Giving Virtual Objects Color using Geometry

Zhe Deng

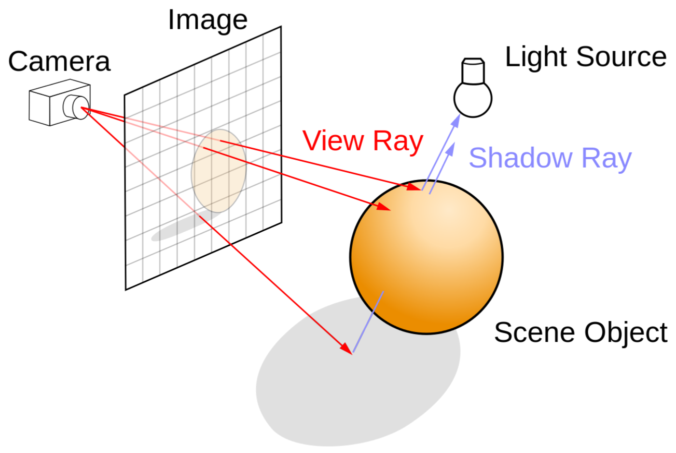
Abstract

Color dominates vision. Intensity, contrast, and colors themselves are essential to sensing spatial awareness. The light that reach our eyes give visual cues as to what type of material and shapes surround us. Simulating this natural phenomenon with computer graphics requires a few fundamental geometric techniques. This paper explores geometry contained in the origins of coloring--or "shading"--virtual images as well as a modern Ray-Tracing through accompanying code.

1. Introduction

Geometry is integral to the field of computer graphics. At its core, a computer screen is an embedding plane in real projective space (RP^2). In other words, virtual "ray" lines from the origin (which represents a camera or the user's point of view) cut through the monitor plane ("Ray Tracing (Graphics)"). Depending on the shading technique and the ray-scene intersection points, each "view ray" and its associated pixel will have a certain associated color. As a result computers capture a single perspectivity or image of the 3D scene on a 2D screen.

Fig. 1: Shading from the Ray-Tracing Perspective



("*Ray Tracing (Graphics)*")

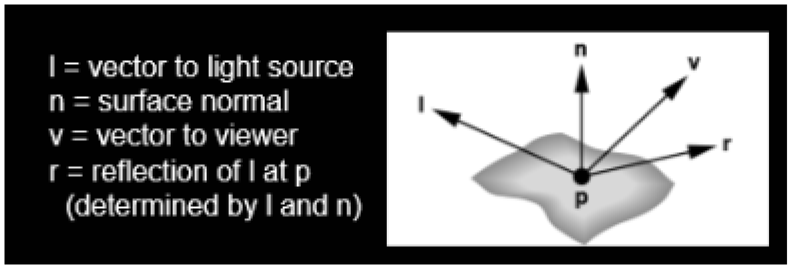
Many types of shading exist today, but they all use at least one of the three fundamental light-surface interactions: 1.) Specular 2.) Diffuse and 3.) Dielectric. Specular surfaces, like metals, are shiny and reflect light at a set angle depending on the surface and light ray. On the other hand, Diffuse surfaces reflect light at random angles to create a matte appearance. Dielectrics or translucent surfaces, like glass, have the same reflection properties of specular surfaces, but a some of the light-surface interaction also involves refraction (see Fig. 5). Early approaches tried to capture these properties via approximations and computationally cheaper methods. On the other hand, modern techniques tend to be simpler and more intuitive, but more computationally expensive.

1.1 Phong Lighting

Perhaps the most famous and applicable style of shading is the one introduced by Professor Bui Tuong Phong in his 1975 paper "Illumination for Computer Generated Pictures" (Phong). In his paper, Phong proposes a shading technique that accounts for two main factors: 1. light and 2. light-surface interactions with one reflection. Similar techniques are now referred to as "traditional rasterization." Since light is not altered by itself, most of the math occurs when it hits a surface.

Each pixel's color is based off of the material's reflection constants for ambient, diffuse, and specular reflections (k=[k_a, k_d, k_s]). The interacting light ray has a similar set of 3 constants (L=[L_a, L_d, L_s]). Note that although there is at most one reflection per light ray, multiple light rays can affect a single point. Both k and L have RGB components, so they are effectively 3x3 matrices. To find the current pixel's color, we first find its corresponding intensities ([I_{ia}, I_{id}, I_{is}]=k.*L where .* represents element-wise multiplication, see Appendix 4.1). Then, using physical laws, I_{is} is scaled by $(r \bullet v)^{\alpha}$, where α is the "shininess coefficient" and "$\bullet$" is the dot product operator (see Fig. 2 for l, n, v, r, and p definitions). I_{id} is scaled by Lambert's law: $(l \bullet n)$. These two multiples suggest that the closer the angle of light (l) is to the normal (n) and the closer the angle of the viewer (v) is to the reflection ray (r), the more pronounced the diffuseness and shininess will be (assuming that k_d and k_s are non-zero). Lastly, the pixel output is $I_i = X * (I_{ia} + I_{id} + I_{is})$, where X is a dimming factor depending on the Euclidean distance ($\sqrt{(l-p) * (l-p)}$) of the light vector to point 'p' (Pollard). Often times a global ambient lighting constant (similar to a bias term), I_a adds to this pixel intensity output, I_i, for visibility (Hilton).

Fig. 2: The Phong Reflection Equation



$$I = \frac{1}{a + bq + cq^2}(k_d L_d (\mathbf{l} \cdot \mathbf{n}) + k_s L_s (\mathbf{r} \cdot \mathbf{v})^{\alpha}) + k_a L_a$$

l = vector from light   r = l reflected about n
n = surface normal   v = vector to viewer

Note: l, n, r, and v are unit vectors. a, b, and c are predetermined constant values of the chosen dimming factor (square law) and q is the Euclidean distance of l *(Pollard).*

1.2 Smoothing and Interpolation

While Phong Lighting is an approximation, its calculations are still computationally expensive when repeated many times. Smooth shading takes advantage of objects' tessellations, or polygon subdivisions, and runs the phong lighting algorithm on a few key points (the vertices). These values then interpolate over faces (in this case, we use triangles).

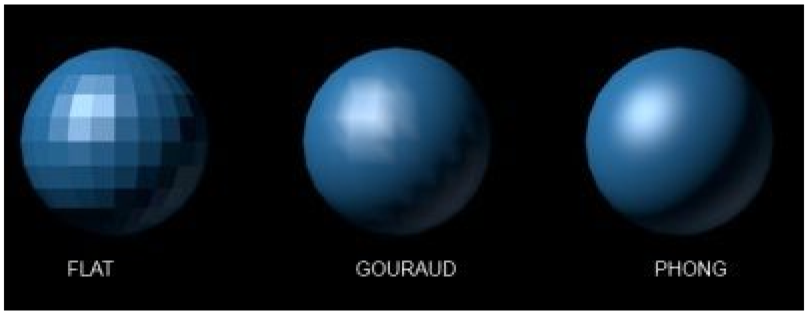
The first approach, Gouraud Shading, uses Barycentric coordinates to interpolate the colors of a triangle's three vertices. Each vertices' normal is the average of the normals of adjacent faces:

$$normal_{vertex} = \sum_{i=0}^{\#Faces-1} normal_i$$

Next, the three Phong Lighting colors of the vertices C=[C_0, C_1, C_2] are calculated, along with the current pixel's barycentric coordinates B=[B_0, B_1, B_2] (see Appendix 4.2). As a result, the pixel's color is the sum of the B.*C (Thorne).

Phong's approach is similar, but removes even more evidence of unwanted edges. This "Mach Band Effect" (see Fig. 3) is still apparent in Gouraud Shading even with smaller polygon sizes (Phong).

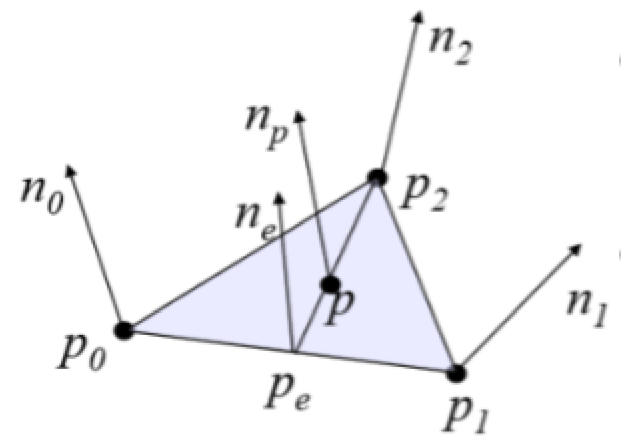
Fig. 3: A comparison between the two smoothing methods and flat shading



(Gouveia)

Rather than interpolating colors, Phong Interpolation interpolates the normals. Assuming a triangle $p_0p_1p_2$ and the target point p, we linearly interpolate on the line p_0p_1 to find p_e such that line p_ep_2 cuts through p (see Appendix 4.3). Next, we linearly interpolate along the line p_ep_2 to find p. In both calculations, the normals of the vertices are also interpolated. As a result, p's normal, n_p, can be used in Phong Lighting to calculate p's color. Unlike Gouraud Shading, Phong Shading does not directly interpolate the output (a point's color). Rather Phong shading takes extra computations to interpret the interpolated normals into a visible RGB color.

Fig. 4: Phong Shading



(1) Interpolate along edge($p_0 p_1$):

$$p_e(\alpha) = \alpha p_0 + (1-\alpha) p_1$$
$$n_e(\alpha) = \alpha n_0 + (1-\alpha) n_1$$
$$0 \le \alpha \le 1$$

(2) Interpolate along line ($p_e p_2$):

$$p(\alpha,\beta) = \beta(\alpha p_0 + (1-\alpha) p_1) + (1-\beta) p_2$$
$$n_p(\alpha,\beta) = \beta(\alpha n_0 + (1-\alpha) n_1) + (1-\beta) n_2$$
$$0 \le \alpha, \beta \le 1$$

(Hilton)

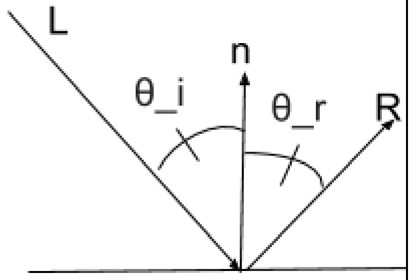
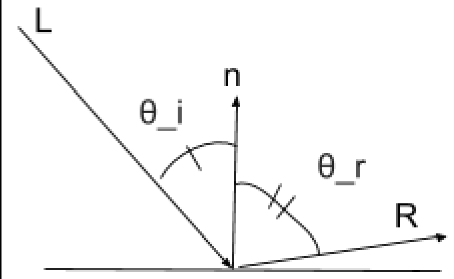
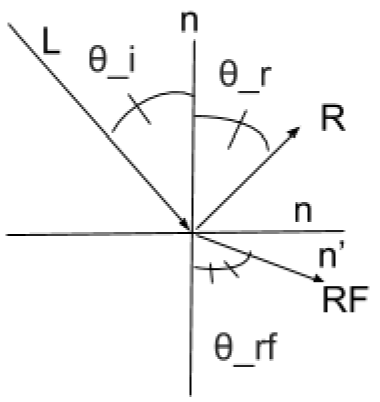
2 Modern Shading

2.1 Ray Tracing

While Phong Lighting excels at diffuse materials, it approximates specular surfaces and ignores the refraction within translucent surfaces. Ray-Tracing tries to solve this issue with a direct approach. While method excels at complex light interactions, diffuse materials are approximately unchanged from Phong Lighting (Hilton).

Surprisingly, Ray-Tracing existed before Phong Shading in IBM researcher Arthur Appel's 1968 paper "Some techniques for shading machine renderings of solids." The paper introduced the idea of rendering images with rays cutting through the monitor plane and reflecting off of surfaces (Appel). While this method seems identical to Phong Lighting, the early Ray-Tracing algorithm was unique because of its ability to create clear, defined shadows. In 1980, Turner Whitted from Bell Laboratories proposed an improvement in his paper "An improved Illumination Model for Shaded Display." The research built off of Appel's approach and included the three fundamental light-surface interactions (see Fig. 5). In addition, Whitted used multiple ray reflections/refractions to create what many now call "global illumination" (Whitted). All of these methods only account for rays coming out for the camera (the origin) and therefore only render what hits the user's "eye." Nonetheless, ray-tracing did not gain popularity until today because it requires many samples of the same image to converge to the ground truth. In fact, the fastest render time of Whitted's test was still 44 minutes.

Fig. 5: The Three Fundamental Types of Light-Surface Interaction (Hilton)

Specular/"Shiny"	Diffuse/"Matte"	Dielectric/"Translucent"
 θ_i equals θ_r.	 θ_r can be any angle in the range $n \pm \pi$. Lambertian or "perfectly diffuse" surfaces have random θ_r values in this range.	 θ_rf follows Snell's law where nsin(θ_i)=n'sin(θ_rf) and n and n' are the refractive indices of the medium above and below the horizontal surface plane ("Snell's law"). θ_i equals θ_r.

Note: L=light ray, θ_i=angle of incidence, n=surface normal, R=reflected ray, θ_r=angle of reflection, RF=refracted ray, and θ_rf=angle of refraction. All rays have red, green, and blue components (RGB). The magnitude of the sum of the components of R is always less than or equal to the magnitude of the sum of the components of L. This difference in magnitude--and therefore the reflected RGB color--of the surface depends on the attenuation factor.

2.2 Applying the Geometric Algorithm (C++)

Physical geometric simulations like Ray-Tracing are possible on almost all computers today. To demonstrate the algorithm, I used spheres for simplified calculations. In typical applications, triangles or quadrilaterals are Ray-Traced (along with interpolation methods like Phong Shading) because most 3D objects can tessellate or divide into simple polygons. It is important to note that in application, rays are actually two vectors, A and B, where A is the ray's starting point and B is the end point. Rays are then parameterized to find all points on ray AB via the equation A+t*B (t is the free variable). Borrowing from spherical geometry, we can calculate ray-sphere intersection point, P, (if one exists) and then perform a spherical reflection of that point over the plane through the sphere's origin and orthogonal to the ray. Because the sphere is not necessarily a unit sphere centered at the origin, we must use a general form (Shirley). The proof is:

$$1.\ \ (P - C) \bullet (P - C) = R^2$$

where C is the sphere's center, R is the sphere's radius, and • is the dot product operator

$$2.\ \ P = A + t * B$$

P is on the ray

$$3.\ \ (A + t * B - C) \bullet (A + t * B - C) = R^2$$

$$4.\ \ (B \bullet B)t^2 + 2t(B \bullet (A - C)) + ((A - C) \bullet (A - C)) = R^2$$

Substitution and Dot product distribution

$$5.\ \ D = b^2 - 4ac$$

D < 0: 0 intersections, don't shade

D=0 or D>0: 1 or 2 intersections, calculate shading

Where D is the discriminant, $b = 2 * B \bullet (A - C)$, $a = (B \bullet B)$, *and* $c = (A - C) \bullet (A - C)$

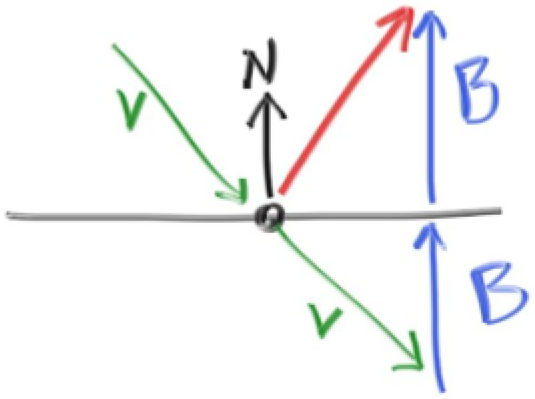
$$6.\ \ t = (-b \pm D)/(2a)$$

Applying the quadratic formula if D is not less than 0

Now that t is solved for, the normals of the circle are simply the unit vector of P(t)-C (surface normals of circles are collinear to the radius, intersect point P, and are of length 1). Combining this intersection calculation with ray tracing gives us the following pseudo-code:

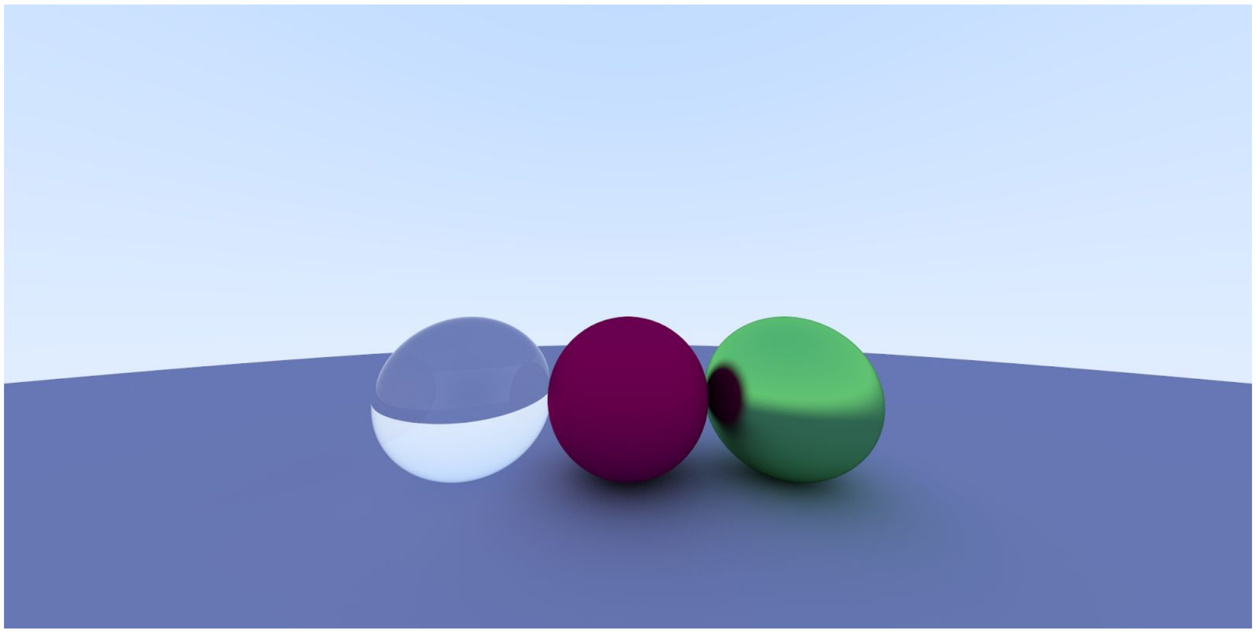
```
RayTrace(ray r, Scene, depth): //returns a pixel color in RGB
     t = INFINITY
     For Each Object O in Scene:
          If r hits O at t′ and t′ < t:
               t=t′
     If t<INFINITY and depth < preset_depth_constant:
          If (surface is specular):
               ray r′ = ray from intersection point to
               v - 2*n*(v•n) //see Fig. 6
               Return attenuation*RayTrace(r′ , Scene, depth+1)
          Else If (surface is diffuse):
               ray r′ = ray from intersection point to
               (intersection point + normal + random direction
               vector) //the random direction vector should be
               chosen so that r′ does not intersect the already
               Intersected sphere/surface
               Return attenuation*RayTrace(r′, Scene, depth+1)
          Else: //refraction where r_i is the refractive index
          //the mix of reflection and refraction may change due
          the specific material of the surface
               If ray starts in the medium with lower r_i:
                    Use Snell′s Law
               Else:
                    Perform specular surface calculation
     Else:
          Return RGB color desired for background

For Each Pixel:
     Find r where r is the current ray associated with that
pixel
     Output = (0,0,0)
     For N where N is the number of samples: //″super sampling″
          r′ = r + small_error
          Output = Output + RayTrace (r′, Scene)
     Return Output/N
```

Fig. 6: A Drawing of the Specular Surface Vector Calculation



*Subtracting 2*n*(v • n) from v gives us the vector reflected over n and pointing away from the surface (Shirley)*

Fig. 7: Output Pixel Array of Ray-Tracing using Circles and 1000 Samples (see Appendix 4.4)



Note: The sphere materials from left to right are translucent, diffuse, and specular. The image is gamma corrected by a factor of 2 (each R, G, and B value is square-rooted). This lighting boost is similar to the I_a constant of Phong shading. Both methods make the scene more visible.

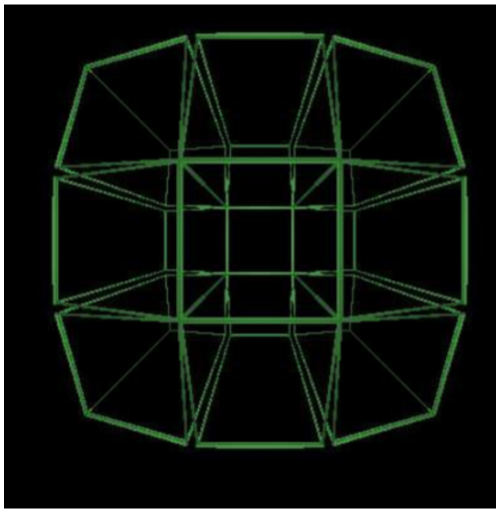
3 The Impact of Accurate Shading

3.1 Industry and Research

The accuracy of Ray-Tracing would not be significant if it were not for the industries and fields that use it. The field of architecture uses lighting for both artistic and practical purposes. Lighting often contributes to the overall feel of a room. With the increasing popularity of glass (dielectric) and reflective (specular) surfaces in modern architecture, Ray-Tracing allows architects to accurately visualize rooms before they commit to building them (ShannonB). Ray-Tracing also produces shadows that are well-defined and visible so that they can be removed or reshaped. In addition, Ray-Tracing helps solve problems involving functionality. Tracing multiple reflections and refractions gives a realistic view a light coverage so that designers can plan spaces that are appropriately lit while also being energy efficient ("Importance of Architectural Lighting").

Ray-Tracing is also an emerging method of visualizing geometry itself. Because of its simple and flexible nature, the algorithm can produce images of non-Euclidean Geometries, with some modifications (i.e. changing distances and angles to fit a geometric space). The faculty of mathematics at the University of Belgrade applied this idea to hyperbolic geometry (Ajdin). Rather than using standard 2D models like Poincare's Ball, researchers used "hyperbolic Ray Tracing" to create 3D models of hyperbolic space (see Fig. 8).
Because Ray-Tracing applies well to non-Euclidean geometries, it can visualize relativistic physics ("Relativistic and non euclidean space"). MIT Game Lab created the "Open Relativity" toolkit using a modified Ray-Tracing algorithm. The 3D kit visualizes how light and non-visible wavelengths deform near light speed. Like "Open Relativity," many Ray-Tracing applications have ties to gaming and entertainment. Other than helping researchers, these interactive implementations also act as powerful educational tools that advance today's researchers and students.

Fig. 8: 3D Hyperbolic Space Tessellated with Identical Cubes



Wireframe cubes show how hyperbolic space distort shape as position from the origin increases. (Ajdin)

4 Appendix

4.1 Element-wise Multiplication

“.*” means element-wise multiplication where:
Given A = [$a_0, a_1 ... a_n$] and B = [$b_0, b_1 ... b_n$] ,
A.*B = [$a_0 b_0, a_1 b_1 ... a_n b_n$]

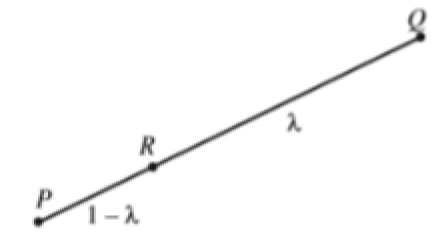
4.2 Barycentric Coordinates

Given a point p = (x,y) and a triangle with with points a = (a_x, a_y), b = (b_x, b_y), and c = (c_x, c_y), the barycentric coordinates of p are B=[B_0, B_1, B_2] where:
x = $B_0 a_x + B_1 b_x + B_2 c_x$,
y = $B_0 a_y + B_1 b_y + B_2 c_y$,
and $B_0 + B_1 + B_2 = 1$.
For the purposes of this paper, we assume that p is within the bounds of triangle abc.

4.3 Vector Form of the Equation of a Line/Linear Interpolation

r = λp+(1−λ)q where:
r is on line pq and divides the vectors p and q in the ratio (1−λ) : λ

Fig. 9: Ratios on a Line



(Brannan)

4.4 Link to Code

Repository to code (outputs in PPM image format):
github.com/thezhe/RayTracingResearch

5 Works Cited

Ajdin, Boris, et al. "Ray Tracing in Poincaré's ball model of hyperbolic space." Faculty of Mathematics, University of Belgrade, Belgrade Serbia. citeseerx.ist.psu.edu/viewdoc/download;jsessionid=F45F22A96459065B2FA 4A8910F93E87E?doi=10.1.1.183.922&rep=rep1&type=pdf.

Appel, Arthur. "Some techniques for shading machine renderings of solids." 1968. graphics.stanford.edu/courses/Appel.pdf.

Brannan, David A., et al. *Geometry*. 2nd Edition. Cambridge University Press, 2012.

Gouveia, David. Comment on "Creating smooth lighting transitions using tiles in HTML5/JavaScript game." Stack Exchange, 27 Mar 2012, 4:43 p.m., gamedev.stackexchange.com/a/26260.

Hilton, Adrian. "Shading." University of Surrey, Guildford, England. Lecture. info.ee.surrey.ac.uk/Teaching/Courses/CGI/lectures_pdf/lecture4.pdf.

"The Importance of Architectural Lighting." *TCP*, www.tcpi.com/importance-architectural-lighting. Accessed 7 May 2019.

Phong, Bui Tuong. "Illumination for Computer Generated Pictures." Edited by W. Newmann, University of Utah, 1975, Salt Lake City, Utah. http://users.cs.northwestern.edu/~ago820/cs395/Papers/Phong_1975.pdf.

Pollard, Nancy. "Lighting and Shading." Carnegie Mellon University, Pittsburgh, PA. Lecture. graphics.cs.cmu.edu/nsp/course/15-462/Spring04/slides/07-lighting.pdf.

"Ray tracing (graphics)." *Wikipedia*, en.wikipedia.org/wiki/Ray_tracing_(graphics). Accessed 6 May 2019.

"Relativistic and non euclidean space rendering." *WordPress*, 9 December 2014, lousodrome.net/blog/light/2014/12/09/relativistic-and-non-euclidean-space-rendering.

ShannonB. "How Are Computers Used in Architecture?" *Career Trend*, 27 December 2018, careertrend.com/about-6601447-computers-used-architecture-.html. Accessed 7 May 2019.

Shirley, Peter. *Ray Tracing in One Weekend*. Peter Shirley, 2018. www.realtimerendering.com/raytracing/Ray%20Tracing%20in%20a%20Weekend.pdf. Accessed 7 May 2019.

"Snell's law." *Wikipedia*, en.wikipedia.org/wiki/Snell%27s_law. Accessed 7 May 2019.

Thorne, Tom. "Computer Graphics 5 - Illumination and Shading." University of Edinburgh, Edinburgh, Scotland. Lecture. www.inf.ed.ac.uk/teaching/courses/cg/ lectures/slides5.pdf.

Whitted, Turner. "An Improved Illumination Model for Shaded Display." *Communications of the ACM*, vol. 23, no. 6, June 1980. *ACM Digital Library*, doi:10.1145/358876.358882. dl.acm.org/citation.cfm?id=358882. Accessed 7 May 2019.