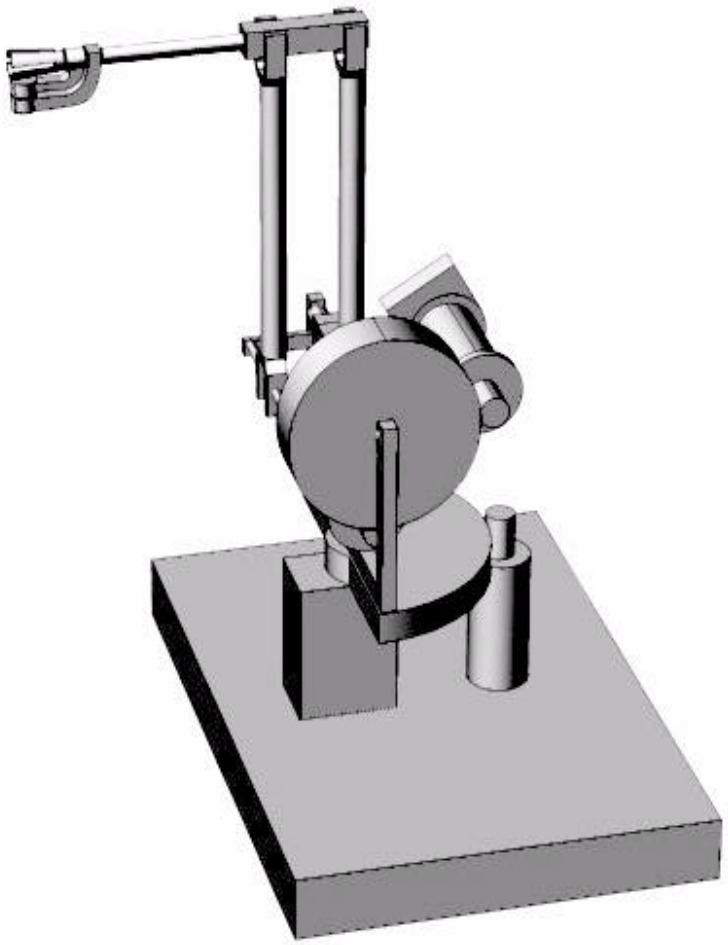
# Application of the Haptic-device PHANToM for the generation of trayectories of a polishing belt over a rigid object defined in VRML.

Gerrit Färber, Luis Basañez

**Institut d´Organització i Control de Sistemes Industrials (IOC)**
**Programa Doctorado: Automatización Avanzada y Robótica**



# Table of contents

PART 1

# *Introduction*

Haptic devices offer new perspectives for areas like simulation and teleoperation. For countless applications in the world of three dimensional space the interaction between the operator and the computer requires more than the simple two dimensional input of the mouse. Even more with the haptic devices it is possible to feedback the forces and torques that result in the collision with the objects being part of the simulation or the amplified area of vision. It is a common technique to scale the length and forces applied by an operator.

One of the applications of automation in the industry is the polishing of workpieces that are passed along a polishing belt. The advantage of the automation of these processes is to avoid accidents of operators, harm caused by the dust in the lungs of the operator and of course the fact that all the workpieces are treated in the same way which then results in a higher quality of the product.
In the “Institut d´Organització i Control de Sistemes Industrials” (IOC), a software-program was developed that permits applying trayectories to a workpiece which is described with a mesh in VRML. The information of the trayectories then are stored in a file which can be utilized by the robot executing the task.

For the positioning of the trayectories the program uses the mouse and the keyboard as interface. Taking advantage of the possibilities that offers a haptic device, the objective of the presented work is to be able to use the program with haptic device PHANToM that is manufactured by the company SensAble Technologies.

The presented work is based on the existing software program that handles and administrates the workpiece and the applied trayectories. Additional functionality was implemented which permits the operator to use the PHANToM to determine the position and the force of each trayectory.

PART 2

# *Application PCG*

In the IOC a software-application was developed, written in $C^{++}$ containing the implementation of the PCG (Polishing Curves Generator). The program was created as part of a final year´s project by Iván Díaz Martín, supervised by Luis Basañez Villaluenga in the year 2001. The title of this project is “Generación de una interface gráfica para determinar trayectorias a realizar por robots en el pulido de piezas”.

The objective of the project was to create a tool on the computer that permits the generation of trayectories over a workpiece (for example a doorknob) using a set of parameters. The stored trayectories then can be used to program a robot executing a task of polishing a workpiece by passing it along a polishing belt.
The parameters include values defining the force with which the workpiece will be pressed against the polishing belt, the velocity of the polishing belt for the particular trayectory and also the width of the polishing belt. Depending on these parameters the robot is able to fulfill his work automatically.

## *2.1 PCG*

The application of the PCG was written for the PC-platform with the operating system Microsoft Windows. The program was implemented in $C^{++}$ using the Microsoft Foundation Classes (MFC) for the programming of the windows and the administration of the data. The visual part of the Application-Interface (API) is implemented with OpenGL, showing the object to be polished, a pointer which indicates where to place the trayectories and the trayectories itself.

The program reads a file that describes the object to be polished using the definition of VRML. Both VRML 1.0 and VRML 2.0 can be used for the description of the object, taking in account that only Mesh-definitions are permitted. The mesh consist of individual triangles that are defined by the position of its vertices. With this information the position and orientation of the triangle can be obtained. The triangles altogether then form the object that is considered the workpiece.

After the workpiece is located in the virtual space the mouse is used to place the trayectories. The visualization of a trayectory consists of two parts: First the path that the robot has to pass the workpiece along the polishing belt, indicated by a thin line on the surface of the object. Second a wide line that represents the resulting surface that is achieved by passing the workpiece along the polishing belt. The second line is a projection inside the object beneath the line that indicates the path of the object along the polishing belt. The

parameters for pressure and width determine the depth under the surface of the object and the width of the line.

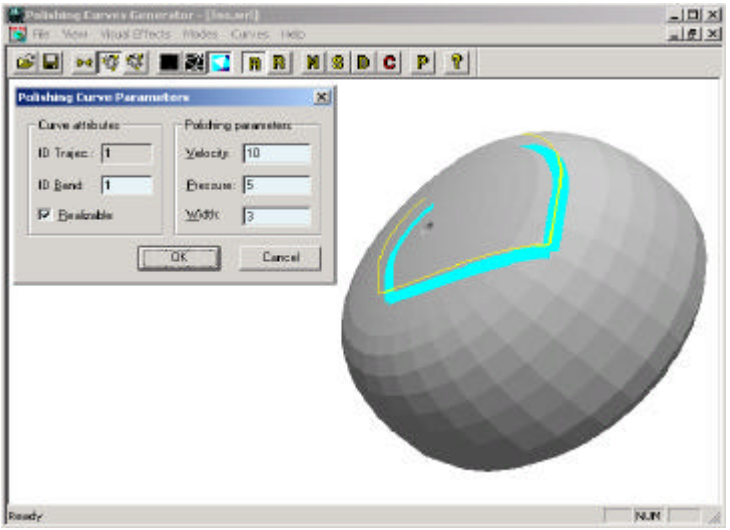



**Figure 1. Program "PCG" utilizing the mouse pointer.**

The positioning of the polishing trayectories can be divided into two types. Once the parameters are set, a series of connected trayectories can be placed, all using the same set of parameters. This series then can be deselected and a new series of trayectories can be positioned with a different set of parameters, even though the values of the parameters might habe the same values.

The program also provides different operating modes for the visualization of the object. The normal operating mode is the mode in which the object is drawn by showing the individual triangles. Besides that a mode exist in which only the vertices of the triangles are shown and also a mode in which only the edges of the triangles are shown. In any case the trayectories are displayed in the same way.

In the case that the definition of the object in VRML results in the wrong orientation of the triangles, the triangles can be displayed using counter-clockwise visualization of the object.

As part of the positioning of the polishing trayectories, the program offers a variety of post-processing of the trayectories. Once several series of trayectories are positioned it is possible to return to the parameters of each of the series, changing the values for velocity, pressure or width. However, it is not possible to change the position of one or more trayectories once they are placed. In this case it is indispensable to replace the trayectory by a new one.

Finally the application permits to store the information of the positioned trayectories in two files with the values for position and the parameters of the trayectories. The basic difference between the two files is the format to store the information. These files then can be used to upload to a robot accomplishing its task of polishing the real workpieces.

PART 3

# *Haptic device*

The Application PCG is programed to utilize the mouse of the computer in order to position the trayectories depending on a set of parameters entered by the keyboard. For the operator it may not be adequate to enter a value of the force that the robot then will apply to the workpiece. It would be ideal to give the operator the possibility to actually feel the force which he wants the robot to press the workpiece against the polishing belt. This sensation of feeling the applied force can be realized by a **haptic device**.

## *3.1 Definition*

The word haptic is of greek origin [greek: háptein] and describes the tactile sense. It is the physical interaction via touch which nowadays mainly takes in consideration touching and interacting with virtual and remote environments. It is used as an addition to optical and acoustical output devices in various fields of applications.

The haptic devices already can be found in areas of Education, Medicine, Entertainment and of course in the industry. Various of the following tasks are already performed using haptic devices:

- Amplification of vision for teleoperation in surgery,Rehabilitation, Computer games, CAD-designs and construction, teleoperation, simulation of virtual environments, position and force scaling, assistance in guided remote equipment, and many more.

The haptic devices form a man-machine-interface which gives different possibilities to the so far utilized interfaces. Amongst these mouse, keyboard, speakers and of course the monitor. With the haptic devices it is possible to enter in a simulation of an authentic world with the advantage of actually feeling the movements and applying forces by the intuitive sense of the operator.

## *3.2 PHANToM*

The haptic interface used in the presented work is the **PHANToM™ 1.5/6DOF** manufactured by the company SensAble Technologies. This haptic device was developed at the Massachusetts Institute of Technology (MIT) and provides the ability to operate in an office/desktop environment, compatibility with standard PCs and a universal design for a

broad range of applications. The operating system may be Microsoft Windows or Linux. Several models of the PHANToM with different features are available.

The PHANToM™ 1.5/6DOF has the ability to sense the position and the orientation (3+3 degrees of freedom) of the handle and also to feedback the three forces and the three torques, indicating a collision of the haptic pointer with the virtual object.The workspace that provides the PHANToM 1.5 is 19.1 x 26.7 x 38.1 cm, or the range of motion approximate to the lower arm pivoting at the user's elbow.

## *3.3 GHOST SDK*

Together with the PHANToM the manufacturer provides a package of software libraries called GHOST SDK (General Haptic Open Software Development Toolkit) which is an object-oriented toolkit, written in $C^{++}$, containing classes and methods for support of the haptic interface PHANToM. It represents the haptic environment as a hierarchical collection of geometric objects and spatial effects (buzzing, inertia, viscosity, etc.). GHOST-libraries support triangular meshes which are used by the existing PCG-application.

PART 4

# *Implementation of PHANToM*

For the implementation of the PHANToM to the existing code of the PCG-Application a series of changes had be included. In this partition first the structure of the source code will be explained and then the necessary changes.

## *4.1 Structure of source-code*

The original application is generated with the MFC (Microsoft Foundation Classes) and which forces a fixed structure of coding. The Application then can be separated into four parts resulting from the original code: The Mainframe-part, the application-part, the document-part and the view-part. Additionally the part of the PHANToM had to be included. The following table shows the structure of the application with the new part for the PHANToM:

| | **h-File** | **c/cpp-File** |
|---|---|---|
| Mainframe: | mainfrm.h | mainfrm.cpp |
| Application: | pcg.h | pcg.cpp |
| Document: | pcgdoc.h | pcgdoc.cpp |
| | wrl2mesh.h | wrl2mesh.c |
| | graph.h | graph.c |
| View: | pcgview.h | pcgview.cpp |
| **PHANToM:** | **template.h** | **template.cpp** |
| | **template_graphics.h** | **template_graphics.cpp** |
| | **template_haptics.h** | **template_haptics.cpp** |

TABEL 1. **Structure of the source-code with additional part for the PHANToM.**

## *4.2 Implementations in source code for PHANToM*

Enabling the PHANToM requires a range of implementations to the program. First of all the type of document had to be changed from MDI (Multi-Document-Interface) to SDI (Single-Document-Interface). In continuation the implemented classes had to be changed which are used by the MFC. The GHOST-SDK uses a derivation of these fundamental classes for the document-part, view-part and application-part with additional functionality to be able to utilize the PHANToM.

### 4.2.1 Change from MDI to SDI Application

The libraries which are utilized allow the user to use the graphical-environment that works with a single document. The existing program is using a multiple document environment, although the advantages are not used. This change makes it necessary to modify the file pcg.cpp. The fundamental change takes place where the document-part gets linked with the view-part and the mainframe-part. The considered code is the following:

| MDI-code: | SDI-code: |
|---|---|
| C**MultiDoc**Template* pDocTemplate;<br>pDocTemplate = new C**Multi**DocTemplate(<br>IDR_VRML1TYPE,<br>RUNTIME_CLASS(CPCGDoc),<br>RUNTIME_CLASS(C**ChildFrame**),<br>RUNTIME_CLASS(CPCGView));<br>AddDocTemplate(pDocTemplate); | C**Single**DocTemplate* pDocTemplate;<br>pDocTemplate = new C**Single**DocTemplate(<br>IDR_VRML1TYPE,<br>RUNTIME_CLASS(CPCGDoc),<br>RUNTIME_CLASS(C**MainFrame**),<br>RUNTIME_CLASS(CPCGView));<br>AddDocTemplate(pDocTemplate); |

**TABEL 2. Fundamental change in source-code to pass the application from a MDI (Multi Document Interface) to a SDI (Single Document Interface).**

Furthermore the files Childfrm.cpp together with Childfrm.h are no longer used and the corresponding include-call is removed. The change from MDI to SDI also implements some minor changes to the menu-bar and the tool-bar in order to disable some of the functions while no workpiece is displayed and to disable the ability to load another workpiece while one is already loaded.

### 4.2.2 Derivation of MFC-classes

The classes used by the MFC in principle are also used by the GHOST-SDK. The difference is that some functionality for the PHANToM has to be included. Therefore the GHOST-SDK uses its own set of classes with a derivation of the ones used by the MFC. The change of classes are the following:

| | Derivation of MFC-classes | include h-file: |
|---|---|---|
| Mainframe: | class CMainFrame : public CMDIFrameWnd | |
| | class CMainFrame : public **CHapticFrame** | HapticFrame.h |
| Application: | class CPCGApp : public CWinApp | |
| | class CPCGApp : public **CHapticApp** | HapticApp.h |
| Document: | class CPCGDoc : public CDocument | |
| | class CPCGDoc : public **CHapticDoc** | HapticDoc.h |
| View: | class CPCGView : public CView | |
| | class CPCGView : public **CHapticView** | HapticView.h |

**TABEL 3. Change of classes to use the functionality necessary for the GHOST-libraries.**

The derived classes for the GHOST-SDK then require a set of functions that are called during the execution of the application.

### 4.2.3 Required functions for PHANToM (pcgview.cpp/h)

The haptic device PHANToM requires a series of functions which are called by the application during execution. These functions are to be included into the files that handle the visualization of the application. The files are pcgview.cpp and pcgview.h and the required functions are the following:

```
void    StartProgram(BOOL);
void    InitGraphics();
void    EnableServoLoop(BOOL bEnable);
void    UpdateGraphics();
void    ResizeGraphics(int cx, int cy);
void    EndProgram();
BOOL    ProgramDone();
void    TermGraphics();
LPCSTR* QueryPHANToMNames();
```

All these functions are required because they are called at a certain instance during the execution of the application. The functions “StartProgram” and “InitGraphics” are called at the very beginning of the execution and this is where the virtual scene will be defined. Here also the initialization of the existing OpenGL part is implemented with the function “Init_GL“.

The function “EnableServoLoop” is used to give the impulse to start or to end the servo-loop which is necessary for generating the loops in which the information of the haptic device gets updated.

This servo-loop then calls the function “UpdateGraphics“ where all the functions are to be placed that require permanent updates. In this case it is the tracking of the position and the forces of the haptic device and also the rendering of the graphical part of the application.

The function “ResizeGraphics“ is called every time the screen is resized, which also occurs at the beginning of the execution.

For the termination of the application the three functions “EndProgram” and “Program-Done” and “TermGraphics“ are performed. Here the haptic device and the scene will be terminated and the occupied memory will be released.

### 4.2.4 Interface to haptic device (Template.cpp/h)

The file template.cpp together with template.h works as an interface between the view-part of the application (pcgview) and the part of the haptic device. It contains the definition of the scene and the haptic device which is necessary for the generation of the virtual

environment. During the initialization of the virtual scene a series of important include files are required:

```
#include "gstScene.h"
#include <ghostGLSyncCamera.h>
#include "GL.h"
```

The file “gstScene.h“ contains the functionality that provides the handling of the scene. This is the basic unit which then gets loaded with objects like the haptic device, the workpiece and the trayectories. Also the file “ghostGLSyncCamera.h“ is necessary which enables certain functionality together with the functions of OpenGL. Finally the file “GL.h” has to be included for the ability to use commands of OpenGL.

### 4.2.5 Visualization with ghostGLManager (Templates_graphics.cpp/h)

As for the graphical part of the virtual environment it is necessary to load the ghostGL-Manager. This manager makes it possible to visualize the scene and all of its contents. Including the file “ghostGLPinchXform.h“ gives additional functionality to the visualization which is the possibility to move the camera position within the scene. This can be achieved by pressing constantly the button on the handle of the PHANToM and displacing or rotating the handle meanwhile. The resulting effect is that the user displaces and rotates the entire virtual scene, in other words the operator changes the position and orientation of the camera

The function “update_graphics“, initiated by the view-part, makes sure that the virtual scene will be redrawn and also the position of the PHANToM is updated and send to the part of the program where the trayectories, programmed in OpenGL, are processed. In this part of the program also the termination of the ghostGLManager can be found, initiated by the view-part, when closing the application.

### 4.2.6 Haptic environment (Templates_haptics.cpp/h)

In this part of the program the information is processed which is in direct relation to the haptic device. Once the scene is initialized, it can be loaded with different “separators” (term according to GHOST-SDK). One of the separators is the haptic device which gets initialized in this partition.
The workpiece also has to be attached to the virtual environment and unlike the trayectories, the workpiece has to be rigid and touchable. The workpiece, defined in VRML, can be added as a separator to the virtual environment, just like the haptic device.

The additional functionality implemented in this part is the starting and the termination of the servo-loops necessary for the haptic device and the virtual environment in order to obtain information on position and orientation of the handle and also to feedback the considered forces and torques, if they exist. In the presented work only the position and

the corresponding forces of the handle returned to the electrical motors of the PHANToM are of interest.

The GHOST-libraries permit to obtain the position of the PHANToM in the virtual environment by calling the function “getSCP_P”. The abbreviation SCP signifies Surface-Contact-Point. The function for obtaining the feedback force can be obtained with the function “getReactionForce_WC“. The abbreviation stands for World-Coordinates.
Once the position and the forces are determined they can be passed to the part where the application handles the trayectories (glcode.cpp) to be placed, depending on the position and the applied force of the haptic device. This is achieved by the function-call “Pass_Haptic_Position“ which contains the arguments position and forces.

In the program PCG the contact point of the object with the mouse-pointer is obtained by comparing the position of the mouse in 2D-coordinates with the projection of the workpiece to screen-coordinates. In the original program a routine is implemented, using OpenGL, that returns the identity of the triangle over which the mouse is located. The GHOST-libraries do not support any 2D values and therefore it is not possible to use the implemented functionality. A new function (“GetTriangleIdent”) had to be implemented surrounding each triangle with a virtual box and then comparing if the actual position of the haptic device is within this surrounding box. With this information and the fact that the applied force is greater than zero the processing is continued in the original way.

As a next step a function (“GLRenderProvBelt“) was implemented that projects a rectangle with the length of the edges equal to the width of the trayectory to be placed. This rectangle then indicates the operator in which depth the trayectory will be placed. The bigger the value of the applied force, the deeper the rectangle and the resulting trayectory will be placed.

In the original program it was only possible to use the left mouse button for the confirmation where to locate a certain trayectory. It is not very convenient to work with the mouse and the PHANToM at the same time. For convenience the confirmation to position a trayectory now also can be done with the space bar or the return key of the keyboard.

With these implemented functions the program PCG is fully functional with the haptic device PHANToM.

## *4.3 Problems in original program*

During the implementation of the functionality of the haptic device PHANToM it was discovered that the original program was not fully functional. Occasionally the program does not respond properly. When a set of trayectories is placed, stored to the pcg-file and then reloaded, the trayectories not always were reloaded correctly. The problem was found in two instances, during the calculation of angle “phi” of the trayectories. A division was used that occasionally has a cero in the denominator. The program, however,

does not interrupt but continues with an undefined value “-1.#IND“. Which results in a wrong execution of the program. This problem only occured when saving and recovering a set of trayectories. As a solution for the problem the values of the respective variables are checked and set to a defined value.

PART 5

# User´s Guide

The presented work is based on the existing software program that handles and administrates the workpiece and the applied trayectories. The software implementation of the additional functionality in order to be able to use the haptic device PHANToM was described in the last partition. In this partition the impact on the utilization of the program is described.

## 5.1 General Note

First of all it has to be highlighted that at the very beginning it might seem difficult to handle the haptic device PHANToM. This is due to the additional dimension that the haptic device provides, compared to the mouse, and due to the possibility to rotate the entire scene with the handle. Therefore it is necessary to get familiarized with its use.

The functionality provided by the original program, using the mouse to rotate and move the workpiece, no longer exists and is replaced by the functionality of the haptic device PHANToM.

## 5.2 Setup and initialization of the PHANToM

Before the program can be used it is indispensable to connect the PHANToM to the parallel port of the computer, connect the supply cable and switch on the two switches on the back-side of the chassis.

The next step is to start the program “PCGHaptic.exe“, described in this document. Once the program is executed, a screen will appear (see Figure2).

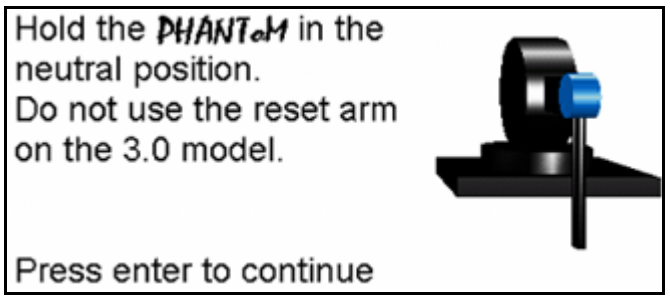



**Figure 2. Initial screen, move the handle of the PHANToM into the neutral position and press “Enter” (”Return”).**

The neutral position of the handle of the PHANToM can be reached by elevating the handle by 10 to 15cm above the table. This position is indicated by the dashed lines in Figure3. The handle also has to be orientated horizontally facing away from the chassis. Once the PHANToM is initialized, the handle can be placed on the table again.

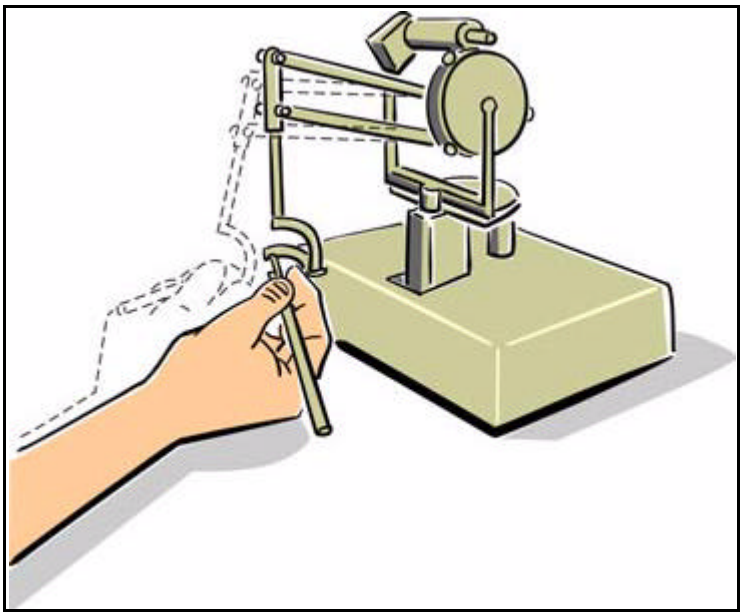


**Figure 3. Neutral position of the haptic indicated by the dashed lines.**

Now it is necessary to stroke the key “Enter” (“Return”) to store this position as reference point. Once the PHANToM is initialized, the handle can be placed on the table again.

## *5.3 Loading the workpiece*

The next step is to load a workpiece, defined in VRML. The option “Open Workpiece” can be found in the menu under “File”. Once the workpiece is selected and loaded, the functionality of the program does only vary when applying the pressure to a certain trayectory and performing a rotation or displacement of the scene.

## *5.4 Rotation and displacement of the scene*

Once the workpiece is loaded to the scene it is possible to rotate and displace it together with the entire scene. This can be achieved by pressing the button that can be found on the handle of the PHANToM. It is necessary to press the button constantly while rotating or displacing the handle.

## *5.5 Placing trayectories*

The main difference in the process of placing the trayectories is that in the dialog “Create a new curve“ it is not necessary to enter a value for “Pressure”. The value for “Pressure” then will be determined by the applied force of the PHANAToM at the beginning of the trayectory.

Like in the original program the trayectories can be placed by stroking the left mouse button. However, naturally a person that is right hander uses the right hand to handle the PHANToM (vice versa for left hander) and is familiar with the use of the mouse with his right hand as well. Handling the PHANToM with the right hand it was found inconvenient to use the left hand to stroke the left mouse button in order to place the trayectories. Therefore this function was extended to the two keys “Space” and “Return” on the keyboard.

Like in the original program, all the following subtrayectories are placed with the same value for “Pressure”, only choosing a new trayectory will allow the user to apply a different value for “Pressure”.

The subsequent process of saving the trayectories and also the format of the files containing the information of the trayectories is performed in the same way as it was implemented in the original program.

PART 6

# Appendix

## 6.1 Source-Code

The following source code was added into the PCG-Application in order to implement the functionality of the haptic device PHANToM:

### 6.1.1 Template.h

```
//==============================================================================
//   Filename   : Template.h
//   Written by : Gerrit Färber
//   Project    : Template Ghost Application
//   Module     : Platform Independent Application Entry
//==============================================================================

#ifndef TEMPLATE_H
#define TEMPLATE_H

#ifdef _WIN32
// Disable data conversion warnings
#pragma warning(disable : 4305)     // X86
#endif

//==============================================================================
//                        Global Variables / Constants / Include-files
//==============================================================================
#define   PHANTOM_NAME    "Default PHANToM"  // PHANToM configuration string
#include "gstScene.h"
#include <ghostGLSyncCamera.h>
#include "GL.h"
//==============================================================================
//                               Function
//==============================================================================
void   start_program(int bResetPHANToM);
void   end_program(void);

static gstScene *myScene = NULL;
static gstSphere *mySphere;
static gstPHANToM *myPHANToM;
static ghostGLSyncCamera  *m_camera;
static gstBoundaryCube    *m_gstWorkspace;
static gstSeparator *rootSep;
static BOOL RenewInc;

#endif // TEMPLATE_H
```

### 6.1.2 Template.cpp

```
//==========================================================================
//   Filename   : Template.cpp
//   Written by : Gerrit Färber
//   Project    : Template Ghost Application
//   Module     : Platform Independent Application Entry
//==========================================================================

#include "Template.h"
#include "Template_haptics.h"
#include "Template_graphics.h"

void start_program (int bResetPHANToM)
{   myScene = new gstScene();// Create a shared static instance of the scene
    myScene = init_haptics(myScene, bResetPHANToM);// Initialize scene graph
    init_graphics(myScene);// Now force init_graphics to load the scene graph
}

void end_program (void)
{   // Perform whatever cleanup needs to be done
    if (myScene)
        delete myScene;
}
```

### 6.1.3 Template_graphics.h

```
//==========================================================================
//   Filename   : Template_graphics.h
//   Written by : Gerrit Färber
//   Project    : Template Ghost Application
//   Module     : Platform Independent Graphics
//==========================================================================

void    init_graphics(gstScene *pScene = NULL);
void    term_graphics(void);
void    reshape(GLint width, GLint height);
void    update_graphics(void);
void    CheckIfVRMLOpen2(BOOL);
void    OnGoHome();
void    MultiplyMatrixPoint();

static  double XHapticPosCurr;
static  double YHapticPosCurr;
static  double ZHapticPosCurr;
static  double XHapticPosPrev;
static  double YHapticPosPrev;
static  double ZHapticPosPrev;
static  double XHapticForce;
static  double YHapticForce;
static  double ZHapticForce;
static  BOOL   m_VRML_FileOpen;
static  double TransMatrix[4][4];
static  double PointInput[4];
static  double PointOutput[4];
static  gstTransformMatrix CameraTransMatrix;

#endif // TEMPLATE_GRAPHICS_H
```

### 6.1.4 Template_graphics.cpp

```
//==========================================================================
//   Filename   : Template_graphics.cpp
//   Written by : Gerrit Färber
//   Project    : Template Ghost Application
//   Module     : Platform Independent Graphics
//==========================================================================


#include "stdafx.h"
#include "Template_graphics.h"
#include "Template_haptics.h"
#include "Template.h"
#include "glcode\glcode.h"

#include <ghostGLSyncCamera.h>
#include <ghostGLManager.h>
#include <ghostGLPinchXform.h>

static ghostGLManager  *myGLManager = NULL;
static gstBoundaryCube *workspaceBounds = NULL;
static ghostGLSyncCamera *myCamera;


void init_graphics (gstScene *pScene)
{
    RenewInc=0;
    m_VRML_FileOpen2=FALSE;

    if (myGLManager && pScene)
        myGLManager->loadScene(pScene);

    if (!myGLManager)
    {
        myCamera= new ghostGLSyncCamera();
        myGLManager = new ghostGLManager(myCamera);

        // setup the pinch transform = movement camera
        myCamera->setSyncMode(ghostGLSyncCamera::SYNC_WORKSPACE_TO_CAMERA);
        static ghostGLPinchXForm *pinchXFormObj = new ghostGLPinchXForm();
        myGLManager->addActionObject(pinchXFormObj);
    }
}


void term_graphics (void)
{   if (myGLManager)
    {
        delete myGLManager;
        myGLManager = NULL;
    }
}
```

```
void reshape (GLint width, GLint height)
{
    if (myGLManager)
        myGLManager->reshape(width, height);
}


void update_graphics (void)
{
    gstPoint CameraPos;
    int    i,j;

    RenewInc++;
    // Graphics updated two out of three cycle in order
    // to permit the toolbar to update
    if (myGLManager && (RenewInc>1 || m_VRML_FileOpen2==0))
    {
        if(RenewInc==2)
            RenewInc=0;
        myGLManager->redraw();

        // Read Haptic: Position and Force and store
        // current position to previous position
        XHapticPosPrev = XHapticPosCurr;
        YHapticPosPrev = YHapticPosCurr;
        ZHapticPosPrev = ZHapticPosCurr;

        XHapticPosCurr=query_phantom_Xpos();
        YHapticPosCurr=query_phantom_Ypos();
        ZHapticPosCurr=query_phantom_Zpos();
        XHapticForce=query_phantom_Xforce();
        YHapticForce=query_phantom_Yforce();
        ZHapticForce=query_phantom_Zforce();

        if(m_VRML_FileOpen2==1)
        {
            // get camera position and orientation in order
            // to rotate and move haptic-position
            CameraTransMatrix =  myCamera->getTransformMatrix(FALSE);
            for(i=0;i<4;i++)
             for(j=0;j<4;j++)
                 TransMatrix[i][j] = CameraTransMatrix._a.m_elements[i][j];

            // rotation and displacement of haptic position
            // due to camera position
            PointInput[0]  = XHapticPosCurr;
            PointInput[1]  = YHapticPosCurr;
            PointInput[2]  = ZHapticPosCurr-350;
            PointInput[3]  = 1;
            MultiplyMatrixPoint();
            XHapticPosCurr=PointOutput[0];
            YHapticPosCurr=PointOutput[1];
            ZHapticPosCurr=PointOutput[2];
        }
        // function to pass position and forces to GL-code
        Pass_Haptic_Position(XHapticPosCurr, YHapticPosCurr, ZHapticPosCurr,
                  XHapticForce,   YHapticForce,   ZHapticForce);
    }
}
```

```
void MultiplyMatrixPoint(void)
{
    int i;

    // Multiplication of Matrix with Point
    // Result is rotation and movement of point
    for(i=0;i<4;i++)
        PointOutput[i]=  TransMatrix[0][i]*PointInput[0]
                 +TransMatrix[1][i]*PointInput[1]
                 +TransMatrix[2][i]*PointInput[2]
                 +TransMatrix[3][i]*PointInput[3];
}


void CheckIfVRMLOpen2(BOOL indicator)
{
    // check if VRML-file open
    m_VRML_FileOpen2=indicator;
    if(m_VRML_FileOpen2==1)
    {
        m_VRML_FileOpen2=m_VRML_FileOpen2;
    }
}


void OnGoHome(void)
{
    int i,j;

    // get camera orientation and position
    CameraTransMatrix =  myCamera->getTransformMatrix(FALSE);

    // write all cells to zero
    for(i=0;i<4;i++)
     for(j=0;j<4;j++)
     {
        CameraTransMatrix._a.m_elements[i][j]=0;
        CameraTransMatrix._aInv.m_elements[i][j]=0;
     }

    // set diagonal to ones
    for(i=0;i<4;i++)
    {
        CameraTransMatrix._a.m_elements[i][i]=1;
        CameraTransMatrix._aInv.m_elements[i][i]=1;
    }

    // set coordinates to 0/0/350
    CameraTransMatrix._a.m_elements[3][2]=350;
    CameraTransMatrix._aInv.m_elements[3][2]=-350;
    // set orientation and position of camera
    myCamera->setTransformMatrix(CameraTransMatrix, FALSE, TRUE);
}
```

### 6.1.5 Template_haptics.h

```
//==============================================================================
//   Filename   : Template_haptics.h
//   Written by : Gerrit Färber
//   Project    : Template Ghost Application
//   Module     : Platform Independent Haptics
//==============================================================================

#ifndef TEMPLATE_HAPTICS_H
#define TEMPLATE_HAPTICS_H
#include "Template.h"

gstScene *init_haptics(gstScene *pScene = NULL, int bResetPHANToM = TRUE);
void   Read_VRML_Haptic(const char *FileName);
void   enable_servo_loop(int bEnable);
int    scene_done(void);
int    query_phantom_pos(double *px, double *py, double *pz);
double query_phantom_Xpos();
double query_phantom_Ypos();
double query_phantom_Zpos();
double query_phantom_Xforce();
double query_phantom_Yforce();
double query_phantom_Zforce();
BOOL   CheckIfVRMLOpen();

static double rotation_angle1;
static gstBoundaryCube *workSpaceBounds;
static gstBoolean bIs6DOF = FALSE;
static double XFce;
static double YFce;
static double ZFce;
static gstPoint position;
static BOOL m_VRML_FileOpen2;// This flag is set when mesh loaded into memory
static BOOL m_VRML_FileOpen2;
#endif // TEMPLATE_HAPTICS_H
```

### 6.1.6 Template_haptics.cpp

```
//==============================================================================
//   Filename   : Template_haptics.cpp
//   Written by : Gerrit Färber
//   Project    : Template Ghost Application
//   Module     : Platform Independent Haptics
//==============================================================================

#include "stdafx.h"
#include "Template_haptics.h"
#include "Template_graphics.h"
#include "gstTransform.h"
#include "Template.h"
#include <gstVRML.h>
#include <GL/gl.h>
#include <GL/glu.h>

// True if servo loop has been suspended with enable_servo_loop()
static gstBoolean bSuspended = FALSE;
static gstSeparator *phantomSep, *geomSep;
```

```
gstScene *init_haptics (gstScene *pScene, int bResetPHANToM)
{
    myScene = new gstScene();
    rootSep = new gstSeparator();

    phantomSep = new gstSeparator();
    geomSep = new gstSeparator();

    myScene->setRoot(rootSep);
    rootSep->addChild(phantomSep);
    rootSep->addChild(geomSep);

    // Create the phantom object. When this line is
    // processed, the phantom position is zeroed
    myPHANToM = new gstPHANToM(PHANTOM_NAME, bResetPHANToM);
    if (!myPHANToM->getValidConstruction()) {
        cerr << "Failure to create a valid construction." << endl;
        exit(-1);
    }
    phantomSep->addChild(myPHANToM);
    return myScene;
}

void Read_VRML_Haptic (const char *FileName)
{
    gstSeparator *vrmlSep = gstReadVRMLFile(FileName);
    vrmlSep = gstReadVRMLFile(FileName);
    rootSep->addChild(vrmlSep);
    m_VRML_FileOpen=TRUE;// This flag is set when mesh loaded into memory

    // reset Camera orientation and position
    OnGoHome();
}

void enable_servo_loop (int bEnable)
{
    if (bEnable)
        myScene->startServoLoop();
    else
        myScene->stopServoLoop();

    bSuspended = !bEnable;
}

int scene_done (void)
{
    return (!bSuspended && myScene->getDoneServoLoop());
}

int query_phantom_pos (double *px, double *py, double *pz)
{
    position = myPHANToM->getPosition_WC();

    *px = position.x();
    *py = position.y();
    *pz = position.z();

    return TRUE;
}
```

```
// Surface-Contact-Point of PHANToM
double query_phantom_Xpos ()
{
    myPHANToM->getSCP_P(position);
    return position.x();
}

double query_phantom_Ypos ()
{
    myPHANToM->getSCP_P(position);
    return position.y();
}

double query_phantom_Zpos ()
{
    myPHANToM->getSCP_P(position);
    return position.z();
}

double query_phantom_Xforce ()
{   gstVector force = myPHANToM->getReactionForce_WC();
    return force.x();
}

double query_phantom_Yforce ()
{   gstVector force = myPHANToM->getReactionForce_WC();
    return force.y();
}

double query_phantom_Zforce ()
{   gstVector force = myPHANToM->getReactionForce_WC();
    return force.z();
}
```

### 6.1.7 glcode.c (Function GetTrianleIdent)

```
PRIVATE int GetTriangleIdent()
{
    int ntriangs, ui, v0;
    int idTriangle = -1;
    double maxX, minX, maxY, minY, maxZ, minZ;

    int test=0;
    ntriangs = m.numTriangs;

    for (ui=0; ui<ntriangs; ui++)
    {
        //get min and max values of triangle
        minX = maxX = m.pVerts[m.pTriangs[ui].vert[0]].x;
        minY = maxY = m.pVerts[m.pTriangs[ui].vert[0]].y;
        minZ = maxZ = m.pVerts[m.pTriangs[ui].vert[0]].z;
        for (v0=1; v0<3; v0++)
        {   if(m.pVerts[m.pTriangs[ui].vert[v0]].x < minX)
                minX = m.pVerts[m.pTriangs[ui].vert[v0]].x;
            if(m.pVerts[m.pTriangs[ui].vert[v0]].x > maxX)
                maxX = m.pVerts[m.pTriangs[ui].vert[v0]].x;
            if(m.pVerts[m.pTriangs[ui].vert[v0]].y < minY)
                minY = m.pVerts[m.pTriangs[ui].vert[v0]].y;
            if(m.pVerts[m.pTriangs[ui].vert[v0]].y > maxY)
                maxY = m.pVerts[m.pTriangs[ui].vert[v0]].y;
            if(m.pVerts[m.pTriangs[ui].vert[v0]].z < minZ)
                minZ = m.pVerts[m.pTriangs[ui].vert[v0]].z;
            if(m.pVerts[m.pTriangs[ui].vert[v0]].z > maxZ)
                maxZ = m.pVerts[m.pTriangs[ui].vert[v0]].z;
        }

        for (v0=0; v0<3; v0++)
        {
            if(  (XPos_Hapt>minX && XPos_Hapt<maxX) &&
                 (YPos_Hapt>minY && YPos_Hapt<maxY) &&
                 (ZPos_Hapt>minZ && ZPos_Hapt<maxZ))

                 idTriangle = ui;
        }
    }
    if(idTriangle == -1)
        idTriangle = idTnglast;
    else
        idTnglast = idTriangle;

    return(idTriangle);
}
```

### 6.1.8 glcode.c (Function GLRenderProvBelt)

```
PRIVATE void GLRenderProvBelt(double depth)
{
double VX     , VY     , VZ;
double VXNorm1, VYNorm1, VZNorm1, VabsNorm1;
double VXNorm2, VYNorm2, VZNorm2, VabsNorm2;

    // V equal to the Directino of the Force of Haptic
    VX = (XForce_Hapt+0.000000001)/2;
    VY = (YForce_Hapt+0.000000002)/2;
    VZ = (ZForce_Hapt+0.000000003)/2;
    // Calculation of Normal to Direction of Force of Haptic
    VXNorm1 = (VY-VZ);
    VYNorm1 = (VZ-VX);
    VZNorm1 = (VX-VY);
    VabsNorm1 = sqrt(pow(VXNorm1,2)+pow(VYNorm1,2)+pow(VZNorm1,2));
    VXNorm1 = VXNorm1/VabsNorm1/2*4/3;
    VYNorm1 = VYNorm1/VabsNorm1/2*4/3;
    VZNorm1 = VZNorm1/VabsNorm1/2*4/3;
    // Calculation of Second Normal, perpendicular to
    // Direction of Force of Haptic and to first Normal
    VXNorm2 = (VY*VZNorm1-VZ*VYNorm1);
    VYNorm2 = (VZ*VXNorm1-VX*VZNorm1);
    VZNorm2 = (VX*VYNorm1-VY*VXNorm1);
    VabsNorm2 = sqrt(pow(VXNorm2,2)+pow(VYNorm2,2)+pow(VZNorm2,2));
    VXNorm2 = VXNorm2/VabsNorm2/2*4/3;
    VYNorm2 = VYNorm2/VabsNorm2/2*4/3;
    VZNorm2 = VZNorm2/VabsNorm2/2*4/3;


    glNewList(BAND_LIST, GL_COMPILE);
    glBegin(GL_QUADS);

        // Provisional Belt indicating pressure
        glNormal3f(0.0, 0.0, 1.0);
        glVertex3f( (float)(XPos_Hapt-VX+VXNorm1*depth),
                    (float)(YPos_Hapt-VY+VYNorm1*depth),
                    (float)(ZPos_Hapt-VZ+VZNorm1*depth)); // back bottom left
        glVertex3f( (float)(XPos_Hapt-VX+VXNorm2*depth),
                    (float)(YPos_Hapt-VY+VYNorm2*depth),
                    (float)(ZPos_Hapt-VZ+VZNorm2*depth)); // back bottom left
        glVertex3f( (float)(XPos_Hapt-VX-VXNorm1*depth),
                    (float)(YPos_Hapt-VY-VYNorm1*depth),
                    (float)(ZPos_Hapt-VZ-VZNorm1*depth)); // back bottom left
        glVertex3f( (float)(XPos_Hapt-VX-VXNorm2*depth),
                    (float)(YPos_Hapt-VY-VYNorm2*depth),
                    (float)(ZPos_Hapt-VZ-VZNorm2*depth)); // back bottom left

    glEnd();
    glEndList();
}
```

### 6.1.9 glcode.c (Function Pass_Haptic_Position)

```
void Pass_Haptic_Position(double Xpos_Haptic  , double Ypos_Haptic  ,
                         double Zpos_Haptic  , double Xforce_Haptic,
                         double Yforce_Haptic, double Zforce_Haptic)
{
    XPos_Hapt   = Xpos_Haptic;
    YPos_Hapt   = Ypos_Haptic;
    ZPos_Hapt   = Zpos_Haptic;

    XForce_Hapt = Xforce_Haptic;
    YForce_Hapt = Yforce_Haptic;
    ZForce_Hapt = Zforce_Haptic;

    Force_Hapt  = sqrt(pow(XForce_Hapt,2) +
                       pow(YForce_Hapt,2) +
                       pow(ZForce_Hapt,2));
}
```

### 6.1.10 glcode.c (Parts of function GLRenderBoxes)

```
    if(it==nt-1)
    {   if(IndChangeOfParam==0)
            h=Force_HaptStore;
        m.pTrajs[it].params.pressure=(float)h;

        GLRenderProvBelt(d); // Introduced for Haptic

        if(Force_Hapt > 0)
            glCallList(BAND_LIST);
    }
```

## *6.2 Bibliography*

**The following bibliography was used during the processing of the project:**

Company and Product-Information, SensAble Technologies Inc.
http://www.sensable.com/

GHOST API Reference Guide for v4.0, SensAble Technologies Inc.
http://www.sensable.com/support/phantom_ghost/datafiles/GHOSTAPIReferenceManual.pdf

Nemcova D., April 2000, Haptic interface, Masaryk University, Czech republic, Faculty of Informatics.
http://www.cg.tuwien.ac.at/studentwork/CESCG/CESCG-2000/DNemcova/index.html

Certec, Non-Visual Haptic Interaction Design. Consulted on 01.07.2001
http://www.certec.lth.se/doc/hapticinteraction/

Palomo L., 2003, Internal document IOC, Barcelona, Spain, "Desenvolupament dúna eine per a la generació de trajectòries i la programació de robots per tasques de pulit"

Gregory K., 1997, Visual $C^{++}5$ Referenzen und Anwendungen, Que-Verlag.

Papula L., 1994, II-Vektorrechnung, In: Mathematische Formelsammlung für Ingenieure und Naturwissenschaftler, Vieweg, pp.41-55