Computer Graphics and Programming

Lecture 12

GLSL

by extending Ray Tracing

GLSL: OpenGL Shading Language

Jeong-Yean Yang
2020/12/8

1 GLSL by Ray Marching

OpenGL Shading Library (GLSL)

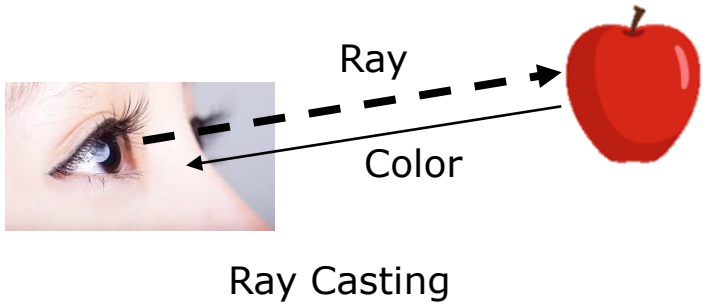
- We have learned OpenGL and Ray Tracing
  - How to Add Ray Casting or Semi Ray Tracing on OpenGL's environment?
  - OpenGL is based on Polygon-based Rendering

- Ray Tracing determines Pixel's Color by calculating color with respect to
  - Normal vector of surfaces
  - Light position, reflection, and refraction vector

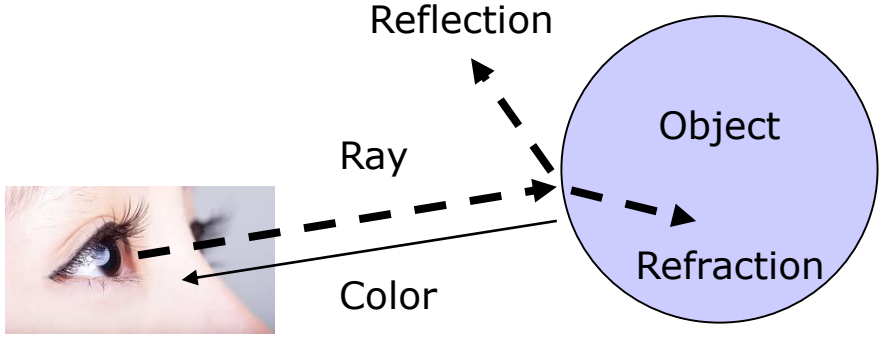
- OpenGL permits **One pixel's color** by GLSL programming

Computational Burden in Ray Tracing

- What is the most Painful works in Ray Tracing?



Ray Casting



Ray Tracing

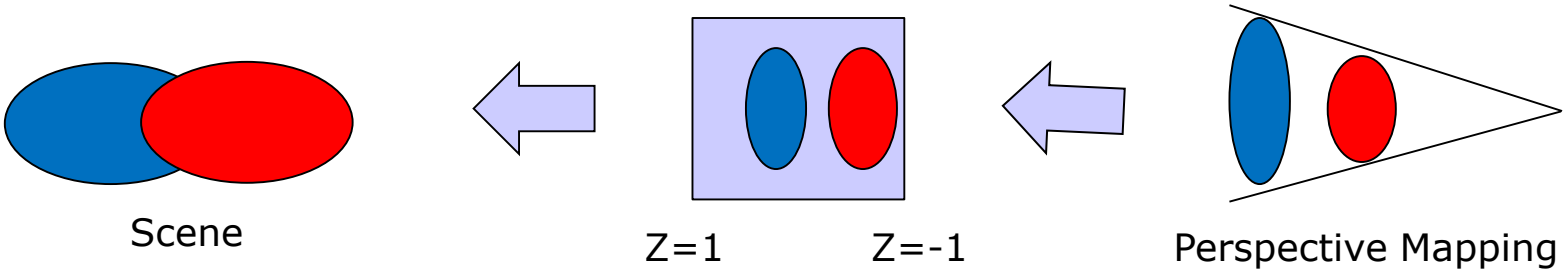
- Finding Intersection point is complex and slow.

- Sampling becomes the Approximation Technique for Intersection by fast computation.

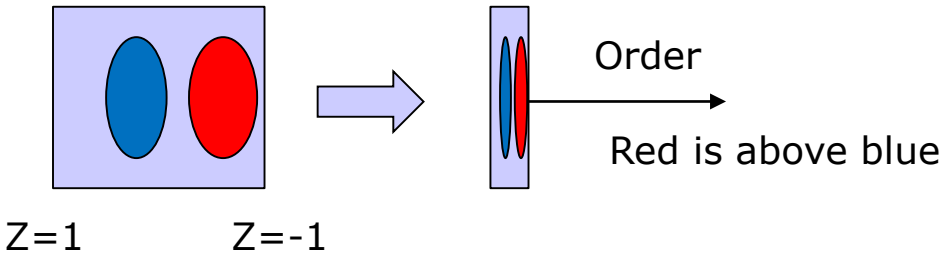
  → Ray Marching technique is used for OpenGL rendering

Ray Marching for Volumetric Rendering

- OpenGL uses Z buffer for Volume Rendering

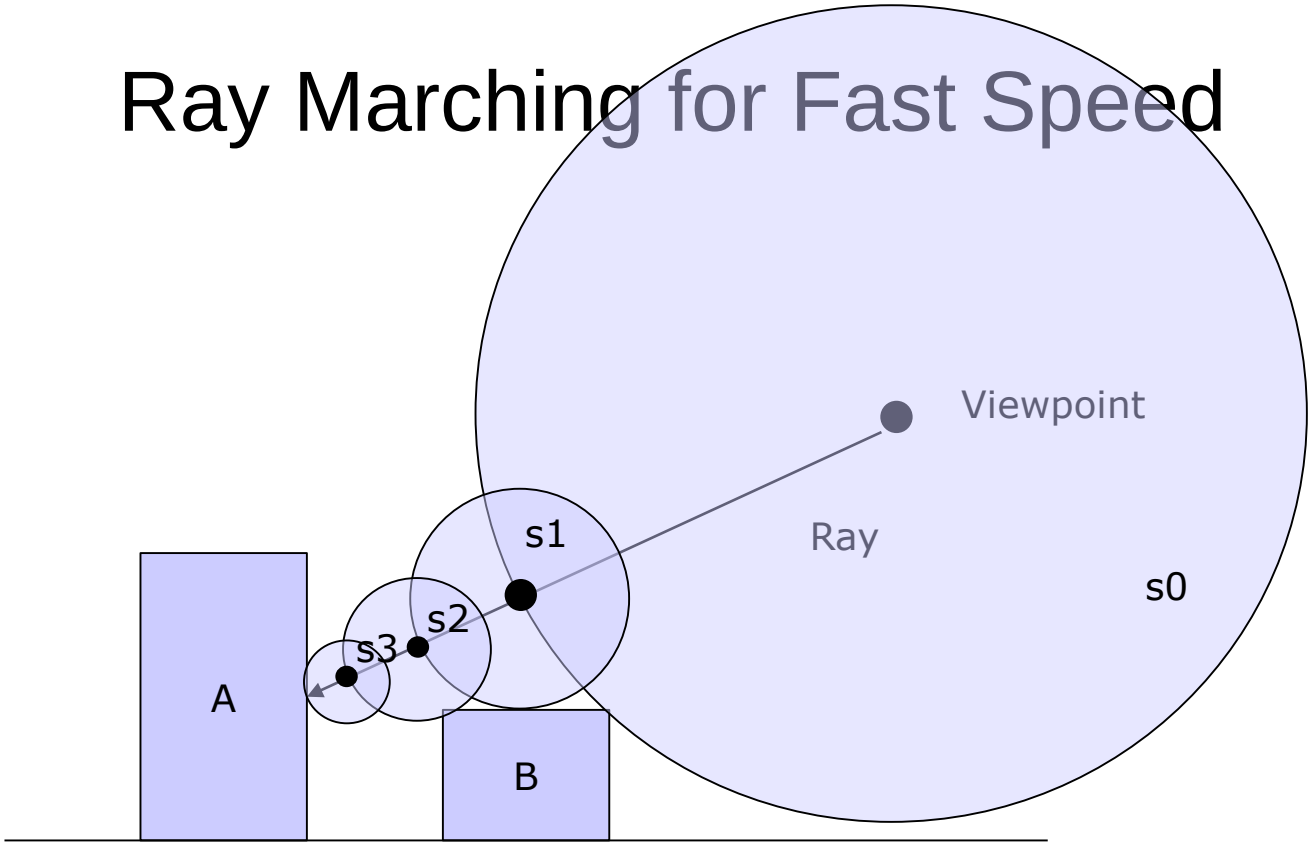


- Z buffering projects all data onto one scene image



- Rendering requires volumetric operation

Ray Marching for Fast Speed



- While doing volumetric rendering,
  - first sphere s0 meets object B and fills the volume
  - Second sphere s1 and s2 meets object B and fill the volume
  - Final sphere, s3 meets object A and Calculate Intersection Point
  - It is Faster than Ray tracing

2 GLSL Structure

How to OpenGL calculate One Pixel Color? → Vertex shader and Fragment Shader

Ref: uGL-39-GLSL-Basic

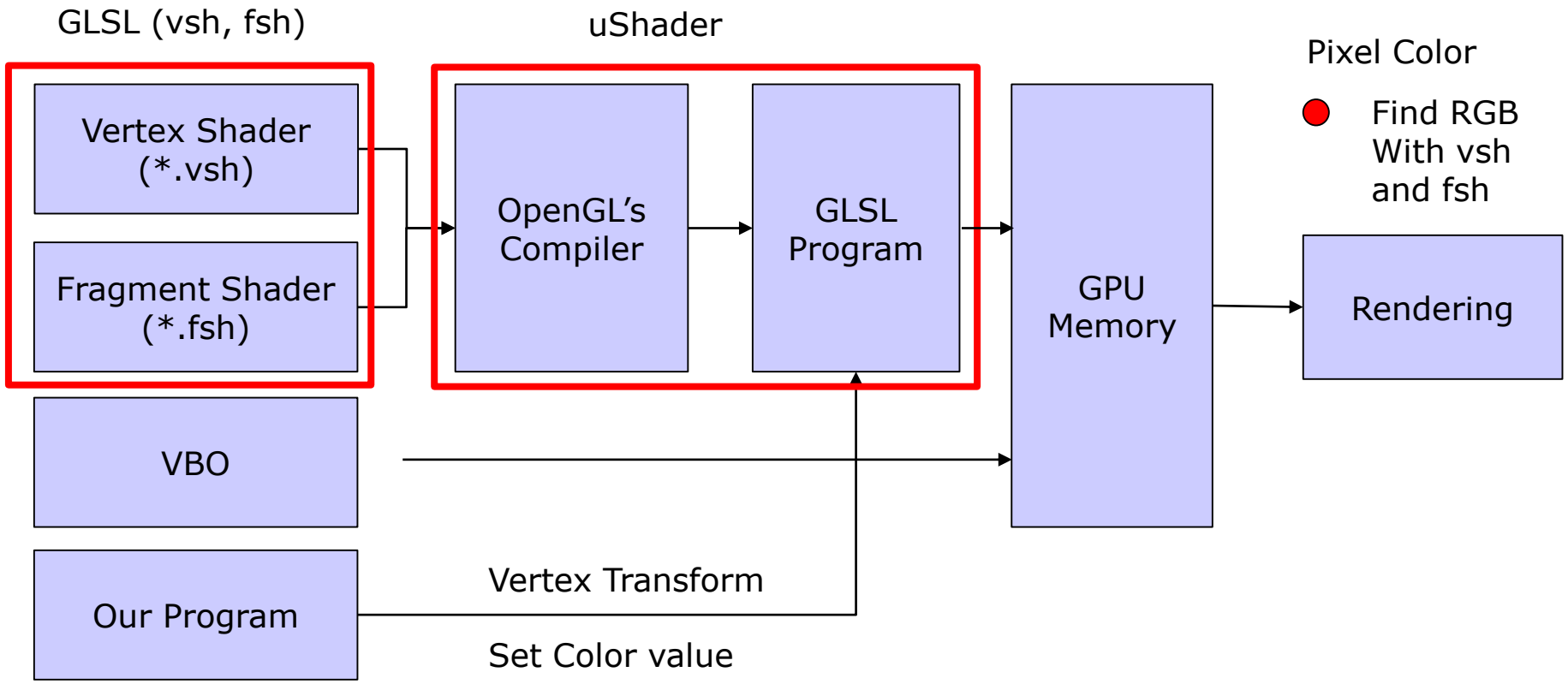
PhongTex.vsh

```
void main()
{
    gl_Position     = screen*model*vec4(vertices,1);

    eyePosition     = model*vec4(vertices,1);
    normal          = model*vec4(normals,0);

    diffuseVarying  = diffuse;
    ambientVarying  = ambient;
    specularVarying = specular;

    texcoordout = texcoord;
}
```

PhongTex.fsh

```
void main()
{
    vec3 P = vec3(eyePosition);
    vec3 N = normalize(normal.xyz);
    vec3 L = normalize(vec3(0,0,0) - P);

    float lamb      = max( dot(L,N),0.0);
    float specular  =0.0;

    if (lamb>0.0)
    {
        vec3 reflectDir = reflect(-L,N);
        vec3 viewDir    = normalize(-P);

        float specAngle = max(dot(reflectDir, viewDir), 0.0);
        specular = pow(specAngle, 10.0);

        specular *= lamb;
    }

    vec4 c    = texture2D(tex, texcoordout.st)*0.6+vec4(ambientVarying+diffuseVarying
    gl_FragColor = c;
}
```


GLSL Architecture



- Script(vsh and fsh) is compiled and uploaded by uShader class

App → VSH → FSH

- Our App transfers VBO handle to vsh
  - Vertices, normal, and textures.
  - Ambient, diffuse, and specular is given to vsh
- VSH file : **do transform** of vertices and normal.
  - Gl's result = P*H*vertices
  - Given color is not used here → pass colors into FSH
- FSH file: **do calculation of colors**
  - VSH provides color and geometry information.
  - FSH calculate RGB.

GLSL Basic Variable Types

```
// arguments.
attribute vec3 vertices;
attribute vec3 normals;
attribute vec2 texcoord;
```

```
// for fsh
varying vec4 diffuseVarying;
varying vec4 normal;
```

```
uniform mat4 screen;
uniform mat4 model;
```

1. attribute

   Connected with VBO

2. varying

   Variables in vsh is transferred to variables in fsh

3. uniform

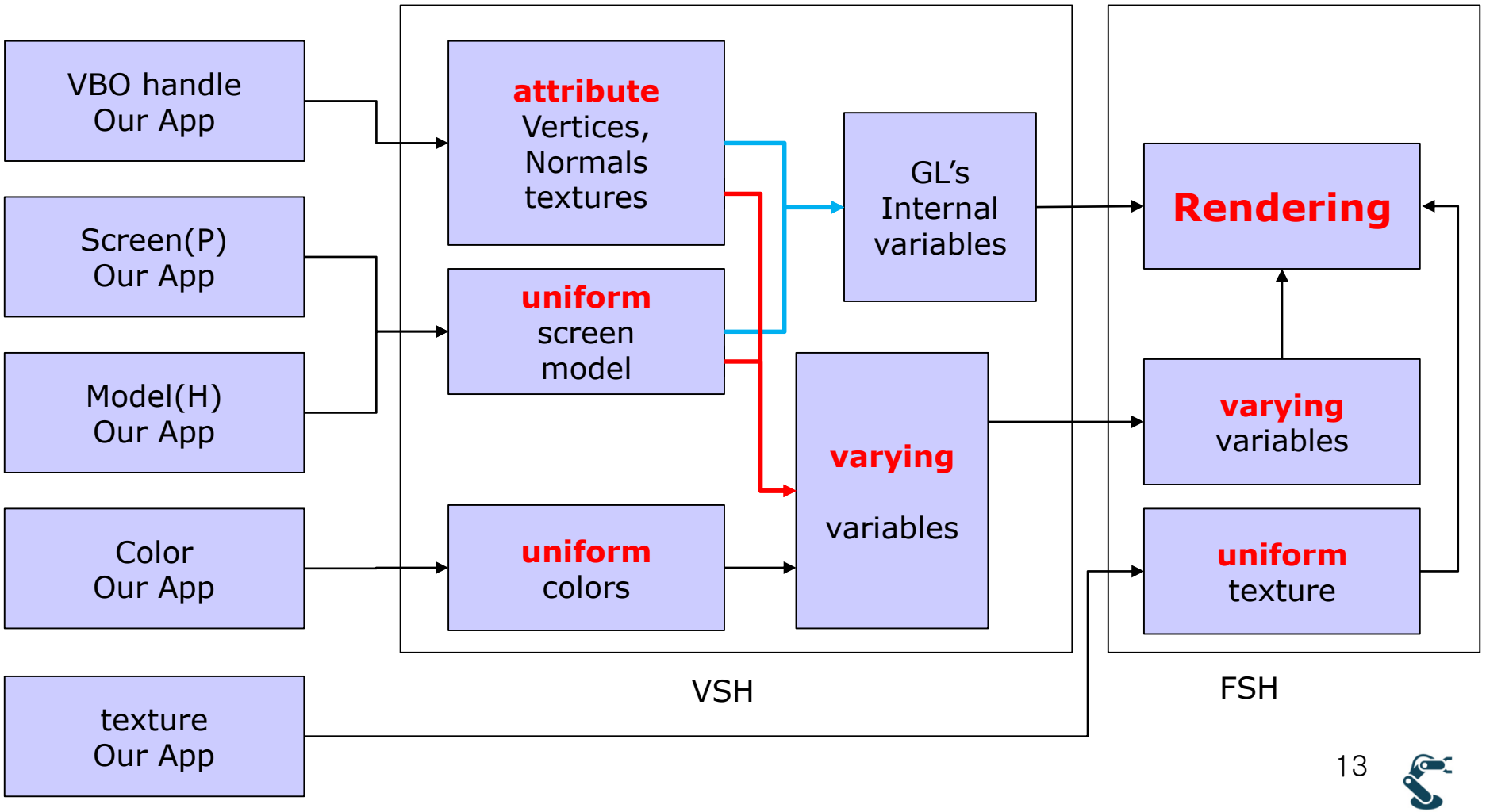
   Connected with my program

```
void main()
{
    gl_Position     = screen*model*vec4(vertices,1);

    eyePosition     = model*vec4(vertices,1);
    normal          = model*vec4(normals,0);
    ...
}
```


GLSL Grammar

- It is similar to C language
- Caution: some variables are very different
  - Ex) A= 1 → A=1.0
  - Ex) vec4 a = vec4(0,0,0,1), a.xyz = vec3(1,2,3)
  - Ex) You cannot modify “varying variable”
    - varying vec4 diffuse;
    - vec4 v = vec4(1,2,3,0);
    - diffuse = v → Error
    - See example of solid.fsh in “Blue example” of uGL-45-Sphere-Gouraud-GLSL
- Reference
  - https://www.khronos.org/opengl/wiki/Data_Type_(GLSL)#Vectors

APP→VSH→FSH
Variable Connection



See Example uGL-39-GLSL-Basic attributes in vsh

```
// arguments.
attribute vec3 vertices;
attribute vec3 normals;
attribute vec2 texcoord;

// for fsh
varying vec4 diffuseVarying;
varying vec4 normal;

uniform mat4 screen;
uniform mat4 model;

void main()
{
    gl_Position     = screen*model*vec4(ve

    eyePosition     = model*vec4(vertices,
    normal          = model*vec4(normals,0
    ...
}
```

VSH file

```
// Texture Mapping
class uVertex // Position(3), normal(3), UV(2)
{
public:
        uVertex(){}
public:
        uVector v;
        uVector n;
        struct
        {
            float u,v;
        } tx;
};
```

Lecture 9
pp. 26

uVertex uses vertex, normal, and texture, u and v

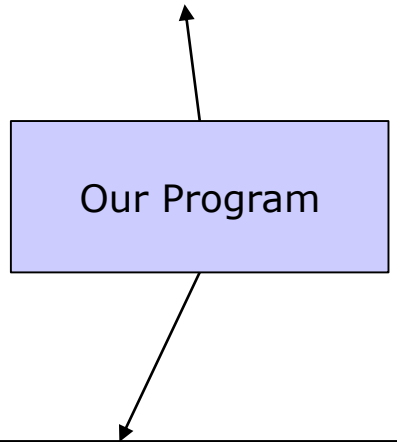
→ Three Attributes are defined.

See Example uGL-39-GLSL-Basic uniform in vsh

```
// arguments.
attribute vec3 vertices;
attribute vec3 normals;
attribute vec2 texcoord;

// for fsh
varying vec4 diffuseVarying;
varying vec4 normal;

uniform mat4 screen;
uniform mat4 model;

void main()
{
    gl_Position     = screen*model*vec4(vertices,1);

    eyePosition     = model*vec4(vertices,1);
    normal          = model*vec4(normals,0);
    ...
}
```

```
// 6. Mapping parameters
model       = glGetUniformLocation(ps, "model");
screen      = glGetUniformLocation(ps, "screen");
```

uShader::Load()




```
void uObj::Draw()
{
    // camera coordinate setting
    hMat h  = pCam->T*pCam->R;
    h   = h*Hg*H*Hp;

    glUniformMatrix4fv(pCurrentShader->screen, 1, 0, pCam->P.v);
    glUniformMatrix4fv(pCurrentShader->model, 1, 0, h.v);
```

uObj::Draw()

See Example uGL-39-GLSL-Basic uniform in vsh

```
// arguments.
attribute vec3 vertices;
attribute vec3 normals;
attribute vec2 texcoord;

// for fsh
varying vec4 diffuseVarying;
varying vec4 normal;

uniform mat4 screen;
uniform mat4 model;

void main()
{
    gl_Position     = screen*model*vec4(vertices,1);

    eyePosition     = model*vec4(vertices,1);
    normal          = model*vec4(normals,0);
    ...
}
```

```
// 6. Mapping parameters
model       = glGetUniformLocation(ps, "model");
screen      = glGetUniformLocation(ps, "screen");
```

uShader::Load()

Our Program

```
void uObj::Draw()
{
    // camera coordinate setting
    hMat h  = pCam->T*pCam->R;
    h    = h*Hg*H*Hp;

    glUniformMatrix4fv(pCurrentShader->screen, 1, 0, pCam->P.v);
    glUniformMatrix4fv(pCurrentShader->model, 1, 0, h.v);
```

uObj::Draw()

See Example uGL-39-GLSL-Basic uniform in vsh

```
// arguments.
attribute vec3 vertices;
attribute vec3 normals;
attribute vec2 texcoord;

// for fsh
varying vec4 diffuseVarying;
varying vec4 normal;

uniform mat4 screen;
uniform mat4 model;

void main()
{
    gl_Position     = screen*model*vec4(vertices,1);

    eyePosition     = model*vec4(vertices,1);
    normal          = model*vec4(normals,0);
    ...
}
```

```
// 6. Mapping parameters
model       = glGetUniformLocation(ps, "model");
screen      = glGetUniformLocation(ps, "screen");
```

uShader::Load()

Our Program

```
void uObj::Draw()
{
    // camera coordinate setting
    hMat h  = pCam->T*pCam->R;
    h   = h*Hg*H*Hp;

    glUniformMatrix4fv(pCurrentShader->screen, 1, 0, pCam->P.v);
    glUniformMatrix4fv(pCurrentShader->model, 1, 0, h.v);
```

uObj::Draw()

See Example uGL-39-GLSL-Basic uniform in vsh

Our App

```
void uObj::Draw()
{
    // camera coordinate setting
    hMat h   = pCam->T*pCam->R;
    h    = h*Hg*H*Hp;

    glUniformMatrix4fv(pCurrentShader->screen, 1, 0, pCam->P.v);
    glUniformMatrix4fv(pCurrentShader->model, 1, 0, h.v);
```

Perspective mapping, H transform

screen model

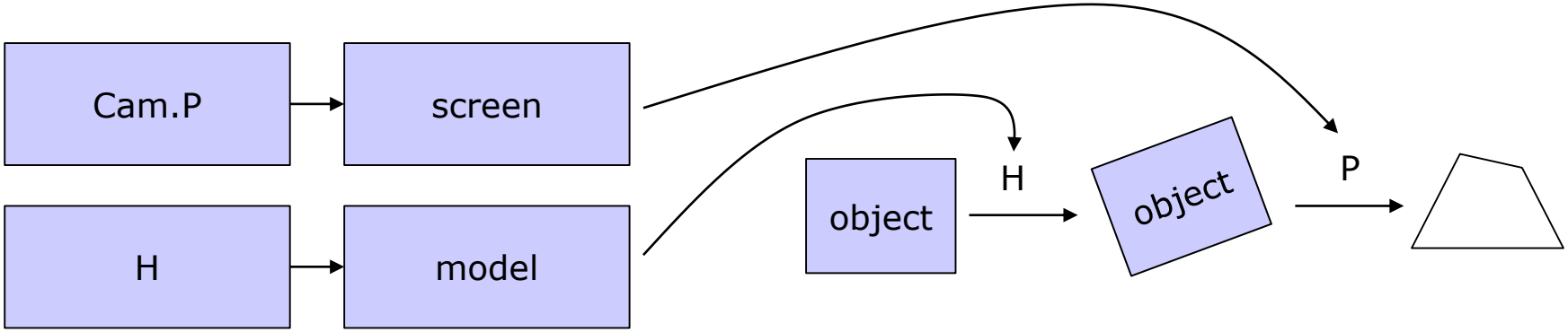
uShader's variable

GLSL in GPU

```
varying vec4 normal;

uniform mat4 screen;
uniform mat4 model;

void main()
{
    gl_Position     = screen*model*vec4(vertices,1);

    eyePosition     = model*vec4(vertices,1);
    normal          = model*vec4(normals,0);
    ...
}
```


3 Vertex and Fragment Shader (VSH and FSH)

Meaning of “screen and model” in vsh



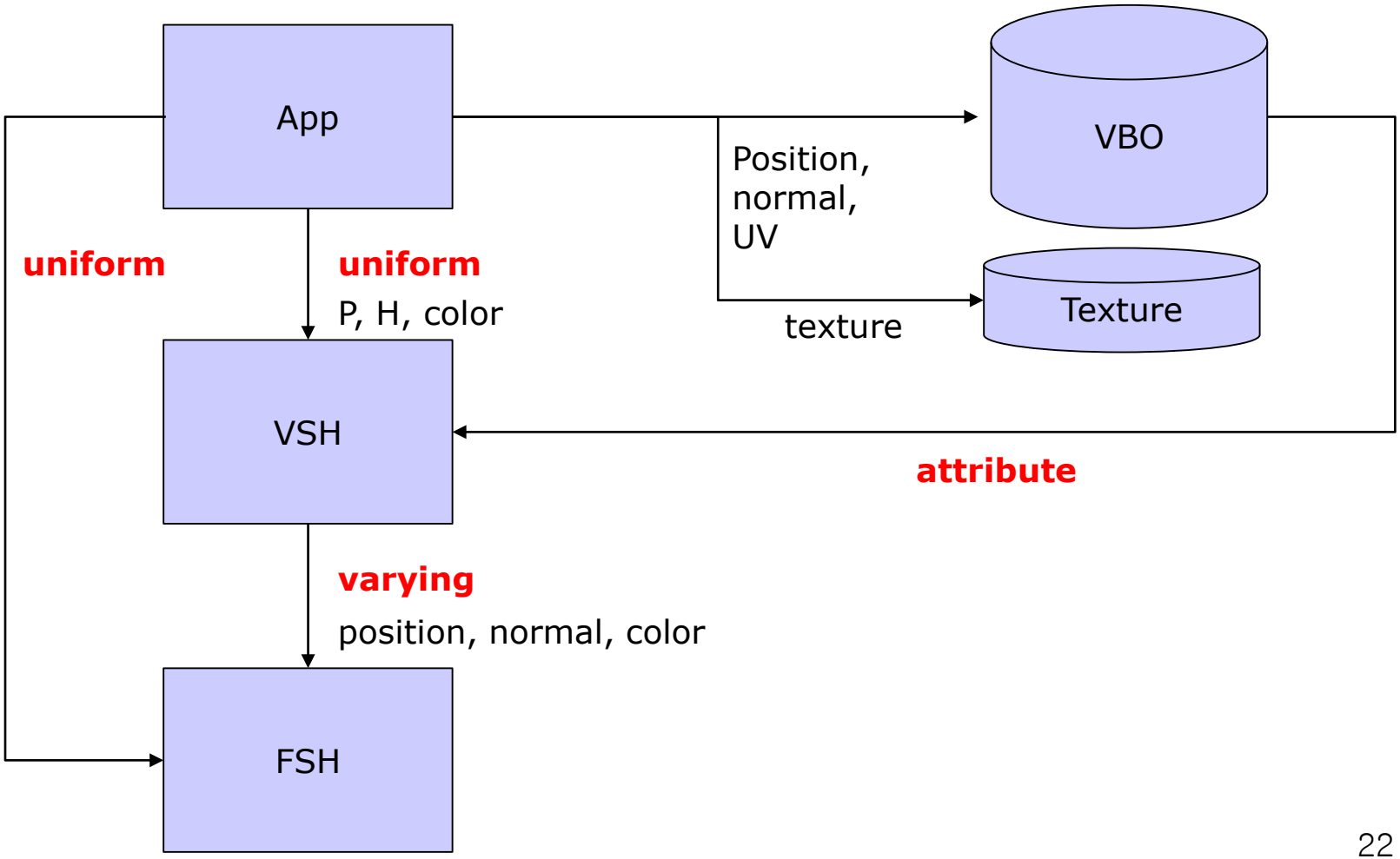

```
gl_Position     = screen*model*vec4(vertices,1);

eyePosition     = model*vec4(vertices,1);
normal          = model*vec4(normals,0);
```

- gl_Position is rendering result on 2D display
  - Screen(perspective mapping) is required
- eyePosition(Viewpoint) and normal vector
  - need not perspective mapping(screen)

Vertex shader Vs. Fragment shader

- VSH for only Transform
  - Bypassing Our App's parameter into FSH
  - APP → VSH → FSH
- VSH is not so unique in many examples
- FSH is designed to calculate Each Pixel Color
- Keep it in your mind
  - VSH and FSH are Not for Objects,
  - But for Each Pixel Color of objects

App, VBO, VSH, and FSH Connection Diagram



FSH EX1) uGL-02-Polygon-Color solid.vsh and solid.fsh

uObj.cpp

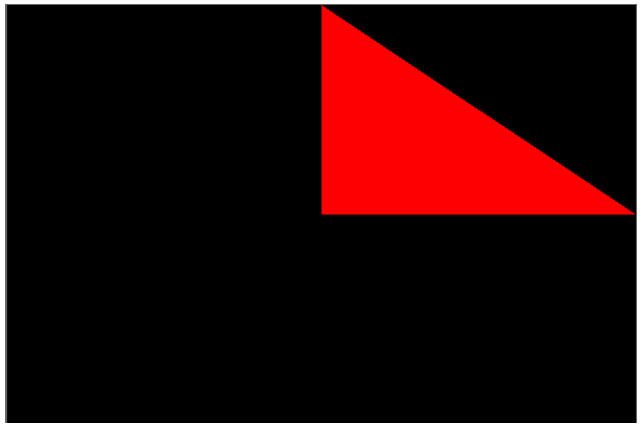
```
uObj::uObj()
{
    pVer     = NULL;
    pTemp    = NULL;
    pPoly    = NULL;

    diffuse = uColor(1,0,0);
```

Solid.vsh

```
varying vec4 diffuseVarying;
varying vec4 ambientVarying;
varying vec4 specularVarying;

// Parameters
uniform vec4 diffuse;
uniform vec4 ambient;
uniform vec4 specular;

void main()
{
    gl_Position     = vec4(vertices,1);

    diffuseVarying  = diffuse;
    ambientVarying  = ambient;
    specularVarying = specular;
}
```

Solid.fsh

```
varying vec4 diffuseVarying;
varying vec4 ambientVarying;
varying vec4 specularVarying;

void main()
{
    vec4 c     = diffuseVarying;
    gl_FragColor = c;
}
```

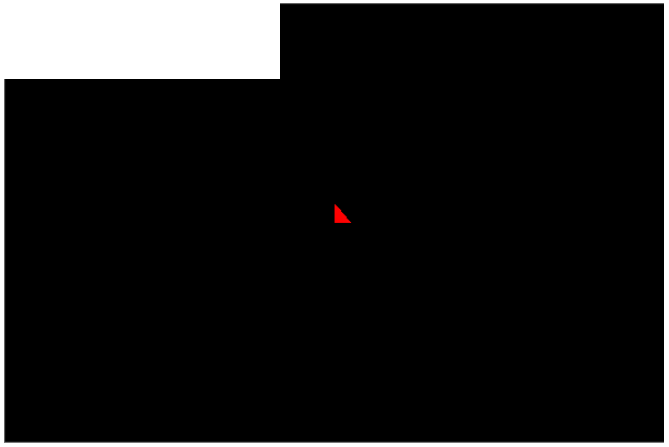


result

FSH Ex2) uGL-03-Object-Camera solid.vsh and solid.fsh

```
void main()
{
    gl_Position     = screen*model*vec4(vertices,1);

    eyePosition     = model*vec4(vertices,1);
    normal          = model*vec4(normals,0);

    diffuseVarying  = diffuse;
    ambientVarying  = ambient;
    specularVarying = specular;
}
```

Solid.vsh



Lambertian Diffuse and Phong's Specular Effect

FSH Ex 3) uGL-10-uWnd-Box-Gouraud

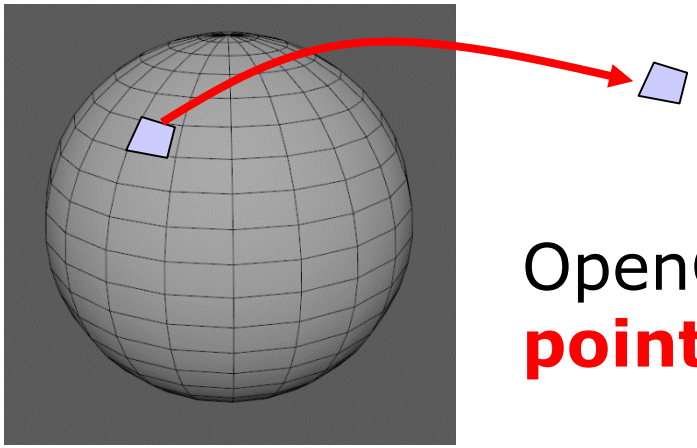
```
eyePosition     = model*vec4(vertices,1);
```

Solid.vsh

```
void main()
{
    vec3 P = vec3(eyePosition);
    vec3 N = normalize(normal.xyz);
    vec3 L = normalize(vec3(0,0,0) - P);

    float cq        = dot(L,N);

    float lamb      = max(cq,0.0);
    float specular  =0.0;

    if (lamb>0.)
    {
        vec3 reflectDir = reflect(-L,N);
        vec3 viewDir    = normalize(-P);

        float specAngle = max(dot(reflectDir, viewDir), 0);
        specular = pow(specAngle, 100.0);
    }

    ...

    vec4 c    = ambientVarying*2 + diffuseVarying*cq*5+vec4(specularVarying*specular)*1.0;
    gl_FragColor = c;
}
```

Solid.fsh

Whenever OpenGL draws one pixel, GLSL(Vsh and Fsh) is called by GPU



```
eyePosition      = model*vec4(vertices,1);
```

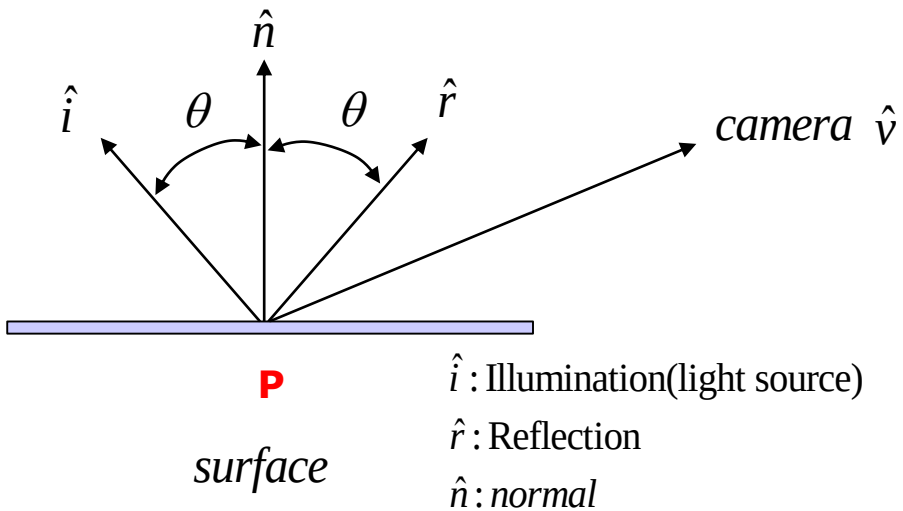
Solid.vsh

OpenGL calculates **intersection point** for painting as in Ray Tracing

- As a result,
- eyePosition by model*vertices is considered as **“Intersection Point” by the Ray from viewpoint(0,0,0)**

Think as in Ray Tracing, Can you Read it Now?

```
void main()
{
    vec3 P = vec3(eyePosition);
    vec3 N = normalize(normal.xyz);
    vec3 L = normalize(vec3(0,0,0) - P);

    float cq        = dot(L,N);

    float lamb      = max(cq,0.0);
    float specular  =0.0;

    if (lamb>0.)
    {
        vec3 reflectDir = reflect(-L,N);
        vec3 viewDir    = normalize(-P);

        float specAngle = max(dot(reflectDir, viewDir), 0);
        specular = pow(specAngle, 100.0);
    }

    ...

    vec4 c    = ambientVarying*2 + diffuseVarying*cq*5+vec4(specularVarying*specular)*1.0;
    gl_FragColor = c;
}
```

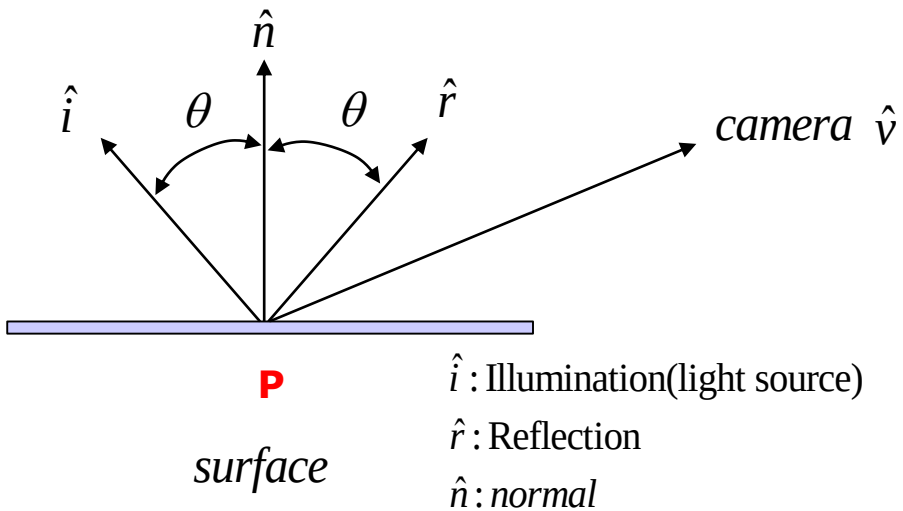


Think as in Ray Tracing, Can you Read it Now?

```
void main()
{
    vec3 P = vec3(eyePosition);
    vec3 N = normalize(normal.xyz);
    vec3 L = normalize(vec3(0,0,0) - P);
    float cq        = dot(L,N);

    float lamb      = max(cq,0.0);
    float specular  =0.0;

    if (lamb>0.)
    {
        vec3 reflectDir = reflect(-L,N);
        vec3 viewDir    = normalize(-P);

        float specAngle = max(dot(reflectDir, viewDir), 0);
        specular = pow(specAngle, 100.0);
    }

    ...

    vec4 c    = ambientVarying*2 + diffuseVarying*cq*5+vec4(specularVarying*specular)*1.0;
    gl_FragColor = c;
}
```

$L = \hat{i}$

Light source



$\hat{i}$: Illumination(light source)

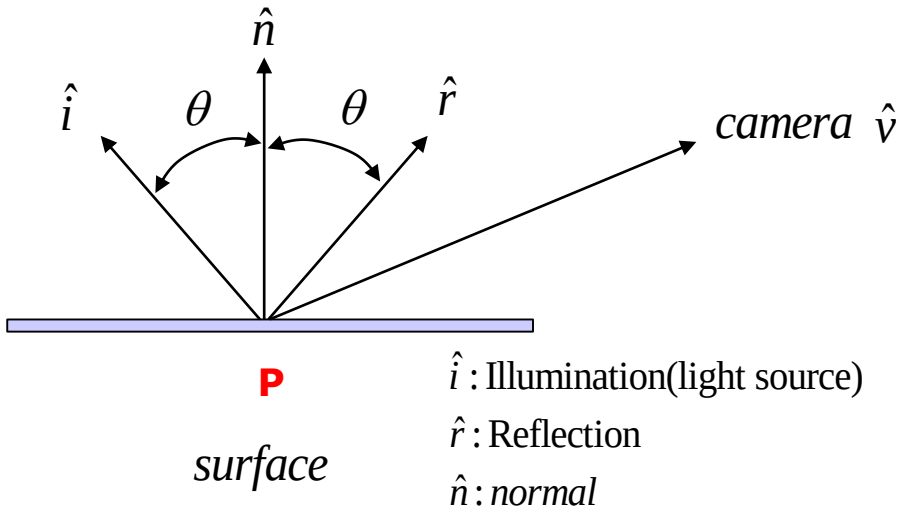
$\hat{r}$: Reflection

$\hat{n}$: *normal*

Lambertian Dot Product for Diffuse

```
void main()
{
    vec3 P = vec3(eyePosition);
    vec3 N = normalize(normal.xyz);
    vec3 L = normalize(vec3(0,0,0) - P);

    float cq        = dot(L,N);

    float lamb      = max(cq,0.0);
    float specular  =0.0;

    if (lamb>0.)
    {
        vec3 reflectDir = reflect(-L,N);
        vec3 viewDir    = normalize(-P);

        float specAngle = max(dot(reflectDir, viewDir), 0);
        specular = pow(specAngle, 100.0);
    }

    ...

    vec4 c    = ambientVarying*2 + diffuseVarying*cq*5+vec4(specularVarying*specular)*1.0;
    gl_FragColor = c;
}
```

$\cos\theta = \hat{i} \circ \hat{n}$



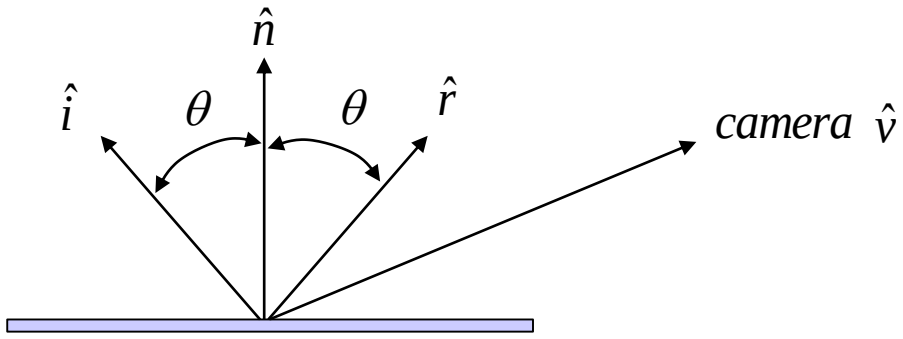
$\hat{i}$: Illumination(light source)

$\hat{r}$: Reflection

$\hat{n}$: *normal*

Lambertian Dot Product for Diffuse

```
void main()
{
    vec3 P = vec3(eyePosition);
    vec3 N = normalize(normal.xyz);
    vec3 L = normalize(vec3(0,0,0) - P);

    float cq       = dot(L,N);

    float lamb     = max(cq,0.0);
    float specular
    if (lamb>0.)
    {
        vec3 reflectDir = reflect(-L,N);
        vec3 viewDir    = normalize(-P);

        float specAngle = max(dot(reflectDir, viewDir), 0);
        specular = pow(specAngle, 100.0);
    }

    ...

    vec4 c    = ambientVarying*2 + diffuseVarying*cq*5+vec4(specularVarying*specular)*1.0;
    gl_FragColor = c;
}
```



$\cos\theta < 0$: Hidden surface

$\hat{i}$: Illumination(light source)

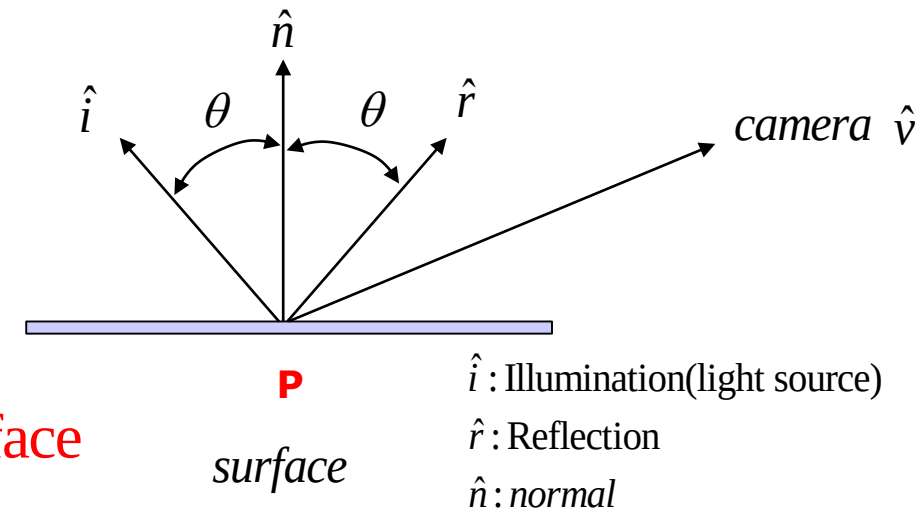
$\hat{r}$: Reflection

$\hat{n}$: *normal*

Phong's Specular Color with Reflected Vector

```
void main()
{
    vec3 P = vec3(eyePosition);
    vec3 N = normalize(normal.xyz);
    vec3 L = normalize(vec3(0,0,0) - P);

    float cq        = dot(L,N);

    float lamb      = max(cq,0.0);
    float specular  =0.0;

    if (lamb>0.)
    {
        vec3 reflectDir = reflect(-L,N);
        vec3 viewDir    = normalize(-P);

        float specAngle = max(dot(reflectDir, viewDir), 0);
        specular = pow(specAngle, 100.0);
    }

    ...

    vec4 c    = ambientVarying*2 + diffuseVarying*cq*5+vec4(specularVarying*specular)*1.0;
    gl_FragColor = c;
}
```

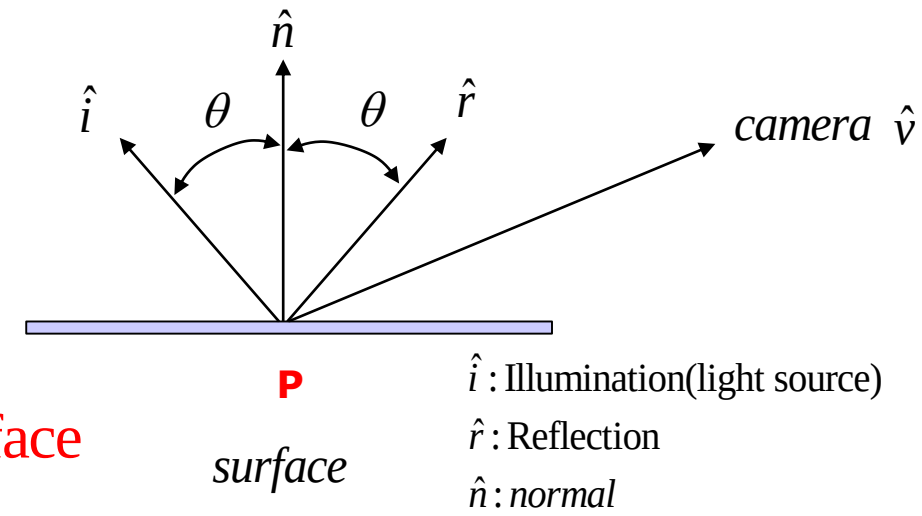
If It is Not a Hidden surface



the Ray from viewpoint(0,0,0)

Phong's Specular Color with Reflected Vector

```
void main()
{
    vec3 P = vec3(eyePosition);
    vec3 N = normalize(normal.xyz);
    vec3 L = normalize(vec3(0,0,0) - P);

    float cq        = dot(L,N);

    float lamb      = max(cq,0.0);
    float specular  =0.0;

    if (lamb>0.)
    {
        vec3 reflectDir = reflect(-L,N);
        vec3 viewDir    = normalize(-P);

        float specAngle = max(dot(reflectDir, viewDir), 0);
        specular = pow(specAngle, 100.0);
    }

    ...

    vec4 c    = ambientVarying*2 + diffuseVarying*cq*5+vec4(specularVarying*specular)*1.0;
    gl_FragColor = c;
}
```



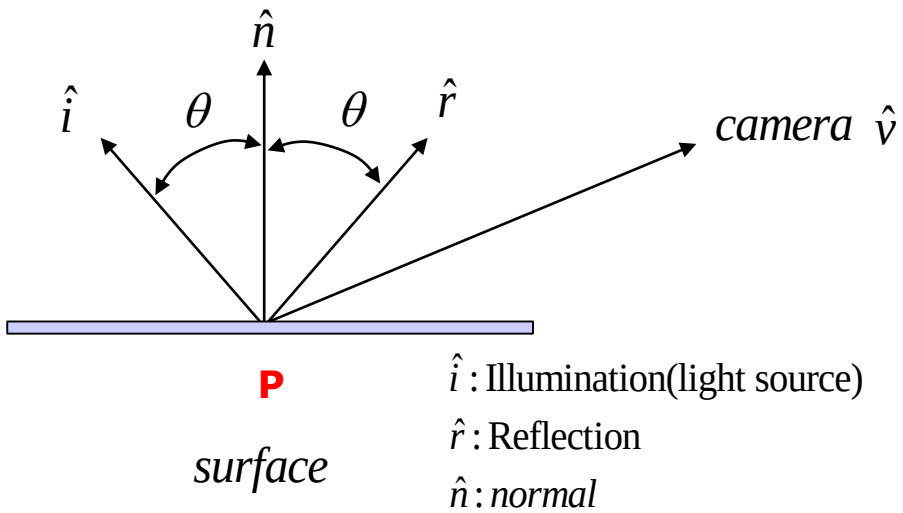
If It is Not a Hidden surface

the Ray from viewpoint(0,0,0)

$$\cos\alpha = \hat{r} \circ \hat{v} \quad \therefore S(\alpha) = \cos\alpha^{s}$$

Phong's Specular Color with Reflected Vector

```
void main()
{
    vec3 P = vec3(eyePosition);
    vec3 N = normalize(normal.xyz);
    vec3 L = normalize(vec3(0,0,0) - P);

    float cq        = dot(L,N);

    float lamb      = max(cq,0.0);
    float specular  =0.0;

    if (lamb>0.)
    {
        vec3 reflectDir = reflect(-L,N);
        vec3 viewDir    = normalize(-P);

        float specAngle = max(dot(reflectDir, viewDir), 0);
        specular = pow(specAngle, 100.0);
    }

    ...
    vec4 c    = ambientVarying*2 + diffuseVarying*cq*5+vec4(specularVarying*specular)*1.0;
    gl_FragColor = c;
}
```

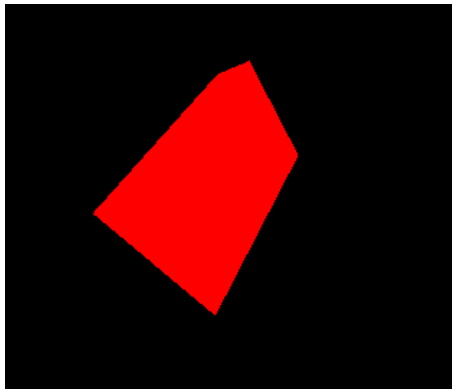


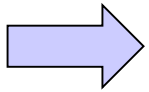
Color = ambient + diffuse * cos(q) + specular * pow(angle,100)

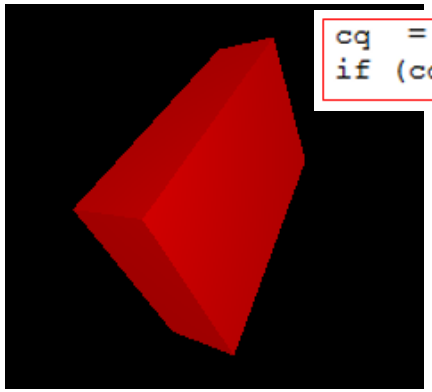
Ex4) uGL-40-GLSL-Box-Gouraud
Two types of Diffuse

```
float cq        = dot(L,N);

float lamb      = max(cq,0.0);
float specular  =0.0;

if (lamb>0.)
{
    vec3 reflectDir = reflect(-L,N);
    vec3 viewDir    = normalize(-P);

    float specAngle = max(dot(reflectDir, viewDir), 0);
    specular = pow(specAngle, 100.0);
}
```

```
vec4 c    = ambientVarying*2 + diffuseVarying*cq*5+vec4(specularVarying*specular)*1.0;
gl_FragColor = c;
```



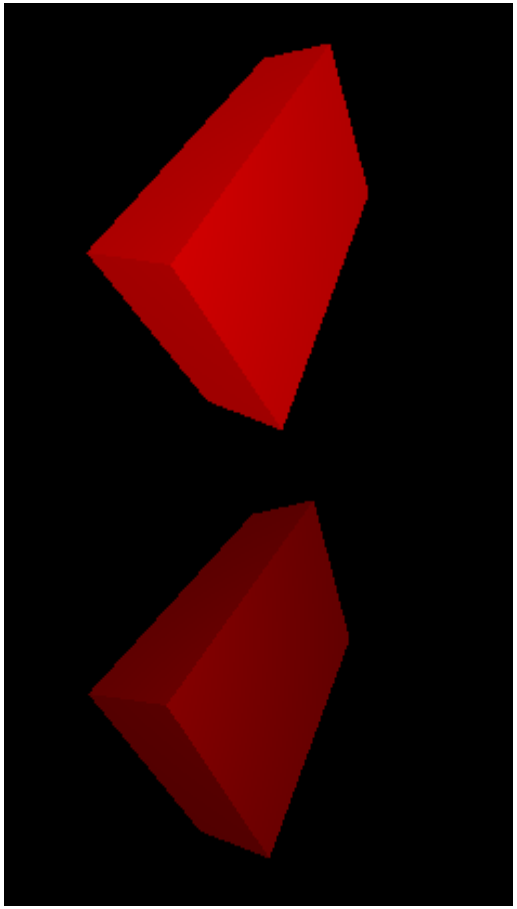




```
cq  = cq/20;
if (cq<0)   cq = 0.;
```

Control diffuse by GLSL

Ex5) uGL-40-GLSL-Box-Gouraud Ambient Effect



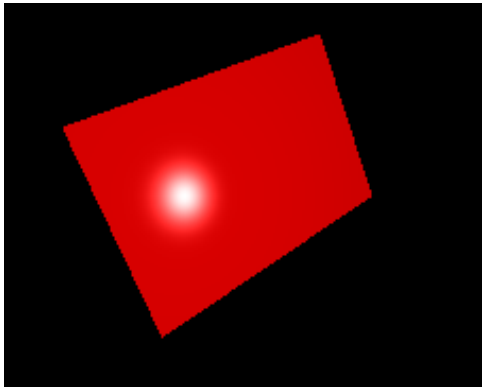
Basic Type

```
vec4 c     = ambientVarying*2 +
             diffuseVarying*cq*5+
             vec4(specularVarying*specular)*1.0;
```

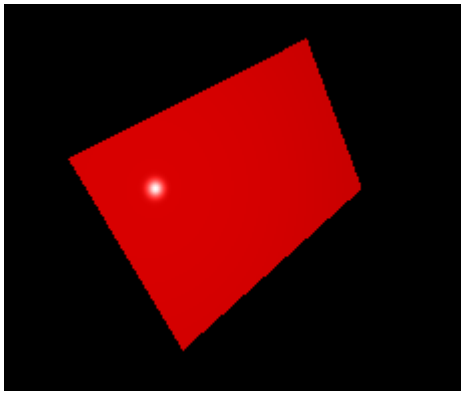
```
vec4 c     = ambientVarying*1 +
             diffuseVarying*cq*5+
             vec4(specularVarying*specular)*1.0;
```

Q: Why lower ambient becomes darker?

Ex6) uGL-40-GLSL-Box-Gouraud
More Specular by Power and by Factor

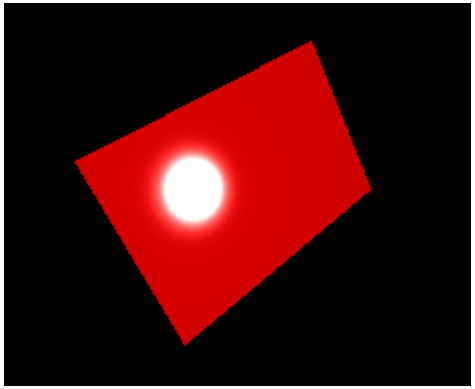


Pow by 100
Specular * 1.0



```
specular = pow(specAngle, 1000.0);
vec4(specularVarying*specular)*1.0;
```

Pow by 1000
Specular * 1.0



Pow by 100
Specular * 5.0

```
vec4 c     = ambientVarying*2 + diffuseVarying*cq*5+vec4(specularVarying*specular)*5.0;
```

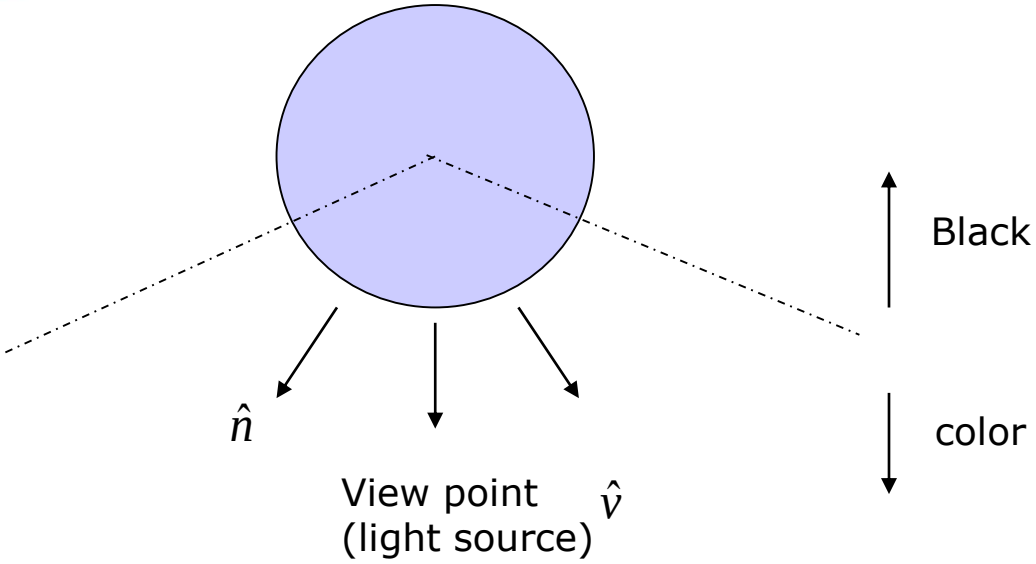
Simple Cartoon Rendering

- Game with Oriental ink painting

Edge is Over colored using Dot Product ex)uGL-41-GLSL-Dog-Rendering

$$L = 0 - P$$

$$\cos\theta = \hat{n} \circ \hat{L}$$



Ex.7) uGL-41-GLSL-Dog-Rendering
Cartoon-Rendering

$$L = 0 - P$$

$$\cos\theta = \hat{n} \circ \hat{L}$$

```
r = dot(N,L);

if (c<0.25)
{
    // the larger c is, the darker color
    // --> Boundary
    c = max(c,0);
    color    = ambientVarying*c;
    color.a = (1-c);

}
else if (c<0.4) // solid for ambient
{
    c = 0.1;
    color    = ambientVarying*c;
    color.a = (1-c);
}
else    // highlighted area
{
    float s = pow(c*r,8);
    float th = 0.8+10*s;

    if (th>1)    th;
    else         th=0.8;

    color    = diffuseVarying*th;
    color.a = 0.99;
}
```