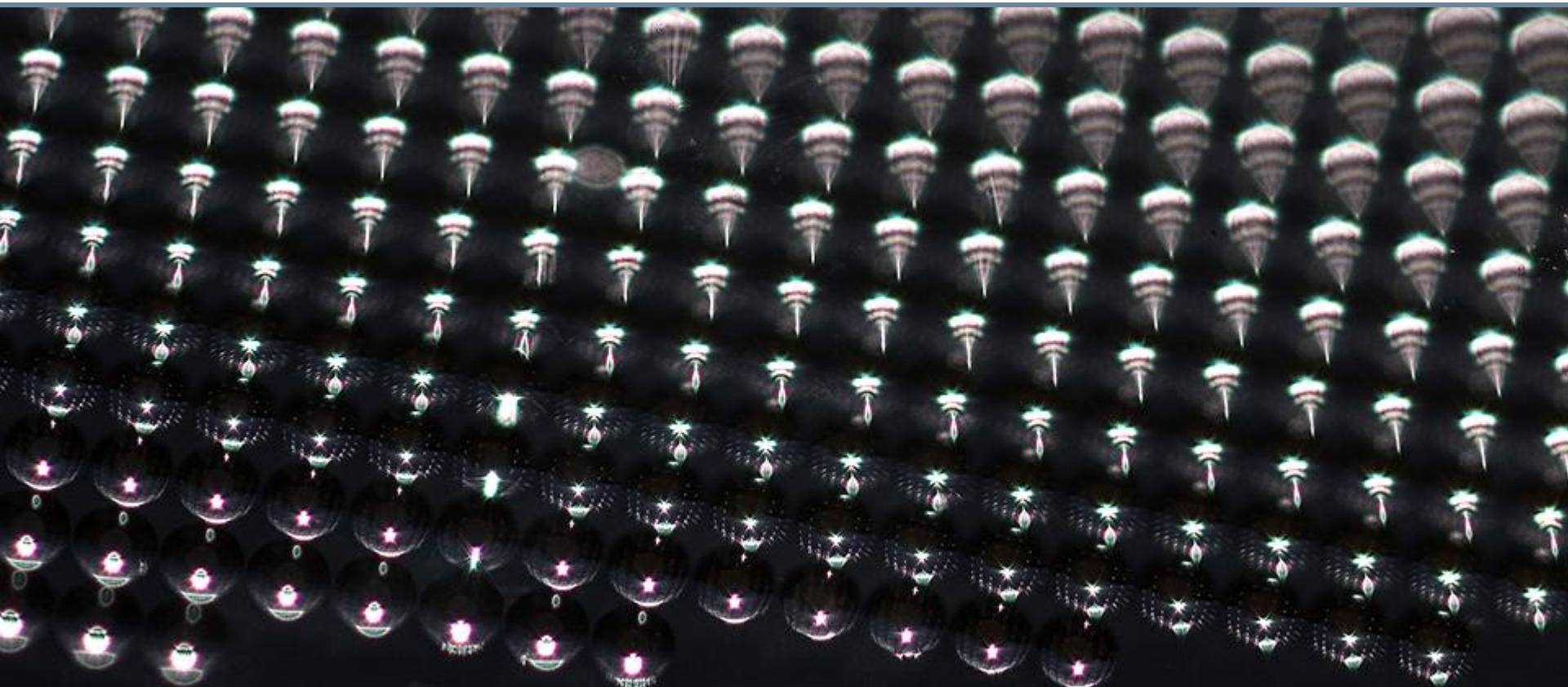


Computational Photography: Real Time Plenoptic Rendering

Andrew Lumsdaine, Georgi Chunev | Indiana University

Todor Georgiev | Adobe Systems



Who was at the Keynote Yesterday?



Overview

- Plenoptic cameras
- Rendering with GPUs
- Effects
 - Choosing focus
 - Choosing viewpoint (parallax)
 - Stereo
 - Choosing depth of field
 - HDR
 - Polarization
 - Super resolution
- Demos
- Conclusion



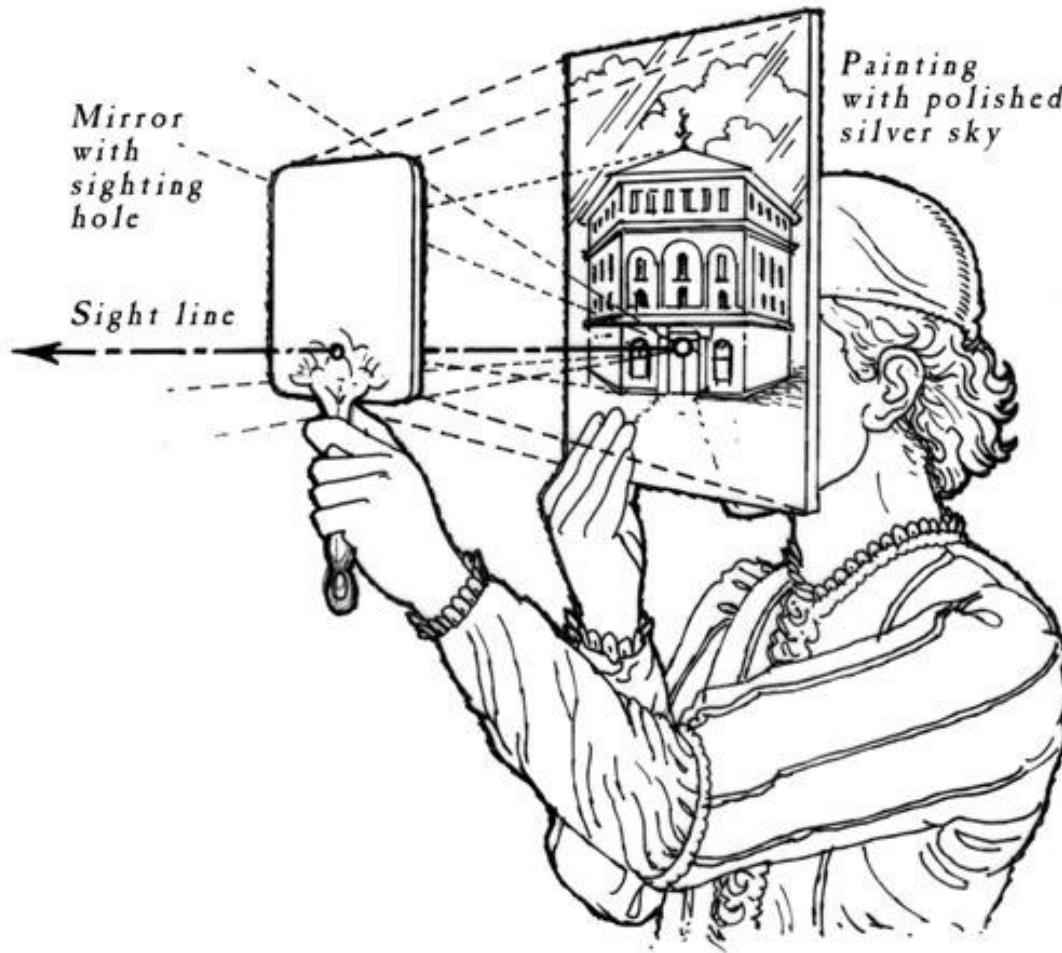
Making (and Recreating) Memories



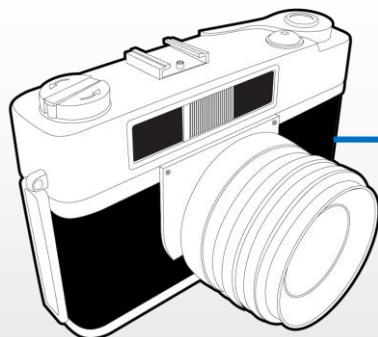
What's Wrong with this Picture?



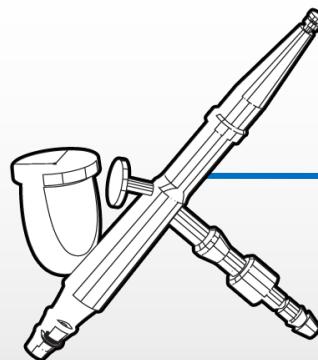
Perspective



CAPTURE



PROCESS



VIEW



Color



Exposure



Focus



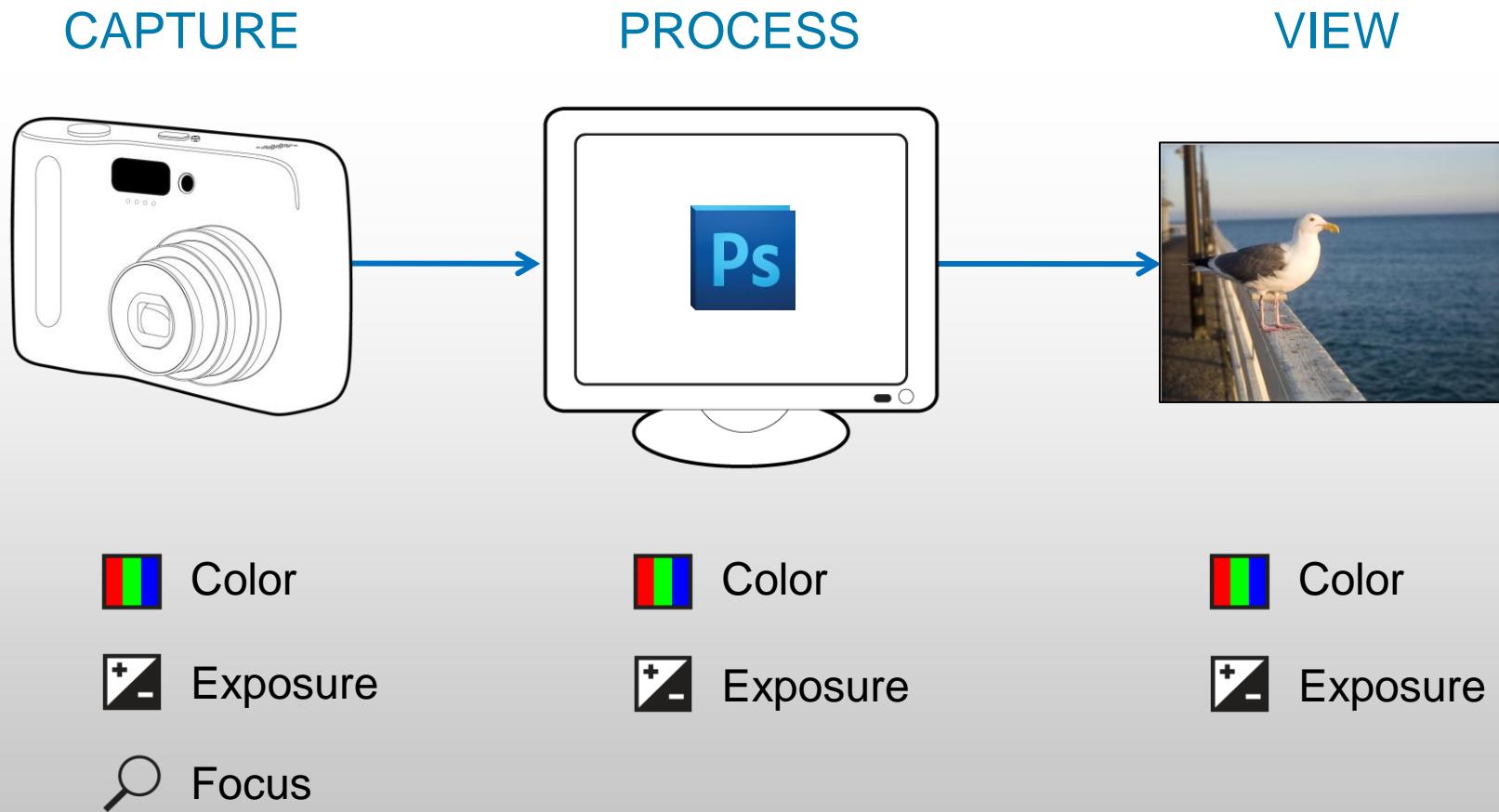
Color



Exposure

Difficult
to Adjust

Along Came Photoshop



What's Wrong with This Picture?



What's Wrong? It's Only a Picture!



Can We Create More than Pictures?

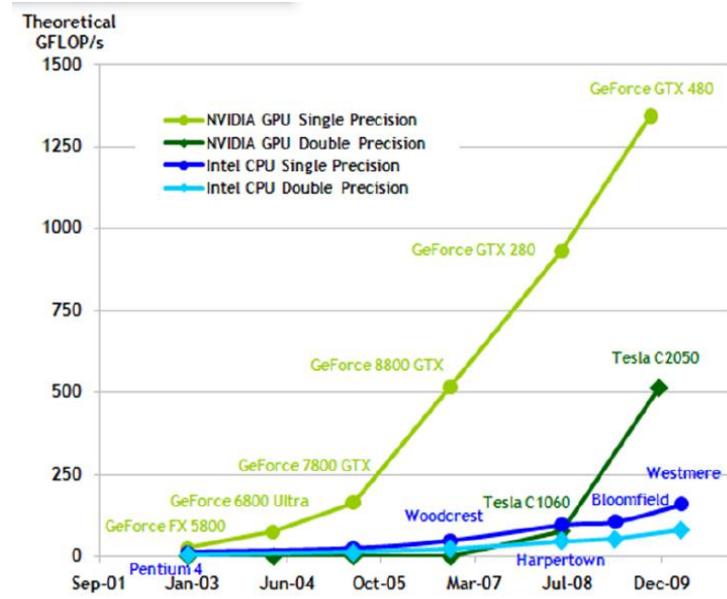
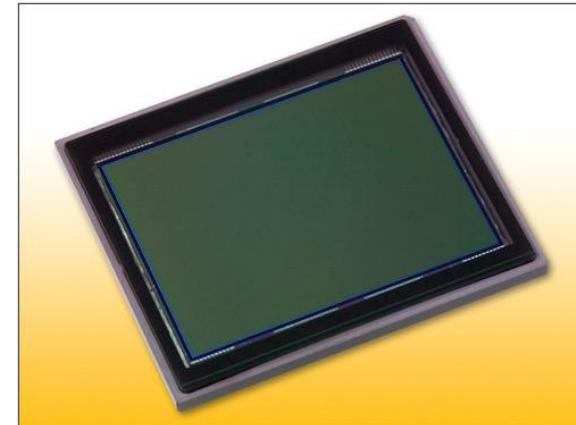


- *Can we request that Photography renders the full variety offered by the direct observation of objects? Is it possible to create a photographic print in such a manner that it represents the exterior world framed, in appearance, between the boundaries of the print, as if those boundaries were that of a window opened on reality.*

Gabriel Lippmann, 1908.

Pixels and Cores

- Moore's Law: Megapixels keep growing
 - 7.2 MP = 8 by 10 at 300dpi
 - Available on cell phones
- 60MP sensors available now
 - Larger available soon (can a use be found?)
- Use pixels to capture richer information about a scene
- Computationally process captured data
 - GPU power also riding Moore's Law curve



Infinite Variety (Focusing)



Focusing



Focusing



Different Views



Different Views

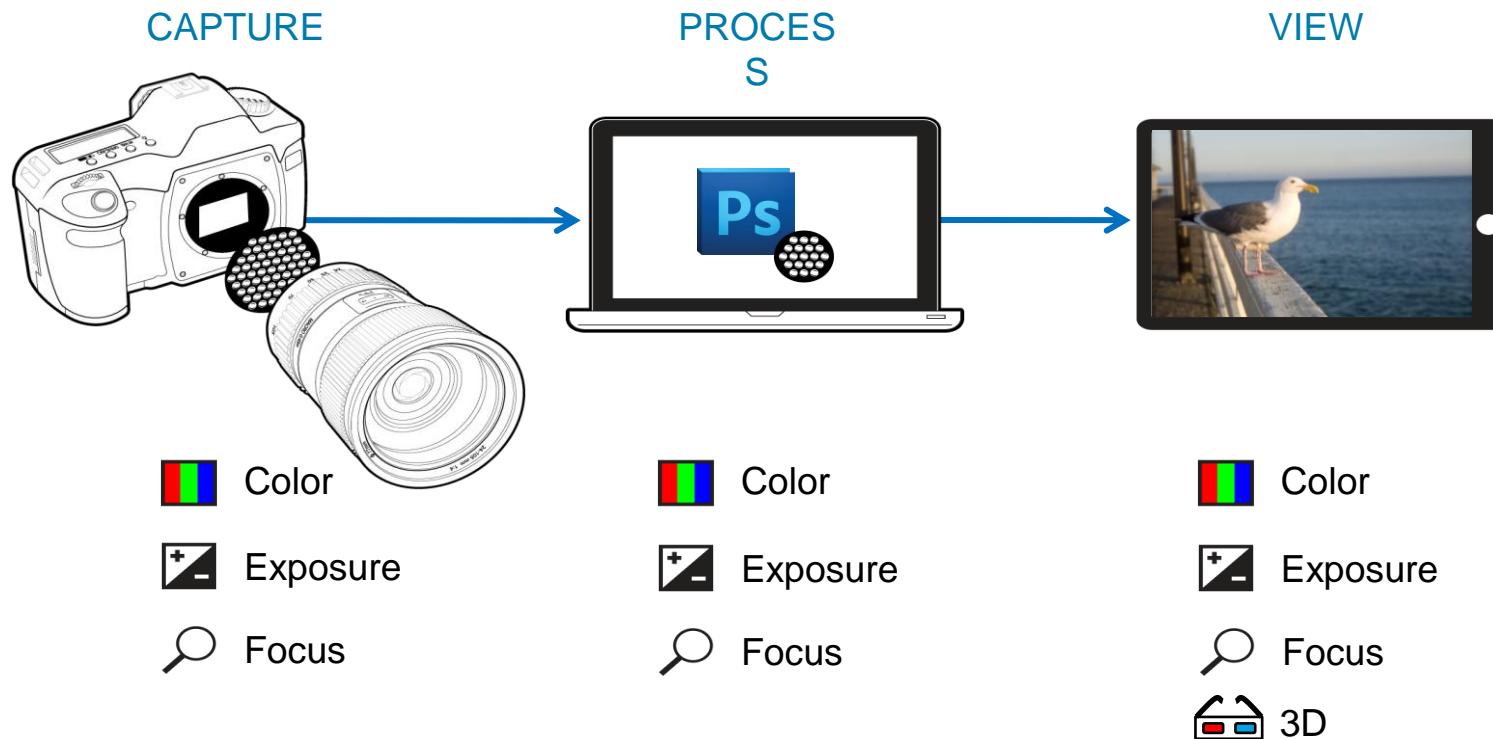


Different Views



Computational Photography

- With traditional photography light rays in a scene go through optical elements and are captured by a sensor
- With computational photography, we capture the light rays and apply optical elements computationally

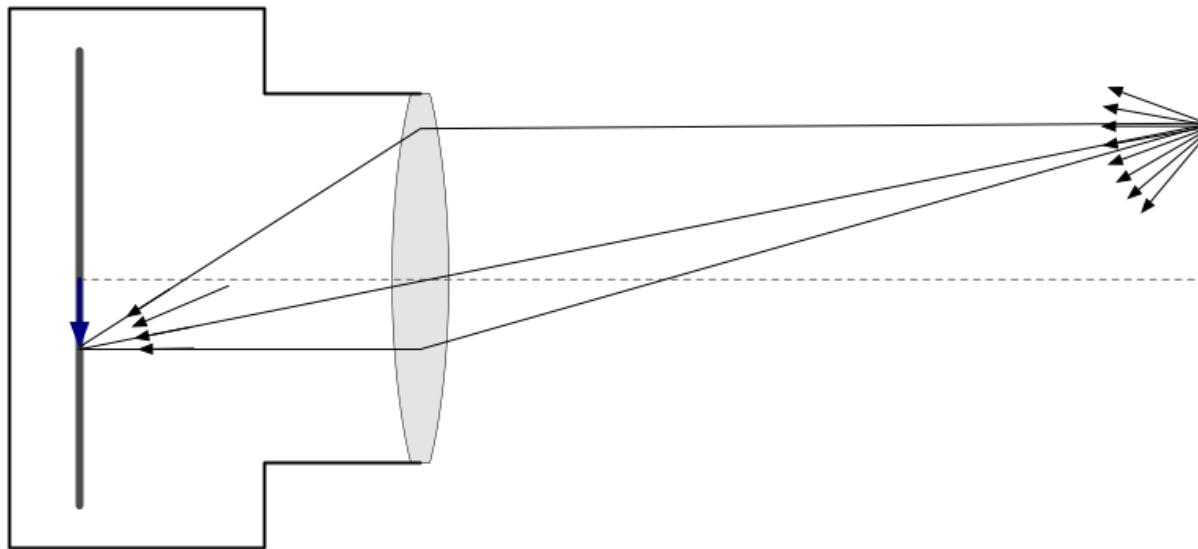


Real Time Plenoptic Rendering

- Capture the information about the intensity of light rays in a scene (the radiance or plenoptic function), not just a 2D picture
- Render images – take pictures – later, computationally
- Explore the “full variety”
- Requires real-time rendering (made possible by GPU)
- Opens new ways of working with photography
- Provides new photographic effects

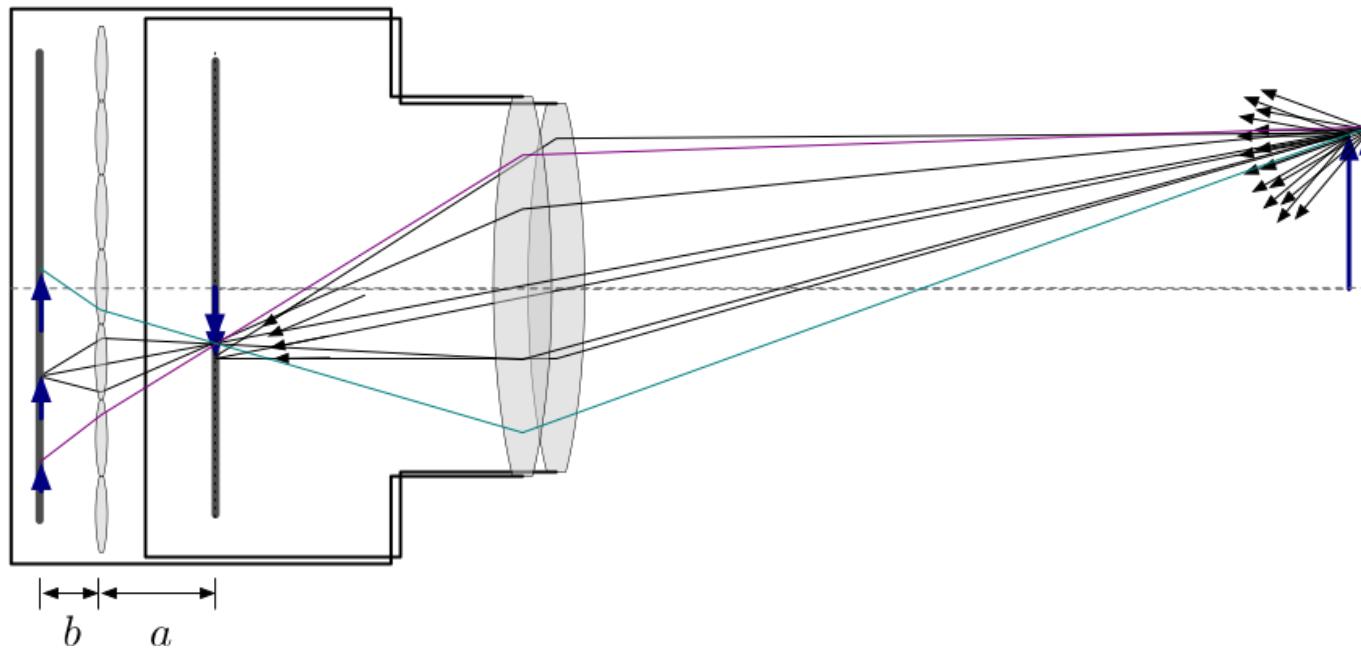
Taking Pictures

- A traditional camera places optical elements into the light rays in a scene
- A pixel on the sensor is illuminated by rays from all directions
- The sensor records the intensity of the **sum** of those rays
- We lose all of the information about individual rays



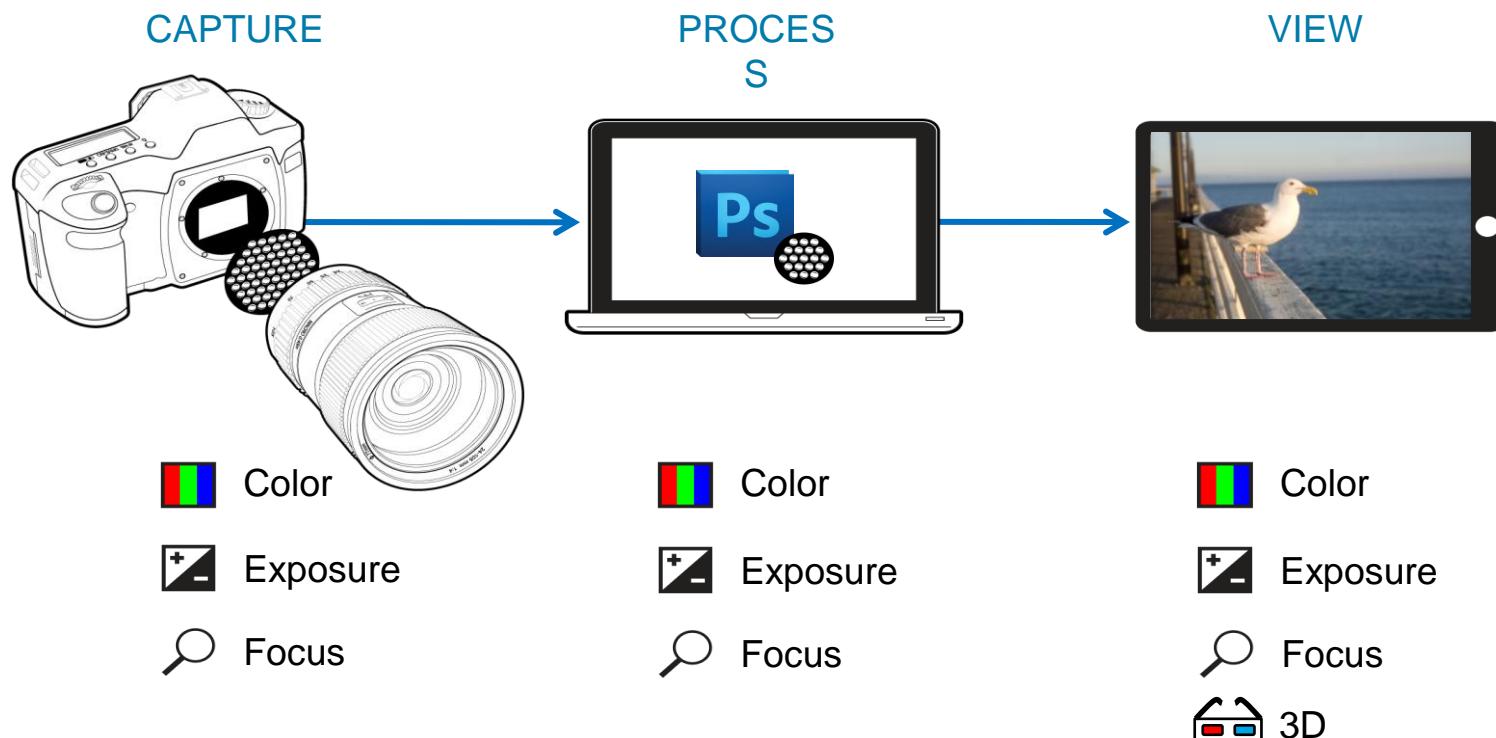
Radiance (Plenoptic Function, Lightfield)

- Instead of integrating rays from all directions, capture the rays themselves (the radiance)
- Record ***all*** the information about the scene



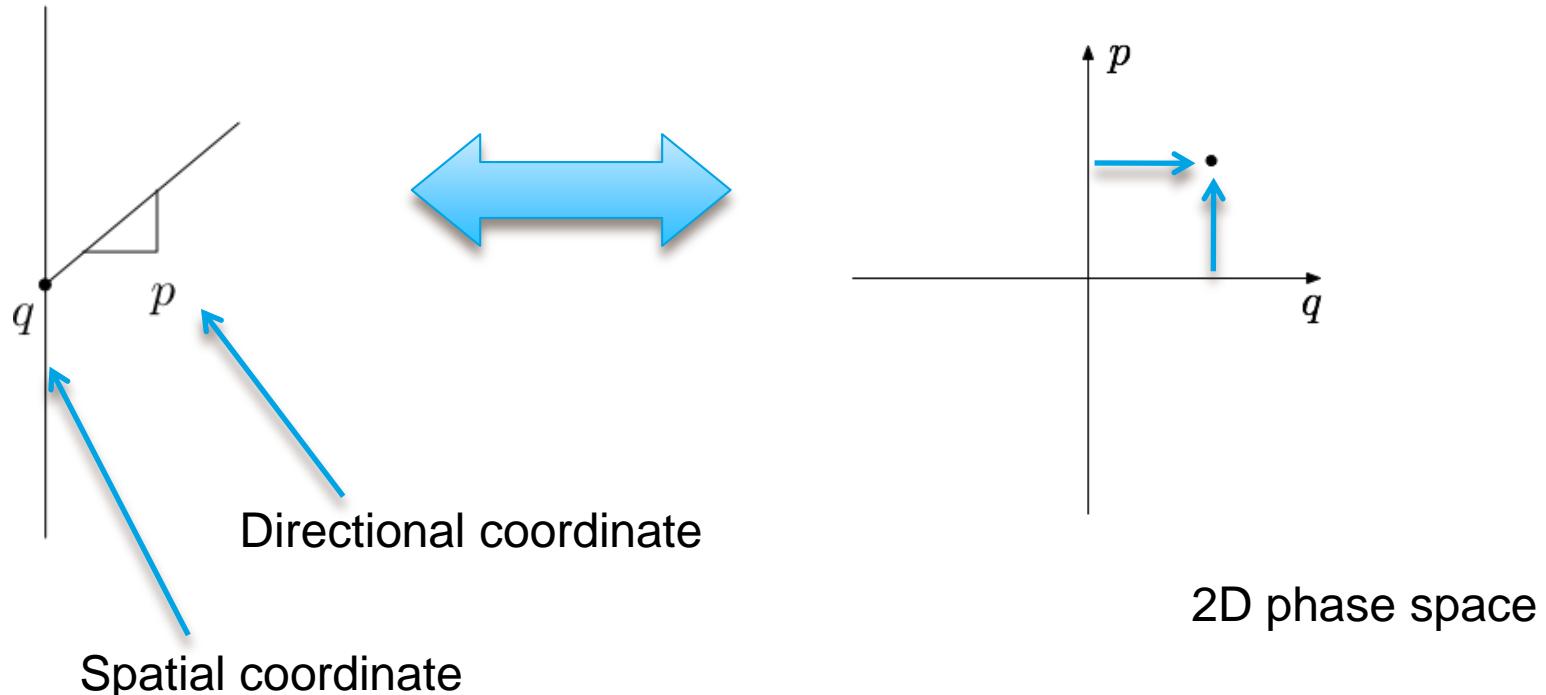
Computational Photography – The Basic Idea

- Using radicance, we can “take the picture” computationally
- Choose and apply optical elements computationally
- Render computationally
- Explore the “full variety” computationally



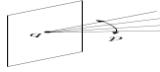
Radiance (and Transformations)

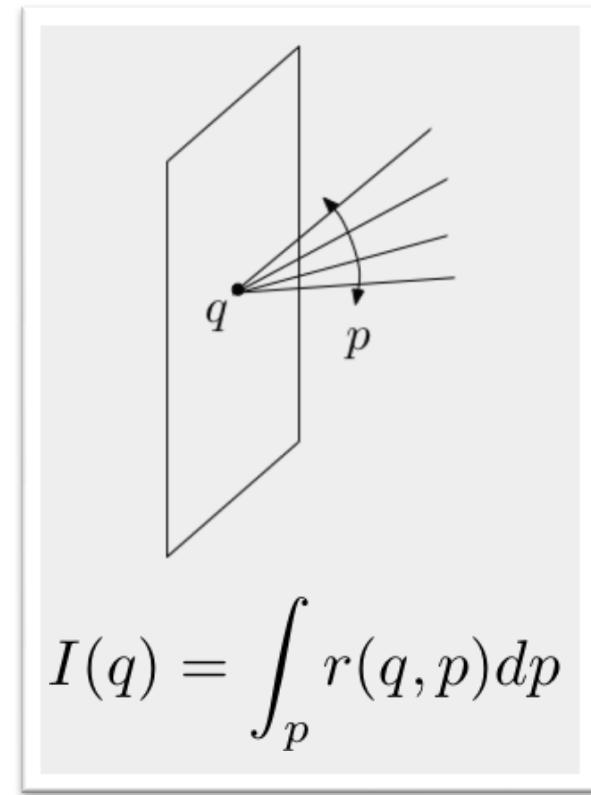
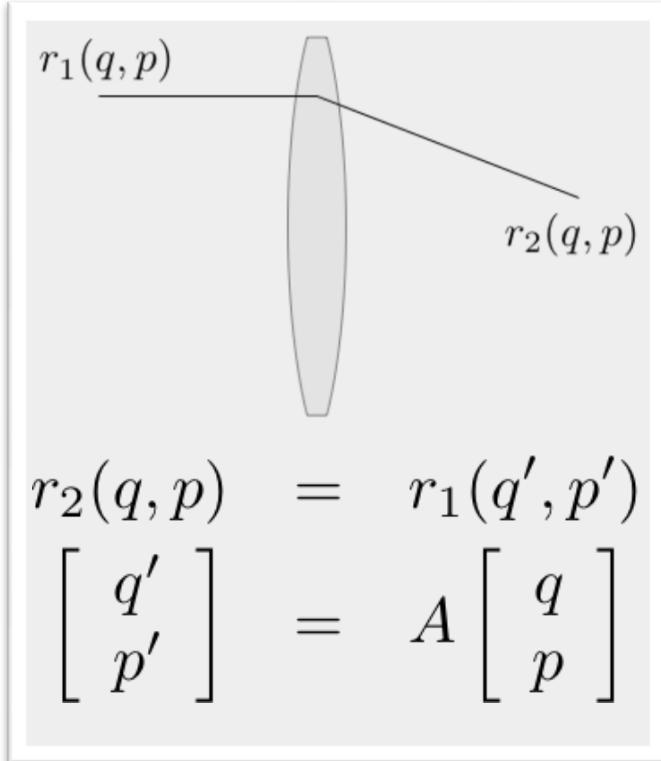
- The radiance $r(q, p)$ is a density function over 4D ray space
- Each ray is a point in 4D ray space



(2D diagrams shown because they are easier to draw.)

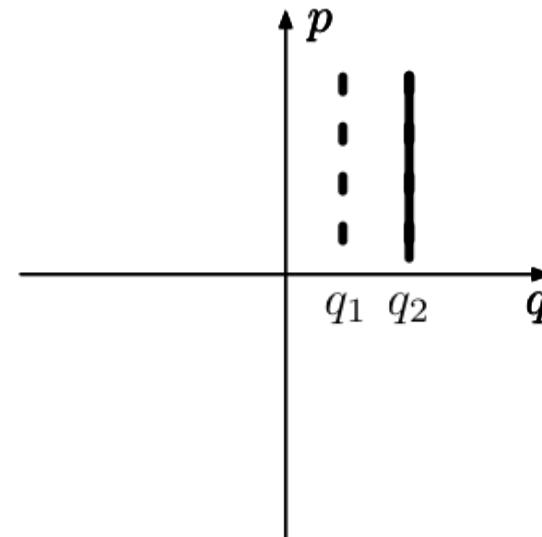
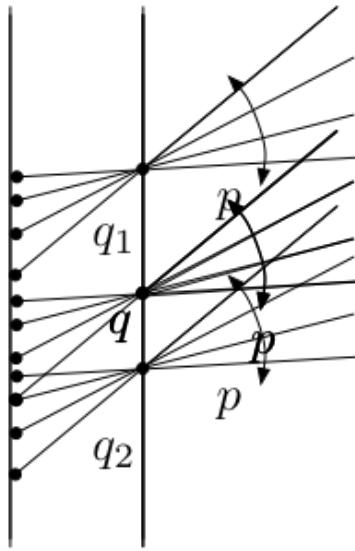
Radiance (and Transformations)

- The radiance  is a density function over 4D ray space
- Effects of optical elements (lenses, free space) are linear transformations
- Rendering (taking a picture) is integration over all p at a given q



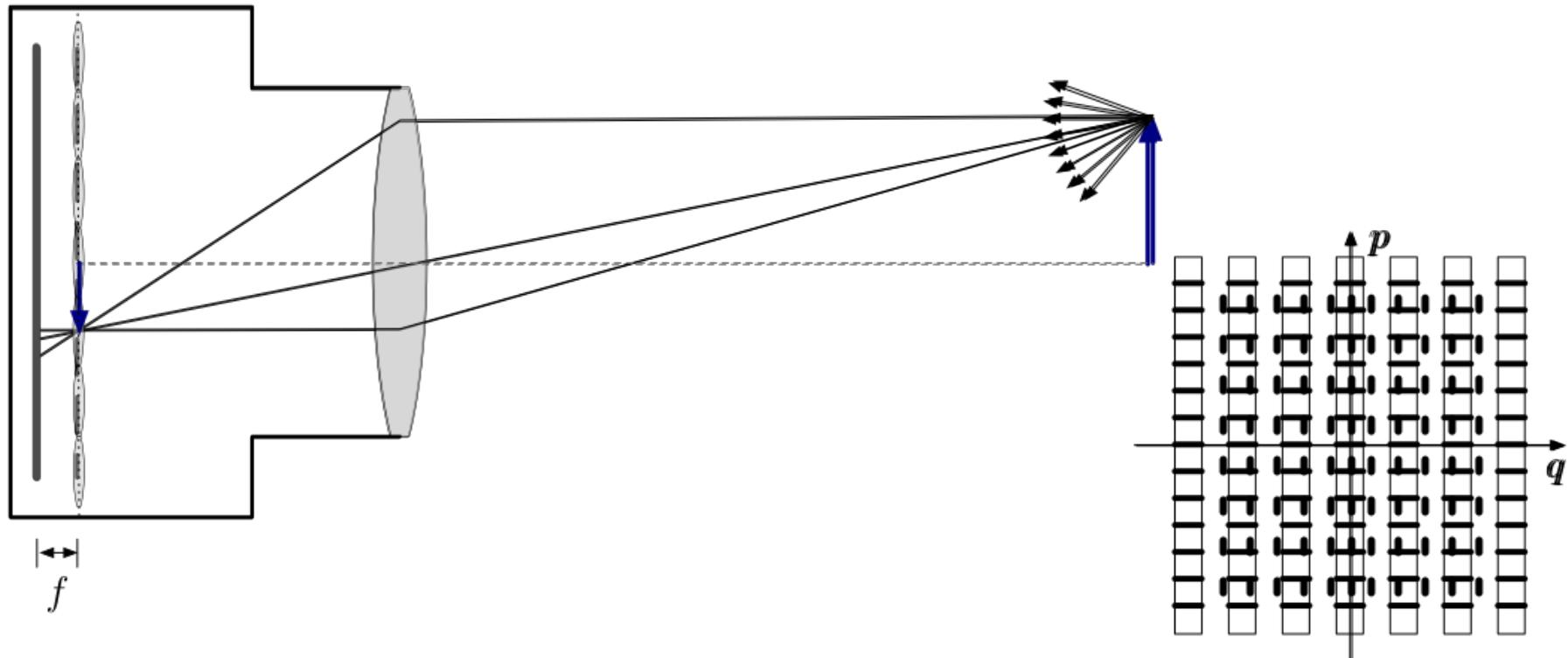
Capturing the 4D radiance with a 2D sensor

- To capture individual rays, first we have to separate them
- At a particular spatial point, we have a set of rays at all directions
- If we let those rays travel through a pinhole, they will separate into distinguishable individual rays
- Two pinholes will sample two positions



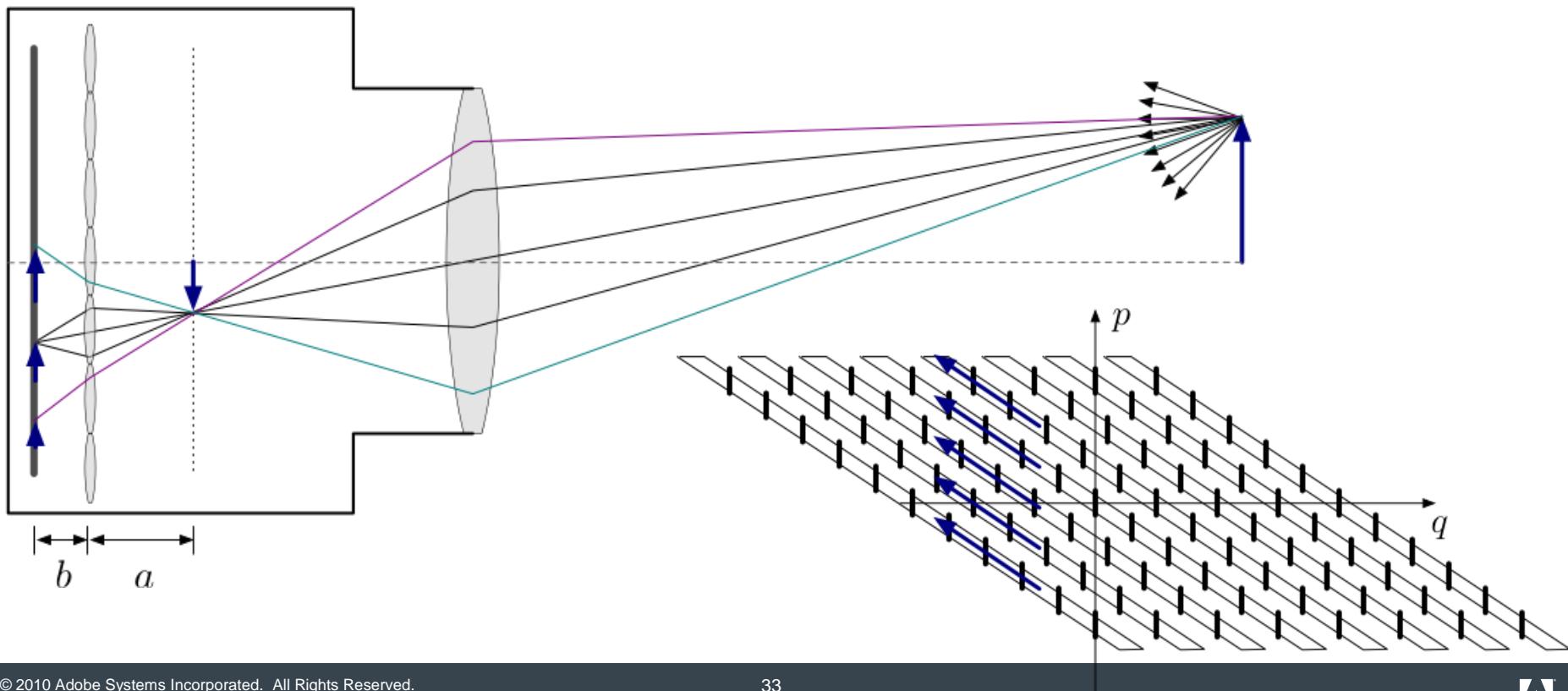
A plenoptic camera

- A camera with an array of pinholes will capture an image that represents the 4D radiance
- In practice, one might use a microlens array to capture more light



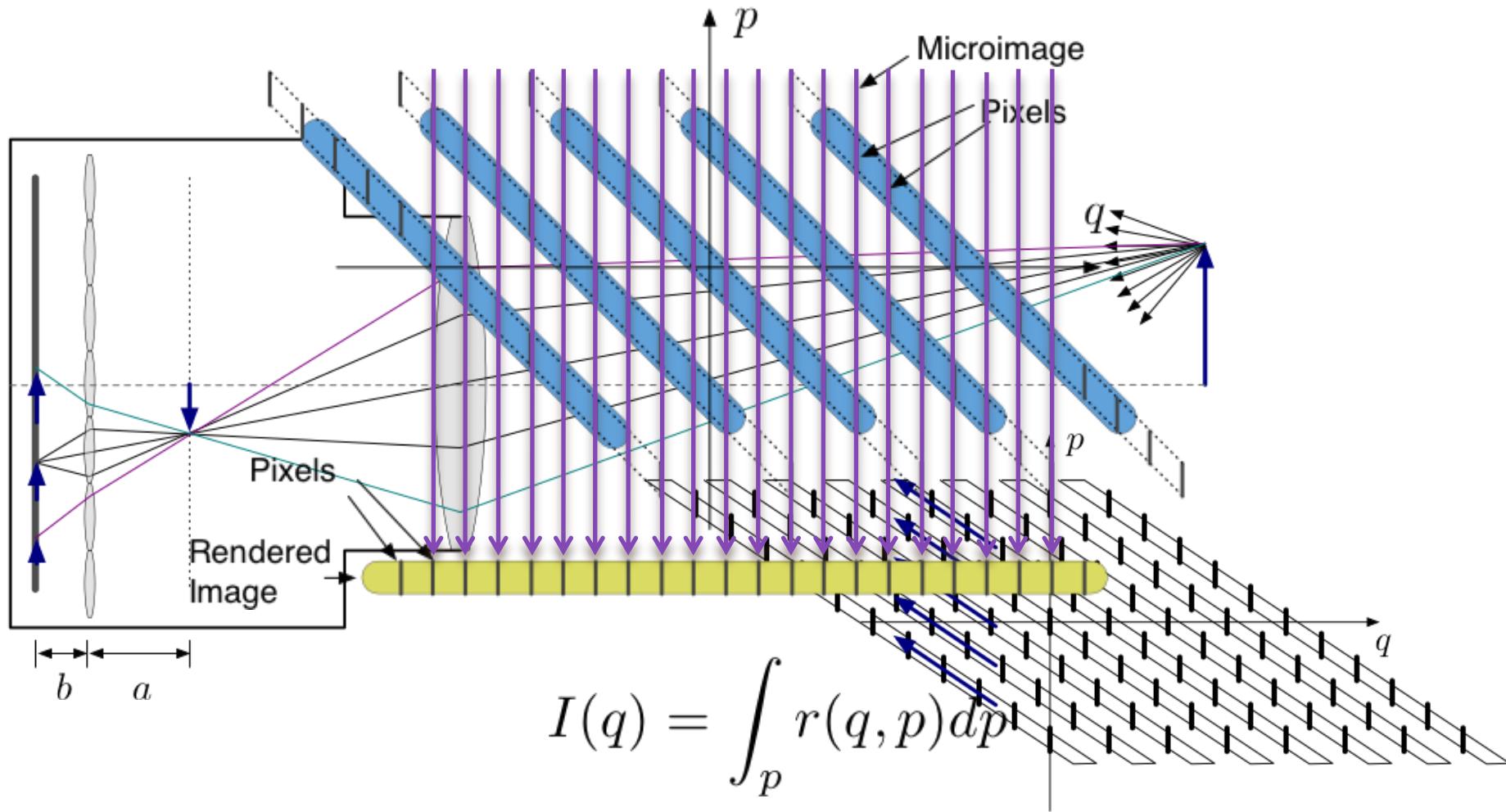
The focused plenoptic camera

- With the Adobe camera, we make one important modification
- We use the microlenses to create an array of relay cameras to sample the plenoptic function with higher spatial resolution
 - Note that image plane can also be behind the sensor (virtual image is captured)



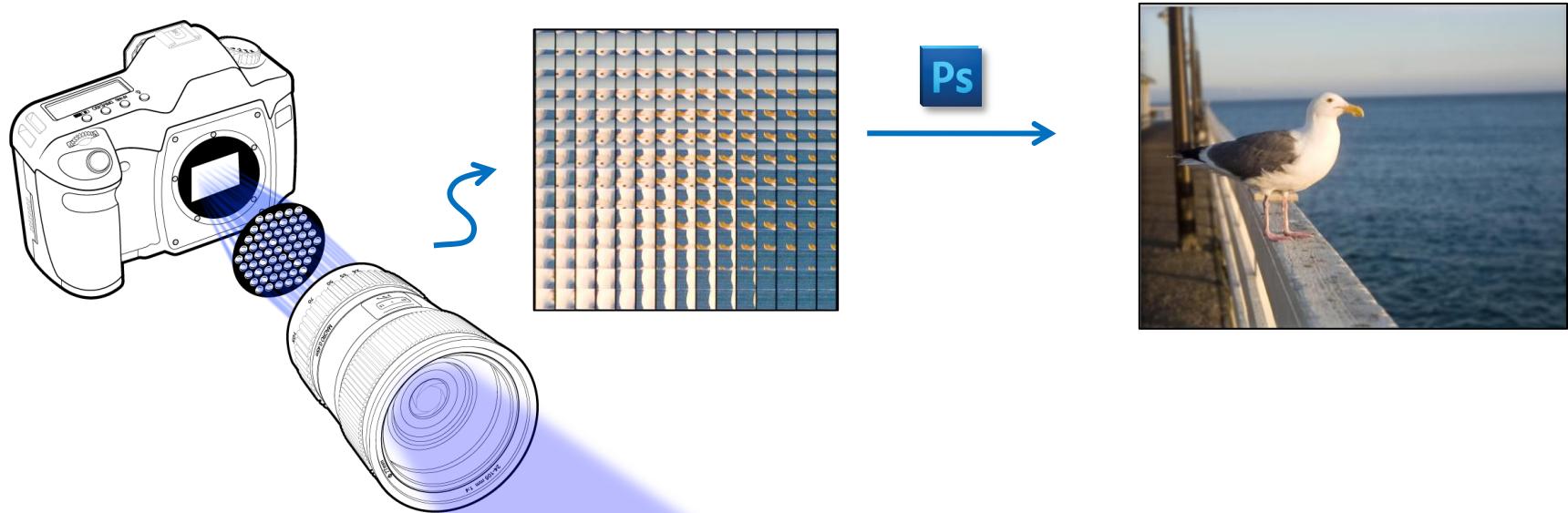
Rendering: Taking a Computational Picture

- To take a picture (render) we integrate over all directions p



The Story So Far

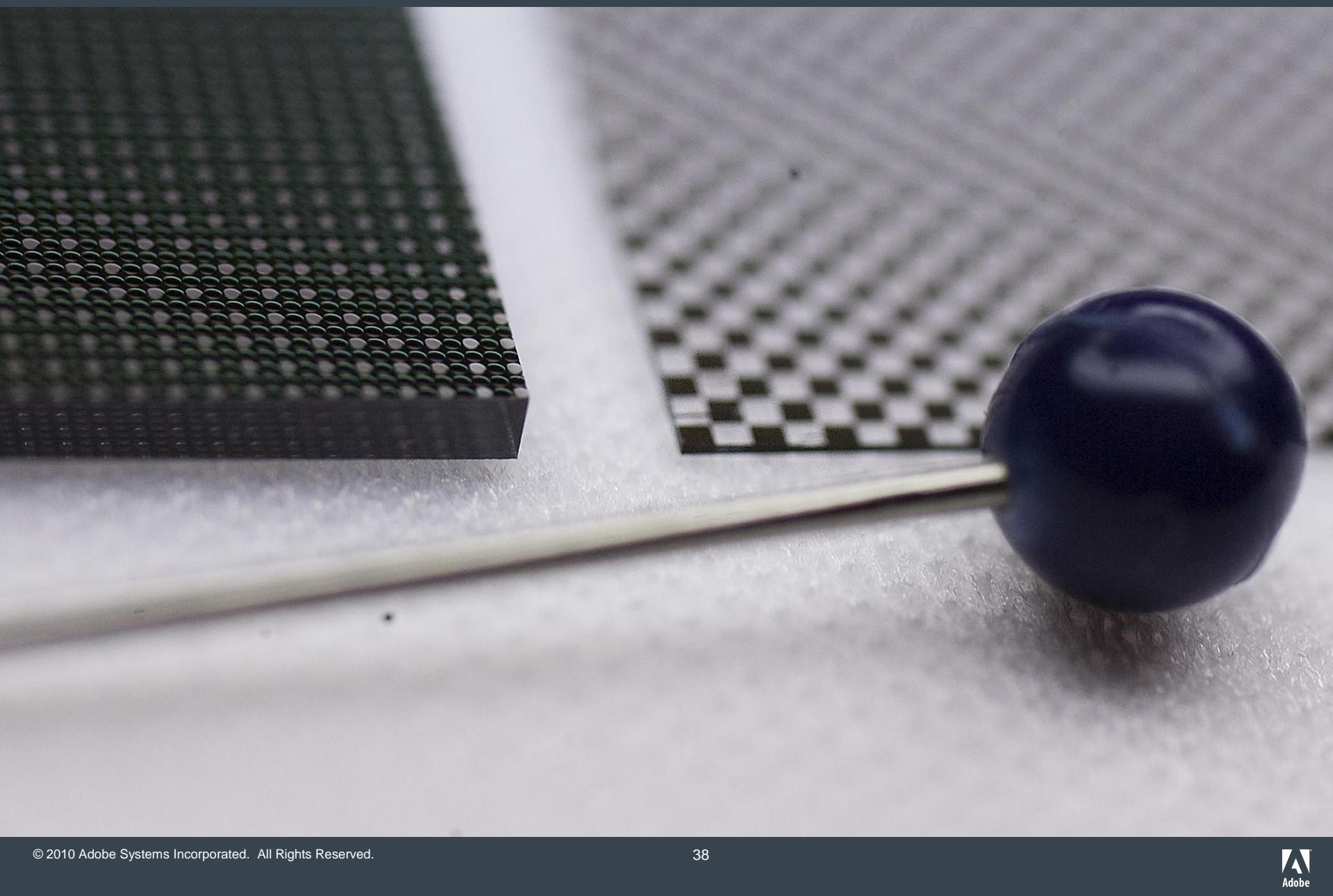
- A plenoptic camera takes a 2D picture – radiance image (or “flat”)
- The pixels are samples of the radiance in the 4D ray space
- Optical elements (lenses, space) transform the ray space
- We take a picture by rendering (computationally)
- We adjust the picture by transforming the ray space (computationally)



The Part of the Talk Where we Reveal the Magic



First, the Camera

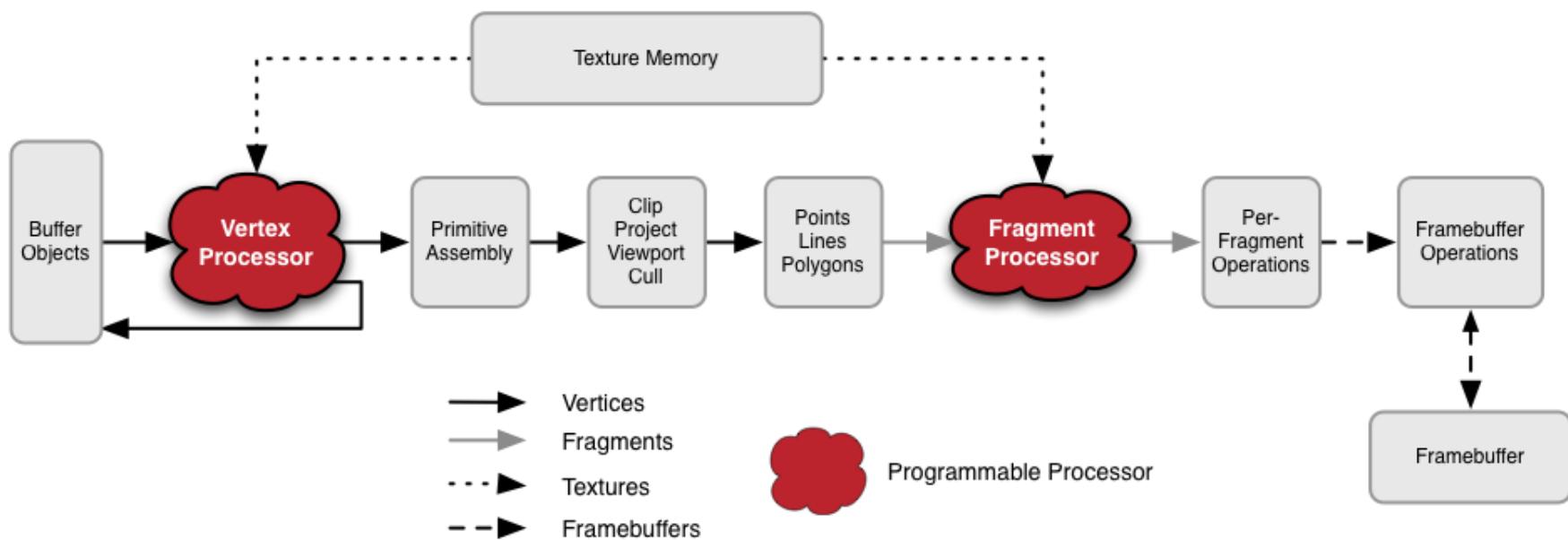


Plenoptic Image (Flat)



GPU Programming

- Basic alternatives for programming GPU: General purpose (CUDA) or graphics-based (GLSL)
- Open GL Shader Language (GLSL) a natural fit
 - Texture mapping



Rendering with GPU using Open GL

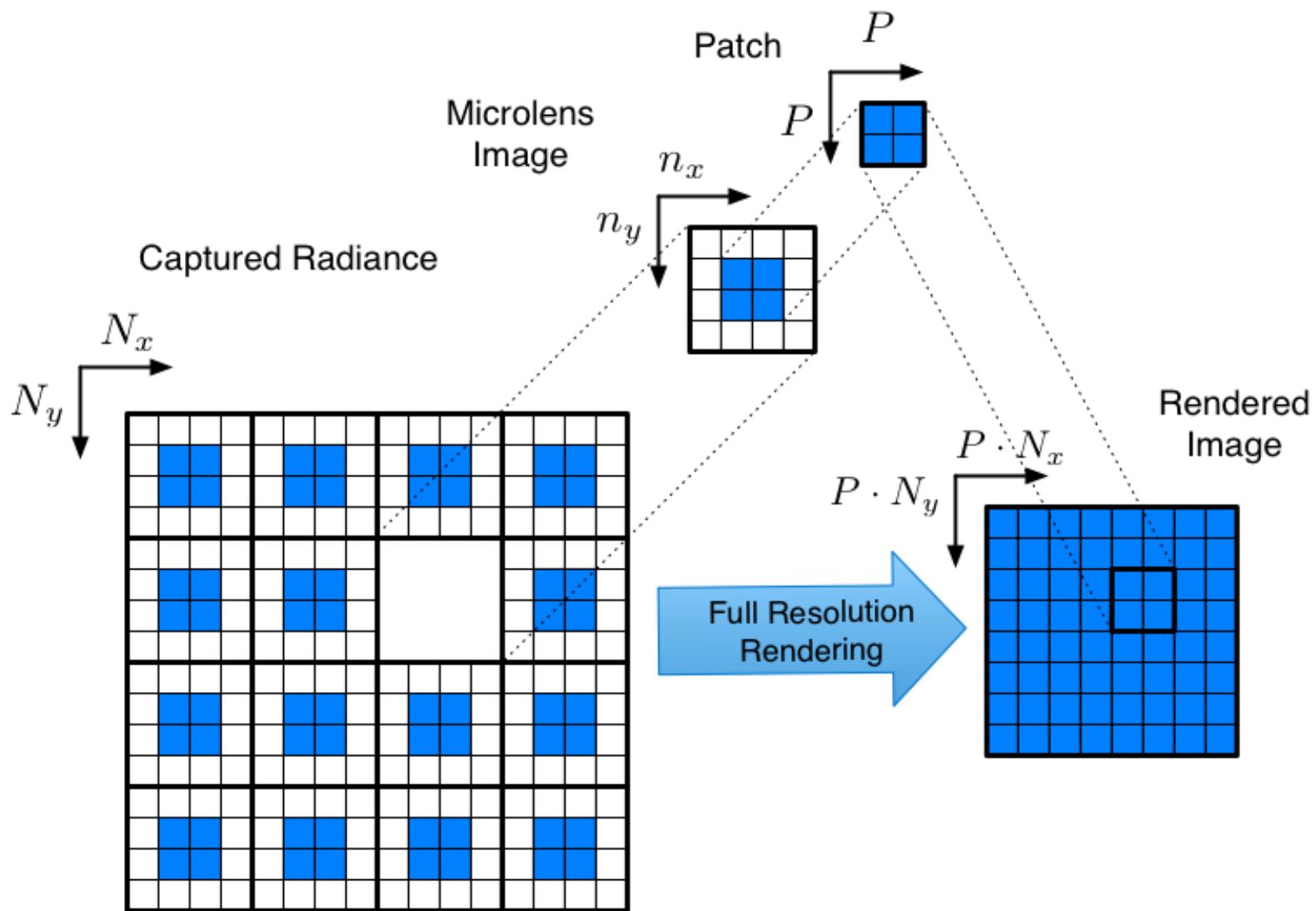
- Read in plenoptic radiance image
- Create 2D texture object for radiance
- Serialize image data to Open GL compatible format
- Define the texture to OpenGL

```
image = Image.open("lightfield.png")
```

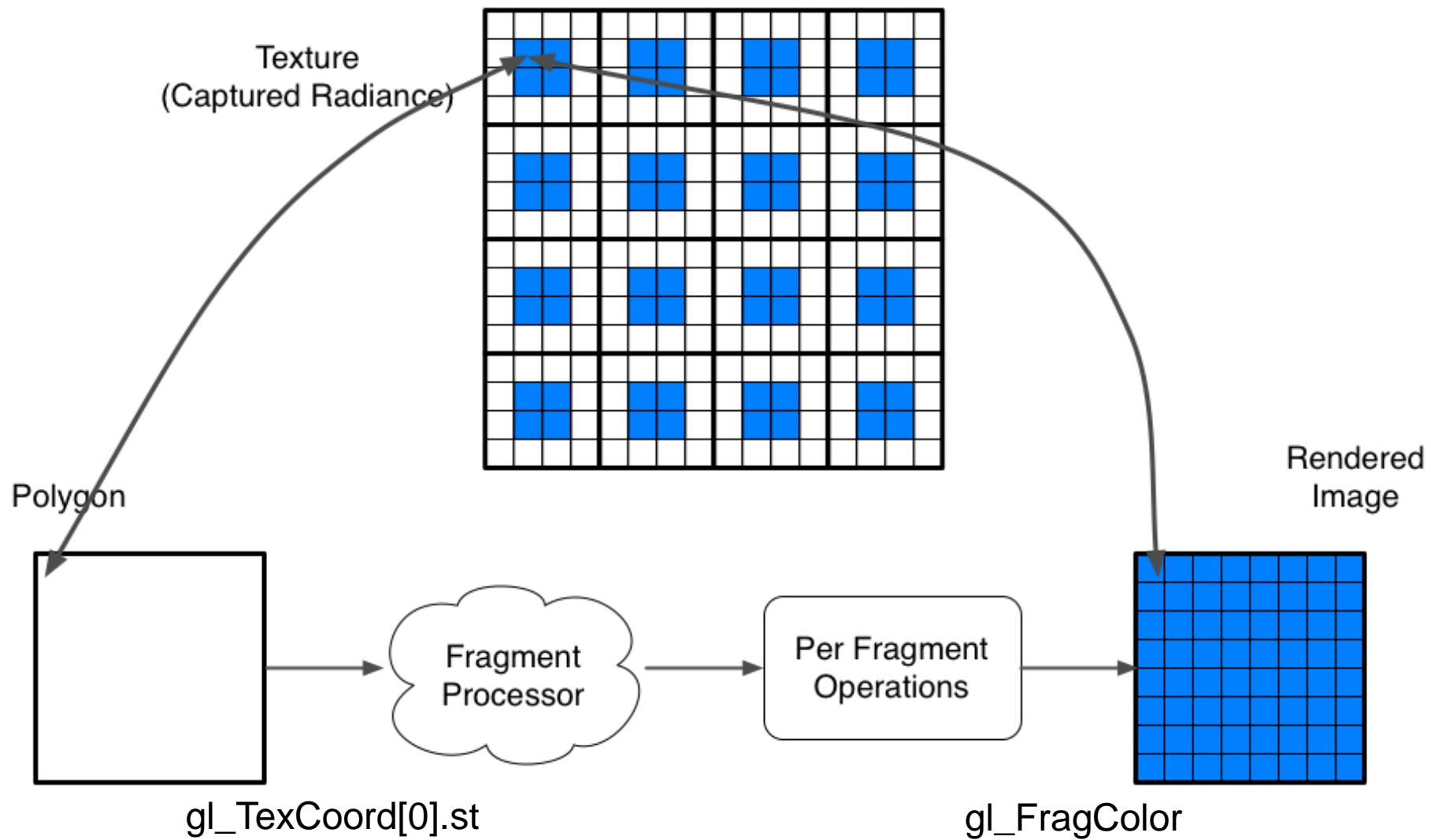
```
str_image = image.tostring("raw", "RGBX", 0, 1)
```

```
glActiveTexture(GL_TEXTURE0)
lfTexture = glGenTextures(1)
glBindTexture(GL_TEXTURE_RECTANGLE_ARB, lfTexture)
glTexImage2D(GL_TEXTURE_RECTANGLE_ARB, 0, 3,
            image.size[0], image.size[1], 0,
            GL_RGBA, GL_UNSIGNED_BYTE, str_image)
```

GLSL Implementation of Rendering



GLSL Implementation of Rendering



GLSL Implementation of Rendering

- Given output pixel coordinate $gl_TexCoord[0].st$

- Find relevant microimage

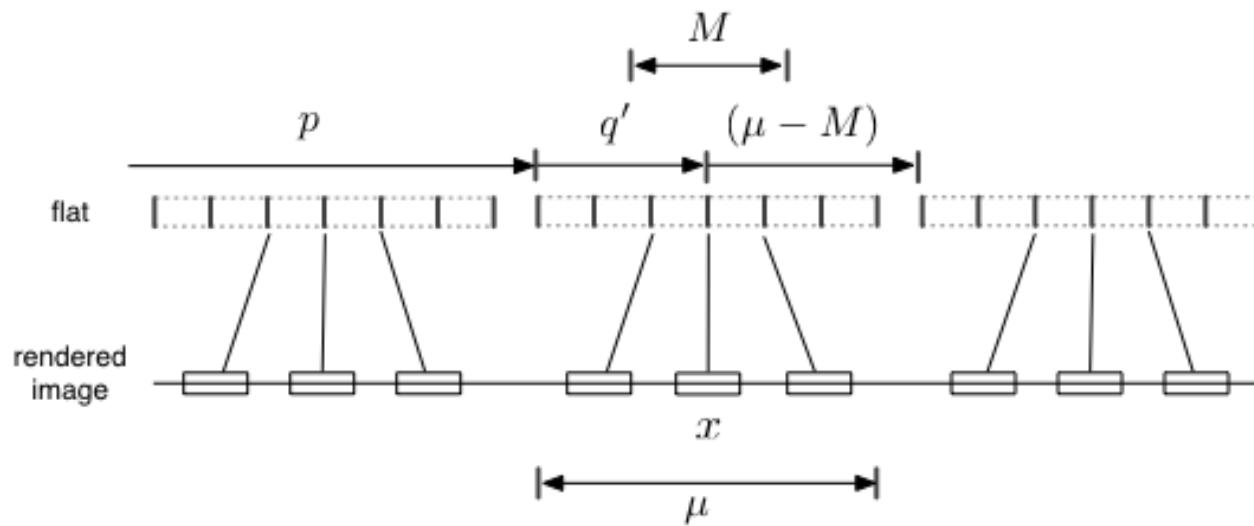
$$p = \lfloor \frac{x}{\mu} \rfloor$$

- Find offset within

$$q = \left(x - \lfloor \frac{x}{\mu} \rfloor \mu \right) \frac{M}{\mu} = \left(\frac{x}{\mu} - p \right) M$$

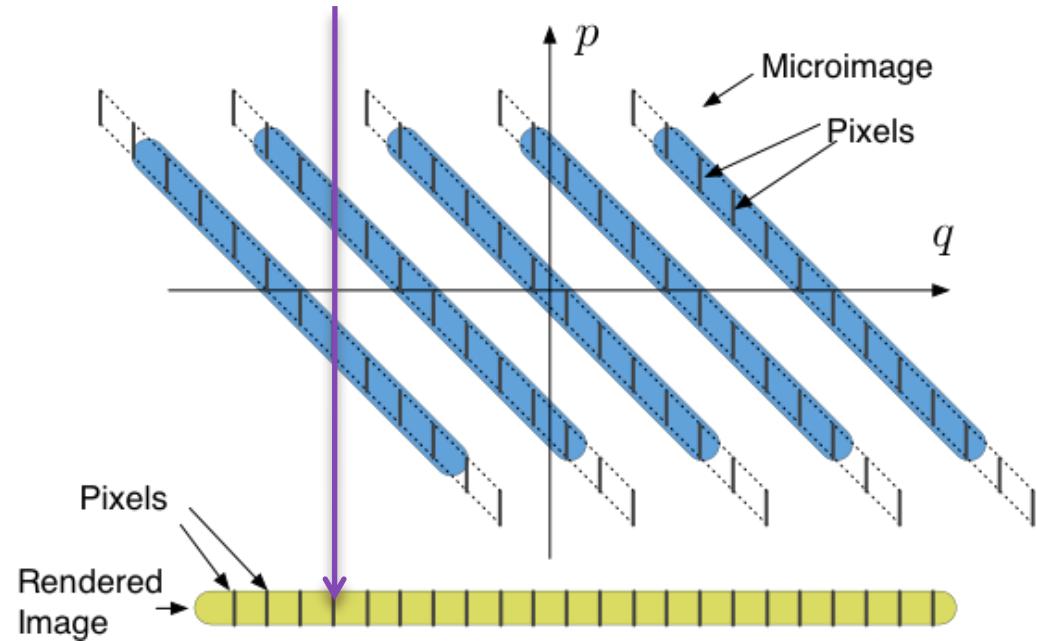
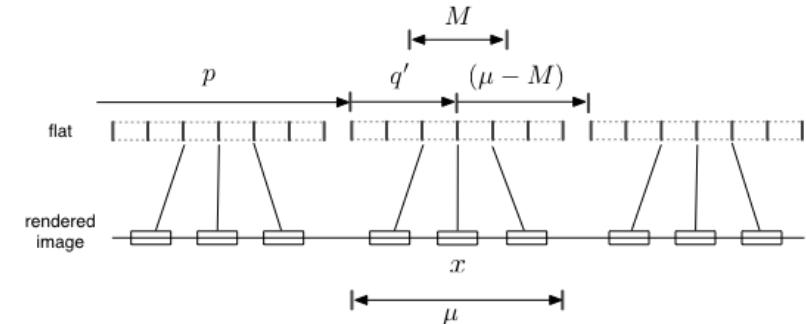
- Center

$$q' = q + \frac{\mu - M}{2} = \left(\frac{x}{\mu} - p \right) M + \frac{\mu - M}{2}$$

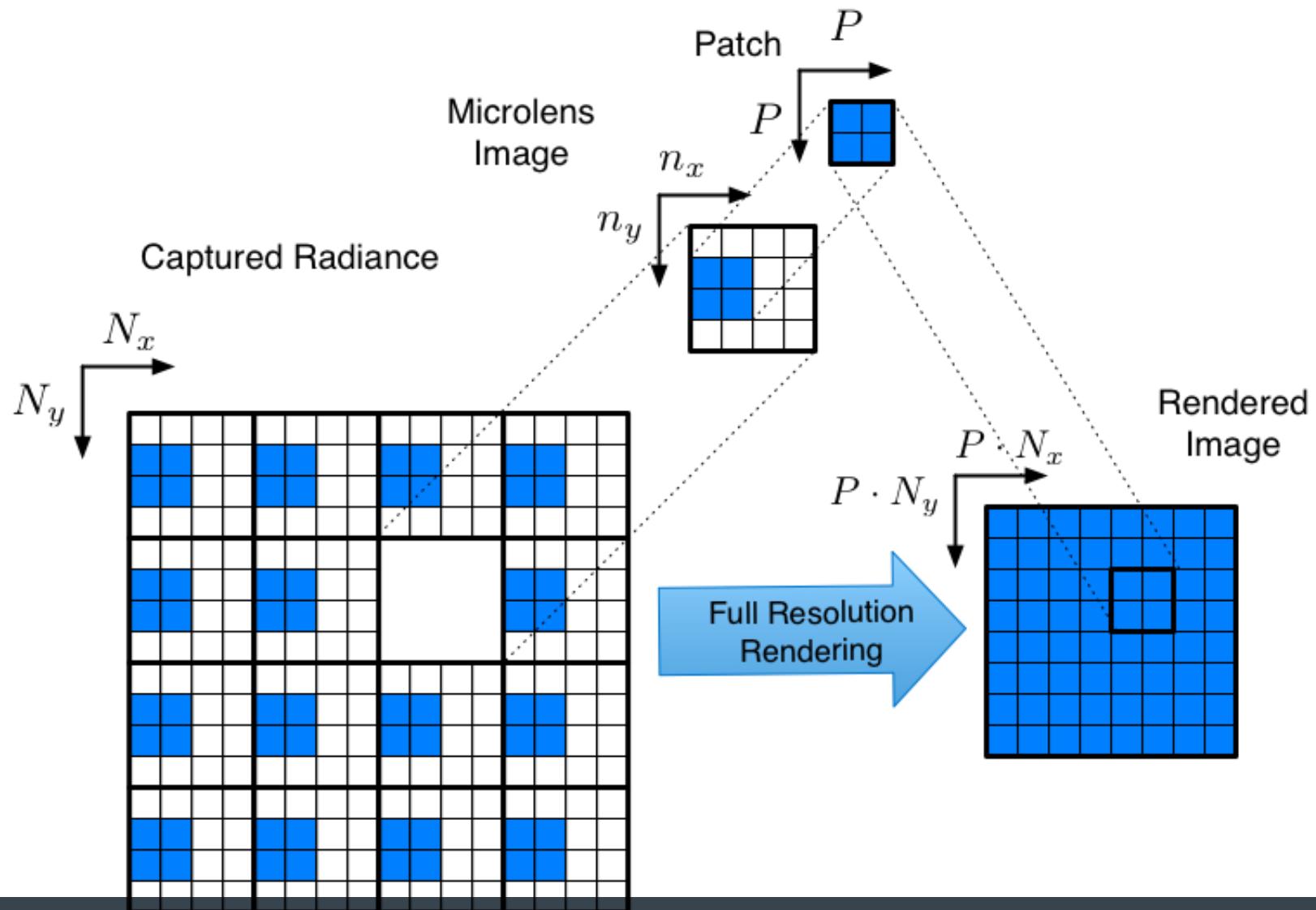


GLSL Rendering

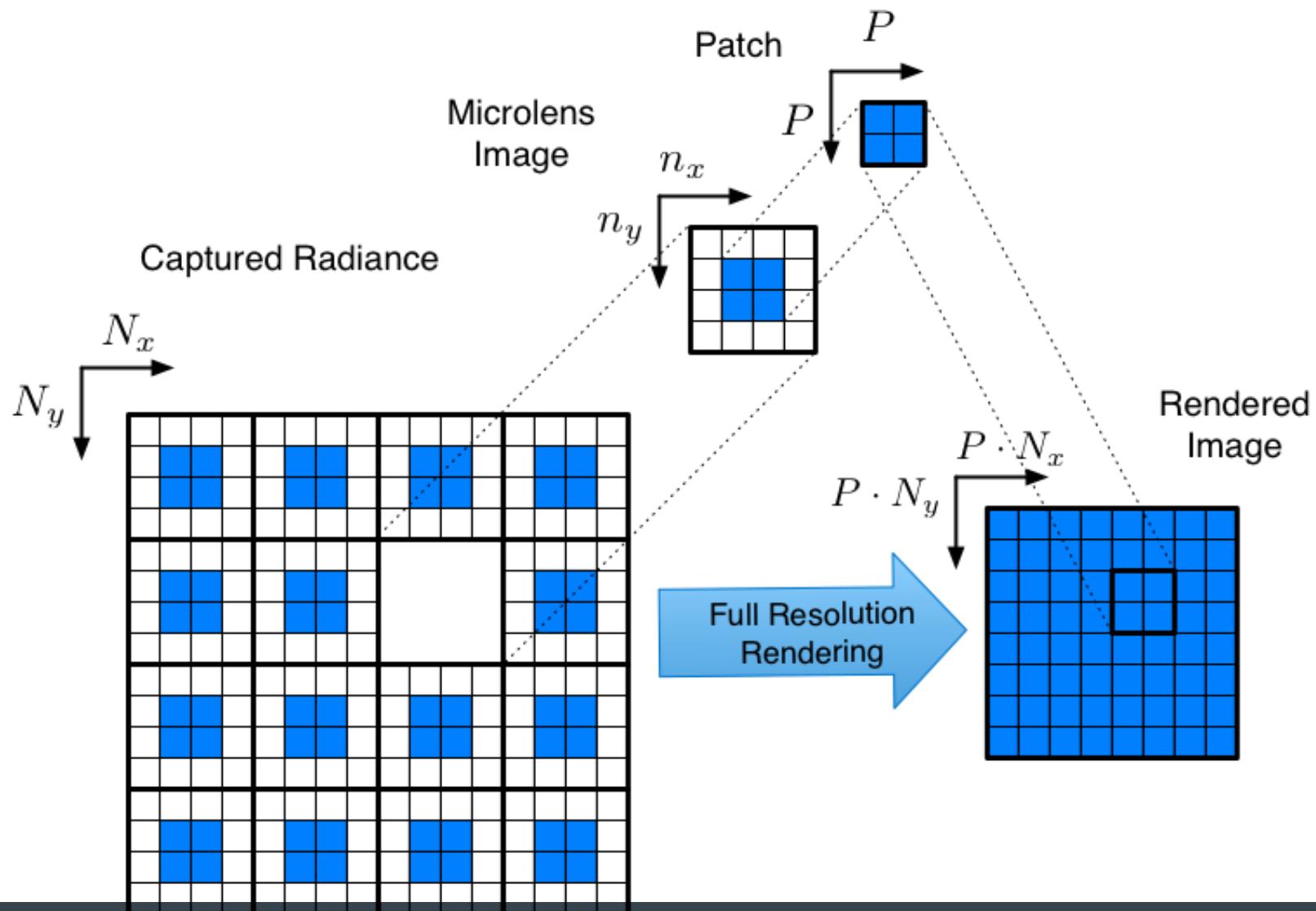
```
uniform sampler2DRect flat;  
  
uniform float M, mu;  
  
void main()  
{  
    vec2 x_mu = gl_TexCoord[0].st/mu; //  $x/\mu$   
    vec2 p = floor(x_mu); //  $p = \lfloor x/\mu \rfloor$   
    vec2 q = (x_mu - p) * M; //  $(x/\mu - p)M$   
    vec2 qp = q + 0.5*(mu - M); //  $q' = q + (\mu - M)/2$   
  
    vec4 colXY = vec4(0.0);  
    for (int i = -1; i <= 1; ++i) {  
        for (int j = -1; j <= 1; ++j) {  
  
            vec2 ij = vec2(float(i), float(j));  
            vec2 dq = qp - ij * M;  
            vec2 fx = (p + ij)*mu + dq;  
  
            colXY += texture2DRect(flat, fx);  
        }  
    }  
    gl_FragColor = colXY / 5.0;  
}
```



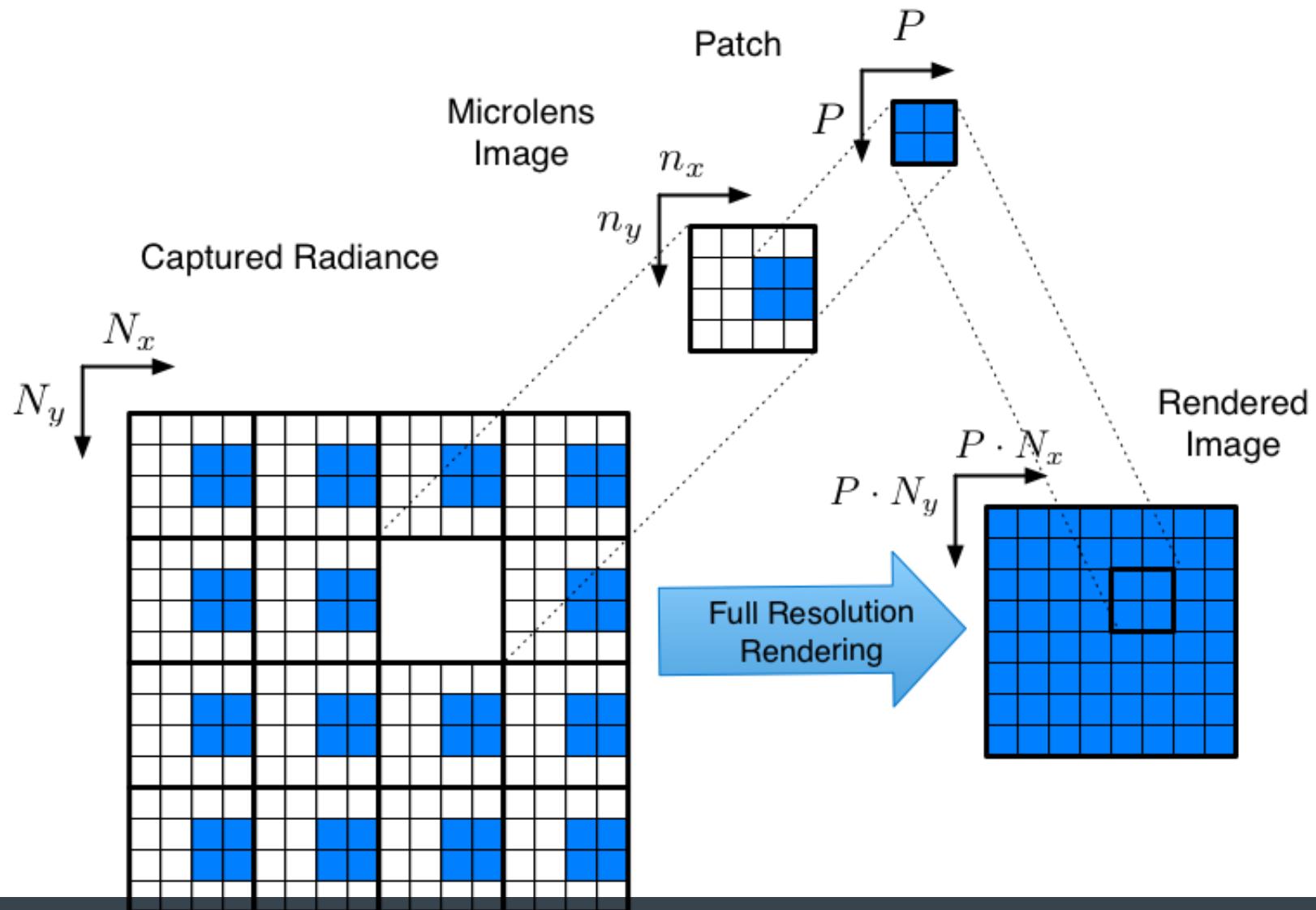
Choosing View



Choosing View



Choosing View



GLSL Rendering

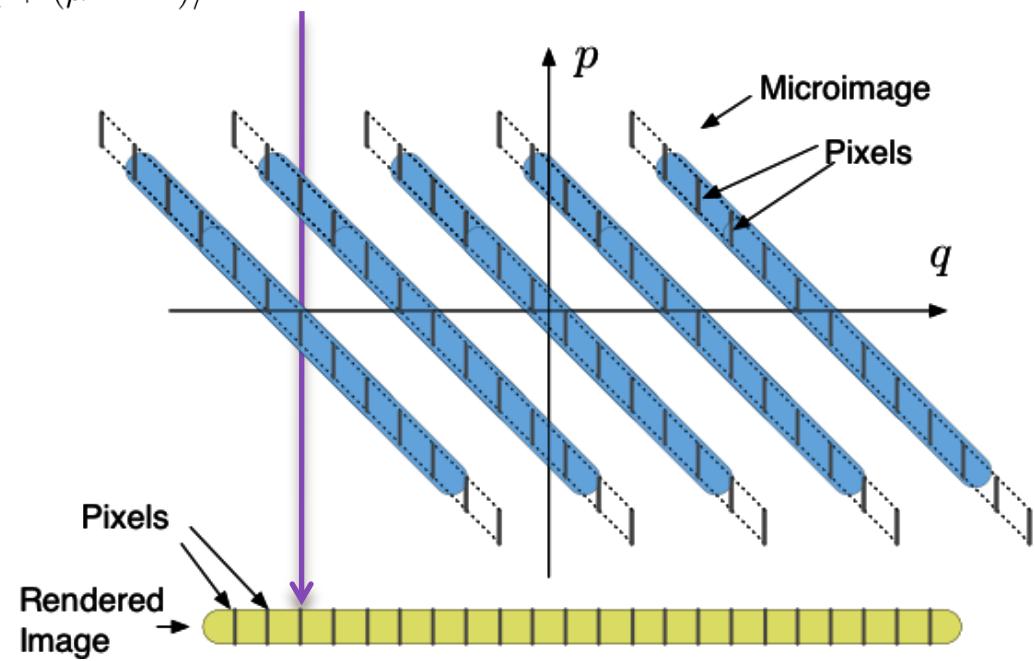
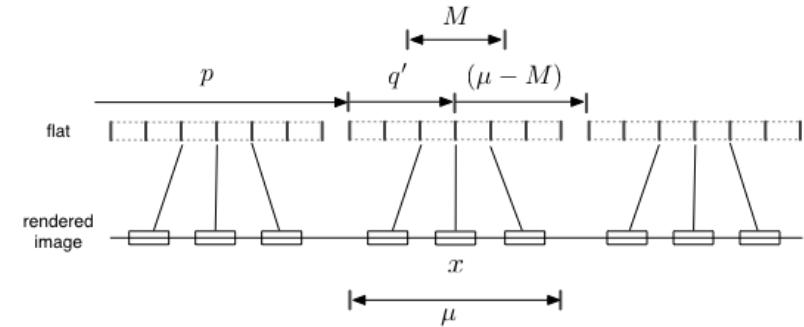
```

uniform sampler2DRect flat;           // Plenoptic image
uniform sampler2DRect flat;
uniform float M, mu;
uniform float mu;
uniform vec2 offset;

void main()
{
    vec2 x_mu = gl_TexCoord[0].st/mu; // x/mu
    vec2 p = floor(x_mu);           // p = floor(x_mu)
    vec2 q = (x_mu - p) * M;        // (x_mu - p) * M
    vec2 qp = q + 0.5*(mu - M);    // q' = q + (mu - M)/2

    vec4 colXX = vec4(0.0);
    vec4 colXY = vec4(0.0);
    for (int i = -1; i <= 1; ++i) {
        for (int j = -1; j <= 1; ++j) {
            vec2 ij = vec2(float(i), float(j));
            vec2 dq = qp - ij * M;
            vec2 fx = (p + ij) * mu + dq + offset;
            colXX += texture2DRect(flat, fx);
            colXY += texture2DRect(flat, fx);
        }
    }
    gl_FragColor = colXY / 5.0;
    gl_FragColor = colXY / 5.0;
}

```



Choosing View



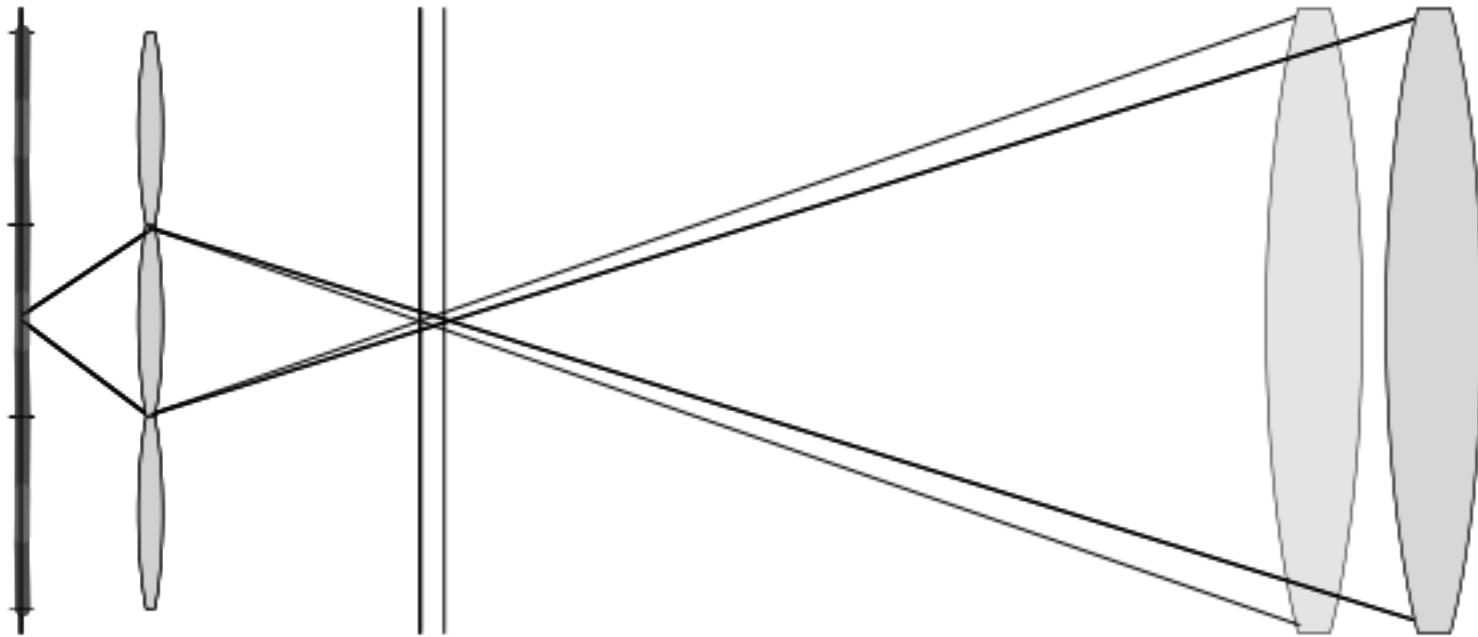
Choosing View



Choosing View



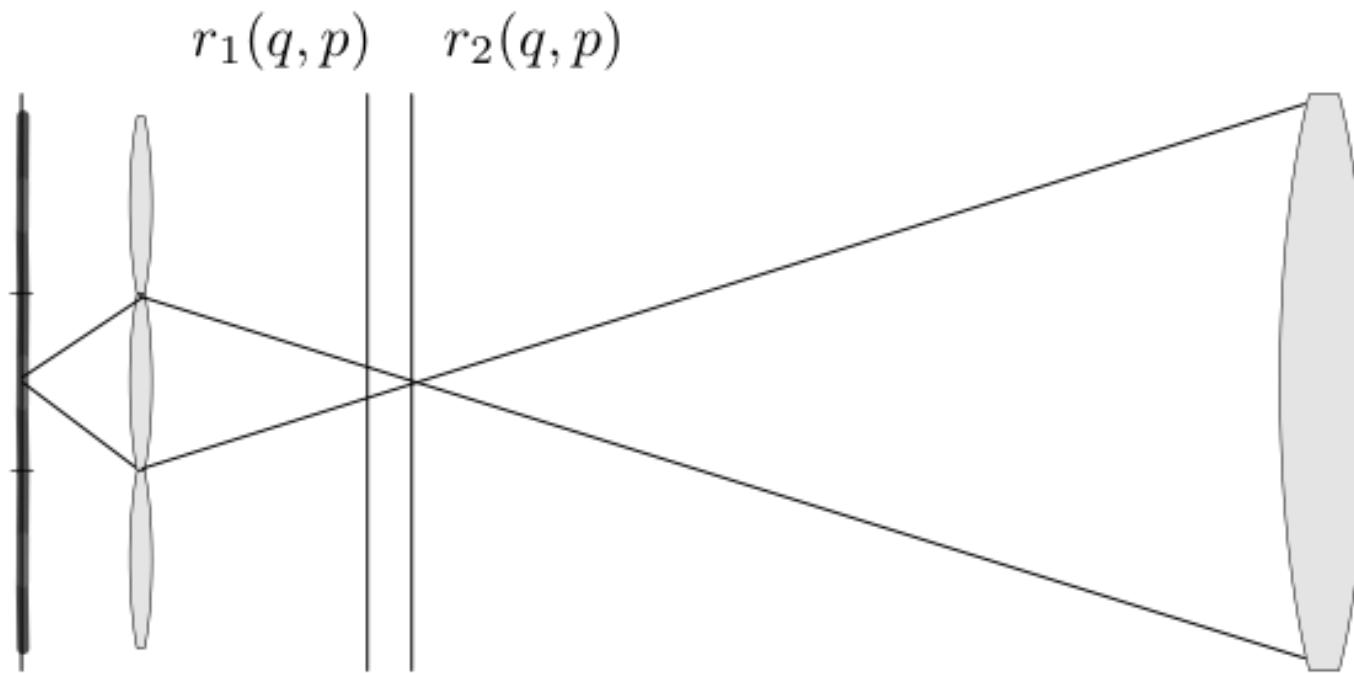
Computational Refocusing



- What does the sensor capture with different focal planes?
- What is this in terms of phase space?

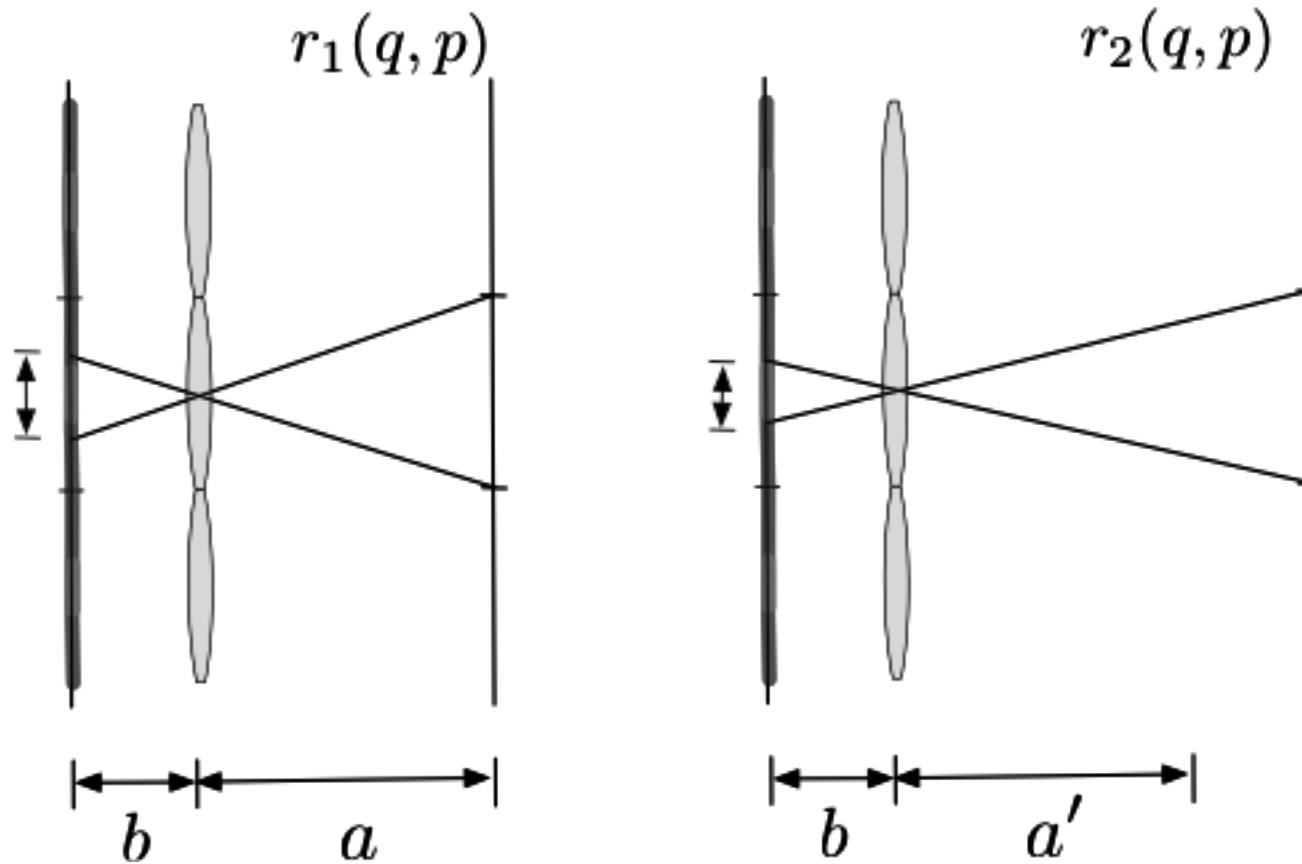
Computational Refocusing

- We capture radiance r_1 . How can we compute r_2 ?
- Apply translation transform of the radiance and render from transformed r
 - Very expensive



Plenoptic 2.0 Refocusing Principle

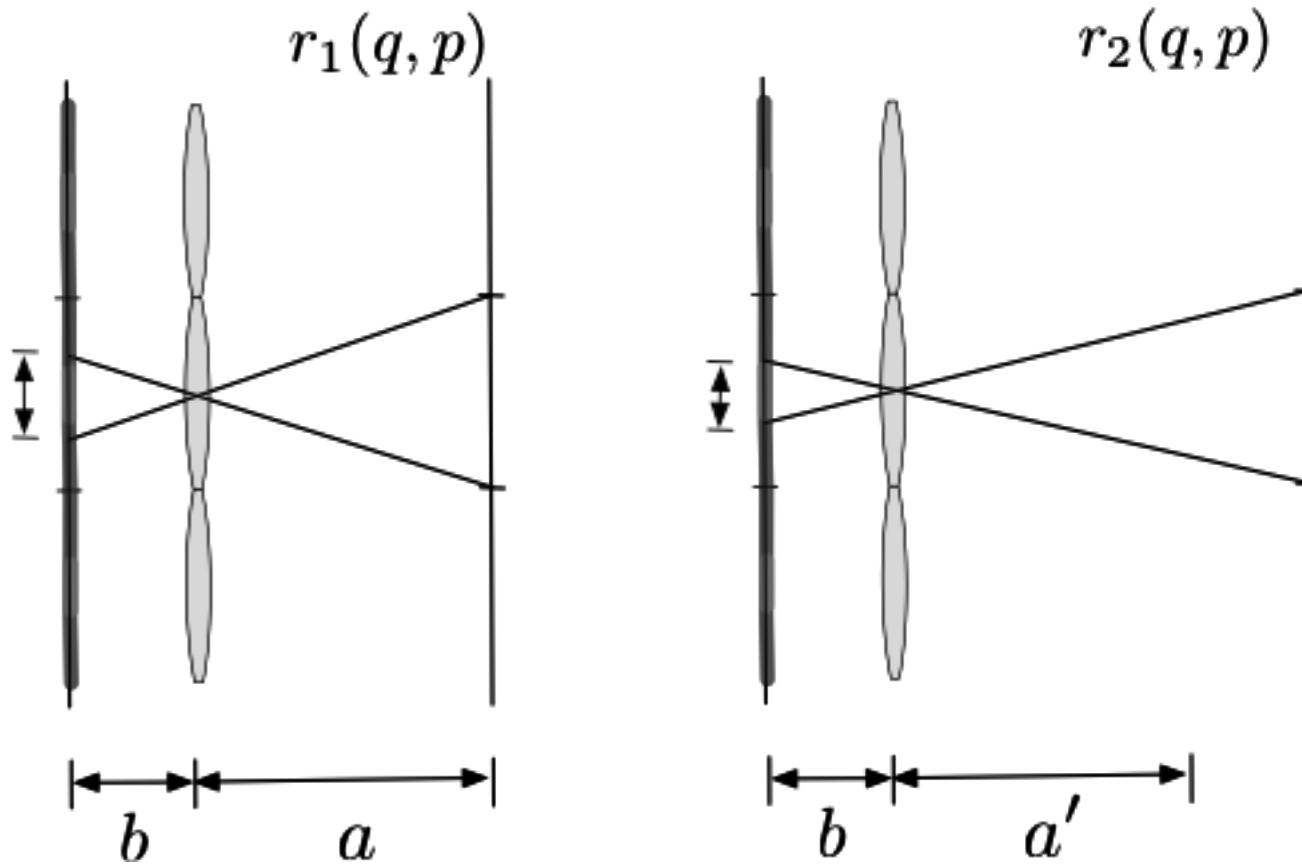
- Rendering for two different focal planes
- Comments?



Plenoptic 2.0 Refocusing Principle

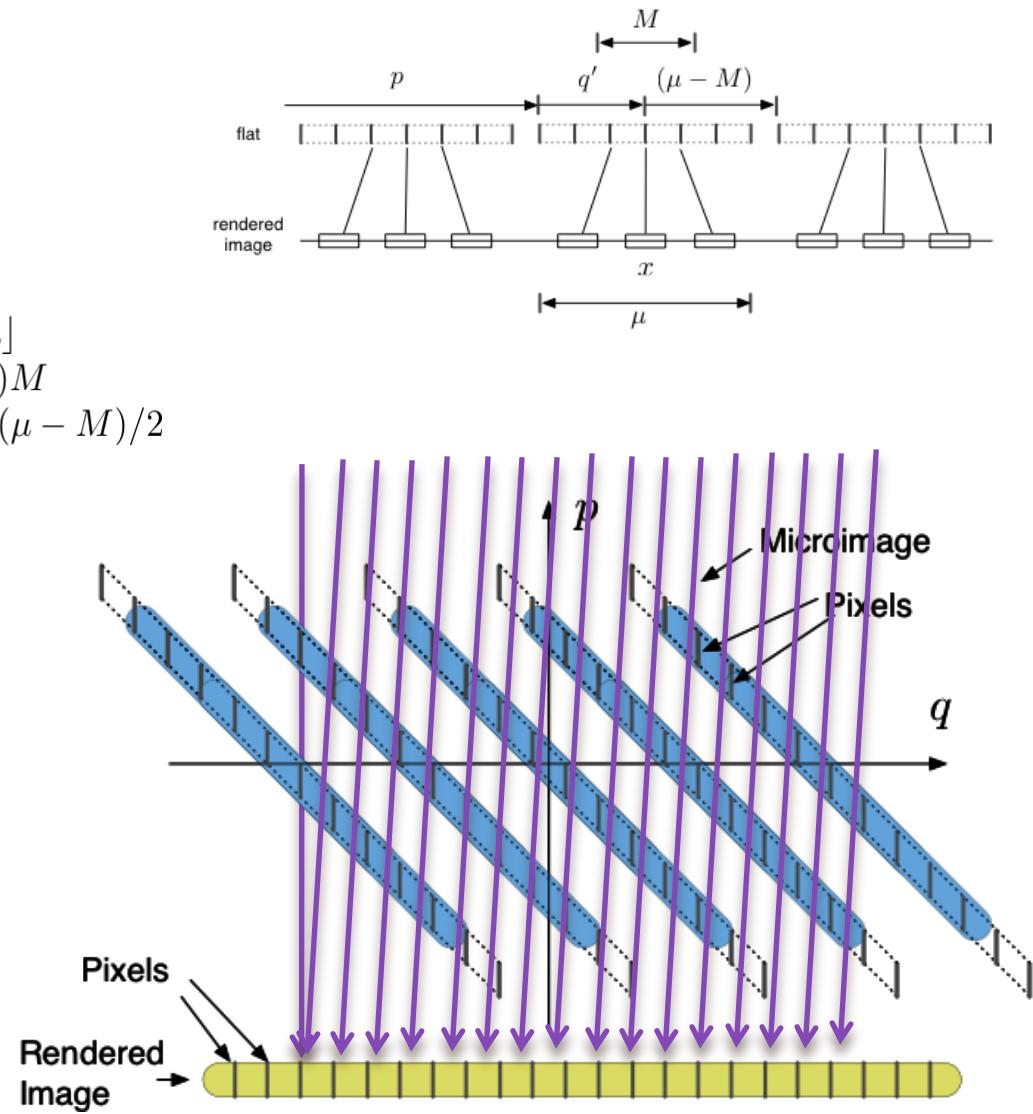
- A new focal plane can be rendered directly from original radiance

$$I_2(q) = \int_p r_2(q, p) dp = \int_p r_1(q', p') dp$$



GLSL Rendering

```
uniform sampler2DRect flat;  
  
uniform float M, mu;  
  
void main()  
{  
    vec2 x_mu = gl_TexCoord[0].st/mu; //  $x/\mu$   
    vec2 p = floor(x_mu); //  $p = \lfloor x/\mu \rfloor$   
    vec2 q = (x_mu - p) * M; //  $(x/\mu - p)M$   
    vec2 qp = q + 0.5*(mu - M); //  $q' = q + (\mu - M)/2$   
  
    vec4 colXY = vec4(0.0);  
    for (int i = -1; i <= 1; ++i) {  
        for (int j = -1; j <= 1; ++j) {  
  
            vec2 ij = vec2(float(i), float(j));  
            vec2 dq = qp - ij * M;  
            vec2 fx = (p + ij)*mu + dq;  
  
            colXY += texture2DRect(flat, fx);  
        }  
    }  
    gl_FragColor = colXY / 5.0;  
}
```



Computational Focusing



Computational Focusing



Computational Focusing



To Find Out More

- Georgiev, T., Lumsdaine, A., “Focused Plenoptic Camera and Rendering,” *Journal of Electronic Imaging*, Volume 19, Issue 2, 2010
- <http://www.tgeorgiev.net/GTC2010/>





GLSL Implementation (Basic Rendering)

```
uniform sampler2DRect flat;           // Plenoptic image

uniform float M, mu;

void main()
{
    vec2 x_mu = gl_TexCoord[0].st/mu; //  $x/\mu$ 
    vec2 p = floor(x_mu);           //  $p = \lfloor x/\mu \rfloor$ 
    vec2 q = (x_mu - p) * M;       //  $(x/\mu - p)M$ 
    vec2 qp = q + 0.5*(mu - M);   //  $q' = q + (\mu - M)/2$ 

    vec2 fx = p * mu + qp;         //  $f(x) = p\mu + q'$ 

    gl_FragColor = texture2DRect(flat, fx);
}
```