Juggler 기반 어플리케이션의 실행이 크게 영향을 받지 않는 범위 내에서 VR Juggler가 사용하지 않도록 설정한다. 이 작업은 vrjuggler.spec 파일을 수정함으로써 진행할 수 있다.

```
#./configure.pl —with-boost=/usr —with-boost-includes=/usr/include          \
#                 --with-gmtl=/usr —with-openal=/usr --with-audiere=/usr    \
#                 --prefix=%{_prefix} %{abi_option} %{?_with_vrpn}           \
#                 %{?_with_ftd2xx} %{?_with_trackdapi}
```

```
./configure.pl —with-boost=/usr —with-boost-includes=/usr/include          \
               --with-gmtl=/usr --with-openal=/usr                          \
               --prefix=%{_prefix} %{abi_option} %{?_with_vrpn}             \
               --with-jdkhome=/usr/java/default                             \
               %{?_with_ftd2xx} %{?_with_trackdapi}
```

그림 24. VR Juggler의 spec 파일 수정

Audiere는 CentOS vault나 EPEL에서 모두 구할 수 없다. VR Juggler가 audiere 라이브러리를 반드시 필요로 하는 것은 아니기 때문에 이 라이브러리를 사용하는 부분은 spec에서 모두 제거했다.

```
#%define have_audiere %(if [ -x /usr/bin/audiere-config ] ; then echo 1; else echo 0; fi)

%package -n sonix-plugin-audiere
Summary: Sonix Audiere Plug-in
Version: %{sonix_version}
Release: %{sonix_release}
URL: http://www.vrjuggler.org/sonix/
Group: Development/C++
Requires: sonix = %{sonix_version}
Requires: audiere >= 1.9.3
BuildRequires: audiere-devel >= 1.9.3

%description -n sonix-plugin-audiere
#Sonix plug-in that uses Audiere to generate sounds.

%if %have_audiere
%files -n sonix-plugin-audiere
%defattr(-, root, root)
%{_libdir}/sonix-%{sonix_version_dist}/plugins/*/Audiere_snd.so
%endif
```

그림 25. vrjuggler.spec에서 주석처리를 해야 하는 부분

CentOS7에서 rpm을 만들 때 일부 파일이 만들어지지 않음에도 불구하고 spec 파일에 기술되어 있기 때문에 rpmbuild가 실패하는 경우가 여러 군데 존재한다. 이 때 문제가 되는 부분들을 주석으로 처리하면 대부분 해결할 수 있다.

```
%{_prefix}/share/jccl-%{jccl_version_dist}/beans/jccl_config.*
%{_prefix}/share/jccl-%{jccl_version_dist}/beans/jccl_editors.*

%dir %{_libdir}/sonix-%{sonix_version_dist}/plugins/dbg/
%dir %{_libdir}/sonix-%{sonix_version_dist}/plugins/opt/

%{_prefix}/share/vrjuggler-%{vrjuggler_version_dist}/java/CommonEditors.jar
```

그림 26. rpmbuild를 실행할 때 문제가 발생하기 때문에 주석으로 처리해야 하는 부분 (vrjuggler.spec)

그리고 spec 파일을 이용해서 VR Juggler rpm을 만들 때 일부 드라이버 모듈이 제대로 만들어지지 않는 경우가 있다. 이는 spec file에서 해당 드라이버를 만드는 부분을 주석 처리함으로써 해결했다.

```
%{_libdir}/gadgeteer-%{gadgeteer_version_dist}/drivers/P5Glove_drv.so
%{_libdir}/gadgeteer-%{gadgeteer_version_dist}/drivers/debug/P5Glove_drv.so
```

그림 27. vrjuggler.spec에서 주석처리를 해야 하는 드라이버 모듈

다. 주의사항

동일한 소스를 이용한다고 해도 VR Juggler를 rpm으로 만들 때 CentOS6과 CentOS7에서 만들어지는 헤더파일의 세부 내용에 차이가 있다.

6. Global Arrays toolkit

Global Arrays toolkit(이하 GA)은 클러스터의 각 노드의 메인메모리를 모아서 거대한 가상 공유메모리를 구성할 수 있도록 해주는 프로그램이다. GA를 사용하면 노드 한 대로는 다룰 수 없는 대규모 데이터를 클러스터의 각 노드에 분산저장하고, 어플리케이션이 필요로 하는 데이터를 임의로 가져와서 활용할 수 있게 된다. 이 때 데이터가 로컬 노드에 있을 수도 있고, 원격 노드에 있을 수도 있는데, GA는 데이터가 어떤 노드에 저장되어 있는지 상관하지 않고 일관된 방법으로 접근할 수 있도록 해준다.

CentOS7의 경우 GA v5.3 베타버전이 이미 EPEL을 패키지로 포함되어 있지만 GLOVE는 v5.5를 사용하고, 자체 패치도 필요하기 때문에 별도의 rpm을 만들어야 한다. 이 장에서는 KISTI 패치를 적용한 GA rpm을 만드는 과정에 대해 설명한다.

가. GA 설치를 위한 준비

GA는 자체적으로 행렬연산을 수행하기 때문에 선형대수 연산이 구현되어 있는 라이브러리를 필요로 하고, MPI 기반으로 클러스터 환경에서 사용되는 것을 전제한다. GLOVE와 함께 제공되는 GA v5.5를 설치하려면 선형대수 라이브러리나 MPI 관련 패키지를 일일이 확인해서 설치해야 하지만, 우선 yum으로 GA v5.3b를 설치하면서 의존성 문제를 해결하고 v5.5로 업그레이드 하는 것이 가장 쉽다. GA v5.3b를 설치할 때 추가로 설치되는 패키지는 표 4에서 확인할 수 있다.

구분	세부 패키지
scalapack	scalapack-mpich, scalapack-mpich-devel scalapack-openmpi, scalapack-openmpi-devel
blacs	blacs-mpich, blacs-mpich-devel blacs-openmpi, blacs-openmpi-devel
mpich	mpich, mpich-devel
lapack	lapack, lapack-devel
atlas	atlas, atlas-devel
openmpi	openmpi, openmpi-devel
gcc	gcc-c++, gcc-gfortran
기타	hwloc-devel, libibverbs-devel, openssh-clients, dos2unix

표 4. GA를 설치할 때 추가로 설치되는 패키지 목록

나. GA rpm 제작

여느 rpm과 마찬가지로 GA v5.5의 rpm을 만들려면 spec 파일이 필요한데, EPEL이 제공하는 v5.3b의 spec 파일을 사용하는 편이 가장 쉽다. spec 파일은 EPEL에서 src.rpm을 받아서 설치하면 ${HOME}/rpmbuild/SPECS 디렉터리에서 찾을 수 있다.

```
# wget https://dl.fedoraproject.org/pub/epel/7/SRPMS/g/ga-5.3b-14.el7.src.rpm
# rpm -i ga-5.3b-14.el7.src.rpm
```

그림 28. GA v5.3b의 src.rpm 설치

GA v5.3b의 spec 파일은 OpenMPI와 MPICH 각각에 대응하는 GA 패키지를 만들 수 있도록 구성되어 있다. 따라서 동일한 spec 파일을 사용하면 GA v5.5 역시 OpenMPI로 컴파일 한 버전과 MPICH로 컴파일 한 버전을 동시에 만들 수 있다. 한 가지 주의해야 할 사항은 v5.5에 대해 KISTI 패치가 적용돼야 한다는 점이다. 이를 위해 별도로 제공하는 패치 파일을 ${HOME}/rpmbuild/SOURCES 디렉터리에 미리 복사해 놓고 spec 파일을 그림 29과 같이 수정한다.

```
License:
Source:
Patch0: ga-5.5-kisti.patch
ExclusiveArch:
%setup -q -c -n %{name}-%{version}
pushd %{name}-%{ga_version}
%patch0 -p0
```

그림 29. spec 파일의 수정

마지막으로 GA 홈페이지에서 다운받은 v5.5 소스파일 원본을 ${HOME}/rpmbuild/SOURCES 디렉터리에 복사하고 rpmbuild를 실행하면 패치가 적용된 GA v5.5의 rpm 패키지가 만들어진다.

```
# rpmbuild -ba ga.spec
```

그림 30. GA rpm 제작

다. GA rpm 설치

EPEL의 GA 5.3b를 이미 설치했다면 이미 의존성 문제가 해결된 상태이기 때문에 v5.5를 바로 설치하면 된다. 실질적인 설치는 이미 설치되어 있는 버전의 업그레이드에 해당하므로 그림 31과 같이 rpm 명령으로 패키지 업그레이드를 진행하면 된다.

```
# rpm -Uvh ga-*.rpm
```

그림 31. GA 업그레이드

라. 설치 후 설정

GA를 rpm으로 설치하면 /etc/sysctl.d 디렉터리에 armci.conf라는 파일이 추가로 설치된다. 이 파일은 리눅스 커널의 max shared memory를 정하는데, 기본 값으로 128MB가 잡혀 있다.

```
# cat /etc/sysctl.d/armci.conf
kernel.shmmax = 134217728
```

그림 32. /etc/sysctl.d/armci.conf의 내용

이는 GLOVE를 실행하는 데에 지나치게 작기 때문에 메인 메모리가 허용하는 한도 내에서 최대한 큰 값으로 잡고 동일한 값을 바로 적용해줘야 한다. 예를 들어서 64GB의 메인메모리를 갖고 있는 노드라면 32GB 정도를 max shared memory로 지정할 수 있을 것이다. 그림 33과 같이 /etc/sysctl.d 디렉터리 내의 파일을 수정해 놓으면 시스템을 재부팅할 때에도 커널 값이 그대로 유지된다.

```
# cat /etc/sysctl.d/armci.conf
kernel.shmmax = 34359738368
# echo 34359738368 > /proc/sys/kernel/shmmax
```

그림 33. /etc/sysctl.d/armci.conf의 내용

7. VTK

VTK(Visualization ToolKit)는 각종 데이터 가시화 알고리즘이 구현되어 있는 라이브러리다. VTK는 미국의 주요 국립연구소와 매우 오랜 시간동안 개발이 진행됐으며 현재도 아주 활발하게 개발이 진행되고 있다. GLOVE 역시 GLORE에서 VTK를 주요 가시화 알고리즘을 구현하는 데에 사용하고 있다.

2017년 9월 현재 버전 8.0.1까지 발표되었지만 GLOVE는 v6.2를 사용하기 때문에 여기서는 v6.2를 기본으로 rpm을 제작하는 방법에 대해 설명한다.

가. 사전 패키지 설치

VTK는 대단히 많은 가시화 기법이 통합 구현되어 있는 만큼, 이 문서에서 소개하는 그 어떤 툴보다도 많은 오픈소스 라이브러리를 사용한다. 그림 34은 VTK가 사용하는 라이브러리의 일부를 나열한 것이다.

```
boost, doxygen, gl2ps, gdal, hdf5, java, libxml2, MPI(OpenMPI, MPICH, MVAPICH 등), jsoncpp,
ffmpeg, oggtheora, png, qt4, qt5, tcl, tk, java, mpi4py 등
```

그림 34. VTK에서 사용하는 오픈소스 프로젝트(일부 발췌)

대부분의 패키지는 Vault와 EPEL에서 yum으로 받아서 설치할 수 있다. 특히 EPEL이 제공하는 VTK v6.1을 yum으로 설치하면 의존성이 걸린 패키지를 같이 설치하기 때문에 우선 VTK v6.1을 설치하고 VTK v6.2로 업그레이드하는 방식을 취하는 것이 편리하다.

나. VTK rpm 제작

우선 VTK rpm을 제작할 때 필요한 외부 소프트웨어를 설치해야 한다(그림 35 참고). 그림 35에 나열한 외부 라이브러리의 대부분은 VTK v6.1을 설치하면서 같이 설치됐을 가능성이 높지만 실제로 VTK rpm을 제작할 때 특정 헤더파일이나 라이브러리가 없기 때문에 에러가 발생할 수도 있다. 이때는 상황에 따라서 필요한 라이브러리를 추가해가면서 오류를 해결하는 수밖에 없다.

```
java-devel, libX11-devel, libXt-devel, libXext-devel, libICE-devel, libGL-devel,
mesa-libOSMesa-devel, tk-devel, tcl-devel, python-devel, expat-devel, freetype-devel,
libjpeg-devel, libpng-devel, gl2ps-devel, libtiff-devel, zlib-devel, libxml2-devel, qt4-devel,
qtwebkit-devel, chrpath, doxygen, graphviz, gnuplot, boost-devel, hdf5-devel, libtheora-devel,
mysql-devel, postgresql-devel, wget
```

그림 35. VTK rpm을 제작할 때 필요한 외부 라이브러리

rpm을 만들기 이한 spec 파일은 EPEL이 제공하는 v6.1의 spec을 수정해서 사용한다. 다만, v6.1 rpm은 MPI 기반 병렬렌더링을 지원하지 않기 때문에 MPI를 지원하도록 수정할 필요가 있다. 이를 위해 VTK 소스 파일로 cmake를 실행한 후, 그 결과로 만들어지는 CMakeCache.txt에서 MPI 지원에 필요한 설정을 추출해서 spec 파일에 포함시켜야 한다.

```
//Cleared
MPI_CXX_COMPILER:FILEPATH=/usr/lib64/openmpi/bin/mpicxx

//MPI CXX compilation flags
MPI_CXX_COMPILE_FLAGS:STRING=

//MPI CXX include path
MPI_CXX_INCLUDE_PATH:STRING=/usr/include/openmpi-x86_64

//MPI CXX libraries to link against
MPI_CXX_LIBRARIES:STRING=/usr/lib64/openmpi/lib/libmpi_cxx.so;/usr/lib64/openmpi/lib/libmpi.so

//MPI CXX linking flags
MPI_CXX_LINK_FLAGS:STRING= -Wl,-rpath  -Wl,/usr/lib64/openmpi/lib  -Wl,--enable-new-dtags

//Cleared
MPI_C_COMPILER:FILEPATH=/usr/lib64/openmpi/bin/mpicc

//MPI C compilation flags
MPI_C_COMPILE_FLAGS:STRING=
```

그림 36. VTK v6.2 소스파일에서 cmake로 만들어낸 CMakeCache.txt(일부발췌)

그림 36은 cmake를 실행한 이후의 CMakeCache.txt 파일의 일부를 보여준다. 이 파일에서 불필요한 부분을 제거한 후 그 결과를 vtk.spec에 포함시켜야 한다. 제거대상은 다음과 같다.

- //로 시작하는 주석문과 공백라인
- 설정 값 중 INTERNAL이라고 되어 있는 부분
- X11로 시작하는 설정

```
//Result of TRY_COMPILE
H5_LDOUBLE_TO_UINT_ACCURATE_RUN:INTERNAL=0
X11_Xcursor_LIB:FILEPATH=/usr/lib64/libXcursor.so
```

그림 37. CMakeCache.txt를 vrjuggler.spec으로 복사할 때 제거해야 하는 정보(일부발췌)

```
mkdir build
pushd build
%if (0%{?rhel} <= 6)
%{cmake} .. \
%else
%{cmake} .. \
%endif
  -DBUILD_DOCUMENTATION:BOOL=ON \
  -DBUILD_EXAMPLES:BOOL=OFF \
  -DBUILD_SHARED_LIBS:BOOL=ON \
  중략
  -DMPIEXEC:FILEPATH=/usr/lib64/mpich/bin/mpiexec \
  -DMPIEXEC_MAX_NUMPROCS:STRING=2048 \
  -DMPIEXEC_NUMPROC_FLAG:STRING=-np \
  -DMPI_CXX_COMPILER:FILEPATH=/usr/lib64/mpich/bin/mpicxx \
  -DMPI_CXX_INCLUDE_PATH:STRING=/usr/include/mpich-x86_64 \
  -DMPI_CXX_LIBRARIES:STRING=/usr/lib64/mpich/lib/libmpichcxx.so \
  -DMPI_C_COMPILER:FILEPATH=/usr/lib64/mpich/bin/mpicc \
  -DMPI_C_INCLUDE_PATH:STRING=/usr/include/mpich-x86_64 \
  -DMPI_C_LIBRARIES:STRING=/usr/lib64/mpich/lib/libmpich.so \
  -DMPI_EXTRA_LIBRARY:STRING=/usr/lib64/mpich/lib/libmpich.so \
  -DMPI_LIBRARY:FILEPATH=/usr/lib64/mpich/lib/libmpichcxx.so \
  중략
  -DVTK_MPIRUN_EXE:FILEPATH=/usr/lib64/mpich/bin/mpiexec \
  -DVTK_MPI_MAX_NUMPROCS:STRING=2048 \
  -DVTK_MPI_NUMPROC_FLAG:STRING=-np \
  생략
```

그림 38. VTK v6.1의 spec 파일에서 MPI를 지원하도록 수정한 내역

그 다음 rpmbuild 를 이용해서 VTK rpm을 만든다

```
# rpmbuild -ba vtk.spec
```

그림 39. rpmbuild의 실행

다. VTK rpm 설치

앞에서도 언급했듯이 CentOS7 EPEL은 VTK v6.1을 제공한다. 따라서 의존성 해결을 위해 VTK v6.1을 먼저 설치하고, 이후 GLOVE와 함께 제공하는 VTK v6.2를 업그레이드 하는 것이 편리하다. VTK v6.1은 EPEL repository를 등록한 상태에서 그림 40과 같이 설치할 수 있다.

```
# yum install vtk vtk-devel
```

그림 40. EPEL의 VTK v6.1 설치

VTK v6.1이 설치된 상태에서 v6.2로의 업그레이드는 rpm 명령을 이용한다. VTK v6.2 rpm 파일이 위치한 디렉터리에서 그림 41과 같이 업그레이드를 진행한다.

```
# rpm -Uvh vtk*.rpm
```

그림 41. VTK v6.2로의 업그레이드

8. 기타 소프트웨어

가. log4cxx

CentOS7 Vault에 log4cxx 0.10.0이 포함되어 있지만 클러스터 환경에서 서로 다른 노드가 로그를 남기는 상황에 제대로 대응하지 못하는 한계가 있다. KISTI에서는 이 문제를 해결한 패치를 제공하고, 그에 기반을 둔 rpm도 별도로 제공하고 있다.

log4cxx rpm을 만드는 방법은 GA rpm을 만드는 과정과 거의 비슷하다. 제일먼저 패치 파일을 ${HOME}/rpmbuild/SOURCES에 복사한 후 spec 파일을 그림 42과 같이 수정한다. log4cxx의 원본 src.rpm을 보면 log4cxx-cstring.patch가 따로 있는데, 이미 동일한 내용이 KISTI 패치에도 있기 때문에 굳이 적용하지 않아도 된다.

```
Source0: http://www.apache.org/dist/logging/log4cxx/%{version}/apache-%{name}-%{version}.tar.gz
# Filed into upstream bugtracker at:
# https://issues.apache.org/jira/browse/LOGCXX-332
Patch0: log4cxx-0.10.0-kisti.patch
BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u} -n)
```

```
%prep
%setup -q -n apache-%{name}-%{version}
%patch0 -p1
```

그림 42. log4cxx spec 파일의 수정

패치 파일은 ~/rpmbuild/SOURCES 디렉터리에 미리 복사해둬야 한다. 일단 한 번 rpm을 만들면 src.rpm 안에 패치 파일이 포함되기 때문에 해당 src.rpm 파일을 재활용할 때에는 일일이 패치 파일을 복사하지 않아도 된다.

그 다음 src.rpm 파일 원본을 ${HOME}/rpmbuild/SOURCES 디렉터리에 복사하고 rpmbuild를 실행하면 KISTI 패치가 적용된 log4cxx의 rpm 패키지가 만들어진다.

```
# rpmbuild -ba log4cxx.spec
```

그림 43. GA rpm 제작

9. GLOVE

앞에서 설명한 프로그램의 설치가 모두 끝나면 비로소 GLOVE를 빌드/설치할 수 있게 된다.GLOVE는 소스 파일을 직접 컴파일해서 설치할 수도 있으나, 시스템 운영의 일관성을 보장하기 위해 rpm으로 설치/업그레이드하는 것이 바람직하다.

가. 설치준비

GLOVE를 정상적으로 컴파일/설치하려면 지금까지 언급한 모든 소프트웨어 패키지가 설치되어 있어야 한다.

나. cmake를 이용한 설치

cmake를 이용해서 GLOVE를 설치할 때에는 특정 사이트에서 문제가 발생했을 때 디버깅 목적으로 임시 설치를 할 때 우선 boost 라이브러리는 멀티쓰레드 버전으로 지정해야(특히 GIVI) 정상적인 작동을 보장한다.

항목	세부내용
BOOST_CHRONO_LIBRARY	/usr/lib64/libboost_chrono-mt.so
BOOST_FS_LIBRARY	/usr/lib64/libboost_filesystem-mt.so
BOOST_GRAPH_LIBRARY	/usr/lib64/libboost_graph-mt.so
BOOST_HOME	/usr
BOOST_INCLUDE_PATH	/usr/include/boost
BOOST_IOSTREAMS_LIBRARY	/usr/lib64/libboost/iostreams-mt.so
BOOST_LIBRARY_PATH	/usr/lib64
BOOST_REGEX_LIBRARY	/usr/lib64/libboost_regex-mt.so
BOOST_SIGNALS_LIBRARY	/usr/lib64/libboost_signals-mt.so
BOOST_SYSTEM_LIBRARY	/usr/lib64/libboost_system-mt.so
BOOST_THREAD_LIBRARY	/usr/lib64/libboost_thread.so

표 5. cmake를 이용한 boost 라이브러리 설정

GA의 주요 라이브러리는 OpenMPI 라이브러리 디렉터리(/usr/lib64/openmpi/lib)에 위치하고, 헤더파일 역시 OpenMPI의 헤더파일 디렉터리에 같이 존재한다.

항목	세부내용
GA_CPP_LIBRARY	/usr/lib64/openmpi/lib/libga++.so
GA_HOME	/usr/
GA_INCLUDE_PATH	/usr/include/openmpi-x86_64
GA_LIBRARY	/usr/lib64/openmpi/lib/libga.so
GA_LIBRARY_PATH	/usr/lib64/openmpi/lib

표 6. cmake를 이용한 GA 관련 설정

MPI는 OpenMPI를 사용하는 것을 전제하기 때문에 모든 설정을 OpenMPI 기준으로 해야 한다. 그런데 OpenMPI는 인피니밴드를 사용하기 때문에 인피니밴드를 사용하지 않는 클러스터에서

항목	세부내용
MPI_HOME	/usr
MPI_INCLUDE_PATH	/usr/include/openmpi-x86_64
MPI_LIBRARY_PATH	/usr/lib64
MPI_MPIEXEC	/usr/lib64/openmpi/bin/mpiexec

표 7. cmake를 이용한 MPI 관련 설정

VR Juggler 관련 설정은 표 8과 같이 한다.

항목	세부내용
VRJuggler_HOME	/usr
VRJuggler_INCLUDE_PATH	/usr/include/vrjuggler-3.1.8
VRJuggler_gadget_INCLUDE_PATH	/usr/include/gadgeteer-2.1.29
VRJuggler_gadget_LIBRARY	/usr/lib64/libgadget-2_1_29.so
VRJuggler_jccl_INCLUDE_PATH	/usr/include/jccl-1.5.1
VRJuggler_jccl_LIBRARY	/usr/lib64/libjccl-1_5_1.so
VRJuggler_sonix_INCLUDE_PATH	/usr/include/sonix-1.5.2
VRJuggler_sonix_LIBRARY	/usr/lib64/libsonix-1_5_2.so
VRJuggler_tweek_INCLUDE_PATH	/usr/include/tweek-1.5.1
VRJuggler_tweek_LIBRARY	/usr/lib64/libtweek-1_5_1.so
VRJuggler_vpr_INCLUDE_PATH	/usr/include/vpr-2.3.5
VRJuggler_vpr_LIBRARY	/usr/lib64/libvpr-2_3_5.so
VRJuggler_vrj_INCLUDE_PATH	/usr/include/vrjuggler-3.1.8
VRJuggler_vrj_LIBRARY	/usr/lib64/libvrj-3_1_8.so
VRJuggler_vrj_ogl_LIBRARY	/usr/lib64/libvrj_ogl-3_1_8.so

표 8. cmake를 이용한 VR Juggler 설정

ENABLE_LICENSE_CHECK는 상용으로 GLOVE를 납품할 때 반드시 ON으로 설정해야 한다. 현재 라이센스 파일은 glorem이 위치한 디렉터리에 같이 있어야 한다(라이브러리 자체 한계) BUILD_GIVI는 마지막으로 DEFAULT_CONFIG_PATH는 아직 CMakeLists.txt 내에서 완전히 정리가 안됐기 때문에 강제로 /usr/share/glove/config를 지정한다. 이 값은 CMAKE_INSTALL_PREFIX를 /usr로 설정했을 때를 가정하고 지정한 것이다.

항목	세부내용
ENABLE_LICENSE_CHECK	ON
BUILD_GIVI	ON
DEFAULT_CONFIG_PATH	/usr/share/glove/config

표 9. cmake를 이용한 GLOVE 설정

cmake 설정이 모두 끝나면 컴파일/설치를 마무리한다. CMAKE_INSTALL_PREFIX를 /usr로 설정했을 때에는 설치과정에 /usr 디렉터리에 파일을 써야 하므로 관리자 권한(root)을 갖고 해당 명령을 실행해야 한다.

```
# make
# make install
```

그림 44. GLOVE의 설치

다. rpm 패키지 제작

한 번 GLOVE 컴파일에 성공하면 rpm은 작업 디렉터리에서 바로 만들 수 있다.

```
# make package
```

그림 45. GLOVE rpm 제작

향후 GLORE와 GIVI가 서로 독립된 패키지로 만들어지도록 설정파일을 수정할 필요가 있다.

라. rpm을 이용한 설치

rpm을 이용해서 설치할 때에는 관리자 권한으로 다음과 같은 명령을 실행한다.

```
# rpm -Uvh --force glove.rpm
```

그림 46. GLOVE rpm 설치