CRDT

The Dining Philosophers

June 1, 2017

Contents

Preface

This document contains the combined SRS - SDD - Test document for the "Real-Time HDR Video Capture and Display Pipeline" project. The following sections include three major parts. (1) System Requirements, (2) Design and (3) Testing.

Introduction

This document is a combined SRS - SDD - Test document for the real-time HDR video capture and display pipeline system. In this document, system requirements, design and testing parts of the project will be explained, respectively.

2.1 Problem Definition

Conventional image and video acquisition methods cannot capture the dynamic range of real world scenes. Such devices will output content with missing information such as saturated or completely dark pixels in bright and dark regions of the scene, respectively. High dynamic range imaging aims to overcome these limitations and capture the full content in a scene. HDR imaging poses mainly two challenges: acquiring and displaying HDR content.

There are several existing commercial products that operate on HDR content. However, these hardware are often inaccessible by researchers or end-users due to their extremely high price tag.

2.2 Purpose

Our project aims to implement a real time HDR imaging pipeline (capture and display) using off the shelf products such as regular cameras and LCD displays, which will arguably cost only a fraction of a dedicated hardware solution.

The fundamental problem our project aims to solve is the cost and availability barrier of real time high definition HDR content acquisition and displaying.

System Requirements

In this part of the document short information about initial and revised system requirements will be provided. All the requirements will be explained briefly as follows.

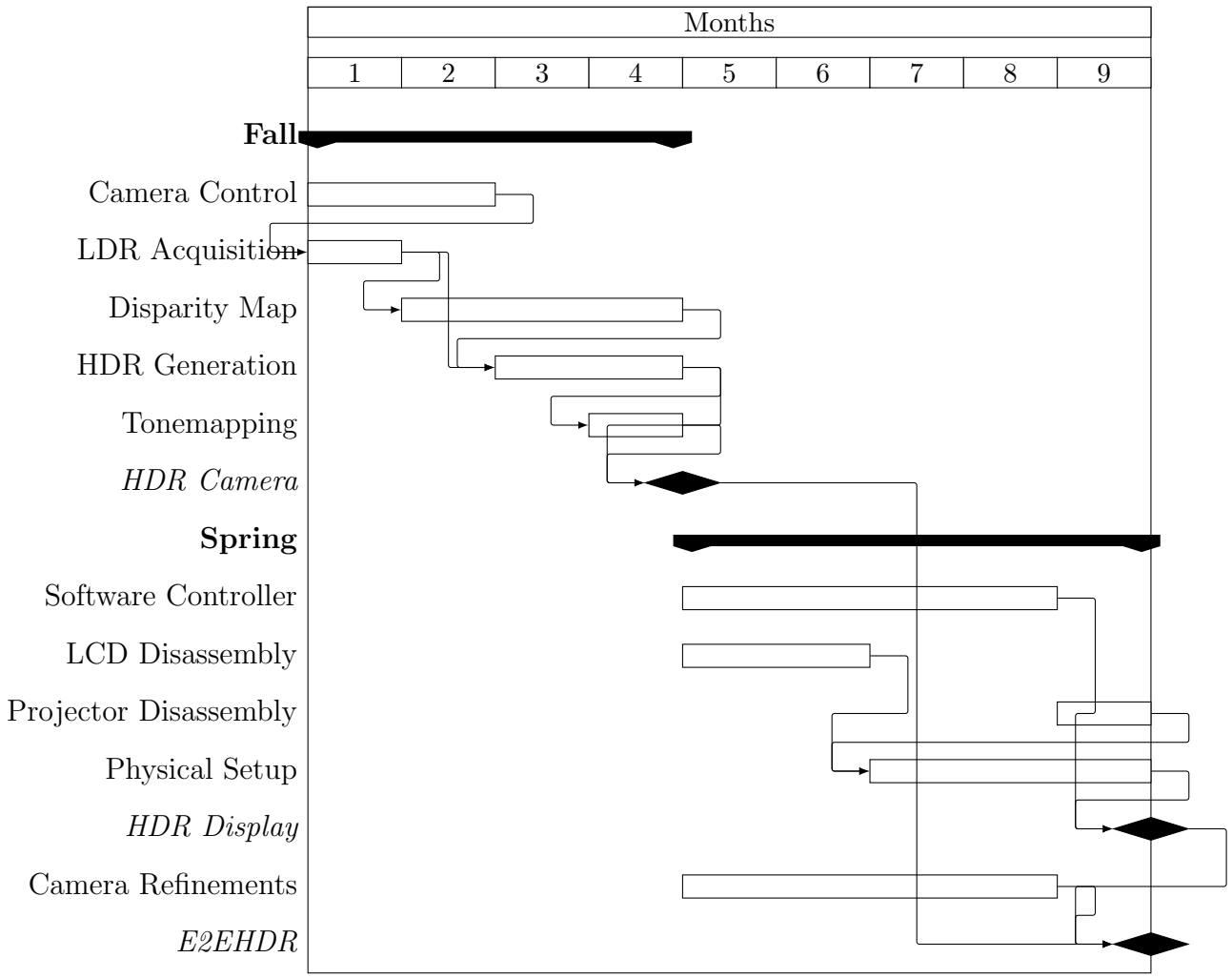
3.1 System Requirements

- Control Cameras : User can control the shutter speed, manual focus and ISO using OpenGL GUI to be able to obtain alternating scenes.
- Acquire and Sync LDR Video Stream From Cameras: User can acquire LDR Video stream from cameras which are working on different exposure settings.
- LDR Image Registration in GPU: Image registration can be done in GPU while frames are acquiered from different cameras. Comparative testing of algorithms should be done.
- Camera Response Curve Recovery: User can recover offline camera response curve for each camera.
- OpenGL GUI Control: User can display parameters, change parameters like shutter speed, exposure, ISO values using OpenGL GUI.
- Generate HDR Frames: HDR frames are generated while acquiring LDR frames from cameras.
- Real Time Video Tone Mapping: Before completing display set up, real time video tone mapping should be done to be able to see HDR frames.
- Breakdown HDR Frames into Projector and LCD Streams: Luminance map will be extracted to the projector and chromoticity map will be extracted to LCD streams.

- Postprocess the streams: User can smooth the luminance map for backgroung also sharpen the chromaticity map for foreground. Postprocessing the streams actully should be done while generating HDR frames.
- HDR Calibration & Display Set up: HDR Calibration and Display Set up should be done to be able to see HDR frames generated by cameras.

3.2 Project Plan

Our project plan over time can be summarized as this:



3.3 Use Case Diagrams

This section includes use case diagrams of the E2EHDR project. Since some of the operations will be done directly on GPU, in this section only user related use cases will be explained.

Use Case ID	UC1
Use Case Name	Start Cameras
Included Use Case(s)	None
Primary Actor(s)	User
Description	User can start the multiple cameras by using OpenGL GUI.
Precondition	None

Use Case Diagram 1: Start Cameras

Use Case ID	UC2
Use Case Name	Stop Cameras
Included Use Case(s)	None
Primary Actor(s)	User
Description	User can stop the multiple cameras by using OpenGL GUI.
Precondition	None

Use Case Diagram 2: Stop Cameras

Use Case ID	UC3
Use Case Name	Video Recording
Included Use Case(s)	None
Primary Actor(s)	User
Description	User can record HDR video from the multiple cameras.
Precondition	None

Use Case Diagram 3: Video Recording

Use Case ID	UC4
Use Case Name	Play Video
Included Use Case(s)	None
Primary Actor(s)	User
Description	User can play recorded video from player shown in HDR display.
Precondition	None

Use Case Diagram 4: Play Video

Use Case ID	UC5
Use Case Name	Pause Video
Included Use Case(s)	None
Primary Actor(s)	User
Description	User can pause video from player shown in HDR display.
Precondition	None

Use Case Diagram 5: Pause Video

Use Case ID	UC6
Use Case Name	Change Aggregation Settings
Included Use Case(s)	None
Primary Actor(s)	User
Description	User can change the aggregation settings from OpenGL GUI. Settings are Cross Aggregation , Aggregate 3x3 , Aggregate with PTKR , Aggregate 3x3 with PTKR
Precondition	None

Use Case Diagram 6: Change Aggregation Settings

Use Case ID	UC7
Use Case Name	Change Refinement Settings
Included Use Case(s)	None
Primary Actor(s)	User
Description	User can change refinement settings by using OpenGL GUI. Refinement settings are Median Filtering, Outlier Detection, Outlier Correction
Precondition	None

Use Case Diagram 7: Change Refinement Settings

Use Case ID	UC8
Use Case Name	Change Luminance Settings
Included Use Case(s)	None
Primary Actor(s)	User
Description	User can control base and maximum luminance settings for global tone map operation.
Precondition	None

Use Case Diagram 8: Change Luminance Settings

Use Case ID	UC9
Use Case Name	Display Disparity
Included Use Case(s)	None
Primary Actor(s)	User
Description	User can display the resuting disparity map obtained from LDR frames.
Precondition	None

Use Case Diagram 9: Display Disparity

Use Case ID	UC10
Use Case Name	Select CRF Profile
Included Use Case(s)	None
Primary Actor(s)	User
Description	User can control which camera response curve will be chosen.
Precondition	None

Use Case Diagram 10: Select CRF Profile

Use Case ID	UC11
Use Case Name	Change Exposure Settings
Included Use Case(s)	None
Primary Actor(s)	User
Description	User can control the exposure settings using OpenGL GUI.
Precondition	None

Use Case Diagram 11: Change Exposure Settings

Use Case ID	UC12
Use Case Name	Change Cost Settings
Included Use Case(s)	None
Primary Actor(s)	User
Description	User can change the cost settings. Cost settings are AD, AD Census, Census.
Precondition	None

Use Case Diagram 12: Change Cost Settings

System Design

4.1 Module Structure

E2EHDR consists of 11 separate projects, some libraries and some applications. Projects use CMake as a meta build system and we are able to build on all major platforms (OS X, Linux, Windows). Descriptions of each module is listed below.

Libraries:

- e2e_gl: This library provides a pipeline to compute disparity map and makes use of it to generate hdr frames. OpenGL and GLSL are used to utilize the computing power of the GPU. The library provides abstractions to handle OpenGL calls. It involves operations related textures, geometry, framebuffers, loading&binding&linking shaders and etc.

- e2e_calib: This library is used to calibrate the stereo camera setup we use to generate intrinsic and extrinsic parameters which are used to undistort and rectify input images.

- e2e_control: This library is used to control Sony IP cameras which have problems being controlled over ONVIF protocol. This library supports controlling the shutter times of cameras in an asynchronous manner.

- e2e_ffmpeg: This library is used to get real time video stream from IP cameras using the real time streaming protocol (RTSP). This library supports pulling frames from cameras encoded using H264 codec and converts the images from YUV420 color space to RGB8 color space, using multiple threads and single producer single consumer frame queues.

- e2e_gphoto: This library is used to get real time video stream from many brands of DSLR cameras using USB protocol. This library supports frames encoded using MJPEG codec and like ffmpeg, converts these frames to regular uncompressed RGB8 frames. This library is also asynchronous.

- e2e_utils: This library provides essential utilities used by almost all other libraries we've written such as image import/export, a zero overhead hierarchical profiler, a thread safe, zero allocation single producer single consumer queue, a thread pool for task based concurrency support, visitor utilities and a generic image frame template.

- e2e_webcam: This library provides support for pulling real time video stream from USB webcams over UVC protocol. This library supports frames encoded using H264/HEVC codec with YUV420 color space. Like other camera providers, this library is also asynchronous and communicates over spsc queues.

- e2e_x264: This library provides our own HDR video compression algorithm. Without details, we simply divide HDR frames in two and compress the two frames in parallel and encode them using H264 codec. Decoding is done vice versa.

- e2e_crf: This library provides unattended camera response function recovery support. It simply captures multiple exposures consecutively and using the method developed by Robertson Et Al.

Applications:

- e2e_camera: This application is the program that drives the stereo camera setup and supports recovering camera response functions, recording videos, changing camera settings, tonemapping, selecting HDR generation parameters and choosing stereo matching parameters.

- e2e_player: This application is used to drive our HDR display setup. Since we are actually using 2 display devices, regular OpenGL calls are unable to use them. Our player breaks HDR frames into two frames in order to be able to be used by the setup.

4.2 Source Code Structure

- `/3rd_party`: This folder includes several external libraries which are used in the project extensively.

- `/cameras`: This folder includes several parameter settings for the cameras.

- `/cmake_modules`: This folder includes several cmake modules for the external libraries.

- `/matlab`: This folder includes some prototypes written in matlab.
- `/samples`: This folder includes the libraries and applications we have written so far. Each of them is explained in the previous section.

4.3 Component Diagram

Since our project is aimed to execute in real-time, we loop over a fixed pipeline that we carefully designed. This loop contains several components that work collaboratively one after another to achieve the goal of producing hdr frames for videos and playing the videos that are produced. Interfaces containing the word "settings" are provided by the components in order to make it possible for users to adjust necessary parameters for different scenes and to leave the choice of performance-quality trade-off to the users. For instance, by altering the parameters of "Exposure Settings", we can set the exposures of the cameras to a lower level if the environment is bright or to a higher level if the environment is darker. Other interfaces related to adjusting parameters are working in the same manner.

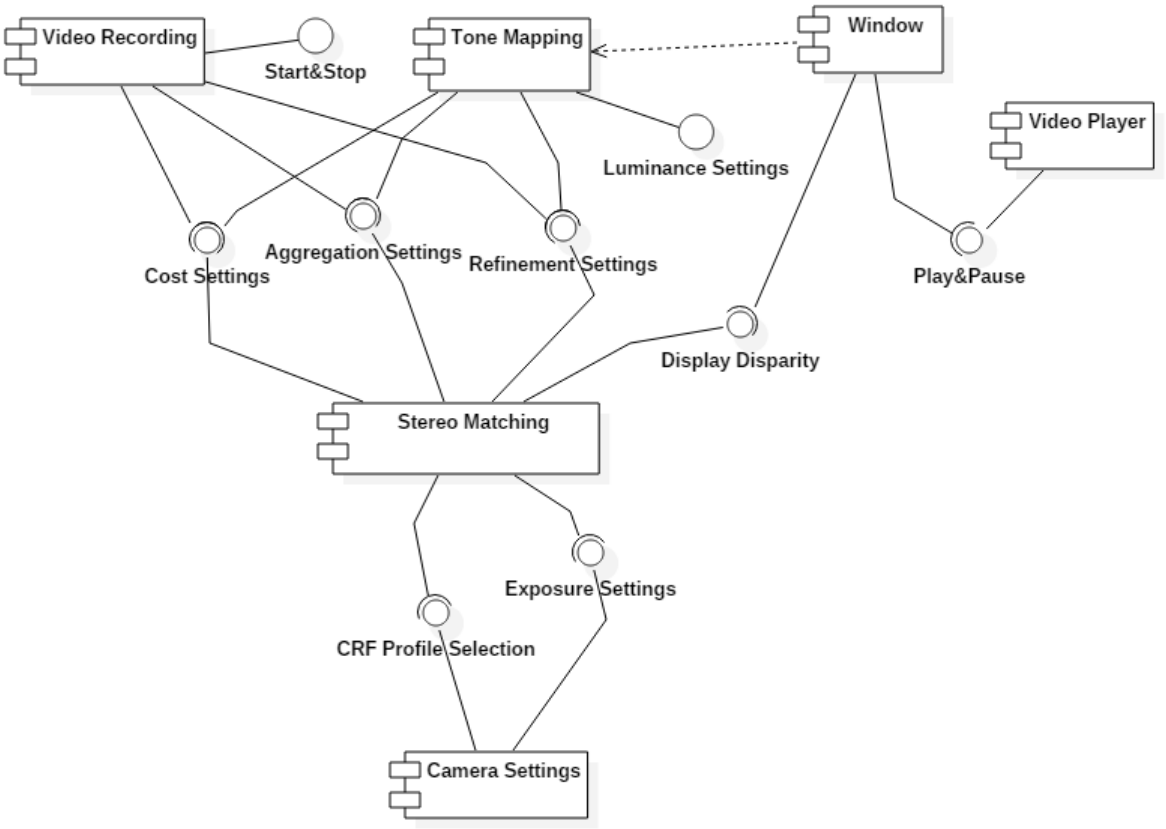


Figure 4.1: Component Diagram

4.4 Deployment Diagram

Deployment diagram, which is provided below, clearly indicates that our project does not depend solely on software or hardware but the balanced combination of them. The computer is the main hardware device which communicates with other hardware devices. The connection between the web cameras and the computer is established by the usb protocol. Frames taken by the cameras are sent to computer and the computer processes frames to produce a final hdr image. Then, it splits the resulted hrd image into two frames which are to be sent to projector and lcd. We use hdmi and vga for connections between the computer and projector and between the computer and the lcd, respectively.

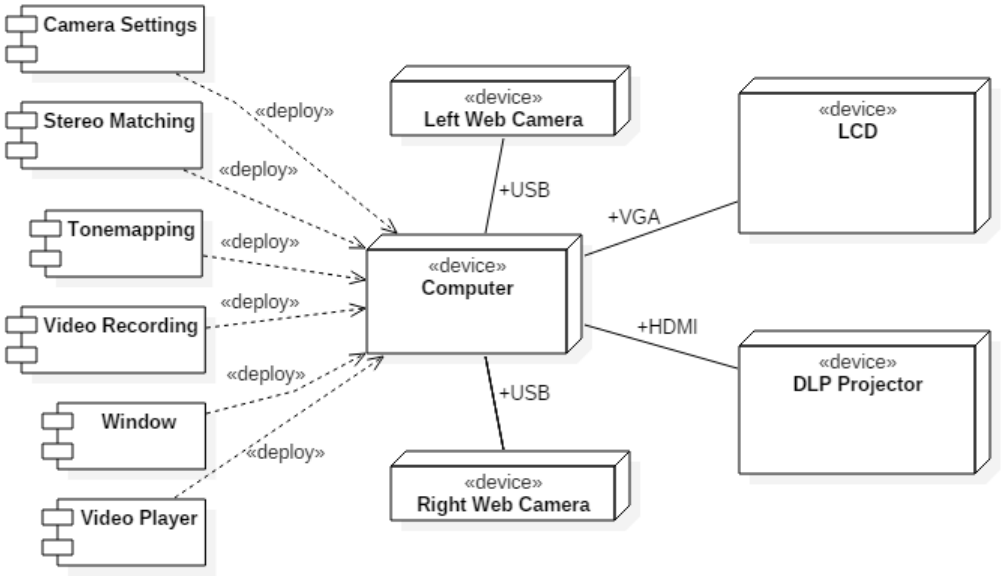


Figure 4.2: Deployment Diagram

Testing

This chapter contains the plan and details of testing our design project, E2EHDR.

5.1 Assumptions

As we are building real time software, performance is the most important aspect for us while programming. Due to this reason, in performance builds it's not possible for us to add checks for any kind of programmer, user or hardware error.

We basically assume these in a normal run or test:

1. Hardware interfacing libraries work reliably: most of the hardware interfacing libraries we use (such as OpenGL, UVC, FFMPEG and gPhoto) are written in C and do not support C++ exceptions and routinely report any kind of diagnostic through global error numbers. Checking for these global error codes after every function call is inadmissible. Therefore, we assume the hardware interfacing libraries do not have any internal defects (such as unimplemented functionality) that we have to check after every request.

One problem of our project is that the UVC library we use to control webcams sometimes silently fail when changing the exposure without reporting any error .In this case, we don't say check the new image whether the exposure actually changed or not.

2. Capture hardware is able to provide 720p video at or under 24 FPS: Most of our internal buffers and structure expect 720p frames, due to this, the cameras should provide this resolution since we cannot afford under or over sizing to fit to the resolution requirement. We store the images in RGB format in the program, however our software accepts both 8 bit RGB frames or YUV420 frames and convert if necessary.

Also, we process the frames at 24 frames per second, and do not do any synchronization with time. What this means is if the camera provides frames say at 30 frames per second, our software won't be able to keep up with the stream and will eventually run out of buffer space and will overwrite previous frames.

3. Input devices provide the same colors when the same software settings are used: we are using two camera frames to calculate disparity maps. The disparity map algorithm uses absolute color differences to determine which pixel corresponds to which pixel between images. If the cameras provide different colors for the same object due to some hardware issue, the disparity map is unable to work reliably.

5.2 Test Plan

As previously mentioned, our project contains of nine libraries and two applications. Six of these libraries are directly used to interface with, control or calibrate the input devices such as IP cameras, webcams or DSLR cameras. Without the devices being connected to the computer, it's not possible to run any unit or integration tests on these projects. Since the behaviour of the applications are meaningless without cameras or the HDR display being present, there are no tests for the applications either.

The remaining three libraries have their unit tests written using the `Catch Unit Test Framework` for C++.

Although there aren't any unit tests written for the remaining projects, all of our projects have proper assertions and in debug mode, almost all of the libraries do extensive checks on the pre-, post- conditions and class invariants.

5.3 Code structure

Our test code structure is quite simple. Any test related file resides in a sub-directory named `tests` under the project folder. We are using CMakes `enable_testing()` feature to integrate tests with the build process. Basically, running a `make tests` command runs the tests and reports any error during the test including crashes.

- `/samples/e2e_gl/tests`: Contains tests for:
  - Window creation
  - Shader compilation and linking
  - Texture creation
  - Framebuffers
  - Render to texture

- `/samples/e2e_utils/tests`: Contains tests for:
  - Thread pool
  - Jpeg image I/O
  - Hierarchical profiler
  - Single producer single consumer queue
  - General concurrency tests
- `/samples/e2e_x264`: Contains tests for:
  - x264 encoding
  - x264 decoding
  - SEI off data payload