1996-12-03 Bo Nordin (bosse@cb.uu.se)

Lecture notes, computer graphics (F16) (12.4, 14.1 14.2, 13.4, 14.7, 14.9)

Rendering shaded images (12.4)

VSD - yesterday
Illumination & shading
Interpolated shading (Gouraud, Phong)
Material properties (dull vs. shiny) & texture
Curved surfaces
Multiple lights, bouncing rays, shadows etc.
Transparency (refraction), reflection
Improved camera models (not everything in focus!, motion blur)

Illumination and shading models:
more complex than simply calculating whether a pixel lies along a projector and whether something lies between the viewer and the projection plane

Illumination (14.1)

Illumination models tell us what determines a surface's colour at a point
Shading model broader
Models approximate without using "real physical" models
first an individual point on a surface and how light affects it directly.

Ambient light
the model used so far (objects displayed with an intensity intrinsic to it)
(unrealistic world of nonreflective, self-luminous objects) $I = k_i$ where k_i is the object's intensity
Diffuse non-directional light: $I_{amb} = I_a k_a$ (k_i is the object's intrinsic intensity)
I_a is the intensity of the ambient light (constant for all objects)
k_a is the ambient-reflection coefficient (0-1) is a *material property*. (colour plate 27)
No counterpart in reality, accounts for light not otherwise accounted for, produces no shadows.

Diffuse reflection:
point light source whose rays emanate uniformly in all directions
the object's brightness varies (depending on direction and distance)
Dull, matte surfaces (chalk) show diffuse reflection (Lambertian reflection)
Equally bright from all viewing angles (reflection equal in all directions)
Brightness depends only on the angle θ between the direction L to the light source & the surface normal N.
2 factors cancel out: beam covers are inv. prop to cosθ. Seen by viewer: prop to sam.
$I_{diff} = I_p k_d \cos\theta$ (or if vectors are normalised: cosθ is $\mathbf{N} * \mathbf{L}$)
I_p is the intensity of the point light (diffuse-reflection coefficient k_d) θ in [0,90] degrees

Assume self-occluding objects
If a light source is far away it is called a directional light source
We often add an ambient light (figures pg. 481)

Light-source attenuation:
$I = I_a k_a + f_{att} I_p k_d (\mathbf{N} * \mathbf{L})$ where f_{att} can be $1/d^2_L$ (falls off as the inverse square of the dist.)

Was also used to simulate atmospheric attenuation!
In practice, this does not work very well: no variation far away, too much variation close:
use $f_{att} = 1/(d^2_L + d_0)$, for instance
f_{att} can be specified by the user instead: $\min(1/(c_1+c^2 d_L+c_3 d^2_L),1)$

Can we use a 3-comp colour model: (not really)
Coloured light: $I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{att} I_{p\lambda} k_d O_{d\lambda} (N*L)$ $O_{d\lambda}$ is the diffuse colour (evaluated at discrete intervals)

Atmospheric attenuation:
depth cueing - more distant objects are rendered with lower intensity than are closer ones
In PHIGS you can specify two depth cueing planes with associated scale factors s_b & s_f.
(blends intensity with depth-cue colour)
$I'_\lambda = s_o I_\lambda + (1-s_0) I_{dc\lambda}$

Specular reflection:
Can be observed on any shiny surface (pg. 485)
(try a ball, for instance)
the highlight moves as you move your head
a mirror is a perfect reflector (reflects light in only one direction) (draw fig, page 485)
Model assumes that maximum specular reflectance occurs when α is zero and then falls off very quickly as α increases:
Simplified model: $I_{spec} = I_p * K_r * f(\theta) * \cos^k \alpha$ (high k -> a mirror) (also book 485)
f() is a function to distinguish between different materials: glass, plastic, metal, paper, etc.
K_r is the specular reflection coefficient (0-1) - light may be absorbed by the material
λ in the formulae is the wavelength of the light
$\cos\alpha$ <--> R*V if vectors are normalised
Multiple light sources:
Just sum them

Summary, lightning:
$I = I_{amb} + I_{diff} + I_{spec}$
(light is attenuated $1/d^2$ but this is often ignored or simplified - and the results looks ok)
Several colours: separate for each colour

Shadows (14.4)

Shadow algorithms determine which surfaces can be seen from the light source.
Visibility is dichotomous: 0 or 1 - only ambient light remains if 0 (i.e. if no light falls):
$I = I_{amb} + \Sigma S_i (I_{i,diff} + I_{i,spec})$ (S is 0 or 1)

Scan-line algorithms with shadow processing exist (light source as COP)

Transparency (14.5)

Surfaces that transmit light can be *transparent* or *translucent*.
Transparent surfaces produce both reflected and transmitted light.
A model that ignores refraction (the fact that the direction of the light path may change as light passes through an object) is *interpolated transparency*:
$I = (1-k_{t1}) I_1 + k_{t1} I_2$ where index 1 refers to a transparent object in front of opaque object 2

$(1-k_{t1})$ is called the *opacity factor*,
Filtered transparency is another approach:
$I = I_1 + k_{t1}OtI_2$ where O_t is the transparency color. (treats surface 1 as a filter).

Scan-line VSD and LPA can be generalized to include transparency
z-buffer algorithms often uses *screen-door transparency*.

Taking refractive transparency into account is much more difficult.

Shading (14.2)
A surface can be shaded by calculating the surface normal at each visible point and apply the illumination model at that point. Brute-force - very expensive. More efficient models exist for surfaces defined by polygons and polygon meshes.

The simplest model is *constant* (*faceted*, *flat*) shading. An illumination model is applied once. We sample the illumination equation once for every polygon. OK if
- the light source is at infinity, so N*L is constant across the face
- the viewer is at infinity, so N*V is constant -"-
- the polygon is not an approximation

Interpolated shading is the opposite (info interpolated from values at vertices)
If faces in a mesh are shaded individually we get a faceted appearance (colour plate 28)
Finer mesh causes Mach band effects (our eyes...) at discontinuities in magnitude or slope of intensity

Gouraud shading:
(pg 494) a.k.a. intensity (colour) interpolation shading (colour plate 29)
uses the normal of each vertex (either available or approximated as the average of the normals of the polygons sharing each vertex).
Then find vertex intensities by using the vertex normals with an illumination model
Finally, each polygon is shaded by linear interpolation along each edge and between edges along each scan line

Phong shading:
(pg 495) a.k.a normal-vector interpolation shading (colour plate 31) interpolates the normal rather
than the intensity. Normals are interpolated along edges from vertices and along scan lines across polygons. NOTE: normals must be normalised

Problems, shading

silhouttes
perspective distortion
orientation dependence
problems at shared vertices
unrepresentative vertex normals

Surface detail

Texttur mapping: Mapping an image onto a surface
Bump mapping: surface normal displacement (approximates roughness)
Displacement mapping: displacement of the surface

Ray tracing (13.4, 14.7)

More complicated strategies:

Reflection, refraction and shadows must often be integrated with VSD - hidden lines aren't necessarily hidden anymore. *Global illumination models*. Example ray tracing.

RT prototypical image precision algorithm

(pg 459) (ray casting sometimes only the VSD part)

Select COP (viewer) and a window, fire rays from COP through the grid.
Use parametric $x = x_0 + t(x_1-x_0)$ and Δx
Simple for spheres (pg. 460) quadratic eq: no real root -> no inter, one -> touch, two -> ok
Surface normal simple for spheres!
Harder to find intersection with polygons

Efficiency: Use projections for polygons
Bounding volume tests
75-95 % time spent in intersection calculations
Hierarchy, spatial partitioning

Shadows et al:
for shadows: fire additional ray from the p. of intersection to the light source
if an object is found: shadow
also spawns array for reflection and refraction (secondary rays) (figure pg 511)
and recursion... ray tree
stop condition: no object intersected, a user-defined max depth, out-of-memory

Rendering pipeline (14.9)
z buffer & Gouraud (important to transform normals correctly) (inv of matrix, 183)
if simple transformations: ordinary matrix
db traversal -> modelling -> trivial acc/rej -> lighting -> viewing -> clipping ->
divide by W, map -> scan conversion (rasterization) -> display

trivial: out of window/ backface culling

if Phong the lighting must be moved to the scan conversion

if listpriority alg: we need a different pipeline (pg. 523)

db traversal -> modelling -> preliminary VSD -> new db -> db traversal -> trivial acc/rej->
viewing trans. -> clipping -> divide by W, map -> rasterization (lighting) -> display

Ray tracing is simpler: db traversal -> modelling -> ray tracing -> display