Realistic Rendering

-- Shading Models--
Mastering 2D & 3D Graphics

- Building Realistic Polygon Surfaces
  - flat shading
  - gouraud shading
  - phong shading
  - fast phong shading

- Specular Reflection Models + Shadows!
  - intensity attenuation
  - texture mapping
  - bump mapping
  - environment mapping
Building Realistic...

*Polygon surfaces*

- Polygon surfaces...
  ...are simple and easy-to-use
  ...form polyhedra
  ...are specified by a boundary as vertices
  ...where each vertex is connected by an edge
  ...and the last vertex is linked to the first
  ...are limited by their flatness
Polygon surfaces can be shaded efficiently by...
  ...flat shading (also called constant or faceted shading)
  ...interpolated shading
  ...Gouraud shading
  ...Phong shading
  ...Fast Phong shading
Building Realistic…

*Polygon surfaces*

- Flat shading...
  - …is the simplest shading model for polygons
  - …applies an illumination model only once to determine a single intensity that is used to shade an entire polygon
  - …each polygon has the same color at all points
  - …shows abrupt color changes from one polygon to another when curved surfaces are approximated
  - …can be improved by subdividing the object
Building Realistic...

*Polygon surfaces*

- Flat shading...provides accurate rendering IF...
  
  ...the object is a polyhedron and is not an approximation of an object with a curved surface
  
  ...all light sources illuminating the object are sufficiently far from the surface so that $N \cdot L$ is constant over the surface
  
  ...the viewing position is sufficiently far from the surface so that $V \cdot R$ is constant over the surface
Subdividing polygons...

...assumes that we can reasonably approximate surface-lighting effects using small polygon facets with flat shading and calculating the intensity for each facet (say the center)

...multiplies the amount of data that must be passed to the raster device (requiring at least four times the number of polygons -- since subdivision occurs vertically & horizontally)
Subdividing polygons...
...but when do we know if we have done enough subdivision?
...can avoid excessive subdivision by comparing the color change from one polygon to another
...what kind of issues other than this do we face with subdivision of polygons? (well, the answer is based on the types of polygons used for the faces)
Building Realistic...

*Polygon surfaces*

- Remember...the orientation of a surface directly affects its appearance: a surface that faces away from a light source appears darker than one that faces toward it
  - This applies to surfaces approximated using polygons where the object's orientation changes abruptly at polygon edges
The orientation of a surface is represented by the surface normal (a vector pointing perpendicular to the surface).
• Interpolated shading...

  ...is an alternative to flat shading

  ...first associates the normal of the curved surface with the polygon vertices, and takes the normal at interior points to be a smoothly varying combination of those at the vertices

  ...this means the intensities are calculated at the vertices and then interpolated for interior points

  ...interior points are evaluated in scan line order using a Z-buffer or some other HLR/HSR algorithm
Building Realistic...

*Polygon surfaces*

- Gouraud shading and Phong interpolation...
  - ...take the next step by applying basic shading algorithms to an entire polygon mesh,
  - ...take advantage of information provided by adjacent polygons (with common edges) to simulate a smooth surface
  - ...are supported by most 3D graphics workstations
Gouraud shading...

...is also called intensity interpolation shading

...eliminates intensity discontinuities, except when there is a rapid change in slope

...extends the concept of interpolated shading by interpolating polygon vertex illumination values for the surface being approximated

...requires that the normal be known for each vertex
Building Realistic...

*Polygon surfaces*

- Gouraud shading...
  ...calculates the intensity at each vertex using an illumination model
  ...then shades each polygon by linear interpolation of vertex intensities along each edge and then between edges along each scan line
  ...can integrate the interpolation process with scan line conversion
Building Realistic...

*Polygon surfaces*

- Gouraud shading...
  
  ...the edges of each scan line are calculated from the vertex intensities and the intensities along a scan line from these

\[
\begin{align*}
I_a &= I_1 - (I_1 - I_2)(y_4 - y_s)/(y_1 - y_2) \\
I_b &= I_1 - (I_1 - I_4)(y_4 - y_s)/(y_1 - y_4)
\end{align*}
\]

For efficiency, the change in intensity from one pixel to the next is:

\[
\Delta I_s = \Delta x/(x_b - x_a) \cdot (I_b - I_a)
\]

and

\[
I_{sn} = I_{sn-1} + \Delta I_s
\]
Building Realistic...
*Polygon surfaces: Gouraud shading*

- So, a visible span on a scan line is filled in by interpolating the intensity values of the two edges bounding the span
  
  ...of course requires that our objects be represented by a polygon mesh model
  
  ...where at each vertex we compute the surface normal which is defined as the average of the normals of the polygons that surround this vertex
Building Realistic...

*Polygon surfaces: Gouraud shading*

- The intensity at each vertex can be computed according to our illumination model...
  ...as soon as the vertex normal is computed

- Once the intensities for all vertices are calculated...
  ...then we can begin to render the object

- Gouraud shading can be combined with a HLHSR algorithm to fill in the visible polygons along each scan line
Building Realistic...

**Polygon surfaces: Gouraud Shading**

- First of all...
  ...our polygon mesh data structure must contain a *vertex normal* for every vertex

- *(if you have an explicit vertex table, then for each (x,y,z) you need to expand it to include (x,y,z,vertex-normal))*
Building Realistic...

Polygon surfaces: Gouraud Shading

- The algorithm to compute the vertex normals is:
  
  For each vertex of the polygon mesh
  
  a. Initialize your \textit{sum} to zero
  
  b. For all polygons in the mesh
      
      i. Check if the polygon contains this vertex (or, \textit{a pointer to this vertex})
      
      ii. If it does, add this polygon's surface normal to \textit{sum}
  
  c. Normalize \textit{sum}
  
  d. Save the \textit{sum} as the vertex normal for this vertex
Next...our polygon mesh data structure should also contain the resulting *intensity* for every vertex. *If you have an explicit vertex table, then for each (x,y,z) you need to expand it to include (x,y,z,vertex-normal, intensity)*

The rendering process is similar to filling polygons with one *essential* difference!...we do not just fill with a single color, but with an intensity that is computed as a linear interpolation of the intensities at the vertices.
Building Realistic...

*Polygon surfaces: Gouraud Shading*

- Where the part of the fill algorithm that actually draws a filled line needs to be replaced by a loop that individually sets each pixel along the fill line
- Now we are ready to compute the intensity of a pixel $P_s$...
  ...where $P_s$ lies on the scan line from $P_a$ to $P_b$...
  ...where $P_a$ and $P_b$ are the intersections of the scan line with the polygon edges
Building Realistic...

*Polygon surfaces: Gouraud shading*

- We must first compute the intensities of Pa, Pb...which are linearly interpolated from the intensities of the vertices spanning the edges.

\[ I_a = I_1 - (I_1 - I_2) \frac{(y_1 - y_s)}{(y_1 - y_2)} \]

\[ I_b = I_1 - (I_1 - I_4) \frac{(y_1 - y_s)}{(y_1 - y_4)} \]
Once the intensities of $I_a$ and $I_b$ are computed...
...
the intensity of $I_s$ can be computed by linear interpolation:

$$I_s = I_b - (I_b - I_a) \frac{(x_b - x_s)}{(x_b - x_a)}$$

Since the intensity can be performed incrementally for a given scan line instead use...

$$\Delta I_s = \frac{\Delta x}{(x_b - x_a)}(I_b - I_a)$$
and $I_{sn} = I_{sn-1} + \Delta I_s$

For efficiency, $\Delta I_s$ is computed only once for every scan line; it then remains constant, so only one addition needs to be done for every pixel!!
Gouraud shading removes the intensity discontinuities associated with constant (flat) shading models
  – but, it has some other deficiencies
  – highlights on the surface are sometimes displayed with anomalous shapes
  – the linear intensity interpolation can cause bright or dark intensity streaks (called mach bands) to appear
  – we can reduce these effects by subdividing or using other shading techniques
Building Realistic...

*Polygon surfaces*

- Phong shading...
  - ...is called normal-vector interpolation shading
  - ...interpolates the surface normals instead of
    the intensity
  - ...calculates normals along polygon edges from
    vertex normals
  - ...interpolates across polygon spans on a scan
    line, between the starting and ending normals of
    the span
Building Realistic...

*Polygon surfaces*

- Phong shading...
  ...is a substantial improvement over Gouraud shading, because highlights are reproduced more faithfully
Building Realistic...

**Polygon surfaces**

- Polygon surfaces...
  ...can be improved by using these smooth shading techniques, especially when polygons are small with very little change in color or orientation
  ...can be costly to make realistic, sometimes requiring thousands of polygons to approximate a smooth surface
Building Realistic...

*Polygon surfaces*

- Phong Shading...
  ...tends to restore the curvature of the original surface, as approximated by the normals
Building Realistic...

*Polygon surfaces: Phong shading*

- Phong shading...
  - uses the same steps as Gouraud shading but...
  - interpolates the surface normals instead of the intensity
  - so that the intensity for each pixel across the polygon surface is computed
  - of course requires that our objects be represented by a polygon mesh model
Phong shading...

...where at each vertex we compute the surface normal which is defined as the average of the normals of the polygons that surround this vertex.

The rendering process fills with an intensity that is computed for each pixel, where the part of the fill algorithm that actually draws a filled line needs to be replaced.
Now we are ready to compute the intensity of a pixel $P_s$...

...where $P_s$ lies on the scan line from $P_a$ to $P_b$...

...where $P_a$ and $P_b$ are the intersections of the scan line with the polygon edges.

- We must first compute the normals of $P_a$ and $P_b$...

...which are linearly interpolated from the normals of the vertices spanning the edges.
Once the normals of $P_a$ and $P_b$ are computed...

...the normal of $P_s$ can be computed by linear interpolation

$$P_{Ns} = P_{Nb} - (P_{Nb} - P_{Na})(xs - x)/(xs - xa)$$
Since the normal can be performed incrementally for a given scan line instead use...

\[ \Delta PNs = \Delta x/(xb-xa) \times (PNb-PNa) \]

and \( PN_{sn} = PN_{snn-1} + \Delta PNs \)

For efficiency, \( \Delta PNs \) is computed only once for every scan line; it then remains constant, so only one addition needs to be done for every pixel!!
Building Realistic...

Polygon surfaces - summary

- Gouraud shading...
  ...is effective for shading surfaces which reflect light diffusely
  ...can be used with specular reflections, but the shape of the specular highlight must cover vertices to be used
  ...is computationally less expensive than Phong shading
With Gouraud Shading...

...the illumination is evaluated at each vertex in VRC; this must be done before view mapping (which may skew and apply perspective) to preserve the correct angle and distance from each light to the surface.
Building Realistic...

*Polygon surfaces - summary*

- Phong shading...
  
  ...can be used with highlights that are less dependent on the underlying polygons
  
  ...requires calculations to interpolate surface normals and evaluate the intensity for each pixel
  
  ...produces more realistic highlights and greatly reduces the Mach-band effect
Building Realistic...

Polygon surfaces - Phong Shading

- Hidden line & Hidden surface
- Map to Device Coordinates
- Drawing Surface Clip
- Apply Color
- Scan- Converting (including lighting)
- Antialiasing (filter sample)
- Exposure and Dithering
Fast Phong shading...

...surface rendering with Phong shading can be improved by using approximations in the illumination model calculations of the normals

...Fast Phong shading approximates the intensity calculations using a Taylor-series expansion and triangular surface patches
Fast Phong shading...
...where every face has vectors \( A, B, C \) computed from the three vertex equations

\[
N_k = Ax_k + By_k + C \quad k = 1, 2, 3
\]

where, \((x_k, y_k)\) represents a vertex position

This makes sense because Phong shading interpolates normal vectors from vertex normals
Building Realistic...

*Polygon surfaces* - others?

- **Fast Phong** shading...
  ...replaces N in our intensity computations with $Ax+By+C$ for each x,y value
  ...which reduces the phong shading calculations
  ...but it still takes approximately twice as long to render a surface than with Gouraud and in some cases takes 6-7 times longer than Gouraud!
Building Realistic...

*Intensity Attenuation*

- You may have heard about attenuation functions
  - radiant energy from a point light source travels through space, its amplitude is attenuated by the factor $1/d^2$
  - where, $d$ is the distance that the light has traveled
  - this means that a surface close to the light source (a small $d$) receives a higher incident intensity from the source than a distant surface (large $d$)
Therefore, to produce realistic lighting effects

- our illumination model should take this intensity attenuation into account

- otherwise, we are illuminating all surfaces with the same intensity, no matter how far they might be from the light source

- if two parallel surfaces with the same optical parameters overlap, they would be indistinguishable from each other and would be displayed as a single surface!
However, our simple point-source illumination model does not always produce realistic pictures:

- if we use the factor $1/d^2$
- $1/d^2$ produces too much intensity variations when $d$ is small and very little variation when $d$ is large
- the real answer is to use light sources other than point light sources -- as they are too simple to accurately describe real lighting effects
- or, use inverse linear or quadratic functions to $d$ to attenuate the intensities (called attenuation function)
When use use various attenuation functions

- you must set a limit of the magnitude to avoid exceeding the maximum allowable value
- these attenuation values are then multiplied for each diffuse and specular computation for each light source
- usually you want the attenuation values to range between 0 and 1
- a sample attenuation function might be:
- \( 1/(a_o + a_1d + a_2d^2) \) where, \( a_o, a_1, a_2 \) are user controlled
To obtain realistic pictures, we must be able to create textures...

...with small variations in each object's shape and shading

...such as the wood grain of furniture, marks of paint on a chipped or cracked wall, the veins on a leaf, the subtle bluses and texture of human skin

...which are not practically created using polygons
So, to do this, we first create our objects...as previously described...

- ...then we apply texture mapping

Texture mapping...

- ...maps a planar image onto a surface (like a decal or stencil)
- ...the image can be digitized or synthesized
- ...applies a *texture map*, with individual elements of *texels*!
Textures, Shadows, Environment

Texture Mapping

- Textures depend on...
  - ...how the texture is defined and mapped to the object
  - ...the antialiasing techniques, since texture mapping tends to produce worse aliasing since pixels and texels usually do not correspond; one pixel is generally covered by many texels which must be considered to avoid aliasing
Texture mapping...
...is the process of transforming a texture onto the surface of a three-dimensional object
...maps 1D, 2D or 3D textures (2D are the most common)
...samples and filters the texture and sets the corresponding pixel
When textures are mapped to pixels in a non-linear way, the texture domain will have a curvilinear quadrilateral whose shape varies as the pixel position changes!

First, each pixel is mapped to the surface; then the pixel's corners are mapped to the texture's coordinate space.
The value for each pixel is computed by summing all texels that cover the pixel, weighting each by the fraction of the texel that lies within the quadrilateral.
The problem with mapping from some “texture space” to pixel space is that
- the selected texture patch usually does not match up with the pixel boundaries
- which requires calculations of the fractional area of pixel coverage (think about this)
- the previous slide uses “pixel space” TO “texture space” to avoid pixel subdivision calculations and allows antialiasing procedures to be applied
Textures, Shadows, Environment

Texture Mapping

- An effective procedure is to
  - project slightly larger pixel area that includes the centers of neighboring pixels and apply a weighting function to “weight” the intensity values in the texture pattern
  - problems? yes, this method of mapping requires calculation of the inverse viewing projection transformation and the inverse texture map transformation
Textures, Shadows, Environment

**Bump Mapping**

- Bump mapping...
  ...simulates a wrinkled or dimpled surface
  ...affects a surface's shading (the surface stays geometrically smooth) but not the silhouette edges
  ...produces surface perturbations by tricking the reflection model with a jittered (i.e., modified) surface normal
Bump mapping...

...takes advantage of the fact that the effect of wrinkles on the perceived intensity is primarily due to their effect on the direction of the surface normal and therefore on the light reflected

...jitters (i.e., modifies) the surface normal prior to intensity calculations
Bump mapping...
...is usually implemented by a lookup table, such as an array of displacements, which displace a point on a surface a little above or below that point's actual position
- for quick access of bump values with linear interpolation on incremental calculations
- can contain random patterns, regular grid patterns or character shapes
Random patterns
- are useful for modeling irregular surfaces
- such as a raisin

Repeating patterns could be used for an orange

The armor for the stained-glass knight in Young Sherlock Holmes was rendered with bump mapping and texture mapping
- additional color was added for spots of dirt, seems and rivets
Textures, Shadows, Environment

Bump Mapping

- Bump mapping examples:

  - Lengthening and shortening normals creating a bump map
  - The bump map's surface normals

Original Smooth Surface

Surface Normals
Textures, Shadows, Environment

**Frame Mapping**

- Frame mapping...
  - is an extension of bump mapping
  - where we perturb both the surface normal and the local coordinate system
  - we can modify the local coordinate system so that the surface tangent vector lies along the “grain” of a surface and then apply perturbations to this (in addition to bump perturbations) to simulate wood grain patterns, cross-thread patterns in cloth, marble streaks (this time via 2 lookup tables)
Textures, Shadows, Environment

Environment Mapping

- Environment mapping...
  ...is the process of reflecting the surrounding environment in a shiny object
  ...is different than texture mapping since the pattern seen on an object is a function of a view ray
  ...causes a particular detail in the environment to move across the object as the view ray changes
Textures, Shadows, Environment

Environment Mapping

Resulting reflected (on the surface)
view ray = view ray + 2*Normal*\cos \theta
(the angle is between the surface normal and the view ray)
Shadow mapping...

...renders shadows resulting from objects blocking light sources

...takes an object with depth and compares the depth with other overlapping objects to determine if there is a shadow for each pixel

...causes any visible points during "depth rendering" to be illuminated by the light source and any point "behind" an illuminated surface to be in the shadow
Shadow mapping...
...takes into account the fact that shadows vary tremendously as a function of the lighting environment:

- a light source closer to an object will create a larger shadow than a light source further away
- The primary type of shadow algorithms are...
  ...scan-line projection of polygons
  ...shadow Z-buffer ...shadow volumes
Textures, Shadows, Environment

Shadow Mapping

- Scan-line shadow generation
  ...uses the light source as the center of projection
  ...projects the edges of polygons that might cast shadows onto the polygons intersecting the current scan line
  ...modifies the colors of the pixels if the scan crosses one of these shadow edges
Textures, Shadows, Environment

Shadow Mapping

Point Light Source

Polygon A

Polygon B

Viewpoint

Scan line

Shadow
Shadow Z-buffers...

- ...are the simplest approach to computing shadows
- ...can be easily integrated into a Z-buffer rendering system
- ...requires separate shadow Z-buffer for each light
- ...starts by storing rendered depth information in a shadow Z-buffer using the light source as the viewpoint; the depth of the image is computed from the light emitted from the polygons that are visible to the light source
Textures, Shadows, Environment

*Shadow Mapping*

- Shadow Z-buffers...
  - ...then, it renders the scene using a Z-buffer algorithm; but, if the depth is greater than the value stored in the shadow Z-buffer for a pixel, then a surface is nearer to the light source than the point being considered -- it is in the shadow...in this case a shadow intensity is used (otherwise the point is normal)
Extended light sources...

...produce an area of light

...cast soft shadows that contain areas only partially blocked from the source

...create shadows with two parts: umbra and penumbra

The umbra is the part of the shadow that is totally blocked from the light source; the penumbra surrounds the umbra
There is a gradual intensity change from a penumbra to an umbra

For point light sources...all of the light source's shadow creates an umbra
• Umbra and penumbra...depend on the size of the light source
Shadow volumes...

...is the invisible volume of space swept out by the shadow of an object

...is the infinite volume defined by lines emanating from a light source through vertices in the object.

..are created by the intersection of the infinite volume with the view volume.
Advanced Rendering
Shadow Volumes

- Planes defined by the light source and the contour edges of the object define the bounding surface of the shadow volume
- Shadow volumes work for...
  ...point light sources
  ...convex polygonal or polyhedral light sources
Advanced Rendering

Shadow Volumes

Point Light Source

Object

Semi-infinite shadow volume produced by the object

Intersection of the semi-infinite shadow volume with the view volume

View point

View volume
For convex polygonal or polyhedral light sources...

...shadow volumes can be determined using similar approaches

...the shadow volume is defined by each vertex of the object with the light source area

...a penumbra volume is defined to be the smallest convex polyhedron containing the shadow volume and the umbra volume is the intersection of the shadow volumes
Advanced Rendering
Shadow Volumes

- For convex polygonal or polyhedral light sources...
  ...any point lying within an umbra volume is not affected by the light source
  ...any point lying within the penumbra volume is computed by those parts of the light source visible from the point on the object
  ...are best simulated using ray-tracing and radiosity methods
Object: a polyhedron which is casting a shadow

- View point
- Shadow volumes
- Umbra volume
- Penumbra volume
- View volume