Computer Graphics

Prof. Feng Liu

Fall 2021

http://www.cs.pdx.edu/~fliu/courses/cs447/

10/18/2021

Last time

- Filtering
- Resampling

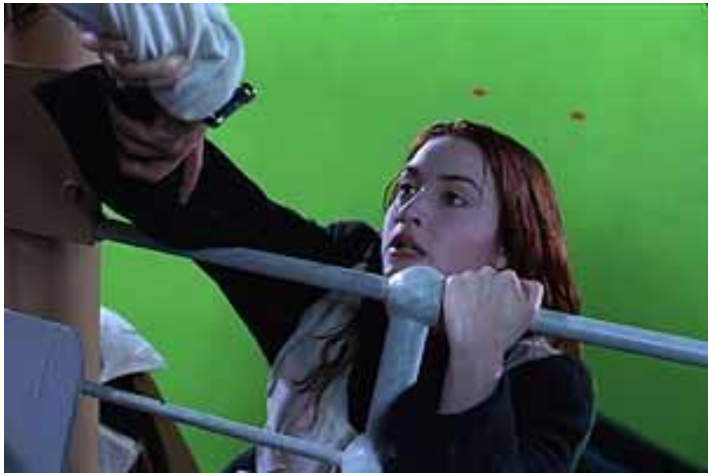
Today

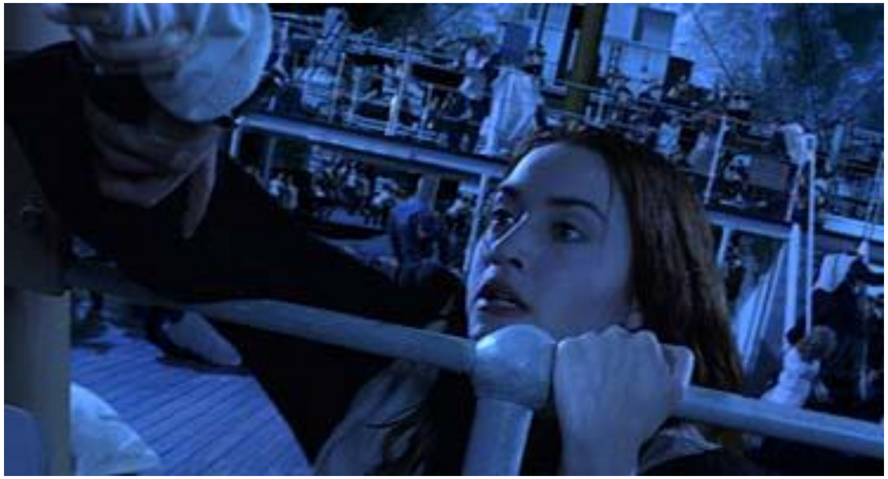
- Compositing
- NPR

From *Weta Digital*: https://www.youtube.com/watch?v=cqczPfWnQMI

Compositing

- Compositing combines components from two or more images to make a new image
  - Special effects are easier to control when done in isolation
  - Even many all live-action sequences are more safely shot in different layers

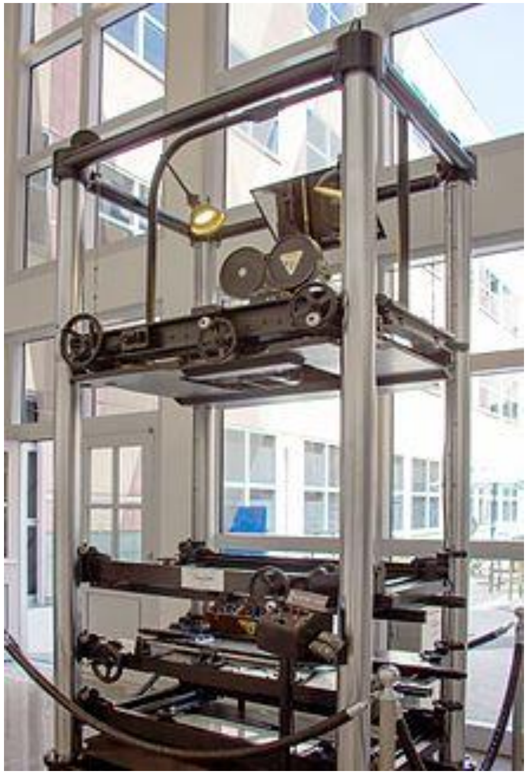




Historically …

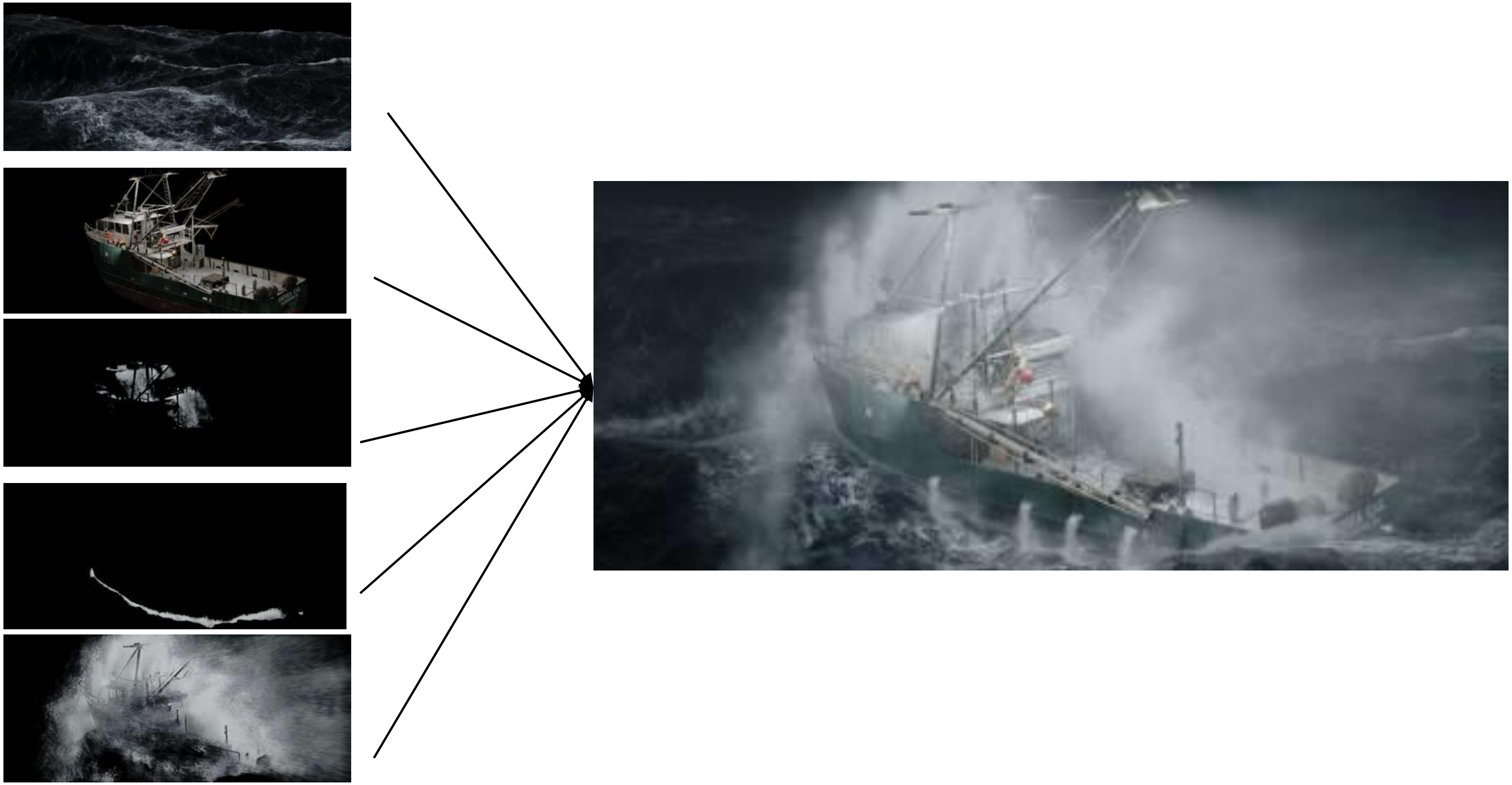
- The basis for film special effects
  - Create digital imagery and composite it into live action
  - It was necessary for films (like Star Wars) where models were used
  - It was done with film and masks, and was time consuming and expensive
- Important part of animation – even hand animation
  - Background change more slowly than foregrounds, so composite foreground elements onto constant background
  - It was a major advance in animation – the *multiplane camera* first used in Snow White (1937)

Multiplane camera

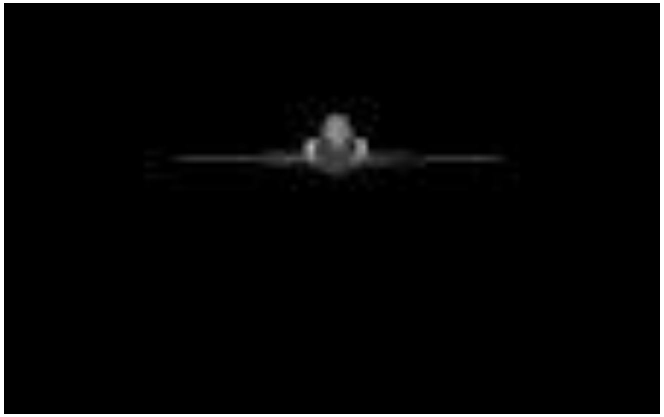


https://en.wikipedia.org/wiki/Multiplane_camera

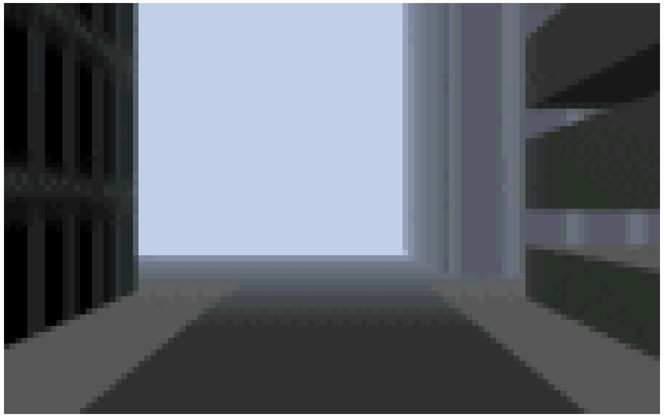
Perfect Storm



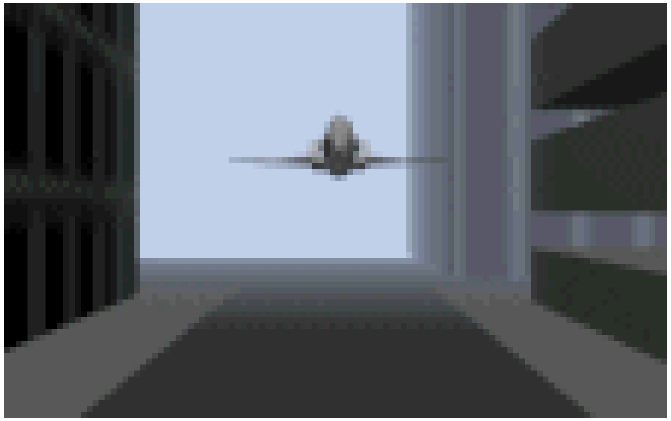
Animated Example



over

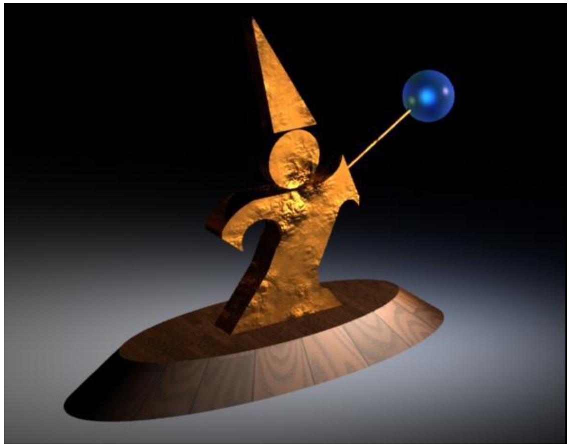


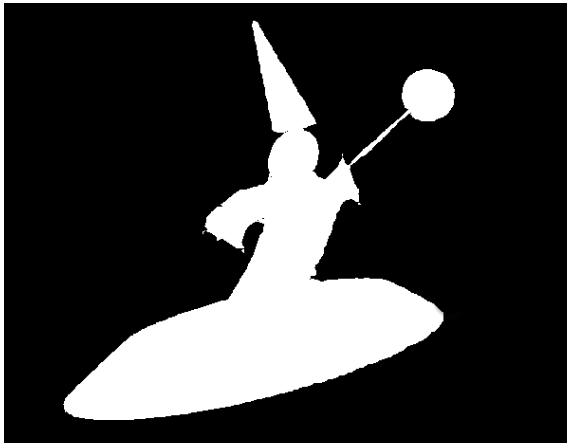
=



Mattes

- □ A *matte* is an image that shows which parts of another image are foreground objects
- □ Term dates from film editing and cartoon production
- □ How would I use a matte to insert an object into a background?
- □ How are mattes usually generated for television?



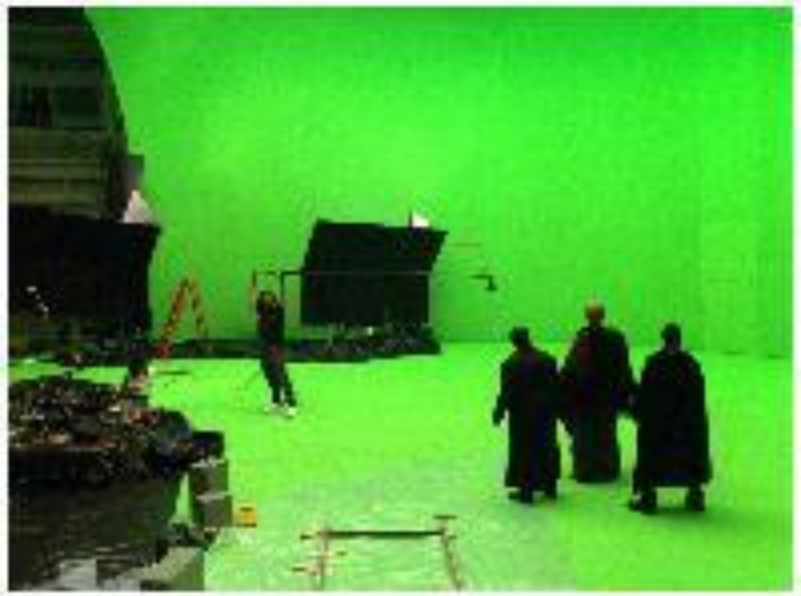


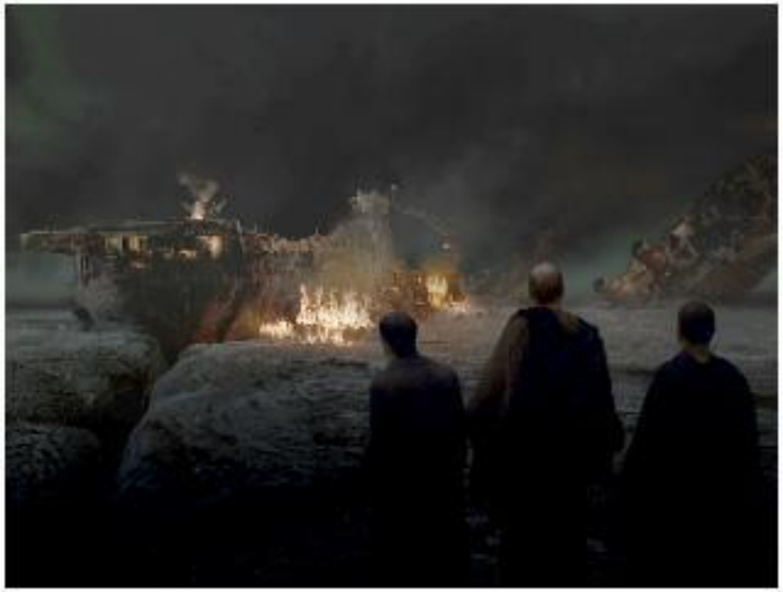
Working with Mattes

- To insert an object into a background
  - Call the image of the object the source
  - Put the background into the destination
  - For all the source pixels, if the matte is white, copy the pixel, otherwise leave it unchanged
- To generate mattes:
  - Use smart selection tools in Photoshop or similar
    - They outline the object and convert the outline to a matte
  - **Blue Screen:** Photograph/film the object in front of a blue background, then consider all the blue pixels in the image to be the background

Compositing

- Compositing is the term for combining images, one over the other
  - Used to put special effects into live action





Alpha

- Basic idea: Encode opacity information in the image
- Add an extra channel, the *alpha* channel, to each image
  - For each pixel, store R, G, B and Alpha
  - alpha = 1 implies full opacity at a pixel
  - alpha = 0 implies completely clear pixels
- There are many interpretations of alpha
  - Is there anything in the image at that point (web graphics)
  - Transparency (real-time OpenGL)
- Images are now in RGBA format, and typically 32 bits per pixel (8 bits for alpha)
- All images in the project are in this format

Pre-Multiplied Alpha

- Instead of storing (R,G,B,α), store (αR,αG,αB,α)
- The compositing operations in the next several slides are easier with pre-multiplied alpha
- **To display and do color conversions, must extract RGB by dividing out α**
  - α=0 is always black
  - Some loss of precision as α gets small, but generally not a big problem

Compositing Assumptions

- We will combine two images, *f* and *g*, to get a third *composite image*
- Both images are the same size and use the same color representation
- Multiple images can be combined in stages, operating on two at a time

Basic Compositing Operation

- □ At each pixel, combine the pixel data from *f* and the pixel data from *g* with the equation:

$$c_o = Fc_f + Gc_g$$

- □ *F* and *G* describe how much of each input image survives, and c_f and c_g are **pre-multiplied pixels**, and **all four channels** are calculated
- □ **To define a compositing operation, define *F* and *G***

Basic Compositing Operation

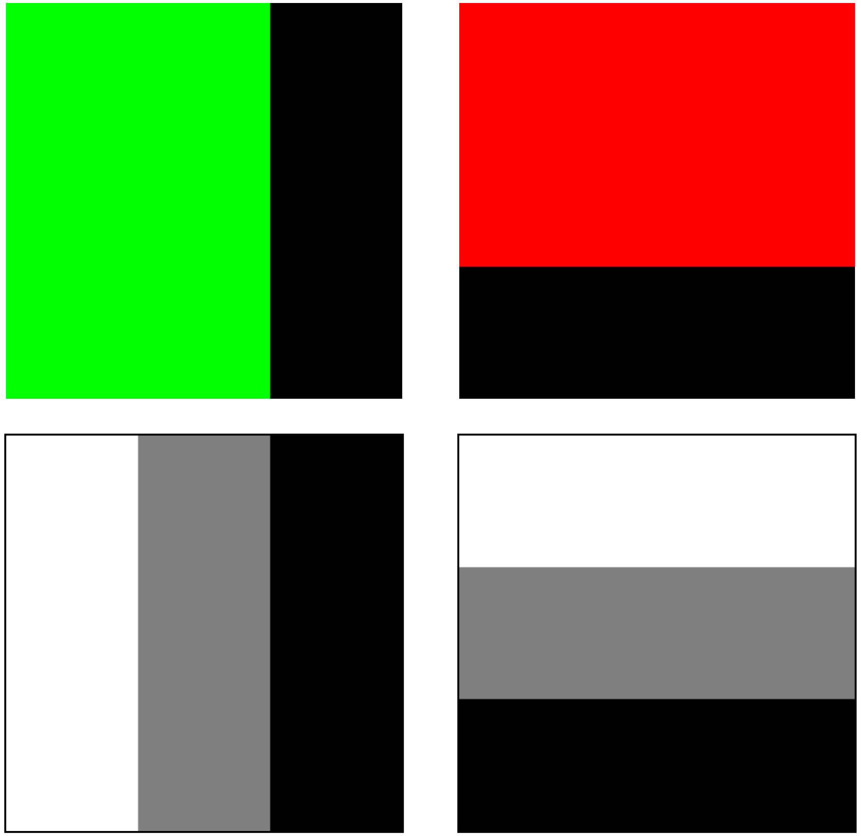
- *F* and *G* are simple **functions** of the alpha values

$$c_o = F(\alpha_f, \alpha_g)c_f + G(\alpha_f, \alpha_g)c_g$$

- *F* and *G* are chosen (independently)
- Different choices give different operations
- To code it, you can write one compositor and give it 6 numbers (3 for *F*, 3 for *G*) to say which function
  - Constant of 0 or 1
  - α_f is multiplied by -1, 0 or 1. Similar for α_g

	$1\alpha_f\alpha_g$
0	$0,0,0$
1	$1,0,0$
α_f	$0,1,0$
$1-\alpha_f$	$1,-1,0$
α_g	$0,0,1$
$1-\alpha_g$	$1,0,-1$

Sample Images

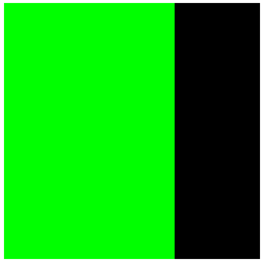


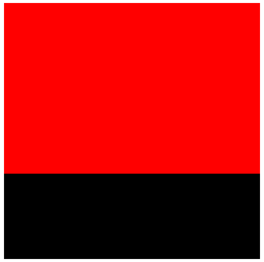
Images

Alphas

Sample Images

Images



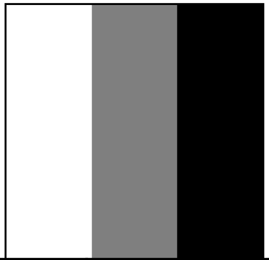


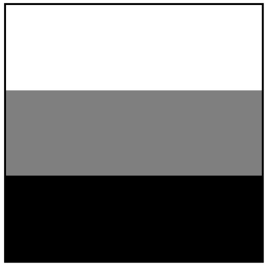
RGB

0,1,0	0,1,0	0, 0, 0
0,1,0	0,1,0	0 ,0, 0
0,1,0	0,1,0	0, 0 ,0

1,0,0	1,0,0	1,0,0
1,0,0	1,0,0	1,0,0
0, 0 ,0	0, 0 ,0	0, 0 ,0

Alphas



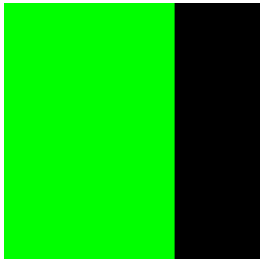


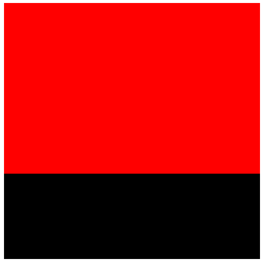
1	0.5	0
1	0.5	0
1	0.5	0

1	1	1
0.5	0.5	0 .5
0	0	0

Sample Images

Images



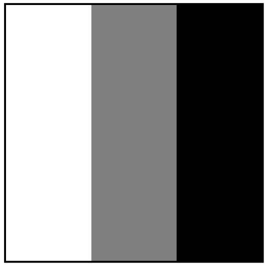


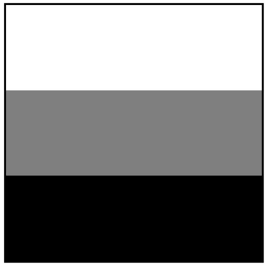
Pre-multiplied RGBA

0,1,0 1	0,0.5,0 0.5	0, 0, 0 0
0,1,0 1	0,0.5,0 0.5	0 ,0, 0 0
0,1,0 1	0,0.5,0 0.5	0, 0 ,0 0

1,0,0 1	1,0,0 1	1,0,0 1
0.5,0,0 0.5	0.5,0,0 0.5	0.5,0,0 0.5
0, 0 ,0 0	0, 0 ,0 0	0, 0 ,0 0

Alphas





“Over” Operator

□ Computes composite with the rule that *f* covers *g*

$$F = 1$$

$$G = 1 - \alpha_f$$

$$\Longrightarrow \quad c_o = Fc_f + Gc_g = c_f + (1 - \alpha_f)c_g$$

“Over” Operator

$$c_o = Fc_f + Gc_g = c_f + (1-\alpha_f)c_g$$

0,1,0 1	0,0.5,0 0.5	0, 0, 0 0
0,1,0 1	0,0.5,0 0.5	0 ,0, 0 0
0,1,0 1	0,0.5,0 0.5	0, 0 ,0 0

f

Over

1,0,0 1	1,0,0 1	1,0,0 1
0.5,0,0 0.5	0.5,0,0 0.5	0.5,0,0 0.5
0, 0 ,0 0	0, 0 ,0 0	0, 0 ,0 0

g

→

0,1,0 1	0.5,0.5,0 1	1,0,0 1
0,1,0 1	0.25,0.5,0 0.75	0.5,0,0 0.5
0,1,0 1	0, 0.5 ,0 0.5	0, 0 ,0 0

result

“Over” Operator

$$c_o = Fc_f + Gc_g = c_f + (1-\alpha_f)c_g$$

0,1,0 1	0,0.5,0 0.5	0, 0, 0 0
0,1,0 1	0,0.5,0 0.5	0 ,0, 0 0
0,1,0 1	0,0.5,0 0.5	0, 0 ,0 0

f

Over

1,0,0 1	1,0,0 1	1,0,0 1
0.5,0,0 0.5	0.5,0,0 0.5	0.5,0,0 0.5
0, 0 ,0 0	0, 0 ,0 0	0, 0 ,0 0

g

⇨

0,1,0 1	0.5,0.5,0 1	1,0,0 1
0,1,0 1	0.25,0.5,0 0.75	0.5,0,0 0.5
0,1,0 1	0, 0.5 ,0 0.5	0, 0 ,0 0

result

$\boxed{\begin{matrix}0,\ 0.5,\ 0\\0.5\end{matrix}} \quad + \quad (1 - 0.5)\boxed{\begin{matrix}1,\ 0,\ 0\\0\end{matrix}} \quad ⇨ \quad \boxed{\begin{matrix}0.5,\ 0.5,\ 0\\1\end{matrix}}$

“Over” Operator: Extract RGB Color

0,1,0 1	0.5,0.5,0 1	1,0,0 1
0,1,0 1	0.25,0.5,0 0.75	0.5,0,0 0.5
0,1,0 1	0, 0.5 ,0 0.5	0, 0 ,0 0

Pre-multiplied RGBA

0,1,0	0.5,0.5,0	1,0,0
0,1,0	1/3,2/3,0	1,0,0
0,1,0	0, 1,0	0, 0 ,0

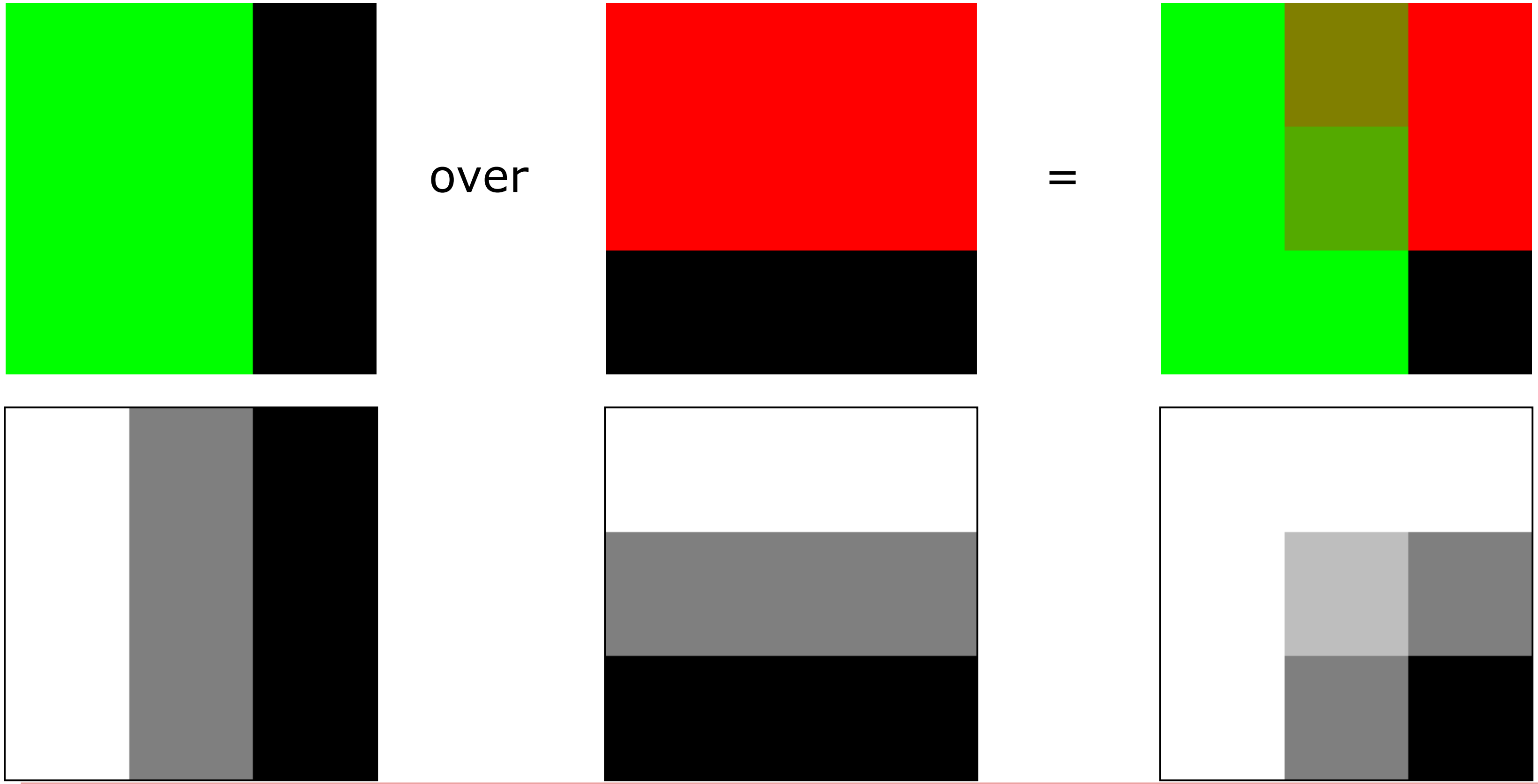
RGB

0,1,0 1	0.5,0.5,0 1	1,0,0 1
0,1,0 1	0.25,0.5,0 0.75	0.5,0,0 0.5
0,1,0 1	0, 0.5 ,0 0.5	0, 0 ,0 0

Alpha

“Over” Operator

- If there’s some *f*, get *f*, otherwise get *g*



“Inside” Operator

- Computes composite with the rule that only parts of *f* that are inside *g* contribute

$$F = \alpha_g$$

$$G = 0$$

“Inside” Operator

$$c_o = Fc_f + Gc_g = \alpha_g c_f$$

0,1,0 1	0,0.5,0 0.5	0, 0, 0 0
0,1,0 1	0,0.5,0 0.5	0 ,0, 0 0
0,1,0 1	0,0.5,0 0.5	0, 0 ,0 0

f

Over

1,0,0 1	1,0,0 1	1,0,0 1
0.5,0,0 0.5	0.5,0,0 0.5	0.5,0,0 0.5
0, 0 ,0 0	0, 0 ,0 0	0, 0 ,0 0

g

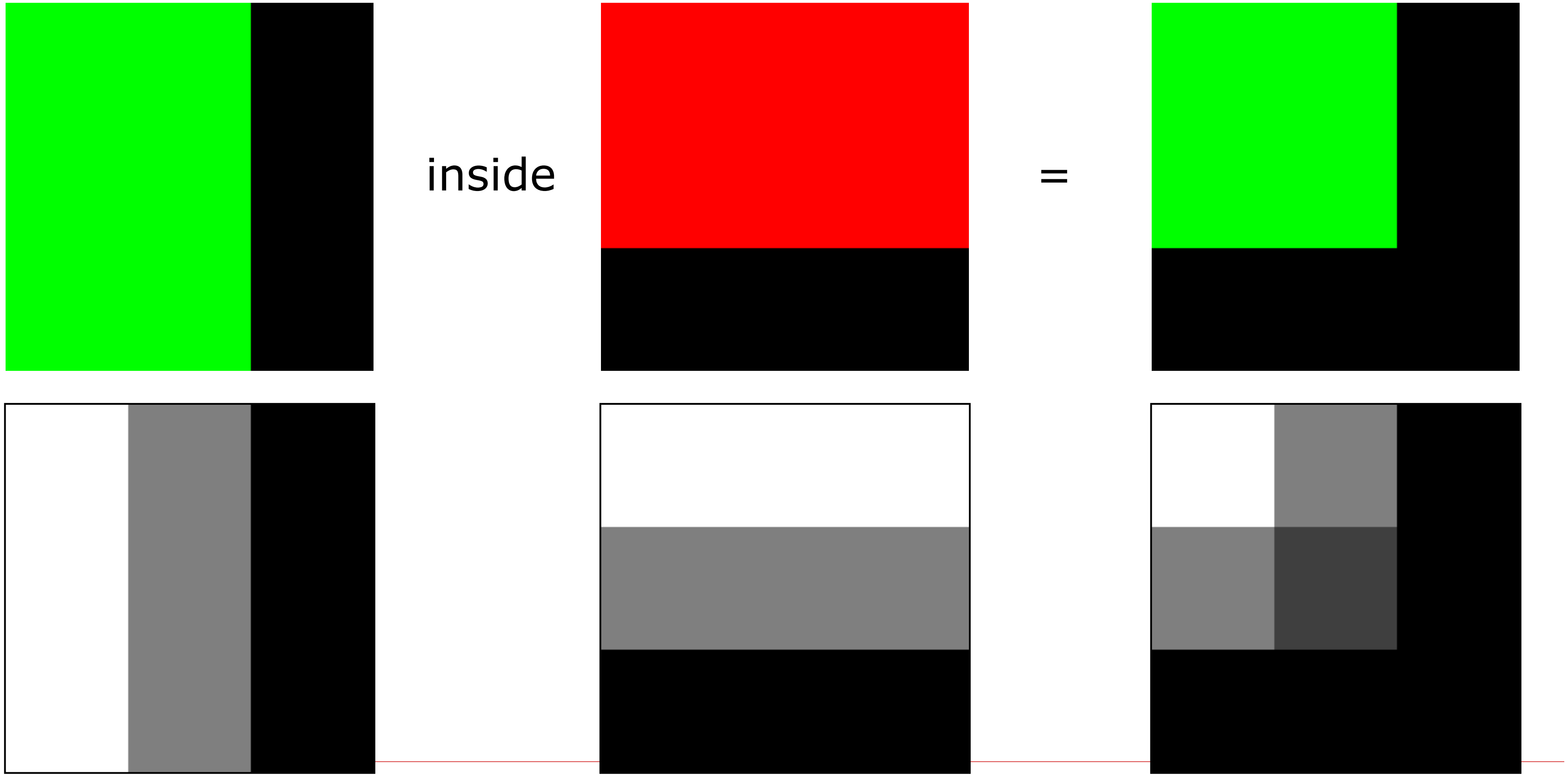
→

0,1,0 1	0,0.5,0 0.5	0,0,0 0
0,0.5,0 0.5	0,0.25,0 0.25	0,0,0 0
0, 0 ,0 0	0, 0 ,0 0	0, 0 ,0 0

result

“Inside” Operator

- Get *f* to the extent that *g* is there, otherwise nothing



inside

=

“Outside” Operator

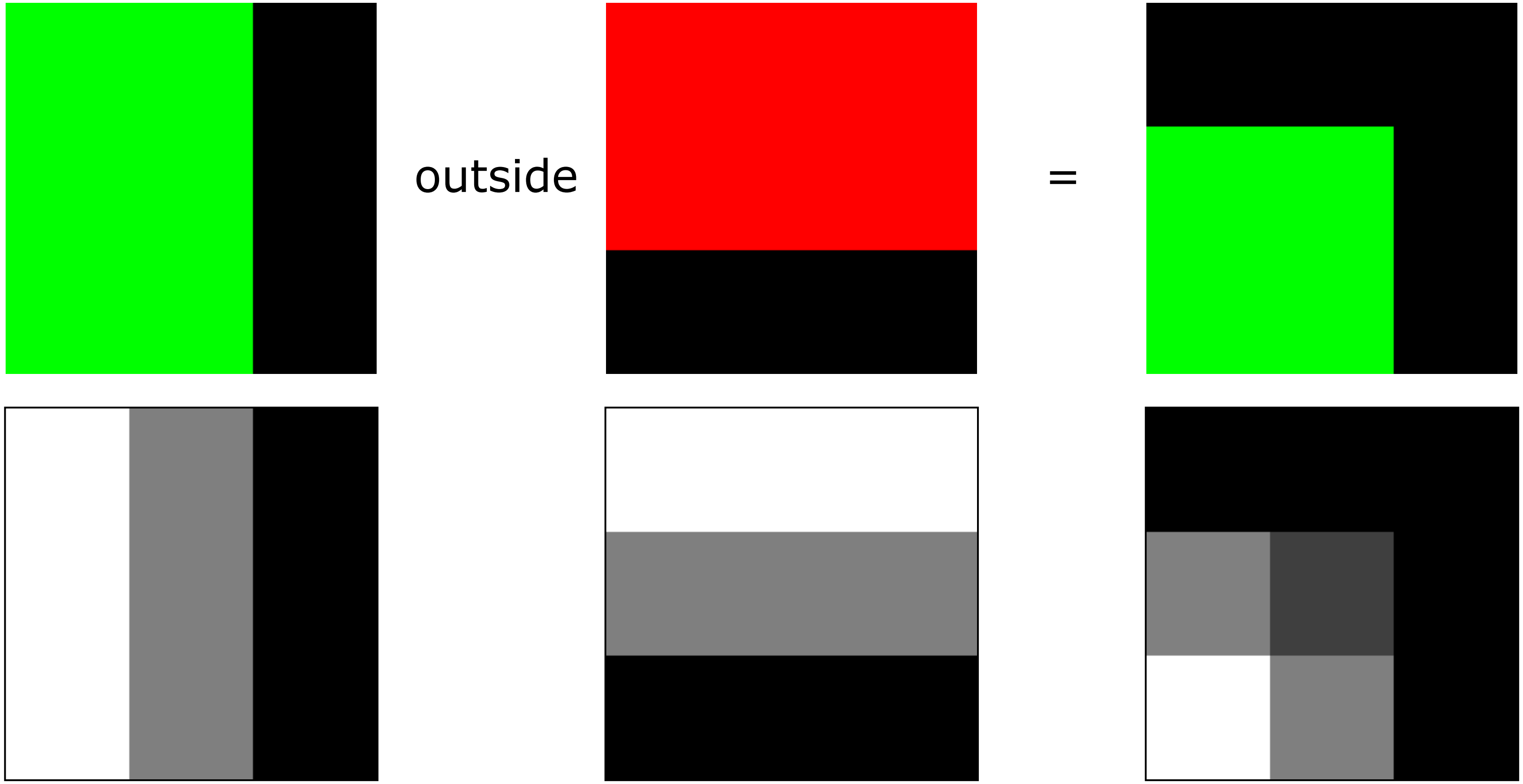
- Computes composite with the rule that only parts of *f* that are outside *g* contribute

$$F = 1 - \alpha_g$$

$$G = 0$$

“Outside” Operator

- □ Get *f* to the extent that *g* is **not** there, otherwise nothing



“Atop” Operator

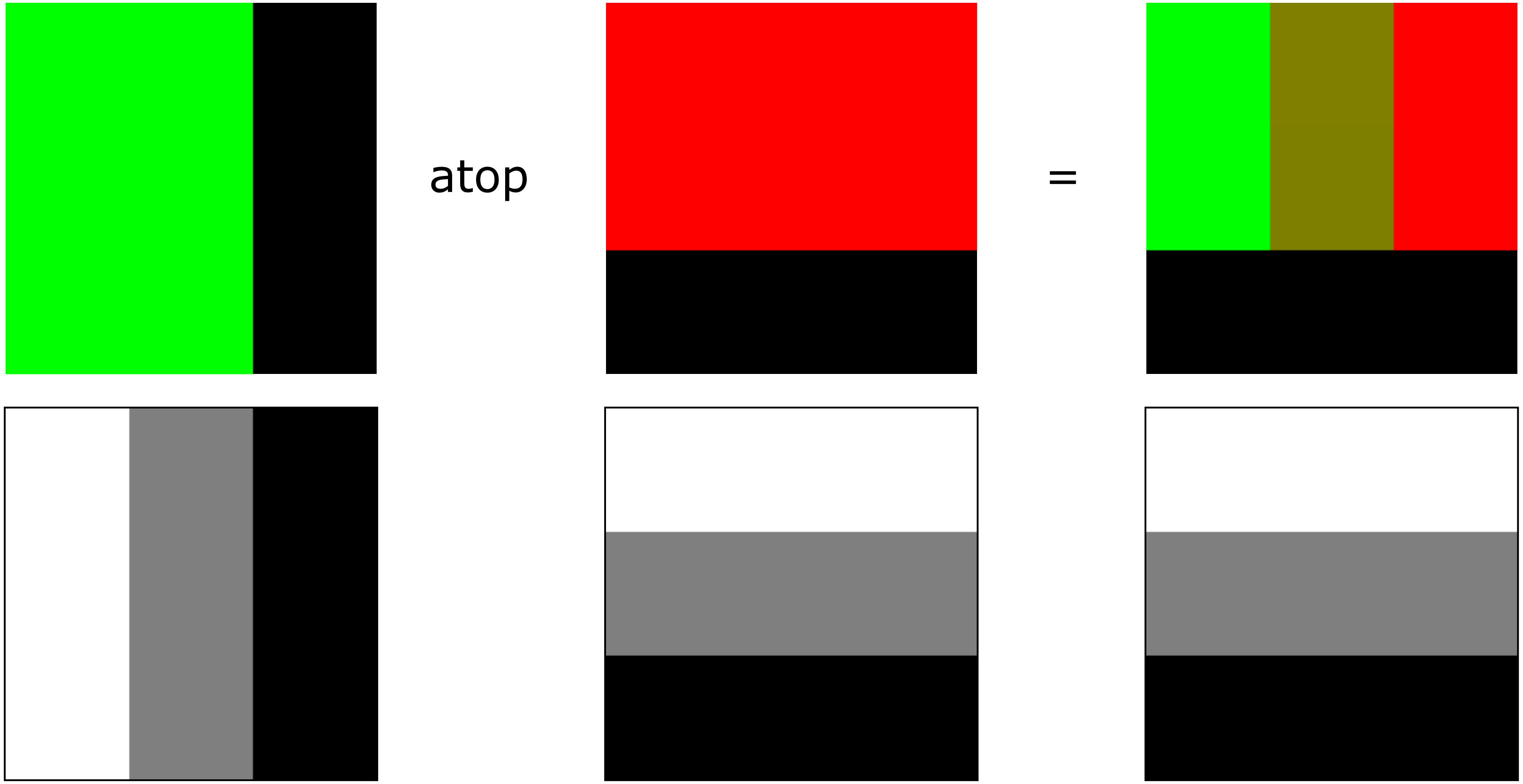
- □ Computes composite with the *over* rule but restricted to places where there is some *g*

$$F = \alpha_g$$

$$G = 1 - \alpha_f$$

“Atop” Operator

- Get *f* to the extent that *g* is there, otherwise *g*



“Xor” Operator

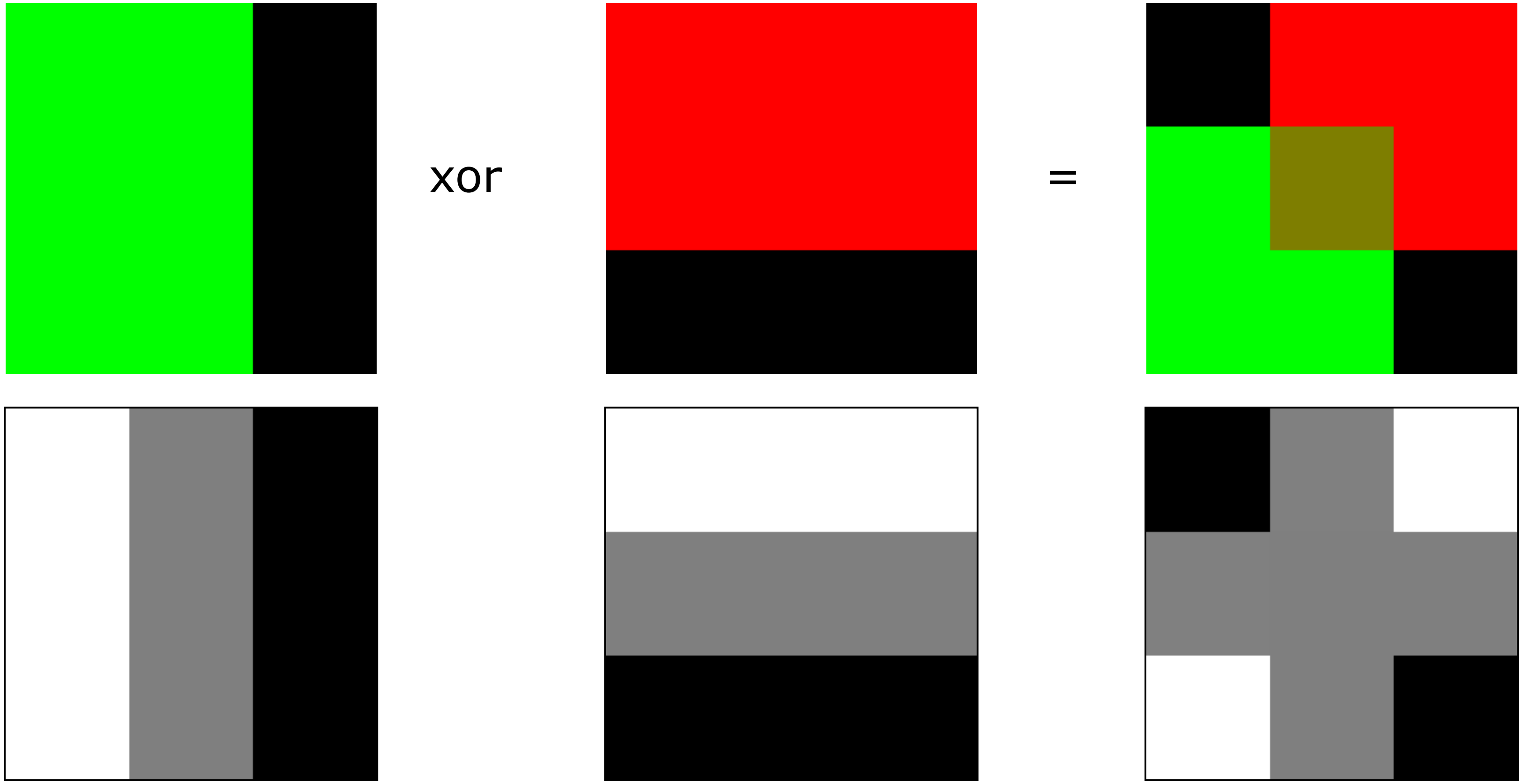
- Computes composite with the rule that *f* contributes where there is no *g*, and *g* contributes where there is no *f*

$$F = 1 - \alpha_g$$

$$G = 1 - \alpha_f$$

“Xor” Operator

- Get *f* to the extent that *g* is not there, and *g* to extent of no *f*



“Clear” Operator

- Computes a clear composite

$$F = 0$$

$$G = 0$$

- Note that $(0,0,0,\alpha>0)$ is a partially opaque black pixel, whereas (0,0,0,0) is fully transparent, and hence has no color

“Set” Operator

- Computes composite by setting it to equal *f*

$$F = 1$$

$$G = 0$$

- Copies *f* into the composite

Compositing Operations

- *F* and *G* describe how much of each input image survives, and c_f and c_g are **pre-multiplied pixels**, and **all four channels** are calculated

$$c_o = Fc_f + Gc_g$$

Operation	F	G
Over	1	$1-\alpha_f$
Inside	α_g	0
Outside	$1-\alpha_g$	0
Atop	α_g	$1-\alpha_f$
Xor	$1-\alpha_g$	$1-\alpha_f$
Clear	0	0
Set	1	0

Unary Operators

- □ Darken: Makes an image darker (or lighter) without affecting its opacity

$$darken(f,\phi) \equiv (\phi r_f, \phi g_f, \phi b_f, \alpha_f)$$

- □ Dissolve: Makes an image transparent without affecting its color

$$dissolve(f,\delta) \equiv (\delta r_f, \delta g_f, \delta b_f, \delta\alpha_f)$$

“PLUS” Operator

- Computes composite by simply adding *f* and *g*, with no overlap rules

$$c_o = c_f + c_g$$

- Useful for defining *cross-dissolve* in terms of compositing:

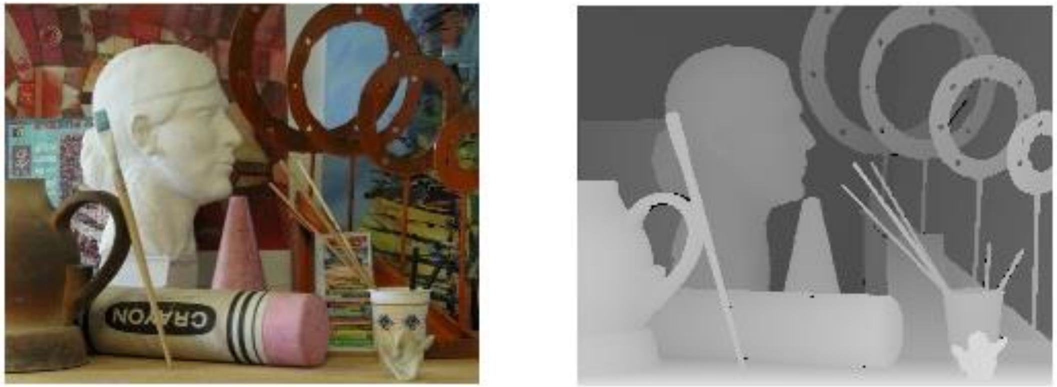
$$cross(f,g,t) = dissolve(f,t) \text{ plus } dissolve(g,1-t)$$

Obtaining α Values

- Hand generate (paint a grayscale image)
- Automatically create by segmenting an image into foreground background:
  - Blue-screening is the analog method
    - Remarkably complex to get right
  - “Lasso” is the Photoshop operation
- With synthetic imagery, use a special background color that does not occur in the foreground
  - Brightest blue or green is common

Compositing With Depth

- Can store pixel “depth” instead of alpha
- Then, compositing can truly take into account foreground and background
- Generally only possible with synthetic imagery
  - Image Based Rendering is an area that, in part, tries to composite photographs taking into account depth



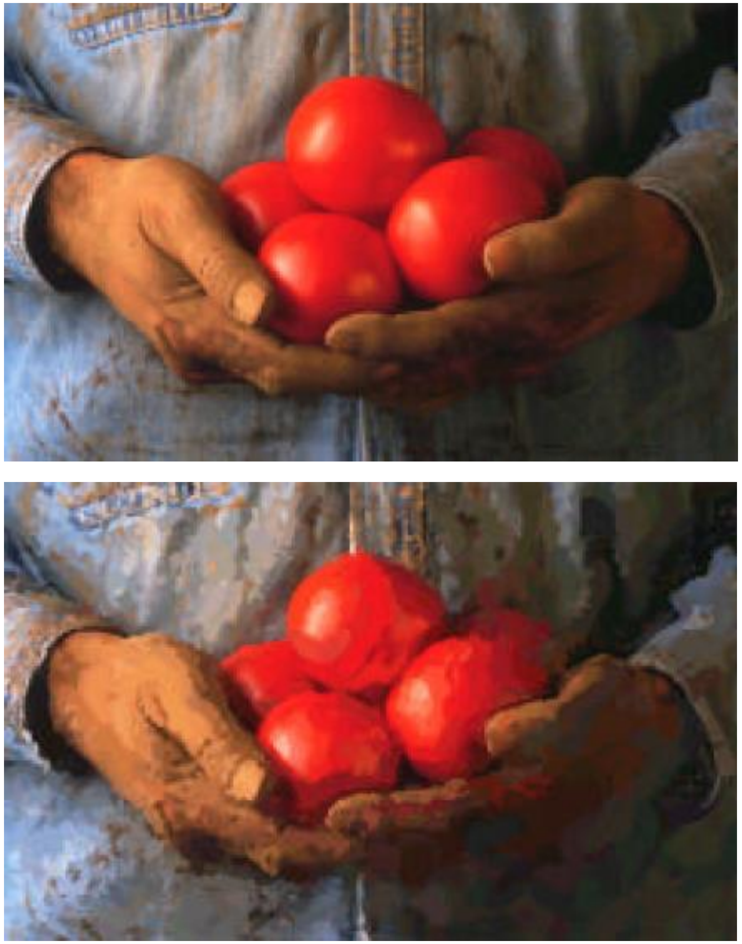
A photo and its depth map

Today

- More Compositing
- Non-photorealistic Rendering (NPR)

Painterly Filters

- Many methods have been proposed to make a photo look like a painting
- Today we look at one: *Painterly-Rendering with Brushes of Multiple Sizes**
- Basic ideas:
  - Build painting one layer at a time, from biggest to smallest brushes
  - At each layer, add detail missing from previous layer



*Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes, SIGGRAPH 1998

Algorithm 1

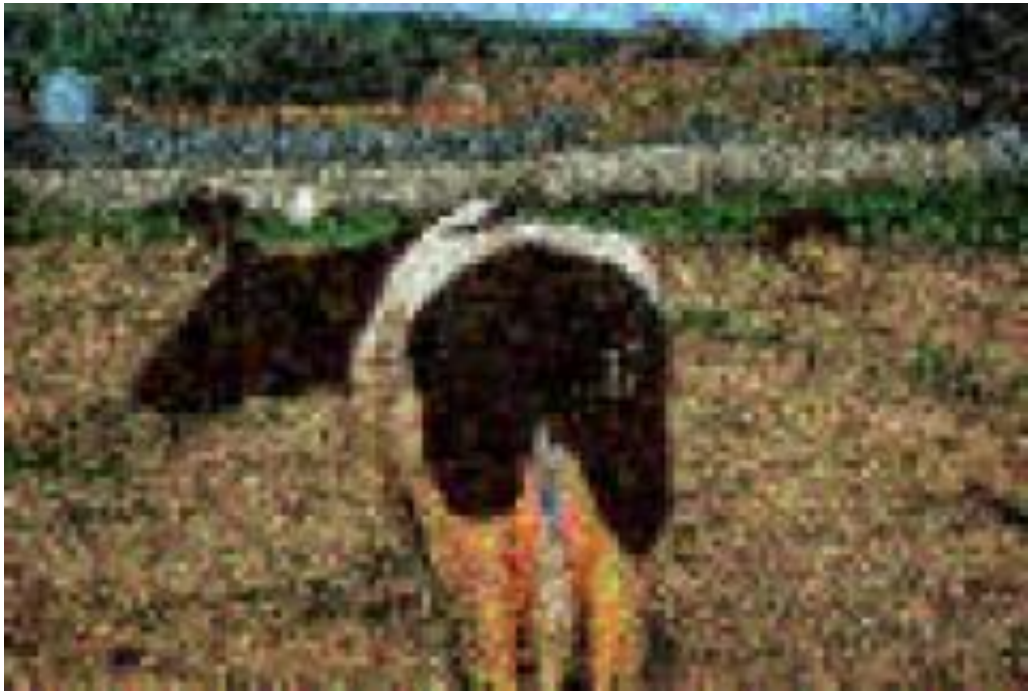
```
function paint(sourceImage,$R_1$ ... $R_n$) // take source and several brush sizes
{
        canvas := a new constant color image
        // paint the canvas with decreasing sized brushes
        for each brush radius $R_i$, from largest to smallest do
        {
        // Apply Gaussian smoothing with a filter of size const * radius
        // Brush is intended to catch features at this scale
                referenceImage = sourceImage * G(fs $R_i$)
                // Paint a layer
                paintLayer(canvas, referenceImage, Ri)
        }
        return canvas
}
```

Algorithm 2

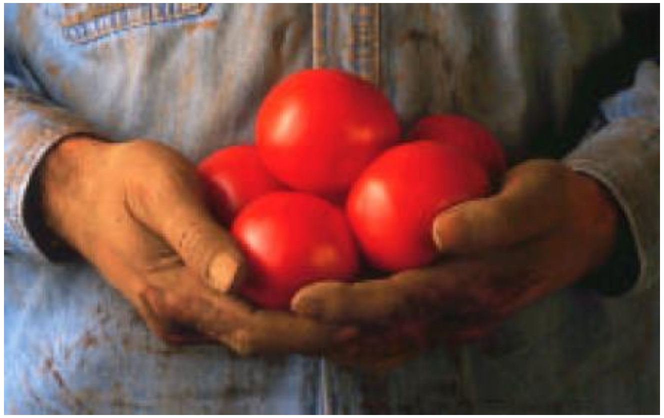
```
procedure paintLayer(canvas,referenceImage, R) // Add a layer of strokes
{
    S := a new set of strokes, initially empty
    D := difference(canvas,referenceImage) // euclidean distance at every pixel
        for x=0 to imageWidth stepsize grid do  // step in size that depends on brush radius
            for y=0 to imageHeight stepsize grid do {
            // sum the error near (x,y)
            M := the region (x-grid/2..x+grid/2, y-grid/2..y+grid/2)
                areaError := sum($D_{i,j}$ for i,j in M) / $grid^2$
                if (areaError > T) then {
                    // find the largest error point
                    (x1,y1) := max $D_{i,j}$ in M
                    s :=makeStroke(R,x1,y1,referenceImage)
                    add s to S
                }
            }
        paint all strokes in S on the canvas, in random order
}
```

Point Style

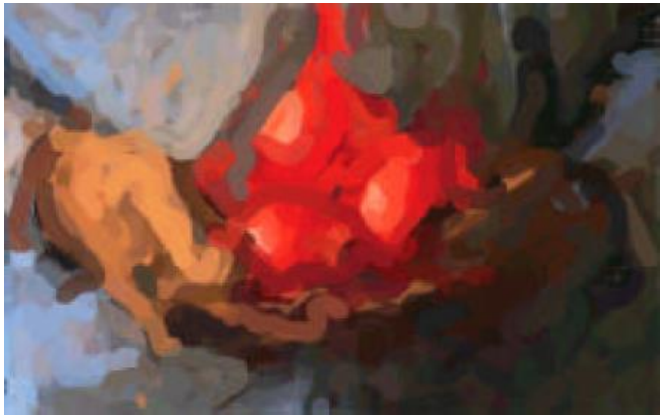
- Uses round brushes
- We provide a routine to "paint" round brush strokes into an image for the project



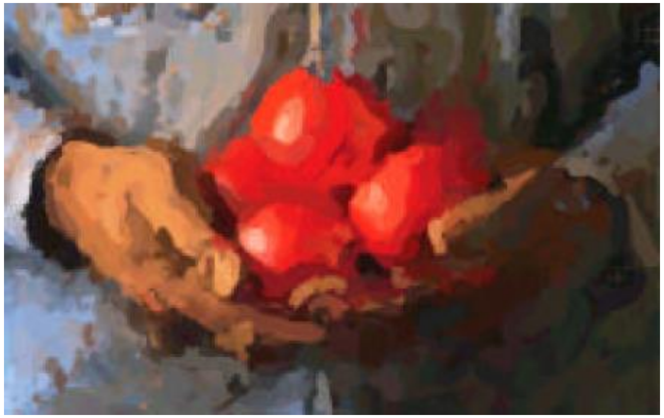
Results



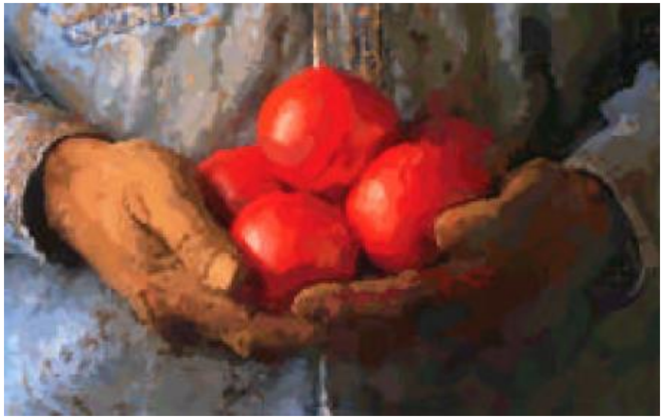
Original



Biggest brush



Medium brush added



Finest brush added

Next Time

- 3D Graphics Toolkits
  - Transformations
- Composing transformations
- 3D Transformations
- Viewing