Real-Time Bidirectional Path Tracing via Rasterization Supplemental Material

Yusuke Tokuyoshi*
Square Enix Co., Ltd.

Shinji Ogaki†
Square Enix Co., Ltd.

A Spatio-Temporal Interleaved Sampling

A.1 Spatial Interleaved Sampling

We employ interleaved sampling and geometry-aware filtering [Segovia. et al. 2006] to accelerate computations similar to [Ritschel et al. 2008]. This is done by evaluating a differing sample subset for neighboring pixels. We then use a geometry-aware blur to combine the illumination of neighboring pixels. Such an operation is reasonable because indirect illumination is often of low frequency.

A.2 Temporal Interleaved Sampling

Computing indirect illumination is still too slow to render a high quality image in only one frame. Additionally, in interactive applications such as video games, response is often more important than image quality. In order to increase frames per second and maintain image quality, we exploit temporal coherence in supersampling (See for example [Knecht 2009; Herzog et al. 2010]). In our system we use several indirect illumination caches corresponding to each subpixel, and only one of them is rendered per frame in a round-robin fashion. To improve indirect illumination quality, we evaluate a differing sample subset for neighboring frames similar to spatial interleaved sampling. To compute the final image of the current frame, subpixels are reused from the cache of past frames. This recycling is done by reverse reprojection caching [Nehab et al. 2007]. If the subpixel is detected in the cache, the cached indirect illumination is reused. The reconstructed indirect illumination L of the pixel $\boldsymbol{s}$ is given as:

$$L(\boldsymbol{s}) = \frac{\sum_i^l h_i(\boldsymbol{s}) L_i^c(\gamma_i(\boldsymbol{s}))}{\sum_i^l h_i(\boldsymbol{s})}, \tag{1}$$

where l is the number of subpixels, L_i^c the cached indirect illumination corresponding to the ith subpixel, γ_i the warping function of the pixel position by reverse reprojection, and $h_i(\boldsymbol{s}) = 1$ when a cache hit is detected, otherwise $h_i(\boldsymbol{s}) = 0$. The cache hit is detected using the difference between the reprojected depth z_i and the cached depth z_i^c. The original reverse reprojection caching is given by the following equation:

$$h_i(\boldsymbol{s}) = \begin{cases} 1 & \text{if } |z_i(\boldsymbol{s}) - z_i^c(\gamma_i(\boldsymbol{s}))| < \epsilon, \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where ϵ is the user-specified acceptable range of a cache hit. However, this detection does not take into account the scale of objects and distance. The appropriate ϵ depends on the distance between the camera and an object. This is less accurate for dynamic scenes. Furthermore, this binary detection may produce aliasing. To address these problems, our cache hit detection is based on log-space as shadow mapping [Wimmer et al. 2004] and Gaussian distribution. The equation is given by:

$$h_i(\boldsymbol{s}) = \exp\left(-\frac{\log^2\left(\frac{z_i(\boldsymbol{s})}{z_i^c(\gamma_i(\boldsymbol{s}))}\right)}{2\sigma^2}\right), \tag{3}$$

where σ^2 is the user-specified variance that represents an acceptable range. In this paper, we use $2\sigma^2 = 0.0001$ for all scenes.

This temporal supersampling is similar to amortized supersampling [Yang et al. 2009], but our updating scheme is not amortization. For amortized sampling, nonperiodic random sample points are used, but this causes flickering if the number of samples per second is insufficient. Hence, we use periodic quasi-random numbers, and completely overwrite the cache in the updating process to avoid flickering. We generate different random numbers in each subpixel using a Quasi-Monte Carlo method in preprocessing. These generated constant numbers are scene-independent. Then, we calculate indirect illumination using sample points based on these constant numbers and update each cache in a round-robin fashion. If all the caches hit, the identical and stratified sample points (i.e. all sample points generated) are used in every frame, and we can completely eliminate flickering. To put it the other way around, a cache miss induces flickering around moving objects shown in Figure 1. However, it is less noticeable if motion blur is used.

(a) output image (b) cache hit ratio

Figure 1: *The scene is composed of animated objects lit by an environment map (86942 triangles scene, 55 ray-bundles per frame). (a) The output image. (b) The denominator of Equation (1) (Brightness represents cache hit ratio). Cache miss occurs around the moving objects.*

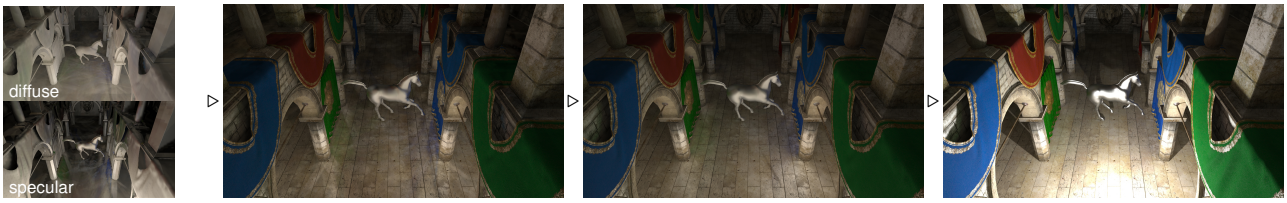
B Implementation Details

B.1 Pipeline

Our system computes two-bounce indirect illumination from point light sources and direct illumination from environment maps. To execute bidirectional path tracing, we add the path tracing procedure to our rendering pipeline. Figure 2 shows the pipeline of our system.

Indirect illumination is computed by iterative multi-pass rendering as instant radiosity. Therefore, the system first creates a G-buffer, reflective shadow maps, and shadow maps of light sources. Cache detection with reverse reprojection is executed simultaneously with G-buffer rasterization. Moreover, we do not create only VPLs but also global ray-bundles. Global ray-bundles are used to trace paths

*e-mail: tokuyosh@square-enix.com
†e-mail: ogaki@square-enix.com



(a) indirect illumination (b) texturing (c) reconstruction with cache (d) addition of direct illumination

Figure 3: *Processing flow of the final image computation (173961 triangles scene, 21 VPLs and 21 ray-bundles per frame). (a) Indirect illumination buffer is computed by bidirectional path tracing with spatial interleaved sampling and geometry-aware filter. (b) The indirect illumination is textured with diffuse maps and specular maps. (c) Reconstruction with the past indirect illumination performs anti-aliasing and error reduction. (d) The final image is obtained with direct illumination.*

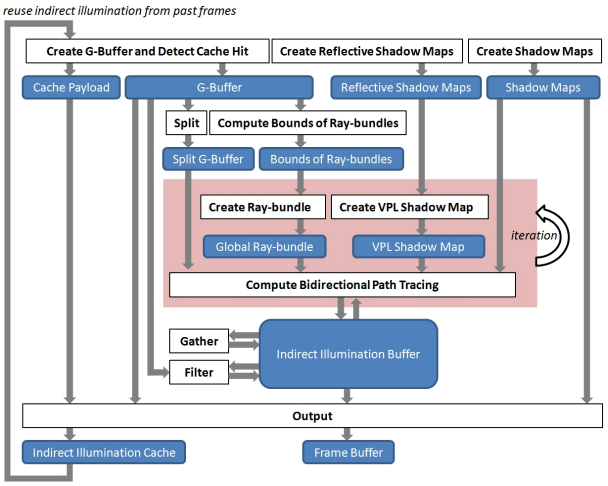


Figure 2: *Rendering pipeline of our system. White blocks: procedures. Blue blocks: memory resources. Gray allows: input-output relationship. Vertex buffer or texture maps are omitted from this figure to simplify. The procedures are executed in order from top to bottom in each frame.*

from a camera. Since we compute only one-bounce camera paths, the screen-space bounding area of the ray-bundles only have to contain all the visible points from a camera, similar to shadow mapping. We compute the bounding volume of the visible points by the G-buffer, and limit the bounds of ray-bundles to reduce alias artifacts.

Next, we compute indirect illumination in an iterative fashion. A VPL and a ray-bundle are randomly generated, and their visibilities are stored into the shadow map and buffer of the ray-bundle, respectively. These visibility tests are done by GPU rasterization. Then, bidirectional path tracing is done by using these visibilities. The computational results of a diffuse and specular term are accumulated into the "Indirect Illumination Buffer" by ping-pong buffering. These results are textured with diffuse maps and specular maps in the post procedure "Output".

The procedures "Split", "Gather" and "Filter" perform spatial interleaved sampling. The procedure "Output" computes texturing of indirect illumination, reconstruction of indirect illumination with "Cache Payload", and direct illumination as shown in Figure 3. It stores the pair of the current indirect illumination (Figure 3 (b)) and the depth value to the "Indirect Illumination Cache". This cache is reused for proceeding frames using temporal coherence to improve the image quality (Figure 3 (c)). Since direct illumination is temporally high frequency, we cache only indirect illumination. Finally, the image with direct illumination added (Figure 3 (d)) is output to the frame buffer.

B.2 Representation of Ray-bundles

Our global ray-bundles are represented by linked-lists [Yang et al. 2010]. This data structure is composed of a head pointer buffer and a node buffer. The head pointer buffer contains the first node index (32 bits UINT type) of the per pixel list. The node buffer has the node data of the list. To represent fragment information, we use 20 bytes per node, as shown in the following structures.

```
struct FragmentData
{
  float depth;      // 31 bits depth and 1 bit face
  uint  normal;     // polar coordinate       (16 bits * 2)
  uint  albedo;     // diffuse and specular   ( 8 bits * 4)
  uint  glossiness; // glossiness and reserved( 8 bits * 4)
};

struct Node
{
  FragmentData data; // payload
  uint         next; // next node index
};
```

Our current system supports RGB diffuse maps, gray scale specular maps, and isotropic roughness. They are stored as 8 bits per element. Three elements (24 bits) are reserved for future extensions such as emissive material. Normal vectors are represented in polar coordinates for compact representation. The face information of a primitive obtained from *SV_IsFrontFace* is stored in the sign bit of the depth value. This face information is used to find a visible front face fragment. Program 1 shows the HLSL code to find the fragment from the node buffer. The input variable "first" is the first node index of the list obtained from the head pointer buffer. The tracing direction is represented by the variable "dir". If the direction is forward, "dir" is 1, otherwise it is -1. The input variable "depth" is the depth value of the starting point of a ray.

The resolution of the head pointer buffer is 256×256 pixels. We allocate 20 M bytes for the node buffer.

B.3 VPL Shadow Maps

We create VPL shadow maps with paraboloid mapping [Brabec et al. 2002]. Since paraboloid mapping is nonlinear, we tessellate polygons with the DirectX 11 tessellator. This approximation is faster than accurate cube mapping if the shadow maps are low-resolution and screen-space level of detail is used, but slower than imperfect shadow maps [Ritschel et al. 2008]. Currently, we adopt a tessellation based approach, because imperfect shadow

(a) spatial interleaved sampling (b) spatio-temporal interleaved sampling (c) cache hit ratio

Figure 4: *The comparison of (a) spatial interleaved sampling and (b) spatio-temporal interleaved sampling (173961 triangles scene, 21 VPLs and 21 ray-bundles per frame). The image (c) shows the cache hit ratio obtained from denominator of Equation (1) (Brightness represents cache hit ratio). The image quality of the cache hit area is improved by temporal interleaved sampling and reconstruction.*

Program 1: *Finding the visible front face fragment.*

```
FragmentData findFragment(in const uint  first,
                          in const float dir,
                          in const float depth )
{
  const float  keyDepth = dir * depth;
  uint         index    = first;
  FragmentData data;
  data.depth = FLT_MAX;

  while( index != HEAD_POINTER_NULL )
  {
    const Node node = nodeBuffer[ index ];

    if( dir * node.data.depth > 0.0  &&
        node.data.depth < data.depth &&
        node.data.depth > keyDepth   )
    {
      data = node.data;
    }

    index = node.next;
  }

  data.depth = abs(data.depth);

  return data;
}
```

maps may produce holes in indirect shadow, and view-adaptive imperfect shadow maps [Ritschel et al. 2011] do not take into account occlusion from camera paths.

The resolution of shadow maps for VPLs is 256×256 pixels. Since reflective shadow maps and shadow maps of direct illumination have to be accurate, we create them with cube mapping. We compute reflective shadow maps with 256×256 resolution for each side. For direct illumination of a point light source, we compute shadow maps with 1024×1024 resolution for each side. The direct shadow lookup is smoothed using a 16-tap percentage-closer filter.

C Results

Figure 4 shows the results of an animated scene rendered with spatio-temporal interleaved sampling. Spatio-temporal interleaved sampling and reconstruction with reverse reprojection improve image quality in a cache hit area with small overhead. Therefore, we can obtain sufficient quality with a small number of samples per frame. However, this approach cannot reduce artifacts such as flickering or aliasing in a cache missed area. Actually, a slight aliasing artifact is visible in the area around the right hind leg of the horse in Figure 4 (b).

D Limitations

Temporal interleaved sampling delays illumination appearance, especially for high-glossy surfaces. If BRDFs are low-frequency, the delay is not distracting in the general case.

References

BRABEC, S., ANNEN, T., AND SEIDEL, H.-P. 2002. Shadow mapping for hemispherical and omnidirectional light sources. In *Proc. of Computer Graphics International*, 397–408.

HERZOG, R., EISEMANN, E., MYSZKOWSKI, K., AND SEIDEL, H.-P. 2010. Spatio-temporal upsampling on the gpu. In *Proc. of I3D 2010*, 91–98.

KNECHT, M. 2009. *Real-Time Global Illumination using Temporal Coherence*. Master's thesis, Vienna University.

NEHAB, D., SANDER, P. V., LAWRENCE, J., TATARCHUK, N., AND ISIDORO, J. R. 2007. Accelerating real-time shading with reverse reprojection caching. In *Proc. of Graphics Hardware 2007*, 25–35.

RITSCHEL, T., GROSCH, T., KIM, M. H., SEIDEL, H.-P., DACHSBACHER, C., AND KAUTZ, J. 2008. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. Graph. 27*, 129:1–129:8.

RITSCHEL, T., EISEMANN, E., HA, I., KIM, J. D., AND SEIDEL, H.-P. 2011. Making imperfect shadow maps view-adaptive: High-quality global illumination in large dynamic scenes. *Comput. Graph. Forum 30*, 2258–2269.

SEGOVIA, B., IEHL, J.-C., MITANCHEY, R., AND PÉROCHE, B. 2006. Non-interleaved deferred shading of interleaved sample patterns. In *Proc. of Graphics Hardware 2006*, 53–60.

WIMMER, M., SCHERZER, D., AND PURGATHOFER, W. 2004. Light space perspective shadow maps. In *Proc. of EGSR 2004*, 143–151.

YANG, L., NEHAB, D., SANDER, P. V., SITTHI-AMORN, P., LAWRENCE, J., AND HOPPE, H. 2009. Amortized supersampling. *ACM Trans. Graph. 28*, 135:1–135:12.

YANG, J. C., HENSLEY, J., GRÜN, H., AND THIBIEROZ, N. 2010. Real-time concurrent linked list construction on the gpu. *Comput. Graph. Forum 29*, 1297–1304.