Deferred Rendering in Killzone 2

Michal Valient
Senior Programmer, Guerrilla

Talk Outline

- Forward & Deferred Rendering Overview
- G-Buffer Layout
- Shader Creation
- Deferred Rendering in Detail
  - Rendering Passes
  - Light and Shadows
  - Post-Processing
- SPU Usage / Architecture

Forward & Deferred Rendering Overview

Forward Rendering – Single Pass

- For each object
  - Find all lights affecting object
  - Render all lighting and material in a single shader
- Shader combinations explosion
  - Shader for each material vs. light setup combination
- All shadow maps have to be in memory
- Wasted shader cycles
  - Invisible surfaces / overdraw
  - Triangles outside light influence

Forward Rendering - Multi-Pass

- For each light
  - For each object
  - Add lighting from single light to frame buffer
- Shader for each material and light type
- Wasted shader cycles
  - Invisible surfaces / overdraw
  - Triangles outside light influence
  - Lots of repeated work
    - Full vertex shaders, texture filtering

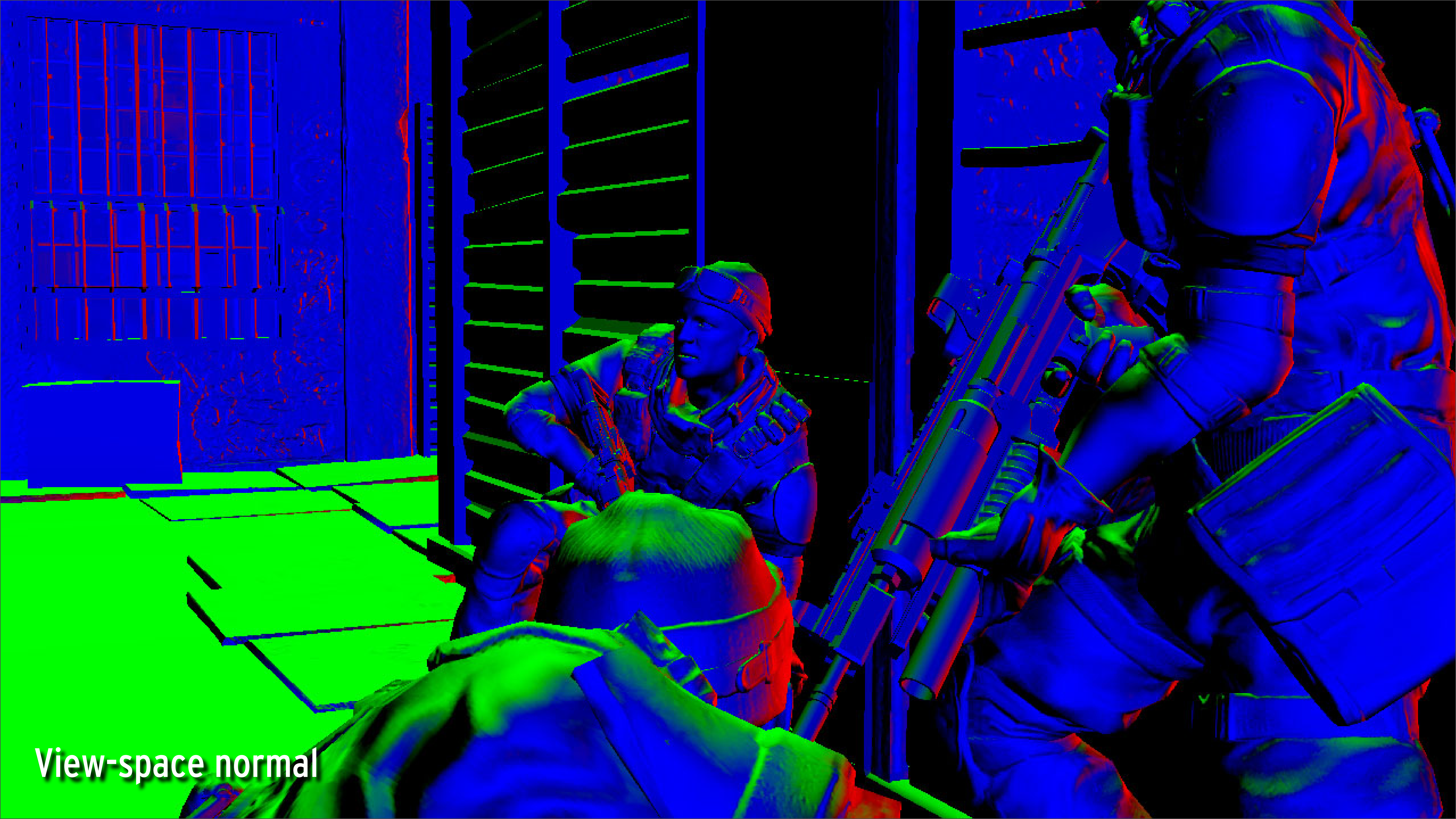
Deferred Rendering

- For each object
  - Render surface properties into the G-Buffer
- For each light and lit pixel
  - Use G-Buffer to compute lighting
  - Add result to frame buffer
- Simpler shaders
- Scales well with number of lit pixels
- Does not handle transparent objects

G-Buffer Layout


Target Image


Depth


View-space normal

Specular intensity


Specular roughness / Power

Screen-space 2D motion vector


Albedo (texture colour)


Deferred composition

Image with post-processing (depth of field, bloom, motion blur, colorize, ILR)

G-Buffer : Our approach

<table>
<tr><th>R8</th><th>G8</th><th>B8</th><th>A8</th><th></th></tr>
<tr><td colspan="3">Depth 24bpp</td><td>Stencil</td><td>DS</td></tr>
<tr><td colspan="3">Lighting Accumulation RGB</td><td>Intensity</td><td>RT0</td></tr>
<tr><td colspan="2">Normal X (FP16)</td><td colspan="2">Normal Y (FP16)</td><td>RT1</td></tr>
<tr><td colspan="2">Motion Vectors XY</td><td>Spec-Power</td><td>Spec-Intensity</td><td>RT2</td></tr>
<tr><td colspan="3">Diffuse Albedo RGB</td><td>Sun-Occlusion</td><td>RT3</td></tr>
</table>

- MRT - 4xRGBA8 + 24D8S (approx 36 MB)
- 720p with Quincunx MSAA
- Position computed from depth buffer and pixel coordinates

G-Buffer : Our approach

R8	G8	B8	A8	
Depth 24bpp			Stencil	DS
Lighting Accumulation RGB			Intensity	RT0
Normal X (FP16)		Normal Y (FP16)		RT1
Motion Vectors XY		Spec-Power	Spec-Intensity	RT2
Diffuse Albedo RGB			Sun-Occlusion	RT3

- Lighting accumulation – output buffer
- Intensity – luminance of Lighting accumulation
  - Scaled to range [0…2]
- Normal.z = sqrt(1.0f - Normal.x^2 - Normal.y^2)

G-Buffer : Our approach

R8	G8	B8	A8	
Depth 24bpp			Stencil	DS
Lighting Accumulation RGB			Intensity	RT0
Normal X (FP16)		Normal Y (FP16)		RT1
Motion Vectors XY		Spec-Power	Spec-Intensity	RT2
Diffuse Albedo RGB			Sun-Occlusion	RT3

- Motion vectors – screen space
- Specular power - stored as log2(original)/10.5
  - High range and still high precision for low shininess
- Sun Occlusion - pre-rendered static sun shadows
  - Mixed with real-time sun shadow for higher quality

G-Buffer Analysis

- Pros:
  - Highly packed data structure
  - Many extra attributes
  - Allows MSAA with hardware support
- Cons:
  - Limited output precision and dynamic range
    - Lighting accumulation in gamma space
    - Can use different color space (LogLuv)
  - Attribute packing and unpacking overhead

Deferred Rendering Passes

Geometry Pass

- Fill the G-Buffer with all geometry (static, skinned, etc.)
  - Write depth, motion, specular, etc. properties
- Initialize light accumulation buffer with pre-baked light
  - Ambient, Incandescence, Constant specular
  - Lightmaps on static geometry
    - YUV color space, S3TC5 with Y in Alpha
    - Sun occlusion in B channel
    - Dynamic range [0..2]
  - Image based lighting on dynamic geometry

Image Based Lighting

- Artist placed light probes
  - Arbitrary location and density
  - Sampled and stored as 2nd order spherical harmonics
- Updated per frame for each object
  - Blend four closest SHs based on distance
  - Rotate into view space
  - Encode into 8x8 envmap IBL texture
  - Dynamic range [0..2]
  - Generated on SPUs in parallel to other rendering tasks


Scene lighting

Decals and Weapon Passes

- Primitives updating subset of the G-Buffer
  - Bullet holes, posters, cracks, stains
  - Reuse lighting of underlying surface
  - Blend with albedo buffer
  - Use G-Buffer Intensity channel to fix accumulation
  - Same principle as particles with motion blur

- Separate weapon pass with different projection
  - Different near plane
  - Rendered into first 5% of depth buffer range
  - Still reacts to lights and post-processing


05

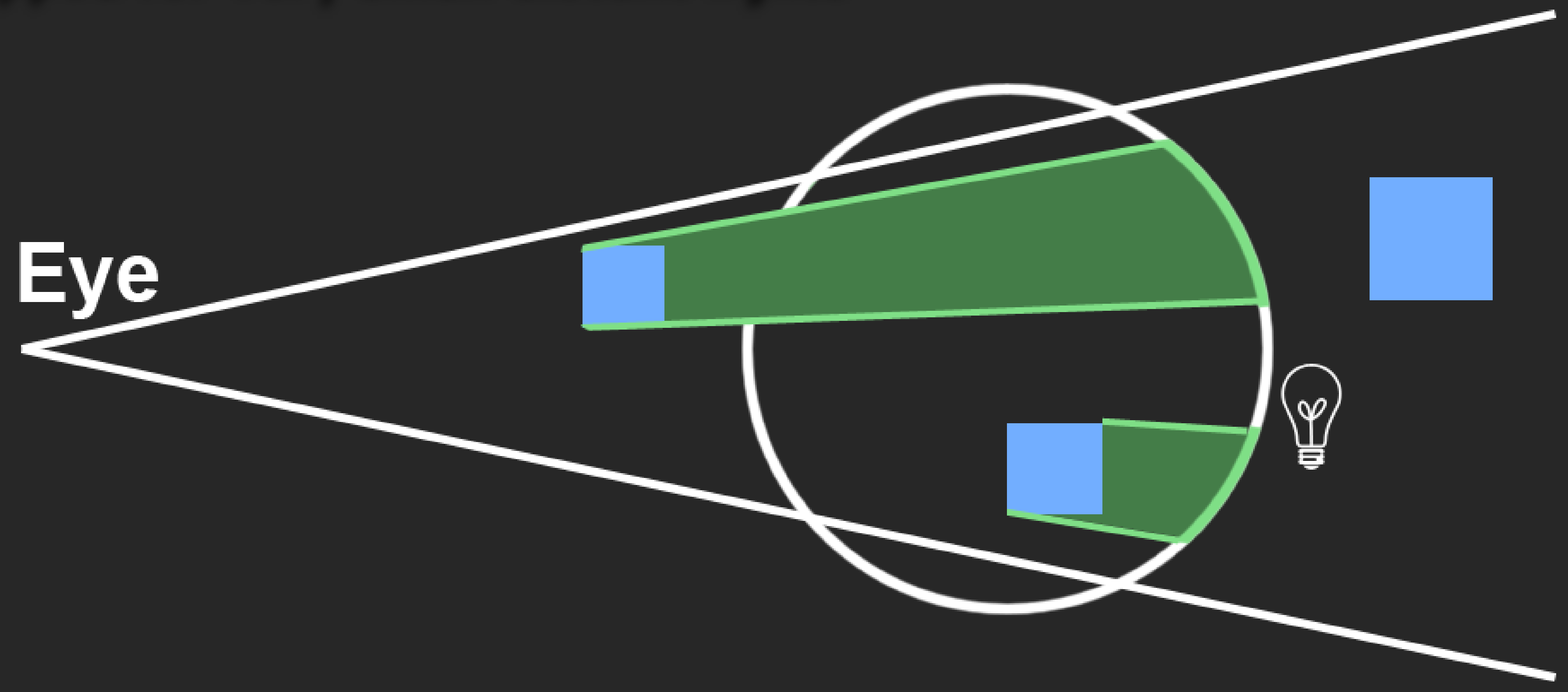
Light Accumulation Pass

- Light is rendered as convex geometry
  - Point light – sphere
  - Spot light – cone
  - Sun – full-screen quad

- For each light…
  - Find and mark visible lit pixels
  - If light contributes to screen
    - Render shadow map
    - Shade lit pixels and add to framebuffer

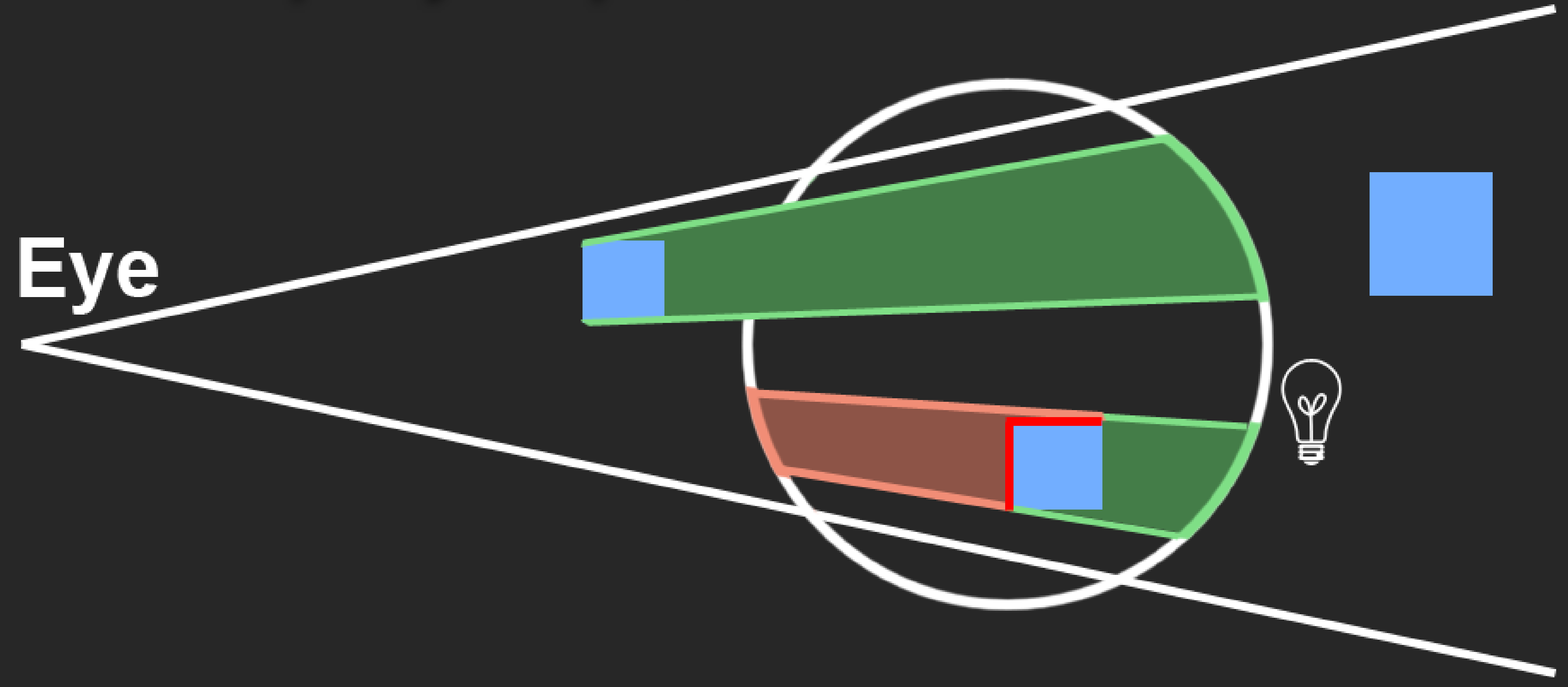
Determine Lit Pixels

- Marks pixels in front of the far light boundary
  - Render back-faces of light volume
  - Depth test GREATER-EQUAL
  - Write to stencil on depth pass
  - Skipped for very small distant lights



Determine Lit Pixels

- Find amount of lit pixels inside the volume
  - Start pixel query
  - Render front faces of light volume
  - Depth test LESS-EQUAL
  - Don't write anything – only EQUAL stencil test



Render Shadow Map

- Enable conditional rendering
  - Based on query results from previous stage
  - GPU skips rendering for invisible lights
- Max 1024x1024xD16 shadow map
  - Fast and with hardware filtering support
  - Single map reused for all lights
- Skip small objects
  - Small in shadow map and on screen
  - Artist defined thresholds for lights and objects

Shade Lit Pixels

- Render front-faces of light volume
  - Depth test - LESS-EQUAL
  - Stencil test - EQUAL
  - Runs only on marked pixels inside light
- Compute light equation
  - Read and unpack G-Buffer attributes
  - Calculate Light vector, Color, Distance Attenuation
  - Perform shadow map filtering
- Add Phong lighting to frame buffer

Light Optimization

- Determine light size on the screen
  - Approximation - angular size of light volume
- If light is "very small"
  - Don't do any stencil marking
  - Switch to non-shadow casting type
- Shadows fade-out range
  - Artist defined light sizes at which:
    - Shadows start to fade out
    - Switch to non-shadow casting light

Sun Rendering

- Full screen quad
- Stencil mark potentially lit pixels
  - Use only sun occlusion from G-Buffer
- Run final shader on marked pixels
  - Approx. 50% of pixels skipped thanks 1st pass
    - Also skybox/background
  - Simple directional light model
  - Shadow = min(RealTimeShadow, Occlusion)

Sun - Real-Time Shadows

- Cascade shadow maps
  - Provide more shadow detail where required
  - Divide view frustum into several areas
    - Split along view distance
    - Split distances defined by artist
  - Render shadow map for each area
    - Max 4 cascades
    - Max 512x512 pixels each in single texture
  - Easy to address cascade in final render

Sun - Real-Time Shadows

- Issue: Shadow shimmering
  - Light cascade frustums follow camera
  - Sub pixel changes in shadow map
- Solution!
  - Don't rotate shadow map cascade
    - Make bounding sphere of cascade frustum
    - Use it to generate cascade light matrix
  - Remove sub-pixel movements
    - Project world origin onto shadow map
    - Use it to round light matrix to nearest shadow pixel corner

Sun - Colored shadow Cascades - Unstable shadow artifacts

MSAA Lighting Details

- Run light shader at pixel resolution
  - Read G-Buffer for both pixel samples
  - Compute lighting for both samples
  - Average results and add to frame buffer
- Optimization in shadow map filtering
  - Max 12 shadow taps per pixel
  - Alternate taps between both samples
  - Half quality on edges, full quality elsewhere
  - Performance equal to non-MSAA case

Forward Rendering Pass

- Used for transparent geometry
- Single pass solution
  - Shader has four uberlights
  - No shadows
  - Per-vertex lighting version for particles
- Lower resolution rendering available
  - Fill-rate intensive effects
  - Half and quarter screen size rendering
  - Half resolution rendering using MSAA HW

Post-Processing Pass

- Highly customizable color correction
  - Separate curves for shadows, mid-tones, highlight colors, contrast and brightness
  - Everything Depth dependent
  - Per-frame LUT textures generated on SPU
- Image based motion blur and depth of field
- Internal lens reflection
- Film grain filter

SPU Usage and Architecture

Putting it all together

SPU Usage

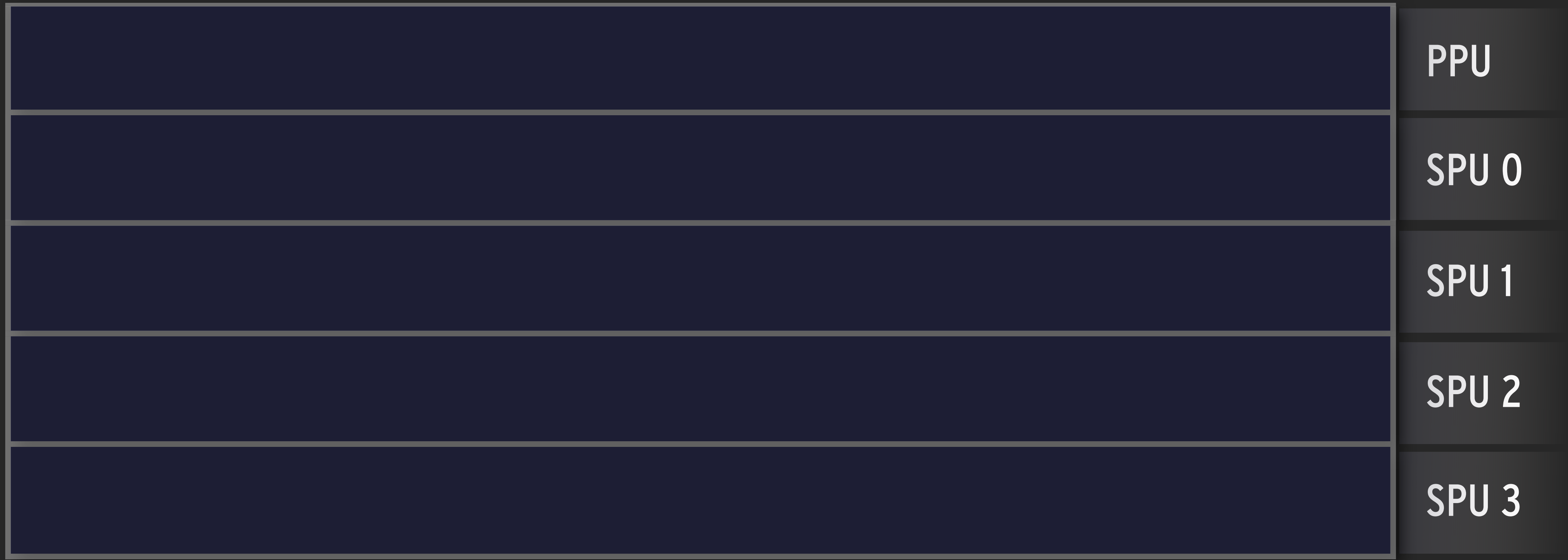
- We use SPU a lot during rendering
  - Display list generation
    - Main display list
    - Lights and Shadow Maps
    - Forward rendering
  - Scene graph traversal / visibility culling
  - Skinning
  - Triangle trimming
  - IBL generation
  - Particles

SPU Usage (cont.)

- Everything is data driven
  - No "virtual void Draw()" calls on objects
  - Objects store a decision-tree with DrawParts
  - DrawParts link shader, geometry and flags
  - Decision tree used for LODs, etc.

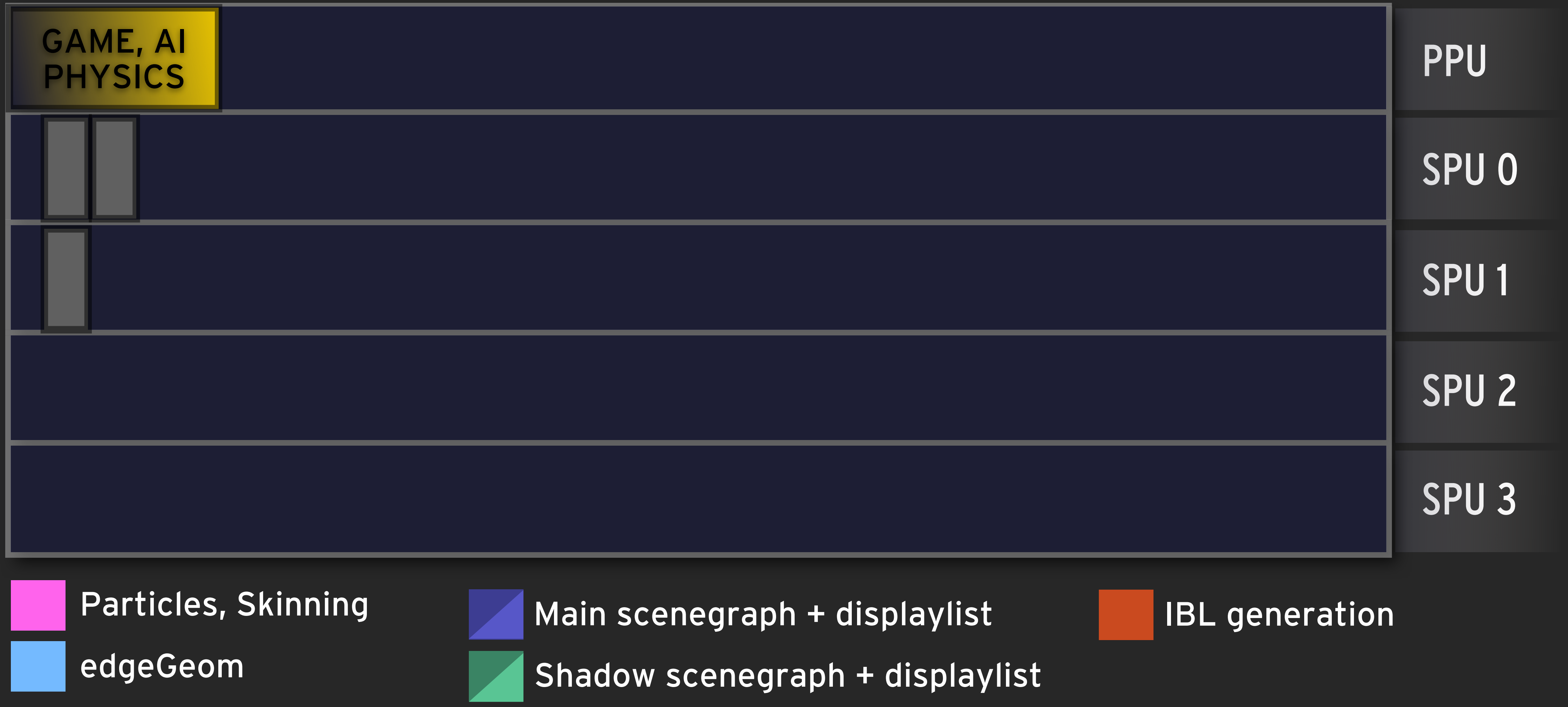
- SPUs pull rendering data directly from objects
  - Traverse scenegraph to find objects
  - Process object's decision-tree to find DrawParts
  - Create displaylist from DrawParts

SPU Architecture

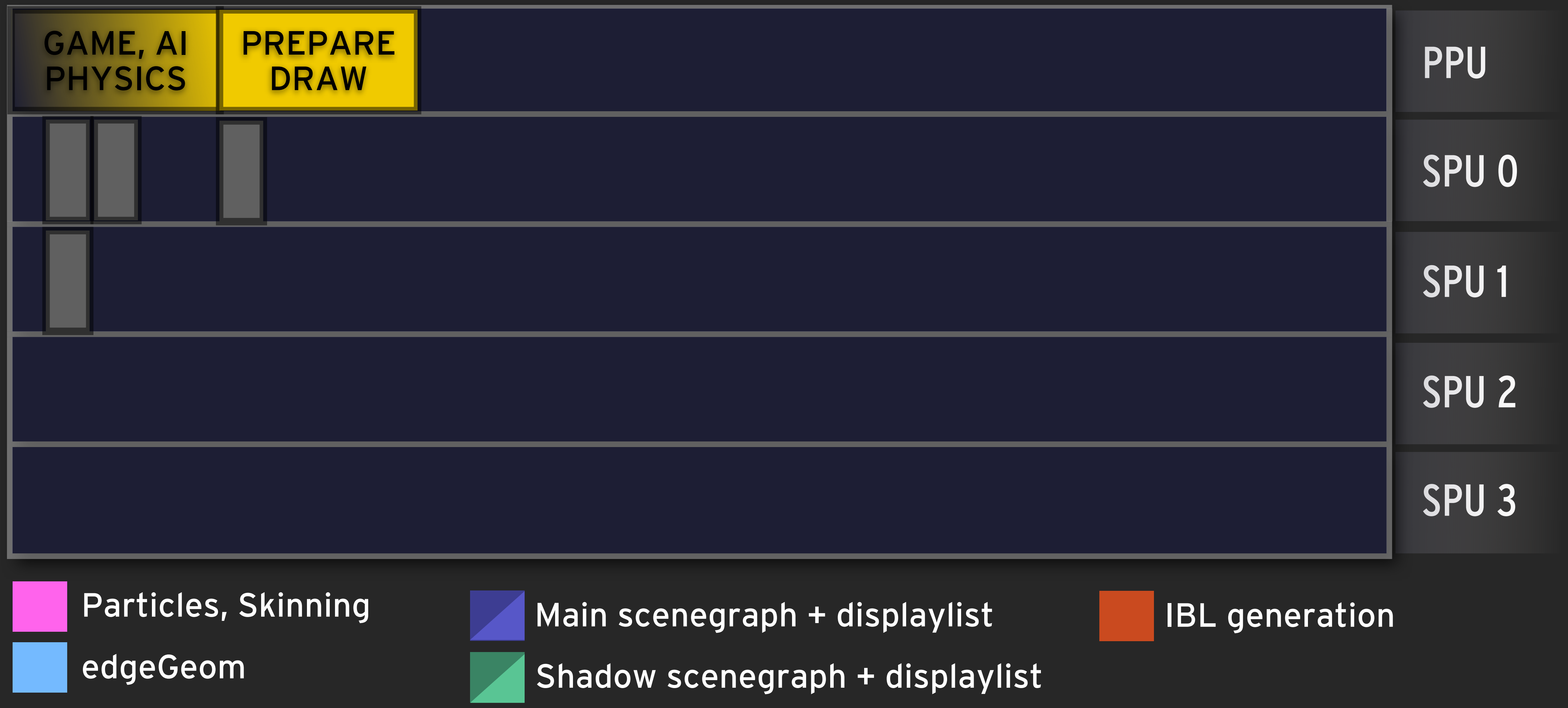


- Particles, Skinning
- edgeGeom
- Main scenegraph + displaylist
- Shadow scenegraph + displaylist
- IBL generation

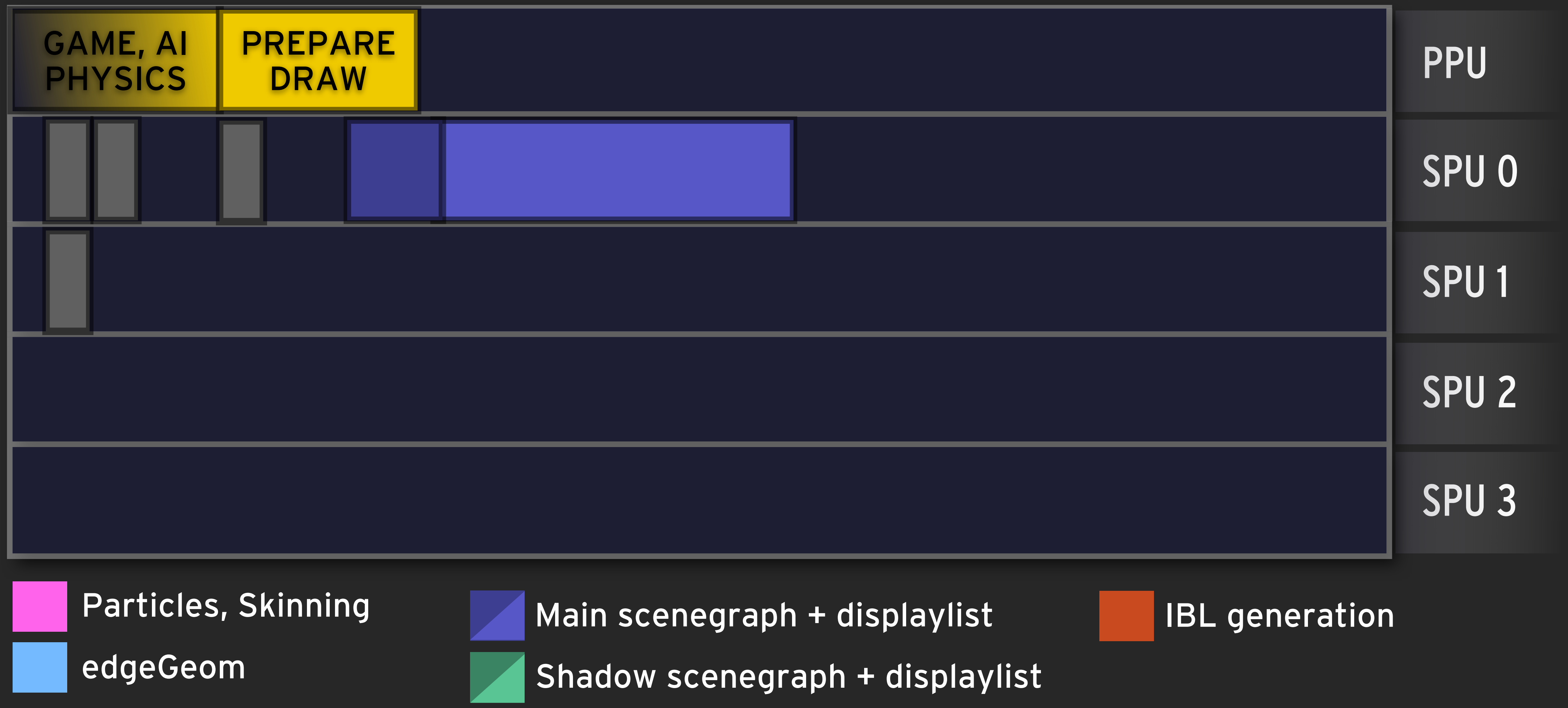
SPU Architecture



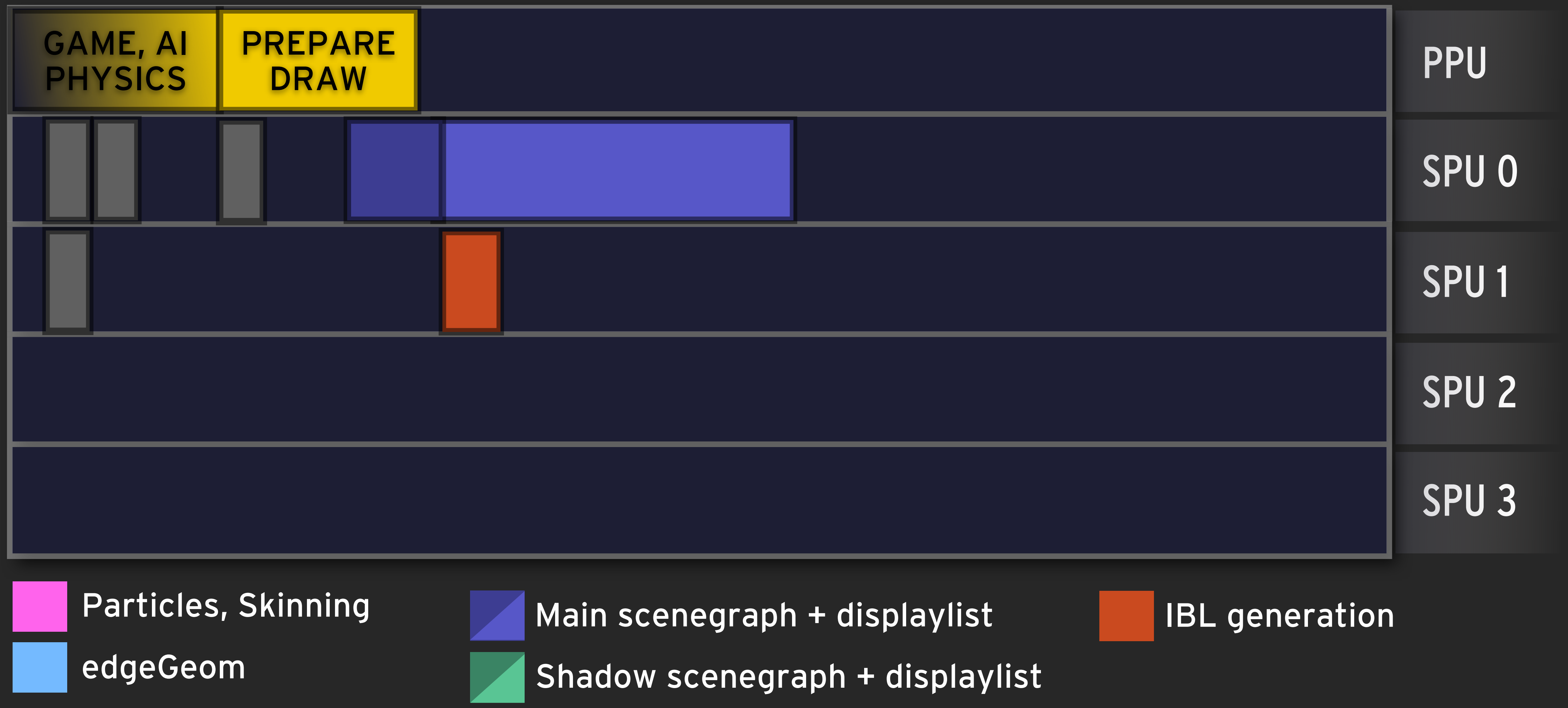
SPU Architecture



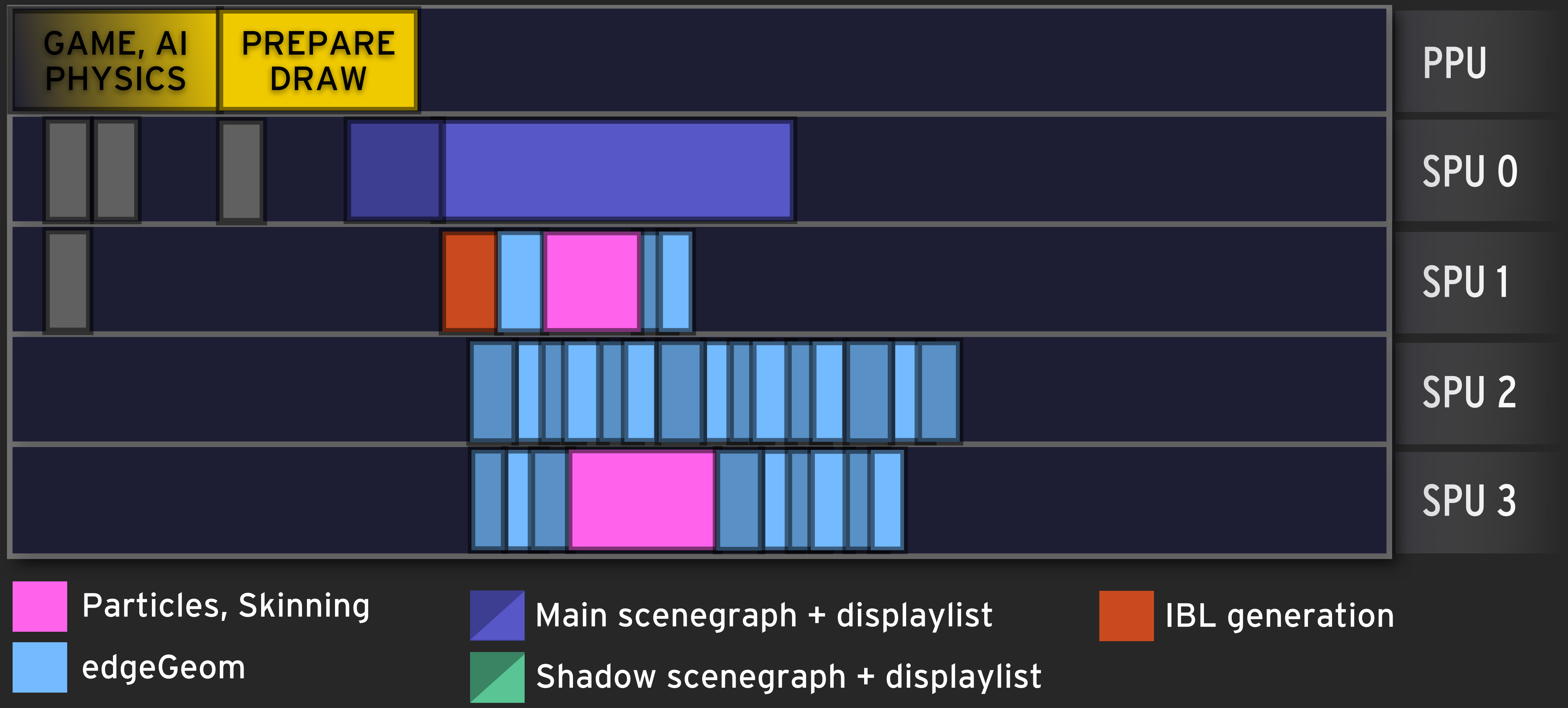
SPU Architecture



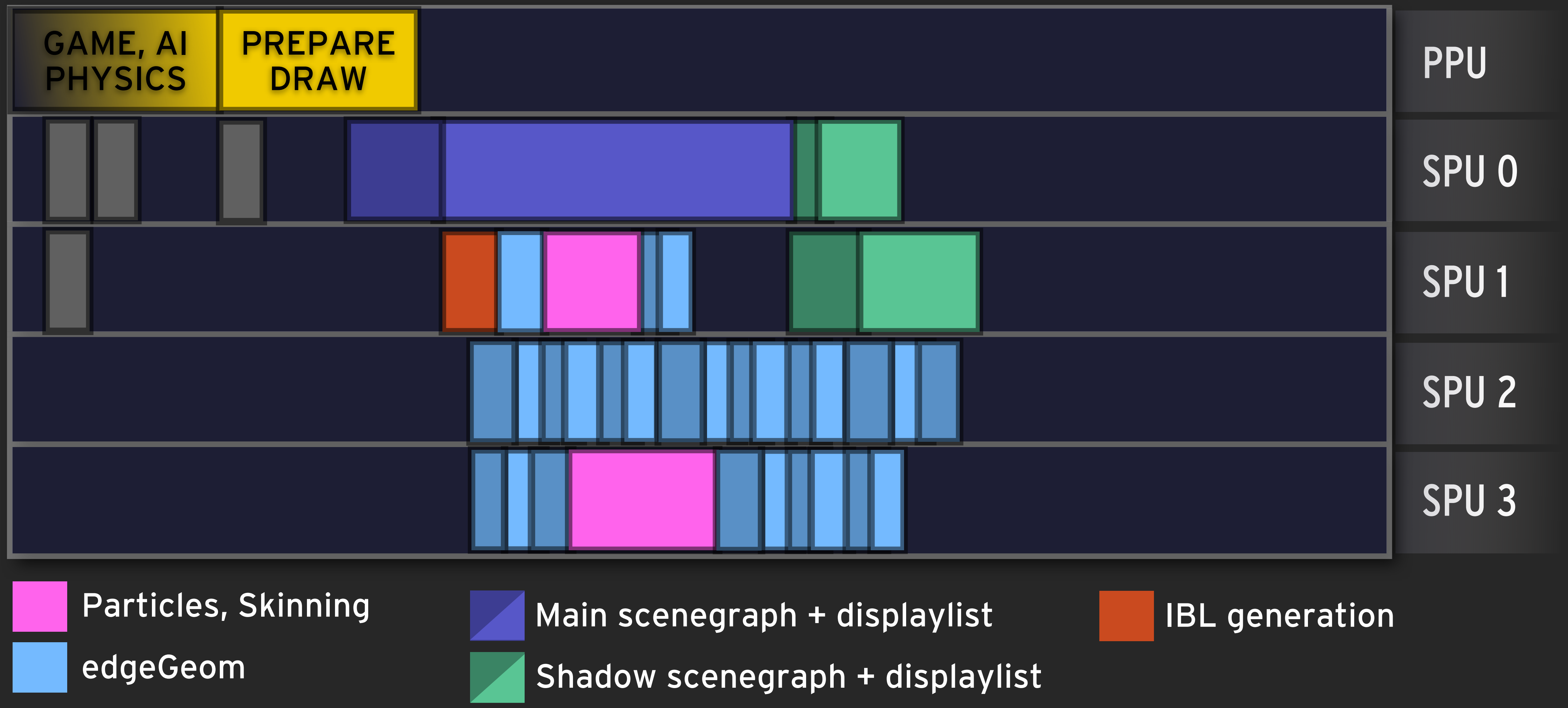
SPU Architecture



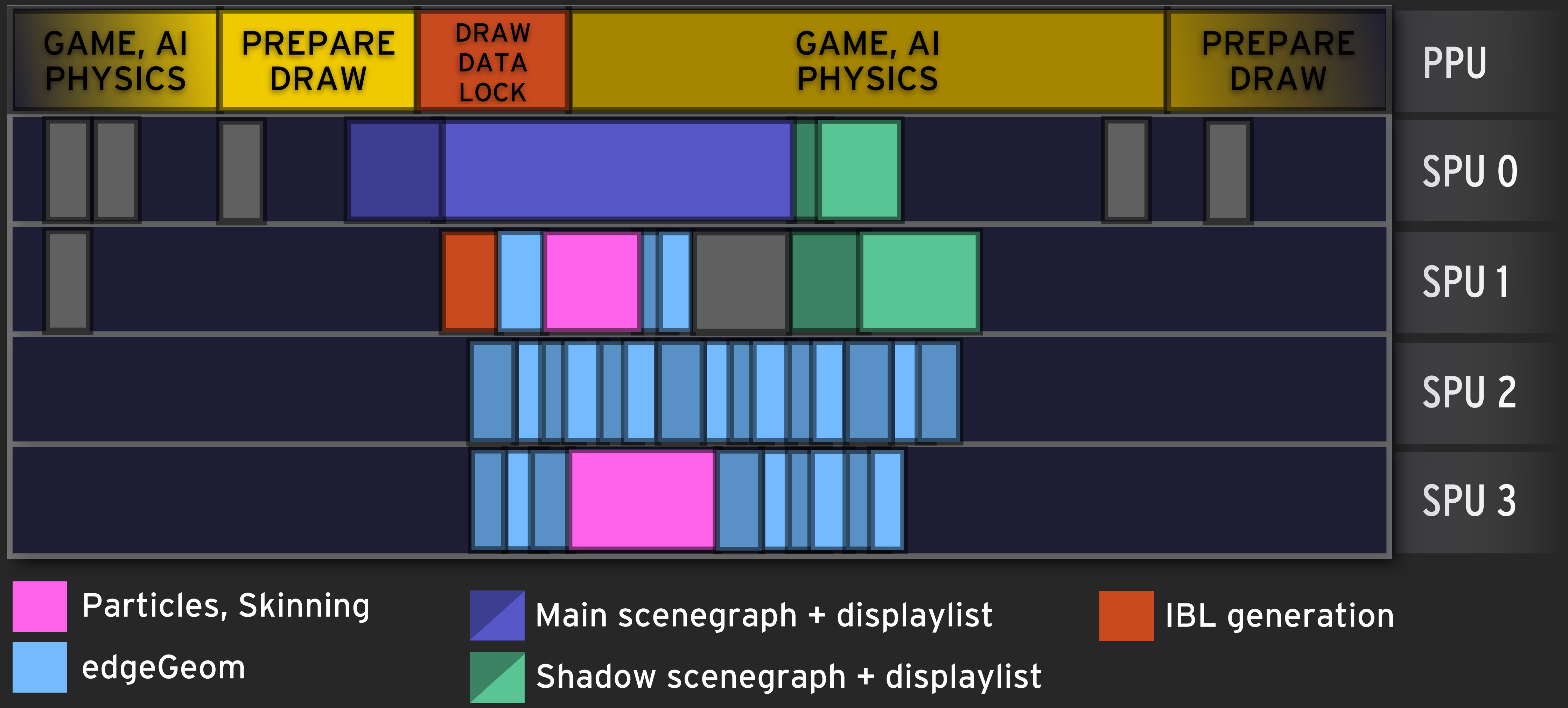
SPU Architecture



SPU Architecture



SPU Architecture



Conclusion

- Deferred rendering works well and gives us artistic freedom to create distinctive Killzone look
  - MSAA did not prove to be an issue
  - Complex geometry with no resubmit
  - Highly dynamic lighting in environments
  - Extensive post-process
- Still a lot of features planned
  - Ambient occlusion / contact shadows
  - Shadows on transparent geometry
  - More efficient anti-aliasing
  - Dynamic radiosity