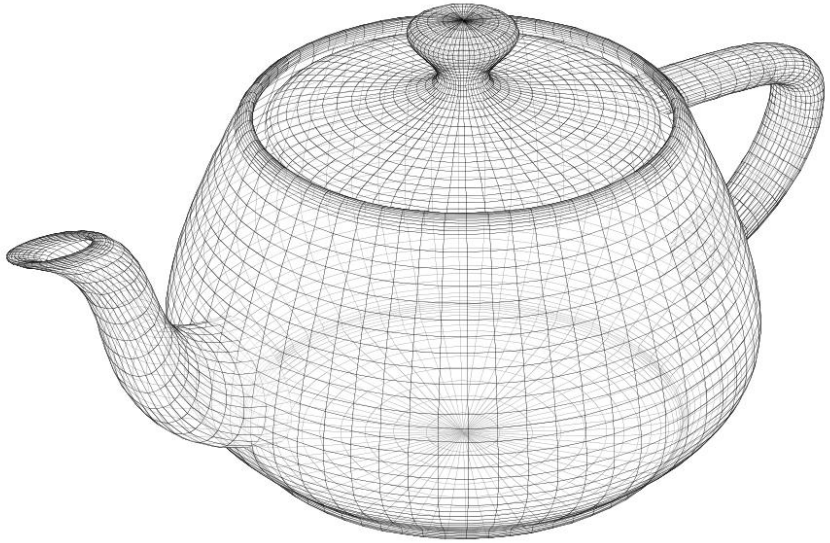


# Hidden Surface Removal

Interactive Computer Graphics

Eric Shaffer

ILLINOIS

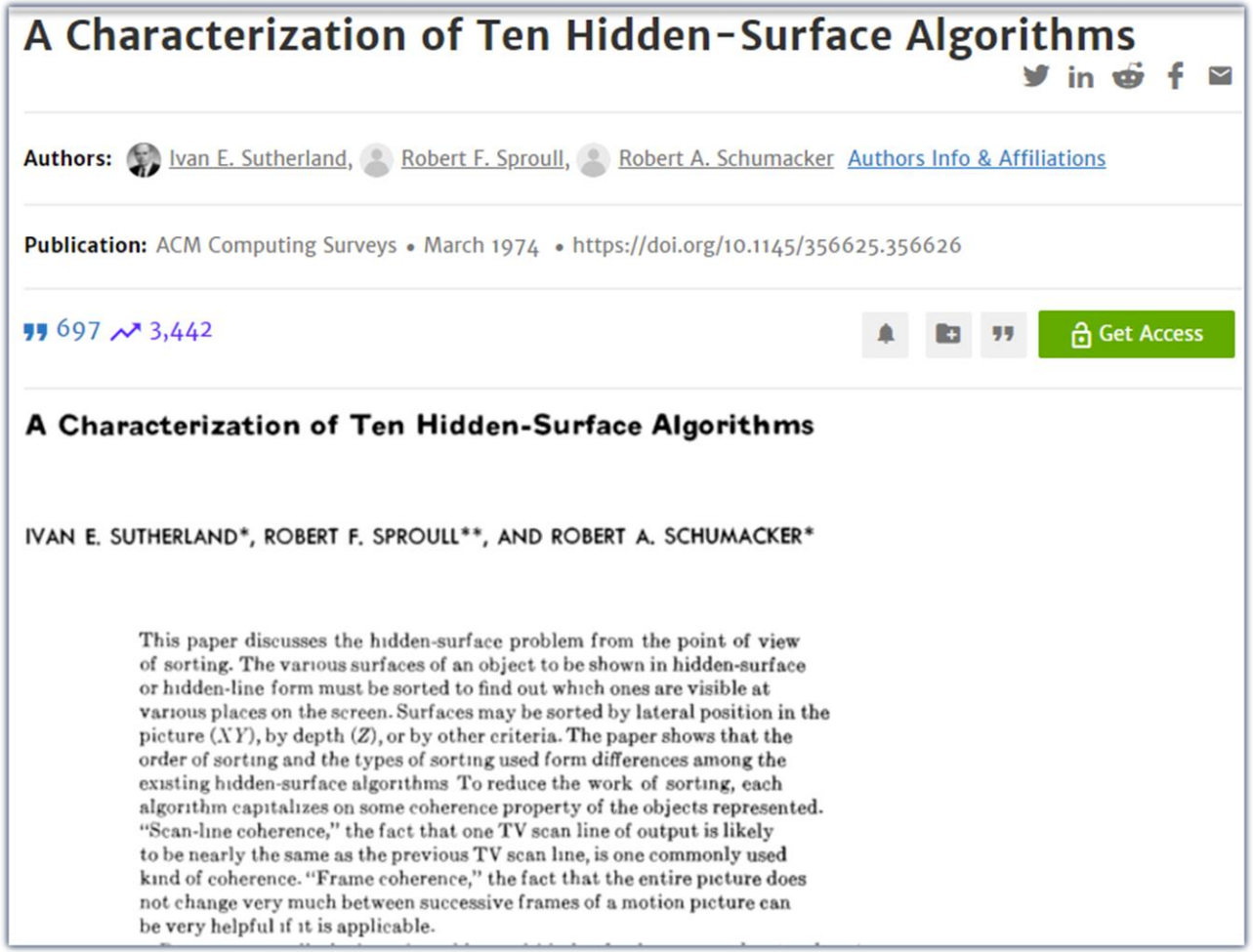
# Hidden Surface Removal

Simple problem statement:

- don’t render surfaces occluded by surfaces in front

Was a significant area of research in early days of CG

- lots of algorithms suggested



A Characterization of Ten Hidden-Surface Algorithms

Authors: Ivan E. Sutherland, Robert F. Sproull, Robert A. Schumacker Authors Info & Affiliations

Publication: ACM Computing Surveys • March 1974 • https://doi.org/10.1145/356625.356626

697 3,442 Get Access

A Characterization of Ten Hidden-Surface Algorithms

IVAN E. SUTHERLAND*, ROBERT F. SPROULL**, AND ROBERT A. SCHUMACKER*

This paper discusses the hidden-surface problem from the point of view of sorting. The various surfaces of an object to be shown in hidden-surface or hidden-line form must be sorted to find out which ones are visible at various places on the screen. Surfaces may be sorted by lateral position in the picture ($XY$), by depth ($Z$), or by other criteria. The paper shows that the order of sorting and the types of sorting used form differences among the existing hidden-surface algorithms To reduce the work of sorting, each algorithm capitalizes on some coherence property of the objects represented. "Scan-line coherence," the fact that one TV scan line of output is likely to be nearly the same as the previous TV scan line, is one commonly used kind of coherence. "Frame coherence," the fact that the entire picture does not change very much between successive frames of a motion picture can be very helpful if it is applicable.

ILLINOIS

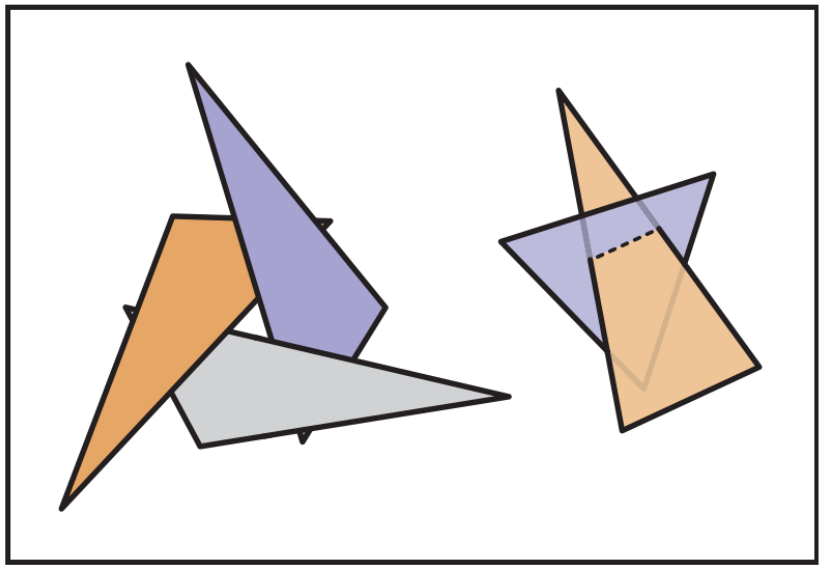
# Hidden Surface Removal

- Hidden Surface Removal
  - …don’t render surfaces occluded by surfaces in front of them
- Was a significant area of research in early days of CG
  - …lots of algorithms suggested
- Painter’s Algorithm
  - Render objects in order from back to front
    - i.e. sort your triangles by depth and render deepest first
  - Can anyone imagine any problems with this approach?

ILLINOIS

# Problems with the Painter's Algorithm

- No correct rendering order for
  - intersecting triangle
  - occlusion cycles



- Sorting is slow…too slow for interactivity in complex scenes

ILLINOIS

# Hidden Surface Removal: Z-Buffer

Key Observation:
Each pixel displays color of only one triangle,
Ignores everything behind it

Don't need to sort triangles, just find, for each pixel, the closest triangle

Z-buffer: one fixed or floating point value per pixel

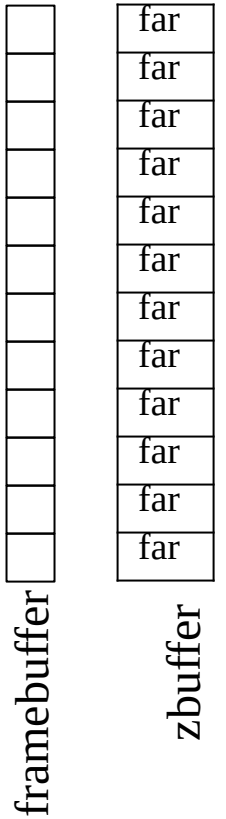
Algorithm:

For each rasterized fragment (x,y)

If z < zbuffer(x,y) then

framebuffer(x,y) = fragment color

zbuffer(x,y) = z




Frame Buffer: buffer that stores the colors for the pixels we will render

ILLINOIS

# Z-Buffer

Key Observation:
Each pixel displays color of only one triangle, ignores everything behind it

Don't need to sort triangles,
just find for each pixel the closest triangle

Z-buffer: one fixed or floating point value per pixel

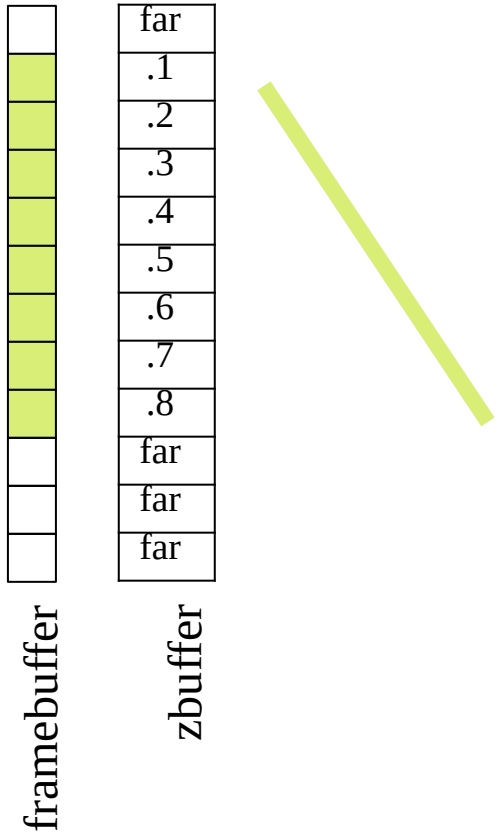
Algorithm:

For each rasterized fragment (x,y)

If z < zbuffer(x,y) then

framebuffer(x,y) = fragment color

zbuffer(x,y) = z




ILLINOIS

# Z-Buffer

Key Observation:
Each pixel displays color of only one triangle, ignores everything behind it

Don't need to sort triangles,
just find for each pixel the closest triangle

Z-buffer: one fixed or floating point value per pixel

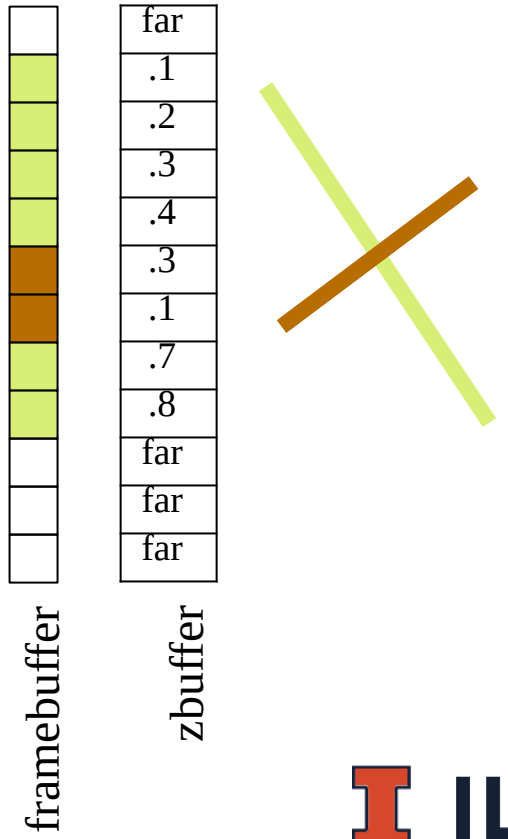
Algorithm:

For each rasterized fragment (x,y)

If z < zbuffer(x,y) then

framebuffer(x,y) = fragment color

zbuffer(x,y) = z



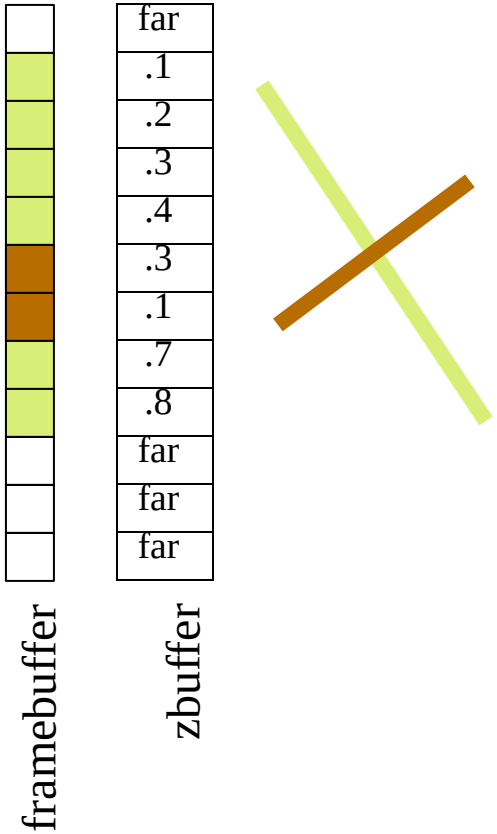

ILLINOIS

# Z-Buffer

Get fragment z-values by interpolating z-values at vertices during rasterization

True perspective projection destroys z-values, setting them all to *–d*

The perspective distortion we use preserves at least the ordering of z-values



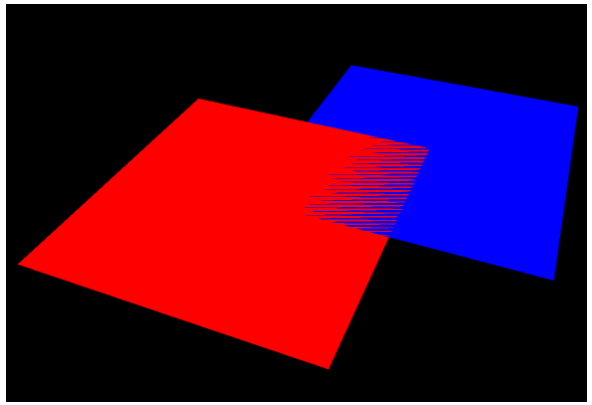

ILLINOIS

# Precision Issues with Z-Buffering

- In practice, depths values are typically converted to non-negative integers
  - Comparison operation needs to be fast…
- Imagine having depth values of {0,1,…,B-1}
  - 0 → near clipping plane distance
  - B-1 → far clipping plane distance
- Depths occur discretely in “buckets”
  - Each bucket covers a range of length $\Delta z = \frac{f-n}{B}$
- If we use b bits for the z-buffer values, B = $2^b$
  - You usually can’t change the value b
  - To maximize z-buffer effectiveness, **need to minimize f-n**

ILLINOIS

# Z-Fighting




How can you fix z-fighting?

1. Move co-planar polygons slightly away from each other
2. Move near and far clipping planes as close together as you feasibly can

ILLINOIS

# Actually, It's Worse Than That….

Post-projection depth value does NOT vary linearly with input depth $p_z$

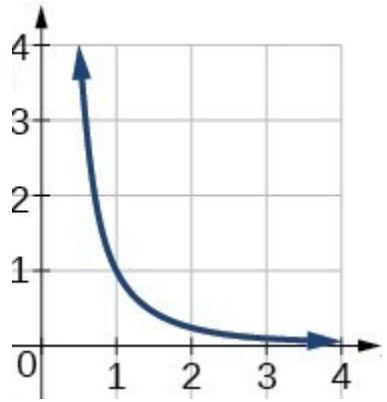
$$\mathbf{P}_{\text{OpenGL}} = \begin{pmatrix} \frac{2n'}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n'}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f'+n'}{f'-n'} & -\frac{2f'n'}{f'-n'} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$\mathbf{v} = \mathbf{Pp} = \begin{pmatrix} \cdots \\ \cdots \\ dp_z + e \\ \pm p_z \end{pmatrix}$$

$$d = -(f'+n')/(f'-n')$$

$$e = -2f'n'/(f'-n')$$

**Output depth is inversely proportional to the input depth**

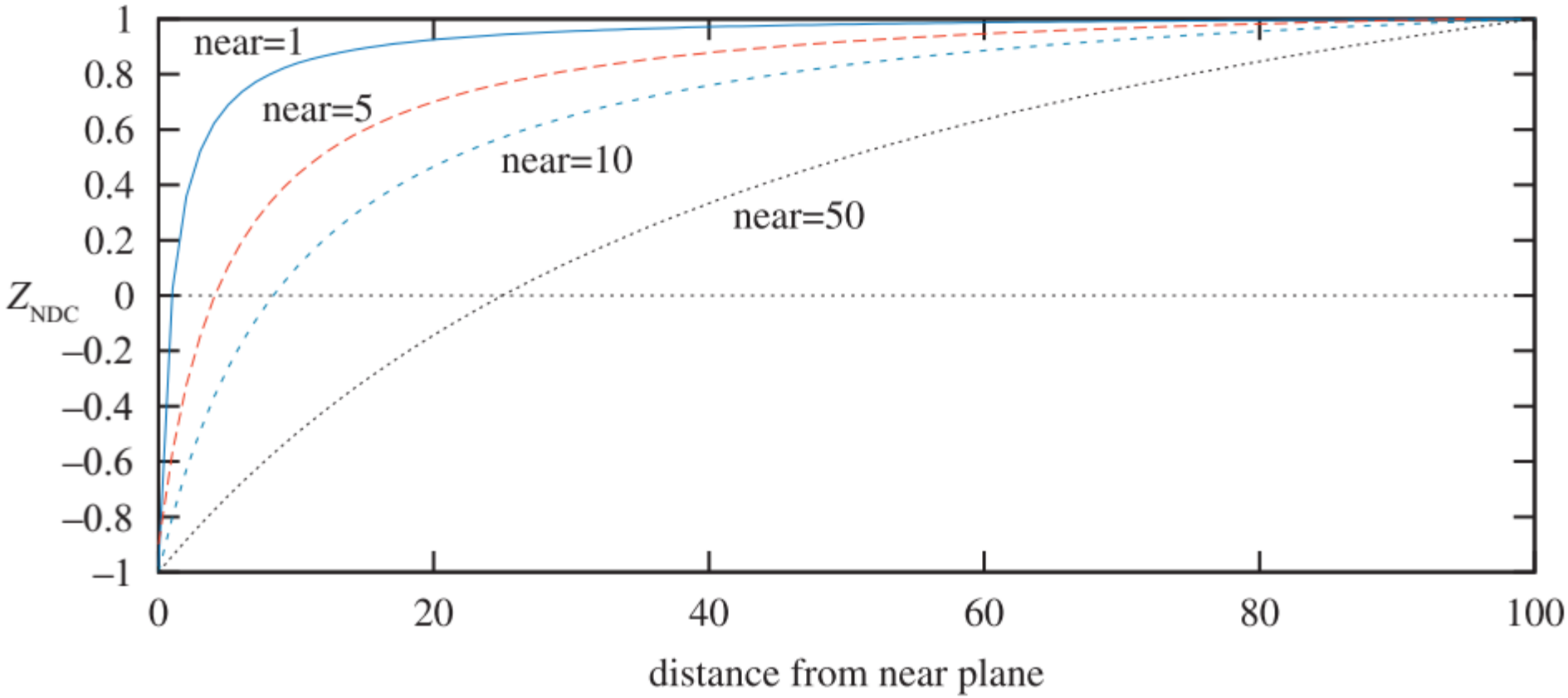



$$z_{\text{NDC}} = \frac{dp_z + e}{-p_z} = d - \frac{e}{p_z}$$

$$z_{\text{NDC}} \in [-1, +1]$$

ILLINOIS

# …More Precision Issues with Z-Buffering




The effect of varying the distance of the near plane from the origin. The distance f−n is kept constant at 100. As the near plane becomes closer to the origin, points nearer the far plane use a smaller range of the normalized device coordinate (NDC) depth space. This has the effect of making the z-buffer less accurate at greater distances.

*Real-Time Rendering*, Fourth Edition

ILLINOIS

# WebGL Hidden Surface Removal

At Startup:

```
gl.enable(gl.DEPTH_TEST);    // use depth test for hidden surface remove
gl.depthFunc(gl.LESS);       //this is the default
```

Each frame:

```
gl.clear(gl.DEPTH_BUFFER_BIT); // clear depth values form previous frame
```

Hidden surface removal uses the depth buffer (z-buffer)

Happens after the fragment shader

ILLINOIS