Illumination & Shading

- Light Sources
- Empirical Illumination
- Shading

Lecture 15

CISC440/640

Spring 2015

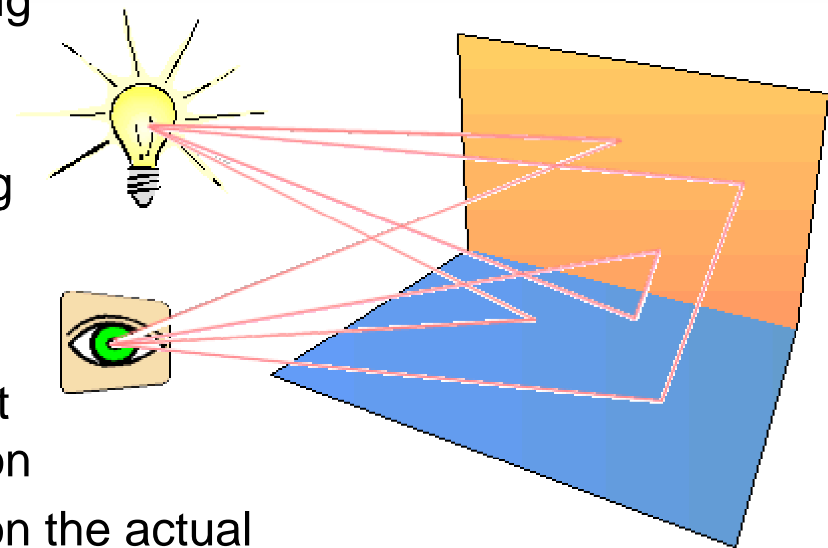
Computer Graphics

Department of Computer and Information Science

Illumination Models

Computer Graphics Jargon:

- **Illumination** - the transport luminous flux from light sources between surfaces via direct and indirect paths
- **Lighting** - the process of computing the luminous intensity reflected from a specified 3-D point
- **Shading** - the process of assigning a colors to a pixels



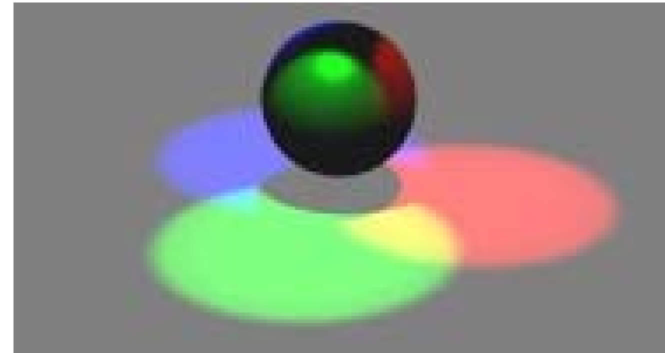
Illumination Models:

- Empirical - simple formulations that approximate observed phenomenon
- Physically-based - models based on the actual physics of light's interactions with matter

Two Components of Illumination

Light Sources:

- Emittance Spectrum (color)
- Geometry (position and direction)
- Directional Attenuation



Surface Properties:

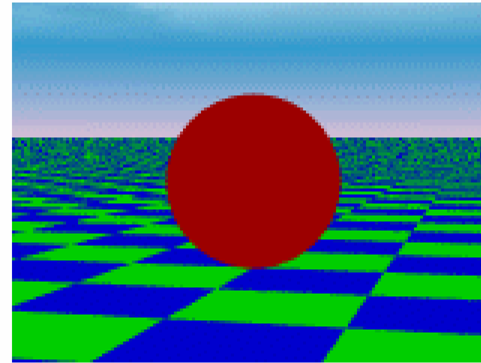
- Reflectance Spectrum (color)
- Geometry (position, orientation, and micro-structure)
- Absorption

Simplifications used by most computer graphics systems:

- Only the direct illumination from the emitters to the reflectors of the scene
- Ignore the geometry of light emitters, and consider only the geometry of reflectors

Ambient Light Source

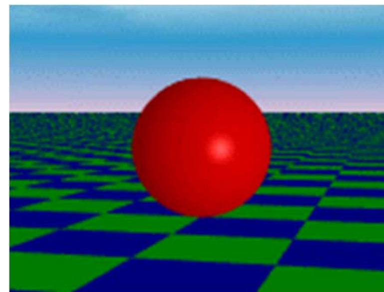
- Even though an object in a scene is not directly lit it will still be visible. This is because light is reflected indirectly from nearby objects. A simple *hack* that is commonly used to model this indirect illumination is to use of an *ambient light source*.



- Ambient light has no spatial or directional characteristics. The amount of ambient light incident on each object is a constant for all surfaces in the scene. An ambient light can have a color.
- The amount of ambient light that is reflected by an object is independent of the object's position or orientation. Surface properties are used to determine how much ambient light is reflected.

Directional Light Sources

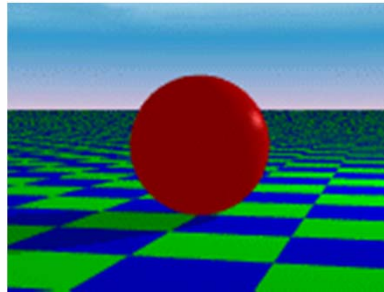
- All of the rays from a directional light source have a common direction, and no point of origin. It is as if the light source was infinitely far away from the surface that it is illuminating. Sunlight is an example of an infinite light source.



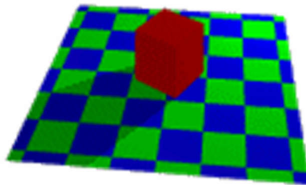
- The direction from a surface to a light source is important for computing the light reflected from the surface. With a directional light source this direction is a constant for every surface.
- A directional light source can be colored.

Point Light Sources

- The rays emitted from a point light radially diverge from the source. A point light source is a fair approximation to a local light source such as a light bulb.



- The direction of the light to each point on a surface changes when a point light source is used. Thus, a normalized vector to the light emitter must be computed for each point that is illuminated.



$$\vec{d} = \frac{\dot{p} - \dot{l}}{\left\| \dot{p} - \dot{l} \right\|}$$

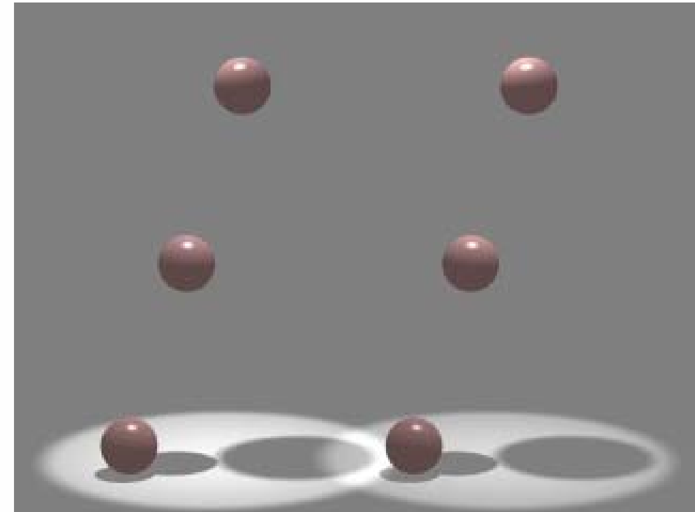
Lights in OpenGL

```
glEnable(GL_LIGHTING);

// set up light's color
glLightfv(GL_LIGHT0, GL_AMBIENT, ambientIntensity);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseIntensity);
glLightfv(GL_LIGHT0, GL_SPECULAR, specularIntensity);

// define a directional light
public float [] lightDirection = {1.0f, 1.0f, 1.0f, 0};
glLightfv(GL_LIGHT0, GL_POSITION, lightDirection);
glEnable(GL_LIGHT0);

// define a point light
public float [] lightPoint = {100.0f, 100.0f, 100.0f, 1.0f};
glLightfv(GL_LIGHT1, GL_POSITION, lightPoint);
glEnable(GL_LIGHT1);
```

Other Light Sources

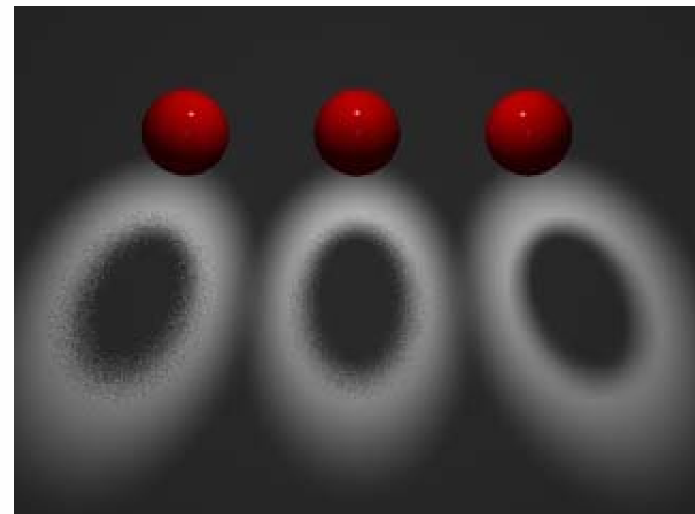
Spotlights

- Point source whose intensity falls off away from a given direction
- Requires a color, a point, a direction, parameters that control the rate of fall off



Area Light Sources

- Light source occupies a 2-D area (usually a polygon or disk)
- Generates *soft* shadows

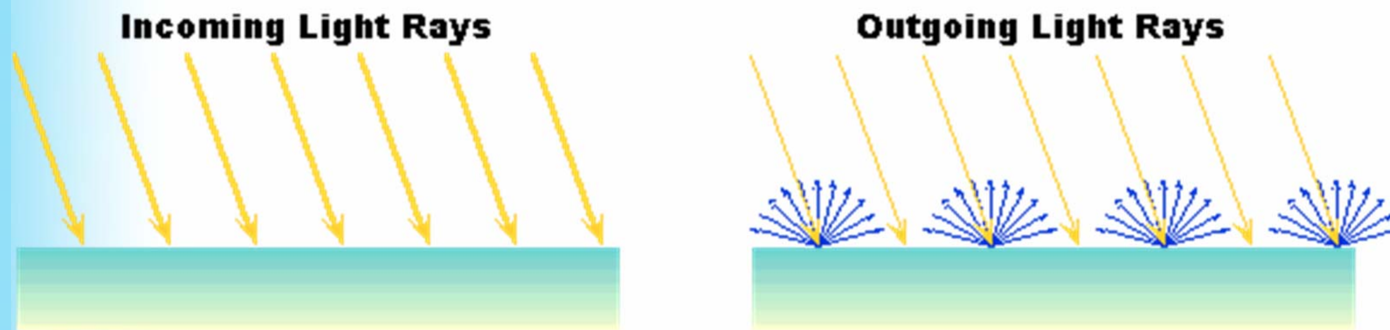


Extended Light Sources

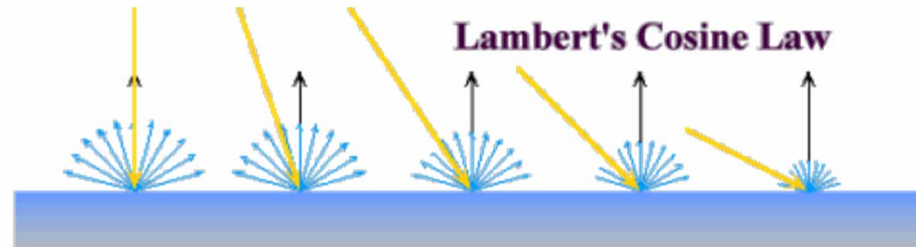
- Spherical Light Source
- Generates *soft* shadows

Ideal Diffuse Reflection

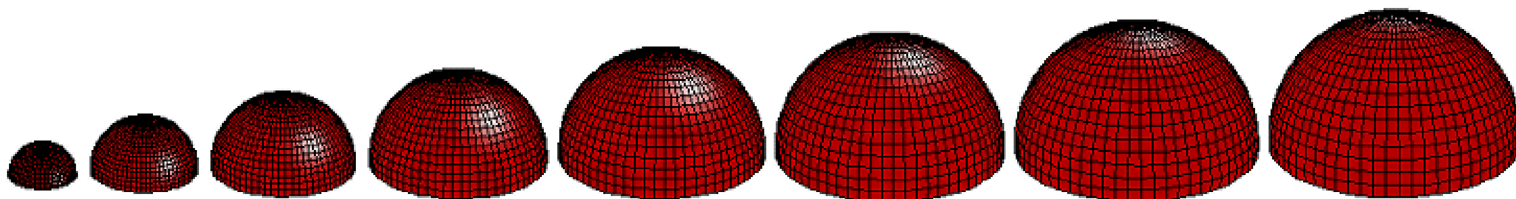
First, we will consider a particular type of surface called an *ideal diffuse reflector*. An ideal diffuse surface is, at the microscopic level a very rough surface. Chalk is a good approximation to an ideal diffuse surface. Because of the microscopic variations in the surface, an incoming ray of light is equally likely to be reflected in any direction over the hemisphere.



Lambert's Cosine Law

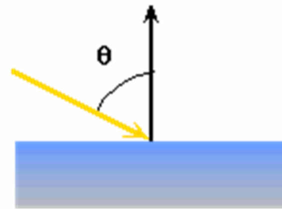


Ideal diffuse reflectors reflect light according to *Lambert's cosine law*, (there are sometimes called Lambertian reflectors). Lambert's law states that the reflected energy from a small surface area in a particular direction is proportional to cosine of the angle between that direction and the surface normal. Lambert's law determines how much of the *incoming* light energy is reflected. Remember that the amount energy that is reflected in any one direction is constant in this model. In other words the reflected intensity is **independent of the viewing direction**. The intensity does however depend on the light source's orientation relative to the surface, and it is this property that is governed by Lambert's law.



Computing Diffuse Reflection

- The angle between the surface normal and the incoming light ray is called the angle of incidence and we can express a intensity of the light in terms of this angle. $I_{diffuse} = k_d I_{light} \cos\theta$



- The I_{light} term represents the intensity of the incoming light at the particular wavelength (the wavelength determines the light's color). The k_d term represents the diffuse reflectivity of the surface at that wavelength.
- In practice we use vector analysis to compute cosine term indirectly. If both the *normal vector* and the incoming *light vector* are normalized (unit length) then diffuse shading can be computed as follows:

$$I_{diffuse} = k_d I_{light} \left(\bar{n} \cdot \bar{l} \right)$$

Diffuse Lighting Examples

We need only consider angles from 0 to 90 degrees. Greater angles (where the dot product is negative) are blocked by the surface, and the reflected energy is 0. Below are several examples of a spherical diffuse reflector with a varying lighting angles.

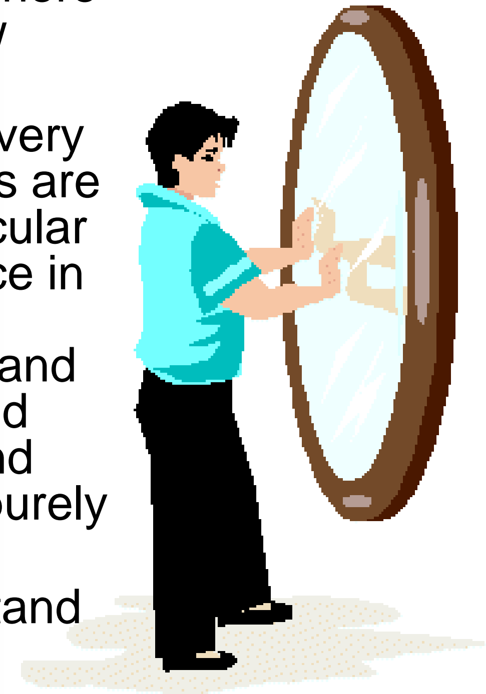
Why do you think spheres are used as examples when shading?

Specular Reflection

A second surface type is called a *specular reflector*. When we look at a shiny surface, such as polished metal or a glossy car finish, we see a highlight, or bright spot. Where this bright spot appears on the surface is a function of where the surface is seen from. This type of reflectance is view dependent.

At the microscopic level a specular reflecting surface is very smooth, and usually these microscopic surface elements are oriented in the same direction as the surface itself. Specular reflection is merely the *mirror reflection* of the light source in a surface. Thus it should come as no surprise that it is viewer dependent, since if you stood in front of a mirror and placed your finger over the reflection of a light, you would expect that you could reposition your head to look around your finger and see the light again. An ideal mirror is a purely specular reflector.

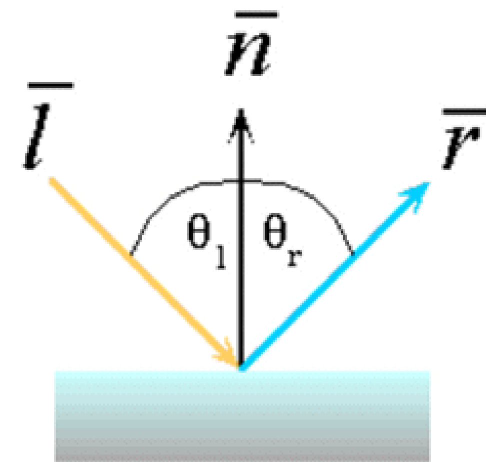
In order to model specular reflection we need to understand the physics of reflection.



Snell's Law

Reflection behaves according to Snell''s law:

- The incoming ray, the surface normal, and the reflected ray all lie in a common plane.
- The angle that the reflected ray forms with the surface normal is determined by the angle that the incoming ray forms with the surface normal, and the relative speeds of light of the mediums in which the incident and reflected rays propogate according to the following expression.



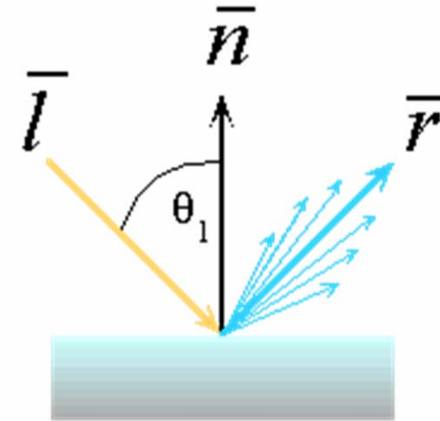
(Note: n_l and n_r are the indices of refraction)

Reflection is a special case of Snell's Law where the incident light's medium and the reflected rays medium is the same. Thus we can simplify the expression to:

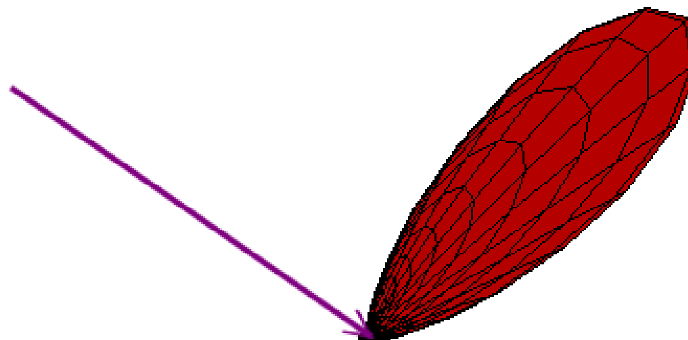
$$\theta_l = \theta_r$$

Non-ideal Reflectors

Snell's law, however, applies only to ideal *mirror* reflectors. Real materials, other than mirrors and chrome tend to deviate significantly from ideal reflectors. *At this point we will introduce an empirical model that is consistent with our experience, at least to a crude approximation.*



In general, we expect most of the reflected light to travel in the direction of the ideal ray. However, because of microscopic surface variations we might expect some of the light to be reflected just slightly offset from the ideal reflected ray. As we move farther and farther, in the angular sense, from the reflected ray we expect to see less light reflected.

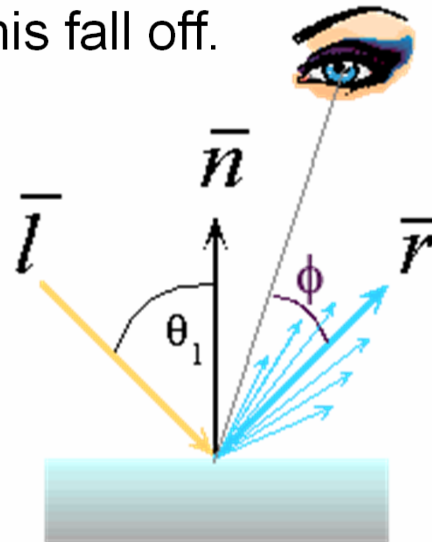


Phong Illumination

- One function that approximates this fall off is called the *Phong Illumination* model. This model has no physical basis, yet it is one of the most commonly used illumination models in computer graphics.

$$I_{specular} = I_{light} (\cos \phi)^{n_{shiney}}$$

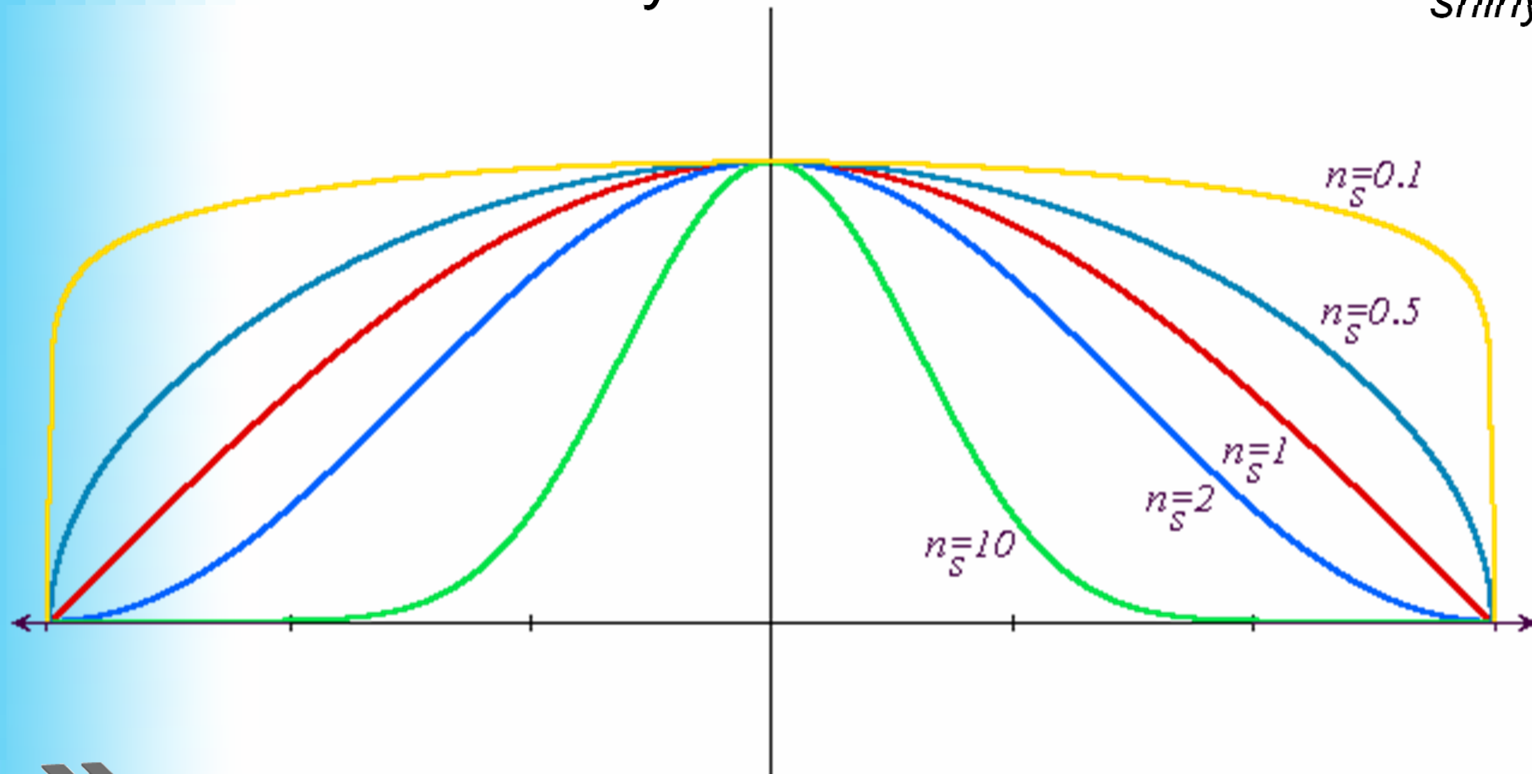
- The *cos* ϕ term is maximum when the surface is viewed from the mirror direction and falls off to 0 when viewed at 90 degrees away from it. The n_{shiny} term controls the rate of this fall off.



COMPUTER GRAPHICS
Department of Computer and Information Science

Effect of n_{shiney}

The diagram below shows the how the Phong reflectance drops off based on the viewers angle from the reflected ray for various values of n_{shiny}.



Computing Phong Illumination

The *cos* term of Phong's specular illumination using the following relationship.

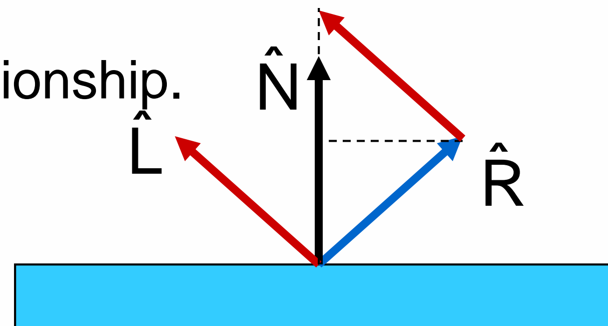
$$I_{specular} = k_s I_{light} (\hat{V} \cdot \hat{R})^{n_{shiney}}$$

The V vector is the unit vector in the direction of the viewer and the R vector is the mirror reflectance direction. The vector R can be computed from the incoming light direction and the surface normal as shown below.

$$\hat{R} = (2(\hat{N} \cdot \hat{L}))\hat{N} - \hat{L}$$

The diagram below illustrates this relationship.

$$\hat{R} + \hat{L} = (2(\hat{N} \cdot \hat{L}))\hat{N}$$



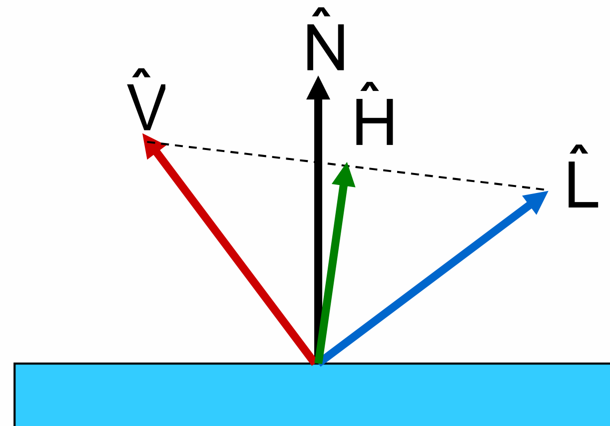
Blinn & Torrance Variation

Jim Blinn introduced another approach for computing Phong-like illumination based on the work of Ken Torrance. His illumination function uses the following equation:

$$I_{specular} = k_s I_{light} (\hat{N} \cdot \hat{H})^{n_{shiney}}$$

In this equation the angle of specular dispersion is computed by how far the surface's normal is from a vector bisecting the incoming light direction and the viewing direction.

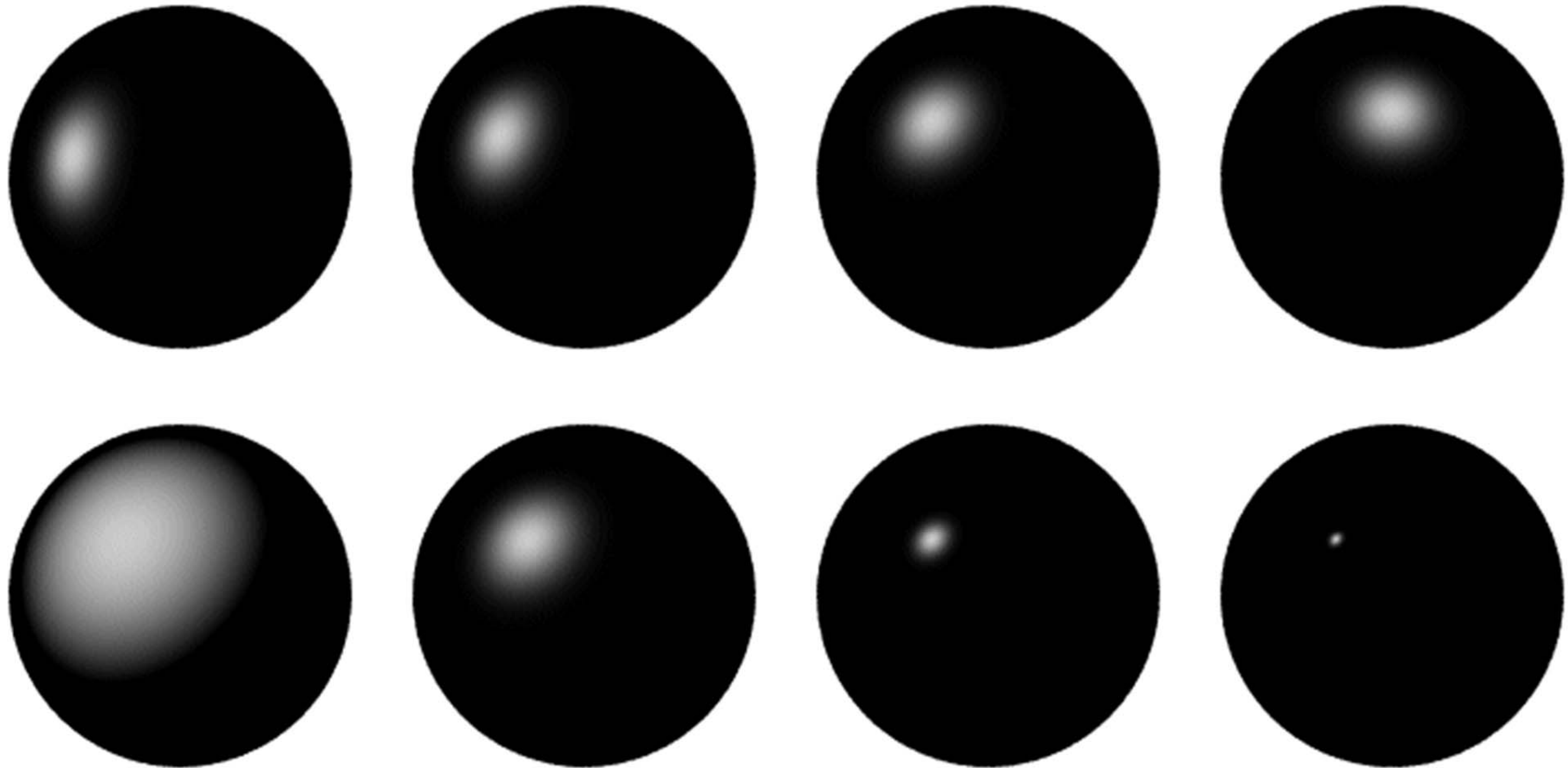
$$\hat{H} = \frac{\hat{L} + \hat{V}}{\left|\hat{L} + \hat{V}\right|}$$



On your own you should consider how this approach and the previous one differ.

Phong Examples

The following spheres illustrate specular reflections as the direction of the light source and the coefficient of shininess is varied.



Putting it all together

Our final empirical illumination model is:

$$I_{total} = k_a I_{ambient} + \sum_{i=1}^{lights} I_i \left(k_d (\hat{N} \cdot \hat{L}) + k_s (\hat{N} \cdot \hat{H})^{n_{shiney}} \right)$$

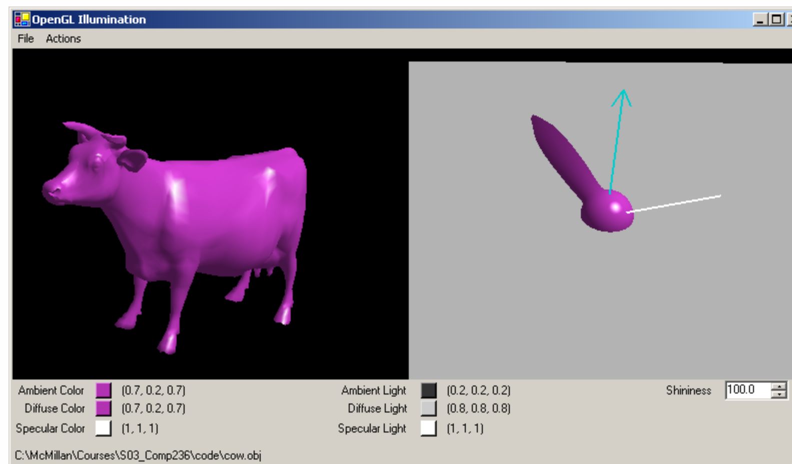
Notes:

- The *Phong Lighting* model
- Once per light
- Once per color component
- Reflectance coefficients, k_a, k_d, and k_s may or may not vary with the color component
- If they do, you need to be careful.

Phong	$\rho_{ambient}$	$\rho_{diffuse}$	$\rho_{specular}$	ρ_{total}
$\phi_i = 60^\circ$				
$\phi_i = 25^\circ$				
$\phi_i = 0^\circ$				

OpenGL Surface Properties

```
public float [] ambientColor = {0.7f, 0.2f, 0.7f, 1.0f};
public float [] diffuseColor = {0.7f, 0.2f, 0.7f, 1.0f};
public float [] specularColor = {0f, 0f, 0f, 1.0f};
public float [] nshininess = { 100.0f };

glMaterialfv(GL_FRONT, GL_AMBIENT, ambientColor);
glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuseColor);
glMaterialfv(GL_FRONT, GL_SPECULAR, specularColor);
glMaterialfv(GL_FRONT, GL_SHININESS, nshininess);
```



Revisit OpenGL's Illumination Model

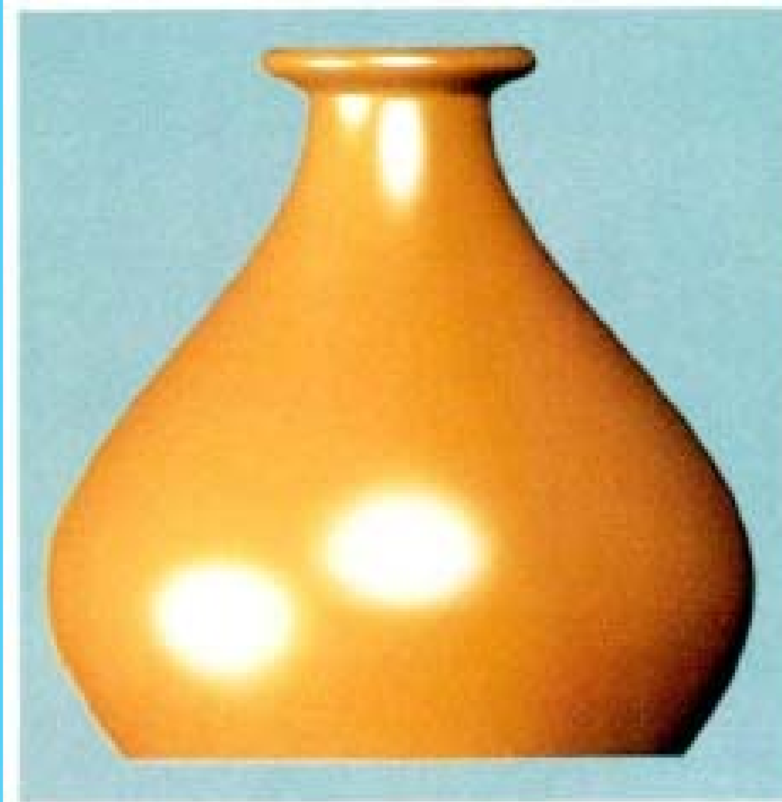
$$I_{total} = \sum_{i=1}^{lights} k_a I_a + k_d I_d Max((\hat{N} \cdot \hat{L}), 0) + k_s I_s Max((\hat{N} \cdot \hat{H}), 0)^{n_{shiney}}$$

Problems with Empirical Models:

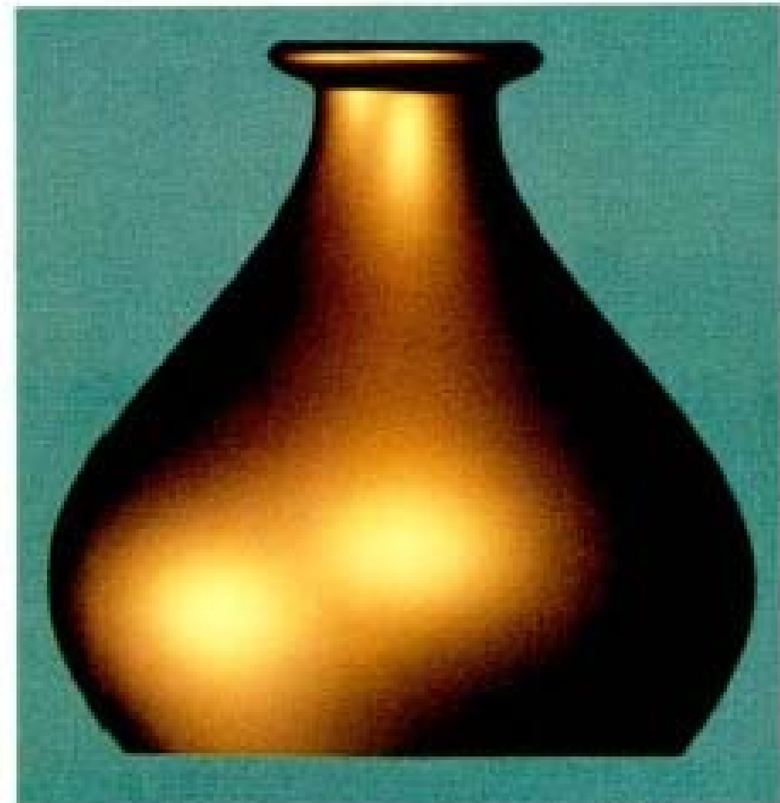
- What are the coefficients for copper?
- What are k_a, k_s, and n_{shiney}?
  Are they measurable quantities?
- How does the incoming light at a point relate to the outgoing light?
  Is energy conserved?
- Just what is light intensity?
- Is my picture accurate?

Results of Cook-Torrance

Plastic-looking copper rendered using Phong model



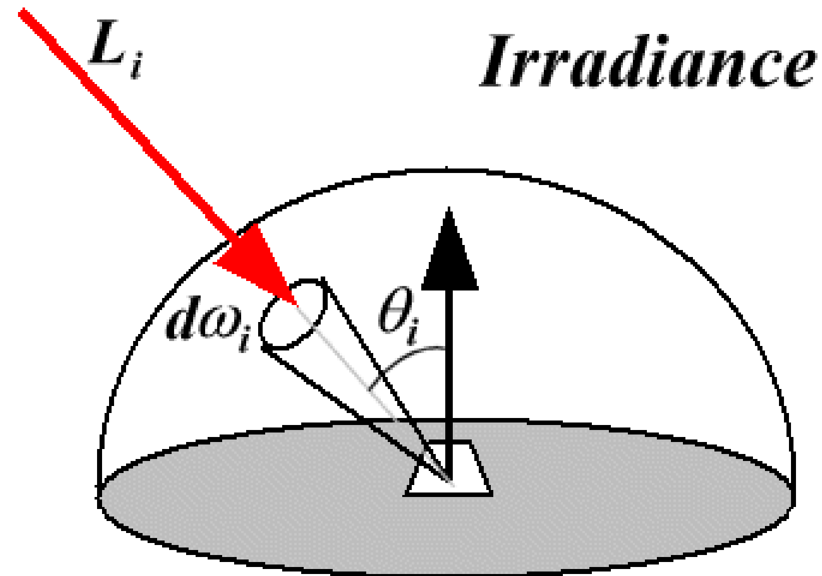
A Copper Vase with a more metallic appearance



More Cook-Torrance results

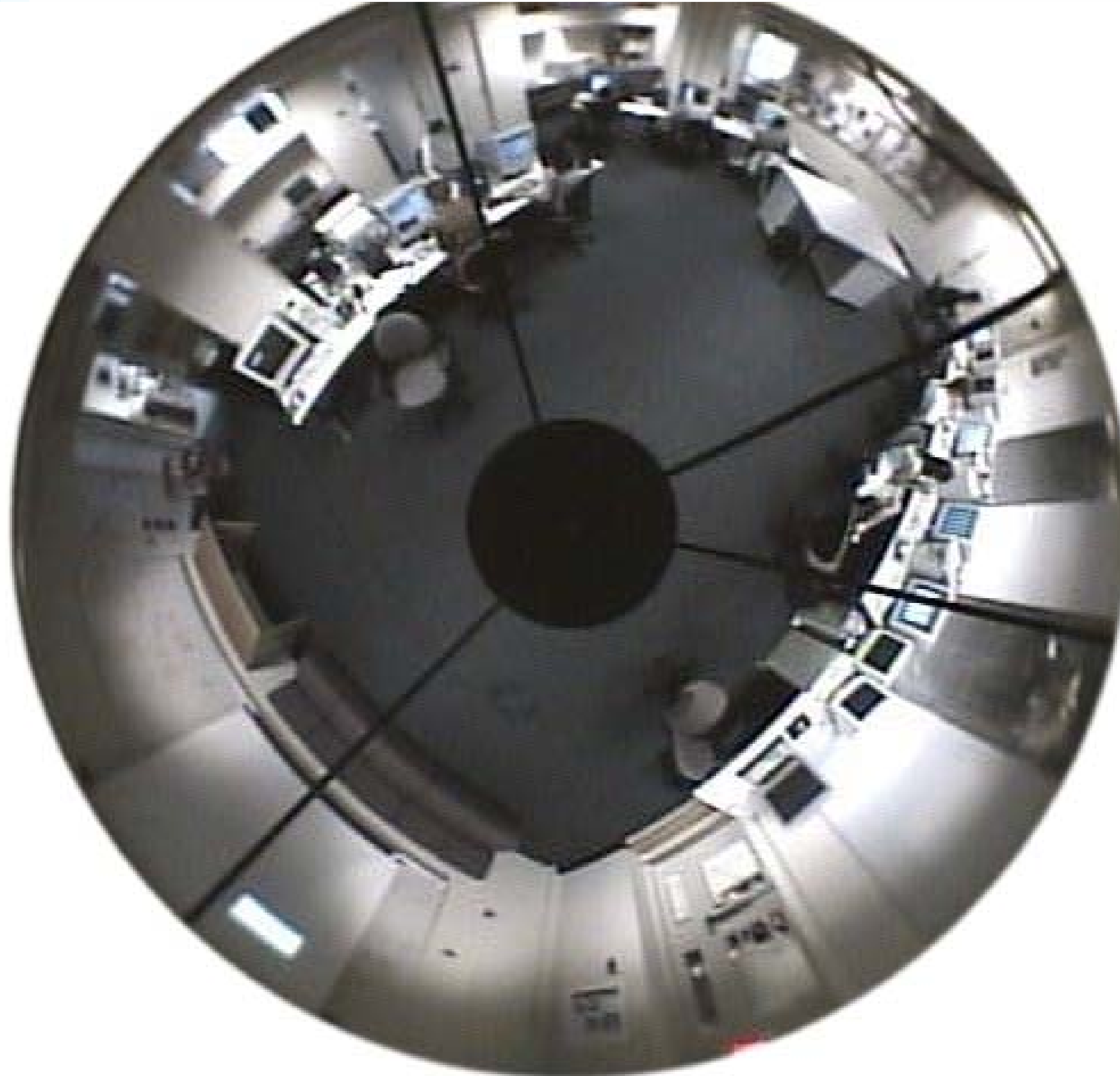
Irradiance

The irradiance function is a two dimensional function describing the incoming light energy impinging on a given point.



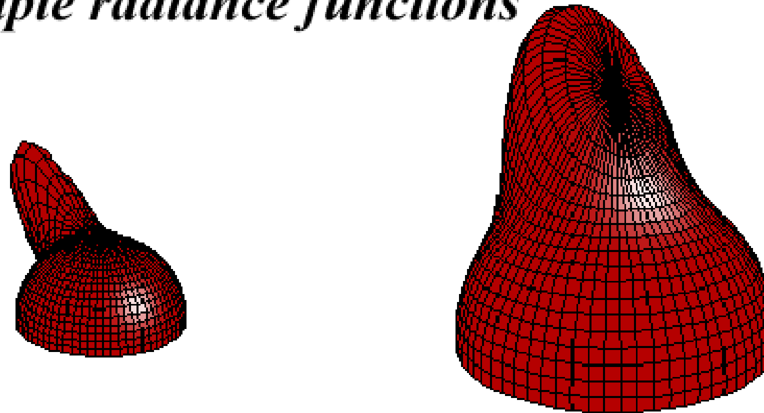
$$E_i = \int_{\Omega_i} L_i \cos\theta_i \, d\omega_i$$

What does Irradiance look like?



Radiance

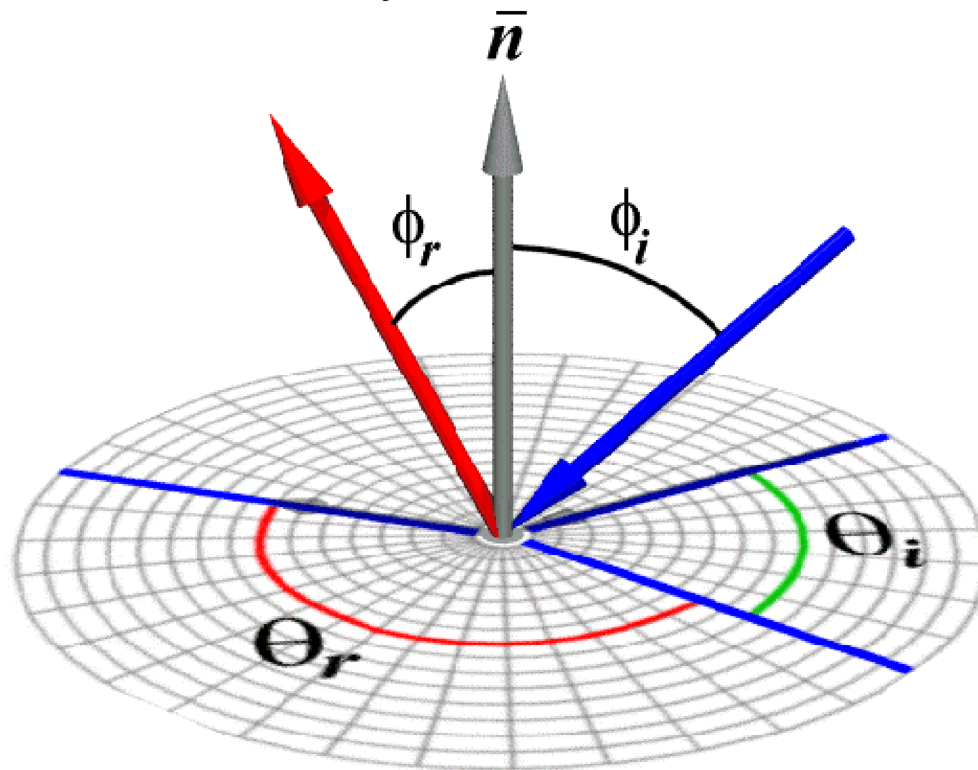
Radiance is a two dimensional function representing the light reflected from a surface. We've already been using plots like these to visualize illumination equations. However, a radiance response integrates over all incoming directions.



Radiance(Irradiance)

Bi-directional Reflectance Distribution Function (BRDF) describes the transport of irradiance to radiance.

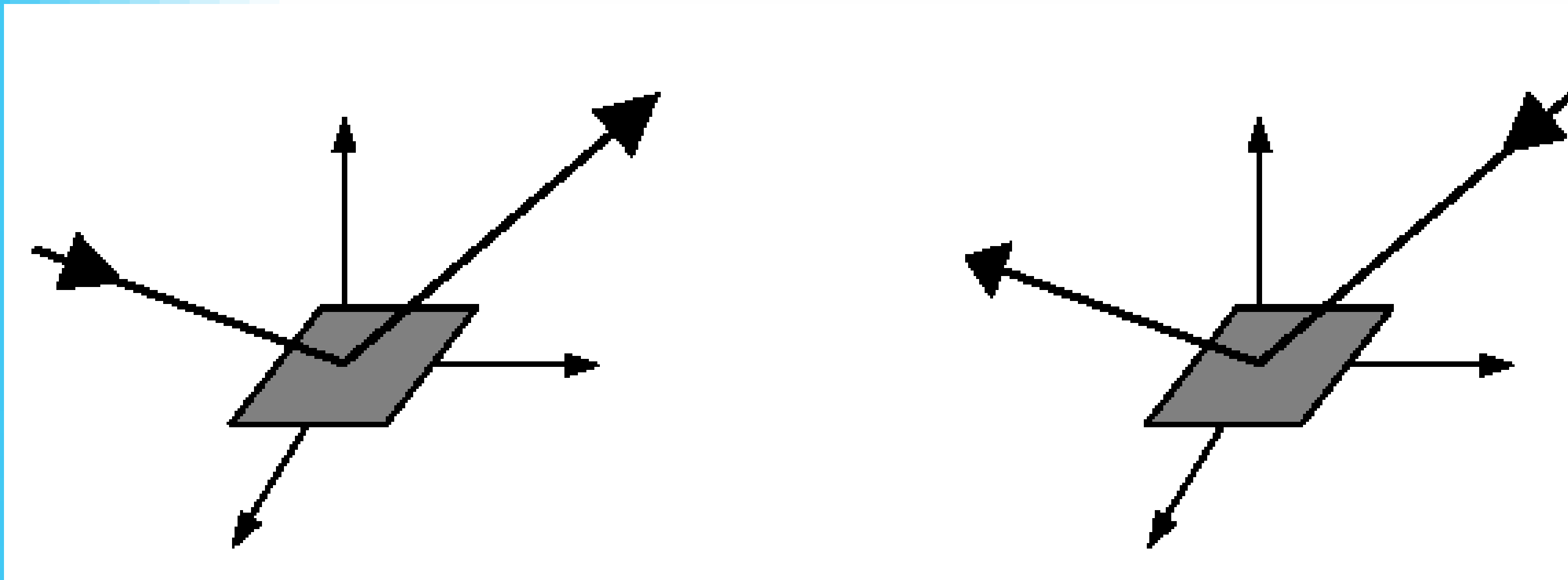
$$\rho(\theta_r, \phi_r, \theta_i, \phi_i)$$



Properties of BRDFs

- Hemholtz Reciprocity

$$BDRF(\theta_i, \phi_i, \theta_r, \phi_r) = BDRF(\theta_r, \phi_r, \theta_i, \phi_i)$$



From “An Introduction to BRDF-Based Lighting” by Wynn

Properties of BRDFs

- Conservation of Energy

$$\int_{\omega_{hemisphere}} BDRF(\omega_i, \omega_r)\, d\omega_i = 1$$

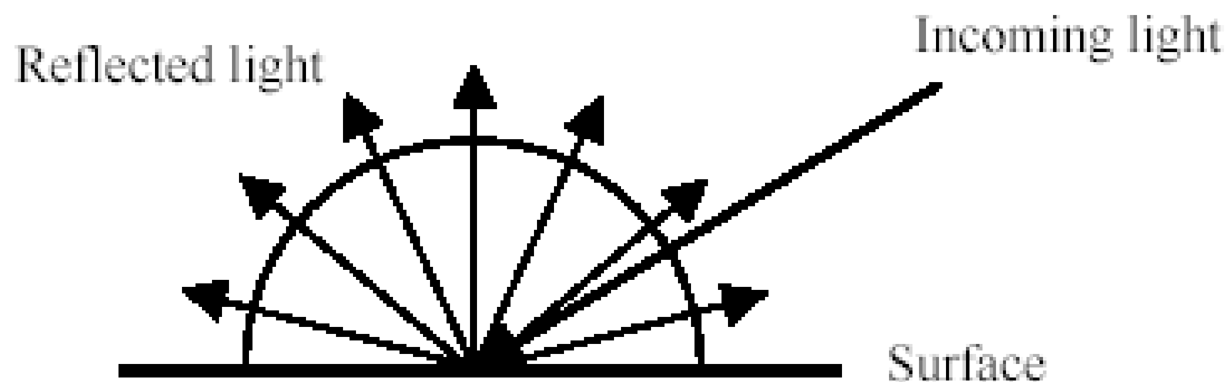


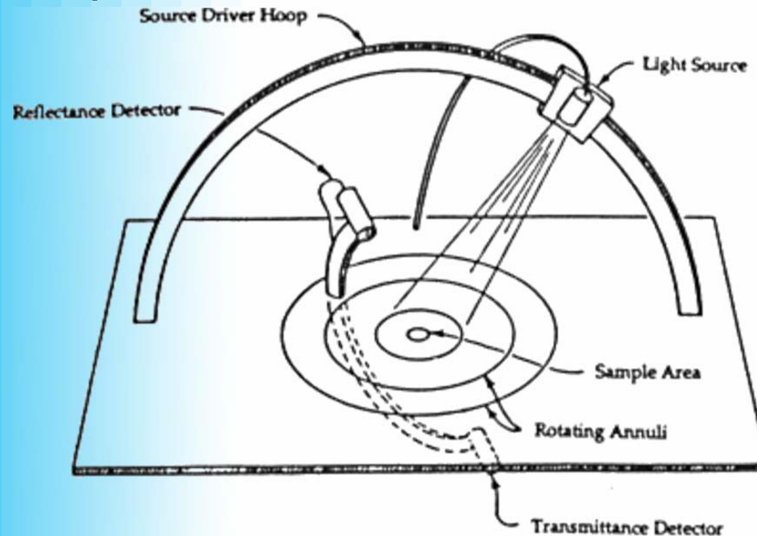
Figure 7. Conservation of Energy- The quantity of light reflected must be less than or equal to the quantity of incident light.

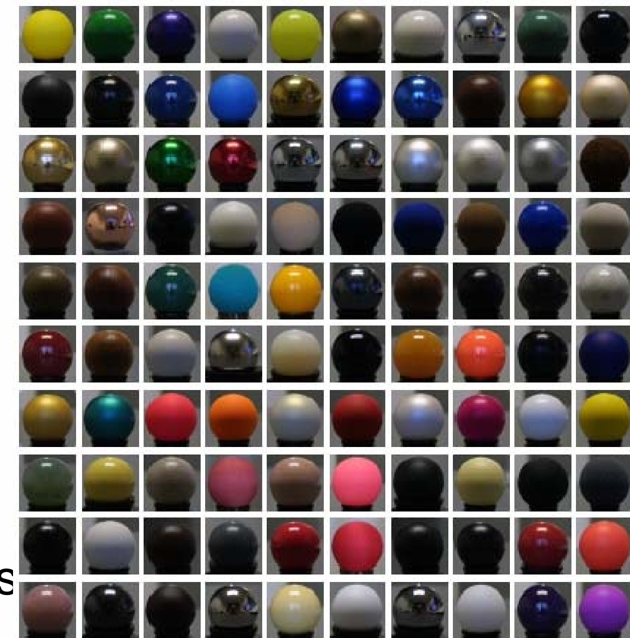
From “An Introduction to BRDF-Based Lighting” by Wynn

Computer Graphics
Department of Computer and Information Science

Measuring BRDFs

- Goniphotometer

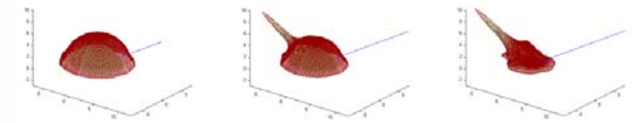




  - One 4-D measurement at a time (slow)
- High-Speed BRDF acquistion
  - Homogeneous sphere w/fixed-camera and orbiting light
  - Each photo = 1000's of BRDF measurements
  - Assumes isotropy

How to Use BRDF Data?

One can make direct use of acquired BRDFs in a renderer.



Affine combinations of acquired BRDFs can also be used to synthesize new materials.

- *Reciprocity?*
- *Energy Conserving?*

Computer Graphics

Department of Computer and Information Science

BRDF Approaches

1) Constraint-based model
   - Ward, Lafortune
   - reciprocal, energy-conserving

2) Measured BRDFs
   - Columbia/Utrecht Reflectance and Texture Database (CUReT)

Video

- BRDF-shop

Next Question: Where do we Illuminate?

To this point we have discussed how to compute an illumination model at a point on a surface. But, at which points on the surface is the illumination model applied? Where and how often it is applied has a noticeable effect on the result.

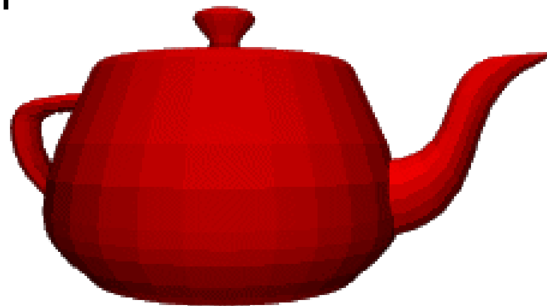
Illuminating can be a costly process involving the computation of and normalizing of vectors to multiple light sources and the viewer.

For models defined by collections of polygonal facets or triangles:

- Each facet has a common surface normal
- If the light is directional then the diffuse contribution is constant across the facet
- If the eye is infinitely far away and the light is directional then the specular contribution is constant across the facet

Flat Shading

The simplest shading method applies only one illumination calculation for each primitive. This technique is called *constant* or *flat shading*. It is often used on polygonal primitives.



Issues:

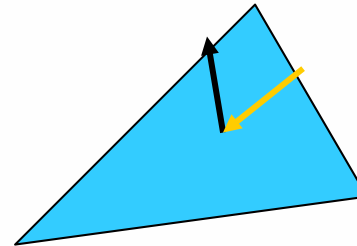
- For point light sources the direction to the light source varies over the facet
- For specular reflections the direction to the eye varies over the facet

None the less, often illumination is computed for only a single point on the facet. Which one? Usually the centroid. For a convex facet the centriod is given as follows:

$$centroid = \frac{1}{vertices} \sum_{i=1}^{vertices} \dot{p}_i$$

Illumination + Flat = Facet Shading

Even when the illumination equation is applied at each point of the faceted nature of the polygonal nature is still apparent.



To overcome this limitation normals are introduced at each vertex.

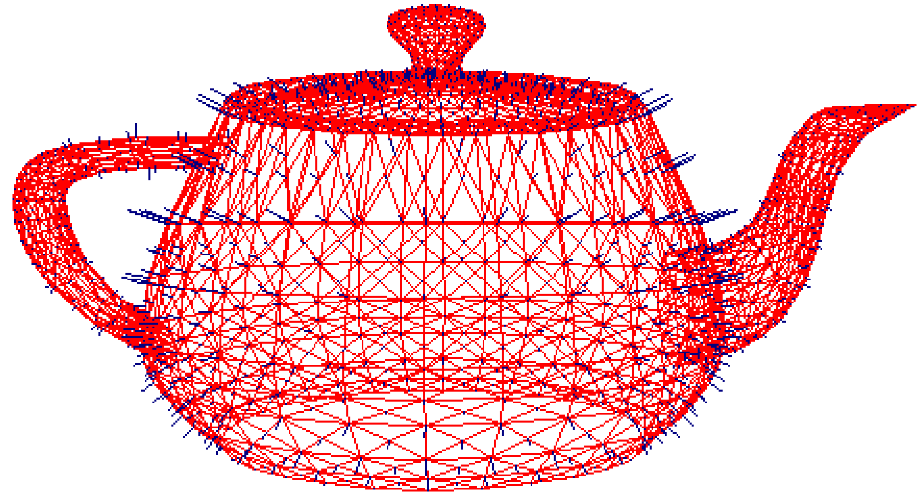
- Usually different than the polygon normal
- Used only for shading (not backface culling or other geometric computations)
- Better approximates the "real" surface
  - Assumes the polygons were a piecewise approximation of the *real* surface (C^0)
  - Normals provide information about the tangent plane at each point (C^1)

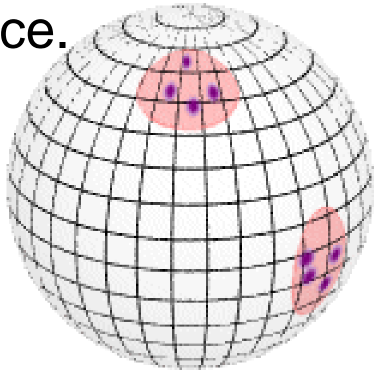
Vertex Normals

If vertex normals are not provided they can often be approximated by averaging the normals of the facets which share the vertex.

$$\vec{n}_v = \sum_{i=1}^{k} \vec{n}_{face_i}$$

This only works if the polygons *reasonably* approximate the underlying surface.

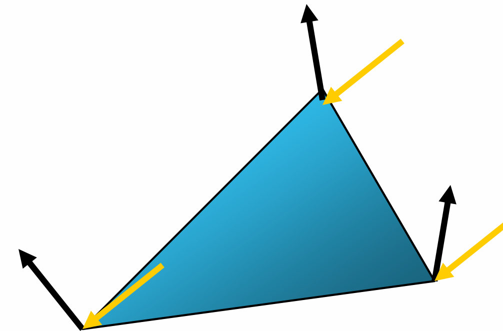




A better approximation can be found using a clustering analysis of the normals on the unit sphere.

Gouraud Shading

The *Gouraud Shading* shading method applies the illumination model on a subset of surface points and interpolates the intensity of the remaining points on the surface. In the case of a polygonal mesh the illumination model is usually applied at each vertex and the colors in the triangles interior are linearly interpolated from these vertex values.



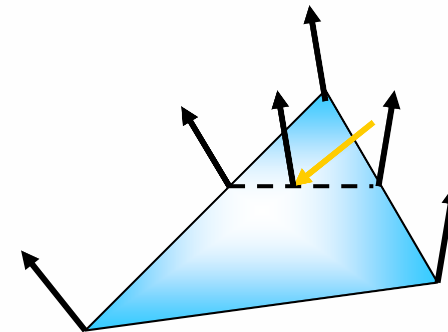
The linear interpolation can be accomplished using the plane equation method discussed in the lecture on rasterizing polygons.

Notice that facet artifacts are still visible.

Phong Shading

In Phong shading (not to be confused with Phong's illumination model), the surface normal is linearly interpolated across polygonal facets, and the Illumination model is applied at every point.

A Phong shader assumes the same input as a Gouraud shader, which means that it expects a normal for every vertex. The illumination model is applied at every point on the surface being rendered, where the normal at each point is the result of linearly interpolating the vertex normals defined at each vertex of the triangle.

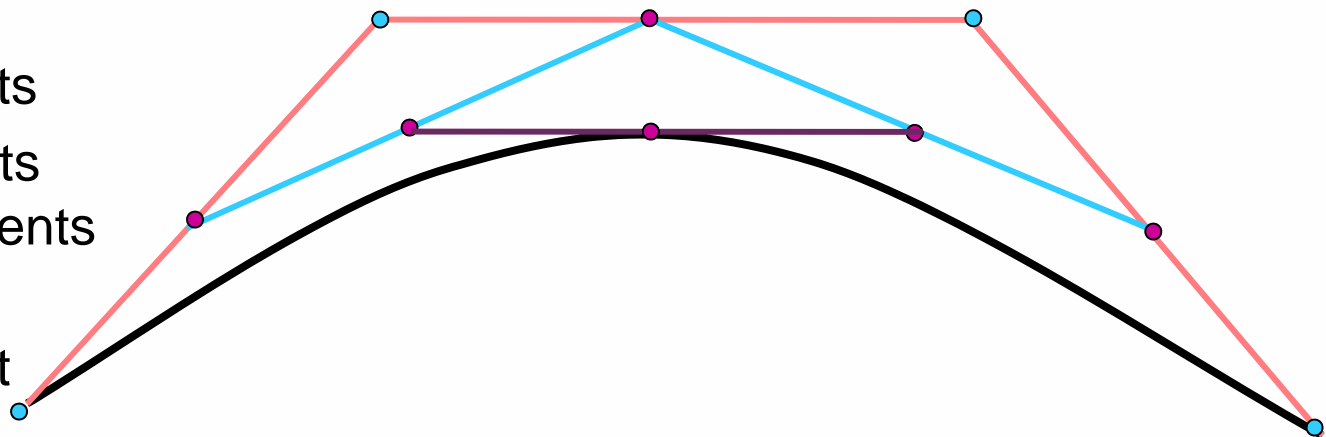


Phong shading will usually result in a very smooth appearance, however, evidence of the polygonal model can usually be seen along silhouettes.

Spline Rendering

There is a very clever alternative method for rendering Bezier Splines using the “de Casteljau” Algorithm. Basically, we can take our set of control points and recursively generate new control points for arbitrary fractions of the domain.

1. Find midpoints of support
2. Connect with new segments
3. Find midpoints of new segments
4. Connect with new segment
5. Find its midpoint



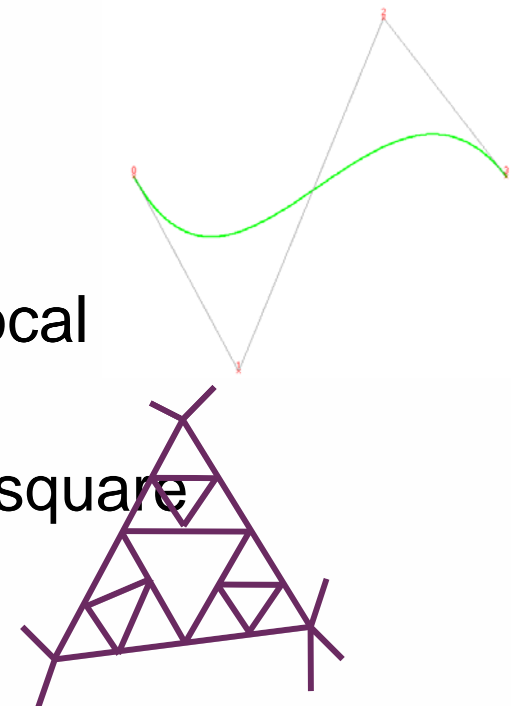
Secret: Subdivision

This process can be repeated recursively… The resulting scaffolding is a good approximation of the actual surface.

Demo Here

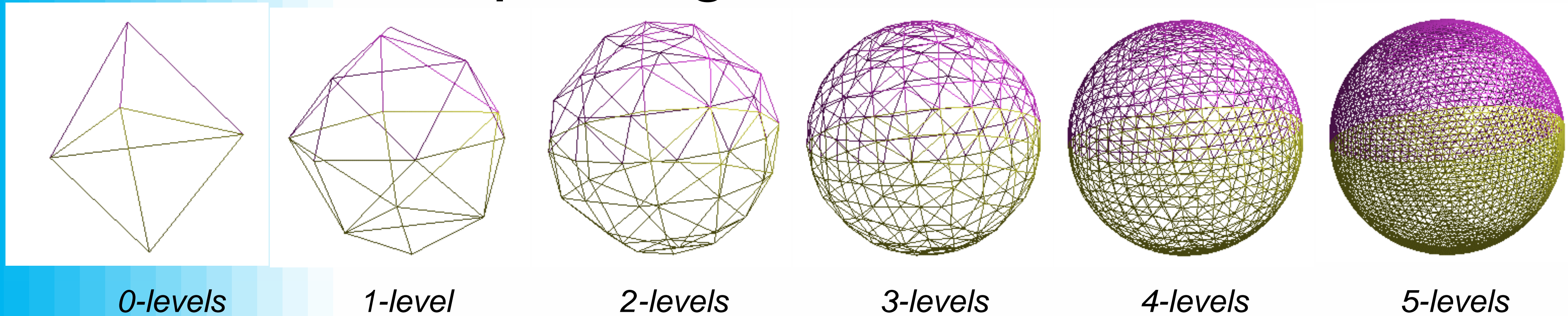
Why use subdivision (recursion) instead of uniform domain sampling (iteration)?

- Stopping conditions can be based on local shape properties (curvature)
- Subdivision can be generalized to non-square domains, in particular to triangular



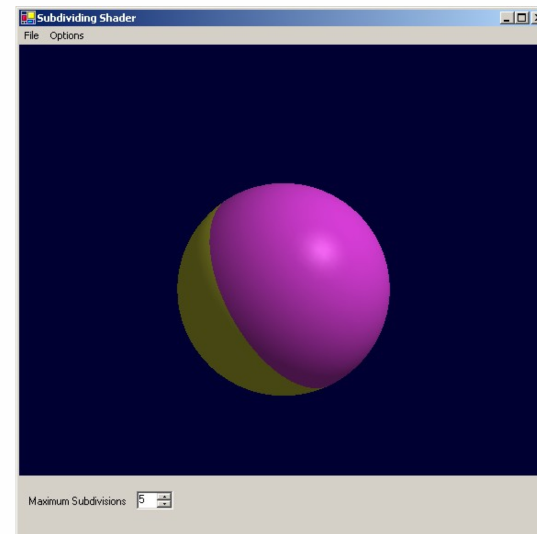
Example of Generalized Subdivision

Here is a sample of generalized subdivision:



0-levels *1-level* *2-levels* *3-levels* *4-levels* *5-levels*

Here is a quick demo:



COMPUTER GRAPHICS

Implementing Subdivision

```
public void drawSphere() {
        Vector v1 = new Vector(0, 1, 0);
        Vector v2 = new Vector(1, 0, 0);
        Vector v3 = new Vector(Math.Cos(2*Math.PI/3), 0, Math.Sin(2*Math.PI/3));
        Vector v4 = new Vector(Math.Cos(4*Math.PI/3), 0, Math.Sin(4*Math.PI/3));
        Vector v5 = new Vector(0, -1, 0);
        glMaterialfv(GL_FRONT, GL_AMBIENT, topColor);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, topColor);
        subSphere(v1, v2, v3, 0);
        subSphere(v1, v3, v4, 0);
        subSphere(v1, v4, v2, 0);
        glMaterialfv(GL_FRONT, GL_AMBIENT, botColor);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, botColor);
        subSphere(v5, v2, v3, 0);
        subSphere(v5, v3, v4, 0);
        subSphere(v5, v4, v2, 0);
    }
```

Implementing Subdivision

```
public void subSphere(Vector v1, Vector v2, Vector v3, int depth) {
        if (depth == maxDepth) {
            Vector t1 = new Vector(v1);
            Vector t2 = new Vector(v2);
            Vector t3 = new Vector(v3);
            glBegin(mode);
            glNormal3d(v1.x, v1.y, v1.z);
            glVertex3d(t1.x, t1.y, t1.z);
            glNormal3d(v2.x, v2.y, v2.z);
            glVertex3d(t2.x, t2.y, t2.z);
            glNormal3d(v3.x, v3.y, v3.z);
            glVertex3d(t3.x, t3.y, t3.z);
            glEnd();
        } else {
            Vector v12 = new Vector(0.5*(v1.x + v2.x), 0.5*(v1.y + v2.y), 0.5*(v1.z + v2.z));
            Vector v23 = new Vector(0.5*(v2.x + v3.x), 0.5*(v2.y + v3.y), 0.5*(v2.z + v3.z));
            Vector v31 = new Vector(0.5*(v3.x + v1.x), 0.5*(v3.y + v1.y), 0.5*(v3.z + v1.z));
            v12.normalize();
            v23.normalize();
            v31.normalize();
            subSphere( v1, v12, v31, depth+1);
            subSphere(v12,  v2, v23, depth+1);
            subSphere(v31, v12, v23, depth+1);
            subSphere(v31, v23,  v3, depth+1);
        }
}
```