INTRODUCTION TO
COMPUTER GRAPHICS AND
INTERACTION

LIGHTING AND SHADING

Christopher Peters
CST, KTH Royal Institute of Technology, Sweden
chpeters@kth.se
http://kth.academia.edu/ChristopherEdwardPeters

Based on DGI12 notes by Carl Henrik Ek
Image Synthesis

In computer graphics, create images based on a *model*

Recall:
An underlying process generates observations
Can control generation through parameters
Nice Results

"Christmas Baubles" by Jaime Vives Piqueres

"Distant Shores" by Christoph Gerber

"Still with Bolts" by Jaime Vives Piqueres

Christopher Peters  DH2323 Lighting & Shading  chpeters@kth.se
Some Constituents I

- Light
- Geometry
- Surface properties
- Anything else?

Christopher Peters  
DH2323 Lighting & Shading  
chpeters@kth.se
Some Constituents II

- Camera Model (pinhole)
Row Vs. Column Format

Remember this?

\[
\begin{align*}
\mathbf{r}_0 &= [x_0, y_0, z_0]^T \\
\mathbf{r}_d &= [x_d, y_d, z_d]^T, \quad ||\mathbf{r}_d|| = 1 \\
\mathbf{r}_t &= \mathbf{r}_0 + t \cdot \mathbf{r}_d
\end{align*}
\]

*One degree-of-freedom*
Row Vs. Column Format

\[ \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \neq [v_1 \ v_2 \ v_3] \quad (= [v_1 \ v_2 \ v_3]^T) \]

\[ \mathbf{Mv} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \]

\[ \mathbf{v}^T \mathbf{M}^T = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix} \]

\[ \mathbf{Mv} = (\mathbf{v}^T \mathbf{M}^T)^T \]
Homogeneous Coordinates

\[
\begin{bmatrix}
    x' \\
y' \\
z' \\
1
\end{bmatrix} =
\begin{bmatrix}
    \cdot & \cdot & \cdot & t_x \\
    \cdot & R & \cdot & t_y \\
    \cdot & \cdot & \cdot & t_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
\]

Allow common operations to be represented as matrices

- Translation, rotation, projection

For positions and vectors, in 3D:

- \((x,y,z,w)^T \Rightarrow (x/w, y/w, z/w)^T\) for \(w\neq 0\)
- \(w = 1.0\): position
- \(w = 0.0\): vector
Lighting and Shading

- In this lecture, you will apply knowledge about:
  - Some applied math, especially vector algebra

- What is shading?
  - Determining the colour of a pixel
  - Usually determined by a lighting model

- Why is it good?
  - Provides depth to perception of images
  - Adds a sense of realism
Applications

Photorealistic
Applications

Non-photorealistic
Lighting Vs. Shading

- **Lighting**
  - Interaction between materials and light sources

- **Shading**
  - Deciding the colour of a pixel
  - Based usually on a lighting model
  - Other methods possible too though
How To Implement?

- Theory
  - General classifications
  - Lighting fundamentals
    - Lambertian illumination
  - Some shading models
    - Flat, Gouraud, Phong
  - Extensions
- Practice
  - Maths programming (vector operations, normals, plane, angles, intersections)
Some Classifications

- **View Dependent**
  - Determine an image by solving the illumination that arrives through the view-port only

- **View Independent**
  - Determine the lighting distribution in an entire scene regardless of viewing position. Views are taken after lighting simulation by sampling the full solution to determine the view through the viewport
Some Classifications

- **Local Illumination**
  - Consider lighting effects only directly from the light sources and ignore effects of other objects in the scene (e.g. reflection off other objects)

- **Global Illumination**
  - Account for all modes of light transport
Why Go Local?

- Usually easy to control and express
  - Director's chair: important when you want a scene to look a certain way
- Fast
  - Easier to obtain real-time performance (or just tractable calculations)
- Do not require knowledge of the entire scene

But …

- Not as accurate or compelling as global models
How Can It Be Modelled?

- Use a *lighting model* as inspiration
- But real light extremely complicated to simulate
  - Light bounces around the environment
  - Heavy processing required even for coarse approximations
  - Simplifications allow real-time performance

- Lighting models:
  - Lambertian – we will consider this first
  - Phong – not to be confused with *Phong shading*
  - Blinn-Phong and others...
Simplifications

- Simplification #1: use *isotropic point* light sources
- Isotropic means that the light source *radiates energy equally* in all directions
  - Simplifies our light source energy equations that we’ll look at
  - When we mention light, we are really talking about *energy*

- Simplification #2: simulate only specific surface types
  - Makes it easier to specify materials and calculate reflections
  - But visually limited
Radiant Intensity

- Light is defined by its Radiant Intensity, $I$
  - Radiant Intensity is measured in Watts/sr
  - $sr$ is the solid angle (in steradians)
  - $I = \phi / 4\pi r^2$
  - $\phi$ is the energy leaving the surface per unit time
  - Known as power or flux and measured in Watts
  - But: it’s a point light source, so it radiates light equally in all directions
  - So $r^2 = 1$ (unit sphere)
    $$\Rightarrow I = \phi / 4\pi$$
- Now know energy leaving light source in any direction
Inverse Square Law

- But we want to know the energy *arriving* at a surface.
- This *irradiance*, E, may now be determined:
  - Irradiance is the flux per unit area at a point x, a distance r from the point light source.
  - We know the source radiates I Watts in all directions.
  - So the power is radiated through a sphere centred at the lightsource.
  - At a distance r from the source, the surface area of this sphere is $4\pi r^2$ => the power per unit area at x is: $E = \frac{\phi}{4\pi r^2}$.
  - This assumes the surface at x is perpendicular to the direction to the light source.
  - To handle all angles, we must apply the cosine rule.
The Cosine Law

- A surface orientated perpendicular to a light source will receive more energy than a surface orientated at an angle to the light source
  - More energy = brighter appearance
- The irradiance $E$ is proportional to $1/\text{area}$
- As the area increases, the irradiance decreases
  - As $\theta$ increases, the irradiance (thus surface brightness) decreases:

\[
\begin{align*}
\text{Light source} & \quad \text{Light source} \\
{0^\circ} & \quad 100\% \quad 30^\circ \quad 87\% \\
\text{d} & \quad \text{d} \cos \theta
\end{align*}
\]
Lambertian Illumination Model

- Cosine rule is used to implement Lambertian surfaces
  - Also known as *diffuse* surfaces
- Diffuse surfaces reflect light equally in all directions
- The surface is characterised by a reflectance parameter $\rho_d$

$$\forall \rho_d(x) = \frac{\phi_i}{\phi_r}$$

$\phi_i$ is the incident power
$\phi_r$ is the reflected power
- So the *reflectance* is the ratio of the total incident power to the total reflected power
Lambertian Illumination Model

- To shade a diffuse surface, we need to know:
  - The normal to the surface at the point to be shaded
  - The diffuse reflectance of the surface
  - The positions and powers of the light sources in the scene
- Assuming contribution is from a single isotropic light source:
  \[
  L_{rd}(x, z) = \left( \frac{\rho_d}{\pi} \right) \cos \theta \left( \frac{\phi_s}{4\pi d^2} \right)
  \]
  - \(\frac{\rho_d}{\pi}\) accounts for the reflectance attribute of the surface
  - \(\cos \theta \left( \frac{\phi_s}{4\pi d^2} \right)\) accounts for the orientation of the surface with respect to the light source
Lambertian Illumination Model

- This is *local illumination*
  - Only concerned with energy hitting the surface directly from light sources
  - Not concerned with light bouncing off other surfaces and hitting the surface

  => Models derived from it are also local
Basic Shading Models

- Flat, gouraud and phong shading
- Flat shading
  - Per polygon shading
- Gouraud shading
  - Interpolate (bilinearly) colour values to get tween pixels
  - Per vertex shading
- Phong shading
  - Interpolate normals
  - Per pixel shading
Flat Shading

- Constant shading
- Very fast
- Very simple
- Does not look very smooth
- Compute the colour of a polygon
- Use this as the colour for the whole polygon
Gouraud Shading

- Calculates the light intensity at each vertex in a polygon
- For each interior point in the polygon, we *interpolate* the light intensity determined at the vertices
- Given a starting value, and an end value, *interpolation* can be used to approximate intermediate values
  - Similar idea to the way in which colours are interpolated across the surface of a polygon
- We only need to do lighting calculations at the vertices
  - Fast!
- But lighting is only correct at the vertices
  - Unrealistic
Gouraud Shading

Gouraud

Light Source

Interior Point

N

N

N

L
Limitations

- Gouraud only calculates the actual lighting at the vertices of the polygon
Phong Shading

- To improve on Gouraud shading, interpolate normals across a surface
  - Lighting model then applied to each interior point in a polygon
- Must take care to ensure that all interpolated normals are of unit length
- This is known as *Phong Shading*
- Phong shading produces more accurate results than Gouraud Shading
- But slower!
Phong Shading

- Phong shading can reproduce highlights in the center of a polygon that Gouraud Shading may miss
Phong Illumination Model

• Lambertian Illumination model: only diffuse surfaces
  • Surfaces reflect light in all directions equally
• What about modelling shiny surfaces too?
  • Reflected radiance depends heavily on the outgoing direction
• Phong Illumination model consists of:
  • Lambertian Model for diffuse surfaces
  • A function to handle specular reflection
  • Ambient term to approximate all other light
Phong Illumination Model

NOT the same as Phong Shading
Phong Illumination Model

- Allows us to model many different types of surfaces
- Easy to control
- Not a very realistic model
  - But produces good results
- Each object has material data associated with it:
  - $\rho_a$ ambient reflectance
  - $\rho_d$ diffuse reflectance
  - $\rho_s$ specular reflectance
  - $n$ phong exponent (shininess parameter)
Materials

- Parameters:
- Interaction with light
- Reflective properties
- Components
  - $m_{\text{ambient}}$, $m_{\text{diffuse}}$, $m_{\text{specular}}$
- Proportion of each colour reflected
Lambertian Vs Phong

Lambertian Surface

Phong Illuminated Specular Surface
A little bit of OpenGL (1.2 ← old)

- Light sources
- LIGHT0 to LIGHT7
- Each light must be enabled ...
  ```c
  glEnable(GL_LIGHT1);
  ```
- Can specify light parameters using
  ```c
  glLightf{iv}(GL_LIGHT0, param, value);
  ```
- Some parameters
  ```c
  GL_AMBIENT
  GL_DIFFUSE
  GL_SPECULAR
  GL_POSITION
  ```
Shading in OpenGL 1.2

- To enable lighting use:
  
  ```
  glEnable(GL_LIGHTING);
  ```

  OpenGL does not support true Phong shading; it interpolates the intensities across each polygon.

  Gouraud shading
  
  ```
  glShadeModel(GL_SMOOTH);
  ```
Material Properties

- We can assign different properties to each side of a polygon
- To assign material properties:

  `glMaterial{if}v(face, param, value);`

  - `GL_FRONT`
  - `GL_BACK`
  - `GL_FRONT_AND_BACK`
  - `GL_AMBIENT`
  - `GL_DIFFUSE`
  - `GL_AMBIENT_AND_DIFFUSE`
  - `GL_SPECULAR`
  - `GL_EMISSION`
  - `GL_SHININESS`

  RGB Values
  Phong Exponent
Bump Mapping

- Lots of cool effects possible
- Bump mapping: modify surface normals for lighting calcs (not actual geometry)
- Query a heightmap
- See also: normal mapping
Shaders

- Modern way of implementing rendering techniques
- Various types:
  - Pixel
  - Vertex
  - Geometry
  - Tessellation
- Shader languages
  - HLSL, GLSL, CG

```
color outputs to pixel shader.

void main() in siv IN, out v2p OUT )
{
Input parameters include view project matrix ModelViewProj, view inverse transpose matrix ModelViewIT, and light vector LightVec.

OUT.Position = mul(IN.Position, ModelViewProj);
}
```

**Modern way of implementing rendering techniques**

**Various types:**

- **Pixel**
- **Vertex**
- **Geometry**
- **Tessellation**

**Shader languages**

- HLSL, GLSL, CG

Next lecture

- Next week (25\textsuperscript{th} April)
- 10:00 – 12:00 B2

- Lab support session
- 15:00-17:00 (starting now)
- 4V2Röd