THE RENDERING PIPELINE

**Vertices and attributes**

- Vertex Shader:
  - Modelview transform
  - Per-vertex attributes

- Rasterization:
  - Scan conversion
  - Interpolation

- Per-Sample Operations:
  - Depth test
  - Blending

- Vertex Post-Processing:
  - Viewport transform
  - Clipping

- Fragment Shader:
  - Texturing/...
  - Lighting/shading

**Framebuffer**
SOME OF YOU WERE WONDERING...

• Why is depth test AFTER the fragment shader?
• Why bother with computing the color if it’s behind something?

• The answer is blending.
OPAQUE VS. TRANSPARENT

• If all objects are opaque, no blending is needed
• As before, simply overwrite the color in framebuffer
• Then depth test can be done BEFORE fragment shader
  • (if, of course, fragment shader does not modify z)
OPAQUE VS. TRASSPARENT

• For transparent objects, every time we’re writing into a fragment buffer, we need to consider what there is already
OPAQUE VS. TRASPARENT

- For transparent objects, every time we’re writing into a fragment buffer, we need to consider what there is already
- Per fragment:
  - Fragment’s color: source color
  - What’s in framebuffer: destination color
OPAQUE VS. TRANSPARENT

• For transparent objects, every time we’re writing into a fragment buffer, we need to consider what there is already

• Per fragment:
  • Fragment’s color: source color
  • What’s in framebuffer: destination color

• Same idea as layers in Photoshop

How to combine those 2 colors into some new color?
BLENDING: THERE ARE MANY WAYS.

• (switch to demo)
• http://threejs.org/examples/webgl_materials_blending.html
BLENDING EQUATIONS

- $D = (r, g, b, \alpha)_D$ - destination color (what’s already in framebuffer)
- $S = (r, g, b, \alpha)_S$ - source color (current fragment)
- $Out = (r, g, b, \alpha)_{out}$ - output color (result of blending)

Blending equations:

\[
Out.rgb = f_1(D.rgb, S.rgb)
\]
\[
Out.\alpha = f_2(D.\alpha, S.\alpha)
\]

\[\alpha = 1.0\] (opaque)
\[\alpha = 0.7\] (semi-transparent)
BLENDING EQUATIONS

Blending equations:
\[ \text{Out.rgb} = f_1(\text{D.rgb}, \text{S.rgb}) \]
\[ \text{Out.\(\alpha\)} = f_2(\text{D.\(\alpha\)}, \text{S.\(\alpha\)}) \]

A user chooses both \( f_1 \) and \( f_2 \) out of those options:

\[
\begin{align*}
  f(D, S) &= d \cdot D + s \cdot S \\
  f(D, S) &= d \cdot D - s \cdot S \\
  f(D, S) &= s \cdot S - d \cdot D \\
  f(D, S) &= \min(D, S) \\
  f(D, S) &= \max(D, S)
\end{align*}
\]

\( d, s \) – some parameters

\( D(S) \) – either \( D.rgb \) (\( S.rgb \)) or \( D.\alpha \) (\( S.\alpha \))
BLENDING EQUATIONS

A user chooses both $f_1$ and $f_2$ out of those options:

$$f(D,S) = d \cdot D + s \cdot S$$
$$f(D,S) = d \cdot D - s \cdot S$$
$$f(D,S) = s \cdot S - d \cdot D$$
$$f(D,S) = \min(D,S)$$
$$f(D,S) = \max(D,S)$$

D (S) – either D.rgb (S.rgb) or $D.\alpha (S.\alpha)$

And $d, s$ out of those:

$$d, s \in \{D.rgb, 1 - D.rgb, S.rgb, 1 - S.rgb, D.\alpha, 1 - D.\alpha, S.\alpha, 1 - S.\alpha \text{ constant}\}$$
WHAT CAN WE DO WITH THOSE?

• Simple transparency (“over operator”):
  • \( f_1 = ADD, f_2 = ADD \)
  • \( d_1 = 1 - S.\alpha \)
  • \( s_1 = S.\alpha \)
  • \( d_2 = 0 \)
  • \( s_2 = 1 \)

\[
\text{Out.rgb} = (1 - S.\alpha) \cdot D.rgb + S.\alpha \cdot S.rgb
\]
\[
\text{Out.}\alpha = 0 \cdot D.\alpha + 1 \cdot S.\alpha
\]

\( \alpha = 1.0 \) (opaque) \hspace{2cm} \alpha = 0.7 \) (semi-transparent)
WHAT CAN WE DO WITH THOSE?

• Simple transparency (“over operator”):
  
  • $f_1 = ADD, f_2 = ADD$
  • $d_1 = 1 - S.\alpha$
  • $s_1 = S.\alpha$
  • $d_2 = 0$
  • $s_2 = 1$

  \[
  \text{rgb: } (1 - 0.7) \cdot (0,0,1) + 0.7 \cdot (1,1,0) = (0,0,0.3) + (0.7,0.7,0) = (0.7,0.7,0.3)
  \]

  \[
  \text{Out.rgb} = (1 - S.\alpha) \cdot D.rgb + S.\alpha \cdot S.rgb
  \]
  \[
  \text{Out.}\alpha = 0 \cdot D.\alpha + 1 \cdot S.\alpha
  \]

$\alpha = 1.0$ (opaque)

$\alpha = 0.7$ (semi-transparent)
OVER OPERATOR

\[ Out.rgb = (1 - S.\alpha) \cdot D.rgb + S.\alpha \cdot S.rgb \]

- Examples: \( A.\alpha = 1, B.\alpha = 0.4 \)

A over B:
\[ Out.rgb = (1) \cdot A.rgb + (1 - 1) \cdot B.rgb \]

B over A:
\[ Out.rgb = (0.4) \cdot A.rgb + (1 - 0.4) \cdot B.rgb \]
OVER OPERATOR

\[ \text{Out.rgb} = (1 - S.\alpha) \cdot D.rgb + S.\alpha \cdot S.rgb \]

- Examples: A. \(\alpha = 0.4\), B. \(\alpha = 1\)

A over B:
\[ \text{Out.rgb} = (1 - 0.4) \cdot A.rgb + (0.4) \cdot B.rgb \]

B over A:
\[ \text{Out.rgb} = (0) \cdot A.rgb + (1) \cdot B.rgb \]
WHAT CAN WE DO WITH THOSE?

• “Multiply”
  • \( f_1 = ADD, f_2 = ADD \)
  • \( d_1 = S.rgb \)
  • \( s_1 = 0 \)
  • \( d_2 = 0 \)
  • \( s_2 = 1 \)

\[
\text{Out.rgb} = S.rgb \cdot D.rgb \\
\text{Out.} \alpha = 0 \cdot D.\alpha + 1 \cdot S.\alpha
\]
WHAT CAN WE DO WITH THOSE?

• “Darken”
  • $f_1 = \text{MIN}$, $f_2 = \text{ADD}$
  • $d_1 = 1$
  • $s_1 = 1$
  • $d_2 = 0$
  • $s_2 = 1$

\[
\begin{align*}
\text{Out.rgb} &= \min(S.rgb, D.rgb) \\
\text{Out.}\alpha &= S.\alpha
\end{align*}
\]

\[\alpha = 1.0 \text{ (opaque)}\]
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
    • E.g. always draw “back” surface first
The same idea can be used even when objects are opaque.
BLENDING EXAMPLE

- The same idea can be used even when objects are opaque
- Boundary pixels are now a bit transparent => smooth border
BLENDING/COMPOSITING IN VFX

• e.g. https://www.youtube.com/watch?v=63o0QJ3CjtY