ViSP 2.6.1: Visual Servoing Platform

Augmented reality using vpAROgre class

Lagadic project
http://www.irisa.fr/lagadic

September 1, 2011

François Chaumette
Bertrand Delabarre
Eric Marchand
Fabien Spindler
Romain Tallonneau

Contents

1 Introduction

This tutorial describes the `vpAROgre` class used for augmented reality. This class is based on the Ogre3D rendering engine.

Ogre 3D is an open-source graphics rendering engine released, since Ogre 1.7, under the terms of the MIT license . More information about Ogre 3D can be found on http://www.ogre3d.org .

2 Ogre3D installation

This chapter describes the Ogre3D installation process. If the additional optional OIS (Object Oriented Input System) library is also installed, you will be able to use input devices (Keyboards, Mice, Joysticks, etc) to interact with the scene.

Ogre can be installed either from source code (as described in the Ogre wiki), or from a prebuild SDK (Ogre wiki). The method based on SDK is suggested as it is the easiest on. Below is described the platform specific installation process.

2.1 Under Linux

On Linux Ubuntu you can install Ogre 3D library with apt-get:

- to get the last Ogre release, add Ogre PPA with: sudo add-apt-repository ppa:ogre-team/ogre
- then update your package list with: sudo apt-get update
- and finally install Ogre packages with: sudo apt-get install libogre-dev ogre-samples-media
- the optional OIS library can be installed with: sudo apt-get install libois-dev

2.2 Under OSX

Under OSX, you can install Ogre from OGRE 1.7.2 prebuilt SDK for Mac OS X. You might need as well CG Toolkit. Below we give the steps to proceed to the installation:

- Install CG Toolkit from existing .dmg
- Go to http://www.ogre3d.org/download/sdk and download the latest OSX SDK
- Double-click the .dmg to mount it
- Drag & drop the OgreSDK folder wherever you like to install the SDK. We now assume that this folder is `$HOME/OgreSDK`. Il will be easy to adapt the next commands if the SDK is in an other folder
- Copy the Ogre framework in /Library/Frameworks by
  `cp -p -r $HOME/OgreSDK/lib/release/Ogre.framework /Library/Frameworks/`
- Set `OGRE_HOME` environment variable to the SDK folder:
  if you use sh, by `export OGRE_HOME="$HOME/OgreSDK"`
  if you use tcsh, by `setenv OGRE_HOME "$HOME/OgreSDK"`

- Set `BOOST_ROOT` environment variable:
  if you use sh, by `export BOOST_ROOT="$OGRE_HOME/boost_1_42"`
  if you use tcsh, by `setenv BOOST_ROOT "$OGRE_HOME/boost_1_42"`

2.3 Under Windows

Under Windows, you can install Ogre from OGRE 1.7.2 prebuilt SDK for Visual C++ . Download the SDK matching your IDE version. As described on Ogre wiki you might also download and install DirectX. Below we give the steps to proceed to the installation:

- Download and install Ogre SDK to a suitable location, for example in `C:\OgreSDK`
- Set `OGRE_HOME` environment variable to the SDK folder (ie, `C:\OgreSDK`)
- Set `BOOST_ROOT` environment variable to the boost folder provided in Ogre SDK (ie, `C:\OgreSDK\boost_1_42`)
- You may also modify the PATH environment variable so that Ogre DLL can be found during execution (ie, in our case, add `C:\OgreSDK\bin\release` folder in the path)
- Download and install DirectX.

3 Building your project

To build your project the simplest way is to use CMake. Here is an example of what your CMakeLists.txt could look like :

`CMakeLists.txt` :

```
1  cmake_minimum_required(VERSION 2.6)
2  PROJECT(OgreTutorial)
3
4  # Add visp
5  FIND_PACKAGE(VISP REQUIRED)
6  IF(VISP_FOUND)
7    INCLUDE(${VISP_USE_FILE})
8  ENDIF(VISP_FOUND)
9
10 ADD_EXECUTABLE(HelloWorldOgre HelloWorldOgre.cpp)
```

4 How to use the vpAROgre class

4.1 Simple approach

In the simple approach, just create a `vpAROgre` object and initialise it with your models. The rendering loop is then fairly straightforward: get an image, compute your pose and display your scene. To stop your application, hit the Escape button. The following `HelloWorldOgre.cpp` [1] file shows how to use the `vpAROgre` class to do a simple rendering.

`HelloWorldOgre.cpp` :

[1] `HelloWorldOgre.cpp` and `CMakeLists.txt` files are available in ViSP source tree in `example/manual/ogre` directory.

```
#include <visp/vpConfig.h>
#include <visp/vpV4l2Grabber.h>
#include <visp/vp1394TwoGrabber.h>
#include <visp/vpDirectShowGrabber.h>
#include <visp/vpOpenCVGrabber.h>
#include <visp/vpHomogeneousMatrix.h>
#include <visp/vpImage.h>
#include <visp/vpCameraParameters.h>
#include <visp/vpAROgre.h>

int main()
{
  // Now we try to find an available framegrabber
#if defined(VISP_HAVE_V4L2)
  // Video for linux 2 grabber
  vpV4l2Grabber grabber;
#elif defined(VISP_HAVE_DC1394_2)
  // libdc1394-2
  vp1394TwoGrabber grabber;
#elif defined(VISP_HAVE_DIRECTSHOW)
  // OpenCV to gather images
  vpOpenCVGrabber grabber;
#elif defined(VISP_HAVE_OPENCV)
  // OpenCV to gather images
  vpOpenCVGrabber grabber;
#else
#  error "You need an available framegrabber to run this example"
#endif

  // Image to stock gathered data
  // Here we acquire a color image. The consequence will be that
  // the background texture used in Ogre renderer will be also in color.
  vpImage<vpRGBa> I;
  // Open frame grabber
  // Here we acquire an image from an available framegrabber to update
  // the image size
  grabber.open(I);

  // Parameters of our camera
  double px = 565;
  double py = 565;
  double u0 = grabber.getWidth() / 2;
  double v0 = grabber.getHeight() / 2;
  vpCameraParameters cam(px,py,u0,v0);
  // The matrix with our pose
  // Defines the pose of the object in the camera frame
  vpHomogeneousMatrix cMo;

  // Our object
  // A simulator with the camera parameters defined above,
  // a grey level background image and of the good size
  vpAROgre ogre(cam, (unsigned int)grabber.getWidth(), (unsigned int)grabber.getHeight());
  // Initialisation
  // Here we load the requested plugins specified in the "plugins.cfg" file
  // and the resources specified in the "resources.cfg" file
  // These two files can be found respectively in ViSP_HAVE_OGRE_PLUGINS_PATH
  // and ViSP_HAVE_OGRE_RESOURCES_PATH folders
  ogre.init(I);

  // Create a basic scene
  // -----------------------------------
  //          Loading things
  // -----------------------------------
  //  As you will see in section 5, our
```

```
65    //  application knows locations where
66    //  it can search for medias.
67    //  Here we use a mesh included in
68    //  the installation files : a robot.
69    // ----------------------------------
70    // Here we load the "robot.mesh" model that is found thanks to the ressources locations
71    // specified in the "resources.cfg" file
72    ogre.load("Robot", "robot.mesh");
73    ogre.setPosition("Robot", vpTranslationVector(0, 0.05, 0.5));
74    ogre.setScale("Robot", 0.001,0.001,0.001);
75    ogre.setRotation("Robot", vpRotationMatrix(vpRxyzVector(M_PI, -M_PI/4, 0)));
76
77
78    // Rendering loop, ended with on escape
79    while(ogre.continueRendering()){
80        // Image Acquisition
81        // Acquire a new image
82        grabber.acquire(I);
83        //Pose computation
84        // ...
85        // cMo updated
86        // Display the robot at the position specified by cMo with vpAROgre
87        ogre.display(I,cMo);
88    }
89    // Release video device
90    grabber.close();
91 }
```

Figure 1 shows the result of `HelloWorldOgre.cpp`.

This approach is really basic and can be not sufficient to achieve a complete 3D application. Indeed, using this method, it is not possible to interact with the application with any other button but the Escape one. In this case, a new class, inheriting from the class `vpAROgre`, must be created.

Figure 1: A snapshot of the rendering produced by `HelloWorldOgre.cpp`. Color images acquired by an available framegrabber are used as a background texture, while a robot is projected in the scene at a static position with translation coordinates $(0, 0.05, 0.05)$ and rotation coordinates $(\pi, -\pi/4, 0)$.

4.2 Advanced approach

If you want a complete Ogre application you should inherit from `vpAROgre`. This way you can redefine your basic scene easilly, choose materials, specify a behaviour and do many other things. In the following example, the robot used in the first example will be animated.
We begin by redefining, in the following `HelloWorldOgreAdvanced.cpp` [2], a class which inherit from `vpAROgre` so we do not have to do everything again from scratch.

`HelloWorldOgreAdvanced.cpp` :

```
#include <visp/vpConfig.h>
#include <visp/vpV4l2Grabber.h>
#include <visp/vp1394TwoGrabber.h>
#include <visp/vpDirectShowGrabber.h>
#include <visp/vpOpenCVGrabber.h>
#include <visp/vpHomogeneousMatrix.h>
#include <visp/vpImage.h>
#include <visp/vpCameraParameters.h>
#include <visp/vpAROgre.h>

class vpAROgreAdvanced : public vpAROgre
{
private:
  // Animation attribute
  Ogre::AnimationState * mAnimationState;

public:
  vpAROgreAdvanced(const vpCameraParameters &cam = vpCameraParameters(),
       unsigned int width = 640, unsigned int height = 480)
    : vpAROgre(cam, width, height)
  {
  }
```

Here, instead of managing the meshes from the main program, the scene is defined by overloading the `createScene()` method defined in `vpAROgre`. For example:

```
protected:
  void createScene()
  {
    // Create the Entity
    Ogre::Entity* robot = mSceneMgr->createEntity("Robot", "robot.mesh");
    // Attach robot to scene graph
    Ogre::SceneNode* RobotNode = mSceneMgr->getRootSceneNode()->createChildSceneNode("Robot");
    RobotNode->setPosition(0, 0.05, 0.5);
    RobotNode->attachObject(robot);
    RobotNode->scale(0.001,0.001,0.001);
    RobotNode->pitch(Ogre::Degree(180));
    RobotNode->yaw(Ogre::Degree(-90));

    // The animation
    // Set the good animation
    mAnimationState = robot->getAnimationState( "Idle" );
    // Start over when finished
    mAnimationState->setLoop( true );
    // Animation enabled
    mAnimationState->setEnabled( true );
  }
```

[2] `HelloWorldOgreAdvanced.cpp` and `CMakeLists.txt` files are available in ViSP source tree in `example/manual/ogre` directory.

In this example, the robot is animated with the *"Idle"* state. It makes the robot move slightly, look around and wait. The animations associated to a 3D model can be checked using tools such as Cegui Mesh Viewer.

The animation speed is specified by giving at each new frame the time since the last one. This is done by overloading the method `customframeEnded()`:

```
  bool customframeEnded( const Ogre::FrameEvent& evt)
  {
    // Update animation
    // To move, we add it the time since last frame
    mAnimationState->addTime( evt.timeSinceLastFrame );
    return true;
  }

};// End of vpAROgreAdvanced class definition
```

The new class can be used exactly like the original one:

```
int main()
{
  // Now we try to find an available framegrabber
#if defined(VISP_HAVE_V4L2)
  // Video for linux 2 grabber
  vpV4l2Grabber grabber;
#elif defined(VISP_HAVE_DC1394_2)
  // libdc1394-2
  vp1394TwoGrabber grabber;
#elif defined(VISP_HAVE_DIRECTSHOW)
  // OpenCV to gather images
  vpOpenCVGrabber grabber;
#elif defined(VISP_HAVE_OPENCV)
  // OpenCV to gather images
  vpOpenCVGrabber grabber;
#else
#  error "You need an available framegrabber to run this example"
#endif

  // Image to store gathered data
  // Here we acquire a grey level image. The consequence will be that
  // the background texture used in Ogre renderer will be also in grey
  // level.
  vpImage<unsigned char> I;
  // Open frame grabber
  // Here we acquire an image from an available framegrabber to update
  // the image size
  grabber.open(I);
  // Parameters of our camera
  double px = 565;
  double py = 565;
  double u0 = grabber.getWidth() / 2;
  double v0 = grabber.getHeight() / 2;
  vpCameraParameters cam(px,py,u0,v0);
  // The matrix with our pose
  vpHomogeneousMatrix cMo;

  // Our object
  vpAROgreAdvanced ogre(cam, (unsigned int)grabber.getWidth(), (unsigned int)grabber.getHeight());
  // Initialisation
  ogre.init(I);

  // Rendering loop
  while(ogre.continueRendering()){
      // Image Acquisition
      grabber.acquire(I);
      // Pose computation
```

```
48        // ...
49        // cMo updated
50        // Display with vpAROgre
51        ogre.display(I, cMo);
52    }
53    // Release video device
54    grabber.close();
55  }
```

As you can see the main program basically stays the same, we just redefined some methods to have a moving entity.
To know more about creating Ogre3D applications, please refer to their official website and particularily their wiki where you will find various tutorials :

http://www.ogre3d.org/tikiwiki/tiki-index.php.

5 Warnings

5.1 Extensions

Ogre3D uses its own 3D model and material extensions. See its website for more information on how to export your models in the good format:

http://www.ogre3d.org/tikiwiki/OGRE+Exporters.

5.2 Running your application

To run, the created application will need a `resources.cfg` file and a `plugin.cfg` file. These files tell Ogre where to look for textures, models, materials and other things like that. They also tell Ogre which plugins to use. Note that ViSP users don't have to create manually these files. ViSP is able to detect [3] and use existing files, or if they are not detected to create automatically such files. Below, we give examples of what these two files could look like.

Plugins

The `plugins.cfg` file is used to set up graphical features of the application like, for example, the rendering system to use. In ViSP, the location of `plugins.cfg` file is specified by the `ViSP_HAVE_OGRE_PLUGINS_PATH` macro defined in `visp/vpConfig.h` file. To ease ViSP usage, if, during ViSP configuration `plugins.cfg` is not found, ViSP creates a `plugins.cfg` file in `data/ogre-simulator` built tree. As shown below, in `plugins.cfg` line 4, the folder that contains the plugins is first defined. It is advised to set an absolute folder. Then the plugins that will be used are defined. Beware if they are not correctly found you could get errors.

`plugins.cfg`:

[3] If you installed Ogre from a prebuilt SDK you should find those files in `OGRE_HOME/bin` directory.

```
# Defines plugins to load

# Define plugin folder
PluginFolder=OGRE_HOME/bin # ( or /usr/local/share/OGRE for example)

# Define plugins
# If we do not like Direct3D we just comment them
# Plugin=RenderSystem_Direct3D9
# Plugin=RenderSystem_Direct3D10
Plugin=RenderSystem_GL
Plugin=Plugin_ParticleFX
Plugin=Plugin_BSPSceneManager
Plugin=Plugin_OctreeSceneManager
Plugin=Plugin_CgProgramManager
```

Resources

In `HelloWorldOgre.cpp` line 72 when we load meshes, we just have to give their name (it is the same with materials and other medias). This is possible thanks to the `resources.cfg` file. In this file the folders where what we will load is located are defined, so that Ogre preparses them and looks for the resources when asked. In ViSP, the location of the `resources.cfg` file is specified by the `ViSP_HAVE_OGRE_RESOURCES_PATH` macro defined in the `visp/vpConfig.h` file.

`resources.cfg`:

```
# Resources required by the sample browser and most samples.
[Essential]
# Here we chose to give absolute path to be independent with relation to the executable folder
Zip=/usr/local/share/OGRE/media/packs/SdkTrays.zip
FileSystem=/usr/local/share/OGRE/media/thumbnails

# Common sample resources needed by many of the samples.
# Rarely used resources should be separately loaded by the
# samples which require them.
[Popular]
FileSystem=/usr/local/share/OGRE/media/fonts
FileSystem=/usr/local/share/OGRE/media/materials/programs
FileSystem=/usr/local/share/OGRE/media/materials/scripts
FileSystem=/usr/local/share/OGRE/media/materials/textures
FileSystem=/usr/local/share/OGRE/media/materials/textures/nvidia
FileSystem=/usr/local/share/OGRE/media/models
FileSystem=/usr/local/share/OGRE/media/particle
FileSystem=/usr/local/share/OGRE/media/RTShaderLib
# We could also give a relative path
FileSystem=../../../../media/RTShaderLib/materials
Zip=/usr/local/share/OGRE/media/packs/cubemap.zip
Zip=/usr/local/share/OGRE/media/packs/cubemapsJS.zip
Zip=/usr/local/share/OGRE/media/packs/dragon.zip
Zip=/usr/local/share/OGRE/media/packs/fresneldemo.zip
Zip=/usr/local/share/OGRE/media/packs/ogretestmap.zip
Zip=/usr/local/share/OGRE/media/packs/ogredance.zip
Zip=/usr/local/share/OGRE/media/packs/Sinbad.zip

[General]
FileSystem=/usr/local/share/OGRE/media
```