# 使用OpenNI进行骨骼跟踪

## 骨骼跟踪程序流程

使用OpenNI对骨骼进行跟踪，流程如下：

1. 定义并初始化有关对象，包括Context对象、深度生成器DepthGenerator、图像生成器ImageGenerator、用户生成器UserGenerator及ImageMetaData数据对象等
2. 定义并设置输出模式XnMapOutputMode
3. 定义并注册有关回调函数，包括以下几个回调函数：

**发现新用户和失去用户的回调函数：**

void XN\_CALLBACK\_TYPE NewUser( xn::UserGenerator& generator, XnUserID user,void\* pCookie )

void XN\_CALLBACK\_TYPE LostUser( xn::UserGenerator& generator, XnUserID user,void\* pCookie )

注册函数，调用UserGenerator.RegisterUserCallbacksb函数进行注册。

**校正开始和校正结束的回调函数：**

void XN\_CALLBACK\_TYPE CalibrationStart( xn::SkeletonCapability& skeleton,XnUserID user,void\* pCookie )

void XN\_CALLBACK\_TYPE CalibrationEnd( xn::SkeletonCapability& skeleton,XnUserID user,XnCalibrationStatus calibrationError,void\* pCookie )

调用UserGenerator.GetSkeletonCap().RegisterToCalibrationStart函数和UserGenerator.GetSkeletonCap().RegisterToCalibrationComplete函数分别进行注册。

也可以使用下面的函数将两个函数一起进行注册：

UserGenerator.GetSkeletonCap().RegisterCalibrationCallbacks

**姿势侦测回调函数：**

void XN\_CALLBACK\_TYPE PoseDetected( xn::PoseDetectionCapability& poseDetection,const XnChar\* strPose,XnUserID user,void\* pCookie)

调用userGenerator.GetPoseDetectionCap().RegisterToPoseDetected函数进行注册。

1. 调用Context的StartGeneratingAll()方法打开所有生成器，开始进行检测跟踪。
2. 在While循环中，调用Context的WaitAndUpdateAll方法更新数据，进行处理，主要包括：

用户生成器检查当前用户的个数（UserGenerator.GetNumberOfUsers），对每一个用户检查是否正被跟踪，若正在跟踪，对骨骼的24个节点坐标进行获取，方法是使用SkeletonCapability对象的GetSkeletonJoint函数，得到绝对坐标，使用DepthGenerator的ConvertRealWorldToProjective函数转换到图像坐标，进行绘制。

1. 停止生成器，关闭Context。

## 实例

#include "stdafx.h"

#include <stdlib.h>

#include <iostream>

#include <vector>

#include <XnCppWrapper.h>

#include <XnModuleCppInterface.h>

#include <opencv2/opencv.hpp>

using namespace std;

using namespace cv;

using namespace xn;

xn::UserGenerator userGenerator;

xn::DepthGenerator depthGenerator;

xn::ImageGenerator imageGenerator;

/\*

XN\_SKEL\_HEAD = 1, XN\_SKEL\_NECK = 2,

XN\_SKEL\_TORSO = 3, XN\_SKEL\_WAIST = 4,

XN\_SKEL\_LEFT\_COLLAR = 5, XN\_SKEL\_LEFT\_SHOULDER = 6,

XN\_SKEL\_LEFT\_ELBOW = 7, XN\_SKEL\_LEFT\_WRIST = 8,

XN\_SKEL\_LEFT\_HAND = 9, XN\_SKEL\_LEFT\_FINGERTIP =10,

XN\_SKEL\_RIGHT\_COLLAR =11, XN\_SKEL\_RIGHT\_SHOULDER =12,

XN\_SKEL\_RIGHT\_ELBOW =13, XN\_SKEL\_RIGHT\_WRIST =14,

XN\_SKEL\_RIGHT\_HAND =15, XN\_SKEL\_RIGHT\_FINGERTIP =16,

XN\_SKEL\_LEFT\_HIP =17, XN\_SKEL\_LEFT\_KNEE =18,

XN\_SKEL\_LEFT\_ANKLE =19, XN\_SKEL\_LEFT\_FOOT =20,

XN\_SKEL\_RIGHT\_HIP =21, XN\_SKEL\_RIGHT\_KNEE =22,

XN\_SKEL\_RIGHT\_ANKLE =23, XN\_SKEL\_RIGHT\_FOOT =24

\*/

//a line will be drawn between start point and corresponding end point

int startSkelPoints[14]={1,2,6,6,12,17,6,7,12,13,17,18,21,22};

int endSkelPoints[14]={2,3,12,21,17,21,7,9,13,15,18,20,22,24};

// callback function of user generator: new user

void XN\_CALLBACK\_TYPE NewUser( xn::UserGenerator& generator, XnUserID user,void\* pCookie )

{

cout << "New user identified: " << user << endl;

generator.GetPoseDetectionCap().StartPoseDetection("Psi", user);

}

// callback function of user generator: lost user

void XN\_CALLBACK\_TYPE LostUser( xn::UserGenerator& generator, XnUserID user,void\* pCookie )

{

cout << "User " << user << " lost" << endl;

}

// callback function of skeleton: calibration start

void XN\_CALLBACK\_TYPE CalibrationStart( xn::SkeletonCapability& skeleton,XnUserID user,void\* pCookie )

{

cout << "Calibration start for user " << user << endl;

}

// callback function of skeleton: calibration end

void XN\_CALLBACK\_TYPE CalibrationEnd( xn::SkeletonCapability& skeleton,XnUserID user,XnCalibrationStatus calibrationError,void\* pCookie )

{

cout << "Calibration complete for user " << user << ", ";

if( calibrationError==XN\_CALIBRATION\_STATUS\_OK )

{

cout << "Success" << endl;

skeleton.StartTracking( user );

}

else

{

cout << "Failure" << endl;

((xn::UserGenerator\*)pCookie)->GetPoseDetectionCap().StartPoseDetection( "Psi", user );

}

}

// callback function of pose detection: pose start

void XN\_CALLBACK\_TYPE PoseDetected( xn::PoseDetectionCapability& poseDetection,const XnChar\* strPose,XnUserID user,void\* pCookie)

{

cout << "Pose " << strPose << " detected for user " << user << endl;

((xn::UserGenerator\*)pCookie)->GetSkeletonCap().RequestCalibration( user, FALSE );

poseDetection.StopPoseDetection( user );

}

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

{

char key=0;

int imgPosX=0;

int imgPosY=0;

// initial context

xn::Context context;

context.Init();

xn::ImageMetaData imageMD;

Mat cameraImg(Size(640,480),CV\_8UC3,3);

// map output mode

XnMapOutputMode mapMode;

mapMode.nXRes = 640;

mapMode.nYRes = 480;

mapMode.nFPS = 30;

// create generator

depthGenerator.Create( context );

depthGenerator.SetMapOutputMode( mapMode );

imageGenerator.Create( context );

userGenerator.Create( context );

// Register callback functions of user generator

XnCallbackHandle userCBHandle;

userGenerator.RegisterUserCallbacks( NewUser, LostUser, NULL, userCBHandle );

// Register callback functions of skeleton capability

xn::SkeletonCapability skeletonCap = userGenerator.GetSkeletonCap();

skeletonCap.SetSkeletonProfile( XN\_SKEL\_PROFILE\_ALL );

XnCallbackHandle calibCBHandle;

skeletonCap.RegisterToCalibrationStart( CalibrationStart,&userGenerator, calibCBHandle );

skeletonCap.RegisterToCalibrationComplete( CalibrationEnd,&userGenerator, calibCBHandle );

// Register callback functions of Pose Detection capability

XnCallbackHandle poseCBHandle;

userGenerator.GetPoseDetectionCap().RegisterToPoseDetected( PoseDetected,&userGenerator, poseCBHandle );

// start generate data

context.StartGeneratingAll();

while( key!=27 )

{

context.WaitAndUpdateAll();

imageGenerator.GetMetaData(imageMD);

memcpy(cameraImg.data,imageMD.Data(),640\*480\*3);

cvtColor(cameraImg,cameraImg,CV\_RGB2BGR);

// get users

XnUInt16 userCounts = userGenerator.GetNumberOfUsers();

if( userCounts > 0 )

{

XnUserID\* userID = new XnUserID[userCounts];

userGenerator.GetUsers( userID, userCounts );

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

{

// if is tracking skeleton

if( skeletonCap.IsTracking( userID[i] ) )

{

XnPoint3D skelPointsIn[24],skelPointsOut[24];

XnSkeletonJointTransformation mJointTran;

for(int iter=0;iter<24;iter++)

{

//XnSkeletonJoint from 1 to 24 skeletonCap.GetSkeletonJoint( userID[i],XnSkeletonJoint(iter+1), mJointTran );

skelPointsIn[iter]=mJointTran.position.position;

} depthGenerator.ConvertRealWorldToProjective(24,skelPointsIn,skelPointsOut);

for(int d=0;d<14;d++)

{

Point startpoint = Point(skelPointsOut[startSkelPoints[d]-1].X,skelPointsOut[startSkelPoints[d]-1].Y);

Point endpoint = Point(skelPointsOut[endSkelPoints[d]-1].X,skelPointsOut[endSkelPoints[d]-1].Y);

circle(cameraImg,startpoint,12,Scalar(255,0,0),3);

circle(cameraImg,startpoint,12,Scalar(255,0,0),3);

line(cameraImg,startpoint,endpoint,Scalar(0,0,255),3);

}

}

}

delete [] userID;

}

imshow("Camera",cameraImg);

key=waitKey(20);

}

context.StopGeneratingAll();

context.Shutdown();

return 0;

}