NAME

imcomp - composite two images together

SYNOPSIS

imcomp [*options*] *infile1name infile2name outfilename*

DESCRIPTION

imcomp combines two input images together using a digital compositing operation that uses the alpha channels (coverage masks) of one or both input images. The resulting composite image is stored to the output file. Input and output image file formats may be different.

OPTIONS

imcomp has a variety of options in the following categories:

File Selection	What input and output files to use
Format Selection	What image file format to use
Format Control	What variant of a file format to generate
Standard	Standard generic options on all SDSC tools
Compositing	How to combine two images
Field(s)	Which field(s) are composited
Positioning	How to position and size foreground

File Selection, *Format Selection*, *Format Control*, and *Standard* options are common to all SDSC image tools and are discussed in depth in the man page for imconv(1IM).

All options can be abbreviated to the first few unique characters.

Compositing

Compositing is a technique that combines pixels of two input images together to create a third *composite* image. Compositing is commonly used to take two separately computed images and combine them, with portions of one obscuring portions of the other. In such cases the first input image would act as a *foreground* image, and the second as a *background* image.

Foreground and background images can be composited together to create a variety of special effects, or as an aid to speed up the rendering of complex scenery. For instance, a complex computer-generated robot must walk across the screen in front of a computer-generated library filled with books. The library could be modeled, book by book, and the robot modeled, joint for joint. An image showing the robot in the library could be rendered by combining both models and sending the entire mass of geometry off to the renderer all at once. A single image would result.

To render an animation of the robot moving in front of the library one could send both masses of geometry to the renderer for each frame. However, the library is static in this scene and the interaction between the robot and its books nil. The time spent rendering, and rerendering, each library book, frame after frame is a waste and dramatically increases the rendering time of the animation.

A more efficient approach would be to render the static library just once, without the robot. The robot would then be rendered for each frame, but without the library behind it (just a solid black background). Once rendering is complete, the separate foreground (robot) and background (library) images can be composited together to create a series of frames showing the robot moving in front of the library. The total rendering time for such an approach is considerably less.

Such compositing tricks are regularly used in Hollywood in order to place actors in front of locations they aren't in, or to make spaceships fly all about in front of swirling planets or sparkley star fields. In video the same technique, called *keying*, is used to place the weather man in front of a computer-generated weather map.

In each of these cases, the compositing operation must know what portions of the foreground image to paste over the background image. This information is supplied in a *coverage mask*, historically referred to as an *alpha channel* or sometimes a *matte*.

Alpha channels are usually generated at the same time the color portion of the image is rendered. Alpha information is stored as a one-byte integer value "behind" each image pixel. An alpha value of 0 indicates the pixel is *transparent*, or wild. An alpha value of 255 indicates the pixel is *opaque*. For instance, in the robot example, all pixels showing colors for the robot itself would have alpha values of 255. All pixels for the blank background behind the robot (remember the robot's rendered separately from the library) would be transparent, or wild, and have alpha values of 0.

When compositing a foreground image over a background image, the portions of the foreground image that are opaque (alpha = 255) will obscure the background image. Portions of the foreground image that are transparent (alpha = 0) will let the background image show through.

Alpha values between 0 and 255 indicate a level of transparency. Foreground image pixels with alpha values of 128, for instance, would be blended with background pixels to create a pseudo-translucency effect. This is most often used to blend the edges of foreground objects so that crisp, noticeable jaggies don't result.

imcomp supports four compositing algorithms considered standard in digital compositors:

Over
Atop
Inside
Outside

The *over* algorithm is the one most often used and the one appropriate for placing portions of a foreground image over a background image.

atop is rarely used, but can be appropriate when certain special effects are desired.

inside and *outside* are opposites of each other and are typically used when creating *mattes*, black and white masks that isolate portions of an image (see matte descriptions in the NOTES section below).

Each of these composition algorithms are discussed below. Discussions use the following nomenclature:

In1Field
: A pixel's field (usually a color attribute) from input image 1.

In2Field
: A pixel's field (usually a color attribute) from input image 2.

OutField
: A pixel's field (usually a color attribute) for the output image.

In1Alpha
: A pixel's alpha field from input image 1.

In2Alpha
: A pixel's alpha field from input image 2.

Over

-over selects the *over* compositing algorithm. *Over* is used to place opaque portions of the first input image over the second input image. *Over* is by far the most common compositing operation and is the default if no specific composite operation is given on the command-line.

Over uses the following formula for each pixel:

```
OutField = In1Field * (In1Alpha/255.0) +
           In2Field * (1.0 - (In1Apha/255.0))
```

The first input image must have an alpha channel. imcomp will report an error and exit if it does not.

An alpha channel on the second input image is not needed or used by the *over* operation.

Atop

-atop selects the *atop* compositing algorithm. *Atop* is used to combine the opaque portions of the two input images, merging their colors by addition of their field values. The *atop* operation is mostly used for special effects and has a rather non-intuitive effect that can cause unexpected color shifts.

Atop uses the following formula for each pixel:

```
OutField = In1Field * (In2Alpha/255.0) +
           In2Field * (1.0 - (In1Alpha/255.0))
```

Both input images must have alpha channels. imcomp will report an error and exit if they do not.

Inside

-inside selects the *inside* compositing algorithm. *Inside* uses pixels from the first image whenever the second image is opaque. This has the effect of substituting the first image into the second image in a controlled way and is most often used when creating *mattes*, a special form of coverage mask that may be used in chains of compositing operations. *inside*'s specialized function makes it rare for typical compositing work.

Inside uses the following formula for each pixel:

```
OutField = In1Field * (In2Alpha/255.0)
```

The second input image must have an alpha channel. imcomp will report an error and exit if it does not.

An alpha channel on the first input image is not needed or used by the *inside* operation.

Outside

-outside selects the *outside* compositing algorithm. *Outside* is the reverse of *inside*. Where *inside* uses pixels from the first image where the second is opaque, *outside* uses pixels from the first image where the second is transparent. Like *inside*, *outside* is most often used for creating *mattes* for use in chains of compositing operations.

Outside uses the following formula for each pixel:

```
OutField = In1Field * (1.0 - (In2Alpha/255.0))
```

The second input image must have an alpha channel. imcomp will report an error and exit if it does not.

An alpha channel on the first input image is not needed or used by the *outside* operation.

Field(s)

imcomp applies the compositing algorithm to fields of the input images in order to generate the output image. By default, all fields in the input images are composited into the output image. Users may restrict or alter imcomp's field use by compositing on specific fields using one or more of the following options:

Option	Composite
-mono	the monochrome field
-index	the color index field
-alpha	the alpha field
-red	the red field
-green	the green field
-blue	the blue field
-hue	on the virtual hue field
-saturation	on the virtual saturation field
-intensity	on the virtual intensity field

In general, composite fields named by any of the above options must exist in both input images. For instance, compositing using -index for color index fields doesn't make sense on RGB images. A certain amount of automatic image type conversion is performed by imcomp in order to make things a little bit more flexible:

-mono Monochrome compositing requires that both input images have monochrome image fields.

-index Color index compositing requires that both input images have color index image fields.

-alpha Alpha channel compositing requires that both input images have alpha image fields. Please note that this is different from the compositing operation's use of alpha channels. The composite operation uses alpha values to decide how to affect a particular field of a pixel. -alpha selects that that field in fact be the alpha channel itself.

-red, -green, -blue
Compositing on red, green, or blue fields works for any image, whether monochrome, grayscale, color indexed, or RGB. Such images are promoted internally to RGB images in order to accomplish the compositing.

-hue, -saturation, -intensity
Compositing on the virtual image fields of hue, saturation, or intensity works for any image type. Input image color field values are converted internally to the HSI color space in order to accomplish the compositing.

Compositing may occur on more than one field at once with the restriction that the four color spaces not be mixed on the same command-line: monochrome, color indexes, RGB, and HSI.

Illegal

imcomp one.x two.x -red -index out.x imcomp one.x two.x -hue -mono out.x imcomp one.x two.x -index -saturation -red out.x

Legal

imcomp one.x two.x -red -green -blue out.x imcomp one.x two.x -hue out.x imcomp one.x two.x -index -alpha out.x imcomp one.x two.x -saturation -alpha out.x

Positioning

There also exist provisions for positioning and sizing the foreground image. This is useful since sometimes only a part of the foreground is desired in the composite. The positioning of the foreground in respect to the background is useful if the foreground needs to be somewhere other than the upper left hand corner of the image.

-xposition x	Specify left edge of foreground into background
-yposition y	Specify top edge of foreground into background
-xstart x	Specify left edge in foreground data
-ystart y	Specify top edge in foreground data
-xsize width	Specify width of foreground data
-ysize height	Specify height of foreground data

-xposition -yposition

Allows you to position the foreground in the background. The foreground can be positioned at a new x and y position other than the default, which is 0 and 0. In other words, if these arguments are not specified, then the foreground image will be composited onto the background extending from the upper left hand corner. Quantities are specified in pixels.

-xstart -ystart

Allows you to pick an upper left hand corner of the foreground. All compositing of the foreground will start from this corner. The default for xstart is 0, or the left hand edge. The default for ystart is 0, or the top edge. Quantities are specified in pixels.

-xsize -ysize

Allows you to choose how much of the foreground to composite onto the background, extending from the corner specified by xstart and ystart. (Note that this corner will be 0,0 if there were no xtstart or ystart quantities given.) The xsize argument is how many pixels to composite out to the right of this corner. The ysize argument is how many pixels to composite downward from this corner. Quantities are specified in pixels.

Alpha Files

-inalphafile

Allows you to give the name of an external file whose alpha channel shall be used in lieu of the alpha channel of the file specified with -infile. This option is particularly useful in conjunction with the -inalphamap option. Here are some ways in which these two options could be used:

Alias (on Silicon Graphics computers) generates files known as "matte" files. These files are a variant of the Alias 'pix' format. They are simply greyscale files whose values correspond to the alpha channel of a corresponding RGB pix file. If you have, say, an rgb file named dinosaur.pix and a matte file named dinomatte.pix, and you wanted to put your dinosaur against a background of swamps (swamps.pix), then you would type:

imcomp dinasaur.pix -inalphafile dinomatte.pix swamps.pix -inalphamap alpha=grey output.rgb

To use the red channel of red.rgb as the alpha channel for car.rgb, while compositing the car against a freeway (freeway.rgb), you would type:

imcomp car.rgb freeway.rgb -inalphafile red.rgb -inalphamap alpha=red out.rgb

NOTES

Compositing historically originated in the analog domain where it was (and still is) used for combining separate film footage into a single composite result. This is used for special effects that can place an alien spaceship in the sky above a sleepy suburban neighborhood, or a giant dinosaur on the streets of New York. It is also used for more subtle effects like stars in a night sky (stars don't film well, so they are usually faked using compositing special effects).

In film, two rolls of film are processed to produce *mattes*. If the matte changes from frame to frame, such as for a moving object, then it's called a *traveling matte*. In either case the matte is a black and white coverage mask that delimits the area of interest of each source film footage.

For instance, to place a starship in front of a planet takes several steps:

1. Film the starship model in front of a solid blue screen (or any color not in the model itself). Move the camera or model around to simulate flying acrobatics.
2. Film a model or painting of a planet, complete with star-filled sky.
3. Create the first traveling matte of the starship model footage. Everywhere there's a blue background in the starship footage, make the matte film black. Everywhere else (where the starship is) make it white. If you watched this film you'd just see a flat white blobby silhouette of the spaceship flying about on a black background.
4. Create the second traveling matte by doing the same thing, but make the blue background area white, and the spaceship area black. If you watched this film you'd see a flat black blobby silhouette of the spaceship flying about on a white background.
5. Run the first matte (white where the starship is) and the raw starship footage through a projector back-to-back. Light from the projector bulb will only make it through the combined matte and starship film where it is light in both pieces of film. Because the matte only lets light through in areas the starship is, only the starship shows up on the projection. The blue background (and model stand, strings, stray cables, floor, and other stage clutter) of the raw starship footage doesn't show up. Expose a new roll of film with this starship-only result.
6. Do the same kind of thing but this time use the second matte (white where the starship isn't) and the planet footage. When run through the projector, light will only make it through the planet film and the second matte for the areas the starship isn't covering. Double-expose the same roll of film from step 5 with this planet-only result. You now have one piece of film with the starship and the planet, and neither overlaps the other.

If we didn't go through all this hassle and just double-exposed one roll of film with both the space ship and the planet, we'd see the planet through the darker parts of the spaceship and it would look weird. If you watch old cheesy science fiction movies you see this kind of effect a lot. It's cheap, but not very realistic. To get realism you have to do compositing using the steps above.

Obviously the above procedure is pretty tedious. If you want two separately filmed spaceships in the scene, then you need to go through another compositing stage. Three spaceships and you've got yet another stage. And so on for each additional independently filmed scene element. The Star Wars epic, for instance, went to extraordinary levels of compositing to get all those independently moving Tie-fighters, X-wing fighters, Y-wing fighters, the Death Star, and the Millennium Falcon all into one scene.

The difficulty with film compositing is that it experiences generation loss. Each time footage is transferred to a new roll of film (steps 5 and 6 of the procedure above), the footage gets grainier and less crisp. The darks get lighter and the image quality goes down a bit. In a complex compositing nightmare like a Star Wars film, the number of compositing passes is huge and this generation loss becomes quite noticeable. The next time you watch a Star Wars film, watch for differences in scene quality from cut to cut during a frantic battle. Scenes with fewer ships in them will look crisper and the darks darker.

It is this generation loss problem that prompted George Lucas to have PIXAR develop its original PIXAR Image Computer. The PIXAR system was used to do digital compositing, much like imcomp does. Raw footage was scanned in, frame by frame, and matte's generated digitally. The mattes and footage could then be combined any number of times, digitally, without any generation loss. The final result is then recorded back to film just as crisp and clean as the original footage. It is this type of system that Disney uses today for combining cell animation and computer animation.

imcomp's compositing operations perform this same kind of digital compositing. -inside and -outside may be used to generate mattes. However, more commonly, -over is used to accomplish in one step the full 6-step compositing operation outlined earlier.

imcomp allows users to composite individual image fields, and to composite in the HSI color space instead of just the RGB color space. The red pixels in the first image can be composited on to the red channel of the second to generate a third image. Blue and green pixels are unchanged. Or, the saturation values of one image can be composited over a second image's colors to create an interesting form of double exposure.

Because of the extra per-pixel operation formula evaluation, imcomp can be somewhat slow on large images. impaste(1IM) is a stripped down version of the compositor that simply pastes the entire first input image atop the second image, without regard to an alpha channel. Without the additional formula evaluation per pixel, impaste(1IM) is considerably faster.

Alpha channels may be added to images that don't already have them by using tools like imfill(1IM) and imadjust(1IM). imfill(1IM) will create a solid or ramped alpha channel useful for special effects. imadjust(1IM) will create alpha channels whose values vary based upon selection criteria, such as "Set alpha to 0 (transparent) for all black pixels." Additionally, imkey(1IM) will automatically add an alpha channel to an image for you while compositing. That is, Bimkey will let you composite based on fields other than the alpha field. For more complex alpha channel work, paint systems are more appropriate.

Color index compositing is rarely a good thing to do. Compositing operations apply math operations to calculate new color index values. Those new values will point into the output image's color table, cloned from the second input image. It is unlikely that those new index values will point to colors that look right. The operation may even create illegal color indexes that point to CLT slots beyond the end of the CLT. Beware color index compositing! It usually makes more sense to composite in RGB space instead.

For notes regarding file format conversion and standard image tool options, see the man page on imconv(1IM).

Error messages are reported to stderr.

EXAMPLES

Composite one Sun Rasterfile image *over* another:

```
imcomp inone.ras intwo.ras out.rgb
```

Do the same operation, but only composite the red field:

```
imcomp inone.ras intwo.ras -red out.ras
```

Composite using the saturation and hue fields and the *outside* operation:

```
imcomp inone.ras intwo.ras -saturation -hue -outside out.ras
```

Composite, but move the foreground image right by 20 pixels and down by 50:

```
imcomp inone.ras intwo.ras -xposition 20 -yposition 50 out.ras
```

Composite only using a rectangle sub-region of the foreground. The left hand edge of the foreground composite will start at 10 pixels into the left edge of foreground and will be 30 pixels wide and 40 pixels high against the background:

```
imcomp inone.ras intwo.ras -xstart 10 -xsize 30 -ysize 40 out.ras
```

TUTORIAL EXAMPLES

The different effects obtained with the four compositing algorithms can sometimes be confusing. The following examples are step-by-step procedures for creating a set of test cases that can be displayed to gain greater insight into compositing trickery.

For these examples, two sample input images are created using imfill(1IM):

Image One: stripes.ras

The following steps create a 100 x 100 RGB Sun Rasterfile image with a red background and three purple stripes. The background has an alpha value of 0 (transparent) and the stripes an alpha value of 255 (opaque):

```
imfill -xsize 100 -ysize 100 -red 200 -alpha 0 stripes.ras
imfill -ypos 10 -ysize 20 -blue 200 -alpha 255 stripes.ras stripes.ras
imfill -ypos 45 -ysize 10 -blue 200 -alpha 255 stripes.ras stripes.ras
imfill -ypos 70 -ysize 20 -blue 200 -alpha 255 stripes.ras stripes.ras
```

Image Two: cee.ras

The following steps create a 100 x 100 RGB Sun Rasterfile image with a green background and a yellow letter C. The background has an alpha value of 0 (transparent), and the C an alpha value of 255 (opaque):

```
imfill -xsize 100 -ysize 100 -green 200 -alpha 0 cee.ras
imfill -xpos 20 -xsize 10 -ypos 10 -ysize 80 -alpha 255 -red 200 cee.ras cee.ras
imfill -xpos 30 -xsize 50 -ypos 10 -ysize 10 -alpha 255 -red 200 cee.ras cee.ras
imfill -xpos 30 -xsize 50 -ypos 80 -ysize 10 -alpha 255 -red 200 cee.ras cee.ras
```

cee.ras over stripes.ras

This test case may be generated using:

```
imcomp cee.ras stripes.ras -over over.ras
```

The resulting image places the opaque portions of cee.ras over pixels of stripes.ras. Since the opaque portions of cee.ras are those pixels that are a part of the yellow C, the resulting image *over.ras* has a yellow C on a background of red with purple stripes.

Pixels involved in this compositing operation fell into two categories:

1. When pixels in cee.ras had alpha values of 255 (opaque), they were copied directly to the output image.
2. All other output image pixels were filled in with pixels from stripes.ras.

stripes.ras over cee.ras

This test case may be generated using:

```
imcomp stripes.ras cee.ras -over over.ras
```

The resulting image places the opaque portions of stripes.ras over pixels of cee.ras. Since the opaque portions of stripes.ras are those pixels that make up the purple stripes, the resulting image *over.ras* has purple stripes overlapping the green background and yellow C of cee.ras.

Pixels involved in this compositing operation fell into two categories:

1. When pixels in stripes.ras had alpha values of 255 (opaque), they were copied directly to the output image.
2. All other output image pixels were filled in with pixels from cee.ras.

cee.ras atop stripes.ras

This test case may be generated using:

```
imcomp cee.ras stripes.ras -atop atop.ras
```

The resulting image looks pretty weird. The image shows the red background from stripes.ras as red stripes on a white background. A yellow C hides behind the stripes. Where the stripes and the C intersect, the color is black.

Recall the *atop* compositing operation formula:

```
OutField = In1Field * In2Alpha + In2Field * (1.0 - In1Alpha)
```

Pixels involved in this compositing operation fell into four categories:

1. When cee.ras (image one) alphas are 0 (transparent) and stripes.ras (image two) alphas are 0 (transparent), the formula evaluates to using stripes.ras's colors. cee.ras has 0 alpha for the green background, and stripes.ras has 0 alpha for the red background between the purple stripes. Combined we get red stripes where the green and red backgrounds overlap.
2. When cee.ras alphas are 255 (opaque) and stripes.ras alphas are 255 (opaque), the formula evaluates to using cee.ras's colors. cee.ras has 255 alpha for the yellow C, and stripes.ras has 255 alpha for the purple stripes. Combined, we get portions of a yellow C everywhere the purple stripes and the C overlap.
3. When cee.ras alpha is 255 (opaque), but stripes.ras is 0 (transparent), the formula evaluates to 0, making these portions black.
4. When stripes.ras alpha is 255 (opaque), but cee.ras is 0 (transparent), the formula includes both image's color components and sums the red, green, and blue values each to 200's, producing white.

stripes.ras atop cee.ras

This test case may be generated using:

```
imcomp stripes.ras cee.ras -atop atop.ras
```

The image shows the green background from cee.ras as green stripes on a black background. A purple C hides behind the stripes. Where the stripes and the C intersect, the color is a purple-yellow.

Pixels involved in this compositing operation fell into four categories:

1. When stripes.ras (image one) alphas are 0 (transparent) and cee.ras (image two) alphas are 0 (transparent), the formula evaluates to using cee.ras's colors. stripes.ras has 0 alpha for the red background, and cee.ras has 0 alpha for the green background. Combined we get green stripes where the green and red backgrounds overlap.
2. When stripes.ras alphas are 255 (opaque) and cee.ras alphas are 255 (opaque), the formula evaluates to using stripes.ras's colors. stripes.ras has 255 alpha for the purple stripes, and cee.ras has 255 alpha for the yellow C. Combined, we get portions of the purple stripes everywhere the purple stripes and the C overlap.
3. When stripes.ras alpha is 255 (opaque), but cee.ras is 0 (transparent), the formula evaluates to 0, making these portions black.
4. When cee.ras alpha is 255 (opaque), but stripes.ras is 0 (transparent), the formula includes both image's color components and sums each of the red, green, and blue values to produce a purple-yellow-ish color.

cee.ras inside stripes.ras

This test case may be generated using:

```
imcomp cee.ras stripes.ras -inside inside.ras
```

The resulting image shows the yellow C and green background of cee.ras behind a set of black stripes.

Pixels involved in this compositing operation fell into two categories:

1. When pixels in stripes.ras have alpha values of 255, pixels from cee.ras are copied to the output image.
2. When pixels in stripes.ras have alpha values of 0, the output pixel value is zero (black).

stripes.ras inside cee.ras

This test case may be generated using:

```
imcomp stripes.ras cee.ras -inside inside.ras
```

The resulting image shows a black background with a C-shaped stencil cut-out inside of which we see the red and purple stripes of stripes.ras.

Pixels involved in this compositing operation fell into two categories:

1. When pixels in cee.ras have alpha values of 255, pixels from stripes.ras are copied to the output image.
2. When pixels in cee.ras have alpha values of 0, the output pixel value is zero (black).

cee.ras outside stripes.ras

This test case may be generated using:

```
imcomp cee.ras stripes.ras -outside outside.ras
```

The resulting image shows black stripes obscuring the green and yellow C image of cee.ras. The image is a reversal of the *inside* image created using imcomp cee.ras stripes.ras -inside inside.ras. Black stripes in the former show color in the later, and black stripes in the later show color in the former. If these two images were combined, stripe by stripe, the result would be cee.ras.

Pixels involved in this compositing operation fell into two categories:

1. When pixels in stripes.ras have alpha values of 0, pixels from cee.ras are copied to the output image.
2. When pixels in stripes.ras have alpha values of 255, the output pixel value is zero (black).

stripes.ras outside cee.ras

This test case may be generated using:

```
imcomp stripes.ras cee.ras -outside outside.ras
```

The resulting image shows a black C atop the red and purple stripes of stripes.ras. The image should look like a reversal of the *inside* image created using imcomp stripes.ras cee.ras -inside inside.ras. The C in the former shows color, while the C in the later is black. If these two images were combined the result would be stripes.ras.

Pixels involved in this compositing operation fell into two categories:

1. When pixels in cee.ras have alpha values of 0, pixels from stripes.ras are copied to the output image.
2. When pixels in cee.ras have alpha values of 255, the output pixel value is zero (black).

SEE ALSO

imadjust (1IM), imdissolve (1IM), imfill (1IM), ImVfbComp (3IM)

For information on SDSC's image library, see imintro(3IM).

AUTHORS

Chris Groening, Dave Nadeau, Shane Cooper, Brian Duggan
San Diego Supercomputer Center

See the individual file format man pages for the authors of the underlying format read and write code. The names of these man pages begin with the letters "im" followed by the format name. For example, the name of the TIFF man page is imtiff. To display it, enter man imtiff.

CONTACT

SDSC consultants, (619)534-5100, consult@y1.sdsc.edu