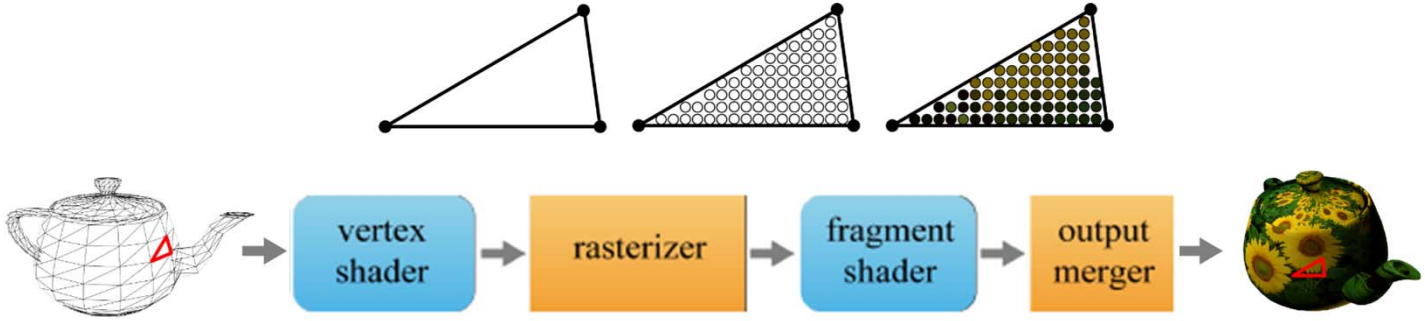
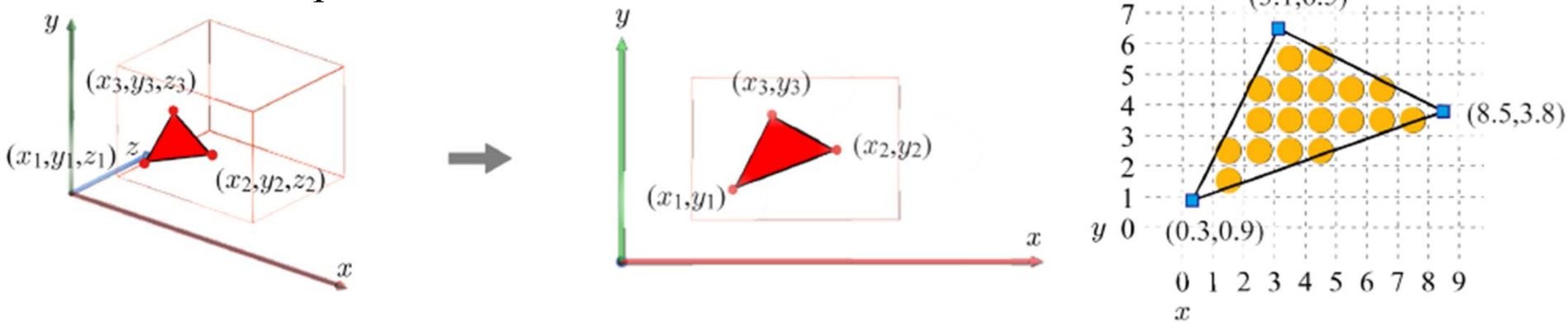
Chapter X
Output Merger

Where are We?

- We've just left the fragment shader that operates on each fragment and determines its color through various operations such as texturing and lighting.



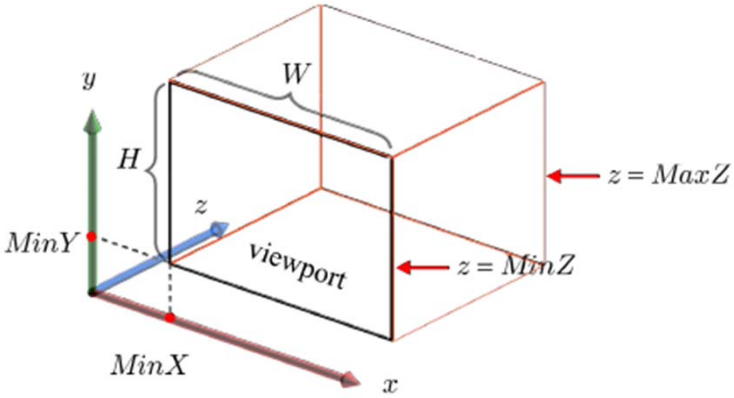
- The fragment shader has processed screen-space fragments and we are still in the screen space.



Color and Depth Buffers

- GL supports three kinds of buffers.
  - Color buffer - a memory space storing the pixels to be displayed on the screen
  - Depth buffer or z-buffer – This has the same resolution as the color buffer, and records the z-values of the pixels currently stored in the color buffer.
  - Stencil buffer – skipped for now

- Given the following viewport, we have *W*x*H*-resolution color and depth buffers.



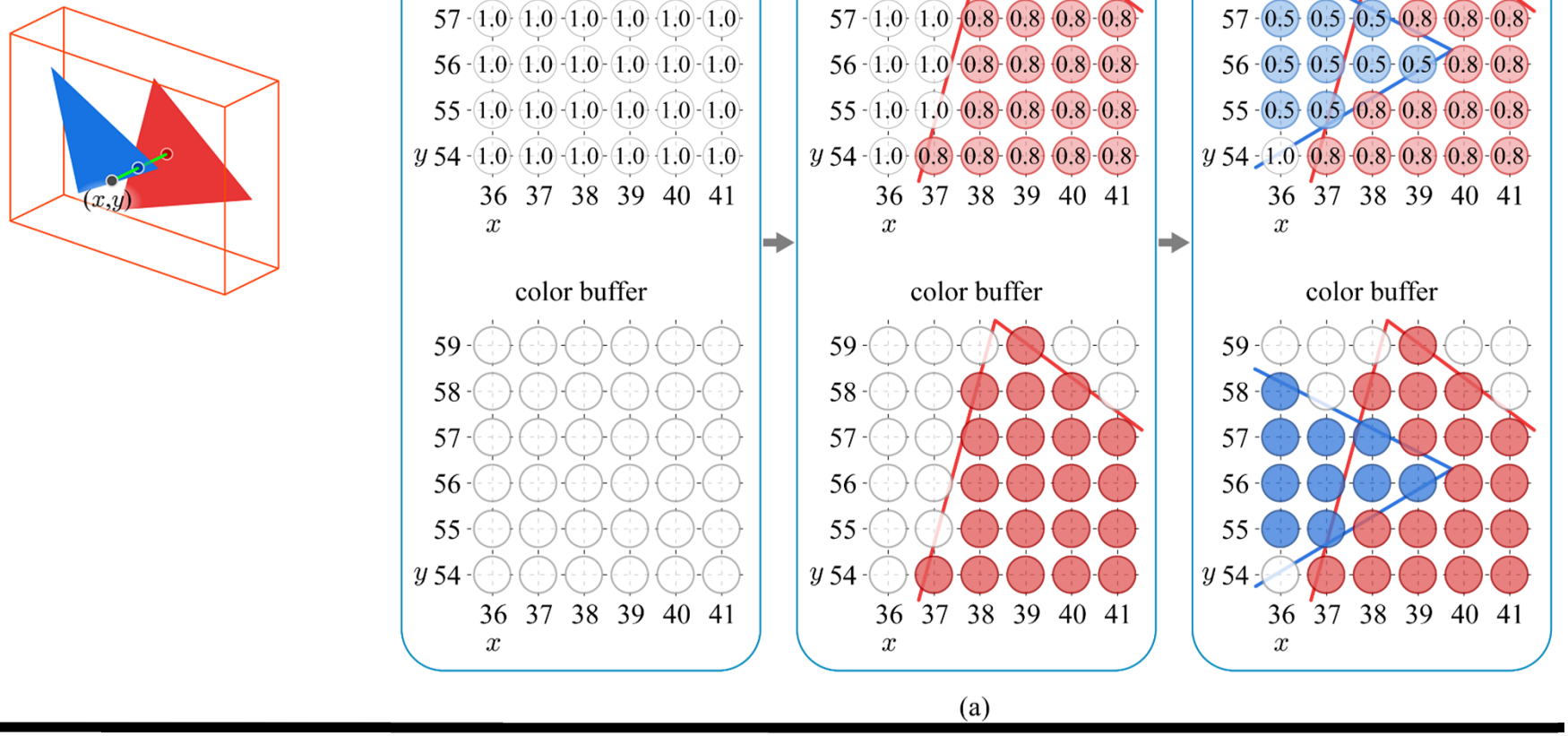
Z-buffering

- The output of the fragment shader is often called the RGBAZ fragment.
  - A: alpha for representing the fragment's opacity
  - Z: depth
- Using alpha and depth values, the fragment competes or is merged with the pixel of the color buffer.

- Z-buffering
  - When a fragment at (x,y) is passed from the fragment shader, its z-value is compared with the z-buffer value at (x,y).
  - If the fragment has a smaller z-value, its color and z-value are used to update the color buffer and z-buffer at (x,y), respectively.
  - Otherwise, the fragment is discarded.

Z-buffering (cont'd)

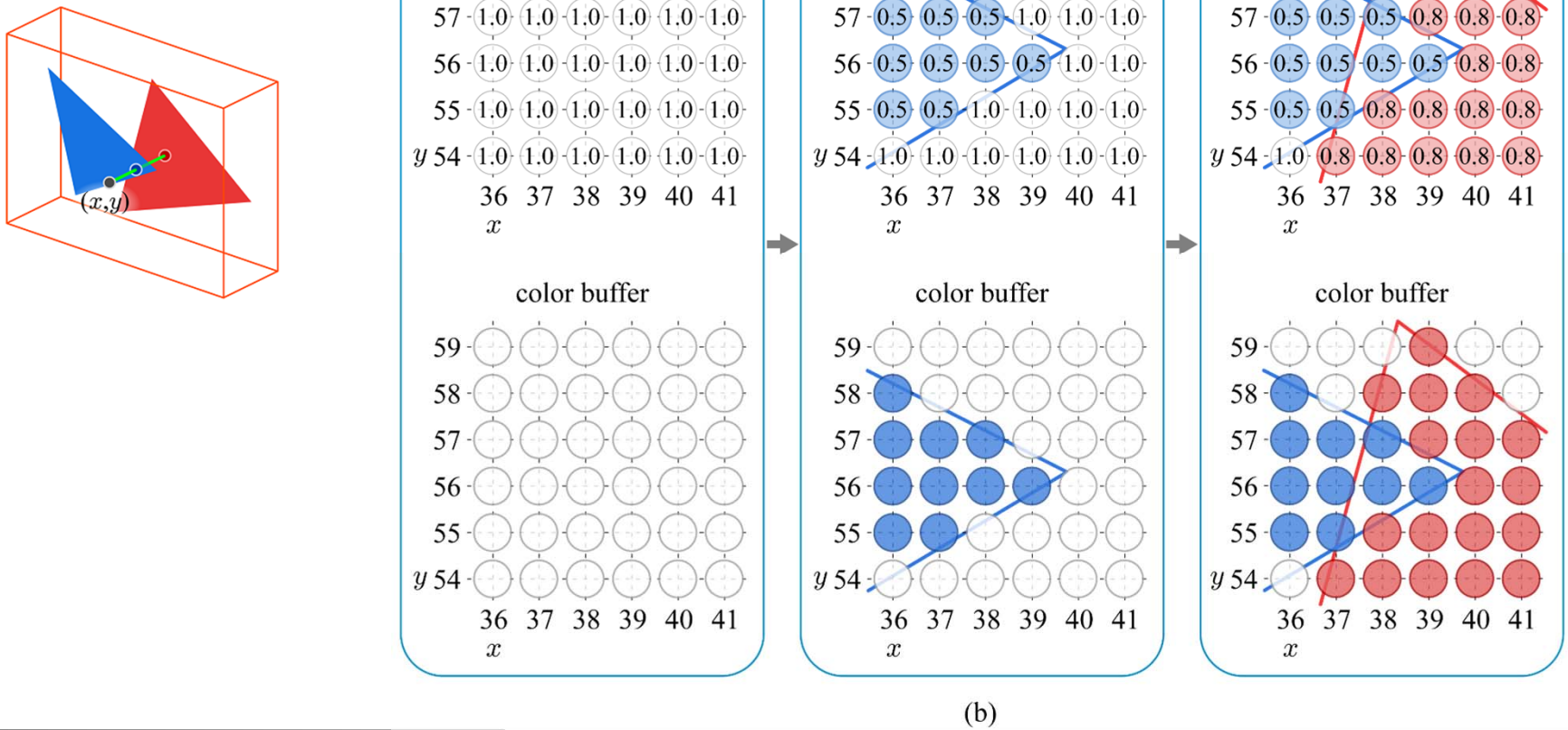
- Assume *MaxZ* is 1.0, the red triangle's depth is 0.8, and the blue triangle's is 0.5.



(a)

Z-buffering (cont'd)

- Rendering-order independence!!

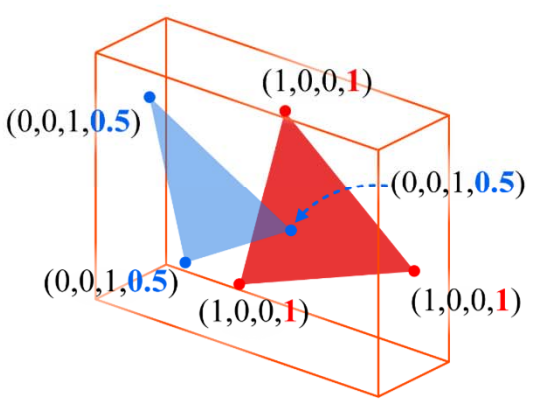


Z-buffering in GL

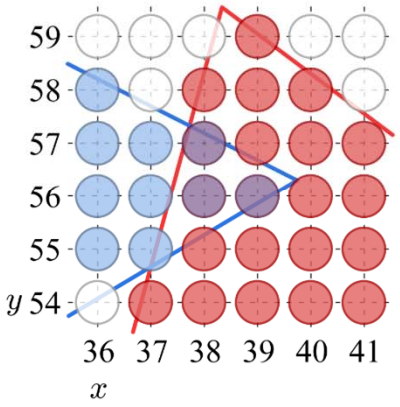
- By calling `glClear` once per frame, those buffers are simultaneously cleared with the default values selected by the following:
  - `glClearColor`
  - `glClearDepthf`
  - `glClearStencil`
- All tests and operations on fragments are disabled by default and are enabled by calling `glEnable` with the following arguments, for example:
  - `GL_DEPTH_TEST`
  - `GL_BLEND` (for blending fragments with colors stored in the color buffer)
- `glDepthFunc(GLenum func)`
  - `func` specifies the conditions under which the fragment is drawn.
  - The default is `GL_LESS`, which means that the fragment is drawn if its depth is less than that of the pixel.
  - It can be `GL_LESS, GL_GREATER, GL_LEQUAL, GL_GEQUAL, GL_EQUAL, GL_NOTEQUAL, GL_ALWAYS,` or `GL_NEVER.`

Alpha Blending

- The alpha channel in the range of [0,1]
  - 0 denotes "fully transparent."
  - 1 denotes "fully opaque."
- A typical blending equation is described as follows: $c = \alpha c_f + (1 - \alpha) c_p$



(a)



(b)

- For alpha blending, the primitives cannot be rendered in an arbitrary order. They must be rendered *after* all opaque primitives, and in *back-to-front order*. Therefore, the partially transparent objects should be *sorted*.

Alpha Blending in GL

- `glBlendFunc(GLenum sfactor, GLenum dfactor);`
  - `sfactor` specifies the blending coefficient for the incoming (source) fragment
  - `dfactor` specifies the blending coefficient for the destination pixel
- Many values such as `GL_ZERO` and `GL_ONE` can be assigned to `sfactor` and `dfactor`, but `GL_SRC_ALPHA` may best fit to `sfactor` and `GL_ONE_MINUS_SRC_ALPHA` to `dfactor`.
- Once the fragment and pixel color are multiplied by `sfactor` and `dfactor`, respectively, they are combined using the operator specified by `glBlendEquation`.
  - Its argument can be `GL_FUNC_ADD`, `GL_FUNC_SUBTRACT`, `GL_FUNC_REVERSE_SUBTRACT`, `GL_MIN`, or `GL_MAX`.
  - The default is `GL_FUNC_ADD`.