1

Notes

2

Atop

- The simplest (useful) and most common form of compositing: put one image "atop" another
  - Image 1 (RGB) on top of image 2 (RGB)
- For each pixel in the final composite, need to know what RGB value to take
  - Where image 1 is opaque, choose RGB_1
  - Where image 1 is "empty", choose RGB_2
  - Where image 1 is partly transparent, or where image 1 only covers part of the pixel?

3

Alpha

- We add another channel, alpha: RGBA
- Encodes whether the pixel of the image is empty (alpha=0) or opaque (alpha=1) or something in between (0<alpha<1)
  - Most important case: at the edges of objects
- When we render a layer, we compute and save alpha along with RGB
  - Or if it's real action, use a "blue screen" behind the actors, estimate alpha
- Premultiplied alpha: instead of storing regular RGB + alpha, store rgb=alpha*R, alpha*G, alpha*B and alpha
  - Simplifies formulas to come

4

Atop operation

- Image 1 "atop" image 2
- Assume independence of sub-pixel structure
  - So for each final pixel, a fraction $alpha_1$ is covered by image 1
  - Rest of final pixel (a fraction of 1-$alpha_1$) is covered partly by image 2 (fraction $alpha_2$) and partly uncovered
- Without premultiplied alpha:
  - $R_{final}=alpha_1*R_1 + (1-alpha_1)*alpha_2*R_2$
  - $G_{final}=alpha_1*G_1 + (1-alpha_1)*alpha_2*G_2$
  - $B_{final}=alpha_1*B_1 + (1-alpha_1)*alpha_2*B_2$
  - $alpha_{final}=alpha_1 + (1-alpha_1)*alpha_2$

5

Premultiplied

- Using standard premultiplied alpha, formulas simplify:
  - $R_{final}=r_1 + (1-alpha_1)*r_2$
  - $G_{final}=g_1 + (1-alpha_1)*g_2$
  - $B_{final}=b_1 + (1-alpha_1)*b_2$
  - $alpha_{final}=alpha_1 + (1-alpha_1)*alpha_2$
- And of course store the result premultiplied:
  - $r_{final}=alpha_{final}*R_{final}$
  - $g_{final}=alpha_{final}*G_{final}$
  - $b_{final}=alpha_{final}*B_{final}$

6

Note on gamma

- Recall gamma: how nonlinear a particular display is
  - When you send a signal for fraction x of full brightness, actual brightness output from display is a nonlinear function of x
    - Called gamma since usually modeled as x^{γ}
  - For final image, for a particular display, should correct for gamma
- But when we're taking linear combinations of RGB values, need to do it before gamma correction!
  - Similarly for real life elements, camera output is distorted, needs to be undone before compositing

7

3D Rendering

8

Sampling and Filtering

- For high quality images need to do
  - Antialiasing - no jaggies
  - Motion blur - no strobing
  - Possibly depth-of-field - no pinhole camera
- Boils down to:
  - Each pixel gets light from a number of different objects, places, times
- Figuring out where: point sampling
  - Find light at a particular place in the pixel, at a particular time, ...
- Combining the nearby point samples into RGBA for each pixel: filtering
  - Simplest is box filter (average the samples in pixel)

How to get point samples

- Three big rendering algorithms
  - Z-buffer / scanline
    - Graphics Hardware - OpenGL etc.
  - Ray tracing
    - Highly accurate rendering
    - Difficult models (e.g. volumetric stuff)
  - REYES
    - Almost everything you see in film/TV

REYES

- Invented at Lucasfilm (later Pixar) by Cook et al. SIGGRAPH '87
- Geometry is diced up into grids of micropolygons (quads about one pixel big)
- Each micropolygon is "shaded" in parallel to get a colour+opacity (RGBA)
- Then sent to "hiding" to determine in which point samples it makes a contribution
- Each point sample keeps a sorted list of visible points, composites them together when done
- Filter blends point samples to get final pixels