Sort-Independent Alpha Blending

Houman Meshkin
Senior Graphics Engineer
Perpetual Entertainment
hmeshkin@perpetual.com
Alpha blending is used to show translucent objects. Translucent objects render by blending with the background. Opaque objects just cover the background.
Varying alpha
Blending order

- The color of a translucent pixel depends on the color of the pixel beneath it. It will blend with that pixel, partially showing its color, and partially showing the color of the pixel beneath it.
- Translucent objects must be rendered from far to near.
Challenge

- It’s very complex and complicated to render pixels from far to near
- Object-center sorting is common
  still can be time consuming
- Object sorting doesn’t guarantee pixel sorting
  objects can intersect each other
  objects can be concave
  pixel sorting is required for correctness
The Formula

- C0: foreground RGB color
- A0: alpha representing foreground’s translucency
- D0: background RGB color
- FinalColor = A0 * C0 + (1 – A0) * D0
  as A0 varies between 0 and 1, FinalColor varies between D0 and C0
Multiple translucent layers
Formula for multiple translucent layers

- **Cn**: RGB from nth layer
- **An**: Alpha from nth layer
- **D0**: background
- **D1** = \( A_0 \times C_0 + (1 - A_0) \times D_0 \)
- **D2** = \( A_1 \times C_1 + (1 - A_1) \times D_1 \)
- **D3** = \( A_2 \times C_2 + (1 - A_2) \times D_2 \)
- **D4** = \( A_3 \times C_3 + (1 - A_3) \times D_3 \)
Expanding the formula

\[ D_4 = A_3 \times C_3 + A_2 \times C_2 \times (1 - A_3) + A_1 \times C_1 \times (1 - A_3) \times (1 - A_2) + A_0 \times C_0 \times (1 - A_3) \times (1 - A_2) \times (1 - A_1) + D_0 \times (1 - A_3) \times (1 - A_2) \times (1 - A_1) \times (1 - A_0) \]
Further expanding…

\[ D_4 = A_3 \cdot C_3 \]

\[ + A_2 \cdot C_2 - A_2 \cdot A_3 \cdot C_2 \]

\[ + A_1 \cdot C_1 - A_1 \cdot A_3 \cdot C_1 - A_1 \cdot A_2 \cdot C_1 + A_1 \cdot A_2 \cdot A_3 \cdot C_1 \]

\[ + A_0 \cdot C_0 - A_0 \cdot A_3 \cdot C_0 - A_0 \cdot A_2 \cdot C_0 + A_0 \cdot A_2 \cdot A_3 \cdot C_0 \]

\[ - A_0 \cdot A_1 \cdot C_0 + A_0 \cdot A_1 \cdot A_3 \cdot C_0 + A_0 \cdot A_1 \cdot A_2 \cdot C_0 - A_0 \cdot A_1 \cdot A_2 \cdot A_3 \cdot C_0 \]

\[ + D_0 - A_3 \cdot D_0 - A_2 \cdot D_0 + A_2 \cdot A_3 \cdot D_0 - A_1 \cdot D_0 \]

\[ + A_1 \cdot A_3 \cdot D_0 + A_1 \cdot A_2 \cdot D_0 - A_1 \cdot A_2 \cdot A_3 \cdot D_0 - A_0 \cdot D_0 \]

\[ + A_0 \cdot A_3 \cdot D_0 + A_0 \cdot A_2 \cdot D_0 - A_0 \cdot A_2 \cdot A_3 \cdot D_0 + A_0 \cdot A_1 \cdot D_0 \]

\[ - A_0 \cdot A_1 \cdot A_3 \cdot D_0 - A_0 \cdot A_1 \cdot A_2 \cdot D_0 + A_0 \cdot A_1 \cdot A_2 \cdot A_3 \cdot D_0 \]
Rearranging…

\[ D_4 = D_0 \]
\[ + A_0 \cdot C_0 + A_1 \cdot C_1 + A_2 \cdot C_2 + A_3 \cdot C_3 \]
\[ - A_0 \cdot D_0 - A_1 \cdot D_0 - A_2 \cdot D_0 - A_3 \cdot D_0 \]
\[ + A_0 \cdot A_3 \cdot D_0 + A_0 \cdot A_2 \cdot D_0 + A_0 \cdot A_1 \cdot D_0 \]
\[ + A_1 \cdot A_3 \cdot D_0 + A_1 \cdot A_2 \cdot D_0 + A_2 \cdot A_3 \cdot D_0 \]
\[ - A_0 \cdot A_3 \cdot C_0 - A_0 \cdot A_2 \cdot C_0 - A_0 \cdot A_1 \cdot C_0 \]
\[ - A_1 \cdot A_3 \cdot C_1 - A_1 \cdot A_2 \cdot C_1 - A_2 \cdot A_3 \cdot C_2 \]
\[ + A_0 \cdot A_1 \cdot A_2 \cdot C_0 + A_0 \cdot A_1 \cdot A_3 \cdot C_0 + A_0 \cdot A_2 \cdot A_3 \cdot C_0 + A_1 \cdot A_2 \cdot A_3 \cdot C_1 \]
\[ - A_0 \cdot A_1 \cdot A_2 \cdot D_0 - A_0 \cdot A_1 \cdot A_3 \cdot D_0 - A_0 \cdot A_2 \cdot A_3 \cdot D_0 - A_1 \cdot A_2 \cdot A_3 \cdot D_0 \]
\[ + A_0 \cdot A_1 \cdot A_2 \cdot A_3 \cdot D_0 \]
\[ - A_0 \cdot A_1 \cdot A_2 \cdot A_3 \cdot C_0 \]
Sanity check

- Let’s make sure the expanded formula is still correct

- case where all alpha = 0
  \[ D_4 = D_0 \]
  - only background color shows (D0)

- case where all alpha = 1
  \[ D_4 = C_3 \]
  - last layer’s color shows (C3)
Pattern recognition

\[ D_4 = D_0 \]
\[ + A_0C_0 + A_1C_1 + A_2C_2 + A_3C_3 \]
\[ - A_0D_0 - A_1D_0 - A_2D_0 - A_3D_0 \]
\[ + A_0A_3D_0 + A_0A_2D_0 + A_0A_1D_0 \]
\[ + A_1A_3D_0 + A_1A_2D_0 + A_2A_3D_0 \]
\[ - A_0A_3C_0 - A_0A_2C_0 - A_0A_1C_0 \]
\[ - A_1A_3C_1 - A_1A_2C_1 - A_2A_3C_2 \]
\[ + A_0A_1A_2C_0 + A_0A_1A_3C_0 + A_0A_2A_3C_0 + A_1A_2A_3C_1 \]
\[ - A_0A_1A_2D_0 - A_0A_1A_3D_0 - A_0A_2A_3D_0 - A_1A_2A_3D_0 \]
\[ + A_0A_1A_2A_3D_0 \]
\[ - A_0A_1A_2A_3C_0 \]

There’s clearly a pattern here
we can easily extrapolate this for any number of layers

There is also a balance of additions and subtractions
with layer colors and background color
Order dependence

\[ D_4 = D_0 + A_0C_0 + A_1C_1 + A_2C_2 + A_3C_3 \]
\[ - A_0D_0 - A_1D_0 - A_2D_0 - A_3D_0 \]
\[ - A_0A_1A_2D_0 - A_0A_1A_3D_0 - A_0A_2A_3D_0 - A_1A_2A_3D_0 \]
\[ + A_0A_1A_2A_3D_0 \]
\[ - A_0A_3C_0 - A_0A_2C_0 - A_0A_1C_0 \]
\[ - A_1A_3C_1 - A_1A_2C_1 - A_2A_3C_2 \]
\[ + A_0A_3D_0 + A_0A_2D_0 + A_0A_1D_0 \]
\[ + A_1A_3D_0 + A_1A_2D_0 + A_2A_3D_0 \]
\[ + A_0A_1A_2C_0 + A_0A_1A_3C_0 + A_0A_2A_3C_0 + A_1A_2A_3C_1 \]
\[ - A_0A_1A_2A_3C_0 \]
Order independent Part

\[ D_4 = D_0 + A_0C_0 + A_1C_1 + A_2C_2 + A_3C_3 - A_0D_0 - A_1D_0 - A_2D_0 - A_3D_0 - A_0A_1A_2D_0 - A_0A_1A_3D_0 - A_0A_2A_3D_0 - A_1A_2A_3D_0 + A_0A_1A_2A_3D_0 \]

\[ \cdots \]

\[ \text{Summation and multiplication are both commutative operations} \]

\[ \text{i.e. order doesn't matter} \]

\[ A_0 + A_1 = A_1 + A_0 \]

\[ A_0 \times A_1 = A_1 \times A_0 \]

\[ A_0C_0 + A_1C_1 = A_1C_1 + A_0C_0 \]
Order independent Part

\[ D_4 = D_0 + A_0C_0 + A_1C_1 + A_2C_2 + A_3C_3 - A_0D_0 - A_1D_0 - A_2D_0 - A_3D_0 - A_0A_1A_2D_0 - A_0A_1A_3D_0 - A_0A_2A_3D_0 - A_1A_2A_3D_0 + A_0A_1A_2A_3D_0 \]

... 

Highlighted part may not be obvious, but here’s the simple proof:

\[ - A_0A_1A_2D_0 - A_0A_1A_3D_0 - A_0A_2A_3D_0 - A_1A_2A_3D_0 = - D_0A_0A_1A_2A_3 * (1/A_0 + 1/A_1 + 1/A_2 + 1/A_3) \]
Order dependent Part

\[ D4 = \ldots \]
\[ - A0*A3*C0 - A0*A2*C0 - A0*A1*C0 \]
\[ + A0*A3*D0 + A0*A2*D0 + A0*A1*D0 \]
\[ + A1*A3*D0 + A1*A2*D0 + A2*A3*D0 \]
\[ + A0*A1*A2*C0 + A0*A1*A3*C0 + A0*A2*A3*C0 + A1*A2*A3*C1 \]
\[ - A0*A1*A2*A3*C0 \]

These operations depend on order
results will vary if transparent layers are reordered
proof that proper alpha blending requires sorting
Can we ignore the order dependent part?

Do these contribute a lot to the final result of the formula?

- not if the alpha values are relatively low
- they’re all multiplying alpha values < 1 together
  - even with just 2 layers each with alpha = 0.25
    - $0.25 \times 0.25 = 0.0625$ which can be relatively insignificant
- more layers also makes them less important as do darker colors
Let’s analyze the ignored order dependent part (error) in some easy scenarios:

- All alphas = 0
  - Error = 0
- All alphas = 0.25
  - Error = 0.375*D0 - 0.14453125*C0 - 0.109375*C1 - 0.0625*C2
- All alphas = 0.5
  - Error = 1.5*D0 - 0.4375*C0 - 0.375*C1 - 0.25*C2
- All alphas = 0.75
  - Error = 3.375*D0 - 0.73828125*C0 - 0.703125*C1 - 0.5625*C2
- All alphas = 1
  - Error = 6*D0 - C0 - C1 - C2
Simpler is better

- A smaller part of the formula works much better in practice
  
  \[
  = D0 + A0*C0 + A1*C1 + A2*C2 + A3*C3 - A0*D0 - A1*D0 - A2*D0 - A3*D0 
  \]

- The balance in the formula is important, it maintains the weight of the formula

- This is much simpler and requires only 2 passes and a single render target
  - 1 less pass and 2 less render targets

- This formula is also exactly correct when blending a single translucent layer
Let’s analyze the simpler formula in some easy scenarios

all alphas = 0

error\text{\textsubscript{simple}} = 0
error\text{\textsubscript{prev}} = 0

all alphas = 0.25

error\text{\textsubscript{simple}} = 0.31640625*D0 - 0.14453125*C0 - 0.109375*C1 - 0.0625*C2
error\text{\textsubscript{prev}} = 0.375*D0 - 0.14453125*C0 - 0.109375*C1 - 0.0625*C2

all alphas = 0.5

error\text{\textsubscript{simple}} = 1.0625*D0 - 0.4375*C0 - 0.375*C1 - 0.25*C2
error\text{\textsubscript{prev}} = 1.5*D0 - 0.4375*C0 - 0.375*C1 - 0.25*C2

all alphas = 0.75

error\text{\textsubscript{simple}} = 2.00390625*D0 - 0.73828125*C0 - 0.703125*C1 - 0.5625*C2
error\text{\textsubscript{prev}} = 3.375*D0 - 0.73828125*C0 - 0.703125*C1 - 0.5625*C2

all alphas = 1

error\text{\textsubscript{simple}} = 3*D0 - C0 - C1 - C2
error\text{\textsubscript{prev}} = 6*D0 - C0 - C1 - C2
Error comparison

- Simpler formula actually has less error, which explains why it looks better.
- This is mainly because of the more balanced formula, where positives canceling out negatives sources colors cancel out background color.
Does it really work?

- Little error with relatively low alpha values
  - good approximation
- Completely inaccurate with higher alpha values
- Demo can show it much better than text
Sorted, alpha = 0.25
Approx, alpha = 0.25
Sorted, alpha = 0.5
Approx, alpha = 0.5
We want to implement the order independent part and just ignore the order dependent part

\[ D_4 = D_0 + A_0 C_0 + A_1 C_1 + A_2 C_2 + A_3 C_3 - A_0 D_0 - A_1 D_0 - A_2 D_0 - A_3 D_0 - A_0 A_1 A_2 D_0 - A_0 A_1 A_3 D_0 - A_0 A_2 A_3 D_0 - A_1 A_2 A_3 D_0 + A_0 A_1 A_2 A_3 D_0 \]

8 bits per component is not sufficient
not enough range or accuracy

Use 16 bits per component (64 bits per pixel for RGBA)
newer hardware support alpha blending with 64 bpp buffers

We can use multiple render targets to compute multiple parts of the equation simultaneously
1st pass

- Use additive blending
  \[ \text{SrcAlphaBlend} = 1 \]
  \[ \text{DstAlphaBlend} = 1 \]
  \[ \text{FinalRGBA} = \text{SrcRGBA} + \text{DstRGBA} \]

- render target #1, \( n^{th} \) layer
  \[ \text{RGB} = \text{An} \times \text{Cn} \]
  \[ \text{Alpha} = \text{An} \]

- render target #2, \( n^{th} \) layer
  \[ \text{RGB} = \frac{1}{\text{An}} \]
  \[ \text{Alpha} = \text{An} \]
1st pass results

After \( n \) translucent layers have been blended we get:

render target #1:

\[ \text{RGB1} = A_0 \times C_0 + A_1 \times C_1 + \ldots + A_n \times C_n \]
\[ \text{Alpha1} = A_0 + A_1 + \ldots + A_n \]

render target #2:

\[ \text{RGB2} = \frac{1}{A_0} + \frac{1}{A_1} + \ldots + \frac{1}{A_n} \]
\[ \text{Alpha2} = A_0 + A_1 + \ldots + A_n \]
2nd pass

- Use multiplicative blending
  \[
  \text{SrcAlphaBlend} = 0 \\
  \text{DstAlphaBlend} = \text{SrcRGBA} \\
  \text{FinalRGBA} = \text{SrcRGBA} \times \text{DstRGBA}
  \]

- render target #3, nth layer
  \[
  \text{RGB} = C_n \\
  \text{Alpha} = A_n
  \]
2nd pass results

- After \( n \) translucent layers have been blended we get:
  
  render target #3:
  
  - \( \text{RGB3} = C_0 \times C_1 \times \ldots \times C_n \)
  - \( \text{Alpha3} = A_0 \times A_1 \times \ldots \times A_n \)

- This pass isn’t really necessary for the better and simpler formula just for completeness
We now have the following background:

D0

render target #1:

- \( \text{RGB1} = A_0 \times C_0 + A_1 \times C_1 + \ldots + A_n \times C_n \)
- \( \text{Alpha1} = A_0 + A_1 + \ldots + A_n \)

render target #2:

- \( \text{RGB2} = \frac{1}{A_0} + \frac{1}{A_1} + \ldots + \frac{1}{A_n} \)
- \( \text{Alpha2} = A_0 + A_1 + \ldots + A_n \)

render target #3:

- \( \text{RGB3} = C_0 \times C_1 \times \ldots \times C_n \)
- \( \text{Alpha3} = A_0 \times A_1 \times \ldots \times A_n \)
Final pass

- Blend results in a pixel shader
- RGB1 - D0 * Alpha1
  \[= A0*C0 + A1*C1 + A2*C2 + A3*C3\]
  \[- D0 * (A0 + A1 + A2 + A3)\]
- D0 * Alpha3
  \[= D0 * (A0*A1*A2*A3)\]
- D0 * RGB2 * Alpha3
  \[= D0 * (1/A0 + 1/A1 + 1/A2 + 1/A3) * (A0*A1*A2*A3)\]
- Sum results with background color (D0) and we get:
  \[= D0\]
  \[+ A0*C0 + A1*C1 + A2*C2 + A3*C3\]
  \[- D0 * (A0 + A1 + A2 + A3)\]
  \[+ D0 * (A0*A1*A2*A3)\]
- That's the whole sort independent part of the blend formula
Application

- This technique is best suited for particles too many to sort. Slight inaccuracy in their color shouldn’t matter too much.
- Not so good for very general case, with all ranges of alpha values.
- For general case, works best with highly translucent objects i.e. low alpha values.
Can we do better?

I hope so…

Keep looking at the order dependent part of the formula to see if we can find more order independent parts out of it.

\[ D_4 = D_0 \]

\[ + A_0*C_0 + A_1*C_1 + A_2*C_2 + A_3*C_3 \]
\[- A_0*D_0 - A_1*D_0 - A_2*D_0 - A_3*D_0 \]
\[- A_0*A_1*A_2*D_0 - A_0*A_1*A_3*D_0 - A_0*A_2*A_3*D_0 - A_1*A_2*A_3*D_0 \]
\[ + A_0*A_1*A_2*A_3*D_0 \]
\[- A_0*A_3*C_0 - A_0*A_2*C_0 - A_0*A_1*C_0 \]
\[- A_1*A_3*C_1 - A_1*A_2*C_1 - A_2*A_3*C_2 \]
\[ + A_0*A_3*D_0 + A_0*A_2*D_0 + A_0*A_1*D_0 \]
\[ + A_1*A_3*D_0 + A_1*A_2*D_0 + A_2*A_3*D_0 \]
\[ + A_0*A_1*A_2*C_0 + A_0*A_1*A_3*C_0 + A_0*A_2*A_3*C_0 + A_1*A_2*A_3*C_1 \]
\[ - A_0*A_1*A_2*A_3*C_0 \]

Or use a completely different algorithm.
Q&A

If you have any other ideas or suggestions I’d love to hear them

email: hmeshkin@perpetual.com