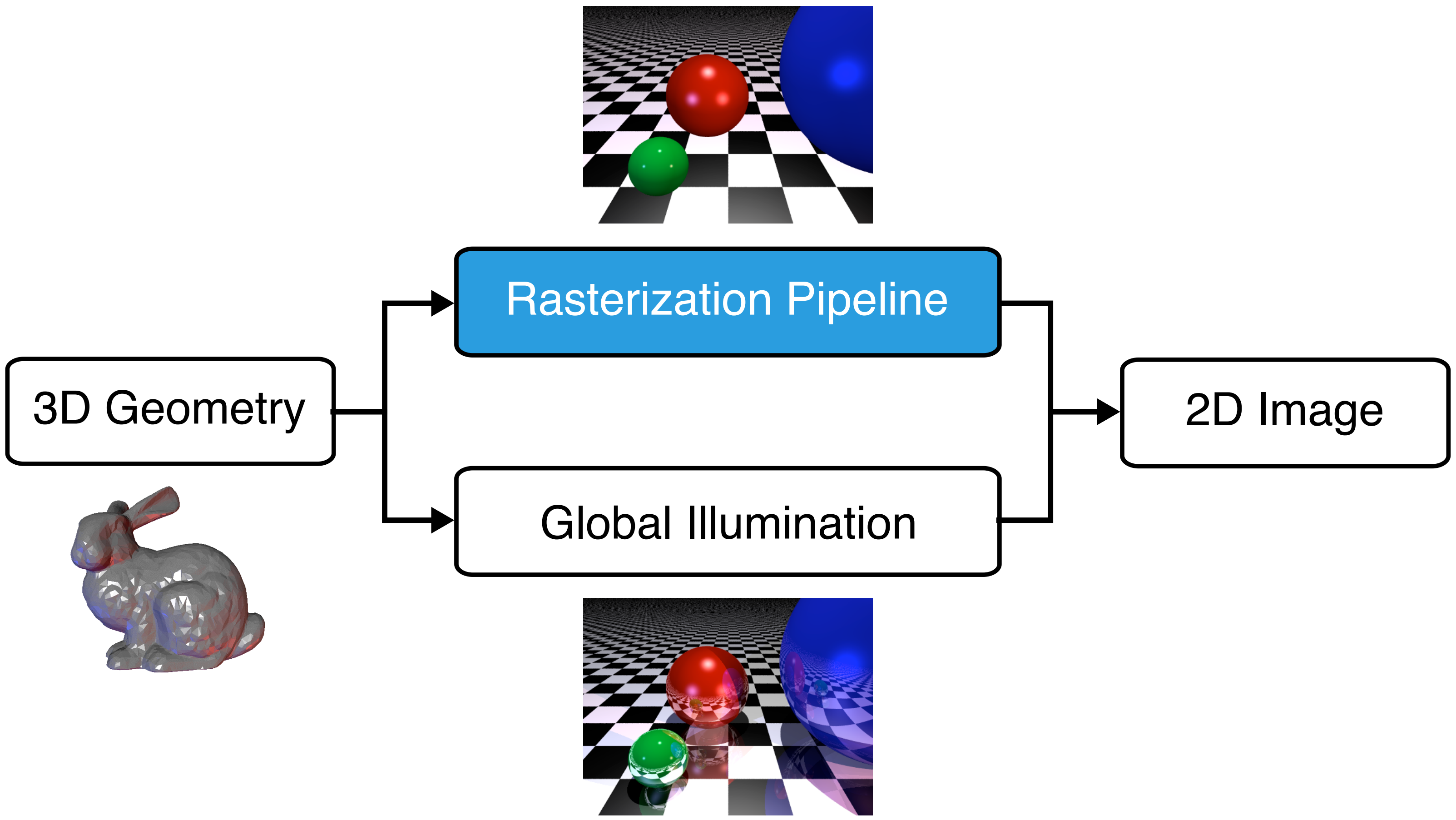
Computer Graphics

Shadow Mapping

Mark Pauly

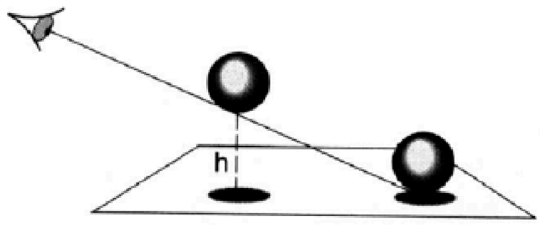
Geometric Computing Laboratory

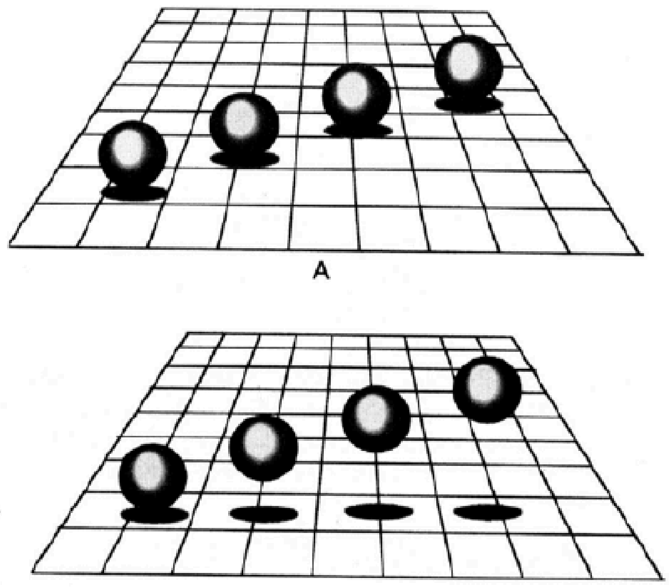
Overview


3D Geometry
Rasterization Pipeline
Global Illumination
2D Image

Shadows

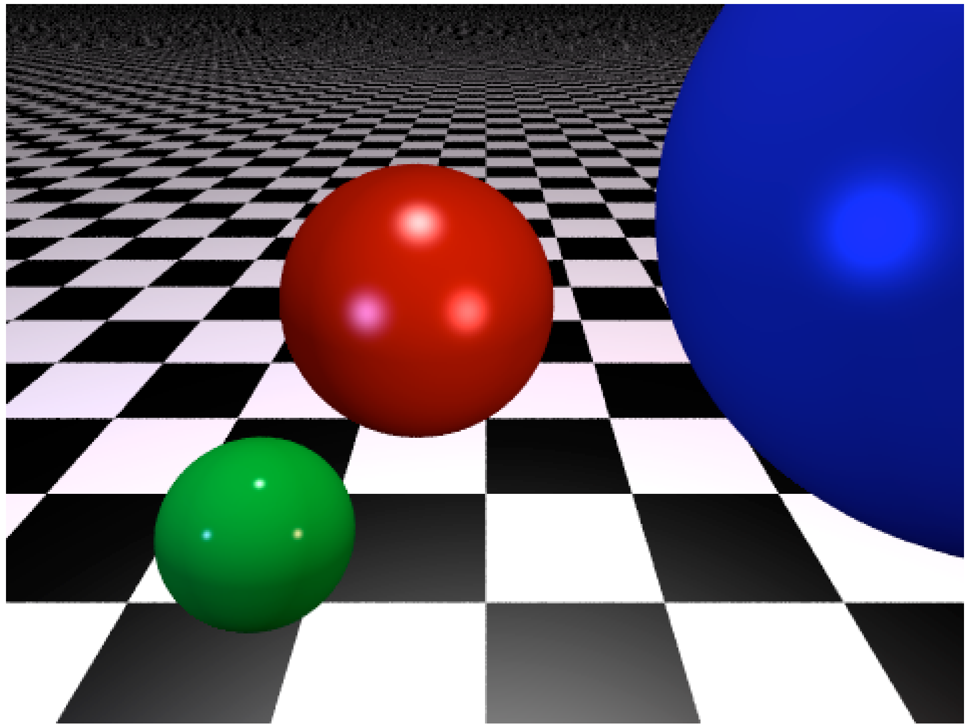
Shadows are important for 3D depth perception

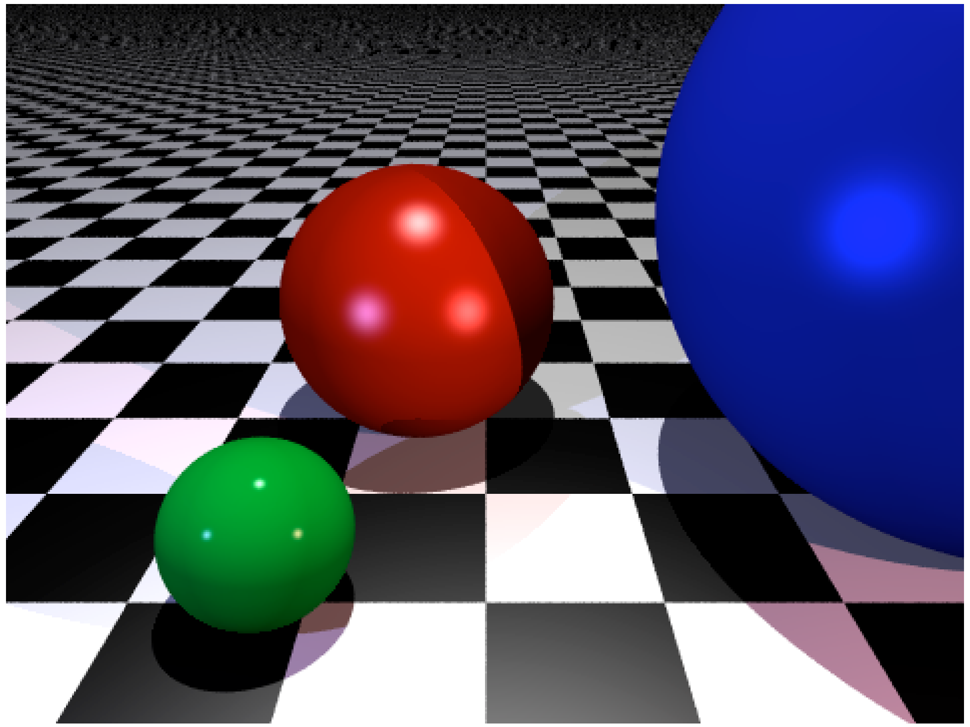




Shadows

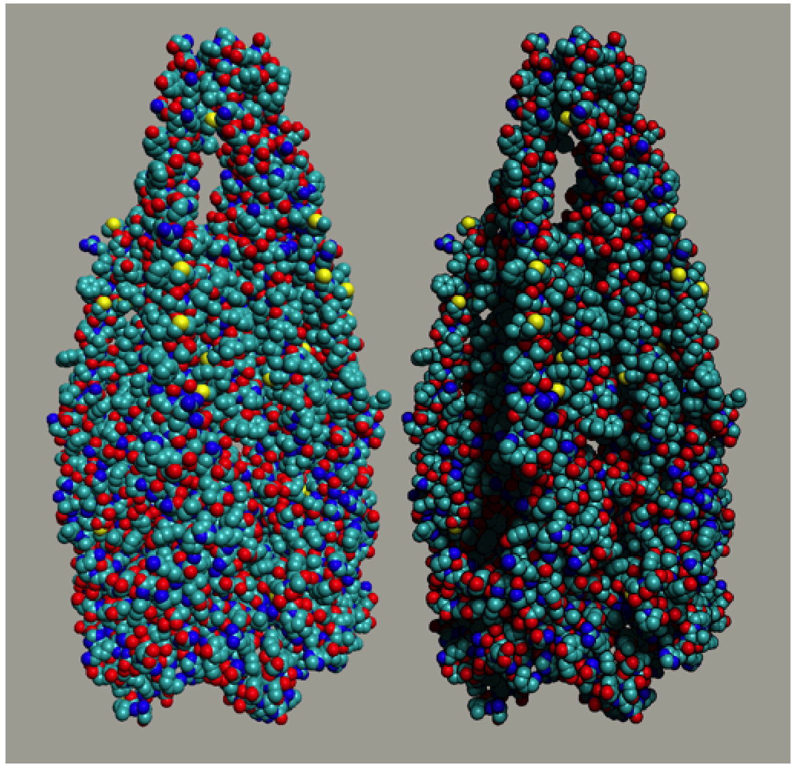
Shadows are important for 3D depth perception





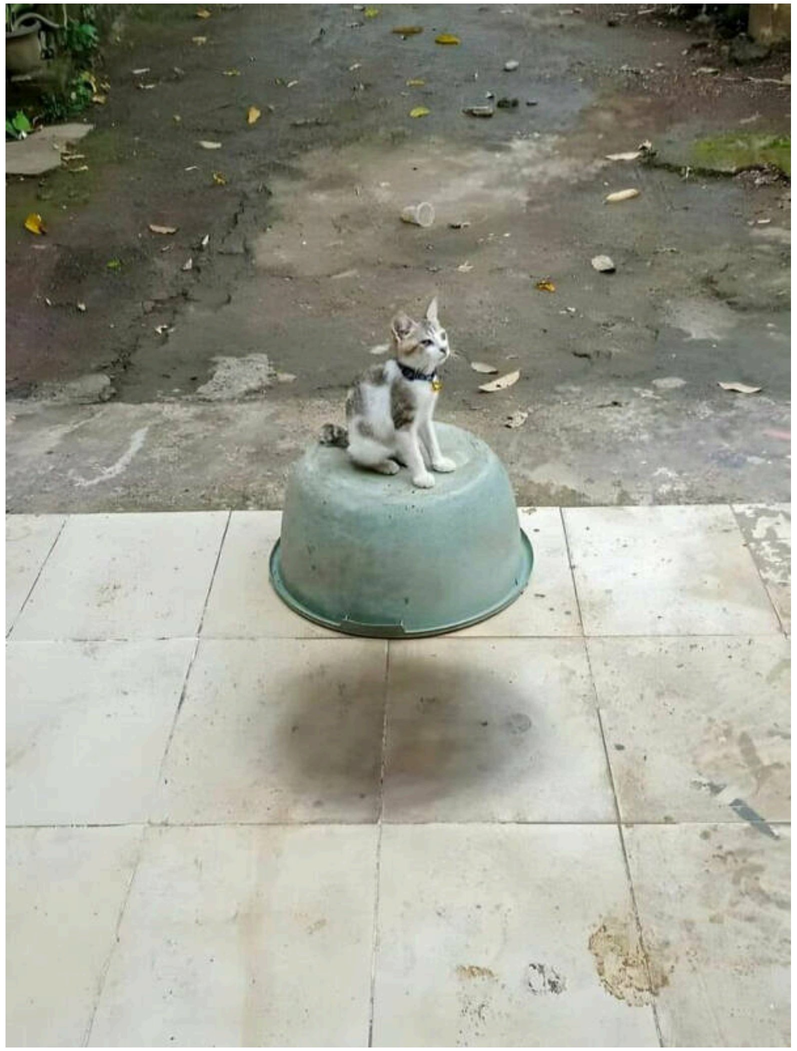
Shadows

Shadows are important for 3D depth perception

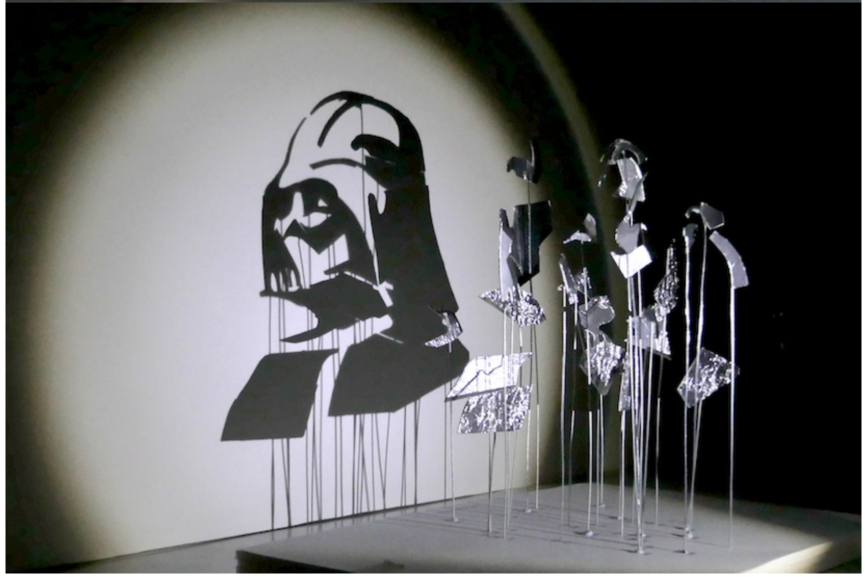
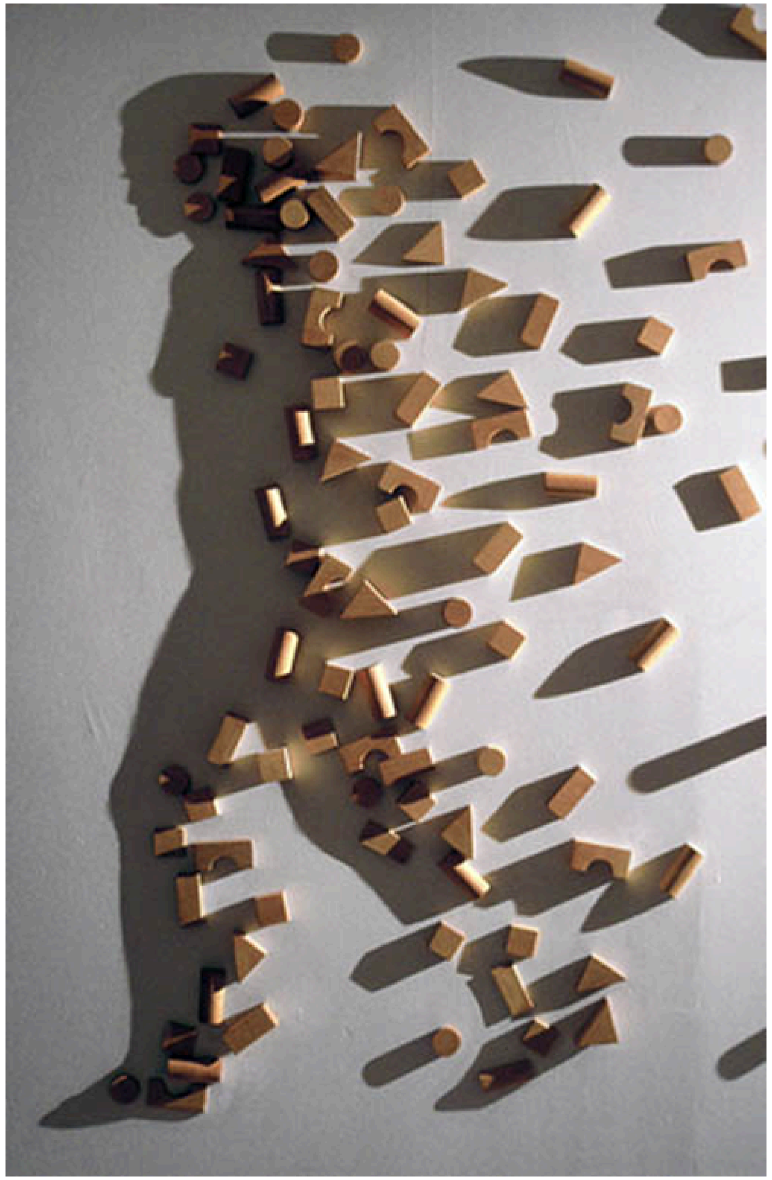


Shadows

Shadows are important for 3D depth perception



Shadow Art

Shadow Art Research

Shadow Art

Niloy J. Mitra
IIT Delhi / KAUST

Mark Pauly
ETH Zurich

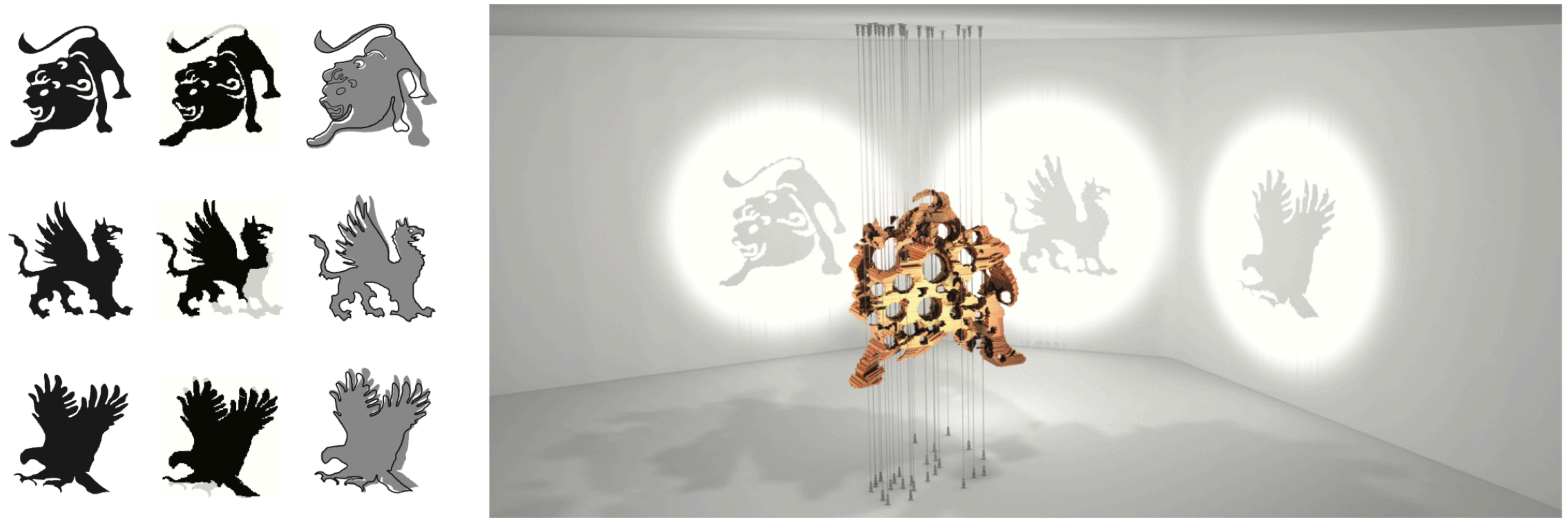


Figure 1: *A 3D shadow art sculpture that simultaneously casts three distinct shadows. The side columns show the desired shadow image provided by the user (left), inconsistencies due to conflicting shadow constraints (middle), and optimized images (gray) that avoid shadow conflicts with the outline of the original for comparison (right).*

Shadow Computation

- Shadow computation is very similar to visibility computation:
  - **Visibility**: Which objects can be seen from the *camera*?
  - **Shadows**: Which objects can be seen from the *light source*?

Lighting & Shadows

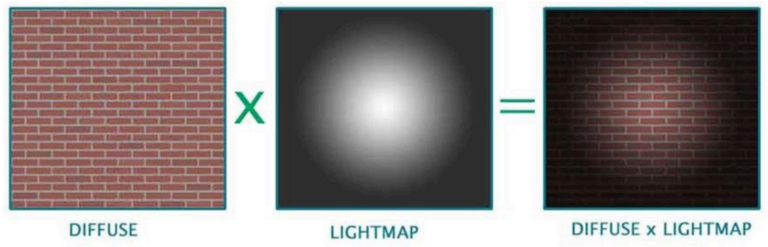
- If a light source is occluded, simply skip its diffuse and specular contribution:

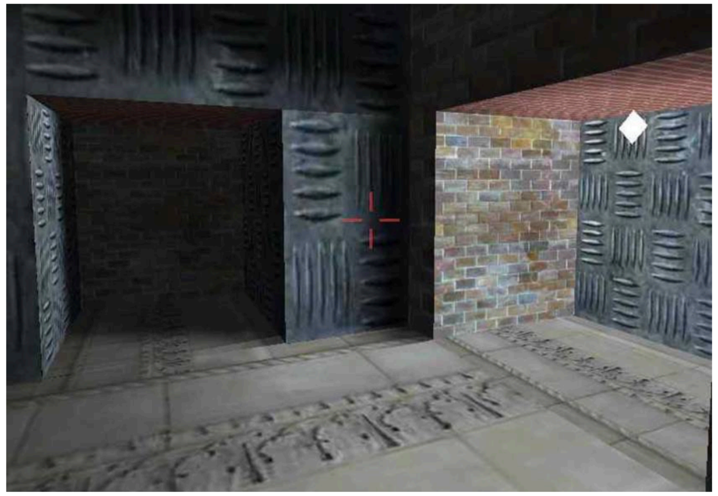
$$\begin{aligned} I(\mathbf{x},\mathbf{n},\mathbf{v}) = I_{\text{ambient}} \\ + \sum_i \text{shadow}(\mathbf{x},\mathbf{l}_i) \cdot \big(I_{\text{diffuse}}(\mathbf{x},\mathbf{n},\mathbf{l}_i) + I_{\text{specular}}(\mathbf{p},\mathbf{n},\mathbf{v},\mathbf{l}_i)\big) \end{aligned}$$

$$\text{with} \quad \text{shadow}(\mathbf{x},\mathbf{l}_i) = \begin{cases} 1 & \text{light source } \mathbf{l}_i \text{ is visible from } \mathbf{x}, \\ 0 & \text{light source } \mathbf{l}_i \text{ is blocked from } \mathbf{x} \end{cases}$$

Light Maps & Shadows

- Store precomputed lighting in textures
  - Precomputation can take shadows into account
  - Works for **static** scenes only





Shadows in OpenGL

- Shadows are a global effect
  - Occluders block light from reciever
  - Occluders can move/deform dynamically
- How to achieve **global** effects with a **local** rendering pipeline?
  - We have to use multiple render passes!
- Two main approaches
  - Shadow volumes (object space algorithm)
  - Shadow maps (image space algorithm)

Shadow Maps

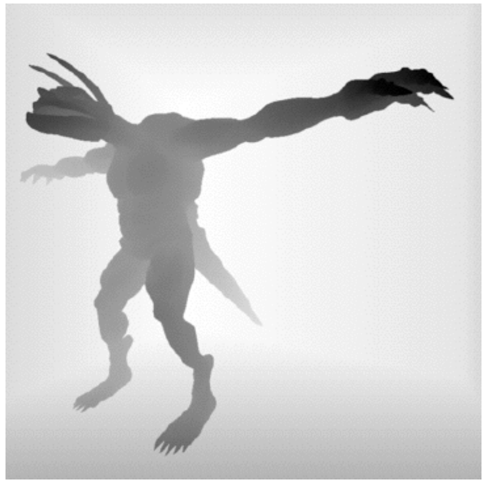
Shadow Computation

- **Visibility**
  - Which objects can be seen from the camera?
- **Shadows**
  - Which objects can be seen from the light source?

Apply standard visibility techniques for shadows:
z-buffer $\longrightarrow$ *shadow map*

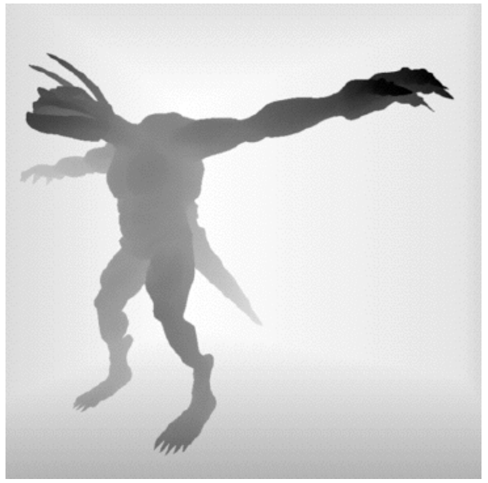
Shadow Maps

1. Render scene as seen from light source
    - Store z-buffer (holds distance to light)
    - Light's z-buffer is called *shadow map*


z-buffer from light source

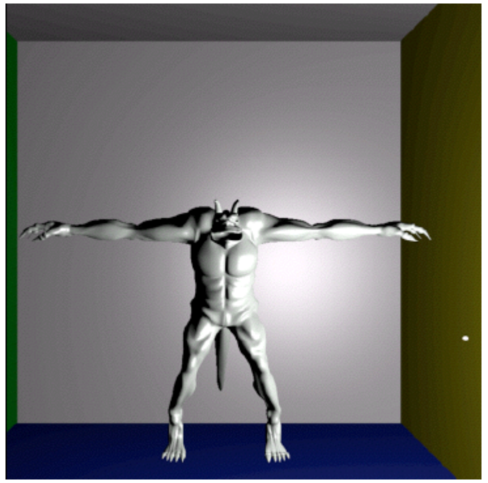
Shadow Maps

1. Render scene as seen from light source
   - Store z-buffer (holds distance to light)
   - Light's z-buffer is called *shadow map*



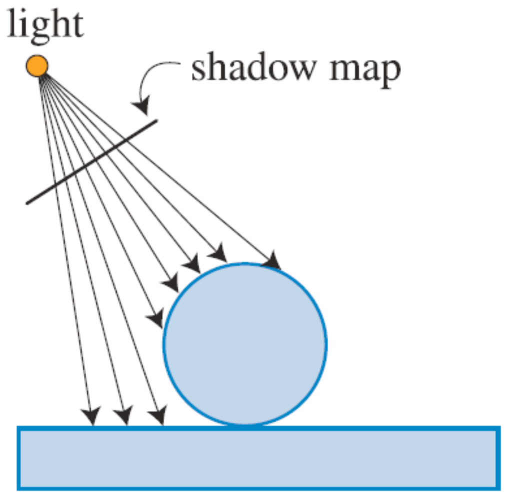
z-buffer from light source

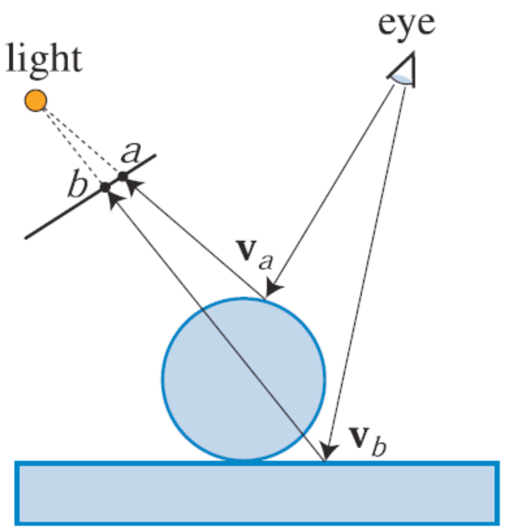
2. Render scene from eye point
   - Light a certain image plane pixel (x, y)?
   - Map it back into word coordinates: (x', y', z')
   - Project it into shadow map: (x'', y'')
   - If distance point-to-light > depth stored in map
     - Point is in shadow!

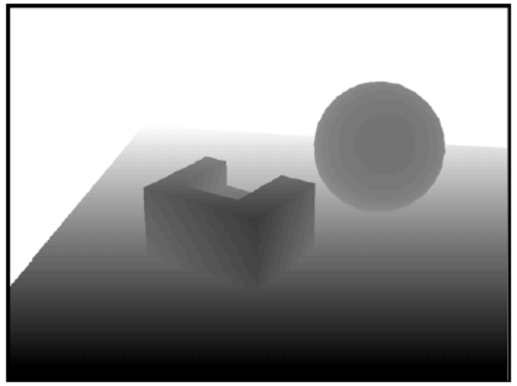


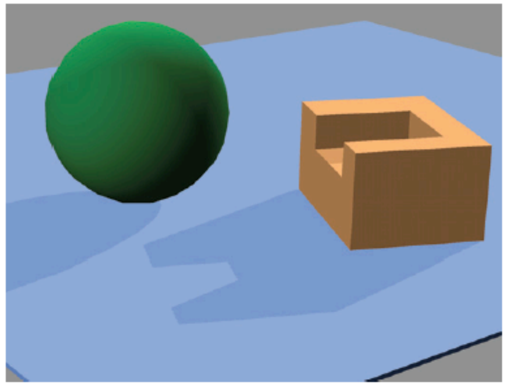
Scene rendered from eye point

Shadow Maps



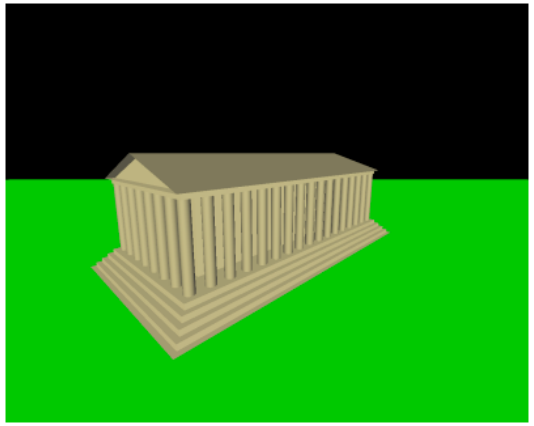


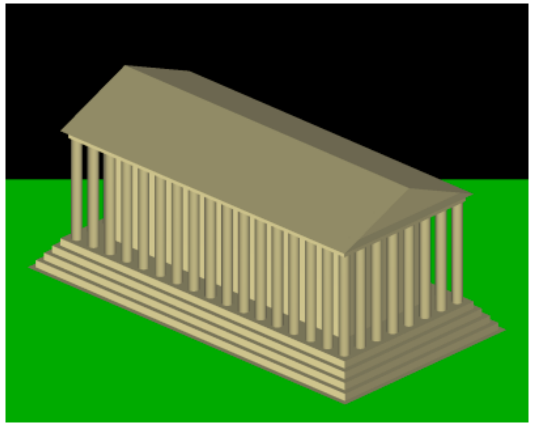


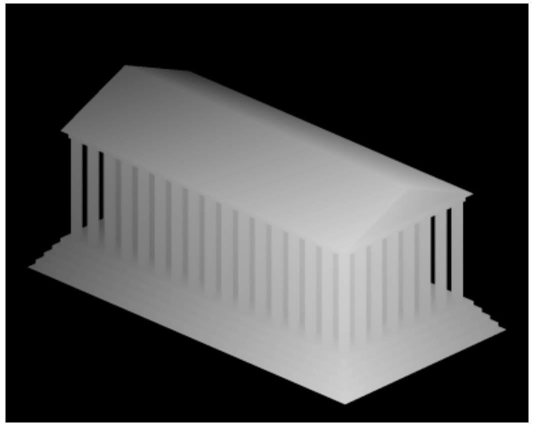


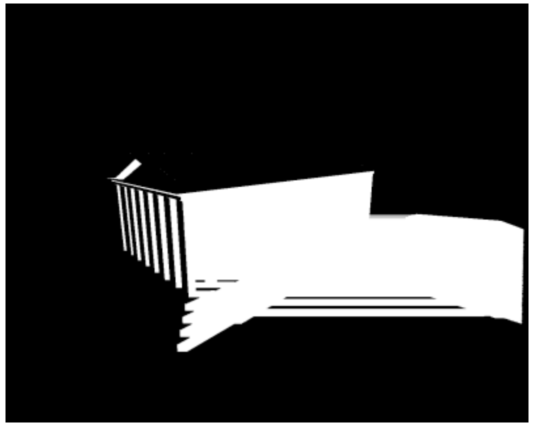
Images from Akenine-Möller et al., Real-Time Rendering

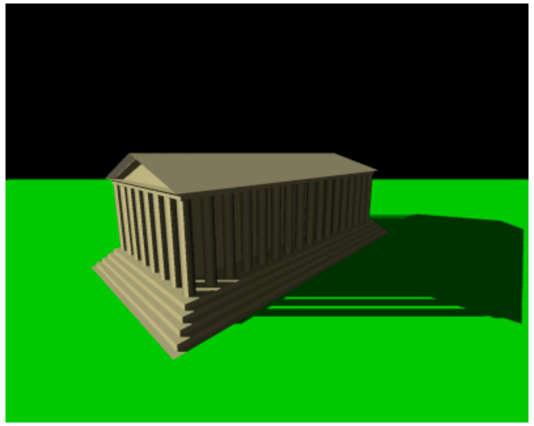
Shadow Maps


Scene without shadows


Seen from light's view

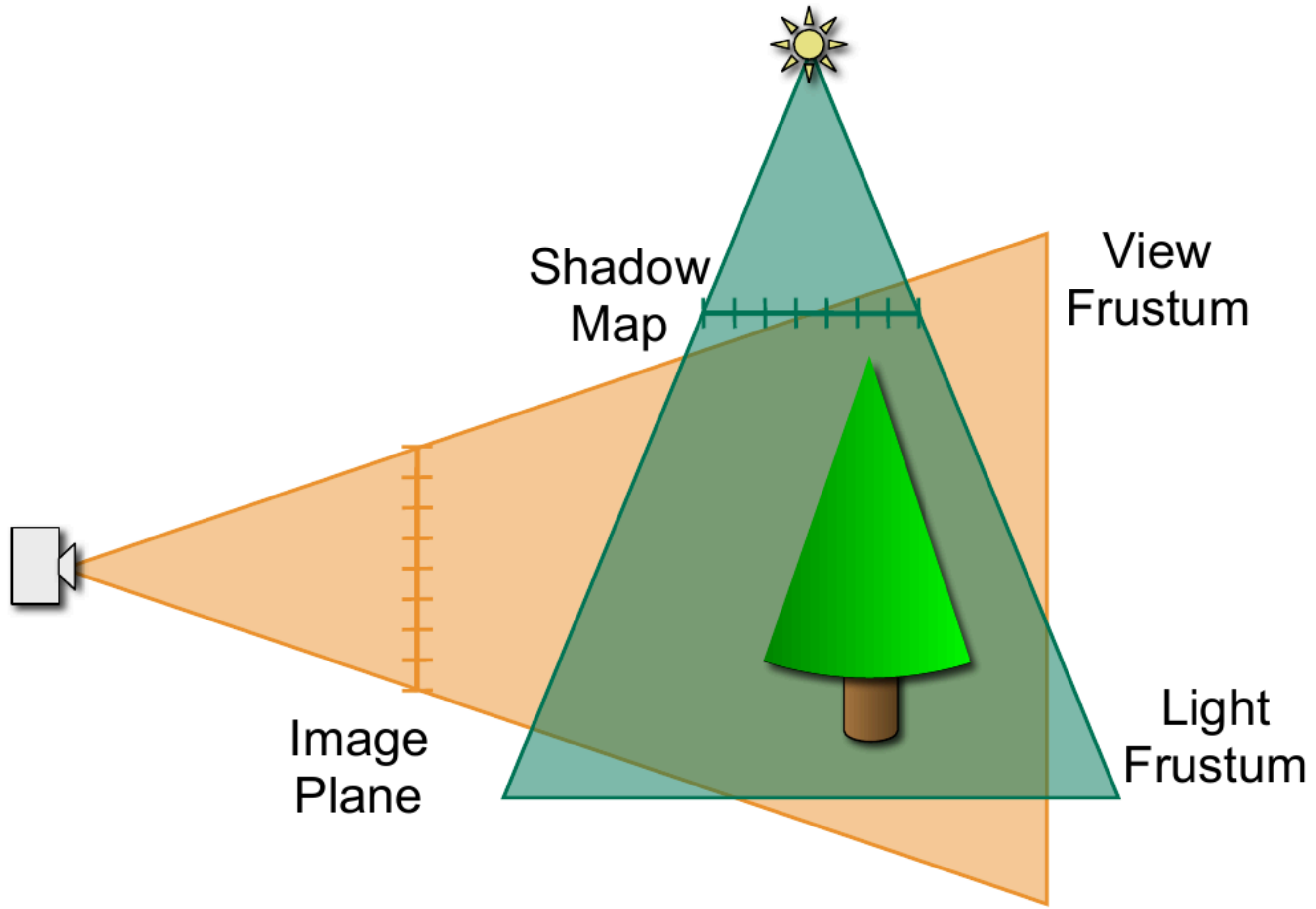

z-buffer of light's view


Pixels in shadow


Scene with shadows

Images from Wikipedia

Shadow Maps



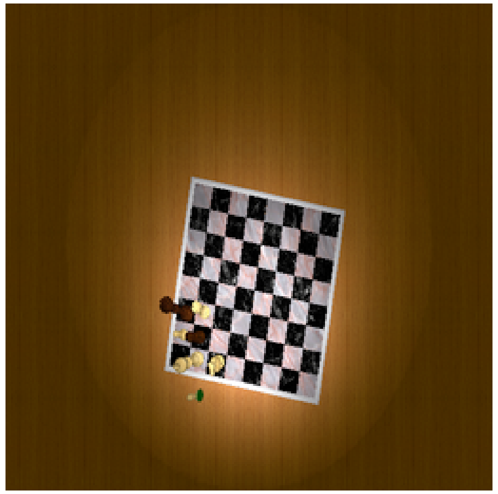
OpenGL Implementation

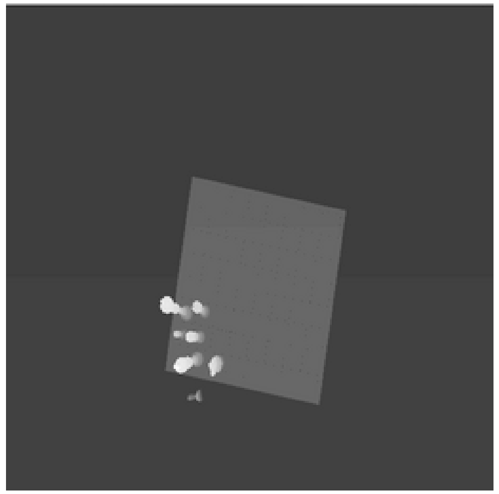
1. Render scene with ambient lighting only
    - Update frame- and z-buffer
2. For each light source
    1. Render scene from light source
        - Store z-buffer in shadow map
    2. Render scene with light contribution (accumulate)
        - Shadow map look-up for each pixel
        - Pixels in shadow are discarded
        - Other pixels are lit and rendered

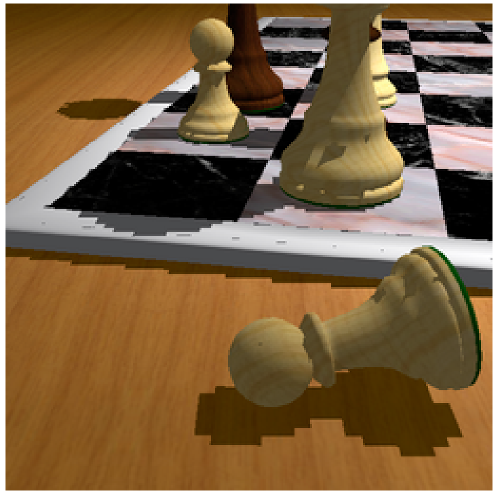
Demo: Shadow Maps

three.js shadow map demo

Shadow Map Aliasing


light view


shadow map


camera view

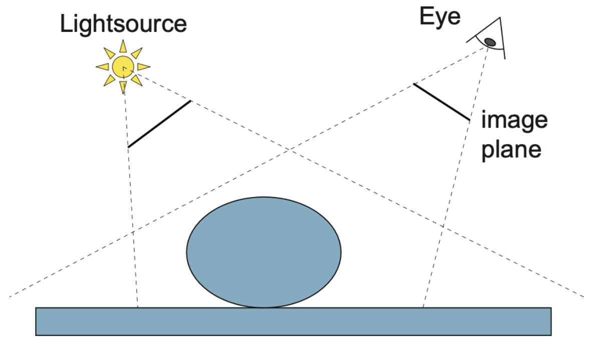
Many image plane pixels map to the same shadow map pixel

Shadow Maps Summary

- Works for everything that can be rasterized
  - Self-shadowing, alpha-textured objects
- Advanced techniques for soft shadow and anti-aliasing
- Omni-direction lights?
  - Needs “shadow cube maps”

Exam Question from 2024

- Consider the scene sketched below with two opaque objects. Assume we have computed a shadow map *S* from the point light source using the indicated view angle. Now we multiply every value in the shadow map by a factor of 2.

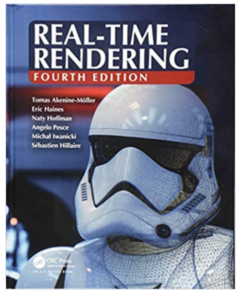


Which of the following statements is true? Please check the corresponding box on the right. (correct answer = 1 point, no answer = 0 points, wrong answer = -0.5 points)

Statement	**True**	**False**
Some points that should not be in shadow are now in shadow.		
Some points that should be in shadow are no longer in shadow.		
All points are still correctly in shadow or not, but the shadows are less dark.		

Literature

- Akenine-Möller et al.: Real-Time Rendering, Taylor & Francis, 2021.
    - Chapter 7



- Shreiner et al: *OpenGL Programming Guide*, 8th edition, Addison-Wesley, 2013.
    - Chapter 7