Dual-Camera 360 Panorama Application

Introduction

Taking panoramic pictures has become a common scenario and is included in most smartphones' and tablets' native camera applications. Instagram alone has close to 1 million pictures tagged as panoramas, and flickr.com has over 1.2 million uploads tagged as panoramas. Traditionally the user will pan the device using a single camera to acquire images and the application will stitch the images together to create the panorama. Since most devices have both front and rear facing cameras, we could potentially utilize both cameras simultaneously to quickly capture a large panorama.

Current panorama applications on the market only support a ~180 to 270 degree maximum rotation, but using two cameras allows us to capture a full 360 degrees with only a 180 degree rotation of the device. There is a lot of value in decreasing the device rotation because it is very difficult to keep a phone or tablet steady when trying to rotate a large amount. Rotating only 180 degrees allows users to complete acquisition much faster by rotating the device in their hands without the need for a full body rotation. This will enable a more consistent and easy experience for the user.

First, I will go through a general overview of the implementation, then talk about challenges and our results. Please note: All sample software in this document is provided under the Intel Sample Software License. See Appendix A for details.

Implementation

In this section, we will discuss the steps necessary to capture images using both cameras and stitch them together to make a complete panorama picture. For reference, I will include code samples in C++ that utilize Microsoft DirectShow* APIs, but you can choose to develop with other APIs such as Microsoft Media Foundation.

First, we need to initialize the cameras and sensors. The method for doing so will depend on the APIs available for your target platform.

```
const int VIDEO_DEVICE_0 = 0; // zero based index of video capture device to use
const int VIDEO_DEVICE_1 = 1;

Capture frontCam = new capture(VIDEO_DEVICE_0);
Capture rearCam = new capture(VIDEO_DEVICE_1);

Gyrometer _gyrometer = new gyrometer();
Compass _compass = new compass();
```

Here we should also specify capture resolution. The images acquired from each camera should be the same resolution. You may also want to control other things, like exposure or autofocus.

Now we will create a function to capture and save images:

```
void acquireImages(int imageNumber)
{
          //save raw captures in memory
          _frontImage = frontCam.Click();
          _rearImage = rearCam.Click();

          //turn raw image into a readable format
          Bitmap front = new Bitmap(_frontImage);
          Bitmap rear = new Bitmap(_rearImage);

          //You may need to rotate the images based on your platform
          front.RotateFlip(RotateFlipType.RotateNoneFlipY);
          rear.RotateFlip(RotateFlipType.RotateNoneFlipY);

          //Save images to working directory
          front.Save("images/" + imageNumber + "_front.jpeg");
          rear.Save("images/" + imageNumber + "_rear.jpeg");
}
```

Next we create a function to acquire images. We tested multiple ways to implement acquisition: timer-based, gyro-based, and compass-based. Different platforms have different sensors available, which may determine what method you can use. In these samples I use NUM_IMAGES to denote the number of images we take with each camera. The number of images varies depending on the field of view of the platform's cameras. If you have too few images, the images won't have enough overlap and won't stitch together well. If you have too many images, you will have duplicates and processing time will be much higher than it needs to be. It takes experimentation to determine the ideal number of images you need to take.

Using a timer is a simple and reliable way to control capture and does not require any special sensors. It is, however, restricting to users since they must follow precise timing intervals for image capture.

Timer-Based Acquisition:

```
for (int currentImage = 0; currentImage < NUM_IMAGES; currentImage++ )
{
          acquireImages(currentImage);

          //specify interval between captures
          Thread.Sleep(750);
}
```

Using a gyroscope is another potential method; however, it can produce inconsistent results. It does allow the user to have control of the speed at which they capture. The gyroscope reports the angular frequency of the device. Since we want to know the angular position of the device, we can use this formula to get angular position:

*new angular position = angular position + (angular velocity * sampling interval)*

The angular position becomes more accurate as the sampling interval gets smaller. Unfortunately we can't sample at a high enough rate to maintain a perfectly accurate angular position, so acquisition intervals can be inconsistent when rotating at different speeds. This leads to inconsistent amounts of overlap on our captured images, which may cause stitching to fail.

Gyro-Based Acquisition:

```
position = 0;
while (currentImage < NUM_IMAGES)
{
	position += _gyrometer.GetCurrentReading() * GYRO_SAMPLE_INTERVAL;

	//capture image when position is at desired position +/- error
	if(position > ((currentImage*angleBetweenImages)-angleErrorTolerance) &&
		position < ((currentImage*angleBetweenImages)+angleErrorTolerance))
	{
			acquireImages(currentImage);
			currentImage++;
	}
	thread.sleep(GYRO_SAMPLE_INTERVAL);
}
```

We found the best method to use is the compass sensor. This method can capture images at very accurate intervals, which means our images will overlap the optimal amount every time. The compass is not available on all devices, however.

Compass-Based Acquisition:

```
while (currentImage < NUM_IMAGES)
{
	if (currentImage == 0)
	{
		//initialize position at first image capture
		startPos = _compass.GetCurrentReading().HeadingMagneticNorth;
	}

	position = startPos - _compass.GetCurrentReading().HeadingMagneticNorth;

	if (position < 0)
	{
		//forces value to be betweeen 0 and 360
		position = 360 + position;
	}

	//capture image when position is at desired position +/- error
	if(position > ((currentImage*angleBetweenImages)-angleErrorTolerance) &&
		position < ((currentImage*angleBetweenImages)+angleErrorTolerance))
	{
			acquireImages(currentImage);
			currentImage++;
	}
 }
```

After we have captured our images, we can stitch the images together. Because panorama stitching is a complex topic in itself, I can only give a high level overview. We used the stitching library in an open source project called OpenCV to do processing.

We will store our images in the “images” folder in our working directory. We can now load the images into our application and call the stitching function.

```
string[] imgs = Directory.GetFiles("images");
stitch(imgs, result);
result.save(“images/result.jpg”);
```

Assuming image stitching was successful, we now have our result panorama.

Challenges

Several issues arise when we tried to implement this idea on different platforms. We ran into issues with camera angle and unsupported camera features. There are workarounds for these issues, but there are some platforms that do not support simultaneous use of both cameras, which makes the application impossible to implement.

On some platforms the manufacturer decided to mount the front camera facing at an upward angle and the rear camera facing a downward angle with the intention of the cameras being used when the tablet is at an angle, similar to a laptop. This mismatch in angles between the front and rear cameras will make it so the acquired images do not overlap and cannot produce a quality panorama when used in landscape mode. The best workaround for this issue is to use the device in portrait orientation, which will make the vertical fields of view equal, which eliminates the issue.

This is an uncropped example of a landscape capture. You can see there is slight mismatch in the camera angles and leaves much of the image unusable after cropping.

Because of the mismatch, we must waste most of the image height during cropping. The original pictures were 1080px tall and the final panorama was 705px tall. We lost 35% of the vertical pixels.

This is an uncropped example of a portrait capture. You can see the images match up well.

Since the images match up well, we don't have to waste much of the height. The original images were 1920px tall and the result is 1640px tall. We only lost 14% of the height.

Even on the same platform, the front and back cameras often have different focal length, sensor size, and available capture modes. The maximum resolution of the resulting panorama is limited by the resolution of the smaller of the two cameras. The different focal lengths can create stitching issues if the difference is too large and also determine how many images need to be taken. A camera with a very wide field of view will be able to take fewer pictures and require less rotation than one with a narrow field of view. On different cameras, the manufacturer will allow different usage modes to be supported like "preview" for fast and low quality streaming, and "capture" for high quality but slow acquisition speed. We found that these features are not available on all platforms, so the application must be tested and modified for each target platform.

Conclusion

Utilizing dual cameras to capture large panoramas is a valuable and worthwhile concept that works well provided the correct hardware and driver support. On a platform with cameras directed perpendicular to the device and a compass sensor available, the application will work with little modification. On a platform with offset cameras and/or missing sensors, it may require additional work to get the application working. Due to the huge amount of variation between platforms, it is difficult to create a one-size-fits-all application, so we must develop and test the application on each target platform. Despite some potential implementation challenges, the concept improves greatly on current panorama capture applications and enables easier and faster image capture and better user experience.

Resources

Microsoft DirectShow API (Camera Interfacing/Streaming)
http://msdn.microsoft.com/en-us/library/windows/desktop/dd375454(v=vs.85).aspx
Microsoft Sensor API (Sensors)
http://msdn.microsoft.com/en-us/library/windows/desktop/dd318953(v=vs.85).aspx
OpenCV (Panorama Stitching)

http://opencv.org/

Appendix A: Intel Sample Software License

Intel Sample Source Code License Agreement

This license governs use of the accompanying software. By installing or copying all or any part of the software components in this package, you ("you" or "Licensee") agree to the terms of this agreement. Do not install or copy the software until you have carefully read and agreed to the following terms and conditions. If you do not agree to the terms of this agreement, promptly return the software to Intel Corporation ("Intel").

1. **Definitions:**

   A. "Materials" are defined as the software (including the Redistributables and Sample Source as defined herein), documentation, and other materials, including any updates and upgrade thereto, that are provided to you under this Agreement.

   B. "Redistributables" are the files listed in the "redist.txt" file that is included in the Materials or are otherwise clearly identified as redistributable files by Intel.

   C. "Sample Source" is the source code file(s) that: (i) demonstrate(s) certain functions for particular purposes; (ii) are identified as sample source code; and (iii) are provided hereunder in source code form.

   D. "Intel's Licensed Patent Claims" means those claims of Intel's patents that (a) are infringed by the Sample Source or Redistributables, alone and not in combination, in their unmodified form, as furnished by Intel to Licensee and (b) Intel has the right to license.

2. **License Grant:** Subject to all of the terms and conditions of this Agreement:

   A. Intel grants to you a non-exclusive, non-assignable, copyright license to use the Material for your internal development purposes only.

   B. Intel grants to you a non-exclusive, non-assignable copyright license to reproduce the Sample Source, prepare derivative works of the Sample Source and distribute the Sample Source or any derivative works thereof that you create, as part of the product or application you develop using the Materials.

   C. Intel grants to you a non-exclusive, non-assignable copyright license to distribute the Redistributables, or any portions thereof, as part of the product or application you develop using the Materials.

   D. Intel grants Licensee a non-transferable, non-exclusive, worldwide, non-sublicenseable license under Intel's Licensed Patent Claims to make, use, sell, and import the Sample Source and the Redistributables.

3. **Conditions and Limitations:**

   A. This license does not grant you any rights to use Intel's name, logo or trademarks.

   B. Title to the Materials and all copies thereof remain with Intel. The Materials are copyrighted and are protected by United States copyright laws. You will not remove any copyright notice

from the Materials. You agree to prevent any unauthorized copying of the Materials. Except as expressly provided herein, Intel does not grant any express or implied right to you under Intel patents, copyrights, trademarks, or trade secret information.

C. You may NOT: (i) use or copy the Materials except as provided in this Agreement; (ii) rent or lease the Materials to any third party; (iii) assign this Agreement or transfer the Materials without the express written consent of Intel; (iv) modify, adapt, or translate the Materials in whole or in part except as provided in this Agreement; (v) reverse engineer, decompile, or disassemble the Materials not provided to you in source code form; or (vii) distribute, sublicense or transfer the source code form of any components of the Materials and derivatives thereof to any third party except as provided in this Agreement.

D. Platform Limitation - The licenses granted in section 2 extend only to the software or derivative works that you create that run directly on a Microsoft Windows operating system product, Microsoft run-time technology (such as the .NET Framework or Silverlight), or Microsoft application platform (such as Microsoft Office or Microsoft Dynamics).

4. **No Warranty:**

THE MATERIALS ARE PROVIDED "AS IS". INTEL DISCLAIMS ALL EXPRESS OR IMPLIED WARRANTIES WITH RESPECT TO THEM, INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR ANY PARTICULAR PURPOSE.

5. LIMITATION OF LIABILITY: NEITHER INTEL NOR ITS SUPPLIERS SHALL BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE, EVEN IF INTEL HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

6. **USER SUBMISSIONS**: You agree that any material, information or other communication, including all data, images, sounds, text, and other things embodied therein, you transmit or post to an Intel website or provide to Intel under this Agreement will be considered non-confidential ("Communications"). Intel will have no confidentiality obligations with respect to the Communications. You agree that Intel and its designees will be free to copy, modify, create derivative works, publicly display, disclose, distribute, license and sublicense through multiple tiers of distribution and licensees, incorporate and otherwise use the Communications, including derivative works thereto, for any and all commercial or non-commercial purposes

7. **TERMINATION OF THIS LICENSE**: This Agreement becomes effective on the date you accept this Agreement and will continue until terminated as provided for in this Agreement. Intel may terminate this license at any time if you are in breach of any of its terms and conditions. Upon termination, you will immediately return to Intel or destroy the Materials and all copies thereof.

8. **U.S. GOVERNMENT RESTRICTED RIGHTS**: The Materials are provided with "RESTRICTED RIGHTS". Use, duplication or disclosure by the Government is subject to restrictions set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor. Use of the Materials by the Government constitutes acknowledgment of Intel's rights in them.

9. **APPLICABLE LAWS**: Any claim arising under or relating to this Agreement shall be governed by the internal substantive laws of the State of Delaware, without regard to principles of conflict of laws. You may not export the Materials in violation of applicable export laws.

Notices