Decomposing Images into Layers with Advanced Color Blending

Yuki Koyama & Masataka Goto
Quick Summary
Our method decomposes a target image into semi-transparent layers ...
... that are associated with advanced color-blend modes such as hard-light and multiply.
Usage Scenario

Import the resulting layers to Photoshop etc.

Perform non-trivial edits (e.g., lighting-aware hue change)

Input images except for the rightmost one courtesy of David Revoy
Background

What is Advanced Color Blending?
A **color-blend mode** defines the mapping from the colors of the layers to the color that will be rendered.

Examples: *multiply, overlay, color-dodge, normal*, etc.
Digital Drawing & Effects with Color Blending

Advanced (non-linear) color blending is used for creating interesting color effects

Example:
digital drawing using the color-dodge mode

David Revoy:
“Painting with Blending-modes”
https://youtu.be/AybFWViT-3Q
It is important to combine various blend modes
It is important to combine various blend modes

- normal
- multiply
- hard-light
- screen
It is important to combine various blend modes

normal  multiply  hard-light  screen

Note: every digital artist has his/her own way to combine blend modes
Our Motivation
Motivation: Color Unblending

Arbitrary combination of blend modes

It should be useful if an existing image can be decomposed into layers with an arbitrary combination of advanced color-blend modes.
Previous Work
Layer Decomposition by Color Unblending
RGB-Space Geometry [Tan+, TOG 16]

- Support **only a specific linear blend mode** (i.e., “normal”)
- Heavily **rely on the linearity** of the blend mode
- Cannot be (easily) extended for non-linear color-blend modes
Per-Pixel Optimization [Aksoy+, TOG 16; TOG 17]

- Support only a specific linear blend mode (and a linear alpha addition)
Per-Pixel Optimization [Aksoy+, TOG 16; TOG 17]

- Support only a specific linear blend mode (and a linear alpha addition)

We generalize this method to support advanced (non-linear) color-blend modes.
Details of Our Method [1/2]
Per-Pixel Unblending Optimization (Concept-Level Explanation)
Problem Setting: Per-Pixel Unblending

Target image
Problem Setting: Per-Pixel Unblending

Target image

Target pixel color

The pixel that we are going to handle now

$[r, g, b, \alpha]$
Problem Setting: Per-Pixel Unblending

Target pixel color

\([r, g, b, \alpha]\)
Problem Setting: Per-Pixel Unblending

**Target pixel color**

\[
[r, g, b, \alpha]
\]

Unblending

- multiply
- screen
- normal
Problem Setting: Per-Pixel Unblending

Target pixel color

\[ [r, g, b, \alpha] \]

Unblending

Blend mode

Desirable color distribution: the distribution of RGB values that the decomposed layer contains

multiply

dot

screen

normal
Problem Setting: Per-Pixel Unblending

Target pixel color

\[ [r, g, b, \alpha] \]
Problem Setting: Per-Pixel Unblending

Target pixel color

\[ [r, g, b, \alpha] \]

Input

Output

\[ [r_n, g_n, b_n, \alpha_n] \]
\[ [r_2, g_2, b_2, \alpha_2] \]
\[ [r_1, g_1, b_1, \alpha_1] \]
Problem Setting: Per-Pixel Unblending

Target
pixel color

Input

Output

Un-blending

multiply

screen

normal

Target
pixel color

Composited
pixel color

Un-blending

multiply

screen

normal

Input

Output

$r, g, b, \alpha$

$r_n, g_n, b_n, \alpha_n$

$r_1, g_1, b_1, \alpha_1$

$r_2, g_2, b_2, \alpha_2$

$\hat{r}, \hat{g}, \hat{b}, \hat{\alpha}$
Problem Setting: Per-Pixel Unblending

**Input**

- **Target pixel color**
  - \([r, g, b, \alpha]\)

**Output**

- **multiply**
  - \([r_n, g_n, b_n, \alpha_n]\)
- **screen**
  - \([r_2, g_2, b_2, \alpha_2]\)
- **normal**
  - \([r_1, g_1, b_1, \alpha_1]\)

**Composited pixel color**

- \([\hat{r}, \hat{g}, \hat{b}, \hat{\alpha}]\)

**Unblending**

- Should equal
Problem Setting: Per-Pixel Unblending

Target pixel color

Input

Multiply

Composited pixel color

Output

Unblending

Blending

Should equal

Should be as consistent as possible

Unblending

Blending

Target pixel color

Composite pixel color

Multiply

Screen

Normal

Should equal

Should be as consistent as possible

Input

Output

\[
\begin{bmatrix}
  r, & g, & b, & \alpha
\end{bmatrix}
\]

\[
\begin{bmatrix}
  r_1, & g_1, & b_1, & \alpha_1
\end{bmatrix}
\]

\[
\begin{bmatrix}
  r_n, & g_n, & b_n, & \alpha_n
\end{bmatrix}
\]

\[
\begin{bmatrix}
  \hat{r}, & \hat{g}, \hat{b}, & \hat{\alpha}
\end{bmatrix}
\]
Example User Interaction
Basics (Not Our Method)

Math of Advanced Color Blending
Descriptor of the $i$-th layer’s pixel

$$\mathbf{x}_i = \begin{bmatrix} r_i & g_i & b_i & \alpha_i \end{bmatrix}^T$$

$$= \begin{bmatrix} c_i^T & \alpha_i \end{bmatrix}^T \in \mathbb{R}^4$$
Descriptor of a Pixel Color

Descriptor of the $i$-th layer’s pixel

$$x_i = \begin{bmatrix} r_i & g_i & b_i & \alpha_i \end{bmatrix}^T$$

$$= \begin{bmatrix} c_i^T & \alpha_i \end{bmatrix}^T \in \mathbb{R}^4$$

Descriptor of all the layers’ pixels

$$x = \begin{bmatrix} x_1^T & \cdots & x_n^T \end{bmatrix}^T \in \mathbb{R}^{4n}$$
General Composition Operator (⊕) for Compositing Two Layers

General composition operator

\[ X_s \oplus X_d \Rightarrow ? \]
General Composition Operator (⊕) for Compositing Two Layers

\[ \mathbf{x}_s \mathbin{\circledast}_\text{rgb} \mathbf{x}_d = \frac{f(\mathbf{c}_s, \mathbf{c}_d) \alpha_s \alpha_d + Y \alpha_s (1 - \alpha_d) \mathbf{c}_s + Z \alpha_d (1 - \alpha_s) \mathbf{c}_d}{\mathbf{x}_s \mathbin{\circledast}^\alpha \mathbf{x}_d} \]

\[ \mathbf{x}_s \mathbin{\circledast}^\alpha \mathbf{x}_d = X \alpha_s \alpha_d + Y \alpha_s (1 - \alpha_d) + Z \alpha_d (1 - \alpha_s) \]

“SVG Compositing Specification” by W3C: https://www.w3.org/TR/2011/WD-SVGCompositing-20110315/
General Composition Operator ($\oplus$) for Compositing Two Layers

\[
x_s \oplus_{\text{rgb}} x_d = \frac{f(c_s, c_d)\alpha_s\alpha_d + Y\alpha_s(1 - \alpha_d)c_s + Z\alpha_d(1 - \alpha_s)c_d}{x_s \oplus^\alpha x_d}
\]

\[
x_s \oplus^\alpha x_d = X\alpha_s\alpha_d + Y\alpha_s(1 - \alpha_d) + Z\alpha_d(1 - \alpha_s)
\]

**Blend function:** Mode-specific function (can be non-linear)

\[
f : [0, 1]^3 \times [0, 1]^3 \rightarrow [0, 1]^3
\]

For example: $f_{\text{normal}}, f_{\text{multiply}}, f_{\text{color-dodge}}, \ldots$

“SVG Compositing Specification” by W3C: https://www.w3.org/TR/2011/WD-SVGCompositing-20110315/
More-Than-Two-Layers Case: **Recursive Layering**

**Recursive rule:**

\[ \hat{x}_k = \begin{cases} 
  x_1 & (k = 1) \\
  x_k \oplus_k \hat{x}_{k-1} & \text{(otherwise)} 
\end{cases} \]

![Recursive layers diagram](image)
Details of Our Method [2/2]

Per-Pixel Unblending Optimization
(Equation-Level Explanation)
Unblending Optimization

**Color consistency:**
Each pixel color should be as consistent to the specified color distribution as possible

**Color reproducibility:**
The target pixel color should be reproduced by compositing the resulting layer colors
Unblending Optimization

Equality-constrained optimization problem

\[
\min_{x \in \mathbb{R}^{4n}} \ E(x) \quad \text{s.t.} \quad C(x) = 0
\]

Objective: Each pixel color should be as consistent to the specified color distribution as possible

Equality constraint: The target pixel color should be reproduced by compositing the resulting layer colors
Equality-constrained optimization problem

\[
\min_{x \in \mathbb{R}^{4n}} E(x) \quad \text{s.t.} \quad C(x) = 0
\]

**Objective:**
Each pixel color should be as consistent to the specified color distribution as possible.

**Equality constraint:**
The target pixel color should be reproduced by compositing the resulting layer colors.
Constraint Function Definition

\[ C(x) = \left( x_n \oplus_n (x_{n-1} \oplus_{n-1} (x_{n-2} \oplus_{n-2} \ldots)) \right) - c_{\text{target}} = 0 \]

E.g., multiply, color-dodge, overlay ...
Constraint Function Definition

C.f., [Aksoy+16; 17]: Assumed a special case of a linear model

\[
C(x) = \sum_{i=1}^{n} \alpha_i c_i - c^{\text{target}} = 0
\]
Constraint Function Definition

C.f., [Aksoy+16; 17]: Assumed a special case of a linear model

\[ C(x) = (x_n \oplus_n (x_{n-1} \oplus_{n-1} (x_{n-2} \oplus_{n-2} (\ldots)))) - c^{\text{target}} = 0 \]

E.g., multiply, color-dodge, overlay ...

Our method is a generalization of [Aksoy+16; 17]
Solving Equality-Constrained Optimization [1/2]

Algorithm: The augmented Lagrangian method

- Iteratively solve (unconstrained) optimizations

while (not converged) {
    solve \[ \min_x \left\{ E(x) - \lambda^T C(x) + \frac{\rho}{2} \|C(x)\|^2 \right\} \]
    update \( \lambda \) and \( \rho \)
}

Solving Equality-Constrained Optimization [1/2]

Algorithm: The augmented Lagrangian method

- Iteratively solve (unconstrained) optimizations

```
while (not converged) {
    solve \( \min_x \left\{ E(x) - \lambda^T C(x) + \frac{\rho}{2} \| C(x) \|^2 \right\} \)
    update \( \lambda \) and \( \rho \)
}
```

Need the derivative of \( C(x) \) to efficiently solve this optimization
It is possible to algorithmically calculate the derivative of $C(x)$ by recursively applying the chain rule

\[
\frac{\partial C(x)}{\partial x_i} = \begin{cases} 
I & (i = k = 1) \\
\frac{\partial}{\partial x_k} \left( x_k \oplus_k \hat{x}_{k-1} \right) & (i = k \neq 1) \\
\frac{\partial}{\partial \hat{x}_{k-1}} \left( x_k \oplus_k \hat{x}_{k-1} \right), & \text{(otherwise)}
\end{cases}
\]
It is possible to algorithmically calculate the derivative of $C(x)$ by recursively applying the chain rule.

\[
\frac{\partial C(x)}{\partial x_i} = \begin{cases} 
1 & \quad (i = k = 1) \\
\frac{\partial}{\partial x_k} \left( x_k \oplus_k \hat{x}_{k-1} \right) \cdot \frac{\partial}{\partial \hat{x}_{k-1}} \left( x_k \oplus_k \hat{x}_{k-1} \right), & \quad (i = k \neq 1) \\
\frac{\partial}{\partial \hat{x}_{k-1}} \left( x_k \oplus_k \hat{x}_{k-1} \right), & \quad \text{(otherwise)}
\end{cases}
\]

Our unblending optimization problem can be efficiently solved by gradient-based algorithms (e.g., L-BFGS).
Unblending using linear blend mode

Identical to the results by [Aksoy+17]

Unblending using various blend modes

Input image courtesy of David Revoy
Post-Processing
Refinement for Smoothness
Problem:

Resulting layers may not be smooth because the unblending optimization is “per-pixel” (no inter-pixel consideration)

Input image courtesy of David Revoy
Refinement Step for Smoothness

■ Problem:

Resulting layers may not be smooth because the unblending optimization is “per-pixel” (no inter-pixel consideration)

■ Solution:

Performing a procedure called “matte refinement” [Aksoy+17] as a post processing to enforce smoothness
Usage Scenario
Example: Photoshop
Applications #1
Lighting-Aware Editing of an Existing Illustration
Applications #2

Bringing a Physical Painting into Digital Workflow
Other Applications
Intrinsic image decomposition: decomposing an image into reflectance and shading layers

Adding gradation to an existing conference logo: extracting a “fill” layer from the logo and re-fill with gradation

Input images are provided by [Innamorati+17]
Remixing existing illustrations:
Harmonize colors of images from different scenes to put them together in a derivative work

Input image courtesy of David Revoy
Summary
Summary

- Solution for a new problem
  - Decomposing an image into layers with advanced (non-linear) color blending

- Techniques
  - Generalizing Aksoy+’s method [16; 17] for handling advanced (non-linear) color blend modes

- Future work
  - Automatic determination of optimal color distributions etc.
We are preparing for releasing our source codes at GitHub so please check our project page later.
Decomposing Images into Layers with Advanced Color Blending

Yuki Koyama & Masataka Goto

Two Main Usages of Layers

- For segmenting regions (not our focus)
- For blending colors (our focus)
Specialization of the Composite Operator

Alpha blending model
C.f., [Tan+16]

\[ f(c_s, c_d) = c_s, \]
\[ X = Y = Z = 1 \]

\[ x_s \oplus_{\text{rgb}} x_d = \frac{\alpha_s c_s + (1 - \alpha_s) \alpha_d c_d}{\alpha_s \oplus^\alpha \alpha_d}, \]
\[ \alpha_s \oplus^\alpha \alpha_d = \alpha_s + (1 - \alpha_s) \alpha_d \]

Linear additive model
C.f., [Aksoy+16; 18]

\[ f(c_s, c_d) = c_s + c_d, \]
\[ X = 2, Y = Z = 1 \]

\[ x_s \oplus_{\text{rgb}} x_d = \frac{\alpha_s c_s + \alpha_d c_d}{\alpha_s \oplus^\alpha \alpha_d}, \]
\[ \alpha_s \oplus^\alpha \alpha_d = \alpha_s + \alpha_d \]
Computational Cost

- Parallelization is effective
- Increase cost along with the number of pixels
- Increase cost along with the number of layers (due to the recursive calculation)
Our method enables a new digital painting workflow by effectively using such a deep-colorized image as a good starting point. Instead of directly editing the generated single-layered image, the user can decompose it into layers with familiar advanced color blending then individually edit each unsatisfactory layer by adding strokes, changing hue, etc.
Example use of our method, in which the blue sky background is replaced with a sunset one. With our method, the character region is decomposed into three layers: the bottom two layers roughly correspond to albedo, and the top layer roughly corresponds to lighting by the blue sky represented as an overlay layer. To match the lighting in sunset, the top layer’s hue is modified. Finally, the layers are re-composited with the sunset background. Note that naively compositing the character region with the sunset background (the leftmost image) does not look good because lighting is mismatched. Input image courtesy of David Revoy.