Lighting

CSE606: Computer Graphics

Jaya Sreevalsan Nair, IIIT Bangalore

February 17-19-24, 2025

Color Theory: A Brief Detour

RGB Model: Introduction

Spectral sensitivities to chromatic light: Normalized spectral sensitivity of human cone cells of short- and long-wavelengths in the electromagnetic visible spectrum.

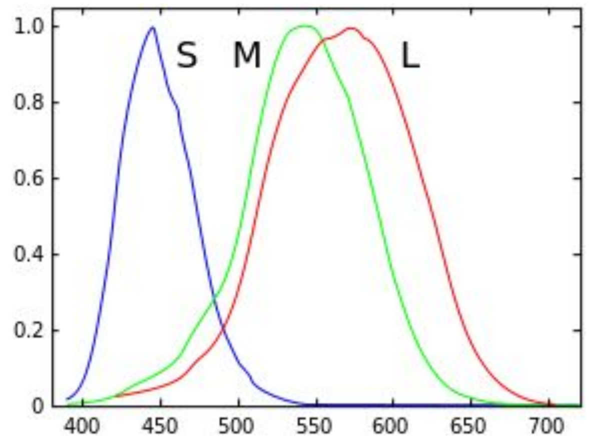


Image courtesy: Wikipedia

RGB Model: Introduction

Spectral sensitivities to chromatic light: Normalized spectral sensitivity of human cone cells of short- and long-wavelengths in the electromagnetic visible spectrum.

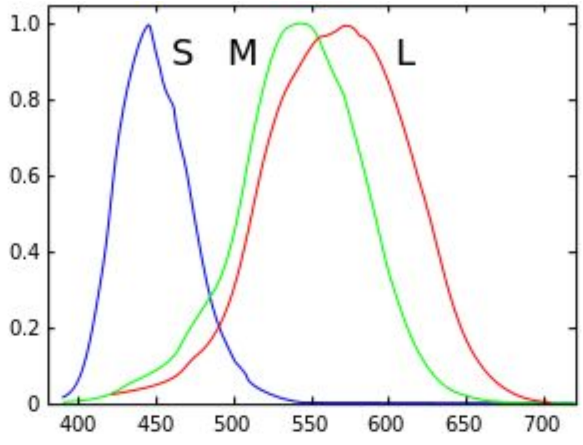


Image courtesy: Wikipedia

RGB model is a *convenient* model for responsivities for SML-wavelengths.

Cones in the retina enable chromatic vision, and **rods** enable achromatic vision (also broadly corresponds to day and night vision, respectively).

RGB Model: Introduction

Spectral sensitivities to chromatic light: Normalized spectral sensitivity of human cone cells of short- and long-wavelengths in the electromagnetic visible spectrum.

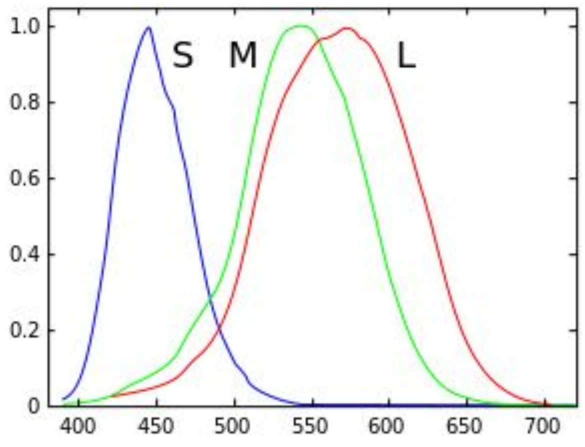


Image courtesy: Wikipedia

RGB model is a *convenient* model for responsivities for SML-wavelengths.

Cones in the retina enable chromatic vision, and **rods** enable achromatic vision (also broadly corresponds to day and night vision, respectively).

Presence of 3 cone types suggests: 3 parameters corresponding to stimulus values to the 3 cones, describe all colors. These stimuli values are called ***tristimulus values***.

Two light sources with different spectral distributions can appear to be the same color. Such pairs are called ***metamers***.

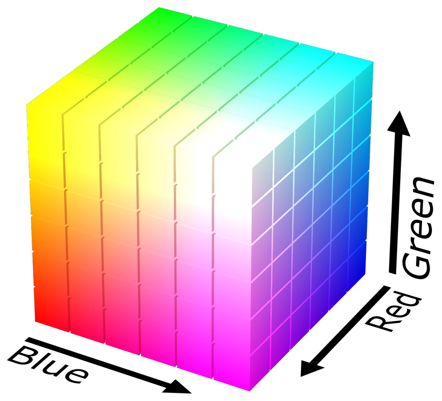
The Legacy of the RGB Model

Why is RGB considered the ubiquitous color model?

The answer lies partly in capabilities of human visual system, and partly in hardware that can regenerate what is perceivable (by the human visual system).

e.g. the purple phosphor is impossible to create, even though human eye perceives purple as distinct from red and blue.

[Source: http://www-personal.umich.edu/~jpboyd/eng403_chap12_colormodels.pdf]



RGB Color Cube Model,
which gives the hexadecimal representation of the 3D color vector.
Image courtesy: wikipedia

CIE 1931 Chromaticity Diagram

Mathematical model of the visible spectrum for the human eye.

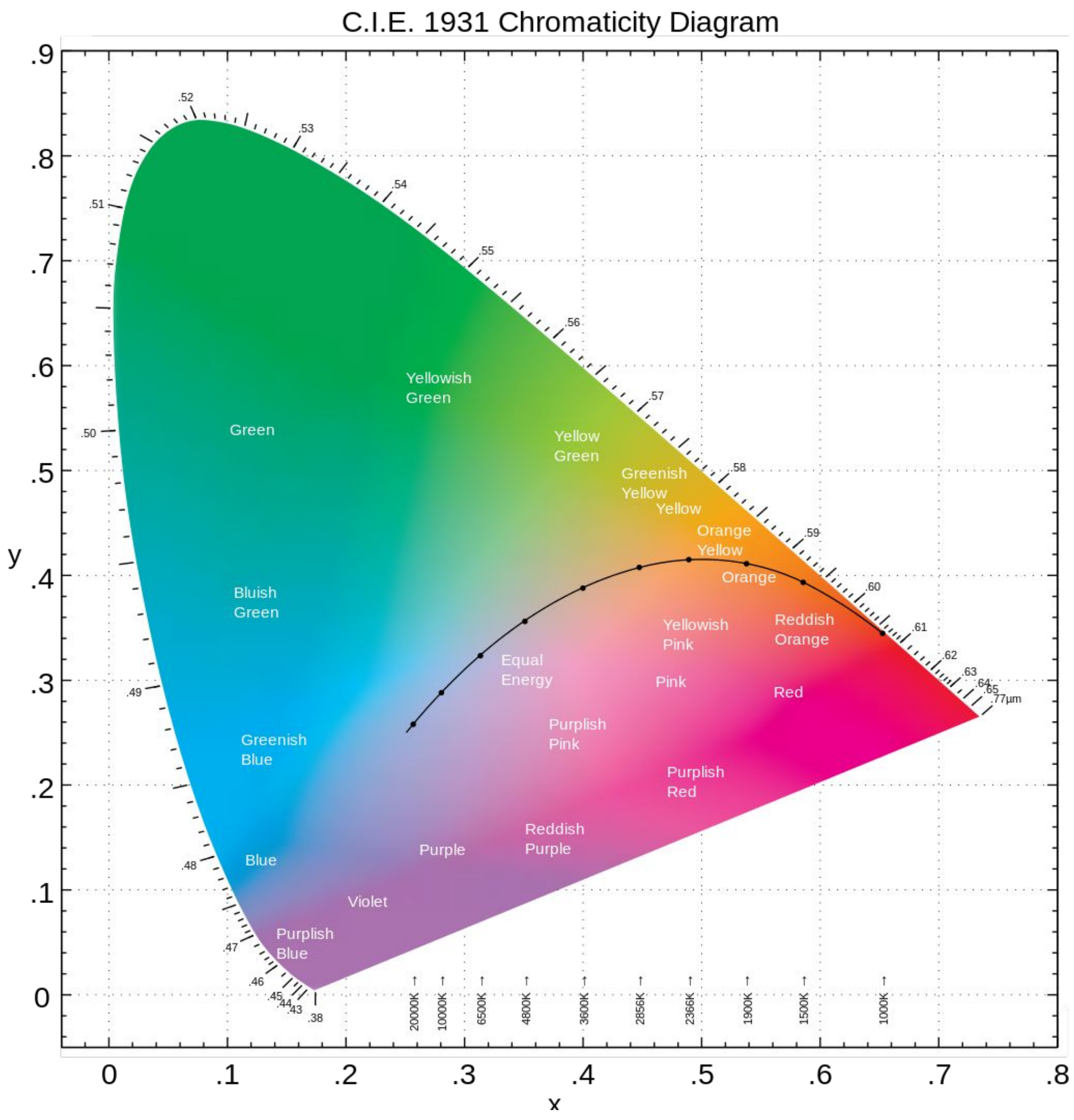
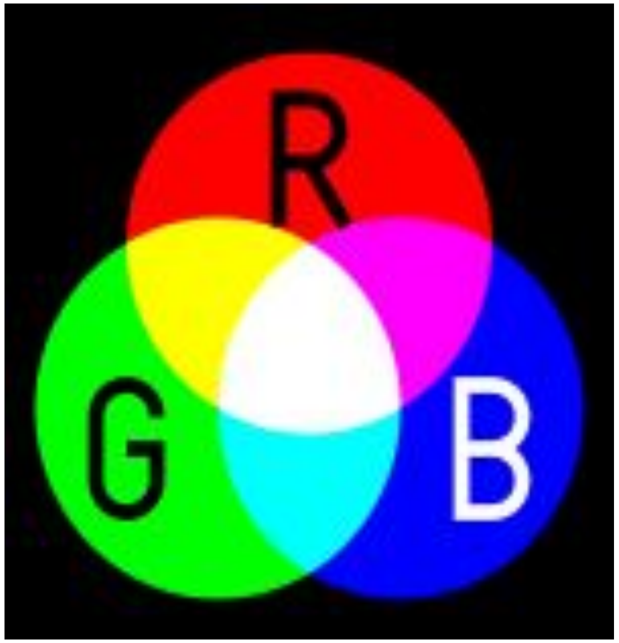


Image courtesy: Wikipedia

Color Models

Additive (RGB): computer graphics

Subtractive (CMYK): printing, painting



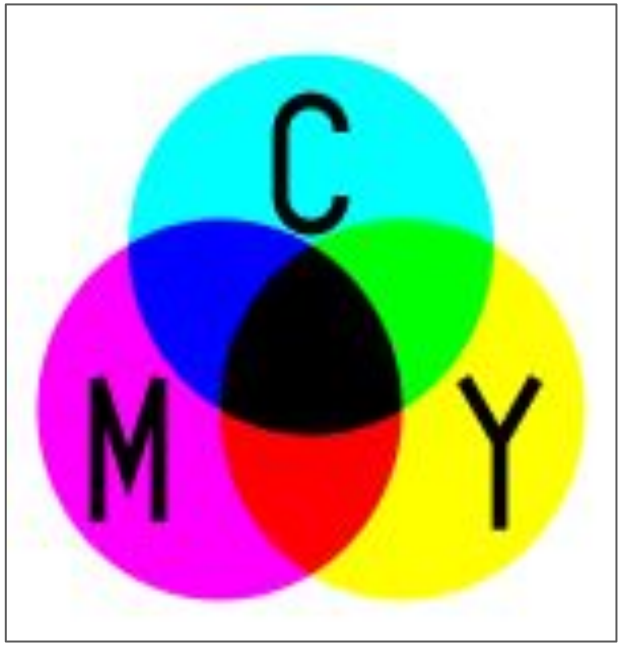


Image courtesy: Wikipedia

Hue-related Color Models

HSV (Hue-Saturation-Value)

HSL (Hue-Saturation-Lightness)

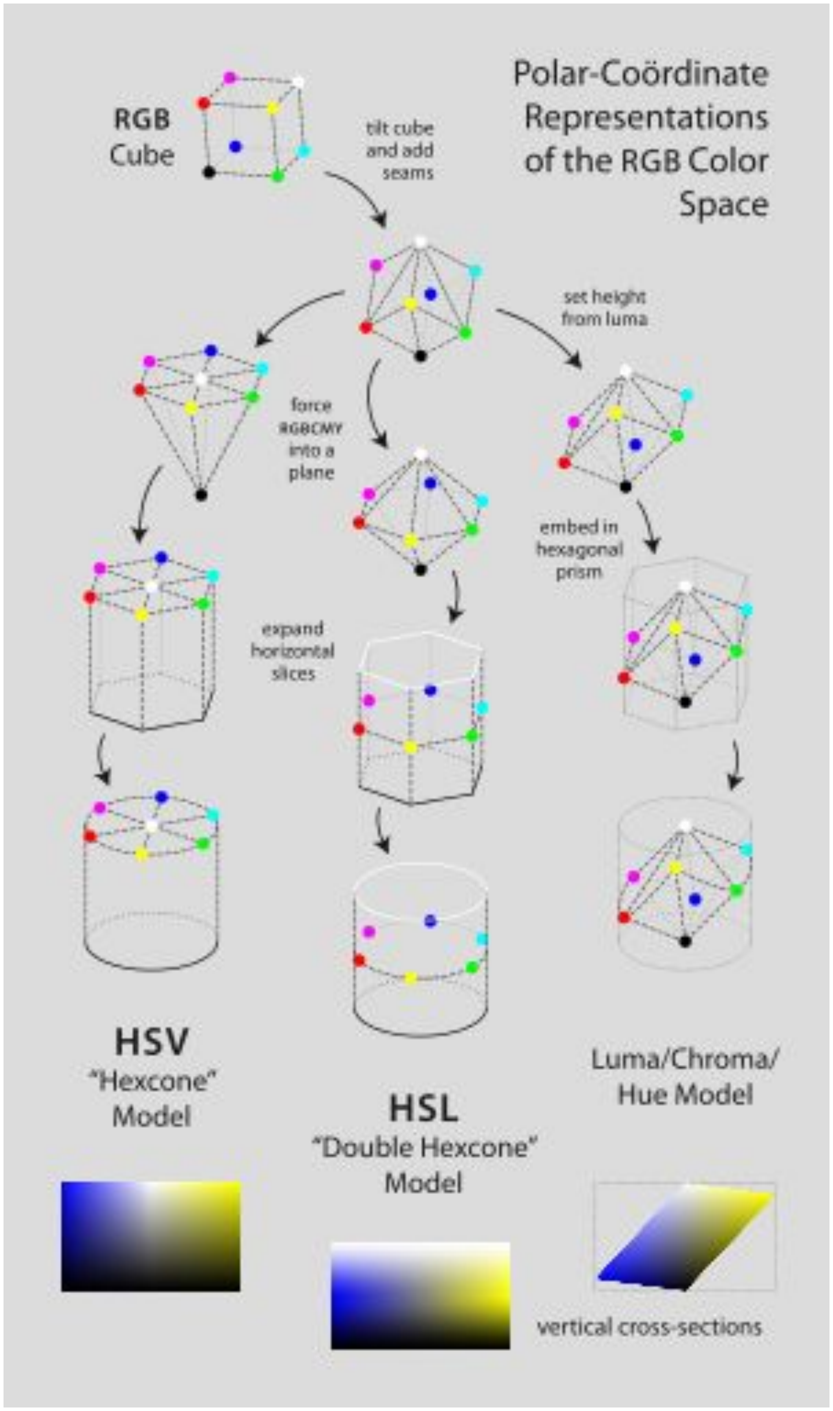


Image courtesy: Wikipedia

Lighting: Introduction

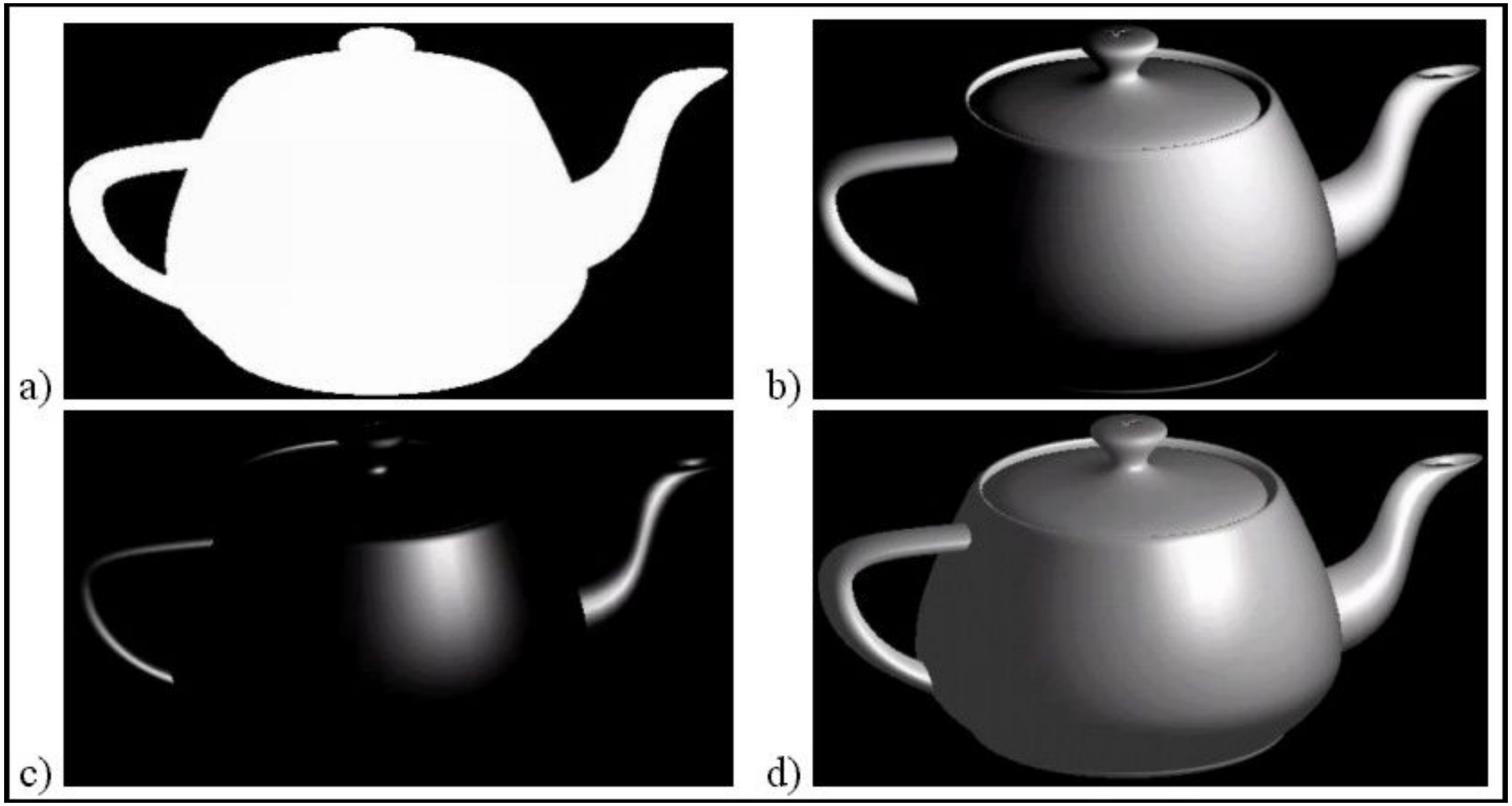
Light and Matter

Most general approach is based on physics of light.

- Broadly, on the principle of conservation of energy.
- A surface can emit light (self-emission) or reflect light falling on it (illumination).
- Color on an object is due to multiple interactions among light surfaces and reflective surfaces – a recursive process.
- Recursive process leads to an integral equation: the **rendering equation**.
    - Its solution gives the shading of all the surfaces in a scene.

Need for Lighting

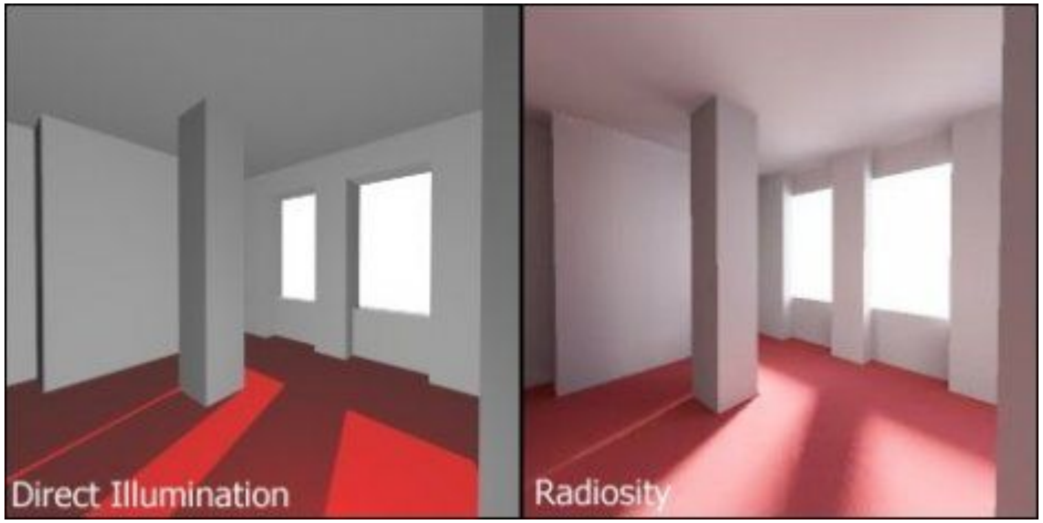
(a) Ambient lighting [or no lighting], (b) diffuse lighting, (c) specular lighting, and (d) combined light.

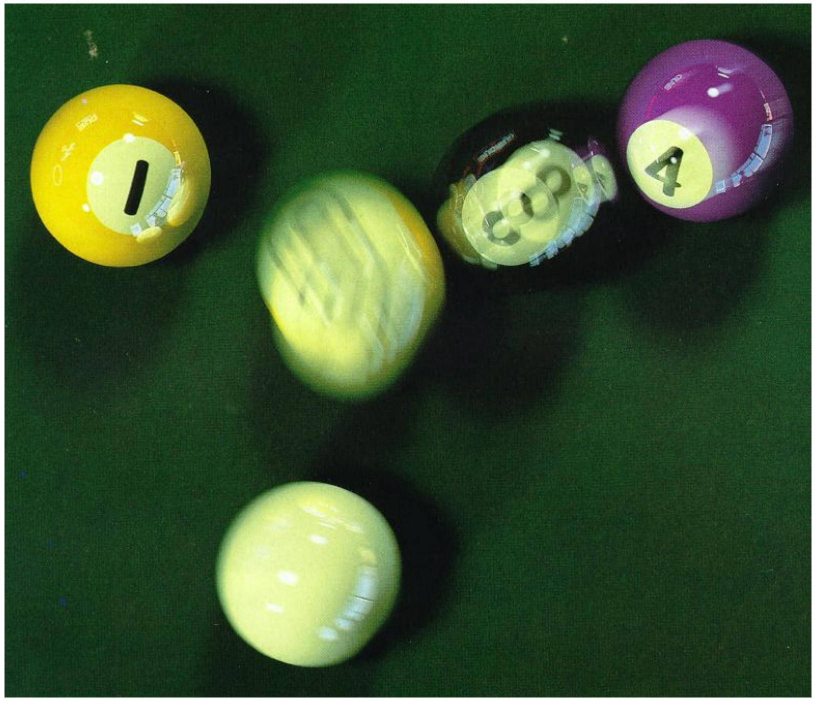


http://www.naturewizard.at/tutorial0107.html

Examples

(Left) Differences in illumination when soft shadows are rendered using radiosity mapping. (Right) Motion blur using ray tracing.





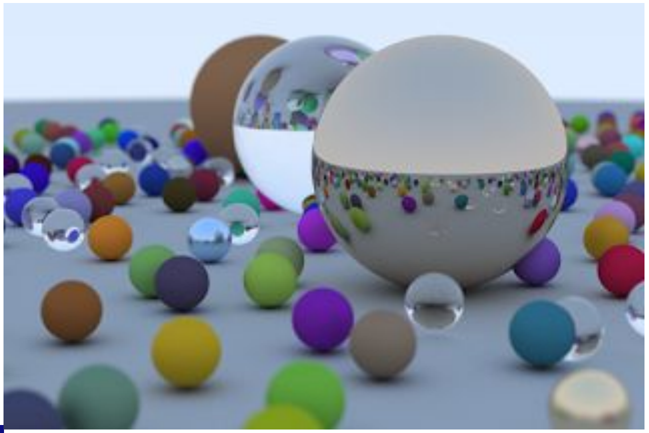
https://en.wikipedia.org/wiki/Radiosity_(computer_graphics)

http://graphics.pixar.com/library/DistributedRayTracing/index.html

Examples

(Bottom-left) Cornell dataset without color bleeding [L], and with color bleeding using radiosity mapping [R].

(Top-right) Ray Tracing with Peter Shirley's tutorial.



http://psgraphics.blogspot.in/2016/01/new-mini-book-ray-tracing-in-one-weekend.html

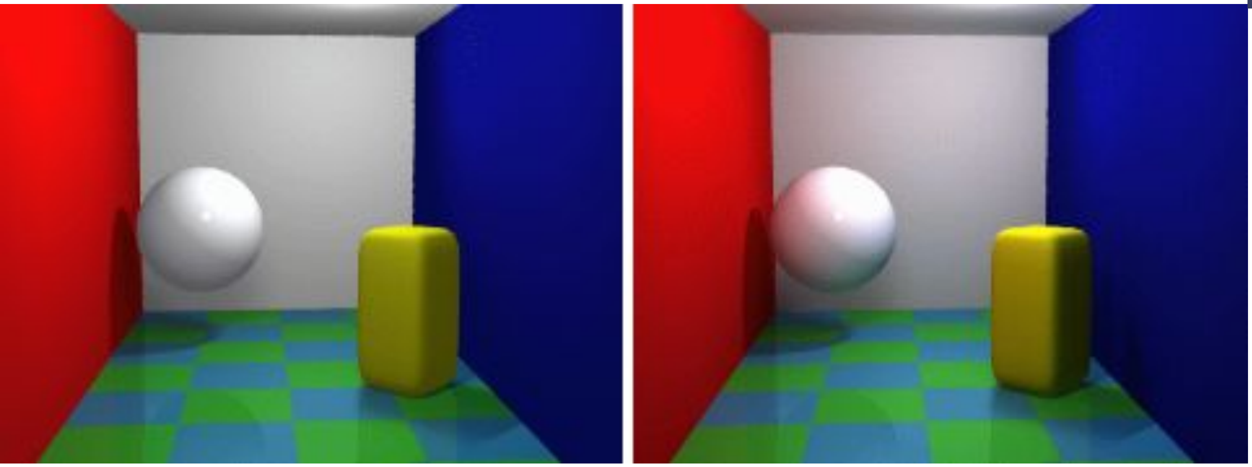
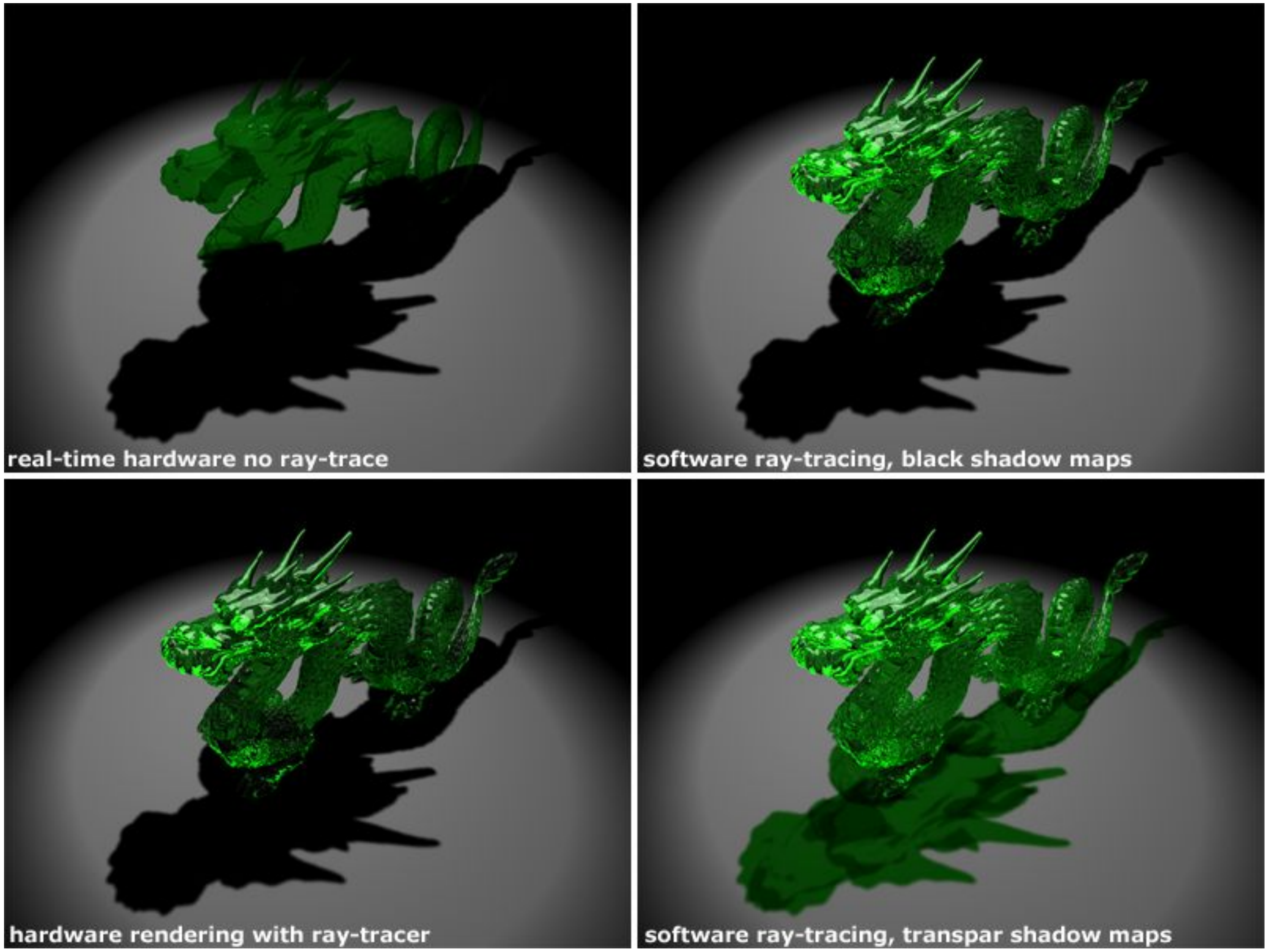


Image courtesy:lhttp://teaching.csse.uwa.edu.au/units/CITS4241/Handouts/Lecture19.html

Examples

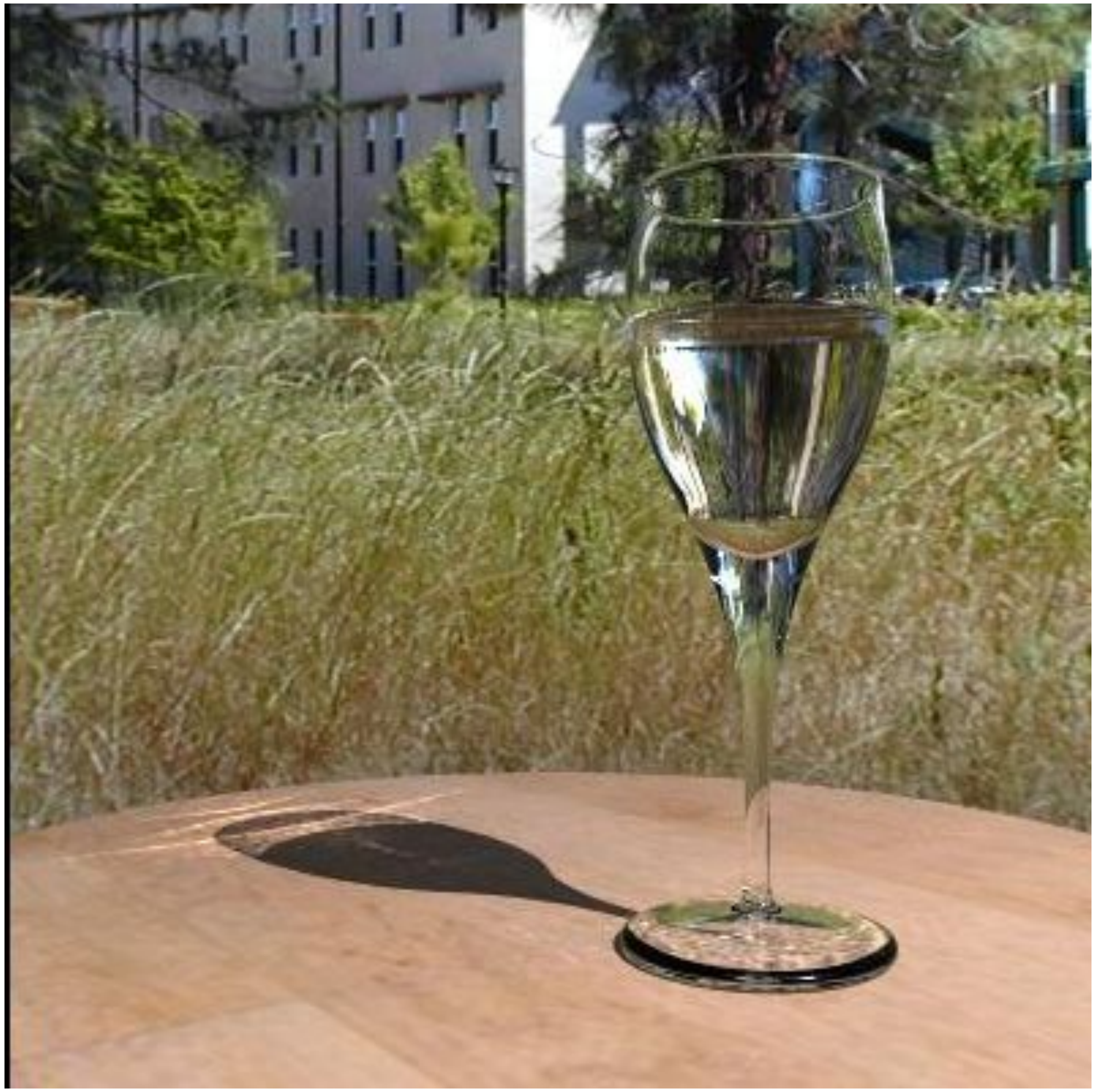
Shadow maps
with transparency


real-time hardware no ray-trace
software ray-tracing, black shadow maps
hardware rendering with ray-tracer
software ray-tracing, transpar shadow maps

http://www.redway3d.com/downloads/public/documentation/bk_re_ray_tracing_options.html

Examples

Caustics using photon mapping.


https://graphics.stanford.edu/courses/cs348b-competition/cs348b-02/brasket_deleon/

Rendering Equation

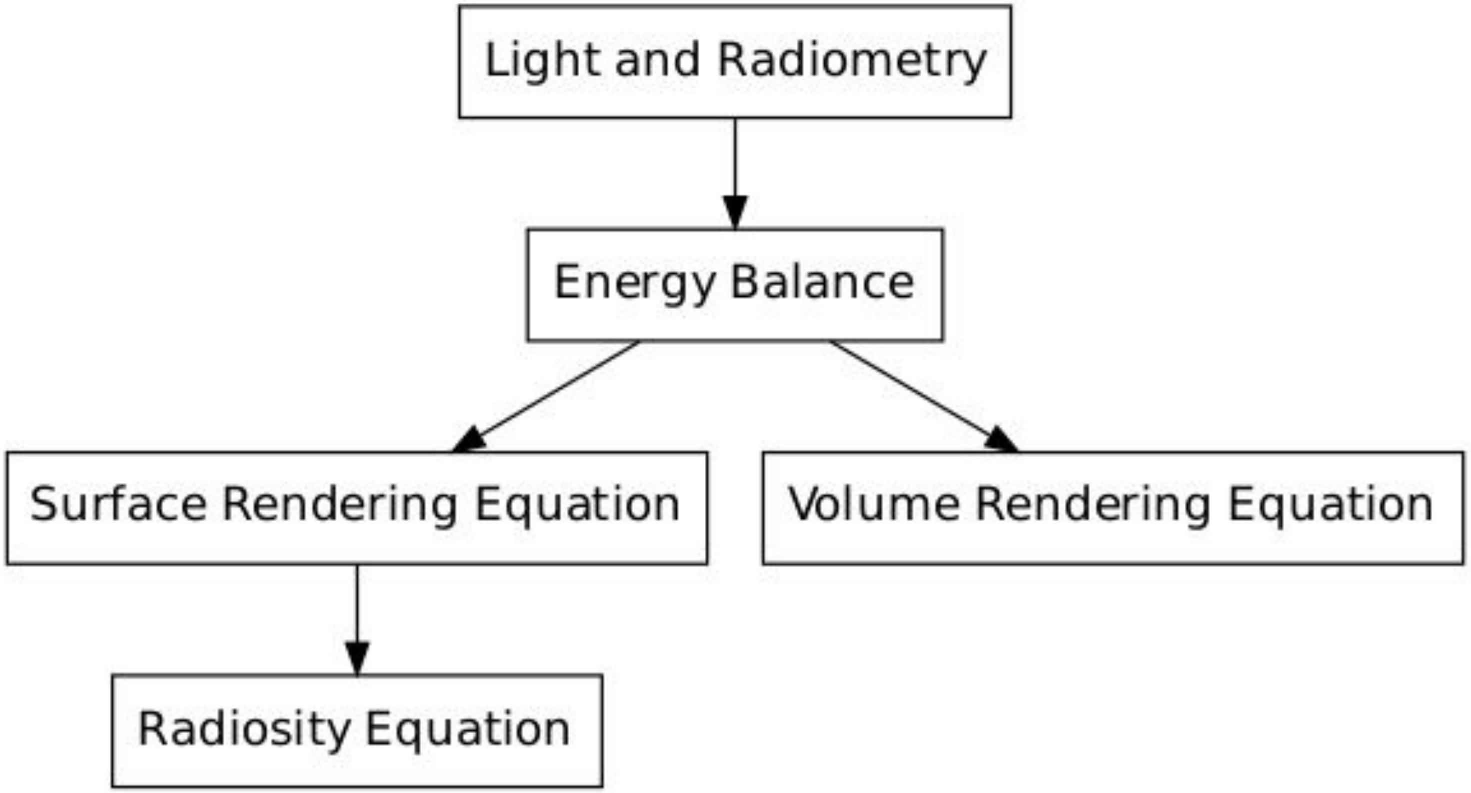
Measuring Light

From Pat Hanrahan’s lectures in CS348B in Spring 2002, Stanford University:

Questions:

1. How is light measured ?
2. How is spatial distribution of light energy described ?
3. How is reflection from a surface characterized ?
4. What are the conditions for equilibrium flow of light in an environment ?

Dependency Diagram of Lighting



Radiometry

From Shirley, Marschner’s “Fundamentals of Computer Graphics”, 3E, 2009.

Collection of photons: amount of energy carried by a photon is $q = hf = \frac{hc}{\lambda}$

How is energy distributed across wavelengths ?

- Partition photons into bins, and histogram them.
- Energy computed in terms of an interval (between λ_1 and λ_2).
- This is called “Spectral Energy”, an intensive quantity represented as Q_λ.

Radiometry

From Shirley, Marschner's "Fundamentals of Computer Graphics", 3E, 2009.

Power: An intensive quantity to describe "Rate of energy production for light sources."

- Measured in steady state, hence rate in terms of time.
- Spectral power: represented as Φ_λ or just Φ.

Irradiance: A density quantity to describe "How much light hits this point ?"

$$H = \frac{\Delta\Phi}{\Delta A} = \frac{1}{\Delta A} \cdot \frac{\Delta q}{\Delta t}$$

Radiometry

From Shirley, Marschner’s “Fundamentals of Computer Graphics”, 3E, 2009.

Radiance: “How much light” from a specific direction ?

$$L = \frac{\Delta H}{\Delta\omega.cos\theta} = \frac{1}{\Delta\omega.cos\theta} \cdot \frac{\Delta q}{\Delta A \Delta t \Delta\lambda}$$

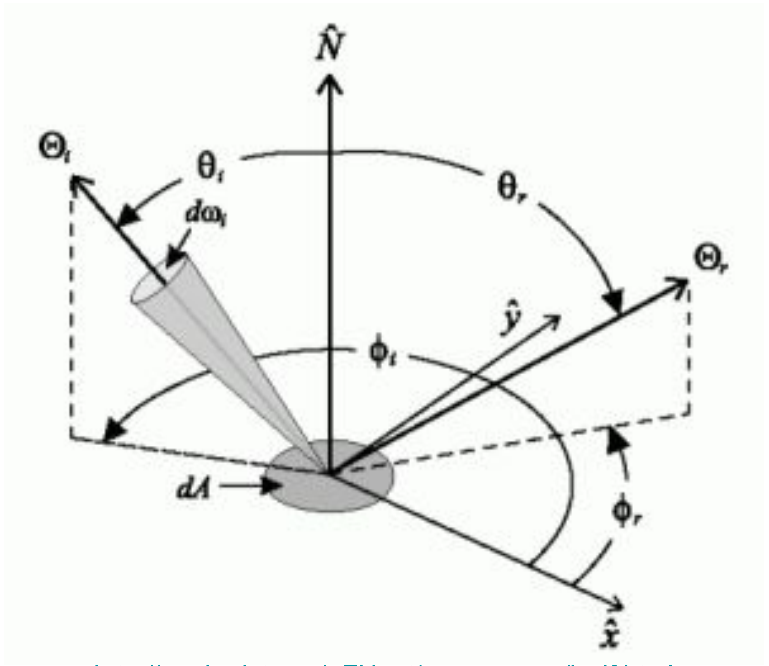
for a baffler with solid angle ω, and the baffler or light making an angle θ with the surface.

BRDF: A function that characterizes how a surface reflects light.

BRDF

BRDF (Bi-directional Reflectance Distribution Function): A single function to describe reflection, refraction and transmission of light incident on a surface. BRDF of a surface is the ratio of reflected radiance to incident irradiance at a particular wavelength.

BRDF is a function map in light reflection geometry



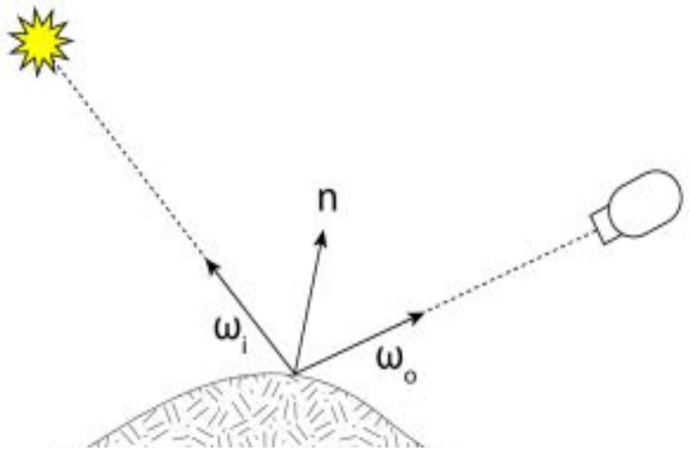
http://math.nist.gov/~FHunt/appearance/brdf.html

BRDF

BRDF is given by:

$$\rho(\omega_i, \omega_o) = \frac{dL_r(\omega_i, \omega_o)}{dH_i(\omega_i)} = \frac{dL_r(\omega_o)}{L_i(\omega_i)\cos\theta_i \, d\omega_i}$$

where L is the radiance, H is the irradiance, and θ_i is the angle made between L_i and the surface normal, **n**; i and r implying incident and reflected, respectively.



ω_i points toward the light source. ω_o points toward the viewer (camera). n is the surface normal. (All unit vectors.)

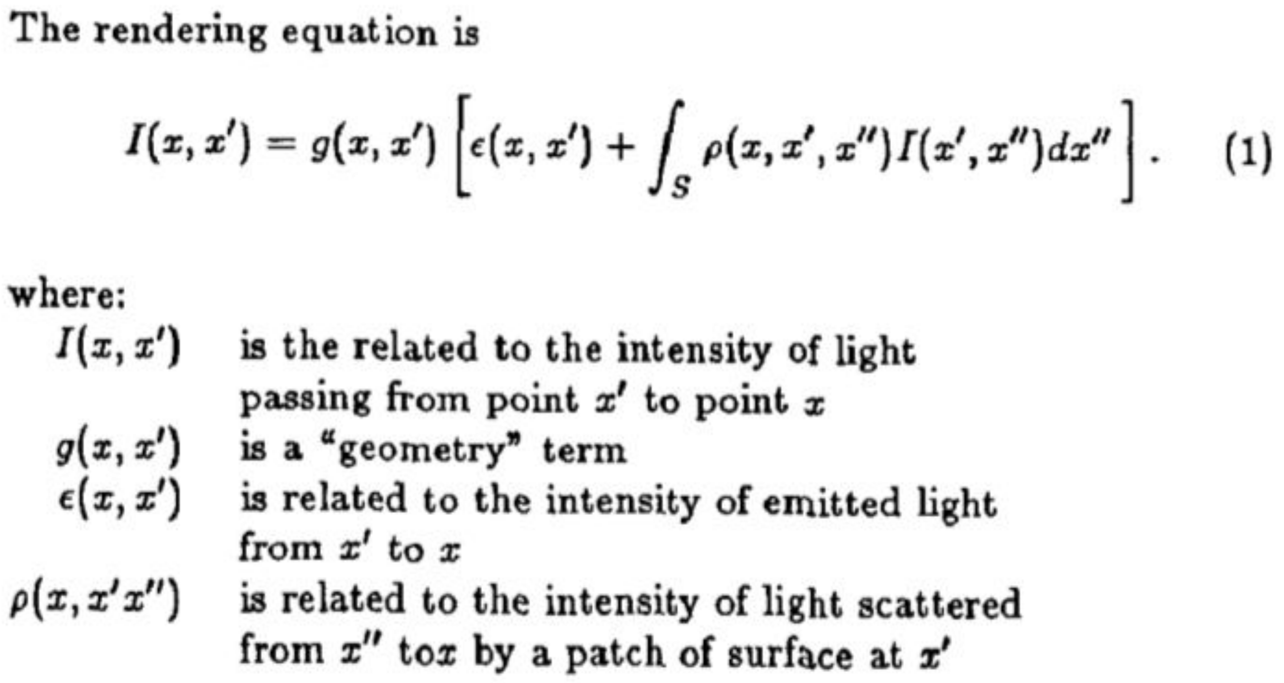
(Image courtesy: Wikimedia Commons.)

Rendering Equation

Surface radiance is given by the equilibrium balance of the flow of light:

$$L_s(\omega_r) = \int_{\forall \omega_i} \rho(\omega_i, \omega_r) L_i(\omega_i) \cos\theta_i \, d\sigma_i$$

Kajiya's original equation:



The rendering equation is

$$I(x, x') = g(x, x') \left[\epsilon(x, x') + \int_S \rho(x, x', x'') I(x', x'') dx'' \right]. \quad (1)$$

where:

$I(x, x')$	is the related to the intensity of light passing from point x' to point x
$g(x, x')$	is a "geometry" term
$\epsilon(x, x')$	is related to the intensity of emitted light from x' to x
$\rho(x, x'x'')$	is related to the intensity of light scattered from x'' to$x$ by a patch of surface at x'

Kajiya, James T. ``The rendering equation.'' In ACM Siggraph Computer Graphics, vol. 20, no. 4, pp. 143-150. ACM, 1986.

Rendering Equation - Solutions

No analytical solution – only numerical approximate solutions.

- Examples of approximate solutions of rendering equation -- Radiosity, Ray-tracing.
- Not fast enough for real-time rendering, but with RTX cards ray-tracing can be done real-time using hardware acceleration.

OpenGL uses Phong lighting model, which is a trade-off between analytical correctness and efficient computation.

Local Illumination Model

Computer Model for Lighting

Source: Self-emitting or self-luminous surfaces.

Lighting process consists of:

1. Modeling light sources in the scene.
2. Building reflection models for various materials in the scene.

Use projection plane and COP to determine colors of pixel:

- Only light rays through COP matter in the final image.
- Hence, start rays from COP and check for each projector through a pixel in the clipping window – gives color of corresponding pixel in FB.
    - Ray casting is more efficient than ray-tracing.

Types of Surfaces based on Light Propagation

Different forms of interactions between incident light and surfaces, characterized by optical properties of the surface.

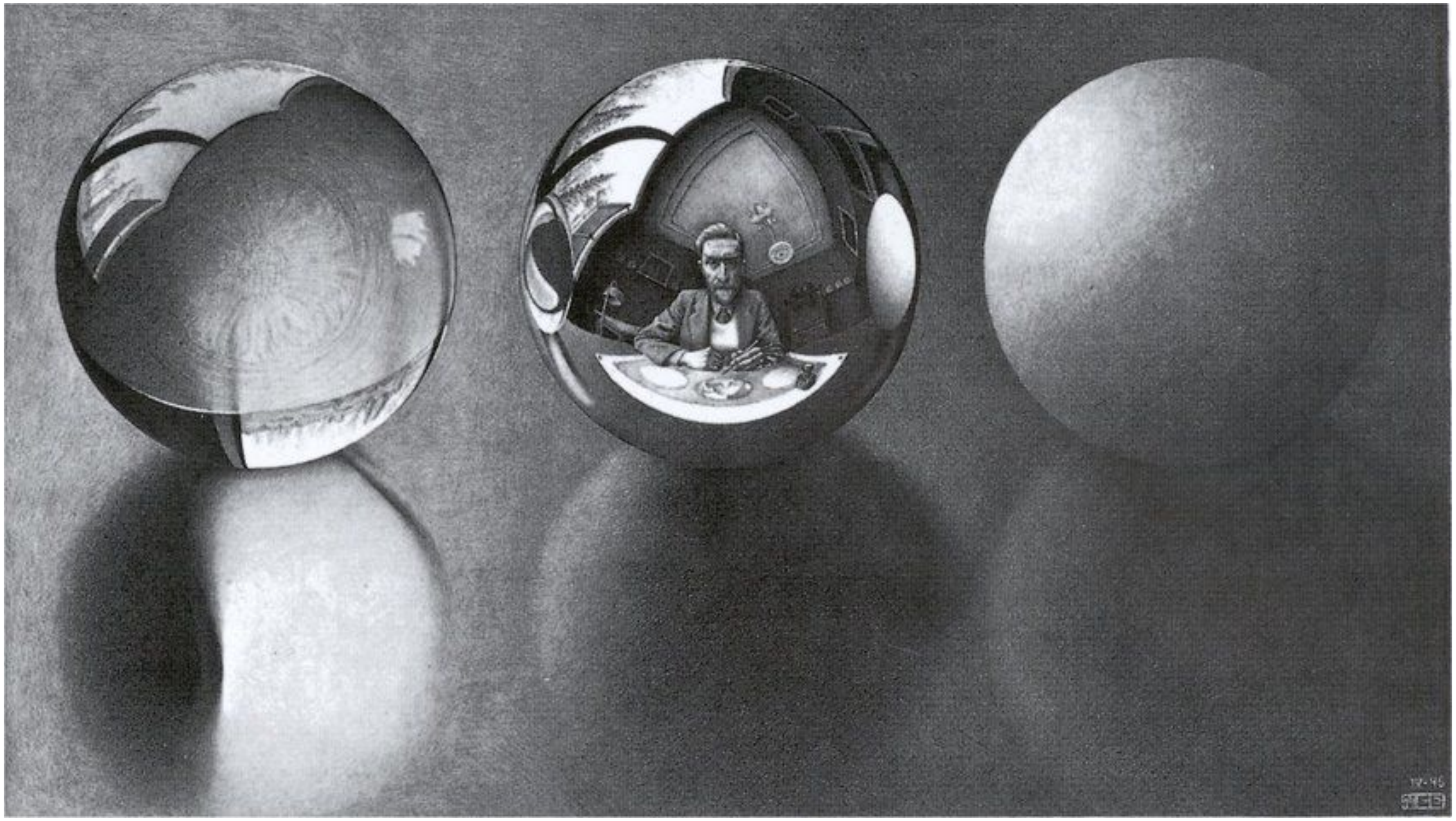
Surface type based on propagation of light:

1. Opaque objects: Absorb some, reflect the rest.
2. Translucent objects: Transmit some, reflect the rest.
3. Transparent objects: Transmit all.

Absorption of light depends on the wavelength of light.

- Shading of objects depends on orientation of the surfaces.
- Rough surfaces look duller; smooth surfaces look shinier.

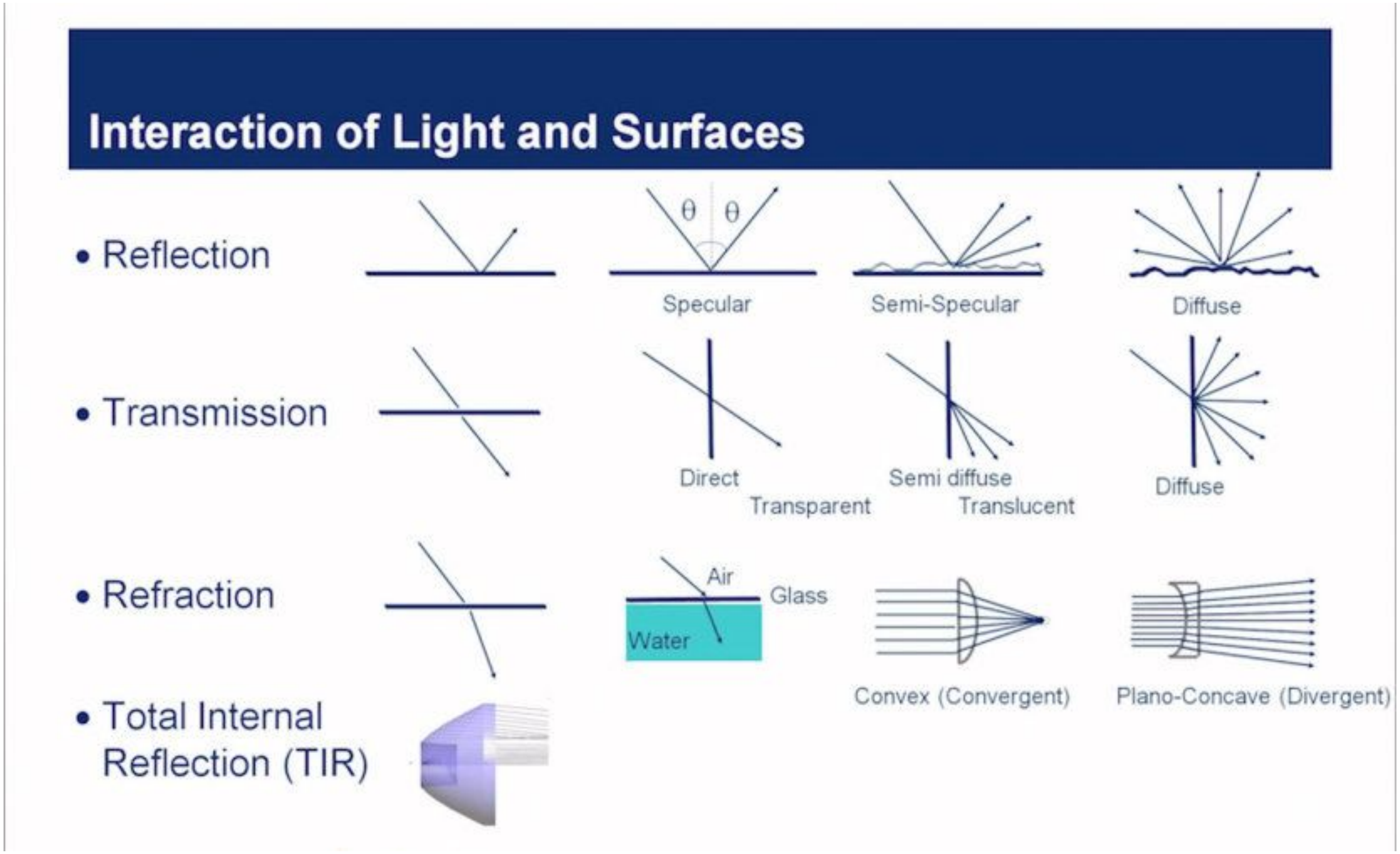
Surface Types



(Left-to-right) Transparent, specular, diffuse surfaces.

“Three Spheres II” by M. C. Escher (1946).

Optical Principles



Interaction of Light and Surfaces

- Reflection
- Transmission
- Refraction
- Total Internal Reflection (TIR)

https://www.osram.jp/os/products/led- general- lighting/tools- and- service/led-fundamentals/optical_principles.jsp

Types of Surface Based on Reflection of Light

Specular Surfaces: Shiny – All reflected/scattered rays emerge close to each other, roughly at the angle of reflection. Mirrors are perfectly specular.

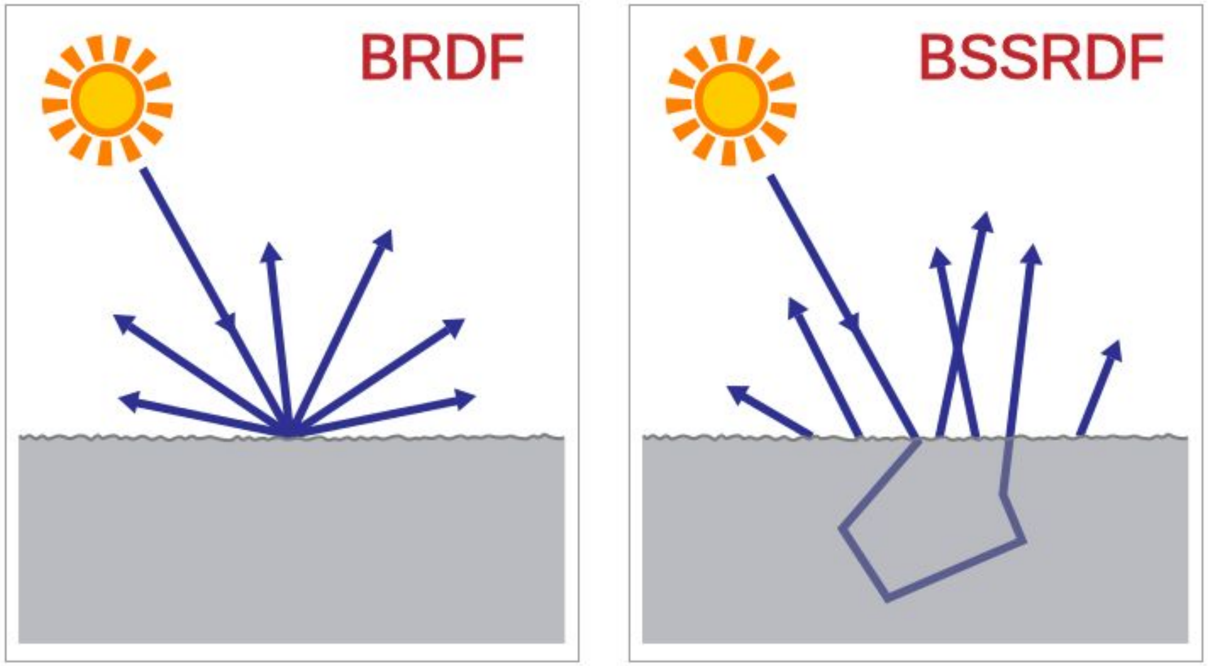
Diffuse Surfaces: Matte – Reflected light scattered in all directions. Perfectly diffuse reflectors appear same at all viewing angles.

Translucent Surfaces: Refractive – Some of light reflected, some transmitted; and some refracted. Glass and water are good refractive surfaces.

Extension to BRDF

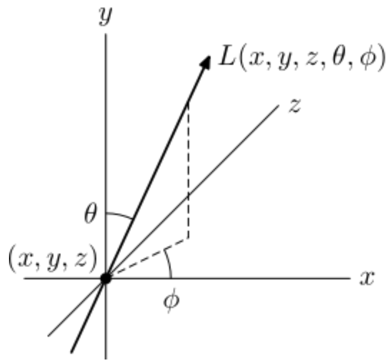
Real surfaces are never perfectly specular nor perfectly diffuse. However, BRDF is complex and computationally intensive.

BRDF is still incomplete – BSSRDF is more complete, inclusive of modeling for subsurface light transport.



More details: http://graphics.stanford.edu/papers/bssrdf/
Image courtesy: Wikipedia

Light Sources in Graphics Systems



In general, in modern graphics system, use simple models for lights:

- whose contributions can be added together.
- which is economical for a pipeline architecture.

Light sources are self-emitting and reflective surfaces; with illumination function I(x , y , z, θ, φ, λ) at wavelength λ.

Contribution of light on surface: Surface integral of I for all emission angles that reach the surface. Also, takes into account the distance between source and surface.

Distributed light sources are difficult to model. Relatively large light sources in a scene are modeled as distributed light sources

Color of Light Sources

Based on three-color theory, light sources are modeled as 3-component sources. $L = (L_r , L_g , L_b)$.

Compute light-material properties separately for 3 components and do a linear combination, per-component.

Usage for computations later: $L \in \{L_r, L_g, L_b\}$.

Basic Types of Light Sources

Ambient Light (L_a): provide uniform illumination, depends on source color. It is the light already present in the scene/environment, e.g. natural light.

Point Sources ($L(p_0)$): An ideal point source is an isotropic emitter; follows inverse-square law for electromagnetic radiation.

Spotlight (L_s): characterized by narrow range of angle for light emission, described as a cone of influence. $L_s = L_s(I_s, \theta, p_s)$, where I_s, θ, and p_s are the axis vector, angle, and apex of the cone.

Distant Source: characterized by parallel rays incident on the surface. It is defined by direction of light.

Light Sources in OpenGL

OpenGL allows a **global ambient contribution**, which lights up objects in scene, by default.

For point source at $\mathbf{p_0}$: $\mathbf{L(p_0) = (L_r(p_0), L_g(p_0), L_b(p_0))}$, intensity of $L(p_0)$ light at **p** is given by**:**

$$\mathbf{L}(\mathbf{p}, \mathbf{p}_0) = \frac{\mathbf{L}(\mathbf{p}_0)}{|\mathbf{p} - \mathbf{p}_0|^2}.$$

Attenuation is the phenomenon by which there is a reduction in the intensity of light, which is caused by distance, in this case.

Distance term is added using the attenuation factor: $\dfrac{1}{a+b.d+c.d^2}$

where a, b, c are coefficients added to soften lighting

Light Sources in OpenGL

Point sources are good to bring in high contrast in lighting objects in a scene. To mitigate high-contrast, one can add ambient light to scene.

For a spotlight source, the intensity at a point is given by an exponential function $(\mathbf{n}.\mathbf{v})^e$, where **n** is the axis of the cone of view; and **v** is the vector joining source and point.

Phong Lighting Model

Phong Lighting Model is the default lighting model in OpenGL

Phong model, which is a local illumination model, supports three types of light-material interactions: **ambient, diffuse and specular.**

- Ambient source color represents interaction of a material with light source present in the environment.
- Specular source color represents desired color of a specular highlight.

Illumination array includes three different types of contributions for three different color channels. Thus,

$$\mathbf{L}_{i(3\times 3)} = \begin{bmatrix} L_{ira} & L_{iga} & L_{iba} \\ L_{ird} & L_{igd} & L_{ibd} \\ L_{irs} & L_{igs} & L_{ibs} \end{bmatrix}.$$

Phong Lighting Model

The corresponding reflection array R would be $\mathbf{R}_{i(3\times3)} = \begin{bmatrix} R_{ira} & R_{iga} & R_{iba} \\ R_{ird} & R_{igd} & R_{ibd} \\ R_{irs} & R_{igs} & R_{ibs} \end{bmatrix}.$

Thus for a light source, the contribution for each color source would be,

$$I_{ik} = L_{ika}R_{ika} + L_{ikd}R_{ikd} + L_{iks}R_{iks} = I_{ika} + I_{ikd} + I_{iks}$$
$$\text{where} k \in \{r, g, b\}$$

For a scene, including all light sources and a global ambient term, intensity for each color using an additive model is:

$$I_k = \sum_i (I_{ika} + I_{ikd} + I_{iks}) + I_{ak}$$

Computations for Phong Lighting Model

- Ambient Reflection
- Diffuse Reflection
- Specular Reflection
- Phong Reflectance Model -- Summing it all up
- Vector Computations: Normal Vector
- Vector Computations: Reflection Vector

Computation: Ambient Light Reflectance

Intensity of ambient light I_a is the same at every point on receiving surface.

Only a fraction (k_a) of light gets reflected – $0 \leq k_a \leq 1$.

- k_a is called **ambient reflection coefficient.**
- $I_a = k_a.L_a$, for individual ambient sources or for global source.
- A surface can have different reflection coefficients for the three different colors – k_{ar} , k_{ag}, k_{ab}.

Computation: Diffuse Light Reflectance

A perfectly diffuse surface scatters reflected light isotropically.

Surface appears same to all viewer.

Some of incoming light is absorbed – depending on material properties.

Roughness of surface causes reflection of incident light in slightly varying angles.
Also called Lambertian surface – modeled after **Lambert’s cosine law.**

Computation: Diffuse Light Reflectance

Theorem (Lambert's cosine law in optics): states that the radiant intensity observed from a "Lambertian" surface is directly proportional to the cosine of the angle θ between the light ray incident on a point on the surface (**l**) and the surface normal at the point (**n**).

Thus, $R_d \propto \cos(\theta) = \mathbf{l}.\mathbf{n}$, where **l** and **n** are unit vectors.

The law is also known as the **cosine emission law** or **Lambert's emission law.**

Potential problem when ($\cos\theta < 0$) – thus, we always take $\max(\cos\theta, 0)$.

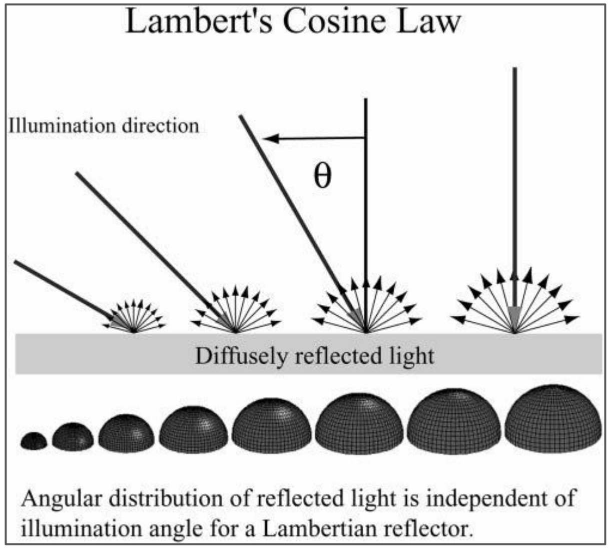

Lambert's Cosine Law
Illumination direction
θ
Diffusely reflected light
Angular distribution of reflected light is independent of illumination angle for a Lambertian reflector.

Image courtesy: Salyer, D. A., Twietmeyer, K., Beaudry, N., Basavanthappa, S., Park, R. I., & Chipman, R. (2005). In vitro multispectral diffuse reflectance measurements of the porcine fundus. *Investigative ophthalmology & visual science*, *46*(6), 2120-2124.

Computation: Diffuse Light Reflectance

Diffuse reflection term is also subjected to **distance-attenuation**.

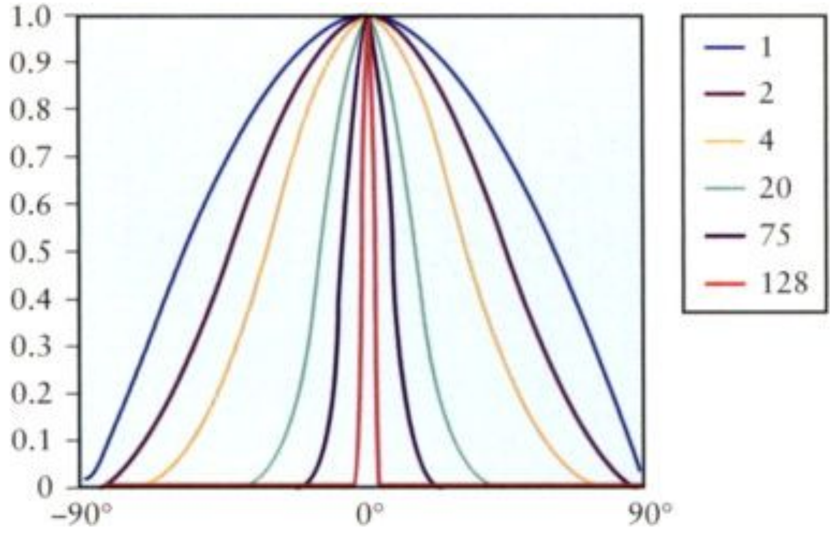
Using reflection coefficient, k_d, where ($0 \leq k_d \leq 1$), we get the diffuse reflection intensity to be,

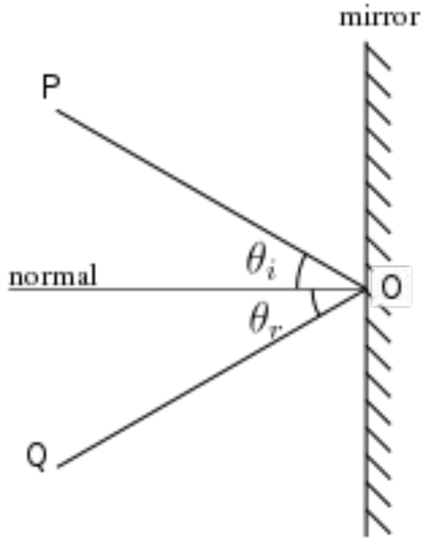
$$I_d = k_d \,.\, \frac{\max(\mathbf{l} \,.\, \mathbf{n}\,, 0)}{a + bd + cd^2} \,.\, L_d$$

Computation: Specular Light Reflectance

Specular highlights make objects shiny – depends on surface color, and smoothness.

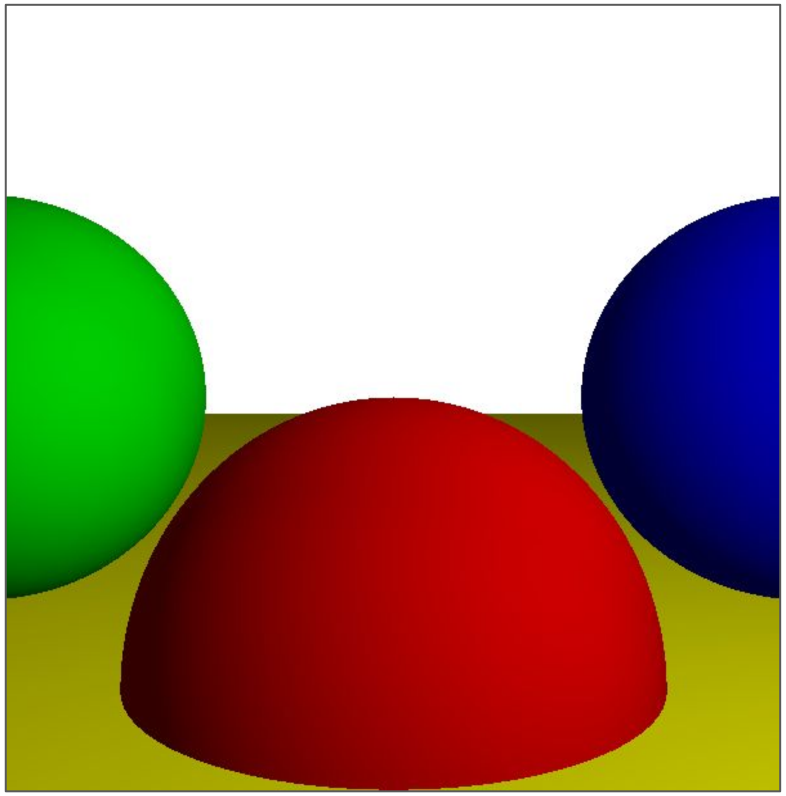
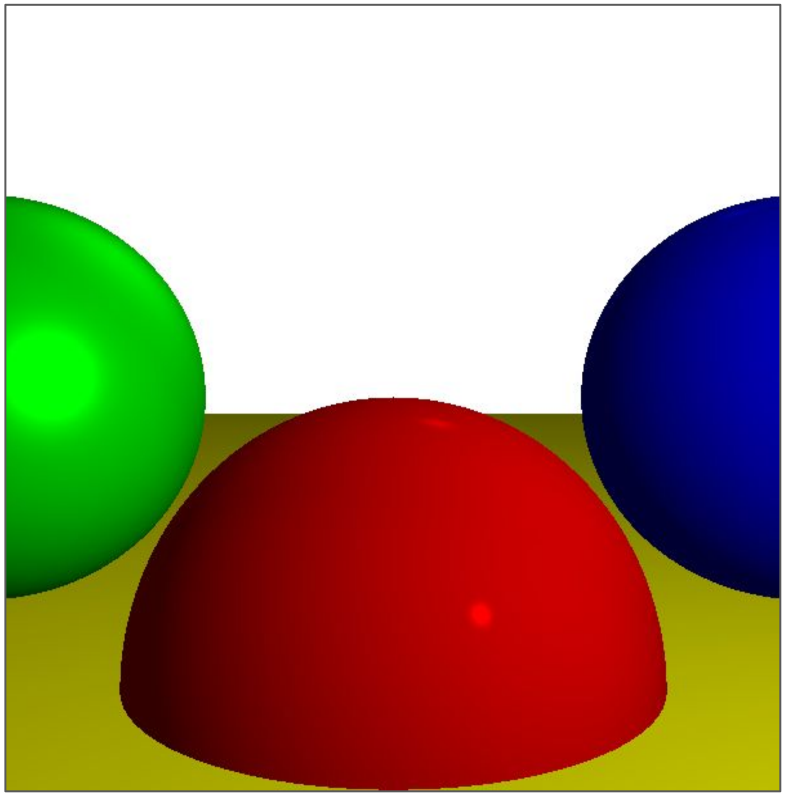
Smoother the surface, the reflected light is concentrated in a smaller range of angles centered about the perfectly reflected surface.





(L) Exponential distribution of shininess in specular reflection; (R) Law of reflection.
(Image Courtesy:https://www.sciencedirect.com/topics/computer-science/lighting-equation || https://en.wikipedia.org/wiki/Specular_reflection)

Example

(L) Diffuse reflection adds a sense of depth and volume to the scene.
(R) The scene rendered with ambient, diffuse, and specular reflection. Not only do we get a sense of depth and volume, but each surface also has a slightly different appearance.

https://gabrielgambetta.com/computer-graphics-from-scratch/03-light.html

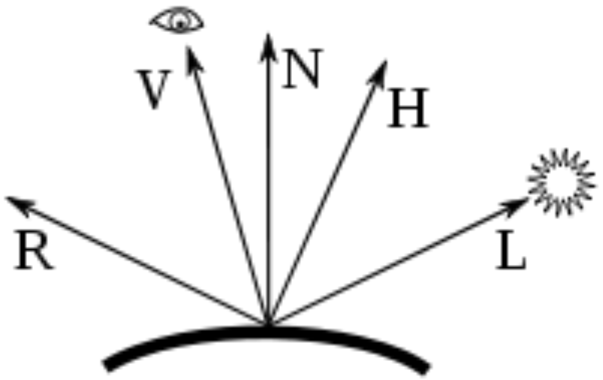
Computation: Specular Light Reflectance

Phong model states that: $I_s = k_s L_s \cos^\alpha \phi = k_s L_s (\mathbf{R} \,.\, \mathbf{v})^\alpha$

where k_s is the specular coefficient for the surface ($0 \le k_s \le 1$); α is the shininess coefficient; **R** and **v** are unit vectors.

Adding the distance-attenuation model, and non-negative value constraint for the cosine function, we get,

$$I_s = \frac{\max((\mathbf{R} \,.\, \mathbf{v})^\alpha, 0)}{a + bd + cd^2} \,.\, k_s L_s$$



https://en.wikipedia.org/wiki/Phong_reflection_model

Computations: Phong Model -- summing all up

Adding all contributions for each color channel and for each light source, we get,

$$I_i = A(d).\,(k_{id} L_{id} \max(\mathbf{l}\,.\,\mathbf{n}\,,0) + k_{is} L_{is} \max((\mathbf{R}\,.\,\mathbf{v})^{\alpha}\,,0)) + k_{ia} L_{ia}$$

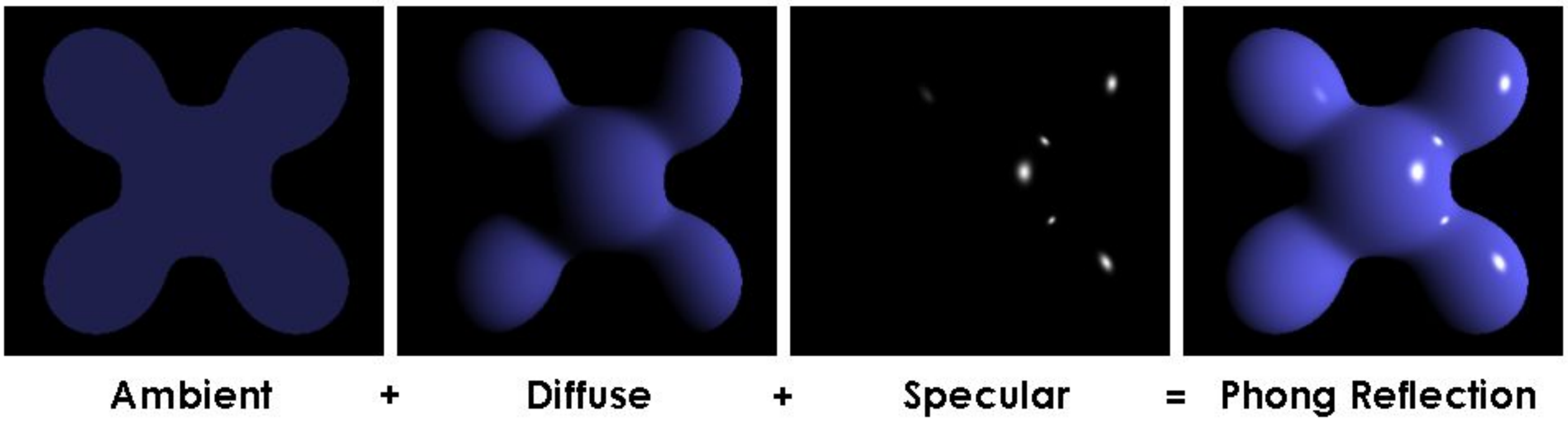
Complete model is sum of contributions of all light sources and a global ambient term (I_{ga}). Thus for each color channel, we get,

$$I = \sum_i I_i + I_{ga}$$

Phong model is an approximation model using additive contributions.

Phong model is good for producing global effects using local calculations.

Computations: Phong Model -- summing all up



https://en.wikipedia.org/wiki/Phong_reflection_model

Computation: Normal Vectors

Given 3 points on a plane, its normal is computed as a cross product:

$n = (p_2 - p_0) \times (p_1 - p_0)$

Note the order of traversal of points – always use **right-hand rule.**

In OpenGL, the application program should explicitly compute the normal.

- Normals are parameters in OpenGL state machine.
- Normal applied to all vertices defined after its definition.
- OpenGL requires application program to explicitly define the normal to vertices.

Computation: Reflection Vector

Theorem (Law of Reflection) : At a point **p** on the surface, the incident ray, normal and the reflected ray at all lie in the same plane; and the angle of incident ray to normal is equal to the angle of reflected ray to the normal, which gives,

$$\begin{aligned} \mathbf{L}\,.\,\mathbf{n} &= \cos\theta_i = \cos\theta_r = \mathbf{R}\,.\,\mathbf{n} \\ \mathbf{R} &= \alpha\mathbf{L} + \beta\mathbf{n} \\ \text{which reduces to: } \mathbf{R} &= 2(\mathbf{L}\,.\,\mathbf{n})\mathbf{n} - \mathbf{L} \end{aligned}$$

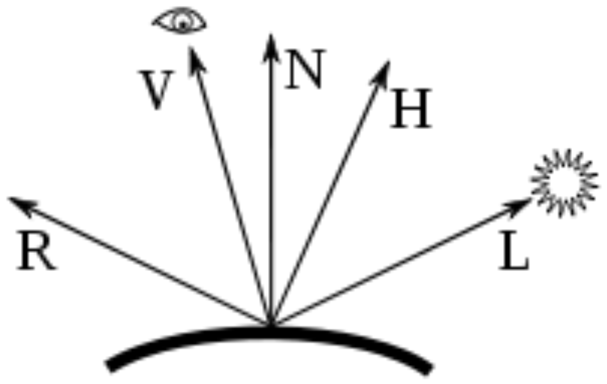
Blinn-Phong Model

For specular calculation, use normal **n** and halfway vector **h** instead of (**R**.**v**),

$$\mathbf{n}\,.\,\mathbf{h} = \mathbf{n}\,.\,\frac{\mathbf{l}+\mathbf{v}}{|\mathbf{l}+\mathbf{v}|}$$

Benefit: avoid computation of **R**, for each pixel depending on surface curvature.

Since (**n.h**) < (**r.v**), use appropriate exponent value for specular highlights such that $(\mathbf{n.h})^{e'} \approx (\mathbf{r.v})^{e}$.

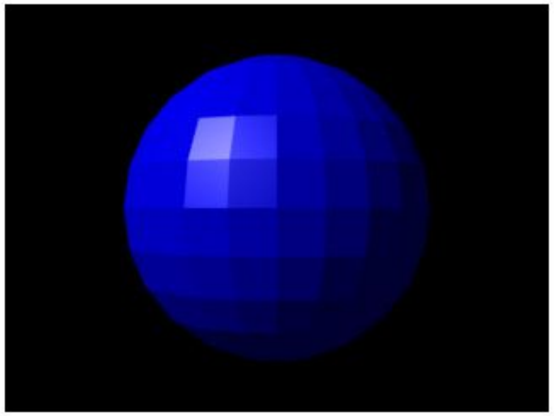


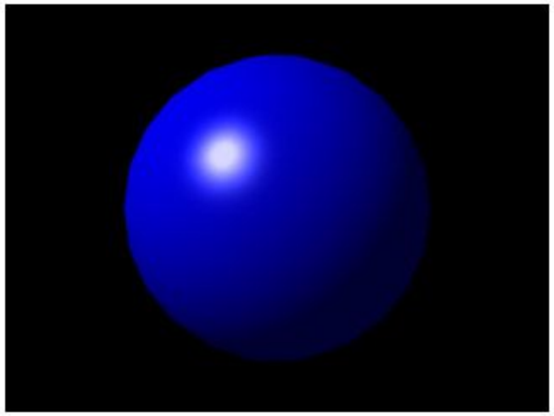
https://en.wikipedia.org/wiki/Phong_reflection_model

Shading Models

Polygonal Shading

- Most interactive graphics systems are polygonal processors.
- Their performance measured using #polygons rendered, lit and shaded per second.
- Normals computed only once for the entire polygon.
- Different shading models: flat, smooth/interpolative/Gouraud, Phong shading.
- Flat and smooth shading models do per-vertex calculations.


FLAT SHADING


PHONG SHADING

https://en.wikipedia.org/wiki/Phong_shading

Flat Shading Model

Though **v, n, l** vary per-vertex,

- **n** is constant for a flat polygon.
- For a distant viewer, **v** is constant over a polygon.
- For a distant light source, **l** is constant over a polygon.

“Distant” implies either (a) source and viewer are at infinity, or (b) size of polygon is comparable with respect to its distance from source and viewer.

In flat/constant shading, shading computation is done only once per polygon.

```
glShadeModel(GL_FLAT) ;
```

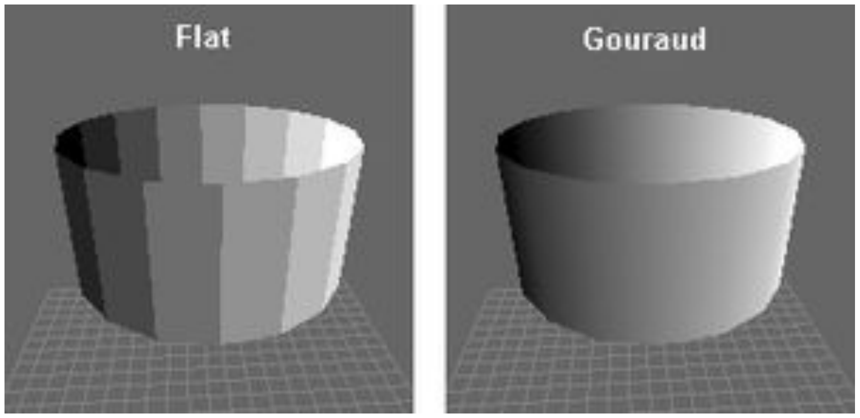
Flat Shading Model

OpenGL uses specific rules for using normals for each primitives.

- It uses the first vertex of a single polygon.
- It uses the third vertex for the first triangle, the fourth vertex for the second, and so on; in the case of triangle strip.

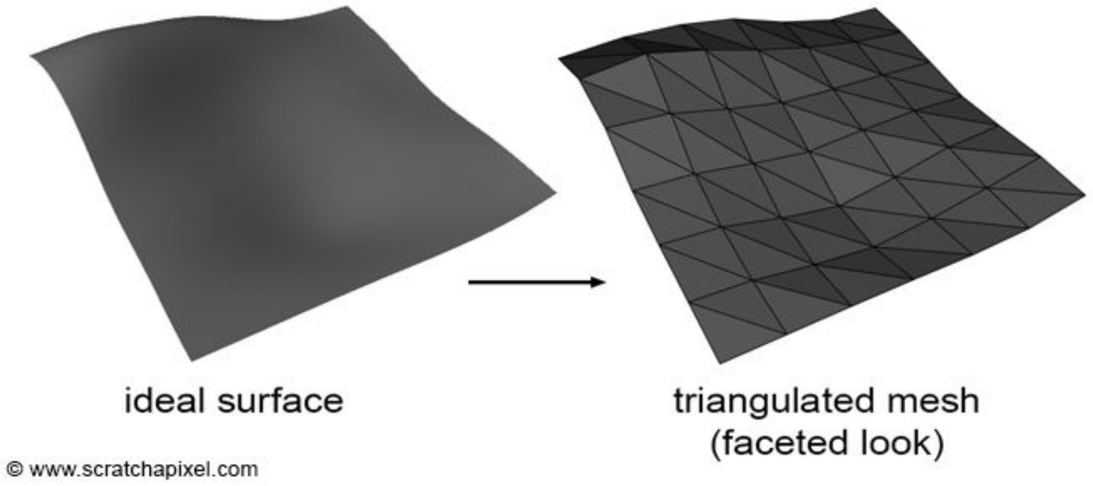
Disadvantage: Human eye is **laterally inhibited**, i.e., sensitive to slight differences in light intensity. Due to which, if l and v are different for each polygon, there will be **Mach bands** in the shading – stripes of colors with slight intensity change demonstrate exaggerated differences.

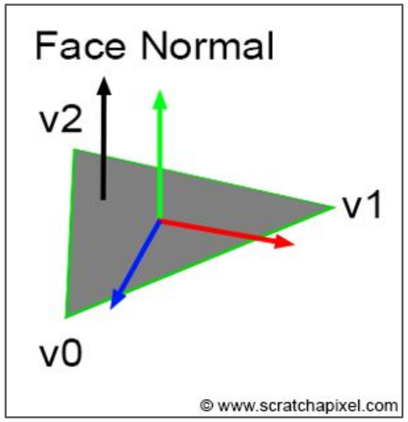
Flat shading gives faceted appearance.

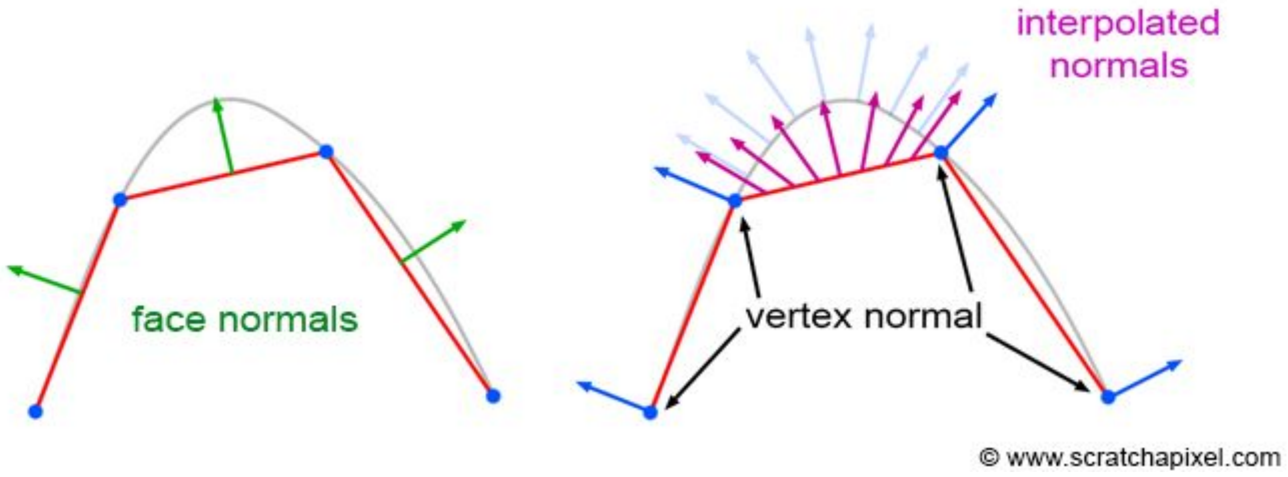


https://en.wikipedia.org/wiki/Gouraud_shading

Vertex, Face and Interpolated Normals







https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/shading-normals

Smooth/Interpolative Shading

Per-vertex calculation of material properties, v and l.

In case of distant source, distant viewer or absence of specular reflection, polygon will be constant shaded.

```
glShadeModel(GL_SMOOTH) ;
```

Gouraud Shading

Normals at vertices as done in smooth shading can be discontinuous.

Gouraud shading: If a vertex P shared by k polygons with normals n_1, n_2, ..., n_k, then normal at vertex P is:

$$\mathbf{n} = \frac{\sum_{i=1}^{k} \mathbf{n}_i}{\left| \sum_{i=1}^{k} \mathbf{n}_i \right|}$$

- Continuous normals at vertices.
- For a pipeline renderer, a data structure required to determine for each vertex the polygons that share it .
  - e.g., Using nested linked lists or vectors.

```
glShadeModel(GL_SMOOTH) ;
glNormalfv(n) ;
glVertexfv(v) ;
```

Phong Shading

To mitigate Mach bands in Gouraud shading, Phong shading interpolates normals across each polygon and lighting model applied to each point in the polygon.

Computing normals across surface makes it curved instead of flat.

This shading is done along with rasterization; off-line.

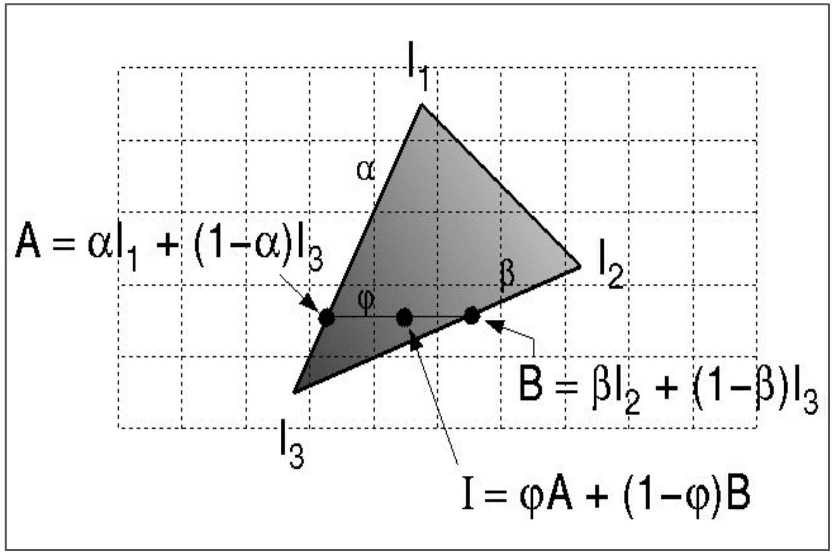
Shading applied to each fragment, called **fragment shading**, can be done using shader program.

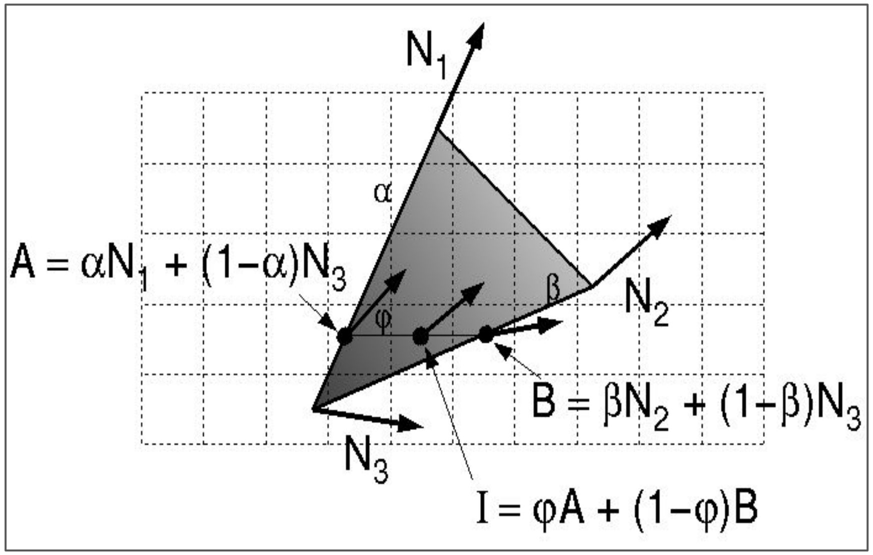
Phong Shading

1. Compute vertex normals using averages (as in Gouraud shading).
2. Use bilinear interpolation in polygons to interpolate normals in fragments over the polygon.
3. Perform shading computation independently for each fragment with interpolated normal.

Phong Shading

1. Compute vertex normals using averages (as in Gouraud shading).
2. Use bilinear interpolation in polygons to interpolate normals in fragments over the polygon.
3. Perform shading computation independently for each fragment with interpolated normal.


I_1
α
$A = \alpha I_1 + (1-\alpha) I_3$
I_2
β
φ
$B = \beta I_2 + (1-\beta) I_3$
I_3
$I = \varphi A + (1-\varphi) B$


N_1
α
$A = \alpha N_1 + (1-\alpha) N_3$
β
N_2
φ
$B = \beta N_2 + (1-\beta) N_3$
N_3
$I = \varphi A + (1-\varphi) B$

Bilinear interpolation of color (left, Gouraud shading) and of normals prior to color computation (right, Phong shading)

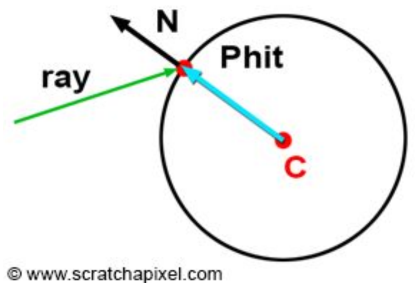
https://www.cs.princeton.edu/courses/archive/fall00/cs426/lectures/scan/scan.pdf

Application: Shading of a Sphere Model

Preparing a spherical object using recursive subdivision:

1. Find a polyhedron (say tetrahedron) with vertices that lie on a unit sphere, centered at origin.
2. Draw the wireframe of tetrahedron – such that vertices of the faces follow right-hand rule.
3. Subdivide each face using bisecting angles, centroid, or bisecting sides.
4. Move new points used for subdivision from plane of triangle to the sphere by normalizing the points.
5. Recursively subdivide all triangular facets and normalize points to obtain approximation of a sphere.

Application: Shading of a Sphere Model

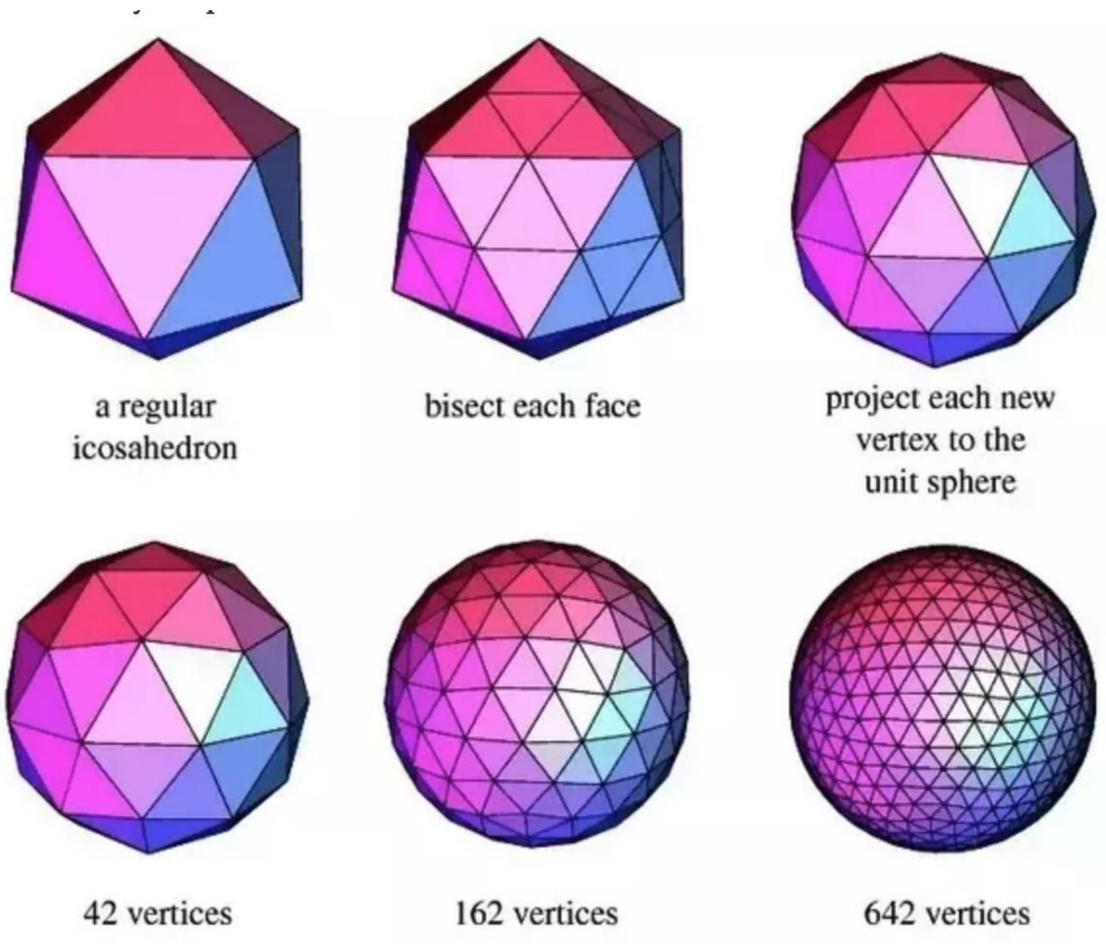


Normal=||P-C||

<u>Proof:</u>
Equation $x^2+y^2+z^2=r^2$
Implicit form:
$f(x,y,z)=(x^2+y^2+z^2-r^2)$

Normal
$= [\partial f/\partial x, \partial f/\partial y, \partial f/\partial z]$
$= [2x, 2y, 2z]$

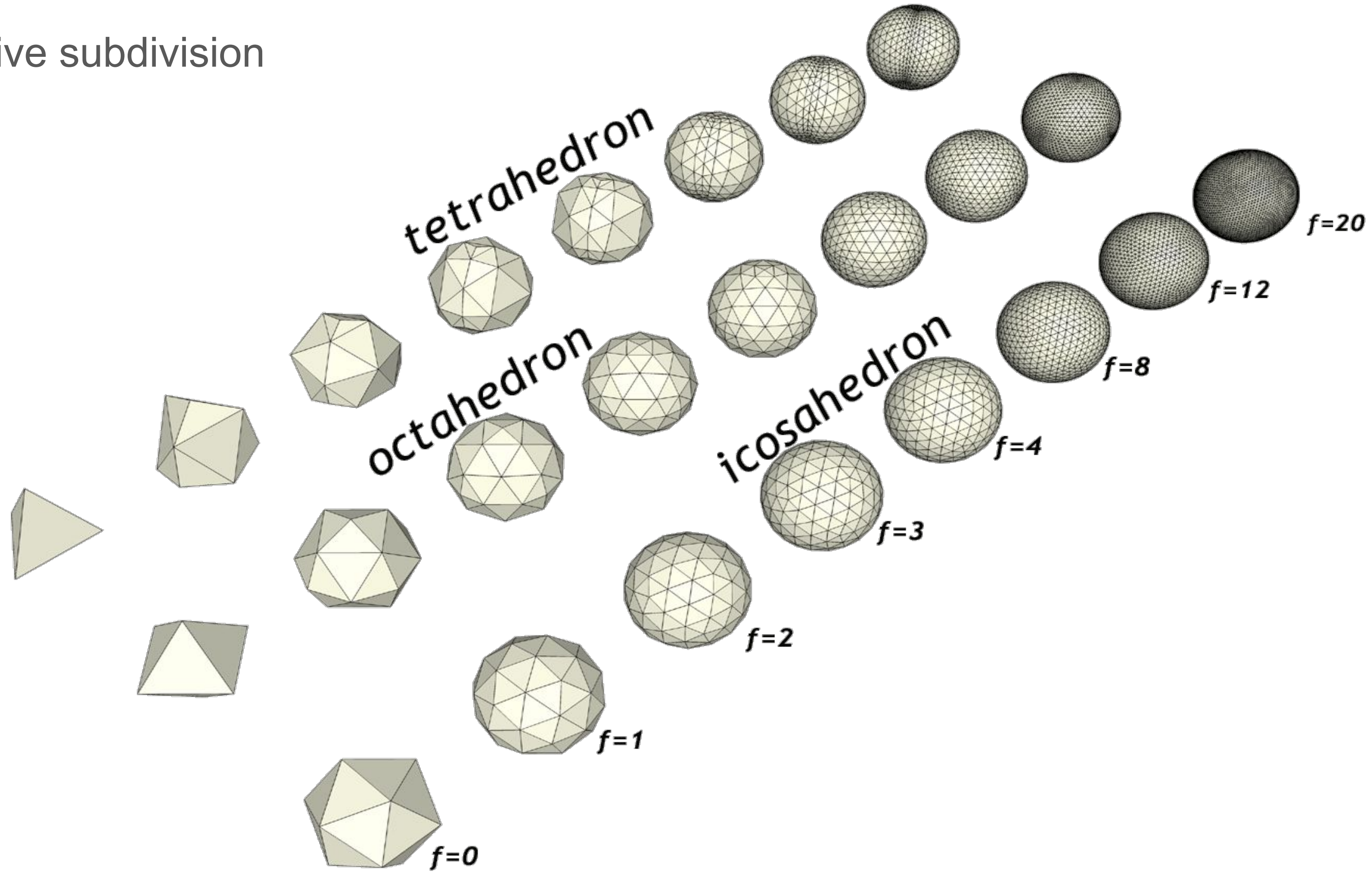
Unit Normal
= [x, y, z] □ point itself



https://stackoverflow.com/questions/48553298/snapping-vector-to-a-point-from-a-grid-on-a-sphere-icosahedron

Application: Shading of a Sphere Model

Recursive subdivision



http://phrogz.net/CSS/Geodesics/VariousGeodesics_planar.png

Implementation: Shading of a Sphere Model

Shading the sphere model obtained after recursive subdivision:

1. Use cross product to calculate normal for each triangle.
2. Normalize the normal and specify normal when rendering each triangle.
3. For smooth shading (i.e., to remove edges of subdivided polygons (silhouette edges)), use interpolative shading by using the true normal to each vertex.
    a. In the case of a sphere centered at the origin, the true normal at each vertex analytically is the vertex itself.

Local Illumination Models in OGL (Fixed Functionality Pipeline)

Model and Light Position Specifications

Local Viewer: More calculations for reflection than distant viewer (OpenGL default).

Shading Surfaces: For an object, one can shade both front and back faces of surfaces. OpenGL default is to shade only the front face.

Light sources are geometric primitives with specific properties – can be affected by model-view transformations in OpenGL.

- Light position/direction are stored in eye coordinates.
- Careful placement of light-sources can give following effects:
    - Objects move, sources stationary.
    - Objects stationary, sources move.
    - Objects and source move together or separately

Materials

Specifying shading properties for front and back faces of surfaces:

```
glMaterialfv(GLenum face, GLenum type,
             GLfloat *ptr_to_array) ;
glMaterialf(GLenum face, GLenum type, GLfloat value) ;
```

- One can specify face to be: `GL_FRONT, GL_BACK, GL_FRONT_AND_BACK`

Material properties are OpenGL state parameters.

- Significant overhead to specify multiple surfaces with different materials.
- `glColorMaterial` useful for changing a single color material property – more efficient and less general than `glMaterial.`

Reflectances

Specify ambient, specular and diffuse reflectance coefficients for RGBA color model using:

```
GLfloat ka[]={0.2,0.2,0.2,1.};
glMaterialfv(GL_FRONT_AND_BACK,GL_AMBIENT,ka);
```

Similarly,

```
glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,kd);
glMaterialfv(GL_FRONT_AND_BACK,GL_SPECULAR,ks);
```

Combine reflectivities, if needed. For example:

```
GL_DIFFUSE_AND_SPECULAR
```

Specular and Emissive Properties

Specify exponent for specular-reflection model:

```
glMaterialf(GL_FRONT,GL_SHININESS,100.);
```

Specify emissive component for self-luminous sources:

```
GLfloat em[]={0.,0.3,0.3,1.};
glMaterialfv(GL_FRONT_AND_BACK,GL_EMISSION,em);
```

Useful to show light source in scene without interacting with other sources or surfaces.

Global Illumination

Introduction

Limitations of local illumination model:

- Renders all objects in scene similarly without blocking light.
- Cannot handle multiple reflections and other light interactions.
- Cannot render shadows.

These global effects require a global illumination model.

- Ray-tracing.
- Radiosity rendering.

Owing to dependency of rendering of polygons, it is not compatible with pipeline architecture.

Ray-tracing and Radiosity

Ray-tracing

- Start with synthetic-camera model.
- Determine for each projector striking a polygon if the point has been illuminated by other sources to compute total local shading of the point.
- A local renderer computes the shade at a point using modified Phong model.
- It takes into consideration the reflection, refraction and scattering of other sources and objects in the scene.

Radiosity

- Based on principle of energy conservation.
- A radiosity calculation based on energy balance of all participants of the scene. It involves solving a large system of equations.

OpenGL Rendering

Ray-tracing best for rendering scene with highly reflective surface; radiosity for scenes with perfectly diffuse surfaces.

OpenGL can implement global illumination using programmable shaders for real-time rendering.

- Offline rendering used 32-bit arithmetic – more precision.
- Real-time rendering uses 8-bit accuracy – loss of detail.

A note on RTX GPUs dedicated for Ray-tracing.

The RTX platform provides software APIs and SDKs running on advanced hardware to provide solutions capable of accelerating and enhancing graphics, photos, imaging and video processing. These include:

- Ray Tracing (OptiX, Microsoft DXR, Vulkan)
- AI-Accelerated Features (NGX)
- Rasterization (Advanced Shaders)
- Simulation (CUDA 10, PhysX, Flex)
- Asset Interchange Formats (USD, MDL)

[From https://developer.nvidia.com/rtx]

Summary

- Color Theory
    - Color Models
- Lighting
    - Introduction
    - Rendering Equation
    - Local Illumination Model
- Shading Models
- Local Illumination Model in OGL
- Global Illumination (in brief)