SIGGRAPH2009 NEW ORLEANS

Bucket Depth Peeling

Fang Liu   Mengcheng Huang   Xuehui Liu   Enhua Wu

Institute of Software Chinese Academy of Science

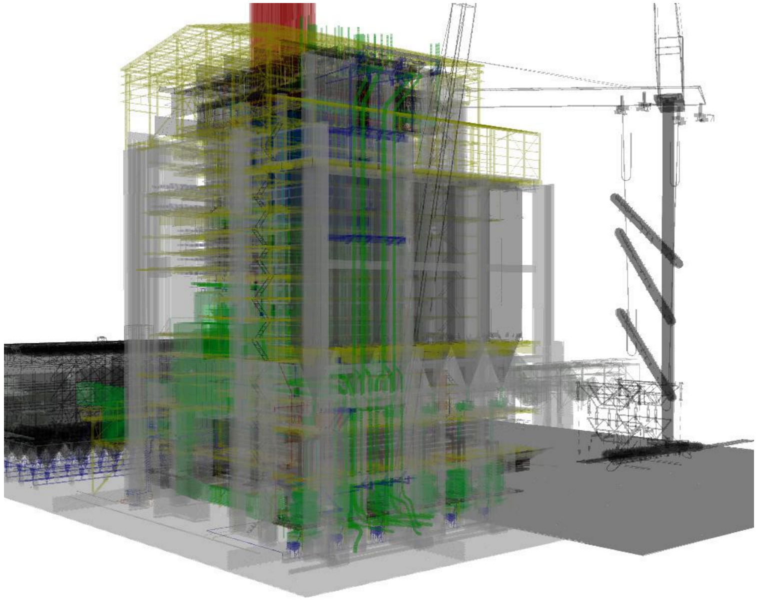
Background

- Multi-fragment effects

  - Operates on multiple fragments per pixel

- Less efficient in current graphics pipeline

  - Optimized to handle opaque surfaces



Related Work

- **Primitive level**
  - Painter's algorithm
  - Visibility ordering [Govindaraju et al. 05]
- **Fragment level**
  - A-Buffer [Carpenter 84]
  - Depth peeling [Mammen 84] [Everitt 01]
  - R-Buffer [Wittenbrink 01]
  - F-Buffer [Mark et al. 01]
  - K-Buffer [Bavoil et al. 07][Liu et al.06]
  - Stencil routed A-Buffer [Myers et al. 07]
  - Dual depth peeling [Bavoil et al. 08]
- **Hybrid methods**
  - Z-Batch [Wexler et al. 05]
  - Coherent layer peeling [Carr et al. 08]

Related Work

- **Depth Peeling**

  - A linear complexity algorithm to capture and sort multiple fragment in single pass

  - Multiple rasterizations of the scenes

- **K-Buffer**

  - Allocate a fix sized buffer per pixel

  - Capture and sort K fragments in single pass

  - Read Modify Write (RMW) hazards

Related Work

- **Stencil routed A-Buffer**

  - Capture fragments in MSAA buffer by stencil routing

  - Post-processing by bitonic sort

- **Dual depth peeling**

  - Peel the scene from front and back simultaneously

  - 2x speedup

Our Solution

- **Bucket depth peeling**

  - Peel off one layer in each geometry pass

  - Bucket sort fragments on GPU

  - No RWM hazards

  - Adaptive bucket peeling

Outline

- Bucket sort on GPU
- Bucket depth peeling
- Adaptive bucket peeling
- Results

Bucket Sort on GPU

- Fixed size buffer per pixel + data scattering
- Multiple Render Target (MRT)
  - 8 MRTs with format RGBA32F (Geforce 8800GTX)
  - A bucket array of size 32 per pixel
- Scatter: update channels of MRT

Explicit Update

- Explicitly write to a specific channel of MRT does not work

M4

0	0 M1	0	0 M2	0	0	M4	0	0	0 M3

The Correct Way

- 32bit Max/Min blending

  - Keep the greater/smaller value

- Guarantee correct results under concurrent update

  - Initial each slot to 0

  - Update other slots by 0

- Collisions: always keep one correct value

Concurrent Update

M1 M5 M4 M6 M2 M3

0	0	0	0	0	0	0	0	0	0

Outline

- Bucket sort on GPU
- Bucket depth peeling
- Adaptive bucket peeling
- Results

Bucket Depth Peeling

- Uniformly divide the depth range at each pixel location into 32 intervals
- A incoming fragment is mapped to the k^{th} bucket according to

$$k = floor\left(\frac{32 \times (d_f - zNear)}{zFar - zNear}\right)$$

- zNear and zFar

  - Exact: Too expensive

  - Approximate: Bounding box or visual hull

Bucket Depth Peeling

- The k^{th} bucket: fragment with Maximum depth value in the k^{th} subinterval
- Non-empty buckets already in correct depth ordering

Bucket Grouping

- Group the bucket array into 16 pairs
- Depth range: 16 sub-intervals
- An incoming fragment is mapped to the k^{th} bucket pair according to

$$k = floor(\frac{16 \times (d_f - zNear)}{zFar - zNear})$$

- Update the k^{th} pair of buckets by $(1 - d_f, d_f)$ simultaneously

Bucket Grouping

- The k^{th} pair of buckets: fragments with minimum and maximum depth values in the k^{th} subinterval

$$d\min_k^1 = 1 - \max_{df \in [d_k, d_{k+1})} (1 - d_f)$$

$$d\max_k^1 = \max_{df \in [d_k, d_{k+1})} (d_f)$$

- Non-empty buckets already in correct depth ordering
- Potentially less collision

Bucket Depth Peeling

- Ideal for uniform distributed scenes
- Non-uniform scenes: Multi-pass approach

  - Sparse layout may cause memory exhaustion
- Multiple fragment attributes: Mismatch
- Uniform division does not consider layer distribution

Outline

- Bucket sort on GPU
- Bucket depth peeling
- Adaptive bucket peeling
- Results

Adaptive Bucket Peeling

- Adapt the partition of the subintervals to the distribution of the fragments

  - One fragment in one bucket

- Cost: Addition geometry pass to obtain layer distribution information

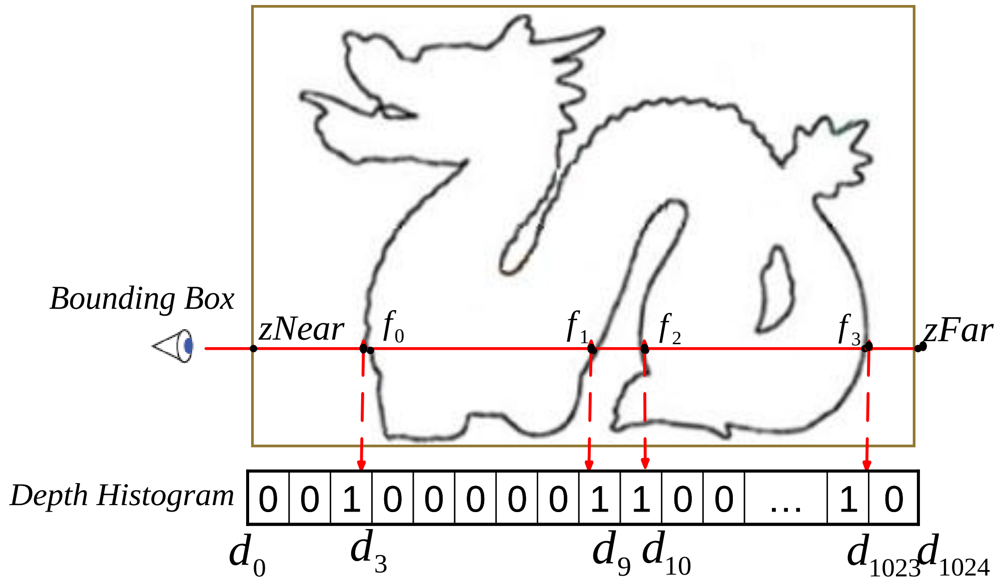
Depth Histogram

- Encode layer distribution
- Interpret 8 MRT as a bit array of 1024 bits
- Depth range: 1024 intervals
- Each bit indicates the presence of fragments

The Algorithm

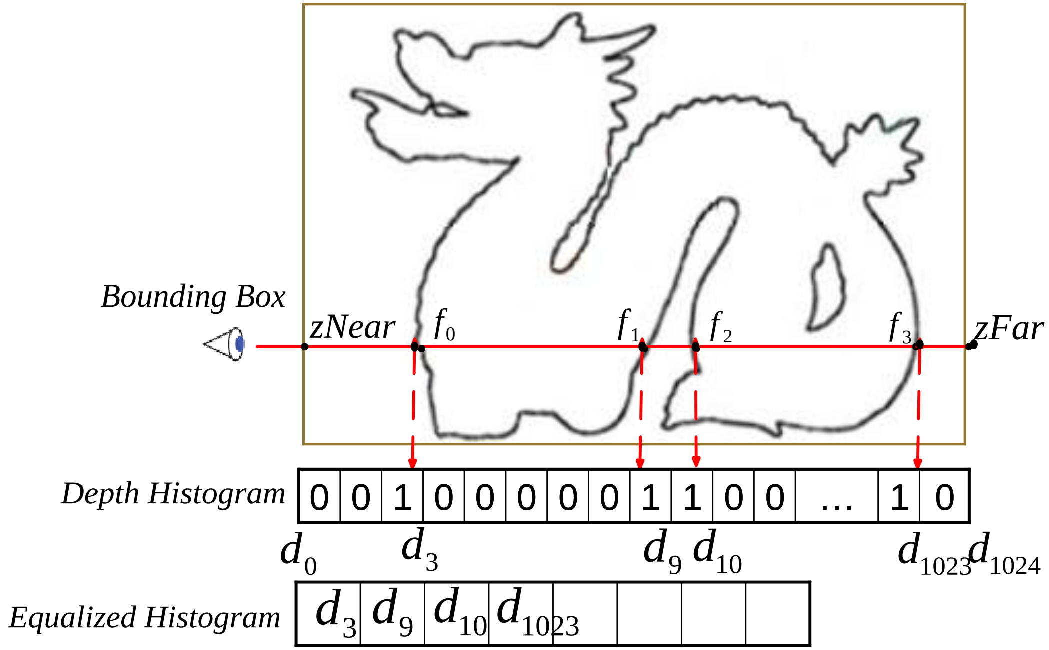
- Create Depth histogram

  - Set the bit using logical operation OR



The Algorithm

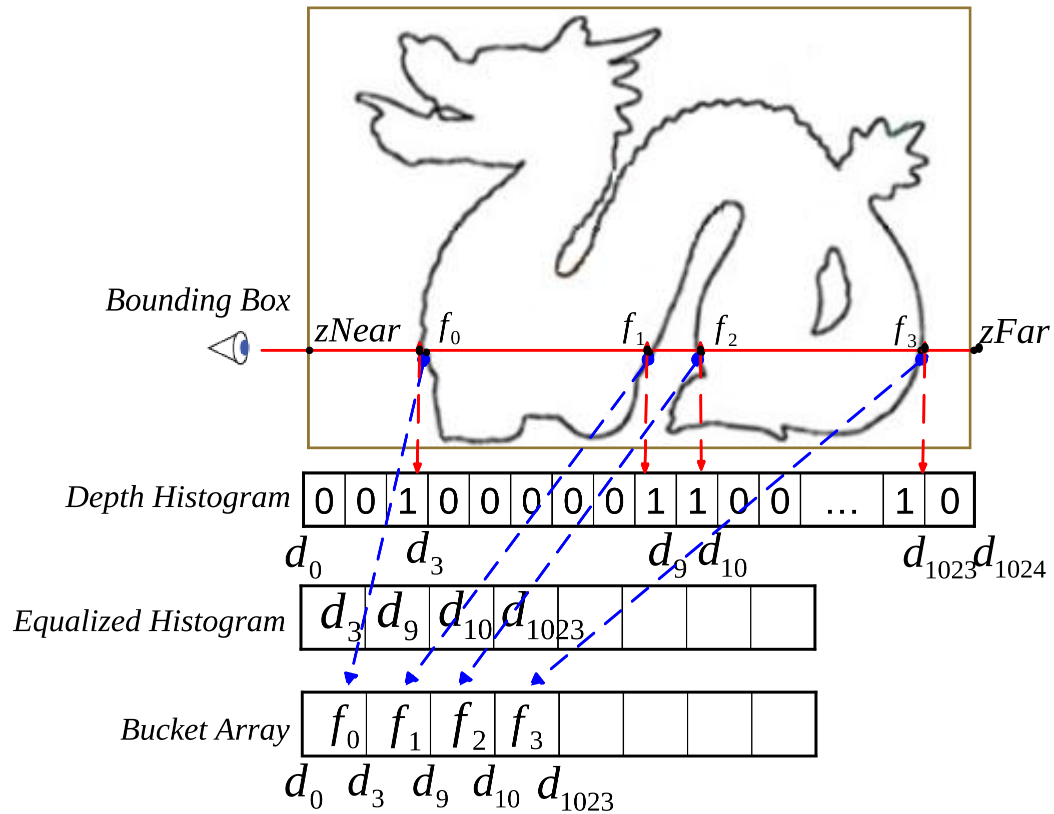
- Equalize the depth histogram
  - Scanning for non-zero bits
  - Store the upper bound of non-empty intervals



The Algorithm

- Perform Bucket Sorting

  - Guided by equalized histogram



Adaptive Bucket Peeling

- Reduce collisions substantially by adaptive partitioning
- Fragment attributes consistency
- Two geometry pass + one screen pass

Outline

- Bucket sort on GPU
- Bucket depth peeling
- Adaptive bucket peeling
- Results

Results

- Implemented using OpenGL and Cg shading language ver.2.1
- All results generated on a commodity PC of Intel Duo Core 2.4G Hz with Nvidia Geforce 8800 GTX

Results

Transparent Effect on Stanford Dragon
(871K triangles)

Results

Translucent Effect on Buddha
(1.0M triangles)

Results

Translucent Effect on Stanford Dragon
(871K triangles)

More Results

Transparent Effect on UNC Powerplant (12.7M triangles)

More Results

Transparent Effect on Six Dragons
(5.2M triangles)

More Results

Transparent Effect on Bunny
with 2778 Spheres

Performance

- Up to N times speedup for large scenes with depth complexity N
- Performance degrades for small models

Model	Dragon	Buddha	Powerplant	Lucy	Stpauls
Tri No.	871K	1,087K	12,748K	28,055K	14K
BDP	256fps	212fps	24.15fps	10.93fps	434fps
BDP2	128fps	106fps	12.79fps	5.71fps	256fps
ADP	106fps	91fps	12.31fps	5.37fps	212fps
K-buffer	206fps	183fps	23.98fps	10.49fps	468fps
[Liu 2006]	49fps	39fps	0.83fps	0.75fps	155fps
	5g	6g	27g	14g	22g
Dual DP	37fps	32fps	1.34fps	0.87fps	199fps
	8g	8g	16g	12g	14g
DP	24fps	20fps	0.76fps	0.54fps	242fps
	13g	13g	32g	21g	26g

Quality Analysis

- Error measure with EPR

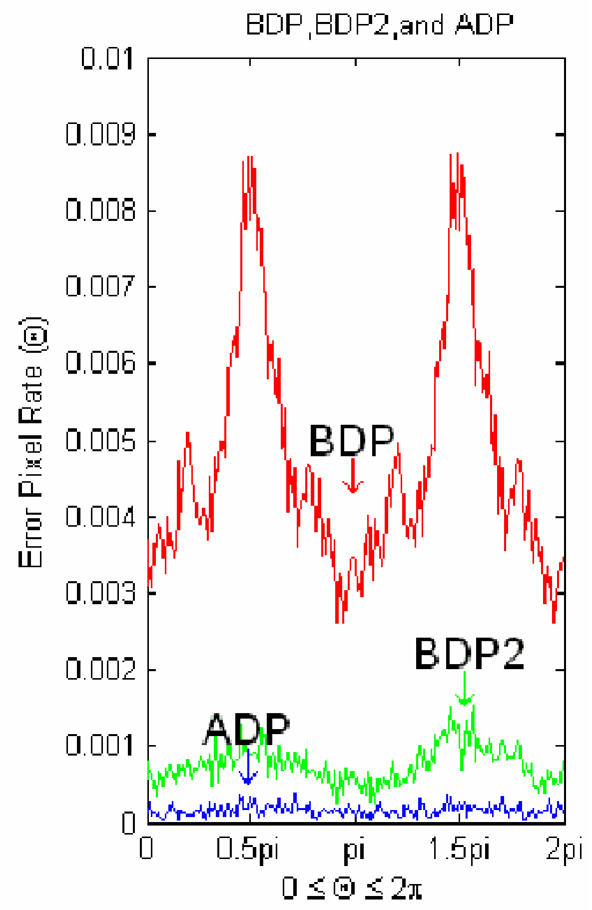
$$EPR = \frac{N_{Err}}{N_{Total}}$$

- EPR for our algorithm

- BDP1: 0.3% - 0.9%

- BDP2: 0.05% - 0.15%

- ADP: 0.01%



Limitations

- Approximate

  - Limited resolution for depth histogram

  - Artifacts appear at sharp edges or details of the model

- Memory overhead

  - 8 MRT textures for depth histogram and 8 MRT for bucket array

  - High screen resolutions

Conclusions

- Bucket depth peeling

 - A novel linear complexity approach to capture and sort multiple fragments on GPU

 - An adaptive two-pass approach that greatly reduce fragment collisions

 - Great speedup to depth peeling

Future Work

- Design a good mapping function with lower collision rate
- Explore programmable graphics pipeline

  - “Single Pass Depth Peeling using CUDA Rasterizer” at SIGGRAPH 2009 talks

  - Exact solution in single pass

Thank You !

Questions ?

Mengcheng Huang <hmcen@ios.ac.cn>