TNM086 – VR-technology
2. Cluster Rendering, Navigation and Interaction

December 1, 2023

1 Introduction

The topic of this lab exercise is rendering, navigation and manipulation together with cluster synchronization. The aim is for you to gain insight in how tracking and display hardware interact with VR software to implement an efficient interface to the virtual environment.

1.1 Examination

The tasks of this exercise must be presented by showing the running program and the code to a supervisor. You must also be able to answer questions regarding the mathematical and Human/Computer interaction principles as well as the tasks and the implementation of these. Complete all tasks before engaging a supervisor for examination.

When done with the tasks, show your implementation to a supervisor during a supervised session to ensure that you've implemented them according to the instructions. The exercise will be examined in the VR laboratory, using the equipment there, so when you have finished all tasks, engage a supervisor to request examination. They will then schedule slots for examination in the VR laboratory.

1.2 Equipment

The most important part of this exercise will be performed on the VR workbench in the VR laboratory. Much of the programming, however, can also be done on any computer with the required software installed. Navigation and interaction experiments must be conducted in the VR lab.

1.3 Software

In this exercise you will be using Gramods to configure a VR display and handle state synchronization. Gramods is a library that includes functionality for cluster data and execution synchronization, for advanced graphics pipelines and for tracking. Your final application, if implemented correctly, will run also in the Dome, in the VR arena and on any other similar supported platform.

The implementation of 3D graphics will be done using OpenSceneGraph (OSG), so the completion of the associated exercises is encouraged before starting this one. You are also encouraged to prepare all tasks on other computers before booking the VR equipment.

There is a simple OSG demo in Gramods, demonstrating a basic way of integrating with OSG, that will serve as a stub. It shows how Gramods is used together with OSG, using wand tracking and wand buttons. Keyboard input can be used in place of the wand during initial development. Use `gmGraphics::Window::addEventHandler` to add a callback receiving events from a window.

Gramods uses Eigen3 for linear algebra while OSG uses its own API with other conventions, so a basic understanding of both and the difference between their respective conventions is of importance.

1.3.1 On Linux over ThinLinc

Both Gramods and OSG are for your convenience installed on the Linux system available over ThinLinc (thinlinc.edu.liu.se). The CMake build system used for the stub will automatically find all dependencies except Gramods. First run the following two command lines in your terminal:

```
source /courses/TNM086/gramods-lnx/setenv.bash
module load courses/TNM086
```

This will set some environment variables for Gramods as well as H3D, for simulating a tracker over VRPN. The environment variables will also be inherited to software opened from this terminal, such as IDEs, but they are not retained between sessions or instances of terminals. Thus, you have to execute these commands again next time you open a terminal.

For CMake to find the correct paths and libraries, set up the build files using the following command line:

```
cmake -B build_lnx -D CMAKE_PREFIX_PATH:PATH="/courses/TNM086/gramods-lnx;cmake_modules"
```

When you build your project (e.g. with `make -C build_lnx`), build files, object files and the final executable will end up in `build_lnx`, out of the source tree. This way you can later create a second, independent build folder for the VR laboratory, that sit side-by-side with the Linux build files.

1.3.2 On Windows in the VR Lab

With the out-of-source build described above, the Windows build necessary in the VR lab can be made without interference with the Linux build. Use the `openenv` batch script found in the `VRLaboratory` folder. This opens a command window and set the necessary paths, and the `GRAMODS_PATH` variable used next. Also, make sure that your `start.bat` script makes use of the corresponding `setenv` script.

Then navigate to your project folder and execute

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

The hat ˆ 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_win --config Release`. Alternatively you can open the solution from the command-line with `start build_win\solution.sln` and build it, however make sure that you select `Release` build configuration. Detailed instructions are provided in a separate document, on the course web page and in the instructions binder in the VR lab.

1.3.3 Configurations

There are several configuration files specially designed for this exercise available together with a stub. These are set to run in single node configurations as well as to simulate clusters, for experiments over ThinLinc, 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:/VRSystem/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.

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 should 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 on Linux:

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

The ampersand (&) makes the command split off in the background of the terminal instead of staying in the foreground waiting for keyboard input. The backslash makes the command line continue on the following line.

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

1.3.4 Tracking Simulation

It is possible to start a VRPN tracking server on Linux over ThinLinc, that reads off mouse movements and use that to simulate tracker movements. These 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.

Start the VRPN server with the following command:

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

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

1.4 Documentation

In this exercise you will be using OpenSceneGraph, so the documentation from the exercises you have earlier done on those APIs may be very useful. Here are listed some sources of additional documentation.

- Gramods Reference Manual
  `http://www.itn.liu.se/~karlu20/work/Gramods-docs`
- OpenSceneGraph Quick Start Guide (free in PDF format — in the models folder)
  `https://www.itn.liu.se/~karlu20/courses/TNM086-2022/labs/openscenegraph_quick_start_guide.pdf`
- OpenSceneGraph Reference Manual
  `https://www.itn.liu.se/~karlu20/tmp/OpenSceneGraph`
- Eigen3 Manual
  `http://eigen.tuxfamily.org/dox/`

2 Basic VR Programming

In this part of the exercise you will use a scene graph API to implement basic graphics and interaction. The exercise has been prepared for OSG, so this is the recommended scene graph API, however each task may also be replaced by an *equivalent* task with the API of choice. It is recommended that you record your progess and document how to select between the different functionalities you have implemented. Also, pieces of your code can be de-/activated using `#if 0` or `#if 1` and `#endif`.

Task 1 — Understand Gramods + OSG:
Read the code of the stub and try to understand what the code does; how the software updates data, synchronizes execution, synchronizes data and renders graphics. A suggestion is that you start reading in the main function and trace the execution path of the thread. You should be able to answer the following questions: which lines reads 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; which lines of code determine which OSG scene is rendered and what function call triggers the rendering; what is the current scene graph structure and how do you change that; where in the code should you change scene graph states?

Now it is time to add your own objects to build your own scene. For this you may use code from the OpenSceneGraph exercise, however keep in mind that the VR view does not automatically adjust to the objects in the scene and that large objects in realistic metric units might end up outside of the view. There is code in the stub that automatically shrink the loaded model to a suitable size and readjusts its origin. This can be handy to make sure that your loaded objects fit the view. A typical configuration will place origin in the middle of the display system.

Task 2 — Create an Application:
Add a more interesting scene to the stub. The scene should include at least two objects, that at a later stage will be navigated around and moved individually.

When run in the VR laboratory, the display should be rendered in stereo and head tracking should be used. Under Linux, the software should be tested to correctly run with any of the provided configuration files; some mistakes only show in certain configurations.

Task 3 — Understand the Scene Graph and Data:
Draw the scene graph of your current program. Which transforms do you have and what are their respective purposes? Which are the most important data used to control the rendering, where in the code are they updated, on which node (primary or replicas), and how are those data used in the scene graph?

Explicitly calculating the selection of objects is most simply done by calculating the intersection or closeness between a line pointing from your wand and spherical objects, such as planets. With a scene graph API there are usually methods for automatically checking for intersection with parts of the scene graph, with bounding spaces and geometries. Previously, you have used an OSG callback to be able to update scenegraph states at run-time. Observe that this no longer is necessary, since we now have an update callback where all processing can be performed.

Task 4 — Interaction with Objects:
Implement selecting objects (at least two different) at a distance in your virtual environment. Indicate the selection in some way, for example by changing the scale of the model transform so that the model becomes slightly larger or changing the colour of the object. With OpenSceneGraph you use the intersection detection feature, and the `nodePath` member of each intersection instance is useful for identifying which model that has been intersected.
[Note: Observe that independent states (e.g. tracker data) should *only* be updated *pre-sync* on the primary node, synchronized with the replicas, and then used to update dependent states (e.g. graphic states to indicate selection) in *post-sync* on *both* the primary and replica nodes. Observe also that changes made on the scene graph during rendering will result in state changes between drawing for the left and the right eye, or for different views in a larger display.]

3 Advanced VR Programming

In this part of the exercise you will implement features where the scene graph API can be of much more assistance. Explicit navigation and manipulation can generate quite complex software if care is not taken.

Both navigation in the scene and manipulation of the pose of objects are a matter of manipulating transforms in the scene. Review your lecture notes when necessary.

Task 5 — Explicit Navigation:
Use the wand information to control the transform of your scene so that you can navigate naturally in your virtual environment. Implement *pointing mode* and *cross hair mode*. Start by navigating with constant speed.

Both the *pointing mode* and the *cross hair mode* define a direction vector in which to fly, in different ways, and add this vector to the translation in the navigation transform. Depending on whether navigation is considered moving the user themselves or the objects, either the vector itself or its inverse (minus the vector) is applied.

One of the most straightforward ways of controlling the speed of navigation is to use the head-hand distance. The distance read off when the user *engages* the navigation mode should be saved away and subsequently be used as a distance representing zero speed. As the user moves their hand closer or further away from the head this new distance can then be used to provide negative or positive speed, respectively.

Task 6 — Control Flying Speed:
Implement hand controlled acceleration or hand controlled speed for the navigation modes from the previous task.

When both navigation and pose manipulation is allowed, there will be an interaction between at least two different transforms: the navigation transform and the individual transform of the object for which the pose is manipulated. In the following task you shall implement *grabbing* of objects in the scene, by controlling transforms when the user press a selected button on the wand. The mathematics necessary to make the model follow the wand motions is available in the slides for the course lectures. When implemented correctly, it will even be possible to grab an object and bring it with you during navigation.

Task 7 — Moving Objects:
Implement *grabbing* so that you can directly and intuitively manipulate the individual poses of at least two different objects in your scene. The manipulation should be based on wand movements and include both translation and rotation of the object. How do you move an object at distance? What is the centre of rotation?