SRSSIS: Super-Resolution Screen Space Irradiance Sampling for Lightweight Collaborative Web3D Rendering Architecture

Huzhiyuan Long*[1][0009−0004−9910−7322] **, Yufan Yang*[1][0009−0006−3919−0376], Chang Liu[2][0000−0002−3047−1597], and Jinyuan Jia[1][0000−0002−7772−4766]

[1] Tongji University, NO.1239 Siping Road, Shanghai, P.R. China
{huzhiyuan.long,yang_yuyfa}@outlook.com jyjia@tongji.edu.cn
[2] Nanchang Hangkong University, No.696, Fenghe South Avenue, Nanchang, China
lcsszz@nchu.edu.cn

Abstract. In traditional collaborative rendering architecture, the front-end computes direct lighting, which imposes certain performance requirements on the front-end devices. To further reduce the front-end load in complex 3D scenes, we propose a Super-resolution Screen Space Irradiance Sampling technique(SRSSIS), which is applied to our designed architecture, a lightweight collaborative rendering system built on Web3D. In our system, the back-end samples low-resolution screen-space irradiance, while the front-end implements our SRSSIS technique to reconstruct high-resolution and high-quality images. We also introduce frame interpolation in the architecture to further reduce the backend load and the transmission frequency. Moreover, we propose a self-adaptive sampling strategy to improve the robustness of super-resolution. Our experiments show that, under ideal conditions, our reconstruction performance is comparable to DLSS and FSR real-time super-resolution technology. The bandwidth consumption of our system ranges from 8% to 66% of pixel streaming at different super-resolution rates, while the back-end's computational cost is approximately 33% to 46% of pixel streaming at different super-resolution rates.

Keywords: Collaborative rendering · Web3D · Super resolution · Distributed algorithms.

1 Introduction

Accurate lighting models and fast visual feedback are two important factors for real-time (e.g., greater than 30 frames per second) rendering in computer graphics. With the development of computer hardware and the widespread use of real-time GI algorithms, realistic real-time graphics have become possible on high-end platforms. [17].

On the other hand, with the rise of portable devices, users tend to experience 3D content on thin clients such as smartphones and laptops. Web3D has the advantage of good cross-platform compatibility, which enables users to access 3D content on the

* both authors contributed equally.
** corresponding author

web without installing any software or plug-ins, but the disadvantage is limited rendering capability. In pure web-based rendering modes, it is difficult to achieve high-quality real-time graphics on thin clients, which significantly reduces the user's quality of experience (QoE) [22].

How to achieve interactive high-quality graphics on thin clients has gradually become a focus of attention in computer graphics research. One possible solution is remote rendering such as Nvidia Geforce-now [19] and PlayStation Now [27], which are rendering systems centered on cloud servers. Cloud rendering only requires uploading control instructions and decoding videos on the front-end, which can be done on almost any low-power device. However, remote rendering also has some drawbacks, such as high network bandwidth consumption, high server computing power cost and delay sensitive issues.

Another possible solution is collaborative rendering, which is a rendering system that distributes rendering tasks between the client and the server. Collaborative rendering makes full use of the computing power of the client devices, reducing the traffic consumption and the server computing cost. However, due to the direct lighting calculation task on the front-end, the performance requirements are high, while the versatility is low, especially when facing complex scenes such as multiple light sources.

Cloud baking is a representative technique in collaborative rendering that encodes the irradiance contribution of indirect lighting of the scene into an irradiance map and transmits it to client through streaming texture[5, 12, 13, 25]. The client then combines the indirect irradiance with the direct lighting calculation to produce the final image. To reduce transmission bandwidth, Shao W et al.[25] proposed updating only the screen space indirect lighting information each update and updating it to the scene's irradiance map.

To reduce the load on the client, we abandoned the data structure that maintains the indirect lighting of the scene on the client and proposed a collaborative rendering architecture that utilizes server sampling and client reconstruction. In order to reduce the computational and transmission costs of the server while making better use of the client's computing power, we present an innovative real-time super-resolution technique for collaborative rendering.

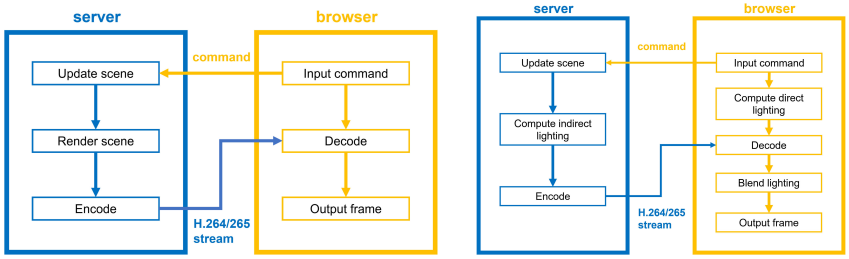
We sample low-resolution screen-space irradiance including direct and indirect illumination at the server, which lacks high-resolution illumination information. Our real-time super-resolution technique reconstructs high-resolution screen-space irradiance on low-resolution irradiance using a joint bilateral filter based on screen-space depth and normal geometry information, which can be obtained at low cost by rasterizing in the client (compared to illumination information). In addition, we exploit the temporal continuity of screen-space irradiance and further reduce both server load and transmission frequency by interpolating irradiance between frames. Finally, the reconstructed irradiance is mixed with albedo to achieve high-quality real-time rendering.

Our technique has several advantages over existing methods. It saves more bandwidth and server cost compared to cloud-centric systems represented by pixel streaming, which require transmitting high-resolution video frames to the client device. And it reduces client performance requirements compared to traditional collaborative ren-

dering systems, which require calculating direct lighting on the client device. Our main contributions compared to previous work are:

1. A collaborative rendering architecture that is interactive, low-latency, and low-cost.
2. The first use of joint/cross bilateral filter for super-resolution in real-time rendering.
3. A proposed Self-adaptive detection filter to assist sampling.
4. A mapping rule for irradiance intensity that is suitable for this architecture.

2 Related Work



(a) Cloud-centric rendering architecture (b) Cloud-end collaborative rendering architecture

Fig. 1: A comparison of architectures between different rendering systems

2.1 Collaborative Rendering

Real-time rendering tasks are divided between the front-end and back-end, and the results are mixed and output to the screen. The mainstream technique of collaborative rendering separates direct and indirect lighting. The front-end is responsible for direct lighting calculations, and the back-end is responsible for indirect lighting calculations. Finally, the lighting is blended and output[5].

Regarding the computation of indirect lighting, there are mainly the following methods:

Irradiance Map: The entire visible surface of the scene is encoded onto a texture map, and the back-end computes the irradiance of the indirect lighting in real-time and transmits the updated irradiance map to the front-end. The front-end mixes it with the direct lighting it has calculated and outputs the final image[5]. There are also optimization measures such as light map trees[12, 13, 25].

Photon Tracing: The back-end computes indirect lighting using photon tracing, updates the front-end's photon information, and calculates the intensity of the indirect lighting using photon density, mixing it with the direct lighting and outputting it to the screen[3, 5].

Voxel & VPL:A sparse voxel octree is used to build voxelized indirect light sources, and the front-end synchronizes this data structure and mixes it with direct lighting to output the frame[5].After voxelization, VPL can also be generated as an indirect lighting representation technique and synchronized with the front-end[14].

Light Probe: Indirect lighting information is stored in light probes, and selective synchronization of light probes is used to transmit lighting information to the front-end, thereby achieving global illumination[28].

Shading Atlas Streaming: All visible object surfaces are shading, encoded onto an atlas texture, and uploaded. The front-end receives the texture and directly obtains the output frame through rasterization[8, 9, 16].

Collaborative rendering fully utilizes the computational power of the front-end. However, there are still some shortcomings:

a. The data structures for maintaining indirect lighting are all proportional to the size of the scene (in order to have high-quality lighting effects at any position in the scene), but for large-scale scenes, the memory consumed by this data structure will be huge, which is still a challenge for portable front-end devices.

b. Limited support for complex scenes. Some front-end devices with low graphics memory bandwidth are unable to support deferred rendering [10], and the forward rendering pipeline is difficult to render dynamic multi-light source scenes. In this case, the front-end is unable to handle the rendering task of direct lighting.

2.2 Cloud Rendering

A remote interactive 3D system based on pixel streaming, where the front-end handles input and uploading instructions, while the back-end receives instructions, processes game logic, renders and encodes frames, and uploads them. Finally, the front-end decodes and displays the frames[11, 21, 26]. Cloud-based gaming solutions rely heavily on back-end computing power and network bandwidth, resulting in high costs. To reduce back-end computation loads and frame transmission costs, alternative methods are needed.

2.3 Real Time Super Resolution

The earliest widely used technique for doubling resolution was Checker Board Rendering (CBR), which relied primarily on reusing temporal information[15, 31]. In recent years, real-time super-sampling has been represented by AMD FSR (FidelityFX Super Resolution)[1] and Nvidia DLSS (Deep Learning Super Sampling)[20]. These techniques use low-resolution images and common rendering engine data, such as depth and motion vectors, as input, and output high-resolution images in real-time.

Neural network-based real-time super-sampling techniques have also emerged in recent years[4, 32–34]. These techniques mainly use CNN or DNN networks to achieve super-resolution effects. For real-time rendering, render time per frame is proportional to the number of pixels in that frame. For rendering complex scenes, it is more efficient to render at lower resolutions and then run a neural network than to render at native resolution. However, for thin clients, simple filtering may be faster than neural networks, and filters are more interpretable than neural networks. On the other hand, filters have scalability. For example, it is difficult to apply the same well-trained network to adapt to any situation for different FOV changes between the server and the client.

2.4 Irradiance Super Sampling Filter

How to efficiently reconstruct signals in real-time with limited sampling information in low sample-per-pixel (SPP) Monte Carlo simulations is a challenging problem. Classic methods mainly rely on edge-aware filtering and joint bilateral filtering based on scene geometry information, with some improvements to achieve good denoising results[2, 6]. Further reuse of temporal information has achieved acceptable results at 1 SPP[23, 24].

Unlike traditional irradiance super-sampling techniques used for denoising, the irradiance obtained by the back-end computation in our system is an accurate and trustworthy value that does not require denoising. Instead, filters are used to cost-effectively super-resolve lighting information.

3 System Design

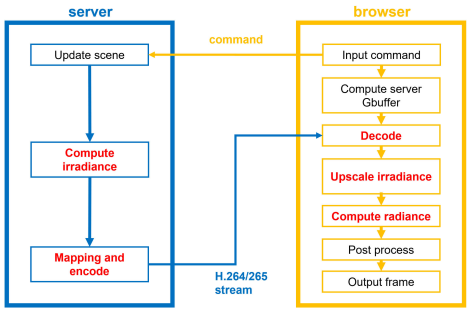


Fig. 2: The architecture of our system

Section 3.1 We describe the overall workflow of our system from a high-level perspective. Section 3.2 We present an approximate estimation technique for irradiance under our framework. Section 3.3, Section 3.4 and Section 3.5 describe the super-resolution technique we use. Section 3.6 We give the mapping rules for irradiance during transmission. Section 3.7 We introduce the synchronization method between the front-end and the back-end in our system. Section 3.8 We provide the technical details of the post-processing techniques applied in our system.

3.1 Overview of System Architecture

Our system consists of four main parts in chronological order:

1. Input and upload of instructions.
2. Rendering of the back-end scene and computation of the low-resolution G-buffer on the front-end.
3. Computation of irradiance, streaming, and super-resolution on the back-end and front-end.
4. Reconstruction of the image.

Figure 2 illustrates the overall pipeline of the front-end and back-end in detail.

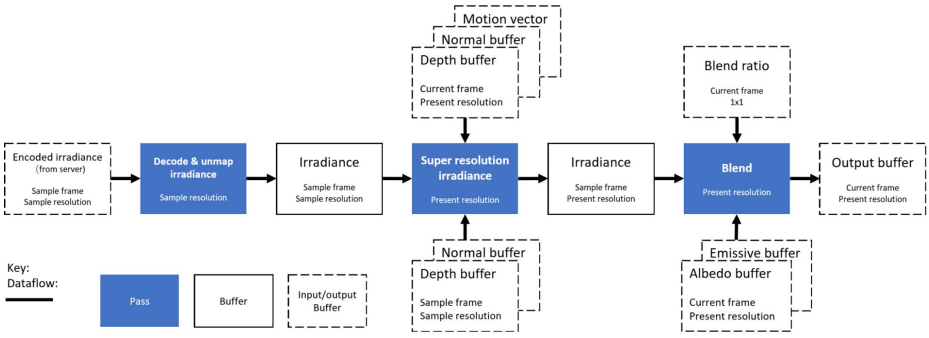


Fig. 3: Data flow diagram of our system

The front-end has two independent rendering pipelines, one for computing the G-buffer information of each pixel under the server-sampled viewpoint, and the other for performing super-resolution and reconstruction of the front-end irradiance.

For the computation of the G-buffer, we need to rasterize the scene under the predefined camera parameters (resolution, FOV, etc.) agreed upon by the front-end and back-end, and obtain the depth buffer in view space and normal buffer in world space. To render the frames, the front-end needs to take the sampled screen-space irradiance, the corresponding G-buffer, and camera information as inputs to produce the result.

The rendering pipeline of the client consists of three passes.

Decode Pixel Streaming. Decode the H.264 encoded pixel streaming and obtain the R8G8B8 format compressed values. Then map them to float points using Equation (9).

Super-resolution Irradiance. Bind the low-resolution G-buffer to the decoded irradiance and take the current frame's G-buffer as input. Compute the irradiance of each pixel by convolving on the irradiance with a joint bilateral filter.

Blend Color Output. Output blended colors by interpolating high-resolution irradiance between two sampled frames if frame interpolation is enabled, and then blending irradiance with albedo.

3.2 Compute Irradiance

According to the rendering equation, the radiance from one point can be represented as:

$$L_o(p,\omega_o) = L_e(p,\omega_o) + \int_\Omega f_r(p,\omega_i,\omega_o) L_i(p,\omega_i)(\omega_i \cdot n) d\omega_i \tag{1}$$

where L_o is the radiance from point p in the direction ω_o, L_e is the emitted radiance, f_r is the BRDF, L_i is the incident radiance, and Ω is the hemisphere of possible incident directions.

Assuming that all objects in the scene are diffuse materials, as the BRDF is independent of direction, the radiance can be expressed as:

$$L_o(p,\omega_o) = L_e(p,\omega_o) + c(p) \int_\Omega L_i(p,\omega_i)(\omega_i \cdot n) d\omega_i \tag{2}$$

where c is the albedo of the material.

Considering the spatial locality, the incident radiance distribution of adjacent pixels is similar, and irradiance can be used to represent its integral to have a good estimate of radiance:

$$L_o(p,\omega_o) = L_e(p,\omega_o) + c(p)I(p) \tag{3}$$

where I is the irradiance at point p.

In practice, render a low-resolution frame without post-processing from the server as radiance input, and calculate the corresponding estimated irradiance:

$$irradiance_{estimated} = \frac{(radiance - emissive)}{albedo} \tag{4}$$

3.3 Super Resolution Irradiance

Algorithm 1: Super-resolution irradiance. The subscript **i** indicates that the variable is related to the inner filter kernel, while the subscript **o** indicates that the variable is related to the outer filter kernel.

Input : position in current screen *pos*
Output: *Irradiance*

$I_i, I_0 \leftarrow 0$
$W_i, W_0 \leftarrow 0$
$p \leftarrow$ `reproject(`pos`)`
`// Initialization.`
for *i in* $window_i(p)$ **do**
  `// Compute with inner filter.`
  $W_i \leftarrow W_i + W_f(s) * W_{Es}(s)$ `// Add filter weight multiplied by Edge-stop weight.`
  $I_i \leftarrow I_i + W_f(s) * W_{Es}(s) * I(s)$ `// Add weighted irradiance.`
end
for *i in* $window_o(p)$ **do**
  `// Compute with outer filter.`
  $W_o \leftarrow W_o + W_f(s) * W_{Es}(s)$
  $I_o \leftarrow I_o + W_o(s) * W_{Es}(s) * I(s)$
end
if $W_o > W_i$ **then**
  `// Decide whether to discard the irradiance sampled through the outer filter.`
  $I \leftarrow I_i + I_0$
  $W \leftarrow W_i + W_0$
else
  $I \leftarrow I_i$
  $W \leftarrow W_i$
end
return I/W

Screen-space irradiance is employed as a medium for transferring lighting information due to the following advantages:

1. The amount of data transferred only depends on the screen resolution and the super-resolution scaling factor, and there is negligible memory overhead for rendering large-scale 3D scenes.

2. the proportion of low-frequency components in irradiance is higher than that in radiance, making it more likely to approach accurate values after joint bilateral filter super-resolution.

To obtain the irradiance of each pixel on the front-end image, the point needs to be first reprojected back to the screen space under the sampling viewpoint, followed by estimating its irradiance using a depth and normal-based joint bilateral filter on the sampled irradiance. Unlike the problems faced by MC ray tracing denoising, our method has noise-free irradiance and G-buffer, and good irradiance estimation can be expected with small filters. The pseudocode is described in Algorithm 1

Reprojection mainly relies on motion vectors [18, 29] in screen space, which can obtain the previous location of the current point.

Our implemented irradiance estimation method mainly refers to the idea in [6]:

$$irradiance = \frac{\sum_{s \in W_p} f(s,p) * w(s,p) * I(s)}{\sum_{s \in W_p} f(s,p) * w(s,p)} \tag{5}$$

Where s is the sampling point, p is the shading point. W_p is the sampling point set covered by the filter kernel centered on the reprojected shading point p. f is the weight assigned by the filter. w is the weight calculated based on Depth and Normal Amplified Edge-stopping function. I is the irradiance of the sampling point.

The formula describes the weighted average sum of the irradiance of sample points near the shading point, with the weight related to the distance between the sample point and the shading point in screen space and whether the two points seem to be on the same plane.

3.4 Self-adaptive Detection Filter

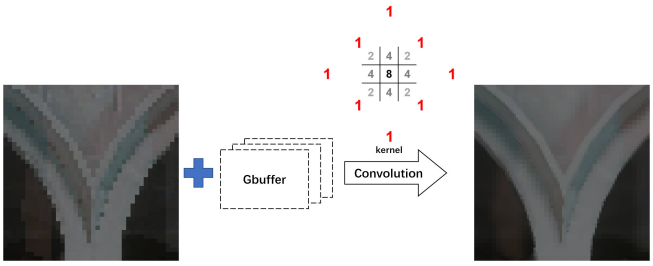


Fig. 4: The application of kernel filter in 2x2 super resolution

Self-adaptive detection filter addresses the issue of low weight sum and low sampling density near the shading point by attempting to sample at a further distance, which

further mitigates shading errors that may occur due to sudden appearance of new mesh surfaces and enhances the robustness of the system.

The shape and weight of the filter kernel are illustrated in Figure 4. The filter kernel consists of two parts, an intrinsic inner layer filter and an adaptive outer layer filter. In most cases, the inner filter is sufficient for accurately estimating irradiance. However, if new regions appear in the view due to camera or object motion, i.e., mesh surfaces that were not sampled under the sampling viewpoint, a wider range of samples is required to estimate accurate irradiance. Thus, an adaptive outer filter is designed for this purpose.

In the experiment, an outer filter with a Manhattan distance of 32 from the center is employed for sampling, which should be adjusted appropriately based on the actual magnification ratio and screen resolution.

3.5 Depth and Normal Amplified Edge-stopping Function

The Depth and Normal Amplified Edge-stopping function utilizes the idea presented in [23], which use depth and normal differences to indicate the similarity in irradiance between sampling points and shading points. By comparing the difference in depth and normal, it is possible to infer whether two points are on the same plane, and then determine the relevance of sampling point irradiance for shading point irradiance.

$$W(s,p) = W_n(s,p) * W_d(s,p) \tag{6}$$

Normal difference. Normal difference is intuitive. If there is a large world normal difference between two points, it implies that they are likely to belong to different planes and have different irradiance, that is, the sampling point cannot offer guidance for the irradiance value of the shading point.

$$W_n(s,p) = max(0.0001, dot(n_s, n_p)^{32}) \tag{7}$$

Where n_s is the normal of the sampling point, n_p is the normal of the shading point.

Depth difference. The effect of normal on depth difference needs to be considered. The formula evaluates the world normal and the vector from the camera to the point to assess the depth difference. When the plane containing the shading point faces the camera, the points on that plane have nearly identical depth; when that plane does not face the camera, even if they are on that plane, their depth values may vary significantly. Therefore, an adaptive strategy is adopted based on the degree of normal difference, where the deeper the angle between the normal and the camera vector of the sampling point, the less impact the depth difference has on the weighting.

$$W_d(s,p) = exp(-\frac{|d_s - d_p|}{0.1 + 0.2 * cos^{-1}(dot(n_p, v))}) \tag{8}$$

Where n_p is the normal of the shading point, v is the vector from the camera to the shading point. d_s is the depth of the sampling point, d_p is the depth of the shading point.

In summary, our super-resolution technique amounts to performing a convolution once every time shading after rasterization. This is acceptable for most thin clients.

3.6 Self-adaptive Irradiance Mapping

The radiance output from the native rendering pipeline ranges from 0 to 1, while the albedo ranges from 0 to 1; thus, it can be inferred that the data range of irradiance is $[0, +\infty)$. However, for most cases, the range of irradiance values remains at a low level. Therefore, a mapping is needed that uses more information to store low-value information and less information to store high-value information. We propose an exponential-based mapping to represent the decoding mapping from compressed values to uncompressed values:

$$irradiance = a * (b^{255 * irradiance_{compressed}} - 1)/255 \tag{9}$$

Where $irradiance_{compressed}$ is the compressed irradiance value ranged from 0 to 1, $irradiance$ is the irradiance computed from Equation (4). a, b are constants calculated from Equation (10)

The encoding mapping is the inverse mapping of the above equation. If the mapped irradiance is greater than 1, it is truncated to 1. To calculate the values of a and b, the maximum value of irradiance estimation $irradiance_{max}$needs to be set a priori, and it is assumed that the mapping in Equation (9) has a gradient of 1 when $irradiance_{compressed} \rightarrow 0$. From this, a system of equations can be derived:

$$\begin{cases} a * (b^{255} - 1) = irradiance_{max} \\ alnb = 1 \end{cases} \tag{10}$$

3.7 Synchronization

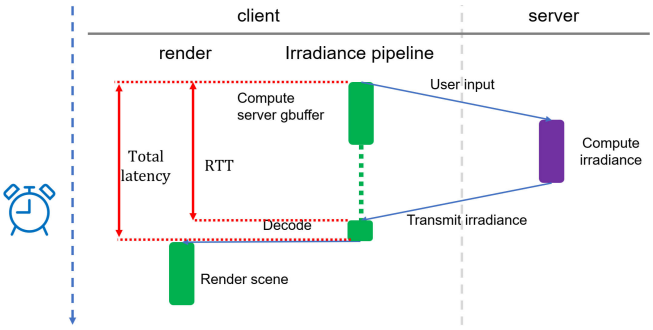


Fig. 5: an ideal process with little network latency

As shown in Figure 5, at the beginning of each frame, the client first uploads user input and starts computing the server-side G-buffer. Once the input is received, the backend begins rendering the scene to obtain radiance, estimates irradiance using Equation (4), and then encodes and transmits it. At this point, the front-end obtains the screen-space irradiance and decodes it, and uses the result as input to render the scene.

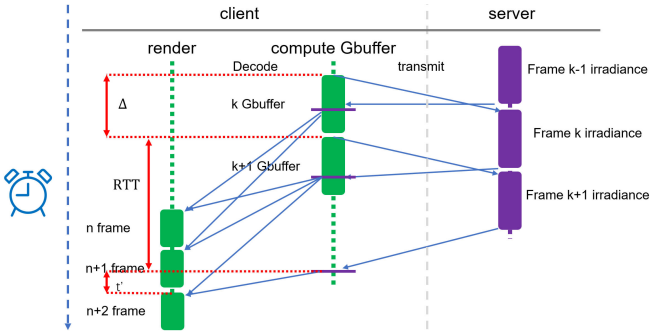


Fig. 6: A synchronizing approach with interpolation

To reduce the bandwidth and computation cost, we propose a synchronization method as shown in Figure 6. Specifically, we delay the use of lighting information and interpolate between irradiance estimates to supplement intermediate frames. Considering the thin client scenario, our method uses linear interpolation to estimate the irradiance. The specific method is as follows:

$$a = min(t^{'}/\Delta, 1) \tag{11}$$

$$irradiance = (1-a) * irradiance^{''} + a * irradiance^{'} \tag{12}$$

Where $irradiance^{'}, irradiance^{''}$ are the estimated values of irradiance for the first closest sampling and the second closest sampling respectively. Δ is the time interval of sampling. $t^{'}$ is the time difference between the current frame and the closest sampled frame. α is the blend ratio in Figure 3.

We use the current rendering frame and the time t' when we obtained the latest sampling information as parameters. t' is used to adjust the ratio of the irradiance estimation values of the second-last and the first-last frames. This results in effective linear interpolation of lighting at the pixel level.

In Figure 6, RTT is mainly determined by the video stream transmission delay. The total latency can be estimated as:

$$Latency = \Delta + RTT \tag{13}$$

3.8 Post-Processing

As the backend is responsible for sampling the illumination information, it needs the radiance in linear space before anti-aliasing and other post-processing effects to compute the irradiance. Conversely, various post-processing effects need to be performed after super resolution on the front-end.

For anti-aliasing, our architecture supports MSAA, which is a widely used anti-aliasing technique that can be used in forward rendering pipeline. This is usually the only pipeline that some thin clients such as mobile devices can afford.

Post-processing effects such as anti-aliasing need to be completed on the client.

4 Results and Discussion

Our approach is implemented using three.js, which is based on WebGL, for the client and UE5.0 for the backend. All front-end tests are conducted on Edge browser with a GeForce RTX 3070 Laptop GPU and a AMD Radeon 780M(an integrated graphics card of AMD). Our evaluation criteria include reconstruction quality, reconstruction time, network bandwidth consumption and computational performance.

4.1 Super Resolution Performance

Scene Polygon count	Room 74.4K				Sponza 262K				Billiards room 1.00M			
Magnification	2.0	3.0	4.0	6.0	2.0	3.0	4.0	6.0	2.0	3.0	4.0	6.0
Ours	**31.84**/86.08	29.93/82.40	28.67/80.25	27.17/77.59	**31.91**/92.05	30.14/89.46	28.75/87.35	27.36/84.73	**33.87**/93.87	**32.34**/92.42	31.40/91.47	29.92/89.89
DLSS3	30.90/**94.31**	**30.90**/**94.31**	N/A	N/A	31.26/**93.93**	**31.25**/**93.92**	N/A	N/A	32.39/**97.11**	32.22/**97.11**	N/A	N/A
FSR2	29.92/91.50	28.80/87.66	N/A	N/A	29.86/90.83	28.32/84.51	N/A	N/A	32.78/95.41	30.26/90.99	N/A	N/A

Table 1: Super resolution ratio and reconstruction quality(PSNR/SSIM) in each test scenario

The proposed technique is compared with the real-time super-resolution techniques DLSS3 and FSR2 in terms of image super-resolution quality. Visual results and quantitative evaluation metrics PSNR and SSIM are provided[30].

In the test scenarios, the front-end and the back-end use the same FOV. The camera orientation of the back-end is also the same as that of the front-end, which maximizes the use of the sampling information from the back-end.

The DLSS and FSR techniques are implemented using UE5.0 plugins in the experiments. All plugin-related parameters are set to default, and the images are generated through Movie Render Queue.

The reference images are generated with MSAAx8 anti-aliasing, and using Movie Render Queue as well. Three scenarios are used for testing: Room, Sponza and Billiards room, with increasing complexity. Since the front-end needs to rasterize the scene, the rasterization speed is affected by the scene size, so the super-resolution time consumption of the three scenes is also different.

According to the experimental results (Figure 7), at the same magnification ratio, our method has clearer details at the model and texture details. This is because our method has high-resolution physical information as guidance, which makes it easier to reconstruct more accurate image compared to other methods.

Due to the difference of their processing of model materials may be different. In terms of results, three.js renders more obvious texture details (such as the ceiling in Room scene in Figure 7), which reduces our SSIM index.

Since we use a convolution-based super-resolution method, our method has a native denoising function. For example, in Figure 7, the noise at the shadow edge on the table in billiards room is caused by the hardware real-time ray tracing of soft shadows in UE. DLSS and FSR cannot handle this kind of noise well, but our method has a better result.

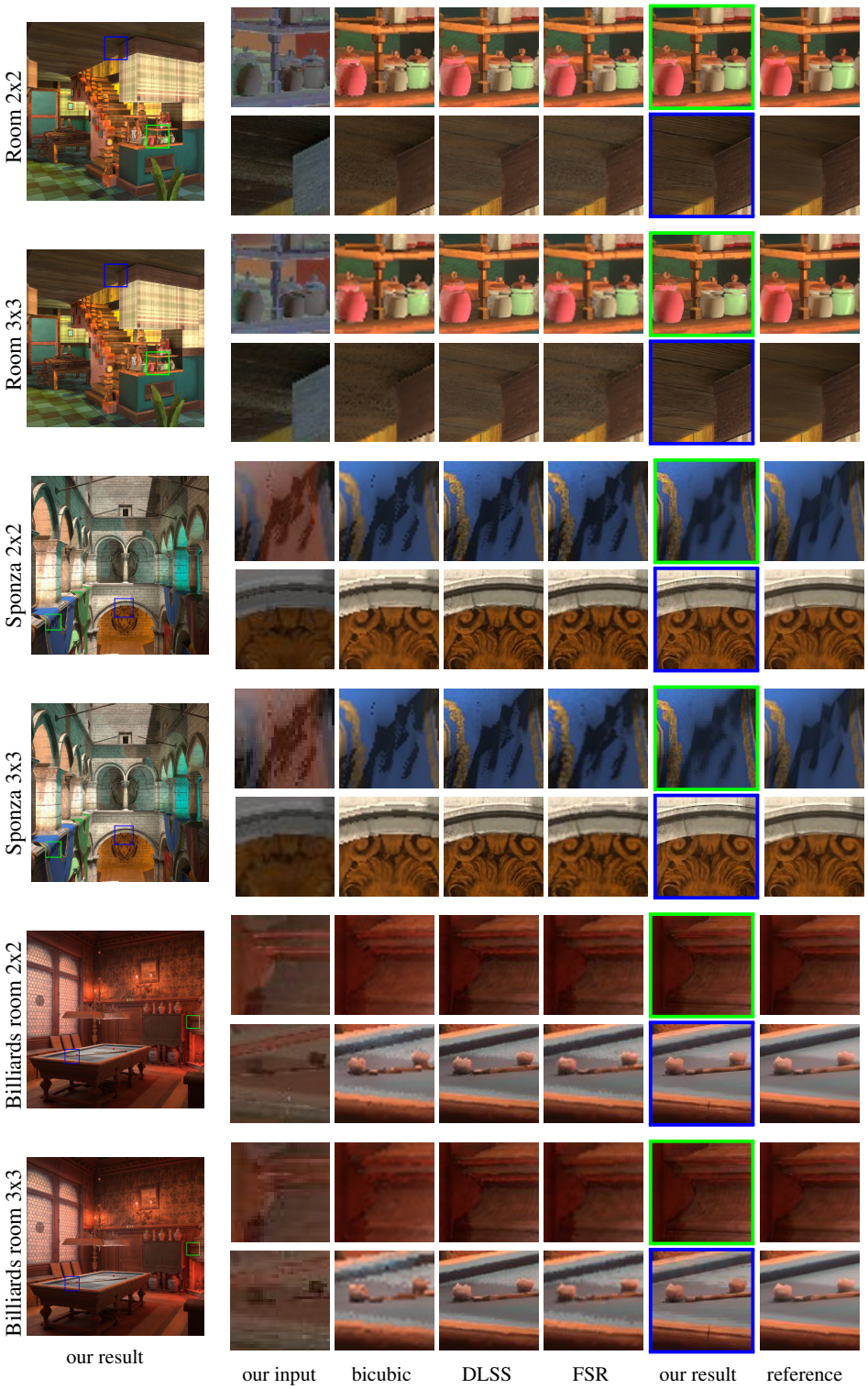


Fig. 7: Visual result of test scenes

Transformation	Move 1m laterally	Move 1m backward	Rotate 15°	Rotate 20°
Enable	25.54	26.88	26.06	26.28
Disable	20.99	24.21	25.98	25.97

Table 2: The performance(PSNR) of outer filter is evaluated in the Sponza scene, which has a size of 29.8 m in length and 18.3 m in width. The scene is rendered with FOV of 90 degrees in 600x600 sample resolution and FOV of 60 degrees in 1200x1200 present resolution.

In Table 2, we test the PSNR under different viewpoint transformations. For the translation case, the improvement of PSNR by our designed self-adaptive detection filter is significant.

(a) Disable outer filter (b) Enable outer filter (c) Reference

Fig. 8: Comparison of the performance of the outer filter while moving 1m laterally

In Figure 8, for the shading points that appear due to viewpoint transformation, a better estimation of irradiance can be obtained by expanding the sampling, thereby improving the rendering quality. For the rotating case, on the one hand, the image quality is not low without the outer filter; on the other hand, when a large area of unsampled mesh appears, it is impossible to complete the rendering based solely on screen space information. Even with the outer filter enabled, the improvement of the image is not significant.

Scene	Room	Sponza	Billiards room
Ours on 780M	1.60	2.07	3.86
Ours on 3070 laptop	0.538	0.754	1.06
Cloud baking on 3070 laptop[25]	/	2.05	/

Table 3: Time consumption(ms) in test scenarios(for 1200x1200 present resolution), which includes rasterization pipeline

We compared the front-end running efficiency of our technique with the implementation of [25]. Since the number of light sources affects the front-end efficiency of the traditional collaborative rendering architecture, we only tested with the addition of one light source in the scene implemented by [25]. Obviously, the front-end efficiency of

our system is much higher than that of the traditional collaborative rendering system, which is friendly to thin clients.

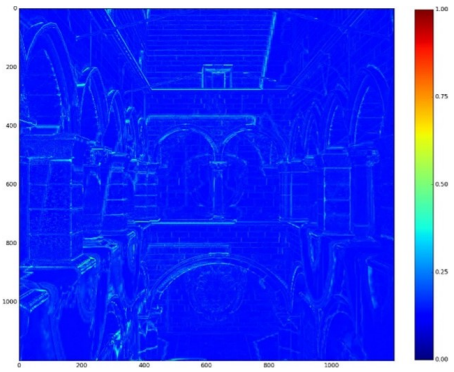


Fig. 9: Error plot between 6x6 super resolution result and a reference image in Sponza. The errors in the image are mainly concentrated on the edges of objects and shadows, which are caused by low sampling frequency. The aliasing of the latter can be improved by changing the sampling method when using irradiance as a texture in the front end (such as bilinear interpolation), but this may increase the overall error of the image.

Figure 9 shows the error plot of our technique at a 6x6 magnification ratio compared to the reference. It gives the sum of absolute values of RGB differences for each pixel. The differences are mainly concentrated at the edges of objects and shadows, both of which are caused by low irradiance sampling frequency.

4.2 Server Load

interpolation magnification	reference	x2.0	x3.0	x4.0	x6.0
disable(60fps)	5.2078	3.8631	1.7838	1.1022	0.5777
enable(15fps)	1.4583	0.9346	0.4364	0.2663	0.1418

Table 4: Consumed network bandwidth(MB/s) across various Magnification

We conducted tests using UE's pixel streaming with H.264 encoder. We transmitted a 60fps, native resolution pixel stream as the reference. The present resolution used in the front-end is 1200x1200. For magnification ratios of x2.0, x3.0, and x4.0, the corresponding pixel streaming resolutions are 600x600, 400x400, and 300x300. For the case of 60fps in the front-end, we test with inserting 3 frames between every two frames.

In the case of only enabling 2x2 super-resolution, the pixel count is 1/4 of the present resolution, and the network bandwidth consumption is about 66% of the original.

The back-end testing uses UE5.0 with lumen enabled on a GeForce RTX 3060 Laptop GPU. In the test scenario, GPU time is the bottleneck of computing power. For

Scene	Room	Sponza	Billiards room
Reference	21.99/21.54	41.13/40.82	34.52/34.22
2x2	8.06/7.74	20.95/20.77	12.90/12.73
3x3	6.44/6.11	17.58/17.43	9.91/9.77
4x4	6.24/5.7	15.75/15.61	9.15/9.02

Table 5: Frame time(ms) and total GPU time(ms) in test scenes

higher magnification ratios, the increase in time consumption is less significant, which may be determined by the parts other than shading in the rendering pipeline.

4.3 Limitation and Future Work

Compared to cloud rendering architectures, our architecture has the same drawback as traditional collaborative rendering architectures, which is that they both require complete geometric information on the client. This means that the necessary geometric information needs to be deployed on the client in advance before providing interactive 3D services. However, since our system is web-based, it can conveniently load resources on the web page, which alleviates this disadvantage to some extent.

In terms of latency, according to Equation (13), the latency of our system is mainly determined by the sampling interval and the pixel streaming latency. If frame interpolation is enabled and a higher number of interpolated frames is used, it will not only reduce the image quality, but also introduce further latency. However, it may be solved by extrapolating screen space irradiance via Extranet [7].

(a) (b)

Fig. 10: Artifacts caused by exceeding the mapping range of the irradiance mapping function. In certain areas of the cloth, the G-channel of irradiance experiences energy loss. Generated with 2x2 super-resolution. (a) Artifacts caused by truncated irradiance.(b) Reference.

In terms of image reconstruction quality, although our reconstruction quality is comparable to DLSS and FSR under ideal situations, our super-resolution technique cannot

replace these real-time super-resolution techniques effectively. If sampling and super-resolution are performed on the same client, an additional high-resolution G-buffer calculation and subsequent interpolation need to be inserted into the original complete rendering pipeline, which is unacceptable in terms of performance. Therefore, our super-resolution technique may only have practical value in collaborative rendering architectures. In addition, our algorithm is hardware platform-independent like FSR, which is friendly to various thin clients.

For regions where the irradiance value is higher than the $irradiance_{max}$ set in Equation (10),as shown in Figure 10, this will result in energy loss. Conversely, if the mapping range of irradiance mapping is too large, this will reduce its representation accuracy and lower the overall image quality.

For errors caused by high-frequency changes of irradiance at shadow edges or object edges, edge sharpening algorithms can be used to sharpen the edges, which may improve the errors that appear in Figure 10.

When a large area of unsampled situation occurs (such as fast movement or rotation of the camera), our method may perform poorly. This is the limitation of only having screen space information in the front-end under high-latency lighting. The following solutions can be considered:

1. Use a larger FOV when sampling irradiance to obtain wide-angle information.

2. The interaction latency of the client can be increased so that it is synchronized with the pixel stream, to avoid unsampled mesh surfaces appearing in the field of view.

3. Similar to collaborative rendering methods based on shading atlas streaming [8, 9, 16]that also only maintain screen space information in the front-end, we can add more sampling viewpoints to improve the robustness of sampling by using a similar idea to [9].

5 Conclusion

Collaborative rendering is a way to solve the high service cost of remote rendering and provide interactive high-quality 3D applications services. The Web3D solution makes this technique more user-friendly, which means that users can enjoy high-quality, low-service-cost 3D content brought by SRSSIS architecture without the need of a client, using only a browser. Our system further improves the traditional cloud baking architecture, playing to its strengths, and avoiding its weaknesses, and further reduces the front-end load, making it more available on thin clients.

In addition, we introduce the traditional technique in MC ray tracing denoising to solve the real-time super-resolution problem. Based on high interpretability, it also shows good performance in practice, which is a valuable exploration.

Acknowledgements

This research is partially supported by the Basic Grant of Natural Science Foundation of China (No.62072339), the Key Project of Regional Joint Grant of Science Natural Foundation of China (No.U19A2063) and a grant from the National Natural Science Foundation of China (No.62262043).

Bibliography

[1] AMD: Amd fidelityFX super resolution. Website (2023), https://www.amd.com/en/technologies/fidelityfx-super-resolution

[2] Bauszat, P., Eisemann, M., Magnor, M.: Guided image filtering for interactive high-quality global illumination. In: Computer Graphics Forum. vol. 30, pp. 1361–1368. Wiley Online Library (2011)

[3] Bugeja, K., Debattista, K., Spina, S.: An asynchronous method for cloud-based rendering. The Visual Computer **35**, 1827–1840 (2019)

[4] Caballero, J., Ledig, C., Aitken, A., Acosta, A., Totz, J., Wang, Z., Shi, W.: Real-time video super-resolution with spatio-temporal networks and motion compensation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4778–4787 (2017)

[5] Crassin, C., Luebke, D., Mara, M., McGuire, M., Oster, B., Shirley, P., Wyman, P.P.S.C.: Cloudlight: A system for amortizing indirect lighting in real-time rendering. Journal of Computer Graphics Techniques Vol **4**(4), 1–27 (2015)

[6] Dammertz, H., Sewtz, D., Hanika, J., Lensch, H.P.: Edge-avoiding a-trous wavelet transform for fast global illumination filtering. In: Proceedings of the Conference on High Performance Graphics. pp. 67–75 (2010)

[7] Guo, J., Fu, X., Lin, L., Ma, H., Guo, Y., Liu, S., Yan, L.Q.: Extranet: Real-time extrapolated rendering for low-latency temporal supersampling. ACM Transactions on Graphics (TOG) **40**(6), 1–16 (2021)

[8] Hladky, J., Seidel, H.P., Steinberger, M.: Tessellated shading streaming. In: Computer Graphics Forum. vol. 38, pp. 171–182. Wiley Online Library (2019)

[9] Hladky, J., Stengel, M., Vining, N., Kerbl, B., Seidel, H.P.: Quadstream: A quad-based scene streaming architecture for novel viewpoint reconstruction. ACM Transactions on Graphics (TOG) p. C32 (2022)

[10] Kaplanyan, A.: Cryengine 3: Reaching the speed of light. Talk, SIGGRAPH (2010)

[11] Laghari, A.A., He, H., Memon, K.A., Laghari, R.A., Halepoto, I.A., Khan, A.: Quality of experience (qoe) in cloud gaming models: A review. multiagent and grid systems **15**(3), 289–304 (2019)

[12] Liu, C., Ooi, W.T., Jia, J., Zhao, L.: Cloud baking: Collaborative scene illumination for dynamic web3d scenes. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) **14**(3s), 1–20 (2018)

[13] Liu, C., Song, H., Fang, T., Ou, Q., Yu, G., You, T., Ying, M., et al.: Web-cloud collaborative mobile online 3d rendering system. Security and Communication Networks **2022** (2022)

[14] Magro, M., Bugeja, K., Spina, S., Debattista, K.: Cloud-based dynamic gi for shared vr experiences. IEEE Computer Graphics and Applications **40**(5), 10–25 (2020)

[15] Mansouri, J.E.E.: Rendering 'rainbow six | siege'. Website (2016), https://www.gdcvault.com/play/1022990/Rendering-Rainbow-Six-SiegeGDC,

[16] Mueller, J.H., Voglreiter, P., Dokter, M., Neff, T., Makar, M., Steinberger, M., Schmalstieg, D.: Shading atlas streaming. ACM Transactions on Graphics (TOG) **37**(6), 1–16 (2018)

[17] Myszkowski, K., Tawara, T., Akamine, H., Seidel, H.P.: Perception-guided global illumination solution for animation rendering. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. pp. 221–230 (2001)

[18] Nehab, D., Sander, P.V., Lawrence, J., Tatarchuk, N., Isidoro, J.R.: Accelerating real-time shading with reverse reprojection caching. In: Graphics hardware. vol. 41, pp. 61–62 (2007)

[19] NVIDIA: GeForce NOW. Website (2023), https://www.nvidia.com/en-us/geforce-now/

[20] NVIDIA: NVIDIA DLSS. Website (2023), https://www.nvidia.com/en-us/geforce/technologies/dlss/

[21] Peñaherrera-Pulla, O.S., Baena, C., Fortes, S., Baena, E., Barco, R.: Measuring key quality indicators in cloud gaming: Framework and assessment over wireless networks. Sensors **21**(4), 1387 (2021)

[22] Perkis, A., Timmerer, C., Baraković, S., Barakovic, J., Bech, S., Bosse, S., Botev, J., Brunnström, K., da Silva Cruz, L.A., Moor, K.D., de Polo Saibanti, A., Durnez, W., Egger-Lampl, S., Engelke, U., Falk, T.H., Hameed, A., Hines, A., Kojić, T., Kukolj, D., Liotou, E., Milovanovic, D., Möller, S., Murray, N., Naderi, B., Pereira, M., Perry, S.W., Pinheiro, A.M.G., Palacios, A.P., Raake, A., Agrawal, S., Reiter, U., Rodrigues, R., Schatz, R., Schelkens, P., Schmidt, S., Sabet, S.S., Singla, A., Skorin-Kapov, L., Suznjevic, M., Uhrig, S., Vlahovic, S., Voigt-Antons, J.N., Zadtootaghaj, S.: Qualinet white paper on definitions of immersive media experience (imex). ArXiv **abs/2007.07032** (2020)

[23] Schied, C., Kaplanyan, A., Wyman, C., Patney, A., Chaitanya, C.R.A., Burgess, J., Liu, S., Dachsbacher, C., Lefohn, A., Salvi, M.: Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination. In: Proceedings of High Performance Graphics, pp. 1–12 (2017)

[24] Schied, C., Peters, C., Dachsbacher, C.: Gradient estimation for real-time adaptive temporal filtering. Proceedings of the ACM on Computer Graphics and Interactive Techniques **1**(2), 1–16 (2018)

[25] Shao, W., Liu, C., Jia, J.: Lightmap-based gi collaborative rendering system for web3d application. Journal of System Simulation **32**(4), 649 (2020)

[26] Shea, R., Liu, J., Ngai, E.C.H., Cui, Y.: Cloud gaming: architecture and performance. IEEE network **27**(4), 16–21 (2013)

[27] SONY: PlayStation Now. Website (2023), https://www.playstation.com/en-us/ps-now/

[28] Stengel, M., Majercik, Z., Boudaoud, B., McGuire, M.: A distributed, decoupled system for losslessly streaming dynamic light probes to thin clients. In: Proceedings of the 12th ACM Multimedia Systems Conference. pp. 159–172 (2021)

[29] Walter, B., Drettakis, G., Parker, S.: Interactive rendering using the render cache. In: Rendering Techniques' 99: Proceedings of the Eurographics Workshop in Granada, Spain, June 21–23, 1999 10. pp. 19–30. Springer (1999)

[30] Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. IEEE transactions on image processing **13**(4), 600–612 (2004)

[31] Wihlidal, G.: 4k checkerboard in 'battlefield 1' and 'mass effect andromeda'. Website (2017), https://www.gdcvault.com/play/1022990/Rendering-Rainbow-Six-SiegeGDC

[32] Xiao, L., Nouri, S., Chapman, M., Fix, A., Lanman, D., Kaplanyan, A.: Neural supersampling for real-time rendering. ACM Transactions on Graphics (TOG) **39**(4), 142–1 (2020)

[33] Zhan, Z., Gong, Y., Zhao, P., Yuan, G., Niu, W., Wu, Y., Zhang, T., Jayaweera, M., Kaeli, D., Ren, B., et al.: Achieving on-mobile real-time super-resolution with neural architecture and pruning search. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 4821–4831 (2021)

[34] Zhang, X., Zeng, H., Zhang, L.: Edge-oriented convolution block for real-time super resolution on mobile devices. In: Proceedings of the 29th ACM International Conference on Multimedia. pp. 4034–4043 (2021)