Any questions from last time?
Go over sampling image plane?
Or intersection algorithms?
Slide Credits

- Leonard McMillan, Seth Teller, Fredo Durand, Barb Cutler - MIT
- G. Drew Kessler, Larry Hodges - Georgia Institute of Technology
- John Hart - University of Illinois
More Geometry & Intersections
Ray/Plane Intersection

Ray is defined by \( R(t) = R_o + R_d * t \) where \( t \geq 0 \)
- \( R_o \) = Origin of ray at \((x_o, y_o, z_o)\)
- \( R_d \) = Direction of ray \([x_d, y_d, z_d]\) (unit vector)

Plane is defined by \([A, B, C, D]\)
- \(Ax + By + Cz + D = 0\) for a point in the plane
- Normal Vector, \(N = [A, B, C]\) (unit vector)
- \(A^2 + B^2 + C^2 = 1\)
- \(D = -N \cdot P_0\) \((P_0\) - point in plane)
Ray/Plane (cont.)

Substitute the ray equation into the plane equation:

\[ A(x_o + x_d t) + B(y_o + y_d t) + C(z_o + z_d t) + D = 0 \]

Solve for \( t \):

\[ t = -\frac{(Ax_o + By_o + Cz_o + D)}{(Ax_d + By_d + Cz_d)} \]
\[ t = -\frac{(N \cdot R_o - N \cdot P_0)}{(N \cdot R_d)} \]
What Can Happen?

\[
N \cdot R_d = 0
\]

\[
N \cdot R_d > 0
\]

\( t < 0 \)

\( t > 0 \)
Ray/Plane Summary

Intersection point:

\[(x_i, y_i, z_i) = (x_o + x_d t_i, y_o + y_d t_i, z_o + z_d t_i)\]

1. Calculate \(N \cdot R_d\) and compare it to zero.
2. Calculate \(t_i\) and compare it to zero.
3. Compute intersection point.
4. Flip normal if \(N \cdot R_d\) is positive
Ray-Parallelepiped Intersection

- Axis-aligned
- From \((X1, Y1, Z1)\) to \((X2, Y2, Z2)\)
- Ray \(P(t) = R_o + R_d t\)
Naïve ray-box Intersection

- Use 6 plane equations
- Compute all 6 intersection
- Check that points are inside box
  \[ Ax + By + Cz + D \leq 0 \]
Factoring out computation

- Pairs of planes have the same normal
- Normals have only one non-0 component
- Do computations one dimension at a time
- Maintain $t_{\text{near}}$ and $t_{\text{far}}$ (closest and farthest so far)
Test if parallel

- If $R_d x = 0$, then ray is parallel
  - If $R_o x < X_1$ or $R_o x > x_2$ return false
If not parallel

- Calculate intersection distance $t_1$ and $t_2$
  - $t_1 = (X_1 - R_o x)/R_d x$
  - $t_2 = (X_2 - R_o x)/R_d x$

![Diagram showing the calculation of intersection distances $t_1$ and $t_2$. The diagram includes lines $y=Y_1$, $y=Y_2$, $x=X_1$, and $x=X_2$, with points marked for $t_1$ and $t_2$. The radii $R_o$ and $R_d$ are also indicated.]
Test 1

- Maintain $t_{\text{near}}$ and $t_{\text{far}}$
  - If $t_1 > t_2$, swap
  - If $t_1 > t_{\text{near}}$, $t_{\text{near}} = t_1$
  - If $t_2 < t_{\text{far}}$, $t_{\text{far}} = t_2$

- If $t_{\text{near}} > t_{\text{far}}$, box is missed
If $t_{far} < 0$, box is behind
Algorithm recap

- Do for all 3 axes
  - Calculate intersection distance $t_1$ and $t_2$
  - Maintain $t_{near}$ and $t_{far}$
  - If $t_{near} > t_{far}$, box is missed; Done
  - If $t_{far} < 0$, box is behind; Done
- If box survived tests, return intersection at $t_{near}$
- If $t_{near}$ is negative, return $t_{far}$
Ellipsoid's surface is defined by the set of points \( \{(x_s, y_s, z_s)\} \) satisfying the equation:

\[
\left(\frac{x_s}{a}\right)^2 + \left(\frac{y_s}{b}\right)^2 + \left(\frac{z_s}{c}\right)^2 - 1 = 0
\]

Cylinder's surface is defined by the set of points \( \{(x_s, y_s, z_s)\} \) satisfying the equation:

\[
(x_s)^2 + (y_s)^2 - r^2 = 0 \quad \text{where} \quad -z_0 \leq z_s \leq z_0
\]

Centers at origin
Ray/Ellipsoid Intersection
Ray/Cylinder Intersection

- Substitute ray equation into surface equations
- This is a quadratic equation in t:
  - $At^2 + Bt + C = 0$
- Analyze as before
- Solve for t with quadratic formula
- Plug t back into ray equation - Done
- Well,… not exactly
Ray/Cylinder Intersection

- Is intersection point $P_i$ between $-Z_0$ and $Z_0$?
- If not, $P_i$ is not valid
- Also need to do intersection test with $z = -Z_0$, $Z_0$ plane
- If $(P_ix)^2 + (P_iy)^2 \leq r^2$, you’ve intersected a “cap”
- Which valid intersection is closer?
Torus

- Product of two implicit circles
  \[(x - R)^2 + z^2 - r^2 = 0\]
  \[(x + R)^2 + z^2 - r^2 = 0\]
  \[((x - R)^2 + z^2 - r^2)((x + R)^2 + z^2 - r^2)\]
  \[= (x^2 - 2Rx + R^2 + z^2 - r^2)(x^2 + 2Rx + R^2 + z^2 - r^2)\]
  \[= x^4 + 2x^2z^2 + z^4 - 2x^2r^2 - 2z^2r^2 + r^4 - 2x^2R^2 + 2z^2R^2 - 2r^2R^2 + R^4\]
  \[= (x^2 + z^2 - r^2 - R^2)^2 + 4z^2R^2 - 4r^2R^2\]

- Surface of rotation: replace \(x^2\) with \(x^2 + y^2\)
  \[f(x, y, z) = (x^2 + y^2 + z^2 - r^2 - R^2)^2 + 4R^2(z^2 - r^2)\]

- Quartic !!!
- Up to four ray torus intersections
Superquadrics

\[
\mathbf{x}(\eta, \omega) = \begin{bmatrix}
    a_1 \cos^{\varepsilon_1} \eta \cos^{\varepsilon_2} \omega \\
    a_2 \cos^{\varepsilon_1} \eta \sin^{\varepsilon_2} \omega \\
    a_3 \sin^{\varepsilon_1} \eta
\end{bmatrix}
\]

\[-\pi/2 \leq \eta \leq \pi/2\]
\[-\pi \leq \omega < \pi\]

\[
\left[ \left( \frac{x}{a_1} \right)^{2/\varepsilon_2} + \left( \frac{y}{a_2} \right)^{2/\varepsilon_2} \right]^{\varepsilon_2/\varepsilon_1} + \left( \frac{z}{a_3} \right)^{2/\varepsilon_1} = 1
\]
Beziers Patch

- Patch of order \((n, m)\) can be defined in terms of a set of \((n + 1)(m + 1)\) control points \(P_{i+1,j+1}\) for integer indices \(i = 0\) to \(n\), \(j = 0\) to \(m\).

\[(u,v) \in [0,1]^2\]

\[
p(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B^n_i(u) B^m_j(v) P_{i+1,j+1}
\]

\[
B^n_i(u) = \binom{n}{i} u^i (1 - u)^{n-i}
\]
Tesselate Patches and Superquadrics
Utah Teapot

- Modeled by 32 Bézier Patches
- Control points available at http://www.holmes3d.net/graphics/teapot
SMF Triangle Meshes

vertices

-1 -1 -1
1 -1 -1
-1 1 -1
1 1 -1
-1 -1 1
1 -1 1
-1 1 1
1 1 1

f 1 3 4
f 1 4 2
f 5 6 8
f 5 8 7
f 1 2 6
f 1 6 5
f 3 7 8
f 3 8 4
f 1 5 7
f 1 7 3
f 2 4 8
f 2 8 6

draw data structure
Triangle Meshes (.iv)

```inventor
#Inventor V2.0 ascii
ShapeHints {
  vertexOrdering     COUNTERCLOCKWISE
}
Separator {
  Coordinate3 {
    point [
      -1.5 -3.0 -0.3,
      1.5 -3.0 -0.3,
      1.5 3.0 -0.3,
      -1.5 3.0 -0.3,
      -1.5 -3.0 0.3,
      1.5 -3.0 0.3,
      1.5 3.0 0.3,
      -1.5 3.0 0.3
    ]
  }
}
IndexedFaceSet {
  coordIndex [
    0, 1, 2, -1,
    0, 2, 3, -1,
    7, 6, 5, -1,
    7, 5, 4, -1,
    0, 3, 7, -1,
    0, 7, 4, -1,
    1, 5, 6, -1,
    1, 6, 2, -1,
    0, 4, 5, -1,
    0, 5, 1, -1,
    3, 2, 6, -1,
    3, 6, 7, -1
  ]
}
```
Slide Credits

- Jonathan Cohen - Johns Hopkins
- Yizhou Yu - University of Illinois
- Kevin Suffern - University of Technology, Sydney, Australia
- G. Drew Kessler, Larry Hodges - Georgia Institute of Technology
Color Models
Frequency Spectrum

Spectrum describes frequency distribution of a light source

Wavelength (nm)

400 450 500 550 600 650 700

blue green yel. red

Amplitude
Definitions

Hue: quality that distinguishes one color family from another (i.e. red, yellow, green, blue, etc.)

Chroma: degree of color’s departure from greyscale

Value/Lightness: quality distinguishing light from dark colors
More definitions

Achromatic light: literally light without chroma, or greyscale light

- fairly uniform frequency distribution

Monochromatic light: light which has all intensity near a single frequency
Color Mixture - Subtractive

Applies when mixing pigments and dyes

- Each substance absorbs certain frequencies
- Combining substances absorbs the union of these frequencies
- Resulting reflected light is intersection of colors reflected by each
Subtractive Mixture Example

CMY(K) is a Subtractive Color Model

• Primary colors:
  – cyan, magenta, yellow

• Secondary colors:
  – blue = cyan \land magenta
  – red = magenta \land yellow
  – green = yellow \land cyan

• All colors:
  – black = cyan \land magenta \land yellow (in theory).
  – Black (K) ink is used in addition to C,M,Y to produce solid black.
  – white = no color of ink (on white paper, of course).

http://prometheus.cecs.csulb.edu/~jewett/colors/
Color Mixture - Additive

Applies to mixing of luminescent colors, such as color CRT and LCD displays, etc.

- Color refers to actual frequency spectrum of light
- Combining lights adds their frequency spectra
Additive Color Example
RGB is an Additive Color Model

• Primary colors:
  – red, green, blue

• Secondary colors:
  – yellow = red + green,
  – cyan = green + blue,
  – magenta = blue + red.

• All colors:
  – white = red + green + blue (#FFFFFF)
  – black = no light (#000000).

http://prometheus.cecs.csulb.edu/~jewett/colors/
3 Types of retinal cones

Efficient Color Computations in Computer Graphics

Represent frequency spectrum as discrete set of samples

- Typically 3 samples: red, green, and blue
- Monitors also use samples corresponding to different phosphors
- Eye also has 3 samples (types of cones)

Does not imply that three samples for initial and intermediate produce accurate computations
Color Space Gamut

Color gamut: subspace of visible colors

No system of mixing colors from fixed number of primaries can represent all visible colors

Color Spaces - RGB cube

Shortcomings:

- perceptually non-linear
- non-intuitive for human specification

from Foley, vanDam, Feiner, and Hughes, Computer Graphics: Principles and Practice, plate II.4
Color Spaces - HSV hexacone

Still not perceptually linear

Axes correspond to more intuitive perceptual qualities

- Selection similar to artist color mixing
- choose hue of base pigment, add white, add black

Derived from projections of RGB cube

From Alan Watt, 3D Computer Graphics, 2nd edition, p. 419
CIE Color Space

Employs 3 artificial primaries: X, Y, Z

- Mathematical abstractions, not physically realizable
- Allow supersaturation

Larger than visible spectrum

Standard for representing colors and converting between spaces
CIE Space and Device Gamuts

Chromaticity Diagram

from Foley, vanDam, Feiner, and Hughes, Computer Graphics: Principles and
Practice, plates II.1 and II.2

Johns Hopkins Department of Computer Science
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen
Gamma Correction

Exponential function converts from device-independent RGB space to device-dependent RGB

- Gamma is exponent
- Every monitor is different
- Monitor color intensities are non-linear with respect to phosphor excitation levels
Light Models
Things to Model

Light sources
- What color, intensity, lines through space

Reflection of light off surfaces
- How much light reflected in each direction
  - How are color and intensity changed
Real Lights

Real lights are complicated

- Sun light, iridescent bulbs, fluorescent bulbs
- Different spectra in different directions

— probably time-varying as well, but we don’t perceive much of that
Simpler Light Models

- Point lights
- Directional lights
- Spot (Warn) lights
- Area lights (not really so simple)
Real Reflection

Again, pretty complicated

• May be described by bidirection reflectance distribution function (BRDF)

• BRDF is 5D function
  — 2D for incoming light direction
  — 2D for outgoing light direction
  — 1D for wavelength of light
Life on a Surface

L: direction to light
N: normal vector
R: reflection of light about normal
V: direction to viewer (i.e. reflection direction of interest)
Point Light

Specified by:
- position \((x,y,z)\)
- intensity \((r,g,b)\)

Radiates equal intensity in all directions

\[ L = P_{\text{light}} - P_{\text{surface}} \]
Directional Light

- Point light at infinity
- Specified by:
  - direction \((x,y,z)\)
  - intensity \((r,g,b)\)
- All light rays are parallel
- \(L = -\text{direction}\)

point light at infinity
Spot Light

- Specified by
  - Position, Direction
  - Cone angle, sharpness

- \[ L = P_l - P_s \]
- if \( \phi > \theta \)
  - \[ I_L = 0 \]
- else
  - \[ I_L = \cos^n\left(\frac{\pi}{2}\left(\frac{\phi}{\theta}\right)\right) \]
Shading Models
Consider the process of looking at (or photographing) a scene through a window.

At every point on the window the light from the scene has a certain intensity $I(x, y)$.

Intensity is a measure of the brightness of the light.

Generally there will be a mixture of light of different wavelengths at each point on the window: $I = I(x, y, \lambda)$ where $\lambda$ is the wavelength.

We don't discuss the wavelength dependency in these notes.
Types of illumination

Direct illumination

Light striking the objects comes directly from the light sources
Indirect illumination

Light striking the objects has been reflected off at least one other object since leaving the light source.

The light could also have been refracted through transparent objects.
Reflection Models

There exists a number of reflection models that model the real world to varying degrees of realism.

In general, the more realistic a model is, the more physics and computation is involved.

Most reflection models incorporate direct illumination.

The indirect illumination is very expensive to calculate.

Only the most advanced rendering techniques incorporate indirect illumination.

Examples are ray tracing and radiosity.
Some more terminology

Direct illumination is referred to as *local* illumination.

Indirect illumination is referred to as *global* illumination.
Determining an Object’s Appearance

Ultimately, we’re interested in modeling light transport in scene
- Light is emitted from light sources and interacts with surfaces
- on impact with an object, some is reflected and some is absorbed
- distribution of reflected light determines “finish” (matte, glossy, …)
- composition of light arriving at eye determines what we see

Let’s focus on the local interaction of light with single surface point
Interaction of light and objects.

We must also model the way that light interacts with the surfaces of objects.

When light hits the surface of an object, it can be:

- Reflected
- Absorbed
- Transmitted (for transparent objects)

We see opaque objects by the light they reflect.
The way in which light interacts with real objects is very complicated.

The interaction depends on:

- the material properties of the object
- the wavelength of the incident light
- the angle of incidence of the light and the surface

The *polarisation* of the light is also involved, but rarely modelled in computer graphics.
Basic Local Illumination Model

We’re only interested in light that finally arrives at view point
  • a function of the light & viewing positions
  • and local surface reflectance

Characterize light using RGB triples
  • can operate on each channel separately

Given a point, compute intensity of reflected light
Phong Illumination

Empirically divides reflection into 3 components

• Ambient
• Diffuse (Lambertian)
• Specular
Diffuse Reflection

This is the simplest kind of reflection
  • also called Lambertian reflection
  • models dull, matte surfaces — materials like chalk

Ideal diffuse reflection
  • scatters incoming light equally in all directions
  • identical appearance from all viewing directions
  • reflected intensity depends only on direction of light source

Light is reflected according to Lambert’s Law
Diffuse reflection

The incident light is scattered equally in all directions

This is characteristic of dull, matt surfaces such as paper, bricks, carpet, etc.
Lambert’s Law for Diffuse Reflection

**Purely diffuse object**

\[ I = I_L k_d \cos \theta \]
\[ = I_L k_d (\mathbf{n} \cdot \mathbf{L}) \]

- \( I \): resulting intensity
- \( I_L \): light source intensity
- \( k_d \): (diffuse) surface reflectance coefficient
  \[ k_d \in [0,1] \]
- \( \theta \): angle between normal & light direction
**Specular Reflection**

Diffuse reflection is nice, but many surfaces are shiny
- their appearance changes as the viewpoint moves
- they have glossy *specular highlights* (or specularities)
- because they reflect light coherently, in a preferred direction

A mirror is a perfect specular reflector
- incoming ray reflected about normal direction
- nothing reflected in any other direction

Most surfaces are imperfect specular reflectors
- reflect rays in cone about perfect reflection direction
Specular reflection

The reflected light is concentrated around the direction of *mirror reflection*, and is spread out.

This can be used to model shiny surfaces.
**Phong Illumination Model**

\[ I = I_L k_d \cos \theta + I_L k_s \cos^n \phi \]
\[ = I_L k_d (\mathbf{n} \cdot \mathbf{L}) + I_L k_s (\mathbf{r} \cdot \mathbf{v})^n \]

One particular specular reflection model
- quite common in practice
- it is purely empirical
- there's *no physical basis* for it

\( I \): resulting intensity
\( I_L \): light source intensity
\( k_s \): (specular) surface reflectance coefficient
\( k_s \in [0,1] \)
\( \phi \): angle between viewing & reflection direction
\( n \): "shininess" factor
Calculating the Reflected Ray

\[ r = \left( 2 \ast (n \cdot L) \ast n \right) - L \]

- Derivation left for the students
- Clamp all dot products to zero. They shouldn’t be negative, but they can be
  - \( \text{MAX} \ (0, n \cdot L) \)
\[
\begin{align*}
1.0 & = \cos \theta \\
& = \cos^2 \theta \\
& = \cos^{10} \theta \\
& = \cos^{25} \theta
\end{align*}
\]
Diffuse + specular reflection
Examples of Phong Specular Model

Diffuse only

Diffuse + Specular (shininess 5)

Diffuse + Specular (shininess 50)
The Ambient Glow

So far, areas not directly illuminated by any light appear black
  • this tends to look rather unnatural
  • in the real world, there’s lots of ambient light

To compensate, we invent new light source
  • assume there is a constant ambient “glow”
  • this ambient glow is *purely fictitious*

Just add in another term to our illumination equation

\[ I = I_L k_d \cos \theta + I_L k_s \cos^n \phi + I_a k_a \]

*\(I_a\): ambient light intensity
*\(k_a\): (ambient) surface reflectance coefficient*
Our Three Basic Components of Illumination

- Diffuse
- Specular
- Ambient
Combined for the Final Result
Light-source Attenuation

Thus far we have ignored the inverse square law: energy decays with the inverse square of the distance \( d_L \) to the light source. Including this term we get

\[
I = I_a k_a + I_p k_d (L \cdot N)/d_L^2
\]

However, due to our previous assumptions of a point light source and uniform ambient light, using the \( d_L^2 \) term gives too rapid of a decrease in illumination intensity to look realistic. The \( 1/d_L^2 \) term is usually replaced by \( f_{\text{att}} \) where

\[
f_{\text{att}} = \text{MIN} \left( \frac{1}{c_1 d_L}, 1 \right)
\]

\[
I = I_a k_a + I_p k_d (N \cdot L)*f_{\text{att}}
\]
Shading Polygons: Flat Shading

Illumination equations are evaluated at surface locations
  • so where do we apply them?

We could just do it once per polygon
  • fill every pixel covered by polygon with the resulting color
Shading Polygons: Gouraud Shading

Alternatively, we could evaluate at every vertex
- compute color for each covered pixel
- linearly interpolate colors over polygon

Misses details that don’t fall on vertex
- specular highlights, for instance
Shading Polygons: Phong Shading

Don’t just interpolate colors over polygons

Interpolate surface normal over polygon
  • evaluate illumination equation at each pixel
Summarizing the Shading Model

We describe local appearance with illumination equations
- consists of a sum of set of components — light is additive
- treat each wavelength independently
- currently: diffuse, specular, and ambient terms

\[ I = I_L k_d \cos \theta + I_L k_s \cos^n \phi + I_a k_a \]

Must shade every pixel covered by polygon
- flat shading: constant color
- Gouraud shading: interpolate corner colors
- Phong shading: interpolate corner normals
Including Surface Color

\[ I = \]

for all lights \{ \]
\[ I_L \times k_d \times C \times \cos(\theta) + I_L \times k_s \times C \times \cos^n(\phi) \]
\[ + I_a \times k_a \times C \]

\[ C \] - RGB surface color

Could define \( C_d \), \( C_s \) and \( C_a \), but you don’t have to
Cook and Torrance Illumination

Replace specular component with more physically accurate model

\[ \rho_s = F_\lambda DG/\pi[(N \cdot V)(N \cdot L)] \]

- \( F_\lambda \) is Fresnel term, which accounts for change of highlight color with respect to angle of incidence
- \( D \) is microfacet distribution term, for more accurate measurement specular reflection off tiny microfacets
- \( G \) is geometry term, which models self-shadowing effects
Cook and Torrance Illumination

(a)  
(b)  
(c)  
(d)  

Distribution Functions
Cook and Torrance Illumination

Results
Phong vs. Cook/Torrance Example

Fig. 16.44 Comparison of Phong and Torrance–Sparrow illumination models for light at a 70° angle of incidence. (By J. Blinn [BLIN77a], courtesy of the University of Utah.)

Too Intense

With multiple light sources, it is easy to generated values of $I > 1$

One solution is to set the color value to be $\text{MIN}(I, 1)$

- An object can change color, saturating towards white
  
  Ex. $(0.1, 0.4, 0.8) + (0.5, 0.5, 0.5) = (0.6, 0.9, 1.0)$

Another solution is to renormalize the intensities to vary from 0 to 1 if one $I > 1$.

- Requires calculating all $I$’s before rendering anything.
- No over-saturation, but image may be too bright, and contrasts a little off.

Image-processing on image to be rendered (with original $I$’s) will produce better results, but is costly.
Calculating Normals

- Create vector structure (for normals) same size as vertex structure
- For each face
  - Calculate unit normal
  - Add to normal structure using vertex indices
- Normalize all the normals
  - \( N(\alpha, \beta, \gamma) = \alpha N_a + \beta N_b + \gamma N_c \)
Normal for Triangle

\[ \text{plane } \mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0 \]

\[ \mathbf{n} = (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0) \]

normalize \( \mathbf{n} \leftarrow \mathbf{n} / |\mathbf{n}| \)

Note that right-hand rule determines outward face
for (i=0; i++; i < h_{res})
    for (j=0; j++; j < v_{res}) {
        Calculate Ray
        for (k=0; k++; k < NumObject) {
            Attempt to intersect Ray with Object_k
            for all intersections
            Save t, ObjectID & xsect pt/param
        }
        Perform shading calculation on closest point
        (intersection with the lowest non-negative t)
        Set pixel value at Image[i,j]
    }
Supersample & Average Image

- $h_{\text{res}}$ and $v_{\text{res}}$ are even

```c
for (i=0; i++; i < h_{\text{res}}/2)
    for (j=0; j++; j < v_{\text{res}}/2) {
        offset = 2 * [i,j];
        newImage[i,j] = (oldImage[offset] +
                        oldImage[offset + [0,1]] +
                        oldImage[offset + [1,0]] +
                        oldImage[offset + [1,1]])/4;
    }
```
Wrap Up

- Discuss next week’s programming assignment
  - Add point lights with color
  - Add color, shading parameters and normals to models
  - Phong shading
  - Supersample image

- Discuss status/problems/issues with this week’s programming assignment