ISBN 978-89-6211-402-7 98560

vjVTK: 분석 및 VRJuggler와의 연동

(Analysis on vjVTK)

허영주 (popea@kisti.re.kr)

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

목차

그림 차례

1. 서론

vjVTK는 VRJuggler 가상현실 프레임워크에서 VTK(Visualization Toolkit)을 자연스럽게 사용할 수 있게 해주는 툴킷으로, VR 환경에서 보다 쉽게 오픈-소스 기반의 가시화를 가능케 하려는 목적으로 설계됐다.

일반적으로 VR 환경에서의 과학 데이터의 가시화는 데이터 생성 -> 포맷 변환 -> 몰입형 가시화 시스템으로의 포팅의 과정을 거쳐서 이뤄진다. 즉, 가시화 데이터는 가시화 소프트웨어를 이용해서 생성되며, 이 가시화 데이터를 VR 시스템에 맞춘 형태로 변환한 후에야 가상현실 시스템에서 볼 수 있는 것이다. 따라서 파라미터의 변경이라던가, 가시화 조건의 변경 같은 상황은 가시화 소프트웨어에서만 적용할 수 있으며, 같은 데이터라도 가시화 조건이 변경되면 바로 가상현실 환경에서 적용해서 볼 수 없는 경우가 많다. vjVTK는 가상현실 환경의 이런 단점을 극복하고, 과학 데이터 가시화 과정을 단순화시킬 뿐만 아니라 실제적으로 사용자와 인터랙션을 수행할 수 있게 하는데 그 목표를 둔다.

본 문서에서는 vjVTK를 사용하는 데 있어 필요한 다양한 지식을 다룬다. 우선 설치 방법을 설명한 뒤, VRJuggler와의 연동 및 클래스에 대한 내용 분석한다. 그리고, 예제 프로그램을 설명함으로써 vjVTK를 이용해서 가상현실 환경에 맞는 애플리케이션을 구현하는 방법에 대해 구체적으로 설명할 것이다.

2. vjVTK 설치 및 VRJuggler 버전에 따른 수정

가. vjVTK의 설치

vjVTK는 윈도우와 리눅스 시스템에서 사용 가능하며, Mac 플랫폼은 지원하지 않는다. vjVTK의 웹 페이지인 http://imve.informatik.uni-hamburg.de/blom/vjVTK.html 사이트에서 제공하는 버전은 VRJuggler 2.0과 VTK 5.0 버전을 사용하고 있으며, OpenSG와 OpenSceneGraph는 선택적으로 사용 가능하다.

vjVTK의 빌드와 설치 작업은 VRJuggler의 빌드 시스템과 다소 유사한 점을 가지며, 리눅스 환경에서는 make에 기반을 둔 빌드 시스템을 사용한다. 빌드 시스템은 라이브러리와 예제만 생성하며, vjVTK를 빌드한 뒤에는 설치 작업이 뒤따라야 한다. 설치 작업에서는 여러 다양한 옵션을 선택할 수 있다.

리눅스 환경에서의 vjVTK를 빌드하는 과정은 다음과 같다.

1. 적절한 디렉토리에서 tarball을 푼다.
2. VJ_BASE_DIR 환경 변수를 설정한다.
3. VTK_DIR 환경 변수를 적절한 디렉토리로 설정한다. 따로 지정하지 않았다면 기본값은 /usr/local/이다.
4. make 명령을 실행한다.

예제와 테스트 프로그램은 make example과 make test 명령을 실행함으로써 컴파일할 수 있다. 원한다면 예제 프로그램은 하나씩 각각 컴파일해볼 수도 있다.

vjVTK의 선택 요소중에는 VTK와 SceneGraph를 연결하는 것이 있다. 실제로 SceneGraph와의 연결을 위해 따로 빌드를 실행할 필요는 없으며, 이 기능은 VTK_OSGApp.h나 VTK_OpenSGApp.h라는 App 파일을 통해 정의하는 것이 가능하다.

vjVTK는 VRJuggler 내에서 설치할 수도 있고, 단일 라이브러리로 설치할 수도 있으며, VRJuggler 내에서 설치하는 방법에도 2가지가 있다. 단하게 VRJuggler 내에 설치해서 사용하려면 vjVTK의 빌드 디렉토리 내에서 다음과 같은 과정을 수행하면 된다.

1. libvjVTK.so를 ${VJ_BASE_DIR}/lib 디렉토리로 복사한다.
2. 헤더 파일들을 적절한 장소로 복사해 넣는다. 즉 cp *.h ${VJ_BASE_DIR}/in

clude 명령을 실행한다.

이런 식으로 설치하면 VRJuggler에서 사용하기에 좀더 직관적이다.

VRJuggler 의 일부로 인식되게 설치하는 방법은 이보다는 좀더 복잡하다. vjVTK 사용자들은 2부분만 설치하면 되는데, 바로 라이브러리와 헤더 파일이다. libvjVTK.so 파일은 ${VJ_BASE_DIR}lib 디렉토리에 설치하고, 헤더 파일은 ${VJ_BASE+DIR}/instlinks/include/vrj/Draw/VTK/ 디렉토리를 새로 생성해서 설치한다.

vjVTK를 단일 라이브러리로 설치하기 위해 사용자가 필요로 하는 것은 libvjVTK.so 라이브러리와 헤더파일 뿐이다. 이 때, 사용자는 라이브러리 경로를 설정하고, 헤더의 위치를 알고 있어야 한다.

나. VRJuggler 2.2.1 버전에 대한 수정

현재 vjVTK는 VRJuggler 2.0 버전에 대해 작동하도록 구현돼 있다. 그런데, VRJuggler 2.2 이상의 버전에 대해 동작하게 하려면 컴파일에 앞서 약간의 수정이 필요하다. 다음은 VRJuggler 2.2버전에서의 수정내용이다.

- VRJuggler 2.2의 vrjconfig는 GUI를 사용하고 과거 버전의 vrjconfig와는 호환되지 않는다. 따라서 vjVTK의 Makefile에 있는 $(shell${VJ_BASE_DIR}/bin/vrjconfig vrj vrj_ogl --include)를 다음과 같이 수정해야 한다.

```
<수정 전: VRJuggler 2.0>

CXX= g++
CXX_OPTIONS= -LANG:std -g -fPIC -Wno-deprecated
VTK_DIR?=/usr/local
#VTK_INC_DIR= ${VTK_DIR}/include/vtk
VTK_INC_DIR= ${VTK_DIR}/include/vtk-5.0

#VTK_LIB_DIR=${VTK_DIR}/lib/vtk
VTK_LIB_DIR=${VTK_DIR}/lib/vtk-5.0

EXTRA_INCLUDE_FLAGS= -I${VTK_INC_DIR} -I. $(shell ${VJ_BASE_DIR}/bin/vrju
  ggler-config vrj vrj_ogl --include)
EXTRA_LIBRARY_FLAGS= -L${VTK_LIB_DIR} -L. $(shell ${VJ_BASE_DIR}/bin/vrju
  ggler-config vrj vrj_ogl --libs)
EXTRA_LIBRARIES=-lvjVTK -lvtkRendering -lvtkWidgets

 SUBDIRS= test examples
```

```
<수정 후: VRJuggler 2.2>

CXX_OPTIONS= -g -fPIC -Wno-deprecated -Wall

VJ_HOME=/usr/local/VJ
VJ_LIB_DIR=${VJ_HOME}/lib64
VPR_INC_DIR=${VJ_HOME}/include/vpr-2.0
JCCL_INC_DIR=${VJ_HOME}/include/jccl-1.2
SONIX_INC_DIR=${VJ_HOME}/include/sonix-1.2
TWEEK_INC_DIR=${VJ_HOME}/include/tweek-1.2
VRJUGGLER_INC_DIR=${VJ_HOME}/include/vrjuggler-2.2
GADGETEER_INC_DIR=${VJ_HOME}/include/gadgeteer-1.2

VTK_DIR=/usr
VTK_INC_DIR= ${VTK_DIR}/include/vtk
VTK_LIB_DIR=${VTK_DIR}/lib64/vtk-5.0

GMTL_INC_DIR=/usr/include/gmtl-0.5.4

EXTRA_INCLUDE_FLAGS= -I${VTK_INC_DIR} -I. -I${GMTL_INC_DIR} -I${VPR_INC_D
  IR} -I${JCCL_INC_DIR} -I${SONIX_INC_DIR} -I${TWEEK_INC_DIR} -I${VRJUGGL
  ER_INC_DIR} -I${GADGETEER_INC_DIR}
EXTRA_LIBRARY_FLAGS= -L${VTK_LIB_DIR} -L. -L${VJ_LIB_DIR}
g
```

[소스 2-1] makefile에 대한 vjVTK의 수정 내용

- vtkVJOpenGLCamera.cxx에서 vrjuggler/vrj/Display/Frustum.h의 버전 업그레이드 사항에 대한 수정 내용은 [소스 2-2]와 [소스 2-3]에 나와 있다.

```
< VR Juggler 2.0 >
...
   Frustum()
   {
      frust[0] = frust[1] = frust[2] = 0.0f;
      frust[3] = frust[4] = frust[5] = 0.0f;
   }
....
public:
   float frust[6];  /**< Left, Right, Bottom, Top, Near, Far */


< VR Juggler 2.2 >
...
Frustum();

.....
```

```
private:
   std::vector<float> mFrust;   /**< Left, Right, Bottom, Top, Near, Far */
}
```

[소스 2-2] Frustum.h에 대한 vjVTK의 업그레이드 내용

```
<수정전: VRJuggler 2.0 >
void vtkVJOpenGLCamera::SetVJProjection(vrj::Projection* projection)
{
...
  float* vjFrustum = projection->getFrustum().frust;
   for(int i = 0; i < 6; i++)
   {
      mVJFrustum[i] = vjFrustum[i];
   }
}

<수정후: VRJuggler 2.0 >
void vtkVJOpenGLCamera::SetVJProjection(vrj::Projection* projection)
{
...
   const std::vector<float> vjFrustum = (projection->getFrustum()).getValu
  es();
   for(int i = 0; i < 6; i++)
   {
      mVJFrustum[i] = vjFrustum[i];
   }
}
```

[소스 2-3] Frustum.h에 대한 vjVTK의 수정 내용

- [소스 2-4], [소스 2-5], [소스 2-6]은 vtkVJOpenGLRenderWindow.cxx에 대한 에러 메시지 및 처리 방법이다. VTK 5.2에서는 vtkVJOpenGLRenderWindow가 상속하는 vtkOpenGLRenderWindow 클래스에 void CreateAWindow()와 void DestroyWindow() 함수가 순수가상함수로 선언돼 있다. 따라서 이 클래스를 상속하는 클래스들은 모두 이 함수들을 실제로 구현해야만 정상적으로 인스턴스를 생성할 수 있다. 그런데 vtkVJOpenGLRenderWindow는 VTK5.0을 기반으로 만들었기 때문에 이들 함수를 구현하고 있지 않다. 따라서 이 함수들을 제대로 구현해야만 VTK5.2에서 정상적으로 컴파일할 수 있다. 일단은 이 함수를 필요로 하는 부분을 주석처리해서 사용한다.

```
vtkVJOpenGLRenderWindow.cxx:525: error: ‘class vtkVJOpenGLRenderWindowInt
  ernal’ has no member named ‘OffScreenContextId’
```

[소스 2-4] vtkOpenGLRenderWindow.cxx에서 발생하는 에러

```
class vtkVJOpenGLRenderWindowInternal
{
  friend class vtkVJOpenGLRenderWindow;
private:
  vtkVJOpenGLRenderWindowInternal(vtkRenderWindow*);

  GLXContext ContextId;

};
```

[소스 2-5] vtkOpenGLRenderWindow.cxx에서 발생하는 에러 발생 원인

```
// 다음과 같은 정의 부분을 모두 없앤다.
#ifdef VTK_OPENGL_HAS_OSMESA
     || this->Internal->OffScreenContextId
#endif
```

[소스 2-5] vtkOpenGLRenderWindow.cxx에서 발생하는 에러 해결방법

- vtkVJRenderWindow 클래스중 VRJuggler 2.0에서 수정된 관련 업그레이드 사항은 vrj/Draw/OGL/GlUserData.h에 있다. [소스 2-6]은 이 업그레이드 사항을, [소스 2-7]은 vjVTK에서 발생하는 에러 메시지를, 그리고 [소스 2-8]은 수정 사항을 보여준다. 또, [소스 2-9]에서는 vtkVJRenderWindow.cxx에서의 처리 내역을 볼 수 있다.

```
< VR Juggler 2.0 >
...
   GlWindow* getGlWindow()
   {
      return mGlWindow;
   }


protected:
   User*       mUser;       /**< The current user we are rendering */
   Projection*  mProj;       /**< The current projection being used */
   Viewport*    mViewport;    /**< The current vrj viewport being used */
```

```
   GlWindow*    mGlWindow;    /**< The current GL window that we are renderi
  ng in. (basically the gl context) */
};


< VR Juggler 2.2 >
const GlWindowPtr getGlWindow() const
   {
      return mGlWindow;
   }

protected:
   User*       mUser;       /**< The current user we are rendering */
   Projection*  mProj;       /**< The current projection being used */
   Viewport*    mViewport;    /**< The current vrj viewport being used */
   GlWindowPtr  mGlWindow;    /**< The current GL window that we are render
  ing in. (basically the gl context) */
};
```

[소스 2-6] vrj/Draw/OGL/GlUserData.h의 업그레이드 내역

```
vtkVJRenderWindow.cxx: In constructor ‘vtkVJRenderWindow::vtkVJRenderWind
  ow()’:
vtkVJRenderWindow.cxx:50: error: no matching function for call to ‘boost::
  shared_ptr<vrj::GlWindow>::shared_ptr(NULL)’
/usr/include/boost/shared_ptr.hpp:144: note: candidates are: boost::share
  d_ptr<T>::shared_ptr() [with T = vrj::GlWindow]
/usr/include/boost/shared_ptr.hpp:131: note:              boost::shared_p
  tr<vrj::GlWindow>::shared_ptr(const boost::shared_ptr<vrj::GlWindow>&)
```

[소스 2-7] vtkVJRenderWindow.h에서 발생하는 에러 메시지

```
< 수정 전>
#include <vrj/Draw/OGL/GlWindow.h>

< 수정 후>
#include <vrj/Draw/OGL/GlWindowPtr.h>
```

[소스 2-8] vtkVJRenderWindow.h에서의 수정 내역

```
#include <vrj/Draw/OGL/GlWindowPtr.h> 추가
```

[소스 2-9] vtkVJRenderWindow.cxx에서의 처리 내역

- vtkVJWin32OpenGLRenderWindow 클래스에서, VTK5.2에서는 이 클래스가 상속하는 vtkWin32OpenGLRenderWindow에서 필요로 하는 vtkWin32RenderWindowInteractor 클래스를 필요로 한다. 따라서 이 클래스에서도 vtkWin32RenderWindowInteractor를 상속하는 vtkVJWin32RenderWindowInteractor와 같은 클래스가 필요하며, Win32 환경에서 진행되는 프로젝트의 경우에는 반드시 이 클래스들을 구현해야 한다. 그러나 리눅스 환경을 기반으로 하는 프로젝트의 경우에는 이 클래스에 대한 구현은 불필요하므로 삭제만으로 컴파일 문제를 해결하는 것이 가능하다.

다. VTK 5.2.1 버전에 대한 수정

현재 제공되고 있는 vjVTK는 VTK 5.0버전을 사용하도록 돼 있다. 그런데, VTK 5.2 이상의 버전에서 사용할 경우는 컴파일 에러를 비롯한 각종 오류가 발생하게 된다. 여기에서는 vjVTK를 VTK 5.2.1 버전과 함께 사용할 수 있게 수정하는 방법에 대해 정리해 본다.

1) 컴파일 에러의 제거

vjVTK의 핵심 클래스인 vtkVJOpenGLRenderer 클래스와 vtkVJOpenGLRenderWindow 클래스는 각각 vtkRenderer와 vtkRenderWindow를 상속받는다. 그런데 VTK가 버전 5.2로 업그레이드되면서 이들 vtkRenderer와 vtkRenderWindow에는 순수 가상함수가 추가되었는데, 이 클래스의 하위 클래스들은 이들 가상함수를 반드시 구현해야 한다. 그러나 vjVTK의 클래스에는 이들 가상함수가 구현돼 있지 않기 때문에 컴파일 에러가 발생할 수밖에 없다. 하위 클래스에서 구현해야 하는 가상함수는 다음과 같다.

- vtkRenderer
  - vtkOpenGLRenderer의 상위 클래스인 vtkViewport에 다음과 같은 변경사항이 있었다.
  - int GetPickedIds(unsigned int atMost, unsigned int *callerBuffer) 가상함수가 추가됨.
  - unsigned int GetNumPickedIds() 가상 함수가 추가됨

- vtkRenderWindow
  - void CreateAWindow() 가상함수가 추가됨
  - void DestroyWindow() 가상함수가 추가됨

이에 대한 해결 방법은 다음과 같다.

- vtkOpenGLRenderer
  - GetPickedIds() 함수와 GetNumPickedIds() 함수를 추가, 구현한다. 이 함수들은 vtkVJOpenGLRenderer의 형제 클래스인 vtkOpenGLRenderer에도 구현돼 있으므로 이를 그대로 차용했다. 이 때, 이 클래스에서 사용하는 클래스 중 vtkGLPickInfo 클래스가 있는데, 여기에도 GLuintNumPicked 멤버 변수를 추가했다.
- vtkVJOpenGLRenderWindow
  - CreateAWindow()와 DestroyWindow()가 순수가상함수이므로 이에 대한 구현이 필요하다. 그런데 이 함수들은 모두 디바이스에 종속적이기 때문에 VRJuggler에 종속적인 함수를 구현해야 하는데, 이는 많은 시간을 필요로 하므로 현재는 일단 빈 함수로 추가했다. 확인 결과 !CreateAWindow() 함수는 호출하지 않고 DestroyWindow() 함수는 최종 종료시 호출되기 때문에 당장 실행하는 데는 문제가 없다.

2) 런타임 에러의 제거

컴파일시 발생하는 에러사항에 대해 위와 같이 조치를 취해서 컴파일한 후, 실제로 실행해 보면 Segmentation Fault 에러가 발생하면서 프로그램이 종료되는 것을 볼 수 있다. 이는 vjVTK에서 vtkOpenGLRenderer와 vtkOpenGLRenderWindow 클래스의 대체 클래스만 구현했을 뿐, vtkOpenGLActor나 vtkOpenGLProperty에 대한 클래스를 구현하지 않고 OpenGL용 클래스를 그대로 가져다 쓰기 때문에 발생한 문제다. 실제로 에러가 발생한 부분은 다음과 같다.

```
void vtkOpenGLProperty::Render(vtkActor *anActor,
                               vtkRenderer *ren)
{
  int i;
  GLenum method;
  float Info[4];
```

```
GLenum Face;
double  color[4];

// unbind any textures for starters
vtkOpenGLRenderer *oRenderer=static_cast<vtkOpenGLRenderer *>(ren);

if(oRenderer->GetDepthPeelingHigherLayer())
  {
  GLint uUseTexture=-1;
  uUseTexture=oRenderer->GetUseTextureUniformVariable();
  vtkgl::Uniform1i(uUseTexture,0);
  }
```

[소스 2-10] Segmentation Fault 에러가 발생하는 지점

여기에서 Render() 함수에 인자로 들어오는 vtkRenderer *ren의 실제 타입은 vtkVJOpenGLRenderer다. 그런데, `vtkOpenGLRenderer *oRenderer=static_cast<vtkOpenGLRenderer *>(ren)` 코드 부분에서 이것을 vtkOpenGLRenderer로 강제 캐스팅하는 것을 볼 수 있다. 이는 실제 이 클래스가 vtkOpenGLProperty이기 때문에 받아오는 renderer도 vtkOpenGLRenderer라고 가정하고 코딩한 것이다. vjVTK에서는 vtkVJOpenGLProperty를 구현하지 않고 vtkOpenGLProperty를 차용해서 사용하기 때문에 이런 문제가 발생한 것이다.

이렇게 강제로 캐스팅된 renderer는 GetDepthPeelingHigherLayer() 함수를 호출하고, 이 값에 따라 조건문의 분기를 결정하게 된다. 이 함수는 vtkOpenGLRenderer의 멤버 변수인 DepthPeelingHigherLayer 값을 돌려주는 함수인데, vtkOpenGLRenderer를 처음 생성할 때 0으로 처기화된다. 그러나 실제 renderer인 vtkVJOpenGLRenderer는 이 값을 초기화하지 않으므로 실제 실행할 때는 어떤 값이 들어가 있을 지 알 수 없다. (사실 vtkVJOpenGLRenderer에는 이 변수가 정의돼 있지도 않기 때문에 어떤 값을 접근할 지도 알 수 없다.) 따라서 우연히 이 값이 0인 경우는 조건문 안으로 들어가지 않게 되고 0이 아닌 경우에는 조건문 안으로 들어가게 된다. 만약 조건문을 실행하게 되면 큰 문제가 발생하게 된다.

vtkOpenGLRenderer에서는 Depth Peeling을 위한 쉐이더를 생성해 놓는데, 이 조건문 안에서는 이 쉐이더의 UseTexture라는 uniform 변수에 접근하도록 한다. 그런데, vtkVJOpenGLRenderer에서는 이런 쉐이더와 관련된 어떤 정의도 존재하지 않으므로 접근할 쉐이더가 없기 EOans에 이부분에서 segmentation fault 에러가 발생하게 되는 것이다.

이 문제를 해결할 수 있는 가장 간단한 방법은 위의 소스 코드중 if 조건문을 모두 없애는 것이다. 이렇게 하면 vjVTK에서 쉐이더와 관련된 부분을 접근하지 않

게 함으로써 segmentation error를 방지할 수 있다. 그러나 이렇게 하면 VTK 애플리케이션들 중 DepthPeeling 기능을 필요로 하는 애플리케이션에서 문제가 발생할 수도 있고, vtkOpenGLRenderer나 기타 다른 클래스들과 연관된 다른 문제가 잠재적으로 존재하게 된다.

당장 vjVTK가 실행되게 하면서 다른 VTK 애플리케이션에 영향을 주지 않도록 하는 가장 간단한 방법은 위 코드에서 ren을 vtkOpenGLRenderer로 캐스팅하기 전에 타입 체크를 하는 것이다. 즉, ren의 실제 클래스 이름을 GetClassName이나 IsA 함수 등으로 파악하고, vtkOpenGLRenderer면 조건문 이하 루틴을 실행하도록 하고 vtkVJOpenGLRenderer면 이 부분을 건너뛰도록 하는 것이다. 하지만 이 방안 역시 근본적인 해결책은 될 수 없을 것이다.

이 문제를 해결할 또 다른 방법은 vtkVJOpenGLRenderer와 vtkVJOpenGLRenderWindow 클래스를 VTK 5.2.1의 vtkOpenGLRenderer와 vtkOpenGLRenderWindow 클래스를 참조해서 수정하는 것이다. 또, vtkVJOpenGLActor와 vtkVJOpenGLProperty도 다시 새로 구현해야 한다. 이는 VTK 5.2.1과 관련된 여러 에러들을 해결할 수 있는 근본적인 방법이 될 수 있다.

3. vjVTK의 클래스

개요 부분에서 설명했듯이, vjVTK는 VR 환경에서 보다 쉽게 오픈-소스 기반의 가시화를 가능케 하려는 목적으로 설계으며, 이를 위해 vjVTK는 VTK를 오픈-소스 VR 소프트웨어 프레임워크인 VRJuggler에 임베드했다. 이 모델은 vtkCAVE의 선례를 따른 것으로, 이 라이브러리는 VTK를 CAVELib에 포함시킨 것이다.

이제, vjVTK의 각 클래스에 대해 알아보자.

1) VTKApp (VTKApp.h)

- VTK 애플리케이션을 캡슐화한 클래스로, VTK를 기반으로 하는 애플리케이션 클래스들이 반드시 포함해야 하는 기본 클래스들을 정의한다.
- OpenGL Draw Manager를 사용한다.
- class VTKApp : public GlApp, public VTKApp_mixin

2) VTKApp_mixin (VTKApp_mixin.h)

- VTK 애플리케이션을 캡슐화한 클래스로, VTK를 기반으로 하는 애플리케이션 클래스들이 반드시 포함해야 하는 mixin 클래스를 정의한다.
- 이 클래스는 렌더링 메소드와 관련 없는 VTK 관련 부분들을 지원한다.
- 루트 클래스다. (상위 클래스가 없다.)
- OpenGL과 VTK의 혼합 클래스
- vtkCamera를 VJ-OpenGL 카메라로 변환하는 기능을 지원
- vtkRenderWindow를 VJ Render Window로 변환하는 기능을 지원
- 렌더링에 vtkRenderer를 사용한다.

3) RenderCommand (vjRenderCommand.h)

- Renderer의 vtkProp 기반 커맨드에 대한 concrete 커맨드로 Props, Actors 및 Volumes를 포함한다.
- Prop 커맨드를 실행한다. (add/remove prop/actor/volume)
- Light 커맨드를 실행한다. (add/remove light)
- 루트 클래스다. (상위 클래스가 없다.)

4) Renderer(vjRenderer.h)

- vtkRenderer 타입에 대한 프록시로, 기존의 설계 패턴과는 다른 양상을 보인다.
- AddActor, AddProp, AddLight 등과 같이 vtkRenderer의 많은 기능을 지원하며, 실제로는 프레임 함수가 AddProps같은 함수를 사용할 때에 대비해서 컨텍스트를 안전하게 저장하는 역할을 수행하는 식으로 동작한다.
- Juggler 애플리케이션을 위한 "vtkRenderer"라 보면 무방하다.
- add/remove prop/actor/volume 및 add/remove light 기능이 구현돼 있으며, 이 클래스 내에서 처리된다.
- prop/actor/volume/light 관련 정보를 저장한다.
- 루트 클래스다. (상위 클래스가 없다.)

5) vtkVJOpenGLRenderer (vtkVJOpenGLRenderer.h)

- vtkRenderer의 함수는 VJOpenGL 커맨드로 이뤄져 있다.
- class VTK_RENDERING_EXPORT vtkVJOpenGLRenderer : public vtkRenderer

6) vtkVJOpenGLRenderWindow (vtkVJOpenGLRenderWindow.h)

- vtkRenderWindow의 함수는 VJOpenGL 커맨드를 사용해서 구현돼 있다.
- vtkVJOpenGLRenderWindow는 vtkRenderWindow 클래스를 구체화한 클래스로, OpenGL 그래픽스 라이브러리에 대한 인터페이스를 담당한다.
- 애플리케이션 프로그래머들은 보통 OpenGL 대신 vtkRenderWindow를 사용해서 프로그래밍 작업을 수행하게 된다.
- class VTK_RENDERING_EXPORT vtkVJOpenGLRenderWindow : public vtkVJRenderWindow

7) vtkVJOpenGLCamera (vtkVJOpenGLCamera.h)

- vtkOpenGLCamera는 vtkCamera라는 추상 클래스를 상속받은 하위 클래스로, OpenGL 렌더링 라이브러리에 대한 인터페이스다.
- class VTK_RENDERING_EXPORT vtkVJOpenGLCamera : public vtkCamera

8) vtkVJRenderWindow (vtkVJRenderWindow.h)

- vtkRenderWindow라는 추상 클래스를 상속받은 하위 클래스로, OpenGL 그래픽스 라이브러리의 해당 기능에 대한 인터페이스를 담당한다.
- VRJuggler Window에 대한 포인터, 윈도우 크기/위치, 사용자 뷰포트 크기/위치 등에 대한 정보를 VRJuggler로부터 가져와서 vtkRenderwindow에서 렌더링하는 데 사용할 수 있게 하는 클래스
- vtkVJOpenGLRenderWindow 클래스의 상위 클래스.
- class VTK_RENDERING_EXPORT vtkVJRenderWindow : publci vtkOpenGL RenderWindow

9) vtkVJInteractor (vtkVJInteractor.h)

- vtkRenderWindowInteractor라는 추상 클래스를 상속받은 하위 클래스.
- class VTK_RENDERING_EXPORT vtkVJInteractor : public vtkRenderWindowInteractor

10) vtkVJPicker (vtkVJPicker.h)

- vtkVJPicker는 그래픽스 인스턴스를 선택하는데 사용된다.
- vtkPicker 클래스에서 파생된 클래스로 3차원 공간에서 오브젝트를 선택하는 기능을 제공한다.
- vtkPicker는 picking 동작을 스크린에서의 위치에 기반하여 수행하는 반면, vtkVJPicker는 3차원 트래킹 디바이스의 위치와 레이(ray)의 방향 및 길이를 기반 정보로 오브젝트를 선택할 수 있게 해준다.
- 레이의 길이가 far clipping plane을 넘어갈 경우에는 오류가 발생할 수 있다. 적절한 길이 설정이 필요하고, scale에 따라 ray의 길이(두께)도 적절하게 설정하는 것이 중요하다.
- 리턴되는 정보: vtkPicker와 같다.

11) vtkVJPointPicker (vtkVJPointPicker.h)

- 그래픽스 인스턴스의 한 지점을 선택하는데 사용되는 클래스
- vtkPointPicker 클래스에서 파생된 클래스로 3차원 공간에서 한 지점을 선택하는 기능을 제공한다.

- vtkVJPicker와 동일한 방식으로 선택 작업을 수행한다.
- 리턴되는 정보: vtkPointPicker와 같다.

12) vtkVJCellPicker (vtkVJCellPicker.h)

- 그래픽스 인스턴스의 한 셀(cell)을 선택하는데 사용되는 클래스.
- vtkCellPicker 클래스에서 하생된 클래스로 3차원 공간에서 한 셀을 선택하는 기능을 제공한다.
- vtkVJpicker와 동일한 방식으로 선택 작업을 수행한다.
- 리턴되는 정보: vtkCellPicker와 같다.

13) vtkVJFactory (vtkVJFactory.h)

- class VTK_RENDERING_EXPORT vtkVJFactory : public vtkObjectFactory

14) vtkVJBoxWidget (vtkVJBoxWidget.h)

- 이 3차원 위젯은 6면체로 나타나는 관심 영역을 정의하는 데 사용된다.
- 이 오브젝트에서는 7개의 핸들을 생성하는데, 이 핸들은 마우스로 조작할 수도 있고, 수정할 수도 있다. 6개의 핸들은 각각의 면에 대응되며, 7번째 핸들은 육면체의 내부에 존재한다. 또, 바운딩 박스의 아웃라인도 나타나기 때문에 육면체의 각 면을 선택해서 오브젝트를 회전시키거나 확대하는 것도 가능하다.
- 이 위젯은 매우 유연성이 높아서 선택, cut, clip 등의 작업에 사용할 수 있으며, linear transform을 이용해서 오브젝트를 옮기는 데도 사용할 수 있다.
- 기본적으로는 StartInteractionEvent, InteractionEvent, EndInteractionEvent 이벤트를 이용해서 사용한다. InteractionEvent는 마우스 움직임에 의해 호출할 수 있으며, 다른 두 이벤트는 button down 및 button up에 의해 호출할 수 있다.
- class VTK_HYBRID_EXPORT vtkVJBoxWidget : public vtk3DWidget

15) vtkVJPlaneWidget (vtkVJPlaneWidget.h)

- vtkVJPlaneWidget은 씬에서 사용자가 상호작용 할 수 있는 평면을 정의하는 데 사용되는 위젯이다.
- 이 위젯에서는 4개의 핸들과 노말 벡터, 그리고 평면이 생성된다. 핸들은 평

면의 크기를 조절하는데, 노말 벡터는 평면을 회전시키는데 사용되며, 평면 자체를 픽킹해서 옮길 수도 있다.

- Set/GetResolution(), GetPolyData(), GetPlane()같은, VTK 오브젝트에서 제공하는 메소드를 제공한다.
- class VTK_HYBRID_EXPORT vtkVJPlaneWidget : public vtkPolyDataSourceWidget

16) VTK_OSGApp (VTK_OSGApp.h)

- VTK 애플리케이션을 캡슐화하는 클래스로, VTK를 기반으로 하는 애플리케이션이 파생될 기본 클래스를 정의한다.
- OpenGL Draw Manager를 사용한다.
- class VTK_OSGApp : public OsgApp, public VTKApp_mixin

17) VTK_OpenSGApp (vtk_OpenSGApp.h)

- VTK 애플리케이션을 캡슐화하는 클래스로, VTK를 기반으로 하는 애플리케이션이 파생될 기본 클래스를 정의한다.
- OpenSG Draw Manager를 사용한다.
- class VTK_OpenSGApp : public vrj::OpenSGApp, public VTKApp_mixin

지금까지 vjVTK의 각 클래스에 대해 알아봤다. vjVTK의 클래스 중 vtkVJWin32OpenGLRenderWindow 클래스는 win32 플랫폼과 관련된 렌더링 클래스로, 여기에서는 다루지 않겠다. 4장에서는 vjVTK를 이용한 예제를 통해 vjVTK를 실제적으로 어떻게 이용하는지에 대해 설명할 것이다.

4. vjVTK를 이용한 애플리케이션

4장에서는 vjVTK를 이용해서 애플리케이션을 구현해 봄으로써 vjVTK의 사용법에 대해 설명하고 VRJuggler와의 연동 관계에 대해서도 알아보기로 한다. 처음으로 소개할 애플리케이션은 뷰 프러스텀(View Frustum) 컬링과 관련된 애플리케이션이다.

가. ViewTester

1) 개요

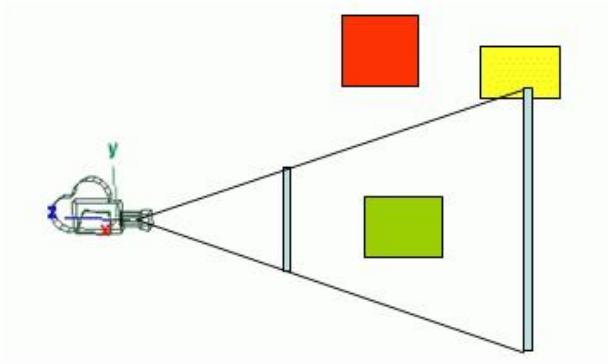
디스플레이 클러스터에서 VRJuggler가 실행될 경우, 모든 노드는 기본적으로 동일한 애플리케이션을 수행해야 하므로 해당 노드의 뷰 프러스텀에서 벗어나는 오브젝트도 함께 렌더링 처리가 된다. 이 때, 뷰 프러스텀에서 벗어나는 오브젝트를 미리 필터링해 두면 렌더링 과정에서 제외시킬 수 있으므로, 렌더링 처리 속도를 보다 높일 수 있다.

프러스텀 컬링(frustum culling)은 대상 오브젝트에 따라 필터링을 수행하는 방식이 달라지게 된다. 이 애플리케이션에서는 사각 평면(axis aligned box)에 대한 프러스텀 컬링 방식을 이용해서 구현했다.

우선, 간단히 VTK의 box 오브젝트를 생성한 다음, 애플리케이션이 디스플레이 클러스터에서 실행될 때, 각 노드가 가지는 뷰 프러스텀 안에 해당 오브젝트가 존재하는지 여부를 조사한다. 이 때, 프러스텀의 테스트 결과는 다음과 같은 형태로 제공된다.

- 오브젝트가 완전히 프러스텀 안에 존재하는 경우: inside
- 오브젝트가 완전히 프러스텀 밖으로 벗어나는 경우: outside
- 오브젝트와 프러스텀이 겹치는 부분이 존재하는 경우: intersect

프러스텀과 박스 오브젝트의 관계를 검사하는 방법은 프러스텀을 이루는 6개의 면에 대해서 박스를 이루는 네 점과이ㅡ 거리를 구하여 결정한다. 이 때, 프러스텀의 각 면에 대한 노멀 벡터(normal vector)는 프러스텀의 내부를 향하게 설정하며, 프러스텀의 각 면에 대해서 검사할 때 면의 4 점중에서 노멀 벡터와 같은 방향으로 존재하면서 가장 거리가 먼 점을 포함하는 벡터를 생성한다. 그런 다



[그림 4-1] 프러스텀과 박스 오브젝트간의 관계

음, 두 벡터의 내적(inner product)을 구해서 그 결과값이 양수이면 박스가 프러스텀의 안쪽, 음수이면 바깥쪽에 위치한다고 판단할 수 있다.

이 방식에 대한 메타 코드는 [소스 4-1]과 같다.

```
int FrustumG::boxInFrustum(Box &b) {

        int result = INSIDE, out,in;

        // for each plane do ...
        for(int i=0; i < 6; i++) {

                // reset counters for corners in and out
                out=0;in=0;
                // for each corner of the box do ...
                // get out of the cycle as soon as a box as corners
                // both inside and out of the frustum
                for (int k = 0; k < 8 && (in==0 || out==0); k++) {

                        // is the corner outside or inside
                        if (pl[i].distance(b.getVertex(k)) < 0)
                                out++;
                        else
                                in++;
                }
                //if all corners are out
                if (!in)
                        return (OUTSIDE);
                // if some corners are out and others are in
                else if (out)
```

```
                        result = INTERSECT;
        }
        return(result);

    }
```

[소스 4-1] 프러스텀 컬링 메타 코드

2) ViewTester 프로그램

VRJuggler와 VTK를 이용한 애플리케이션에서는 애플리케이션을 VTKApp 클래스의 하위 클래스로 선언하고 필요에 따라 initScene(), draw(), preFrame(), postFrame(), intraFrame(), latePreFrame() 등의 함수를 작성한다.

- initScene()
  - 프로그램이 초기에 도링될 때 실행되는 함수.
  - 지속적인 업데이트가 필요하거나 테스트를 필요로 하는 코드는 이 함수를 피하는 것이 좋다.
- pre/post/intra/latePreFrame()
  - preFrame() 함수는 시스템이 drawing 작업을 시작할 때 호출되는 메소드로, 이 때 애플리케이션 오브젝트는 입력 디바이스 상태에 따라 데이터에 대한 마지막 업데이트 작업을 수행한다. intraFrame() 함수는 시스템이 drawing 작업을 수행하는 도중에 수행되며, 주로 렌더링과 상관 없이 다음 프레임을 위한 연산을 수행해야 할 때 이용할 수 있다. postFrame() 함수는 drawing 작업이 끝난 뒤, 다음 프레임이 시작되기에 앞서 실행되므로, 입력 데이터에 종속적이지 않거나 렌더링 작업과 함께 수행할 수 없는 데이터 업데이트 작업을 수행하게 된다.또, 프레임이 렌더링된 뒤, 혹은 계산 작업이 완료된 뒤의 클리닝 업(cleaning up) 작업에도 사용할 수 있다. LatePreFrame() 함수는 preFrame 함수가 호출된 뒤, 그리고 클러스터 노드 사이에 공유되는 데이터가 동기화된 뒤, 씬이 렌더링되기 전에 호출된다.
  - VRJ의 디스플레이 환경 정보, 특히 vrj::Frustum에 관한 정보를 접근할 수 없게 구성돼 있으므로 이 함수들을 통해 분석, 접근하는 과정이 필요하다.
- draw()
  - 사용자 이벤트가 발생하거나 화면을 업데이트할 필요가 있을 때 자동으로 실행되는 함수로, VRJ 요소인 vrj::DrawManager, vrj::GlUserData, vrj::Pro

jection, vrj::Viewport, vrj::Frustum에 대한 접근이 가능하다.

ViwTester에 대한 클래스 선언은 다음 [소스 4-2]와 같다.

```
class ViewTester: public VTKApp
  {
    public:
      ViewTester(vrj::Kernel* kern) : VTKApp(kern), mVTKWorldXForm(NULL){}

      ViewTester(): VTKApp(){}

      virtual ~ViewTester(){ }
      ...
  }
```

[소스 4-2] ViewTester 클래스

이 때, draw() 함수는 다음과 같이 구성된다.

```
void ViewTester::draw()
   {
     VTKApp::draw();
     drawBoxVTK();
     checkViewPort();
   }
```

[소스 4-3] draw() 함수의 구성

VRJuggler의 프러스텀 정보를 얻는 과정은 다음과 같다. ([소스 4-4])

1. VRJuggler의 DrawManager를 얻는다.
2. GlUserData를 얻어온다.
3. GlUserData에 대한 프로젝션 정보를 가져온다.
4. 프로젝션 정보로부터 프러스텀 정보를 가져온다.

```
vrj::GlDrawManager *drawManager = (vrj::GlDrawManager*)getDrawManager();
```

```
assert(drawManager);
vrj::GlUserData *userData = drawManager->currentUserData();
assert(userData);
vrj::Projection *proj = userData->getProjection();
assert(proj);
vrj::Viewport *viewport = userData->getViewport();
assert(viewport);
vrj::Frustum frustum = proj->getFrustum();
```

[소스 4-4] 프러스텀 정보를 얻는 과정

VRJuggler가 제공하는 프러스텀 정보는 apex, near plane의 4개 코너, near distance, 그리고 far distance다. 따라서 far plane에 대한 정보는 이들 정보로부터 유추해낼 수 있다.

프러스텀의 near plane과 far plane에 대하여 모두 8개의 점이 구해지면 프러스텀의 6면을 모두 알 수 있다. 이 때, 면의 노멀 벡터가 프러스텀의 dksWHr을 향하도록 설정한다.

```
// Declare a corner and width, height, and depth for a box
Vec3 P1(-1, 2, -4); // minx, miny, minz
Vec3 P2(1, 4, -4);  // maxx, maxy, maxz

AABox box(P1, P2.x, P2.y, P2.z);

Vec3 ntr; // near upper right
Vec3 nbr; // near lower right
Vec3 ntl; // near upper left
Vec3 nbl; // near lower right

Vec3 ftr;  // far upper right
Vec3 fbr;  // far lower right
Vec3 ftl;  // far upper left
Vec3 fbl;  // far lower right

frustumG->pl[FrustumG::TOP].set3Points(ntr, ntl, ftl);
frustumG->pl[FrustumG::BOTTOM].set3Points(nbl, nbr, fbr);
frustumG->pl[FrustumG::LEFT].set3Points(ntl, nbl, fbl);
frustumG->pl[FrustumG::RIGHT].set3Points(nbr, ntr, fbr);
frustumG->pl[FrustumG::NEARP].set3Points(ntl, ntr, nbr);
```

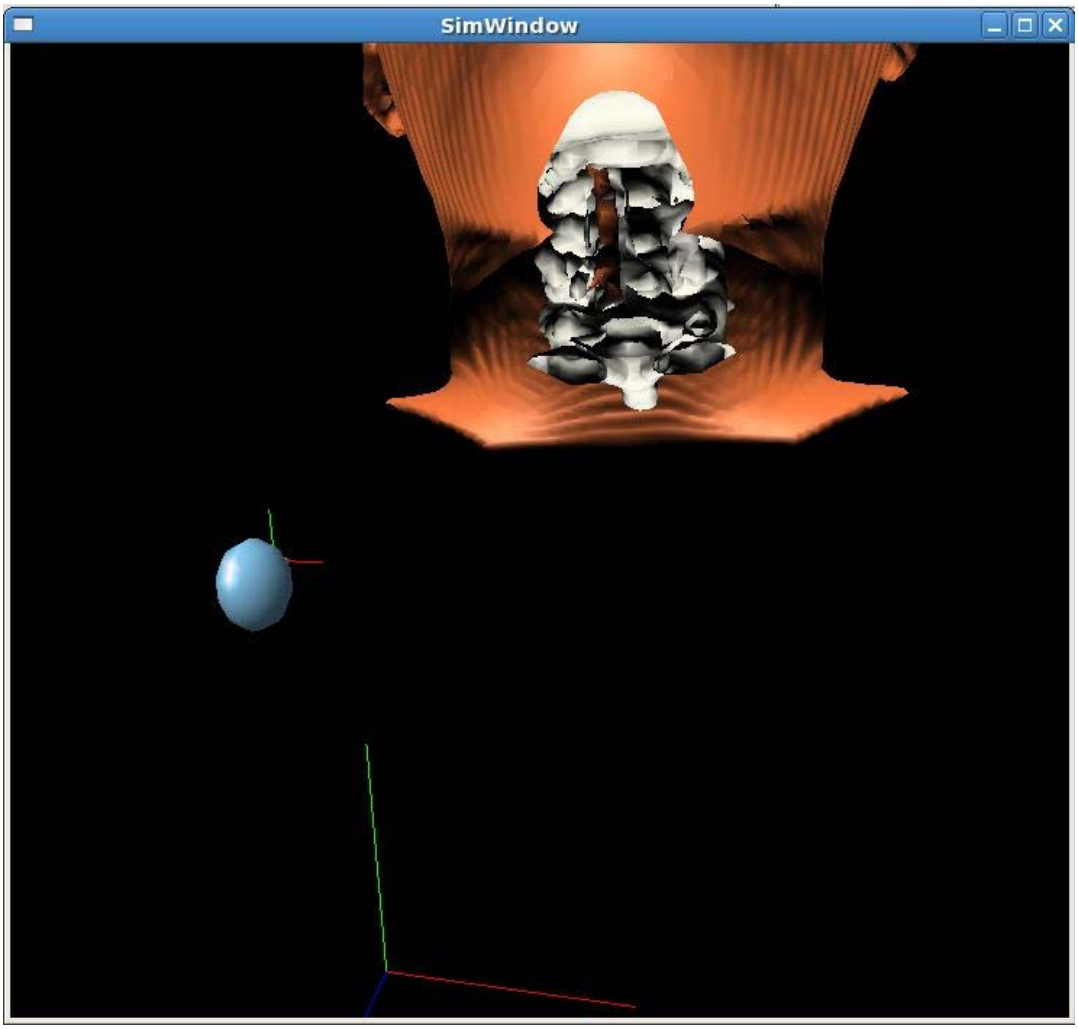
```
frustumG->pl[FrustumG::FARP].set3Points(ftr, ftl, fbl);

int result = frustumG->boxInFrustum(box);
```

[소스 4-5] 프러스텀의 계산

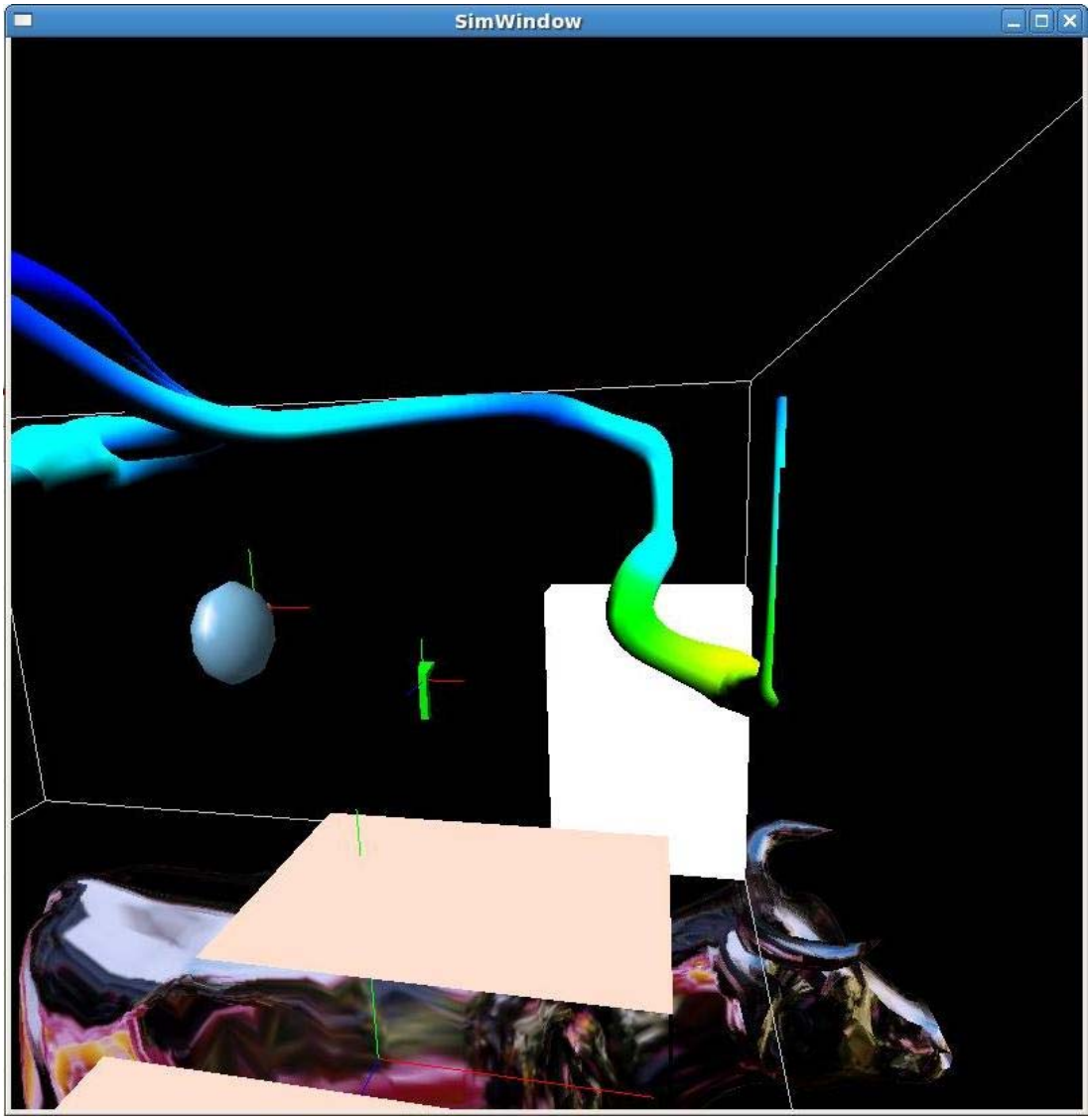
이런 방식으로 VRJuggler에서 각 노드에 주어지는 뷰 프러스텀을 계산할 수 있다.

나. vjVTK 예제

[그림 4-2]와 [그림 4-3]은 vjVTK를 이용해서 데이터를 가시화한 사례다. [그림 4-2]는 VTK의 Medical Example을 VRJuggler상에서 가시화한 것이고, [그림 4-3]은 OSG를 이용해서 VRJuggler에서 여러 다양한 VTK 데이터를 한번에 그려본 것이다.



[그림 4-2] vj-VTK를 이용해서 VRJuggler상에서 가시화한 VTK Medical Example



[그림 4-3] vjVTK를 이용한 VTK-OSG 사용 사례

5. 결론

vjVTK는 VR 환경에서 보다 쉽게 오픈-소스 기반의 가시화를 가능케 하려는 목적으로 설계된 툴킷으로, VRJuggler 가상현실 프레임워크에서 VTK 툴킷을 자연스럽게 사용할 수 있게 해주는 기능을 제공한다. 따라서 vjVTK를 이용하면 VTK로 구성한 가시화 데이터를 따로 수정하지 않고 가상현실 환경에서 3차원 입체 영상으로 분석할 수 있을 뿐만 아니라, 사용자의 상호작용을 통해 바로 파라미터를 입력해서 원하는 데이터를 추출해낼 수도 있다.

그러나 현재 vjVTK는 개발이 중단된 상태로, 개발이 계속 진행되고 있는 VTK나 VRJuggler의 버전-업과 관련된 기능 업그레이드를 따라잡지 못하고 있는 상황이다. 띠라서 현 버전의 VRJuggler 및 VTK의 기능을 살리는 것은 무리가 있으며, vjVTK도 자체적으로 수정을 해야 현 버전에서 그대로 사용이 가능하다.

본 문서에서는 vjVTK의 설치 방법에 대해 설명하면서, 이런 문제점을 해결하는 방법에 대해서도 설명했다. 그러나 본 문서에서 설명한 것은 임시 방편에 불과하며, 근본적으로는 vjVTK에 대한 추가 개발이 필요한 실정이다. 본 문서에서는 이에 대한 인식을 하고, 문제점을 파악하였으므로, 향후에 시간을 들여서 해결 방법을 모색한다면 근본적인 해결이 가능하게 될 것이다.

또, vjVTK의 각 클래스에 대해 설명하고, vjVTK를 이용한 몇가지 프로젝트와 예제 프로그램의 실행 결과를 설명했다.

vjVTK는 VTK 프로그램을 VRJuggler와 연동해서 가상현실 환경에서 가시화할 수 있게 해줌으로써, 과학 데이터를 보다 손쉽게 가시화할 수 있게 할 뿐만 아니라, 가상환경에서 사용자가 직접 제어하는 기능을 제공한다. 현재 개발이 중단된 부분에 대해 지속적인 보완이 필요하며, 향후에는 VRJuggler와 VTK의 기능 추가와 더불어 vjVTK에 대한 개발도 지속돼야 할 것이다.