



The **Software Development Kit (SDK)** is a comprehensive set of development tools including libraries, complete and user-friendly documentation, and code samples that allow a software engineer to integrate the ImmerVision library (.dll for Windows, .so for Linux, Framework for Mac OS X).

This library allows users to visualize, without distortion, panoramic videos captured by **standard CCTV, analog or IP cameras equipped with the ImmerVision *Enables* panomorph lens** and to navigate within these videos using digital Pan-Tilt-Zoom functionality in both live and playback modes.

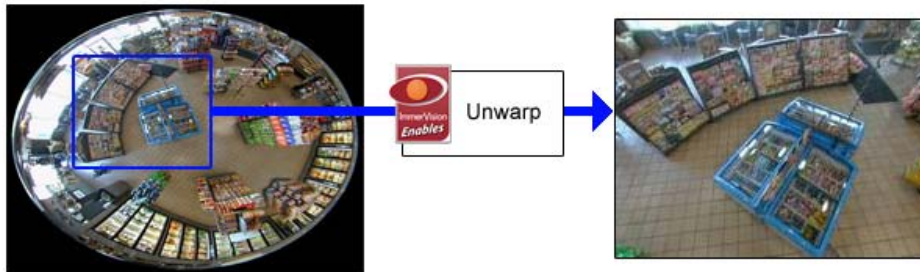


Figure 1: Unwarping of a Panomorph Video

Integration Overview

The library, **independent** of other components, was designed to be **non-invasive** when incorporated in standard CCTV systems. The library is **not a camera SDK** but an image processing library dedicated to panomorph

picture processing. It simply needs to be integrated in the viewing software (client or viewer application) at the end of the video processing stream.

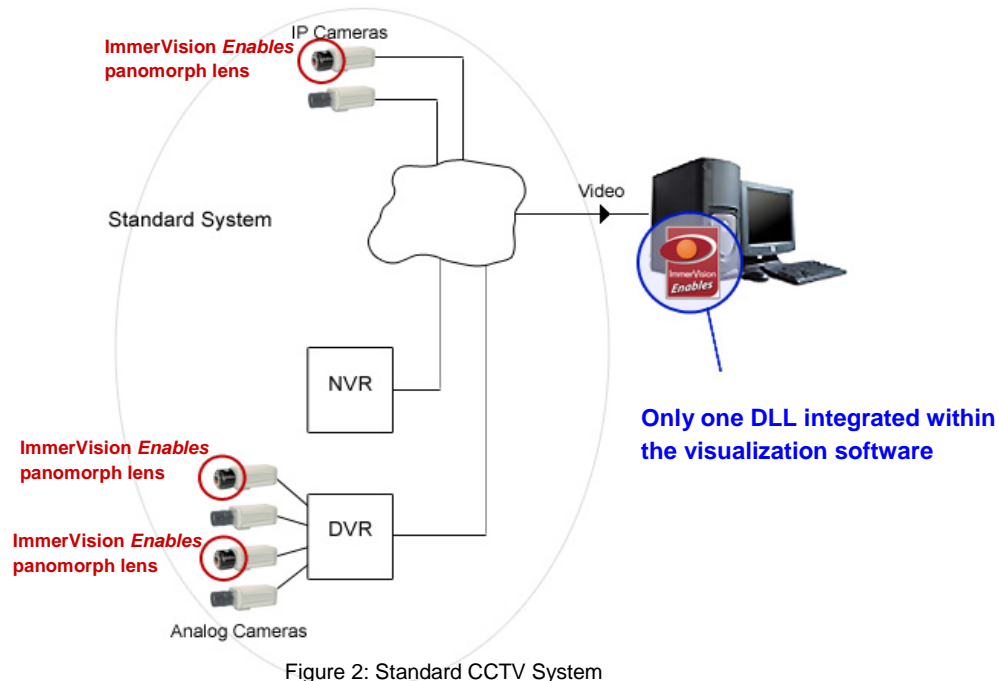


Figure 2: Standard CCTV System

The viewing software displays videos from any standard CCTV source (e.g. IP, analog camera, DVR, capture card protocols). Like other standard video processes (e.g. resize, brightness, contrast), the IMV1 library processes

uncompressed video frames captured by cameras equipped with ImmerVision *Enables* panomorph lenses to generate an uncompressed unwarped picture (undistorted view).

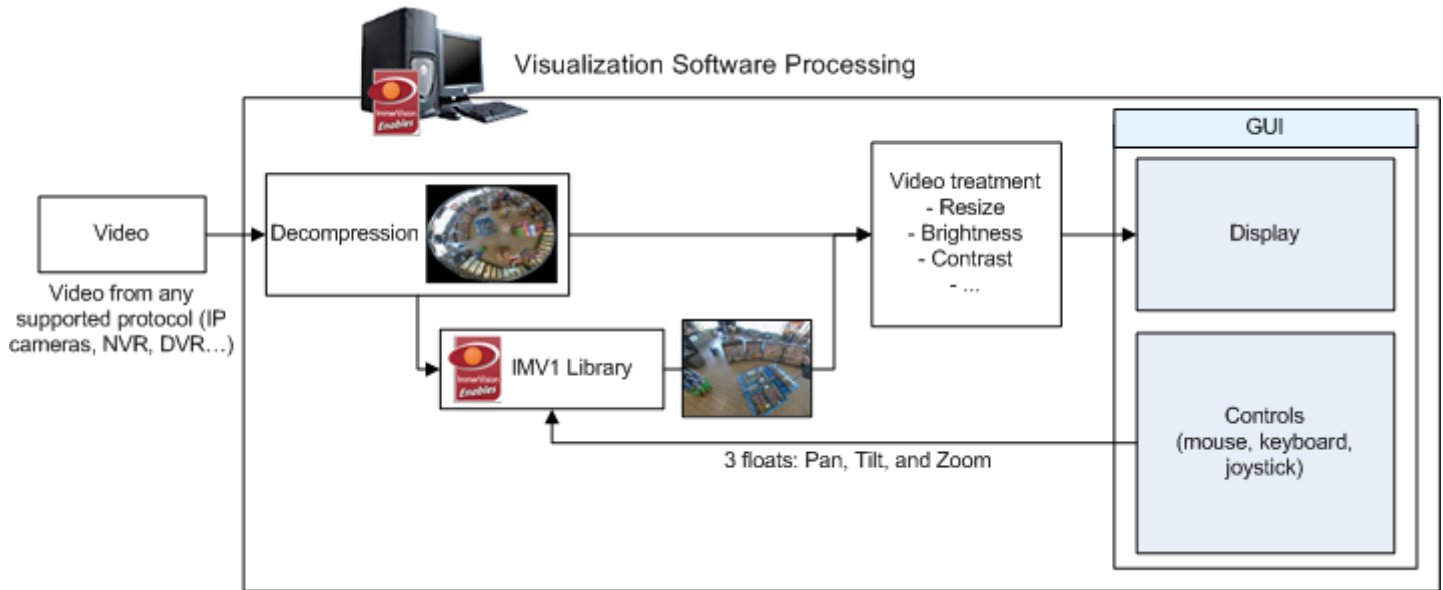


Figure 3: Visualization Software Processing Diagram

The IMV1 Library uses an **Automatic Calibration System (ACS)** allowing DVRs, NVRs and VMS to **detect and calibrate** the panomorph lens, and ensuring easy installation and trouble-free use. The library then enables panoramic viewing functionalities. Only the

panomorph lens type (**RPL**) needs to be defined.

When a standard video (non panomorph) is detected, the library won't process the frames; therefore, the video source is displayed as usual, without any panoramic functionality.

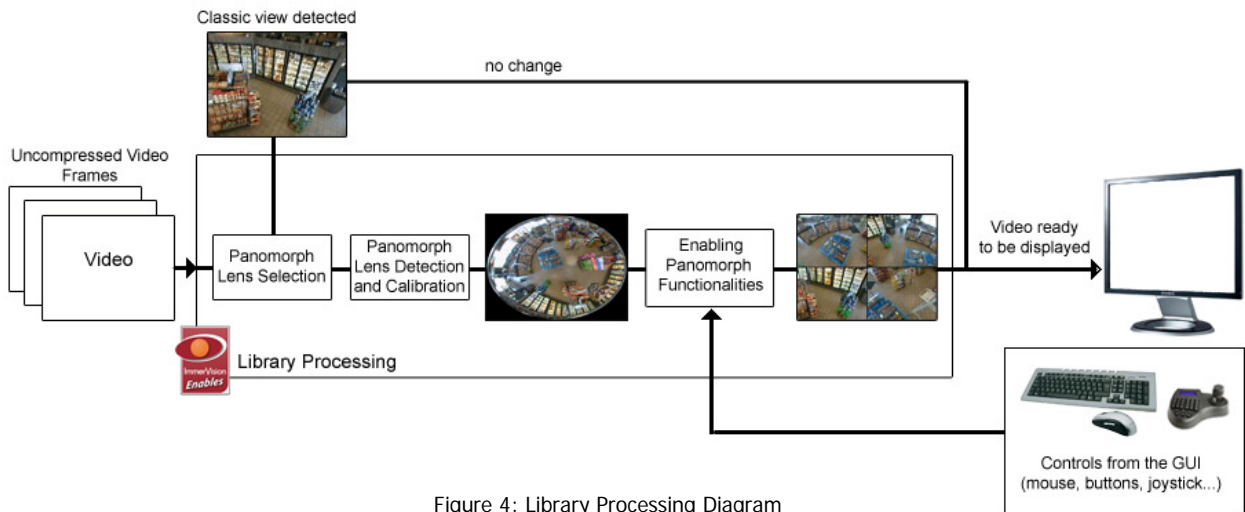

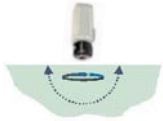


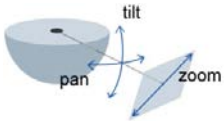

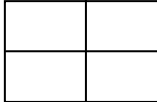






Figure 4: Library Processing Diagram

IMV1 Library Input / Output Parameters

Input: Lens RPL	Define the panomorph lens type to unwarp (five character string ex: A0**V)
Input: Panomorph video	Decompressed video (picture buffers) in supported format ⁽¹⁾ (minimal resolution: 640x480) 
Input: Camera position	Ceiling, Ground or Wall perspective defines the camera orientation <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">  <p>Ceiling perspective</p> </div> <div style="text-align: center;">  <p>Ground perspective</p> </div> <div style="text-align: center;">  <p>Wall perspective</p> </div> </div>
Input: View position	Pan, Tilt and Zoom angles allow you to navigate within the immersive view 
Input: View type	<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">  <p>PTZ View enables 360° navigation on one channel screen or external window.</p> </div> <div style="text-align: center;">  <p>Quad View enables 360° navigation within four independent PTZ views on one channel screen or external window.</p> </div> <div style="text-align: center;">  <p>Perimeter View displays the 360° perimeter viewing of two 180° strips in Ceiling and Ground perspective or of one 180° strip in Wall perspective on one channel screen or external window.</p> </div> </div>
Output: Unwarped video	Decompressed video (picture buffers) calculated by the library. Note that the output format is the same as the input format ⁽¹⁾ . <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">  <p>PTZ View</p> </div> <div style="text-align: center;">  <p>Quad View</p> </div> <div style="text-align: center;">  <p>Perimeter View</p> </div> </div>

(1) Supported formats: RGB (16, 24 or 32 bits) and YUV (UYVY, YUYV, YVYU, YUY2, YV12, IYUV, I420 or YV12).

Simplified Code Example

This is a simplified code example; all definitions and complete samples are explained in the SDK Programmer Guide.

Your visualization software receives compressed videos from DVR, NVR, IP Cameras, etc.

Your visualization software uncompresses the videos in order to process the frames

After uncompressing your video frames, you should have a buffer representing the picture data



If you display it in your visualization software GUI, you will observe the original picture

Get the complete list of panomorph lens types included in the library (name and RPL)

```
const SLensDescription* lenses =  
IMV_CameraInterface::StaticGetLensDescription();  
  
char* selectedRPL = lenses[selectedItem].RPL;
```

Instantiate an IMV_CameraInterface Object. The library's functions are accessible through this object method

```
IMV_CameraInterface *camera = new IMV_CameraInterface();
```

Set the panomorph lens type (RPL)

```
camera->SetLens(selectedRPL);
```

Check if the video is panomorph. If not, you can close the library and process the video for standard viewing.

```
camera->CheckCameraType(inBuffer, //videoFrameData  
                        videoFormat); //Supported formats  
  
//return true or false
```

In order to unwarped the video stream, you must initialize the library with the SetVideoParams(...) method. As soon as the library is initialized, you can use all the other methods to process the video.

```
camera->SetVideoParams( inBuffer, //videoFrameData  
                       outBuffer, //videoFrameResult  
                       ColorFormat, //ColorFormat  
                       ViewType //ptz, quad, perimeter  
                       CameraPosition); //ceiling, wall, ground
```



In this example, the camera was on the ceiling; CameraPosition was set to Ceiling, and we unwarped the image in a virtual PTZ view:

To process the input video and feed the output buffer with the result, simply call up the Update() method. Each time a new video frame is ready to be displayed, just call up the Update() method again to update the output buffer.



In the GUI, your controls (mouse, keyboard) will be able to change the viewing position of the virtual camera.



Some other methods are available to change the buffer parameters: view mode, camera position, and so on.

(See the SDK Programmer Guide)

To close the library, simply delete the IMV_CameraInterface you have created.

```
//Get a new video frame
inBuffer->data=newVideoFrameData;//pointer to the new video frame
camera->Update();//updates outBuffer->data with
//the new rendering
```

Now the outBuffer->data is ready to be displayed in your visualization software.

```
// Increment/decrement pan depending on X-axis mouse movements.
// Increment/decrement tilt depending on Y-axis mouse movements.
// Increment is relative
// Set the new position
camera->SetPosition( &panIncrement, &tiltIncrement,
zoomIncrement);
...
//update the output buffer
camera->Update();
```

Display the new outBuffer->data to see the virtual PTZ view in the new position.

```
delete camera;
```

Advanced Functionalities: ImmerVision *Enables* SDK Extended Version

ImmerVision *Enables*[®] advanced functionalities allow developers to add different features such as:

- **Privacy zone masking**



Figure 5: Panomorph video



Figure 6: Panomorph video with privacy masking



Figure 7: Immersive view with privacy masking

- **Merge a video stream from a secondary standard or motorized camera into 360°**



Figure 8: Panomorph video



Figure 9: Standard video



Figure 10: Standard video merged into a panomorph view

- **Target localization in the immersive view or in the real-world referential**

Convert 2D, 3D (real-world referential) and angular position from one referential to another (panomorph image, immersive viewer referential, 2D floor plan coordinates, and real-world referential for target tracking)



Figure 11: Target on 2D Panomorph picture



Figure 12: Target on immersive view

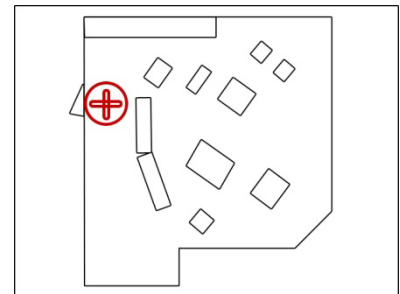


Figure 13: Target on floor plan

SDK Content

- Library (DLL for Windows™ 32 and 64 bits, SO for Linux, Framework for Mac OS X) to be included with the executable of the host application;
- Samples (code and documentation for C/C++, VB6, C# and Java);
- API to integrate the library in your IDE (Visual Studio, Eclipse, Xcode, KDevelop);
- API reference documentation.

System Requirements

Windows:

Operating System: Microsoft® Windows® 2000/XP/Vista or 7, 32 and 64 bits.
Applications: Microsoft® Visual® C++ 6 up to .NET

Linux PC:

Operating System: Linux PC with glibc 2.3.6
Applications: KDevelop and QT to compile the samples

Mac OS X:

Operating System: Apple™ Mac OS X 10.4 and 10.5 Intel™ based
Applications: Xcode to compile the sample