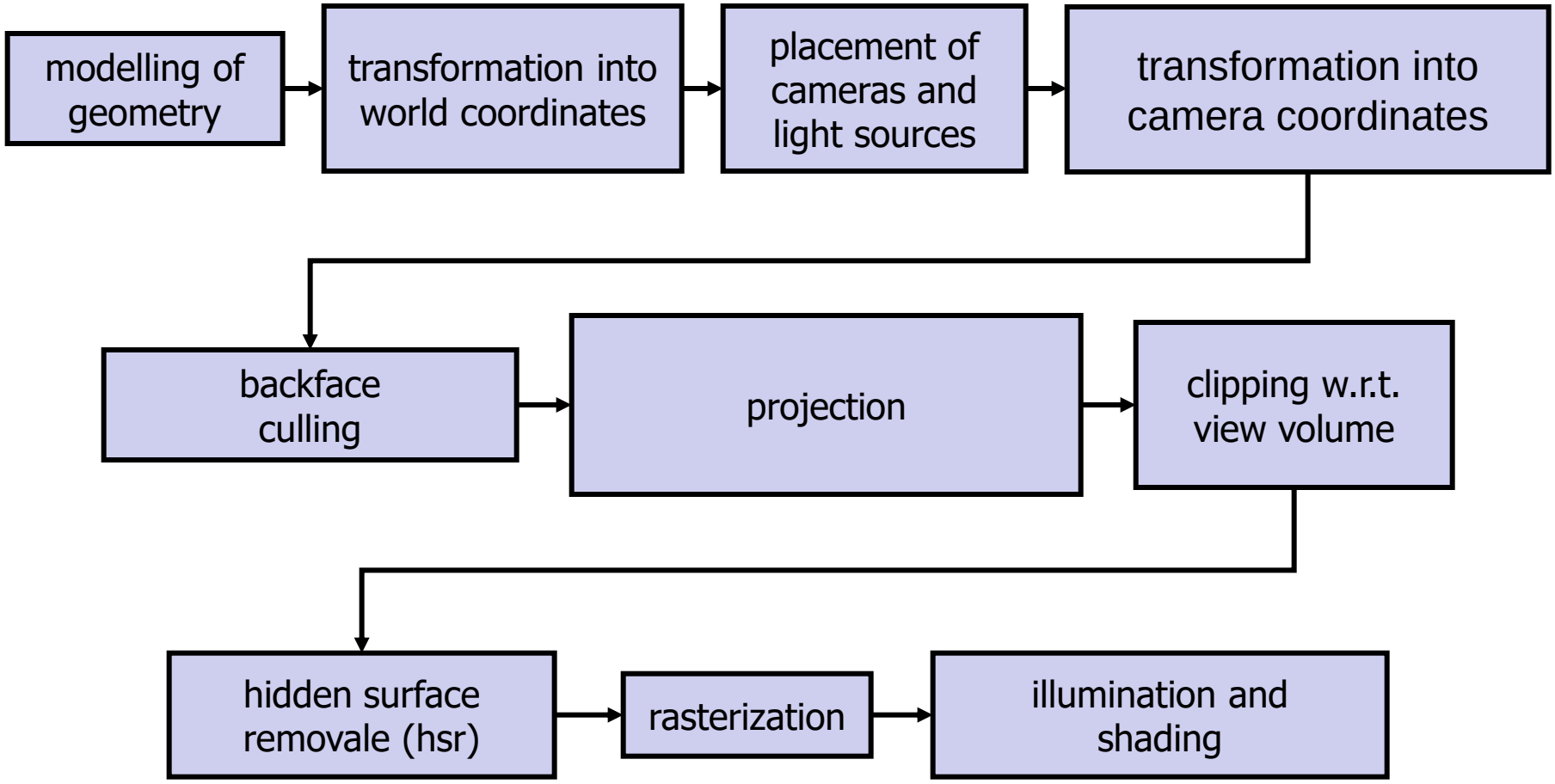
Computer Graphics

Hidden Surface Removal

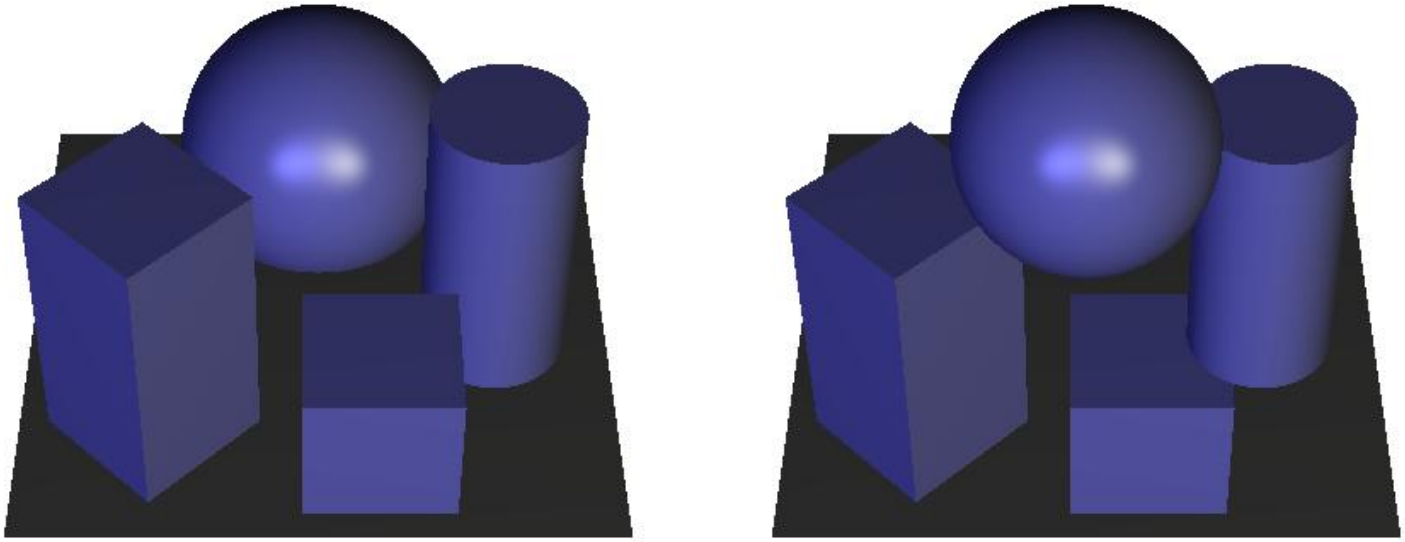
Rendering Pipeline



Hidden Surface Removal

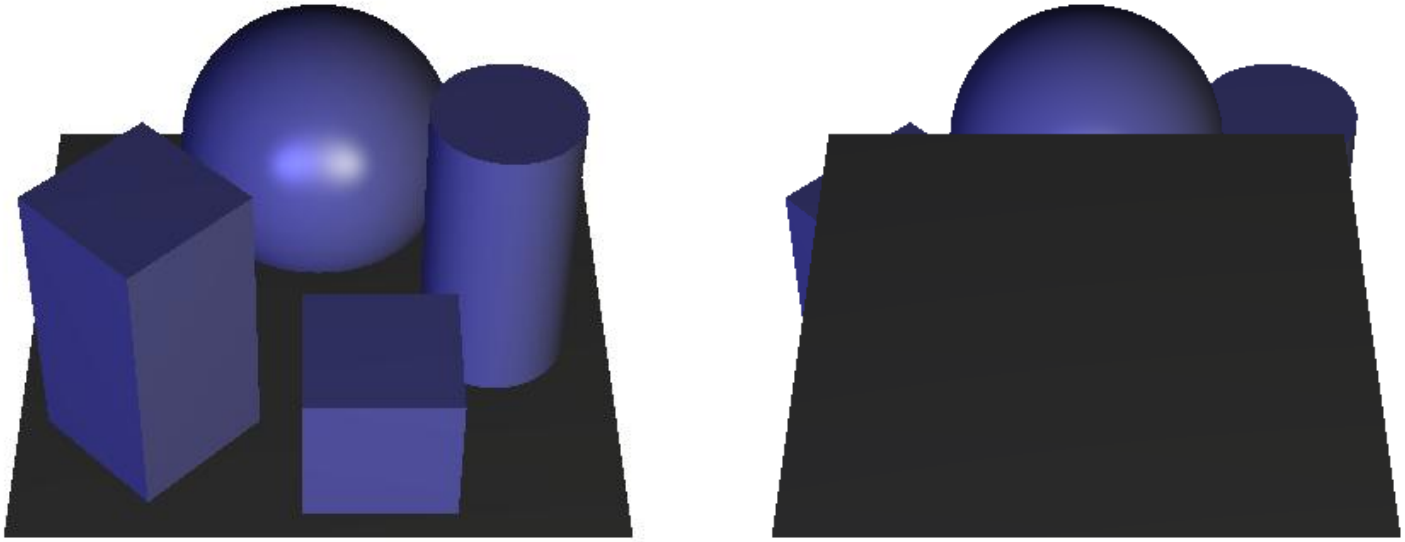
Hidden Surface Removal: Motivation

- goals:
  - model parts independently processed
  - at the same time: show front parts only
  - avoid unnecessary processing



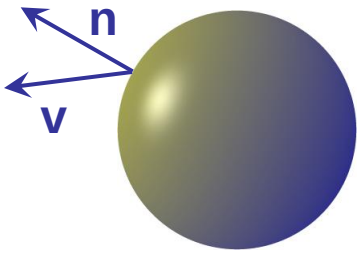
Hidden Surface Removal: Motivation

- goals:
  - model parts independently processed
  - at the same time: show front parts only
  - avoid unnecessary processing



Back Face Culling

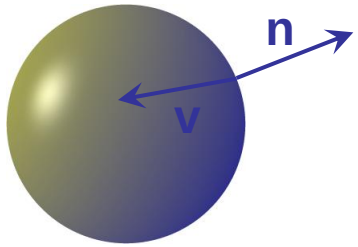
- back faces (usually) not visible
- remove these: reduction of computation
- removal early in the pipeline
- reduction of polygon count by approx. ½ of the total polygon number
- computation: compare dot product of surface normal with view direction with 0



$$\mathbf{n} \bullet \mathbf{v} > 0$$

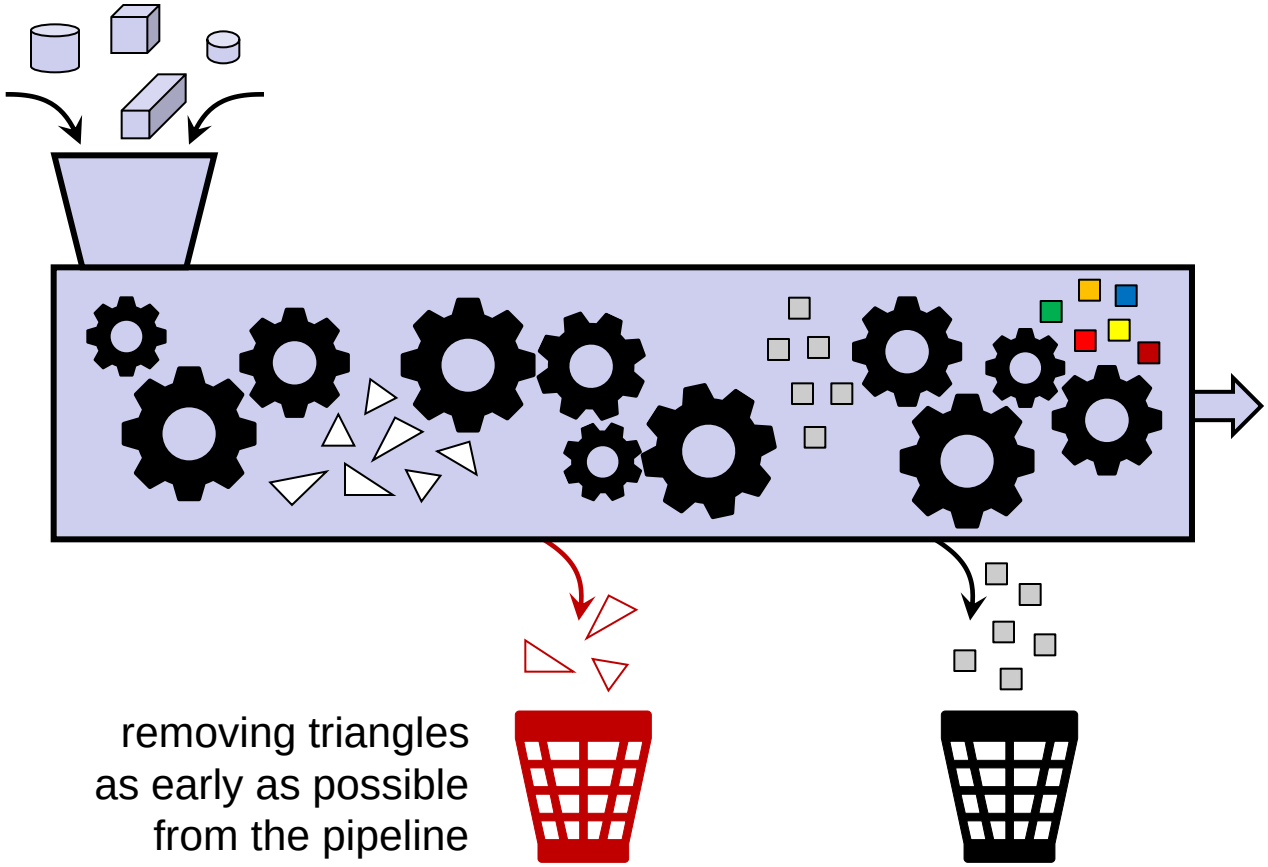
Back Face Culling

- back faces (usually) not visible
- remove these: reduction of computation
- removal early in the pipeline
- reduction of polygon count by approx. ½ of the total polygon number
- computation: compare dot product of surface normal with view direction with 0



$$\mathbf{n} \bullet \mathbf{v} < 0$$

Back Face Culling: Early Removal

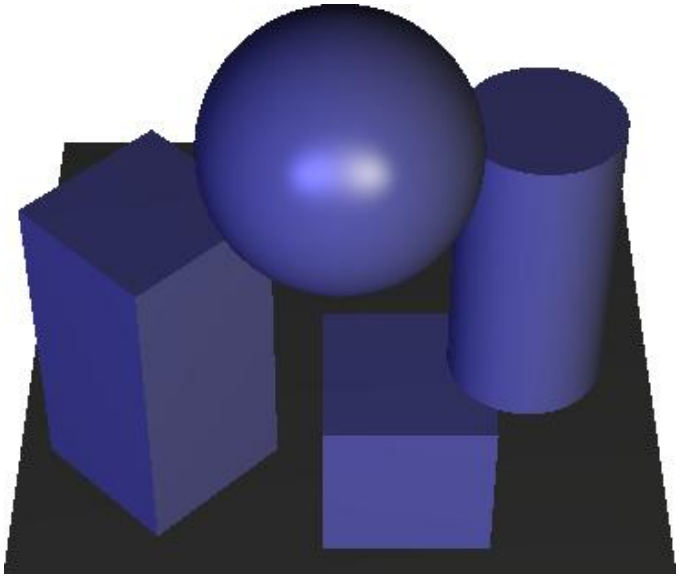


removing triangles
as early as possible
from the pipeline

iStock

Back Face Culling

- back face culling as HSR technique?
  - (usually) not sufficient due to partial overlaps of several objects in the scene
  - would work reliably only if no overlaps occur
  - e.g., if only one convex object is shown

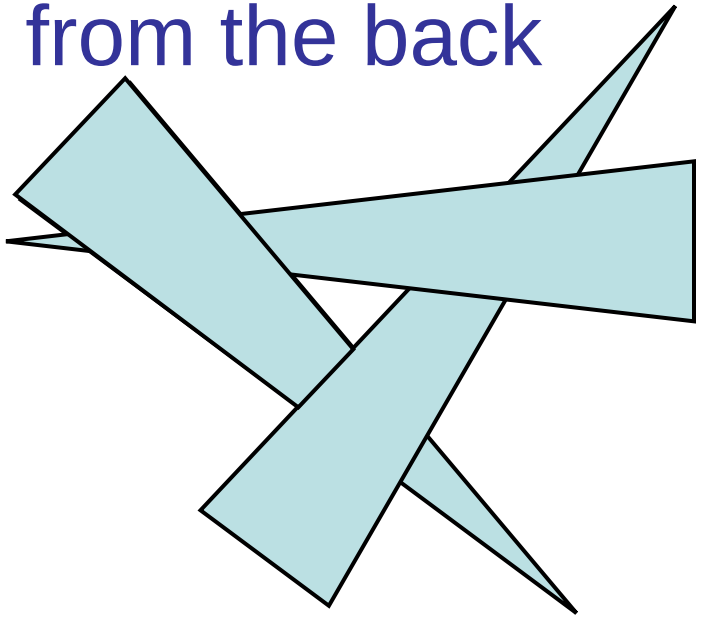


Back Face Culling

- back face culling as HSR technique?
  - (usually) not sufficient due to partial overlaps of several objects in the scene
  - would work reliably only if no overlaps occur
  - e.g., if only one convex object is shown
- order of depicted objects (overlapping) with only back face culling still depends on rendering sequence
  $\rightarrow$ idea for real HSR technique

HSR: Painter's Algorithm

- new approach to hidden surface removal (borrowed from van Gogh & co):
  draw scene from back to front
- sort scene's triangles from back to front
- start rendering triangles from the back
- problems:
  - cyclic overlaps
  - performance:
    sorting is O(n log n)

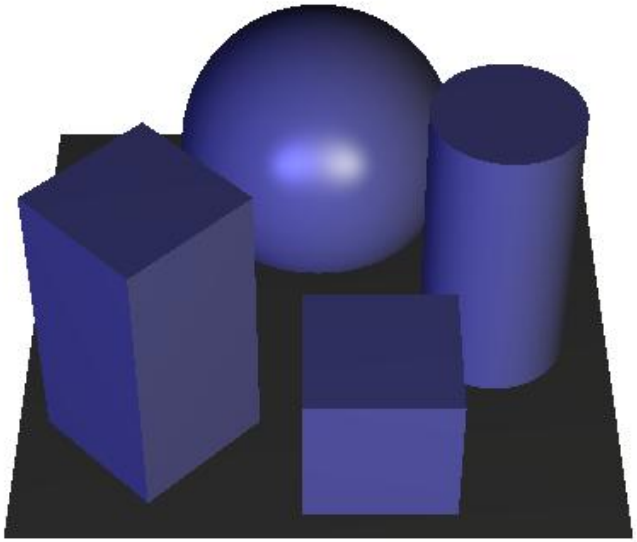


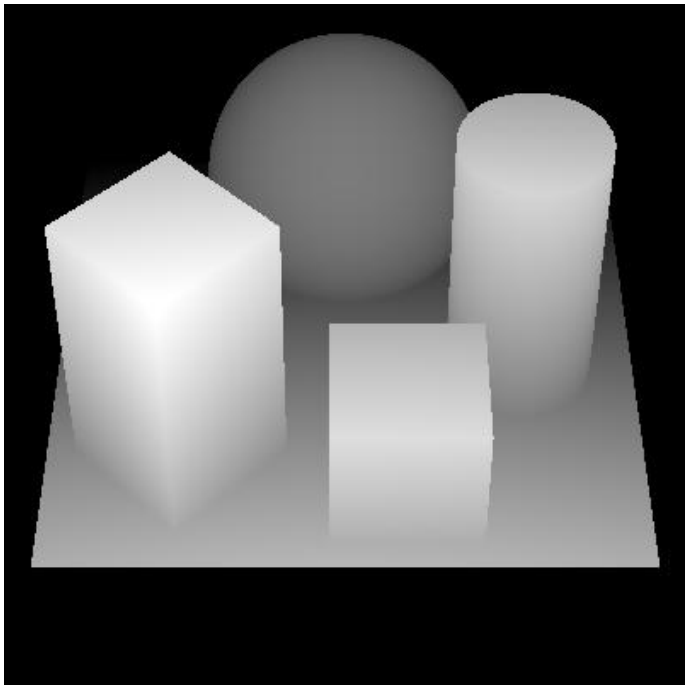
z-Buffering

- idea:
  - use advantages of Painter's algorithm (rendering from back to front)
  - avoid its disadvantages (e.g., need to sort & problems with cyclic triangles)
- realization by
  - trading speed for memory usage (memory is cheap [now anyway], time is not)
  - trading speed for accuracy (only compute what we really need/want)

z-Buffering

- introduce new pixel buffer: *z*-buffer
  (in addition to the frame buffer for image)
- *z*-buffer stores *z*-values of pixels





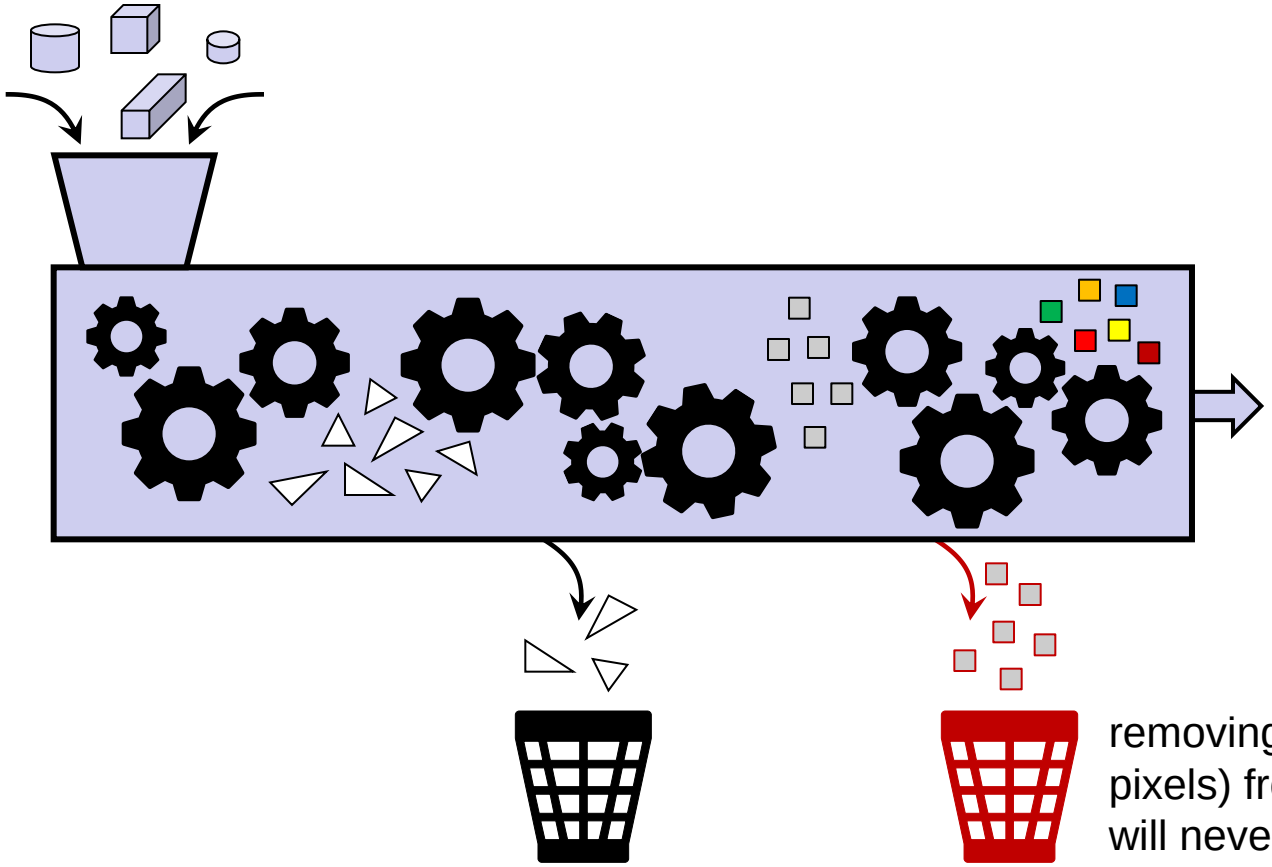
z-Buffering

- treat each primitive (triangle) individually: scene $\rightarrow$ objects $\rightarrow$ triangles $\rightarrow$ pixels
- use *z*-buffer data to determine if a new triangle is (partially) hidden or not
- at each time, the part of scene that has been processed thus far is correctly displayed

z-Buffering: Algorithm

- initialize *z*-buffer and frame buffer
- for each triangle
  - project all vertices of the triangle
  - interpolate *z*-values for each pixel (scan line)
  - before shading a pixel, test if its *z*-value is closer to camera (i.e. higher) than the current *z*-buffer value
  - if so: update *z*-buffer value and shade pixel
  - otherwise: discard pixel and continue
- after scene processing *z*-buffer contains depth map of scene

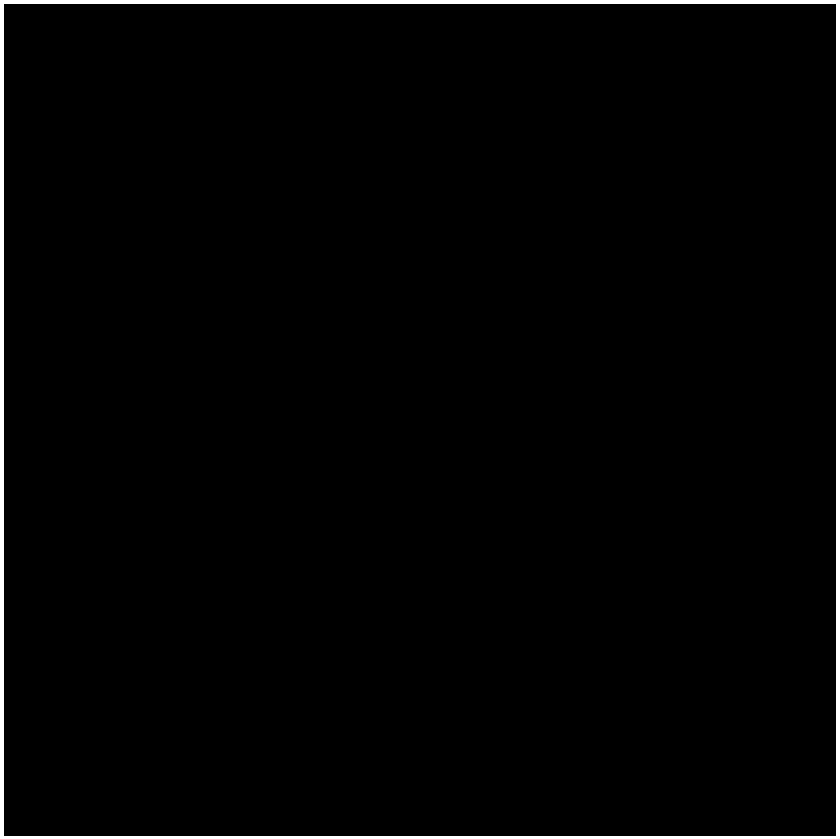
z-Buffering: Discarding pixels



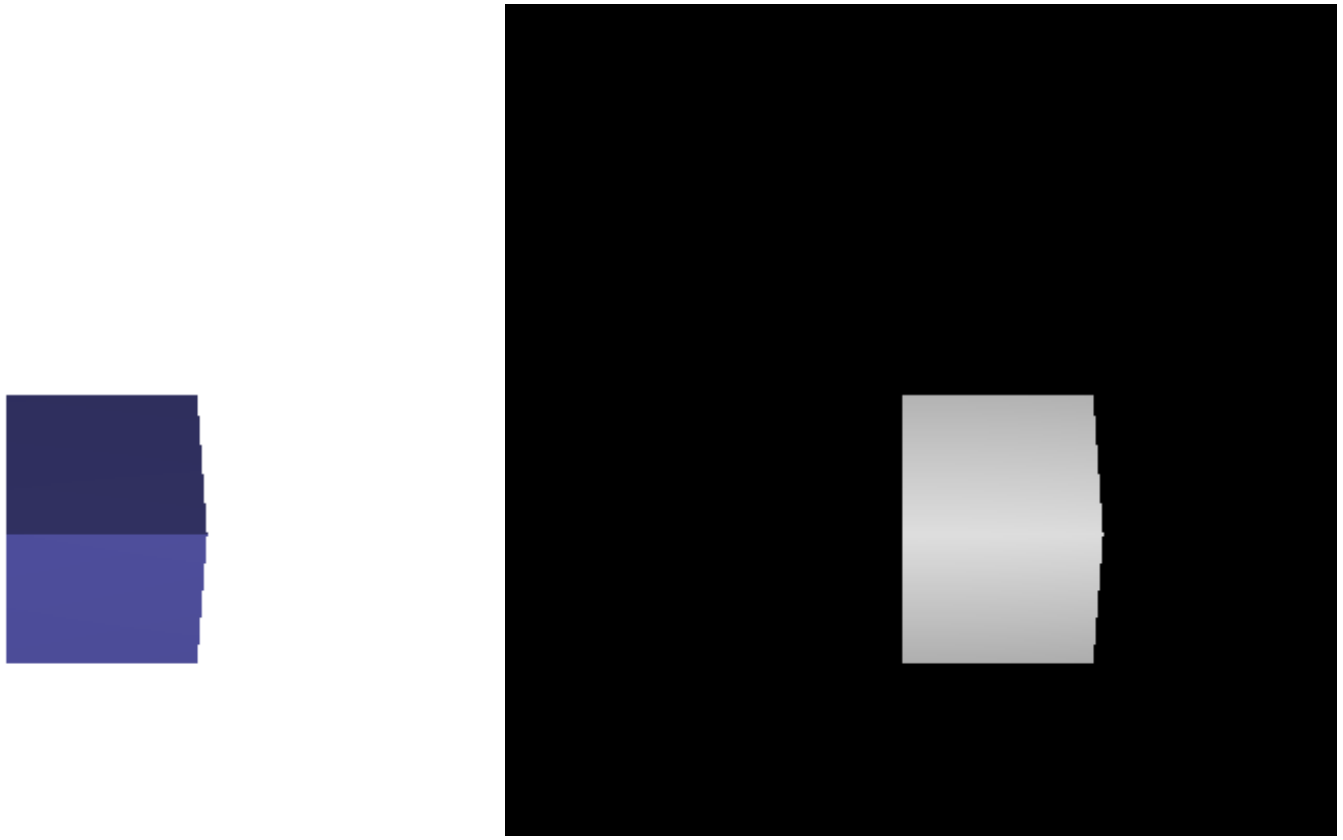
iStock

removing fragments (i.e., potential pixels) from the pipeline that will never be seen in the picture

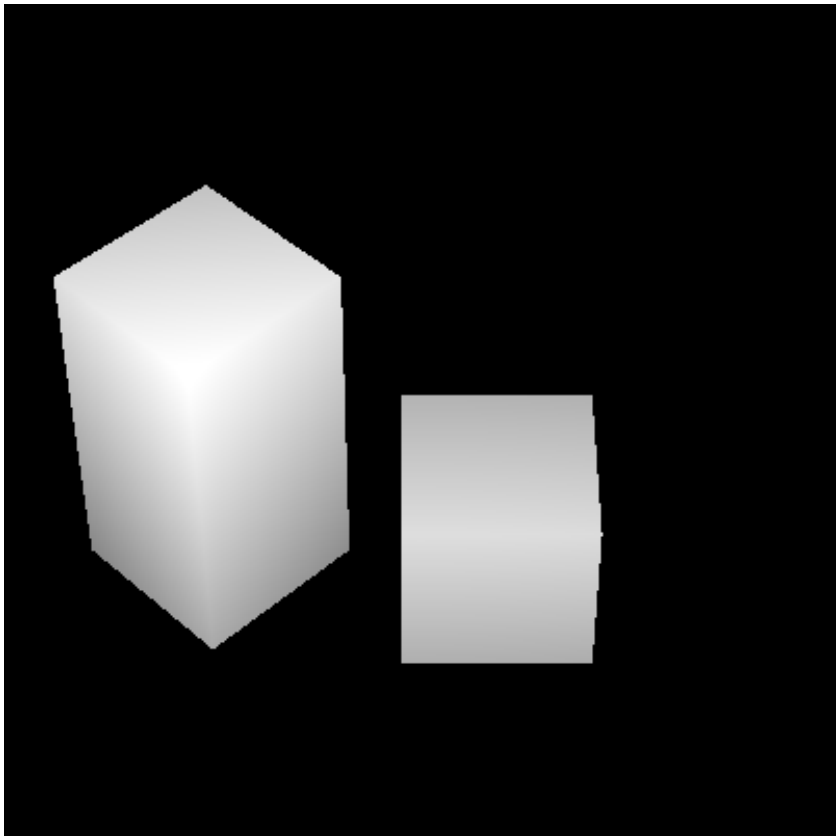
z-Buffering: Example (shown by object)



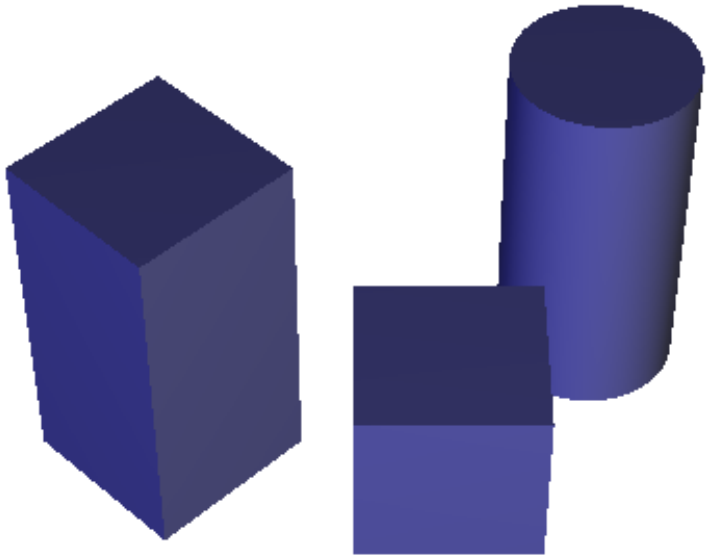
z-Buffering: Example (shown by object)

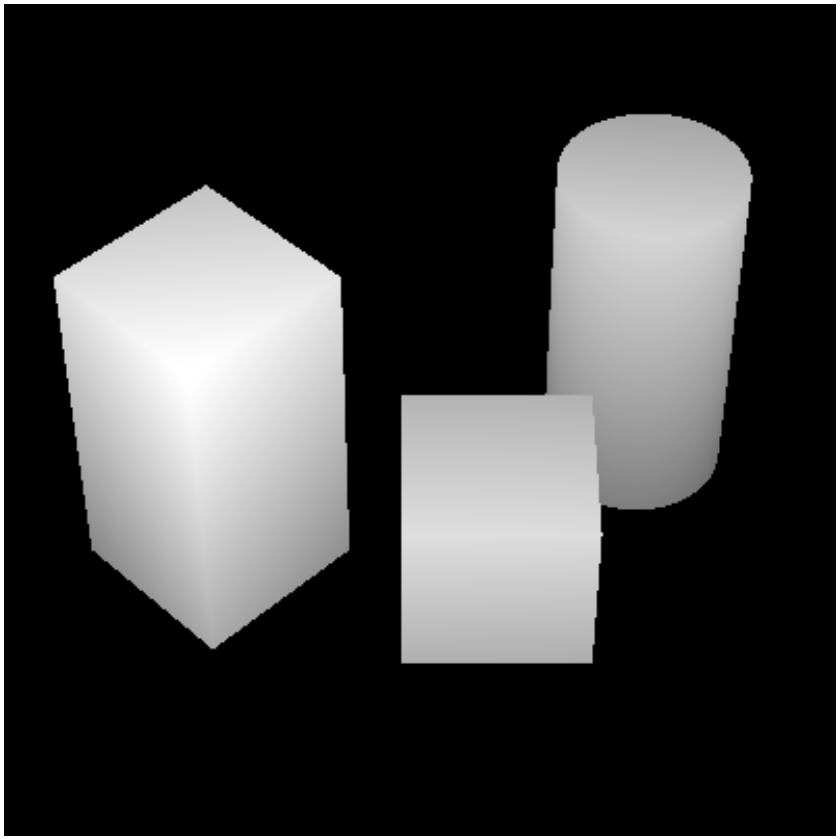


z-Buffering: Example (shown by object)

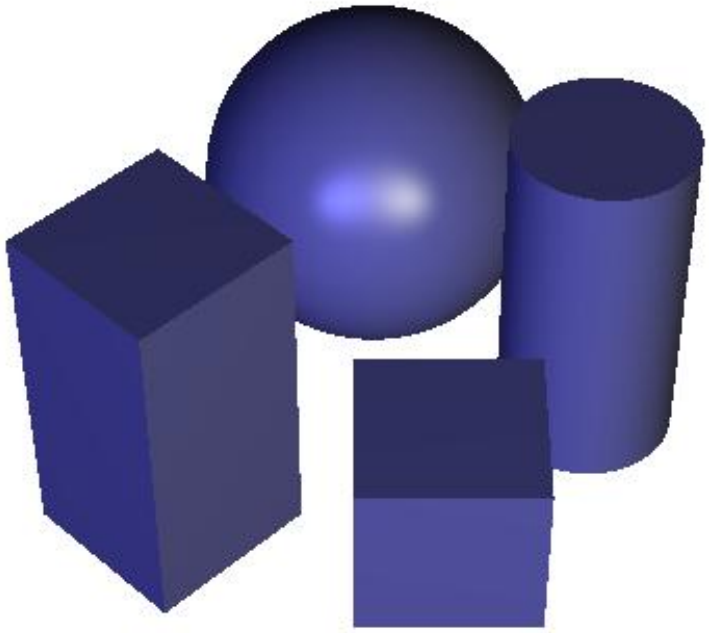


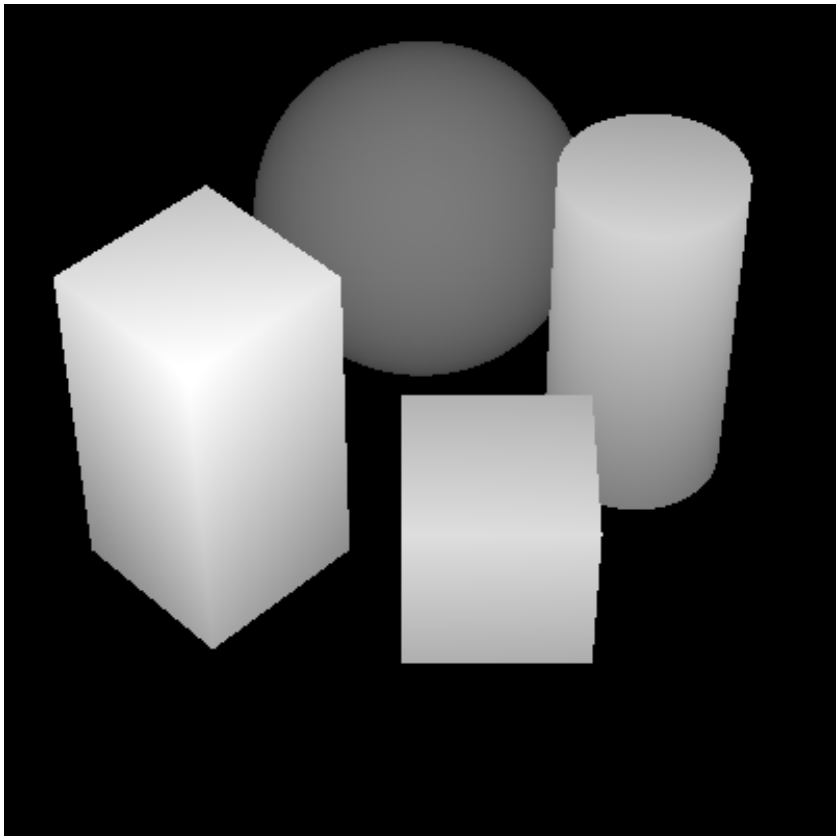
z-Buffering: Example (shown by object)



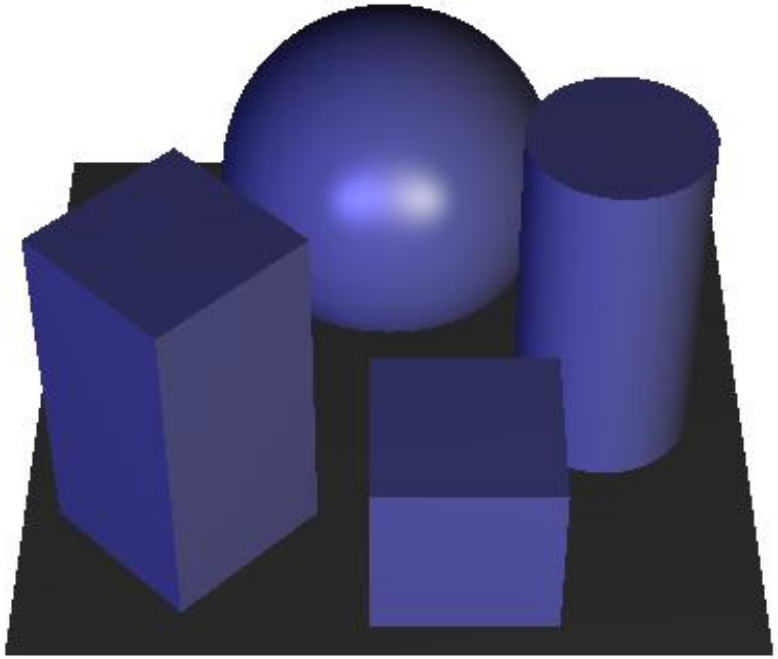


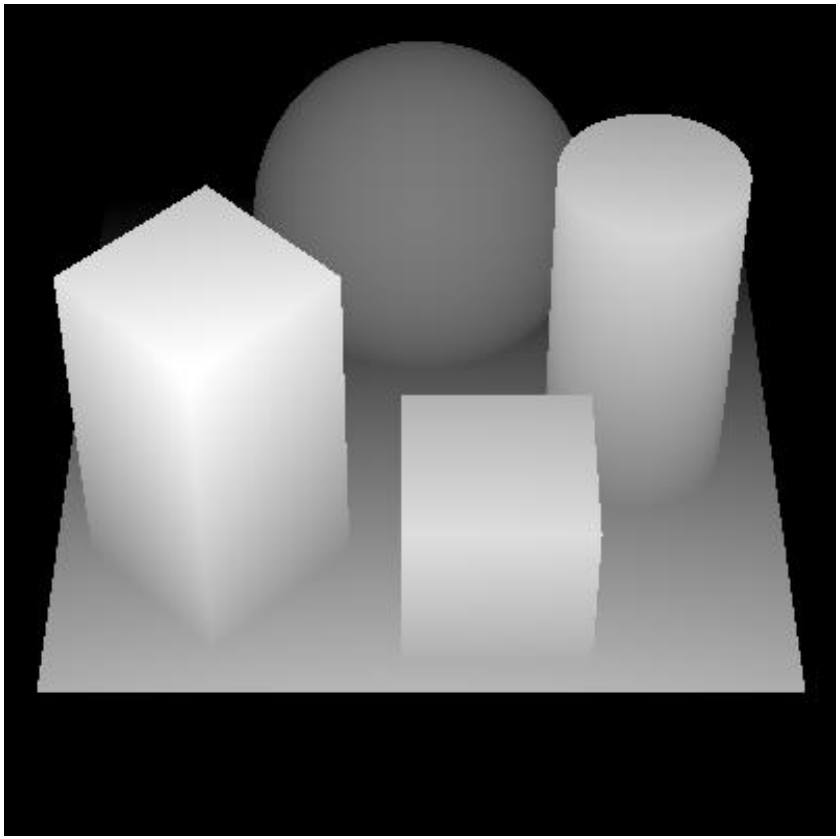
z-Buffering: Example (shown by object)





z-Buffering: Example (shown by object)





In reality the *z*-buffering treats the model triangle by triangle, not object by object!

z-Buffering

- advantages
  - can be implemented in hardware
  - can process infinitely many primitives
  - does not need sorting of primitives (only need to know distance to camera)
  - can handle cyclic and penetrating triangles
- disadvantages
  - needs memory to keep all *z*-values (image size @ 8bit or 16bit)
  - cannot handle transparency properly

Other HSR Algorithms

- hierarchical *z*-buffer
  - smaller versions in faster GPU memory (cache)
  - stores most distant *z*-depth
  - most fragments will be rejected anyway
  - faster simple rejection for many of them

Other HSR Algorithms

- hierarchical *z*-buffer
- Appel’s algorithm (for lines)
- Warnock's algorithm
- Octrees
- Binary Space Partitioning (BSP-trees)
- A-buffer (with anti-alising)
- etc.

Hidden Surface Removal – Summary

- needed to reduce rendering effort
- backface culling
  - remove all triangles that point away
  - not sufficient as HSR
- *z*-buffering
  - use of additional *z*-buffer
  - pixel-based technique
  - no polygon sorting needed
  - only pixel accuracy