**Fichiers à modifier pour avoir une compatibilité chai3d 2.0 et chai3d 3.1.1**

version 32bits et 64 bits

Ajouter et modifier dans le dossier src\devices et dans la solution CHAI3D :

**CVirtualDevice.cpp**  (version modifiée format 3.1.1)

//===========================================================================

/\*

This file is part of the CHAI 3D visualization and haptics libraries.

Copyright (C) 2003-2009 by CHAI 3D. All rights reserved.

This library is free software; you can redistribute it and/or modify

it under the terms of the GNU General Public License("GPL") version 2

as published by the Free Software Foundation.

For using the CHAI 3D libraries with software that can not be combined

with the GNU GPL, and for taking advantage of the additional benefits

of our support services, please contact CHAI 3D about acquiring a

Professional Edition License.

\author <http://www.chai3d.org>

\author Francois Conti

\version 2.0.0 $Rev: 244 $

\version modifie le 16/02/2016 Wilfrid Grassi pour fonctionner avec chai3d 3.1.1

\*/

//===========================================================================

//---------------------------------------------------------------------------

#include "system/CGlobals.h"

#include "devices/CVirtualDevice.h"

//---------------------------------------------------------------------------

#if defined(C\_ENABLE\_VIRTUAL\_DEVICE\_SUPPORT)

//---------------------------------------------------------------------------

//------------------------------------------------------------------------------

namespace chai3d {

//------------------------------------------------------------------------------

//===========================================================================

/\*!

Constructor of cVirtualDevice.

\*/

//===========================================================================

cVirtualDevice::cVirtualDevice(unsigned int a\_deviceNumber)

{

// the connection to your device has not yet been established.

// device is not yet available or ready

m\_deviceAvailable = false;

m\_deviceReady = false;

// settings:

// haptic device model (see file "CGenericHapticDevice.h")

m\_specifications.m\_model = C\_HAPTIC\_DEVICE\_VIRTUAL;

// name of the device manufacturer,

m\_specifications.m\_manufacturerName = "CHAI 3D-WG";

// name of my device Wilfrid Grassi VirtualDevice

m\_specifications.m\_modelName = "VirtualDevice-WG";

//--------------------------------------------------------------------------

// CHARACTERISTICS: (The following values must be positive or equal to zero)

//--------------------------------------------------------------------------

// the maximum force [N] the device can produce along the x,y,z axis.

m\_specifications.m\_maxLinearForce = 10.0; //# [N]

// the maximum closed loop linear stiffness in [N/m] along the x,y,z axis

m\_specifications.m\_maxLinearStiffness = 2000.0; //# [N/m]

// the maximum amount of torque your device can provide arround its

// rotation degrees of freedom.

m\_specifications.m\_maxAngularTorque = 0.0; // [N\*m]

// the maximum amount of angular stiffness

m\_specifications.m\_maxAngularStiffness = 0.0; // [N\*m/Rad]

// the maximum amount of torque which can be provided by your gripper

m\_specifications.m\_maxGripperForce = 0.0; // [N]

// the maximum amount of stiffness supported by the gripper

m\_specifications.m\_maxGripperLinearStiffness = 0.0; //# [N\*m]

// the radius of the physical workspace of the device (x,y,z axis)

m\_specifications.m\_workspaceRadius = 0.15; //# [m]

// the maximum opening angle of the gripper

m\_specifications.m\_gripperMaxAngleRad = cDegToRad(30.0);

//--------------------------------------------------------------------------

// CHARACTERISTICS: (The following are of boolean type: (true or false)

//--------------------------------------------------------------------------

// does your device provide sensed position (x,y,z axis)?

m\_specifications.m\_sensedPosition = true;

// does your device provide sensed rotations (i.e stylus)?

m\_specifications.m\_sensedRotation = false;

// does your device provide a gripper which can be sensed?

m\_specifications.m\_sensedGripper = false;

// is you device actuated on the translation degrees of freedom?

m\_specifications.m\_actuatedPosition = true;

// is your device actuated on the rotation degrees of freedom?

m\_specifications.m\_actuatedRotation = false;

// is the gripper of your device actuated?

m\_specifications.m\_actuatedGripper = false;

// can the device be used with the left hand?

m\_specifications.m\_leftHand = true;

// can the device be used with the right hand?

m\_specifications.m\_rightHand = true;

//------------------------ MAP FILE ---------------------------------------------

// search for virtual device

m\_hMapFile = OpenFileMapping(

FILE\_MAP\_ALL\_ACCESS,

FALSE,

"dhdVirtual");

// no virtual device available

if (m\_hMapFile == NULL)

{

// device is not yet available or ready

m\_deviceAvailable = false;

m\_deviceReady = false;

return;

}

// open connection to virtual device

m\_lpMapAddress = MapViewOfFile(

m\_hMapFile,

FILE\_MAP\_ALL\_ACCESS,

0,

0,

0);

// check whether connection succeeded

if (m\_lpMapAddress == NULL)

{

// device is not yet available or ready

m\_deviceAvailable = false;

m\_deviceReady = false;

return;

}

// map memory

m\_pDevice = (cVirtualDeviceData\*)m\_lpMapAddress;

if (m\_pDevice == NULL)

{

// device is not yet available or ready

m\_deviceAvailable = false;

m\_deviceReady = false;

return;

}

// virtual device is available

m\_deviceAvailable = true;

}

//===========================================================================

/\*!

Destructor of cVirtualDevice.

\*/

//===========================================================================

cVirtualDevice::~cVirtualDevice()

{

if (m\_deviceReady)

{

close();

}

}

//===========================================================================

/\*!

This method opens a connection to your device.

\return \_\_true\_\_ if the operation succeeds, \_\_false\_\_ otherwise.

\*/

//===========================================================================

bool cVirtualDevice::open()

{

// check if the system is available

if (!m\_deviceAvailable) return (C\_ERROR);

// if system is already opened then return

if (m\_deviceReady) return (C\_SUCCESS);

m\_deviceReady = true;

return (C\_SUCCESS);

}

//===========================================================================

/\*!

This method closes the connection to your device.

\return \_\_true\_\_ if the operation succeeds, \_\_false\_\_ otherwise.

\*/

//===========================================================================

bool cVirtualDevice::close()

{

// check if the system has been opened previously

if (!m\_deviceReady) return (C\_ERROR);

CloseHandle(m\_hMapFile);

// update status

m\_deviceReady = false;

return (C\_SUCCESS);

}

//==============================================================================

/\*!

This method calibrates your device.

This method calibrates your device.

\return \_\_true\_\_ if the operation succeeds, \_\_false\_\_ otherwise.

\*/

//==============================================================================

bool cVirtualDevice::calibrate(bool a\_forceCalibration)

{

// check if the device is read. See step 3.

if (!m\_deviceReady) return (C\_ERROR);

return (C\_SUCCESS);

}

//==============================================================================

/\*!

This method returns the number of devices available from this class of device.

\return \_\_true\_\_ if the operation succeeds, \_\_false\_\_ otherwise.

\*/

//==============================================================================

unsigned int cVirtualDevice::getNumDevices()

{

int numberOfDevices = 1; // At least set to 1 if a device is available.

return (numberOfDevices);

}

//===========================================================================

/\*!

This method returns the position of your device. Units are meters [m].

\param a\_position Return value.

\return \_\_true\_\_ if the operation succeeds, \_\_false\_\_ otherwise.

\*/

//===========================================================================

bool cVirtualDevice::getPosition(cVector3d& a\_position)

{

// check if the device is read. See step 3.

if (!m\_deviceReady)

{

a\_position.set(0.0, 0.0, 0.0);

return (C\_ERROR);

}

double x,y,z;

x = 0.0; // x = getMyDevicePositionX()

y = 0.0; // y = getMyDevicePositionY()

z = 0.0; // z = getMyDevicePositionZ()

x = (double)(\*m\_pDevice).PosX;

y = (double)(\*m\_pDevice).PosY;

z = (double)(\*m\_pDevice).PosZ;

// store new position values

a\_position.set(x, y, z);

// estimate linear velocity

#if !defined(MACOSX) & !defined(LINUX)

// estimateLinearVelocity(a\_position);

#endif

// exit

return (C\_SUCCESS);

}

//===========================================================================

/\*!

This method returns the orientation frame of your device end-effector

\param a\_rotation Return value.

\return \_\_true\_\_ if the operation succeeds, \_\_false\_\_ otherwise.

\*/

//===========================================================================

bool cVirtualDevice::getRotation(cMatrix3d& a\_rotation)

{

// check if the device is read.

if (!m\_deviceReady)

{

a\_rotation.identity();

return (C\_ERROR);

}

a\_rotation.identity();

// exit

return (C\_SUCCESS);

}

//==============================================================================

/\*!

This method sends a force [N] and a torque [N\*m] and gripper torque [N\*m]

to your haptic device.

\param a\_force Force command.

\param a\_torque Torque command.

\param a\_gripperForce Gripper force command.

\return \_\_true\_\_ if the operation succeeds, \_\_false\_\_ otherwise.

\*/

//==============================================================================

bool cVirtualDevice::setForceAndTorqueAndGripperForce(const cVector3d& a\_force,

const cVector3d& a\_torque,

const double a\_gripperForce)

{

// check if the device is read. See step 3.

if (!m\_deviceReady) return (C\_ERROR);

// store new force value.

m\_prevForce = a\_force;

m\_prevTorque = a\_torque;

m\_prevGripperForce = a\_gripperForce;

// retrieve force, torque, and gripper force components in individual variables

((\*m\_pDevice).ForceX) = a\_force(0);

((\*m\_pDevice).ForceY) = a\_force(1);

((\*m\_pDevice).ForceZ) = a\_force(2);

double ((\*m\_pDevice).TorqueA) = a\_torque(0);

double ((\*m\_pDevice).TorqueB) = a\_torque(1);

double ((\*m\_pDevice).TorqueG) = a\_torque(2);

// exit

return (C\_SUCCESS);

}

//===========================================================================

/\*!

This method return the last force [N] sent to the device.

\param a\_force Force command.

\return \_\_true\_\_ if the operation succeeds, \_\_false\_\_ otherwise.

\*/

//===========================================================================

bool cVirtualDevice::getForce(cVector3d& a\_force)

{

// check if the device is read. See step 3.

if (!m\_deviceReady)

{

a\_force.set(0,0,0);

return (C\_ERROR);

}

a\_force(0) = ((\*m\_pDevice).ForceX);

a\_force(1) = ((\*m\_pDevice).ForceY);

a\_force(2) = ((\*m\_pDevice).ForceZ);

return (C\_SUCCESS);

}

//==============================================================================

/\*!

This method returns status of all user switches

[\_\_true\_\_ = \_\_ON\_\_ / \_\_false\_\_ = \_\_OFF\_\_].

\param a\_userSwitches Return the 32-bit binary mask of the device buttons.

\return \_\_true\_\_ if the operation succeeds, \_\_false\_\_ otherwise.

\*/

//==============================================================================

bool cVirtualDevice::getUserSwitches(unsigned int& a\_userSwitches)

{

// check if the device is read. See step 3.

if (!m\_deviceReady) return (C\_ERROR);

////////////////////////////////////////////////////////////////////////////

/\*

STEP 11:

Here you shall implement code that reads the status all user switches

on your device. For each user switch, set the associated bit on variable

a\_userSwitches. If your device only has one user switch, then set

a\_userSwitches to 1, when the user switch is engaged, and 0 otherwise.

\*/

////////////////////////////////////////////////////////////////////////////

// \*\*\* INSERT YOUR CODE HERE \*\*\*

a\_userSwitches = 0;

return (C\_SUCCESS);

}

//------------------------------------------------------------------------------

} // namespace chai3d

//------------------------------------------------------------------------------

#endif // C\_ENABLE\_VIRTUAL\_DEVICE\_SUPPORT

//------------------------------------------------------------------------------

**CVirtualDevice.h** (version modifiée format 3.1.1)

//===========================================================================

/\*

This file is part of the CHAI 3D visualization and haptics libraries.

Copyright (C) 2003-2009 by CHAI 3D. All rights reserved.

This library is free software; you can redistribute it and/or modify

it under the terms of the GNU General Public License("GPL") version 2

as published by the Free Software Foundation.

For using the CHAI 3D libraries with software that can not be combined

with the GNU GPL, and for taking advantage of the additional benefits

of our support services, please contact CHAI 3D about acquiring a

Professional Edition License.

\author <http://www.chai3d.org>

\author Francois Conti

\version 2.0.0 $Rev: 251 $

\version modifie le 16/02/2016 Wilfrid Grassi pour fonctionner avec chai3d 3.1.1

\*/

//===========================================================================

//---------------------------------------------------------------------------

#ifndef CVirtualDeviceH

#define CVirtualDeviceH

//------------------------------------------------------------------------------

#if defined(C\_ENABLE\_VIRTUAL\_DEVICE\_SUPPORT)

//------------------------------------------------------------------------------

//---------------------------------------------------------------------------

#include "devices/CGenericHapticDevice.h"

//---------------------------------------------------------------------------

//---------------------------------------------------------------------------

namespace chai3d {

//------------------------------------------------------------------------------

//===========================================================================

/\*!

\file CVirtualDevice.h

\brief

<b> Devices </b> \n

Virtual Haptic Device.

\*/

//===========================================================================

//------------------------------------------------------------------------------

#if defined(WIN32) | defined(WIN64)

//------------------------------------------------------------------------------

#ifndef DOXYGEN\_SHOULD\_SKIP\_THIS

//------------------------------------------------------------------------------

//------------------------------------------------------------------------------

#endif // DOXYGEN\_SHOULD\_SKIP\_THIS

//------------------------------------------------------------------------------

#endif // WIN32

//------------------------------------------------------------------------------

//------------------------------------------------------------------------------

class cVirtualDevice;

typedef std::shared\_ptr<cVirtualDevice> cVirtualDevicePtr;

//------------------------------------------------------------------------------

//==============================================================================

struct cVirtualDeviceData

{

double ForceX; // Force component X.

double ForceY; // Force component Y.

double ForceZ; // Force component Z.

double TorqueA; // Torque alpha.

double TorqueB; // Torque beta.

double TorqueG; // Torque gamma.

double PosX; // Position X.

double PosY; // Position Y.

double PosZ; // Position Z.

double AngleA; // Angle alpha.

double AngleB; // Angle beta.

double AngleG; // Angle gamma.

bool Button0; // Button 0 status.

bool AckMsg; // Acknowledge Message

bool CmdReset; // Command Reset

};

//===========================================================================

/\*!

\class cVirtualDevice

\ingroup devices

\brief

Class which interfaces with the virtual device

\*/

//===========================================================================

class cVirtualDevice : public cGenericHapticDevice

{

//-----------------------------------------------------------------------

// CONSTRUCTOR & DESTRUCTOR:

//-----------------------------------------------------------------------

public:

//! Constructor of cVirtualDevice.

cVirtualDevice(unsigned int a\_deviceNumber = 0);

//! Destructor of cVirtualDevice.

virtual ~cVirtualDevice();

//! Shared cVirtualDevice allocator.

static cVirtualDevicePtr create(unsigned int a\_deviceNumber = 0) { return (std::make\_shared<cVirtualDevice>(a\_deviceNumber)); }

//--------------------------------------------------------------------------

// PUBLIC METHODS:

//--------------------------------------------------------------------------

public:

//! This method opens a connection to the haptic device.

virtual bool open();

//! This method closes the connection to the haptic device.

virtual bool close();

//! This method calibrates the haptic device.

virtual bool calibrate(bool a\_forceCalibration = false);

//! This method returns the position of the device.

virtual bool getPosition(cVector3d& a\_position);

//! This method returns the orientation frame of the device end-effector.

virtual bool getRotation(cMatrix3d& a\_rotation);

//! This method returns the status of all user switches [\_\_true\_\_ = \_\_ON\_\_ / \_\_false\_\_ = \_\_OFF\_\_].

virtual bool getUserSwitches(unsigned int& a\_userSwitches);

//! This method sends a force [N] and a torque [N\*m] and gripper force [N] to the haptic device.

virtual bool setForceAndTorqueAndGripperForce(const cVector3d& a\_force, const cVector3d& a\_torque, double a\_gripperForce);

//! Read a force [N] from the haptic device.

virtual bool getForce(cVector3d& a\_force);

//--------------------------------------------------------------------------

// PUBLIC STATIC METHODS:

//--------------------------------------------------------------------------

public:

//! This method returns the number of devices available from this class of device.

static unsigned int getNumDevices();

//--------------------------------------------------------------------------

// PROTECTED MEMBERS:

//--------------------------------------------------------------------------

////////////////////////////////////////////////////////////////////////////

/\*

INTERNAL VARIABLES:

If you need to declare any local variables or methods for your device,

you may do it here below.

\*/

////////////////////////////////////////////////////////////////////////////

private:

//! Shared memory connection to virtual haptic device.

HANDLE m\_hMapFile;

//! Pointer to shared memory.

LPVOID m\_lpMapAddress;

//! Pointer to shared memory data structure.

cVirtualDeviceData\* m\_pDevice;

};

//------------------------------------------------------------------------------

} // namespace chai3d

//------------------------------------------------------------------------------

#endif // C\_ENABLE\_VIRTUAL\_DEVICE\_SUPPORT

//---------------------------------------------------------------------------

#endif

//------------------------------------------------------------------------------

Modifier dans src

**CHapticDeviceHandler.cpp**

//####

#include "..\src\extras\CExtras.h"

.

.

.

#if defined(C\_ENABLE\_VIRTUAL\_DEVICE\_SUPPORT)

#include "devices/CVirtualDevice.h"

#endif

.

.

.

//-----------------------------------------------------------------------

// search for CHAI 3D Virtual Device

// Note:

// Virtual devices should always be listed last. The desired behavior

// is that an application first searches for physical devices. If none

// are found, it may launch a virtual device

//-----------------------------------------------------------------------

#if defined(C\_ENABLE\_VIRTUAL\_DEVICE\_SUPPORT)

// if there are one or more devices available, then store it in the device table

/\* if (count > 0)

{

// check for how many devices of this type that are available

count = cVirtualDevice::getNumDevices();

// create a first device of this class

device = cVirtualDevice::create(count);

// store first device

m\_devices[m\_numDevices] = device;

m\_numDevices++;

}

else\*/

if (count == 0) //m\_numDevices == 0) // if no devices have been found then we try to launch a virtual haptic device

{

// we try to launch the virtual device.

spawnlp(\_P\_NOWAIT, "VirtualDevice.exe", "VirtualDevice.exe", NULL);

cSleepMs(750);

// create again a first device of this class

device = cVirtualDevice::create(count);

// check for how many devices of this type that are available

count = cVirtualDevice::getNumDevices();

// if there are one or more devices available, then store it in the device table

if (count > 0)

{

// store first device

m\_devices[m\_numDevices] = device;

m\_numDevices++;

}

}

#endif

**CGenericHapticDevice.h**

//==============================================================================

/\*!

Defines the list of devices currently supported by CHAI3D.

\*/

//==============================================================================

enum cHapticDeviceModel

{

**C\_HAPTIC\_DEVICE\_VIRTUAL,**

C\_HAPTIC\_DEVICE\_DELTA\_3,

C\_HAPTIC\_DEVICE\_DELTA\_6,

C\_HAPTIC\_DEVICE\_OMEGA\_3,

C\_HAPTIC\_DEVICE\_OMEGA\_6,

C\_HAPTIC\_DEVICE\_OMEGA\_7,

C\_HAPTIC\_DEVICE\_SIGMA\_6P,

C\_HAPTIC\_DEVICE\_SIGMA\_7,

C\_HAPTIC\_DEVICE\_FALCON,

C\_HAPTIC\_DEVICE\_XTH\_1,

C\_HAPTIC\_DEVICE\_XTH\_2,

C\_HAPTIC\_DEVICE\_MPR,

C\_HAPTIC\_DEVICE\_PHANTOM\_TOUCH,

C\_HAPTIC\_DEVICE\_PHANTOM\_OMNI,

C\_HAPTIC\_DEVICE\_PHANTOM\_DESKTOP,

C\_HAPTIC\_DEVICE\_PHANTOM\_15\_6DOF,

C\_HAPTIC\_DEVICE\_PHANTOM\_30\_6DOF,

C\_HAPTIC\_DEVICE\_PHANTOM\_OTHER,

C\_TRACKER\_DEVICE\_SIXENSE,

C\_TRACKER\_DEVICE\_LEAP,

C\_HAPTIC\_DEVICE\_CUSTOM

};

**Modifier le fichier src/system/CGlobals.h en 3 endroits Windows/Linux/Mac**

//--------------------------------------------------------------------

// HAPTIC DEVICES

//--------------------------------------------------------------------

#define C\_ENABLE\_VIRTUAL\_DEVICE\_SUPPORT

version 64bits

**Ajouter le dossier extras dans src**

il contient les fichiers :

CExtras.cpp

CExtras.h

CGenericType.h

CGlobals.h