INVRS

interactive networked virtual reality system

Tutorial I
-
Medieval Town

Christoph Anthes **Roland Landertshamer** **Marina Lenger**

JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Overview

interactive networked virtual reality system

Introduction and Architecture Overview

Introduction

- ***inVRs*** (**i**nteractive **n**etworked **V**irtual **R**eality **s**ystem)
  - Pronounced ɪnˈvɜːs
  - C++ Application Framework for Networked Virtual Environments (NVEs)

- Publicly available under www.invrs.org (under LGPL)
  - Uses Trac system for distribution
    - Alternatively latest stable release as zip
  - Available for Linux, Windows, Mac OS X and maybe IRIX

- Variety of out-of-the-box Features
  - Network communication supporting close coupled collaboration
  - Clear definition for navigation and interaction (with many techniques provided)
  - Designed for reusability of developed components
  - Easy-to-use configuration for immersive devices

Introduction

- Documentation available on Trac System
  - Several publications (BiBTex files)
  - Tutorial examples
  - Medieval Town Tutorial (on which these slides are based on)
  - Programmers' Guide (coming soon)
  - Doxygen (http://doxygen.invrs.org/)
- Libraries
  - Required
    - OpenSG v1.8 (http://www.opensg.org/)
    - GMTL distributed with source (http://ggt.sourceforge.net)
    - IrrXML distributed with source (http://www.ambiera.com/irrxml/)
    - CMake (http://www.cmake.org/)
  - Optional
    - OpenAL and ALUT (http://www.openal.org/)
    - ODE (http://www.ode.org/)
    - trackD (comes with CAVELib)

Introduction

- Where to position *inVRs* in the development approaches for VR applications
  - Graphical Editors (using software components)
    - (e.g. EON, VirTools, Quest3D)
  - VR systems incorporating scripting languages
    - (e.g. DIVE, ALICE, AVOCADO)
  - Application Frameworks
    - (e.g. VRJuggler)
  - Scene Graphs
    - (e.g. OpenSG, OpenSceneGraph)
  - Full development from scratch via C or C++
    - OpenGL presents comparably high effort for new development of VR applications

JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Introduction

interactive networked virtual reality system

- Application Areas

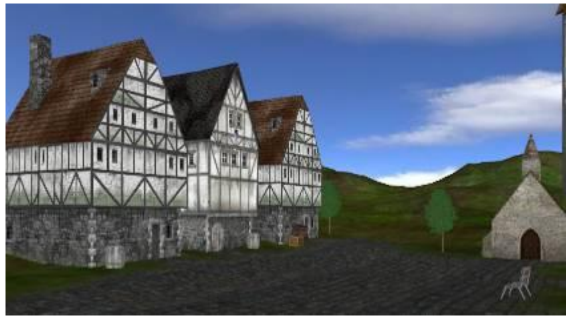
Safety Training

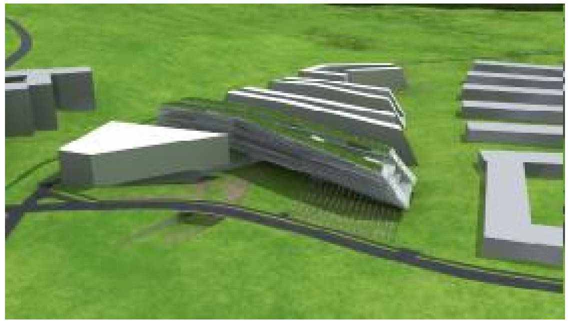
Collaborative Work

Entertainment

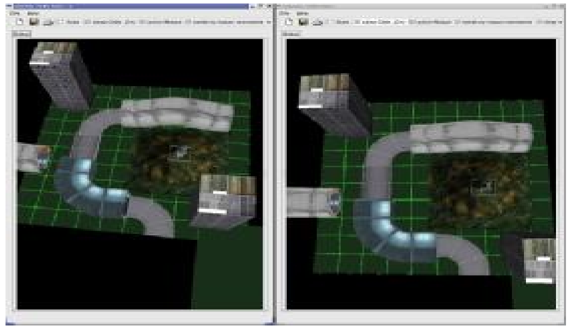
Art

Scientific Visualization


Education

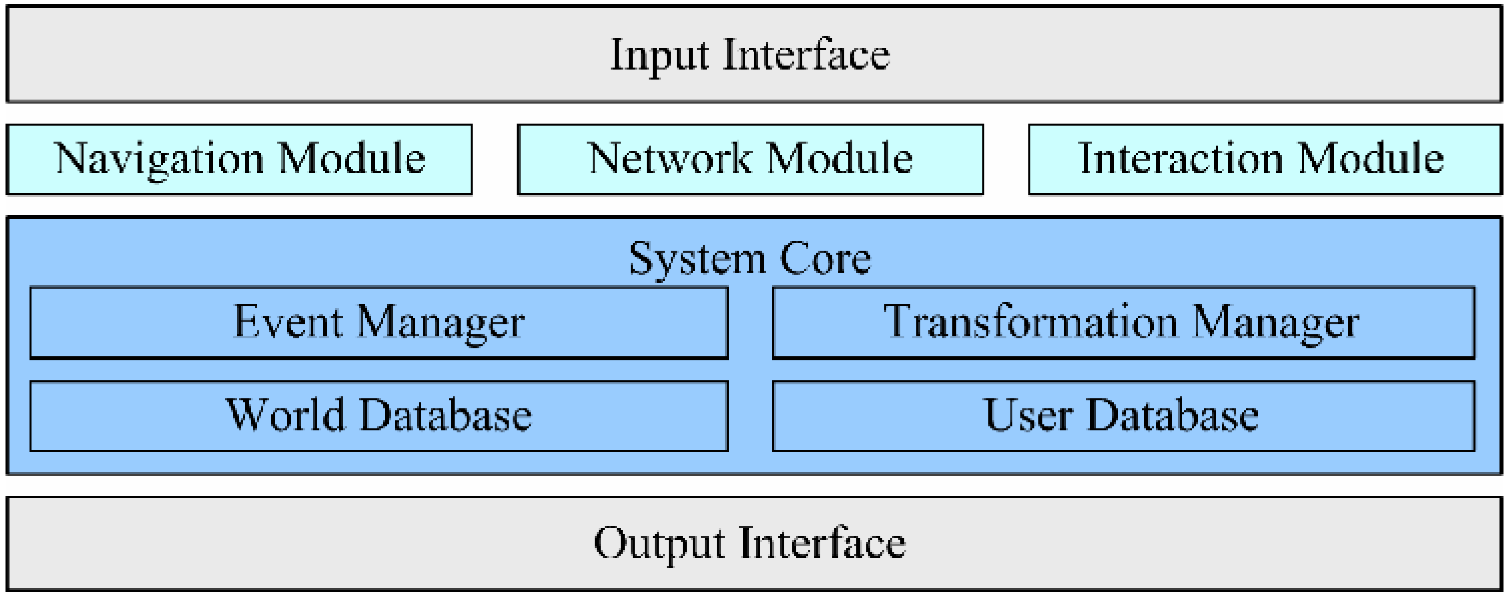

Architecture Visualization

Product Presentation


World Layout

Architecture Overview

- Overview – 3 different types of *inVRs* components
  - System Core (blue)
  - Modules (cyan)
  - Interfaces (grey)



  - Communication typically happening top down in the diagram
  - Strong dependencies between Interfaces and System Core
  - Weak dependencies between Modules and other components

JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Architecture Overview

interactive networked virtual reality system

- System Core
  - System Core contains databases and communication mechanisms
  - Offers interface between external modules and databases
  - Core functionality (data types, configuration handling, logging, timer, etc.)
- Modules
  - Implemented as plugins
  - Follow a structured approach (init, loadConfig, cleanup, etc.)
  - Standard package contains modules for interaction, navigation, and network communication
  - Physics simulation and animation are available as additional modules
- Interfaces
  - Abstraction of input devices
  - Abstraction of graphical and audio output
  - On the graphics side only OpenSG is supported so far, plans to support OpenSceneGraph exist as well

Tutorial Structure

- The tutorial is based on many copy and paste operations in order to safe time
- Four basic file classes are needed for this tutorial
  - MedievalTown.cpp
    - Contains the application parts from where the development can start
  - Configuration Files
    - Stored in the config subdirectorys
    - Have to be altered through snippets (only in offline version with manual)
  - CodeFile - Snippets
    - Contain code snippets which should be copied into the MedievalTown.cpp source file
    - Can be found under the subdirectory snipptes
  - ConfigFile - Snippets
    - Contain XML configuration which has to be copied in the actual configuration files (only in offline version with manual)
    - Can be found under the subdirectory snipptes
- Open the file **`MedievalTown.cpp`** with the editor of your choice

Chapter 1
Basic Application Development

Where a simple OpenSG application makes use of the WorldDatabase in order to load the scene graph of a Medieval Town.

Basic Application Development

- Predefined Functions
  - **`void cleanup()`**
    - System cleanup for OpenSG and later for inVRs
  - **`void display(void)`**
    - Main display loop which is invoked by a GLUT callback.
  - **`void reshape(int w, int h)`**
    - The method is used for reaction on changing of the window size.
  - **`void mouse(int button, int state, int x, int y)`**
    - This method reacts to button presses of the mouse.
  - **`void motion(int x, int y)`**
    - Forwards the coordinates of the mouse during mouse motion.
  - **`void keyboard(unsigned char k, int x, int y)`**
    - This method reacts to keyboard input.
  - **`void keyboardUp(unsigned char k, int x, int y)`**
    - Reacts on keyboard input. It is invoked when a key is released.
  - **`int setupGLUT(int *argc, char *argv[])`**
    - Sets up the GLUT system and registers required callback functions.

JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Basic Application Development

INVRS interactive networked virtual reality system

- Writing a simple *inVRs* application using OpenSG and GLUT
  - OpenSG and a GLUT Window have to be initialized first

```
int main(int argc, char **argv) {
  osgInit(argc, argv);                    // initialize OpenSG
  int winid = setupGLUT(&argc, argv);     // initialize GLUT

  // the connection between GLUT and OpenSG is established
  GLUTWindowPtr gwin = GLUTWindow::create();
  gwin->setId(winid);
  gwin->init();
```

MedievalTown.cpp

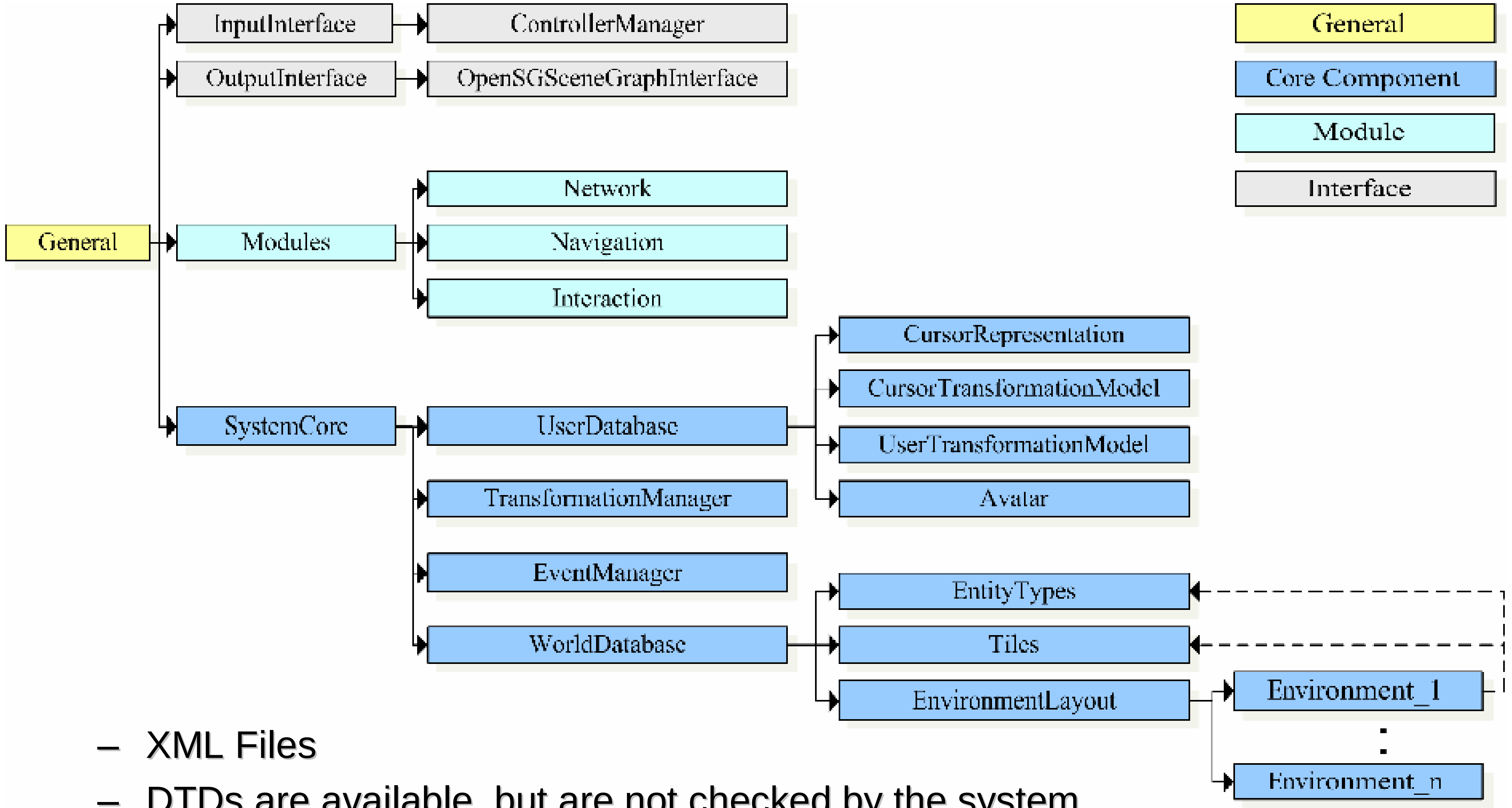
  - Create a simple OpenSG Node and set a Group Core. Attach it to the scene.

```
  NodePtr root = Node::create();
  beginEditCP(root);
    root->setCore(Group::create());
  endEditCP(root);

  mgr = new SimpleSceneManager;  // create the SimpleSceneManager
  mgr->setWindow(gwin);          // tell the manager what to manage
  mgr->setRoot(root);            // attach the scenegraph to the  root node
  mgr->showAll();                // show the whole scene
  mgr->getCamera()->setNear(0.1);

  glutMainLoop(); // GLUT main loop
  return 0;
}
```

MedievalTown.cpp

Basic Application Development

- Configuring *inVRs*



  - XML Files
  - DTDs are available, but are not checked by the system

Basic Application Development

- Configuring *inVRs*
  - **`general.xml`** sets up *inVRs* components (in <general>) and all system paths (in <paths>), you have to update your own plugin path here.
  - Paths refer to other configurations, models and tools

```
<?xml version="1.0"?>
<!DOCTYPE generalConfig SYSTEM "http://dtd.inVRs.org/generalConfig_v1.0a4.dtd">
<generalConfig version="1.0a4">
  <!-- This is the configuration for the inVRs Framework -->
  <general>

<!-- **************************** Snippet-2-1 **************************** -->

    <Interfaces>
      <option key="outputInterfaceConfiguration" value="outputInterface.xml"/>
    </Interfaces>
    <SystemCore>
      <option key="systemCoreConfiguration" value="systemCore.xml"/>
    </SystemCore>
  </general>
  <paths>
    <root directory="./"/>
    <path name="Plugins"
        directory="/please/insert/your/inVRs/libs/path/here/"/>
    <path name="SystemCoreConfiguration" directory="config/systemcore/"/>
    <path name="OutputInterfaceConfiguration"
```

general.xml

Basic Application Development

- Loading *inVRs* Configuration Data
  - Load the general configuration of the *inVRs* framework
  - Call the loadConfig method with the configuration setup at which we had a look at on the previous slide

```
// very first step: load the configuration of the file structures, basically
// paths are set. The Configuration always has to be loaded first since each
// module uses the paths set in the configuration-file
if (!Configuration::loadConfig("config/general.xml")) {
  printf("Error: could not load config-file!\n");
  return -1;
}
```

Snippet 1-1

  - After having set the paths of the needed components the core configuration is to be triggered
  - Triggering this configuration in general results in subsequent internal configure calls, for example for the individual core components, the components of the interfaces or the registered modules

Basic Application Development

- Triggering the Configuration and Adding a Scene
  - This call causes the *inVRs* system to fill the WorldDatabase with its data
  - The medieval town is loaded at this stage
  - The interface to the OpenSG scene graph is established in this step as well

```
std::string systemCoreConfigFile = Configuration::getString(
    "SystemCore.systemCoreConfiguration");
std::string outputInterfaceConfigFile = Configuration::getString(
    "Interfaces.outputInterfaceConfiguration");

// !!!!!! Remove in tutorial part 2, Snippet-2-1 - BEGIN
if (!SystemCore::configure(systemCoreConfigFile, outputInterfaceConfigFile)) {
  printf("Error: failed to setup SystemCore!\n");
  return -1;
}
// !!!!!! Remove - END
```

Snippet 1-2

- Interconnecting OpenSG and *inVRs*
  - Request the SceneGraphInterface from the OutputInterface
  - Retrieve the root node of the scene graph interface
  - Attach it to the to the root node of our so far empty OpenSG scene

Basic Application Development

```
OpenSGSceneGraphInterface* sgIF =
  dynamic_cast<OpenSGSceneGraphInterface*>(OutputInterface::
      getSceneGraphInterface());
if (!sgIF) {
  printf("Error: Failed to get OpenSGSceneGraphInterface!\n");
  printf("Please check if the OutputInterface configuration is correct!\n");
  return -1;
}
// retrieve root node of the SceneGraphInterface (method is OpenSG specific)
NodePtr scene = sgIF->getNodePtr();

root->addChild(scene);
```

Snippet 1-3

- Cleaning Up
  - Since we have instantiated the *inVRs* SystemCore, including its sub-components we have to clean up at exit
  - Insert the following snippet in the cleanup method

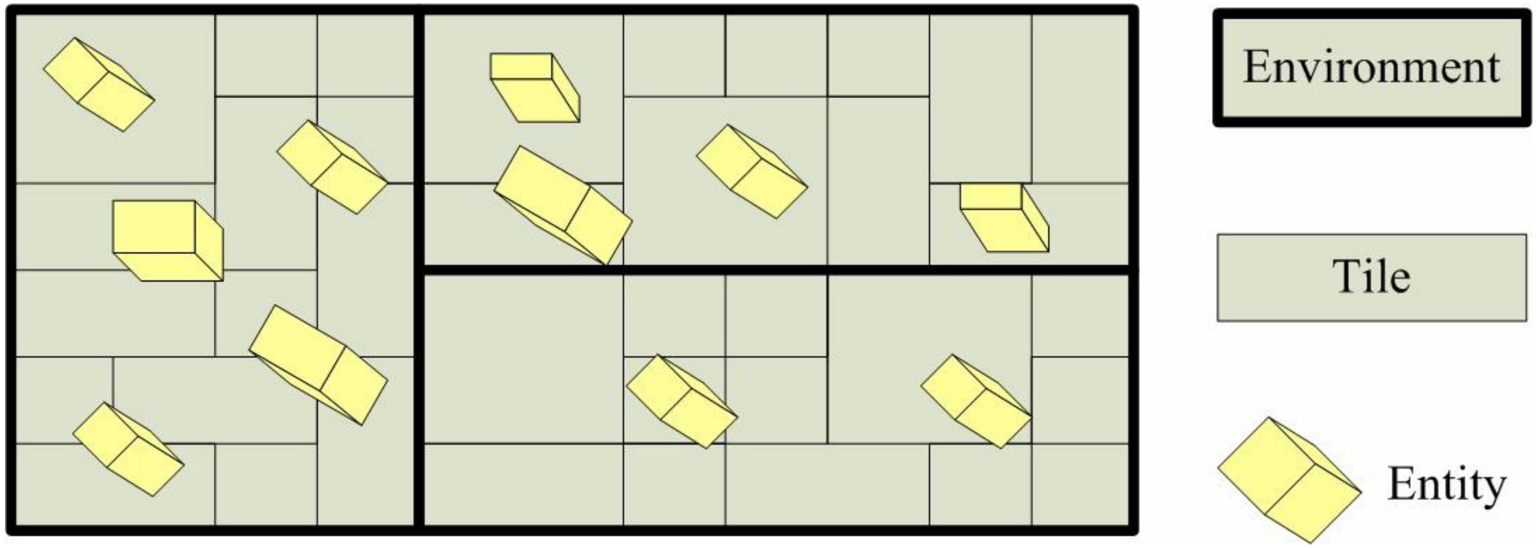
```
SystemCore::cleanup(); // clean up SystemCore and registered components
```

Snippet 1-4

- Recompile and Run
  - You should now see a medieval town behind a black background
  - A basic navigation is provided by the OpenSG SimpleSceneManager

Basic Application Development

- Working with the World Database
  - An *inVRs* VE consists of Environments, Entities and Tiles



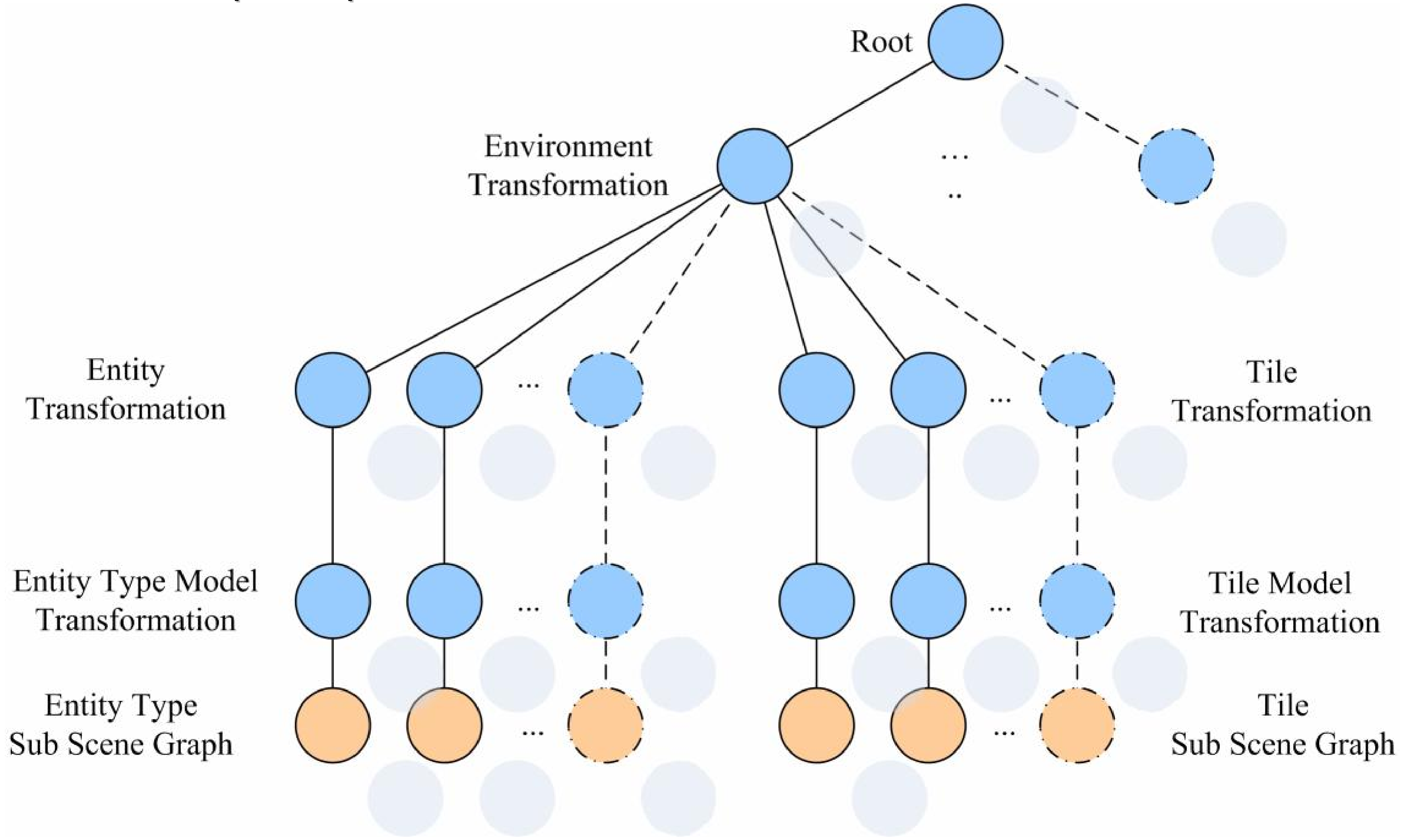
  - They are typically used for partitioning and functionality (Entities) in the VE
  - Configurations for these types are stored in **`worldDatabase.xml`**

```
<?xml version="1.0"?>
<!DOCTYPE worldDatabase SYSTEM "http://dtd.inVRs.org/worldDatabase_v1.0a4.dtd">
<worldDatabase version="1.0a4">
  <entityTypes configFile="entities.xml"/>
  <tiles configFile="tiles.xml"/>
  <environmentLayout configFile="environmentLayout.xml"/>
</worldDatabase>
```

worldDatabase.xml

Basic Application Development

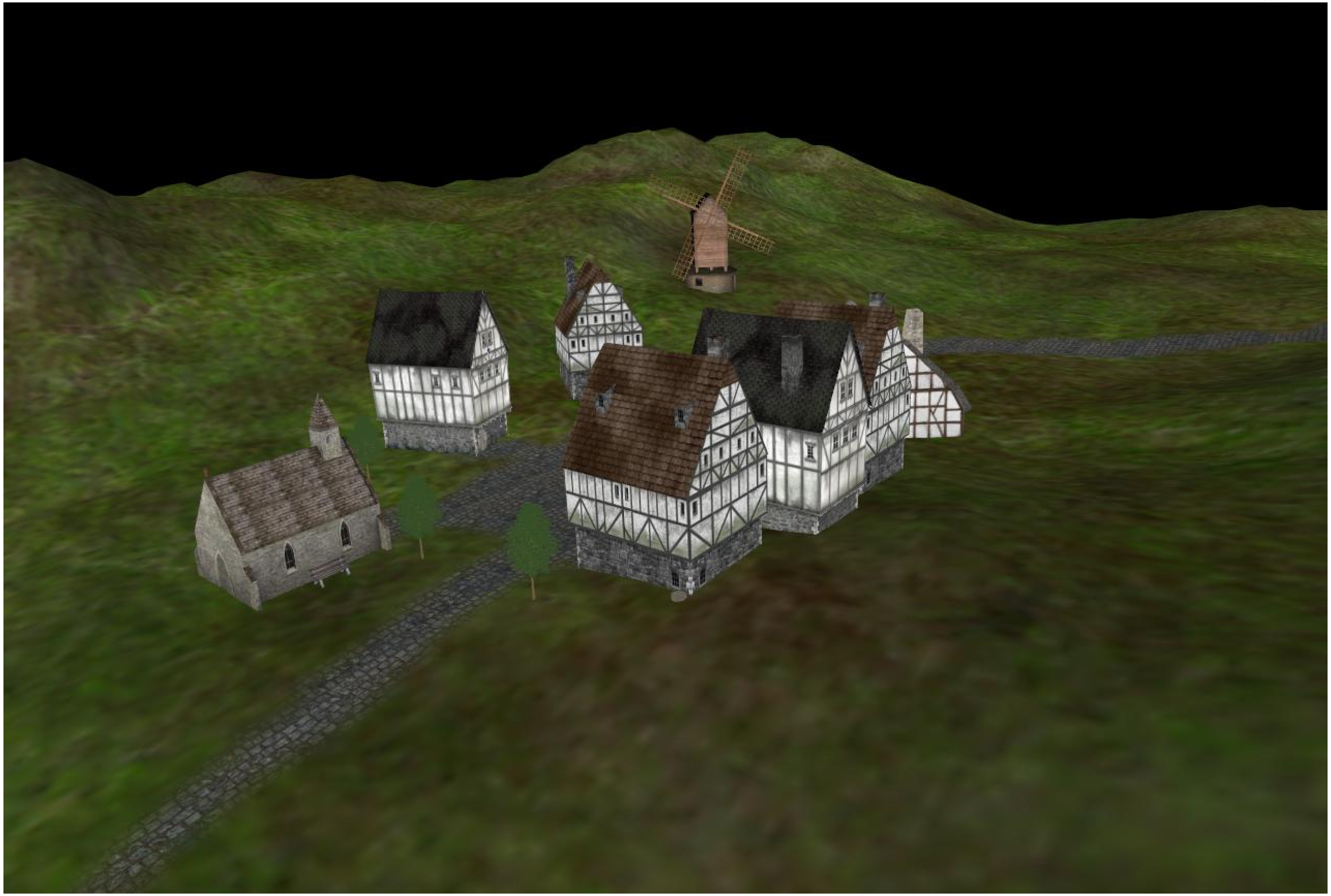
- Scene Graph Representation


Root
Environment Transformation
…
..
Entity Transformation
Tile Transformation
Entity Type Model Transformation
Tile Model Transformation
Entity Type Sub Scene Graph
Tile Sub Scene Graph
...

JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Basic Application Development

INVRS
interactive networked virtual reality system



Chapter 2
Navigation and Skybox

Where new *inVRs* components will be added, the user is able to travel through the scene and the environment is lightened up.

Navigation and Skybox

- Adding *inVRs* Components
  - In order to use the navigation several components are needed
    - The Navigation module – to provide new user or camera coordinates
    - The InputInterface – to gather input from devices and expose it to the Navigation module
    - The UserDatabase – to set camera and user transformations
    - The TransformationManager – to pass the transformations generated by the Navigation module to the UserDatabase
  - This requires
    - Setting up the configurations of these components if they are not already provided
    - Registering these components unless they are core components
    - Implementing callbacks to access these components during the initialization

Navigation and Skybox

- Adding *inVRs* Components
  - The configuration-files for the two component types, the InputInterface and the modules, have to be set as well at the top of general.xml

```
<Modules>
  <option key="modulesConfiguration" value="modules.xml" />
</Modules>
<Interfaces>
  <option key="inputInterfaceConfiguration" value="inputInterface.xml" />
</Interfaces>
```

general.xml

  - Interfaces and modules are available as individual libraries
  - They can be loaded as plugins
  - The paths provided in this snippet define the location where the configuration files for these component types are located

```
<path name="InputInterfaceConfiguration" directory="config/inputinterface/" />
<path name="ModulesConfiguration" directory="config/modules/" />
```

general.xml

Navigation and Skybox

- Adding *inVRs* Components
  - The paths for the configurations of the individual components, not to be mixed up with the component types, have to be provided in general.xml

```
<!-- Path for Interfaces Datastructure -->
<path name="ControllerManagerConfiguration"
  directory="config/inputinterface/controllermanager/" />

<!-- Paths for Module Datastructure -->
<path name="NavigationModuleConfiguration"
    directory="config/modules/navigation/" />
```

general.xml

  - More configuration paths have to be set, since we are working in this chapter with the UserDatabase and later on with the representation of the user too

```
<avatar configFile="avatar.xml"/>
```

userDatabase.xml

  - In the module configuration file the Navigation modules configuration file has to be provided

```
<module name="Navigation" configFile="navigation.xml" />
```

modules.xml

  - The configuration of the system might seem cumbersome so far, typically standard configurations are used and only single parts for components are to be exchanged

Navigation and Skybox

- Adding *inVRs* Components
  - Initialization process has to be changed
    - Paths for modules and interfaces are requested
    - The interfaces and the modules have to be configured as well
    - Thus the overloaded configure function is used

```
// !!!!!! Remove part of Snippet-1-2 (right above)
// in addition to the SystemCore config file, modules and interfaces config
// files have to be loaded.
std::string modulesConfigFile = Configuration::getString(
      "Modules.modulesConfiguration");
std::string inputInterfaceConfigFile = Configuration::getString(
    "Interfaces.inputInterfaceConfiguration");

if (!SystemCore::configure(systemCoreConfigFile, outputInterfaceConfigFile,
    inputInterfaceConfigFile, modulesConfigFile)) {
  printf("Error: failed to setup SystemCore!\n");
  printf("Please check if the Plugins-path is correctly set to the inVRs-lib
      directory in the ");
  printf("'final/config/general.xml' config file, e.g.:\n");
  printf("<path name=\"Plugins\" path=\"/home/guest/inVRs/lib\"/>\n");
  return -1;
}
```

Snippet 2-1

- Watch out with inserting this snippet code, parts above have to be removed

Navigation and Skybox

- Adding *inVRs* Components
  - Both callbacks request pointers to the components at the initialization

```
void initInputInterface(ModuleInterface* moduleInterface) {
  // store ControllerManger and the Controller as soon as the ControllerManager
  // is initialized
  if (moduleInterface->getName() == "ControllerManager") {
    controllerManager = (ControllerManager*)moduleInterface;
    controller = (Controller*)controllerManager->getController();
  }
}
```

Snippet 2-2

  - Watch out when inserting this snippet, it contains cross references to others

```
void initModules(ModuleInterface* module) {
  // store the Navigation as soon as it is initialized
  if (module->getName() == "Navigation") {
    navigation = (Navigation*)module;
  }
  //----------------------------------------------------------------------//
  // Snippet-4-1                                                          //
  //----------------------------------------------------------------------//

  //----------------------------------------------------------------------//
  // Snippet-5-1                                                          //
  //----------------------------------------------------------------------//
}
```

Snippet 2-3

JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Navigation and Skybox

interactive networked virtual reality system

- Adding *inVRs* Components
  - Finally the callback functions for the interfaces and the modules have to be registered

```
// register callbacks
InputInterface::registerModuleInitCallback(initInputInterface);
SystemCore::registerModuleInitCallback(initModules);
```

Snippet 2-4

  - As with the XML configuration this approach might seem cumbersome, but is very generic and can be avoided by using an additional helper class
  - This helper class is introduced in the following tutorial the *Going Immersive Tutorial*

JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Navigation and Skybox

interactive networked virtual reality system

- Navigation Concepts
  - Navigation or travel consists of three independent models
    - Speed
    - Orientation
    - Translation
  - Are composed by the Navigation module in order to create a resulting transformation
  - This transformation is to be passed on an object (e.g. camera, avatar)
  - Typically the camera is used, but effects like dangling camera as known from third-person VEs can be implemented easily
  - Each model takes data from an abstract controller which is part of the input interface
    - The controller takes input from devices and exposes it as
      - Buttons (boolean values)
      - Axes (linear values)
      - Sensors (6DOF position and orientation values)

Navigation and Skybox

- Implementing Navigation
  - At first gather pointers to the relevant objects
    - User
    - Camera
    - Avatar
  - The camera and the avatar object are retrieved from the user object
  - In our case we are working with the local user
    - These objects can be retrieved as well from remote users once they are connect and the data is stored in the UserDatabase
  - The display of the avatar is set to false
  - The initial transformation of the user is retrieved from the environment
    - The entryPoint as described in the previous chapter is taken into account
  - This transformation is then set on the navigated transformation of the user object

JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Navigation and Skybox

interactive networked virtual reality system

- Implementing Navigation

```
// fetch users camera, it is used to tell the Navigator where we are
localUser = UserDatabase::getLocalUser();
if (!localUser) {
  printd(ERROR, "Error: Could not find localUser!\n");
  return -1;
}

camera = localUser->getCamera();
if (!camera) {
  printd(ERROR, "Error: Could not find camera!\n");
  return -1;
}

avatar = localUser->getAvatar();
if (!avatar) {
  printd(ERROR, "Error: Could not find avatar!\n");
  return -1;
}
avatar->showAvatar(false);

// set our transformation to the start transformation
TransformationData startTrans =
  WorldDatabase::getEnvironmentWithId(1)->getStartTransformation(0);
localUser->setNavigatedTransformation(startTrans);
```

Snippet 2-5

Navigation and Skybox

- Implementing Navigation
  - The OpenSG SimpleSceneManger functionality has to be decoupled
  - Pointer to the Navigator is retrieved from the manager and disabled
  - A Timer is initialized
  - The camera matrix is initialized

```
// Navigator is part of SimpleSceneManager and not of the inVRs framework
Navigator *nav = mgr->getNavigator();
nav->setMode(Navigator::NONE);      // turn off the navigator
lastTimeStamp = timer.getTime();    // initialize timestamp;
camMatrix = gmtl::MAT_IDENTITY44F; // initial setting of the camera matrix
```

Snippet 2-6

  - Navigation and input processing, as well as timer updates and transformation management has to take place on a per-frame basis
  - The display() - method which was registered as GLUT callback has to be altered

JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Navigation and Skybox

INVRS interactive networked virtual reality system

- Implementing Navigation
  - The timestamp is updated and a delta value describing the time difference from the last update is set
  - The abstract controller is updated
    - Devices are polled and their values are exposed
  - Navigation is updated based on these values
  - The TransformationManager processes the navigation
  - The transformation matrix of the *inVRs* camera is requested

```
float currentTimeStamp;
Matrix osgCamMatrix;
float dt; // time difference between currentTimestamp and lastTimestamp

currentTimeStamp = timer.getTime(); //get current time
dt = currentTimeStamp - lastTimeStamp;

controller->update();    // poll/update associated devices
navigation->update(dt);  // update navigation

// process transformations which belong to the pipes with priority 0x0E000000
TransformationManager::step(dt, 0x0E000000);

camera->getCameraTransformation(camMatrix); // get camera transformation
```

Snippet 2-7

JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Navigation and Skybox

INVRS
interactive networked virtual reality system

- Implementing Navigation
  - The *inVRs* camera matrix is converted in to a OpenSG camera matrix
  - The navigator of the SimpleSceneManager is requested
  - The converted matrix is set in the navigator
  - The TransformationManager is invoked to process the remaining transformations (more detail on this in the next chapter)
  - The timeStamp is kept as the previous timeStamp

```
set(osgCamMatrix, camMatrix);        // convert gmtl matrix into OpenSG matrix

Navigator* nav = mgr->getNavigator();
nav->set(osgCamMatrix);              // plug new camera matrix into navigator

TransformationManager::step(dt);    // process the remaining pipes

lastTimeStamp = currentTimeStamp;
```

Snippet 2-8

JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Navigation and Skybox

INVRS
interactive networked virtual reality system

- Decoupling the OpenSG Input – Integrating *inVRs* Input Handling
  - Alter the predefined GLUT – callback functions
  - Data is now passed to *inVRs* (GlutMouseDevice and GlutKeyboardDevice) instead of the SimpleSceneManager
  - Gather knowledge on current window size for calculation of the cursor position

```
// the mouse device must be aware of the window size in pixel
GlutMouseDevice::setWindowSize(w, h);
```

Snippet 2-9

  - Poll the mouse button state

```
// (replace the lines above in this function) instead of calling the
// SimpleSceneManager we delegate the message to our mouse device
GlutMouseDevice::cbGlutMouse(button, state, x, y);
```

Snippet 2-10

  - Gather input on the mouse movement

```
// (replace the lines above in this function) instead of calling the
// SimpleSceneManager we delegate the message to our mouse device
GlutMouseDevice::cbGlutMouseMove(x, y);
```

Snippet 2-11

  - Gather input on pressed keyboard keys

```
// notify keyboard device about GLUT message
GlutCharKeyboardDevice::cbGlutKeyboard(k, x, y);
```

Snippet 2-12

Navigation and Skybox

- Decoupling the OpenSG Input – Integrating *inVRs* Input Handling
  - Additional keys should be checked to trap the mouse cursor in the active window

```
// grab the mouse
case 'm':
case 'M': {
  grabMouse = !grabMouse;
  GlutMouseDevice::setMouseGrabbing(grabMouse);
} break;
```

Snippet 2-13

  - Gather input on released keyboard keys

```
GlutCharKeyboardDevice::cbGlutKeyboardUp(k, x, y);
```

Snippet 2-14

- Configuring the Navigation
  - Each model has its specific configuration
  - Typically the arguments passed to the models perform a mapping from the abstract input into model specific parameters
  - Setup in navigation.xml
  - Many predefined models exist

JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Navigation and Skybox

interactive networked virtual reality system

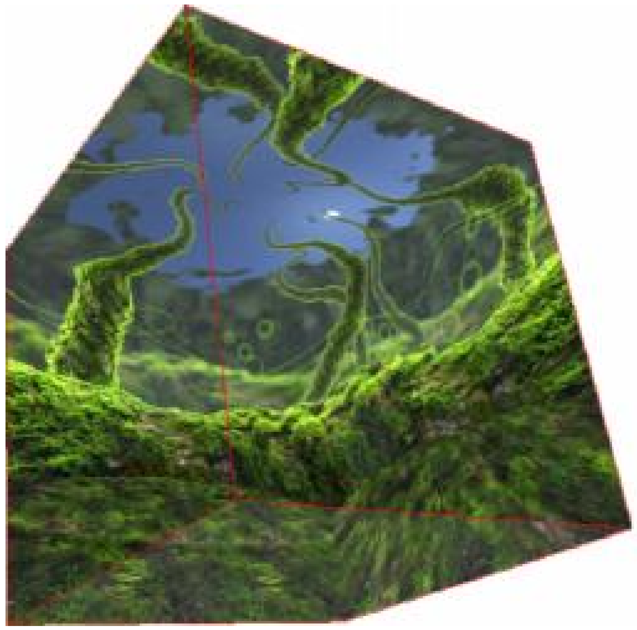
- Configuring the Navigation

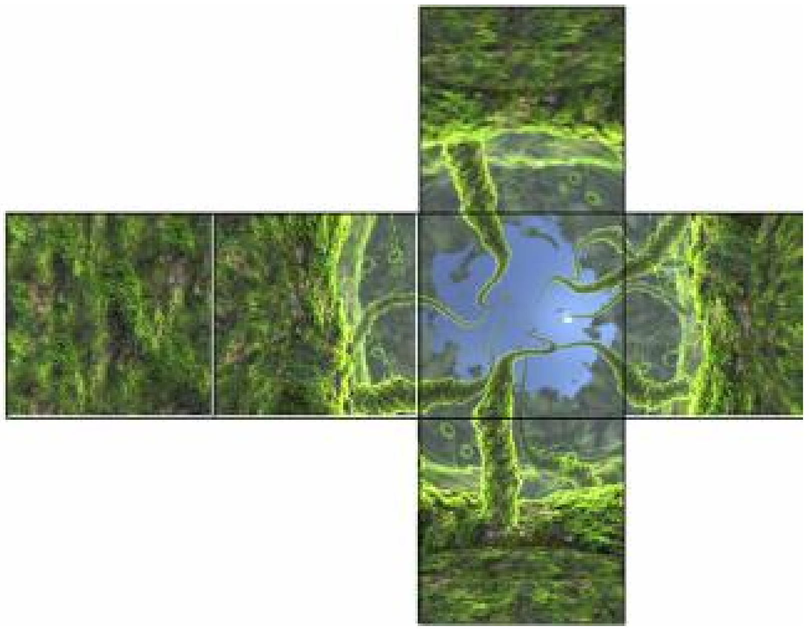
```
<?xml version="1.0"?>
<!DOCTYPE navigation SYSTEM "http://dtd.inVRs.org/navigation_v1.0a4.dtd">
<navigation version="1.0a4">
  <translationModel type="TranslationViewDirectionButtonStrafeModel">
    <arguments>
      <arg key="frontIndex" type="uint" value="3"/>
      <arg key="backIndex" type="uint" value="4"/>
      <arg key="leftIndex" type="uint" value="5"/>
      <arg key="rightIndex" type="uint" value="6"/>
    </arguments>
  </translationModel>
  <orientationModel type="OrientationDualAxisModel" angle="20">
    <arguments>
      <arg key="xAxisIndex" type="int" value="0"/>
      <arg key="yAxisIndex" type="int" value="1"/>
      <arg key="buttonIndex" type="int" value="1"/>
    </arguments>
  </orientationModel>
  <speedModel type="SpeedMultiButtonModel" speed="10">
    <arguments>
      <arg key="accelButtonIndices" type="string" value="3 4 5 6"/>
    </arguments>
  </speedModel>
</navigation>
```

navigation.xml

Navigation and Skybox

- Skybox Tool
  - Skyboxes are used to represent scene surroundings
  - Skyboxes move with the camera
  - The Skybox orientation is always fixed to the scene
  - Typically 6 textures mapped on cubes, *inVRs* offers different box shapes





Navigation and Skybox

- Adding a Skybox
  - The path is retrieved from the configuration and a skybox is set up with 6 image files
  - Dimensions and far clipping plane have to be set as well

```
// generate and configure the SkyBox
std::string skyPath = Configuration::getPath("Skybox");
skybox.init(5,5,5, 1000, (skyPath+"lostatseaday/lostatseaday_dn.jpg").c_str(),
  (skyPath+"lostatseaday/lostatseaday_up.jpg").c_str(),
  (skyPath+"lostatseaday/lostatseaday_ft.jpg").c_str(),
  (skyPath+"lostatseaday/lostatseaday_bk.jpg").c_str(),
  (skyPath+"lostatseaday/lostatseaday_rt.jpg").c_str(),
  (skyPath+"lostatseaday/lostatseaday_lf.jpg").c_str());
```

Snippet 2-15

  - The Skybox is internally generated and a node pointer containing the Skybox scene graph is requested
  - This node is attached to the top node of the scene

```
root->addChild(skybox.getNodePtr());
```

Snippet 2-16

  - The box position is set to the camera position

```
skybox.setupRender(camera->getPosition());
```

Snippet 2-17

  - Compile and run

JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Navigation and Skybox

INVRS
interactive networked virtual reality system

Chapter 3
Transformation Management

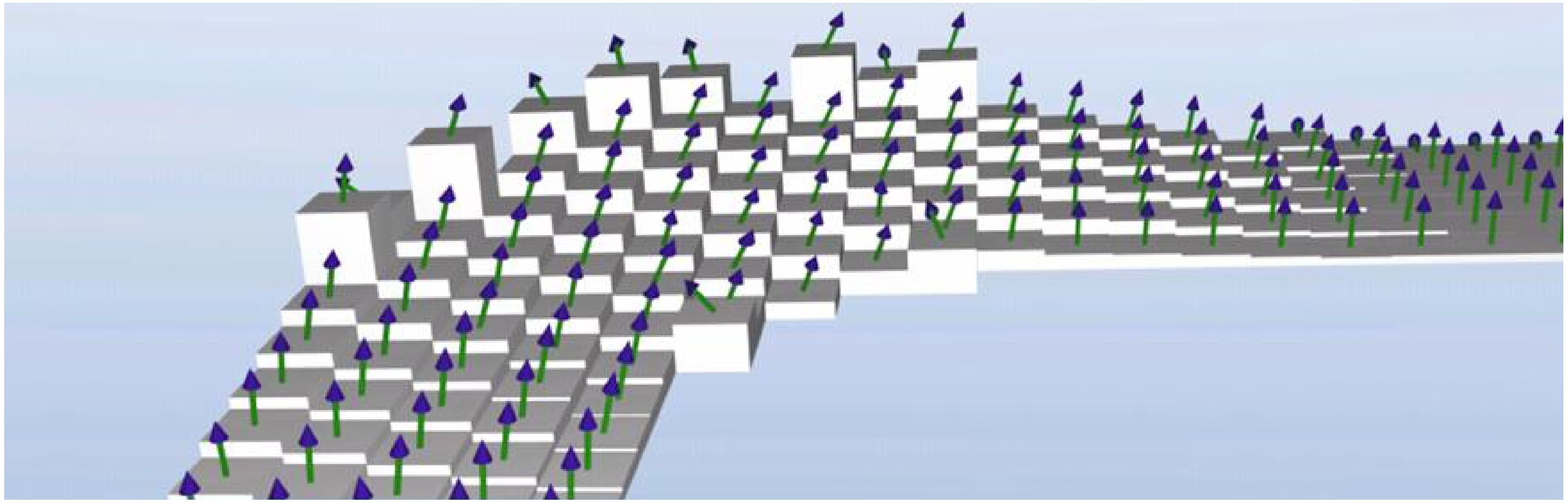
Where the navigation results are altered that the terrain can be followed and collisions with the buildings can be detected and resolved.

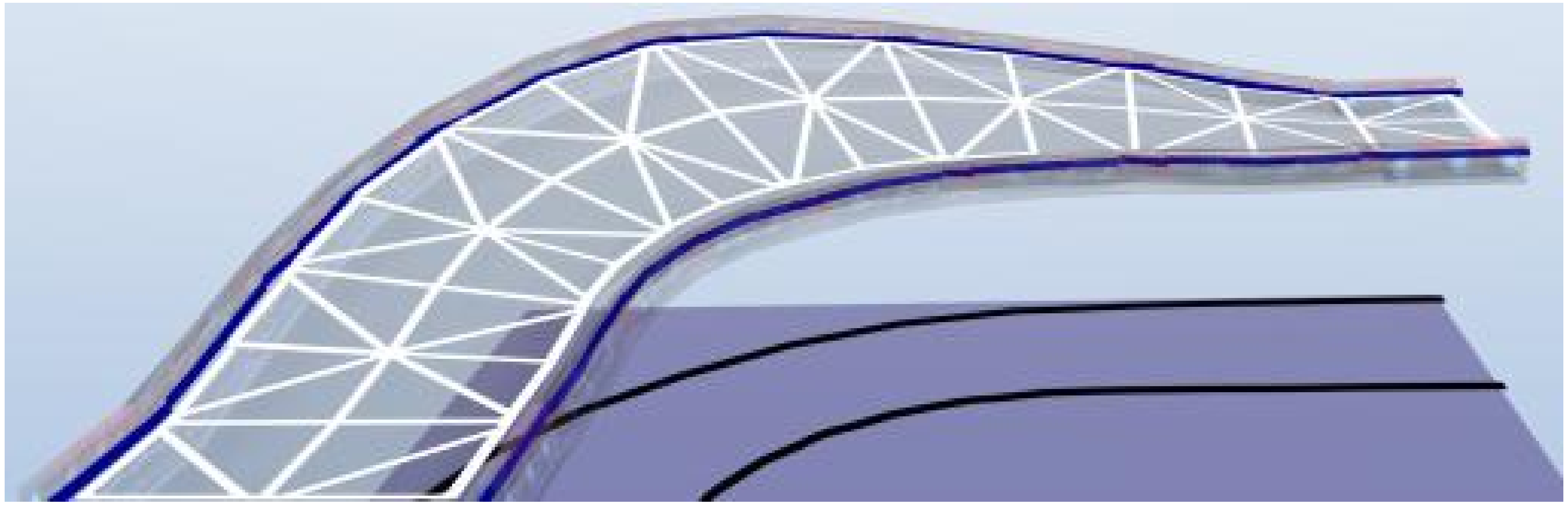
Transformation Management

- Height Maps and Collision Maps
  - Height maps
    - 2D Grid structure defining the height value underneath a 3D geometry
    - Additional normal vectors are provided which describe the orientation of the triangles at given grid location
  - Collision Maps
    - Line sets defining non-passable areas in a VE
    - Mapping from 3D objects on a plane
    - Plane is placed in the scene
    - Lines are drawn where geometries at a defined height are cut
  - Both are implemented as modifiers and can alter the generated transformation matrix

Transformation Management

- Height Maps and Collision Maps





JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Transformation Management

INVRS interactive networked virtual reality system

- Generating Height Maps
  - Either offline or online
  - Take tiles as input
  - If height maps have to be generated during runtime this can take a significant amount of time
  - In case no pre-defined height map is available for a tile and the method generate is called a height map will be automatically created
  - The geometry of a tile sampled and the grid is generated

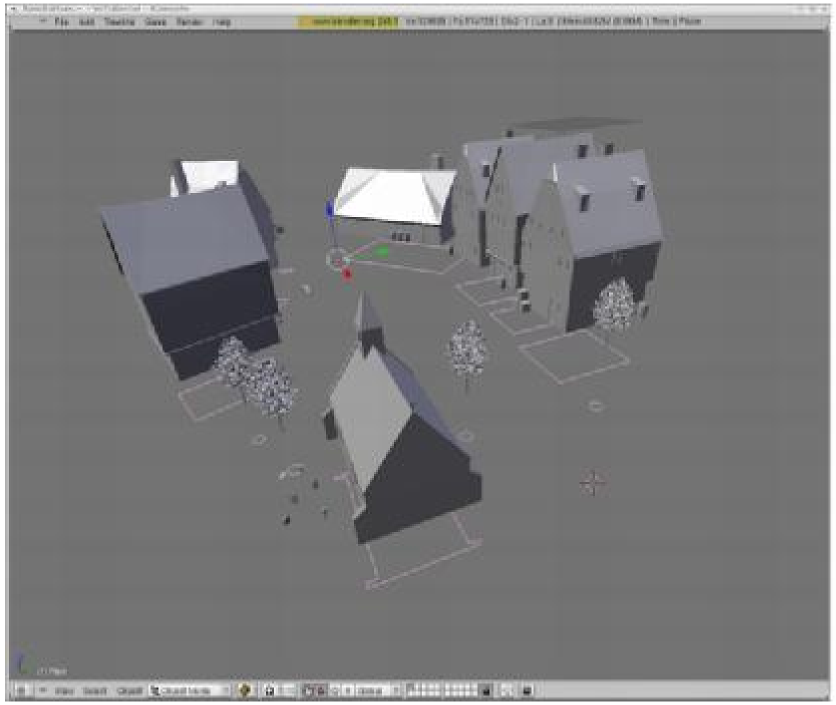
```
HeightMapManager::generateTileHeightMaps();
```

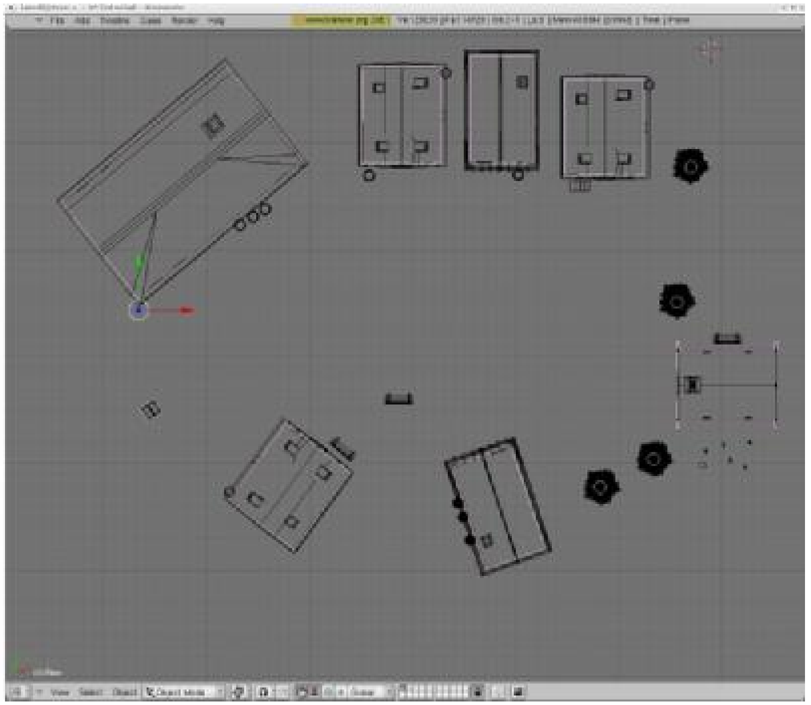
Snippet 3-1

- Generating Collision Maps
  - Always have to be created offline
  - Can be generated automatically by an external tool
  - Modeling tools (e.g. Blender, MAYA, 3D Max) are often used to manually draw collision lines

Transformation Management

- Generating Collision Maps
  - Typically the final scene is dumped to disk from the application
  - It is loaded into the modeling tool
  - The lines are drawn from top view
  - The line sets are again written to disc in a VRML file format





JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Transformation Management

interactive networked virtual reality system

- Transformation Manager Architecture
  - Consists of Pipes and Modifiers
  - TransformationData is sent from a source component to a destination component via the TransformationManager
  - At the destination component the transformation is to be applied on a target
- Pipes
  - Transformations are routed through pipes inside the transformation manager
  - Have a key consisting of source and destination component, target, received from network, etc.
  - Have several stages where the transformation can be modified
- Modifiers
  - Pipe stage
  - Different types e.g. for reading and writing data from and to external components exist

Transformation Management

- Setting up the Transformation Manager
  - Pipe is identified by a key
  - Contains list of modifiers

```xml
<?xml version="1.0"?>
<!DOCTYPE transformationManager SYSTEM "http://dtd.inVRs.org/
    transformationManager_v1.0a4.dtd">
<transformationManager version="1.0a4">
  <mergerList/>
  <pipeList>
    <pipe srcComponentName="NavigationModule"
        dstComponentName="TransformationManager" pipeType="Any"
        objectClass="Any" objectType="Any" objectId="Any"
        fromNetwork="0">
      <modifier type="ApplyNavigationModifier"/>

<!-- *************************** Snippet-3-1 ****************************** -->

<!-- *************************** Snippet-3-2 ****************************** -->

<!-- *************************** Snippet-5-3 ****************************** -->

      <modifier type="UserTransformationWriter"/>
      <modifier type="CameraTransformationWriter">

<!-- *************************** Snippet-3-3 ****************************** -->
```

modifiers.xml

Transformation Management

- Integrating the Height Maps and Collision Maps
  - Height map modifier has to be integrated after the navigated transformation is put in the pipe

```
<modifier type="HeightMapModifier" />
```

modifiers.xml

  - CheckCollisionModifier is applied afterwards
  - Radius for collision detection is to be set
  - Filename of the collision map has to be provided

```
<modifier type="CheckCollisionModifier">
  <arguments>
    <arg key="radius" type="float" value="1" />
    <arg key="fileName" type="string" value="MedievalTownCollisionMap.wrl"/>
  </arguments>
</modifier>
```

modifiers.xml

  - The height of the camera has to be modified

```
<arguments>
  <arg key="cameraHeight" type="float" value="1.8"/>
  <arg key="useGlobalYAxis" type="bool" value="true"/>
</arguments>
```

modifiers.xml

Transformation Management

- Adding the Transformation Managers' Setup
  - The factories for generating the used modifiers have to be registered since the modifiers are not provided as core features
  - All additional modifiers which are not provided by the SystemCore have to register their factories at the initialization callback

```
void initCoreComponents(CoreComponents comp) {
  // register factory for HeightMapModifier as soon as the
  // TransformationManager is initialized
  if (comp == TRANSFORMATIONMANAGER) {
    TransformationManager::registerModifierFactory
      (new HeightMapModifierFactory());
  // register factory for CheckCollisionModifier
    TransformationManager::registerModifierFactory
      (new CheckCollisionModifierFactory);
  }
}
```

Snippet 3-2

  - The TransformationManager has to be registered for a callback

```
  SystemCore::registerCoreComponentInitCallback(initCoreComponents);
```

Snippet 3-3

  - Compile and run

JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Transformation Management

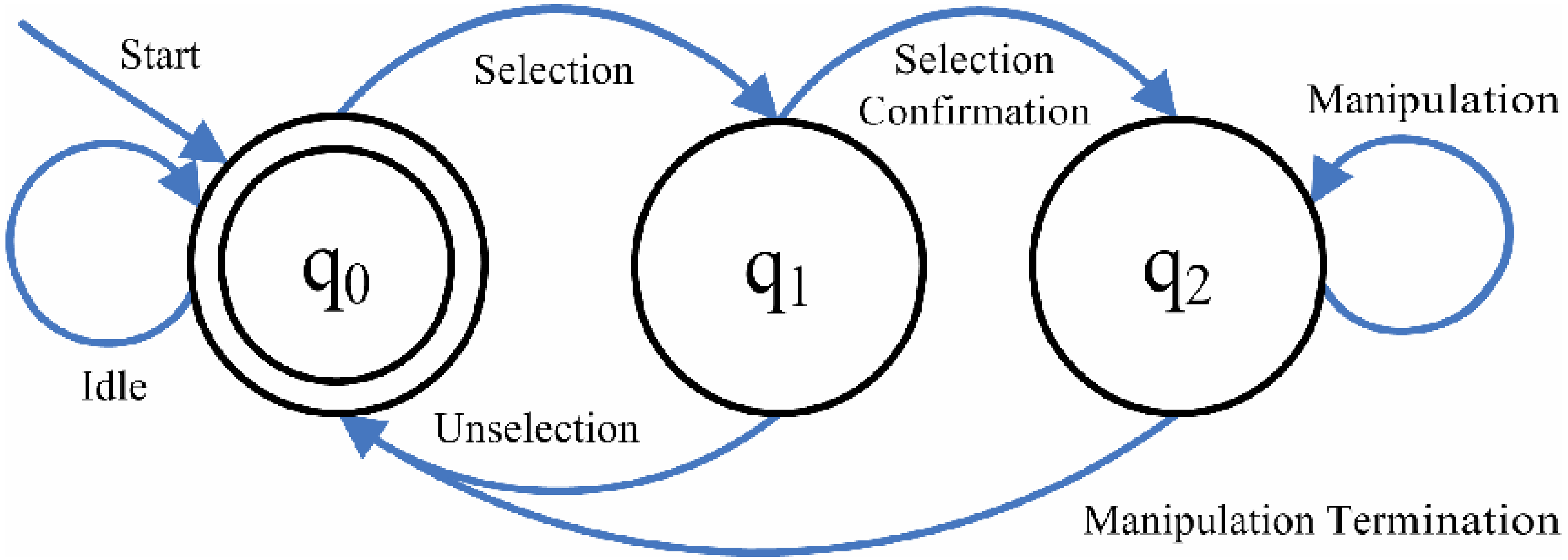
INVRS
interactive networked virtual reality system

Chapter 4
Interaction

Where the user is able to pick up and drop boxes and benches in the town.

Interaction

- Implemented as a deterministic finite automaton (DFA)
  - Input alphabet Σ defined through components of the abstract input device and system states
  - $q = \{q_0$ (Idle), q_1, (Selection) q_2, (Manipulation)$\}$
  - Transition function δ defined through interaction techniques
  - Start- and end state = q_0



JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Interaction

inVRS
interactive networked virtual reality system

- Configuring the State Machine

```
<?xml version="1.0"?>
<!DOCTYPE interaction SYSTEM "http://dtd.inVRs.org/interaction_v1.0a4.dtd">
<interaction version="1.0a4">

  <stateActionModels>
    <selectionActionModel type="HighlightSelectionActionModel">
      <arguments>
        <arg key="modelType" type="string" value="OSG"/>
        <arg key="modelPath" type="string" value="box.osg"/>
      </arguments>
    </selectionActionModel>
    <manipulationActionModel type="HomerManipulationActionModel">
      <arguments>
        <arg key="usePickingOffset" type="bool" value="true"/>
      </arguments>
    </manipulationActionModel>
  </stateActionModels>

  <stateTransitionModels>
    <selectionChangeModel type="LimitRayCastSelectionChangeModel">
      <arguments>
        <arg key="rayDistanceThreshold" type="float" value="5"/>
      </arguments>
    </selectionChangeModel>
    <unselectionChangeModel type="LimitRayCastSelectionChangeModel">
      <arguments>
```

interaction.xml

JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Interaction

INVRS
interactive networked virtual reality system

- Adding the Interaction Module to our Application
  - Set the path for the interaction configuration first

```
<path name="InteractionModuleConfiguration"
      directory="config/modules/interaction/" />
```
general.xml

  - Provide a configuration file for the Interaction module

```
<module name="Interaction" configFile="interaction.xml" />
```
modules.xml

  - Retrieve the module in callback function

```
// store the Interaction as soon as it is initialized
else if (module->getName() == "Interaction") {
  interaction = (Interaction*)module;
}
```
Snippet 4-1

Interaction

- Setting up the Transformation Management
  - For interaction the transformation pipe has to be altered again
  - In the navigation pipe the updated position has to write out the cursor position, which is always relative to the user

```
<modifier type="ApplyCursorTransformationModifier" />
<modifier type="CursorTransformationWriter" />
```

modifiers.xml

  - An additional pipe has to be configured, which takes as a source the Interaction module and as destination the WorldDatabase
  - The pipe is opened by the modules when needed
  - This pipe is the interaction pipe using modifiers for
    - Adding manipulation offset
    - Writing out the transformation

```
<pipe srcModuleName="InteractionModule" dstModuleName="WorldDatabase"
  pipeType="Any" objectClass="Any" objectType="Any" objectId="Any"
  fromNetwork="0">
  <modifier type="ManipulationOffsetModifier"/>
  <modifier type="TransformationDistributionModifier"/>
  <modifier type="EntityTransformationWriter" />
</pipe>
```

modifiers.xml

Interaction

- Adding a Cursor Representation
  - Helpful to have a visual representation of the cursor for selection and manipulation
  - Paths for model and representation have to be set as well

```
<path name="CursorConfig"
  path="config/systemcore/userdatabase/cursorRepresentation/" />
<path name="CursorModelConfiguration"
  path="config/systemcore/userdatabase/cursorModel/" />
```

general.xml

  - Additionally the configuration files have to be set in the UserDatabase

```
<cursorRepresentation configFile="handRepresentation.xml" />
<cursorTransformationModel configFile="homerCursorModel.xml" />
```

userDatabase.xml

- Invoking Interaction and Changing the Cursor Appearance
  - In the display loop the interaction has to be invoked for checking the transition functions, changing the state, writing transformations and event processing
  - The cursor is updated based on information of the state machine

```
interaction->step(dt);

UserDatabase::updateCursors(dt);
```

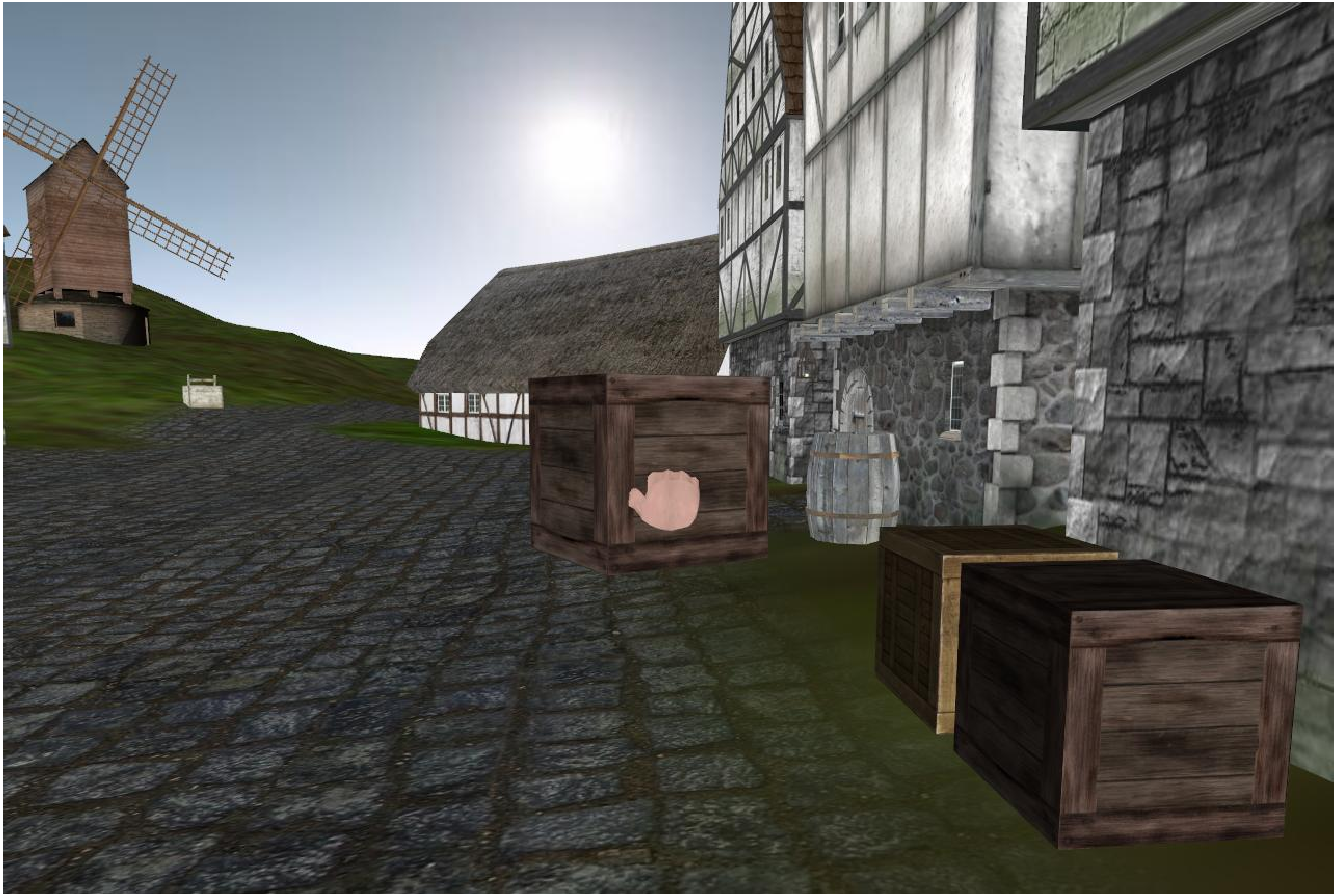
Snippet 4-2

  - Compile and execute

JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Interaction

INVRS
interactive networked virtual reality system



Chapter 5
Using Network Communication

Where the virtual world is shared with other participants and remote interaction and navigation can be perceived.

Using Network Communication

- Network Concepts
  - Module designed for exchangeability
    - Many implementations exist already
  - Standard module
    - Communication topology represents a complete graph
    - Databases fully replicated
  - Communication between components and network uses a high level interface
    - Sending and receiving of TransformationData and Events
    - Sending and receiving of user defined messages
    - Connection establishment and disconnecting

Using Network Communication

- Adding the Network Module to our Application
  - As usual set the path for the network configuration first

```
<path name="NetworkModuleConfiguration"
    directory="config/modules/network/" />
```

general.xml

  - Provide a configuration file for the Network module

```
<module name="Network" configFile="network.xml" />
```

modules.xml

  - Retrieve the module in callback function

```
// store the NetworkInterface as soon as it is initialized
else if (module->getName() == "Network") {
  network = (NetworkInterface*)module;
}
```

Snippet 5-1

  - Connection establishment using the first command line argument
  - Synchronization between the different instances (e.g. updates on the WorldDatabase)

```
// try to connect to network first command line argument is {hostname|IP}:port
if (argc > 1) {
  printf("Trying to connect to %s\n", argv[1]);
  network->connect(argv[1]);
}
SystemCore::synchronize();   // synchronize both VEs
```

Snippet 5-2

Using Network Communication

- Using the Network Module
  - An update on the SystemCore has to be triggered at the beginning of the display loop
  - Internal events like updating the UserDatabase and synchronizing the VEs can be subsequently trigged in in the core

```
SystemCore::step();  //update the system core, needed for event handling
```
Snippet 5-3

  - The ports which are used for communication are defined in network.xml
    - TCP is typically used for Events
    - UDP is typically used for TransformationData

```
<?xml version="1.0"?>
<!DOCTYPE network SYSTEM "http://dtd.inVRs.org/network_v1.0a4.dtd">
<network version="1.0a4">
  <ports TCP="8081" UDP="8082"/>
</network>
```
network.xml

- Altering the Transformation Management
  - In the interaction and navigation pipe a distributor has to be included
  - The current value at this stage is distributed to remote users and passed inside a similar pipe

Using Network Communication

```
    <modifier type="TransformationDistributionModifier" />
```
modifiers.xml

– An additional pipe for remote interaction input has to be set up
  - The attribute fromNetwork is in that case set to true

```
<pipe srcComponentName="InteractionModule" dstComponentName="WorldDatabase"
  pipeType="Any" objectClass="Any" objectType="Any" objectId="Any"
  fromNetwork="1">
  <modifier type="EntityTransformationWriter" />
</pipe>
```
modifiers.xml

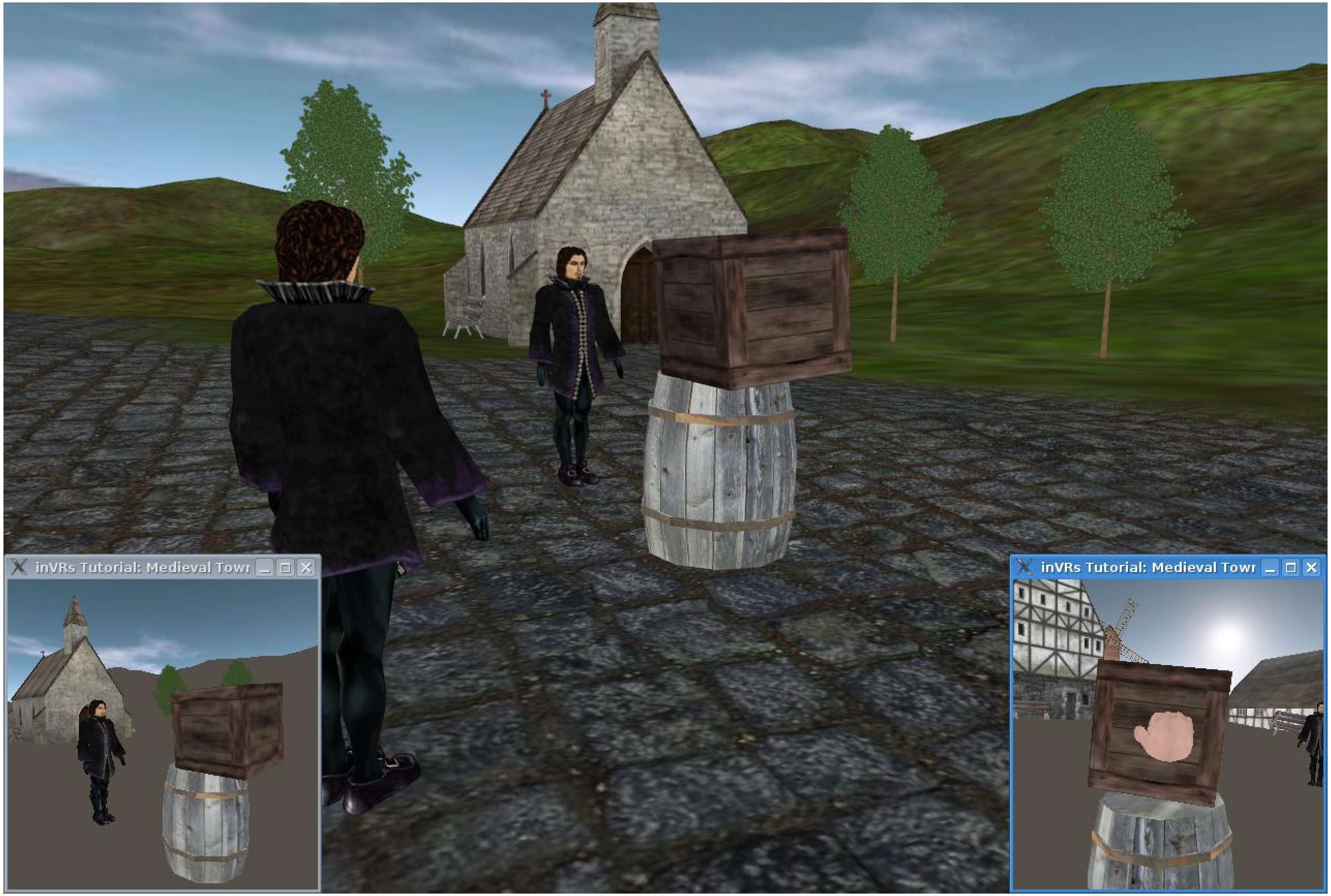
– An additional pipe for remote navigation input has to be set up
  - The attribute fromNetwork is in that case set to true

```
<pipe srcComponentName="NavigationModule" dstComponentName="
    TransformationManager"
  pipeType="Any" objectClass="Any" objectType="Any" objectId="Any"
  fromNetwork="1">
  <modifier type="UserTransformationWriter" />
  <modifier type="AvatarTransformationWriter" >
    <arguments>
      <arg key="clipRotationToYAxis" type="bool" value="true" />
    </arguments>
  </modifier>
</pipe>
```
modifiers.xml

JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Using Network Communication

INVRS
interactive networked virtual reality system



Chapter 6
Developing own Application Logic

Where new code is introduced into the application to make the sails of a windmill turn.

Developing own Application Logic

- Own Application Logic
  - Goal: to manipulate a scene graph of an entity based on user input
  - Gather exposed user input in display loop
  - Variable containing a rotational speed is used
  - In case the defined button is pressed speed is increased up to a threshold
  - If the button is not pressed the speed will be decreased to zero

```
if (controller->getButtonValue(2)) { // the right mouse button is pressed
  windMillSpeed += dt*0.5;           // increase speed of the windmill
  if (windMillSpeed > 2*M_PI) {
    windMillSpeed = 2*M_PI;
  }
} else if (windMillSpeed > 0) {      // pressing mouse button stopped
  windMillSpeed -= dt*0.5;           // decrease speed of windmill
} else if (windMillSpeed < 0) {
  windMillSpeed = 0;
}
```

Snippet-6-1

  - In next step the calculated speed has to be applied on the sails of the windmill model

Developing own Application Logic

- Own Application Logic
  - First get the entity from the WorldDatabase
  - Get the scene graph of the Entity
  - Get the sub scene graph of the previously retrieved scene graph with the name “Sails” (this name is stored in the loaded model)
  - Make sure it is a transformation node
  - Cast the *inVRs* scene graph node into an *inVRs* transformation node
  - Request the transformation from the node
  - Apply the rotational change to the transformation
  - Apply the final transformation on the transformation node

JOHANNES KEPLER UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Developing own Application Logic

INVRS
interactive networked virtual reality system

- Own application logic

```
if (windMillSpeed > 0) {                // rotate sails
  // retrieve the windmill entity
  Entity* windMill = WorldDatabase::getEntityWithEnvironmentId(1, 27);
  ModelInterface* windMillModel = windMill->getVisualRepresentation();

  // retrieve the windmill's sails
  SceneGraphNodeInterface* sceneGraphNode =
    windMillModel->getSubNodeByName("Sails");

  // make sure this node is a transformation node
  assert(sceneGraphNode->getNodeType() ==
    SceneGraphNodeInterface::TRANSFORMATION_NODE);
  TransformationSceneGraphNodeInterface* transNode =
    dynamic_cast<TransformationSceneGraphNodeInterface*>(sceneGraphNode);
  assert(transNode);

  // rotate the sails
  TransformationData trans = transNode->getTransformation();
  gmtl::AxisAnglef axisAngle(windMillSpeed*dt, 0, 0, 1);
  gmtl::Quatf rotation;
  gmtl::set(rotation, axisAngle);
  trans.orientation *= rotation;
  transNode->setTransformation(trans);
}
```

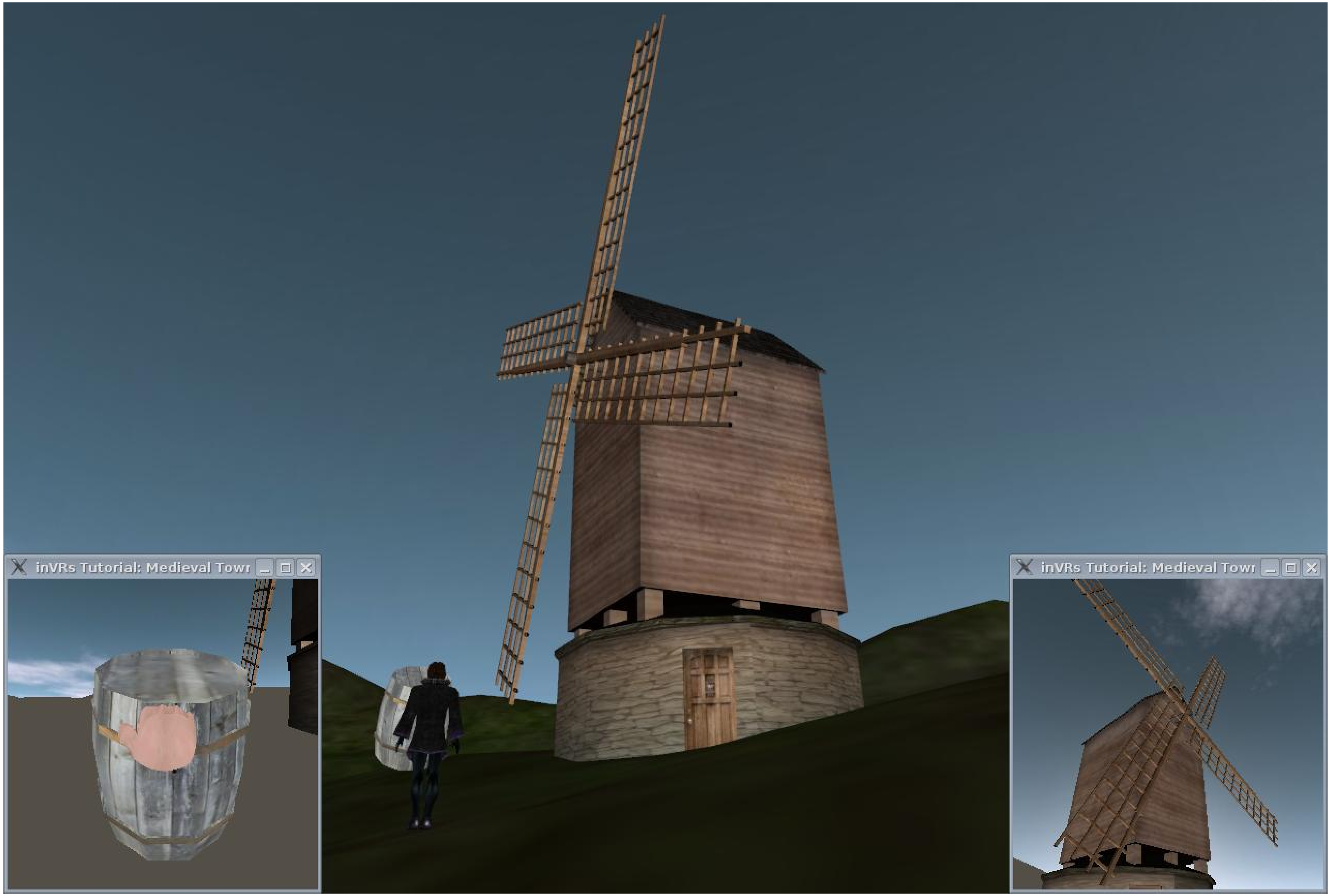
Snippet-6-1

- Recompile and execute

JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Developing own Application Logic

interactive networked virtual reality system



Outlook

Outlook

- Variety of additional tools are available
  - Modules
    - Physics Support (2D and 3D Physics)
    - Animation and Camera path
  - External Tools
    - Collaborative Editor
  - Scene Graph Specific Enhancements (OpenSG)
    - Avatars (with export from 3D Max, MAYA and Blender)
    - CAVESceneManager
    - Particle Wrapper
    - Skybox
    - 3D Menu System

Outlook

- Novel concepts
  - Clear distinction in the area of communication patterns
  - Events and transformations are handled independently
  - Transformation management allows for concurrent object manipulation
  - Composeable and clearly defined interaction and navigation
  - Automatic data distribution mechanism
  - Full abstraction from graphics, input, network, etc.; the developer should be able to focus on the application logic
  - Development and integration of networked physics module (Roland Landertshamer)

- Does not provide
  - Scripting support
  - Logic support

Acknowledgements

- Contributors
  - Architecture and Main Development

    Christoph Anthes, Helmut Bressler, Roland Landertshamer, Marina Lenger

  - Tools

    Helmut Garstenauer, Martin Garstenauer, Adrian Haffegee, Marlene Hochrieser, Roland Hopferwieser, Robert Owen, Stephan Reiter, Thomas Weberndorfer, Christian Wressnegger, Johannes Zarl

Bibliography

- Christoph Anthes; Paul Heinzlreiter; Gerhard Kurka & Jens Volkert, “Navigation Models for a Flexible, Multi-Mode VR Navigation Framework“, *ACM SIGGRAPH on Virtual Reality Continuum and Its Applications in Industry (VRCAI '04), ACM Press,* 2004, pages 476-479

- Christoph Anthes; Roland Landertshamer; Helmut Bressler & Jens Volkert, “Managing Transformations and Events in Networked Virtual Environments“ *ACM International MultiMedia Modeling Conference (MMM '07), Springer,* 2007*, 4352*, pages 722-729

- Christoph Anthes; Roland Landertshamer & Jens Volkert, “Physically-Based Interaction for Networked Virtual Environments.“, *International Conference on Computational Science (ICCS '07), Springer,* 2007*, 4488*, pages 776-783

- Christoph Anthes & Jens Volkert, “inVRs - A Framework for Building Interactive Networked Virtual Reality Systems“, *International Conference on High Performance Computing and Communications (HPCC '06), Springer,* 2006*, 4208*, pages 894-904

Bibliography

- Christoph Anthes; Alexander Wilhelm; Roland Landertshamer; Helmut Bressler & Jens Volkert, “Net’’O’’Drom -- An Example for the Development of Networked Immersive VR Applications“, *International Conference on Computational Science (ICCS '07), Springer,* 2007*, 4488*, pages 752-759

- Helmut Bressler; Roland Landertshamer; Christoph Anthes & Jens Volkert, “An Efficient Physics Engine for Virtual Worlds“, *medi@terra '06,* 2006, pages 152-158

- Adrian Haffegee; Ronan Jamieson; Christoph Anthes & Vassil N. Alexandrov, “Tools For Collaborative VR Application Development“, *International Conference on Computational Science (ICCS '05), Springer,* 2005*, 3516*, pages 350-358