**版本信息**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本信息记录表** | | | | |
| 版本号 | 修订内容 | 修订人 | 修订日期 | 审核人 |
| 1.0 | 初版 | 党慧杰 | 2018.08.20 |  |
| 2.0 | 修订 | 党慧杰 | 2018.10.08 |  |
|  |  |  |  |  |

# 获取彩色和深度图像

## 【实验目的】

1、获取Kinect彩色图像

2、获取Kinect深度图像

## 【实验原理】

Kinect设备提供一个彩色摄像头，可以拍摄1920x1080分别率的图像。Kinect采集的图像，通过USB3.0传输给电脑，软件获取传输的图像，并显示在界面上。

同时，Kinect搭载Depth传感器，可以取得Depth数据（和传感器的距离信息）。Kinect通过ToF(Time of Flight)的方式获取Depth的数据，Depth数据的分辨率为512x424。Kinect的红外光源发射出固定频率调制的正弦波，摄像头同步拍摄多帧发射图像，根据每个像素亮度的变化规律计算发射光的相位差，再乘以光速得到每个像素的距离。

## 【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。

**语言**

C/C++

**硬件**

Kinect

## 【实验步骤】

### 一、打开项目

打开“Visual Studio”。点击“文件/打开/项目/解决方案”，选择工程KinectExp，如图中所示。

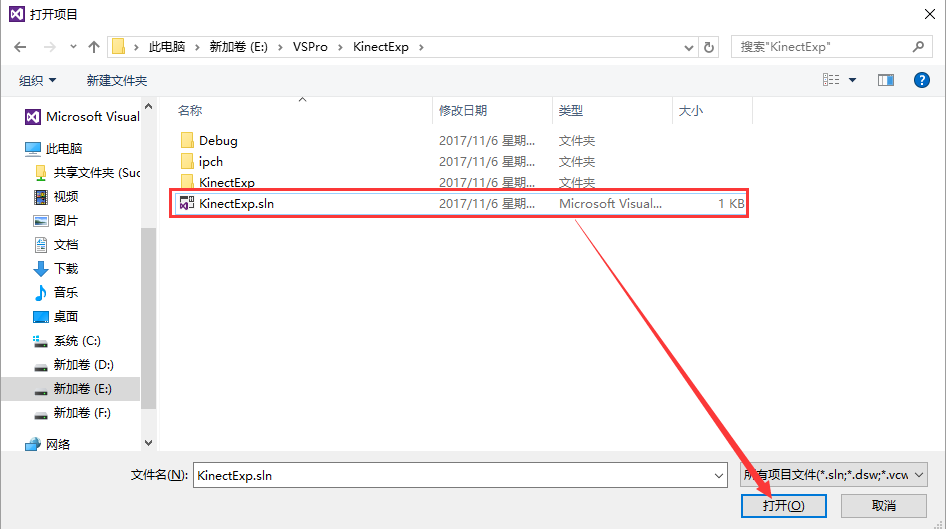


图1

点击“打开”，进入项目，初始项目结构如下：

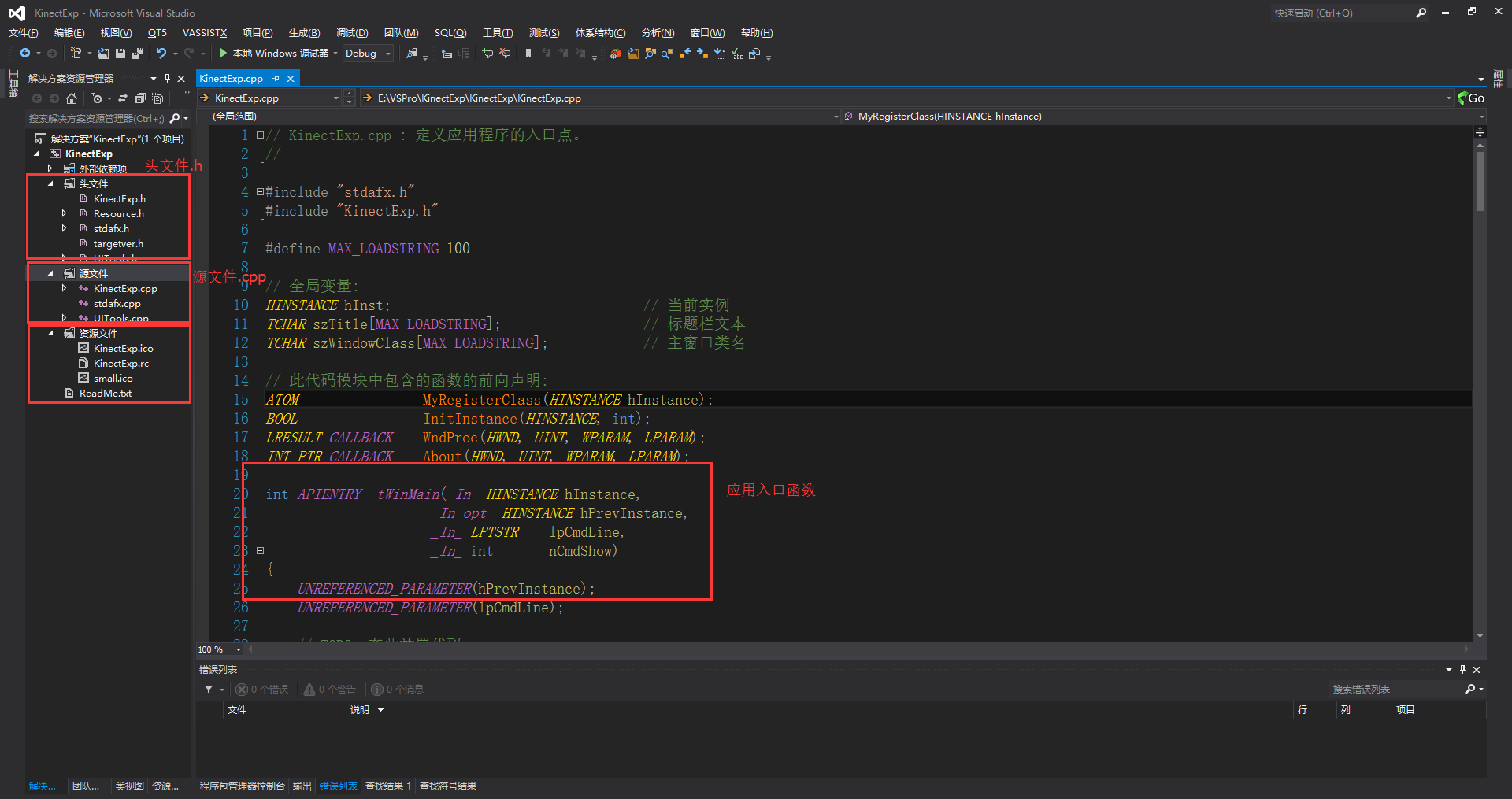


图2

### 二、在界面添加按钮

在界面添加二个按钮，当点击按钮时分别跳转到Kinect彩色图像采集子窗口和深度图像采集子窗口。

1）在KinectExp.cpp中找到函数CreateMainWindow(HWND)。添加如下代码，创建连个按钮：

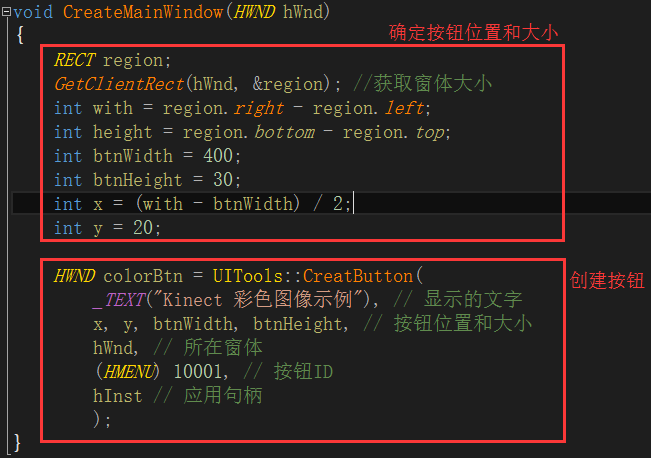


图3

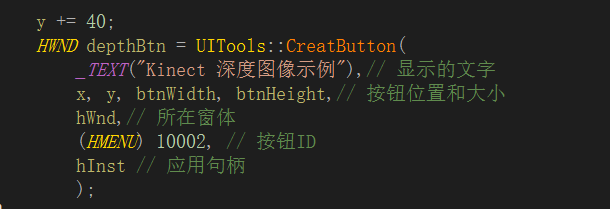


图4

2）添加按钮的点击事件代码

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

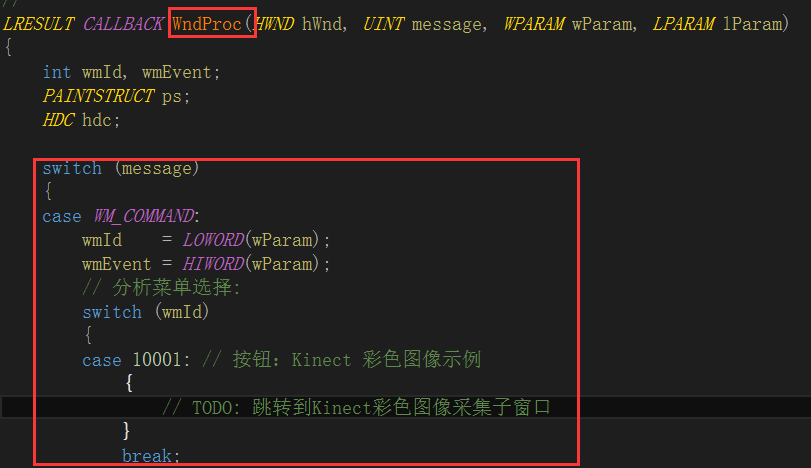


图 5



图 6

至此，可以编译运程，查看界面效果。

### 三、创建Kinect彩色图像子窗口

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：

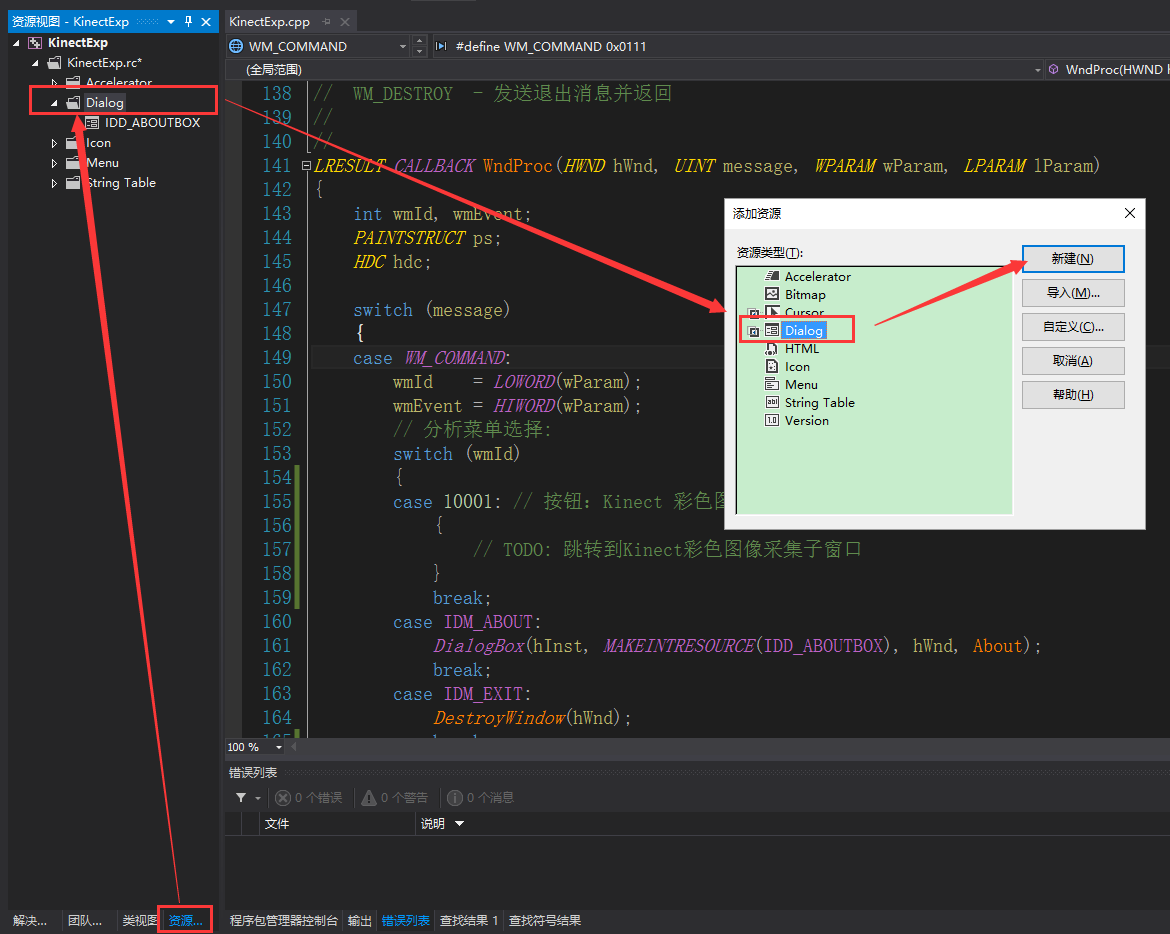


图 7

将窗体，拖动到适当大小（641x373）。在左侧的属性栏中，将ID修改为：IDD\_COLOR，Caption修改为：Kinect彩色图像，ClassName修改为：ColorDlgWndClass。如下图所示：

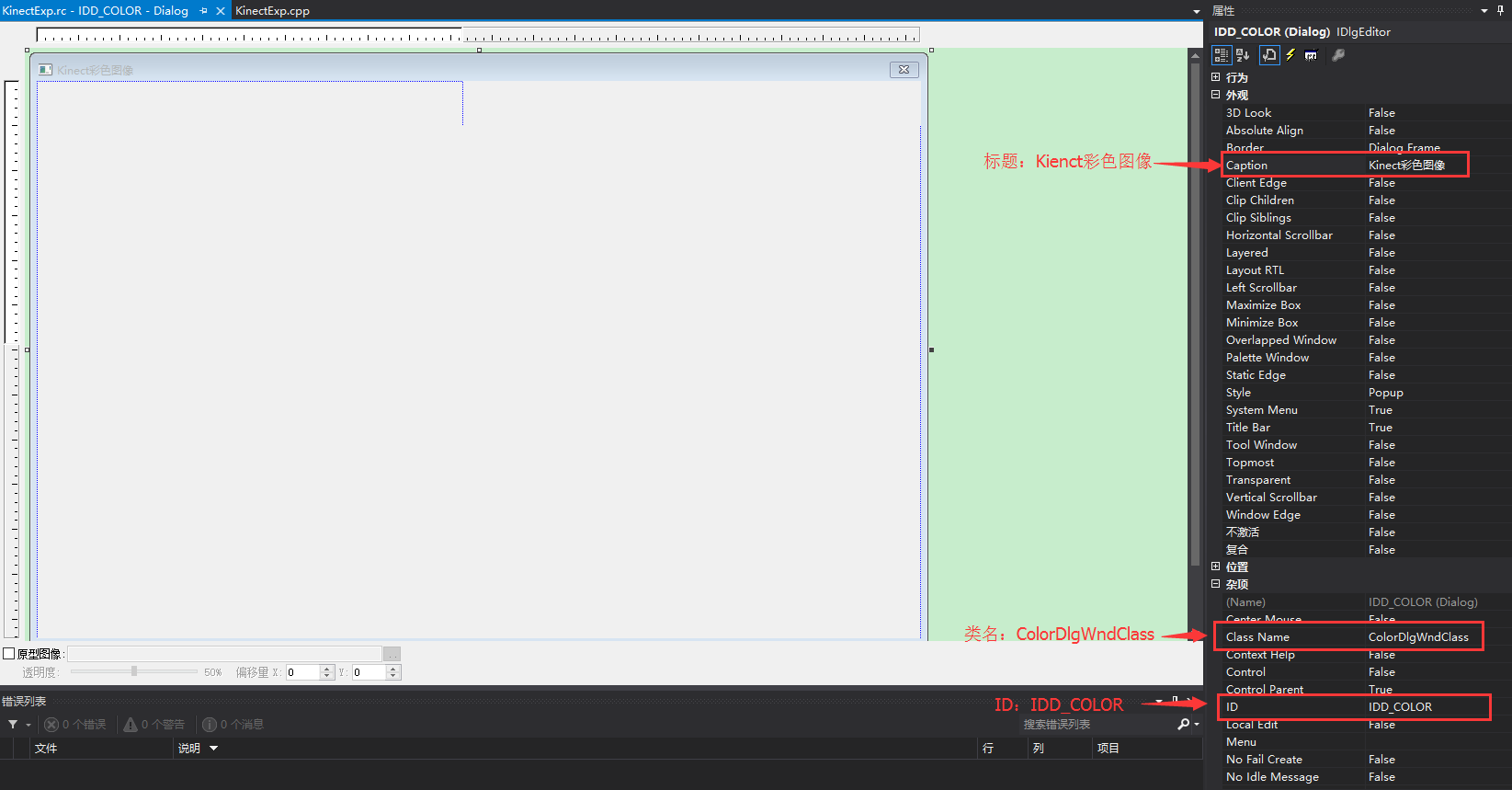


图 8

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\_COLOR。如下图所示。

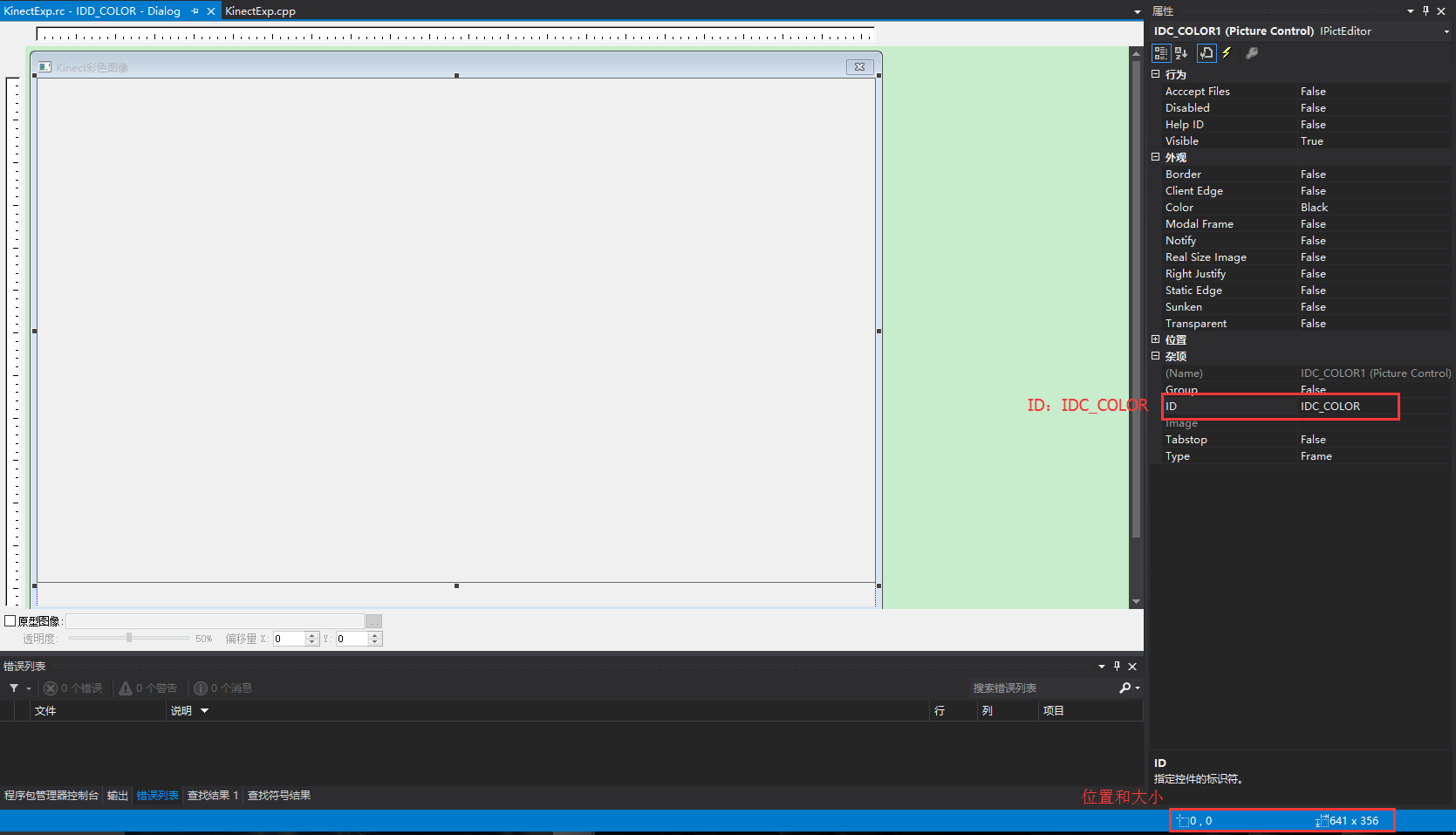


图 9

用同样的方式添加静态文本(Static Text)，ID设置为：IDC\_KinectColorStatus，将Sunken属性设置为：True，如下图所示：

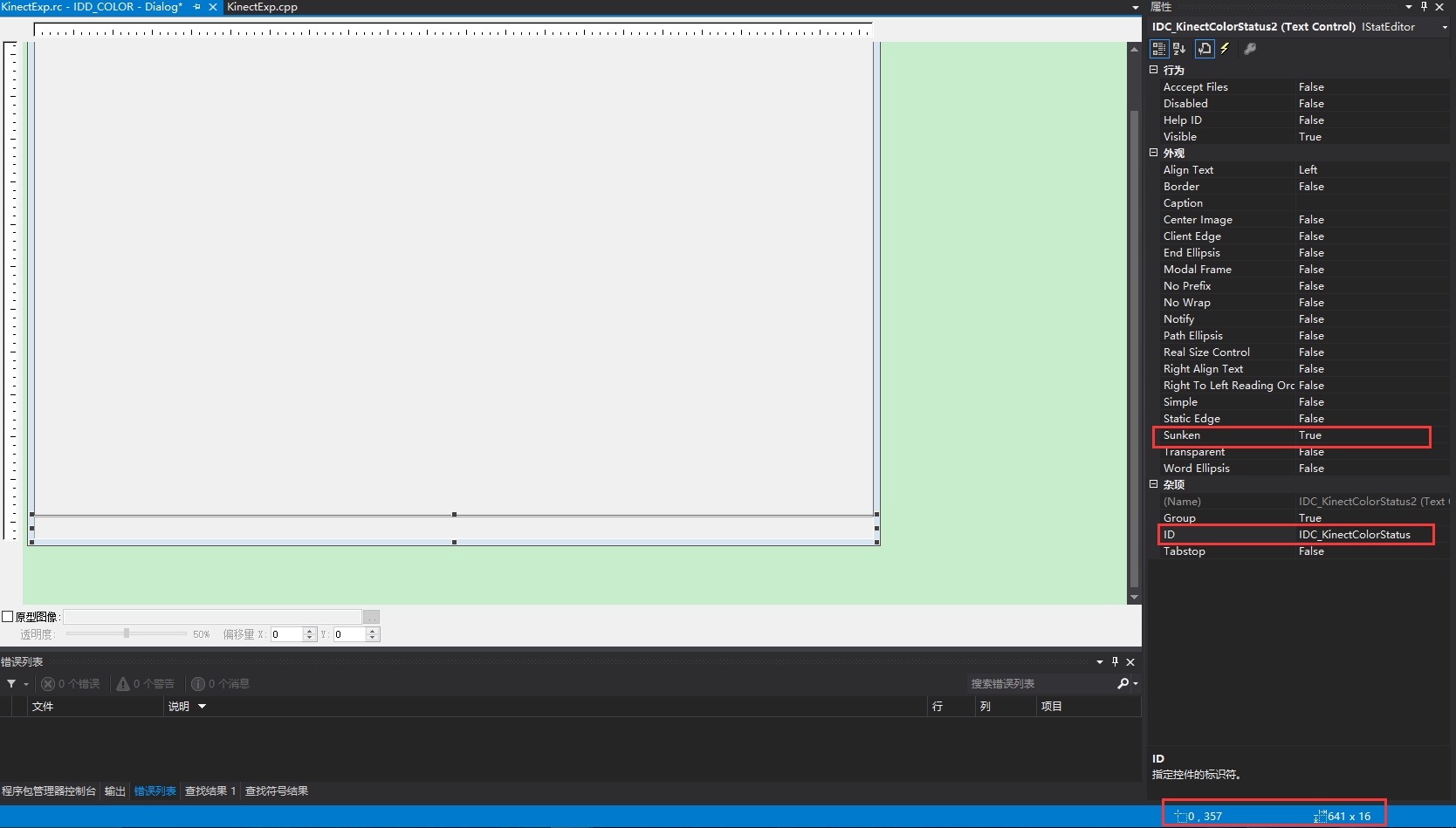


图 10

### 四、创建一个新类KinectColorBasics作为子窗口的载体

创建方法：鼠标右击解决方案，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KinectColorBasics，如下图所示。



图 11

**1．声明函数和变量**

在KinectColorBasics.h中添加KinectDevice.h头文件如下所示：

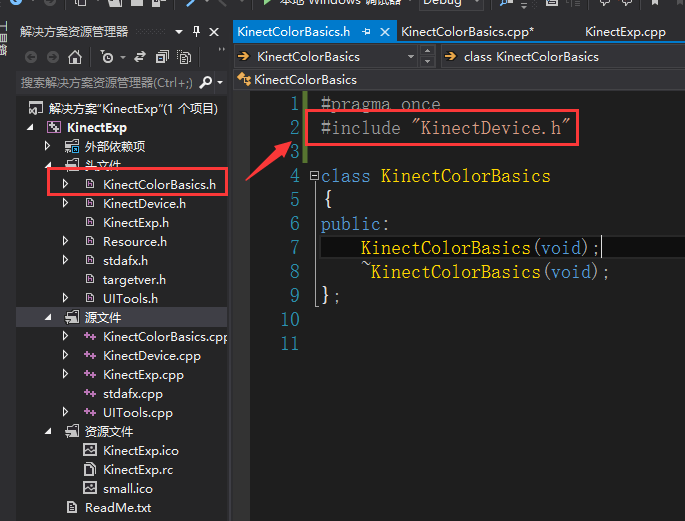


图 12

同时在KinectColorBasics.h中声明必要的函数和变量如下图所示：

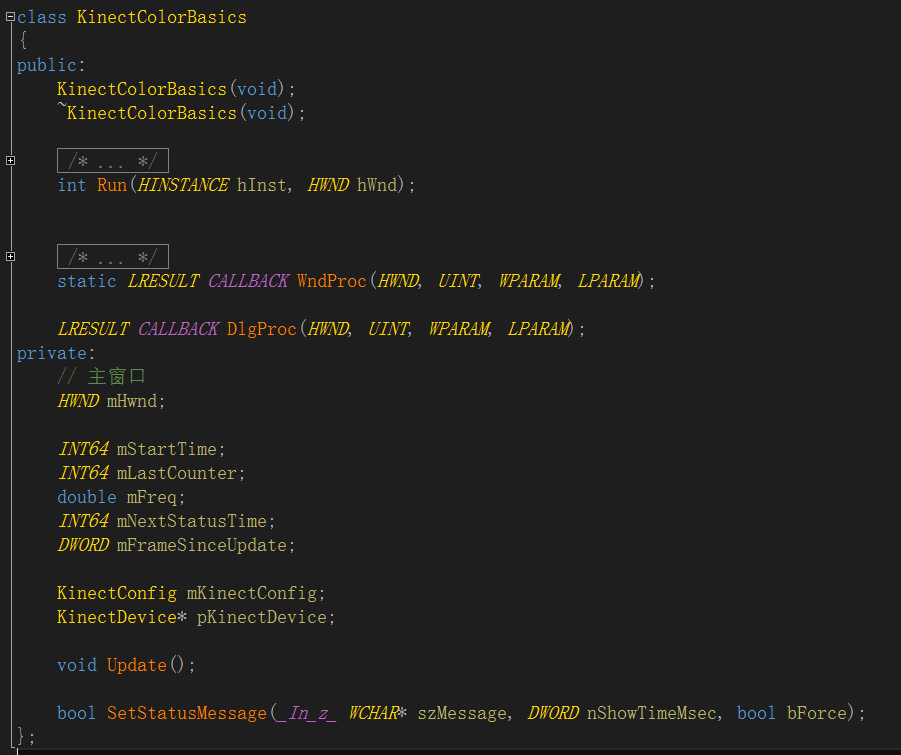


图 13

**2. 函数的实现**

在KinectColorBasics.cpp中，我们来实现声明的函数。首先在文档的开头引入几个头文件：#include “Resource.h”，#include “UITools.h”和#include <strsafe.h>。定义一个宏变量：

#define COLOR\_WINDOW\_CLASS\_NAME L"ColorDlgWndClass"

1） 初始化参数，在KinectColorBasics类的构造函数中添加如下的代码：

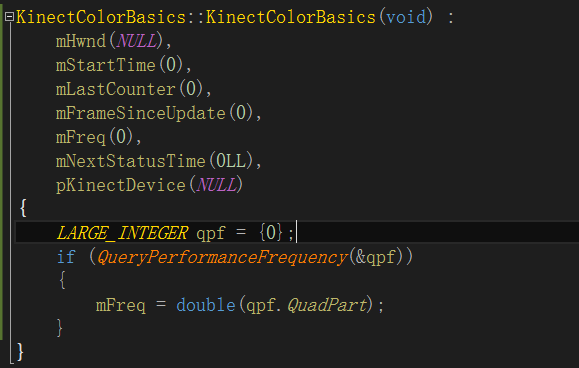


图 14

2）Run函数，主要工作是创建窗体并显示，代码如下：

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

**{**

EnableWindow**(**parent**,** FALSE**);**

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

WNDCLASS wc **=** UITools**::**CreateSubWindowClass**(**COLOR\_WINDOW\_CLASS\_NAME**,** hInst**);**

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

**{**

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

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

**return** 0**;**

**}**

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

**NULL,**

MAKEINTRESOURCE**(**IDD\_COLOR**),**

**NULL,**

**(**DLGPROC**)** KinectColorBasics**::**WndProc**,**

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

**);**

ScreenSize screenSize **=** UITools**::**GetScreenSize**();**

RECT region**;**

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

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

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

int x **=** **(**screenSize**.**width **-** w**)** **/** 2**;**

int y **=** **(**screenSize**.**height **-** 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**(**COLOR\_WINDOW\_CLASS\_NAME**,** hInst**);**

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

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

**return** 1**;**

**}**

3）WndPro函数负责窗体消息的分发，代码如下：

LRESULT CALLBACK KinectColorBasics**::**WndProc**(**HWND hWnd**,** UINT uMsg**,** WPARAM wParam**,** LPARAM lParam**)**

**{**

KinectColorBasics**\*** pThis **=** **NULL;**

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

**{**

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

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

**}**

**else**

**{**

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

**}**

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

**{**

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

**}**

**return** 0**;**

**}**

4）DlgProc函数负责窗体消息的处理，代码如下：

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

**{**

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

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

int mouseX**,** mouseY**;**

WCHAR mouseMsg**[**64**];**

RECT region**;**

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

**{**

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

**{**

**}**

**break;**

**case** WM\_MOUSEMOVE**:**

mouseX **=** LOWORD**(**lParam**);**

mouseY **=** HIWORD**(**lParam**);**

GetWindowRect**(**GetDlgItem**(**hWnd**,** IDC\_COLOR**),** **&**region**);**

MapWindowPoints**(NULL,** hWnd**,** **(**LPPOINT**)(&**region**),** 2**);**

mouseX **=** mouseX **-** region**.**left**;**

mouseY **=** mouseY **-** region**.**top**;**

**if** **((**mouseX **>=** 0 **&&** mouseX **<=** **(**region**.**right **-** region**.**left**))** **&&**

**(**mouseY **>=** 0 **&&** mouseY **<=** **(**region**.**bottom **-** region**.**top**)))**

**{**

StringCchPrintf**(**mouseMsg**,** \_countof**(**mouseMsg**),** L" x = %d y=%d"**,** mouseX**,** mouseY**);**

SetStatusMessage**(**mouseMsg**,** 2000**,** **true);**

**}**

**break;**

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

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

**break;**

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

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

**break;**

**default:**

**break;**

**}**

**return** FALSE**;**

**}**

5）SetStatusMessage函数显示状态信息，代码如下：

bool KinectColorBasics**::**SetStatusMessage**(**\_In\_z\_ WCHAR**\*** szMessage**,** DWORD nShowTimeMsec**,** bool bForce**)**

**{**

INT64 now **=** GetTickCount64**();**

**if** **(**mHwnd **&&** **(**bForce **||** **(**mNextStatusTime **<=** now**)))**

**{**

SetDlgItemText**(**mHwnd**,** IDC\_KinectColorStatus**,** szMessage**);**

mNextStatusTime **=** now **+** nShowTimeMsec**;**

**return** **true;**

**}**

**return** **false;**

**}**

### 五、采集并显示彩色图像

**1. 初始化和配置Kinect设备**

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

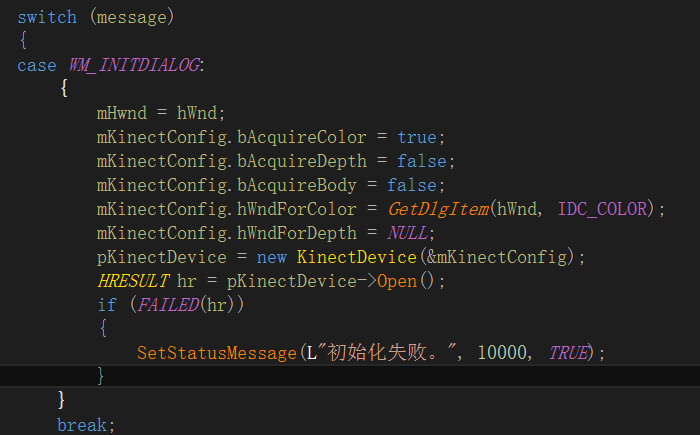


图 15

**2. 更新Kinect数据帧**

实现KinectColorBasics的Update函数，如下所示：

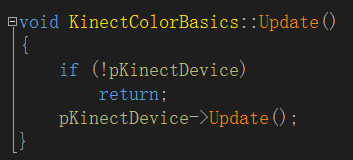


图 16

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

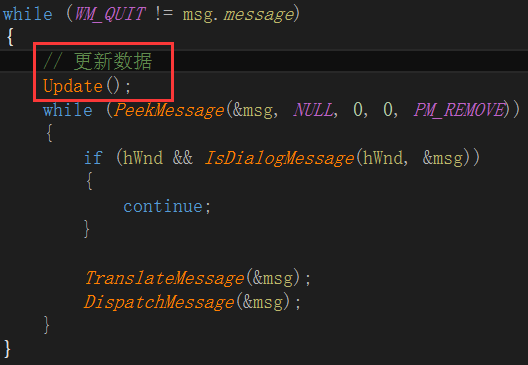


图 17

**3. 释放Kinect设备**

当我们不需要使用Kinect设备是应该关闭kinect并释放相应资源。在KinectColorBasics.cpp的析构函数~KinectColorBasics()中添加如下代码：

**if** **(**pKinectDevice**)**

**{**

pKinectDevice**->**Close**();**

**delete** pKinectDevice**;**

pKinectDevice **=** **NULL;**

**}**

### 六、跳转到Kinect彩色图像窗口

1）在KinectExp.cpp中引入头文件：#include “KinectColorBasics.h”。

2）在按钮的点击响应中添加跳转代码，在WndProc函数中添加如下代码：



图 18

### 七、创建Kinect深度图像子窗口

创建流程与步骤【创建Kinect彩色图像子窗口】一样。注意各个ID的命名区分。

### 八、创建一个新类KinectDepthBasics作为子窗口的载体

创建方法：鼠标右击解决方案，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KinectDepthBasics。

**1．声明函数和变量**

在KinectDepthBasics.h中添加KinectDevice.h头文件如下所示：

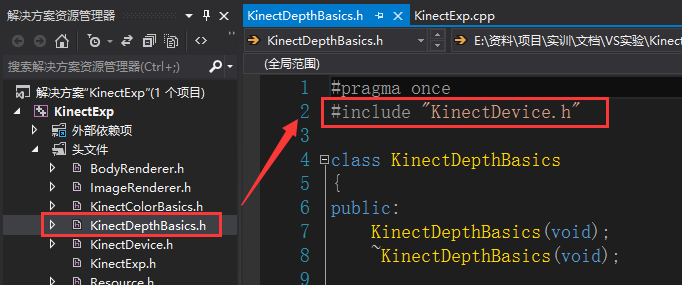


图19

同时在KinectDepthBasics.h中声明必要的函数和变量如下图所示：

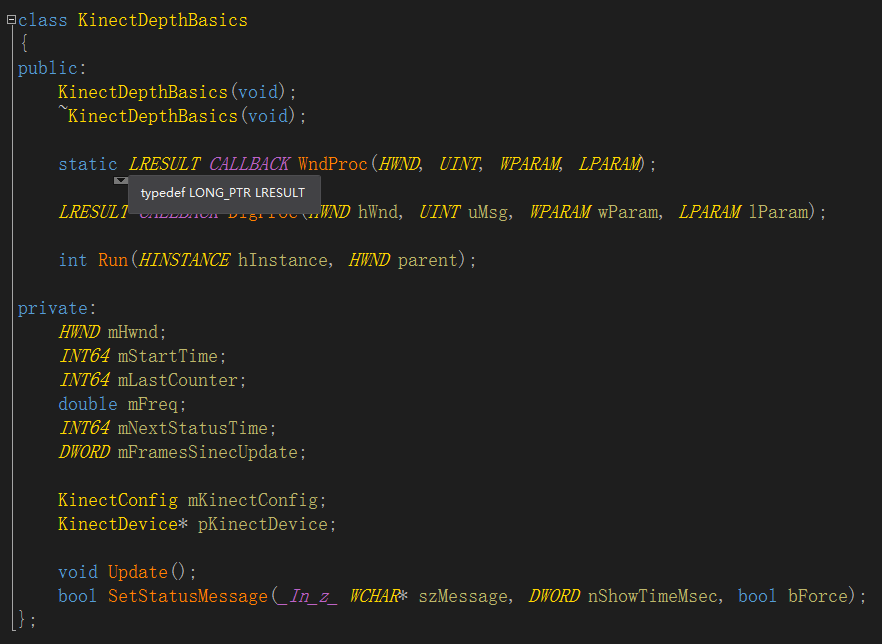


图20

**2. 函数的实现**

在KinectDepthBasics.cpp中，我们来实现声明的函数。首先在文档的开头引入几个头文件：#include “Resource.h”，#include “UITools.h”和#include <strsafe.h>。定义一个宏变量：

#define DEPTH\_WINDOW\_CLASS\_NAME L"DepthDlgWndClass"

1）初始化参数，在KinectDepthBasics类的构造函数中添加如下的代码：

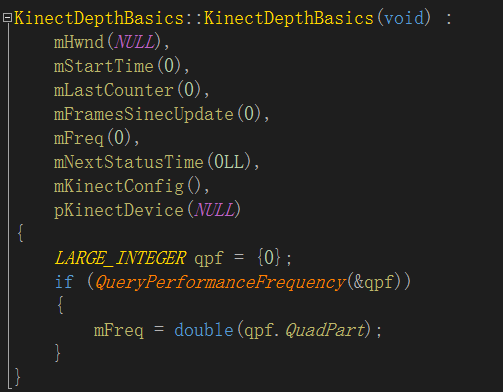


图21

2）Run函数，主要工作是创建窗体并显示，代码如下：

int KinectDepthBasics::Run(HINSTANCE hInstance, HWND parent)

{

EnableWindow(parent, FALSE);

MSG msg = {0};

WNDCLASS wc = UITools::CreateSubWindowClass(DEPTH\_WINDOW\_CLASS\_NAME, hInstance);

if (!RegisterClass(&wc))

{

EnableWindow(parent, TRUE);

SetForegroundWindow(parent);

return 0;

}

HWND hWnd = CreateDialogParam(

NULL,

MAKEINTRESOURCE(IDD\_DEPTH),

NULL,

(DLGPROC)KinectDepthBasics::WndProc,

reinterpret\_cast<LPARAM>(this)

);

ScreenSize screenSize = UITools::GetScreenSize();

RECT region;

GetWindowRect(hWnd, &region);

int w = region.right - region.left;

int h = region.bottom - region.top;

int x = (screenSize.width - w) / 2;

int y = (screenSize.height - h) / 2;

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

ShowWindow(hWnd,SW\_SHOW);

while (WM\_QUIT != msg.message)

{

Update();

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

{

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

{

continue;

}

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

UnregisterClass(DEPTH\_WINDOW\_CLASS\_NAME, hInstance);

EnableWindow(parent, TRUE);

SetForegroundWindow(parent);

return 1;

}

3）WndPro函数负责窗体消息的分发，代码如下：

LRESULT CALLBACK KinectDepthBasics::WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

KinectDepthBasics\* pThis = NULL;

if (WM\_INITDIALOG == uMsg)

{

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

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

}

else

{

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

}

if (pThis)

{

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

}

return 0;

}

4）DlgProc函数负责窗体消息的处理，代码如下：

LRESULT CALLBACK KinectDepthBasics::DlgProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

UNREFERENCED\_PARAMETER(wParam);

UNREFERENCED\_PARAMETER(lParam);

int mouseX, mouseY;

WCHAR mouseMsg[64];

RECT region;

switch (uMsg)

{

case WM\_INITDIALOG:

{

}

break;

case WM\_MOUSEMOVE:

mouseX = LOWORD(lParam);

mouseY = HIWORD(lParam);

GetWindowRect(GetDlgItem(hWnd, IDC\_KinectDepthStatus), &region);

MapWindowPoints(NULL, hWnd, (LPPOINT)(&region), 2);

mouseX = mouseX - region.left;

mouseY = mouseY - region.top;

if ((mouseX >= 0 && mouseX <= (region.right - region.left)) &&

(mouseY >= 0 && mouseY <= (region.bottom - region.top)))

{

StringCchPrintf(mouseMsg, \_countof(mouseMsg), L" x = %d y=%d", mouseX, mouseY);

SetStatusMessage(mouseMsg, 2000, true);

}

break;

case WM\_CLOSE:

DestroyWindow(hWnd);

break;

case WM\_DESTROY:

// Quit the main message pump

PostQuitMessage(0);

break;

default:

break;

}

return FALSE;

}

5）SetStatusMessage函数显示状态信息，代码如下：

bool KinectDepthBasics::SetStatusMessage(\_In\_z\_ WCHAR\* szMessage, DWORD nShowTimeMsec, bool bForce)

{

INT64 now = GetTickCount64();

if (mHwnd && (bForce || (mNextStatusTime <= now)))

{

SetDlgItemText(mHwnd, IDC\_KinectDepthStatus, szMessage);

mNextStatusTime = now + nShowTimeMsec;

return true;

}

return false;

}

### 九、采集并显示深度图像

**1. 初始化和配置Kinect设备**

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



图22

**2. 更新Kinect数据帧**

实现KinectDepthBasics的Update函数，如下所示：

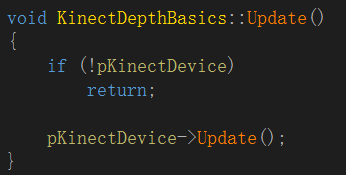


图23

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

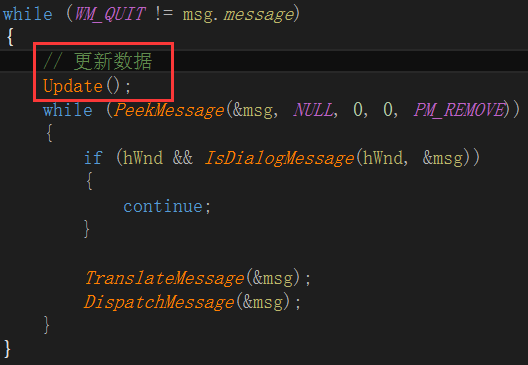


图24

**3. 释放kinect设备**

当我们不需要使用Kinect设备是应该关闭kinect并释放相应资源。在KinectDepthBasics.cpp的析构函数~KinectDepthBasics ()中添加如下代码:

if (pKinectDevice)

{

pKinectDevice->Close();

delete pKinectDevice;

pKinectDevice = NULL;

}

### 十、跳转到Kinect深度图像窗口

1）在KinectExp.cpp中引入头文件：#include “KinectDepthBasics.h”。

2）在按钮的点击响应中添加跳转代码，在WndProc函数中添加如下代码：

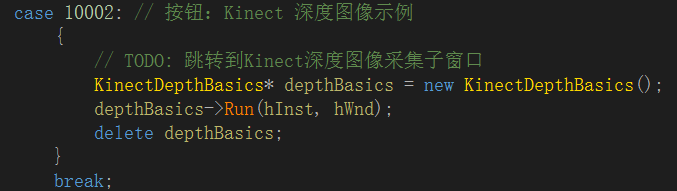


图25

### 十一、编译运行

点击进行编译，如果编译通过，可以看到如下界面：



图 26

点击界面上的按钮‘kinect 彩色图像示例’按钮，将跳转到下面的图像，图像中将显示kinect设备拍摄的图像：



图 27

点击界面上的按钮‘kinect 深度图像示例’按钮，将跳转到下面的图像，图像中将显示kinect设备拍摄的图像：

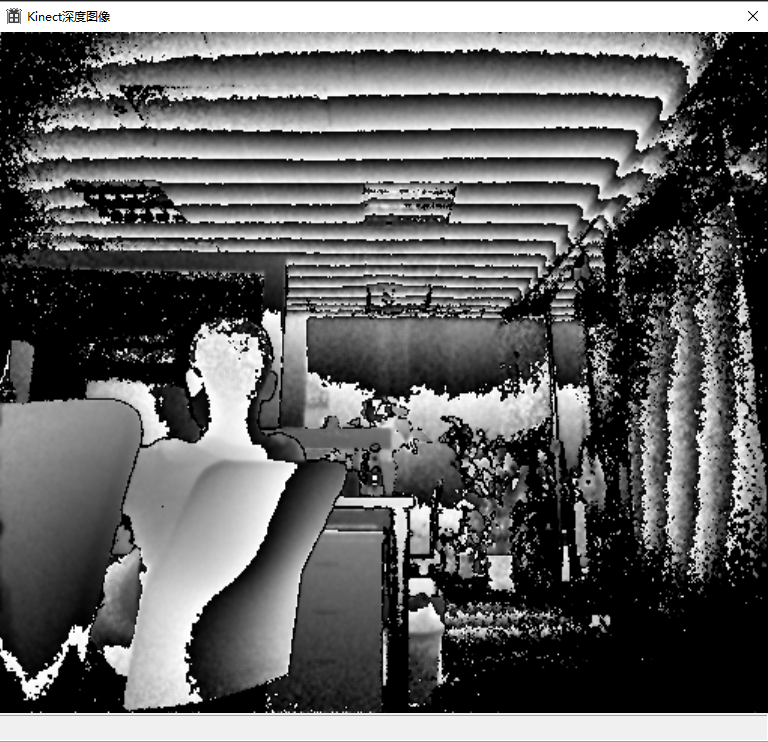


图 28

## 【思考题】

### 选择题

题目1：Kinect获取的彩色图像和深度图像的分辨率分别为：(C)

A. 1080 x 920；512 x 420

B. 1080 x 1920；512 x 424

C. 1920 x 1080；512 x 424

D. 1920 x 1080；512 x 420

题目2：在类定义的外部，可以被访问的成员有（C）

A. 所有类成员

B. private的类成员

C. public的类成员

D. public或private的类成员

### 2、简答题

简述 C/C++程序编译的内存分配情况 。

C/C++中内存分配方式可以分为三种：

（1）从静态存储区域分配：内存在程序编译时就已经分配好，这块内存在程序的整个运行期间都存在。速度快、不容易出错，因为有系统会善后。例如全局变量，static变量等。

（2）在栈上分配：在执行函数时，函数内局部变量的存储单元都在栈上创建，函数执行结束时这些存储单元自动被释放。栈内存分配运算内置于处理器的指令集中，效率很高，但是分配的内存容量有限。

（3）从堆上分配：即动态内存分配。程序在运行的时候用malloc或new申请任意大小的内存，程序员自己负责在何时用free或delete释放内存。动态内存的生存期由程序员决定，使用非常灵活。如果在堆上分配了空间，就有责任回收它，否则运行的程序会出现内存泄漏，另外频繁地分配和释放不同大小的堆空间将会产生 堆内碎块。

一个 C/C++程序编译时内存分为5大存储区：堆区、栈区、全局区、文字常量区、程序代码区。