Computer Graphics

Shadow Computation

Tobias Isenberg

informatics mathematics Inria

Overview

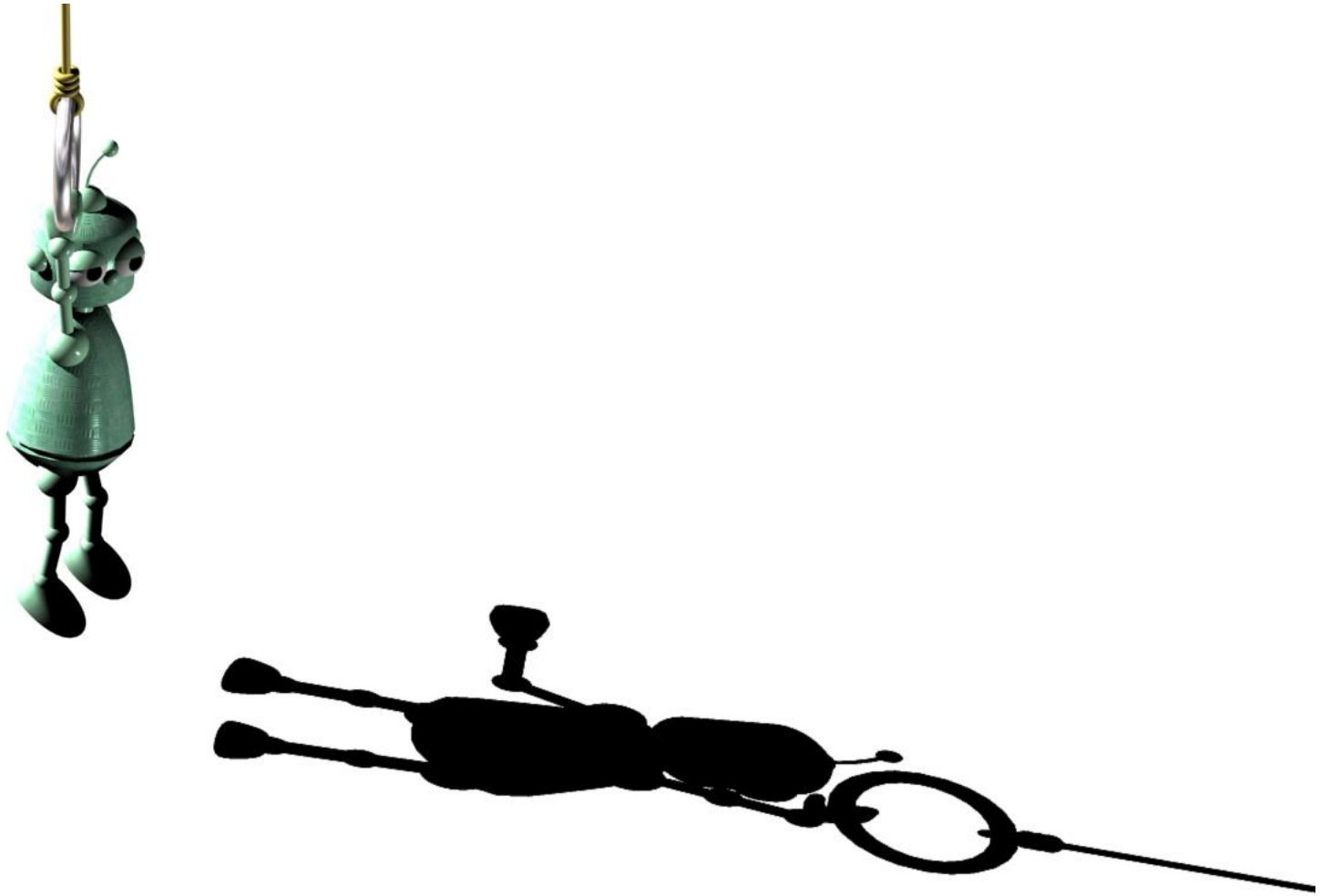
- motivation/introduction

- several shadow computation techniques
  - projected shadows & shadow textures
  - shadow mapping
  - shadow volumes
  - soft shadows

Shadows: Important for perception

© 2003 - Artis

Shadows: Important for perception



© 2003 - Artis

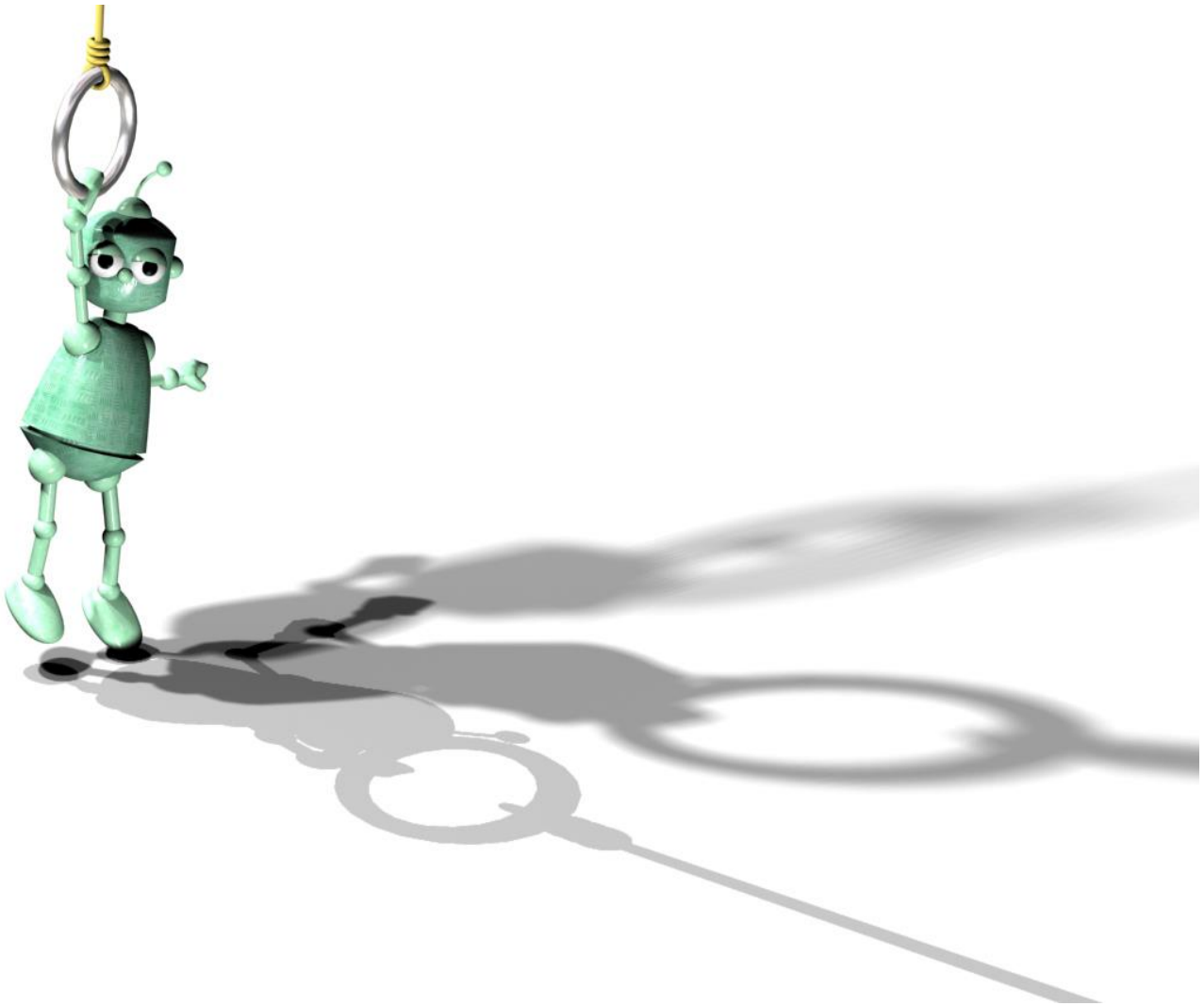
Shadows: Important for perception

© 2003 - Artis

Shadows: Important for perception

© 2003 - Artis

Shadows: Important for perception



© 2003 - Artis

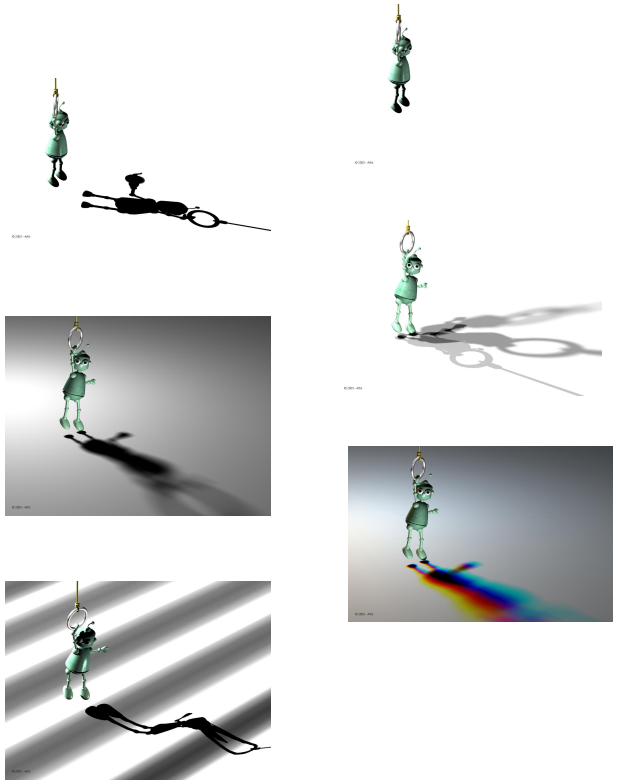
Shadows: Important for perception

Shadows: Important for perception

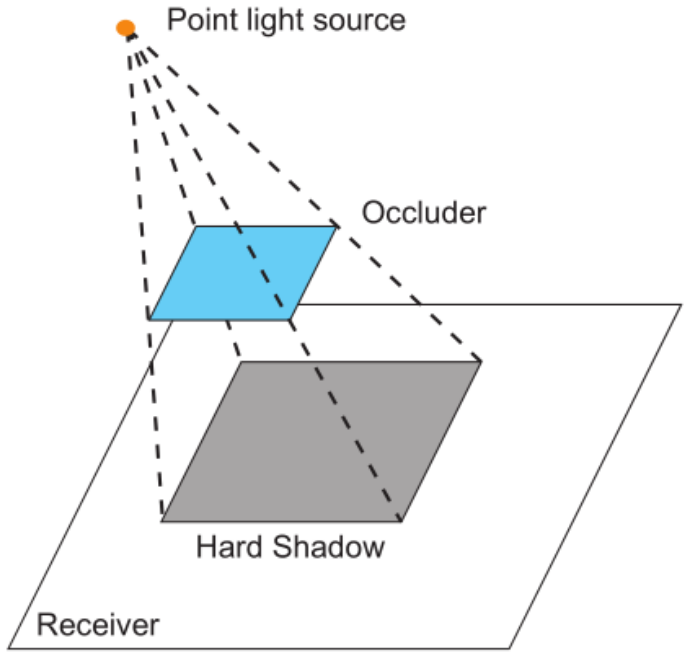
Shadows: Important for perception

Shadows: Important for perception

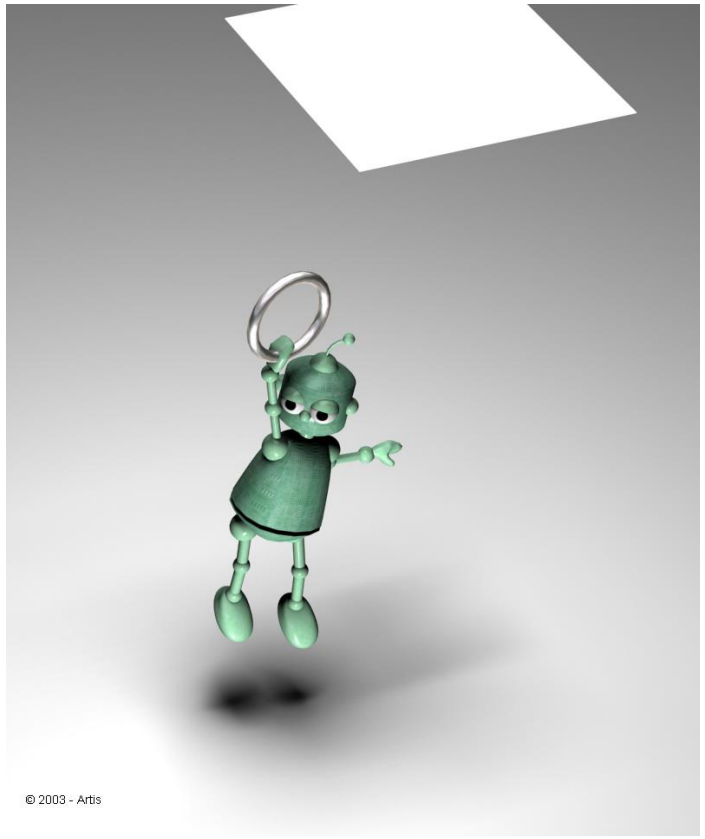
- position in 3D space
- recognition of hidden shapes
- depends on the number of light sources
- depends on the type of light sources
- depends on the color of light sources
- reveals shape of (back)ground surface

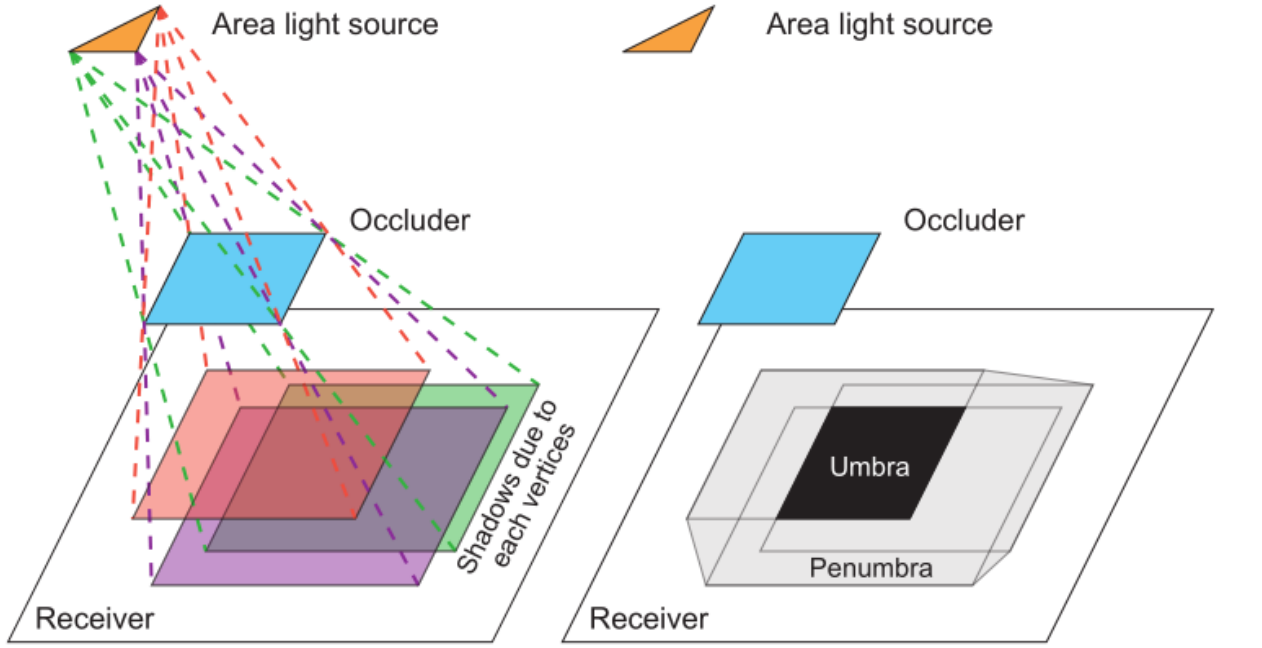


Shadows: Terminology/shadow types

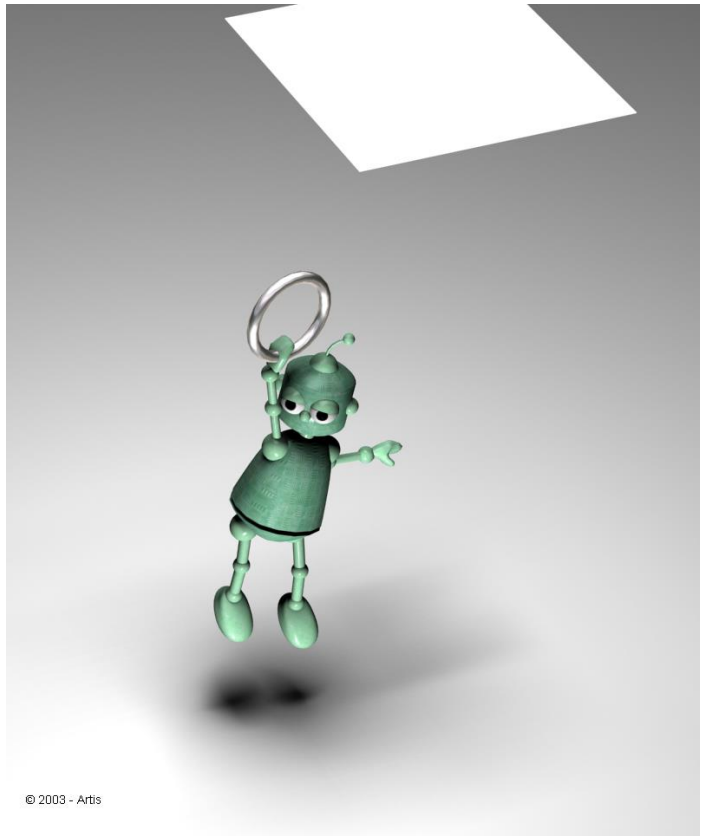


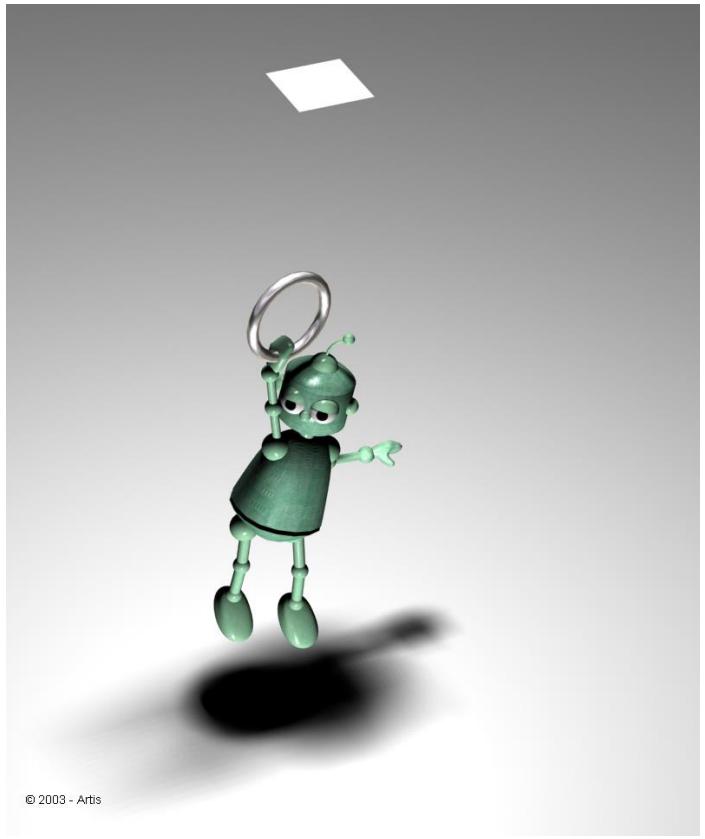
Shadows: Terminology/shadow types





Shadows: Terminology/shadow types

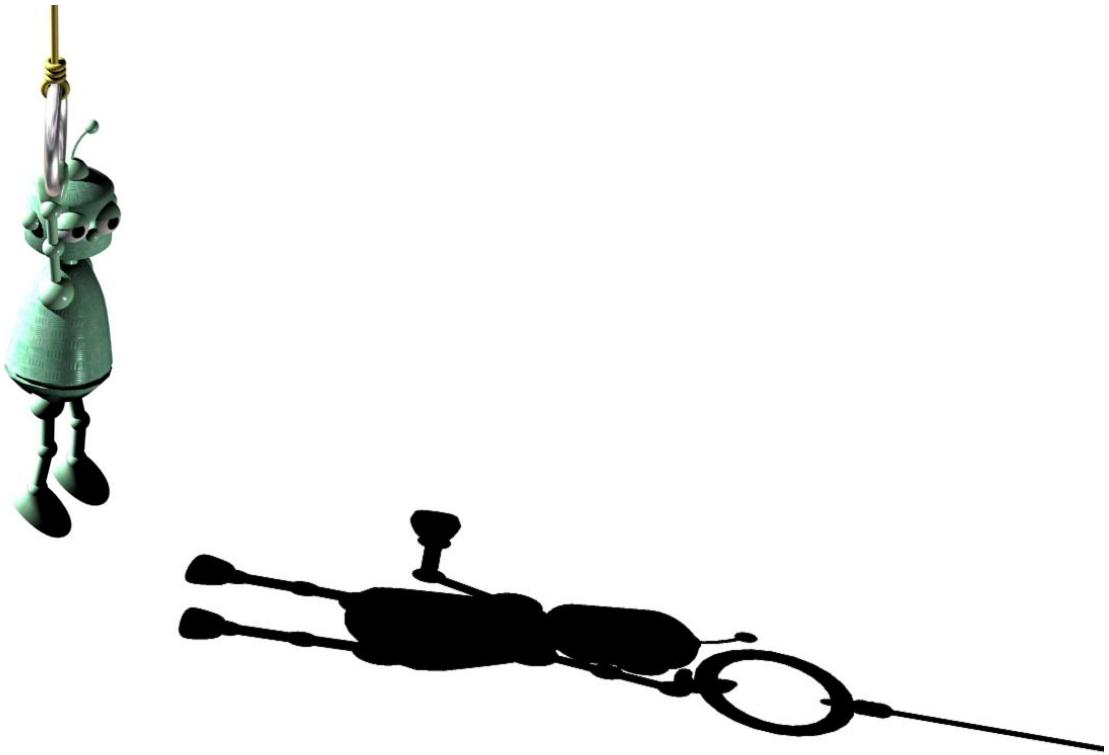




Planar Shadows & Shadow Textures

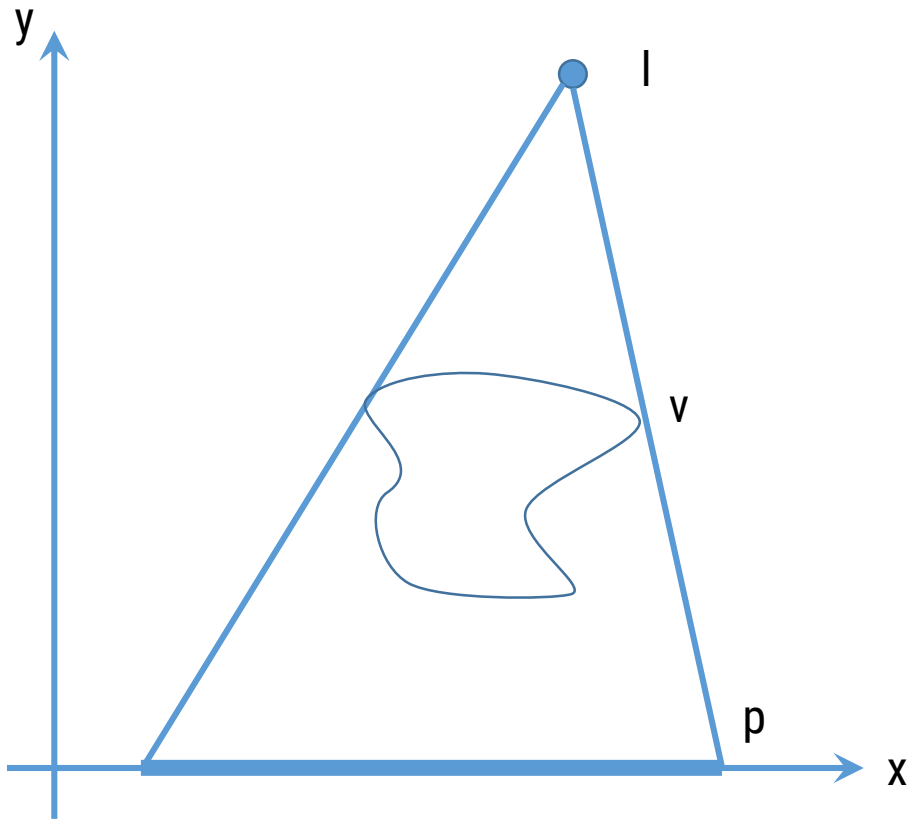
Planar shadows

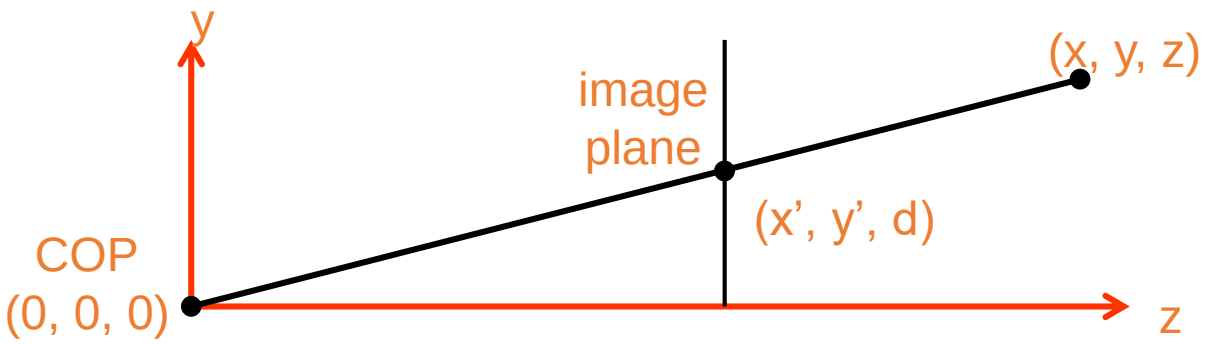
- typical situation: objects in space, one ground plane
- goal: determine shadow on the (infinite) plane



Planar shadows

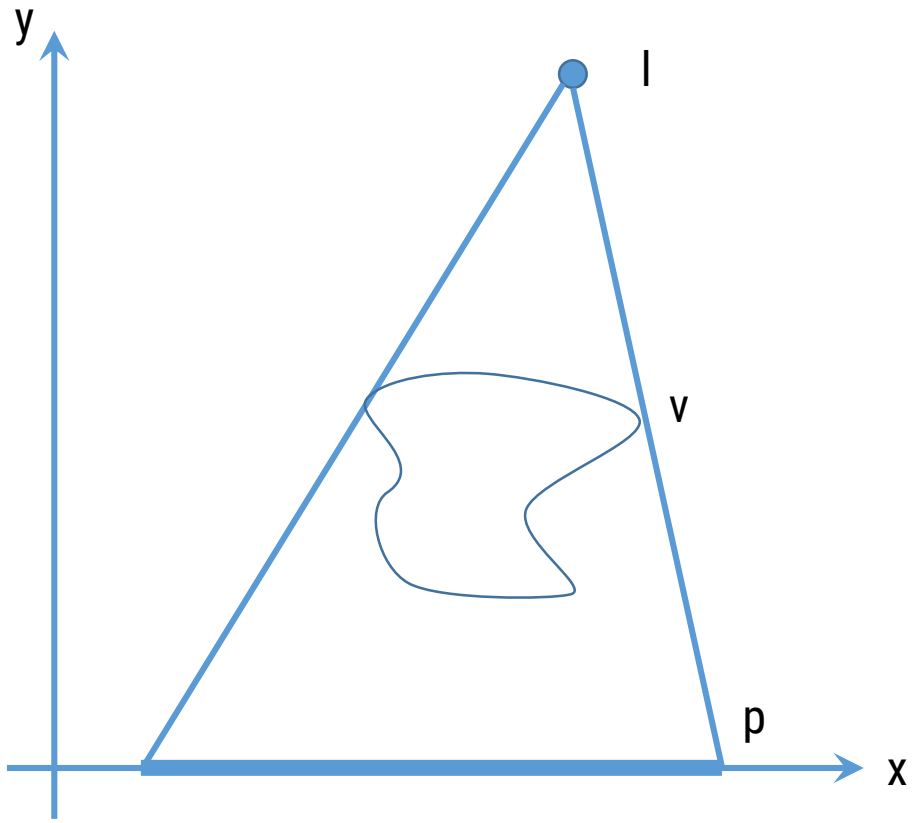
- constraints: point light source
- approach: project object (polygons, its vertices) onto (axis-aligned) plane
- same approach as in image projection of graphics pipeline





Planar shadows

- constraints: point light source
- approach: project object (polygons, its vertices) onto (axis-aligned) plane
- same approach as in image projection of graphics pipeline



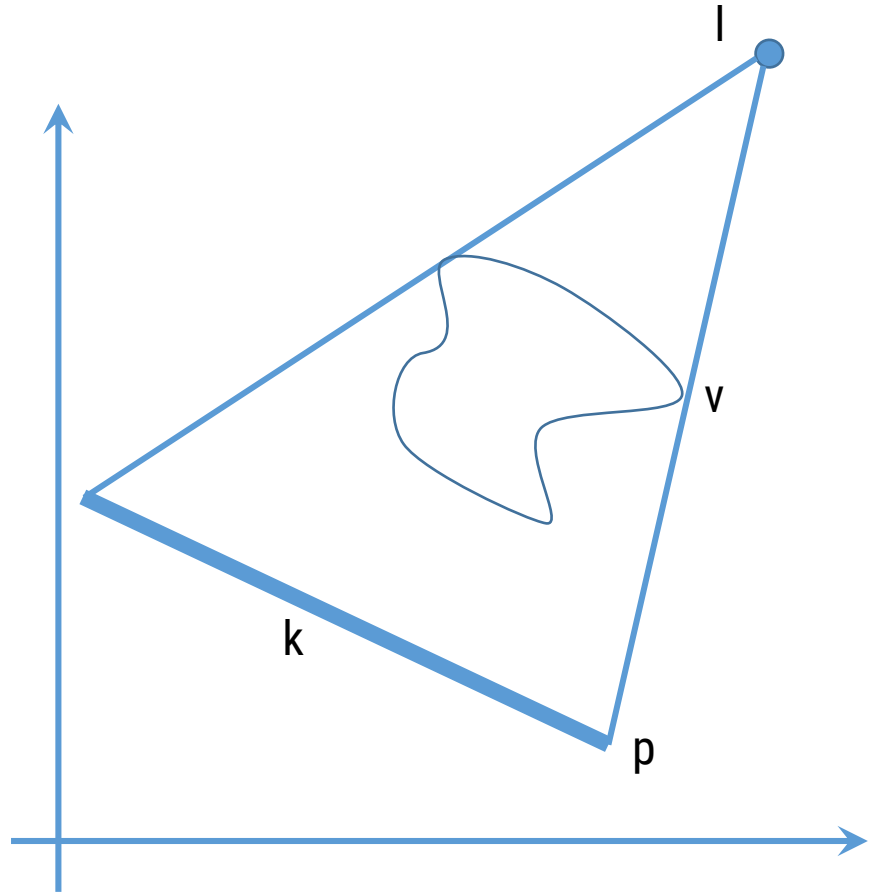
$$\frac{p_x - l_x}{v_x - l_x} = \frac{l_y}{l_y - v_y}$$

$$p_x = \frac{l_y v_x - l_x v_y}{l_y - v_y}$$

Planar shadows

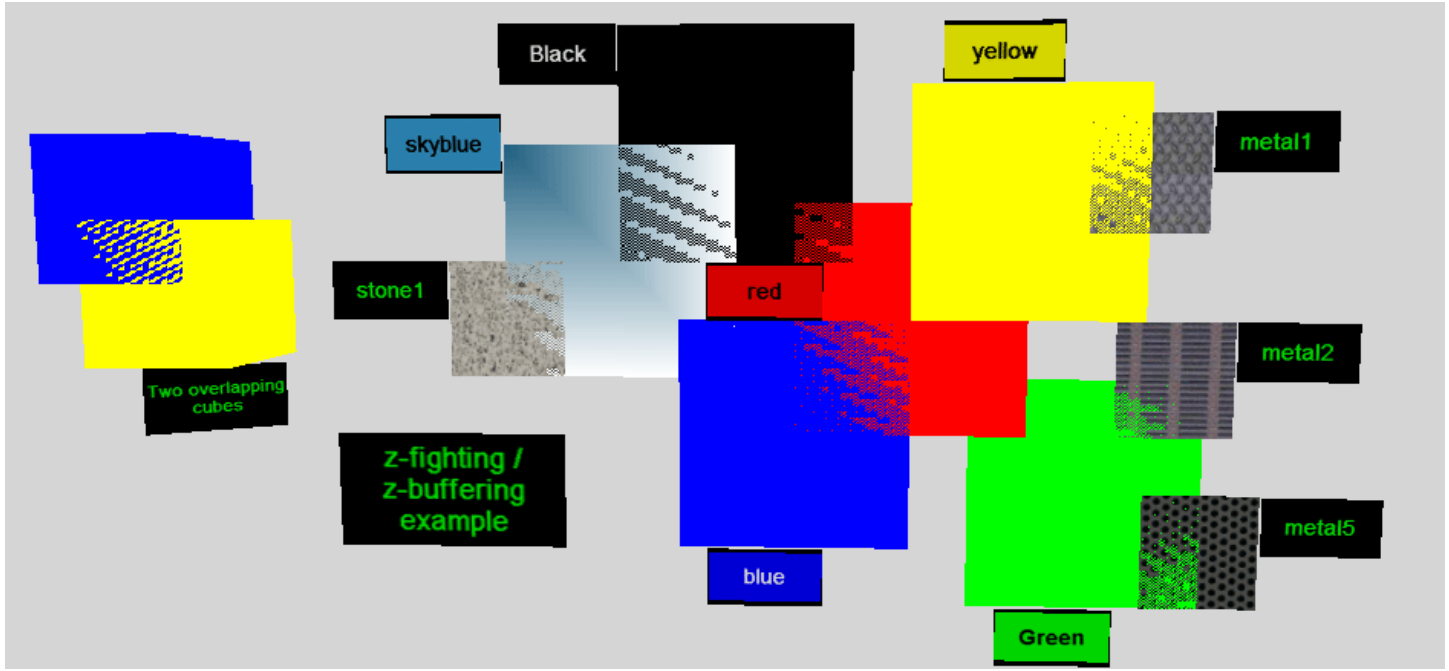
- arbitrary plane
- computation based on plane equation

$$n \cdot (p - k) = 0$$
$$n \cdot p = n \cdot k$$
$$p = l + \alpha(v - l)$$
$$n \cdot \big(l + \alpha(v - l)\big) = n \cdot k$$
$$n \cdot l + \alpha n \cdot (v - l) = n \cdot k$$
$$\alpha = \frac{n \cdot k - n \cdot l}{n \cdot (v - l)}$$
$$p = l + (v - l)\frac{n \cdot k - n \cdot l}{n \cdot (v - l)}$$



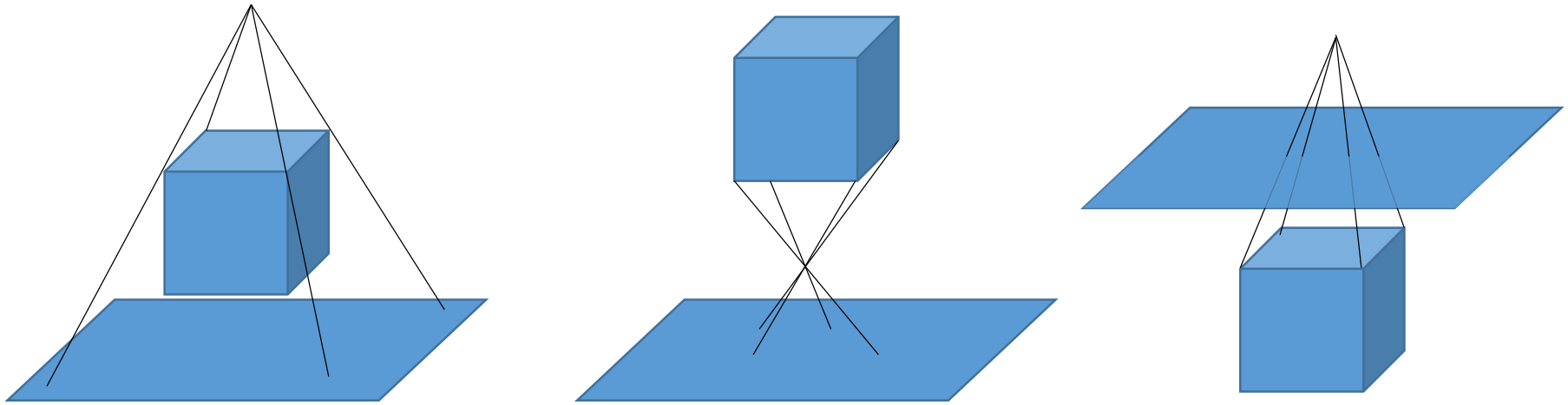
Planar shadows: Limitations

- restricted to projection onto one planar object
- shadows generated as projected triangles (polygons): results in *z*-fighting



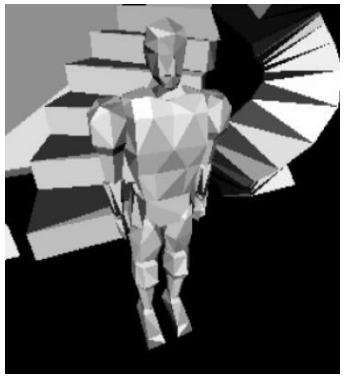
Planar shadows: Limitations

- need to be re-computed for each frame
- mathematic formulation is generic, issues with some setups:



Shadow textures

- rendering-based method to avoid some of these issues
- approach: two-pass rendering
  - first identify the occluder
  - light-source-rendering: produces grayscale texture of occlude
  - texture-map shadow texture on background, render from camera



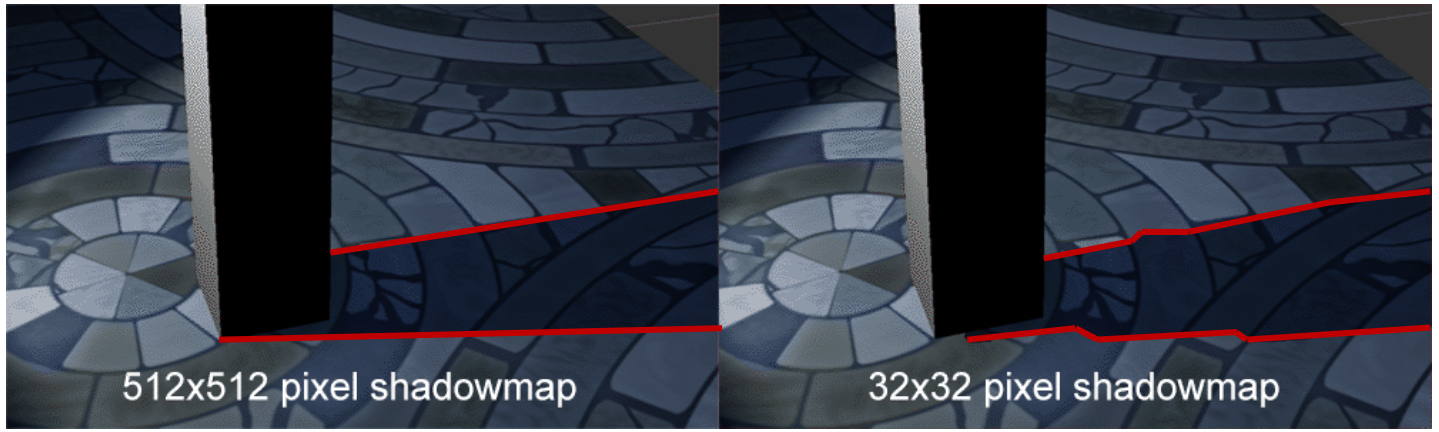
view from light source

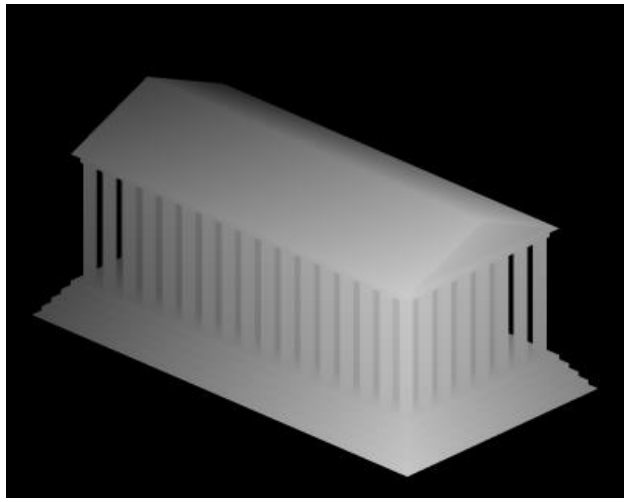
shadow texture

view from camera

Shadow textures

- pros
  - works for large receivers, simple to realize
  - no analytic projection of triangles, no *z*-fighting
- cons
  - occluders and receivers need to be explicitly identified
  - texture resolution may lead to sampling artifacts

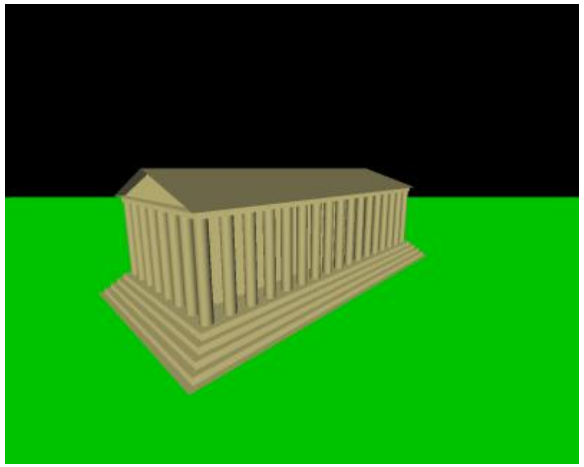


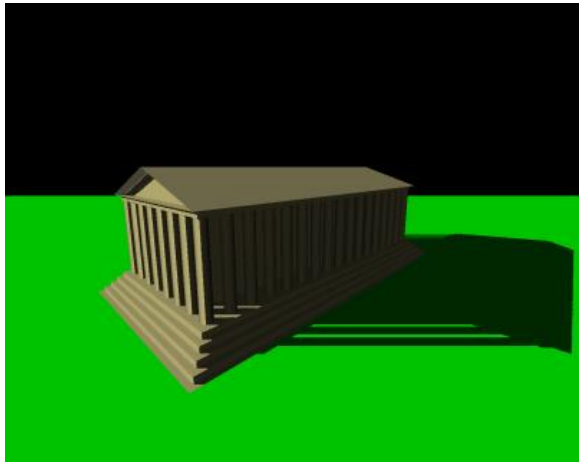


Shadow Mapping

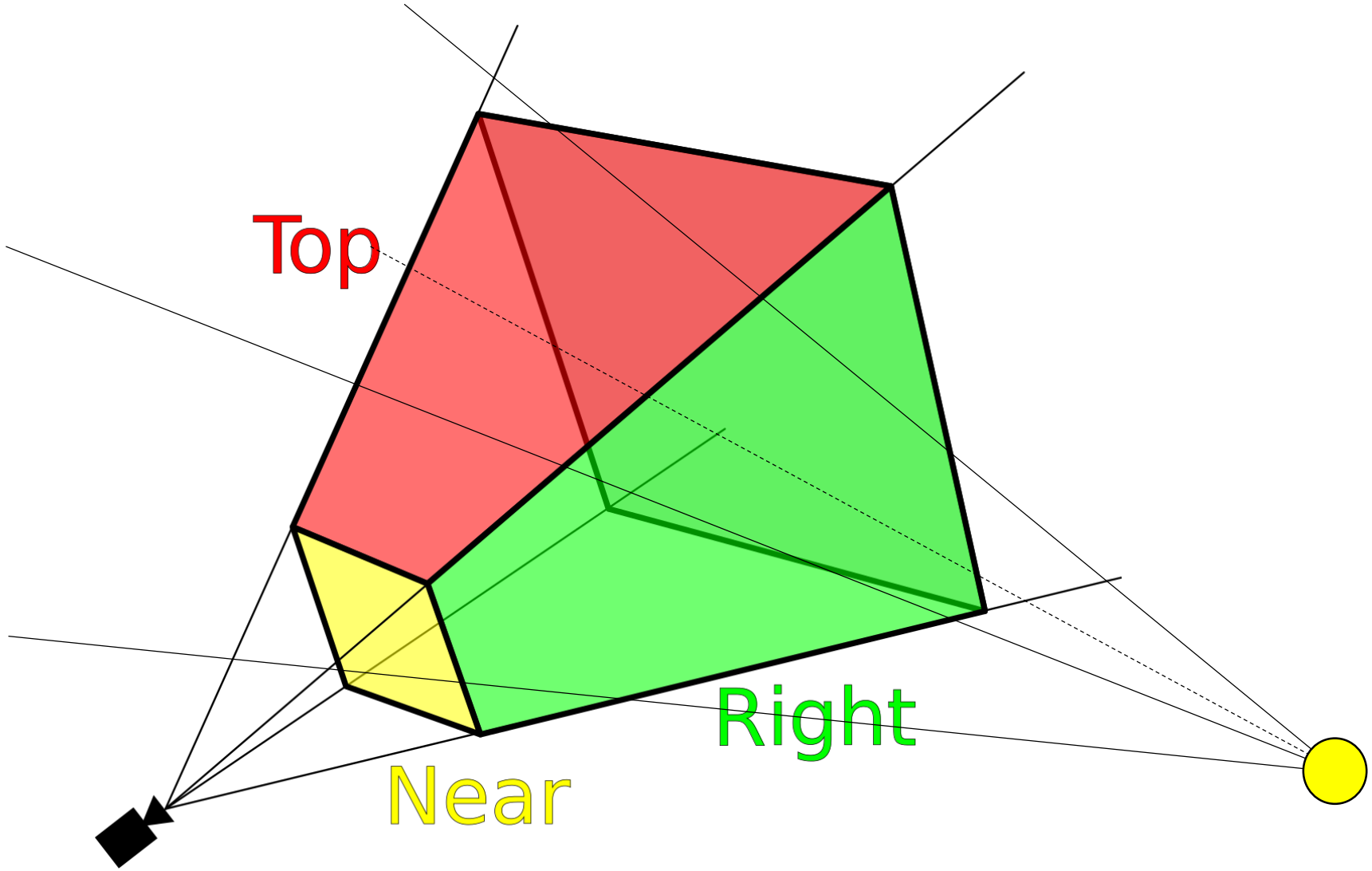
Shadow mapping

- improvement on shadow texture approach to realistic, complex scenes
- idea: remove need to identify occluder/ receiver by recording the distance from the light source
  $\rightarrow$ use of *z*-buffer
  $\rightarrow$ simple shadows: points are considered to be fully in shadow or fully illuminated
- same two-pass rendering as before



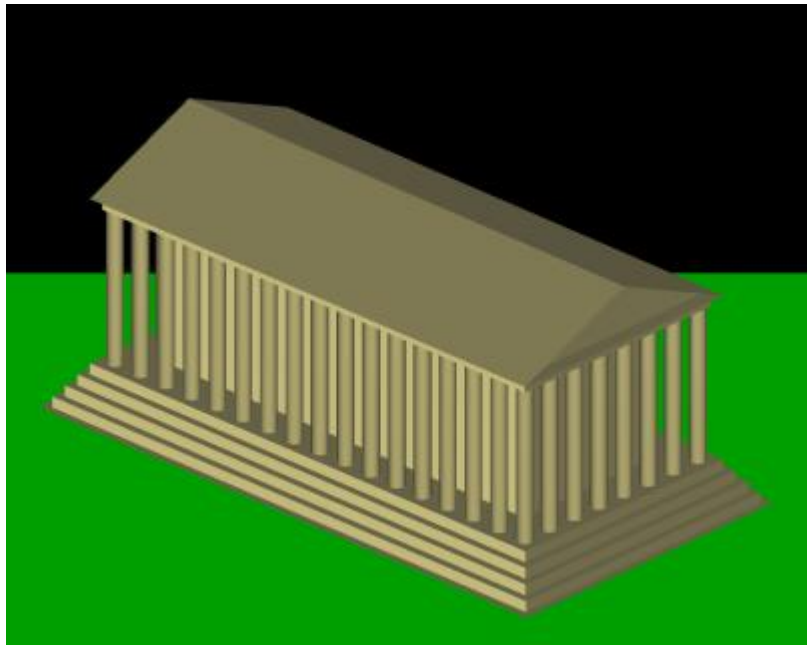


View frustum and light frustum

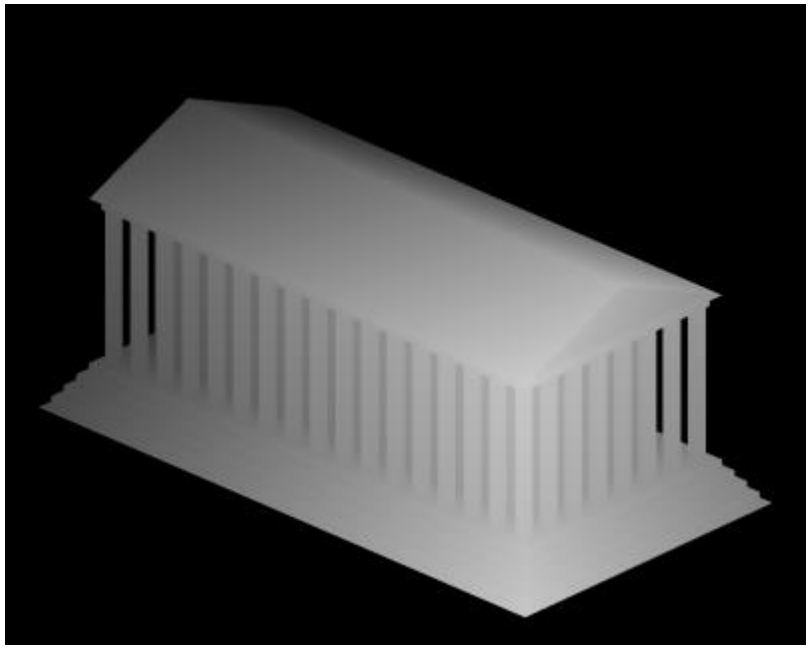


Shadow mapping

- first render pass: generate *z*-buffer from the light source
- *z*-buffer records **distance of objects from light source**



scene viewed from the light source

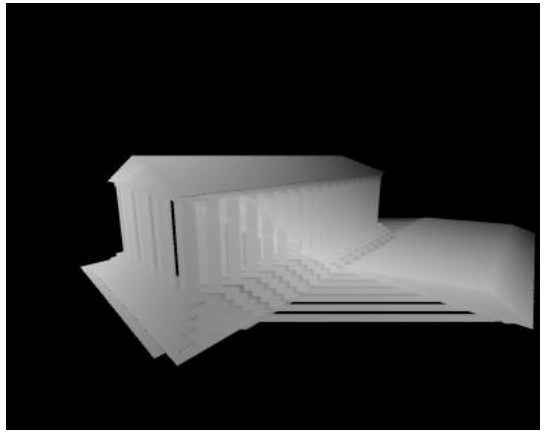


z-buffer from the light source

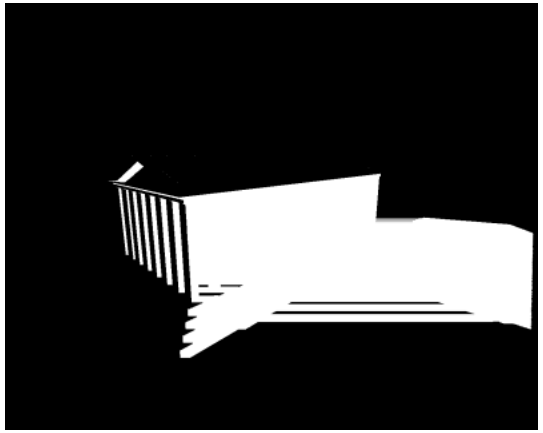
Shadow mapping

- second pass: “normal” rendering from camera with additional shadow map depth test
  - project vertex to camera coordinates (for normal shading)
  - project vertex to light source coordinates (model-view and projection matrices from 1st rendering pass)
  - compare light source coordinates with shadow map depth buffer

visualization of depth map projected onto the scene

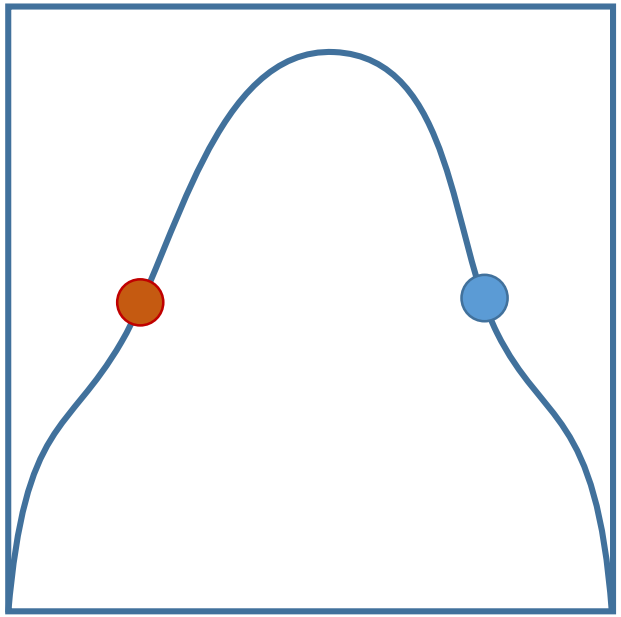


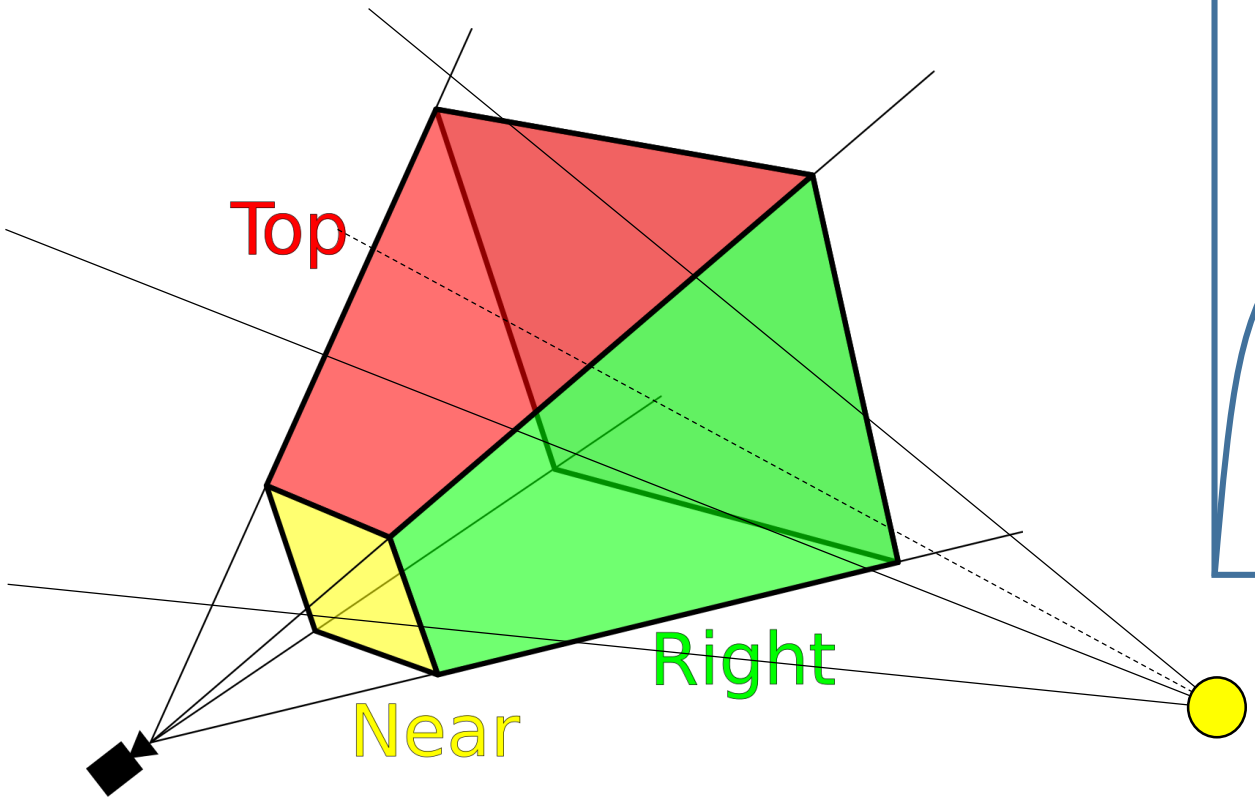
shadow map test failures shown in white



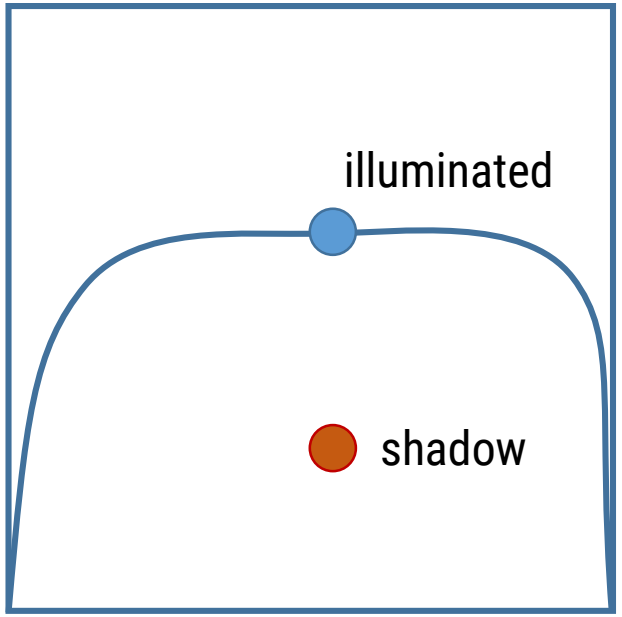
Shadow mapping

depth map from camera



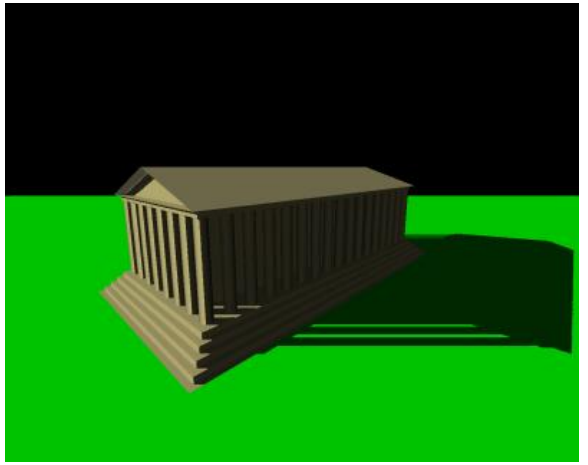


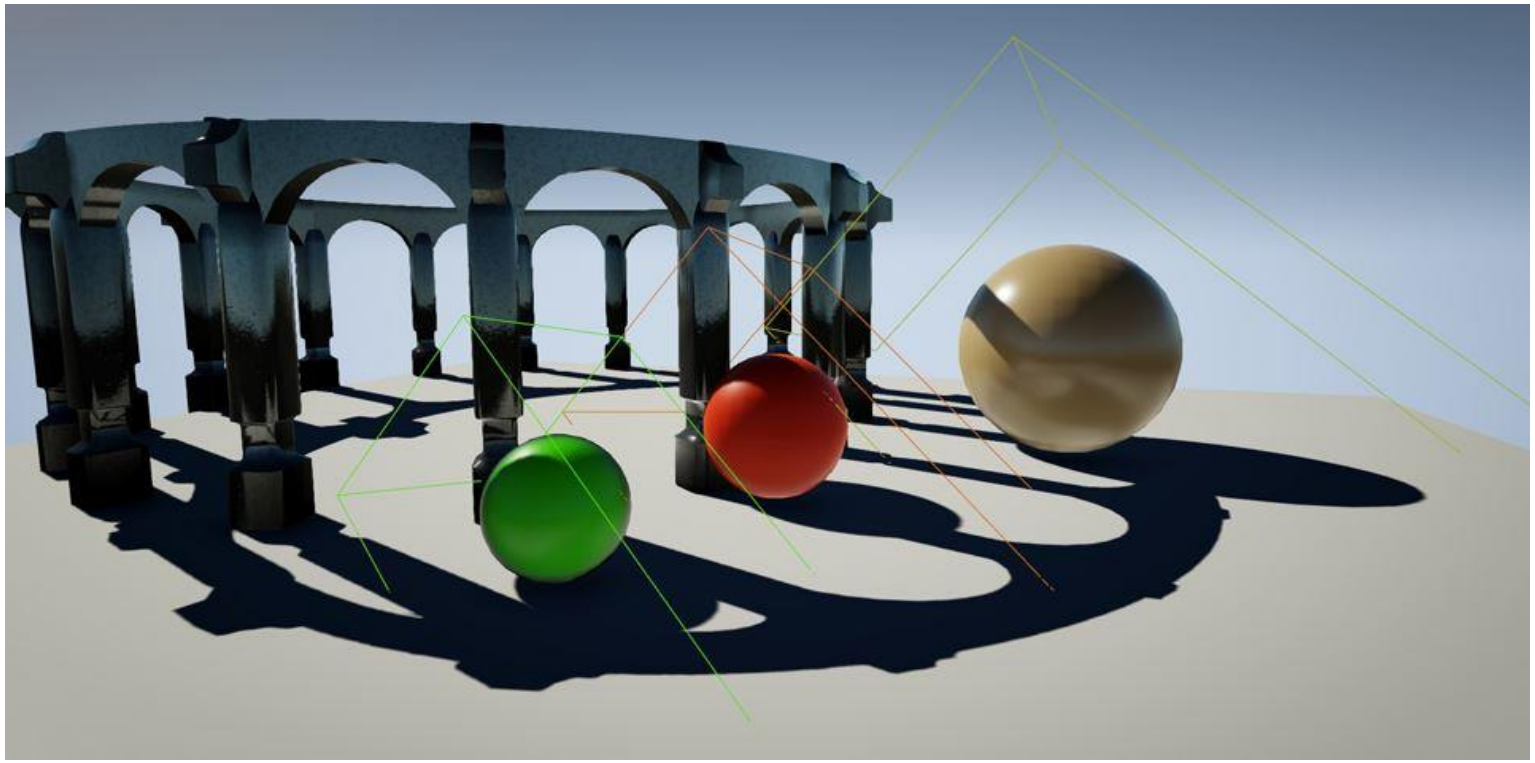
depth map from light source



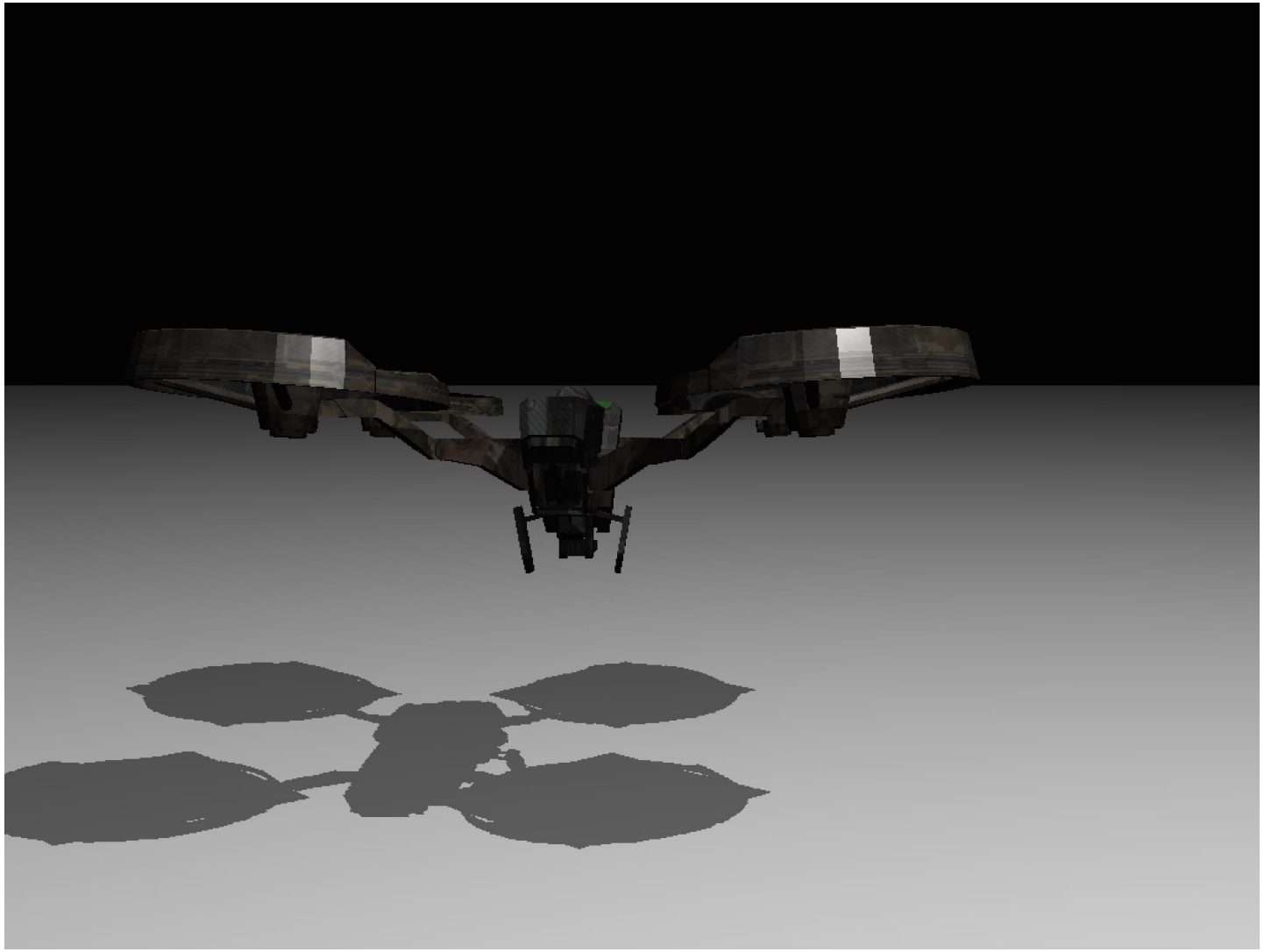
Shadow mapping: advantages

- no prior geometry analysis needed anymore
- works for any geometry



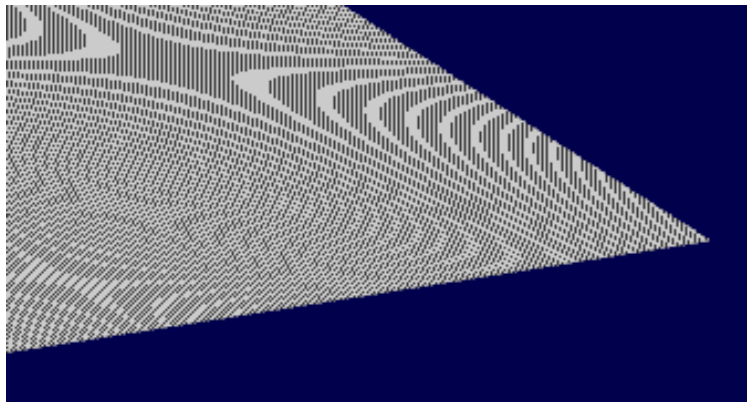


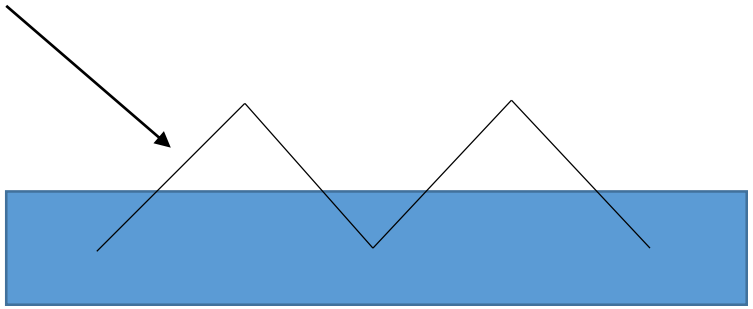
Shadow mapping: result



Shadow mapping: disadvantages

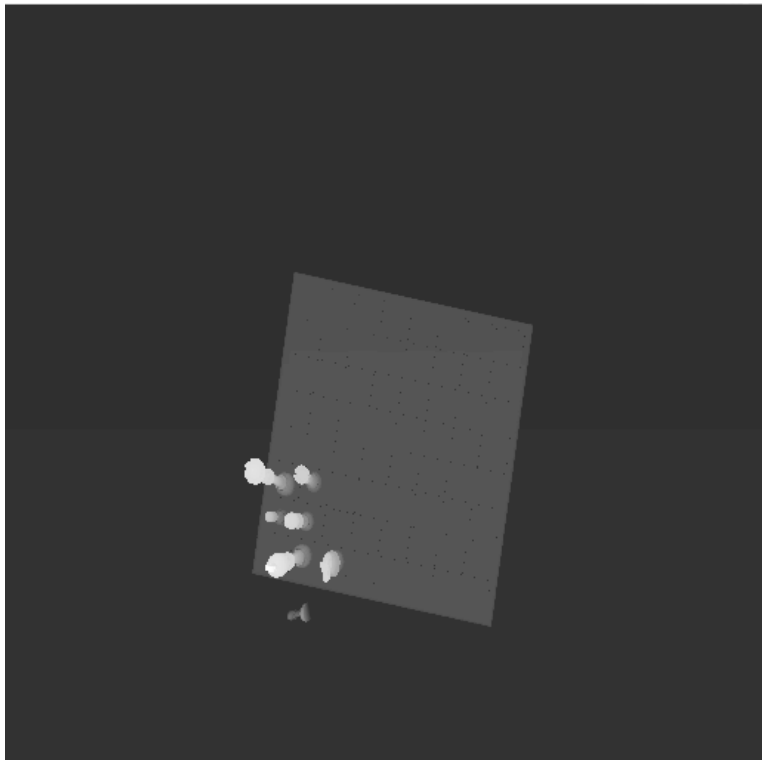
- works only for point light sources
- can exhibit shadow acne
- sampling artifacts





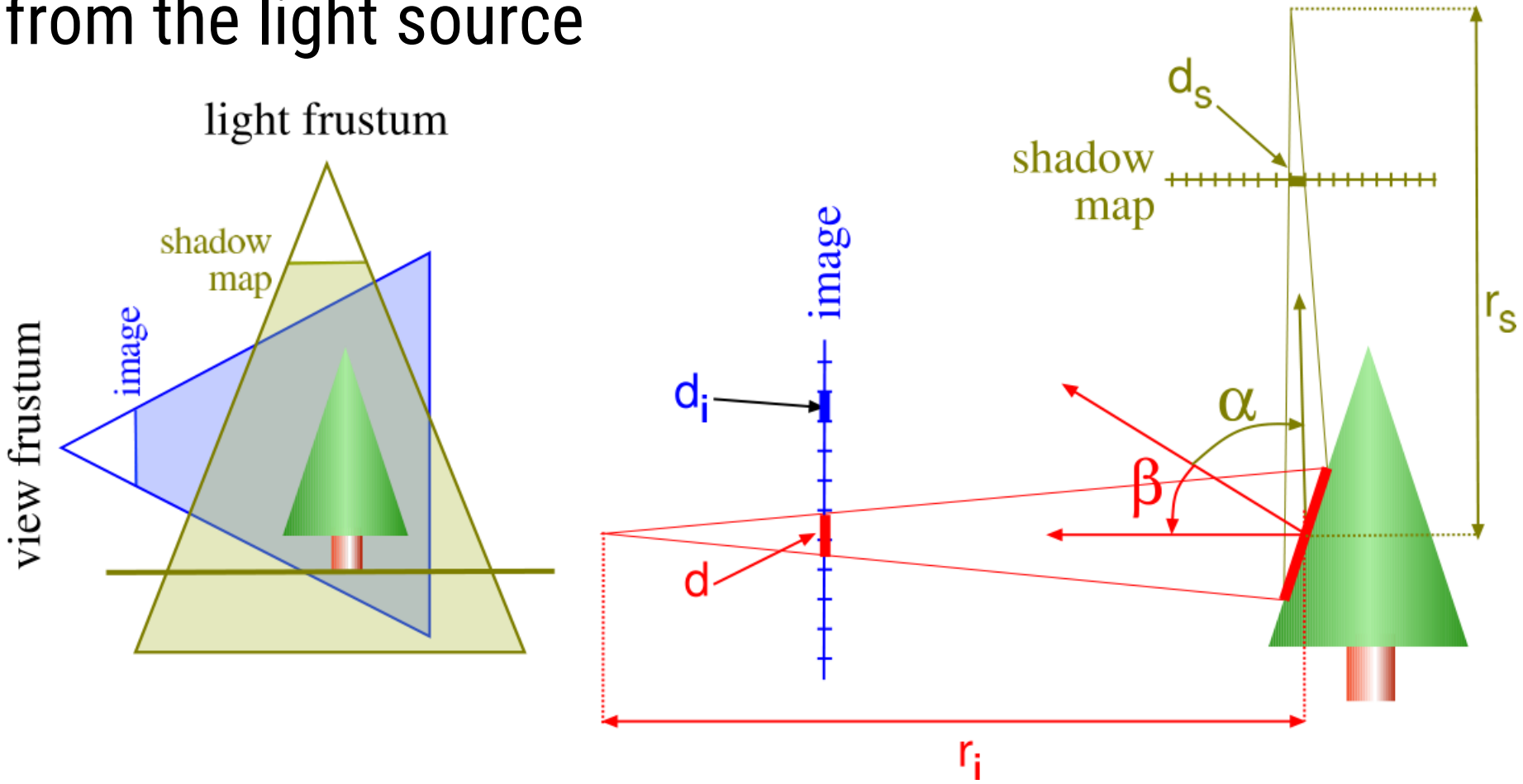
Perspective shadow maps

- quality of shadows depends on distance of objects from the light source



Perspective shadow maps

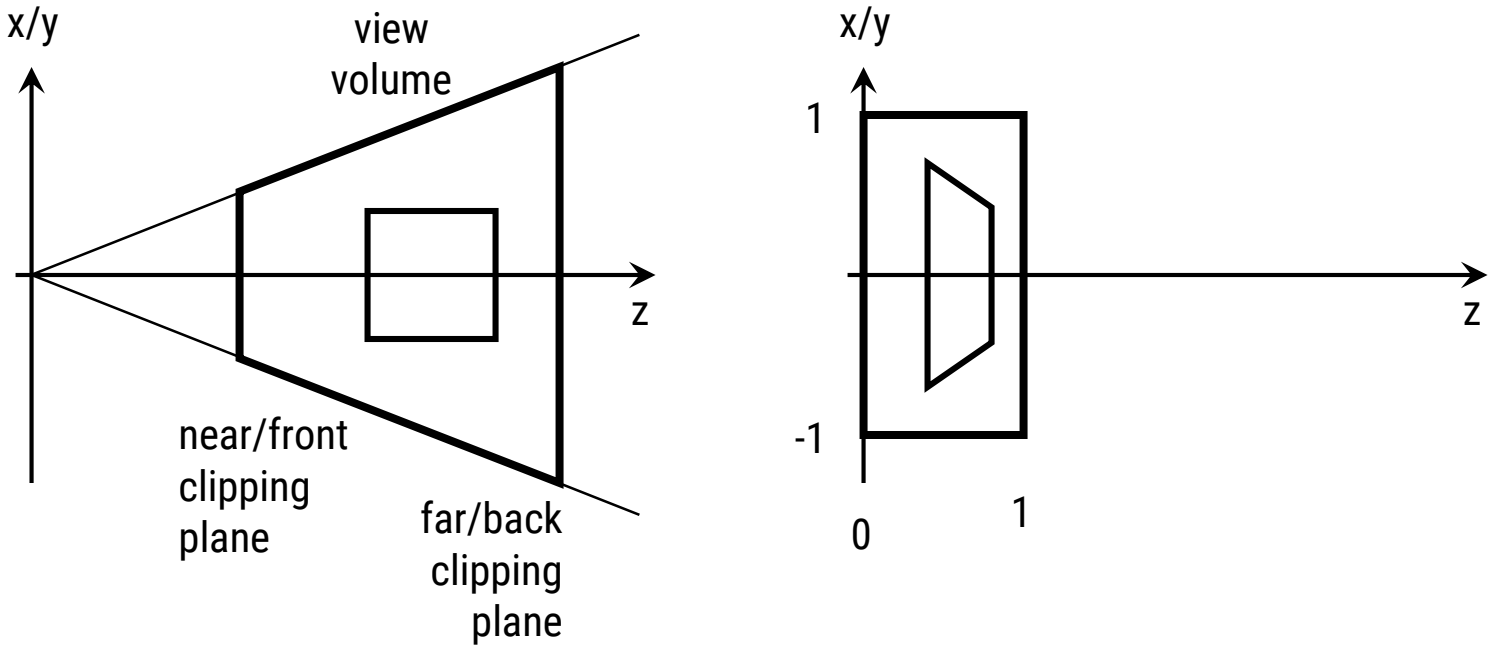
- quality of shadows depends on distance of objects from the light source



far-away light sources mean that only few pixels of the light source's z-buffer are used to compute the shadow of close objects

Perspective shadow maps

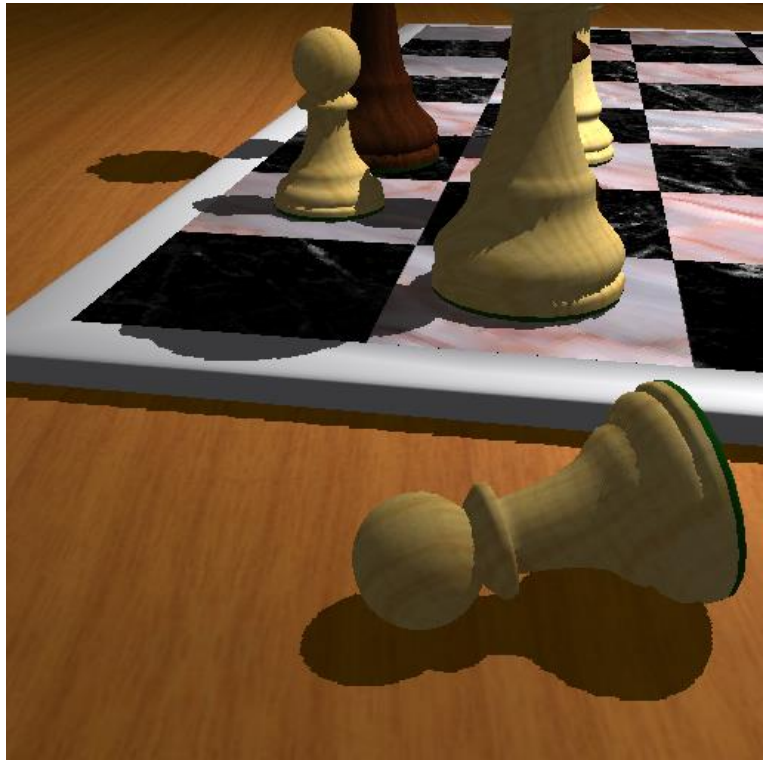
- solution: compute shadow map in post-perspective space (from the perspectively projected position of the light source)

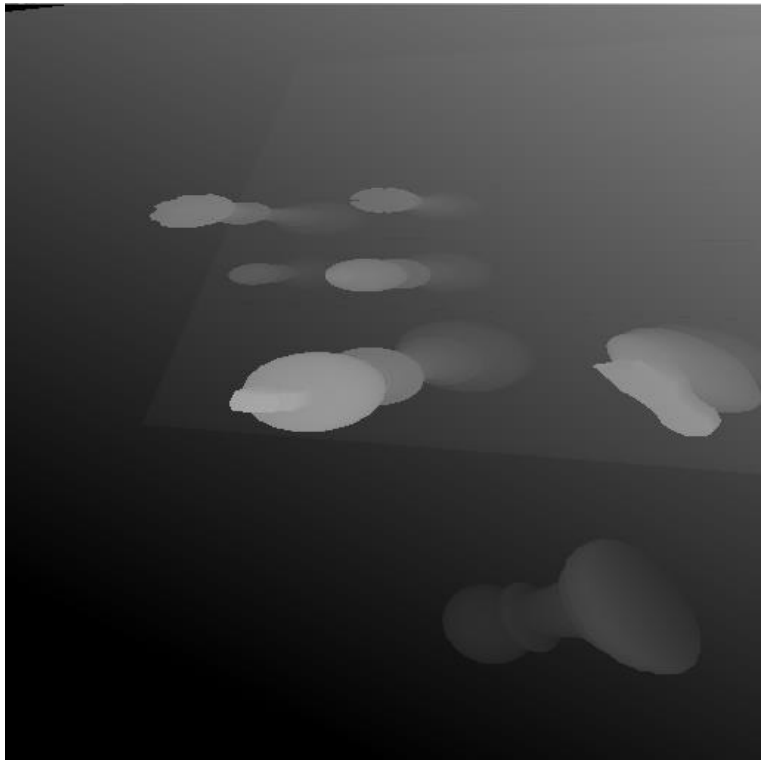


close objects thus become larger and are better covered in shadow map

Perspective shadow maps

- solution: compute shadow map in post-perspective space (from the perspectively projected position of the light source)

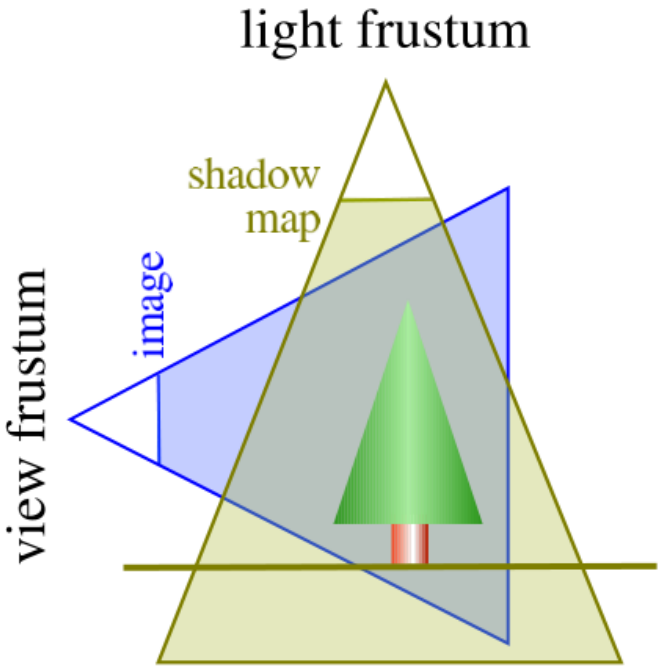


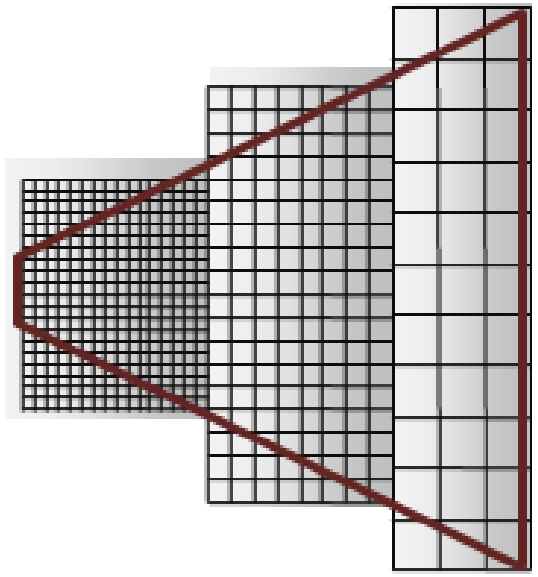


close objects thus become larger and are better covered in shadow map

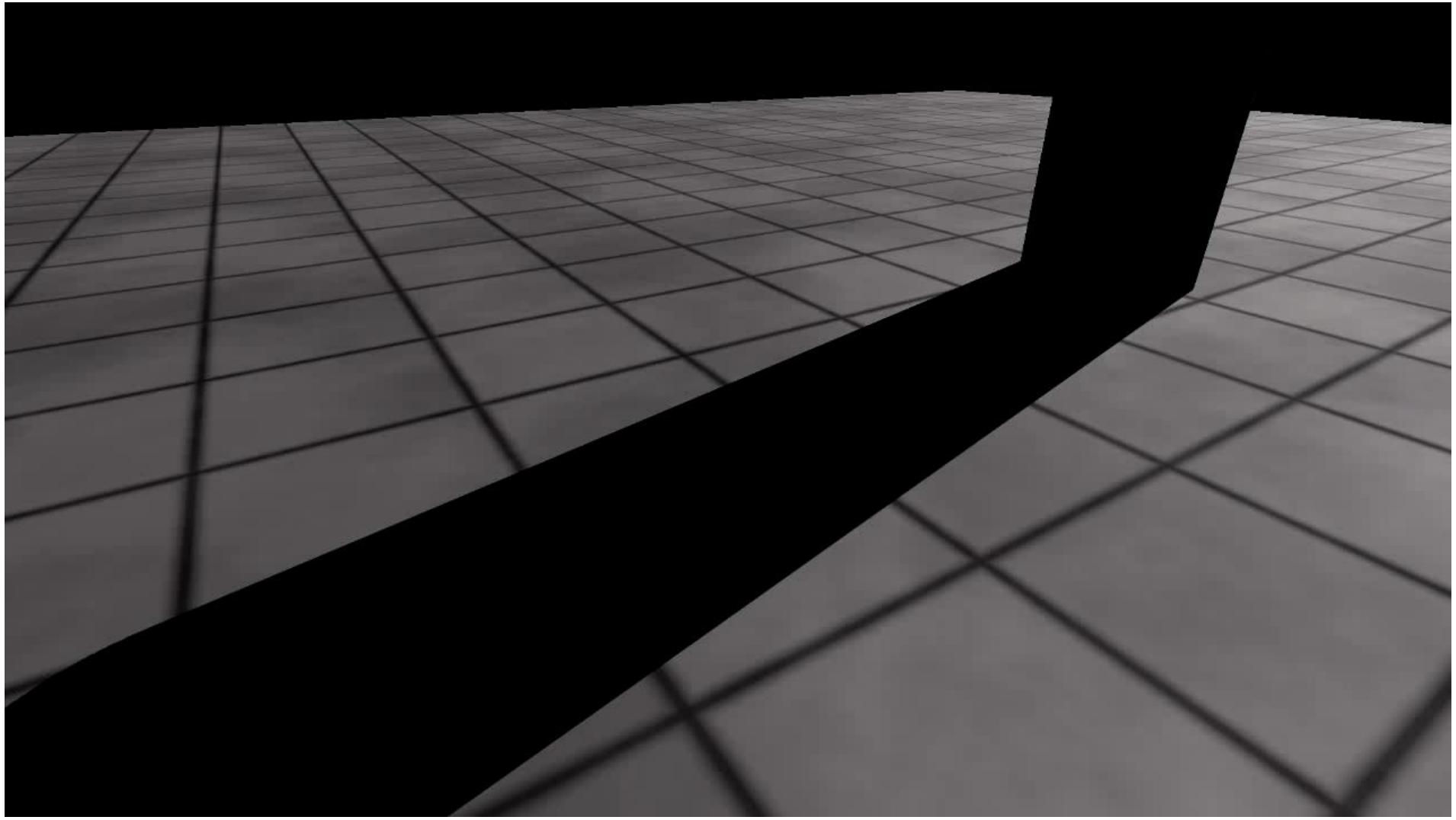
Cascaded shadow maps

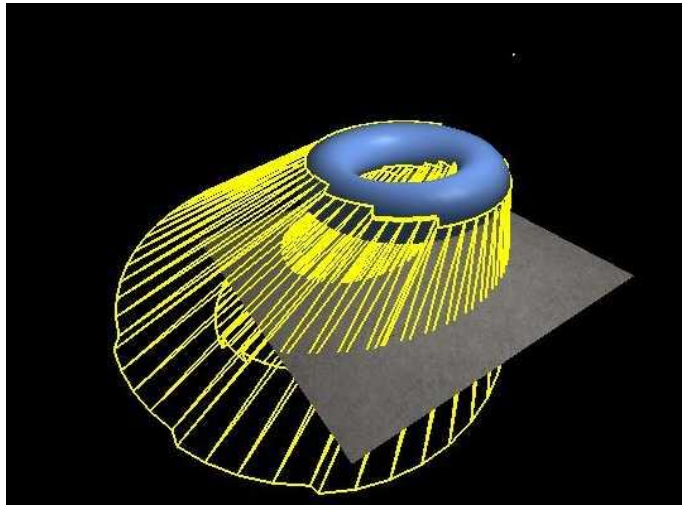
- problem with aliasing depending on distance from the camera
- compute different shadow maps for different distances from the viewer





Cascaded Shadow Map

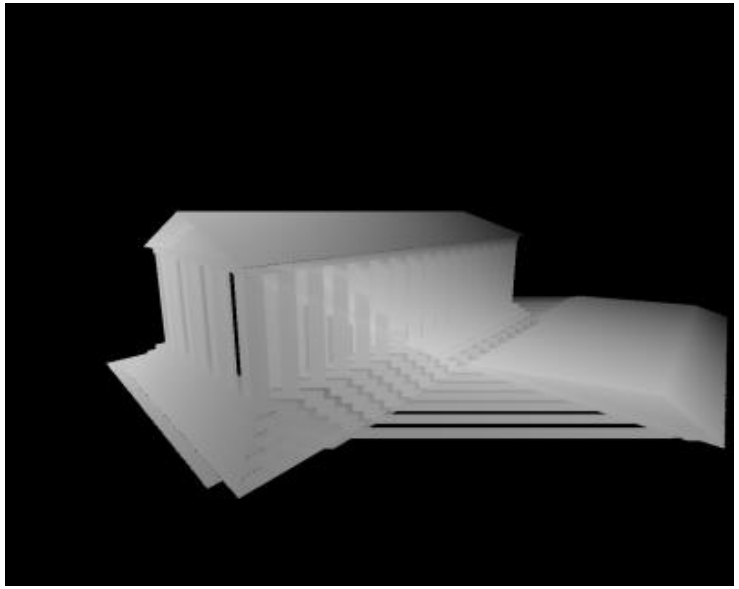




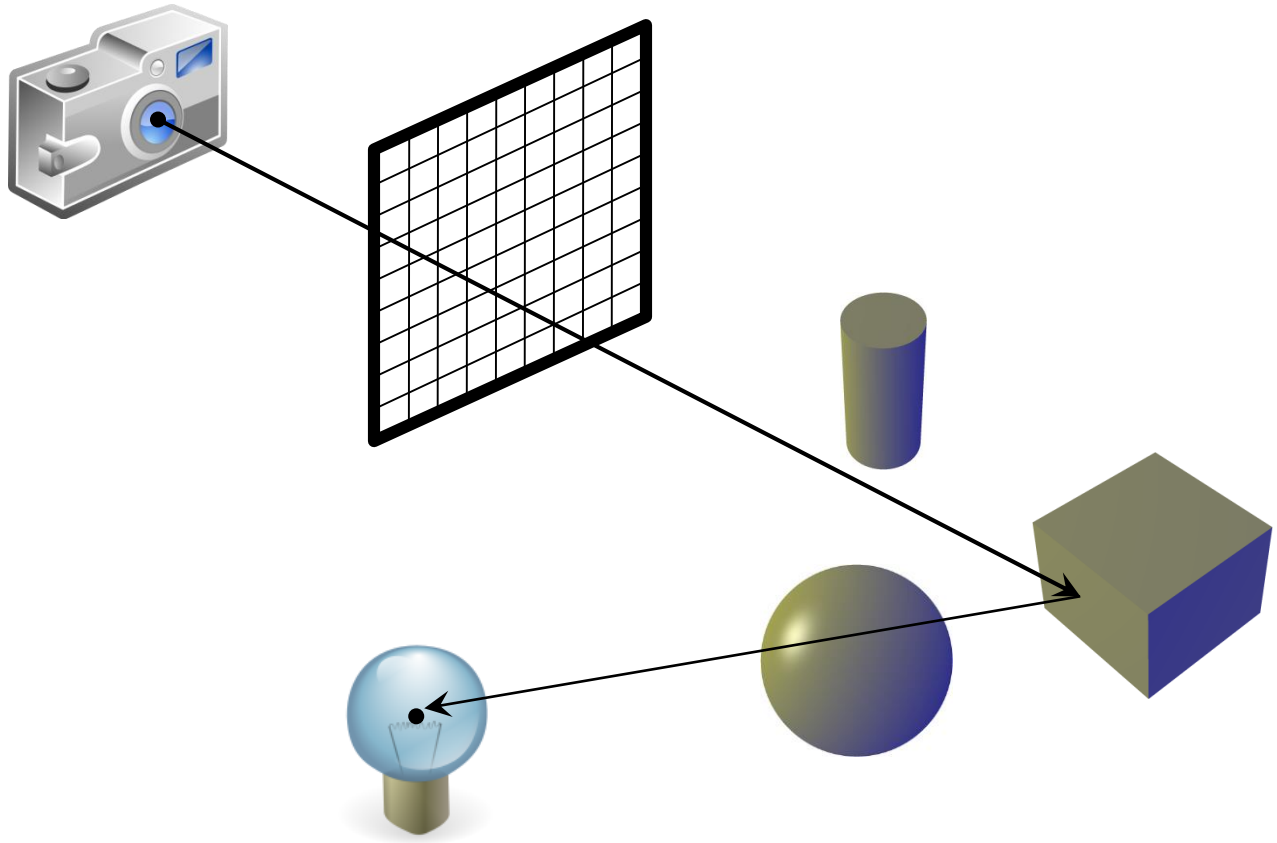
Shadow Volumes

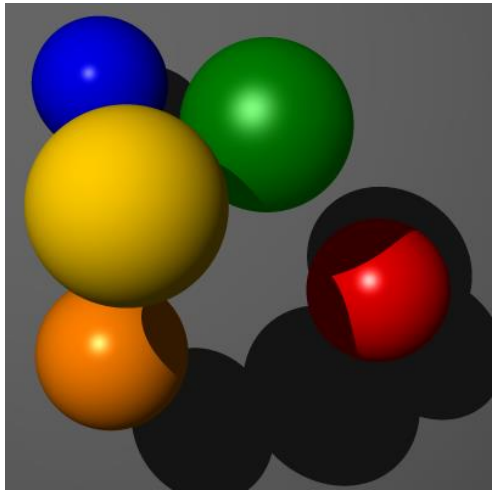
Shadow volumes

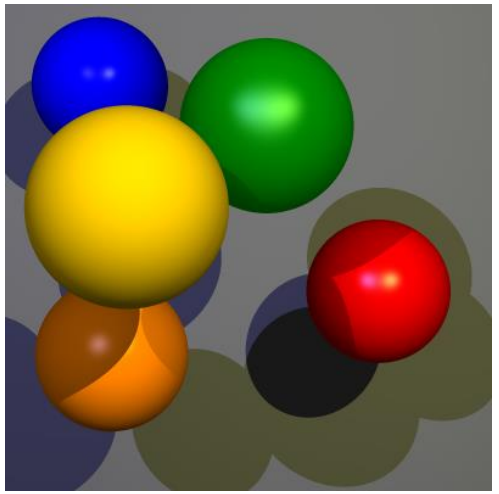
- problem with shadow maps:
  - quality depends on pixel resolution of the shadow map *z*-buffer
  - higher quality need multiple render passes (cascaded shadow maps) or generally more memory for shadow map
  - several light sources need several render passes
- take idea of shadow map *z*-buffer and extend to an analytic process
  - shadow map represents a volume of space that is in the shadow



Inspiration: Shadows in raytracing

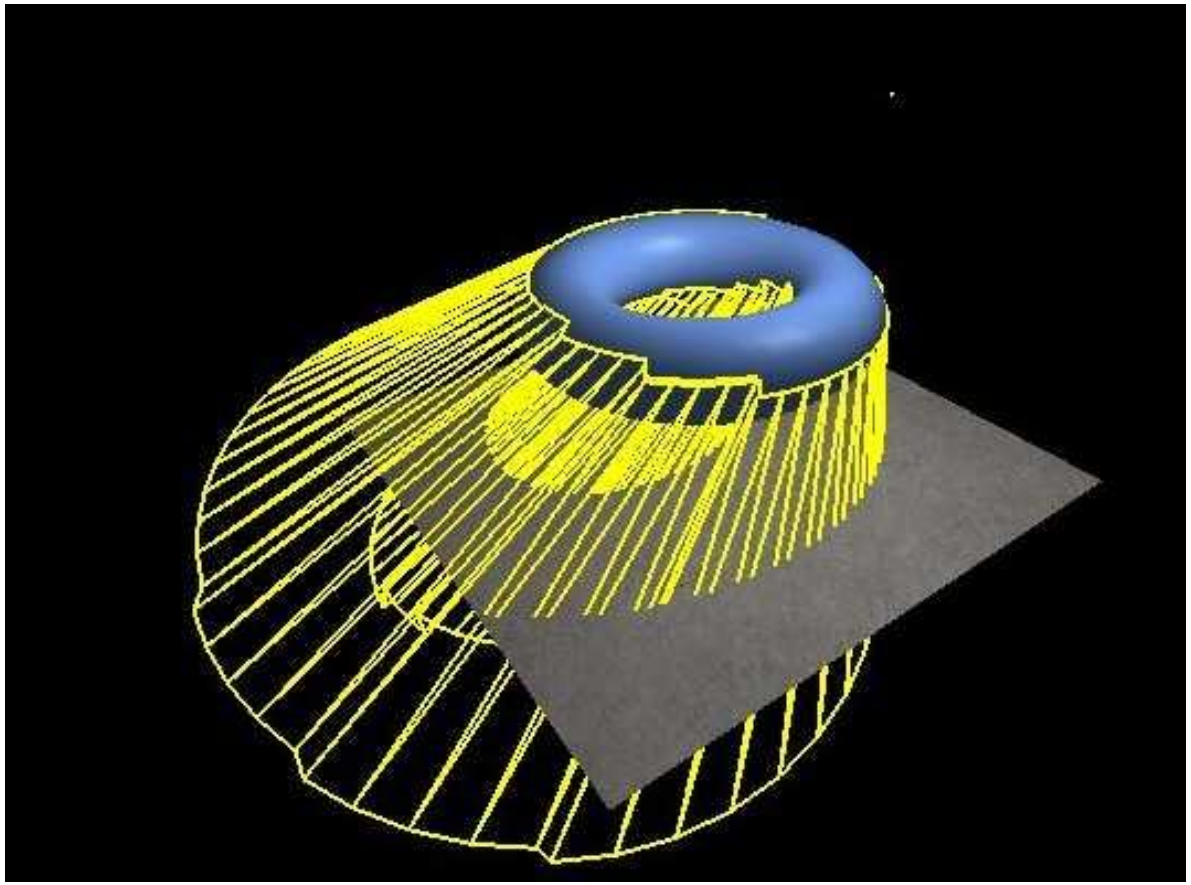




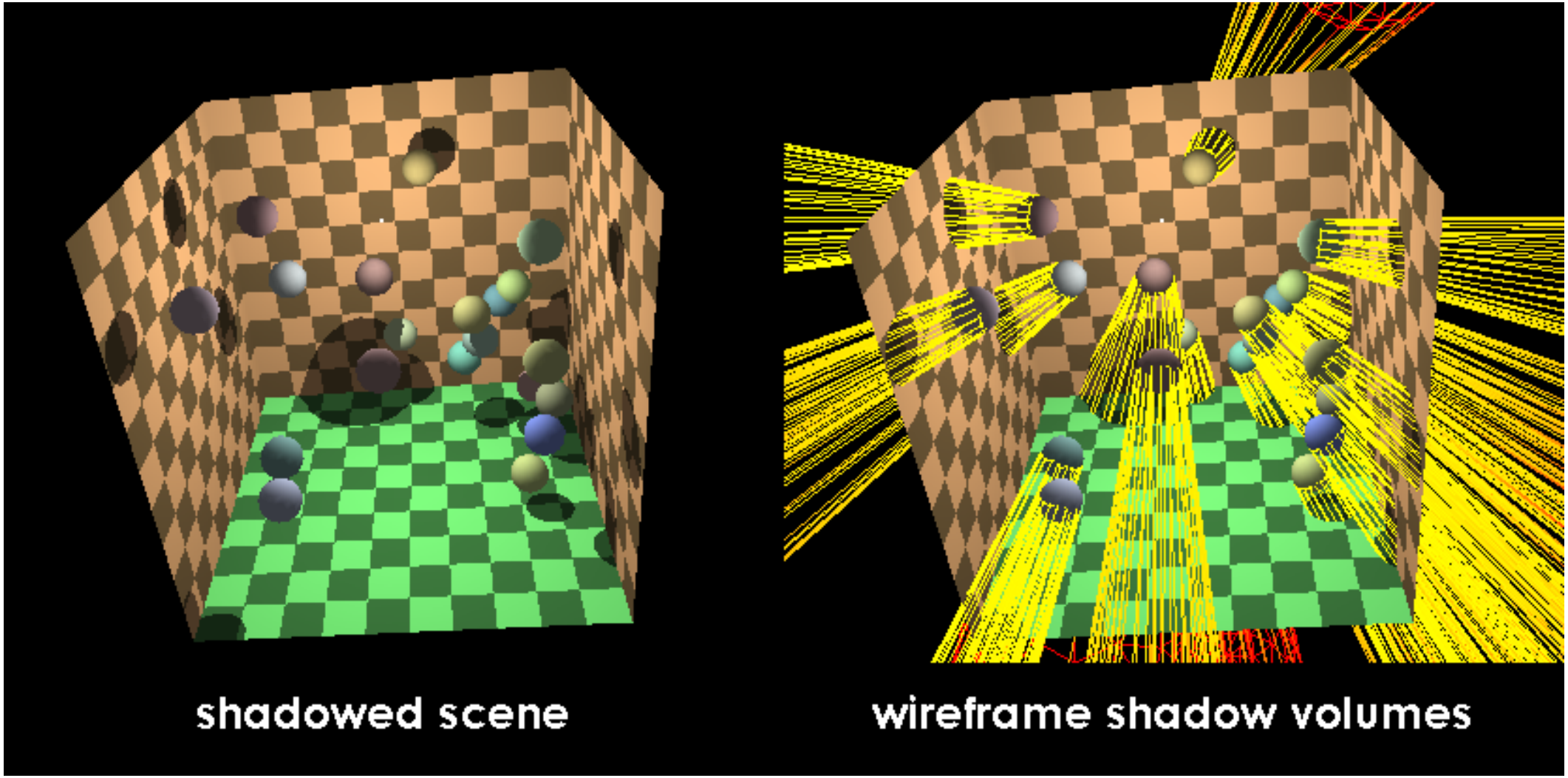


Shadow volumes

- idea: extend rays from light source to object silhouette and beyond
- connect to (analytic) polygonal shapes that enclosed the area that is in the shade

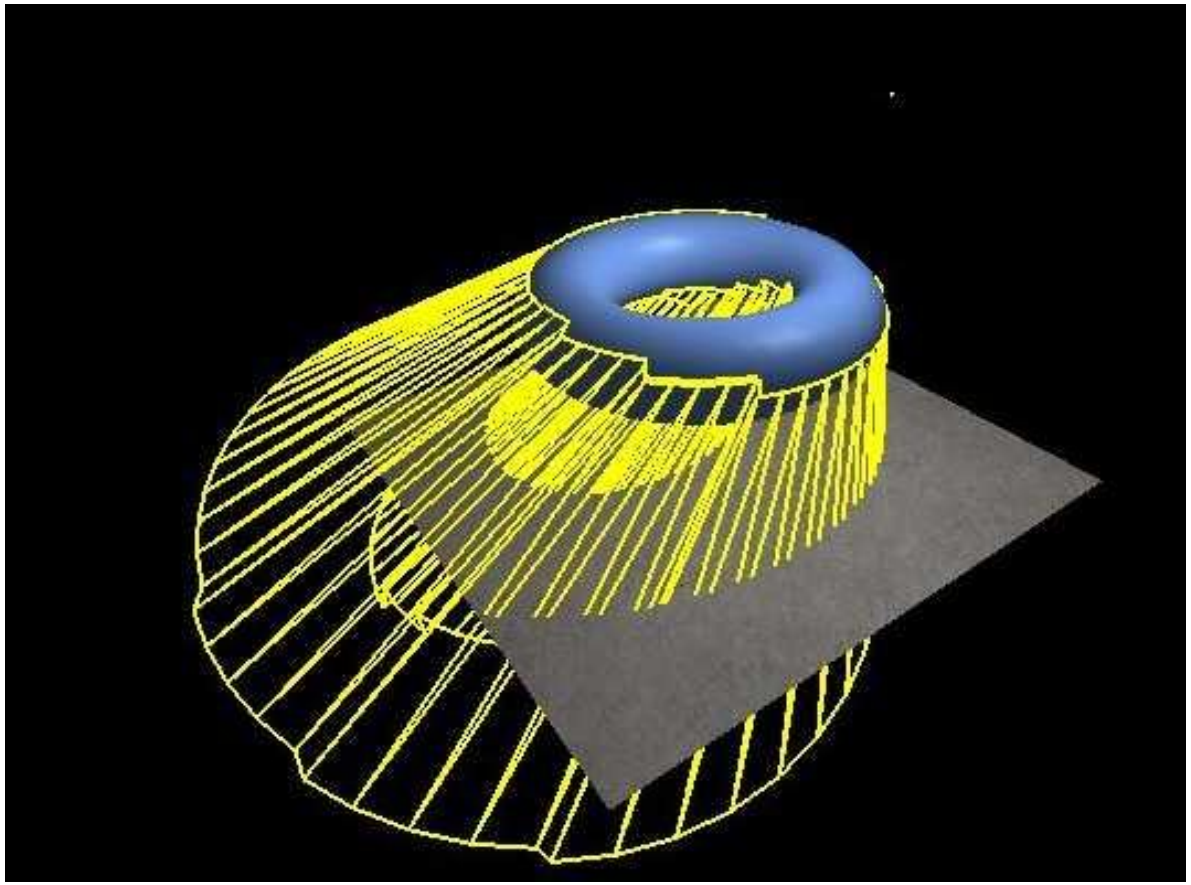


Shadow volumes



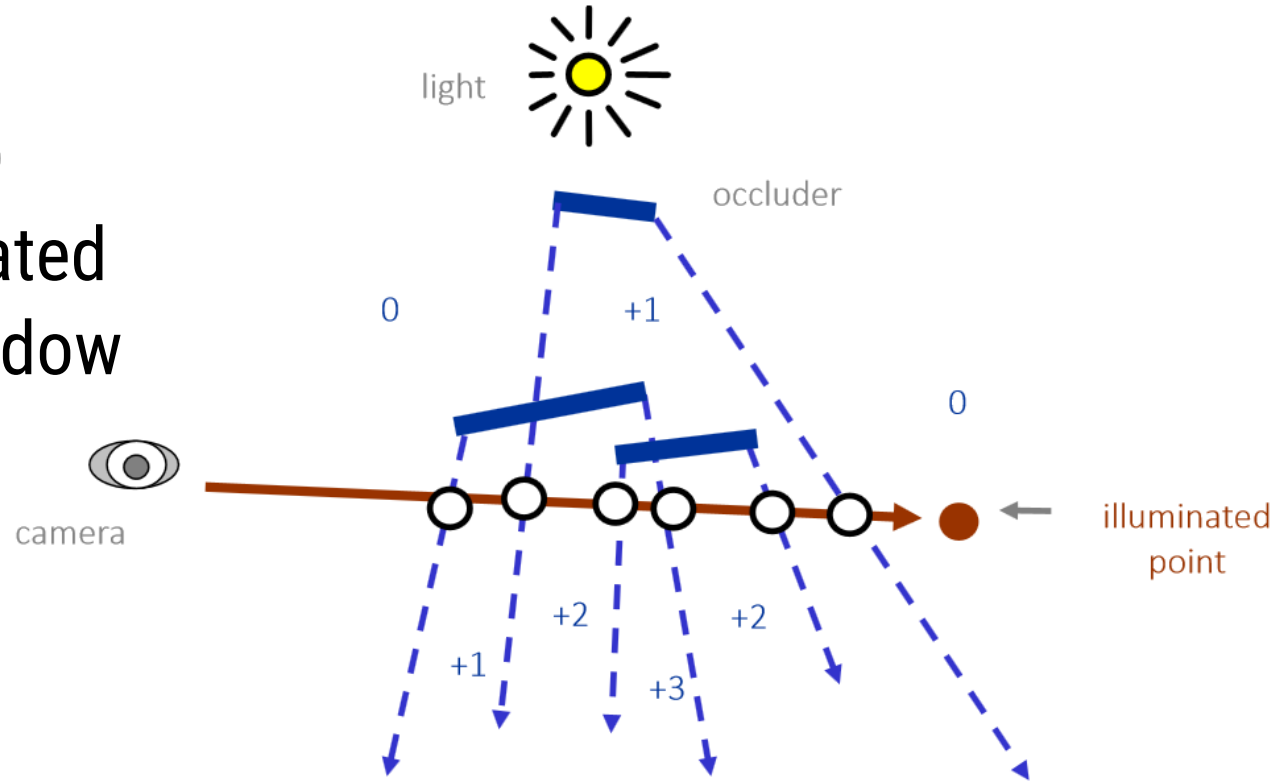
Shadow volumes: construction

- use geometry shader:
  - extrude shadow volume along the direction of light
  - each illuminated (front-facing to the light source) polygon is extruded
  - result: shadow **geometry**
  - use stream output to store result



Shadow volumes: At render time

- "count" all intersections of a ray through the pixel with shadow volumes (entering = +1; leaving = -1)
  - count = 0 → point is illuminated
  - count ≠ 0 → point in the shadow

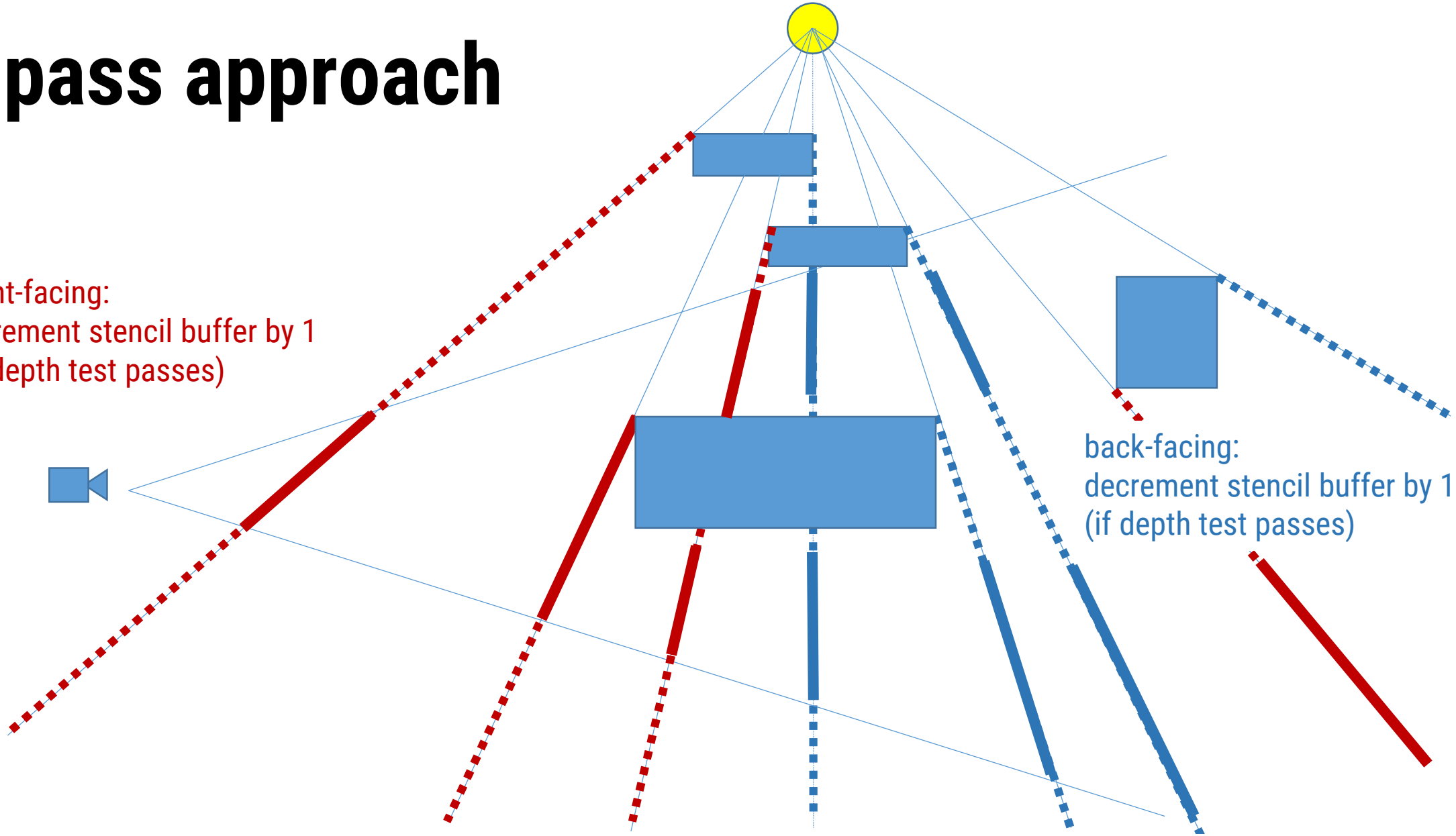


Shadow volumes: At render time

- counting using the stencil buffer; ***z*-pass approach**
  1. 1st render to initialize *z*-buffer (ambient light only, flat shading)
  2. turn off *z*-buffer/frame buffer, turn on stencil buffer;
     stencil buffer counts intersections by **rendering shadow volumes**
     - 1st stencil pass with shadow volume: with **back face culling** enabled, font faces of shadow volume **increment** the stencil buffer if the depth test **passes**
     - 2nd stencil pass with shadow volume: with **front face culling** enabled, back faces of the shadow volume **decrement** the stencil buffer if the depth test **passes**

  → counts the number of **shadow volumes in front of the objects**

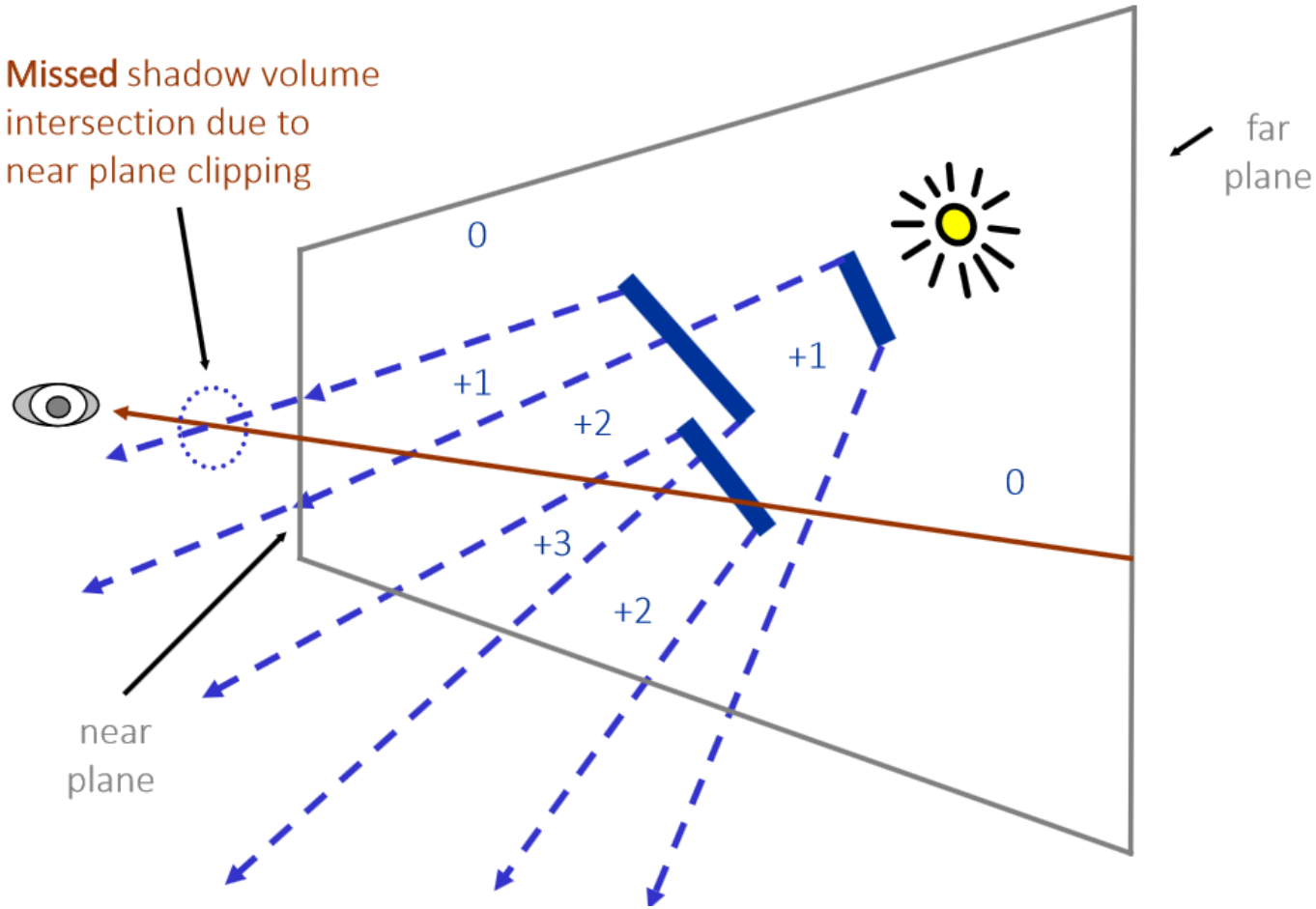
z-pass approach



result: stencil buffer has the number of shadow volumes in front of an object, thus we know what is in the shadow (stencil > 0)

Problems with *z*-pass approach

- eye in shadow volume: count number of volumes that enclose the eye
- shadow volumes that intersect near plane: can miss intersections

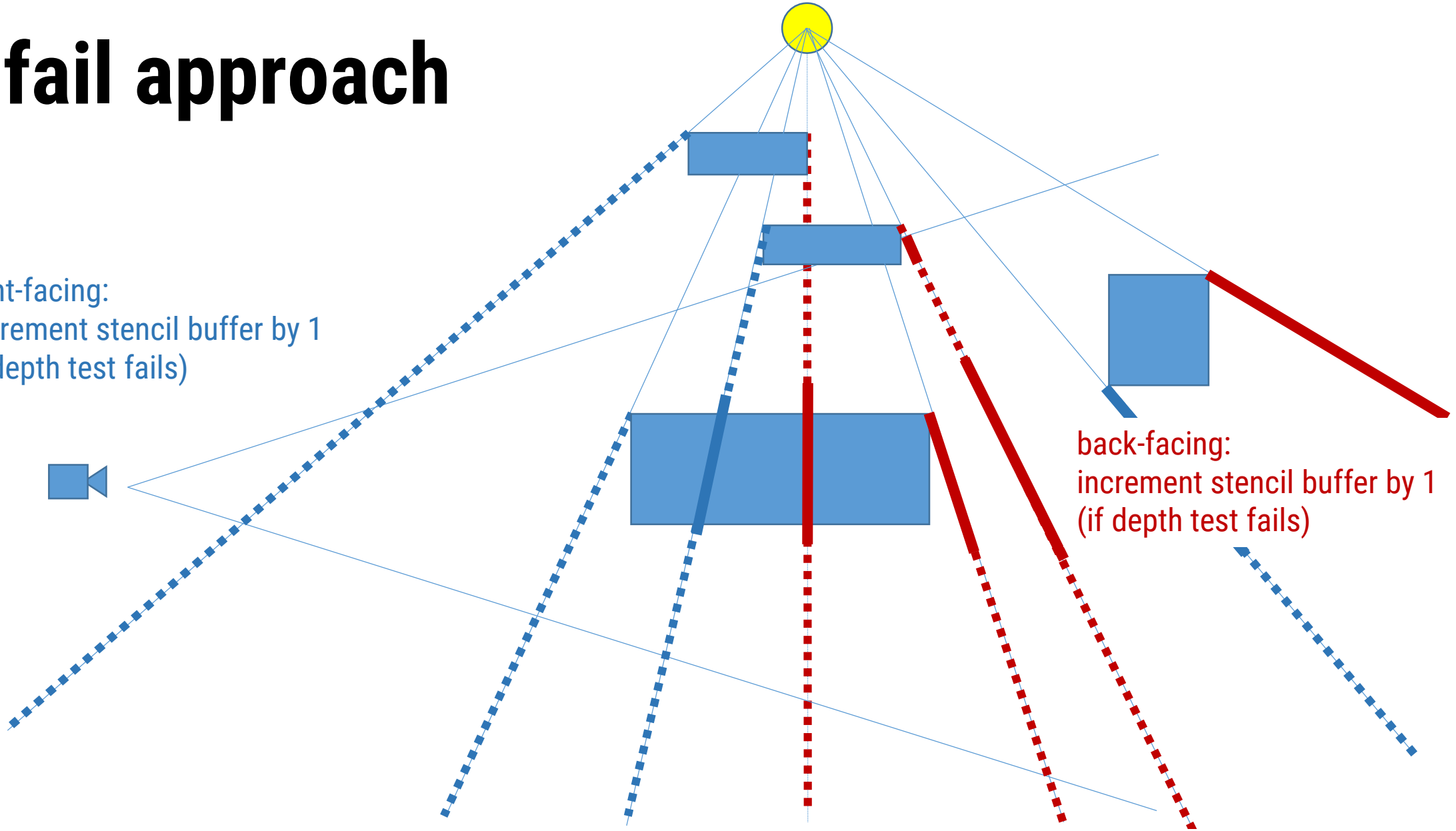


Solution: *z*-fail approach “Carmack's Reverse”

1. 1st render to initialize *z*-buffer (ambient light only, flat shading)
2. turn off *z*-buffer/frame buffer, turn on stencil buffer; stencil buffer counts intersections by **rendering shadow volumes**
   - 1st stencil pass with shadow volume: with **front face culling** enabled, font faces of shadow volume **increment** the stencil buffer if the depth test **fails**
   - 2nd stencil pass with shadow volume: with **back face culling** enabled, back faces of the shadow volume **decrement** the stencil buffer if the depth test **fails**

   → counts the number of **shadow volumes behind the objects**

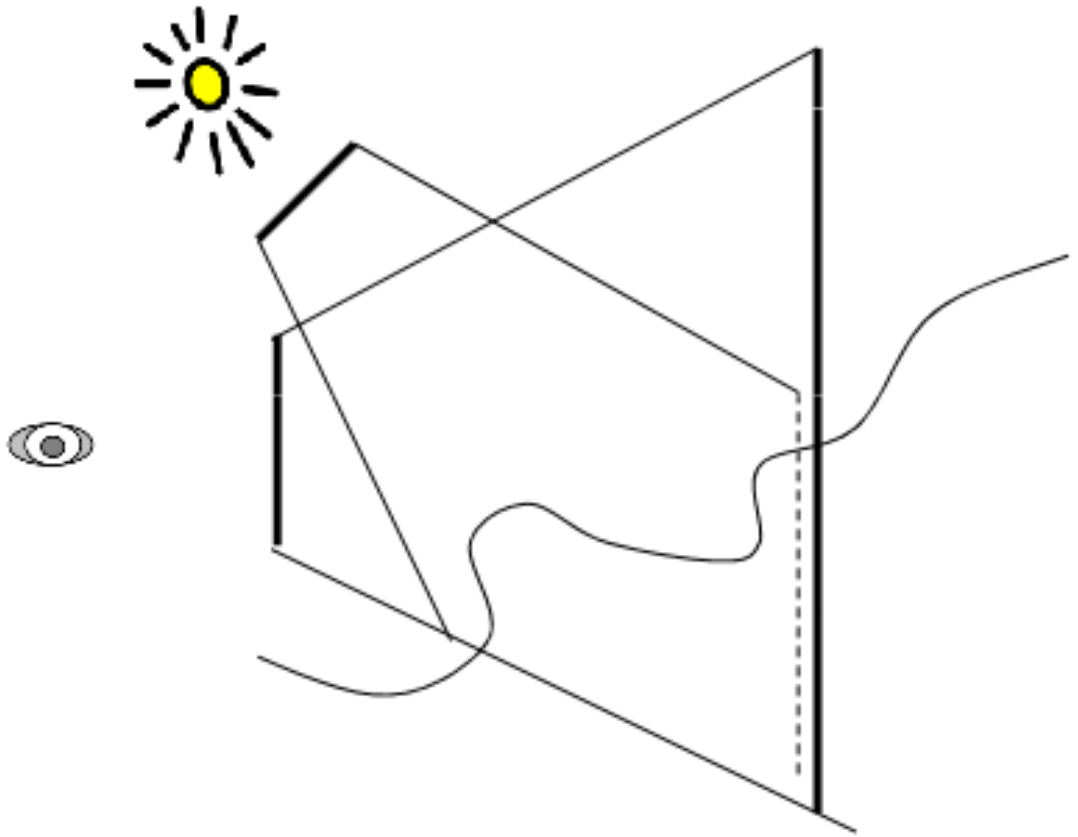
z-fail approach



result: stencil buffer has the number of shadow volumes behind an object, thus we know what is in the shadow (stencil > 0)
→ shadows behind an object mean that there's also shadows in front of them

Solution: *z*-fail approach “Carmack's Reverse”

- similar issues as before,
  but at the back plane
- shadow-casting object
  fully behind back plane
  no longer a problem
- problem only from shadow
  objects intersecting
  back plane
- solution: cap the
  shadow volumes
  at the back plane

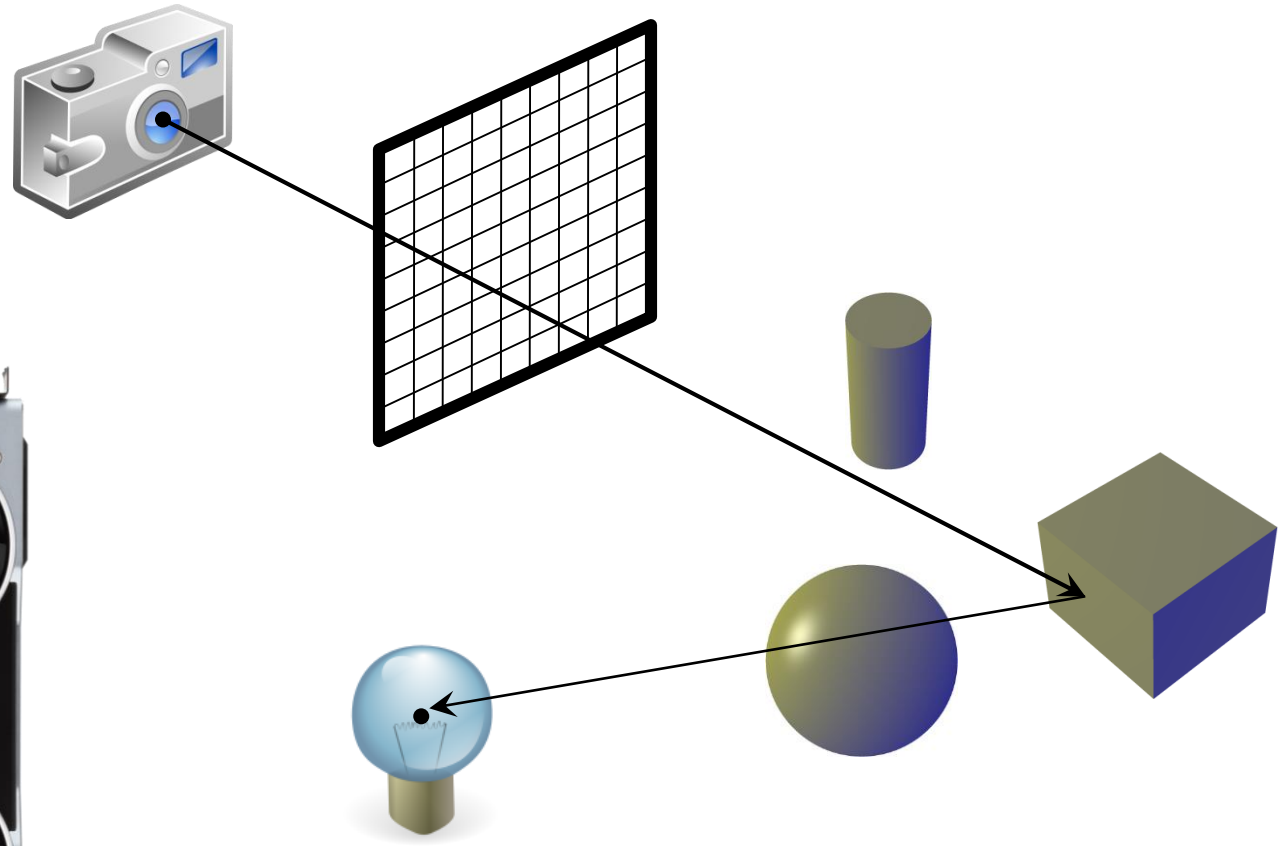


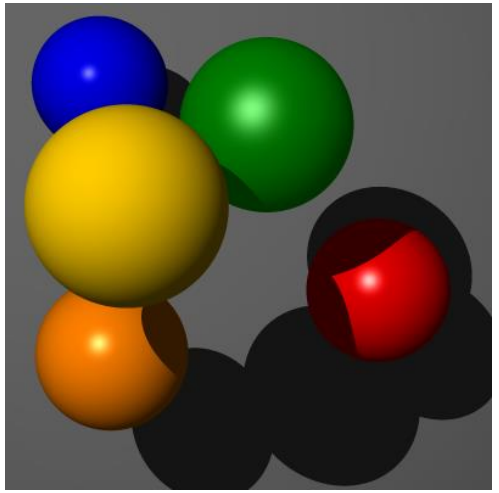
Shadow Volumes: First used in Doom 3

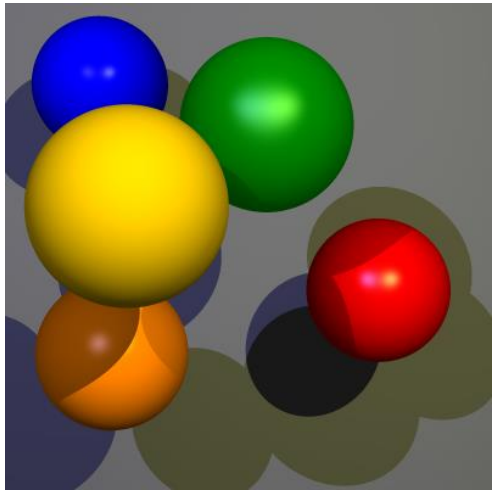
Evaluation

- pros
  - occluders can shadow themselves
  - no need to analyze geometry to extract occluders/receivers
  - high precision, not bound by resolution of a shadow map
- cons
  - 4 render passes plus shadow volume extraction needed
  - many shadow volumes cover many pixels (rendering costs)
  - slower than shadow mapping for many shadows
  - still only discrete/hard shadows

Alternative: Shadows from raytracing

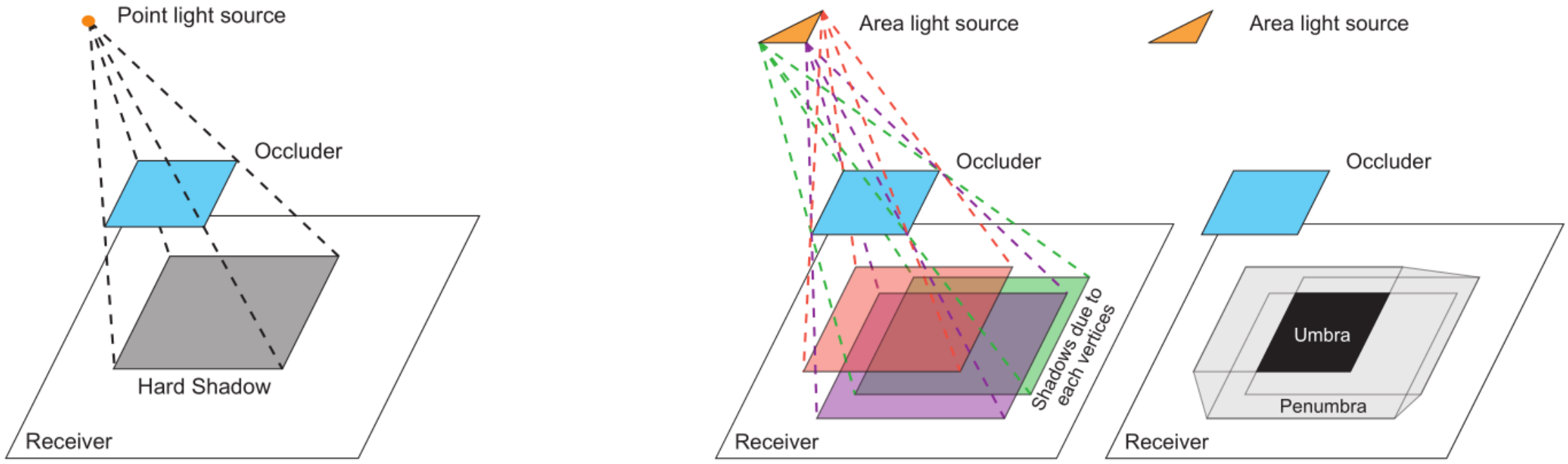






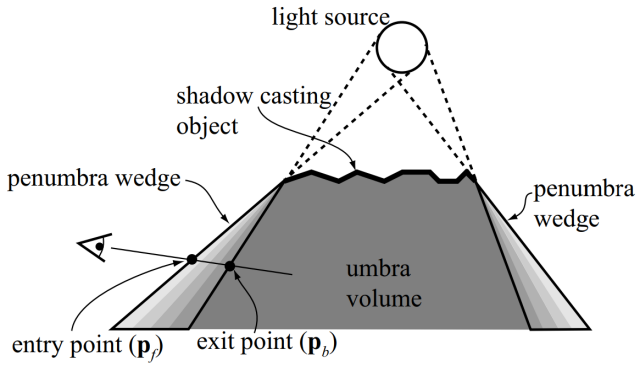
Soft Shadows

Point light sources vs. areal lights



Soft shadows

- several possible approaches
- image-based techniques, e.g.
  - combine several occlusion maps
  - shadow map with quantitative information
- object-based techniques, e.g.
  - point light source position sampling
  - smoothies
  - soft shadow volumes

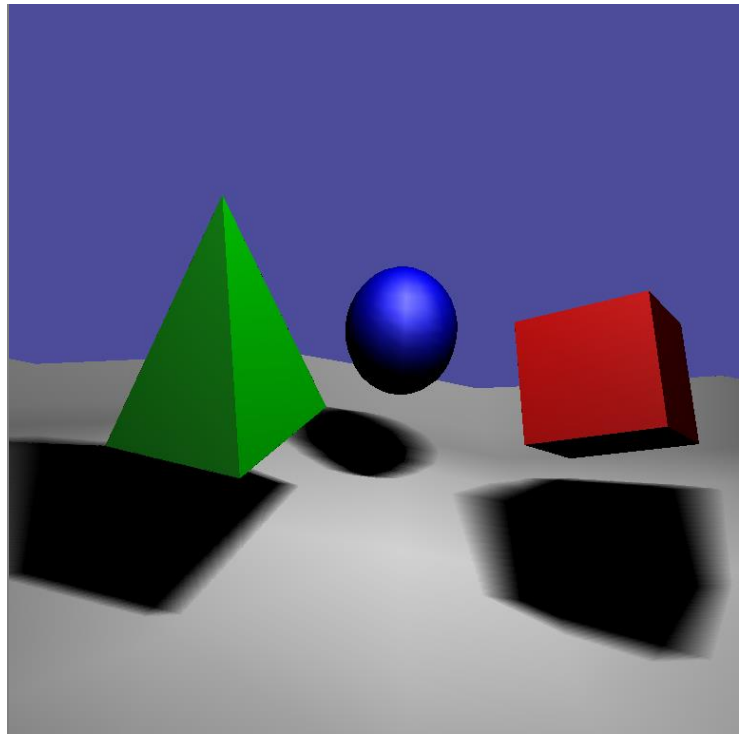


Several occlusion maps

- render planar shadow from several points close to the light source
- each render leads to an occlusion map
- combine all to an attenuation map
- use in illumination of receiver

Shadow map with quantitative information

- simple case: linear light source
- compute 2 shadow maps: 1 per end
- detect discontinuities in shadow maps
- create vertical polygons that connect occluder and receiver w.r.t. light source
- render these from the other sample with Gouraud shading: **visibility channel**
- real render: check both shadow maps, for unclear situations use visibility channel

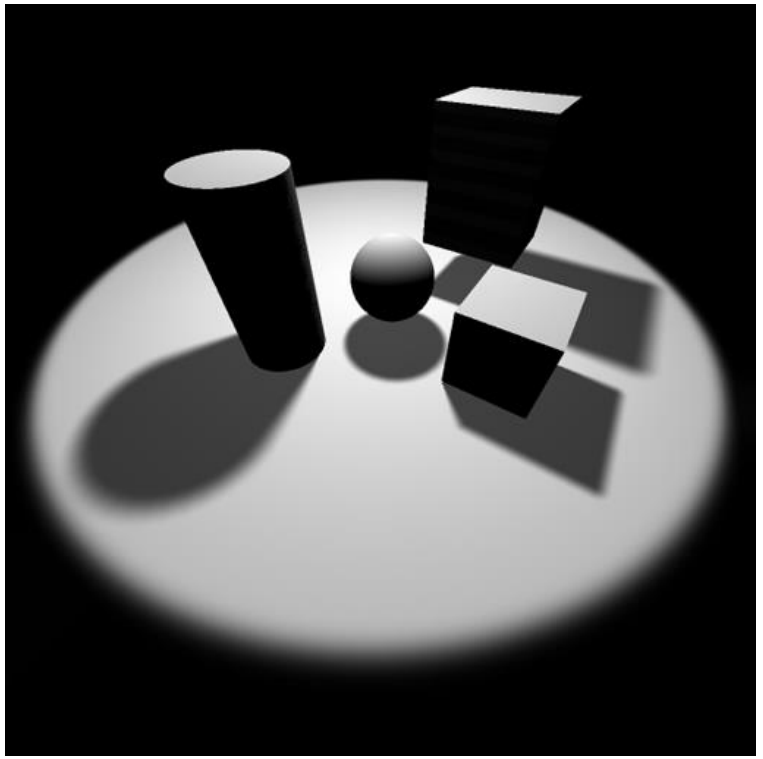


Point light source position sampling

- render scene several times, each time slightly changing the location of the light source
- blend the resulting images for a soft shadow effect

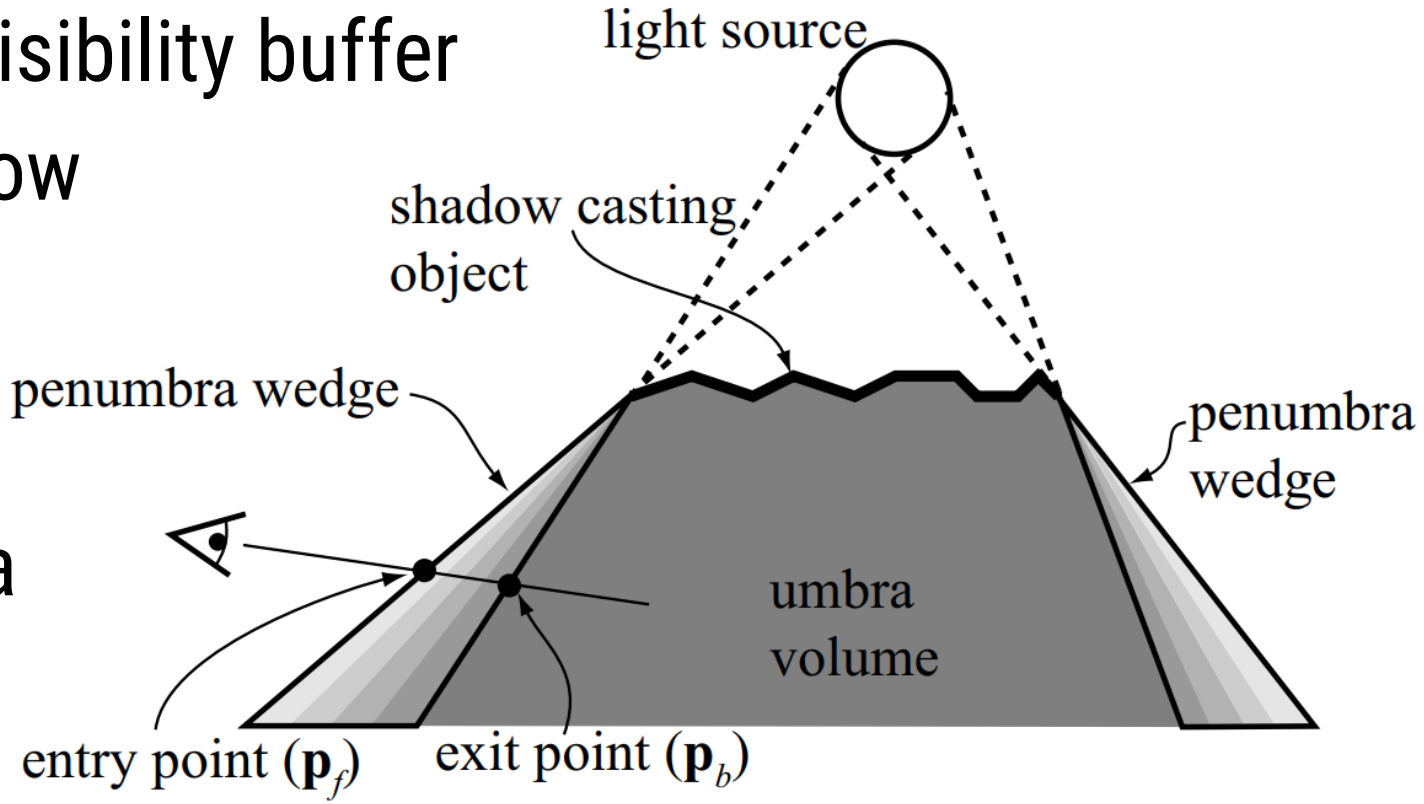
Smoothies

- generate shadow maps
- generate smoothie buffer that extends the objects' silhouette from the view of the light
- smoothie buffer alpha records ratio of distances between light, occluder, and receiver
- final render blends results from shadow map and smoothie buffer

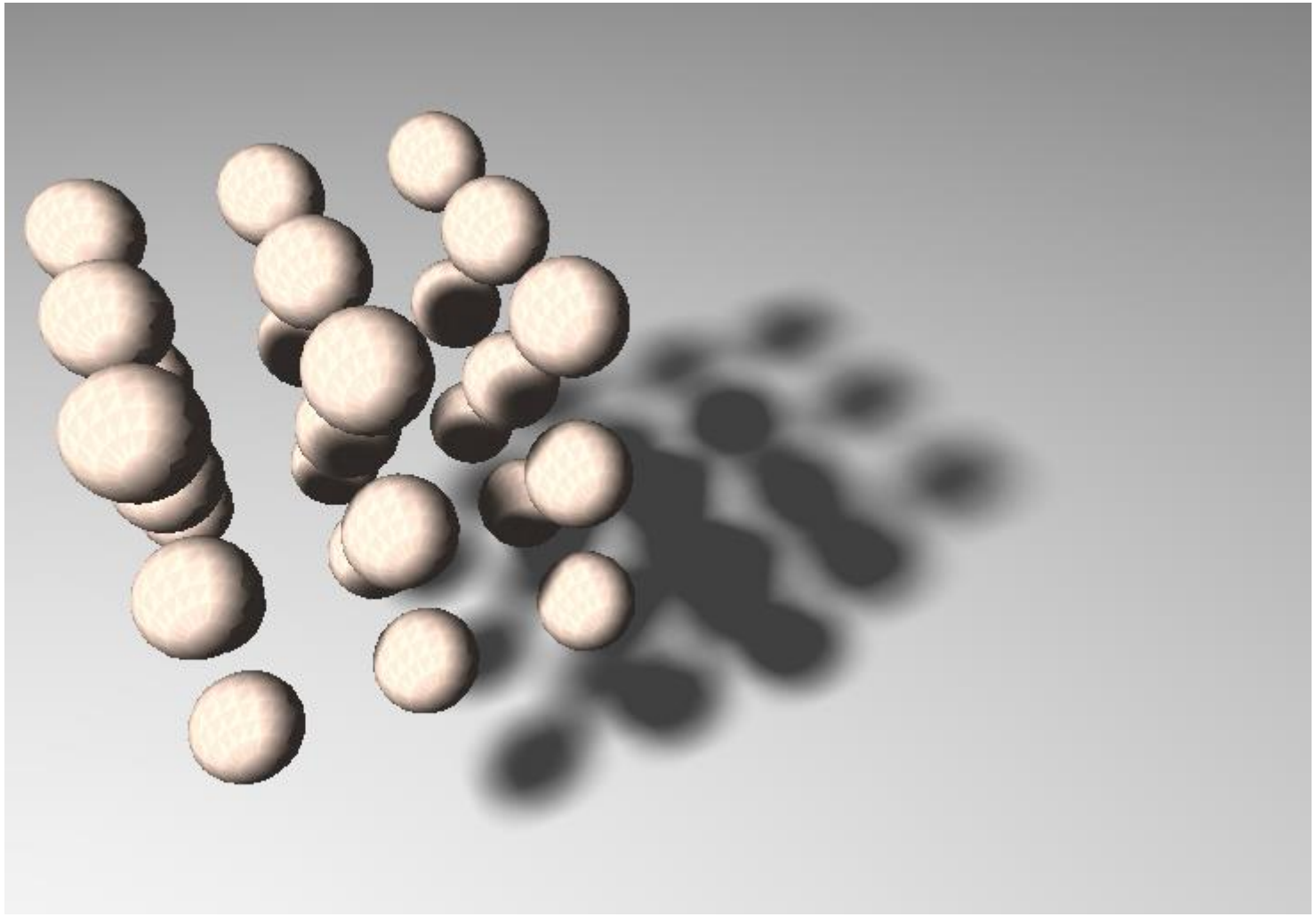


Soft shadow volumes

- based on regular shadow volumes
- create penumbra wedges from silhouette
- use of separate buffer: visibility buffer
- first render normal shadow volume: hard shadows
- then render wedges using fragment shader that computes penumbra
- use visibility buffer

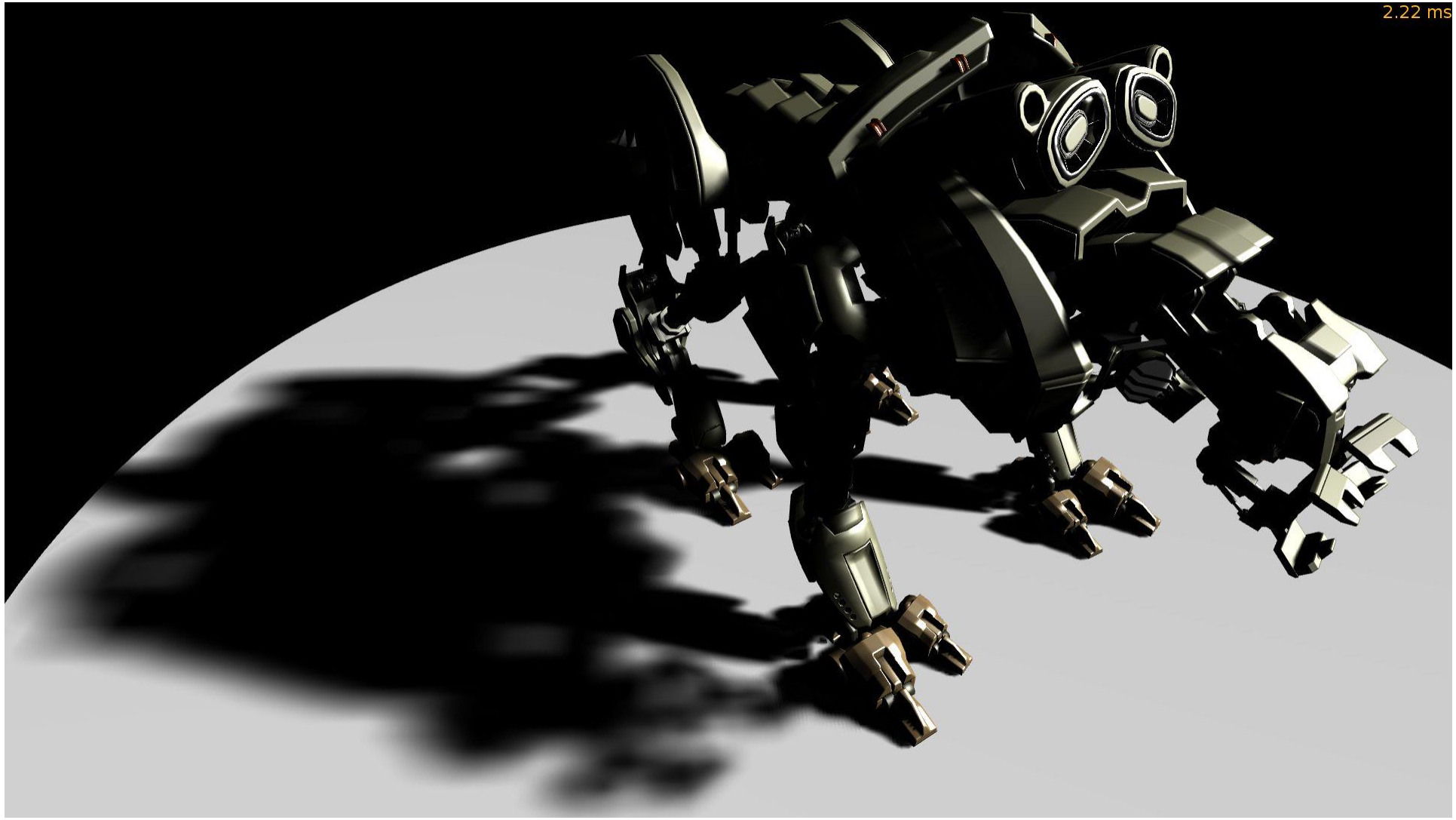


Soft shadow volumes: Result



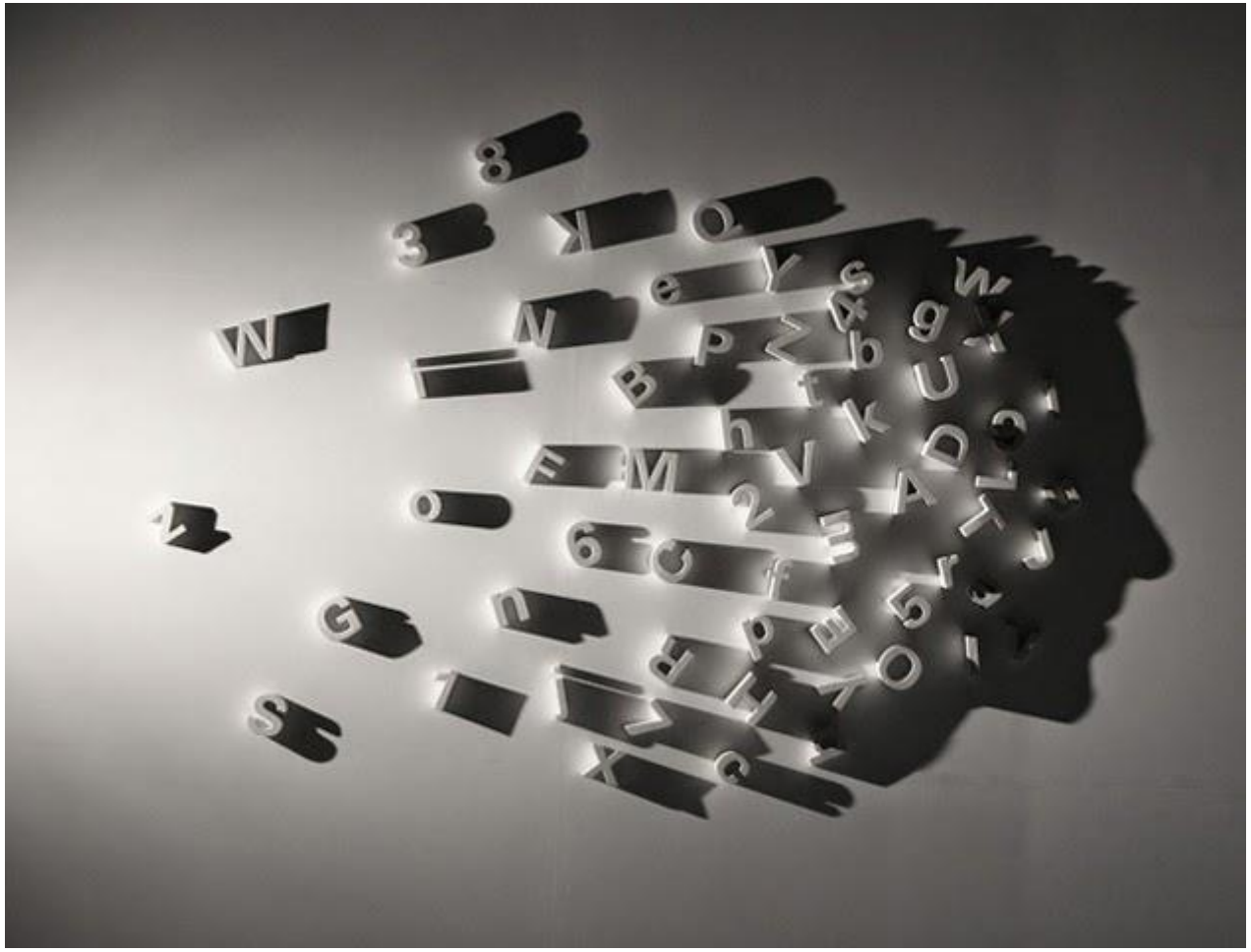
Soft shadows: Other results

Soft shadows: Other results



Shadow Art

Shadow art



Shadow art


SIGGRAPHASIA2009
Shadow Art
Niloy J. Mitra
IIT Delhi / KAUST
Mark Pauly
ETH Zurich

Shadow computation: Summary

- shadows not included in plain rendering,
  but essential for scene perception
- variety of algorithms for computation
  - shadow mapping
  - shadow volumes
- choice depends on scene complexity
- soft shadows more complicated, more expensive
- heuristics for soft shadows