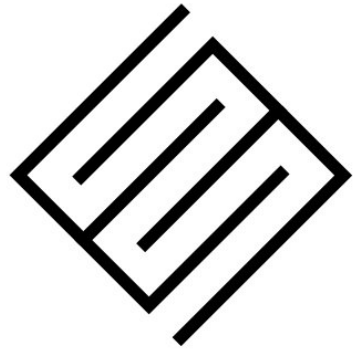


TEXTURE LEVEL OF DETAIL STRATEGIES FOR REAL-TIME RAY TRACING

Colin Barré-Brisebois (SEED, Electronic Arts)
@ZigguratVertigo
cbb@ea.com
seed.ea.com

BEFORE WE BEGIN

A nice NVIDIA & SEED collaboration

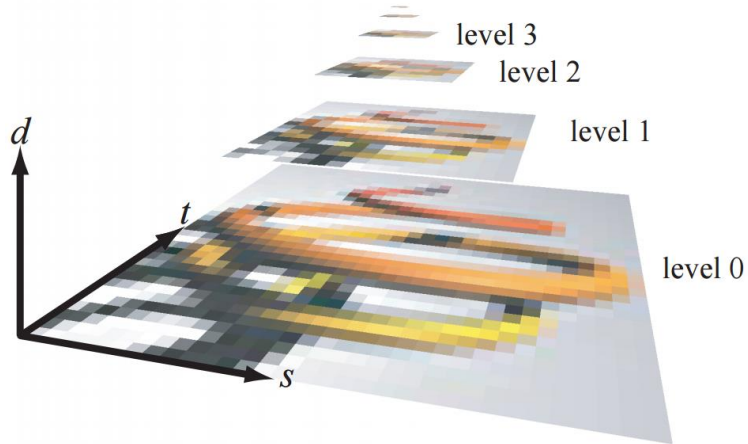
NVIDIA Co-Authors

Tomas Akenine-Möller
Jim Nilsson
Magnus Andersson
Robert Toth
Tero Karras

- Collaboration started during inception of *Microsoft's DirectX Raytracing* & *NVIDIA RTX*
- Aaron Lefohn: "Hey Colin, Tomas and his team are also looking at this problem. Maybe you guys should team-up!" *Thanks Aaron* ☺

MOTIVATION

Texturing is vital to most graphics applications



Want to avoid aliasing when sampling textures

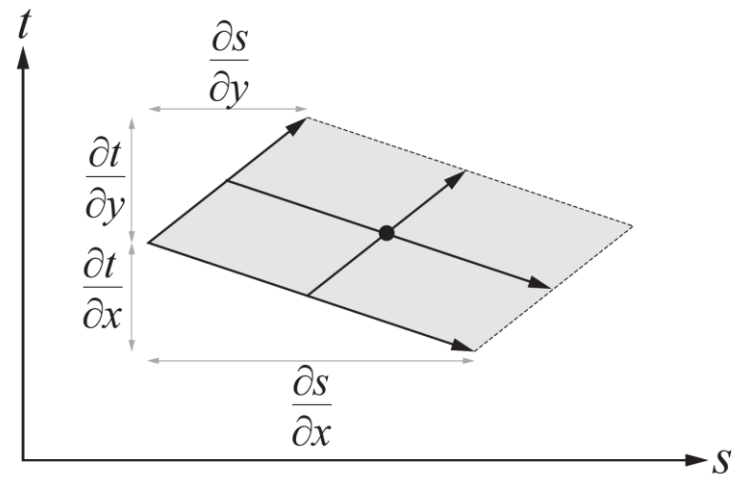
Well-known mechanism: **mip mapping** [Williams83]

Requires footprint to select which mip level(s) λ to sample

For rasterization → 2x2 pixel quad

Differentials are simply/generally the horizontal and vertical differences within a quad

Footprint size is estimated from the differentials



$$\lambda(x,y) = \log_2\lceil \rho(x,y) \rceil \qquad \rho(x,y) = \max\left\{ \sqrt{\left(\frac{\partial s}{\partial x}\right)^2 + \left(\frac{\partial t}{\partial x}\right)^2}, \sqrt{\left(\frac{\partial s}{\partial y}\right)^2 + \left(\frac{\partial t}{\partial y}\right)^2} \right\}$$

WHAT ABOUT RAY TRACING?

Rasterization can handle screen-space texturing

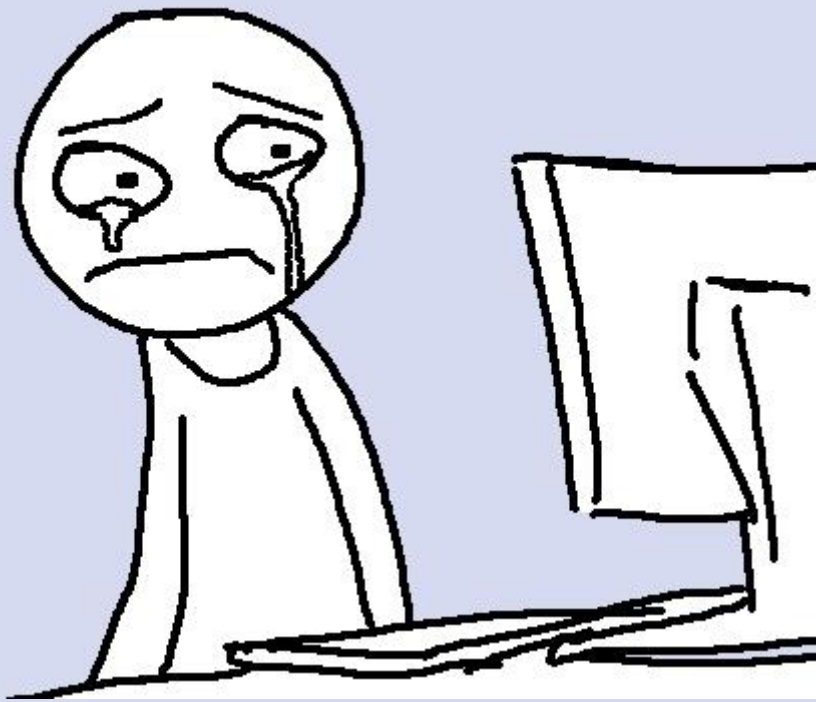
But not for ray-traced reflections and refractions...

No concept of *pixel quad*

Could sample all textures bilinearly at mip 0 and use *temporal anti-aliasing*...

X Visual implications

X Performance implications



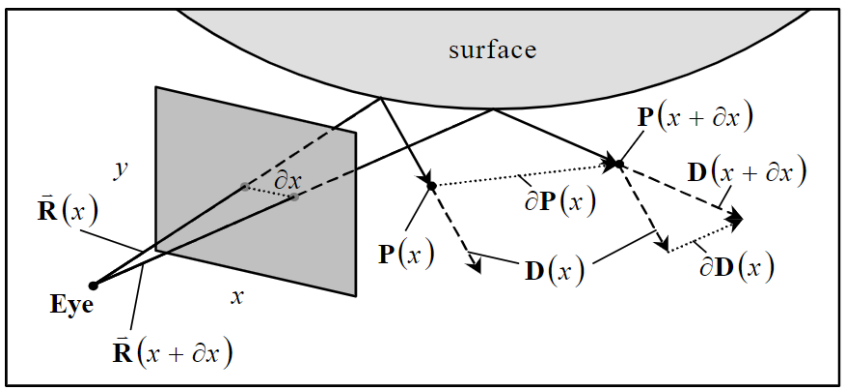
TEXTURE FILTERING FOR RAY TRACING

Not as straightforward

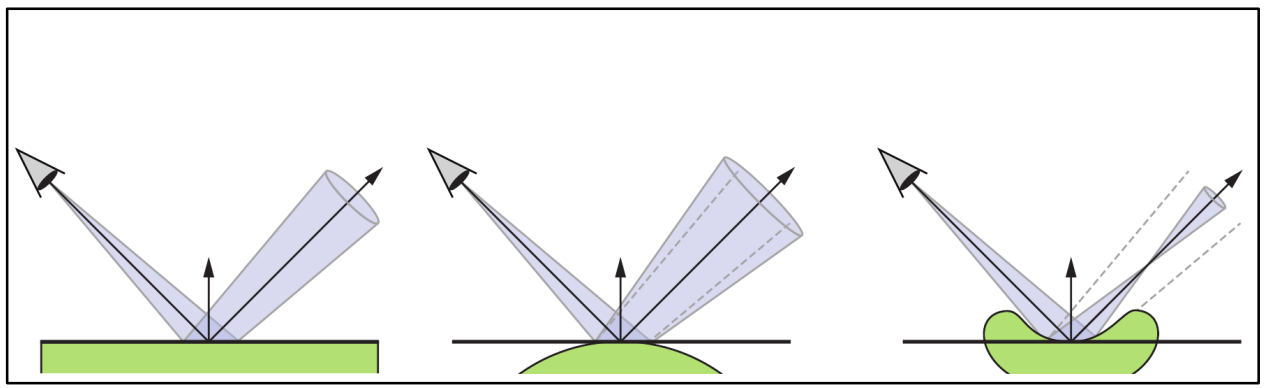
Can shoot primary rays in 2x2 quads in screen space and compute differentials

However, after that, the rays can diverge and the differentials do not make sense

Two methods: *Ray Differentials* [Igehy 1999] and *Ray Cones* [Amanatides 1984]



Ray Differentials [Igehy 1999]



Ray Cones [Amanatides 1984]

RAY DIFFERENTIALS

[Igehy 1999]

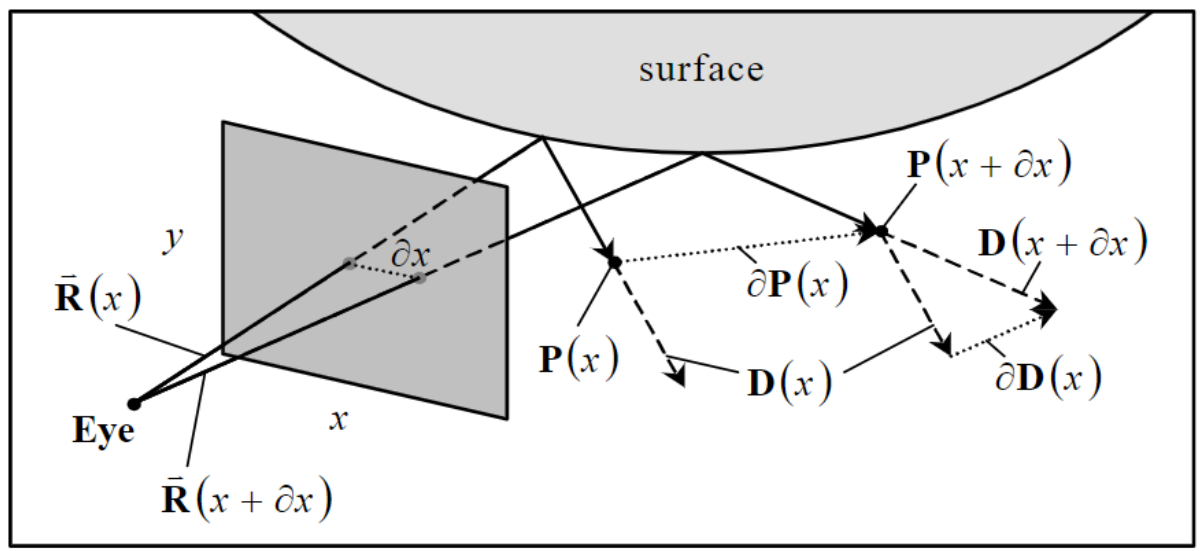
Estimate the footprint of a pixel by computing world-space derivatives of the ray, with respect to the image plane

A ray: $R(t) = O + t\hat{\mathbf{d}}$

A ray differential (RD):

$$\left\{ \frac{\partial O}{\partial x}, \frac{\partial O}{\partial y}, \frac{\partial \hat{\mathbf{d}}}{\partial x}, \frac{\partial \hat{\mathbf{d}}}{\partial y} \right\}$$

Figure from Igehy



In [Akenine-Möller et al. 2019] we present an optimized approach for computing differential barycentric coordinates, and how to use RDs with a G-buffer as a first rendering pass

RTRT: requires 12 floats in the ray payload

RAY DIFFERENTIALS FROM THE G-BUFFER?

Pretty straightforward

Access the G-buffer to the right *(x+1 , y)* and above *(x , y+1)* the current pixel *(x, y)* and create a ray differential from these values.

Normal: $\hat{\mathbf{n}}$

Distance: *t*

Eye ray: $\hat{\mathbf{e}}$

reflect(): **r**

$$\frac{\partial O}{\partial x} = t_{+1:0}\hat{\mathbf{e}}_{+1:0} - t_{0:0}\hat{\mathbf{e}}_{0:0}$$

First hit

$$\frac{\partial \hat{\mathbf{d}}}{\partial x} = \mathbf{r}(\hat{\mathbf{e}}_{+1:0}, \hat{\mathbf{n}}_{+1:0}) - \mathbf{r}(\hat{\mathbf{e}}_{0:0}, \hat{\mathbf{n}}_{0:0})$$

Direction

$$\left\{ \frac{\partial O}{\partial x}, \frac{\partial O}{\partial y}, \frac{\partial \hat{\mathbf{d}}}{\partial x}, \frac{\partial \hat{\mathbf{d}}}{\partial y} \right\}$$

$|t_{+1:0} - t_{0:0}| > \epsilon$? if so, access the G-buffer at -1:0 instead and use the smallest difference in *t*. *Repeat for other axis.*

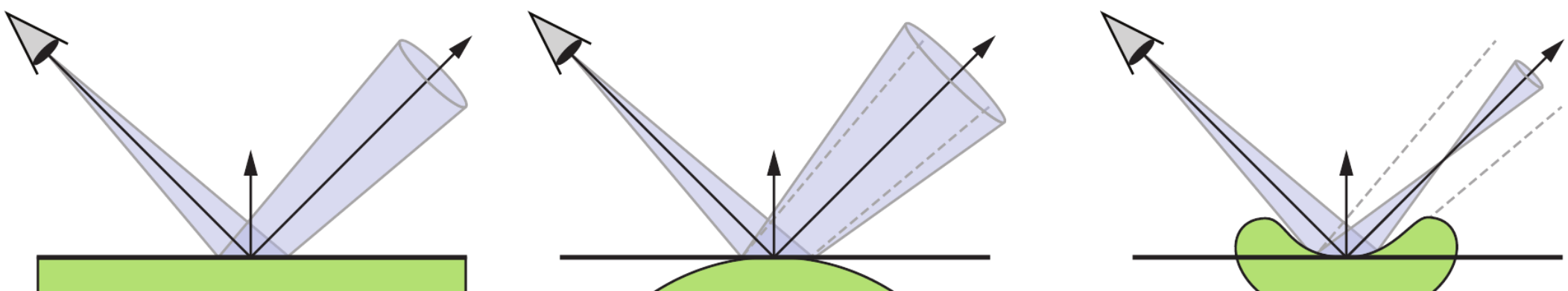
RAY CONES

[Amanatides 1984]

Ray Cones estimate the footprint of a pixel via a cone's distance, angle & spread

Used in offline / film [Christensen et al. 2018], but few implementation details in papers ☹

Our work derives and provides these details, with some approximations tailored for real-time
RTRT: our approach requires a single float in the payload (spread angle)

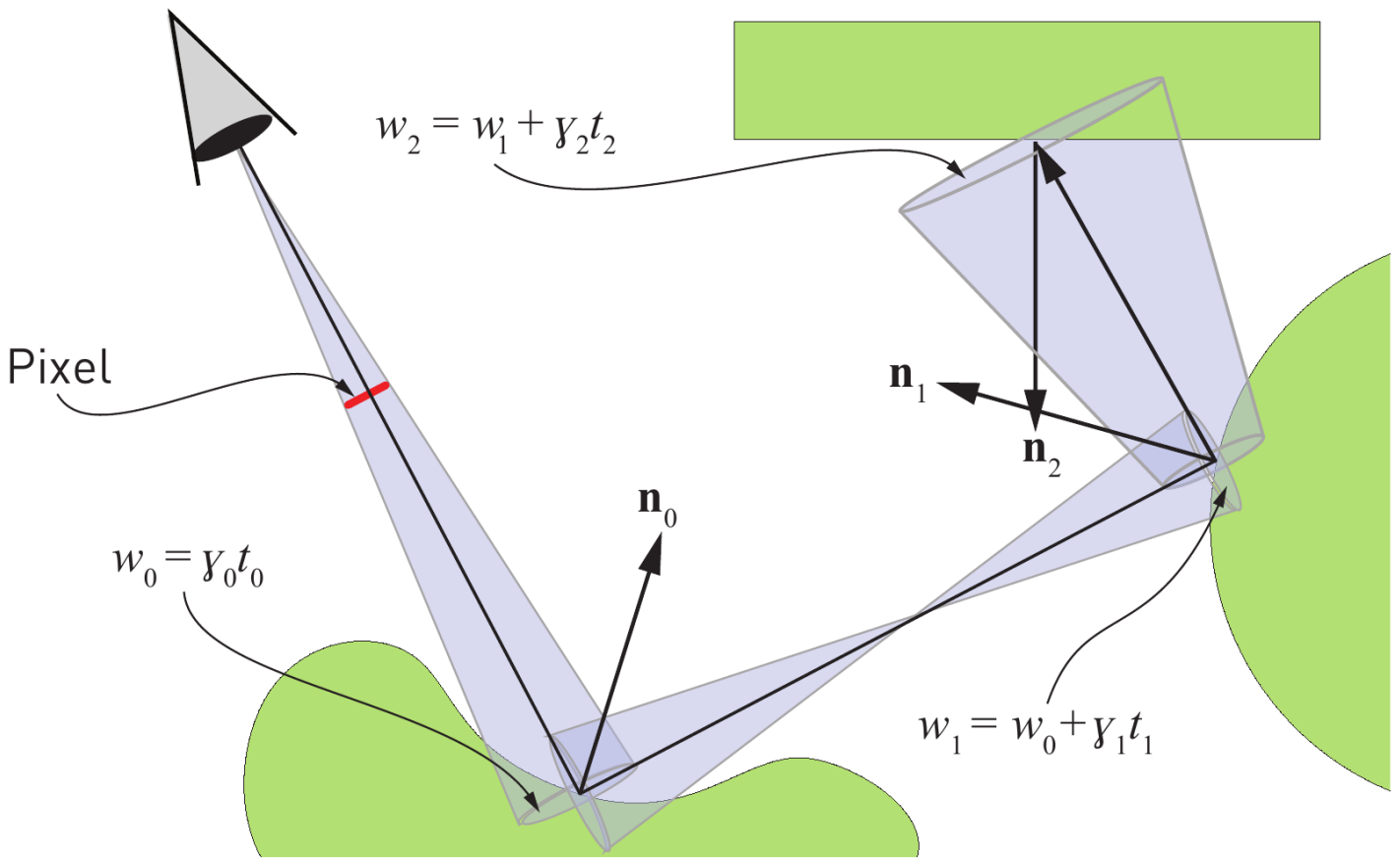


RAY CONE DEFINITION

A ray cone consists of
1) a ray (origin & direction),
2) a spread angle, γ
3) a start width, w

General Idea: track a cone during reflections (and refractions) in the scene

The width w of the cone at a hit is used when computing the mip level

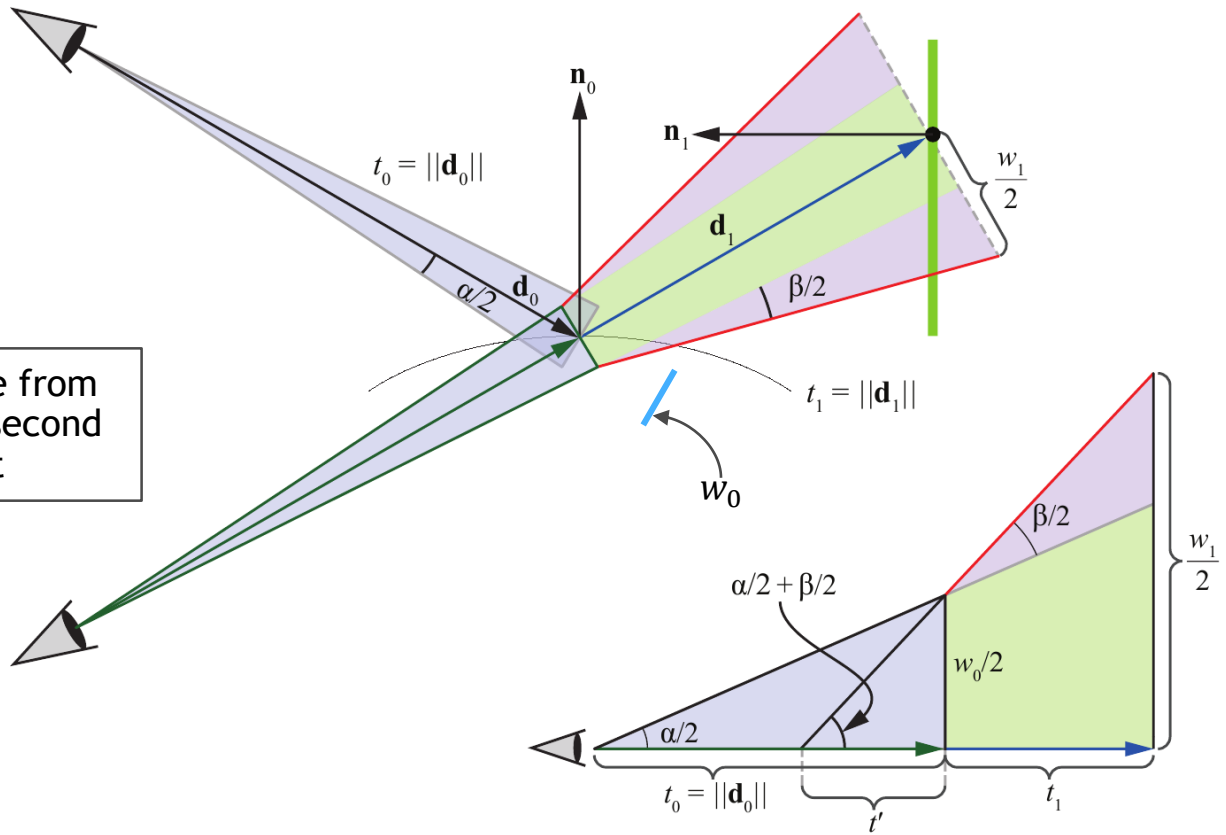


REFLECTION

After some simplification & approximations

$$w_1 \approx w_0 + (\alpha + \beta)t_1$$

- w_1: Width of cone at second hit
- w_0: Width of cone at first hit
- α: Spread angle of cone from the eye
- β: Pixel spread angle at first hit
- t_1: Distance from first to second hit



PIXEL SPREAD ANGLE β

A single float to approximate convex or concave

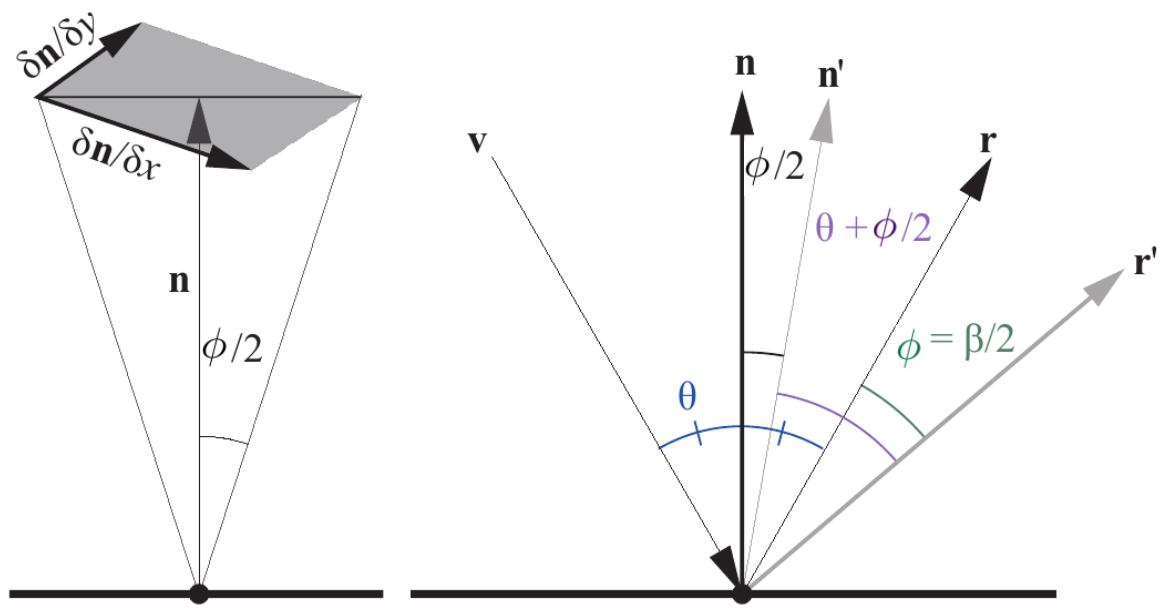
This is not possible in theory: need two values to represent (anisotropic) curvature

Approximation:

$$\phi = 2\arctan\left(\frac{1}{2}\left\|\frac{\partial \mathbf{n}}{\partial x} + \frac{\partial \mathbf{n}}{\partial y}\right\|\right) \approx \left\|\frac{\partial \mathbf{n}}{\partial x} + \frac{\partial \mathbf{n}}{\partial y}\right\|$$

$$\beta = 2s\phi$$

$$s = \texttt{sign}\left(\frac{\partial P}{\partial x} \cdot \frac{\partial \mathbf{n}}{\partial x} + \frac{\partial P}{\partial y} \cdot \frac{\partial \mathbf{n}}{\partial y}\right)$$



We use the normal **n** of the G-buffer
Limitation: can only be computed at first hit

PUTTING IT ALL TOGETHER

$$\lambda_i = \underbrace{\Delta_i}_{\text{triangle LOD}} + \underbrace{\log_2 |w_i|}_{\text{distance}} - \underbrace{\log_2 \left| \hat{\mathbf{n}}_i \cdot \hat{\mathbf{d}}_i \right|}_{\text{normal}}$$

λ_i is the mip level at bounce i

Δ_i triangle LOD (from texture resolution and texture coords), from [Ewins et. al. 1998]

w_i is the width of the cone

$\mathbf{n}_i$ and $\mathbf{d}_i$ are the normals and ray directions

Use λ_i as input to `SampleLevel()` to access correct mip level

TRIANGLE LOD

Single LOD for entire triangle [Ewins et. al. 1998]

$$\Delta = \log_2\left(\sqrt{\frac{t_a}{p_a}}\right) = 0.5\log_2\left(\frac{t_a}{p_a}\right)$$

$$t_a = wh\,\left|(t_{1x}-t_{0x})(t_{2y}-t_{0y}) - (t_{2x}-t_{0x})(t_{1y}-t_{0y})\right|$$

$$p_a = \left|(p_{1x}-p_{0x})(p_{2y}-p_{0y}) - (p_{2x}-p_{0x})(p_{1y}-p_{0y})\right| = ||(P_1-P_0)\times(P_2-P_0)||$$

w = texture width
h = texture height
t = texture coordinates (UV)
p = triangle vertices

triangle index is 0-1-2

From world space positions

PUTTING IT ALL TOGETHER

$$\lambda_i = \underbrace{\Delta_i}_{\text{triangle LOD}} + \underbrace{\log_2 |w_i|}_{\text{distance}} - \underbrace{\log_2 \left| \hat{\mathbf{n}}_i \cdot \hat{\mathbf{d}}_i \right|}_{\text{normal}}$$

λ_i is the mip level at bounce i

Δ_i triangle LOD (from texture resolution and texture coords), from [Ewins et. al. 1998]

w_i is the width of the cone

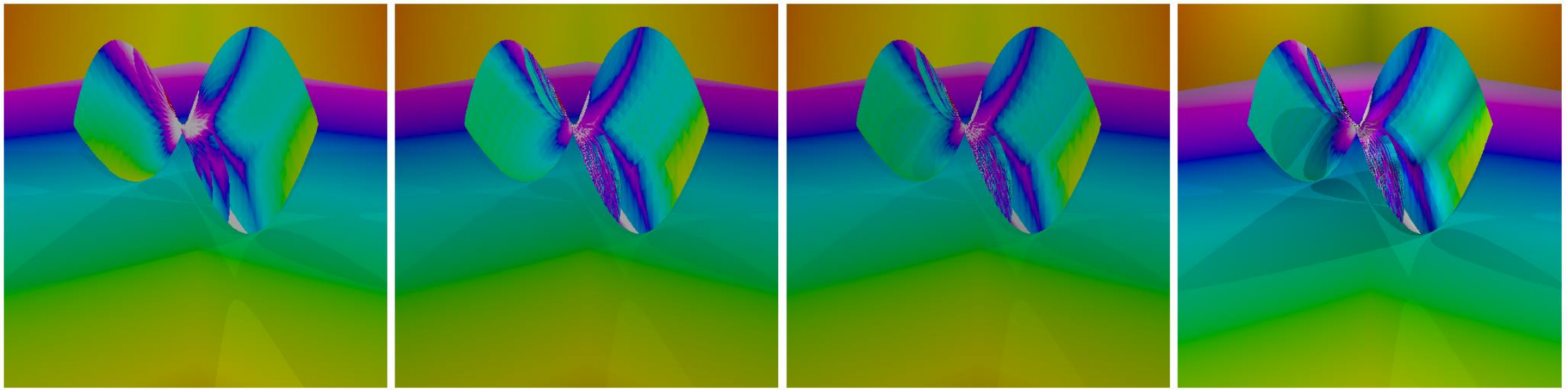
$\mathbf{n}_i$ and $\mathbf{d}_i$ are the normals and ray directions

Use λ_i as input to `SampleLevel()` to access correct mip level

RESULTS

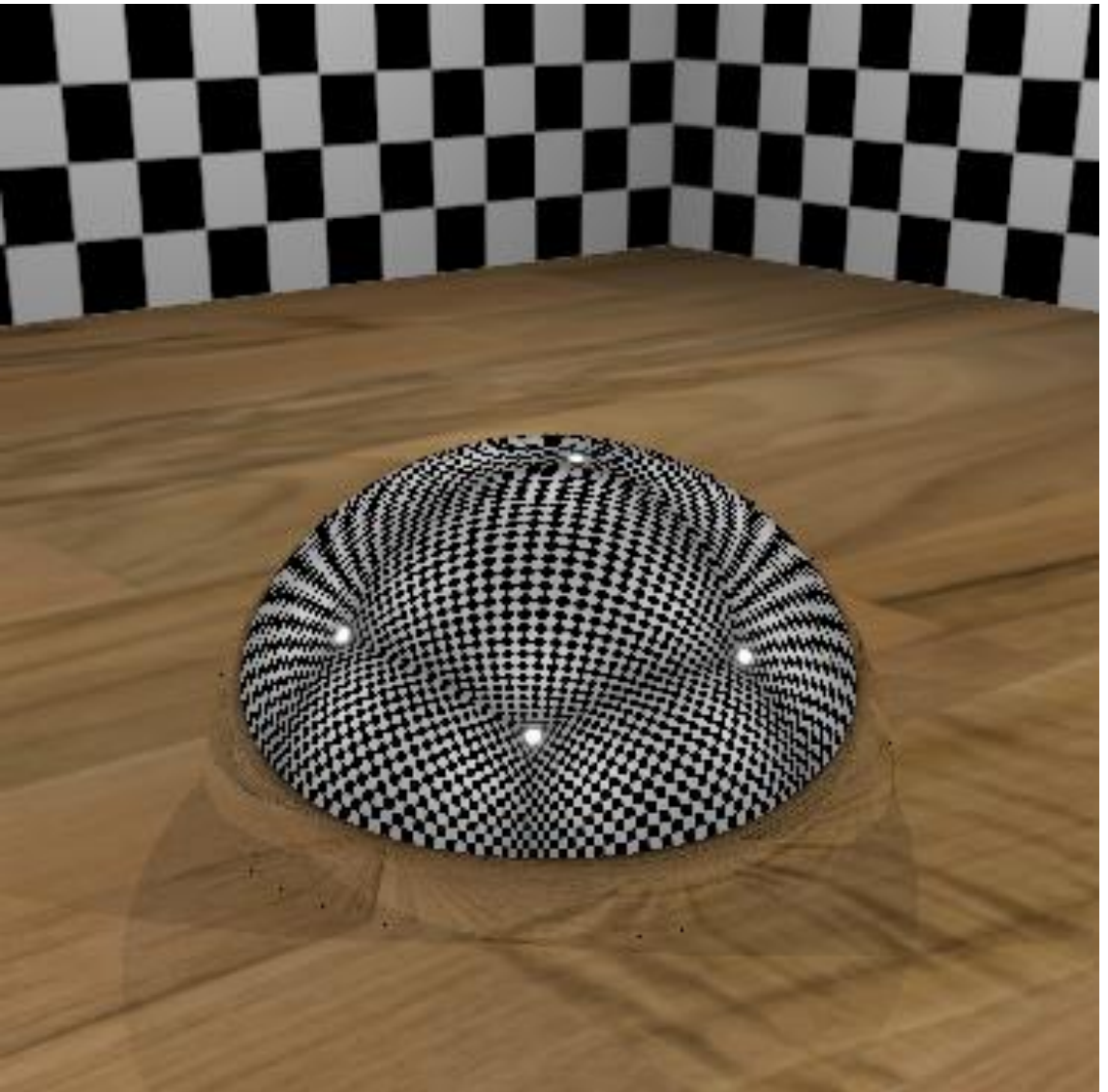
VALIDATE AGAINST GROUND TRUTH

Visualizing mip map selection



RayCones | RayDiffs GB | RayDiffs RT | RayDiffs PBRT

MIP 0



GROUND TRUTH @ 1024 SPP

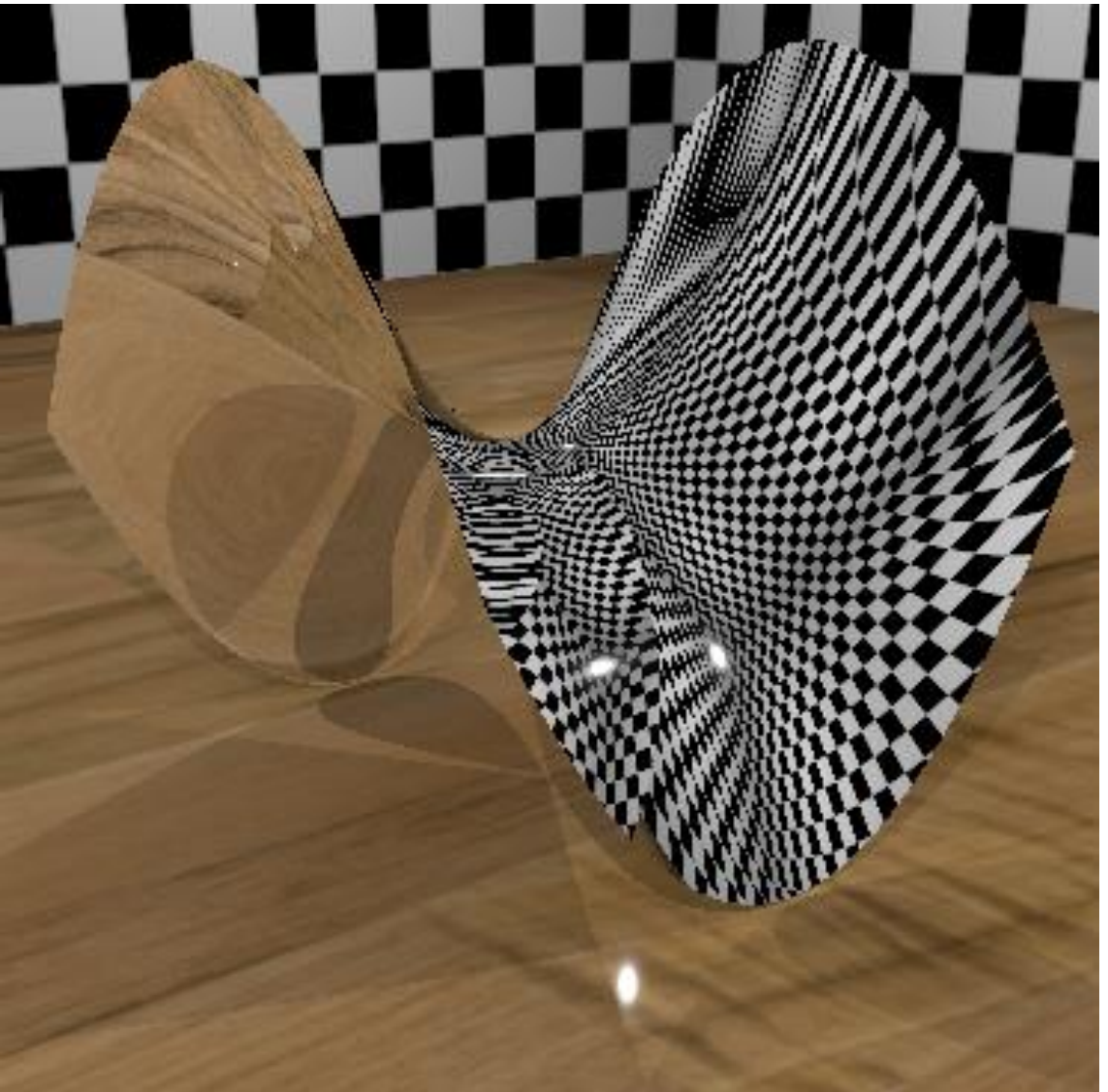
RAY CONES

GROUND TRUTH @ 1024 SPP

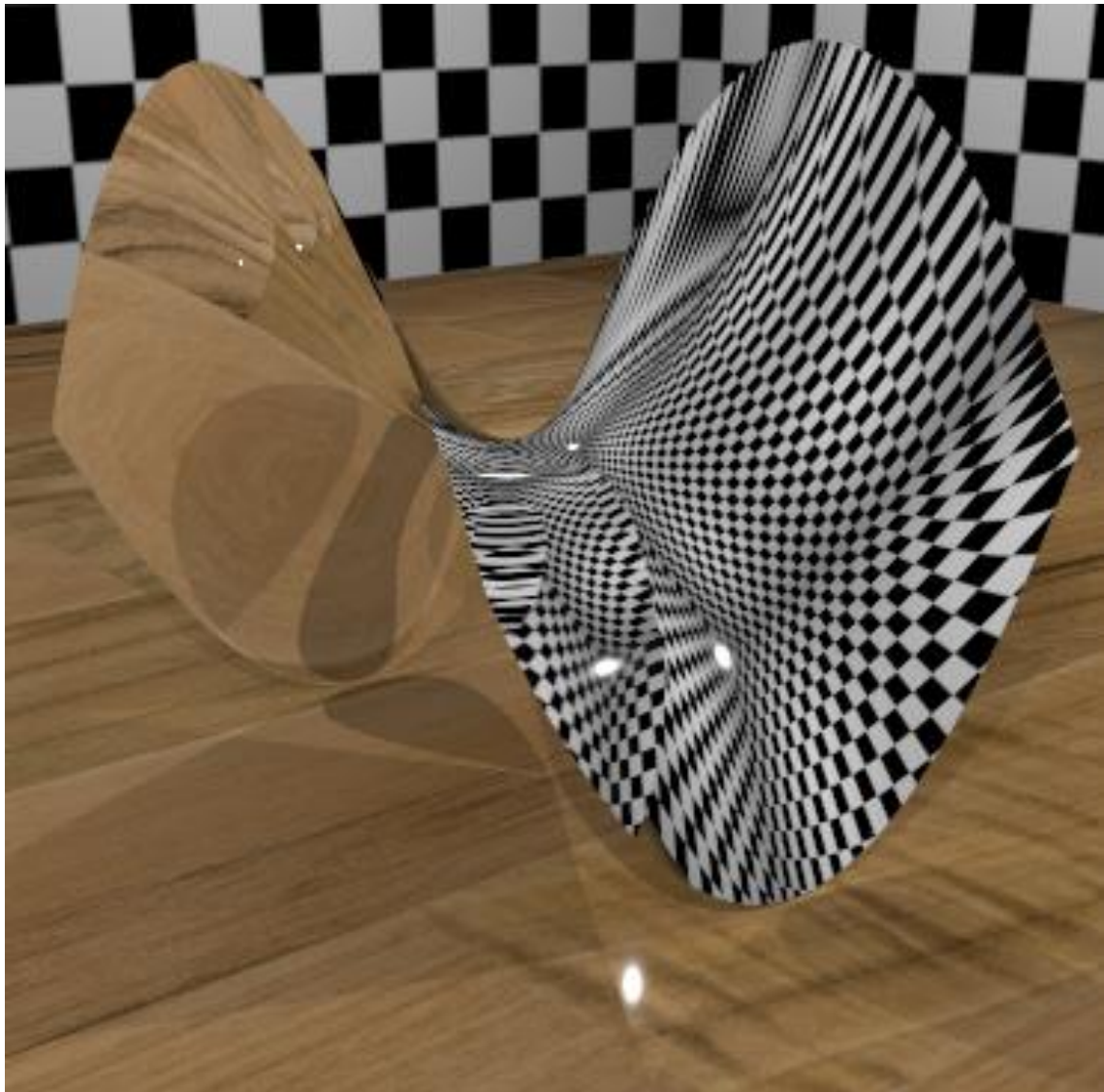
RAY DIFFERENTIALS

GROUND TRUTH @ 1024 SPP

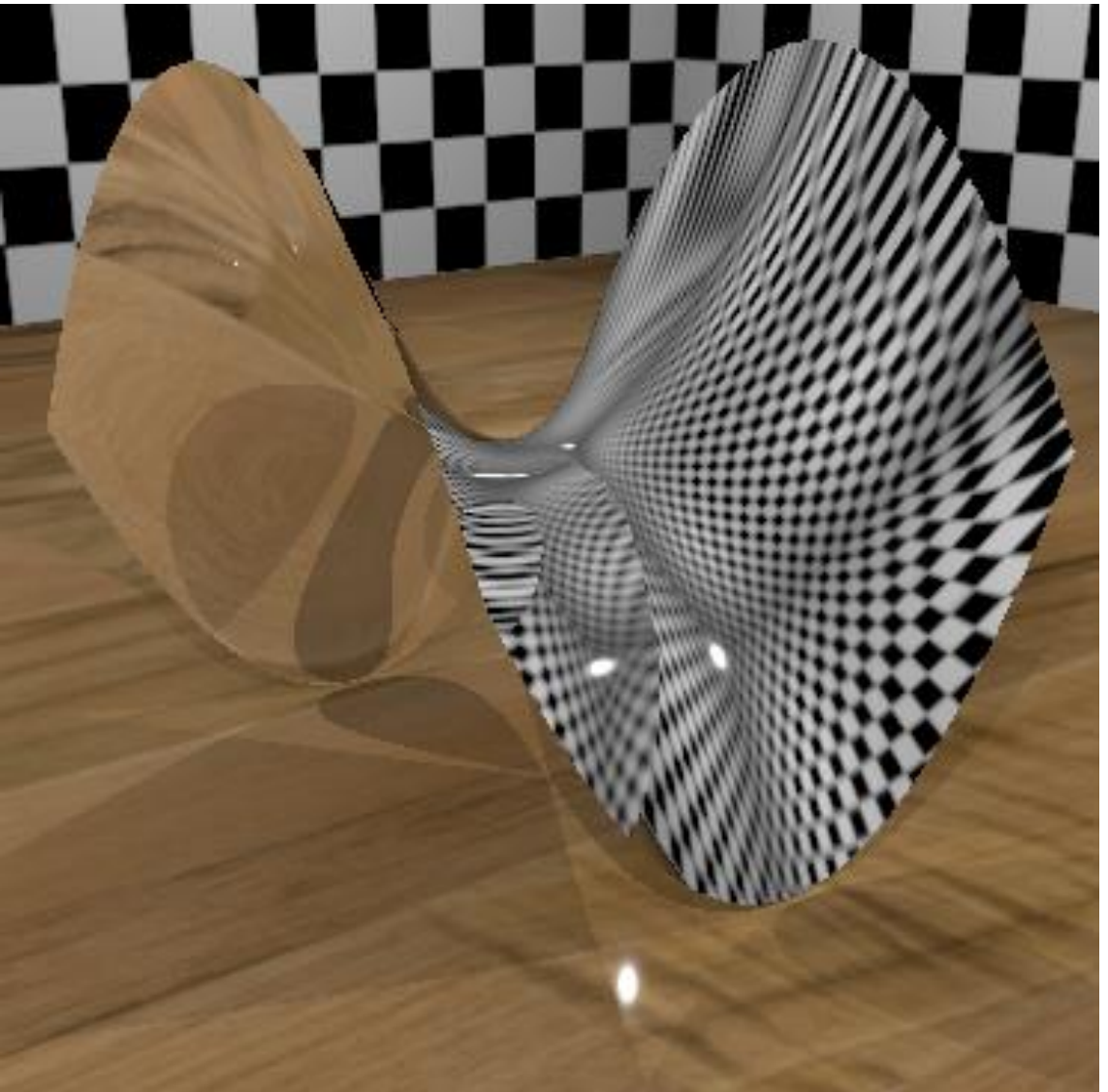
MIP 0



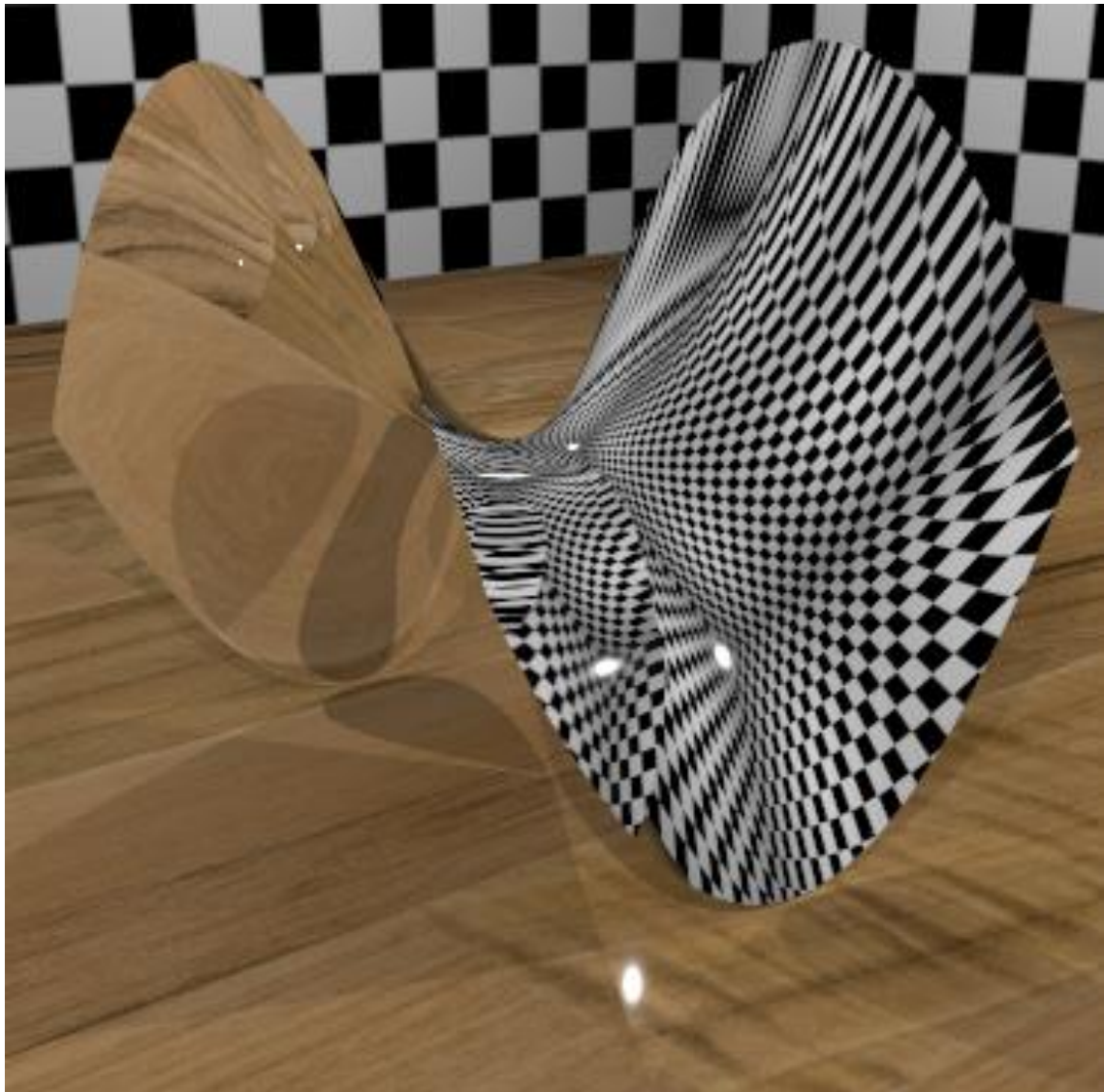
GROUND TRUTH @ 1024 SPP



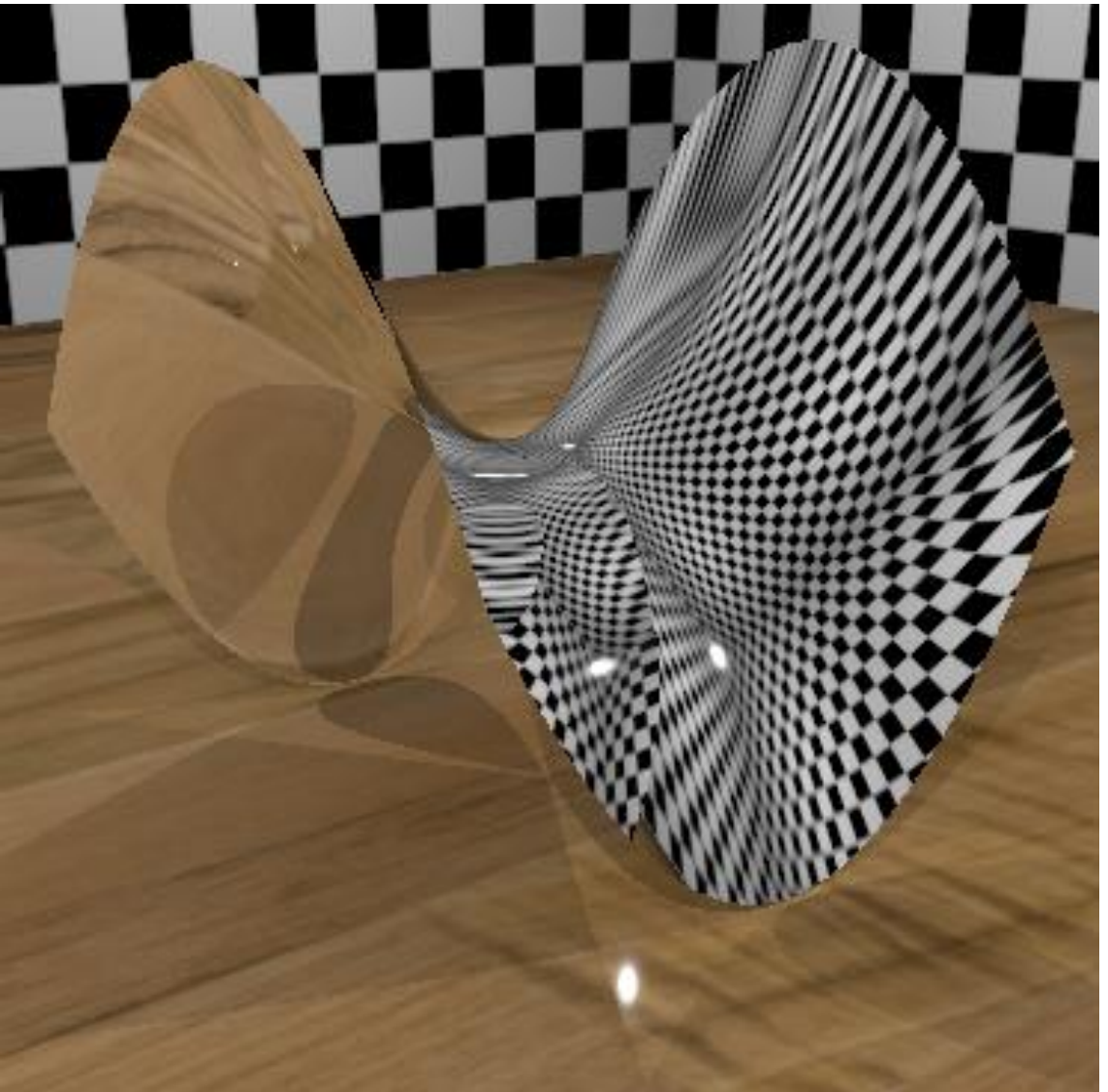
RAY CONES



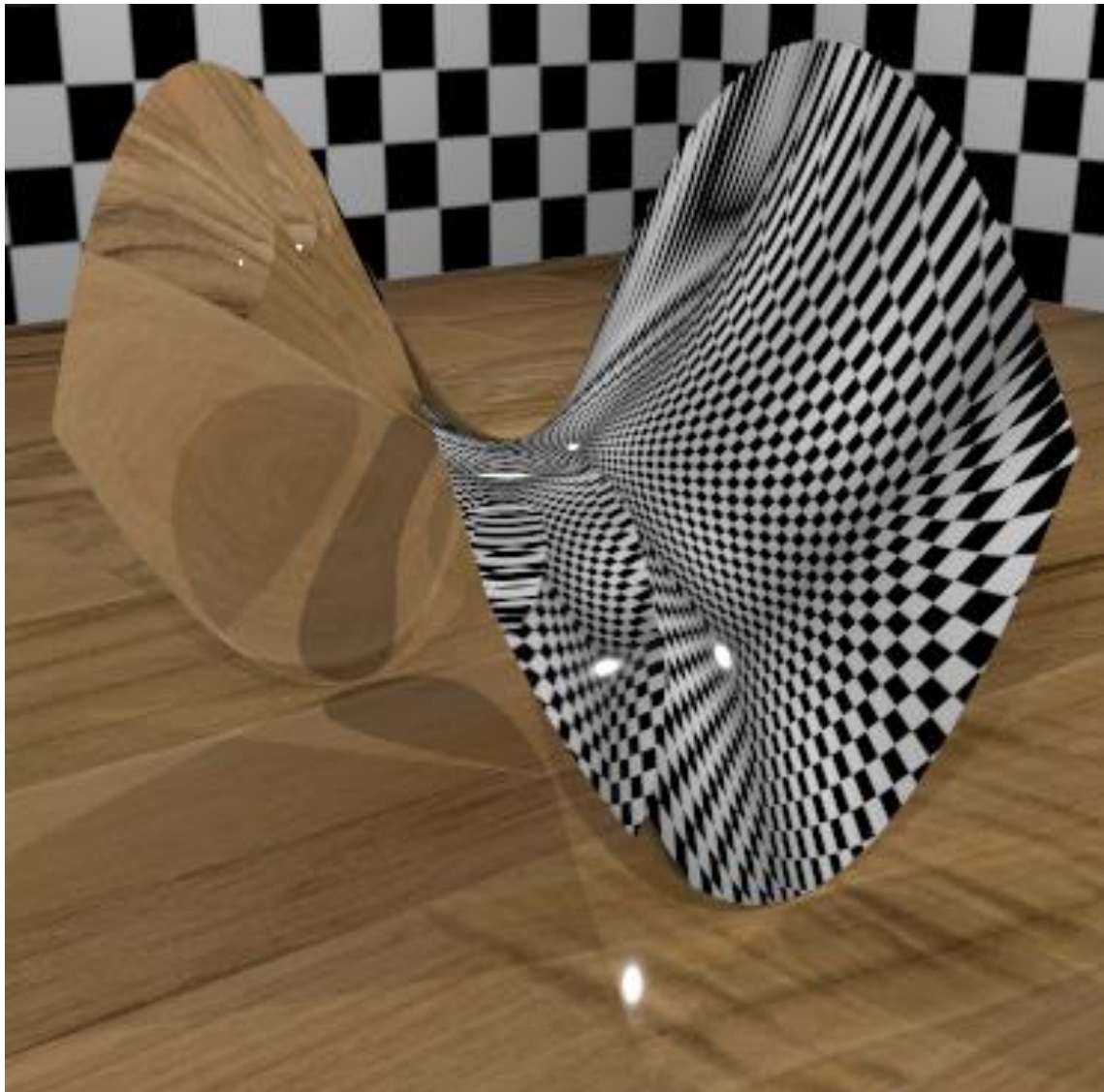
GROUND TRUTH @ 1024 SPP



RAY DIFFERENTIALS



GROUND TRUTH @ 1024 SPP




SEED’S *PICA PICA* USES RAY CONES

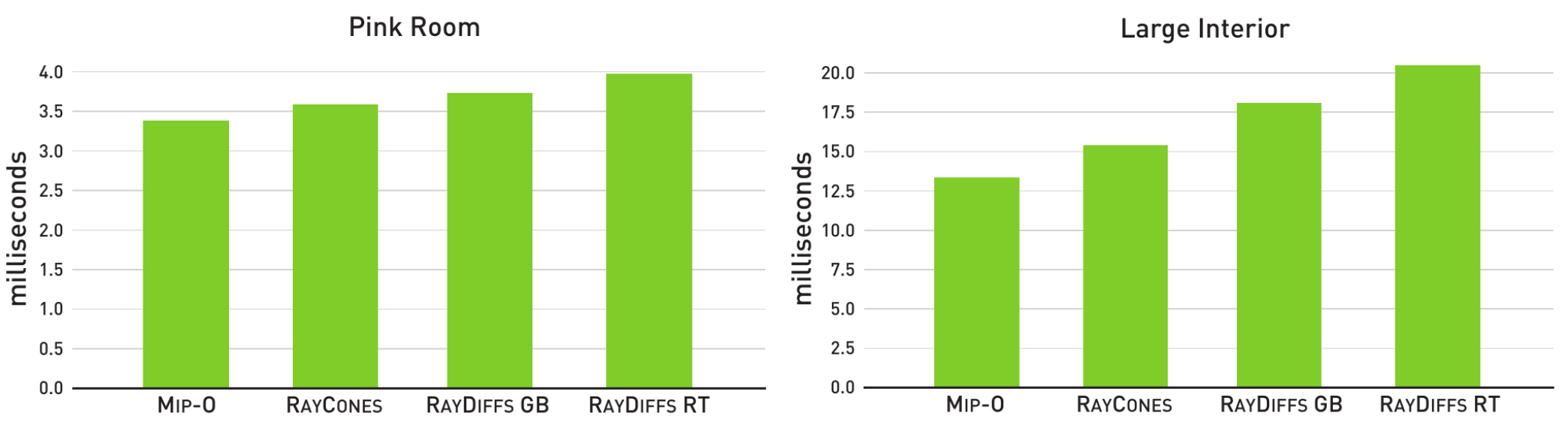
All PBRT renderings
use ray differentials

A rainbow texture is used
to visualize mip levels

All center objects are reflective

PERFORMANCE

3840x2160 resolution on RTX 2080 Ti



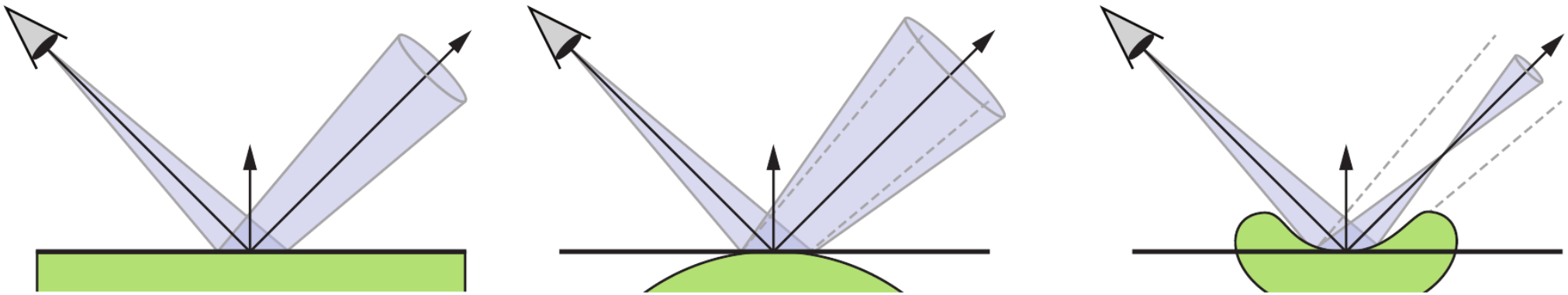
RayDiffs GB (with first pass using G-Buffer) adds about 2x the cost compared to RayCones

RayDiffs RT (with ray traced eye rays) adds about 3x the cost compared RayCones

Low mip0 cost due to very simple materials (single texture)

FUTURE IMPROVEMENTS

A few improvements come to mind



- Ray Cones:
  - Triangle LOD doesn’t handle user-animated UVs. Need to solve for dynamic materials!
  - Highly deforming animated geometry: Could possibly compute triangle LOD in the VS
  - Improve sharpness compared to screen-space mip level selection
- …

CODE

Available in the paper, and soon on github

```
struct RayCone
{
    float width;
    float spreadAngle;
};

struct Ray
{
    float3 origin;
    float3 direction;
};

struct SurfaceHit
{
    float3 position;
    float3 normal;
    float  surfaceSpreadAngle;
    float  distance;
};
```

```
void rayGenerationShader(SurfaceHit gbuffer)
{
    RayCone firstCone = computeRayConeFromGBuffer(gbuffer);
    Ray viewRay = getViewRay(pixel);
    Ray reflectedRay = computeReflectedRay(viewRay, gbuffer);
    TraceRay(closestHitProgram, reflectedRay, firstCone);
}

RayCone propagate(RayCone cone, float surfaceSpreadAngle, float hitT)
{
    RayCone newCone;
    newCone.width = cone.spreadAngle * hitT + cone.width;
    newCone.spreadAngle = cone.spreadAngle + surfaceSpreadAngle;
    return newCone;
}

RayCone computeRayConeFromGBuffer(SurfaceHit gbuffer)
{
    RayCone rc;
    rc.width = 0;                          // No width when ray cone starts
    rc.spreadAngle = pixelSpreadAngle(pixel); // Eq. 30
    // gbuffer.surfaceSpreadAngle holds a value generated by Eq. 32
    return propagate(rc, gbuffer.surfaceSpreadAngle, gbuffer.distance);
}

void closestHitShader(Ray ray, SurfaceHit surf, RayCone cone)
{
   // Propagate cone to second hit
   cone = cone.propagate(0, hitT); // Using 0 since no curvature
                                   // measure at second hit
    float lambda = computeTextureLOD(ray, surf, cone);
    float3 filteredColor = textureLookup(lambda);
    // use filteredColor for shading here
    if (isReflective)
    {
        Ray reflectedRay = computeReflectedRay(ray, surf);
        TraceRay(closestHitProgram, reflectedRay, cone);  // Recursion
    }
}

float computeTextureLOD(Ray ray, SurfaceHit surf, RayCone cone)
{
    // Eq. 34
    float lambda = getTriangleLODConstant();
    lambda += log2(abs(cone.width));
    lambda += 0.5 * log2(texture.width * texture.height);
    lambda -= log2(abs(dot(ray.direction, surf.normal)));
    return lambda;
}

float getTriangleLODConstant()
{
    float P_a = computeTriangleArea();                  // Eq. 5
    float T_a = computeTextureCoordsArea();             // Eq. 4
    return 0.5 * log2(T_a/P_a);                         // Eq. 3
}
```

CONCLUSION

Texture filtering is important for real-time ray tracing, just as it is for rasterization

Ray Cones and *Ray Differentials* are both viable. Code provided in the paper / online ☺

Ray Cones:

→ Faster than *ray differentials*
→ **RTRT**: less register pressure, as our approaches stores a single value in the payload

Ray Differentials:

→ More accurate than *ray cones*
→ Better after the first bounce
→ **RTRT:** requires 12 floats in the payload

For real-time considerations:

→ We chose *Ray Cones* for their simplicity, and low memory footprint.
→ For large scenes/games, filtering improves performance with T$ hits [Pharr 2018]

Additionally

→ See *Simple Environment Map Filtering Using Ray Cones and Ray Differentials* in Ray Tracing Gems.

THANKS

NVIDIA Co-Authors
Tomas Akenine-Möller (main)
Jim Nilsson
Magnus Andersson
Robert Toth
Tero Karras

NVIDIA
Aaron Lefohn
Eric Haines
Alex Hyder

Special Thanks
Tomasz Stachowiak (brainstorming @ SEED)

REFERENCES

- Tomas Akenine-Möller, Jim Nilsson, Magnus Andersson, Colin Barré-Brisebois, Robert Toth, and Tero Karras, “Texture Level of Detail Strategies for Real-Time Ray Tracing,” chapter 20 in *Ray Tracing Gems*, edited by Eric Haines and Tomas Akenine-Möller, 2019.
- Tomas Akenine-Möller and Jim Nilsson, “Simple Environment Map Filtering Using Ray Cones and Ray Differentials,” chapter 21 in *Ray Tracing Gems*, edited by Eric Haines and Tomas Akenine-Möller, 2019.
- Per Christensen et al., “RenderMan: An Advanced Path-Tracing Architecture for Movie Rendering,” *ACM Transactions on Graphics*, vol. 37, no. 3, Article no. 30, August 2018.
- Homan Igehy, “Tracing Ray Differentials,” *Proceedings of SIGGRAPH*, pp. 179—186, 1999.
- John Amanatides, “Ray Tracing with Cones,” Computer Graphics (SIGGRAPH), vol. 18, no. 3, pp 129—135, 1984.
- Matt Pharr, “Swallowing the elephant (part 5), blog post, July 16, 2018.
- Lance Williams, “Pyramidal Parametrics”, Computer Graphics (SIGGRAPH) vol. 17, no 3, 1-11 1983
- Ewins, J. P., Waller, M. D., White, M., and Lister, P. F. MIP-Map, “Level Selection for Texture Mapping”, IEEE Transactions on Visualization and Computer Graphics 4, 4 (1998), 317-329