Computer Graphics 1

Chapter 6 (May 27th, 2010, 2-4pm):
Material descriptions - appearance
The 3D rendering pipeline (our version for this class)
Surfaces in nature

• What does a surface do to light? (mini-Brainstorming)
The rendering equation [Kajiya ‘86]

\[ I_o(x, \vec{\omega}) = I_e(x, \vec{\omega}) + \int_{\Omega} f_r(x, \vec{\omega}', \vec{\omega}) I_i(x, \vec{\omega}') (\vec{\omega}' \cdot \vec{n}) d\vec{\omega}' \]

- \( I_o \) = outgoing light
- \( I_e \) = emitted light
- Reflectance Function
- \( I_i \) = incoming light
- angle of incoming light

- Describes all flow of light in a scene in an abstract way
- doesn’t describe some effects:

\[ I_o = I_{amb} + I_{diff} + I_{spec} \]

- strong simplification and specialization of the situation
  - just 1 light source from a clear direction \( l \)
  - viewing direction is given as \( v \)
- only 3 components:
  - ambient component: reflection of ambient light source from and in all directions
  - diffuse component: diffuse reflection of the given light source in all directions
  - specular component: “glossy“ reflection creating specular highlights
Ambient component

- $I_a = \text{Intensity of the ambient light source}$
- independent of any directions
- can simulate a „glowing in the dark“
- can be seen as the equivalent to emitted light $I_e$ in the rendering equation

$$I_{amb} = I_a k_a$$
Diffuse component

- diffuse reflection is equal in all directions
- depends on the angle of incident light
  - light along the surface normal: maximum
  - light perpendicular to the normal: 0

- cosine function describes the energy by which a given area is lit, dep. on angle
  - hence, cosine is used here

- visual equivalent in nature: paper

\[ I_{diff} = I_i k_d \cos \phi = I_i k_d (\vec{l} \cdot \vec{n}) \]
Specular reflection

• let \( r \) be the reflection of \( l \) on the surface

• specular reflection depends on the angle between \( v \) and \( r \)

• \( v = -r \): maximum
• \( v \) and \( r \) perpendicular: minimum

• function \( \cos^n \) behaves correctly
  – exponent \( n \) determines how wide the resulting specular highlight is
  – other functions could be used as well

\[
I_{\text{spec}} = I_i k_s \cos^n \theta = I_i k_s (\vec{r} \cdot \vec{v})^n
\]
Tweaking the parameters

$$I_o = I_{amb} + I_{diff} + I_{spec} = I_a k_a + I_i k_d (\vec{l} \cdot \vec{n}) + I_i k_s (\vec{r} \cdot \vec{v})^n$$

- choose $k_s = 0$ for perfectly matte material
- choose $k_a > 0$ to avoid harsh shadows
- keep $k_a$ small to avoid „glowing“ objects
- add in some $k_s > 0$ to add gloss
- adjust the size of specular highlights with $n$

- all of these calculations generalize to (RGB) color, of course!
The VRML material node

Material {
  exposedField SFFloat  ambientIntensity  0.2  # [0,1]
  exposedField SFColor  diffuseColor         0.8 0.8 0.8 # [0,1]
  exposedField SFColor  emissiveColor      0 0 0          # [0,1]
  exposedField SFFloat   shininess             0.2             # [0,1]
  exposedField SFColor  specularColor      0 0 0          # [0,1]
  exposedField SFFloat   transparency       0                 # [0,1]
}

shininess in VRML is multiplied by 128 to produce $n$ in the lighting model.

Textures and maps

• one of the simplest and oldest ways to achieve good looking objects with simple geometry
• texture design is a very complex task, needs a lot of imagination!
• idea: use a bitmap image, shrink wrap around the object
• use bitmap contents for object surface color: image map
  – can be used for other parameters, e.g., normal, elevation, transparency, reflection
• problem: what does shrink wrap mean exactly?

[Image of world map and globe]
Texture coordinates and UV mapping

- each texture is mapped to a 1x1 square
- each object defines u,v coordinates
  - such that u,v are both between 0 and 1

- straightforward for geometric primitives
  - different possibilities
  - conventions exist

- not so easy for polygon models
  - can be defined per vertex
  - ...but who wants to do this?
  - simplifications: shrink a sphere onto the object
    - works fine with convex objects
    - always tricky for complicated objects
Texture filtering

• During rasterization, for each rendered pixel of the textured object we need to look up a color value from the texture
  – will almost always fall between texture pixels (texels)
  – texture may have too much resolution: sampling or integration
  – texture may have too little resolution: interpolation

• naive approach: pick the nearest neighbor pixel
  – leads to blocky textures

• better approach: bilinear filtering
  – pick the 4 neighboring pixels and linearly interpolate
  – eliminates excessive integration over pixels

• Mip map: image pyramid with image scaled to 1/4 area in each step

• trilinear filtering: find the 2 best levels of the mip map and interpolate between them

http://wiki.aqsis.org/dev/texture_filtering
Bump Mapping

• texture file is only greyscale
• grey value determines the elevation of the surface
  – e.g., black = dent, white = bulge

• can simulate complex 3D surface structure on very simple geometry
• often used together with image maps to enhance realism
• only modifies surface color, not silhouette!

• introduced by Jim Blinn in 1978
  – related and improved techniques with similar look in use today: normal mapping, displacement mapping

http://en.wikipedia.org/wiki/Bump_mapping
Environment maps

• maps show the environment of the object
  – inside out view, 360 degrees in all directions
  – can be represented as 6 sides of a cube
  – can be photographed in a real environment

• can be used to calculate appropriate reflections
  – problem: ________________________________

• can also be used for lighting
  – record map in real environment
  – light a 3D model with it
  – this model will seem as if lit in the real environment
  – useful for combining real and virtual objects
Procedural surface descriptions

• programming languages for surface descriptions
• can influence various stages of the rendering pipeline
  – in particular: can implement textures and the phong model
  – but also much more...
• can describe real 3D structures
  – not just surface color
• state of the art in high end 3D graphics
  – e.g., RenderMan, used in PIXAR movies
  – also in OpenGL, DirectX
• detailed implementation varies depending on the platform
• in OpenGL: vertex shaders and fragment shaders
  – fragments = parts of an object that cover 1 screen pixel
OpenGL: Vertex and Fragment shaders

• A vertex shader can do the following:
  – transform the vertex position using the modelview and projection matrices
  – transform normals, and if required normalize them
  – generate and transform texture coordinates
  – lighting per vertex or compute values for lighting per pixel
  – color computation

• A fragment shader can do the following:
  – compute colors, and texture coordinates per pixel
  – apply a texture
  – fog computation
  – compute normals if you want lighting per pixel

• This, and more details at: http://www.lighthouse3d.com/opengl/gls/
Links, various

• You’ve been doing 3DCG too long if ...when people ask you, "What's up?", you reply "Y": http://www.deakin.edu.au/~agoodman/scc308/toolong.html

• Detailed class material from one of the world‘s leading groups: http://graphics.stanford.edu/courses/

• Compact overviews from the wisdom of the masses ;-) : http://en.wikipedia.org/wiki/Rendering_(computer_graphics)


• Some nice tutorials related to this class: http://www.lighthouse3d.com/