Tencent Games GDC

Next Level of Mobile Graphics

Ray Tracing in Arena Breakout Mobile

Junhong Wang

Rendering Team Lead Programmer, MoreFun Studio, Tencent Games

Tencent Games GDC

01 Background

02 Ray tracing in Arena Breakout Mobile

03 Conclusion

01 Background

About Arena Breakout Mobile

- Next-Gen Immersive Tactical FPS on Mobile [Sun23]
- Android (OpenGL ES 3.1) and iOS (Metal)
- Based on Unreal Engine 4
- Large Open World
  - The Mine Map: 4km * 4km
  - 104366 Mesh Instances
  - 3000+ streaming levels

About Arena Breakout Mobile

- Forward Rendering
- Physical Based Rendering
- Dynamic Weather System [Chan22]
- GI based on Precomputed Radiance Transfer for indoor scene

About Arena Breakout Mobile

- Forward Rendering
- Physical Based Rendering
- Dynamic Weather System [Chan22]
- GI based on Precomputed Radiance Transfer for indoor scene

About Arena Breakout Mobile

- Forward Rendering
- Physical Based Rendering
- Dynamic Weather System [Chan22]
- GI based on Precomputed Radiance Transfer for indoor scene

Mobile Ray Tracing

- The latest smartphone are now ready for Hardware-accelerated Ray Query (Inline Ray Tracing)
- Ray Query can be used in any shader stage
- Most Frame Capture Tools or Profilers don’t support Mobile Vulkan Ray Query

Preparation

- Enable Vulkan RHI for Android
- Update Shader Cross Compiler (Shader Conductor)
- Extension UE4 Vulkan and Metal RHI

02 Ray Tracing in Arena Breakout

Ray Tracing in Arena Breakout Mobile

- Ray Tracing has been released in the game
  - Android Ray Tracing has been released
  - iOS Ray Tracing is under development
- Reflection
- Soft Shadow
- AO

Ray Query Reflection Off

Ray Query Reflection On

Ray Query AO Off

Ray Query AO On

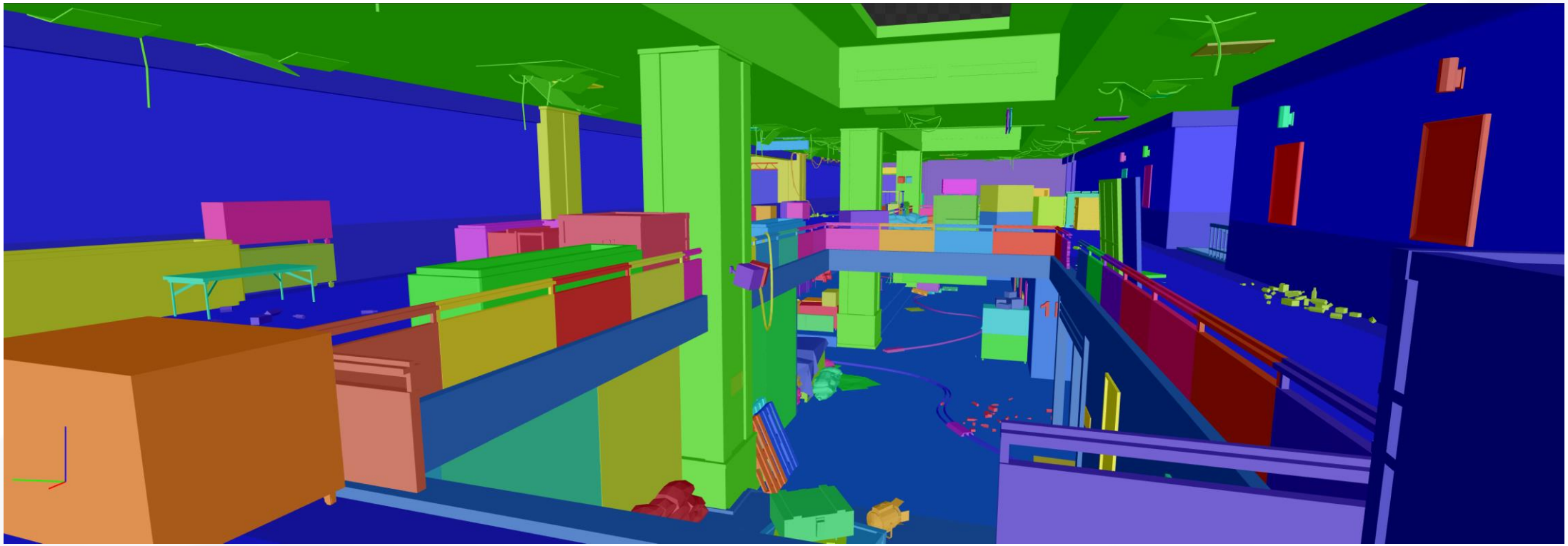
CSM Shadow

Ray Query Soft Shadow + AO

Ray Tracing Scene Management

Challenges

- Bottom-level acceleration structure (BLAS) Memory Usage
- Build BLAS is expensive
- Tracing top-level acceleration structure (TLAS) with too many instances is expensive



TLAS (Captured by NVIDIA Nsight Graphics on PC)

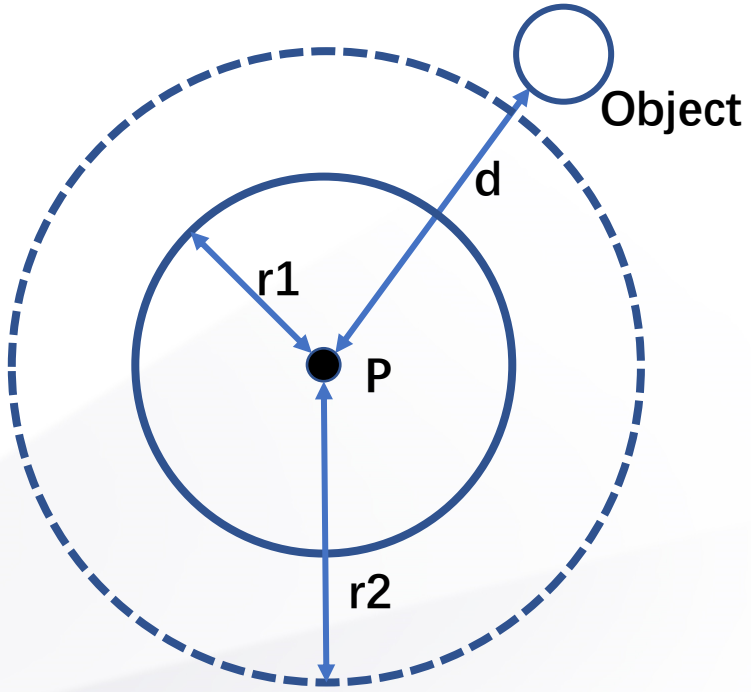
World Overview

- Levels are managed by level streaming system
- Low LOD of Mesh is loaded when package is loaded
- Higher LOD is loaded by mesh streaming system
- Meshes with Mask, World Position Offset or Transparent Material will not create BLAS

- Create BLAS when Mesh LOD is Loaded?
  - 4700+ BLAS when load map
  - The Video Memory is up to 4.4G then crashed because of out of memory

Build and Destroy BLAS

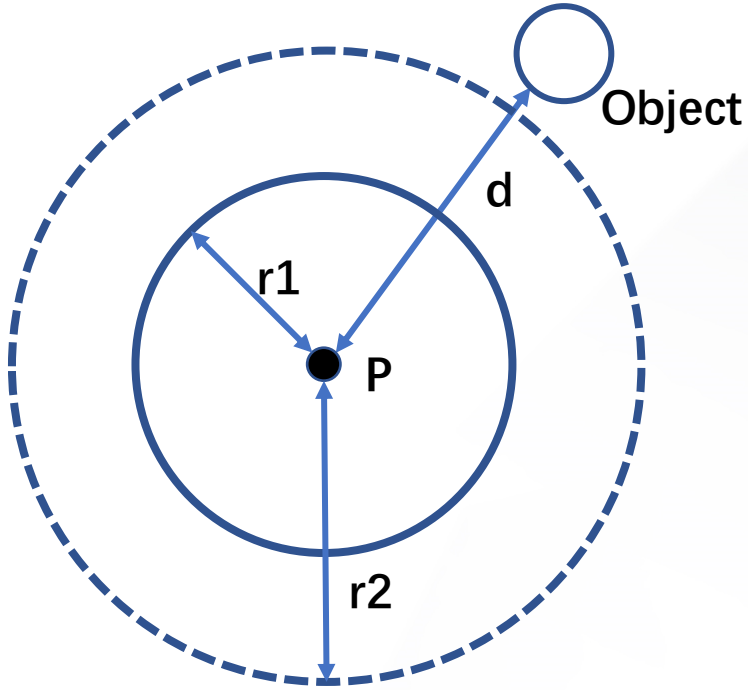
- P is the location of camera
- r1 is the BLAS creation radius
- r2 is the BLAS destruction radius
- d is the distance from camera to Object's Bounds Sphere




```
for(Camera : Cameras)
{
    for(SceneProxy : PrimitiveSceneProxies)
    {
        // Calculate the value d
        d = distance from Camera to SceneProxy Bounds

        if(d < r1)
            // Mark Object as request
            SceneProxy->MarkRequestFrameNum = CurrentFrameNumber;
        else if(d < r2)
            // Mark Object as keep
        SceneProxy->MarkKeepFrameNum = CurrentFrameNumber;
    }
}



for(SceneProxy : PrimitiveSceneProxies)
{
    if((CurrentFrameNumber - SceneProxy->MarkRequestFrameNum < CreateThreshold) && (SceneProxy->BLAS == nullptr))
    {
        // Create BLAS
        ...
    }
    else if ((CurrentFrameNumber - SceneProxy->MarkKeepFrameNum > ReleaseThreshold) && (SceneProxy->BLAS != nullptr))
    {
        // Destroy BLAS
        ...
    }
}
```

Build and Destroy BLAS

- Lazy create BLAS only when object is in r1
- Delay destroy BLAS by n frames to avoid frequent creation and destruction
- All parameters are scalable



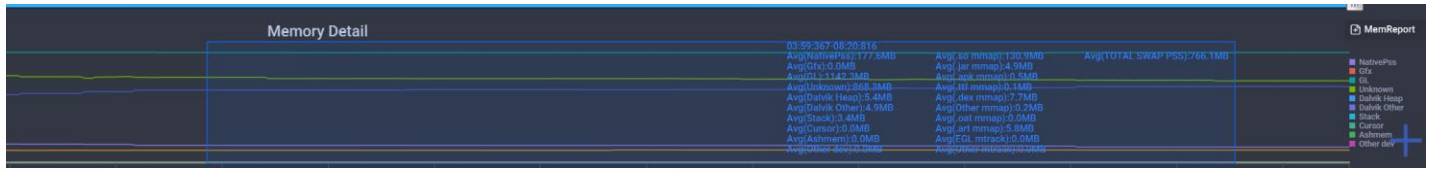
Result

- Old:

4700+ BLAS
over 4.4G then OOM

- New:

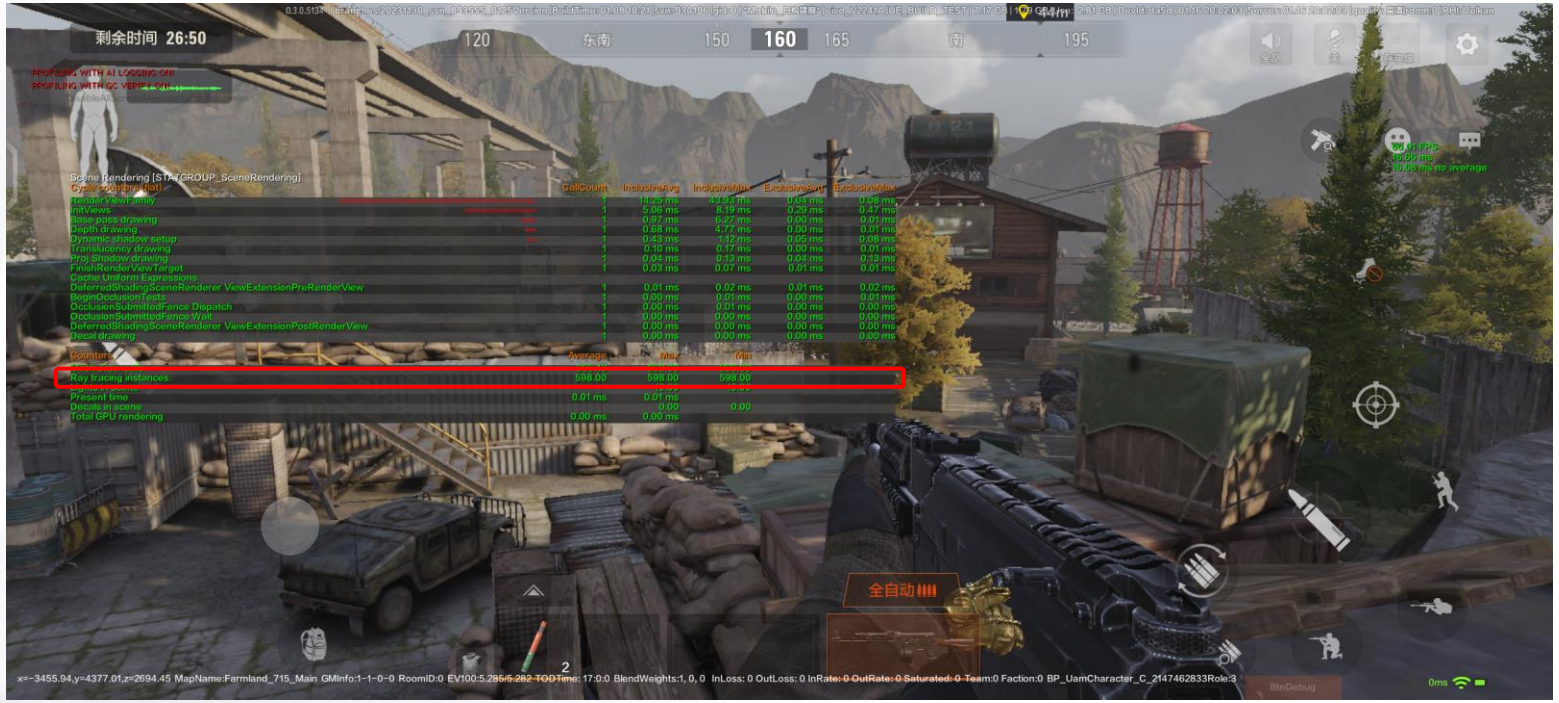
700+ BLAS
1.1G (Game total video memory usage)



Ray Tracing Scene Management

TLAS

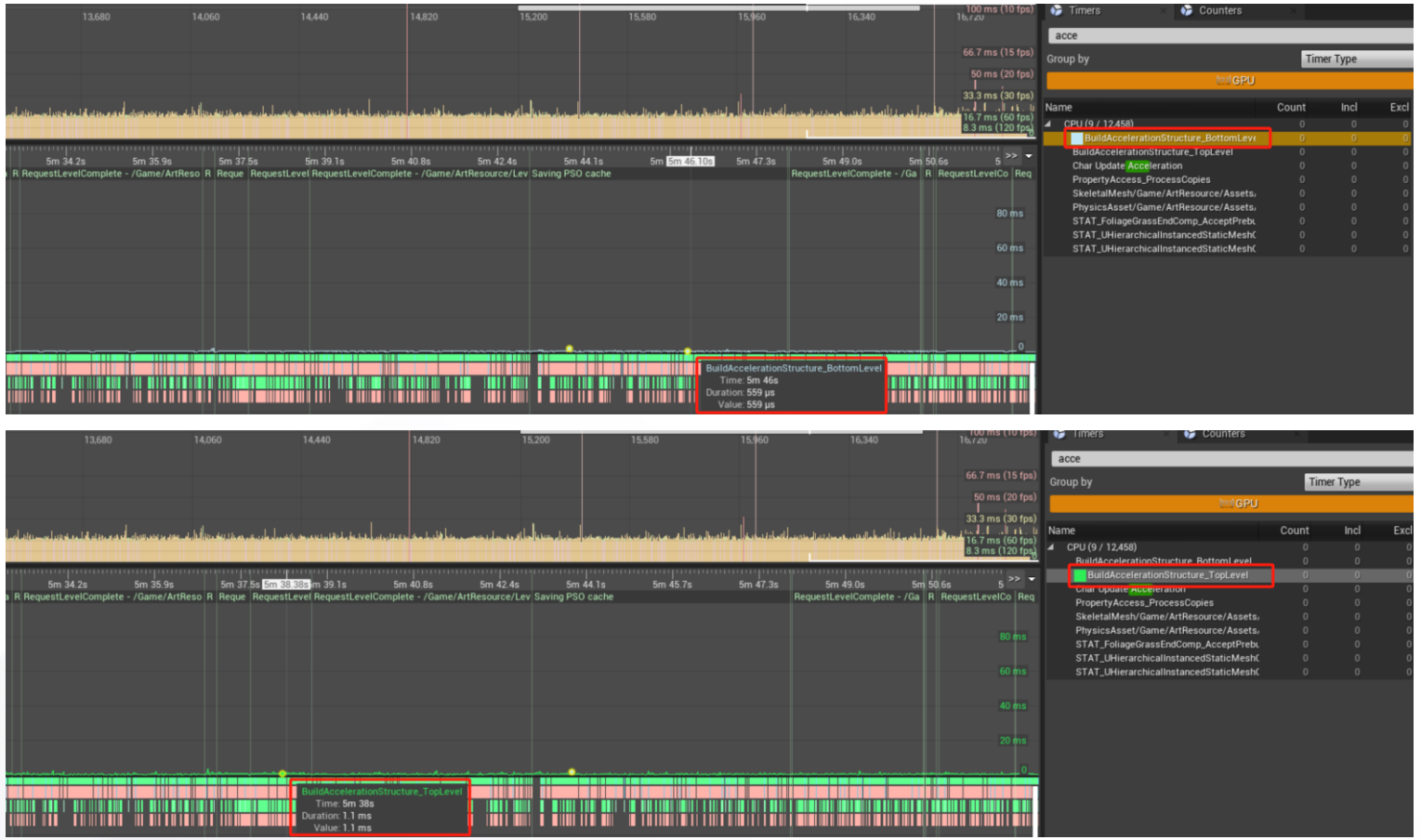
- Update Every Frame
- Enable VK_BUILD_ACCELERATION_STRUCTURE_PREFER_FAST_TRACE_BIT_KHR
- Instance Culling
  - Distance Culling [Netzel22]
  - Projected Angle Culling [Deligiannis19]



TLAS Instance Count 598

Performance

- Build BLAS < 0.5ms
- Build TLAS: 1ms



Vivo X90 (SOC: MTK Dimensity 9200, Mali G715)

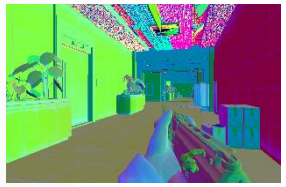
Ray Tracing Reflection

Challenges

- Shading Reflection Pixels
  - No RTPSO
  - No Bindless Texture
  - Thousands of different Materials
- Hybird Rendering Pipeline
  - Current Rendering Pipeline is Forward Rendering
- Performance
  - Energy Cost
  - FPS

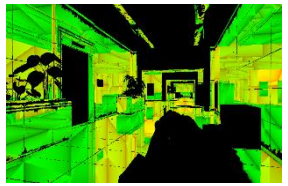
Rendering Pipeline

Opaque SceneColor



Normal
+ Roughness
+ Flags

Depth



TriangleID
+ Barycentric Coord

Mesh Instance ID

Reflection Texture

Reflection Texture

Reflection Texture

Blend Result

Final Result

Base Pass → Query Scene Pass → Visibility Resolve Pass → Joint Bilateral Filtering → Joint Bilateral Filtering → Blend → Post Processing

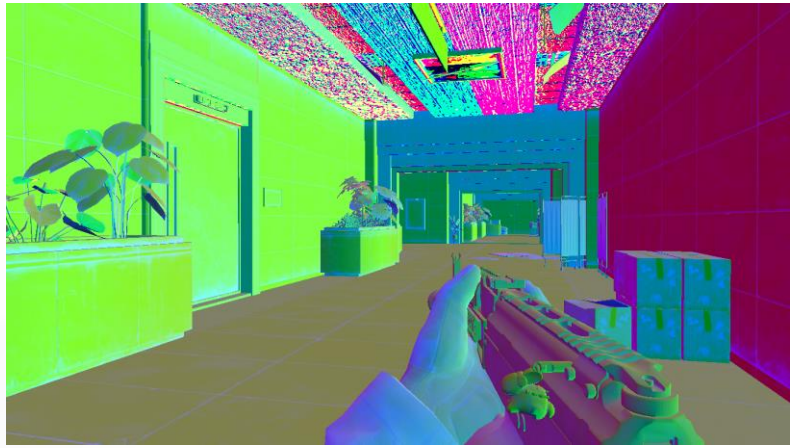
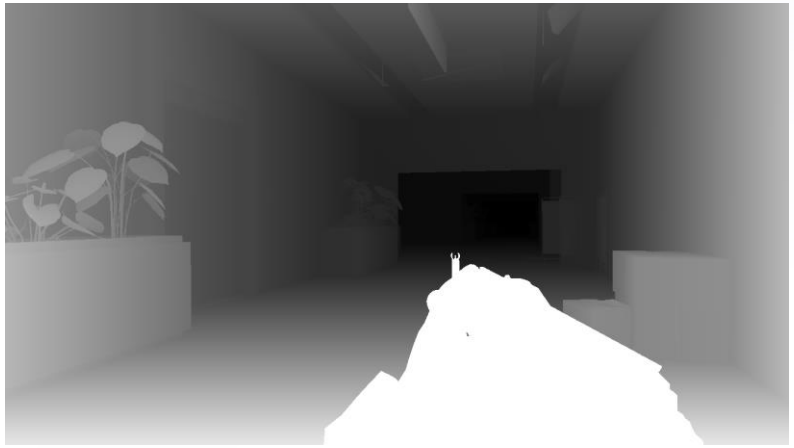
Preparation

- Create Special Mesh Shader for Ray Query Reflection Shading
- Collect Ray Query Mesh Draw Commands
- Reallocate Mesh Instance ID for Mesh Draw Commands Every Frame
- When Build TLAS, assign Mesh Instance ID and HitGroupId

```
1   struct VkGeometryInstance
2   {
3       /// Transform matrix, containing only the top 3 rows
4       float Transform[12];
5       /// Instance index
6       uint32 InstanceID : 24;
7       /// Visibility mask
8       uint32 Mask : 8;
9       /// Index of the hit group which will be invoked when a ray hits the instance
10      uint32 HitGroupId : 24;
11      /// Instance flags, such as culling
12      uint32 Flags : 8;
13      /// Opaque handle of the bottom-level acceleration structure
14      uint64 AccelerationStructureHandle;
15  };
```

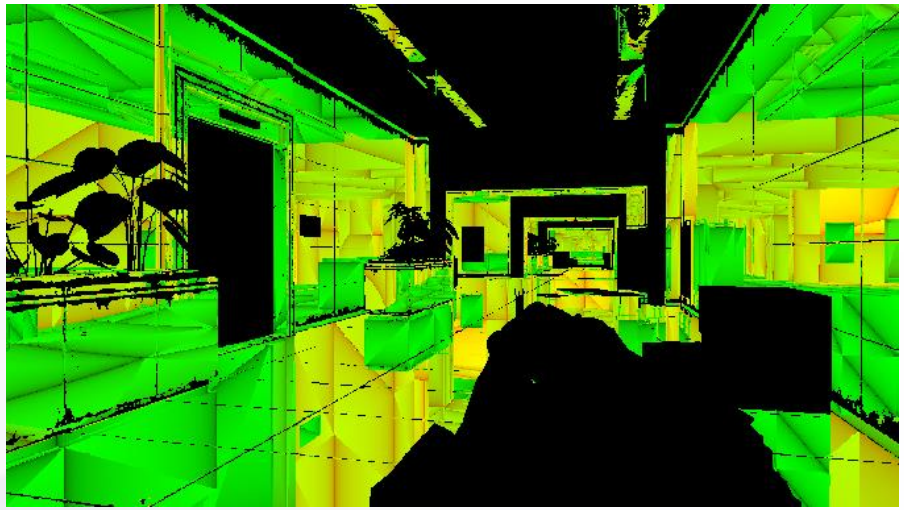
Base Pass

- MRT
  - RT0: SceneColor (R11G11B10)
  - RT1: ThineGBuffer (Normal + Roughness + Flags) (A2B10G10R10)
- Depth Stencil
  - D24S8
- Optional
  - Reflection Color: For Metal Material

Screen Space Query Scene Pass

- RT0: Packed Triangle ID and Barycentric Coordinate the ray hit
- RT Optinal: Hit Ray input direction
- Depth Buffer: Mesh Draw ID the ray hit
- Algorithm:
  1. Reconstruct Camera Relative World Position and reflection ray direction
  2. Emit ray, if hit any TLAS Instance, get Mesh Instance ID, Triangle ID and Triangle Barycentric Coordinate of the hit point
  3. Write the result to RT0 and Depth Buffer



Visibility Resolve Pass: CPU

- Dispatch Ray Query Visibility Resolve Draw Calls
  - The Draw ID is committed by uniform buffer
- Each Draw Call is a full screen quad

Visibility Resolve Pass: GPU

- If Draw ID is mismatch, discard the pixel
- Interpolate PBR parameters in Pixel Shader
- Emit Shadow Ray
- Calculate direct light and indirect light
- Write the reflection RT

Ray Tracing Reflection

Pros

- Good hardware compatibility

Cons

- Draw Call
  - Draw Call count is the same as TLAS instance count (600+)
- Overdraw
  - Too many full screen quad draw call

Visibility Resolve Pass

- Draw Call
  - Use GPU Occlusion Query to remove invisible Mesh Draw Commands
- Overdraw
  - Assign the Mesh Draw ID to the quad depth in vertex shader
  - Enable depth test to avoid overdraw

Visibility Resolve Result

- No invisible draw call
  - Before: 600+ draw call
  - After: 110+ draw call
- No overdraw

Blur Pass 1

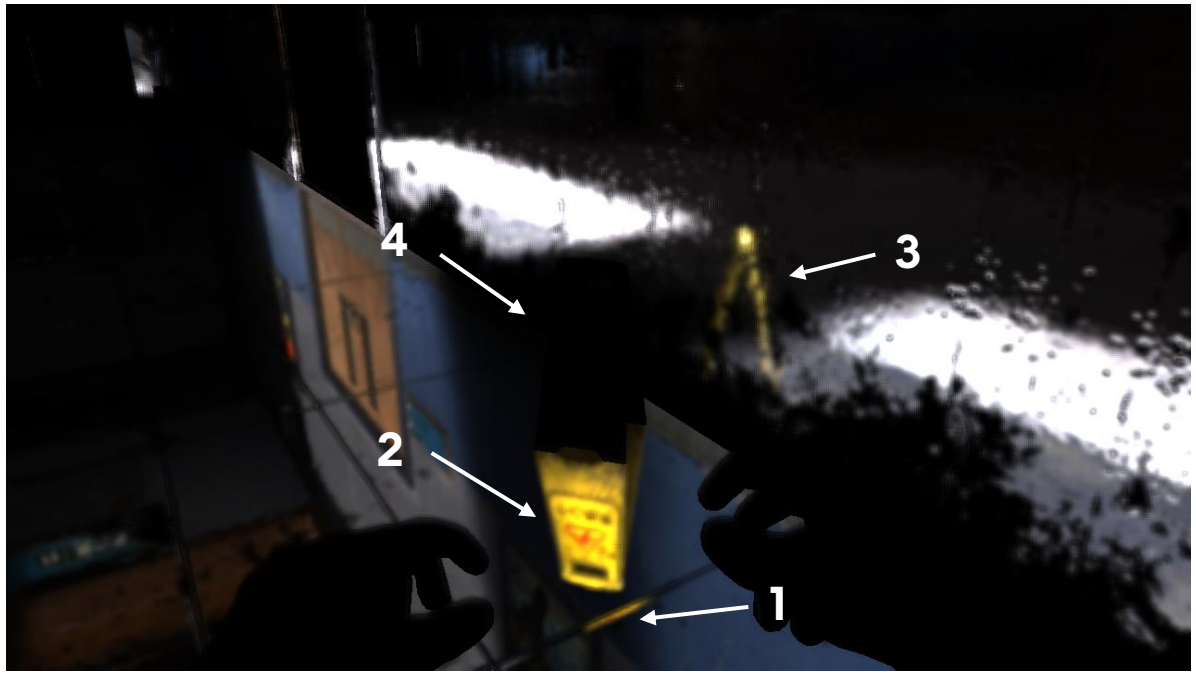
- Input Textures
  - Reflection Texture
  - ThinGBuffer Texture(Normal + Roughness + Flags)
  - Depth Texture
- Algorithm
  - Joint Bilateral Filter
  - Kernel offset size is adjust by roughness

Blur Pass 2

- Joint Bilateral Filter
- Kernel Offset is larger

Blend Pass

1. Edge-preserving
2. Sharp reflection for smooth floor
3. Blurry reflection for rough wall
4. No emitted ray for fully rough material



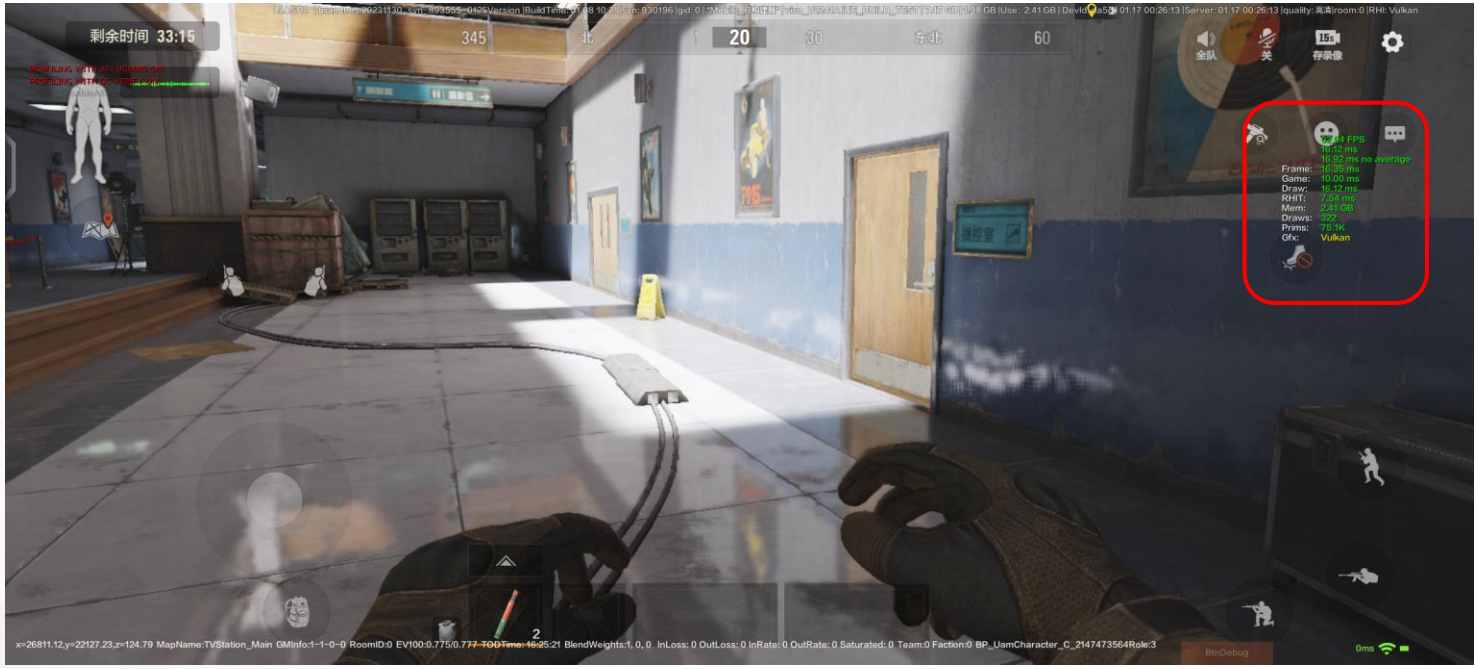
Ray Tracing Reflection

Pros

- Good hardware compatibility
- High performance (>60 FPS on MTK Dimensity 9200)

Cons

- Difficult to process transparent or mask material



Vivo X90 (SOC: MTK Dimensity 9200)

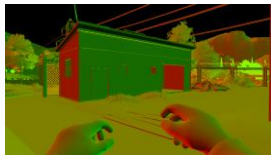
Ray Tracing Soft Shadow and AO

Challenges

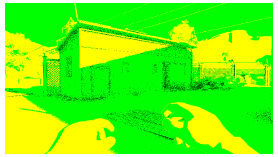
- Both need many rays
- Performance

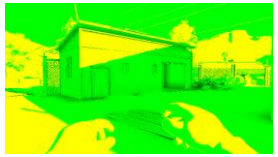
Shadow: 1 ray
AO: 1 ray
No Denoise

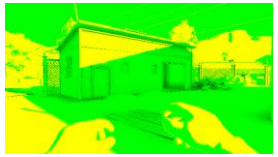
Rendering Pipeline

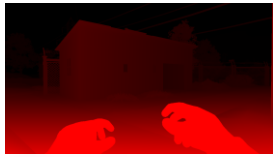

Geometry Normal

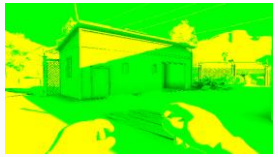

Depth


Shadow + AO


Shadow + AO


Shadow + AO


PreDepth


PreShadow + PreAO

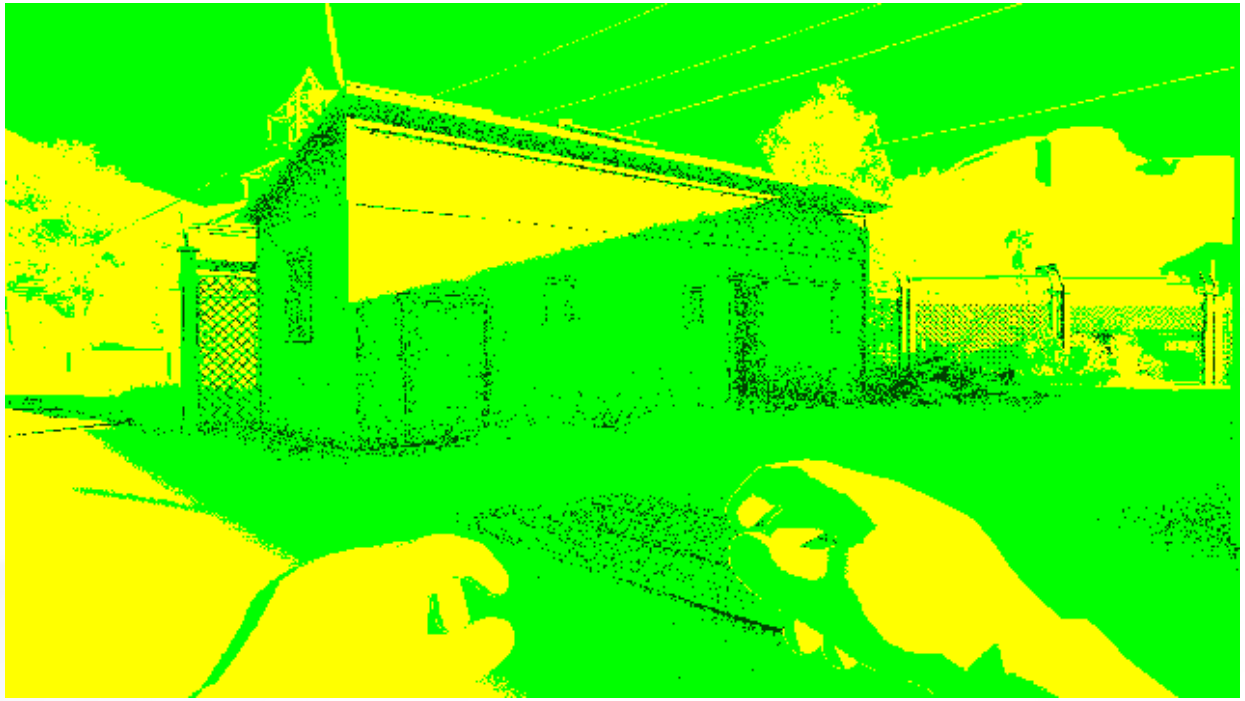

Base Color


Result

Pre Pass → Query Scene Pass → Temporal Denoise → A-Trous Filter (2 pass) → Base Pass → Post Processing

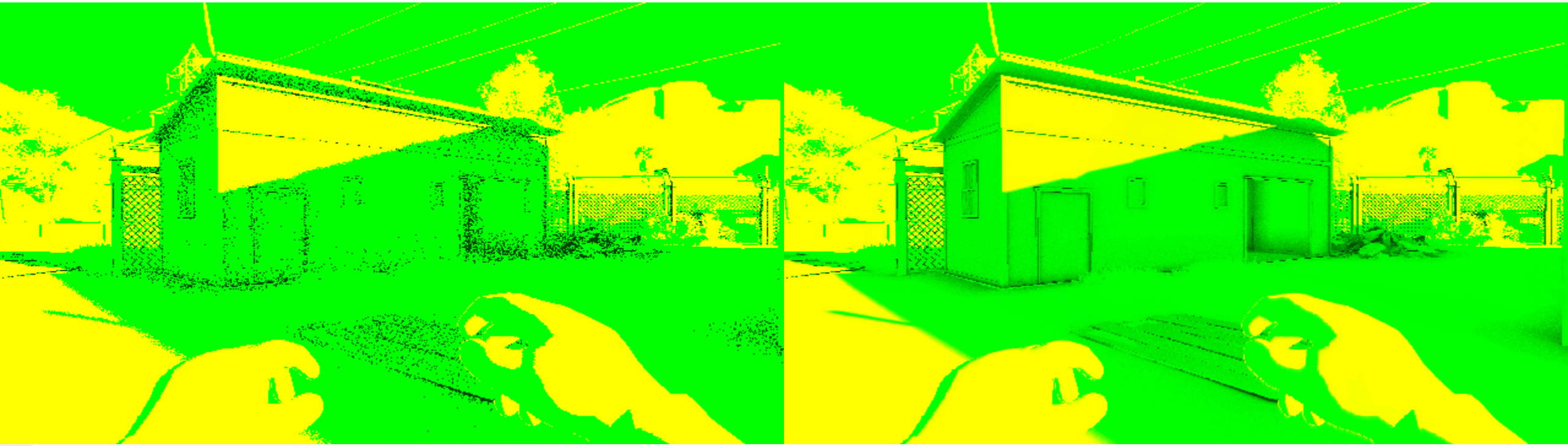
Query Scene Pass

- Regard sun as sphere
- Use depth to reconstruct ray origin, use Halton sequence to generate sample direction
- 1 ray for shadow, 1 ray for AO
- Channel R is shadow, G is AO
- Back face culling for shadow ray
- Lower resolution



Temporal Denoise Pass

- Reproject screen position to previous shadow & AO texture
- Reject by depth

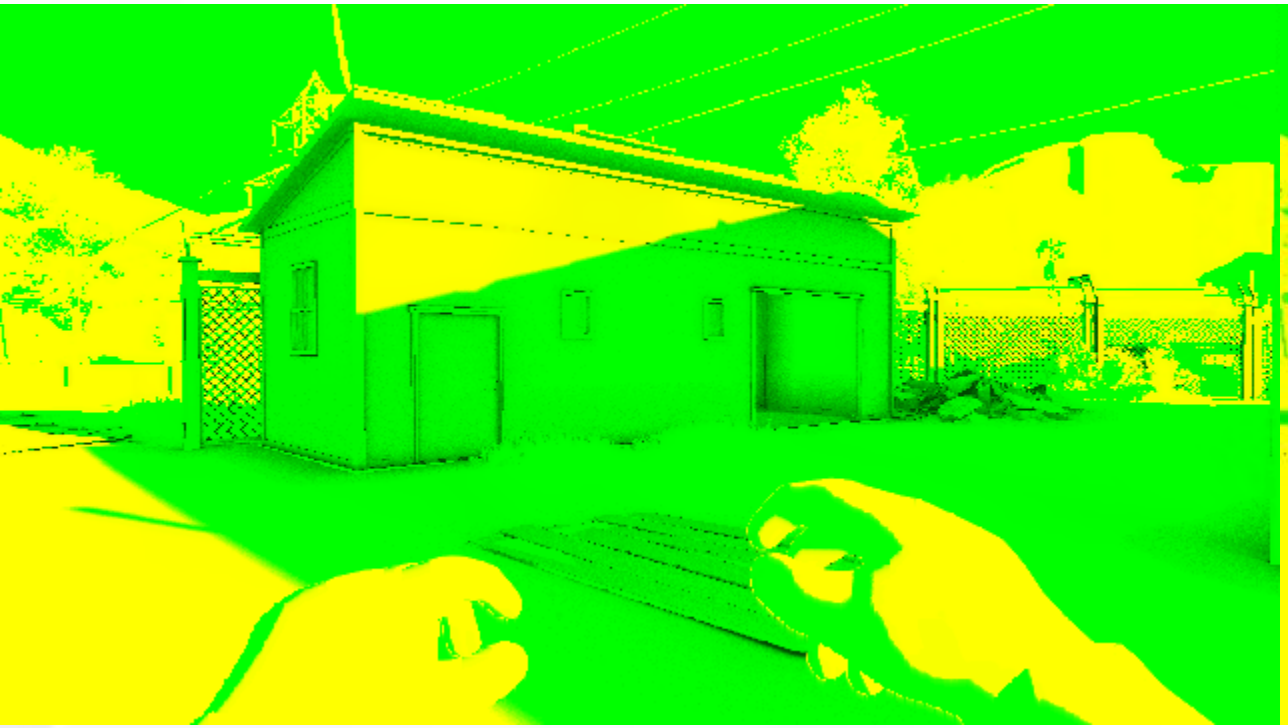


Before

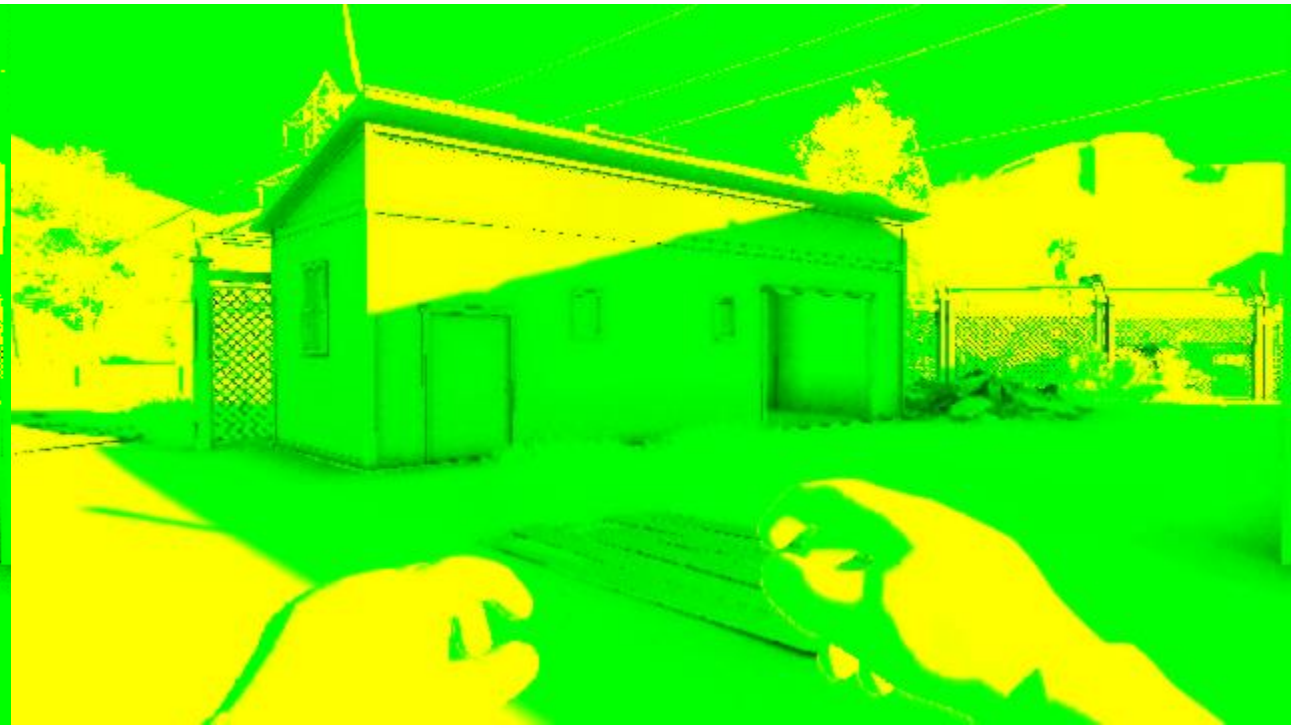
After

A-Trous Filter

- 2 Pass A-Trous Filter
- Kernel Offset is larger for the 2th Pass



Before

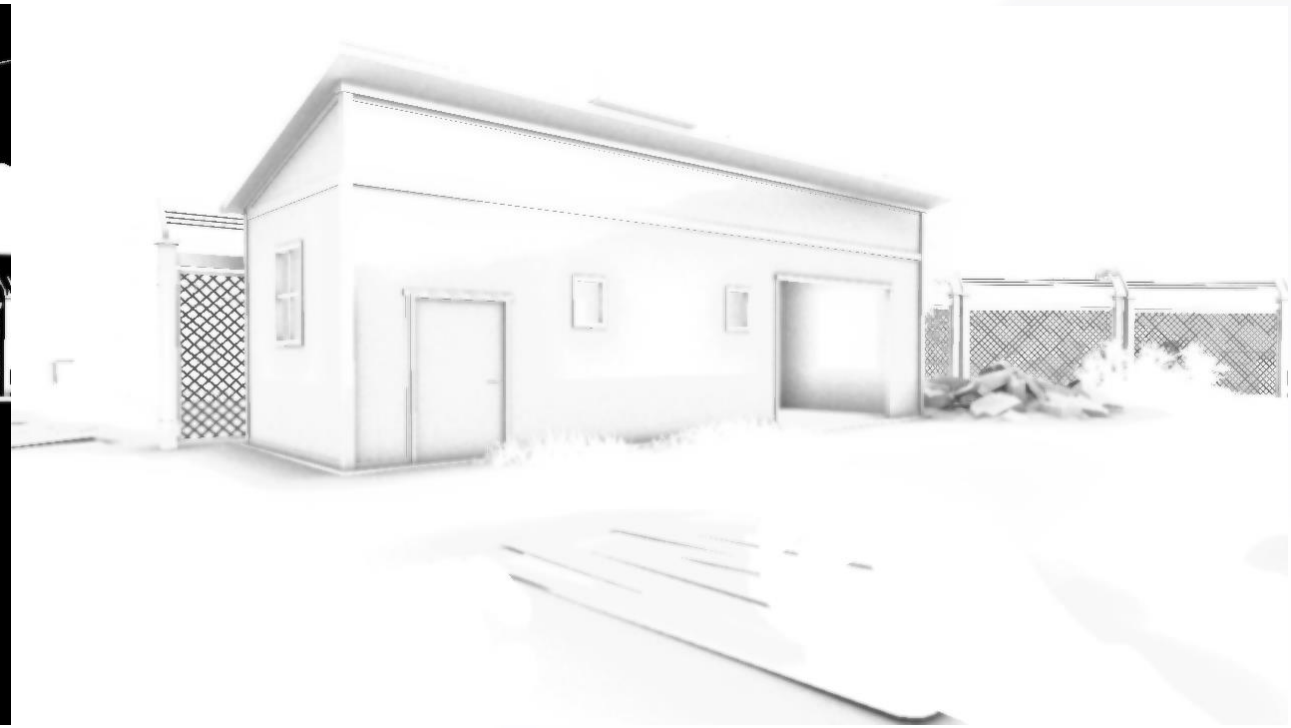


After

Result

- Here I modify size of sun by doubling it, to get softer shadow

Shadow



AO

Result

- Scene with soft shadow and AO

Pros

- High Performance(>70 FPS on MTK Dimensity 9200)

Vivo X90(SOC: MTK Dimensity 9200)

Cons

- Mesh with mask material (Foliage) should use other shadow algorithm
- Mesh with transparent material can't sample correct shadow & AO (Have to emit new rays for it)
- Too many rendering pass and texture sampling lead to high power cosumption

CSM Shadow

Ray Query Shadow

Glass sampling Ray Query Shadow

Frame Predication

- Use frame predication to create half of the ray query intermediate frames [Yue24]
- My colleague Yue Qi has introduced the frame predication in this conference

03 Conclusion

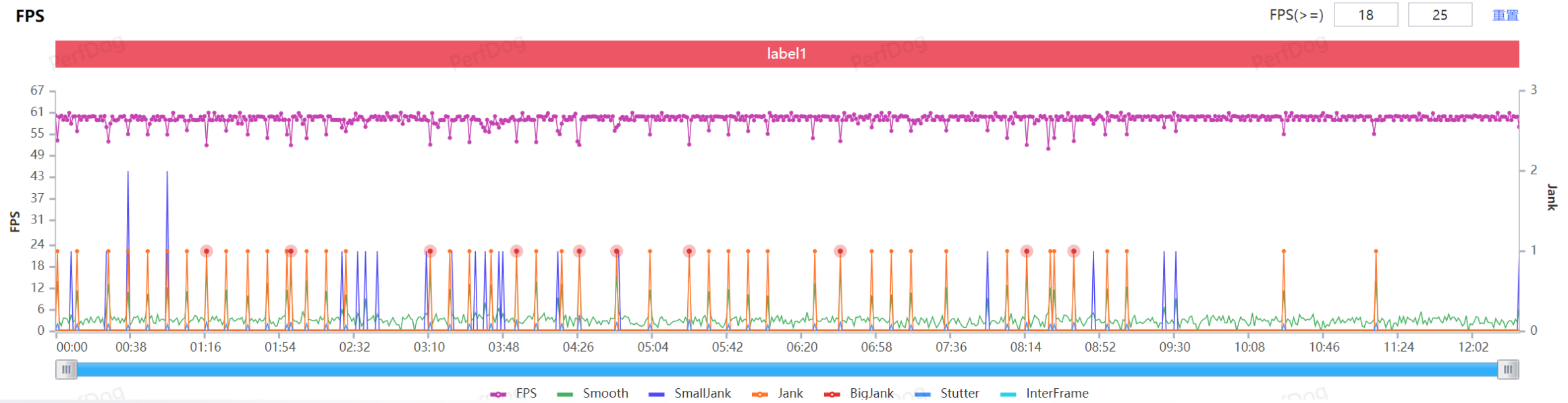
Performance

- FPS on Vivo X90 (MTK Dimensity 9200)

FPS

Avg(FPS)	Smooth(稳帧指数)	1%Low(FPS)	Std(FPS)	Var(FPS)	FPS>=18 [%]	FPS>=25 [%]	Drop(FPS) [/h]	Min(FPS)	Median(FPS)	MedRange(FPS)[%]	SmallJank(微小卡顿) (/10min)
59.2	3.7	22.9	1.5	2.3	100	100	9.7	51	59.9	100	53.9

Jank(卡顿) (/10min)	BigJank(严重卡顿) (/10min)	Stutter(卡顿率) [%]	Avg(InterFrame)	Avg(FPS+InterFrame)	Avg(FTime) [ms]	Std(FTime)	Var(FTime)	FTime>=100ms [%]	Delta(FTime)>100ms [/h]
37	8	0.7	0	59.2	16.9	4.6	21.3	0.1	57.8

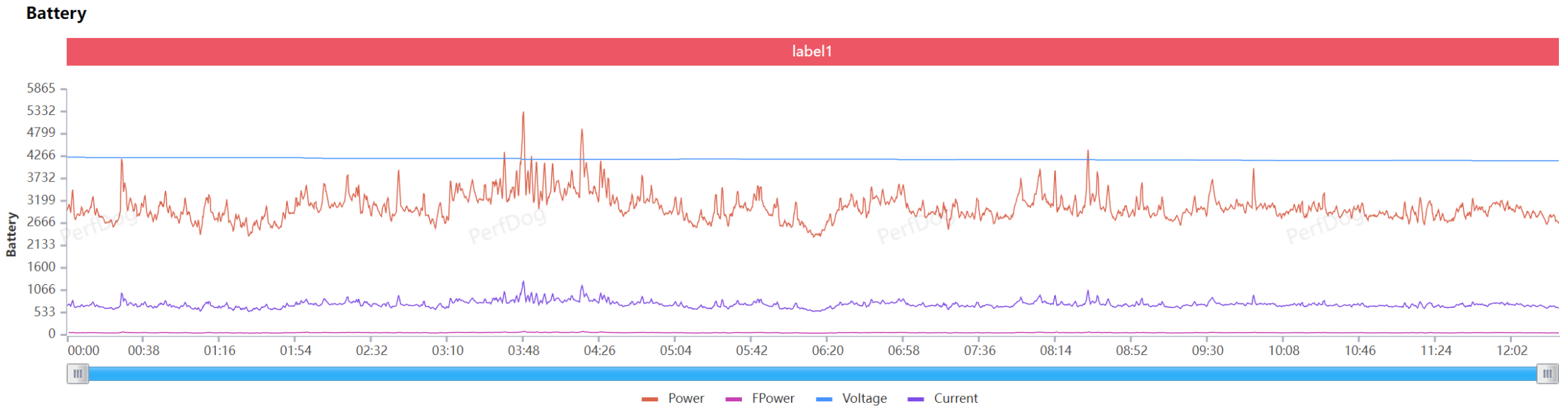


Performance

- Power consumption on Vivo X90 (MTK Dimensity 9200)

Battery

Avg(Power) [mW]	FPower(帧能耗) [mW]	Sum(Battery)[mWh]	Avg(Voltage) [mV]	Avg(Current) [mA]
3044.7	51.4	630.9	4192.4	726.4



Acknowledgements

Ray Tracing Project Members

- Jianbin Zhong
- Kaigeng Gao
- Yue Qi
- Shiwen Cui

Arena Breakout Rendering Team

MediaTek and Vivo

References

- [Sun23] – Arena Breakout: Create a Next-gen FPS Game on Mobile; Yiming Sun; GDC 2023
- [Chan22] – Advanced Weather System for Mobile Game: 'Dark Area'; Ka Ming Chan, Ruochen Ying; GDC 2022
- [Netzel22] - Ray Tracing Open Worlds in Unreal Engine 5; Aleksander Netzel, Tiago Costa; SIGGRAPH 2022
- [Deligiannis19] – It Just Works: Ray-Traced Reflection in 'Battlefield V'; Johannes Deligiannis, Jan Schmid; GDC 2019
- [Yue24] – 144FPS Rendering on Mobile: Frame Prediction in Arena Breakout; Yue Qi; GDC 2024

Thank You

Tencent Games

GDC