6.1 Introduction

In the previous chapter, we introduced a framework to perform interactive 3D rendering on mobile devices. However, an issue associated with the mobile devices is that processing takes considerable time if the reference image is of large size. As the mobile devices have limited screen-size, therefore, methods to down-sample these large images can be utilized to further improve the performance. In this chapter, we propose a framework that extends the computation ally efficient 3D warping technique in wavelet-domain to render novel views of compressed images on android smartphones. The proposed framework is implemented using Android Development Tools (ADT). The performance of the proposed framework is evaluated with Android Virtual Device (AVD) emulator and real Android smartphones. The experimental results show that the proposed framework gives 39-41% better results in terms of rendering time as compared to the computation ally efficient 3D warping technique if the framework down-sample the images to one quarter of its original size. Further, the compressed rebuilt
image has visual quality close to the original image. Such a framework is very useful in visualization of large images on many smartphone applications such as virtual tours, virtual reality, e-commerce etc. at interactive rates.

6.2 Haar Wavelet Transform for Image Compression

Various image compression techniques are available in computer graphics literature which can be used to transform large images into considerably smaller representations. However, in the past few years, wavelets have shown to be more efficient than many other image compression methods [101]. Although wavelets have their roots in signal processing and approximation theory, they have recently been applied to many problems in computer graphics like image compression, image editing, animation, global illumination etc [102]. In general, the goal of wavelet theory is to create a mother wavelet function or wavelet transform that will give an informative, efficient, and useful description of the signal of interest. The power of wavelets is Multi-Resolution Analysis (MRA) [103] which allows representing different levels of detail of signal. For the purpose of multi-resolution formulation, there is also a need for a second function, known as scaling function, to allow analysis of the signal to finite number of components.

An image is a matrix of pixel (or intensity) values; therefore, it can be thought of as two dimensional signal, which change horizontally and vertically. Over the past few years, various wavelet-based image compression techniques such as Discrete Cosine Transform (DCT) [104], Haar transform [105], Daubechies transform [106, 107] etc. are available. Among these techniques, Haar transform is one that is suitable for image compression in
smartphones due to its low computing requirements. Haar transform uses mother wavelet function \( \psi(x) \) and scaling function \( \phi(x) \) as described below to approximate the original signal [103].

\[
\psi(x) = \begin{cases} 
1 & x \in [0, 1/2) \\
-1 & x \in [1/2, 1) \\
0 & \text{otherwise}
\end{cases} \quad (6.1)
\]

\[
\phi(x) = \begin{cases} 
1 & x \in [0, 1) \\
0 & \text{otherwise}
\end{cases} \quad (6.2)
\]

For Haar MRA, Eq. (6.1) and (6.2) gives the impulse response corresponding to the high pass filter (H) and the low pass filter (L) respectively. Thus, Haar wavelet analysis can be performed on images using the concepts of filters. As image is a 2D signal, we can get the scaling and mother wavelet function in 2D by multiplying 1D functions. The scaling function in 2D, \( \phi(x, y) \) can be obtained as

\[
LL : \phi(x, y) = \phi(x)\phi(y) \quad (6.3)
\]

Similarly, there exist three wavelets functions in 2D as

\[
\begin{align*}
HL : \psi^1(x, y) &= \phi(x)\psi(y) \\
LH : \psi^2(x, y) &= \psi(x)\phi(y) \\
HH : \psi^3(x, y) &= \psi(x)\psi(y)
\end{align*} \quad (6.4)
\]

Thus by applying the filters defined in Eq. (6.3) and (6.4), an image is decomposed into 4 bands: one approximation sub-image (LL), which shows the general trend of pixel
values; and three detail sub-images, which show the horizontal (HL), vertical (LH) and diagonal (HH) details or changes in the image. Fig. 6.1 shows 2D Haar wavelet transformation that divides $N \times N$ image into 4 sub-images, each having $(N/2) \times (N/2)$ dimension. The operations can be repeated on the LL band to get the next level of decomposition (Fig. 6.1).

![Haar wavelet decomposition of an image](image)

**Figure 6.1:** Haar wavelet decomposition of an image

To get a better idea about the implementation of this wavelet in image compression, consider a $512 \times 512$ pixels grayscale image of the woman (elaine_512.gif) as shown in Fig. 6.2. By applying Haar wavelet transform we can represent this image in terms of a

![Haar wavelet transform on grayscale image](image)

**Figure 6.2:** Haar wavelet transform on grayscale image
low-resolution image and a set of detail coefficients (Fig. 6.2). The detail coefficients can be used for the reconstruction of the original image.

In computer graphics to compress an image, we can use Haar wavelet low-pass (average) filter, $H_a$, and high-pass (difference) filter, $H_d$, as described below:

$$H_a = \frac{(P_i + P_{i+1})}{2} \quad \text{and} \quad H_d = \frac{(P_i - P_{i+1})}{2}$$  \hspace{1cm} (6.5)

where $P_i$ and $P_{i+1}$ are pixel values of the image. Taking one row at a time, first apply the filters defined in Eq. (6.5) for each pair of pixel values. This operation provides us an average value along with detail coefficients for each row. Next, these transformed rows are treated as if they were themselves as image and apply the filters defined in Eq. (6.5) to each column. This produces a matrix containing approximation part (storing the general trend of the image) and detail part (containing most values close to zero). In images, low frequency (changing slowly over the image) information is usually a lot more than high frequency (quickly changing) information. Due to this, most of the values resulting from the high-pass filter are close to 0. To illustrate how 2D Haar wavelet transformation is performed, consider the upper left $8 \times 8$ section of grayscale image in Fig. 6.2. Fig. 6.3 shows the resultant matrix by applying the filters defined in Eq. (6.5) on this matrix first row-wise and then column-wise.

Grayscale image consists of a single matrix, but for RGB images, there are 3 matrices of same size to represent three colors: red, blue and green. Therefore, for RGB images we apply Haar wavelet transform on 3 different matrices separately. Fig. 6.4 (b) shows the
256×256 pixels approximation part of 512×512 pixels RGB image (Lena.bmp) in Fig. 6.4 (a) after applying one level Haar wavelet transform which is visually close to original image.

(a) Original Image (512×512 pixels)    (b) Compressed Image (256×256 pixels)

**Figure 6.4:** Haar wavelet transform for RGB image compression

### 6.3 Android Platform for Smartphones

Android [108] is a Linux-based operating system for mobile and tablet devices. It is developed by Open Handset Alliance (OHA) [109] lead by Google Inc. It includes touchscreen user interface, camera, network data monitoring and all other features that
enable a mobile phone to be a smartphone. It also provides a platform that allow end users to develop, install and use their own applications.

### 6.3.1 Android Architecture

Android is a software stack of different layers (also known as android architecture) as shown in Fig. 6.6. Each layer groups several programs and provides different services to the layer just above it. The android software stack includes the layers of operating system, services and libraries, run-time environment, middleware, and applications as described below:

![Android Architecture Diagram](image)

**Figure 6.5: Android architecture**

- **Linux Kernel**: Android OS is built on the top of the Linux 2.6 kernel which acts as an abstraction layer between the hardware and rest of the software stack. It is
responsible for services like memory and process management, security, network stack, threading, camera and audio devices etc.

- **Android Runtime:** Android runtime consists of Dalvik Virtual Machine [110] and core Java libraries. Dalvik VM is a type of JVM to run applications and is optimized for low power and low memory requirements. The Dalvik VM allows multiple instances of virtual machine to be created simultaneously and each android application runs separately on its own Dalvik VM instance.

- **Libraries:** This layer holds the android native libraries. These libraries are written in C/C++ and offer services to the above layers. Some of the important libraries are Surface Manager, System C libraries, SQLite, WebKit, OpenGL etc.

- **Application Framework:** The application framework provides services to the built-in and user-defined android applications. Some of the important components of the application framework are:
  - Activity Manager that governs the application life cycle.
  - Content Providers that enable applications to either access data from other applications or to share their own data.
  - Resource Manager that provides access to non-code resources (such as graphics).
  - Notification Manager that enables applications to display custom alerts.

- **Applications:** Applications (build-in or user-defined) are the top layer in the android architecture. Important build-in applications include dialer app, camera app, maps app, email app and others. As developer, we are able to write user-
defined applications using the Java language which can replace any existing built-in application.

### 6.3.2 Android Activities

An activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email etc. A single android application may consist of multiple activities. When an application is launched in android, a starting activity is appeared as the application UI which may start another activity in response to a user action. During its lifetime, an activity can exist in three states as described below:

- **Active/ Running state:** An activity is in active or running state, if it is in the foreground (i.e. at the top of the activity stack). In this state, an activity has the highest priority and as such will only be killed by the android OS in extreme situations such as UI become unresponsive.

- **Paused state:** An activity is in paused state, if it has lost focus but is partially visible (i.e. if a new non-full-sized activity such as a notification appears). A paused activity is completely alive, that is, it maintains all state and member information. This is deemed to be the second highest priority activity in android and as such will only be killed by the OS if killing this activity will satisfy the resource requirements needed to keep the Active/ Running activity stable and responsive.
- **Stopped/ Background state:** An activity is in stopped or background state, if it is completely obscured by another activity. Stopped activities still retain all state and member information. However, they are deemed to be the lowest priority activities in android, and as such the OS will kill activities in this state first to satisfy the resource requirements of higher priority activities.

The life cycle of an activity is automatically managed by the android via a set of methods defined in the android.app.Activity class [111] – `onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, `onRestart()` and `onDestroy()`. As a developer we can override these methods to control the state changes. Figure 6.6 illustrates the activity lifecycle.

When an activity is first created, `onCreate()` is called to do the initialization. `onStart()` is always called by the system after `onCreate()` to put the activity in *Active* state. Activities may override this method to perform any specific tasks right before an activity becomes visible (i.e. active). `onPause()` is called when the system is about to put the activity into *Paused* state. Activities may override this method if they have to commit unsaved changes to persistent data or to cleanup objects consuming resources. In the *Paused* state, the system may either call the `onResume()` to resume the activity in the *Active* state or call the `onStop()` to place the activity in the *Stopped* state, depending on the user activity. In Stopped state, an activity can’t be resumed but it can be restated by calling `onRestart()` and it is always followed by `onStart()`. `onDestroyed()` is called just before an activity is destroyed to reclaim memory and other resources.
6.4 Proposed Framework for Android Smartphones

In this section, we propose a framework that extends the computationally efficient 3D warping technique in wavelet-domain to render novel views of compressed images on
android smartphones. The input for the framework is reference image (JPEG format) and disparity image (GIF format) of size $N \times N$ along with camera and viewing parameters such as center of projection for reference view $C_r$, center of projection for desired view $C_d$, and pinhole camera viewing parameters (Projection Matrix) $P_r$ and $P_d$ for reference and desired image view respectively. The output for this framework is novel views of the down-sampled reference image corresponding to desired viewing parameters on the smartphone screen.

**Figure 6.7:** Flowchart of proposed framework
Fig. 6.7 shows the flowchart of the proposed framework. The framework first down-sample the reference and disparity images by applying Haar-wavelet transform and then apply the computationally efficient 3D warping technique to generate the new novel views of down-sampled images on smartphones. The proposed framework is implemented using Android Development Tools (ADT) version 23.0.2 [112]; which can

```java
public class WaveletWarp extends Activity {
    ImageView desImage;  // To display desired output Image
    Bitmap img, dimg;    // For storing reference and disparity images
    HaarTransform ht1, ht2; // Defined in Fig. 6.9

    ...  
    public void onCreate(...) {
        ...  
        init();  
        ht1=new HaarTransform(img, ...);  
        ht2=new HaarTransform(dimg, ...);  
        warp(img, dimg, ...);  
        ...  
        draw();
    }
    ...  
    public void init() {
        /* Reads reference image together with its disparity image and corresponding  
           camera parameters such as Cr, Cd, Pr etc. */
    }
    public boolean onKeyDown(...){
        /* Handles DPAD buttons of android smartphone to perform zoom-in, zoom-  
           out and reset operations or to change the reference image itself */
    }
    public boolean onTouch(...) {
        /* To change the orientation and position of viewing parameters using touch  
           screen */
    }
    ...  
    // To draw the desired output image  
    protected void draw() {
        ...  
        desImage = Image.createRGBImage(...);
    }
}

Figure 6.8: WaveletWarp class of the proposed framework.
```
run on any android smartphone. Fig. 6.8 summarizes the WaveletWarp class of the proposed framework which extends Activity, a pre-defined fundamental class in android to create a window with UI [111]. The onCreate() method of the class invokes procedure init() to read the reference image together with its disparity image and corresponding camera parameters such as \( C_r, C_d, P_r \) etc. The reference and disparity images are then passed to the HaarTransform class described in Fig. 6.9 which applies level-one Haar wavelet transform on these images. In order to perform the transform, it first invoke procedure sepRGB() which separates the red, green and blue components of the image. Then it calls procedure HWT() on various image components to perform Haar-wavelet transform. Finally, the procedure comRGB() is invoked to combine the red, green and blue components of the image. This result decomposition of given images into their corresponding four sub-images (one approximation and three details). By ignoring the detail sub-images, the reference and disparity images are reduced to one quarter of its original size.

Then the framework invokes the procedure warp() to apply the computationally efficient 3D warping technique (described in chapter 4) on the compressed images. As the rendering time of the warping technique is directly proportional to the image size rather than image complexity, therefore the proposed framework improves the interactivity and performance while generating the novel views of the reference image on smartphones.

The onTouch() method of the class allows the user to navigate through the touch screen of the smartphone to change the orientation and position of the camera. Whenever the user performs the navigation, the new values for the camera parameters are computed and
the procedure to render the new view is started. With onKeyDown() method defined in the class the user can also perform zoom-in, zoom-out and reset operations or to change the reference image itself through the DPAD buttons (i.e. four-way directional arrow keys) available with android smartphones.

6.5 Experimental Results and Discussion

The proposed framework has been implemented using JavaSE (Java Platform, Standard Edition) 1.7 (update 67) [94] and Android Development Tools (ADT) version 23.0.2

Figure 6.9: HaarTransform class of the proposed framework.

```java
public class HaarTransform {
    int pixel[], r[], g[], b[]; // For storing the pixel values of the input image and its RGB components

    // Read input image data in pixel array
    sepRGB(...);
    HWT(r);
    HWT(g);
    HWT(b);
    comRGB(...);

    public HaarTransform(...) {
        // Perform Haar Wavelet Transform on various components of the input images
        public void sepRGB(...) {
            /* Separate the Red, Green and Blue components of input image in r, g and b arrays */
        }
        public void HWT(...) {
            /* Performs Haar Wavelet Transform on various components of the input images */
        }
        public void comRGB(...) {
            /* Combines the Red, Green and green components to generate the down-sampled output image */
        }
    }
}
```
The experiments have been conducted on a machine having 2.5 GHz Intel(R) Core(TM) i5 CPU and 4.0 GB RAM. A set of images taken from a dataset of images [113] has been used in our experiments. Fig. 6.10 shows a subset of images along with their gray scale images (having depth information).

![Images](512x512 pixels)

**Figure 6.10:** Images (with their depth information)

The ADT can be used to create Android Virtual Device (AVD) emulators [114] that enable us to simulate the smartphone environment on a PC. For experiments we have defined AVD having screen size 3.2" with 512 MB RAM. Fig. 6.11 shows the down-sampled novel view (size 256×256 pixels) of 512×512 pixels Books image using the proposed framework in the AVD emulator. Fig. 6.11 shows that the visual quality of the compressed rebuilt image is close to the original image.

We have conducted experiments on some real android devices to evaluate the performance of proposed framework. Fig. 6.12 shows the output of the proposed framework on the android devices – Motorola Moto G (XT1033), Motorola Droid Razr HD (XT926) and Samsung S3 Neo (GT-I9300I). Table 6.1 gives the rendering times (in milliseconds) of the different images using the proposed framework and its comparison with the computationally efficient 3D warping framework on various android devices having different processing enviroments. It is worth to note that performance of the
proposed framework is approximately 39-41% better in terms of rendering time compared to the computationally efficient 3D warping technique if the framework down-sample the images to one quarter of its original size.

In computationally efficient 3D warping framework, each pixel of the $N \times N$ image is computed exactly once; therefore the time complexity is $O(N^2)$. However, in proposed wavelet-based framework, only $(N/2) \times (N/2)$ approximate sub-image of $N \times N$ image is utilized; therefore the time complexity is $O(M^2)$ where $M = N/2$. Thus the proposed framework gives better results for down-sampled images in terms of rendering time as shown in the experimental results.

Figure 6.11: Rendering using proposed framework in AVD emulator
Figure 6.12: Generated reference image views using proposed framework on
(a) Motorola Moto G (XT1033) (b) Motorola Droid Razr HD (XT926)
(c) Samsung S3 Neo (GT-I9300I)

Table 6.1: Rendering time comparison of images shown in Fig. 6.10 on real android devices

<table>
<thead>
<tr>
<th>Devices</th>
<th>Description</th>
<th>Images</th>
<th>Rendering time using computationally efficient 3D warping framework (in ms)</th>
<th>Rendering time using proposed framework (in ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motorola Moto G</td>
<td>Android 4.4.2 with Quad Core 1.2 GHz, 1 GB RAM, 4.5” display</td>
<td>Books</td>
<td>767</td>
<td>302</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Art</td>
<td>771</td>
<td>305</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dolls</td>
<td>775</td>
<td>307</td>
</tr>
<tr>
<td>Motorola Droid Razr HD</td>
<td>Android 4.4.2 with Dual Core 1.5 GHz, 1 GB RAM, 4.7” display</td>
<td>Books</td>
<td>804</td>
<td>321</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Art</td>
<td>816</td>
<td>332</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dolls</td>
<td>806</td>
<td>319</td>
</tr>
<tr>
<td>Samsung Galaxy S3 Neo</td>
<td>Android 4.3 with Quad Core 1.2 GHz, 1.5 GB RAM, 4.8” display</td>
<td>Books</td>
<td>824</td>
<td>332</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Art</td>
<td>830</td>
<td>340</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dolls</td>
<td>832</td>
<td>344</td>
</tr>
</tbody>
</table>
6.6 Summary

In this chapter we propose a framework that uses the computationally efficient 3D warping technique in wavelet-domain to render novel views of a reference image on android smartphones. By applying Haar-wavelet transform we represent the reference and disparity images in terms of low-resolution images and a set of detail coefficients. By ignoring the detail coefficients and simply warping the approximation image we get the novel view of the reference image. As the rendering time of the warping technique is directly proportional to the image size rather than image complexity, this improves the rendering time. The framework is implemented with Android Development Tools (ADT) and its performance is evaluated. The experimental results show that the proposed framework gives 39-41% better results in terms of rendering time as compared to the computationally efficient 3D warping technique if the framework down-sample the images is to one quarter of its original size. Further, the compressed rebuilt image has visual quality close to the original image.