

Computer Graphics

CSE 167 [Win 19], Lecture 13: Raster Graphics, Color

Ravi Ramamoorthi

<http://viscomp.ucsd.edu/classes/cse167/wi19>

Lecture Overview

- Many basic things tying together course
 - Is part of the material, will be covered (briefly) on midterm
- Raster graphics*
- Gamma Correction
- Color
- Hardware pipeline and rasterization
- Displaying Images: Ray Tracing and Rasterization
 - Essentially what this course is about (HW 2 and HW 4)
- Introduced now so could cover basics for HW 1,2,3

Some images from wikipedia

Images and Raster Graphics

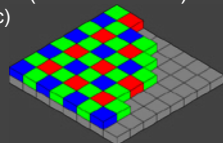
- Real world is continuous (almost)
- How to represent images on a display?
- Raster graphics: use a bitmap with discrete *pixels*
- Raster scan CRT (paints image line by line)
- Cannot be resized without loss
- Compare to vector graphics
 - Resized arbitrarily. For drawings
 - But how to represent photos, CG?

Displays and Raster Devices

- CRT, flat panel, television (rect array of pixels)
- Printers (scanning: no physical grid but print ink)
- Digital cameras (grid light-sensitive pixels)
- Scanner (linear array of pixels swept across)
- Store image as 2D array (of RGB [sub-pixel] values)
 - In practice, there may be resolution mismatch, resize
 - Resize across platforms (phone, screen, large TV)
- Vector image: description of shapes (line, circle, ...)
 - E.g., line art such as in Adobe Illustrator
 - Resolution-Independent but must *rasterize* to display
 - Doesn't work well for photographs, complex images

Resolutions

- Size of grid ($1920 \times 1200 = 2,304,000$ pixels)
 - 32 bit of memory for RGBA framebuffer 8+ MB
- For printers, pixel density (300 dpi or ppi)
 - Printers often binary or CMYK, require finer grid
 - iPhone "retina display" > 300 dpi. At 12 inches, pixels closer than retina's ability to distinguish angles
- Digital cameras in Mega-Pixels (often > 10 MP)
 - Color filter array (Bayer Mosaic)
 - Pixels really small (micron)



Monitor Intensities

- Intensity usually stored with 8 bits [0...255]
- HDR can be 16 bits or more [0...65535]
- Resolution-independent use [0...1] intermediate
- Monitor takes input value [0...1] outputs intensity
 - Non-zero intensity for 0, black level even when off
 - 1.0 is maximum intensity (output 1.0/0.0 is contrast)
 - Non-linear response (as is human perception)
 - 0.5 may map to 0.25 times the response of 1.0
 - Gamma characterization and gamma correction
 - Some history from CRT physics and exponential forms

Lecture Overview

- Many basic things tying together course
- Raster graphics
- *Gamma Correction*
- Color
- Hardware pipeline and rasterization
- Displaying Images: Ray Tracing and Rasterization
 - Essentially what this course is about (HW 2 and HW 5)

Some images from wikipedia

Nonlinearity and Gamma

- Exponential function $I = a^\gamma$
- I is displayed intensity, a is pixel value
- For many monitors γ is between 1.8 and 2.2
- In computer graphics, most images are linear
 - Lighting and material interact linearly
- Gamma correction $a' = a^{\frac{1}{\gamma}}$
- Examples with $\gamma = 2$
 - Input $a = 0$ leads to final intensity $I = 0$, no correction
 - Input $a = 1$ leads to final intensity $I = 1$, no correction
 - Input $a = 0.5$ final intensity 0.25. Correct to 0.707107
 - Makes image "brighter" [brightens mid-tones]

Gamma Correction

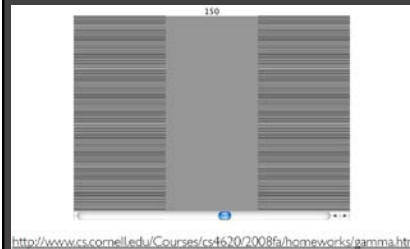
- Can be messy for images. Usually gamma on one monitor, but viewed on others...
- For television, encode with gamma (often 0.45, decode with gamma 2.2)
- CG, encode gamma is usually 1, correct



www.dfstudios.co.uk/wp-content/uploads/2010/12/graph_gammacor.png

Finding Monitor Gamma

- Adjust grey until match 0-1 checkerboard to find mid-point a value i.e., a for $I = 0.5$



$$\gamma = \frac{\log 0.5}{\log a}$$

Human Perception

- Why not just make everything linear, avoid gamma
- Ideally, 256 intensity values look linear
- But human perception itself non-linear
 - Gamma between 1.5 and 3 depending on conditions
 - Gamma is (sometimes) a feature
 - Equally spaced input values are perceived roughly equal

Lecture Overview

- Many basic things tying together course
- Raster graphics
- Gamma Correction
- *Color*
- Hardware pipeline and rasterization
- Displaying Images: Ray Tracing and Rasterization
 - Essentially what this course is about (HW 2 and HW 5)

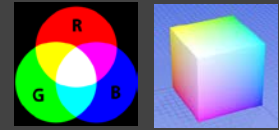
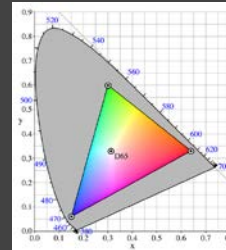
Some images from wikipedia

Color

- Huge topic (can read textbooks)
 - Schrodinger much more work on this than quantum
- For this course, RGB (red green blue), 3 primaries
- Additive (not subtractive) mixing for arbitrary colors
- Grayscale: $0.3 R + 0.6 G + 0.1 B$
- Secondary Colors (additive, not paints etc.)
 - Red + Green = Yellow, Red + Blue = Magenta, Blue + Green = Cyan, R+G+B = White
- Many other color spaces
 - HSV, CIE etc.

RGB Color

- Venn, color cube
- Not all colors possible



Images from wikipedia

Eyes as Sensors

The human eye contains cells that sense light

- Rods
 - No color (sort of)
 - Spread over the retina
 - More sensitive
- Cones
 - Three types of cones
 - Each sensitive to different frequency distribution
 - Concentrated in fovea (center of the retina)
 - Less sensitive

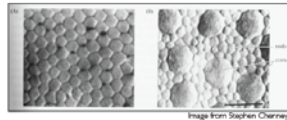
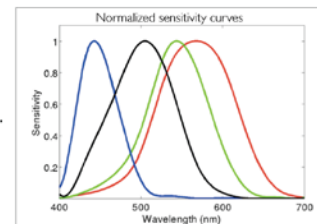


Image from Stephen Chalmers

Slides courtesy Prof. O'Brien

Cones (Trichromatic)

- Each type of cone responds to different range of frequencies/wavelengths
 - Long, medium, short
- Also called by color
 - Red, green, blue
 - Misleading: "Red" does not mean your red cones are firing...



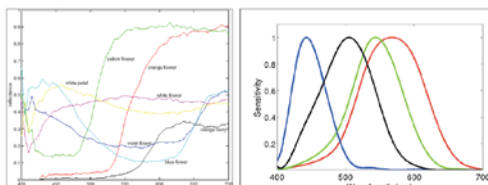
Cone Response

Response of a cone is given by a convolution integral :

$$L = \int \Phi(\lambda) L(\lambda) d\lambda \quad \text{continuous version of a dot product}$$

$$M = \int \Phi(\lambda) M(\lambda) d\lambda$$

$$S = \int \Phi(\lambda) S(\lambda) d\lambda$$



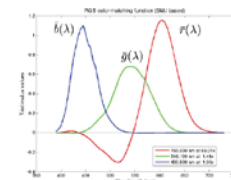
Color Matching Functions

Using Color Matching Functions

Given color matching functions in matrix form and new light

$$C = \begin{pmatrix} \bar{r}(\lambda_1) & \dots & \bar{r}(\lambda_N) \\ \bar{g}(\lambda_1) & \dots & \bar{g}(\lambda_N) \\ \bar{b}(\lambda_1) & \dots & \bar{b}(\lambda_N) \end{pmatrix}$$

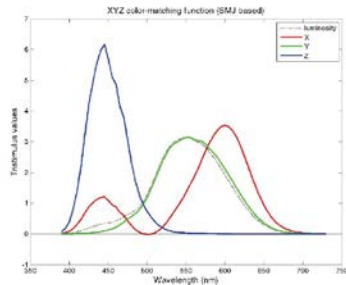
$$\Phi = \begin{pmatrix} \phi(\lambda_1) \\ \vdots \\ \phi(\lambda_N) \end{pmatrix}$$



amount of each primary necessary to match is given by $C\Phi$

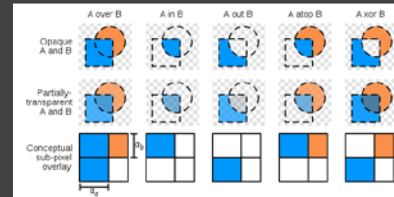
CIE XYZ

Imaginary set of color primaries with positive values, X, Y, Z



Alpha Compositing

- RGBA (32 bits including alpha transparency)
 - You mostly use 1 (opaque)
 - Can simulate sub-pixel coverage and effects
- Compositing algebra



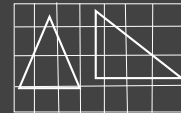
Lecture Overview

- Many basic things tying together course
- Raster graphics
- Gamma Correction
- Color
- *Hardware pipeline and rasterization*
- Displaying Images: Ray Tracing and Rasterization
 - Essentially what this course is about (HW 2 and HW 5)

Read chapter 8 more details

Hardware Pipeline

- **Application** generates stream of vertices
- **Vertex shader** called for each vertex
 - Output is transformed geometry
- **OpenGL** rasterizes transformed vertices
 - Output are fragments
- **Fragment shader** for each fragment
 - Output is Framebuffer image

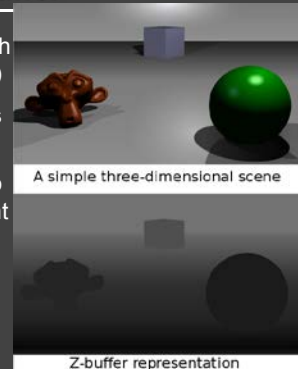


Rasterization

- In modern OpenGL, really only OpenGL function
 - Almost everything is user-specified, programmable
 - Basically, how to draw (2D) primitive on screen
- Long history
 - Bresenham line drawing
 - Polygon clipping
 - Antialiasing
- What we care about
 - OpenGL generates a fragment for each pixel in triangle
 - Colors, values interpolated from vertices (Gouraud)

Z-Buffer

- Sort fragments by depth (only draw closest one)
- New fragment replaces old if depth test works
- OpenGL does this auto can override if you want
- Must store z memory
- Simple, easy to use



Lecture Overview

- Many basic things tying together course
- Raster graphics
- Gamma Correction
- Color
- Hardware pipeline and rasterization
- *Displaying Images: Ray Tracing and Rasterization*
 - Essentially what this course is about (HW 2 and HW 4)

What is the core of 3D pipeline?

- For each object (triangle), for each pixel, compute shading (do fragment program)
- Rasterization (OpenGL) in HW 2
 - For each object (triangle)
 - For each pixel spanned by that triangle
 - Call fragment program
- Ray Tracing in HW 4: flip loops
 - For each pixel
 - For each triangle
 - Compute shading (rough equivalent of fragment program)
- HW 2, 4 take almost same input. Core of class

Ray Tracing vs Rasterization

- Rasterization complexity is $N * d$
 - (N = objs, p = pix, d = pix/object)
 - Must touch each object (but culling possible)
- Ray tracing naïve complexity is $p * N$
 - Much higher since $p \gg d$
 - But acceleration structures allow $p * \log(N)$
 - Must touch each pixel
 - Ray tracing can win if geometry very complex
- Historically, OpenGL real-time, ray tracing slow
 - Now, real-time ray tracers, OpenRT, NVIDIA Optix
 - NVIDIA Turing RT (2018) claims 10 billion rays/second!!
 - Ray tracing has advantage for shadows, interreflections
 - Hybrid solutions now common

Course Goals and Overview

- Generate images from 3D graphics
- Using both rasterization (OpenGL) and Raytracing
 - HW 2 (OpenGL), HW 4 (Ray Tracing)
- Both require knowledge of transforms, viewing
 - HW 1
- Need geometric model for rendering
 - Splines for modeling (HW 3)