Sort-Independent Alpha Blending

Houman Meshkin
Senior Graphics Engineer
Perpetual Entertainment
hmeshkin@perpetual.com

Alpha blending

- 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

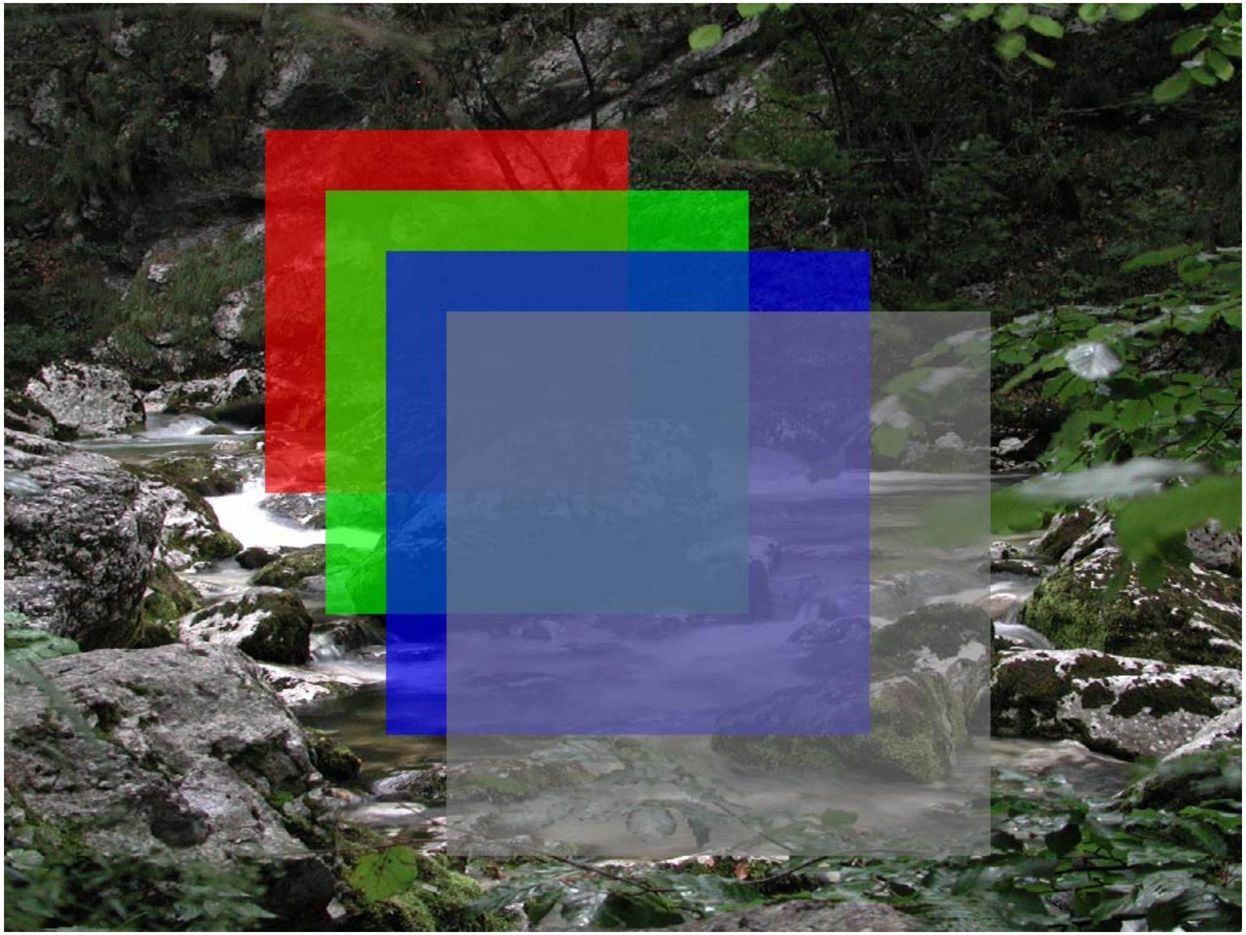
Challenge

- 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 n^{th} layer
- An: Alpha from n^{th} layer
- D0: background
- $D1 = A0*C0 + (1 - A0)*D0$
- $D2 = A1*C1 + (1 - A1)*D1$
- $D3 = A2*C2 + (1 - A2)*D2$
- $D4 = A3*C3 + (1 - A3)*D3$

Expanding the formula

```
D4 = A3*C3
   + A2*C2*(1 - A3)
   + A1*C1*(1 - A3)*(1 - A2)
   + A0*C0*(1 - A3)*(1 - A2)*(1 - A1)
   +    D0*(1 - A3)*(1 - A2)*(1 - A1)*(1 - A0)
```

Further expanding…

```
D4 = A3*C3
   + A2*C2 - A2*A3*C2
   + A1*C1 - A1*A3*C1 - A1*A2*C1 + A1*A2*A3*C1
   + A0*C0 - A0*A3*C0 - A0*A2*C0 + A0*A2*A3*C0
   - A0*A1*C0 + A0*A1*A3*C0 + A0*A1*A2*C0 - A0*A1*A2*A3*C0
   + D0 - A3*D0 - A2*D0 + A2*A3*D0 - A1*D0
   + A1*A3*D0 + A1*A2*D0 - A1*A2*A3*D0 - A0*D0
   + A0*A3*D0 + A0*A2*D0 - A0*A2*A3*D0 + A0*A1*D0
   - A0*A1*A3*D0 - A0*A1*A2*D0 + A0*A1*A2*A3*D0
```

Rearranging…

```
D4 = D0
   + A0*C0 + A1*C1 + A2*C2 + A3*C3
   - A0*D0 - A1*D0 - A2*D0 - A3*D0
   + A0*A3*D0 + A0*A2*D0 + A0*A1*D0
   + A1*A3*D0 + A1*A2*D0 + A2*A3*D0
   - A0*A3*C0 - A0*A2*C0 - A0*A1*C0
   - A1*A3*C1 - A1*A2*C1 - A2*A3*C2
   + A0*A1*A2*C0 + A0*A1*A3*C0 + A0*A2*A3*C0 + A1*A2*A3*C1
   - A0*A1*A2*D0 - A0*A1*A3*D0 - A0*A2*A3*D0 - A1*A2*A3*D0
   + A0*A1*A2*A3*D0
   - A0*A1*A2*A3*C0
```

Sanity check

- Let's make sure the expanded formula is still correct
- case where all alpha = 0

  D4 = D0

  - only background color shows (D0)
- case where all alpha = 1

  D4 = C3

  - last layer's color shows (C3)

Pattern recognition

```
D4 = D0
    + A0*C0 + A1*C1 + A2*C2 + A3*C3
    - A0*D0 - A1*D0 - A2*D0 - A3*D0
    + A0*A3*D0 + A0*A2*D0 + A0*A1*D0
    + A1*A3*D0 + A1*A2*D0 + A2*A3*D0
    - A0*A3*C0 - A0*A2*C0 - A0*A1*C0
    - A1*A3*C1 - A1*A2*C1 - A2*A3*C2
    + A0*A1*A2*C0 + A0*A1*A3*C0 + A0*A2*A3*C0 + A1*A2*A3*C1
    - A0*A1*A2*D0 - A0*A1*A3*D0 - A0*A2*A3*D0 - A1*A2*A3*D0
    + A0*A1*A2*A3*D0
    - A0*A1*A2*A3*C0
```

- 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

```
D4 = D0
     + A0*C0 + A1*C1 + A2*C2 + A3*C3
     - A0*D0 - A1*D0 - A2*D0 - A3*D0
     - A0*A1*A2*D0 - A0*A1*A3*D0 - A0*A2*A3*D0 - A1*A2*A3*D0
     + A0*A1*A2*A3*D0
     - A0*A3*C0 - A0*A2*C0 - A0*A1*C0
     - A1*A3*C1 - A1*A2*C1 - A2*A3*C2
     + 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
```

← *order independent part* (lines 2–5)

← *order dependent part* (lines 6–11)

Order independent Part

- D4 = D0
- + A0*C0 + A1*C1 + A2*C2 + A3*C3
- - A0*D0 - A1*D0 - A2*D0 - A3*D0
- - A0*A1*A2*D0 - A0*A1*A3*D0 - A0*A2*A3*D0 - A1*A2*A3*D0
- + A0*A1*A2*A3*D0
- …
- Summation and multiplication are both commutative operations

  i.e. order doesn't matter

  - A0 + A1 = A1 + A0
  - A0 * A1 = A1 * A0
  - A0*C0 + A1*C1 = A1*C1 + A0*C0

Order independent Part

- `D4 = D0`
- `+ A0*C0 + A1*C1 + A2*C2 + A3*C3`
- `- A0*D0 - A1*D0 - A2*D0 - A3*D0`
- **`- A0*A1*A2*D0 - A0*A1*A3*D0 - A0*A2*A3*D0 - A1*A2*A3*D0`**
- `+ A0*A1*A2*A3*D0`
- …
- Highlighted part may not be obvious, but here's the simple proof:
- `- A0*A1*A2*D0 - A0*A1*A3*D0 - A0*A2*A3*D0 - A1*A2*A3*D0`
- `=`
- `- D0*A0*A1*A2*A3 * (1/A0 + 1/A1 + 1/A2 + 1/A3)`

Order dependent Part

```
D4 = …
    - A0*A3*C0 - A0*A2*C0 - A0*A1*C0
    - A1*A3*C1 - A1*A2*C1 - A2*A3*C2
    + 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 * 0.25 = 0.0625 which can be relatively insignificant
  - more layers also makes them less important
  - as do darker colors

Error analysis

- 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

Error analysis

- Let's analyze the simpler formula in some easy scenarios
  - all alphas = 0
    - $error_{simple} = 0$
    - $error_{prev} = 0$
  - all alphas = 0.25
    - $error_{simple} = 0.31640625*D0 - 0.14453125*C0 - 0.109375*C1 - 0.0625*C2$
    - $error_{prev} = 0.375*D0 - 0.14453125*C0 - 0.109375*C1 - 0.0625*C2$
  - all alphas = 0.5
    - $error_{simple} = 1.0625*D0 - 0.4375*C0 - 0.375*C1 - 0.25*C2$
    - $error_{prev} = 1.5*D0 - 0.4375*C0 - 0.375*C1 - 0.25*C2$
  - all alphas = 0.75
    - $error_{simple} = 2.00390625*D0 - 0.73828125*C0 - 0.703125*C1 - 0.5625*C2$
    - $error_{prev} = 3.375*D0 - 0.73828125*C0 - 0.703125*C1 - 0.5625*C2$
  - all alphas = 1
    - $error_{simple} = 3*D0 - C0 - C1 - C2$
    - $error_{prev} = 6*D0 - C0 - C1 - C2$

Error comparison

- Simpler formula actually has less error
  - explains why it looks better
- This is mainly because of the more balanced formula
  - positives cancelling out negatives
  - source colors cancelling out background color

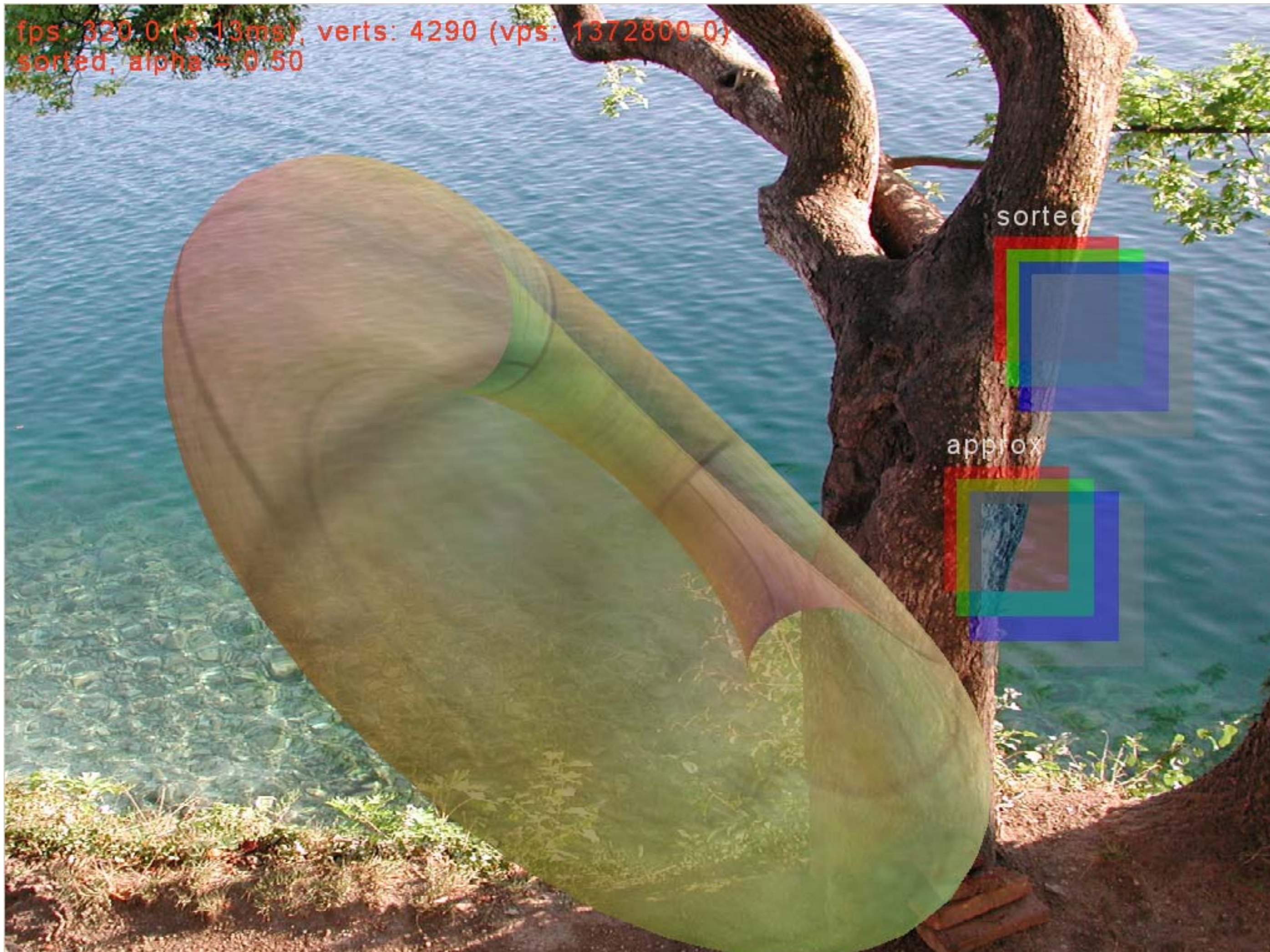
Does it really work?

- Little error with relatively low alpha values
  - good approximation
- Completely inaccurate with higher alpha values
- Demo can show it much better than text

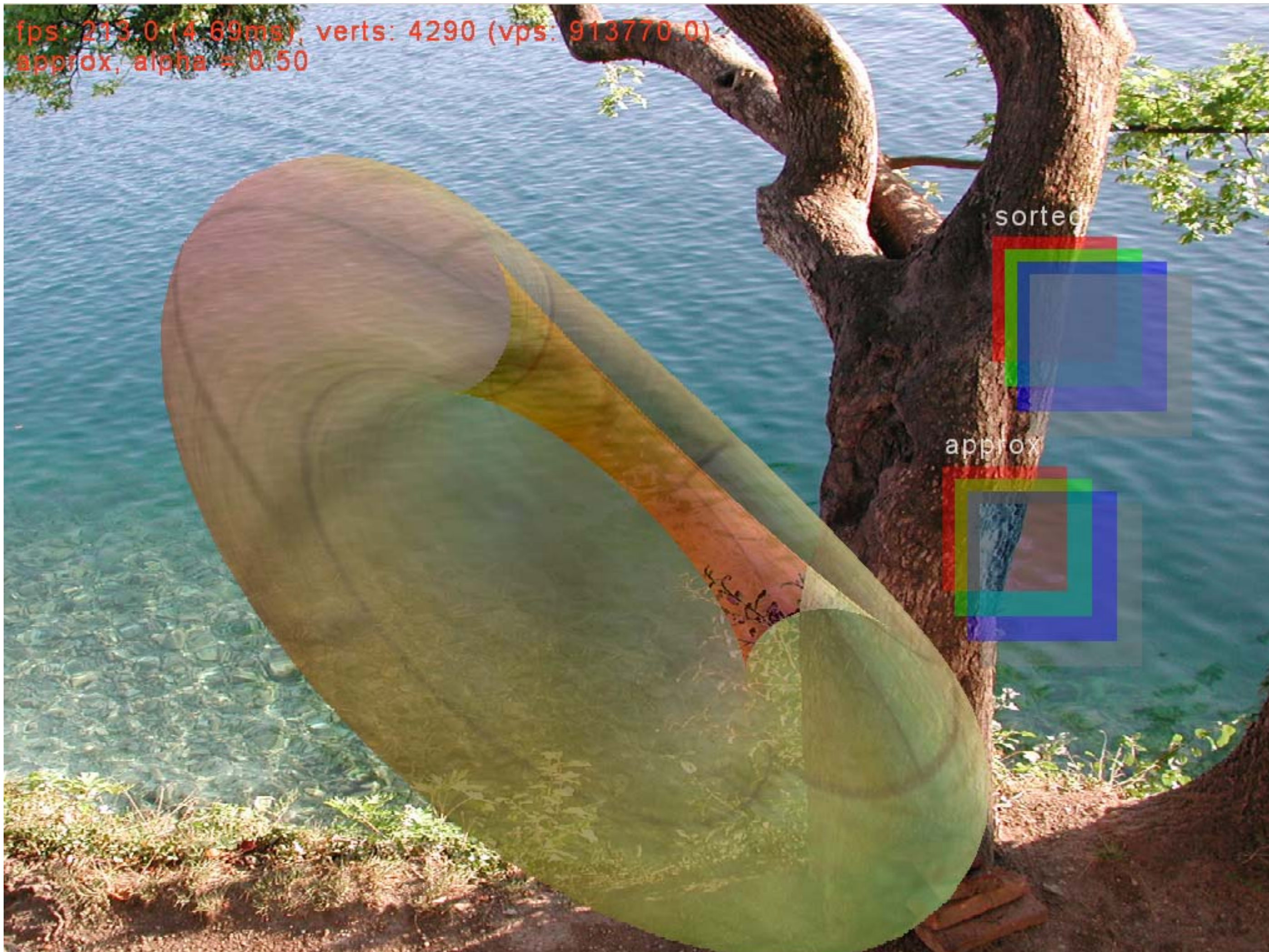
Sorted, alpha = 0.25

Approx, alpha = 0.25

Sorted, alpha = 0.5



Approx, alpha = 0.5



Implementation

- We want to implement the order independent part and just ignore the order dependent part
- `D4 = D0`
- `    + A0*C0 + A1*C1 + A2*C2 + A3*C3`
- `    - A0*D0 - A1*D0 - A2*D0 - A3*D0`
- `    - A0*A1*A2*D0 - A0*A1*A3*D0 - A0*A2*A3*D0 - A1*A2*A3*D0`
- `    + A0*A1*A2*A3*D0`
- 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

1^{st} pass

- Use additive blending
  - SrcAlphaBlend = 1
  - DstAlphaBlend = 1
  - FinalRGBA = SrcRGBA + DstRGBA
- render target #1, **n^{th}** layer
  - RGB = An * Cn
  - Alpha = An
- render target #2, **n^{th}** layer
  - RGB = 1 / An
  - Alpha = An

1^{st} pass results

- After **n** translucent layers have been blended we get:
  - render target #1:
    - $RGB1 = A0 * C0 + A1 * C1 + \ldots + An * Cn$
    - $Alpha1 = A0 + A1 + \ldots + An$
  - render target #2:
    - $RGB2 = 1 / A0 + 1 / A1 + \ldots + 1 / An$
    - $Alpha2 = A0 + A1 + \ldots + An$

2nd pass

- Use multiplicative blending
  - SrcAlphaBlend = 0
  - DstAlphaBlend = SrcRGBA
  - FinalRGBA = SrcRGBA * DstRGBA
- render target #3, **n**th layer
  - RGB = Cn
  - Alpha = An

2nd pass results

- After **n** translucent layers have been blended we get:
  - render target #3:
    - RGB3 = C0 * C1 * … * Cn
    - Alpha3 = A0 * A1 * … * An
- This pass isn't really necessary for the better and simpler formula
  - just for completeness

Results

- We now have the following
  - background
    - D0
  - render target #1:
    - RGB1 = A0 * C0 + A1 * C1 + … + An * Cn
    - Alpha1 = A0 + A1 + … + An
  - render target #2:
    - RGB2 = 1 / A0 + 1 / A1 + … + 1 / An
    - Alpha2 = A0 + A1 + … + An
  - render target #3:
    - RGB3 = C0 * C1 * … * Cn
    - Alpha3 = A0 * A1 * … * An

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

$$= D0 * (1/A0 + 1/A1 + 1/A2 + 1/A3) * (A0*A1*A2*A3)$$
$$= D0 * (A1*A2*A3 + A0*A2*A3 + A0*A1*A3 + A0*A1*A2)$$

- Sum results with background color (D0) and we get:

$$= D0$$
$$+ A0*C0 + A1*C1 + A2*C2 + A3*C3$$
$$- D0 * (A0 + A1 + A2 + A3)$$
$$+ D0 * (A0*A1*A2*A3)$$
$$- D0 * (A1*A2*A3 + A0*A2*A3 + A0*A1*A3 + A0*A1*A2)$$

- 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
- **D4 = D0**
- **+ A0*C0 + A1*C1 + A2*C2 + A3*C3**
- **- A0*D0 - A1*D0 - A2*D0 - A3*D0**
- **- A0*A1*A2*D0 - A0*A1*A3*D0 - A0*A2*A3*D0 - A1*A2*A3*D0**
- **+ A0*A1*A2*A3*D0**
- **- A0*A3*C0 - A0*A2*C0 - A0*A1*C0**
- **- A1*A3*C1 - A1*A2*C1 - A2*A3*C2**
- **+ 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**
- 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