Compositing Digital Images

Thomas Porter
Tom Duff †

Computer Graphics Project
Lucasfilm Ltd.

ABSTRACT

Most computer graphics pictures have been computed all at once, so that the rendering program takes care of all computations relating to the overlap of objects. There are several applications. however, where elements must be rendered separately, relying on compositing techniques for the anti-aliased accumulation of the full image. This paper presents the case for four-channel pictures, demonstrating that a matte component can be computed similarly to the color channels. The paper discusses guidelines for the generation of elements and the arithmetic for their arbitrary compositing.

CR Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generations — Display algorithms; I.3.4 [**Computer Graphics**]: Graphics Utilities — Software support; I.4.1 [**Image Processing**]: Digitization — Sampling.

General Terms: Algorithms

Additional Key Words and Phrases: compositing, matte channel, matte algebra, visible surface algorithms, graphics systems

† Author's current address: AT&T Bell Laboratories, Murray Hill, NJ 07974, Room 2C465

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1984 ACM 0-89791-138-5/84/007/0253 $00.75

1. Introduction

Increasingly, we find that a complex three dimensional scene cannot be fully rendered by a single program. The wealth of literature on rendering polygons and curved surfaces, handling the special cases of fractals and spheres and quadrics and triangles, implementing refinements for texture mapping and bump mapping, noting speed-ups on the basis of coherence or depth complexity in the scene, suggests that multiple programs are necessary.

In fact, reliance on a single program for rendering an entire scene is a poor strategy for minimizing the cost of small modeling errors. Experience has taught us to break down large bodies of source code into separate modules in order to save compilation time. An error in one routine forces only the recompilation of its module and the relatively quick reloading of the entire program. Similarly, small errors in coloration or design in one object should not force the "recompilation" of an entire image.

Separating the image into *elements* which can be independently rendered saves enormous time. Each element has an associated *matte,* coverage information which designates the shape of the element. The *compositing* of those elements makes use of the mattes to accumulate the final image.

The compositing methodology must not induce aliasing in the image; soft edges of the elements must be honored in computing the final image. Features should be provided to exploit the full associativity of the compositing process; this affords flexibility, for example, for the accumulation of several foreground elements into an aggregate foreground which can be examined over different backgrounds. The compositor should provide facilities for arbitrary dissolves and fades of elements during an animated sequence.

Several highly successful rendering algorithms have worked by reducing their environments to pieces that can be combined in a 2 1/2 dimensional manner, and then overlaying them either front-to-back or back-to-front [3]. Whitted and Weimar's graphics test-bed [6] and Crow's image generation environment [2] are both designed to deal with heterogenously rendered elements. Whitted

GOOGLE EXHIBIT 1025
GOOGLE v. 138 EAST
IPR2024-00958

and Weimar's system reduces all objects to horizontal spans which are composited using a Warnock-like algorithm. In Crow's system a supervisory process decides the order in which to combine images created by independent special-purpose rendering processes. The imaging system of Warnock and Wyatt [5] incorporates 1-bit mattes. The Hanna-Barbera cartoon animation system [4] incorporates soft-edge mattes, representing the opacity information in a less convenient manner than that proposed here. The present paper presents guidelines for rendering elements and introduces the algebra for compositing.

2. The Alpha Channel

A separate component is needed to retain the matte information, the extent of coverage of an element at a pixel. In a full color rendering of an element, the RGB components retain only the color. In order to place the element over an arbitrary background, a mixing factor is required at every pixel to control the linear interpolation of foreground and background colors. In general, there is no way to encode this component as part of the color information. For anti-aliasing purposes, this mixing factor needs to be of comparable resolution to the color channels. Let us call this an *alpha* channel, and let us treat an alpha of 0 to indicate no coverage, 1 to mean full coverage, with fractions corresponding to partial coverage.

In an environment where the compositing of elements is required, we see the need for an alpha channel as an integral part of all pictures. Because mattes are naturally computed along with the picture, a separate alpha component in the frame buffer is appropriate. Off-line storage of alpha information along with color works conveniently into run-length encoding schemes because the alpha information tends to abide by the same runs.

What is the meaning of the quadruple (r,g,b,α) at a pixel? How do we express that a pixel is half covered by a full red object? One obvious suggestion is to assign (1,0,0,.5) to that pixel: the .5 indicates the coverage and the (1,0,0) is the color. There are a few reasons to dismiss this proposal, the most severe being that all compositing operations will involve multiplying the 1 in the red channel by the .5 in the alpha channel to compute the red contribution of this object at this pixel. The desire to avoid this multiplication points up a better solution, storing the *pre-multiplied* value in the color component, so that (.5,0,0,.5) will indicate a full red object half covering a pixel.

The quadruple (r,g,b,α) indicates that the pixel is α covered by the color (r/α, g/α, b/α). A quadruple where the alpha component is less than a color component indicates a color outside the [0,1] interval, which is somewhat unusual. We will see later that luminescent objects can be usefully represented in this way. For the representation of normal objects, an alpha of 0 at a pixel generally forces the color components to be 0. Thus the RGB channels record the true colors where alpha is 1, linearly darkened colors for fractional alphas along edges, and black where alpha is 0. Silhouette edges of RGBA elements thus exhibit their anti-aliased nature when viewed on an RGB monitor.

It is important to distinguish between two key pixel representations:

black = (0,0,0,1);
clear = (0,0,0,0).

The former pixel is an opaque black; the latter pixel is transparent.

3. RGBA Pictures

If we survey the variety of elements which contribute to a complex animation, we find many complete background images which have an alpha of 1 everywhere. Among foreground elements, we find that the color components roll off in step with the alpha channel, leaving large areas of transparency. Mattes, colorless stencils used for controlling the compositing of other elements, have 0 in their RGB components. Off-line storage of RGBA pictures should therefore provide the natural data compression for handling the RGB pixels of backgrounds, RGBA pixels of foregrounds, and A pixels of mattes.

There are some objections to computing with these RGBA pictures. Storage of the color components premultiplied by the alpha would seem to unduly quantize the color resolution, especially as alpha approaches 0. However, because any compositing of the picture will require that multiplication anyway, storage of the product forces only a very minor loss of precision in this regard. Color extraction, to compute in a different color space for example, becomes more difficult. We must recover (r/α, g/α, b/α), and once again, as alpha approaches 0, the precision falls off sharply. For our applications, this has yet to affect us.

4. The Algebra of Compositing

Given this standard of RGBA pictures, let us examine how compositing works. We shall do this by enumerating the complete set of binary compositing operations. For each of these, we shall present a formula for computing the contribution of each of two input pictures to the output composite at each pixel. We shall pay particular attention to the output pixels, to see that they remain pre-multiplied by their alpha.

4.1. Assumptions

When blending pictures together, we do not have information about overlap of coverage information within a pixel; all we have is an alpha value. When we consider the mixing of two pictures at a pixel, we must make some assumption about the interplay of the two alpha values. In order to examine that interplay, let us first consider the overlap of two semi-transparent elements like haze, then consider the overlap of two opaque, hard-edged elements.

If α_A and α_B represent the opaqueness of semi-transparent objects which fully cover the pixel, the computation is well known. Each object lets $(1-\alpha)$ of the background through, so that the background shows through only $(1-\alpha_A)(1-\alpha_B)$ of the pixel. $\alpha_A(1-\alpha_B)$ of the background is blocked by object A and passed by object B; $(1-\alpha_A)\alpha_B$ of the background is passed by A and blocked by B. This leaves $\alpha_A\alpha_B$ of the pixel which we can consider to be blocked by both.

If α_A and α_B represent subpixel areas covered by opaque geometric objects, the overlap of objects within the pixel is quite arbitrary. We know that object A divides the pixel into two subpixel areas of ratio α_A:$1-\alpha_A$. We know that object B divides the pixel into two subpixel areas of ratio α_B:$1-\alpha_B$. Lacking further information, we make the following assumption: *there is nothing special about the shape of the pixel; we expect that object B will divide each of the subpixel areas inside and outside of object A into the same ratio* α_B:$1-\alpha_B$. The result of the assumption is the same arithmetic as with semi-transparent objects and is summarized in the following table:

description	*area*
$\bar{A}\cap\bar{B}$	$(1-\alpha_A)(1-\alpha_B)$
$A\cap\bar{B}$	$\alpha_A(1-\alpha_B)$
$\bar{A}\cap B$	$(1-\alpha_A)\alpha_B$
$A\cap B$	$\alpha_A\alpha_B$

The assumption is quite good for most mattes, though it can be improved if we know that the coverage seldom overlaps (adjacent segments of a continuous line) or always overlaps (repeated application of a picture). For ease in presentation throughout this paper, let us make this assumption and consider the alpha values as representing subpixel coverage of opaque objects.

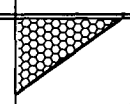
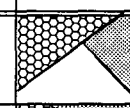
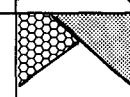
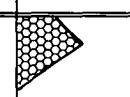
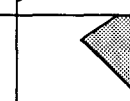
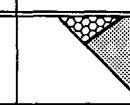
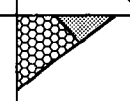
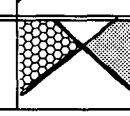
4.2. Compositing Operators

Consider two pictures A and B. They divide each pixel into the 4 subpixel areas

B	A	*name*	*description*	*choices*
0	0	0	$\bar{A}\cap\bar{B}$	0
0	1	A	$A\cap\bar{B}$	0, A
1	0	B	$\bar{A}\cap B$	0, B
1	1	AB	$A\cap B$	0, A, B

listed in this table along with the choices in each area for contributing to the composite. In the last area, for example, because both input pictures exist there, either could survive to the composite. Alternatively, the composite could be clear in that area.

A particular binary compositing operation can be identified as a quadruple indicating the input picture which contributes to the composite in each of the four subpixel areas 0, A, B, AB of the table above. With three choices where the pictures intersect, two where only one picture exists and one outside the two pictures, there are $3\times2\times2\times1=12$ distinct compositing operations listed in the table below. Note that pictures A and B are diagrammed as covering the pixel with triangular wedges whose overlap conforms to the assumption above.

operation	quadruple	diagram	F_A	F_B
clear	(0,0,0,0)		0	0
A	(0,A,0,A)		1	0
B	(0,0,B,B)		0	1
A **over** *B*	(0,A,B,A)		1	$1-\alpha_A$
B **over** *A*	(0,A,B,B)		$1-\alpha_B$	1
A **in** *B*	(0,0,0,A)		α_B	0
B **in** *A*	(0,0,0,B)		0	α_A
A **out** *B*	(0,A,0,0)		$1-\alpha_B$	0
B **out** *A*	(0,0,B,0)		0	$1-\alpha_A$
A **atop** *B*	(0,0,B,A)		α_B	$1-\alpha_A$
B **atop** *A*	(0,A,0,B)		$1-\alpha_B$	α_A
A **xor** *B*	(0,A,B,0)		$1-\alpha_B$	$1-\alpha_A$

Useful operators include *A* **over** *B*, *A* **in** *B*, and *A* **held out by** *B*. *A* **over** *B* is the placement of foreground A in front of background B. *A* **in** *B* refers only to that part of A inside picture B. *A* **held out by** *B*, normally shortened to *A* **out** *B*, refers only to that part of A outside picture B. For completeness, we include the less useful operators *A* **atop** *B* and *A* **xor** *B*. *A* **atop** *B* is the union of *A* **in** *B* and *B* **out** *A*. Thus, *paper* **atop** *table* includes *paper* where it is on top of *table*, and *table* otherwise; area beyond the edge of the table is out of the picture. *A* **xor** *B* is the union of *A* **out** *B* and *B* **out** *A*.

4.3. Compositing Arithmetic

For each of the compositing operations, we would like to compute the contribution of each input picture at each pixel. This is quite easily solved by recognizing that each input picture survives in the composite pixel only within its own matte. For each input picture, we are looking for that fraction of its own matte which prevails in the output. By definition then, the alpha value of the composite, the total area of the pixel covered, can be computed by adding α_A times its fraction F_A to α_B times its fraction F_B.

The color of the composite can be computed on a component basis by adding the color of the picture A times its fraction to the color of picture B times its fraction. To see this, let c_A, c_B, and c_O be some color component of pictures A, B and the composite, and let C_A, C_B, and C_O be the true color component before pre-multiplication by alpha. Then we have

$$c_O = \alpha_O C_O$$

Now C_O can be computed by averaging contributions made by C_A and C_B, so

$$c_O = \alpha_O \frac{\alpha_A F_A C_A + \alpha_B F_B C_B}{\alpha_A F_A + \alpha_B F_B}$$

but the denominator is just α_O, so

$$\begin{aligned} c_O &= \alpha_A F_A C_A + \alpha_B F_B C_B \\ &= \alpha_A F_A \frac{c_A}{\alpha_A} + \alpha_B F_B \frac{c_B}{\alpha_B} \\ &= c_A F_A + c_B F_B \qquad (1) \end{aligned}$$

Because each of the input colors is pre-multiplied by its alpha, and we are adding contributions from non-overlapping areas, the sum will be effectively pre-multiplied by the alpha value of the composite just computed. The pleasant result that the color channels are handled with the same computation as alpha can be traced back to our decision to store pre-multiplied RGBA quadruples. Thus the problem is reduced to finding a table of fractions F_A and F_B which indicate the extent of contribution of A and B, plugging these values into equation 1 for both the color and the alpha components.

By our assumptions above, the fractions are quickly determined by examining the pixel diagram included in the table of operations. Those fractions are listed in the F_A and F_B columns of the table. For example, in the *A* **over** *B* case, picture A survives everywhere while picture B survives only outside picture A, so the corresponding fractions are 1 and $(1-\alpha_A)$. Substituting into equation 1, we find

$$c_O = c_A \times 1 + c_B \times (1-\alpha_A).$$

This is almost the well used linear interpolation of foreground F with background B

$$B' = F \times \alpha + B \times (1-\alpha),$$

except that our foreground is pre-multiplied by alpha.

4.4. Unary operators

To assist us in dissolving and in balancing color brightness of elements contributing to a composite, it is useful to introduce a darken factor ϕ and a dissolve factor δ:

$$\mathbf{darken}(A,\phi) \equiv (\phi r_A, \phi g_A, \phi b_A, \alpha_A)$$
$$\mathbf{dissolve}(A,\delta) \equiv (\delta r_A, \delta g_A, \delta b_A, \delta \alpha_A)\ .$$

Normally, $0 \leq \phi, \delta \leq 1$ although none of the theory requires it.

As ϕ varies from 1 to 0, the element will change from normal to complete blackness. If $\phi > 1$ the element will be brightened. As δ goes from 1 to 0 the element will gradually fade from view.

Luminescent objects, which add color information without obscuring the background, can be handled with the introduction of a opaqueness factor ω, $0 \leq \omega \leq 1$:

$$\mathbf{opaque}(A,\omega) \equiv (r_A, g_A, b_A, \omega \alpha_A)\ .$$

As ω varies from 1 to 0, the element will change from normal coverage over the background to no obscuration. This scaling of the alpha channel alone will cause pixel quadruples where α is less than a color component, indicating a representation of a color outside of the normal range. This possibility forces us to clip the output composite to the [0,1] range.

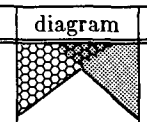
An ω of 0 will produce quadruples $(r,g,b,0)$ which do have meaning. The color channels, pre-multiplied by the original alpha, can be plugged into equation 1 as always. The alpha channel of 0 indicates that this pixel will obscure nothing. In terms of our methodology for examining subpixel areas, we should understand that using the **opaque** operator corresponds to shrinking the matte coverage with regard to the color coverage.

4.5. The PLUS operator

We find it useful to include one further binary compositing operator **plus** . The expression *A* **plus** *B* holds no notion of precedence in any area covered by both pictures; the components are simply added. This allows us to dissolve from one picture to another by specifying

$$\mathbf{dissolve}(A,\alpha)\ \mathbf{plus}\ \mathbf{dissolve}(B,1-\alpha).$$

In terms of the binary operators above, **plus** allows both pictures to survive in the subpixel area AB. The operator table above should be appended:

operation	diagram	F_A	F_B
(0,A,B,AB) *A* **plus** *B*		1	1

5. Examples

The operations on one and two pictures are presented as a basis for handling compositing expressions involving several pictures. A normal case involving three pictures is the compositing of a foreground picture A over a background picture B, with regard to an independent matte C. The expression for this compositing operation is

$$(A \textbf{ in } C) \textbf{ over } B.$$

Using equation 1 twice, we find that the composite in this case is computed at each pixel by

$$c_O = c_A\alpha_C + c_B(1-\alpha_A\alpha_C).$$

As an example of a complex compositing expression, let us consider a subwindow of Rob Cook's picture *Road to Point Reyes* [1]. This still frame was assembled from many elements according to the following rules:

Foreground = *FrgdGrass* **over** *Rock* **over** *Fence* **over** *Shadow* **over** *BkgdGrass*;

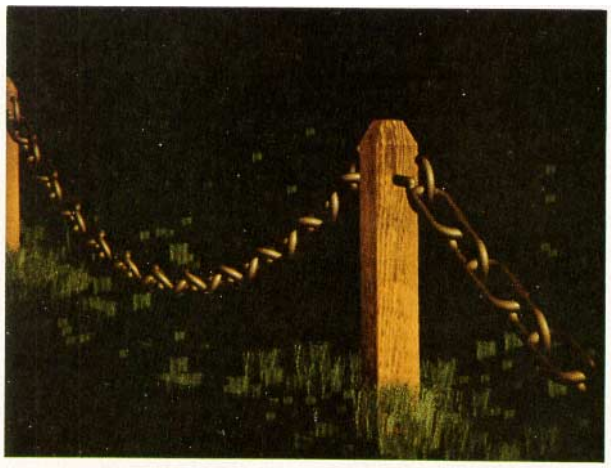
GlossyRoad = *Puddle* **over** (*PostReflection* **atop** (*PlantReflection* **atop** *Road*));

Hillside = *Plant* **over** *GlossyRoad* **over** *Hill*;

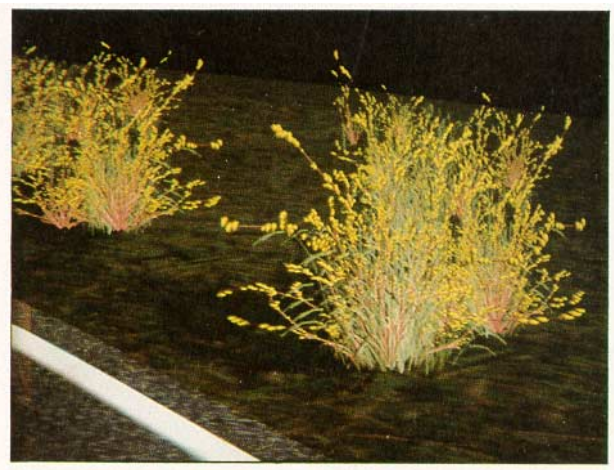
Background = *Rainbow* **plus** *Darkbow* **over** *Mountains* **over** *Sky*;

Pt.Reyes = *Foreground* **over** *Hillside* **over** *Background*.

Figure 1 shows three intermediate composites and the final picture.

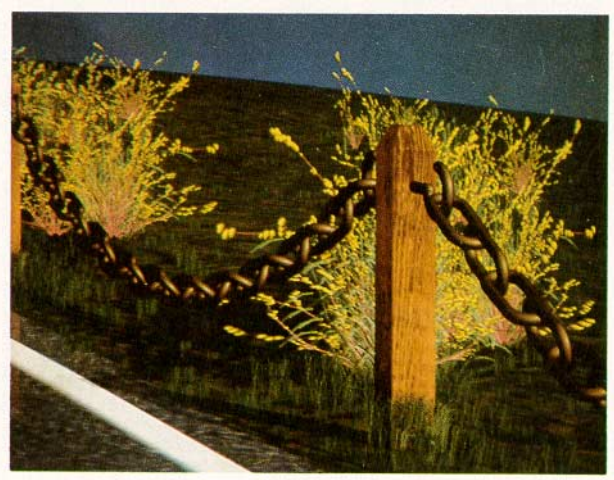


Foreground = *FrgdGrass* **over** *Rock* **over** *Fence* **over** *Shadow* **over** *BkgdGrass*;



Hillside = *Plant* **over** *GlossyRoad* **over** *Hill*;

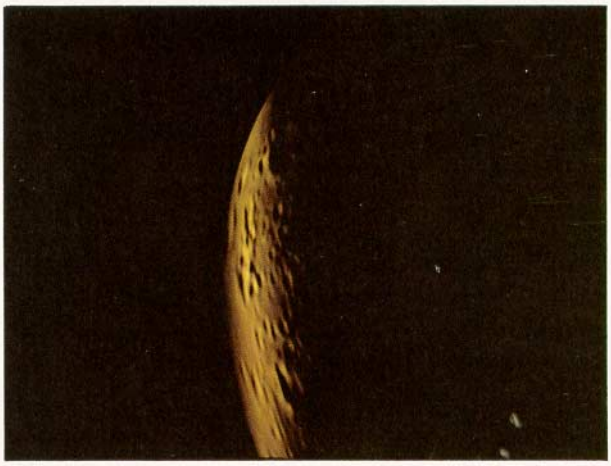
Background = *Rainbow* **plus** *Darkbow* **over** *Mountains* **over** *Sky*;



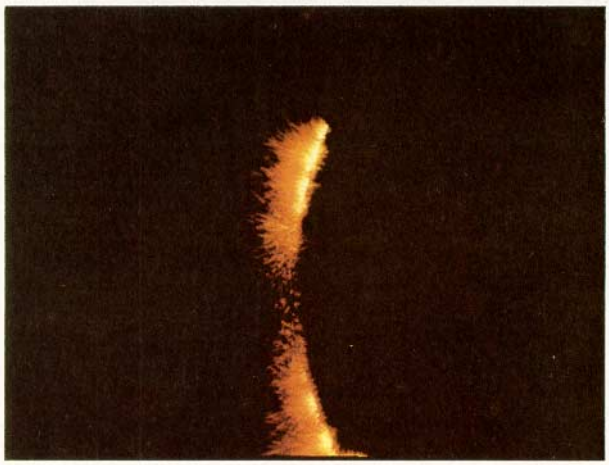
Pt.Reyes = *Foreground* **over** *Hillside* **over** *Background*.

Figure 1

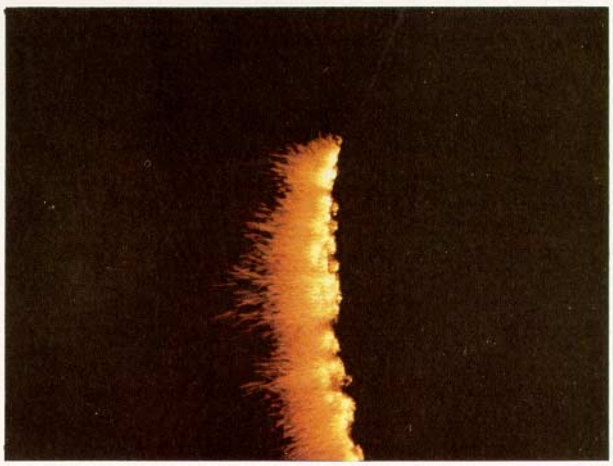
Stars



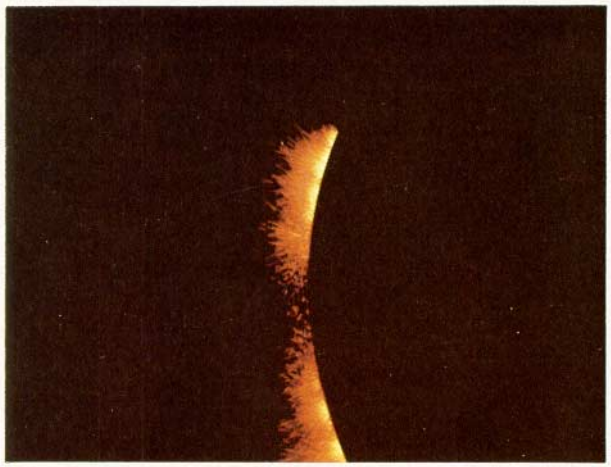
Planet



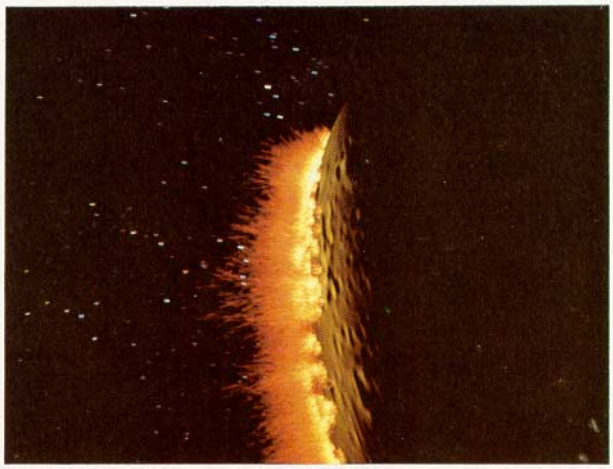
BFire



FFire



BFire **out** Planet



Composite

Figure 2

A further example demonstrates the problem of *correlated mattes.* In Figure 2, we have a star field background, a planet element, fiery particles behind the planet, and fiery particles in front of the planet. We wish to add the luminous fires, obscure the planet, darkened for proper balance, with the aggregate fire matte, and place that over the star field. An expression for this compositing is

(*FFire* **plus** (*BFire* **out** *Planet*))
over darken(*Planet*,.8) **over** *Stars* .

We must remember that our basic assumption about the division of subpixel areas by geometric objects breaks down in the face of input pictures with correlated mattes. When one picture appears twice in a compositing expression, we must take care with our computations of F_A and F_B. Those listed in the table are correct only for uncorrelated pictures.

To solve the problem of correlated mattes, we must extend our methodology to handle n pictures: we must examine all 2^n subareas of the pixel, deciding which of the pictures survives in each area, and adding up all contributions. Multiple instances of a single picture or pictures with correlated mattes are resolved by aligning their pixel coverage. Example 2 can be computed by building a table of survivors (shown below) to accumulate the extent to which each input picture survives in the composite.

FFire	BFire	Planet	Stars	Survivor
			•	Stars
		•		Planet
		•	•	Planet
	•			BFire
	•		•	BFire
	•	•		Planet
	•	•	•	Planet
•				FFire
•			•	FFire
•		•		FFire
•		•	•	FFire
•	•			FFire,BFire
•	•		•	FFire,BFire
•	•	•		FFire
•	•	•	•	FFire

6. Conclusion

We have pointed out the need for matte channels in synthetic pictures, suggesting that frame buffer hardware should offer this facility. We have seen the convenience of the RGBA scheme for integrating the matte channel. A language of operators has been presented for conveying a full range of compositing expressions. We have discussed a methodology for deciding compositing questions at the subpixel level, deriving a simple equation for handling all composites of two pictures. The methodology is extended to multiple pictures, and the language is embellished to handle darkening, attenuation, and opaqueness.

There are several problems to be resolved in related areas, which are open for future research. We are interested in methods for breaking arbitrary three dimensional scenes into elements separated in depth. Such elements are equivalent to clusters, which have been a subject of discussion since the earliest attempts at hidden surface elimination. We are interested in applying the compositing notions to Z-buffer algorithms, where depth information is retained at each pixel.

7. References

1. Cook, R. Road to Point Reyes. *Computer Graphics* Vol 17, No. 3 (1983), Title Page Picture.
2. Crow, F. C. A More Flexible Image Generation Environment. *Computer Graphics* Vol. 16, No. 3 (1982), pp. 9-18.
3. Newell, M. G., Newell, R. G., and Sancha, T. L.. A Solution to the Hidden Surface Problem, pp. 443-448. *Proceedings of the 1972 ACM National Conference.*
4. Wallace, Bruce. Merging and Transformation of Raster Images for Cartoon Animation. *Computer Graphics* Vol. 15, No. 3 (1981), pp. 253-262.
5. Warnock, John, and Wyatt, Douglas. A Device Independent Graphics Imaging Model for Use with Raster Devices. *Computer Graphics* Vol. 16, No. 3 (1982), pp. 313-319.
6. Whitted, Turner, and Weimer, David. A Software Test-Bed for the Development of 3-D Raster Graphics Systems. *Computer Graphics* Vol. 15, No. 3 (1981), pp. 271-277.

8. Acknowledgments

The use of mattes to control the compositing of pictures is not new. The graphics group at the New York Institute of Technology has been using this for years. NYIT color maps were designed to encode both color and matte information; that idea was extended in the Ampex AVA system for storing mattes with pictures. Credit should be given to Ed Catmull, Alvy Ray Smith, and Ikonas Graphics Systems for the existence of an alpha channel as an integral part of a frame buffer, which has paved the way for the developments presented in this paper.

The graphics group at Lucasfilm should be credited with providing a fine test bed for working out these ideas. Furthermore, certain ideas incorporated as part of this work have their origins as idle comments within this group. Thanks are also given to Rodney Stock for comments on an early draft which forced the authors to clarify the major assumptions.