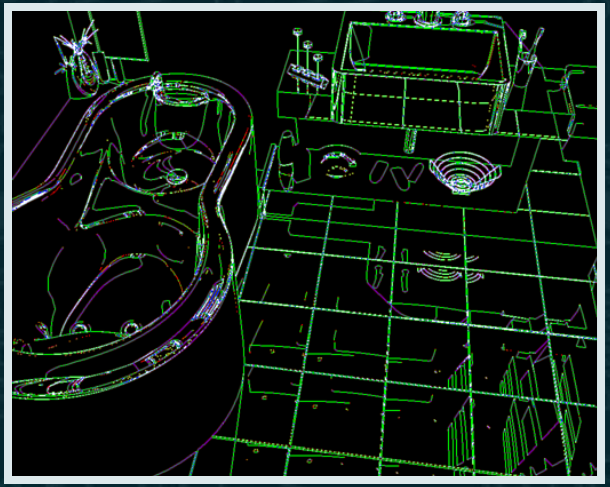


Selective and Adaptive Supersampling for Real-Time Ray Tracer

Bongjun Jin[†], Insung Ihm[†], Byungjoon Chang[†]
Chanmin Park[‡], Wonjong Lee[‡], and Seokyoon Jung[‡]

[†]Department of Computer Science and Engineering, Sogang University, Korea

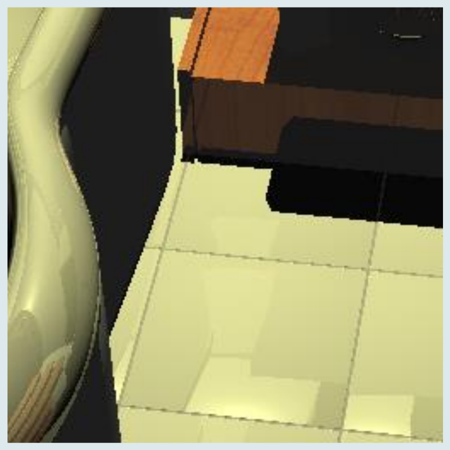
[‡]System Architecture Lab., Samsung Electronics, Korea

Contents

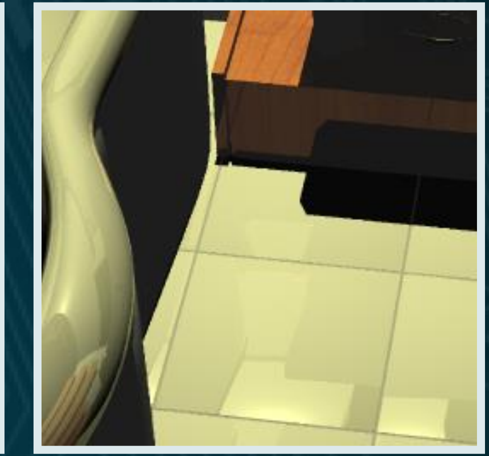
- Backgrounds
- Our contributions
- Previous work
- Basic idea
- Extension of the basic idea
- Experimental results
- Conclusion

Backgrounds

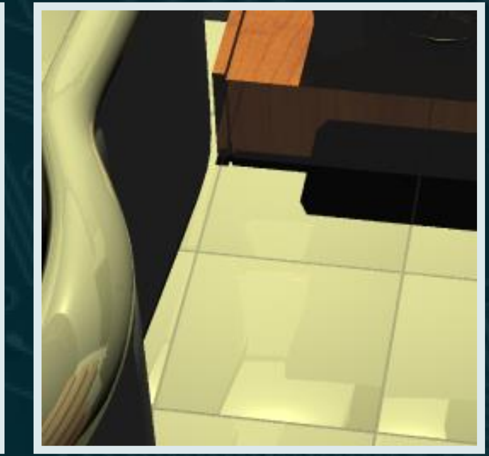
- In general, *high sampling rates such as 9 to 16 samples per pixel* are necessary for producing high quality renderings.
- Such sampling rates are still *too heavy for real-time ray tracing*.
- It is desirable to develop an effective real-time ray tracing technique *that minimizes the total number of processed rays while keeping the image quality*.



One sample per pixel



9 samples per pixel

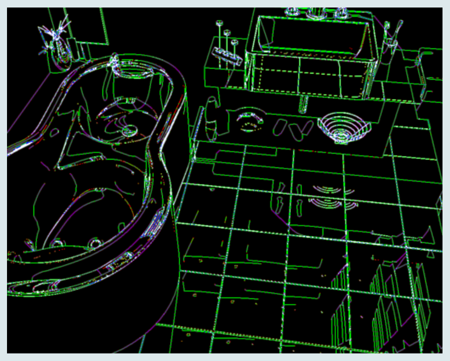


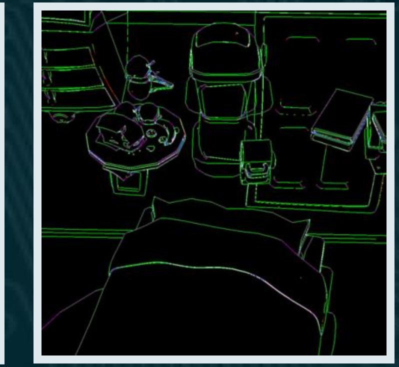
16 samples per pixel

Our Contributions

- We propose *a selective and adaptive supersampling technique for real-time ray tracing* that

  1. was designed to offer high sampling rates as effective as 9 to 16 samples per pixel,
  2. explores both image-space color measures and object-space geometry attributes,
  3. enables users to focus computing effort on selectively chosen rendering features, and
  4. allows an efficient parallel computation on many-core processors.





Previous Work

- CROW, F. 1977. The aliasing problem in computer-generated shaded images. *Communications of the ACM 20, 11, 799–805.*
- BOLIN, M., AND MEYER, G. 1998. A perceptually based adaptive sampling algorithm. In *Proceedings of SIGGRAPH 1998, 299*–309.
- GENETTI, J., GORDON, D., AND WILLIAMS, G. 1998. Adaptive supersampling in object space using pyramidal rays. *Computer Graphics Forum 17, 1, 29–54.*
- HECKBERT, P., AND HANRAHAN, P. 1984. Beam tracing polygonal objects. In *Proceedings of SIGGRAPH 1984, 119–127.*
- LONGHURST, P., DEBATTISTA, K., GILLIBRAND, R., AND CHALMERS, A. 2005. Analytic antialiasing for selective high fidelity rendering. In *Proceedings of SIBGRAPI 2005, 359–366.*
- MITCHELL, D. 1987. Generating antialiased images at low sampling densities. In *Proceedings of SIGGRAPH 1987, 65–72.*
- OHTA, M., AND MAEKAWA, M. 1990. Ray-bound tracing for perfect and efficient anti-aliasing. *The Visual Computer 6, 3*,125–133.
- THOMAS, D., NETRAVALI, A., AND FOX, D. 1989. Antialiased ray tracing with covers. *Computer Graphics Forum 8, 4, 325*–336.
- WHITTED, T. 1980. An improved illumination model for shaded display. *Communications of the ACM 23, 6, 343–349.*

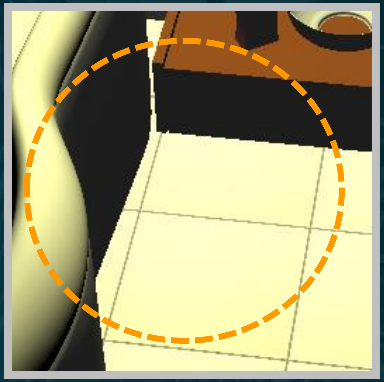
- While effective, they, in their current forms, are not best suited for effective implementation on the computing architecture of today’s many-core processors such as GPU.

Basic Idea (Primary Ray Sampling Case Only)

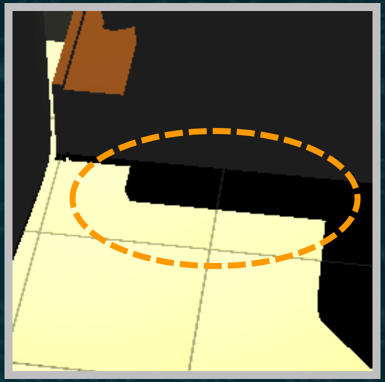
Observation

- In ray tracing, annoying alias artifacts often occur *due to insufficient sampling of various rendering features*.

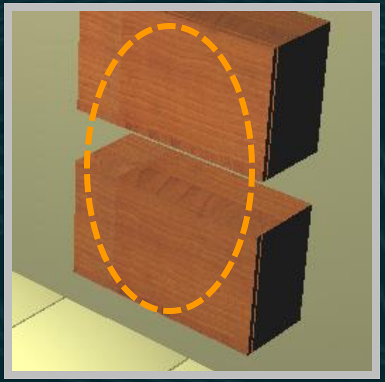
Scene



Primary ray sampling

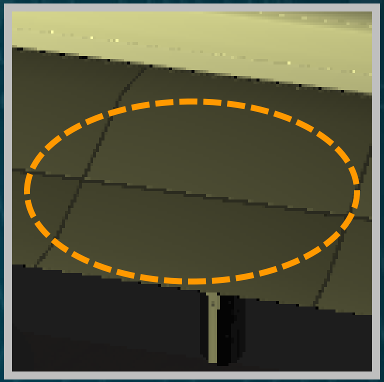


Shadow ray sampling

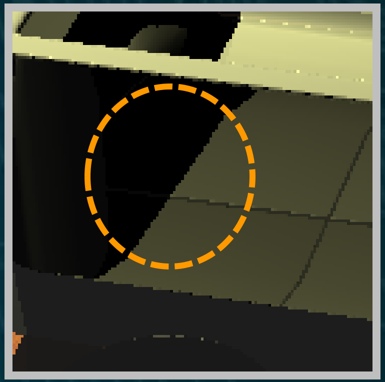


Texture sampling

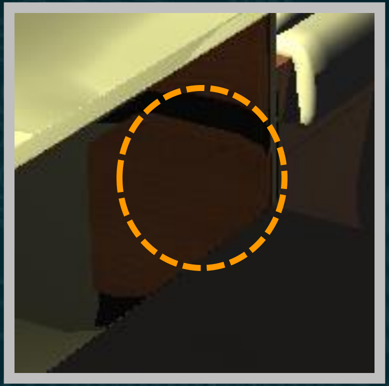
Reflection ray sampling



Reflection ray sampling(object)



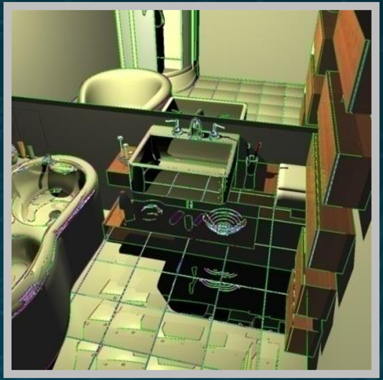
Reflection ray sampling(shadow)



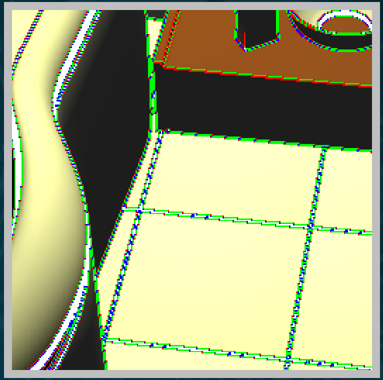
Reflection ray sampling(texture)

Our Attempt

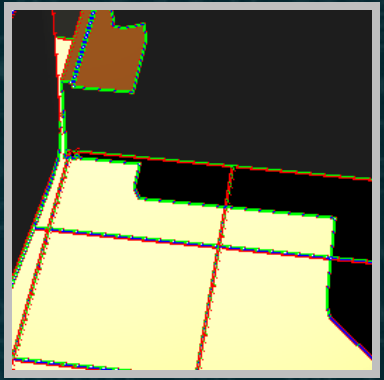
- Detect such *problematic pixel regions* on the fly, and shoot more sampling rays to them *selectively and adaptively*.



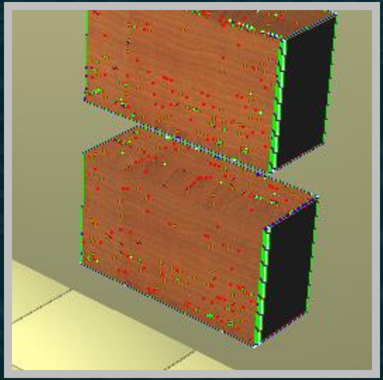
Scene



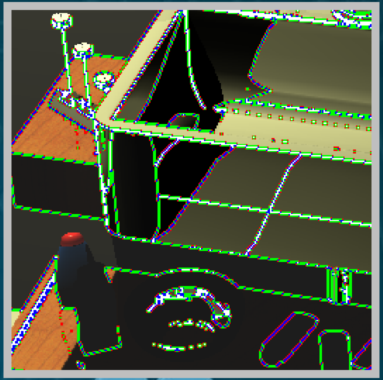
Primary ray sampling



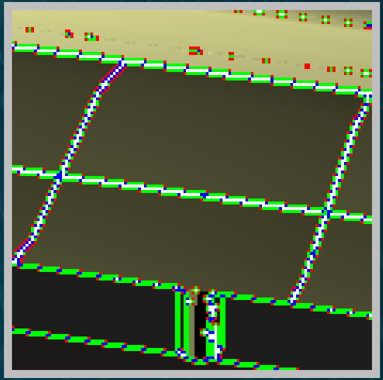
Shadow ray sampling



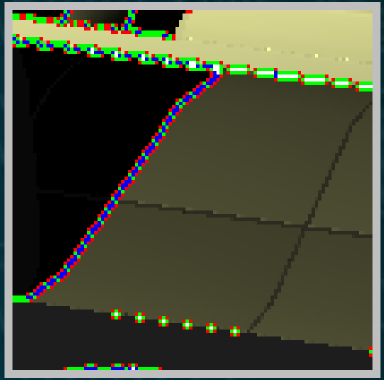
Texture sampling



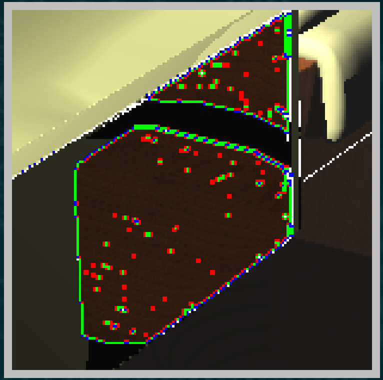
Reflection ray sampling



Reflection ray sampling(object)



Reflection ray sampling(shadow)



Reflection ray sampling(texture)

Design Goal (Pr. Ray Sampling Case Only)

- Distribute extra sample rays to the pixel region if
  - an object silhouette curve crosses it (*object-space measure*), or
  - a color disparity with adjacent pixels is found (*image-space measure*).

- Design *a selective sampling mechanism* that allocates more of the very limited computing time to possibly more troublesome rendering features.

- Achieve high sampling rates that are *as effective as 9 to 16 samples per pixel*.

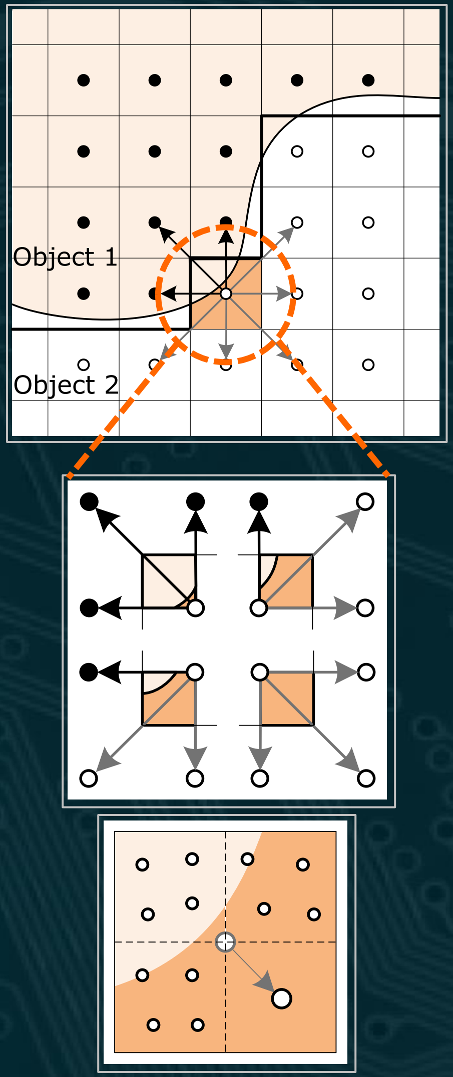
- Design a simple sampling algorithm that is *well suited to highly parallel, multithreaded, many-core GPUs*.
  - In particular, *minimize data-dependent, unpredictable control flows*.

Our Solution

- Get the two pixel attributes at pixel centers by tracing one ray per pixel.
  - *Geometry attribute:* object id. number (ObjectID)
  - *Color reference:* shaded color

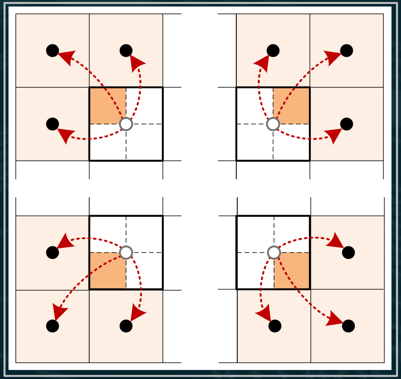
- Subdivide each image pixel into four subpixels, and independently perform *two simple tests* against each subpixel *to see if its region needs extra sampling.*

- Shoot *four extra rays per problematic subpixel,* and blend the results into the pixel color.

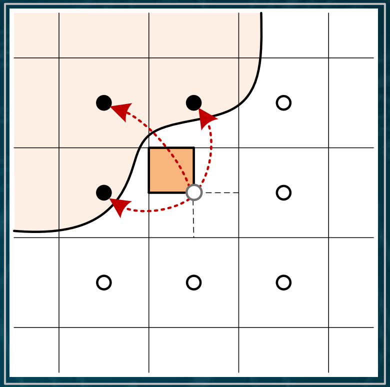


Test I: *Geometry Attribute Comparison*

- For each subpixel, compare its ObjectID respectively with those of three adjacent pixels.

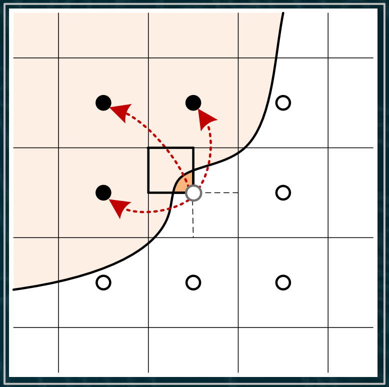


- When there is at least one disparity
  - Two extreme cases



Do not need extra samples!

Do need extra samples!



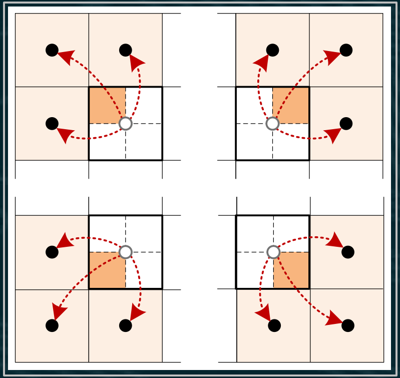
An optimistic case

A pessimistic case

  - We always assume *the pessimistic case* to achieve a simple control flow, *taking four extra samples*.

Test 2: *Color Reference Comparison*

- Threshold τ for color reference comparison
  - $\tau \in [0, 1]$ is set properly as a result of the geometry attribute comparison. (will be explained shortly)



- Contrast-based disparity test [Mitchell 1987]
  - For the four color references, compute the contrast values.

$$I_\lambda = \frac{I_\lambda^{max} - I_\lambda^{min}}{I_\lambda^{max} + I_\lambda^{min}}, \quad \lambda = R, G, B$$

  - Use the following threshold vector for a disparity check.

$$(\tau \cdot 1.36, \tau \cdot 1.02, \tau \cdot 2.04), \quad \tau \in [0, 1]$$

✓ When τ is 0.3, the vector roughly becomes Mitchell's default (0.4, 0.3, 0.6).

Three-Stage Supersampling Algorithm

[Stage 1] Per-pixel attribute presampling

- *Build three 2D arrays by presampling image pixels at their centers by performing ray tracing.*
  - *Shaded color image*
    - □ Stores ray-traced, shaded colors.
    - □ *Functions as a color buffer* to which extra subsampled colors are accumulated.
  - *Geometry attribute map*
    - □ Stores the IDs of objects that were hit by the primary rays.
  - *Color reference map*
    - □ Stores pixel values that are referred to when a test for color disparity is performed.
    - □ Our current implementation uses the shaded colors for color reference.

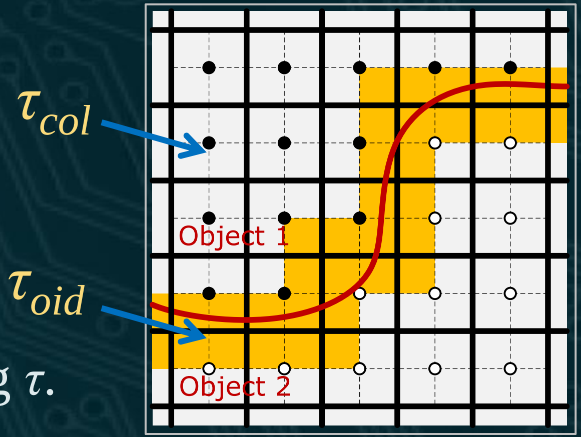
[Stage 2] Two-step subpixel test

- *Determine whether and where to take extra sample rays based on the pixel attributes in the two maps.*

① Choose a threshold τ through the geometry attribute comparison.

□ If there is at least one mismatch, assign a user-defined threshold τ_{oid} to τ.

□ Otherwise, assign another user-controlled threshold τ_{col} to τ.



② Perform the color reference comparison using τ.

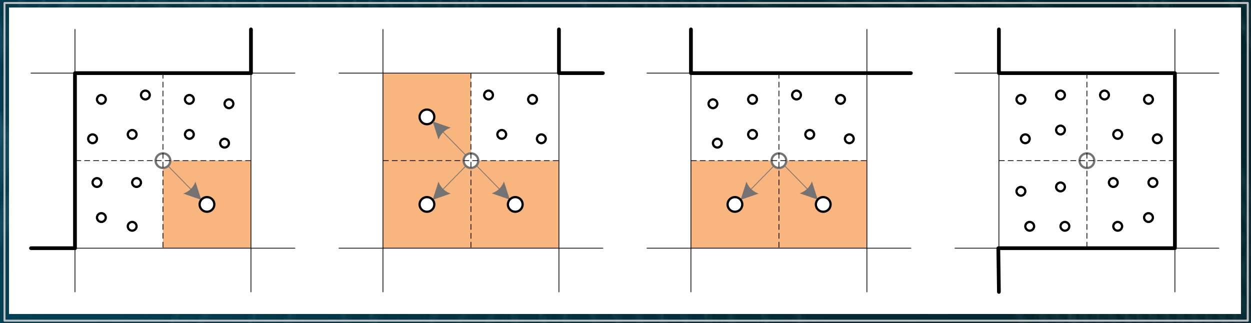
□ If a color disparity is found, the subpixel is considered as *problematic* and marked as *active*.

✓ *By using a stricter value of τ_{oid}, we can selectively focus computing resources more on reducing jagged edges!*

[Stage 3] Subpixel sampling and color summing

- *Take four extra samples for each active subpixel through ray tracing, and sum the shaded colors with those of the inactive subpixels.*

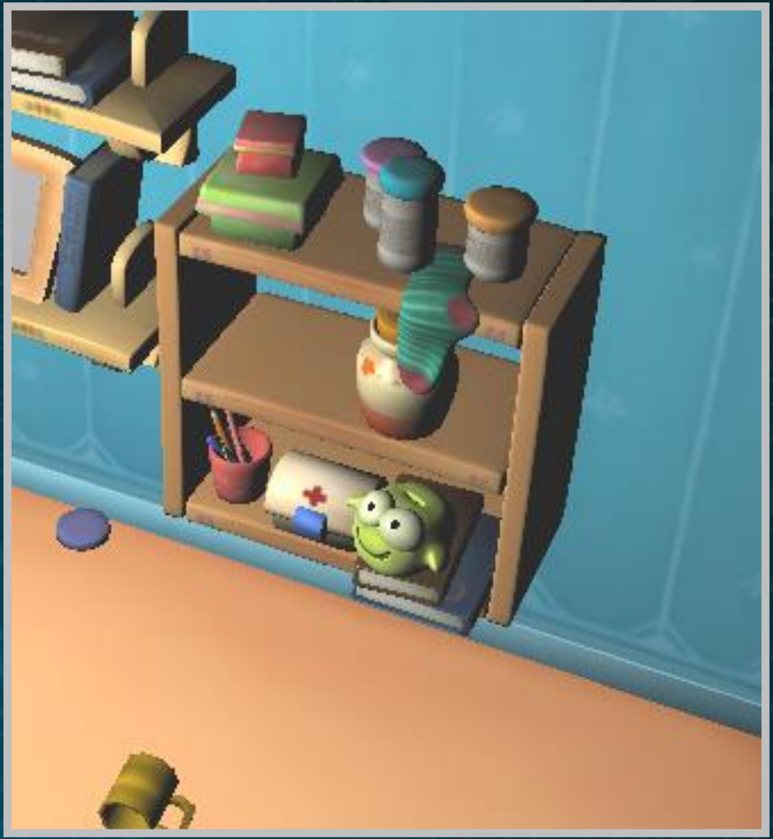
  - *For color summing*
    - □ Multiply a weight (# of inactive subpixels)/4 to each pixel in the shaded color image.
    - □ Simply accumulate the subsampled extra colors to the shaded color image, multiplied with a weight of 1/16.



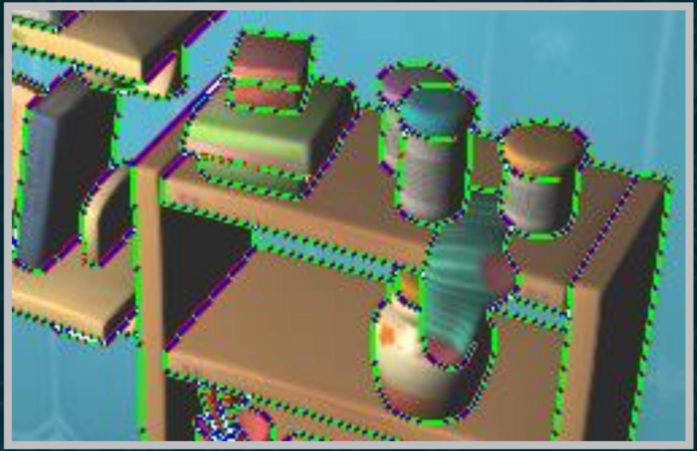
Four examples of adaptive sampling

Selective Supersampling Example

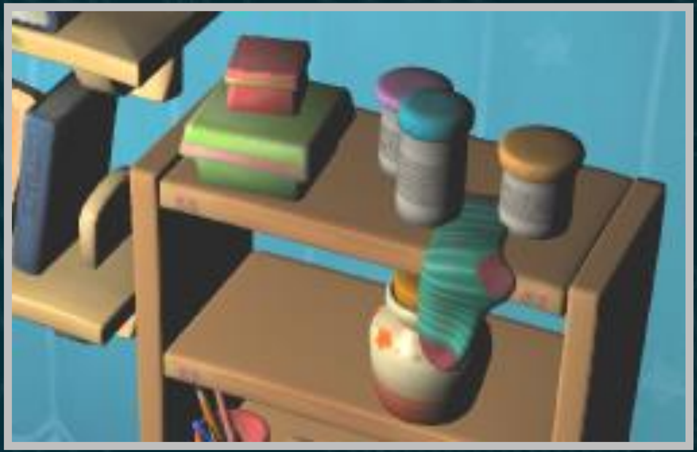
# of problematic subpixels
Red: 1, Green: 2, Blue: 3, White: 4



One sample per pixel



Problematic subpixels



After selective supersampling

Extension of the Basic Idea

Considering More Geometry Attributes

- Three classes of pixel attributes

User-controlled selective supersampling parameters

Collection point	Attribute	Threshold
At pixel center	Color Reference	τ_{col}
At primary ray hit point	Object ID	τ_{poid}
	Surface Normal	τ_{psn}
	Shadow Count	τ_{psc}
	Texture Existence	τ_{pte}
At secondary ray hit point	Object ID	τ_{soid}
	Surface Normal	τ_{ssn}
	Shadow Count	τ_{ssc}
	Texture Existence	τ_{ste}

For reducing artifacts in the secondary rendering effects (reflection/refraction)

For detecting such an edge formed by polygons that meet at an acute angle

For shadow antialiasing (records the no. of light sources invisible from the intersection point)

For texture antialiasing (TRUE if and only if a texture is applied to the intersection point)

Item-to-Item Disparity Test

- Object ID
  - YES iff the IDs are different.
- Surface Normal
  - YES iff the cross product is less than a preset value.

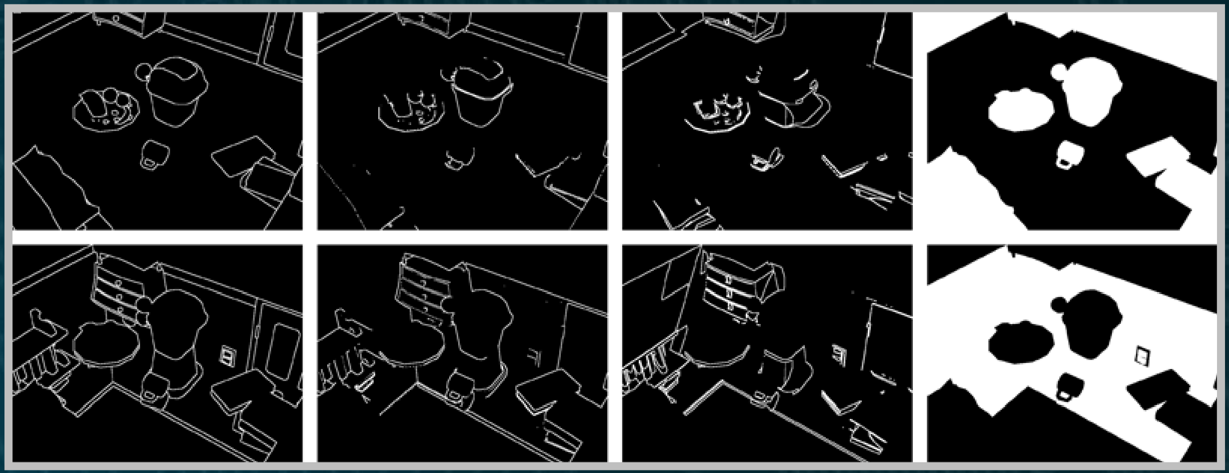
17 to 24%

- Shadow Count
  - YES iff the counts are different.
- Texture Existence
  - YES iff the current subpixel's attribute is TRUE.

Extended Geometry Attribute Comparison

- A subpixel is called possibly problematic if discordance is found for at least one geometry attribute.
  - The smallest value of the thresholds set to the disagreeing attributes is used in the color reference comparison.
  - When no discordance is found, τcol is used as default.

Test Scene

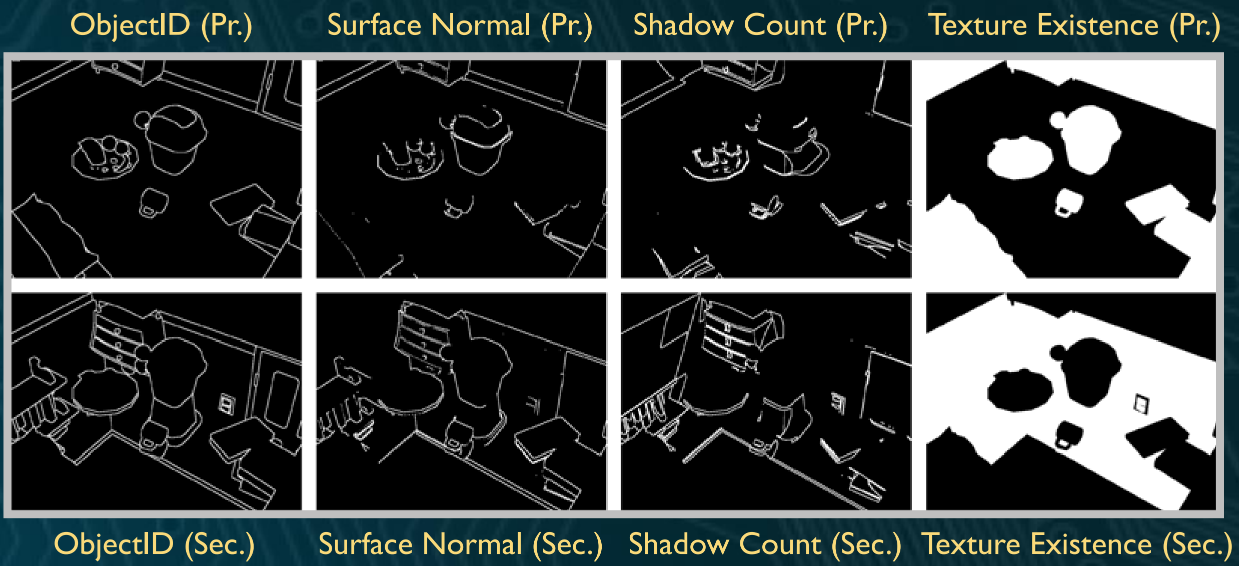


Textures are applied to all, diffusive objects except the only specular, floor object.

Extended Geometry Attribute Comparison

- A subpixel is called *possibly problematic* if discordance is found for *at least one* geometry attribute.
  - The *smallest value* of the thresholds set to the disagreeing attributes is used in the color reference comparison.
  - When no discordance is found, τ_{col} is used as default.

Test Scene

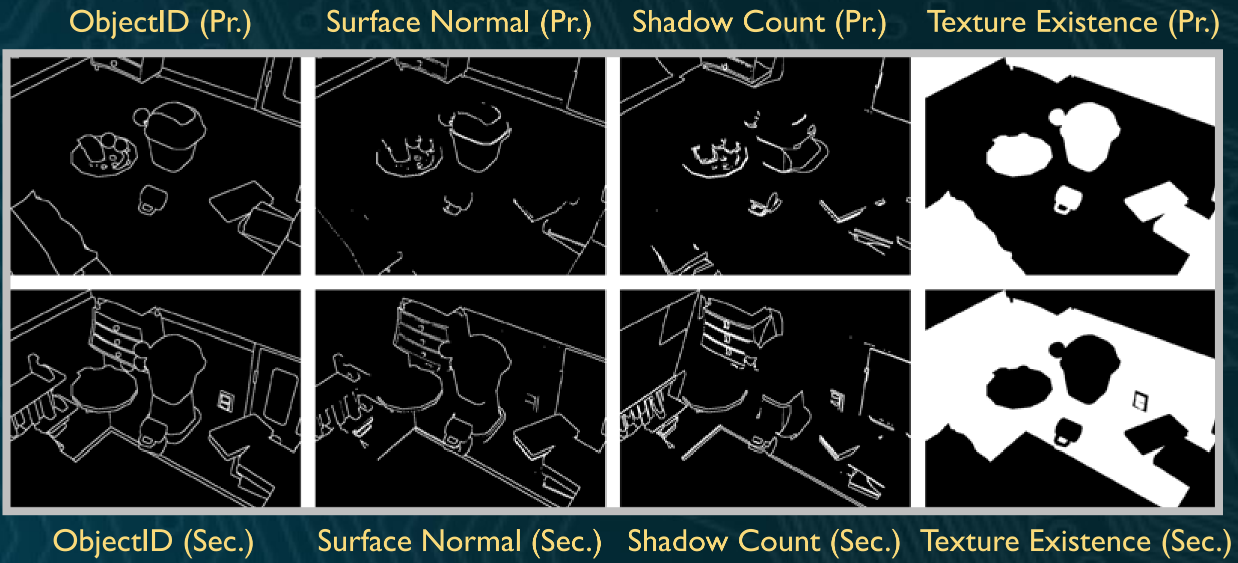


Textures are applied to all, diffusive objects except the only specular, floor object.

Extended Geometry Attribute Comparison

- A subpixel is called *possibly problematic* if discordance is found for *at least one* geometry attribute.
  - The *smallest value* of the thresholds set to the disagreeing attributes is used in the color reference comparison.
  - When no discordance is found, τ_{col} is used as default.

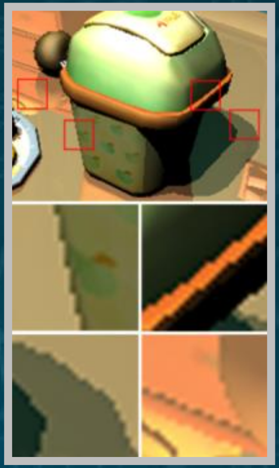
Test Scene



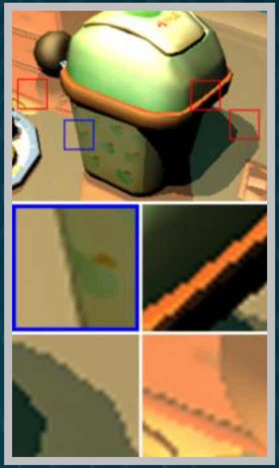
Textures are applied to all, diffusive objects except the only specular, floor object.

Selective Supersampling Examples

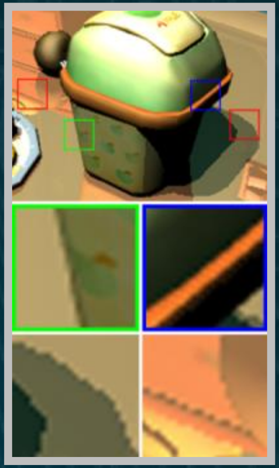
- A user can distribute ray samples to rendering elements according to priorities *by selectively setting the nine thresholds*.
  - If shadows are important, use rigorous, i.e., small τ_{psc} (and τ_{ssc}).
  - If they are really important, set them to zero.



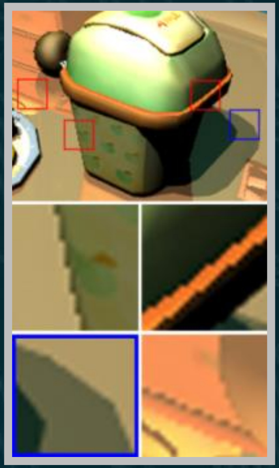
One sample per pixel



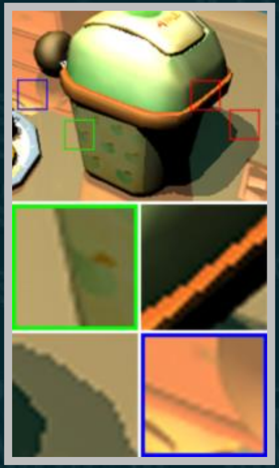
Object ID (Pr.)
$\tau_{poid} = 0.0$



Surface Normal (Pr.)
$\tau_{psn} = 0.05$



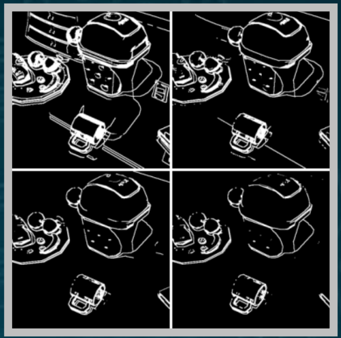
Shadow Count (Pr.)
$\tau_{psc} = 0.05$



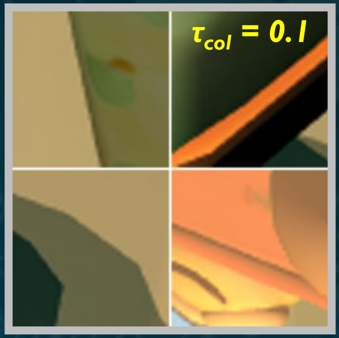
All except texture existence (Sec.)
$\tau_{soid} = \tau_{ssn} = \tau_{ssc} = 0.05$

Single- vs. Multiple-Valued Thresholds

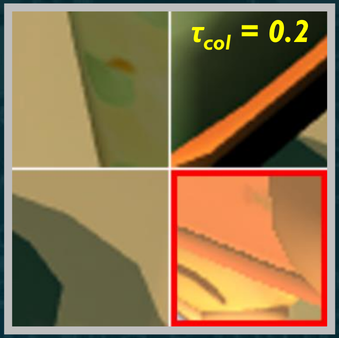
- Color measure only



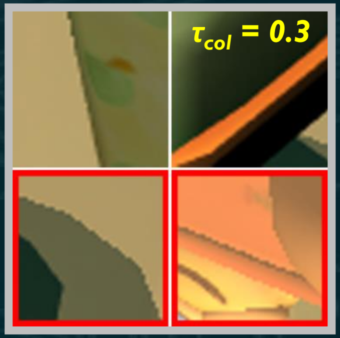
Problematic subpixels



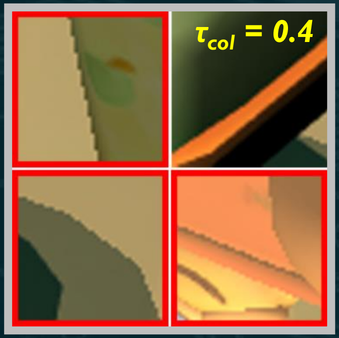
Sample Ratio = 22.62%
PSNR = 46.20



Sample Ratio = 16.09%
PSNR = 44.17

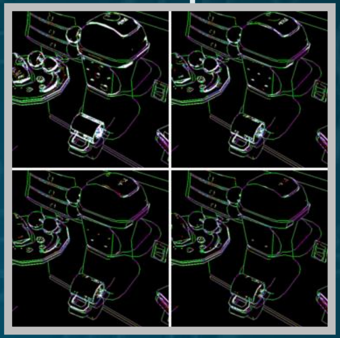


Sample Ratio = 13.76%
PSNR = 42.96

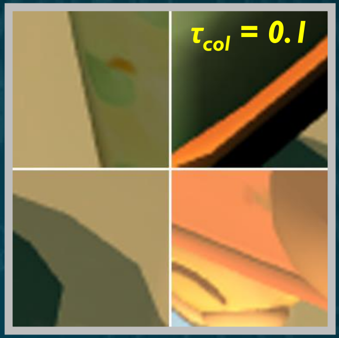


Sample Ratio = 11.86%
PSNR = 41.21

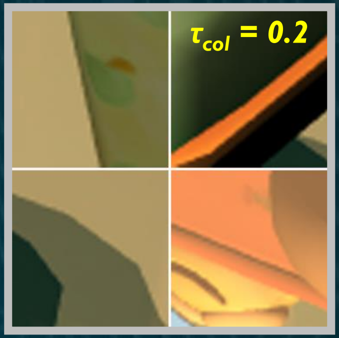
- Multiple measures



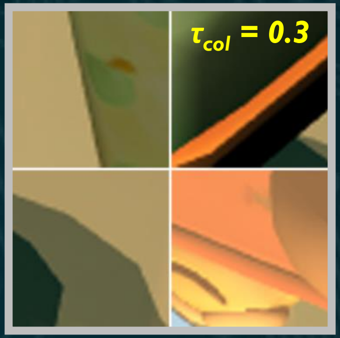
Problematic subpixels



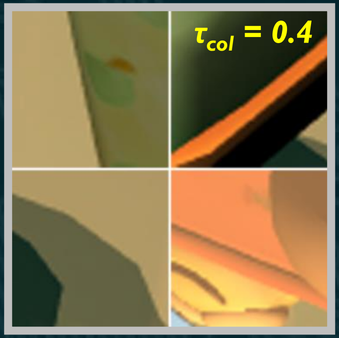
Sample Ratio = 15.75%
PSNR = 46.17



Sample Ratio = 13.06%
PSNR = 45.75



Sample Ratio = 12.20%
PSNR = 45.46



Sample Ratio = 11.83%
PSNR = 45.20

Implementation on the GPU Architecture

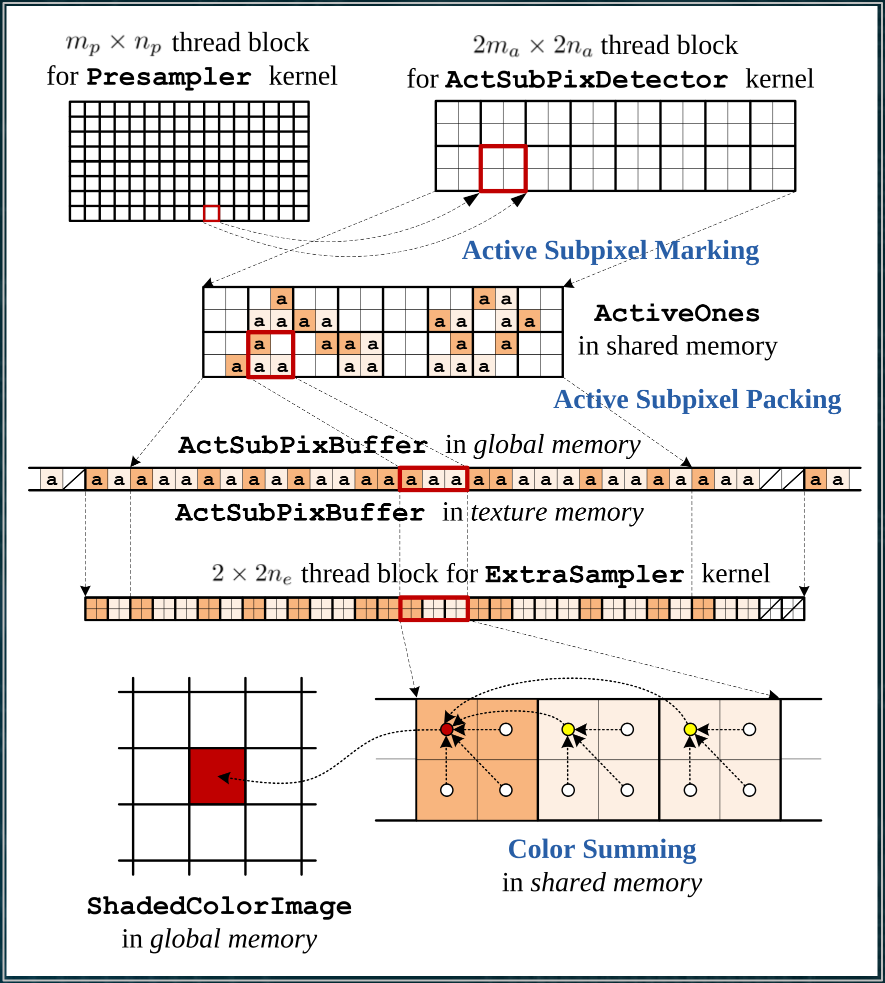
Efforts towards Efficiency

- *Launch a kernel with an enough number of concurrent threads*.
  - Pixels in the first stage, and subpixels in the next two stages becomes threads.
- *Exploit the fast registers and shared memory, and minimize use of the expensive off-chip memories*.
  - Store the on-the-fly active subpixel information in shared memory.
  - Perform the color sum operation in shared memory.
- *Exploit spatial and temporal locality in data access*.
  - Store the color reference map, geometry attribute map, and active subpixel buffer in texture memory.
- *Use a simple control structure that minimizes unpredictable, data-dependent branches*.
  - Subdivide each image pixel into 4 subpixels to which computations are performed independently.

Implementation on the CUDA Architecture

- Three computation kernels
  - `Presampler` kernel
  - `ActSubPixDetector` kernel
  - `ExtraSampler` kernel

✓ Implementation details are given in the paper.


$m_p \times n_p$ thread block for **Presampler** kernel
$2m_a \times 2n_a$ thread block for **ActSubPixDetector** kernel
Active Subpixel Marking
ActiveOnes in shared memory
Active Subpixel Packing
ActSubPixBuffer in *global memory*
ActSubPixBuffer in *texture memory*
$2 \times 2n_e$ thread block for **ExtraSampler** kernel
Color Summing in *shared memory*
ShadedColorImage in *global memory*

Experimental Results

Implementations

- Tested on an NVIDIA GeForce GTX 280 GPU
  - 16,384 registers and 16 Kbytes of shared memory per multiprocessor

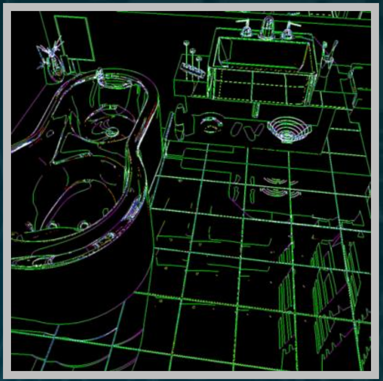
- Our ray tracer *with selective and adaptive supersampling*
  - The *short stack* method [Horn et al. 2007] was used for kd-tree traversal.
  - The kernels consumed up to 58 registers, limiting the total # of possible threads in a block to 282.
  - 7 stack elements per thread (8 bytes each) were allocated in shared memory.

- A test ray tracer *with fixed-density supersampling*
  - Built by slightly modifying our ray tracer.
  - A similar shared memory technique was applied for an efficient GPU implementation.
    - The overheads were measured in the range of *8.68 to 10.12* for the case of *16 fixed samples per pixel*.

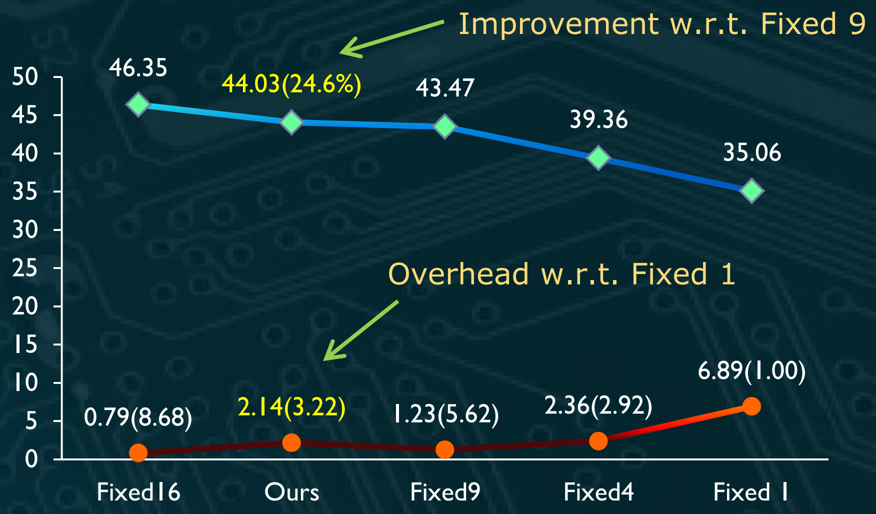
Experimental Results

Resolution: 1024x1024
Fixed n: n fixed samples per pixel
PSNR: compared with 256 fixed sampling

Bathroom(268K)



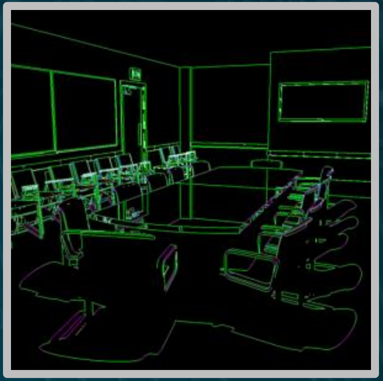
Problematic subpixels



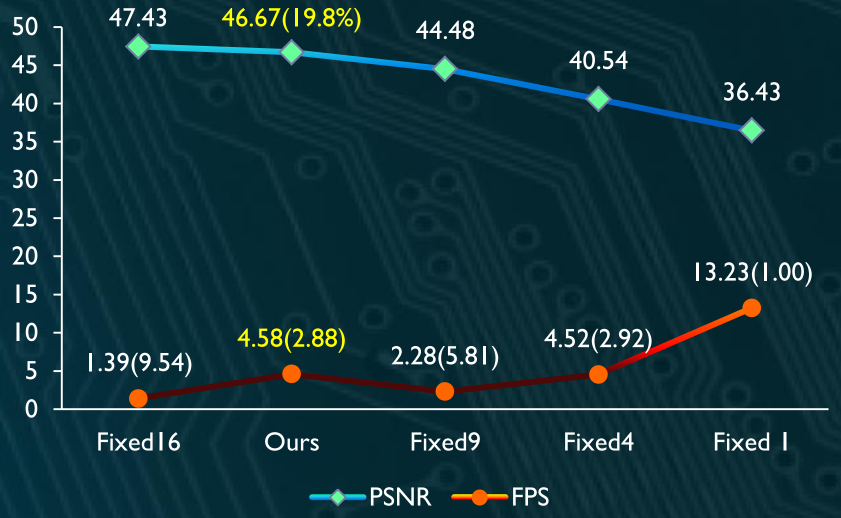
Improvement w.r.t. Fixed 9

Overhead w.r.t. Fixed 1

Conference(190K)

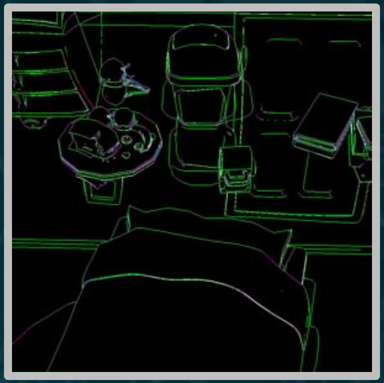


Problematic subpixels

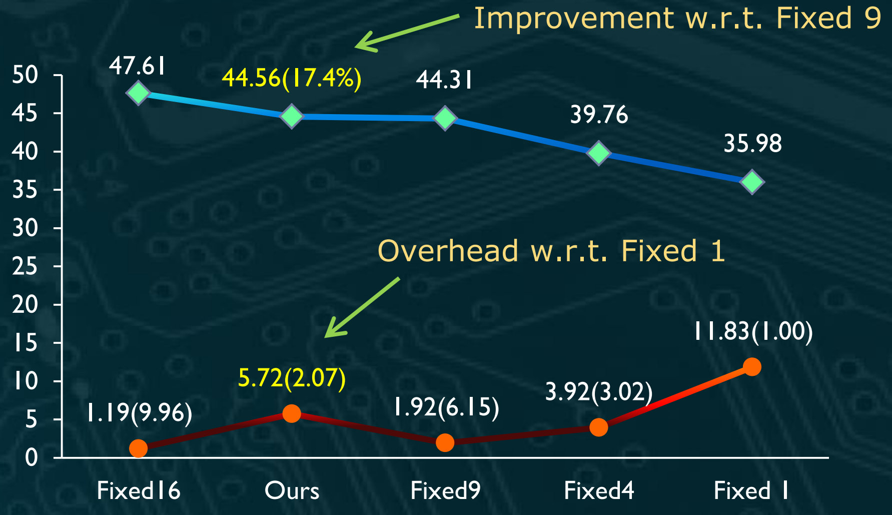


Resolution: 1024x1024
Fixed n: n fixed samples per pixel
PSNR: compared with 256 fixed sampling

Room(117K)



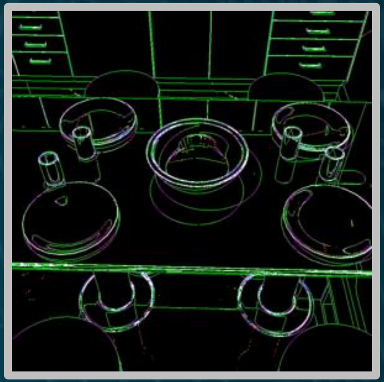
Problematic subpixels



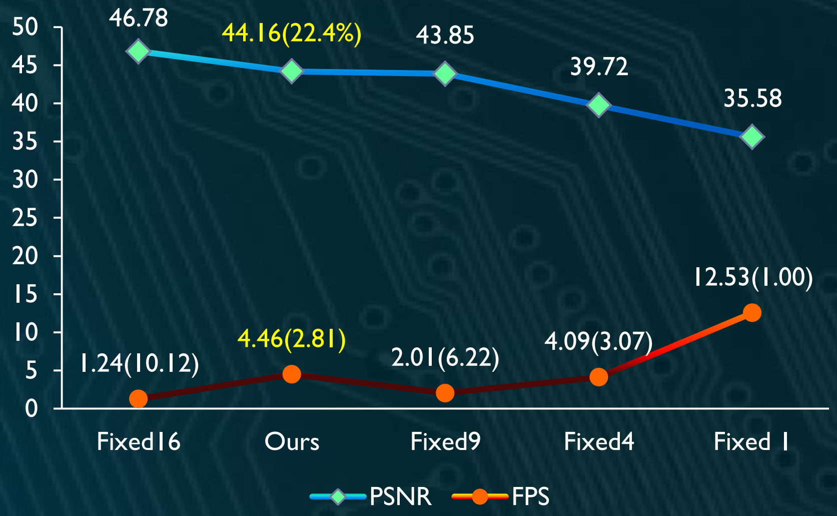
Improvement w.r.t. Fixed 9

Overhead w.r.t. Fixed 1

Kitchen(101K)

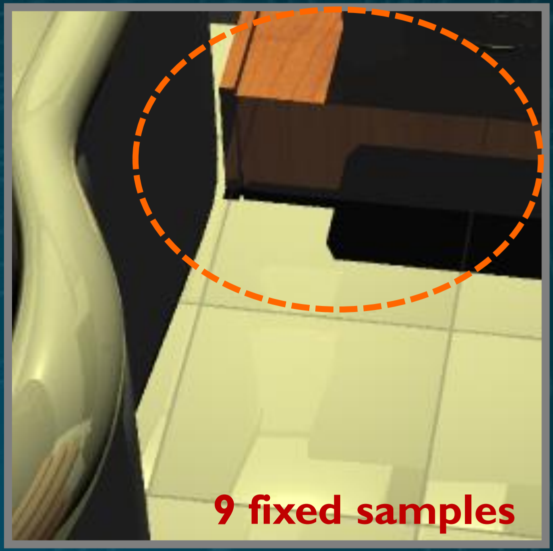


Problematic subpixels

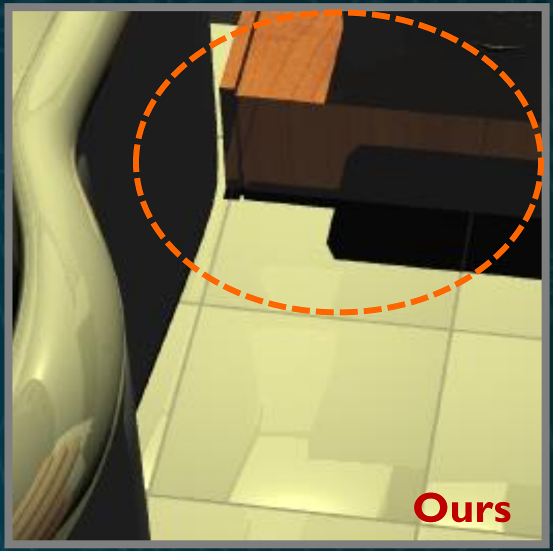


For the Four Test Scenes,

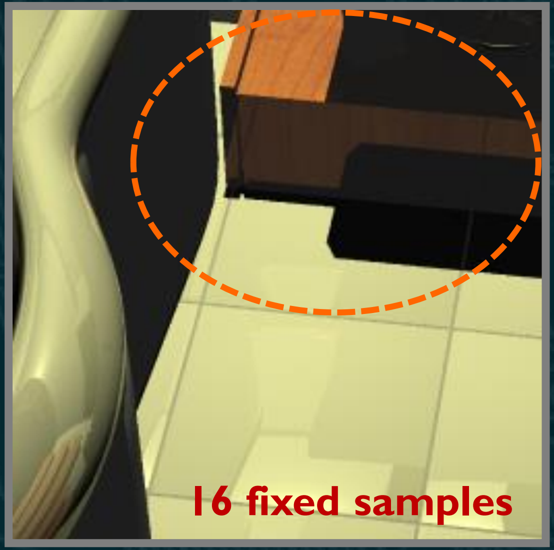
- Compared to the 9 fixed density sampling
  - 2 to 3 times faster (taking only around 17 to 24% of samples).
  - PSNR values falling between those of the 9 and 16 fixed samplings.



9 fixed samples

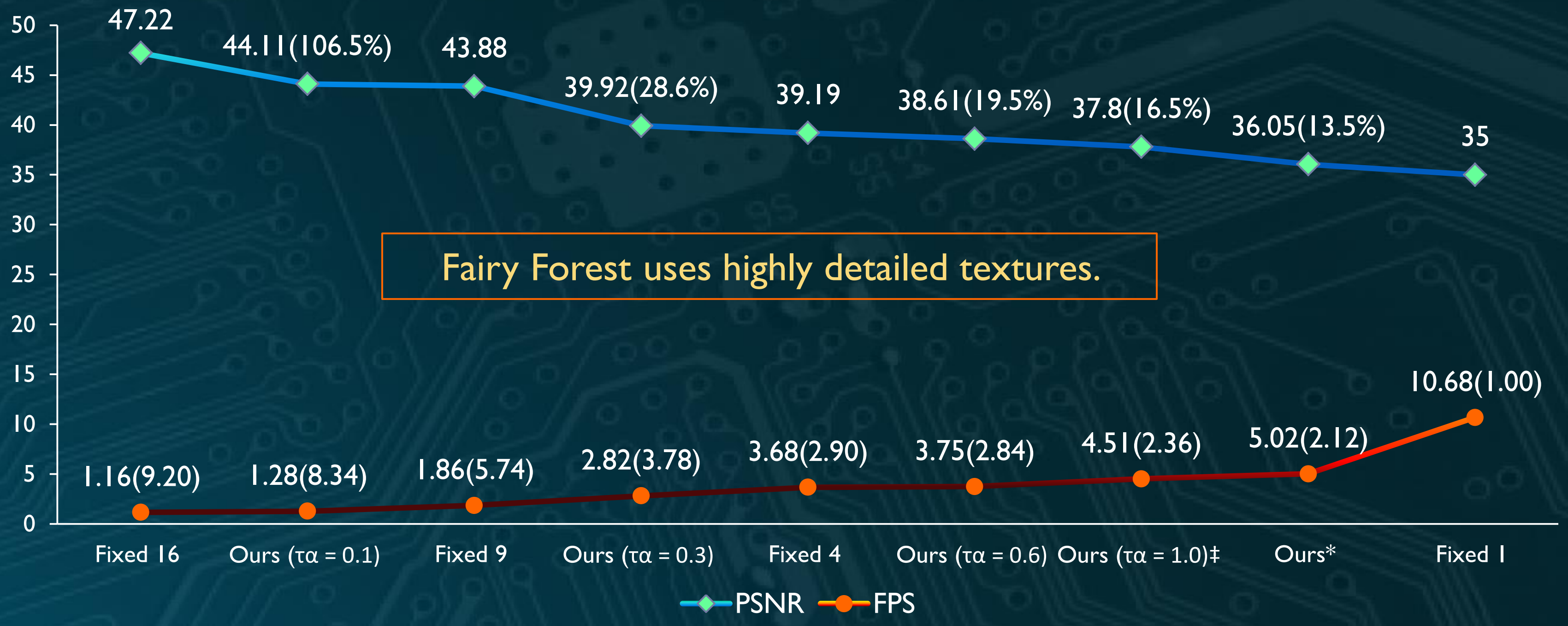


Ours



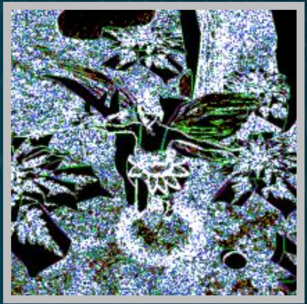
16 fixed samples

Problem with Highly Detailed Textures

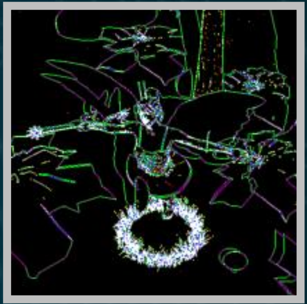


Fairy Forest uses highly detailed textures.

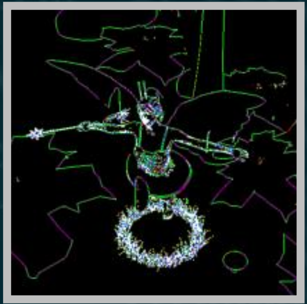
Fairy Forest(174K)



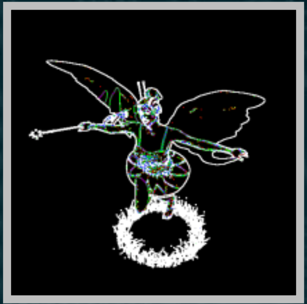
$\tau_\alpha = 0.1$



$\tau_\alpha = 0.3$



$\tau_\alpha = 0.6$

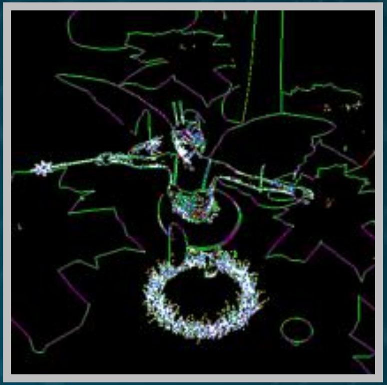


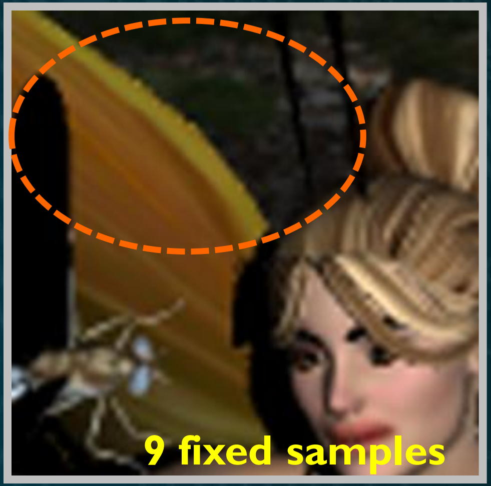
Fairy focused

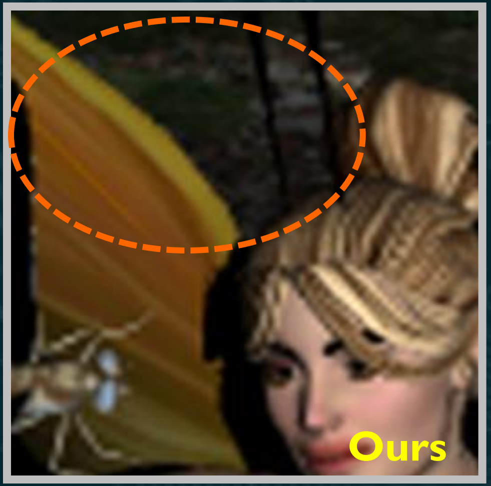
$\tau_\alpha = \tau_{col}, \tau_{pte}, \tau_{ste}$

- Stricter thresholds had to be used to achieve a high PSNR values.
  - **Ours (τ_α = 0.1)**: 44.11 (1.28 fps)
  - **Fixed 9**: 43.88 (1.86 fps)

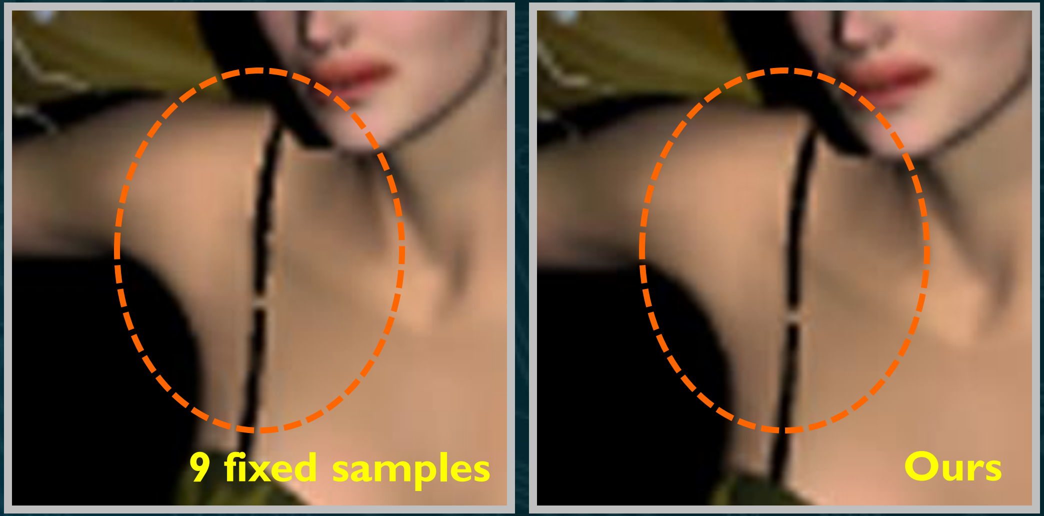
- Directing more ray samples to *perceptually more visible artifacts*
  - **Ours (τ_α = 0.6)**: 38.61 (3.75 fps)







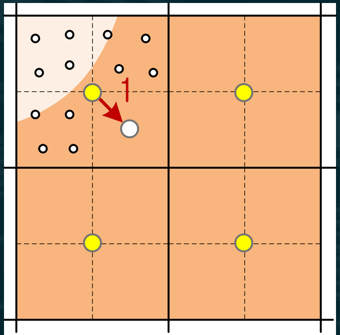
- When the viewer’s attention is focused on the fairy only
  - Consider only the fairy, grass, and dragonfly objects.
  - **Ours**: 5.02 fps
  - **Fixed 9**: 1.86 fps

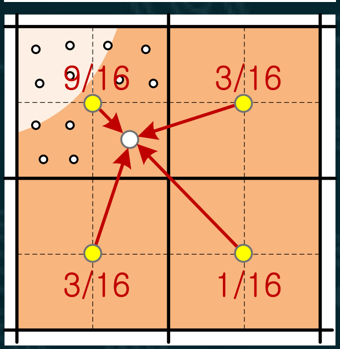


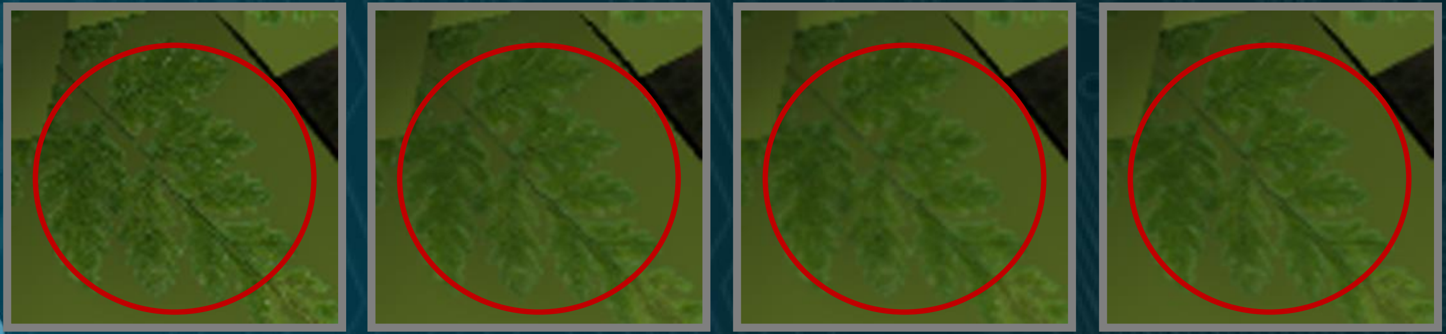
- Solution to the texture aliasing problem
  - Use a low-pass filtered textures. :-)
  - Use a known texture filtering method like mip-mapping.
  - *Develop a filtering scheme for our supersampling technique*.

Bilinear Filtering for Texture Antialiasing

- Idea
  - In the original implementation, the *nearest-neighbor filter* was applied to get the colors of inactive subpixels.
  - A *bilinear filtering* scheme could reduce texture aliases when highly detailed textures are applied.
- Our preliminary test shows
  - only a little cost increase (almost the same fps), and
  - a slight noise reduction (PSNS 38.61 → 41.09 for Fairy Forest).







Ours (nearest)

Fixed (4 rays/pixel)

Ours (bilinear)

Fixed (9 rays/pixel)

Conclusion

To Wrap Up

- For efficient ray tracing, it is important to *minimize the total number of processed rays while maintaining the image quality*.

- We have presented *a selective and adaptive supersampling technique* suitable for *real-time ray tracing on many-core processors*.

- The presented technique will also be easily mapped on the upcoming many-core processors, such as the Intel Larrabee processor.

Thank you!