Adaptive Undersampling for Efficient Mobile Ray Tracing

Youngwook Kim[1] Woong Seo[1] Yongho Kim[2]
Yeongkyu Lim[3] Jae-ho Nah[3] Insung Ihm[1]

[1] Sogang University, Korea [2] NCSOFT, Korea [3] LG Electronics, Korea

Motivation

- **Interactive ray tracing on the mobile GPU**
  - A high screen resolution such as QHD (2,560x1,440) is nowadays common for mobile devices.
  - Without a specialized mobile accelerator, interactive ray tracing is still beyond the processing power of available processors.
  - **Optimizing the mobile ray tracing computation** remains essential when images of nontrivial pixels are to be ray-traced interactively.

Kitchen (101K triangles)

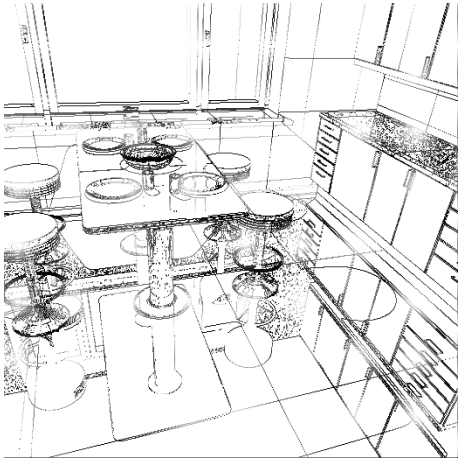
Bathroom (269K triangles)

Full ray tracing time using kd-tree on an Adreno 420 GPU (ms)

	Kitchen	Bathroom
512x512	356.0	319.4
1,024x1,024	1,296.9	1,169.5
2,048x2,048	4,777.8	4,293.7

- **Adaptive undersampling**
  - Aim to minimize the total number of ray shootings while introducing only a small reduction in ray-tracing quality.
  - **Fire fewer than one ray per pixel adaptively**.
    - Shoot rays through **problematic pixels** while applying cheaper interpolation for the remaining pixels.

Full ray tracing (1,024x1,024)

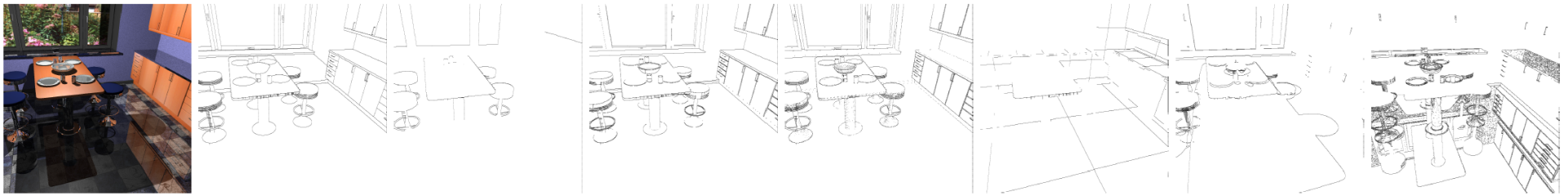


Ray-traced pixels (34.5%)

Adaptive ray tracing
(1.56x faster, PSNR 39.14)

Our Contributions

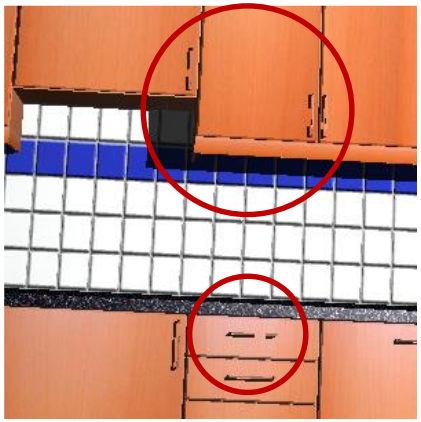
- **Present an adaptive undersampling technique** well-suited to effective im plementation of a mobile GPU ray tracer.
  - **Similarity checks to decide problematic pixels** based upon various pixel attrib utes collected during rendering
    - Explore both image-space color measures and object-space geometry attributes.
    - Enable users to focus ray tracing efforts on specific rendering features.



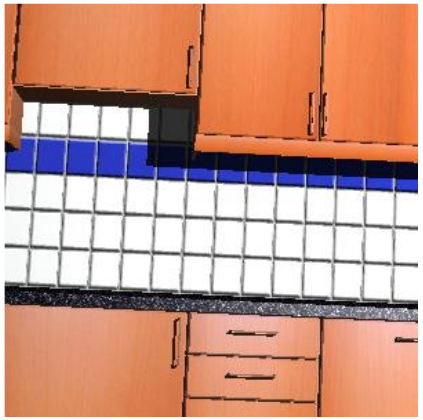
Problematic pixels found through seven similarity check measures

  - **A seven-kernel GPU implementation** of the undersampling algorithm **with a si mple control structure**
    - Allow an efficient computation on the mobile GPU that is more vulnerable to the com plexity of parallel algorithm.

- **Present a low-cost postcorrection method** to reduce the occurrence of aliases due to insufficient pixel sampling.
  - An optional postprocessing algorithm **to correct missing parts**
    - Effectively handle the missing object problem occurring in the interpolated adaptive pixels.



Before correction



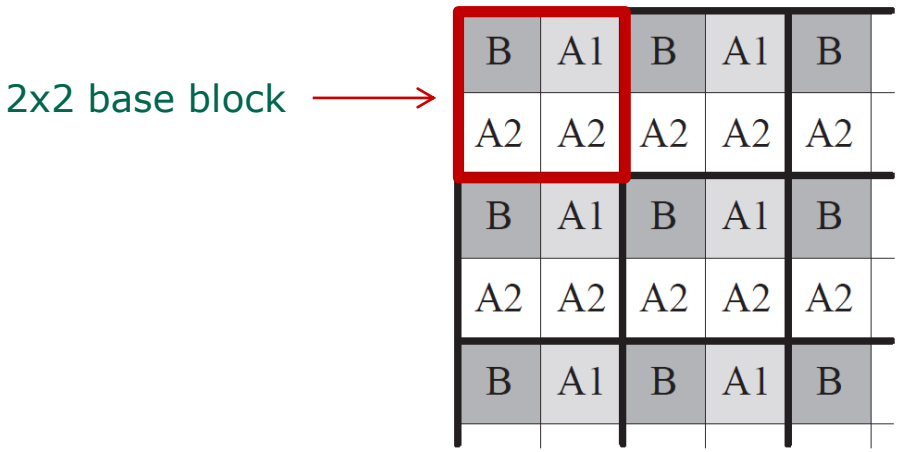
After correction

Full ray tracing

  - **A GPU efficient implementation** of the correction algorithm
    - Propose a multi-step pixel scanning technique designed for the GPU's many-core processing.

Two-level Pixel Partitioning for Efficient Undersampling

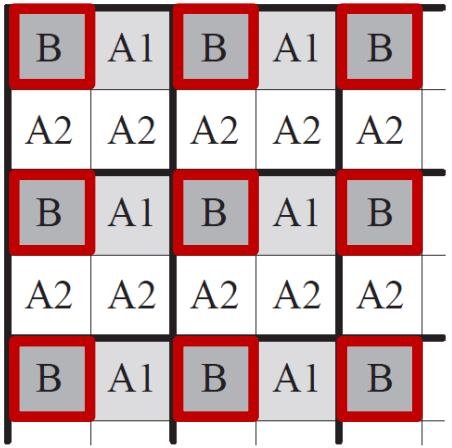
- **Three types of pixels in the 2x2 base blocks**
  - **Base pixels (B):** always ray-traced
  - **Adaptive pixels:** ray-traced or interpolated
    - Vertical (A1)
    - Horizontal (A2)



- **Fixed-order pixel traversal for a simple control structure**
  - Process pixels **in parallel** in the order of B pixels → A1 pixels → A2 pixels.
  - Permit efficient implementation on a mobile GPU platform.

Three-step Adaptive Undersampling Algorithm

- **Stage I : Regular sampling of base pixels**
  - *Trace a ray in parallel through each of regularly distributed B pixels.*
  - During processing each pixel, **collects various ray attributes at the first hit of the ray**.
    - Object identification number (OID)
    - Position vector (POS)
    - Normal vector (NORM)
    - Shadow bits (SHDBIT)
    - Texture coordinates (TCOORD)
    - Global shaded color (GCOL)



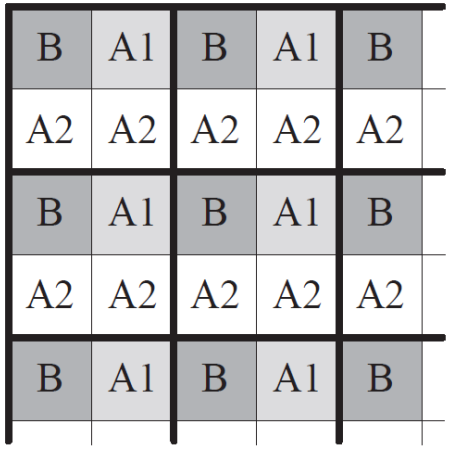
Stage I

➢ The final shaded color of a pixel can always be produced from its ray attributes.

- **Stage II: Adaptive sampling of vertical adaptive pixels**
  - *Process each A1 pixel in parallel.*
    - Take the attribute vectors of two horizontally adjacent B pixels as input.
    - Compute the attribute vector of the current A1 pixel
      - via **expensive ray tracing** or **cheap linear interpolation**.

  ➢ A set of elementary tests called **similarity checks** will provide an answer on which operation to be selected.

- **Stage III: Adaptive sampling of horizontal adaptive pixels**
  - *Process each A2 pixel in parallel.*
    - Take the attribute vectors of two vertically adjacent B or A1 pixels as input.
    - Compute the attribute vector of the current A2 pixel
      - via expensive ray tracing or cheap linear interpolation.



Stage II & III

Similarity Checks

- **A series of elementary tests** made with the ray attributes of two reference pixels
  - Tell if the ray attributes of the current pixel may safely be interpolated linearly from those of the reference pixels.
  - Perform a linear interpolation only if all the tests succeed.
  - Must be **simple and effective** to pursue high efficiency on the **mobile** GPU.

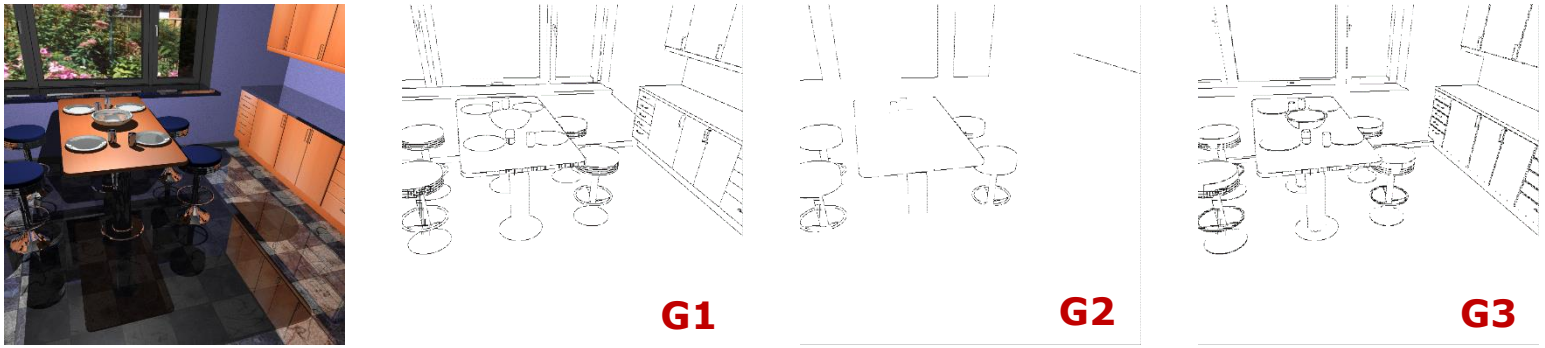
- **Four kinds of tests**
  - Four local geometry tests (G1, G2, G3, G4)
  - A texture test (TX)
  - A shadow test (SH)
  - A global color test (GC)

	Test conditions
G1	$\mathrm{OID}_0 = \mathrm{OID}_1$
G2	$\|\|\mathrm{POS}_0 - \mathrm{POS}_1\|\| \leq T_{pos}$
G3	$\mathrm{NORM}_0 \circ \mathrm{NORM}_1 \geq T_{norm}$
G4	$\{(\mathrm{POS}_1 - \mathrm{POS}_0) \circ \mathrm{NORM}_0\} \cdot \{(\mathrm{POS}_0 - \mathrm{POS}_1) \circ \mathrm{NORM}_1\} \geq 0$
TX	$\mathrm{TCOORD}_\alpha.\beta \geq T_{tex}$ or $\mathrm{TCOORD}_{1-\alpha}.\beta \leq 1.0 - T_{tex}$
SH	$\mathrm{SHDBIT}_0 = \mathrm{SHDBIT}_1$
GC	$\|\| \mathrm{GCOL}_0 - \mathrm{GCOL}_1 \|\| \leq T_{gcol}$

The subscripts 0 and 1 respectively denote two reference pixels.

- **Four local geometry tests**
  - **G1:** check if the two object IDs are identical.
  - **G2:** check if the two first hits are sufficiently close.
  - **G3:** check if the two normal directions at the first hits are sufficiently similar.

	Test conditions
G1	$\mathrm{OID}_0 = \mathrm{OID}_1$
G2	$\|\|\mathrm{POS}_0 - \mathrm{POS}_1\|\| \leq T_{pos}$
G3	$\mathrm{NORM}_0 \circ \mathrm{NORM}_1 \geq T_{norm}$

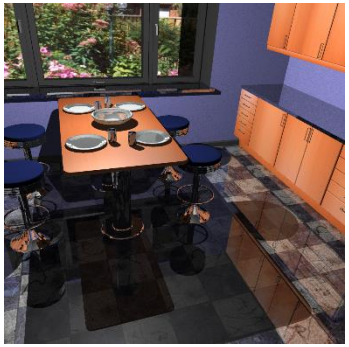


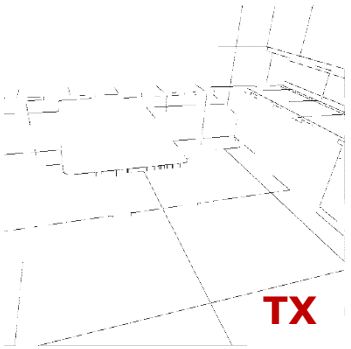
Problematic adaptive pixels detected by each similarity check

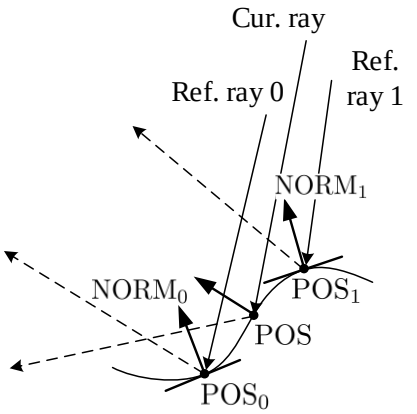
– **G4 (convexity check):** check if there possibly is **a surface fluctuation** between the two reference pixels.

- **Effective for removing normal-related aliasing** caused during local shading and secondary ray generation.

	Test conditions
G4	$\{(\mathrm{POS}_1 - \mathrm{POS}_0) \circ \mathrm{NORM}_0\} \cdot \{(\mathrm{POS}_0 - \mathrm{POS}_1) \circ \mathrm{NORM}_1\} \geq 0$







A troublesome situation detected by G4

- **A texture test**
  - **TX:** check if the two texture coordinates exist across a texture boundary.
    - Effective particularly when a texture is repeatedly applied.

	Test conditions
TX	$\mathrm{TCOORD}_{\alpha}.\beta \geq T_{tex}$ or $\mathrm{TCOORD}_{1-\alpha}.\beta \leq 1.0 - T_{tex}$

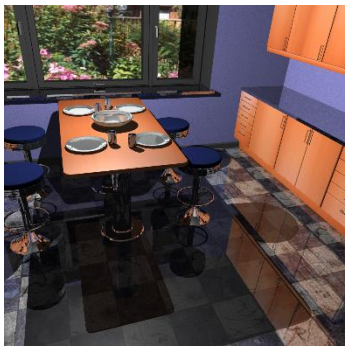
(α can be either 0 or 1, and β refers to either the s or t texture coordinate.)

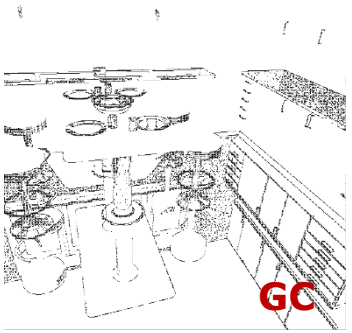
- **A shadow test**
  - **SH:** check if all the corresponding shadow flags at the two hit points are identical.

	Test conditions
SH	$\mathrm{SHDBIT}_0 = \mathrm{SHDBIT}_1$

- **A global color test**
  - **GC:** check if the two reflection/refraction colors are sufficiently similar.
    - Any kind of global color could be compared in the extended work.

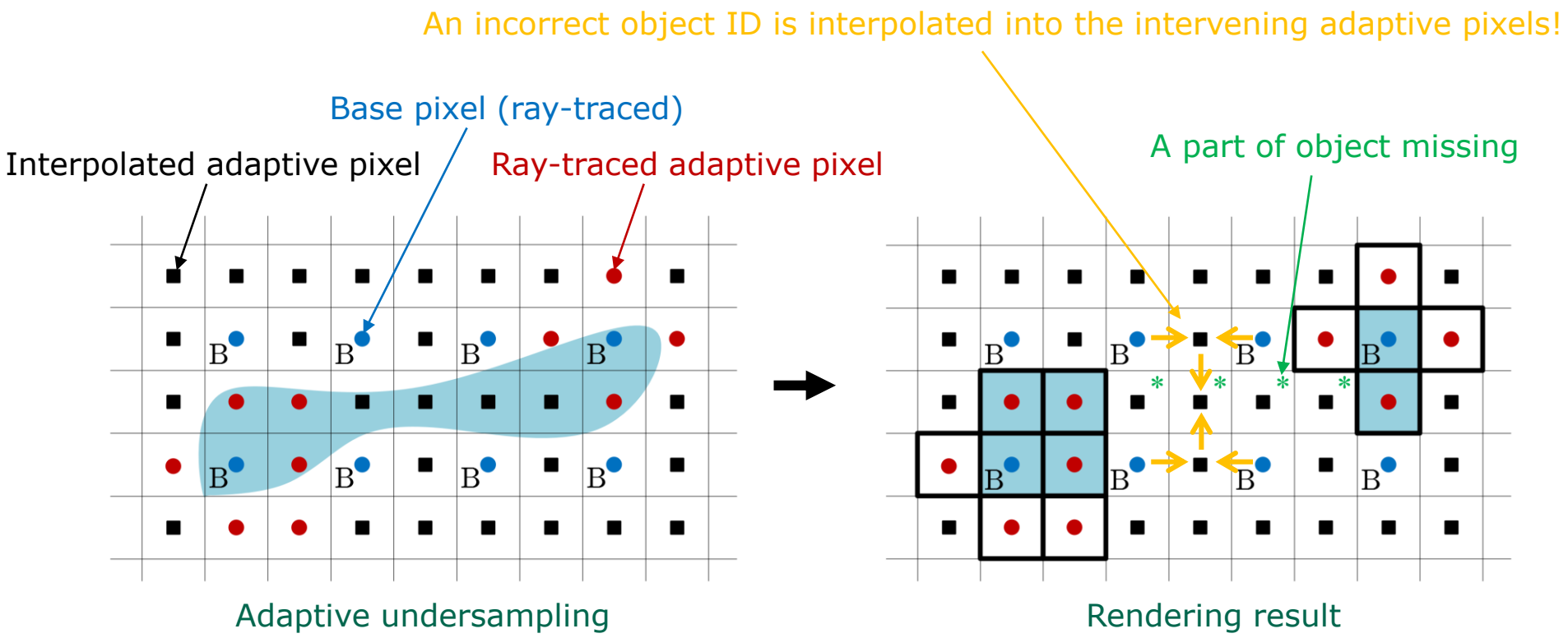
	Test conditions
GC	$\| \mathrm{GCOL}_0 - \mathrm{GCOL}_1 \| \leq T_{gcol}$



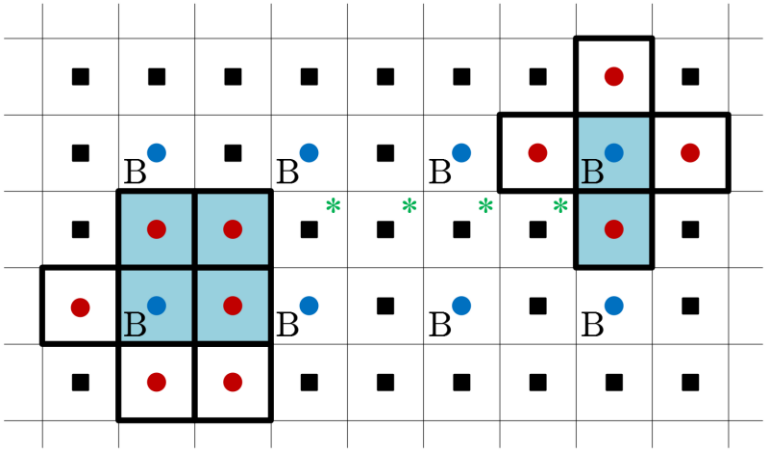


Postcorrection of Undersampled Images

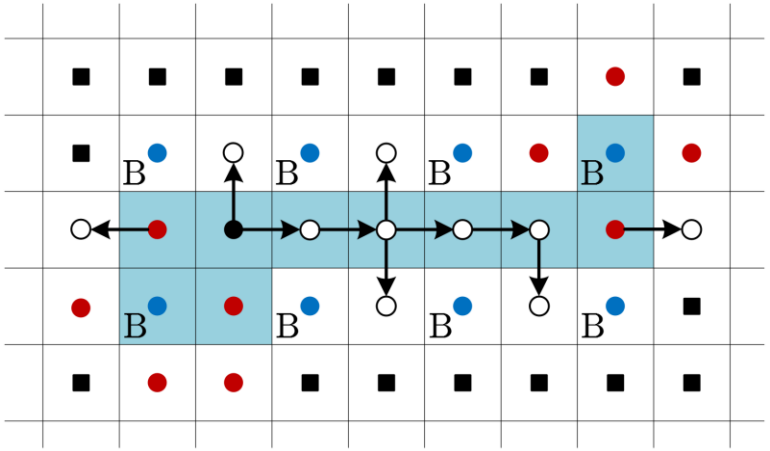
- **Missing object problem**
  - Due to insufficient sampling, objects, or parts of objects, can **fall between ray-traced samples and be missed**.



- **Observation**
  - The problem always occurs in the **interpolated adaptive** pixels that can be traced from **ray-traced adaptive** pixels.
- **Solution**
  - Starting from **each ray-traced adaptive pixel** (●), propagate its correct ray-object intersection information into **its interpolated neighbors**.
    - When a neighboring pixel has a different object ID, fire a ray through the pixel.
    - If the new object ID differs from the old one, **repeat the propagation process from that pixel**.



Before correction



Propagation of the object ID information

- **Implementation**
  - **An 8-neighbor propagation** versus **a 4-neighbor propagation**
    - For the 2x2 base block, the 4-neighbor propagation is sufficiently effective.
  - **GPU-friendly propagation**
    - Each concurrent thread scans an associated row from left to right, and then right to left, correcting problematic pixels progressively.
    - Then, each concurrent thread scans an associated column from top to bottom, and then bottom to top, correcting problematic pixels progressively.
    - **Scanning in just two orthogonal directions** produces sufficiently good results.
- **Note**
  - Our correction method is **selective**.
    - For instance, "missing shadow" can be reduced by checking the shadow flags.
  - The separation of error correction from adaptive sampling allows **a simpler, GPU-friendly implementation**.
    - Different from the previous approach such as pixel selected ray tracing (Akimoto et al. 1991)

GPU Implementation of Adaptive Undersampling

- **Seven-kernel Implementation**

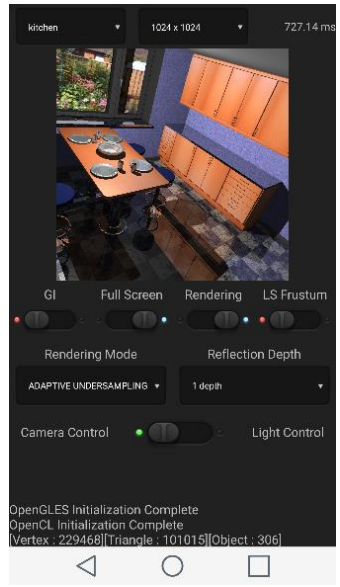
Kernel	Operations
Step-I	• Trace a ray for each B pixel, and shade it. • Store the ray attribute vectors of the B pixels in global memory.
Step-II-a	• Perform the similarity checks for each A1 pixel. • If they pass, interpolate the attribute vector, and shade the pixel. • If not, store the address of the current pixel in global memory.
Step-II-b	• **Pack the A1 pixels to be ray-traced through parallel scans.**
Step-II-c	• Same as Step-I except that the packed A1 pixels are processed.
Step-III-a	• Same as Step-II-a except that the A2 pixels are processed.
Step-III-b	• **Same as Step-II-b except that the A2 pixels are processed.**
Step-III-c	• Trace a ray for each packed A2 pixel, and shade it.

  - To avoid the branch divergences between concurrent threads,
    - Fire a ray through the adaptive pixel if there is at least a failure in the similarity checks.
    - Pack the pixels to be ray-traced into a contiguous region through a parallel scan operation.

Experiments

- **Compared ray tracing algorithms**
  - **Full ray tracing:** a kd-tree-based full Whitted-style ray tracing
  - **Ours:** a kd-tree-based adaptive undersampling
    - Applied tolerance values: T_{pos}= 0.03, T_{norm}= 0.9, T_{tex}= 0.3, T_{gcol}= 0.15
  - ➢ Based on OpenCL 1.2 (workgroup size = 8x8)

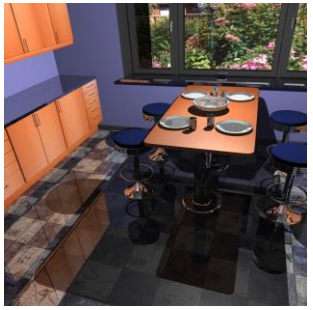
- **Tested mobile platform:** an LG G3 Cat.6 mobile phone
  - A Qualcomm Snapdragon 805 chipset with an Adreno 420 GPU



- **Tested scenes and camera views**


Café (29,359)


Ben (78,039)


Kitchen (101,015)


Conference (190,947)


Bathroom (268,725)


San Miguel (588,402)

Performance on Rendering 1,024x1,024 Images

Scene	Full RT	Ours			
	Time (ms)	Time (ms)	Speedup (x)	RT ratio	PSNR (dB)
Café	1,211.7	591.2	2.04	0.294	43.99
Ben	1,275.8	575.2	2.21	0.270	45.46
Kitchen	1,296.9	828.0	1.56	0.345	39.14
Conference	841.6	508.1	1.65	0.300	48.92
Bathroom	1,169.5	733.6	1.59	0.318	40.54
San Miguel	1,344.3	905.4	1.48	0.340	39.16

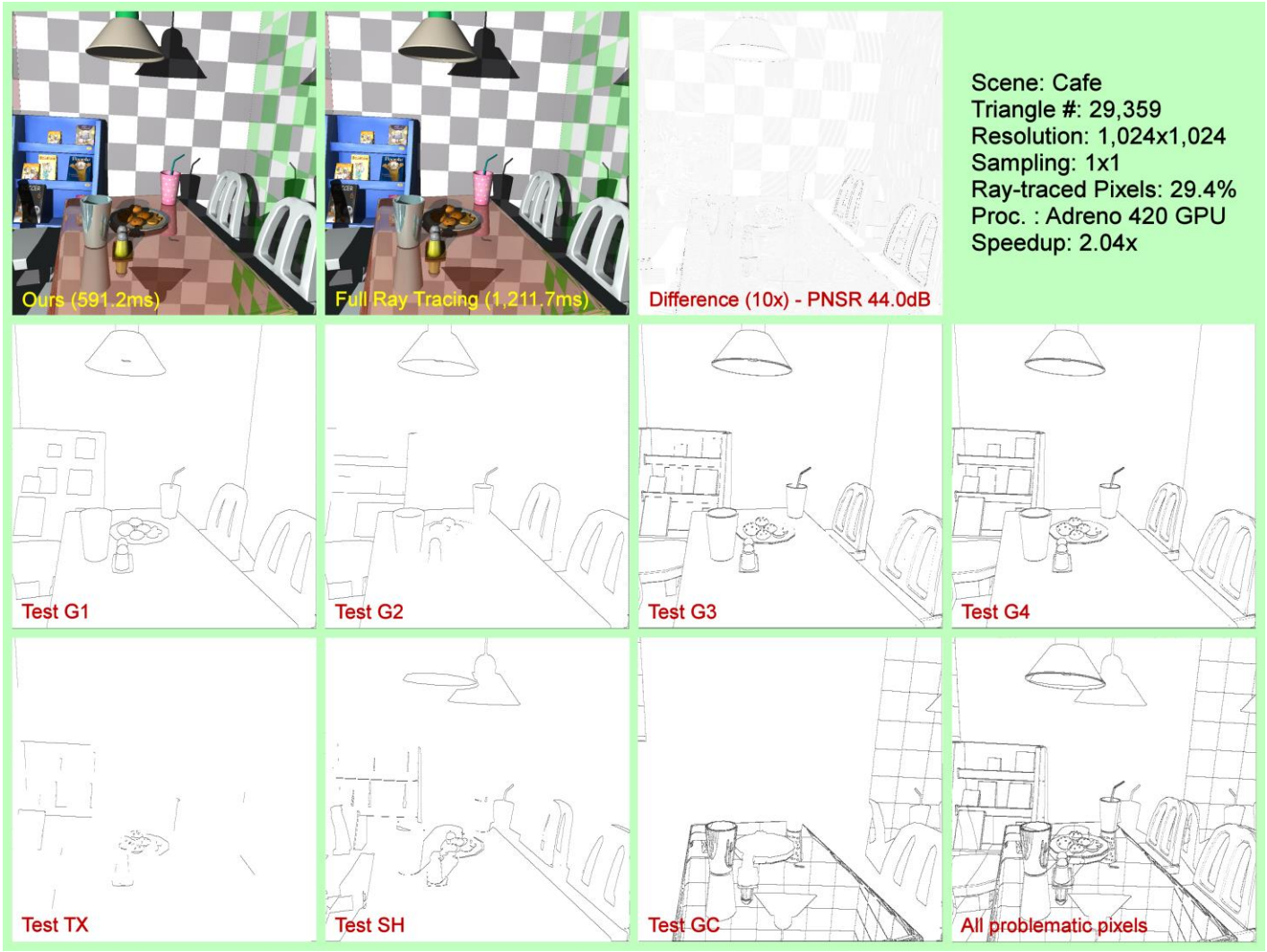
Ours is **1.48 to 2.21 times faster** when the 2x2 base block was used.

Only **27.0% to 34.5%** of image pixels **including base pixels** were actually ray-traced.

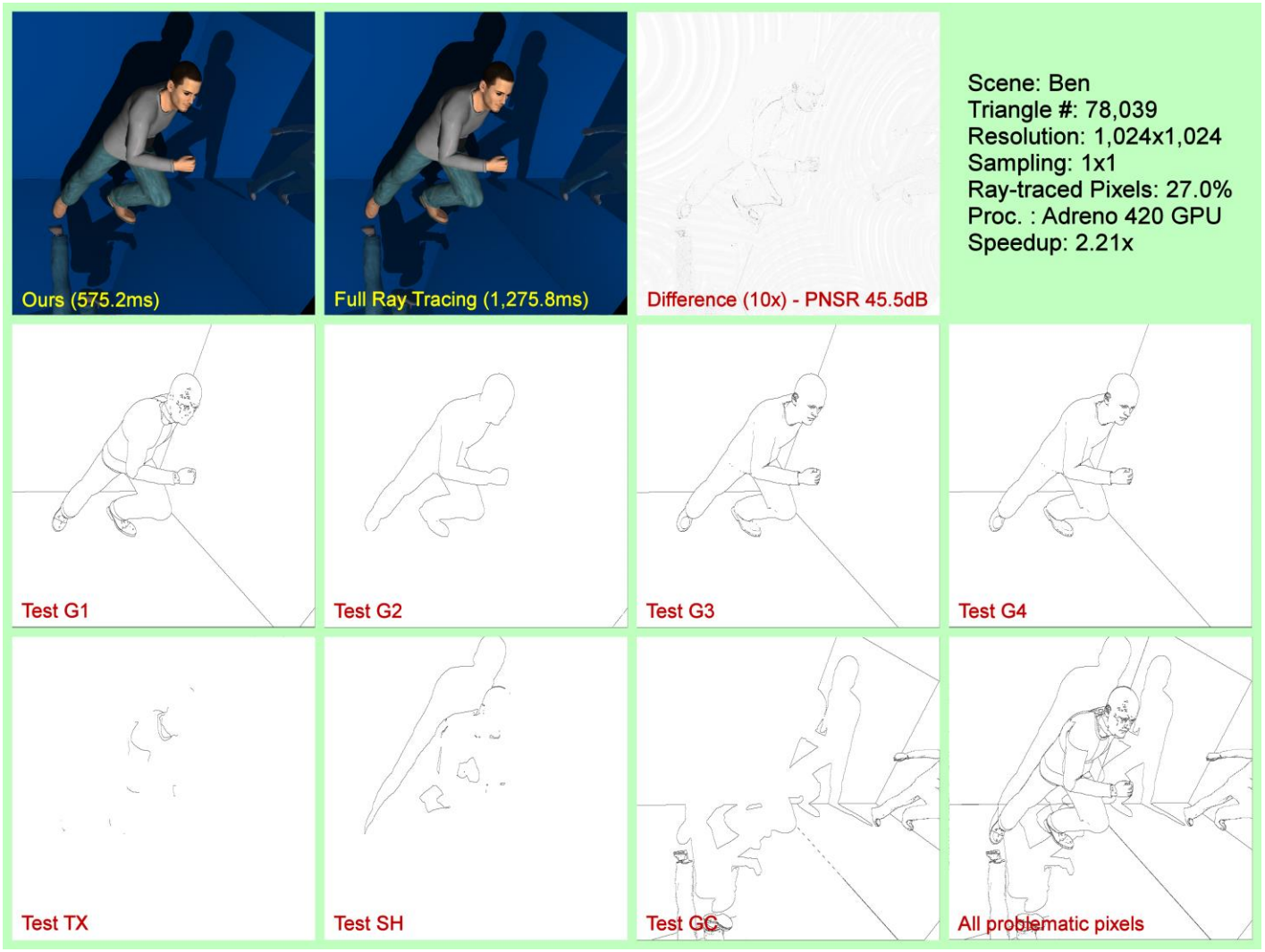
Good image quality is maintained despite the reduced numbers of ray shots.

Detected Problematic Adaptive Pixels

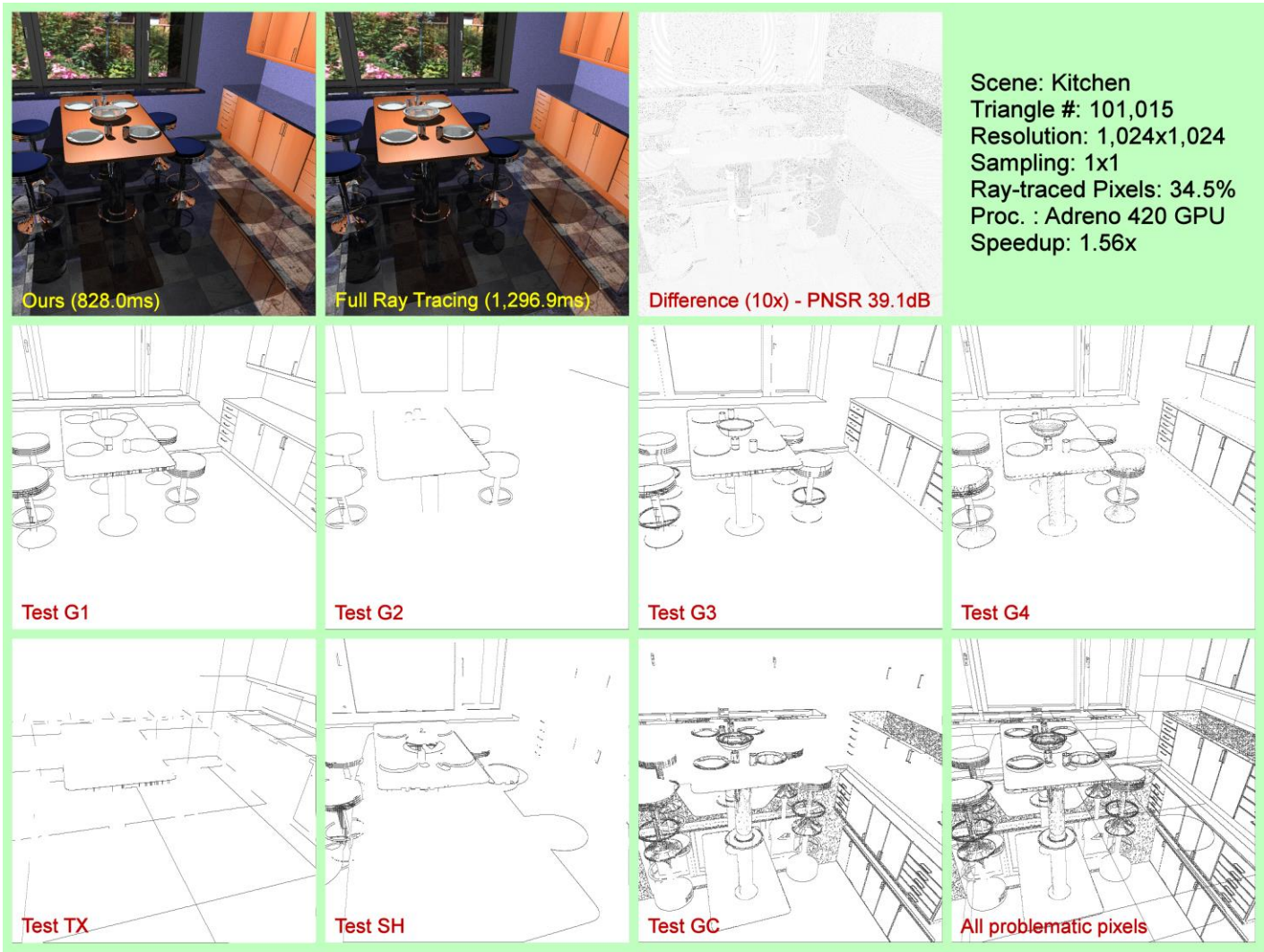
- Café (4.4% + 25.0%)



- **Ben (2.0% + 25.0%)**


Ours (575.2ms)
Full Ray Tracing (1,275.8ms)
Difference (10x) - PNSR 45.5dB
Scene: Ben
Triangle #: 78,039
Resolution: 1,024x1,024
Sampling: 1x1
Ray-traced Pixels: 27.0%
Proc. : Adreno 420 GPU
Speedup: 2.21x
Test G1
Test G2
Test G3
Test G4
Test TX
Test SH
Test GC
All problematic pixels

- **Kitchen (9.5% + 25.0%)**



- **Conference (5.0% + 25.0%)**

- **Bathroom (6.8% + 25.0%)**

- **San Miguel (9.0% + 25.0%)**

Dissection of Kernel Execution Time (ms)

Ray tracing: I, II-c, III-c
Similarity checks & Interpolation: II-a, III-a
Pixel packing (parallel scan): II-b, III-b
Postcorrection: IV
Data transfer & Kernel Launch: ETC

Scene	I	II-a	II-b	II-c	III-a	III-b	III-c	IV	ETC
Café	284.3	25.9	10.4	53.7	38.5	11.1	94.6	60.8	11.8
Ben	329.8	25.4	11.5	40.2	36.8	10.7	66.7	44.3	9.8
Kitchen	341.7	26.3	10.2	99.4	37.5	10.7	207.1	81.9	13.2
Conference	206.4	23.8	10.0	38.7	37.3	11.6	81.8	86.7	11.6
Bathroom	300.7	27.0	10.0	84.9	37.1	11.1	185.2	66.5	11.0
San Miguel	362.7	24.0	10.7	126.0	34.3	11.0	198.1	130.0	8.7

Postcorrection is optional, which can often be turned off for such scenes as Café, Ben, and Conference.

Ray tracing still accounts for a large portion of rendering time. (Conference: 64.3%, Kitchen: 78.3%)
This fact paradoxically shows the importance of adaptive undersampling **on the mobile GPU**.

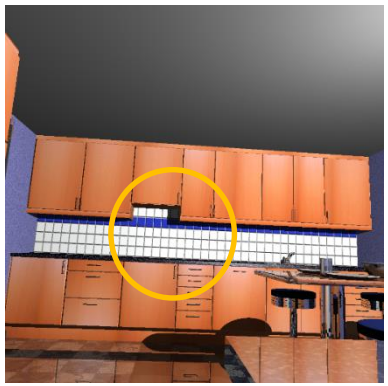
Performance on Supersampling 1,024x1,024 Images

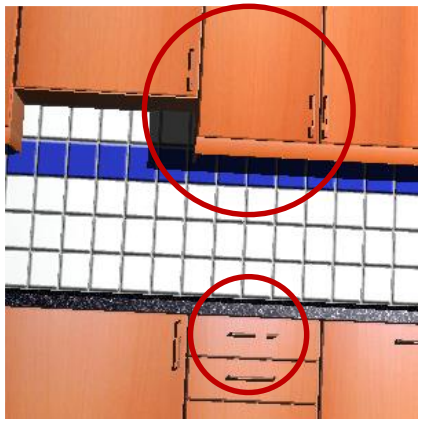
Sampling	Scene	Full RT	Ours			
		Time (ms)	Time (ms)	Speedup (x)	RT ratio	PSNR (dB)
2x2	Café	4,396.7	2,079.5	2.11	0.274	44.03
	Ben	4,586.9	1,946.6	2.35	0.260	49.55
	Kitchen	4,796.9	2,656.5	1.80	0.312	45.22
	Conference	3,284.0	1,803.9	1.82	0.277	50.35
	Bathroom	4,331.8	2,561.0	1.69	0.294	46.41
	San Miguel	4,833.8	2,923.6	1.65	0.300	43.18
4x4	Café	16,663.8	7,382.3	2.25	0.263	44.26
	Ben	18,007.2	7,230.0	2.49	0.255	51.66
	Kitchen	18,244.1	8,978.5	2.03	0.291	48.83
	Conference	12,700.4	6,729.4	1.88	0.265	51.58
	Bathroom	16,759.8	8,480.6	1.97	0.280	49.12
	San Miguel	17,917.2	9,987.0	1.79	0.278	48.08

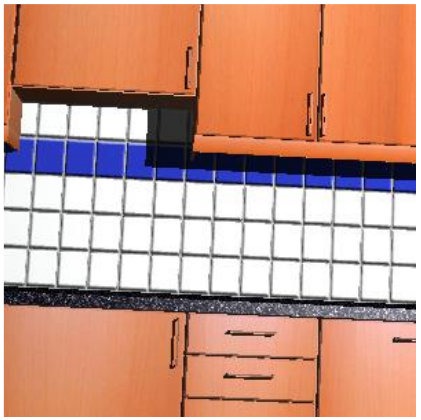
Given a fixed scene complexity,
more samples per pixel **decreases the ray-tracing pixel ratio**,
thereby **improving rendering efficiency**.

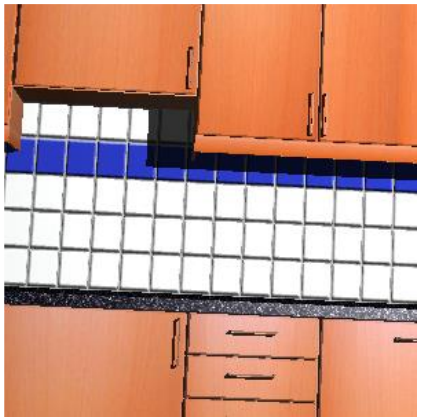
Supersampling is often essential for producing high-quality images.

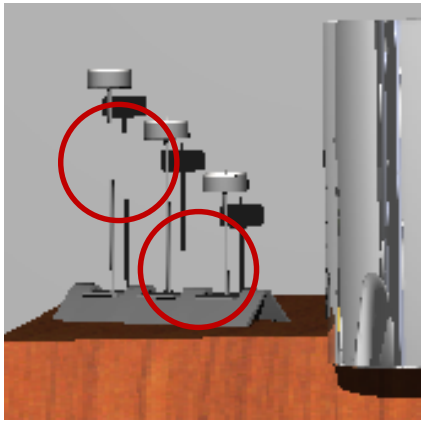
Postcorrection Using Object ID and Shadow Flags

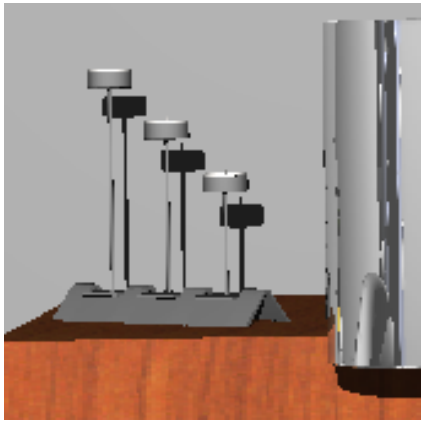


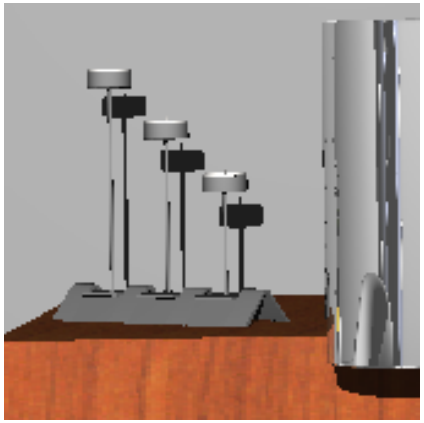








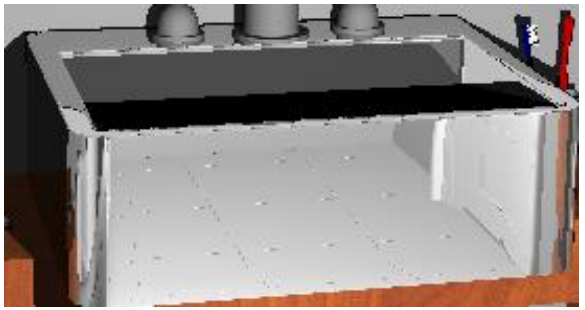




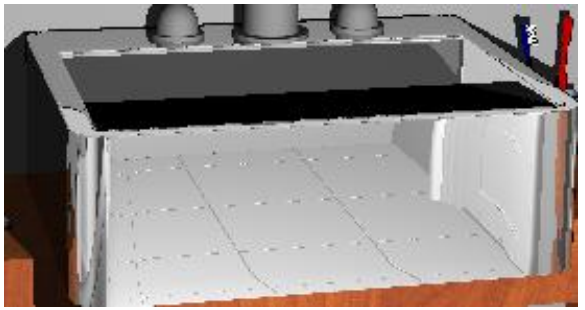
Before correction | After correction | Full ray tracing

- Effectively reconstructed the vanishing parts of **thin objects and the shadow cast by them**.
- Required **7.7%(Ben)** to **17.1%(Conference)** of the entire rendering time.
- Can **often be turned off** for the scene that do not contain very thin objects such as Café, Ben, and Conference.

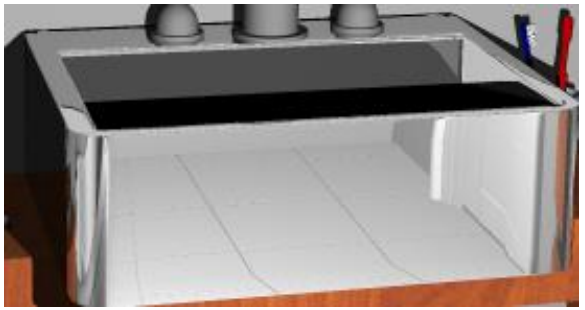
Small Details beyond Sampling Density



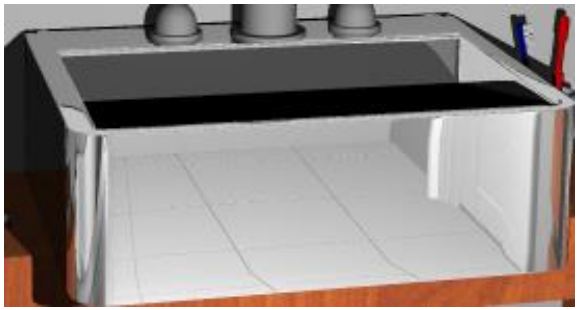
Ours (**1x1 sampling**)



Full ray tracing (**1x1 sampling**)



Ours (**4x4 sampling**)



Full ray tracing (**4x4 sampling**)

- To handle the small details beyond the capability of the applied sampling density, the appropriate solution is supersampling rather than postcorrection.

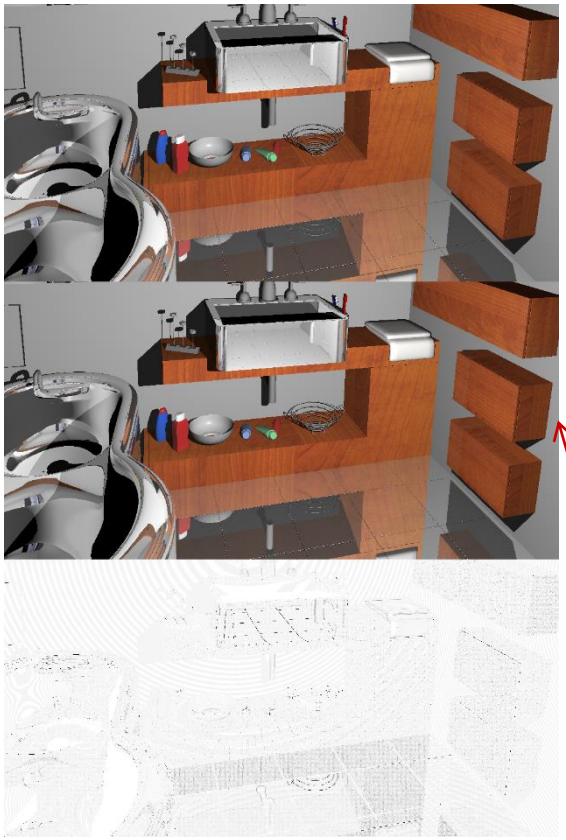
Image Quality (1,024x1,024 Pixels/1x1 Sampling)

CGI 2016

Full ray tracing

Ours

Difference (10x)



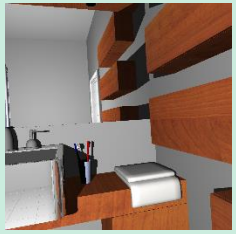
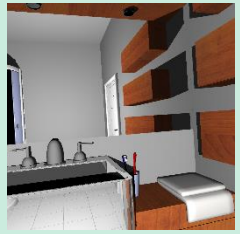
Café (2.04x/43.99dB)

Bathroom (1.59x/40.54dB)

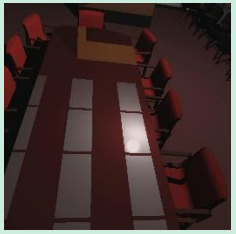
The most obvious **visual errors** usually occur around corners or for highly curved objects, which are **hard to detect using similarity checks**.

Good image quality is maintained despite the reduced numbers of ray shots.

Further Analysis

Reflective pixels	19.9%	41.1%	60.9%	82.0%	99.9%
Full ray tracing (ms)	813.3	870.2	992.8	1,054.4	1,077.3
Ours (ms)	454.0 (1.79x)	451.8 (1.92x)	495.1 (2.00x)	493.5 (2.13x)	437.0 (2.46x)
Camera view					

- Ours is usually **more effective when** more shadow and/or secondary rays are to be ray-traced.
- The rendering cost for handling the increasing reflection rays increases sublinearly.

Ray-traced pixels	25.9%	26.6%	28.0%	29.1%	32.1%
Full ray tracing (ms)	944.6	884.7	866.4	859.8	904.7
Ours (ms)	362.1 (2.60x)	414.4 (2.13x)	460.4 (1.88x)	521.7 (1.64x)	606.1 (1.49x)
Camera view					

- **The ratio of ray-traced pixels** primarily affects the timing performance of our method.

Conclusion

- Presented an **adaptive undersampling** method for efficient **mobile GPU ray tracing**.
  - Attempt to replace expensive ray-tracing operations by much cheaper linear interpolation as much as possible.
  - Selectively controllable in that the quality of respective rendering effects can be adjusted through the corresponding tolerance values.

- Presented a **low-cost postcorrection** method for effectively reducing the aliases due to incomplete undersampling.

- Tailoring our adaptive undersampling algorithm to **best fit the PC GPU** remains as a future research topic.

demo


Scene: Kitchen
Triangle #: 101,015
Resolution: 1,024x1,024
Sampling: 1x1
Proc. : Adreno 420 GPU

demo


Scene: Bathroom
Triangle #: 268,725
Resolution: 1,024x1,024
Sampling: 1x1
Proc. : Adreno 420 GPU

☺ Thank you.

http://grmanet.sogang.ac.kr
kimyu7@sogang.ac.kr