Kinect骨骼和动作识别实验

# 【实验目的】

利用Kinect捕获的人体骨骼特征数据，实现人体手势识别算法。Kinect的使用，可详见文档《[Kinect使用说明](Kinect使用说明.doc)》。

# 【实验原理】

Kinect2 采用TOF技术来获取被测物体的深度。红外发射器主动投射经调制的近红外光线，红外光线照到视野里的物体上就会发生反射，红外相机接收反射回来的红外线，采用TOF技术测量深度，计算光的时间差（通常是通过相位差来计算的），从而得到物体的深度（即物体到深度相机的距离）。

骨架追踪系统将检测到的深度图像数据转换到人体各个关节点坐标。该系统能同时追踪6个人，识别2个人的动作；每个人包含25个关节点数据，包括了X、Y、Z数据。Kinect采用分隔策略将人体从复杂的背景中区分出来，接下来kinect会对深度图像进行评估（机器学习），来判断人体的不同部位。

# 【实验环境】

**开发环境：**

Windows 8或Windows 10系统、VS2012及以上版本、 Kinect设备、PC机必须具备USB3.0接口

**编程语言:**

C/C++

# 【实验步骤】

## 新建项目

打开“Visual Studio”。点击“文件/新建/项目/”来创建一个新的工程，创建工程的界面如下所示，选择Visual C++的Win32项目，依次输入应用名称，和保存路径。如图中所示。

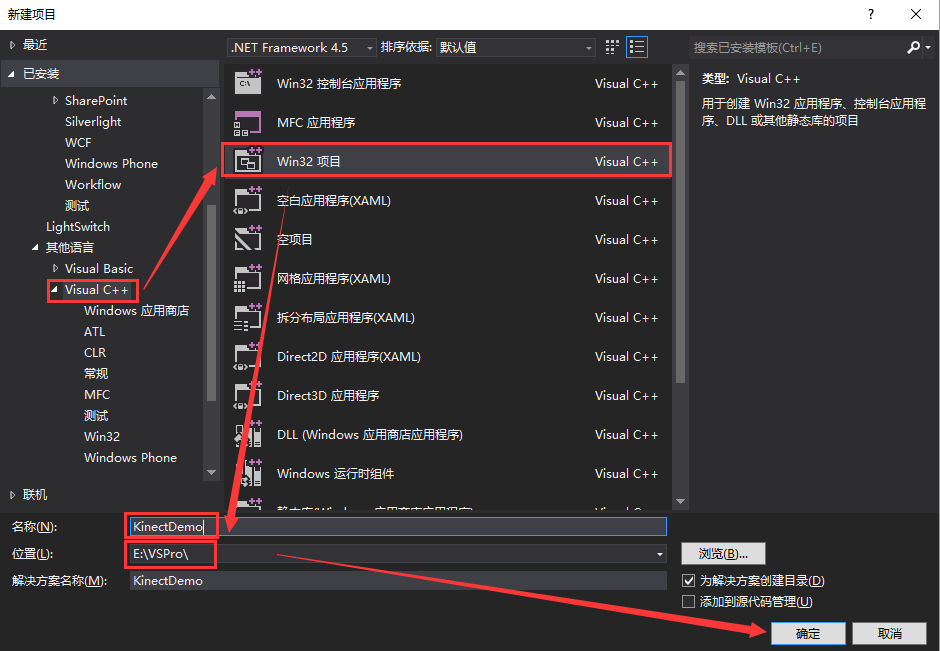


图1

点击“确定”，进入应用程序向导。选择默认设置，点击“下一步”，直至完成。至此一个空项目创建完毕，初始项目如下图所示。头文件：一般为类、变量和函数的定义；源文件：为类和函数的实现；资源文件：包含窗体、图标等；应用程序入口：应用运行时，首先执行的函数。可以运行该项目，查看一下效果。

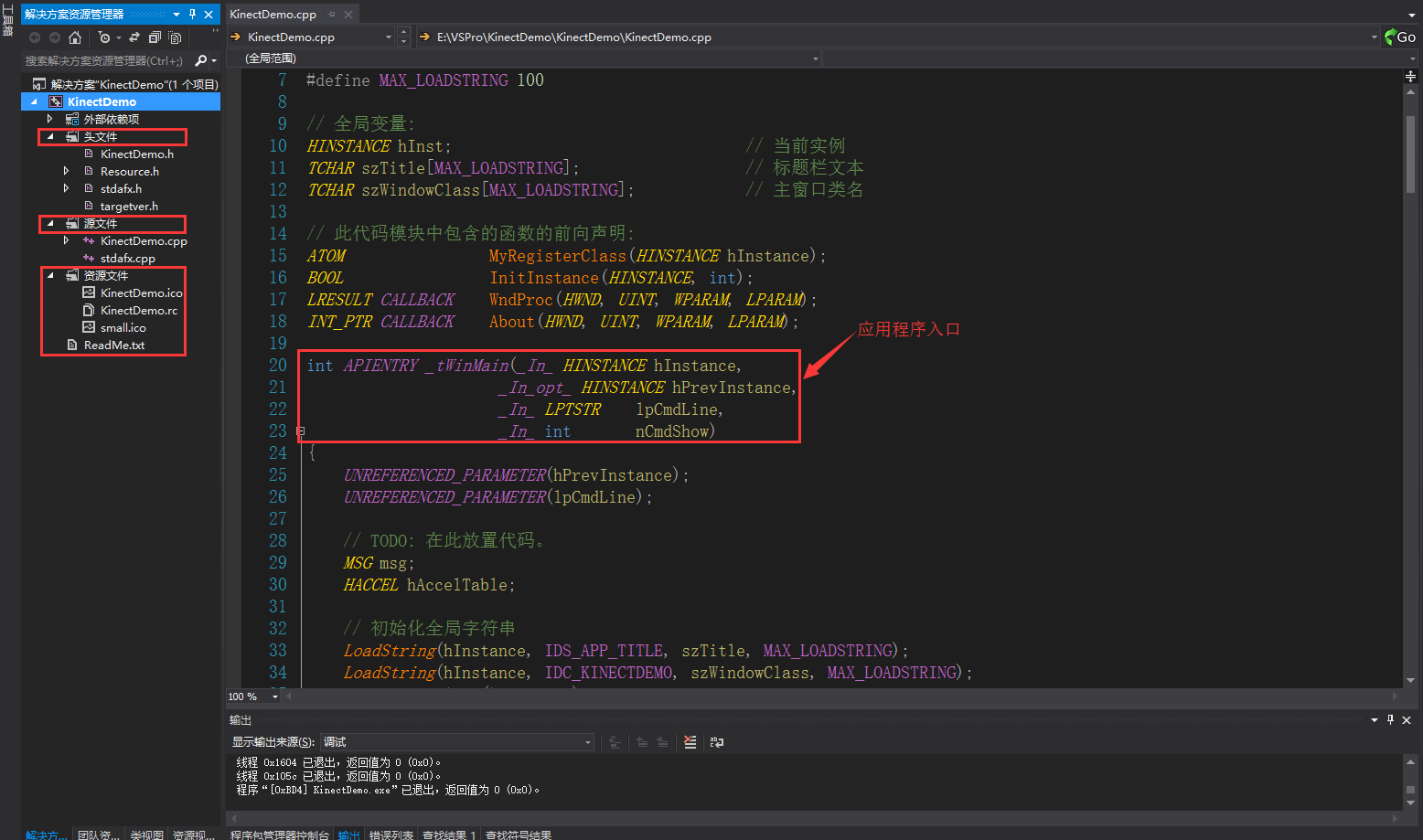


图2

# 二、创建手势识别子窗口

为了结构清晰，我们在主窗口上包含一个按钮，点击该按钮后进入子窗口进行手势的识别。首先对主窗口进行一些改动。

1）修改主窗口的大小

在主窗口的初始化函数InitInstance中做如下修改，更改窗口大小。

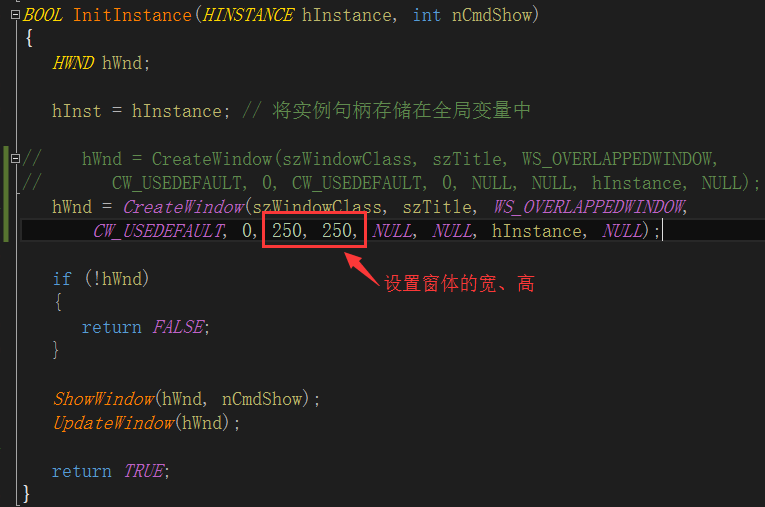


图3

2）创建一个按钮

在主窗口中创建一个按钮，当点击该按钮后，跳转到手势识别窗口。在主窗口的消息处理函数WndProc的switch代码中块添加如下代码，完成按钮的添加，并运行查看效果。

case WM\_CREATE:

{

// 获取主窗体的位置和大小

RECT region;

GetClientRect(hWnd, &region);

// 窗体宽高

int with = region.right - region.left;

int height = region.bottom - region.top;

// 按钮宽高

int btnWidth = with - 20;

int btnHeight = 30;

// 按钮在窗体中的位置

int x = (with - btnWidth) / 2;

int y = 20;

HWND button = CreateWindow(

\_TEXT("Button"),// 类型

\_TEXT("手势识别"),// 名称

WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON, // 风格：可视，子窗口，按钮

x, y, btnWidth, btnHeight, // 位置和大小

hWnd, // 父窗口

(HMENU)10001, // 按钮的ID

hInst,

NULL

);

SendMessage(button, WM\_SETFONT, (WPARAM)GetStockObject(DEFAULT\_GUI\_FONT), 0); // 设置字体

}

break;



图4

3）响应按钮的点击事件

在主窗口的消息处理函数WndProc的switch代码块中添加按钮点击事件响应，如下图所示：

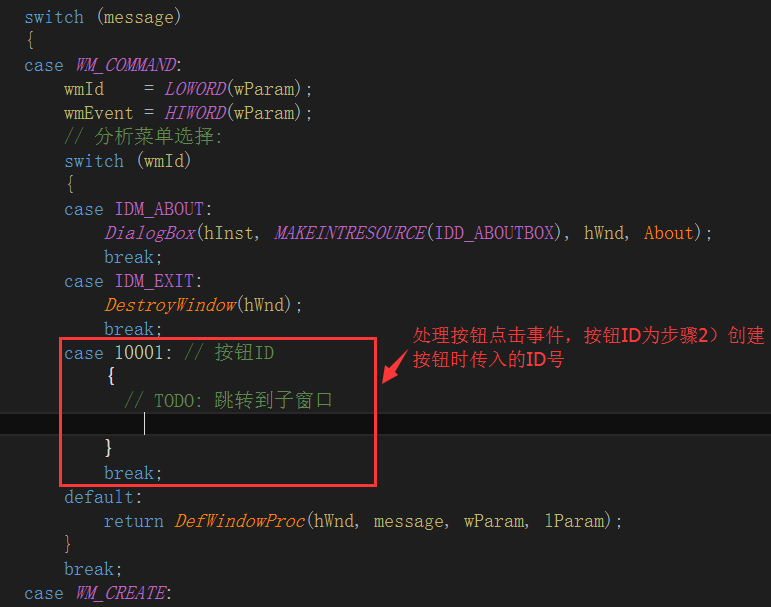


图5

4）创建子窗口

点击“解决方案资源管理器”底部的资源视图，找到“Dialog”，右击鼠标，选择“添加资源”，在弹出的框中选择“Dialog”，点击“新建”。如下图所示：

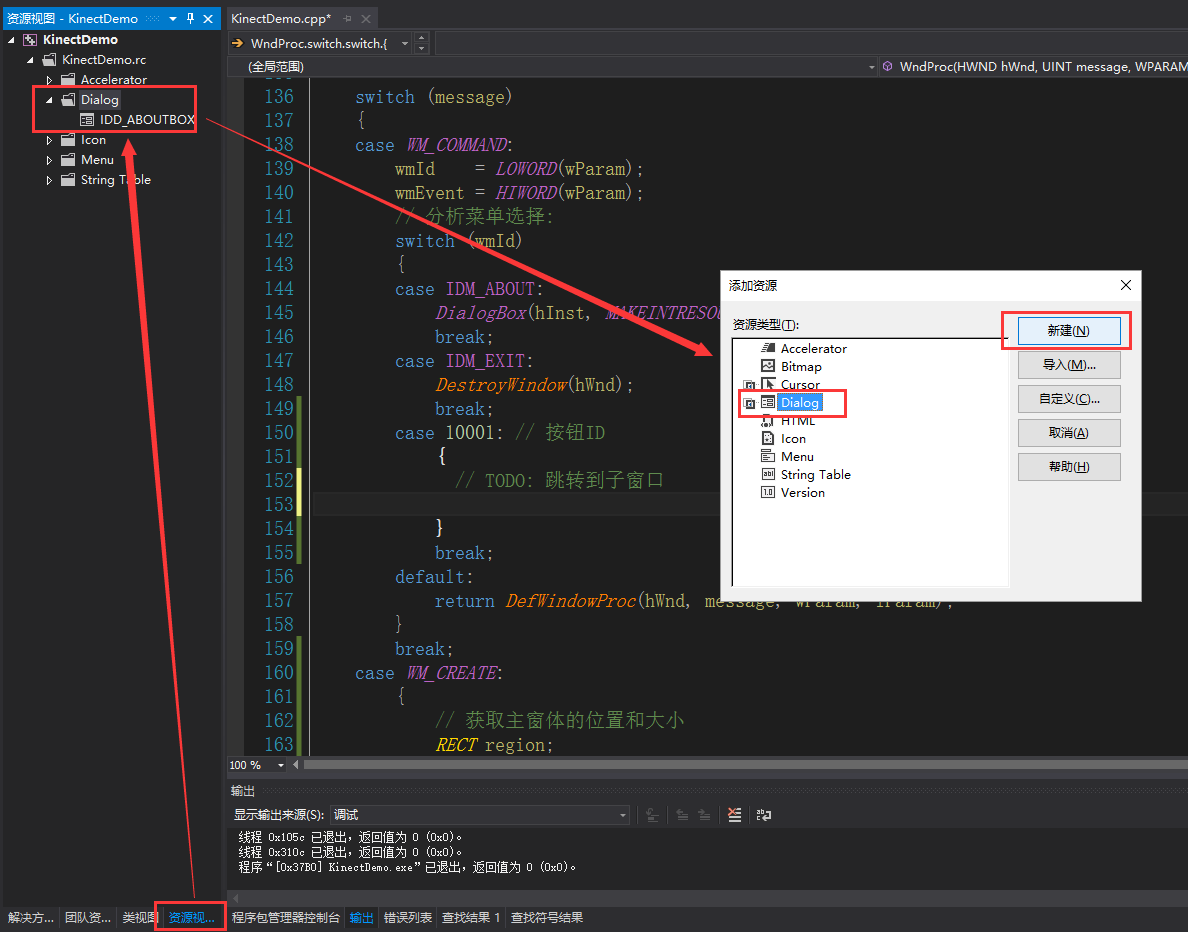


图6

在左侧的属性栏中，将ID修改为：IDD\_BODY，Caption修改为：手势识别，ClassName修改为：KinectBodyCls。如下图所示：

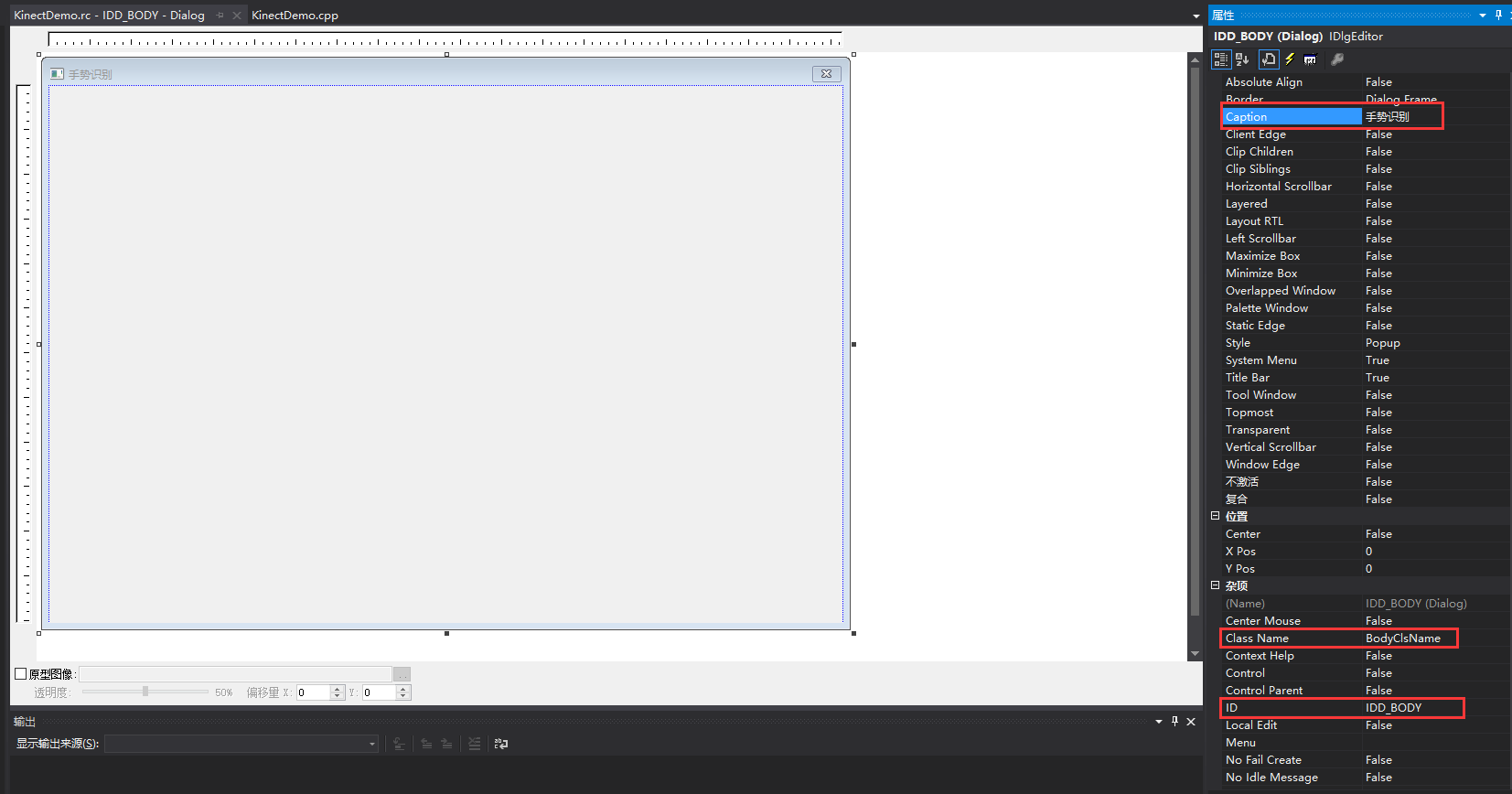


图7

在窗口中添加一个Picture Control控件，点击“工具箱”，找到Picture Control控件，并将其拖至窗口中合适的位置，适当调整大小，并设置如下的属性：将ID设置为：IDC\_BODY。如下图所示。

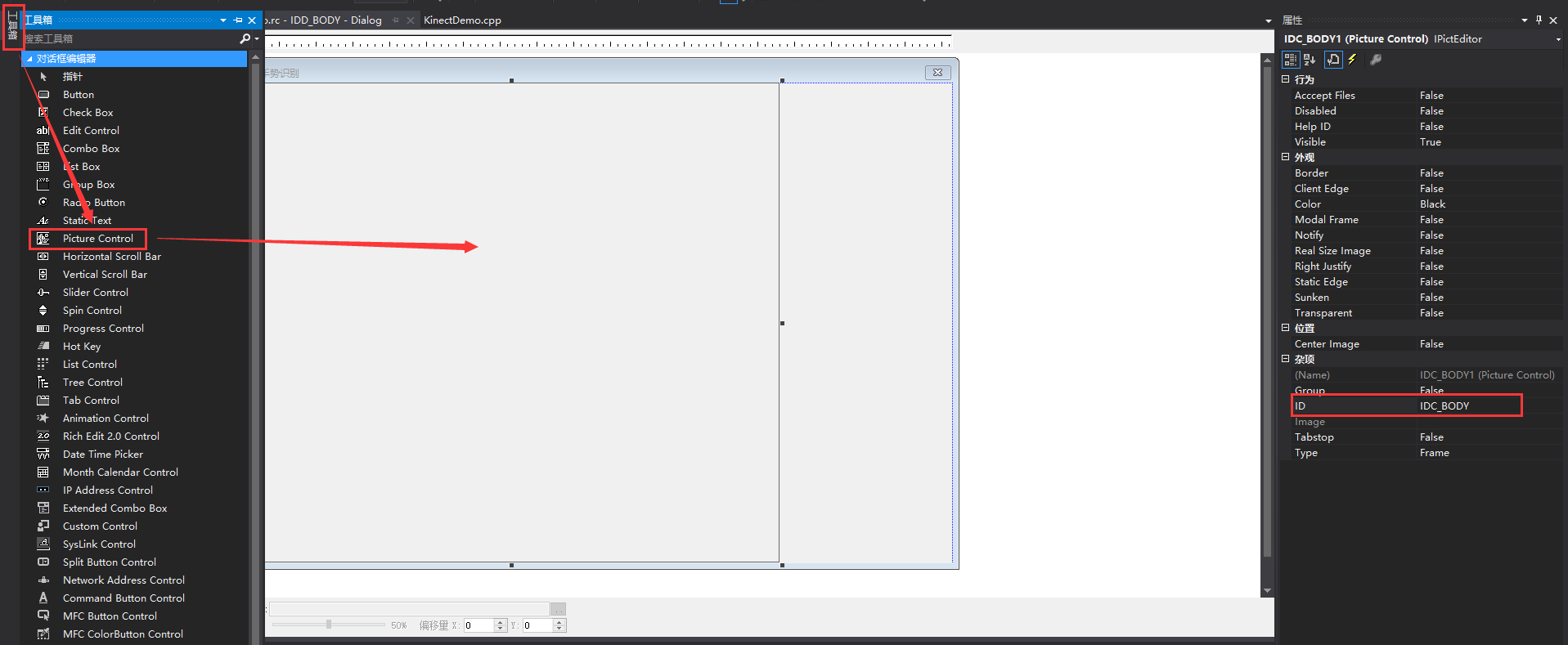


图8

用同样的方式添加静态文本(Static Text)，并将其Caption设置为：识别结果。添加三个静态文本，并将Caption分别设置为：右手状态、左手状态、手势。再添加三个静态文本，其ID分别设置为：IDC\_RIGHT\_HAND、IDC\_LEFT\_HAND、IDC\_GESTURE；Border设置为true；Sunken设置为true。最终如下图所示：

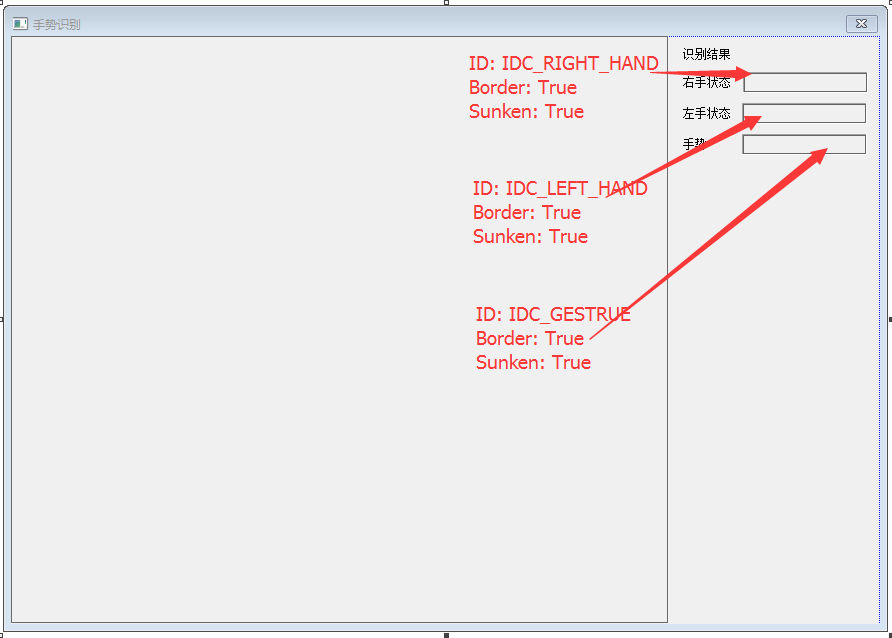


图9

5）创建一个新类KinectBody作为子窗口的载体

创建方法：解决方案右击，选择“添加”，选择“类”，在添加类的对话框中选择“C++类”，点击“创建”出现类创建向导，键入类名为KinectBody，如下图所示。

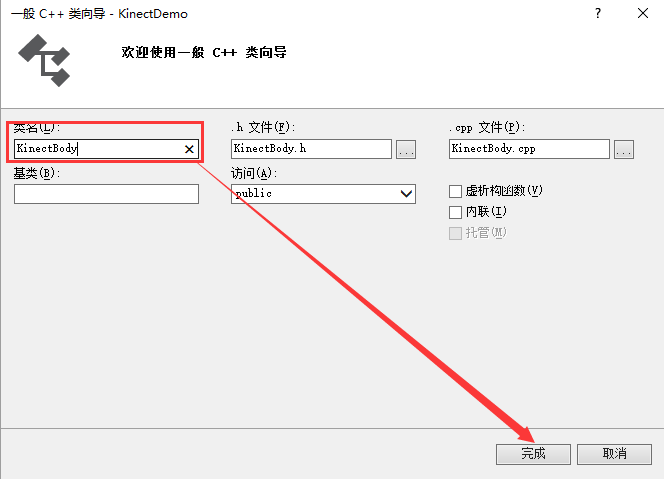


图10

在KinectBody.h中定义三个公有函数和一个私有变量。如下图所示：

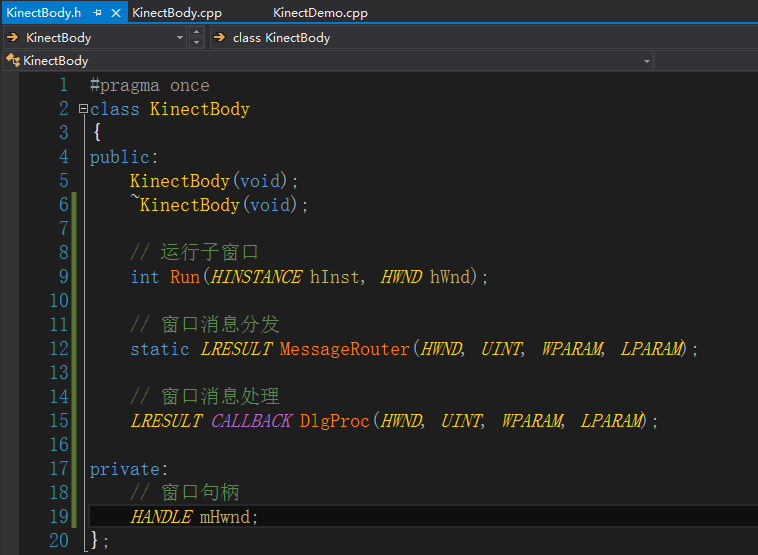


图11

在KinectBody.cpp中，我们来实现这个三个函数。首先在文档的开头引入资源文件：#include “resource.h”。

Run函数，主要工作是创建窗体并显示，代码如下：

int KinectBody**::**Run**(**HINSTANCE hInst**,** HWND parent**)**

**{**

EnableWindow**(**parent**,** **false);**

MSG msg **=** **{**0**};**

hGlobalInst **=** hInst**;**// 保存对应用的全局引用

// 初始化窗体类

WNDCLASS wc**;**

ZeroMemory**(&**wc**,** **sizeof(**wc**));**

wc**.**style **=** CS\_HREDRAW **|** CS\_VREDRAW**;**

wc**.**cbWndExtra **=** DLGWINDOWEXTRA**;**

wc**.**hCursor **=** LoadCursor**(NULL,** IDC\_ARROW**);**

wc**.**hIcon **=** LoadIcon**(**hInst**,** MAKEINTRESOURCE**(**IDI\_KINECTDEMO**));**

wc**.**lpfnWndProc **=** DefDlgProc**;**

wc**.**lpszClassName **=** \_TEXT**(**"KinectBodyCls"**);**

**if** **(!**RegisterClass**(&**wc**))**

**{**

EnableWindow**(**parent**,** TRUE**);**

SetForegroundWindow**(**parent**);**

**return** 0**;**

**}**

// 创建窗体

HWND hWnd **=** CreateDialogParam**(**

**NULL,**

MAKEINTRESOURCE**(**IDD\_BODY**),**

**NULL,**

**(**DLGPROC**)** KinectBody**::**MessageRouter**,**

**reinterpret\_cast<**LPARAM**>** **(this)**

**);**

// 屏幕大小

int screenWidth **=** GetSystemMetrics**(**SM\_CXSCREEN**);**

int screenHeight **=** GetSystemMetrics**(**SM\_CYSCREEN**);**

RECT region**;**

GetWindowRect**(**hWnd**,** **&**region**);**

int w **=** region**.**right **-** region**.**left**;**

int h **=** region**.**bottom **-** region**.**top**;**

int x **=** **(**screenWidth **-** w**)** **/** 2**;**

int y **=** **(**screenHeight **-** h**)** **/** 2**;**

MoveWindow**(**hWnd**,** x**,** y**,** w**,** h**,** FALSE**);**

ShowWindow**(**hWnd**,**SW\_SHOW**);**

**while** **(**WM\_QUIT **!=** msg**.**message**)**

**{**

**while** **(**PeekMessage**(&**msg**,** **NULL,** 0**,** 0**,** PM\_REMOVE**))**

**{**

**if** **(**hWnd **&&** IsDialogMessage**(**hWnd**,** **&**msg**))**

**{**

**continue;**

**}**

TranslateMessage**(&**msg**);**

DispatchMessage**(&**msg**);**

**}**

**}**

UnregisterClass**(**\_TEXT**(**"KinectBodyCls"**),** hInst**);**

EnableWindow**(**parent**,** TRUE**);**

SetForegroundWindow**(**parent**);**

**return** 1**;**

**}**

MessageRouter函数负责窗体消息的分发，代码如下：

LRESULT CALLBACK KinectBody**::**MessageRouter**(**HWND hWnd**,** UINT uMsg**,** WPARAM wParam**,** LPARAM lParam**)**

**{**

KinectBody**\*** pThis **=** **NULL;**

**if** **(**WM\_INITDIALOG **==** uMsg**)**

**{**

pThis **=** **reinterpret\_cast<**KinectBody**\*>** **(**lParam**);**

SetWindowLongPtr**(**hWnd**,** GWLP\_USERDATA**,** **reinterpret\_cast<**LONG\_PTR**>** **(**pThis**));**

**}**

**else**

**{**

pThis **=** **reinterpret\_cast<**KinectBody**\*>** **(**GetWindowLongPtr**(**hWnd**,** GWLP\_USERDATA**));**

**}**

**if** **(**pThis**)**

**{**

**return** pThis**->**DlgProc**(**hWnd**,** uMsg**,** wParam**,** lParam**);**

**}**

**return** 0**;**

**}**

DlgProc函数负责窗体消息的处理，代码如下：

LRESULT CALLBACK KinectBody**::**DlgProc**(**HWND hWnd**,** UINT message**,** WPARAM wParam**,** LPARAM lParam**)**

**{**

int wmId**,** wmEvent**;**

UNREFERENCED\_PARAMETER**(**wParam**);**

UNREFERENCED\_PARAMETER**(**lParam**);**

**switch** **(**message**)**

**{**

**case** WM\_INITDIALOG**:**

**break;**

**case** WM\_CLOSE**:**

DestroyWindow**(**hWnd**);**

**break;**

**case** WM\_DESTROY**:**

PostQuitMessage**(**0**);**

**break;**

**default:**

**break;**

**}**

**return** FALSE**;**

**}**

添加完成后可以运行查看效果。

# 三、添加库类

对于Kinect的一些基本操作已经编写好了，这里只要将其添加到工程中。

1）修改“stdafx.h”，在该文件中添加一段代码，该代码用来释放创建的指针。代码如下所示：

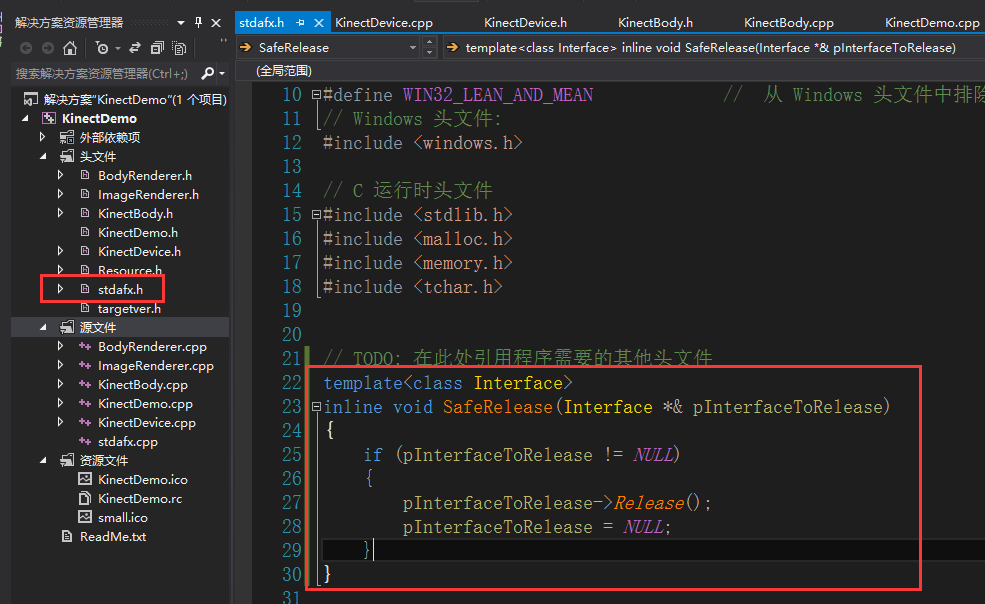


图12

2）添加类库，先将source文件夹中的文件全部拷贝到工程目录/KinectDemo/下。在解决方案资源管理器中的“头文件”上右击鼠标，选择“添加”，选择“现有项”，如图所示：

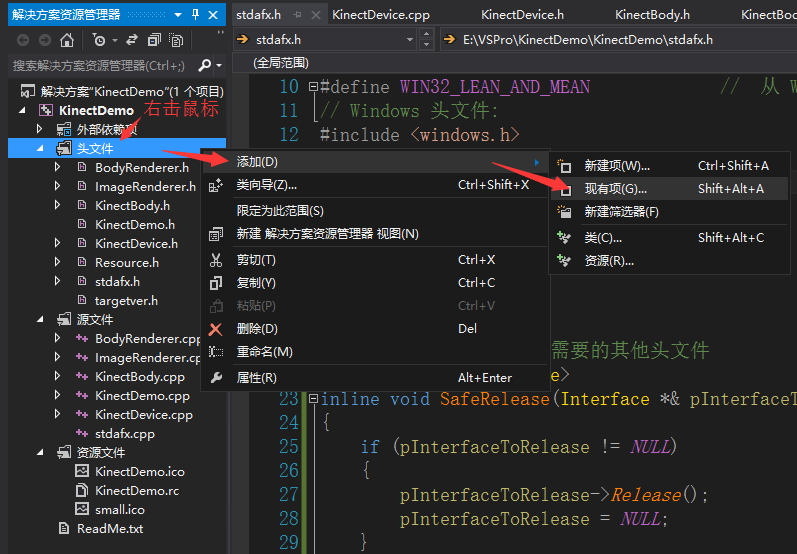


图13

在弹出的文件选择器中，选择所有新加入的.h文件，再点击添加。

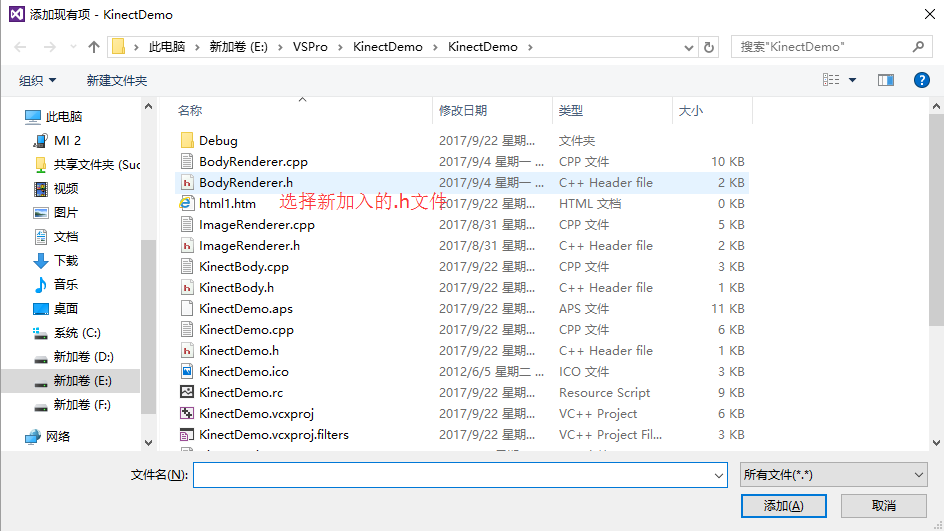


图14

用同样的方法，将新加入的.cpp文件添加到“源文件”中。

# 四、显示骨骼图像

1）引入头文件

在KinectBoyd.h头文件中添加”KinectDevice.h”头文，并添加如下的变量和方法：



图15

2）初始化和配置Kinect设备

在KinectBody.cpp中定义个一个全局的函数HandleBodyFrame，该函数由kinect设备进行回调。如下图所示：

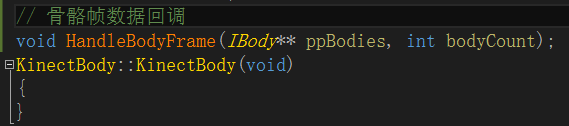


图16

在窗体消息处理函数DlgProc中switch代码块中添加kinect设备的初始代码：

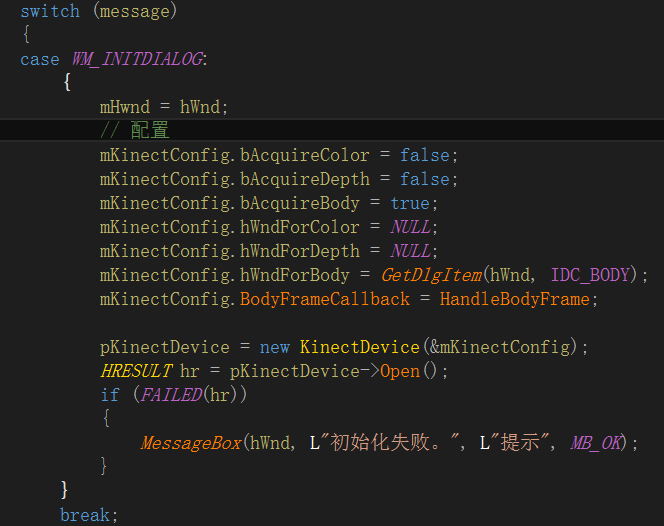


图17

3）更新Kinect数据帧

实现KinectBody的Update函数，如下所示：

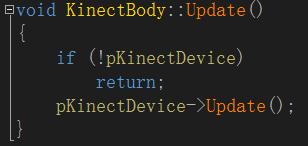


图18

在KinectBody.cpp的Run函数中添加，对Update函数的调用，如下图所示：

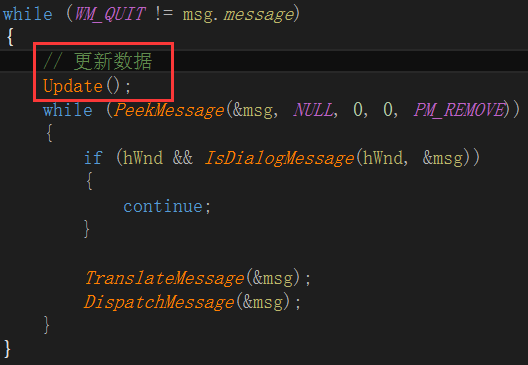


图19

在KinectBody.cpp中实现HandleBodyFrame函数，先不添加任何代码。

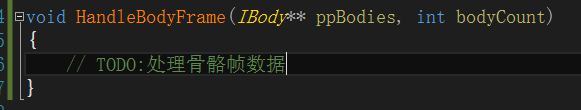


图20

4）运行程序，查看效果

编译运行，无错误的情况下将看到如下的画面：

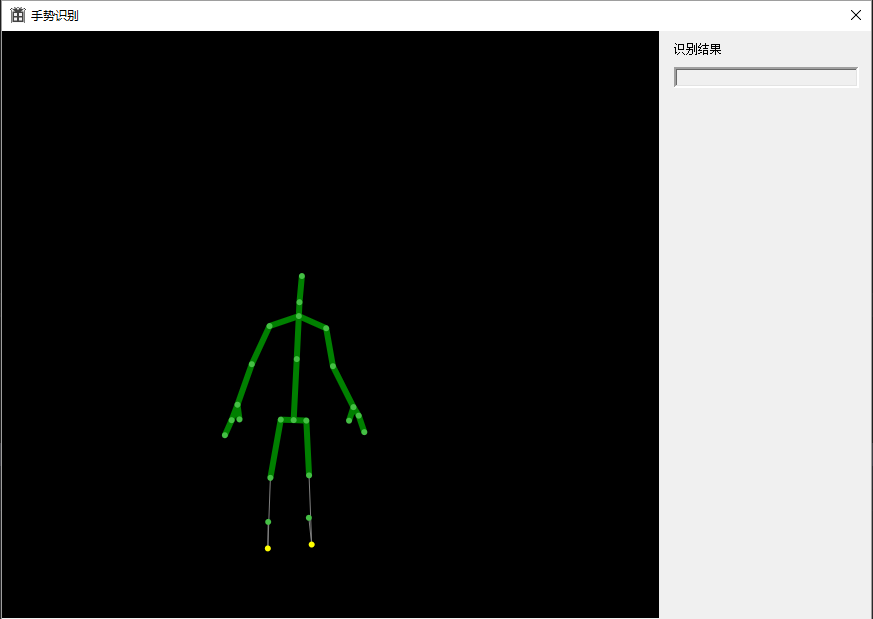


图21

# 五、手势识别

1）左右手状态识别

在Kinect骨骼数据中，包含了人体左右手的状态：Open，Closed，Lasso，Not tracked，Unknown。这里只处理离镜头最近的人的手势。在骨骼帧数据回调函数HandleBodyFrame中添加如下代码：

void HandleBodyFrame**(**IBody**\*\*** ppBodies**,** int bodyCount**)**

**{**

// TODO:处理骨骼帧数据

IBody**\*** pBody **=** **NULL;**

float distance **=** **-**1**;**

// 寻找离镜头最近的人体

**for** **(**int i **=** 0**;** i **<** bodyCount**;** i **++)**

**{**

IBody**\*** tmpBody **=** ppBodies**[**i**];**

**if** **(**tmpBody**)**

**{**

BOOLEAN bTracked **=** FALSE**;**

tmpBody**->**get\_IsTracked**(&**bTracked**);**

**if** **(**bTracked**)**

**{**

Joint joints**[**JointType\_Count**];**

tmpBody**->**GetJoints**(**\_countof**(**joints**),** joints**);**

Joint head **=** joints**[**JointType\_Head**];**

float tmpDis **=** head**.**Position**.**Z**;**

**if** **(**distance **>=** 0**)**

**{**

**if** **(**tmpDis **<** distance**)**

**{**

distance **=** tmpDis**;**

pBody **=** tmpBody**;**

**}**

**}**

**else**

**{**

distance **=** tmpDis**;**

pBody **=** tmpBody**;**

**}**

**}**

**}**

**}**

**if** **(**pBody **==** **NULL)**

**return;**

WCHAR result**[**60**];**

HandState righHandState **=** HandState\_Unknown**;**

HRESULT hr **=** pBody**->**get\_HandRightState**(&**righHandState**);**// 获取右手状态

**if** **(**SUCCEEDED**(**hr**))**

**{**

StringCchPrintf**(**result**,** \_countof**(**result**),** L"状态：%s\t"**,** **(**righHandState **==** HandState\_Open**?** L"打开" **:** L"握拳"**));** // 只判断2种

SetDlgItemText**(**hGlobalHwnd**,** IDC\_RIGHT\_HAND**,** result**);**

**}**

HandState leftHandState **=** HandState\_Unknown**;**

hr **=** pBody**->**get\_HandLeftState**(&**leftHandState**);** // 获取左手状态

**if** **(**SUCCEEDED**(**hr**))**

**{**

StringCchPrintf**(**result**,** \_countof**(**result**),** L"状态：%s\t"**,** **(**leftHandState **==** HandState\_Open**?** L"打开" **:** L"握拳"**));**

SetDlgItemText**(**hGlobalHwnd**,** IDC\_LEFT\_HAND**,** result**);**

**}**

**}**

注意：这里有个全局变量hGlobalHwnd，保存了对当前窗体的引用。在KinectBody.cpp的开始进行定义，如下图所示：



图22

在消息处理函数DlgProc中switch代码块的WM\_INITDIALOG分支中进行初始化，如下图所示：

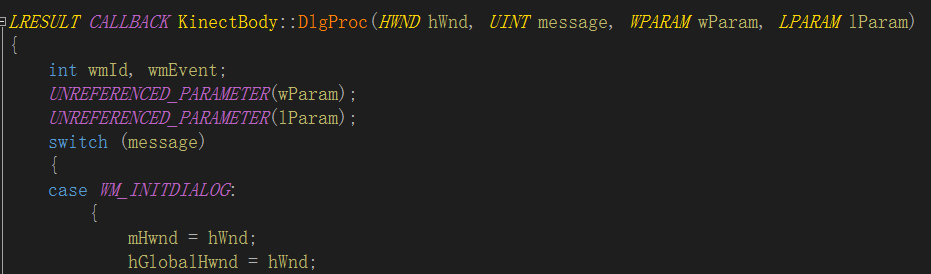


图23

编译运行查看效果。

2）手势识别

这里我们识别两种手势：向右摆手、向左摆手。向右摆手的手势使用左手的骨骼数据进行处理；向左摆手使用左手的骨骼数据进行处理。按照上面的方法，将所给source/gesture/中的库文件加入到工程中。按照如下步骤进行使用：

首先，在KinectBody.cpp中引入头文件“GestureDetection.h”，并定义一个全局变量和手势识别回调函数。如下图所示：

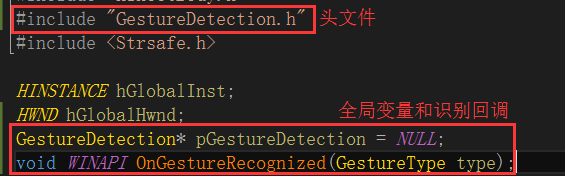


图24

其次，初始化pGestureDetection。在消息处理函数DlgProc中switch的WM\_INITDIALOG分支中添加如下初始代码：

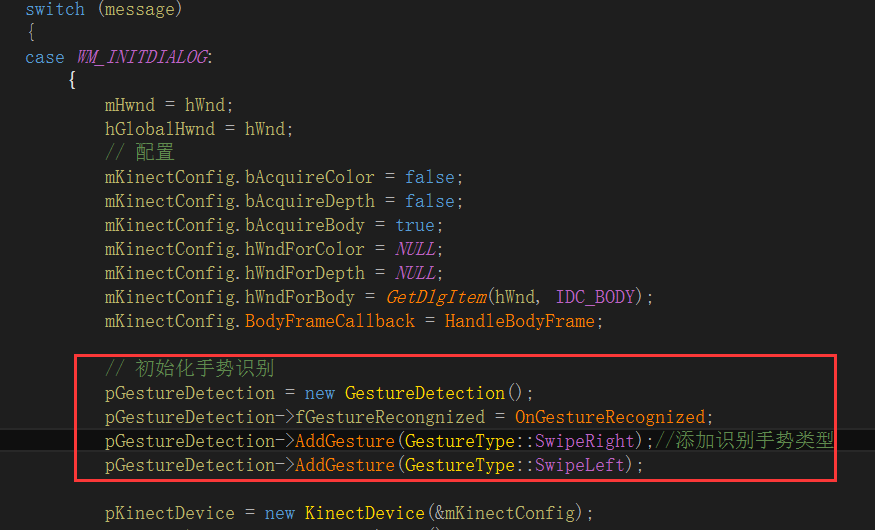


图25

其次，在骨骼帧数据处理函数BodyFrameHandle中在左右手状态识别代码下面添加如下代码：

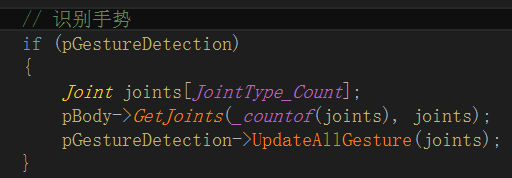


图26

最后，处理手势识别回调函数OnGrestureRecognized。代码如下：

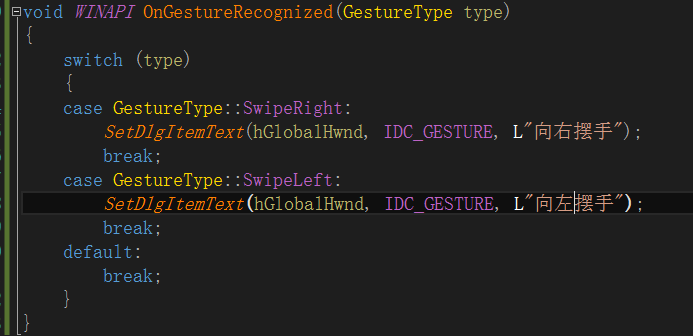


图27

# 六、编译运行

点击工具栏中的运行按钮，如果没有编译错误程序可以正常运行。可以看到如下界面：



图28

# 【实验思考】

1. Kinect2最多能够同时追踪多少个人?
2. 3
3. 4
4. 5
5. 6
6. Kinect2 能够追踪多少个人体骨骼关节点？
7. 24
8. 25
9. 26
10. 27
11. 在实验中很多类都是使用了new关键词来创建一个对象，那么使用new创建的对象保存在什么内存中？怎样释放这些内存？本实验并没有对这些对象进行释放，请完成这部分代码。