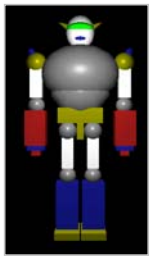
Color and Texture

CS148: Intro to CG
Instructor: Dan Morris
TA: Sean Walker
July 19, 2005

Pre-Lecture Business

- Get going on pp3!
- TOBOR's greatest hits
- Exam question review

TOBOR's Greatest Hits

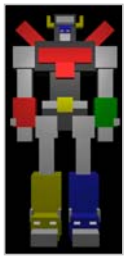


Tough-guy TOBOR

Ms. TOBOR

TOBOR's Greatest Hits

Ripped TOBOR



TOBOR 2050

Transformation Question (3b)

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

glRotatef( -90.0 , 0.0 , 1.0 , 0.0 );   (1,0,0)
glScalef( 1.0 , 1.0 , 0.5 );            (0,0,-1)
glTranslatef ( 0.0 , 2.0 , 0.0 );       (0,0,-2)
glScalef( 0.0 , 2.0 , 2.0 );            (0,-2,-2)
glTranslatef( 0.0 , -2.0 , 0.0 );       (-1,-1,-1)
glRotatef( 180.0 , 0.0 , 1.0 , 0.0 );   (-1,1,-1)
glVertex3f( 1.0 , 1.0 , 1.0 );          (1,1,1)
```

Transformation Question (3a)

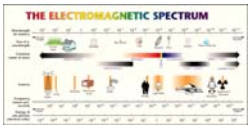
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef( 0.0 , 0.0 , 1.0 );        (0,0,0)

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glRotatef( 90.0 , 0.0 , 0.0 , 1.0 );    (0,-1,0)
glTranslatef(-1.0 , 0.0 , 0.0 );        (-1,0,0)

glMatrixMode(GL_MODELVIEW);
glRotatef( 90.0 , 0.0 , 1.0 , 0.0 );    (0,0,-1)
glTranslatef( -1.0 , 0.0 , 0.0 );       (1,0,0)

glVertex3f( 2.0 , 0.0 , 0.0 );          (2,0,0)
```

Outline for today

- Color and Color Spaces
- Texture Mapping

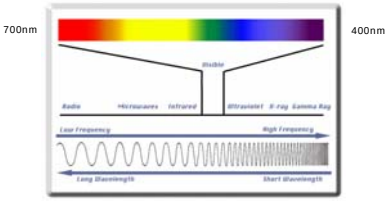
What is light?

- Photons: “particles of energy”
- The *intensity* of a radiation source is defined by its rate of photon emission
- The “character” of a photon is defined by its wavelength



What is a color?

- The wavelength of a single photon defines a color
- The wavelength of a beam of “pure” light (one wavelength) defines a color

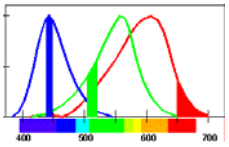


What’s wrong with this definition?

- Intuitively, white is a color, but white doesn’t have a wavelength.
- Most “colors” are really combinations of wavelengths
- How many colors are there in this rectangle?

Human Color Perception in 1 Slide

- Photons hit our eyes and activate cone photoreceptors
- A signal is sent up the optic nerve to our brain, and we perceive color
- There are only three kinds of cones in our eye, each sensitive to a range of wavelengths



Why do cone responses overlap?

A Better Definition of Color

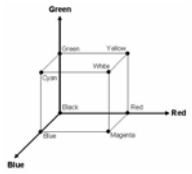
- A color is a particular pattern of cone activation: a *perceived* color
- There are many combinations of wavelengths that can produce the same *color*

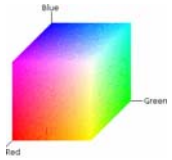
Representing Color

- We need a way to represent this concept of color for applications in CG and print
- An intuitive way based on what we just learned:
  - A color is three numbers, each of which *roughly* represents the amount of activation of one cone type
- This is the theoretical basis for the RGB color model
- In practice, R, G, and B are wavelengths that are intended to optimally stimulate the RGB cones

The RGB Color Space

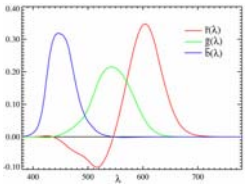
- Colors are combinations of the primaries R, G, and B:
  C = rR + gG + bB
- This is called an "additive color space"
- Can represent colors as points in the RGB coordinate space:





Limitations of RGB

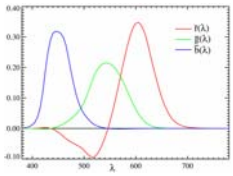
- Not all colors can be expressed in a purely additive space
- There are visible colors you *can't describe* with RGB!



What's a negative color?

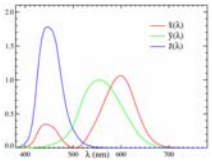
CIE color experiments

- 1931: folks at CIE gave people dials to control red green, and blue guns
- Gave them a "test color" to match with the dials
- If they couldn't match a color, they could *add* some RGB to the *test* color (this counted as "negative color")
- Generated standard "color matching" curves



CIE Color

- Defined a standard set of "primary colors" (X, Y, and Z) that could be mixed together to form the visible spectrum
- These "colors" were actually mixtures of many wavelengths

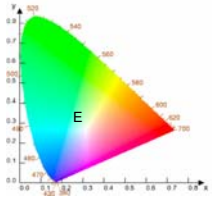


CIE XYZ Color Space

- X, Y, and Z can be treated as a color space just like R, G, and B
- C = X + Y + Z
- If we normalize for total brightness, we can get values that truly describe *color* (tone)
- x = X / (X + Y + Z)
  y = Y / (X + Y + Z)
  z = Z / (X + Y + Z)
- If we pick a constant brightness and call it 1.0, we can plot all possible colors on the x,y space…

CIE XYZ Chromaticity Diagram

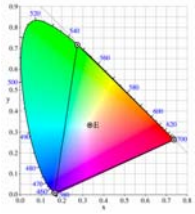
- Represents all the visible colors in terms of (x,y)
- Edges represent pure colors
- E is the “white point”



Color gamuts

- Gamut: the set of colors a particular device or color space can represent
- The triangle shows the standard RGB gamut… pretty disappointing…
- Most display devices are built to roughly match this

Can any three visible colors have a gamut that fills the visible space?



Why not *use* CIE XYZ?

- CIE XYZ is a great theoretical space with an ideal gamut, but it doesn’t represent:
  - How we see (cone activation)
  - How monitors work
  - How printers work
  - Anything intuitive to an artist or programmer
- Used to define every other color space

RGB revisited

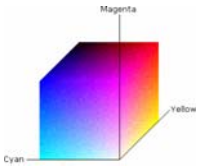
- What an RGB space *really* is: a set of primaries R, G, and B defined in terms of X and Y
- There are many RGB spaces (sRGB, Adobe RGB, NTSC (TV) RGB)
- People keep making up new ones because they better represent RGB hardware or because they have bigger gamuts

What other spaces do we use?

- RGB is fairly intuitive and represents monitor activation well
- Doesn’t map well to what printers can produce
- Most printers print on white paper and the ink *removes* reflected color
- So we define a *subtractive* color space for printers…

CMY: A Subtractive Color Space

- color = C + M + Y
  - C → cyan, M → magenta, Y → yellow
- (0,0,0) is white, (1,1,1) is black
- C = 1 – R, M = 1 – G, Y = 1 – B



CMYK: A hack to fix CMY

- Printers are built with cyan, yellow, and magenta ink
- (1,1,1) *should* be black, but in practice it's not
- So printers add black ink to make true black

Why else use black ink?

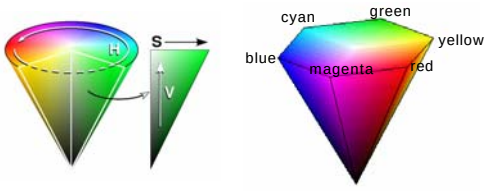
Limitations of RGB and CMY

- Some operations are hard to express in RGB/CMY: make this color "more pale", make this color "more vivid"
- Image processing applications and artists often need access to these operations
- Presenting a 2D color chooser in RGB is tough, since a "slice" from the RGB cube contains very similar colors

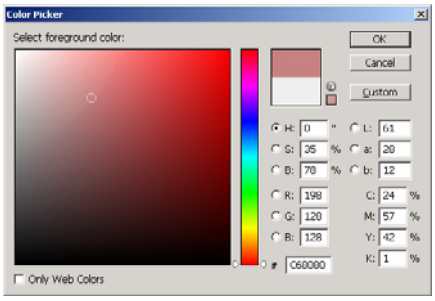
The HSV color space

- Hue: what *tone* is this color (red, blue, teal, etc.)?
  - Red is 0˚ or 360˚
- Saturation: how *colorful* is this color?
  - 0 is grayscale, no color
- Value: how bright is this color?
  - 0 is black

Visualizing the HSV color space



What color space does Photoshop's color picker display?



OpenGL Trivia

- There is another (very rarely used) matrix mode in OpenGL:

  glMatrixMode(GL_COLOR);

- The color matrix is applied to all (RGBA) colors before they're displayed

What might I use the color matrix for?

Outline for today

- Color and Color Spaces
- Texture Mapping

What have we done so far?

- We can render 3D objects to the screen in glorious color
- We can apply realistic lighting to objects using geometry and surface normals
- But rendering something like this would require hella vertices:

Texture Mapping

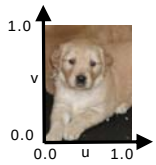
- Pasting a 2D image onto the surface of a 3D object
- Extremely important feature of OpenGL

Texture Coordinates

- I have some .jpg file that I want on my object…
- Define a 2D "texture coordinate space" for the pixels in my image
  - Usually called (u,v), usually uses the interval (0,1)

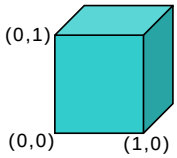
How can I access the *texel* (texture pixel) at coordinate (u,v)?

```
color3 getTextureValue(float u, float v) {
   return myImage[(int)u*width,(int)v*height];
}
```



Texture Coordinates

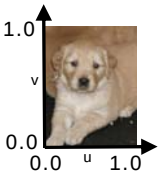
- Assign a texture coordinate to each vertex in your object
- Done manually for simple objects, stored in mesh files with each vertex for more complex objects

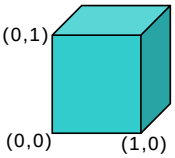


Texture Mapping Overview

- OpenGL *maps* from texture coordinates to pixels

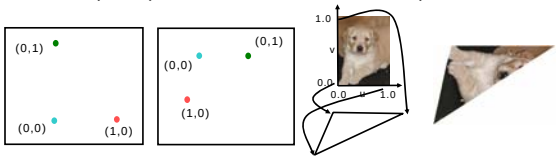
How many vertices were used to define the "puppy cube"?





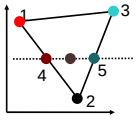
Texture Mapping Overview

- User supplies texture coordinates with each vertex
- OpenGL stores the texture coordinate with each vertex as it moves through the pipeline
- When it comes time to rasterize a polygon, use the texture coordinates at each vertex to find a texture coordinate for each *pixel*
- Look up the corresponding *texels* (texture pixels) and use them to color each pixel



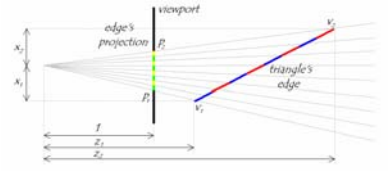
Interpolating Texcoords: Take One

- When we're rasterizing a polygon, we only have texture coordinates at the corners
- We need to interpolate to find tex coords for all the pixels in between
- Let's do that just like we did with smooth shading:
  - Use linear interpolation to find the the tex coords where the scanline enters the polygon
  - Use linear interpolation as we move across that scanline



What's wrong with this approach?

- Interpolating in 2D means that if two vertices are 10 pixels apart, I take 10 equal steps as I move across my texture
- Because of perspective distortion, equal 2D steps don't necessarily represent equal 3D steps!



The solution (the short version)

- What OpenGL really does is interpolate in 3D, by mapping vertices *backward* through the projection matrix and interpolating there… slower but necessary.

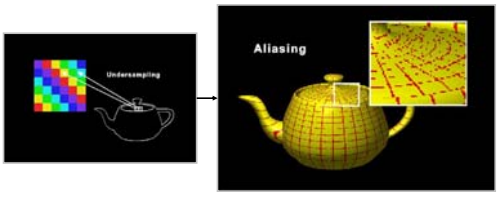
Why didn't we have to interpolate in 3D for Gouraud shading?

Texture Sampling: Take One

- Now we now how OpenGL finds the texture coordinates for each pixel in our polygon… how do we assign a color to that pixel?
- The easiest approach:
  - I know the (u,v) for a pixel
  - Use the getTextureValue function we wrote earlier to get the *nearest* pixel to our (u,v)
  - Use that to color our pixel

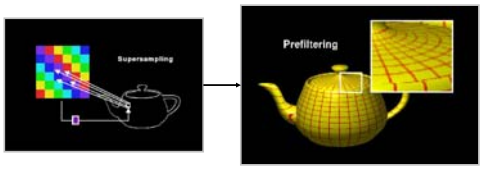
What's wrong with this approach?

- If we have more texture pixels than screen pixels (i.e. if our texture is being "squished"), we can *miss* pixels in the texture and get nasty artifacts and flickering



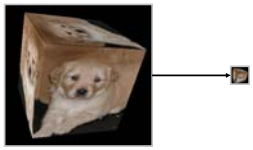
Texture Filtering

- What did we do the last time we had to deal with aliasing?
- We used *averaging* to deal with aliasing in lines
- So we'll use averaging to deal with aliasing in textures… each pixel should use an average of nearby texels



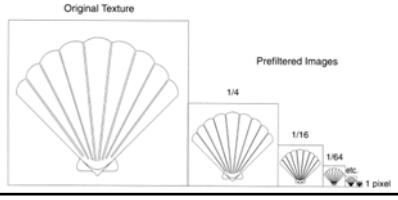
What's still wrong with this approach?

- Doing all that averaging is a *lot* of floating point math
- If a textured object is really far away, it seems wasteful to access lots of extra texels when I don't *need* that much accuracy



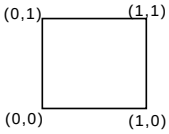
A Solution: Mipmapping

- *Precompute* averages for blocks of pixels to generate smaller, filtered versions of my texture
- Each copy of the texture is called a "mipmap"
- When it's time to access texels, access the ones that are closest to the resolution I need… already filtered!
- Can also interpolate *between* mipmaps to avoid "jumping" from one mipmap to the next



Texture mapping in OpenGL (1)

- Choose or load texture coordinates for each of your vertices… we'll do a simple quad as today's example
- Load your image into memory or make it yourself… OpenGL has no standard file format, so it's your job to load your images
  - We provide you with tga.cpp to load TGA files



Texture Mapping in OpenGL (2)

- Ask OpenGL for a "name" for a new texture (works just like display lists):

```
int myTextureID;
glGenTextures(1, & myTextureID);
```

- Tell OpenGL that *this* is the texture we're going to be working with for now

```
glBindTexture (GL_TEXTURE_2D,
  myTextureID);
```

Texture Mapping in OpenGL (3)

- Send your image to OpenGL… i.e. make a *texture* from your image

```
glTexImage2D (GL_TEXTURE_2D, 0,
  format, width, height, 0, format,
  GL_UNSIGNED_BYTE, data);
```

- This only supports power-of-two textures and doesn't make mipmaps, so we often use the helpful glu version:

```
gluBuild2DMipmaps(GL_TEXTURE_2D,
  components, width, height, format,
  GL_UNSIGNED_BYTE, data);
```

Texture Mapping in OpenGL (4)

```
// What should OpenGL do with texcoords > 1.0?
glTexParameteri (GL_TEXTURE_2D,
    GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri (GL_TEXTURE_2D,
    GL_TEXTURE_WRAP_S, GL_CLAMP);

// How should OpenGL filter my textures?
glTexParameteri (GL_TEXTURE_2D,
    GL_TEXTURE_MAG_FILTER,
    GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri (GL_TEXTURE_2D,
    GL_TEXTURE_MIN_FILTER,
    GL_NEAREST);

// What should OpenGL do with my texture maps?
glTexEnvf (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
    GL_MODULATE);
glTexEnvf (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
    GL_DECAL);
```

Texture Mapping in OpenGL (5)

- Time to render… enable texture-mapping:

```
glEnable(GL_TEXTURE_2D);
```

- Make sure the current texture is the one you want:

```
glBindTexture (GL_TEXTURE_2D, myTextureID);
```

- Render and provide texture coordinates with each vertex:

```
glBegin( GL_QUADS );
    glTexCoord2d(0.0,0.0); glVertex2d(0.0,0.0);
    glTexCoord2d(1.0,0.0); glVertex2d(1.0,0.0);
    glTexCoord2d(1.0,1.0); glVertex2d(1.0,1.0);
    glTexCoord2d(0.0,1.0); glVertex2d(0.0,1.0);
glEnd();
```

Texture Mapping in OpenGL (6)

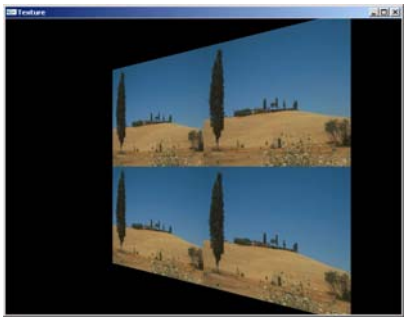
- Usually clean up after yourself…
- After you're done drawing textured objects in each frame:

```
glDisable(GL_TEXTURE_2D);
```

- After you're done with your textures:

```
glDeleteTextures(1, & myTextureID);
```

OpenGL example: loadtexture.cpp



Next Time

- Advanced Texture Mapping
- Curves and Curved Surfaces