Computer Science 112
Art & Science of Computer Graphics
The College of Saint Rose
Fall 2015

Topic Notes: Colors and Materials

Colors

We've used Ambrosia's predefined colors and materials, but now it's time to define our own, as we study colors and materials more closely next.

Colors in Ambrosia (and in much of computer graphics) use the *RGB* (red-green-blue) color system.

Red, green, and blue are the *primary colors of light*. Computer monitors (and televisions, etc.) are typically made of lots of red, green, and blue light sources.

In Ambrosia, the values for red, green, and blue are specified in the range 0-1. 0 means do not turn that color on, 1 means turn that color on as bright as possible.

The predefined colors are simple assignments made by the Ambrosia system, so Python knows what they represent:

```
black = (0,0,0)
dkGray = (0.25,0.25,0.25)
gray = (0.5,0.5,0.5)
ltGray = (0.75,0.75,0.75)
white = (1,1,1)
red = (1,0,0)
green = (0,1,0)
blue = (0,0,1)
cyan = (0,1,1)
magenta = (1,0,1)
yellow = (1,1,0)
purple = (89/255,17/255,142/255)
```

We see the primary colors of light here, some shades of gray, and the *secondary colors of light*, which are mixtures of the primaries: cyan is geen plus blue, magenta is red plus blue, and yellow is red plus green (what?!).

Other colors we might think about:

```
ltYellow = (1, 1, .5)
dkBlue = (0, 0, .5)
```

There are many programs that can be used to visualize the colors specified by different RGB values. I like this one:

`http://www.cs.rit.edu/~ncs/color/a_spaces.html`

Any color we can display has a unique RGB representation.

It's not always obvious how to create the color we are looking for in the RGB scheme. An alternate specification is Hue-Saturation-Value (HSV).

Again, we specify a color by three values:

1. *Hue*: a value in the range 0-360 that specifies the angle on a color wheel
   - red is at 0 degrees
   - we pass through orange, yellow, then on to...
   - green at 120 degrees
   - we pass through cyan
   - blue is at 240 degrees
   - we pass through indigo, violet/magenta...
   - and back to red at 360
2. *Saturation*: a value from 0-1 that specifies the amount of the hue color to mix with white
   - when sat=1, color is the hue
   - when sat=0, color is white

   This allows us to make lighter versions of colors by specifying smaller values of the saturation.
3. *Value*: a value in the range 0-1 that specified the amount of this color that you "add" to black
   - when val=1, color is hue+sat
   - when val=0, color is black

What does this all mean? Let's consider a few ranges of values:

- HSV=`(0, *, *)` gives us a shade of red (ranging from white through the reds, to black)
- HSV=`(*, 0, 0)` is black (hue is irrelevant)
- HSV=`(*, 0, 1)` is white (again, hue is irrelevant)

(here, $\star$ can be replaced by any value in the range 0-1)

And some examples of HSV colors (note that these are not defined by Ambrosia):

```
HSVRed = (0, 1, 1)
HSVGreen = (120, 1, 1)
HSVWhite = (0, 0, 1)
HSVWhite = (90, 0, 0)  # either one!
HSVBlack = (0, 0, 0)
HSVGray = (0, 0, .5)
HSVOrange = (30, 1, 1)  # 30 degrees around
HSVBrown = (30, 1, 0.5) # 30 degrees around, darker
HSVPink = (0, .5, 1) # "pastels" have lower saturation
```

A neat feature of the HSV representation is that averaging two HSV values will produce an HSV representation of the "average" color.

Ambrosia, however, does not support HSV colors directly. All colors must be in RGB format.

Fortunately, there is an Ambrosia function that will convert an HSV value to its RGB equivalent: `hsv2rgb`

For example, `hsv2rgb(HSVWhite)` returns `(0,0,0)`, the RGB representation of white.

There is also an `rgb2hsv` but we should note that there are many representations of white, for example. The function cannot be defined uniquely for all inputs.

Materials

We have seen the predefined "plaster" and "plastic" materials provided by Mead:

On the Wiki: Plasterandplastic

Next, we'll see how to create our own materials.

To do this, we create a new object of type `Material` and specify its attributes. This `Material` can then be used for objects we add to our scene.

The attributes we can set for our materials:

- `color`: the material's color in RGB. Remember that each component ranges 0-1. Ambrosia will allow values greater than 1 for the components, but this will result in excessively "bright" objects that are probably not what you are going for. If you want a bright material, instead adjust the parameters below.
- `ambient`: a value in the range 0-1 that specifies how much ambient light illuminates the material. Note that this value is usually very small, since larger values will make the object appear to be unnaturally illuminated without regard to light from actual sources. A non-zero value allows objects that are not illuminated by a direct light source or reflection of light sources off of other objects, to take on their color.

  The `ambient` value is 0.1 for plastic materials, 0.2 for plaster materials.

Note: unlike in real life, where the ambient light which is reflected by an object can illuminate other objects, POVray (and hence, Ambrosia) does not model ambient light in this way.

- `diffuse`: a value in the range 0-1 that specifies how much of the incident light is scattered using the native color of the object. A diffuse index value of 0 means that an object's color comes only from ambient light, specularity, transparency, and reflection. A diffuse index value of 1 (for an opaque object) means that all incident light of colors reflected by the surface is reflected only in the object's color.

  The `diffuse` index value is 0.5 for plastic materials, 0.9 for plaster.

- `roughness`: a value in the range 0 and up, 0 is perfectly smooth, 1 is rough (like a piece of chalk), higher numbers indicate even more rough surfaces. On smooth surfaces, the angle of incidence is the same as the angle of reflection. On rough surfaces, the angle of reflection varies to a degree based on imperfections in the surface. Rough surfaces don't reflect perfect images, smooth surfaces do. Realistically, values below 0.05 are not useful.

  The `roughness` is 0.05 for plastic materials, 1 for plaster materials.

- `specularity`: a value in the range 0-1 that specifies how much incident light from a source is reflected (as if the surface were reflective), leading to "specular highlights". Specular reflection results in, for example, the ability of a glossy paint to reflect some of its incident light. Specular highlights are spots that appear where light reflects directly back toward the viewer from the source, especially on metallic or "shiny" surfaces.

  The `specularity` is 0 for plaster materials, 0.7 for plastic.

- `reflection`: a value in the range 0-1 that specified how much reflection of adjacent objects we see in the surface – 0 is no reflection at all, 1 is a perfect mirror. In most circumstances, values up to about 0.1 are sufficient to add realistic reflectivity to non-mirror objects (such as glass).

  The default `reflection` is 0.

- `transparency`: a value in the range 0-1 that specifies how much light passes through the object – 0 is completely opaque, 1 is a completely transparent object (we won't see it at all).

  The default `transparency` is 0.

- `refraction`: When light travels at an angle across an interface between two media of different densities, it bends. The amount of this bend is a ratio called the *index of refraction*. For transparent materials, it is important to set the index of refraction to a value greater than 1, otherwise the object will not be visible. The index of refraction is set with the `refraction` message, which takes a value, typically 1 or more. (Water is 1.33, glass is typically 1.5, and diamond is 2.4. Notice the increase in the refractive index with increased density.)

  The default `refraction` is 1, meaning the light does not bend.

In addition to the above parameters, Ambrosia provides four material "types" that can be used to specify a group of appropriate parameters. We have seen `plaster` and `plastic` materials, but there are also `mirror` and `glass` types.

These are specified in a `Material` definition with the `type`. So for example, to define plaster with a color of our own specification:

```
ltBluePlaster = Material()
ltBluePlaster.type("plaster")
ltBluePlaster.color(hsv2rgb((240,.5,1)))
scene.add(sphere, ltBluePlaster)
```

Specifying one of these types sets material attributes as follows:

- `plastic`: `ambient(0.1).diffuse(0.5).roughness(0.05).specularity(0.7)`
- `plaster`: `ambient(0.2).diffuse(0.9).roughness(1).specularity(0)`
- `mirror`: `ambient(0).diffuse(0.1).reflection(0.9)`
- `glass`: `ambient(0).diffuse(0.1).transparency(0.8).refraction(1.5)`

The best way to understand these attributes is to experiment with them.

On the Wiki: Materialsplayroom

See also the Materials page on the wiki:

`http://ascg-wiki.teresco.org/index.php/Materials`

This example introduces a new object type: the `Plane`. Several of you have wanted to use such an object, which is an infinite two-dimensional plane that exists (and can be rendered) in three-dimensional space.

The default position of a `Plane` is along the xz-plane. It can be manipulated with transformations just like our familiar basic object types.

On the Wiki: Diffuseandambient

On the Wiki: Key

This example demonstrates more custom objects and adds some shiny materials.

It also includes a new transformation: a *mirroring*. This does exactly what you'd expect: it mirrors the object to which it is applied across the specified plane (in this case, the xy-plane).

Lenses

We can build a lenses out of a refractive material.

A convex lens is like a maginfying glass, a concave lens is like what you'd find in a peephole on a door.

See the Wikipedia page about Lens Optics:

`http://en.wikipedia.org/wiki/Lens_%28optics%29`

On the Wiki: Lenses

In addition to demonstrating refraction, this example introduces a new message we can send to our `image`:

```
image.dimensions(1024, 768)
```

This changes the size of the generated image. By default, it is 640×480.

Designing a Material

There are many ways to go about creating a new Ambrosia `Material`. Here is one possible approach.

1. Does your material have a noticable color that is independent of light color?

   Yes? This is the diffuse color. Set the RGB color, and diffuse to its intensity.

   No? The value of diffuse is close to zero. Color is unimportant; set it to white.

2. What happens when you shine a light on the object?

   - Does light travel through the object?

     Yes? (*e.g.*, glass)
     - Set transparency to a value greater than zero. For very clear objects, set this close to 1; for objects with obvious diffuse color, set it closer to 0.
     - Set the refractive index. The ratio of the density of the object over the density of its environment. The refractive index is typically greater than 1. If you're in water, and the object is an air bubble, the value is less than 1 (rare).

     No? (*e.g.*, metals)
     - The transparency may be left at its default value (0), and the refractive index will be ignored.

   - Does light bounce off the object?

     Yes? (*e.g.*, shiny things)
     - Specularity and/or reflection are nonzero.
     - Can you see a detailed image in the reflection?

       Yes? (*e.g.*,. mirrors, stainless, mylar)
         Reflection close to 1, and roughness near 0.

No? (*e.g.*, nonsmooth plastics)
Reflection is small, and the roughness is nearer 1.

- Do you see highlights, bright spots the color of the light (typically white)

  Yes? Is it small with obvious boundaries?
  If yes, it's like a metal. Specularity is high (close to 1).
  If no, like plastics, specularity is lower (0.5 or less).

  No? Specularity is very low. Set it to zero.

No? (*e.g.*, chalk) Set reflection and specularity to zero.