Appendix

Using the NaturalPoint Tracking System

Appendix

Online Documentation

You can find the complete online documentation online by following this link:

http://www.naturalpoint.com/optitrack/support/manuals/trackingtools/tt-index.html

However, for our projects, we only need a small subset of its functionality. The following slides describe the essential functions needed for tracking in the Visualization classroom in Russ 154A.

Appendix

Initial Setup

The tracking systems only supports the Windows operating system at this point. In order to use it in Visual Studio, you will have to include its header file:

```
#include "NPTrackingTools.h”
```

This file is found in `C:\Program Files\NaturalPoint\TrackingTools\inc`, which you will have to add to the include directories. Similarly, you will have to add the library NPTrackingTools.lib from the directory `C:\Program Files\NaturalPoint\TrackingTools\lib` to your project.

Appendix

Initialization

The cameras are already calibrated and a calibration file is available in your `My Documents` folder. To start the tracking system all you need to do is issue the following command in your code:

```
TT_Initialize( )
```

You should see numbers shown on the LED displays on the cameras after you issued this command (assuming you got `NPRESULT_SUCCESS` as return value).

Appendix

Calibration

The camera system is already calibrated so that you do not have to perform that step (unless you move the cameras). To load the provided calibration, simply use this command with the calibration file as the parameter:

```
NPRESULT TT_LoadCalibration(const char *filename)
```

Appendix

Trackables

Similarly, you can load the file defining the trackables:

```
NPRESULT TT_LoadTrackables(const char
*filename)
```

For project 2, we may not have to use these trackables, though. They merely define the configurations for the head tracker and the input device.

Appendix

Loading an entire Project

Alternatively, you can load an entire project using this command:

```
NPRESULT TT_LoadProject(const char *filename)
```

This then loads the calibration, trackables, and sets up camera settings.

Appendix

Updating

In order to retrieve tracking information from the tracking system, it has to be updated constantly. I.e. you will have to call the following function on a regular bases, for example, using the idle callback function in GLUT:

```
NPRESULT TT_Update()
```

Appendix

Tracking Points

To track individual points using the tracking system, you can first ask the system for the number of points it identified:

```
int TT_FrameMarkerCount()
```

Based on this number, you can then simply loop through all the points and get their coordinates as described on the next slide.

Appendix

Tracking Points (continued)

To get the coordinates for each identified point, the following functions are provided:

```
float TT_FrameMarkerX(int index)
float TT_FrameMarkerY(int index)
float TT_FrameMarkerZ(int index)
```

This then gives you the 3D coordinate of the point associated with the given `index`.

Appendix

Rigid Body Tracking

The tracking of a rigid body, i.e. the location and orientation of a pre-defined object, works similarly. Using the following function, gives you the number of objects that you can track

```
int TT_TrackableCount ()
```

You can then loop through all tracked objects.

Appendix

Rigid Body Tracking (continued)

Since the tracking system may not have recognized an object, e.g. due to invisibility of one of the markers, you should check first whether the object was found:

```
bool TT_IsTrackableTracked (int index)
```

This then tells you whether the object with the given index was tracking in the current frame.

Appendix

Rigid Body Tracking (continued)

Assuming the object was tracked, you can ask for its name to distinguish between different tracked objects (the tracking system can track up to three rigid body objects at a time):

```
const char* TT_GetTrackableName(int index)
```

In Russ 154A, two objects are defined named `Head` and `Wand` to identify the input device and glasses.

Appendix

Rigid Body Tracking (continued)

To get the orientation and location of a tracked object you can then use this function:

```
void TT_TrackableLocation(int RigidIndex,
          float *x, float *y, float *z,
          float *qx, float *qy,
          float *qz, float *qw,
          float *yaw, float *pitch,
          float *roll)
```

Appendix

Rigid Body Tracking (continued)

This then provides the location (`x`, `y`, and `z`) and the orientation of the tracked rigid body object using both Euler Angles (`yaw`, `pitch`, and `roll`) and quaternions (`qx`, `qy`, `qz`, and `qw`).

Appendix

Shutdown

Once you are done using the tracking system, your program should issue the following command to shut down the tracking system:

```
TT_Shutdown( )
```

This turns off the cameras, which will be indicated by the numbers on the cameras disappearing.

Appendix

Error Handling

As you noticed many functions return a value of type `NPRESULT`. Even though this resembles just an integer type value that is difficult to interpret, the NaturalPoint library provides a function that lets you retrieve a more human readable description, similar to the `glutErrorString` function:

```
const char *TT_GetResultString(NPRESULT result)
```

Printing out the resulting string will then give you a description of the error.