TNM116 — XR Principles and Programming
2. Cluster Rendering, Navigation, Selection, and Manipulation

December 8, 2024

1 Introduction

The purpose of this exercise is to experiment with navigation, selection and manipulation. You will also be working with code already prepared for cross platform rendering, allowing for use on HMD, workbench and CAVE systems.

1.1 Software

In this exercise you will be using Gramods and Eigen. Both are open source projects and both can be automatically connected to your project through `CMake`. You will be building and running these both in the computer lab and in the VR lab. `CMake` can keep these builds separated by allowing individual build folders, e.g. `build_K450x` and `build_G503` respectively.

1.1.1 Build Configuration and Building

With the out-of-source build described above, the Windows build necessary in the VR lab can be made without interference with the computer lab build. Use the `openenv` batch script found at `K:/TNM116/gramods-win` in computer lab room and in the `VRLaboratory` folder in the VR lab. This opens a command window and sets the necessary paths, and the `GRAMODS_PATH` variable used next.

Navigate to your project folder and execute (replacing `build_X` with a suitable room specific folder name)

```
cmake -B build_X ^
-D CMAKE_TOOLCHAIN_FILE="%GRAMODS_PATH:\=/%/vcpkg/scripts/buildsystems/vcpkg.cmake" ^
-D CMAKE_PREFIX_PATH="%GRAMODS_PATH:\=/%/lib/gramods/cmake;cmake_modules"
```

The hat ^ means *continue command on next line* and should be omitted when writing all on one line, however care must be taken to maintain spaces between command and hat. Copy-paste should work. Subsequently you can build directly from command-line using

```
cmake --build build_X --target install --config Release
```

Alternatively you can open the solution from the command-line with `start build_X\solution.sln` and build it. Detailed instructions are provided in a separate document, available on the course web page and in the instructions binder in the VR lab.

1.1.2 Configurations

There are several configuration files specially designed for this exercise available together with the stub. These are set to run in single node configurations as well as to simulate clusters, with and without simulated tracking, and in the VR lab with real tracking. The configuration file to be used on the VR workbench in the VR lab is installed on the workbench computer (`C:/Apps/gramods.bin-*/share/gramods/config`). This way we can make sure that you always have an up-to-date configuration to work with. The `start.bat` script will automatically use this.

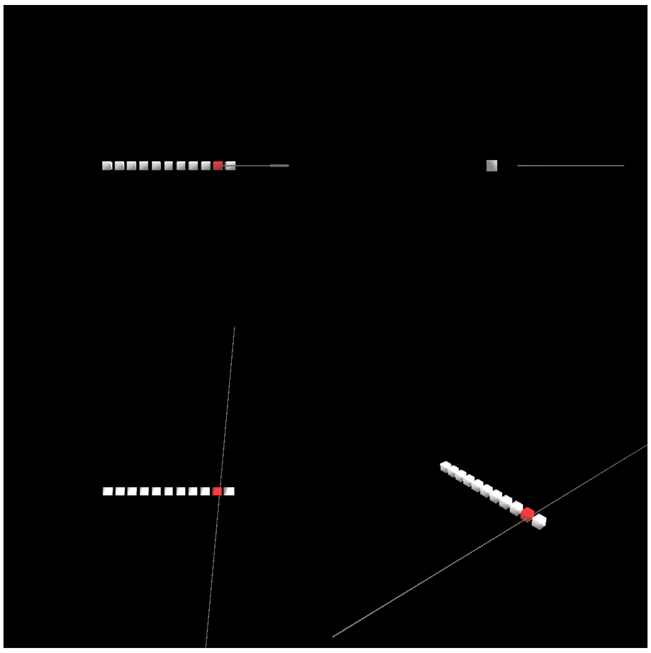

Figure 1: The provided configuration files for testing without VR equipment open a window with four views (front, side, top and perspective). Some also create a `Controller` object with wand pose data either read from VRPN or playing back preset values over time.

A time-based tracking configuration provides a basic interaction pattern simulated through preset samples. You are welcome to adjust this to fit your needs. Also, keyboard input can be used both in place of real wand input during initial development, and as a replacement of a more advanced menu for selecting between modes of interaction. Use `addEventHandler`[1] in `Window` to add a callback receiving key events from a window.

In cluster mode, you will need to start two instances of the program to make it run. The configuration files set the local peer idx to an invalid value, so that you get an error if you forget to set this to the actual node idx of the instance as you start it. Use these or equivalent command lines:

```
.\bin\main ^
     --config config/desktop-2-node-VRPN-tracking.xml ^
     --param SyncNode.localPeerIdx=0
.\bin\main ^
     --config config/desktop-2-node-VRPN-tracking.xml ^
     --param SyncNode.localPeerIdx=1 &
```

There are start and stop scripts available, `start_K450x.bat` and `start_G503.bat` for computer lab and the VR laboratory respectively, that can be modified and used for your purposes. Please read these to understand what happens.

1.1.3 Tracking Simulation

It is possible to start a VRPN tracking server that reads off mouse movements and use that to simulate tracker movements. The data are sent through VRPN as sensor 0 on tracker H3D@localhost. Only a single button is supported (triggered by the right mouse button) but more can be simulated by creating your own combination of the Gramods configuration files for VRPN tracked input and time-based simulation of input.

Open an H3D command window from the Windows menu and start the VRPN server with the following command:

```
H3DLoad urn:candy:x3d/VRPNServer.x3d
```

[1]E.g. in your `main.cpp` use
`window->addEventHandler([myapp](const gmGraphics::Window::event *e) -> bool { myapp->myfunc(dynamic_cast<gmGraphics::Window::key_event*>(e)->key); return true; }, this);`

Depth movements of the stylus are performed by dragging with the middle button, up for away and down for closer.

1.2 Documentation

Eigen is a powerful tool. Thus, its documentation is comprehensive and might feel overwhelming at first glance.

- Gramods Reference Manual
  `https://www.itn.liu.se/~karlu20/work/Gramods-docs/index.html`
- Eigen3 Reference Manual
  `https://eigen.tuxfamily.org/dox/`
- Eigen3 Getting Started Guide
  `https://eigen.tuxfamily.org/dox/GettingStarted.html`
- Eigen3 Long Tutorial
  `https://eigen.tuxfamily.org/dox/group__TutorialMatrixClass.html`
- Eigen3 Quick Reference
  `https://eigen.tuxfamily.org/dox/group__QuickRefPage.html`

2 Cluster Application Structure

Typical HMD applications are run as a single process. For many RMD systems, however, the application is distributed on the nodes of a cluster. With the right program design we can make our software run on both HMD and RMD, both on a single computer and on a cluster. Depending on display system, rendering may be done once (mono), twice (stereo) or more times (e.g. VR theater and Dome).

Task 1 — Code Structure:
Read the code of the stub and try to understand the general code structure, i.e. what methods there are and how they are called from different parts of the program. A suggestion is that you start reading in the main function and trace the execution path of the thread. Observe also that the pimpl[a] class *is-an* `Updateable`. Check the documentation for details.
You should be able to answer the following questions: how is the program initialized and where is the code for its main execution; what is the current scene graph structure and how do you change that; how are the graphics changed over time, e.g. for animation; where in the code should you change scene graph states?

[a]pimpl (pointer-to-implementation) is a way to move the declaration of *internal* (i.e. private) methods to the .cpp file.

Task 2 — Configuration De-coupling:
Identify code that connects to or responds to run-time configuration, e.g. the graphics output, data synchronization and input, such as head tracking and wand data. Since the configuration can be controlled from command-line, a suggestion is that you trace data uses starting with `argc` and `argv`.
You should be able to answer the following questions: which lines of code determine how the scene is rendered and is there any function call that triggers the rendering; what happens if there is no head or wand tracking available; what controls how one instance communicates with its peers and which is the primary; is there code that is only run on the primary and only on the replicas, respectively?

Your application may run simultaneously on multiple computers each rendering a part of the full 3D experience for the user. Thus, you will have to understand how the internal states are synchronized between nodes and make sure that any further data that you need will be synchronized as well.

Task 3 — Independent and Dependent States:
Identify the independent and dependent states, respectively, and see how they are read, writte or otherwise controlled in the code.
You should be able to answer the following questions: which lines of code read off wand position 1) in the primary node and 2) in replicas, and what decides what data the primary will read off; which states are currently shared between the nodes and at which places do you need to add code to share another variable; what happens if there is only one instance of the software and how does it differ from running multiple; are there any new data calculated from the synchronized data and/or where is/should that code be located?

3 Basic VR Programming

As a first step we will create a virtual environment with some objects and start navigating in this world. The library we work with has only rudimentary support for reading models, supporting only the Wavefront OBJ file format through the `ObjRenderer` class. Positioning, scaling and rotating the model can be done e.g. with `PoseTransform`, however to be able to perform *grabbing* we will use `MatrixTransform` as manipulation transform. Consider creating a scene graph structure that supports both navigation and manipulation already at this stage.

Task 4 — Load Objects:
Create a method for loading an .obj and adding it to your scene graph, and call this method either from a loop or multiple times to load at least five objects, distributed over the 3D space. Make sure that your object's positions and sizes are approriate for the tasks below; we will soon be able to navigate but the workbench is only a bit more than a meter wide which will limit the size of objects you can see and interact with at close range.

Now that you have a set of objects in your environment, you will need the ability to navigate. There are many ways to navigate but we will start with a simple continuous flight approach.

Task 5 — Wand-based Navigation:
Make sure that you have a scene graph structure that supports navigation and implement cross-hair mode navigation, i.e. the mode where the vector from the user's eye/head to their wand defines the flight direction. Use the "main button" to activate navigation and use constant speed at this point.

A simple way to find objects to select is to use an intersection visitor to find the objects that are intersected with the wand ray. A visitor is applied to a node and will automatically traverse the scene graph and collect data. Observe that a visitor will itself apply transforms it encounter, so the intersection data need only conform with the space of the first node it is applied to.

To simplify interaction programming, please consider *states*. Highlight an object only when *transitioning* between states. For example, a pointer to the "selected object" can be used as "state", so that you change the highlighting when the (closest) intersected object is *not* the same as the selected object. A simple way to highlight an object in Gramods is to read off the object's materials, set the first material's color to something different and then set them back to the object. Do not forget to save away the original material data so that you can un-highlight the object when needed.

Task 6 — Wand-based Selection:
Implement wand-based selection so that when the user points at an object, that specific object is highlighted. Make sure that only the closest object is selected even if more are intersected and check that selection also works after navigating.

4 More Navigation

At this point we should be able to navigate the scene and select different objects from the position we might be at. Let us adjust the navigation to give us more control over our position.

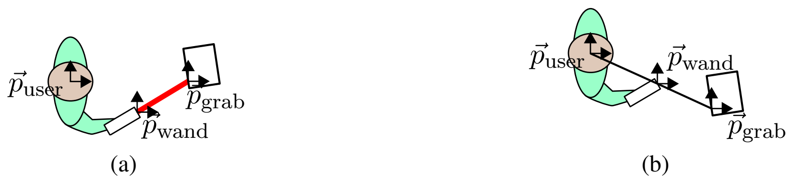

Figure 2: Original HOMER, illustrating a) selection and then b) grabbing, resulting in a jump in position. There are ways to remedy this, however these are out of the scope of this exercise.

Task 7 — Wand-controlled Speed:
Adjust the previously implemented navigation so that the user can control the speed of navigation. When you initiate navigation the speed is still zero, but when the user moves the hand further away from their head speed is larger the further they move the hand. Also, moving the hand closer should make navigation go backwards.

Fly navigation is intuitive, however over large distances it will either take a long time or use a speed that might lead to cyber sickness. A popular alternative is teleportation, which simply means that we instantaneously make a large change to the navigation transform. We will also add an orientational change, just to experiment a bit with user coordination; this is *not* recommended in products. The selection of teleportation target should typically also provide better distance control.

Task 8 — Teleportation:
Implement a second mode of navigation where the user is teleported to a position on the wand ray, a fixed distance forwards. Start with simple teleportation but once this is working to satisfaction, then also rotate the scene 180° around the Y axis upon teleportation.

Hint: Make sure that the center of rotation is at the user's new position.

Hint: The wand size is encoded in its Obj structure (1000 m long), however you can control its drawn length by applying scaling in the z direction.

5 Means for Pose Manipulation

The final thing to look at in this exercise is pose manipulation. Here we will work with the manipulation transform so make sure that you have a scene graph that supports navigation and manipulation.

Task 9 — Grab Manipulation:
Implement grabbing so that if an object is selected, the user can hold down a button to move and rotate that object. When correctly implemented, the object should follow the movements of the wand as if they were firmly welded together via the wand ray. You will have to make use of some variable to represent the state of "grabbing", to make calculations upon the transition from "not grabbing" to "grabbing"; use this state to also stop testing for intersection while the user is manipulating the object's pose, until the button is released. Do *not* use the same button as for navigation; you want to be able to navigate and grab at the same time.

You will notice that grabbing is an inuitive means of pose manipulation, but also that it is not ideal for rotating an object at a distance. Consider the use-case when the user wants to place objects at different positions in a room, e.g. vases on shelves. For this the HOMER mode of pose manipulation is much more suitable.

Task 10 — Hand-centered Object Manipulation Extending Ray-casting:
Implement a simple version of the HOMER mode of pose manipulation, allowing the user to move the object closer, farther and sideways, and adjust its orientation. You may use another button to switch between which pose manipulation mode to use.

6 Final Words

We hope that you have, after performing these tasks, achieved an understanding of programming different types of navigation, selection and manipulation in XR. Different game engines and graphics APIs will provide different structure and tools, but the basic principles stay the same. These are also only examples, but by implementing a few different modes and seeing the how the mathematics translates into interaction design we believe that you are at a great position for finding the most inuitive and efficient mode for any task at hand.