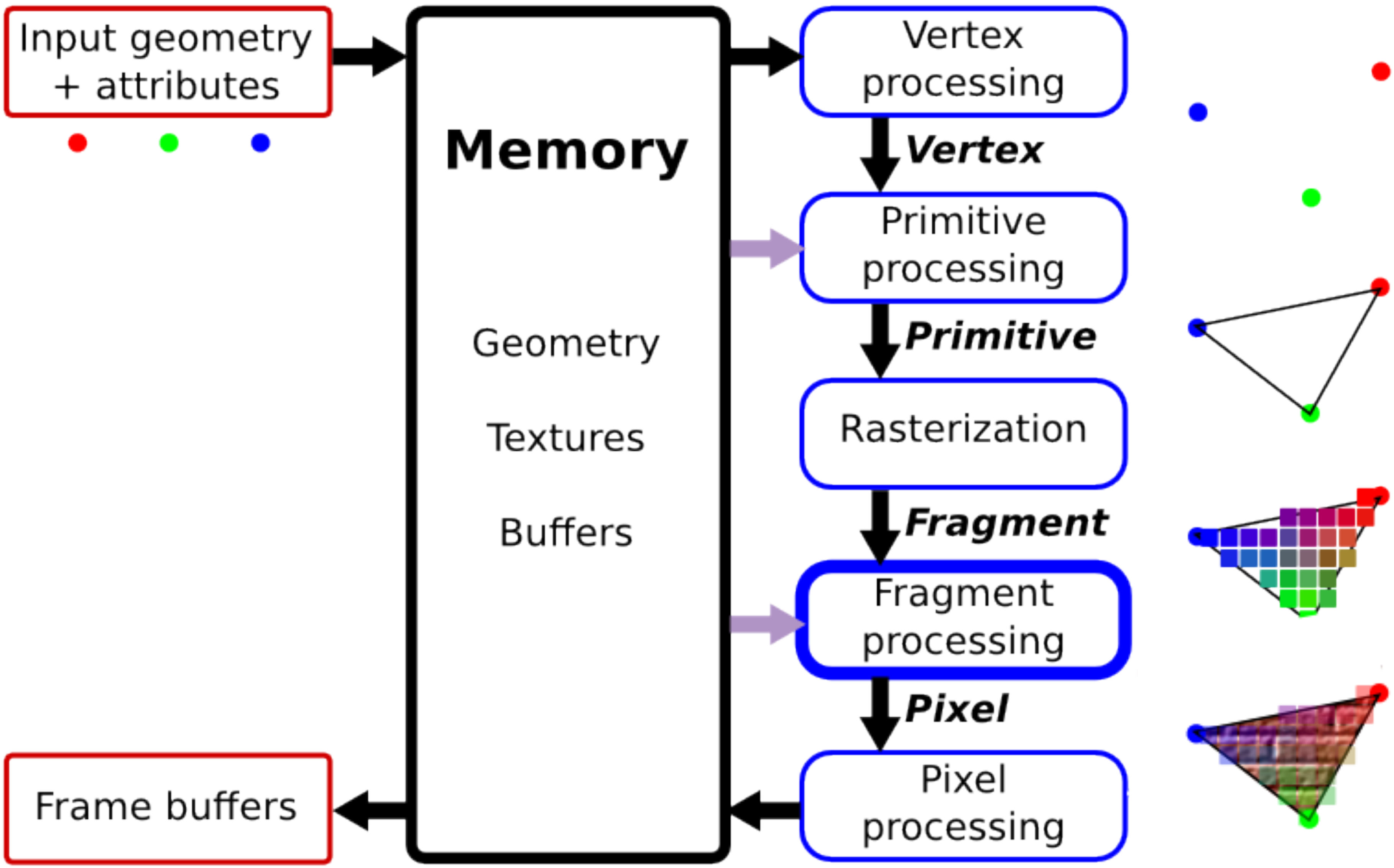
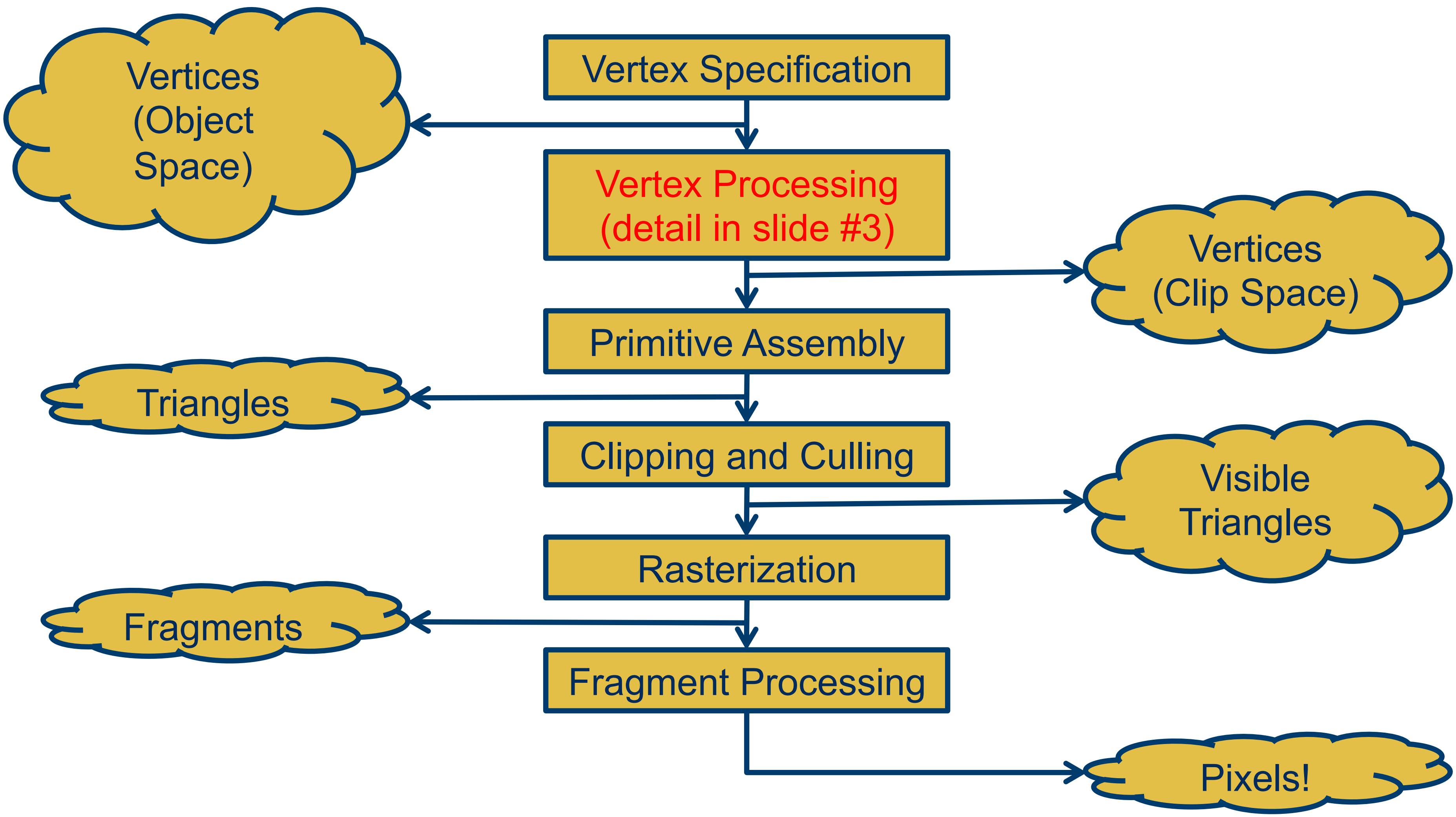
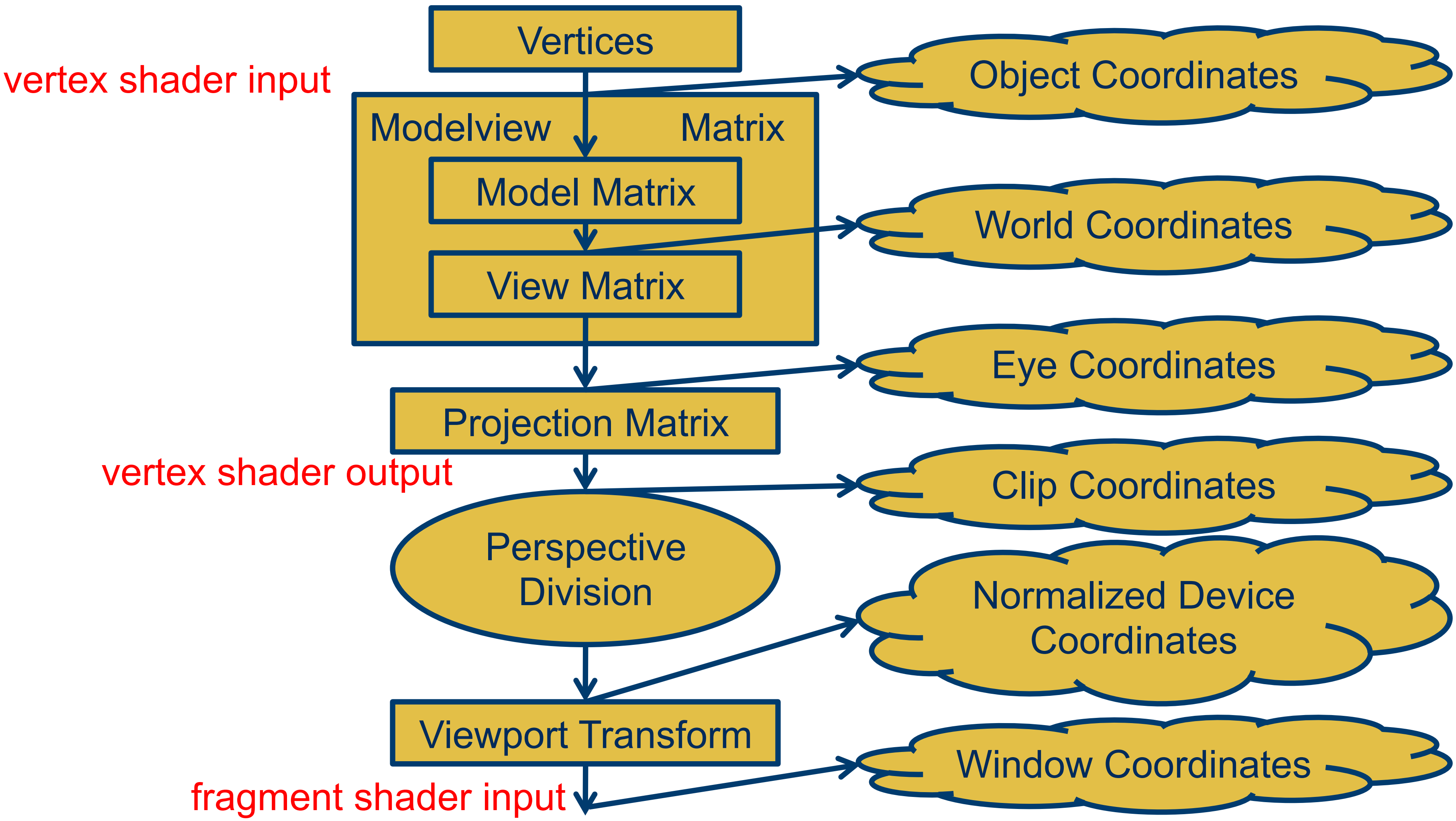

Input geometry + attributes
Memory
Geometry
Textures
Buffers
Frame buffers
Vertex processing
Vertex
Primitive processing
Primitive
Rasterization
Fragment
Fragment processing
Pixel
Pixel processing

WebGL Rendering Pipeline



Vertex Transformation/Processing Pipeline



Shaders

- Vertex shader
  - `gl_Position = projectionMatrix * viewingMatrix * modelingMatrix * vPosition;`
  - `vPosition` goes from object space to world space (`modelingMatrix`) to eye space (`viewingMatrix`) to clip space (`projectionMatrix`)
- Fragment shader
  - `gl_FragColor = color;`
  - `color` could be a constant color, vertex color interpolated over fragment, modulated by lighting and/or texturing colors

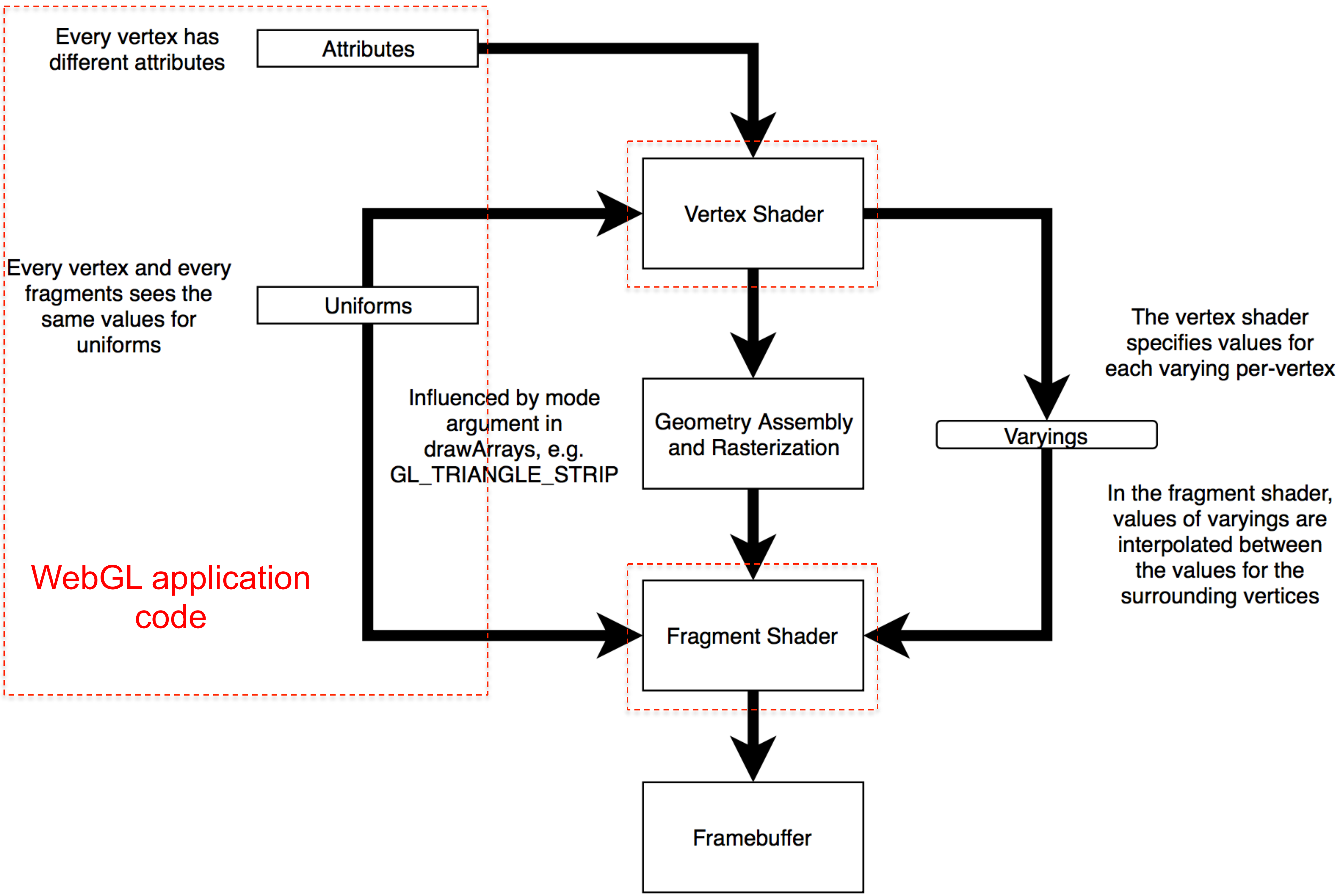
Sending vertex data to GPU

```
// Load the data into the GPU
var vBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer );
gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices),
   gl.STATIC_DRAW );

// Associate shader variables with data buffer
var vPositionLoc = gl.getAttribLocation( program,
      "vPosition" );
gl.vertexAttribPointer( vPositionLoc, 3, gl.FLOAT,
         false, 0, 0 );
gl.enableVertexAttribArray( vPositionLoc );
```

Shader Qualifiers

- `attribute`
  - Variables can change at most once per vertex
  - `vPosition` (must have)
  - `vColor, vNormal, vTexCoord` (optional)
- `uniform`
  - Variables that are constant for an entire primitive
  - Can be changed in application and sent to shaders
- `varying`
  - Variables that are passed from vertex shader to fragment shader
  - Automatically interpolated by the rasterizer


Every vertex has different attributes
Attributes
Vertex Shader
Every vertex and every fragments sees the same values for uniforms
Uniforms
The vertex shader specifies values for each varying per-vertex
Influenced by mode argument in drawArrays, e.g. GL_TRIANGLE_STRIP
Geometry Assembly and Rasterization
Varyings
In the fragment shader, values of varyings are interpolated between the values for the surrounding vertices
WebGL application code
Fragment Shader
Framebuffer

Modeling – Translation Matrix

- We can also express translation using a 4 × 4 matrix $\mathbf{T}$ in homogeneous coordinates $\mathbf{p}' = \mathbf{T}\mathbf{p}$ where

$$\mathbf{T} = \mathbf{T}(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

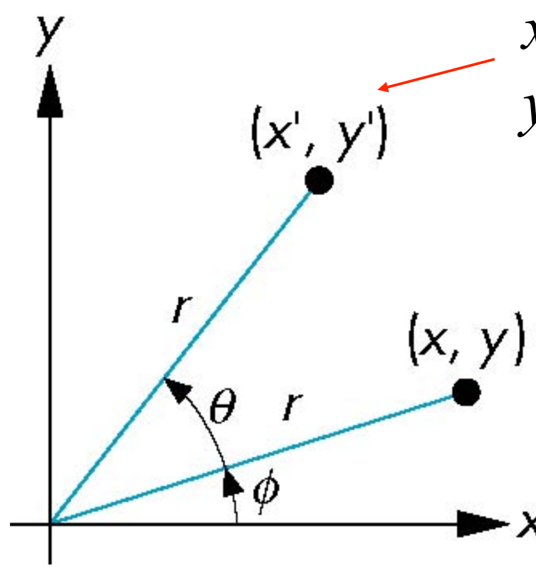
note that the translation matrix is multiplied to the left of point

- This form is better for implementation because all affine transformations can be expressed this way and multiple transformations can be concatenated together

Modeling – Rotation Matrix

$$\mathbf{R} = \mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boxed{\begin{aligned} x' &= x \cos\theta - y \sin\theta \\ y' &= x \sin\theta + y \cos\theta \end{aligned}}$$



$$x' = r \cos(\phi + \theta)$$
$$y' = r \sin(\phi + \theta)$$

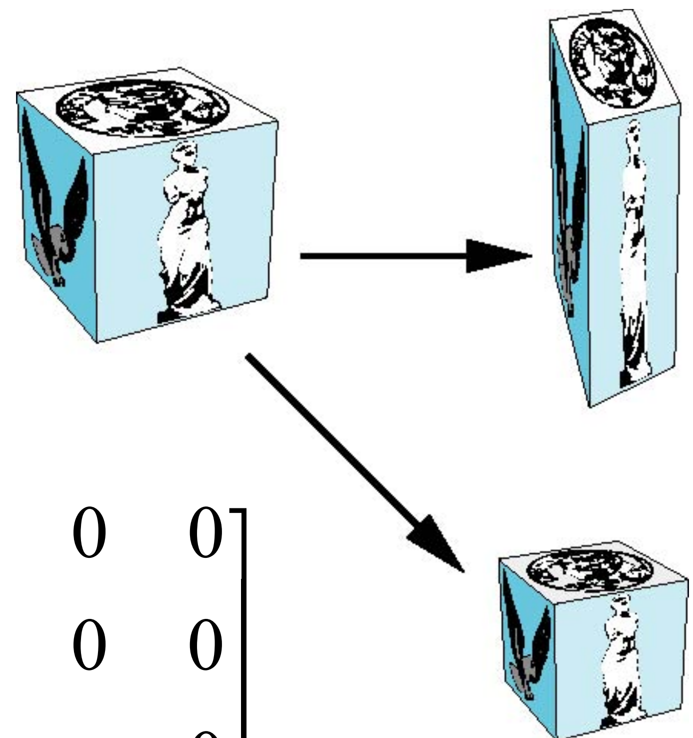
$$x = r \cos\phi$$
$$y = r \sin\phi$$

Modeling – Scaling Matrix

Expand or contract along each axis (fixed point of origin)

$$x' = s_x x$$
$$y' = s_y x$$
$$z' = s_z x$$

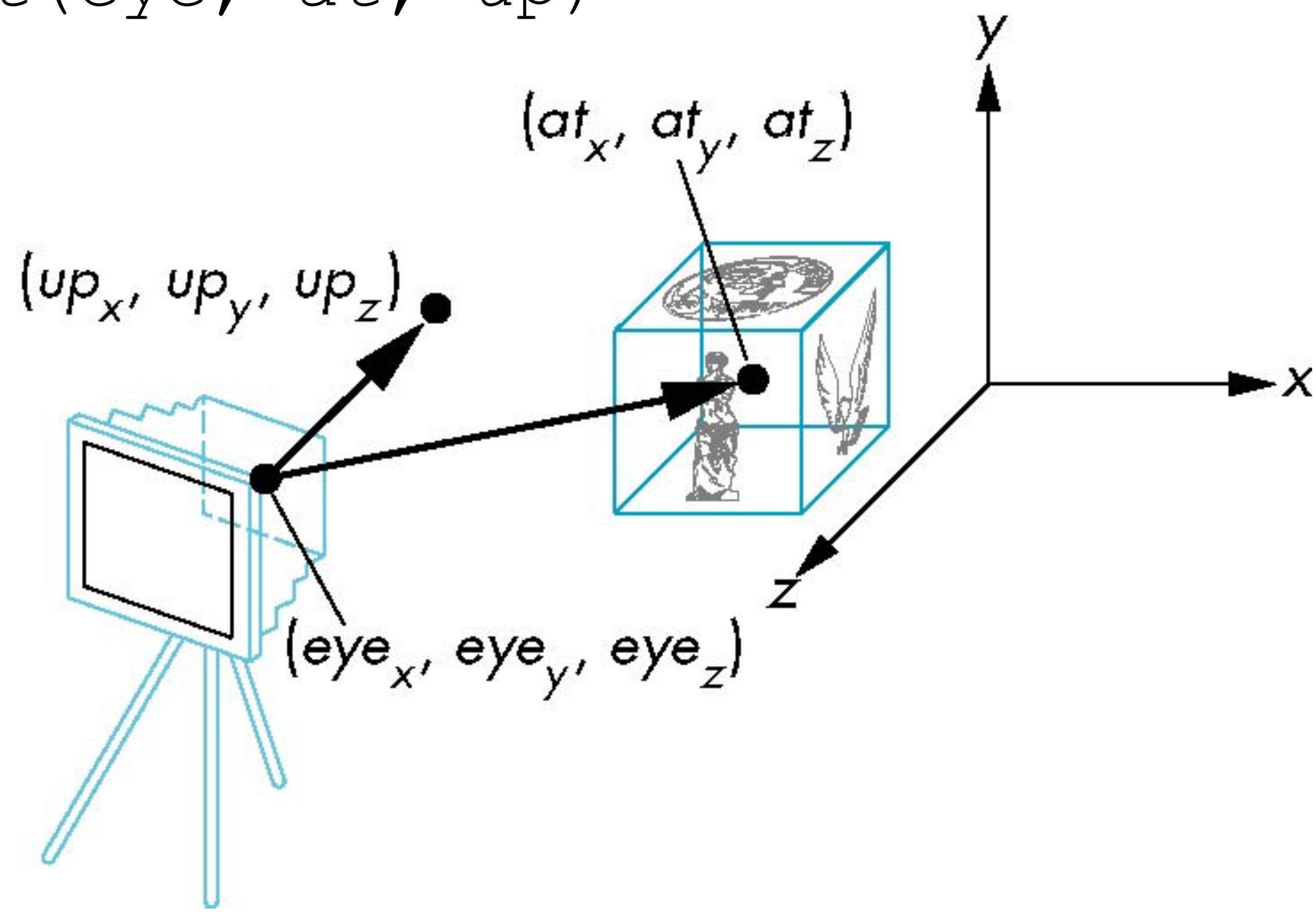
$$\mathbf{p}' = \mathbf{S}\mathbf{p}$$



$$\mathbf{S} = \mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Viewing – The `lookAt` Function

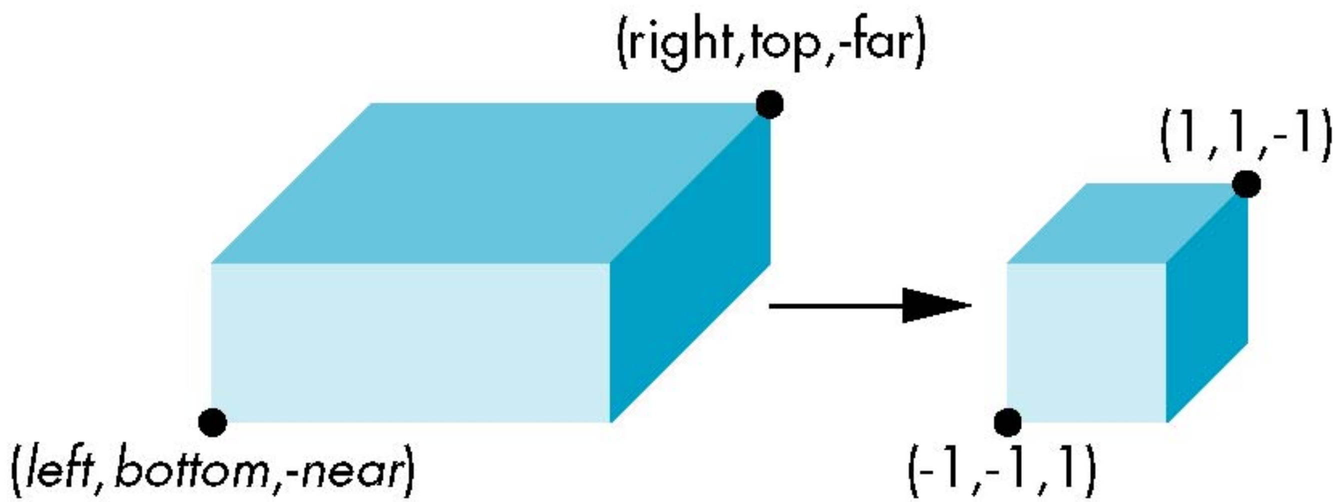
`lookAt(eye, at, up)`



Projection – Parallel

```
ortho(left,right,bottom,top,near,far);
```

View normalization ⇒ find transformation to convert specified clipping volume to default



Parallel Projection Matrix

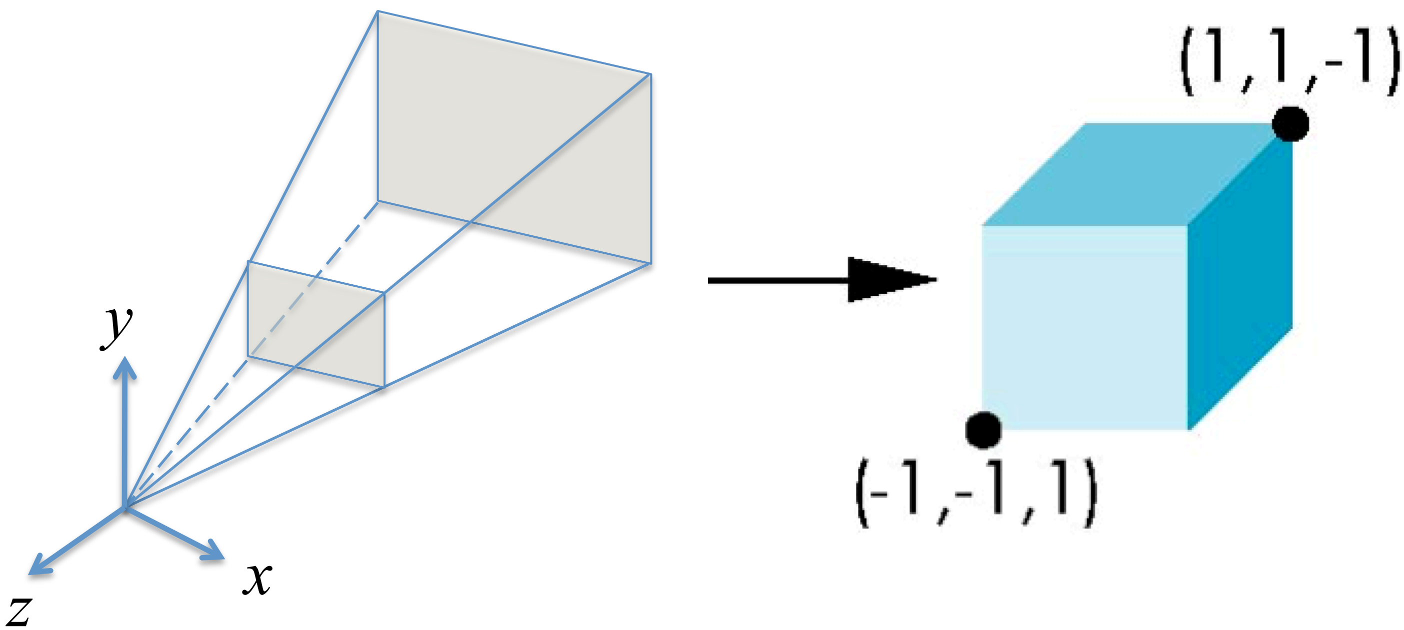
$$\mathbf{P} = \mathbf{ST} = \begin{bmatrix} \frac{2}{right - left} & 0 & 0 & 0 \\ 0 & \frac{2}{top - bottom} & 0 & 0 \\ 0 & 0 & -\frac{2}{far - near} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{right + left}{2} \\ 0 & 1 & 0 & -\frac{top + bottom}{2} \\ 0 & 0 & 1 & \frac{far + near}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{2}{right - left} & 0 & 0 & -\frac{right + left}{right - left} \\ 0 & \frac{2}{top - bottom} & 0 & -\frac{top + bottom}{top - bottom} \\ 0 & 0 & -\frac{2}{far - near} & -\frac{far + near}{far - near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projection – Perspective

```
Perspective(fov,aspect,near,far);
frustum(left,right,bottom,top,near,far);
```

View normalization ⇒ find transformation to convert specified clipping volume to default



Perspective Projection Matrix

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2 \times near}{right - left} & 0 & \frac{right + left}{right - left} & 0 \\ 0 & \frac{2 \times near}{top - bottom} & \frac{top + bottom}{top - bottom} & 0 \\ 0 & 0 & \frac{-(far + near)}{far - near} & \frac{-2 \times far \times near}{far - near} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

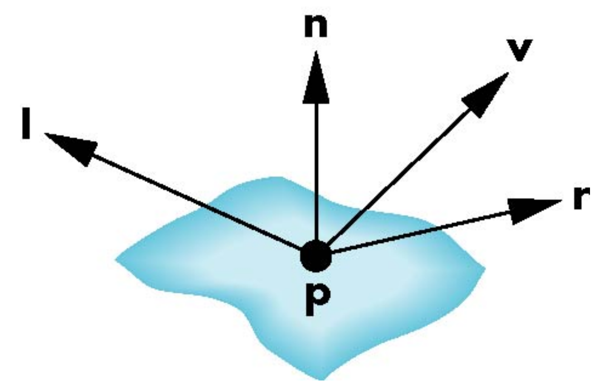
Lighting Computation

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

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

diffuse specular ambient

- For each color component we add contributions from all sources



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

Texture Mapping