SIGGRAPH 2020 S2020.SIGGRAPH.ORG

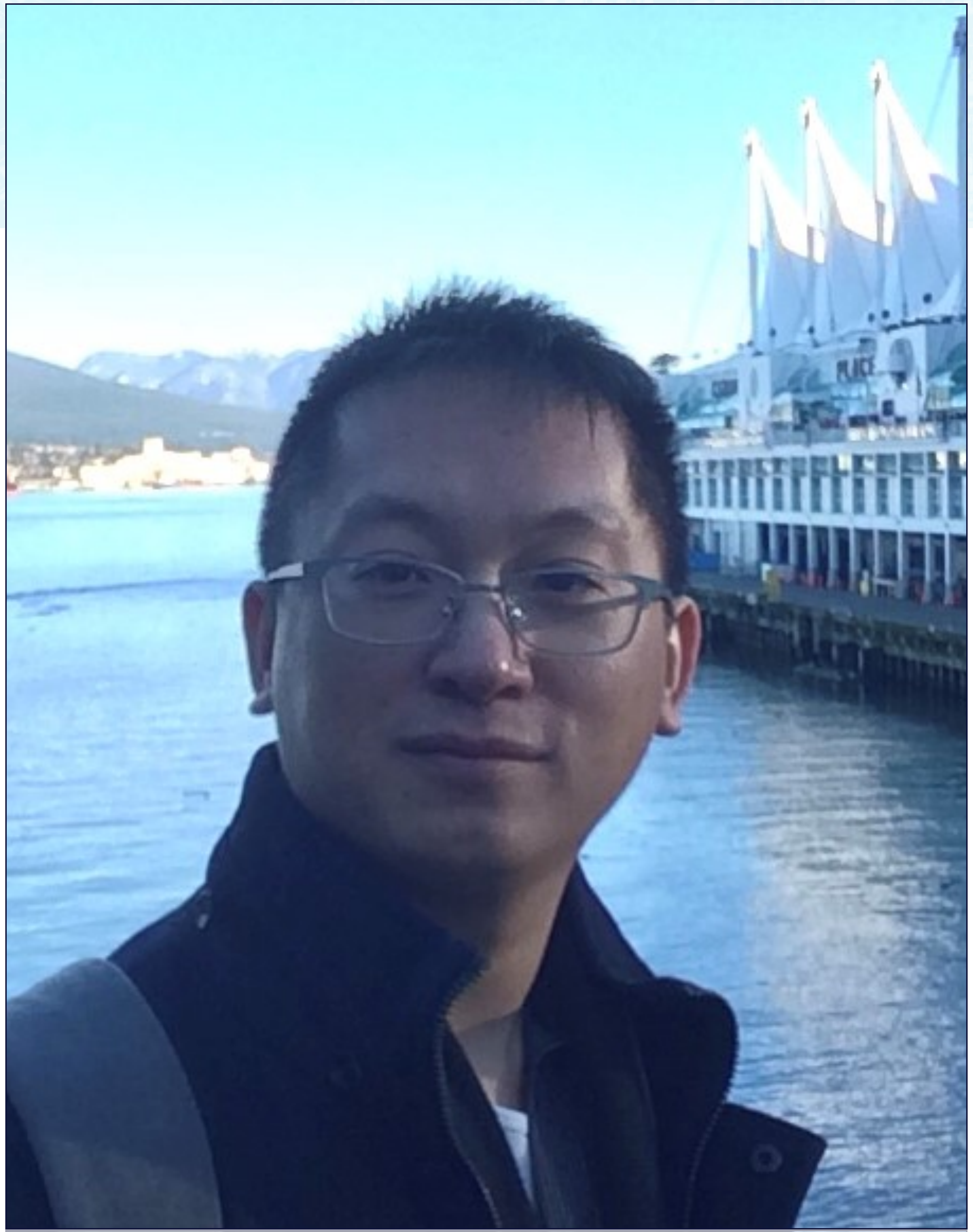
THINK BEYOND

Deferred Shading techniques in the Unity Universal Render Pipeline

Kay Chang
(Unity Technologies)

RECORDING POLICY

It is important to recognize that many of the words, images, sounds, objects, and technologies presented at SIGGRAPH are protected by copyrights or patents. They are owned by the people who created them. Please respect their intellectual-property rights by refraining from making recordings from your device or taking screenshots. If you are interested in the content, feel free to reach out to the contributor or visit the ACM SIGGRAPH Digital library after the event, where the proceedings will be made available.



KAY CHANG

SENIOR SOFTWARE ENGINEER, GRAPHICS

Vancouver (Canada) / Unity Technologies

In the video games industry for +14 years

- Shipped games on all major consoles
- Specialised in graphics rendering

AGENDA

https://github.com/Verasl/BoatAttack

- The Universal Render Pipeline?
- Platforms considerations
- Stencil-based deferred shading
- Tiled-based deferred shading
- Performance results
- Moving forward

THE UNIVERSAL RENDER PIPELINE

unity

UNIVERSAL RENDER PIPELINE

SIGGRAPH THINK BEYOND
2020 S2020.SIGGRAPH.ORG

```
public sealed class DeferredRenderer : ScriptableRenderer
{
    public static readonly int k_DepthStencilBufferBits = 32;

    static readonly string k_CreateCameraTextures = "Create Camera Texture";

    ColorGradingLutPass m_ColorGradingLutPass;
    DepthOnlyPass m_DepthPrepass;
    MainLightShadowCasterPass m_MainLightShadowCasterPass;
    AdditionalLightsShadowCasterPass m_AdditionalLightsShadowCasterPass;
    ScreenSpaceShadowResolvePass m_ScreenSpaceShadowResolvePass;
    GBufferPass m_GBufferPass;
    TileDepthRangePass m_TileDepthRangePass;
    TileDepthRangePass m_TileDepthRangeExtraPass;
    DeferredPass m_DeferredPass;
    DrawObjectsPass m_RenderOpaqueForwardOnlyPass;
    DrawSkyboxPass m_DrawSkyboxPass;
    CopyDepthPass m_CopyDepthPass0; // first copy for deferred shading pass
    CopyDepthPass m_CopyDepthPass1; // second copy after forward-only pass
    CopyColorPass m_CopyColorPass;
    TransparentSettingsPass m_TransparentSettingsPass;
    DrawObjectsPass m_RenderTransparentForwardPass;
    InvokeOnRenderObjectCallbackPass m_OnRenderObjectCallbackPass;
    PostProcessPass m_PostProcessPass;
    PostProcessPass m_FinalPostProcessPass;
    FinalBlitPass m_FinalBlitPass;
    CapturePass m_CapturePass;

#if UNITY_EDITOR
    SceneViewDepthCopyPass m_SceneViewDepthCopyPass;
#endif

    RenderTargetHandle m_ActiveCameraColorAttachment;
    RenderTargetHandle m_ActiveCameraDepthAttachment;
    RenderTargetHandle m_CameraColorTexture;
    RenderTargetHandle m_CameraDepthTexture;
    RenderTargetHandle m_CameraDepthAttachment;
    RenderTargetHandle[] m_GBufferAttachments;
    RenderTargetHandle m_OpaqueColor;
    RenderTargetHandle m_AfterPostProcessColor;
    RenderTargetHandle m_ColorGradingLut;
    RenderTargetHandle m_DepthInfoTexture;
    RenderTargetHandle m_TileDepthInfoTexture;

    ForwardLights m_ForwardLights; // Required for transparent pass
    DeferredLights m_DeferredLights;
    bool m_PreferDepthPrepass;
    StencilState m_DefaultStencilState;

    Material m_BlitMaterial;
    Material m_CopyDepthMaterial;
    Material m_SamplingMaterial;
    Material m_ScreenspaceShadowsMaterial;
    Material m_TileDepthInfoMaterial;
    Material m_TileDeferredMaterial;
    Material m_StencilDeferredMaterial;
```

- Historically, Unity rendering is performed via a built-in rendering pipeline (opaque C++ code)
- Scriptable Render Pipeline (SRP) is a C# API for writing render pipelines
- Unity Out-of-the-box implementations:
  - High-Definition Render Pipeline (**HDRP**)
  - Universal Render Pipeline (**URP**)
- URP is the future Unity replacement for the built-in render pipeline
- Mostly everything in C# land, users have more leverage customising the render pipeline to game requirements
- Hosted on ***GitHub*** to let users access latest development: https://github.com/Unity-Technologies/Graphics

UNIVERSAL RENDER PIPELINE

- To reach parity with built-in pipeline, URP needs:
  - Deferred shading
  - Decals
  - Point lights shadows
  - Advanced baked lighting modes
  - Advanced reflection probes
  - Separate light layers
  - etc.
- Unity aiming to reach feature parity in 2021

Working on this next …

UNIVERSAL RENDER PIPELINE

- Deferred shading is a broad technique, several possible implementations:
  - Stencil: Render and shade light shapes using stencil buffer (traditional)
  - Tiled based deferred shading (software): classify lights into screen space tiles and render tiles using compute shaders (popular for desktop GPUs)
- URP must run on a large set of platforms
  - low-end mobile devices
  - high-end desktops / home consoles
  - ***Everything in between!***
- Must reach good performance on all platforms
  - **No solutions to fit them all**

PLATFORM CONSIDERATIONS

PLATFORM CONSIDERATIONS

SIGGRAPH THINK BEYOND 2020 S2020.SIGGRAPH.ORG

What are our platforms constraints?

- Mobile GPUs:
  - Often have a tiled-based deferred shading architecture (hardware)
  - Low fill-rate
  - Expensive ALUs
  - Overall limited capabilities
    - Limited Multiple Render Target Count
    - Limited uniform buffer size for best performance

Photo by Obi Onyeador on Unsplash

PLATFORM CONSIDERATIONS

What are our platforms constraints?

- Desktop GPUs (Desktop PC, home consoles):
  - Cheap ALUs
  - fast dynamic branching
  - Fast memory access
  - Efficient compute pipes / async compute pipes
  - Overall very capable

Photo by Steven Binotto on Unsplash

PLATFORM CONSIDERATIONS

SIGGRAPH THINK BEYOND 2020 S2020.SIGGRAPH.ORG

Main takeaways

- Prefer fragment shaders to compute shaders
  - Compute shaders are as fast as fragment shaders only on desktop/home consoles
- Prefer uniform buffers to structured buffers/SSBO
  - Uniform buffers are better cached even on most modern GPUs
- Problem: uniform buffers are limited to 64KiB
  - May need to split draw calls and extra management of buffers
- Use half types in shaders
  - Decrease register pressure on GPUs that support it

33% improvement by moving light data from structured buffer to uniform buffer on Nvidia

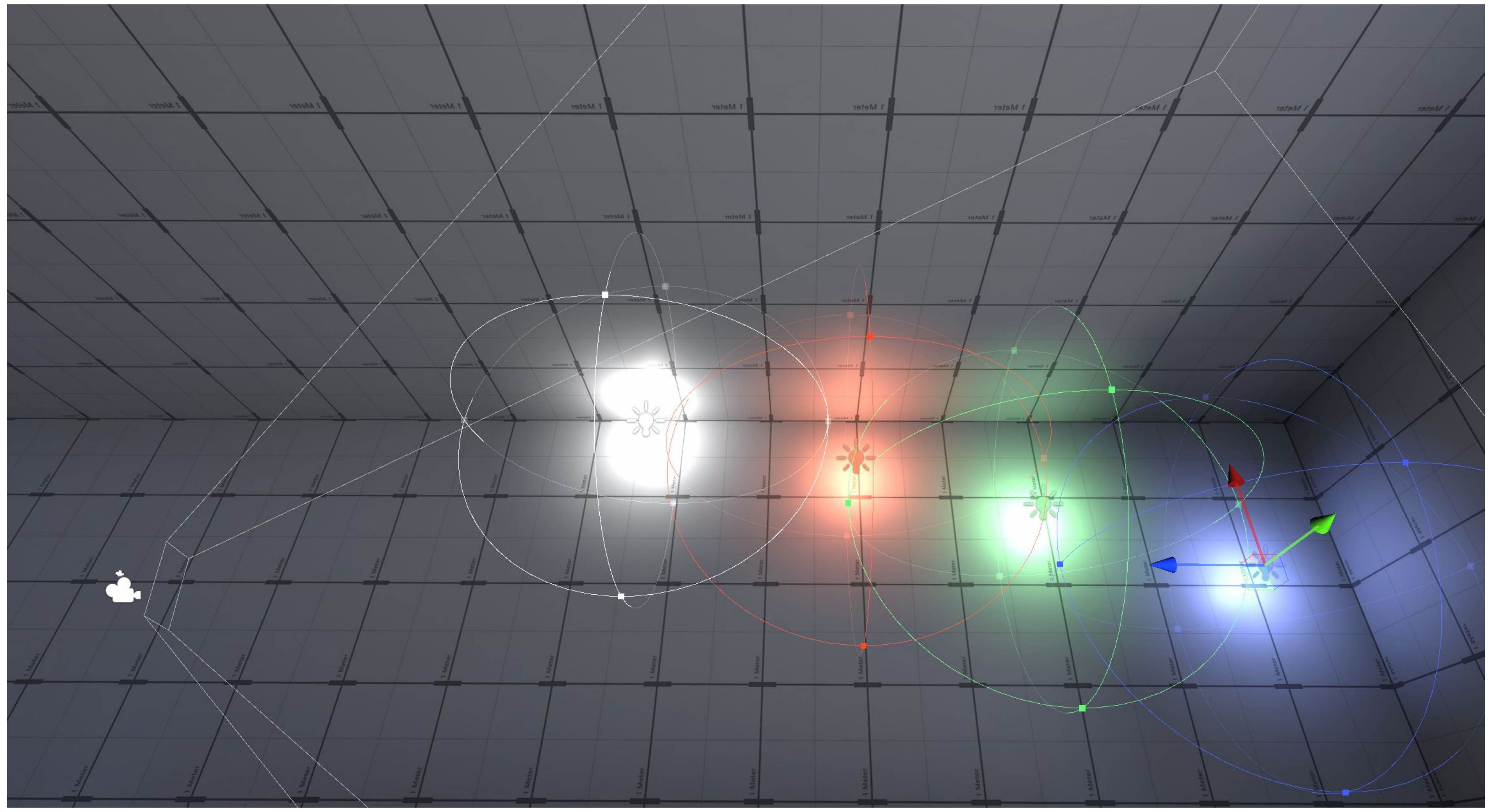
STENCIL-BASED DEFERRED SHADING

STENCIL-BASED DEFERRED SHADING

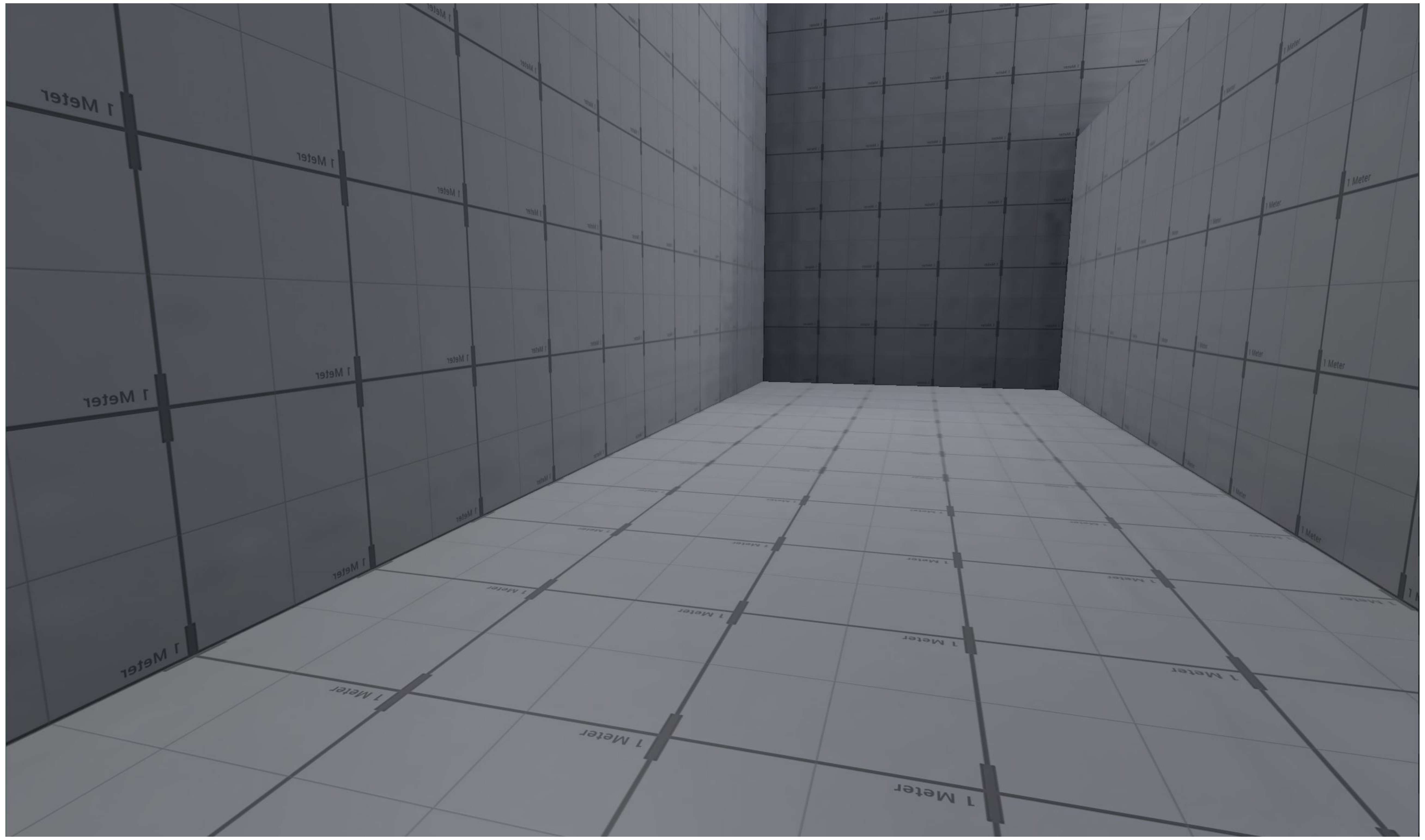
- Algorithm overview
  - Render g-buffer to store all material properties
  - Light shapes are rendered as convex geometry
    - Spheres for point lights
    - Capped rounded cones for spot lights
  - Stencil buffer is used to mask fragments inside light shapes for rendering
    - Each light shape is rendered twice
      - First time to set up stencil mask
      - Second time to to perform actual shading by sampling g-buffer and clear the stencil mask

STENCIL-BASED DEFERRED SHADING

Example stencil rendering

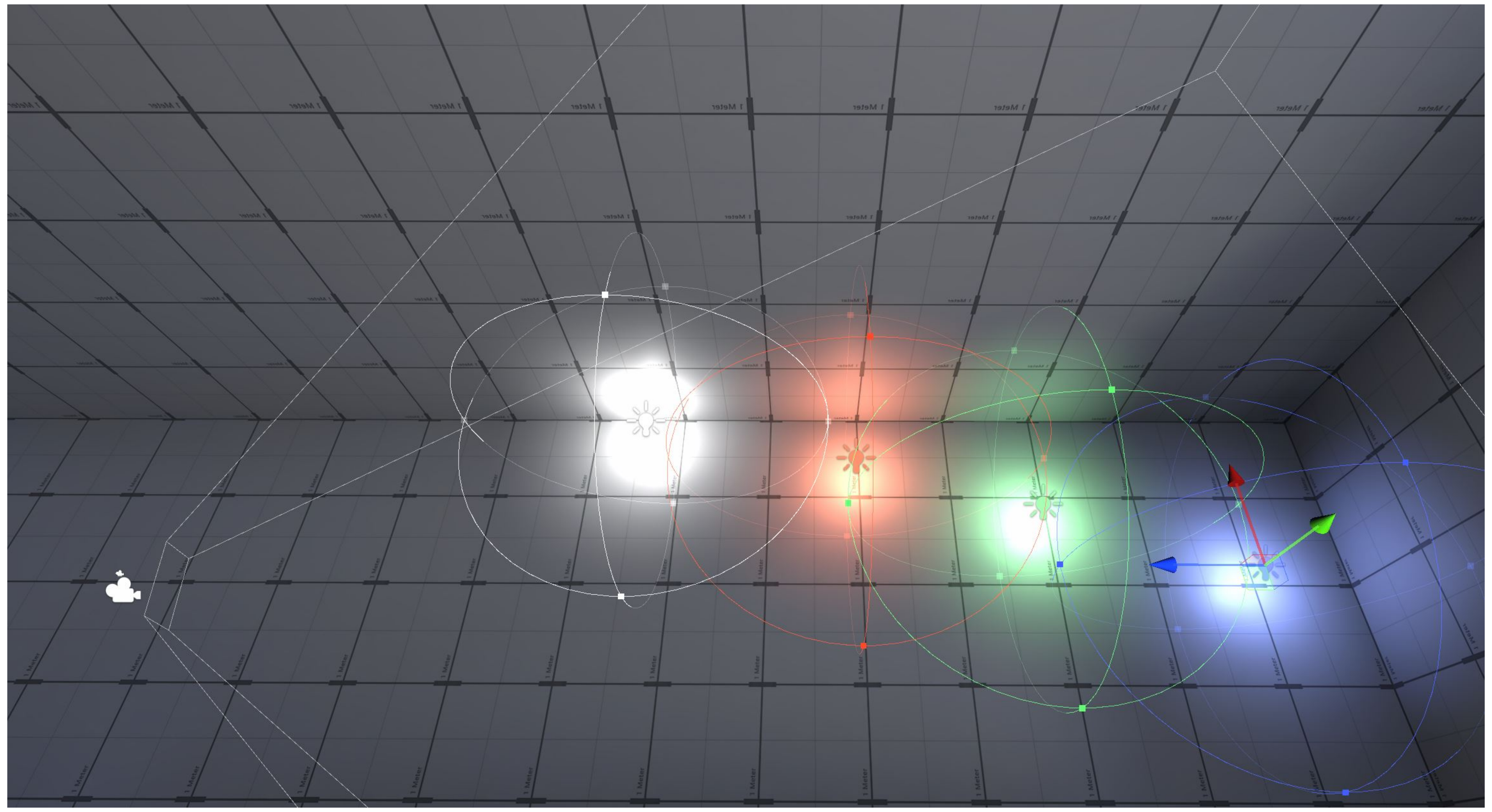


View from above

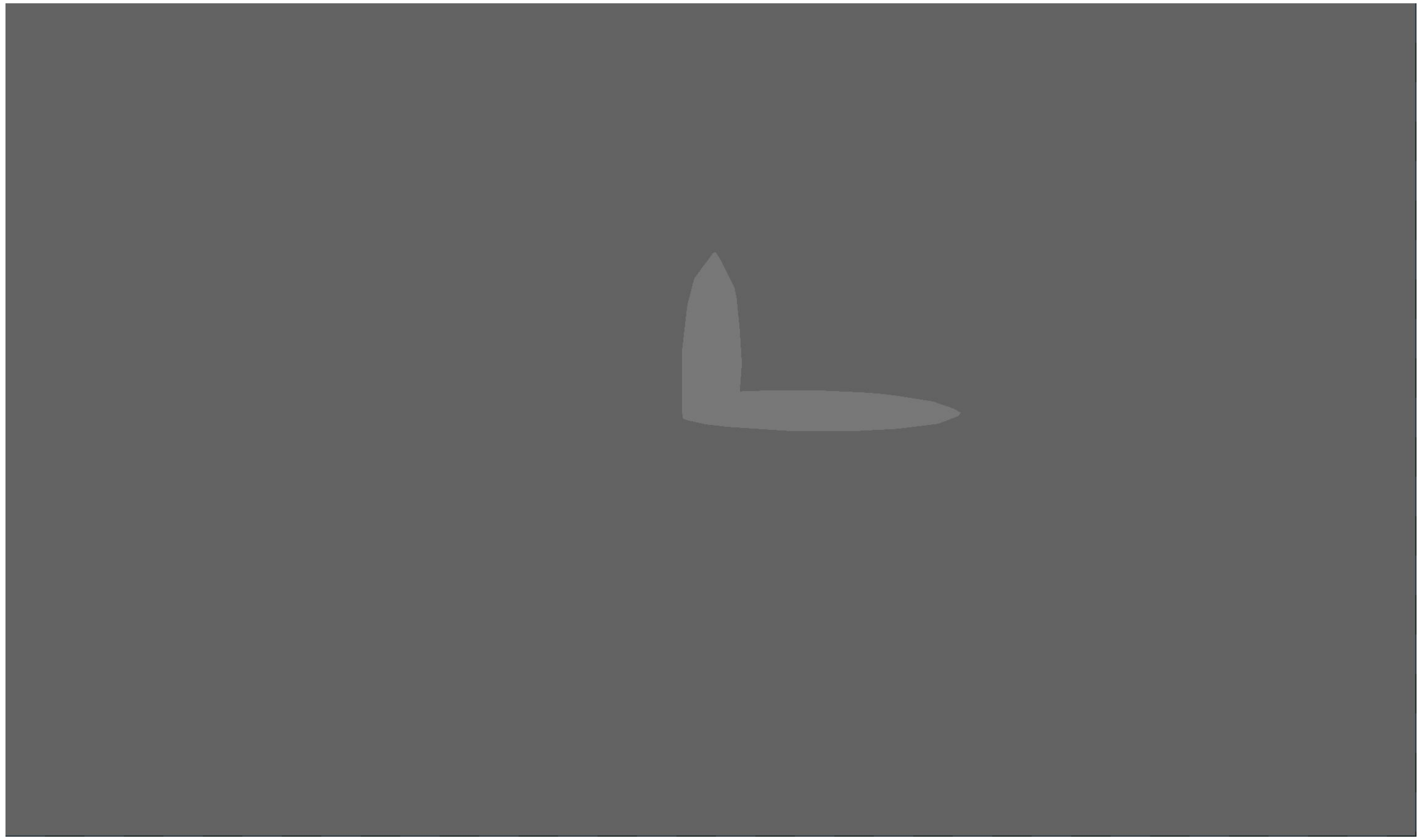


STENCIL-BASED DEFERRED SHADING

Example stencil rendering



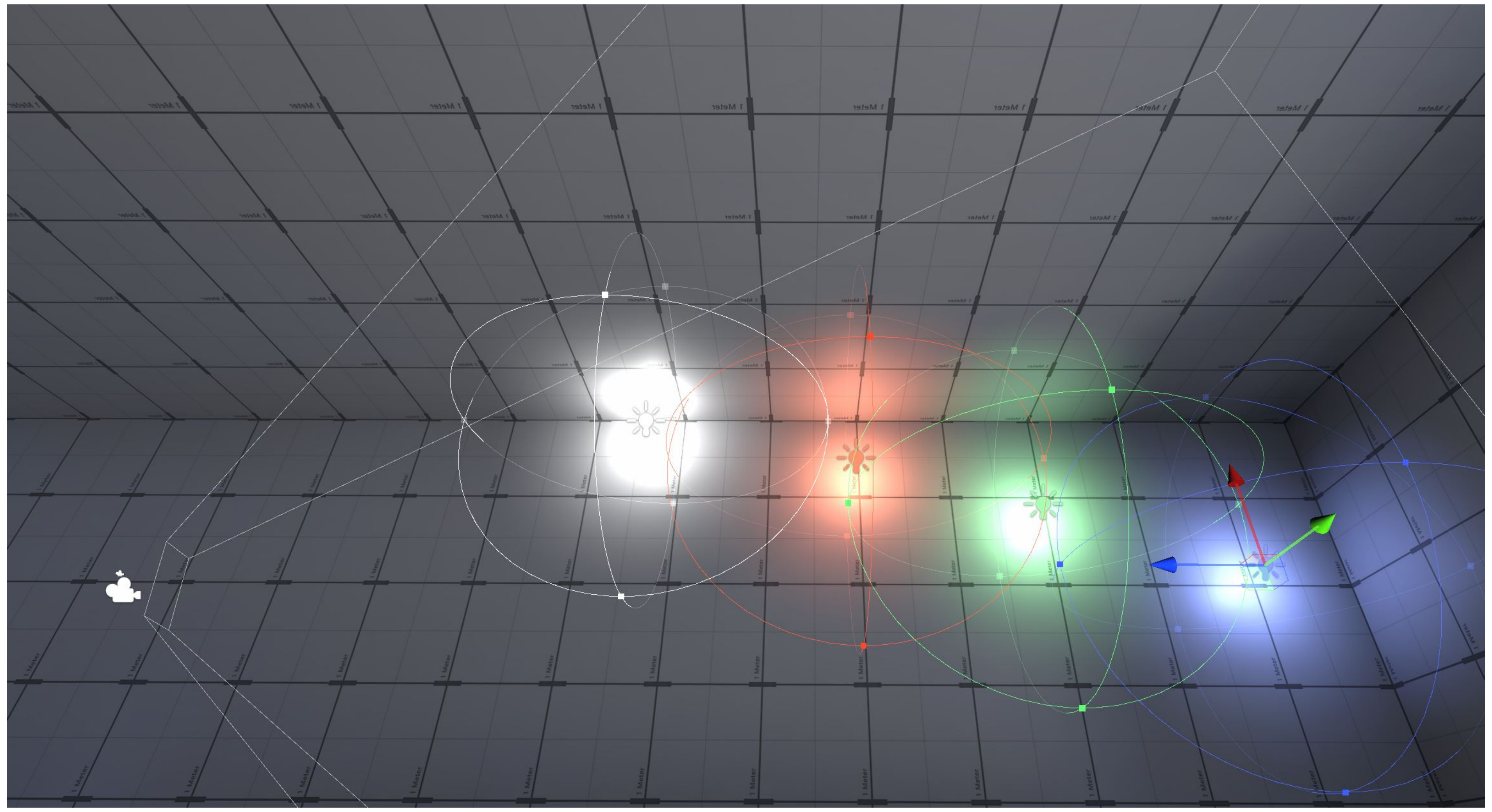
View from above



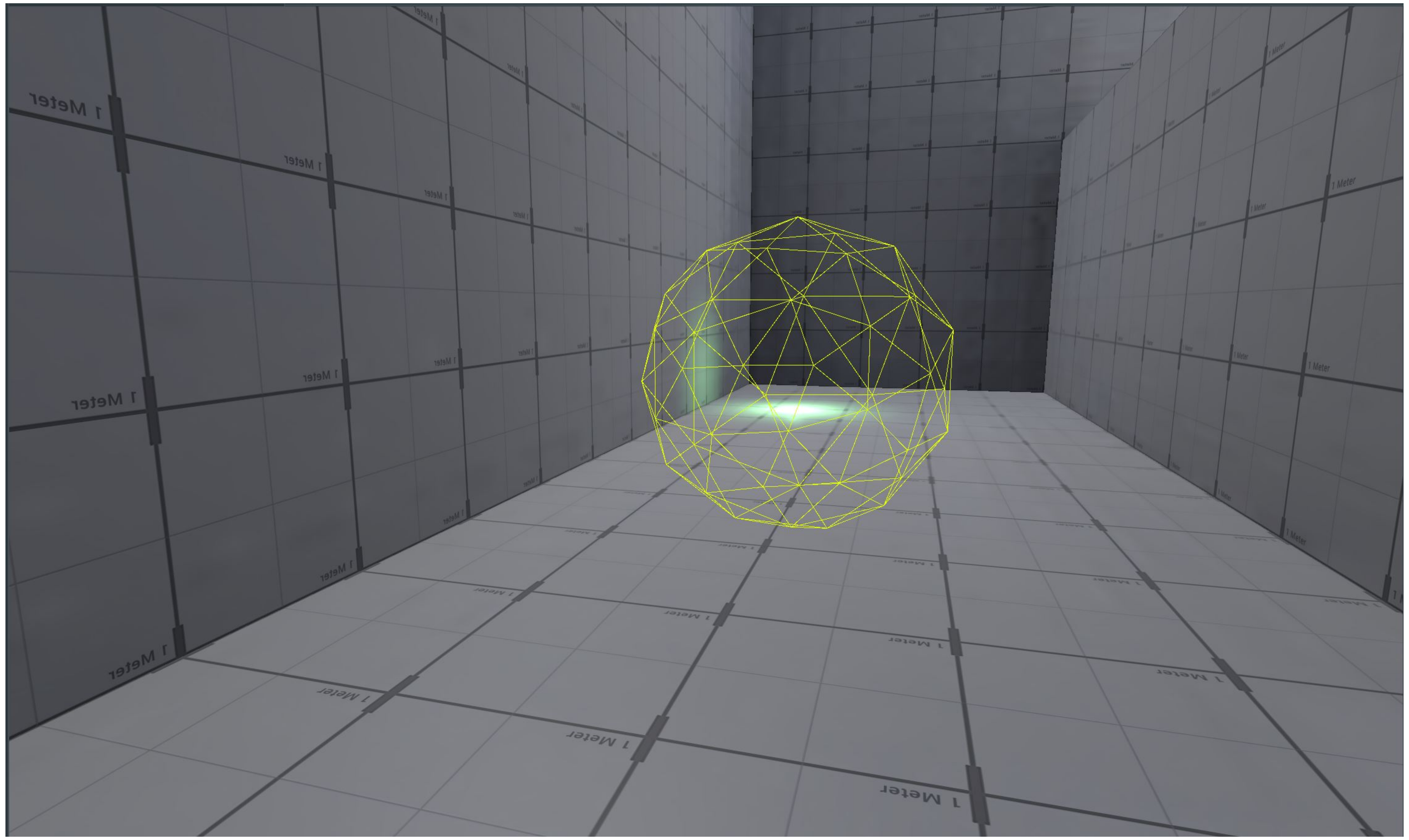
1st light stencil mask

STENCIL-BASED DEFERRED SHADING

Example stencil rendering



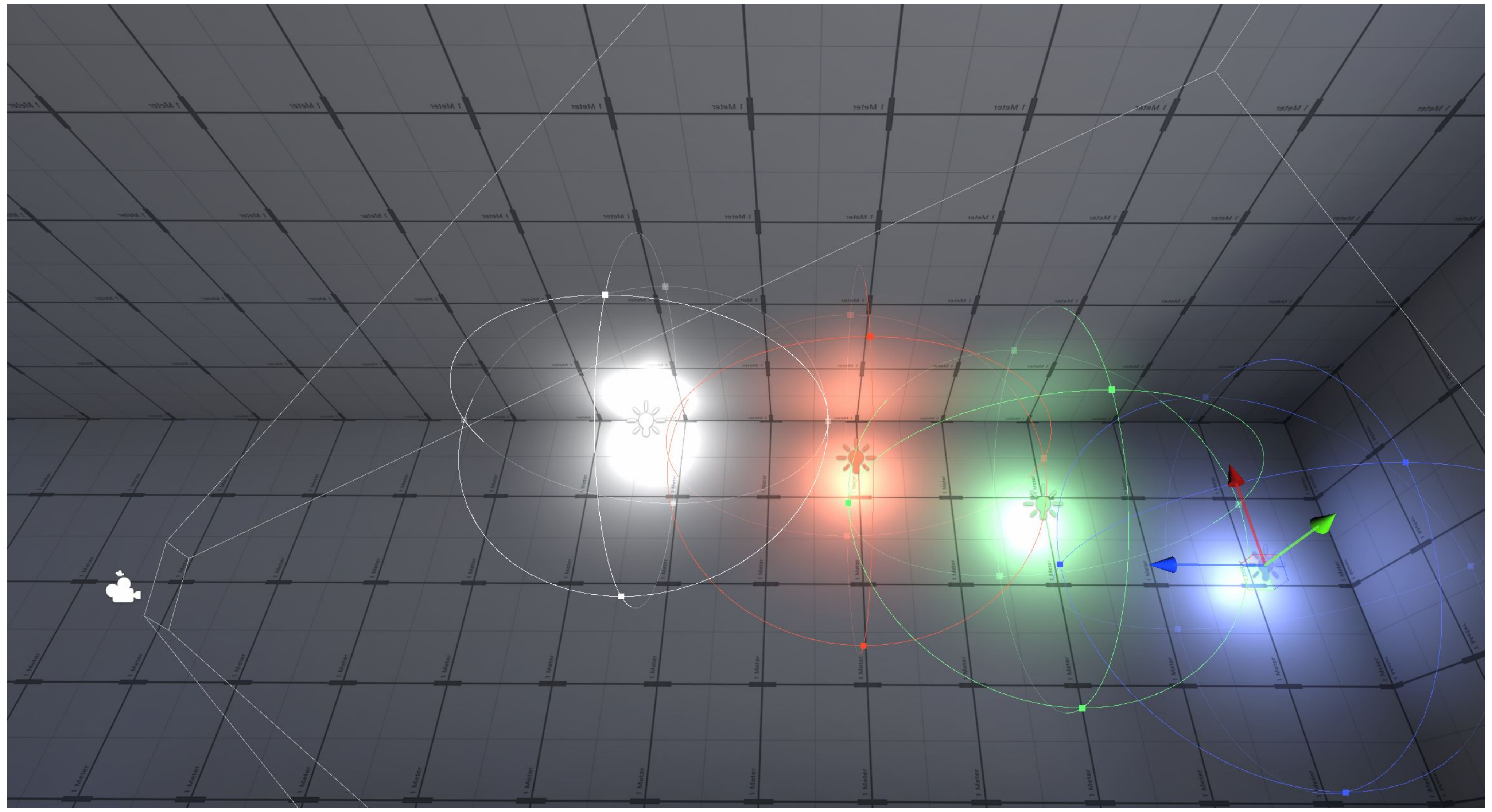
View from above



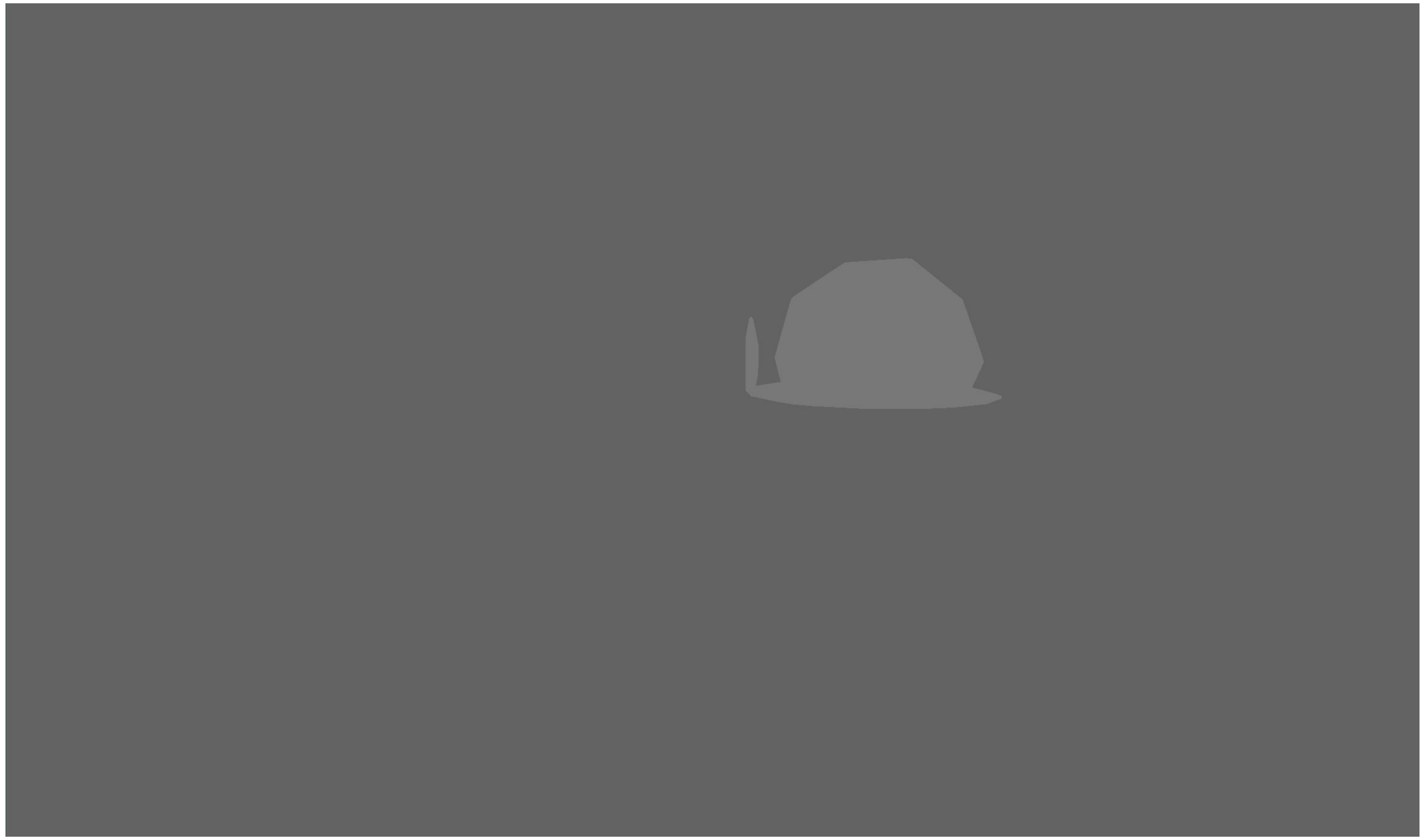
1st light

STENCIL-BASED DEFERRED SHADING

Example stencil rendering



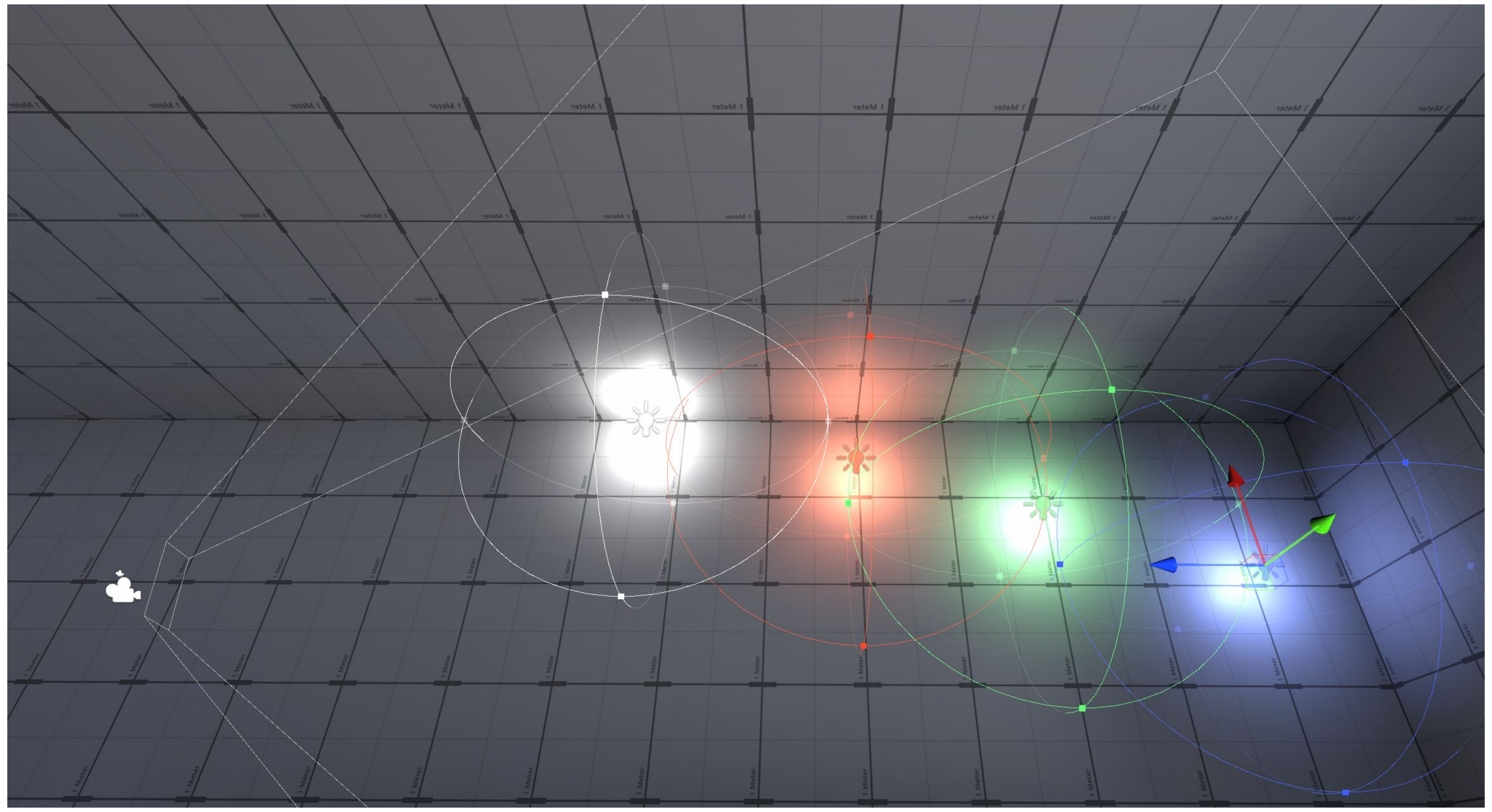
View from above



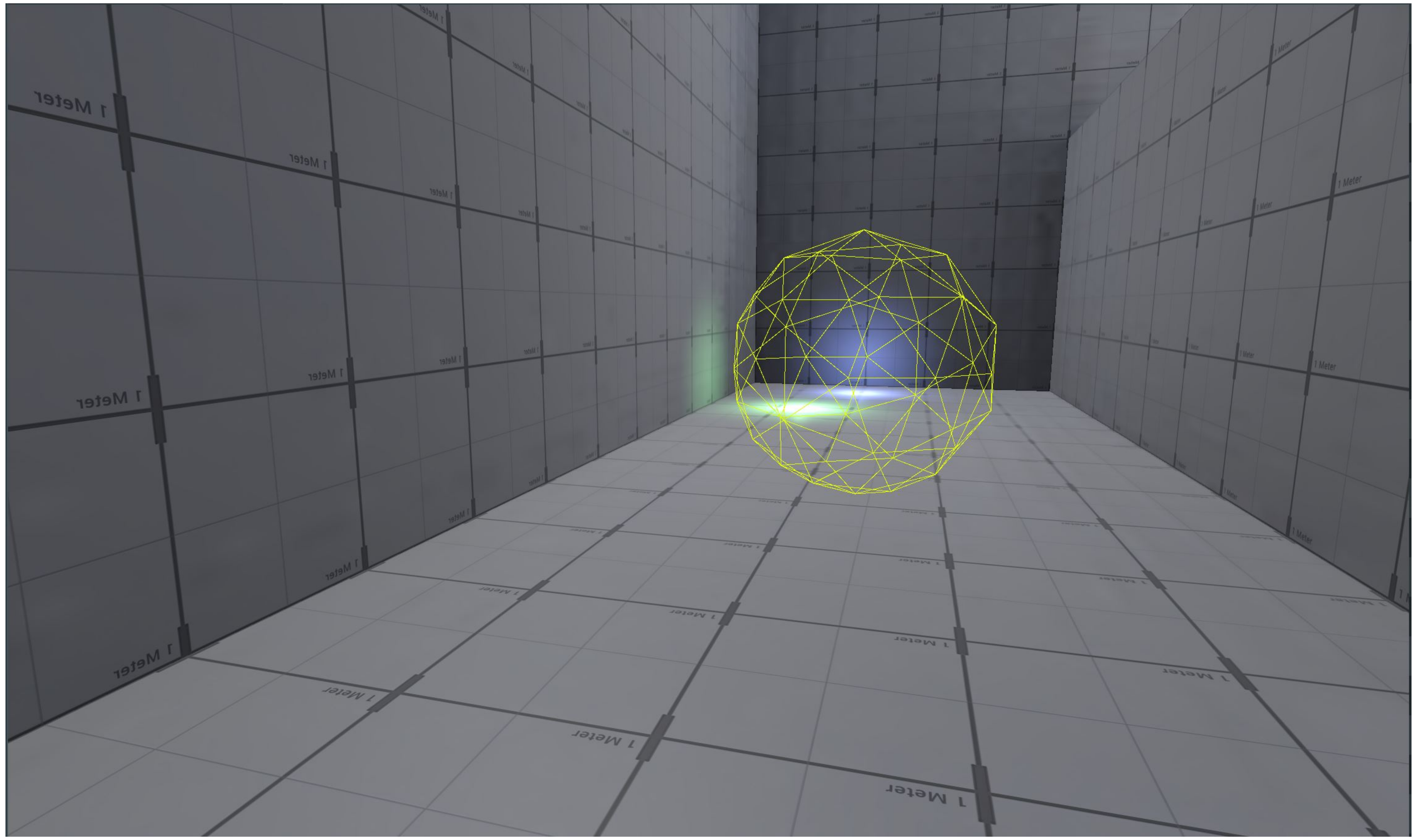
2nd light stencil mask

STENCIL-BASED DEFERRED SHADING

Example stencil rendering



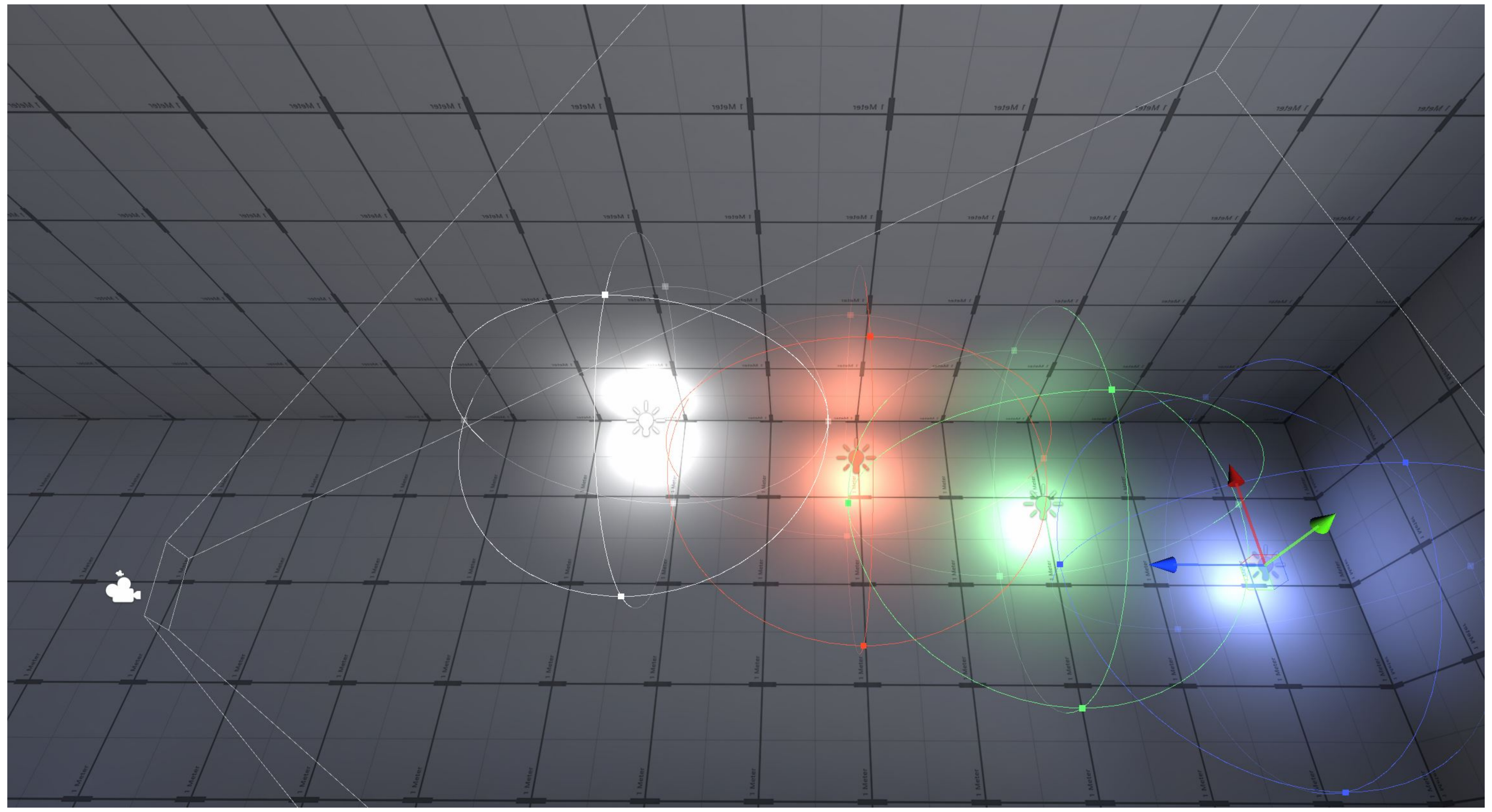
View from above



2nd light

STENCIL-BASED DEFERRED SHADING

Example stencil rendering

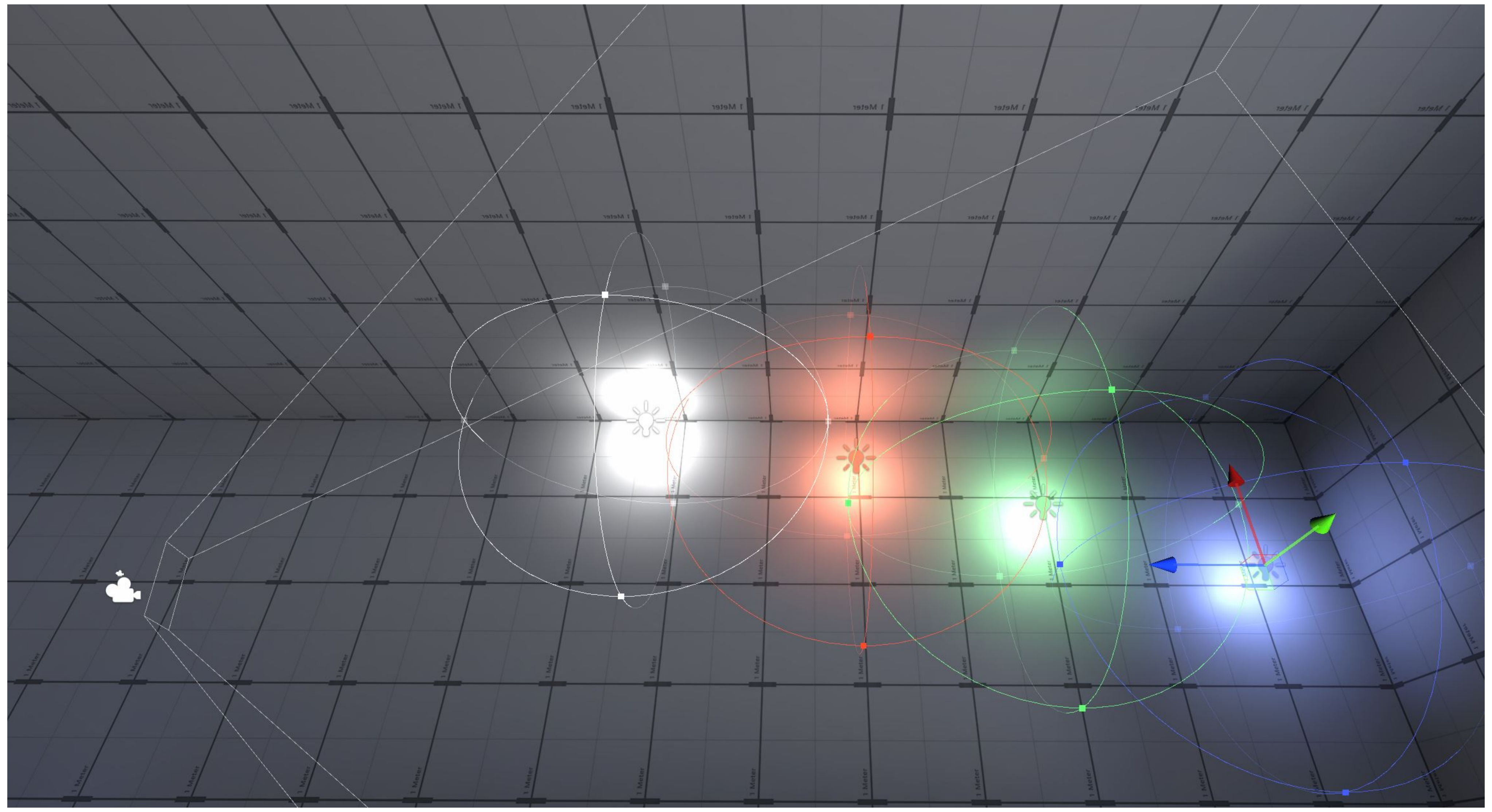


View from above

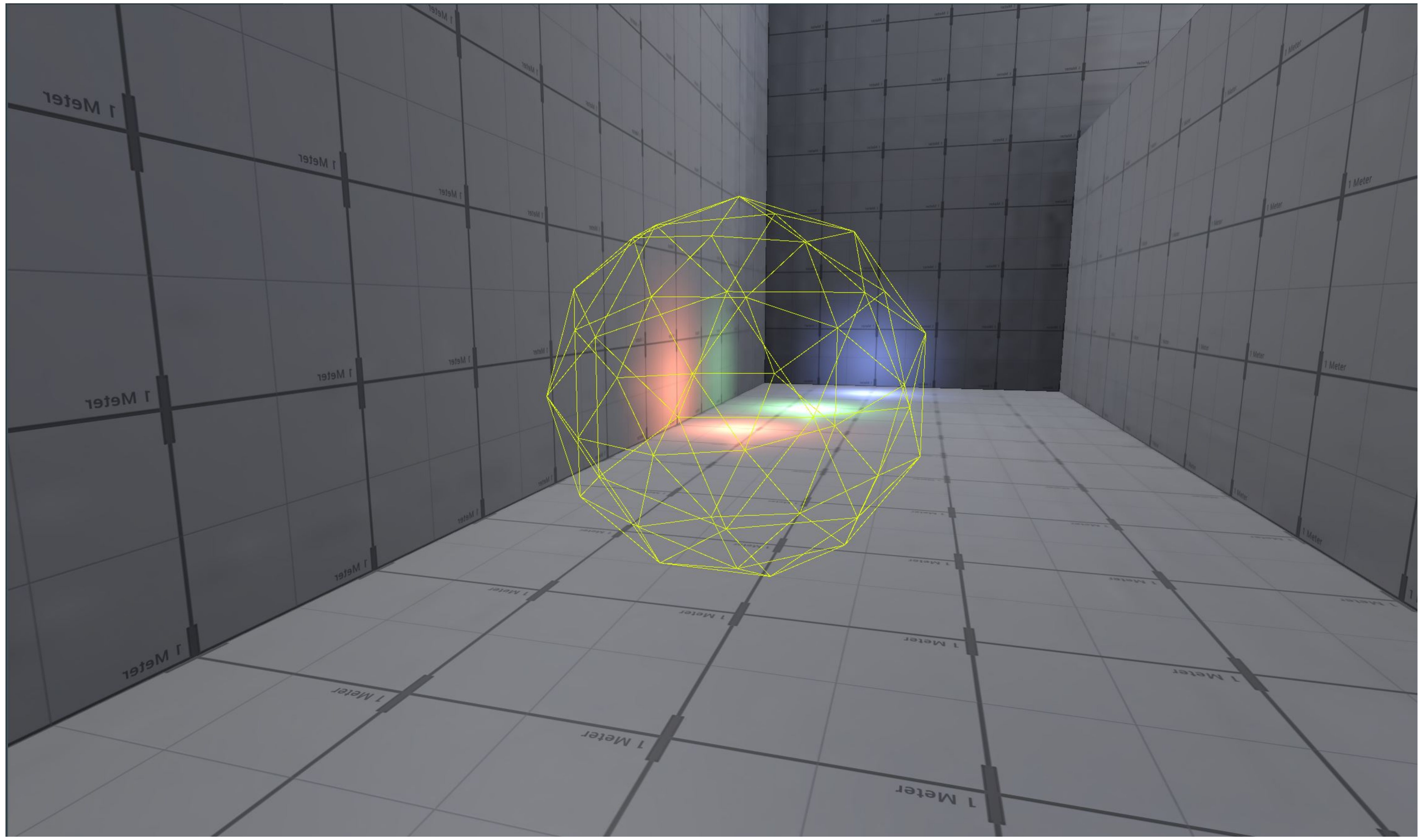
3rd light stencil mask

STENCIL-BASED DEFERRED SHADING

Example stencil rendering



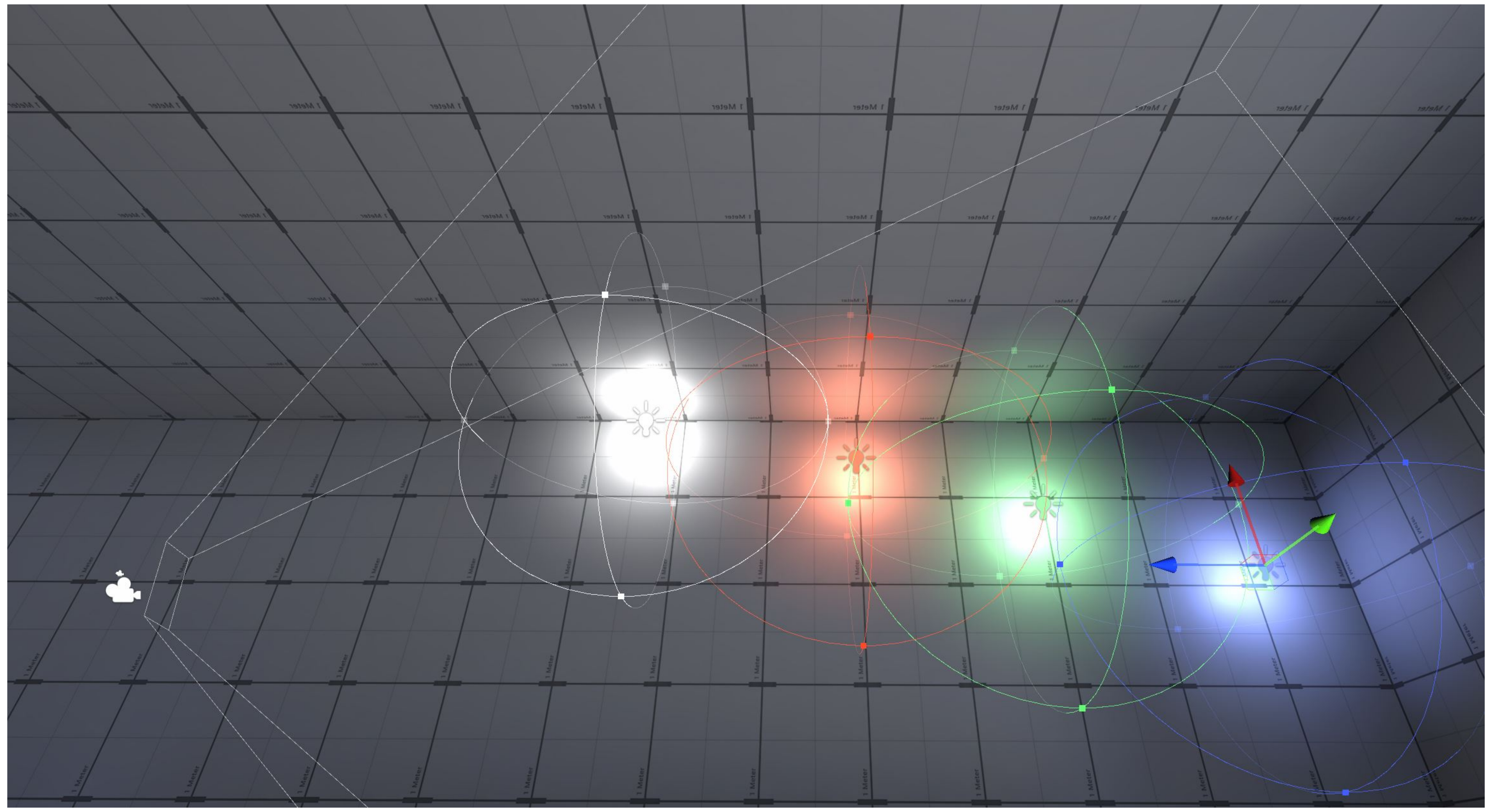
View from above



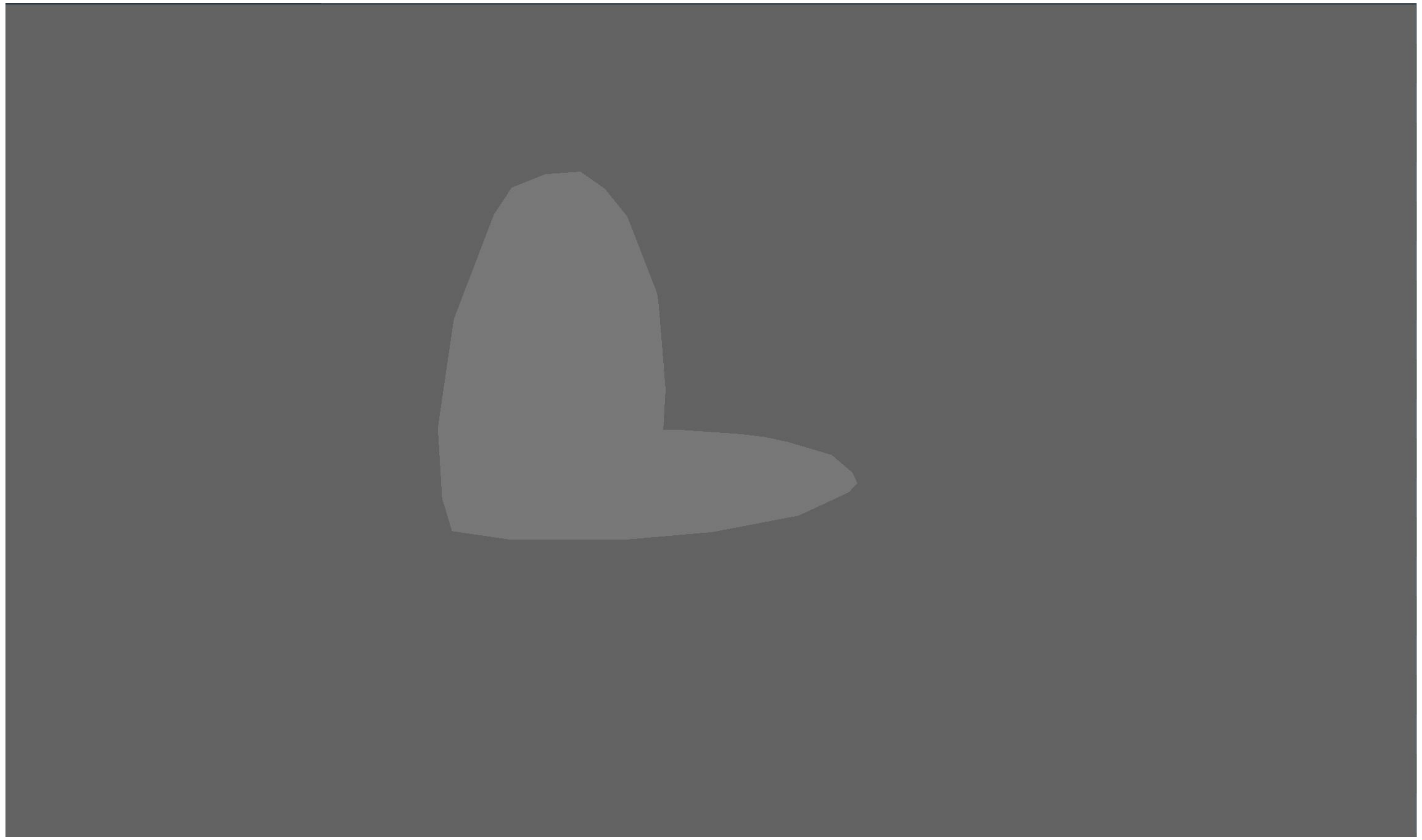
3rd light

STENCIL-BASED DEFERRED SHADING

Example stencil rendering



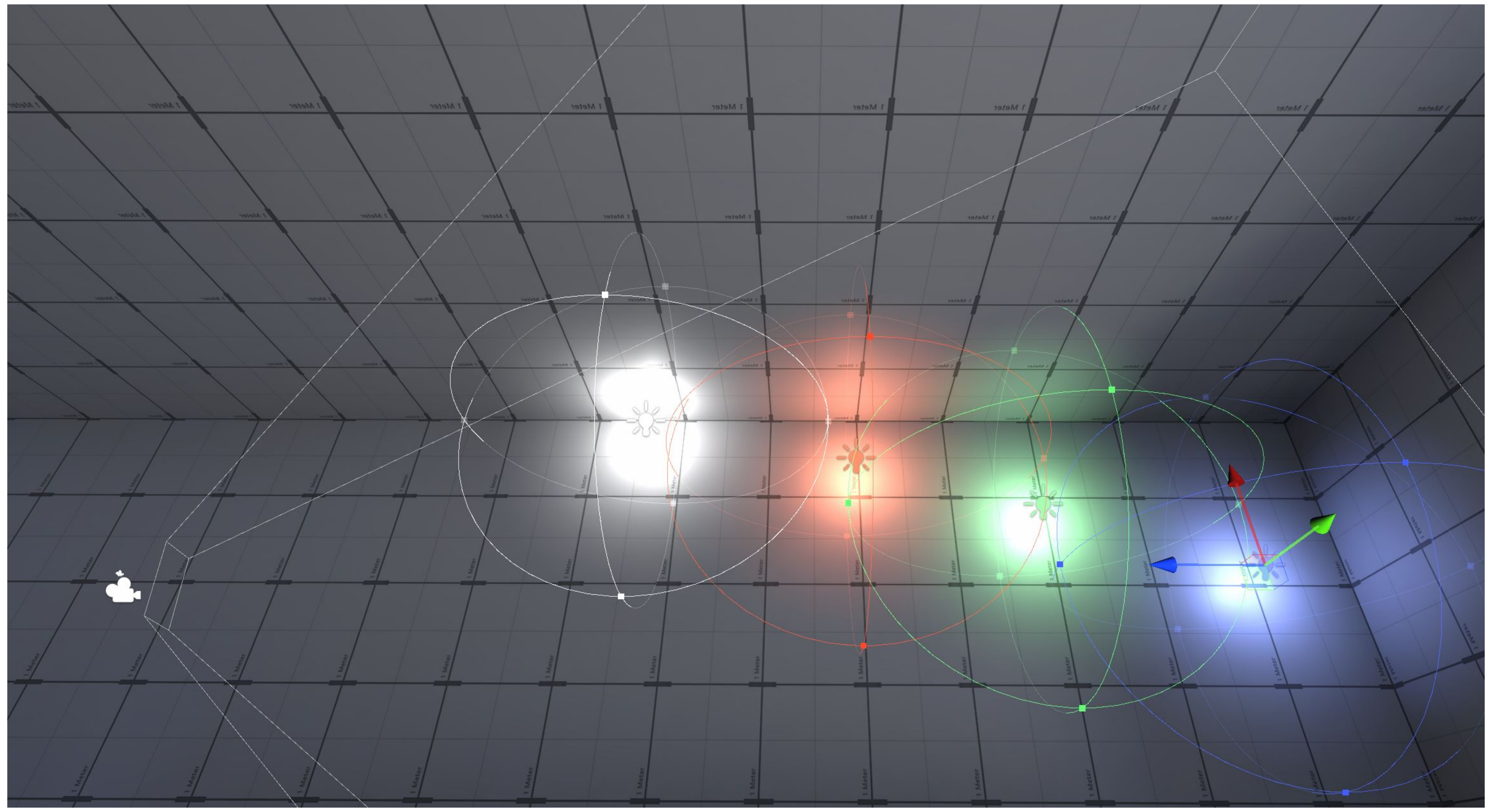
View from above



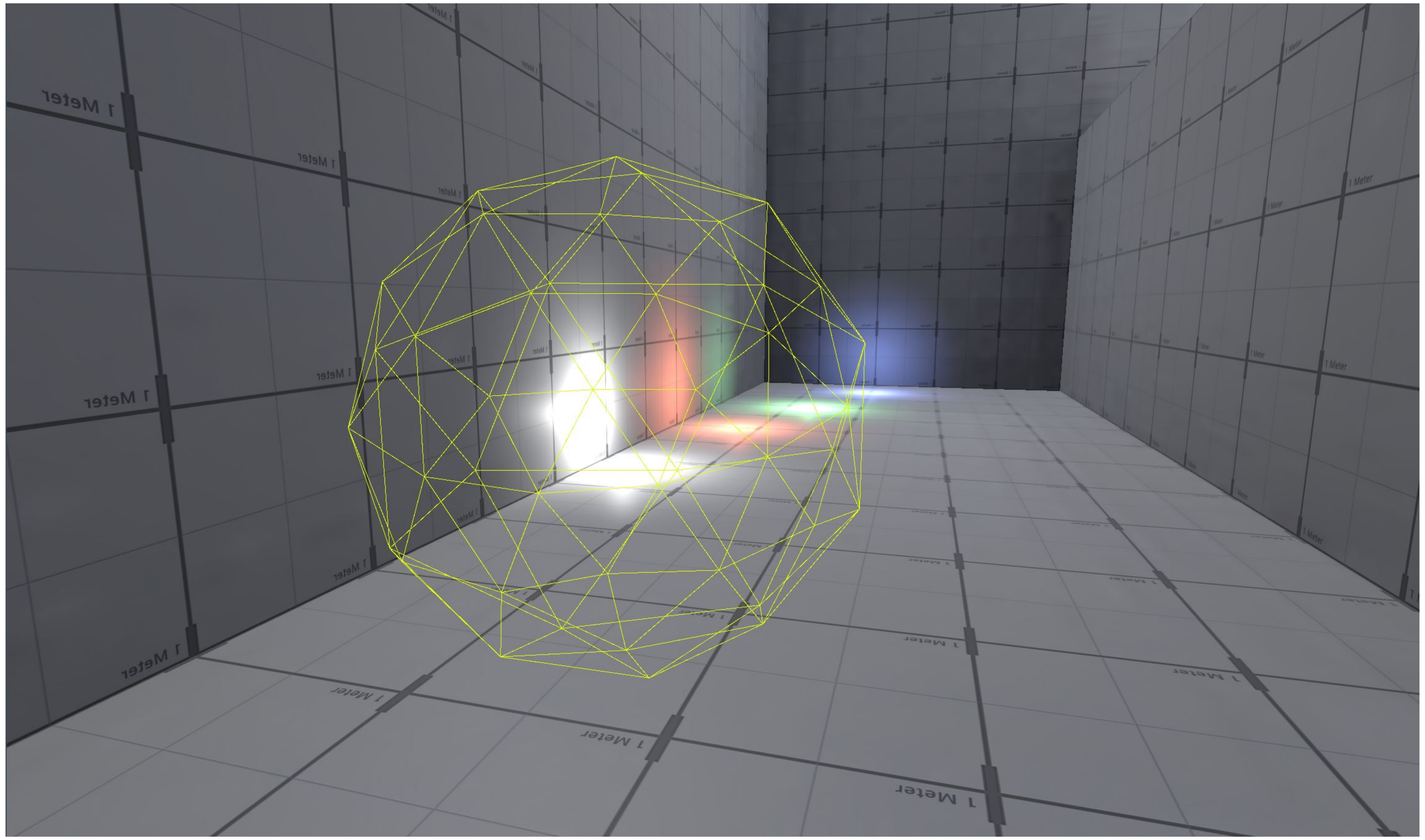
4th light stencil mask

STENCIL-BASED DEFERRED SHADING

Example stencil rendering



View from above



4th light

STENCIL-BASED DEFERRED SHADING

- Pros
  - Simple shaders run much faster
  - Most reasonable GPUs can support it (very old technique)
  - Now can take advantage of mobile-GPU hardware tiled architectures.

Render Pass API on Vulkan,
ImageBlocks on Metal

- Cons
  - Many draw calls and render state changes still limit how many lights can be rendered
  - Memory bandwidth intensive (repeated fetch of g-buffer textures).

STENCIL-BASED DEFERRED SHADING

SIGGRAPH THINK BEYOND 2020 S2020.SIGGRAPH.ORG

Our G-buffer layout

Format	R	G	B	A
R8G8B8A8	Albedo			Flags
R8G8B8A8	Specular			Unused
R8G8B8A8	Normal			Smoothness
R11G11B12	Emissive/baked lighting			
(R32F	Depth as color			)
D24S8	Depth			Stencil

Unity supports metallic and specular workflow for materials, forcing us to consume 3 floats for specular, instead of 1!

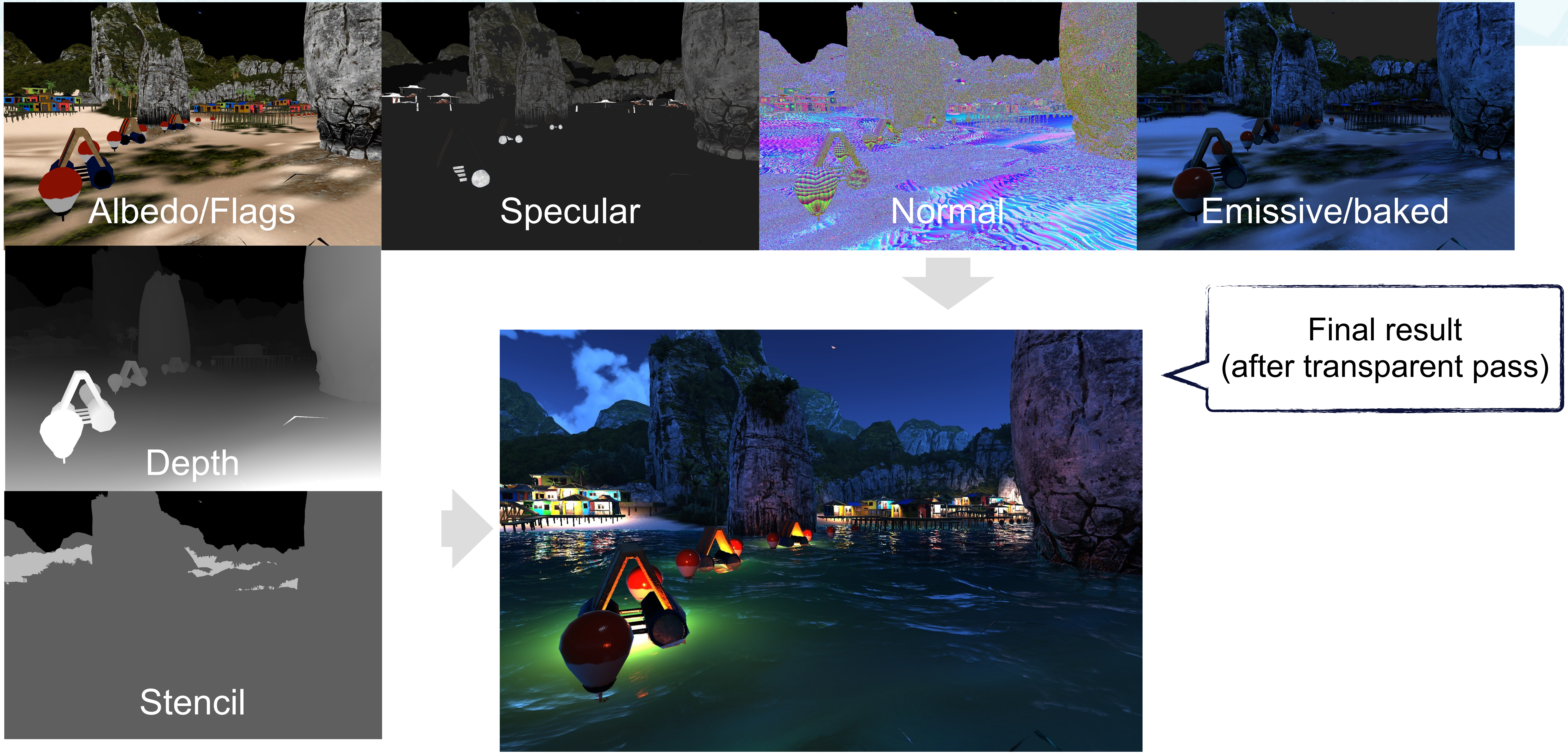
Optional octahedron encoding for normals, but always 24 bits

Optional for render-pass

Stencil stores some material

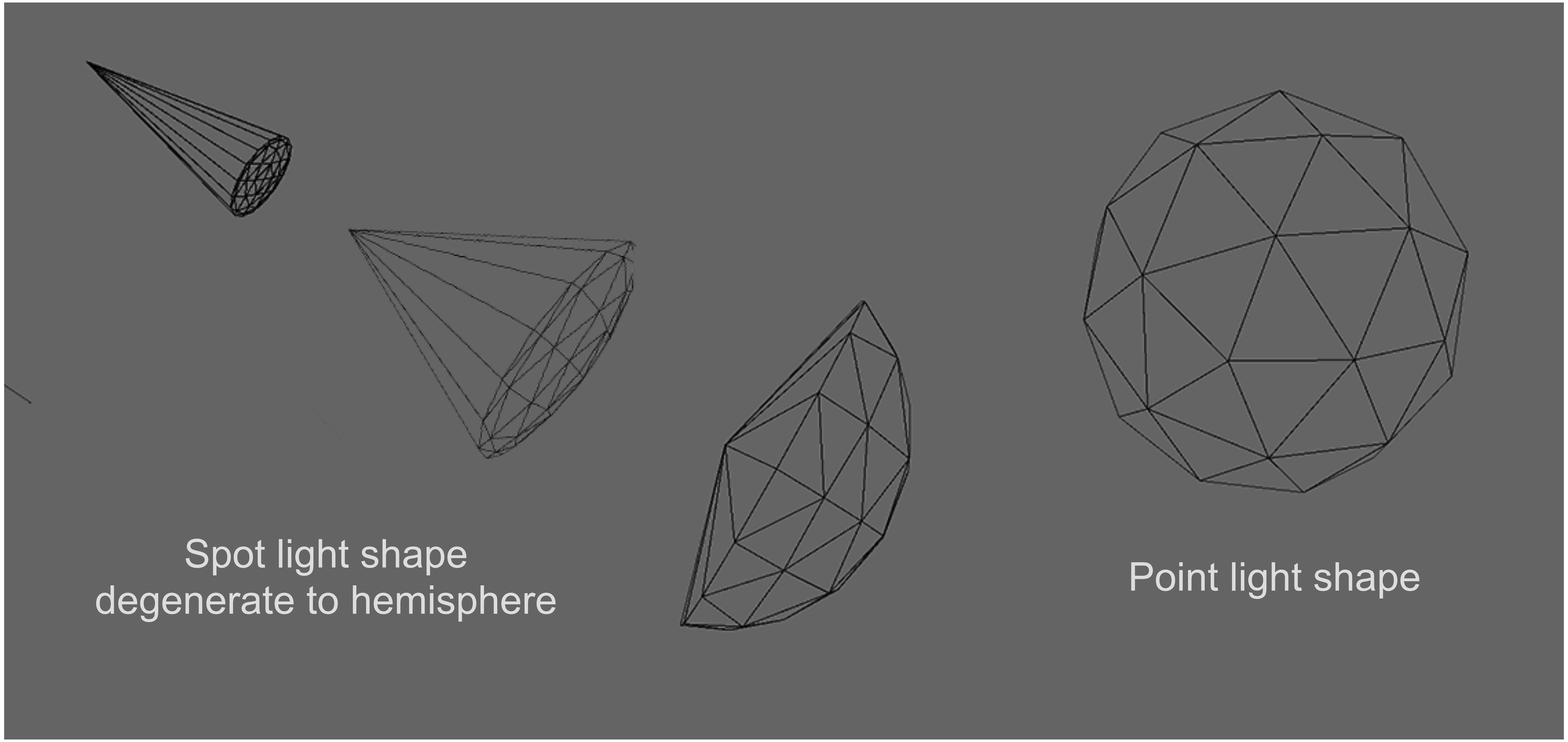
Light layers will requires an extra render-target, as well as motion vectors!

STENCIL-BASED DEFERRED SHADING



STENCIL-BASED DEFERRED SHADING

Geometric shapes for lights are tightly fitted



STENCIL-BASED DEFERRED SHADING

Render-Pass API

- Mobile GPUs with tiled-based architecture can avoid the cost refetching g-buffer textures for each light shape by caching them into on-tile memory.
- Depth must be rendered as extra colour render-target
- We have an implementation for it, but not ready for release yet
- Driver issues on older Vulkan drivers on some mobile devices
- Release pushed back to 2021

TILED-BASED DEFERRED SHADING

(NOT GPU ARCHITECTURE)

TILED-BASED DEFERRED SHADING (NOT GPU ARCHITECTURE)

SIGGRAPH THINK BEYOND 2020 S2020.SIGGRAPH.ORG

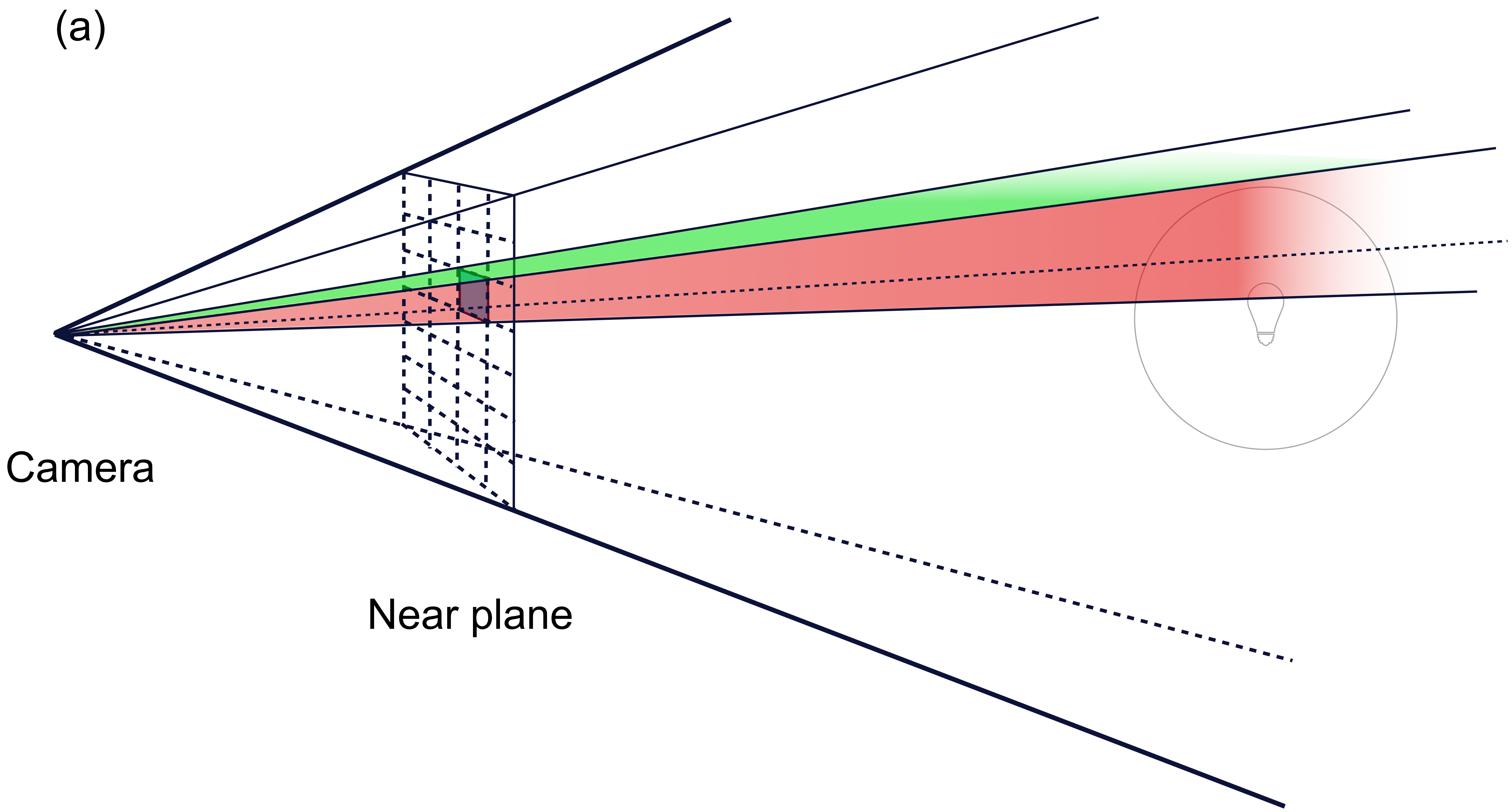
- Algorithm overview
  - Same name but different from GPU tiled-based architecture
  - Conceptually “re-use same principles” but without hardware support
  - Since done in “software”, can be further improved in many ways
  - Most implementations use compute shaders
  - Very popular with desktop GPUs

> It tries to minimise memory bandwidth by fetching g-buffer textures only once for shading

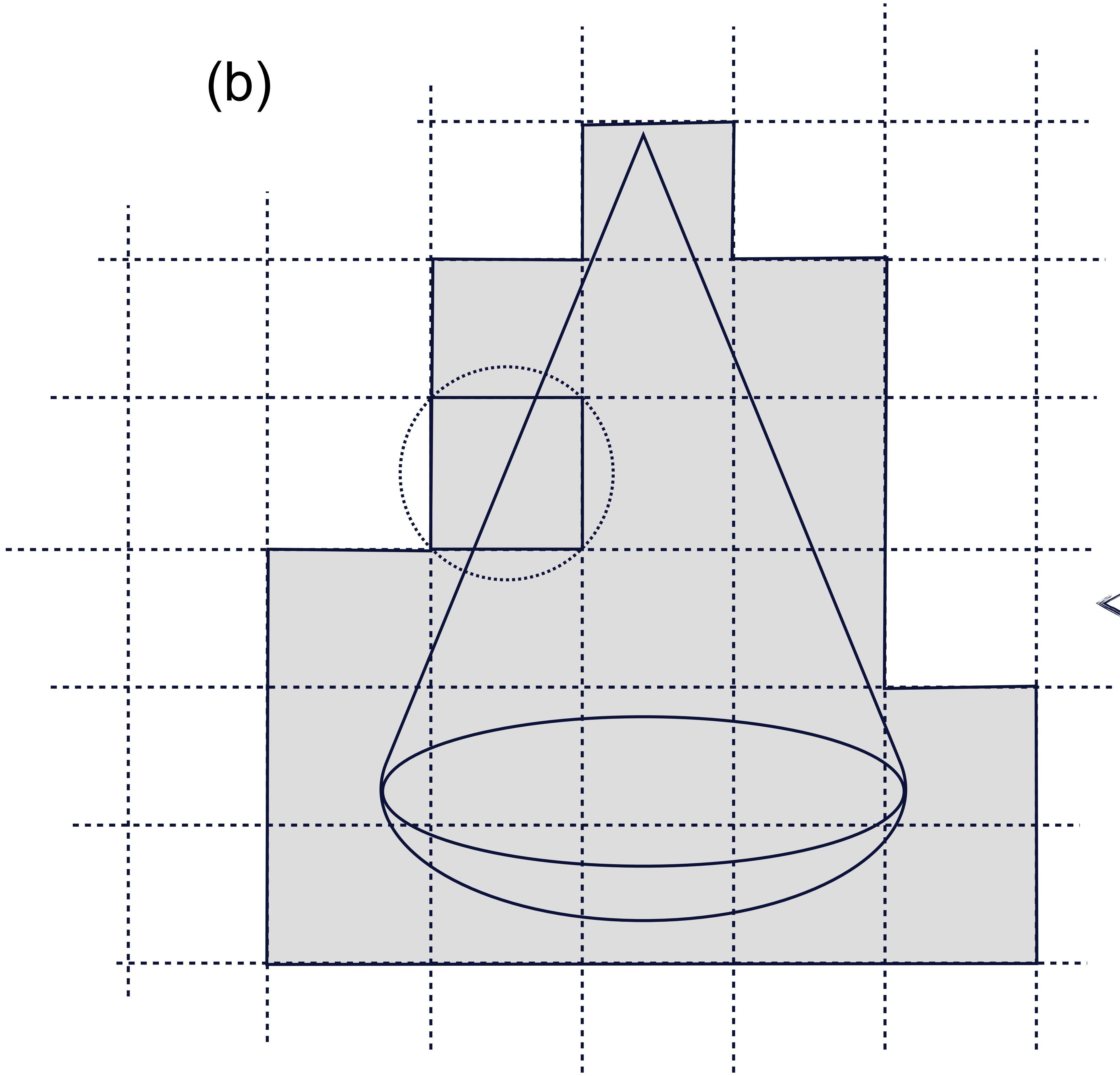
TILED-BASED DEFERRED SHADING (NOT GPU ARCHITECTURE)

- Algorithm overview (continue)
  - Render g-buffer as for stencil method (same layout)
  - Partition the screen into a grid of tiles (8x8 pixels or 16x16 pixels)
  - Use compute shaders to build for each tile the list of lights that affect it
    - (a) Each tile is extruded in 3D as a tight view frustum
    - (b) Check if a light intersects a given tile frustum
      (depth range of intersected parts must be known too)
    - (c) Check if a light intersects any scene geometry inside the tile

TILED-BASED DEFERRED SHADING (NOT GPU ARCHITECTURE)



TILED-BASED DEFERRED SHADING (NOT GPU ARCHITECTURE)



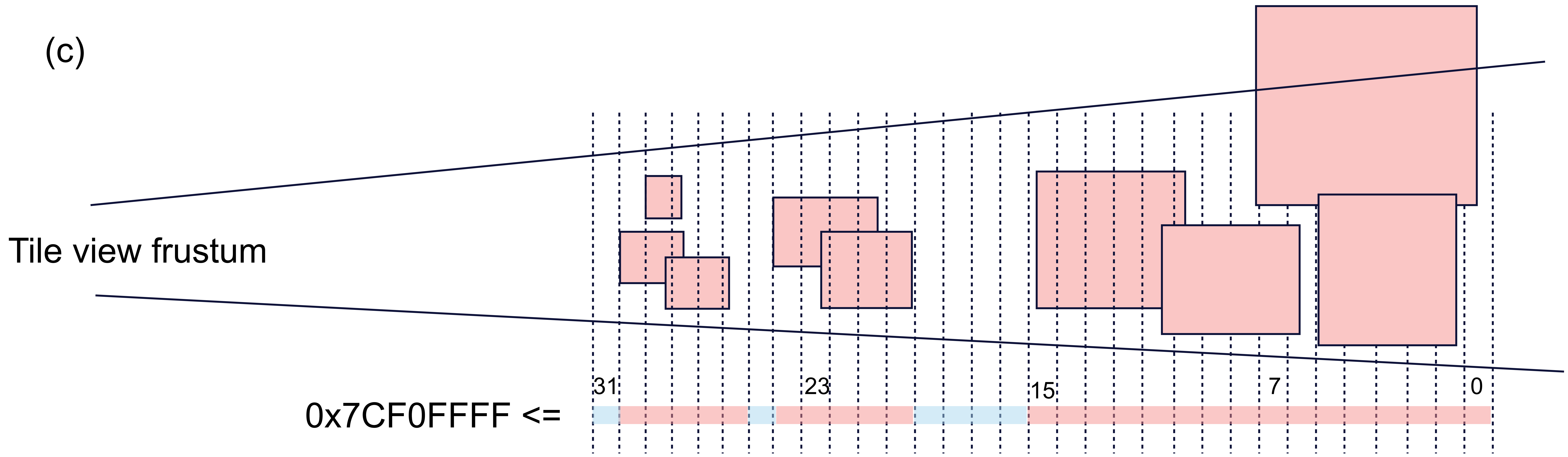
- Algorithm overview (continue)
  - (b) Check if a light intersects a given tile frustum
    - Intersection between light shapes and tight frustums can be tricky to calculate
    - naive intersection tests will trigger many false negatives
      - Especially for spot lights (capped rounded cone shape)

> Knowing the tile coordinate and the spot light parameters, how to calculate the depth range in view space?

TILED-BASED DEFERRED SHADING (NOT GPU ARCHITECTURE)

- Algorithm overview (continue)
  - (c) Check if a light intersects any scene geometry inside the tile
    - Scene geometry inside a tile is inferred by sampling the depth buffer
    - Simply a disjoint collection of depth values (16x16 pixels tile gives 256 depth values)
    - Can “compress” depth values into a bit mask (uint32_t)
    - Can also compress all lights depth ranges into a bit mask (uint32_t)
    - Intersection between lights and geometry becomes
      a bit-wise AND operation (2.5D culling)

TILED-BASED DEFERRED SHADING (NOT GPU ARCHITECTURE)



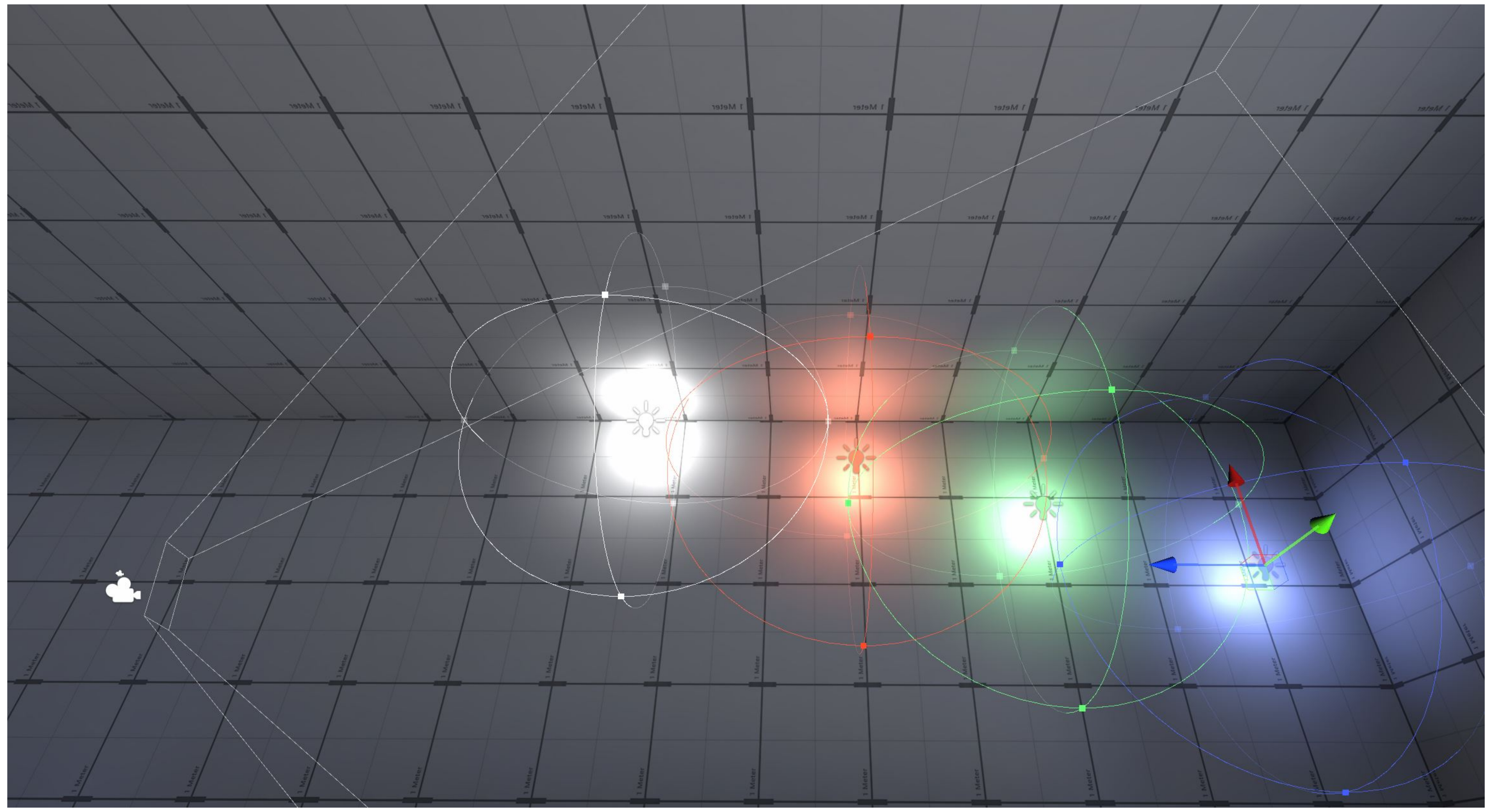
TILED-BASED DEFERRED SHADING (NOT GPU ARCHITECTURE)

- Our approach:
  - Cull lights against screen-space tiles without depth-information
    - Allow the culling to be done on GPU or CPU
  - Further trim light lists using scene depth-information just before shading
  - The tiles are not rendered using compute shaders but using fragment shaders
    - Draw instanced quads using an indirect draw call
  - A potential future path without compute shaders (very experimental)

TILED-BASED DEFERRED SHADING (NOT GPU HARDWARE)

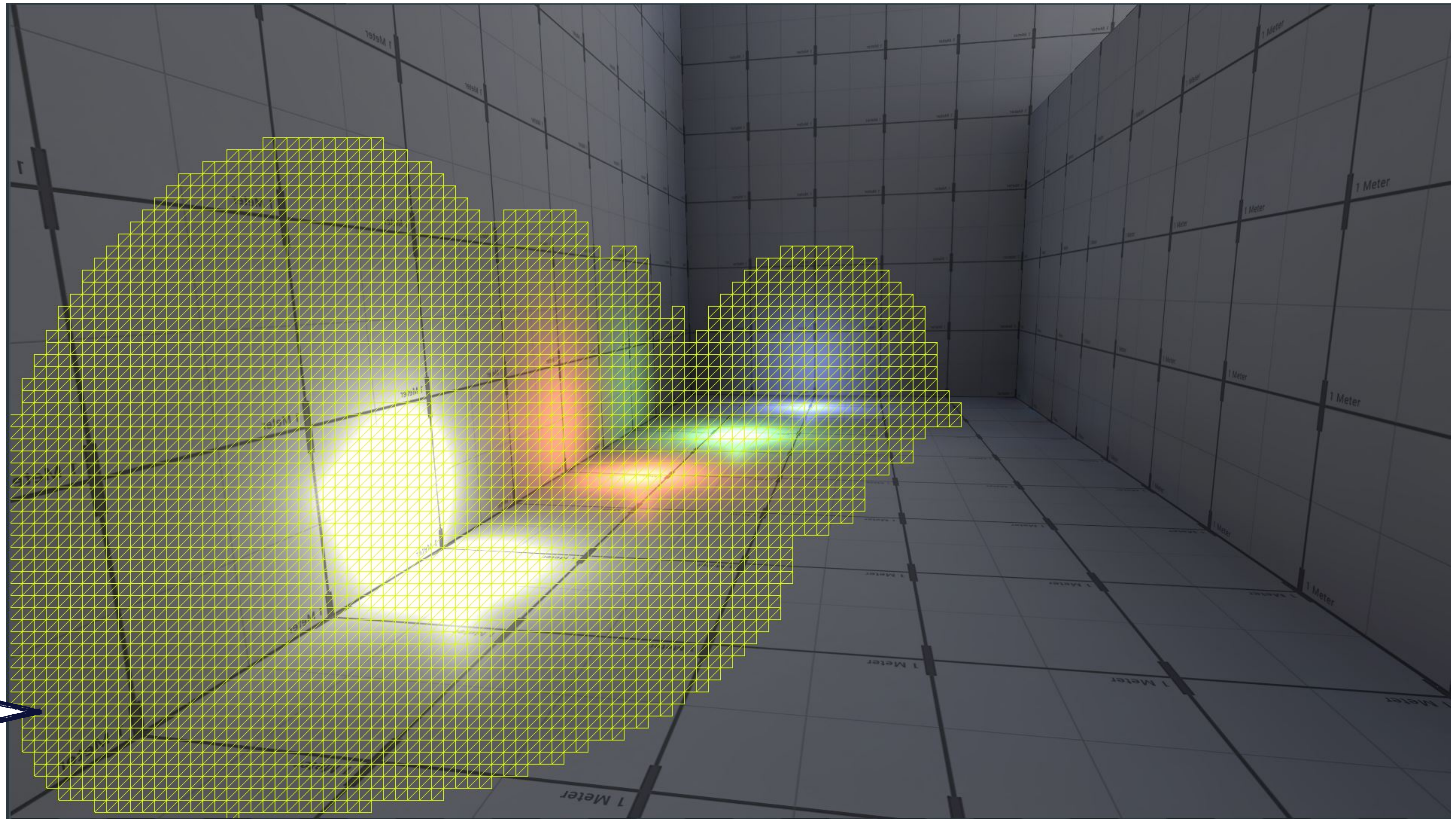
SIGGRAPH 2020 S2020.SIGGRAPH.ORG THINK BEYOND

Example tile rendering



View from above

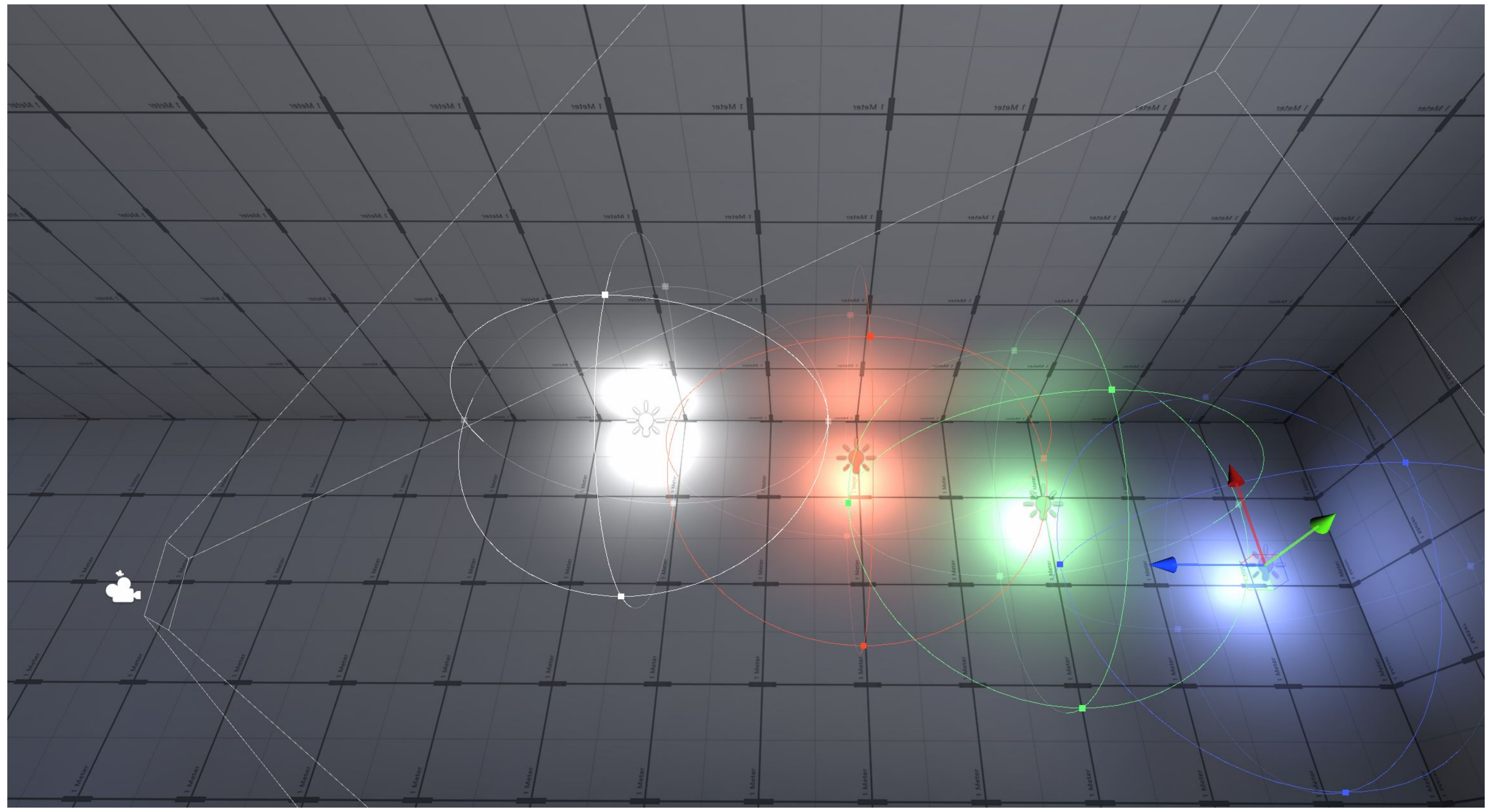
Tiles are rendered as triangles or quad topology if supported. We did not see performance degradation from double shading on common triangle edge.



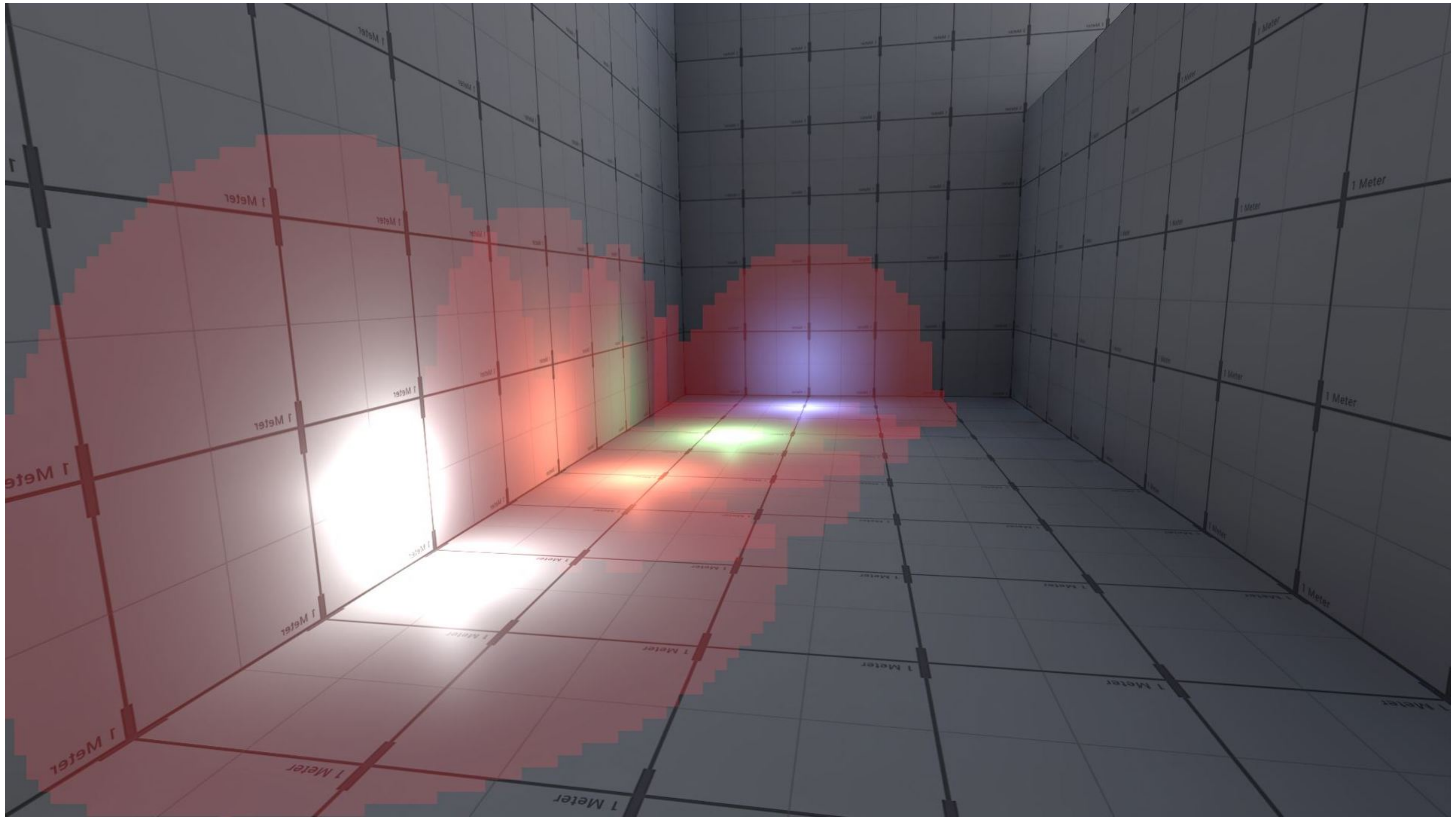
Generated tiles

TILED-BASED DEFERRED SHADING (NOT GPU HARDWARE)

Example tile rendering



View from above

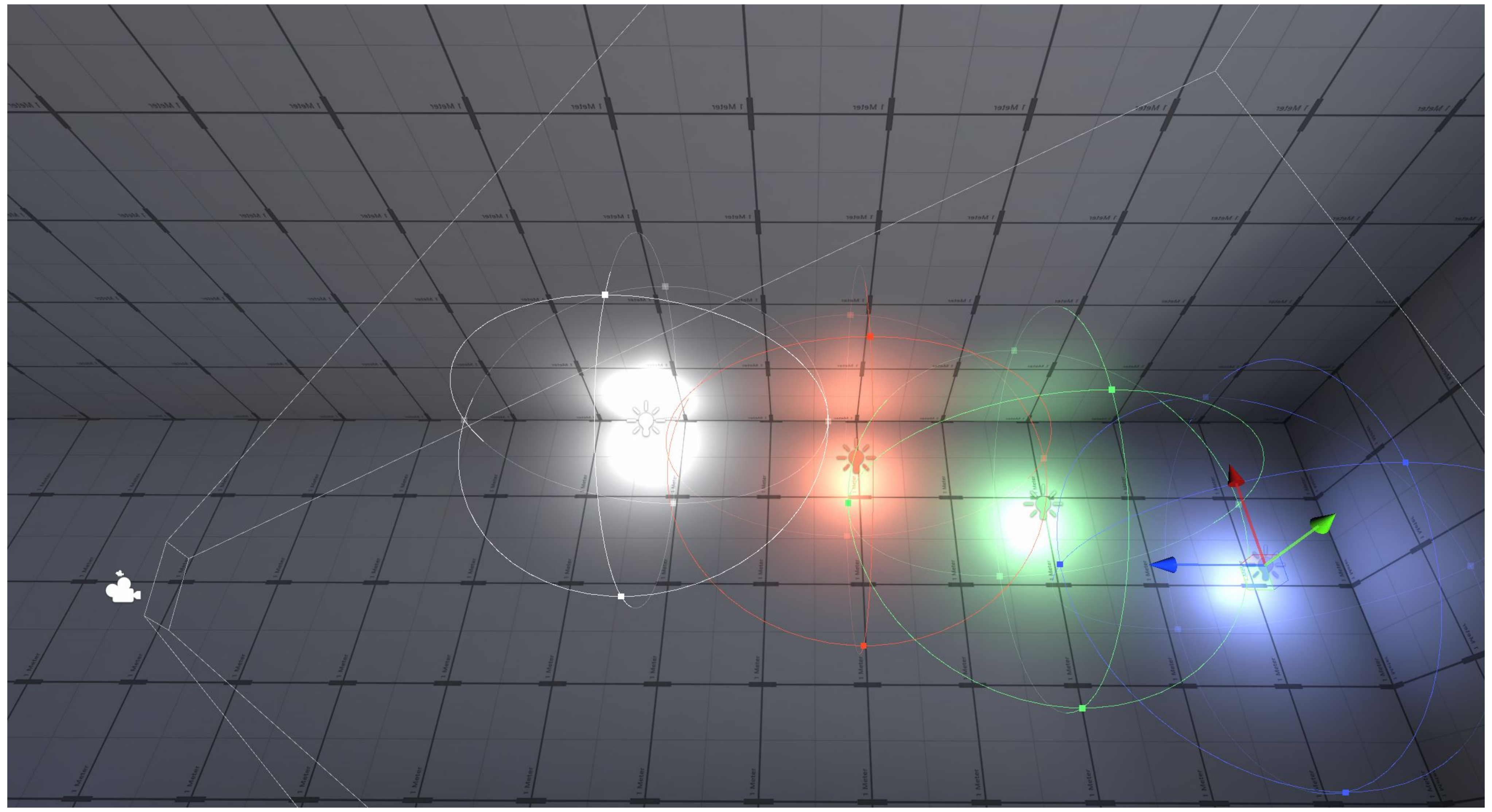


Heat map (number of lights per tile)

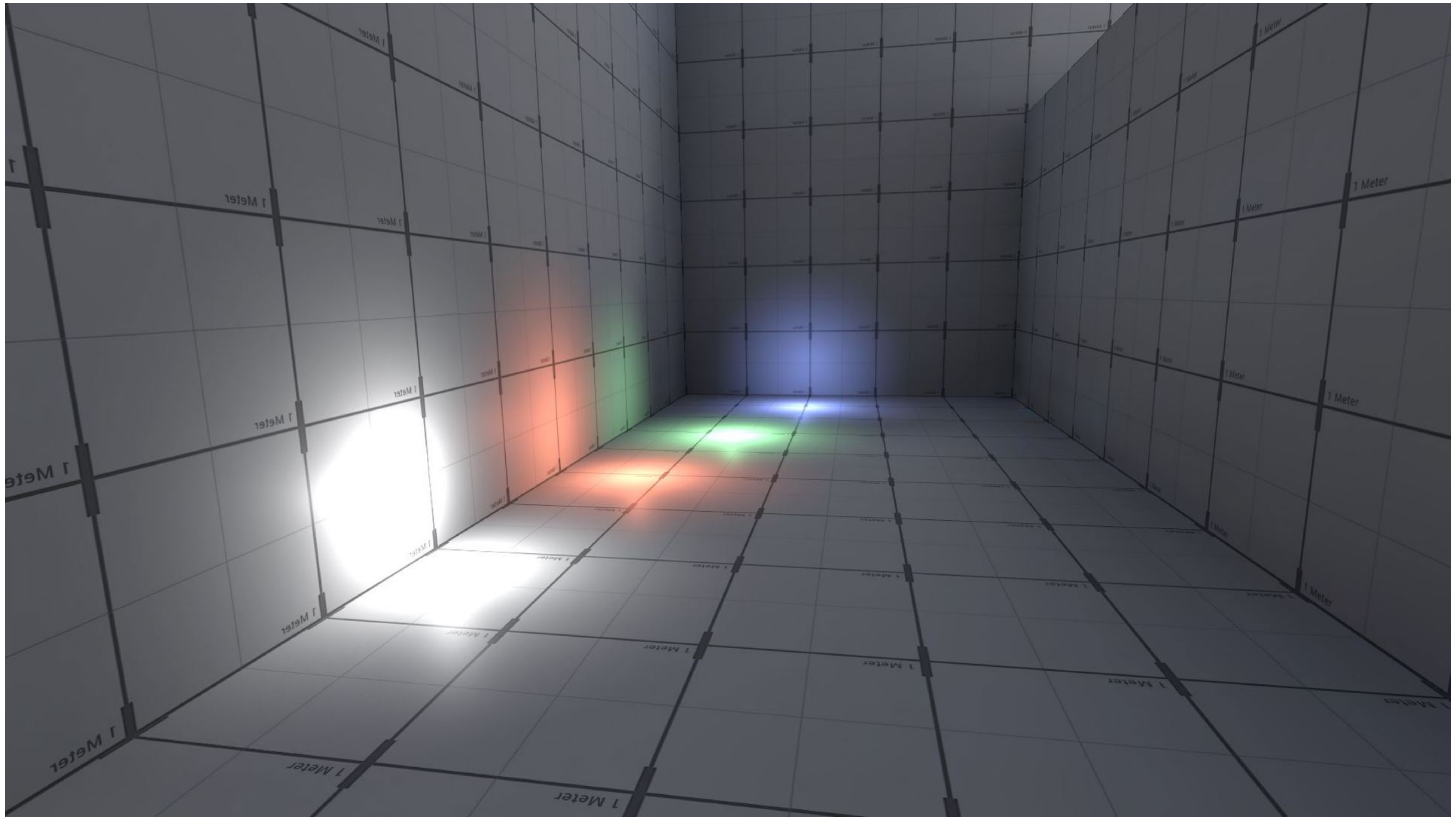
In this simple example, tiles have 1 or 2 lights per list, hard maximum is usually 32 or 64 lights per tiles.

TILED-BASED DEFERRED SHADING (NOT GPU HARDWARE)

Example tile rendering

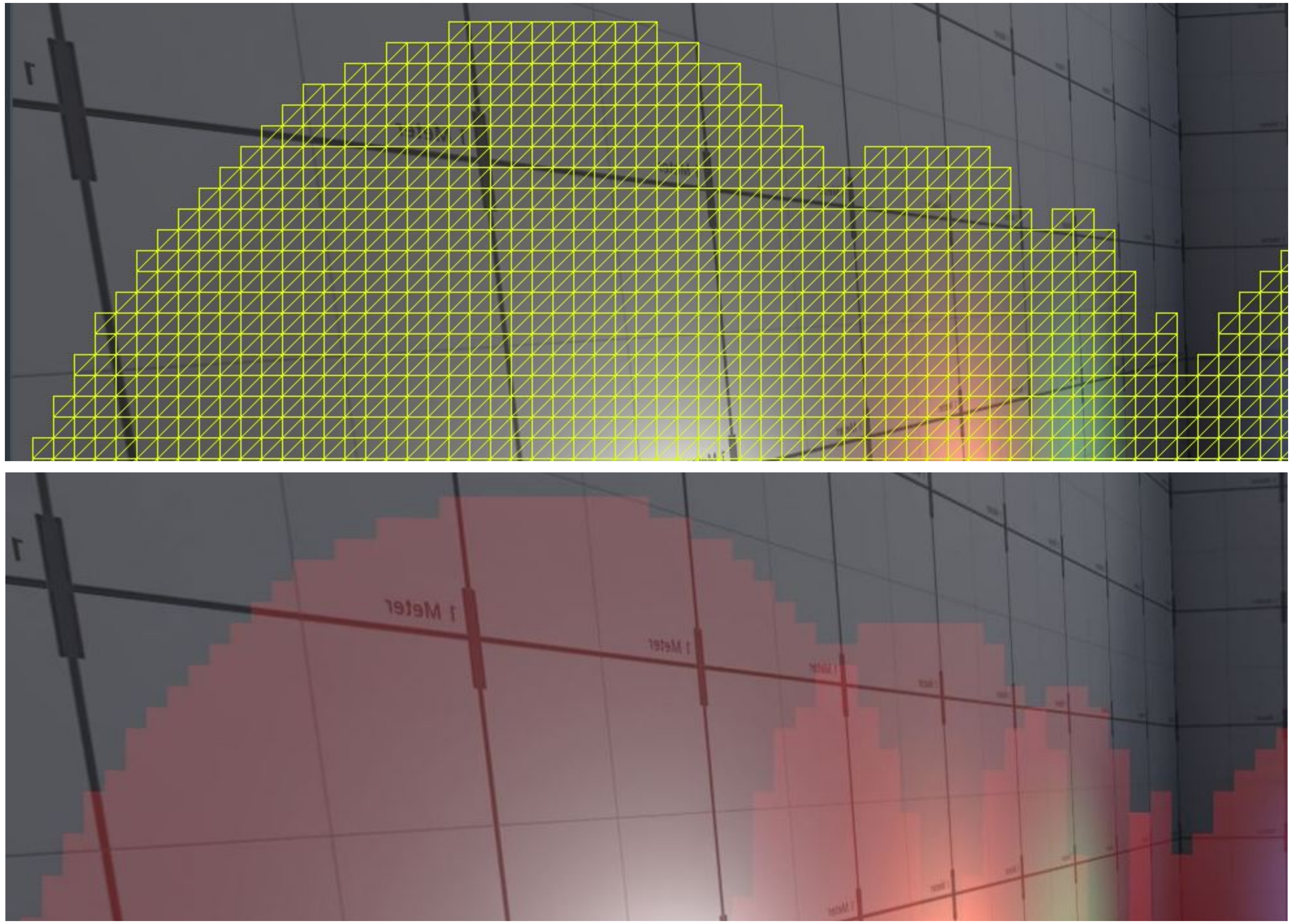


View from above



Final result

TILED-BASED DEFERRED SHADING (NOT GPU HARDWARE)

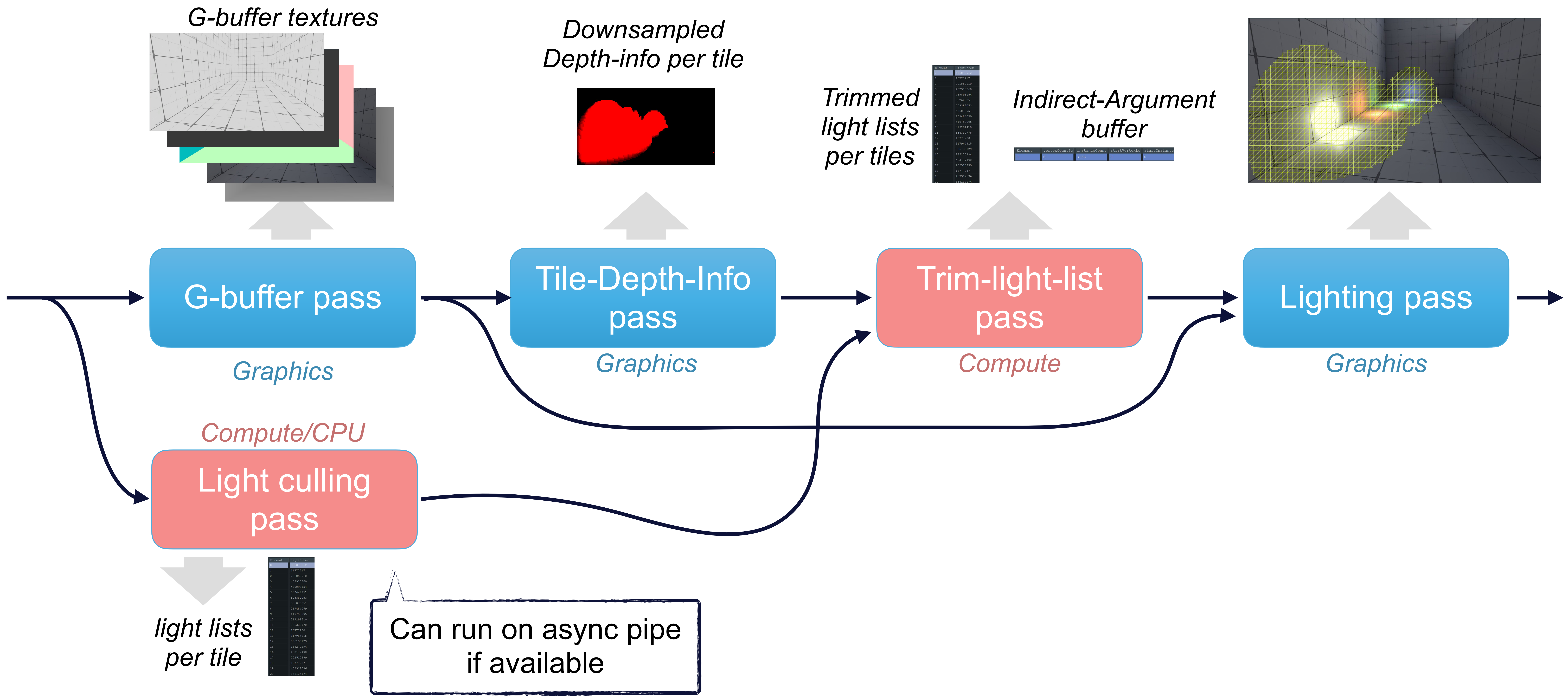


Visualising the tiles

The tile data structures are stored into StructuredBuffer/SSBO of uints.
No benefit in storing them in uniform buffer because no repeated access pattern.

Light list indices can be up to 32-64 lights.
In this simple case we only see up to 2 overlapping lights.

TILED-BASED DEFERRED SHADING (NOT GPU HARDWARE)



TILED-BASED DEFERRED SHADING (NOT GPU ARCHITECTURE)

- Pros
  - Very fast (on desktop GPUs)
  - Low memory bandwidth: g-buffer is fetched only once for shading
  - Light culling can be done on async compute pipes
  - Can further optimise shader used for shading tiles depending material types in the tile, …
  - Open up path to forward+ for forward passes (ex: transparent pass)
  - Open up path to Clustered shading (3D grid instead of 2D grid)

> Same advantage as GPU tiled-based architecture and Render pass API. But avoid repeated cost for g-buffer decoding …

> Material classification: allow skipping complex shader permutations per tile

> Forward-only pass is used for special material types (clear coat, hair, skin. …)

- Cons
  - Not fast (on mobile GPUs)
  - Reliable hardware support?

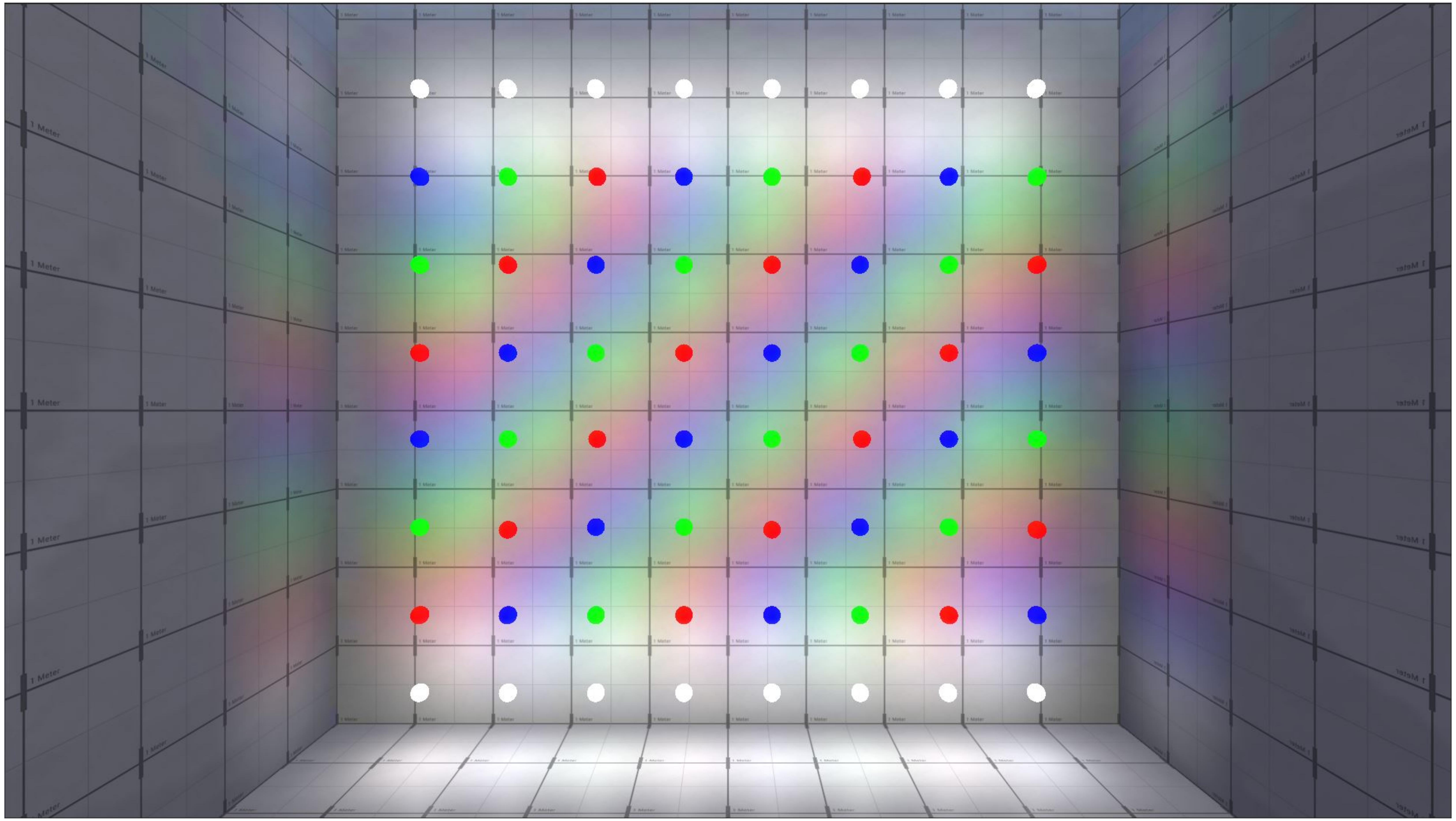
> Compute shaders may rely synchronisation mechanisms (memory barriers), atomics, group shared memory, wave-intrinsics, etc.

TILED-BASED DEFERRED SHADING (NOT GPU ARCHITECTURE)

- Experimental work: could we move the initial light culling pass on the CPU?
  - Possible because no dependencies with GPU at this stage
  - Not entirely novel, past work offloaded some processing on CPU too
  - With Unity Job System and Burst compiler we run 25%-50% slower than the compute shaders
  - Our Data layout not Structure-of-Array yet (SoA), so potential for more speed-up
  - Ultimately depends on CPU and GPU characteristics
  - More work needed!

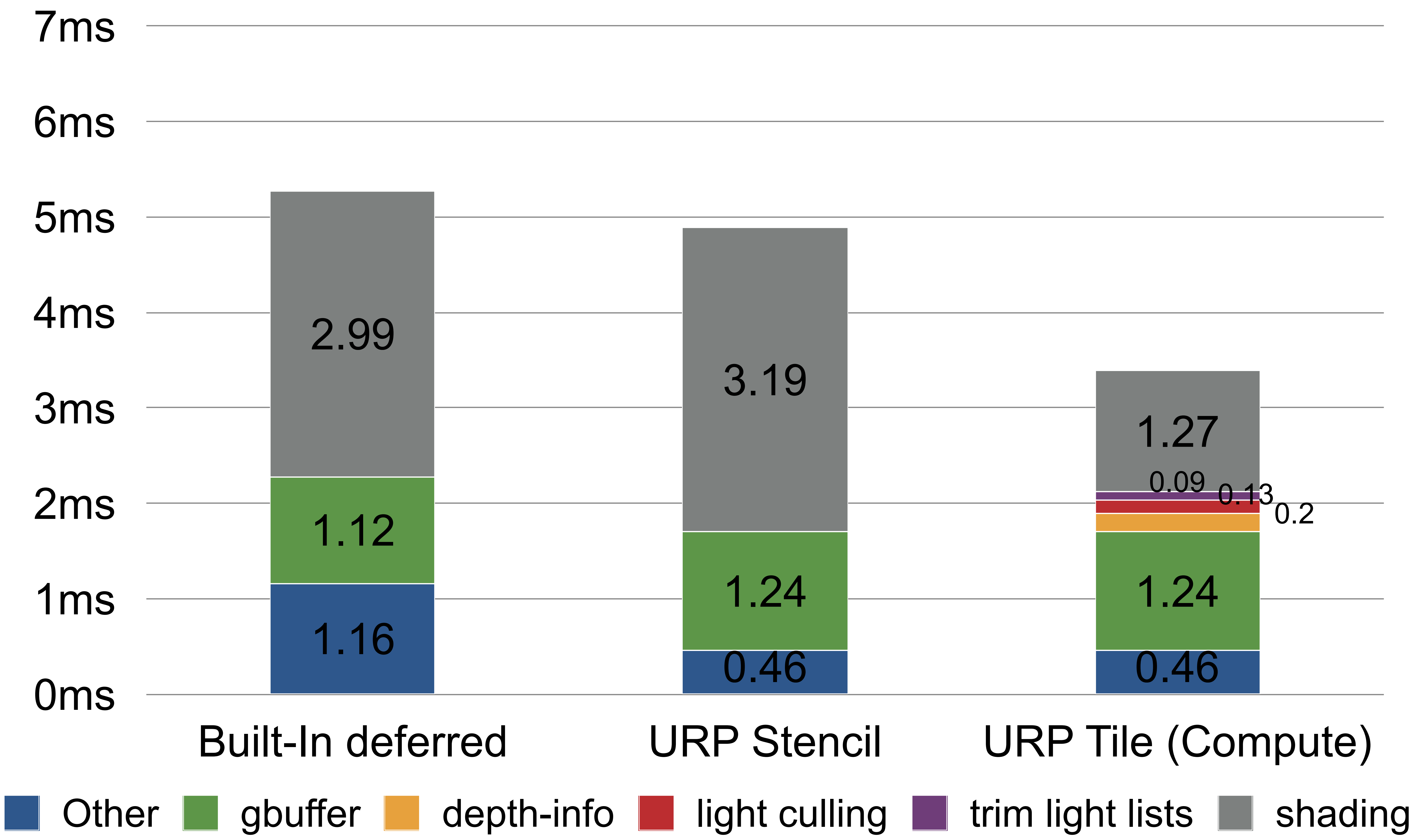
PERFORMANCE RESULTS

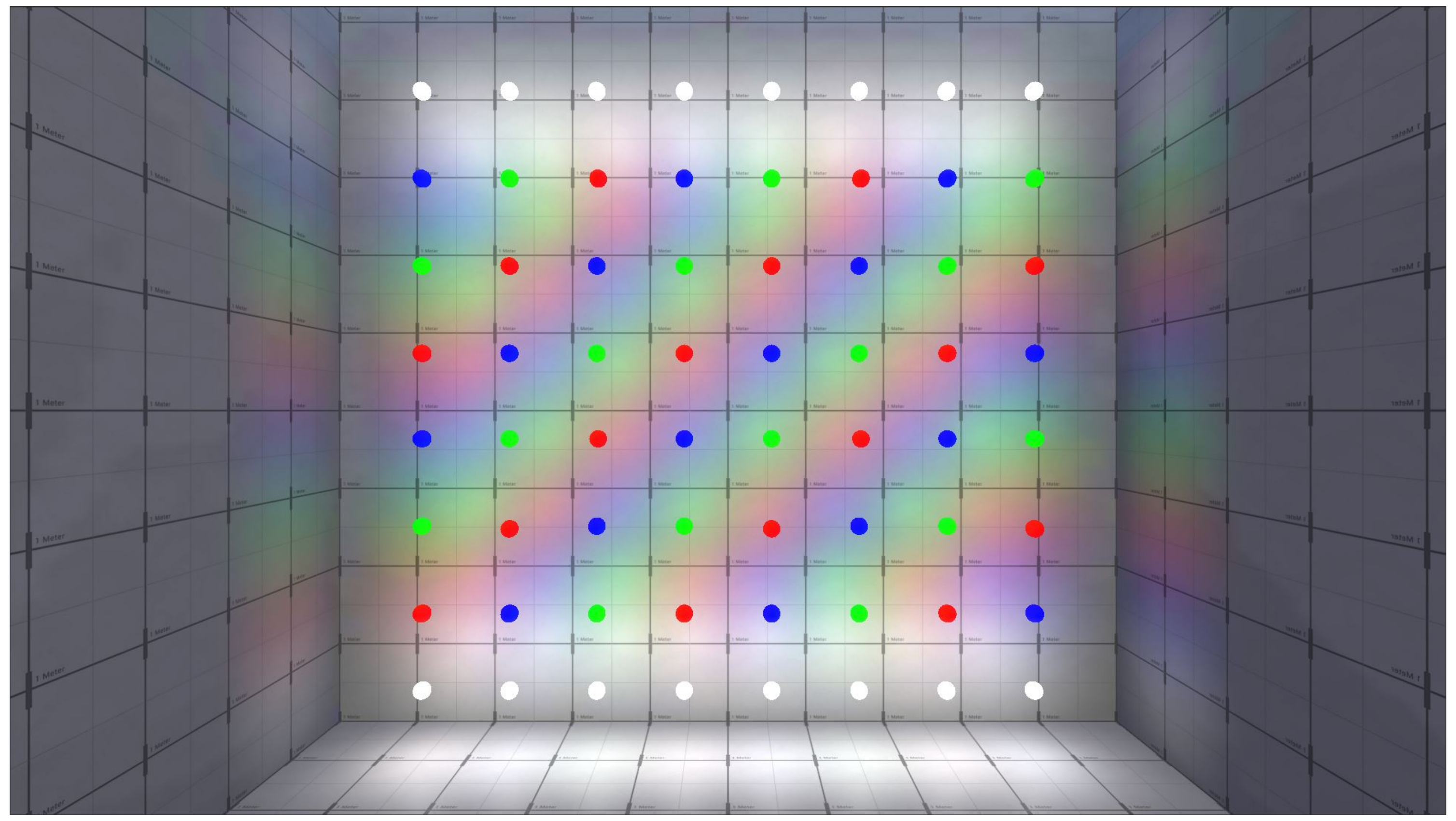
PERFORMANCE RESULTS



1 main directional light, 64 point lights, baked lighting

PERFORMANCE RESULTS

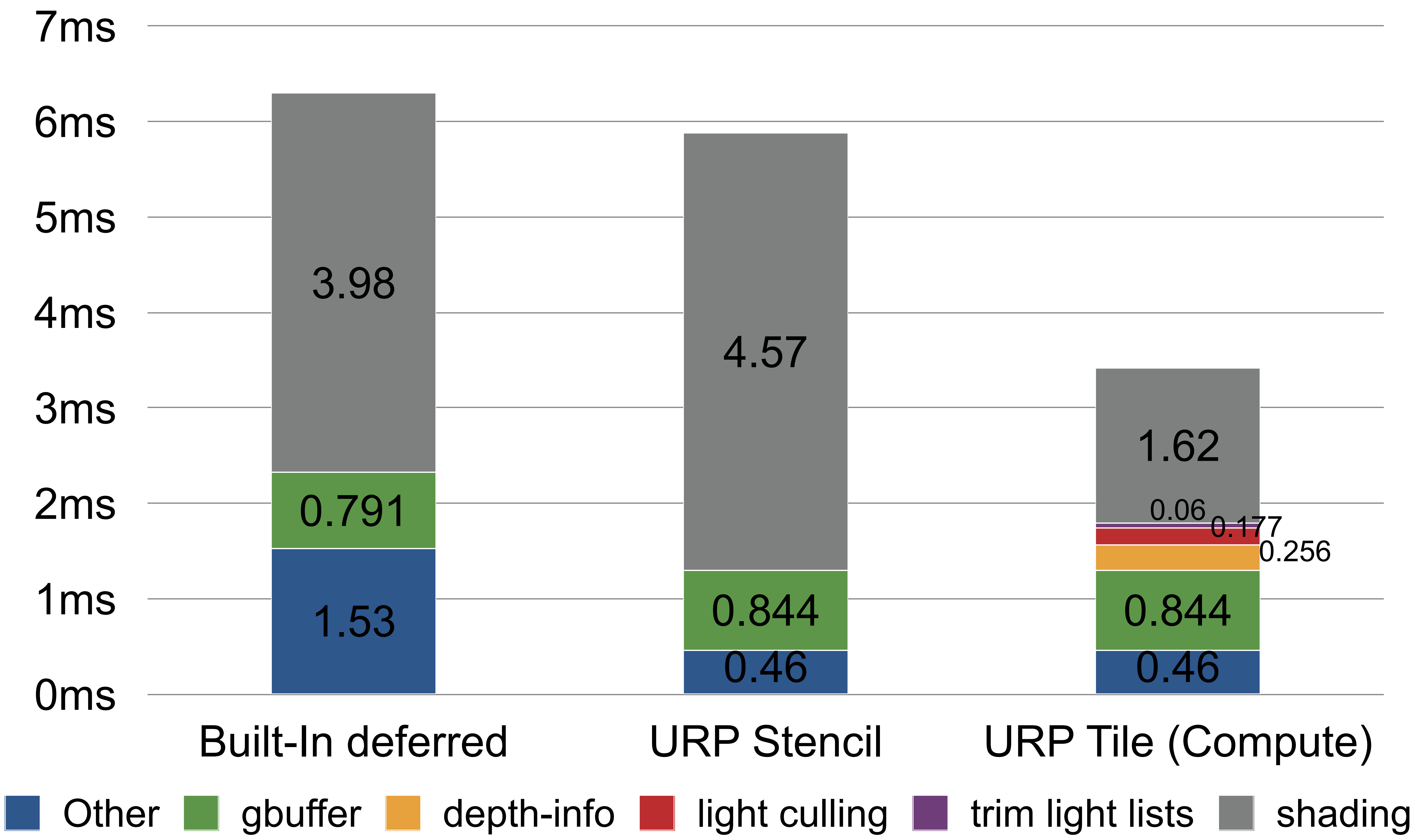


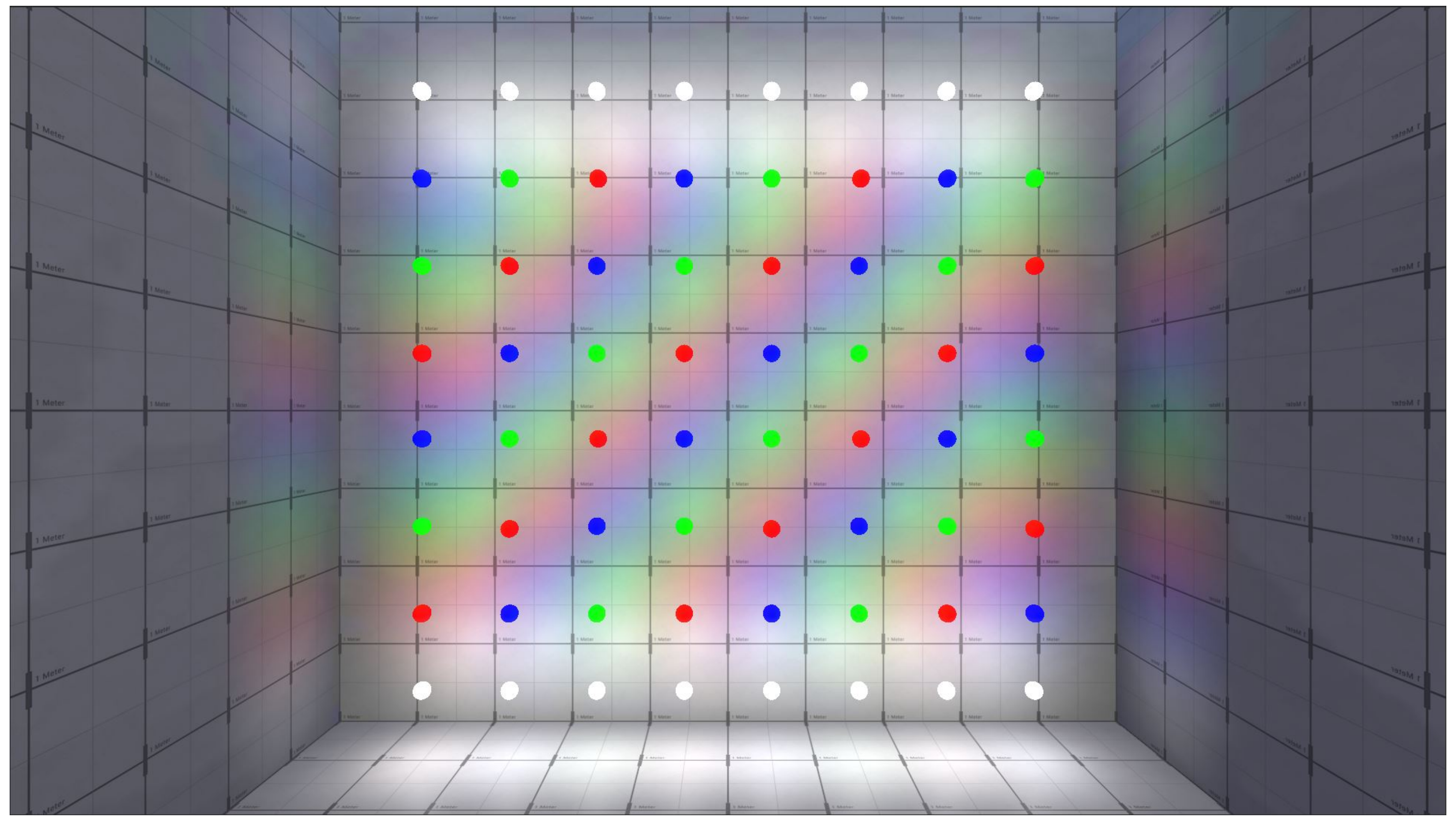


1 main directional light, 64 point lights, baked lighting

Nvidia Gfx Geforce 1050
1920x1080

PERFORMANCE RESULTS

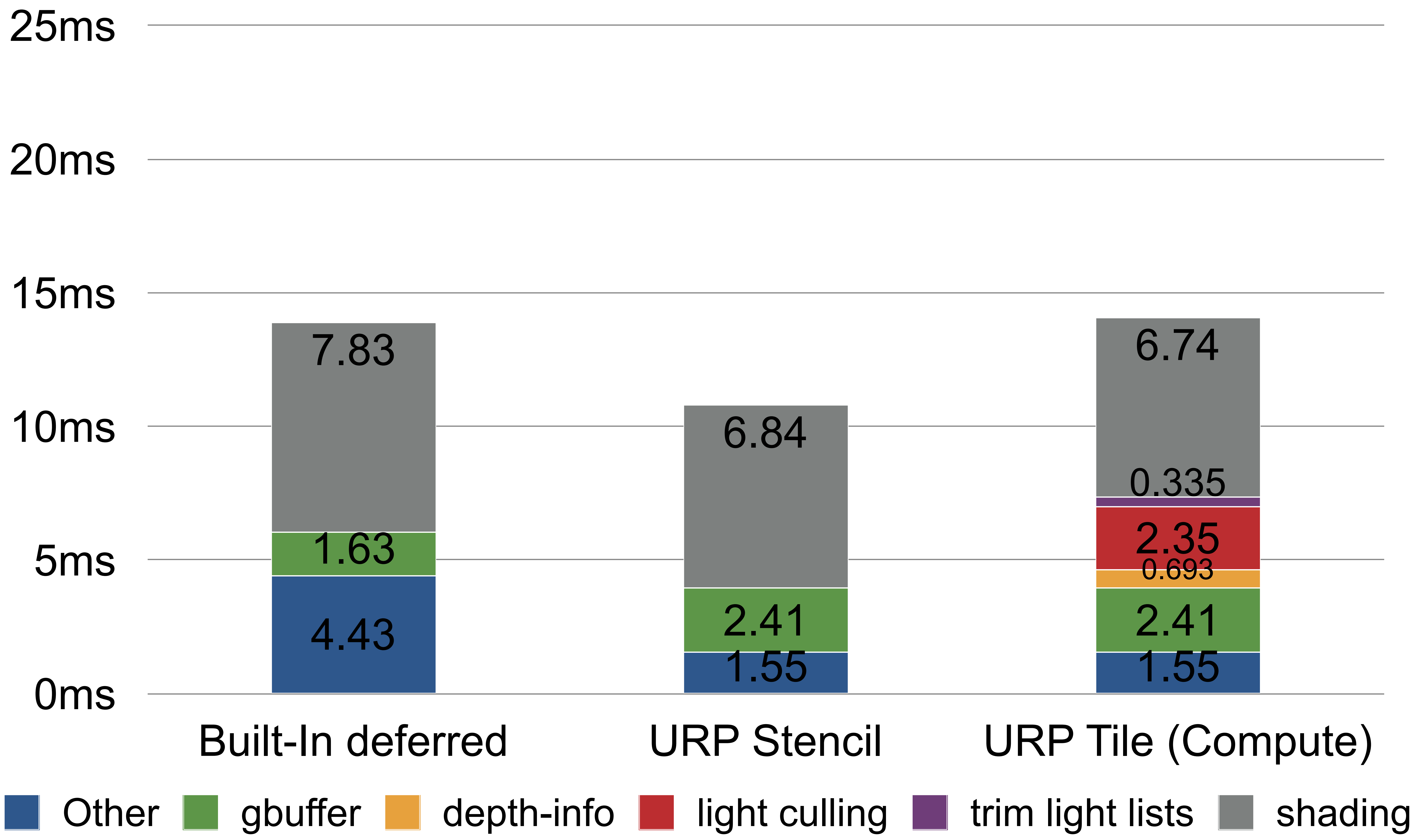


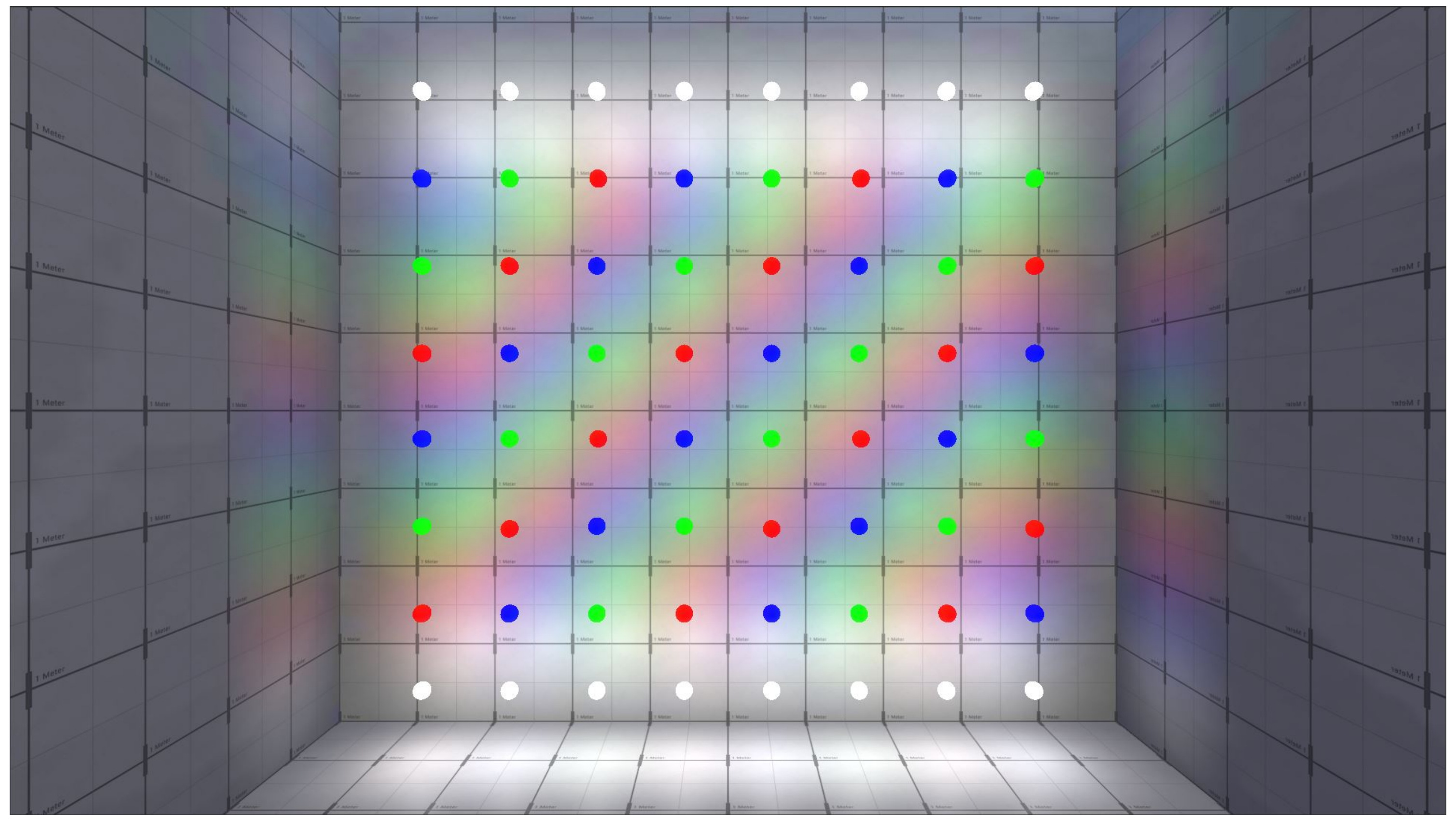


1 main directional light, 64 point lights, baked lighting

AMD Radeon Pro 460
1920x1080

PERFORMANCE RESULTS

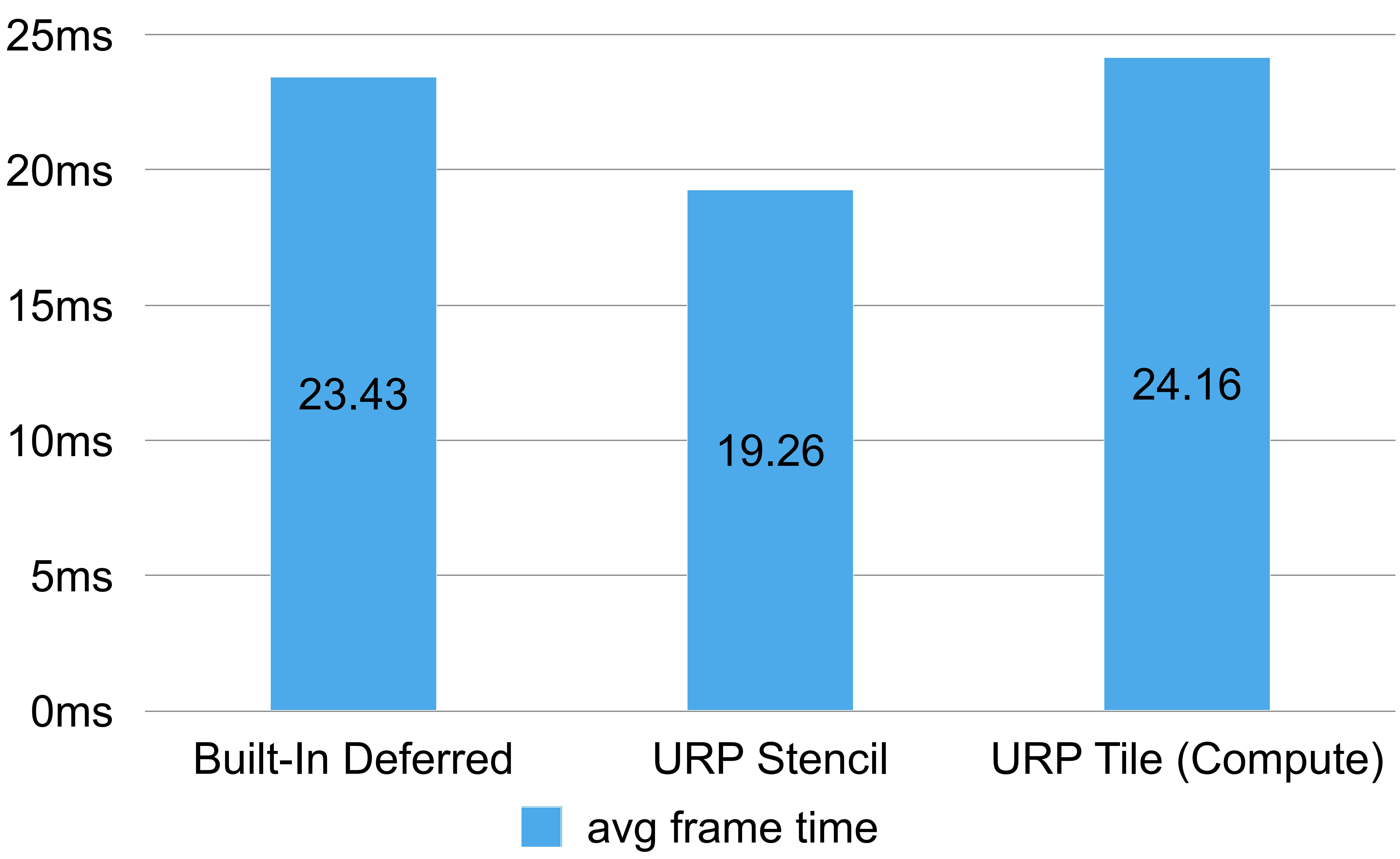


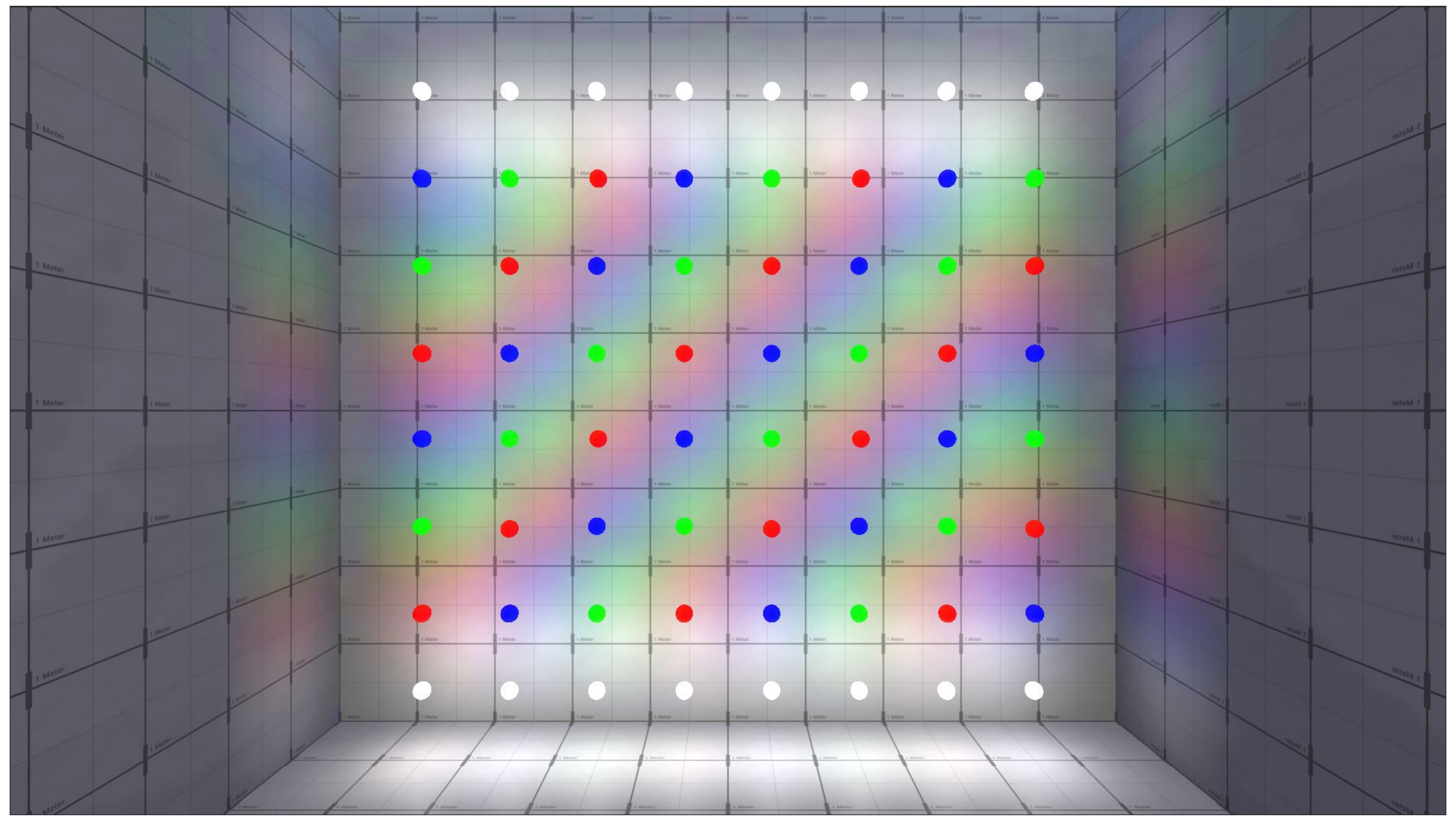


1 main directional light, 64 point lights, baked lighting

iPhone 8 A11 Bionic
1334x750

PERFORMANCE RESULTS





1 main directional light, 64 point lights, baked lighting

Galaxy S9 Mali-G72 MP18
2220x1080

MOVING FORWARD

MOVING FORWARD

- Stencil solution is mature and stable on all platforms, shipping soon …
- Tiled-based deferred solution need to be tested on more platforms because it relies on advanced compute shader features
- Experimental CPU solution is still in R&D
- Need to add async pipes support on platforms where it is available
- Clustered shading implementation
- Forward+ for forward passes

REFERENCES

- https://www.gamedevs.org/uploads/rendering-in-battlefield3.pdf
- http://twvideo01.ubm-us.net/o1/vault/gdc2011/slides/Christina_Coffin_Programming_SPU_Based_Deferred.pdf
- http://www.humus.name/Articles/PracticalClusteredShading.pdf
- https://wickedengine.net/2018/01/10/optimizing-tile-based-light-culling/
- https://bartwronski.com/2017/04/13/cull-that-cone/
- https://www.iquilezles.org/www/articles/ellipses/ellipses.htm
- http://lousodrome.net/blog/light/2017/01/03/intersection-of-a-ray-and-a-cone/

QUESTIONS?