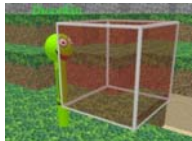


Computer Graphics

Blending

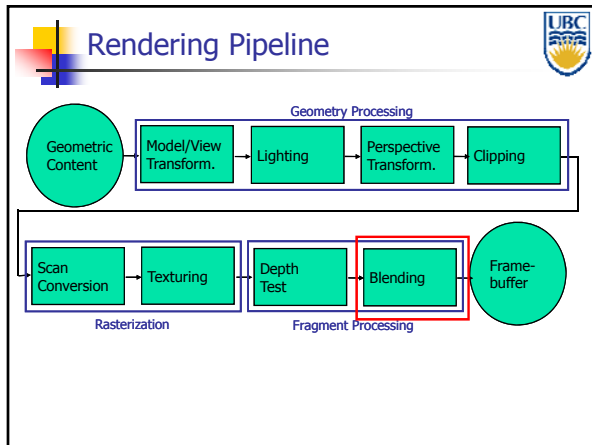
Chapter 13

Blending



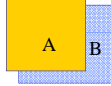
Alpha Blending (OpenGL)

- Parameters:
 - s = source color
 - d = destination color
 - b = source blend factor
 - c = dest blend factor
 - $d' = bs + cd$ (?)
- Where
 - "Source" means "color/alpha of currently rendered primitive"
 - "Destination" means framebuffer value

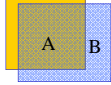


Over operator

- $d' = a_s s + (1-a_s)d$
- Examples: $a_A=1$ $a_B=0.4$



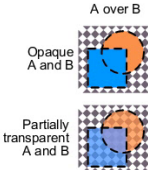
A over B:
 $d' = 1 * C_A + (1-1) * C_B$



B over A:
 $d' = 0.4 * C_B + (0.6) * C_A$

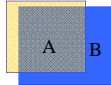
Blending

- How might you combine multiple elements?
 - New color A, old color B

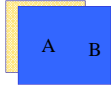


Over operator

- $d' = a_s s + (1-a_s)d$
- Examples: $a_A=0.4$ $a_B=1.0$



A over B:
 $d' = 0.4 * C_A + (0.6) * C_B$




B over A:
 $d' = 1 * C_B + (0) * C_A$


Comparison:
 $d' = bs + cd$

Computer Graphics


Blending





Over operator




- $d' = a_s s + (1-a_s)d$
- $a' = a_s a_s + (1-a_s) a_d$




Double Buffer




OpenGL Blending




- In OpenGL:
 - Enable blending
 - `glEnable(GL_BLEND)`
 - Specify alpha channel for colors
 - `glColor4f(r, g, b, alpha)`
 - Specify blending function
 - E.g: `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`
 - $C = \alpha_{new} * C_{new} + (1 - \alpha_{new}) * C_{old}$
 - Other options available




Double Buffering




- Framebuffer:
 - Piece of memory where the final image is written
 - Problem:
 - The display needs to read the contents, cyclically, while the GPU is already working on the next frame
 - Could result in display of partially rendered images on screen
 - Solution:
 - Have TWO buffers
 - Currently displayed (front buffer)
 - Render target for the next frame (back buffer)




OpenGL Blending



- Caveats:
 - Note: alpha blending is an order-dependent operation!
 - It matters which object is drawn first AND
 - Which surface is in front
 - For 3D scenes, this makes it necessary to keep track of rendering order explicitly
 - Possibly also viewpoint-dependent!
 - E.g. always draw "back" surface first
 - Also note: interaction with z-buffer



Double Buffering



- Front/back buffer:
 - Each buffer has both color channels and a depth channel
 - Important for advanced rendering algorithms
 - Doubles memory requirements!
- Switching buffers:
 - At end of rendering one frame, simply exchange the pointers to the front and back buffer
 - GLUT toolkit: `glutSwapBuffers()` function
 - Different functions under windows/X11 if not using GLUT