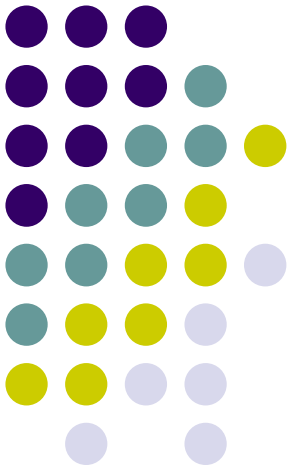
Computer Graphics (CS 563)
Lecture 2: Advanced Computer Graphics
Visual Appearance

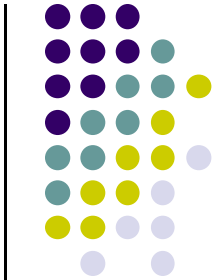
Prof Emmanuel Agu

Computer Science Dept.

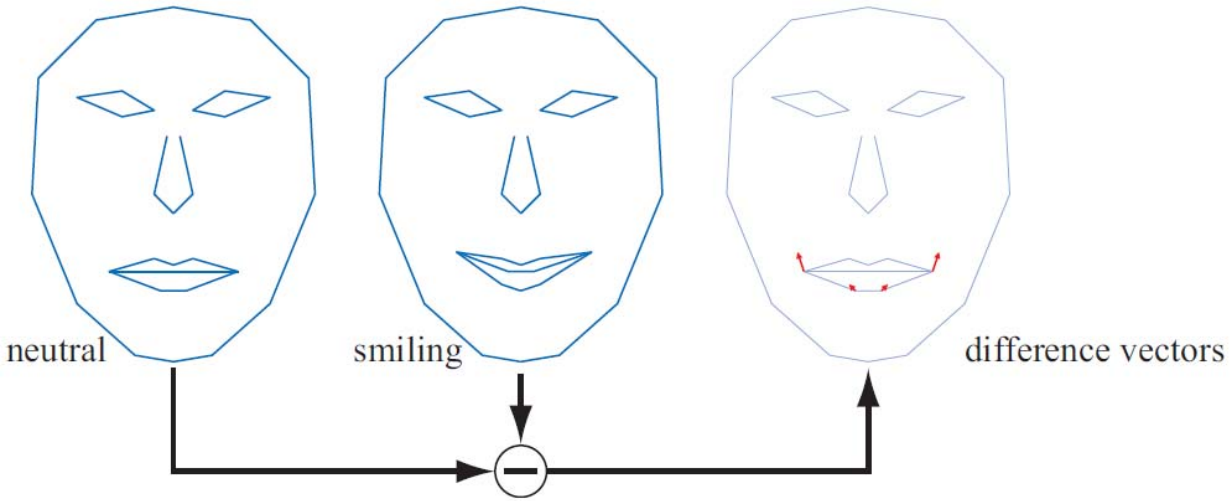
Worcester Polytechnic Institute (WPI)



RTR Chapter 4: Vertex Morphing



- Chapter 4 reviews previously covered transforms (Scale, rotate, translate, etc)
- Now describe vertex morphing of meshes
- For each vertex, (**start** and **end**) positions are defined
- Calculate set of difference vectors



Vertex Morphing

- For each frame calculate + render intermediate position
- Linear vertex blend equation

$$\boldsymbol{m} = (1 - s)\boldsymbol{p}_0 + s\boldsymbol{p}_1$$

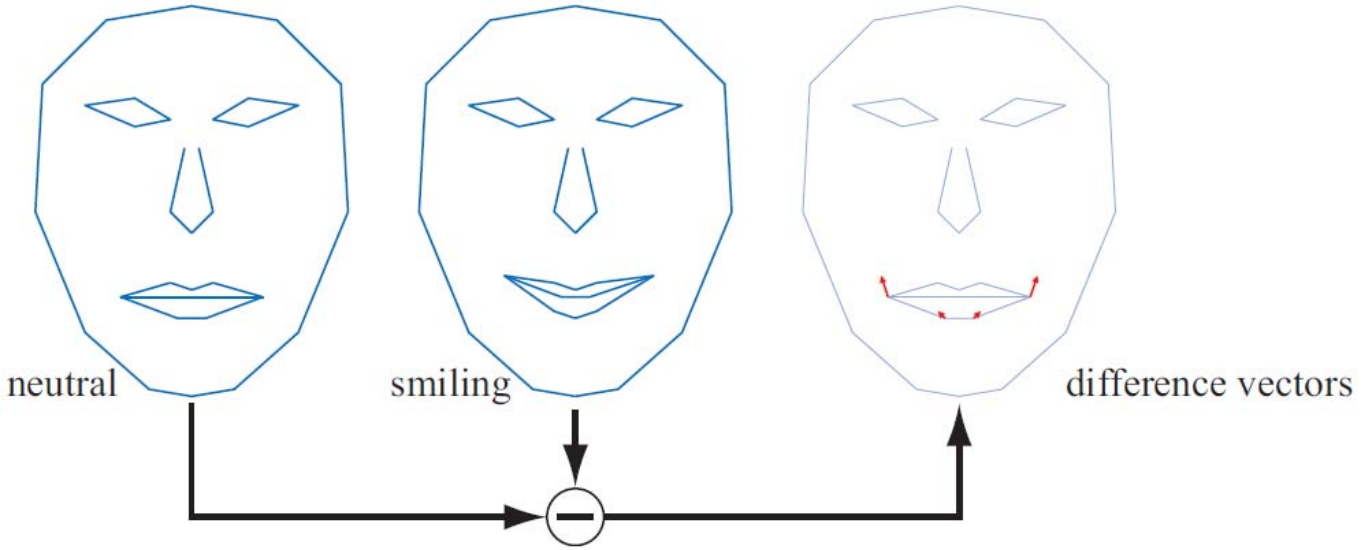
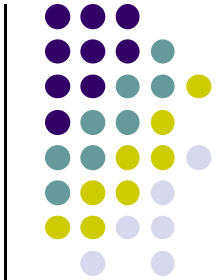
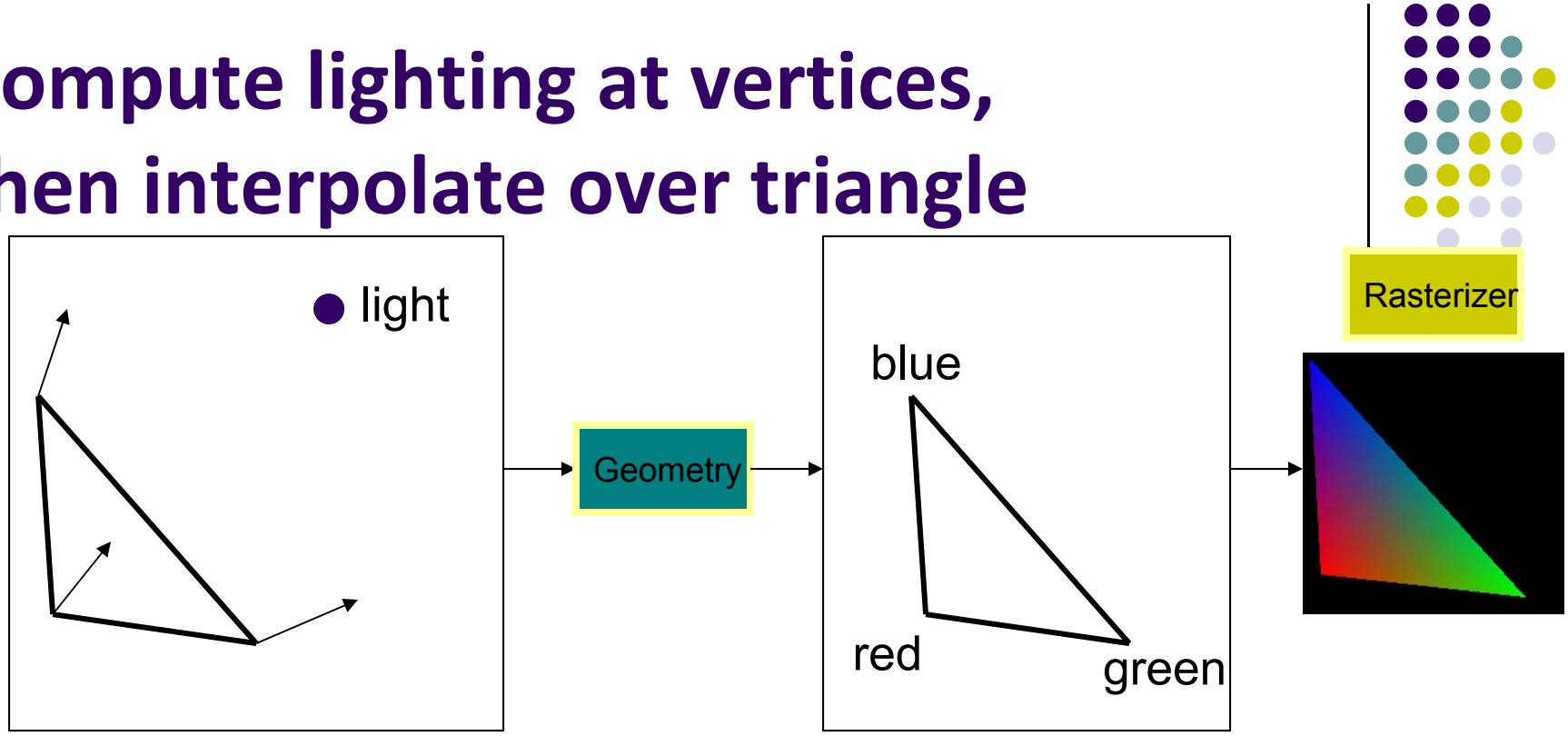


Image Morphing



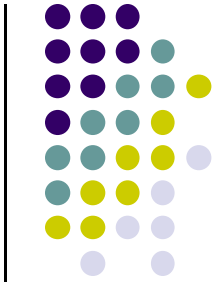
- Mark similar points on the images (e.g. nose)
- Distort nose position + fade image 1 into image 2

Compute lighting at vertices, then interpolate over triangle

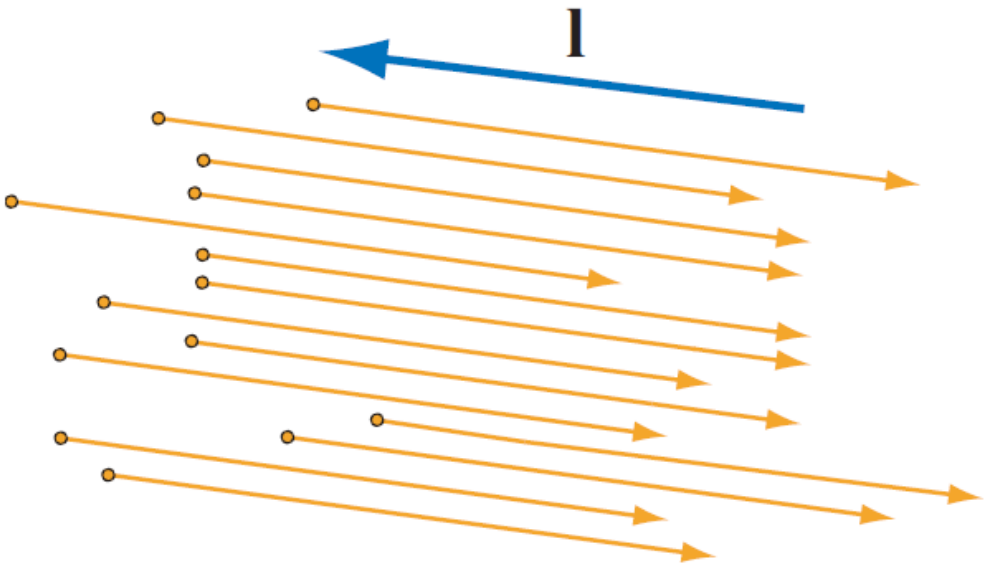


- How compute lighting?
- We could set colors per vertex manually
- For a **little** more realism, compute lighting from
  - Light sources
  - Material properties
  - Geometrical relationships

Light Sources

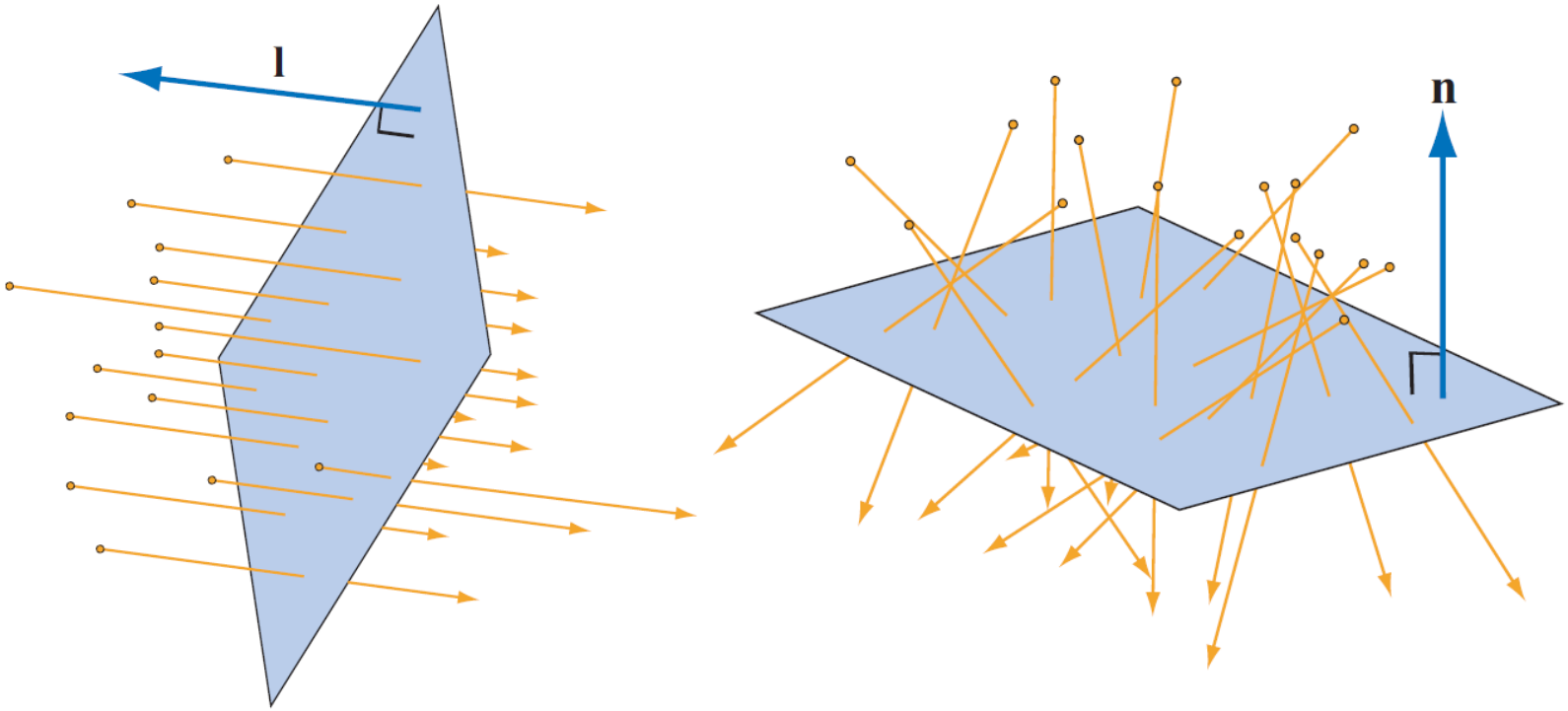


- Light vector: Opposite to direction of light travel



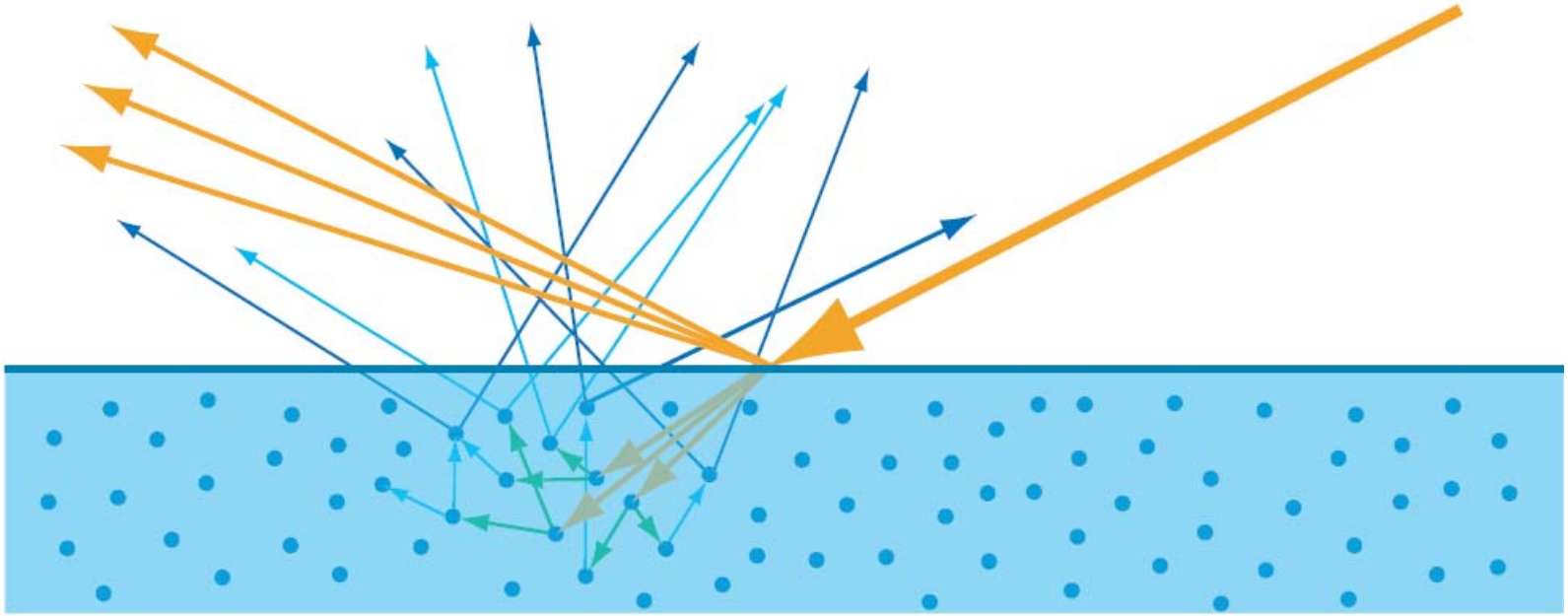
Light Sources

- **Irradiance:** sum of energies of photons passing through surface in one second
- Irradiance measures light from all incoming directions



Materials

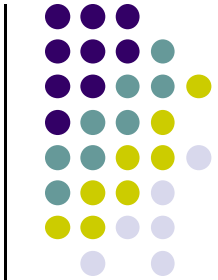
- Some light reflected, some refracted



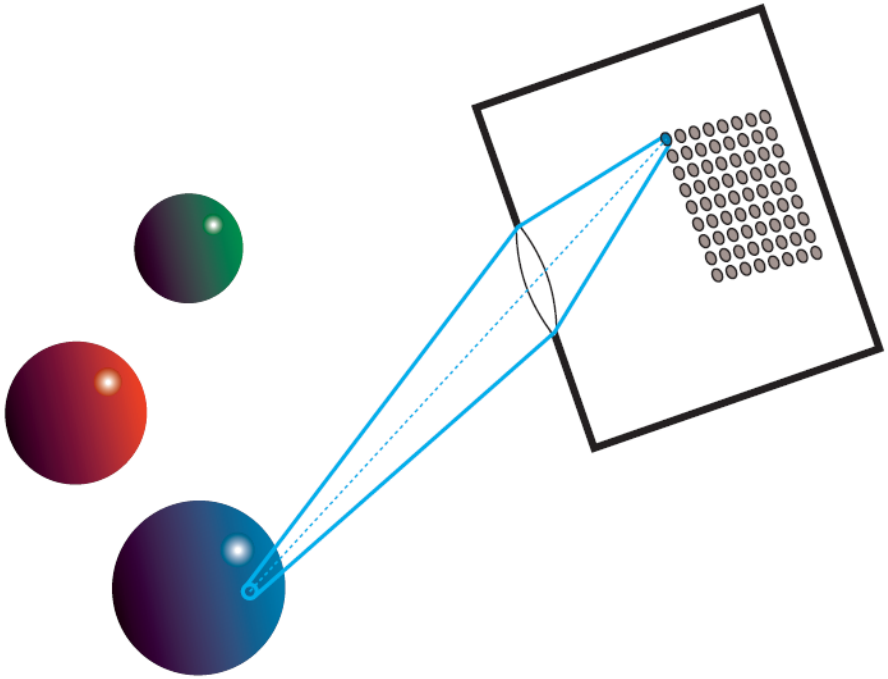
Material: Smoothness

- Shinier material focusses light in narrow beam

Sensors

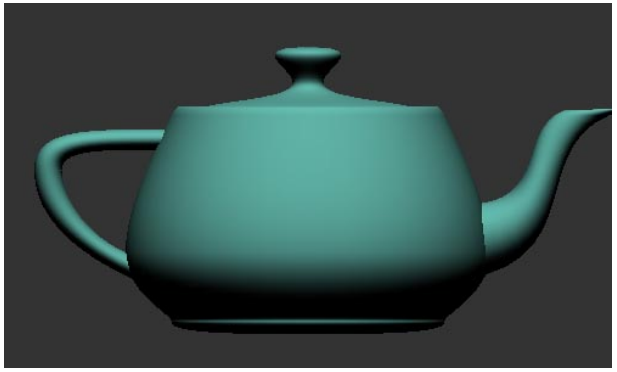


- Light emanating from scene objects detected by sensor (camera, eye, or film)
- Sensors measure **radiance** (brightness and color of single ray of light)



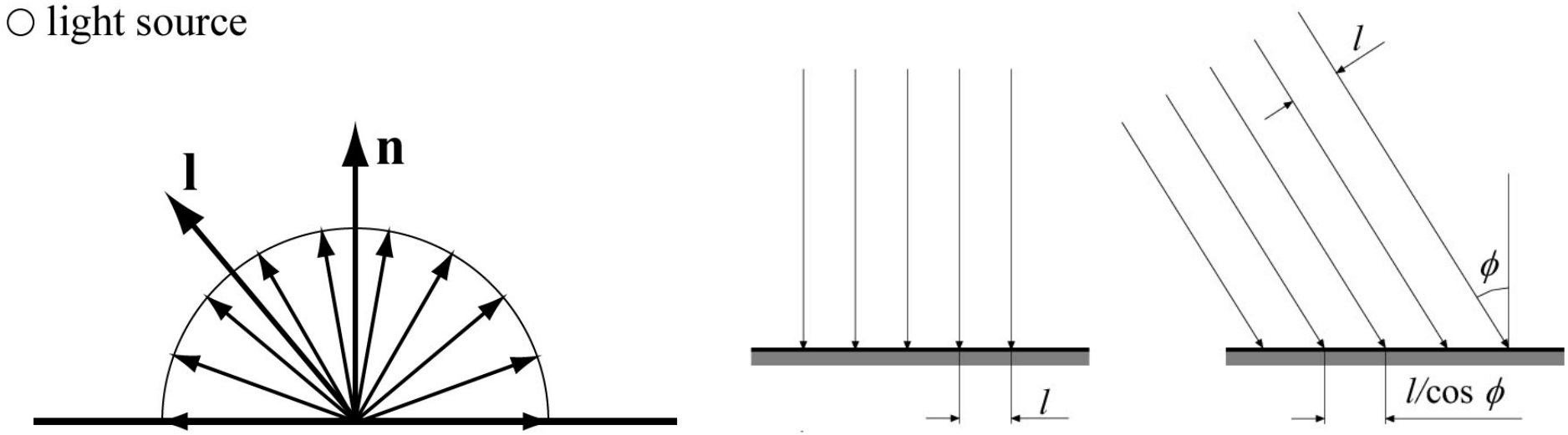
Refresher on lighting

Diffuse component : $\mathbf{i}_{diff}$

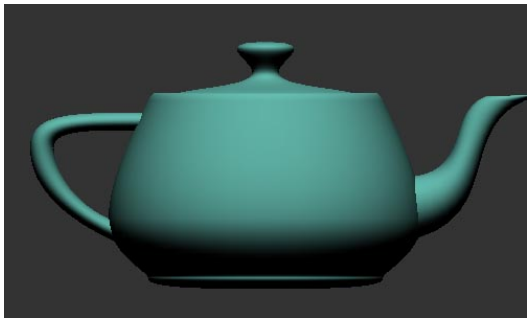


- $\mathbf{i}=\mathbf{i}_{amb}+\mathbf{i}_{diff}+\mathbf{i}_{spec}$
- Diffuse is Lambert's law: $i_{diff} = \mathbf{n}\cdot\mathbf{l} = \cos\phi$
- Photons are scattered equally in all directions

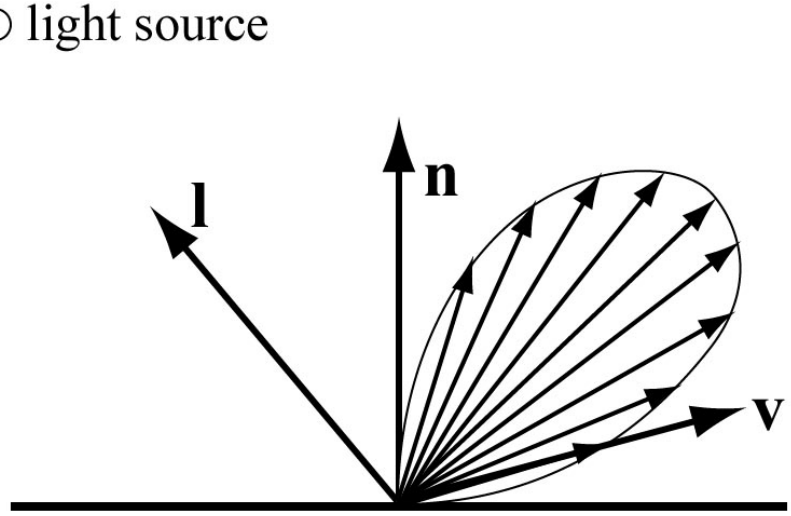
$$\mathbf{i}_{diff} = (\mathbf{n}\cdot\mathbf{l})\mathbf{m}_{diff} \otimes \mathbf{s}_{diff}$$



Lighting
Specular component : i_{spec}




- Diffuse is dull (left)
- Specular: simulates a highlight

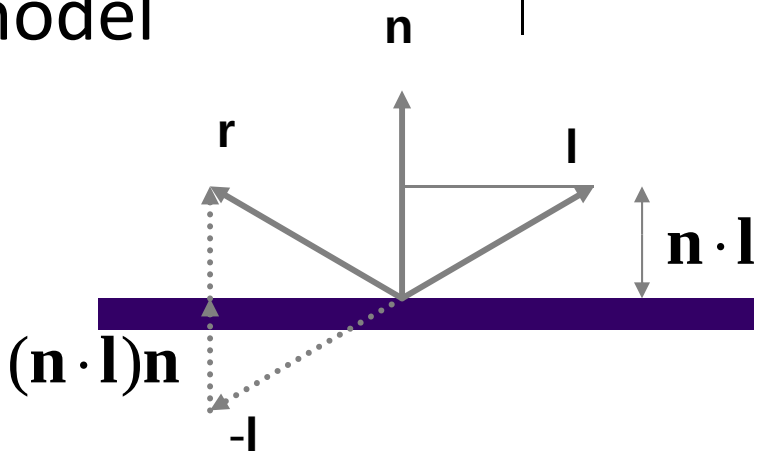


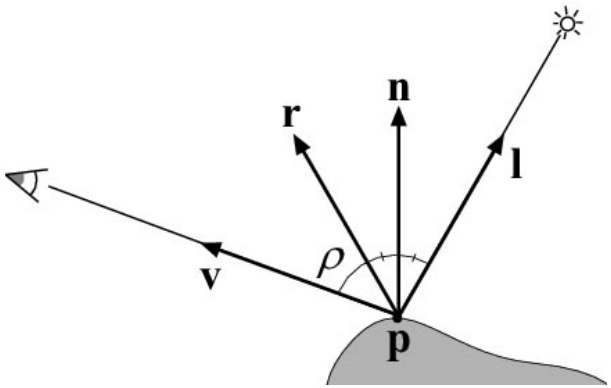
Specular component: Phong

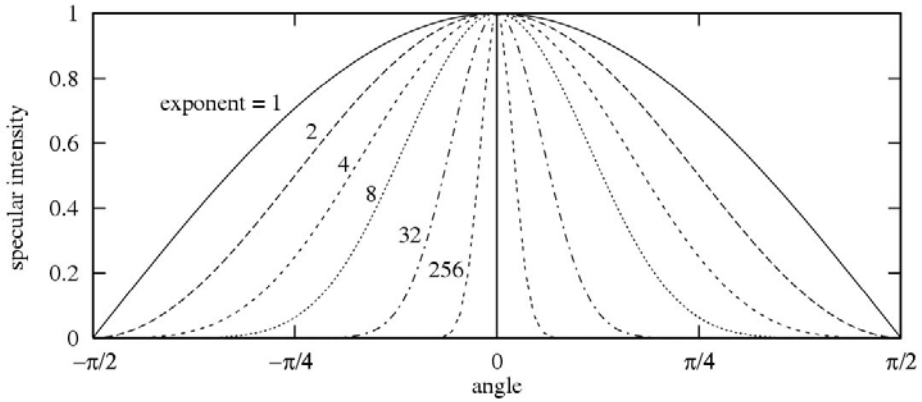
- Phong specular highlight model
- Reflect **l** around **n**:

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

$$i_{spec} = (\mathbf{r}\cdot\mathbf{v})^{m_{shi}} = (\cos\rho)^{m_{shi}}$$



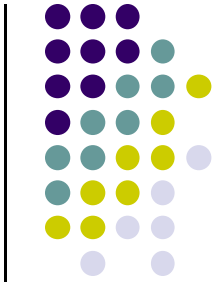




$$\mathbf{i}_{spec} = \max(0,(\mathbf{r}\cdot\mathbf{v})^{m_{shi}})\mathbf{m}_{spec} \otimes \mathbf{s}_{spec}$$

- Can also use Blinns highlight formula: $(n\cdot h)^m$

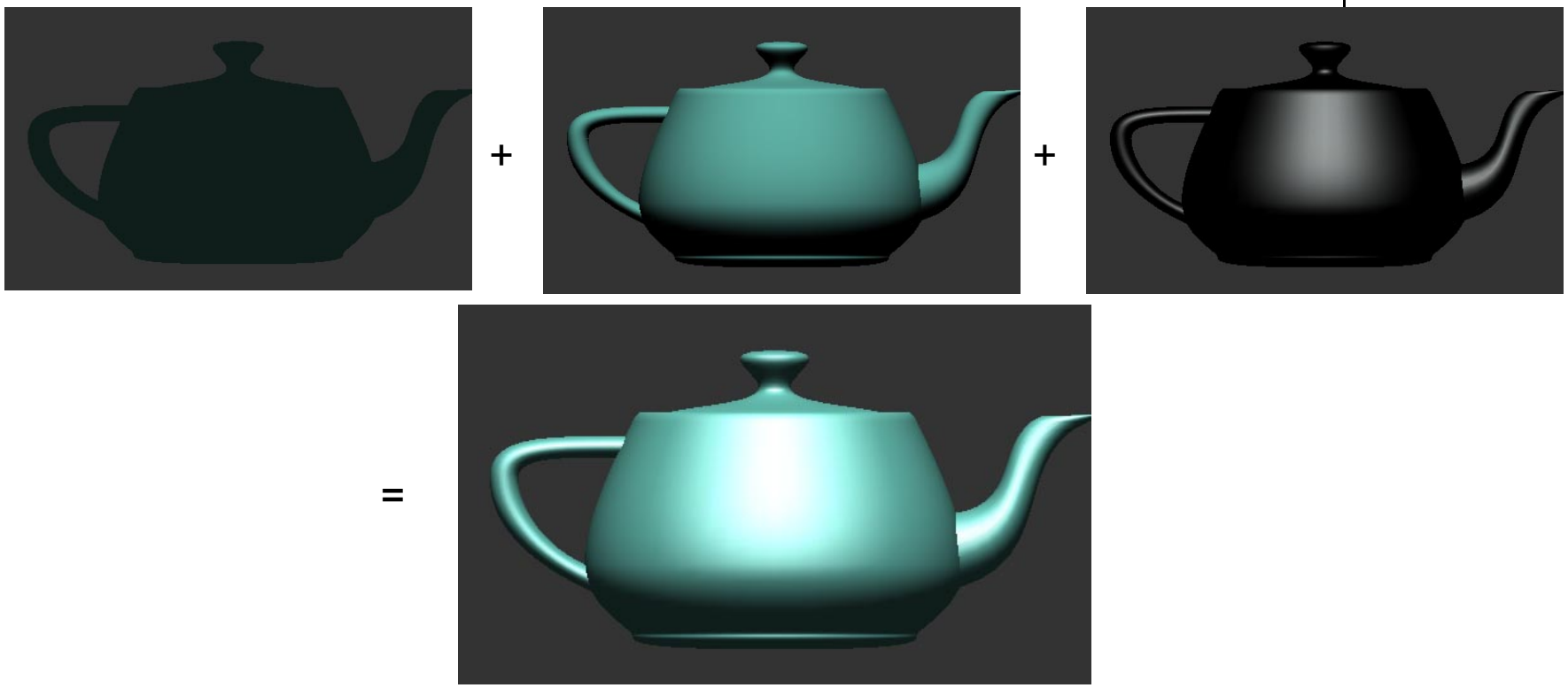
Ambient component: $\mathbf{i}_{amb}$



- Ad-hoc – tries to account for light coming from other surfaces
- Just add a constant color:

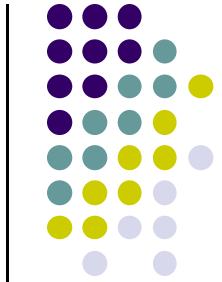
$$\mathbf{i}_{amb} = \mathbf{m}_{amb} \otimes \mathbf{s}_{amb}$$

Lighting: $i = i_{amb} + i_{diff} + i_{spec}$

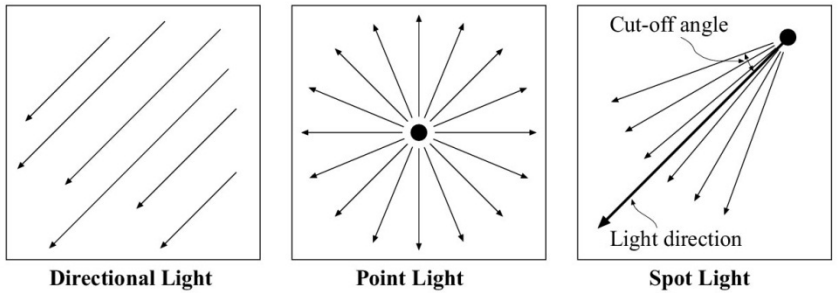


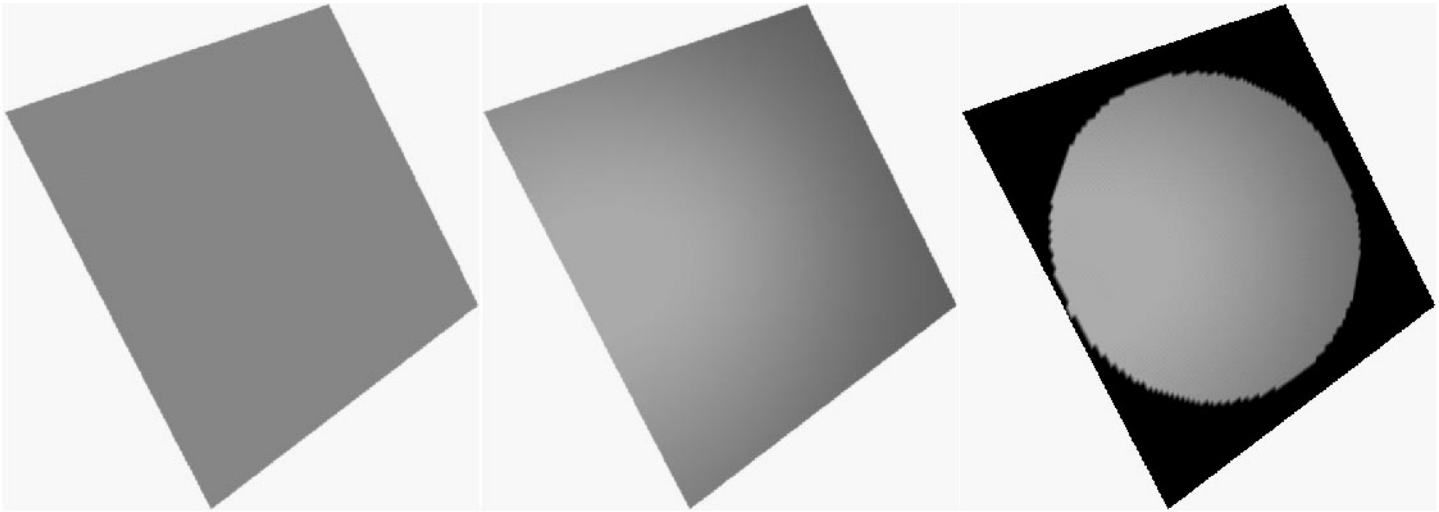
- This is just a hack!
- Has little to do with how reality works!

Additions to the lighting equation

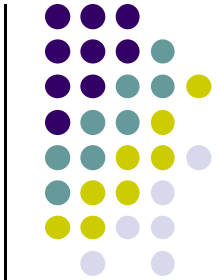


- Depends on distance: $1/(a+bt+ct^2)$
- Can have more lights: just sum their respective contributions
- Different light types:

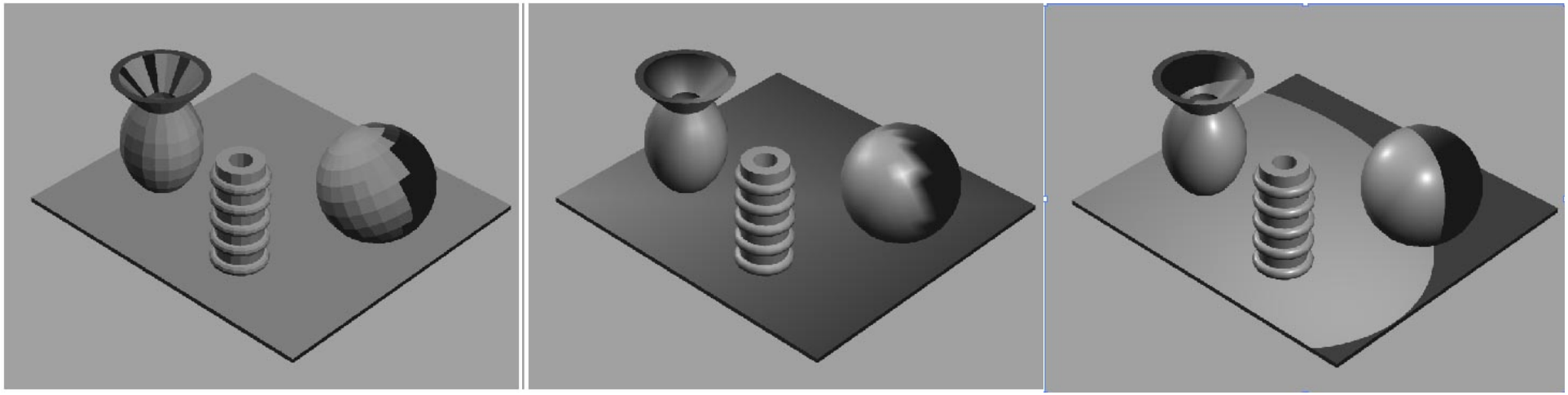




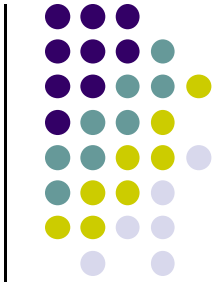
What's lighting and what's shading?



- Lighting: the interaction between light and matter
- Shading: do lighting (at vertices) and determine pixel's colors from these
- Three types of shading:
  - Flat, Goraud, and Phong

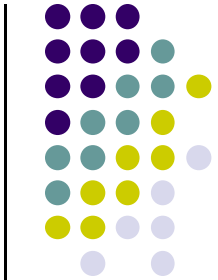


Aliasing and Antialiasing



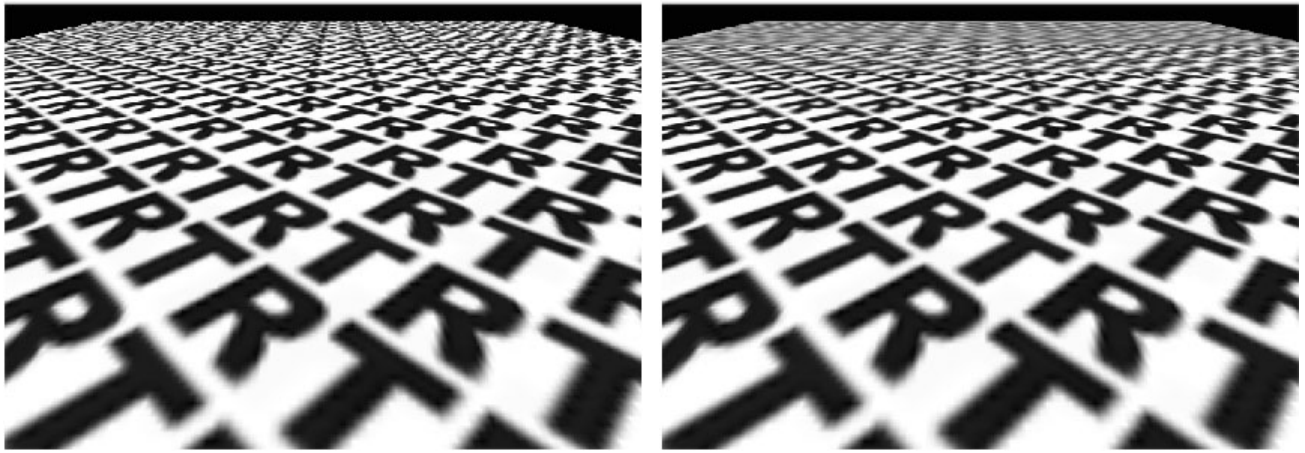
- Why care at all? Quality!!
- Example: Final fantasy
  - The movie against the game
  - In a broad way, and for most of the scenes, the only difference is in the number of samples and the quality of filtering

Computer graphics is a SAMPLING & FILTERING process!

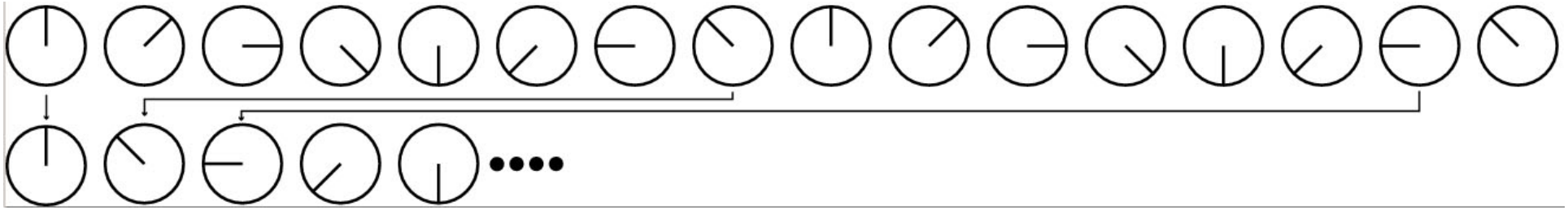


- Pixels

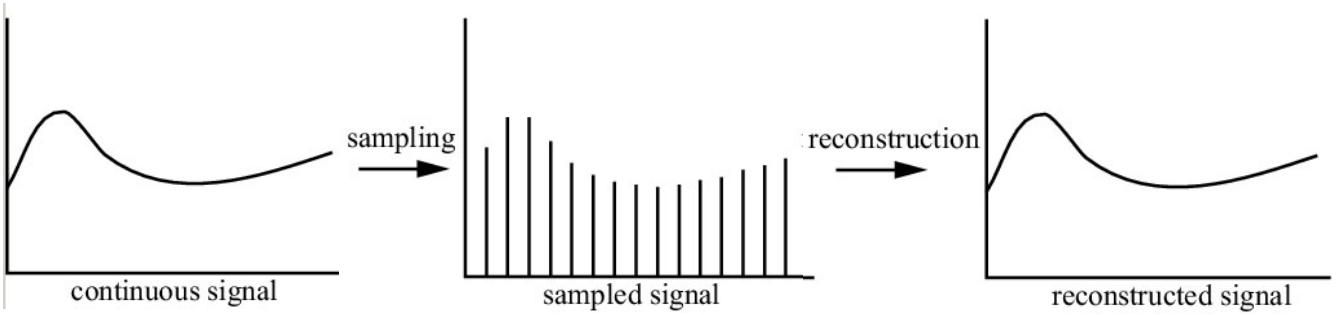
- Texture



- Time



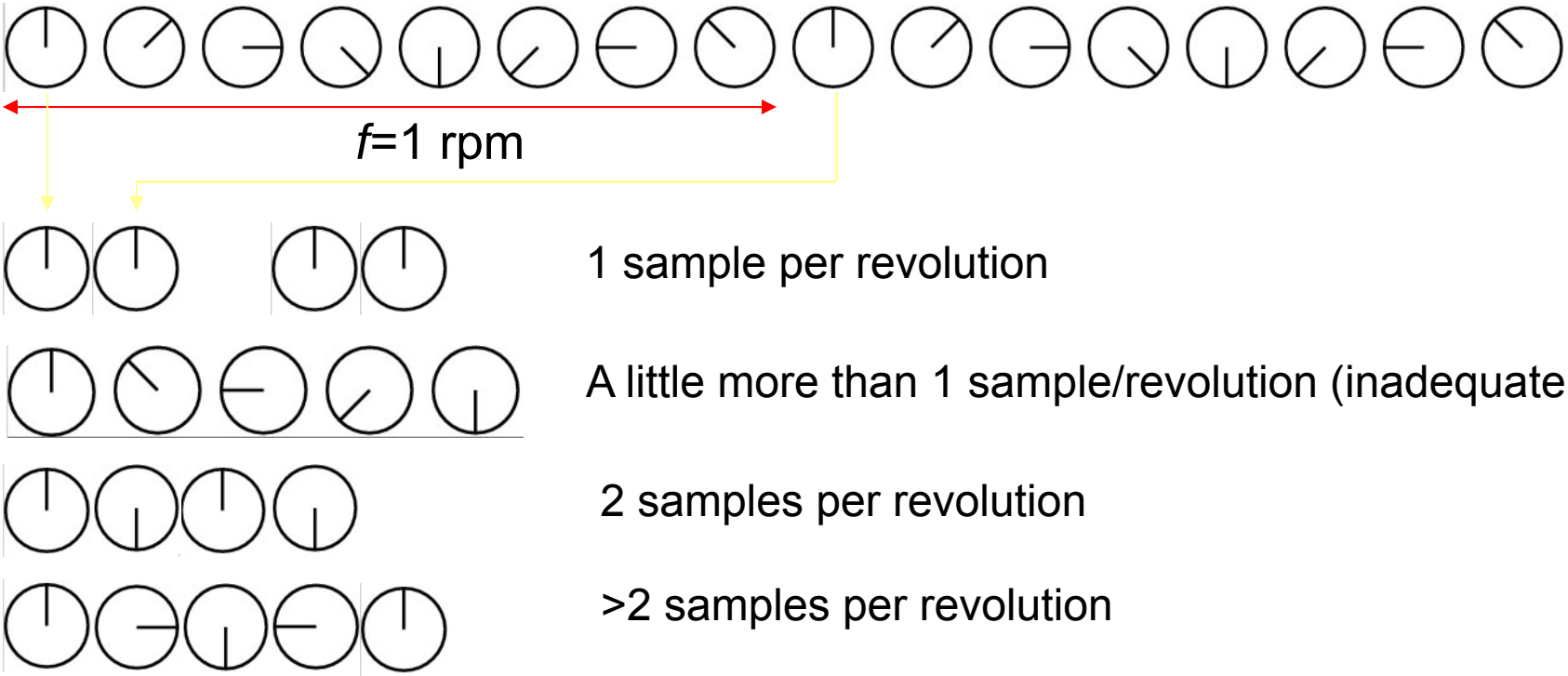
Sampling and reconstruction



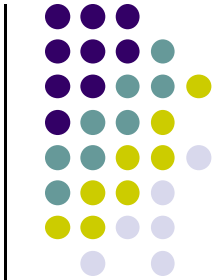
- Sampling: from continuous signal to discrete
- Reconstruction recovers the original signal
- Care must be taken to avoid aliasing
- Nyquist theorem: *the sampling frequency should be at least 2 times the max frequency in the signal*
- Often impossible to know max frequency (bandlimited signal), or the max frequency is often infinite…

Sampling theorem

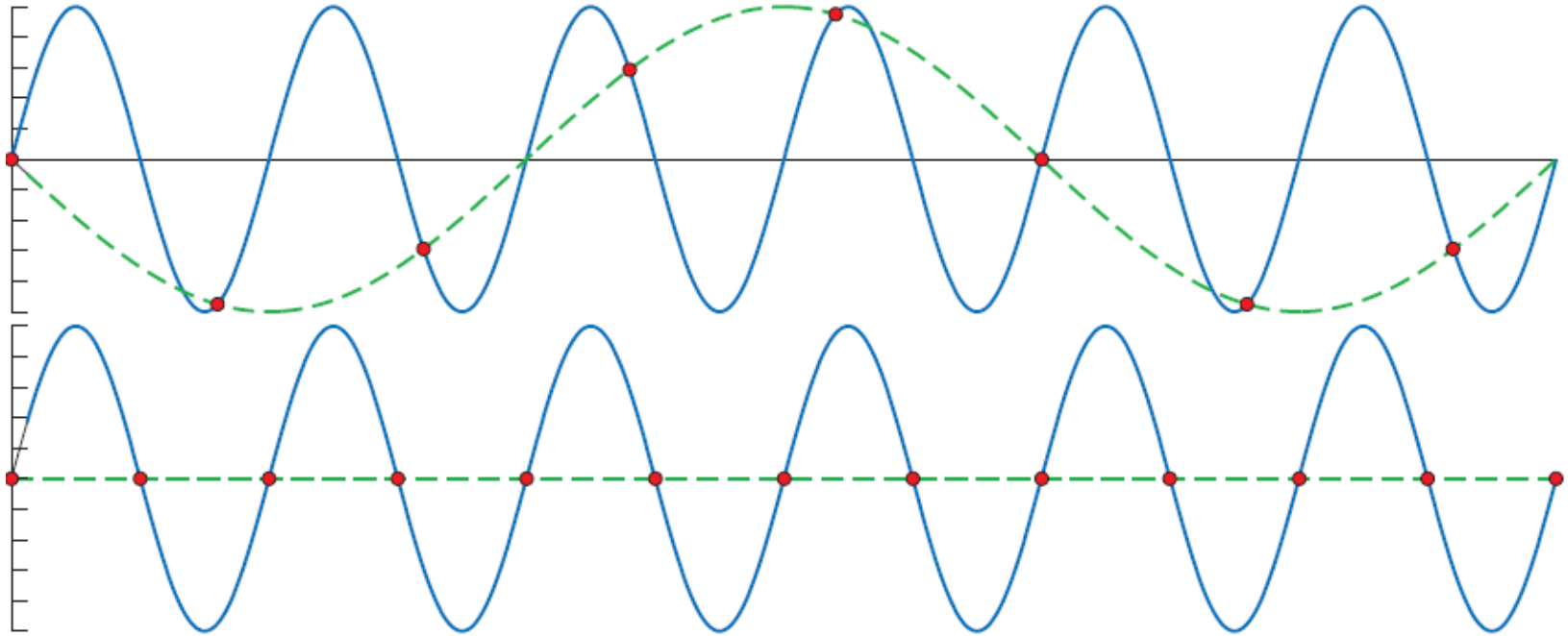
- Nyquist theorem: *the sampling frequency should be at least 2 times the max frequency in the signal*



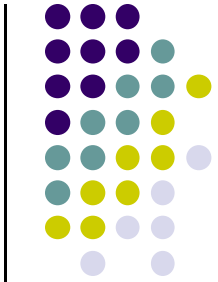
Sampling Rates



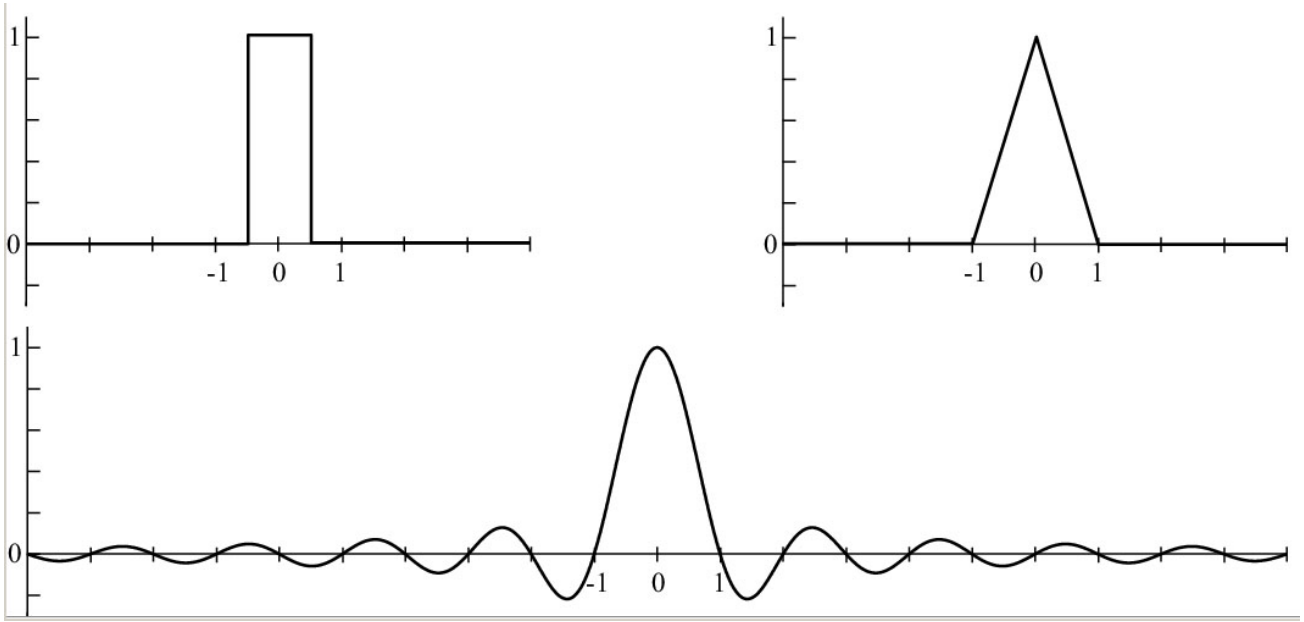
- **Top:** low sampling rate makes signal appear low frequency
- **Bottom:** 2x sampling rate recovers the correct signal



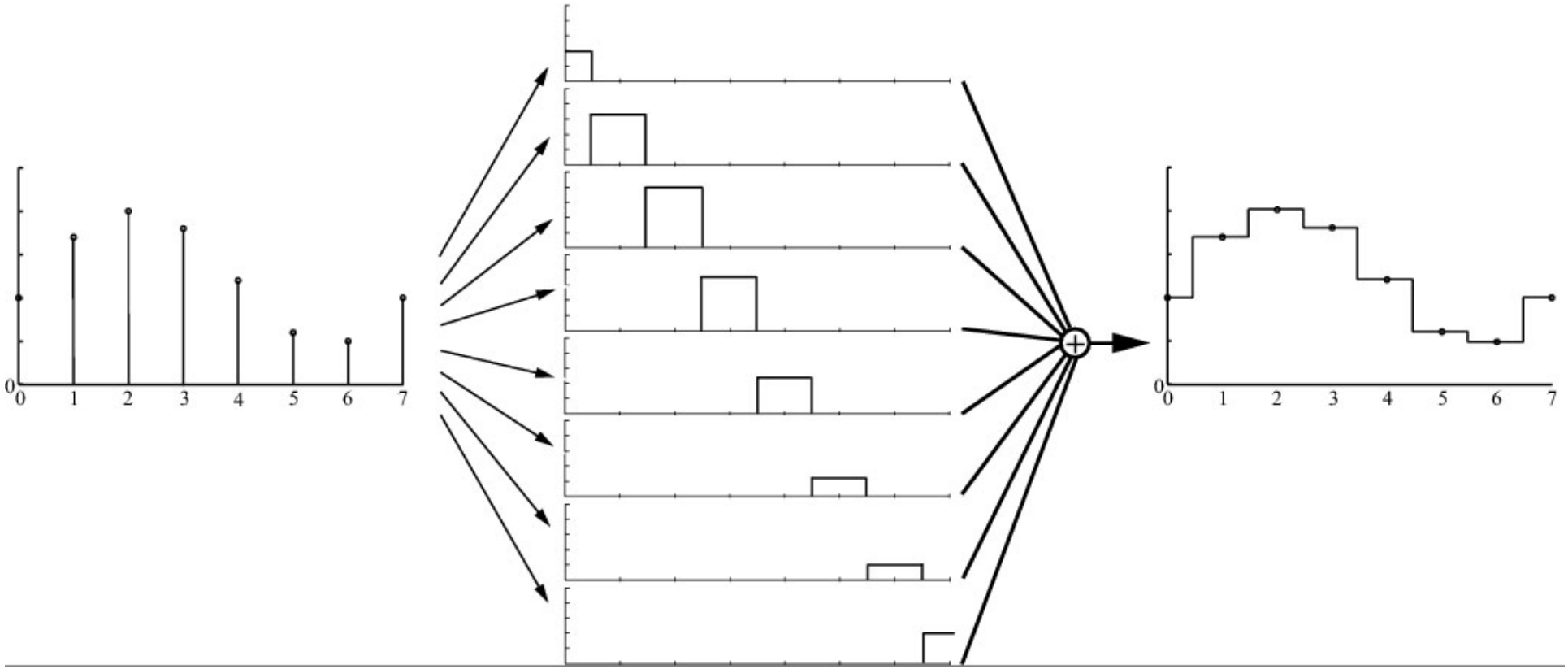
Sampling is simple, now turn to: Reconstruction



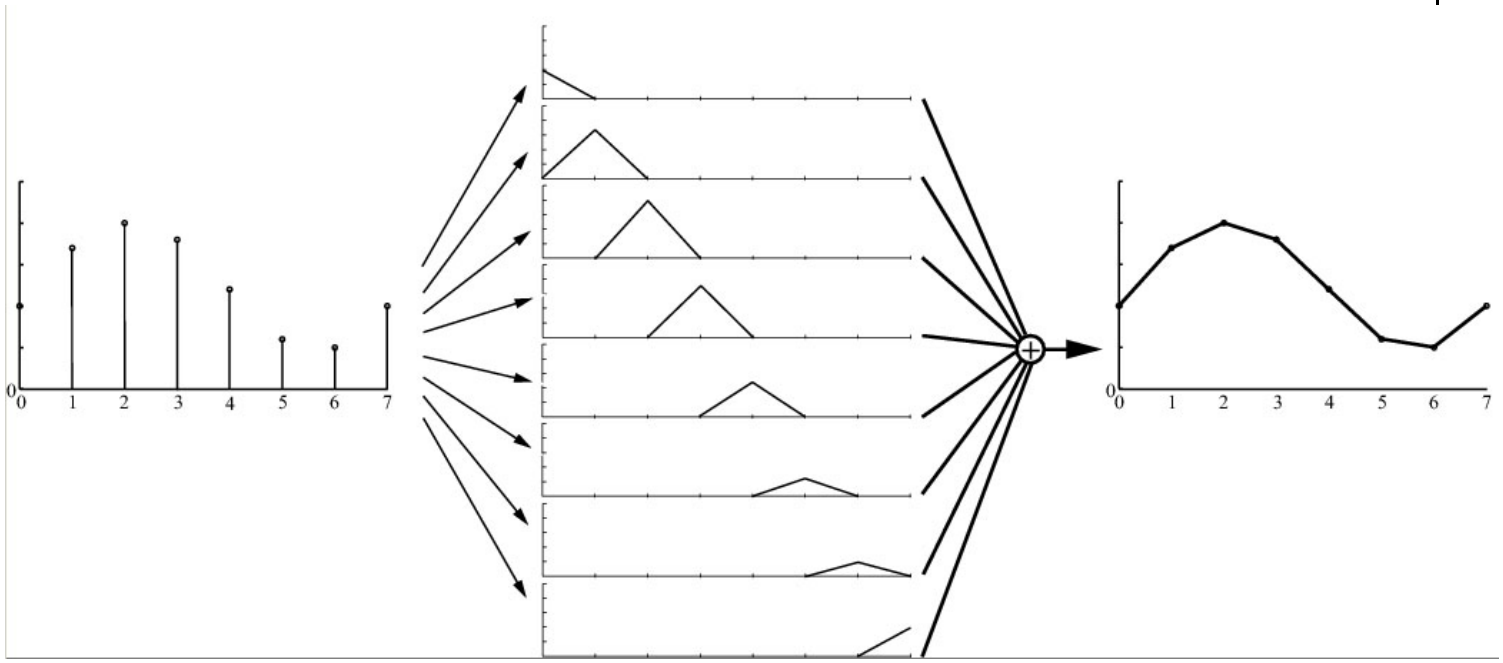
- Assume we have a bandlimited signal (e.g., a texture)
- Use filters for reconstruction



Reconstruction with box filter (nearest neighbor)



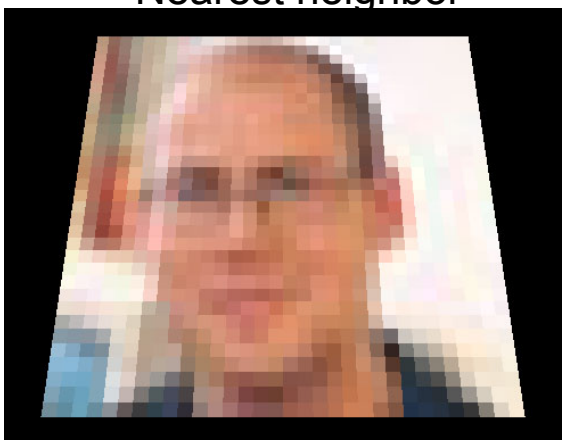
Reconstruction with tent filter



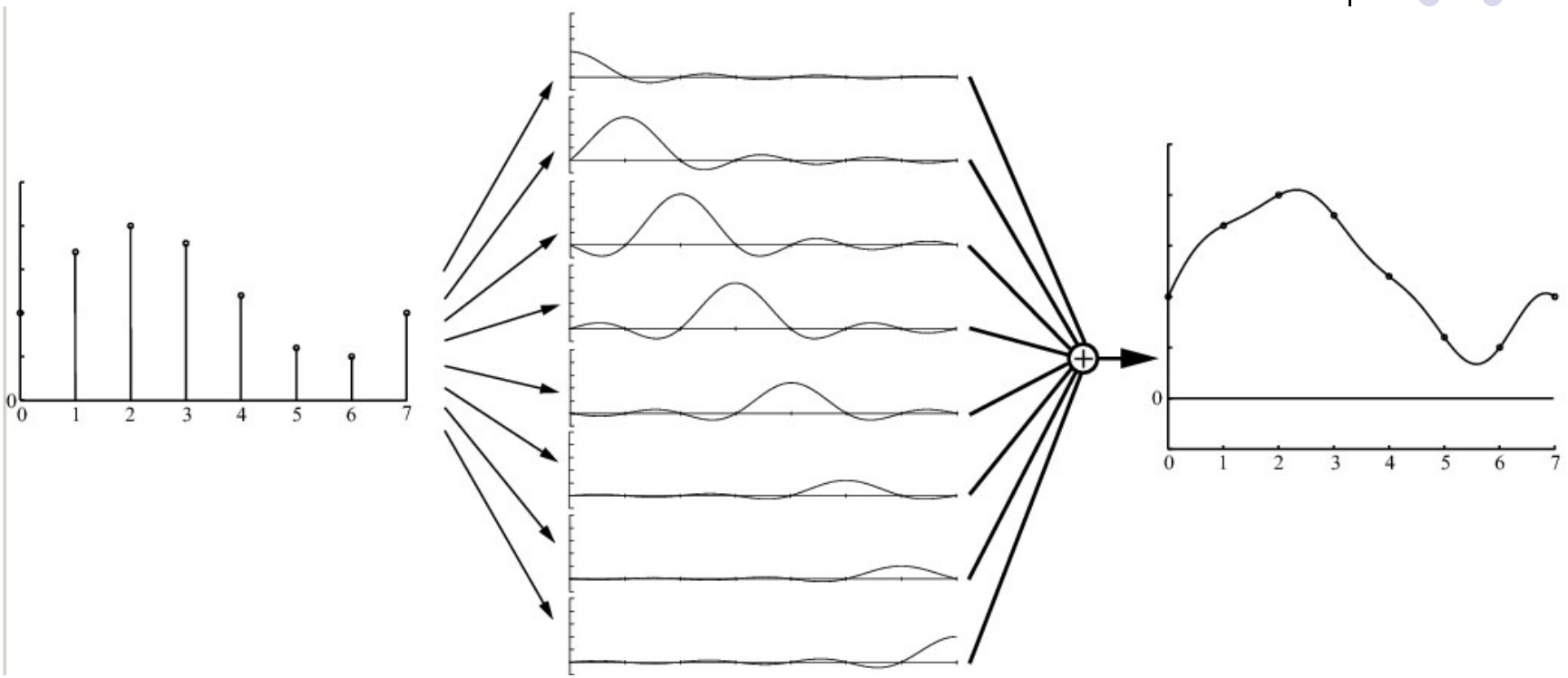
Nearest neighbor

Linear

32x32 texture

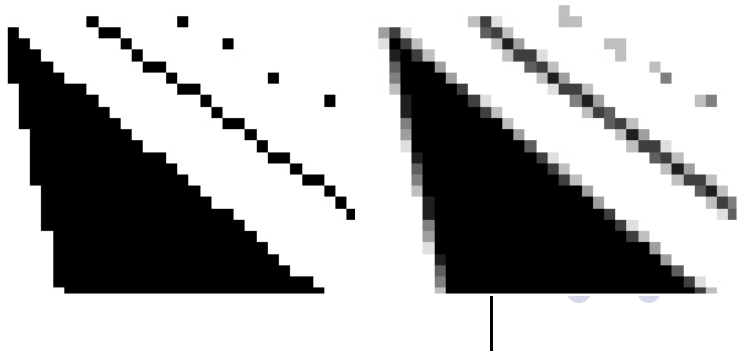


Reconstruction with sinc filter

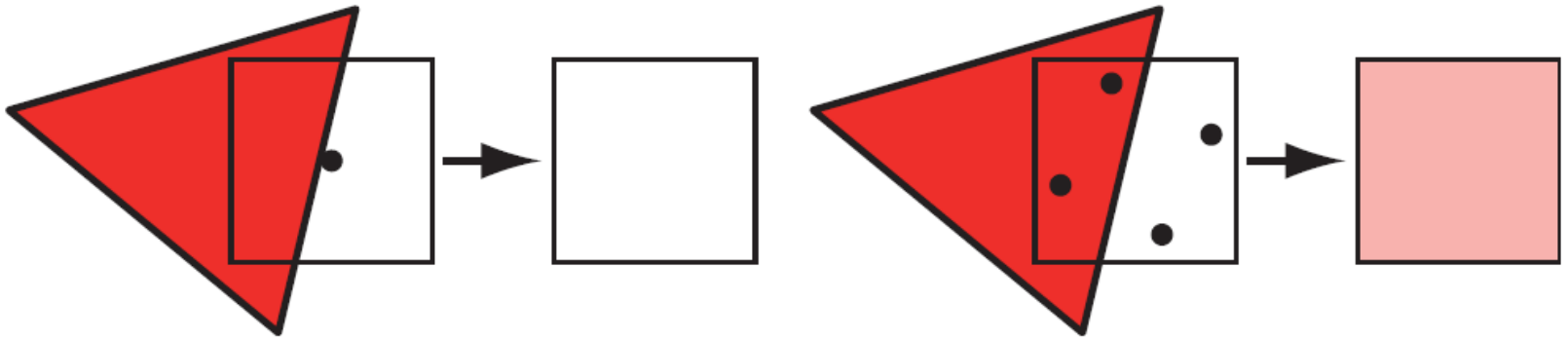


- In theory, the ideal filter
- Not practical (infinite extension, negative)

Screen-based Antialiasing



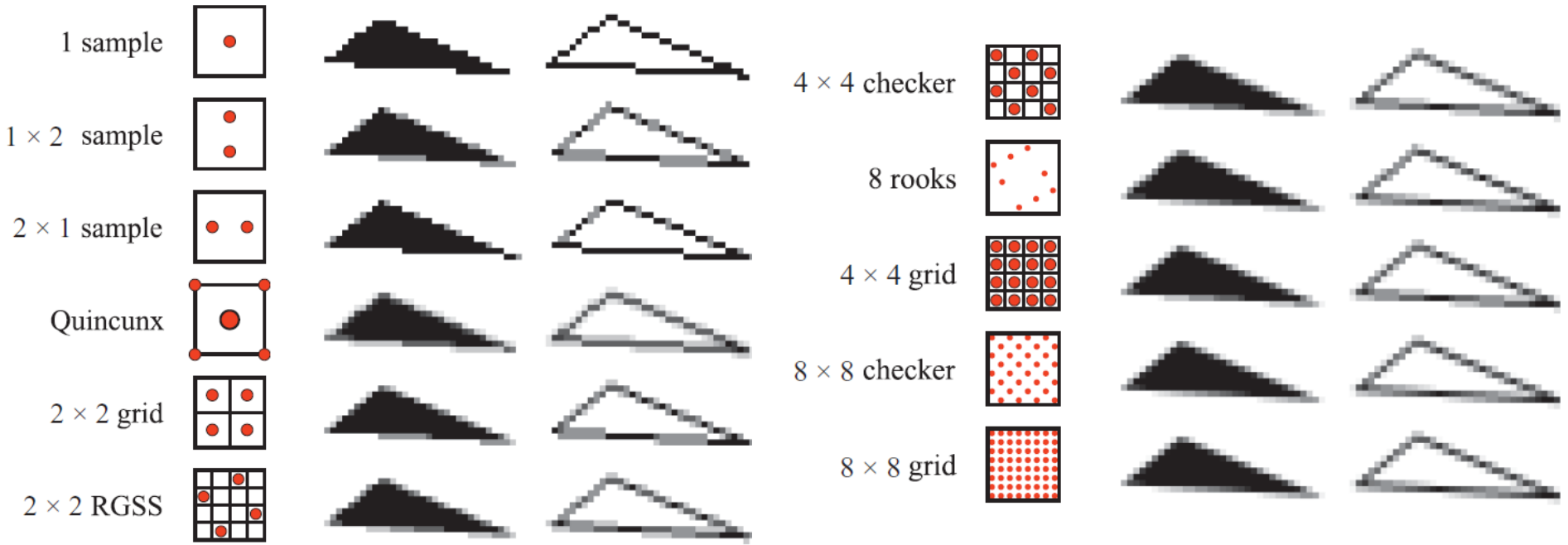
- Hard case: edge has infinite frequency
- Supersampling: use more than one sample per pixel



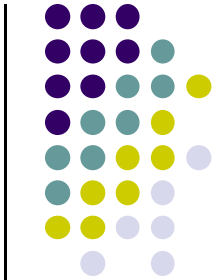
Formula and… examples of different patterns

$$\mathbf{p}(x, y) = \sum_{i=1}^{n} w_i \mathbf{c}(i, x, y)$$

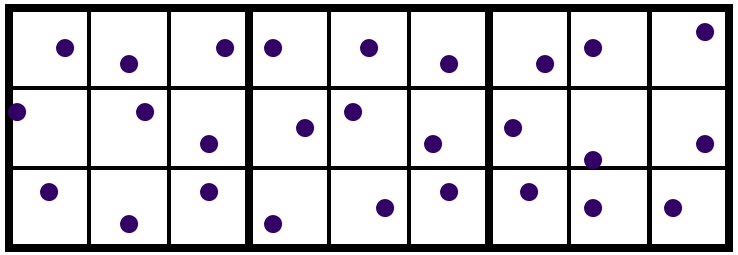
- w_i are the weights in [0,1]
- $\mathbf{c}(i,x,y)$ is the color of sample i inside pixel



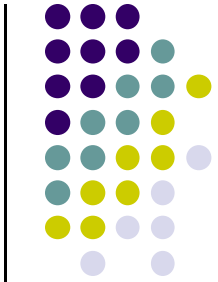
Jittered sampling



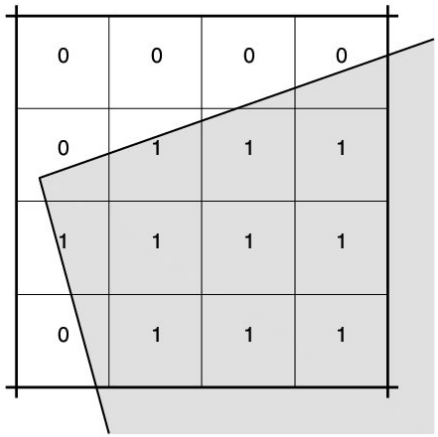
- Regular sampling cannot eliminate aliasing – only reduce it!
- Why? Because edges represent infinite frequency
- Better to use Jittering
- Jittering replaces aliasing with noise
- Example:



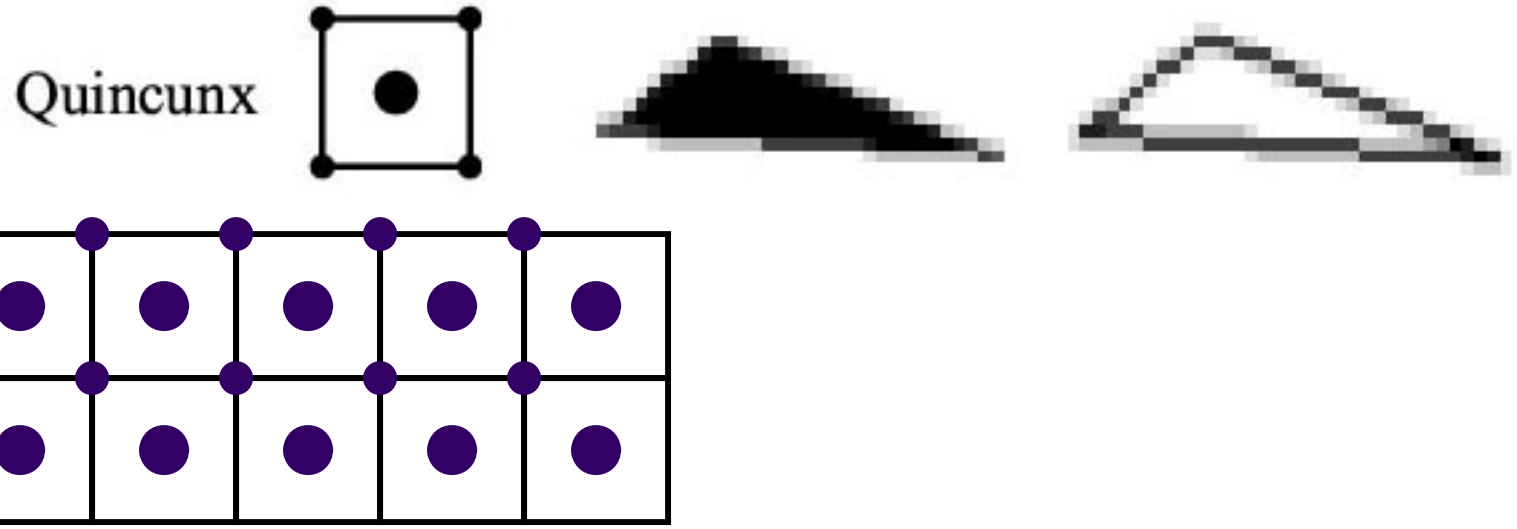
The A-buffer
Multisampling technique



- Takes >1 samples per pixel, and shares compuations between samples inside a pixel
- To deal better with edges: use coverage mask for rendering per pixel
- Coverage mask, depth, & color make up a fragment

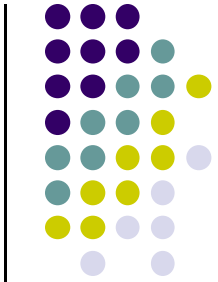


Another multisampling techniqe Quincunx

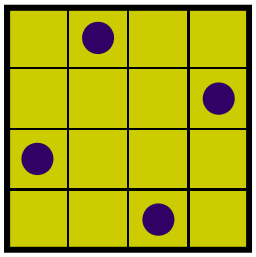


- Generate 2 samples per pixel at the same time
- mid pixel = 0.5 weight, corner pixels = 4*0.125=0.5)
- Is available on NVIDIA GeForce3 and up

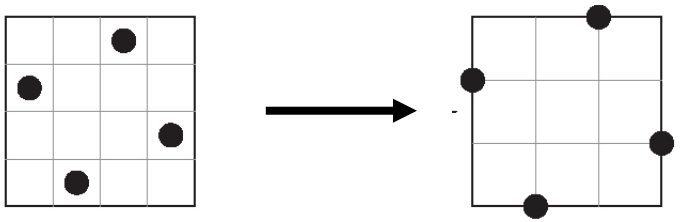
Yet another scheme: FLIPQUAD multisampling



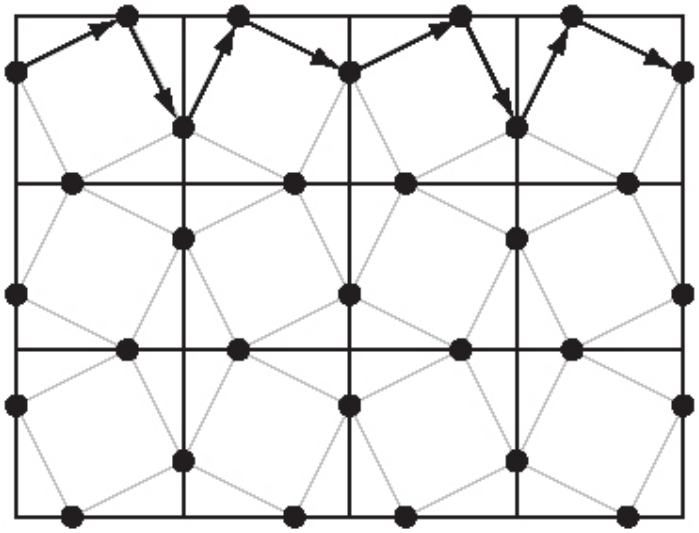
- Recap, RGSS:
  - One sample per row and column



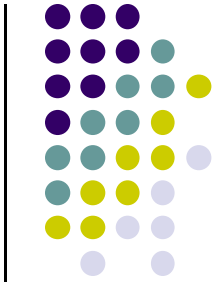
- Combine good stuff from RGSS and Quincunx



- Weights: 0.25 per sample
- Performs better than Quincunx



Fog



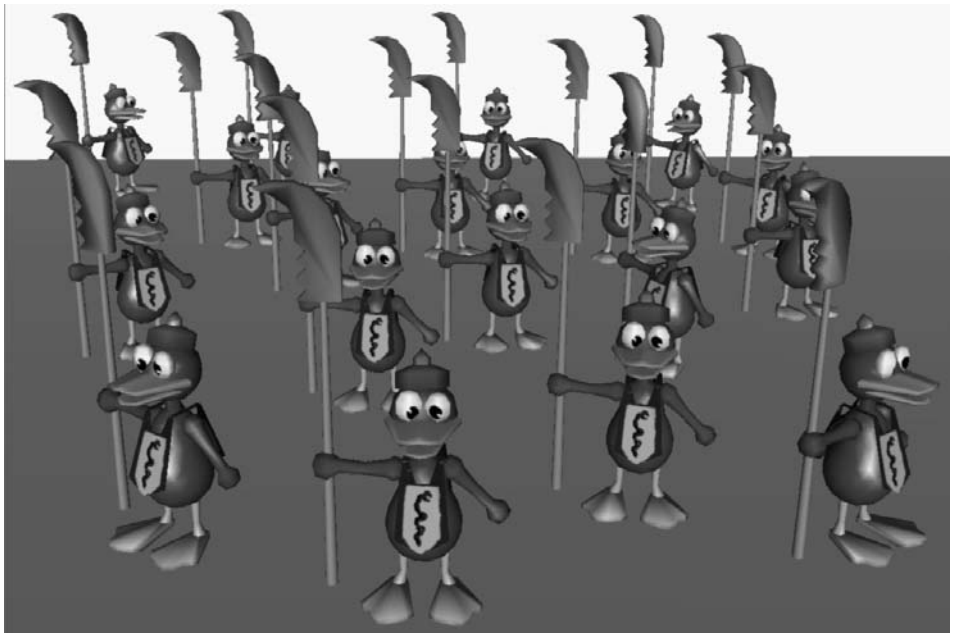
- Simple atmospheric effect
  - A little better realism
  - Help in determining distances
- Color of fog: $\mathbf{c}_f$ color of surface: $\mathbf{c}_s$

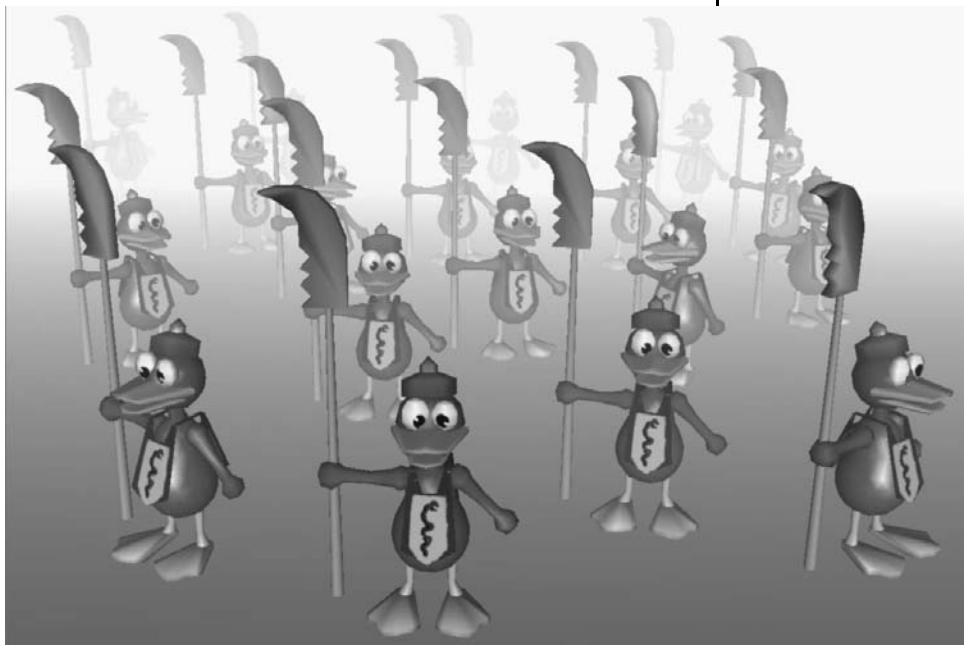
$$\mathbf{c}_p = f\mathbf{c}_f + (1-f)\mathbf{c}_s \qquad f \in [0,1]$$

- How to compute *f* ?
- 3 ways: linear, exponential, exponential-squared
- Linear:

$$f = \frac{z_{end} - z_p}{z_{end} - z_{start}}$$

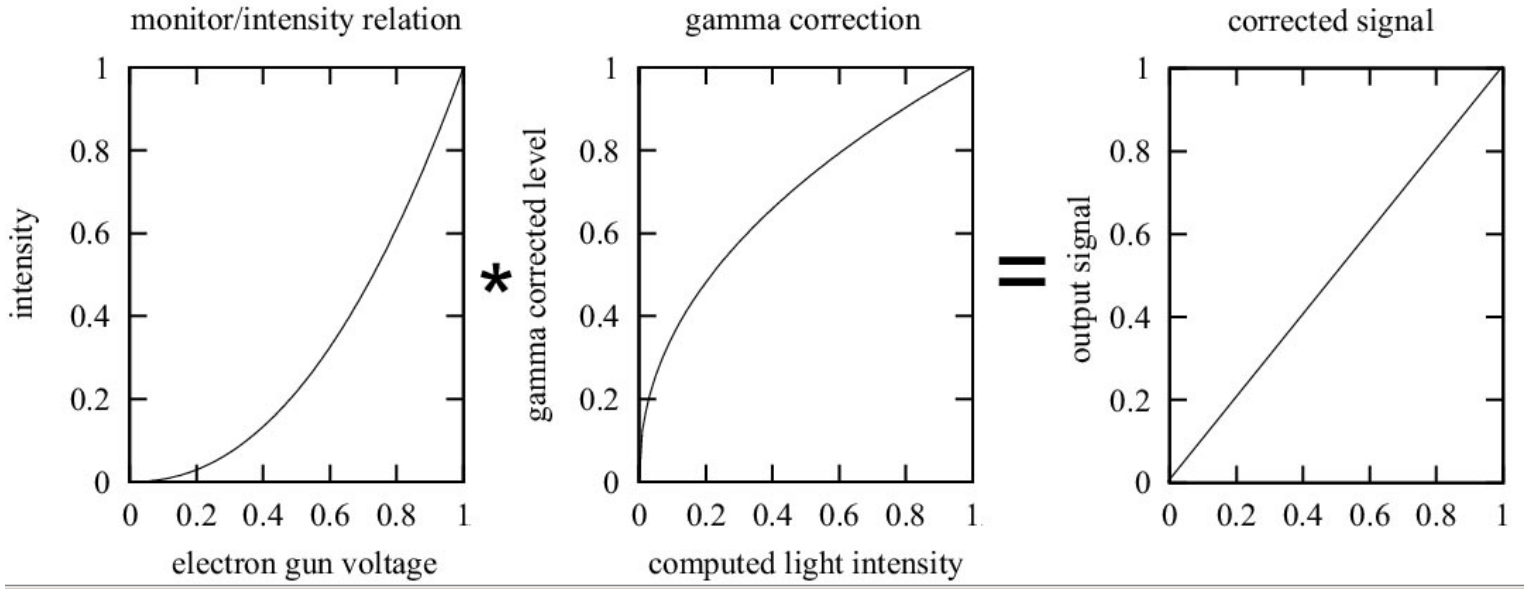
Fog example





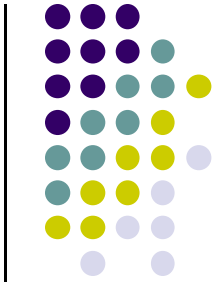
- Often just a matter of
  - Choosing fog color
  - Choosing fog model
  - Turning it on

Gamma correction



- Monitor intensity is non-linear
- If input to gun is 0.5, then you don't get 0.5 as intensity
- Need to gamma correct signal: gives linear relationship

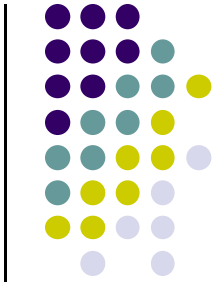
Gamma correction



$$I = a(V + \varepsilon)^{\gamma}$$

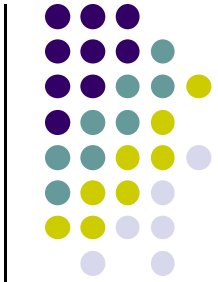
- I=intensity on screen
- V=input voltage (electron gun)
- $a, \varepsilon,$ *and* γ *are* constants for each system
- Common gamma values: 2.3-2.6

Why is it important to care about gamma correction?



- Portability/image quality across platforms
  - Texturing
  - Interpolation
- One solution is to put gamma correction in hardware…

Transparency, alpha and Compositing

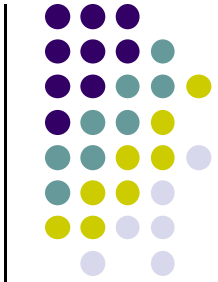


- Transparency
  - Very simple in real-time contexts
- **Alpha blending:** mix two colors
- Alpha (α) is another component in the frame buffer, or on triangle
  - Represents the opacity
  - 1.0 is totally opaque
  - 0.0 is totally transparent
- The over operator:

$$\mathbf{c}_o = \alpha \mathbf{c}_s + (1-\alpha)\mathbf{c}_d$$

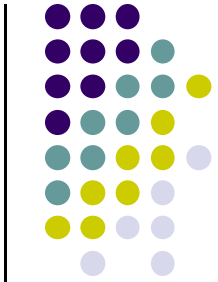
Rendered object

Transparency



- Need to sort the transparent objects
  - Render back to front
  - Why?
- Lots of different other blending modes

- Can store RGBα in textures as well



References

- UIUC CS 319, Advanced Computer Graphics Course
- David Luebke, CS 446, U. of Virginia, slides
- Chapter 1-6 of RT Rendering
- CS 543/4731 course slides
- Hanspeter Pfister, CS 175 Introduction to Computer Graphics, Harvard Extension School, Fall 2010 slides
- Christian Miller, CS 354, Computer Graphics, U. of Texas, Austin slides, Fall 2011
- Ulf Assarsson, TDA361/DIT220 - Computer graphics 2011, Chalmers Instititute of Tech, Sweden