Framebuffer Compression Using Dynamic Color Palettes

Ayub A. Gubran
ayoubg@ece.ubc.ca
The University of British Columbia

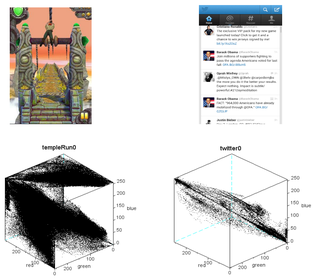
Tor M. Aamodt
aamodt@ece.ubc.ca
The University of British Columbia

a place of mind
THE UNIVERSITY OF BRITISH COLUMBIA

Problem and Motivation

- Mobile devices spend significant amount of energy to access off-chip memory.
- Graphics operations are a large consumer of off-chip bandwidth for operations like framebuffer reading/writing.
- Compression helps to reduce the amount of bandwidth substantially.
- Mobile devices usage statistics show that most of the time is spent using UI applications:

Time Spent on iOS and Android Connected Devices

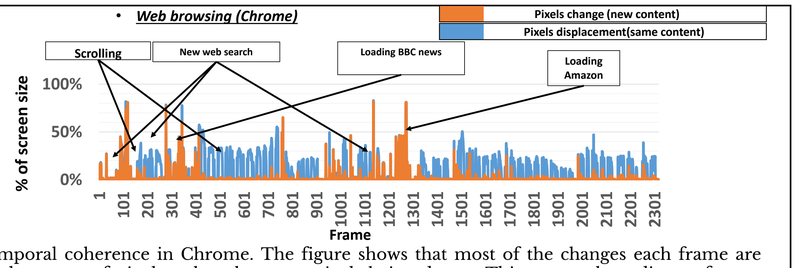


The RGB color space used by UI applications (Twitter on the right) is small compared to 3D applications (Temple Run 2) on the left.

- UI applications are highly compressible, compared to 3D applications, due to the simple content (solid background colors, text, icons…etc.).
- The simple content of UI applications has a potential for good compression using simple schemes like color palettes (dictionaries).
- However, to use color palettes, we need to predict the values of the palette of each frame before drawing it.

Main Observation and Contribution

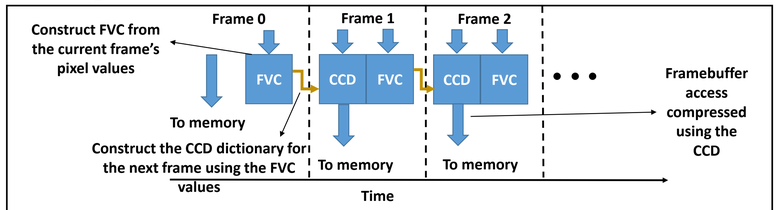
- Our main goal is to design a framebuffer compression scheme that is more effective to the most common use cases in mobile devices.
- We use a simple elegant scheme that exploits temporal coherence in graphics. Unlike other compression schemes, our scheme uses temporal, rather spatial coherence.



Temporal coherence in Chrome. The figure shows that most of the changes each frame are displacement of pixels rather than new pixels being drawn. This means that adjacent frames look very much alike, so we can predict the compression parameters for each new frame using the information from the previous frame(s).

Dynamic Color Palettes

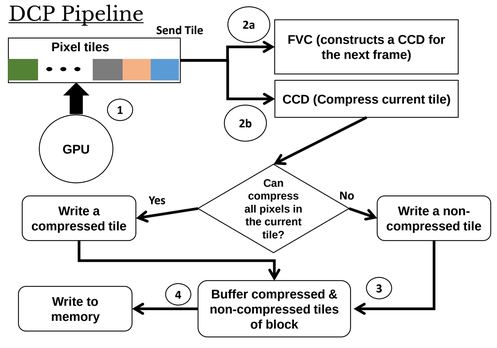
- **Dynamic Color Palettes (DCP) collects the most common pixel values in each frame to predict a compression palette (dictionary) that will be used to compress the next frame.**



In DCP two hardware structures are used. The first structure is the Frequent Values Collector (FVC), an associative structure that collects the most frequent color values. At the end of each frame, the FVC holds frequent values information, which is used to construct the Common Colors Dictionary (CCD). The CCD encodes the most frequent colors of the previous frame contained in the FVC. CCD is used to compress pixel tiles that exclusively contain color values in the CCD.

In stage **1** pixel tiles are sent to the framebuffer. Tiles can be organized in blocks (i.e., cache blocks) that contain one or more tiles.
At stage **2** each tile in a block is sent to the FVC and the CCD. In 2a , each pixel value in a tile accesses the FVC and changes the corresponding frequency value. In parallel at 2b, the tile's pixel values are checked by the CCD to see if all the pixel values exist in the CCD.

DCP Pipeline



In step **3** if all the pixels in a tile exist in the CCD then the tile is compressible. Each color value in a compressible tile is represented by log2(CCD size) bits. So for an RGBA color value of 32 bits and CCD size of 64 entries, each color value is compressed to 6 bits.
In **4**, compressed/non-compressed tiles are buffered and written to memory.

DCP Algorithms

- **We used three versions of the DCP compression. The first version is the base DCP where a basic dictionary encoding is used.**

Algorithm 1 Finding best number of CCD entries to use

```
1:  opt_bandwidth_cost = FrameSize*FullPixelSize   ▷ holds the frame size with the best compression rate
2:  opt_CCD = 0                                    ▷ Optimal CCD entries = 2^opt_CCD
3:  for i=0 to log2(FVC Size) do
4:      sum = SumFrequencies(FVC.Val(0) to FVC.Val(2^i) − 1)
5:      bandwidth_cost = sum * i + (FrameSize-sum)* FullPixelSize
6:
7:      if bandwidth_cost < opt_bandwidth_cost then
8:          opt_bandwidth_cost = bandwidth_cost
9:          opt_CCD = i
10:     end if
11: end for
```

The first variation is Variable DCP (VDCP) where the color palette size changes every frame to obtain the largest possible compression. By looking at the frequency of the most common pixel values, we resize the color palette and the encoding size as a consequence.

The second algorithm is Adaptive DCP (ADCP). Here we choose the encoding size for each color based on its frequency. We use an auxiliary buffer, Compression Status Buffer (CSB), which indicates the encoding lengths of each compressed tile. More common pixel values use shorter encodings. In this aspect, ADCP is similar to Huffman encoding but with a simpler implementation.

CCD (sorted by frequency)	
C_0	000
C_1	001
C_2	010
C_3	011
C_4	100
C_5	101
C_6	110
C_7	111

Tile colors	Encoding	CSB
$\{C_0, C_2\}$	{00,01}	010
$\{C_0, C_1\}$	{0,1}	001
$\{C_0, C_0\}$	{ϕ}	000
$\{C_0, C_3\}$	{000,101}	011
$\{C_0, C_2\}$	{110,010}	011
$\{C_0, C_3\}$	{110,111}	011
$\{C_0, C_7\}$	$\{C_0, C_7\}$	111
$\{C_7, C_2\}$	$\{C_7, C_2\}$	111

Evaluation

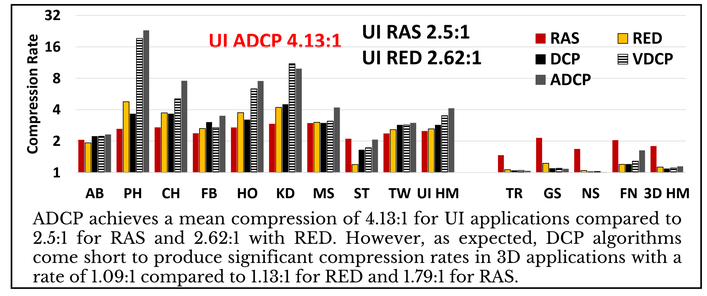
- **We evaluated DCP using a set of frames from mobile UI and 3D workloads.**

UI Workloads		
AB: Angry Birds	PH: Android Phone	CH: Chrome
FB: Facebook	HO: Android Home Screen	KD: Amazon Kindle
MS: Android Messaging	ST: Android Settings	TW: Twitter

3D Workloads	
TR: Temple Run 2	GS: Gunship 2
NS: Need for Speed	FN: Fruit Ninja

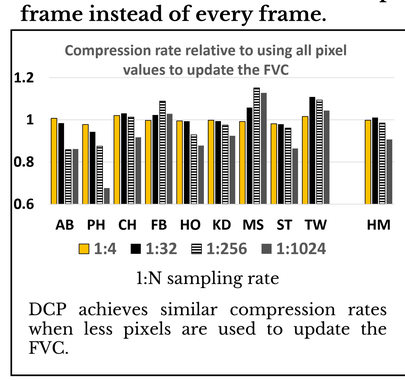
- **DCP is compared against two framebuffer algorithms. The first one by Rasmusson et al., (RAS)[1] which uses spatial coherence to encode difference between predicted and actual pixel colors.**
- **The second algorithm (we refer to it by RED)[2] compresses tiles of similar color values. This algorithm is efficient when there are large spaces covered by similar color values, as the case with UI apps.**
- **DCP configuration that we used are as follows:**

DCP Configurations			
FVC Size	64	**CCD Size**	64
FVC Replacement Policy	Least-frequent value	**Compression tile size**	2x2



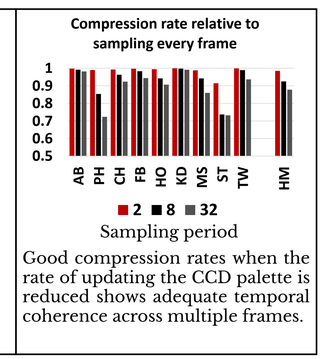
ADCP achieves a mean compression of 4.13:1 for UI applications compared to 2.5:1 for RAS and 2.62:1 with RED. However, as expected, DCP algorithms come short to produce significant compression rates in 3D applications with a rate of 1.09:1 compared to 1.13:1 for RED and 1.79:1 for RAS.

- **As pixels are required to update the FVC to be able to collect frequent values, this may create a bottleneck as FVC updates are atomic. So we evaluated the effect of reducing the number of pixels that update the FVC, so that only 1 out of N pixel values is used.**
- **Also we evaluated the rate of creating new CCD dictionaries. Where we construct a new compression dictionary every Nth frame instead of every frame.**



1:N sampling rate

DCP achieves similar compression rates when less pixels are used to update the FVC.



Sampling period

Good compression rates when the rate of updating the CCD palette is reduced shows adequate temporal coherence across multiple frames.

(1) RASMUSSON et al., T. 2007. Exact and error-bounded approximate color buffer compression and decompression. In the Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware
(2) NVIDIA, 2015. NVIDIA Tegra X1 Whitepaper. URL: international.download.nvidia.com/pdf/tegra/Tegra-X1-whitepaper-v1.0.pdf