ISBN 978-89-6211-391-4 98560


# GLOVE 시스템 개발을 위한 vjVTK의 확장

(Extension of vjVTK for GLOobal Virtual Environment for collaboration research)

김 민 아 (petimina@kisti.re.kr)

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

# 목차

# 그림 차례

# 소스 차례

# 1. 개 요

vjVTK는 VRJuggler의 Virtual Reality Framework 위에 Visualization Toolkit (VTK) 을 통합하는 하나의 toolkit이다.

VRJuggler 는 기본적으로 OpenGL에 근간해 가시화를 구현한다. VRJuggler 에서의 가시화는 OpenGL 프로그래머의 몫이다. 이것은 기본적인 가시화 기술을 모두 프로그래머가 구현해야 한다는 것을 의미한다. 이 때문에 VRJuggler 에서의 가시화는 offline에서 이루어진 다음 그 결과를 단순히 VR환경의 출력 장치에 뿌려주는 일을 수행한다.

VTK는 OpenGL의 기능을 최대한 활용하여 잘 알려진 기본적인 가시화 기능을 이미 구현된 API 로 제공하는 toolkit이다. 그러나 VTK는 VR 환경을 지원하는 입출력 장치인, wand, tracker, tiled-display 들의 입력을 받을 수 없고 또한 그 장치들에 출력을 할 수 없다.

vjVTK는 인터렉티브한 사용자 입력에 따라 가시화를 수행하기 위해 VRJuggler의 프레임워크 내에서 VTK를 사용할 수 있게 함으로써, VRJuggler에서 수준 높은 가시화를 쉽게 구현할 수 있게 해준다.

계산 과학이나 실험을 통해 생성된 대용량의 데이터를 가시화 하는 데는 고해상도의 tiled-display나 입체 영상 지원 등의 분석하기 좋은 환경이 매우 중요한 화두로 떠오르고 있다. 이러한 응용 분야에 있어 vjVTK는 유용한 toolkit 이 될 것이다. 그러나 vjVTK는 안정화 되어 있지 않으며, 대학 연구실 수준의 프로젝트로 VTK 5.0에 최적화 되어 VTK의 버전이 업데이트 되는 데에 따른 업그레이드도 일어나고 있지 않다. 본 문서에서는 vjVTK의 기본 구조에 대해 살펴보고, GLOVE 시스템 개발에서 vjVTK를 사용함으로써 발견한 vjVTK 의 근본적인 오류와 VTK 5.2.1 상위 버전과 함께 사용했을 때의 문제점에 대해 기술한다. 또한, 이들 문제를 해결한 방법과 필요 기능을 위한 확장 개발의 범위와 방법에 대해 기술한다.

# 2. VRJuggler와 vjVTK의 동작 구조

1절에서 언급한 바와 같이 VRJuggler 는 tiled-display와 VR환경의 입출력 장치들을 사용할 수 있는 Framework를 제공한다. 본 절에서는 VRJuggler 상에서 vjVTK 가 어떻게 동작하는지 살펴본다.

# 가. VRJuggler 의 동작 시나리오

## 1) main() function의 구조

VRJuggler의 main() 함수는 아래와 같이 구성된다. simpleApp,h 내에 선언된 simpleApp 클래스가 실제 응용이 구현된 클래스이다. VRJuggler 는 Kernel 클래스내에 루핑을 통해 VR환경의 입출력 장치들로부터 입력을 받고 이에 따른 처리를 수행한다. loadConfigFile() 함수의 인자는 사용자 입력으로 받아들여진 config 파일의 이름이다. config 파일의 자세한 구조와 내용은 [5]를 참조한다.

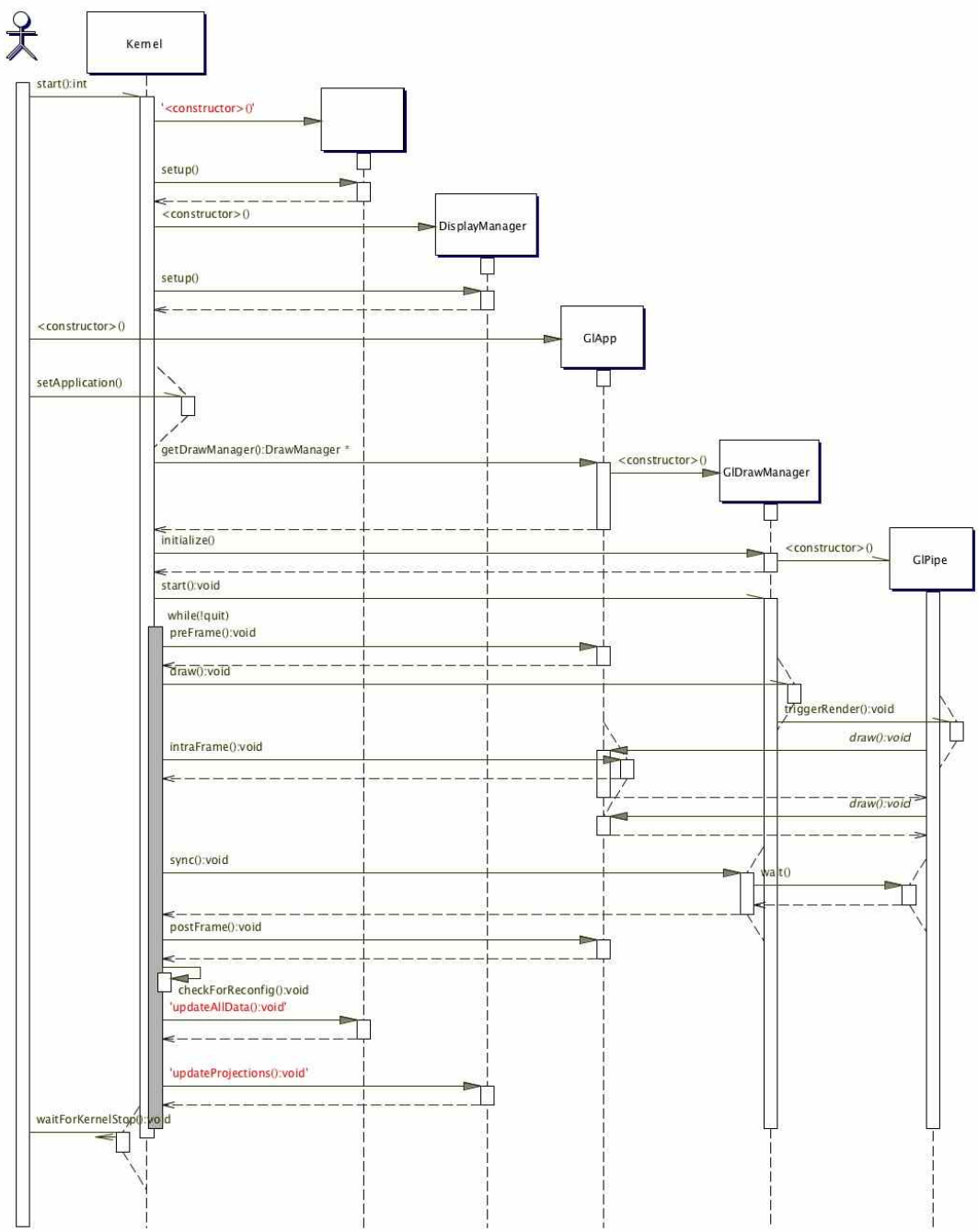
```
#include <iostream>
#include <cstdlib>
#include <vrj/Kernel/Kernel.h>
#include <simpleApp.h>

int main(int argc, char* argv[])
{
   if ( argc <= 1 )
   {
      std::cout << "Usage: " << argv[0]
                << "cfgfile[0] cfgfile[1] ... cfgfile[n]"
                << std::endl;
      std::exit(EXIT_FAILURE);
   }

   vrj::Kernel* kernel = vrj::Kernel::instance(); // Get the kernel
   simpleApp* app      = new simpleApp();         // Create the app object

   kernel->loadConfigFile(...);              // Configure the kernel
   kernel->start();                          // Start the kernel thread
   kernel->setApplication(app);              // Give application to kernel
   kernel->waitForKernelStop();              // Block until kernel stops

   return 0;
}
```

[소스 1-1 VRJuggler main() 예제]

kernel을 start 하고, application을 새로 생성한 simpleApp 클래스인 인스턴스인 app 로 setApplication 하면, VRJuggler의 framework 내에서 구현된 응용이 동

작한다. Kernel 클래스의 인스턴스인 kernel이 start 함수를 부르면, 일련의 Kn ernel loop sequence 가 수행된다. main() 함수 내의 kernel 인스턴스는 waitFor KernelStop 으로 다른 스레드들이 종료하기를 기다린다.

## 2) VRJuggler Kernel Sequence




[그림 1-1 VRJuggler Kernel Loop Sequence]

VRJuggler는 멀티 스레드로 동작한다. Sequence diagram의 Kernel, DisplayMan

ager, GlApp, GlDrawManager, GlPipe들은 스레드로 존재하며, 상호 함수 호출과 동기화에 의해 동작한다.

Kernel 이 start 한 후에 while 내에서 Application은 preFrame(), draw(), intra Frame(), postFrame()을 반복한다. Application 의 실질적인 동작은 preFrame (), draw(), intraFrame(), postFrame() 내에 구현한다.

### 3) VRJuggler 내에서 vjVTK의 동작

vjVTK는 VTKApp 라는 VTK Application을 위한 베이스 클래스를 제공한다. V TKApp는 GlApp와 VTKApp_mixin 클래스를 상속받는다.

GlApp는 VRJuggler가 제공하는 클래스로 OpenGL-based application object의 기본 클래스이다. kernel 과 DrawManager는 제어 루프에서 GlApp의 멤버 함수를 호출하여 GlApp의 기능을 수행한다. 아래 코드는 Kernel 루프 상에서 GlApp가 수행될 경우의 순서를 보여준다. 실제 수행은 kernel 과 DrawManager에 의해 수행 되겠지만, GlApp의 입장에서는 아래코드의 순서로 보일 것이다.

```
glapp_obj->contextInit();          // called for each context
 while (drawing)
 {
    glapp_obj->preFrame();
    glapp_obj->latePreFrame();
    glapp_obj->bufferPreDraw();    // called for each draw buffer
    glapp_obj->contextPreDraw();   // called for each context
    glapp_obj->draw();             // called for each viewport
    glapp_obj->contextPostDraw(); // called for each context
    glapp_obj->intraFrame();       // called in parallel to the draw functions
    sync();
    glapp_obj->postFrame();
    updateAllDevices();
 }
 glapp_obj->contextClose();        // called for each context
```

[ 소스 1-2 GlApp Pseudo Code]

VTKApp_mixin은 VTK Application을 둘러싸는 클래스로 VTK의 렌더링을 제외한 부분을 다룬다. 다음은 vjVTK의 VTKApp_mixin의 정의부분이다.

```
class VTKApp_mixin
{
  public:
   struct context_data
   {
         vtkVJOpenGLCamera* mVTKCamera;
         vtkVJRenderWindow* mVTKWindow;
         vtkRenderer * mVTKRenderer;
   };

   VTKApp_mixin()
   {
      mVJRenderer = new vjRenderer();

      vtkVJFactory* vj_factory = vtkVJFactory::New();
      vtkObjectFactory::RegisterFactory(vj_factory);
      vj_factory->Delete();

   }
   virtual ~VTKApp_mixin()
   {
      delete mVJRenderer;
   }
   virtual void draw();
   void contextInit();
   virtual void contextClose();
   virtual void contextPreDraw();
   virtual void preFrame();
   //Return the vjRenderer object
   vjRenderer* getRenderer(){return mVJRenderer;}

protected:
   vrj::GlContextData< context_data > mVTKContextData;
   vjRenderer* mVJRenderer;
};
```

[소스 1-3 VTKApp_mixin Class Definition]

VTKApp_mixin은 하나의 새로운 context를 가진다. 그 context 내에는 VTK의 context인 VTKCamera와 vtkVJRenderWindow, 그리고, vtkRenderer를 포함한다. 또한, context내의 vtkRenderer와는 별개로 vjRender를 멤버 변수로 가지고 있다.

vjRenderer는 vjVTK가 제공하는 renderer로, vtkRenderer의 속성 중, vtkPropCollection, vtkActorCollection, vtkVolumeCollection, vtkLightCollection 만을 관리한다. 이에 더하여 자신만의 멤버로 RendererCommand의 리스트를 관리한다.

vjVTK는 이 두 개의 서로 다른 context, VRJuggler와 VTK를 contextPreDraw() 함수 내에서 동기화 한다. VTKApp_mixin 클래스를 상속받은 application은 vtkRenderer에 vtkActor를 등록하듯 vjRenderer에 vtkActor를 등록한다. 이렇게 등록된 VRJuggler의 프레임워크 내에서 등록된 vtkActor들은 contextPreDraw() 함수에서 vjRenderer의 updateRenderer() 함수에 의해 모두 vtkRenderer로 옮겨진다.  다음은 contextPreDraw()와 updateRenderer()의 코드이다.

```
 void VTKApp_mixin::contextPreDraw()
{
   // If there is anything new to add/remove from the scene
   // do it now... :)
   mVJRenderer->updateRenderer(mVTKContextData->mVTKRenderer);

}
void vjRenderer::updateRenderer(vtkRenderer* renderer)
{
   for(std::vector<RendererCommand*>::iterator it = mCommandList.begin();
       it != mCommandList.end(); it++)
   {
      (*it)->execute(renderer);
   }
}
```

[소스 1-4 Render Command Execution in vtkRenderer]

vjRenderer의 updateRenderer() 함수에서는 vjRenderer에 등록된 모든 vtk 명령들을 다시 vtkRenderer로 등록하는 일을 수행한다. RenderCommand를 상속받은 PopCommand의 execute 함수를 보면 명령어와 execute 함수의 의미를 알 수 있다. 다음은 PropCommand의 execute 함수 코드이다.

```
void PropCommand::execute(vtkRenderer* renderer)
{
   switch(mCommand)
   {
      // Both ACTOR and VOLUME are wrappers for the AddViewProp
      // command in the renderer... so all handled the same
      case ADD_PROP:
      case ADD_ACTOR:
      case ADD_VOLUME:
         renderer->AddViewProp(mProp);
         break;

      // Actor and Volume do more than Prop...
      // so we have to call them individually
      case REMOVE_PROP:
         renderer->RemoveViewProp(mProp);
         break;
      case REMOVE_ACTOR:
         renderer->RemoveActor(mProp);
         break;
      case REMOVE_VOLUME:
         renderer->RemoveVolume(mProp);
         break;
   }
}
```

[소스 1-5 Render Command Conversion ]

# 3. vjVTK의 오류와 수정

vjVTK는 2005년에 개발되어 0.8까지의 버전을 내놓는 동안 VTK 5.0 과 Windows 시스템을 지원하게 되었다. 그러나, vjVTK의 0.8 버전을 VTK 5.2.2에 적용하면서, vjVTK에는 compile time 오류부터, run time 오류, 논리적 오류까지 많은 오류들이 발견되었다. 여기서는 run time 오류와 논리적 버그 수정에 대한 부분만 다루기로 한다.

## 가. Runtime 오류와 해결책

VTK 5.2.2 와 함께 vjVTK로 작성한 프로그램을 실행하면, Segmentation fault

에러가 나면서 프로그램이 종료된다. 이는 vjVTK에서 vtkOpenGLRenderer와 vtkOpengLRenderWindow 클래스의 대체 클래스만을 구현하고 vtkOpenGLActor나 vtkOpenGLProperty 등을 구현하지 않아 OpenGL 용 클래스를 그대로 가져다 쓰기 때문에 발생한 문제이다. 실제로 gdb를 이용해서 프로그램을 trace했을 때 에러가 발생하는 부분은 다음과 같다.

```
1 void vtkOpenGLProperty::Render(vtkActor *anActor,
2                                 vtkRenderer *ren)
3 {
4   int i;
5   GLenum method;
6   float Info[4];
7   GLenum Face;
8   double  color[4];
9
10  // unbind any textures for starters
11  vtkOpenGLRenderer *oRenderer=static_cast<vtkOpenGLRenderer *>(ren);
12
13  if(oRenderer->GetDepthPeelingHigherLayer())
14  {
15    GLint uUseTexture=-1;
16    uUseTexture=oRenderer->GetUseTextureUniformVariable();
17    vtkgl::Uniform1i(uUseTexture,0);
18  }
```

[소스 2-1 Code that cause Segmentation Fault]

여기서 Render() 함수에 인자로 들어오는 vtkRenderer *ren의 실제 type은 vtkVJOpenGLRenderer 이다. 그런데, 11번째 라인에서 이것을 vtkOpenGLRenderer로 강제 type casting한다. 이는 실제 이 클래스가 vtkOpenGLProperty이기 때문에 받아오는 renderer도 vtkOpenGLRenderer라고 가정하였기 때문이다. 그러나, vjVTK에서 vtkVJOpenGLProperty를 구현하지 않고 vtkOpenGLProperty를 빌려 사용한다. 코드에서 강제 casting한 renderer가 GetDepthPeelingHigherLayer() 함수를 호출하면 그 리턴 값에 따라 조건문의 분기가 결정된다. 이 함수는 vtkOpenGLRenderer의 멤버 변수인 DepthPeelingHigherLayer 값을 돌려주는 함수로 vtkOpenGLRenderer를 처음 생성할 때 0으로 초기화 된다. 그러나 실제 renderer인 vtkVJOpenGLRenderer는 이 값을 초기화하지 않으므로 실행할 때 어떤 값이 들어가 있을지 알 수 없다. (vtkVJOpenGLRenderer에는 이 변수가 정의되어 있

지도 않기 때문에 어떤 값을 access할 지 알 수 없다.) 따라서 우연히 이 값이 0인 경우는 조건문 안으로 들어가지 않게 되고 0이 아닌 경우는 조건문 안으로 들어가게 된다. 조건문 안으로 들어가게 되면, vtkOpenGLRenderer에서는 depth peeling을 위한 shader를 생성한다. 이 조건문 안에서는 shader의 UseTexture라는 uniform variable을 접근한다. 그런데 vtkVJOpenGLRenderer에서는 이러한 shader와 관련된 어떤 정의도 존재하지 않으므로, 접근할 shader가 없기 때문에 segmentation fault 에러가 발생하게 된다.

이러한 문제를 해결하기 위한 가장 간단한 방법은 소스 코드 중 13번째 라인부터 18번째 라인을 코멘트 처리하는 것이다. 그러나, 이럴 경우 depth peeling 기능을 수행할 수 없게 된다. 따라서, 추가된 기능인 depth peeling을 vjVTK에서만, 사용하지 않도록 type cast 전에 class name을 check해서 vtkVJOpenGLRenderer 일 경우, 13~18 라인을 수행하지 않도록 수정하였다.

## 나. vjRenderer의 논리적 오류와 해결책

2절에서 이미 기술한 바와 같이, vjVTK는 vjRenderer와 vtkRender를 함께 가지고 있다가, contextPreDraw() 함수를 호출할 때 vjRenderer에 등록된 내용을 vtkRenderer에 반영한다. 다시 한 번 updateRender() 함수를 살펴보면, updateRenderer() 함수의 논리적 오류를 바로 발견할 수 있다.

```
void vjRenderer::updateRenderer(vtkRenderer* renderer)
{
   for(std::vector<RendererCommand*>::iterator it = mCommandList.begin();
       it != mCommandList.end(); it++)
   {
      (*it)->execute(renderer);
   }
}
```

[소스 2-2 vjRenderer::updateRenderer()]

updateRenderer() 함수가 최초에 수행되었을 때는 별 문제가 없다. 그러나, 두 번 수행되었다고 생각해 보자. 이 코드의 어디에도 실행한 명령어를 지우는 부분은 존재 하지 않는다. 즉 두 번째 updateRenderer 함수를 실행하면, 이전의 명령어를 그대로 다시 한 번 더 실행하게 되는 것이다. 만일 명령어 중에 AddActor 존재 한다면, 이것은 시스템의 메모리가 충분할 때까지 vtkRenderer에 계속 vtkActor가 추가되어 메모리가 증가할 것이다. 또한, DeleteActor가 추가되면, 이미 존재하지 않는 vtkActor를 한 번 더 지우게 된다. 이 때문에, DeleteActor()로 v

tkActor를 지울 때 마다 segmentation fault가 발생한다. 이것은 매우 간단하게 해결할 수 있다. 아래와 같이 명령을 다 수행한 후 이들을 command list를 비우면 된다.

```
void vjRenderer::updateRenderer(vtkRenderer* renderer)
{
   for(std::vector<RendererCommand*>::iterator it = mCommandList.begin();
       it != mCommandList.end(); it++)
   {
      (*it)->execute(renderer);
   }

   // delete commads that already was performed
   mCommandList.clear();
}
```

[소스 2-3 Modified vjRenderer::updateRenderer()]

# 4. vjVTK 확장 개발

vjVTK가 제공하는 기능은 매우 한정적이다. 이 중 vtkVJInteractor의 기능은 VR Juggler의 VR환경 입력장치의 이벤트를 처리하는 기본 프레임워크만 가지고 있으며, 많은 부분이 프로그래머에 의해 재 구현 되어야 한다. 또한 애니메이션을 구현할 경우, VTK의 vtkAnimationCue이벤트 처리와 연결이 되지 않아 애니메이션 중에 사용자의 입력을 받는 것이 불가능하다. 본 절에서는 이러한 vjVTK의 기능 확장을 위한 개발 내용에 대해 기술한다.

## 가. vtkVJInteractor 확장 개발

vtkVJInteractor는 picking과 frame rate 제어를 포함한 interaction을 제공하는 vtkRenderWindowInteractor를 상속받는다. 그러나, 이벤트를 처리하고 이벤트의 발생을 감지하는 vtkInteractorObserver는 NULL이다. vtkVJInteractor는 단지 vtkRenderWindowInteractor의 형태를 가장하고 있을 뿐 모든 이벤트의 처리는 VR Juggler가 제공하는 VR 입력 장치의 변화를 감지하여 직접 처리해 주어야 한다. 본 문서에서는 wand를 입력 장치로 하는 vtkVJInteractor의 확장 개발을 기술한다.

### 1) vtkVJInteractor의 이벤트 처리 동작

vjVTK 0.8에서 vtkVJInteractor의 이벤트 처리 동작을 예제를 통해 살펴 보자. 다음은 testVTKApp 클래스를 정의한 부분 이다.

```
class testVTKApp: public VTKApp
{
  public:
      testVTKApp(vrj::Kernel* kern) : VTKApp(kern), mActor(NULL),
                                      mVJInteractor(NULL)
   {
   }
   testVTKApp(): VTKApp(), mVJInteractor(NULL)
   {
      ;
   }
    virtual ~testVTKApp()
    {
       ;
    }
    void init();
    void initScene();
    void contextInit();
    void preFrame();

  private:
    vtkActor* mActor;
    vtkVJInteractor* mVJInteractor;

   gadget::PositionInterface mWand;
   gadget::DigitalInterface  mLeftButton;       /**< Button to go forward */
   gadget::DigitalInterface  mMiddleButton;        /**< Button to rotate */

   gadget::DigitalInterface  mRightButton;
   gadget::DigitalInterface  mDumpStateButton;
};
```

[소스 3-1 testVTKApp Class Definition]

먼저 VTKApp를 상속받는 testApp의 initScene() 함수에서(13 라인) vtkVJInteractor를 만든다. 14라인에서 vtkVJInteractor를 사용하고자 하는 vtkVJBoxWidget에 interactor로 새로이 만든 mVJInteractor를 등록한다. 20라인에서 이벤트

가 발생했을 때, 동작할 callback함수를 생성하고, 21라인에서 AddObserver 함수로 이벤트와 callback을 함께 등록한다. InteractionEvent는 입력장치에서 발생한 모든 이벤트에 대해 동시에 발생하는, 즉 interaction이 있기만 하면 발생하는 이벤트이다.

```
void testVTKApp::init()
{
   VTKApp::init();
   mLeftButton.init("VJButton0");
   mMiddleButton.init("VJButton1");
   mRightButton.init("VJButton2");
   mWand.init("VJWand");
}
void testVTKApp::initScene()
{
      vtkConeSource *cone = vtkConeSource::New();
      cone->SetHeight( 3.0 );
      cone->SetRadius( 1.0 );
      cone->SetResolution( 10 );

      vtkPolyDataMapper *coneMapper = vtkPolyDataMapper::New();
      coneMapper->SetInput( cone->GetOutput() );

      mActor = vtkActor::New();
      mActor->SetMapper( coneMapper );
      mActor->SetPosition(0.0, 5.0, -2.0);
      mActor->SetScale(4,4,4);
      mVJRenderer->AddActor(mActor);

 13   mVJInteractor = vtkVJInteractor::New();
      vtkVJBoxWidget *boxWidget = vtkVJBoxWidget::New();
 14   boxWidget->SetInteractor(mVJInteractor);
      boxWidget->SetVJRenderer(mVJRenderer);
      boxWidget->SetPlaceFactor(1.25);

      boxWidget->SetProp3D(mActor);
      boxWidget->PlaceWidget();
[ 20  vtkMyCallback *callback = vtkMyCallback::New();
 21   boxWidget->AddObserver(vtkCommand::InteractionEvent, callback);

      boxWidget->On();
}
```

[소스 3-2 Event and Callback Registration using vtkVJInteractor]

testVTKApp는 vtkVJInteractor의 이벤트 처리동작을 수행하기 위해 다음의 함수를 preFrame() 함수에서 수행하여야 한다.

```
void testVTKApp::preFrame()
{
   VTKApp::preFrame();
   mVJInteractor->SetSelectorInformation(mWand);
   mVJInteractor->SetButtonInformation(0, mLeftButton);
   mVJInteractor->SetButtonInformation(1, mMiddleButton);
   mVJInteractor->SetButtonInformation(2, mRightButton);
}
```

[소스 3-3 Check Device Information Change in testVTKApp::preFrame()]

2절에서 보았듯이 preFrame()은 kernel loop에서 draw()를 하기 전 가장 먼저 수행되는 함수이다. 따라서, 입력장치의 변화는 preFrame()에서 감지하여야 한다 .

```
void vtkVJInteractor::SetButtonInformation(const int button_number, gadget::DigitalInterface
& button)
{
   if (!this->Enabled) return;
   switch (button_number)
   {
      case 0:
         if(gadget::Digital::TOGGLE_ON == button->getData()||
            gadget::Digital::ON == button->getData())
            this->InvokeEvent(vtkCommand::LeftButtonPressEvent,NULL);
         else if(gadget::Digital::TOGGLE_OFF == button->getData())
            this->InvokeEvent(vtkCommand::LeftButtonReleaseEvent,NULL);
         break;
      case 1:
         if(gadget::Digital::TOGGLE_ON == button->getData())
            this->InvokeEvent(vtkCommand::MiddleButtonPressEvent,NULL);
         else if(gadget::Digital::TOGGLE_OFF == button->getData())
           this->InvokeEvent(vtkCommand::MiddleButtonReleaseEvent,NULL);
         break;
      case 2:
         if(gadget::Digital::TOGGLE_ON == button->getData())
            this->InvokeEvent(vtkCommand::RightButtonPressEvent,NULL);
         else if(gadget::Digital::TOGGLE_OFF == button->getData())
           this->InvokeEvent(vtkCommand::RightButtonReleaseEvent,NULL);
   }
}
```

[소스 3-4 InvokeEvent in SetButtonInformation()]

vtkVJInterctor의 SetInformation()은 wand의 세 버튼 mLeftButton, mMiddleButton, mRightButton에 변화가 있는지 체크하는 함수이다. vtkVJInteractor에서는 SetInformation 함수에서 감지된 변화를 InvokeEvent()를 호출함으로써, VTK의 이벤트로 변환한다.

### 2) vtkVJInteractor의 이벤트 포워딩을 위한 확장 개발

vjVTK 0.8에서 위의 예제 프로그램은 동작하지 않는다. vtkVJInteractor가 발생한 이벤트를 boxWidget까지 전달하지 않기 때문이다. 즉, 이벤트는 InvokeEvent 함수를 통해 vtkVJInteractor에만 전달된 것이다.

메뉴를 구성하고, 그 메뉴를 선택하였을 때, 어떤 콜백 함수가 동작하기를 원할 경우, vtkVJInteractor는 picking한 오브젝트를 알아야 하고, 그 오브젝트에 이벤트를 포워딩 해야 한다. 따라서, SetInformation()에서는 InvokeEvent를 정확히 그 오브젝트에 전달해야 한다. 이를 수행하기 위해 먼저 picking이 있었는지 알아야 하고, picking이 있었다면, 정확히 어느 오브젝트인지 알아야 한다. 이 코드에서는  wand의 오른쪽 버튼을 picking을 위해 사용하고 있으므로, wand의 오른쪽 버튼이 눌러 졌을 때, RightButtonPressEvent를 picking 한 오브젝트로 전달한다. 아래 코드는 picking 한 오브젝트로 이벤트를 포워드하는 함수 ForwardPickEvent()함수이다.

```
void vtkVJInteractor::ForwardPickEvent()
{
   if (mPicker->GetPickFromList())
   {
      vtkActor* pickedActor = mPicker->GetActor();
      if (pickedActor != NULL)
      {
         std::cout << "Invoke Pick Event1 : " << pickedActor << std::endl;
         pickedActor->InvokeEvent(vtkCommand::RightButtonPressEvent, NULL);
      }
   }
}
```

[소스 3-5 ForwardPickEvent in vtkVJInteractor]

GetPickFromList의 값이 1인 경우는 picking list에 등록한 actor 들에 대해서만 pick을 event를 전달하는 것이므로 GetPickFromList()의 값이 1인 경우에만 이벤트를 포워드한다. vtkVJPicker 타입의 mPicker로부터 pick한 오브젝트를 얻어온 다음 이 actor의 InvokeEvent 함수를 부르면 actor로 이벤트가 전달된다.

```
void vtkVJInteractor::SetButtonInformation(const int button_number, gadget::DigitalInterface& button)
{
   switch (button_number)
   {
      case 0:
         if(gadget::Digital::TOGGLE_ON == button->getData()||
           gadget::Digital::ON == button->getData())
         {
            mPicker->Pick(mWand->GetWandPosition(), mWand->GetWandEnd(),
                          mRenderer);
            if (this->IsPick())
            {
               this->ForwardPickAndMouseMoveEvent();
            }
            this->InvokeEvent(vtkCommand::LeftButtonPressEvent,NULL);
         }
         else if(gadget::Digital::TOGGLE_OFF == button->getData())
         {
            this->InvokeEvent(vtkCommand::LeftButtonReleaseEvent,NULL);
         }
         break;
      case 1:
         if(gadget::Digital::TOGGLE_ON == button->getData())
         {
             mPicker->Pick(mWand->GetWandPosition(), mWand->GetWandEnd(),
                           mRenderer);
             this->InvokeEvent(vtkCommand::MiddleButtonPressEvent,NULL);
         }
         else if(gadget::Digital::TOGGLE_OFF == button->getData())
         {
              this->InvokeEvent(vtkCommand::MiddleButtonReleaseEvent,NULL);
         }
         break;
      case 2:
          if(gadget::Digital::TOGGLE_ON == button->getData())
         {
            mPicker->Pick(mWand->GetWandPosition(), mWand->GetWandEnd(),
                          mRenderer);
            if (this->IsPick())
            {
               this->ForwardPickEvent();
            }
            this->InvokeEvent(vtkCommand::RightButtonPressEvent,NULL);
          }
          else if(gadget::Digital::TOGGLE_OFF == button->getData())
          {
                this->InvokeEvent(vtkCommand::RightButtonReleaseEvent,NULL);
         }
   }
}
```

[소스 3-6 Modified Event Invoking and Event Forwarding]

ForwardPickEvent()는 SetInformation() 함수 내에서, case 2 즉 wand의 오른쪽 버튼이 눌러졌을 때, IsPick()이 true이면, 즉 오른쪽 버튼이 눌러졌을 때 그 위치에 특정 오브젝트가 있었다면, 이벤트를 포워드하기 위해 호출된다.

## 나. 애니메이션을 위한 vjVTK 클래스 개발

VTK는 애니메이션을 위해 vtkAnimationCue를 제공한다. 그러나, vtkAnimationCue는 애니메이션 동안 어떤 사용자 입력도 받지 않는다. 이 절에서는 VRJuggler의 프레임워크 내에서 애니메이션 중에 사용자 입력을 받을 수 있는 vjAnimation 클래스의 개발과 그 동작에 대해 기술 한다.

### 1) vjAnimationList

```
class vjAnimationList
{
public:
    vjAnimationList();
    ~vjAnimationList();
    int   AddAnimation(void(*f)(vtkObject *caller, unsigned long eid,
                    void *clientdata, void *calldata),void *clientdata,
                    int nTimeStep, bool* isNew);
    int   AddAnimation(vtkCallbackCommand* aniCallback, int nStartTimeStep,
                        int nEndTimeStep);
    void DeleteAnimation(int nRegIndex);
    void StartAnimation(int nRegIndex);
    void StopAnimation(int nRegIndex);
    bool IsContinueAnimation(int nRegIndex);
    int   GetAnimationActorNum();
    void SetObserver(unsigned long observerId);
    vjAnimationActor*   GetCurrentAnimation();
    vjAnimationActor*   GetAnimationActor(int nRegIndex);
private:
    vjAnimationActor*           mAniActorList[MAX_CB_NUM];
    int                         mAniNum;
    vjAnimationActor*           mCurrentAnimation;
};
```

[소스 3-7 vjAnimation Class Definition]

vjAmimationList는 사용자가 등록한 vjAnimationActor의 리스트를 관리한다. vtkVJInteractor는 이 리스트에 있는 모든 vjAnimationActor에 애니메이션을 위한

이벤트를 전달한다. AddAnimation은 애니메이션을 수행할 콜백 함수와 전체 time step을 입력으로 하나의 vjAnimationActor를 생성하고, 결과 값으로 vjAnimationList에서의 index를 돌려준다. index는 리스트에서 vjAnimationActor를 삭제할 때와 vjAnimationActor를 얻어올 때 필요한 값이므로 생성 후 관리해야 한다.

또한, 이 index로 vjAnimationList에 index 번째에 있는 특정 vjAnimationActor를 start하고 stop 할 수 있다.

### 2) vjAnimationActor

```
class vjAnimationActor
{
public:
    vjAnimationActor();
    ~vjAnimationActor();
    void StartAnimation();
    void StartAnimation(int startTimeStep, int endTimeStep);
    void ProceedNextScene();
    void StopAnimation();
    bool IsContinueAnimation();
    bool IsOnPlay();
    int  GetCurrentStep();
    int  GetTimeStep();
    void SetTimeStep(int timeStep);
    void SetStartTimeStep(int nTimeStep);
    int  GetStartTimeStep();
    void SetIndex(int index);
    void SetCallback(vtkCallbackCommand* callback);
    void SetObserver(unsigned long observerId);
    unsigned long GetObserver();
    vtkCallbackCommand*  GetCallback();
private:
    vtkCallbackCommand* mCallback;
    int mStartTimeStep;
    int mCurrentTimeStep;
    int mEndTimeStep;
    int mMyIndex;
    bool mIsOnPlay;
    unsigned long mEventObserver;
};
```

[소스 3-8 vjAnimationActor Class Definition]

vjAnimationActor는 애니메이션을 위한 time step과 콜백 함수를 관리한다. StartAnimation()함수로 애니메이션을 시작할 수 있다. StartAnimation()의 패러메터

로 애니메이션의 time step을 선택할 수 있다. StopAnimation()은 애니메이션을 멈춘다.

### 3) VRJuggler 프레임워크 내에서의 애니메이션의 동작

VRJuggler의 프레임워크 내에서 사용자 입력을 받는 동시에 애니메이션을 진행하기 위해 우리는 다음의 방법을 사용한다. VRJuggler의 kernel loop을 한번 수행할 때, 애니메이션을 위한 한 time step의 scene을 만든다. 이럴 경우, preFrame에서 사용자 입력을 check 한 다음 draw()에서 만들어진 한 time step의 scene을 그리고 다시 다음 loop에서 preFrame() 수행하기 때문에, 사용자의 입력을 받는 동시에 애니메이션을 진행할 수 있다.

애니메이션 이벤트의 전달을 위해 vtkVJInteractor에도 수정이 필요하다. VRJuggler의 프레임워크 내에서 사용자 입력을 받는 동시에 애니메이션을 진행하기 위해 VTKApp의 preFrame() 함수 내에서 입력장치의 변화 감지를 수행하는 checkChangeInformation() 내 Animation에 관한 다음 코드를 추가한다.

```
void vtkVJInteractor::checkChangeInformation()
{
   this->SetSelectorInformation(mWand->GetPositionInterface());
   this->SetButtonInformation(0, mLeftButton);
   this->SetButtonInformation(1, mMiddleButton);
   this->SetButtonInformation(2, mRightButton);

   if (mIsAnimation == true)
   {
      ForwardAnimationEvent();
   }
}
```

[소스 3-9 Animation Event Forwarding]

만일 등록된 애니메이션이 하나라도 있을 경우 mIsAnimation은 true 이므로 애니메이션 이벤트를 포워드 한다.

다음은 ForwardAnimationEvent() 함수이다. ForwardAnimationEvent함수에서 포워드 되는 이벤트는 vtkAnimationCue에서 사용하는 AnimationCueTickEvent이다. vjAnimationList를 모두 check하여 현재 플레이 중인 vjAnimationActor에 대해서만 이벤트를 포워드 한다. 이 때, InvokeEvent의 calldata로 index를 넘긴다. calldata는 AddObserver()로 등록된 콜백 함수 내에서, vjAnimationList로부

터 vjAnimationActor를 얻기 위해 사용될 것이다.

```
void
vtkVJInteractor::ForwardAnimationEvent()
{
   int aniNum = mAnimationList->GetAnimationActorNum();

   for (int i = 0; i < aniNum; i++)
   {
      int nRegIndex;

      nRegIndex = i;

      vjAnimationActor* aniActor = mAnimationList->GetAnimationActor(i);

      if(aniActor != NULL)
      {
         if (aniActor->IsOnPlay() && aniActor->IsContinueAnimation())
            this->InvokeEvent(vtkCommand::AnimationCueTickEvent,&nRegIndex);
      }
   }
}
```

[소스 3-10 vtkVJInteractor::ForwardAnimationEvent()]

```
void vtkVJInteractor::StartAnimation(int nRegIndex)
{
   vjAnimationActor* aniActor = mAnimationList->GetAnimationActor(nRegIndex);

   mAnimationList->StartAnimation(nRegIndex);
}
void vtkVJInteractor::StopAnimation(int nRegIndex)
{
   vjAnimationActor* aniActor = mAnimationList->GetAnimationActor(nRegIndex);

   aniActor->StopAnimation();
}
```

[소스 3-11 vtkVJInteractor::StartAnimation()]

vtkVJInteractor는 이벤트를 vjAnimationActor에 전달해야 하므로 멤버 변수로 vjAnimationList를 가진다. VTKApp들은 vtkVJInteractor를 통해 vjAnimationList와 vjAnimationActor에 접근할 수 있다.

따라서, vtkVJInteractor는 vjAniamtionList를 위한 함수도 제공한다.

```
int vtkVJInteractor::GetAnimationActorNum()
{
   return mAnimationList->GetAnimationActorNum();
}
vjAnimationList* vtkVJInteractor::GetAnimationList()
{
   return mAnimationList;
}
int
vtkVJInteractor::AddAnimation(void(*f)(vtkObject *caller, unsigned long eid,
                    void *clientdata, void *calldata), void *clientdata,
                    int nTimeStep)
{
   int regIndex;
   bool isNew;

   if (mAnimationList == NULL)
   {
      cout << "mAnimationList is NULL" << endl;
      return -1;
   }
   regIndex = mAnimationList->AddAnimation(f, clientdata, nTimeStep, &isNew);
   if (regIndex < 0)
   {
      return regIndex;
   }
   if (isNew == true)
   {
      vjAnimationActor* aniActor = mAnimationList->GetAnimationActor(regIndex);
      vtkCallbackCommand* aniCallback = aniActor->GetCallback();

      unsigned long observerId;

      observerId = this->AddObserver(vtkCommand::AnimationCueTickEvent,
                                     aniCallback);
      aniActor->SetObserver(observerId);
   }
   mIsAnimation = true;
   return regIndex;
}
```

[소스 3-12 Supplementary Functions for Animation provided by vtkVJInteractor]

이제 실제 하나의 애니메이션을 vtkVJInteractor에 등록하고, 애니메이션을 수행하기 위해서는 AddAnimation과 그에 따른 콜백을 구현하여야 한다. 다음은 하나의 애니메이션을 생성하는 코드이다.

```
void AniPlayCB(vtkObject *cObject, unsigned long eid, void *clientdata, void *call
data)
{
   .....

   vtkVJInteractor* interactor = aData->mClData->mVJInteractor;

   if (aData->mClData->mVJInteractor == NULL)
   {
      cout << "mVJInteractor is NULL in Animation Data" << endl;
      return;
   }
   .........

   int                  nRegIndex;

   nRegIndex = interactor->AddAnimation(ProcessAniPlayCB, aData, 10000);

   if (nRegIndex < 0)
   {
      glvLog::Write("AnimationList is not initialized\n", 0);
      cout << "AnimationList is not initialized" << endl;
      return;
   }
   interactor->StartAnimation(nRegIndex);
}
```

[소스 3-13 Add and Start Animation ]

vtkVJInteractor에 AddAnimation()함수로 애니메이션을 더하고 생성된 vjActor의 vjAniamtionList내의 index를 받아온다. 이 index로 StartAnimation을 수행한다. 애니메이션을 시작하면, 다음 kernel loop의 preFrame()에서 AnimationCueTickEvent를 발생하고 이를 vjAnimationList에 포워드 한다. 이때, 등록된 ProcessAniPlayCB()가 수행된다. 다음은 ProcessAniPlayCB()의 코드이다.

```
void
ProcessAniReverseCB(vtkObject *cObject, unsigned long eid, void *clientdata, void
*calldata)
{
   char azimuth[256];
   AnimationData *aData = reinterpret_cast<AnimationData*>(clientdata);
   int* nRegIndex = reinterpret_cast<int*>(calldata);

  ..........

   vjAnimationList* aniList = clData->mVJInteractor->GetAnimationList();
   vjAnimationActor* aniActor = aniList->GetAnimationActor((*nRegIndex));

   ....................
   // render current scene
   ....................

   RenderCurrentScene(clData->mGLVRenderer, clData->mRenderer,
                      clData->mVJInteractor, clData->mRoot, clData->mStatus);

   if (aniActor->IsContinueAnimation())
   {
       aniActor->ProceedNextScene();
   }
   else
   {
       aniActor->StopAnimation();
   }
}
```

[소스 3-14 Rendering Current Scene and Processing for Next Scene ]

ProcessAniPlayCB()는 코드 내에서 현재 scene을 만든 다음 vjAnimationActor의 animation을 계속할 것인지 아닌지를 check해 ProceedNextScene을 수행할 것인지 StopAnimation을 수행할 것인지를 결정한다. 이러한 과정을 거치면 ProcessAniReverseCB()내에서 time step 만큼의 애니메이션을 수행할 수 있다.

# 5. 결론

vjVTK는 VRJuggler와 VTK를 활용할 수 있는 좋은 도구이다. 그러나, 아직 0.8

이라는 버전이 말해주듯 많은 부분 버그 수정과 기능 확장이 필요하다. 본 문서에서는 VTK가 상위 버전으로 업그레이드되면서 발생한 vjVTK의 문제점과 인터렉티브한 메뉴구성에 필수적인 picking 이벤트로 콜백 함수를 호출하는 기능, 애니메이션 기능 등의 확장 기능 개발에 대해 기술 하였다. 본 문서에서 기술한 내용 외에도 VTK의 버전확장에 따른 문제는 지속적으로 발생할 것이고, vjVTK는 이를 지원하도록 많은 부분 추가 구현이 필요할 것이다. 또한 vtkVIInteractor가 VTK가 다루는 이벤트들을 제대로 처리 할 수 있기 위해서도 많은 부분이 확장되어야 할 것이다. 그럼에도 불구하고, vjVTK는 VR 환경과 가시화에 대한 요구가 커지고 적용하는 응용이 많아져 기능 확장이 이루어진다면, VTK와 VRJuggler를 통합하는 도구로 자리매김할 수 있을 것이다.

# 6. 참고문헌


[1] vjVTK, http://imve.informatik.uni-hamburg.de/blom/vjVTK.html

[2] http://imve.informatik.uni-hamburg.de/blom/vjVTK.pdf

[3] VRJuggler, http://www.vrjuggler.org

[4] "VRJuggler Programer's Guide", Dec, 2007

[5] "VRJuggler Configuration Guide", May, 2002