Intel(R) Perceptual Computing SDK Demo Application

Head Coupled Perspective

Table of Contents

1 Legal Disclaimer

1.1 Optimization Notice

2 Introduction

The Intel® Perceptual Computing SDK is a library of pattern detection and recognition algorithm implementations exposed through standardized interfaces. The library aims at lowering barriers to using these algorithms and shifting the application developers' focus from coding the algorithm details to innovating on the usage of these algorithms for next generation human computer experience.

This document describes the `HeadCoupledPerspective` demo application.

Document Convention

The SDK document uses the Calibri typeface for normal prose.

With the exception of section headings, captions and the table of contents, all code-related items appear in the Courier New typeface (`pxcStatus`).

Hyperlinks appear underlined in blue, such as pxcStatus.

This is a note that provides additional information to aid your understanding of the procedure or concept.

This is a tip that provides alternate methods or shortcuts.

This is a result statement which indicates what you can expect to see or happen after performing a step.

3 Head Coupled Perspective

Head-Coupled Perspective (HCP) refers to a technique of rendering a scene that takes into account the position of the viewer relative to the display. As a viewer moves in front of the display, the scene's projection is adjusted to match, creating the impression of the display as a window onto a scene existing behind it. The viewer's head motion controls the scene intuitively; leaning left reveals more of the scene to the right, ducking reveals more of the scene above, etc. As the viewer leans in, features in the scene remain a constant size (measured in view angle), but a wider field of view becomes visible.

The necessary input data for creating a HCP display is knowledge of the viewer's eye position relative to the physical display. This sample utilizes a feature of Intel® Perceptual Computing SDK to track the viewer's face location using a user-facing webcam, or the built-in camera of an Ultrabook™. The sample scene is controlled using a standard mouse/keyboard camera control, plus position and zoom offsets derived from the stream of face detection data provided by the SDK.

The sample includes two scenes that illustrate potential uses of HCP. The first is an indoor scene (see Figure 1), with objects stretching from the near- to middle-distance. This scene is representative of a first-person shooter style game, with foreground objects that can occlude more distant objects. The user moves about the scene with mouse and keyboard. When the HCP camera is enabled, the user can move their head to look over, under, or around a foreground object without changing their in-game position. This view mechanic might be used in a game to peek out from behind cover, or around a corner.

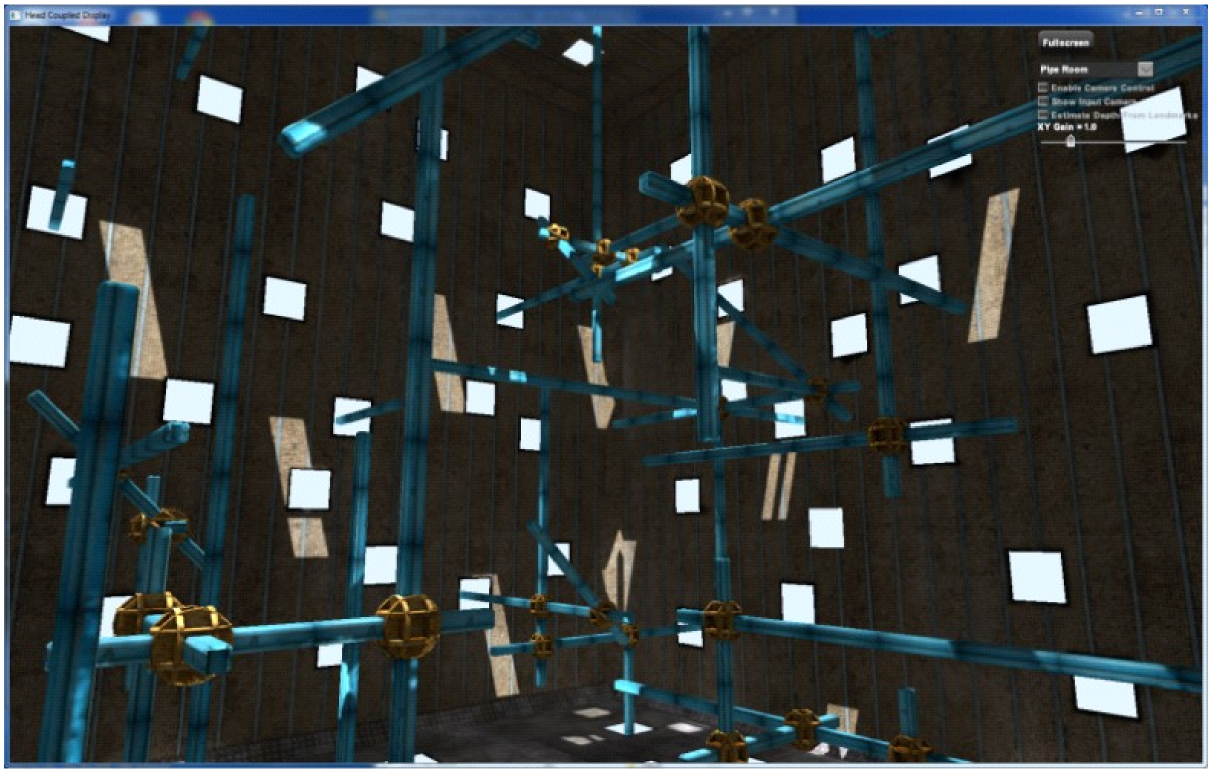

Figure 1: Pipe Room Scene

The second scene is an outdoor mountain scene with a fixed camera position, as illustrated in Figure 2. This scene represents a fixed window onto a rendered scene, and the user interacts just by moving relative to the display. This mode might be used to represent a view from an airplane window, or from a box seat in a stadium. Another application would be as a retail kiosk or trade show display, where passerby would see the rendered scene shift as they walk by.

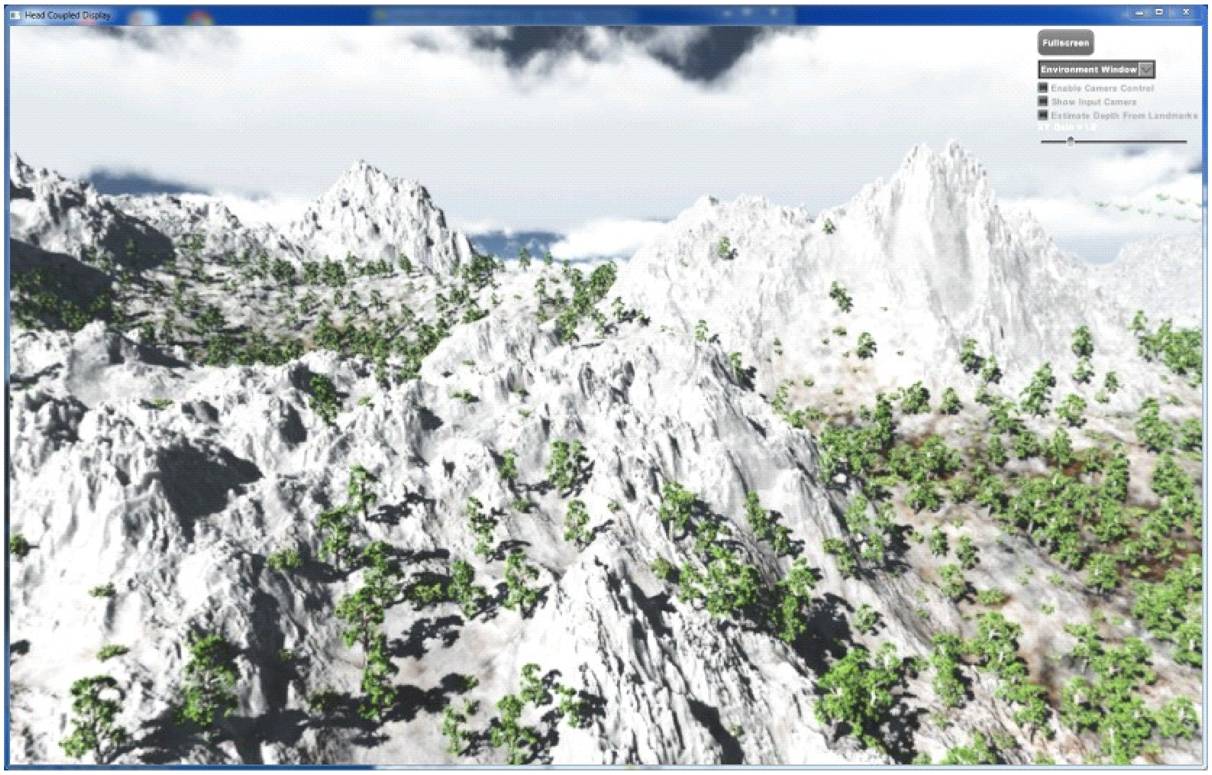

Figure 2: Environment Window Scene

3.1 Operation Instructions

Run the prebuilt executable: `HeadCoupledDisplay.exe`.

There are a few rerequisits:

- The executable must be placed at the same level as the `Media` directory as the sample will need to locate the directory for data access.
- The sample requires Microsoft* DirectX* End-User Runtimes (June 2010). If the system reports that `D3DCompiler_43.dll` is missing, please reinstall the DirectX runtime.

By default, the sample shows the pipe room scene as illustrated in Figure 1. The user can move the scene with mouse movements, or use the drop down menu to switch between the pipe room scene and the environment window scene, as in Figure 2.

The user can check the `Enable Camera Control` box to enable perspective change based on view angles. The scene will move according to head positions.

3.2 Build Instructions

The application comes with source code. The solution file is `HeadCoupledDisplay.sln`. The sample requires the following building software requirements:

- Microsoft Visual Studio* 2010 with SP1, or Microsoft Visual Studio 2012, or Intel® C++ Compiler version 11
- Microsoft DirectX SDK (June 2010)

3.3 Design Points

This section describes the application design that is related to Intel® Perceptual Computing SDK and is not meant to be a code workthrough of all application designs.

The location of the sample viewer is obtained from the SDK, using a color camera sensor. Facial data obtained from the SDK consists of image size, face position, face size, and position of six facial landmarks in (X,Y) coordinates. The scene's rendering (virtual) camera is implemented as two coupled cameras, initially aligned and with the second set slightly behind the first (i.e. offset along the first camera's Z axis.) Mouse and keyboard inputs are handled by the first camera, and head-coupled offsets are applied to the move second camera about the shared axis. The scene is then rendered from the second camera's point of view.

When facial input is enabled, the second camera is offset in X, Y (using the first camera's frame of reference) proportionally to the calculated offset of the face from the center of the captured image. The second camera's 'look-at' point remains coincident with the first's, so the position offset translates into a yaw/tilt effect on the scene. The sample initially assumes that the image acquisition camera has a horizontal field of view similar to that of the synthetic rendering camera, about 60°. If this assumption is not true, the scene will appear to yaw too little or too much, so an 'XY Gain' slider allows the user to adjust to suit.

The sample includes the option to zoom in/out of the scene corresponding to how near the detected face is to the camera. This effect is intended to mimic the effect of a scene existing at a distance behind the display. In a normal, non-HCP display the scene is unchanged as the viewer gets closer, and individual features grow larger. In the real world as one approaches a window with a distant scene behind it, the scene's features remain nearly unchanged in size, but the overall field of view expands. When the sample's zoom effect is enabled, an estimate is made of how close the face is to the display and the rendered field of view is adjusted to match. The proximity estimate is based both on the size of the face rectangle provided by the SDK, supplemented with a set of measurements of the distance between selected pairs of facial landmarks. This data, landmarks especially, is particularly noisy unless in very optimal lighting conditions, so a heavy smoothing is applied to reduce the resulting jitter in zoom values.

3.3.1 Face Tracking with the SDK

The Intel® Perceptual Computing SDK provides a wide range of sensor data and control inputs from camera, microphone, and other hardware sensors. Within the camera sensor subset, the SDK provides face and hand/gesture data. Within the facial data set are number, position, size, facial landmarks, individual face recognition and classification of age, gender, and expression. From this available data, the HCP sample utilizes just the position and size of a single face, and location of facial landmarks within that face.

The SDK provides a COM-like API that allows fine-grain control over all aspects of capture and processing of camera images. It also supplies a UtilPipeline class that abstracts away much of the complexity and provides a simple, asynchronous callback interface applicable to many common uses; this sample utilizes the `UtilPipeline` class.

When the `UtilPipeline`-derived capture class is initialized a separate thread is spawned to capture camera images. Within the rendering loop a non-blocking check is made for new image data, and when available the `OnNewFrame()` method is called to extract facial data. The new facial data is merged with the existing data using linear interpolation, based on the elapsed time since the last detection; the more time elapsed, the more weight given to the new data. After the new face data (if any) has been incorporated, the scene camera is updated to reflect the face position. A linear smoothing is again applied during the camera position calculation, to minimize position stutter in frames where new facial data has been added vs. those without new data.

3.4 Limitations

This sample is intended as a proof of concept of how HCP might be integrated into games or kiosks, and to spur thought among game developers. Responsiveness is quite dependent on environmental lighting and the particular camera sensor in use, and it is reasonable to expect improved HCP latency as the SDK and the available sensors mature.

The sample's coupled camera setup currently implements only pointing angle and field of view adjustments. This leaves out adjustment of the projection matrix to compensate for the off-center view of the display. Consider two equally-sized (in screen pixels) objects at either side of the screen. When the viewer is positioned nearer to one side of the screen, the object at the closer edge appears larger to the viewer than the one at the far edge, and the display outline becomes trapezoidal. To compensate, the projection should be transformed with a shear to maintain the apparent size of the two objects.