6. Lighting and Shading

Outline

- Lighting and Shading I
- Lighting and Shading II
- Lighting and Shading in WebGL
- Polygonal Shading
- Per Vertex and Per Fragment Shaders
- Sample Programs

Lighting and Shading I

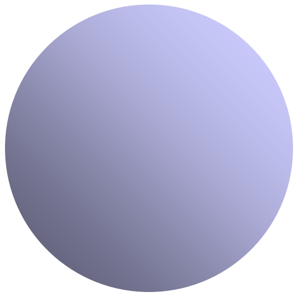
Objectives

- Learn to shade objects so their images appear three-dimensional
- Introduce the types of light-material interactions
- Build a simple reflection model---the Phong model--- that can be used with real time graphics hardware

Why we need shading

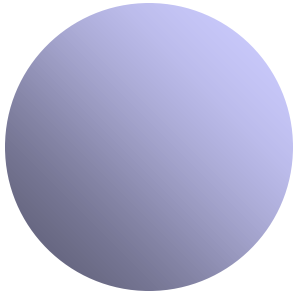
- Suppose we build a model of a sphere using many polygons and color it with **`glColor`**. We get something like

- But we want



Shading

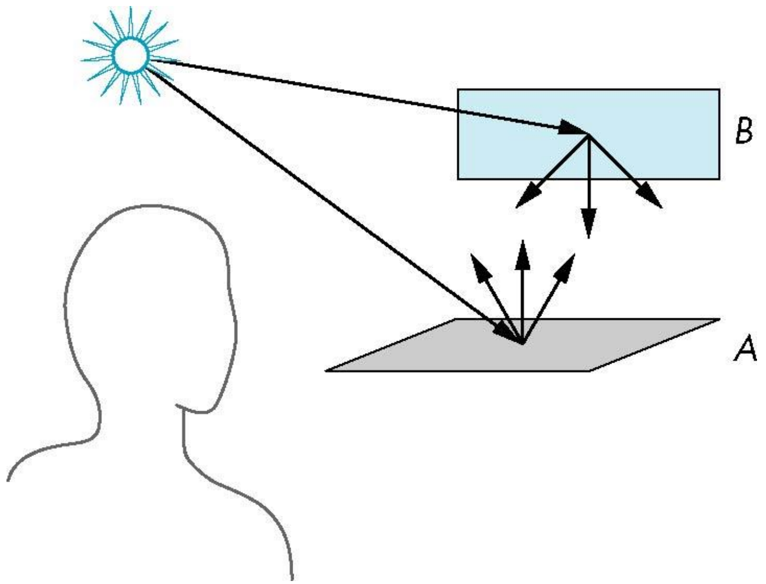
- Why does the image of a real sphere look like



- Light-material interactions cause each point to have a different color or shade
- Need to consider
  - Light sources
  - Material properties
  - Location of viewer
  - Surface orientation

Scattering

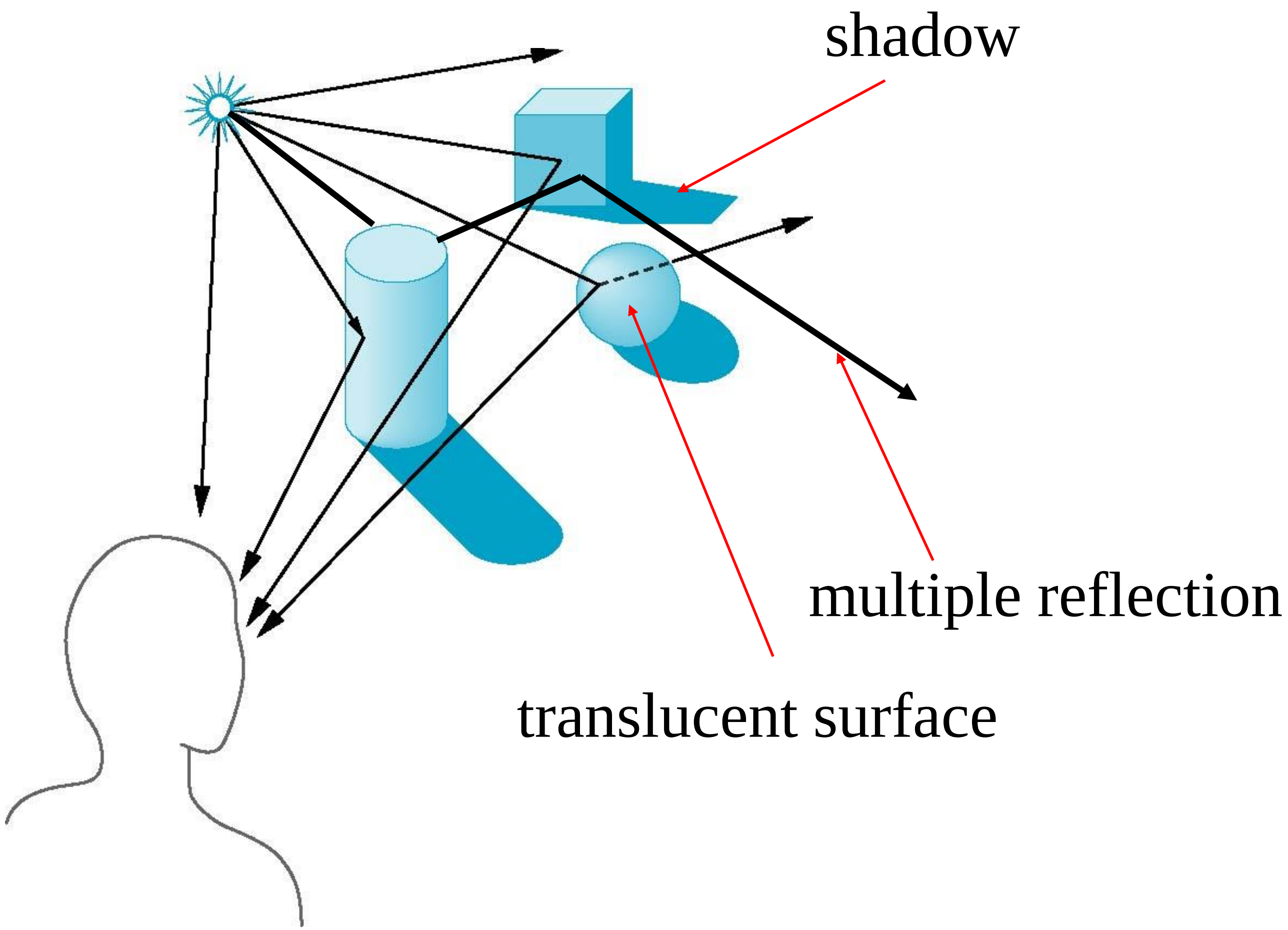
- Light strikes A
  - Some scattered
  - Some absorbed
- Some of scattered light strikes B
  - Some scattered
  - Some absorbed
- Some of this scattered light strikes A and so on



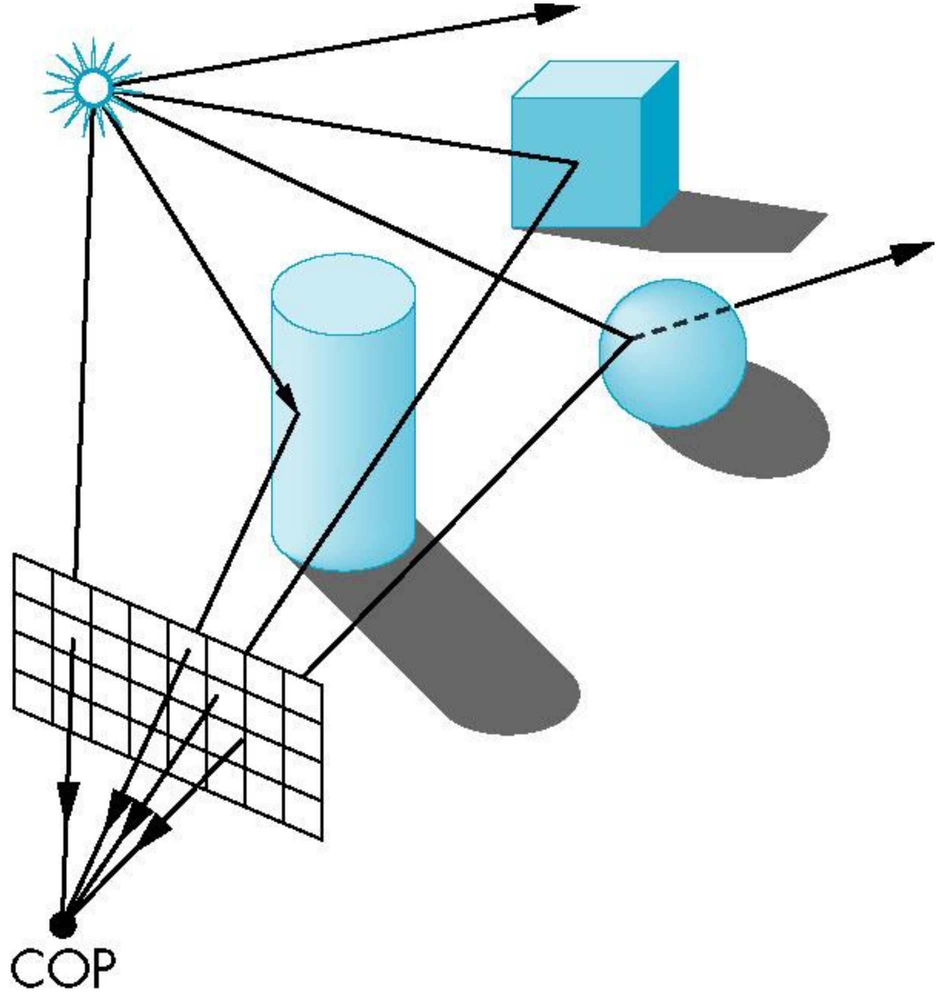
Rendering Equation

- The infinite scattering and absorption of light can be described by the *rendering equation*
  - Cannot be solved in general
  - Ray tracing is a special case for perfectly reflecting surfaces
- Rendering equation is global and includes
  - Shadows
  - Multiple scattering from object to object

Global Effects



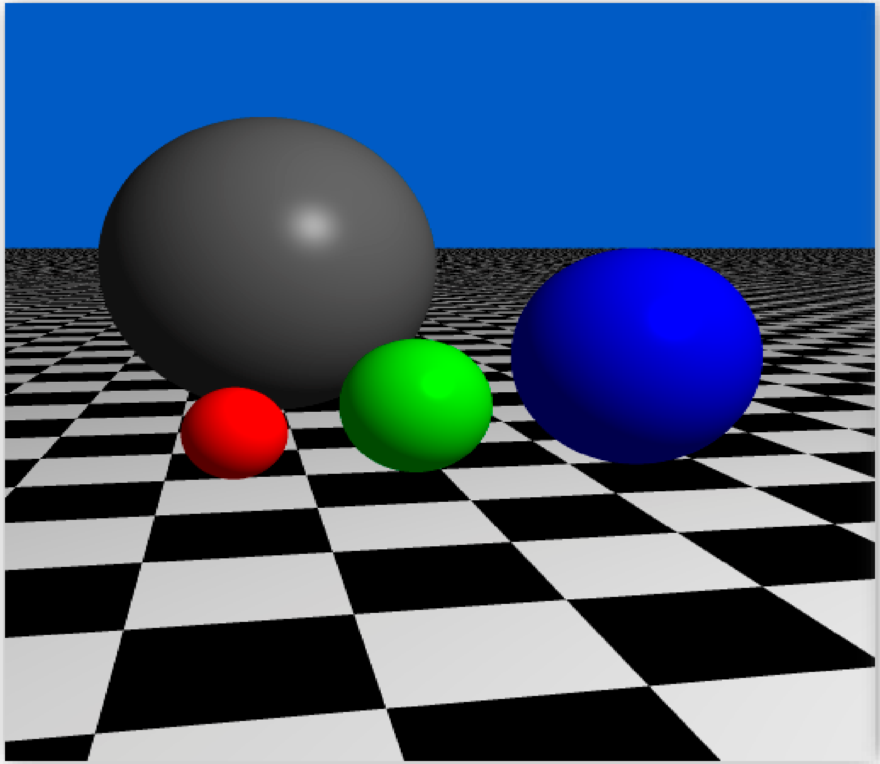
Light, Surfaces, and Computer Imaging



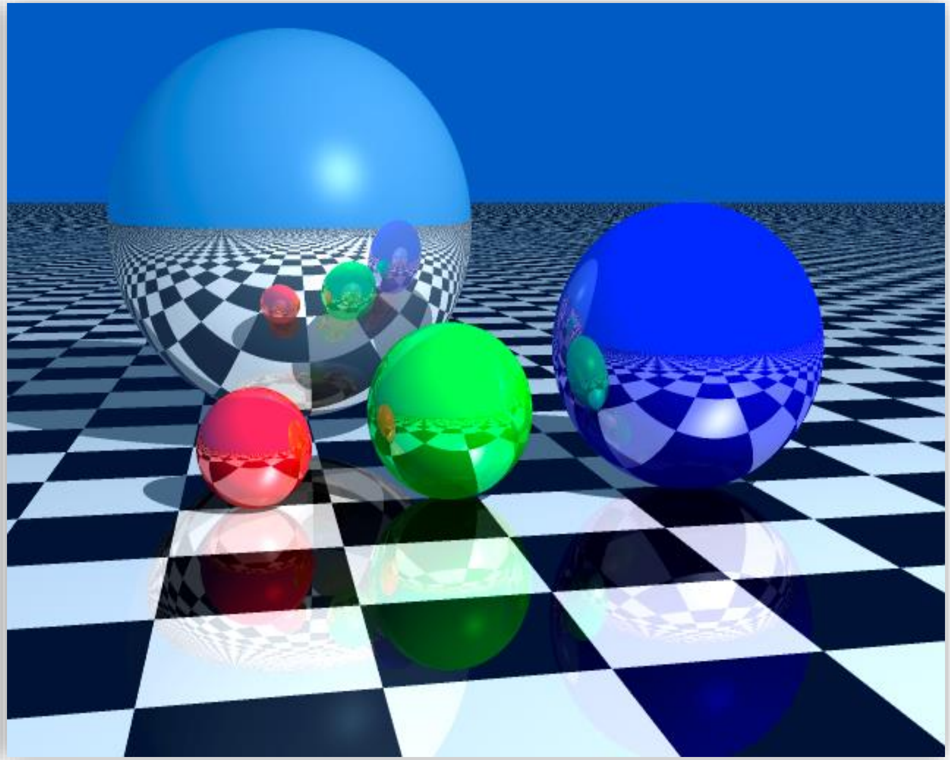
Local vs Global Rendering

- Correct shading requires a global calculation involving all objects and light sources
  - Incompatible with pipeline model which shades each polygon independently (local rendering)
- However, in computer graphics, especially real time graphics, we are happy if things “look right”
  - Exist many techniques for approximating global effects

Local Effect



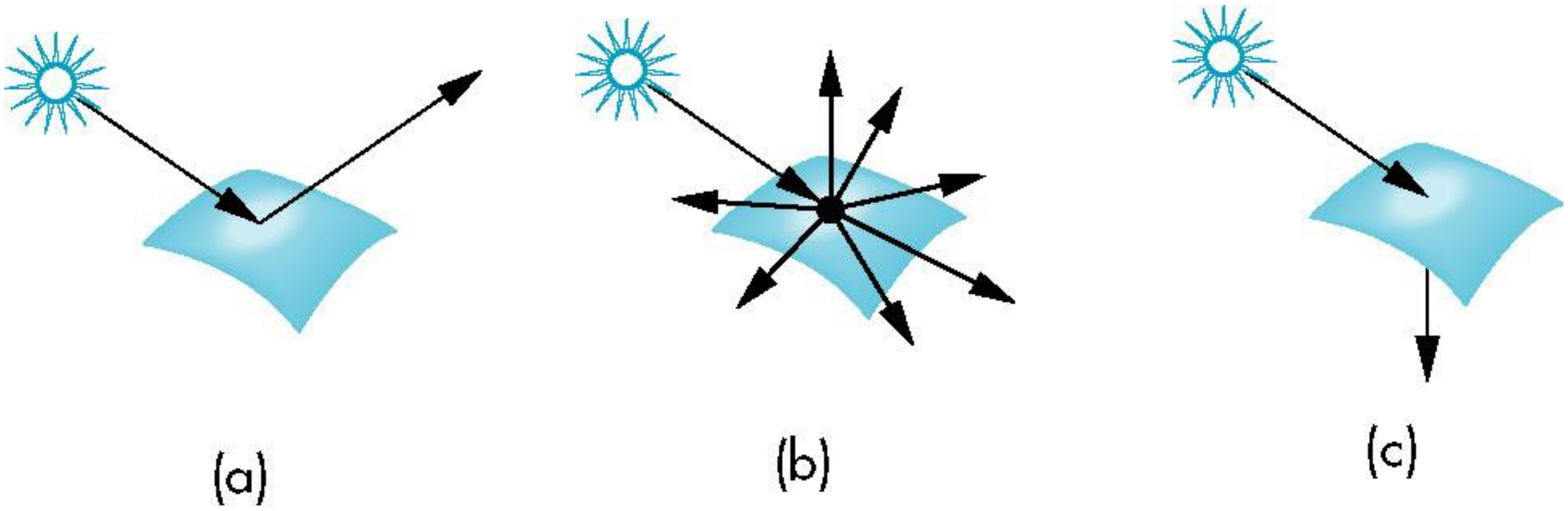
Global Effect



Light-Material Interaction

- Light that strikes an object is partially absorbed and partially scattered (reflected)
- The amount reflected determines the color and brightness of the object
  - A surface appears red under white light because the red component of the light is reflected and the rest is absorbed
- The reflected light is scattered in a manner that depends on the smoothness and orientation of the surface

Light-Material Interactions



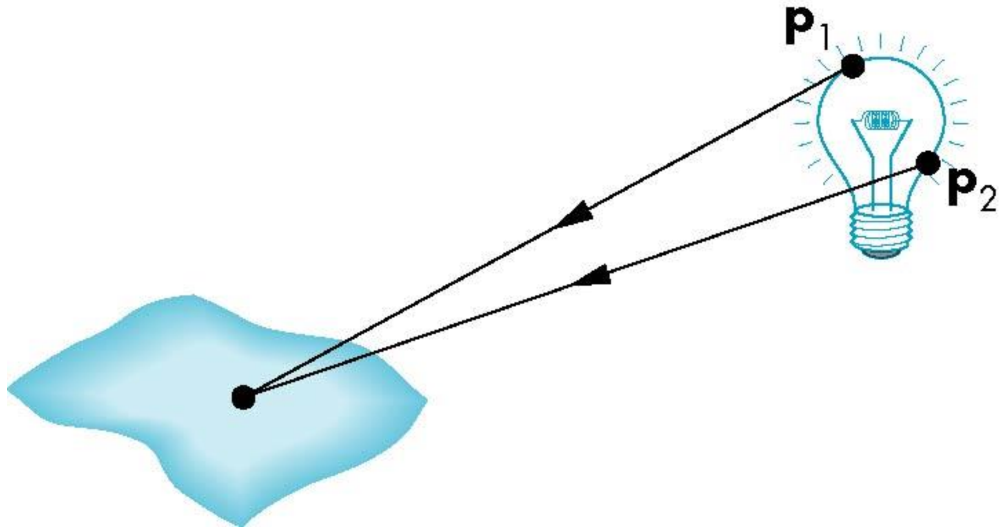
Specular surface

Diffuse surface

Translucent surface

Light Sources

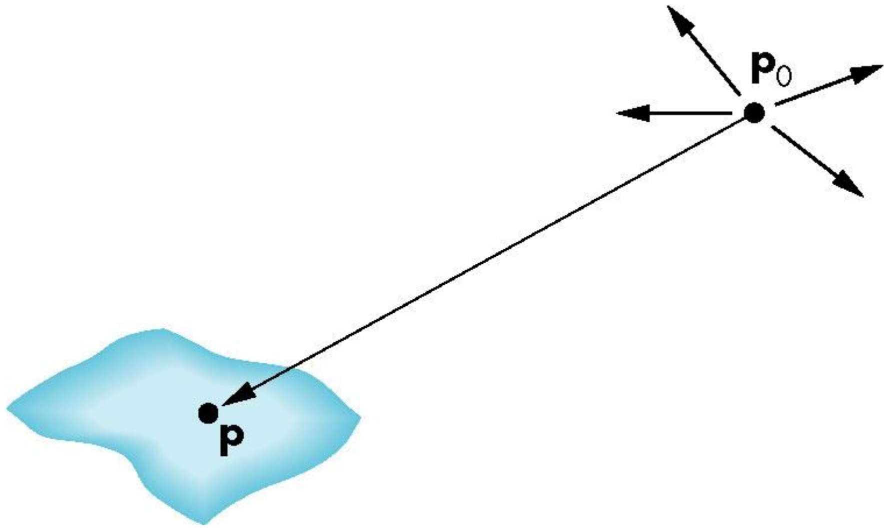
General light sources are difficult to work with because we must integrate light coming from all points on the source



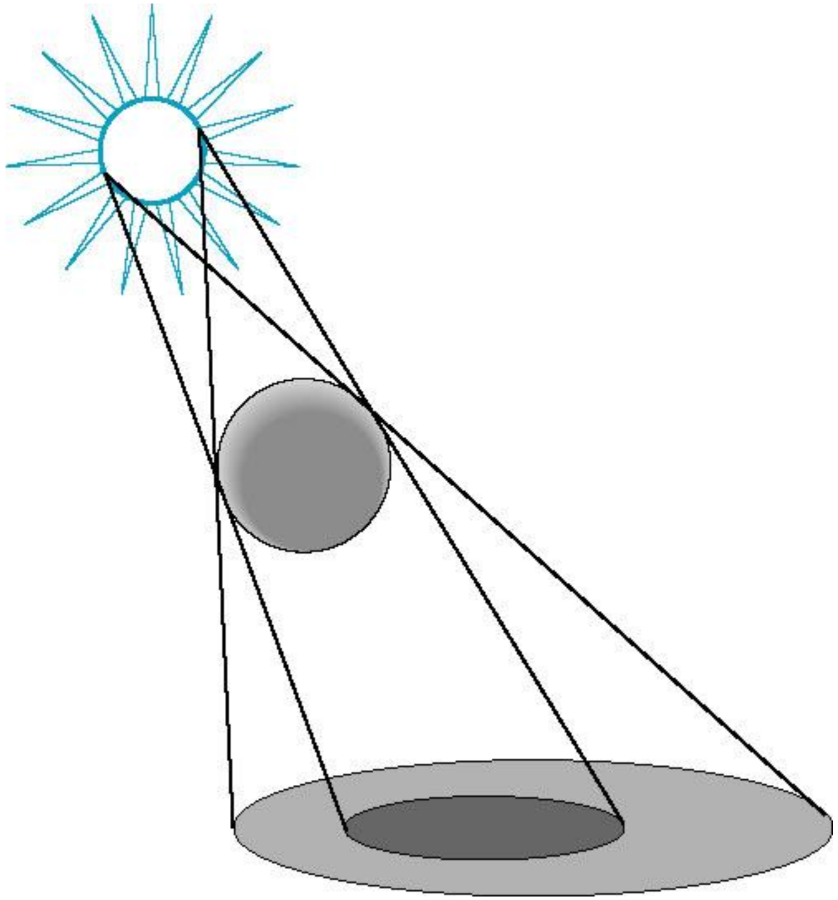
Simple Light Sources

- Point source
  - Model with position and color
  - Distant source = infinite distance away (parallel)
- Spotlight
  - Restrict light from ideal point source
- Ambient light
  - Same amount of light everywhere in scene
  - Can model contribution of many sources and reflecting surfaces

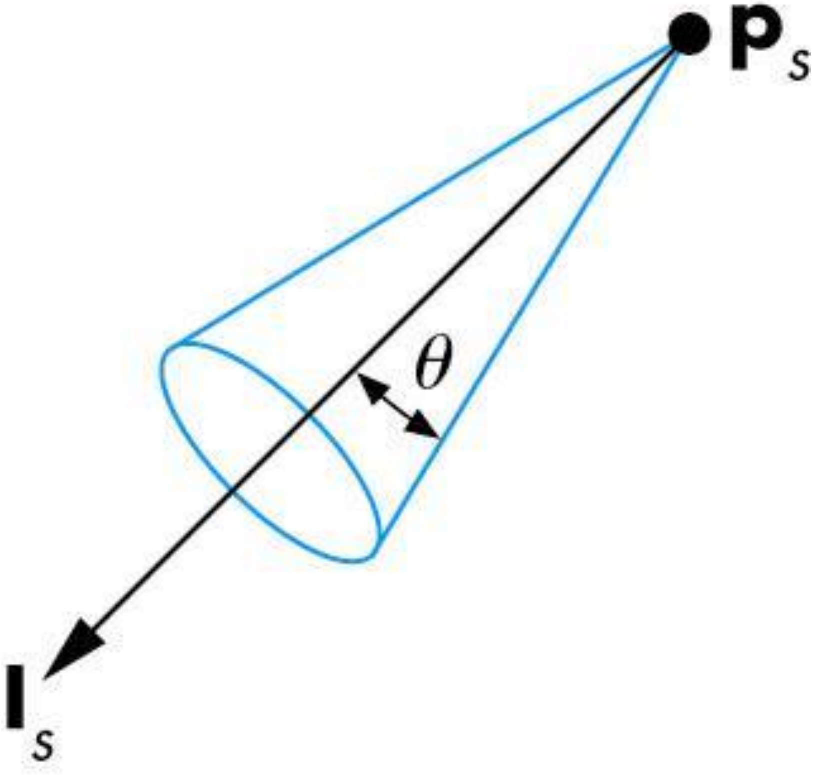
Point source illuminating a surface



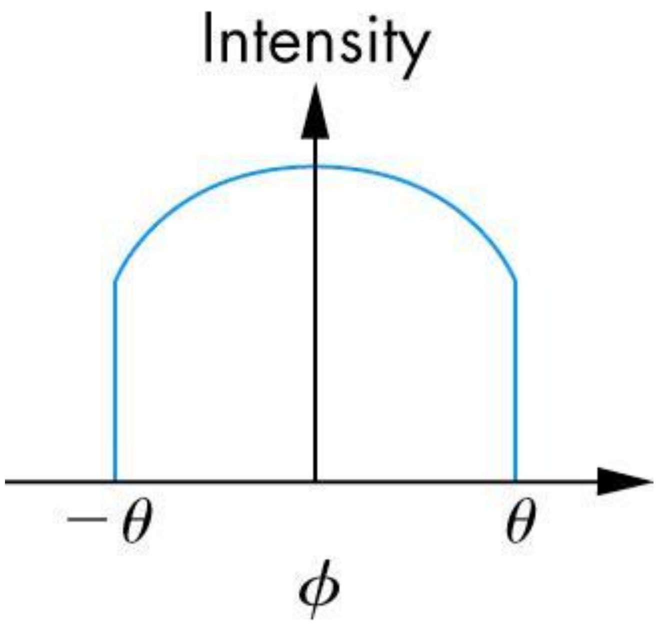
Shadow created by finite-size light source



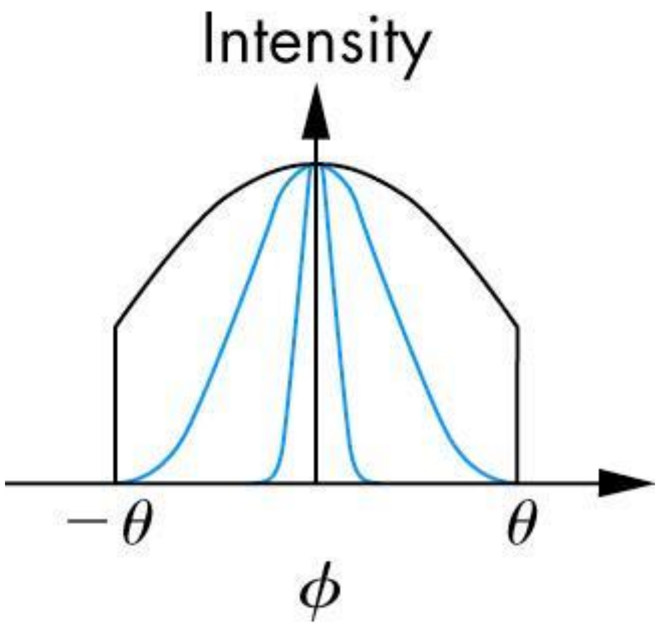
Spotlight



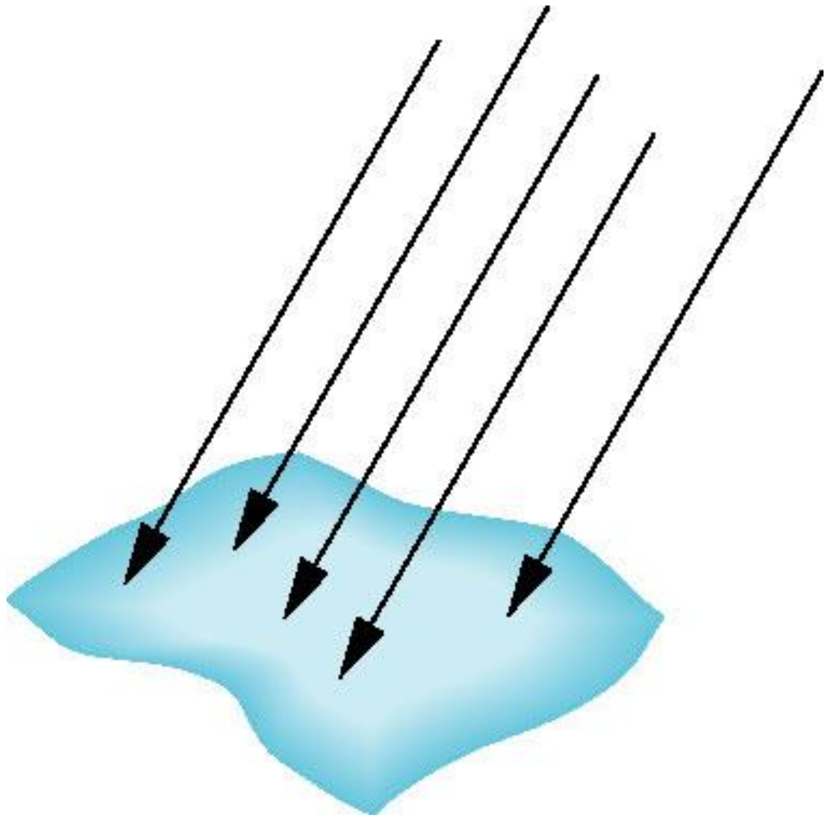
Attenuation of a spotlight



Spotlight exponent

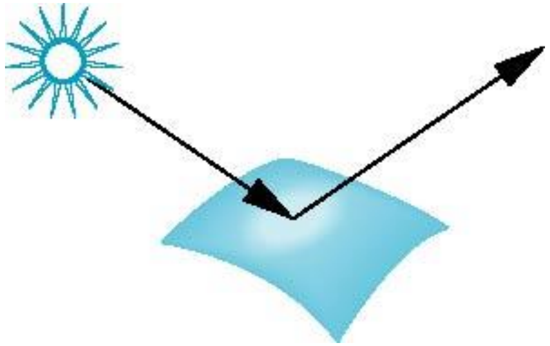


Parallel Light Source

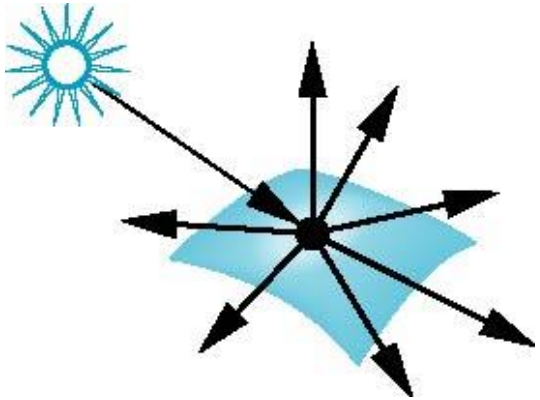


Surface Types

- The smoother a surface, the more reflected light is concentrated in the direction a perfect mirror would reflected the light
- A very rough surface scatters light in all directions



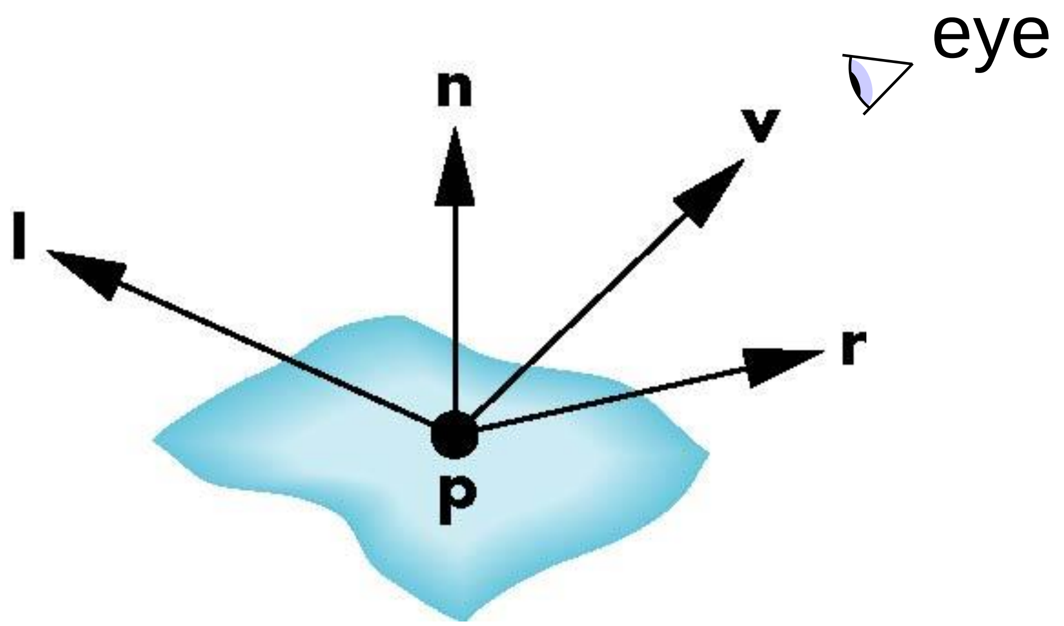
smooth surface



rough surface

Phong Model

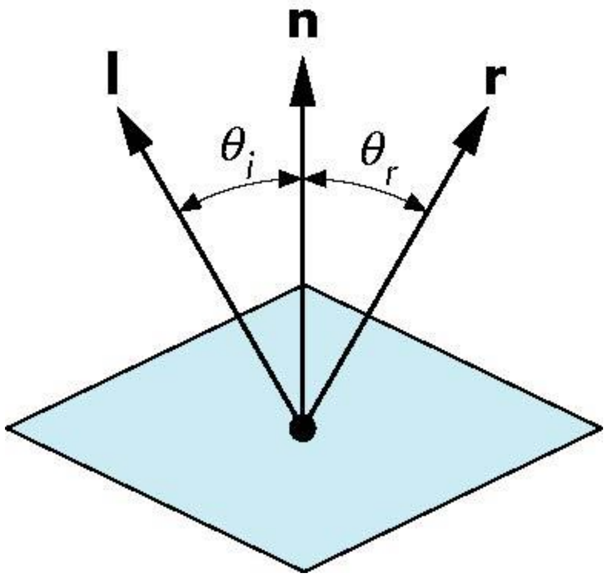
- A simple model that can be computed rapidly
- Has three components
  - Diffuse
  - Specular
  - Ambient
- Uses four vectors
  - To source
  - To viewer
  - Normal
  - Perfect reflector



Ideal Reflector

- Normal is determined by local orientation
- Angle of incidence = angle of relection
- The three vectors must be coplanar

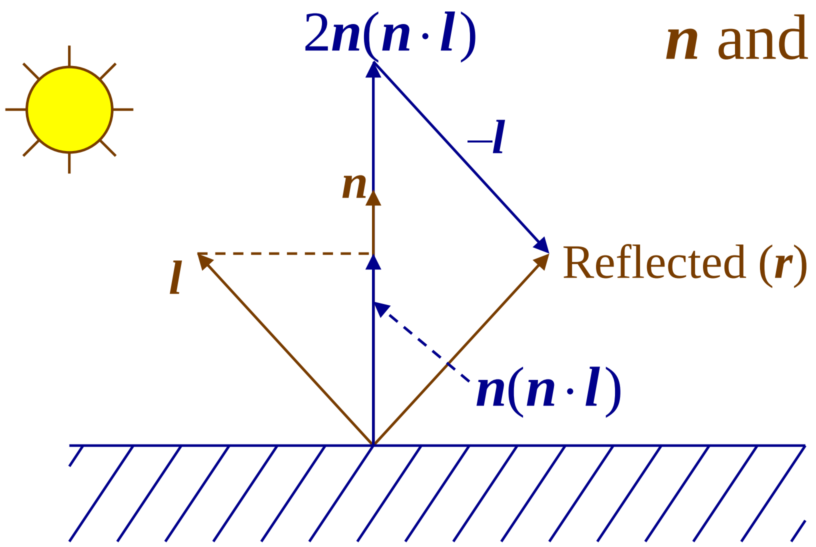
$$\mathbf{r} = 2\,(\mathbf{l} \cdot \mathbf{n})\,\mathbf{n} - \mathbf{l}$$



Phong Illumination: Computing *r*

- How can we compute the reflection vector $\boldsymbol{r}$?

n and **l** are unit vectors

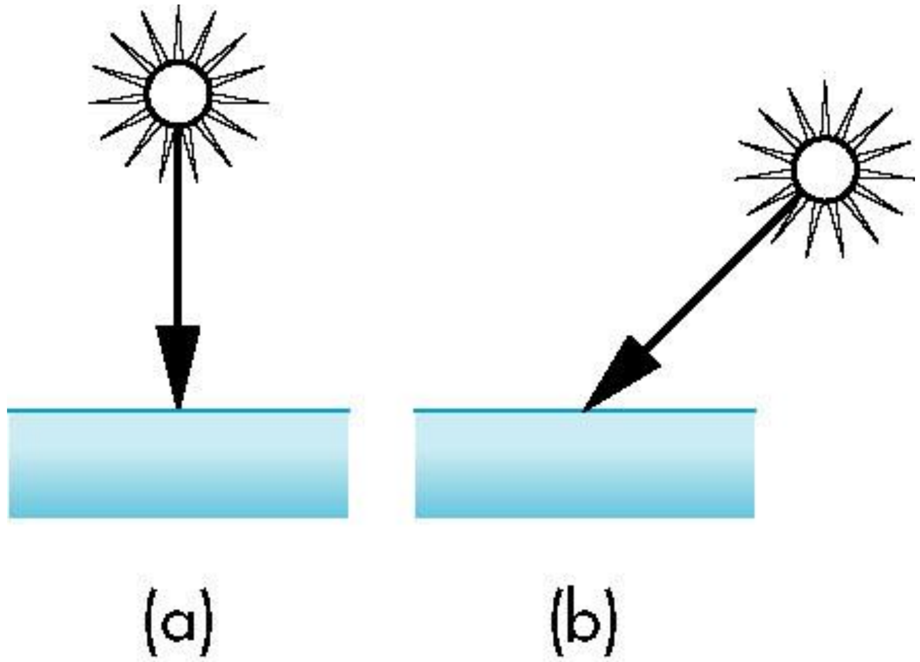


$$\boldsymbol{r} = 2\boldsymbol{n}(\boldsymbol{n}\cdot\boldsymbol{l}) - \boldsymbol{l}$$

Lambertian Surface

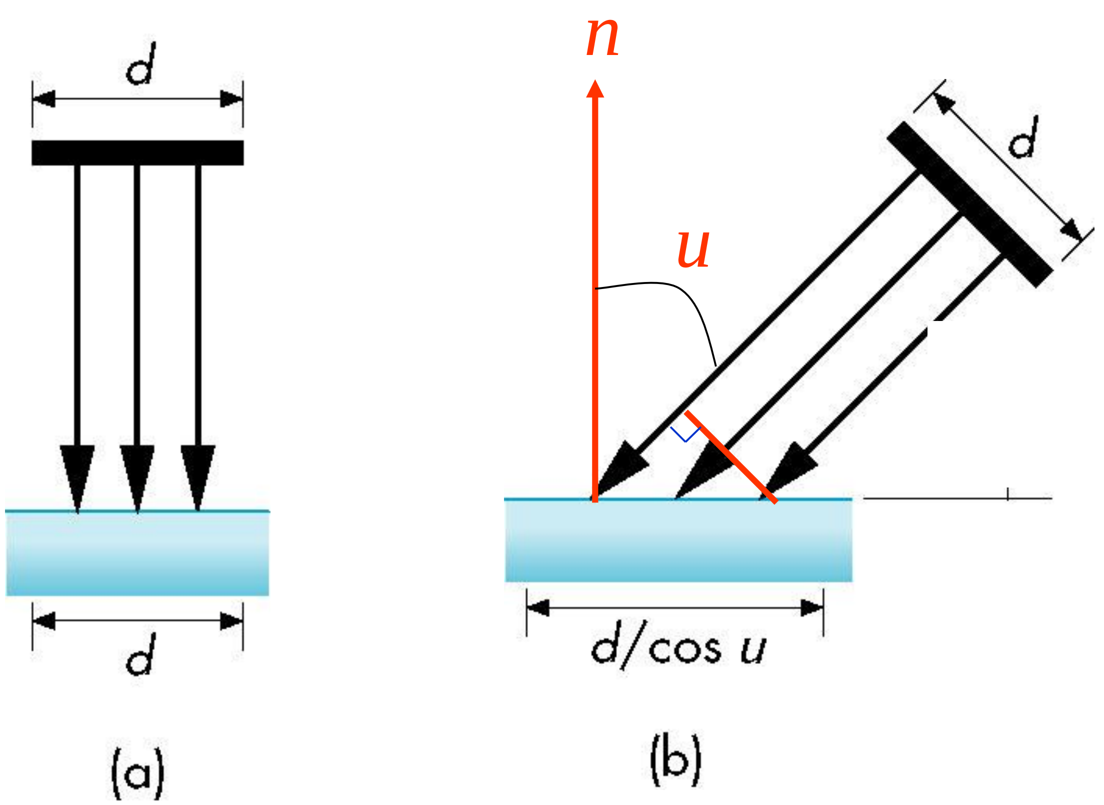
- Perfectly diffuse reflector
- Light scattered equally in all directions
- Amount of light reflected is proportional to the vertical component of incoming light
  - reflected light ~$\cos\theta_i$
  - $\cos\theta_i = \mathbf{l} \cdot \mathbf{n}$ if vectors normalized
  - There are also three coefficients, k_r, k_b, k_g that show how much of each color component is reflected

Diffuse Reflection



Illumination of a diffuse surface.
(a) at noon. (b) In the afternoon.

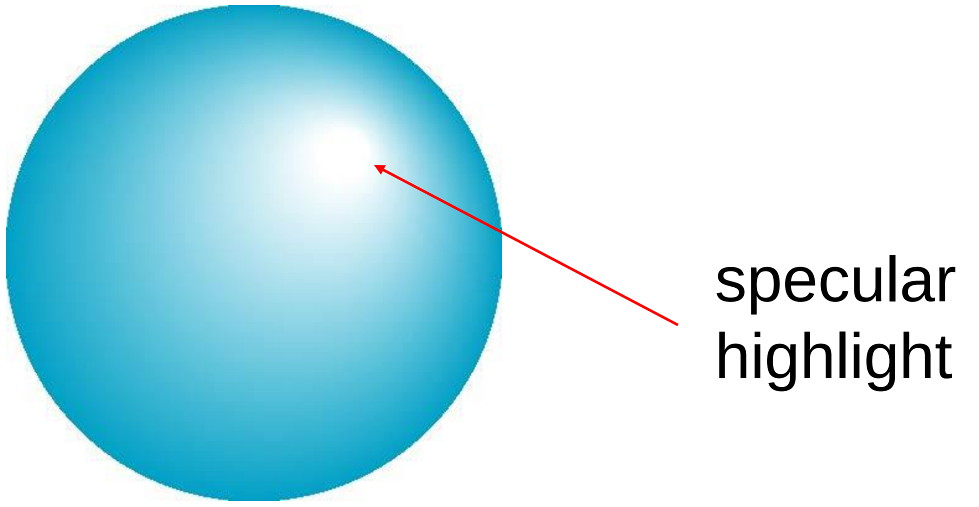
Diffuse Reflection



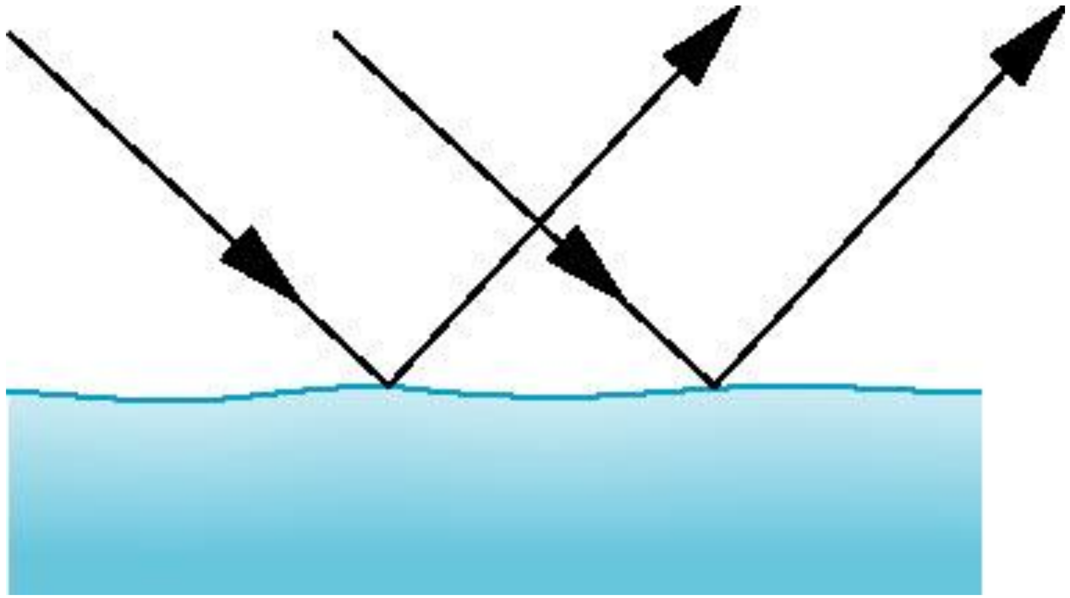
Vertical contributions by Lambert's Law.
(a) At noon. (b) In the afternoon.

Specular Surfaces

- Most surfaces are neither ideal diffusers nor perfectly specular (ideal reflectors)
- Smooth surfaces show specular highlights due to incoming light being reflected in directions concentrated close to the direction of a perfect reflection



Specular Surfaces



Modeling Specular Relections

- Phong proposed using a term that dropped off as the angle between the viewer and the ideal reflection increased

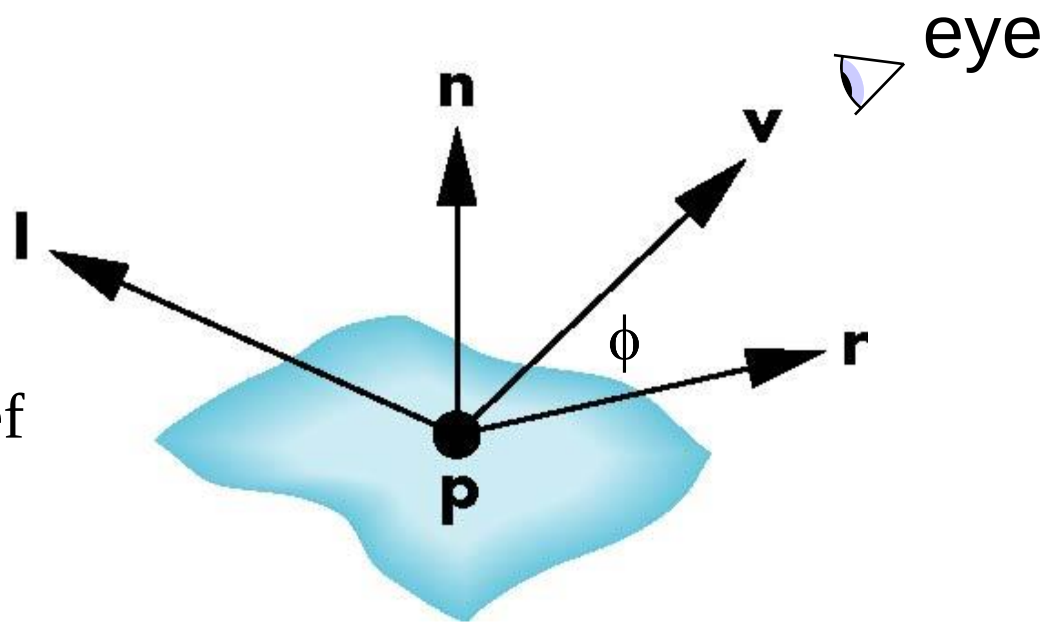
$$\mathbf{I}_r \sim k_s\, \mathbf{I} \cos^{\alpha}\phi$$

reflected intensity

absorption coef

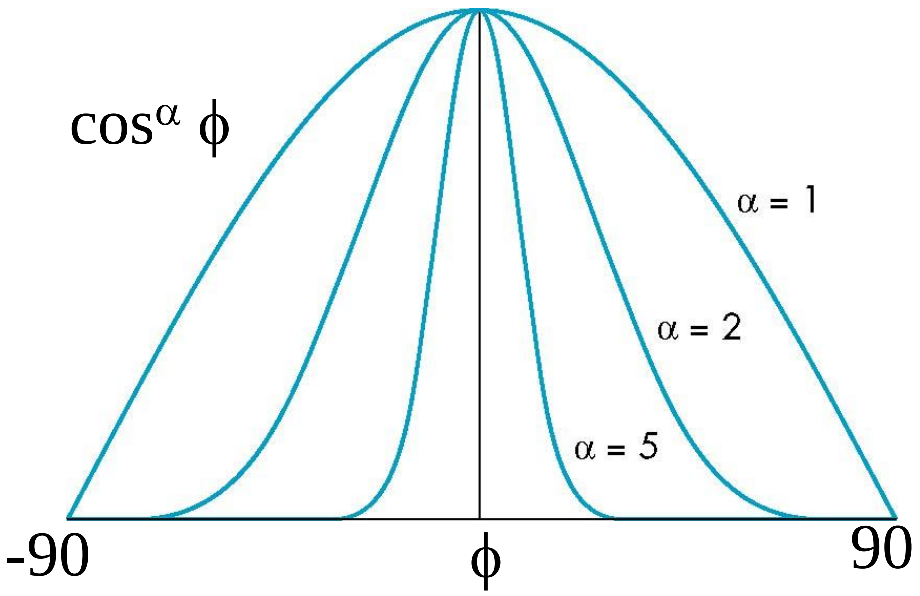
incoming intensity

shininess coef



The Shininess Coefficient

- Values of α between 100 and 200 correspond to metals
- Values between 5 and 10 give surface that look like plastic



Lighting and Shading II

Objectives

- Continue discussion of shading
- Introduce modified Phong model
- Consider computation of required vectors

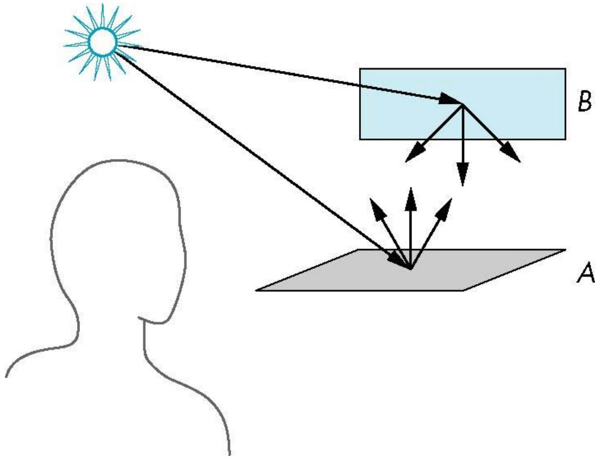
Ambient Light

- Ambient light is the result of multiple interactions between (large) light sources and the objects in the environment
- Amount and color depend on both the color of the light(s) and the material properties of the object
- Add $k_a \; I_a$ to diffuse and specular terms

reflection coef intensity of ambient light

Distance Terms

- The light from a point source that reaches a surface is inversely proportional to the square of the distance between them
- We can add a factor of the form $1/(a + bd + cd^2)$ to the diffuse and specular terms



- The constant and linear terms soften the effect of the point source

Light Sources

- In the Phong Model, we add the results from each light source
- Each light source has separate diffuse, specular, and ambient terms to allow for maximum flexibility even though this form does not have a physical justification
- Separate red, green and blue components
- Hence, 9 coefficients for each point source
  - I_{dr}, I_{dg}, I_{db}, I_{sr}, I_{sg}, I_{sb}, I_{ar}, I_{ag}, I_{ab}

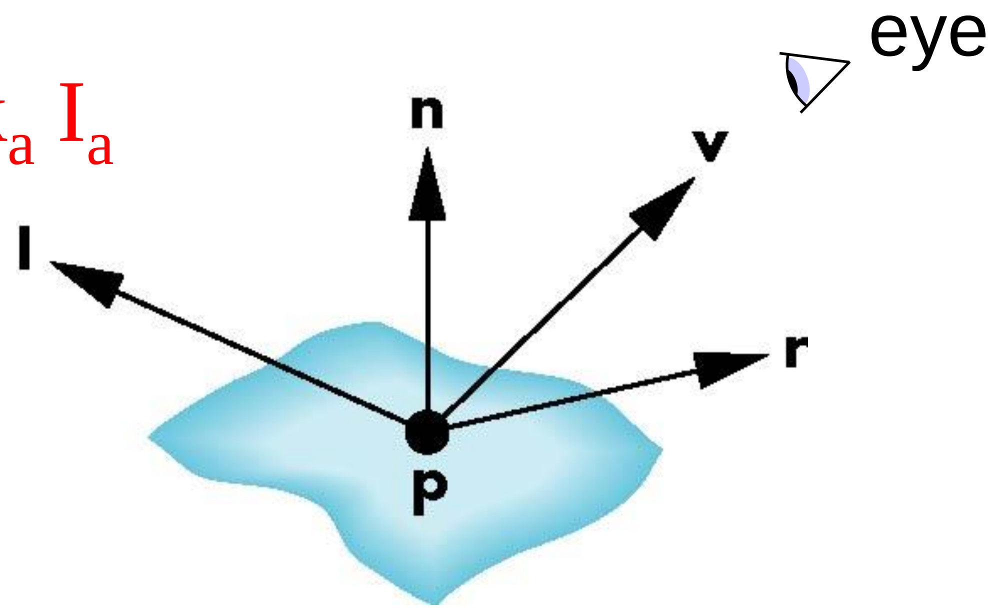
Material Properties

- Material properties match light source properties
  - Nine absorbtion coefficients
    - k_{dr}, k_{dg}, k_{db}, k_{sr}, k_{sg}, k_{sb}, k_{ar}, k_{ag}, k_{ab}
  - Shininess coefficient α

Adding up the Components

For each light source and each color component, the Phong model can be written (without the distance terms) as

$$I = k_d I_d \, \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{v} \cdot \mathbf{r})^{\alpha} + k_a I_a$$



For each color component
we add contributions from
all sources

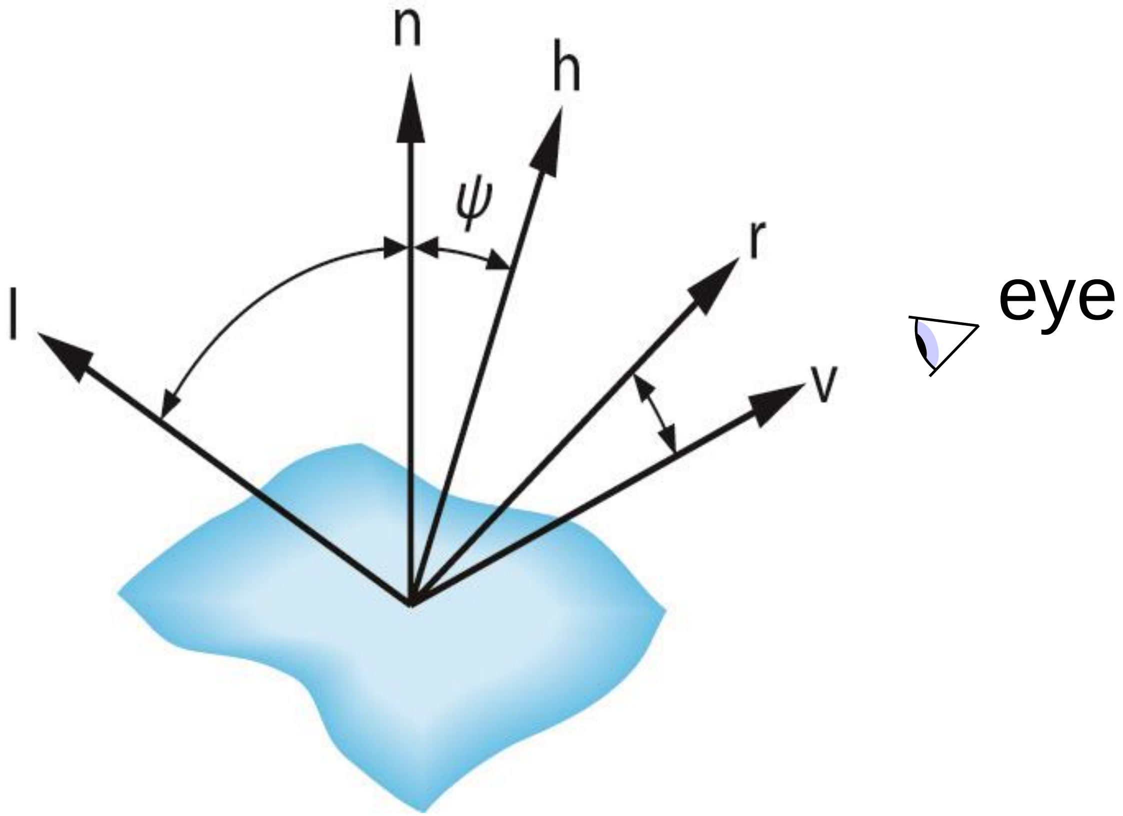
Modified Phong Model

- The specular term in the Phong model is problematic because it requires the calculation of a new reflection vector and view vector for each vertex
- Blinn suggested an approximation using the halfway vector that is *more efficient*

The Halfway Vector

- **h** is normalized vector halfway between **l** and **v**

$$\mathbf{h} = (\, \mathbf{l} + \mathbf{v}\,)/\,|\,\mathbf{l} + \mathbf{v}\,|$$



Using the halfway vector

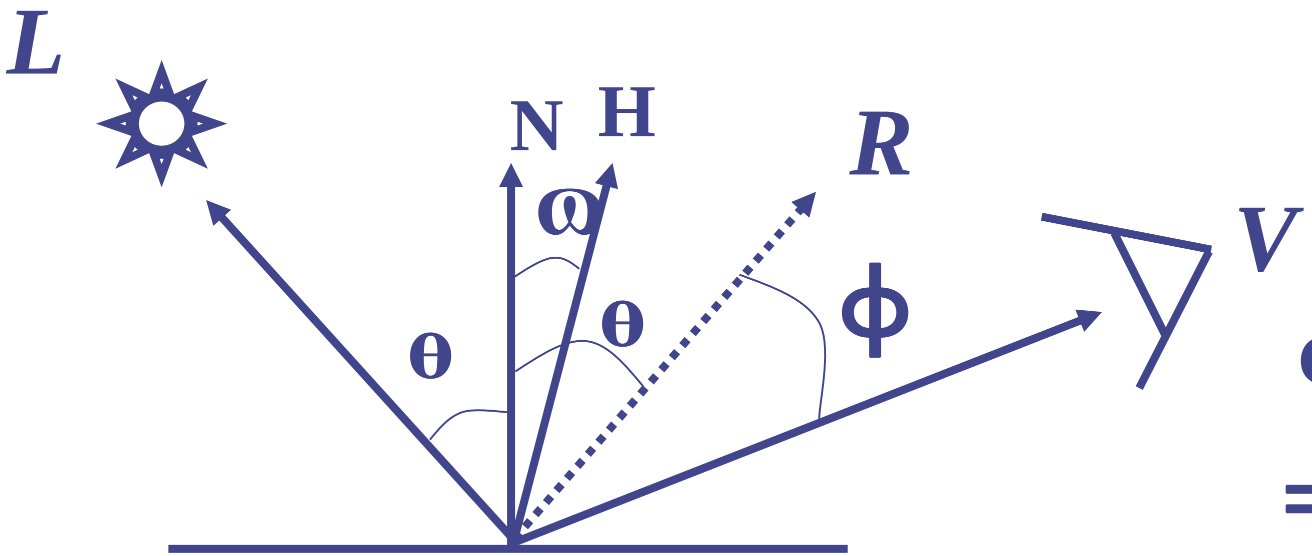
- Replace $(\mathbf{v} \cdot \mathbf{r})^{\alpha}$ by $(\mathbf{n} \cdot \mathbf{h})^{\beta}$
- β is chosen to match shininess
- Note that halfway angle is half of angle between $\mathbf{r}$ and $\mathbf{v}$ if vectors are coplanar
- Resulting model is known as the modified Phong or Phong-Blinn lighting model
  - Specified in OpenGL standard

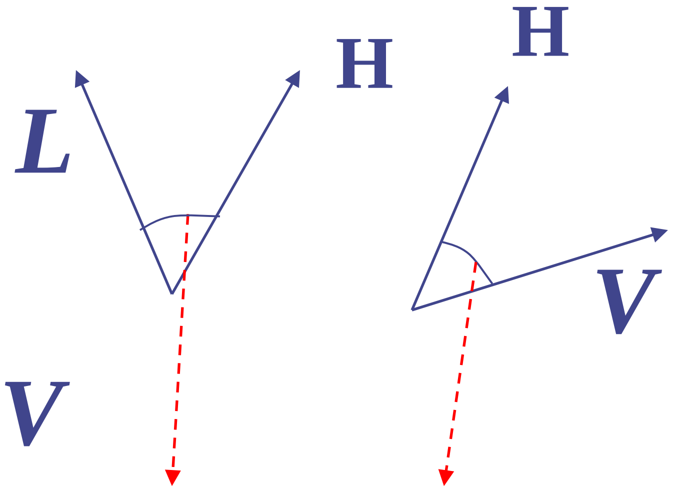
$$\boxed{I = k_d I_d \, \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{n} \cdot \mathbf{h})^{\beta} + k_a I_a}$$

Using the halfway vector (Blinn-Phong)

$(R \cdot V)^p$ *can be replaced by* $(N \cdot H)^p$ *where* $H=(L+V)/2$.

The angle between *N* and *H* is half the size of the angle between *R* and *V*.





$$\theta+\omega = (\theta-\omega)+\phi$$

$$\Rightarrow 2\omega=\phi$$

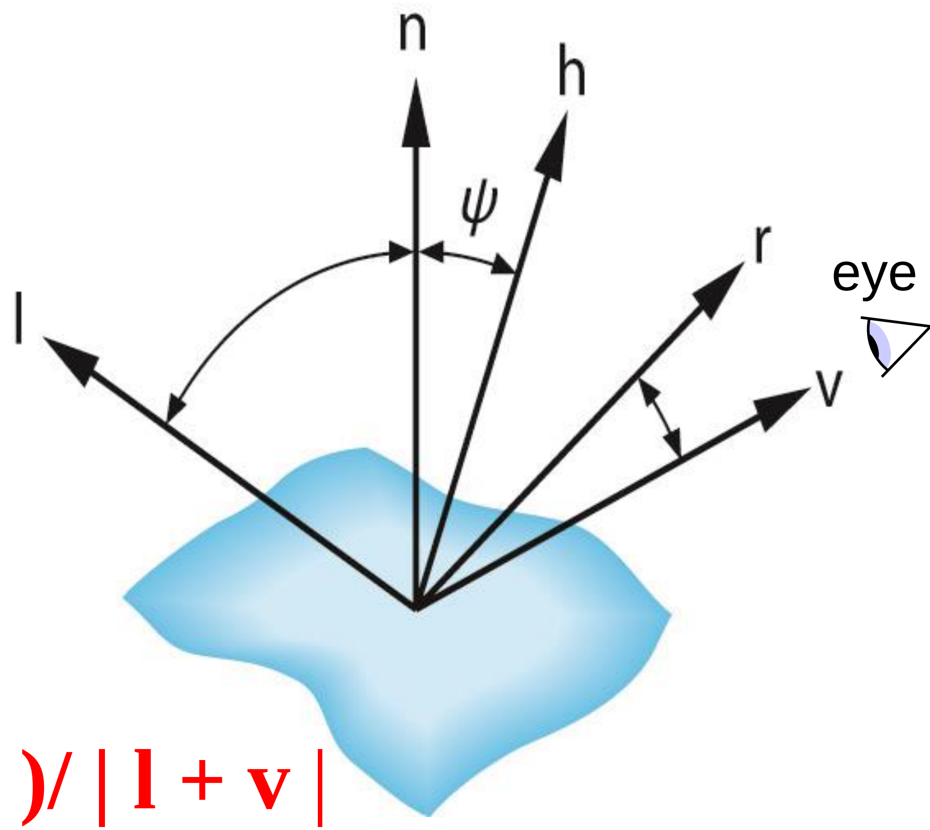
Modified Phong Model: Using the halfway vector

Phong Model

$$I = k_d I_d \, \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{v} \cdot \mathbf{r})^{\alpha} + k_a I_a$$

$$I = k_d I_d \, \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{n} \cdot \mathbf{h})^{\beta} + k_a I_a$$

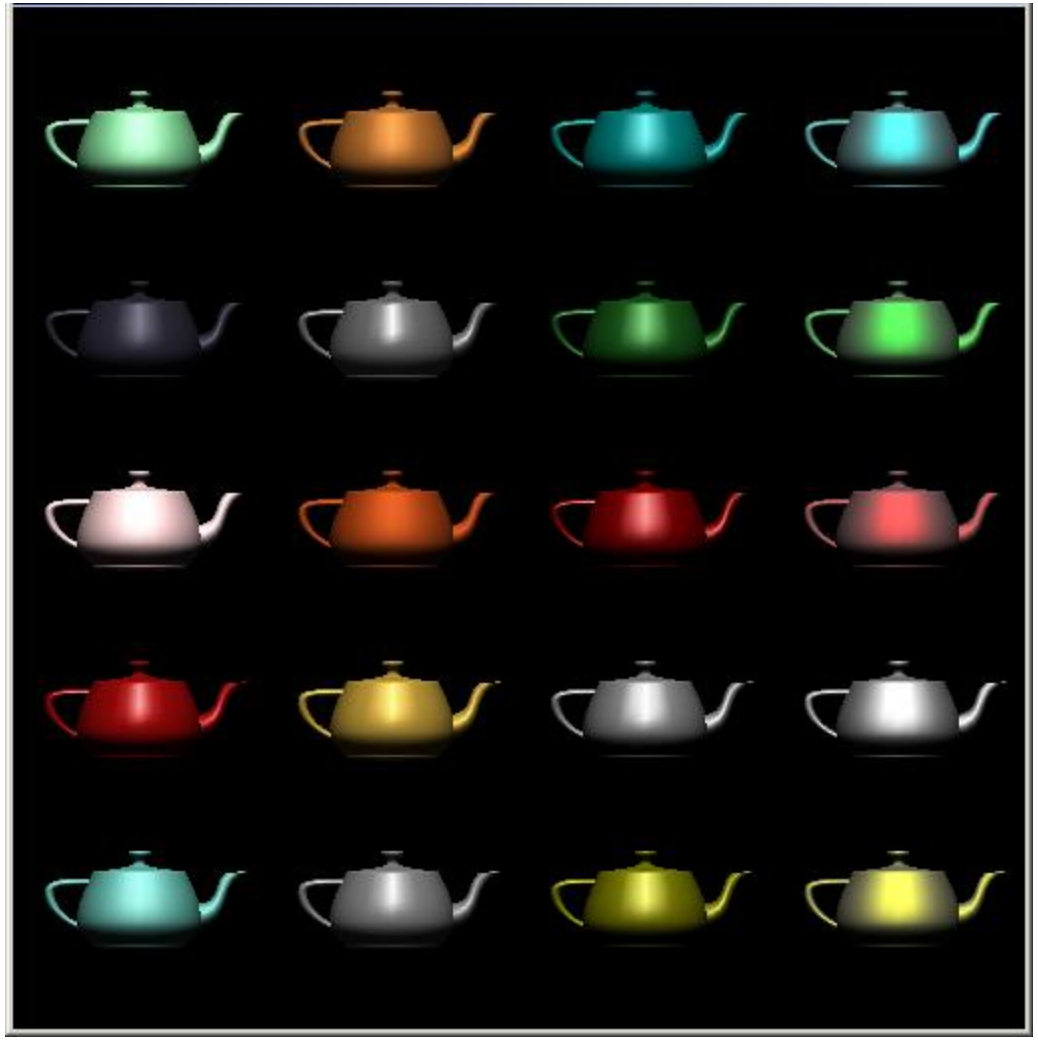
Modified Phong Model



$$\mathbf{h} = (\mathbf{l} + \mathbf{v}) / |\mathbf{l} + \mathbf{v}|$$

Example

Only differences in these teapots are the parameters in the modified Phong model



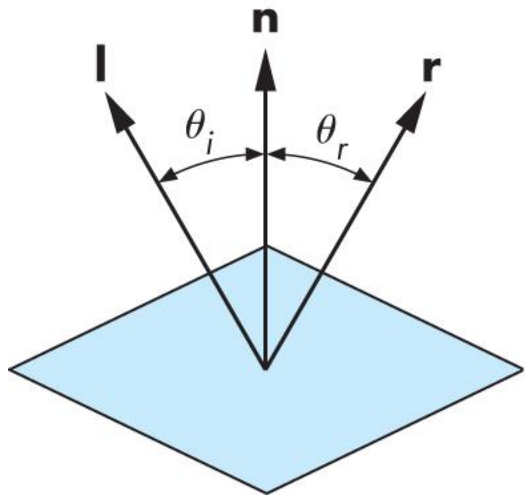
Computation of Vectors

- **l** and **v** are specified by the application
- Can computer **r** from **l** and **n**
- Problem is determining **n**
- For simple surfaces **n** can be determined but how we determine **n** differs depending on underlying representation of surface
- OpenGL leaves determination of normal to application
  - Exception for GLU quadrics and Bezier surfaces was deprecated

Computing Reflection Direction

- Angle of incidence = angle of reflection
- Normal, light direction and reflection direction are coplaner
- Want all three to be unit length

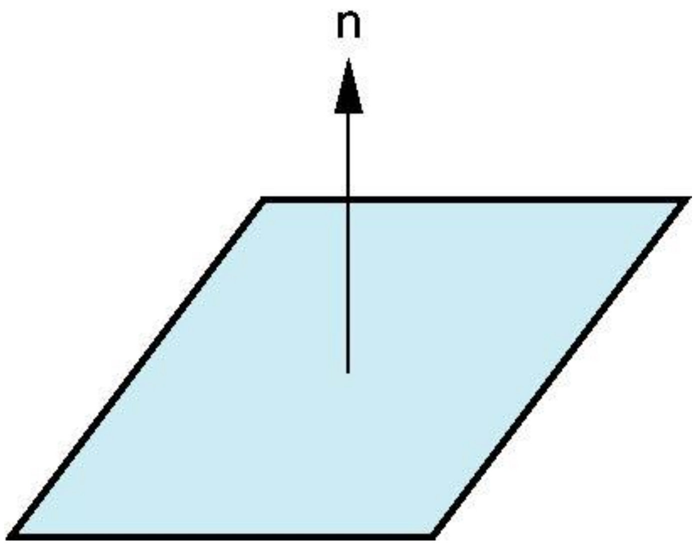
$$r = 2(l \bullet n)n - l$$



Plane Normals

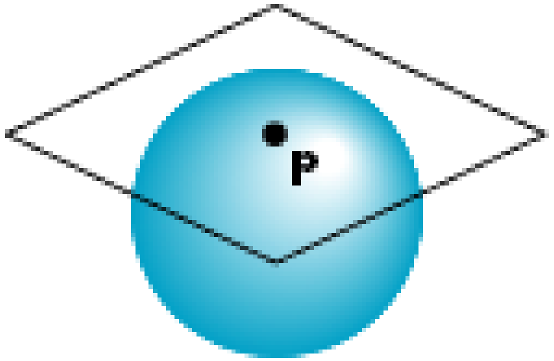
- Equation of plane: $ax+by+cz+d = 0$
- From Chapter 4 we know that plane is determined by three points p_0, p_2, p_3 or normal $\mathbf{n}$ and p_0
- Normal can be obtained by

$$\mathbf{n} = (p_2 - p_0) \times (p_1 - p_0)$$



Normal to Sphere

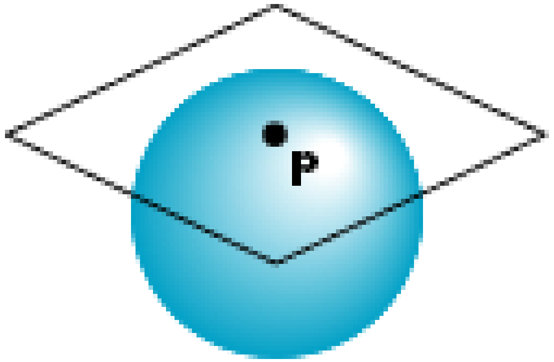
- Implicit function $f(x,y.z)=0$
- Normal given by gradient
- Sphere $f(\mathbf{p})=\mathbf{p}\cdot\mathbf{p}-1$
- $n = [\partial f/\partial x, \partial f/\partial y, \partial f/\partial z]^T=\mathbf{p}$



Parametric Form

- For sphere

$$x=x(u,v)=\cos u \sin v$$
$$y=y(u,v)=\cos u \cos v$$
$$z= z(u,v)=\sin u$$



- Tangent plane determined by vectors

$$\partial \mathbf{p}/\partial u = [\partial x/\partial u, \partial y/\partial u, \partial z/\partial u]T$$
$$\partial \mathbf{p}/\partial v = [\partial x/\partial v, \partial y/\partial v, \partial z/\partial v]T$$

- Normal given by cross product

$$\mathbf{n} = \partial \mathbf{p}/\partial u \times \partial \mathbf{p}/\partial v$$

General Case

- We can compute parametric normals for other simple cases
  - Quadrics
  - Parametric polynomial surfaces
    - Bezier surface patches (Chapter 11)

Lighting and Shading in WebGL

Objectives

- Introduce the WebGL shading methods
  - Light and material functions on MV.js
  - per vertex vs per fragment shading
  - Where to carry out

WebGL lighting

- Need
  - Normals
  - Material properties
  - Lights
- State-based shading functions have been deprecated (glNormal, glMaterial, glLight)
- Compute in application or in shaders

Normalization

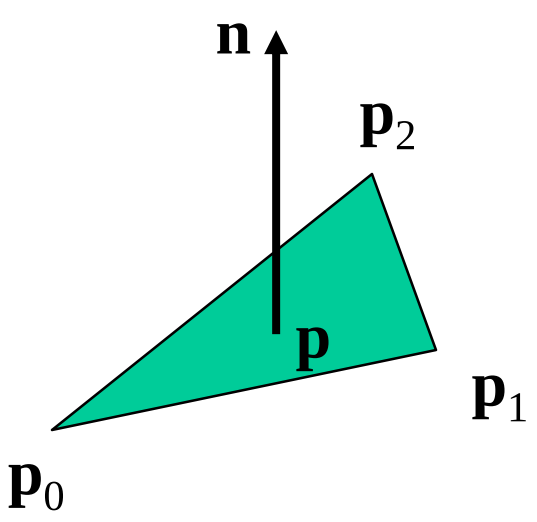
- Cosine terms in lighting calculations can be computed using dot product
- Unit length vectors simplify calculation
- Usually we want to set the magnitudes to have unit length but
    - Length can be affected by transformations
    - Note that scaling does not preserved length
- GLSL has a normalization function

Normal for Triangle

plane $\mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0$

$\mathbf{n} = (\mathbf{p}_2 - \mathbf{p}_0) \times (\mathbf{p}_1 - \mathbf{p}_0)$

normalize $\mathbf{n} \leftarrow \mathbf{n}/|\mathbf{n}|$



Note that right-hand rule determines outward face

Specifying a Point Light Source

- For each light source, we can set an RGBA for the diffuse, specular, and ambient components, and for the position

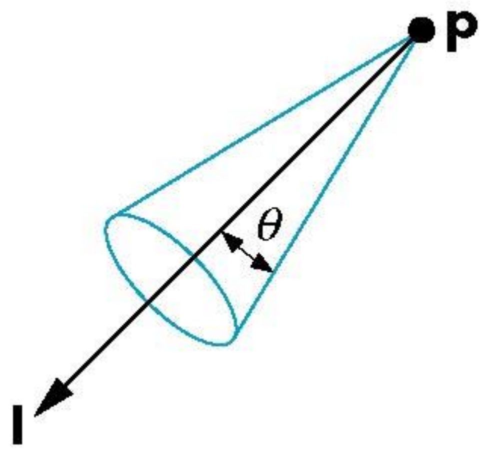
```
var diffuse0    = vec4(1.0, 0.0, 0.0, 1.0);
var ambient0  = vec4(1.0, 0.0, 0.0, 1.0);
var specular0  = vec4(1.0, 0.0, 0.0, 1.0);
var light0_pos = vec4(1.0, 2.0, 3,0, 1.0);
```

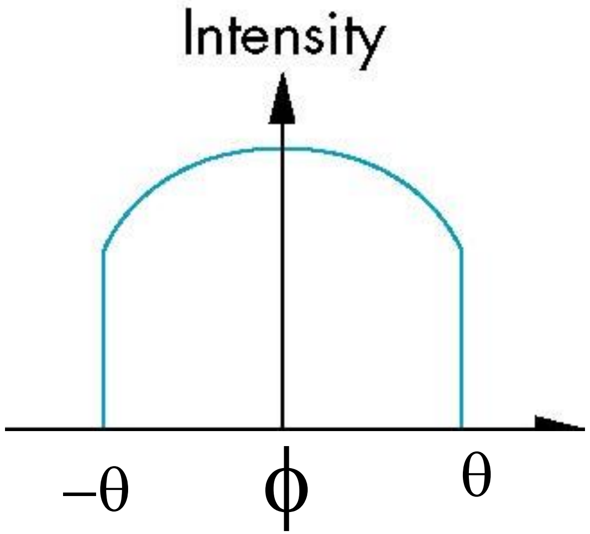
Distance and Direction

- The source colors are specified in RGBA
- The position is given in homogeneous coordinates
  - If w =1.0, we are specifying a finite location
  - If w =0.0, we are specifying a parallel source with the given direction vector
- The coefficients in distance terms are usually quadratic (1/(a+b*d+c*d*d)) where d is the distance from the point being rendered to the light source

Spotlights

- Derive from point source
  - Direction
  - Cutoff
  - Attenuation Proportional to $\cos^{\alpha}\phi$





Global Ambient Light

- Ambient light depends on color of light sources
  - A red light in a white room will cause a red ambient term that disappears when the light is turned off
- A global ambient term that is often helpful for testing

Moving Light Sources

- Light sources are geometric objects whose positions or directions are affected by the model-view matrix
- Depending on where we place the position (direction) setting function, we can
  - Move the light source(s) with the object(s)
  - Fix the object(s) and move the light source(s)
  - Fix the light source(s) and move the object(s)
  - Move the light source(s) and object(s) independently

Light Properties

```
var lightPosition  = vec4(1.0, 1.0, 1.0, 0.0 );
var lightAmbient = vec4(0.2, 0.2, 0.2, 1.0 );
var lightDiffuse   = vec4(1.0, 1.0, 1.0, 1.0 );
var lightSpecular = vec4(1.0, 1.0, 1.0, 1.0 );
```

Material Properties

- Material properties should match the terms in the light model
- Reflectivities
- w component gives opacity

```
var materialAmbient  = vec4( 1.0, 0.0, 1.0, 1.0 );
var materialDiffuse   = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialSpecular  = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialShininess = 100.0;
```

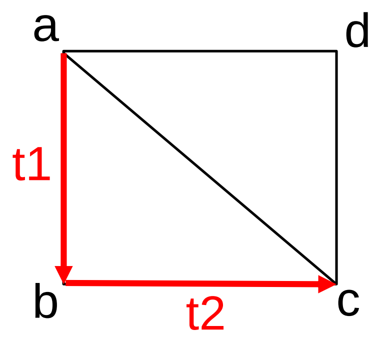
Using MV.js for Products

```
var ambientProduct = mult(lightAmbient, materialAmbient);
var diffuseProduct   = mult(lightDiffuse, materialDiffuse);
var specularProduct = mult(lightSpecular, materialSpecular);

gl.uniform4fv(gl.getUniformLocation(program, "ambientProduct"), flatten(ambientProduct));
gl.uniform4fv(gl.getUniformLocation(program, "diffuseProduct"),   flatten(diffuseProduct));
gl.uniform4fv(gl.getUniformLocation(program, "specularProduct"), flatten(specularProduct));
gl.uniform4fv(gl.getUniformLocation(program, "lightPosition"),     flatten(lightPosition));
gl.uniform1f  (gl.getUniformLocation(program,  "shininess"),        materialShininess);
```

$$\boxed{I = k_d I_d \, \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{v} \cdot \mathbf{r})^{\alpha} + k_a I_a}$$

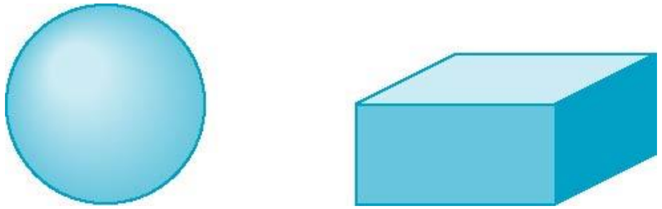
Adding Normals for Quads

```
function quad(a, b, c, d) {
    var t1 = subtract(vertices[b], vertices[a]);
    var t2 = subtract(vertices[c], vertices[b]);
    var normal = cross(t1, t2);
    var normal = vec3(normal);
    normal = normalize(normal);
      .
      .
    pointsArray.push(vertices[a]);
    normalsArray.push(normal);
     .
     .
     .
```

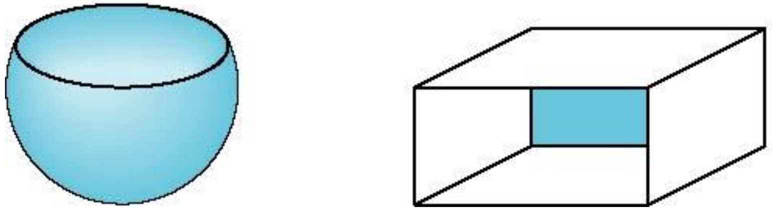


Front and Back Faces

- Every face has a front and back
- For many objects, we never see the back face so we don't care how or if it's rendered
- If it matters, we can handle in shader



back faces not visible



back faces visible

Emissive Term

- We can simulate a light source in WebGL by giving a material an emissive component
- This component is unaffected by any sources or transformations

Transparency

- Material properties are specified as RGBA values
- The A value can be used to make the surface translucent
- The default is that all surfaces are opaque
- Later we will enable blending and use this feature
- However with the HTML5 canvas, A<1 will mute colors

Polygonal Shading

Polygonal Shading

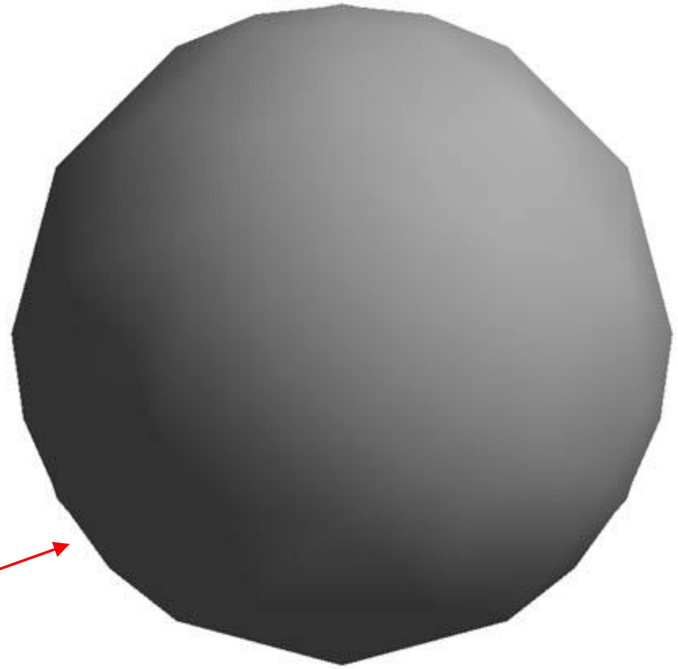
- In per vertex shading, shading calculations are done for each vertex
  - Vertex colors become vertex shades and can be sent to the vertex shader as a vertex attribute
  - Alternately, we can send the parameters to the vertex shader and have it compute the shade
- By default, vertex shades are interpolated across an object if passed to the fragment shader as a varying variable (smooth shading)
- We can also use uniform variables to shade with a single shade (flat shading)

Polygon Normals

- Triangles have a single normal
  - Shades at the vertices as computed by the modified Phong model can be almost same
  - Identical for a distant viewer (default) or if there is no specular component
- Consider model of sphere
- Want different normals at each vertex even though this concept is not quite Correct mathematically

Smooth Shading

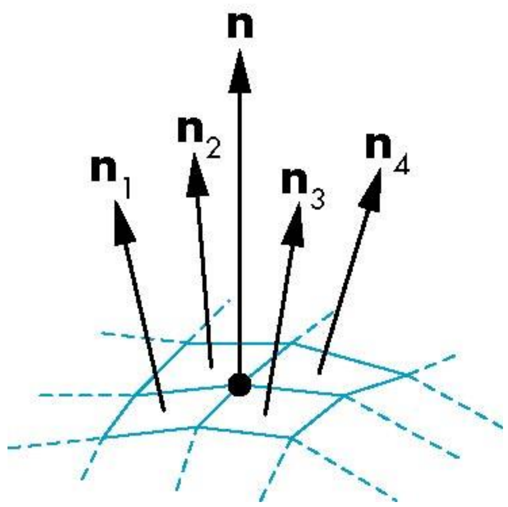
- We can set a new normal at each vertex
- Easy for sphere model
  - If centered at origin $\mathbf{n} = \mathbf{p}$
- Now smooth shading works
- Note *silhouette edge*



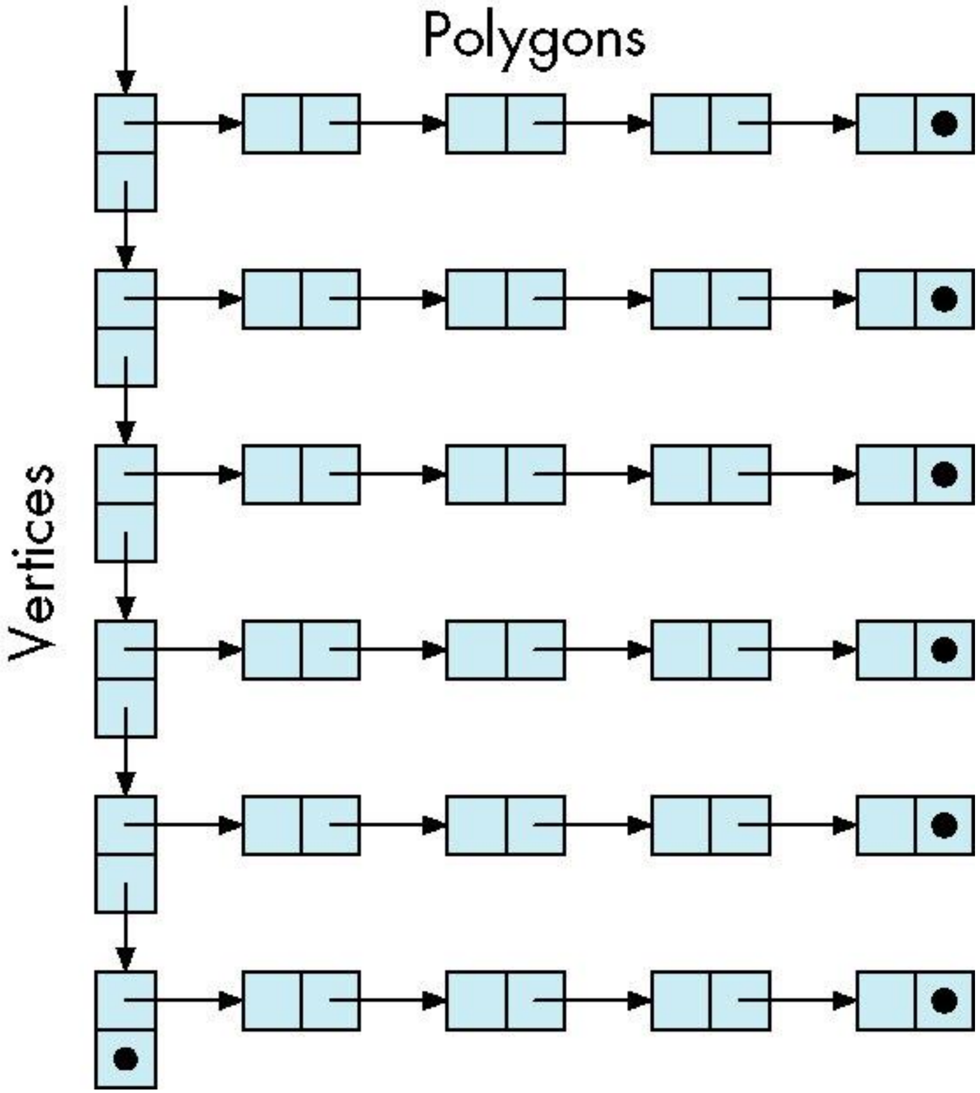
Mesh Shading

- The previous example is not general because we knew the normal at each vertex analytically
- For polygonal models, Gouraud proposed we use the average of the normals around a mesh vertex

$$\mathbf{n} = (\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4) / |\mathbf{n}_1+\mathbf{n}_2+\mathbf{n}_3+\mathbf{n}_4|$$



Mesh Data Structure

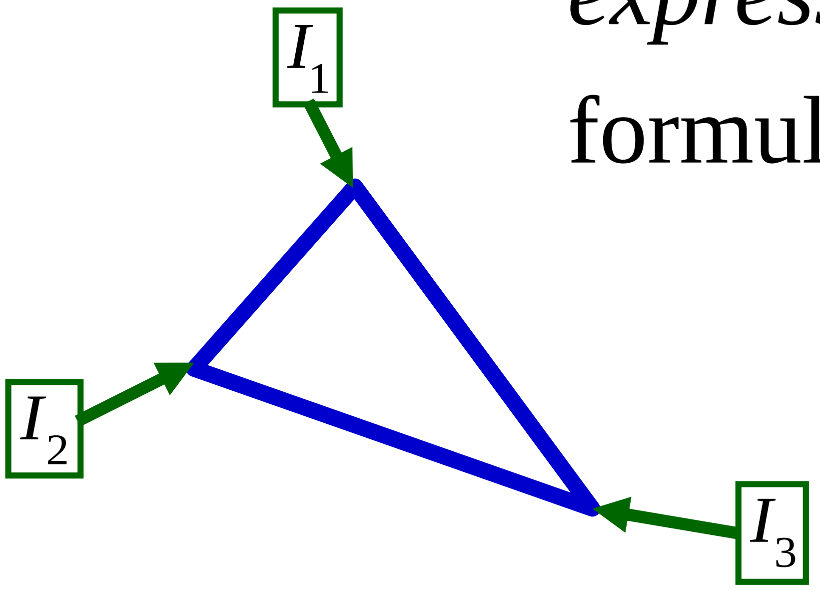


Gouraud and Phong Shading

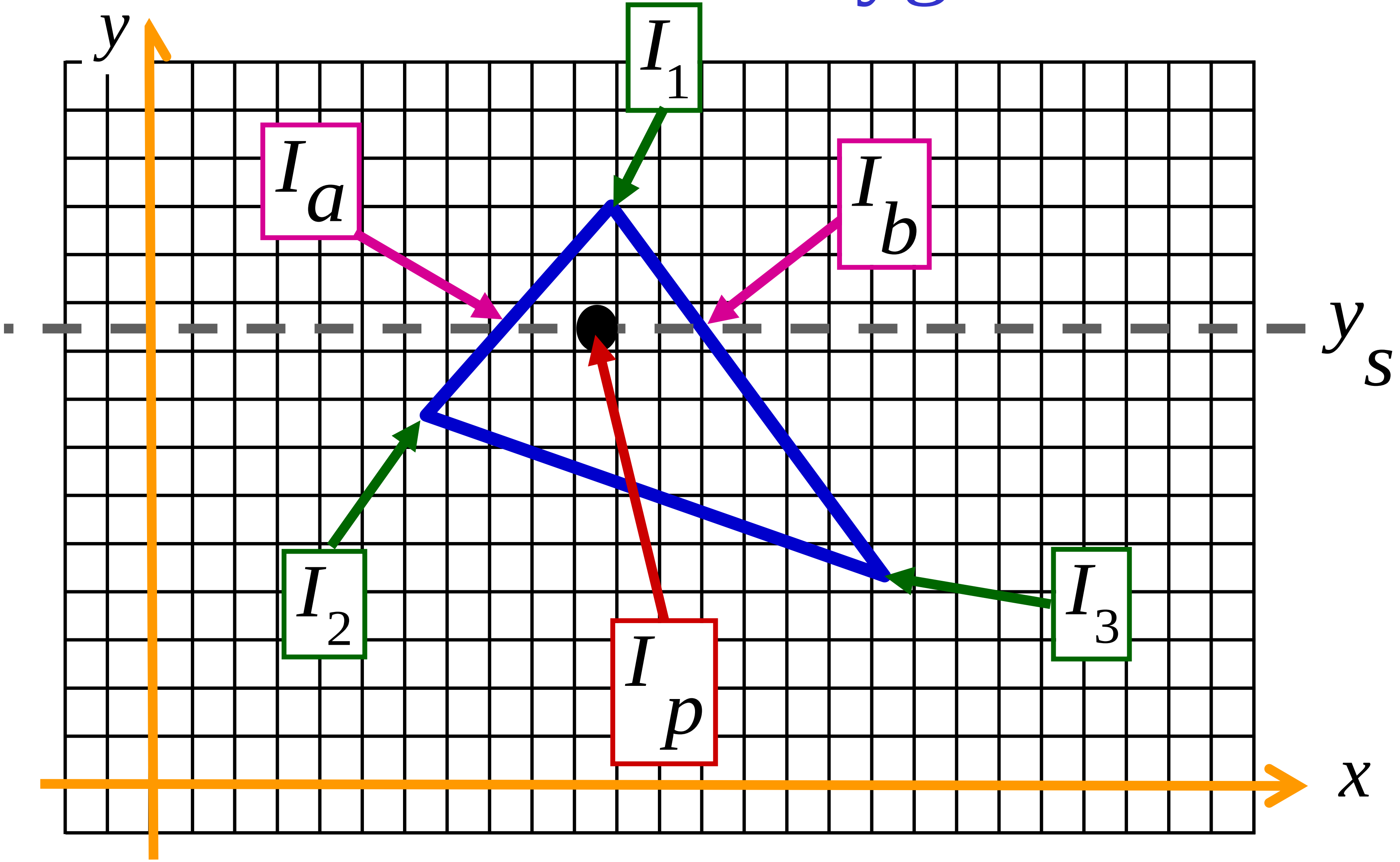
- Gouraud Shading
  - Find average normal at each vertex (vertex normals)
  - Apply modified Phong model at each vertex
  - Interpolate vertex shades across each polygon
- Phong shading
  - Find vertex normals
  - Interpolate vertex normals across edges
  - Interpolate edge normals across polygon
  - Apply modified Phong model at each fragment

Gouraud Shading: Intensity Interpolation

I_1, I_2, I_3： Compute by direction evaluation of *illumination expression*, whichever formula is being used

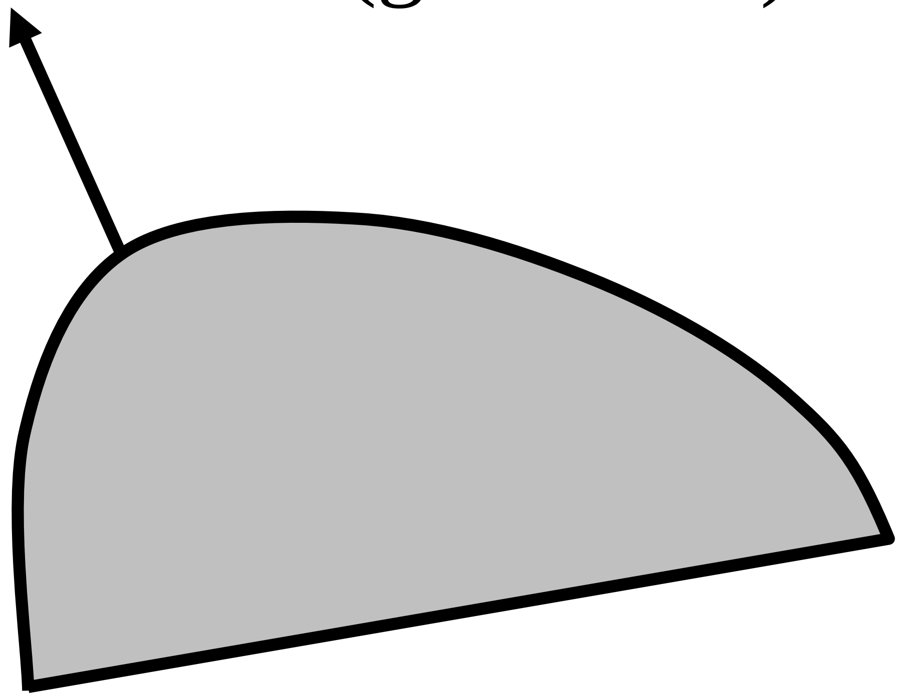


Scan Convert Polygon P

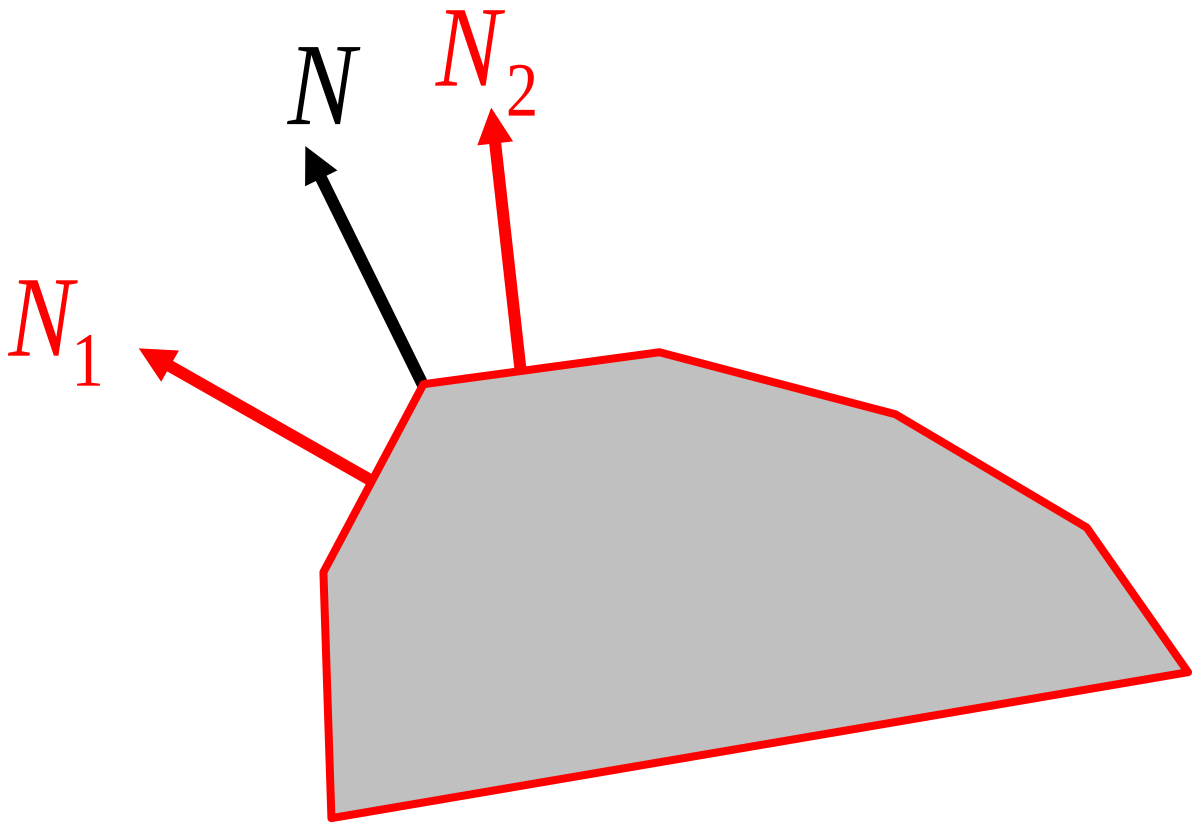


Using Average Normals

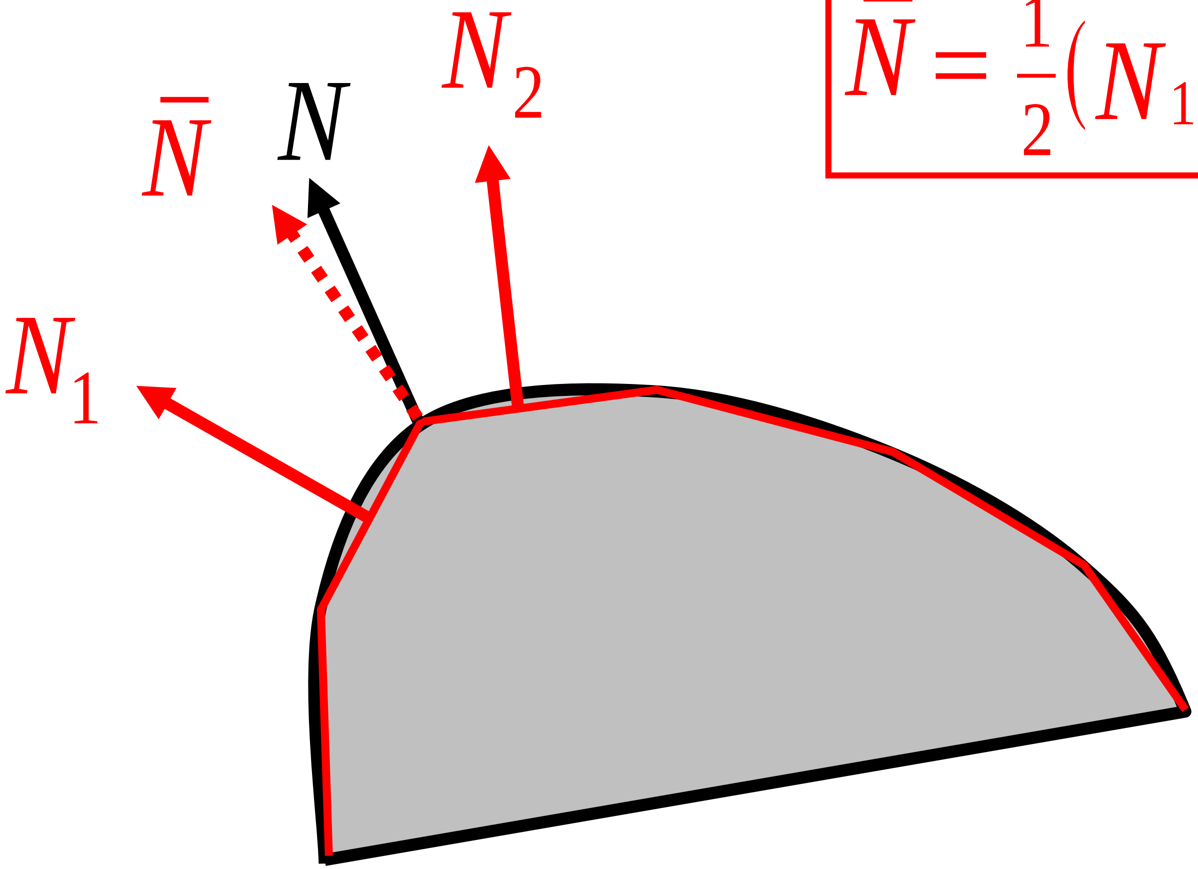
N = true (geometric) normal



Using Average Normals



Using Average Normals



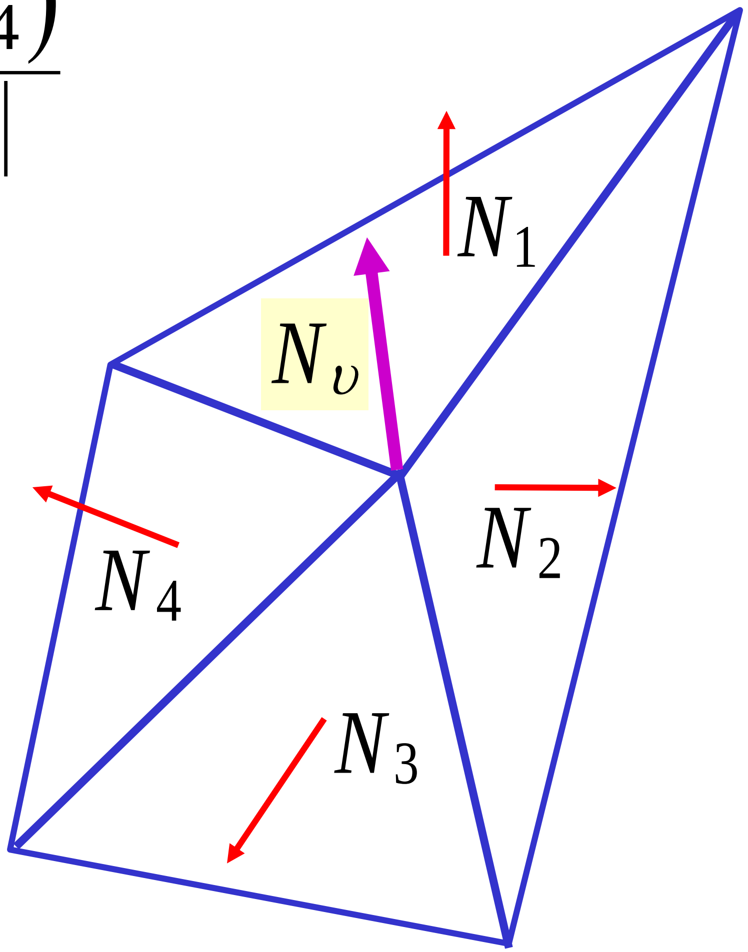
$$\boxed{\bar{N} = \frac{1}{2}(N_1 + N_2)}$$

What should corner normals be?

$$N_{\upsilon} = \frac{(N_1 + N_2 + N_3 + N_4)}{\|N_1 + N_2 + N_3 + N_4\|}$$

More generally,

$$\boxed{N_{\upsilon} = \frac{\sum_{i=1}^{n} N_i}{\left|\sum_{i=1}^{n} N_i\right|}}$$

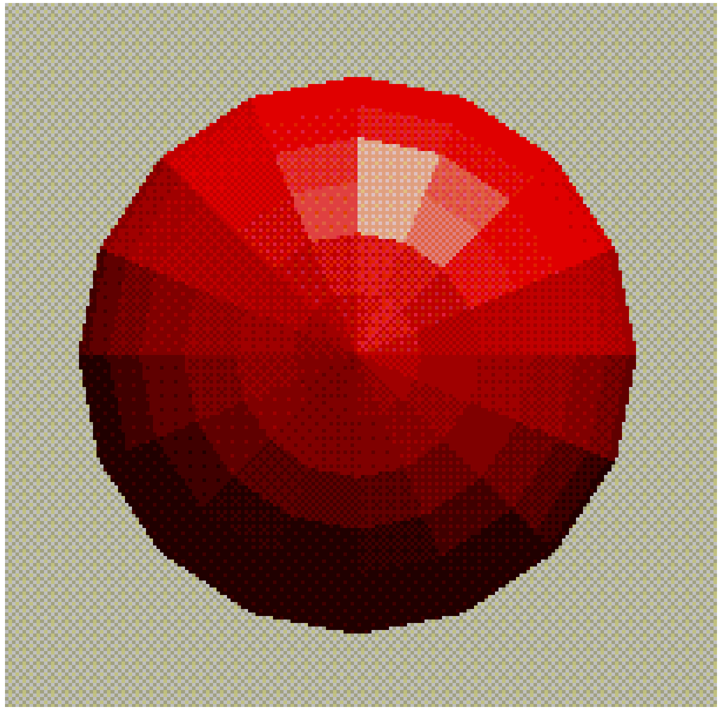


Types of Shading

- There are several well-known / commonly-used shading methods
  - Flat shading
  - Gouraud shading
  - Phong shading

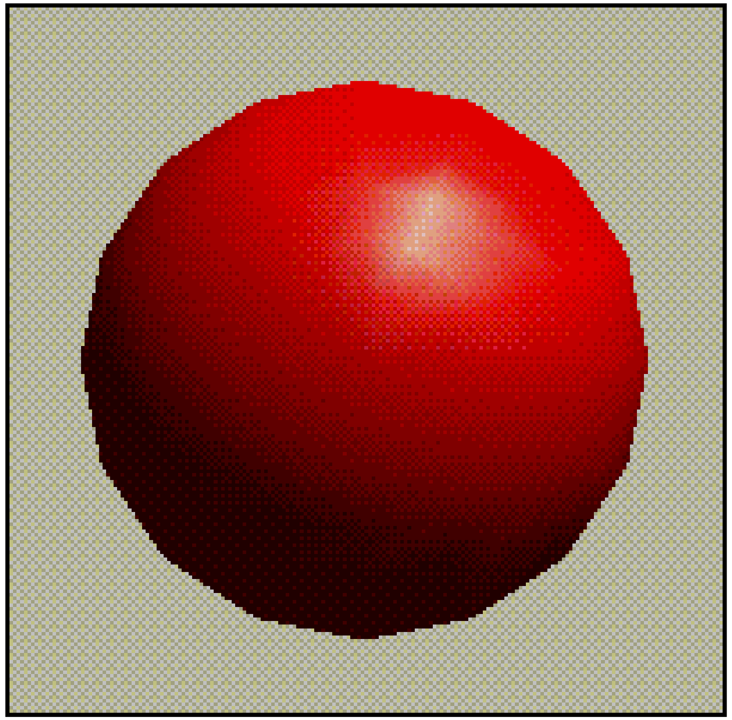
Shading Schemes

Flat Shading: same shade to entire polygon



Shading Schemes

Gouraud Shading: smoothly blended intensity across each polygon



Shading Schemes

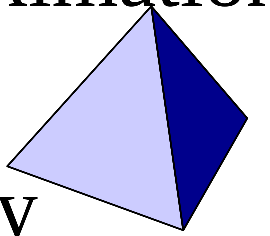
Phong Shading:

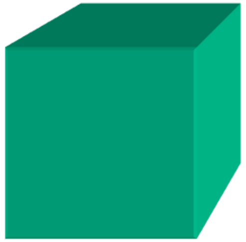
interpolated *normals* to compute intensity at each point

Bui Toung Phong Thesis

Flat Shading

- Compute 1 normal for the polygon
- Assumes light (and viewer if using Phong illumination model) are at infinity
- Assumes polygon exactly represents the actual surface, not an approximation:
  - i.e. cube vs. cylinder



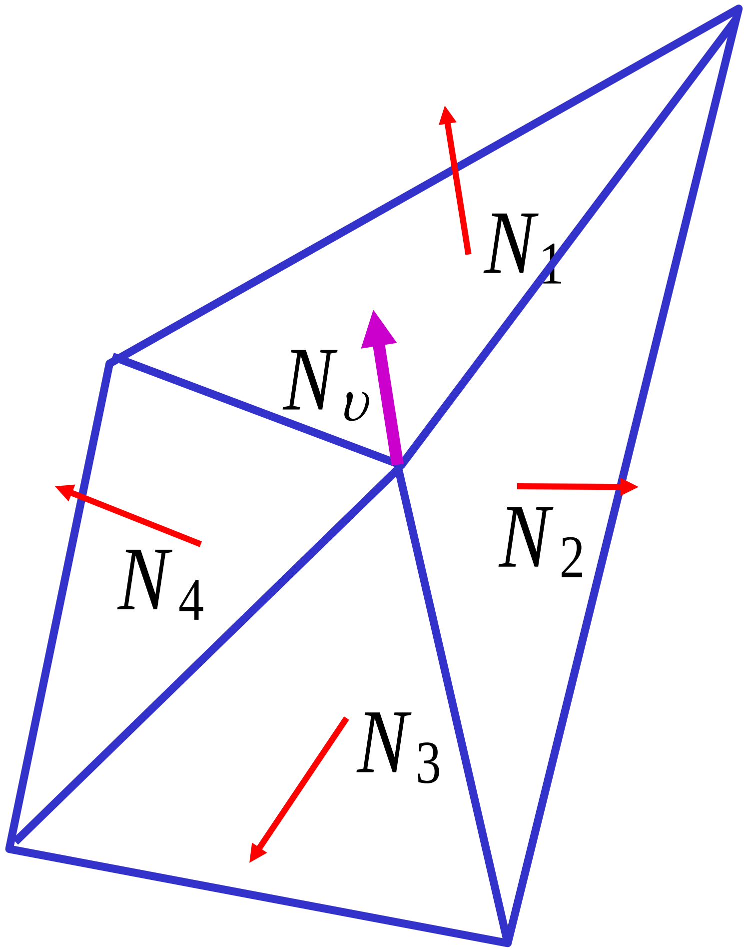


- No interpolation is necessary
- Faceting occurs when used on approximate surfaces

Flat (Cosine) Shading

- Compute constant shading function, over each polygon, based on simple cosine term
- Same normal and light vector across whole polygon
- Constant shading for polygon

$$\sim N \cdot L$$



Flat (Cosine) Shading

$$I = I_p k_d \cos(\theta)$$

$$= I_p k_d N \cdot L \quad , \quad \text{for unit } N, L$$

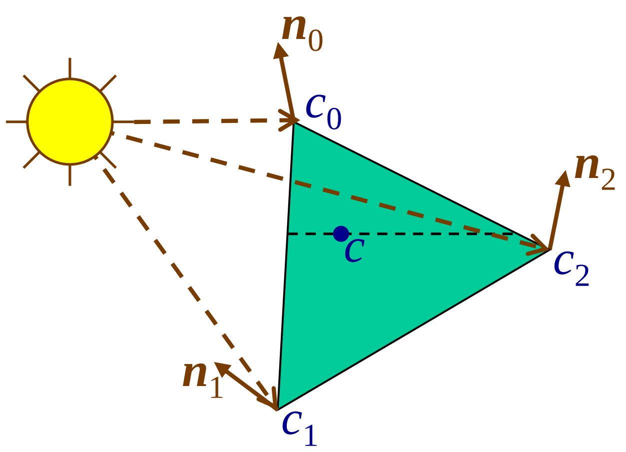
Where,

$I_p =$ intensity of point light source

$k_d =$ diffuse reflection coefficient

Gouraud Shading

- Compute the normal, $\boldsymbol{n}_i$, and color at each vertex:
  - If $\boldsymbol{n}_i$ not already provided → average of the normal of the faces that share the vertex
  - Compute color at each vertex using illumination model
- Interpolate colors across the projected polygon during scan conversion
- Assumes the polygons approximate the surface
- Problems?

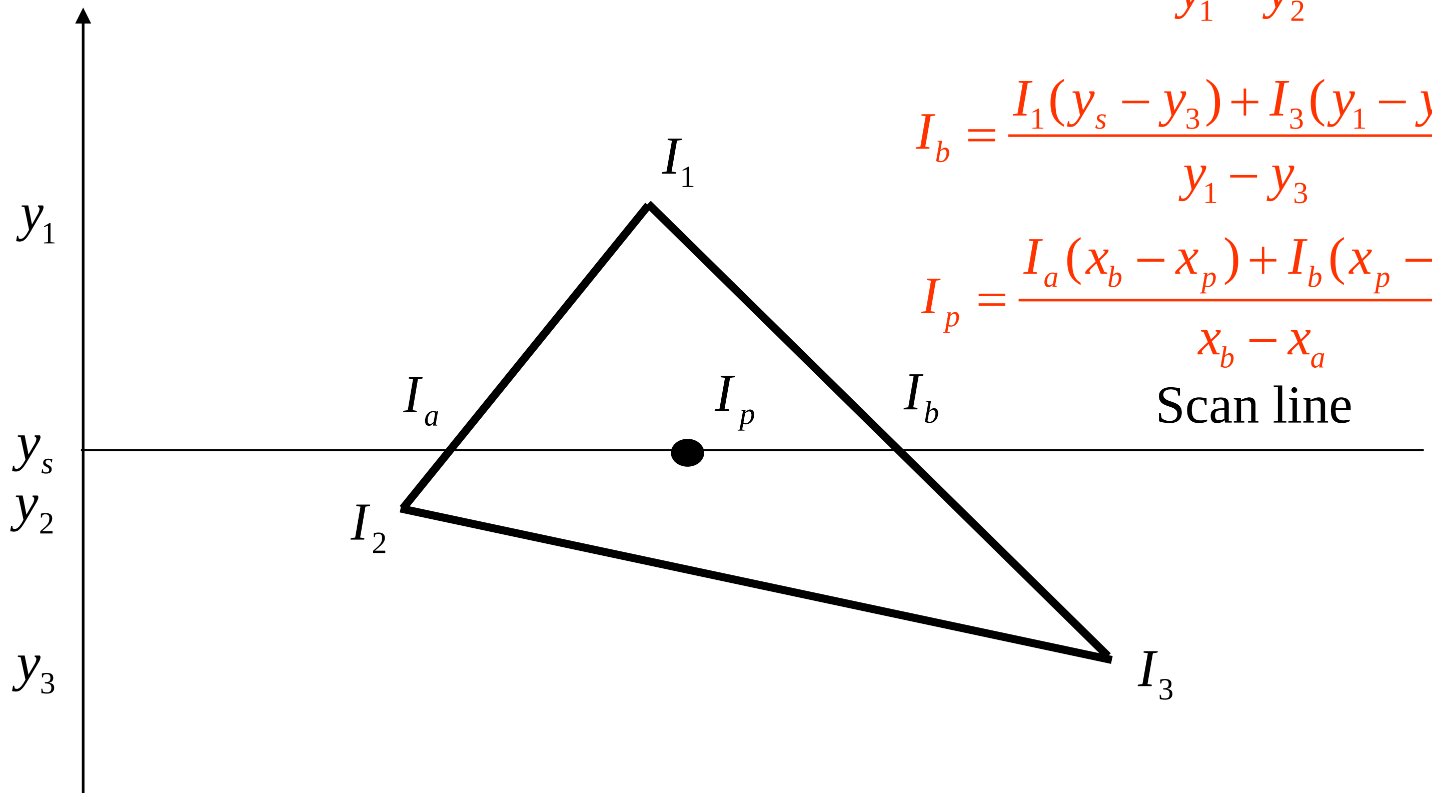


Gouraud Shading – Details

$$I_a = \frac{I_1(y_s - y_2) + I_2(y_1 - y_s)}{y_1 - y_2}$$

$$I_b = \frac{I_1(y_s - y_3) + I_3(y_1 - y_s)}{y_1 - y_3}$$

$$I_p = \frac{I_a(x_b - x_p) + I_b(x_p - x_a)}{x_b - x_a}$$



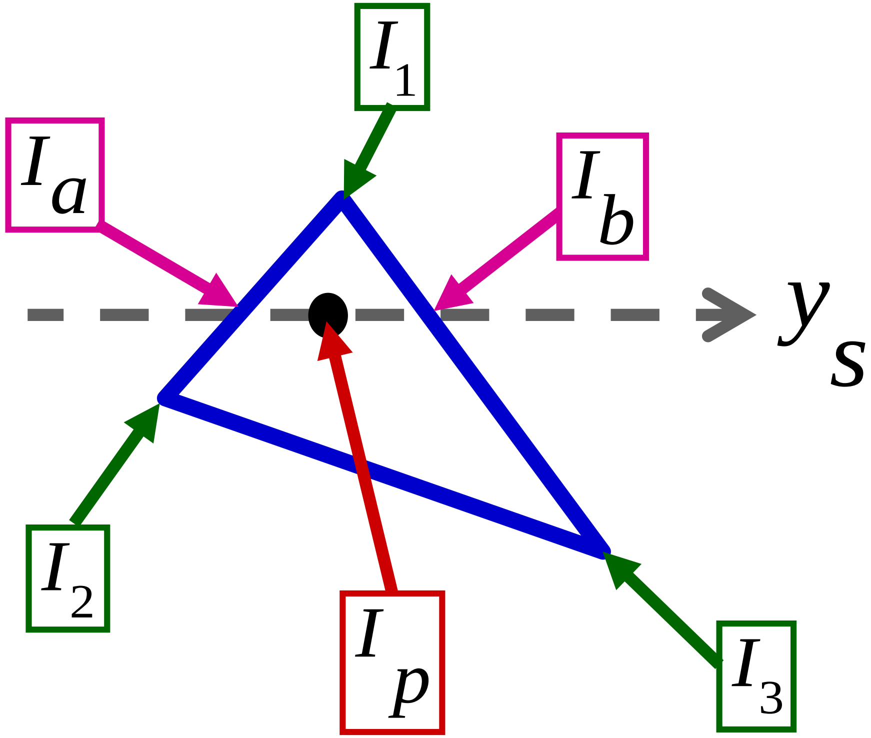
Actual implementation efficient: difference equations while scan converting

Intensity Interpolation (Gouraud Shading)

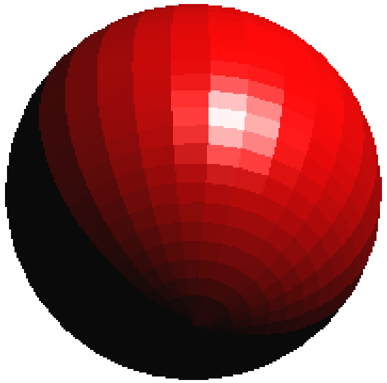
$$\boxed{I_a = I_1 \frac{y_s - y_2}{y_1 - y_2} + I_2 \frac{y_1 - y_s}{y_1 - y_2}}$$

$$\boxed{I_b = I_1 \frac{y_s - y_3}{y_1 - y_3} + I_3 \frac{y_1 - y_s}{y_1 - y_3}}$$

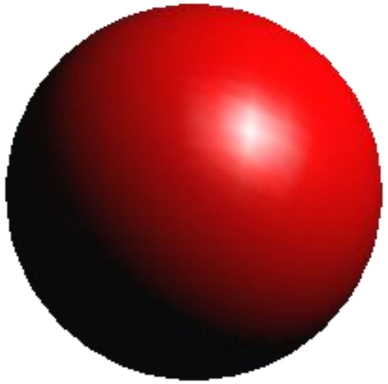
$$\boxed{I_p = I_a \frac{x_b - x_p}{x_b - x_a} + I_b \frac{x_p - x_a}{x_b - x_a}}$$



Flat vs. Gouraud Shading



glShadeModel(GL_FLAT)



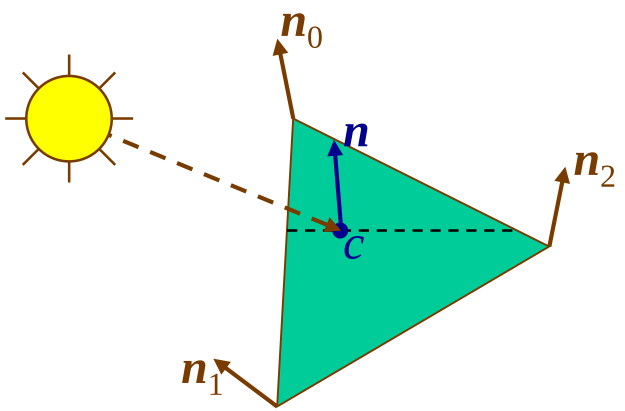
glShadeModel(GL_SMOOTH)

Flat - Determine that each face has a single normal, and color the entire face a single value, based on that normal.

Gouraud – Determine the color at each vertex, using the normal at that vertex, and interpolate linearly for the pixels between the vertex locations.

Phong Shading

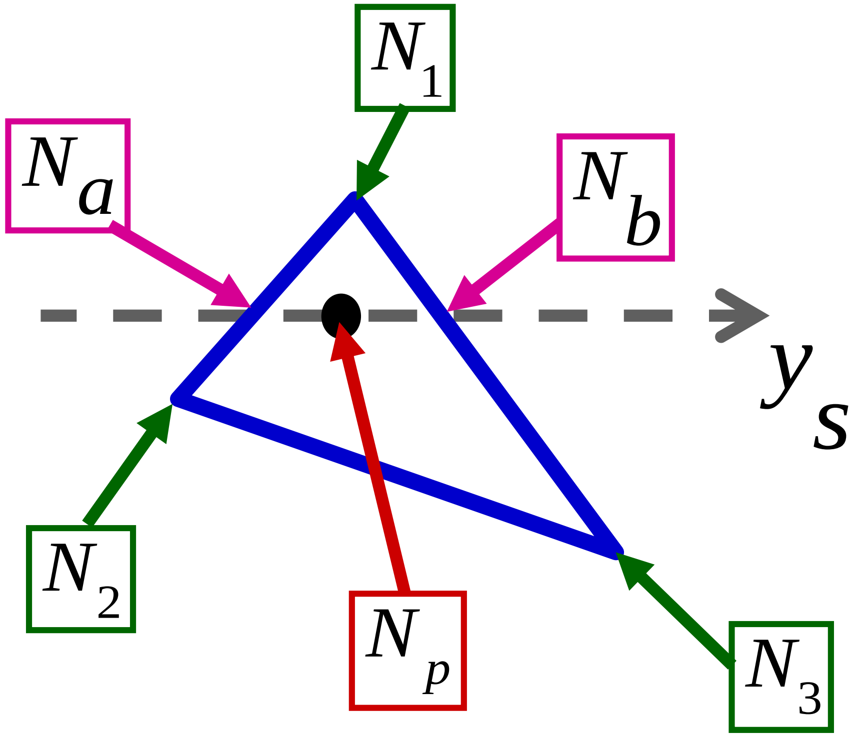
- Compute normal, $\boldsymbol{n}_i$, at each vertex (if not already given)
- Interpolate normals during scan conversion
- Compute color with the interpolated normals
  - Expensive: compute illumination for every visible point on a surface
  - Captures highlights in the middle of a polygon
  - Looks smoother across edges



Normal Interpolation (Phong Shading)

$$N_a = N_1 \frac{y_s - y_2}{y_1 - y_2} + N_2 \frac{y_1 - y_s}{y_1 - y_2}$$

$$N_b = N_1 \frac{y_s - y_3}{y_1 - y_3} + N_3 \frac{y_1 - y_s}{y_1 - y_3}$$



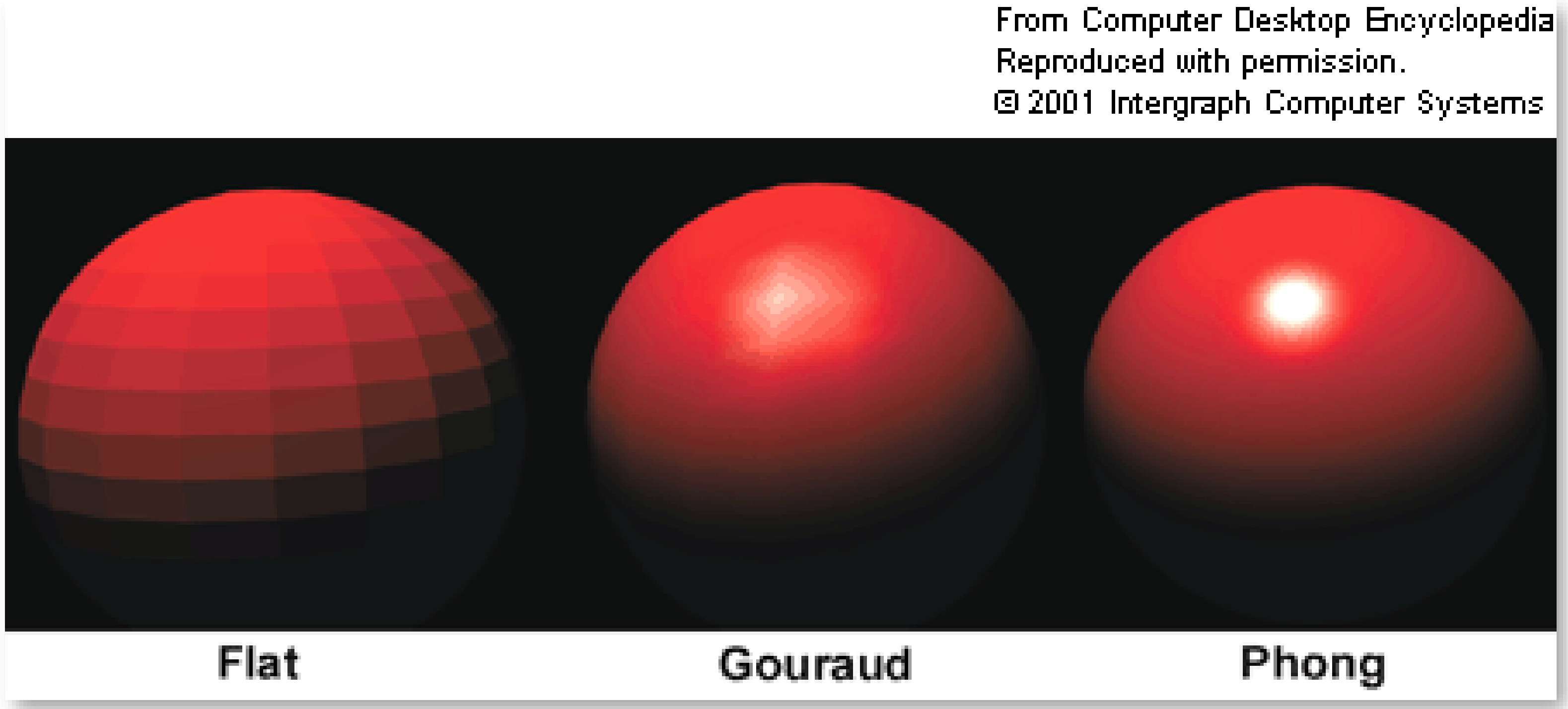
Normal Interpolation (Phong Shading)

$$\tilde{N}_p = \frac{N_a}{\|N_a\|}\left[\frac{x_b - x_p}{x_b - x_a}\right] + \frac{N_b}{\|N_b\|}\left[\frac{x_p - x_a}{x_b - x_a}\right]$$

$$N_p = \frac{\tilde{N}_p}{\left\|\tilde{N}_p\right\|}$$

Normalizing makes this a unit vector

Shading Comparison



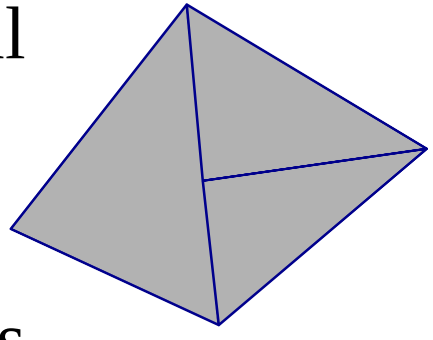
Problems with Interpolated Shading

- Silhouettes are still polygonal
- Interpolation in screen, not object space: perspective distortion
- Not rotation or orientation-independent
- How to compute vertex normals for sharply curving surfaces?

- But at end of day, polygons is mostly preferred to explicitly representing curved objects like spline patches for rendering

Problems with Interpolated Shading

- Silhouettes
- Perspective distortion causes problems: Imagine a polygon with 1 vertex at a very different depth than others
  - Interpolation considers equal steps in *y*, but foreshortening produces unequal steps in depth
  - Problem reduced by using many small polygons
- Mach banding
- Orientation dependence for non-triangles
- Shared vertices on an edge



Comparison

- If the polygon mesh approximates surfaces with a high curvatures, Phong shading may look smooth while Gouraud shading may show edges
- Phong shading requires *much more work* than Gouraud shading
  - Until recently not available in real time systems
  - *Now can be done in fragment shaders*
- Both need data structures to represent meshes so we can obtain vertex normals

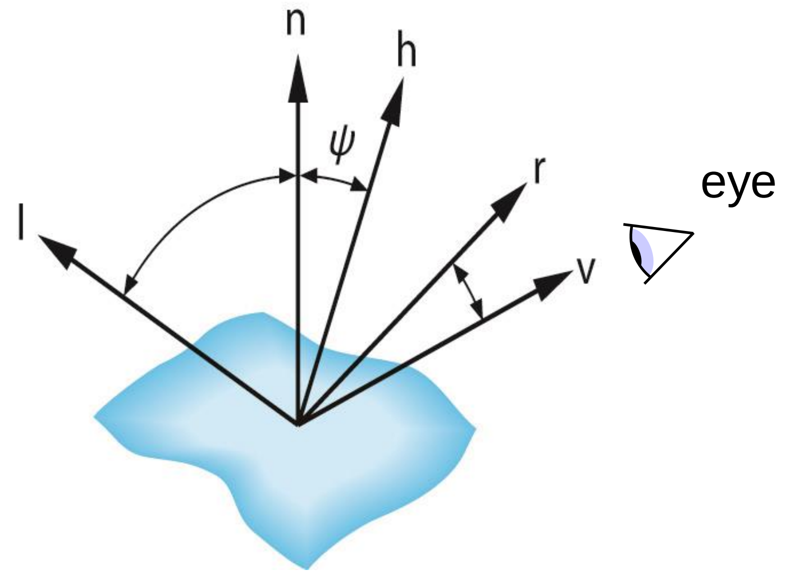
Per Vertex and Per Fragment Shaders

Vertex Lighting Shaders I

```
// vertex shader

attribute vec4 vPosition;
attribute vec4 vNormal;
varying vec4 fColor;
uniform vec4 ambientProduct, diffuseProduct, specularProduct;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;
uniform float shininess;

void main()
{
```

Vertex Lighting Shaders II

```
vec3 pos = (modelViewMatrix * vPosition).xyz;
vec3 light = lightPosition.xyz;
vec3 L = normalize( light - pos );   // l
vec3 E = normalize( -pos );          // v
vec3 H = normalize( L + E );         //  h

// Transform vertex normal into eye coordinates

     vec3 N = normalize( (modelViewMatrix*vNormal).xyz);

// Compute terms in the illumination equation
```

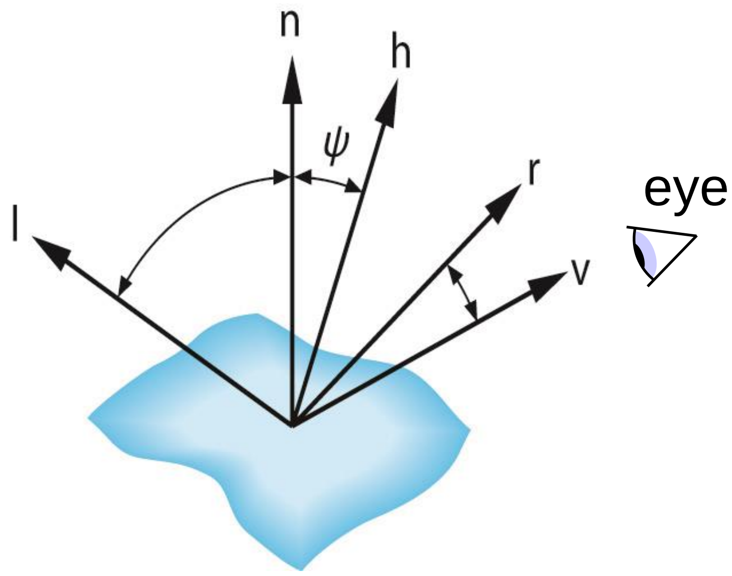


h = (l + v)/ | l + v |

Vertex Lighting Shaders III

```
// Compute terms in the illumination equation
    vec4 ambient = AmbientProduct;

    float Kd = max( dot(L, N), 0.0 );
    vec4  diffuse = Kd*DiffuseProduct;
    float Ks = pow( max(dot(N, H), 0.0), Shininess );
    vec4  specular = Ks * SpecularProduct;
    if( dot(L, N) < 0.0 )  specular = vec4(0.0, 0.0, 0.0, 1.0);
    gl_Position = Projection * ModelView * vPosition;

    color = ambient + diffuse + specular;
    color.a = 1.0;
}
```

$$\boxed{I = k_d I_d \, \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{n} \cdot \mathbf{h})^{\beta} + k_a I_a}$$



$$\mathbf{h} = (\mathbf{l} + \mathbf{v}) / |\mathbf{l} + \mathbf{v}|$$

Vertex Lighting Shaders IV

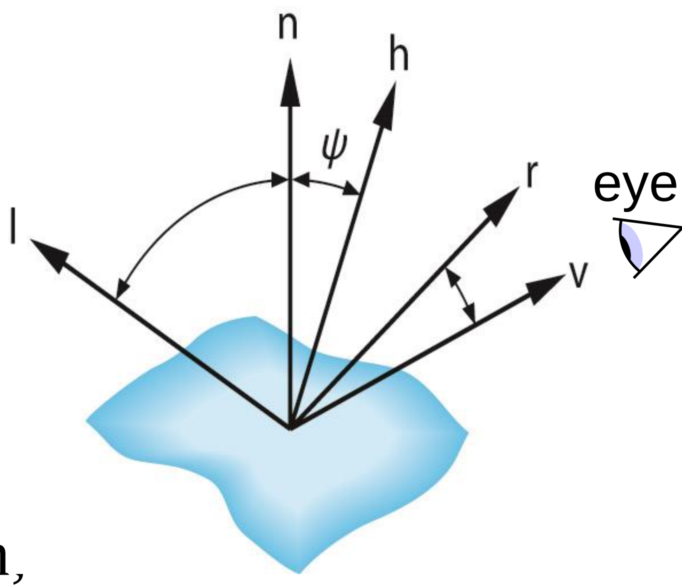
```
// fragment shader

precision mediump float;

varying vec4 fColor;

voidmain()
{
    gl_FragColor = fColor;
}
```

Fragment Lighting Shaders I

```
// vertex shader

attribute vec4 vPosition;
attribute vec4 vNormal;
varying vec3 N, L, E;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;
```

Fragment Lighting Shaders II

```
void main()
{
   vec3 pos = (modelViewMatrix * vPosition).xyz;
   vec3 light = lightPosition.xyz;
   L = normalize( light - pos );
   E =  -pos;
   N = normalize( (modelViewMatrix*vNormal).xyz);
   gl_Position = projectionMatrix * modelViewMatrix * vPosition,
  };
```



$$\mathbf{h} = (\mathbf{l} + \mathbf{v}) / |\mathbf{l} + \mathbf{v}|$$

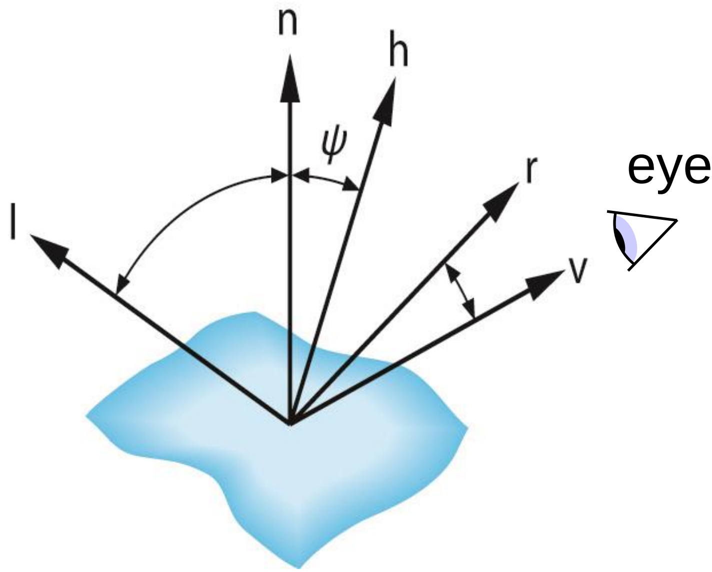
Fragment Lighting Shaders III

```
// fragment shader

precision mediump float;

uniform vec4 ambientProduct;
uniform vec4 diffuseProduct;
uniform vec4 specularProduct;
uniform float shininess;
varying vec3 N, L, E;

void main()
{
```

Fragment Lighting Shaders IV

```
vec4 fColor;
vec3 H = normalize( L + E );
vec4 ambient = ambientProduct;
float Kd = max( dot(L, N), 0.0 );
vec4  diffuse = Kd*diffuseProduct;
float Ks = pow( max(dot(N, H), 0.0), shininess );
vec4  specular = Ks * specularProduct;
    if( dot(L, N) < 0.0 ) specular = vec4(0.0, 0.0, 0.0, 1.0);
fColor = ambient + diffuse +specular;
fColor.a = 1.0;
gl_FragColor = fColor;
}
```

$$\boxed{I = k_d I_d \; \mathbf{l} \cdot \mathbf{n} \; + k_s I_s (\mathbf{n} \cdot \mathbf{h})^{\beta} + k_a I_a}$$



$$\mathbf{h} = (\mathbf{l} + \mathbf{v}) / |\mathbf{l} + \mathbf{v}|$$

Teapot Examples

Using per-vertex lighting

Area near highlight

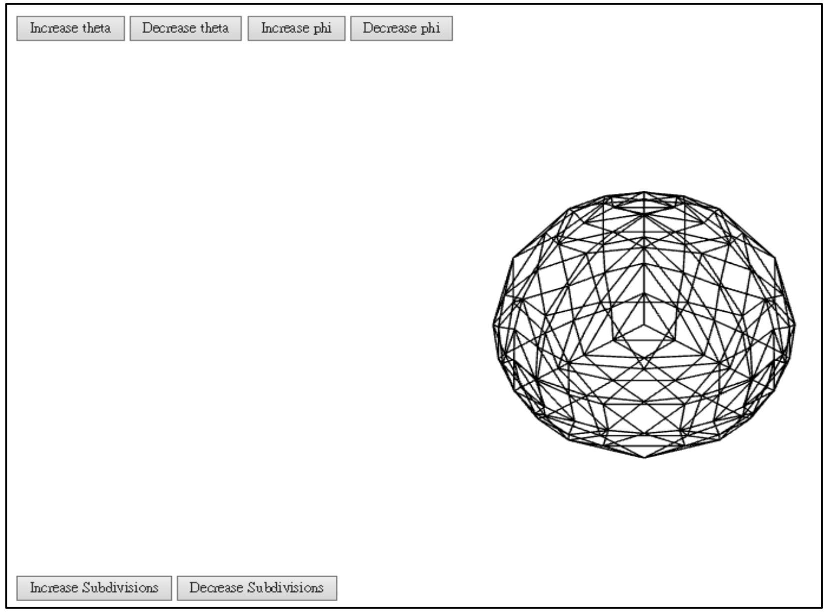
Using per-fragment lighting

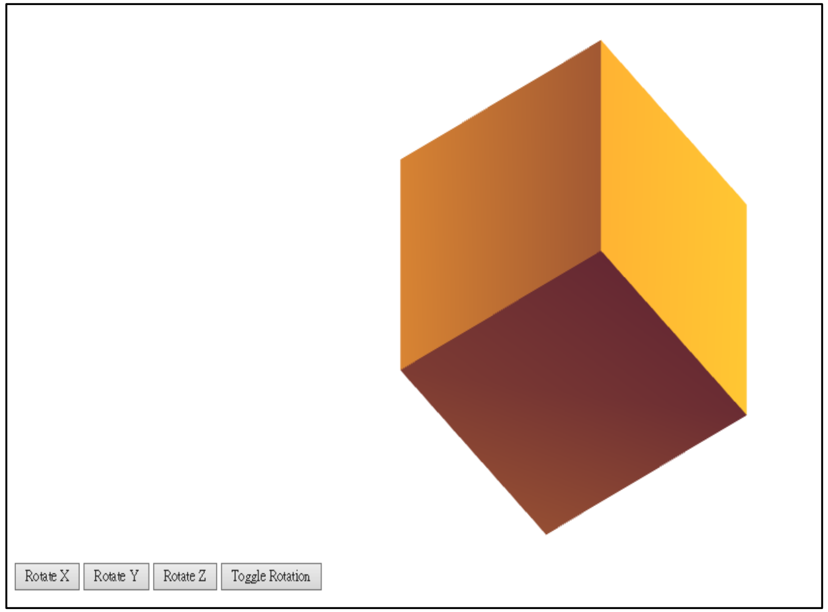
Area near highlight

Sample Programs

wireSphere

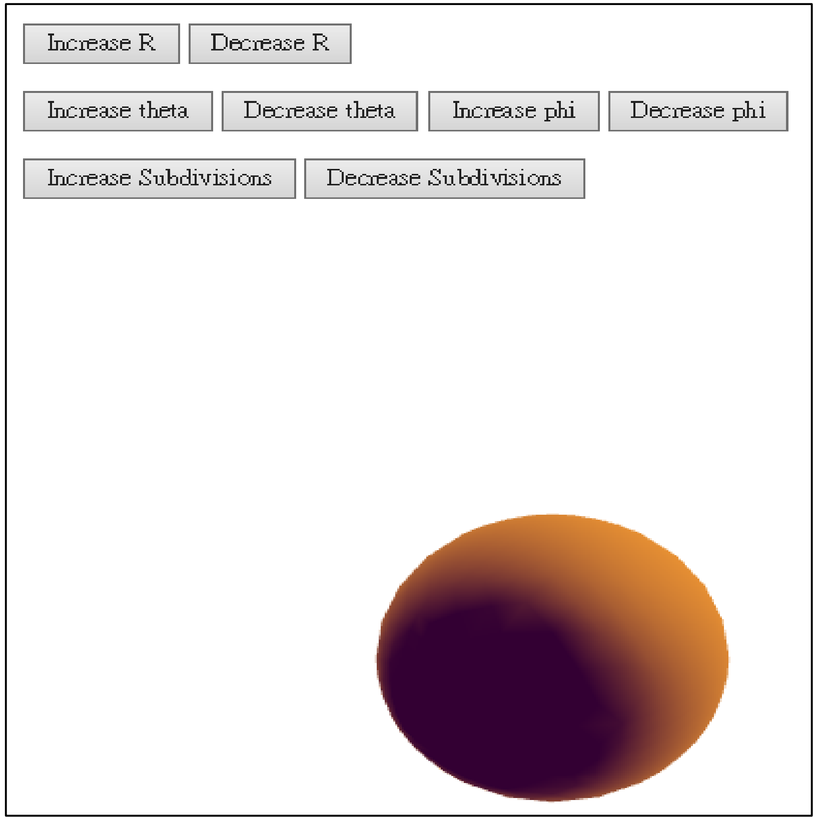
shadedCube: rotating cube with modified Phong shading



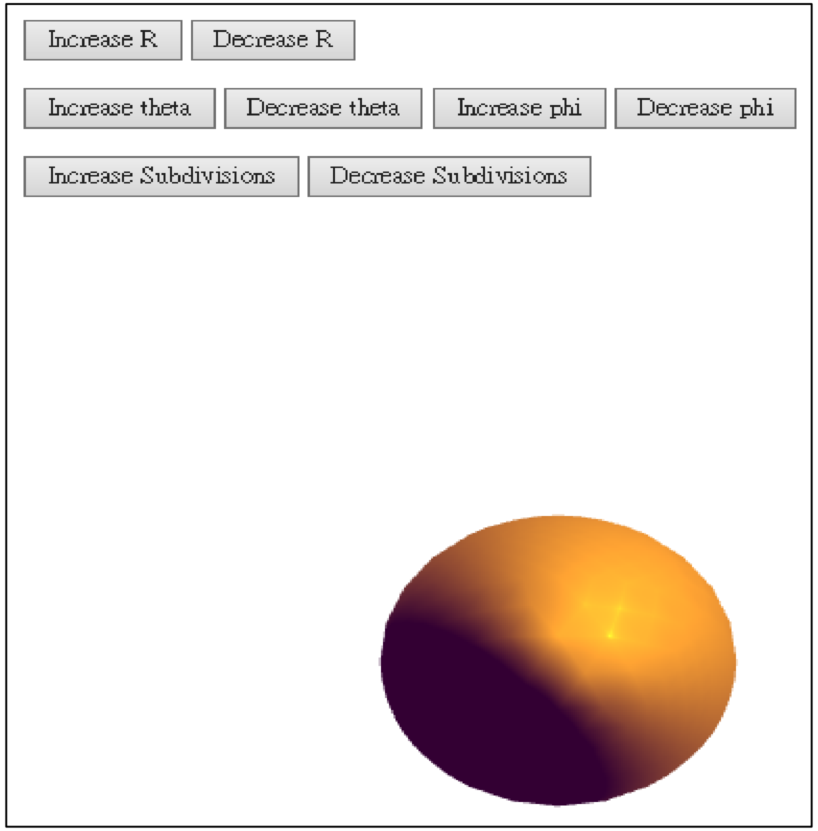


Sample Programs

Shadedsphere1 (per vertex shading)

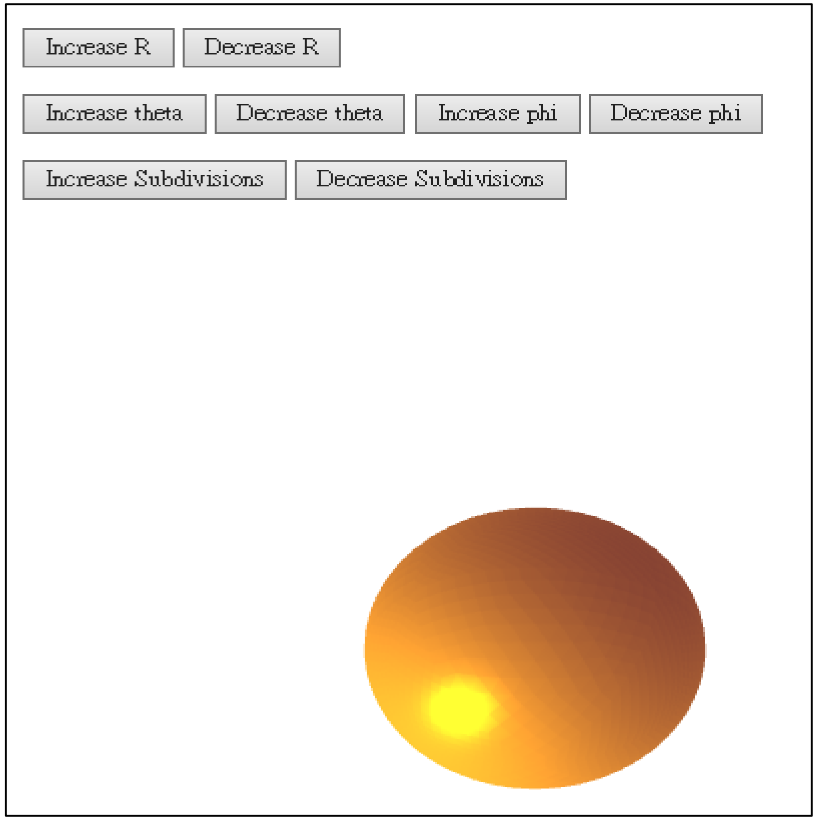


Shadedsphere2 (per fragment shading)

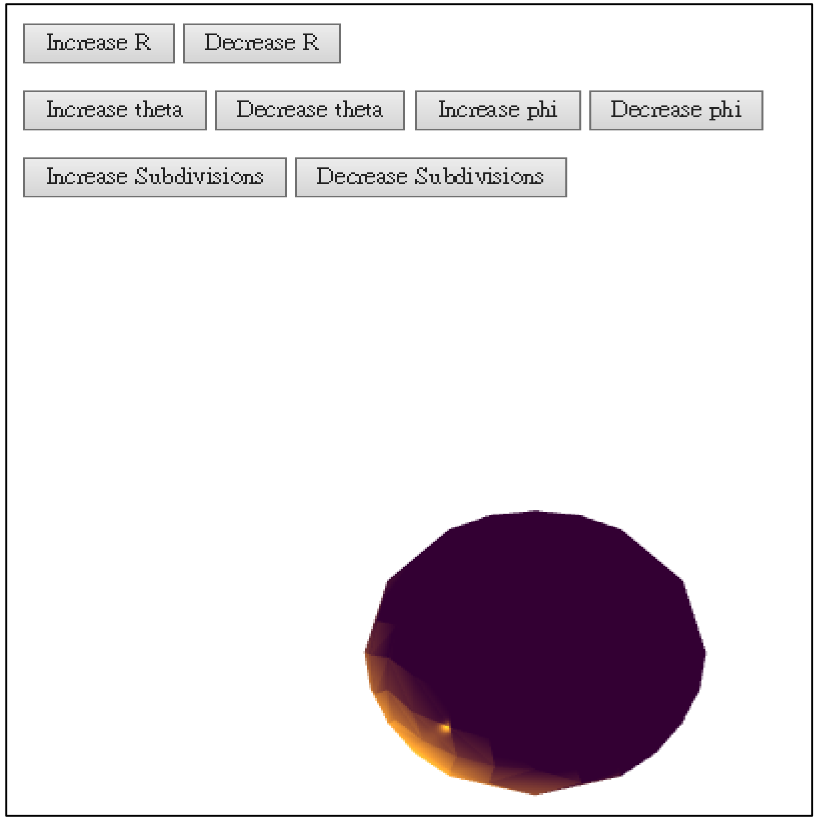


Sample Programs

Shadedsphere3 (per vertex shading)

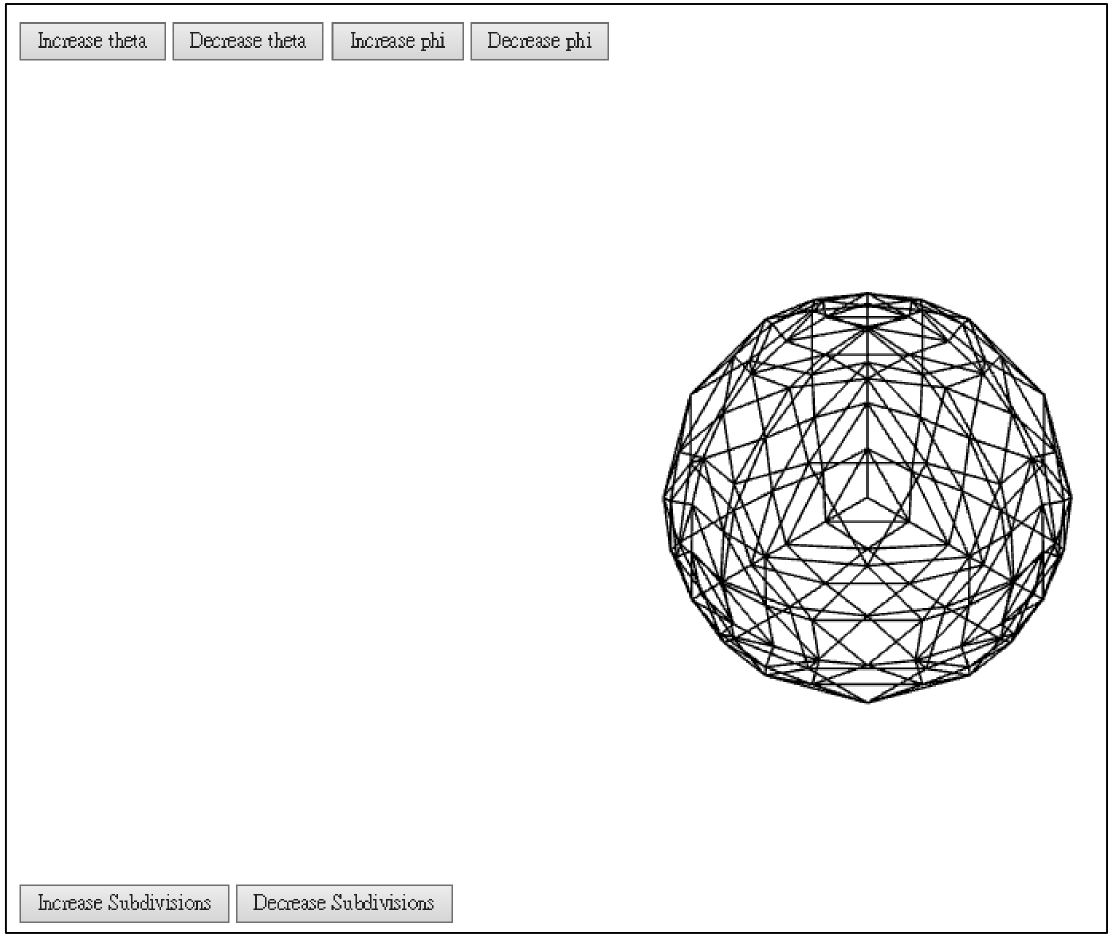


Shadedsphere4 (per fragment shading)



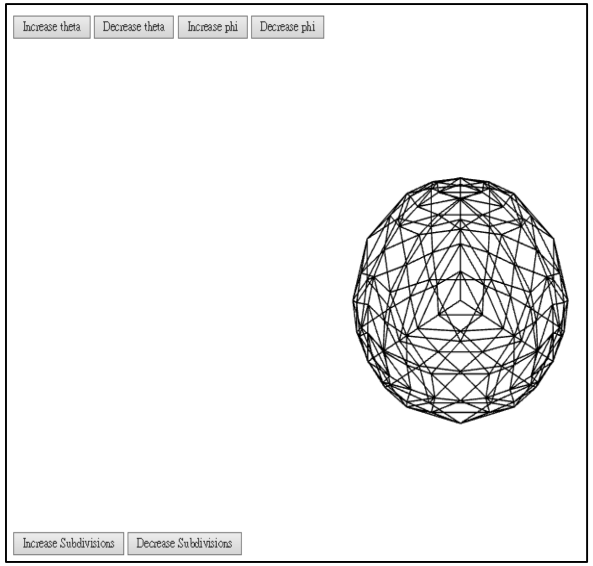
Sample Programs: wireSphere.html, wireSphere.js

Wire frame of recursively generated sphere



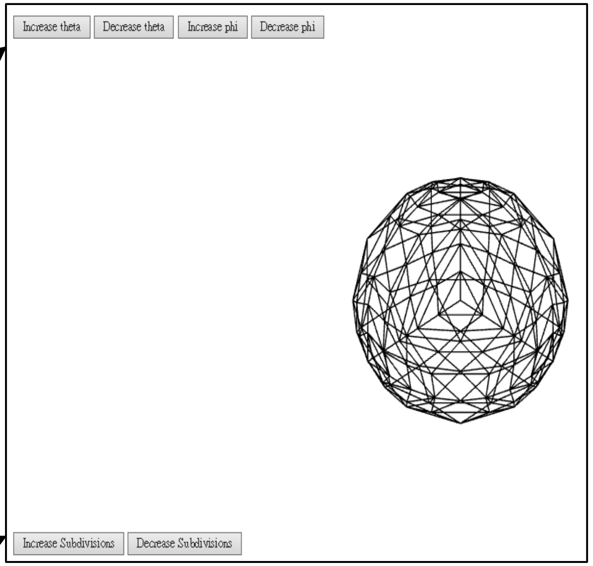
wireSphere.html (1/3)

```
<!DOCTYPE html>
<html>

<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

void
main()
{
    gl_Position = projectionMatrix*modelViewMatrix*vPosition;
}
</script>
```



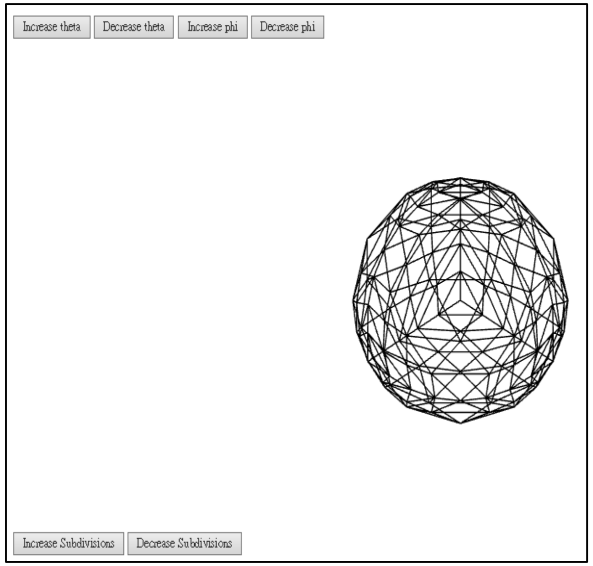
wireSphere.html (2/3)

```
<script id="fragment-shader" type="x-shader/x-fragment">

precision mediump float;

void main()
{  gl_FragColor = vec4(0.0, 0.0, 0.0, 1.0); }
</script>

<p> </p>
<button id = "Button0">Increase theta</button>
<button id = "Button1">Decrease theta</button>
<button id = "Button2">Increase phi</button>
<button id = "Button3">Decrease phi</button>
<p> </p>
<button id = "Button4">Increase Subdivisions</button>
<button id = "Button5">Decrease Subdivisions</button>
```



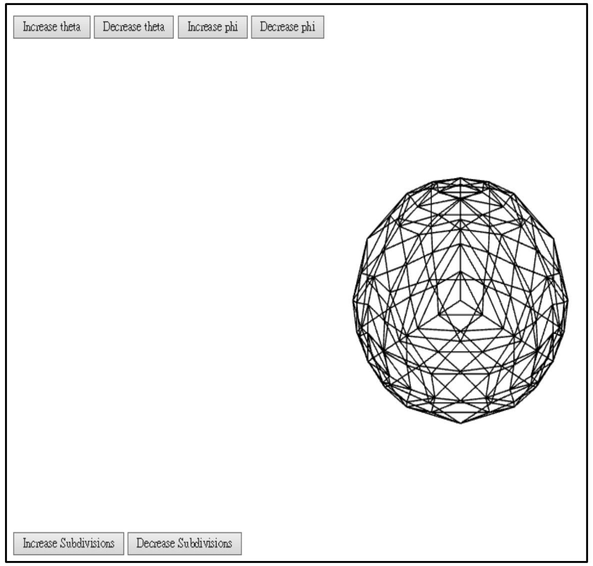
wireSphere.html (3/3)

```
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="wireSphere.js"></script>

<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



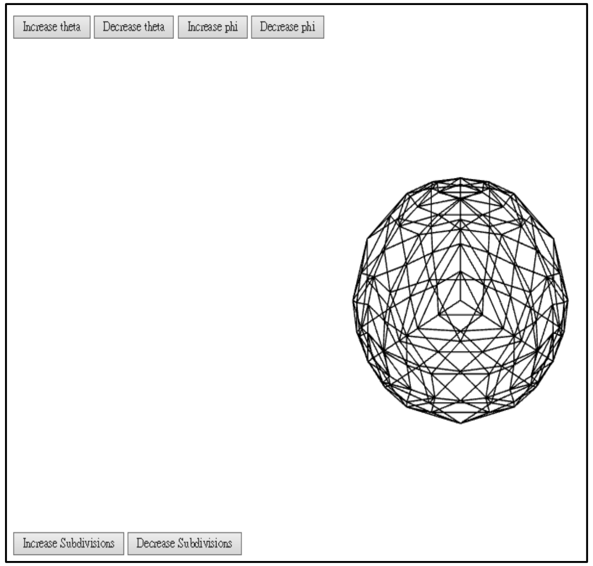
wireSphere.js (1/9)

```
var canvas;
var gl;

var numTimesToSubdivide = 3;

var index = 0;

var pointsArray = [];

var near = -10;
var far = 10;
var radius = 6.0;
var theta  = 0.0;
var phi    = 0.0;
var dr = 5.0 * Math.PI/180.0;
```



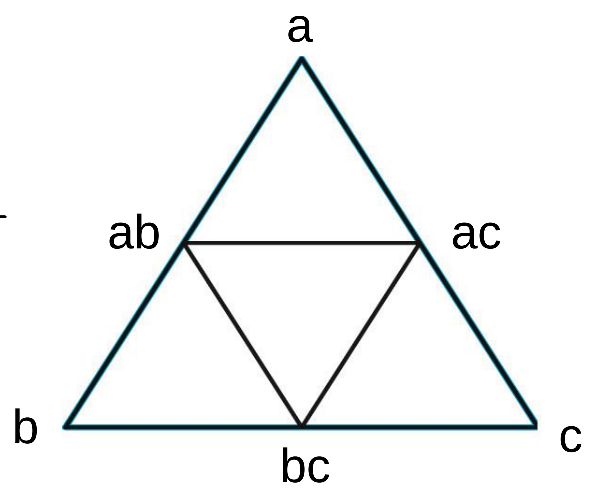
wireSphere.js (2/9)

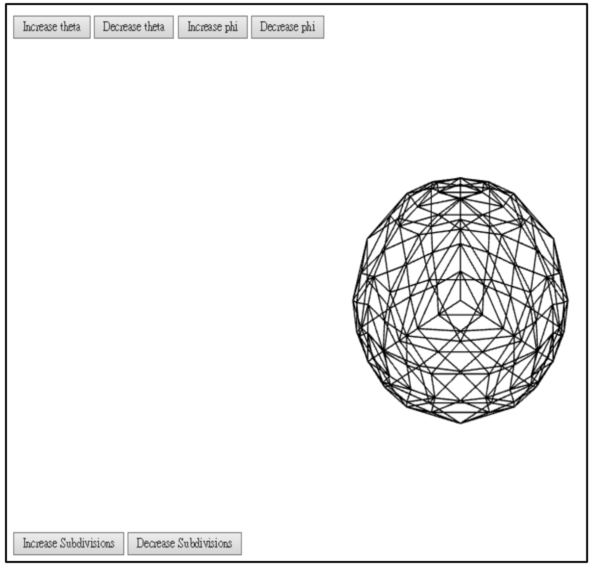
```
var left = -2.0;
var right = 2.0;
var ytop = 2.0;
var bottom = -2.0;

var modelViewMatrix, projectionMatrix;
var modelViewMatrixLoc, projectionMatrixLoc;
var eye;
const at  = vec3(0.0, 0.0, 0.0);
const up = vec3(0.0, 1.0, 0.0);

function triangle(a, b, c) {
    pointsArray.push(a);
    pointsArray.push(b);
    pointsArray.push(c);
    index += 3;
}
```



wireSphere.js (3/9)

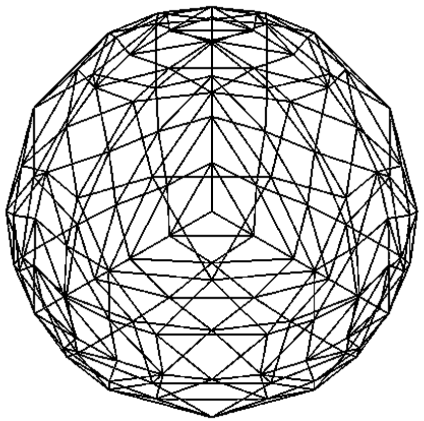
```
function divideTriangle(a, b, c, count) {
    if ( count > 0 ) {

        var ab = normalize(mix( a, b, 0.5), true);
        var ac = normalize(mix( a, c, 0.5), true);
        var bc = normalize(mix( b, c, 0.5), true);

        divideTriangle(   a, ab, ac, count - 1 );
        divideTriangle( ab,   b, bc, count - 1 );
        divideTriangle( bc,   c, ac, count - 1 );
        divideTriangle( ab, bc, ac, count - 1 );
    }
    else { // draw tetrahedron at end of recursion
        triangle( a, b, c );
    }
}
```

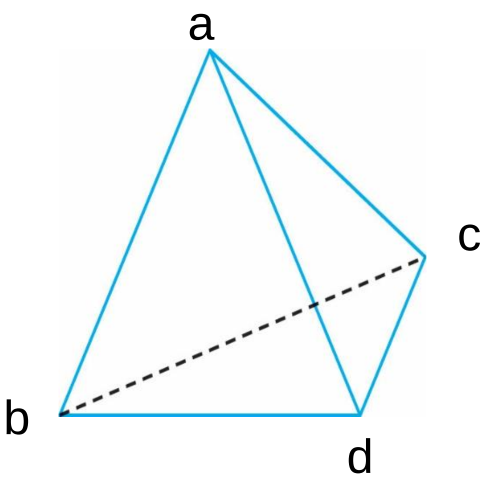


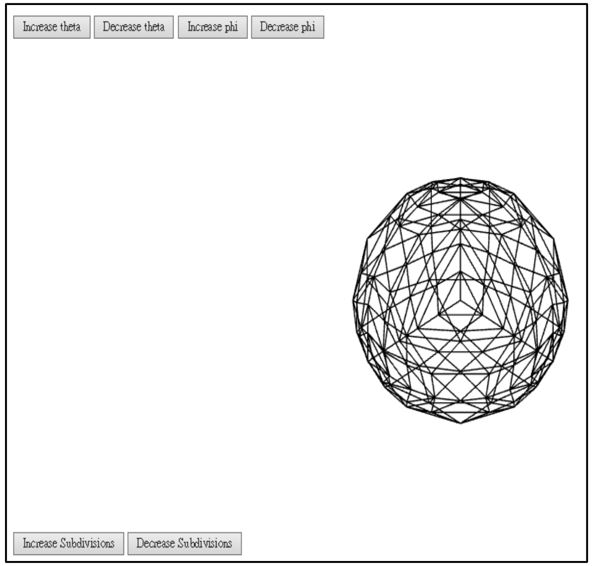


wireSphere.js (4/9)

```
function tetrahedron(a, b, c, d, n) {
    divideTriangle(a, b, c, n);
    divideTriangle(d, c, b, n);
    divideTriangle(a, d, b, n);
    divideTriangle(a, c, d, n);
}
```

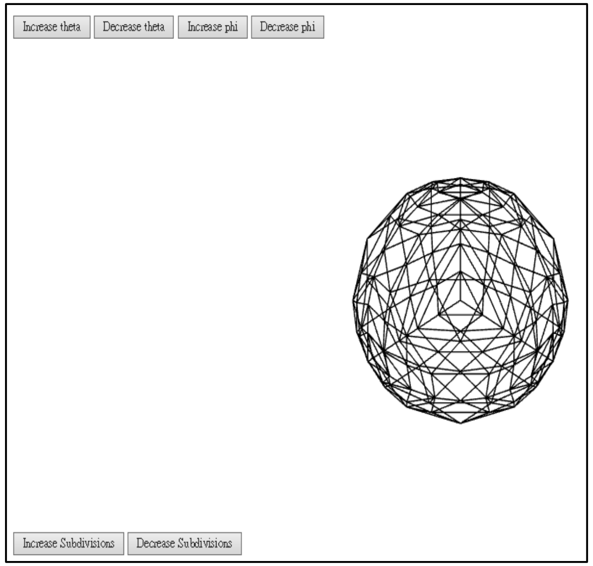






wireSphere.js (5/9)

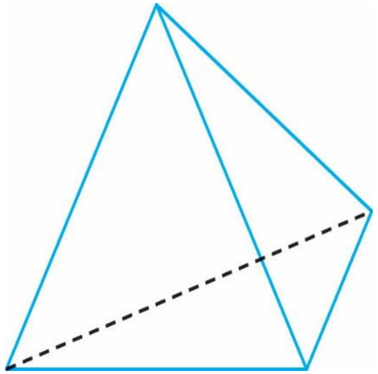
```
window.onload = function init() {
    canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );
```

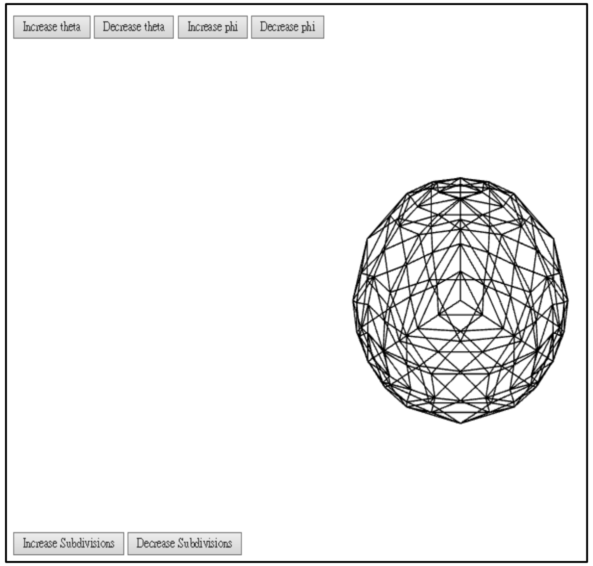


wireSphere.js (6/9)

```
//
//  Load shaders and initialize attribute buffers
//
var program = initShaders( gl, "vertex-shader", "fragment-shader" );
gl.useProgram( program );

var va = vec4( 0.0,            0.0,          -1.0,          1);
var vb = vec4( 0.0,            0.942809, 0.333333, 1);
var vc = vec4(-0.816497, -0.471405, 0.333333, 1);
var vd = vec4( 0.816497, -0.471405, 0.333333, 1);

tetrahedron(va, vb, vc, vd, numTimesToSubdivide);
```

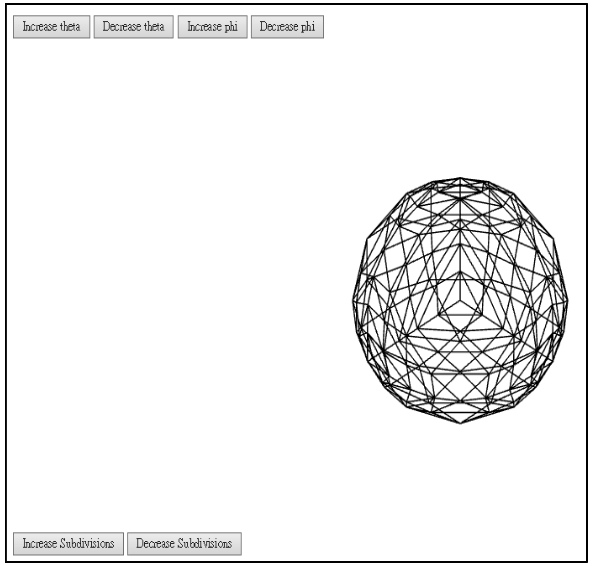
$$\left\{\begin{array}{l}(0.0,\ 0.0,\ -1.0)\\ (0.0, 2\sqrt{2}/3,\ 1/3)\\ (-\sqrt{6}/3, -\sqrt{2}/3,\ 1/3)\\ (\sqrt{6}/3, -\sqrt{2}/3,\ 1/3)\end{array}\right.$$





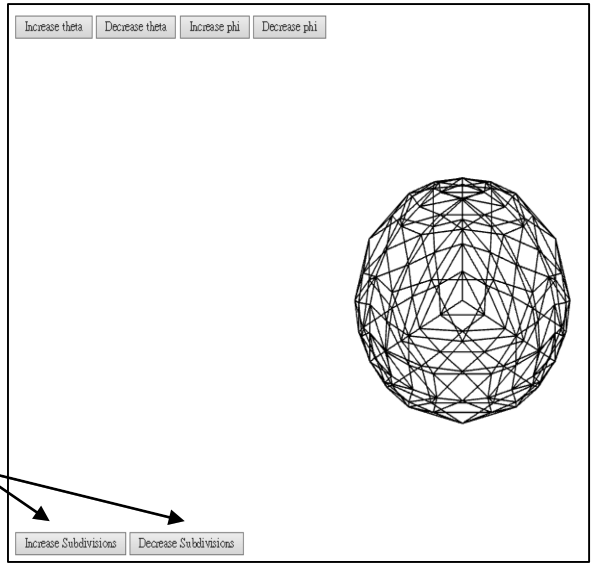
wireSphere.js (7/9)

```
vBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer);
gl.bufferData( gl.ARRAY_BUFFER, flatten(pointsArray), gl.STATIC_DRAW);

var vPosition = gl.getAttribLocation( program, "vPosition");
gl.vertexAttribPointer( vPosition, 4, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray( vPosition);

modelViewMatrixLoc = gl.getUniformLocation( program, "modelViewMatrix" );
projectionMatrixLoc   = gl.getUniformLocation( program, "projectionMatrix" );

document.getElementById("Button0").onclick = function() {theta += dr;};
document.getElementById("Button1").onclick = function() {theta -= dr;};
document.getElementById("Button2").onclick = function() {phi += dr;};
document.getElementById("Button3").onclick = function() {phi -= dr;};
```



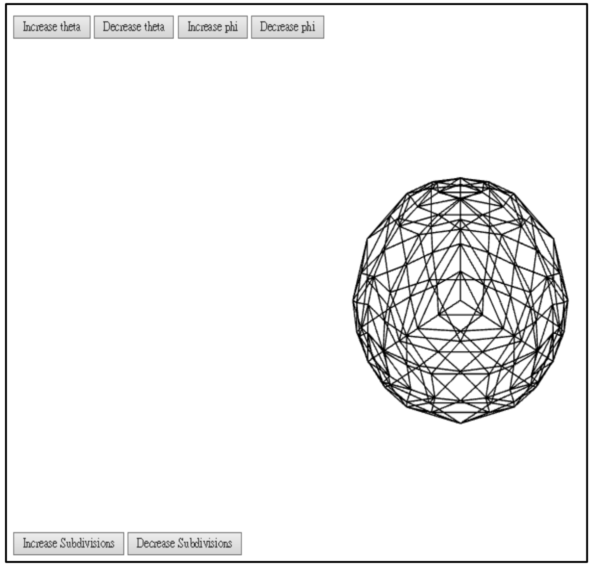
wireSphere.js (8/9)

```
document.getElementById("Button4").onclick = function(){
        numTimesToSubdivide++;
        index = 0;
        pointsArray = [];
        init();
    };
document.getElementById("Button5").onclick = function(){
        if(numTimesToSubdivide) numTimesToSubdivide--;
        index = 0;
        pointsArray = [];
        init();
    };
    render();
}   // end of window.onload
```



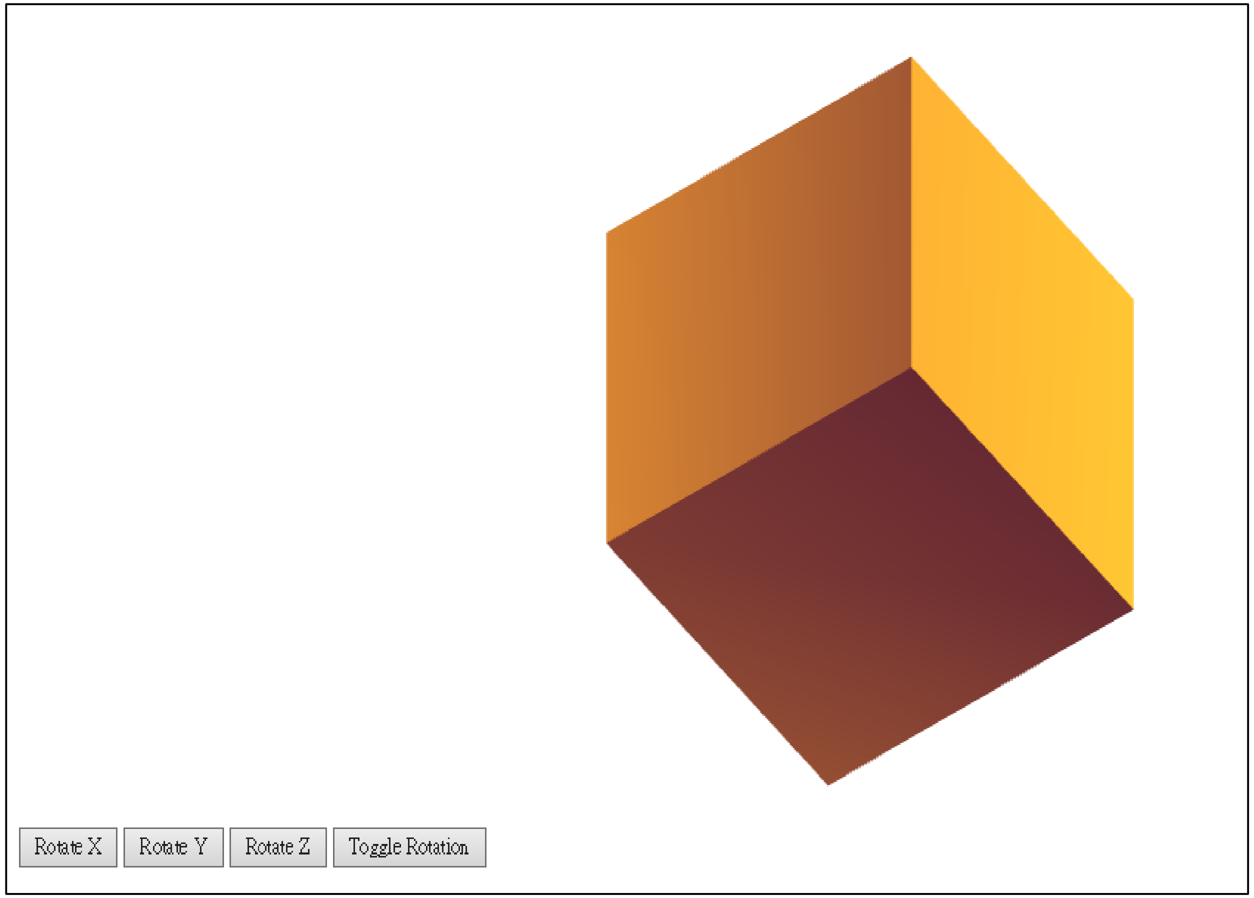
wireSphere.js (9/9)

```
function render() {

    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    eye = vec3(radius*Math.sin(theta)*Math.cos(phi),
                    radius*Math.sin(theta)*Math.sin(phi),
                    radius*Math.cos(theta));
    modelViewMatrix = lookAt(eye, at , up);
    projectionMatrix   = ortho(left, right, bottom, ytop, near, far);

    gl.uniformMatrix4fv( modelViewMatrixLoc, false, flatten(modelViewMatrix) );
    gl.uniformMatrix4fv( projectionMatrixLoc,   false, flatten(projectionMatrix) );

    for( var i=0; i<index; i+=3)
      gl.drawArrays( gl.LINE_LOOP, i, 3 );

    window.requestAnimFrame(render);
}   // end of render()
```



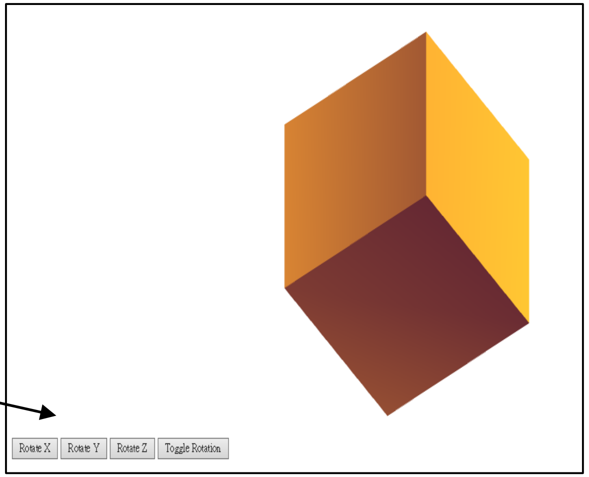
Sample Programs: shadedCube.html, shadedCube.js

Rotating cube with modified Phong shading



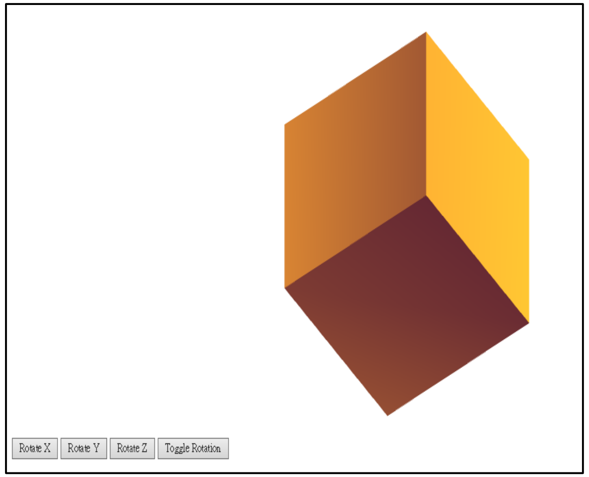
shadedCube.html (1/5)

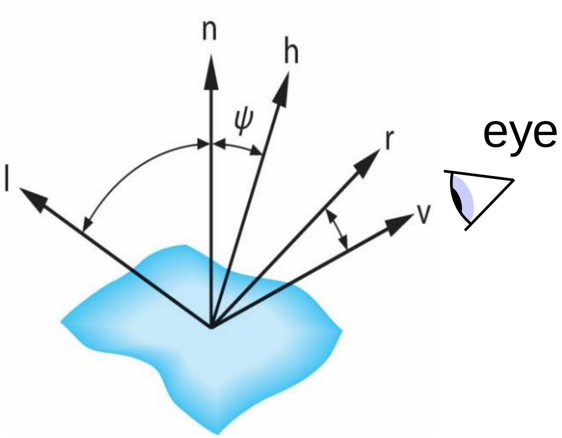
```
<!DOCTYPE html>
<html>

<button id = "ButtonX">Rotate X</button>
<button id = "ButtonY">Rotate Y</button>
<button id = "ButtonZ">Rotate Z</button>
<button id = "ButtonT">Toggle Rotation</button>
```



shadedCube.html (2/5)

```
<script id="vertex-shader" type="x-shader/x-vertex">
attribute  vec4 vPosition;
attribute  vec3 vNormal;
varying vec4 fColor;

uniform vec4 ambientProduct, diffuseProduct, specularProduct;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;
uniform float shininess;
void main()
{  vec3 pos = (modelViewMatrix * vPosition).xyz;
    vec3 light = lightPosition.xyz;
    vec3 L = normalize( light - pos );
    vec3 E = normalize( -pos );     //  viewer at the origin (0,0,0)
    vec3 H = normalize( L + E );
    vec4 NN = vec4(vNormal,0);
```



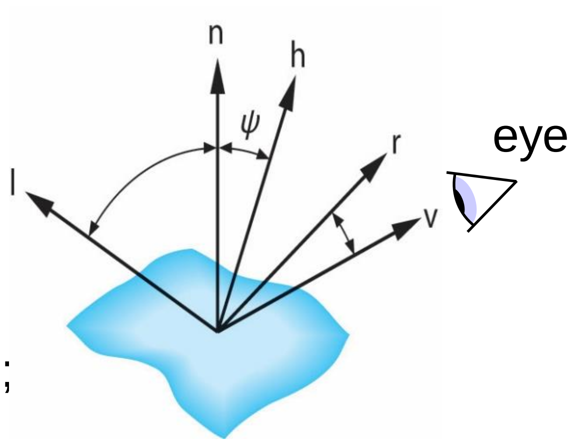


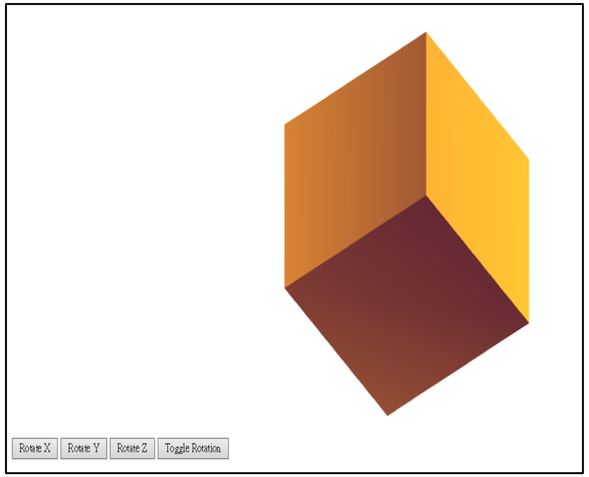
$$I = k_d I_d \ (\mathbf{L} \cdot \mathbf{N}) \ + k_s I_s (\mathbf{N} \cdot \mathbf{H})^{\alpha} + k_a I_a$$

where $\mathbf{H} = (\mathbf{L}+\mathbf{E})/2$ and α is shineness

shadedCube.html (3/5)

```
    // Transform vertex normal into eye coordinates

    vec3 N = normalize( (modelViewMatrix*NN).xyz);

    // Compute terms in the illumination equation
    vec4 ambient = ambientProduct;
    float Kd = max( dot(L, N), 0.0 );
    vec4  diffuse = Kd*diffuseProduct;

    float Ks = pow( max(dot(N, H), 0.0), shininess );
    vec4  specular = Ks * specularProduct;

    if( dot(L, N) < 0.0 ) { specular = vec4(0.0, 0.0, 0.0, 1.0);

    gl_Position = projectionMatrix * modelViewMatrix * vPosition;
    fColor = ambient + diffuse +specular;
    fColor.a = 1.0;
}
</script>
```



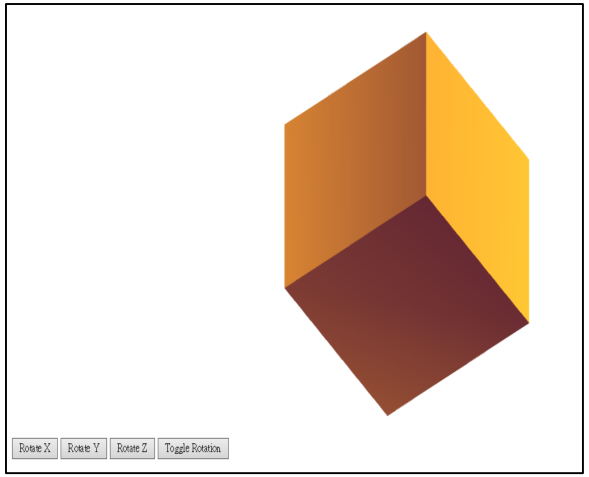


$I = k_d I_d \; (\mathbf{L} \cdot \mathbf{N}) \; + k_s I_s (\mathbf{N} \cdot \mathbf{H})^{\alpha} + k_a I_a$

where $\mathbf{H} = (\mathbf{L}+\mathbf{E})/2$ and α is shineness

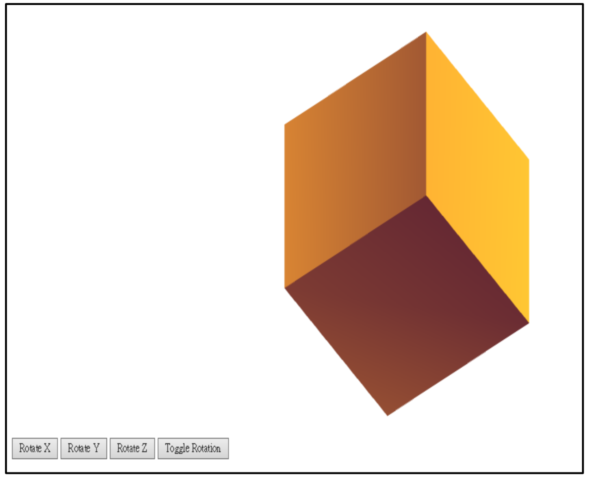
shadedCube.html (4/5)

```
<script id="fragment-shader" type="x-shader/x-fragment">

#ifdef GL_ES
precision highp float;
#endif

varying vec4 fColor;

void
main()
{
    gl_FragColor = fColor;
}
</script>
```



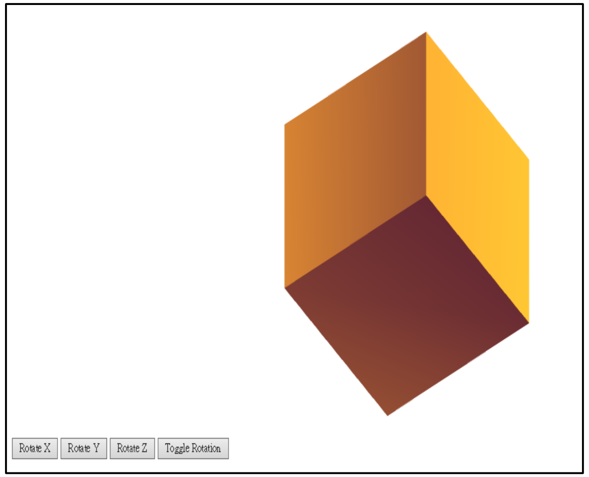
shadedCube.html (5/5)

```
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="shadedCube.js"></script>

<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



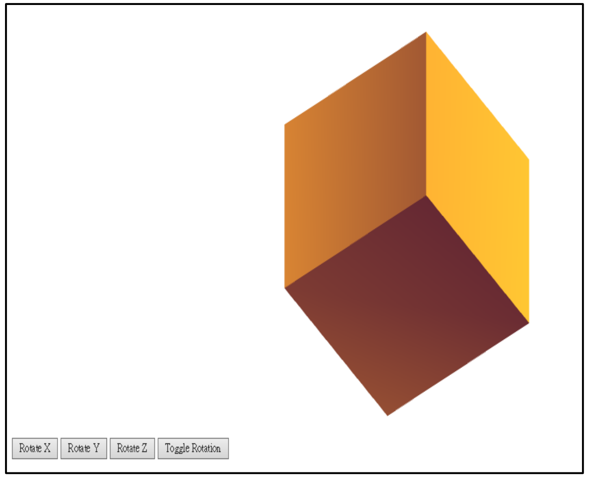
shadedCube.js (1/11)

```
var canvas;
var gl;

var numVertices  = 36;

var pointsArray = [];
var normalsArray = [];

var vertices = [
        vec4( -0.5, -0.5,  0.5, 1.0 ),
        vec4( -0.5,  0.5,  0.5, 1.0 ),
        vec4(  0.5,  0.5,  0.5, 1.0 ),
        vec4(  0.5, -0.5,  0.5, 1.0 ),
        vec4( -0.5, -0.5, -0.5, 1.0 ),
        vec4( -0.5,  0.5, -0.5, 1.0 ),
        vec4(  0.5,  0.5, -0.5, 1.0 ),
        vec4(  0.5, -0.5, -0.5, 1.0 )
    ];
```



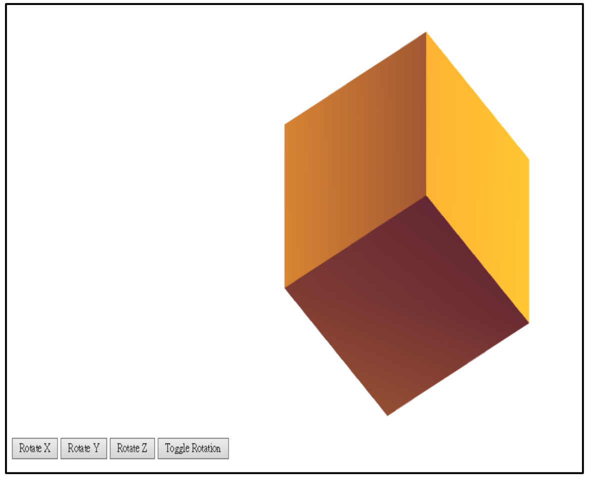
shadedCube.js (2/11)

```
var lightPosition  = vec4(1.0, 1.0, 1.0,  0.0 );
var lightAmbient  = vec4(0.2, 0.2, 0.2, 1.0 );
var lightDiffuse     = vec4(1.0, 1.0, 1.0, 1.0 );
var lightSpecular = vec4(1.0, 1.0, 1.0, 1.0 );

var materialAmbient   = vec4( 1.0, 0.0, 1.0, 1.0 );
var materialDiffuse     = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialSpecular  = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialShininess = 100.0;
```



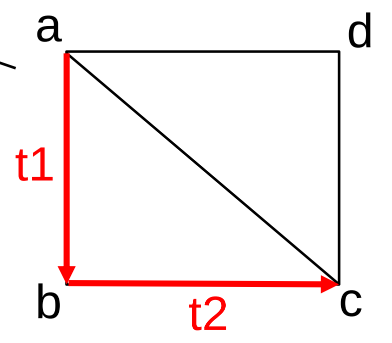
shadedCube.js (3/11)

```
var ctm;
var ambientColor, diffuseColor, specularColor;
var modelView, projection;
var viewerPos;
var program;

var xAxis = 0;
var yAxis = 1;
var zAxis = 2;
var axis = 0;
var theta =[0, 0, 0];

var thetaLoc;

var flag = true;
```

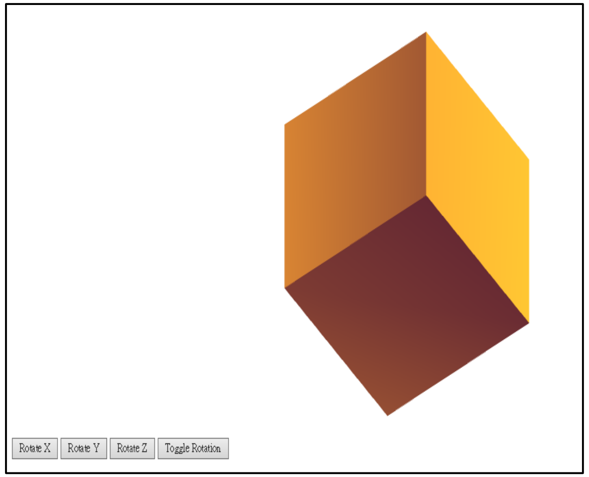


shadedCube.js (4/11)

```
function quad(a, b, c, d) {     // a, b, c, d: counterclockwise sequence
     var t1 = subtract(vertices[b], vertices[a]);
     var t2 = subtract(vertices[c], vertices[b]);
     var normal = cross(t1, t2);
     var normal = vec3(normal);
     normal = normalize(normal);
     pointsArray.push(vertices[a]);
     normalsArray.push(normal);
     pointsArray.push(vertices[b]);
     normalsArray.push(normal);
     pointsArray.push(vertices[c]);
     normalsArray.push(normal);
     pointsArray.push(vertices[a]);
     normalsArray.push(normal);
     pointsArray.push(vertices[c]);
     normalsArray.push(normal);
     pointsArray.push(vertices[d]);
     normalsArray.push(normal);
}
```

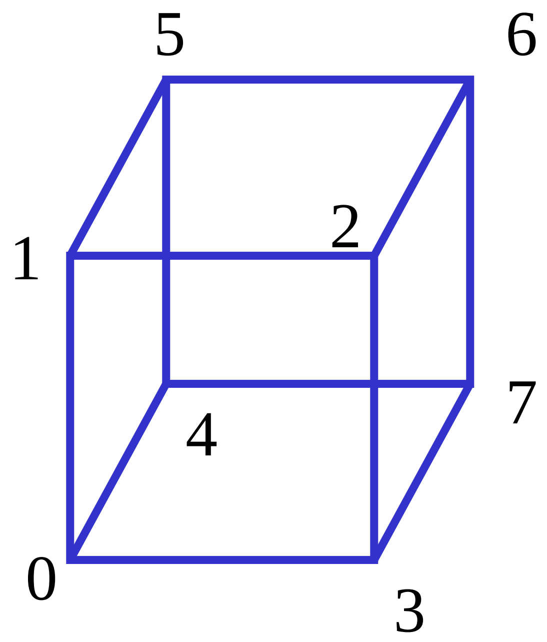


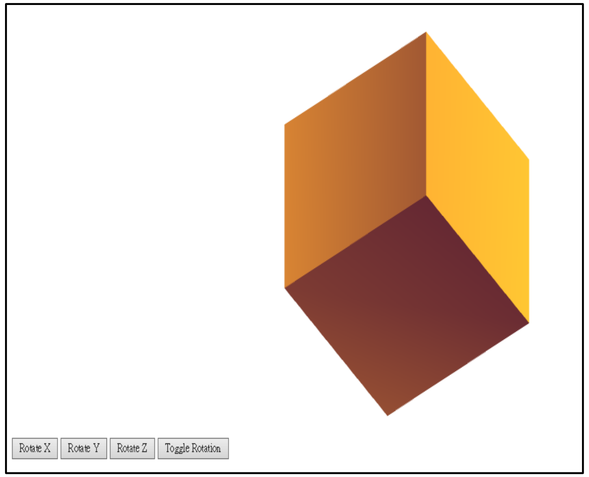
gl.TRIANGLES



shadedCube.js (5/11)

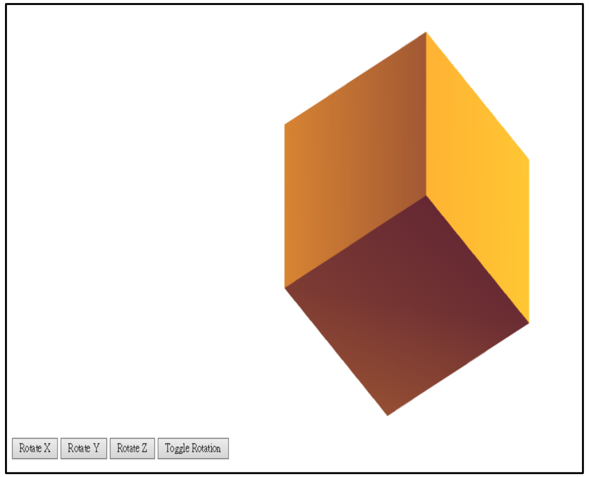
```
function colorCube()
{
   quad( 1, 0, 3, 2 );
   quad( 2, 3, 7, 6 );
   quad( 3, 0, 4, 7 );
   quad( 6, 5, 1, 2 );
   quad( 4, 5, 6, 7 );
   quad( 5, 4, 0, 1 );
}
```





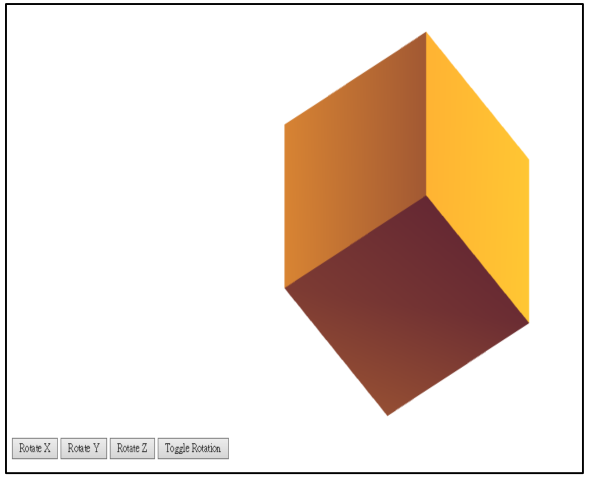
shadedCube.js (6/11)

```
window.onload = function init() {
    canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );

    gl.enable(gl.DEPTH_TEST);
```



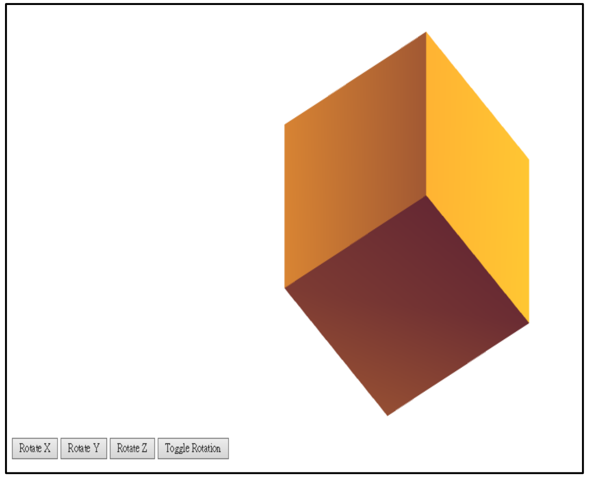
shadedCube.js (7/11)

```
//
//  Load shaders and initialize attribute buffers
//
program = initShaders( gl, "vertex-shader", "fragment-shader" );
gl.useProgram( program );

colorCube();

var nBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, nBuffer );
gl.bufferData( gl.ARRAY_BUFFER, flatten(normalsArray), gl.STATIC_DRAW );

var vNormal = gl.getAttribLocation( program, "vNormal" );
gl.vertexAttribPointer( vNormal, 3, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vNormal );
```



shadedCube.js (8/11)

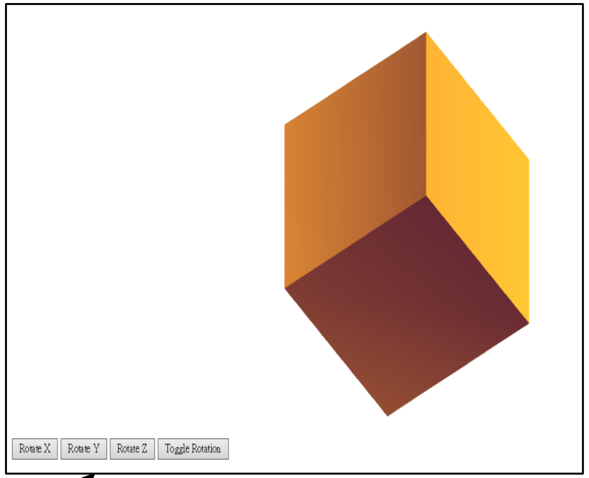
```
var vBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer );
gl.bufferData( gl.ARRAY_BUFFER, flatten(pointsArray), gl.STATIC_DRAW );

var vPosition = gl.getAttribLocation(program, "vPosition");
gl.vertexAttribPointer(vPosition, 4, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);
```



shadedCube.js (9/11)

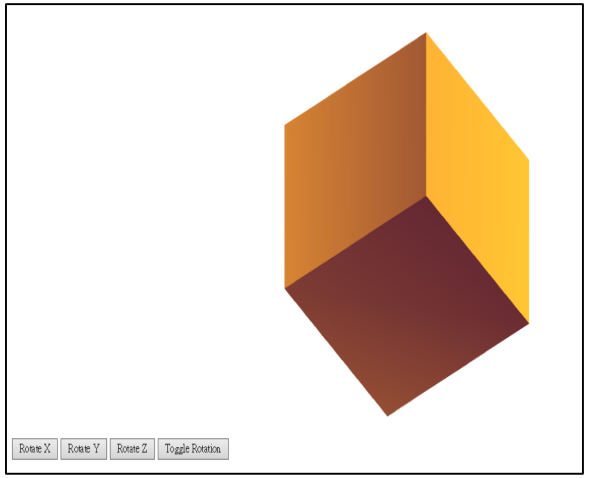
```
thetaLoc = gl.getUniformLocation(program, "theta");

// viewerPos = vec3(0.0, 0.0, -20.0 );

projection = ortho(-1, 1, -1, 1, -100, 100);

ambientProduct  = mult(lightAmbient, materialAmbient);
diffuseProduct   = mult(lightDiffuse,    materialDiffuse);
specularProduct = mult(lightSpecular, materialSpecular);

document.getElementById("ButtonX").onclick = function() {axis = xAxis;};
document.getElementById("ButtonY").onclick = function() {axis = yAxis;};
document.getElementById("ButtonZ").onclick = function() {axis = zAxis;};
document.getElementById("ButtonT").onclick = function() {flag = !flag;};
```

$$I = k_d I_d (\mathbf{L} \cdot \mathbf{N}) + k_s I_s (\mathbf{N} \cdot \mathbf{H})^{\alpha} + k_a I_a$$



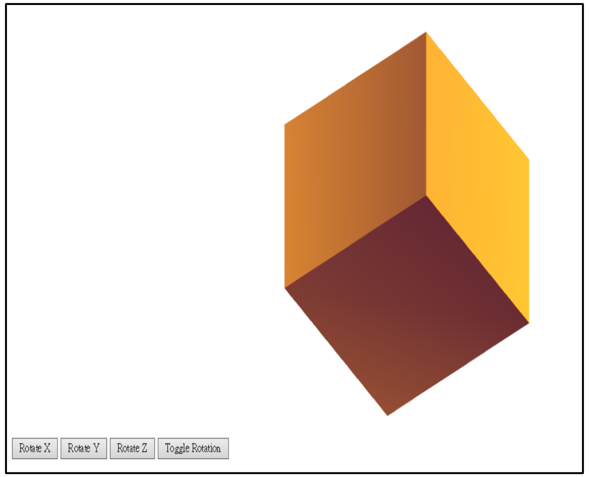
shadedCube.js (10/11)

```
    gl.uniform4fv(gl.getUniformLocation(program, "ambientProduct"),  flatten(ambientProduct) );
    gl.uniform4fv(gl.getUniformLocation(program, "diffuseProduct"),    flatten(diffuseProduct) );
    gl.uniform4fv(gl.getUniformLocation(program, "specularProduct"), flatten(specularProduct) );
    gl.uniform4fv(gl.getUniformLocation(program, "lightPosition"),       flatten(lightPosition) );

    gl.uniform1f(gl.getUniformLocation(program,  "shininess"),materialShininess);

    gl.uniformMatrix4fv( gl.getUniformLocation(program, "projectionMatrix"), false, flatten(projection));

    render();
}   // end of window.onload
```



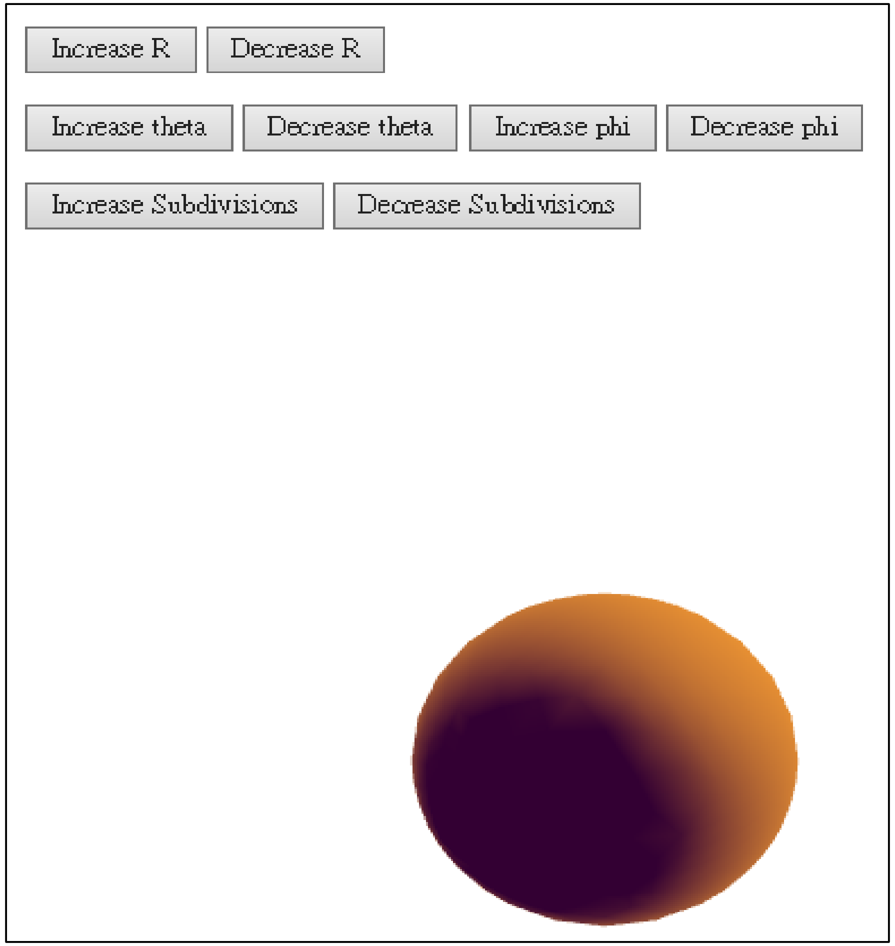
shadedCube.js (11/11)

```
var render = function() {

    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    if(flag) theta[axis] += 2.0;

    modelView = mat4();
    modelView = mult(modelView, rotate(theta[xAxis], [1, 0, 0] ));
    modelView = mult(modelView, rotate(theta[yAxis], [0, 1, 0] ));
    modelView = mult(modelView, rotate(theta[zAxis], [0, 0, 1] ));

    gl.uniformMatrix4fv( gl.getUniformLocation(program, "modelViewMatrix"), false, flatten(modelView) );

    gl.drawArrays( gl.TRIANGLES, 0, numVertices );

    requestAnimFrame(render);

}  // end of render()
```



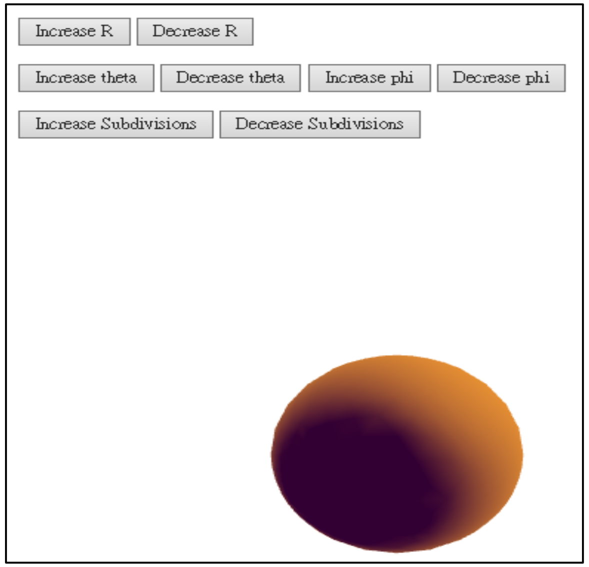
Sample Programs: shadedSphere1.html, shadedSphere1.js

Shaded sphere using true normals and per vertex shading



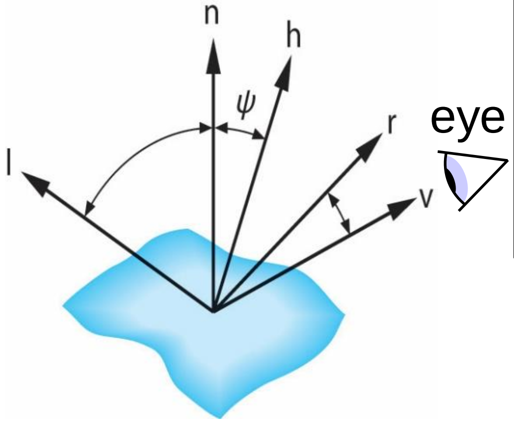
shadedSphere1.html (1/6)

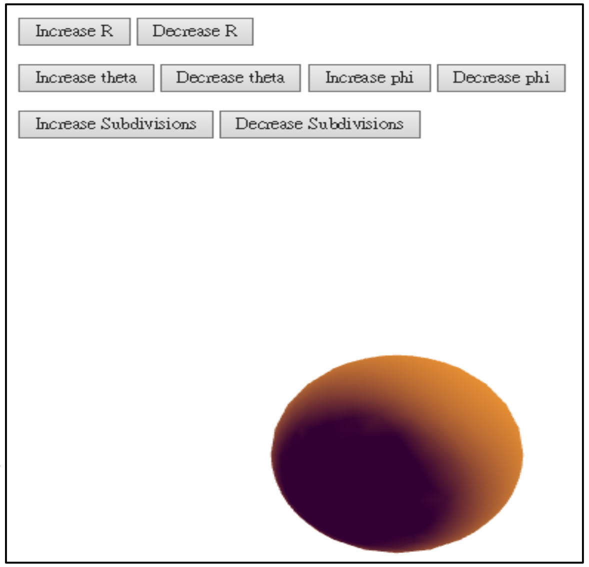
```
<!DOCTYPE html>
<html>

<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;
attribute vec4 vNormal;

varying vec4 fColor;

uniform vec4 ambientProduct, diffuseProduct, specularProduct;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;
uniform float  shininess;
```



shadedSphere1.html (2/6)

```
void
main()
{
    vec3 pos = -(modelViewMatrix * vPosition).xyz;
    vec3 light = lightPosition.xyz;
    vec3 L = normalize( light - pos );

    vec3 E = normalize( -pos );
    vec3 H = normalize( L + E );

    // Transform vertex normal into eye coordinates

    vec3 N = normalize( (modelViewMatrix*vNormal).xyz);
```



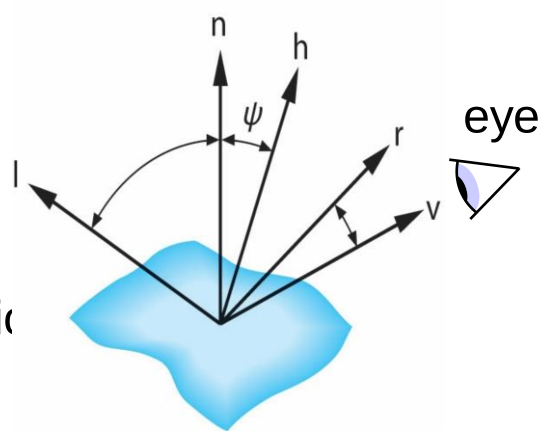


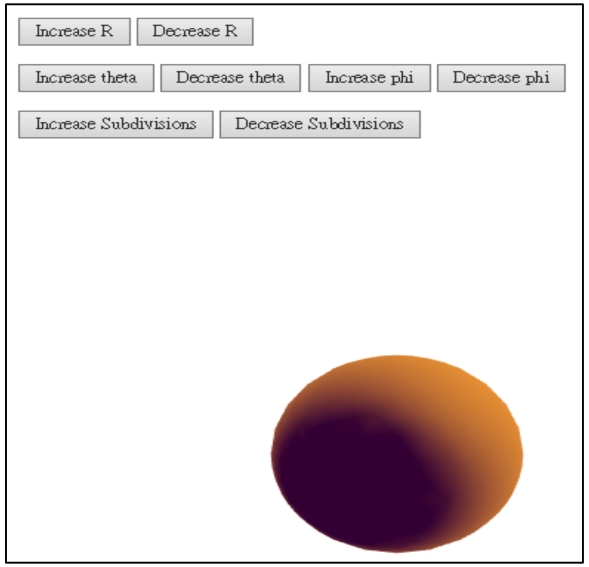
$I = k_d I_d \; (\mathbf{L} \cdot \mathbf{N}) \; + k_s I_s \, (\mathbf{N} \cdot \mathbf{H})^{\alpha} + k_a I_a$

where $\mathbf{H} = (\mathbf{L}+\mathbf{E})/2$ and α is shineness

shadedSphere1.html (3/6)

```
    // Compute terms in the illumination equation
    vec4 ambient = ambientProduct;

    float Kd = max( dot(L, N), 0.0 );
    vec4  diffuse = Kd*diffuseProduct;

    float Ks = pow( max(dot(N, H), 0.0), shininess );
    vec4  specular = Ks * specularProduct;

    if( dot(L, N) < 0.0 ) { specular = vec4(0.0, 0.0, 0.0, 1.0);  }

    gl_Position = projectionMatrix * modelViewMatrix * vPositi

    fColor = ambient + diffuse +specular;

    fColor.a = 1.0;
}
</script>
```



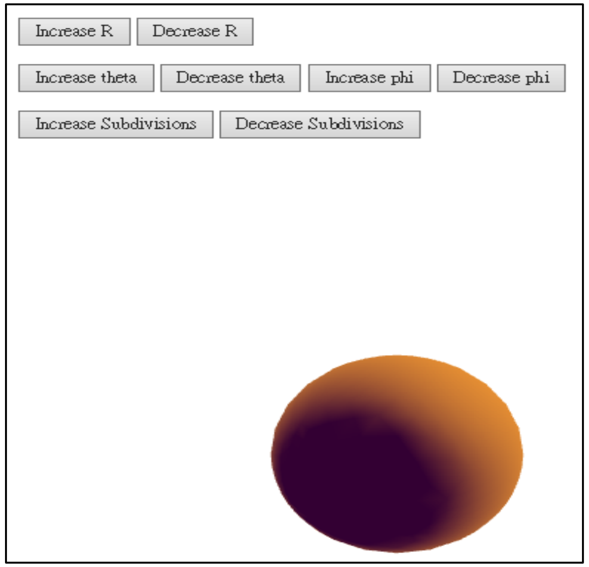


$$I = k_d I_d \ (\mathbf{L} \cdot \mathbf{N}) \ + k_s I_s (\mathbf{N} \cdot \mathbf{H})^{\alpha} + k_a I_a$$

where $\mathbf{H} = (\mathbf{L}+\mathbf{E})/2$ and α is shineness

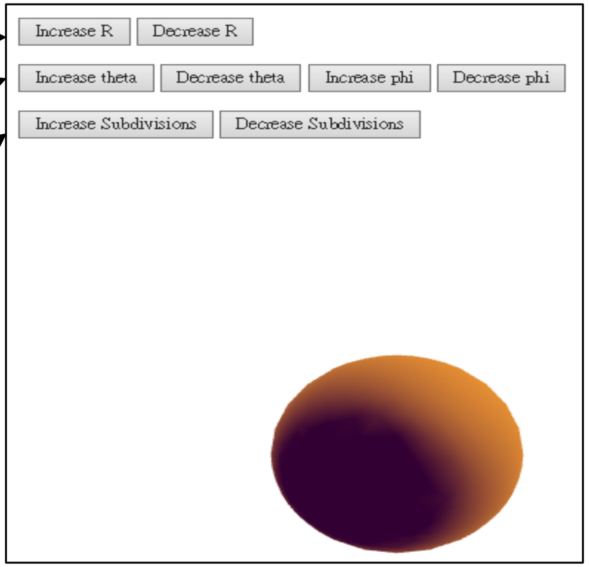
shadedSphere1.html (4/6)

```
<script id="fragment-shader" type="x-shader/x-fragment">

precision mediump float;

varying vec4 fColor;

void
main()
{

    gl_FragColor = fColor;

}
</script>
```



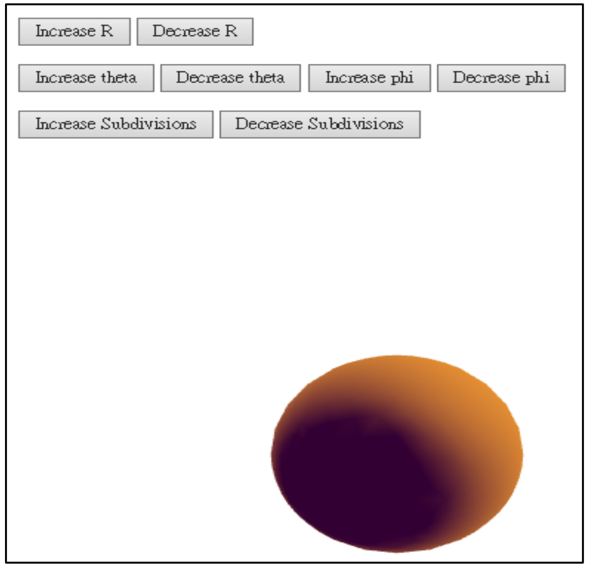
shadedSphere1.html (5/6)

```
<p> </p>
<button id = "Button0">Increase R</button>
<button id = "Button1">Decrease R</button>

<p> </p>
<button id = "Button2">Increase theta</button>
<button id = "Button3">Decrease theta</button>
<button id = "Button4">Increase phi</button>
<button id = "Button5">Decrease phi</button>
<p> </p>
<button id = "Button6">Increase Subdivisions</button>
<button id = "Button7">Decrease Subdivisions</button>

<p></p>
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="shadedSphere1.js"></script>
```



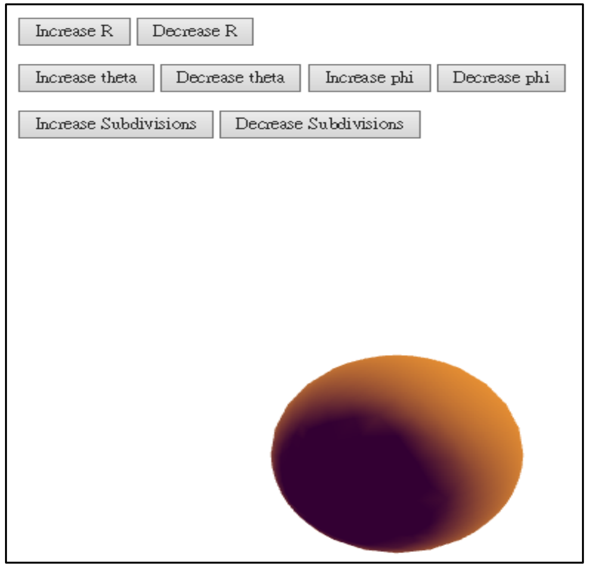
shadedSphere1.html (6/6)

```
<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



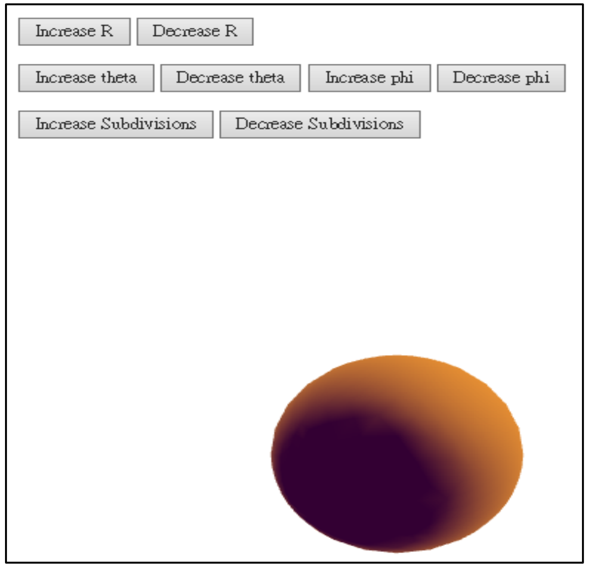
shadedSphere1.js (1/13)

```
var canvas;
var gl;

var numTimesToSubdivide = 3;

var index = 0;

var pointsArray = [];
var normalsArray = [];
```



shadedSphere1.js (2/13)

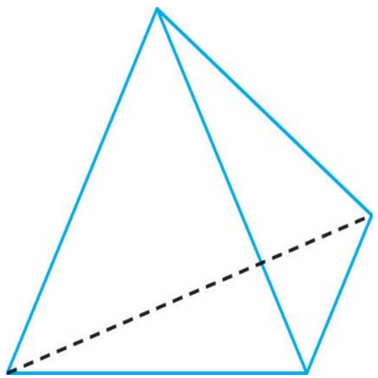
```
var near   = -10;
var far      = 10;
var radius = 1.5;
var theta  = 0.0;
var phi      = 0.0;
var dr        = 5.0 * Math.PI/180.0;

var left   = -3.0;
var right = 3.0;
var ytop = 3.0;
var bottom = -3.0;
```

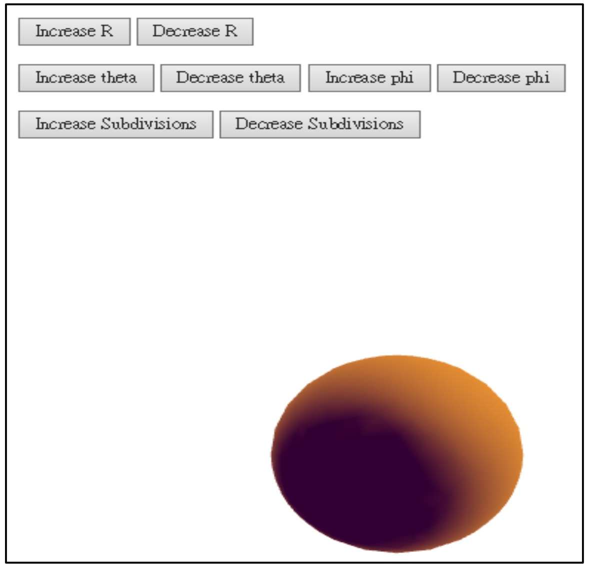


shadedSphere1.js (3/13)

```
var va = vec4(0.0,           0.0,       -1.0,      1);
var vb = vec4(0.0,           0.942809, 0.333333, 1);
var vc = vec4(-0.816497, -0.471405, 0.333333, 1);
var vd = vec4( 0.816497, -0.471405, 0.333333, 1);

var lightPosition   = vec4(1.0, 1.0, 1.0, 0.0 );
var lightAmbient  =  vec4(0.2, 0.2, 0.2, 1.0 );
var lightDiffuse    =  vec4(1.0, 1.0, 1.0, 1.0 );
var lightSpecular =  vec4(1.0, 1.0, 1.0, 1.0 );

var materialAmbient   = vec4( 1.0, 0.0, 1.0, 1.0 );
var materialDiffuse     = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialSpecular  = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialShininess = 100.0;
```

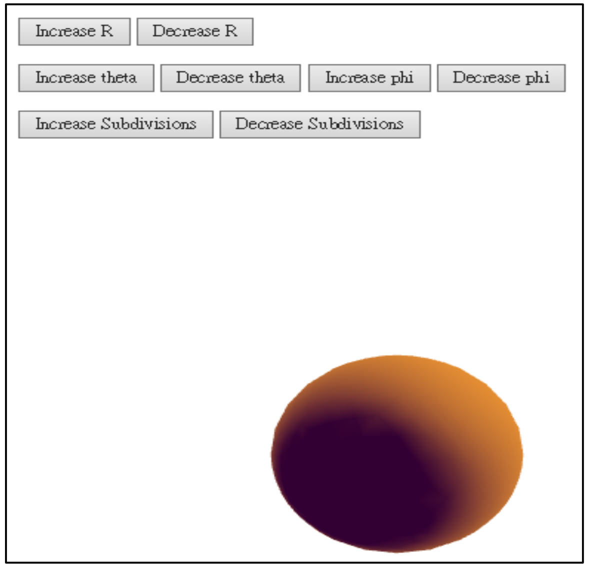
$$\left\{\begin{array}{l}(0.0, 0.0, -1.0)\\ (0.0, 2\sqrt{2}/3, 1/3)\\ (-\sqrt{6}/3, -\sqrt{2}/3, 1/3)\\ (\sqrt{6}/3, -\sqrt{2}/3, 1/3)\end{array}\right.$$





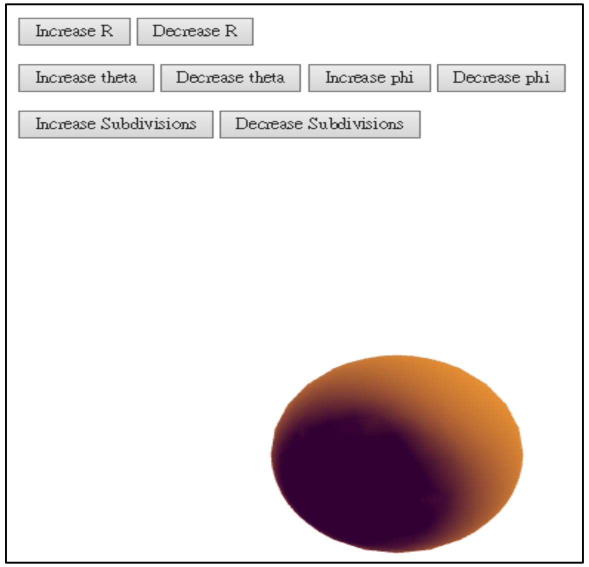
shadedSphere1.js (4/13)

```
var ctm;
var ambientColor, diffuseColor, specularColor;

var modelViewMatrix, projectionMatrix;
var modelViewMatrixLoc, projectionMatrixLoc;
var eye;
var at  = vec3(0.0, 0.0, 0.0);
var up = vec3(0.0, 1.0, 0.0);
```



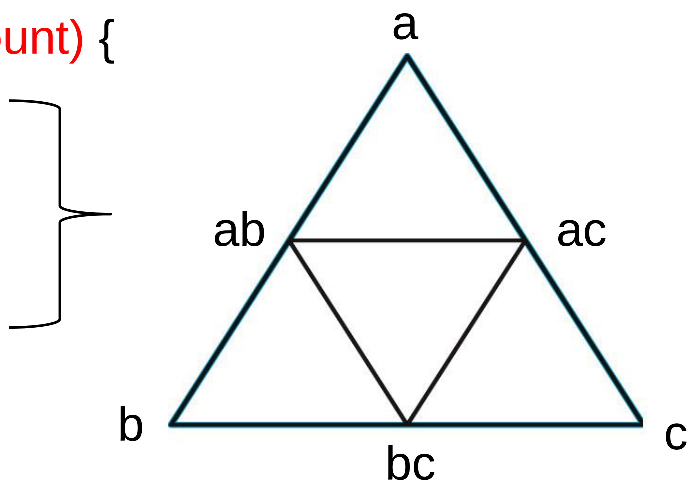
shadedSphere1.js (5/13)

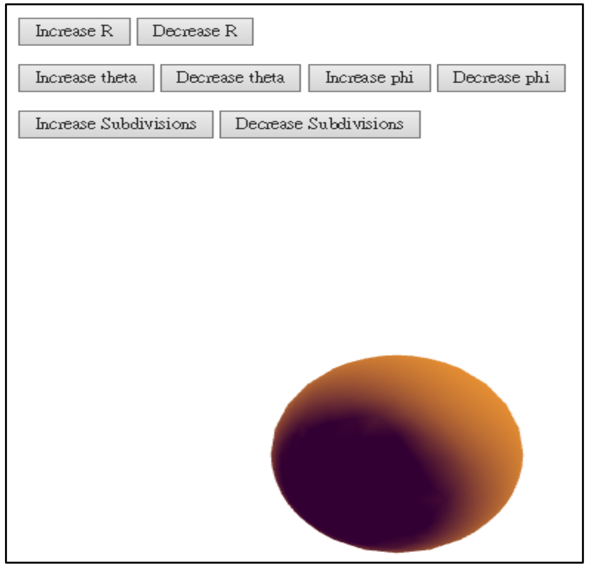
```
function triangle(a, b, c) {

    normalsArray.push(a);
    normalsArray.push(b);
    normalsArray.push(c);

    pointsArray.push(a);
    pointsArray.push(b);
    pointsArray.push(c);

    index += 3;
}
```



shadedSphere1.js (6/13)

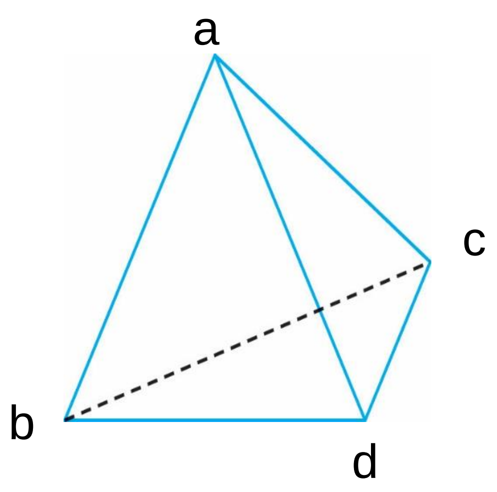
```
function divideTriangle(a, b, c, count) {
    if ( count > 0 ) {
        var ab = mix( a, b, 0.5);
        var ac = mix( a, c, 0.5);
        var bc = mix( b, c, 0.5);

        ab = normalize(ab, true);
        ac = normalize(ac, true);
        bc = normalize(bc, true);

        divideTriangle( a,  ab, ac, count - 1 );
        divideTriangle( ab,  b, bc, count - 1 );
        divideTriangle( bc,   c, ac, count - 1 );
        divideTriangle( ab, bc, ac, count - 1 );
    }
    else {
        triangle( a, b, c );
    }
}
```

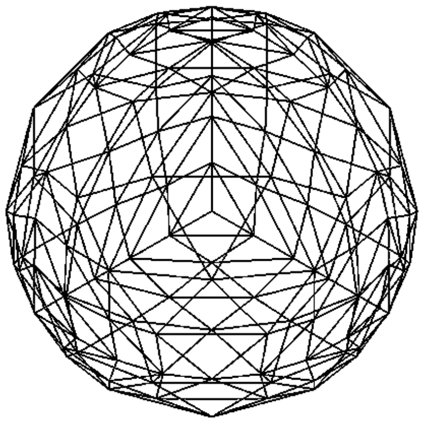


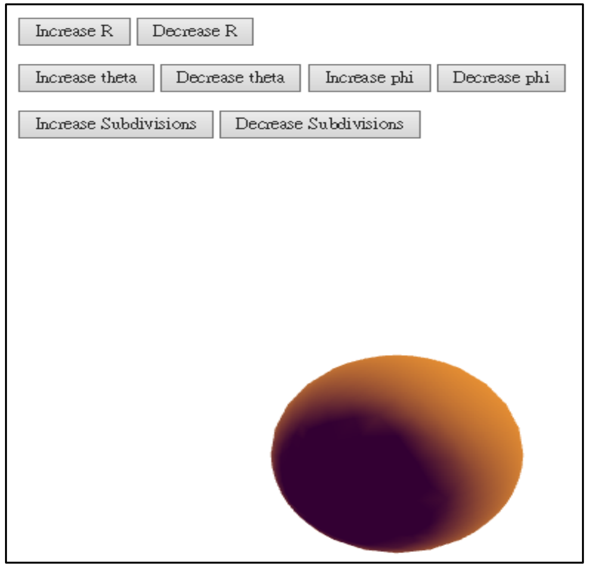


shadedSphere1.js (7/13)

```
function tetrahedron(a, b, c, d, n) {
    divideTriangle(a, b, c, n);
    divideTriangle(d, c, b, n);
    divideTriangle(a, d, b, n);
    divideTriangle(a, c, d, n);
}
```

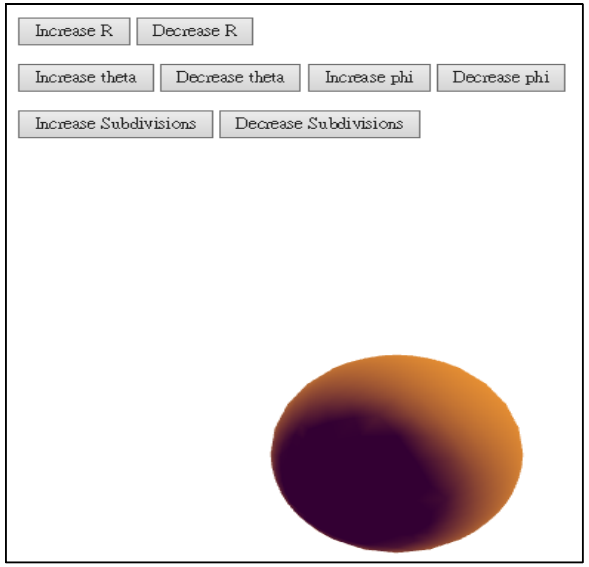






shadedSphere1.js (8/13)

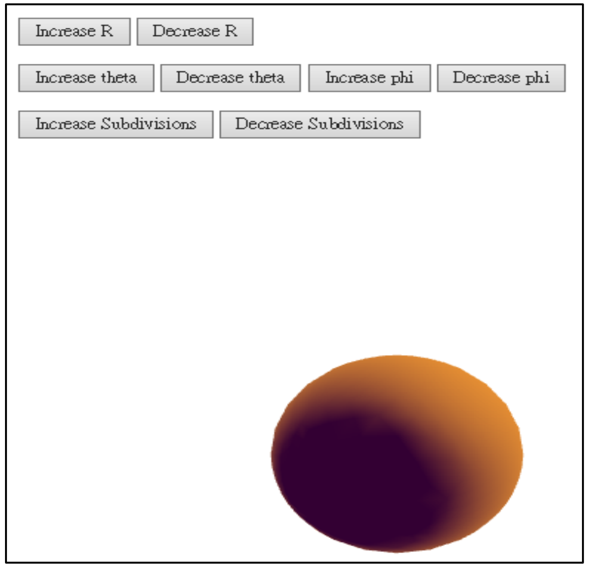
```
window.onload = function init() {

    canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );

    gl.enable(gl.DEPTH_TEST);
```



shadedSphere1.js (9/13)

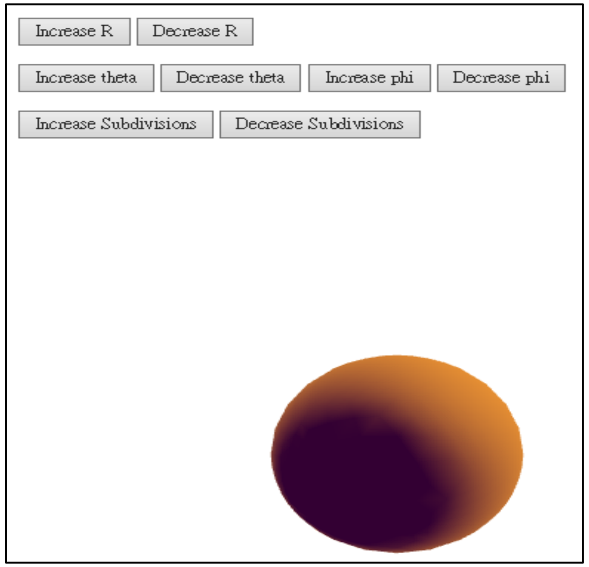
```
//  Load shaders and initialize attribute buffers

var program = initShaders( gl, "vertex-shader", "fragment-shader" );
gl.useProgram( program );

ambientProduct  = mult(lightAmbient, materialAmbient);
diffuseProduct    = mult(lightDiffuse, materialDiffuse);
specularProduct = mult(lightSpecular, materialSpecular);

tetrahedron(va, vb, vc, vd, numTimesToSubdivide);

var nBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, nBuffer);
gl.bufferData( gl.ARRAY_BUFFER, flatten(normalsArray), gl.STATIC_DRAW );

var vNormal = gl.getAttribLocation( program, "vNormal" );
gl.vertexAttribPointer( vNormal, 4, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray(vNormal);
```

$$I = k_d I_d \, (\mathbf{L} \cdot \mathbf{N}) + k_s I_s \, (\mathbf{N} \cdot \mathbf{H})^{\alpha} + k_a I_a$$



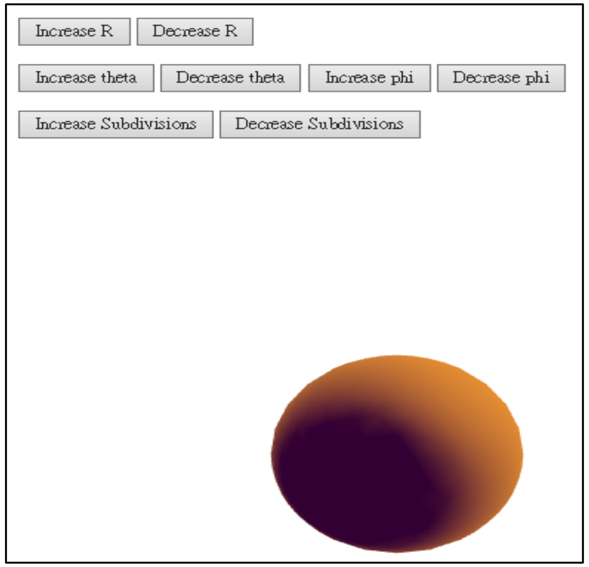
shadedSphere1.js (10/13)

```
var vBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
gl.bufferData(gl.ARRAY_BUFFER, flatten(pointsArray), gl.STATIC_DRAW);

var vPosition = gl.getAttribLocation( program, "vPosition");
gl.vertexAttribPointer(vPosition, 4, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);

modelViewMatrixLoc = gl.getUniformLocation( program, "modelViewMatrix" );
projectionMatrixLoc   = gl.getUniformLocation( program, "projectionMatrix" );
```



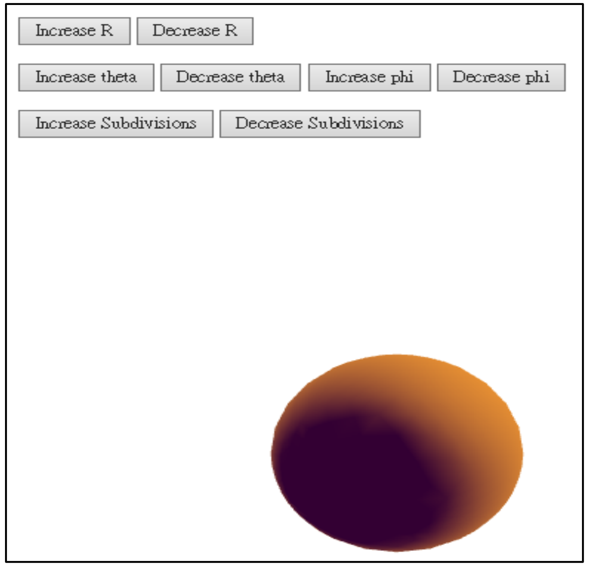
shadedSphere1.js (11/13)

```
document.getElementById("Button0").onclick = function() {radius *= 2.0;};
document.getElementById("Button1").onclick = function() {radius *= 0.5;};
document.getElementById("Button2").onclick = function() {theta += dr;};
document.getElementById("Button3").onclick = function() {theta -= dr;};
document.getElementById("Button4").onclick = function() {phi += dr;};
document.getElementById("Button5").onclick = function() {phi -= dr;};

document.getElementById("Button6").onclick = function() {
    numTimesToSubdivide++;
    index = 0;
    pointsArray = [];
    normalsArray = [];
    init();
};
```



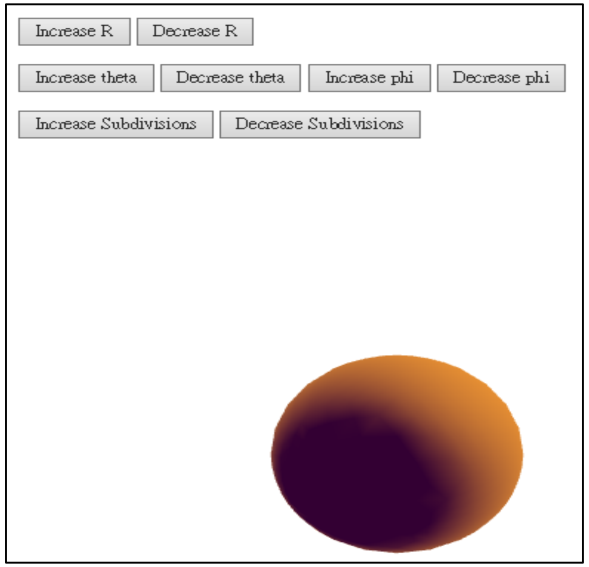
shadedSphere1.js (12/13)

```
document.getElementById("Button7").onclick = function() {
        if(numTimesToSubdivide) numTimesToSubdivide--;
        index = 0;
        pointsArray = [];
        normalsArray = [];
        init();
    };

gl.uniform4fv( gl.getUniformLocation(program,  "ambientProduct"), flatten(ambientProduct) );
gl.uniform4fv( gl.getUniformLocation(program,  "diffuseProduct"),   flatten(diffuseProduct) );
gl.uniform4fv( gl.getUniformLocation(program,  "specularProduct"),flatten(specularProduct) );
gl.uniform4fv( gl.getUniformLocation(program,  "lightPosition"),      flatten(lightPosition) );
gl.uniform1f(  gl.getUniformLocation(program,    "shininess"),materialShininess );

 render();
}   // end of window.onload
```



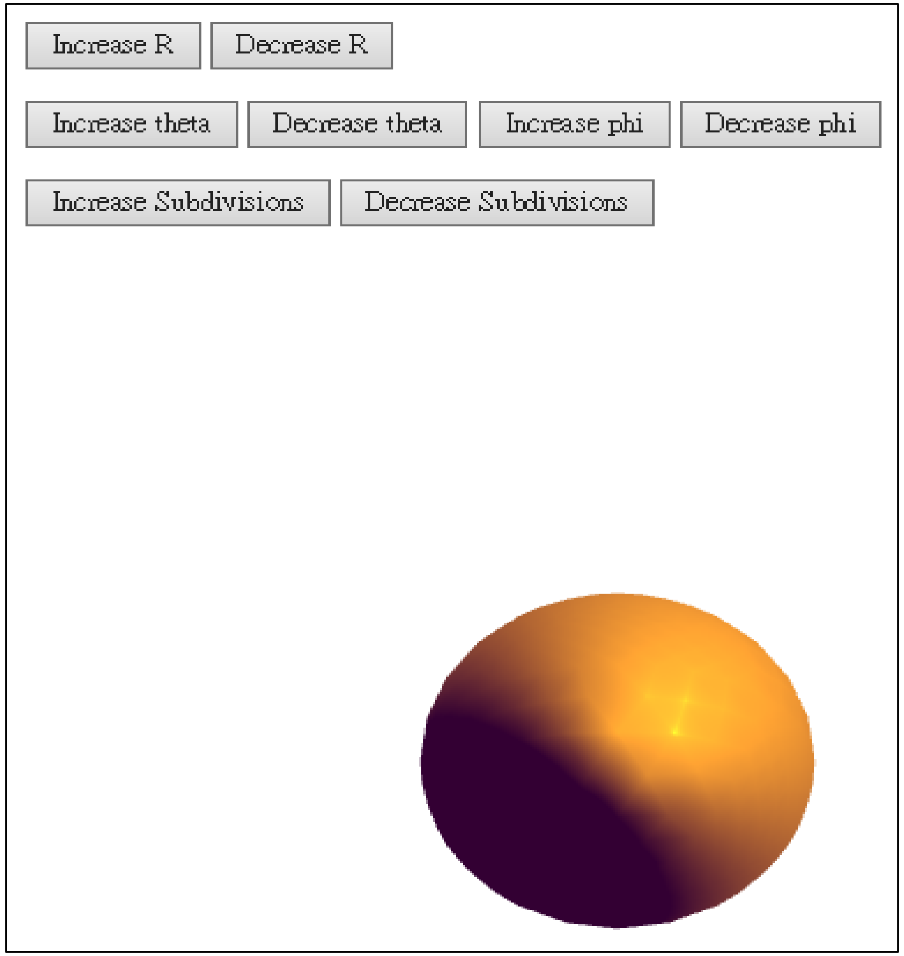
shadedSphere1.js (13/13)

```
function render() {

    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    eye = vec3(radius*Math.sin(theta)*Math.cos(phi),
                        radius*Math.sin(theta)*Math.sin(phi),
                        radius*Math.cos(theta));
    modelViewMatrix = lookAt(eye, at , up);
    projectionMatrix   = ortho(left, right, bottom, ytop, near, far);

    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(modelViewMatrix) );
    gl.uniformMatrix4fv(projectionMatrixLoc,   false, flatten(projectionMatrix) );

    for( var i=0; i<index; i+=3)
        gl.drawArrays( gl.TRIANGLES, i, 3 );

    window.requestAnimFrame(render);
}   // end of render()
```



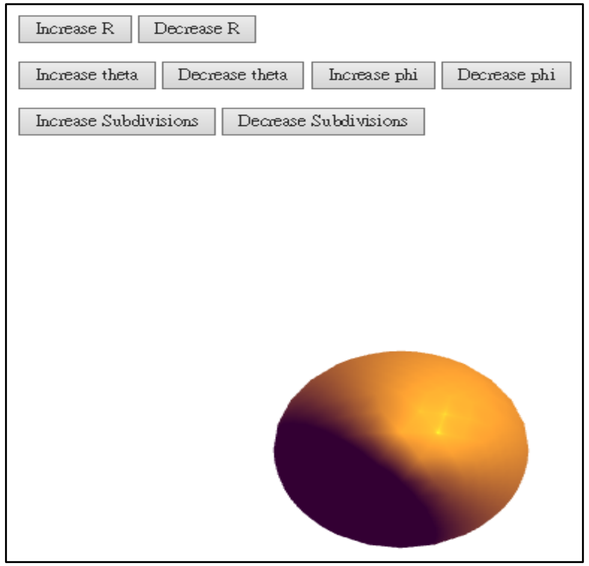
Sample Programs: shadedSphere2.html, shadedSphere2.js

Shaded sphere using true normals and per fragment shading



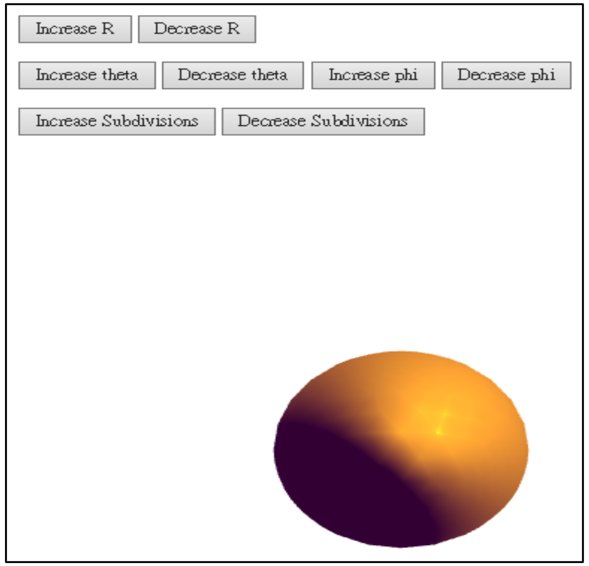
shadedSphere2.html (1/6)

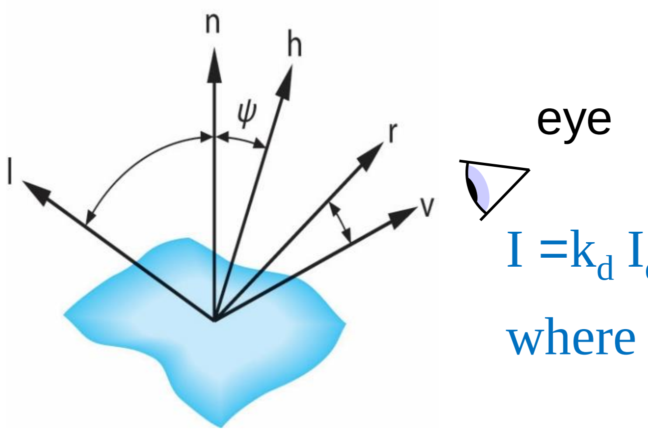
```
<!DOCTYPE html>
<html>

<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;
attribute vec4 vNormal;

varying vec3 N, L, E;

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;
```



shadedSphere2.html (2/6)

```
void main()
{
    vec3 pos = -(modelViewMatrix * vPosition).xyz;
    vec3 light = lightPosition.xyz;
    L = normalize( light - pos );
    E =  -pos;
    N = normalize( (modelViewMatrix*vNormal).xyz);
    gl_Position = projectionMatrix * modelViewMatrix * vPosition;

}
</script>
```



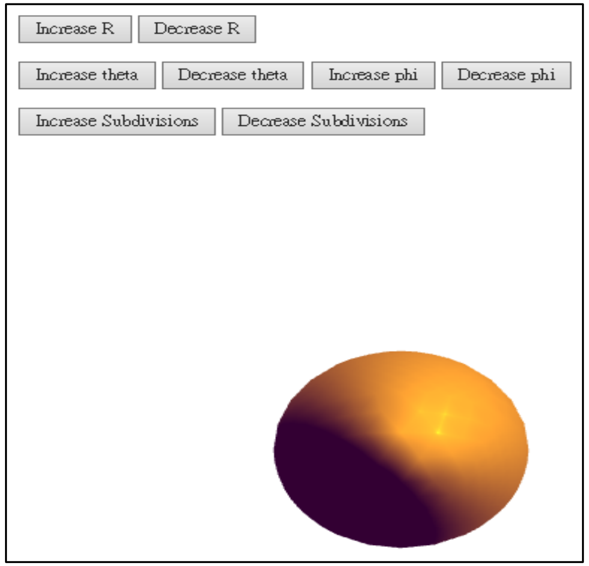


$I = k_d I_d \; (\mathbf{L} \cdot \mathbf{N}) \; + k_s I_s (\mathbf{N} \cdot \mathbf{H})^{\alpha} + k_a I_a$

where $\mathbf{H} = (\mathbf{L}+\mathbf{E})/2$ and α is shineness

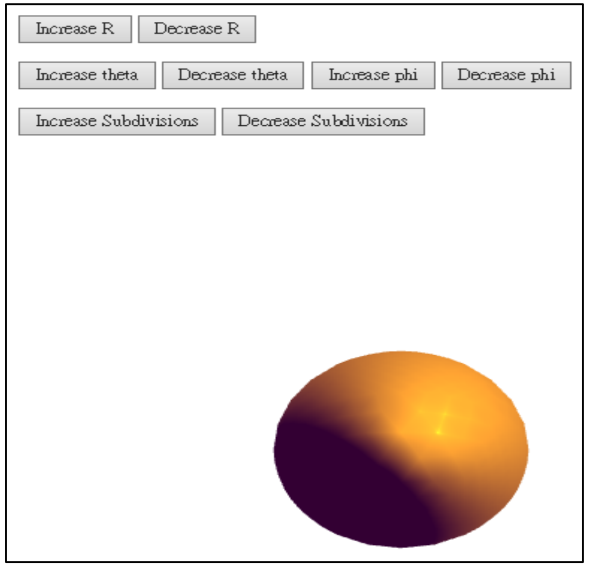
shadedSphere2.html (3/6)

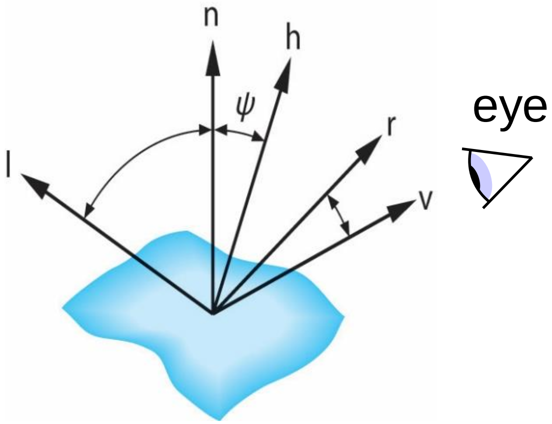
```
<script id="fragment-shader" type="x-shader/x-fragment">

precision mediump float;

uniform vec4 ambientProduct;
uniform vec4 diffuseProduct;
uniform vec4 specularProduct;
uniform float shininess;
varying vec3 N, L, E;
```



shadedSphere2.html (4/6)

```
void main()
{  vec4 fColor;

    vec3 H = normalize( L + E );
    vec4 ambient = ambientProduct;

    float Kd = max( dot(L, N), 0.0 );
    vec4  diffuse = Kd*diffuseProduct;

    float Ks = pow( max(dot(N, H), 0.0), shininess );
    vec4  specular = Ks * specularProduct;
    if( dot(L, N) < 0.0 ) specular = vec4(0.0, 0.0, 0.0, 1.0);

    fColor = ambient + diffuse +specular;
    fColor.a = 1.0;
    gl_FragColor = fColor;
}
</script>
```



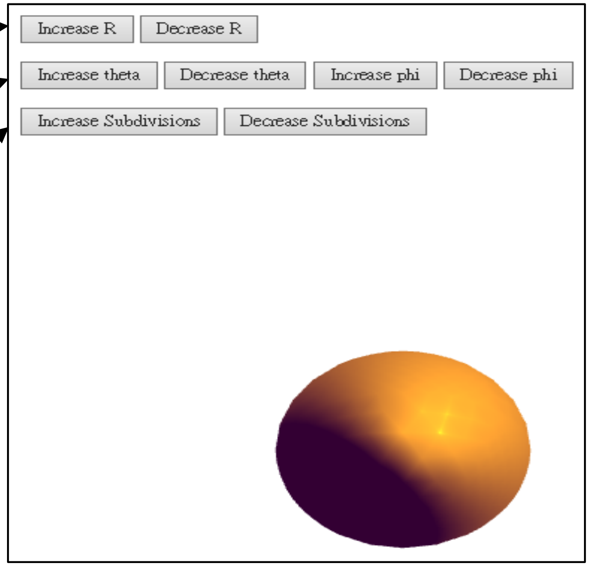


$$I = k_d I_d \ (\mathbf{L} \cdot \mathbf{N}) \ + k_s I_s (\mathbf{N} \cdot \mathbf{H})^{\alpha} + k_a I_a$$

where $\mathbf{H} = (\mathbf{L}+\mathbf{E})/2$ and α is shineness

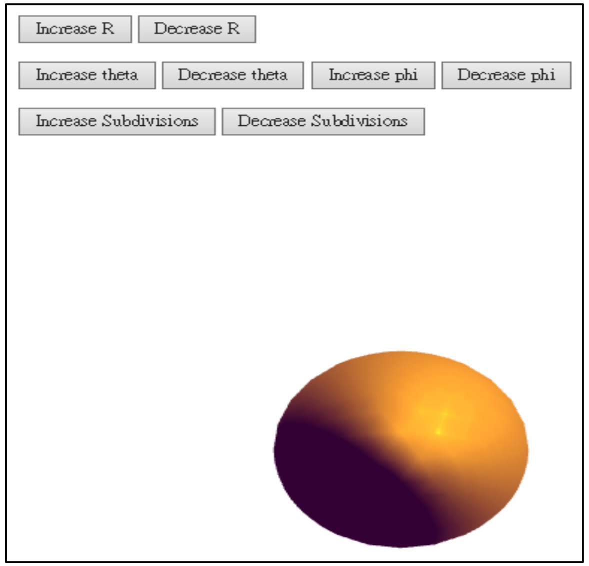
shadedSphere2.html (5/6)

```
<p> </p>
<button id = "Button0">Increase R</button>
<button id = "Button1">Decrease R</button>

<p> </p>
<button id = "Button2">Increase theta</button>
<button id = "Button3">Decrease theta</button>
<button id = "Button4">Increase phi</button>
<button id = "Button5">Decrease phi</button>
<p> </p>
<button id = "Button6">Increase Subdivisions</button>
<button id = "Button7">Decrease Subdivisions</button>

<p></p>
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="shadedSphere2.js"></script>
```



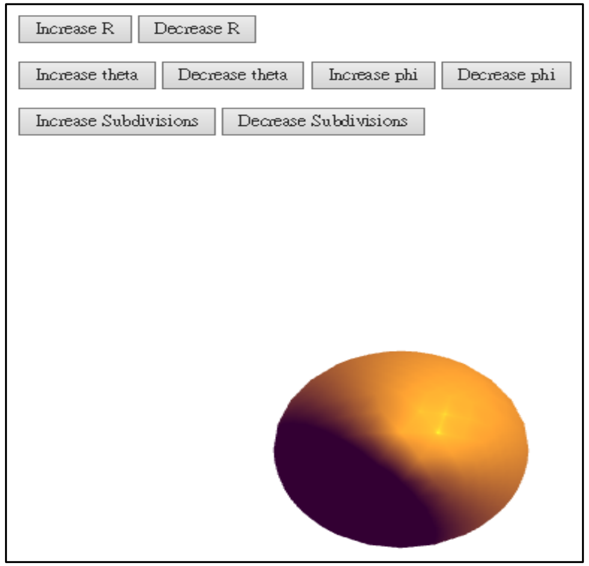
shadedSphere2.html (6/6)

```
<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



shadedSphere2.js (1/12)

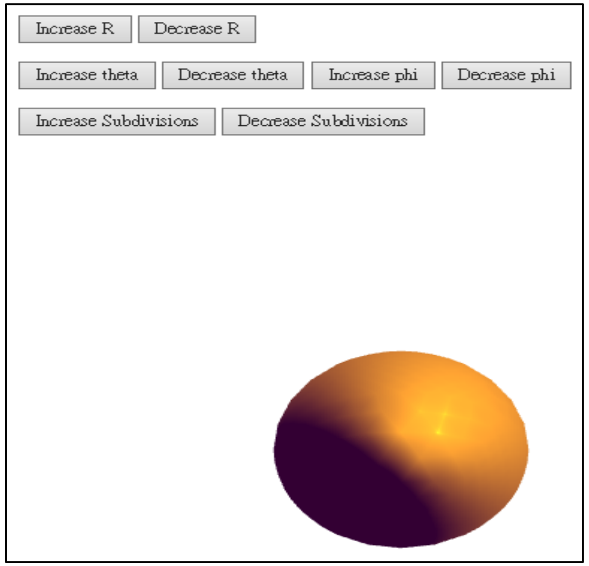
```
var canvas;
var gl;

var numTimesToSubdivide = 3;

var index = 0;

var pointsArray = [];
var normalsArray = [];

var near   = -10;
var far     =  10;
var radius = 1.5;
var theta  = 0.0;
var phi     = 0.0;
var dr       = 5.0 * Math.PI/180.0;
```

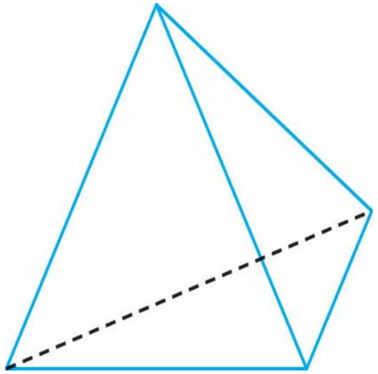


shadedSphere2.js (2/12)

```
var left = -3.0;
var right = 3.0;
var ytop =3.0;
var bottom = -3.0;

var va = vec4( 0.0,           0.0,        -1.0,       1);
var vb = vec4( 0.0,           0.942809, 0.333333, 1);
var vc = vec4(-0.816497, -0.471405, 0.333333, 1);
var vd = vec4(0.816497,  -0.471405, 0.333333, 1);

var lightPosition = vec4(1.0, 1.0, 1.0, 0.0 );
var lightAmbient = vec4(0.2, 0.2, 0.2, 1.0 );
var lightDiffuse   = vec4(1.0, 1.0, 1.0, 1.0 );
var lightSpecular= vec4(1.0, 1.0, 1.0, 1.0 );
```

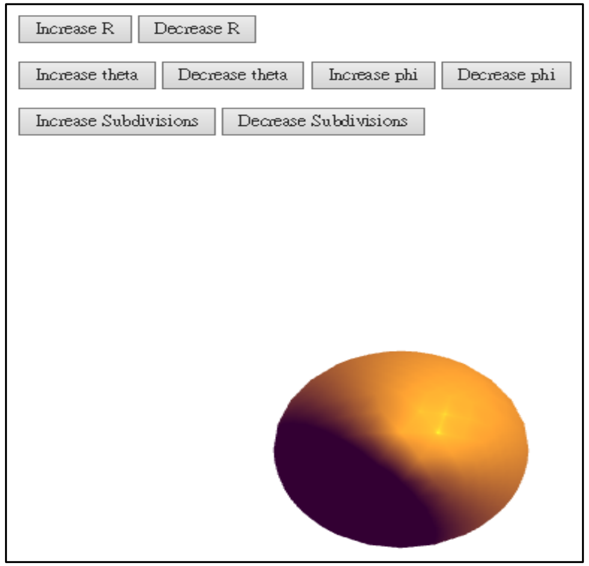
$$\begin{cases} (0.0,\ 0.0,\ -1.0) \\ (0.0, 2\sqrt{2}/3,\ 1/3) \\ (-\sqrt{6}/3, -\sqrt{2}/3,\ 1/3) \\ (\sqrt{6}/3, -\sqrt{2}/3,\ 1/3) \end{cases}$$





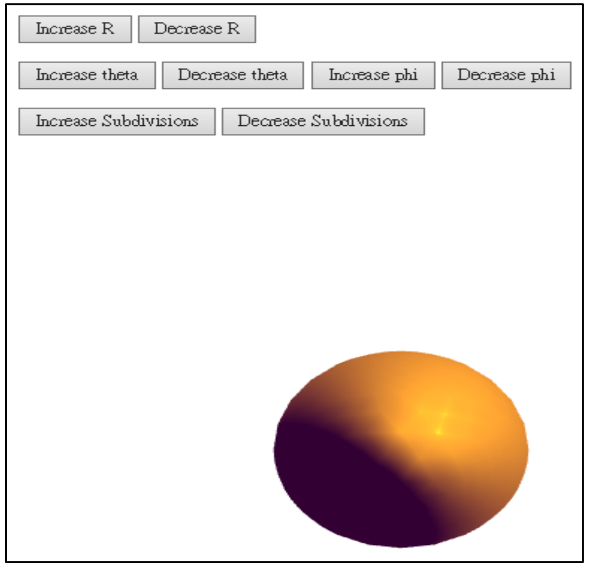
shadedSphere2.js (3/12)

```
var materialAmbient = vec4( 1.0, 0.0, 1.0, 1.0 );
var materialDiffuse    = vec4(1.0, 0.8, 0.0, 1.0 );
var materialSpecular = vec4(1.0, 0.8, 0.0, 1.0 );
var materialShininess = 100.0;

var ctm;
var ambientColor, diffuseColor, specularColor;

var modelViewMatrix, projectionMatrix;
var modelViewMatrixLoc, projectionMatrixLoc;
var eye;
var at  = vec3(0.0, 0.0, 0.0);
var up = vec3(0.0, 1.0, 0.0);
```



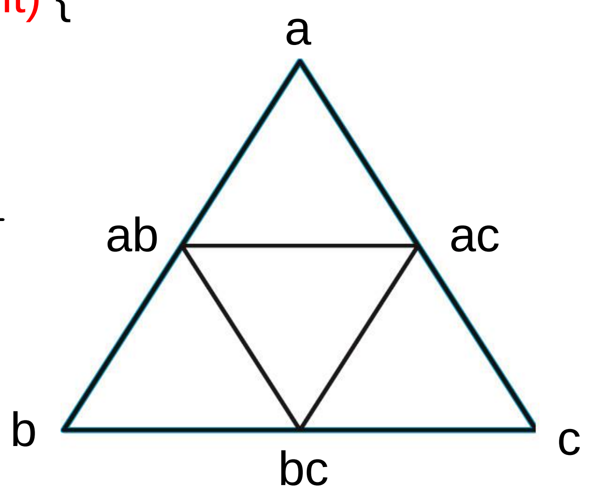
shadedSphere2.js (4/12)

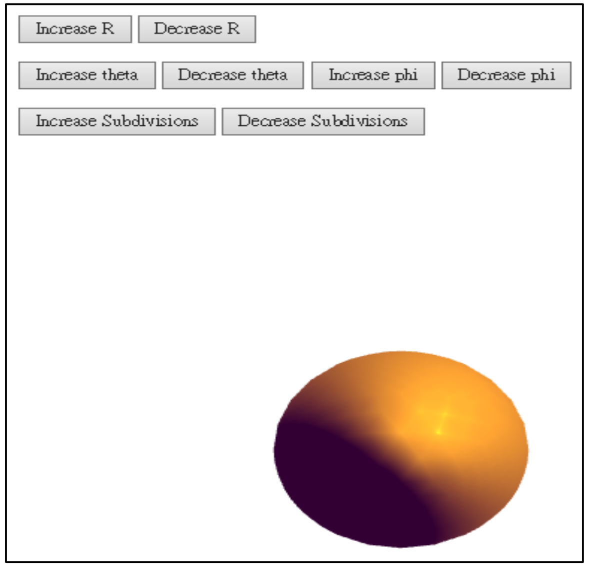
```
function triangle(a, b, c) {

    normalsArray.push(a);
    normalsArray.push(b);
    normalsArray.push(c);

    pointsArray.push(a);
    pointsArray.push(b);
    pointsArray.push(c);

    index += 3;
}
```



shadedSphere2.js (5/12)

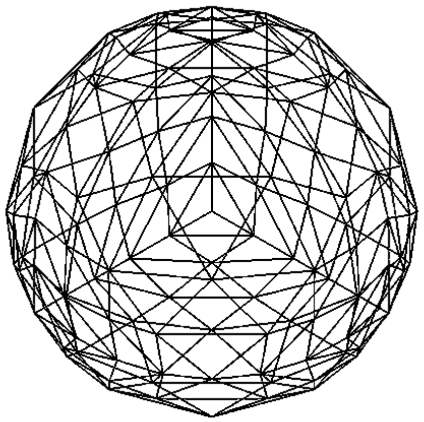
```
function divideTriangle(a, b, c, count) {
   if ( count > 0 ) {

      var ab = mix( a, b, 0.5);
      var ac = mix( a, c, 0.5);
      var bc = mix( b, c, 0.5);

      ab = normalize(ab, true);
      ac = normalize(ac, true);
      bc = normalize(bc, true);

      divideTriangle( a, ab, ac, count - 1 );
      divideTriangle( ab, b, bc, count - 1 );
      divideTriangle( bc, c, ac, count - 1 );
      divideTriangle( ab, bc, ac, count - 1 );
   }
   else {  triangle( a, b, c );  }
}
```

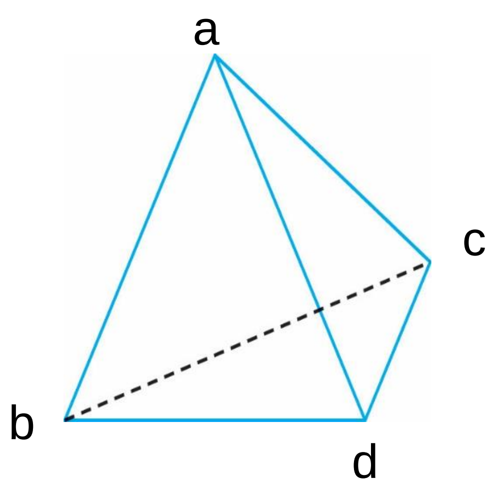


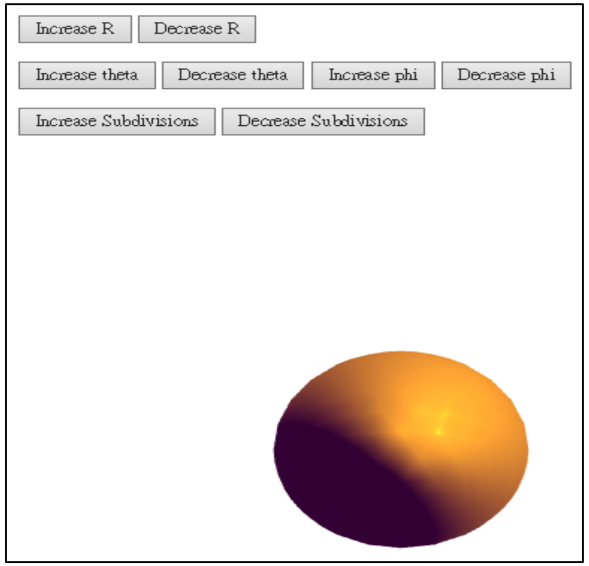


shadedSphere2.js (6/12)

```
function tetrahedron(a, b, c, d, n) {
    divideTriangle(a, b, c, n);
    divideTriangle(d, c, b, n);
    divideTriangle(a, d, b, n);
    divideTriangle(a, c, d, n);
}
```

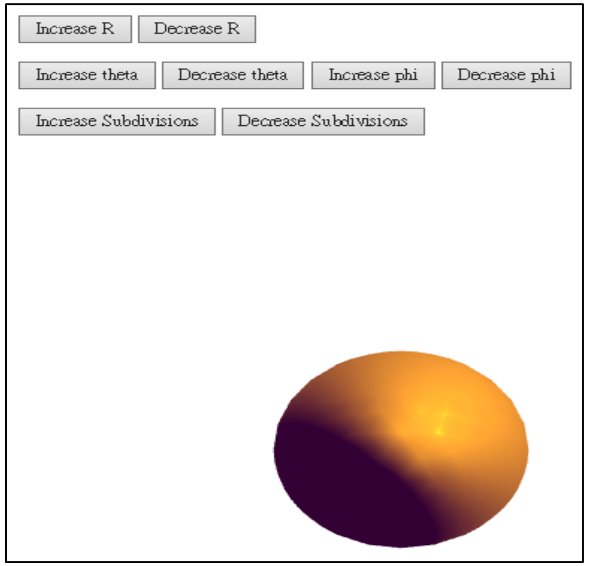






shadedSphere2.js (7/12)

```
window.onload = function init() {

    canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );

    gl.enable(gl.DEPTH_TEST);
```

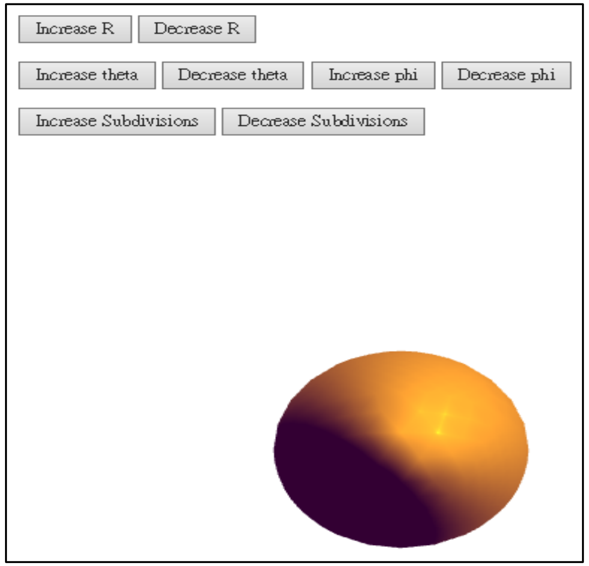


shadedSphere2.js (8/12)

```
//
//  Load shaders and initialize attribute buffers
//
var program = initShaders( gl, "vertex-shader", "fragment-shader" );
gl.useProgram( program );
```

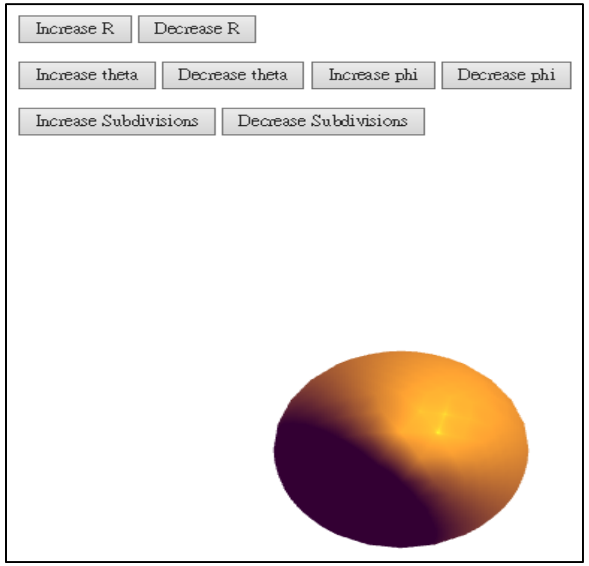
$$I = k_d I_d \, (\mathbf{L} \cdot \mathbf{N}) + k_s I_s \, (\mathbf{N} \cdot \mathbf{H})^{\alpha} + k_a I_a$$

```
ambientProduct  = mult(lightAmbient, materialAmbient);
diffuseProduct    = mult(lightDiffuse,    materialDiffuse);
specularProduct = mult(lightSpecular, materialSpecular);


tetrahedron(va, vb, vc, vd, numTimesToSubdivide);
```



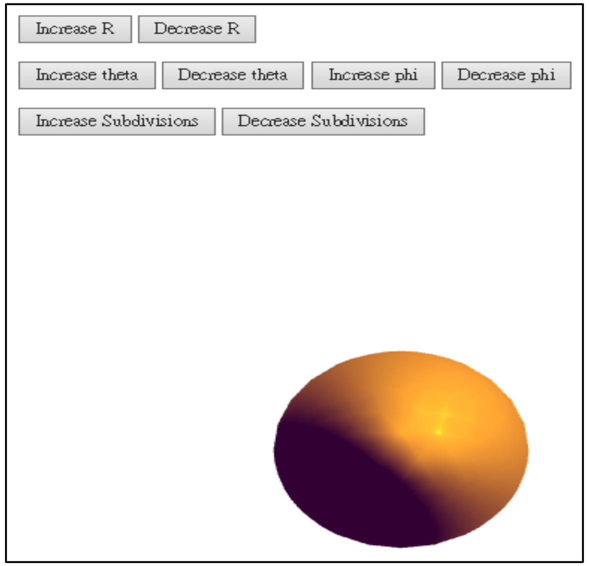
shadedSphere2.js (9/12)

```
var nBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, nBuffer);
gl.bufferData( gl.ARRAY_BUFFER, flatten(normalsArray), gl.STATIC_DRAW );

var vNormal = gl.getAttribLocation( program, "vNormal" );
gl.vertexAttribPointer( vNormal, 4, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vNormal);


var vBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
gl.bufferData(gl.ARRAY_BUFFER, flatten(pointsArray), gl.STATIC_DRAW);

var vPosition = gl.getAttribLocation( program, "vPosition");
gl.vertexAttribPointer(vPosition, 4, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);
```



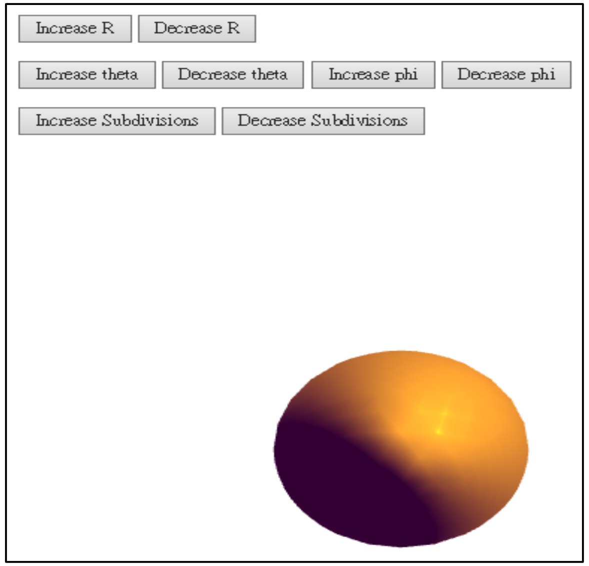
shadedSphere2.js (10/12)

```
modelViewMatrixLoc = gl.getUniformLocation( program, "modelViewMatrix" );
projectionMatrixLoc   = gl.getUniformLocation( program, "projectionMatrix" );

document.getElementById("Button0").onclick = function() {radius *= 2.0;};
document.getElementById("Button1").onclick = function() {radius *= 0.5;};
document.getElementById("Button2").onclick = function() {theta += dr;};
document.getElementById("Button3").onclick = function() {theta -= dr;};
document.getElementById("Button4").onclick = function() {phi += dr;};
document.getElementById("Button5").onclick = function() {phi -= dr;};

document.getElementById("Button6").onclick = function() {
    numTimesToSubdivide++;
    index = 0;
    pointsArray = [];
    normalsArray = [];
    init();
};
```



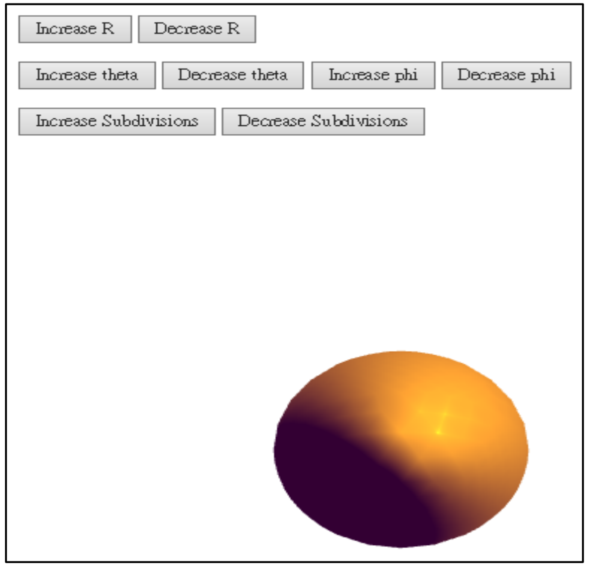
shadedSphere2.js (11/12)

```
document.getElementById("Button7").onclick = function(){
    if(numTimesToSubdivide) numTimesToSubdivide--;
    index = 0;
    pointsArray = [];
    normalsArray = [];
    init();
};

gl.uniform4fv( gl.getUniformLocation(program, "ambientProduct"), flatten(ambientProduct) );
gl.uniform4fv( gl.getUniformLocation(program, "diffuseProduct"),   flatten(diffuseProduct) );
gl.uniform4fv( gl.getUniformLocation(program, "specularProduct"),flatten(specularProduct) );
gl.uniform4fv( gl.getUniformLocation(program, "lightPosition"),      flatten(lightPosition) );
gl.uniform1f( gl.getUniformLocation(program,   "shininess"),          materialShininess );

render();
}   // end of window.onload
```



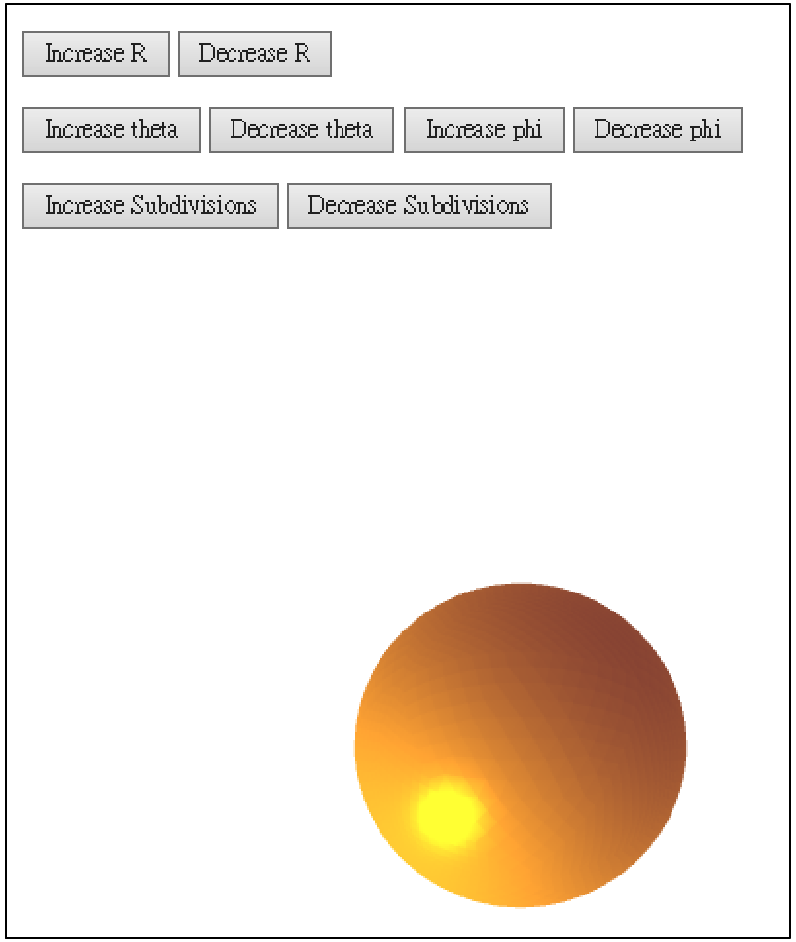
shadedSphere2.js (12/12)

```
function render() {

    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    eye = vec3(radius*Math.sin(theta)*Math.cos(phi),
                       radius*Math.sin(theta)*Math.sin(phi),
                       radius*Math.cos(theta));
    modelViewMatrix = lookAt(eye, at , up);
    projectionMatrix   = ortho(left, right, bottom, ytop, near, far);

    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(modelViewMatrix) );
    gl.uniformMatrix4fv(projectionMatrixLoc,   false, flatten(projectionMatrix) );

    for( var i=0; i<index; i+=3)
        gl.drawArrays( gl.TRIANGLES, i, 3 );

    window.requestAnimFrame(render);
}   // end of render()
```



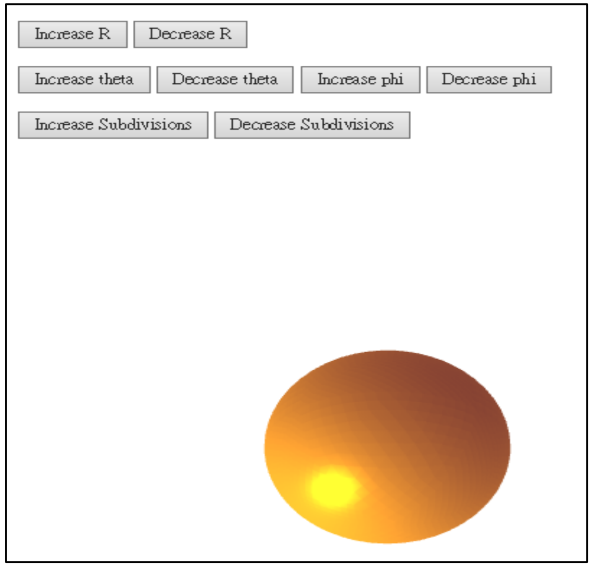
Sample Programs: shadedSphere3.html, shadedSphere3.js

Shaded sphere using vertex normals and per vertex shading



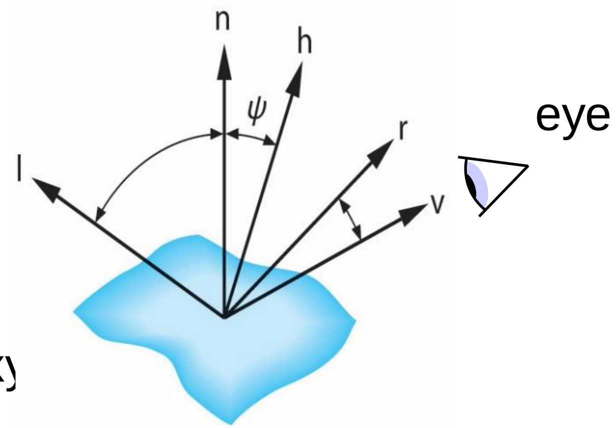
shadedSphere3.html (1/6)

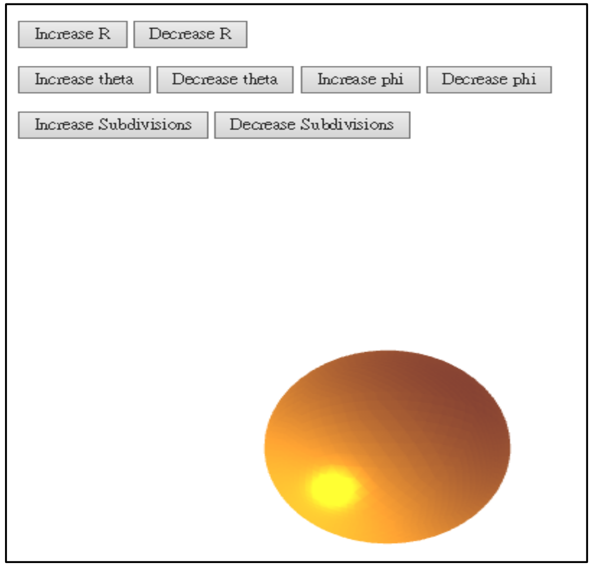
```
<!DOCTYPE html>
<html>

<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;
attribute vec4 vNormal;

varying vec4 fColor;

uniform vec4 ambientProduct, diffuseProduct, specularProduct;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;
uniform float  shininess;
```



shadedSphere3.html (2/6)

```
void
main()
{
    vec3 pos = -(modelViewMatrix * vPosition).xyz;
    vec3 light = lightPosition.xyz;
    vec3 L = normalize( light - pos );

    vec3 E = normalize( -pos );
    vec3 H = normalize( L + E );

    // Transform vertex normal into eye coordinates

    vec3 N = normalize( (modelViewMatrix*vNormal).xy
```



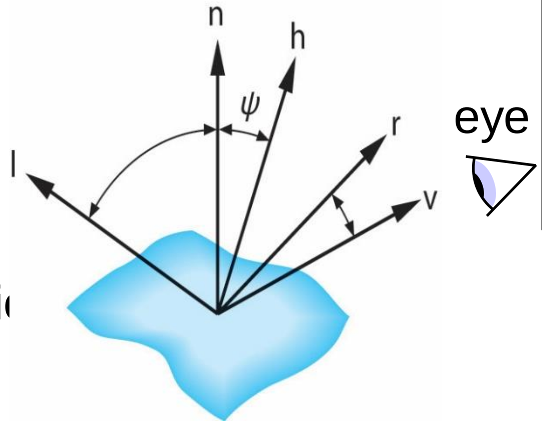


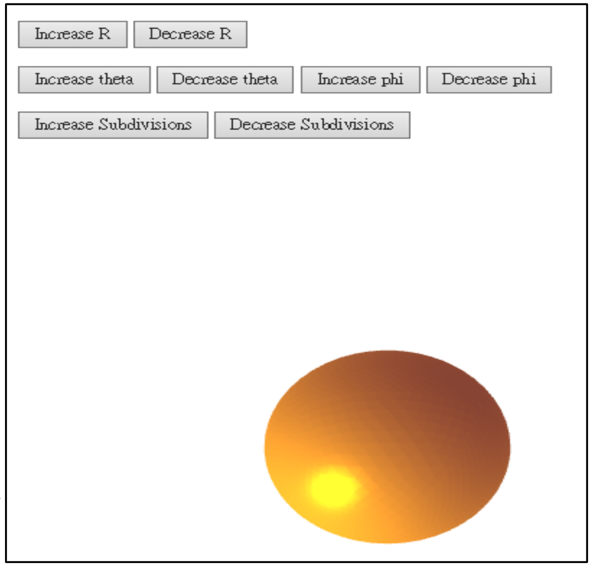
$I = k_d I_d \ (\mathbf{L} \cdot \mathbf{N}) \ + k_s I_s (\mathbf{N} \cdot \mathbf{H})^{\alpha} + k_a I_a$

where $\mathbf{H} = (\mathbf{L}+\mathbf{E})/2$ and α is shineness

shadedSphere3.html (3/6)

```
// Compute terms in the illumination equation
vec4 ambient = ambientProduct;

float Kd = max( dot(L, N), 0.0 );
vec4  diffuse = Kd*diffuseProduct;

float Ks = pow( max(dot(N, H), 0.0), shininess );
vec4  specular = Ks * specularProduct;

if( dot(L, N) < 0.0 ) { specular = vec4(0.0, 0.0, 0.0, 1.0); }

gl_Position = projectionMatrix * modelViewMatrix * vPositi

fColor = ambient + diffuse +specular;

fColor.a = 1.0;
}
</script>
```



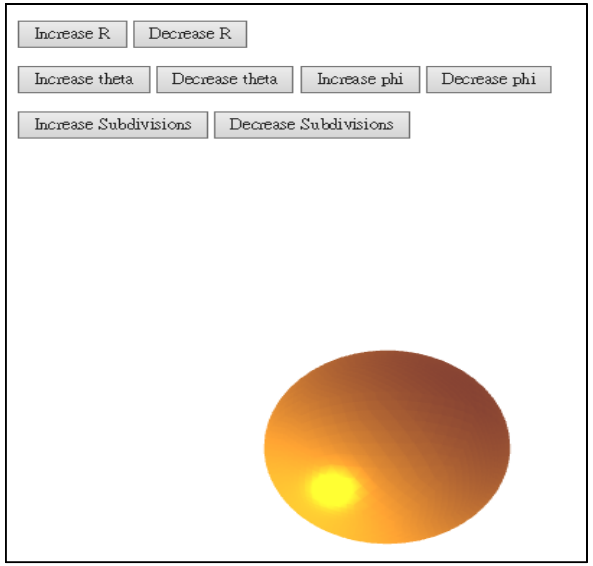


$$I = k_d I_d \; (\mathbf{L} \cdot \mathbf{N}) \; + k_s I_s (\mathbf{N} \cdot \mathbf{H})^{\alpha} + k_a I_a$$

where $\mathbf{H} = (\mathbf{L}+\mathbf{E})/2$ and α is shineness

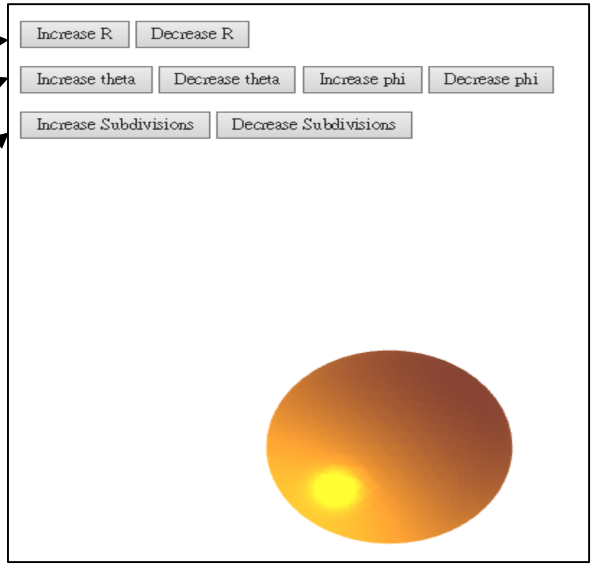
shadedSphere3.html (4/6)

```
<script id="fragment-shader" type="x-shader/x-fragment">

precision mediump float;

varying vec4 fColor;

void
main()
{

    gl_FragColor = fColor;

}
</script>
```



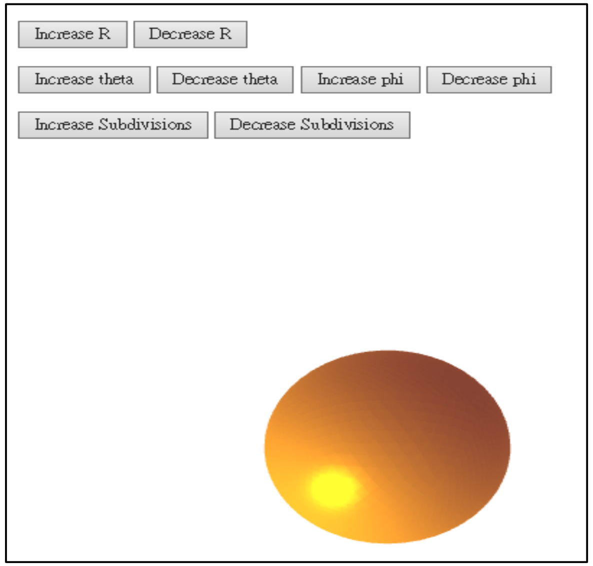
shadedSphere3.html (5/6)

```
<p> </p>
<button id = "Button0">Increase R</button>
<button id = "Button1">Decrease R</button>

<p> </p>
<button id = "Button2">Increase theta</button>
<button id = "Button3">Decrease theta</button>
<button id = "Button4">Increase phi</button>
<button id = "Button5">Decrease phi</button>
<p> </p>
<button id = "Button6">Increase Subdivisions</button>
<button id = "Button7">Decrease Subdivisions</button>

<p></p>
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="shadedSphere3.js"></script>
```



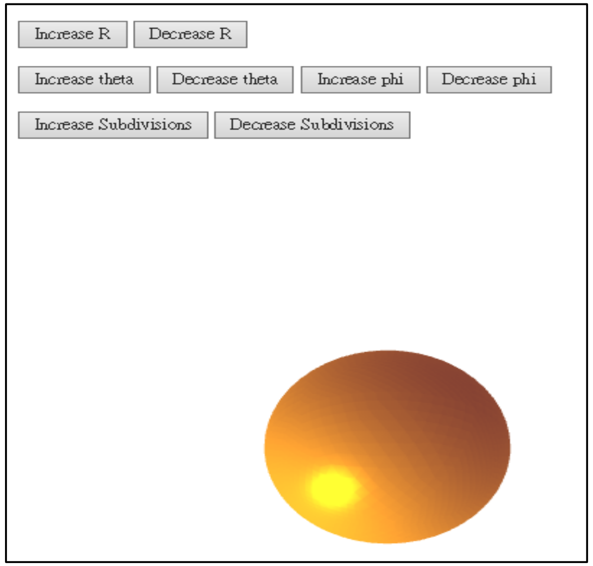
shadedSphere3.html (6/6)

```
<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



shadedSphere3.js (1/12)

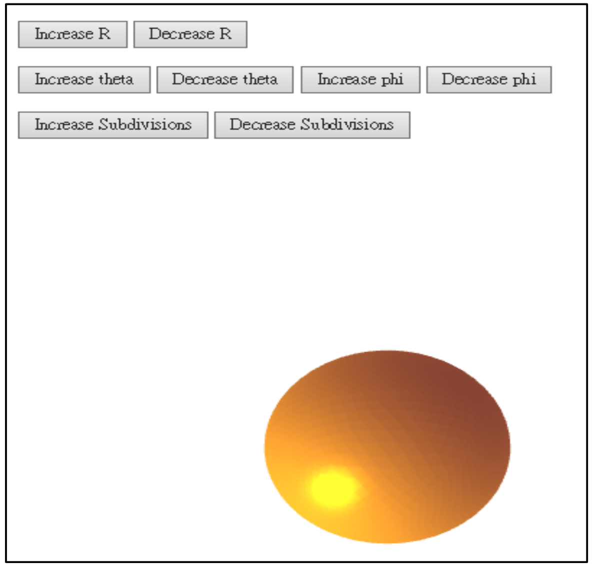
```
var canvas;
var gl;

var numTimesToSubdivide = 3;

var index = 0;

var pointsArray = [];
var normalsArray = [];

var near = -10;
var far    = 10;
var radius = 1.5;
var theta  = 0.0;
var phi     = 0.0;
var dr       = 5.0 * Math.PI/180.0;
```

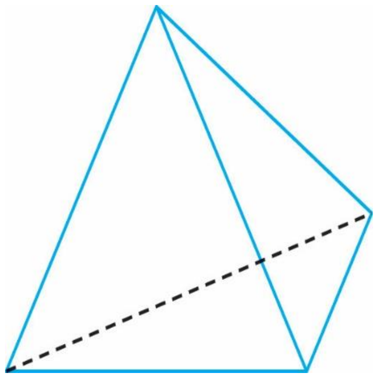


shadedSphere3.js (2/12)

```
var left = -3.0;
var right = 3.0;
var ytop =3.0;
var bottom = -3.0;

var va = vec4(0.0,          0.0,        -1.0,       1);
var vb = vec4( 0.0,         0.942809, 0.333333, 1);
var vc = vec4(-0.816497, -0.471405, 0.333333, 1);
var vd = vec4( 0.816497, -0.471405, 0.333333, 1);

var lightPosition  = vec4(1.0, 1.0, 1.0, 0.0 );
var lightAmbient  = vec4(0.2, 0.2, 0.2, 1.0 );
var lightDiffuse     = vec4(1.0, 1.0, 1.0, 1.0 );
var lightSpecular = vec4(1.0, 1.0, 1.0, 1.0 );
```

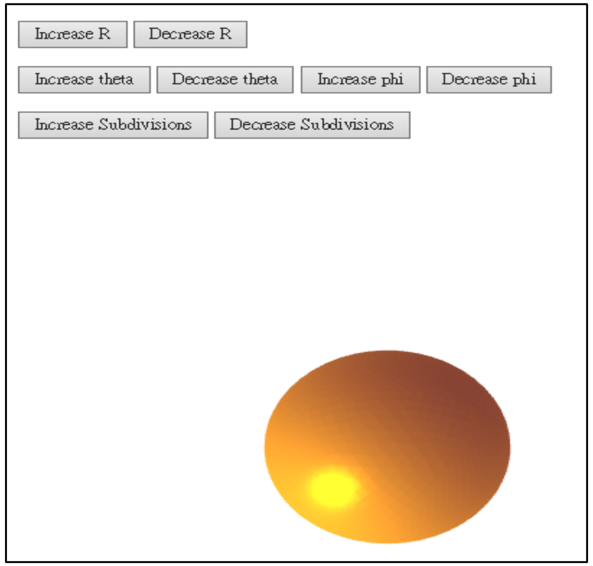
$$\left\{\begin{array}{l}(0.0,\ 0.0,\ -1.0)\\(0.0, 2\sqrt{2}/3,\ 1/3)\\(-\sqrt{6}/3, -\sqrt{2}/3,\ 1/3)\\(\sqrt{6}/3, -\sqrt{2}/3,\ 1/3)\end{array}\right.$$





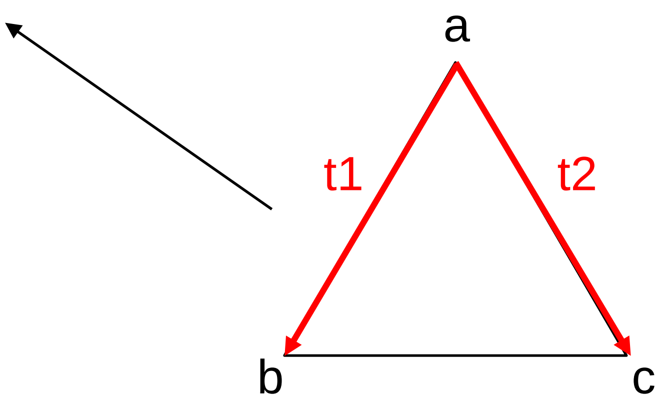
shadedSphere3.js (3/12)

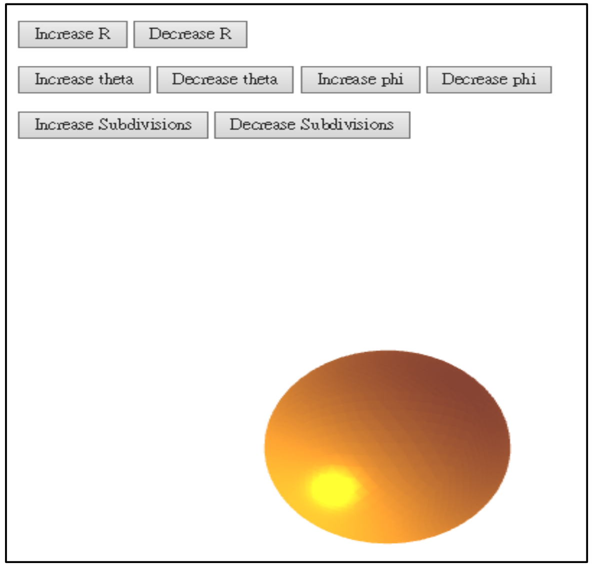
```
var materialAmbient  = vec4( 1.0, 0.0, 1.0, 1.0 );
var materialDiffuse    = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialSpecular = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialShininess = 100.0;

var ctm;
var ambientColor, diffuseColor, specularColor;

var modelViewMatrix, projectionMatrix;
var modelViewMatrixLoc, projectionMatrixLoc;
var eye;
var at = vec3(0.0, 0.0, 0.0);
var up = vec3(0.0, 1.0, 0.0);
```



shadedSphere3.js (4/12)

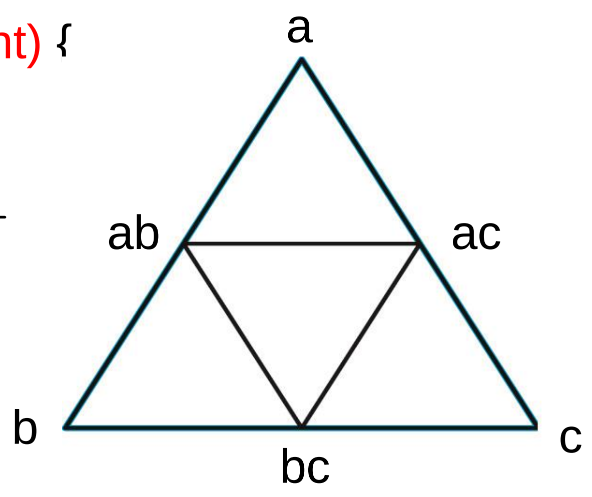
```
function triangle(a, b, c) {

    var t1 = subtract(b, a);
    var t2 = subtract(c, a);
    var normal = normalize(cross(t1, t2));
    normal = vec4(normal);

    normalsArray.push(normal);
    normalsArray.push(normal);
    normalsArray.push(normal);

    pointsArray.push(a);
    pointsArray.push(b);
    pointsArray.push(c);

    index += 3;
}
```

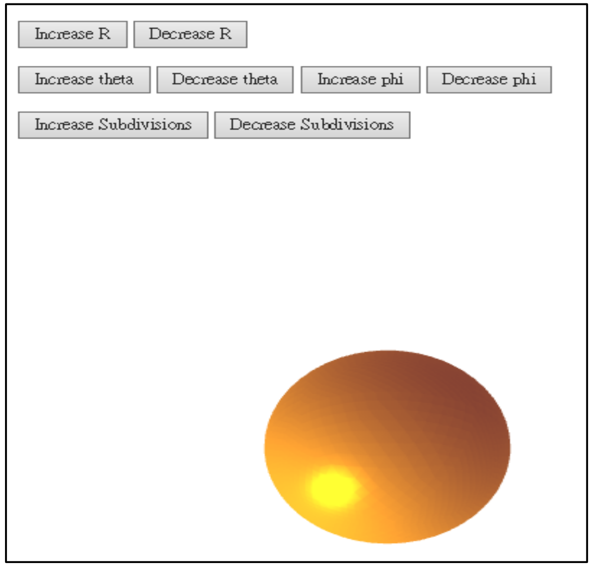




shadedSphere3.js (5/12)

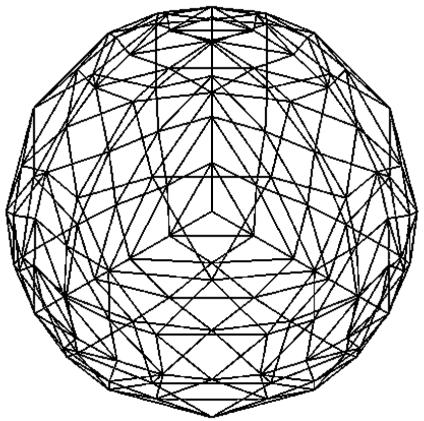
```
function divideTriangle(a, b, c, count) {
    if ( count > 0 ) {
        var ab = mix( a, b, 0.5);
        var ac = mix( a, c, 0.5);
        var bc = mix( b, c, 0.5);

        ab = normalize(ab, true);
        ac = normalize(ac, true);
        bc = normalize(bc, true);

        divideTriangle(  a, ab, ac, count - 1 );
        divideTriangle( ab,  b, bc, count - 1 );
        divideTriangle( bc,  c, ac, count - 1 );
        divideTriangle( ab,bc, ac, count - 1 );
    }
    else { triangle( a, b, c ); }
}
```

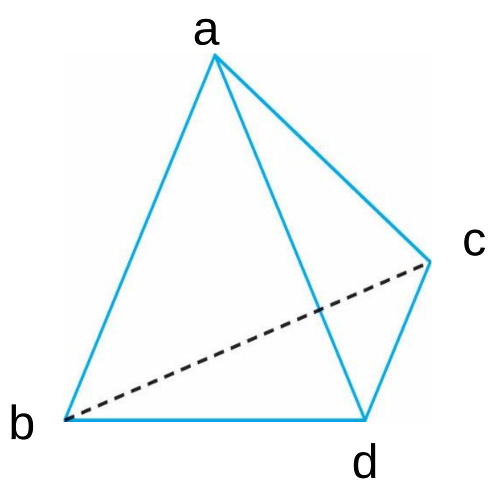


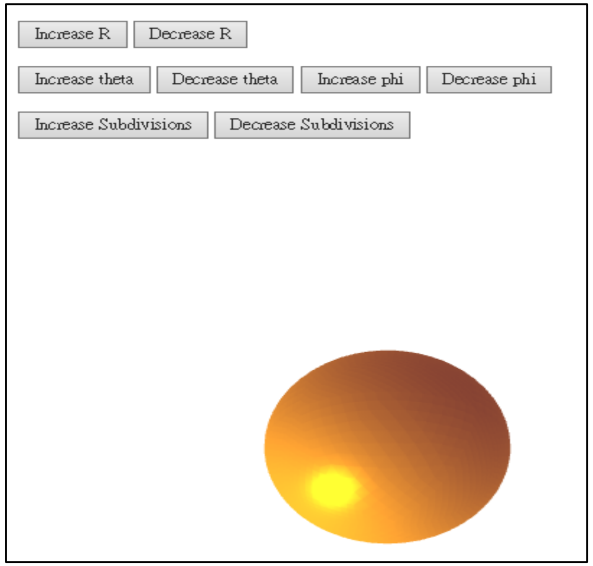


shadedSphere3.js (6/12)

```
function tetrahedron(a, b, c, d, n) {
    divideTriangle(a, b, c, n);
    divideTriangle(d, c, b, n);
    divideTriangle(a, d, b, n);
    divideTriangle(a, c, d, n);
}
```

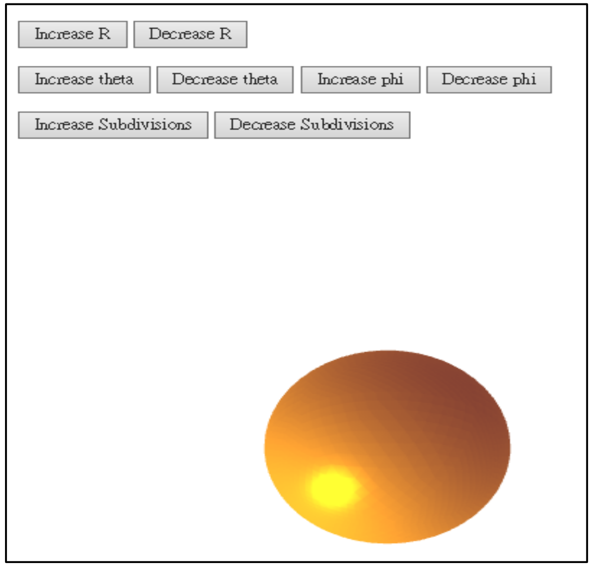






shadedSphere3.js (7/12)

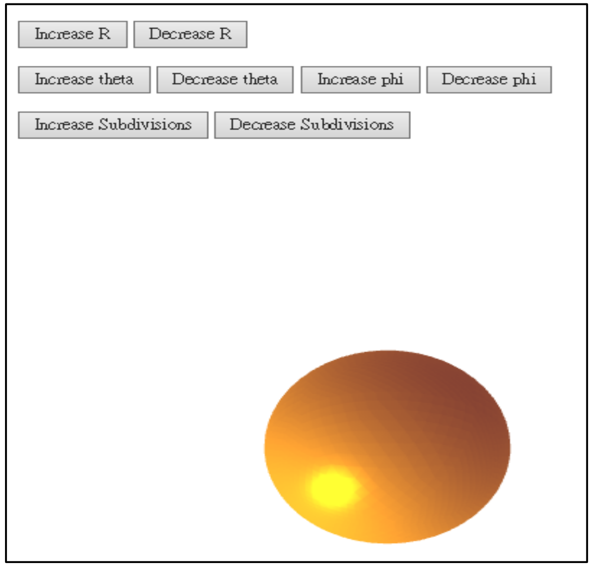
```
window.onload = function init() {

    canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );

    gl.enable(gl.DEPTH_TEST);
```



shadedSphere3.js (8/12)

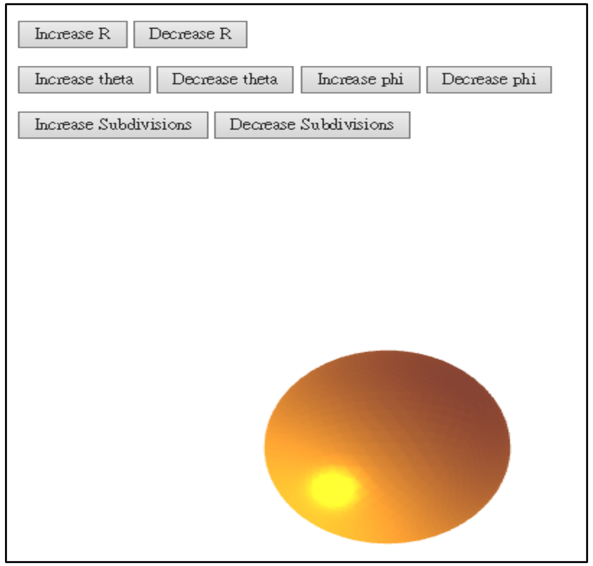
```
//
//  Load shaders and initialize attribute buffers
//
var program = initShaders( gl, "vertex-shader", "fragment-shader" );
gl.useProgram( program );

ambientProduct  = mult(lightAmbient, materialAmbient);
diffuseProduct     = mult(lightDiffuse, materialDiffuse);
specularProduct = mult(lightSpecular, materialSpecular);

tetrahedron(va, vb, vc, vd, numTimesToSubdivide);
```

$$I = k_d I_d \ (\mathbf{L} \cdot \mathbf{N}) + k_s I_s (\mathbf{N} \cdot \mathbf{H})^{\alpha} + k_a I_a$$



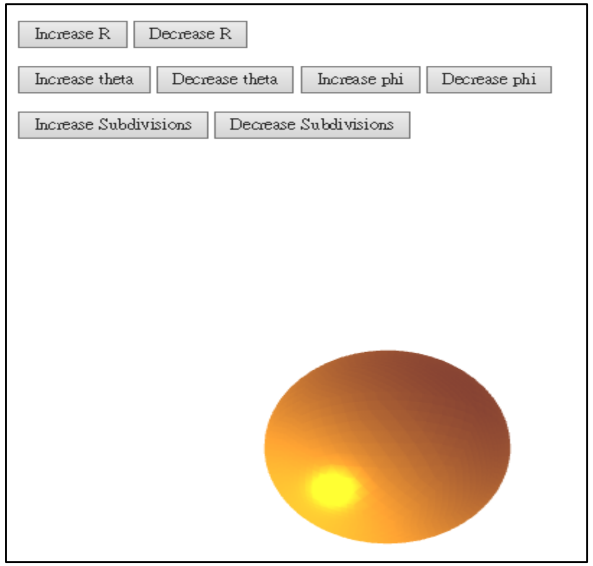
shadedSphere3.js (9/12)

```
var nBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, nBuffer);
gl.bufferData( gl.ARRAY_BUFFER, flatten(normalsArray), gl.STATIC_DRAW );

var vNormal = gl.getAttribLocation( program, "vNormal" );
gl.vertexAttribPointer( vNormal, 4, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vNormal);

var vBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
gl.bufferData(gl.ARRAY_BUFFER, flatten(pointsArray), gl.STATIC_DRAW);

var vPosition = gl.getAttribLocation( program, "vPosition");
gl.vertexAttribPointer(vPosition, 4, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);
```

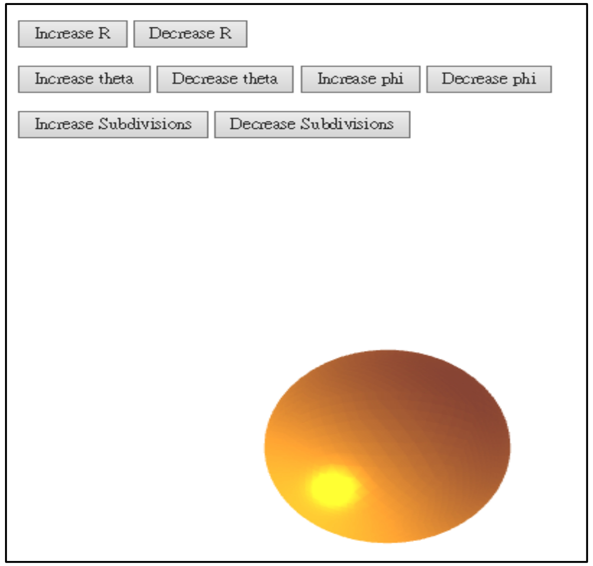


shadedSphere3.js (10/12)

```
modelViewMatrixLoc = gl.getUniformLocation( program, "modelViewMatrix" );
projectionMatrixLoc   = gl.getUniformLocation( program, "projectionMatrix" );

document.getElementById("Button0").onclick = function() {radius *= 2.0;};
document.getElementById("Button1").onclick = function() {radius *= 0.5;};
document.getElementById("Button2").onclick = function() {theta += dr;};
document.getElementById("Button3").onclick = function() {theta -= dr;};
document.getElementById("Button4").onclick = function() {phi += dr;};
document.getElementById("Button5").onclick = function() {phi -= dr;};

document.getElementById("Button6").onclick = function() {
    numTimesToSubdivide++;
    index = 0;
    pointsArray = [];
    normalsArray = [];
    init();
};
```



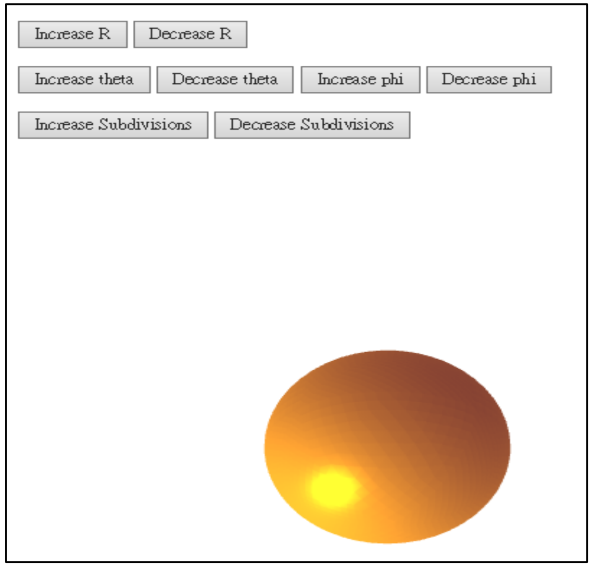
shadedSphere3.js (11/12)




```
document.getElementById("Button7").onclick = function(){
    if(numTimesToSubdivide) numTimesToSubdivide--;
    index = 0;
    pointsArray = [];
    normalsArray = [];
    init();
};

gl.uniform4fv( gl.getUniformLocation(program, "ambientProduct"), flatten(ambientProduct) );
gl.uniform4fv( gl.getUniformLocation(program, "diffuseProduct"),   flatten(diffuseProduct) );
gl.uniform4fv( gl.getUniformLocation(program, "specularProduct"),flatten(specularProduct) );
gl.uniform4fv( gl.getUniformLocation(program,  "lightPosition"),     flatten(lightPosition) );
gl.uniform1f(   gl.getUniformLocation(program,  "shininess"),         materialShininess );

render();
}   // end of window.onload
```

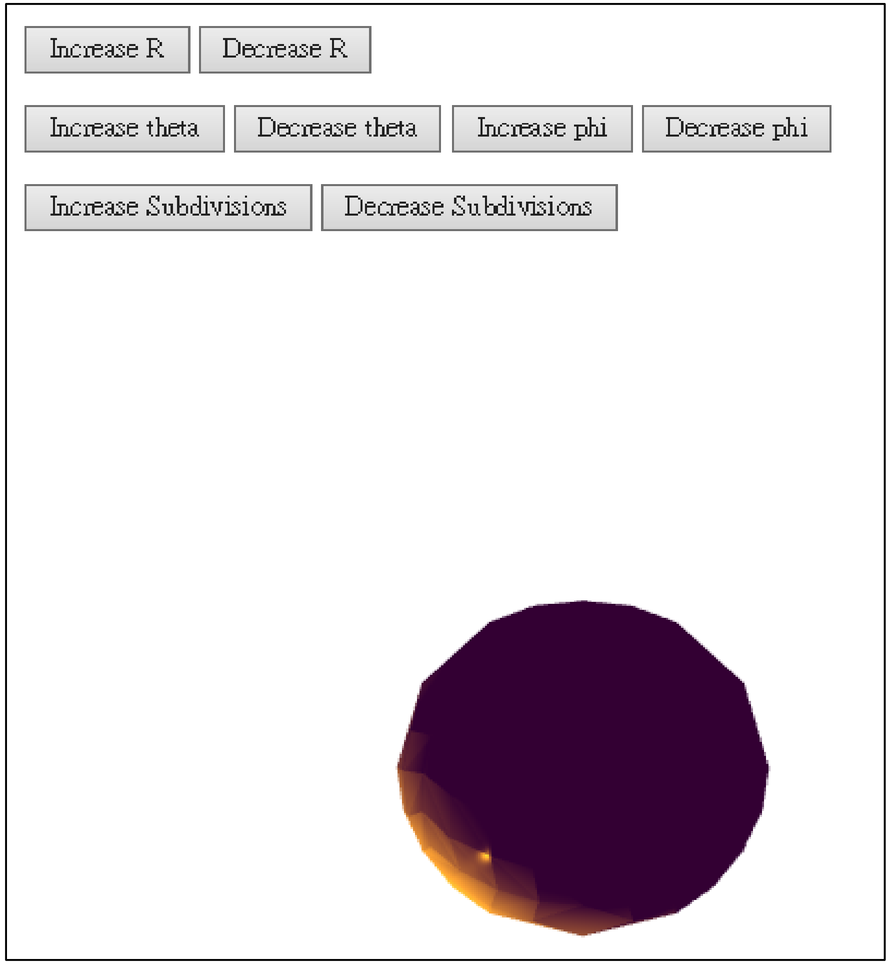
shadedSphere3.js (12/12)

```
function render() {

    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    eye = vec3(radius*Math.sin(theta)*Math.cos(phi),
                      radius*Math.sin(theta)*Math.sin(phi),
                      radius*Math.cos(theta));
    modelViewMatrix = lookAt(eye, at , up);
    projectionMatrix   = ortho(left, right, bottom, ytop, near, far);

    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(modelViewMatrix) );
    gl.uniformMatrix4fv(projectionMatrixLoc,   false, flatten(projectionMatrix) );

    for( var i=0; i<index; i+=3)
       gl.drawArrays( gl.TRIANGLES, i, 3 );

    window.requestAnimFrame(render);
}   // end of window.onload
```



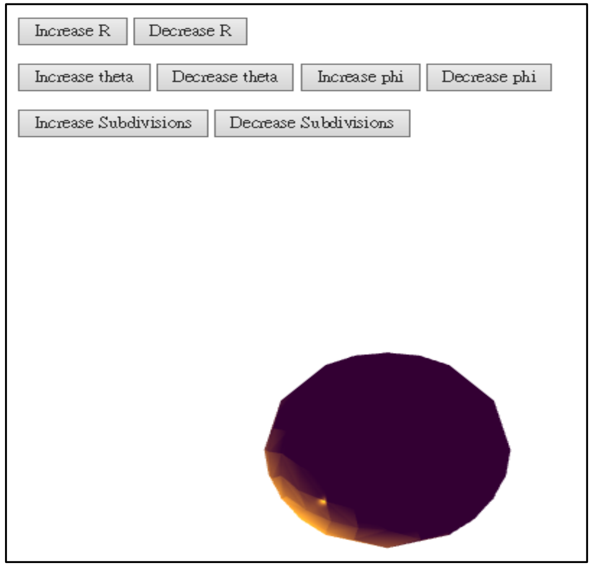
Sample Programs: shadedSphere4.html, shadedSphere4.js

Shaded sphere using vertex normals and per fragment shading



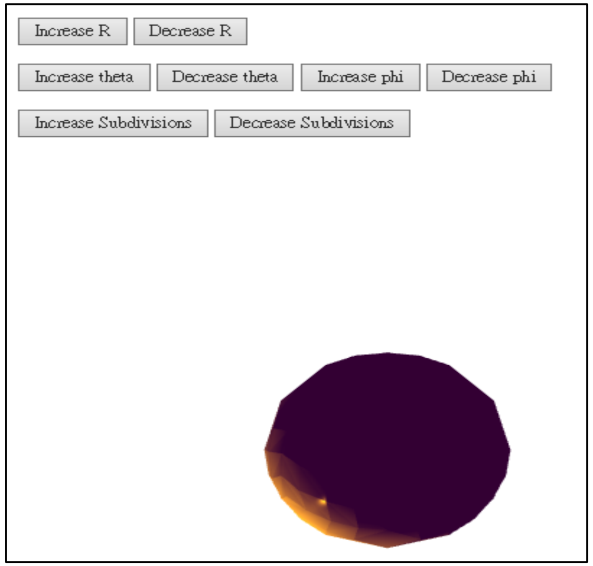
shadedSphere4.html (1/6)

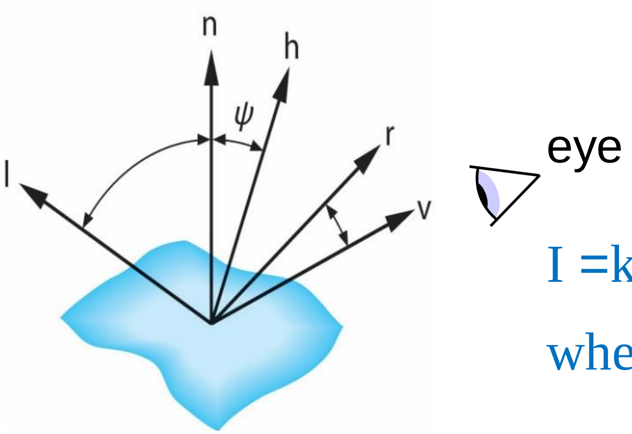
```
<!DOCTYPE html>
<html>

<script id="vertex-shader" type="x-shader/x-vertex">

attribute vec4 vPosition;
attribute vec4 vNormal;
varying vec3 N, L, E;
uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;
uniform vec4 lightPosition;
```



shadedSphere4.html (2/6)

```
void main()
{
    vec3 pos = -(modelViewMatrix * vPosition).xyz;
    vec3 light = lightPosition.xyz;
    L = normalize( light - pos );
    E =  -pos;
    N = normalize( (modelViewMatrix*vNormal).xyz);
    gl_Position = projectionMatrix * modelViewMatrix * vPosition;

}
</script>
```



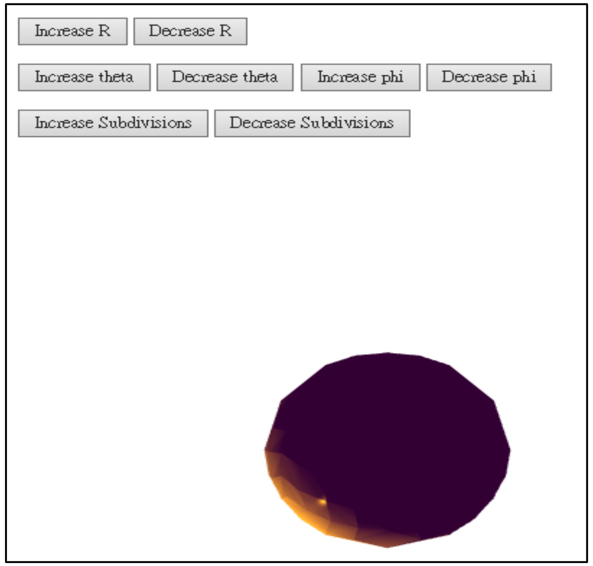


$I = k_d I_d \ (\mathbf{L} \cdot \mathbf{N}) \ + k_s I_s (\mathbf{N} \cdot \mathbf{H})^{\alpha} + k_a I_a$

where $\mathbf{H} = (\mathbf{L}+\mathbf{E})/2$ and α is shineness

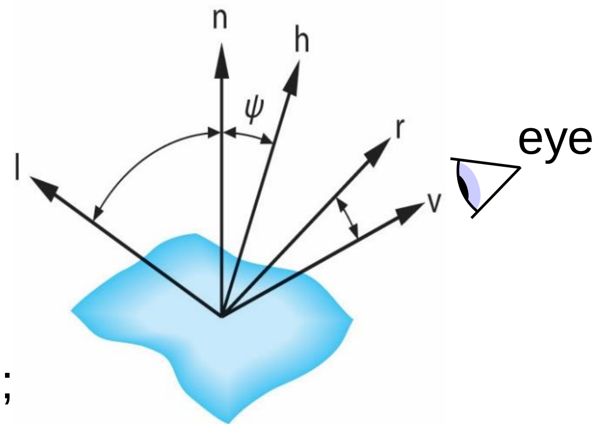
shadedSphere4.html (3/6)

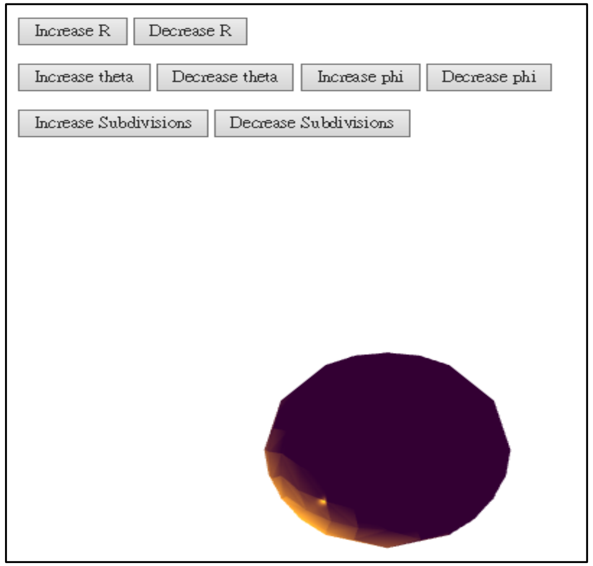
```
script id="fragment-shader" type="x-shader/x-fragment">

precision mediump float;

uniform vec4 ambientProduct;
uniform vec4 diffuseProduct;
uniform vec4 specularProduct;
uniform float shininess;
varying vec3 N, L, E;
```



shadedSphere4.html (4/6)

```
void main()
{
    vec4 fColor;

    vec3 H = normalize( L + E );
    vec4 ambient = ambientProduct;

    float Kd = max( dot(L, N), 0.0 );
    vec4  diffuse = Kd*diffuseProduct;

    float Ks = pow( max(dot(N, H), 0.0), shininess );
    vec4  specular = Ks * specularProduct;
    if( dot(L, N) < 0.0 ) specular = vec4(0.0, 0.0, 0.0, 1.0);

    fColor = ambient + diffuse +specular;
    fColor.a = 1.0;
    gl_FragColor = fColor;
}
</script>
```



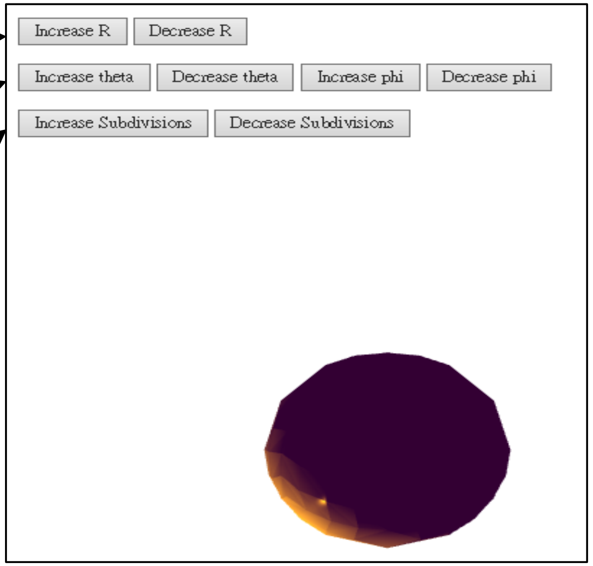


$$I = k_d I_d \ (\mathbf{L} \cdot \mathbf{N}) \ + k_s I_s (\mathbf{N} \cdot \mathbf{H})^{\alpha} + k_a I_a$$

where $\mathbf{H} = (\mathbf{L}+\mathbf{E})/2$ and α is shineness

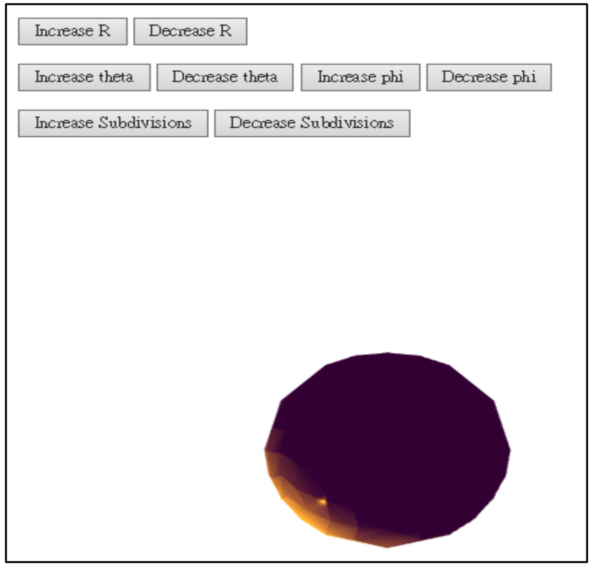
shadedSphere4.html (5/6)

```html
<p> </p>
<button id = "Button0">Increase R</button>
<button id = "Button1">Decrease R</button>

<p> </p>
<button id = "Button2">Increase theta</button>
<button id = "Button3">Decrease theta</button>
<button id = "Button4">Increase phi</button>
<button id = "Button5">Decrease phi</button>
<p> </p>
<button id = "Button6">Increase Subdivisions</button>
<button id = "Button7">Decrease Subdivisions</button>
<p></p>
<script type="text/javascript" src="../Common/webgl-utils.js"></script>
<script type="text/javascript" src="../Common/initShaders.js"></script>
<script type="text/javascript" src="../Common/MV.js"></script>
<script type="text/javascript" src="shadedSphere4.js"></script>
```



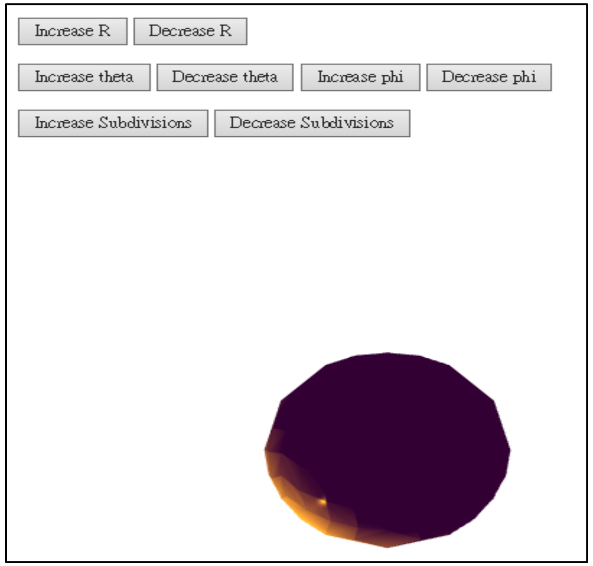
shadedSphere4.html (6/6)

```
<body>
<canvas id="gl-canvas" width="512" height="512">
Oops ... your browser doesn't support the HTML5 canvas element
</canvas>
</body>
</html>
```



shadedSphere4.js (1/12)

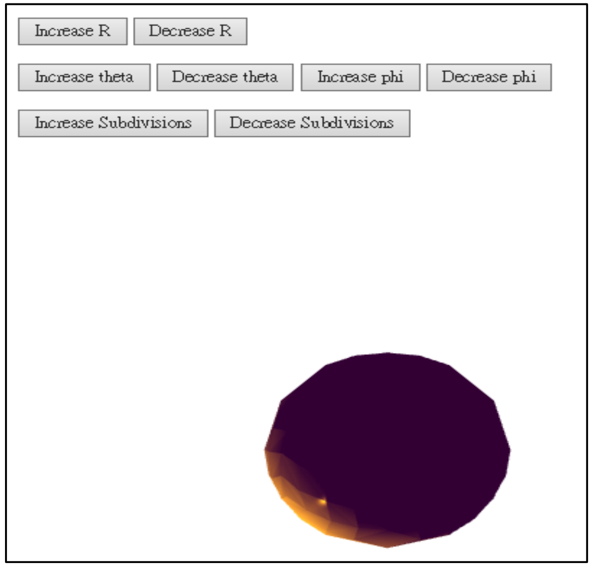
```
var canvas;
var gl;

var numTimesToSubdivide = 3;

var index = 0;

var pointsArray = [];
var normalsArray = [];

var near = -10;
var far = 10;
var radius = 1.5;
var theta  = 0.0;
var phi    = 0.0;
var dr = 5.0 * Math.PI/180.0;
```

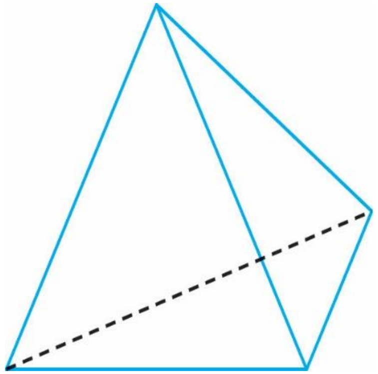


shadedSphere4.js (2/12)

```
var left = -3.0;
var right = 3.0;
var ytop =3.0;
var bottom = -3.0;

var va = vec4( 0.0,          0.0,       -1.0,      1);
var vb = vec4( 0.0,          0.942809, 0.333333, 1);
var vc = vec4(-0.816497, -0.471405, 0.333333, 1);
var vd = vec4( 0.816497, -0.471405, 0.333333, 1);
```

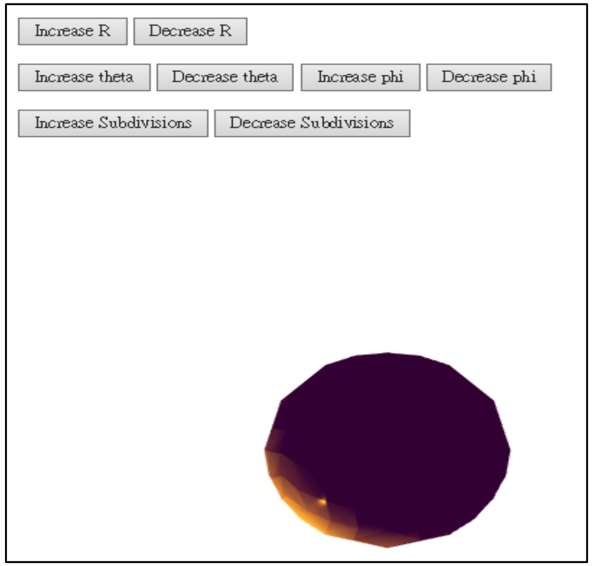
$$\left\{\begin{array}{l} (0.0,\ 0.0,\ \text{-}1.0) \\ (0.0, 2\sqrt{2}/3,\ 1/3) \\ (-\sqrt{6}/3, -\sqrt{2}/3,\ 1/3) \\ (\sqrt{6}/3, -\sqrt{2}/3,\ 1/3) \end{array}\right.$$





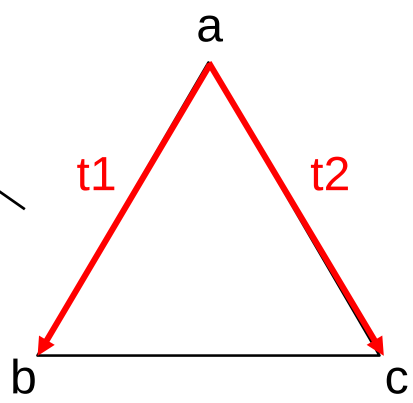
shadedSphere4.js (3/12)

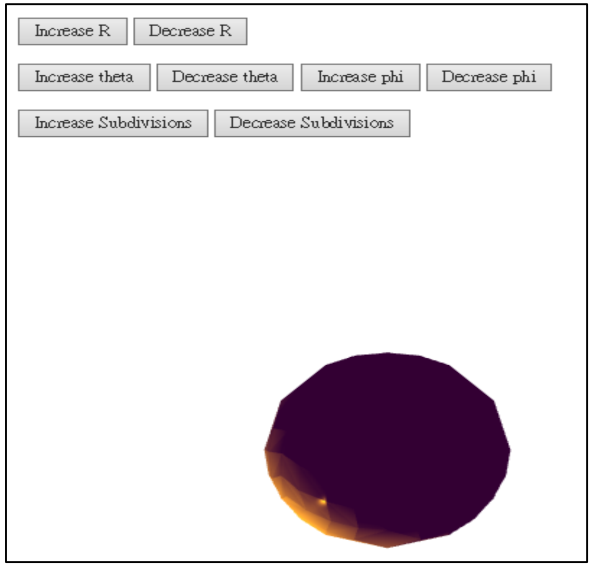
```
var lightPosition  = vec4( 1.0, 1.0, 1.0, 0.0 );
var lightAmbient  = vec4( 0.2, 0.2, 0.2, 1.0 );
var lightDiffuse    = vec4( 1.0, 1.0, 1.0, 1.0 );
var lightSpecular = vec4( 1.0, 1.0, 1.0, 1.0 );

var materialAmbient   = vec4( 1.0, 0.0, 1.0, 1.0 );
var materialDiffuse     = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialSpecular  = vec4( 1.0, 0.8, 0.0, 1.0 );
var materialShininess = 100.0;

var ctm;
var ambientColor, diffuseColor, specularColor;

var modelViewMatrix, projectionMatrix;
var modelViewMatrixLoc, projectionMatrixLoc;
var eye;
var at  = vec3(0.0, 0.0, 0.0);
var up = vec3(0.0, 1.0, 0.0);
```



shadedSphere4.js (4/12)

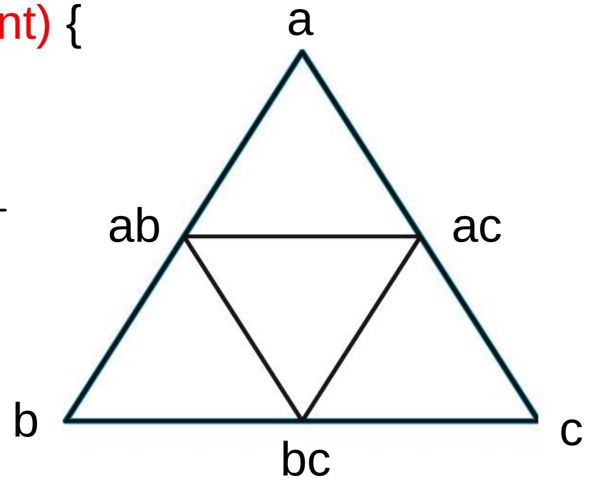
```
function triangle(a, b, c) {

     var t1 = subtract(b, a);
     var t2 = subtract(c, a);
     var normal = normalize(cross(t1, t2));
     normal = vec4(normal);

     normalsArray.push(normal);
     normalsArray.push(normal);
     normalsArray.push(normal);


     pointsArray.push(a);
     pointsArray.push(b);
     pointsArray.push(c);

     index += 3;
}
```

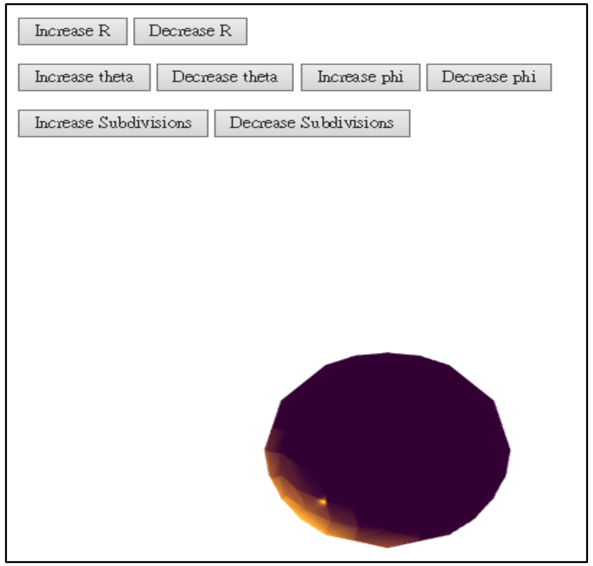




shadedSphere4.js (5/12)

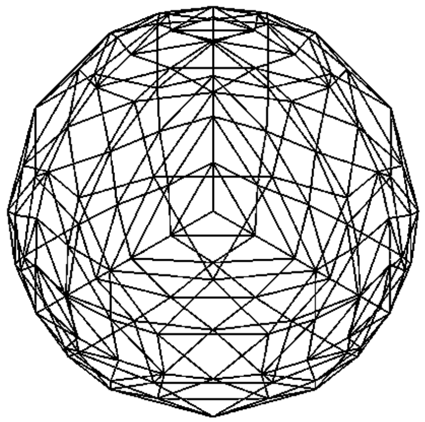
```
function divideTriangle(a, b, c, count) {
    if ( count > 0 ) {
        var ab = mix( a, b, 0.5);
        var ac = mix( a, c, 0.5);
        var bc = mix( b, c, 0.5);

        ab = normalize(ab, true);
        ac = normalize(ac, true);
        bc = normalize(bc, true);

        divideTriangle( a,   ab, ac, count - 1 );
        divideTriangle( ab,   b, bc, count - 1 );
        divideTriangle( bc,   c, ac, count - 1 );
        divideTriangle( ab, bc, ac, count - 1 );
    }
    else {  triangle( a, b, c ); }
}
```

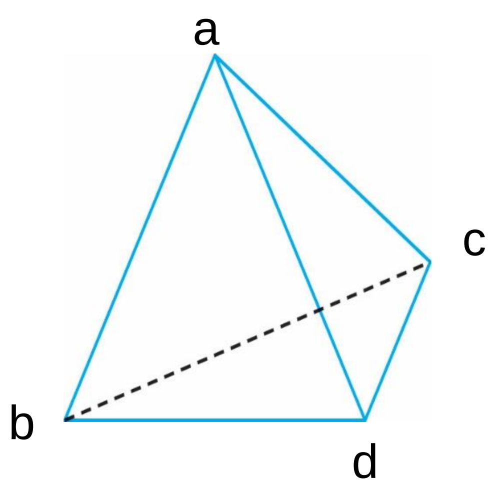


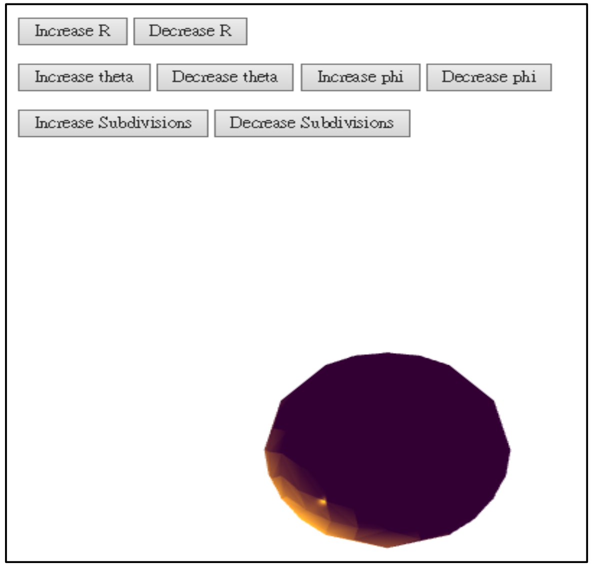


shadedSphere4.js (6/12)

```
function tetrahedron(a, b, c, d, n) {
    divideTriangle(a, b, c, n);
    divideTriangle(d, c, b, n);
    divideTriangle(a, d, b, n);
    divideTriangle(a, c, d, n);
}
```

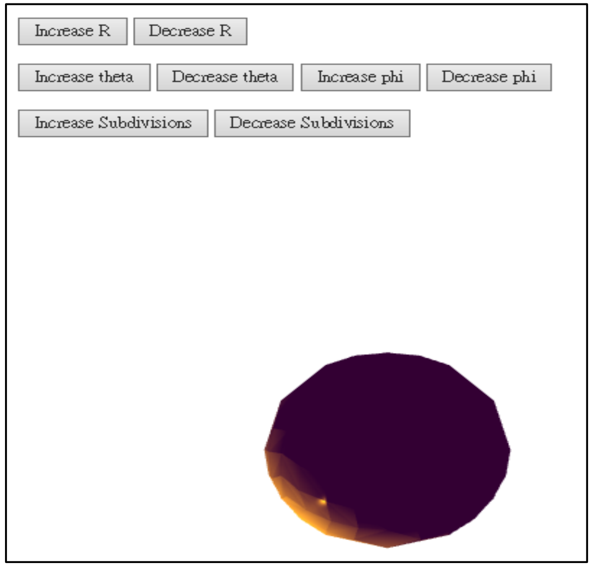






shadedSphere4.js (7/12)

```
window.onload = function init() {

    canvas = document.getElementById( "gl-canvas" );

    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) { alert( "WebGL isn't available" ); }

    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 1.0, 1.0, 1.0, 1.0 );

    gl.enable(gl.DEPTH_TEST);
```

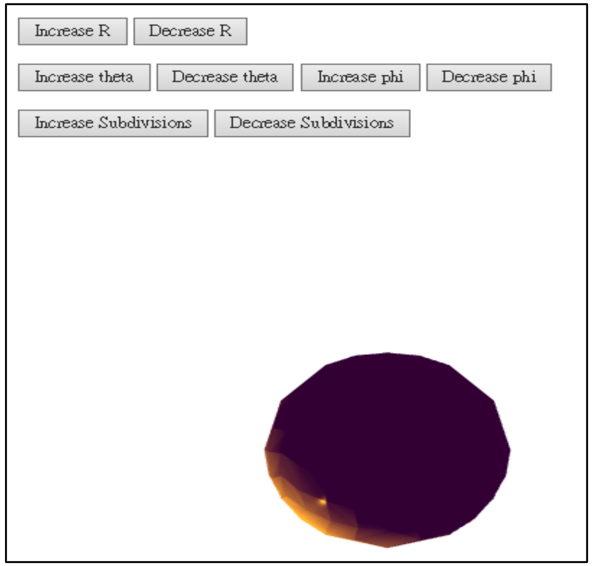


shadedSphere4.js (8/12)

```
//
//  Load shaders and initialize attribute buffers
//
var program = initShaders( gl, "vertex-shader", "fragment-shader" );
gl.useProgram( program );
```

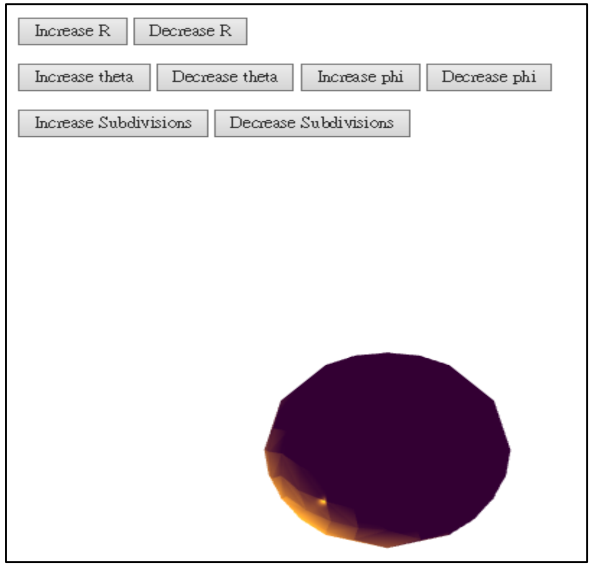
$$I = k_d I_d (\mathbf{L} \cdot \mathbf{N}) + k_s I_s (\mathbf{N} \cdot \mathbf{H})^{\alpha} + k_a I_a$$

```
ambientProduct  = mult(lightAmbient, materialAmbient);
diffuseProduct    = mult(lightDiffuse, materialDiffuse);
specularProduct = mult(lightSpecular, materialSpecular);

tetrahedron(va, vb, vc, vd, numTimesToSubdivide);
```



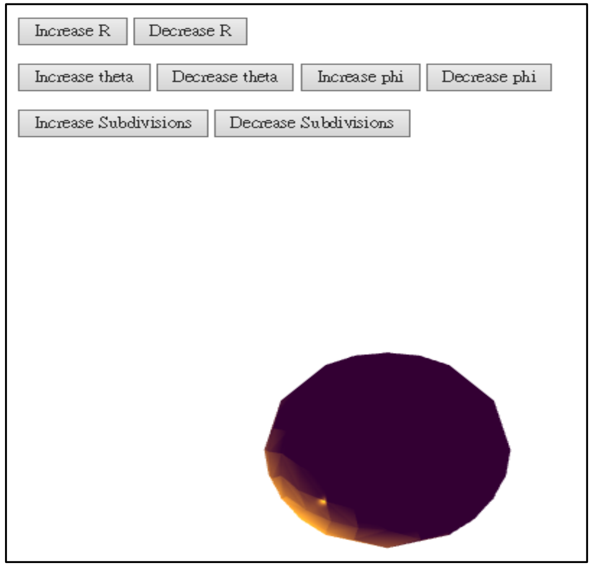
shadedSphere4.js (9/12)

```
var nBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, nBuffer);
gl.bufferData( gl.ARRAY_BUFFER, flatten(normalsArray), gl.STATIC_DRAW );

var vNormal = gl.getAttribLocation( program, "vNormal" );
gl.vertexAttribPointer( vNormal, 4, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vNormal);

var vBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
gl.bufferData(gl.ARRAY_BUFFER, flatten(pointsArray), gl.STATIC_DRAW);

var vPosition = gl.getAttribLocation( program, "vPosition");
gl.vertexAttribPointer(vPosition, 4, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);
```

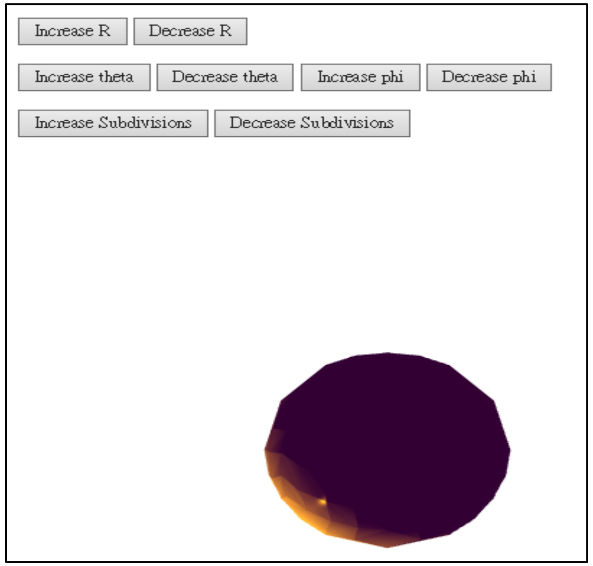


shadedSphere4.js (10/12)

```
modelViewMatrixLoc = gl.getUniformLocation( program, "modelViewMatrix" );
projectionMatrixLoc   = gl.getUniformLocation( program, "projectionMatrix" );

document.getElementById("Button0").onclick = function() {radius *= 2.0;};
document.getElementById("Button1").onclick = function() {radius *= 0.5;};
document.getElementById("Button2").onclick = function() {theta += dr;};
document.getElementById("Button3").onclick = function() {theta -= dr;};
document.getElementById("Button4").onclick = function() {phi += dr;};
document.getElementById("Button5").onclick = function() {phi -= dr;};

document.getElementById("Button6").onclick = function() {
    numTimesToSubdivide++;
    index = 0;
    pointsArray = [];
    normalsArray = [];
    init();
};
```



shadedSphere4.js (11/12)




```
document.getElementById("Button7").onclick = function() {
        if(numTimesToSubdivide) numTimesToSubdivide--;
        index = 0;
        pointsArray = [];
        normalsArray = [];
        init();
    };

    gl.uniform4fv( gl.getUniformLocation(program, "ambientProduct"), flatten(ambientProduct) );
    gl.uniform4fv( gl.getUniformLocation(program, "diffuseProduct"),   flatten(diffuseProduct) );
    gl.uniform4fv( gl.getUniformLocation(program, "specularProduct"),flatten(specularProduct) );
    gl.uniform4fv( gl.getUniformLocation(program, "lightPosition"),       flatten(lightPosition) );
    gl.uniform1f(   gl.getUniformLocation(program,  "shininess"),          materialShininess );

    render();
}   // end of window.onload
```

shadedSphere4.js (12/12)

```
function render() {

    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    eye = vec3(radius*Math.sin(theta)*Math.cos(phi),
        radius*Math.sin(theta)*Math.sin(phi), radius*Math.cos(theta));

    modelViewMatrix = lookAt(eye, at , up);
    projectionMatrix   = ortho(left, right, bottom, ytop, near, far);

    gl.uniformMatrix4fv(modelViewMatrixLoc, false, flatten(modelViewMatrix) );
    gl.uniformMatrix4fv(projectionMatrixLoc,   false, flatten(projectionMatrix) );

    for( var i=0; i<index; i+=3)
        gl.drawArrays( gl.TRIANGLES, i, 3 );

    window.requestAnimFrame(render);
}   // end of render()
```