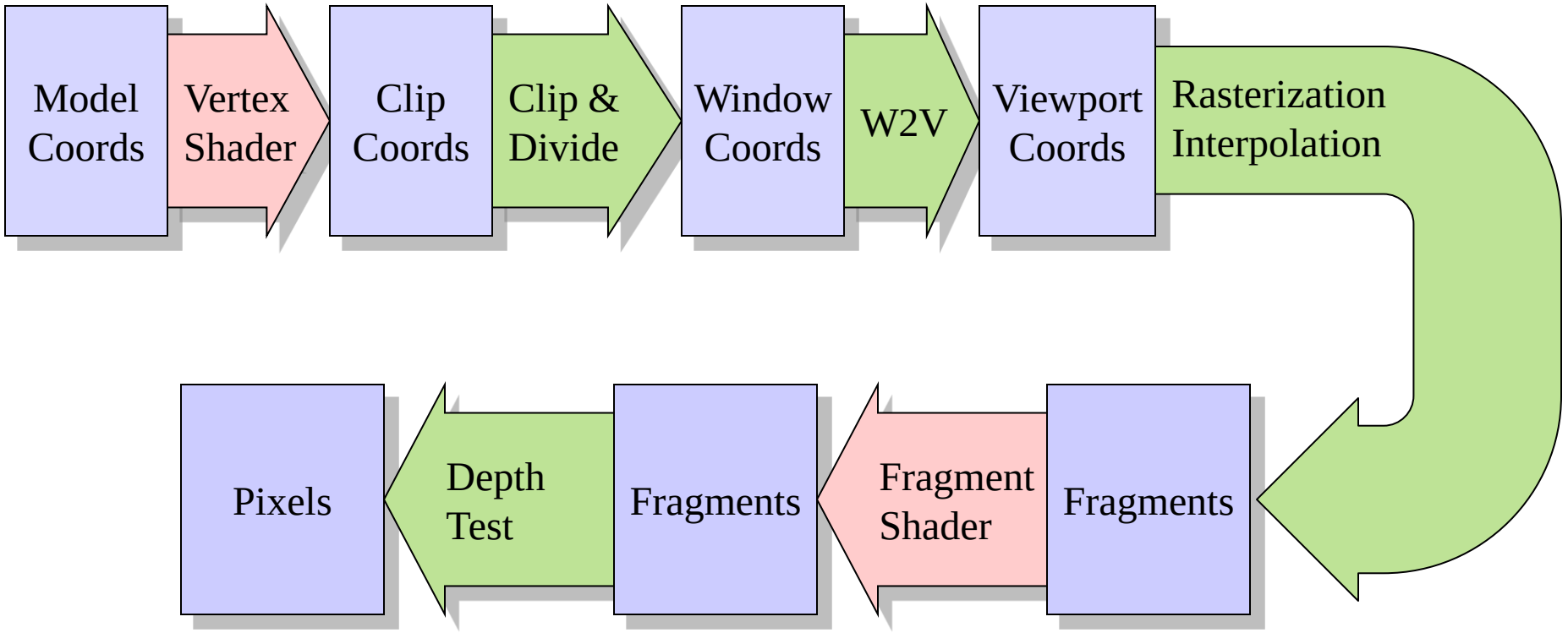
The Fragment Shader

CS418 Computer Graphics

John C. Hart

Fragment Pipeline



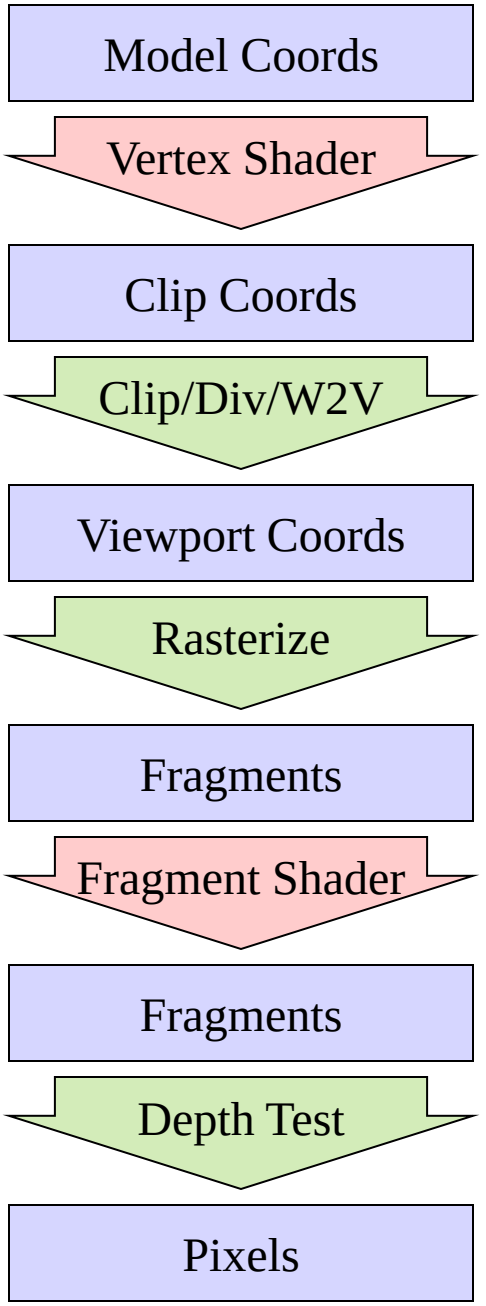
GLSL Fragment Shader

Inputs:

- vec4 gl_FragCoord (viewport coordinates)
- bool gl_FrontFacing
- vec4 gl_Color, gl_SecondaryColor
- vec4 gl_TexCoord[gl_MaxTextureCoords]
- float gl_FogFragCoord

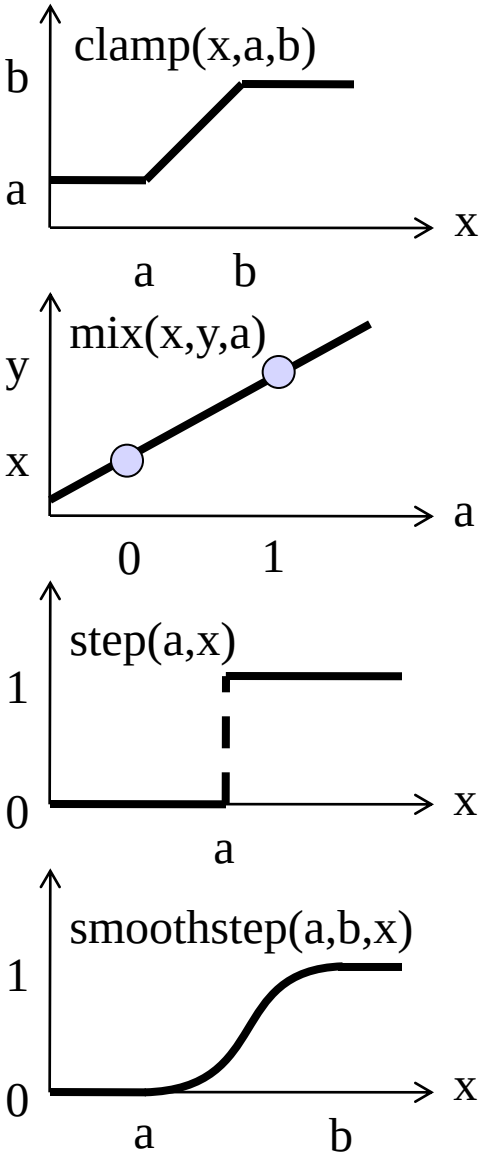
Outputs:

- vec4 gl_FragColor
- vec4 gl_FragData[gl_MaxDrawBuffers]
- float gl_FragDepth (= glFragCoord.z)



GLSL Functions

- sin, cos, tan, asin, acos, atan, atan(n,d)
- radians(deg), degrees(rads)
- pow, exp, log, sqrt
- exp2, log2, inversesqrt(x)
- abs, ceil, floor, fract, max, min, mod, sign,
- clamp(x,a,b) = min(max(x,a),b)
- mix(x,y,a) = (1.-a)*x + a*y
- step(a,x) = (x < a) ? 0.0 : 1.0
- smoothstep(a,b,x) = 0.0 if x < a; 1.0 if x > b; else t = clamp((x-a)/(b-a),0,1), t*t*(3-2*t)

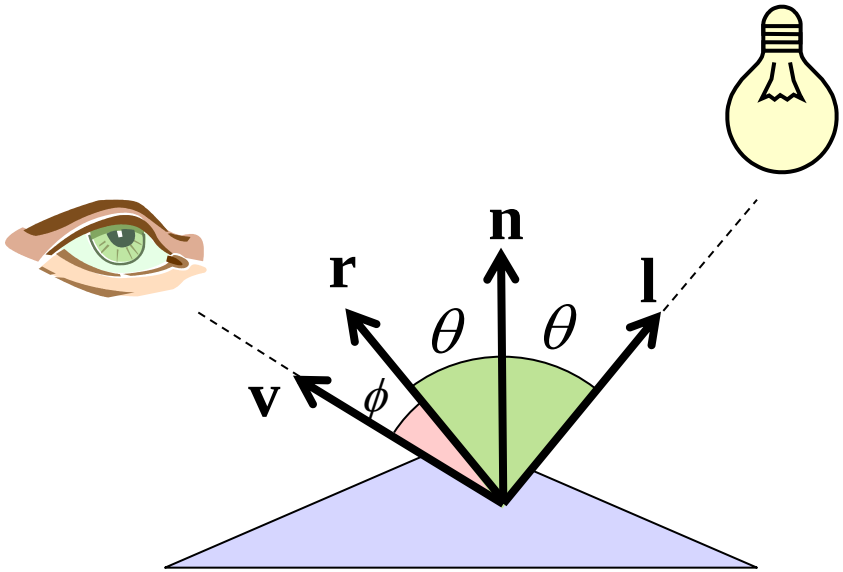


GLSL Vector Math

- dot(a,b) = a.x*b.x + a.y*b.y + a.z*b.z + …
- length(a) = sqrt(dot(a,a))
- distance(a,b) = length(b – a)
- cross(a,b) = vec3(a.y*b.z – a.z*b.y, …)
- normalize(a) = a/length(a) = a*inversesqrt(dot(a,a))
- faceforward(n,v,nref) = dot(nref,v) < 0.0 ? -n : n
- reflect(l,n) = l – 2*dot(l,n)*n
- refract(l,n,eta)
  - refracts a vector l through a surface w/normal n and index of refraction eta
  - derivation in CS419 Production Graphics

Specular Reflection

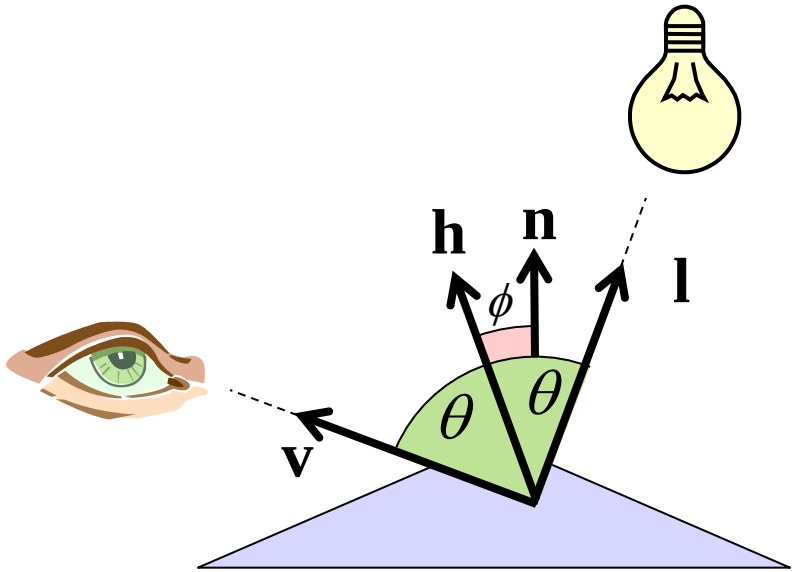
Phong



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

$$L_o = L_i\, k_s\, \mathbf{c}_s \cos^n \phi$$
$$= L_i\, k_s\, \mathbf{c}_s\, (\mathbf{v}\cdot\mathbf{r})^n$$

Blinn



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

$$L_o = L_i\, k_s\, \mathbf{c}_s \cos^n \phi$$
$$= L_i\, k_s\, \mathbf{c}_s\, (\mathbf{n}\cdot\mathbf{h})^n$$

Blinn Vertex Shader

```
void main() {
  /* compute unit normal, vertex position, light vector and
     view vector in viewing coordinates */
  vec3 n = normalize(gl_NormalMatrix*gl_Normal);
  vec3 p = gl_ModelViewMatrix*gl_Vertex;
  vec3 l = normalize(gl_LightSource[0].position - p.xyz);
  vec3 v = -normalize(p);          /* eye is at origin */
  /* compute halfway vector */
  vec3 h = normalize(l + v);
  /* initialize color with reflection of ambient light */
  frontColor = gl_FrontMaterial.ambient*gl_LightSource[0].ambient;
  /* f indicates if vertex faces light (f=1) or on dark side (f=0) */
  float f = (dot(n,l) > 0.0) ? 1.0 : 0.0;
  /* add Lambertian diffuse reflection of direct light */
  frontColor += f*dot(n,l)*gl_FrontMaterial.diffuse*gl_LightSource[0].diffuse;
  /* add Blinn specular reflection of direct light */
  frontColor += f*pow(dot(n,h),gl_FrontMaterial.shininess) *
                gl_FrontMaterial.specular*gl_LightSource[0].specular;
  gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;
}
```

Blinn Fragment Shader

```
varying vec3 n; varying vec4 p;
void main() {
  n = gl_NormalMatrix*gl_Normal;
  p = gl_ModelViewMatrix*gl_Vertex;
  gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;
}
```

Vertex Shader

```
varying vec3 n; varying vec4 p;
void main() {
  vec3 nhat = normalize(n);
  vec3 l = normalize(gl_LightSource[0].position - p.xyz);
  vec3 v = -normalize(p);          /* eye is at origin */
  vec3 h = normalize(l + v);
  vec4 c = gl_FrontMaterial.ambient*gl_LightSource[0].ambient;
  c += max(0,dot(n,l))*gl_FrontMaterial.diffuse*gl_LightSource[0].diffuse;
  int f;
  if (dot(n,l) > 0.0)
     c += pow(max(0,dot(n,h)),gl_FrontMaterial.shininess) *
                gl_FrontMaterial.specular*gl_LightSource[0].specular;
  gl_FragColor = c;
}
```