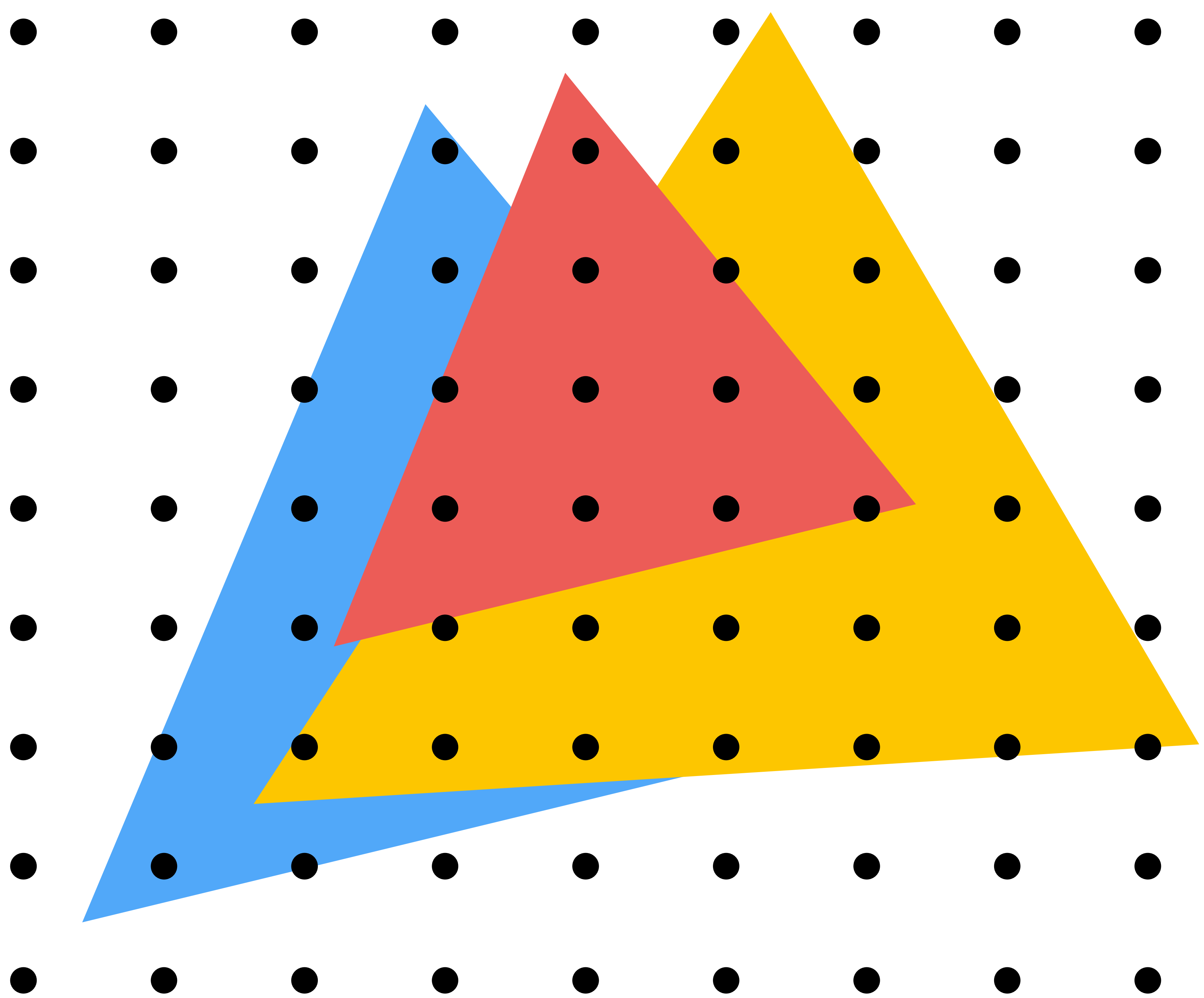
The Rasterization Pipeline

Computer Graphics
CMU 15-462/15-662, Fall 2016

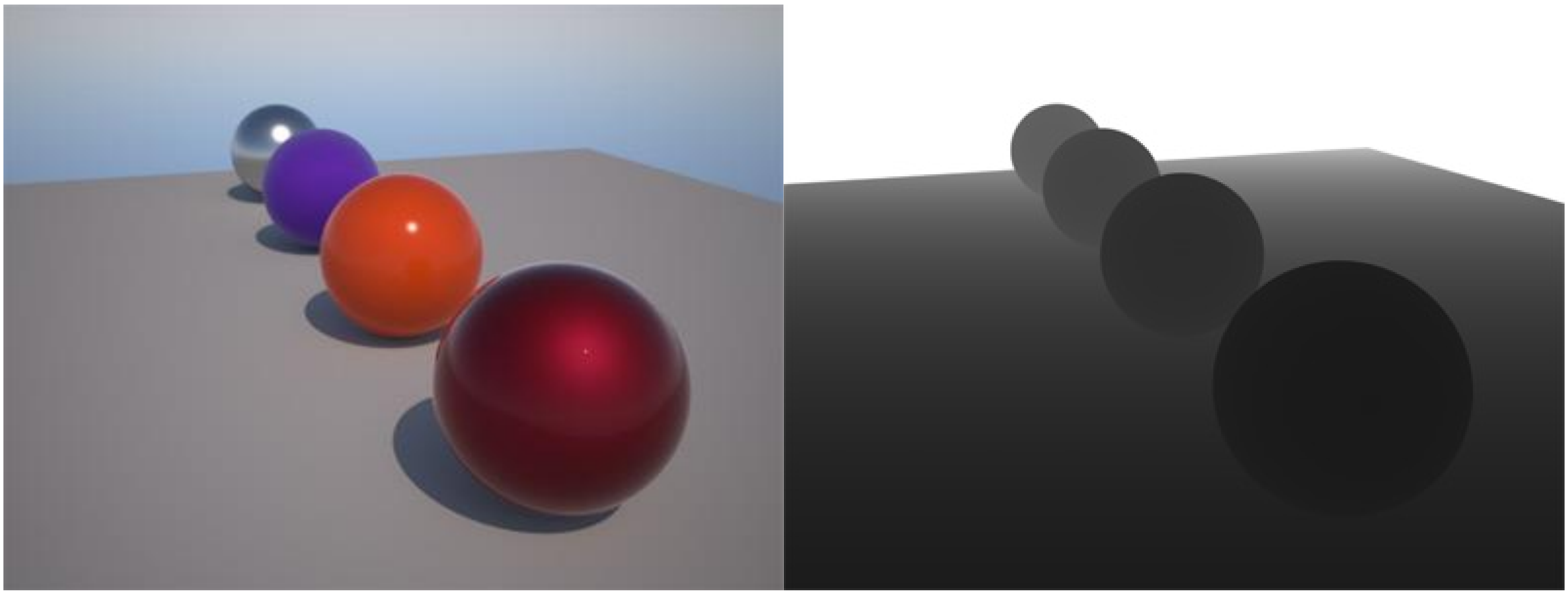
Occlusion

Which triangle is visible at each pixel?



Opaque Triangles

The depth buffer (Z-buffer)



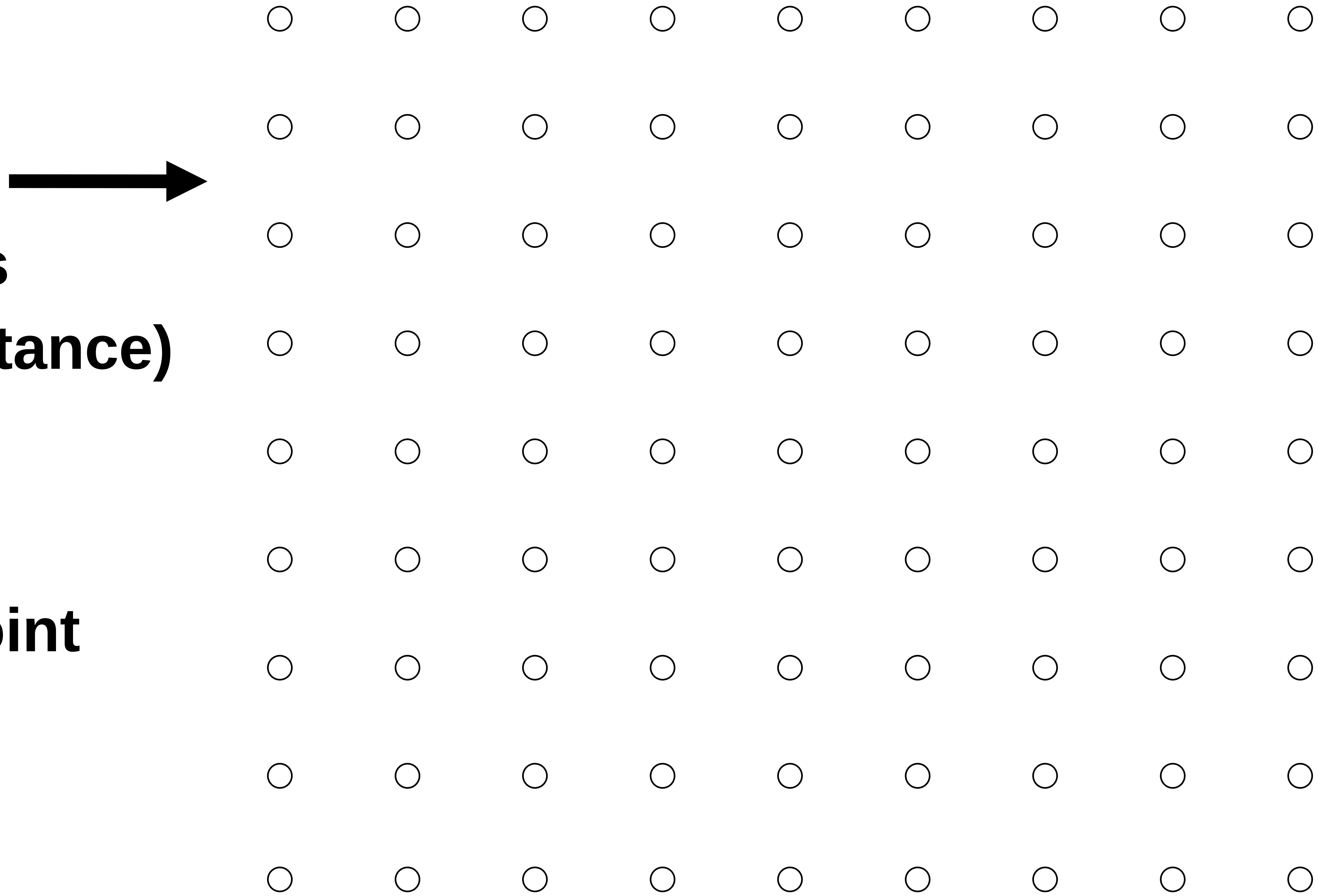
Q: How do we compute the depth of sampled points on a triangle?

Interpolate it just like any other attribute that varies linearly over the surface of the triangle.

Occlusion using the depth-buffer (Z-buffer)

For each coverage sample point, depth-buffer stores depth of closest triangle at this sample point that has been processed by the renderer so far.

Initial state of depth buffer before rendering any triangles (all samples store farthest distance)

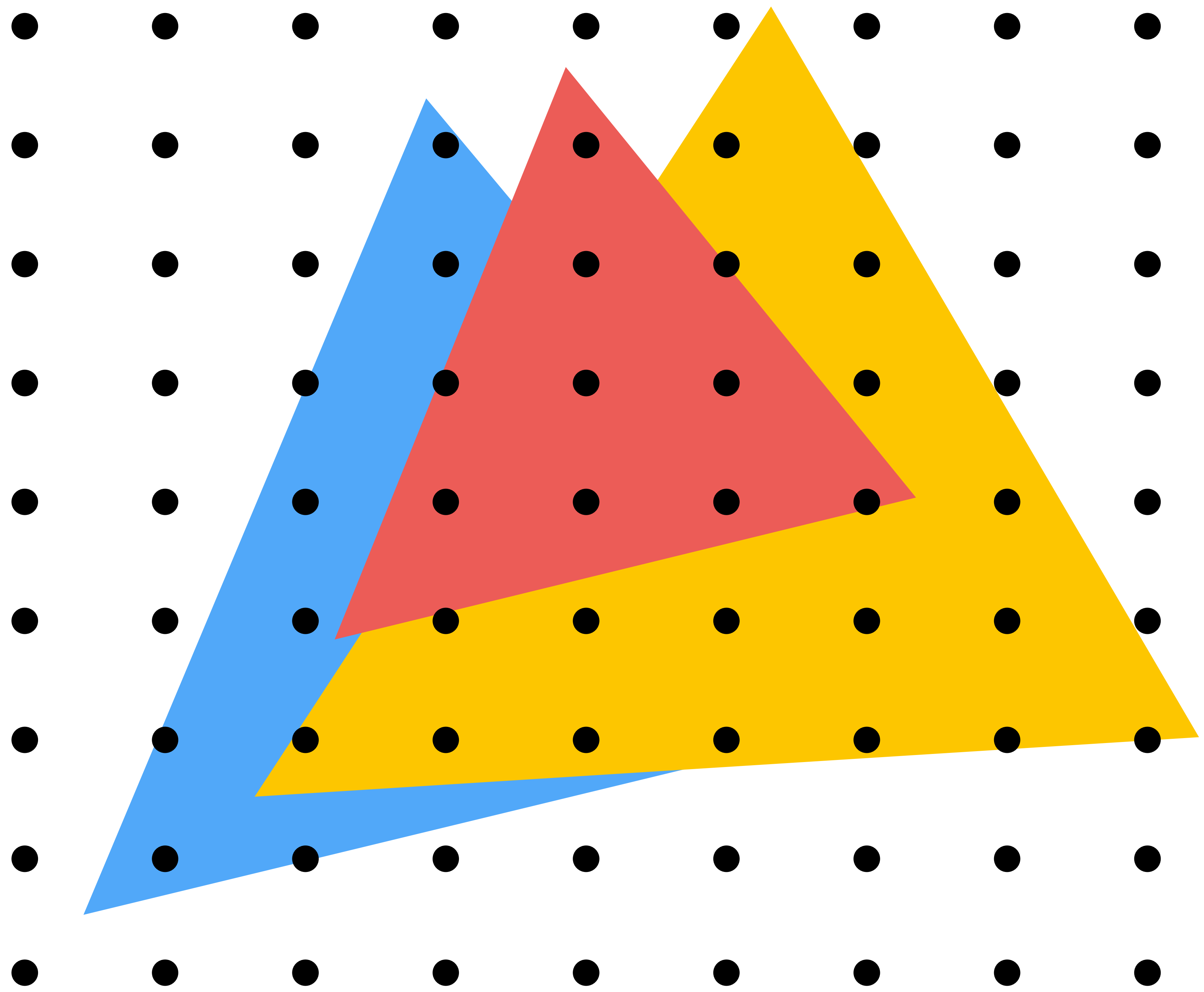


Grayscale value of sample point used to indicate distance

Black = small distance

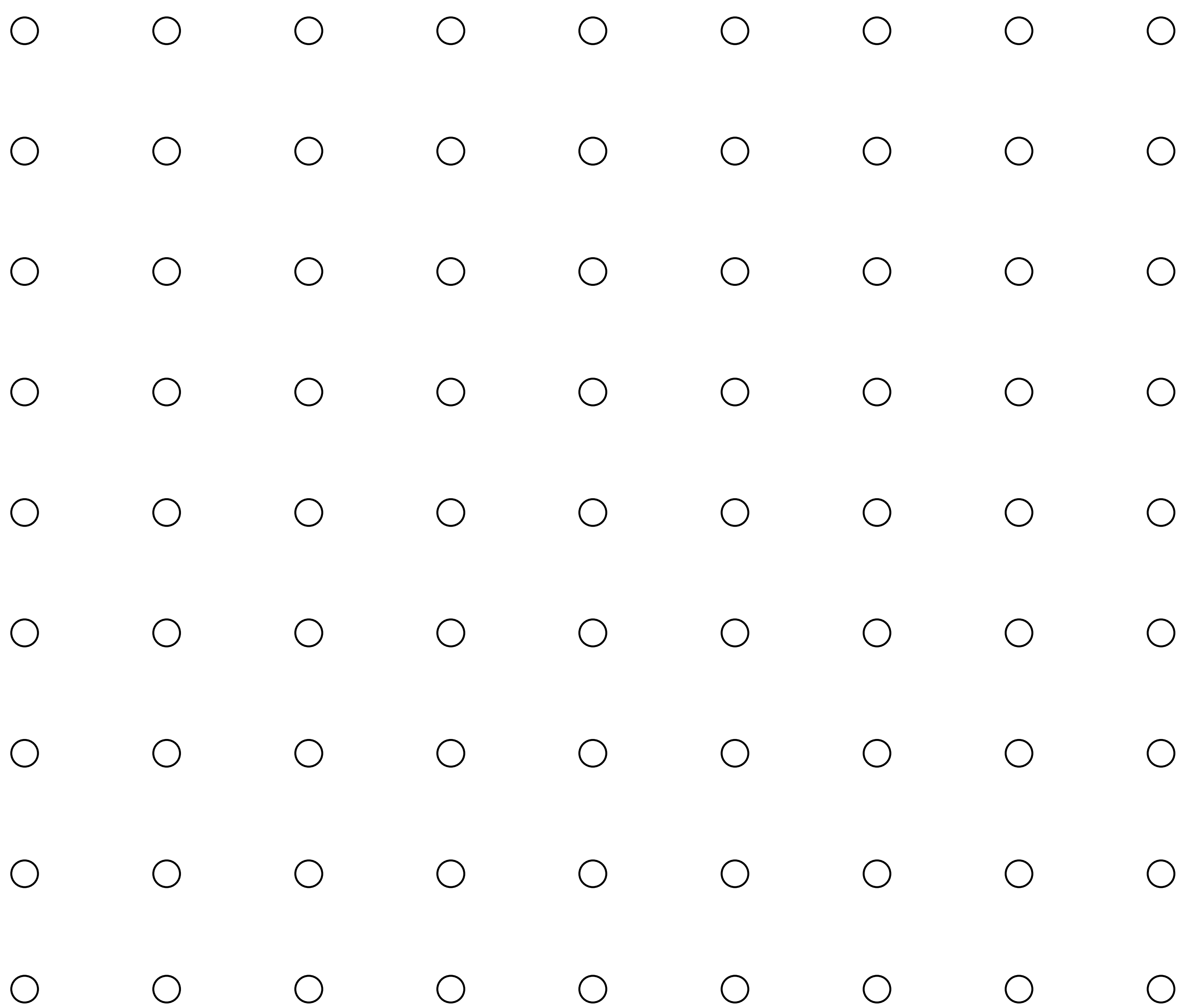
White = large distance

Example: rendering three opaque triangles



Occlusion using the depth-buffer (Z-buffer)

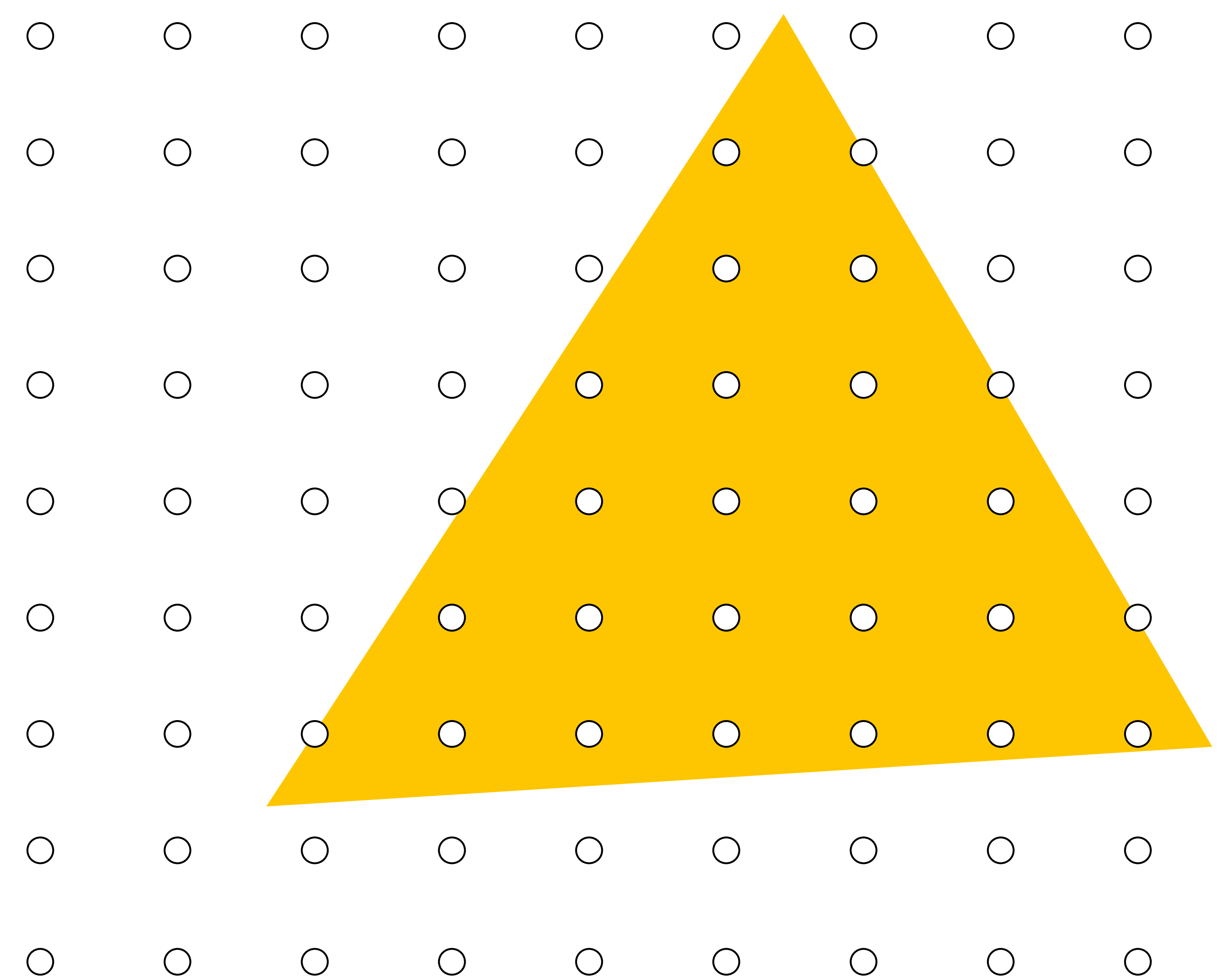
Processing yellow triangle:
depth = 0.5



Color buffer contents

Grayscale value of sample point used to indicate distance

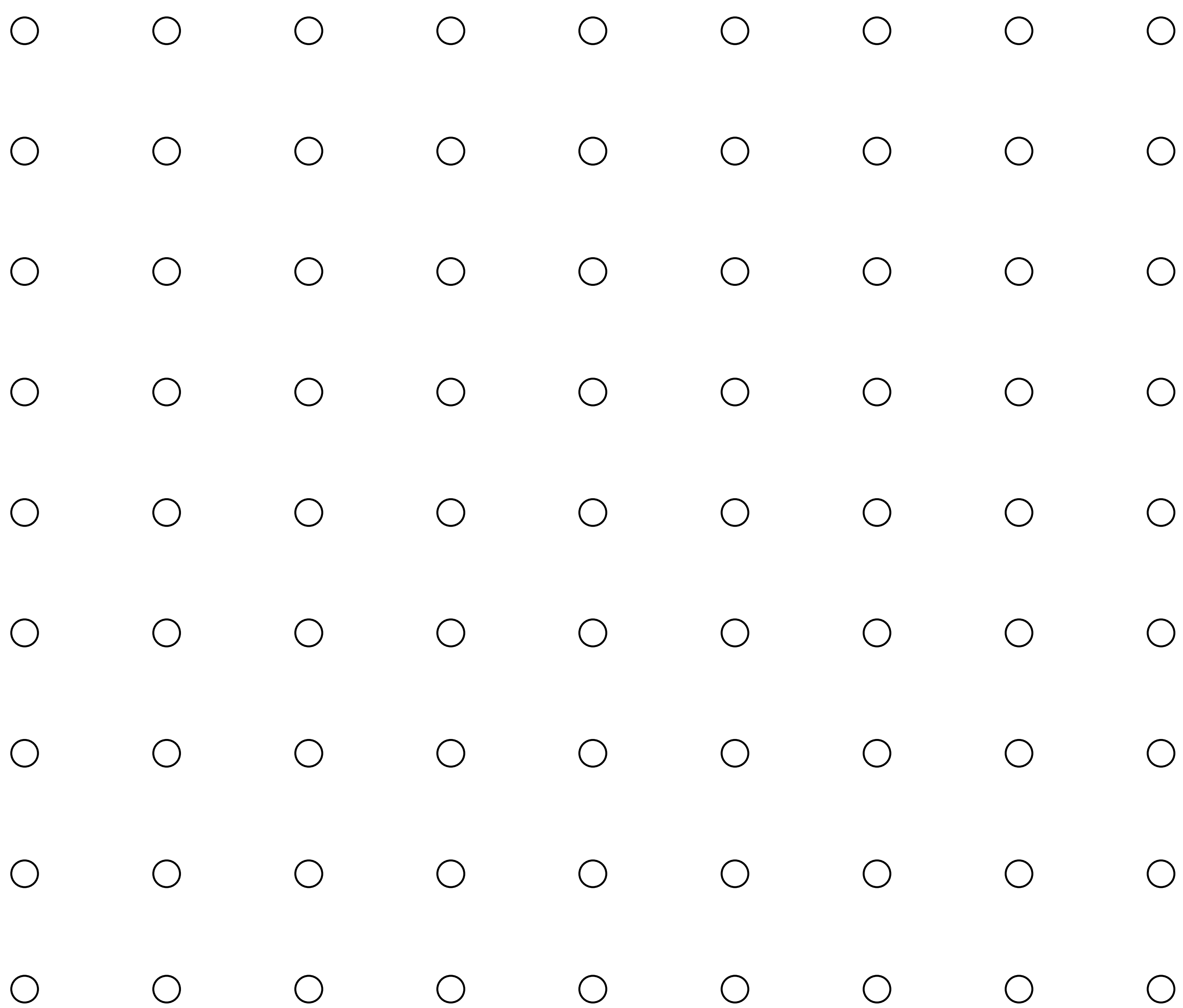
White = large distance
Black = small distance
Red = sample passed depth test



Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

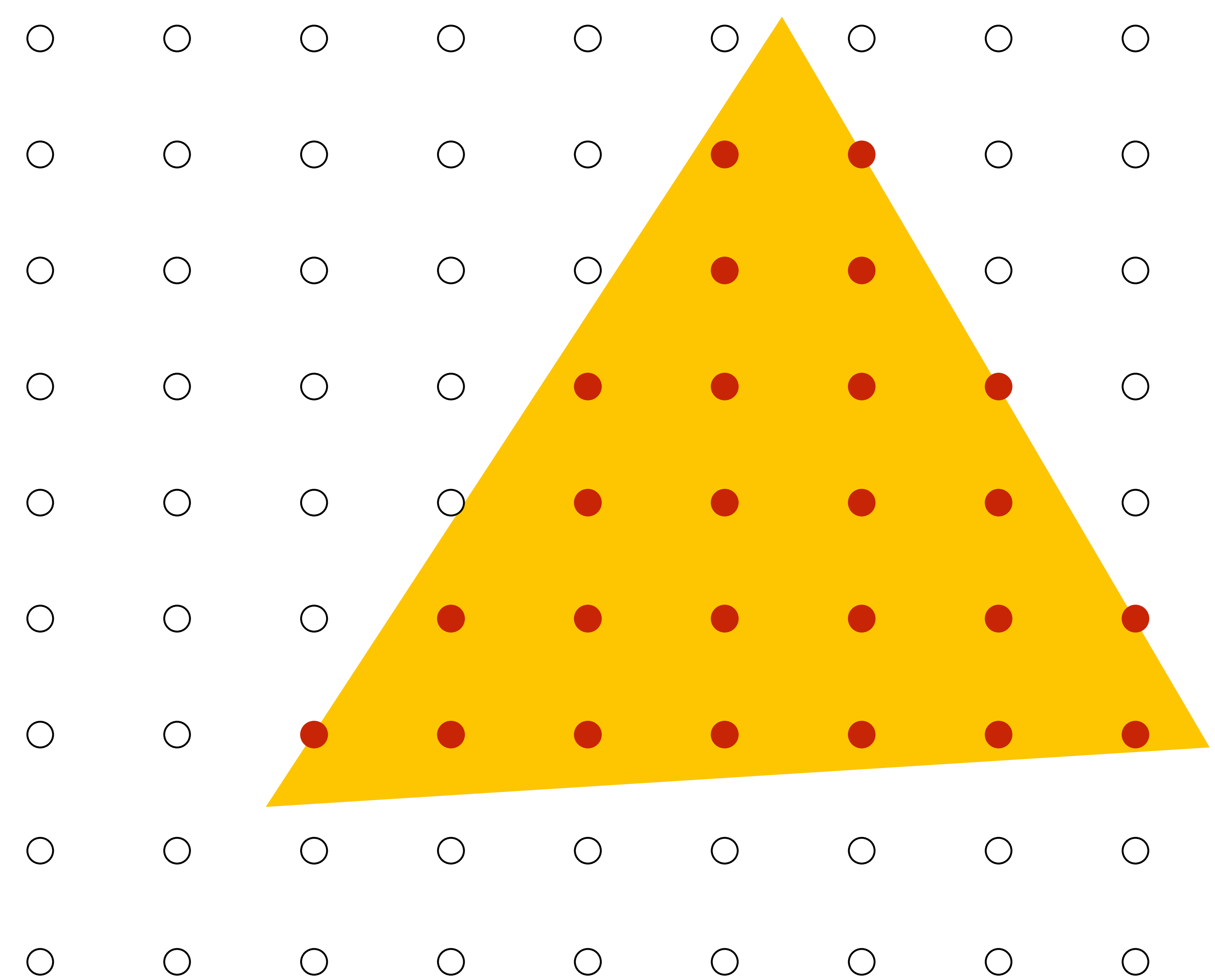
Processing yellow triangle:
depth = 0.5



Color buffer contents

Grayscale value of sample point used to indicate distance

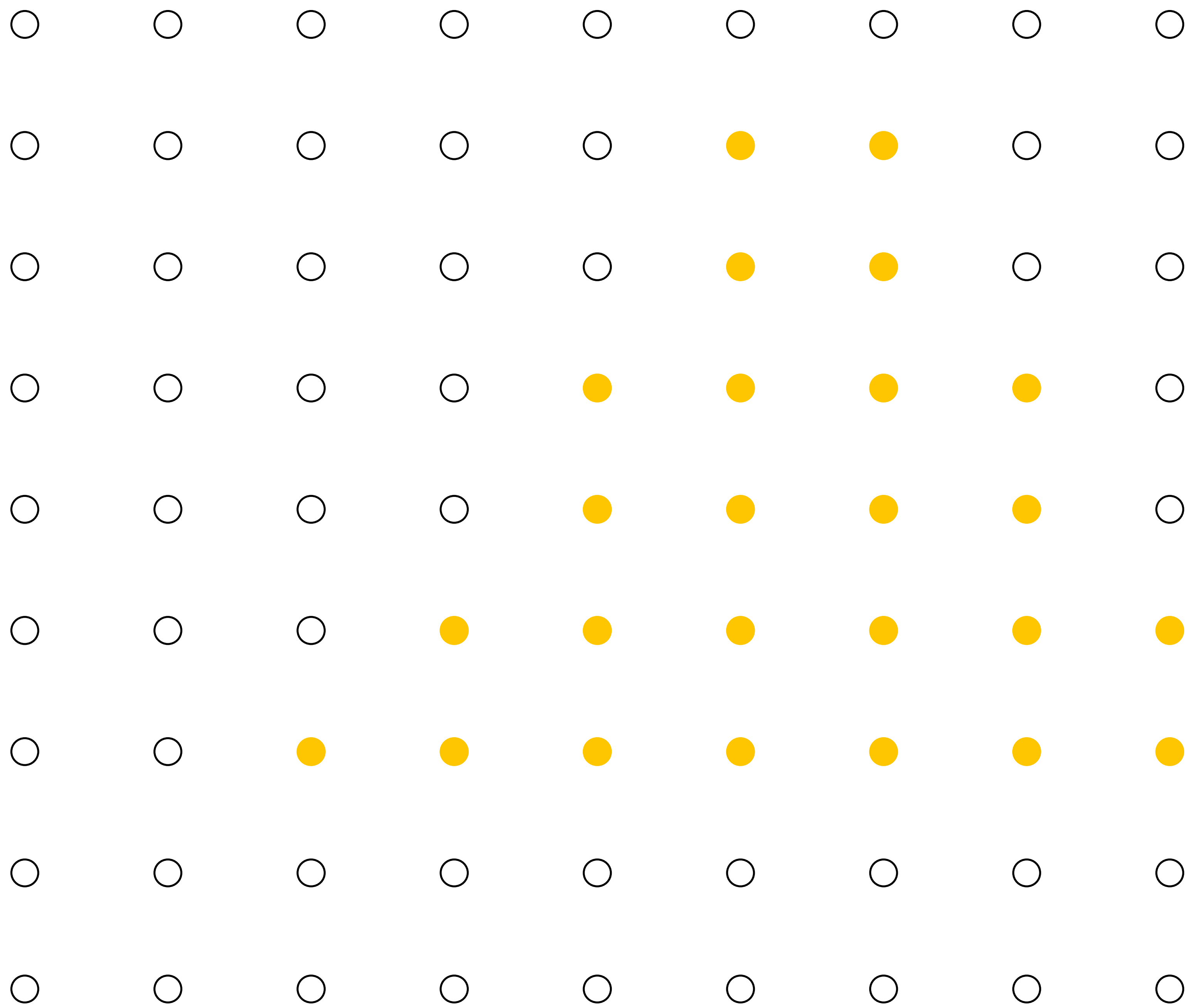
White = large distance
Black = small distance
Red = sample passed depth test



Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

After processing yellow triangle:



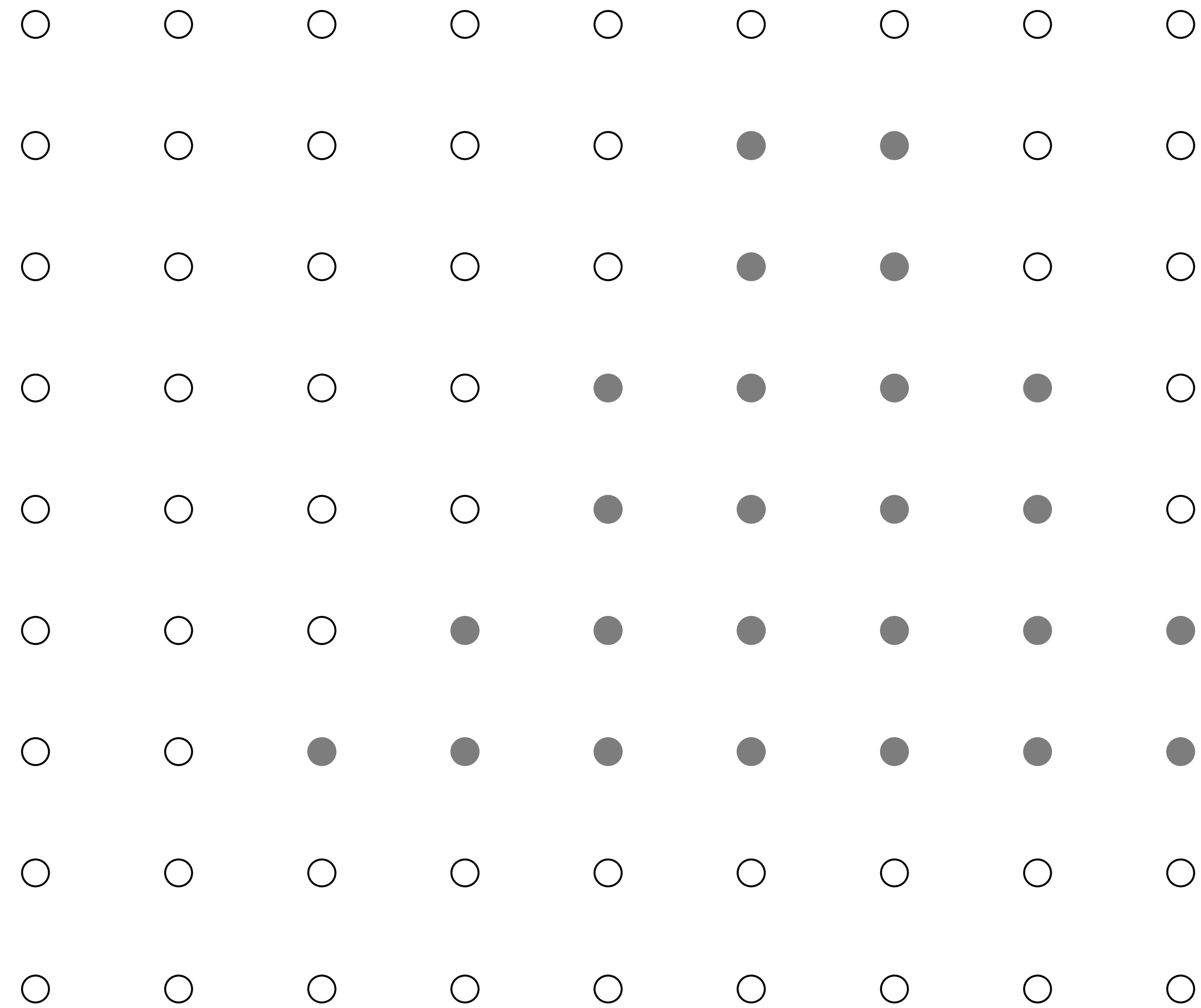
Color buffer contents

Grayscale value of sample point used to indicate distance

White = large distance

Black = small distance

Red = sample passed depth test



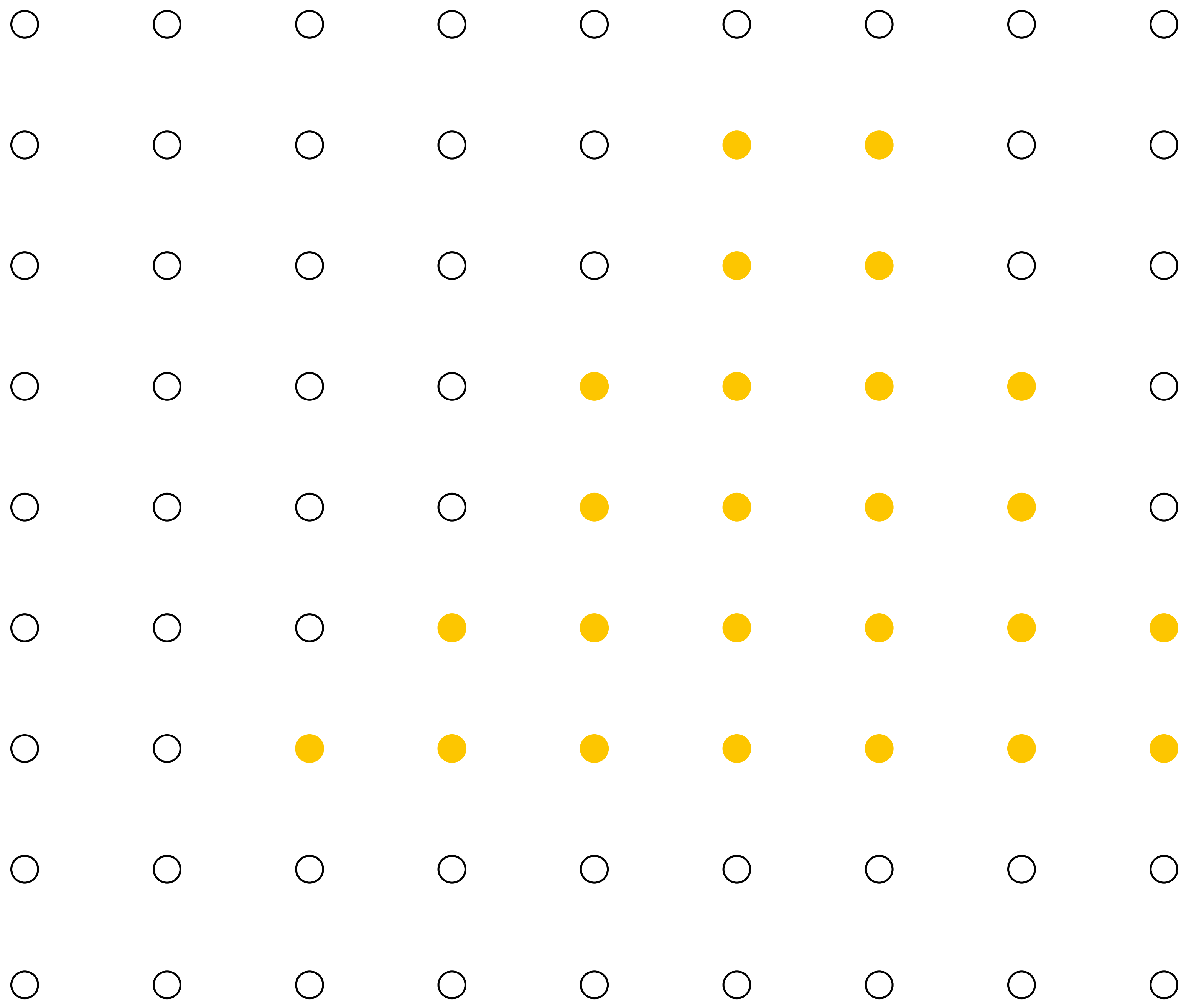
Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

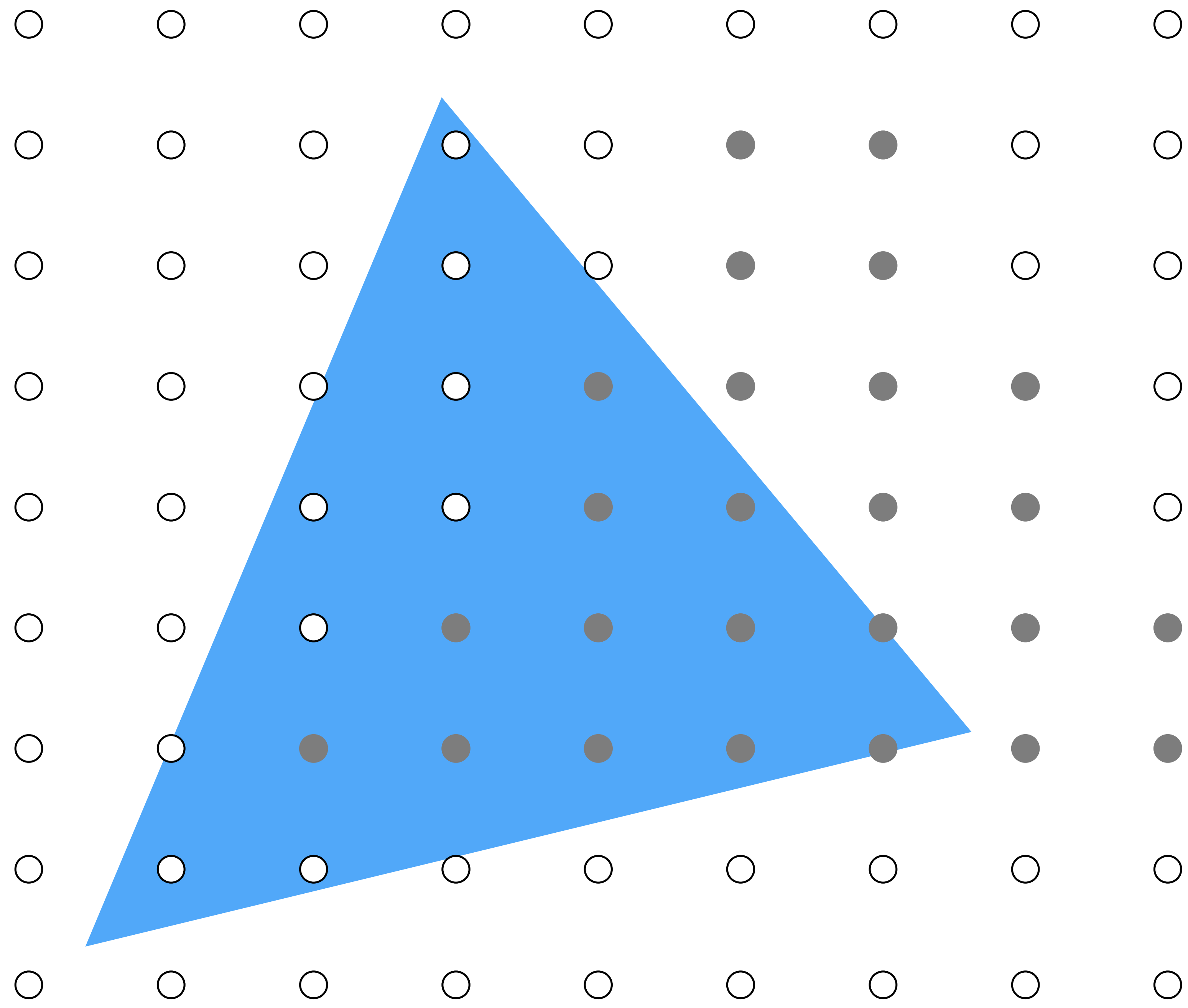
Processing blue triangle:
depth = 0.75

Grayscale value of sample point used to indicate distance

White = large distance
Black = small distance
Red = sample passed depth test



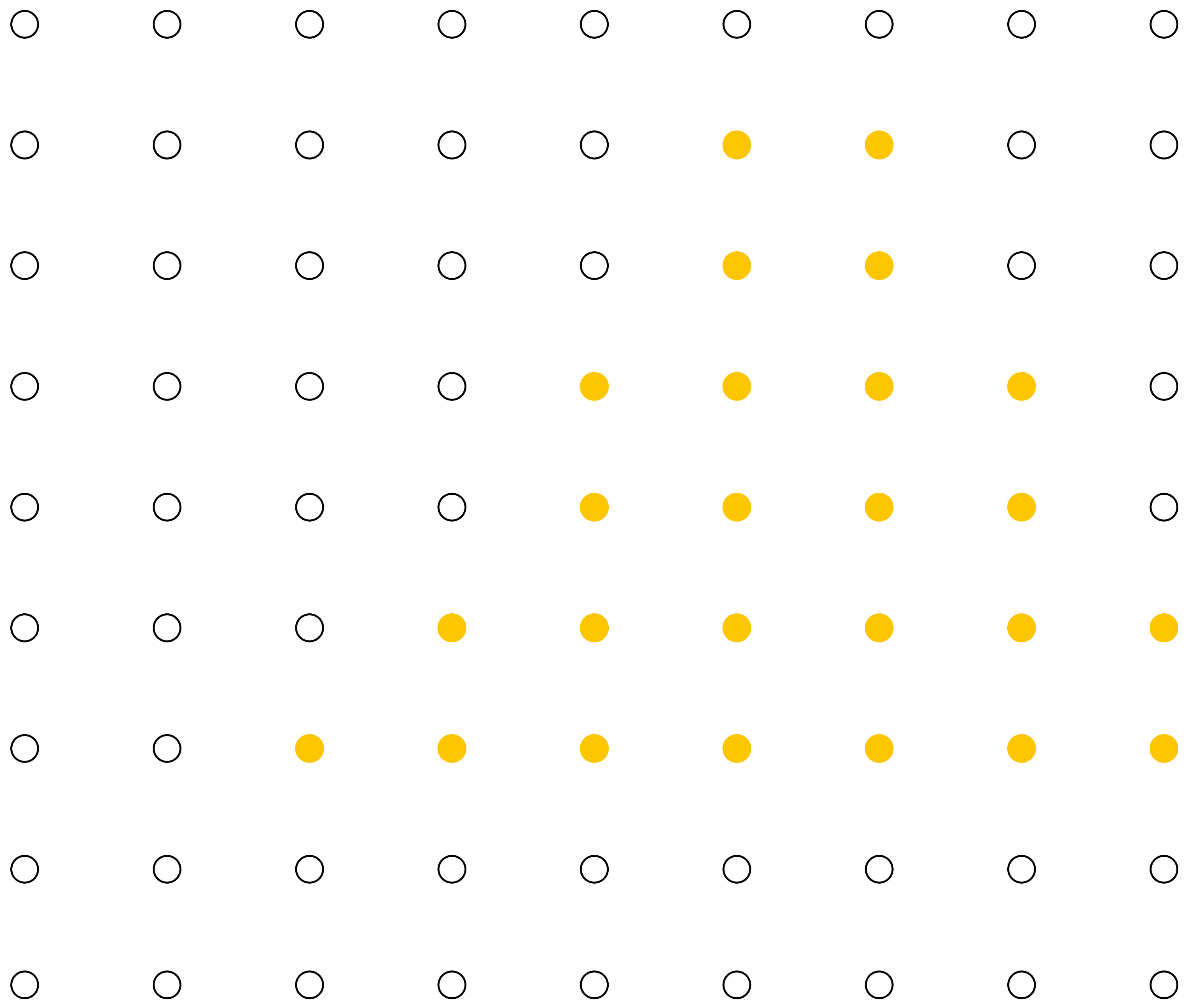
Color buffer contents



Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

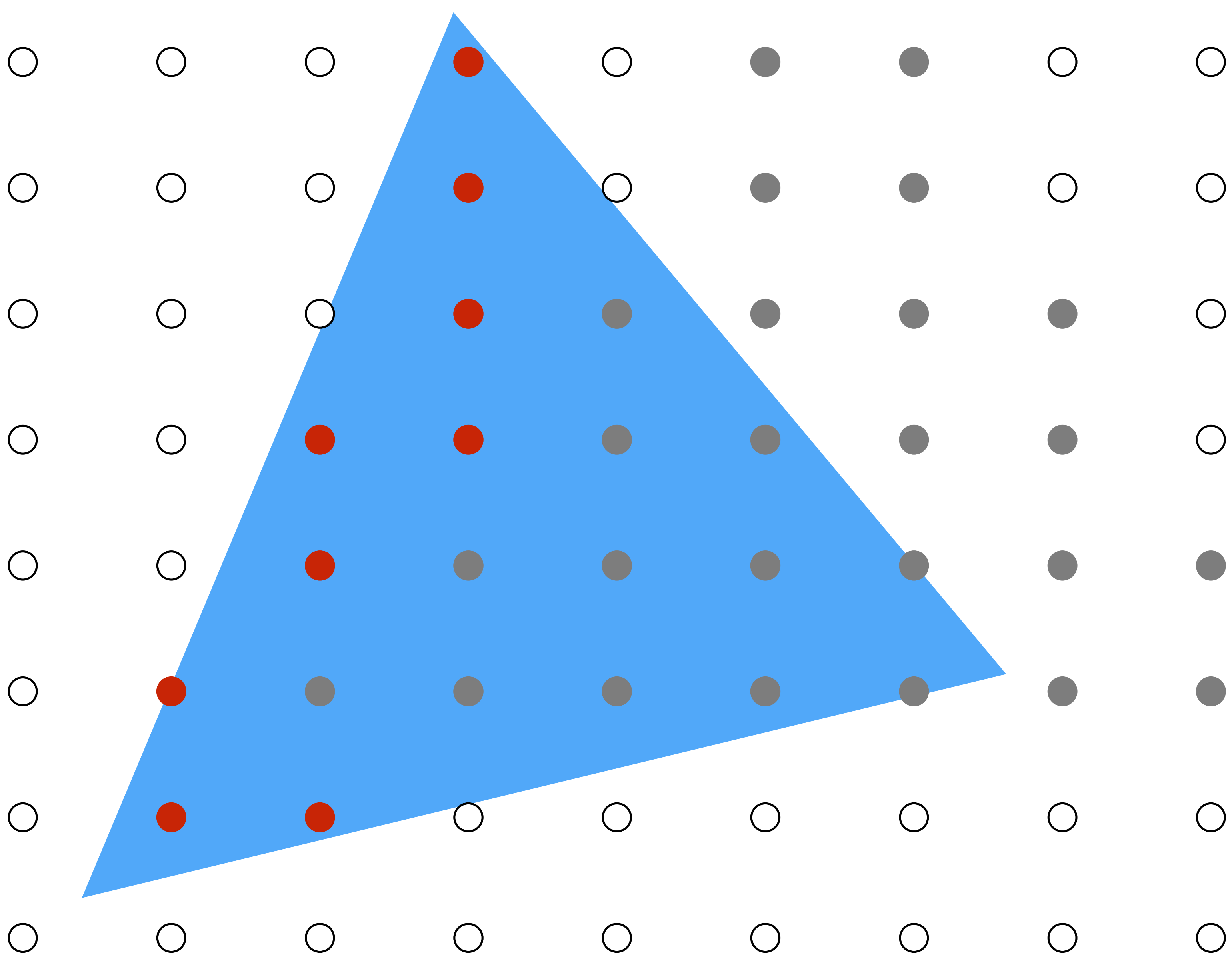
Processing blue triangle:
depth = 0.75



Color buffer contents

Grayscale value of sample point used to indicate distance

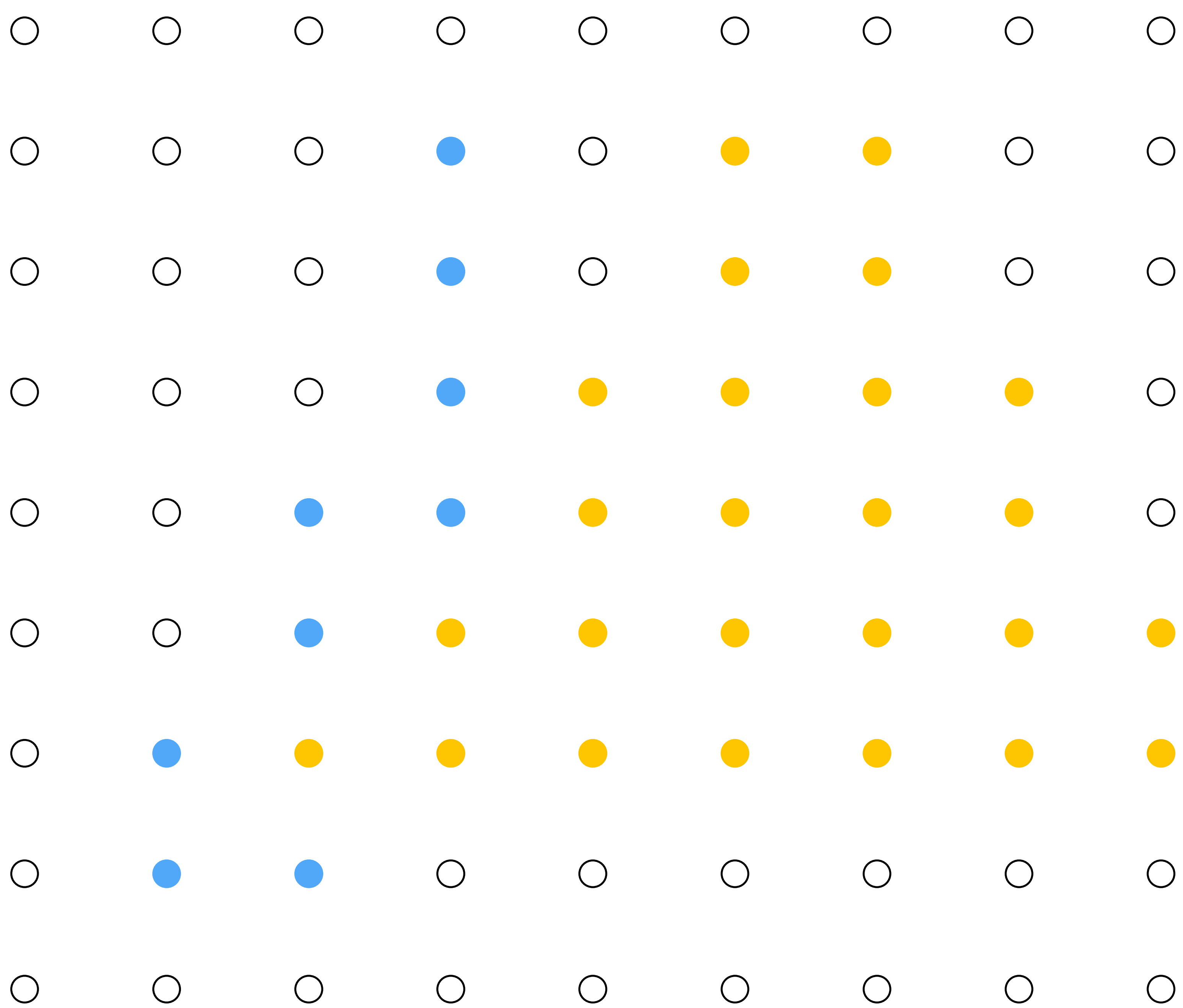
White = large distance
Black = small distance
Red = sample passed depth test



Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

After processing blue triangle:



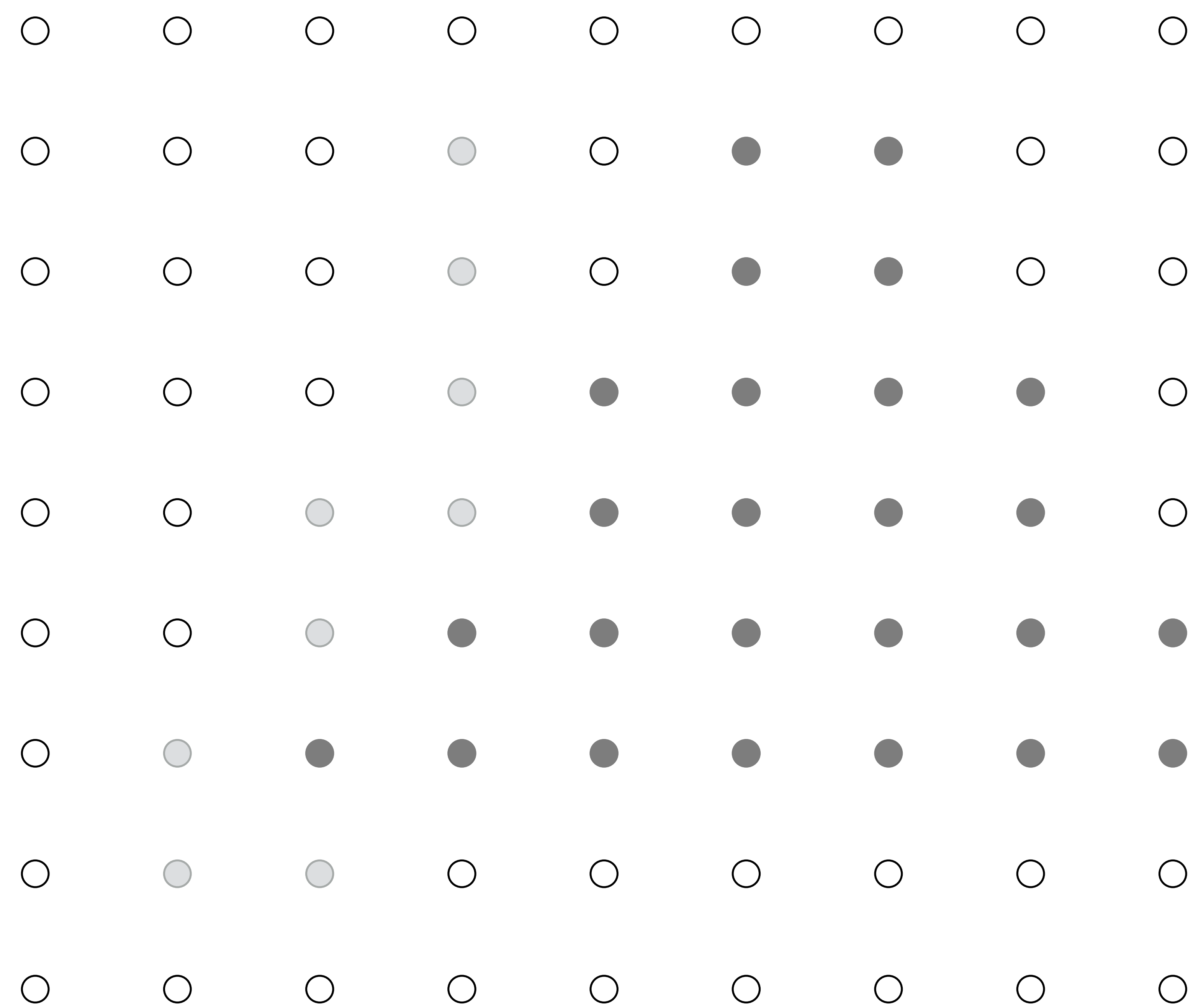
Color buffer contents

Grayscale value of sample point used to indicate distance

White = large distance

Black = small distance

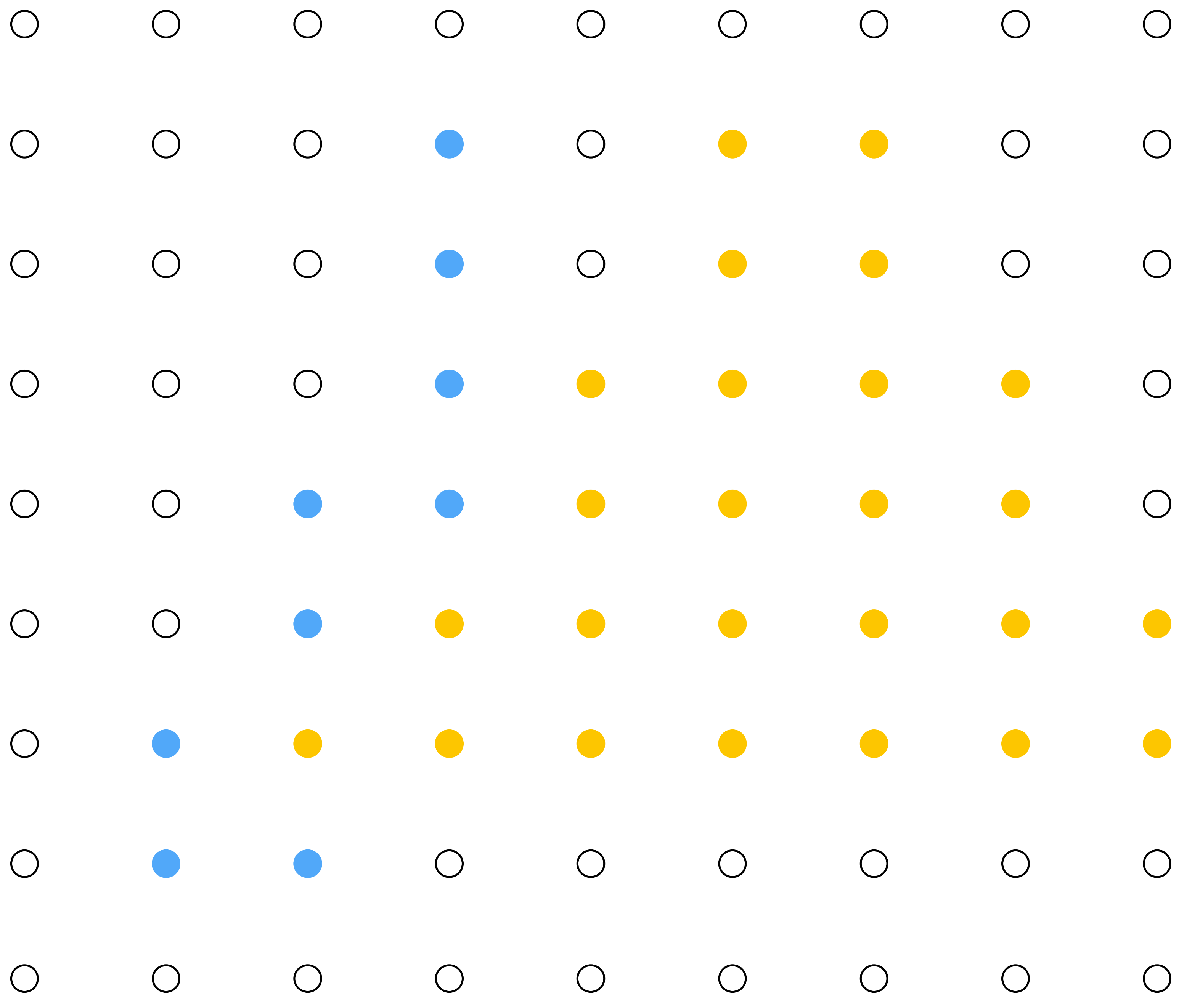
Red = sample passed depth test



Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

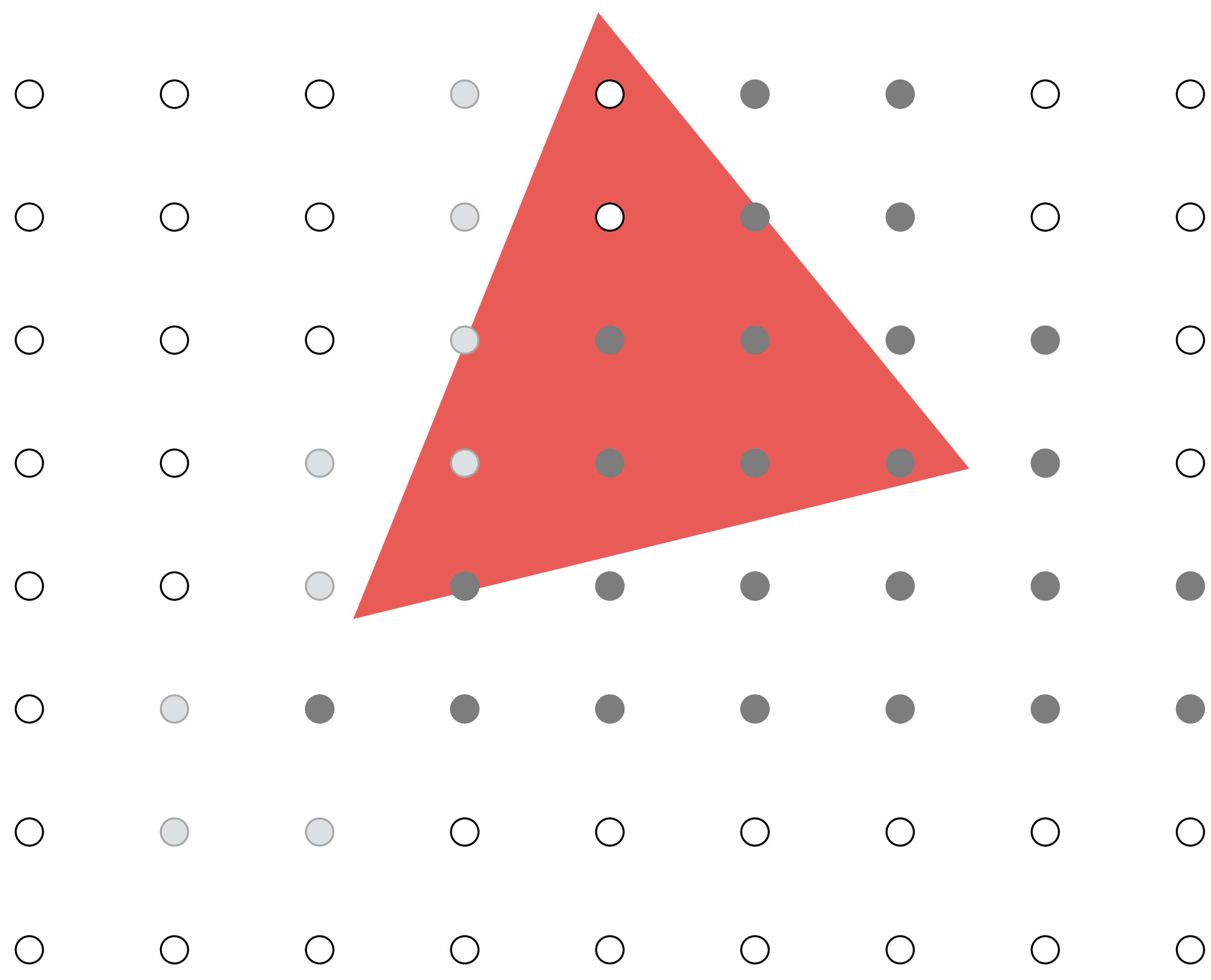
Processing red triangle:
depth = 0.25



Color buffer contents

Grayscale value of sample point used to indicate distance

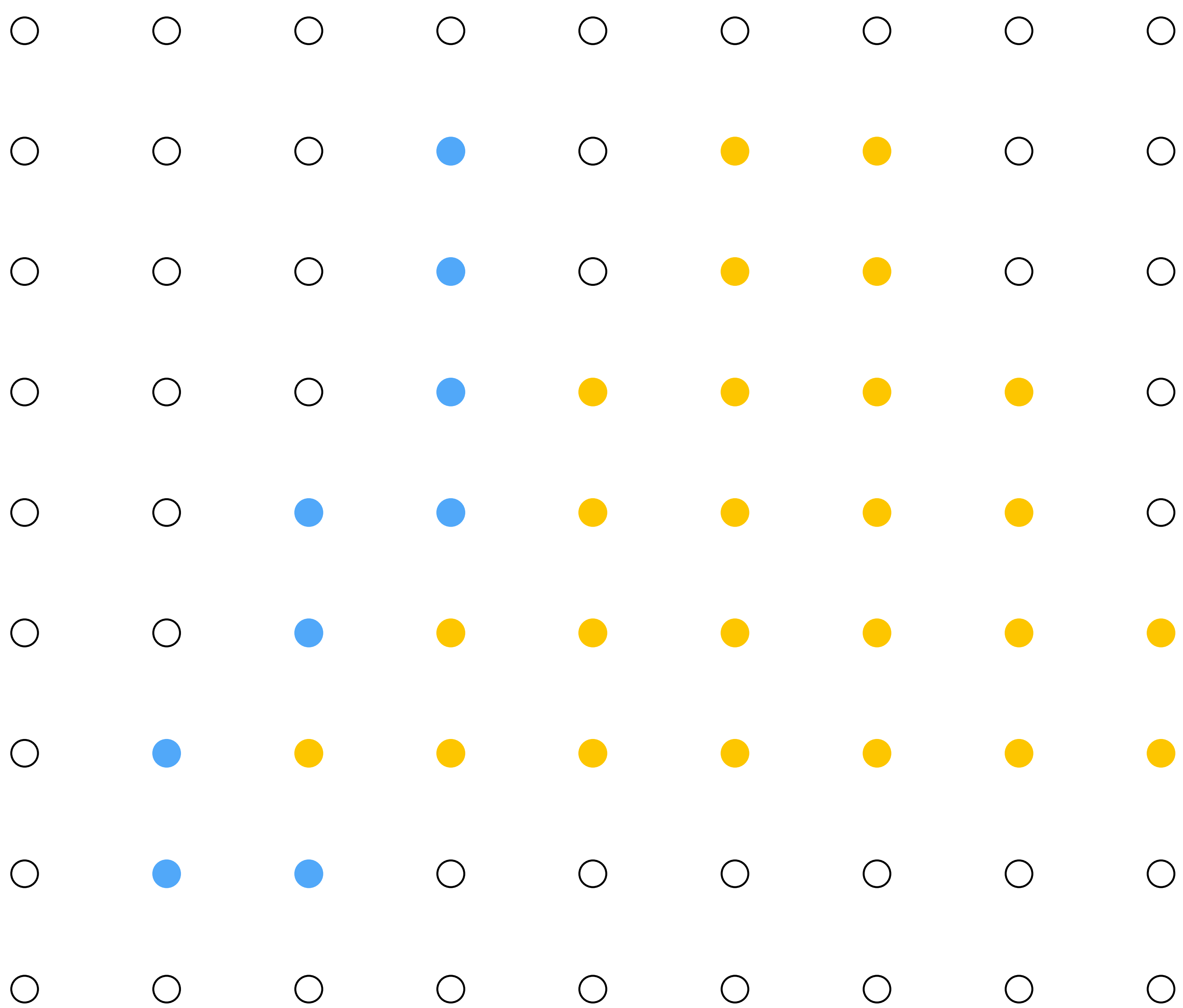
White = large distance
Black = small distance
Red = sample passed depth test



Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

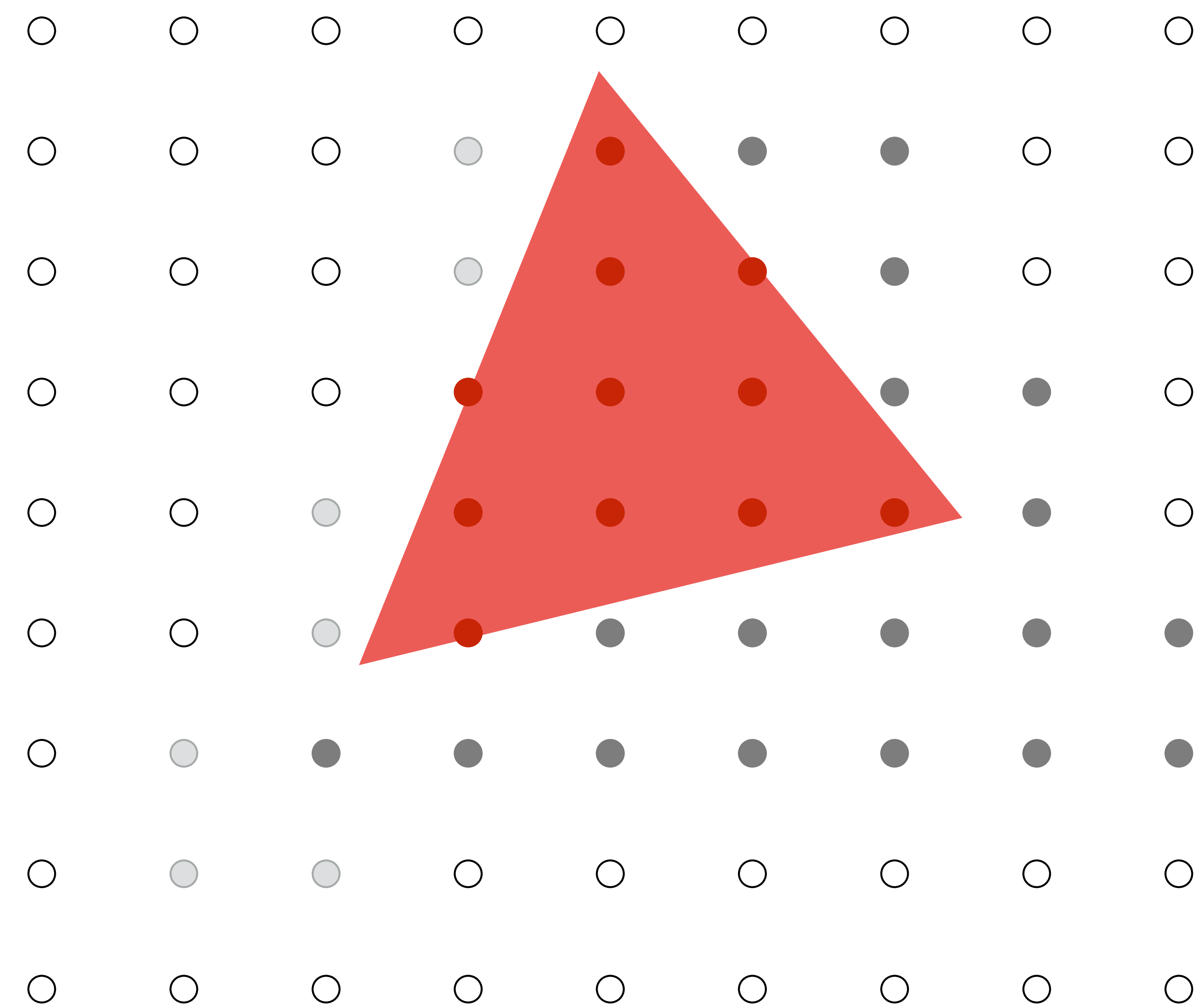
Processing red triangle:
depth = 0.25



Color buffer contents

Grayscale value of sample point used to indicate distance

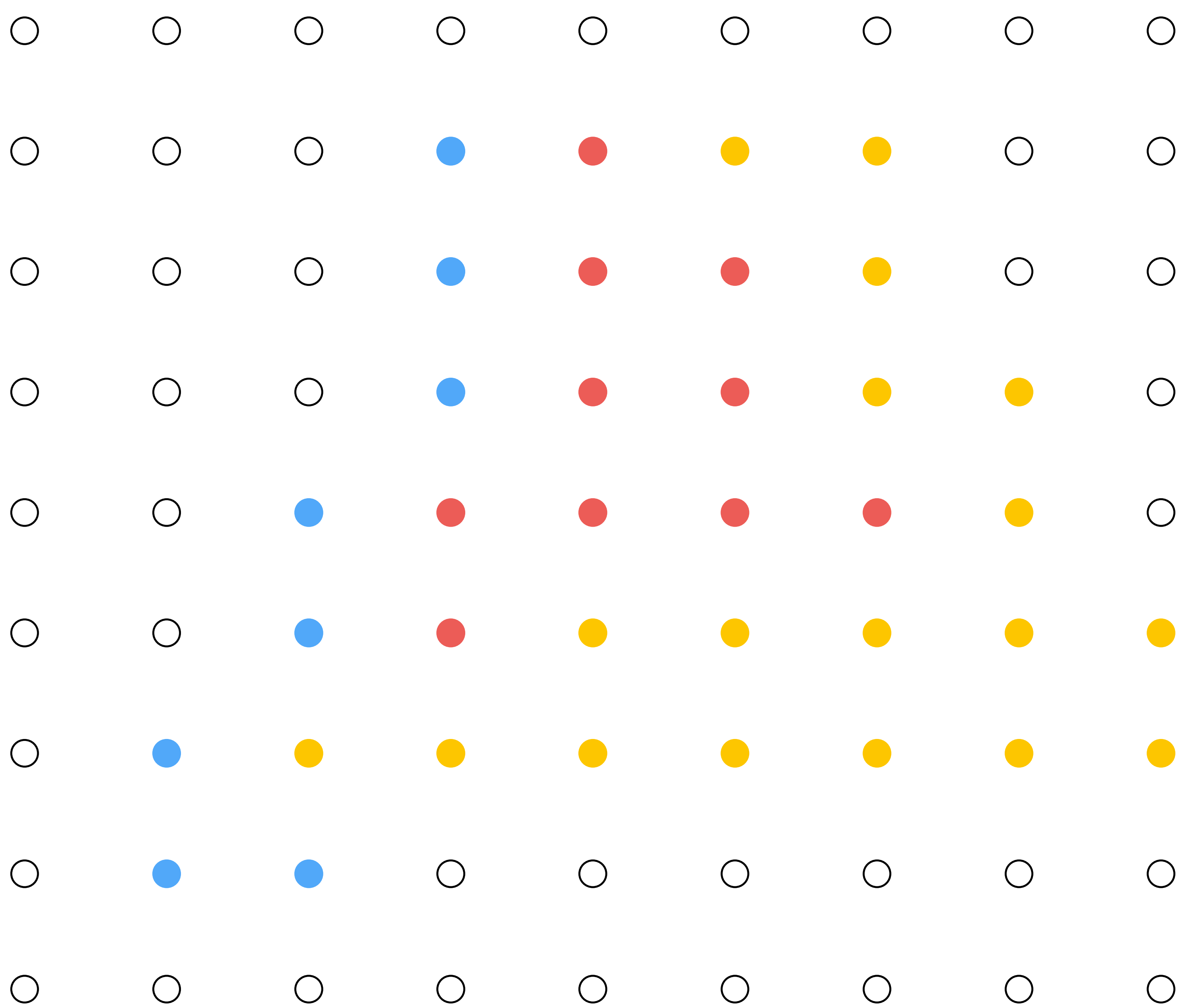
White = large distance
Black = small distance
Red = sample passed depth test



Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

After processing red triangle:



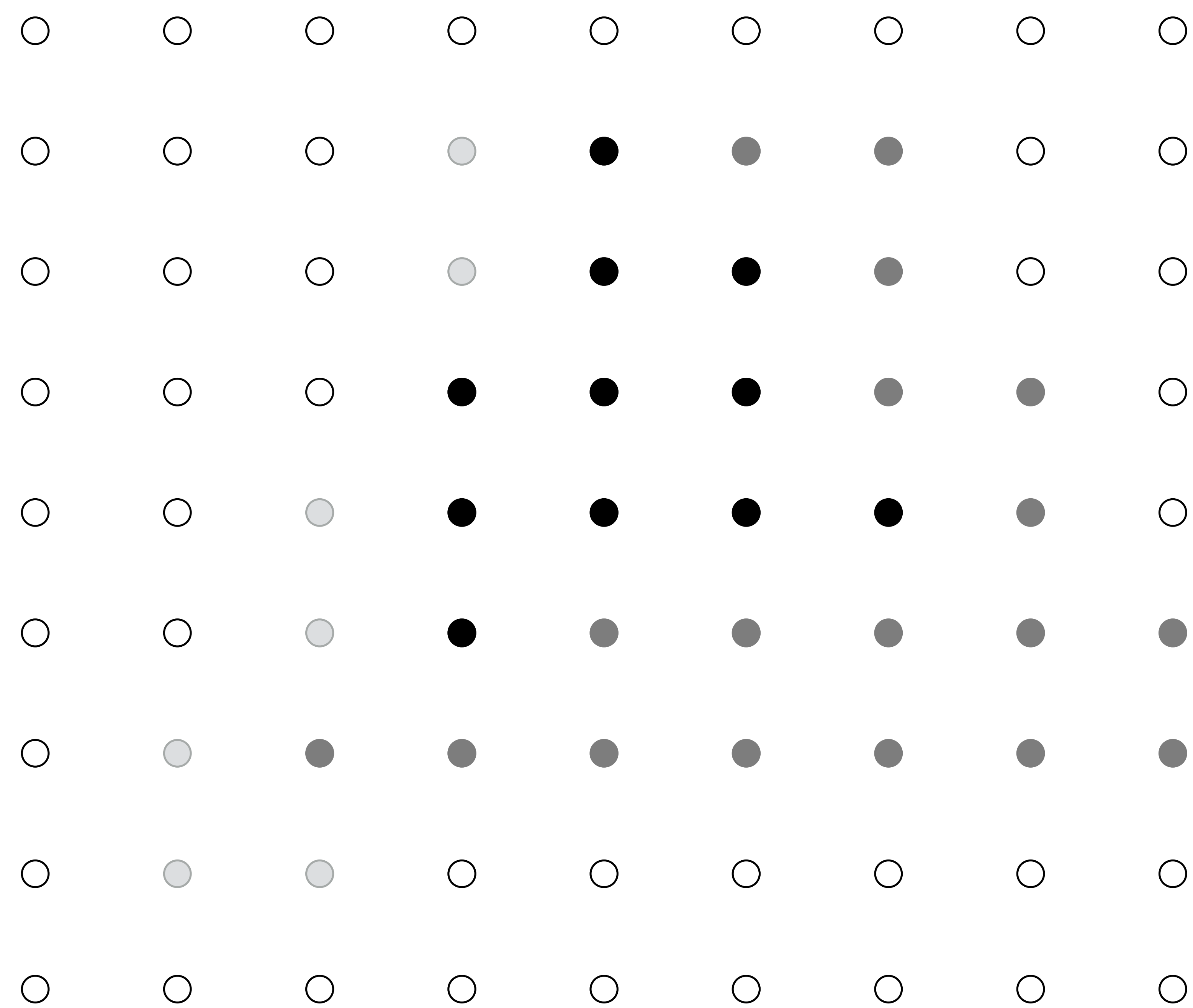
Color buffer contents

Grayscale value of sample point used to indicate distance

White = large distance

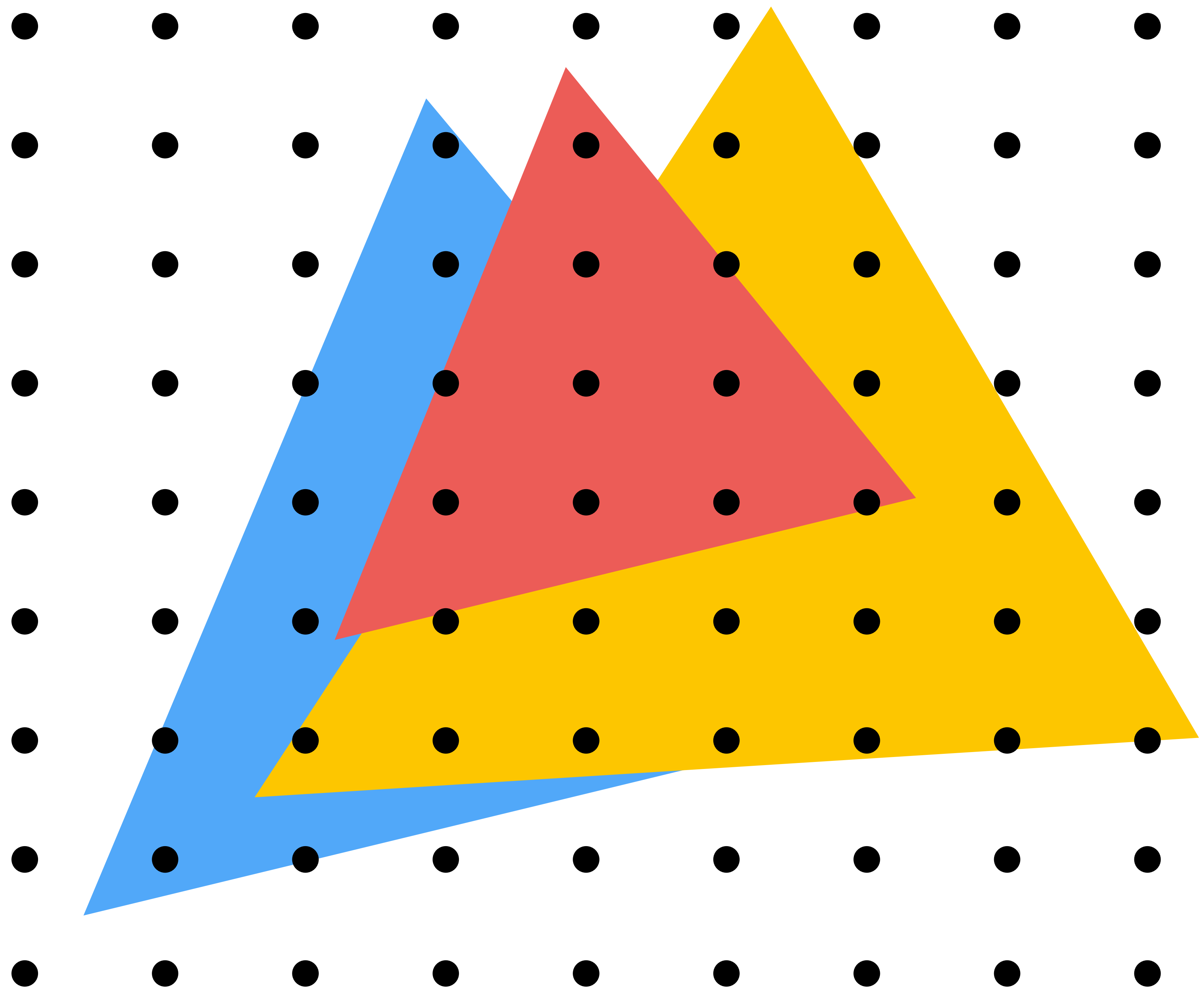
Black = small distance

Red = sample passed depth test



Depth buffer contents

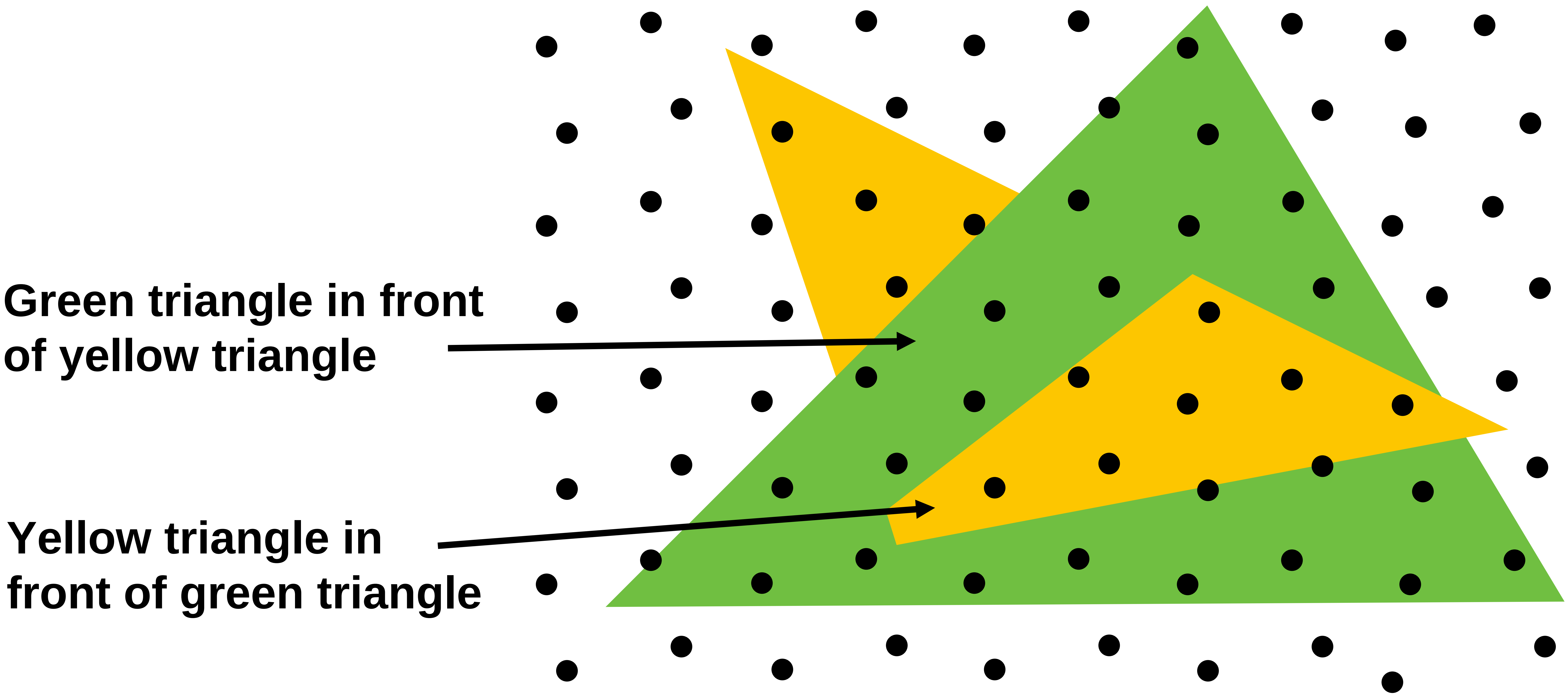
Example: rendering three opaque triangles



Does the depth-buffer algorithm handle interpenetrating surfaces?

Of course!

Occlusion test is based on depth of triangles at a given sample point.
The relative depth of triangles may be different at different sample points.

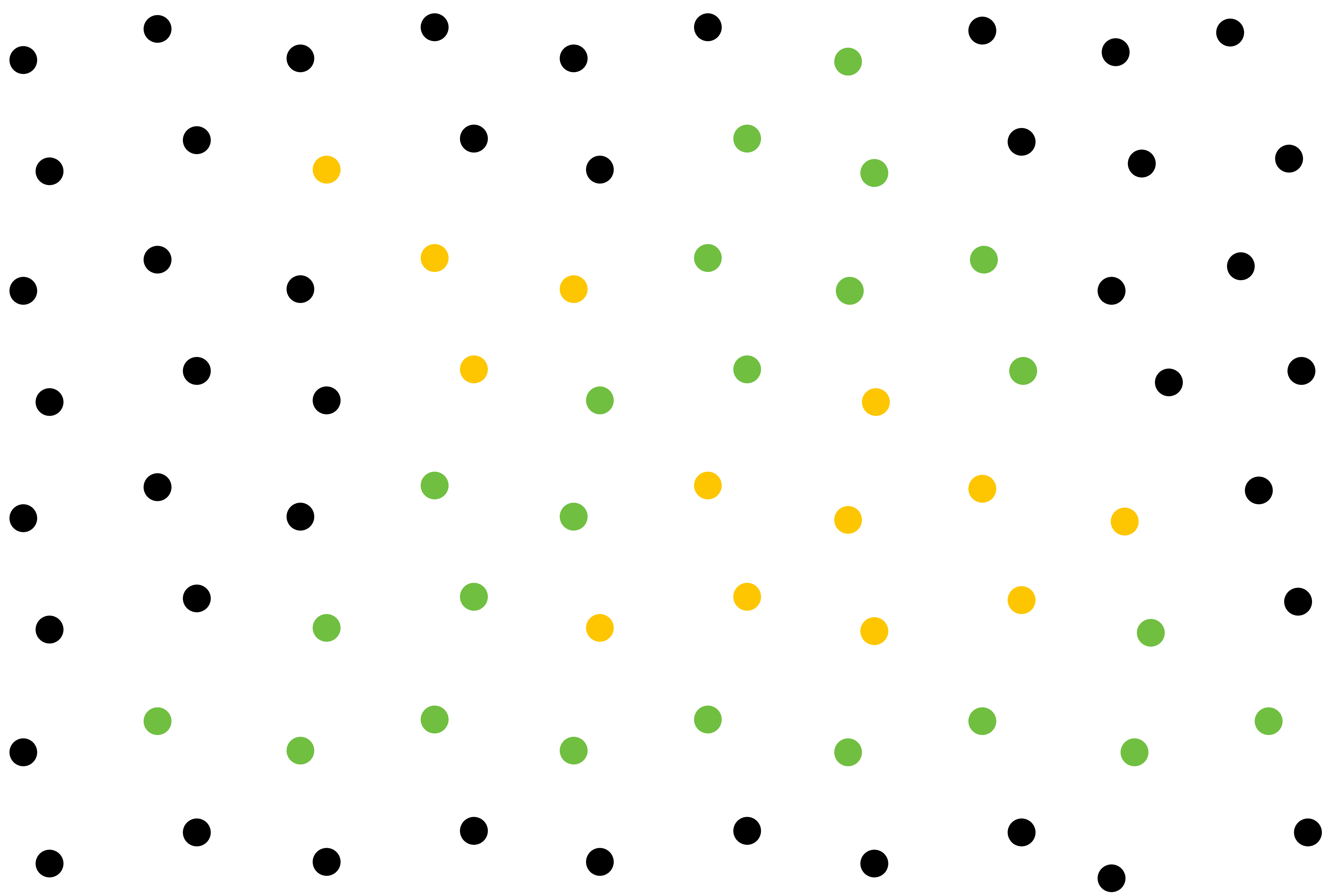


Does the depth-buffer algorithm handle interpenetrating surfaces?

Of course!

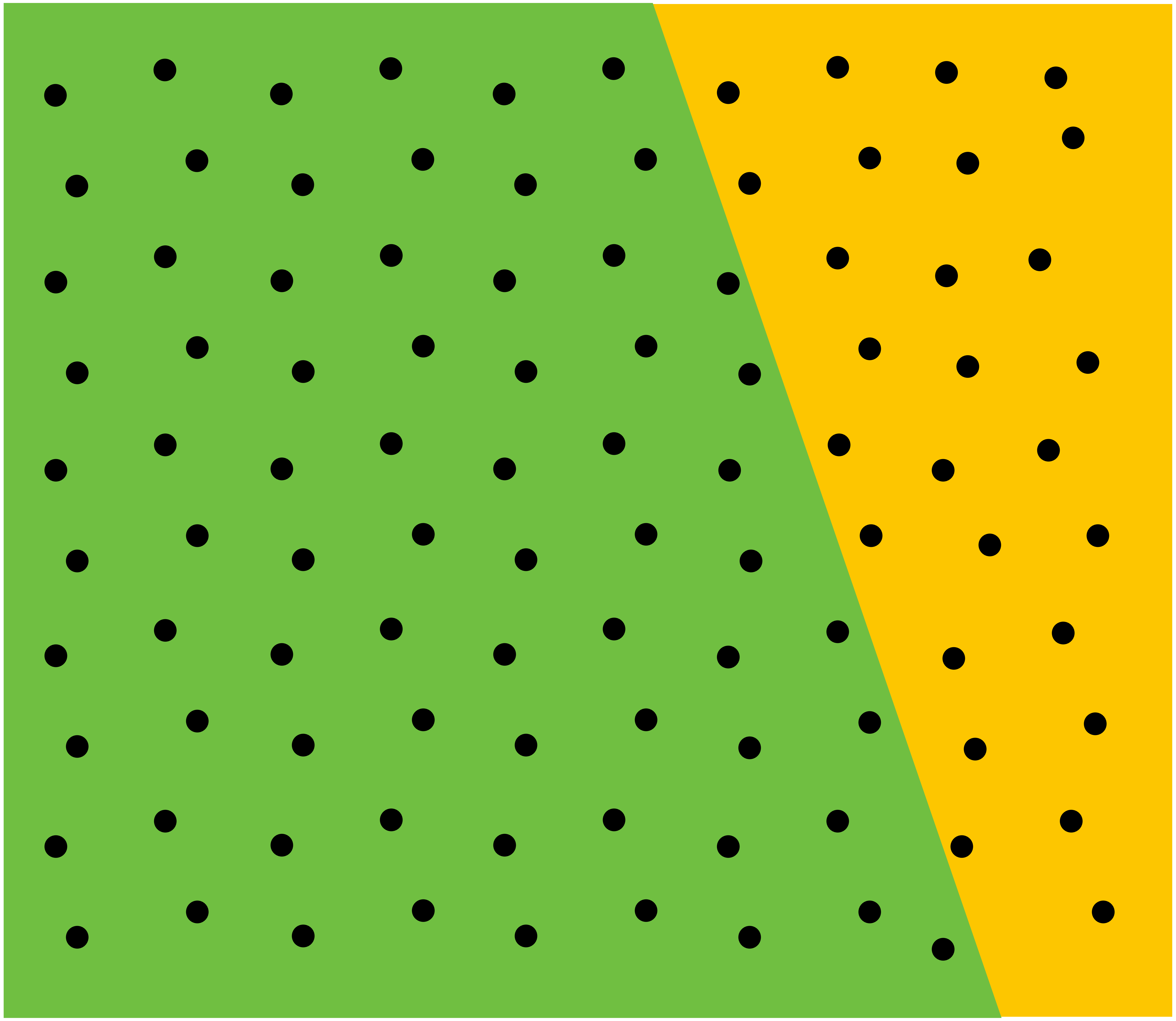
Occlusion test is based on depth of triangles at a given sample point.

The relative depth of triangles may be different at different sample points.



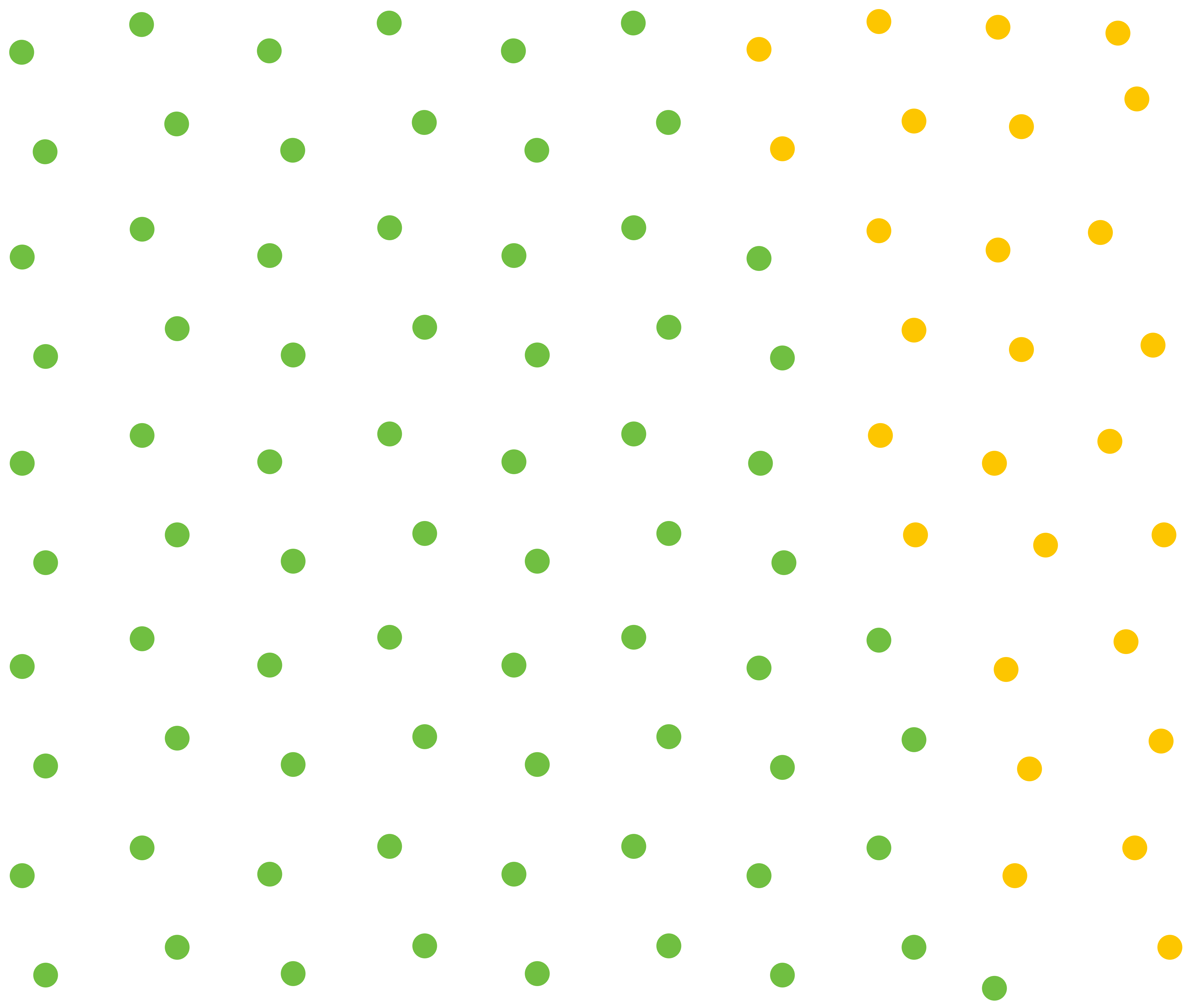
Does it work with super sampling?

Of course! Occlusion test is per sample, not per pixel!

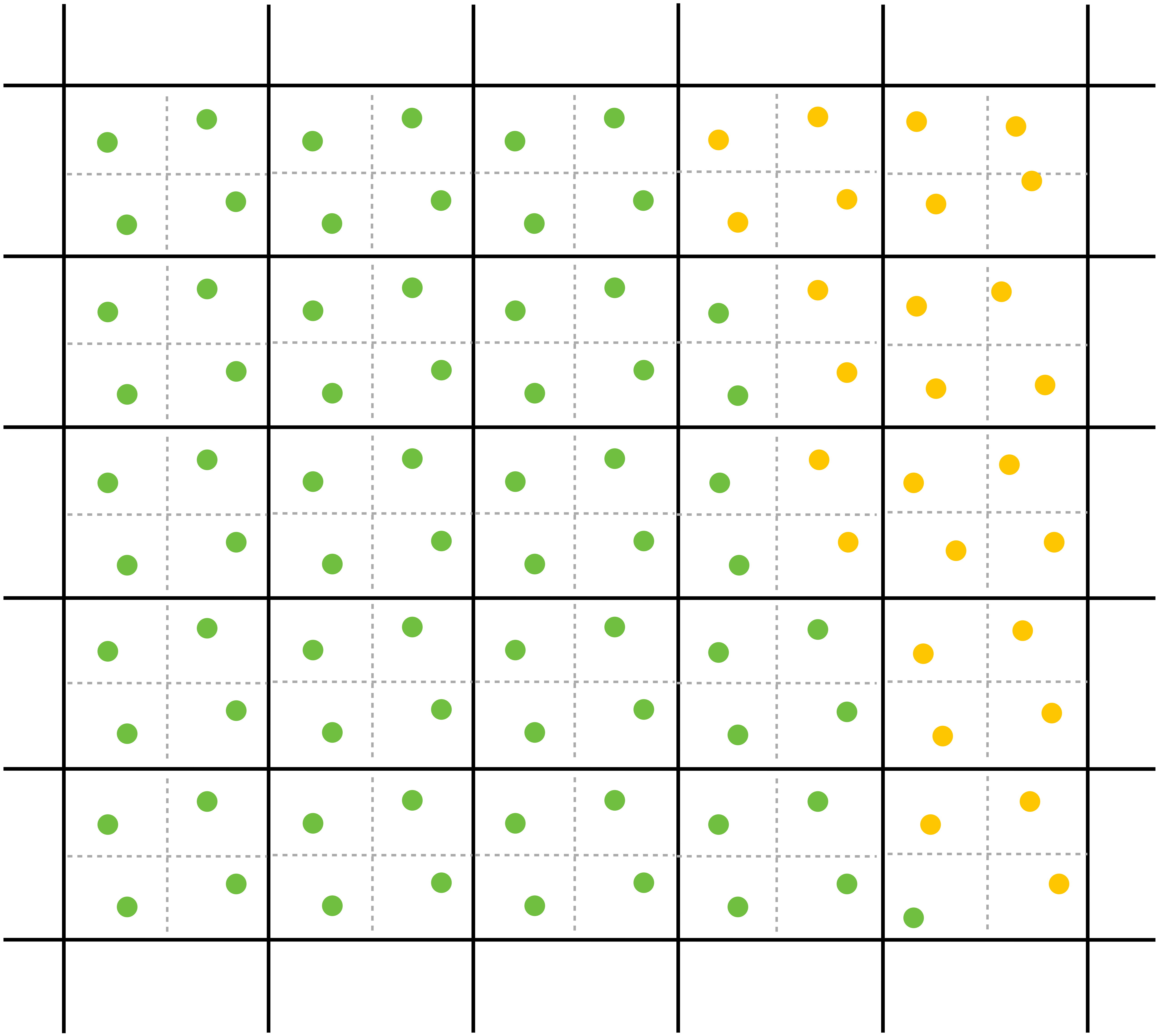


This example: green triangle occludes yellow triangle

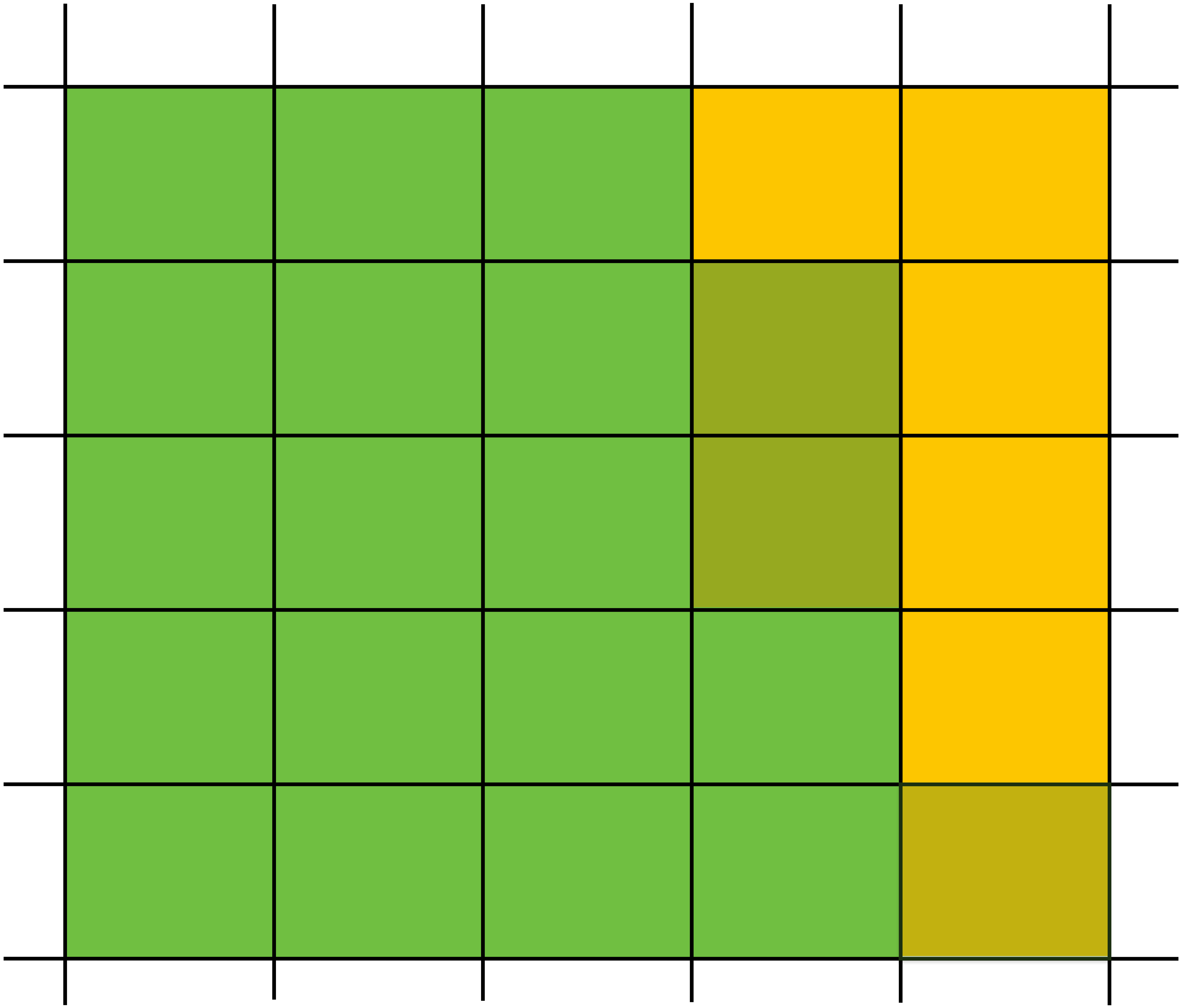
Color buffer contents



Color buffer contents (4 samples per pixel)



Final resampled result



Note anti-aliasing of edge due to filtering of green and yellow samples.

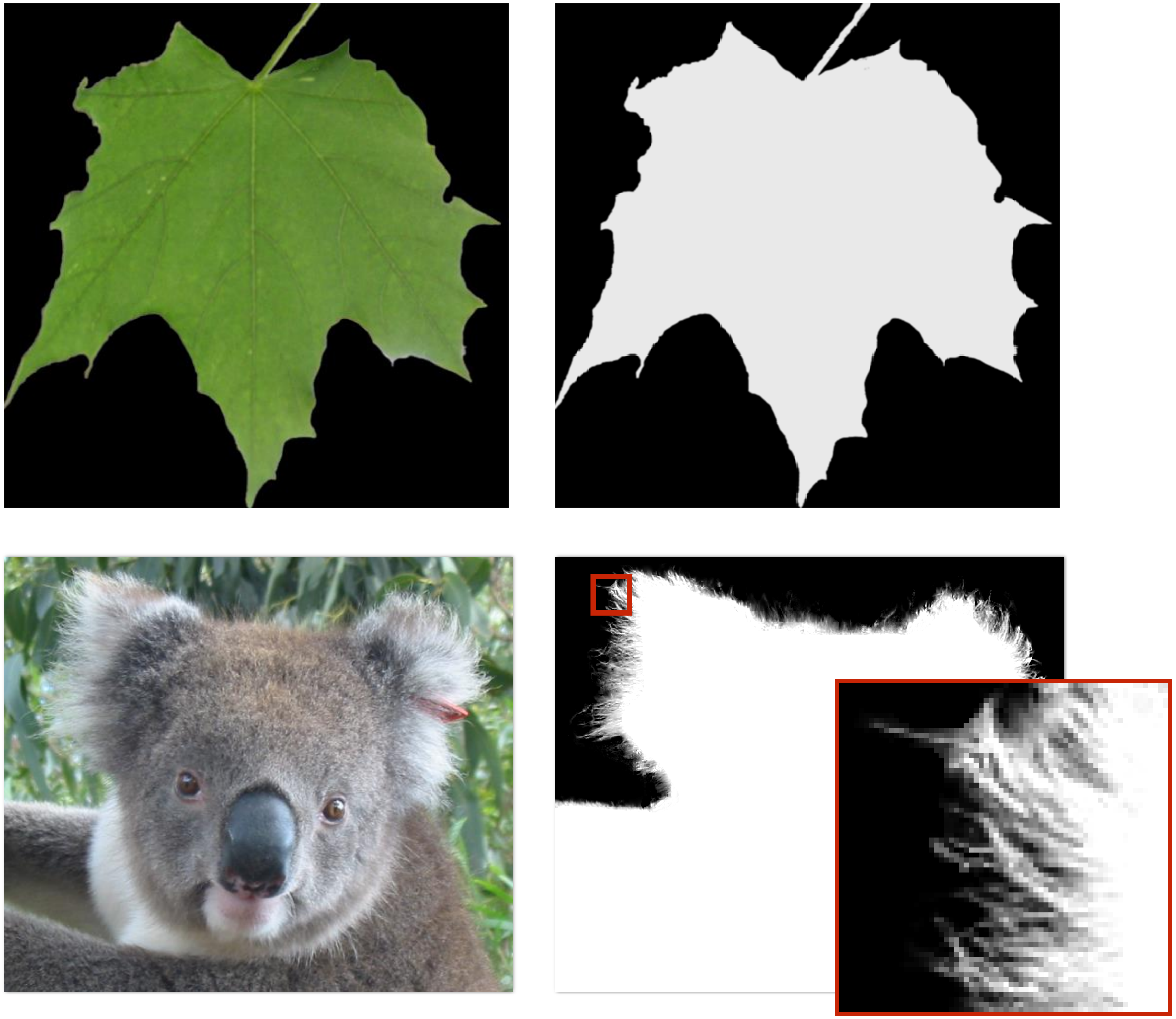
Summary: occlusion using a depth buffer

- Store one depth value per coverage sample (not per pixel!)
- Constant space per sample
  - Implication: constant space for depth buffer
- Constant time occlusion test per covered sample
  - Read-modify write of depth buffer if “pass” depth test
  - Just a read if “fail”
- Not specific to triangles: only requires that surface depth can be evaluated at a screen sample point
- Range of depth values is limited. That’s why the near and far planes are used in defining the view frustum!

But what about semi-transparent objects?

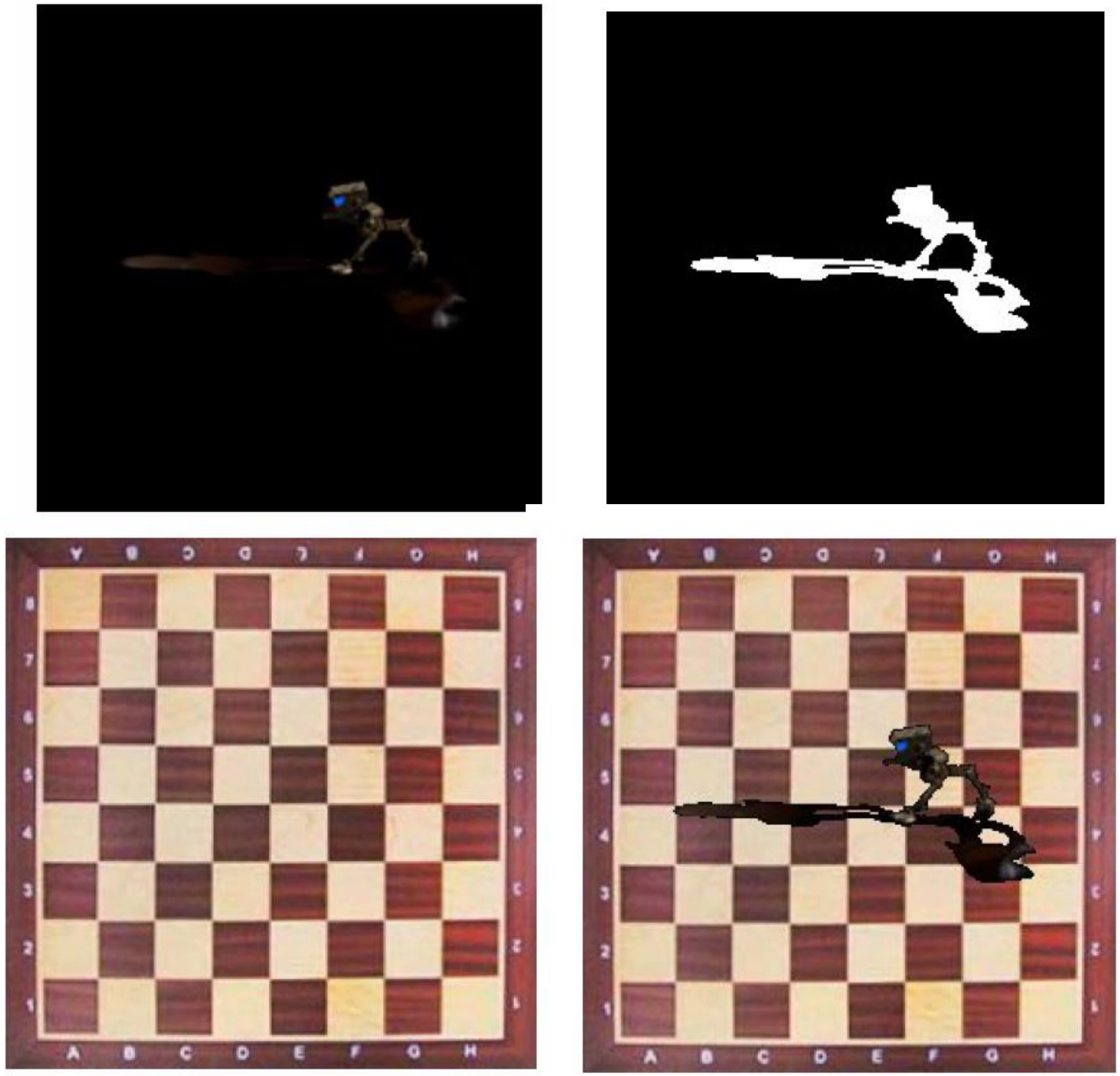
Compositing

Alpha: additional channel of image (rgba)



α of foreground object

Just replacing pixels rarely works



Binary mask

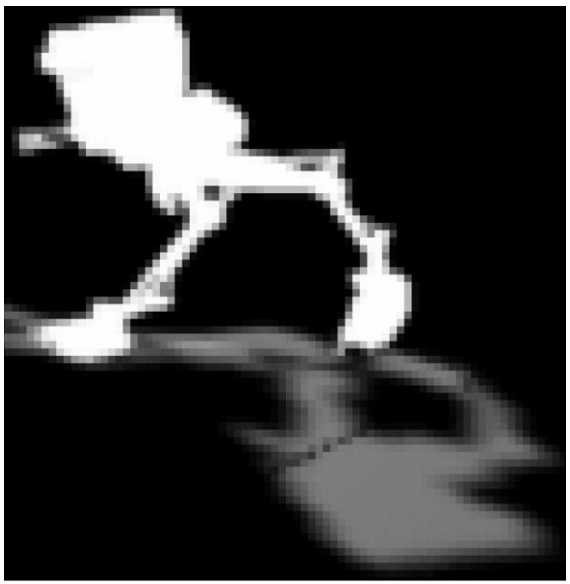
Problems: boundries & transparency (shadows)

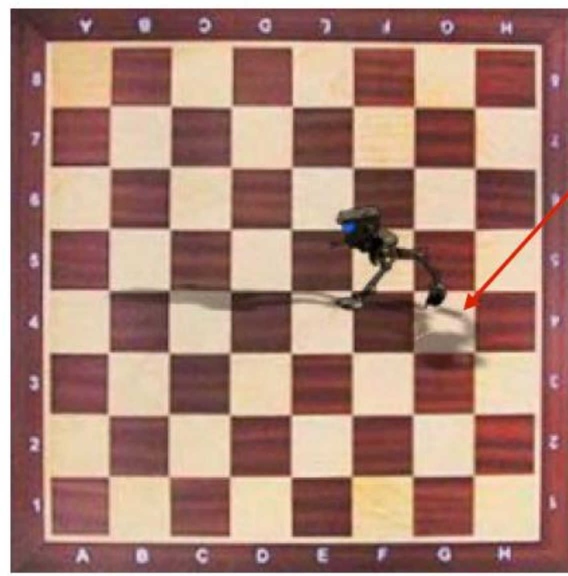
Alpha Blending

$$I_{comp} = \alpha I_{fg} + (1-\alpha) I_{bg}$$

alpha
mask





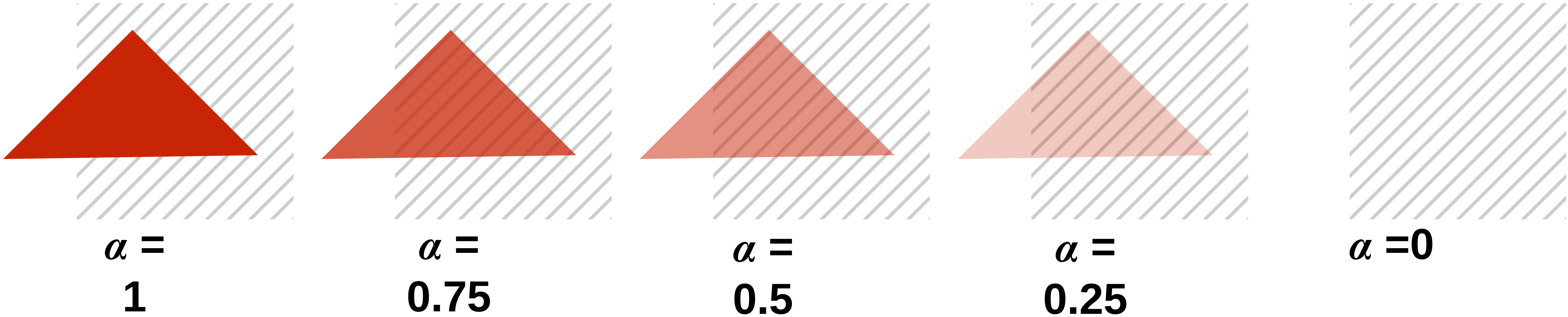
shadow

Representing opacity as alpha

Alpha describes the opacity of an object

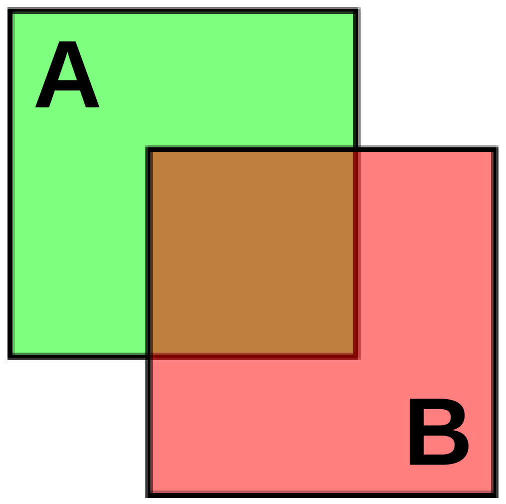
- **Fully opaque surface: α = 1**
- **50% transparent surface: α = 0.5**
- **Fully transparent surface: α = 0**

Red triangle with decreasing opacity

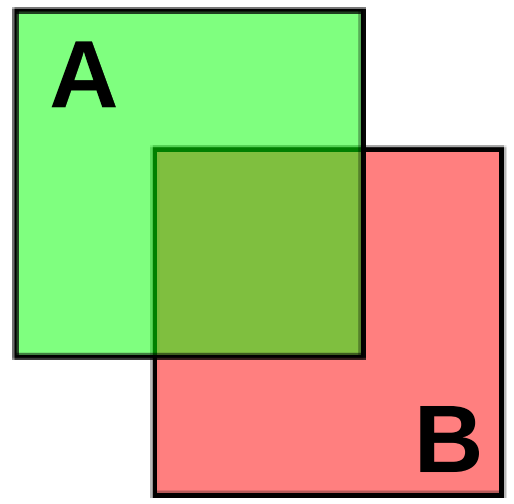


“Over" operator

Composite image B with opacity α_B over image A with opacity α_A



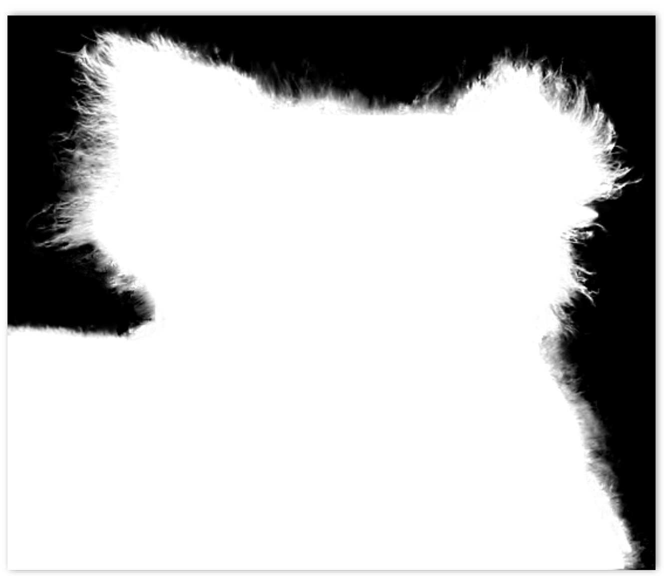
B over A

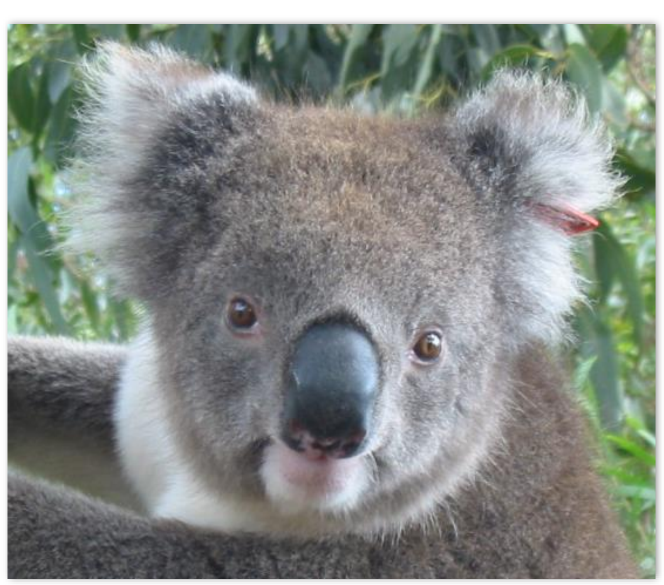


A over B

A over B != B over A

“Over” is not commutative



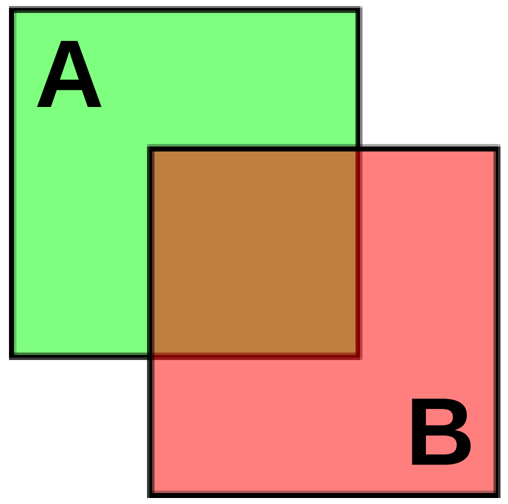


Koala over NYC

"Over" operator

Composite image B with opacity α_B over image A with opacity α_A

$$A = \begin{bmatrix} A_r & A_g & A_b \end{bmatrix}^T$$
$$B = \begin{bmatrix} B_r & B_g & B_b \end{bmatrix}^T$$



B over A

Composited color:

$$C = \underbrace{\alpha_B B}_{\text{Appearance of semi-transparent B}} + \underbrace{(1 - \alpha_B)}_{\text{What B lets through}} \overbrace{\alpha_A A}^{\text{Appearance of semi-transparent A}}$$

A over B != B over A

"Over" is not commutative

What is α_C?

$$\alpha_C = \alpha_B + (1 - \alpha_B)\alpha_A$$

“Over" operator

Composite image B with opacity α_B over image A with opacity α_A

First attempt:

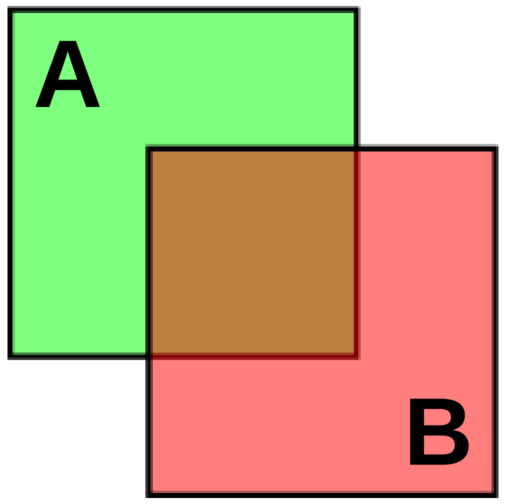
$$A = \begin{bmatrix} A_r & A_g & A_b \end{bmatrix}^T$$

$$B = \begin{bmatrix} B_r & B_g & B_b \end{bmatrix}^T$$

$$C = \alpha_B B + (1 - \alpha_B)\alpha_A A$$

two multiplies, one add (referring to vector ops on colors)

$$\alpha_C = \alpha_B + (1 - \alpha_B)\alpha_A$$



B over A

Premultiplied alpha (equivalent):

$$A' = \begin{bmatrix} \alpha_A A_r & \alpha_A A_g & \alpha_A A_b & \alpha_A \end{bmatrix}^T$$

$$B' = \begin{bmatrix} \alpha_B B_r & \alpha_B B_g & \alpha_B B_b & \alpha_B \end{bmatrix}^T$$

$$C' = B + (1 - \alpha_B)A$$

one multiply, one add

Color buffer update: semi-transparent surfaces

Color buffer values and tri_color are represented with premultiplied alpha

```
over(c1, c2) {
   return c1 + (1-c1.a) * c2;
}

update_color_buffer(tri_d, tri_color, x, y) {

   if (pass_depth_test(tri_d, zbuffer[x][y]) {
       // update color buffer
       // Note: no depth buffer update
       color[x][y] = over(tri_color, color[x][y]);
  }
}
```

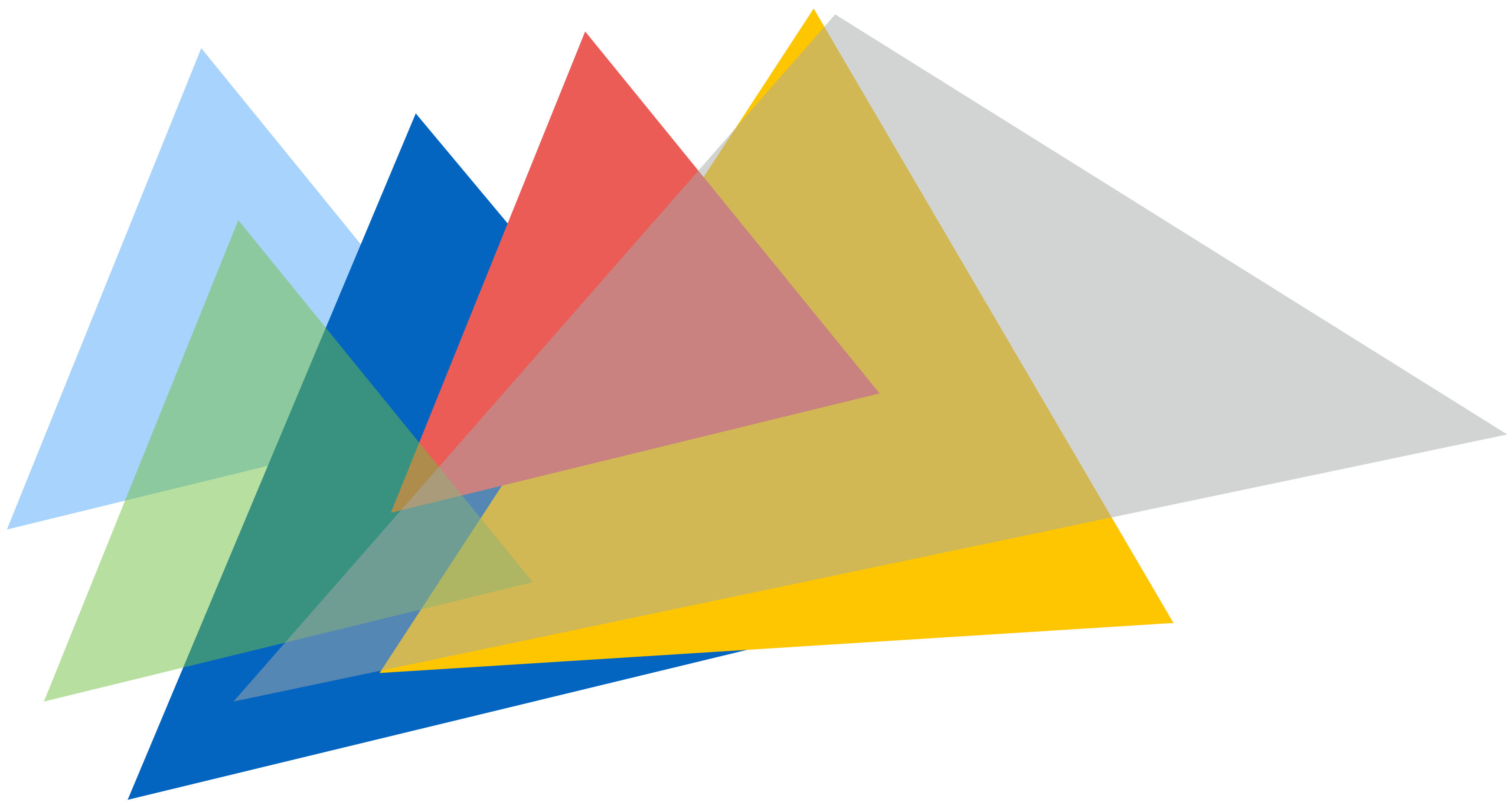
Q: What is the assumption made by this implementation?

Triangles must be rendered in back to front order!

Rendering a mixture of opaque and transparent triangles

Step 1: render opaque surfaces using depth-buffered occlusion (If pass depth test passed, triangle overwrites value in color buffer at sample)

Step 2: disable depth buffer update, render semi-transparent surfaces in back-to-front order. If depth test passed, triangle is composited OVER contents of color buffer at sample



Putting it all together

Unreal Engine Kite Demo (Epic Games 2015)

Summary

- Occlusion resolved independently at each screen sample using the depth buffer

- Alpha compositing for semi-transparent surfaces
  - Premultiplied alpha forms simply repeated composition
  - "Over" compositing operations is not commutative: requires triangles to be processed in back-to-front (or front-to-back) order

- Graphics pipeline:
  - Structures rendering computation as a sequence of operations performed on vertices, primitives (e.g., triangles), fragments, and screen samples
  - Behavior of parts of the pipeline is application-defined using shader programs.
  - Pipeline operations implemented by highly optimized parallel processors and fixed-function hardware (GPUs)