# Post-rendering Composition for 3D Scenes

Cindy Grimm[†]

**Abstract**

*In traditional art a painter displays a 3D scene on a 2D image plane in a manner that is aesthetically pleasing. The arrangement of objects and colors is called* composition *and is the subject of many art books and classes. While a painter may use perspective to create depth in a scene they may also alter the perspective and color, either subtly or dramatically, to influence the focus of viewer and the effect of the image. To date, traditional 3D graphics packages have largely concentrated on modeling, textures, and lighting to create images and provide few tools for altering the composition post-rendering.*

*In this paper we present several simple techniques for creating images with non-standard perspective and color using standard 3D rendering packages. The scene is modeled in 3D but each object has its own camera, color balance, and image size, allowing the user to alter the composition* after *the 3D rendering step. The purpose of this paper is not to present a complete composition system but rather to illustrate the potential of composition-based tools.*

## 1. Introduction

Composition is the art of arranging objects and colors in a 2D image. There are no hard and fast rules for successful compositions, but nearly every art [1] or photography book [2] offers some guidelines on how to place objects and intensify or flatten-out colors. In photography this is achieved primarily through positioning the camera, changing filters, and possibly placing lights in the scene. Painters, however, have a great deal more flexibility when "rendering" a scene. They can shift objects on the image plane, elide or elucidate details, use a different perspective for each object, and alter the color and value. Figures 1 and 2 illustrate the types of changes an artist might introduce when painting a scene. To date, 3D rendering engines have taken the photography approach, letting talented lighting designers and camera manipulators produce a "good" composition by manipulating a 3D model.

In this paper we present some simple techniques for letting the user play with the image after it has been rendered. This combines the benefit of rendering packages, where perspective and lighting are "free", with image manipulation techniques. The three composition techniques we describe in this paper are camera angle, placement on the screen, and color adjustment. We show how to implement these techniques as a post-rendering process in OpenGL and Radiance [11], a radiance-based renderer.

We first describe previous work, then how to implement the post-rendering effects. We also provide some tools for maintaining image-space constraints such as "the bowl must always remain on the table". We close with a discussion on how these tools affect the original 3D rendering.

## 2. Previous Work

There exists work on choosing camera positions based on image-space constraints [3], and reverse engineering lighting based on desired light values in the scene [9108]. These techniques greatly help in manipulating the 3D scene data to produce a desired 2D image. The composition techniques presented here are complimentary to these since they are applied *after* the scene is rendered.

There are a variety of image processing techniques which can be applied to any 2D image to produce a more interesting, painterly rendered image, an idea first explored in [4]. These techniques do not use information from the 3D scene. Several papers on non-photorealistic rendering use the 2D image to determine how many and what strokes to draw [7 12] when rendering a 3D scene. Recently, this work has been ex-

[†] Dept. of Computer Science, Washington University in St. Louis

tended to allow composition effects to be specified with the scene, to control how and where strokes are placed [6].

It should be pointed out that the movie industry does post-rendering manipulations all the time, re-lighting scenes, adding reflections and objects into the scene, and combining graphics with film. (For a particularly good example of this I recommend the DVD of "Contact", which has detailed commentary on how several scenes were put together from disparate parts).

## 3. Overview

We use the 3D rendering systems to produce images which are then warped, color balanced, and composited together. Each independent object produces a single image with just the pixels from the object (all other pixels are masked out). To produce these images each of the objects is rendered using its own camera (Section 3.1) with the pixels not representing the object masked out. In systems supporting shadows and self-reflections the scene is rendered twice, once with all the objects present and once with just the selected object and the background. This second pass is needed to fill in any missing pixels which are behind other objects. In systems without secondary object-lighting interactions just the object needs to be rendered.

The rendered images are then color adjusted in HSV space (Section 3.2), screen space adjusted (Section 3.3) and composited together. To ensure proper compositing, the depth values of the scene from the scene camera, not the object's, are used (Section 3.1).

The composition data can be stored in one of two ways. If the scene has a key-framed camera path, the image transformation data is stored for each key frame. Alternatively, the data can be parametrized by camera location. For example, the camera for the table is specified to always be up and to the right of the camera for the scene. Another example might be the color of the object depends on where the object appears in the image.

### 3.1. Per-object camera

Each object has its own camera (which may be the same as the scene's camera). These cameras can be used to tilt objects up without changing their interreflections with other objects in the scene (as in the bowl in Figure 3) or to provide substantially different perspectives to maximize seeing the "interesting" parts of the scene. A third use is to define behavior such as having the vase always point towards the viewer (Figure 3). Some of these effects could be achieved by transforming the object in the scene, but this would change the object's light interaction with other elements in the scene.

During the rendering phase each object is first rendered with all the other objects in the scene. Only the pixels corresponding to the object are kept, with all other pixels masked out. Any object pixels which were obscured can be filled in by rendering the object again, this time without all the other objects present (or with only the background object). In addition to storing the image we also store the distance to the object as calculated from the *scene's* camera in the alpha channel. If $C_o$ is the object's camera and $C_s$ is the scene's camera we find, for each pixel $(x,y)$ the point $p$ on the object such that $C_o p = (x,y)$. The depth stored at $(x,y)$ is then $||C_s \rightarrow \text{from} - p||$. This ensures that the compositing step will correctly layer the objects as they would be layered when rendered with the scene's camera.

Some systems can return the first point or face encountered, simplifying the calculation of the point $p$. For our OpenGL system we do a ray intersection for each object pixel, which is unfortunately expensive.

To reduce image copying time we take the bounding box of the object and project the six corners onto the image. The bounding box in the image plane is then the minimum and maximum $x$ and $y$ coordinates of the six projected points. We only need to keep, or look at, pixels within the image plane bounding box.

### 3.2. Colors

Objects with brighter value or fuller saturation are much more noticeable in an image. Increasing and decreasing the variance in the value and saturation brings out or hides detail in the object. The user controls these effects indirectly through lighting in 3D systems; here we let the user adjust them after rendering.

We find the average $(H_a, S_a, V_a)$ of the object's image in HSV space. We then apply, for each pixel,

$$S'_{x,y} = (S_{x,y} - S_a)\sigma_o + S_a + S_o$$

$$V'_{x,y} = (V_{x,y} - V_a)\nu_o + V_a + V_o$$

where $S_a$ and $V_a$ are the saturation and value shift, and $\sigma_o$ and $\nu_o$ are the saturation and value scaling values for the object. By default $S_a = V_a = 0$ and $\sigma_o = \nu_o = 1$, which does not affect the image. This color shift is applied before the screen space adjustment.

### 3.3. Screen space adjustment

The position of the object on the image can be moved in the image plane $(v_x, v_y)$, in and out by $\Delta z$ (changes the depth value), rotated by $\theta$, and scaled by $(s_x, s_y)$. This transformation (except for the depth change) can be represented as a $3 \times 3$ matrix $T_i = C^{-1}TRSC$ where $C$ is the translation matrix that takes the center of the image plane bounding box to $(0,0)$. The $T$, $R$, and $S$ matrices are, in order, the translation, rotation, and scaling matrices. To compute the color and depth at a point $(x,y)$ in the final image we take $T_i^{-1}(x,y)$, being careful not to blend or use pixels which are masked out in the object's image. We then add $\Delta z$ to every depth value. The $\Delta z$ offset is useful for ensuring that an object stays in front of (or behind) another object even when tilted.

For objects that should be attached (for example, the bowl on the table) the screen space offset can be set automatically to ensure that the bottom of the bowl remains on the same spot on the table. In this case the value of $(v_x, v_y)$ is set automatically. We require that a point $p$ on object $i$ map to the point $q$ on object $j$ in image space. Let $C_i$ be the camera for object $i$ and $T_j$ be the image space transformation for object $j$. Then the translation vector should be:

$$v = T_j C_j q - C_i p$$

### 3.4. Compositing

Compositing is simply a matter of combining all of the object images, layering them by the depth stored in the alpha channel. Again, we can reduce compositing time by only copying pixels from the warped (by $T_i^{-1}$) image bounding box.

## 4. Discussion

The techniques presented here are fairly easy to implement but rather more difficult to control. A useful interface would allow the user to interactively adjust the parameters in the image plane. For the images and scenes in this paper computing an entire scene takes a few seconds on a 700MHz PC, most of which is spent pulling images off of the image buffer and performing object intersections. Since the user would only be altering a single object's parameters at a time, interactive rates are possible. Currently we can adjust the parameters using a dialog box and interactively change the screen space and color in real time for $300 \times 300$ images.

For scenes where the viewer specifies the camera (instead of a key-framed camera path), some sort of nearest neighbor with interpolation is needed to store the object's data, like the system in [6].

There are two places where this compositing can introduce errors. The first is when an object's camera position results in another object occluding the first object when this occlusion would not have happened with the original scene camera. In this case we fill in the missing pixels by rendering a second time with all auxiliary objects removed. The filled-in occluded part may therefore have incorrect secondary lighting effects. Note that with the Radiance ray tracer this is not a problem because we can cast the camera's rays from just above the object's surface, instead of from the camera's eye point, essentially "jumping" over the occluding object without having to remove it from the scene.

The second problem occurs when the object is moved in the image plane so that the shadow it was casting (or reflection) is no longer in the correct place or is the wrong size. Careful use of image space constraints (the bowl is stuck to the table) addresses some of these problems, but clearly there is room here for better constraints and more extensive image manipulation.

This set of techniques is only a subset of possible post-rendering manipulations; other common composition techniques include boundary enhancement and removal (which lets objects in the background become a single mass instead of clearly delineated images). Value and saturation matching across objects is useful for the same reason.

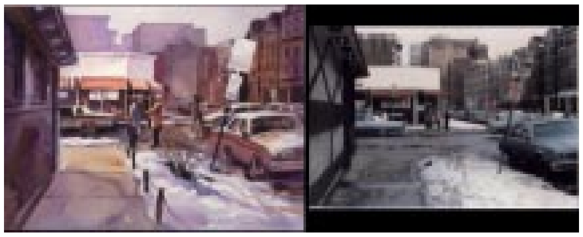
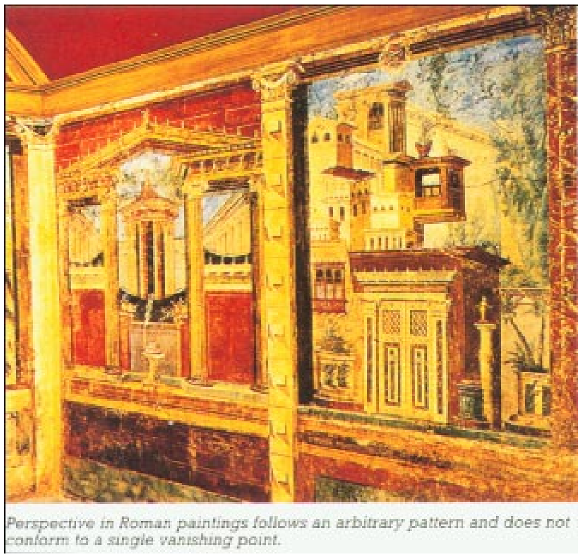


**Figure 1:** *A picture and painting of the same scene by David Becker (Copyright 1999)* [1]. *Notice the scale changes in the cars and shop, the removal of much of the detail on the signs and buildings, the increased lighting on the shop and the color shifts.*



Perspective in Roman paintings follows an arbitrary pattern and does not conform to a single vanishing point.

**Figure 2:** *An ancient Roman wall mural where the perspective on each building is random, Copyright 1999 Barron's* [5].

## Acknowledgements

I would like to thank Greg Ward for making Radiance publicly available.

## References

1. David Becker. *Watercolor Composition made easy*. North Light Books, 1999. 1, 3
2. Tom Grill and Mark Scanlon. *Photographic Composition*. American Photographic Book Publishing, 1990. 1
3. Michael Gleicher and Andrew Witkin. Through-the-lens camera control. *Computer Graphics (Proceedings of SIGGRAPH 92)*, 26(2):331–340, July 1992. ISBN 0-201-51585-7. Held in Chicago, Illinois. 1
4. Paul E. Haeberli. Paint by numbers: Abstract image representations. *Computer Graphics (Proceedings of SIGGRAPH 90)*, 24(4):207–214, August 1990. ISBN 0-201-50933-4. Held in Dallas, Texas. 1
5. Barron's Art Handbooks. *Perspective and Composition*. Barron's, 1999. 3
6. Michael Kowalski, John Hughes, Cynthia Rubin, and Jun Ohya. User-guided composition effects for art-based rendering. *ACMSymposium on Interactive 3D Graphics (to appear)*, 2001. 2, 3
7. Michael A. Kowalski, Lee Markosian, J. D. Northrup, Lubomir Bourdev, Ronen Barzel, Loring S. Holden, and John Hughes. Art-based rendering of fur, grass, and trees. *Proceedings of SIGGRAPH 99*, pages 433–438, August 1999. 1
8. Pierre Poulin and Alain Fournier. Lights from highlights and shadows. *1992 Symposium on Interactive 3D Graphics*, 25(2):31–38, March 1992. ISBN 0-89791-467-8. 1
9. Pierre Poulin and Alain Fournier. Painting surface characteristics. *Eurographics Rendering Workshop 1995*, pages 160–169, June 1995. Held in Dublin, Ireland. 1
10. Chris Schoeneman, Julie Dorsey, Brian Smits, James Arvo, and Donald Greenberg. Painting with light. *Proceedings of SIGGRAPH 93*, pages 143–146, August 1993. ISBN 0-201-58889-7. Held in Anaheim, California. 1
11. Gregory J. Ward. The radiance lighting simulation and rendering system. *Proceedings of SIGGRAPH 94*, pages 459–472, July 1994. ISBN 0-89791-667-0. Held in Orlando, Florida. 1
12. Georges Winkenbach and David H. Salesin. Computer-generated pen-and-ink illustration. *Proceedings of SIGGRAPH 94*, pages 91–100, July 1994. 1

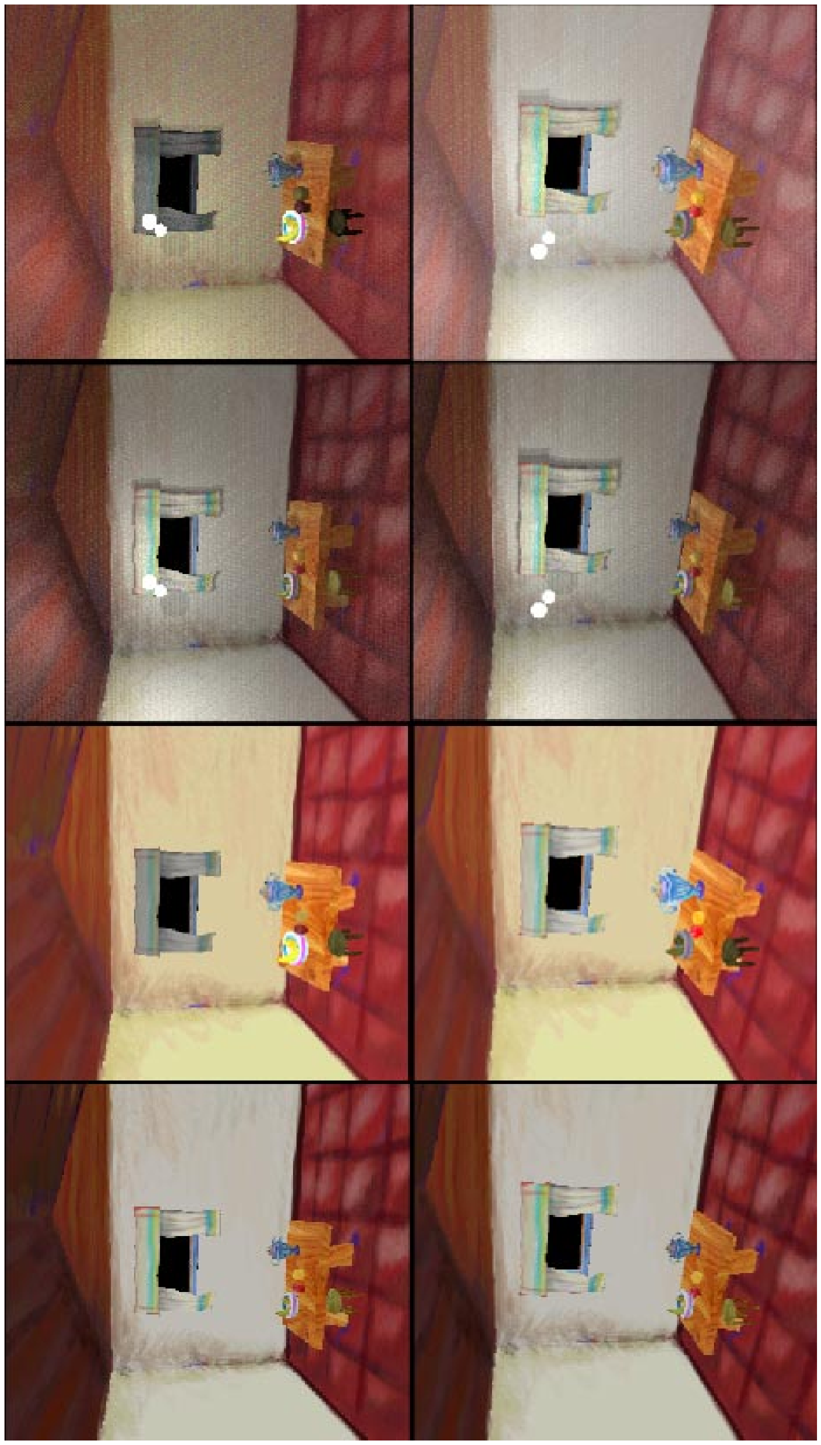


**Figure 3:** *From left to right (as viewed sideways) we have the original OpenGL image, the OpenGL image after compositing, the Radiance image, and the Radiance image after compositing. In the top image the bowl has been tilted forward and brightened, while in the bottom image it is the vase which is turned to face the viewer and increased in size. These are frames 60 and 124 from a 180 frame sequence.*