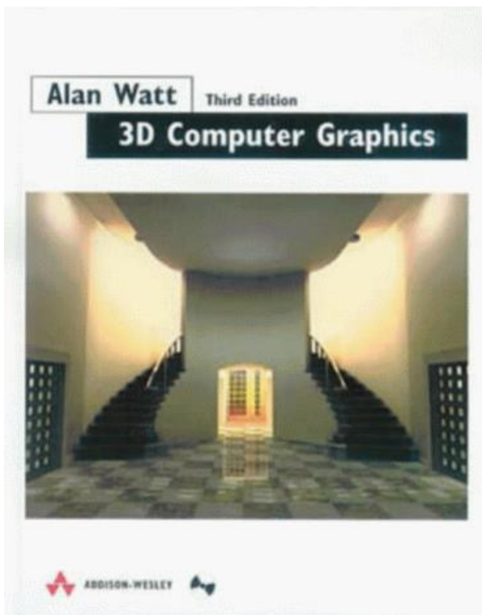
The University Of Sheffield.

COM3503/4503/6503: 3D Computer Graphics

Lecture 8: Rendering: Part 2

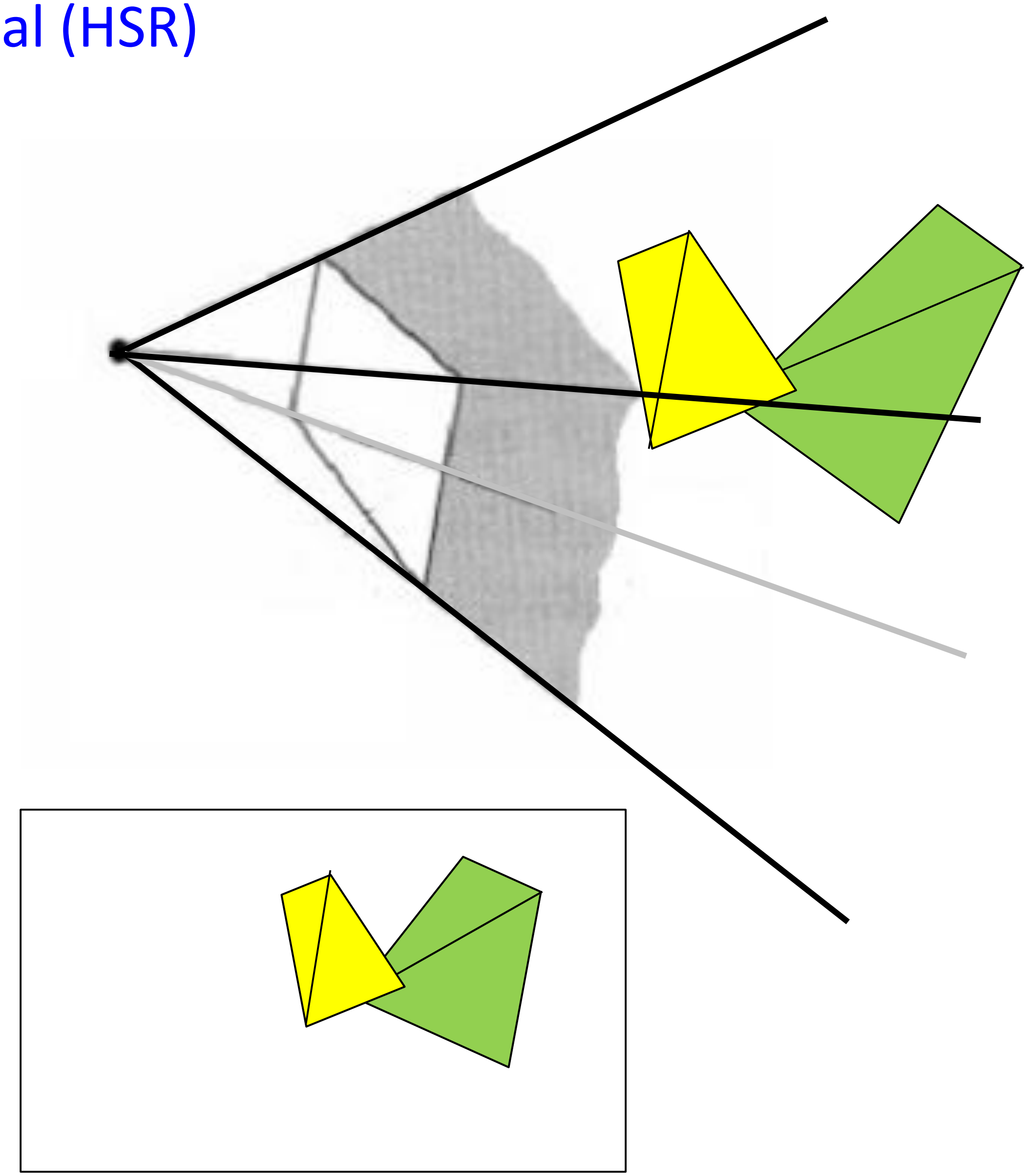


Dr. Steve Maddock
s.maddock@sheffield.ac.uk

1. Hidden Surface Removal (HSR)

- Input: a set of polygons and a viewpoint in world space

- Output: A 2D image of projected polygons, containing only visible portions



1. Hidden Surface Removal (HSR)

Alternatives

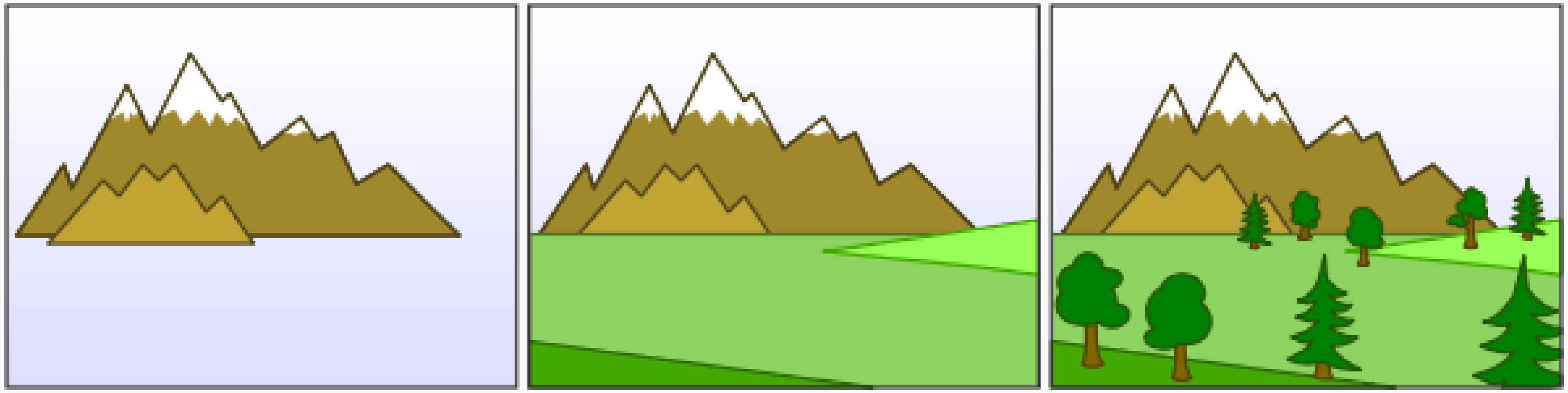
- Z-buffer (Catmull, 1975)
  - Most popular
  - Eliminated the others as memory prices decreased
- Scan-line based (y,x,z) (e.g. Watkins, 1970)
- Area subdivision (e.g. Warnock, 1969)
- Depth List or depth priority schemes (z, x or y), (e.g. Newell, Newell and Sancha, 1972) (generalisation of Painter's algorithm)

Catmull's PhD thesis:
http://www.pixartouchbook.com/blog/2009/1/4/ed-catmulls-phd-thesis.html

See Watt's book for references

Wolfgang Straßer also described this idea in his 1974 Ph.D. thesis. Strasser, W.: Schnelle Kurven- und Flaechendarstellung auf graphischen Sichtgeraeten),
Ph.D. thesis, TU Berlin 1974 – see W.K. Giloi, J.L. Encarnação, W. Straßer. "The 'Giloi's School' of Computer Graphics". Computer Graphics 35 4:12–16.

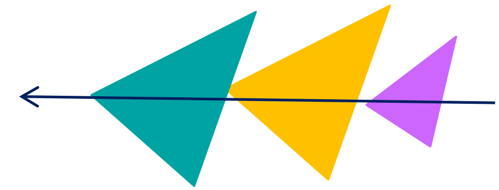
2. The Painter’s algorithm

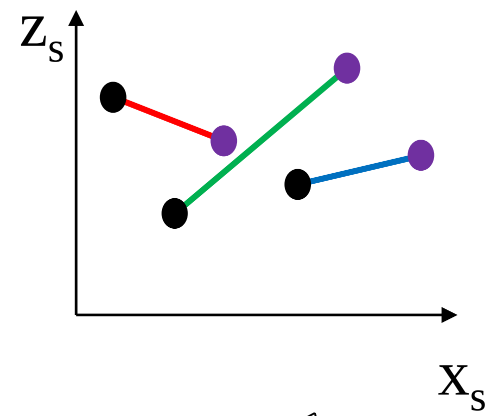


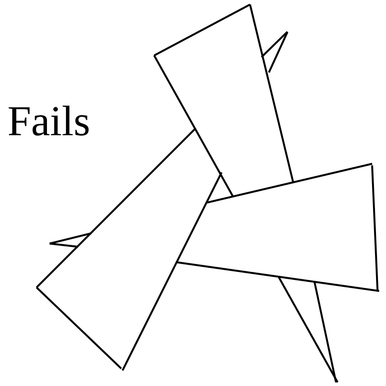
The distant mountains are painted first, followed by the closer meadows; finally, the trees, are painted. Although some trees are more distant from the viewpoint than some parts of the meadows, the ordering (mountains, meadows, trees) forms a valid depth order, because no object in the ordering obscures any part of a later object.

2. The Painter's algorithm

- An object-space (world space) algorithm
- Sort polygons in order of increasing z and then render in reverse order
    - (cf. 2D windows in a GUI)
- Requires efficient sorting algorithm
- Which point on the polygon to choose to order in z?
- Fails for intersecting polygons and mutually occluding polygons
- Can be solved (expensively) by splitting of polygons (Newell et al, 72)
    - Use plane of a polygon to split other polygons in two - leads to idea of BSP tree



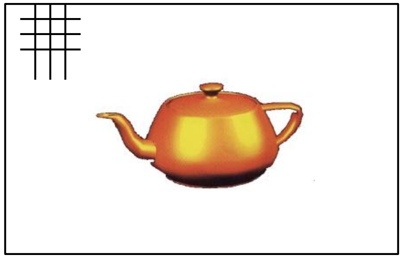




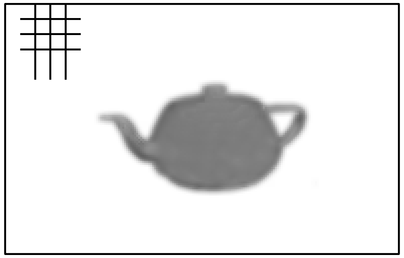
3. Z-Buffer

- An image/screen space algorithm
- OLD: Most popular combination for fixed-function pipeline: Phong reflection model, Gouraud interpolation, Z-buffer
- Modern: Programmable pipeline: Phong, Phong, Z-buffer
- Ubiquitous in hardware
- Requires a colour buffer for the final screen image and a Z-buffer to store a depth for each pixel

Screen / monitor



Framebuffer / screen buffer / colour buffer

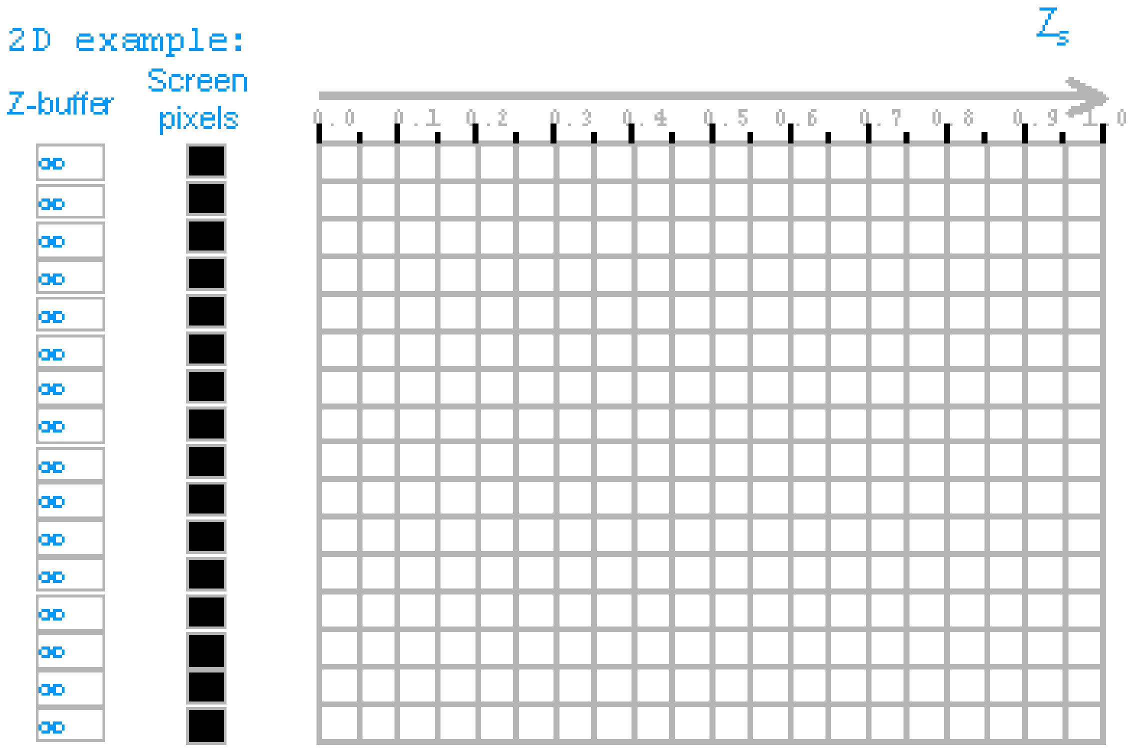


Z buffer (map depth to greyscale)

4. Imagine a 2D world

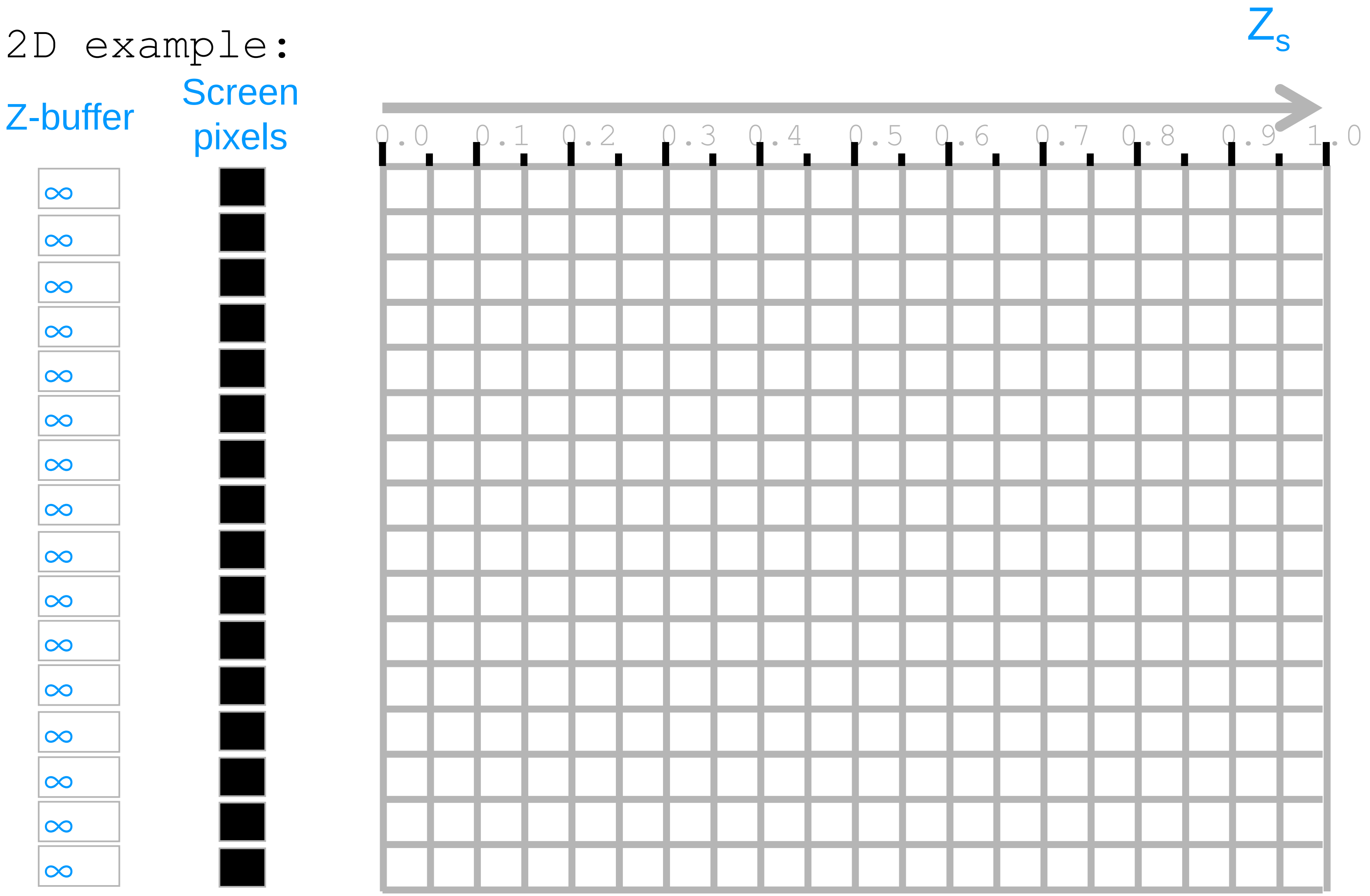
- Z-buffer is initialised with a z equal to the furthest possible point.

```
gl.glClearColor(0, 0, 0, 1);
gl.glEnable(GL.GL_DEPTH_TEST);
gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
```

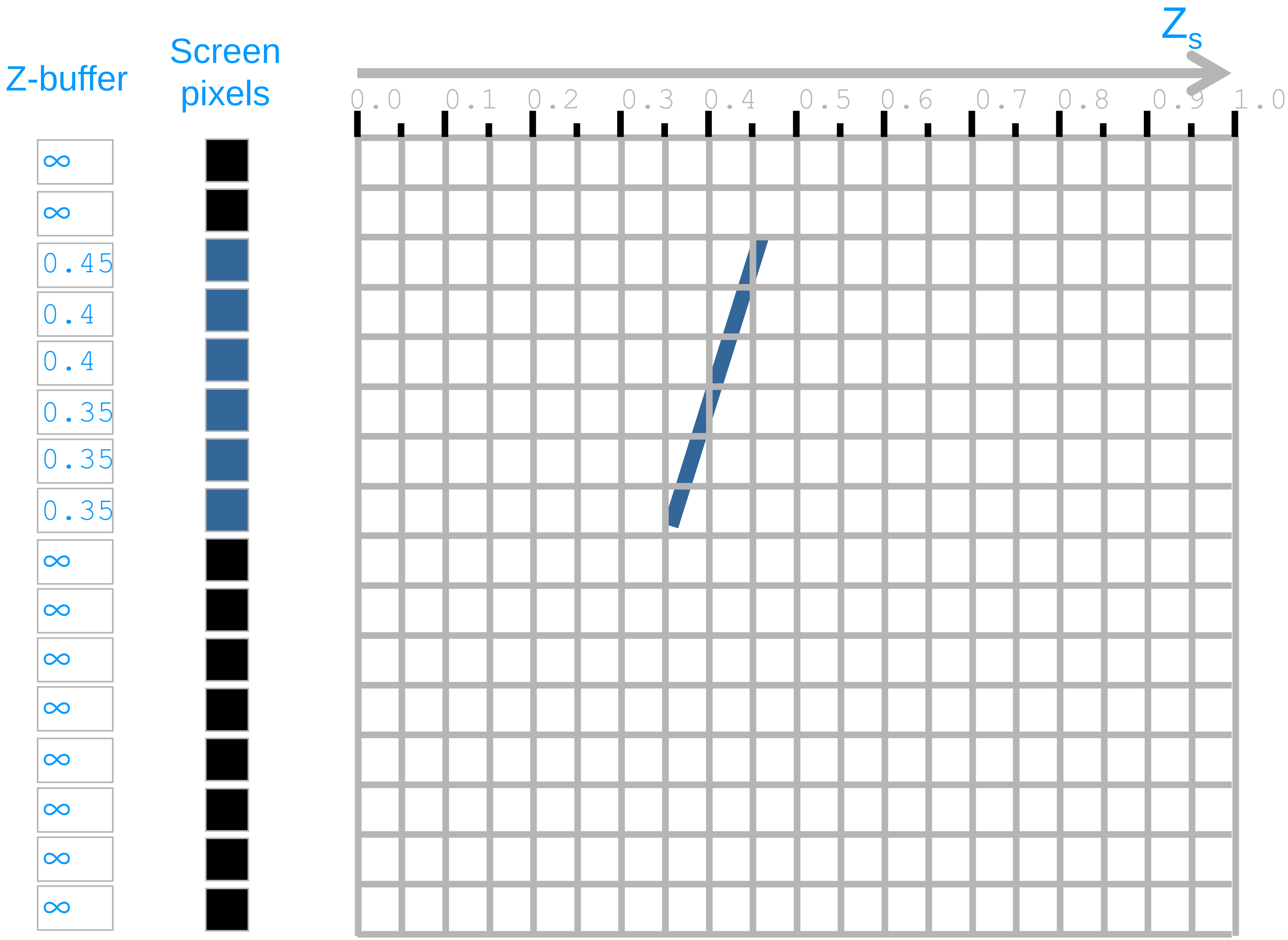


Z-Buffer – initialisation [Imaginary 2D world]

- `2D example:`

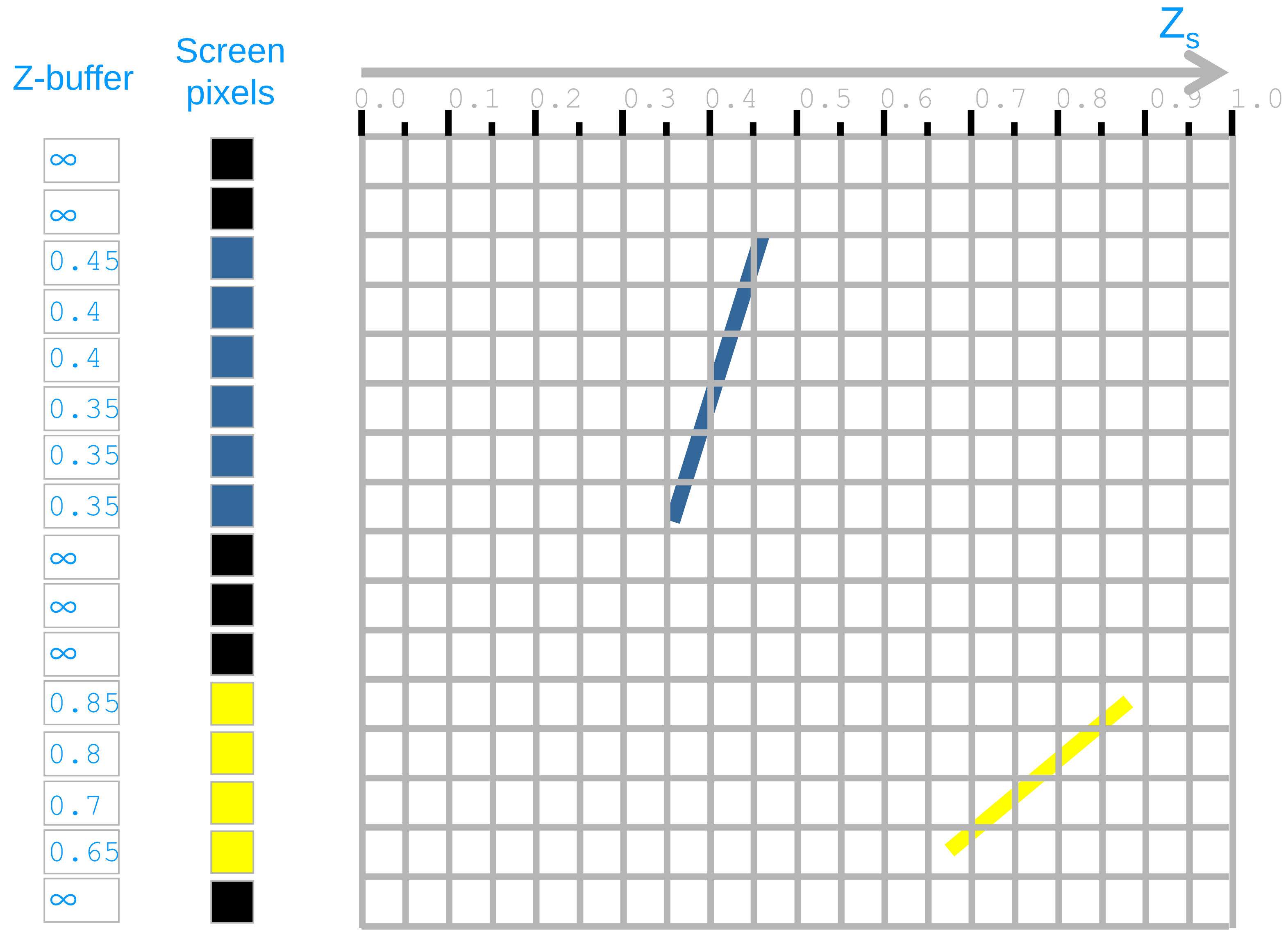


Draw a line

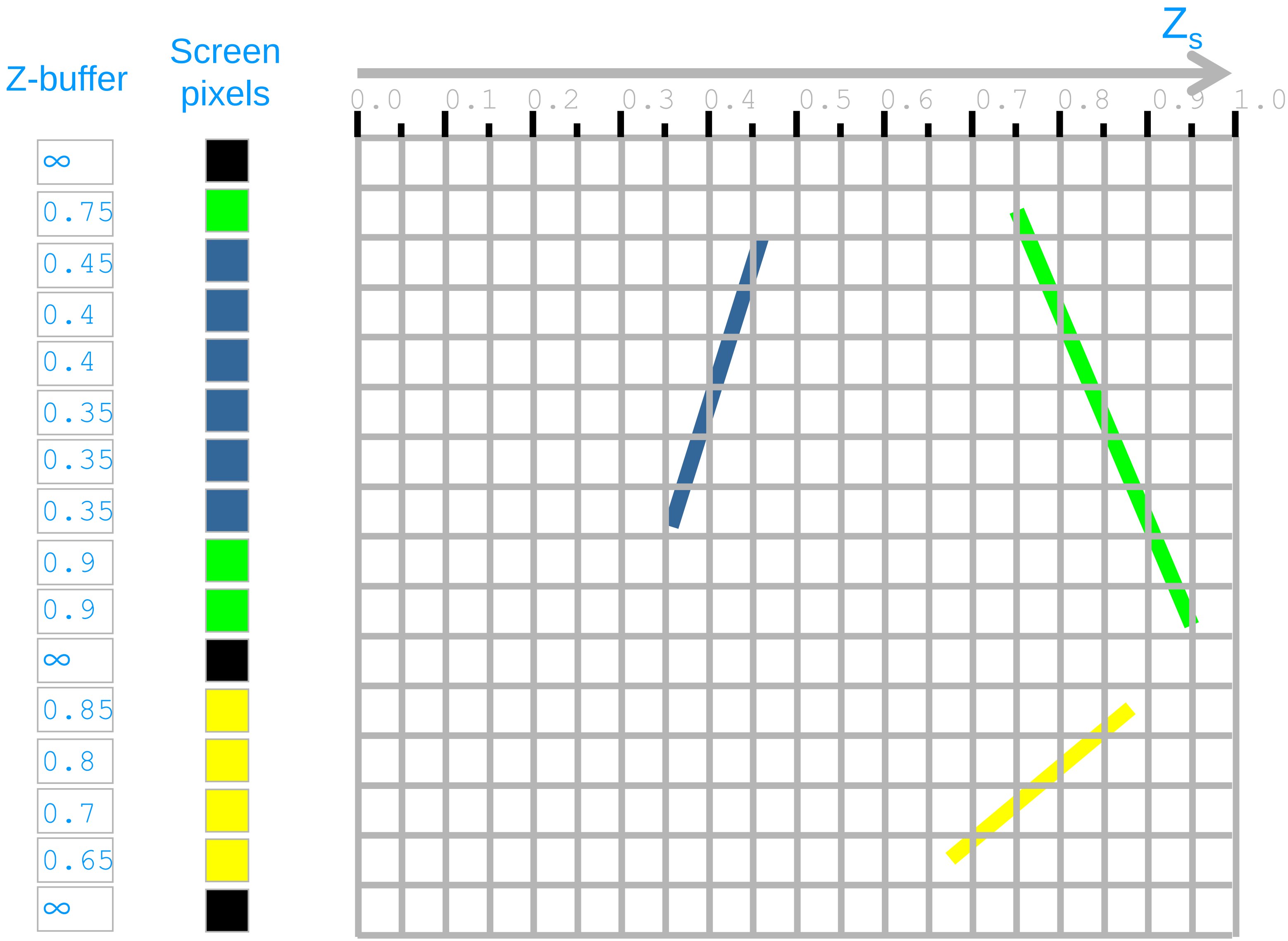


Z-buffer
Screen pixels
Z_s
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
∞
∞
0.45
0.4
0.4
0.35
0.35
0.35
∞
∞
∞
∞
∞
∞
∞
∞

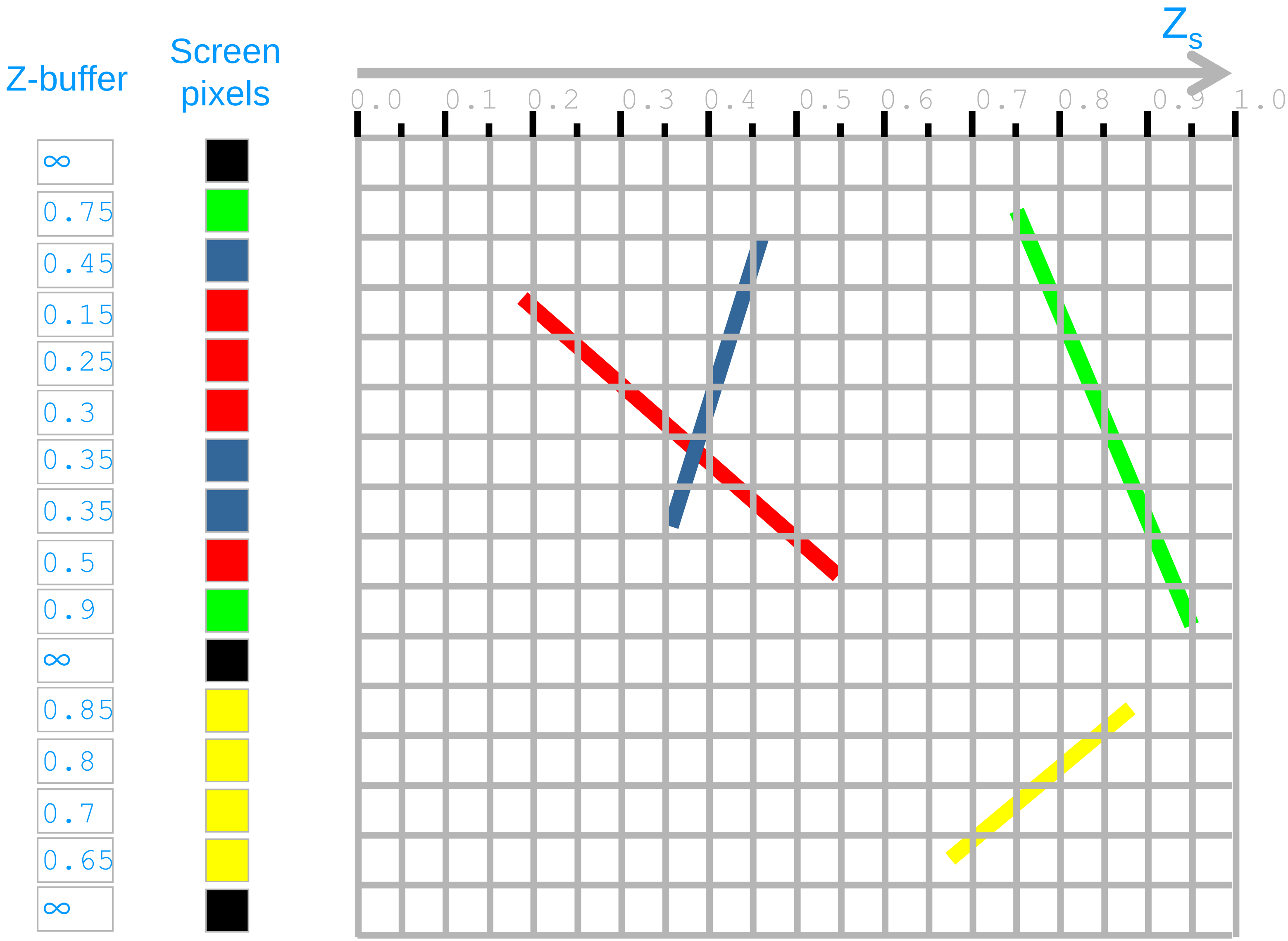
Draw another line



And another line



And another line

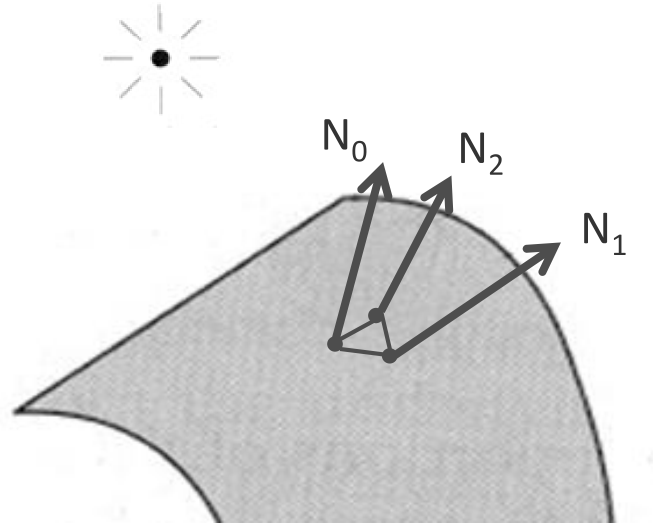


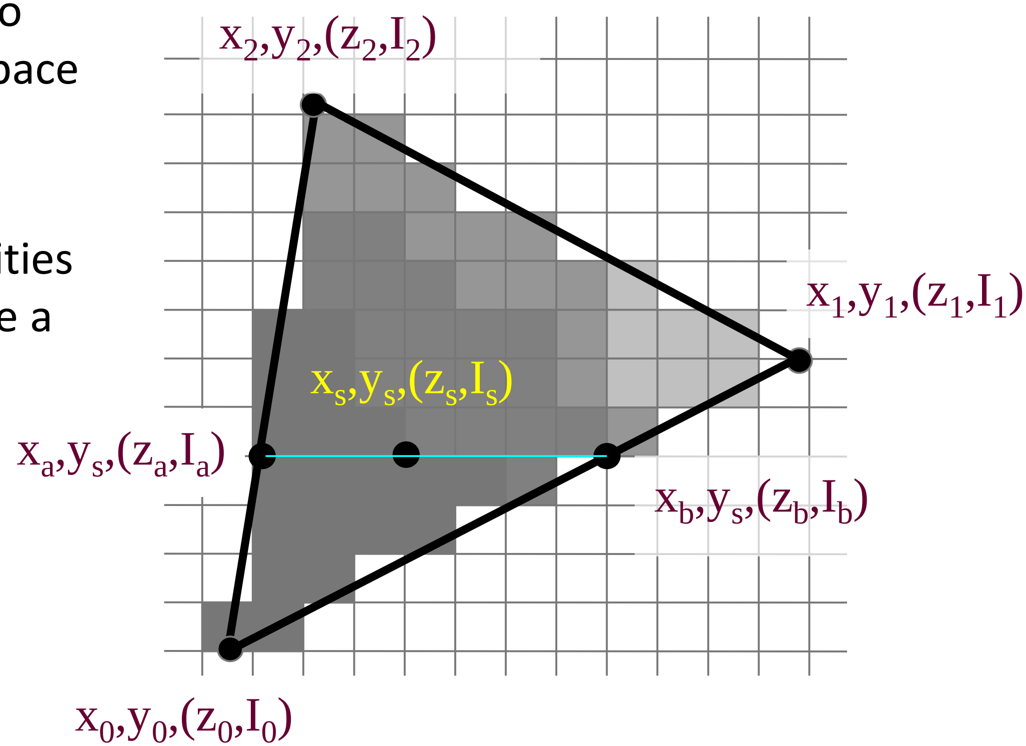
5. 3D

Last lecture we looked at Gouraud and Phong interpolative shading

Gouraud interpolative shading:

- For a triangle, apply local reflection model at vertices to determine intensity (world space calculation)
- Rasterisation: Bilinearly interpolate the vertex intensities across the triangle to produce a value for each 3D screen fragment
- Use the fragments to colour pixels in 2D screen space

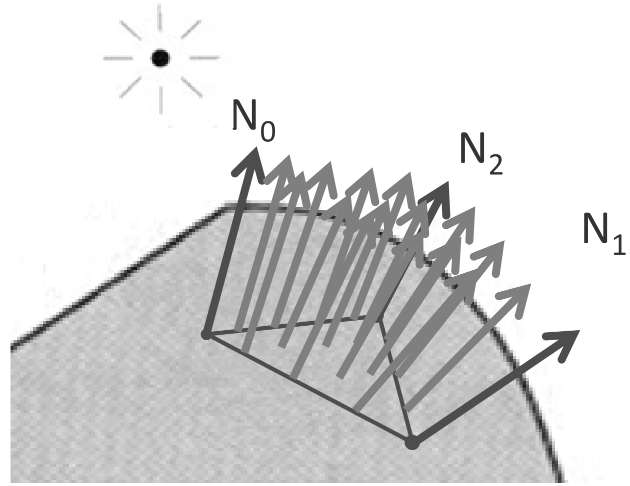


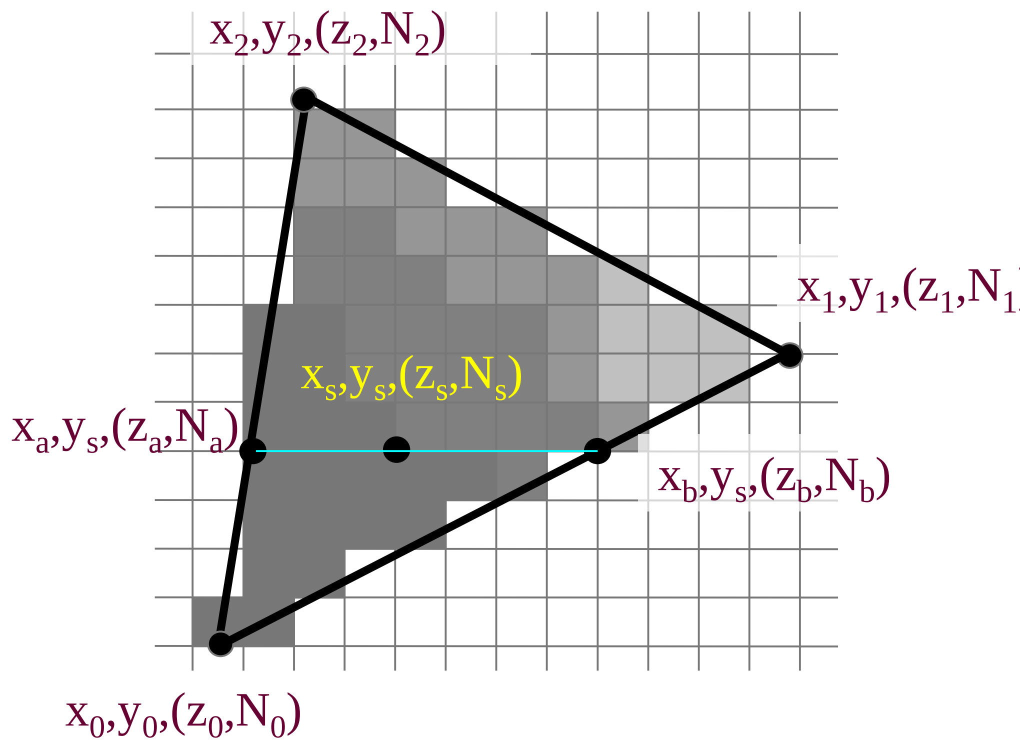


5. 3D

Phong interpolative shading:

- Each vertex has a normal (N_i)
- Rasterisation: Bilinearly interpolate the normals across the triangle to produce a normal per fragment
- Interpolation process is driven from screen space, but the normal is in world space
- Use the interpolated (normalized) normal (in world space) to calculate intensity for a fragment (in 3D screen space)
- Use the fragment to colour a pixel in 2D screen space



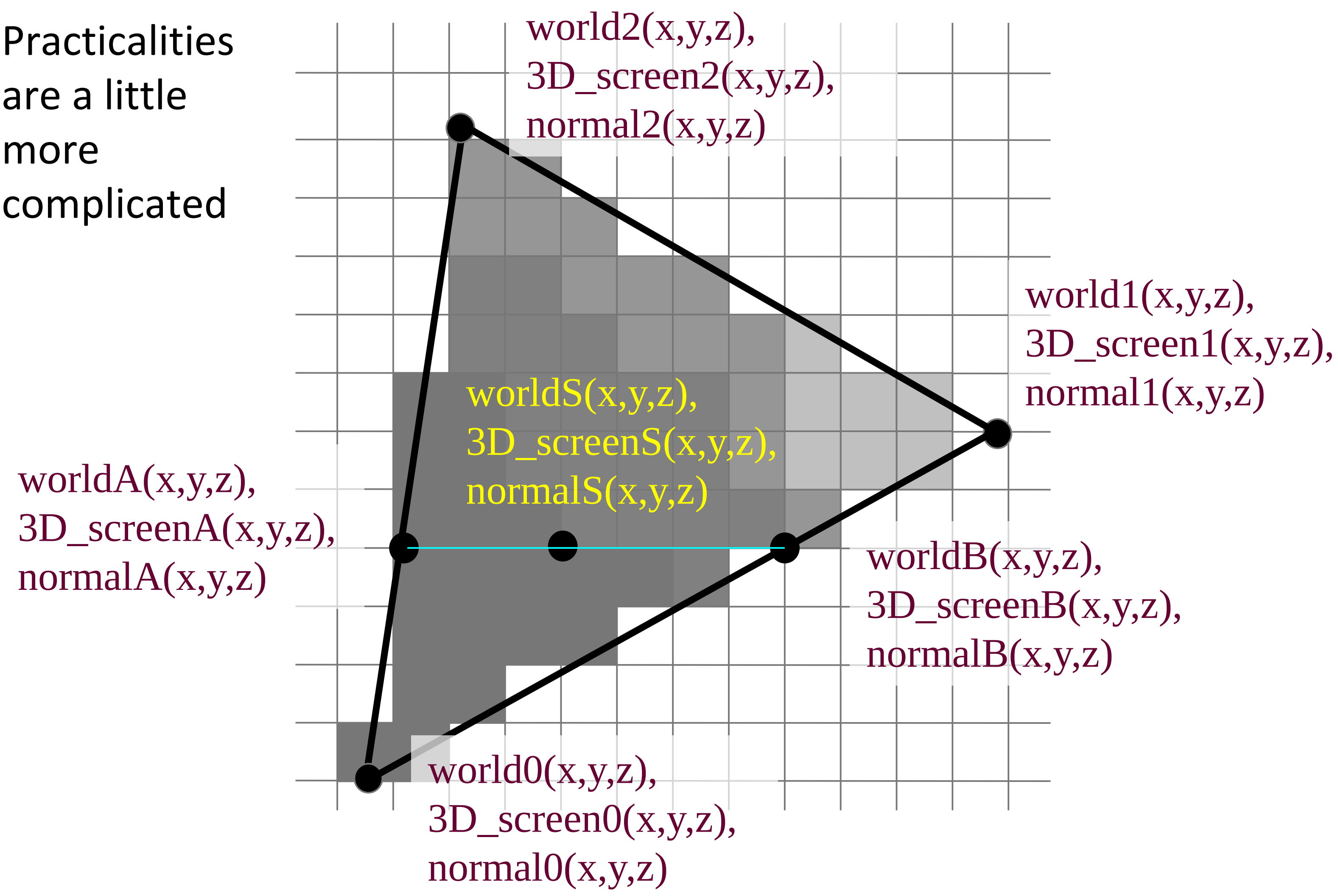


World space position = model_matrix * vertex_position_in_local_space

World space normal = normal_matrix * normal_in_vertex_local_space

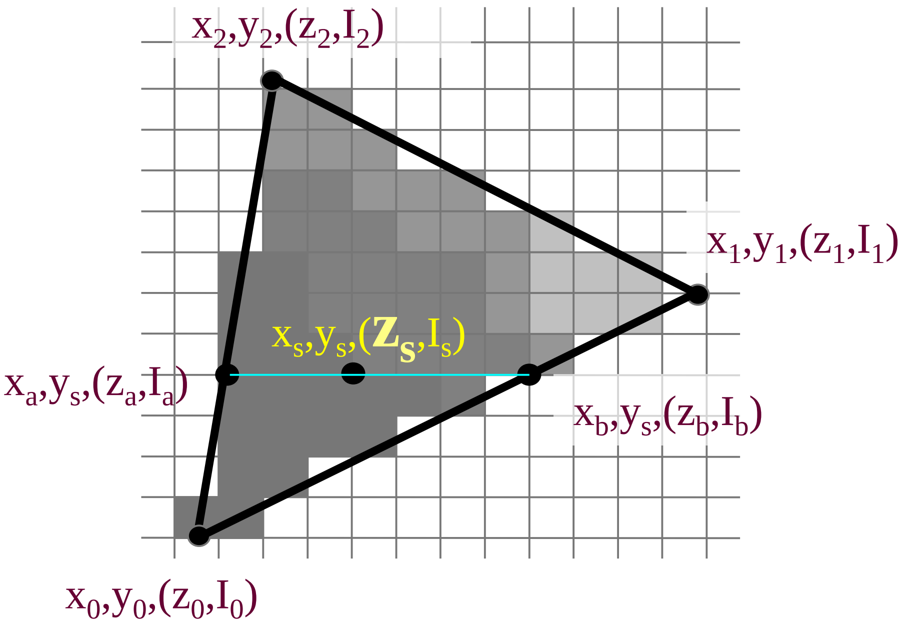
3D screen space position = projection_matrix * view_matrix * model_matrix * vertex_position_in_local_space

- Practicalities are a little more complicated



6. 3D HSR: Z buffer

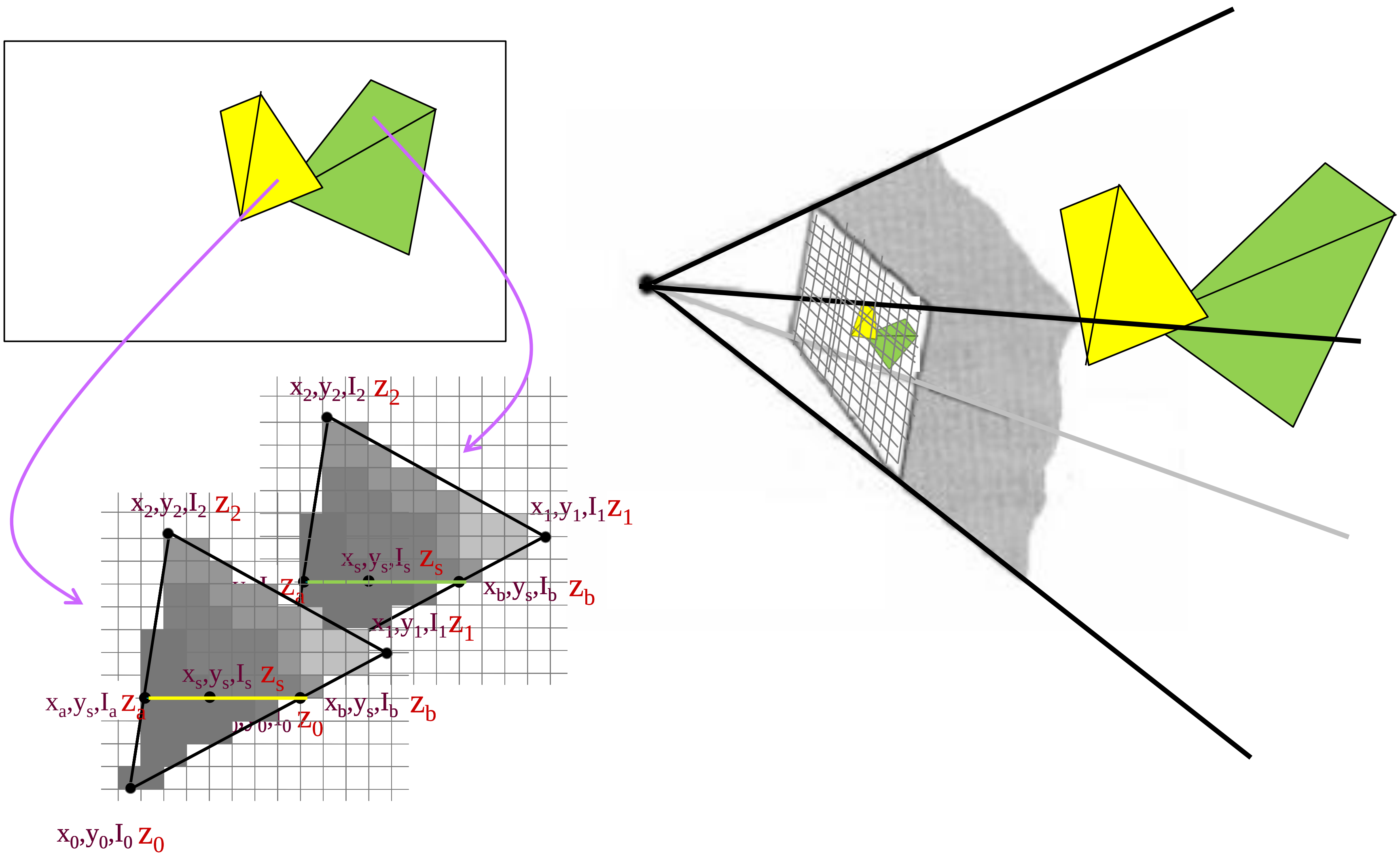
- Whether Gouraud or Phong interpolative shading, an important part in each case is that we have a 3D screen space, a **z coordinate for each fragment**



- For each pixel (x_s,y_s) in the polygon:

```
IF zs < zbuffer[x,y] THEN
  write intensity to framebuffer[x,y];
  write zs to zbuffer[x,y];
END;
```

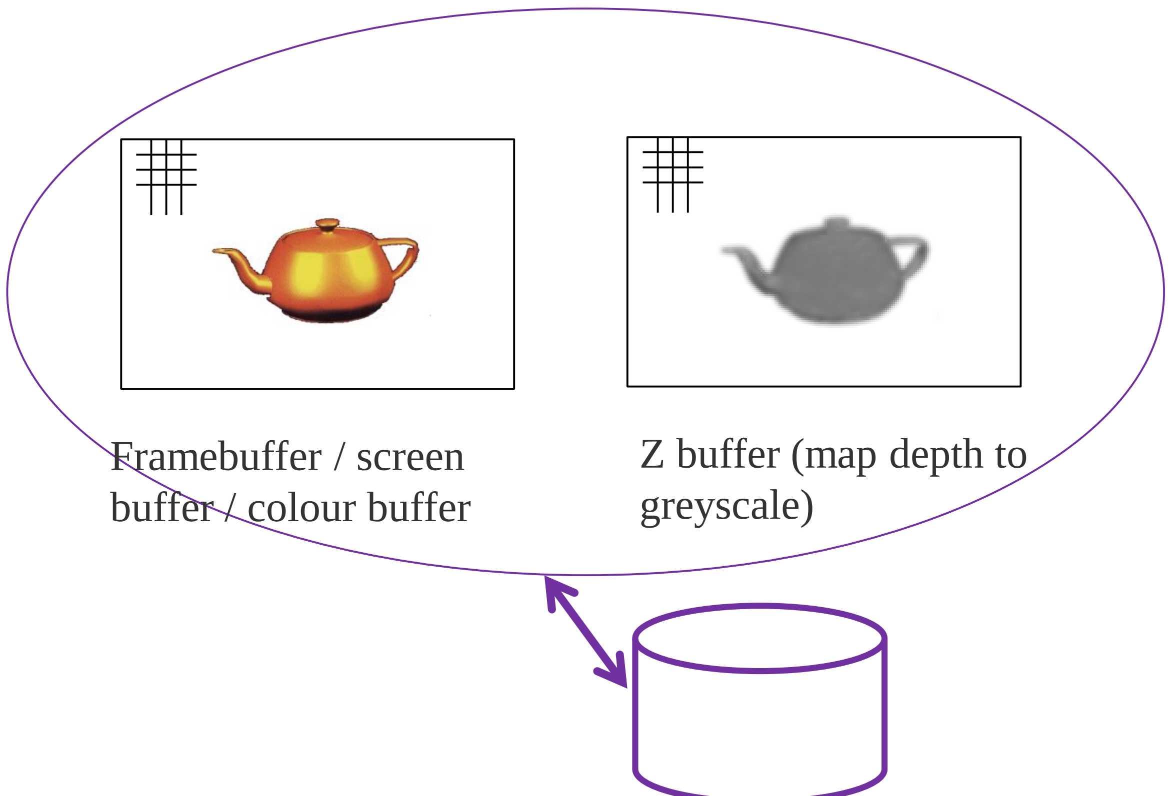
6.1 Illustrating the Z buffer in 3D



6.2 Advantages

- Simplicity – single if test.
- No upper limit on scene complexity
  - Polygon-by-polygon (in any order);
- Can save state of z-buffer and screen buffer and render more at a later time

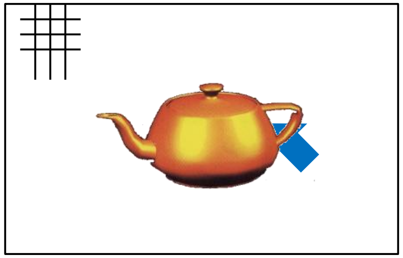
Screen / monitor



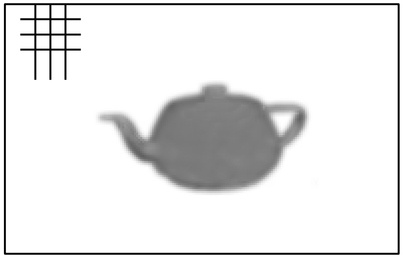
6.3 A use of the Z-buffer (Foley et al, 1990)

- Move an interactive 3D cursor  around a 3D scene
- The Z-buffer can be used to perform hidden-surface removal without updating the Z-buffer during cursor movement

Screen / monitor



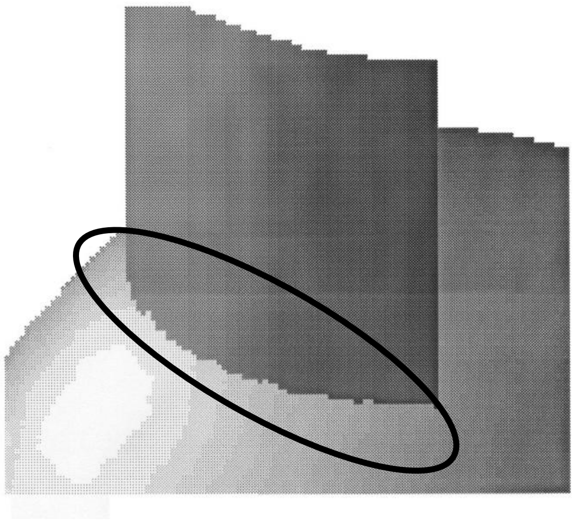
Framebuffer / screen buffer / colour buffer



Z buffer (map depth to greyscale)

6.4 Disadvantages

- Over-renders – pixels overwrite existing pixels, unnecessary info is calculated;
- Amount of memory for Z-buffer:
  - Between 20 and 32 bits is usually deemed sufficient.
  - 20 bits per pixel, resolution 1024x768 gives ~2Mb.
  - The scene has to be scaled to this fixed range to maximise accuracy.
- Possible depth quantization errors →



6.5 Transparency/Translucency

- Extension
- Save all pixel values
  - Every pixel in the Z-buffer maintains a linked list of values for all polygons that overlap that pixel
- On completion of processing polygons, there is a list of intensity values and depth values for each pixel.
- Order the list by depth
- Weight the different list items in order of depth, front to back
  - Once a non-transparent polygon (value) is reached, stop processing for that pixel
- Using OpenGL, rendering translucent objects requires special consideration (http://www.opengl.org/wiki/Transparency_Sorting)

6.6 Issues – clip plane settings

Far plane too distant – z accuracy reduced causing ‘z fighting’ artifacts

Far plane too close, so visible in scene

Can use ‘fog’

7. Summary

- Rasterising polygons – calculate the pixels a polygon covers
- For programmable pipeline, most popular rendering combination is Z-buffer, Phong reflection model, Phong interpolation
  - Z-buffer is ubiquitous in hardware
  - (Alternative: PowerVR - Tile-based deferred rendering)
- Because of inherent inefficiency, cannot leave visibility to Z-buffer for real-time graphics when dealing with complex scenes
  - Must have pre-Z-buffer strategies, e.g. BSP trees, Potentially Visible Sets, View Frustum Culling
- r,g,b,α. The value of α is used to blend a new value with the existing value in a pixel