

# Rendering Multiple Surface Layers for the Purpose of Representing Metallic Patinas

Douglas Thunen  
Department of Computer Science  
Princeton University

May 22, 2003

## 1 Introduction

A typical complaint about computer generated images is that they are too pure and cannot accurately reflect the way objects look in the real world. For example, a copper statue does not forever remain the clear, shiny copper color it originally was; over time moisture, pollutants, and other airborne particles contribute to the tarnishing of, and eventually the development of a patina on, metallic (and other) surfaces; in the case of copper, a patina of that familiar greenish hue, such as is displayed by the Statue of Liberty. It is this problem of modeling the aging of surfaces that we address in a manner similar to that of [DH96].

In our original CS 426 raytracer, as is the case with many computer graphics rendering systems, there was no way to represent multiple surface layers—it was possible only to declare a base material and optional texture map to be applied to objects. While it is conceivable that we could manipulate the texture maps in such a way that they could represent an entire stack of layers to be applied to a surface, *e.g.* a texture map could easily represent lacquered laminate, a thin layer of wood covered by another thin layer of lacquer, and then be applied to some object. This, however, is unsatisfactory for a number of reasons: it does not allow for specifying the many interactions of reflectivity and transmittance through the entire stack of layers; the process of describing the scene, specifically the surface layers, has moved out of the realm of image rendering and into that of image processing; and, as such, can lack some of the flexibility and control that could otherwise be possible.

To address this issue, we sought to implement the ability to support multiple surface layers in our raytracer for the specific purpose of modeling and rendering surface patinas in a manner similar to that proposed in [DH96]. We modify our original rendering algorithm to use the Kubelka-Munk equations of painting and ink printing in order to account for interactions between multiple layers. In addition to providing this support for multiple layers, we allow for the modulation of the thicknesses of these layers through the use of specialized texture maps.

In order to model the aging and weathering of metals, notably the growth of tarnish and patina on copper, we provide mean for automatically changing the thickness of a layer

at individual surface points. In an effort to model the processes that occur in nature, the growth functions take into account surface incline and exposure, where flatter, less exposed regions provide the best conditions for growth, and thus exhibit the highest growth rates.

In Section 2 we provide some background on the problem and various approaches to solving it. Section 3 describes the Kubelka-Munk equations and how we implemented a version of them in our raytracer. In Section 4 we then present a couple of growth functions that allow for the modulation of thicknesses over surfaces, following this with a description of our thickness map encoding. Finally we present some results and conclude.

## 2 Background

The problem of accurately modeling the weathering and aging of metallic surfaces has further importance than simply the ability to generate better and more realistic looking pictures. For example, as stated in [DH96], sufficiently accurate models of aged surfaces can be of great value in predicting how something will look in the future, or conversely, how something looked in the past, or even the conditions that an object has suffered through over time<sup>1</sup>. Accurate models can also be of use in the matter of modeling artificial weathering and aging, such as that which goes into creating antique reproductions or even prewashed jeans.

The primary work on which we are basing this project [DH96] had the goal of developing models for aging and weathering that were highly customizable and as physically faithful as possible, with the focus on modeling the weathering and aging of copper surfaces in various environments. It provided different models for marine, urban, and rural environments, where each takes into account the conditions specific to each environment—for example, high salt concentrations in the air in a marine environment, high levels of airborne pollutants in an urban environment, and more stable conditions in a rural environment.

[WNH97] also addresses some of the problems of modeling the aging process of materials, but with a broader focus of surface imperfections in general, discussing surface peeling (*e.g.* paints, metallic leafs, and such) and the gathering of dust, as well as tarnish and patinas. Perhaps the part of that work that is most applicable to this project is the idea of using surface exposure information to help determine the growth rate of additional layers such as patinas.

While the modeling of aging and weathering processes is an interesting topic and the ultimate goal of this project, in order to get to that problem, we must first address the issue of modeling multiple surface layers at a single point, for example a layer of patina atop a layer of tarnish atop the base metal.

## 3 Rendering Multiple Layers

One common method for modeling multiple surface layers is through the use of the Kubelka-Munk model [DH96, Pha01, WNH97, HM92]. Born in the 1920s, this theory is based on the interactions of photons of light with pigment particles in thin layers of paint on a surface. As light rays meet the surface, they are not totally reflected, but rather some

---

<sup>1</sup>that is, if an object can indeed “suffer.”

fraction travels into the surface, and light that reflects from some point within that surface will be traveling back out through the layer. Some amount,  $K$ , of the light is absorbed by the material, while another amount,  $S$ , is scattered. The fraction  $S$  is also referred to as backscattering, since light from one direction that is scattered is assumed to then be traveling in the opposite direction. This absorption and scattering occurs according to the following differential equations:

$$\frac{\partial B_+}{\partial z} = -(K + S)B_+ + SB_- \quad (1)$$

$$\frac{\partial B_-}{\partial z} = SB_+ - (K + S)B_- \quad (2)$$

where  $B_+$  and  $B_-$  represent the flux density (energy per unit area) in the inward and outward directions, respectively. As shown in [DH96], given the solutions to these equations, the reflectance,  $R$ , and transmittance,  $T$ , through a layer can be computed thus

$$R = \frac{\sinh bSd}{a \sinh bSd + b \cosh bSd} \quad (3)$$

$$T = \frac{b}{a \sinh bSd + b \cosh bSd} \quad (4)$$

with  $a = (S+K)/S$  and  $b = \sqrt{a^2 - 1}$ . Since  $S$  and  $K$  are taken to be functions of wavelength, these calculations must be performed individually for each color sample.

This is good, but from where do the parameters  $S$  and  $K$  come? In order to determine these parameters it is helpful to use the formula for reflectance of an infinitely thick layer: That, when inverted, yields the formula Which allows for the straightforward derivation of  $K$  and  $S$  when  $R_\infty$  is known. Usually  $S$  is chosen to be some reasonable value, perhaps 1.0 and then  $K$  is computed. The units of  $K$  and  $S$  are unimportant, as are there absolute values, as the important value is simply their ratio. Light-surface interactions are, like most other lighting phenomena, incredibly complicated processes that are all but impossible solve without making at least some simplifying assumptions; the Kubelka-Munk theory is no exception. Among the necessary assumptions is that the pigmented solution, i.e. the layer we are modeling, is a homogeneous solution, with an even distribution of particles; this is not always the case as differences in density can be common, manifesting themselves in such forms as clumping or floating particles. In addition, this model assumes completely diffuse lighting, with no provisions for surface specularity. Lastly, it is assumed that surfaces are entirely planar, meaning that light can only be absorbed, scattered, or unaffected. The process as discussed above describes the interactions of light with a single material, but one of our primary goals was to extend our rendering algorithm to support multiple surface layers. This is not a problem, however, since it is relatively straightforward to extend the single layer Kubelka-Munk equations to multiple layer situations. The equations for computing composited reflectance and transmittance values,  $R'$  and  $T'$  are as follows:

$$R' = R_1 + \frac{T_1^2 R_2}{1 - R_1 R_2} \quad (5)$$

$$T' = \frac{T_1 T_2}{1 - R_1 R_2} \quad (6)$$

These operations can then be composited any number of times to compute values for a whole stack of layers. A nice property of these equations is that they are commutative, *i.e.* the order in which we perform the compositions is unimportant. It should be noted that there is a  $T^2$  factor in each  $R'$  calculation, which is due to light travelling through each layer twice (once in each direction). Also, there is a factor of  $\frac{1}{1-R_1R_2}$ , which reflects the multiple scattering that occurs in each layer. As [DH96] notes, this is an important property unaccounted for in the standard computer graphics model for partially transparent surfaces that does not consider these multiple interactions.

### 3.1 Kubelka-Munk and specularity

The major difficulties in our implementation arose when we began considering material with non-zero specularity. By definition, when using the Kubelka-Munk framework, any attempts to deal with specularity will be hacks since one of the primary assumptions of the Kubelka-Munk model is of entirely diffuse lighting. There have, however, been many successful, if not physically correct, solutions to adding specularity to this model. [DH96] take the approach of modeling glossiness at each layer interface in the standard manner, and then also accounting for attenuation due to absorption and scattering by multiplying the specular color by the transmittance of all of the layers above it,  $T'[i]$  for each layer  $i$  twice. The total specularity, then, is the weighted sum of the specular components for each layer. As [DH96] states, however, this is only an approximation since calculating attenuation in both directions (the multiplying by a factor of 2) using diffuse transmittance ( $T'$ ) is not entirely accurate, as attenuation is actually also a function of direction. Furthermore, these calculations only use diffuse attenuation, and do not take into account any scattering or absorption of the specular reflections.

A similar approach is described in [Pha01], but this approach, in addition to computing the diffuse component in the standard Kubelka-Munk fashion, updates the specular component in a similar manner. The new specular coefficients are calculated using the same reflectance equation as for diffuse components, except here  $R_1$  for the top layer is set to zero, since it is assumed that the top layer is completely diffuse. Once these new values are computed,  $k_{spec}$  can also be updated by multiplying the original  $k_{spec}$  value for a material by the ratio of the new specular coefficients to the old. Just as in the solution of [DH96], however, this too is not faithful to physics. Nevertheless, it produces acceptable images.

We follow an approach similar to that of [Pha01], computing new specularity coefficients using the reflection and transmittance of intervening layers and then updating  $k_{spec}$  by using the square of the ratio of the new coefficients to the old. This method seems simple enough, but some complications arise in its implementation due to the fact that, unlike for diffuse calculations, for specular lighting calculations  $k_{spec}$  must be known beforehand, and thus the calculation will be different for each layer as  $k_{spec}$  is different for each layer. Because of this, we must precompute the new  $R_{spec}$  and  $k_{spec}$  values for each intersection before we do the lighting calculations. With this information in hand, though, we can perform the necessary calculations, computing the total illumination for each pixel.

## 4 Growth Functions

Given that we are attempting to model the tarnishing of a metal surface and the formation of patinas on that surface, we would like to provide some methods for defining the growth of layers, *i.e.* the increasing of the thickness of layers. To this end, we provide two such functions, steady thickening (ST) and random deposition (RD), similar to those described in [DH96].

### 4.1 Steady Thickening (ST)

Steady thickening is the simpler of the two, assigning some thickness to a small sampling of points in the image and then using bilinear interpolation between these control points to get thickness values for all of the other surface points in the image. A user-defined growth parameter specifies a constant amount to add to the thickness at the control points over time; in an effort to keep the image from looking too ordered and unnatural, a small amount of noise is also added<sup>2</sup>. The user also has the option of whether to use the surface incline and exposure scaling factors or to allow the growth to be independent of surface parameters and more uniform.

Our implementation of steady thickening involves sampling at three x-values every five horizontal lines in the image, meaning that the samples in a 200x200 image would be at (0,0),(99,0),(199,0),(0,5),(99,5),(199,5),... This can place control points on pixels not covered by the specified layer, but we do not see this as an issue since ST simply defines a growth rate for a layer and intermediate pixel values are determined by interpolation of the control points. With this definition, it does not matter whether a control point actually lies on the surface of interest—the results will be the same regardless.

### 4.2 Random Deposition (RD)

Random deposition does exactly what its name suggests: it involves dropping a particle above a random point on the surface, allowing it to fall until it hits the surface, at which point it increases the thickness by a user-specified value. Due to randomness this can result in a very uneven surface so, as in [DH96], we also allow the option of “surface relaxation” that allows a particle to find a local minimum within a certain distance of the initial point of contact. Again, the user may specify whether to use scaling parameters.

Our specification for the RD (and RD<sub>r</sub>, for relaxation) is as follows: The program reads all of the thickness information from each of the layers specified in the layer stack, coming up with a total thickness at each point. This thickness is used to determine the final resting place of a particle when surface relaxation is turned on. The user-specified factor determines how many times the program should increase the thickness, *e.g.* a factor of 0.5 means that, on average, one half of the “active” pixels, that is the pixels that are covered by the top layer, will have a particle deposited on them, and a factor of 3.0 would mean that 3 particles are dropped per each active pixel. Finally, *amt* is the growth rate, here the increase in thickness associated with each particle (an integer value between 0 and 255).

---

<sup>2</sup>We arbitrarily specified the noise to be 1/5 times the growth rate.

## 5 Texture or thickness maps

Tarnishes and patinas do not grow uniformly, nor completely randomly, on surfaces. Rather, their growth is a function of many parameters, two of the most important of which are surface incline and surface exposure. A flat surface, *i.e.* one parallel to the ground plane, will allow better for the accumulation of moisture and other particles, thereby providing a surface more conducive to patina formation than a steeply sloped one. Moisture and other particles cannot adhere as easily to a vertical or near vertical surface and therefore patina growth is slower on these surfaces than flatter ones.

In order to model this phenomenon, we maintain indices of the surface inclines in our texture maps. We calculate the surface slope factor as  $sign(N * U)(1 - |NxU|)$ , where  $N$  is the surface normal at the point of intersection and  $U$  is the model up vector (typically  $(0.0,1.0,0.0)$ ). In this manner, the factor lies in the range  $[-1,1]$ , with the highest values occurring where the surface is parallel to the ground and facing in the same direction as the ground, decreasing as the slope increases, approaching 0 at a completely vertical surface, and then decreasing to -1 at a surface parallel to the ground and facing towards the ground.

As well as taking surface slopes into account, we keep track of surface exposure. We determine surface exposure in a manner similar to that of shadow detection. The difference, however, is that when performing shadow calculations only a single ray need be cast, in this situation, more work must be done. Ideally, to determine the exposure at a point we would cast an infinite number of rays from that point and then take the average of the values for all of these rays to get the exposure of that point. This, of course, is impractical due to its (infinitely) high computational cost, and we thus approximate this function by casting a small number of rays from each point, similar to the process described in [WNH97]. We make the further approximation of casting ray randomly from the surface rather than according to a specific distribution. While this introduces a certain amount of noise into our value, it is a simple means of obtaining a good approximation of surface exposures. In choosing how many rays to cast, we are faced with the common tradeoff of speed vs. accuracy. Casting rays and calculating intersection information is a costly process, so if we must cast a large number of additional rays to determine exposure information, we can again run into the problem of prohibitively high computational costs. More ray, conversely, mean a better chance of approaching the true value, and therefore also less noise and more accurate values. Because the exposure calculations must only be performed once for a model view (if we intend to render a model at many stages of the weathering process, we need only calculate exposures once, so long as we maintain the same view, *i.e.* we do not change camera position or direction), we can afford somewhat more computation in this component of the rendering system.

## 6 Goals and results

Our goal was to recreate the results of [DH96], namely to be able to simulate the weathering and aging of metallic surfaces, specifically copper due to its distinctive (and attractive) patina. In order to reach this goal, there were, as mentioned previously, four primary tasks: create a texture/thickness map format to encapsulate all necessary information (thickness,

environmental scaling factors); develop growth functions for the thickness maps; develop operators on the thickness maps, e.g. coat, polish, erode, that would help model the aging process; extend our raytracer to support multiple surface layers using the Kubelka-Munk theory. We successfully completed all of these tasks, but actually modeling the aging process proved more difficult.

## 6.1 Results

In terms of the first task of creating a thickness map that both held all necessary information and was easy to manipulate in tasks 2 and 3, the use of standard 24bit RGB bitmaps seemed optimal—not only was it able to encapsulate thickness, incline, and exposure information, but also it provided simple manipulation and decent visualization. For growth functions, we chose to implement steady thickening (ST) and random deposition (RD) as well as random deposition with surface relaxation. While these functions produce results that conform to our specifications of the functions, that is ST does indeed steadily thicken a thickness map according to its parameters, and RD (and RD with relaxation) demonstrates random growth according to its parameters, the use of these functions alone seems insufficient to provide believable tarnish and patina growth—ST is too uniform, while RD, even with relaxation, produces results that are too spotty. As with our growth functions, our thickness map operators perform according to specification, but we were unable to determine the best parameters and combinations of these operators and growth functions to produce a realistic aging process. Finally, we were able to extend our raytracer to fully support the rendering of stacks of multiple surface layers.

## 6.2 Shortcomings

In general, our approach seems promising, although it does have some drawbacks, most notably that our thickness maps, including the incline and exposure scaling factors, are view dependent. This approach means that this information is not intrinsically maintained with the model, but rather is tied to the final rendered image. Unfortunately, this requires recomputing all of this information any time the view (or even image size) is changed. It does, however, have a number of beneficial properties: if we do not care about recursive rays it allows for very quick renderings of the same view with different thickness values; it provides a meaningful visualization of our parameters; it provides detail at an optimal granularity, that is it provides exactly pixel-level granularity. Most other imaginable representations would require some form of interpolation between points since there would not be a one-to-one pixel correspondence between thicknesses and pixels in the final rendered image. Our use of bitmaps of the same size as that of the rendered image is also somewhat space inefficient—24 bits for every pixel in the final image, when in many cases, i.e. those pixels where that layer is not present, this is clearly overkill. It is likely that there is some more complex format that would be able to maintain all of the necessary information in a much smaller format. Also, the use of floating point values for thickness and scaling factors would allow for a finer level of control than the 256 distinct integer pixel values that the use of 24 bit bitmaps provide. Despite this shortcoming, we chose the bitmaps for the simpler manipulation and better visualization.

As mentioned earlier, rendering of complex scenes with our raytracer is painfully slow—the Statue of Liberty model that we had hoped to use consisted of 7,910 vertices and approximately 15,000 triangles<sup>3</sup>, and the preprocessing, *i.e.* the creation of the thickness maps, took 63 minutes to complete even when rendered at the relatively low resolution of 200x200 under WindowsXP on our 1.8Ghz P4 IBM Thinkpad with 256MB of RAM. Because of this slowness, some form of acceleration, e.g. bounding boxes, octrees, or BSP trees, would be beneficial. Also because of our speed problems we provide no support for recursive rays—this is not a problem for the simple scenes we intended to render, but would be important for a more generalized system.

## 7 Conclusion

Currently our system provides the basic functionality that we had hoped for from the outset. This is not to say, however, that it is in any way complete; in addition to the possible improvements noted in the previous section, some form of more cohesive integration of the various components of the system would be helpful—as it now stands, in order to produce a final image of a scene that contains objects with multiple surface layers, the user must first perform the preprocessing step that creates the appropriate thickness maps, then apply any appropriate operators or functions to these maps, and finally feed the `.ray` file and the modified thickness maps to the renderer, which outputs the final image. While it would certainly be better if this could all be done in a single step—the user inputs a `.ray` file and appropriate aging parameters to the system, which subsequently outputs the final image—the most important thing is that each of these steps *can* be done, even if only individually.

In completing this project we were introduced to another model for calculating light-surface interactions in a rendering system, namely the Kubelka-Munk model, which is applicable not only to computer graphics, as in this project, but also to the actual mixing of and combining different pigments in paints, inks, and the like. Furthermore, we were able to use this system to investigate the properties of multiple surface layers of varying thickness, thereby producing images that more accurately reflect the interactions that occur in the world.

## References

- [DH96] Julie Dorsey and Pat Hanrahan. Modeling and rendering of metallic patinas. In *Computer Graphics (SIGGRAPH 1996 Proceedings)*, pages 387–396, 1996.
- [HM92] C.S. Haase and G.W. Meyer. Modeling pigmented materials for realistic image synthesis. *ACM Tran. Graphics*, 11(4):305–335, 1992.
- [Pha01] Matt Pharr. Layered media for surface shaders. In *SIGGRAPH 01 Course Notes*, pages 41–63, 2001.

---

<sup>3</sup>The model actually consisted of over 10,000 vertices and 34,000 triangles, but we removed the base in an effort to achieve slightly more reasonable processing times.

- [WNH97] Tien-Tsin Wong, Wai-Yin Ng, and Pheng-Ann Heng. A geometry dependent texture generation framework for simulating surface imperfections. In *Proceedings of the Eighth Eurographics Workshop on Rendering*, pages 139–150, 1997.