Supplementary Material: Importance-Based Ray Strategies for Dynamic Diffuse Global Illumination

ZIHAO LIU, Huawei Technologies Canada Co., Ltd., Canada
JING HUANG, Huawei Technologies Canada Co., Ltd., Canada
ALLAN ROCHA, Huawei Technologies Canada Co., Ltd., Canada
JIM MALMROS, Huawei Technologies Canada Co., Ltd., Canada
JERRY ZHANG, Huawei Technologies Canada Co., Ltd., Canada

In this document, we provide algorithms, implementation details, cost estimation, and proofs for other aspects of our IS-DDGI approach not covered in the main paper.

1 Algorithms

1.1 Random Ray Orientation

The Algorithm 1 presents how we generate ray samples per ds_{ij} of a probe. $noiseFunction$ samples a 2D random value with techniques in [1, 3–5].

Algorithm 1 Random Ray Orientation

Input
- t — the frame number
- ds_{ij} — the jth direction set of the ith probe
- $noise_{tex}$ — a noise texture with values in $[0, 1]^2$.

Output
- A 3 dimensional vector in the solid angle of ds_{ij}.

$offset \leftarrow noiseFunction(i, j, t, noise_{tex}).xy$ ▷ generate a random value $\in [0, 1]^2$ for ds_{ij}.
$ONV \leftarrow Encode(offset, ds_{ij})$ ▷ generate octahedral normal vector $\in [-1, 1]^2$ for a probe.
return $OctDecode(ONV)$ ▷ convert to a 3D vector with octahedral mapping [2].

1.2 Probe State Updates

Algorithm 2 describes the method to compute probe states by iterating through all valid ray samples on a probe $r \in \Omega_i$. Note that because each ds_{ij} contains a variable number of valid ray samples, we read from $rayHitInfo_{tex}$ to get the actual number of valid samples to compute probe states.

1.3 Irradiance and Visibility Integration

Algorithm 3 describes how IS-DDGI utilizes $rayHitInfo_{tex}$ to iterate through all valid ray samples $r \in \Omega_i$ to integrate irradiance and visibility values.

2 Implementation

2.1 Uniform Rays

Uniform Ray Allocation Our ray allocation shader computes ray allocation information per ds_{ij} in the texture. Each probe is assigned a 16 pixels by 16 pixels tile, which is mapped to probe's sphere with octahedral mapping [2]. The coordinate of ds_{ij} in this tile is $(j \mod 16, j/16)$.

The predefined uniform ray allocation pattern (Figure 1, texels marked 0) is implemented by creating consecutive ray allocation cells within each row of the ray allocation tile. The number of

Authors' addresses: Zihao Liu, zihaol1@alumni.cmu.edu, Huawei Technologies Canada Co., Ltd., Canada; Jing Huang, jing@cs.toronto.edu, Huawei Technologies Canada Co., Ltd., Canada; Allan Rocha, rocha.allanc@gmail.com, Huawei Technologies Canada Co., Ltd., Canada; Jim Malmros, jim.robert.malmros@huawei.com, Huawei Technologies Canada Co., Ltd., Canada; Jerry Zhang, jerryzhang0101@gmail.com, Huawei Technologies Canada Co., Ltd., Canada.

Algorithm 2 Update Probe State

Input
- i the ith probe.
- $rayHitInfo_{tex}$ ray-hit information computed with equation 14, 16, and 17.
- $HitSample_{tex}$ ray-hit samples with dimension $(|\Omega_i^{ds}| * RAYSLIMIT, numProbes)$.

Output
- $ProbeState_{tex}$ The probe state texture with dimension $numProbes$.

```
backFaceCount = 0
closestFrontFaceDist = MAX_POS_VALUE
for j = 0; j < |Ω_i^ds|; ++j do
    (num_ij, prefix_ij, total_i) ← fetchRayHitInfo(ds_ij, rayHitInfo_tex)
    for k ← [0, num_ij) do
        rayIndex ← j * RAYSLIMIT + k
        rayHitData ← fetchRayHit(i, rayIndex, HitSample_tex)
        if rayHitData.distance < 0 then
            backFaceCount++
        end if
        closestFrontFaceDist = min(closestFrontFaceDist, rayHitData.distance)
    end for
end for

probeFlag = OFF
if closestFrontFaceDist < MAX_POS_VALUE and backFaceCount/total_i < BACKFACE_THRESHOLD then
    probeFlag = ON
end if

Save probeFlag to coordinate i of ProbeState_tex.
```

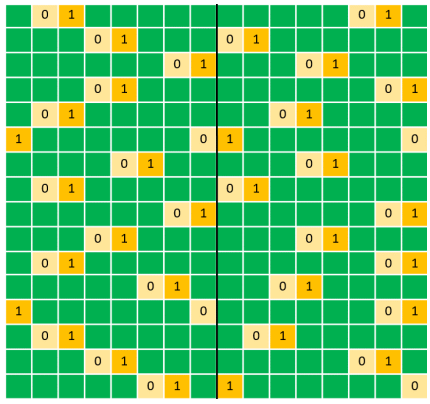

Fig. 1. This diagram shows an example uniform ray allocation for 32 uniform rays. Texels marked 0 are allocated rays in frame 0; texels marked 1 are allocated rays in frame 1.

cells in each row is determined by the formula $\frac{x_i}{16}$ (if $x_i < 16$, each cell covers $\frac{16}{x_i}$ consecutive rows); in addition, each cell also has a number of consecutive texels $\frac{256}{x_i}$. In every frame, only one texel of every cell in a tile is allocated a ray.

Uniform Ray Rotation Our uniform ray rotation method is implemented by shifting one ray allocated in each cell one texel to the right on the ray allocation tile in every frame, as shown in

Algorithm 3 Irradiance and Visibility Integration

Input
- BS — The compute block size for a work group.
- ds_{ij} — The jth direction set of the ith probe.
- $Direction^{int}_{tex}$ — ray directions for integration with dimension $(|\Omega^{ds}_i| * RAYSLIMIT, numProbes)$
- $rayHitInfo_{tex}$ — ray-hit information computed with equation 14, 16, and 17.
- $HitSample_{tex}$ — ray-hit samples (e.g. color and hit distance) with dimension $(|\Omega^{ds}_i| * RAYSLIMIT, numProbes)$.

Output
- $irradiance_{tex}$ — The irradiance texture that stores irr_{ij} per ds_{ij}.
- $visibility_{tex}$ — The visibility texture that stores vis_{ij} per ds_{ij}.

```
hitSharedMem ← [|Ω_i^ds| * RAYSLIMIT]        ▷ amortize texture look-ups among threads in one compute block.
dirSharedMem ← [|Ω_i^ds| * RAYSLIMIT]
for batchIndex = 0; batchIndex < |Ω_i^ds|/BS; ++batchIndex do
    k ← batchIndex * BS + localThreadID
    (num_ik, prefix_ik, total_i) ← fetchRayHitInfo(ds_ik, rayHitInfo_tex)
                                                  ▷ fetch ray-hit data computed using equation 14, 16, and 17.
    for l ← [0, num_ik) do
        rayIndex ← k * RAYSLIMIT + l              ▷ fetch the lth ray sample of the kth direction set on the ith probe.
        rayHitData ← fetchRayHit(i, rayIndex, HitSample_tex)
        rayDirectionData ← fetchRayDir(i, rayIndex, Direction^int_tex)
        ...
        doWorkWithRayHitData(rayHitData, rayDirectionData)
        ...
        hitSharedMem[prefix_ik + l] ← rayHitData
        dirSharedMem[prefix_ik + l] ← rayDirectionData
    end for
end for
sync_threads_in_block()
                  ▷ synchronize threads in a compute block for loading data consecutively in shared memory.
irr_ij, vis_ij ← integrate(hitSharedMem, dirSharedMem, total_i, ds_ij)
                                                  ▷ perform integration for ds_ij using equation 2.
Save irr_ij to irradiance_tex and vis_ij to visibility_tex for ds_ij.
```

Figure 1. We wrap around the ray allocation for each cell to the starting texel of that cell in case the shifting exceeds cell boundary. Note that this rotation method provides the property that every $\frac{256}{x_i}$ frames, all $ds_{ij} \in \Omega^{ds}_i$ are sampled once more.

Random Ray Shifting We implement random ray shifting by generating a random number using probe index i as seed value; this random number is then used to permute the predefined ray allocation pattern differently for every probe. We generate this random number through either GPU or blue-noise methods [1, 3–5].

Adaptive Uniform Ray Strategy Our method keeps track of the number of remaining frames a probe would use x^t_i for its uniform ray allocation. In addition, probe i sees scene changes if and only if $\sum_{j \in |\Omega^{ds}_i|} numDynamicRays_{ij} \neq 0$.

3 Cost Evaluation

We derive formulas for computing additional memory cost of using our IS-DDGI methods in terms of the number of probes, $numProbes$, and the number of direction sets on the probe, $|\Omega^{ds}_i|$, in table 1.

Table 1. Additional Memory Storage Costs

# Textures	One-Ray (bytes)	Multi-Rays (bytes)
Ray Allocation	$1024 * numProbes$	$1024 * numProbes$
Adaptive Uniform Ray	$2 * numProbes$	$2 * numProbes$
Ray-Hit Information	-	$4 * \lvert\Omega_i^{ds}\rvert * numProbes$
Ray Tracing Directions	$(-4) * \lvert\Omega_i^{ds}\rvert * numProbes$	$(-4) * \lvert\Omega_i^{ds}\rvert * numProbes$
Ray Integration Directions	$4 * \lvert\Omega_i^{ds}\rvert * numProbes$	$16 * \lvert\Omega_i^{ds}\rvert * numProbes$
Ray Hit Samples	-	$24 * \lvert\Omega_i^{ds}\rvert * numProbes$
Total	$1026 * numProbes$	$1026 * numProbes + 40 * \lvert\Omega_i^{ds}\rvert * numProbes$

4 Proofs

4.1 Deriving the Expectation of Sum of Dot Products over the Hemisphere

As samples are uniformly distributed on the hemisphere, H_{ij} with normal $\vec{ds}_{ij}$, $p(\omega) = \frac{1}{2\pi}$ where $\omega \in H_{ij}$, we write:

$$
\begin{aligned}
E[\sum_{r\in\Omega_i} \langle \hat{ds}_{ij}, \hat{r}\rangle^+] &= E[\sum_{r\in\Omega^{H_{ij}}} \langle \hat{ds}_{ij}, \hat{r}\rangle^+ + \sum_{r\notin\Omega^{H_{ij}}} \langle \hat{ds}_{ij}, \hat{r}\rangle^+] \\
&= E[\sum_{r\in\Omega^{H_{ij}}} \langle \hat{ds}_{ij}, \hat{r}\rangle + 0] \\
&= E[\sum_{r\in\Omega^{H_{ij}}} \langle \hat{ds}_{ij}, \hat{r}\rangle] \\
&= \sum_{r\in\Omega^{H_{ij}}} E[\langle \hat{ds}_{ij}, \hat{r}\rangle] \\
&= \sum_{r\in\Omega^{H_{ij}}} (\int_{H_{ij}} p(\omega) * \langle \hat{ds}_{ij}, \omega\rangle d\omega) \\
&= \sum_{r\in\Omega^{H_{ij}}} (\frac{1}{2\pi}\int_{H_{ij}} \langle \hat{ds}_{ij}, \omega\rangle d\omega) \\
&= \sum_{r\in\Omega^{H_{ij}}} (\frac{1}{2\pi}\pi) \\
&= \sum_{r\in\Omega^{H_{ij}}} \frac{1}{2} \\
&= \frac{\lvert\Omega^{H_{ij}}\rvert}{2}
\end{aligned}
$$

This means that the expected value of the sum of dot products for ray samples uniformly distributed over the hemisphere of the probe is half of the number of samples on the hemisphere.

4.2 Deriving the Diffuse Radiance

The irradiance in equation 2 also follows the following relation where H_{ij} defines the hemisphere with normal $\vec{ds}_{ij}$:

$$irr_{ij} \approx \frac{2}{|\Omega^{H_{ij}}|} * \sum_{r \in \Omega^{H_{ij}}} \langle \hat{ds}_{ij}, \hat{r} \rangle^{+} * radiance(r)$$

because $E[\sum_{r \in \Omega_i} \langle \hat{ds}_{ij}, \hat{r} \rangle^{+}] = \frac{|\Omega^{H_{ij}}|}{2}$ (Section 4.1). To get the final diffuse radiance r_{diff}, we multiply irradiance with BRDF:

$$r_{diff} = \frac{\rho}{\pi} * \pi * irr_{ij} = \rho * irr_{ij} \tag{1}$$

Note that irr_{ij} does not compute irradiance exactly and is off by a factor of π, as the irradiance formula sample the hemisphere uniformly with probability $p(\omega) = \frac{1}{2\pi}$.

4.3 Deriving the Mixture Probability Distribution for Ray Allocation

We reformulate balance heuristic by formulating a stochastic selection of sampling strategies followed by sampling using a selected sampling strategy:

$$\begin{aligned} w_s(x) &= \frac{N_s p_s(x)}{\sum_k N_k p_k(x)} \\ &= \frac{\frac{N_s}{\sum_l N_l} p_s(x)}{\sum_k \frac{N_k}{\sum_l N_l} p_k(x)} \\ &= \frac{q_s p_s(x)}{\sum_k q_k p_k(x)} \text{ where } q_x = \frac{N_x}{\sum_l N_l} \end{aligned} \tag{2}$$

Plugging equation 2 back into MIS formulation, we have

$$\begin{aligned} F \approx \hat{F} &= \sum_{s=1}^{S} \frac{1}{N_s} \sum_{i=1}^{N_s} w_s(X_i) \frac{f(X_i)}{p_s(X_i)} \\ &= \sum_{s=1}^{S} \frac{1}{N_s} \sum_{i=1}^{N_s} \frac{q_s p_s(X_i)}{\sum_k q_k p_k(X_i)} \frac{f(X_i)}{p_s(X_i)} \\ &= \sum_{s=1}^{S} \frac{1}{N_s} \sum_{i=1}^{N_s} \frac{q_s f(X_i)}{\sum_k q_k p_k(X_i)} \\ &= \sum_{s=1}^{S} \frac{q_s}{N_s} \sum_{i=1}^{N_s} \frac{f(X_i)}{\sum_k q_k p_k(X_i)} \\ &= \sum_{s=1}^{S} \frac{1}{\sum_l N_l} \sum_{i=1}^{N_s} \frac{f(X_i)}{\sum_k q_k p_k(X_i)} \\ &= \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{\sum_k q_k p_k(X_i)} \text{ where } N = \sum_{l=1}^{S} N_l \end{aligned}$$

Hence, this shows that MIS combined with the balance heuristic samples scene changes with a mixture probability density distribution over all sampling strategies.

4.4 Proving Ideal Sampling Distribution Provides Exact Estimates of Scene Changes

Given the ideal ray sampling distribution based on measured scene changes in formula 3

$$p_i(x) = \frac{f_i(x)}{\int_{\Omega_i^{ds}} f_i(x)dx} \tag{3}$$

, we show that it provides exact estimate of scene changes around a probe. Let $\hat{F}_i$ be the Monte-Carlo estimates of F_i. Substituting equation 3 back into $\hat{F}_i$, we get

$$\begin{aligned}
\hat{F}_i &= \frac{1}{N}\sum_{j=1}^{N}\frac{f_i(x_j)}{p_i(x_j)} \\
&= \frac{1}{N}\sum_{j=1}^{N}\frac{f_i(x_j)}{\frac{f_i(x_j)}{\int_{\Omega_i^{ds}} f_i(x)dx}} \\
&= \frac{1}{N}\sum_{j=1}^{N}\int_{\Omega_i^{ds}} f_i(x)dx \\
&= \int_{\Omega_i^{ds}} f_i(x)dx \\
&= F_i
\end{aligned}$$

References

[1] David Blackman and Sebastiano Vigna. 2016. xoroshiro64. https://prng.di.unimi.it/xoroshiro64star.c

[2] Zina H. Cigolle, Sam Donow, Daniel Evangelakos, Michael Mara, Morgan McGuire, and Quirin Meyer. 2014. A Survey of Efficient Representations for Independent Unit Vectors. Journal of Computer Graphics Techniques (JCGT) 3, 2 (17 April 2014), 1–30. http://jcgt.org/published/0003/02/01/

[3] Eric Heitz, Laurent Belcour, V. Ostromoukhov, David Coeurjolly, and Jean-Claude Iehl. 2019. A Low-Discrepancy Sampler That Distributes Monte Carlo Errors as a Blue Noise in Screen Space. In ACM SIGGRAPH 2019 Talks (Los Angeles, California) (SIGGRAPH '19). Association for Computing Machinery, New York, NY, USA, Article 68, 2 pages. https://doi.org/10.1145/3306307.3328191

[4] George Marsaglia. 2003. Xorshift RNGs. Journal of Statistical Software 8, 14 (2003), 1–6. https://doi.org/10.18637/jss.v008.i14

[5] Nathan Reed. 2013. Quick And Easy GPU Random Numbers In D3D11. https://www.reedbeta.com/blog/quick-and-easy-gpu-random-numbers-in-d3d11/