Image processing on GPUs

Rahul Narain

COMP790-058: GPGP

March 7, 2007
Image processing

• Image = 2D array of color values (1D or 3D)
• Most image processing algorithms are inherently parallel
  Do “the same thing” for every pixel
• Memory intensive with coherent lookups
Image processing

Image processing maps well to GPUs

2D image 2D texture
Per-pixel operations Fragment program
Memory intensive Fast texture lookup
Accuracy is not critical Good!
Image processing on GPUs

Input image bound to texture

CPU

Read input pixels via texture lookup

Screen-aligned quad of output image size

Vertices

Pixels processed using fragment shader

Output in texture or framebuffer
Topics

• Color correction
• Convolution
• Wavelet transforms
• Anisotropic diffusion and depth of field
• HDR and tone mapping
Color correction

• Brightness/contrast, hue/saturation, gamma, thresholding, Levels and Curves, …
Color correction

• Process each pixel independently

\[ t : \mathbb{R}^3 \rightarrow \mathbb{R}^3 \]

• Usually process each channel independently

\[ t_R, t_G, t_B : \mathbb{R} \rightarrow \mathbb{R} \]

• Pass three lookup tables as a 1D RGB texture

\[ g_R[x,y] = t_R[f_R[x,y]] \]
Convolution

\[ g[x,y] = \sum f[x+i,y+j] h[i,j] \]

- Pass kernel \( h \) and sampling coordinates \([i,j]\) as uniform data arrays
- Requires \( N \) or \( N^2 \) texture lookups per pixel
  Used to be a problem on old graphics cards
  \texttt{EXT\_convolution} is only supported by SGI
Convolution

Convolution with limited texture lookups:

1. Clear output buffer
2. For each pass:
   1. In vertex program, generate $k$ texture coordinates corresponding to adjacent pixels
   2. In fragment program, compute partial sum of $k$ terms and add to output buffer

Requires $N/k$ passes
Convolution

- Now only limited by fragment program instruction length
- All texture lookups access nearby pixels
  Very fast due to cache coherence
Convolution

- Fialka and Čadík: NVIDIA GeForce 6600
- GPU outperforms CPU in all cases
Convolution

- 3D convolution for volume data
- Current GPUs don’t allow high-precision 3D textures
  Load slices into several 2D textures instead
- Multiple passes to loop over slices
- Only 16 textures can be bound at a time
  Use multi-pass algorithm if kernel is wider in z
Non-linear filtering

- Median filter
  \[ g[x,y] = \text{median} \{ f[x+i,y+j] \} \]
- Can be done naïvely for smallish filter sizes
  Known fast algorithms are not parallelizable
- Even then, naïve GPU is faster than fast CPU
- Viola et al: 1.17× speedup on 5×5×5 volume filter using NVIDIA GeForce FX 5800
Non-linear filtering

- Bilateral filter
  \[ g[x] = k^{-1} \sum f[x'] h_s[x'-x] h_r[f[x']-f[x]] \]
  \[ k = \sum h_s[x'-x] h_r[f[x']-f[x]] \]

- Naïve approach: 1.52× speedup [Viola et al]

- Paris and Durand’s fast approximation [2006] should be parallelizable on GPU
Wavelet transforms

- Multi-resolution decomposition of a signal
- Basis functions are localized in both position and frequency
Wavelet transforms

Decomposition

Reconstruction
Wavelet transforms

- All wavelet coefficients stored in a texture
  Two for ping-pong
- Each pass reads/writes a subset of the texture
- Convolutions are separable
Wavelet transforms

- **Forward DWT:**
  \[ c_{j-1}[n] = \sum h[k] c_{j}[2n-k], \quad d_{j-1}[n] = \sum g[k] c_{j}[2n+1-k] \]
  \[ z_{j-1} = [c_{j-1} \ d_{j-1}] \]

- **Boundary extension using indirection texture**
Wavelet transforms

- Inverse DWT:
  \[ c_j[n] = \sum h[k] c'_{j-1}[\frac{n - k}{2}] + \sum g[k] d'_{j-1}[\frac{n - k}{2}] \]

- Two cases depending on whether \( n \) is even

- Avoid conditionals using precomputed indirection texture
Wavelet transforms
Wavelet transforms

- Wong et al: NVIDIA GeForce 7800 GTX
- Performance gain over CPU for large images
Diffusion

- Diffuse intensities over image at varying rates
- Anisotropic diffusion
  - Low diffusion at edges
- Depth of field
  - Radius of confusion
Diffusion

\[ u' = \nabla \cdot (g \, \nabla u) \]

- Discretize differential equation over pixel grid
  - Finite differences in space
  - Implicit 1\(^{st}\)-order Euler in time
- Solve linear system of equations per iteration
  \[ A^k(u^k) \, u^{k+1} = r^k(u^k) \]
Diffusion

- $A$ is sparse, banded with known structure
- Don’t want to represent whole matrix in memory
- Structure of $A$ allows simplification
Diffusion

Rumpf and Strzodka [2001]:

- Use Jacobi or conjugate gradient iterations
  
  \[ x^{i+1} = F(x^i) = D^{-1}(r - (A - D)x^i) \]

- Corresponds directly to image blending

- Can be implemented directly in OpenGL!

- NVIDIA GeForce 3: 8ms per iteration on 256×256 image
Diffusion

1. Upload original image $u^0$ to texture

2. For each timestep $k$:
   1. Initialize r.h.s. $r^k$ (usually equals $u^k$)
   2. (If necessary) calculate image of diffusion coefficients $g^k$ using lookup table
   3. Initialize $x^0 = r^k$
   4. For each iteration $i$:
      - Calculate $x^{i+1} = F(x^i)$ using image blending
   5. Store the solution $u^{k+1} = x^{i+1}$
Diffusion

Kass et al [2005]:

• Approximate by two 1D diffusions instead
• $n$ linear systems for $n$ rows, tridiagonal A’s
• Represent A’s using 3 channels of each row of 2D texture
• Solve in parallel using cyclic reduction
• NVIDIA GeForce 7800: 0.15s for 1024×1024
Diffusion

1. Gaussian elimination on odd rows in parallel
2. Copy smaller system of even rows to new texture; solve recursively
3. Propagate solution to odd rows
HDR

- OpenEXR: `half` datatype = 16-bit floating point
- Identical to native `half` datatype on GPUs
- Floating-point textures allow HDR
Tone mapping

- Displaying HDR images on LDR devices
- Reduce the dynamic range of an HDR image while “looking the same”
- Several techniques
- Reinhard et al.’s method has been implemented in real-time on the GPU
Tone mapping

- Compute log average luminance
- Rescale pixel luminances by average
- Find local average luminance of each pixel
  - Convolve with Gaussian filters of various widths
  - Compare to find best scale for each pixel
- Apply transfer function based on per-pixel local average luminance
Tone mapping

First pass

- Compute log average luminance
  
  Sum over entire image by repeated reduction

Several passes

- Convolve rescaled image with Gaussian filters of various widths and compare
  
  Accumulate results for “best” scale in texture

Final pass

- Apply transfer function
Tone mapping

- Goodnight et al: ATI Radeon 9800
- GPU is faster than CPU in all cases
Conclusion

• GPUs significantly accelerate image processing
  Pixel-level parallelism
  High memory bandwidth
• Previously slow operations now run at interactive rates on GPU
References


References


