11 Shadow Volumes

Steve Marschner

CS5625 Spring 2019

References

F. Crow, "Shadow Algorithms for Computer Graphics." SIGGRAPH 1977.

- http://dx.doi.org/10.1145/965141.563901

M. McGuire, "Efficient Shadow Volume Rendering." ***GPU Gems*****, 2004.**

- https://developer.nvidia.com/gpugems/GPUGems/gpugems_ch09.html

M. Stich et al., "Efficient and Robust Shadow Volumes Using Hierarchical Occlusion Culling and Geometry Shaders." ***GPU Gems 3*****, 2008.**

- https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch11.html

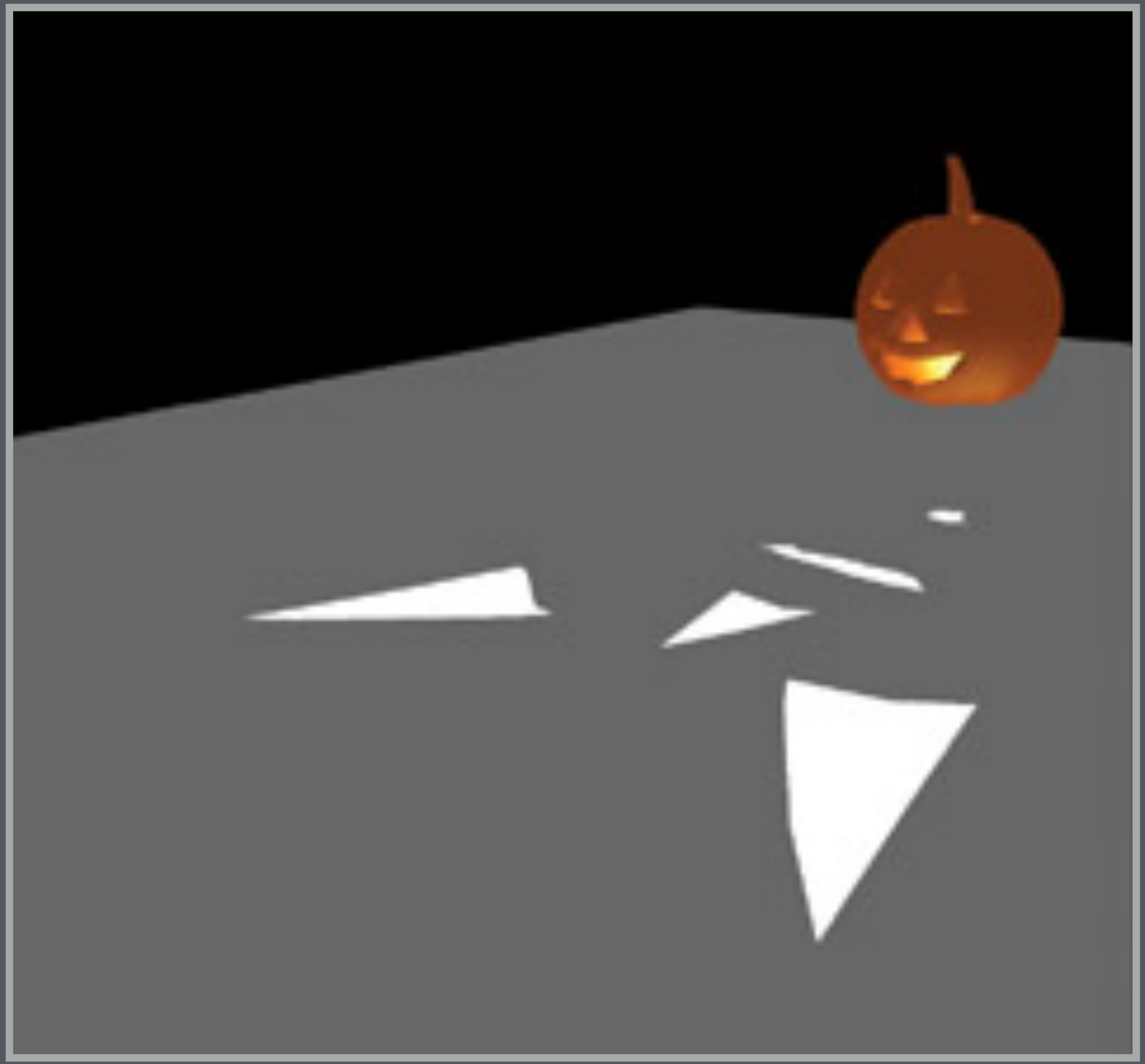
E. Lengyel, "Projection Matrix Tricks." Presentation at GDC 2007.

- http://www.terathon.com/gdc07_lengyel.pdf

J. Gerhards et al. "Partitioned Shadow Volumes." EUROGRAPHICS 2015.

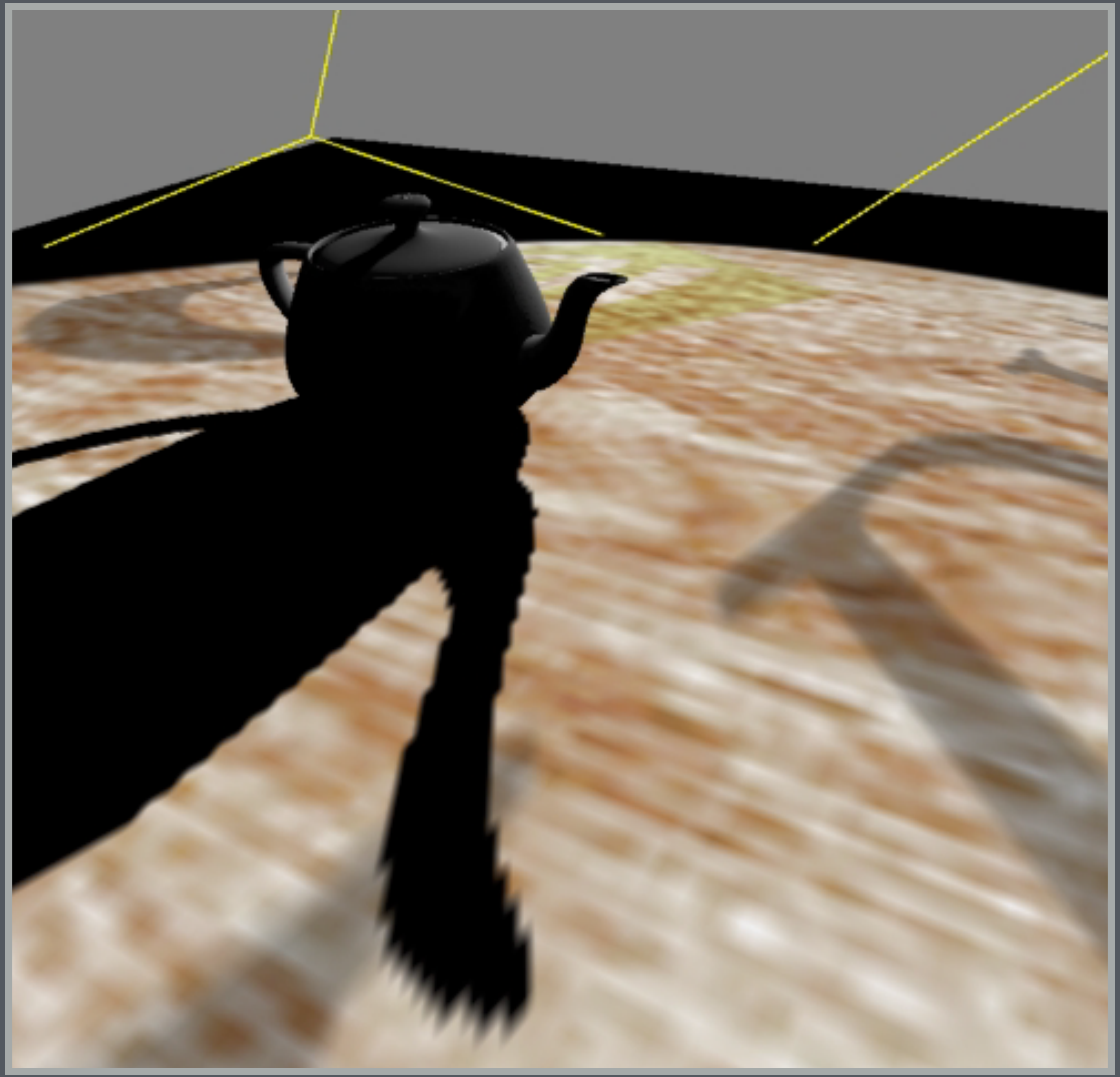
- http://www.unilim.fr/pages_perso/frederic.mora/pdf/psv.pdf

Problem cases for shadow maps



Morgan McGuire, GPU Gems

Problem cases for shadow maps



Mark Kilgard, NVIDIA Inc.

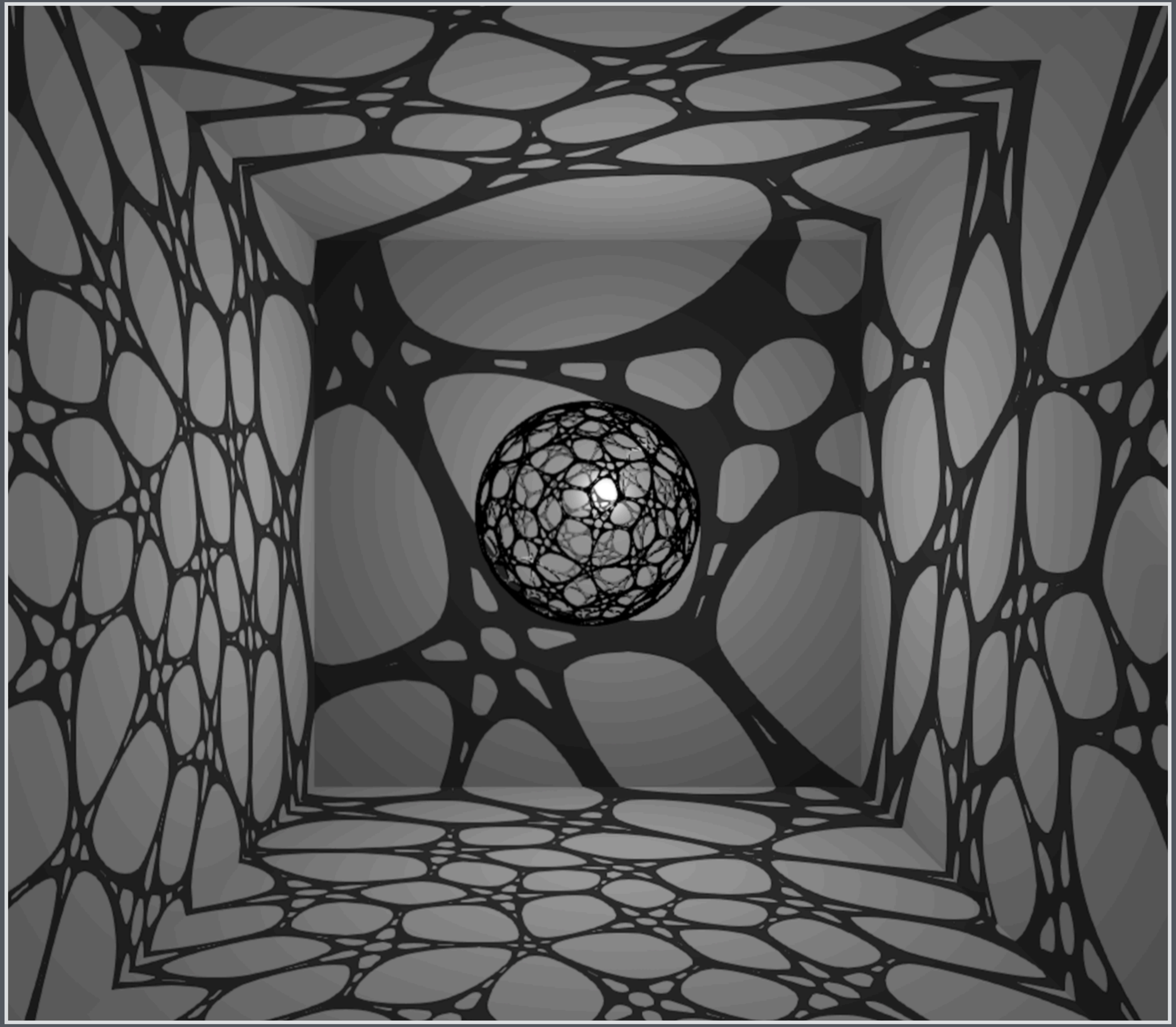
Shadow Volumes

- Crow 1977
- Accurate shadows

Image courtesy of BioWare Neverwinter Nights

slide courtesy of Kavita Bala, Cornell University

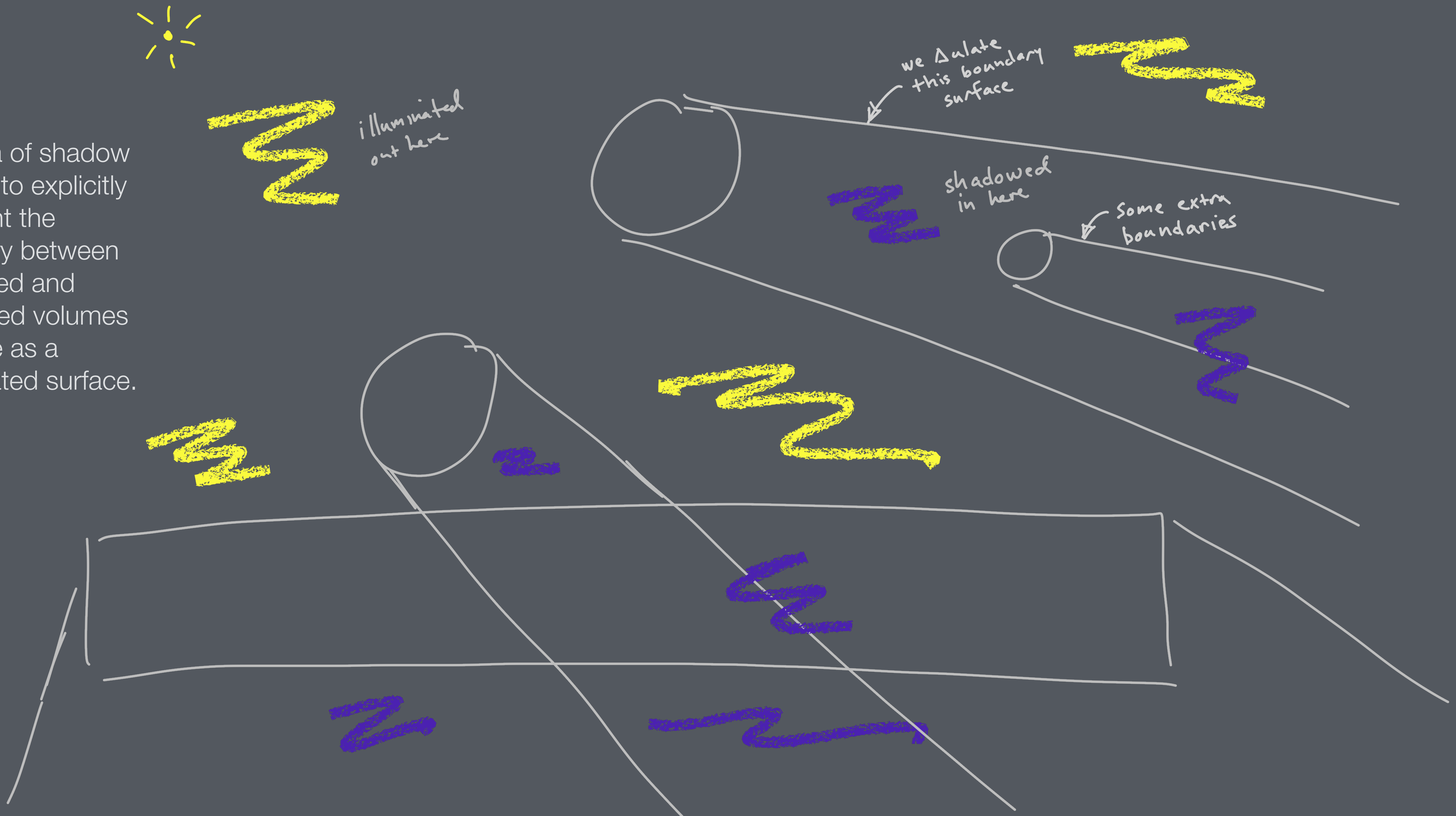
Shadow volume robustness



Gerhards et al. EG 2015

Illuminated volume

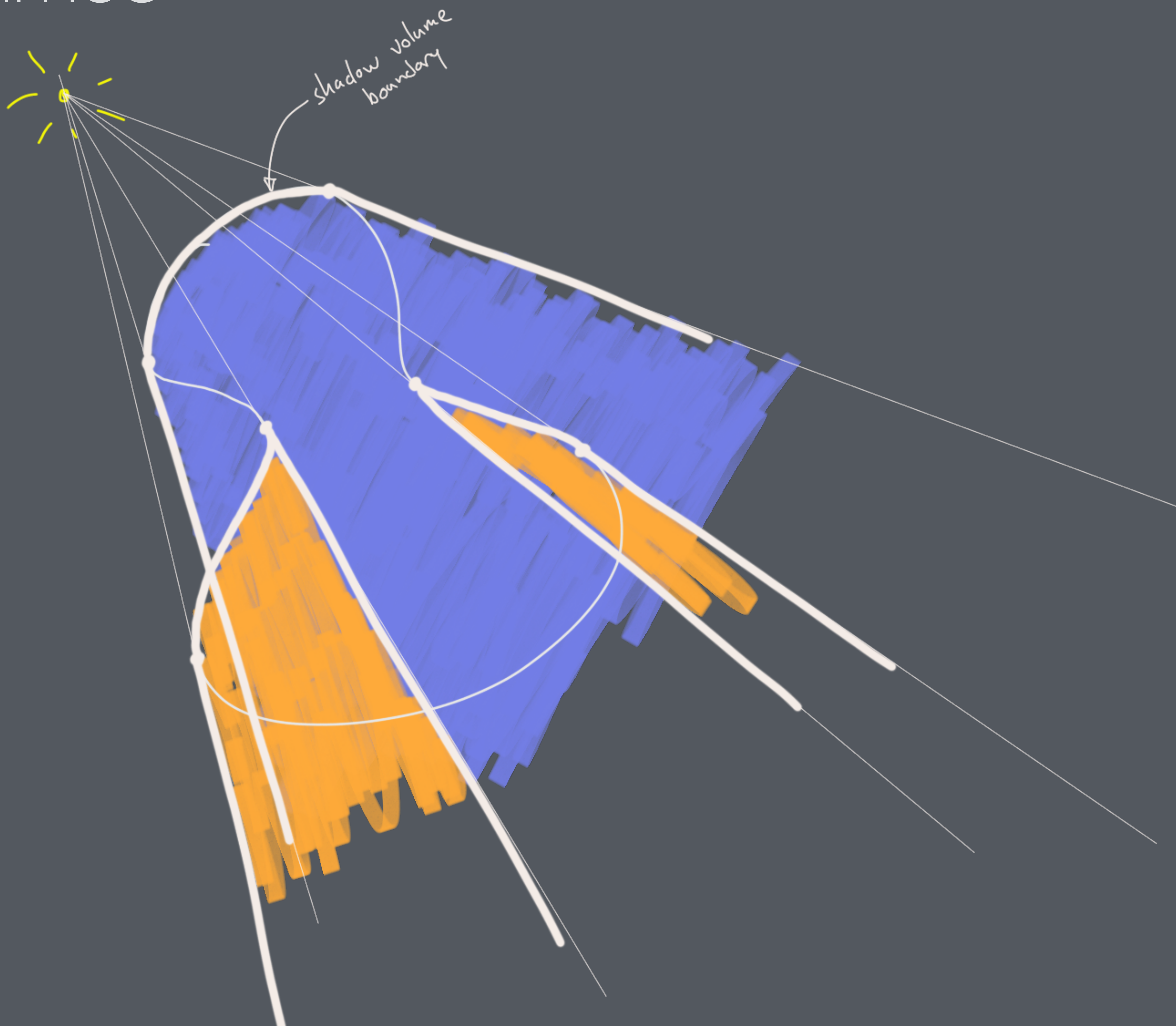
The idea of shadow maps is to explicitly represent the boundary between shadowed and illuminated volumes of space as a triangulated surface.



Overlap of shadow volumes

In 2D, silhouette points divide closed curves into segments that face **toward** and **away** from the light. Each light-facing segment creates a shadow area.

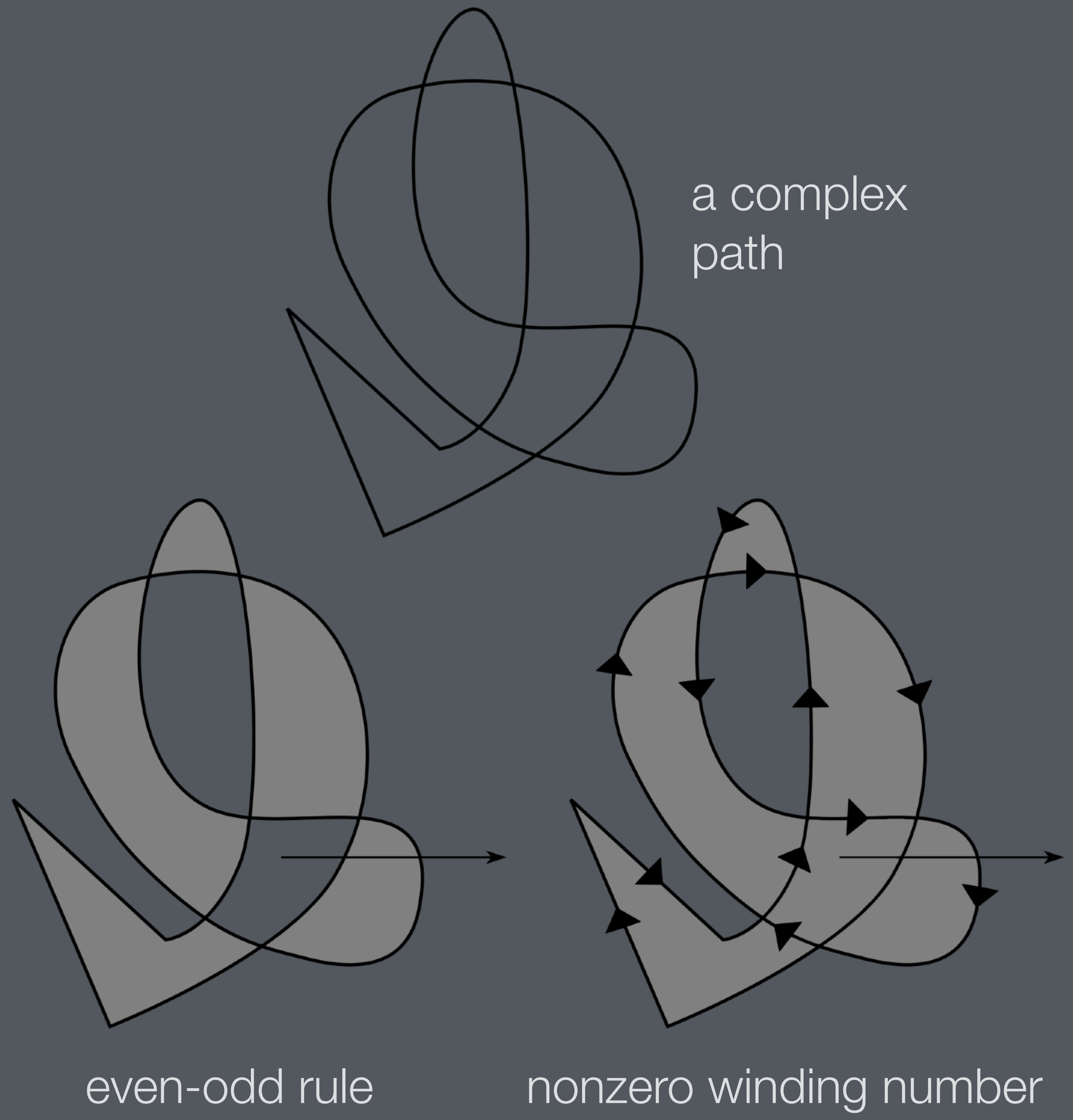
In 3D, silhouette edges divide closed surfaces into regions that are **front-facing** and **back-facing** to the light. Each front-facing region creates a shadow volume.



Determining insideness

Filling 2D shapes, at least two ways to define filled area

- even-odd rule: if a ray starting at the point crosses the boundary an odd number of times, the point is inside.
  - nice: don’t need oriented path
  - not so nice: you end up with a lot of holes
- nonzero winding number rule: if the total number of clockwise and counterclockwise crossings of the ray with the boundary are unequal, the point is inside.
  - nice: enclosing a point twice keeps it inside
  - need to have oriented boundary (but you do anyway)



Wikimedia Commons

Determining insideness

In 3D, same rules apply

- nonzero winding number rule will give us the union, which is what we want

For ray, use viewing ray

- traced implicitly by rasterization
- intersections with a ray are fragments that land at a pixel

For counting, use stencil buffer

Stencil buffer

an auxiliary buffer like the depth buffer

integer valued

stencil operation controls how fragments affect stencil buffer

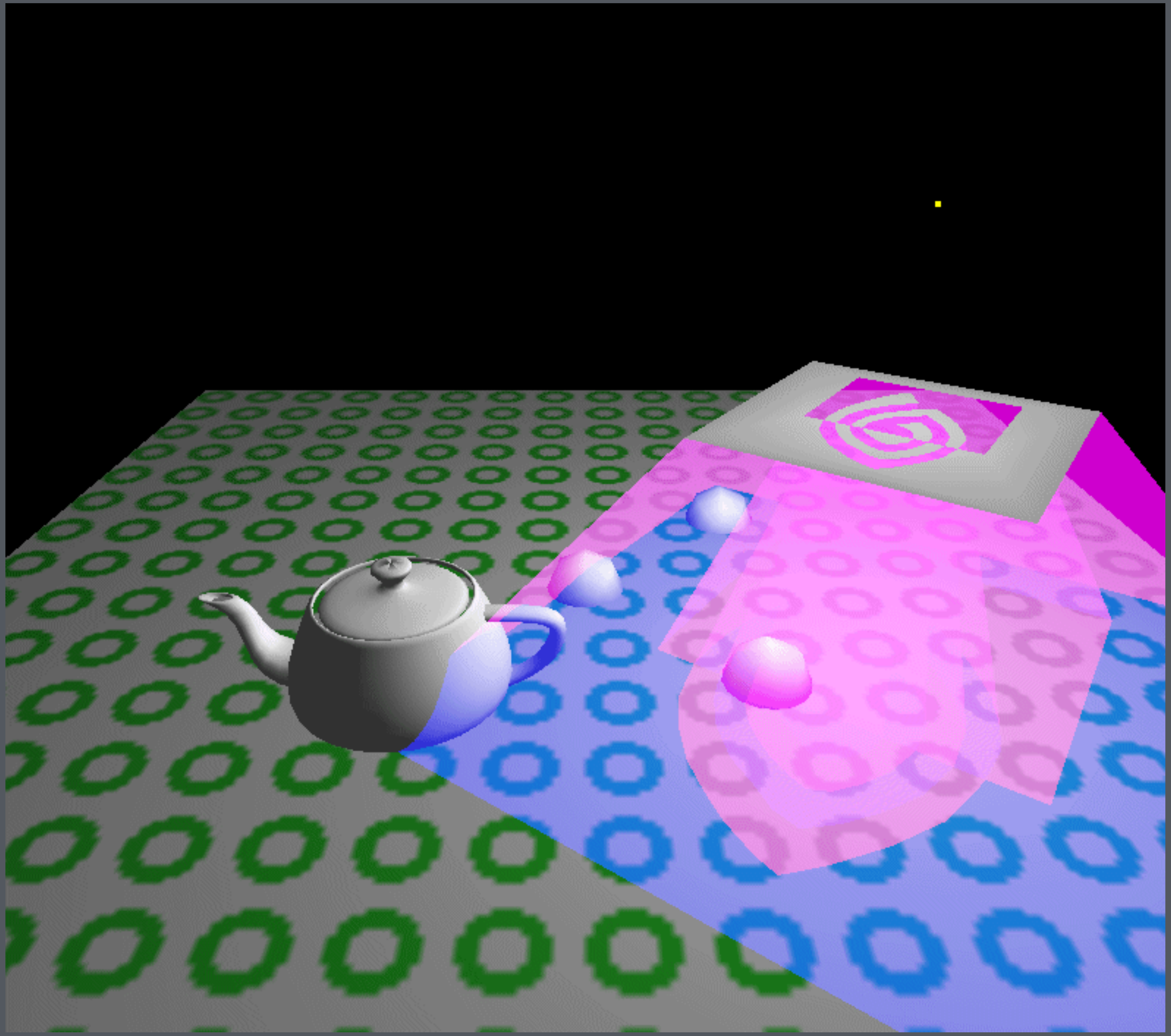
- value can be incremented or decremented
- can have different behavior for front or back facing fragments
- can choose to process only fragments that pass or fail the depth test

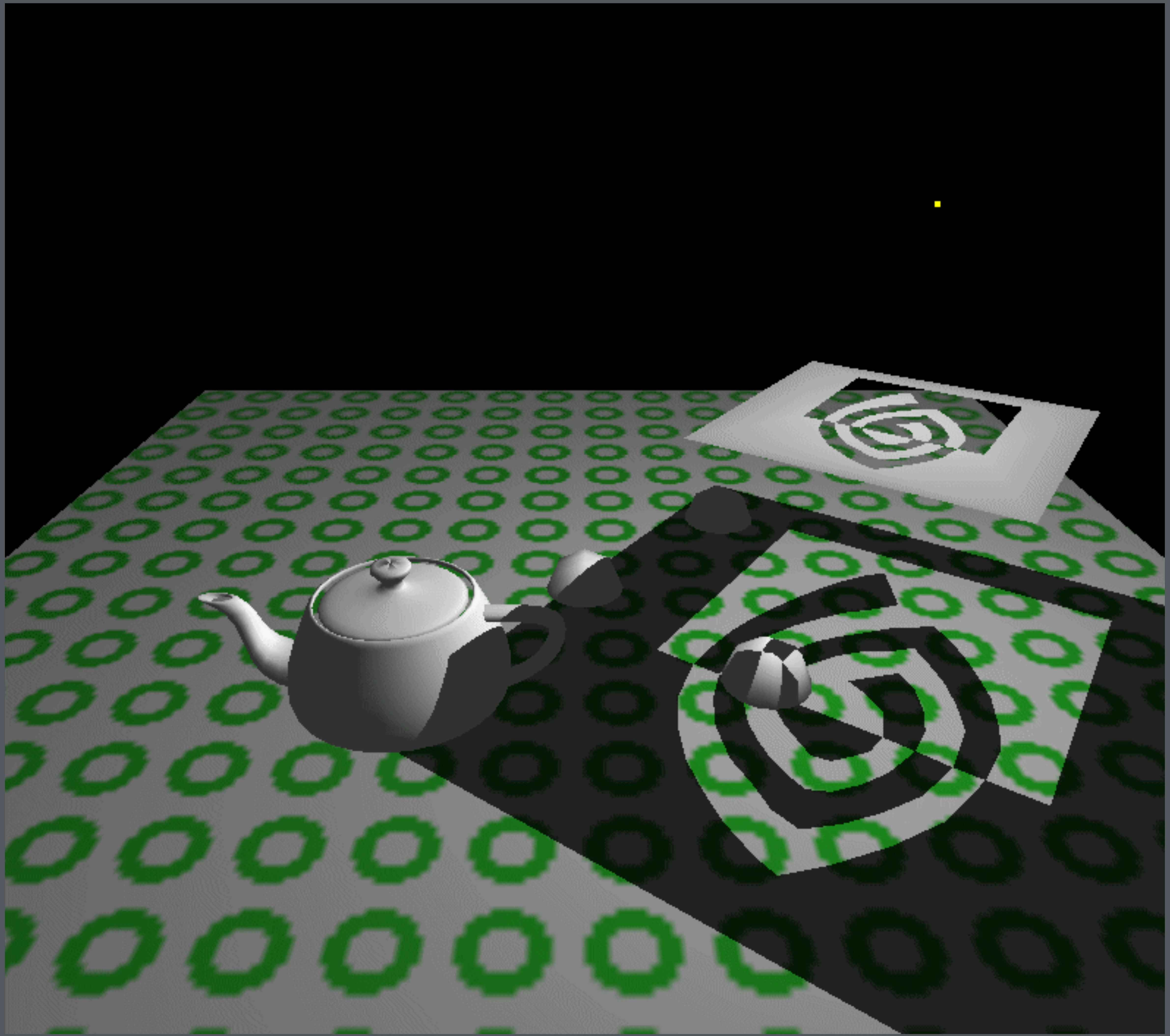
stencil test controls discarding of fragments based on stencil buffer

- similar to depth test
- can discard fragments when value is greater than, less than, etc. a constant value

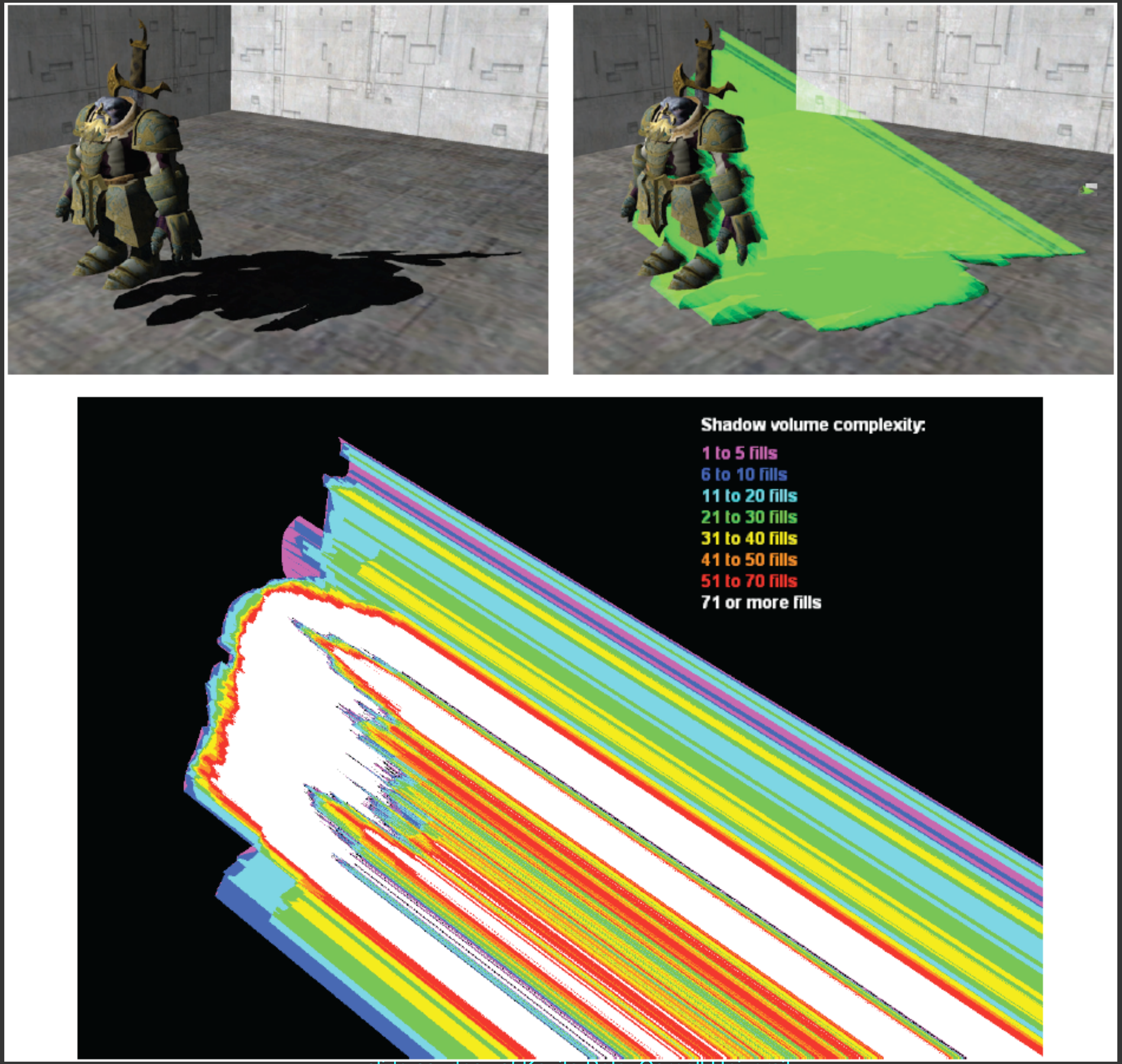
Stencil buffer and shadow volumes

1. **Draw the scene normally but omitting direct light**
    - result: color buffer, depth buffer
2. **Draw the shadow volume boundary**
    - configure stencil operation to add up entries and exits along viewing ray
    - use ray from fragment position towards eye: pay attention only to shadow boundary fragments that **pass** the depth test (are closer than the z-buffer depth)
3. **Draw the scene again, this time adding direct light**
    - configure stencil test to discard fragments with nonzero winding number
    - only unshadowed fragments are drawn





Mark Kilgard, NVIDIA Inc.



slide courtesy of Kavita Bala, Cornell University

Details

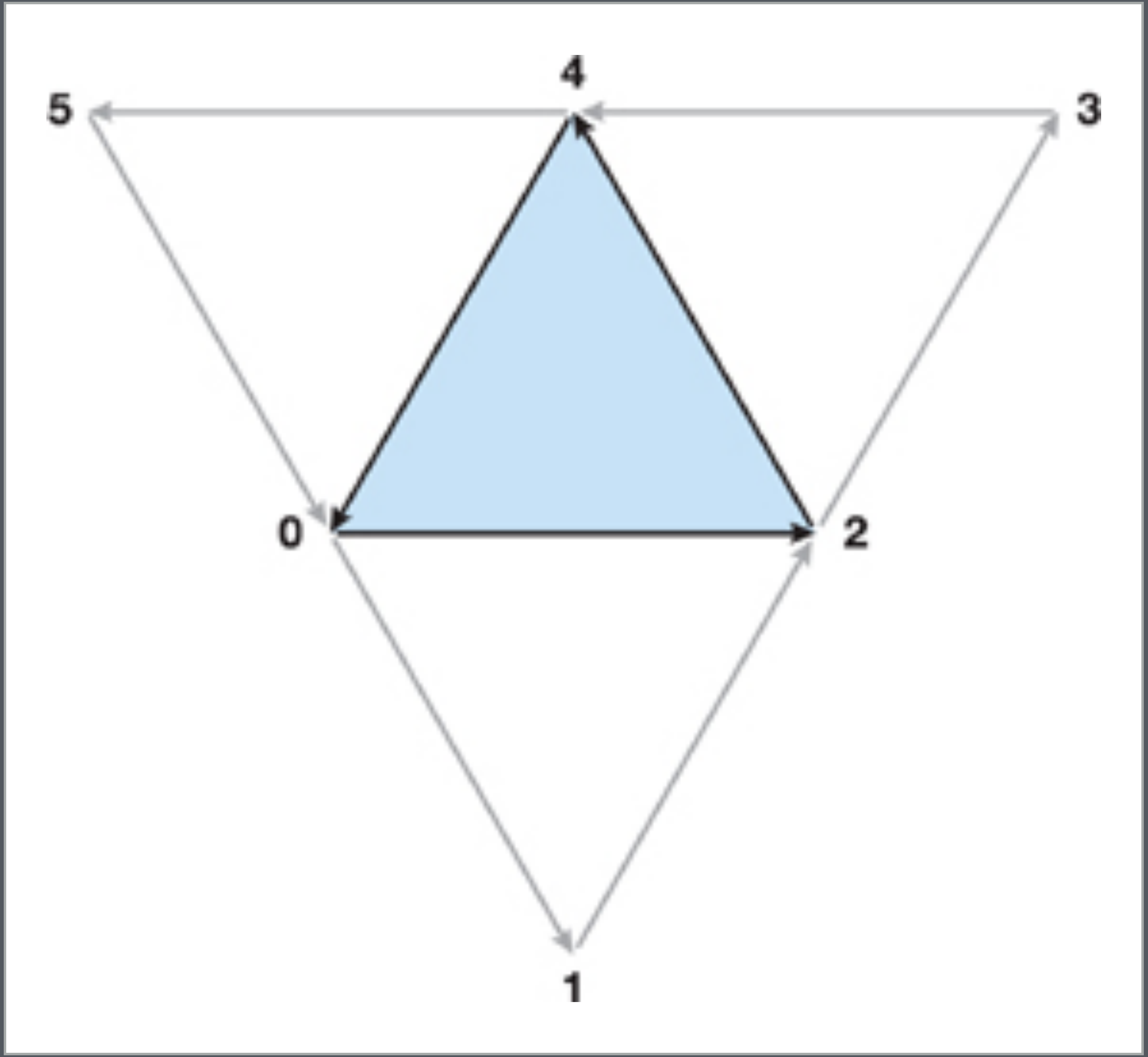
What polygons to draw

- a quad per shadow volume edge
- 2 vertices are at infinity

Generating these polygons

- can use a geometry shader for this

Geometry shader for shadow volumes



Stich et al. GPU Gems 3

Shader outputs:

- one quad for each silhouette edge
  - check for silhouettes using adjacent vertex information
- for z-fail version, the triangle (front cap)
- for z-fail version, the triangle projected to infinity and inverted (back cap)

Primitive type:
GL_TRIANGLES_ADJACENCY
or GL_TRIANGLE_STRIP_ADJACENCY

Problems

Viewpoint in shadow: wrong answers

- the ray doesn't exit the volume to get its winding number to 0
- same problem if shadow volume surfaces are clipped by near plane

Solution: switch to z-fail version

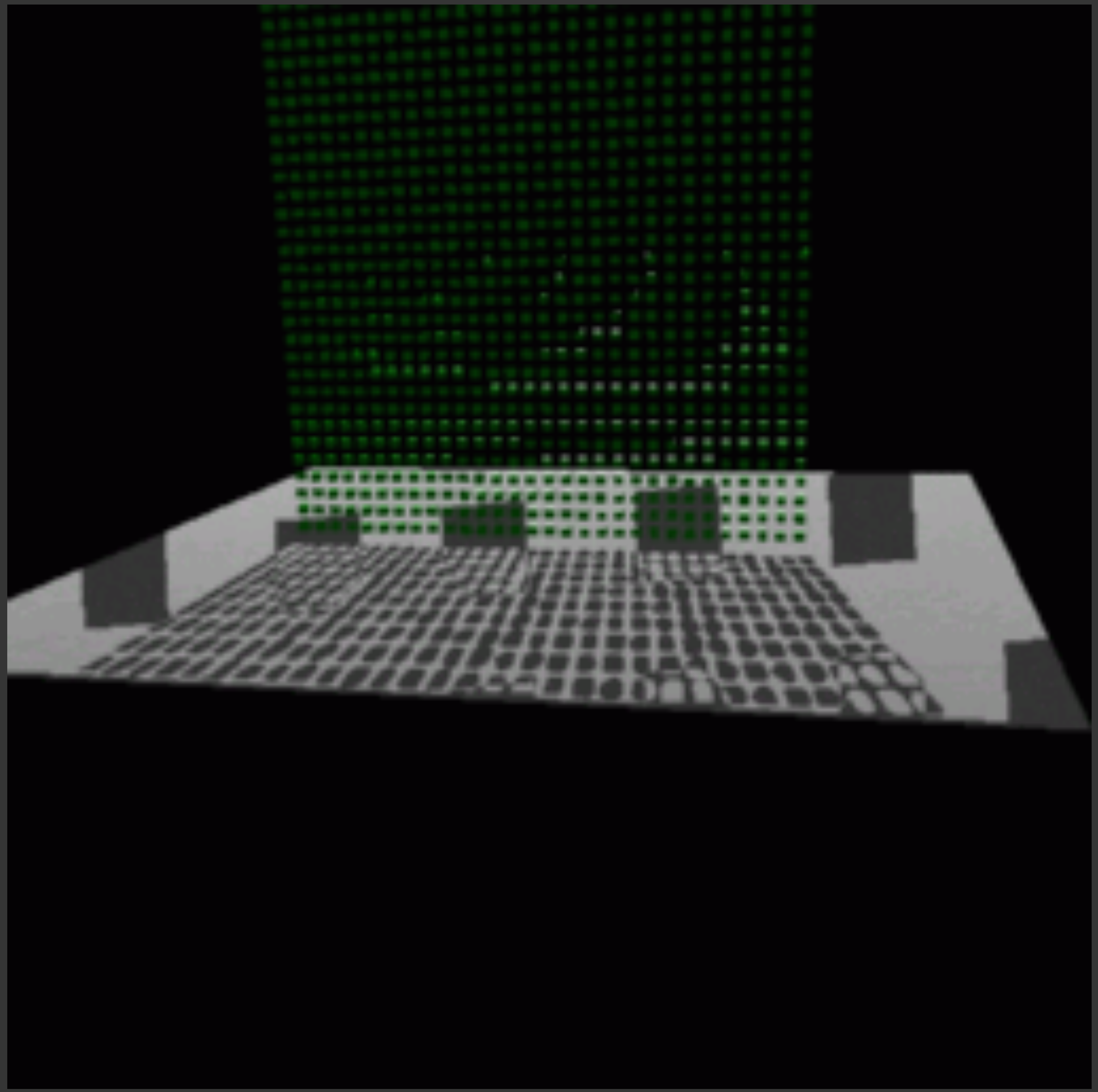
- use the other half of the viewing ray (from visible surface to infinity)
- simply reverse depth test when drawing volumes

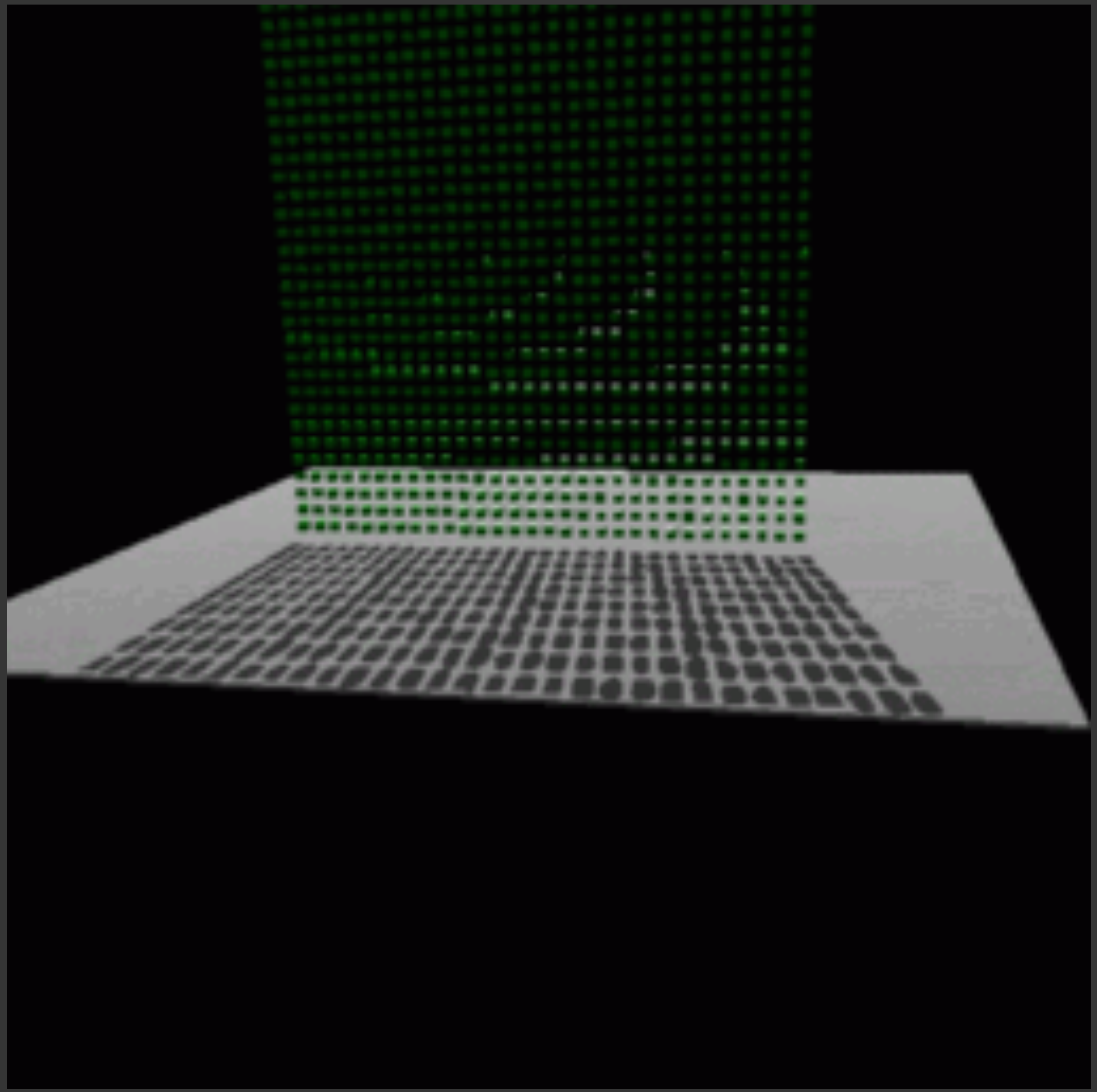
New problem: far plane clips volumes

- solution 1: set up projection matrix with infinite far distance
- solution 2: use depth clamping if available

For z-fail we need the volumes to be closed

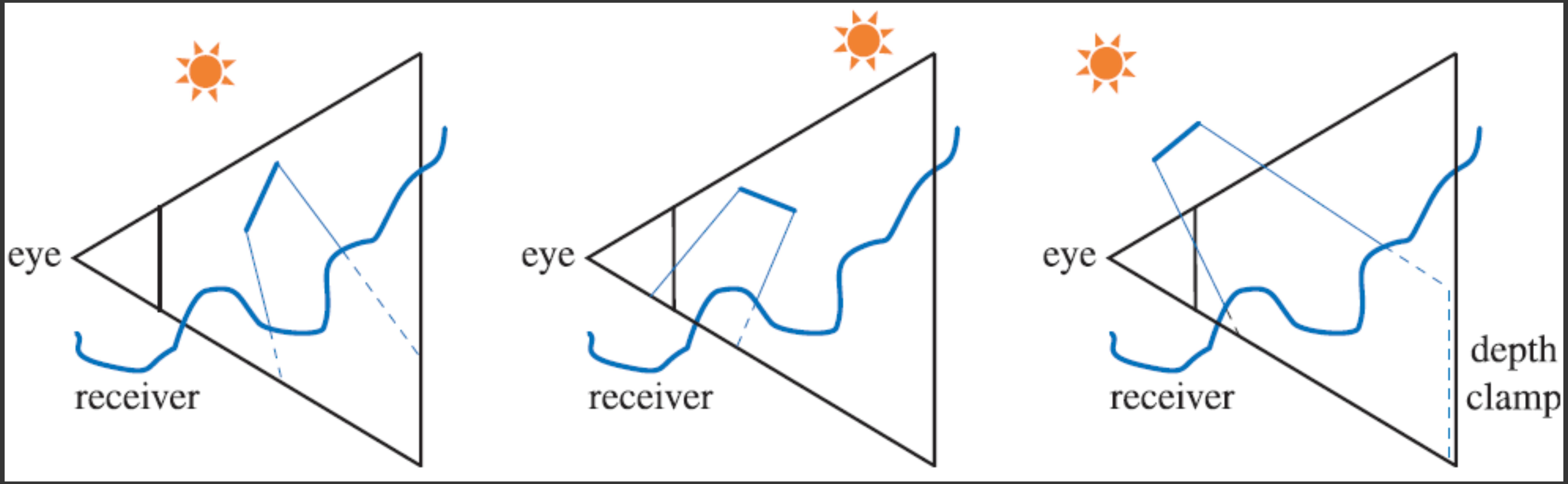
Near Plane Clip Issues





- Near plane clip discards part of shadow volume, messes up count

Clip plane issues



slide courtesy of Kavita Bala, Cornell University

Bottom line: maps vs. volumes

Shadow maps

- usually faster, less fill-limited
- easier to get working
- but… prone to sampling artifacts
- but… require management of shadow fields of view

Shadow volumes

- are always pixel accurate
- can be made very robust
- much less tuning than shadow maps
- but… uses a ton of fragment processing (“fill rate”)