Shading

CS425: Computer Graphics I

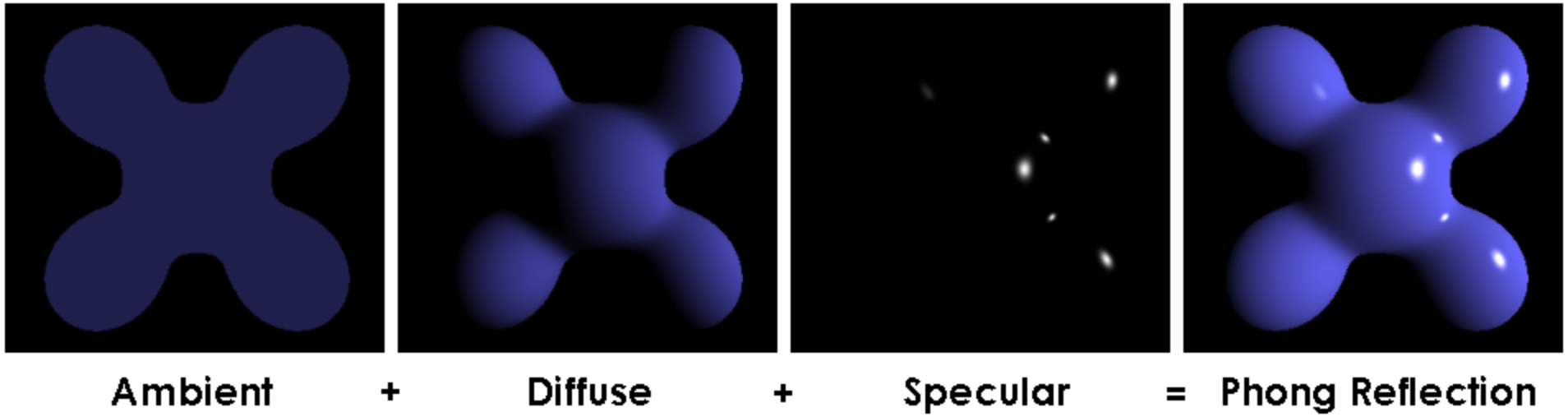
Khairi Reda

Shading

- How do we compute the color for the whole surface?
- Illumination models (e.g., Phong) provide a way to compute the color at specific points.
- **How do we color the entire object?**

Phong model

$$f_{Phong}(\mathbf{L}_{light}, \mathbf{l}, \mathbf{v}, \mathbf{n}) = k_{ambient}\mathbf{L}_{ambient} + \sum_{m \in lights} k_{diffuse}(\mathbf{l_m} \bullet \mathbf{n})\mathbf{L}_{m,diffuse} + k_{specular}(\mathbf{r_m} \bullet \mathbf{v})^{\alpha}\mathbf{L}_{m,specular}$$



Shading (in shaders)

- What we need in the shaders?

$$f_{Phong}(\mathbf{L}_{light}, \mathbf{l}, \mathbf{v}, \mathbf{n}) = k_{ambient}\mathbf{L}_{ambient} + \sum_{m \in lights} k_{diffuse}(\mathbf{l_m} \bullet \mathbf{n})\mathbf{L}_{m,diffuse} + k_{specular}(\mathbf{r_m} \bullet \mathbf{v})^{\alpha}\mathbf{L}_{m,specular}$$

  - Light and material properties (uniforms)

Shading (in shaders)

- What we need in the shaders?

$$f_{Phong}(\mathbf{L}_{light}, \mathbf{l}, \mathbf{v}, \mathbf{n}) = k_{ambient}\mathbf{L}_{ambient} + \sum_{m \in lights} k_{diffuse}(\mathbf{l_m} \bullet \mathbf{n})\mathbf{L}_{m,diffuse} + k_{specular}(\mathbf{r_m} \bullet \mathbf{v})^{\alpha}\mathbf{L}_{m,specular}$$

  - Light and material properties (uniforms)
  - Normals and reflect (per vertex, optionally interpolated per fragment)

Shading (in shaders)

- What we need in the shaders?

$$f_{Phong}(\mathbf{L}_{light}, \mathbf{l}, \mathbf{v}, \mathbf{n}) = k_{ambient}\mathbf{L}_{ambient} + \sum_{m \in lights} k_{diffuse}(\mathbf{l_m} \bullet \mathbf{n})\mathbf{L}_{m,diffuse} + k_{specular}(\mathbf{r_m} \bullet \mathbf{v})^{\alpha}\mathbf{L}_{m,specular}$$

  - Light and material properties (uniforms)
  - Normals and reflect (per vertex, optionally interpolated per fragment)
  - Light position (uniform)

Shading (in shaders)

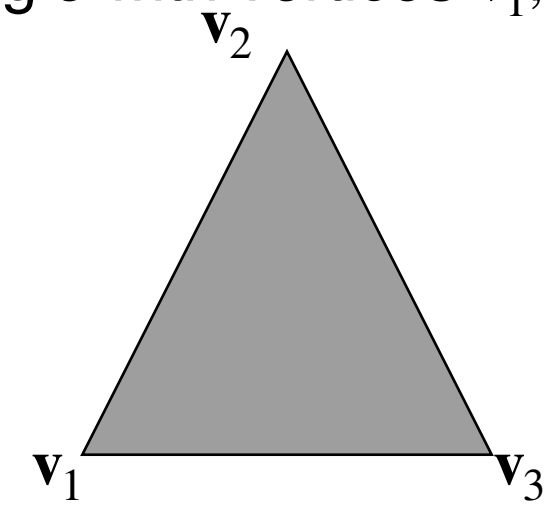
- What we need in the shaders?

$$f_{Phong}(\mathbf{L}_{light}, \mathbf{l}, \mathbf{v}, \mathbf{n}) = k_{ambient}\mathbf{L}_{ambient} + \sum_{m \in lights} k_{diffuse}(\mathbf{l_m} \bullet \mathbf{n})\mathbf{L}_{m,diffuse} + k_{specular}(\mathbf{r_m} \bullet \mathbf{v})^{\alpha}\mathbf{L}_{m,specular}$$

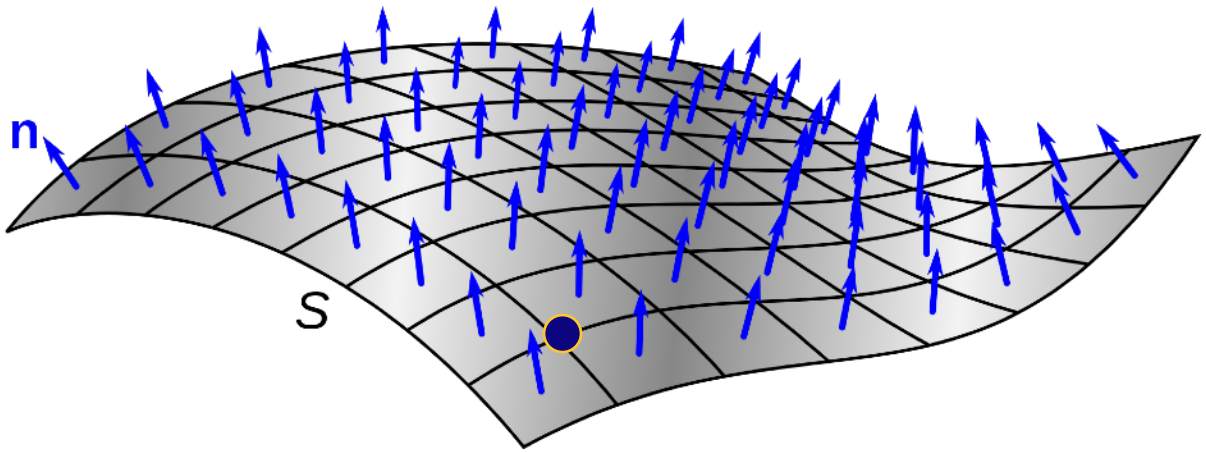
  - Light and material properties (uniforms)
  - Normals and reflect (per vertex, optionally interpolated per fragment)
  - Light position (uniform)
    - Do we need to send $\mathbf{v}$? We can work on camera space! (viewer is always at (0, 0, 0) after model-view transform)

Computing normals

Triangle with vertices $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$:

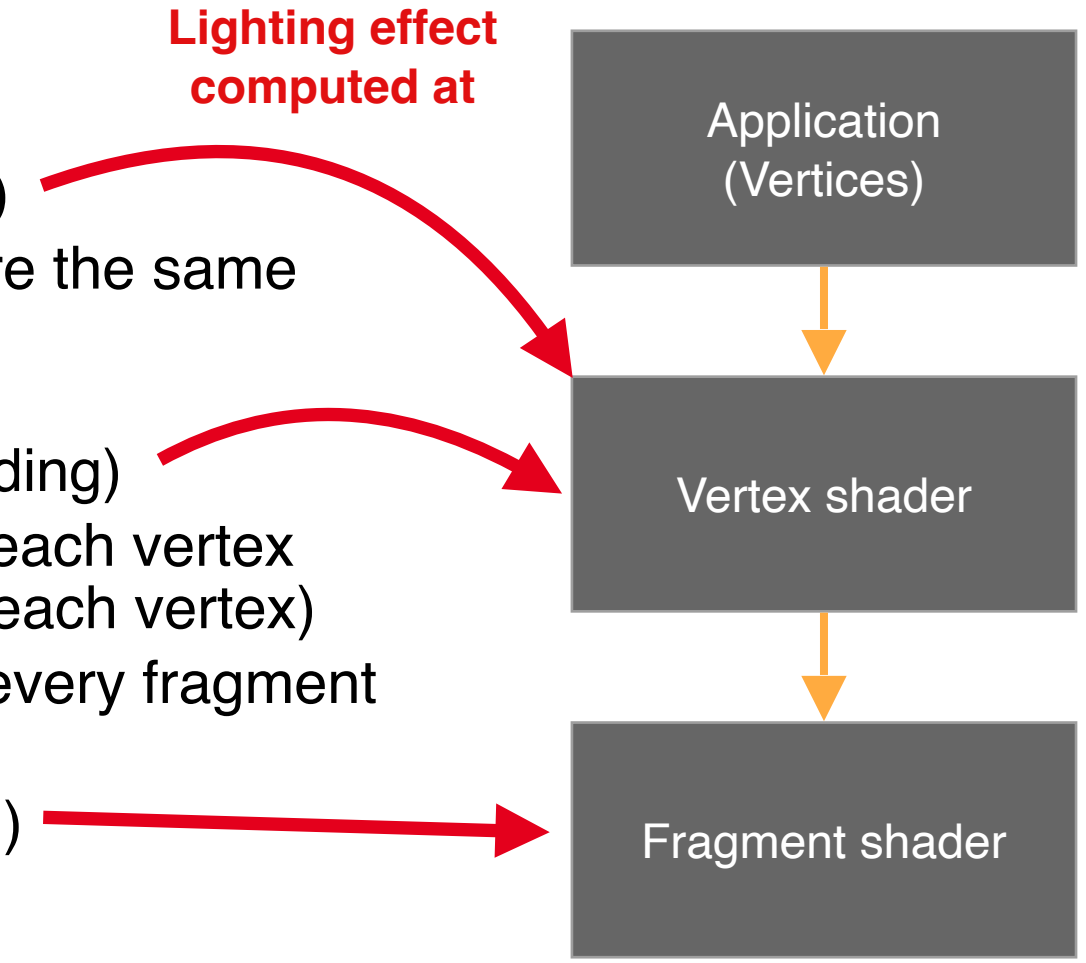


$$\mathbf{n} = (\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_3 - \mathbf{v}_1)$$



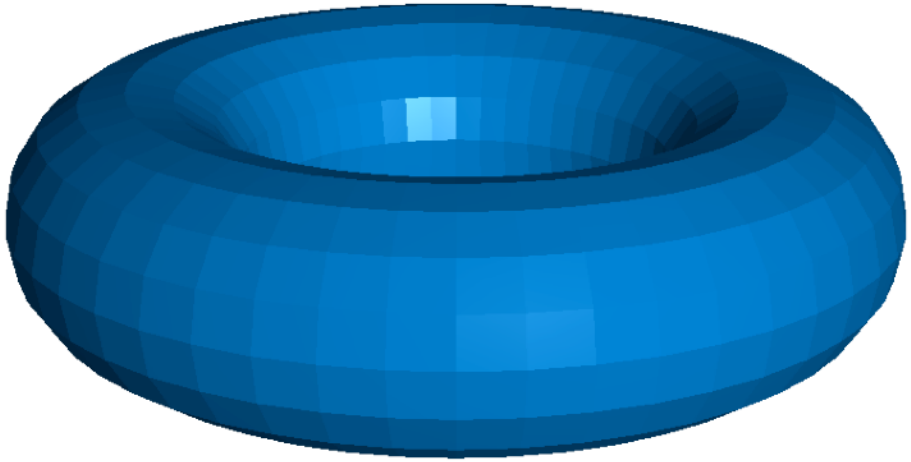
Shading: per-triangle, per-vertex, per-pixel

- **Once per triangle** (flat shading)
  - All vertices of a triangle share the same normal
- **Once per vertex** (Gouraud shading)
  - Shading color calculated at each vertex (slightly different normal for each vertex)
  - Vertex color interpolated at every fragment
- **Once per pixel** (Phong shading)



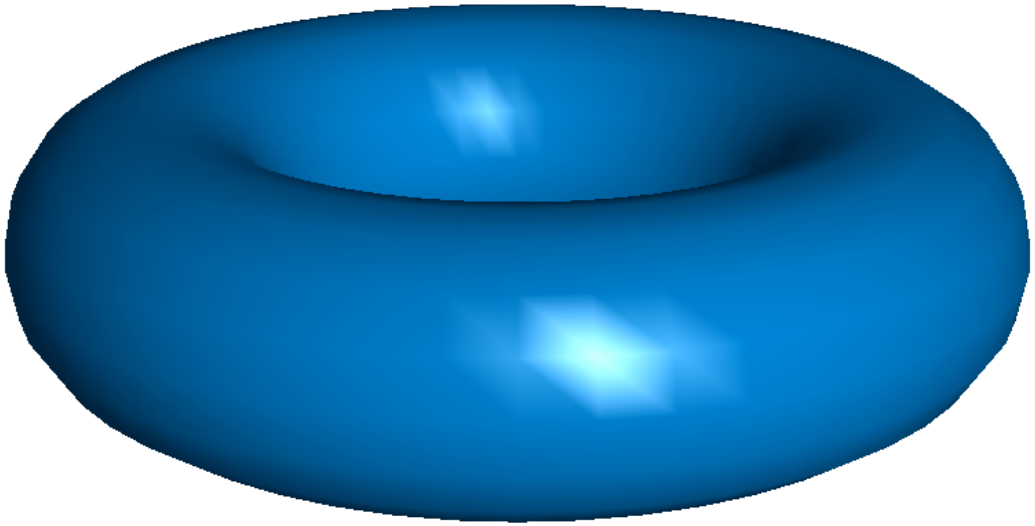
Flat shading

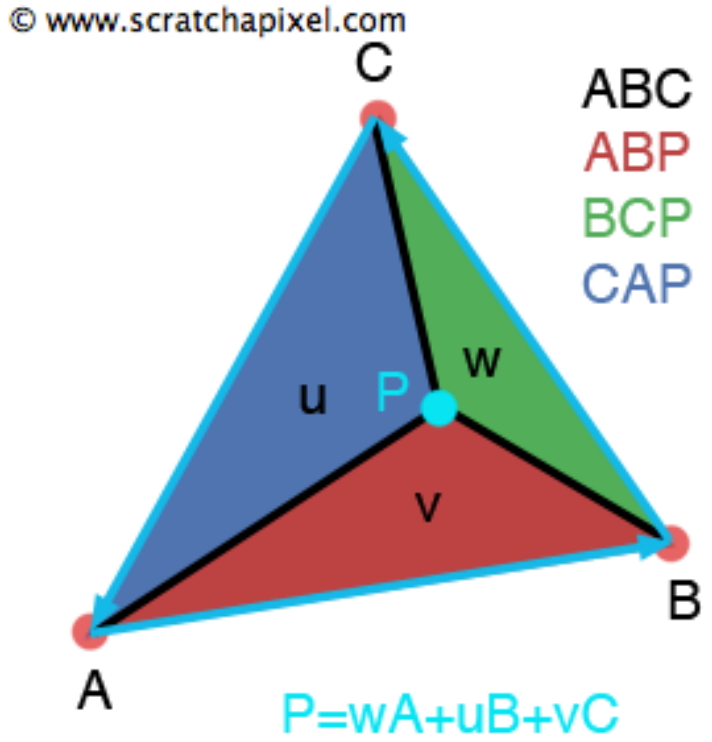
- Compute one color per polygon.
- All pixels in the same polygon are colored by the same color.



Gouraud shading

- Compute one color per vertex.
- Interpolate vertex **colors** across triangles.

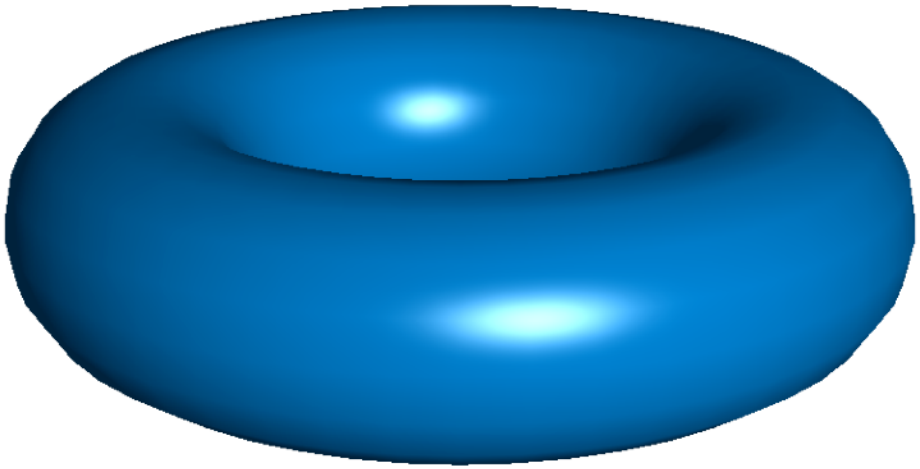


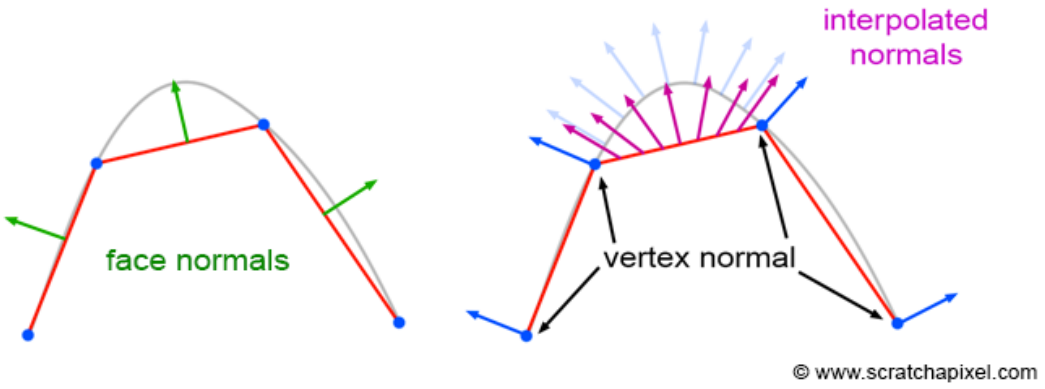


w = Area of BCP / area of ABC
u = Area of CAP / area of ABC
v = Area of ABP / area of ABC

Phong shading

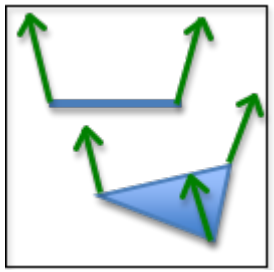
- One color per pixel.
- Interpolates vertex **normals** across triangles.
- Illumination model evaluated at each pixel.



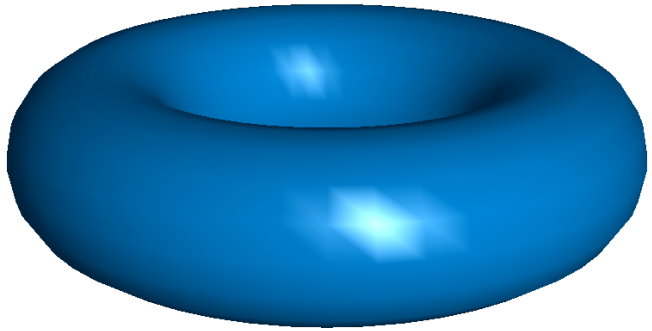


Comparison

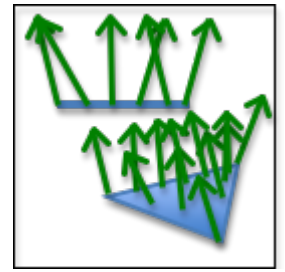
Gouraud shading



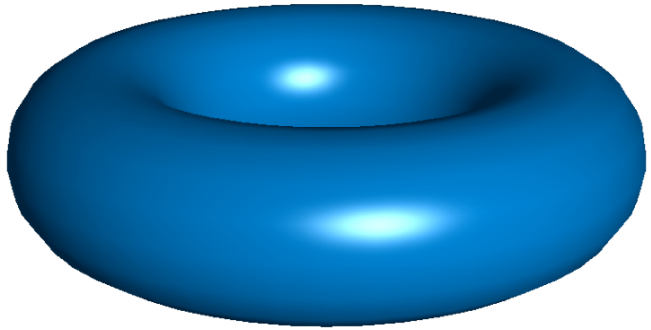
Vertex shader → color → Fragment shader



Phong shading

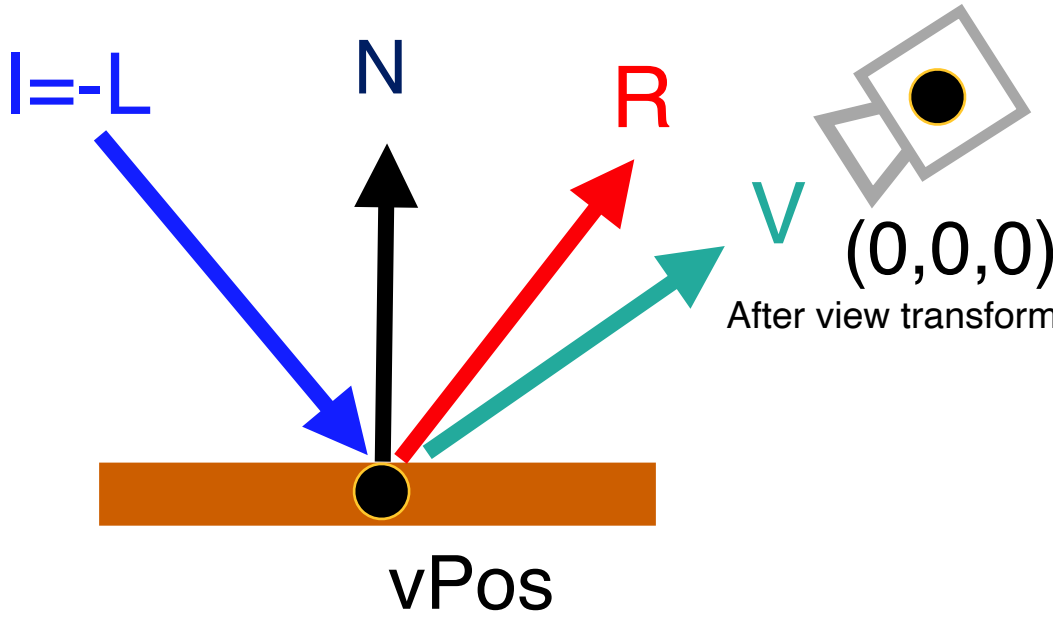


Vertex shader → normal position → Fragment shader



Gouraud vertex shader anatomy

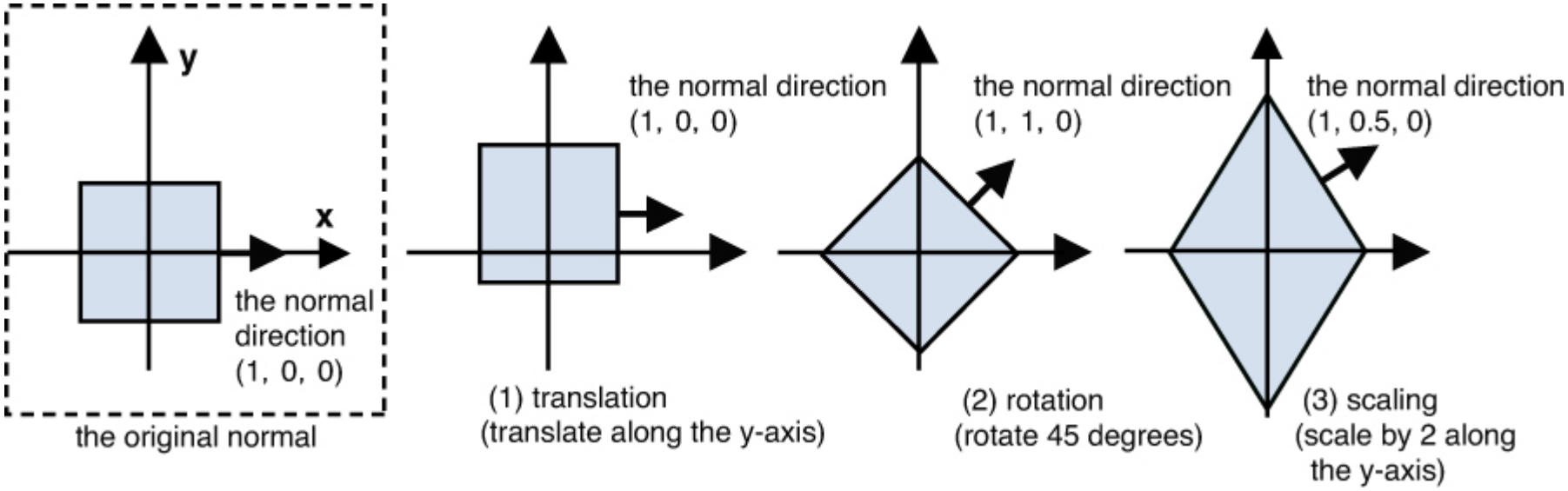
```
4     // position and normal (vertex attributes)
5     layout(location=0) in vec3 aPos;
6     layout(location=1) in vec3 aNormal;
7
8     // model, view, and projection matrices
9     uniform mat4 uModel, uView, uProj;
10
11    // light direction
12    uniform vec3 uLightDir;
13
14    // output the color of the vertex
15    out vec3 vColor;
16    void main()
17    {
18        // transform and normalize the normals
19        vec3 N = normalize(mat3(uModel) * aNormal);
20        vec3 L = normalize(uLightDir);
21
22        vec4 vPos = uView * uModel * vec4(aPos, 1.0);
23        vec3 V = normalize(-vPos.xyz);
24        vec3 R = reflect(-L, N);
25
26        float diffuse = max(dot(N, L), 0.0);
27        float specular = pow(max(dot(R, V), 0.0), 16.0);
28
29        // evaluate model at the vertex
30        vec3 color = vec3(0.2)              // amient
31            + diffuse * vec3(0.6, 0.8, 1.0)     // diffuse
32            + specular * vec3(1.0, 1.0, 1.0);   // specular
33
34        vColor = color;      // vertex color (to be interpolated
35        gl_Position = uProj * uView * uModel * vec4(aPos, 1.0);
36    }
```



$$k_{diffuse}(\mathbf{l_m} \bullet \mathbf{n})\mathbf{L}_{m,diffuse}$$

$$k_{specular}(\mathbf{r_m} \bullet \mathbf{v})^{\alpha}\mathbf{L}_{m,specular}$$

Transforming normals



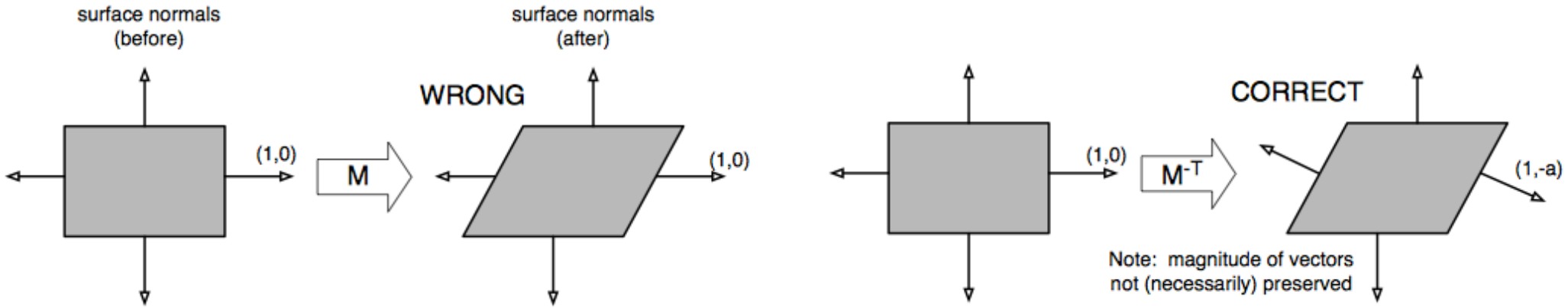
How can we transform normals?

- Vertex:

```
v = modelViewMatrix * vec4(position, 1);
```

- Normal:

```
n = uNormal * vec4(normal, 1);
```



$$\mathbf{NormalMatrix} = (\mathbf{ModelView}^{-1})^{\mathbf{T}}$$

How can we transform normals?

$$\mathbf{t} = \mathbf{p_2} - \mathbf{p_1}$$
$$\mathbf{M} * \mathbf{t} = \mathbf{M} * (\mathbf{p_2} - \mathbf{p_1})$$
$$\mathbf{t}' = \mathbf{p_2}' - \mathbf{p_1}'$$

$\mathbf{n}$ and $\mathbf{t}$ are perpendicular

$\mathbf{n}'$ and $\mathbf{t}'$ must also be perpendicular

We find normal matrix by solving: $\mathbf{n}' . \mathbf{t}' = \mathbf{0}$

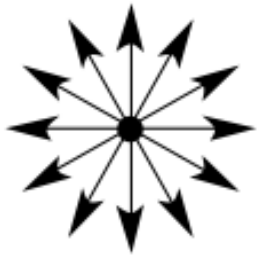
What is $\mathbf{n}'$? $\mathbf{G} * \mathbf{n}$

What is $\mathbf{t}'$? $\mathbf{M} * \mathbf{t}$

$$(\mathbf{G} * \mathbf{n}) . (\mathbf{M} * \mathbf{t}) = \mathbf{0}$$
$$\mathbf{G} = (\mathbf{M}^{-1})^{\mathbf{T}}$$

Point and directional light

Directional Light



Point Light

```
uniform vec3 lightDir;
```

```
lightDir = normalize(lightPos - pos);
```