CS130 : Computer Graphics

Lecture 14: Lighting and Shading (cont.)

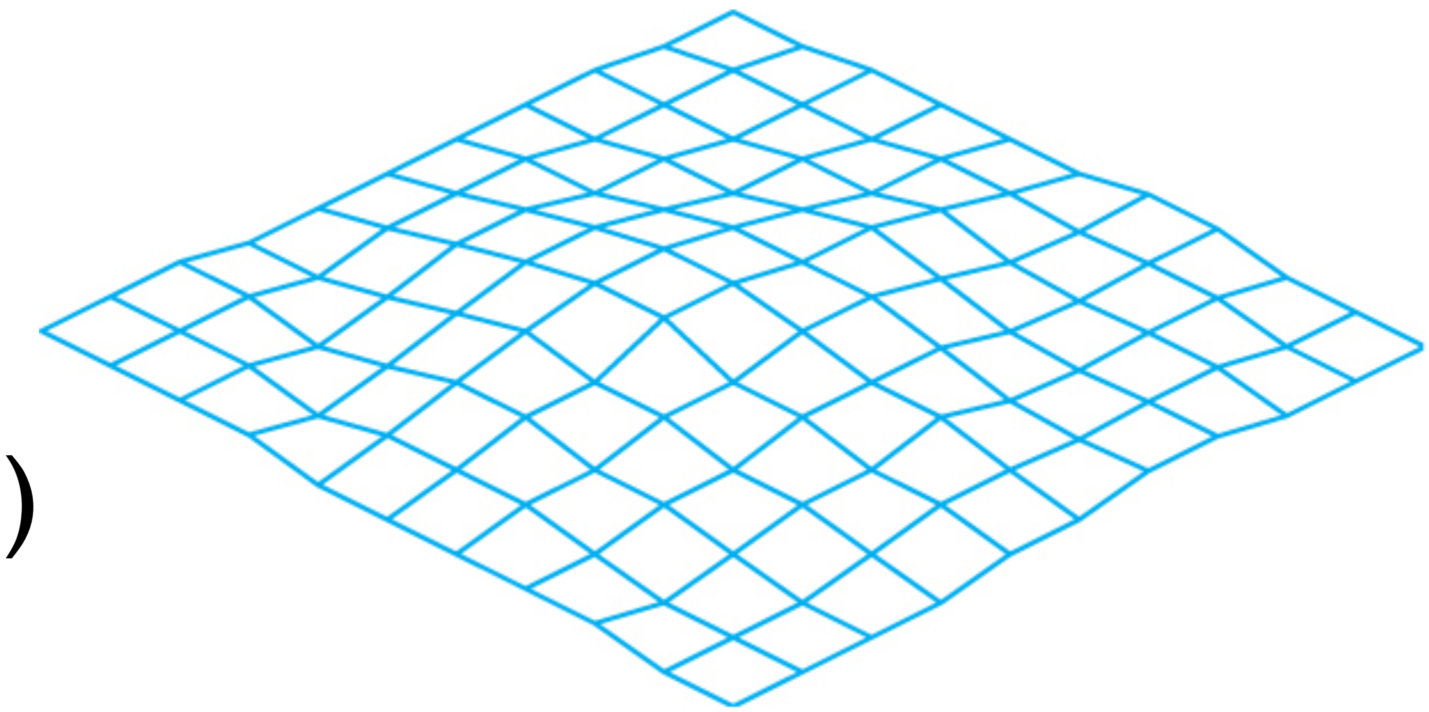
Tamar Shinar
Computer Science & Engineering
UC Riverside

Shading Polygonal Geometry

Smooth surfaces are often approximated by polygons

Shading approaches:

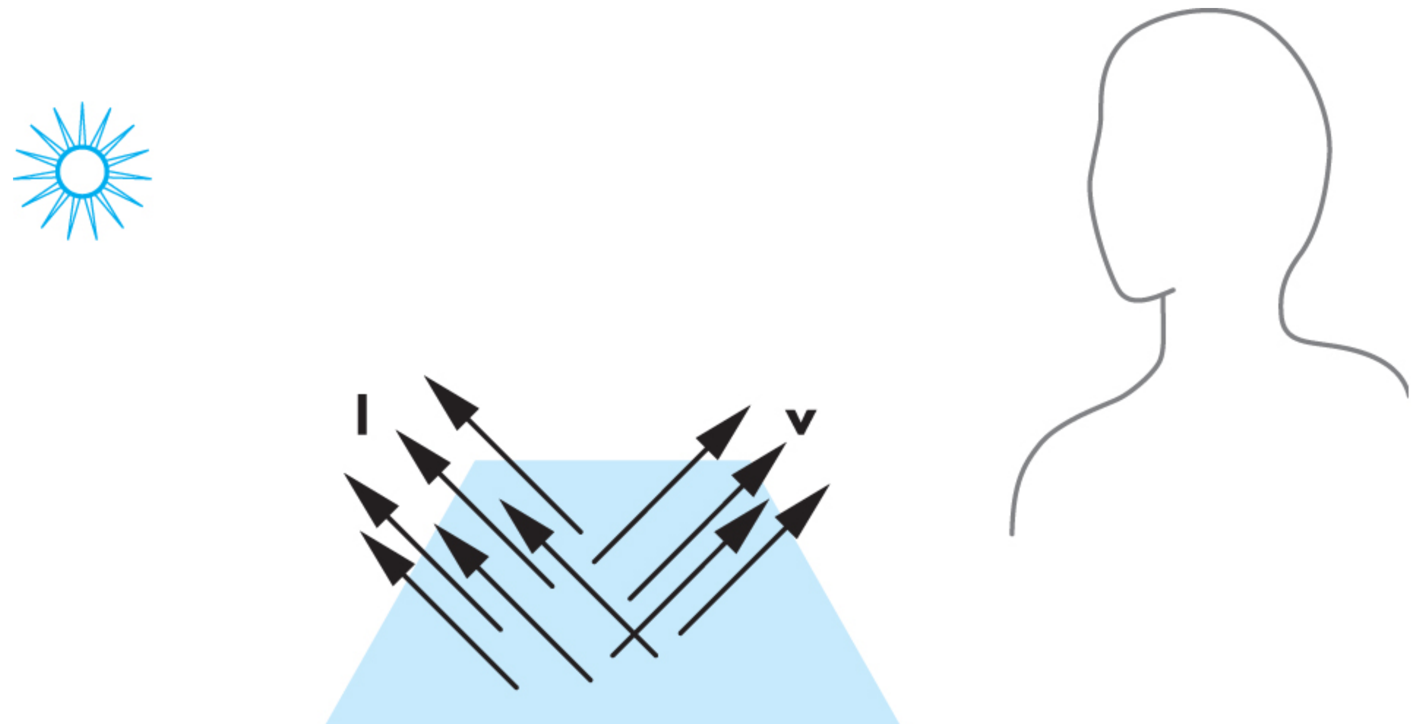
1. Flat
2. Smooth (Gouraud)
3. Phong



each polygon is flat and has a well-defined normal

Flat Shading

do the shading
calculation once
per **polygon**

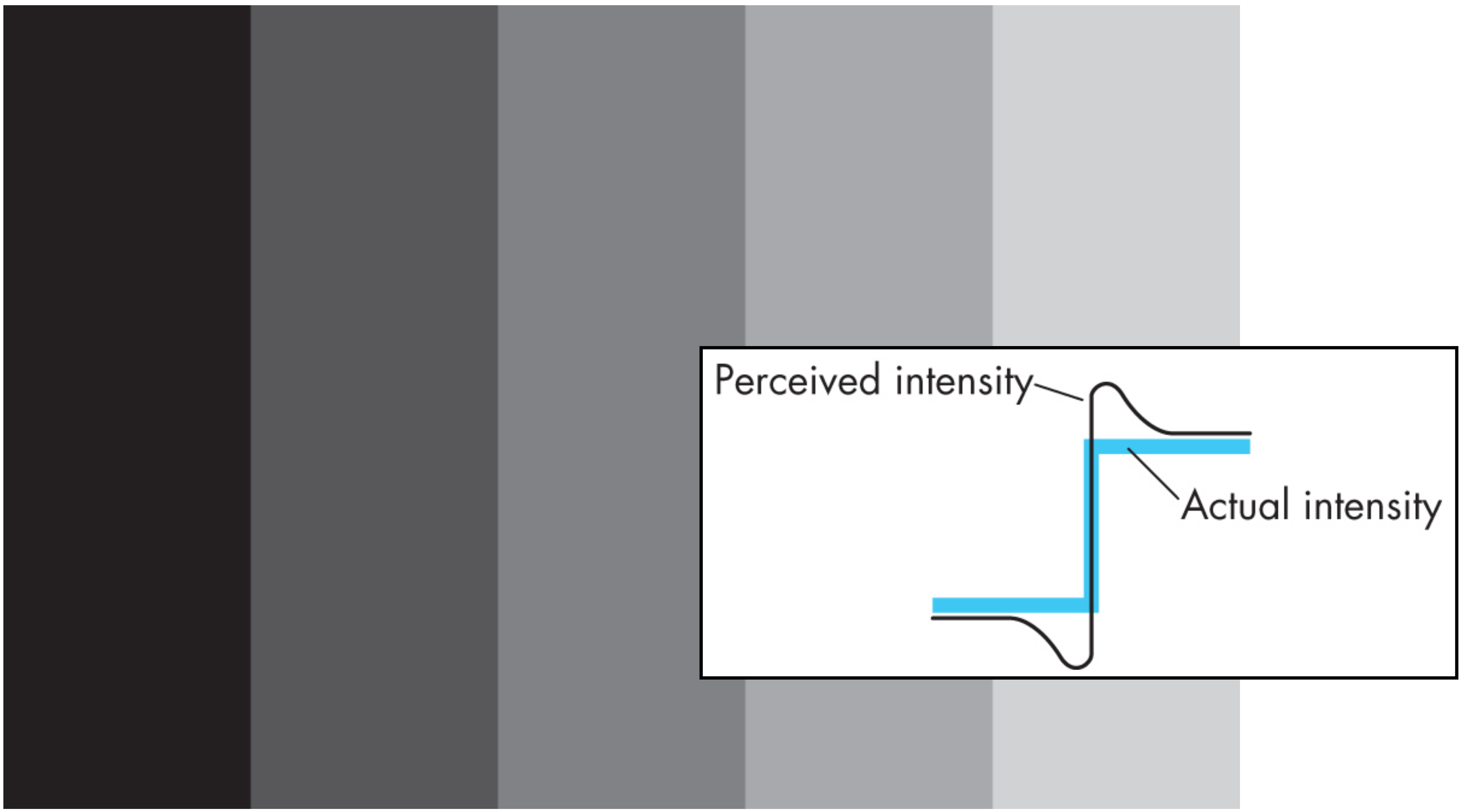


valid for light at ∞
and viewer at ∞
and faceted surfaces

In general, l, n, and v vary from point to point on a surface. If we assume a distant viewer, v can be thought of as constant. If we assume a distant light source, l can be thought of as constant. For a flat polygon, n is constant.

If the light source or viewer is not at inf, we need heuristic for picking color – e.g., first vertex, or polygon center

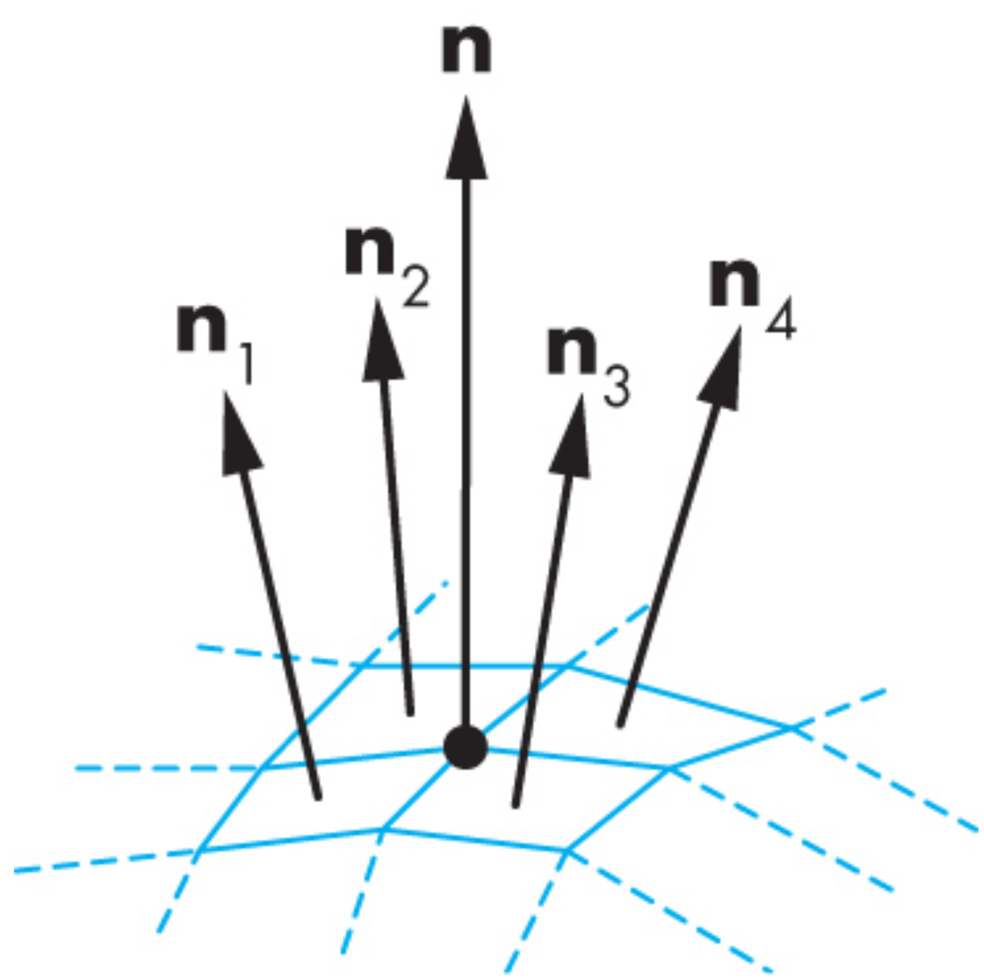
Mach Band Effect



Flat shading doesn't usually look too good.
The **lateral inhibition** effect makes flat shading seem even worse.

Smooth Shading

$$\mathbf{n} = \frac{\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4}{||\mathbf{n}_1 + \mathbf{n}_2 + \mathbf{n}_3 + \mathbf{n}_4||}$$

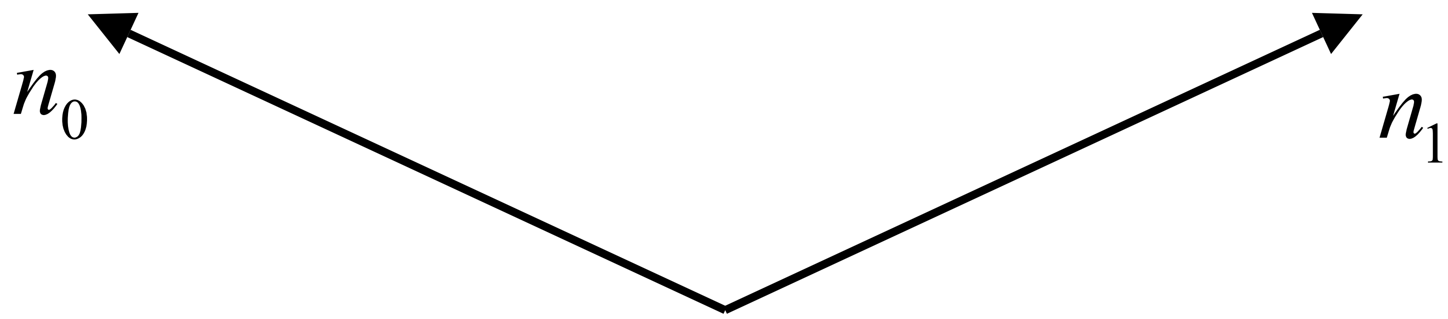


do the shading calculation once per **vertex**

We assign the vertex normals based on the surrounding polygon normals

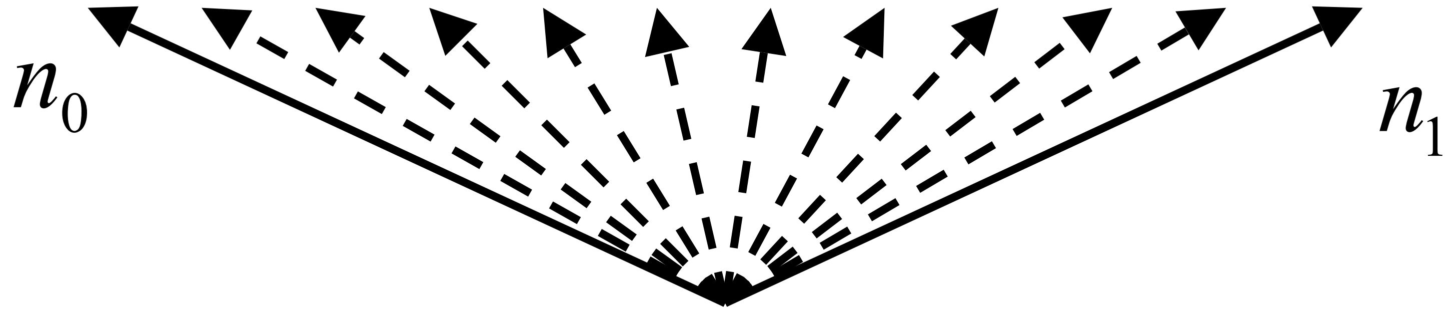
Interpolating Normals

- Must renormalize



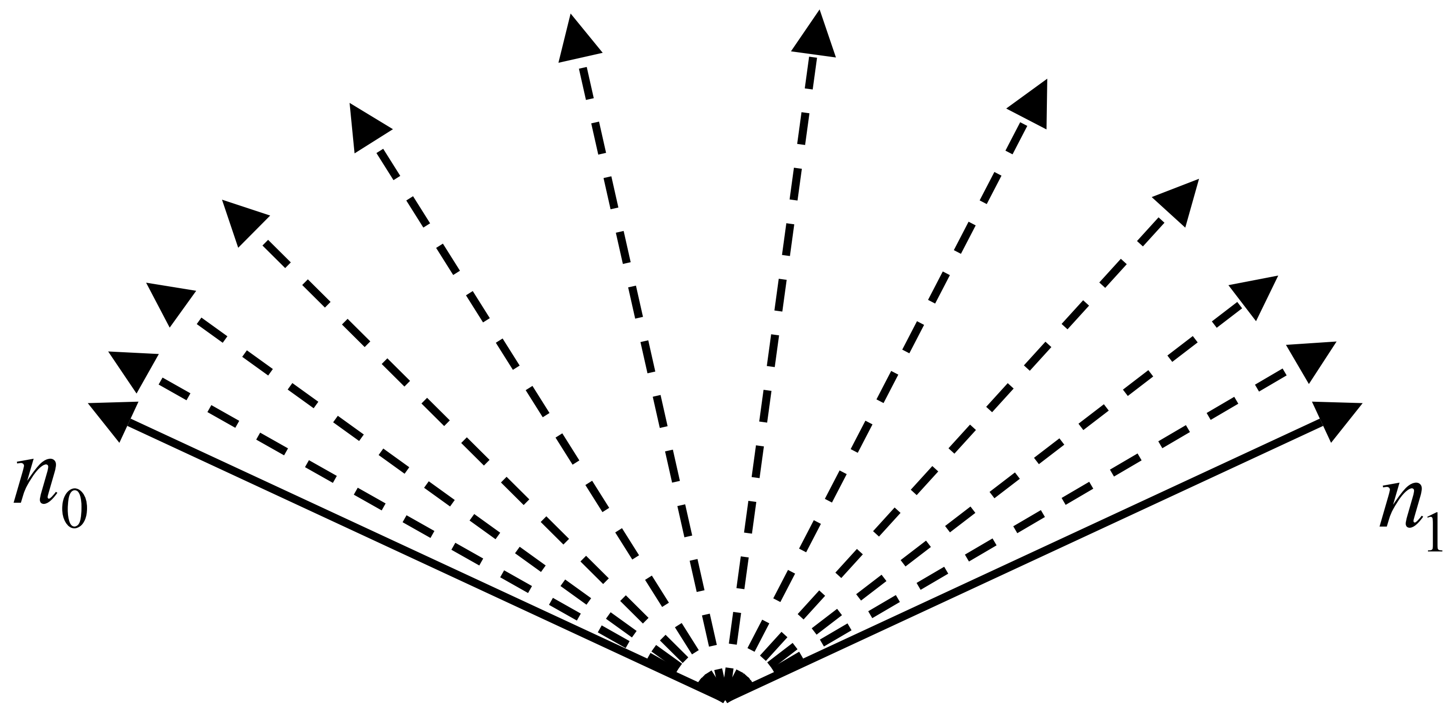
Interpolating Normals

- Must renormalize

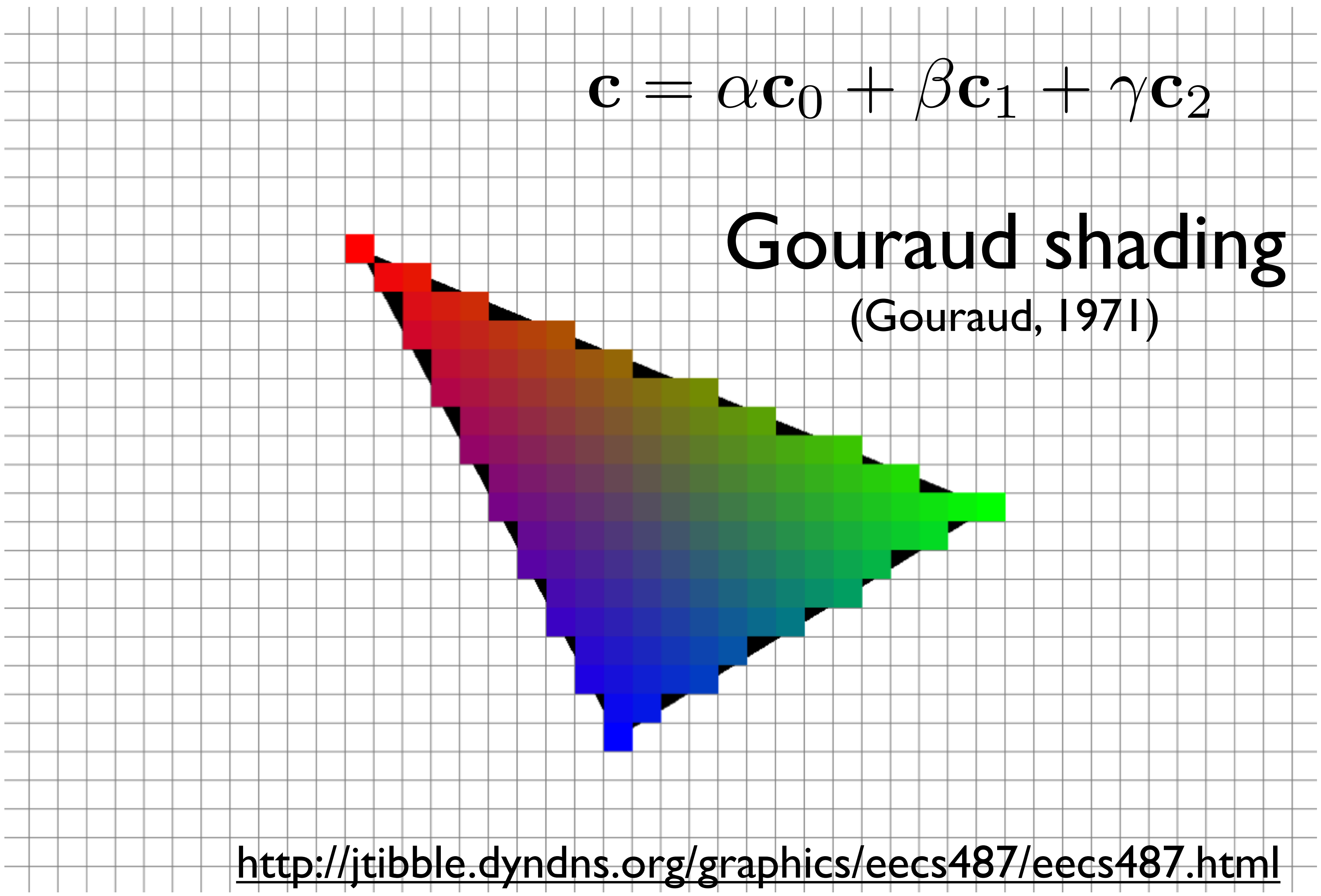


Interpolating Normals

- Must renormalize



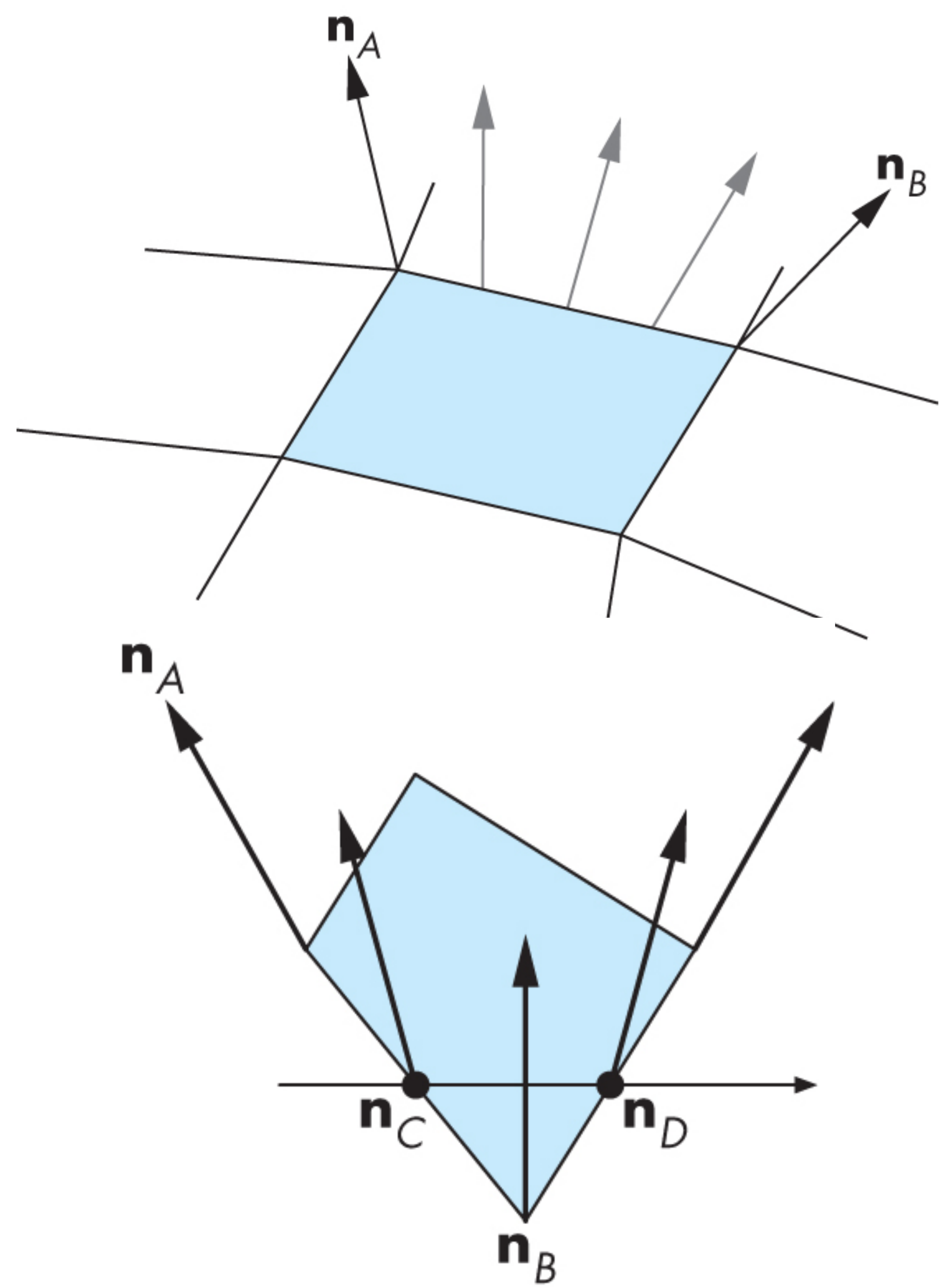
We can interpolate attributes using barycentric coordinates



Using barycentric coordinates also has the advantage that we can easily interpolate colors or other attributes from triangle vertices

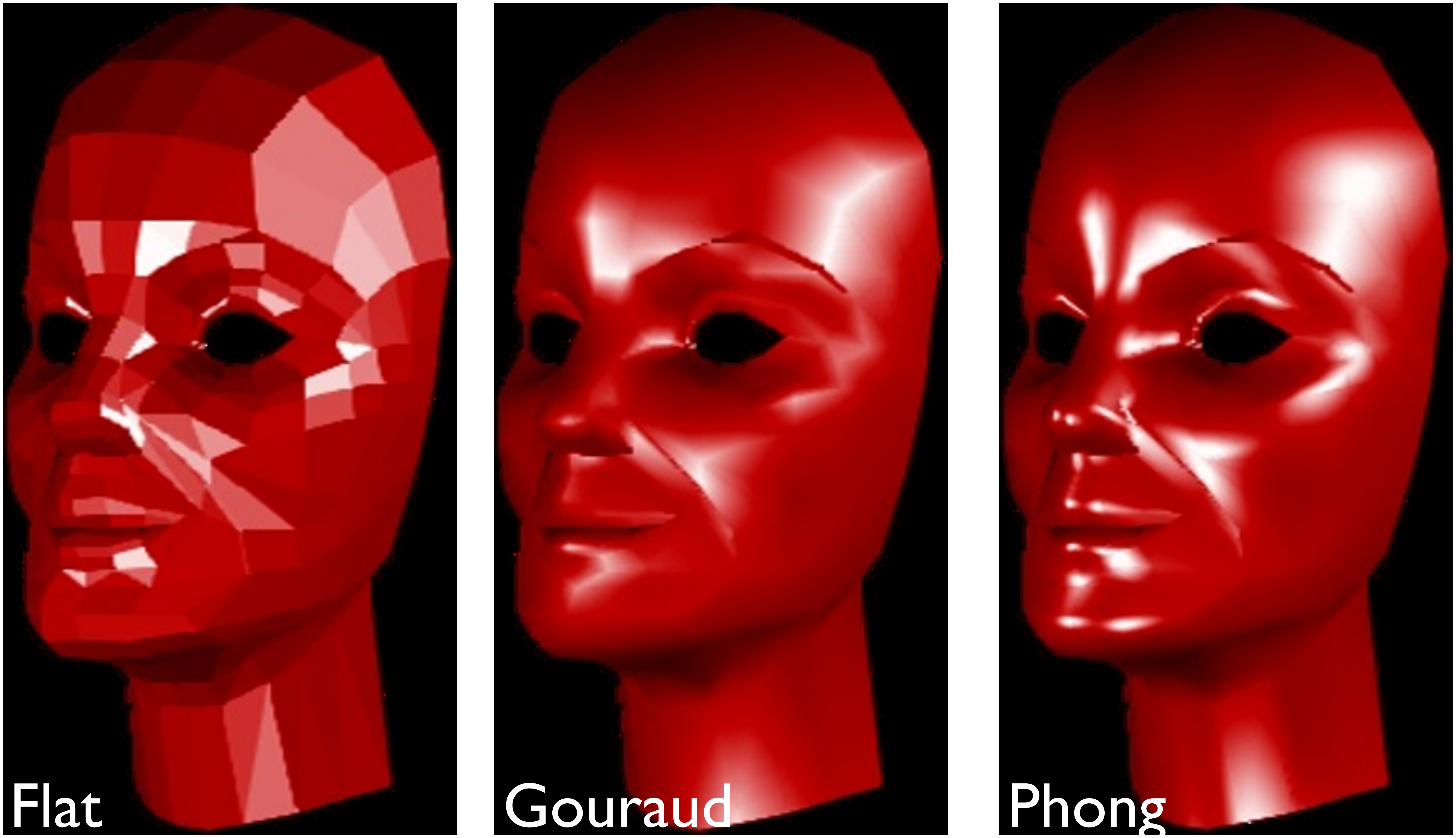
Phong Shading

do the shading calculation once per **fragment**



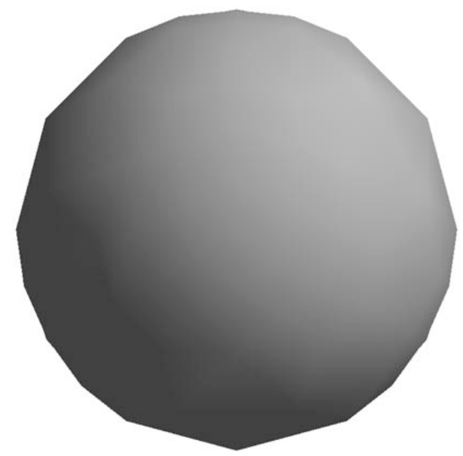
Phong shading requires normals to be interpolated across each polygon -- this wasn't part of the fixed function pipeline.
This can now be done in the pipeline in the fragment shader.

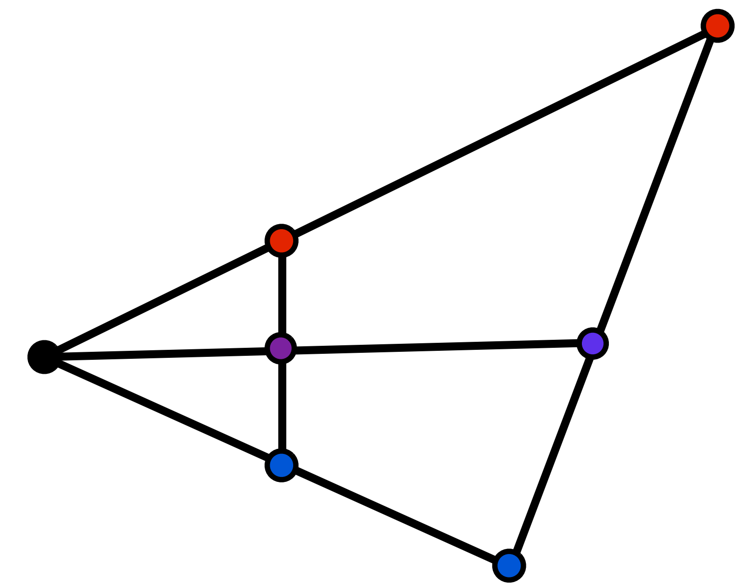
Comparison



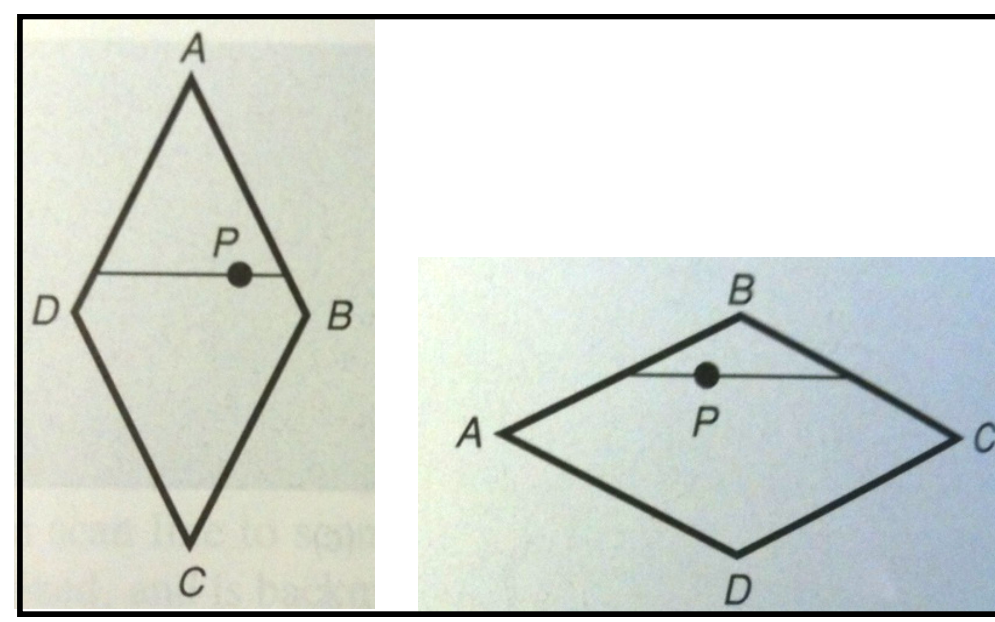
- Phong interpolation looks smoother -- can see edges on the Gouraud model
- but Phong is a lot more work
- both Phong and Gouraud require vertex normals
- both Phong and Gouraud leave silhouettes

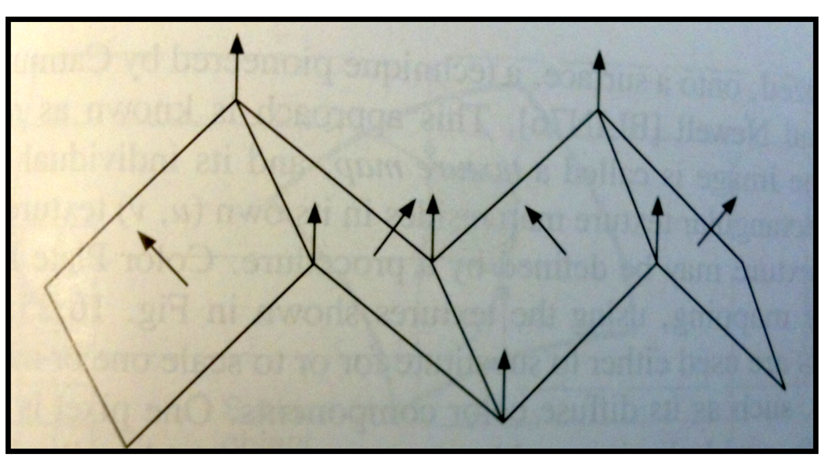
Problems with Interpolated Shading





- Polygonal silhouette
- Perspective distortion
- Orientation dependence
- Unrepresentative surface normals

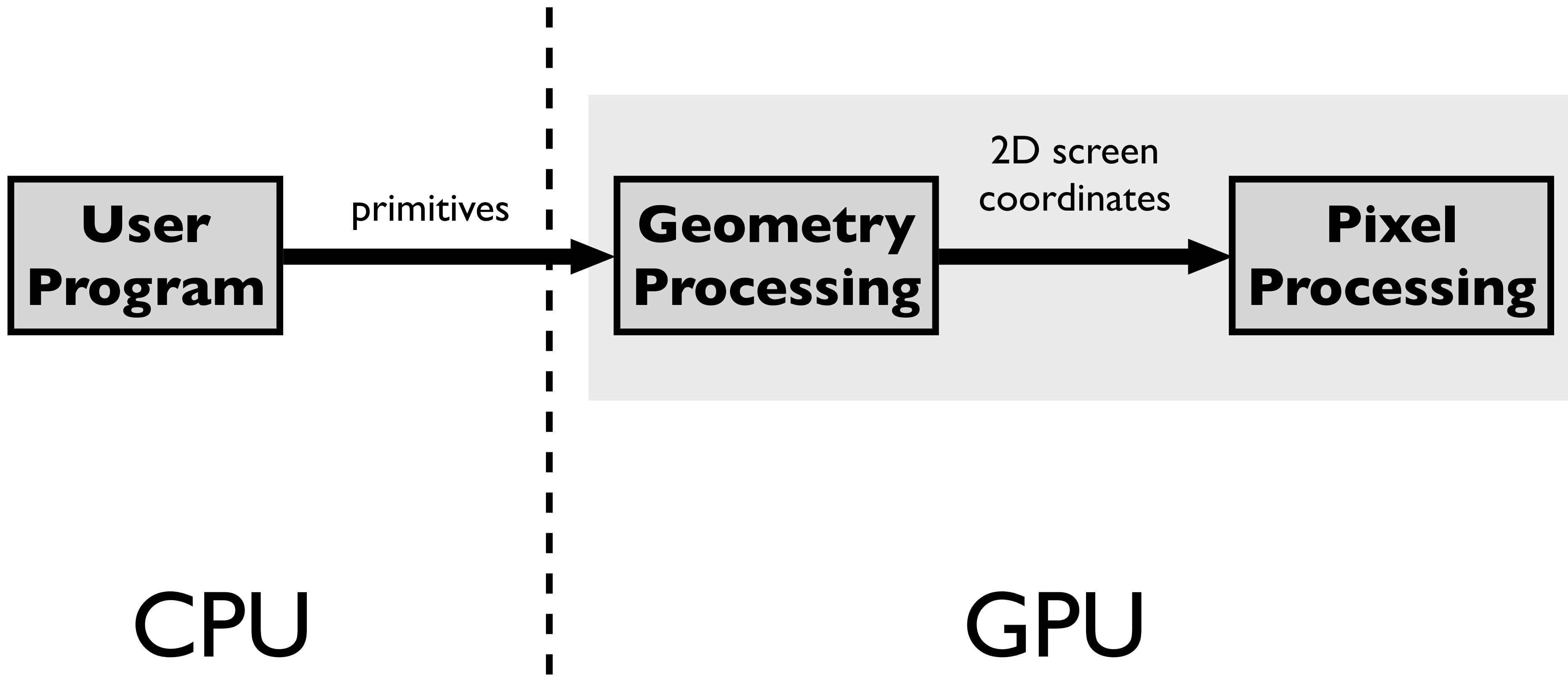




[Foley, van Dam, Feiner, Hughes]

Programmable Shading

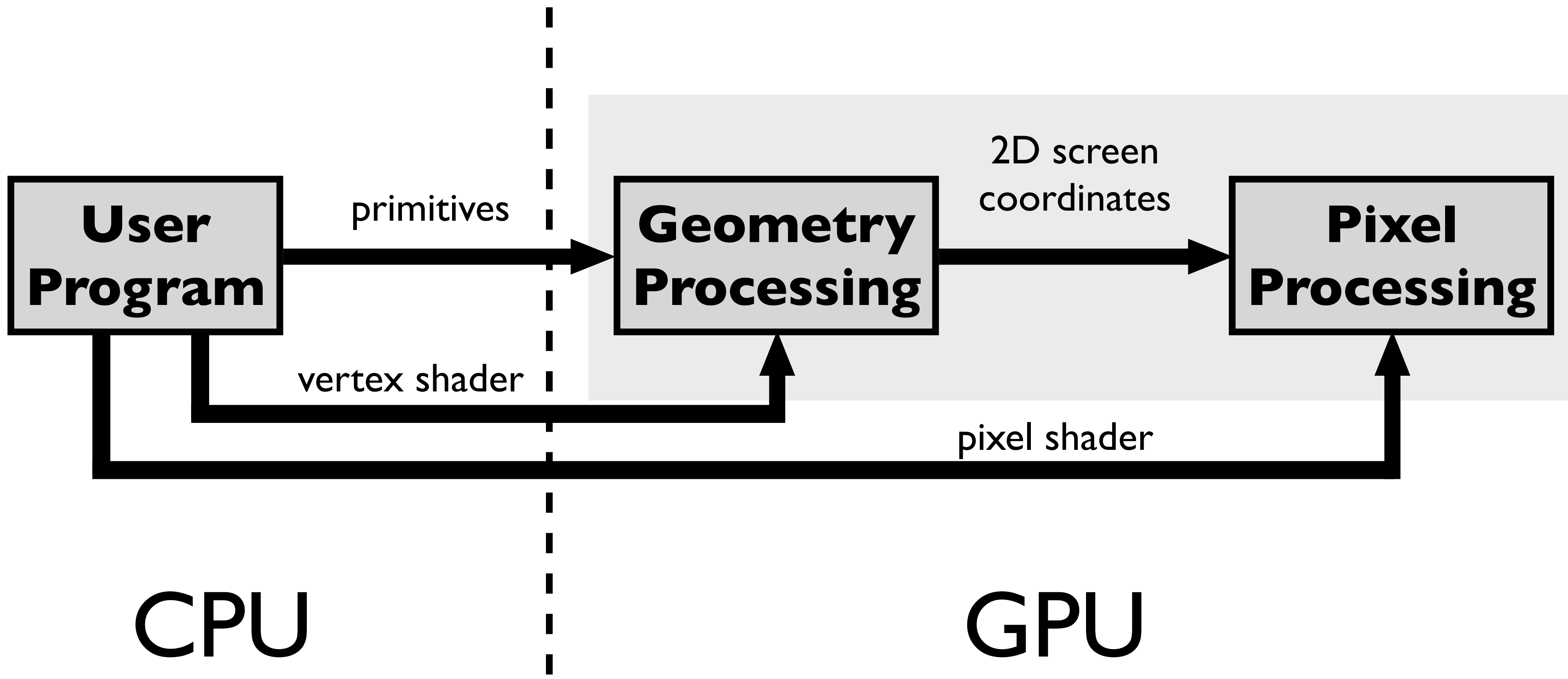
Fixed-Function Pipeline



Control pipeline through GL state variables

- The application supplies geometric primitives through a graphics API such as **OpenGL** or **DirectX**
- control of pipeline operation through state variables only

Programmable Pipeline



Supply shader programs to be executed on GPU as part of pipeline

- can supply shader programs to carry out vertex processing, geometry processing, and pixel processing

Phong reflectance in vertex and pixel shaders using GLSL

```
void main(void)
{
    vec4 v = gl_modelView_Matrix * gl_Vertex;
    vec3 n = normalize(gl_NormalMatrix * gl_Normal);
    vec3 l = normalize(gl_lightSource[0].position - v);
    vec3 h = normalize(l - normalize(v));

    float p = 16;
    vec4 cr = gl_FrontMaterial.diffuse;
    vec4 cl = fl_LightSource[0].diffuse;
    vec4 ca - vec4(0.2, 0.2, 0.2, 1.0);

    vec4 color;
    if (dot(h,n) > 0)
        color = cr * (ca + cl * max(0,dot(,n,l)))
            + cl* pow(dot(h,n), p);
    else
        color = cr * (ca + cl * max(0,dot(,n,l)));

    gl_FrontColor = color;
    gl_Position = ftransform();
}
```

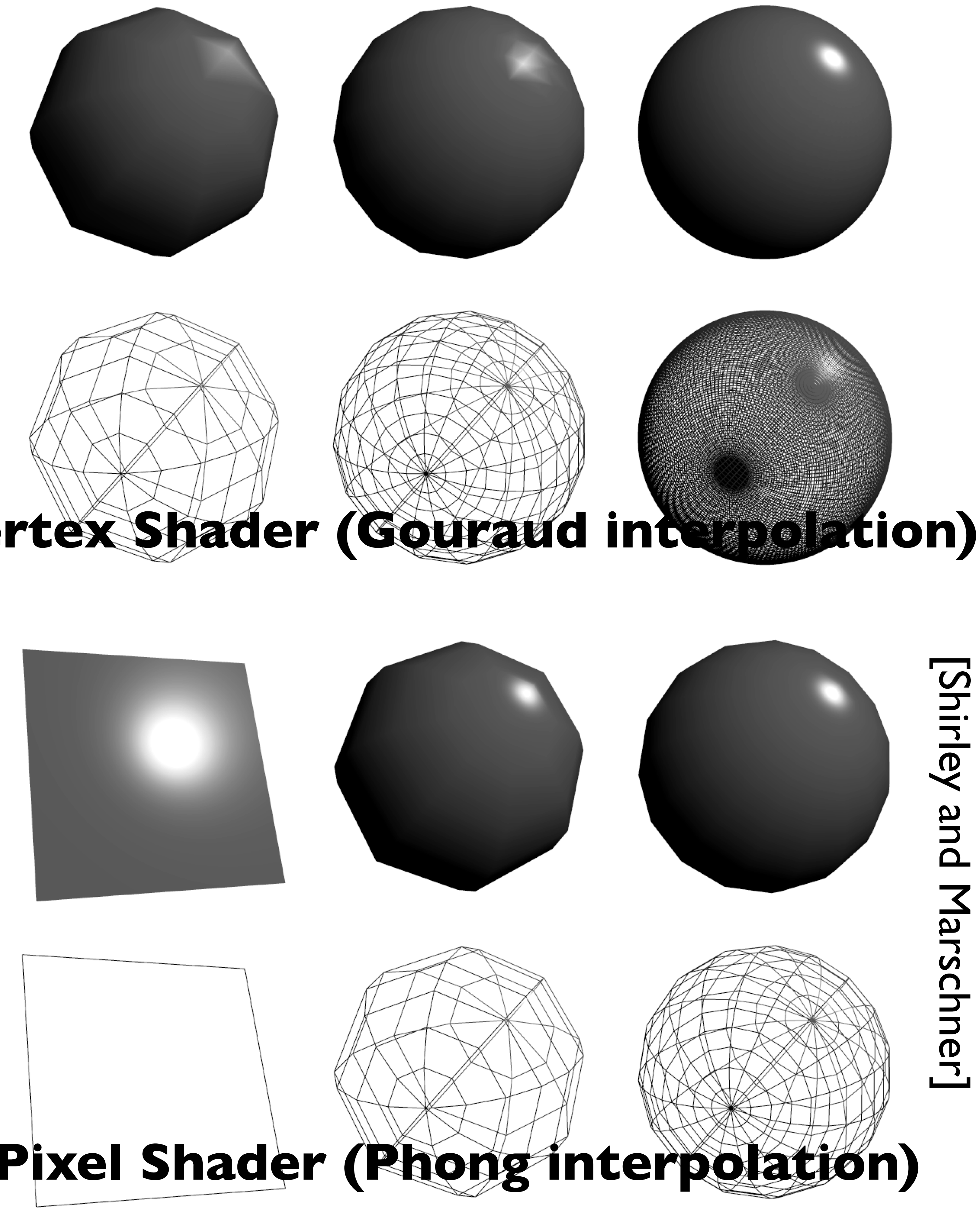
```
varying vec4 v;
varying vec3 n;

void main(void)
{
    vec3 l = normalize(gl_lightSource[0].position - v);
    vec3 h = normalize(l - normalize(v));

    float p = 16;
    vec4 cr = gl_FrontMaterial.diffuse;
    vec4 cl = fl_LightSource[0].diffuse;
    vec4 ca - vec4(0.2, 0.2, 0.2, 1.0);

    vec4 color;
    if (dot(h,n) > 0)
        color = cr * (ca + cl * max(0,dot(,n,l)))
            + cl* pow(dot(h,n), p);
    else
        color = cr * (ca + cl * max(0,dot(,n,l)));

    gl_FragColor = color;
}
```

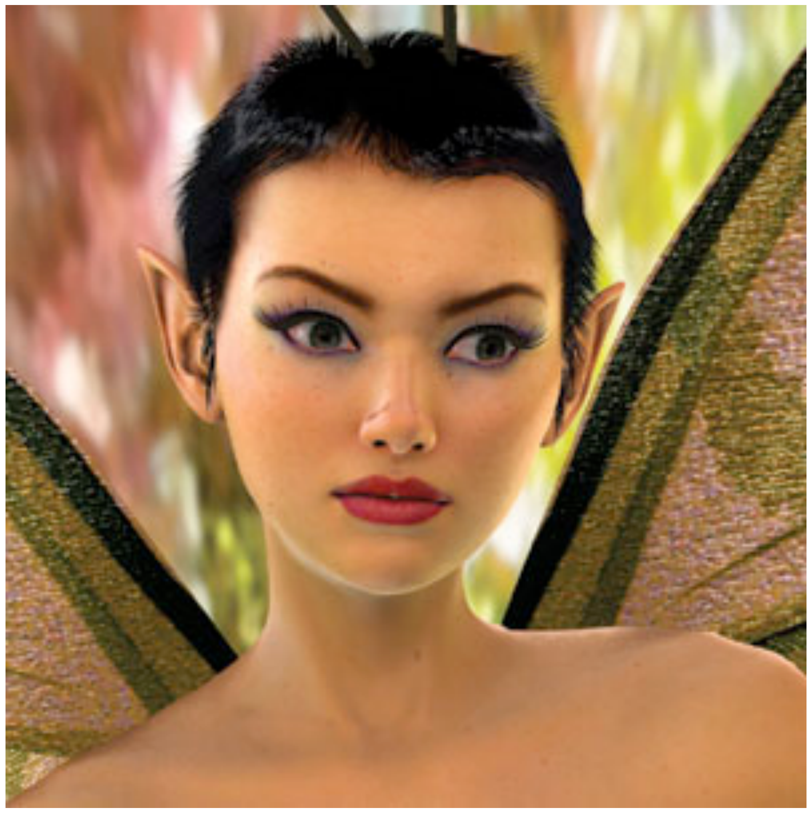


Phong reflectance as a vertex shader
- vertex shaders can be used to move/animate verts
- linear interpolation of vertex lighting

as a fragment shader
- each fragment is calculated individually - don't know about neighboring pixles

Call of Juarez DX10 Benchmark, ATI

Rusty car shader, NVIDIA



Dawn, NVIDIA

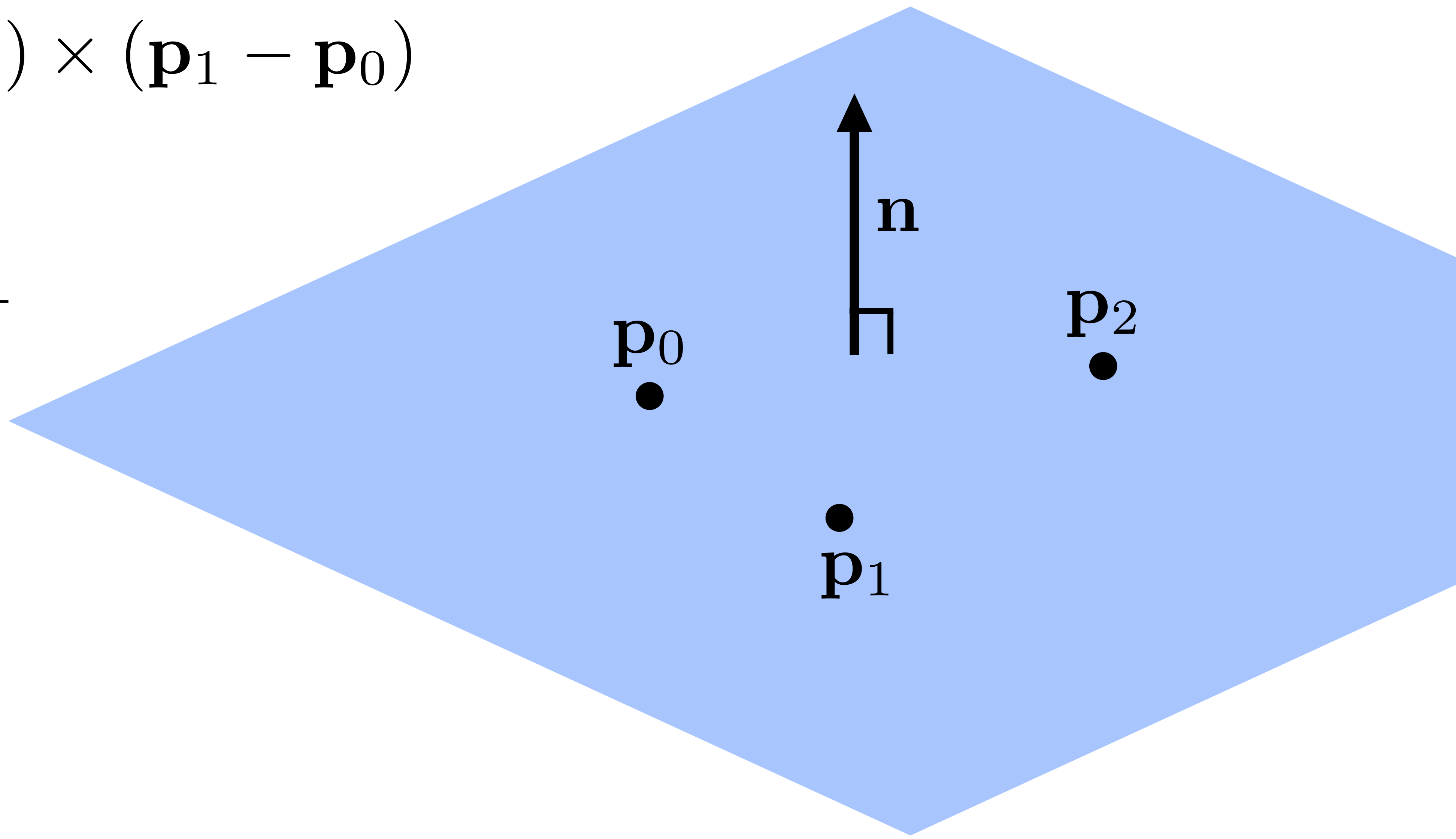
Programmable shader examples from NVIDIA and ATI

Computing Normal Vectors

Plane Normals

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

$$\mathbf{n} = \frac{\mathbf{v}}{||\mathbf{v}||}$$



Implicit function normals

$$f(\mathbf{p}) = 0$$

$$\nabla f(\mathbf{p})$$

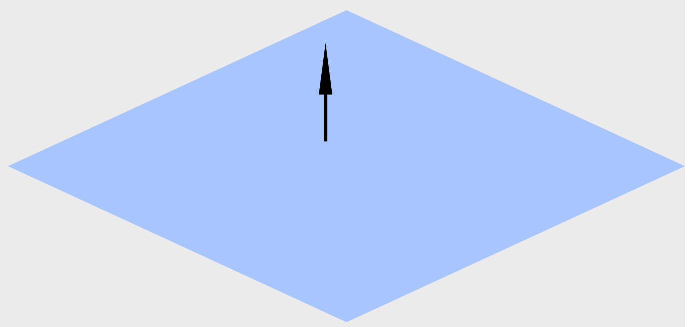
$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{pmatrix}$$

sphere

$$\mathbf{p} \cdot \mathbf{p} - r^2 = 0$$

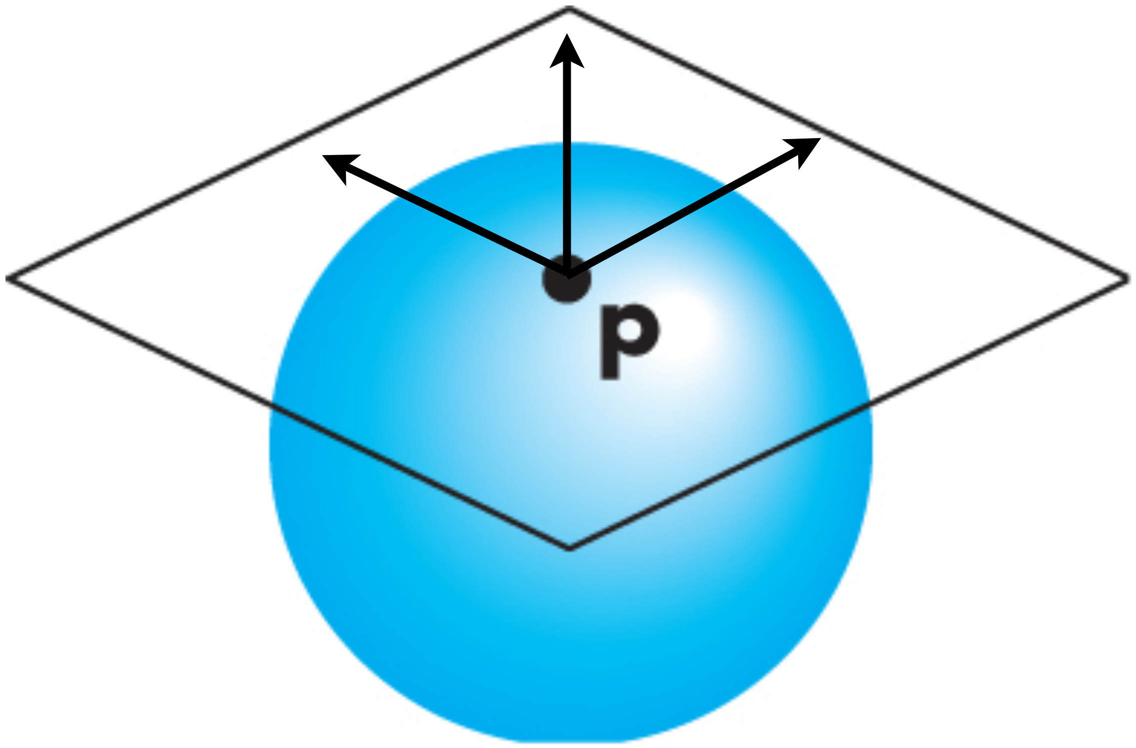
plane

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



Parametric form

$$\mathbf{p}(u,v) = \left(\begin{array}{c} x(u,v) \\ y(u,v) \\ z(u,v) \end{array} \right)$$



tangent vectors $\quad \dfrac{\partial \mathbf{p}}{\partial u} \quad \dfrac{\partial \mathbf{p}}{\partial v}$

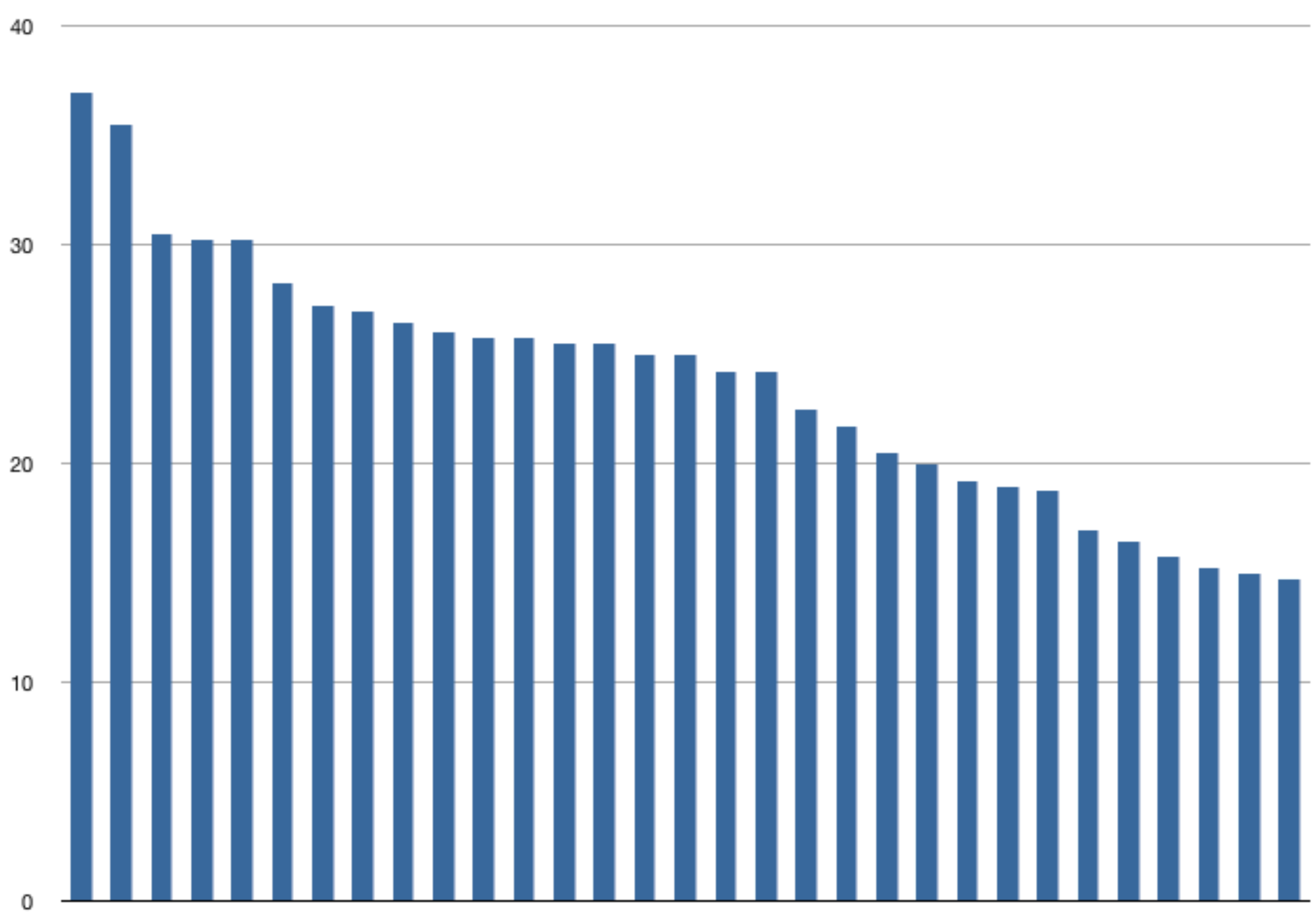
normal $\quad \dfrac{\frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial v}}{||\frac{\partial \mathbf{p}}{\partial u} \times \frac{\partial \mathbf{p}}{\partial v}||}$

Column	Exam ()
Points Possible	40
Description	

Statistics	
Count	31
Minimum Value	14.75
Maximum Value	37.00
Range	22.25
Average	23.73
Median	25.00
Standard Deviation	5.70
Variance	32.49

Status Distribution	
Null	2
In Progress	0
Needs Grading	0
Exempt	0

Grade Distribution	
greater than 100	0
90 - 100	1
80 - 89	1
70 - 79	4
60 - 69	12
50 - 59	4
40 - 49	5
30 - 39	4
20 - 29	0
10 - 19	0
0 - 9	0
less than 0	0



Exam I