Modeling Stained Glass

Jiwon Shin

May 5, 2005

Abstract

We present an algorithm that models stained glass in a physically-realistic manner. Stained glass is made by melting glass material with metal oxides that determine the color. Metal oxides are scattered in the glass, and we model it by distributing metal oxides in the volume defined by the glass. We render our stained glass model using a modified photon mapping. Our model is capable of modeling stained glass and render a scene with stained glass realistically.

1 Introduction

The computer graphics community has developed nonphotorealistic rendering algorithms for various types of art such as sumi-e painting[15], oil painting[10], technical illustration[5], and escherization[9]. However, relatively little works has been done on stained glass windows.

Stained glass window is a popular art form of the medieval period. A stained glass window consists of pieces of colored glass held together by lead. As each piece of glass can be only so small, stained glass windows could only show a limited amount of detail. To overcome this problem, many stained glass window artists of the medieval period added a thin layer of black or brown paint on the glass window, detailing faces, hands, drapery, etc.[12].

The basic methods for making stained glass have not changed much since the twelfth century. Stained glass is made by melting sand, potash, and lime together in clay pots, and adding metal oxides to the mixture to color the glass. Metal oxides used for staining include copper for red, iron for green, cobalt for blue. This method produces acceptable stained glass sheets for many colors. However, for certain color such as red, this technique produces glass pieces that are too dark to transmit much light. For these, another method is used to make stained glass. They are produced by dipping a lump of pure glass (without metal oxides) on a blowpipe into a pot of red glass and blowing the fused material. This creates sheets of glass with a thin layer of colored surface, preserving the color while letting much more light to transmit [12].

In this paper, we propose an algorithm for accurately modeling and rendering stained glass. Section 2 discusses previous work on stained glass and other relavent topics. The theory behind the work is presented in section 3, and the

algorithm is presented in section 4. The results of the work are presented in section 5, and section 5 presents conclusions about the work. Future work is mentiond in section 7.

2 Previous Work

Literature on stained glass windows has focused on segmentation aspect of stained glass techniques. The PhotoShop stained glass filter [13] employs a technique based on Voronoi regions [14]. The algorithm is simple, but it places no regard for the input content, resulting in glass tiles that do not reflect the underlying image. It produces an image that fails to resemble stained glass windows. Mould [11] proposed a stained glass image filter algorithm that generates glass tiles based on the input image. It uses a simple segmentation algorithm to segment the input, then postprocesses segmented regions to smooth out boundaries, merge small regions together, and split large regions into smaller ones. Glass is modeled using a displacement map, which introduces irregularities to the glass surface. This method does not model structure and optical properties of stained glass, and fails to create realistic stained glass images.

Our approach is based on physical realism. Although, to our knowledge, there has not been any work specifically on realistic modeling of stained glass, work on topics such as particle simulations [4, 6] is closely related to our method. Dobashi et al. [4] simulate cloud by dividing world into voxels, where each voxel stores information on physical properties of cloud at the location. Jensen, Legakis, and Dorsey [6] model wet surfaces by dividing water into two regions - thin layers for the water surface and a subsurface region and treating subsurface scattering of materials as participating media. Participating media refer to particles that are introduced to otherwise homogeneous medium, or materials that are inhomogeneous by nature. Examples of participating media include dusty air, clouds, and silty water, as well as translucent materials such as marble, skin, and plants [7].

Displaying photorealistic images requires a rendering algorithm that is based on physical reality. Dobashi et al. [4] use a two-pass volume rendering technique to render clouds. They first compute the light intensity reaching the center of each voxel, and then renders the scene using a spherical shell. Jensen, Legakis, and Dorsey [6] uses the photon mapping algorithm [8] to render wet materials such as wet sand and cognac spilled on wood table.

Our rendering algorithm is based on the photon mapping algorithm presented in [7].

3 Theory

3.1 Stained Glass

Stained glass is made by melting silicon dioxides with metal oxides. When metal oxides are introduced to molten glass and mixed, they are randomly distributed

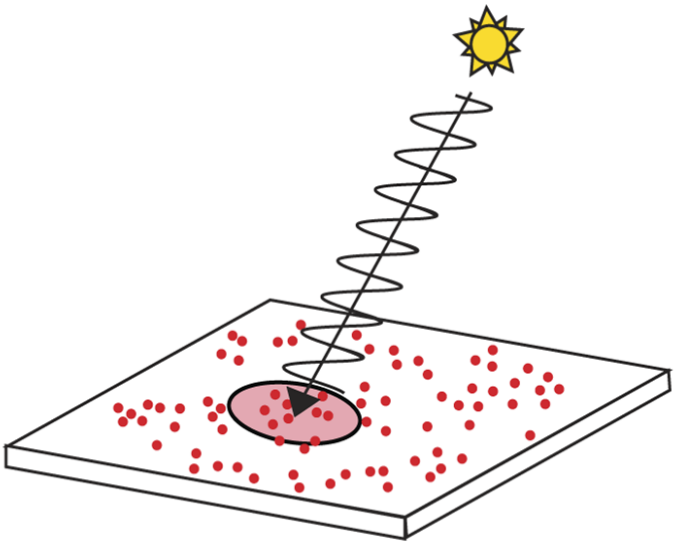


Figure 1: Light wave interaction with stained glass

throughout the glass. However, as the glass cools down, metal oxides attract other metal oxides and separate themselves from it. As the distance a metal oxide can travel is limited, the degree of clumping of metal oxides is determined by the speed at which stained glass is cooled.

The chemical composition of stained glass creates stained glass' complex optical properties. For homogeneous materials, a light wave's interaction can be easily determined by the material properties at the hit point. Stained glass, on the other hand, is an inhomogeneous dielectric. Metal oxides in the glass are not evenly distributed and they are much smaller than light waves. Therefore, when a light ray reaches stained glass, it acts according to the average property of the metal oxides in the area determined by the ray (Figure 1). The transmission of the ray is determined by the density of the metal oxides collected. A region with high density of metal oxides is opaque where one with low density of metal oxides are translucent.

3.2 Photon Mapping

Photon mapping [7] is a global illumination algorithm that generates, stores, and uses illumination as points named *photons*. Each photon contains information on its position, power, and incoming direction. These photons are stored in a data structure called *photon map*. A photon map is the backbone data structure for a photon mapping algorithm.

The photon mapping algorithm is a two-step algorithm - photon tracing and

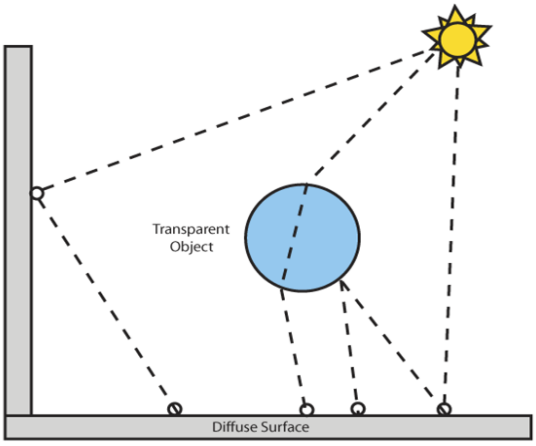


Figure 2: Photons that interact with diffuse surfaces are stored in a photon map

rendering by ray tracing. Photon tracing is the process of building a photon map by tracing photons from the light sources into the scene. These photons stored in the photon map are then used to render the scene efficiently.

3.2.1 Photon Tracing

Photon tracing is the process of emitting photons from the light sources and tracing them through the objects, recording their interactions along the way.

The first step of the process is the photon emission. Photons are emitted into the scene from the light sources. A light source can be of any type, from a simple point light source to physically-based sources with an arbitrary geometry. As in nature, each light source emits a large number of photons. The power of a light source is divided among all the photons it emits; hence, each photon has a fraction of the power of the source.

Each emitted photon is traced through the scene using photon tracing (Figure 2). Photon tracing works the same as ray tracing with only one difference: a photon used in photon tracing propagates flux whereas a ray in ray tracing gathers radiance. This means that the radiance depends on the index of reflection of the material while flux does not.

When a photon hits an object, it can be reflected, transmitted, or absorbed. Whether a photon is reflected, transmitted, or absorbed depends on the object's material properties. If a photon hits a mirror surface, it is reflected in the mirror direction. The reflected direction, $\vec{\omega}$ is:

$$\vec{\omega} = 2(\vec{n} \cdot \vec{\omega'})\vec{n} - \vec{\omega'} \tag{1}$$

given a normal, $\vec{n}$ and an incoming direction, $\vec{\omega'}$. The incoming direction is assumed to be pointing away from the intersection point. The power of the outgoing photon is scaled by the specular reflectivity of the surface unless an important sampling technique such as Russian roulette sampling is used, as explained later in this section.

When a photon interacts with a diffuse surface, it is stored in the photon map. The direction of the outgoing photon is determined by picking a random direction in the hemisphere above the intersection point, where the probability of picking one direction is proportional to the cosine of the angle with the normal. Just as in specular reflection, the power of the outgoing photon is scaled by the diffuse reflectivity of the surface.

Photon scattering is a computationally expensive process. To speed up the process, a stochastic sampling technique called *Russian roulette sampling* is used. The basic idea behind Russian roulette sampling is that we can eliminate photons that are not important, hence save computation time, but still produce the correct result.

Given a material with reflectivity, d with a range of [0, 1], and an incoming photon with flux Φ_P, we decide whether a photon is absorbed or reflected using Russian roulette. We generate a uniformly distributed random number, ζ, in [0, 1]. If ζ is less than the probability of reflection, then the photon is reflected with the power Φ_P. Otherwise, the photon is absorbed. If the surface has both specular and diffuse reflection, then the decision is made the following way:

$$\begin{aligned} \zeta \in [0, \rho_d] &\longrightarrow \text{diffuse reflection} \\ \zeta \in (\rho_d, \rho_s] &\longrightarrow \text{specular reflection} \\ \zeta \in (\rho_s, 1] &\longrightarrow \text{absorption} \end{aligned}$$

where the sum of the diffuse reflectance,ρ_d, and the specular reflectance, ρ_s, is less than or equal to 1. If the photon is reflected, the photon leaves the surface in the reflected direction with the same power as that of the incoming photon. Photons interact differently with stained glass, as explained in the later section.

Only those photons that hit diffuse surfaces are stored in the photon map. Storing photons that hit specular surfaces is not necessary since this information can be gathered by ray tracing, as described in the next section.

3.2.2 Rendering by Ray Tracing

Ray tracing [16] is a rendering algorithm that sihmulates the specular interaction among objects in a scene. It is based on the idea that the photons that the light sources emit can be traced back to the observer. A portion of the rays reaches the observer, but most of the rays do not. Hence, tracing all the rays from the light sources is a computationally expensive process. Instead, we used a backward ray tracer for this project, a technique of tracing rays from the observer back to the light sources [1].

A ray tracer takes the position of the observer, an image plane, and the information about the scene such as the location, geometry, and material properties of the objects and the light sources. For each pixel in the image plane,

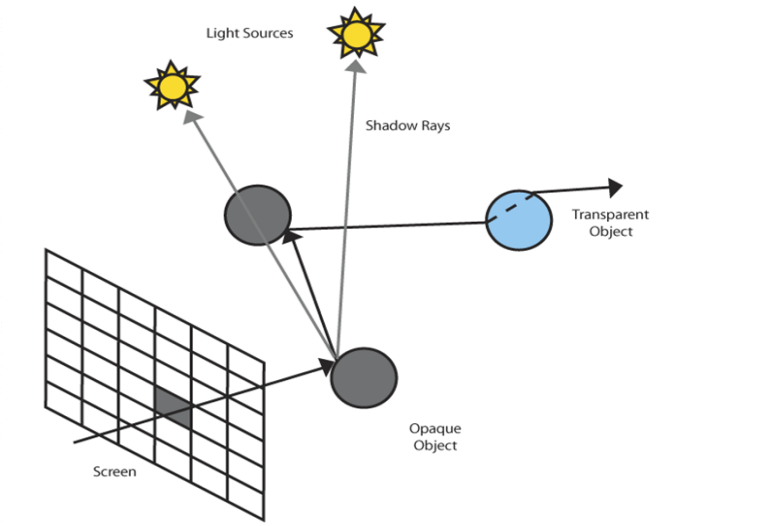


Figure 3: A backward ray tracer with shadow rays from the hit point to the light sources

a ray is shot out into the scene. When the ray hits an object, it sends out a reflection and transmission ray. These rays, in turn, interact with objects in the scene (Figure 3).

At every hit point, we calculate the contribution of each light source to the illumination of the point, known as local illumination. To do this correctly, a shadow ray is set to each light source from the hit point. If the shadow ray reaches the light source, then the hit point is fully visible by the light source, and its illumination is calculated appropriately. If not, then it is assumed that the light source makes no contribution in the illumination of the hit point.

The illumination of a hit point is determined by summing up the local illumination and global illumination gathered by the reflection and refraction rays. In addition, if the surface is diffuse, photons are gathered at the point, the average flux is calculated, and the result is added to the total illumination.

4 Algorithm

4.1 Modeling

Each sheet of glass is defined by its location in the scene, geometrical structure, and metal oxides that are used to color the glass. A metal oxide is a data structure that stores the position, power, and index of refraction of a metal oxide. In the modeling phase, metal oxides are randomly generated to be inside the glass, and they are stored in a k-d tree (explained below). Once all the metal oxides are generated, they are clumped together to simulate the clumping

of metal oxides. The degree of clumping varies depending on the cooling speed of the molten glass. The parameter is defined by the user.

A key in modeling stained glass is picking out a data structure that is capable of storing a large amount of three-dimensional data compactly and has a short access time for locating a set of data points. We decided to use a k-d tree as our data structure as it fulfills both requirements.

k-d tree[3] is a k-dimensional binary tree. Each node divides the space into two halves, where all the nodes left of the node are below the node in the bisecting dimension and all the nodes right of the node are above. The advantage of k-d tree is that when it is balanced, the worst case search time for a single node in the tree takes $O(\log\ n)$ as supposed to a regular binary tree, which takes $O(n)$. Using a k-d tree is especially advantageous for locating nearest neighbors. k-d tree can find m nearest neighbors in $O(m + \log\ n)$ [2], much faster than other algorithm.

We use a mapping techique to model stained glass. When the user adds a piece of stained glass to the scene, the algorithm generates a corresponding sheet of glass in the model coordinates, and determines the mapping matrix between the glass map in the model coordinates and its position in the world coordinates. The glass map is always defined as a rectangular sheet whose x and y are twice as long as that of the desired piece. We used the mapping technique for two reasons: first, a piece of stained glass in a stained glass window is a section of a large sheet of stained glass, and mapping techinque allows us to mimic the process. Second, if metal oxides are only distributed in the area defined by the piece of glass, then it can cause aliasing around the edges of the piece. With a sheet that is four times the size of the desired piece, we can avoid the aliasing problem.

4.2 Rendering

We used a slightly modified version of photon mapping algorithm to render a scene with stained glass. In photon mapping described previous, the flux of a photon does not depend the objects it interacts with in the scene. However, when light passes through a transparent object such as stained glass window, its properties are slightly modified. In particular, glass acts as a filter, blocking certain wavelengths of light. For a regular glass object, photon mapping algorithm as suggested by [7] is good enough as glass only blocks certain invisible light waves. For a stained glass windows, however, this is not the case. In addition to blocking invisible lights, it also blocks certain visible lights based on the metal oxides. Hence, rendering a scene with stained glass requires the photons that are transmitted trough stained glass to be modified.

When a photon is transmitted through a piece of stained glass, the material properties of the stained glass are determined by gathering metal oxides along the path of the photon. The flux of the photon is then multiplied by the average power of the metal oxides, and sent out into the scene. This modification makes the simulation of color bleeding possible.

In addition to modifying photon tracing, we modified ray tracing to render

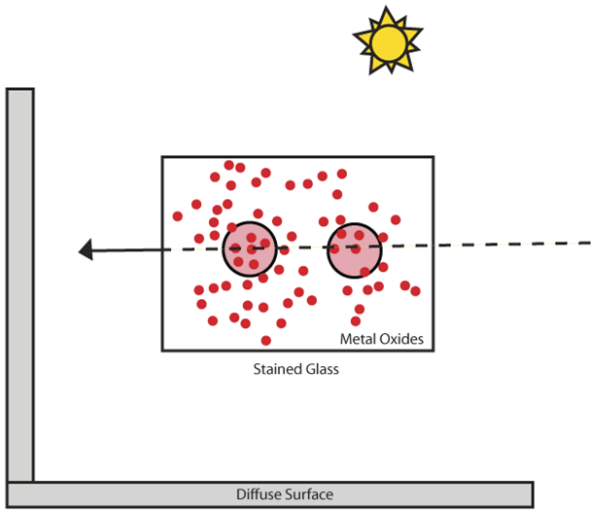


Figure 4: A graphical representation of ray marching inside stained glass

stained glass more realistically. In regular ray tracing, when a light ray is transmitted, its radiance is not gathered until it intersects the other side of the object. This is, however, not sufficient to correctly render stained glass.

As mentioned earlier, stained glass is inhomogeneous. This means that a light ray that hits stained glass at point A is not transmitted the same way a ray that intersects with the glass at point B. In addition, the properties constantly change as the ray passes through the glass. To take this ever changing local properties into account in determining illumination, we employ a ray mayrching algorithm [7] to march through the glass.

Ray marching works like the following (Figure 5): when a light ray enters stained glass, it marches through the glass until it reaches the other end of the glass. From the hit point to the intersection point on the other side of the glass, the ray moves forward by a uniform step with some perturbation to avoid aliasing. At the end of each step, the ray gathers m nearest metal oxides, and determines the material properties of the location by averaging the properties of the metal oxides gathered. The material properties then determine the contribution of upcoming illumination to the overall illumination. If metal oxides are densely populated, then only a small amount of light is tranmitted through the layer, adding little to the overall illumination of the glass. On the other hand, if metal oxides are distributed sparsely, most of the light is transmitted, and the illumination gathered by the transmitted ray plays a significant role in the overall illumination. Ray marching allows us to correctly render stained glass even when metal oxides are distributed sparsely near the surface but densely

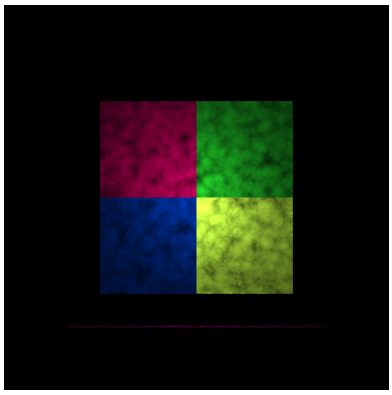
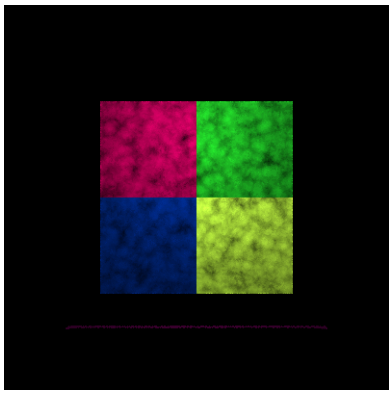
(a)  (b) 

Figure 5: Images with a point light source at two different locations: (a) directly behind (b) away from the glass

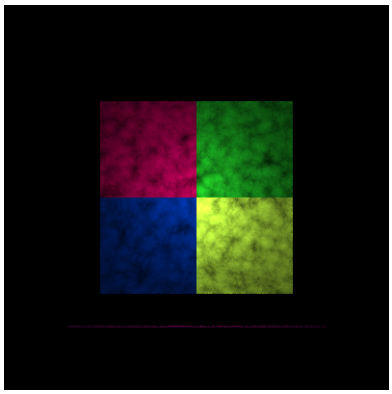
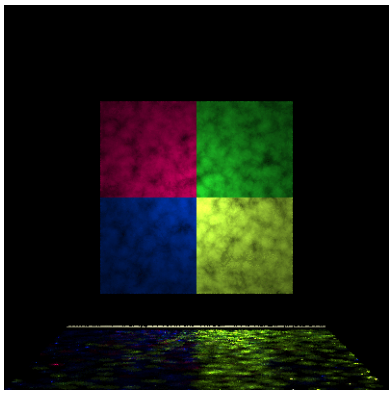
(a)  (b) 

Figure 6: Images of stained glass (a) without photon mapping (b) with photon mapping

populated everywhere else.

5 Results

Here, we present some images generated using the algorithm to show how our algorithm works.

Figure 6 shows two images of the same setup, with a point light source behind the stained glass. The image on the left has the light source directly behind the stained glass. The light source on the right is also behind the glass but farther away. We can notice that the stained glass on the right is more evenly lit up than the one on the left.

Figure 7 shows the result of our algorithm. Both images contain four pieces

of stained glass with a light souce at the center, behind the glass, and a diffuse surface perpendicular to the image plane. The image on the left is rendered by ray tracing while the one on the right is rendered using the modified photon mapping. The diffuse surface on the left is not lit up, hence invisible, because the stained glass blocks the only light source in the scene. The same surface is visible in the image on the right because photons that pass through the stained glass are collected on the surface. Notice how stained glass filters photon power. All the photons that are transmitted through the glass only carry the components that are not filtered out by the stained glass.

6 Conclusions

We have presented an algorithm for modeling and rendering stained glass in a physically realistic manner. Our model is based on the chemical and optical properties of stained glass. We have shown that our model generates sheets of stained glass that are convincing even when photon mapping is not used. We used a photon mapping algorithm to correctly simulate the interaction of photons in a scene with stained glass. The original photon mapping algorithm by Jensen[7] could not to correctly render stained glass' interaction with other objects, but our modified rendering algorithm successfully filtered the photons that pass through stained glass and produced color bleeding effect accurately.

7 Future Work

This project has focused on modeling and rendering stained glass. A stained glass window, however, consists of pieces of stained glass glued together by lead. To generate a stained glass window from an input image, we need to add a segmentation algorithm that can cartoon the input in a stylistic manner. In addition, a realistic model of lead needs to be added.

We can also add a paint layer to our glass modeling algorithm to simulate painting of glass windows.

Once above goals are accomplished, it would be nice to be able to animate a scene with stained glass windows. To do this, it is necessary to optimize our algorithm. While the current implementation is capable of generating a single image in a reasonable amount of time, it is too slow to be used to generate an animation. The algorithm can achieve a speedup by using an efficient algorithm for ray tracing.

Acknowledgements

I would like to thank my advisor, Dr. Bruce A. Maxwell for his guidance and patience throughout the project. I also would like to thank Dr. Carl Grossman for his help with optics. Yavor Georgiev '06 helped me greatly with the figures

used in this paper, and I thank him for that. Lastly, I would like to thank all my professors and friends for making my college experience memorable.

References

[1] James Arvo. *Backward Ray Tracing.* SIGGRAPH 86, Dallas, TX, August 1986.

[2] Jon L. Bentley. Multidimensional binary search trees in database applications. In *IEEE Trans. on Software Engineering*, volume 5(4), pages 333–340, July 1979.

[3] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[4] Yoshinori Dobashi et al. A simple, efficient method for realistic animation of clouds. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 19–28, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[5] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A non-photorealistic lighting model for automatic technical illustration. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 447–452, New York, NY, USA, 1998. ACM Press.

[6] Justin Legakis Henrik Wann Jensen and Julie Dorsey. Rendering of wet materials.

[7] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping.* AK Peters, Ltd, Natick, MA, USA, 2001.

[8] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scences with participating media using photon maps. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 311–320, New York, NY, USA, 1998. ACM Press.

[9] Craig S. Kaplan and David H. Salesin. Escherization. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 499–510, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[10] Barbara J. Meier. Painterly rendering for animation. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 477–484, New York, NY, USA, 1996. ACM Press.

[11] David Mould. A stained glass image filter. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 20–25, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[12] The Stained Glass Museum. Stained glass - a brief history, 2005 (retrieved on May 5, 2005).

[13] Donnie O'Quinn. Photoshop 6 shop manual, 2001.

[14] Joseph O'Rourke. *Computational geometry in C.* Cambridge University Press, New York, NY, USA, 1994.

[15] Steve Strassmann. Hairy brushes. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 225–232, New York, NY, USA, 1986. ACM Press.

[16] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, 1980.