Exact and Error-bounded Approximate Color Buffer Compression and Decompression

Jim Rasmusson\textsuperscript{1,2}  
Jon Hasselgren\textsuperscript{1}  
Tomas Akenine-Möller\textsuperscript{1}

\textsuperscript{1} Lund University  
\textsuperscript{2} Ericsson
Outline

• Rationale
• Prior art
• A new color buffer compression algorithm
  • Exact (lossless) mode
  • Approximate (lossy) mode
• Results
• Conclusions
• Future work
Why color buffer compression?

- Graphics processing speeds: ~ +70% /yr
- Memory bandwidth speeds: ~ +25% /yr

  ➡ increasing need for compression techniques aimed at saving bandwidth

- Texture, depth and color buffer compression
Prior art

1. Color plane compression [Molnar et al 04]
2. Color offset compression [Morein and Natale 03]
3. Entropy coded pixel differences [van Hook 06]

• All are tiles based
• All are lossless
• Implemented and benchmarked
Prior art 1: Color plane compression

- Calculate predictor plane using 3 fixed reference pixels \((p, dx, dy)\)
- Store differences between plane and actual color
- 2x4 pixel tiles

- Simple and robust
- Low compression ratios
Prior art 2: Color offset compression

- Identify and store reference values (e.g. max and min)

- Store offset values and reference index

- Simple and robust
- Medium compression ratios
Prior art 3: Entropy coded pixel differences

- Calculate adjacent pixel differences
  - Different traversal modes

- Entropy coding (“Exponent” encode)
  - Similar to Golomb-Rice

- Medium complexity
- Good compression ratios

[Bar chart showing pixel color values and adjacent differences]
Our algorithm, Exact mode

- New combination of known building blocks
  - Exact reversible color transform (RGB to Luminance-Chrominance)
  - Predictor (2x2 pixels)
  - Golomb-Rice encoding (adaptive variable bit rate)
- 8x8 pixel tiles
Exact reversible color transform

- Bit-exact reversible
- Decorrelates RGB components
  - typically improves compression rates
- We use Malvar and Sullivan [03]
- An alternative is RCT (JPEG2000)
- Enables different compression modes for luminance and chrominance

**RGB to YCoCg:**

- $Co = R - B$
- $temp = B + (Co >> 1)$
- $Cg = G - temp$
- $Y = temp + (Cg >> 1)$

**YCoCg to RGB:**

- $temp = Y - (Cg >> 1)$
- $G = Cg + temp$
- $B = temp - (Co >> 1)$
- $R = B + Co$
RGB vs Luminance-Chrominance
Predictor

• Weinberger et al [96]

\[ \hat{x} = \begin{cases} 
    \min(x_1, x_2), & x_3 \geq \max(x_1, x_2) \\
    \max(x_1, x_2), & x_3 \leq \min(x_1, x_2) \\
    x_1 + x_2 - x_3, & \text{otherwise.} 
\end{cases} \]

• 2x2 pixel blocks
• Built-in “edge detection”
• Low complexity, decent performance
Golomb-Rice coding

- We then encode the residuals using the Golomb-Rice scheme [Golomb66], [Rice79]
  - Compact coding of small values
  - Adaptive and variable bit rate
  - High compression ratios
  - Medium complexity
Golomb-Rice coding

- Find divisor, $2^k$, (we do exhaustive search)
- Divide values in 2x2 block, $v_i$, with $2^k$
  - Quotient $q_i = \text{int}(v_i/2^k)$
  - Remainder $r_i = v_i \mod 2^k$
- Store results
  - $q$ with unary coding (see table)
  - $r$ with binary coding
  - Termination zero to separate $q$ and $r$
- $k$ as a 3 bit header per 2x2 block

<table>
<thead>
<tr>
<th>Value</th>
<th>Unary coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>110</td>
</tr>
<tr>
<td>3</td>
<td>1110</td>
</tr>
<tr>
<td>4</td>
<td>11110</td>
</tr>
<tr>
<td>5</td>
<td>111110</td>
</tr>
<tr>
<td>6</td>
<td>1111110</td>
</tr>
</tbody>
</table>
Example

- Example $v = (3, 0, 9, 1)$
  - We find $k=1$
  - After division $\Rightarrow (q,r)$-pairs: $(1,1), (0,0), (4,1), (0,1)$
  - $q$ unary and $r$ binary coded $\Rightarrow$
    $(10,1), (0,0), (11110,1), (0,1)$

- Algo (once again):
  - Find divisor $2^k$
  - Divide with $2^k$
    - Quotient $q_i = \text{int}(v_i/2^k)$, Remainder $r_i = v_i \% 2^k$
  - $q$ unary and $r$ binary
  - Termination zero to separate

<table>
<thead>
<tr>
<th>Value</th>
<th>Unary coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>110</td>
</tr>
<tr>
<td>3</td>
<td>1110</td>
</tr>
<tr>
<td>4</td>
<td>11110</td>
</tr>
<tr>
<td>5</td>
<td>111110</td>
</tr>
<tr>
<td>6</td>
<td>1111110</td>
</tr>
</tbody>
</table>
Tested scenes

- Logging OpenGL driver
- Full incremental rasterization
- “Ducks” (homebrew)
- “Square” (Quake 3)
- “Car” (Quake 3)
- “Quake 4″
Results (lossless)

<table>
<thead>
<tr>
<th>Compression ratio</th>
<th>Ducks</th>
<th>Square</th>
<th>Car</th>
<th>Quake4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plane</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Entropy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Our</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Approximate mode (lossy)

- Why is lossy color buffer compression so “unpopular”?
- Tandem lossy compression accumulates errors incrementally and can grow very large
Approximate mode (lossy)

• Why is lossy color buffer compression so “unpopular”?
  • Tandem lossy compression accumulates errors incrementally and can grow very large
Approximate mode (lossy)

- Why is lossy color buffer compression so “unpopular”?
- Tandem lossy compression accumulates errors incrementally and can grow very large
Approximate mode (lossy)

- Why is lossy color buffer compression so “unpopular”?
  - Tandem lossy compression accumulates errors incrementally and can grow very large
Approximate mode (lossy)

- Why is lossy color buffer compression so “unpopular”?
- Tandem lossy compression accumulates errors incrementally and can grow very large

Lossy compress again → C
Approximate mode (lossy)

- Why is lossy color buffer compression so “unpopular”?
  - Tandem lossy compression accumulates errors incrementally and can grow very large

⇒ An error control mechanism is needed that bounds the error!
Error control mechanism

- Track and store an accumulated error metric for each tile, $T_{\text{accum}}$
Error control mechanism

- Track and store an accumulated error metric for each tile, $T_{\text{accum}}$
Error control mechanism

- Track and store an accumulated error metric for each tile, $T_{\text{accum}}$
Error control mechanism

- Track and store an accumulated error metric for each tile, $\tau_{\text{accum}}$
- Before applying more lossy compression:
  - Calculate $\tau_{\text{new}}$ (new error contribution)
Error control mechanism

- Track and store an accumulated error metric for each tile, $T_{\text{accum}}$

- Before applying more lossy compression:
  - Calculate $T_{\text{accum}} + T_{\text{new}}$
Error control mechanism

• Track and store an accumulated error metric for each tile, $T_{\text{accum}}$
• Before applying more lossy compression:
  • Check if $T_{\text{accum}} + T_{\text{new}} < T_{\text{thresh}}$?
Error control mechanism

- Track and store an accumulated error metric for each tile, $T_{\text{accum}}$
- Before applying more lossy compression:
  - Check if $T_{\text{accum}} + T_{\text{new}} < T_{\text{thresh}}$?
    - Yes $\Rightarrow$ Lossy compress
    - No $\Rightarrow$ Lossless compress
- We used RMSE as error metric
  - MSE or SAD also useful
Bounded errors

- The error control mechanism
  - Good control
  - Upper error bound
  - Minimize visual impact
Our compression scheme

- Chrominance 2x2 sub-sampling
- Only sub-sample if errors are below $T_{\text{thresh}}$
- Luminance is always losslessly compressed
Results (lossless vs lossy)

<table>
<thead>
<tr>
<th></th>
<th>Lossless</th>
<th>RMSE=2</th>
<th>RMSE=5</th>
<th>RMSE=15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ducks</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Square</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Car</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Compression ratio

- Ducks: Lossless = 3, RMSE=2 = 4, RMSE=5 = 4, RMSE=15 = 5
- Square: Lossless = 3, RMSE=2 = 4, RMSE=5 = 4, RMSE=15 = 5
- Car: Lossless = 3, RMSE=2 = 4, RMSE=5 = 4, RMSE=15 = 5
Visual quality - PSNR

• PSNR reasonably good for “square” and “car”
• “Ducks” - quality drops fast!
Visual quality - Structural Similarity Index Metric (SSIM)

- SSIM [Wang02]
  - “Square” and “car” - high quality
  - “Ducks” - something is not good
Visual quality - Artifacts

- “Square” and “car” have very small artifacts
- “Ducks” has clearly visible artifacts
Visual quality - Artifacts

• Ducks artifacts - why?
  • Chrominance errors leak into luminance component due to sub-sampling, color transform and tandem compression
  • On slowly changing color areas (small color gradients) this becomes visible
And animated?

- Lossless vs lossy for animated content
- Next slide shows video for simple animation
- Lossy configuration
  - Max chrominance RMSE error is 15 (the highest error threshold featured in the paper)
- Note the diagonally split screen
And animated?
Conclusions

• Lossless color buffer compression
  • Reversible color transform
  • Simple and effective predictor (2x2 pixels)
  • Golomb-Rice coding of residuals
  • Higher performance than state-of-the-art patents

• Lossy mode
  • Lossless algo with chrominance sub-sampling
  • Improves compression rates even more
  • Error control mechanism bounds the error
  • Very small visual impact
    for most scenes
Future work

- More sophisticated lossy algorithms
- Multi-sampling
- Floating-point and high dynamic range
- Approximate depth buffer compression?
Thank You!

- http://graphics.cs.lth.se