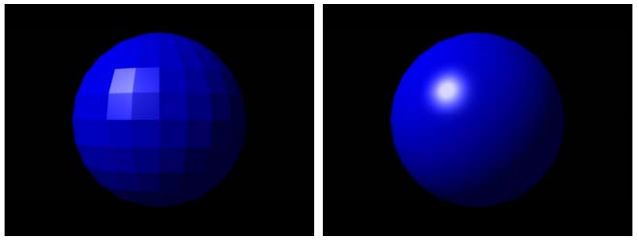
Computer Graphics: Illumination III

Part 2 – Lecture 6

Today's Outline

- Shading Algorithms
  - Flat Shading
  - Gouraud Shading
  - Phong Shading
- Shadows
  - Ground-Plane Projection
  - Shadow Buffer



FLAT SHADING PHONG SHADING

SHADING ALGORITHMS

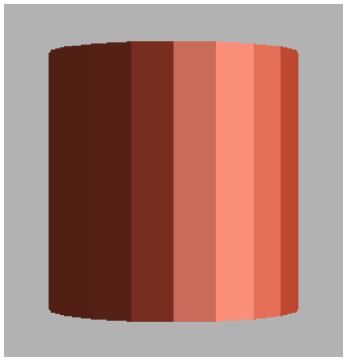
Shading Algorithms

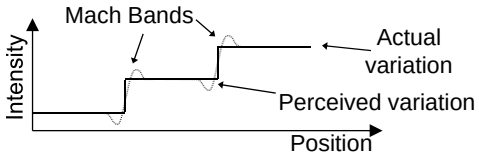
- Phong illumination equation:
  how to calculate color on every point of surface
  (given lights, materials, etc.)
- **Problem**: calculating Phong equations at every single point (pixel) would be extremely slow!
- **Solution**: use a shading algorithm
  - Uses Phong equation only at some points (usually vertices)
  - Then uses interpolation to get colors for in-between points (in-between pixels)
- Three popular shading algorithms:
  - Flat shading (fastest but worst quality)
  - Gouraud shading (balance of speed and quality)
  - Phong shading (slowest but best quality)

Flat Shading (Constant Shading)

- Apply Phong equation once per face (using face normal)
- Shade whole face that color

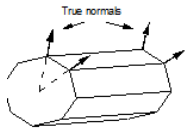
- Advantage: simple and fast
- Disadvantage: very poor display of polygon-mesh approximations to curved surfaces
  - Human eye very sensitive to discontinuities
  - Exaggerates them into *Mach Bands*

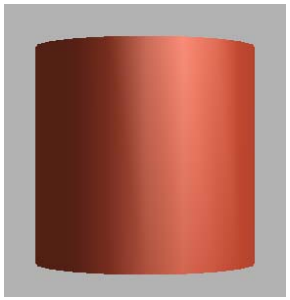


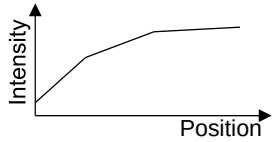


Gouraud Shading

- Apply Phong equation at each vertex (using "true" surface normal)
- Linearly interpolate colors between vertices
- Advantages:
  - Still fast
  - Avoids 0th-order color discontinuities over polygon mesh (color continuous between faces)
- Disadvantages:
  - Still 1st order color discontinuity (→ slight Mach bands)
  - Invariance problem with quadrilaterals
  - Problems with highlights



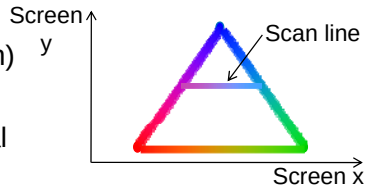




Gouraud Shading Contd.

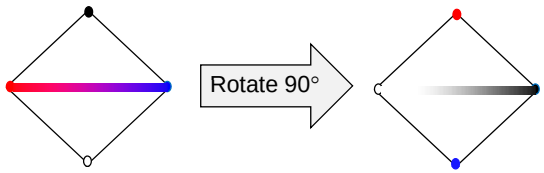
Triangles

1. Get color for each vertex (Phong equation)
2. Interpolate pixel colors between vertices
3. Interpolate pixel colors along all horizontal scan lines



Quadrilaterals

Problem: not rotationally invariant



When rotating the quad, the color of the middle pixel changes (first purple, then gray)

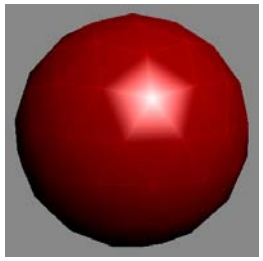
Solution: cut each quadrilateral into two triangles

Gouraud Shading: Highlights

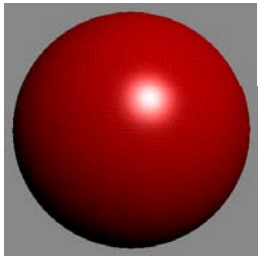
Problem: highlights can only be rendered on a vertex

- Highlight may not be sharp, i.e. gets smeared over adjacent faces
- Highlight may not be visible if not near a vertex

Solution: use more vertices in your mesh



Low number of vertices

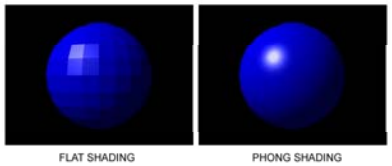


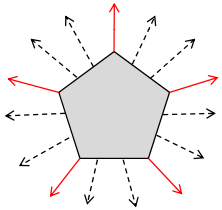
High number of vertices

Images thanks to Zom-B

Phong Shading

- To get crisp specular highlights with Gouraud shading, we need many vertices
- Bui Tuong-Phong suggested Phong shading to solve this
  1. Linearly interpolate the **normal** over the polygon (instead of **color** as in Gouraud shading)
  2. Then evaluate Phong equation at each pixel

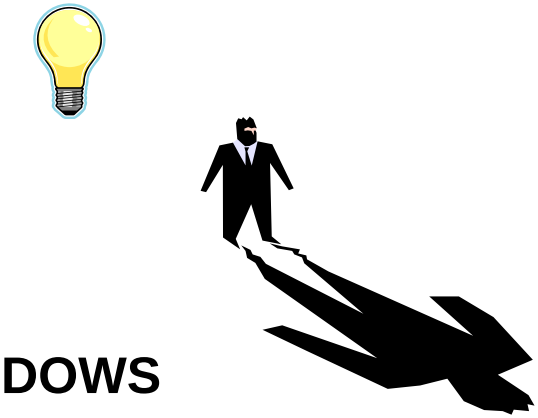




- Advantage: crisp highlights with few vertices
- Disadvantage: slower because Phong calculation for every Pixel

Cost of Shading and OpenGL

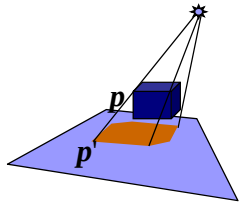
- **Flat shading**: `glShadeModel(GL_FLAT);`
  - Pixel colors constant for entire triangle
  - 1 normal calculation per triangle
  - 1 color calculation per triangle (Phong equation)
- **Gouraud shading**: `glShadeModel(GL_SMOOTH);`
  - 1 normal calculation per vertex
  - 1 color calculation per vertex (Phong equation)
  - 1 color interpolation calculation per pixel
- **Phong shading**: not available in OpenGL
  - 1 normal calculation per vertex
  - 1 normal interpolation between vertex normals per pixel
  - 1 color calculation per pixel (Phong equation)

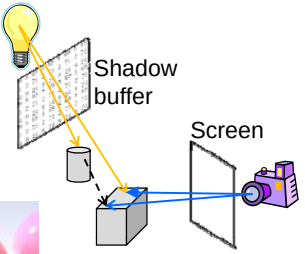


SHADOWS

How to Render Shadows?

- **Where?**
  Points that can be seen but are not illuminated by a particular light source
- **How?** Several possibilities…
  1. **Ground-plane projection**: Draw shadows of objects as separate (flat and dark) objects onto a plane (fast but limited possibilities)
  2. **Shadow buffer**: Efficient way to determine if a visible point is illuminated by a particular light source
  3. **Ray tracing**: trace the path of light rays (slow but high quality)

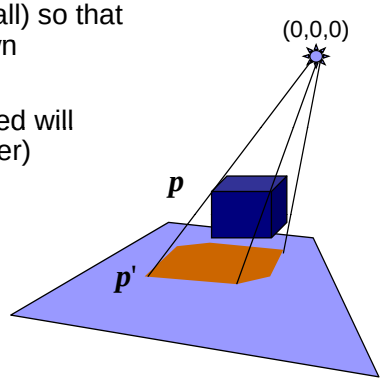




Ground-Plane Projection

Project objects onto plane (e.g. ground, wall) so that they appear as flat shadows when drawn

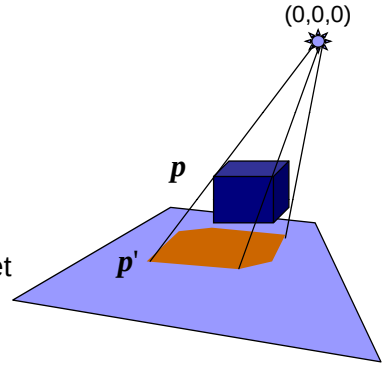
1. Draw the projection plane
2. Turn off depth testing (so pixels rendered will "paint" over those already in frame buffer)
3. Turn off lighting, set shadow color
4. Push MODELVIEW matrix
5. Shift origin to light source position
6. Set plane projection transformation
7. Draw all shadow-casting objects
8. Pop MODELVIEW matrix
9. Turn on lighting and depth testing
10. Draw all other scene objects



Disadvantage:
projects shadows only onto plane, not onto other objects

Plane Projection Transformation

1. Assume light source is at origin
2. Line from light source through $\boldsymbol{p}$ is
   $\boldsymbol{q}(t) = t\,\boldsymbol{p}$
3. Let plane be $ax+by+cz+d = 0$
4. Then at $\boldsymbol{p'}$ have
   $a\,t\,p_x + b\,t\,p_y + c\,t\,p_z + d = 0$
5. Solve for t, calculate $\boldsymbol{q}(t)=\boldsymbol{p'}$ and hence get
   $\boldsymbol{p'} = -d\,(p_x, p_y, p_z) / (a\,p_x + b\,p_y + c\,p_z)$
   $= -d\,\boldsymbol{p} / (a\,p_x + b\,p_y + c\,p_z)$



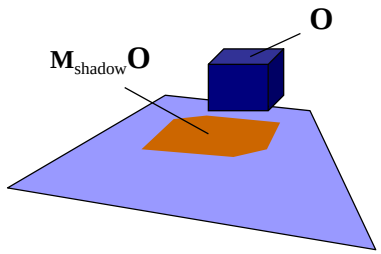
Plane Projection Transformation Contd.

$$\boldsymbol{p'} = -d\,\boldsymbol{p} / (a\,p_x + b\,p_y + c\,p_z)$$

can be written as:

$$\mathbf{p}' = \begin{pmatrix} -d & 0 & 0 & 0 \\ 0 & -d & 0 & 0 \\ 0 & 0 & -d & 0 \\ a & b & c & 0 \end{pmatrix} \mathbf{p}$$

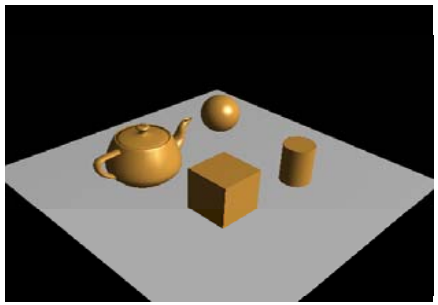
$$= \mathbf{M}_{shadow}\mathbf{p}$$



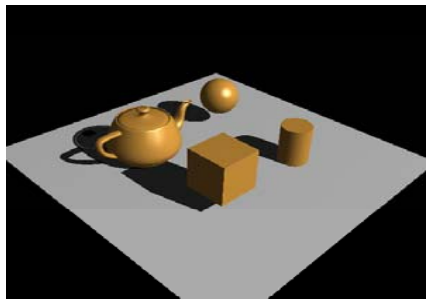
Applying $\mathbf{M}_{shadow}$ to an object yields its planar projection onto the given plane (with center of projection at origin)

Ground-Plane Projection Example

Demo program, LightAndShadows, available in 372 Lecture Notes web page,
http://www.cs.auckland.ac.nz/compsci372s2c/christofLectures/LightAndShadowsNET.zip



No shadows
(can't see "floating" objects)

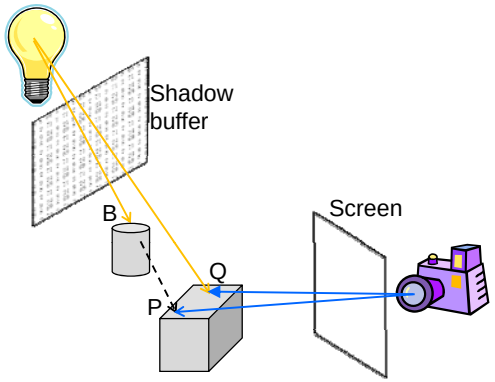


Shadows

Shadow Buffer

Idea: points that are hidden from the light source are in shadow

- Calculate depth buffer from light source position (shadow buffer), i.e. values for distance between light and closest object
- For each screen pixel pointing to a point P:
  1. Get pseudodepth d_P from light source to P
  2. Find element d[i,j] in shadow buffer that points towards P
  3. If d[i,j]< d_P then draw only ambient light (shadow), otherwise full illumination



Advantage:
shadows can be cast from all objects onto all other objects

SUMMARY

Summary

- **Flat shading**: one color calculation per face
- **Gouraud shading**:
  one color calculation per vertex, interpolate over faces
- **Phong shading**:
  interpolate vertex normals and calculate color for every pixel
- Project objects from light sources onto planes to get simple shadow effect
- Use **shadow buffer** to detect covered points for better shadows

References:

- Shading Algorithms: Hill, Chapter 8.3
- Shadows: Hill, Chapter 8.6

Quiz

1. Describe one disadvantage of Flat shading.
2. Why can Gouraud shading render a highlight only on a vertex?
3. What is a shadow buffer?
4. How can we use a shadow buffer to render shadows?