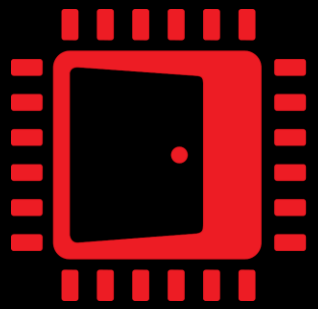


AMD
GPUOpen

GDC

GLOBAL ILLUMINATION WITH AMD FIDELITYFX™ BRIXELIZER, PLUS AMD FIDELITYFX SDK UPDATES

DIHARA WIJETUNGA

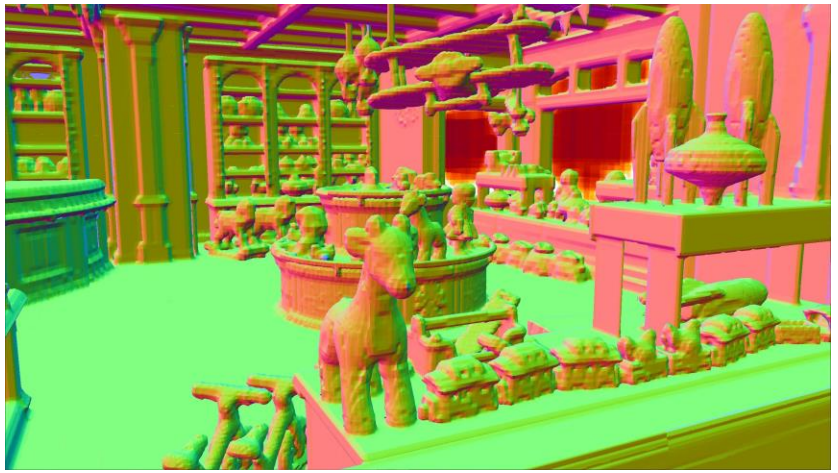
AMD
together we advance_

AGENDA

- Brixelizer recap
- Brixelizer GI overview
- Algorithm
- Ground truth comparisons
- Performance
- Integration
- Demo
- AMD FidelityFX SDK updates
- Conclusion

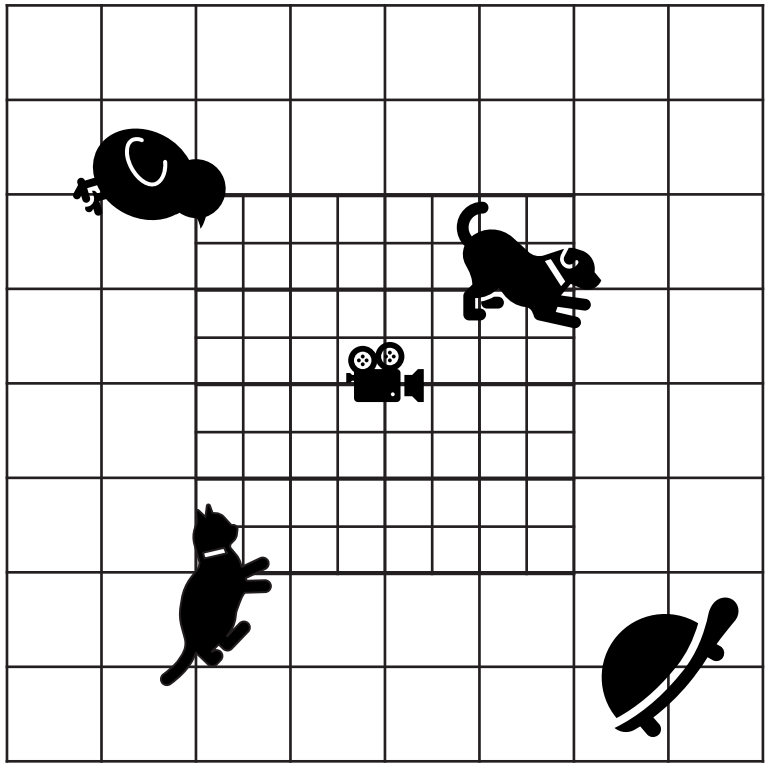
BRIXELIZER RECAP

- Real-time sparse distance field builder that takes triangle geometry as input
- Works with static and dynamic geometry
- Shader API to trace rays against the distance field
- To be available in the next AMD FidelityFX SDK



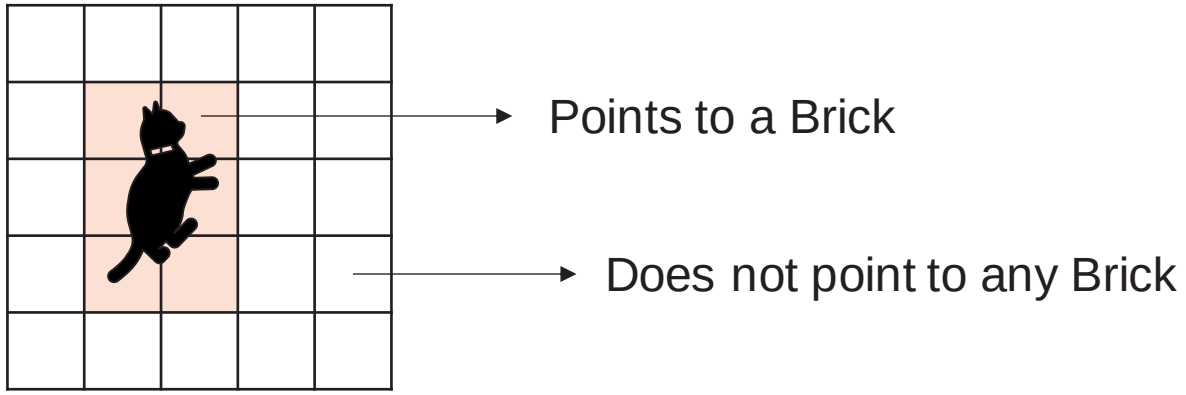
BRIXELIZER RECAP

- Generates cascades of sparse distance fields around a given position
  - Typically the camera
- Each cascade is split into 64x64x64 voxels



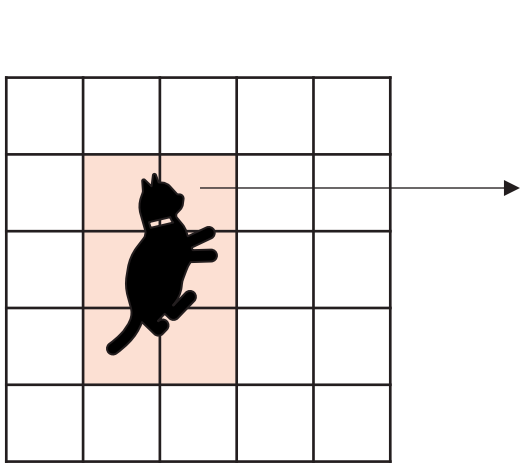
BRIXELIZER RECAP

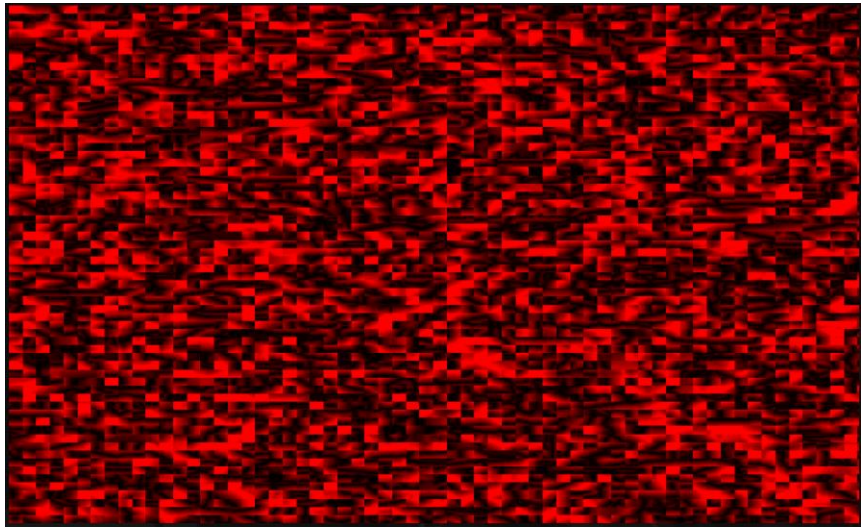
- If a voxel intersects any geometry, a local distance field is generated for said voxel
- This local distance field is known as a Brick



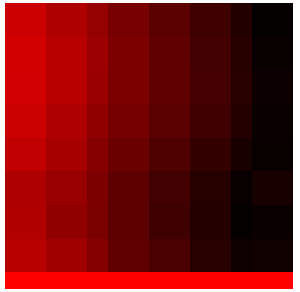
BRIXELIZER RECAP

- A Brick is of resolution 8x8x8 and contains the local distance field of a grid cell, that intersected with a surface
- All cascades will allocate their Bricks in the same 3D Brick atlas
- A texel within a Brick is known as a Brixel

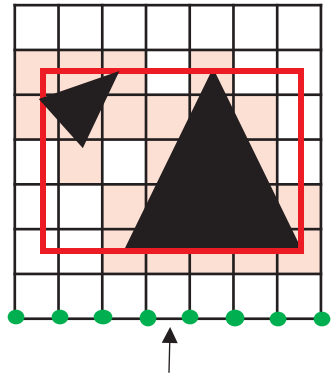




3D Brick atlas, R8_UNORM



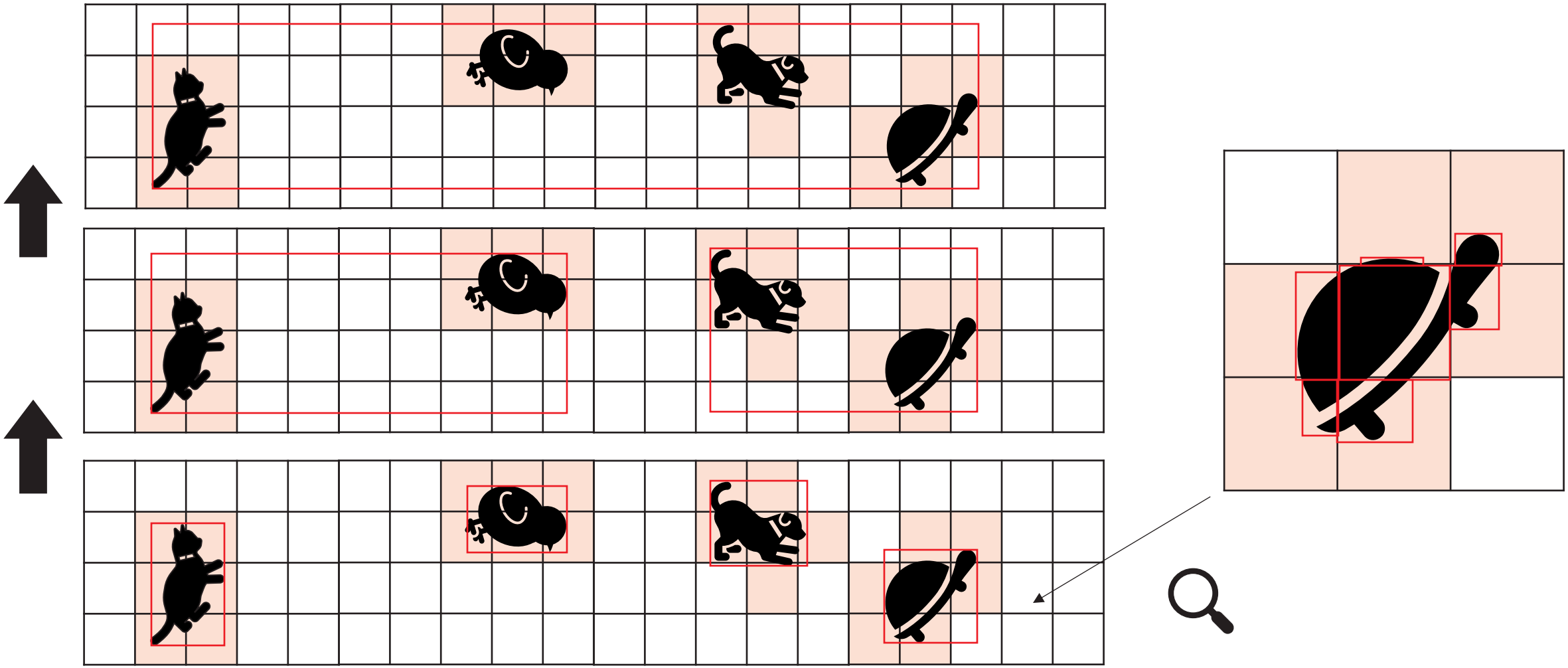
Brick



Brixel

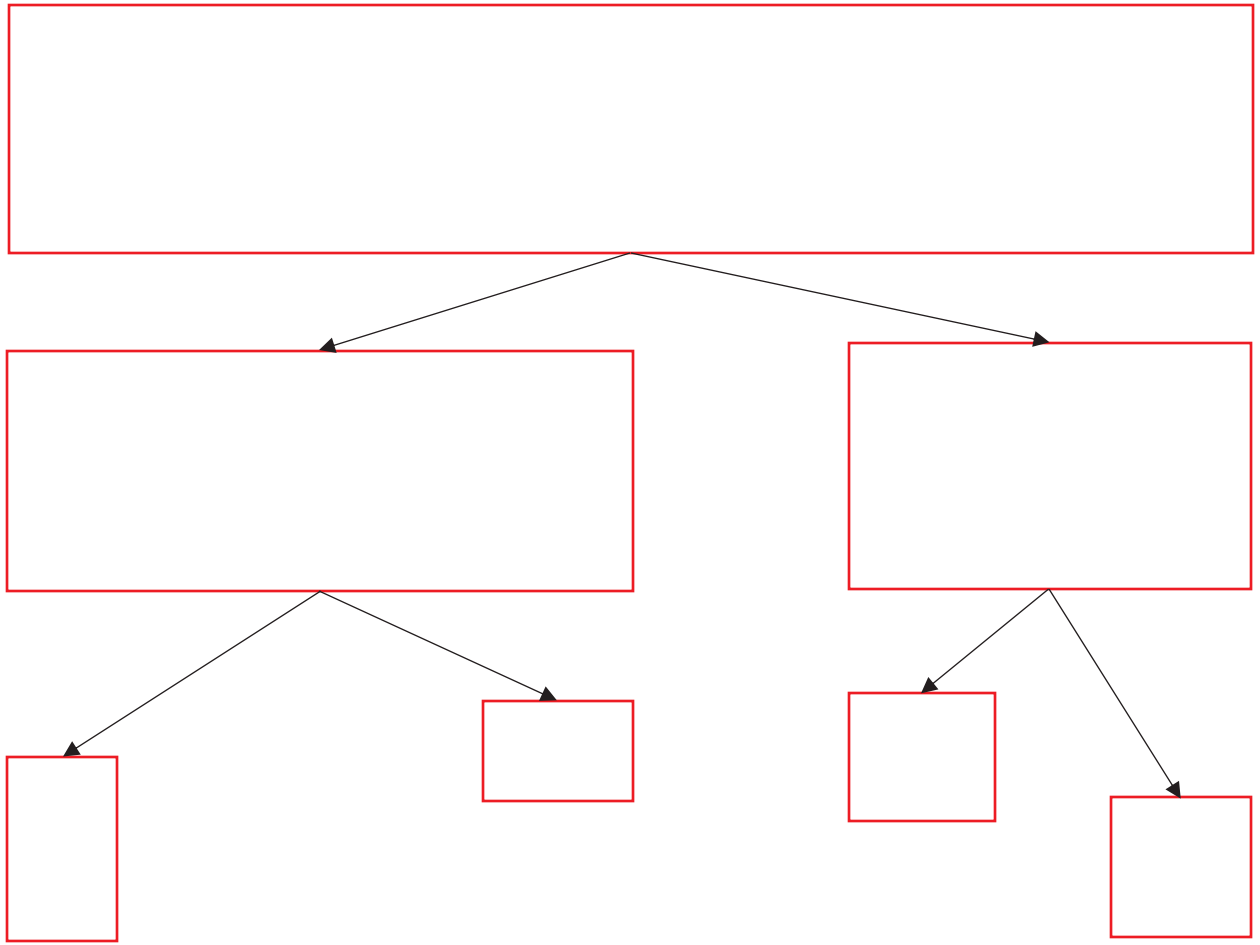
BRIXELIZER RECAP

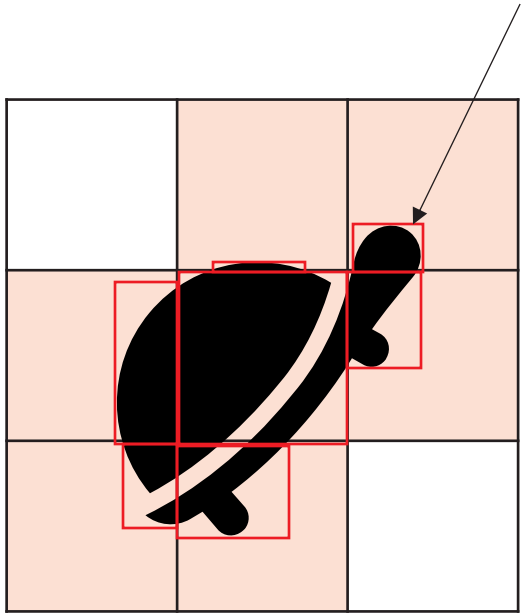
- Once the Bricks are generated, they are used to build bottom up a 3-level AABB tree for each cascade



BRIXELIZER RECAP

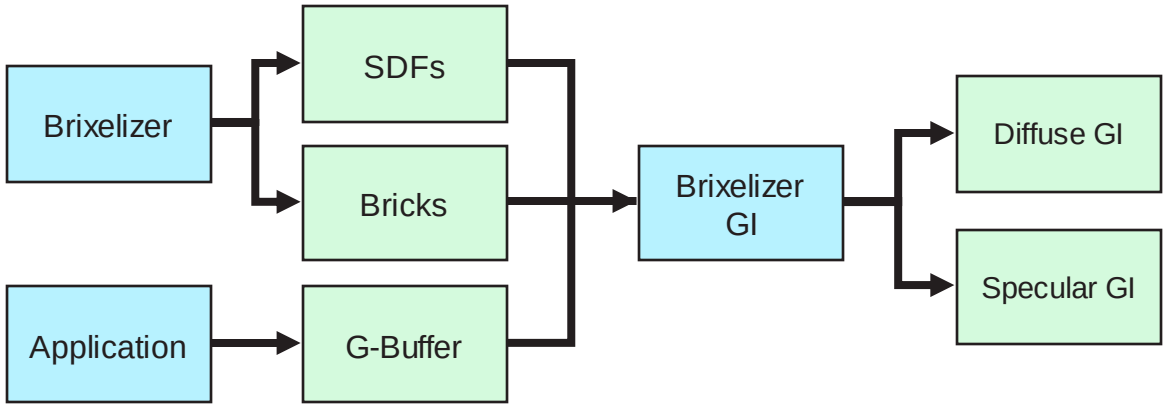
- The AABB tree is used for ray-scene traversal





BRIXELIZER GI

- Fast and approximate dynamic global illumination solution built upon Brixelizer
  - No hardware-accelerated ray-tracing
  - Purely compute-based
- A fallback for ray-traced global illumination on lower-end hardware
- New addition to the AMD FidelityFX SDK
- Provided as a library that complements Brixelizer
- C++ and HLSL/GLSL source to be available under the MIT license



BRIXELIZER GI

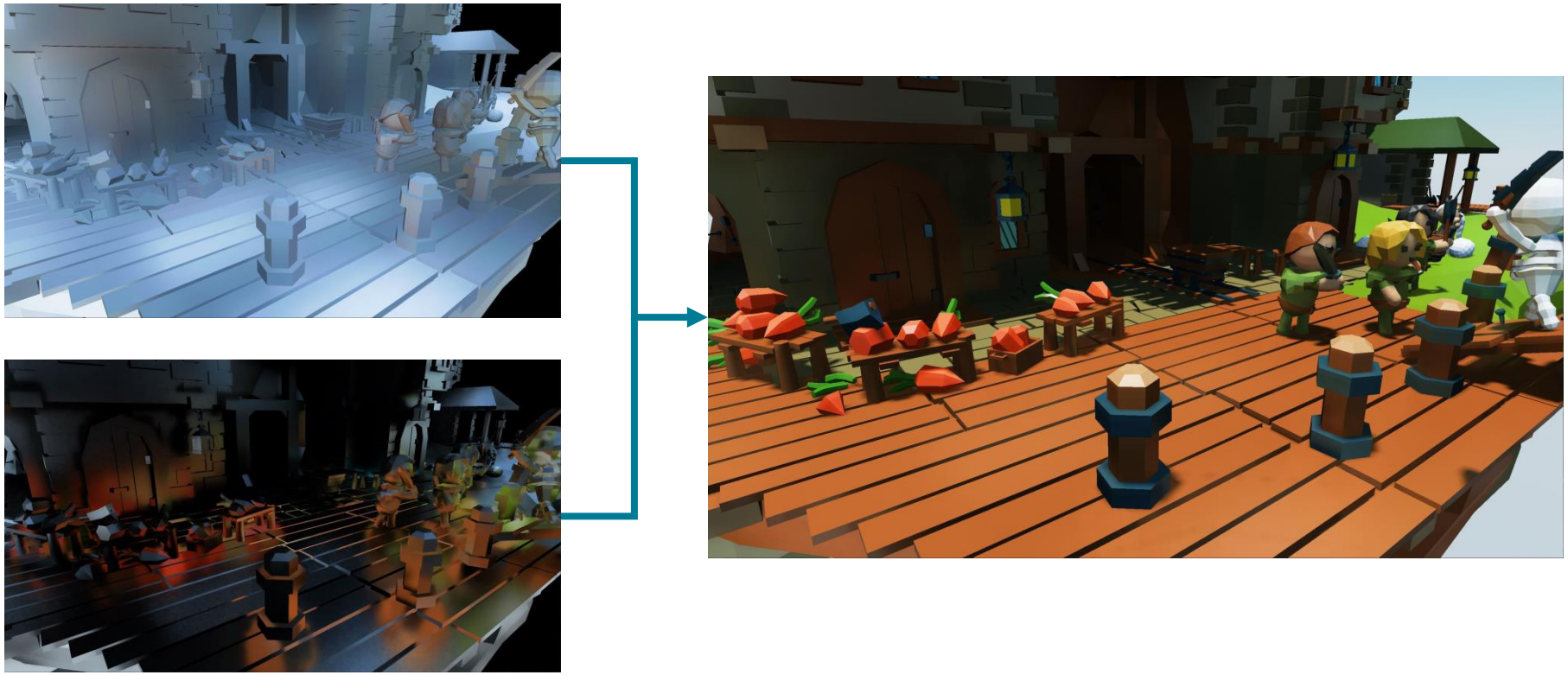
BRIXELIZER GI

BRIXELIZER GI

BRIXELIZER GI

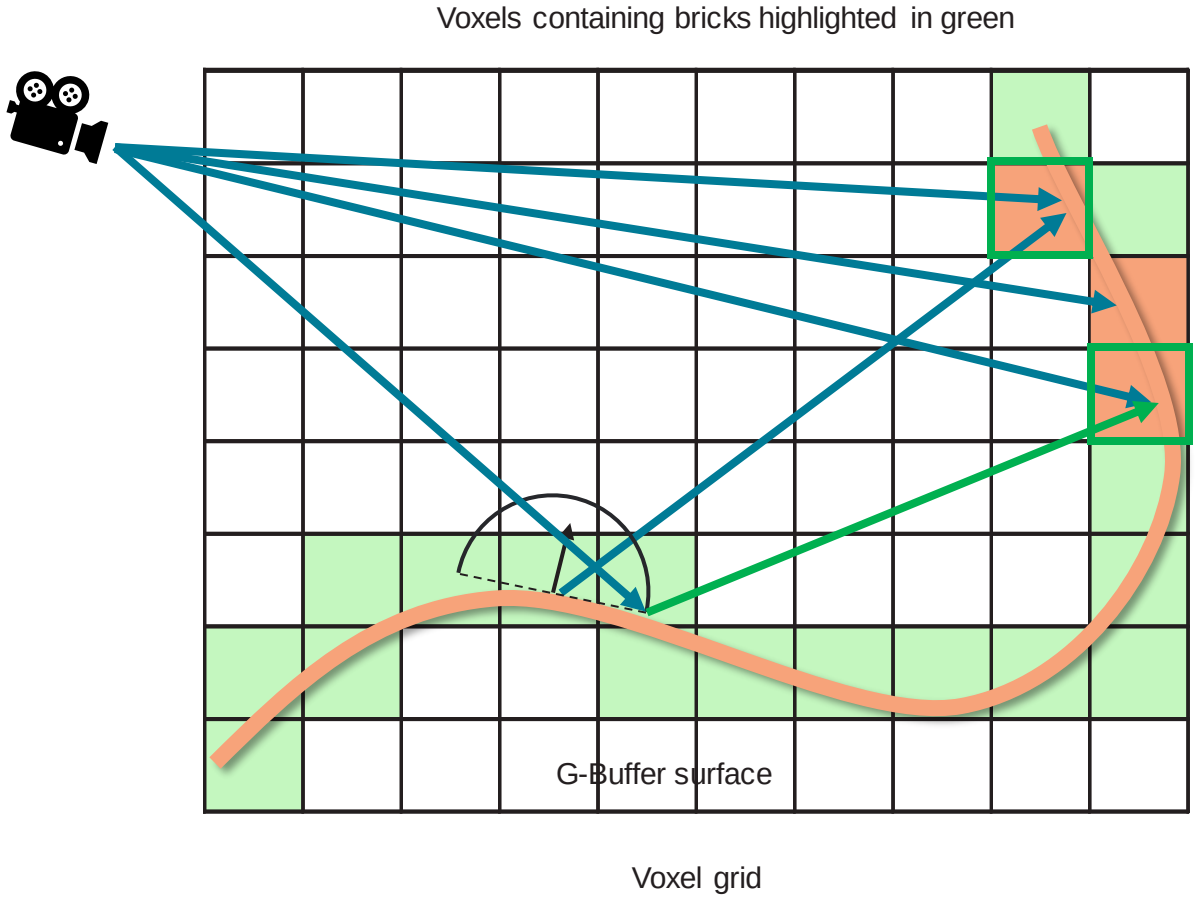
ALGORITHM

- Based on AMD GI-1.0
  - Two-level caching approach
  - Screen Space probes backed by a World Space radiance and irradiance cache

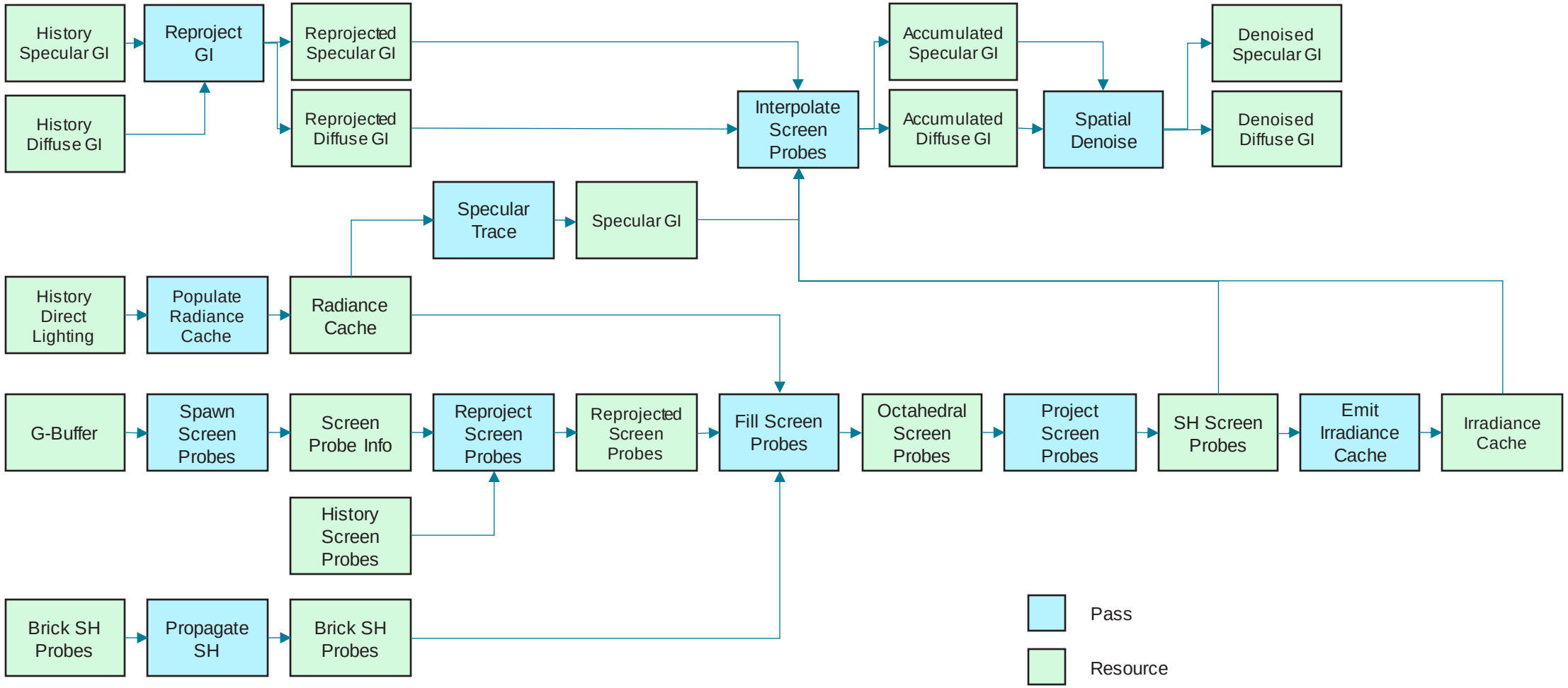


ALGORITHM

- Inject direct lighting from last frame into radiance cache
- Spawn screen probes on G-Buffer
- Trace diffuse rays from screen probes and sample radiance cache for shading
- Feed world-space irradiance cache using screen probes
- Trace specular rays from G-Buffer and sample radiance cache for shading
- Resolve diffuse GI using screen probes and irradiance cache

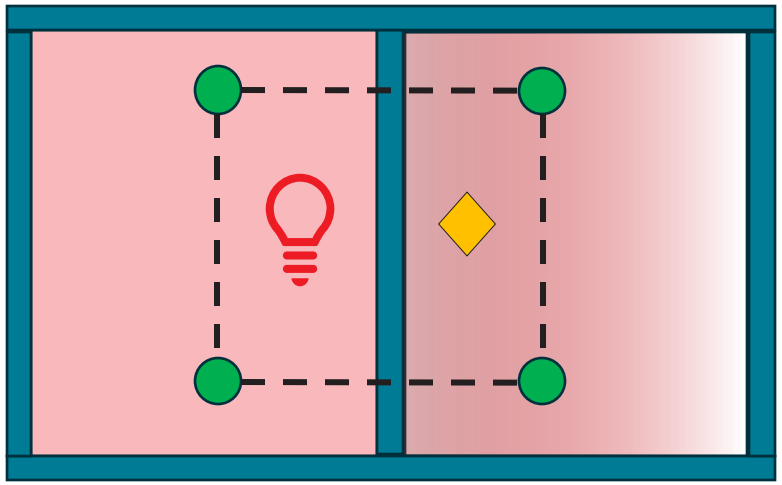


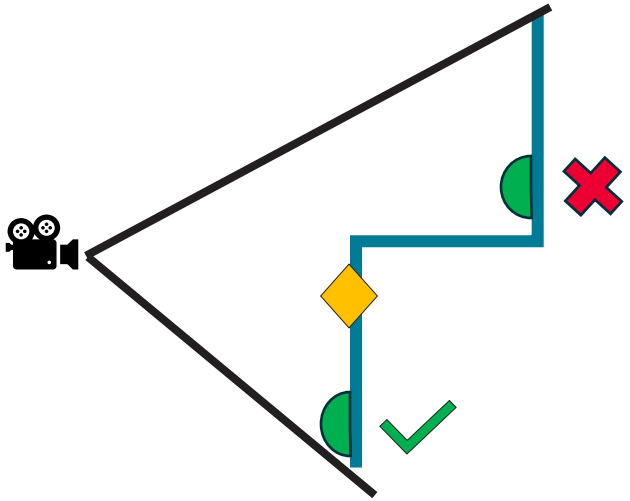
ALGORITHM


History Specular GI
Reproject GI
Reprojected Specular GI
History Diffuse GI
Reprojected Diffuse GI
Interpolate Screen Probes
Accumulated Specular GI
Accumulated Diffuse GI
Spatial Denoise
Denoised Specular GI
Denoised Diffuse GI
Specular Trace
Specular GI
History Direct Lighting
Populate Radiance Cache
Radiance Cache
G-Buffer
Spawn Screen Probes
Screen Probe Info
Reproject Screen Probes
Reprojected Screen Probes
Fill Screen Probes
Octahedral Screen Probes
Project Screen Probes
SH Screen Probes
Emit Irradiance Cache
Irradiance Cache
History Screen Probes
Brick SH Probes
Propagate SH
Brick SH Probes
Pass
Resource

ALGORITHM

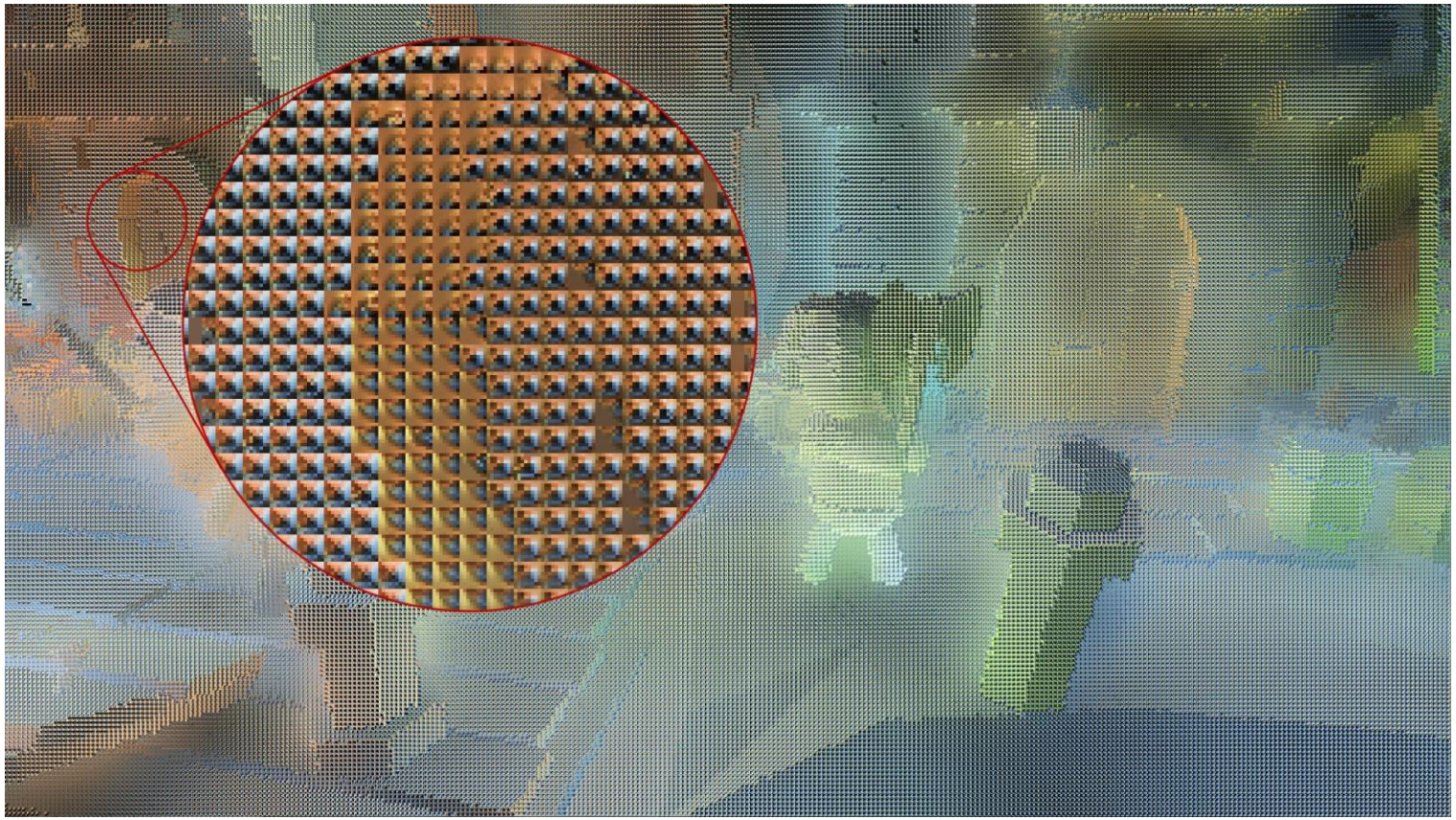
- Why screen probes?
- A common problem with traditional probes is light leaking
- DDGI solves this to an extent
  - Stores depth values at each probe
- Screen probes sidestep the issue by placing probes only on visible surfaces





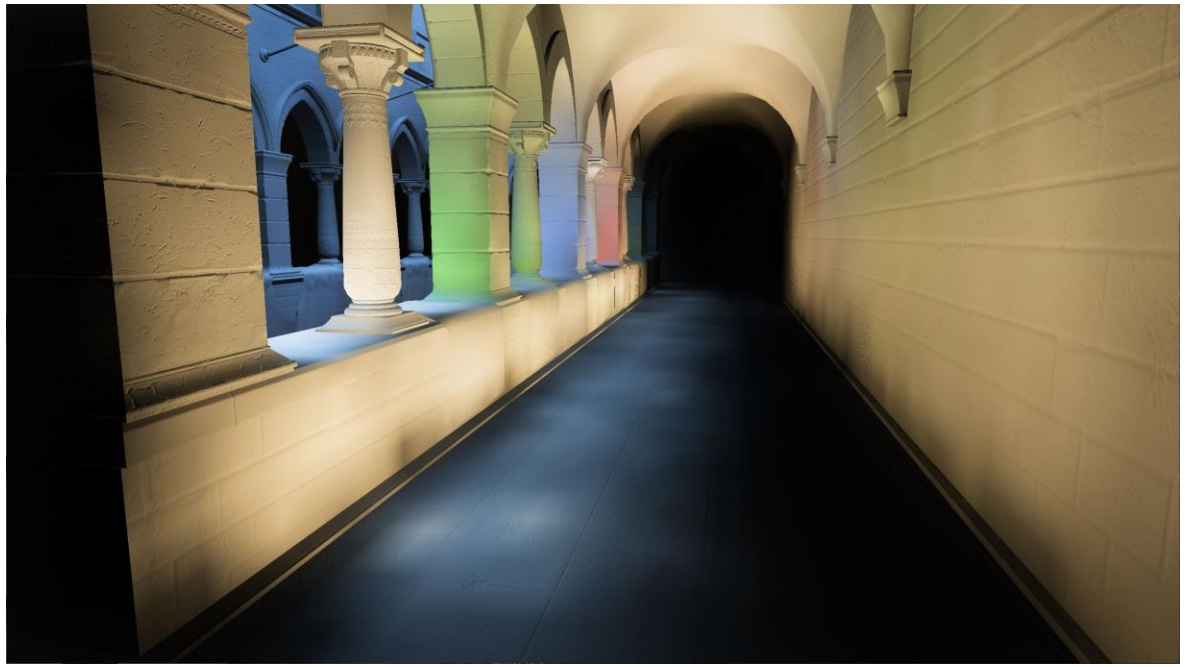
ALGORITHM

- Screen Probes represented using an 8x8 octahedral encoding



ALGORITHM

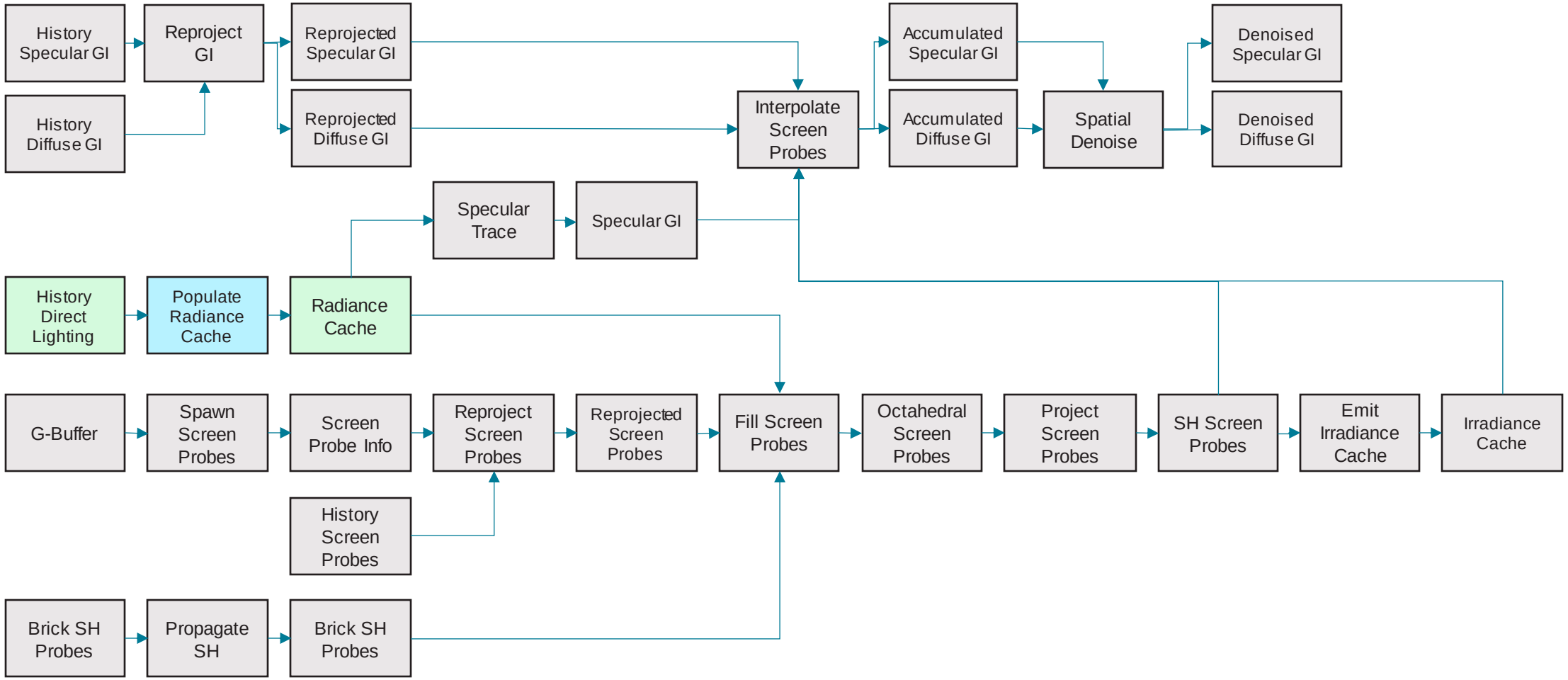
- Why use a world-space irradiance cache?
- The screen probes are more responsive but can be disoccluded with motion
- World-space irradiance cache is coarse but more stable
- Serves as a fallback
- Stored as 2nd Order Spherical Harmonics in a single large buffer



ALGORITHM

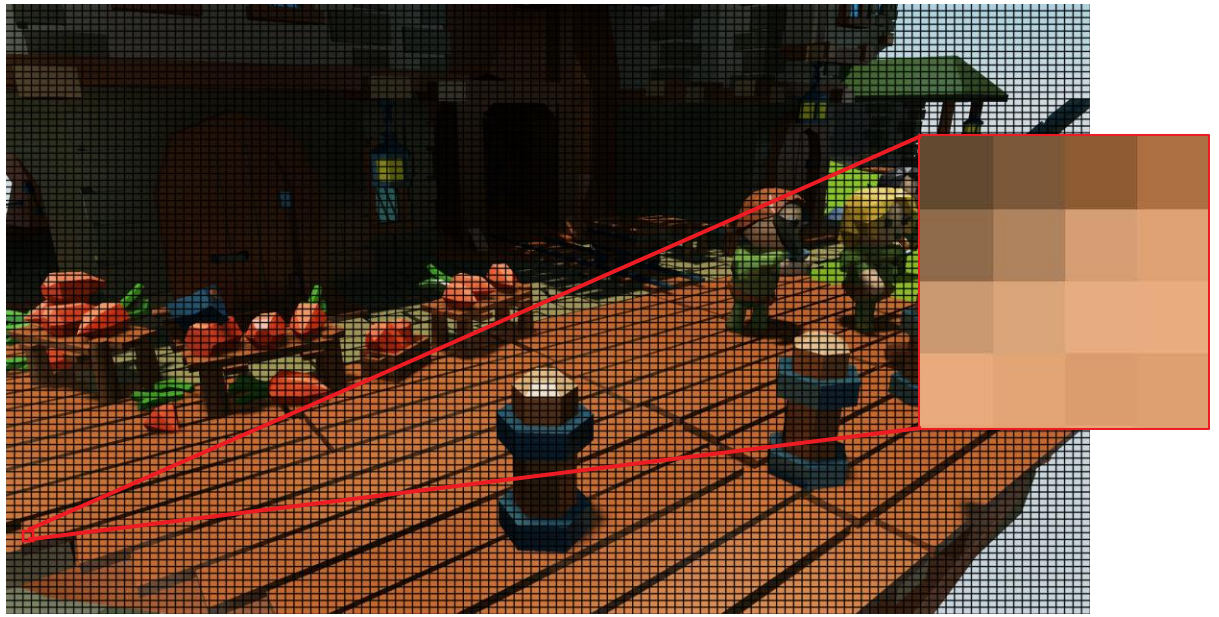
- Why use a radiance cache?
- Material data and UVs not available in the distance field
- Simplifies integration
  - Does not require light sources to be exposed to Brixelizer GI
  - Works regardless of shading and material model
- 256x256x256 R11G11B10 3D Texture atlas
- A radiance cache Brick is 4x4x4

ALGORITHM – EMIT PRIMARY RAY RADIANCE



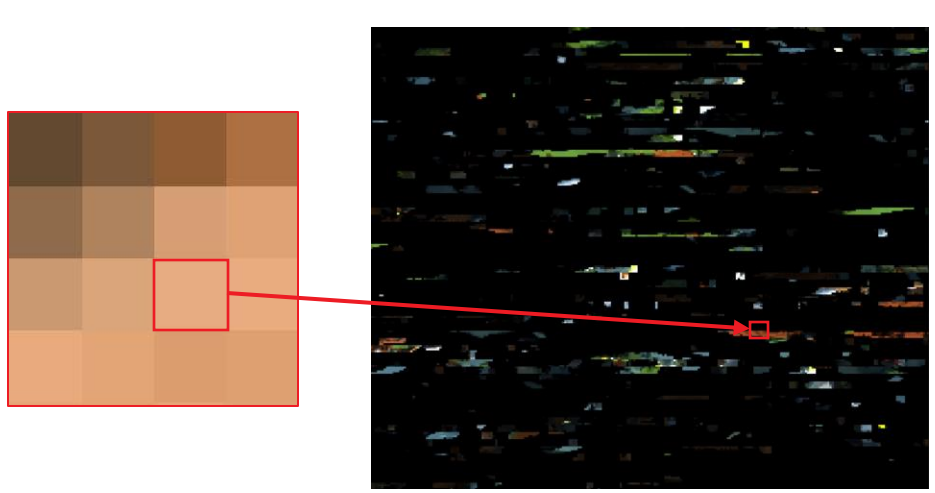
ALGORITHM – EMIT PRIMARY RAY RADIANCE

- Populate Radiance Cache
- Inject direct radiance using previous lit frame
- Done at quarter resolution
- Inject one value at each 4x4 tiles



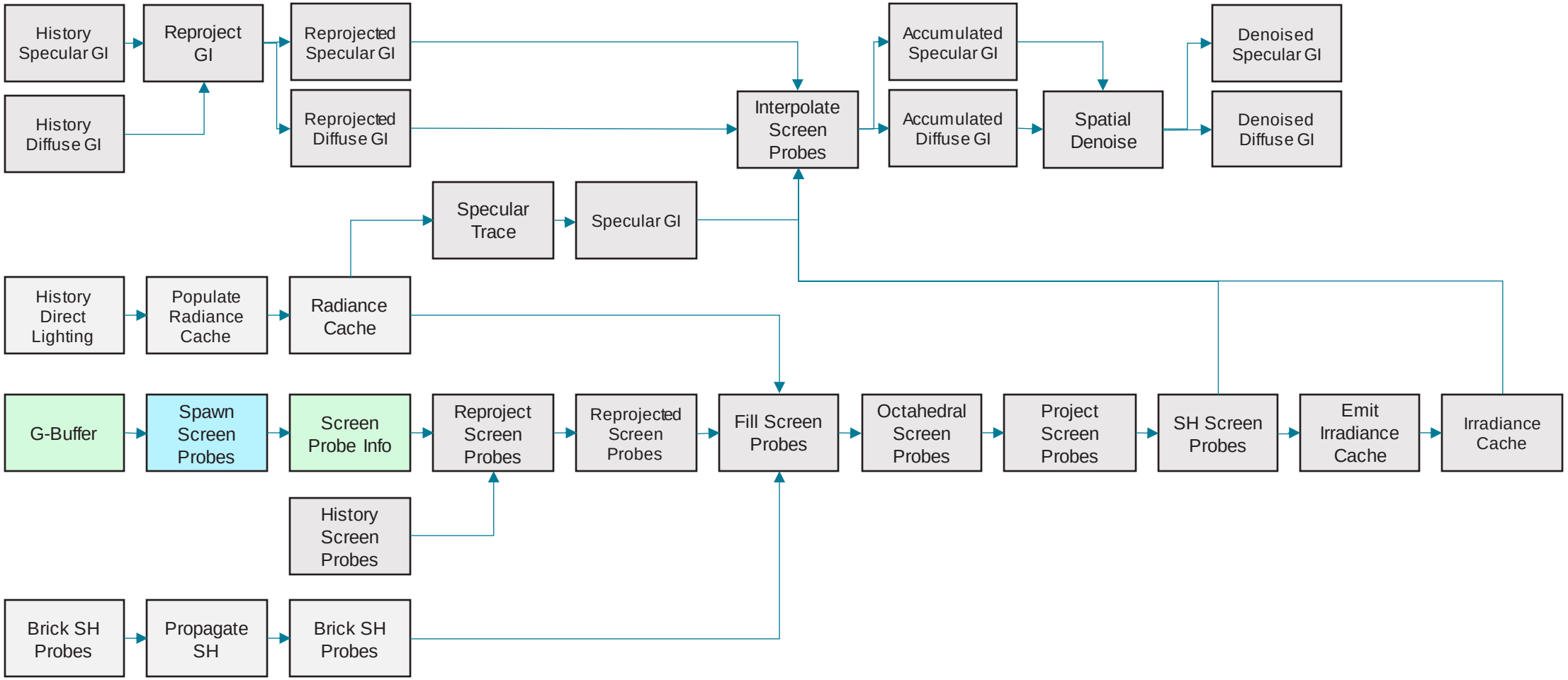
ALGORITHM – EMIT PRIMARY RAY RADIANCE

- Pick a random point within the tile and reproject direct lighting from the previous frame
- Reconstruct world space position and find associated Brick within each cascade
- Compute UVW coordinate within the Brick
- Accumulate the reprojected radiance into Radiance Cache at UVW coordinate



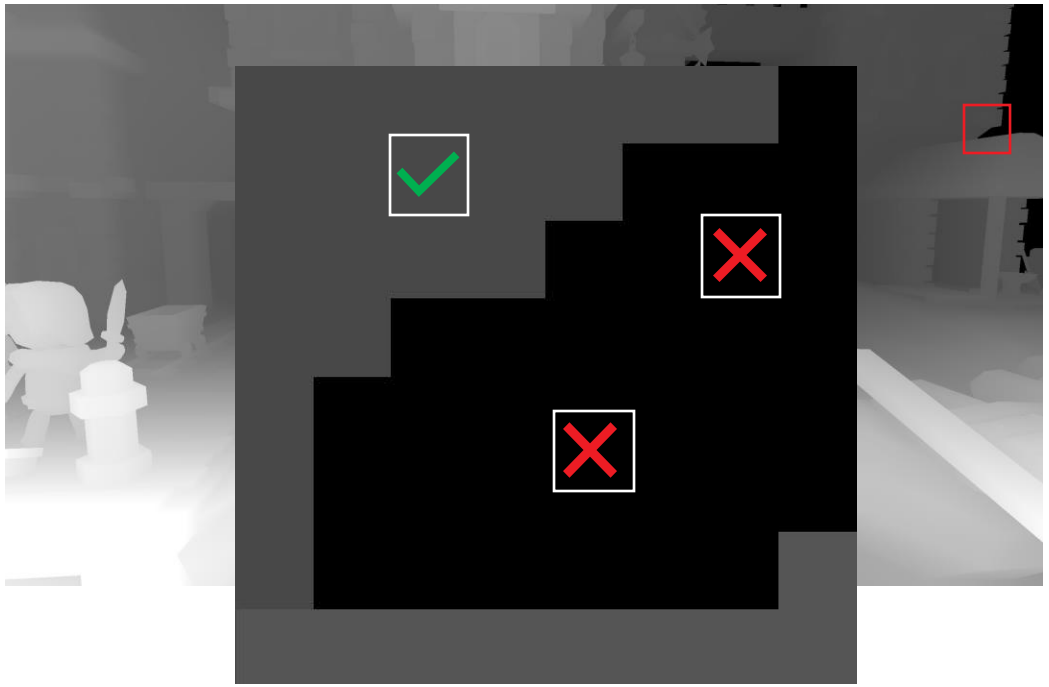
Visualization of the radiance cache

ALGORITHM – SPAWN SCREEN PROBES

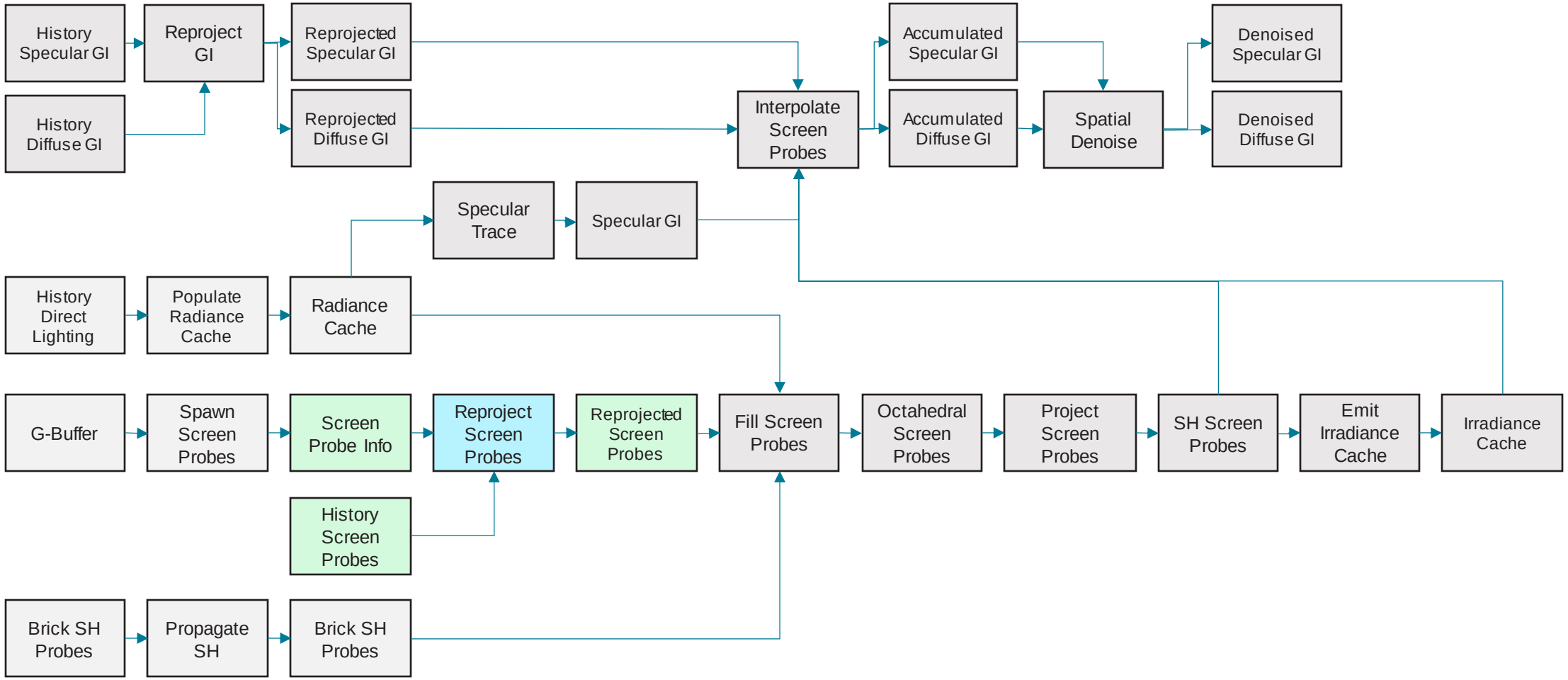


ALGORITHM – SPAWN SCREEN PROBES

- Screen Probes internally maintained in 8x8 Octahedral format
- Spawn a Screen Probe for each 8x8 tile of the frame
  - Maximum of 8 attempts
  - Jitter each attempt with a Hammersley sequence
  - Use jittered coordinate to sample depth buffer
  - Accept if depth value is valid
- Initialize an empty probe at the reconstructed world space position and store probe information
  - Depth
  - Normal
  - Random seed

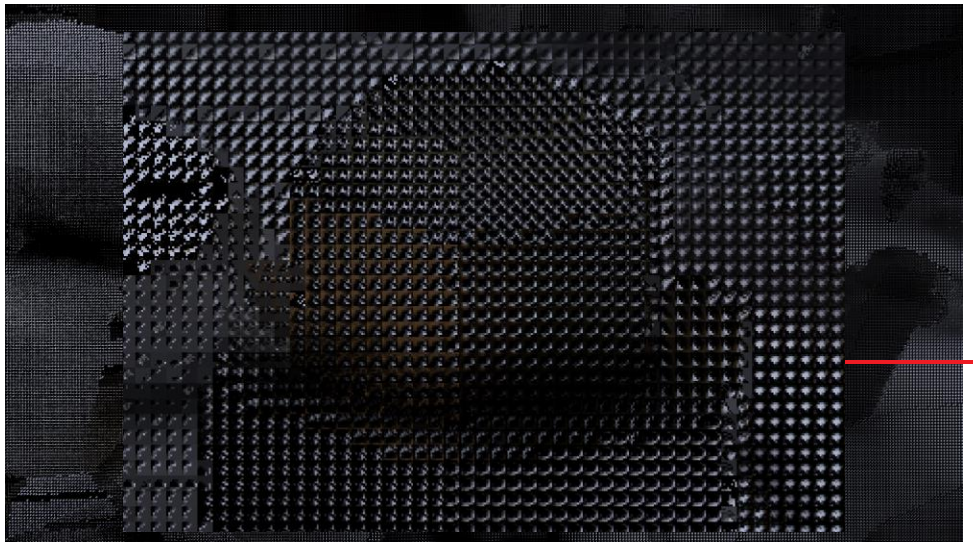


ALGORITHM – REPROJECT SCREEN PROBES

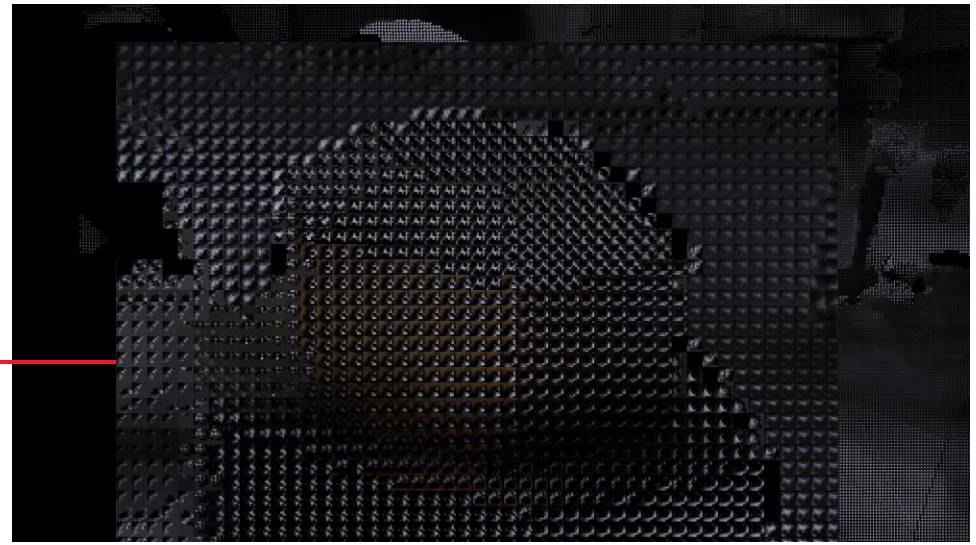


ALGORITHM – REPROJECT SCREEN PROBES

- Reproject the probes from last frame into the newly spawned probes
- Dispatch 8x8 threads per-screen probe
- Each thread handles a single pixel and attempts to find a reprojection candidate
- Store reprojection weight in LDS and pick probe with highest weight



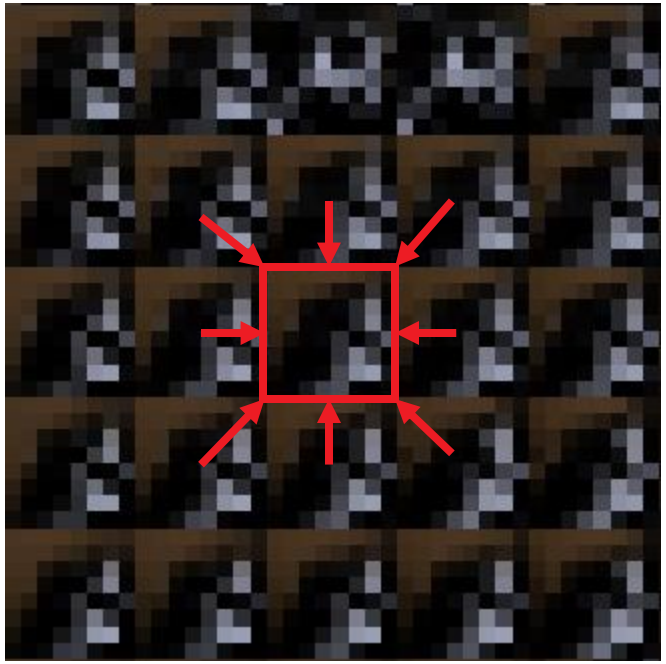
Frame N-1



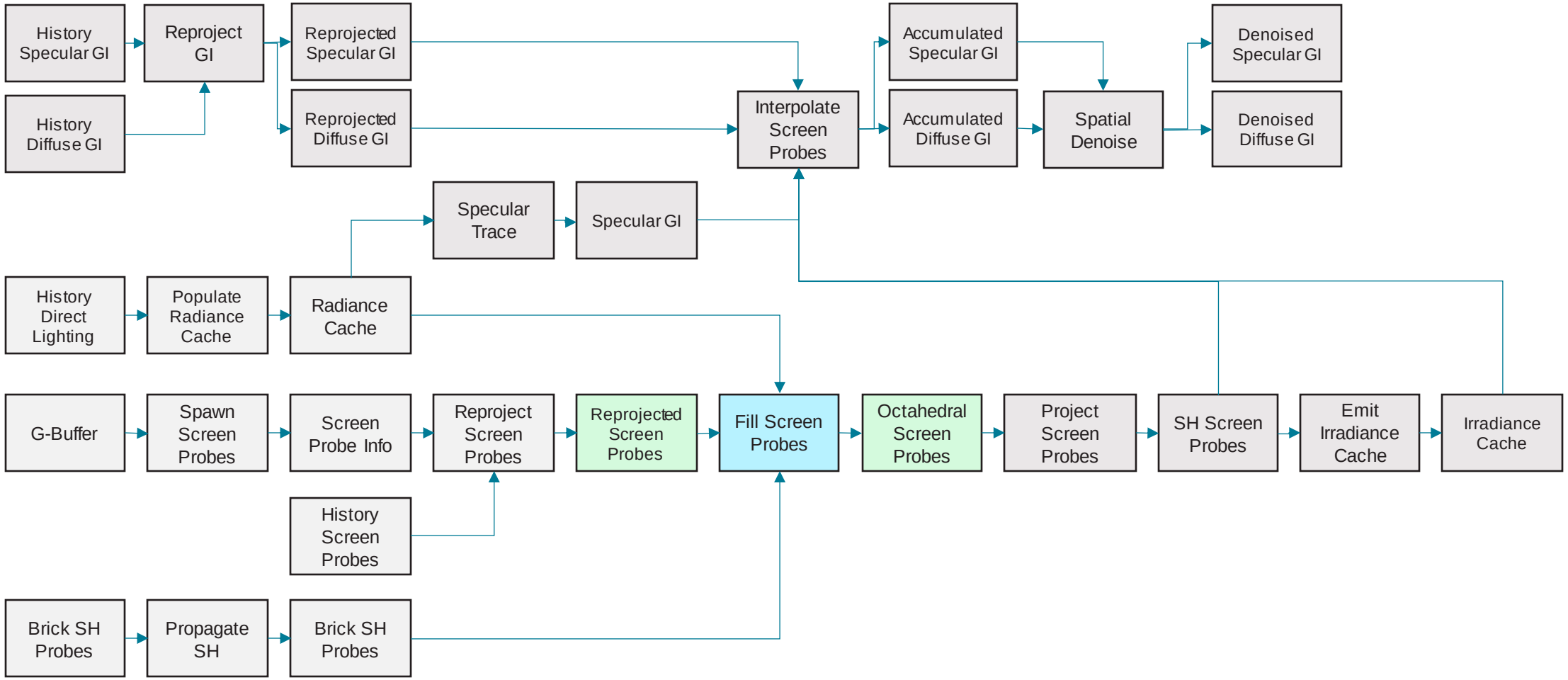
Frame N

ALGORITHM – REPROJECT SCREEN PROBES

- Accumulate irradiance from probes surrounding the reprojected probe
  - Use position and normal similarity to weigh contribution

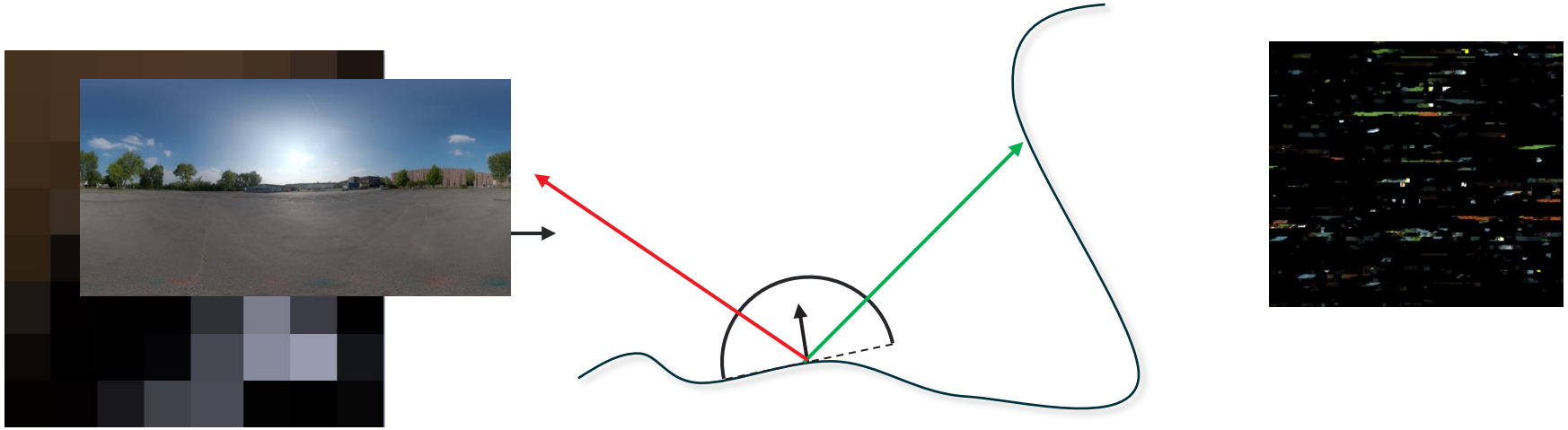


ALGORITHM – FILL SCREEN PROBES

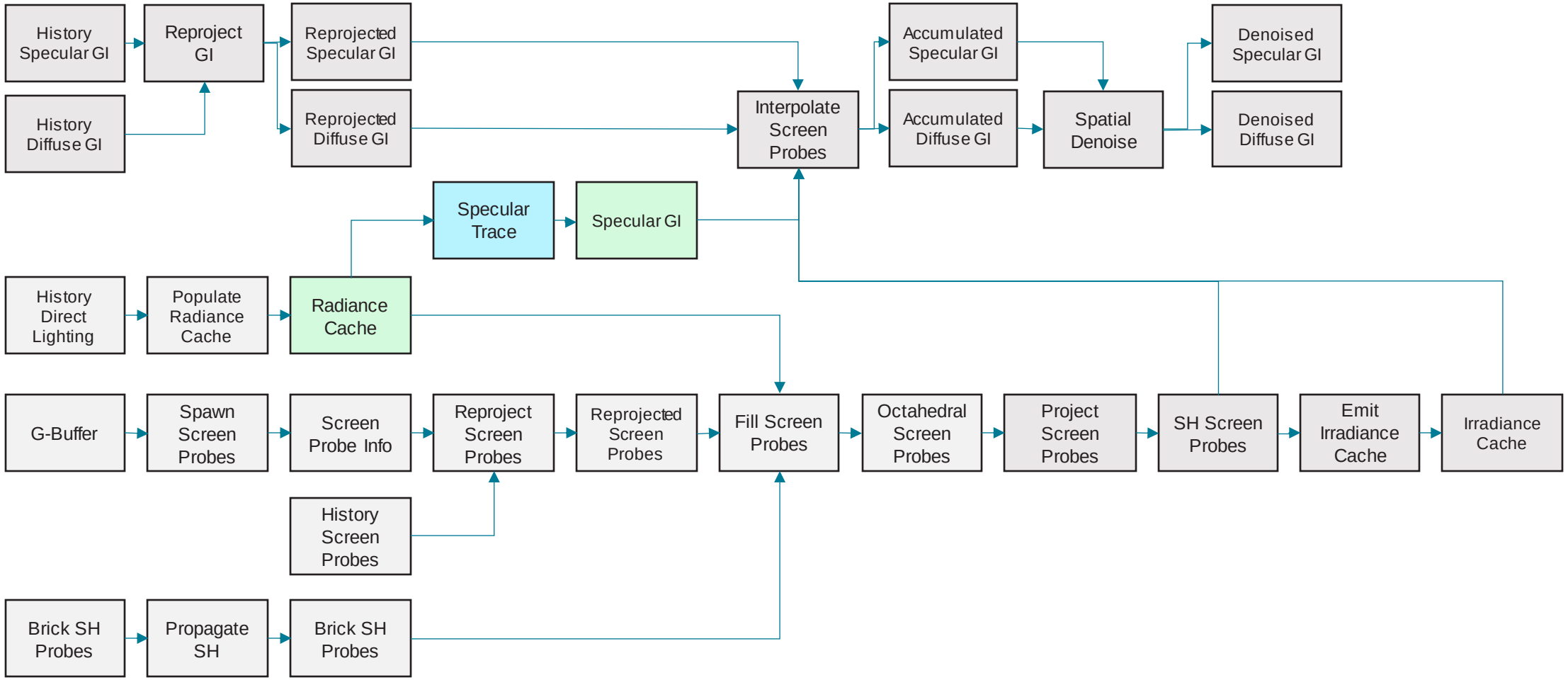


ALGORITHM – FILL SCREEN PROBES

- Inject radiance into screen probes that failed reuse
- For each probe pixel, uniformly sample the hemisphere and trace a ray
- If the ray hits,
  - Get Brick ID and UVW coordinate
  - Sample radiance cache
- If the ray misses,
  - Sample the environment map



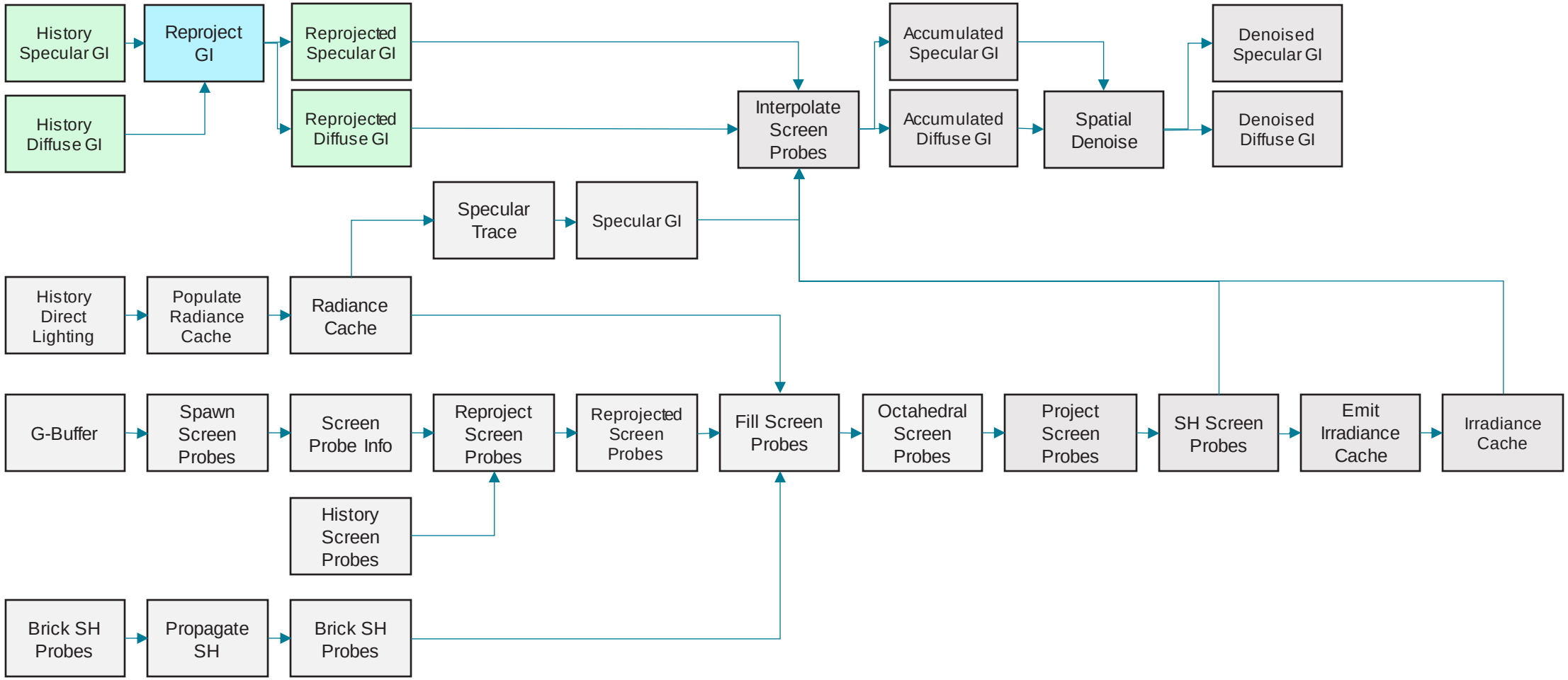
ALGORITHM – SPECULAR TRACE



ALGORITHM – SPECULAR TRACE

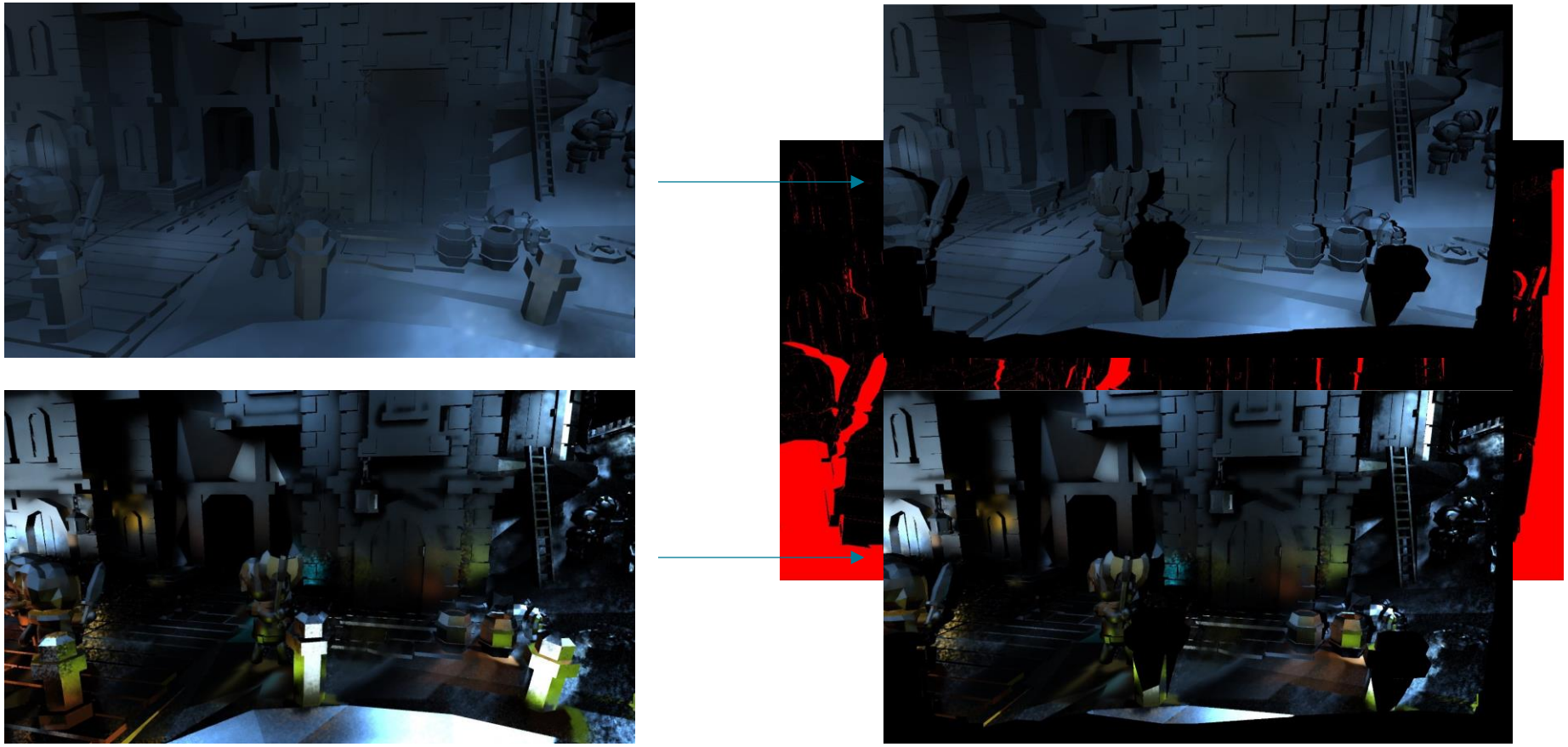
- Initially pre-trace Brixelizer rays at quarter resolution and store Brick IDs
  - Reduces number of traced rays
- Load Brick IDs in a full-resolution dispatch
- Find intersection point within Brick
  - Cheaper than a full trace as we can skip directly to the ray march within the Brick
- Sample Radiance Cache using Brick ID and UVW coordinate

ALGORITHM – REPROJECT GI

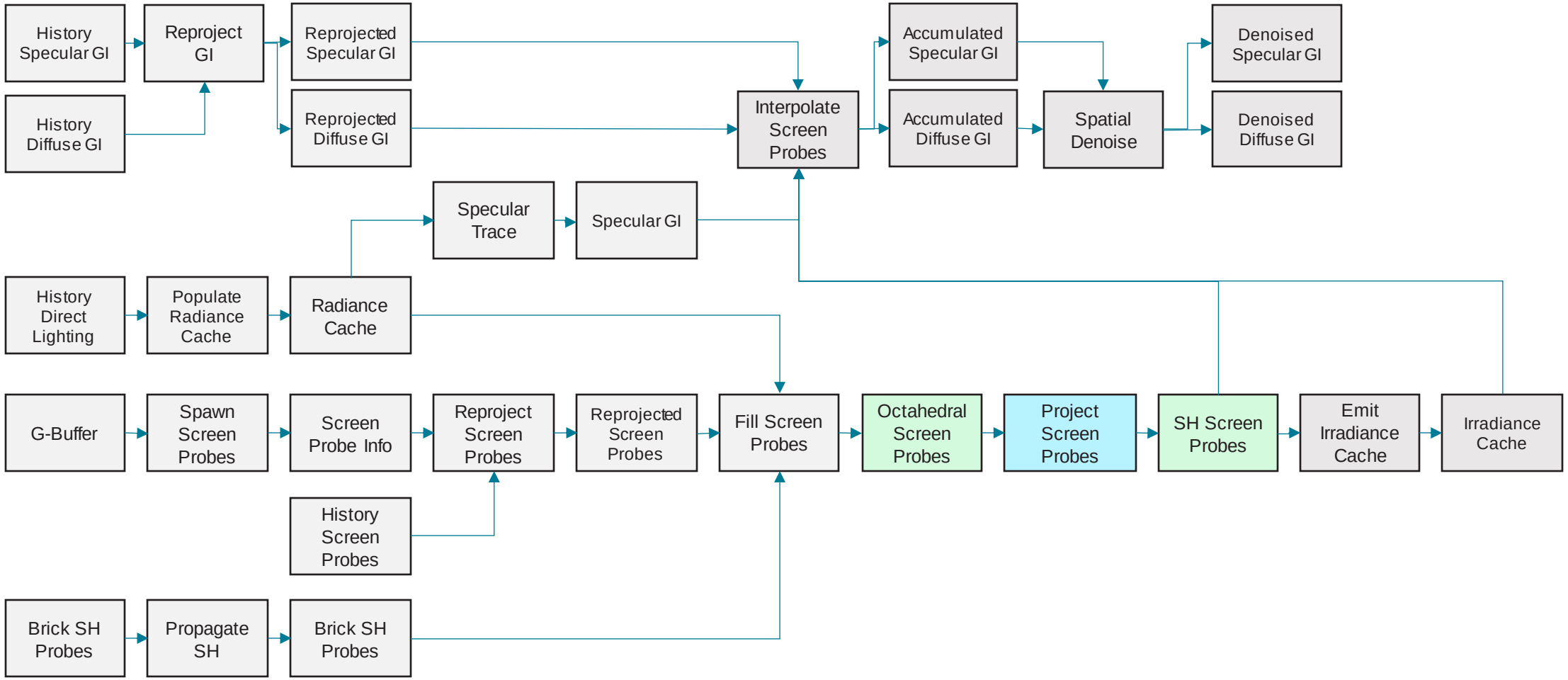


ALGORITHM – REPROJECT GI

- Reproject previous frames' Diffuse and Specular output
- Use a disocclusion mask to reject history

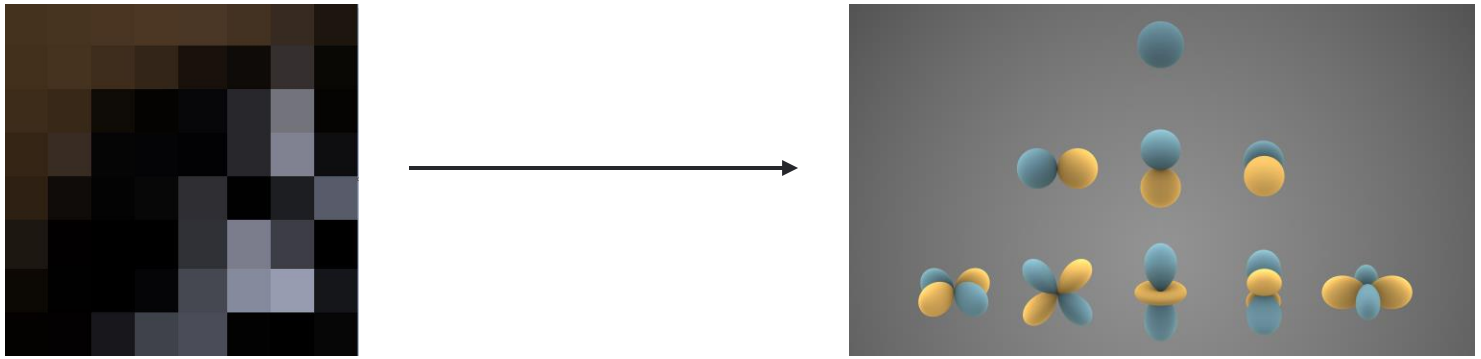


ALGORITHM – PROJECT SCREEN PROBES



ALGORITHM – PROJECT SCREEN PROBES

- Project octahedral screen probes into spherical harmonics
  - Used to feed the world space irradiance cache
- 8x8 thread group per screen probe
- Project onto SH2 and store in an intermediate buffer

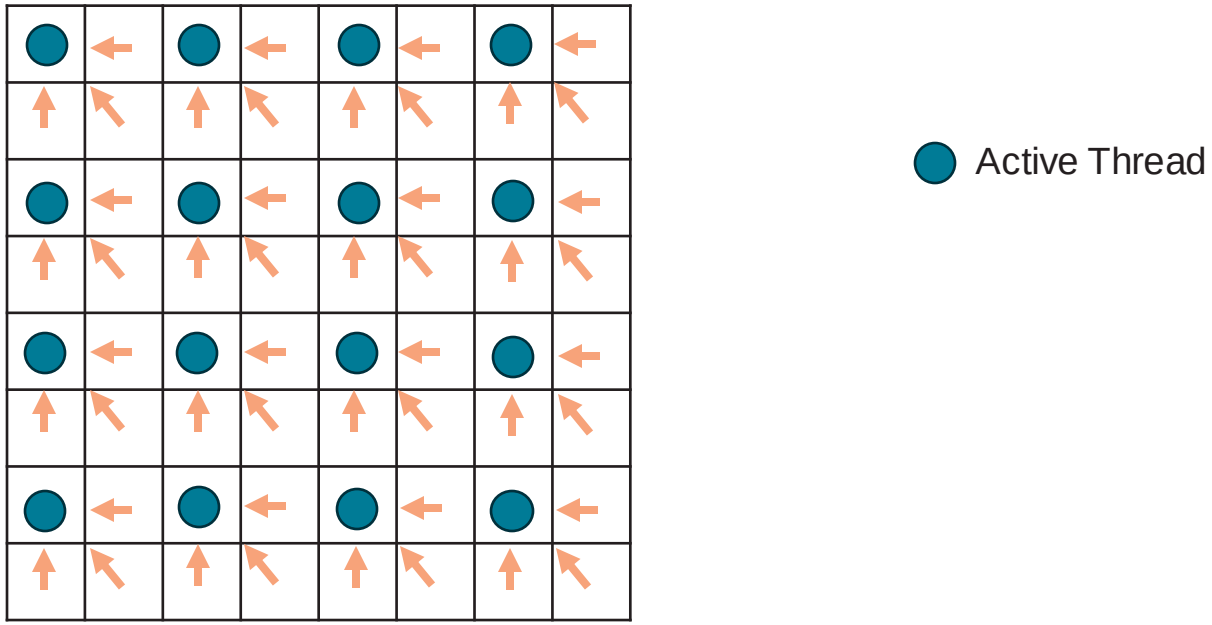


Screen Probe in Octahedral encoding

Screen Probe in SH2 Encoding

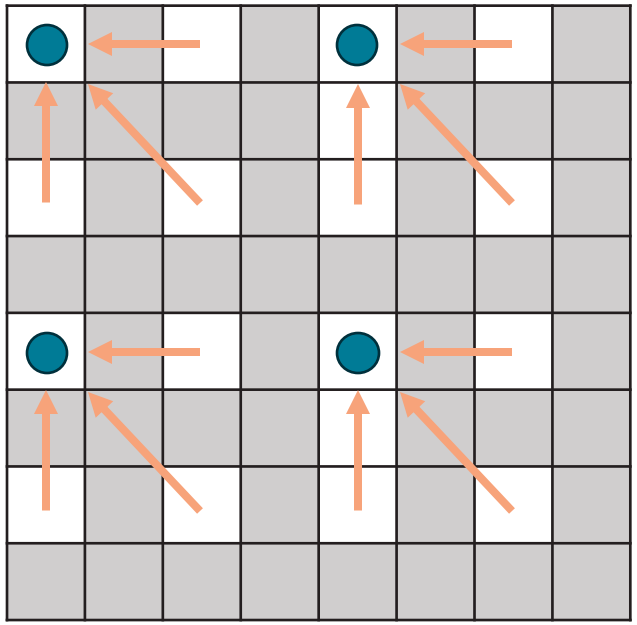
ALGORITHM – PROJECT SCREEN PROBES

- Each thread loads 1 of 64 radiance values and reconstructs ray direction
- Project each threads' radiance value onto SH coefficients and store into shared memory
- Perform parallel reduction in shared memory to combine all 64 SH radiance values



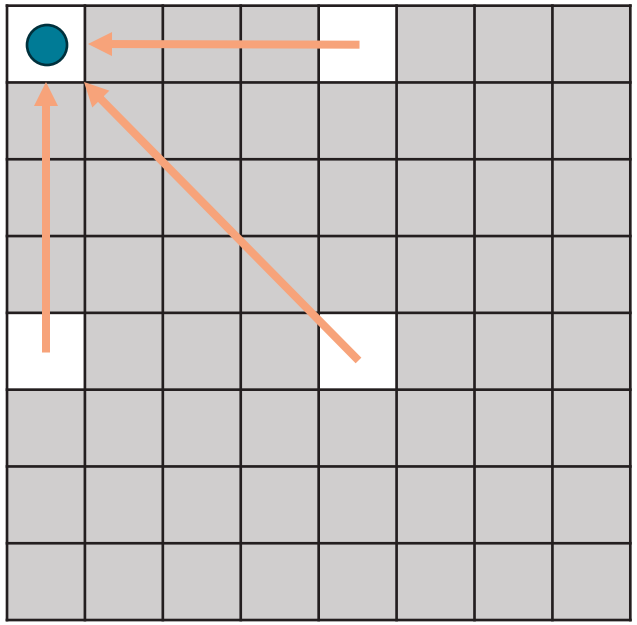
ALGORITHM – PROJECT SCREEN PROBES

- Each thread loads 1 of 64 radiance values and reconstructs ray direction
- Project each threads' radiance value onto SH coefficients and store into shared memory
- Perform parallel reduction in shared memory to combine all 64 SH radiance values

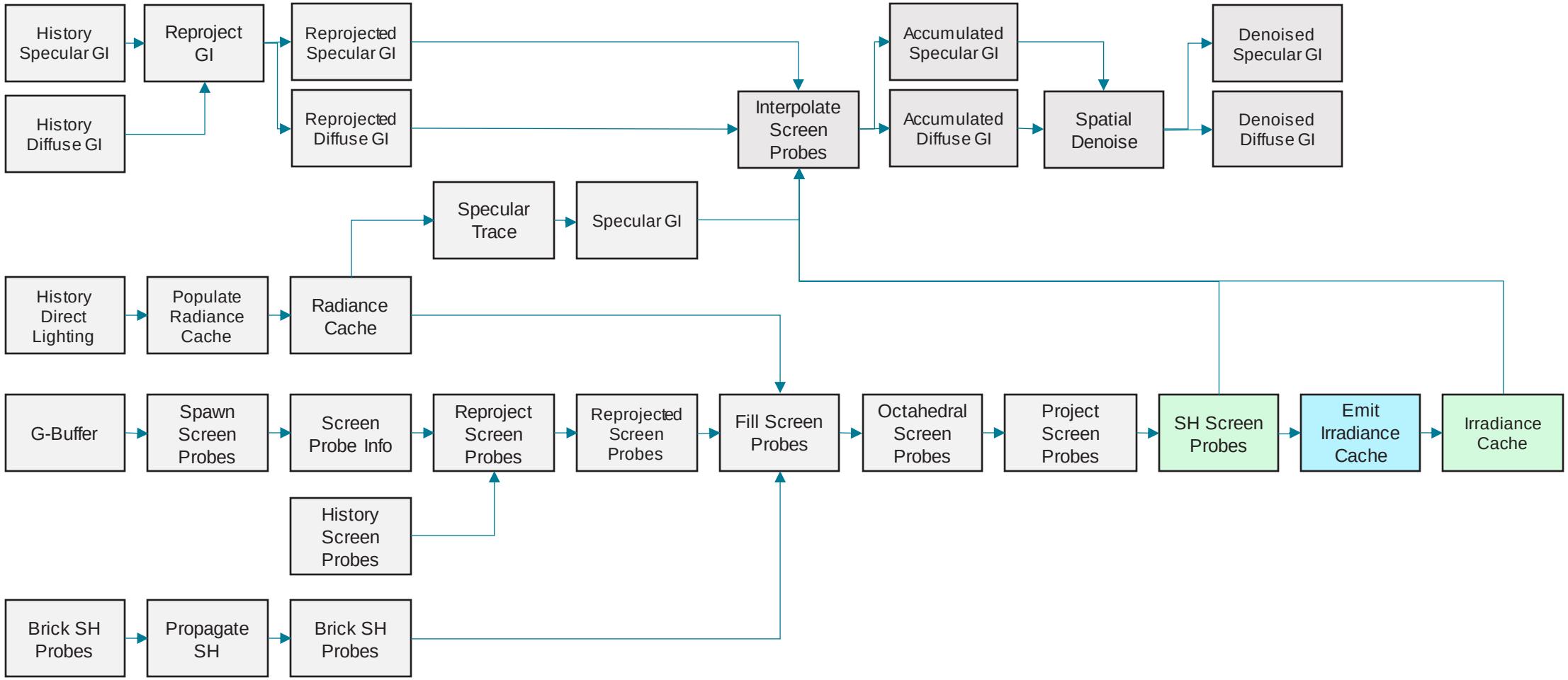


ALGORITHM – PROJECT SCREEN PROBES

- Each thread loads 1 of 64 radiance values and reconstructs ray direction
- Project each threads' radiance value onto SH coefficients and store into shared memory
- Perform parallel reduction in shared memory to combine all 64 SH radiance values

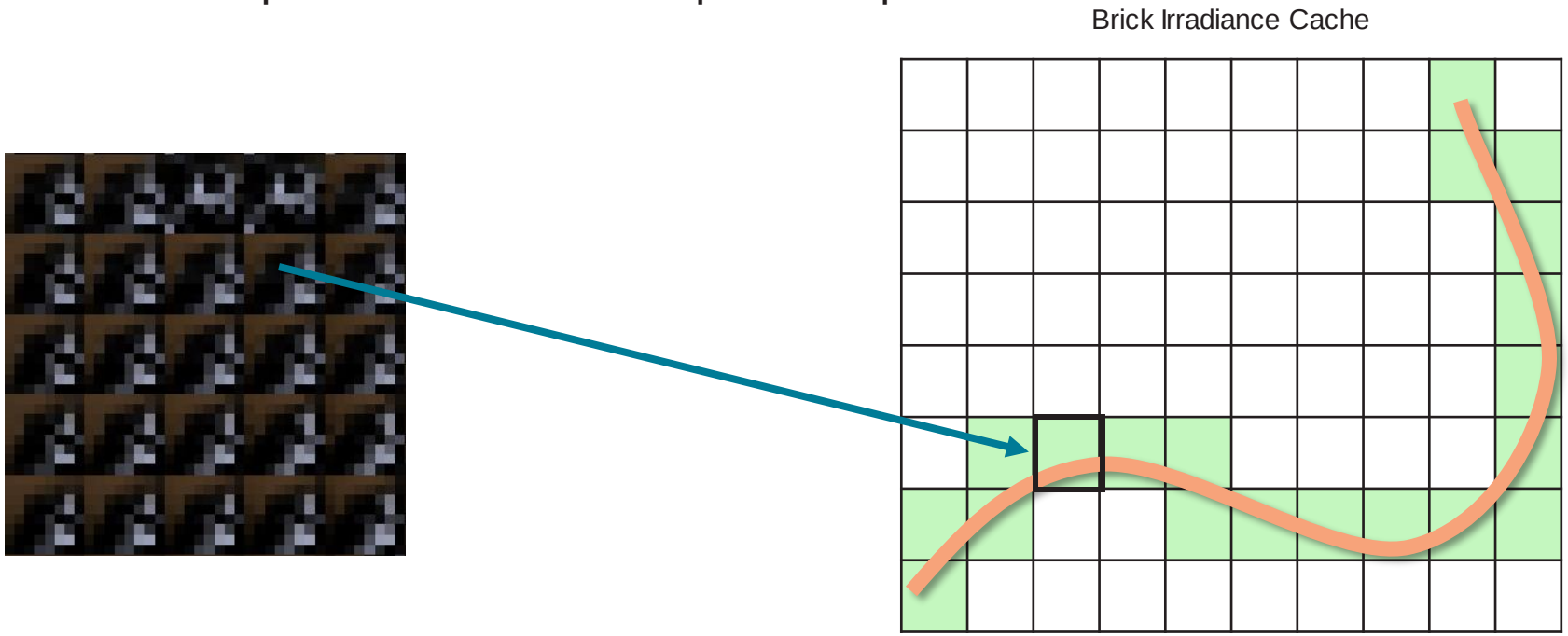


ALGORITHM – EMIT IRRADIANCE CACHE

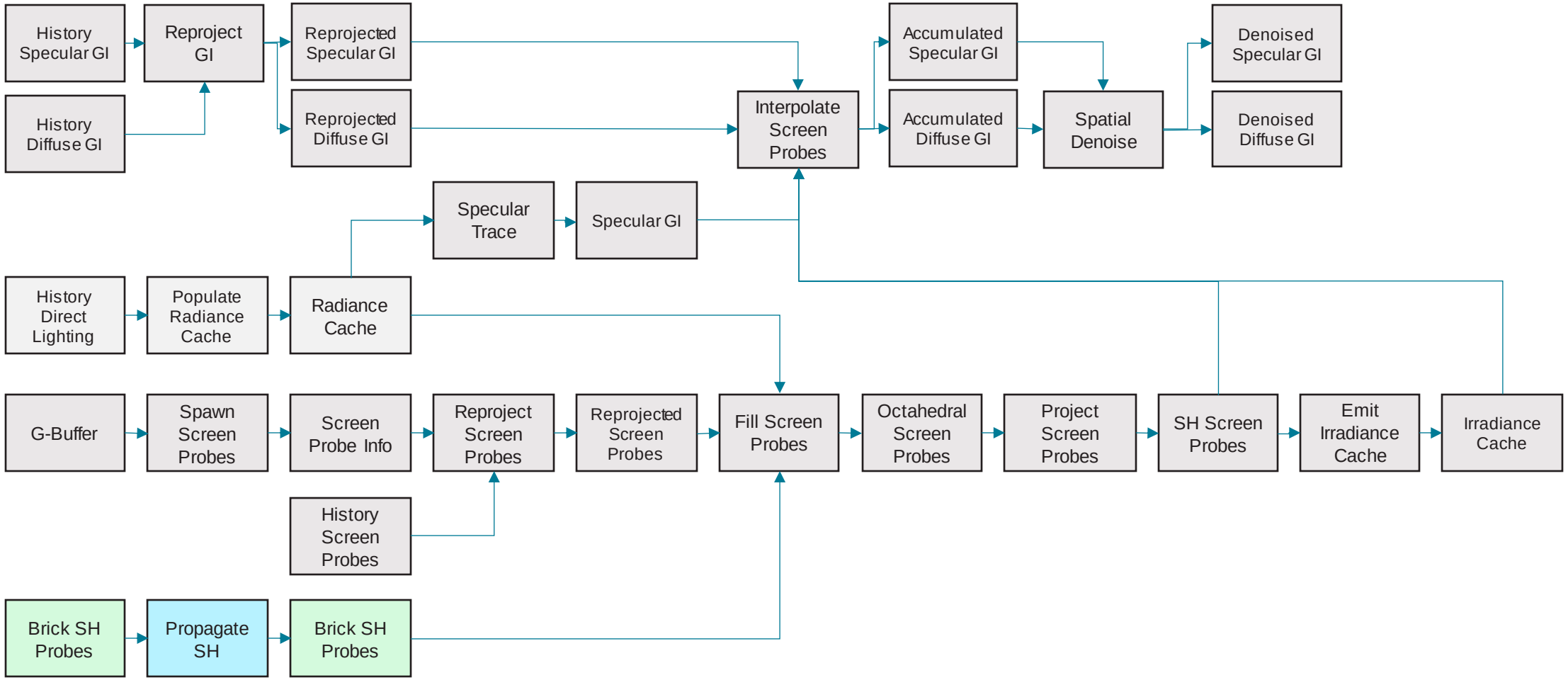


ALGORITHM – EMIT IRRADIANCE CACHE

- Feed world space irradiance cache using SH screen probes
- Each thread loads the SH values for a probe
- For each Brixelizer cascade,
  - Find which Brick the screen probe intersects
  - Blend the screen probe with the world space SH probe



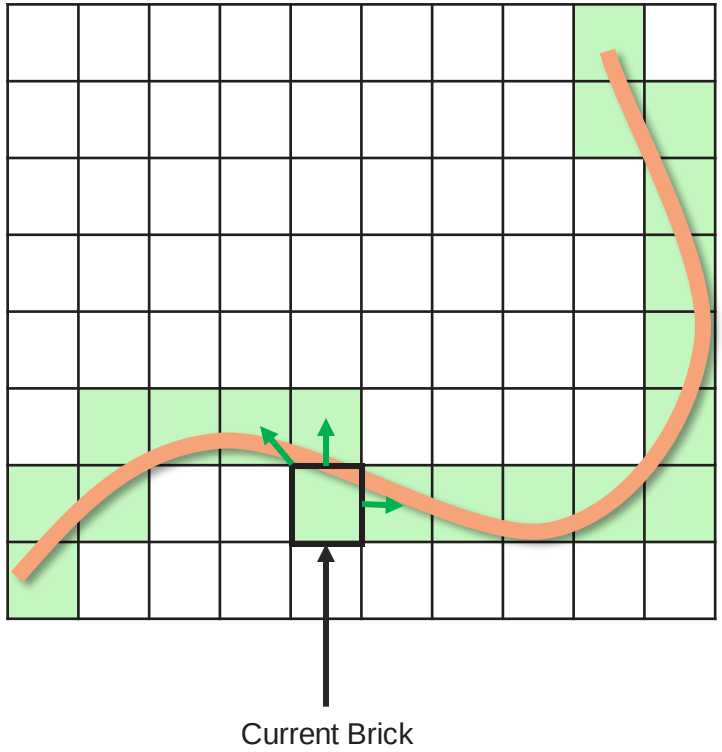
ALGORITHM – PROPAGATE SH



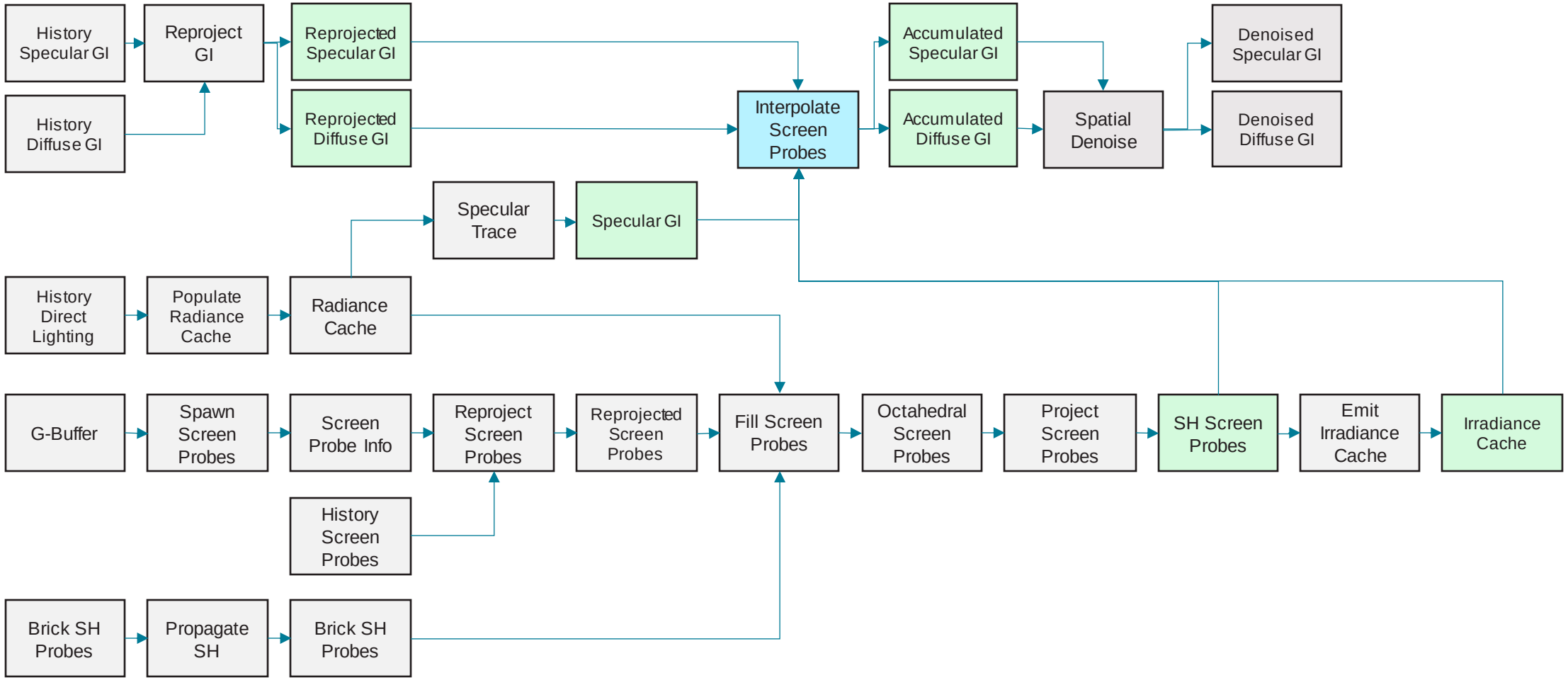
ALGORITHM – PROPAGATE SH

- Propagate Spherical Harmonics within each Brick of the World Space Irradiance Cache into neighboring Bricks

- Time sliced update
  - One cascade per-frame
  - Most detailed cascades updated more often

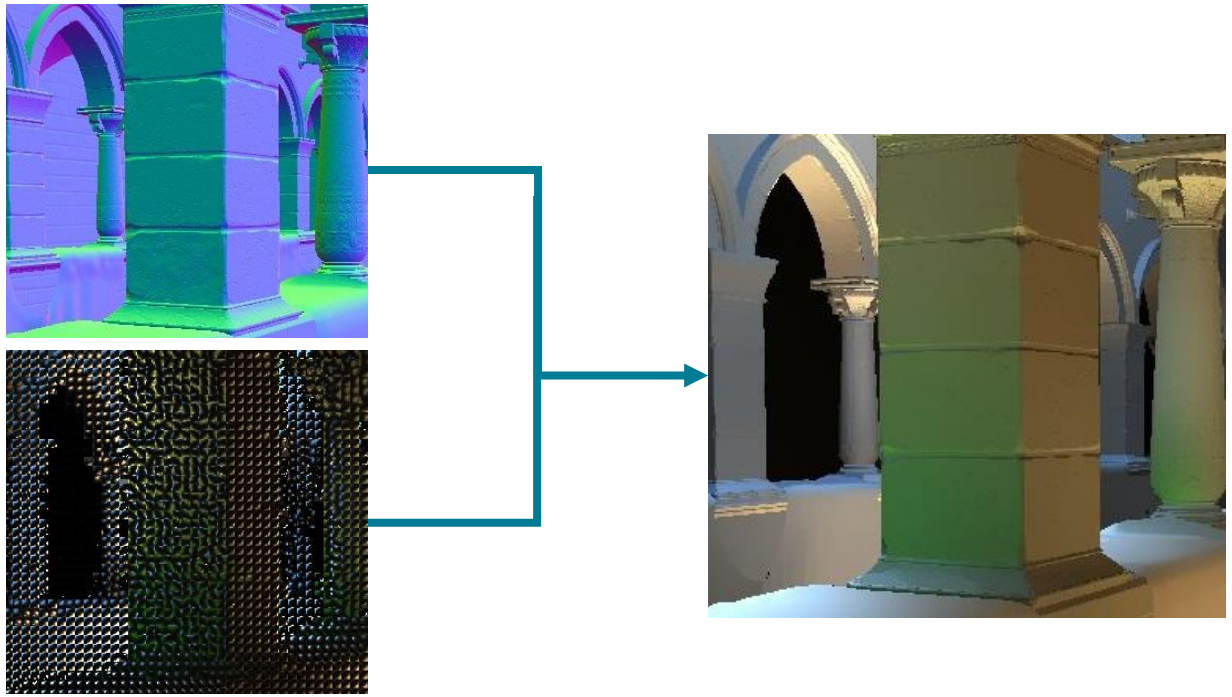


ALGORITHM – INTERPOLATE SCREEN PROBES



ALGORITHM – INTERPOLATE SCREEN PROBES

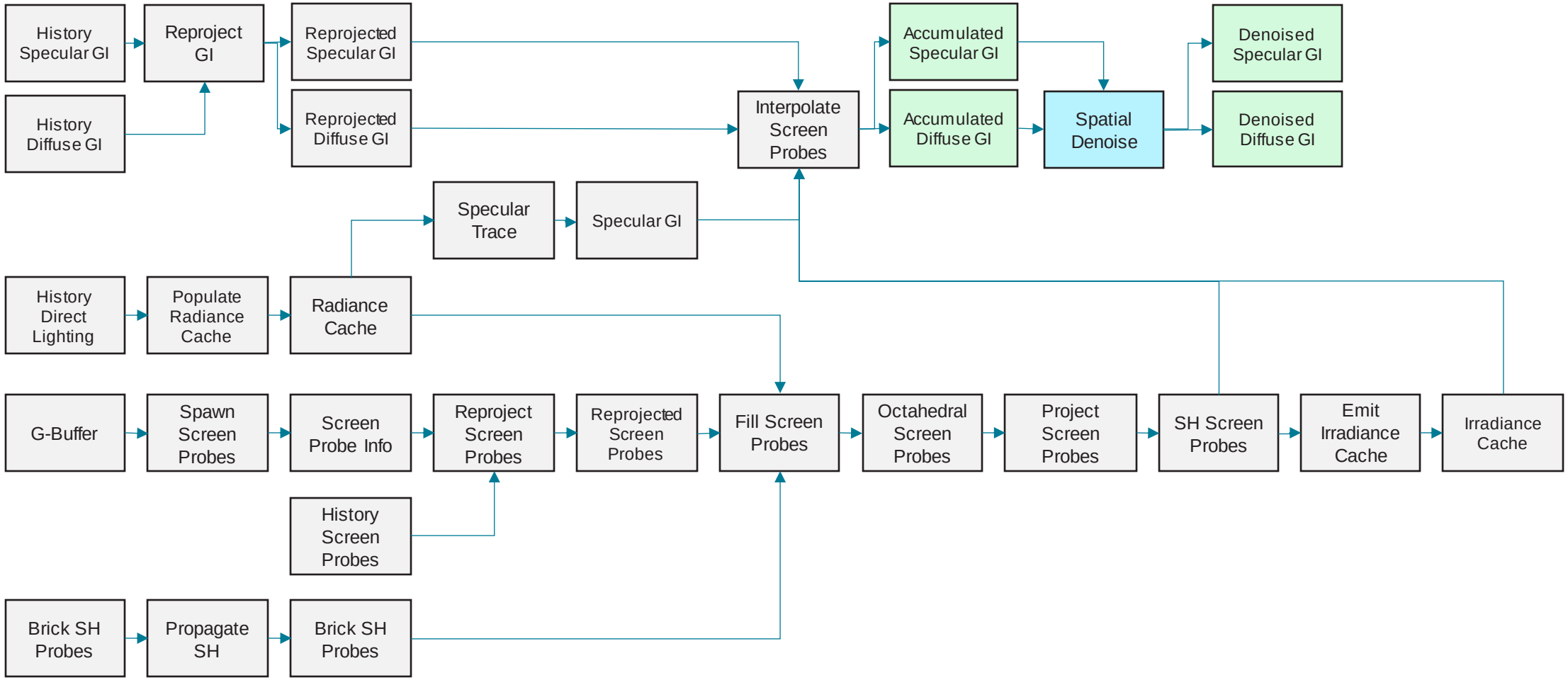
- Reconstruct screen space irradiance
  - Projecting nearest 2x2 SH screen probes to the G-Buffer Normal
- Reconstruct world space irradiance
  - Interpolate SH values from surrounding 8 probes
- Blend both values together for final irradiance



ALGORITHM – INTERPOLATE SCREEN PROBES

- Blend current frame Diffuse GI and Specular GI output with reprojected outputs

ALGORITHM – SPATIAL DENOISE



ALGORITHM – SPATIAL DENOISE

- A simple bilateral blur to further denoise the Diffuse and Specular GI outputs
- Radius determined by the number of samples
  - Less samples = Larger radius

ALGORITHM – CLEAR CACHE

- SDF cascades implemented as clip maps
- Bricks get invalidated with camera movement
- Clear Cache entries associated with invalidated Bricks
- Uses internal Brixelizer data structures
- Indirect dispatch to reset Direct Radiance Cache and Irradiance Cache

GROUND TRUTH COMPARISONS

Brixelizer GI

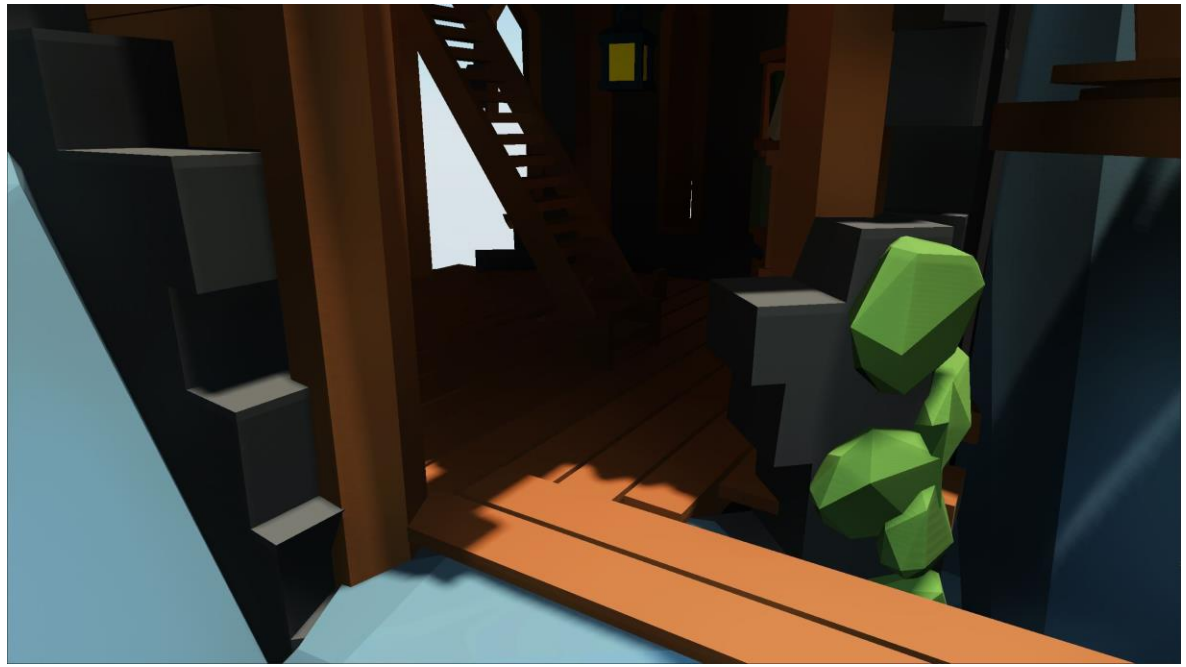
Path Tracer

GROUND TRUTH COMPARISONS

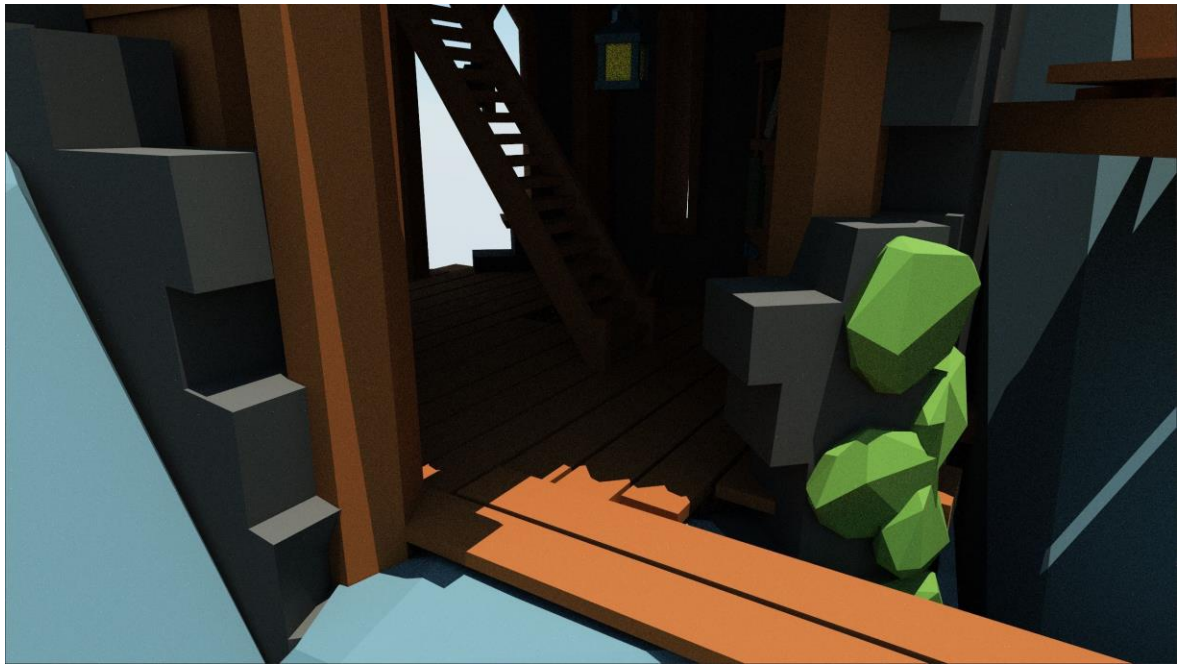
Brixelizer GI

Path Tracer

GROUND TRUTH COMPARISONS

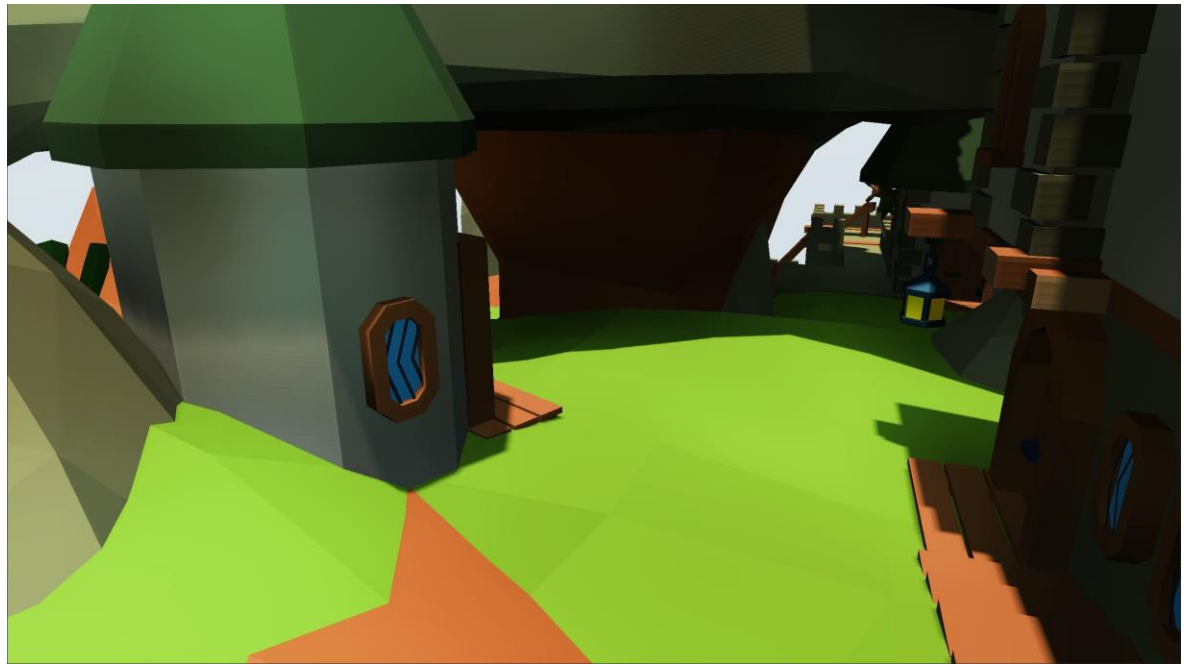


Brixelizer GI

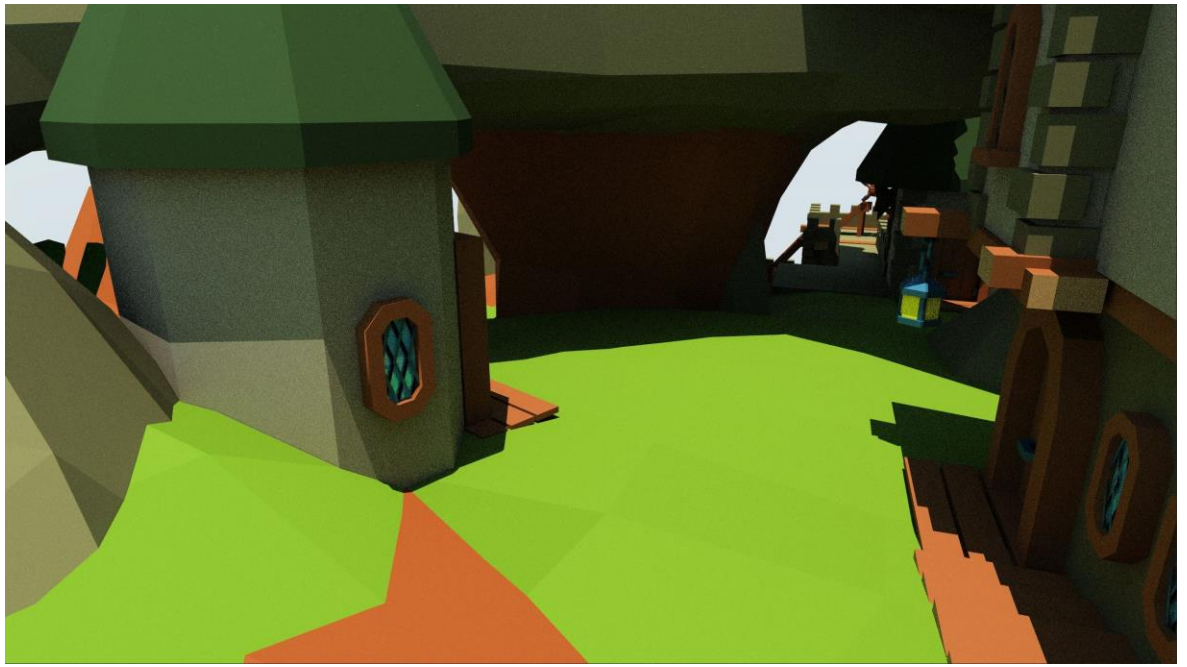


Path Tracer

GROUND TRUTH COMPARISONS



Brixelizer GI



Path Tracer

PERFORMANCE

- Passes that update Screen Probes are resolution-dependent
- Passes that update the Radiance and Irradiance Cache are scene-dependent
- Default quality setting outputs at half-resolution internally (0.5x scale)
  - Optional 0.75x scale and native-resolution quality settings

- AMD Radeon RX 7900 XTX: ~2ms (4K)
- NVIDIA GeForce RTX 4080: ~2.2ms (4K)
- AMD Radeon RX 7600 XT: ~1.7ms (1080p)

*Tested on AMD Software: Adrenalin Edition 24.1.1 and NVIDIA GeForce Game Ready Driver 551.61. See disclaimer for full system specification.

MEMORY FOOTPRINT

- Brixelizer
  - Internal Buffers – 16MB
  - SDF Texture Atlas – 128MB
  - Cascade Buffers – 1MB (x Number of Cascades)
  - Scratch Buffer – Scene Dependent

- Brixelizer GI

Resolution	0.5x Scale	0.75x Scale	1x Scale
1920x1080	~110MB	~150MB	~200MB
2560x1440	~130MB	~200MB	~270MB
3840x2160	~200MB	~320MB	~500MB

LIMITATIONS

- Radiance cache filled using screen space information
  - Surfaces must be in view before bounced lighting can take on its colour
  - Changes to lighting/geometry take effect only when they are in view
  - Can be mitigated by using other views to inject radiance (dynamic probes, secondary views etc)
- Loses small-scale details
  - Can be recovered using SSAO or SSGI

INTEGRATION

- Requires an existing Brixelizer integration
  - AMD FidelityFX backend for your API of choice
  - Brixelizer context
- Link against the Brixelizer GI library and include ffx_brixelizer_gi.h
- Create Brixelizer GI context

```
FfxBrixelizerGICreateContextDesc desc = {};

desc.flags            = FFX_BRIXELIZER_GI_ENABLE_DEPTH_INVERTED;
desc.quality          = FFX_BRIXELIZER_GI_QUALITY_DEFAULT;
desc.outputSize       = {ouputWidth, outputHeight};
desc.backendInterface = m_BackendInterface;

FfxErrorCode errorCode = ffxBrixelizerGICreateContext(m_pBrixelizerGIContext, &desc);
assert(errorCode == FFX_OK);
```

INTEGRATION

- Next dispatch the Brixelizer GI compute workloads
- Fill out dispatch description
- Assign Brixelizer context and output resources

```
FfxBrixelizerGIDispatchDescription desc = {};

desc.brixelizerContext = m_pBrixelizerContext;
desc.sdfAtlas          = ffxGetResourceDX12(sdfAtlas, "SDF Atlas", FFX_RESOURCE_STATE_COMPUTE_READ);
desc.bricksAABBs       = ffxGetResourceDX12(brickAABBs, "Brick AABBs", FFX_RESOURCE_STATE_COMPUTE_READ);

for (uint32_t i = 0; i < numCascades; ++i)
{
    desc.cascadeAABBTrees[i] = ffxGetResourceDX12(cascadeAABBTrees[i], "Cascade AABB Tree", FFX_RESOURCE_STATE_COMPUTE_READ);
    desc.cascadeBrickMaps[i] = ffxGetResourceDX12(cascadeBrickMaps[i], "Cascade Brick Map", FFX_RESOURCE_STATE_COMPUTE_READ);
}
```

INTEGRATION

- Assign G-Buffer resources and environment map
- Assign Diffuse and Specular GI output resources

```
desc.prevLitOutput    = ffxGetResourceDX12(m_pHistoryLitOutput->GetResource(), L"PrevLitOutput");
desc.depth            = ffxGetResourceDX12(m_pDepthBuffer->GetResource(), L"Depth");
desc.historyDepth     = ffxGetResourceDX12(m_pHistoryDepth->GetResource(), L"HistoryDepth");
desc.normal           = ffxGetResourceDX12(m_pNormalTarget->GetResource(), L"Normal");
desc.historyNormal    = ffxGetResourceDX12(m_pHistoryNormals->GetResource(), L"HistoryNormal");
desc.roughness        = ffxGetResourceDX12(m_pRoughnessTarget->GetResource(), L"Roughness");
desc.motionVectors    = ffxGetResourceDX12(m_pVelocityBuffer->GetResource(), L"MotionVectors");
desc.environmentMap   = ffxGetResourceDX12(m_pEnivornmentMap->GetResource(), L"EnvironmentMap");

desc.outputDiffuseGI  = ffxGetResourceDX12(m_pDiffuseGI->GetResource(), L"OutputDiffuseGI");
desc.outputSpecularGI = ffxGetResourceDX12(m_pSpecularGI->GetResource(), L"OutputSpecularGI");
```

INTEGRATION

- Assign camera matrices and Brixelizer tracing constants
- Dispatch compute workloads

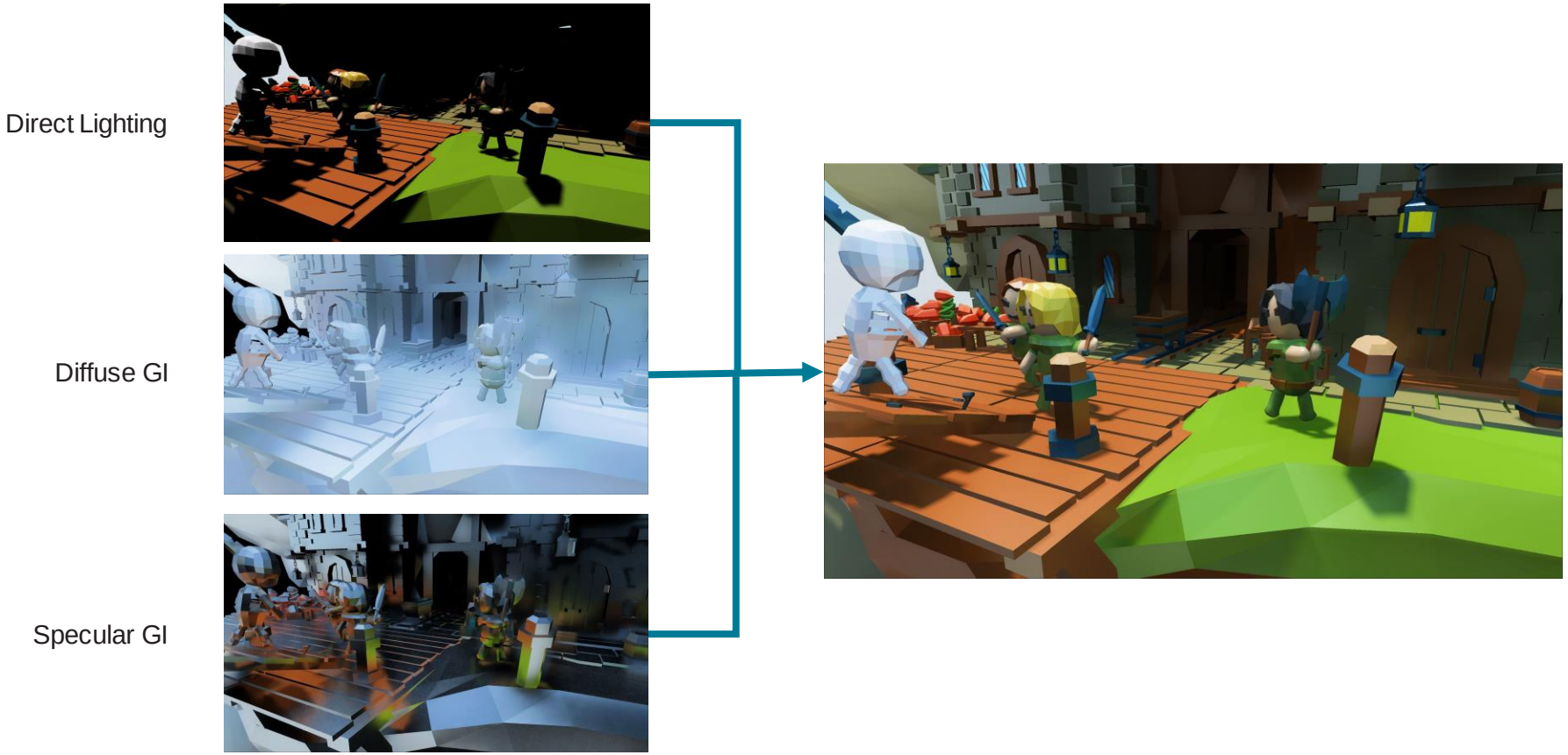
```
desc.view           = m_camera->getView();
desc.projection     = m_camera->getProjection();
desc.prevView       = m_camera->getPreviousView();
desc.prevProjection = m_camera->getPreviousProjection();

desc.startCascade = m_startCascadeIdx;
desc.endCascade   = m_endCascadeIdx;
desc.rayPushoff   = m_rayPushOff;
desc.sdfSolveEps  = m_sdfSolveEps;
desc.tMin         = m_TMin;
desc.tMax         = m_TMax;

ffxBrixelizerGIDispatch(m_pBrixelizerGIContext, &desc, ffxGetCommandList(pCmdList));
```

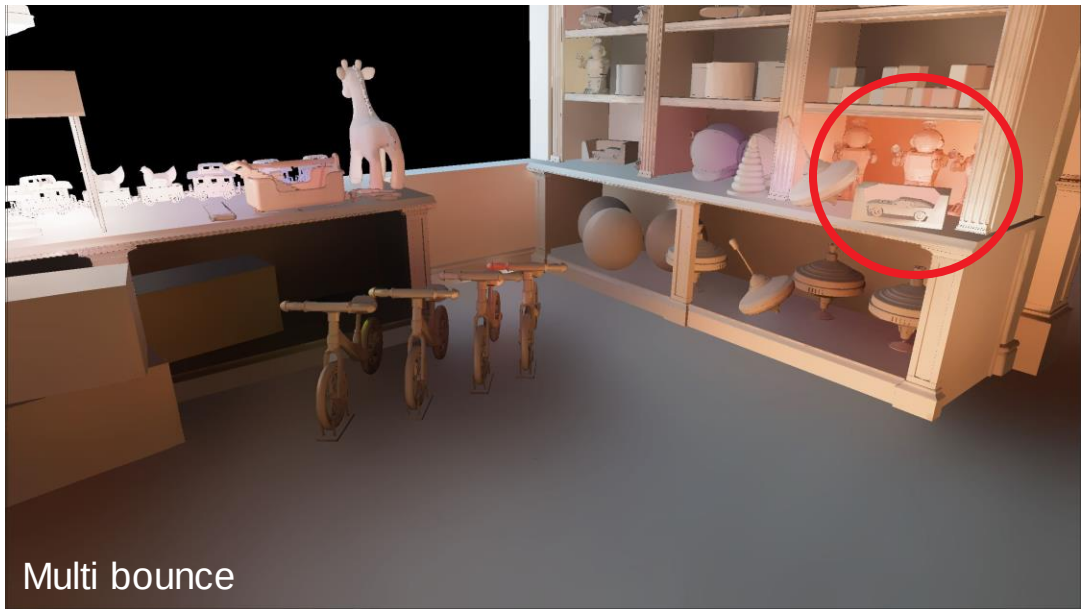
INTEGRATION

- Composite Diffuse and Specular outputs with direct lighting
- Drop-in replacement for Image Based Lighting



INTEGRATION

- For multiple diffuse bounces use the composited output from the previous frame as input to Brixelizer GI
- This updates the radiance cache with bounced lighting



INTEGRATION – DEBUG VISUALIZATION

- Optional debug visualization pass to inspect radiance and irradiance cache

INTEGRATION – DEBUG VISUALIZATION

- Assign Brixelizer context, Depth and Normal buffers, camera matrices and output buffer
- Directly outputs chosen debug visualization into the output resource

```
FfxBrixelizerGIDebugDescription desc = {};

desc.brixelizerContext = m_pBrixelizerContext;
desc.sdfAtlas          = ffxGetResourceDX12(sdfAtlas, "SDF Atlas", FFX_RESOURCE_STATE_COMPUTE_READ);
desc.bricksAABBs       = ffxGetResourceDX12(brickAABBs, "Brick AABBs", FFX_RESOURCE_STATE_COMPUTE_READ);

for (uint32_t i = 0; i < numCascades; ++i)
{
    desc.cascadeAABBTrees[i] = ffxGetResourceDX12(cascadeAABBTrees[i], "Cascade AABB Tree", FFX_RESOURCE_STATE_COMPUTE_READ);
    desc.cascadeBrickMaps[i] = ffxGetResourceDX12(cascadeBrickMaps[i], "Cascade Brick Map", FFX_RESOURCE_STATE_COMPUTE_READ);
}

desc.depth  = ffxGetResourceDX12(m_pDepthBuffer->GetResource(), L"Depth");
desc.normal = ffxGetResourceDX12(m_pNormalTarget->GetResource(), L"Normal");

desc.view       = m_camera->getView();
desc.projection = m_camera->getProjection();

desc.debugMode = FFX_BRIXELIZER_GI_DEBUG_MODE_RADIANCE_CACHE;
desc.output    = ffxGetResourceDX12(m_pGIDebug->GetResource(), L"OutputDebugGI");

ffxBrixelizerGIDebugVisualization(m_pBrixelizerGIContext, &desc, ffxGetCommandList(pCmdList));
```

DEMO

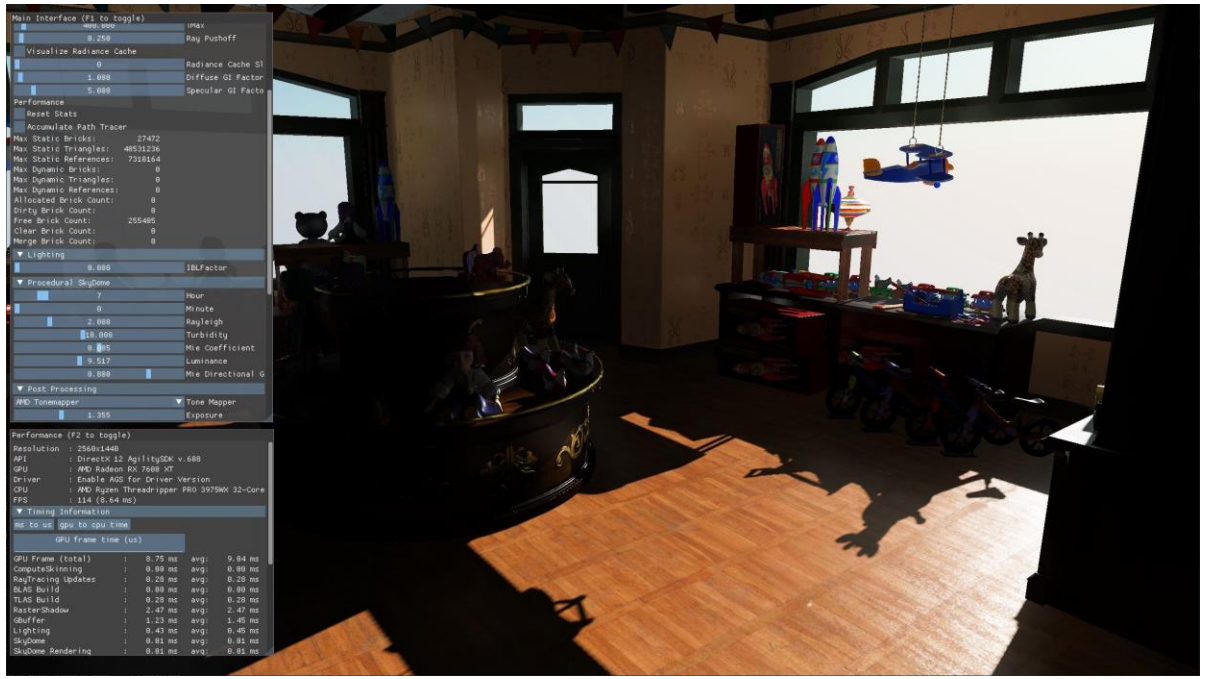
Watch the demo on YouTube

Version 1.1

AMD FIDELITYFX SDK UPDATES – WHAT'S NEW?

Brixelizer and Brixelizer GI

- DirectX® 12 and Vulkan® samples
- Demonstrates the use of both libraries
  - Static and Dynamic scene elements
  - Various debug views and counters



AMD FIDELITYFX SDK UPDATES – WHAT'S NEW?

Breadcrumbs

- A new library to help developers debug GPU crashes
- Places a trail of "breadcrumbs" around GPU work
- Dumps a tree of breadcrumbs when device is lost
- Helps determine the workload that was executed during the crash
- DirectX 12 and Vulkan samples

```
[BREADCRUMBS]
<Frame 250>
 - [>] Queue type <0>, submission no. 0, command list 1: "VK test command list"
    ├─[X] RESOURCE_BARRIER: "Backbuffer barrier to RT"
    ├─[>] Main Rendering
    │  ├─[X] CLEAR_RENDER_TARGET: "Reset current backbuffer contents"
    │  └─[>] DRAW_INDEXED: "Draw simple triangle"
    └─[ ] RESOURCE_BARRIER: "Backbuffer barrier to PRESENT"
<Frame 251>
 - [ ] Queue type <0>, submission no. 0, command list 1: "VK test command list"
    ├─[ ] RESOURCE_BARRIER: "Backbuffer barrier to RT"
    ├─[ ] Main Rendering
    │  ├─[ ] CLEAR_RENDER_TARGET: "Reset current backbuffer contents"
    │  └─[ ] DRAW_INDEXED: "Draw simple triangle"
    └─[ ] RESOURCE_BARRIER: "Backbuffer barrier to PRESENT"
<Frame 252>
 - [ ] Queue type <0>, submission no. 0, command list 1: "VK test command list"
    ├─[ ] RESOURCE_BARRIER: "Backbuffer barrier to RT"
    ├─[ ] Main Rendering
    │  ├─[ ] CLEAR_RENDER_TARGET: "Reset current backbuffer contents"
    │  └─[ ] DRAW_INDEXED: "Draw simple triangle"
    └─[ ] RESOURCE_BARRIER: "Backbuffer barrier to PRESENT"
```

AMD FIDELITYFX SDK UPDATES – WHAT'S NEW?

Breadcrumbs

- Different from AMD Radeon GPU Detective (RGD) and D3D12 Device Removed Extended Data (DRED)
- RGD is a tool that works on AMD Radeon GPUs and requires no changes to game code
- Breadcrumbs and DRED are vendor-agnostic and uses WriteBufferImmediate
- DRED operates at API-call level granularity
  - Cannot specify custom names

- Breadcrumbs allows you to choose the granularity of writes (e.g. – per-pass, per-draw)
  - ffxBreadcrumbsBeginMarker/ffxBreadcrumbsEndMarker
  - Can be plugged into existing profiler markers in your engine

AMD FIDELITYFX SDK UPDATES – WHAT'S NEW?

- Improvements to FSR sample
- Improvement to Hybrid Reflections sample
- Various fixes and improvements across all samples and framework

- New GDK backend (Desktop and Xbox Series)
  - Version that supports Xbox Series will be available to Xbox developers

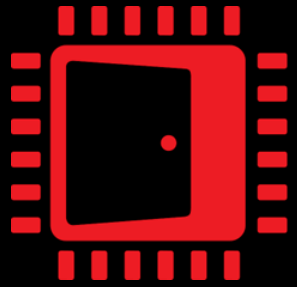
- Scheduled to be released post-GDC!

CONCLUSION

- Brixelizer GI is a library meant as a fallback for Ray Traced dynamic GI on lower-end platforms
- Take in the G-Buffer and direct lighting as input and output Diffuse and Specular GI
- Requires no hardware accelerated ray tracing support
- Works on DirectX 12 and Vulkan
- Fully open-source via the MIT license
- Will be available with AMD FidelityFX SDK 1.1
- Can serve as the basis for a more advanced GI solution

SPECIAL THANKS

- Anton Schreiner
- Jay Fraser
- Lou Kramer
- Petter Blomkvist
- Guillaume Boissé
- Colin Riley
- Nick Thibieroz
- Jason Lacroix
- Minesh Parekh
- Marek Machlinski
- Mark Simpson
- Aurelien Serandour
- Rosanna Ashworth-Jones
- Tom Lewis
- Francois Guthmann
- Alex Pecoraro
- Stephan Hodes
- Hao Zheng
- Meith Jhaveri
- Li (Leo) Qiu
- Fabio Camaiora
- Ivor Li
- Hui Zhang



AMD
GPUOpen

Visit our website
https://gpuopen.com

QUESTIONS?

Dihara.Wijetunga@amd.com

Follow us on X
https://twitter.com/GPUOpen

Follow us on Mastodon
https://mastodon.gamedev.place/@gpuopen

知乎

Follow us on Zhihu
https://www.zhihu.com/org/gpuopen-7

DISCLAIMER

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

Use of third-party marks / products is for informational purposes only and no endorsement of or by AMD is intended or implied. GD-83

Performance testing done on following system: AMD Ryzen™ Threadripper™ PRO 3975WX, 128GB DDR4-3600 memory, ASUS Pro WS WRX80E-SAGE SE WIFI Motherboard, 1TB M.2 NVME SSD, Windows® 10 Pro 22H2

together we advance_