#include <winsock2.h>

#include <stdlib.h>

#include <stdio.h>

#include <iostream>

#include <fstream>

#include <dos.h>

#include "chai3d.h"

#include <GLFW/glfw3.h>

#include <chrono>

#include <algorithm>

#include <stack>

#include <numeric>

#include <valarray>

#include <conio.h>

#include <ctime>

#include <thread>

#pragma comment(lib, "ws2\_32.lib")

////////////////////////////////////////////////////////////////////////////////

using namespace chai3d;

using namespace std;

////////////////////////////////////////////////////////////////////////////////

// Device Param.

////////////////////////////////////////////////////////////////////////////////

#define DOF 3

double pos\_curr[DOF] = { 0, 0, 0 };

double pos\_prev[DOF] = { 0, 0, 0 };

double pos\_recv[DOF] = { 0, 0, 0 };

double Fd[DOF];

double XsdP[DOF];

double DXsd[DOF];

double Xsd[DOF];

////////////////////////////////////////////////////////////////////////////////

// UDP Param.

////////////////////////////////////////////////////////////////////////////////

#define BUFSIZE 4096

#define SERVER\_IP "192.168.0.100" //"127.0.0.1"

#define UDP\_PORT 4000

#define SENDDATA\_NUM 3

#define READDATA\_NUM 3

int a = 0, b = 0, c = 0, d = 0, e = 0, f = 0, g = 0, h = 0, j = 0, k = 0, l = 0, m = 0, o = 0, q = 0, r = 0, s = 0, t = 0;

int retval;

WSADATA wsa;

SOCKET sock;

// µ¥ÀÌÅÍ Åë½Å¿¡ »ç¿ëÇÒ º¯¼ö

SOCKADDR\_IN serveraddr;

SOCKADDR\_IN clientaddr;

int addrlen;

char sbuf[BUFSIZE + 1], rbuf[BUFSIZE + 1];

float SendData[SENDDATA\_NUM], ReadData[READDATA\_NUM];

bool isFirstRecv = false;

////////////////////////////////////////////////////////////////////////////////

// Delay Param.

////////////////////////////////////////////////////////////////////////////////

#define DELAY 1

#define DELAYDATA\_NUM 3

double ClientData[DELAYDATA\_NUM];

double DelayDataBuf[DELAYDATA\_NUM][DELAY];

double DelayData[DELAYDATA\_NUM];

////////////////////////////////////////////////////////////////////////////////

// GUI - GENERAL SETTINGS

////////////////////////////////////////////////////////////////////////////////

// stereo Mode

/\*

C\_STEREO\_DISABLED: Stereo is disabled

C\_STEREO\_ACTIVE: Active stereo for OpenGL NVDIA QUADRO cards

C\_STEREO\_PASSIVE\_LEFT\_RIGHT: Passive stereo where L/R images are rendered next to each other

C\_STEREO\_PASSIVE\_TOP\_BOTTOM: Passive stereo where L/R images are rendered above each other

\*/

cStereoMode stereoMode = C\_STEREO\_DISABLED;

// fullscreen mode

bool fullscreen = false;

double Xd[DOF];

double DXd[DOF];

double dFe[DOF];

double XdP[DOF];

//////////////////////////////////////////////////////////////////////////

// Device Parameters //

//////////////////////////////////////////////////////////////////////////

double POS, HIP, pHIP, IHIP, DHIP, vHIP;

double FOR, pFOR, FORCE, F\_sensing, T\_sensing;

double Vd, Xe, XeP, Ve, Fe, FdP, rDXd;

double MaxForce = 8.5;

//////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////

// EBA Parameters //

//////////////////////////////////////////////////////////////////////////

//double c1, c2; // N/m

//VECTOR uBETA, BETA, Bmax, Bmin;

double uBETA[3], BETA[3], Bmax[3], Bmin[3];

//c1=b/T

//b=KT/2

//high force K = 20kN/m

//b=10

//c1=0.01

double c1 = 50;

double c2 = 100;

//extern lwtButton bEBA;

//double hRATE, cRATE; // Hz

double Th, Tc;

double Ke = 1.0;

//////////////////////////////////////////////////////////////////////////

// Flag Parameters //

//////////////////////////////////////////////////////////////////////////

bool EnableMultiRate = false;

bool EnableEBA = true;

bool EnableImpulse = false;

bool EnableMoving = false;

//////////////////////////////////////////////////////////////////////////

// mirrored display

bool mirroredDisplay = false;

double delayn = 0;

double delaym = 0;

double range = 0;

int i;

double sum = 0;

double average = 0;

int n = 0;

double p[500];

int countd = 0;

int timeUDPS;

int timeUDPR;

double\* arr2;

double arr[100];

double\* pvalue = 0;

//float\* p = new float[(int)i];

stack<double> delayarray;

std::vector<double> delay2;

struct Timer

{

chrono::time\_point<std::chrono::steady\_clock>start, end;

chrono::duration<float>duration;

Timer()

{

start = std::chrono::high\_resolution\_clock::now();

}

~Timer()

{

end = std::chrono::high\_resolution\_clock::now();

duration = end - start;

float ms = duration.count() \* 1000.0f;

timeUDPR = ms;

//cout << "Timer took" << ms << "ms" << endl;

}

};

struct Timer2

{

chrono::time\_point<std::chrono::steady\_clock>start, end;

chrono::duration<float>duration;

Timer2()

{

start = std::chrono::high\_resolution\_clock::now();

}

~Timer2()

{

end = std::chrono::high\_resolution\_clock::now();

duration = end - start;

float ms = duration.count() \* 1000.0f;

timeUDPS = ms;

//cout << "Timer took" << ms << "ms" << endl;

}

};

////////////////////////////////////////////////////////////////////////////////

// GUI - DECLARED VARIABLES

////////////////////////////////////////////////////////////////////////////////

// a world that contains all objects of the virtual environment

cWorld\* world;

// a camera to render the world in the window display

cCamera\* camera;

// a light source to illuminate the objects in the world

cDirectionalLight\* light;

// a haptic device handler

cHapticDeviceHandler\* handler;

// a pointer to the current haptic device

cGenericHapticDevicePtr hapticDevice;

// a label to display the haptic device model

cLabel\* labelHapticDeviceModel;

// a label to display the position [m] of the haptic device

cLabel\* labelHapticDevicePosition;

// a global variable to store the position [m] of the haptic device

cVector3d hapticDevicePosition;

// a global variable to store the velocity [m/s] of the haptic device

cVector3d hapticDeviceVelocity;

// a font for rendering text

cFontPtr font;

// a label to display the rate [Hz] at which the simulation is running

cLabel\* labelRates;

cLabel\* labelMax;

cLabel\* labelMin;

cLabel\* labelXd;

cLabel\* labelXd\_Prev;

cLabel\* labelX\_Recv;

cLabel\* labelFd;

cLabel\* labeldFe;

cLabel\* labelIP;

cLabel\* labelDelay;

cLabel\* labelDashes\_1;

cLabel\* labelServerLabel;

cLabel\* labelDashes\_2;

cLabel\* labelKeyboardOptions;

cLabel\* labelPotentialField;

cLabel\* labelExitApplication;

cLabel\* labelScopeInfo;

cLabel\* labelScope1Info;

cLabel\* labelMirror;

cLabel\* labelToggleScreen;

cLabel\* labelavg;

cLabel\* labelmax;

cLabel\* labelmax2;

// a small sphere (cursor) representing the haptic device

cShapeTorus\* cursor;

// a line representing the velocity of the haptic device

cShapeLine\* velocity;

// a scope to monitor position values of haptic device

cScope\* scope;

cScope\* scope1;

// a flag for using force field (ON/OFF)

bool useForceField = true;

// a flag to indicate if the haptic simulation currently running

bool simulationRunning = false;

// a flag to indicate if the haptic simulation has terminated

bool simulationFinished = true;

// a frequency counter to measure the simulation graphic rate

cFrequencyCounter freqCounterGraphics;

// a frequency counter to measure the simulation haptic rate

cFrequencyCounter freqCounterHaptics;

// haptic thread

cThread\* hapticsThread;

// a handle to window display context

GLFWwindow\* window = NULL;

// current width of window

int width = 0;

// current height of window

int height = 0;

// swap interval for the display context (vertical synchronization)

int swapInterval = 1;

////////////////////////////////////////////////////////////////////////////////

// GUI - DECLARED FUNCTIONS

////////////////////////////////////////////////////////////////////////////////

// callback when the window display is resized

void windowSizeCallback(GLFWwindow\* a\_window, int a\_width, int a\_height);

// callback when an error GLFW occurs

void errorCallback(int error, const char\* a\_description);

// callback when a key is pressed

void keyCallback(GLFWwindow\* a\_window, int a\_key, int a\_scancode, int a\_action, int a\_mods);

// this function renders the scene

void updateGraphics(void);

// this function contains the main haptics simulation loop

void updateHaptics(void);

// this function closes the application

void close(void);

void InitUDP(void);

void modifiedEBA(void);

void CloseUDP();

void UDPSendData(float sData[SENDDATA\_NUM]);

void UDPReadData();

void err\_quit(const char\* msg);

void err\_display(const char\* msg);

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

int main(int argc, char\* argv[])

{

InitUDP();

////////////////////////////////////////////////////////////////////////////////

// INITIALIZATION

////////////////////////////////////////////////////////////////////////////////

cout << endl;

cout << "-----------------------------------" << endl;

cout << "Tele-HandShake Client" << endl;

cout << "-----------------------------------" << endl << endl << endl;

cout << "Keyboard Options:" << endl << endl;

cout << "[1] - Enable/Disable potential field" << endl;

cout << endl;

cout << "[q] - Exit application" << endl;

cout << endl << endl;

////////////////////////////////////////////////////////////////////////////////

// OPEN GL - WINDOW DISPLAY

////////////////////////////////////////////////////////////////////////////////

// initialize GLFW library

if (!glfwInit())

{

cout << "failed initialization" << endl;

cSleepMs(1000);

return 1;

}

// set error callback

glfwSetErrorCallback(errorCallback);

// compute desired size of window

const GLFWvidmode\* mode = glfwGetVideoMode(glfwGetPrimaryMonitor());

int w = 1.2 \* mode->height;

int h = 0.7 \* mode->height;

int x = 0.7 \* (mode->width - w);

int y = 0.7 \* (mode->height - h);

// set OpenGL version

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 2);

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 1);

// set active stereo mode

if (stereoMode == C\_STEREO\_ACTIVE)

{

glfwWindowHint(GLFW\_STEREO, GL\_TRUE);

}

else

{

glfwWindowHint(GLFW\_STEREO, GL\_FALSE);

}

// create display context

window = glfwCreateWindow(w, h, "Tele-HandShake (Client)", NULL, NULL);

if (!window)

{

cout << "failed to create window" << endl;

cSleepMs(1000);

glfwTerminate();

return 1;

}

// get width and height of window

glfwGetWindowSize(window, &width, &height);

// set position of window

glfwSetWindowPos(window, x, y);

// set key callback

glfwSetKeyCallback(window, keyCallback);

// set resize callback

glfwSetWindowSizeCallback(window, windowSizeCallback);

// set current display context

glfwMakeContextCurrent(window);

// sets the swap interval for the current display context

glfwSwapInterval(swapInterval);

#ifdef GLEW\_VERSION

// initialize GLEW library

if (glewInit() != GLEW\_OK)

{

cout << "failed to initialize GLEW library" << endl;

glfwTerminate();

return 1;

}

#endif

////////////////////////////////////////////////////////////////////////////////

// WORLD - CAMERA - LIGHTING

////////////////////////////////////////////////////////////////////////////////

// create a new world.

world = new cWorld();

// set the background color of the environment

world->m\_backgroundColor.setBlack();

// create a camera and insert it into the virtual world

camera = new cCamera(world);

world->addChild(camera);

// position and orient the camera

camera->set(cVector3d(0.5, 0.0, 0.0), // camera position (eye)

cVector3d(0.0, 0.0, 0.0), // look at position (target)

cVector3d(0.0, 0.0, 1.0)); // direction of the (up) vector

// set the near and far clipping planes of the camera

camera->setClippingPlanes(0.01, 10.0);

// set stereo mode

camera->setStereoMode(stereoMode);

// set stereo eye separation and focal length (applies only if stereo is enabled)

camera->setStereoEyeSeparation(0.005);

camera->setStereoFocalLength(0.5);

// set vertical mirrored display mode

camera->setMirrorVertical(mirroredDisplay);

// create a directional light source

light = new cDirectionalLight(world);

// insert light source inside world

world->addChild(light);

// enable light source

light->setEnabled(true);

// define direction of light beam

light->setDir(-1.0, 0.0, 0.0);

// create a sphere (cursor) to represent the haptic device

cursor = new cShapeTorus(0.03, 0.05);

// insert cursor inside world

world->addChild(cursor);

// create small line to illustrate the velocity of the haptic device

velocity = new cShapeLine(cVector3d(0, 0, 0), cVector3d(0, 0, 0));

// insert line inside world

world->addChild(velocity);

////////////////////////////////////////////////////////////////////////////////

// HAPTIC DEVICE

////////////////////////////////////////////////////////////////////////////////

// create a haptic device handler

handler = new cHapticDeviceHandler();

// get a handle to the first haptic device

handler->getDevice(hapticDevice, 0);

// open a connection with the haptic device

hapticDevice->open();

// retrieve information about the current haptic device

cHapticDeviceInfo info = hapticDevice->getSpecifications();

// if the device has a gripper, enable the gripper to behave like a user switch

hapticDevice->setEnableGripperUserSwitch(true);

////////////////////////////////////////////////////////////////////////////////

// WIDGETS

////////////////////////////////////////////////////////////////////////////////

// create a font

font = NEW\_CFONTCALIBRI20();

// create a label to display the haptic device model

labelHapticDeviceModel = new cLabel(font);

camera->m\_frontLayer->addChild(labelHapticDeviceModel);

labelHapticDeviceModel->setText(info.m\_modelName);

labelHapticDeviceModel->m\_fontColor.setBlueRoyal();

// create a label to display the position of haptic device

labelHapticDevicePosition = new cLabel(font);

camera->m\_frontLayer->addChild(labelHapticDevicePosition);

// create a label to display the haptic and graphic rate of the simulation

labelRates = new cLabel(font);

camera->m\_frontLayer->addChild(labelRates);

// create a scope to plot haptic device position data

scope = new cScope();

camera->m\_frontLayer->addChild(scope);

scope->setLocalPos(100, 60);

scope->setRange(-0.1, 0.1);

scope->setSignalEnabled(true, true, true, true);

scope->setTransparencyLevel(0.6);

scope1 = new cScope();

camera->m\_frontLayer->addChild(scope1);

scope1->setLocalPos(100, 60);

scope1->setRange(0, 10); // try static scale within range

scope1->setSignalEnabled(true, true, true, true);

scope1->setTransparencyLevel(0.3);

//

//arr[100] = delay; // Error using it

labelMin = new cLabel(font);

camera->m\_frontLayer->addChild(labelMin);

labelMin->setText("0");

labelMin->setLocalPos(85, 55);

labelMax = new cLabel(font);

camera->m\_frontLayer->addChild(labelMax);

labelMax->setText("0.1");

labelMax->setLocalPos(70, 300);

labelMin = new cLabel(font);

camera->m\_frontLayer->addChild(labelMin);

labelMin->setText("-0.1");

labelMin->setLocalPos(65, 120);

//labelMax = new cLabel(font);

//camera->m\_frontLayer->addChild(labelMax);

//labelMax->setText("Max");

//labelMax->setLocalPos(65, 100); // x = 50 for range 1000 or 4 digits, x = 75 for range of 2 digits

labelMax = new cLabel(font);

camera->m\_frontLayer->addChild(labelMax);

labelMax->setText("Delay Scope (ms)");

labelMax->setLocalPos(385, 100);

labelMax = new cLabel(font);

camera->m\_frontLayer->addChild(labelMax);

labelMax->setText("Position scope");

labelMax->setLocalPos(380, 300);

labelXd\_Prev = new cLabel(font);

labelXd = new cLabel(font);

labelX\_Recv = new cLabel(font);

labelFd = new cLabel(font);

labeldFe = new cLabel(font);

labelIP = new cLabel(font);

labelDelay = new cLabel(font);

labelDashes\_1 = new cLabel(font);

labelServerLabel = new cLabel(font);

labelDashes\_2 = new cLabel(font);

labelKeyboardOptions = new cLabel(font);

labelPotentialField = new cLabel(font);

labelExitApplication = new cLabel(font);

labelScopeInfo = new cLabel(font);

labelmax = new cLabel(font);

labelMirror = new cLabel(font);

labelToggleScreen = new cLabel(font);

labelavg = new cLabel(font);

camera->m\_frontLayer->addChild(labelXd\_Prev);

camera->m\_frontLayer->addChild(labelXd);

camera->m\_frontLayer->addChild(labelX\_Recv);

camera->m\_frontLayer->addChild(labelFd);

camera->m\_frontLayer->addChild(labeldFe);

camera->m\_frontLayer->addChild(labelavg); // Add on

camera->m\_frontLayer->addChild(labelIP);

camera->m\_frontLayer->addChild(labelDashes\_1);

camera->m\_frontLayer->addChild(labelServerLabel);

camera->m\_frontLayer->addChild(labelDashes\_2);

camera->m\_frontLayer->addChild(labelKeyboardOptions);

camera->m\_frontLayer->addChild(labelPotentialField);

camera->m\_frontLayer->addChild(labelExitApplication);

camera->m\_frontLayer->addChild(labelScopeInfo);

camera->m\_frontLayer->addChild(labelMirror);

camera->m\_frontLayer->addChild(labelToggleScreen);

camera->m\_frontLayer->addChild(labelDelay);

camera->m\_frontLayer->addChild(labelmax);

//labelDashes\_1->setText(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

//labelDashes\_1->setLocalPos(160, height - 60);

//labelDashes\_1->m\_fontColor.setBlueLight();

labelServerLabel->setText("Dr.Riaz Uddin Telepresence (Client)");

labelServerLabel->setLocalPos(350, height - 90);

labelServerLabel->m\_fontColor.setBlueLight();

//labelDashes\_2->setText(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

//labelDashes\_2->setLocalPos(160, height - 100);

//labelDashes\_2->m\_fontColor.setBlueLight();

labelKeyboardOptions->setText("Keyboard Options : ");

labelKeyboardOptions->setLocalPos(500, height - 120);

labelKeyboardOptions->m\_fontColor.setWhite();

labelMirror->setText("Press 'm' for getting mirrored screen ");

labelMirror->setLocalPos(500, height - 140);

labelMirror->m\_fontColor.setRed();

labeldFe->setText("dFe");

labeldFe->setLocalPos(300, height - 200);

labeldFe->m\_fontColor.setPink();

labelFd->setText("Fd: ");

labelFd->setLocalPos(300, height - 180);

labelFd->m\_fontColor.setRed();

labelPotentialField->setText("Press '1' to Enable/Disable Potential Field");

labelPotentialField->setLocalPos(500, height - 160);

labelPotentialField->m\_fontColor.setBlue();

labelExitApplication->setText("Press '+' to increase offset Delay");

labelExitApplication->setLocalPos(500, height - 200);

labelExitApplication->m\_fontColor.setBlue();

labelScopeInfo->setText("Scope Information: ");

labelScopeInfo->setLocalPos(20, height - 120);

labelScopeInfo->m\_fontColor.setWhite();

labelIP->setText("Server IP: ");

labelIP->setLocalPos(20, height - 140);

labelIP->m\_fontColor.setRed();

//arr[100] = delay; // error using

labelavg->setText("Average Delay: ");

labelavg->setLocalPos(110, height - 500);

labelavg->m\_fontColor.setWhite();

labelXd\_Prev->setText("Prev Xd: ");

labelXd\_Prev->setLocalPos(20, height - 160);

labelXd\_Prev->m\_fontColor.setBlue();

labelXd->setText("Curr Xd: ");

labelXd->setLocalPos(20, height - 180);

labelXd->m\_fontColor.setRed();

labelX\_Recv->setText("Recv: ");

labelX\_Recv->setLocalPos(20, height - 200);

labelX\_Recv->m\_fontColor.setBlue();

labelRates->setText("Frequency: ");

labelRates->setLocalPos(20, height - 240);

labelRates->m\_fontColor.setWhite();

labelDelay->setText("Delay: ");

labelDelay->setLocalPos(680, height - 500);

labelDelay->m\_fontColor.setWhite();

labelmax->setText("Max: ");

labelmax->setLocalPos(10, 100);

labelmax->m\_fontColor.setWhite();

////////////////////////////////////////////////////////////////////////////////

// START SIMULATION

////////////////////////////////////////////////////////////////////////////////

// create a thread which starts the main haptics rendering loop

hapticsThread = new cThread();

hapticsThread->start(updateHaptics, CTHREAD\_PRIORITY\_HAPTICS);

// setup callback when application exits

atexit(close);

////////////////////////////////////////////////////////////////////////////////

// MAIN GRAPHIC LOOP

////////////////////////////////////////////////////////////////////////////////

// call window size callback at initialization

windowSizeCallback(window, width, height);

while (!glfwWindowShouldClose(window))

{

// get width and height of window

glfwGetWindowSize(window, &width, &height);

// render graphics

updateGraphics();

// swap buffers

glfwSwapBuffers(window);

// process events

glfwPollEvents();

// signal frequency counter

freqCounterGraphics.signal(1);

}

// close window

glfwDestroyWindow(window);

// terminate GLFW library

glfwTerminate();

// exit

return (0);

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void updateGraphics(void)

{

/////////////////////////////////////////////////////////////////////

// UPDATE WIDGETS

/////////////////////////////////////////////////////////////////////

// update position data

labelHapticDevicePosition->setText(hapticDevicePosition.str(3));

// update haptic and graphic rate data

labelRates->setText(cStr(freqCounterGraphics.getFrequency(), 0) + " Hz / " +

cStr(freqCounterHaptics.getFrequency(), 0) + " Hz");

// update position of label

labelRates->setLocalPos((int)(0.5 \* (width - labelRates->getWidth())), 15);

delayarray.push(delayn);

range = max(0, delayn);

range = range + 2;

/\*while (delayn > 0)

{

arr2 = delayn;

(150);

}\*/

//while (1 > 0)

//{

// arr2 = delayn;

// delay;

// return;

//}

if (delayn <= 10)

{

delaym = delayn;

}

else if (delayn <= 20)

{

delaym = delayn / 2;

}

else if (delayn <= 50)

{

delaym = delayn / 5;

}

else if (delayn <= 100)

{

delaym = delayn / 10;

}

else if (delayn <= 200)

{

delaym = delayn / 20;

}

else if (delayn <= 400)

{

delaym = delayn / 40;

}

else if (delayn <= 600)

{

delaym = delayn / 60;

}

else if (delayn <= 800)

{

delaym = delayn / 80;

}

else if (delayn <= 1000)

{

delaym = delayn / 100;

}

else if (delayn <= 1500)

{

delaym = delayn / 150;

}

else if (delayn <= 2000)

{

delaym = delayn / 200;

}

else if (delayn <= 2500)

{

delaym = delayn / 250;

}

// number of values

//for (int i = 0; i < n; i++) {

// arr2 = arr2 + p[i];

//}

// update information to scope

scope->setSignalValues(hapticDevicePosition.x(), hapticDevicePosition.y(), hapticDevicePosition.z());

scope1->setSignalValues(delaym);

labelIP->setText("Server IP: " + string(inet\_ntoa(serveraddr.sin\_addr)) + ":" + cStr(ntohs(serveraddr.sin\_port)));

labelXd\_Prev->setText("Prev Xd: " + cStr(pos\_prev[0], 3U) + " " + cStr(pos\_prev[1], 3U) + " " + cStr(pos\_prev[2], 3U));

labelXd->setText("Xd: " + cStr(pos\_curr[0], 3U) + " " + cStr(pos\_curr[1], 3U) + " " + cStr(pos\_curr[2], 3U));

labelX\_Recv->setText("Recv X: " + cStr(pos\_recv[0], 3U) + " " + cStr(pos\_recv[1], 3U) + " " + cStr(pos\_recv[2], 3U));

labelFd->setText("Fd: " + cStr(Fd[0], 3U) + " " + cStr(Fd[1], 3U) + " " + cStr(Fd[2], 3U));

labelDelay->setText("Delay:" + cStr(delayn, 3U) + "ms");

labelavg->setText("Average Delay:" + cStr(average, 3U) + "ms");

labelmax->setText("Max :" + cStr(range, 3U) + "ms");

labeldFe->setText("dFe: " + cStr(dFe[0], 3U) + " " + cStr(dFe[1], 3U) + " " + cStr(dFe[2], 3U));

/////////////////////////////////////////////////////////////////////

// RENDER SCENE

/////////////////////////////////////////////////////////////////////

// update shadow maps (if any)

world->updateShadowMaps(false, mirroredDisplay);

// render world

camera->renderView(width, height);

// wait until all OpenGL commands are completed

glFinish();

// check for any OpenGL errors

GLenum err;

err = glGetError();

if (err != GL\_NO\_ERROR) cout << "Error: %s\n" << gluErrorString(err);

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void updateHaptics(void)

{

// simulation in now running

simulationRunning = true;

simulationFinished = false;

// main haptic simulation loop

//

//

//ofstream myFile;

//myFile.open("Felcon.csv");

//myFile.open("Delay.csv");

while (simulationRunning)

{

/////////////////////////////////////////////////////////////////////

// READ HAPTIC DEVICE

/////////////////////////////////////////////////////////////////////

n = n++;

// read position

cVector3d position;

hapticDevice->getPosition(position);

// read linear velocity

cVector3d linearVelocity;

hapticDevice->getLinearVelocity(linearVelocity);

//cout << "Linear Velocity" << "(" << linearVelocity << ")" << endl;

//cout << endl;

//cout << "Prev position:" << "(" << pos\_prev[0] << "," << pos\_prev[1] << "," << pos\_prev[2] << ")" << endl;

//cout << "Curr position:" << "(" << pos\_curr[0] << "," << pos\_curr[1] << "," << pos\_curr[2] << ")" << endl;

//cout << "Recv position:" << "(" << pos\_recv[0] << "," << pos\_recv[1] << "," << pos\_recv[2] << ")" << endl;

//myFile << pos\_prev[0] << "," << pos\_prev[1] << "," << pos\_prev[2] << "," << pos\_curr[0] << "," << pos\_curr[1] << "," << pos\_curr[2] << "," << pos\_recv[0] << "," << pos\_recv[1] << "," << pos\_recv[2] << endl;

//myFile << delayn << endl;

if (n >= 0 & n < 500) // if we add more iterations it will change dynamically

{

a = a++;

p[a] = delayn;

sum = sum + p[a];

average = sum / a;

}

else if (n >= 1000 & n < 1500)

{

b = b++;

p[b] = delayn;

k = k + p[b];

average = k / b;

}

else if (n >= 2000 & n < 2500)

{

c = c++;

p[c] = delayn;

l = l + p[c];

average = l / c;

}

else if (n >= 3000 & n < 3500)

{

d = d++;

p[d] = delayn;

m = m + p[d];

average = m / d;

}

else if (n >= 4000 & n < 4500)

{

e = e++;

p[e] = delayn;

o = o + p[e];

average = o / e;

}

else if (n >= 4000 & n < 4500)

{

f = f++;

p[f] = delayn;

q = q + p[f];

average = sum / f;

}

else if (n >= 5000 & n < 5500)

{

g = g++;

p[g] = delayn;

r = r + p[g];

average = r / g;

}

else if (n >= 6000 & n < 6500)

{

h = h++;

p[h] = delayn;

s = s + p[h];

average = s / h;

}

else if (n >= 7000 & n < 7500)

{

j = j++;

p[j] = delayn;

t = t + p[j];

average = t / j;

}

// read user-switch status (button 0)

bool button0, button1, button2, button3;

button0 = false;

button1 = false;

button2 = false;

button3 = false;

hapticDevice->getUserSwitch(0, button0);

hapticDevice->getUserSwitch(1, button1);

hapticDevice->getUserSwitch(2, button2);

hapticDevice->getUserSwitch(3, button3);

/////////////////////////////////////////////////////////////////////

// UPDATE 3D CURSOR MODEL

/////////////////////////////////////////////////////////////////////

// update arrow

velocity->m\_pointA = position;

velocity->m\_pointB = cAdd(position, linearVelocity);

// update position and orientation of cursor

cursor->setLocalPos(position);

// adjust the color of the cursor according to the status of

// the user-switch (ON = TRUE / OFF = FALSE)

if (button0)

{

cursor->m\_material->setGreenMediumAquamarine();

}

else if (button1)

{

cursor->m\_material->setYellowGold();

}

else if (button2)

{

cursor->m\_material->setOrangeCoral();

}

else if (button3)

{

cursor->m\_material->setPurpleLavender();

}

else

{

cursor->m\_material->setBlueRoyal();

}

// update global variable for graphic display update

hapticDevicePosition = position;

hapticDeviceVelocity = linearVelocity;

/////////////////////////////////////////////////////////////////////

// COMPUTE AND APPLY FORCES

/////////////////////////////////////////////////////////////////////

for (int i = 0; i < DOF; i++)

{

pos\_curr[i] = hapticDevicePosition(i);

}

SendData[0] = Fd[0];

SendData[1] = Fd[1];

SendData[2] = Fd[2];

UDPSendData(SendData);

// µ¥ÀÌÅÍ ¹Þ±â

UDPReadData();

pos\_recv[0] = ReadData[0];

pos\_recv[1] = ReadData[1];

pos\_recv[2] = ReadData[2];

cVector3d force(0, 0, 0);

double gripperForce = 0.0;

for (int i = 0; i < 3; i++)

{

XdP[i] = Xd[i];

//cout << "XdP is " << XdP[i] << endl;

Xd[i] = pos\_curr[i];

//cout << "Xd " << Xd[i] << endl;

//DXd[2] = Xd[2] - XdP[2]+1;

DXd[i] = Xd[i] - XdP[i];

// cout << "DXd is " << DXd[i] << endl;

XsdP[i] = Xsd[i];

//cout << "XdP is " << XdP[i] << endl;

Xsd[i] = pos\_recv[i];

//cout << "Xd " << Xd[i] << endl;

//DXd[2] = Xd[2] - XdP[2]+1;

DXsd[i] = Xsd[i] - XsdP[i];

// cout << "DXd is " << DXd[i] << endl;

}

// apply force field

if (useForceField)

{

// compute linear force

double Kp = 200; //25; // [N/m]

cVector3d forceField = -Kp \* position;

force.add(forceField);

for (int i = 0; i < 3; i++)

{

Fd[i] = -Kp \* (pos\_recv[i] - pos\_curr[i]);

//force(i) = Fd[i];

pos\_prev[i] = pos\_curr[i];

if (EnableEBA)

{

c1 = 50;//c1 = 50;

c2 = 200;//c2 = 100;

if (c2 > c1)

c2 = c1;

//Vd[i] = DXd[i] / Tc;

//XeP[i] = Xe[i];

//cout << "Normal DXd is " << DXd[i] << endl;

//if ((DXd[i] - DXsd[i]) > 0.001 || (DXd[i] - DXsd[i]) < -0.001)

if ((DXd[i] - DXsd[i])!=0.0)

{

//cout << "XdP is" << XdP << endl;

//cout << "EBA DXd is " << DXd[i] << endl;

//cout << "DXd222 " << endl;// Control law

// Calculation of Beta

uBETA[i] = BETA[i] = (Fd[i] - dFe[i]) / (DXd[i] - DXsd[i]);

// Calculation of Gamma\_max & Gamma\_min

Bmax[i] = c2 - dFe[i] / (DXd[i] - DXsd[i]) + sqrt(dFe[i] \* dFe[i] / (DXd[i] - DXsd[i]) / (DXd[i] - DXsd[i]) + c2 \* c2); //Gamma\_max

Bmin[i] = c2 - dFe[i] / (DXd[i] - DXsd[i]) - sqrt(dFe[i] \* dFe[i] / (DXd[i] - DXsd[i]) / (DXd[i] - DXsd[i]) + c2 \* c2); //Gamma\_min

if (Bmax[i] > c1)

Bmax[i] = c1; // min(c1, Gamma\_max)

// Bounding Law

if (BETA[i] < Bmin[i])

BETA[i] = Bmin[i];

if (BETA[i] > Bmax[i])

BETA[i] = Bmax[i];

}

else

{

//cout << "beta\_else is running" << endl;

uBETA[i] = BETA[i];

BETA[i] = BETA[i];

}

dFe[i] = dFe[i] + BETA[i] \* (DXd[i] - DXsd[i]);

//cout << "beta" << BETA[i] << endl;

force(i) = -dFe[i];

//pos\_prev[i] = pos\_curr[i];

//cout << countd << i << " Fd " << Fd[i] << endl;

//cout << countd << i << " force " << force(i) << endl;

//cout << countd << i << " dFe " << dFe[i] << endl;

countd++;

}

}

}

else

{

for (int i = 0; i < 3; i++)

{

//cout << "else is running" << endl;

dFe[i] = Fd[i];

force(i) = dFe[i];

}

}

// send computed force

hapticDevice->setForce(force);

// update frequency counter

freqCounterHaptics.signal(1);

}

// exit haptics thread

simulationFinished = true;

} // KP

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void windowSizeCallback(GLFWwindow\* a\_window, int a\_width, int a\_height)

{

// update window size

width = a\_width;

height = a\_height;

// update position of label

labelHapticDevicePosition->setLocalPos(20, width - 60, 0);

// update position of label

labelHapticDeviceModel->setLocalPos(20, height - 40, 0);

// update position of scope

scope->setSize(width - 200, 260);

scope1->setSize(width - 200, 60);

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void errorCallback(int a\_error, const char\* a\_description)

{

cout << "Error: " << a\_description << endl;

}

//////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void keyCallback(GLFWwindow\* a\_window, int a\_key, int a\_scancode, int a\_action, int a\_mods)

{

// filter calls that only include a key press

if ((a\_action != GLFW\_PRESS) && (a\_action != GLFW\_REPEAT))

{

return;

}

// option - exit

if ((a\_key == GLFW\_KEY\_ESCAPE) || (a\_key == GLFW\_KEY\_Q))

{

glfwSetWindowShouldClose(a\_window, GLFW\_TRUE);

}

// option - enable/disable force field

if (a\_key == GLFW\_KEY\_1)

{

useForceField = !useForceField;

if (useForceField)

cout << "> Enable force field \r";

else

cout << "> Disable force field \r";

}

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void close(void)

{

CloseUDP();

// stop the simulation

simulationRunning = false;

// wait for graphics and haptics loops to terminate

while (!simulationFinished)

{

cSleepMs(100);

}

// close haptic device

hapticDevice->close();

// delete resources

delete hapticsThread;

delete world;

delete handler;

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void InitUDP()

{

// À©¼Ó ÃÊ±âÈ

if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)

{

MessageBox(NULL, "Socket Open Error!!", "UDP", MB\_OK);

}

// socket()

sock = socket(AF\_INET, SOCK\_DGRAM, 0);

if (sock == INVALID\_SOCKET)

{

err\_quit("socket()");

}

// bind()

ZeroMemory(&serveraddr, sizeof(serveraddr));

serveraddr.sin\_family = AF\_INET;

serveraddr.sin\_port = htons(UDP\_PORT);

serveraddr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void CloseUDP()

{

// closesocket()

closesocket(sock);

// À©¼Ó Á¾·á

WSACleanup();

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void UDPSendData(float sData[SENDDATA\_NUM])

{

Timer time;

// sprintf(sbuf, "%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f", sData[0], sData[1], sData[2], sData[3], sData[4], sData[5], sData[6], sData[7], sData[8], sData[9], sData[10], sData[11], sData[12], sData[13], sData[14], sData[15], sData[16], sData[17], sData[18], sData[19]);

sprintf(sbuf, "%f %f %f", sData[0], sData[1], sData[2]);

retval = sendto(sock, sbuf, strlen(sbuf), 0, (SOCKADDR\*)&serveraddr, sizeof(serveraddr));

if (retval == SOCKET\_ERROR)

{

err\_display("sendto()");

exit(-1);

}

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void UDPReadData()

{

Timer2 time;

addrlen = sizeof(clientaddr);

retval = recvfrom(sock, rbuf, BUFSIZE, 0, (SOCKADDR\*)&clientaddr, &addrlen);

if (retval == SOCKET\_ERROR)

{

err\_display("recvfrom()");

exit(-1);

}

delayn = timeUDPS - timeUDPR;

countd = countd++;

//cout << countd << " Delay calculated " << " " << delayn << " " << " milliseconds" << endl;

// sscanf(rbuf, "%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f %f", &ReadData[0], &ReadData[1], &ReadData[2], &ReadData[3], &ReadData[4], &ReadData[5], &ReadData[6], &ReadData[7], &ReadData[8], &ReadData[9], &ReadData[10], &ReadData[11], &ReadData[12], &ReadData[13], &ReadData[14], &ReadData[15], &ReadData[16], &ReadData[17], &ReadData[18], &ReadData[19]);

sscanf(rbuf, "%f %f %f", &ReadData[0], &ReadData[1], &ReadData[2]);

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// ¼ÒÄÏ ÇÔ¼ö ¿À·ù Ãâ·Â ÈÄ Á¾·á

void err\_quit(const char\* msg)

{

LPVOID lpMsgBuf;

FormatMessage(

FORMAT\_MESSAGE\_ALLOCATE\_BUFFER |

FORMAT\_MESSAGE\_FROM\_SYSTEM,

NULL, WSAGetLastError(),

MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT),

(LPTSTR)&lpMsgBuf, 0, NULL);

MessageBox(NULL, (LPCTSTR)lpMsgBuf, msg, MB\_ICONERROR);

LocalFree(lpMsgBuf);

exit(-1);

}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// ¼ÒÄÏ ÇÔ¼ö ¿À·ù Ãâ·Â

void err\_display(const char\* msg)

{

LPVOID lpMsgBuf;

FormatMessage(

FORMAT\_MESSAGE\_ALLOCATE\_BUFFER |

FORMAT\_MESSAGE\_FROM\_SYSTEM,

NULL, WSAGetLastError(),

MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT),

(LPTSTR)&lpMsgBuf, 0, NULL);

printf("[%s] %s", msg, (LPCTSTR)lpMsgBuf);

LocalFree(lpMsgBuf);

}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////