Lighting, Illumination, and Shading

CPSC 314

The Rendering Pipeline

Geometry Database → Model/View Transform. → Lighting → Perspective Transform. → Clipping

Geometry Processing

Scan Conversion → Texturing

Fragment Processing

Depth Test → Blending → Frame-buffer
Homogeneous Coordinates

**Homogeneous representation of points:**

- Add an additional component $w=1$ to all points
- All multiples of this vector are considered to represent the same 3D point
- All points are represented as column vectors

\[
\begin{pmatrix}
  x \\
  y \\
  z \\
  1
\end{pmatrix}
= 
\begin{pmatrix}
  x' \\
  y' \\
  z' \\
  w
\end{pmatrix}
= 
\begin{pmatrix}
  x \\
  y \\
  z \\
  w
\end{pmatrix}
\]

\( \forall w \neq 0 \)

Homogeneous Matrices

**Affine Transformations**

\[
\begin{bmatrix}
  x' \\
  y' \\
  z'
\end{bmatrix}
= 
\begin{bmatrix}
  m_{1,1} & m_{1,2} & m_{1,3} & 0 \\
  m_{2,1} & m_{2,2} & m_{2,3} & 0 \\
  m_{3,1} & m_{3,2} & m_{3,3} & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}
\cdot
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
+ 
\begin{bmatrix}
  t_x \\
  t_y \\
  t_z \\
  0
\end{bmatrix}
\]

\[
= 
\begin{bmatrix}
  m_{1,1} & m_{1,2} & m_{1,3} & 0 \\
  m_{2,1} & m_{2,2} & m_{2,3} & 0 \\
  m_{3,1} & m_{3,2} & m_{3,3} & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}
\cdot
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
+ 
\begin{bmatrix}
  0 & 0 & 0 & t_x \\
  0 & 0 & 0 & t_y \\
  0 & 0 & 0 & t_z \\
  0 & 0 & 0 & 0
\end{bmatrix}
\cdot
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
\]
Homogeneous Matrices

Affine Transformations

\[
T \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & t_x \\ m_{2,1} & m_{2,2} & m_{2,3} & t_y \\ m_{3,1} & m_{3,2} & m_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}
\]

Perspective Projection

Example:

- Assume image plane at \( z = -1 \)
- A point \([x, y, z, 1]^T\) projects to

\[
[-x/z, -y/z, -z/z, 1]^T = [x, y, z, -z]^T
\]
**Perspective Projection**

**Analysis:**
- This is a special case of a general family of transformations called *projective transformations*.
- These can be expressed as 4x4 homogeneous matrices!
  - *E.g. in the example:*
    \[
    T \begin{bmatrix}
    x \\
    y \\
    z \\
    1
    \end{bmatrix} = \begin{bmatrix}
    1 & 0 & 0 & 0 \\
    0 & 1 & 0 & 0 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & -1 & 0
    \end{bmatrix} \begin{bmatrix}
    x \\
    y \\
    z \\
    1
    \end{bmatrix} = \begin{bmatrix}
    x \\
    y \\
    z \\
    -z
    \end{bmatrix} = \begin{bmatrix}
    -x/z \\
    -y/z \\
    -1 \\
    1
    \end{bmatrix}
    \]

---

**Transformation Hierarchy**

A detailed diagram showing the hierarchical relationship between different body parts and their transformations, including rotations and translations.

*E.g.*
- Trans(0.3,0,0) Rot(z,θ)

© Wolfgang Heidrich
Transformation Hierarchy

glTranslatef(x, y, 0);
glRotatef(\(\theta\), 0, 0, 1);
DrawBody();
glPushMatrix();
glTranslatef(0, 0, 1);
DrawHead();
glPopMatrix();
glPushMatrix();
glTranslatef(2.5, 5.5, 0);
DrawUArm();
glTranslatef(0, -3.5, 0);
DrawLArm();
glPopMatrix();
... (draw other arm)

Display Lists

**Concept:**
- If multiple copies of an object are required, it can be compiled into a display list:

```c
glNewList(listId, GL_COMPILE);

  glBegin( ...);
  // geometry goes here
  glEndList();

// render two copies of geometry offset by 1 in z-direction:

glCallList(listId);
glTranslatef(0.0, 0.0, 1.0);

glCallList(listId);
```
Display Lists

**Advantages:**

- More efficient than individual function calls for every vertex/attribute
- Can be cached on the graphics board (bandwidth!)
- Display lists exist across multiple frames
  - Represent static objects in an interactive application

Triangle Strips and Fans

**Triangle strips:**

- Encode neighboring triangles that share vertices
- Use an encoding that requires only a constant-sized part of the whole geometry to determine a single triangle
- N triangles need n+2 vertices
Triangle Strips and Fans

Transformations:
- $n + 2$ for $n$ triangles
- Only requires 3 vertices to be stored according to simple access scheme
-Ideal for pipeline (local knowledge)

Generation:
- E.g. from directed edge data structure
- Optimize for longest strips/fans

---

Triangle Strips and Fans

In OpenGL

```gl
glBegin (GL_TRIANGLE_STRIP );
glVertex3f( x1, y1, z1 ); // tri 1, vert 1
eglVertex3f( x2, y2, z2 ); // tri 1, vert 2 and tri 2, vert 1
glVertex3f( x3, y3, z3 ); // tri 1, vert 3 and tri 2, vert 2
glVertex3f( x4, y4, z4 ); // tri 2, vert 3...
glEnd();
```
**Vertex Arrays**

**Concept:**
- Store array of vertex data for meshes with arbitrary connectivity (topology)

```c
GLfloat *points[3*nvertices];
GLfloat *colors[3*nvertices];
Glint *tris[numtris]=
  {0,1,3, 3,2,4, ...};
glVertexPointer( ..., points );
glColorPointer( ...,colors );
glDrawElements( GL_TRIANGLES, ...,tris );
```

**Benefits:**
- Ideally, vertex array fits into memory on graphics chip
- Then all vertices are transformed exactly once

**In practice:**
- Graphics memory may not be sufficient to hold model
- Then either:
  - Cache only parts of the vertex array on board (may lead to cache trashing!)
  - Transform everything in software and just send results for individual triangles (bandwidth problem: multiple transfers of same vertex!)
The Rendering Pipeline

Illumination

**Goal**

Model interaction of light with matter in a way that appears realistic and is fast

- Phenomenological reflection models
  - Ignore real physics, approximate the look
  - Simple, non-physical
  - Phong, Blinn-Phong

- Physically based reflection models
  - Simulate physics
  - BRDFs: Bidirectional Reflection Distribution Functions
Photorealistic Illumination

[electricimage.com]

© Wolfgang Heidrich

Photorealistic Illumination

[electricimage.com]

© Wolfgang Heidrich
Fast Local Illumination

Illumination

- Transport of energy from light sources to surfaces & points
  - Includes direct and indirect illumination

Images by Henrik Wann Jensen
Components of Illumination

Two components
- Light sources and surface properties

Light sources (or emitters)
- Spectrum of emittance (i.e., color of the light)
- Geometric attributes
  - Position
  - Direction
  - Shape
- Directional attenuation
- Polarization

Components of Illumination

Surface properties
- Reflectance spectrum (i.e., color of the surface)
- Subsurface reflectance
- Geometric attributes
  - Position
  - Orientation
  - Micro-structure
Illumination as Radiative Transfer

**Radiative heat transfer analogon**
- Substitute light for heat
- Light as packets of energy (photons)
  - *Particles not waves*
- Model light transport as packet flow

![Diagram showing heat/light source, energy packets, and reflective objects](image)

Light Transport Assumptions

**Geometrical optics:**
- Light is photons not waves
- No diffraction
- No polarization (some sunglasses)
  - *Light of all orientations gets through*
- No interference (packets don’t interact)
  - *Which visual effects does this preclude?*

![Diagram showing light particles, single slit, light waves, and bent ray](image)
Light Transport Assumptions II

- **Color approximated by discrete wavelengths**
  - Quantized approx of dispersion (rainbows)
  - Quantized approx of fluorescence (cycling vests)

- **No propagation media (surfaces in vacuum)**
  - No
    - Atmospheric scattering (fog, clouds)
    - Refraction (mirages)
    - Gravity lenses
  - But methods exist for all these effects

- **Superposition (lights can be added)**
  - No nonlinear reflection models
    - Pretty good assumption (only few non-linear materials)

Light Sources and Materials

- **Appearance depends on**
  - Light sources, locations, properties
  - Material (surface) properties
  - Viewer position

- **Local illumination**
  - Compute at material, from light to viewer

- **Global illumination (later in course)**
  - Ray tracing: from viewer into scene
  - Radiosity: between surface patches
Illumination in the Rendering Pipeline

**Local illumination**
- Only models light arriving directly from light source
- No interreflections and shadows
  - Can be added through tricks, multiple rendering passes

**Light sources**
- Simple shapes

**Materials**
- Simple, non-physical reflection models

---

**Light Sources**

**Types of light sources**
- Directional/parallel lights
  - E.g. sun
  - Homogeneous vector
- (Homogeneous) point lights
  - Same intensity in all directions
  - Homogeneous point
- Spot lights
  - Limited set of directions
  - Point + direction + cutoff angle
Light Sources

**Area lights:**
- Light sources with a finite area
- Can be considered a continuum of point lights
- Not available in many rendering systems

Light Sources

**ambient lights**
- no identifiable source or direction
- hack for replacing true global illumination
  - *(light bouncing off from other objects)*
**Ambient Light Sources**

- Scene lit only with an ambient light source

- Light Position: Not Important
- Viewer Position: Not Important
- Surface Angle: Not Important

**Directional Light Sources**

- Scene lit with directional and ambient light

- Surface Angle: Important
- Light Position: Not Important
- Viewer Position: Not Important
Point Light Sources

- Scene lit with ambient and point light source

Light Sources

Geometry: positions and directions
- Standard: world coordinate system
  - Effect: lights fixed wrt world geometry
  - Demo: http://www.xmission.com/~nate/tutors.html
- Alternative: camera coordinate system
  - Effect: lights attached to camera (car headlights)
- Points and directions undergo normal model/view transformation

illuminatioon calculations: camera coords
Types of Reflection

- *Specular* (a.k.a. *mirror* or *regular*) reflection causes light to propagate without scattering.

- *Diffuse* reflection sends light in all directions with equal energy.

- *Mixed* reflection is a weighted combination of specular and diffuse.

Types of Reflection

- *retro-reflection* occurs when incident energy reflects in directions close to the incident direction, for a wide range of incident directions.

- *gloss* is the property of a material surface that involves mixed reflection and is responsible for the mirror like appearance of rough surfaces.
**Reflectance Distribution Model**

*Most surfaces exhibit complex reflectances*

- Vary with incident and reflected directions.
- Model with combination

\[
\text{specular + glossy + diffuse = reflectance distribution}
\]

**Surface Roughness**

- at a microscopic scale, all real surfaces are rough
- cast shadows on themselves
- “mask” reflected light:

\[
\text{shadow} \quad \text{shadow}
\]

\[
\text{Masked Light}
\]
Surface Roughness

**Notice another effect of roughness:**
- Each “microfacet” is treated as a perfect mirror.
- Incident light reflected in different directions by different facets.
- End result is mixed reflectance.
  - Smoother surfaces are more specular or glossy.
  - Random distribution of facet normals results in diffuse reflectance.

Physics of Diffuse Reflection

**Ideal diffuse reflection**
- Very rough surface at the microscopic level
  - *Real-world example: chalk*
- Microscopic variations mean incoming ray of light equally likely to be reflected in any direction over the hemisphere
- Reflected intensity only depends on light direction!
Lambert’s “Law”

Intuitively: cross-sectional area of the “beam” intersecting an element of surface area is smaller for greater angles with the normal.

Computing Diffuse Reflection

- Depends on angle of incidence: angle between surface normal and incoming light
  - \( I_{\text{diffuse}} = k_d I_{\text{light}} \cos \theta \)
- In practice use vector arithmetic
  - \( I_{\text{diffuse}} = k_d I_{\text{light}} (n \cdot l) \)
- Always normalize vectors used in lighting
  - \( n, l \) should be unit vectors
- Scalar (B/W intensity) or 3-tuple or 4-tuple (color)
  - \( k_d \): diffuse coefficient, surface color
  - \( I_{\text{light}} \): incoming light intensity
  - \( I_{\text{diffuse}} \): outgoing light intensity (for diffuse reflection)
**Diffuse Lighting Examples**

- Lambertian sphere from several lighting angles:

- need only consider angles from 0° to 90°

**Specular Reflection**

*Shiny surfaces exhibit specular reflection*

- Polished metal
- Glossy car finish

*Specular highlight*

- Bright spot from light shining on a specular surface

*View dependent*

- Highlight position is function of the viewer’s position
Physics of Specular Reflection

- At the microscopic level a specular reflecting surface is very smooth
- Thus rays of light are likely to bounce off the microgeometry in a mirror-like fashion
- the smoother the surface, the closer it becomes to a perfect mirror

Optics of Reflection

**Reflection follows Snell’s Law:**
- Incoming ray and reflected ray lie in a plane with the surface normal
- Angle the reflected ray forms with surface normal equals angle formed by incoming ray and surface normal

\[ \theta_{(l)ight} = \theta_{(r)eflection} \]
Non-Ideal Specular Reflectance

- Snell’s law applies to perfect mirror-like surfaces, but aside from mirrors (and chrome) few surfaces exhibit perfect specularity
- How can we capture the “softer” reflections of surface that are glossy, not mirror-like?
- One option: model the microgeometry of the surface and explicitly bounce rays off of it

or...

Empirical Approximation

- We expect most reflected light to travel in direction predicted by Snell’s Law
- But because of microscopic surface variations, some light may be reflected in a direction slightly off the ideal reflected ray
- As angle from ideal reflected ray increases, we expect less light to be reflected
Empirical Approximation

*Angular falloff*

*how might we model this falloff?*

Phong Lighting

*Most common lighting model in computer graphics*

– (Phong Bui-Tuong, 1975)

\[
I_{\text{specular}} = k_s I_{\text{light}} (\cos \phi)^{n_{\text{shiny}}}
\]

- \(n_{\text{shiny}}\): purely empirical constant, varies rate of falloff
- \(k_s\): specular coefficient, highlight color
  no physical basis, works ok in practice
**Phong Lighting: The $n_{shiny}$ Term**

- Phong reflectance term drops off with divergence of viewing angle from ideal reflected ray.

**what does this term control, visually?**

Viewing angle – reflected angle

---

**Phong Examples**

- Varying $I$

- Varying $n_{shiny}$
Calculating Phong Lighting

\[ I_{\text{specular}} = k_s I_{\text{light}} (v \cdot r)^n_{\text{shiny}} \]

- \( v \): unit vector towards viewer/eye
- \( r \): ideal reflectance direction (unit vector)
- \( k_s \): specular component
  - highlight color
- \( I_{\text{light}} \): incoming light intensity

*how to efficiently calculate \( r \) ?*

Computing the Reflected Direction

**Specular/Glossy**

- Computing reflection direction \( r_1 \) of \( l \)
  - \( n \) and \( l \) are unit length!
**Phong Lighting: Intensity Plots**

<table>
<thead>
<tr>
<th>Phong</th>
<th>$\rho_{\text{ambient}}$</th>
<th>$\rho_{\text{diffuse}}$</th>
<th>$\rho_{\text{specular}}$</th>
<th>$\rho_{\text{total}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi = 60^\circ$</td>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
</tr>
<tr>
<td>$\phi = 25^\circ$</td>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
<td><img src="image7.png" alt="Image" /></td>
<td><img src="image8.png" alt="Image" /></td>
</tr>
<tr>
<td>$\phi = 0^\circ$</td>
<td><img src="image9.png" alt="Image" /></td>
<td><img src="image10.png" alt="Image" /></td>
<td><img src="image11.png" alt="Image" /></td>
<td><img src="image12.png" alt="Image" /></td>
</tr>
</tbody>
</table>

---

**Alternative Model**

**Blinn-Phong model (Jim Blinn, 1977)**

- Variation with better physical interpretation
  - $h$: halfway vector; $r$: roughness

\[
I_{out}(x) = k_s \cdot (h \cdot n)^{1/r} \cdot I_{in}(x); \text{ with } h = (1 + v) / 2
\]
Light Source Falloff

**Quadratic falloff**

- Brightness of objects depends on power per unit area that hits the object
- The power per unit area for a point or spot light decreases quadratically with distance

![Diagram of quadratic falloff](image)

- Area $4\pi r^2$
- Area $4\pi (2r)^2$

**Light Source Falloff**

**Non-quadratic falloff**

- Many systems allow for other falloffs
- Allows for faking effect of area light sources
- OpenGL / graphics hardware
  - $I_o$: intensity of light source
  - $x$: object point
  - $r$: distance of light from $x$

\[
I_{in}(x) = \frac{1}{ar^2 + br + c} \cdot I_o
\]
Lighting Review

**Lighting models**

- Ambient
  - Normals don’t matter
- Lambert/diffuse
  - Angle between surface normal and light
- Phong/specular
  - Surface normal, light, and viewpoint

Lighting in OpenGL

**Light source: amount of RGB light emitted**

- Value represents percentage of full intensity
  E.g., (1.0, 0.5, 0.5)
- Every light source emits ambient, diffuse, and specular light

**Materials: amount of RGB light reflected**

- Value represents percentage reflected
  E.g., (0.0, 1.0, 0.5)

**Interaction: multiply components**

- Red light (1, 0, 0) x green surface (0, 1, 0) = black (0, 0, 0)
Lighting in OpenGL

```c
glLightfv(GL_LIGHT0, GL_AMBIENT, amb_light_rgba);
glLightfv(GL_LIGHT0, GL_DIFFUSE, dif_light_rgba);
glLightfv(GL_LIGHT0, GL_SPECULAR, spec_light_rgba);
glLightfv(GL_LIGHT0, GL_POSITION, position);
glEnable(GL_LIGHT0);

glMaterialfv(GL_FRONT, GL_AMBIENT, ambient_rgba);
glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse_rgba);
glMaterialfv(GL_FRONT, GL_SPECULAR, specular_rgba);
glMaterialfv(GL_FRONT, GL_SHININESS, n);
```
Lighting vs. Shading

**Lighting**
- Process of computing the luminous intensity (i.e., outgoing light) at a particular 3-D point, usually on a surface

**Shading**
- The process of computing pixel colors

Applying Illumination

**Lighting:**
- We now have an illumination model for a point on a surface

*If surface defined as mesh of polygonal facets, which points should we use?*
- Fairly expensive calculation
- Several possible answers, each with different implications for visual quality of result
Applying Illumination

*Polygonal/triangular models*

- Each facet has a constant surface normal
- If light is directional, diffuse reflectance is constant across the facet.
- *why?*

Flat Shading

- Simplest approach calculates illumination at a single point for each polygon
- obviously inaccurate for smooth surfaces
Flat Shading Approximations

If an object really is faceted, is this accurate?

no!

- For point sources, the direction to light varies across the facet

- For specular reflectance, direction to eye varies across the facet
**Improving Flat Shading**

*What if evaluate Phong lighting model at each pixel of the polygon?*

- Better, but result still clearly faceted

*For smoother-looking surfaces we introduce vertex normals at each vertex*

- Usually different from facet normal
- Used *only* for shading
- Think of as a better approximation of the *real* surface that the polygons approximate

---

**Vertex Normals**

*Vertex normals may be*

- Provided with the model
- Computed from first principles
- Approximated by averaging the normals of the facets that share the vertex
**Gouraud Shading**

*Most common approach, and what OpenGL does*

- Perform Phong lighting at the vertices
- Linearly interpolate the resulting colors over faces
  - Along edges
  - Along scanlines

Does this eliminate the facets?

**Gouraud Shading Artifacts**

*often appears dull, chalky*

*lacks accurate specular component*

- if included, will be averaged over entire polygon

This interior shading missed!

This vertex shading spread over too much area
Gouraud Shading Artifacts

**Mach bands**
- Eye enhances discontinuity in first derivative
- Very disturbing, especially for highlights

Discontinuity in rate of color change occurs here
Gouraud Shading Artifacts

**Perspective transformations**

- Affine combinations only invariant under affine, **not** under perspective transformations
- Thus, perspective projection alters the linear interpolation!

---

Gouraud Shading Artifacts

**Perspective transformation problem**

- Colors slightly “swim” on the surface as objects move relative to the camera
- Usually ignored since often only small difference
  - Usually smaller than changes from lighting variations
- To do it right
  - Either shading in object space
  - Or correction for perspective foreshortening
  - Expensive – thus hardly ever done for colors
Phong Shading

**Linearly interpolating surface normal across the facet, applying Phong lighting model at every pixel**

- Same input as Gouraud shading
- Pro: much smoother results
- Con: considerably more expensive

**Not the same as Phong lighting**

- Common confusion
- Phong lighting: empirical model to calculate illumination at a point on a surface

---

Phong Shading

**Linearly interpolate the vertex normals**

- Compute lighting equations at each pixel
- Can use specular component

\[ I_{total} = k_a I_{ambient} + \sum_{i=1}^{\# lights} I_i \left( k_d (n \cdot l_i) + k_s (v \cdot r_i)^{n_{shiny}} \right) \]

*remember: normals used in diffuse and specular terms*

Discontinuity in normal’s rate of change harder to detect
Phong Shading Difficulties

*Computationally expensive*
- Per-pixel vector normalization and lighting computation!
- Floating point operations required

*Lighting after perspective projection*
- Messes up the angles between vectors
- Have to keep eye-space vectors around

*no direct support in hardware*
- But can be simulated with texture mapping

Shading Artifacts: Silhouettes

*Polygonal silhouettes remain*

Gouraud  Phong
Shading Artifacts: Orientation

- Interpolation dependent on polygon orientation
- View dependence!

Interpolate between AB and AD

Interpolate between CD and AD

Rotate -90° and color same point

Shading Artifacts: Shared Vertices

- Vertex B shared by two rectangles on the right, but not by the one on the left

- First portion of the scanline is interpolated between DE and AC

- Second portion of the scanline is interpolated between BC and GH

- A large discontinuity could arise
**Shading Models Summary**

*Flat shading*
- Compute Phong lighting once for entire polygon

*Gouraud shading*
- Compute Phong lighting at the vertices and interpolate lighting values across polygon

*Phong shading*
- Compute averaged vertex normals
- Interpolate normals across polygon and perform Phong lighting across polygon

---

**The Rendering Pipeline**

<table>
<thead>
<tr>
<th>Geometry Database</th>
<th>Model/View Transform.</th>
<th>Lighting</th>
<th>Perspective Transform.</th>
<th>Clipping</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Geometry Processing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Rasterization</strong></td>
<td>Scan Conversion</td>
<td>Texturing</td>
<td>Depth Test</td>
<td>Blending</td>
</tr>
<tr>
<td><strong>Fragment Processing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Frame-buffer</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Coming Up

**Friday:**
- Polygon Clipping
- Quiz 1

**Monday:**
- Victoria Day

**Wednesday:**
- Polygon Clipping / Visibility