```
/
*****************************************************************
**************
*
*
*  OpenNI 1.x Alpha
*
*  Copyright (C) 2011 PrimeSense Ltd.
*
*
*
*  This file is part of OpenNI.
*
*
*
*  OpenNI is free software: you can redistribute it and/or
modify           *
*  it under the terms of the GNU Lesser General Public License
as published *
*  by the Free Software Foundation, either version 3 of the
License, or     *
*  (at your option) any later version.
*
*
*
*  OpenNI is distributed in the hope that it will be useful,
*
*  but WITHOUT ANY WARRANTY; without even the implied warranty
of           *
*  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
the             *
*  GNU Lesser General Public License for more details.
*
*
*
*  You should have received a copy of the GNU Lesser General
Public License *
*  along with OpenNI. If not, see
<http://www.gnu.org/licenses/>.           *
*
*
*****************************************************************
**************/
//---------------------------------------------------------------
---------------
// Includes
//---------------------------------------------------------------
---------------
#include <XnOpenNI.h>
#include <XnCodecIDs.h>
#include <XnCppWrapper.h>
#include "SceneDrawer.h"
#include <XnPropNames.h>

//---------------------------------------------------------------
---------------
// Globals
```

```
//---------------------------------------------------------------
---------------
xn::Context g_Context;
xn::ScriptNode g_scriptNode;
xn::DepthGenerator g_DepthGenerator;
xn::UserGenerator g_UserGenerator;
xn::Player g_Player;

XnBool g_bNeedPose = FALSE;
XnChar g_strPose[20] = "";
XnBool g_bDrawBackground = TRUE;
XnBool g_bDrawPixels = TRUE;
XnBool g_bDrawSkeleton = TRUE;
XnBool g_bPrintID = TRUE;
XnBool g_bPrintState = TRUE;
// B1_UD5 CODE ------------------------------------
XnBool g_bPrintHandLabels = TRUE;
// END B1_UD5 CODE --------------------------------

#ifndef USE_GLES
#if (XN_PLATFORM == XN_PLATFORM_MACOSX)
      #include <GLUT/glut.h>
#else
      #include <GL/glut.h>
#endif
#else
      #include "opengles.h"
#endif

#ifdef USE_GLES
static EGLDisplay display = EGL_NO_DISPLAY;
static EGLSurface surface = EGL_NO_SURFACE;
static EGLContext context = EGL_NO_CONTEXT;
#endif

#define GL_WIN_SIZE_X 720
#define GL_WIN_SIZE_Y 480

XnBool g_bPause = false;
XnBool g_bRecord = false;

XnBool g_bQuit = false;

//---------------------------------------------------------------
---------------
// Code
//---------------------------------------------------------------
---------------

void CleanupExit()
{
      g_scriptNode.Release();
      g_DepthGenerator.Release();
      g_UserGenerator.Release();
      g_Player.Release();
      g_Context.Release();
```

```
	exit (1);
}

// Callback: New user was detected
void XN_CALLBACK_TYPE User_NewUser(xn::UserGenerator&
generator, XnUserID nId, void* pCookie)
{
	XnUInt32 epochTime = 0;
	xnOSGetEpochTime(&epochTime);
	printf("%d New User %d\n", epochTime, nId);
	// New user found
	if (g_bNeedPose)
	{
		g_UserGenerator.GetPoseDetectionCap
().StartPoseDetection(g_strPose, nId);
	}
	else
	{
		g_UserGenerator.GetSkeletonCap().RequestCalibration
(nId, TRUE);
	}
}
// Callback: An existing user was lost
void XN_CALLBACK_TYPE User_LostUser(xn::UserGenerator&
generator, XnUserID nId, void* pCookie)
{
	XnUInt32 epochTime = 0;
	xnOSGetEpochTime(&epochTime);
	printf("%d Lost user %d\n", epochTime, nId);
}
// Callback: Detected a pose
void XN_CALLBACK_TYPE UserPose_PoseDetected
(xn::PoseDetectionCapability& capability, const XnChar*
strPose, XnUserID nId, void* pCookie)
{
	XnUInt32 epochTime = 0;
	xnOSGetEpochTime(&epochTime);
	printf("%d Pose %s detected for user %d\n", epochTime,
strPose, nId);
	g_UserGenerator.GetPoseDetectionCap().StopPoseDetection
(nId);
	g_UserGenerator.GetSkeletonCap().RequestCalibration(nId,
TRUE);
}
// Callback: Started calibration
void XN_CALLBACK_TYPE UserCalibration_CalibrationStart
(xn::SkeletonCapability& capability, XnUserID nId, void*
pCookie)
{
	XnUInt32 epochTime = 0;
	xnOSGetEpochTime(&epochTime);
	printf("%d Calibration started for user %d\n", epochTime,
nId);
}
// Callback: Finished calibration
void XN_CALLBACK_TYPE UserCalibration_CalibrationComplete
(xn::SkeletonCapability& capability, XnUserID nId,
```

```
XnCalibrationStatus eStatus, void* pCookie)
{
      XnUInt32 epochTime = 0;
      xnOSGetEpochTime(&epochTime);
      if (eStatus == XN_CALIBRATION_STATUS_OK)
      {
            // Calibration succeeded
            printf("%d Calibration complete, start tracking
user %d\n", epochTime, nId);
            g_UserGenerator.GetSkeletonCap().StartTracking
(nId);
      }
      else
      {
            // Calibration failed
            printf("%d Calibration failed for user %d\n",
epochTime, nId);
        if(eStatus==XN_CALIBRATION_STATUS_MANUAL_ABORT)
        {
             printf("Manual abort occured, stop attempting to
calibrate!");
             return;
        }
            if (g_bNeedPose)
            {
                  g_UserGenerator.GetPoseDetectionCap
().StartPoseDetection(g_strPose, nId);
            }
            else
            {
                  g_UserGenerator.GetSkeletonCap
().RequestCalibration(nId, TRUE);
            }
      }
}

#define XN_CALIBRATION_FILE_NAME "UserCalibration.bin"

// Save calibration to file
void SaveCalibration()
{
      XnUserID aUserIDs[20] = {0};
      XnUInt16 nUsers = 20;
      g_UserGenerator.GetUsers(aUserIDs, nUsers);
      for (int i = 0; i < nUsers; ++i)
      {
            // Find a user who is already calibrated
            if (g_UserGenerator.GetSkeletonCap().IsCalibrated
(aUserIDs[i]))
            {
                  // Save user's calibration to file
                  g_UserGenerator.GetSkeletonCap
().SaveCalibrationDataToFile(aUserIDs[i],
XN_CALIBRATION_FILE_NAME);
                  break;
            }
      }
```

```
}
// Load calibration from file
void LoadCalibration()
{
	XnUserID aUserIDs[20] = {0};
	XnUInt16 nUsers = 20;
	g_UserGenerator.GetUsers(aUserIDs, nUsers);
	for (int i = 0; i < nUsers; ++i)
	{
		// Find a user who isn't calibrated or currently in
pose
		if (g_UserGenerator.GetSkeletonCap().IsCalibrated
(aUserIDs[i])) continue;
		if (g_UserGenerator.GetSkeletonCap().IsCalibrating
(aUserIDs[i])) continue;

		// Load user's calibration from file
		XnStatus rc = g_UserGenerator.GetSkeletonCap
().LoadCalibrationDataFromFile(aUserIDs[i],
XN_CALIBRATION_FILE_NAME);
		if (rc == XN_STATUS_OK)
		{
			// Make sure state is coherent
			g_UserGenerator.GetPoseDetectionCap
().StopPoseDetection(aUserIDs[i]);
			g_UserGenerator.GetSkeletonCap().StartTracking
(aUserIDs[i]);
		}
		break;
	}
}

// this function is called each frame
void glutDisplay (void)
{

	glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	// Setup the OpenGL viewpoint
	glMatrixMode(GL_PROJECTION);
	glPushMatrix();
	glLoadIdentity();

	xn::SceneMetaData sceneMD;
	xn::DepthMetaData depthMD;
	g_DepthGenerator.GetMetaData(depthMD);
#ifndef USE_GLES
	glOrtho(0, depthMD.XRes(), depthMD.YRes(), 0, -1.0, 1.0);
#else
	glOrthof(0, depthMD.XRes(), depthMD.YRes(), 0, -1.0,
1.0);
#endif

	glDisable(GL_TEXTURE_2D);

	if (!g_bPause)
	{
```

```
            // Read next available data
            g_Context.WaitOneUpdateAll(g_UserGenerator);
      }

            // Process the data
            g_DepthGenerator.GetMetaData(depthMD);
            g_UserGenerator.GetUserPixels(0, sceneMD);
            DrawDepthMap(depthMD, sceneMD);

#ifndef USE_GLES
      glutSwapBuffers();
#endif
}

#ifndef USE_GLES
void glutIdle (void)
{
      if (g_bQuit) {
            CleanupExit();
      }

      // Display the frame
      glutPostRedisplay();
}

void glutKeyboard (unsigned char key, int x, int y)
{
      switch (key)
      {
      case 27:
        CleanupExit();
// B1_UD5 CODE ------------------------------------
         case 'h':
        //Print hand labels?
        g_bPrintHandLabels = !g_bPrintHandLabels;
        break;
// END B1_UD5 CODE --------------------------------
      case 'b':
        // Draw background?
        g_bDrawBackground = !g_bDrawBackground;
        break;
      case 'x':
        // Draw pixels at all?
        g_bDrawPixels = !g_bDrawPixels;
        break;
      case 's':
        // Draw Skeleton?
        g_bDrawSkeleton = !g_bDrawSkeleton;
        break;
      case 'i':
        // Print label?
        g_bPrintID = !g_bPrintID;
        break;
      case 'l':
        // Print ID & state as label, or only ID?
        g_bPrintState = !g_bPrintState;
        break;
```

```
	case'p':
	  g_bPause = !g_bPause;
	  break;
	case 'S':
	  SaveCalibration();
	  break;
	case 'L':
	  LoadCalibration();
	  break;
	}
}
void glInit (int * pargc, char ** argv)
{
	glutInit(pargc, argv);
	glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
	glutInitWindowSize(GL_WIN_SIZE_X, GL_WIN_SIZE_Y);
	glutCreateWindow ("User Tracker Viewer");
	//glutFullScreen();
	glutSetCursor(GLUT_CURSOR_NONE);

	glutKeyboardFunc(glutKeyboard);
	glutDisplayFunc(glutDisplay);
	glutIdleFunc(glutIdle);

	glDisable(GL_DEPTH_TEST);
	glEnable(GL_TEXTURE_2D);

	glEnableClientState(GL_VERTEX_ARRAY);
	glDisableClientState(GL_COLOR_ARRAY);
}
#endif // USE_GLES

#define SAMPLE_XML_PATH "../../Config/SamplesConfig.xml"

#define CHECK_RC(nRetVal, what)
                        \
	if (nRetVal != XN_STATUS_OK)
                        \
	{
                                    \
		printf("%s failed: %s\n", what, xnGetStatusString
(nRetVal));\
		return nRetVal;
                                \
	}

int main(int argc, char **argv)
{
	XnStatus nRetVal = XN_STATUS_OK;

	if (argc > 1)
	{
		nRetVal = g_Context.Init();
		CHECK_RC(nRetVal, "Init");
		nRetVal = g_Context.OpenFileRecording(argv[1],
g_Player);
		if (nRetVal != XN_STATUS_OK)
```

```
            {
                  printf("Can't open recording %s: %s\n", argv
[1], xnGetStatusString(nRetVal));
                  return 1;
            }
      }
      else
      {
            xn::EnumerationErrors errors;
            nRetVal = g_Context.InitFromXmlFile
(SAMPLE_XML_PATH, g_scriptNode, &errors);
            if (nRetVal == XN_STATUS_NO_NODE_PRESENT)
            {
                  XnChar strError[1024];
                  errors.ToString(strError, 1024);
                  printf("%s\n", strError);
                  return (nRetVal);
            }
            else if (nRetVal != XN_STATUS_OK)
            {
                  printf("Open failed: %s\n", xnGetStatusString
(nRetVal));
                  return (nRetVal);
            }
      }

      nRetVal = g_Context.FindExistingNode(XN_NODE_TYPE_DEPTH,
g_DepthGenerator);
      if (nRetVal != XN_STATUS_OK)
      {
            printf("No depth generator found. Using a default
one...");
            xn::MockDepthGenerator mockDepth;
            nRetVal = mockDepth.Create(g_Context);
            CHECK_RC(nRetVal, "Create mock depth");

            // set some defaults
            XnMapOutputMode defaultMode;
            defaultMode.nXRes = 320;
            defaultMode.nYRes = 240;
            defaultMode.nFPS = 30;
            nRetVal = mockDepth.SetMapOutputMode(defaultMode);
            CHECK_RC(nRetVal, "set default mode");

            // set FOV
            XnFieldOfView fov;
            fov.fHFOV = 1.0225999419141749;
            fov.fVFOV = 0.79661567681716894;
            nRetVal = mockDepth.SetGeneralProperty
(XN_PROP_FIELD_OF_VIEW, sizeof(fov), &fov);
            CHECK_RC(nRetVal, "set FOV");

            XnUInt32 nDataSize = defaultMode.nXRes *
defaultMode.nYRes * sizeof(XnDepthPixel);
            XnDepthPixel* pData = (XnDepthPixel*)
xnOSCallocAligned(nDataSize, 1, XN_DEFAULT_MEM_ALIGN);
```

```
            nRetVal = mockDepth.SetData(1, 0, nDataSize,
pData);
            CHECK_RC(nRetVal, "set empty depth map");

            g_DepthGenerator = mockDepth;
      }

      nRetVal = g_Context.FindExistingNode(XN_NODE_TYPE_USER,
g_UserGenerator);
      if (nRetVal != XN_STATUS_OK)
      {
            nRetVal = g_UserGenerator.Create(g_Context);
            CHECK_RC(nRetVal, "Find user generator");
      }

      XnCallbackHandle hUserCallbacks, hCalibrationStart,
hCalibrationComplete, hPoseDetected, hCalibrationInProgress,
hPoseInProgress;
      if (!g_UserGenerator.IsCapabilitySupported
(XN_CAPABILITY_SKELETON))
      {
            printf("Supplied user generator doesn't support
skeleton\n");
            return 1;
      }
      nRetVal = g_UserGenerator.RegisterUserCallbacks
(User_NewUser, User_LostUser, NULL, hUserCallbacks);
      CHECK_RC(nRetVal, "Register to user callbacks");
      nRetVal = g_UserGenerator.GetSkeletonCap
().RegisterToCalibrationStart
(UserCalibration_CalibrationStart, NULL, hCalibrationStart);
      CHECK_RC(nRetVal, "Register to calibration start");
      nRetVal = g_UserGenerator.GetSkeletonCap
().RegisterToCalibrationComplete
(UserCalibration_CalibrationComplete, NULL,
hCalibrationComplete);
      CHECK_RC(nRetVal, "Register to calibration complete");

      if (g_UserGenerator.GetSkeletonCap
().NeedPoseForCalibration())
      {
            g_bNeedPose = TRUE;
            if (!g_UserGenerator.IsCapabilitySupported
(XN_CAPABILITY_POSE_DETECTION))
            {
                  printf("Pose required, but not supported\n");
                  return 1;
            }
            nRetVal = g_UserGenerator.GetPoseDetectionCap
().RegisterToPoseDetected(UserPose_PoseDetected, NULL,
hPoseDetected);
            CHECK_RC(nRetVal, "Register to Pose Detected");
            g_UserGenerator.GetSkeletonCap().GetCalibrationPose
(g_strPose);
      }

      g_UserGenerator.GetSkeletonCap().SetSkeletonProfile
```

```
(XN_SKEL_PROFILE_ALL);

      nRetVal = g_UserGenerator.GetSkeletonCap
().RegisterToCalibrationInProgress(MyCalibrationInProgress,
NULL, hCalibrationInProgress);
      CHECK_RC(nRetVal, "Register to calibration in progress");

      nRetVal = g_UserGenerator.GetPoseDetectionCap
().RegisterToPoseInProgress(MyPoseInProgress, NULL,
hPoseInProgress);
      CHECK_RC(nRetVal, "Register to pose in progress");

      nRetVal = g_Context.StartGeneratingAll();
      CHECK_RC(nRetVal, "StartGenerating");

#ifndef USE_GLES
      glInit(&argc, argv);
      glutMainLoop();
#else
      if (!opengles_init(GL_WIN_SIZE_X, GL_WIN_SIZE_Y,
&display, &surface, &context))
      {
            printf("Error initializing opengles\n");
            CleanupExit();
      }

      glDisable(GL_DEPTH_TEST);
      glEnable(GL_TEXTURE_2D);
      glEnableClientState(GL_VERTEX_ARRAY);
      glDisableClientState(GL_COLOR_ARRAY);

      while (!g_bQuit)
      {
            glutDisplay();
            eglSwapBuffers(display, surface);
      }
      opengles_shutdown(display, surface, context);

      CleanupExit();
#endif
}
```