Image Processing, Warping, and Compositing

Michael Kazhdan

(601.457/657)

HB Ch. 4.8
FvDFH Ch. 14.10
Outline

• Image Processing
  • Image Warping
  • Image Compositing
Image Processing

• What about the case when the modification that we would like to make to a pixel depends on the pixels around it?
  ◦ Blurring
  ◦ Edge Detection
  ◦ Etc.
Multi-Pixel Operations

Stationary/Local Filtering

• In the simplest case, we define a mask of weights telling us how values at adjacent pixels should be combined to generate the new value.
Blurring

• To blur across pixels, define a mask:
  ○ Whose values are non-negative
  ○ Whose value is larger near the center of the mask
  ○ Whose entries sum to one.

Original Blur

\[
\begin{bmatrix}
\frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\
\frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\
\frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\
\end{bmatrix}
\]

Original \quad \rightarrow \quad Blur
Blurring

Pixel(x,y): red = 36
green = 36
blue = 0

Filter = \[
\begin{bmatrix}
\frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\
\frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\
\frac{1}{16} & \frac{2}{16} & \frac{1}{16}
\end{bmatrix}
\]
Blurring

**Pixel(x,y):**
- red = 36
- green = 36
- blue = 0

**Filter:**
\[
\begin{bmatrix}
\frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\
\frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\
\frac{1}{16} & \frac{2}{16} & \frac{1}{16}
\end{bmatrix}
\]

**Pixel(x,y).red and its red neighbors:**

<table>
<thead>
<tr>
<th></th>
<th>X - 1</th>
<th>X</th>
<th>X + 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y - 1</td>
<td>36</td>
<td>109</td>
<td>146</td>
</tr>
<tr>
<td>Y</td>
<td>32</td>
<td>36</td>
<td>109</td>
</tr>
<tr>
<td>Y + 1</td>
<td>32</td>
<td>36</td>
<td>73</td>
</tr>
</tbody>
</table>
Blurring

Original

<table>
<thead>
<tr>
<th>Y-1</th>
<th>X-1</th>
<th>X</th>
<th>X+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>109</td>
<td>146</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>36</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>36</td>
<td>73</td>
<td></td>
</tr>
</tbody>
</table>

Pixel(x,y).red and its red neighbors

New value for Pixel(x,y).red =
\[
\begin{align*}
(36 \times \frac{1}{16}) + (109 \times \frac{2}{16}) + (146 \times \frac{1}{16}) \\
(32 \times \frac{2}{16}) + (36 \times \frac{4}{16}) + (109 \times \frac{2}{16}) \\
(32 \times \frac{1}{16}) + (36 \times \frac{2}{16}) + (73 \times \frac{1}{16})
\end{align*}
\]

Filter =
\[
\begin{bmatrix}
\frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\
\frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\
\frac{1}{16} & \frac{2}{16} & \frac{1}{16}
\end{bmatrix}
\]
Blurring

Original

Pixel(x,y).red and its red neighbors

New value for Pixel(x,y).red = 62.69

Filter =

\[
\begin{bmatrix}
\frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\
\frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\
\frac{1}{16} & \frac{2}{16} & \frac{1}{16}
\end{bmatrix}
\]
Blurring

New value for Pixel(x,y).red = 63

Original

Blur

\[
\begin{bmatrix}
\frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\
\frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\
\frac{1}{16} & \frac{2}{16} & \frac{1}{16}
\end{bmatrix}
\]
Blurring

- Repeat for each pixel and each color channel.
- Keep source and destination separate to avoid “drift”.
- For boundary pixels, not all neighbors are used.
  - Normalize the mask so the values sum to one, or
  - Assume that the exterior values are black, or
  - Assume the exterior values can be obtained by reflecting the image across the boundary, or
  - Assume…
Blurring

- In general, the mask can have arbitrary size:
  - We can express a smaller mask as a bigger one by padding with zeros.

\[
\begin{bmatrix}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1 \\
\end{bmatrix}
/16
\]

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 1 & 0 \\
0 & 2 & 4 & 2 & 0 \\
0 & 1 & 2 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
/16
\]
Blurring

- In general, the mask can have arbitrary size:
  - We can have more non-zero entries to give rise to a wider blur.

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 1 & 0 \\
0 & 2 & 4 & 2 & 0 \\
0 & 1 & 2 & 1 & 0 \\
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 1 & 2 & 1 & 0 \\
1 & 2 & 4 & 2 & 1 \\
2 & 4 & 8 & 4 & 2 \\
1 & 2 & 4 & 2 & 1 \\
0 & 1 & 2 & 1 & 0
\end{bmatrix}
\] /48

Original  Narrow Blur  Wide Blur
Blurring

- A general way for defining the entries of an $n \times n$ mask is to use the values of a Gaussian:

$$\text{GaussianMask}[i][j] = e^{-\frac{(i-\sigma)^2+(j-\sigma)^2}{4\sigma^2}}$$

$i, j \in [0, 2\sigma + 1]$

- $\sigma$ is the mask radius (for $n = 2\sigma + 1$)
- $i$ is the horizontal position in the mask
- $j$ is the vertical position in the mask
- Don’t forget to normalize!
Edge Detection

• An edge is a point in the image where the image is “far” from constant:
  ◦ The difference between the value at the pixel and the value of neighboring pixels is large (in absolute value)
Edge Detection

• To find the edges, define a mask:
  ◦ whose value is one at the center pixel,
  ◦ Negative at neighboring pixels, and
  ◦ Whose entries sum to zero.

• (Upper) edge pixels are those whose value is larger than the average of its neighbors.

Original
Detected Edges

Filter = \[
\frac{1}{8} \begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix}
\]
Edge Detection

Pixel(x,y): red = 36
green = 36
blue = 0

Filter = \[
\frac{1}{8} \begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1
\end{bmatrix}
\]
Edge Detection

Pixel\((x, y)\): red = 36  
green = 36  
blue = 0

Pixel\((x, y)\).red and its red neighbors

\[
\begin{array}{ccc}
  X - 1 & X & X + 1 \\
Y - 1 & 36 & 109 & 146 \\
Y     & 32 & 36 & 109 \\
Y + 1 & 32 & 36 & 73 \\
\end{array}
\]

Filter = \[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix}
\]

Original
Edge Detection

Original

<table>
<thead>
<tr>
<th></th>
<th>X - 1</th>
<th>X</th>
<th>X + 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y - 1</td>
<td>36</td>
<td>109</td>
<td>146</td>
</tr>
<tr>
<td>Y</td>
<td>32</td>
<td>36</td>
<td>109</td>
</tr>
<tr>
<td>Y + 1</td>
<td>32</td>
<td>36</td>
<td>73</td>
</tr>
</tbody>
</table>

New value for Pixel(x,y).red =

\[
(36 \times -1/8) + (109 \times -1/8) + (146 \times -1/8) \\
(32 \times -1/8) + (36 \times 1) + (109 \times -1/8) \\
(32 \times -1/8) + (36 \times -1/8) + (73 \times -1/8)
\]

Filter = \[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix}
\]
Edge Detection

New value for Pixel(x,y).red = -285/8

Filter = \[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1
\end{bmatrix}
\]
Edge Detection

New value for Pixel(x,y).red = -35.625

Filter = \[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1
\end{bmatrix}
\]

Note: Edge values are not colors, so we have to rescale/remap for visualization.
Outline

• Image Processing
• Image Warping
• Image Compositing
Image Warping

• Move pixels of image
  ◦ Mapping
  ◦ Resampling

Source image  Warp  Destination image
Overview

• Mapping
  ◦ Forward
  ◦ Inverse

• Resampling
  ◦ Point sampling
  ◦ Triangle filter
  ◦ Gaussian filter
Mapping

- Define transformation
  - Describe the destination \((x, y) = \Phi(u, v)\) for every location \((u, v)\) in the source
Example Mappings

• Scale by $\sigma$:
  
  $\Phi(u, v) = (\sigma u, \sigma v)$

Scale $\sigma = 0.8$
Example Mappings

- Rotate by $\theta$ degrees:
  - $\Phi(u, v) = (u \cos \theta - v \sin \theta, u \sin \theta + v \cos \theta)$
Example Mappings

- Shear in $x$ by $\sigma_x$:
  - $\Phi(u, v) = (u + \sigma_x \cdot v, v)$

- Shear in $y$ by $\sigma_y$:
  - $\Phi(u, v) = (u, v + \sigma_y \cdot u)$
Other Mappings

• Any function of $u$ and $v$:
  ○ $\Phi(u, v) = \cdots$

Fish-eye

“Swirl”

“Rain”
Image Warping Implementation I

• Forward mapping:

```c
for( v=0 ; v<vmax ; v++ )
    for( u=0 ; u<umax ; u++ )
        (x,y) = \Phi(u,v);
        dst(x,y) = src(u,v);
```

Source image \( (u,v) \)    Destination image \( (x,y) \)
Forward Mapping

- Iterate over source image
Forward Mapping – BAD!

- Iterate over source image

Multiple source pixels can map to same destination pixel

Rotate + Translate
Forward Mapping – BAD!

- Iterate over source image

Multiple source pixels can map to same destination pixel

Some destination pixels may not be covered

[Diagram showing source and destination pixels, with arrows indicating rotation and translation.]
Image Warping Implementation II

• Inverse mapping:

\[
\text{for( } \ y=0 \ ; \ y<\text{ymax} \ ; \ y++ \ \text{)} \\
\text{for( } \ x=0 \ ; \ x<\text{xmax} \ ; \ x++ \ \text{)} \\
(u,v) = \Phi^{-1}(x,y); \\
dst(x,y) = src(u,v); \\
\]

Source image \hspace{2cm} \Phi \hspace{2cm} \text{Destination image}

\((u,v)\) \hspace{2cm} \((x,y)\)
Reverse Mapping – GOOD!

- Iterate over destination image
  - Must resample source

Rotate -30
+ Translate
Resampling

- Evaluate source image at \((u, v) = \Phi^{-1}(x, y)\)
Overview

• Mapping
  ○ Forward
  ○ Inverse

• Resampling
  ○ Nearest Point Sampling
  ○ Bilinear Sampling
  ○ Gaussian Sampling
Nearest Point Sampling

- Take value at closest pixel:
  
  \[
  \text{int } iu = \text{floor}(u+0.5); \\
  \text{int } iv = \text{floor}(v+0.5); \\
  \text{dst}(x,y) = \text{src}(iu,iv); \\
  \]
Bilinear Sampling

- Bilinearly interpolate four closest source pixels

\[ \text{dst}(x, y) = \text{Weighted average of source at } (u_1, v_1), (u_2, v_1), (u_1, v_2), \text{ and } (u_2, v_2) \]
Linear Sampling

- Linearly interpolate two closest source pixels

$$\text{dst}(x) = \text{linear interpolation of } u_1 \text{ and } u_2$$

$$u_1 = \text{floor}(u);$$
$$u_2 = u_1 + 1;$$
$$du = u - u_1;$$
$$\text{dst}(u) = \text{src}(u_1)*(1-du) + \text{src}(u_2)*du;$$

$$0 \leq du \leq 1$$
Bilinear Sampling

- Bilinearly interpolate four closest source pixels

  \(a = \text{linear interpolation of } src(u_1, v_1) \text{ and } src(u_2, v_1)\)

  \(b = \text{linear interpolation of } src(u_1, v_2) \text{ and } src(u_2, v_2)\)

  \(\text{dst}(x, y) = \text{linear interpolation of } a \text{ and } b\)

\[\begin{align*}
  u_1 &= \text{floor}(u), \quad u_2 = u_1 + 1; \\
  v_1 &= \text{floor}(v), \quad v_2 = v_1 + 1; \\
  d_u &= u - u_1; \\
  a &= \text{src}(u_1,v_1)*(1-d_u) \\
  &\quad + \text{src}(u_2,v_1)*(d_u); \\
  b &= \text{src}(u_1,v_2)*(1-d_u) \\
  &\quad + \text{src}(u_2,v_2)*d_u; \\
  d_v &= v - v_1; \\
  \text{dst}(x,y) &= a*(1-d_v) + b*d_v;
\end{align*}\]
Bilinear Sampling

- Bilinearly interpolate four closest source pixels
  
  \[ a = \text{linear interpolation of } \text{src}(u_1, v_1) \text{ and src}(u_2, v_1) \]
  
  \[ b = \text{linear interpolation of } \text{src}(u_1, v_2) \text{ and src}(u_2, v_2) \]
  
  \[ \text{dst}(x, y) = \text{linear interpolation of } a \text{ and } b \]

\[
\begin{align*}
u_1 &= \text{floor}(u), \quad &u_2 &= u_1 + 1; \\
v_1 &= \text{floor}(v), \quad &v_2 &= v_1 + 1; \\
du &= u - u_1; \\
&= (1 - du) \text{src}(u_1, v_1) + du \text{src}(u_2, v_1); \\
a &= \text{src}(u_1, v_1) (1 - du) + \text{src}(u_2, v_1) du; \\
b &= \text{src}(u_1, v_2) (1 - du) + \text{src}(u_2, v_2) du; \\
dv &= v - v_1; \\
\text{dst}(x, y) &= a(1 - dv) + b dv;
\end{align*}
\]

Make sure to test that the pixels \((u_1, v_1), (u_2, v_2), (u_1, v_2), \) and \((u_2, v_1)\) are within the image.
Gaussian Sampling

• Compute weighted sum of pixel neighborhood:
  ○ The blending weights are the normalized values of a Gaussian function.
Filtering Methods Comparison

- Trade-offs
  - Jagged edges versus blurring
  - Computation speed
Image Warping Implementation

- Inverse mapping:

```plaintext
for( y=0 ; y<ymax ; y++ )
    for( x=0 ; x<xmax ; x++ )
        (u,v) = Φ⁻¹(x,y);
        dst(x,y) = resample_src(u,v,w);
```

Source image  Destination image

\[(u,v)\]  \[\Phi\]  \[(x,y)\]
Image Warping Implementation

- Inverse mapping:

\[
\begin{align*}
&\text{for}( \ y = 0 \ ; \ y < y_{\text{max}} \ ; \ y++ ) \\
&\quad \text{for}( \ x = 0 \ ; \ x < x_{\text{max}} \ ; \ x++ ) \\
&\quad (u,v) = \Phi^{-1}(x,y); \\
&\quad \text{dst}(x,y) = \text{resample}_\text{src}(u,v,w);
\end{align*}
\]
Example: Scale

\[ \text{Scale}( \text{src}, \text{dst}, \sigma ) : \]
\[
\begin{align*}
w & \equiv \frac{1}{\sigma} ; \\
& \quad \text{for}( y=0 ; y<y_{\text{max}} ; y++ ) \\
& \quad \quad \text{for}( x=0 ; x<x_{\text{max}} ; x++ ) \\
& \quad \quad \quad (u,v) = (x,y) / \sigma ; \\
& \quad \quad \quad \text{dst}(x,y) = \text{resample}_\text{src}(u,v,w) ; 
\end{align*}
\]
Example: Rotate

Rotate(src, dst, θ):

\[ w \approx \frac{w}{?}; \]
for(y = 0; y < ymax; y++)
  for(x = 0; x < xmax; x++)
    \[ (u, v) = \left( x \cos(-\theta) - y \sin(-\theta), x \sin(-\theta) + y \cos(-\theta) \right); \]
    \[ dst(x, y) = \text{resample}_src(u, v, w); \]
      \[ w = 1 \]

\[ x = u \cos \theta - v \sin \theta \]
\[ y = u \sin \theta + v \cos \theta \]

\[ \theta = 30 \]
Example: Fun

Fun(src, dst, θ):

\[ w \cong \?; \]
for( y=0 ; y<ymax ; y++ )
    for( x=0 ; x<xmax ; x++ )
        \((u,v) = \text{fun}(x,y)\);
        \(\text{dst}(x,y) = \text{resample_src}(u,v,w)\);
Sampling Questions

Q: Inverse mapping requires sampling the source image. Which sampling method should we use:

- Nearest Point Sampling?
- Bilinear Sampling?
- Gaussian Sampling?
- Something Else?
Outline

• Image Processing
• Image Warping

• Image Compositing
  ○ Blue-screen mattes
  ○ Alpha channel
Image Compositing

• Separate an image into “elements”
  ◦ Render independently
  ◦ Composite together

• Applications
  ◦ Cel animation
  ◦ Blue-screen matting

Bill makes ends meet by going into film
Blue-Screen Matting

• Composite foreground and background images
  ◦ Create background image
  ◦ Create foreground image with blue background
  ◦ Insert non-blue foreground pixels into background
Blue-Screen Matting

• Composite foreground and background images
  ○ Create background image
  ○ Create foreground image with blue background
  ○ Insert non-blue foreground pixels into background

Problem: lack of partial coverage results in a haloing effect along the boundary!
Alpha Channel

• Encodes pixel coverage information
  ◦ $\alpha = 0$: no coverage (or transparent)
  ◦ $\alpha = 1$: full coverage (or opaque)
  ◦ $0 < \alpha < 1$: partial coverage (or semi-transparent)

• Single Pixel Example: $\alpha = 0.3$
Compositing with Alpha

Controls the blending of foreground and background pixels when elements are composited.

\[ \alpha = 1 \]

\[ 0 < \alpha < 1 \]

\[ \alpha = 0 \]
Semi-Transparent Objects

Typically, we represent RGBA colors as not pre-multiplied by the $\alpha$ value.
Semi-Transparent Objects

Typically, we represent RGBA colors as not pre-multiplied by the $\alpha$ value.

Pixel $A = (C_A, \alpha_A)$:
Semi-Transparent Objects

Typically, we represent RGBA colors as not pre-multiplied by the $\alpha$ value.

Pixel $A = (C_A, \alpha_A)$:
- Opacity of $A$ is $\alpha_A$
Semi-Transparent Objects

Typically, we represent RGBA colors as \textbf{not} pre-multiplied by the $\alpha$ value.

Pixel $A = (C_A, \alpha_A)$:

- Opacity of $A$ is $\alpha_A$
- Transparency of $A$ is $1 - \alpha_A$
Semi-Transparent Objects

Typically, we represent RGBA colors as not pre-multiplied by the $\alpha$ value.

Pixel $A = (C_A, \alpha_A)$:
- Opacity of $A$ is $\alpha_A$
- Transparency of $A$ is $1 - \alpha_A$
- Apparent color of $A$ is $C_A \alpha_A$
Semi-Transparent Objects

Typically, we represent RGBA colors as **not** pre-multiplied by the $\alpha$ value.

So given the pixel representation $A = (C_A, \alpha_A)$ the apparent color is:

$$\bar{C}_A = C_A \alpha_A$$
Semi-Transparent Objects

Typically, we represent RGBA colors as not pre-multiplied by the $\alpha$ value.

Conversely, given the opacity, $\alpha_A$, and the apparent color, $\overline{C_A}$, the pixel representation is:

$$A = \left( \overline{C_A} \cdot \frac{1}{\alpha_A}, \alpha_A \right)$$
Semi-Transparent Objects

Typically, we represent RGBA colors as not pre-multiplied by the $\alpha$ value.

If we place pixel $A$ over pixel $B$, what is the resulting pixel value?
Semi-Transparent Objects

Pixel $A = (C_A, \alpha_A)$ over $B = (C_B, \alpha_B)$:

1. $1 - \alpha_A$
2. $\alpha_A$

$C_A$

$A$

$A$ over $B$

$C_B$

$B$

$\alpha_B$
Semi-Transparent Objects

Pixel $A = (C_A, \alpha_A)$ over $B = (C_B, \alpha_B)$:

- Opacity of $A$ in $(A$ over $B)$ is $\alpha_A$
Semi-Transparent Objects

Pixel $A = (C_A, \alpha_A)$ over $B = (C_B, \alpha_B)$:

- Opacity of $A$ in ($A$ over $B$) is $\alpha_A$
- Apparent color of $A$ in ($A$ over $B$) is $C_A \alpha_A$
Semi-Transparent Objects

Pixel $A = (C_A, \alpha_A)$ over $B = (C_B, \alpha_B)$:

- Opacity of $A$ in $(A$ over $B)$ is $\alpha_A$
- Apparent color of $A$ in $(A$ over $B)$ is $C_A \alpha_A$
- Opacity of $B$ in $(A$ over $B)$ is $(1 - \alpha_A) \alpha_B$
Semi-Transparent Objects

Pixel $A = (C_A, \alpha_A)$ over $B = (C_B, \alpha_B)$:

- Opacity of $A$ in $(A$ over $B)$ is $\alpha_A$
- Apparent color of $A$ in $(A$ over $B)$ is $C_A \alpha_A$
- Opacity of $B$ in $(A$ over $B)$ is $(1 - \alpha_A) \alpha_B$
- Apparent color of $B$ in $(A$ over $B)$ is $C_B (1 - \alpha_A) \alpha_B$
Semi-Transparent Objects

Pixel $A = (C_A, \alpha_A)$ over $B = (C_B, \alpha_B)$:

- Opacity of $A$ in $(A$ over $B)$ is $\alpha_A$
- Apparent color of $A$ in $(A$ over $B)$ is $C_A \alpha_A$
- Opacity of $B$ in $(A$ over $B)$ is $(1 - \alpha_A) \alpha_B$
- Apparent color of $B$ in $(A$ over $B)$ is $C_B (1 - \alpha_A) \alpha_B$
- Opacity of $(A$ over $B)$ is $\alpha_A + (1 - \alpha_A) \alpha_B$
Semi-Transparent Objects

Pixel $A = (C_A, \alpha_A)$ over $B = (C_B, \alpha_B)$:
- Opacity of $A$ in $(A \text{ over } B)$ is $\alpha_A$
- Apparent color of $A$ in $(A \text{ over } B)$ is $C_A\alpha_A$
- Opacity of $B$ in $(A \text{ over } B)$ is $(1 - \alpha_A)\alpha_B$
- Apparent color of $B$ in $(A \text{ over } B)$ is $C_B(1 - \alpha_A)\alpha_B$
- Opacity of $(A \text{ over } B)$ is $\alpha_A + (1 - \alpha_A)\alpha_B$
- Apparent color of $(A \text{ over } B)$ is $C_A\alpha_A + C_B(1 - \alpha_A)\alpha_B$
Semi-Transparent Objects

Pixel $A = (C_A, \alpha_A)$ over $B = (C_B, \alpha_B)$:

- Opacity of $A$ in $(A$ over $B)$ is $\alpha_A$
- Apparent color of $A$ in $(A$ over $B)$ is $C_A \alpha_A$
- Opacity of $B$ in $(A$ over $B)$ is $(1 - \alpha_A) \alpha_B$
- Apparent color of $B$ in $(A$ over $B)$ is $C_B (1 - \alpha_A) \alpha_B$
- Opacity of $(A$ over $B)$ is $\alpha_A + (1 - \alpha_A) \alpha_B$
- Apparent color of $(A$ over $B)$ is $C_A \alpha_A + C_B (1 - \alpha_A) \alpha_B$

Pixel $(A$ over $B) = \left( \frac{C_A \cdot \alpha_A + C_B (1 - \alpha_A) \alpha_B}{\alpha_A + (1 - \alpha_A) \alpha_B}, \alpha_A + (1 - \alpha_A) \alpha_B \right)$
Image Composition “Goofs”

- Visible hard edges
- Incompatible lighting/shadows
- Incompatible camera focal lengths
- Etc.

[Kee et al., Exposing Photo Manipulation from Shading and Shadows]