Reflection Modeling

- When a beam of light $i(\lambda)$ from an illumination source hits a surface, some of that light is absorbed and some reflected.
- specify how much of each wavelength is reflected using a reflectance function $r(\lambda)$.
- model reflection as per wavelength multiply

$$l(\lambda) = i(\lambda)r(\lambda)$$

- metameric beams under one illuminant, but may produce distinguishable beams under a second illuminant:

$$\begin{aligned} & \vec{c}(i_1(\lambda)r_a(\lambda)) = \vec{c}(i_1(\lambda)r_b(\lambda)) \\ ! \Leftrightarrow \quad & \vec{c}(i_2(\lambda)r_a(\lambda)) = \vec{c}(i_2(\lambda)r_b(\lambda)) \end{aligned}$$

- see slide
- cannot be correctly simulated with only 3 color numbers
  - should be modeled in graphics with spectra,
- but in CG, we typically hack this as rgb anyway

White Balance

- altered illuminants will alter the colors in a camera
- see web images
- may want to adjust image to appear as it would under daylight.
- simplest transform is to apply RGB scales.
- cannot hope to do full correction from just captured color.
- see slide for success and failure

Adaptation

- altered illuminants will alter the retinal colors
- human brain also tries to do adjustment
- this results in *color constancy*
- it allows us to think of colors as belonging to an object.
- we can actually do this locally in the field of view
  - like with the checkershadow example
- we still need white balancing when looking at an image in a canonical surround environment.

Perceptual Distance

- Euclidean distance between two colors in any linear color space is not a good predictor as to how "different" they will appear to a human observer.
- For example, humans are much more sensitive to changes in dark colors than they are to bright ones.
  - see bw ramp
- there are a bunch of non-linear color spaces designed for this purpose
- this would be useful, for example, when we are quantizing colors
- one such space is called L^*ab

- The L^* coordinate is called "lightness" and is computed as

$$L^* = 116 \left(\frac{Y}{Y_n} \right)^{\frac{1}{3}} - 16 \tag{1}$$

gamma corrected

- a simpler way to get this effect is simply use a power on RGB

$$R' = R^{.45} \tag{2}$$
$$G' = G^{.45} \tag{3}$$
$$B' = B^{.45} \tag{4}$$

- see ramp figure
- A related but slightly more involved non-linear transform can be applied to $[R, G, B]^t$, instead of Equation (2), to obtain sRGB coordinates, called $[R'_{\text{srgb}}, G'_{\text{srgb}}, B'_{\text{srgb}}]^t$.
- Modern LCD displays are programmed to assume input in these coordinates.

Gamma and Graphics

- images storage reps, and the monitor, expect gamma corrected.
- CG computation are linearly related to light beams
  - reflection, blending
- OpenGL we can request an sRGB frame buffer using the call `glEnable(GL_FRAMEBUFFER_SRGB)`.
- Then we can pass linear $[R, G, B]^t$ values out from the fragment shader, and they will be gamma corrected into the sRGB format before begin sent to the screen.

srgb textures

- Additionally, for texture mapping, we can specify that the image being input to a texture is in sRGB format.
- This is done using the call
  `glTexImage2D(GL_TEXTURE_2D, 0, GL_SRGB, twidth, weight, 0, GL_RGB, GL_UNSIGNED_BYTE, pixdata)`
- Whenever this texture is accessed in a fragment shader, the data is first converted to linear $[R, G, B]^t$ coordinates, before given to the shader.