144FPS Rendering on Mobile:
Frame Prediction in Arena Breakout

Yue Qi
Mobile Programmer of Arena Breakout, MoreFun Studio, Tencent Games
E-mail: cinyqi@tencent.com

Tencent Games GDC

01 Background
Motivation

02 Frame Prediction
The implementation of frame prediction algorithm

03 Rendering Pipelines
The corresponding rendering pipelines with frame prediction

04 Conclusion
Analysis of performance and further applications

01 Background

Motivation

Background

Situation

- More and more mobile devices support high screen refresh rates (90, 120, 144Hz and higher)
- Players demand smoother experience without reducing the quality of graphics

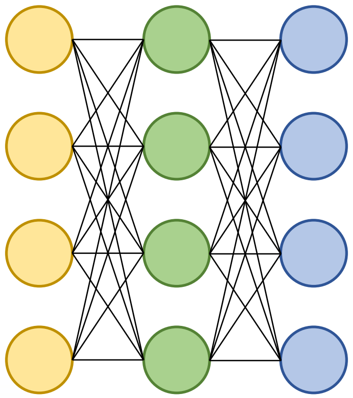
Challenges

- Conflict between graphics quality and frame rate: Higher graphic quality needs more rendering time, but it will reduce the frame rate
- Battery consumption and overheating

Background

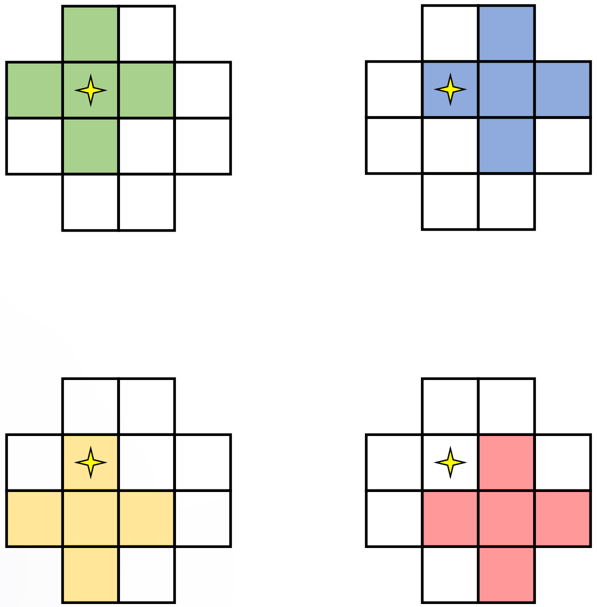
Existing solutions?

- Deep learning based solution?
  - Specific hardware dependencies ×
  - High cost and low response ×



Other solutions?

- Deep learning based solution ×
- Software super-resolution?
    - High bandwidth cost (large number of neighborhood pixels sampling) ×
    - Only a small increase in frame rate ×

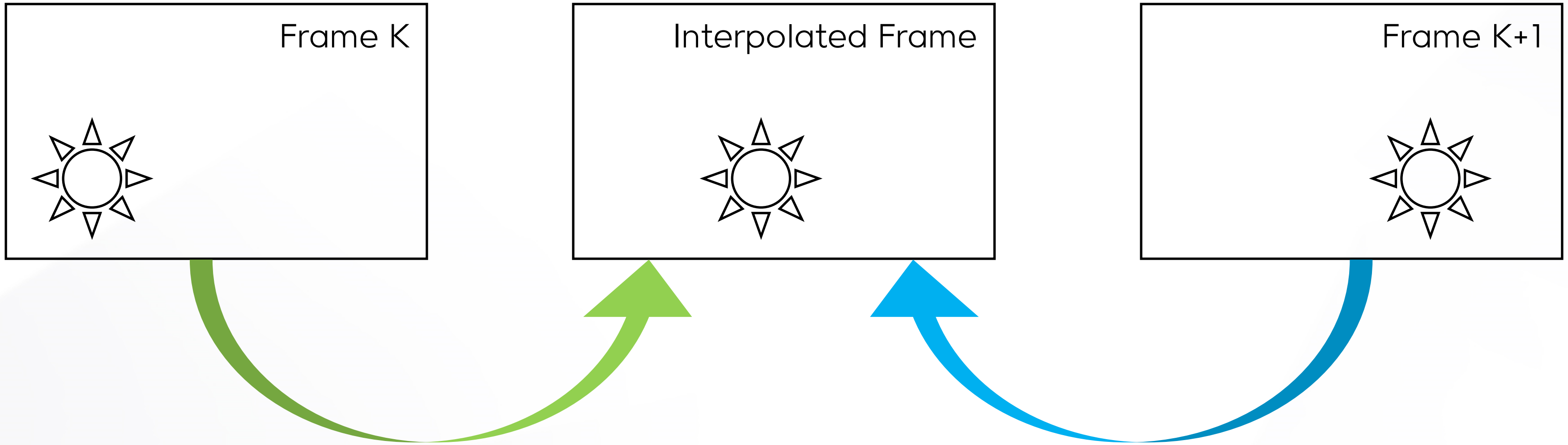


Edge calculation for upsampling

Background

Other solutions?

- Deep learning based solution ×
- Software super-resolution ×
- Frame interpolation?
    - Additional operation response latency ×
    - Interpolation cost ×

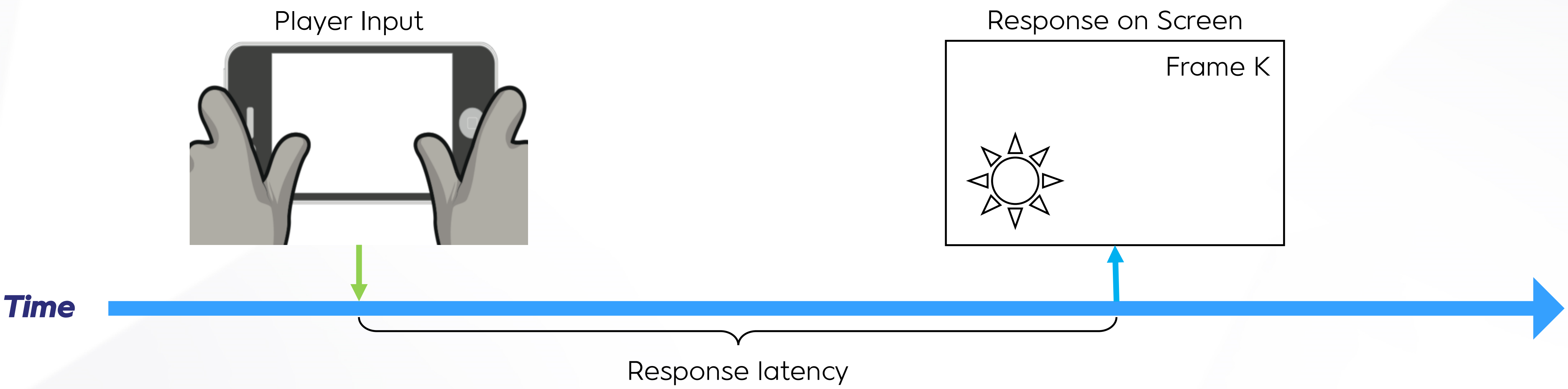


Frame interpolation

Background

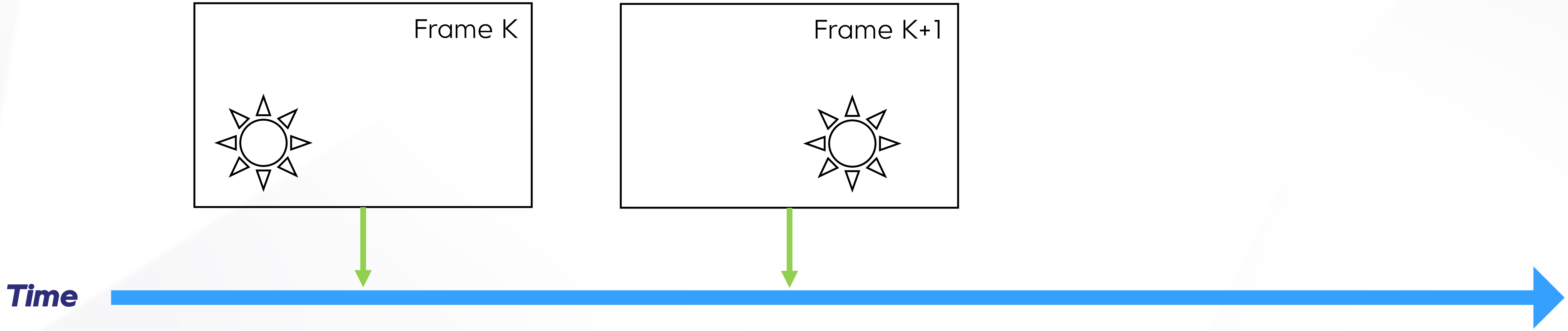
Other solutions?

- Deep learning based solution ×
- Software super-resolution ×
- Frame interpolation?
  - Additional operation response latency ×
  - Interpolation cost ×



Other solutions?

- Deep learning based solution ×
- Software super-resolution ×
- Frame interpolation?
  - Additional operation response latency ×
  - Interpolation cost ×

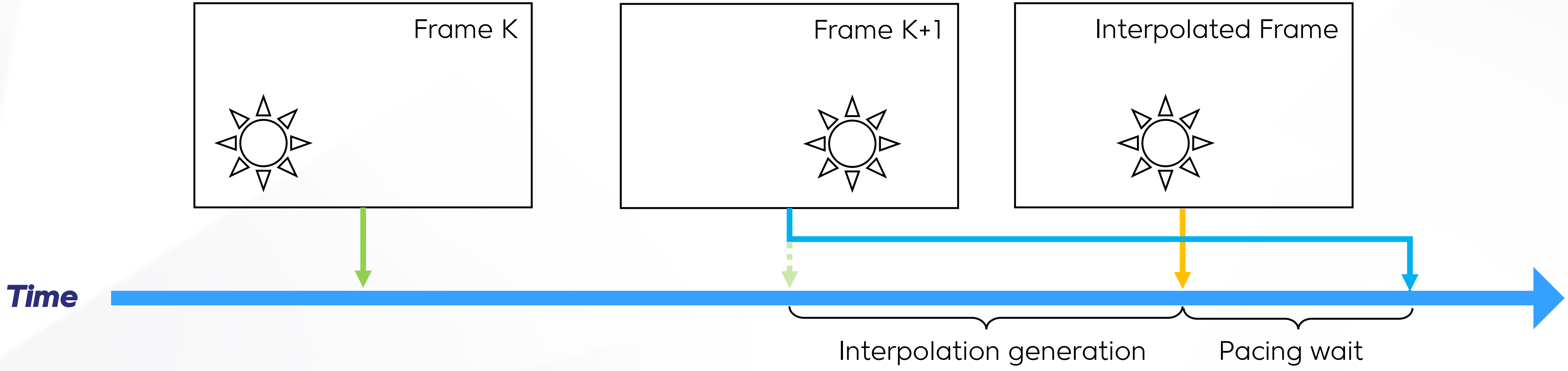


Frame display without frame interpolation

Background

Other solutions?

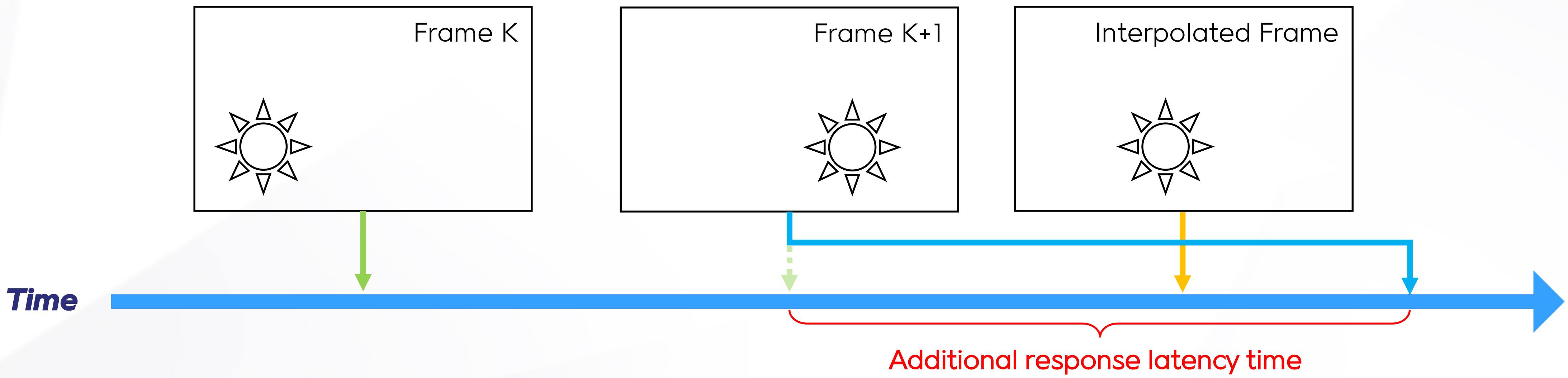
- Deep learning based solution ×
- Software super-resolution ×
- Frame interpolation?
    - Additional operation response latency ×
    - Interpolation cost ×



Frame display with frame interpolation

Other solutions?

- Deep learning based solution ×
- Software super-resolution ×
- Frame interpolation?
  - Additional operation response latency ×
  - Interpolation cost ×



Frame display with frame interpolation

Requirement

A solution is required to increase the frame rate on mobile devices. It should be:

- Simple, fast, no additional latency
- Robust
- Effective and efficient
- Highly compatible (no specific hardware dependence)

02 Frame Prediction

The implementation of frame prediction algorithm

Frame Prediction

Key idea

- Reusing the rendered pixels from the previous frame!
    - Predicted frame = previous frame + gameplay info

Most of rendered pixels from previous frame are reusable

Tencent Games GDC

Key idea

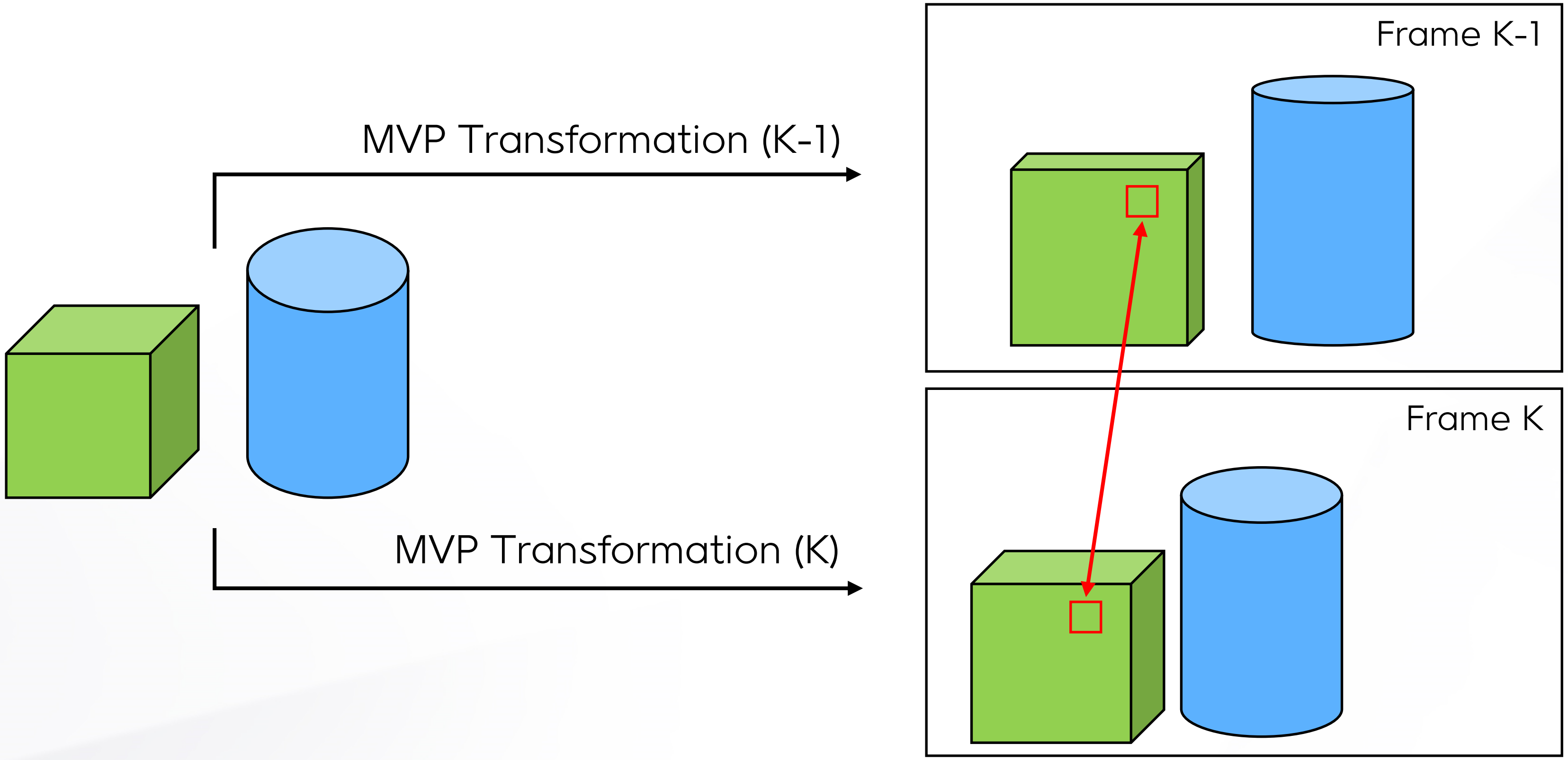
- Reusing the rendered pixels from the previous frame!
- Separate the rendering of dynamic and static objects
  - Motion of dynamic objects' pixels: complex and even unpredictable; needs segmentation
  - Pixels from static objects: can be easily reused through reprojection (majority of all pixels)

Static objects

Static and dynamic objects

Finding Corresponding

- Scene objects are transformed to screen projection plane by MVP Transformation
- Corresponding between frames can be established with reprojection

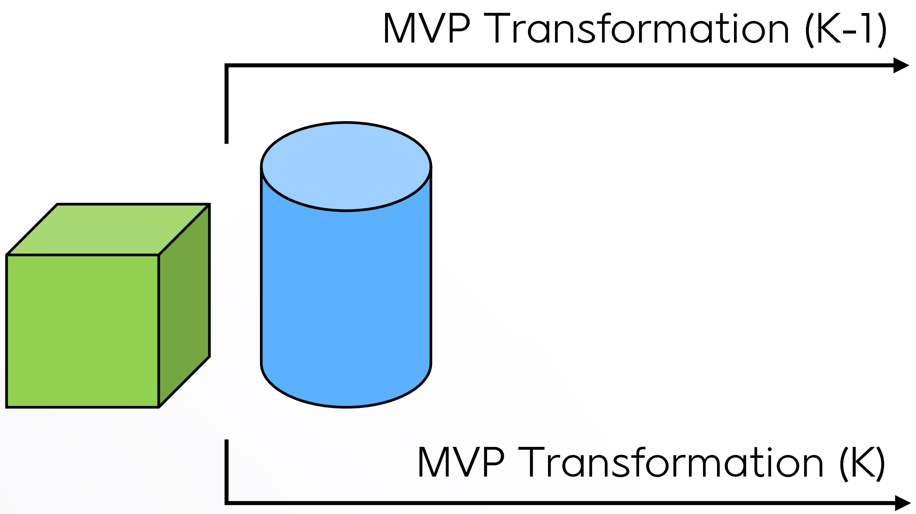


MVP transformation in frames

Frame Prediction

Finding Corresponding

- Scene objects are transformed to screen projection plane by MVP Transformation
- Corresponding between frames can be established with reprojection



$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_{k-1} = P_{k-1}\, V_{k-1}\, M_{k-1} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_{object}$$

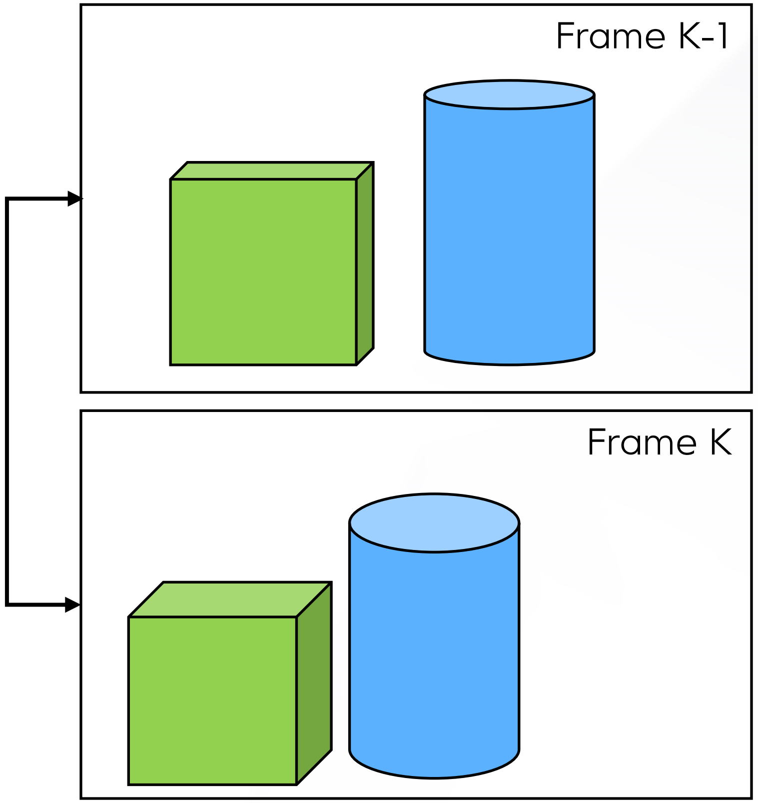
$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_{k} = P_{k}\, V_{k}\, M_{k} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_{object}$$

MVP transformation in frames

Finding Corresponding

- Scene objects are transformed to screen projection plane by MVP Transformation
- Corresponding between frames can be established with reprojection

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_{k-1} = P_{k-1}\, V_{k-1}\, \underbrace{M_{k-1}\, M_k^{-1}}_{=I}\, V_k^{-1}\, P_k^{-1} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_{k}$$



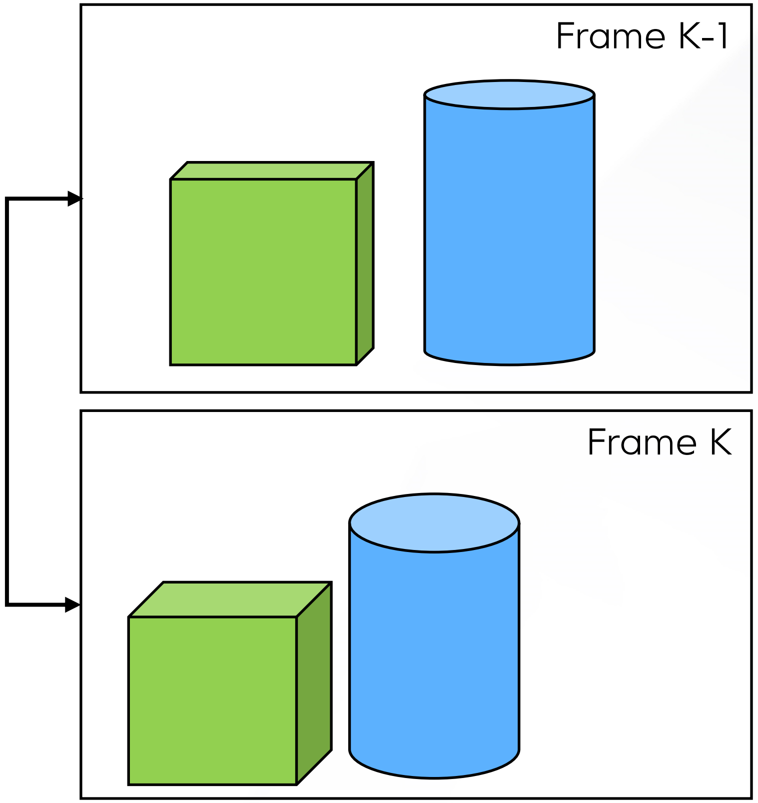
Reprojection between frames

Finding Corresponding

- Scene objects are transformed to screen projection plane by MVP Transformation
- Corresponding between frames can be established with reprojection

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_{k-1} = \boxed{P_{k-1}\, V_{k-1} \underbrace{M_{k-1}\, M_k^{-1}}_{=I} V_k^{-1}\, P_k^{-1}} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_{k}$$

Reprojection matrix



Reprojection between frames

Thinking

- Separate the rendering of dynamic and static objects
  - Motion of dynamic objects: complex and even unpredictable; needs segmentation
  - Static objects: can be easily reused by reprojection
- Per-pixel reprojection?
  - From frame k to frame k-1: missed z-component (depth) without rendering ×

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_{k-1} = P_{k-1} V_{k-1} \underbrace{M_{k-1} M_k^{-1}}_{=I} V_k^{-1} P_k^{-1} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_k$$

None!

Reprojection from frame k to frame k-1

Frame Prediction

Thinking

- Separate the rendering of dynamic and static objects
  - Motion of dynamic objects: complex and even unpredictable; needs segmentation
  - Static objects: can be easily reused by reprojection
- Per-pixel reprojection?
  - From frame k to frame k-1: missed z-component (depth) without rendering ×

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_k = P_k \, V_k \, \underbrace{M_k \, M_{k-1}^{-1}}_{=I} \, V_{k-1}^{-1} \, P_{k-1}^{-1} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_{k-1}$$

Reprojection from frame k-1 to frame k

Frame Prediction

Crack and overlap of per-pixel reprojection

Frame Prediction

Crack and overlap of per-pixel reprojection

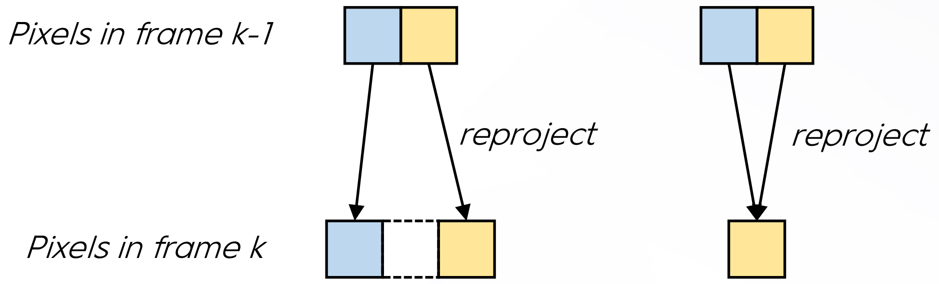
Frame Prediction

Thinking

- Separate the rendering of dynamic and static objects
  - Motion of dynamic objects: complex and even unpredictable; needs segmentation
  - Static objects: can be easily reused by reprojection
- Per-pixel reprojection?
  - From frame k to frame k-1: missed z-component (depth) without rendering ×
  - From frame k-1 to frame k: crack and overlap ×

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_k = P_k\, V_k\, \underbrace{M_k\, M_{k-1}^{-1}}_{=I}\, V_{k-1}^{-1}\, P_{k-1}^{-1} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_{k-1}$$

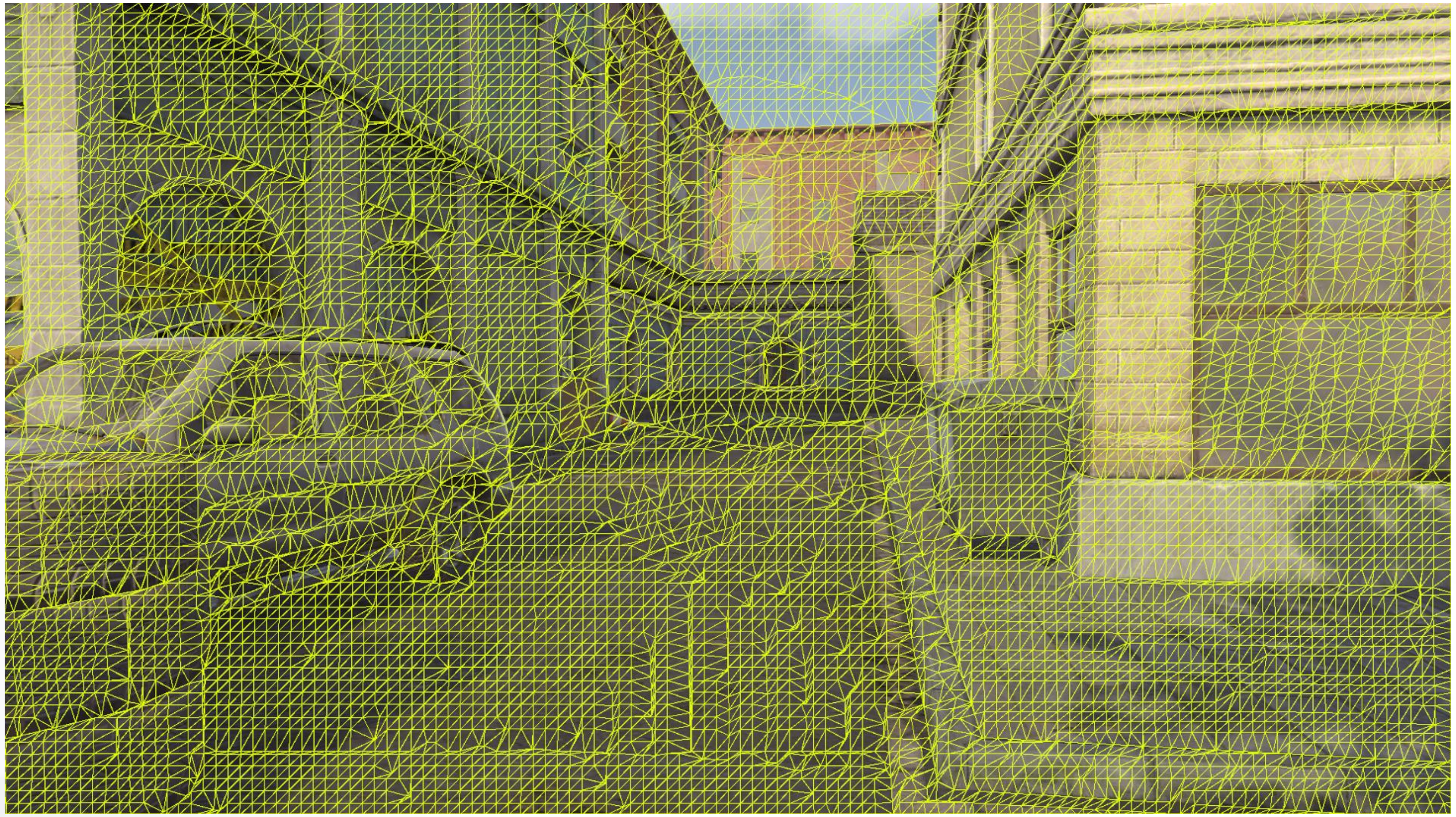
Reprojection from frame k-1 to frame k



Crack and overlap

Solution: per-vertex reprojection

- Key idea: reconstruct the mesh in screen space and reproject to generate the predicted frame
- This algorithm is called *Mesh Projection Estimation* within the project team of Arena Breakout.



Reconstructed mesh

Solution: per-vertex reprojection

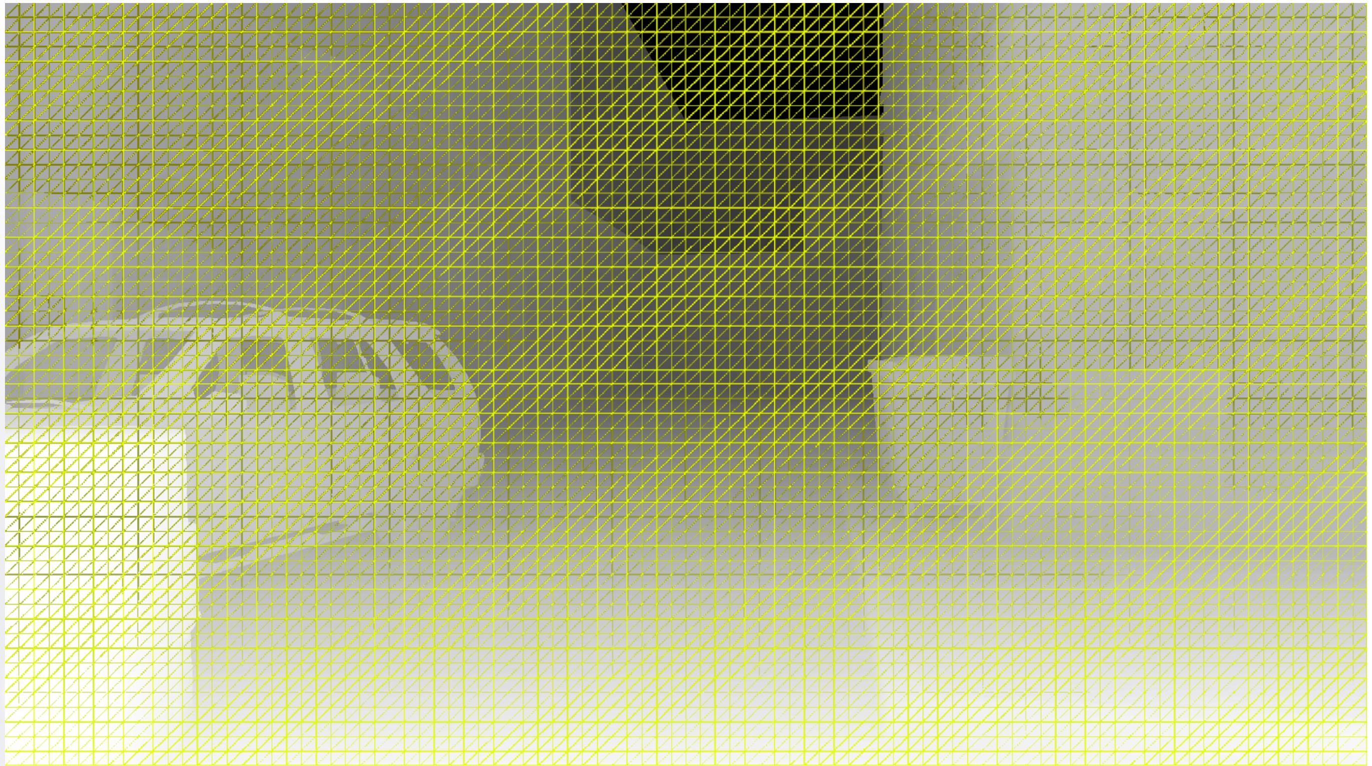
- Key idea: reconstruct the mesh in screen space and reproject to generate the predicted frame
  - Step 1: Find the pixels on the SceneDepth where the vertices are most likely to exist

Cached SceneDepth

Tencent Games GDC

Solution: per-vertex reprojection

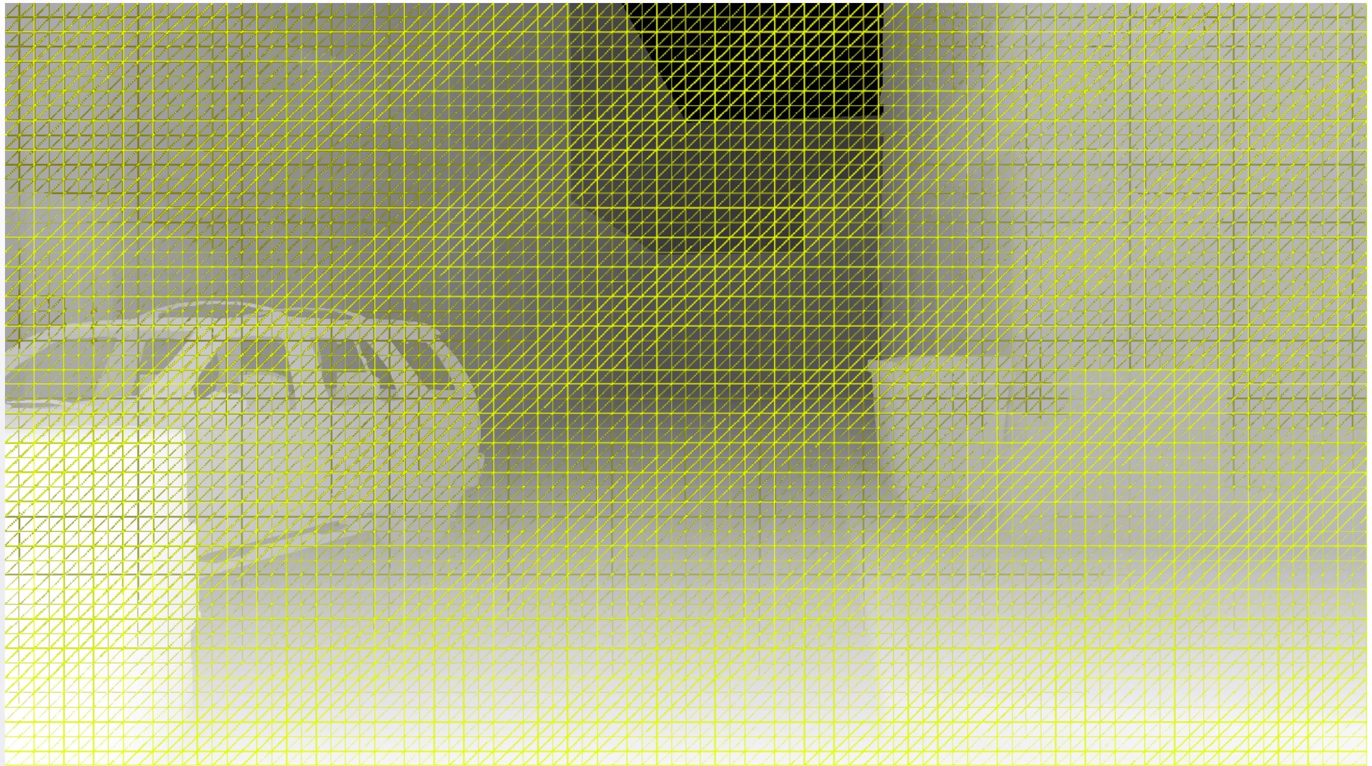
- Key idea: reconstruct the mesh in screen space and reproject to generate the predicted frame
  - Step 1: Find the pixels on the SceneDepth where a vertex is most likely to exist
    - Every n×n pixels as a tile (with n between 8 and 16)



Initial vertices state (upper left of Tile)

Solution: per-vertex reprojection

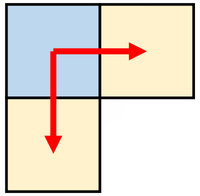
- Key idea: reconstruct the mesh in screen space and reproject to generate the predicted frame
  - Step 1: Find the pixels on the SceneDepth where a vertex is most likely to exist
    - Every n×n pixels as a tile (with n between 8 and 16)
    - Search for the pixel with the maximum sum of squared gradients within tiles, which is considered the most likely location of the vertex.



Initial vertices state (upper left of Tile)

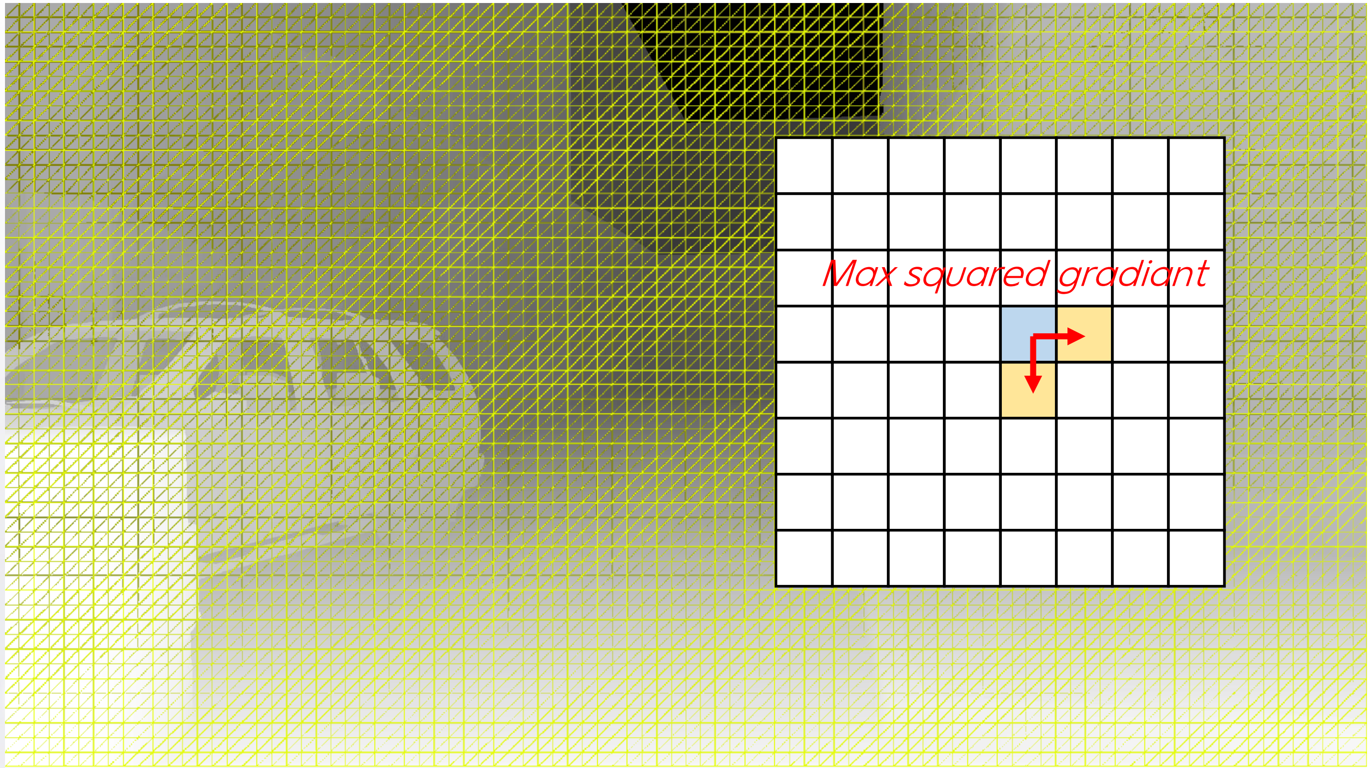
$$\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2$$

$$\left(Depth_{x+1,y} - Depth_{x,y}\right)^2 + \left(Depth_{x,y+1} - Depth_{x,y}\right)^2$$



Solution: per-vertex reprojection

- Key idea: reconstruct the mesh in screen space and reproject to generate the predicted frame
  - Step 1: Find the pixels on the SceneDepth where a vertex is most likely to exist
    - Every n×n pixels as a tile (with n between 8 and 16)
    - Search for the pixel with the maximum sum of squared gradients within tiles, which is considered the most likely location of the vertex.



Initial vertices state (upper left of Tile)

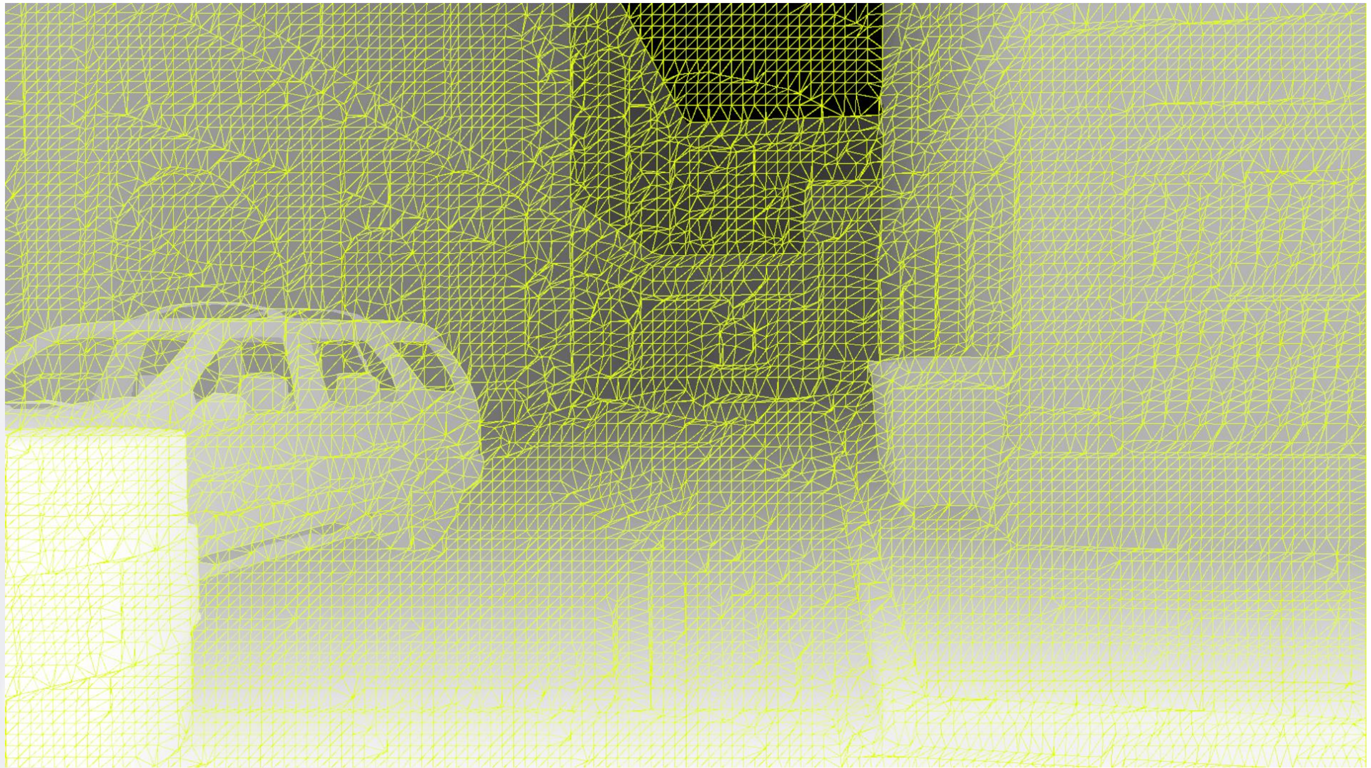
```
void FramePredictionCS()
{
    // Find pixel with max gradiant in tile
    float maxSqSum;
    uint2 maxGradientPixelPos;
    foreach (pixel in Tile)
    {
        if (SqSum(pixel.gradiant) > maxSqSum)
        {
            maxGradientPixelPos = pixel.pos;
        }
    }

    // ...
}
```

Pseudo-code for searching vertices

Frame Prediction

Solution: per-vertex reprojection

- Key idea: reconstruct the mesh in screen space and reproject to generate the predicted frame
  - Step 1: Find the pixels on the SceneDepth where a vertex is most likely to exist
    - Every n×n pixels as a tile (with n between 8 and 16)
    - Search for the pixel with the maximum sum of squared gradients within tiles, which is considered the most likely location of the vertex.



Move vertices to position with max depth gradient in tile

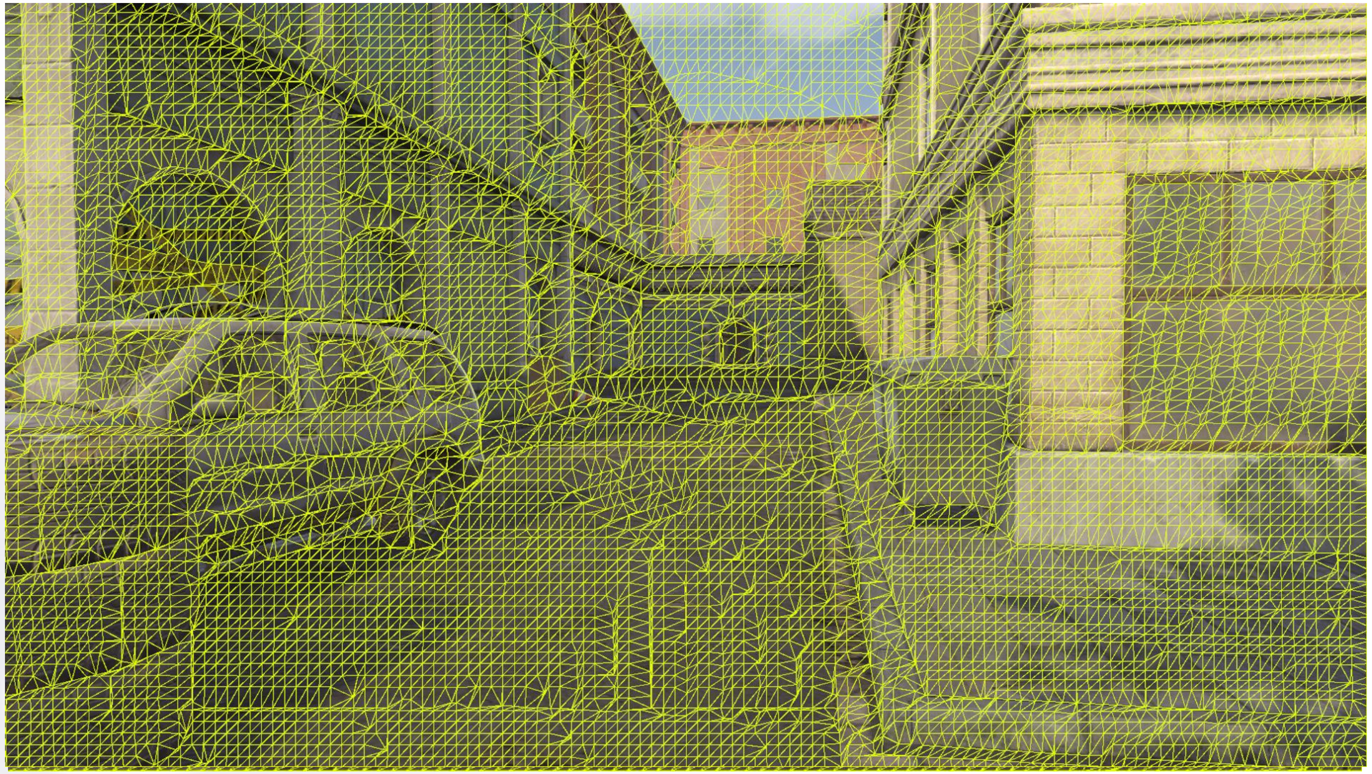
```
void FramePredictionCS()
{
    // Find pixel with max gradiant in tile
    float maxSqSum;
    uint2 maxGradientPixelPos;
    foreach (pixel in Tile)
    {
        if (SqSum(pixel.gradiant) > maxSqSum)
        {
            maxGradientPixelPos = pixel.pos;
        }
    }

    // ...
}
```

Pseudo-code for searching vertices

Frame Prediction

Solution: per-vertex reprojection

- Key idea: reconstruct the mesh in screen space and reproject to generate the predicted frame
  - Step 1: Find the pixels on the SceneDepth where a vertex is most likely to exist
    - Every n×n pixels as a tile (with n between 8 and 16)
    - Search for the pixel with the maximum sum of squared gradients within tiles, which is considered the most likely location of the vertex.



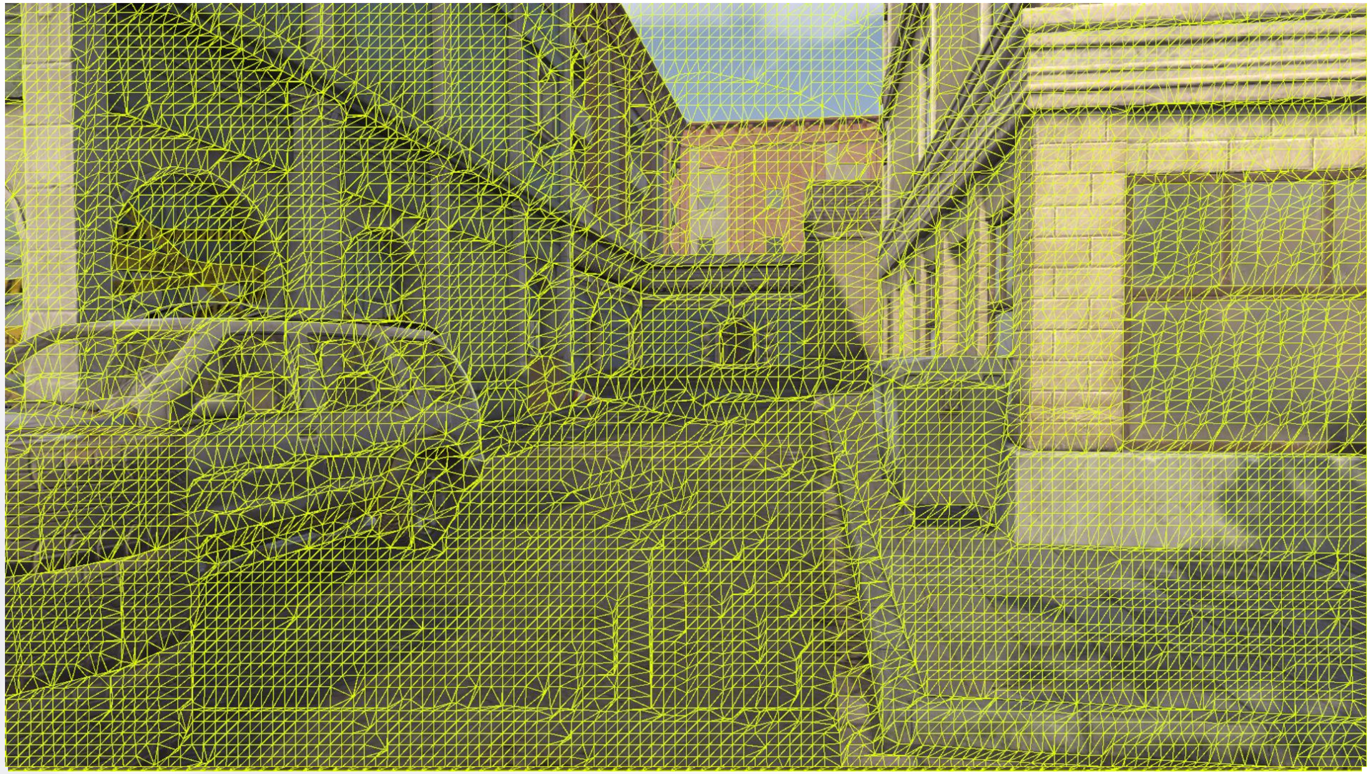
View vertices in SceneColor

```
void FramePredictionCS()
{
    // Find pixel with max gradiant in tile
    float maxSqSum;
    uint2 maxGradientPixelPos;
    foreach (pixel in Tile)
    {
        if (SqSum(pixel.gradiant) > maxSqSum)
        {
            maxGradientPixelPos = pixel.pos;
        }
    }

    // ...
}
```

Pseudo-code for searching vertices

Solution: per-vertex reprojection

- Key idea: reconstruct the mesh in screen space and reproject to generate the predicted frame
  - Step 1: Find the pixels on the SceneDepth where a vertex is most likely to exist
  - Step 2: Reproject the reconstructed vertices and cache position and UV



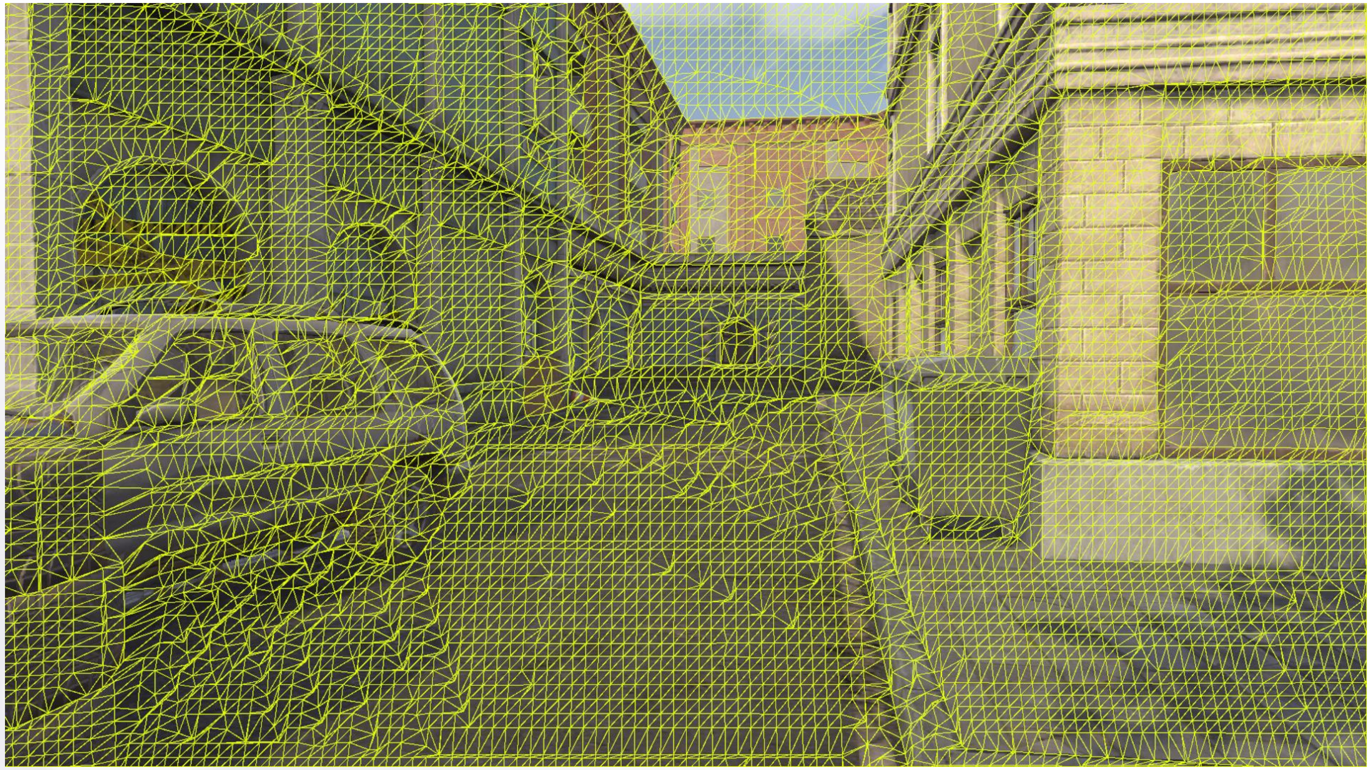
View vertices in SceneColor

```
void FramePredictionCS()
{
    // Find pixel with max gradiant in tile
    {...}

    // Reproject the clip-space position of vertices
    float4 reprojPos = mul(
        float3(maxGradientPixel.ClipPos, 1.0f),
        ReprojectionMatrix
    );
    reprojPos.xyzw /= reprojPos.w;

    CacheUAV[vertPos] =
        float4(reprojPos.xy, maxGradientPixel.uv);
}
```

Pseudo-code for reprojection

Solution: per-vertex reprojection

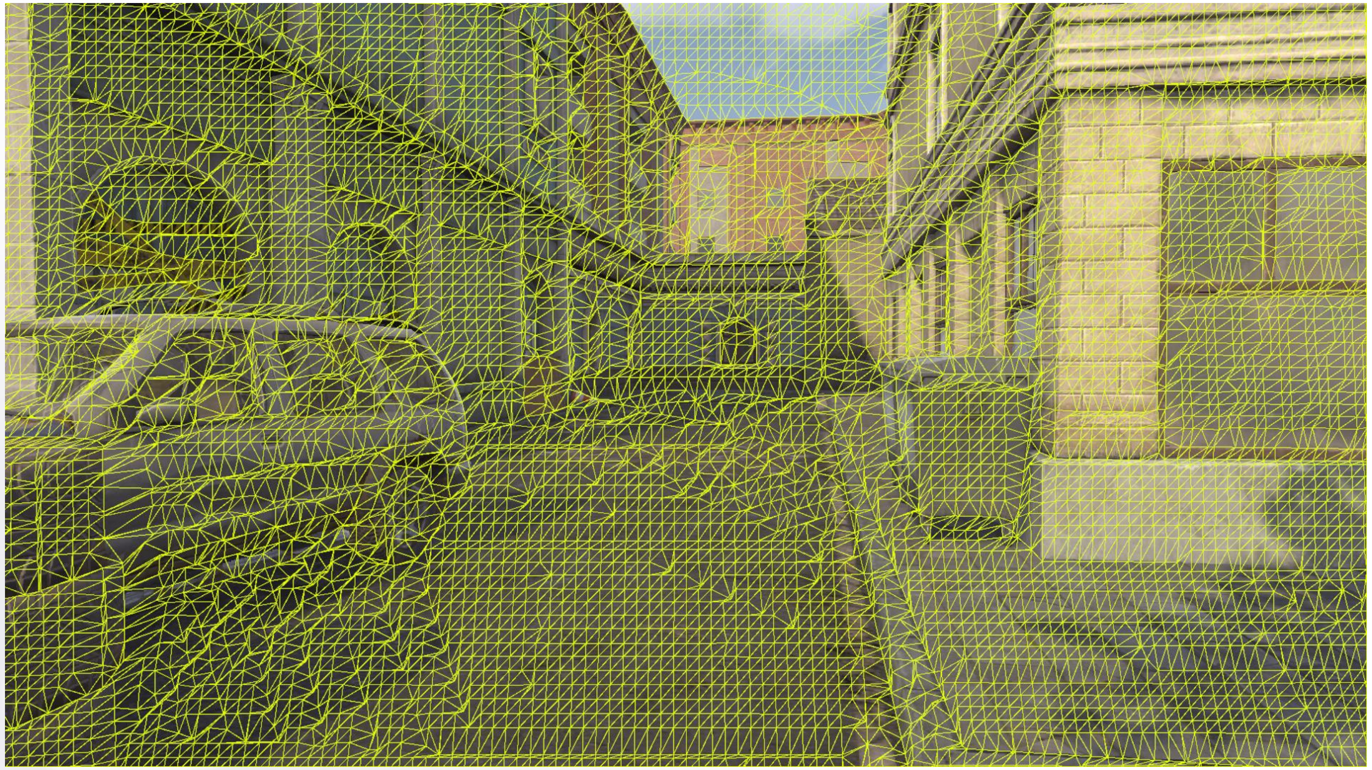
- Key idea: reconstruct the mesh in screen space and reproject to generate the predicted frame
    - Step 1: Find the pixels on the SceneDepth where a vertex is most likely to exist
    - Step 2: Reproject the reconstructed vertices and cache position and UV
    - Step 3: Redraw the reconstructed screen space aggregated mesh (SSAM) to generate predicted frame



Redrawn SSAM (camera moves forward)

Solution: per-vertex reprojection

- Key idea: reconstruct the mesh in screen space and reproject to generate the predicted frame
  - Step 1: Find the pixels on the SceneDepth where a vertex is most likely to exist
  - Step 2: Reproject the reconstructed vertices and cache position and UV
  - Step 3: Redraw the reconstructed screen space aggregated mesh (SSAM) to generate predicted frame



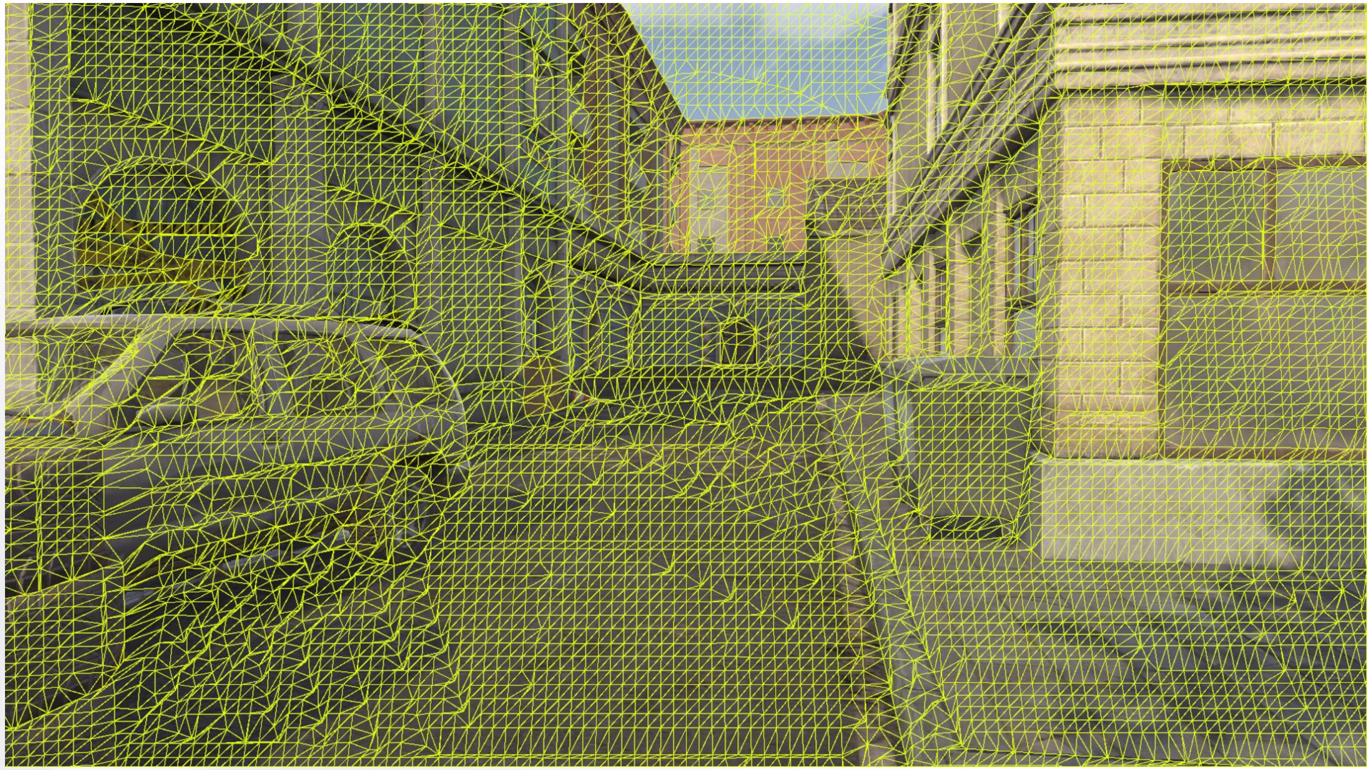
Redrawn SSAM (camera moves forward)

```
void FramePredictionVS(uint VertexID)
{
    uint2 VertPos = CalcPos(VertexID);
    OutVertClip = float3(CacheUAV[VertPos].xy, 1);
    OutVertUV = CacheUAV[VertPos].zw;

    // Anchored screen edge vertices
    if (VertPos.x == 0 or CacheUAV.Size.x)
    {
        OutVertClip.x = -1 or 1;
        OutVertUV.x = 0 or 1;
    }
    if (VertPos.y == 0 or CacheUAV.Size.y)
    {
        OutVertClip.y = -1 or 1;
        OutVertUV.y = 0 or 1;
    }
}
```

Pseudo-code of redrawn vertex shader

Solution: per-vertex reprojection

- Key idea: reconstruct the mesh in screen space and reproject to generate the predicted frame
  - Step 1: Find the pixels on the SceneDepth where a vertex is most likely to exist
  - Step 2: Reproject the reconstructed vertices and cache position and UV
  - Step 3: Redraw the reconstructed screen space aggregated mesh (SSAM) to generate predicted frame



Redrawn SSAM (camera moves forward)

```
void FramePredictionPS(float3 InSvPos, float2 InUV)
{
    // Reproject Depth for higher precision
    float prevDepth = CachedDepthTex.Sample(InUV).x;
    float4 projClip = mul(
        ToClipPos(InUV.xy, prevDepth, 1.0f),
        ReprojectionMatrix
    );
    projClip.xyzw /= projClip.w;

    OutDepth = ToDepth(projClip.z); //current depth
    OutColor = CachedColorTex.Sample(InUV).xyz;
}
```

Pseudo-code of redrawn pixel shader

Frame Prediction

Solution: per-vertex reprojection

- Key idea: reconstruct the mesh in screen space and reproject to generate the predicted frame
    - Step 1: Find the pixels on the SceneDepth where a vertex is most likely to exist
    - Step 2: Reproject the reconstructed vertices and cache position and UV
    - Step 3: Redraw the reconstructed screen space aggregated mesh (SSAM) to generate predicted frame

Generated prediction frame

```
void FramePredictionPS(float3 InSvPos, float2 InUV)
{
    // Reproject Depth for higher precision
    float prevDepth = CachedDepthTex.Sample(InUV).x;
    float4 projClip = mul(
        ToClipPos(InUV.xy, prevDepth, 1.0f),
        ReprojectionMatrix
    );
    projClip.xyzw /= projClip.w;

    OutDepth = ToDepth(projClip.z); //current depth
    OutColor = CachedColorTex.Sample(InUV).xyz;
}
```

Pseudo-code of redrawn pixel shader

Solution: per-vertex reprojection

- Key idea: reconstruct the mesh in screen space and reproject to generate the predicted frame
  - Step 1: Find the pixels on the SceneDepth where a vertex is most likely to exist
  - Step 2: Reproject the reconstructed vertices and cache position and UV
  - Step 3: Redraw the reconstructed screen space aggregated mesh (SSAM) to generate predicted frame

Compare with cached SceneColor

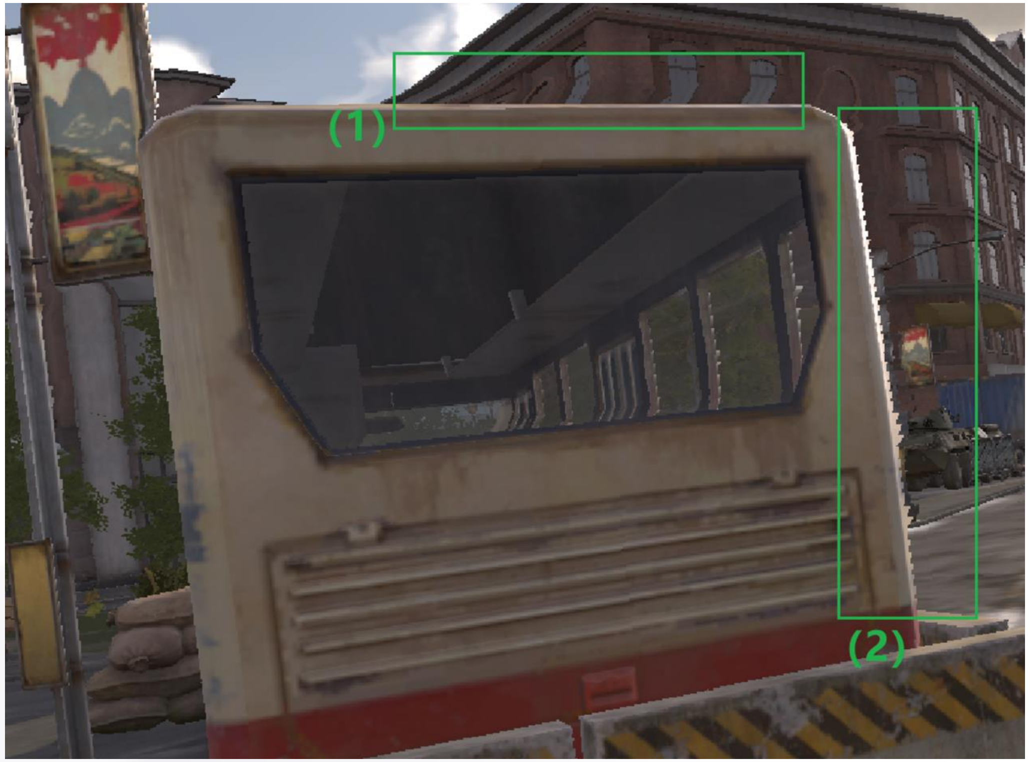
```
void FramePredictionPS(float3 InSvPos, float2 InUV)
{
    // Reproject Depth for higher precision
    float prevDepth = CachedDepthTex.Sample(InUV).x;
    float4 projClip = mul(
        ToClipPos(InUV.xy, prevDepth, 1.0f),
        ReprojectionMatrix
    );
    projClip.xyzw /= projClip.w;

    OutDepth = ToDepth(projClip.z); //current depth
    OutColor = CachedColorTex.Sample(InUV).xyz;
}
```

Pseudo-code of redrawn pixel shader

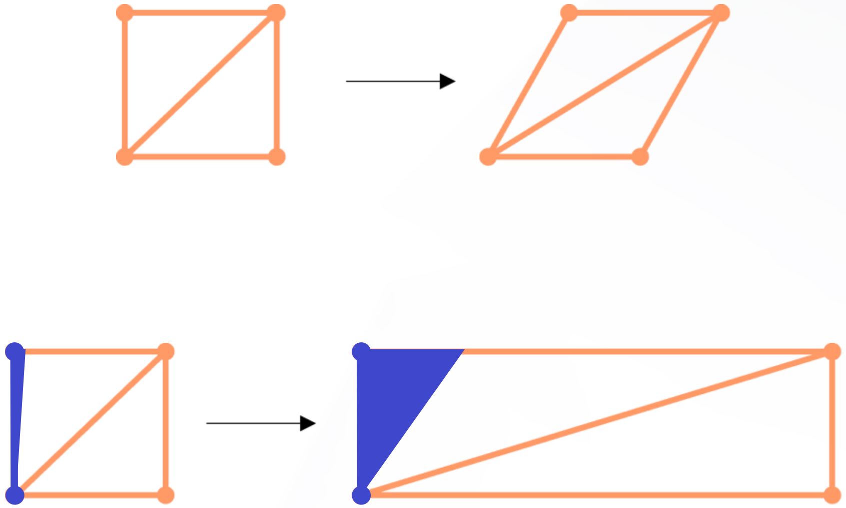
Correction of distortion

Naïve redrawn SSAM could cause:

- Shear distortion: The windows in the background are bent
- Tensile distortion: Bleeding color from the foreground into the background



Shear (1) and tensile (2) distortion



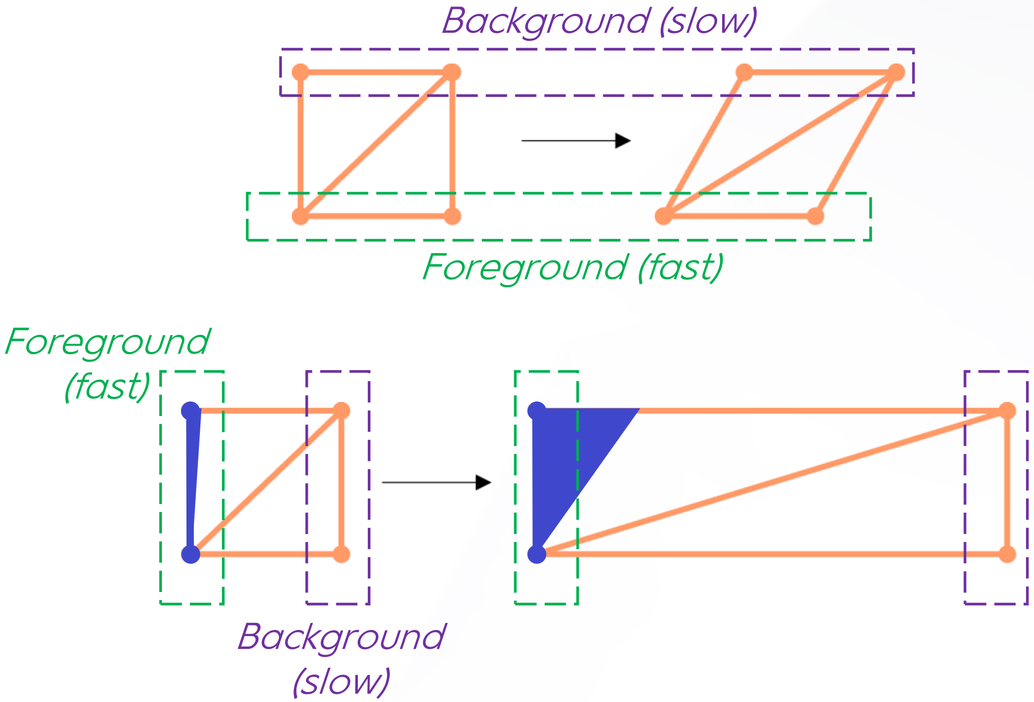
Schematic of distortion

Correction of distortion

Naïve redrawn SSAM could cause:

- Shear distortion: The windows in the background are bent
- Tensile distortion: Bleeding color from the foreground into the background

Shear (1) and tensile (2) distortion

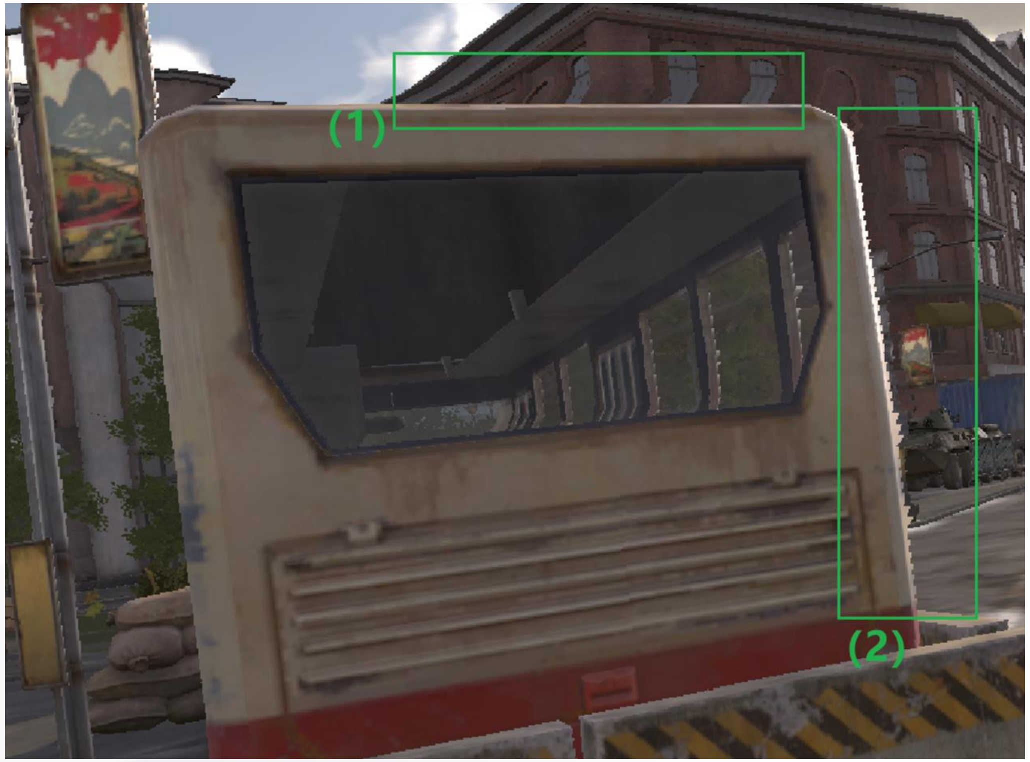


Schematic of distortion

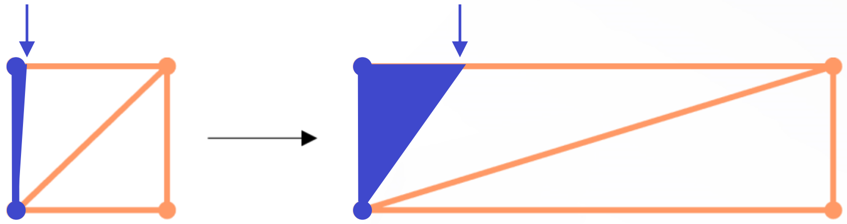
Correction of distortion

Correction of tensile distortion: Compare the relative displacements of neighboring pixels

- If the relative displacement is very small: in the same depth semantic layer
- If the relative displacement is large: in different depth semantic layers, needs correction



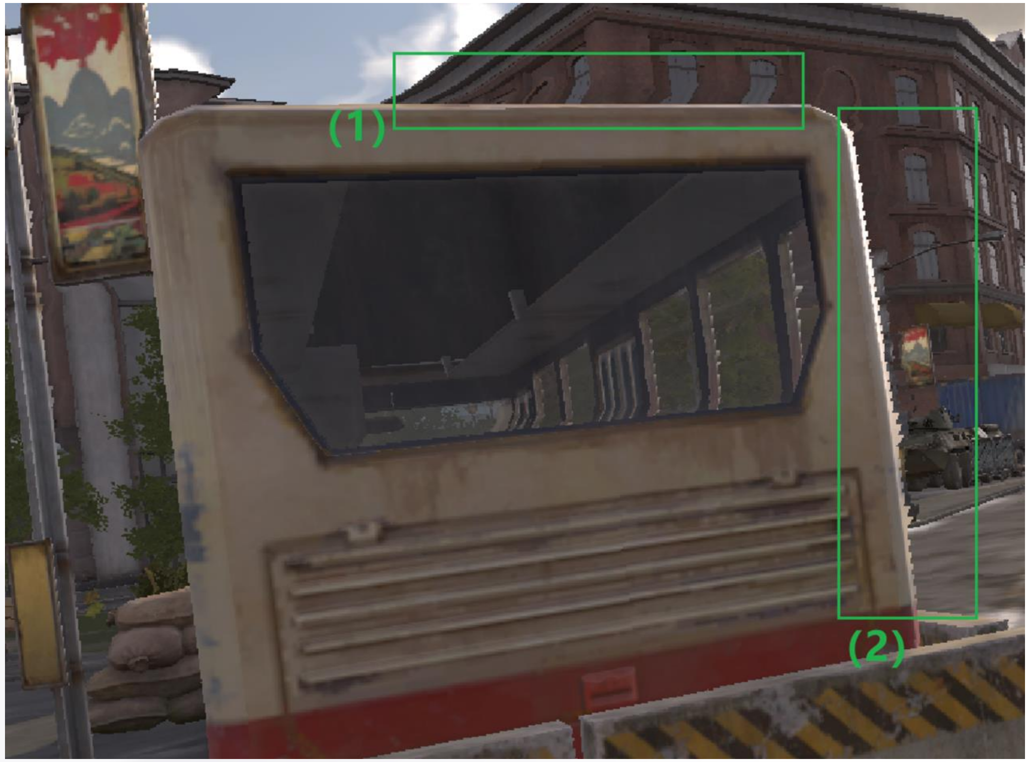
Shear (1) and tensile (2) distortion



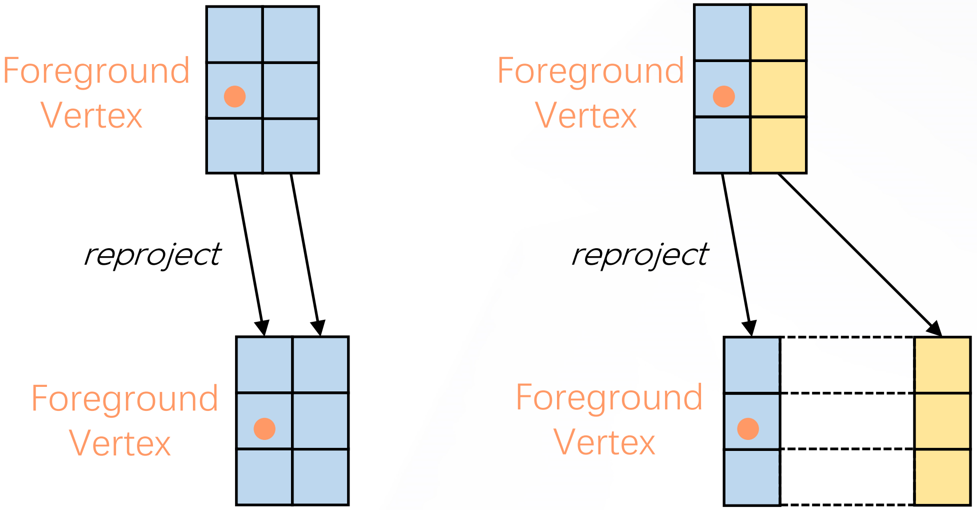
Correction of distortion

Correction of tensile distortion: Compare the relative displacements of neighboring pixels

- If the relative displacement is very small: in the same depth semantic layer
- If the relative displacement is large: in different depth semantic layers, needs correction



Shear (1) and tensile (2) distortion



Small and large relative displacements

Tencent Games GDC

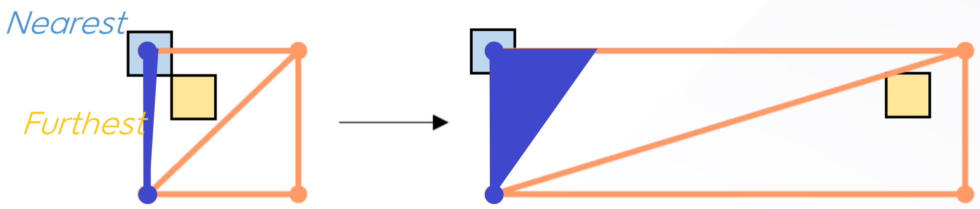
Correction of distortion

Correction of tensile distortion: Compare the relative displacements of neighboring pixels

- If the relative displacement is very small: in the same depth semantic layer
- If the relative displacement is large: in different depth semantic layers, needs correction
    - Solution using a tiny UV-bias: apply the foreground pixel's clip-space position with the background pixel's UV

```
void FramePredictionCS()
{
    // Find pixel with max gradiant in tile
    {...}

    // Find foreground pixel and background pixel
    PixelInfo foregroundPixel, backgroundPixel;
    foreach (neighbors of maxGradientPixel)
    {
        if (CloserThan(neighbor.depth, foregroundPixel.depth))
        {
            foregroundPixel.clipPos = neighbor.clipPos;
            foregroundPixel.uv = neighbor.uv;
        }
        if (FurtherThan(neighbor.depth, backgroundPixel.depth))
        {
            backgroundPixel.clipPos = neighbor.clipPos;
            backgroundPixel.uv = neighbor.uv;
        }
    }

    //...
}
```

Pseudo-code of tensile distortion correction



Schematic of tensile distortion correction

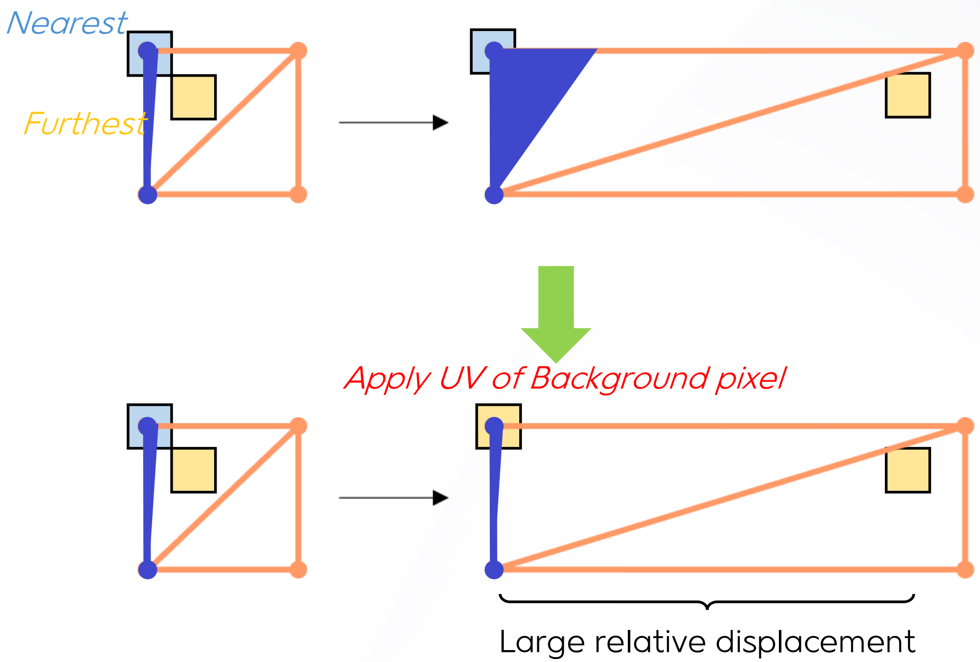
Correction of distortion

Correction of tensile distortion: Compare the relative displacements of neighboring pixels

- If the relative displacement is very small: in the same depth semantic layer
- If the relative displacement is large: in different depth semantic layers, needs correction
  - Solution using a tiny UV-bias: apply the foreground pixel's clip-space position with the background pixel's UV

```
void FramePredictionCS()
{
    // Find pixel with max gradiant in tile
    // Find foreground pixel and background pixel
    {...}

    // Reprojection of foreground pixel and background pixel
    float4 foregroundReprojPos = mul(
        float3(foregroundPixel.clipPos, 1.0f),
        ReprojectionMatrix);
    foregroundReprojPos.xyzw /= foregroundReprojPos.w;

    float4 backgroundReprojPos = mul(
        float3(backgroundPixel.clipPos, 1.0f),
        ReprojectionMatrix);
    backgroundReprojPos.xyzw /= backgroundReprojPos.w;

    if (SameRelativeDir(foregroundPixel.clipPos - backgroundPixel.clipPos,
     foregroundReprojPos.xy - backgroundReprojPos.xy) &&
     abs(foregroundReprojPos.xy - backgroundReprojPos.xy) > Threshold)
    {
        foregroundPixel.uv = backgroundPixel.uv; //tiny UV-bias
    }

    CacheUAV[vertPos] = float4(foregroundReprojPos.xy, foregroundPixel.uv);
}
```

Pseudo-code of tensile distortion correction



Schematic of tensile distortion correction

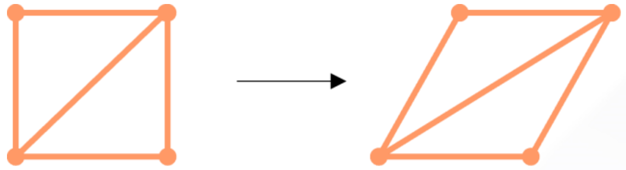
Correction of distortion

Correction of shear distortion: inverse reprojection and compare UV during redrawing

```
void FramePredictionPS(float3 InSvPos, float2 InUV)
{
    // Reproject Depth for higher precision
    float prevDepth = CachedDepthTex.Sample(InUV).x;
    float4 projClip = mul(
        ToClipPos(InUV.xy, prevDepth, 1.0f),
        ReprojectionMatrix
    );
    projClip.xyzw /= projClip.w;

    OutDepth = ToDepth(projClip.z); //current depth
    OutColor = CachedColorTex.Sample(InUV).xyz;
}
```

Got depth in PS

Pseudo-code of shear distortion correction



Schematic of shear distortion correction

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_{k-1} = P_{k-1}\, V_{k-1} \underbrace{M_{k-1}\, M_k^{-1}}_{=I} V_k^{-1}\, P_k^{-1} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}_k$$

Got! (z)

Reprojection from frame k to frame k-1 in pixel shader

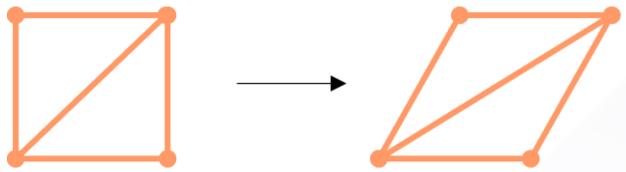
Frame Prediction

Correction of distortion

Correction of shear distortion: inverse reprojection and compare UV during redrawing

```
void FramePredictionPS(float3 InSvPos, float2 InUV)
{
    // Reproject Depth for higher precision
    {...}
    OutDepth = ToDepth(projClip.z); //current depth

    // Inverse reprojection correction
    float2 ScreenUV = ClipToUv(InSvPos.xy);
    float4 InvReprojClipPos = mul(
        ToClipPos(ScreenUV, OutDepth, 1.0f),
        InverseReprojectionMatrix);
    foregroundReprojPos.xyzw /= foregroundReprojPos.w;

    float2 prevUV = ClipToUv(InvReprojClipPos.xy);
    float DepthBeforeReproj = CachedDepthTex.Sample(prevUV).x;

    if (ApproxEqual(prevDepth, DepthBeforeReproj) &&
     abs(InUV - prevUV) > ThresholdValue)
    {
        OutColor = CachedColorTex.Sample(prevUV).xyz;
    }
    else
    {
        OutColor = CachedColorTex.Sample(InUV).xyz;
    }
}
```

Pseudo-code of shear distortion correction



Schematic of shear distortion correction

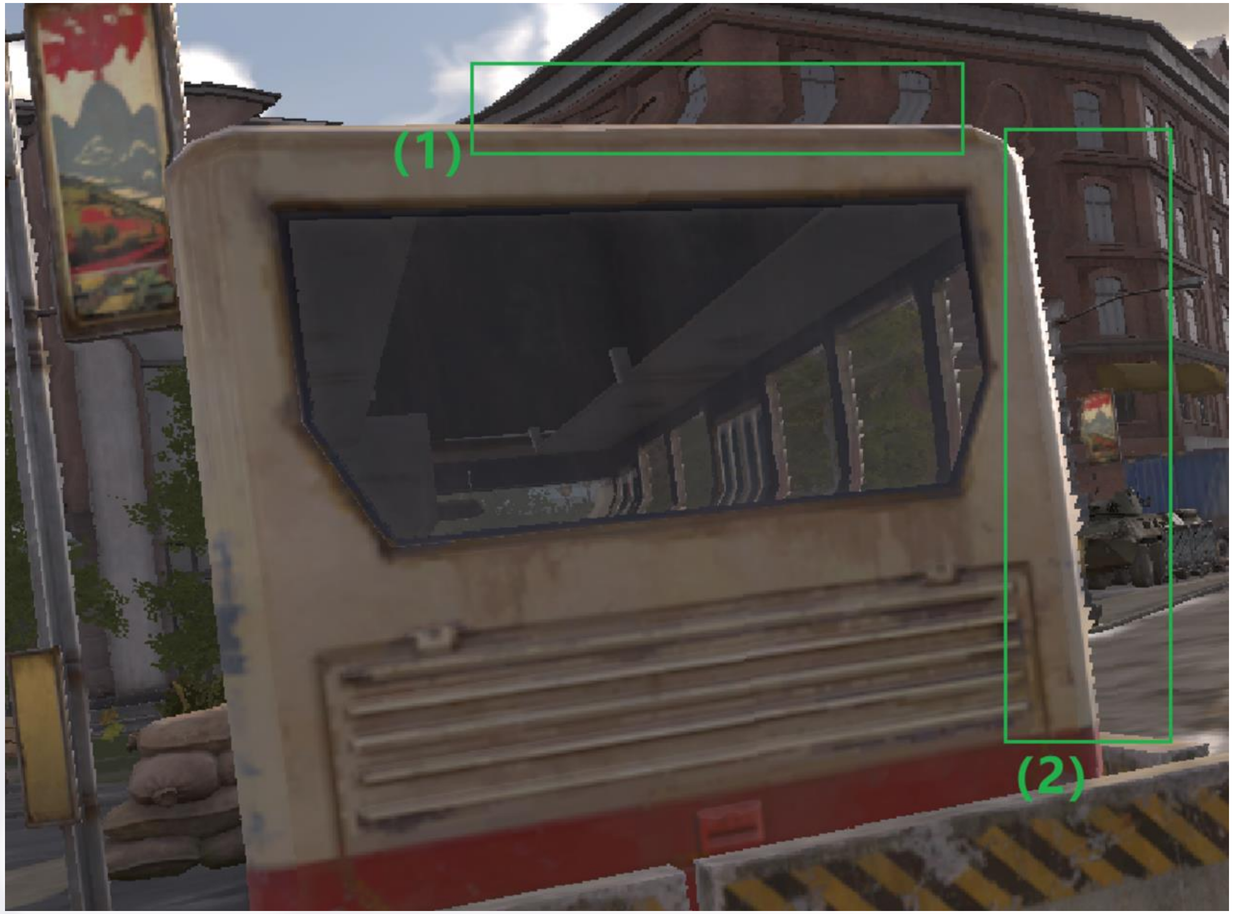
$$\begin{bmatrix}x\\y\\z\\w\end{bmatrix}_{k-1} = P_{k-1}\, V_{k-1}\, \underbrace{M_{k-1}\, M_k^{-1}}_{=I}\, V_k^{-1}\, P_k^{-1} \begin{bmatrix}x\\y\\z\\w\end{bmatrix}_{k}$$

Got!

Reprojection from frame k to frame k-1 in pixel shader

Correction of distortion

Comparison:



Shear (1) and tensile (2) distortion

With corrections

Frame Prediction

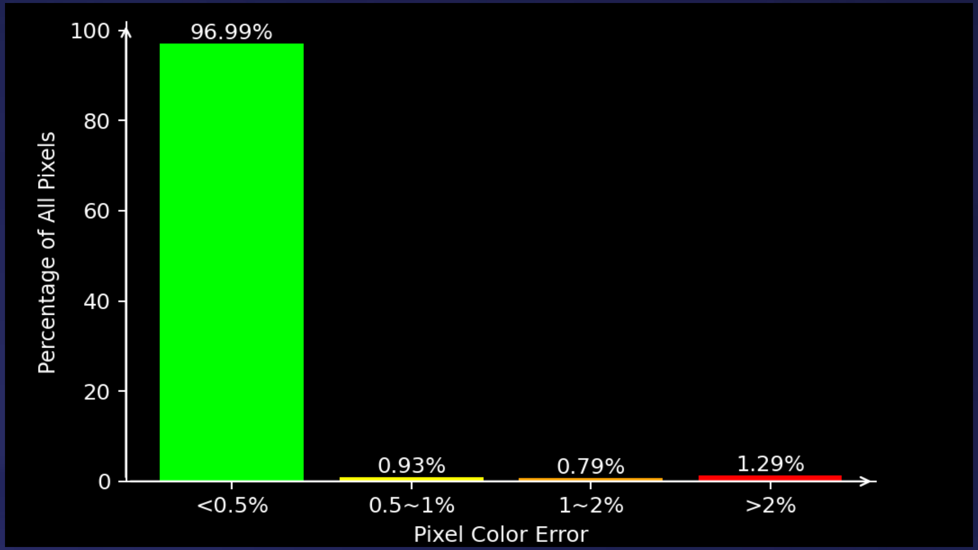
Prediction accuracy analysis: Camera moves forward, DeltaTime = 16.67ms

Frame Prediction

Prediction accuracy analysis: Camera moves forward, DeltaTime = 16.67ms


Generated Frame


Rendered Reference



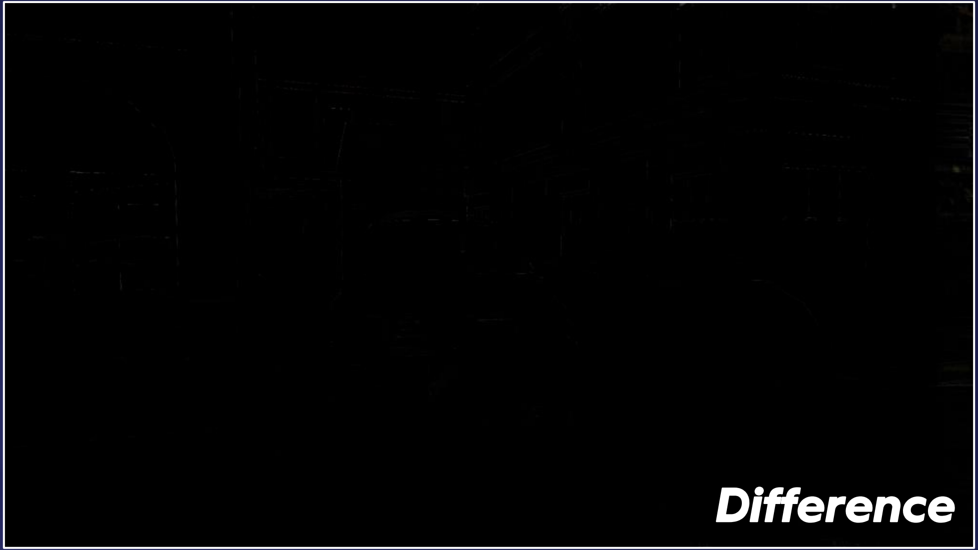

Difference (exaggerated color diff)

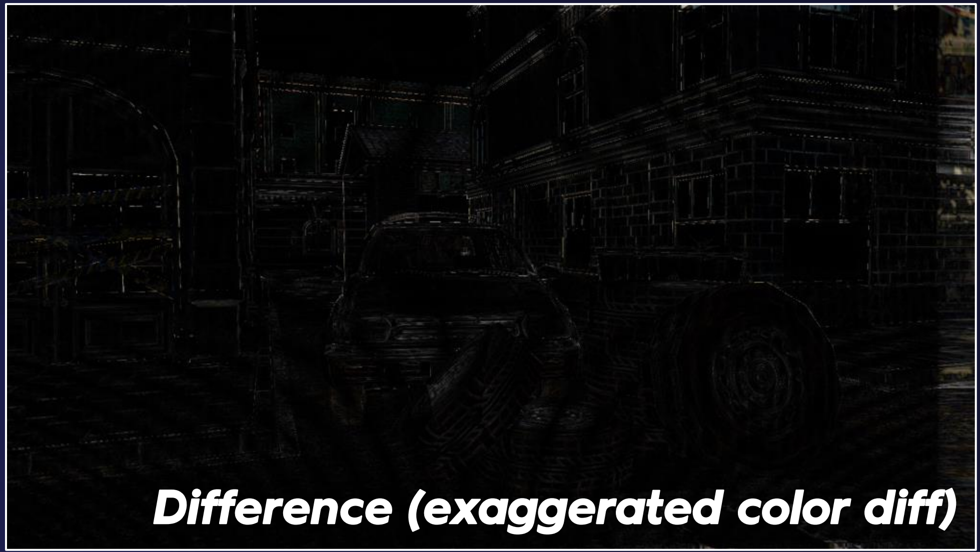
Frame Prediction

Prediction accuracy analysis: Camera rotates to right, DeltaTime = 16.67ms


Generated Frame


Rendered Reference

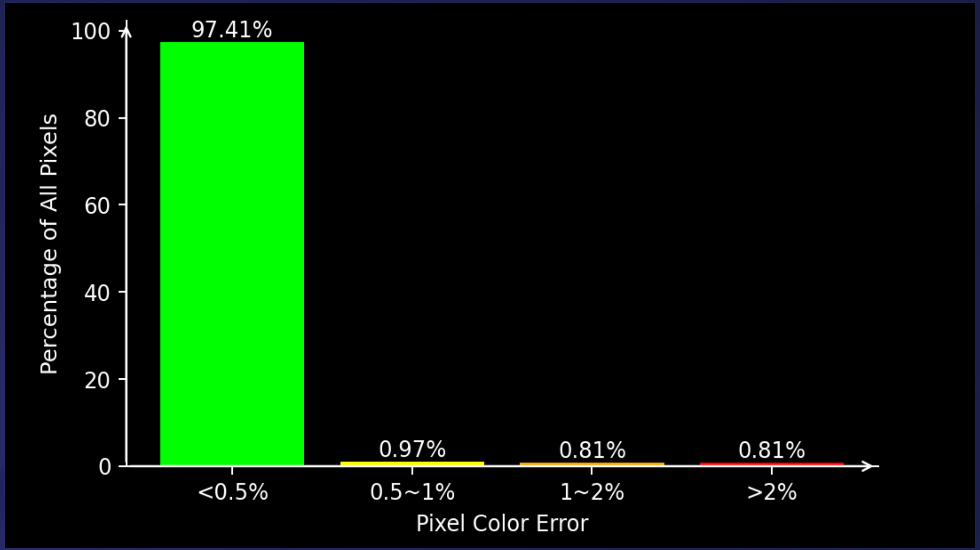

Difference

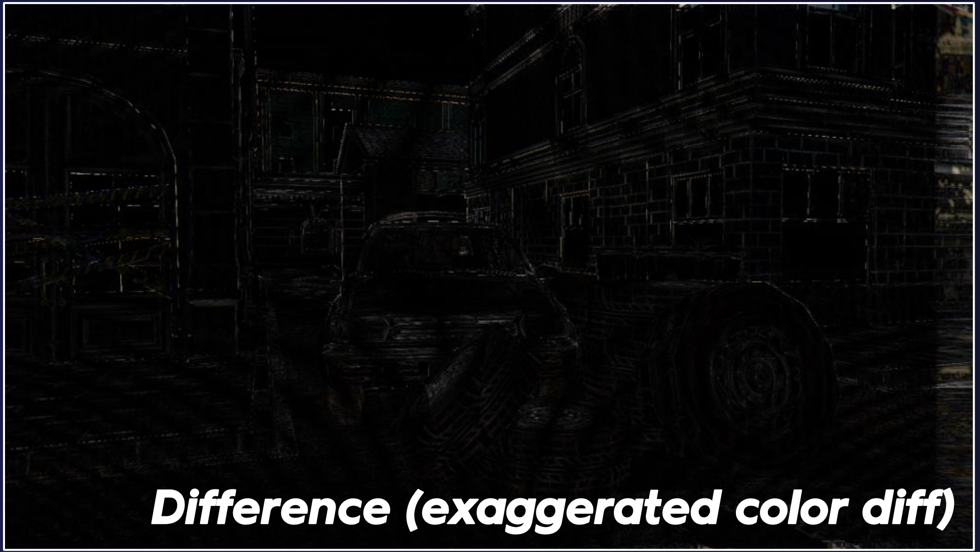

Difference (exaggerated color diff)

Frame Prediction

Tencent Games GDC

Prediction accuracy analysis: Camera rotates to right, DeltaTime = 16.67ms





Correction of missing pixels at the edge of the screen

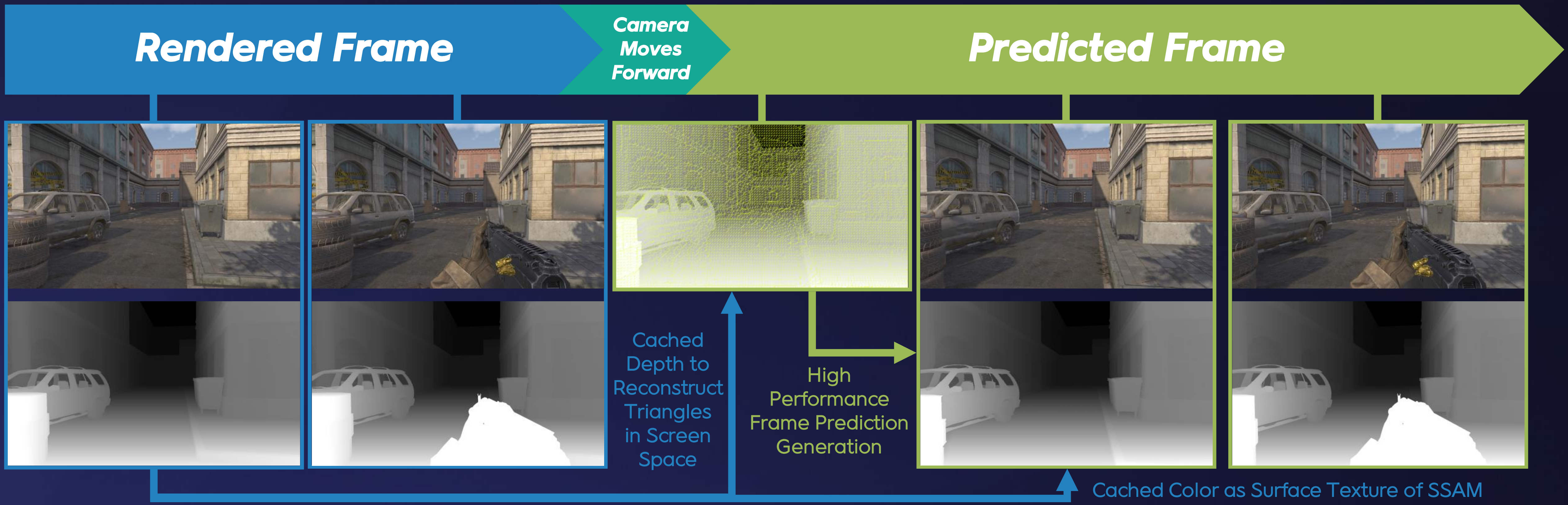
If it's necessary to correct the interpolated pixels through rasterization at the screen edge, we can predict the motion of camera, render additional pixels in the previous frame, and clip when used.

Rendering additional pixels and clip when used

03 Rendering Pipelines

The corresponding rendering pipelines with frame prediction

Structure of Frame Prediction Based Rendering



Draw static scene objects and cache scene color and depth

Draw dynamic objects

Reconstruct the triangles of static objects by cached scene depth, which are combined to form a "Screen Space Aggregated Mesh (SSAM)"

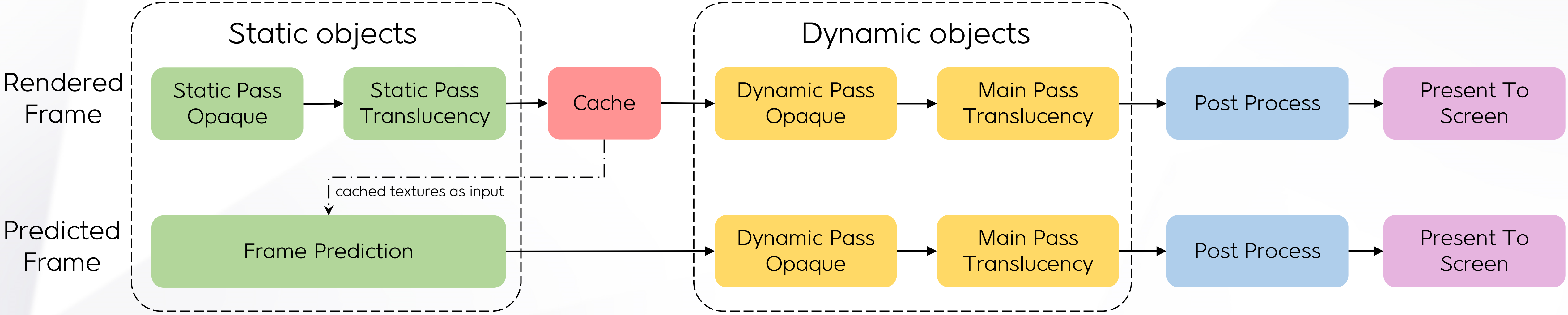
Reproject SSAM by the camera parameters (such as VP matrices) to generate render result of static objects

Draw dynamic objects

Pipeline 1: Rendered and predicted frame in different logic frame

This pipeline pairs every two frames into a "rendered frame-predicted frame" set:

- One logic frame corresponds to one graphic frames
- Perfect game control feel in high frame rate
- Reduce the workload of drawing static objects by half

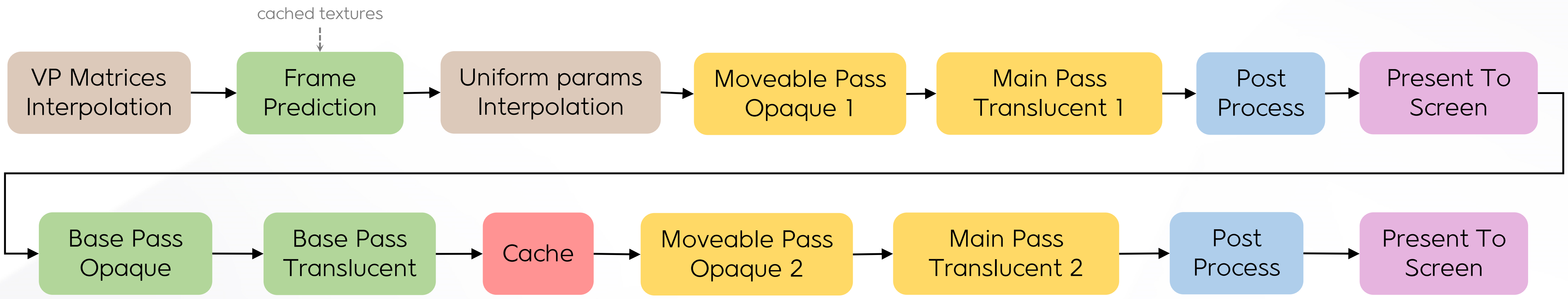


Structure of rendering pipeline 1

Pipeline 2: Rendered and predicted frame in one logic frame

Make intermediate frame by frame prediction (static) and interpolated uniform parameters (dynamic):

- One logic frame corresponds to two graphic frames
- No negative impact on game control feel (rendered frame can still be presented immediately)
- Very high efficiency

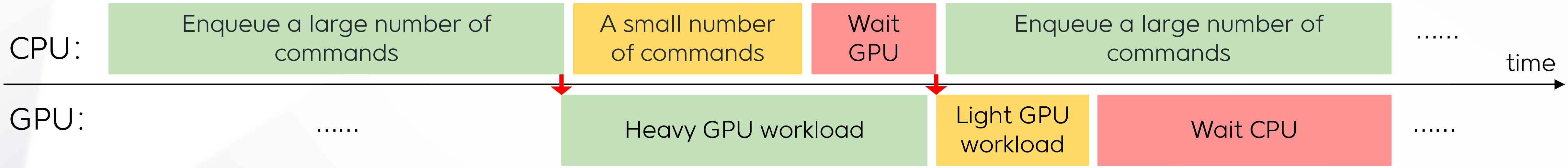


Structure of pipeline with smoothed intermediate frame

Workload balance

The imbalanced frame workload could be inefficient with some device driver strategies (e.g. Qualcomm DCVS)

- CPU and GPU wait each other

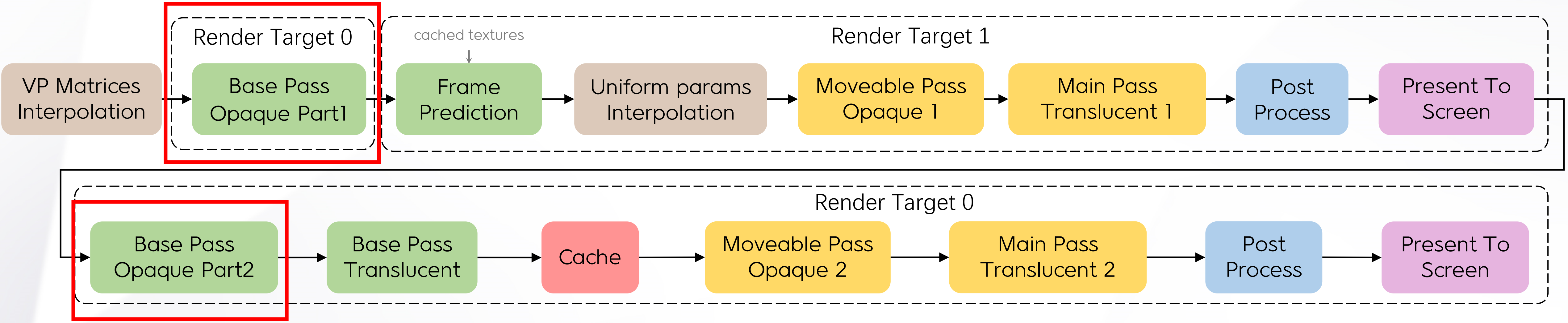


CPU and GPU wait each other

Workload balance

The imbalance frame workload could be inefficient with some device driver strategies (e.g. Qualcomm DCVS)

- CPU and GPU wait each other
- Solution: Split the rendering of base pass and use two render targets —— but additional bandwidth
- Looking forward specific GPU/API optimizations for inhomogeneous workloads



Workload balance for pipeline with smoothed intermediate frame

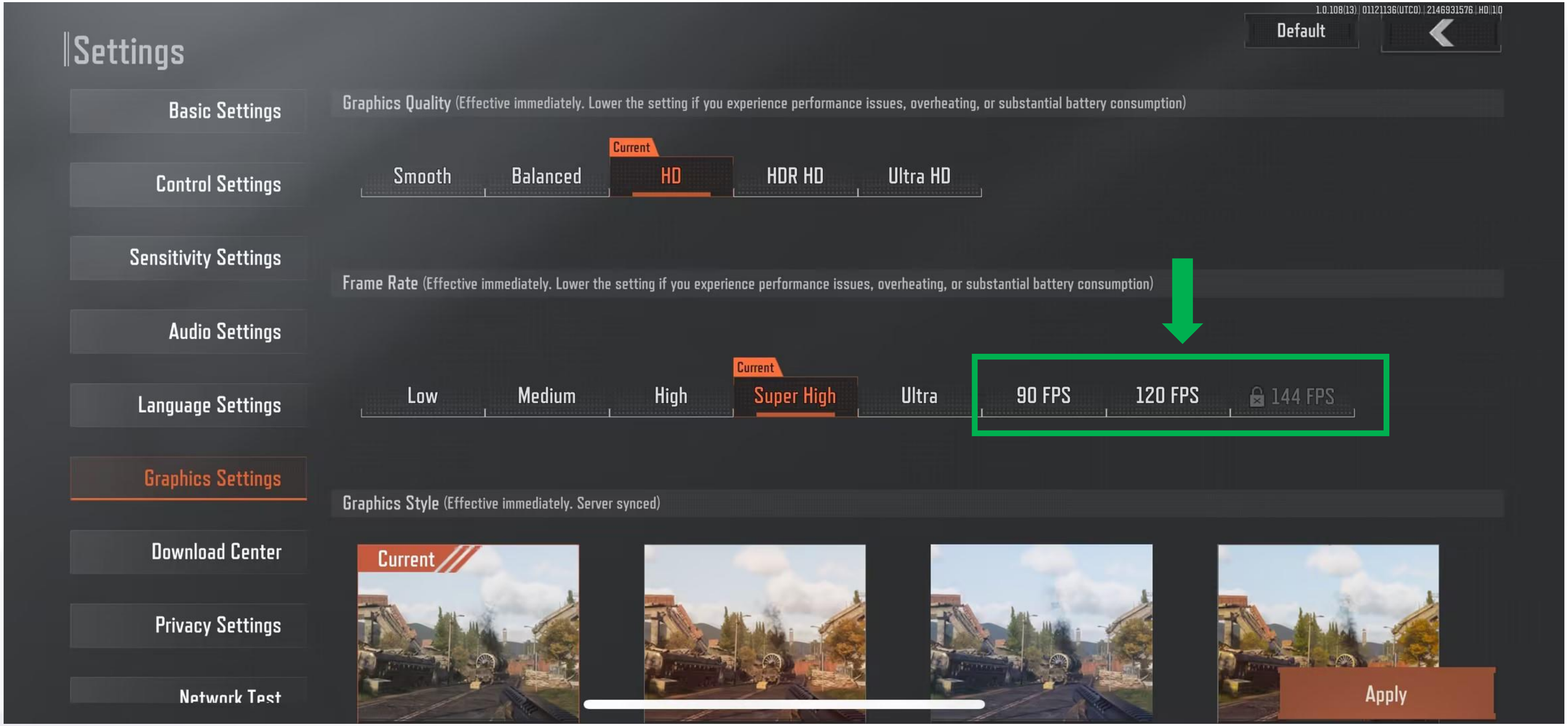
04 Conclusion

Analysis of performance and further applications

Conclusion

Frame Prediction is successfully applied in the released game!

Select 90, 120 and 144 FPS in setting of *Arena Breakout* to activate rendering pipelines with frame prediction!



Analysis of Performance

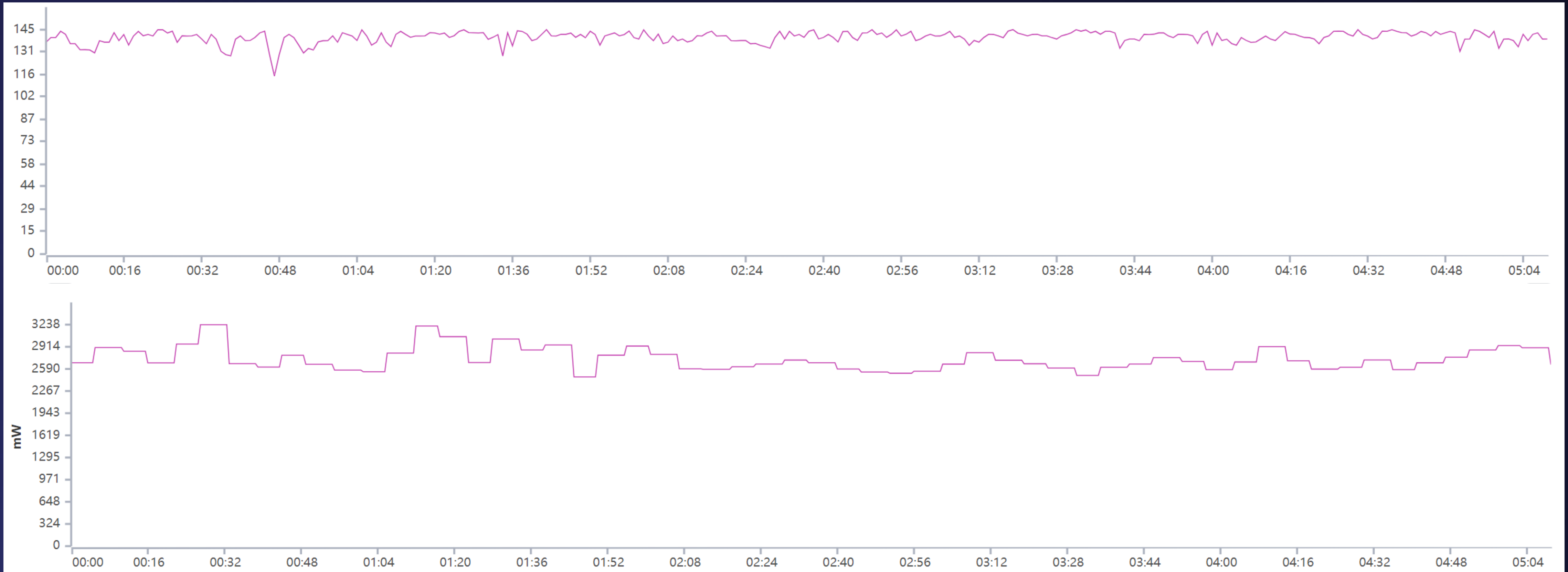
Performance data comparison

With the frame prediction on iPhone 14 Pro, the average frame rate has increased from 97.7 to 118.3 FPS, the surface temperature has been reduced from 40.3°C (104.5°F) to 36.4°C (97.5°F), and the battery power consumption has been reduced by 19%.

Data source: Perfdog

Performance data comparison

With the frame prediction on Android smartphone equipped with Qualcomm Snapdragon 7+ Gen 2, the average frame rate of 720P can reach up to the impressive 140.2FPS.



Data source: Perfdog

Reuse ray's info in mobile ray tracing

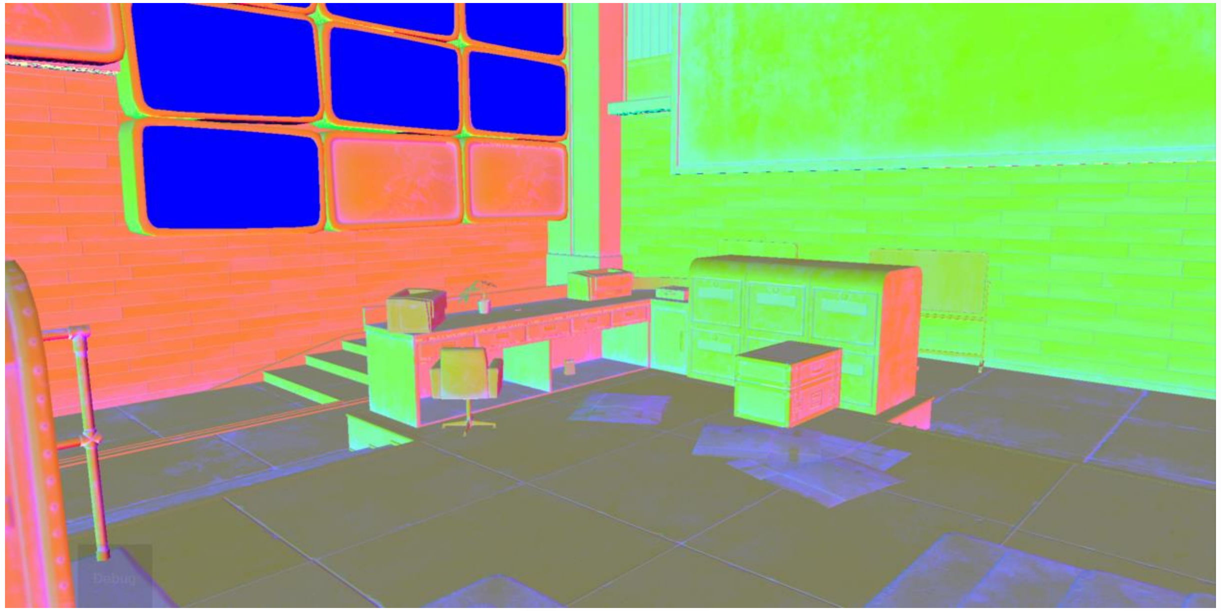
Frame prediction can also be used in mobile ray tracing to reuse the ray-infos in screen space.

Mobile ray tracing OFF

Mobile ray tracing ON

Reuse ray's info in mobile ray tracing

Frame prediction can also be used in mobile ray tracing to reuse the ray-infos in screen space.



Reusable features in screen space

Tencent Games GDC

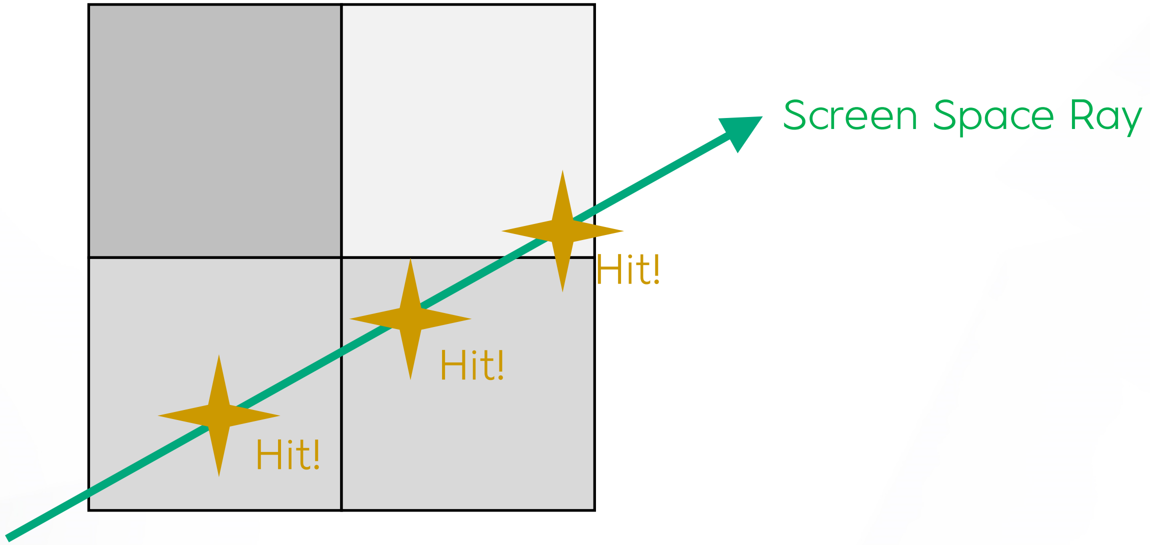
Performance data comparison

With the frame prediction on Android smartphone equipped with Mediatek Dimensity 9300, the average frame rate for mobile ray-tracing reflection has increased from 62.5 to 89.2 FPS, and frame prediction avoids also the overheating protection of chip and the frame rate limitation from OS.

Data source: Perfdog

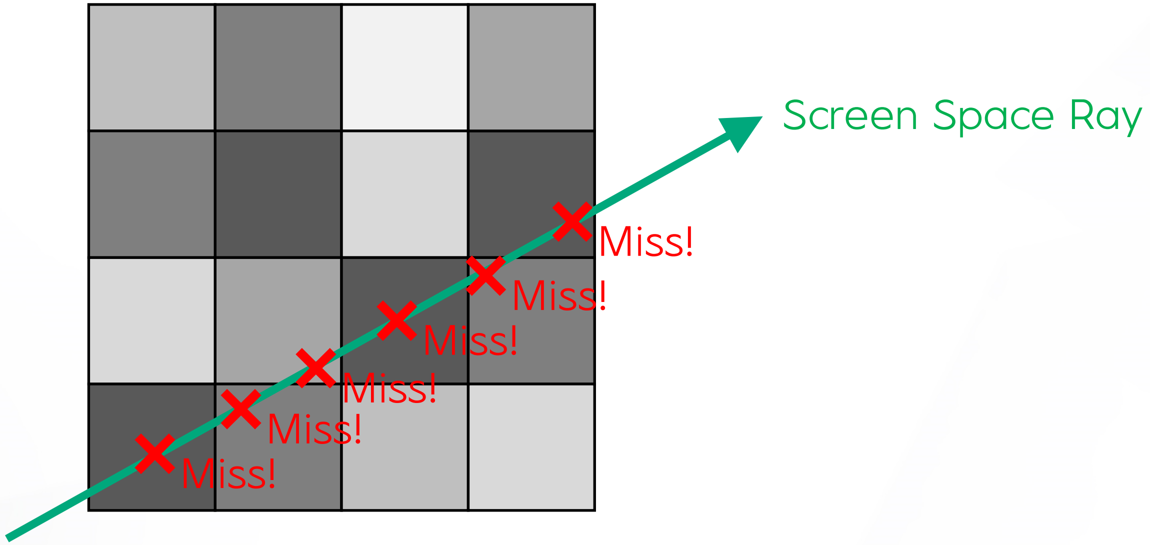
Accelerate the screen space global illumination

- Mipmap acceleration: has “Canyon-Effect” (rays hit in higher-level mipmap but missed in lower-level mipmap, because mipmap uses the nearest depth pooling) – can cause a lot of additional sampling



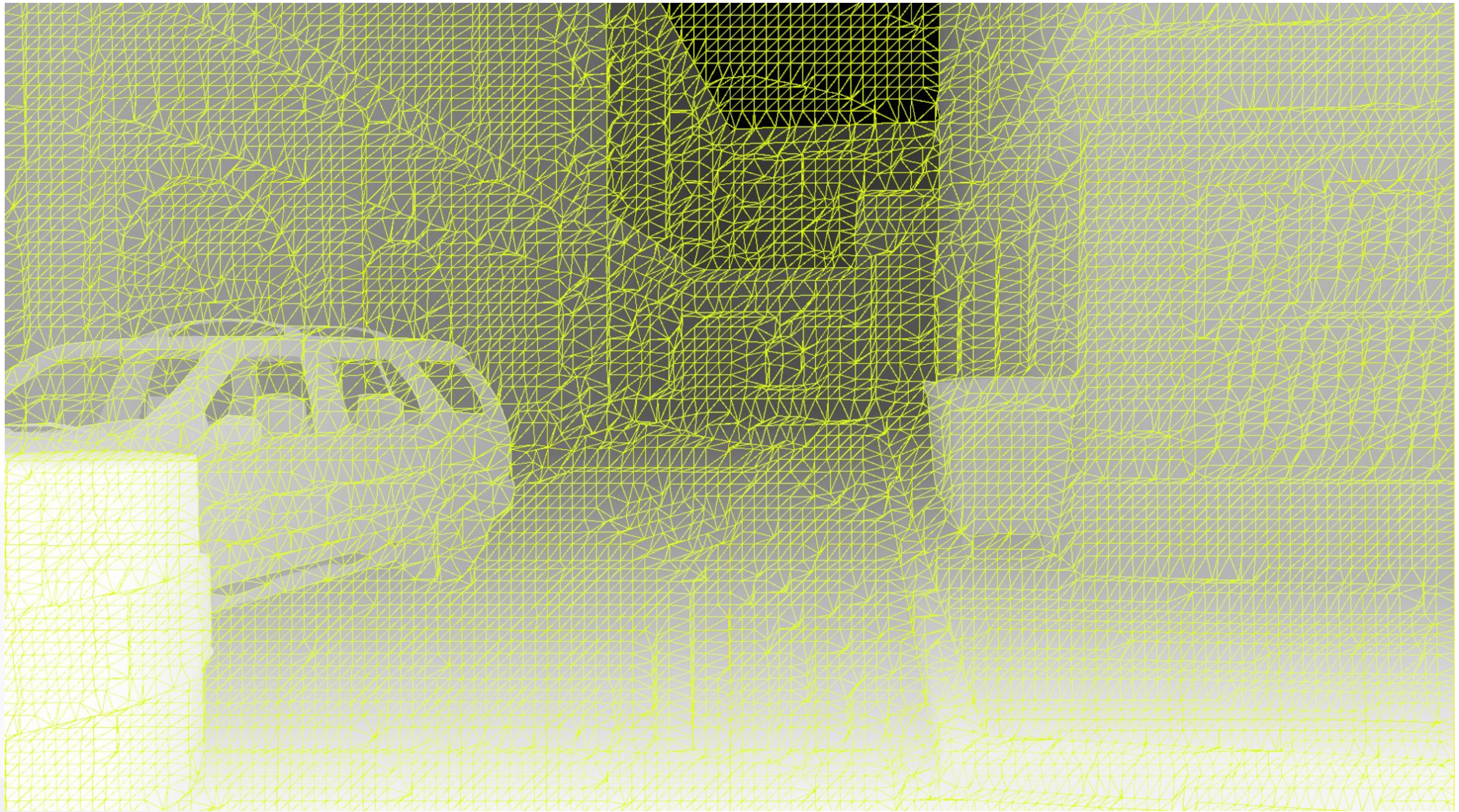
Accelerate the screen space global illumination

- Mipmap acceleration: has "Canyon-Effect" (rays hit in higher-level mipmap but missed in lower-level mipmap, because mipmap uses the nearest depth pooling) – can cause a lot of additional sampling



Accelerate the screen space global illumination

- Mipmap acceleration: has "Canyon-Effect" (rays hit in higher-level mipmap but missed in lower-level mipmap, because mipmap uses the nearest depth pooling) – can cause a lot of additional sampling
- Screen space aggregated mesh (SSAM) acceleration: ray-pixel intersection → ray-triangle intersection



Reconstructed triangles of SSAM

Thank You

Tencent Games GDC