```
/****************************************************************************
*                                                                           *
*  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/>.           *
*                                                                           *
****************************************************************************/
/****************************************************************************
*  Modified by Juan Pedro Bandera Rubio
****************************************************************************/
//---------------------------------------------------------------------------
// Includes
//---------------------------------------------------------------------------
#include <XnOS.h>
#if (XN_PLATFORM == XN_PLATFORM_MACOSX)
        #include <GLUT/glut.h>
#else
        #include <GL/glut.h>
#endif
#include <math.h>

#include <XnCppWrapper.h>
// B1_UD3 code----------------------------------------------------------------
#include <time.h>
// end B1_UD3 code------------------------------------------------------------
using namespace xn;

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

#define GL_WIN_SIZE_X 1280
#define GL_WIN_SIZE_Y 1024

#define DISPLAY_MODE_OVERLAY    1
#define DISPLAY_MODE_DEPTH              2
#define DISPLAY_MODE_IMAGE              3
#define DEFAULT_DISPLAY_MODE    DISPLAY_MODE_DEPTH

#define MAX_DEPTH 10000
// B1_UD3 code----------------------------------------------------------------
#define GAME_TIME 60.0
// end B1_UD3 code------------------------------------------------------------

//---------------------------------------------------------------------------
// Globals
//---------------------------------------------------------------------------
float g_pDepthHist[MAX_DEPTH];
XnRGB24Pixel* g_pTexMap = NULL;
unsigned int g_nTexMapX = 0;
unsigned int g_nTexMapY = 0;

unsigned int g_nViewState = DEFAULT_DISPLAY_MODE;

Context g_context;
ScriptNode g_scriptNode;
DepthGenerator g_depth;
```

```
ImageGenerator g_image;
DepthMetaData g_depthMD;
ImageMetaData g_imageMD;
// B1_UD3 code---------------------------------------------------------------
XnPoint3D target2D;
int targetHalfSize = 10; // Half the width of the target
int hits = 0;
XnBool targetAvailable = false;
time_t start;
time_t end;
double dif;
// end B1_UD3 code-----------------------------------------------------------

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

void glutIdle (void)
{
	// Display the frame
	glutPostRedisplay();
}

void glutDisplay (void)
{
	XnStatus rc = XN_STATUS_OK;

	// Read a new frame
	rc = g_context.WaitAnyUpdateAll();
	if (rc != XN_STATUS_OK)
	{
		printf("Read failed: %s\n", xnGetStatusString(rc));
		return;
	}

	g_depth.GetMetaData(g_depthMD);
	g_image.GetMetaData(g_imageMD);

	const XnDepthPixel* pDepth = g_depthMD.Data();
	const XnUInt8* pImage = g_imageMD.Data();

	unsigned int nImageScale = GL_WIN_SIZE_X / g_depthMD.FullXRes();

	// Copied from SimpleViewer
	// Clear the OpenGL buffers
	glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

	// Setup the OpenGL viewpoint
	glMatrixMode(GL_PROJECTION);
	glPushMatrix();
	glLoadIdentity();
	glOrtho(0, GL_WIN_SIZE_X, GL_WIN_SIZE_Y, 0, -1.0, 1.0);

	// Calculate the accumulative histogram (the yellow display...)
	xnOSMemSet(g_pDepthHist, 0, MAX_DEPTH*sizeof(float));

	unsigned int nNumberOfPoints = 0;
	for (XnUInt y = 0; y < g_depthMD.YRes(); ++y)
	{
		for (XnUInt x = 0; x < g_depthMD.XRes(); ++x, ++pDepth)
		{
			if (*pDepth != 0)
			{
				g_pDepthHist[*pDepth]++;
				nNumberOfPoints++;
			}
		}
	}
	for (int nIndex=1; nIndex<MAX_DEPTH; nIndex++)
	{
		g_pDepthHist[nIndex] += g_pDepthHist[nIndex-1];
	}
```

```
        if (nNumberOfPoints)
        {
                for (int nIndex=1; nIndex<MAX_DEPTH; nIndex++)
                {
                        g_pDepthHist[nIndex] = (unsigned int)(256 * (1.0f - (g_pDepthHist[nIndex] /
nNumberOfPoints)));
                }
        }

        xnOSMemSet(g_pTexMap, 0, g_nTexMapX*g_nTexMapY*sizeof(XnRGB24Pixel));

        // check if we need to draw image frame to texture
        if (g_nViewState == DISPLAY_MODE_OVERLAY ||
                g_nViewState == DISPLAY_MODE_IMAGE)
        {
                const XnRGB24Pixel* pImageRow = g_imageMD.RGB24Data();
                XnRGB24Pixel* pTexRow = g_pTexMap + g_imageMD.YOffset() * g_nTexMapX;

                for (XnUInt y = 0; y < g_imageMD.YRes(); ++y)
                {
                        const XnRGB24Pixel* pImage = pImageRow;
                        XnRGB24Pixel* pTex = pTexRow + g_imageMD.XOffset();

                        for (XnUInt x = 0; x < g_imageMD.XRes(); ++x, ++pImage, ++pTex)
                        {
                                *pTex = *pImage;
                        }

                        pImageRow += g_imageMD.XRes();
                        pTexRow += g_nTexMapX;
                }
        }

        // check if we need to draw depth frame to texture
        if (g_nViewState == DISPLAY_MODE_OVERLAY ||
                g_nViewState == DISPLAY_MODE_DEPTH)
        {
                const XnDepthPixel* pDepthRow = g_depthMD.Data();
                XnRGB24Pixel* pTexRow = g_pTexMap + g_depthMD.YOffset() * g_nTexMapX;

                for (XnUInt y = 0; y < g_depthMD.YRes(); ++y)
                {
                        const XnDepthPixel* pDepth = pDepthRow;
                        XnRGB24Pixel* pTex = pTexRow + g_depthMD.XOffset();

                        for (XnUInt x = 0; x < g_depthMD.XRes(); ++x, ++pDepth, ++pTex)
                        {
// B1_UD3 code -------------------------------------
                          // Check distance to print red pixels if distance < 1000 mm.
                          if ( (*pDepth != 0)&&(*pDepth<1000) )
                            {
                              int nHistValue = g_pDepthHist[*pDepth];
                              pTex->nRed = nHistValue;
                              pTex->nGreen = 0;
                              pTex->nBlue = 0;
                              // Check if the target has been hit
                              if ( (targetAvailable)&&(x==target2D.X)&&(y==target2D.Y) )
                                {
                                  targetAvailable = false;
                                  hits++;
                                  printf("Collected! Hits: %d\n",hits);
                                }
                            }
                          else if (*pDepth != 0)
                            {
                              int nHistValue = g_pDepthHist[*pDepth];
                              pTex->nRed = nHistValue;
                              pTex->nGreen = nHistValue;
                              pTex->nBlue = 0;
                            }
                        }
// end B1_UD3 code ---------------------------------
```

```
                        pDepthRow += g_depthMD.XRes();
                        pTexRow += g_nTexMapX;
                }
        }

// B1_UD3 code ----------------------------------
        // if there is no target, set one
        if (!targetAvailable)
          {
            int tempX = g_depthMD.XRes() - (2*targetHalfSize) - 1;
            int tempY = g_depthMD.YRes() - (2*targetHalfSize) - 1;
            target2D.X = (rand())%(tempX) + targetHalfSize;
            target2D.Y = (rand())%(tempY) + targetHalfSize;
            targetAvailable = true;
          }
        // Print the target
        if (g_nViewState == DISPLAY_MODE_OVERLAY ||
                g_nViewState == DISPLAY_MODE_DEPTH)
        {
                const XnDepthPixel* pDepthRow = g_depthMD.Data();
                XnRGB24Pixel* pTexRow = g_pTexMap + g_depthMD.YOffset() * g_nTexMapX;

                for (XnUInt y = 0; y < g_depthMD.YRes(); ++y)
                {
                        const XnDepthPixel* pDepth = pDepthRow;
                        XnRGB24Pixel* pTex = pTexRow + g_depthMD.XOffset();

                        for (XnUInt x = 0; x < g_depthMD.XRes(); ++x, ++pDepth, ++pTex)
                        {
                          if ( (x>target2D.X-10)&&(x<target2D.X+10)&&(y>target2D.Y-10)&&(y<target2D.Y
+10) )
                            {
                              int nHistValue = g_pDepthHist[*pDepth];
                              pTex->nRed = 255;
                              pTex->nGreen = 128;
                              pTex->nBlue = 0;
                            }
                        }

                        pDepthRow += g_depthMD.XRes();
                        pTexRow += g_nTexMapX;
                }
        }
// end B1_UD3 code ------------------------------

        // Create the OpenGL texture map
        glTexParameteri(GL_TEXTURE_2D, GL_GENERATE_MIPMAP_SGIS, GL_TRUE);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, g_nTexMapX, g_nTexMapY, 0, GL_RGB, GL_UNSIGNED_BYTE,
g_pTexMap);

        // Display the OpenGL texture map
        glColor4f(1,1,1,1);

        glBegin(GL_QUADS);

        int nXRes = g_depthMD.FullXRes();
        int nYRes = g_depthMD.FullYRes();

        // upper left
        glTexCoord2f(0, 0);
        glVertex2f(0, 0);
        // upper right
        glTexCoord2f((float)nXRes/(float)g_nTexMapX, 0);
        glVertex2f(GL_WIN_SIZE_X, 0);
        // bottom right
        glTexCoord2f((float)nXRes/(float)g_nTexMapX, (float)nYRes/(float)g_nTexMapY);
        glVertex2f(GL_WIN_SIZE_X, GL_WIN_SIZE_Y);
        // bottom left
        glTexCoord2f(0, (float)nYRes/(float)g_nTexMapY);
        glVertex2f(0, GL_WIN_SIZE_Y);
```

```
        glEnd();

        // Swap the OpenGL display buffers
        glutSwapBuffers();

// B1_UD3 code-----------------------------------------------------------------
        // End capture
        time(&end);
        dif = difftime(end,start);
        if (dif >= GAME_TIME)
          {
            // End game
            printf("GAME OVER. Collected: %d\n",hits);
            exit(0);
          }
// end B1_UD3 code-------------------------------------------------------------

}

void glutKeyboard (unsigned char key, int x, int y)
{
        switch (key)
        {
                case 27:
                        exit (1);
                case '1':
                        g_nViewState = DISPLAY_MODE_OVERLAY;
                        g_depth.GetAlternativeViewPointCap().SetViewPoint(g_image);
                        break;
                case '2':
                        g_nViewState = DISPLAY_MODE_DEPTH;
                        g_depth.GetAlternativeViewPointCap().ResetViewPoint();
                        break;
                case '3':
                        g_nViewState = DISPLAY_MODE_IMAGE;
                        g_depth.GetAlternativeViewPointCap().ResetViewPoint();
                        break;
                case 'm':
                        g_context.SetGlobalMirror(!g_context.GetGlobalMirror());
                        break;
        }
}

int main(int argc, char* argv[])
{
        XnStatus rc;

        EnumerationErrors errors;
        rc = g_context.InitFromXmlFile(SAMPLE_XML_PATH, g_scriptNode, &errors);
        if (rc == XN_STATUS_NO_NODE_PRESENT)
        {
                XnChar strError[1024];
                errors.ToString(strError, 1024);
                printf("%s\n", strError);
                return (rc);
        }
        else if (rc != XN_STATUS_OK)
        {
                printf("Open failed: %s\n", xnGetStatusString(rc));
                return (rc);
        }

        rc = g_context.FindExistingNode(XN_NODE_TYPE_DEPTH, g_depth);
        if (rc != XN_STATUS_OK)
        {
                printf("No depth node exists! Check your XML.");
                return 1;
        }

        rc = g_context.FindExistingNode(XN_NODE_TYPE_IMAGE, g_image);
        if (rc != XN_STATUS_OK)
```

```
        {
                printf("No image node exists! Check your XML.");
                return 1;
        }

        g_depth.GetMetaData(g_depthMD);
        g_image.GetMetaData(g_imageMD);

        // Hybrid mode isn't supported in this sample
        if (g_imageMD.FullXRes() != g_depthMD.FullXRes() || g_imageMD.FullYRes() != g_depthMD.FullYRes
())
        {
                printf ("The device depth and image resolution must be equal!\n");
                return 1;
        }

        // RGB is the only image format supported.
        if (g_imageMD.PixelFormat() != XN_PIXEL_FORMAT_RGB24)
        {
                printf("The device image format must be RGB24\n");
                return 1;
        }

        // Texture map init
        g_nTexMapX = (((unsigned short)(g_depthMD.FullXRes()-1) / 512) + 1) * 512;
        g_nTexMapY = (((unsigned short)(g_depthMD.FullYRes()-1) / 512) + 1) * 512;
        g_pTexMap = (XnRGB24Pixel*)malloc(g_nTexMapX * g_nTexMapY * sizeof(XnRGB24Pixel));

        // OpenGL init
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
        glutInitWindowSize(GL_WIN_SIZE_X, GL_WIN_SIZE_Y);
        glutCreateWindow ("OpenNI Simple Viewer");
        glutFullScreen();
        glutSetCursor(GLUT_CURSOR_NONE);

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

        glDisable(GL_DEPTH_TEST);
        glEnable(GL_TEXTURE_2D);

// B1_UD3 code-----------------------------------------------------------------
        // Start capture
        time(&start);
// end B1_UD3 code-------------------------------------------------------------

        // Per frame code is in glutDisplay
        glutMainLoop();

        return 0;
}
```