Hardware-Accelerated Stochastic Rasterization on Conventional GPU Architectures

Morgan McGuire **Eric Enderton** **Peter Shirley** **David Luebke**

NVIDIA Research
Williams College

NVIDIA Research

NVIDIA

hpg 2010

Real-time stochastic rasterization of micropolygons would require 70 of today's GPUs. [BRUNHAVER ET AL. 10]

Micropolygons are too small for efficient sampling. [BRUNHAVER ET AL. 10, BOULOS ET AL. 10]

What about efficient macropolygon sampling?

Goals

- Defocus blur from non-pinhole lens
- Motion blur from non-zero exposure time
- Interoperate with spatial antialiasing (MSAA)
- Real time on current hardware architectures and APIs

Assumptions

- Vertices move with **constant velocity** during a frame

  [e.g., Akenine-Möller et al. 07, Fatahalian et al. 09, Ragan-Kelly et al. 10]

- Shading does not vary significantly across a pixel or an exposure interval (i.e., separate visibility and shading)

  [e.g., Reeves et al. 87, Cook et al. 87, Fatahalian et al. 09, Ragan-Kelly et al. 10]

- Target "**macro triangles**" with edges longer than:
  - intra-frame motion
  - circle of confusion (defocus) radius

  (works for any *triangles, but efficiency increases with projected area)*

hpg 2010

Conclusions

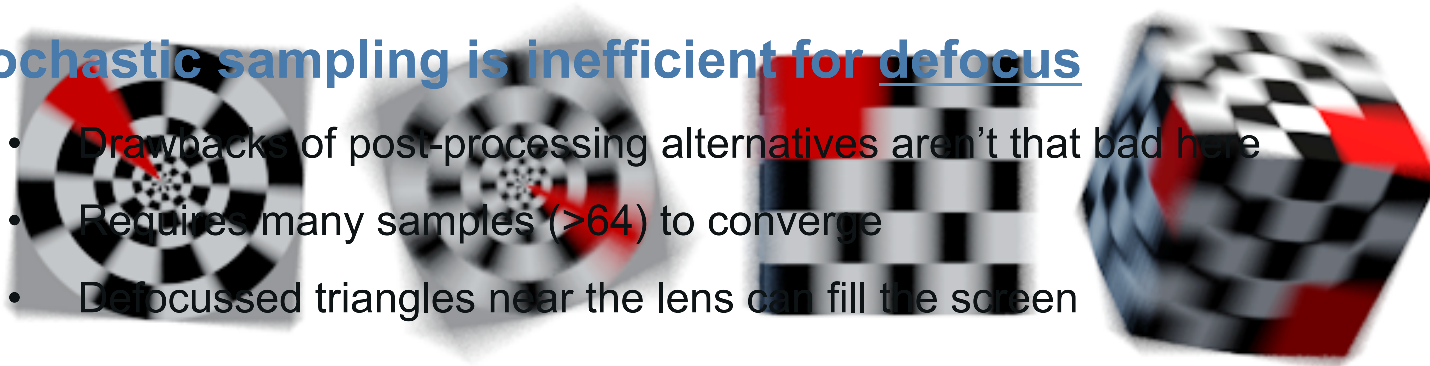
Real-time stochastic rasterization is possible *now*

Motion blur is very different from defocus blur

Stochastic rasterization works well for motion blur

- Substantially better interactions than post-processing methods
- 8x MSAA is probably good enough and today's GPUs are built for it
- Constant velocity approximation yields surprisingly good results

Stochastic sampling is inefficient for defocus



- Drawbacks of post-processing alternatives aren't that bad here
- Requires many samples (>64) to converge
- Defocussed triangles near the lens can fill the screen

State of the Art

Scene-Specific 2D Filtering and Compositing

- Unavoidable artifacts: loss of parallax, dim bokeh, color

Just Cause 2 [Avalance & Eidos 10]

hpg 2010

State of the Art

Scene-Specific 2D Filtering and Compositing

- Unavoidable artifacts: loss of parallax, dim bokeh, color bleeding, black halos
- Extremely fast for controlled scenes and scenarios, e.g., cinematic, car game or small camera rotations

Accumulation Buffering (brute force)

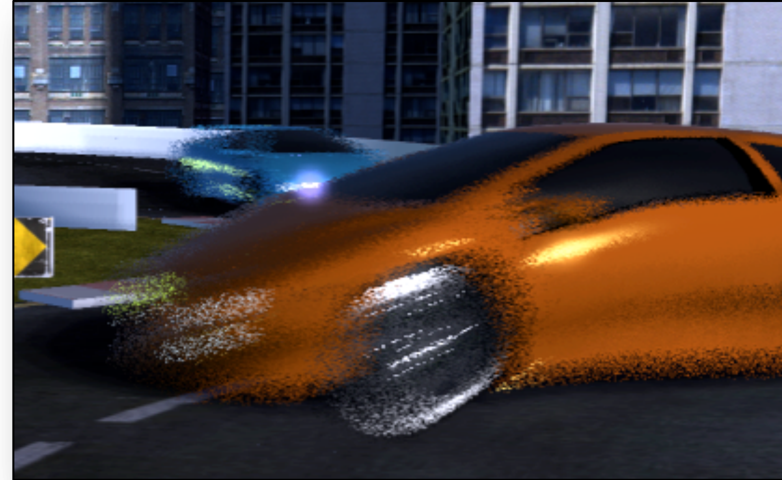
- Physically correct
- Straightforward, but massively overshades

Stochastic Rendering

- Physically correct and reasonable shading cost
- Easy for rays/REYES, tricky to make efficient

conventional(t, u, v): $O(W \bullet H)$ no blur

stochastic(): $O(W \bullet H)$ noisy



[Cook et al. 84, 86, 87]

accumulation(): $O(W \bullet H \bullet N)$ ghosts

5 Sampling Parameters:

xy position on screen

t time

uv lens position

---

t^* intersection time

t_s shade time

t iteration parameter

5D ~~2D~~ Rasterization

Geometry Shader	1. Bound (e.g., with a rectangle) the triangle's screen-space extent due to its shape (*xy*), motion (*t*), and defocus (*uv*)
2D Rasterizer	2. Iterate over the samples in that bound
Pixel Shader	3. Perform some *xytuv* inside-outside test per sample
	4. Shade the samples that pass

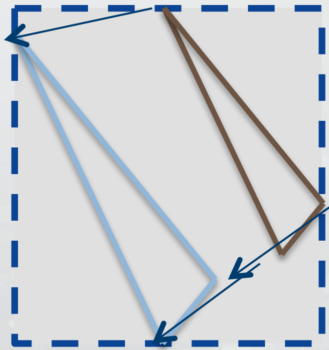
hpg 2010

5D ~~2D~~ Rasterization

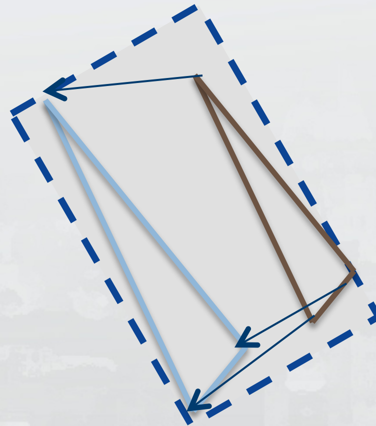
Geometry Shader — 1. Bound (e.g., with a rectangle) the triangle's screen-space extent due to its shape (*xy*), motion (*t*), and defocus (*uv*)

2D Rasterizer — 2. Iterate over the samples in that bound

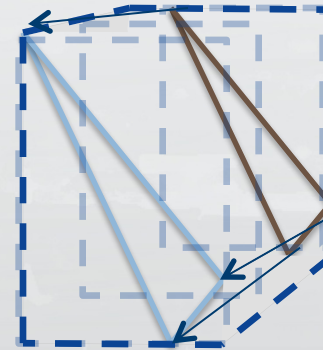
Pixel Shader —
3. Perform some *xytuv* inside-outside test per sample
4. Shade the samples that pass



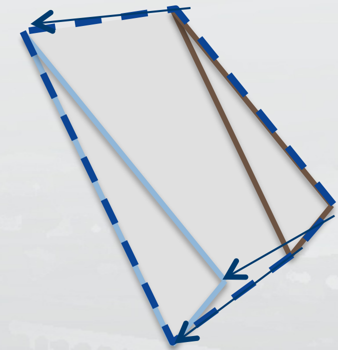
[Wexler et al. 05]



[Akenine-Möller et al. 07]



[Fatahalian al. 09]

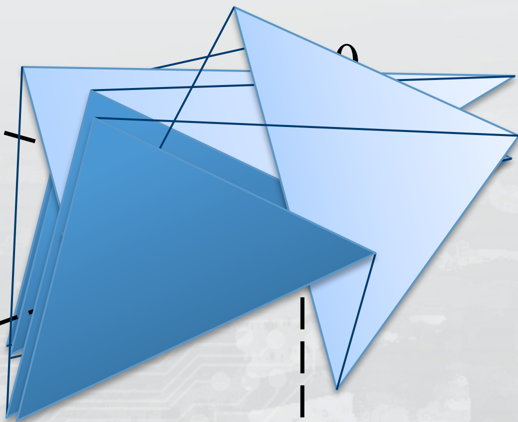


New: 2.5D Convex Hull

hpg 2010

~~2D~~ 5D Rasterization

Geometry Shader	1.	Bound (e.g., with a rectangle) the triangle's screen-space extent due to its shape (xy), motion (t), and defocus (uv)
2D Rasterizer	2.	Iterate over the samples in that bound
Pixel Shader	3.	Perform some $xytuv$ inside-outside test per sample
	4.	Shade the samples that pass



The challenge:

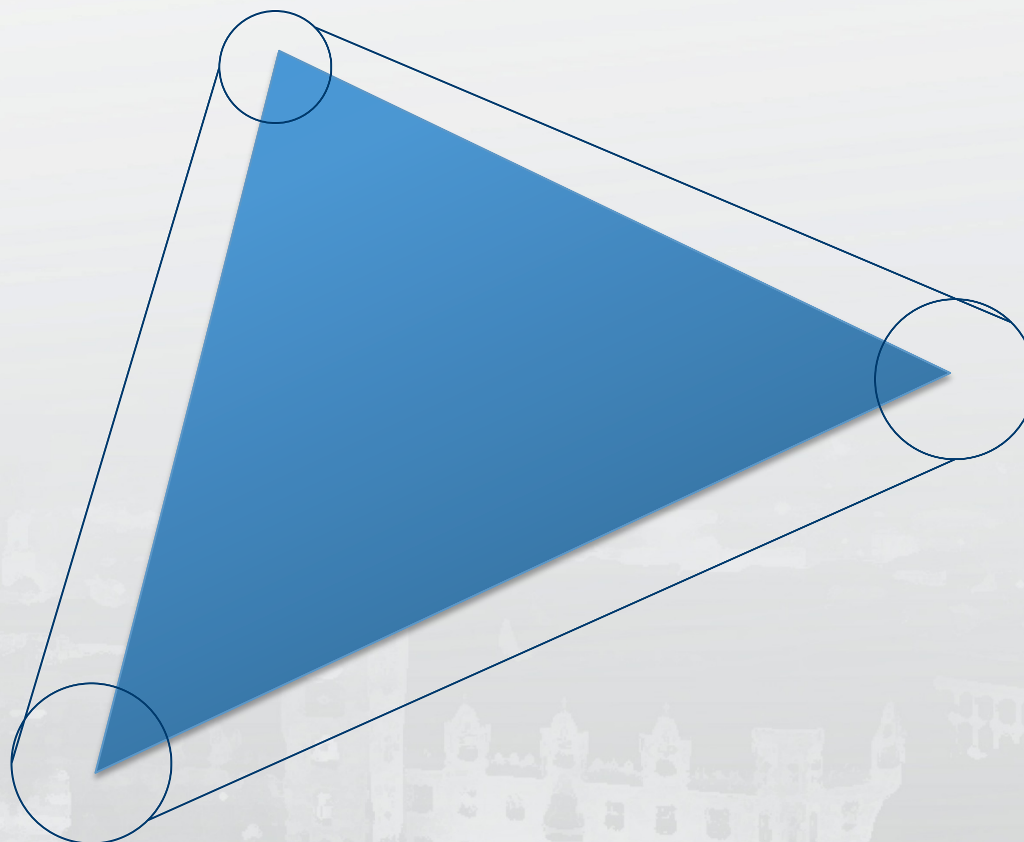
Tight 2D bounds yield high sample-test efficiency…

but are hard to compute because of:

- Geometric Issues
  - Moving triangles carve bilinear patch edges
  - Triangles may cross z = 0 in space or time
- Hardware Constraints
  - Bound must be expressed as a triangle strip
  - Need small, constant space and time algorithm

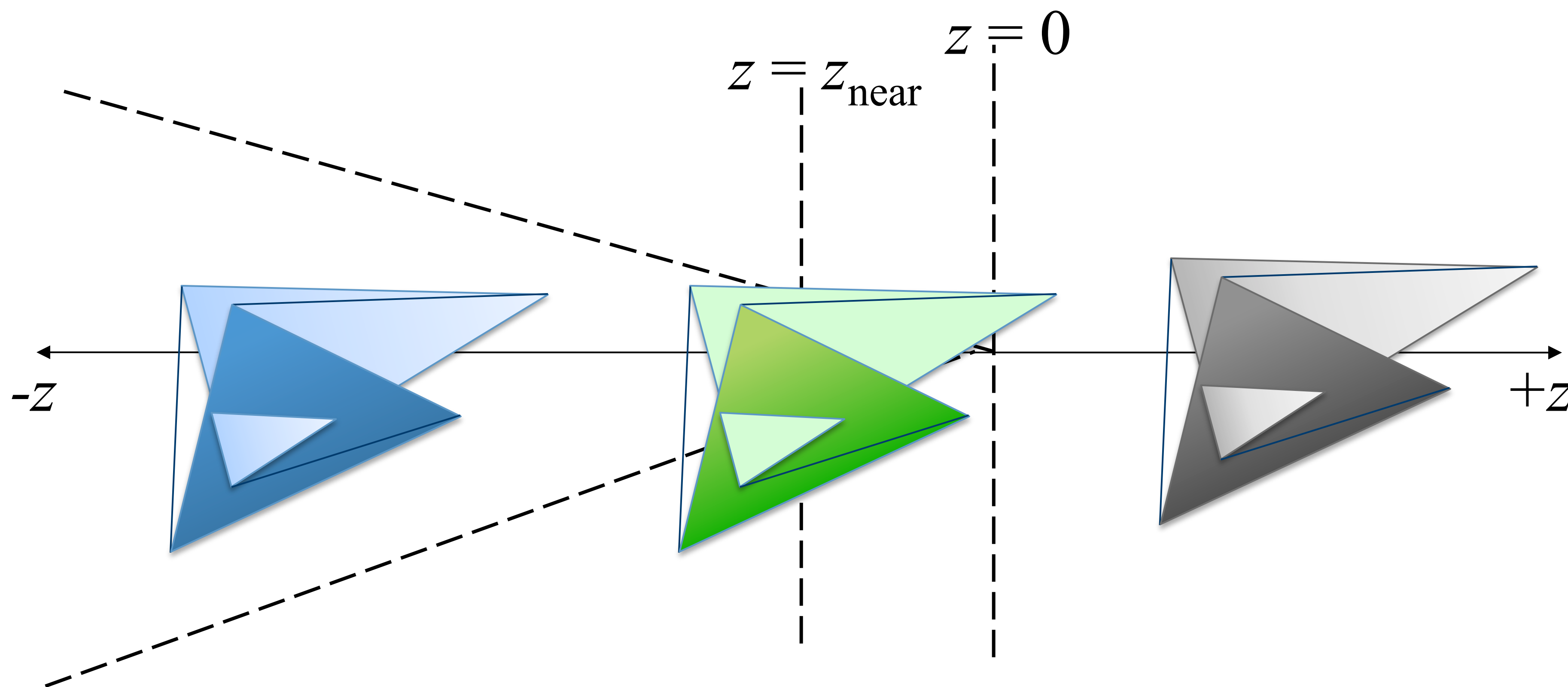
hpg 2010

Bounding Defocus has been Solved



[Toth and Linder 08]

Our Bounding Solution

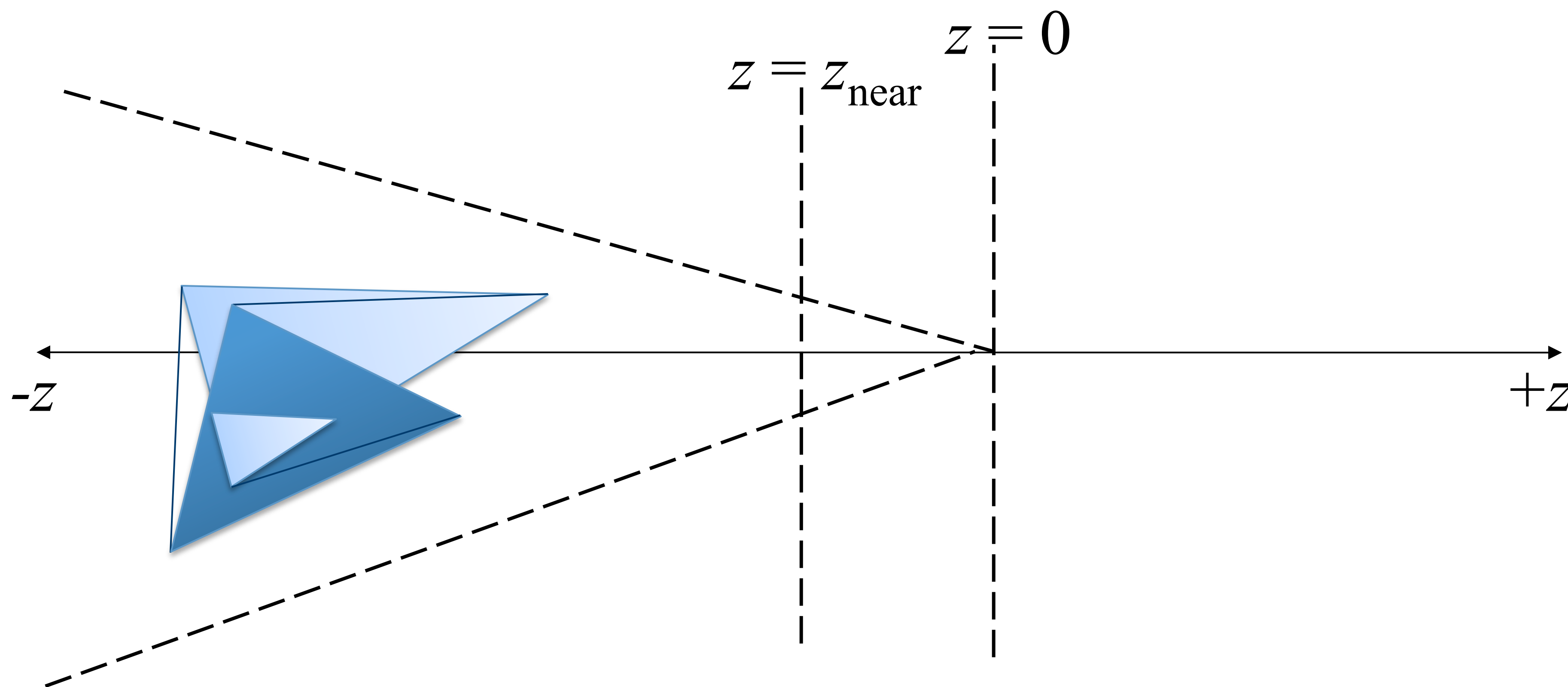


"Normal": All $z < 0$: **Projected Hull**

All $z > z_{near}$: **Cull**

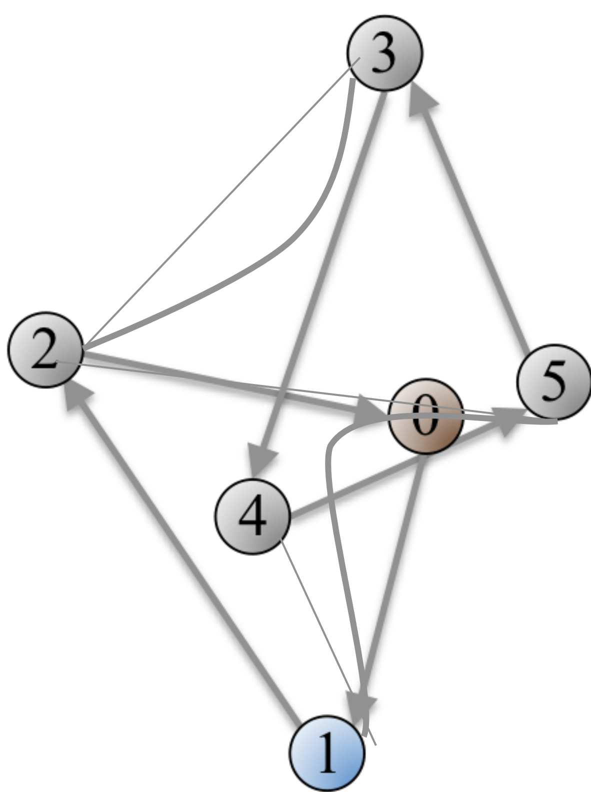
"z=0 crossing": $z_{min} < z_{near}$ and $z_{max} > 0$: **Clip and Box**

Normal Case

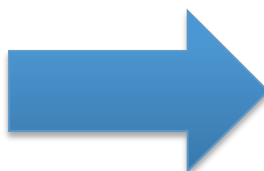


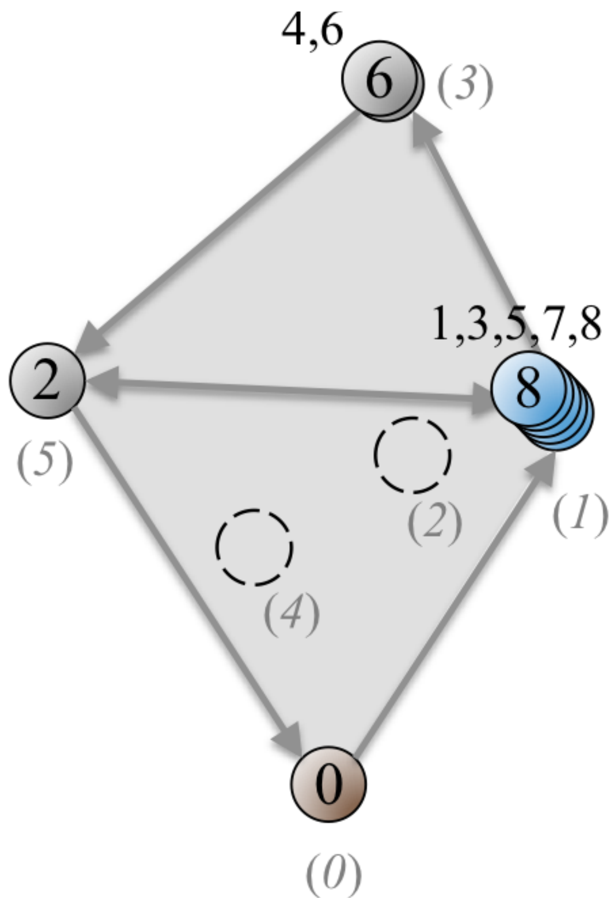
"Normal": All $z < 0$: **Projected Hull**

Normal Case



Sample Input





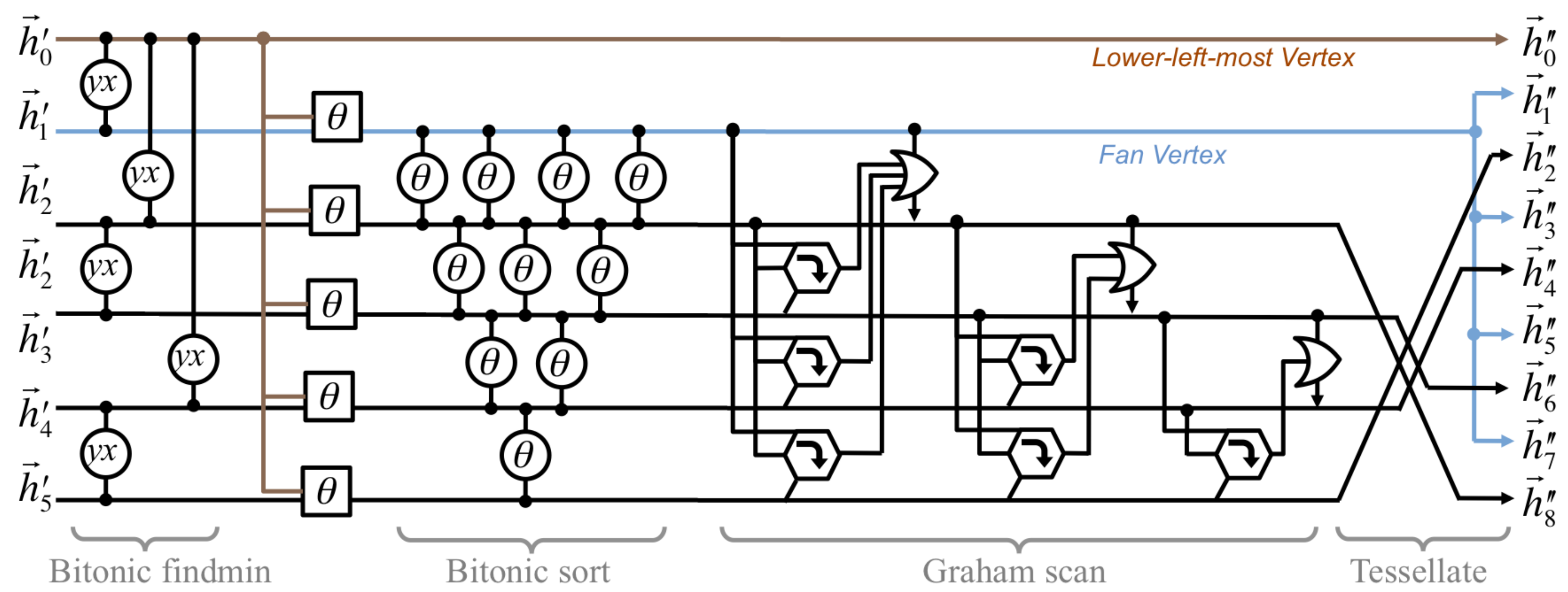
Corresponding Output

(and pre-scan order)

Projected Time-Continuous Triangle

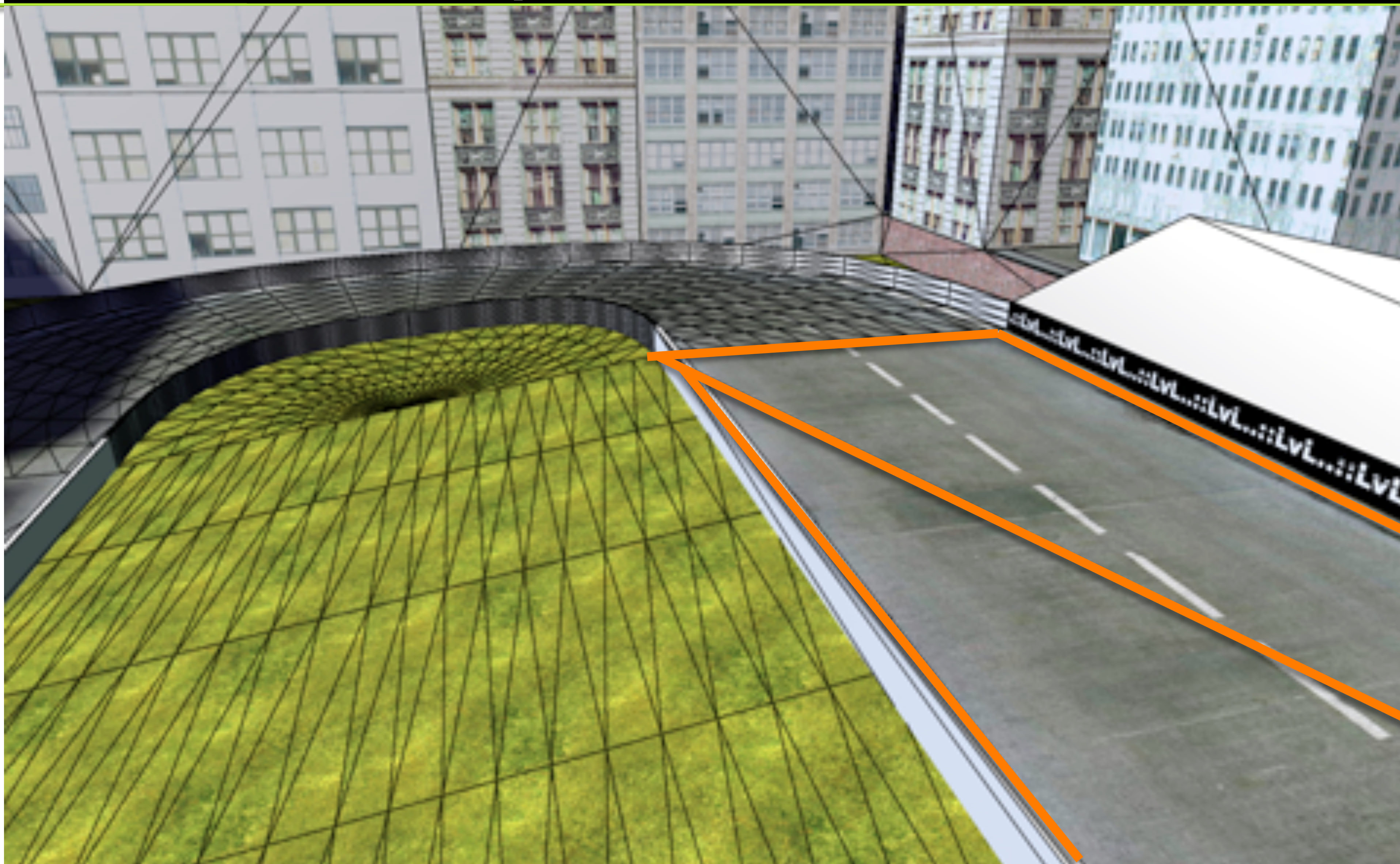
2.5D Convex Hull Tristrip

Normal Case

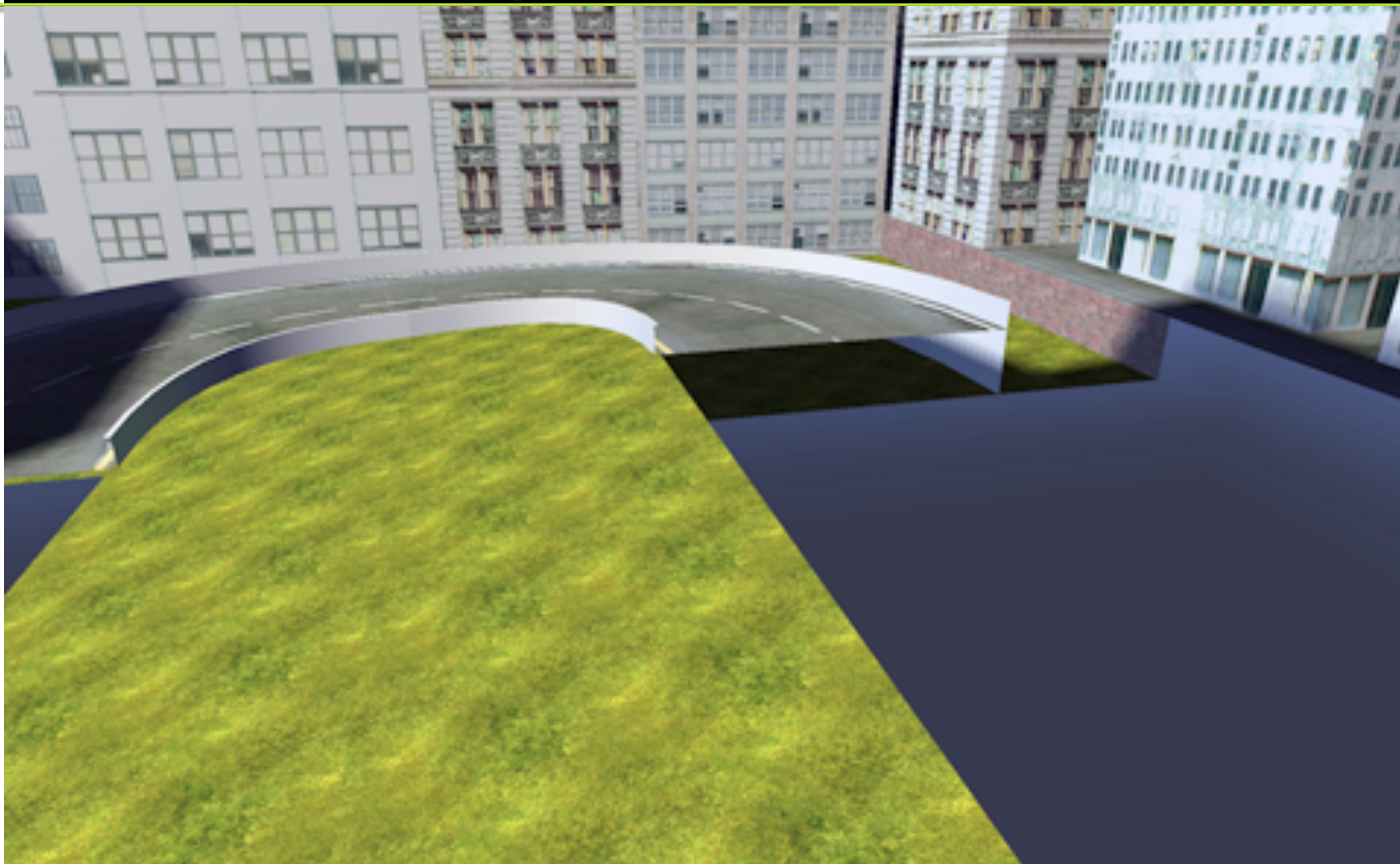


z = 0 Crossing Case

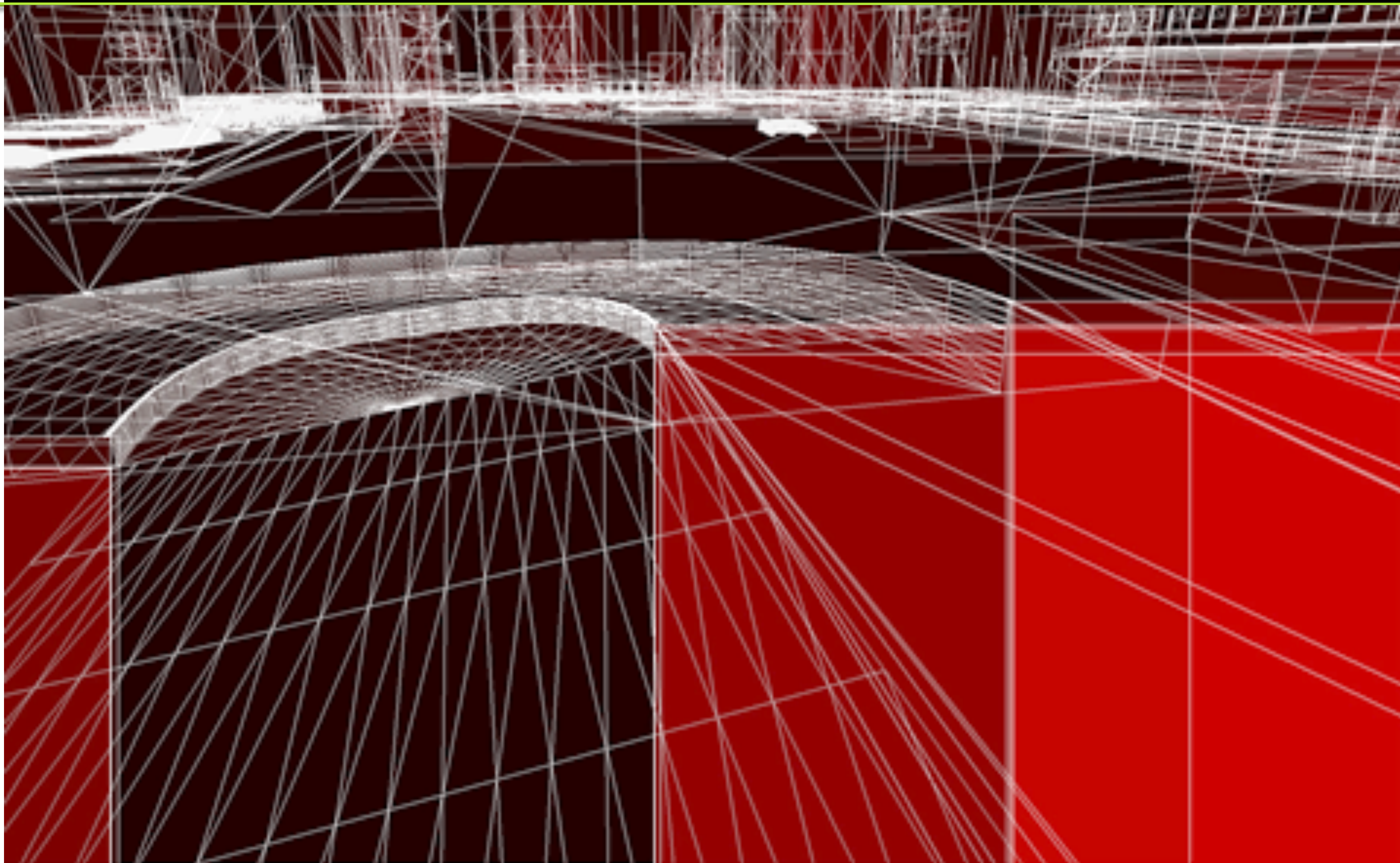
Extreme Example of $z = 0$ Case



Extreme Example of $z = 0$ Case



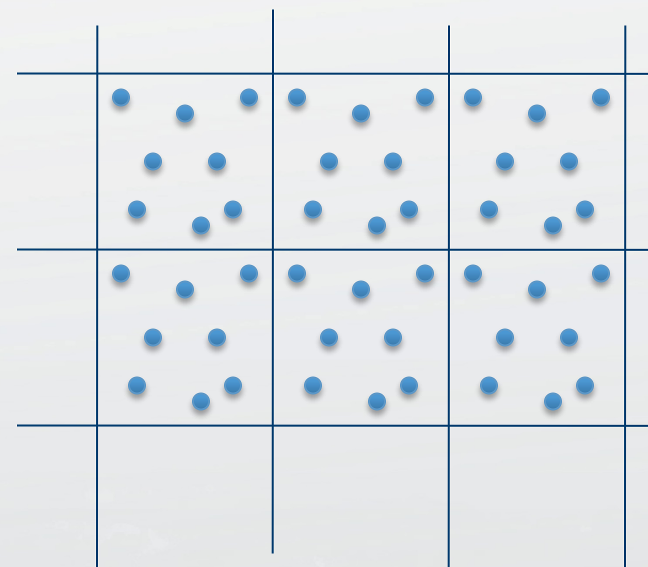
Extreme Example of $z = 0$ Case



Correct result for a moving camera

MSAA

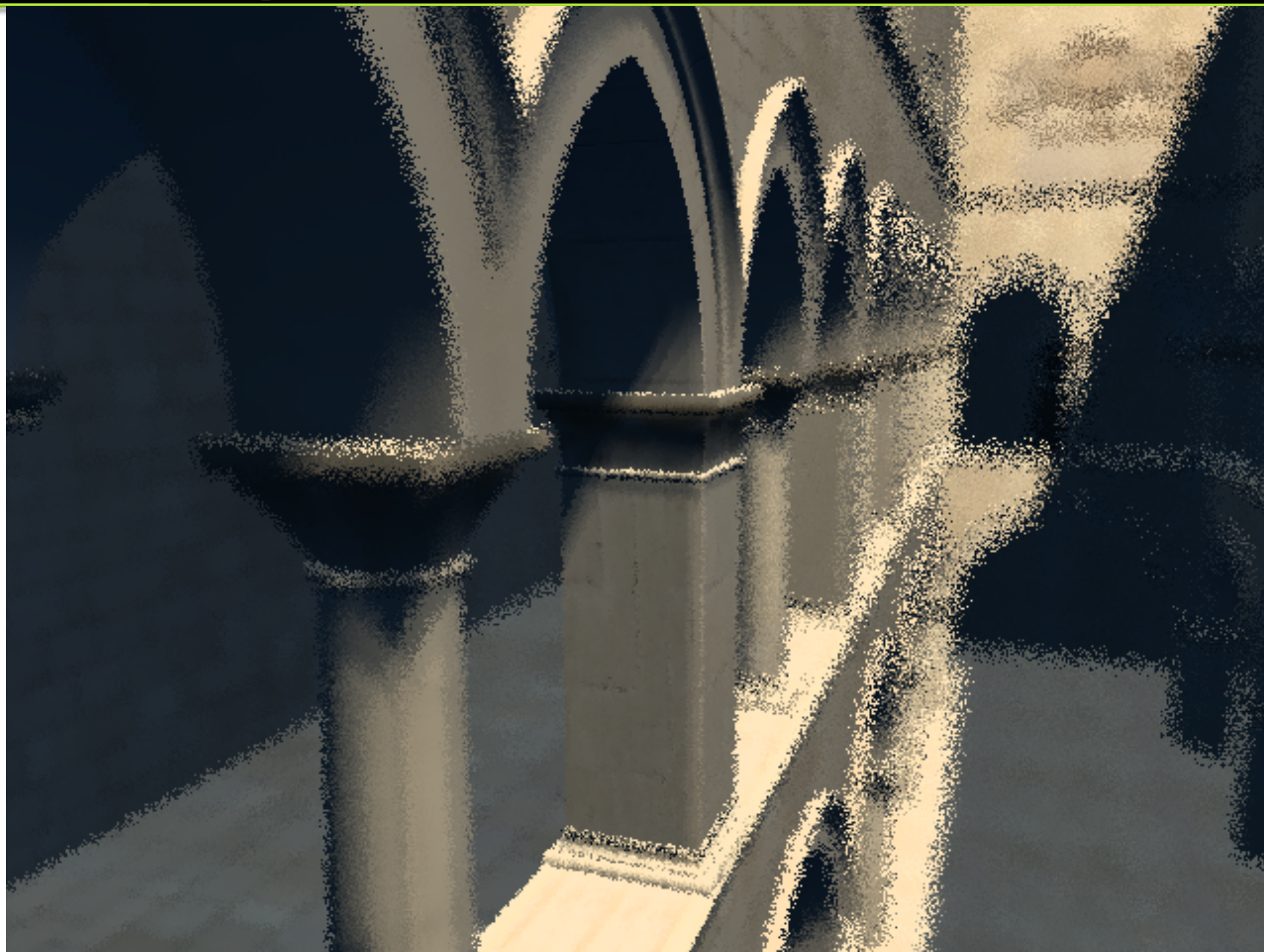
- Cast one visibility ray per sample
- Set the pixel's coverage mask
- Shade at most once per pixel



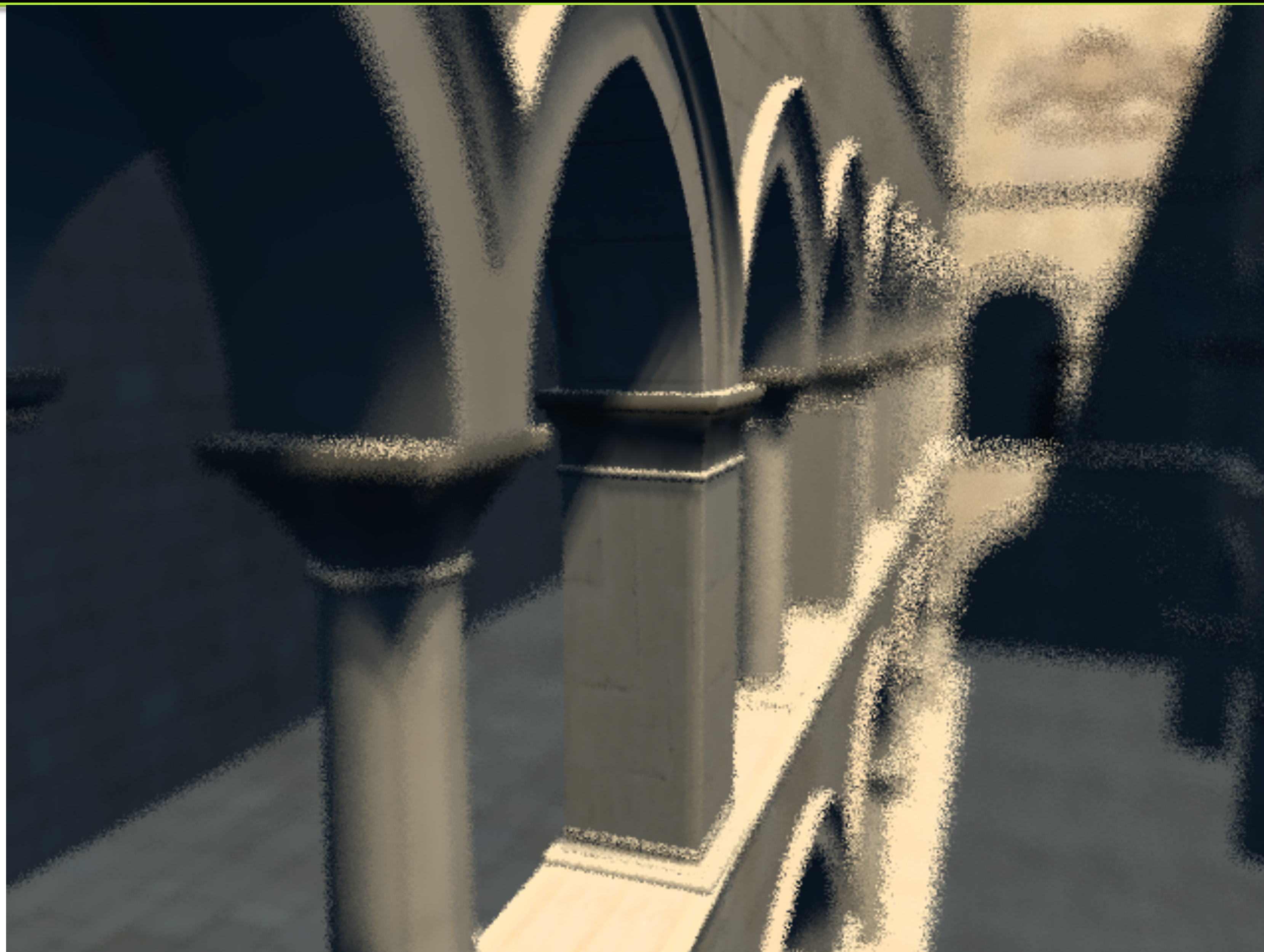
- Use ray differentials to determine anisotropic *xytuv* MIP-map filtering and level [Loviscach 05]

RESULTS

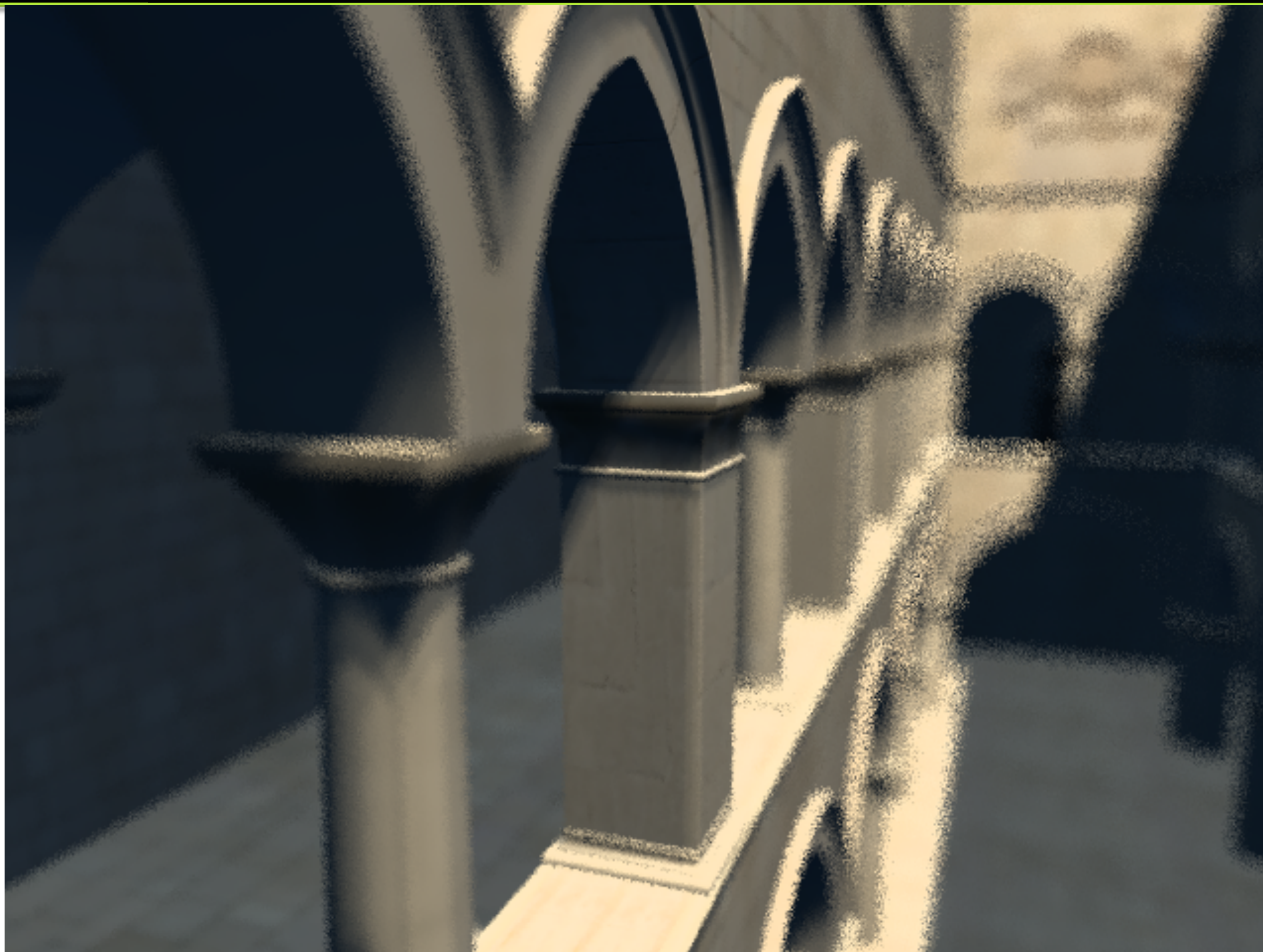
Multisample Rate: 1x



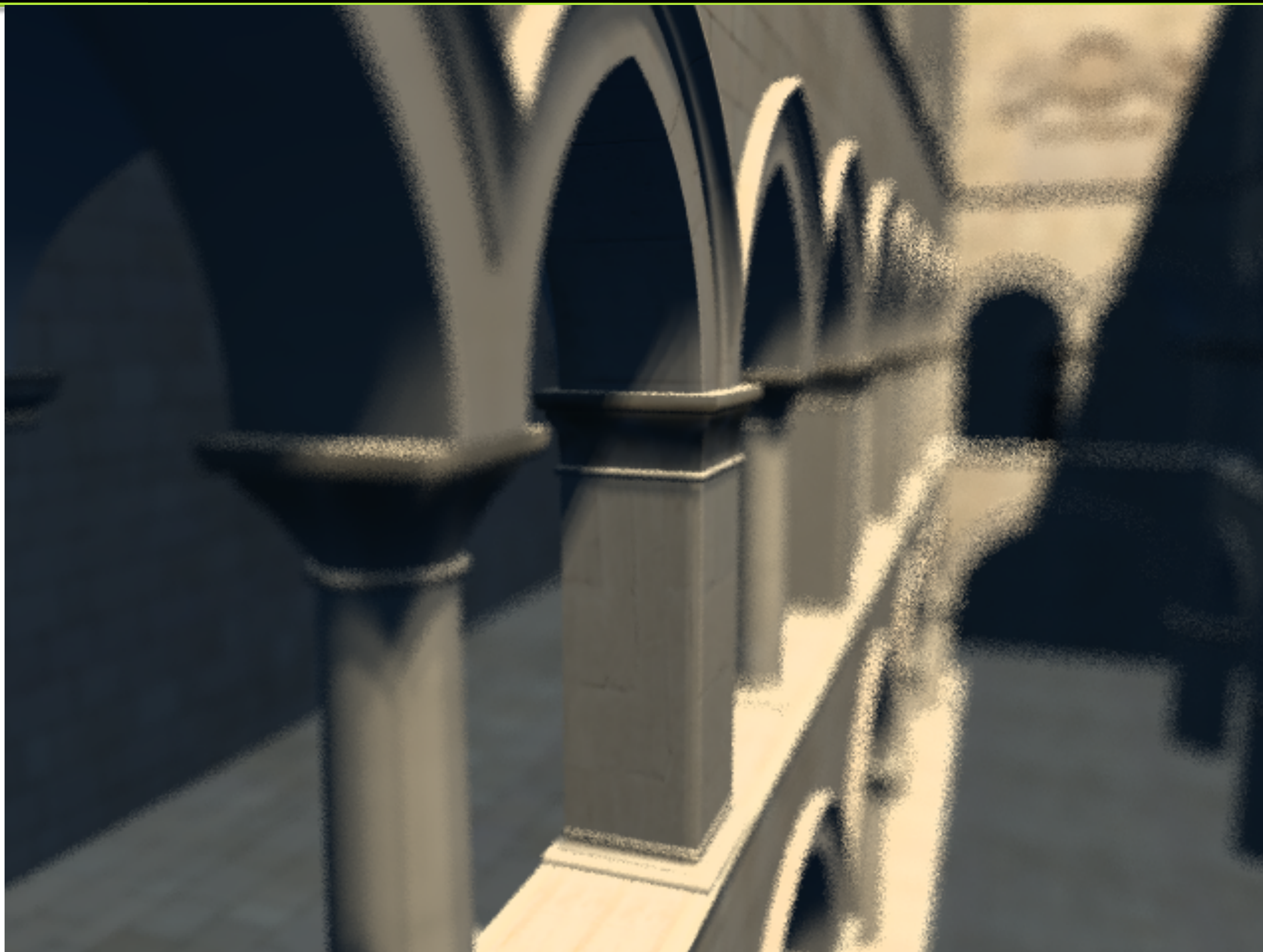
Multisample Rate: 4x



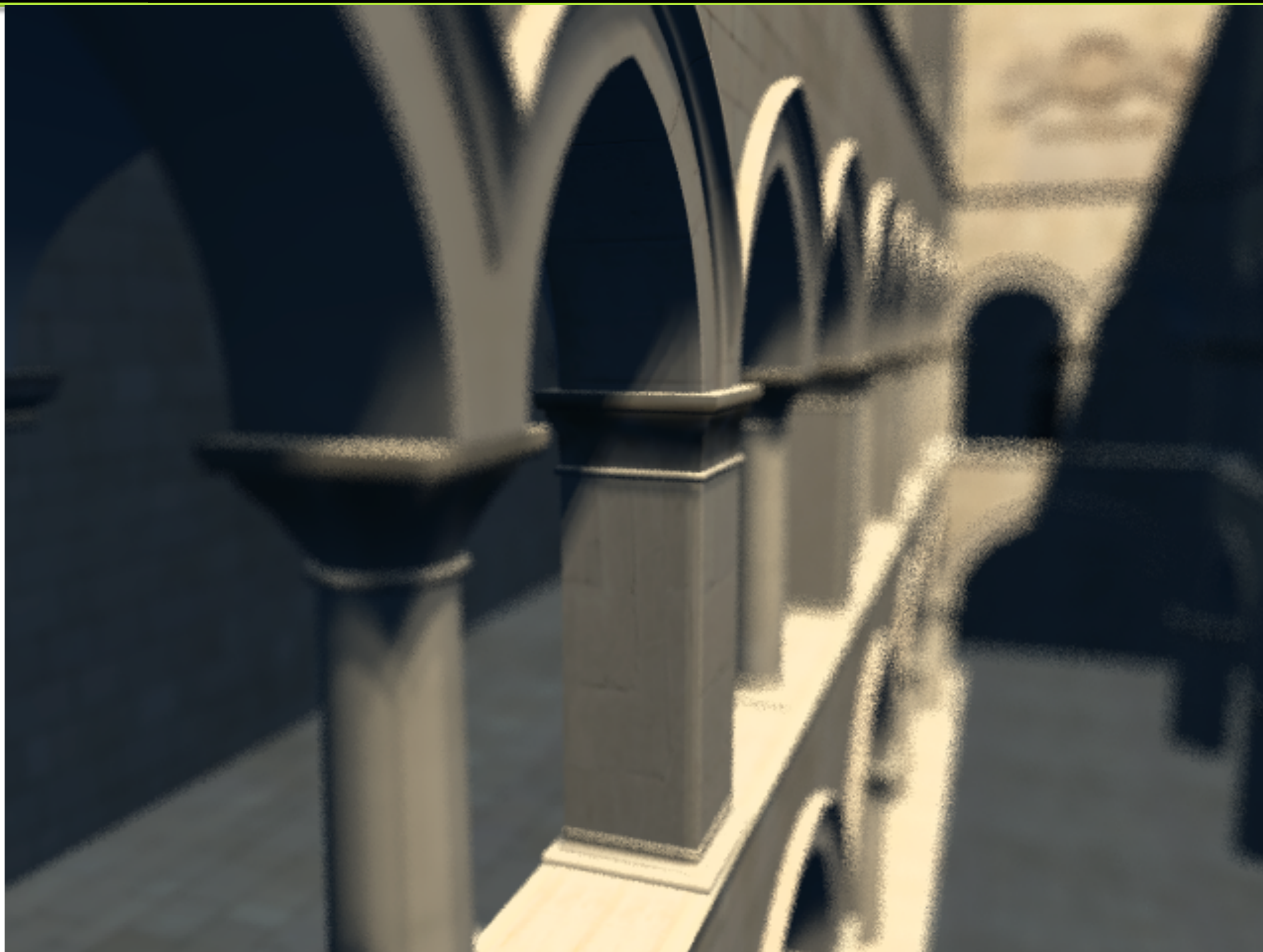
Multisample Rate: 8x



Multisample Rate: 16x



Multisample Rate: 64x



Defocus Blur

Fairy

Defocus blur

174k Triangles

8 vis, 4 tex, 1 shade / pix

1280 × 720 @ 10 fps

GeForce GTX 480

Multisample Rate: 1x

Multisample Rate: 4x

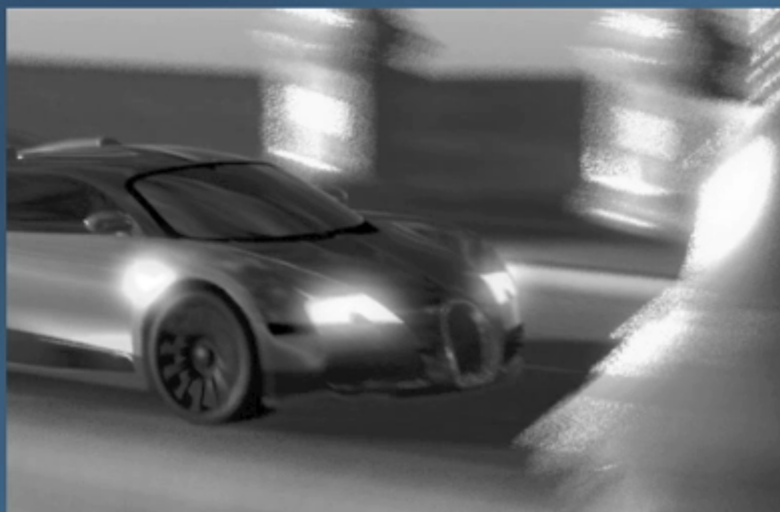
Multisample Rate: 8x

Multisample Rate: 16x

Multisample Rate: 64x

Multisample Rate: 256x

Motion Blur



Bridge

Motion blur

1.8M Triangles

8 vis, 4 tex, 1 shade / pix

1280 × 720 @ 19 fps

GeForce GTX 480

Extended Motion Blur Example

NVIDIA

Race

Motion blur

130k Triangles

8 vis, 4 tex, 1 shade / pix

1280 × 720 @ 72 fps

GeForce GTX 480

Conclusions & Future Work

Real-time stochastic rasterization is possible *now*

- Macrotriangles are much more efficient than micropolygons
- Convex hull radically increases STE
- New efficient hull and z=0 fallback solutions make hull viable

Motion blur is very different from defocus blur

- Integrate post-processed defocus with stochastic motion blur and antialiasing

Fixed-function rasterizer is a power-efficient iterator

- Address the warp branch-coherence problem
- Build a fixed function, time-continuous rasterization unit?

hpg 2010

http://research.nvidia.com

Special thanks to **Heiko Friedrich** for helping to render images in this talk.

ADDITIONAL MATERIAL

Performance

	Vis	Shade	Race	Motion Bridge	Motion Fairy	Defocus Fairy	Defocus Cubes
			130 ktri	1.8 Mtri	174 ktri	174 ktri	50 tri
			Fig 6.	Fig. 1	Fig. 5ur	Fig. 5ll	Fig. 4
	1	1	43.8	9.5	22.8	5.8	104.4
MSAA 4x	4	1	36.2	4.8	15.5	2.6	59.2
SSAA 4x	4	4	16.4	4.3	14.1	3.2	31.0

Frames per second at 1920x1080 on GeForce GT 280
including shadow maps, shading, and tone mapping

GeForce GTX 480 Results are 2x-3x faster

z = 0 Crossing Case

- Project all non-culled (≤ 6) vertices
- Project all (≤ 12) intersections of edges with $z=z_n$
- Bound these (≤ 18) points with a 2D box
- (For defocus, grow the box by the worst circle of confusion)
- Cull if the box is outside the viewport (common)

- *Intuition:*
  - A triangle *is* the hull of its vertices
  - A quadratic patch lies within the hull of its control points
  - The hull of the edges is the hull of the moving triangle

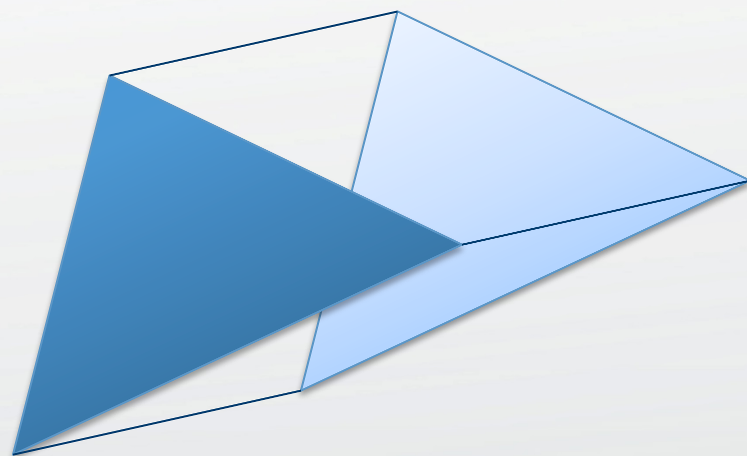
Our Temporal Bounding Solution

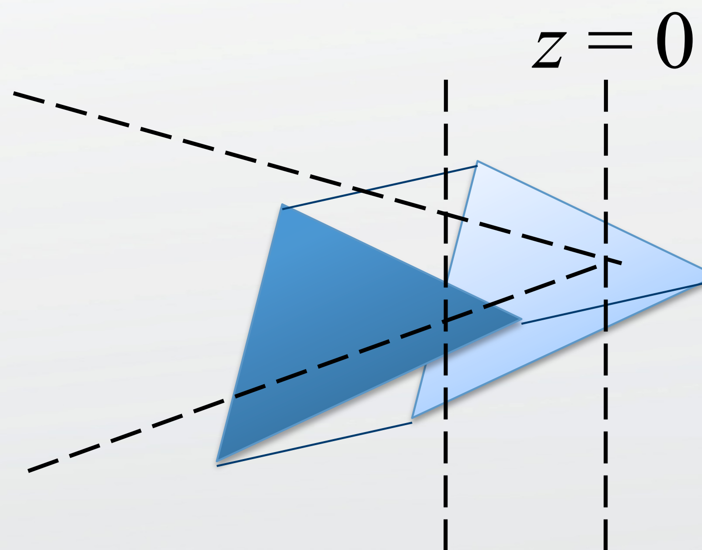
NVIDIA

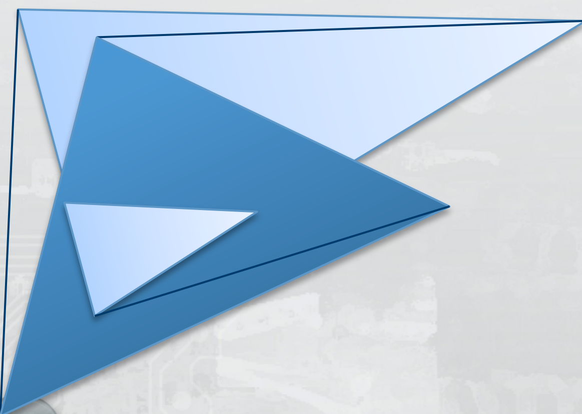
- For triangles entirely culled by z=0
  - Cull!
- For triangles entirely not culled by z=0
  - Project all 6 vertices
  - Solve for 2D convex hull [new algorithm]
- For triangles crossing z=0
  - Solve for 2D bounding box of non-clipped portion [new algorithm]

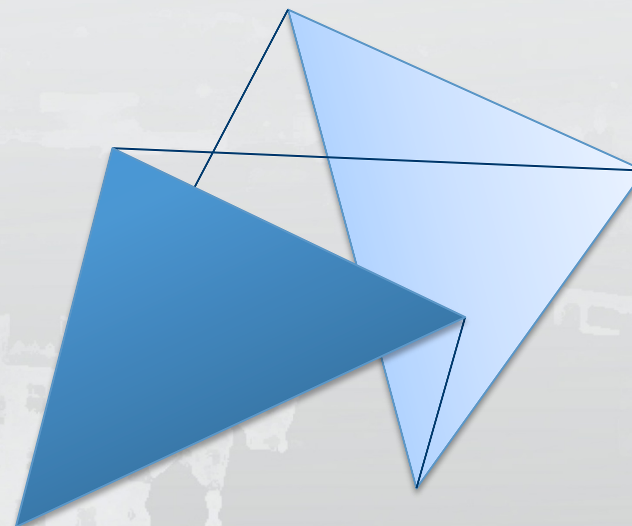
hpg 2010

Bounding Motion is Hard

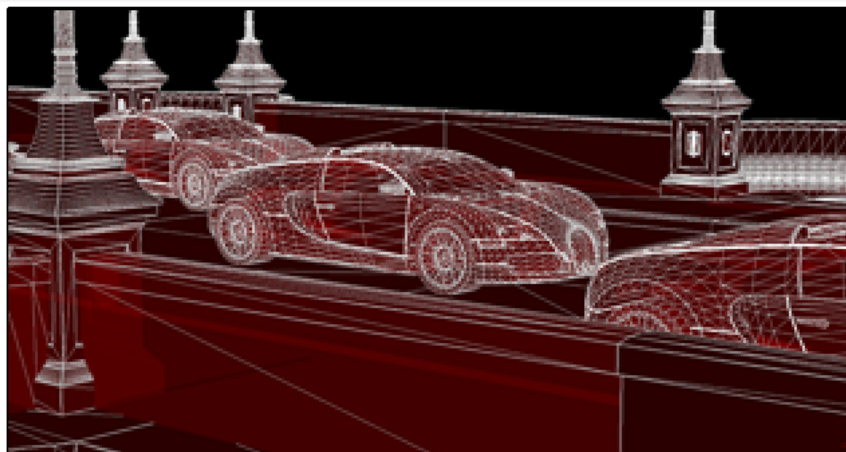




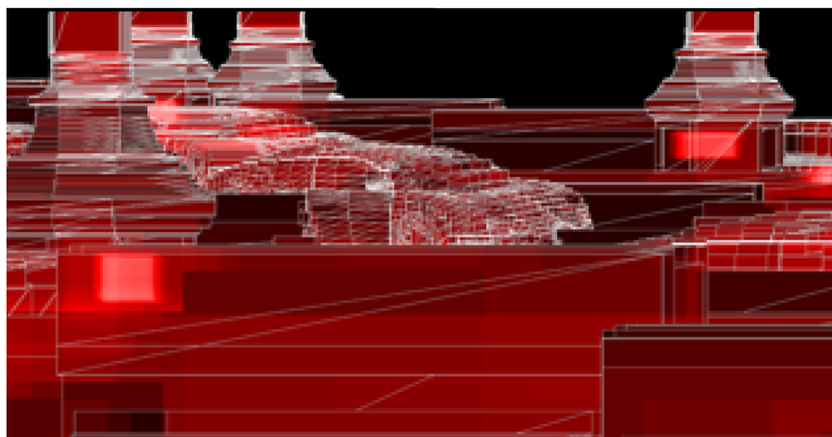




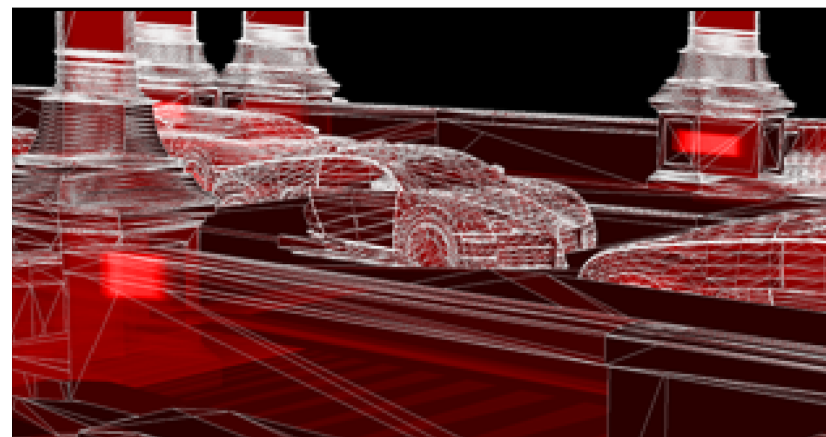
Sample Test Efficiency



Scene



AABB



Convex Hull