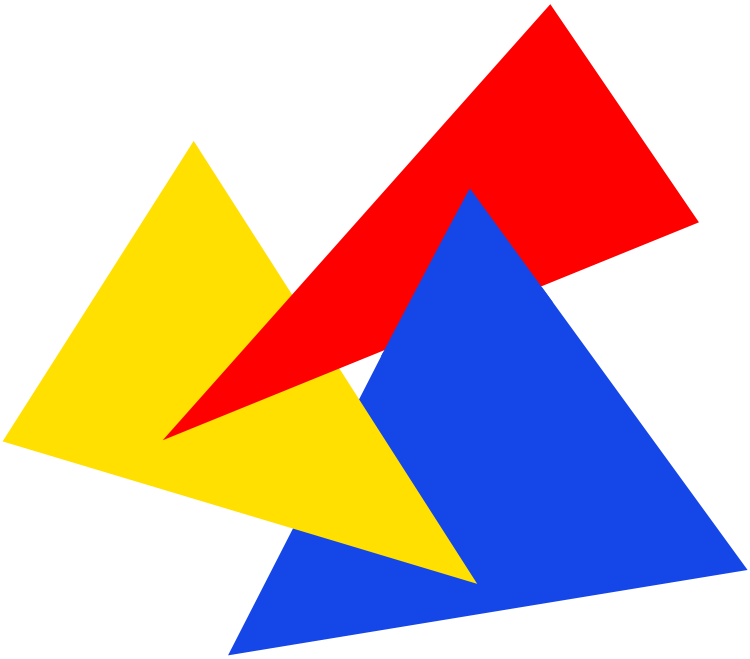
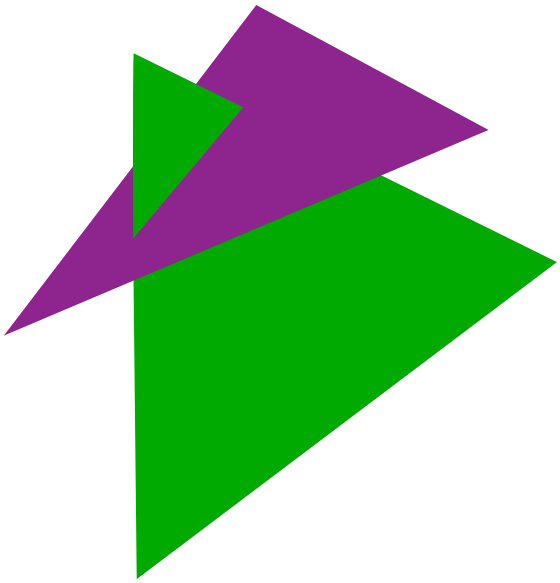
Exercises 08 (Depth and Transparency)

Thinking deeply about transparency

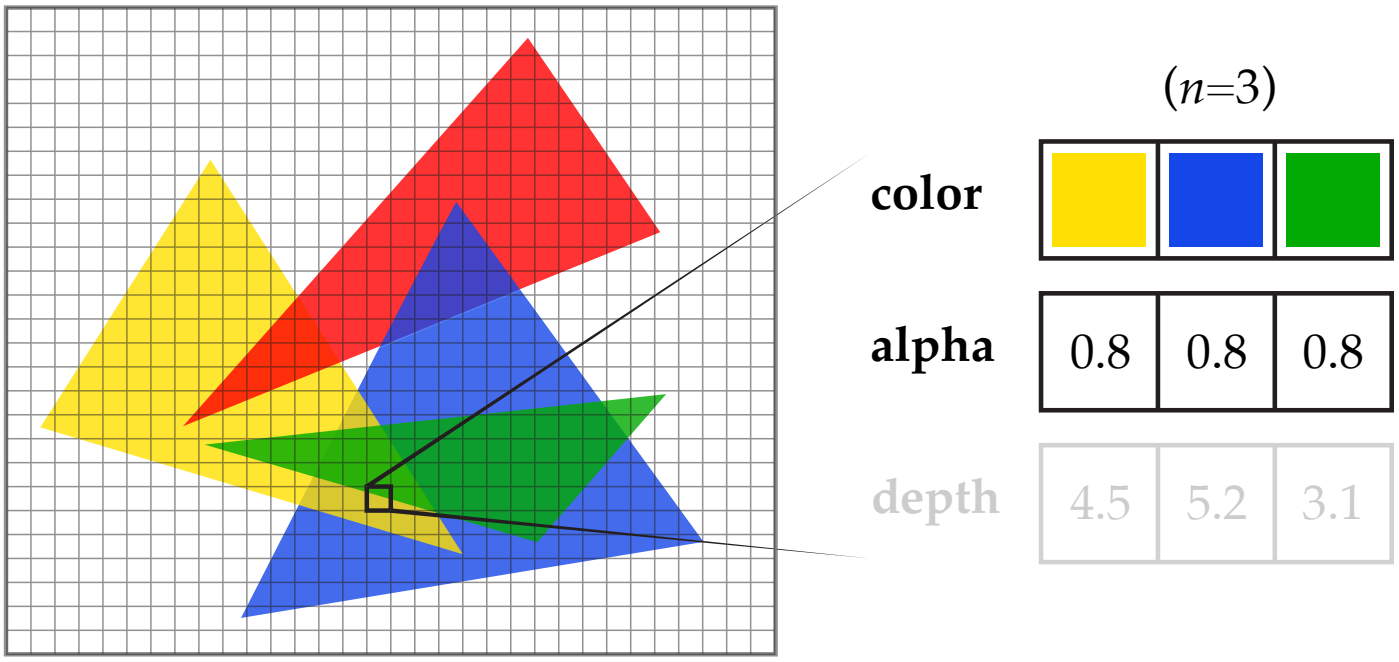
Correctly rasterizing scenes with both depth and transparency is tricky. In these exercises we'll think carefully about strategies that will (and won't) work for getting a correct image---and also think about how to use depth buffering and transparency to achieve interesting effects beyond ordinary transparency.





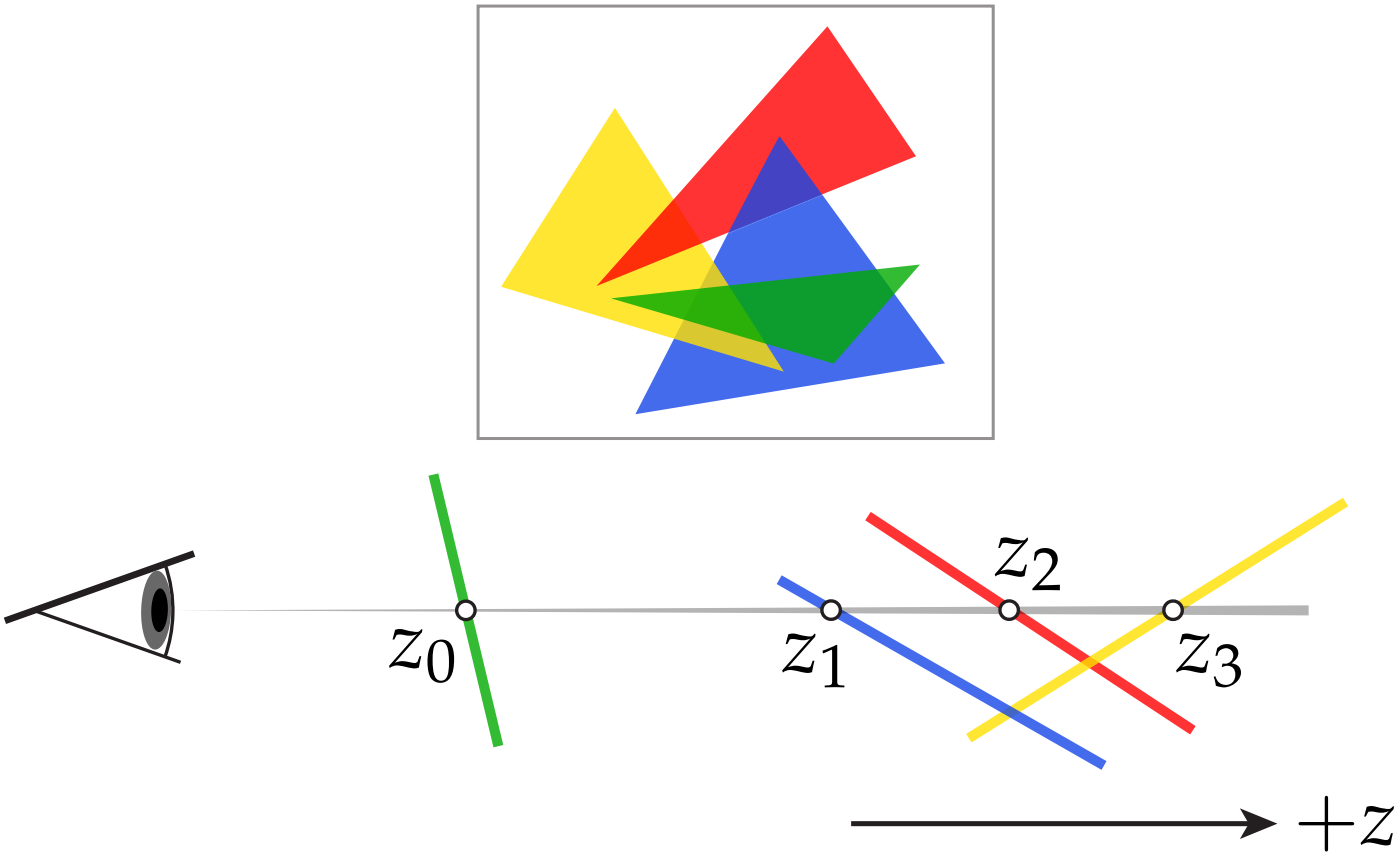
1. One problem the depth buffer nicely solves is the fact that it may be impossible to order triangles by depth---consider for instance the overlapping triangles above. In some cases, however, you may still need to draw triangles in an environment that does not support depth buffering. For instance, the images above are stored in the SVG format, which does not support depth---much less depth buffering! How could you still correctly draw the overlapping triangles? Sketch a general procedure, being detailed as possible. Your solution does not need to be efficient, but it should be correct. Think about the different steps of processing that are really needed to implement an end-to-end algorithm.

2. In lecture, we mentioned that one possibility is to rasterize opaque triangles using depth buffering (in any order), then rasterize semi-transparent triangles using the "over" blending operator (in back-to-front order). But as just seen in the previous problem, organizing triangles in back-to-front order can be really tough! For this reason, there are a variety of *order independent transparency* techniques that produce the correct image for primitives drawn in any order. One idea is to store, for every pixel, a *list* of RGBA values giving the color and transparency of every primitive that has ever covered that pixel. Is this information sufficient to produce a correct image? Why or why not? Assume we want to blend colors using the over operator.



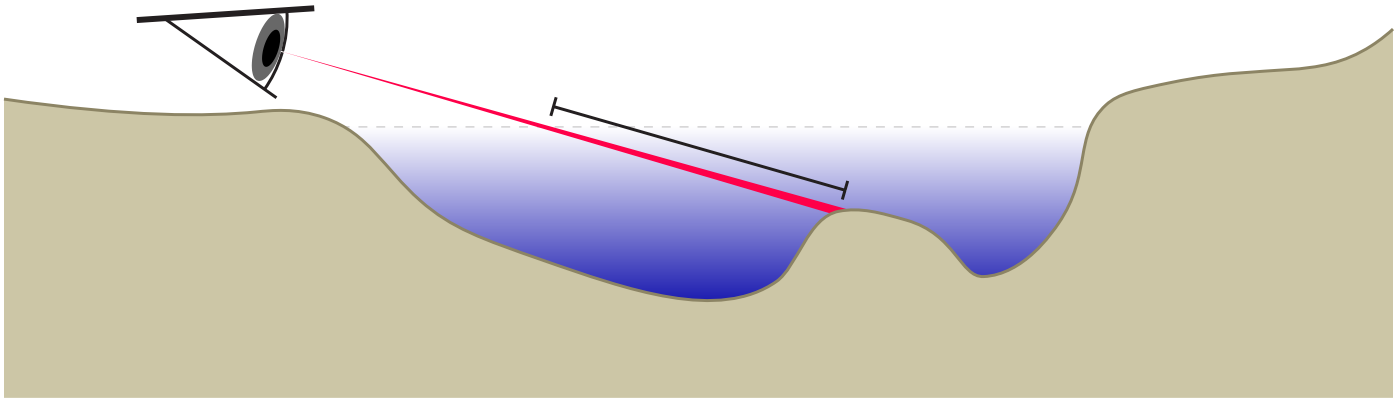
3. Another possibility is to use "alpha-buffering", which for each pixel stores a list of RGBA and depth values. Suppose you are given an array `color` of RGB values, an array `alpha` of alpha values, and an array `depth` of z-values, all of the same length `n` (and all from the same pixel). Give a short algorithm to compute the correct color value as determined by the over operator.

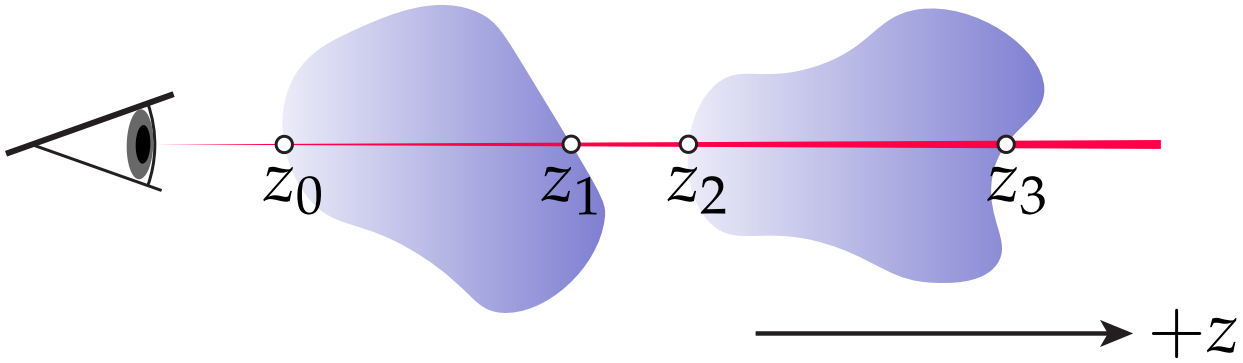
4. Why do you think alpha buffering might be difficult/slow/costly to implement in real graphics hardware? Give at least two reasons.



5. An alternative order-independent strategy, used quite often in practice, is "depth peeling." The basic idea is to render the triangles multiple times, in several "passes." Each pass produces a new depth and color buffer. This depth and color information can then be used to modify the way we rasterize primitives in the next pass. By intelligently ordering these passes, and thinking about how we perform the depth test, we can accumulate the correct image after enough passes. Describe a depth peeling strategy that produces the correct image (there are several possibilities).

6. Suppose you wanted to render a sandy beach with clear blue water. One thing you notice about water is that it gets darker as it gets deeper. How can you use Z-buffering and transparency to approximate this "attenuation" effect, and rasterize this serene ocean vista?





7. Is it possible or impossible to correctly render attenuation without the depth buffer? For simplicity, assume there is no beach---just a few splashes of water flying up against the sky.
