graphics hardware 06

Efficient Video Decoding on GPUs by Point Based Rendering

Bo Han, Bingfeng Zhou

Peking University

Outline

- Motivation and Goal
- Previous work
- Review of video decoding
- Point based decoding framework
- Results
- Discussion

Motivation

- Diverse video applications
  - Range from HDTV to mobile devices
  - Multi video standards coexist
  - most concerns: video playback
  - Computation and Bandwidth
- Successful decoding system need:
  - High performance and programming flexibility
  - CPU + additional hardware

Motivation

- GPUs are powerful and flexible
  - Attractive coprocessors for GPGPU
  - Spreading to everywhere
- History of offloading video decoding tasks
  - Overlay surface for YUV to RGB
  - dedicated hardware for DVD (DXVA)
  - Programmable Video Engine (PureVideo, AVIVO)
  - What’s the next? (shader based?)

Our Goals

- Video decoding framework
  - Built on Graphics pipeline and Shader programs
  - Hardware performance + Software flexibility

- Additional advantages
  - Independent of Hardware and platform
    - Graphics API and shader languages
  - Save hardware resources
  - Amazing growth rate over Moore’s law

Previous work

- Video/image decoding process
  - Motion compensation on GPUs [Shen. etc 2005]
  - DCT/IDCT on GPUs [NVIDIA 2005][Fang. etc 2005]
  - Fast interpolation for ME [Kelly. etc 2004]
  - H.263 decoder on GPUs [Hirvonen. etc 2005]
- Limitation and weakness
  - Single quad-texture for the whole picture
  - Ignore the features of video data
  - Performance and flexibility not satisfying

Our Contributes

- Generic video decoding framework
  - Flexible point-based representation
  - Easily exploit parallelisms of decoding process
  - Efficiently map to graphics tasks
  - Both performance and flexibility

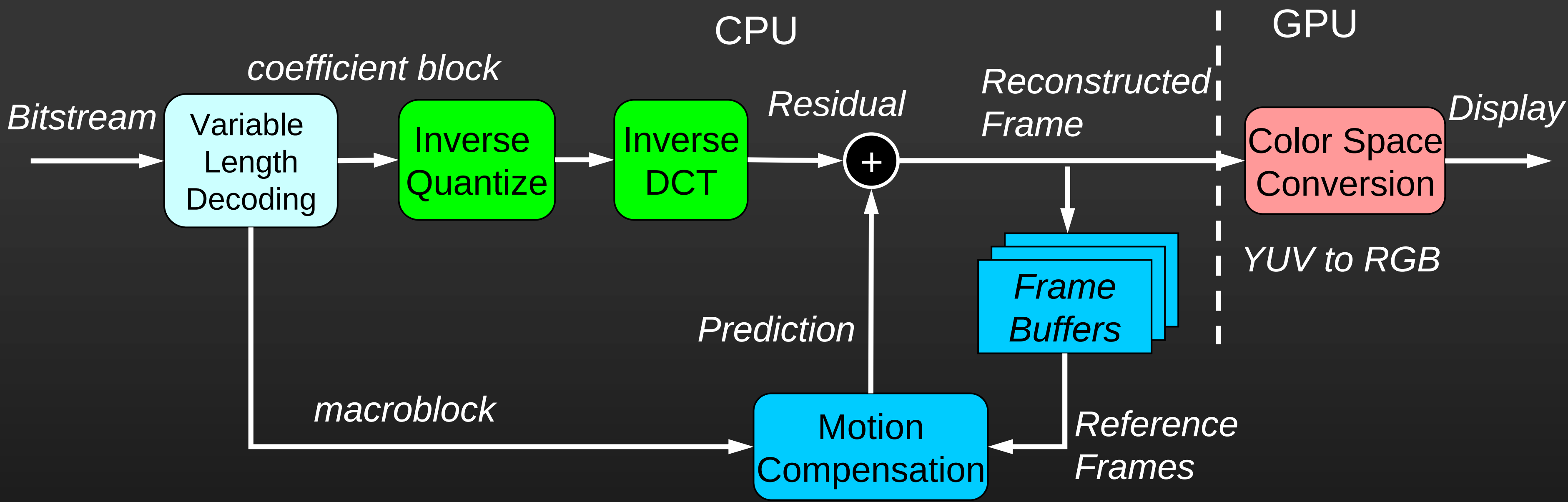
Outline

- Motivation and Goals
- Previous work
- Review of video decoding
- Point based decoding framework
- Results
- Discussion

Review of video decoding

- DCT-MCP hybrid coding
  - DCT & Motion compensation and prediction
- Block based structure
  - Block and macroblock (basic processing units)

Review of video decoding

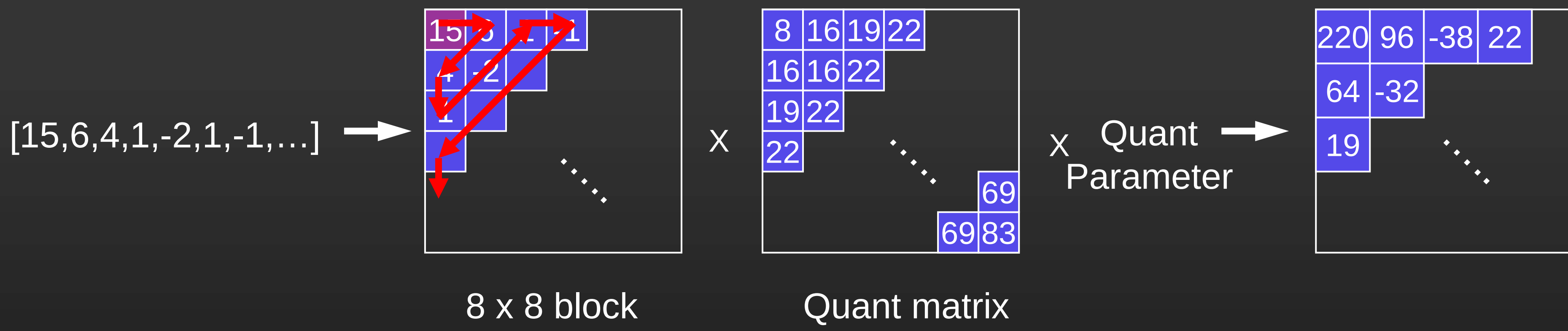


- VLD is sequential bit-wise operation
- Others show parallelism and streaming

For "each coefficient block" Do
perform IQ and IDCT

For "each macroblock" Do
perform MC

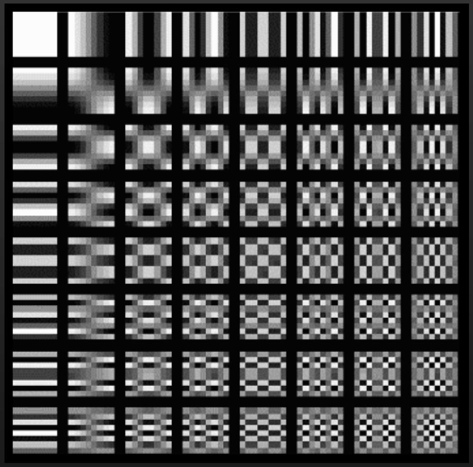
Inverse Quantize (IQ)



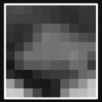
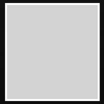
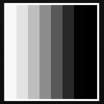
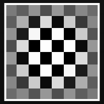
- Inverse Zigzag scan: reconstruct block
- IQ: $X_{IQ}(u,v) = X_Q(u,v) \times QM(u,v) \times qp$
- Characteristics:
  - Sparse and Coefficient-level parallelism

Inverse DCT

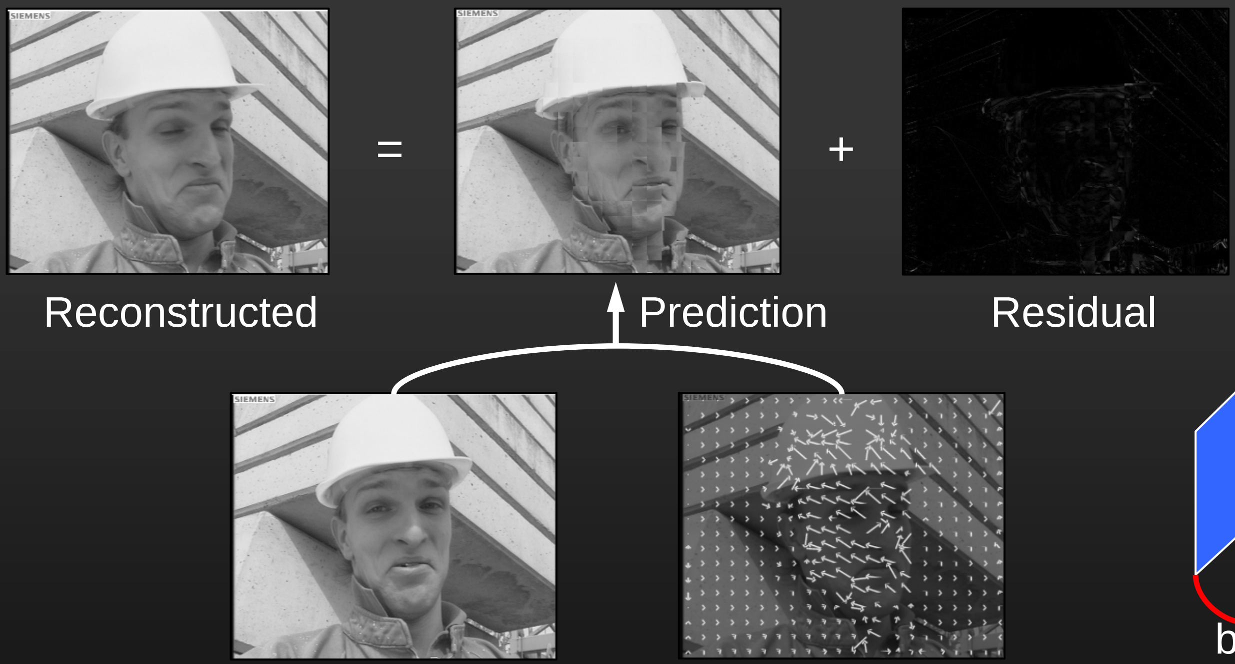
- IDCT is typically computation intensive
  - Many fast algorithms, but not for GPU
  - Coefficient and its basis image
  - Parallel and stream processing

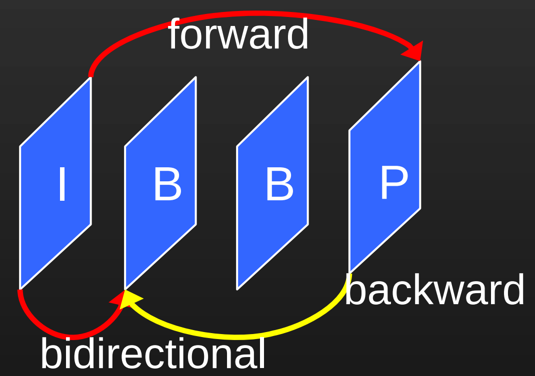


$$x = T^T X T = \sum_{u=0}^{8} \sum_{v=0}^{8} X(u,v)[T(u)^T T(v)]$$

 $= X(0,0)\times$  $+ X(1,0)\times$  $+ \ldots\ldots + X(7,7)\times$ 

Motion Compensation



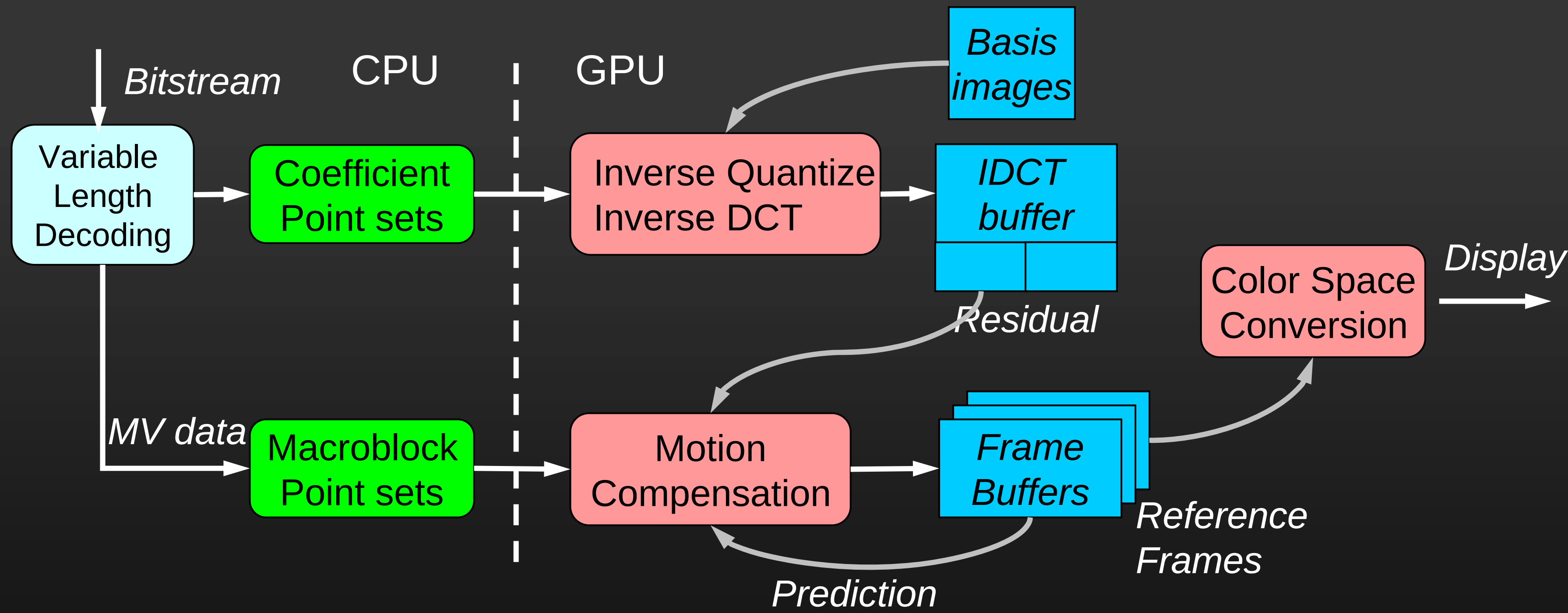


- Memory and Computation intensive
  - Block translation according to motion vectors
  - Per-pixel arithmetic operations
- Fit well with texture sampling scheme

Outline

- Motivation and Goals
- Previous work
- Review of video decoding
- Point based decoding framework
- Results
- Discussion

Overview of our framework



- Convey block-wise information with point's attributes
- Batch points into vertex arrays
- Render points to active shader programs

Map Video blocks to Graphics points

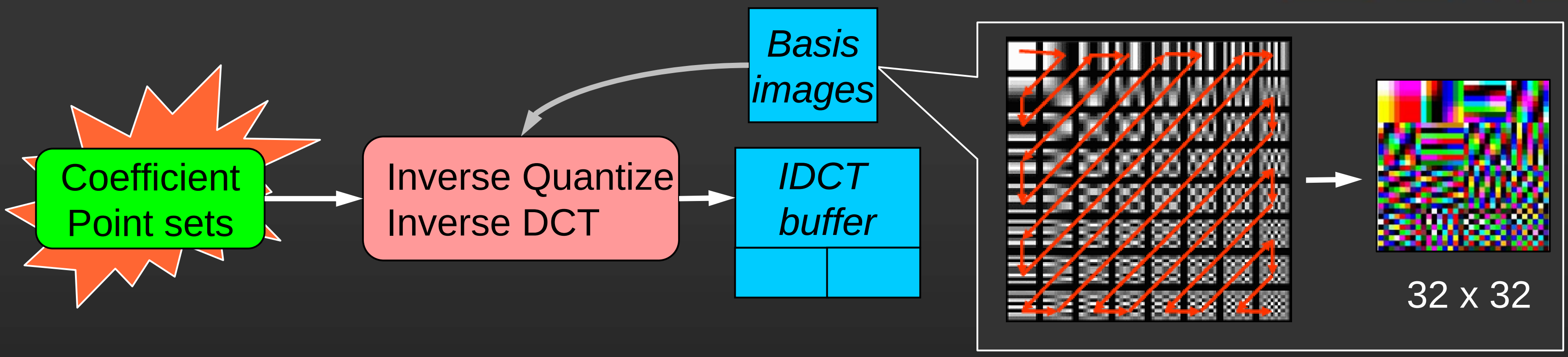
	Size	Attributes
Point primitive	Variable	position, normal, color, texcoords0-7…
Macroblock	16x16	position, motion vectors, MB type, DCT coding type
Coefficient block	8x8	position, quant parameter, sparse coefficients

- Natural for vertex processing
- Rasterized to fragment blocks (flexible size)
- Fragment processing
  - Point sprite extension and *WPOS* semantics

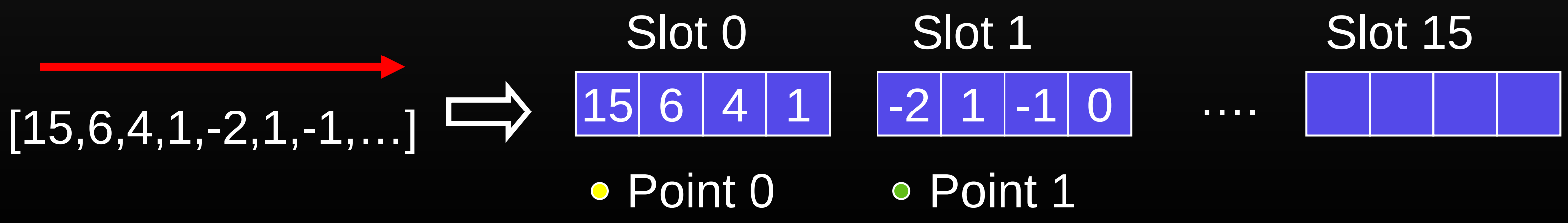
Batch points to feed GPUs

- Challenge
  - Various video block prediction or coding types
  - Irregular distribution and number of coefficients
  - Highly regular and well batched for GPU
  - Expensive branch penalty on GPU
- Solution
  - Divide and conquer
  - Use CPU to classify points into different sets

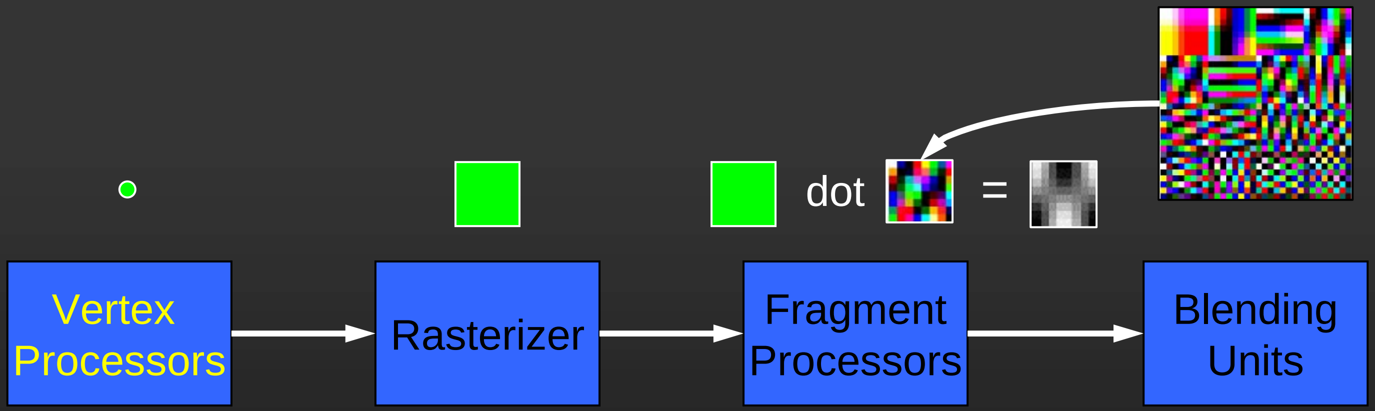
Coefficient Points



- Apply a regular pattern to generate points
  - Solve irregular distribution of coefficients
  - Only convey non-zero 4D Vector and its index
  - Balance visual quality and computation complexity

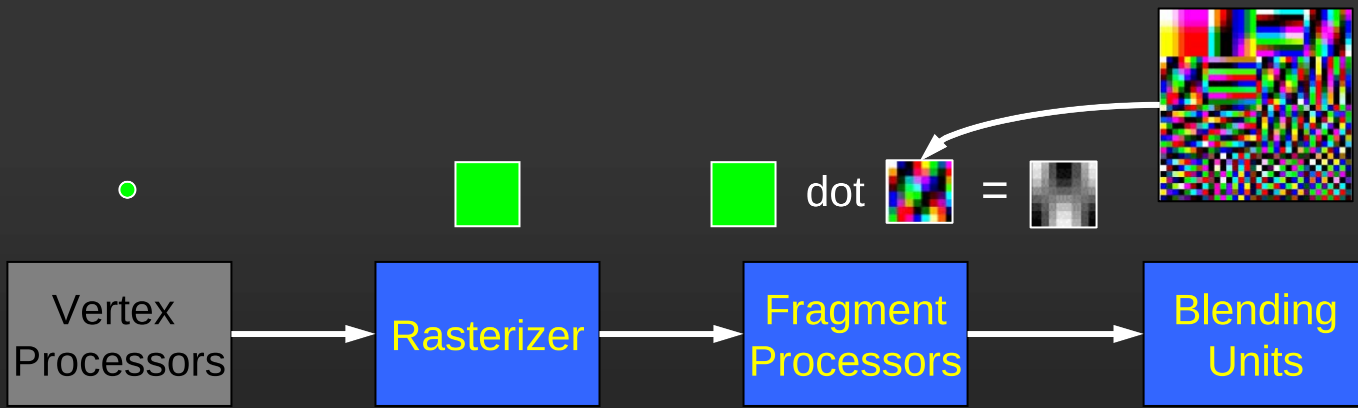


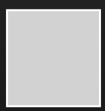
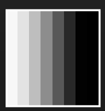
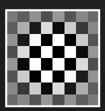
Render coefficient points (IQ)



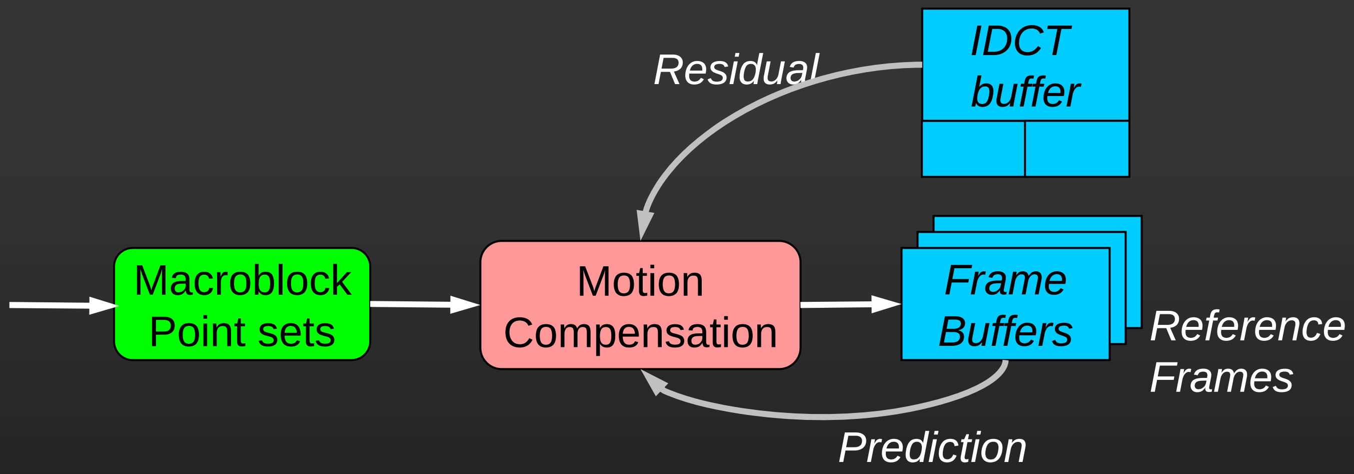
- Single pass to perform both IQ and IDCT
- Vertex processors:
  - Perform IQ $X_{IQ}(u,v) = X_Q(u,v) \times QM(u,v) \times qp$
    - Quant matrix as uniform parameters
    - Quant parameter and slot index in point's attributes
  - Locate coordinates of the basis image

Render coefficient points (IDCT)



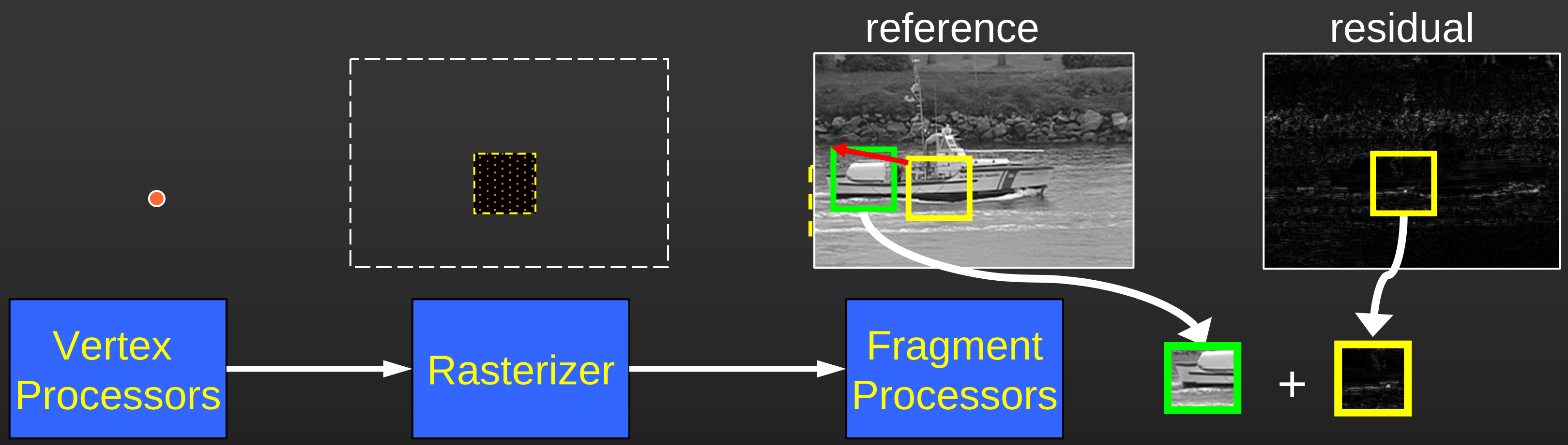
- IDCT:  $= X(0,0) \times$  $+ X(1,0) \times$  $+ \ldots.. + X(7,7) \times$ 
- Rasterizer: scalar-matrix →per-fragment
- Fragment processors: sample texels ; dot product
- Blending units: set function to *Add*
  - Accumulate the results from multi points

Macroblock points



- Arrange MB-points to different sets
  - According to different MB type (intra, forward, bidir…)
  - Convey MVs in point’s attributes
- Set texture access mode
  - Bilinear filter for sub-pixel MVs
  - *Clamp* address mode for unrestricted MVs

Render MB points (MC)



- Vertex processers
  - Output position and size
  - Preprocess MVs :
    - Set proper decimal parts
    - field prediction; field DCT
- Fragment processors:
  - offset *WPOS* with MVs
  - Sample textures
  - Sum and saturate

Outline

- Motivation and Goals
- Previous work
- Review of video decoding
- Point based decoding framework
- Results
- Discussion

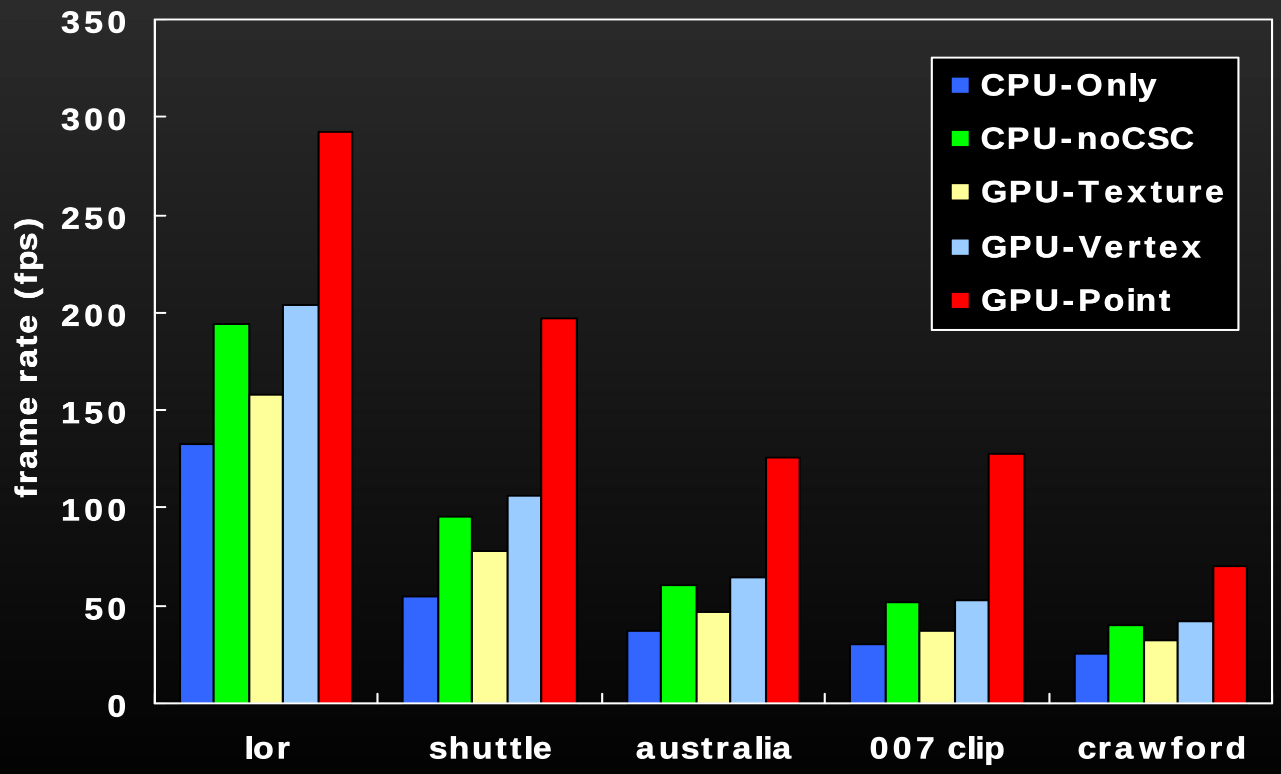
Evaluation Results

- Our experimental environment
  - 2.8G Pentium 4 with an Nvidia Geforce 6800GT
  - MPEG-2 decoder with OpenGL and Cg 1.4
- Five different implementations and test clips
  - CPU-only
  - CPU-noCSC
  - GPU-Texture
  - GPU-Vertex
  - GPU-Point

lor	480p	4.6Mbps
shuttle	720p	15.5Mbps
australia	1080i	12.3Mbps
007	1080p	10.9Mbps
crawford	1080i	30.0Mbps

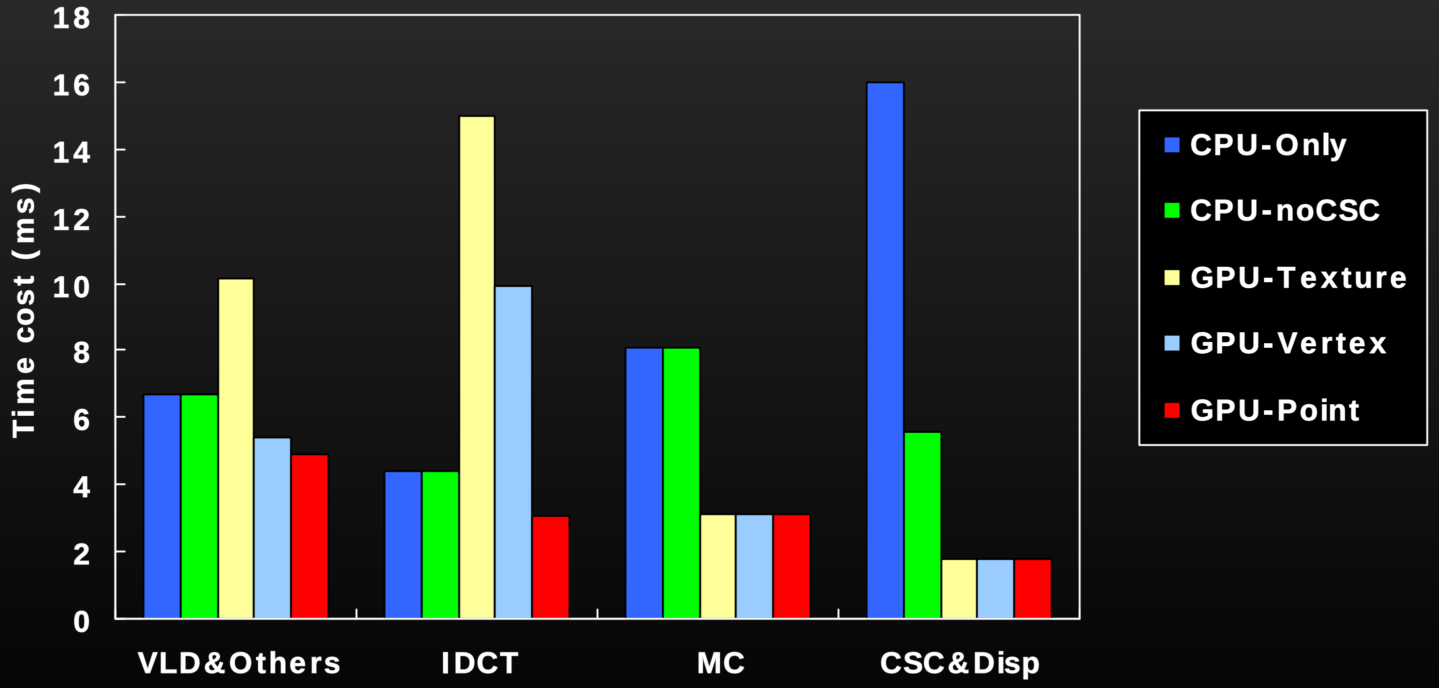
Performance

- Overall decoding frame rates
  - Significantly outperform other competitors



Performance

- Time costs of decoding stages
  - statistics on the clip *“australia” (1440x1080)*



Picture Quality

- Nearly degradation free of the quality
  - MPEG test sequences (CIF) GOP=15, 2.0Mbps
  - No drift-error accumulation observed
  - Slight degradation: different rounding control for sub-pixel interpolation (P and B frames)

Sequences	Average PSNR (db)	Y-PSNR Degradation (db) I	P	B
• stefan	31.722	0.006	0.008	0.021
• mobilecal	31.134	0.003	0.010	0.030
• foreman	37.245	-0.011	0.027	0.055

graphics hardware 06

Discussion

- Strength and advantages
  - Save bandwidth and computation
  - Fully utilize the graphics pipeline
  - Neat and flexible framework
- Weakness
  - High pixel fill-rate for performance
  - Floating point blending for precision
  - Constrain shape to be a square
  - Non-bilinear interpolation benefit less

Conclusion

- An efficient decoding framework on GPU
  - Analyze parallelism and features of decoding
  - Flexible point-based representation for video block
  - Efficient IQ, IDCT and MC by rendering points
  - Results demonstrate efficiency and flexibility
- Future work
  - Apply to more standards, even HDR video
  - Video encoding and transcoding

graphics hardware 06

Question

- Thanks for your attention…

....Question?

- Contact:
  - hanbo@icst.pku.edu.cn
  - zbf@pku.edu.cn