CMPS 415/515, Fall 2018 presentations

Notes:

1) *Do not redistribute without permission* except according to "fair use" standards. Assume content is copyrighted, and that distributing more than small excerpts exceeds fair use. (See policies in the syllabus, UL's Copyright Handbook, and the publishers' licenses for instructors).

2) The lectures are mostly whiteboard-based, rather than slide-based, so content will not match exactly.

Note

[*HB*] identifies figures from *Computer Graphics with OpenGL: Third Edition*, by Hearn and Baker, ©2004 Pearson Education

[*FVD*] identifies figures from *Computer Graphics: Principles and Practice: 2nd Edition*, by Foley, van Dam, Feiner, and Hughes, ©1996 Addison-Wesley

[*Red*] identifies figures from *OpenGL Programming Guide*, by Shreiner et al., ©1994-2013, Addison-Wesley ("Redbook")

[*RTR*] identifies figures from *Real-Time Rendering, 2nd Edition,* by Tomas Akenine-Möller and Eric Haines, ©2002, A. K. Peters

Modeling Transform Review:

- Know the basic homogeneous transforms for 3D: Translation, scale, and at least one each of the rotations and shears
- Know why the homogeneous form is useful
- Know how to rotate about an arbitrary point
- Know basic properties, e.g., rotation matrix properties, when do transforms commute, how do we invert them
- Representations of orientation:
  - Understand the main concept of each method
  - Know their relative strengths and weaknesses
  - Understand the diagrams illustrating weaknesses
- Know how to use transforms and scene graphs as done in your programming assignments

3D viewing

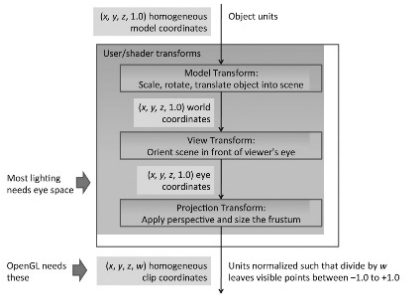


Figure 5.3 User coordinate systems unseen by OpenGL

[*Red, 8th Edition*]

View transform

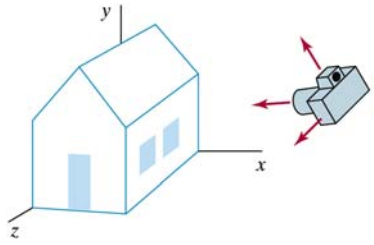


Figure 7-10

Photographing a scene involves selection of the camera position and orientation.

[*HB*]

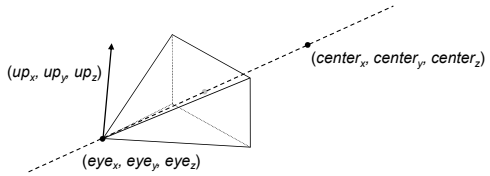
View transform

- As we saw previously, the view transform is the inverse of a camera pose matrix
  - View transform converts world-relative coordinates into eye-relative coordinates (direct application of camera pose matrix would convert from eye coordinates to world)
  - Real-world example (analogy): to create the impression of a camera moved left by 1 unit from an origin, we could move all other objects right by 1 unit with a camera fixed at the origin.

View transform

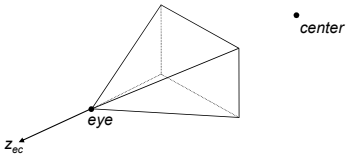
- View transform is sometimes set from camera descriptions with parameters like:
  - Eye coordinate (projection reference point, center of projection)
    - The position of the eye (camera, viewer)
  - Center coordinate (lookat point, target point)
    - A point for aiming the eye
  - Up vector
    - Projected into the eye's x-y plane to define eye's "up" direction (to simplify use, up is not assumed to actually be perpendicular to view direction -z)

Parameters eye, center, and up



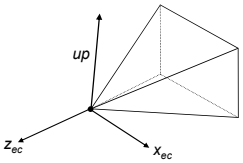
(all expressed with respect to world)

Eye system's z-axis



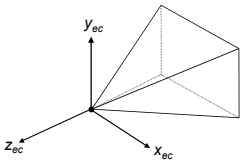
Eye coordinate system has origin at *eye* and has z-axis:
$z_{ec} = (eye - center)$, normalized

Eye system's x-axis



Then eye coordinate's x-axis is:
$x_{ec} = (up \times z_{ec})$, normalized

Eye system's y-axis



And its y-axis is:
$y_{ec} = (z_{ec} \times x_{ec})$,
which is already unit length if z_{ec} and x_{ec} were normalized

View Matrix is world-wrt-eye

$$ {}^{eye}_{world}T = \begin{bmatrix} \text{—} & x_{ec} & \text{—} & 0 \\ \text{—} & y_{ec} & \text{—} & 0 \\ \text{—} & z_{ec} & \text{—} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

This is inverse of eye-wrt-world. Recall how to invert a rigid body transform.

Rotation matrix rows are the eye system axes described w.r.t. the world orientation. (Rows instead of columns due to inversion, done by transpose.)

Note

- A view transform is separate from a projection matrix (described next).
- It is typically combined with model matrix to form modelview.

  (you may actually get correct geometry with a view transform combined with projection matrix, but this does not follow convention and tends to mess up lighting when calculation should be done in eye space)

3D viewing

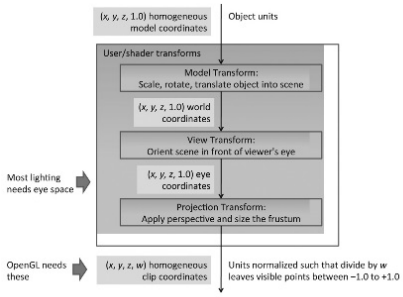
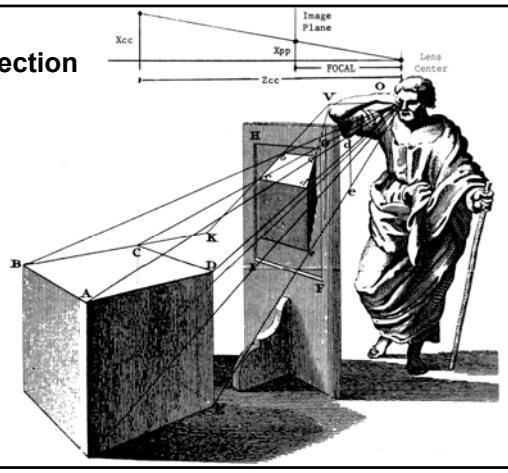
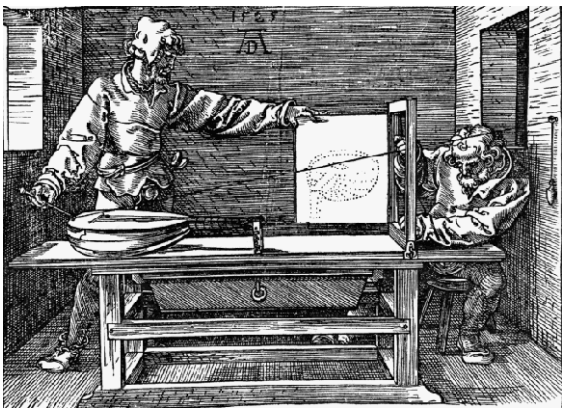


Figure 5.3 User coordinate systems unseen by OpenGL

[*Red, 8th Edition*]

Projection





Perspective projection mechanism, Albrecht Dürer, 1525

Perspective device at Virginia Museum of Fine Arts.
Slide from Stanford Institute for Reading and Learning.

Drawing by Julian Beever, using a sidewalk as the projection plane

Projection to a plane

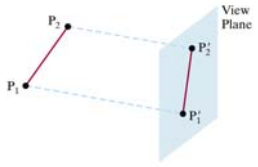


Figure 7-22

Parallel projection

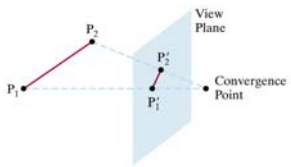


Figure 7-23

Perspective projection

[*HB*]

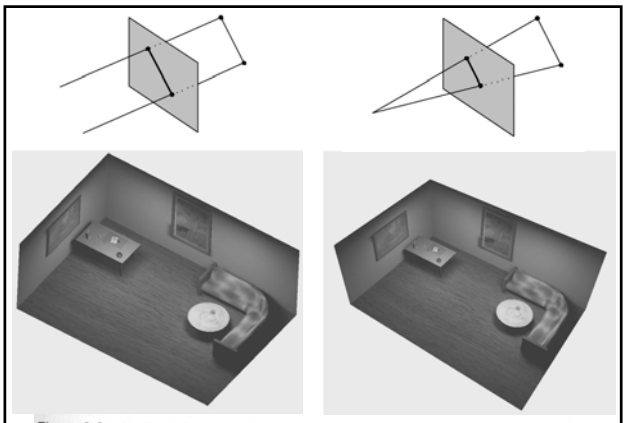


Figure 2.6. On the left is ~~an orthographic,~~ or parallel, projection; on the right is a perspective projection.

[*RTR*]

Perspective projections

- Perspective foreshortening
- Vanishing points and axis vanishing points

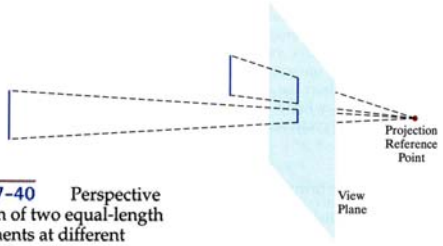


FIGURE 7-40 Perspective projection of two equal-length line segments at different distances from the view plane.

[*HB*]

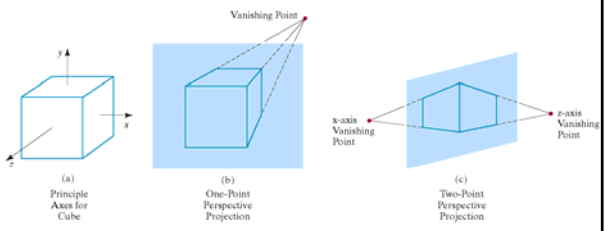


Figure 7-44

Principal vanishing points for perspective-projection views of a cube. When the cube in (a) is projected to a view plane that intersects only the z axis, a single vanishing point in the z direction (b) is generated. When the cube is projected to a view plane that intersects both the z and x axes, two vanishing points (c) are produced.

[*HB*]

Parallel projections

- No perspective foreshortening
- Orthographic (orthogonal): direction of projection (DOP) normal to projection plane
  - top-, front-, side-elevation
  - axonometric
    - E.g., isometric: DOP makes equal angles with axes
- Oblique: DOP not normal to projection plane
  - e.g., cavalier, cabinet

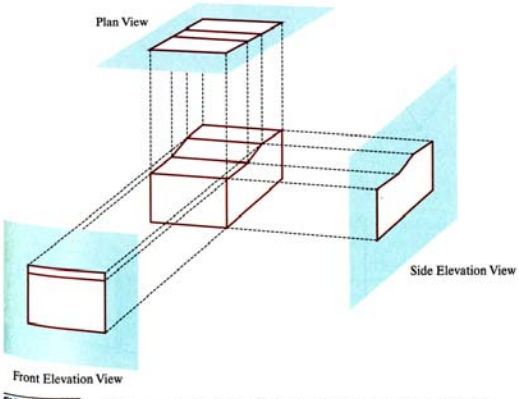


FIGURE 7-24 Orthogonal projections of an object, displaying plan and elevation views.

[*HB*]

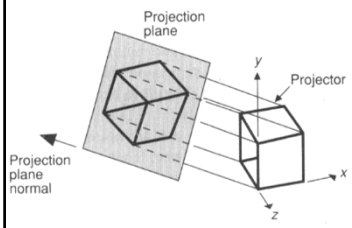


Fig. 6.7 Construction of an isometric projection of a unit cube. (Adapted from [CARL78], Association for Computing Machinery, Inc.; used by permission.)

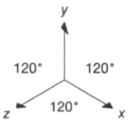


Fig. 6.8 Isometric projection of unit vectors, with direction of projection (1, 1, 1).

[Orthographic projection with direction parallel to a cube diagonal]

[*FVD*]

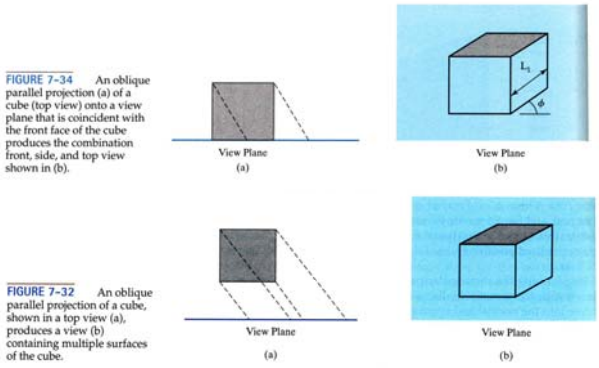


FIGURE 7–34 An oblique parallel projection (a) of a cube (top view) onto a view plane that is coincident with the front face of the cube produces the combination front, side, and top view shown in (b).

FIGURE 7–32 An oblique parallel projection of a cube, shown in a top view (a), produces a view (b) containing multiple surfaces of the cube.

[Oblique parallel projection: direction of projection is not normal to projection plane]

[*HB*]

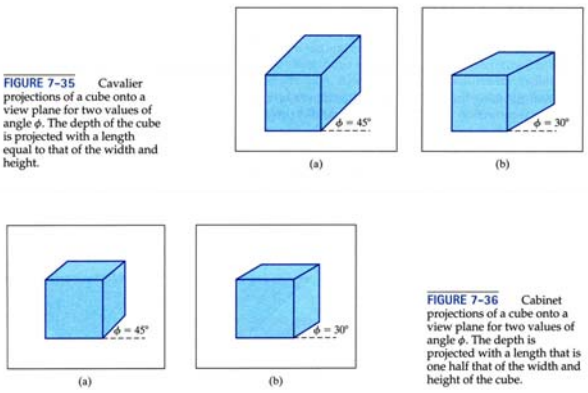
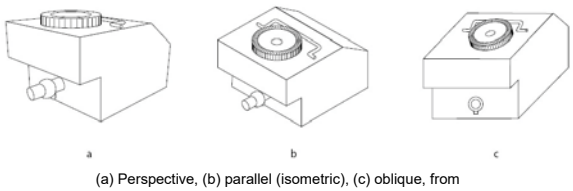


FIGURE 7–35 Cavalier projections of a cube onto a view plane for two values of angle ϕ. The depth of the cube is projected with a length equal to that of the width and height.

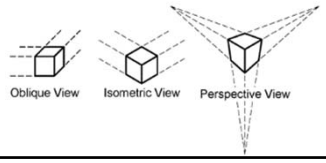
FIGURE 7–36 Cabinet projections of a cube onto a view plane for two values of angle ϕ. The depth is projected with a length that is one half that of the width and height of the cube.

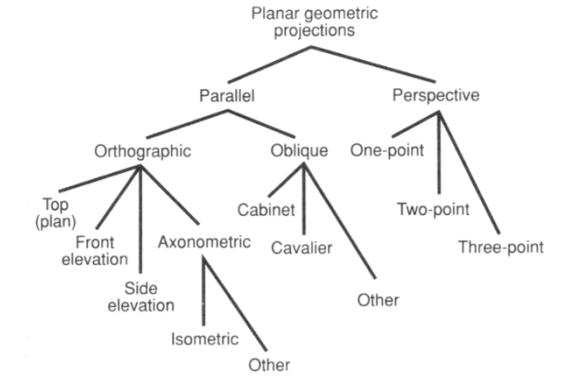
[Two types of oblique projection]

[*HB*]



(a) Perspective, (b) parallel (isometric), (c) oblique, from *Aviation Maintenance Technician Handbook*, faa.gov





[*FVD*]

(*Note: [HB] also uses "oblique" for a type of perspective projection*)

Projection matrices: a simple example

Assume orthographic projection with DOP in *z* direction, projection plane is $z = 0$.

Question:

What are the projected coords of a point (x,y,z)?

$x_p =$

$y_p =$

$z_p =$

The projection matrix for this example is:

$$M_{ort} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A system like OpenGL converts the z-value to a depth value to be passed to visible surface determination (discussed later), and the matrix becomes:

$$M_{ort} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

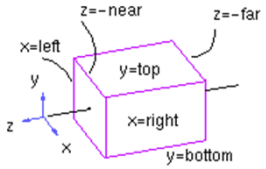
(*As an exception to our right-handed conventions, the post-projection x-y-depth frame will be left-handed*)

Changing DOP in orthographic projection

- What if we want a different direction of projection for the orthographic case?
  - Orthographic projection is always along the camera model's z_{ec}, and we can already change the view transform to set up the camera with z_{ec} pointing in any direction
  - If we mean a d.o.p. that is not perpendicular to the view plane, it would be an oblique projection instead of orthographic

Changing the orthographic view volume

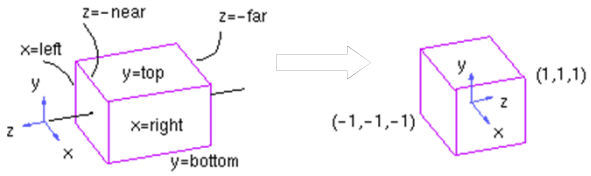
- View volume: the portion of 3D space to be viewed
- For orthographic viewing, the view volume can be described by 6 numbers specifying the range *in eye coordinates*



[picture from U. of Toronto CSC 418 notes]

Canonical view volume

- To simplify and optimize later operations, all view volumes are transformed into a *canonical (or normalized) view volume*



(*again note left-handed frame after projection*)

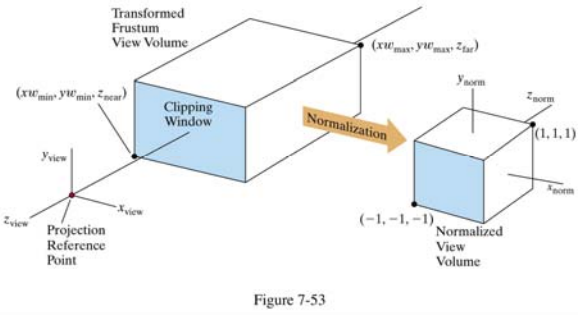


Figure 7-53

[*HB*]

Normalizing transform, orthographic case

- Projection matrix includes a *normalizing transform* to transform the desired view volume to the canonical view volume.
- In OpenGL:
  - (left, bottom, -near) is mapped to (-1,-1,1) (*before* z-flip for left-handed depth space)
  - (right, top, -far) is mapped to (1,1,-1)
  - This requires: a translation of the v.v. center to the origin, followed by a scale to the final size
  - The basic orthographic matrix with z-flip will be applied after this

Extended orthographic projection matrix

- So, the complete orthographic matrix is:

$$M_{ort,norm} =$$

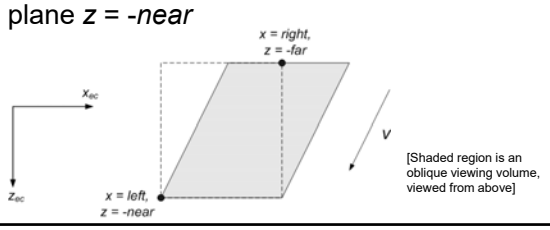
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2/(r-l) & 0 & 0 & 0 \\ 0 & 2/(t-b) & 0 & 0 \\ 0 & 0 & 2/(f-n) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -(l+r)/2 \\ 0 & 1 & 0 & -(b+t)/2 \\ 0 & 0 & 1 & (n+f)/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where l = left, r = right, b = bottom, t = top, n = near, f = far

You can find this result in the red book's appendix (7th or older editions), expressed as a single matrix

Oblique projection (not detailed in class)

- An oblique direction of projection, *v,* can be implemented by adding a shear matrix
- Assume *v* is given in eye coordinates, and coordinates should be unchanged at the plane *z = -near*

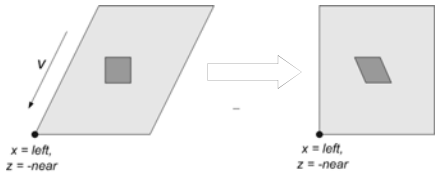


[Shaded region is an oblique viewing volume, viewed from above]

Oblique projection matrix

Add normalizing steps for the oblique viewing volume shape:

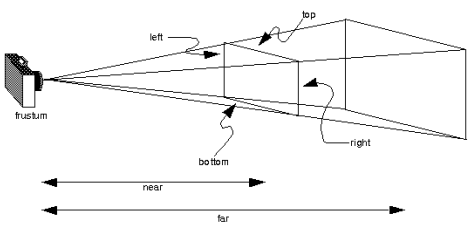
1) Translate the near plane to z = 0
2) Apply shear (so view volume becomes box)
3) Translate back



- So the oblique matrix is:

$$M_{oblique,norm} = M_{ort,norm} \bullet T(-near) \bullet SH(v) \bullet T(near) =$$

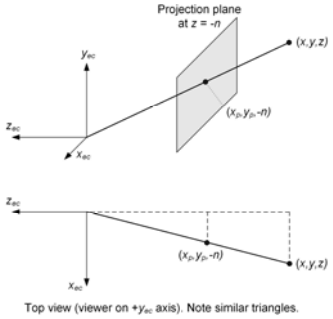
$$M_{ort,norm} \bullet \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -near \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -v_x/v_z & 0 \\ 0 & 1 & -v_y/v_z & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & near \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



[*Red*]

An example perspective projection matrix

Eye at the origin, projection to plane $z = -n$



Using similar triangles:

$$\frac{x_p}{-n} = \frac{x}{z}, \qquad \frac{y_p}{-n} = \frac{y}{z}.$$

$$x_p = \frac{-n \cdot x}{z}, \quad y_p = \frac{-n \cdot y}{z}, \quad (and\ z_p = \frac{-n \cdot z}{z} = -n).$$

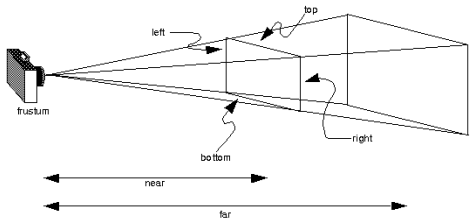
How do we get the division (by z) with a homogeneous transform?

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = M_{per} \bullet p = \begin{bmatrix} -n & 0 & 0 & 0 \\ 0 & -n & 0 & 0 \\ 0 & 0 & -n & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

This gives $W = z$. Recall any nonzero multiple of a homogeneous coordinate represents the same 3D point. Homogenize (perspective division):

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} \text{ represents same 3D point as } \begin{bmatrix} X/W \\ Y/W \\ Z/W \\ 1 \end{bmatrix} = \begin{bmatrix} -nx/z \\ -ny/z \\ -n \\ 1 \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}.$$

- Before we generalize this, consider the perspective view volume (frustum) defined by 6 numbers, as illustrated



z from –far (-f) to –near (-n), x from left (l) to right (r) at the near boundary, and y from bottom (b) to top (t) at the near boundary, w.r.t. eye coordinate system [*Red*]

Generalized M_{per}

$$M_{per} = \begin{bmatrix} -n & 0 & 0 & 0 \\ 0 & -n & 0 & 0 \\ 0 & 0 & -(n+f) & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- This preserves some depth info (Z-coordinate)
- Z-coordinate is unchanged at near and far boundaries, and a depth-mapping-related adjustment is done for values in between
- It transforms the frustum into an orthographic view volume (distant portions get smaller)

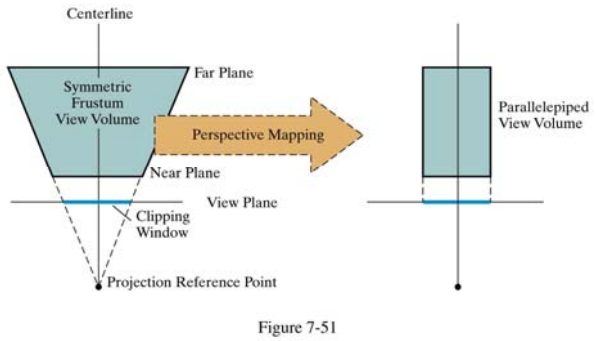


Figure 7-51

[*HB*: Ignore the details of this diagram; just get the main point that it shows the perspective foreshortening from the division]

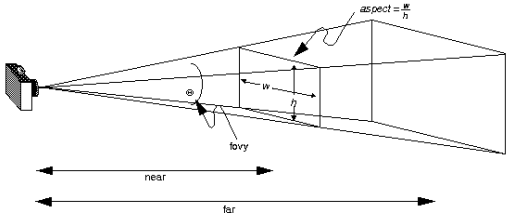
- Finally, the orthographic normalization and projection can be applied:

$$M_{per,norm} = M_{ort,norm} \bullet M_{per}$$

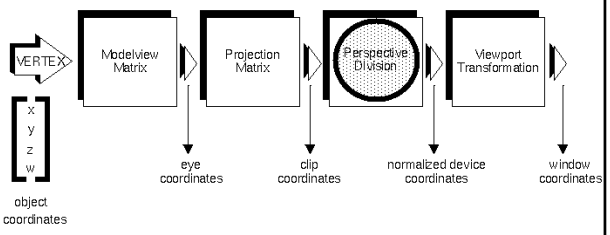
(Look up glFrustum documentation to see the final result. All terms may appear negated: this results in the same 3D point since it is just multiplying a homogeneous transform by a constant)

Field of view (aperture) model

- Camera sometimes described with just 4 parameters: field of view, aspect, near, far
- Doesn't allow off-center eye (3D or head-tracked views)



[*Red*]



[*Red*]

Clipping to view volume

Diagram shows extension of Cohen-Sutherland clipper to 3D.
Clipping can also be done in homogeneous space before division.
Clipping is optimized for canonical view volume.

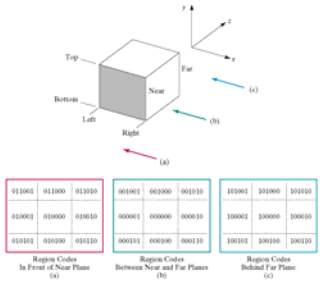


Figure 7-57

Viewport transformation

- After a perspective division step, a viewport transform converts coordinates from the normalized values (range [-1,1]) to final device coordinates
- Similar to the window-to-viewport transform that we already discussed for 2D
- glViewport(*x, y, width, height*)

Illumination (lighting, shading)

$$\mathbf{i}_{tot} = \mathbf{i}_{amb} + \mathbf{i}_{diff} + \mathbf{i}_{spec}$$

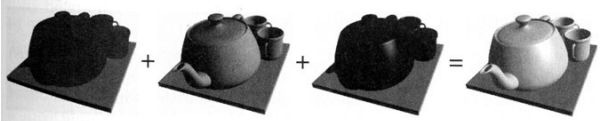


Figure 4.13. The basic lighting equation is illustrated for a teapot by adding (from the left) the ambient, the diffuse, and the specular components. The resulting lighting is shown at the right. *(Tea cup model is reused courtesy of Joachim Helenklaken.)*

[*RTR*]

Illumination (lighting, shading)

- Determine color of surfaces based on properties of lights and surfaces
- Simplicity vs. physical accuracy
- The basic standard model is the Phong illumination model, developed in 1973

A very simple lighting equation

- Simplest model: each object is assigned some color (O_R, O_G, O_B)

Lighting equation is $I = O$

$I = O$ is shorthand for:

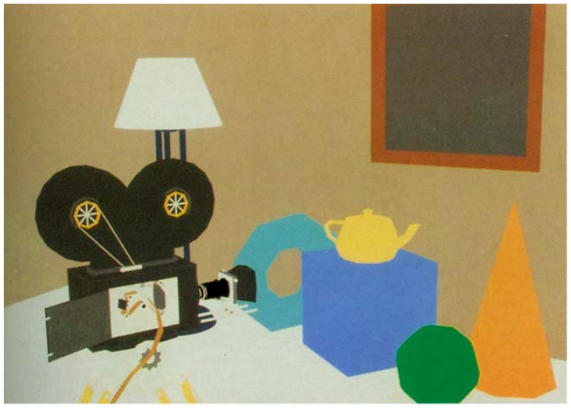
$$I_R = O_R \text{ , } I_G = O_G \text{ , } I_B = O_B$$

(Intensities (I) and colors (O) are assumed to have multiple channels)

Ambient light

- Assume ambient light impinges equally on all surfaces from all directions
- Illumination equation: $I = k_a I_a O$

I_a: intensity of ambient light in scene
(same for all objects)
k_a: object's ambient-reflection coefficient



Pixar's "Shutterbug" (1990), with ambient-only lighting

Diffuse (Lambertian) reflection

- Assume light arrives at surface from some direction (e.g., direction to a point light source)
- Lambert's law: reflected light intensity in any direction from a small fixed-size surface area is proportional to the cosine of the angle between that direction and the surface normal
- The area's projected size is proportional to the cosine of the angle between the normal and the direction to viewer, so reflected light concentrates to a smaller viewed area as this angle increase
- These two cancel in the sense that the area appears equally bright from all viewing angles

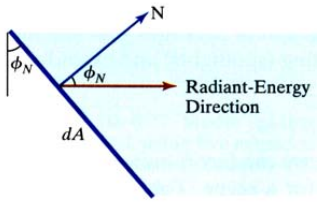


FIGURE 10-10 Radiant energy from a surface area element dA in direction ϕ_N relative to the surface normal direction is proportional to $\cos \phi_N$.

[*HB*]

Diffuse (Lambertian) reflection

- Although brightness does not depend on viewing angle, it does depend on angle between surface normal and direction to light (from surface)

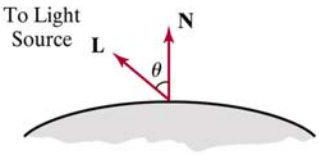


Figure 10-13
Angle of incidence θ between the unit light-source direction vector L and the unit normal vector N at a surface position.

[*HB*]

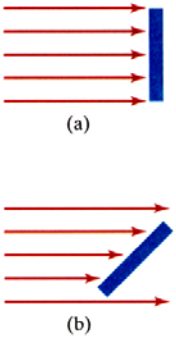


FIGURE 10–11 A surface that is perpendicular to the direction of the incident light (a) is more illuminated than an equal-sized surface at an oblique angle (b) to the incoming light direction.

[*HB*]

Diffuse illumination equation

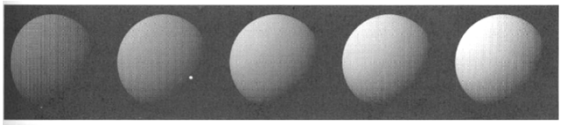
$$I = k_d I_l O_d \cos\theta = k_d I_l O_d (N \cdot L)$$

I_l : light source intensity
k_d : object's diffuse-reflection coefficient
O_d: diffuse color term
N, L : normalized vectors as illustrated
(where do they come from?)

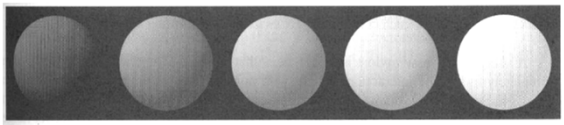
Combined with ambient light:

$$I = k_a I_a O_d + k_d I_l O_d (N \cdot L)$$

(assumes diffuse and ambient object color is the same)

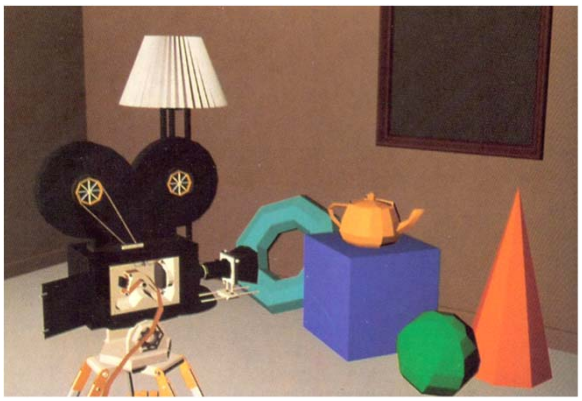


diffuse component, increasing k_d



diffuse + ambient, increasing k_a

[*FVD*]



Ambient + diffuse lighting

Specular reflection

- Sharp highlights found on shiny surfaces
- Typically, highlight color is similar to incident light, not object diffuse color
- View-dependent

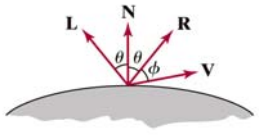
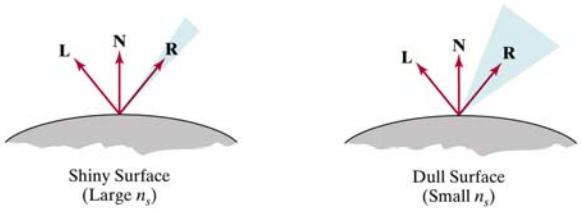


Figure 10-16
Specular reflection angle equals angle of incidence θ.

[*HB*]

Specular exponent n (called n_s in diagrams)



Shiny Surface (Large n_s) | Dull Surface (Small n_s)

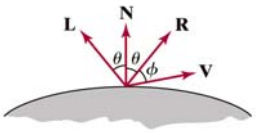
Figure 10-17

Modeling specular reflections (shaded area) with parameter n_s.

[*HB*]

Specular exponent *n*

Dependence on angle ϕ, modeled by $\cos^n(\phi)$



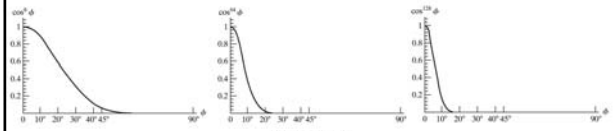
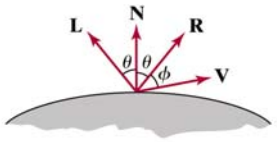


Figure 10-18

Plots of $\cos^{n_s}\phi$ using five different values for the specular exponent n_s.

[*HB*]

Equation with specular reflection added



$$I = k_a I_a O_d + k_d I_l O_d (\overline{N} \cdot \overline{L}) + k_s I_l O_s (\overline{V} \cdot \overline{R})^n$$

noting $\overline{V} \cdot \overline{R} = \cos\phi$ (assuming normalized vectors)

k_s : specular reflection coefficient

O_s : object's specular color

[*HB*]

Calculation of R

Projection of $\overline{L}$ onto $\overline{N}$ is $\overline{N} \cdot \overline{L}$ (vectors normalized)

$$\overline{R} + \overline{L} = 2(\overline{N} \cdot \overline{L})\overline{N}$$

$$\overline{R} = 2(\overline{N} \cdot \overline{L})\overline{N} - \overline{L}$$

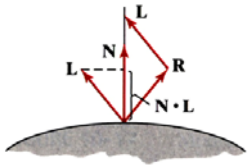


FIGURE 10-20 The projection of either **L** or **R** onto the direction of the normal vector **N** has a magnitude equal to **N** • **L**.

[*HB*]

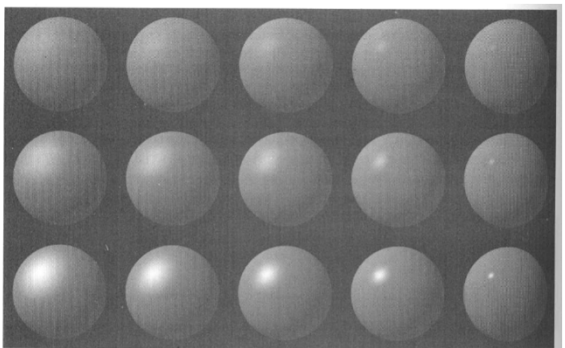


Fig. 16.10 Spheres shaded using Phong's illumination model (Eq. 16.14) and different values of k_s and n. For all spheres, $I_a = I_p = 1.0$, $k_a = 0.1$, $k_d = 0.45$. From left to right, n = 3.0, 5.0, 10.0, 27.0, 200.0. From top to bottom, k_s = 0.1, 0.25, 0.5. (By David Kurlander, Columbia University.)

[*FVD*]

Alternative model: "halfway vector"

Some models replace $(\overline{V} \cdot \overline{R})$ with $(\overline{N} \cdot \overline{H})$

$$\overline{H} = \frac{\overline{L} + \overline{V}}{\left|\overline{L} + \overline{V}\right|}$$

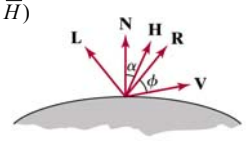


Figure 10-22

Halfway vector **H** along the bisector of the angle between **L** and **V**.

and $\overline{L}$ and/or $\overline{V}$ are sometimes approximated as constant for efficiency

[*HB*]

Adding more light sources (sum them)

$$I = k_a I_a O_d + \sum_l I_l \left[k_d O_d (\bar{N} \cdot \bar{L}) + k_s O_s (\bar{V} \cdot \bar{R})^n \right]$$

(Clamp I at maximum displayable value)

Adding attenuation factors (radial and angular: $f_{l,rad}$ and $f_{l,ang}$)

$$I = k_a I_a O_d + \sum_l I_l f_{l,rad} f_{l,ang} \left[k_d O_d (\bar{N} \cdot \bar{L}) + k_s O_s (\bar{V} \cdot \bar{R})^n \right]$$

Light-source attenuation $f_{l,rad}$

- (Radial intensity attenuation)
- Reduction in light intensity with distance between light source and point being colored
- Inverse square seems like a good choice:

$f_{l,rad} = \frac{1}{d_l^2}$, where d_l is distance from light

$f_{l,rad}$ cont'd

A wider range of effects is achieved with:

$$f_{l,rad} = \min\left(\frac{1}{a_0 + a_1 d_l + a_2 d_l^2}, 1 \right)$$

Real light sources are not just points, so inverse square is not always adequate; or, we may want other effects for artistic purposes

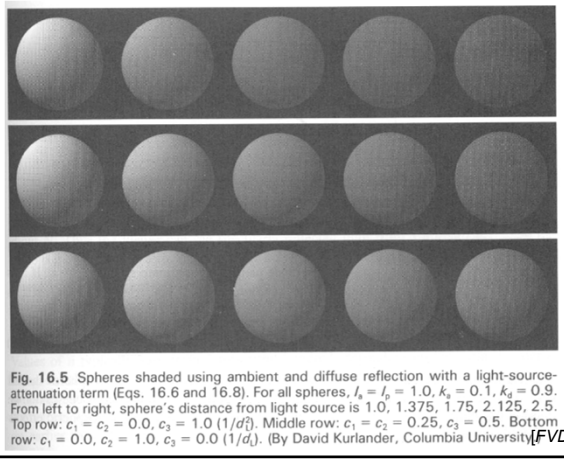


Fig. 16.5 Spheres shaded using ambient and diffuse reflection with a light-source-attenuation term (Eqs. 16.6 and 16.8). For all spheres, $I_a = I_p = 1.0$, $k_a = 0.1$, $k_d = 0.9$. From left to right, sphere's distance from light source is 1.0, 1.375, 1.75, 2.125, 2.5. Top row: $c_1 = c_2 = 0.0$, $c_3 = 1.0$ $(1/d_l^2)$. Middle row: $c_1 = c_2 = 0.25$, $c_3 = 0.5$. Bottom row: $c_1 = 0.0$, $c_2 = 1.0$, $c_3 = 0.0$ $(1/d_l)$. (By David Kurlander, Columbia University)[*FVD*]

Angular intensity attenuation $f_{l,ang}$

Intensity can be varied according to angle between *L* and a light cone axis, for effects such as spotlights (we will not study the details)

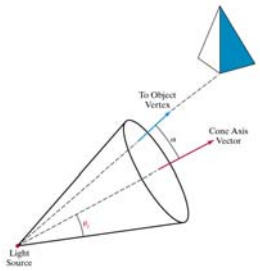


Figure 10-4
An object illuminated by a directional point light source.

[*HB*]

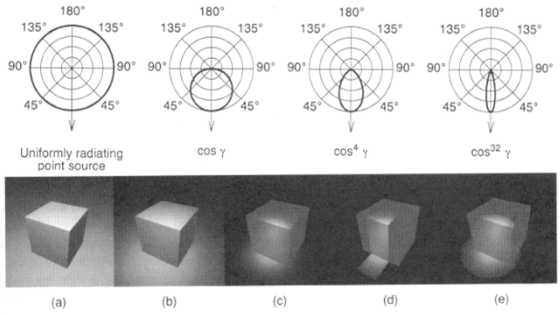


Fig. 16.15 Cube and plane illuminated using Warn lighting controls. (a) Uniformly radiating point source (or $p = 0$). (b) $p = 4$. (c) $p = 32$. (d) Flaps. (e) Cone with $\delta = 18°$. (By David Kurlander, Columbia University.)

[*FVD*]

Atmospheric effects (depth cueing, fog)

- Change in intensity based on distance of eyepoint from object
- Adjust result I from previous equation with:

$$I' = sI + (1 - s)I_{dc}$$

s : is a scale factor between 0 and 1 (as a function of the distance)

I_{dc} : gives a depth-cueing (or fog) color

Paremeter *s* as a function of distance

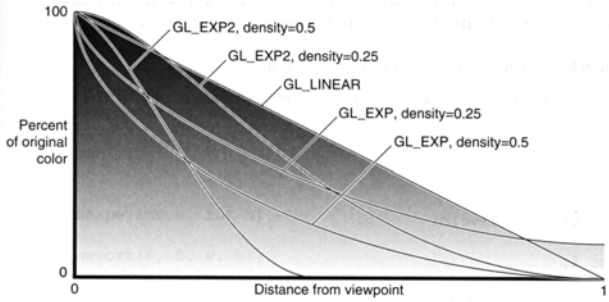


Figure 6-4 Fog-Density Equations

[*Red*]

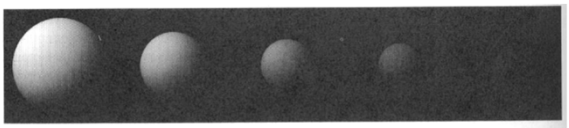


Fig. 16.7 Spheres shaded using depth cueing (Eqs. 16.5, 16.11, and 16.12). Distance from light is constant. For all spheres, $I_a = I_p = 1.0$, $k_a = 0.1$, $k_d = 0.9$, $z_l = 1.0$, $z_b = 0.0$, $s_l = 1.0$, $s_b = 0.1$, radius = 0.09. From left to right, z at front of sphere is 1.0, 0.77, 0.55, 0.32, 0.09. (By David Kurlander, Columbia University.)

[*FVD*]

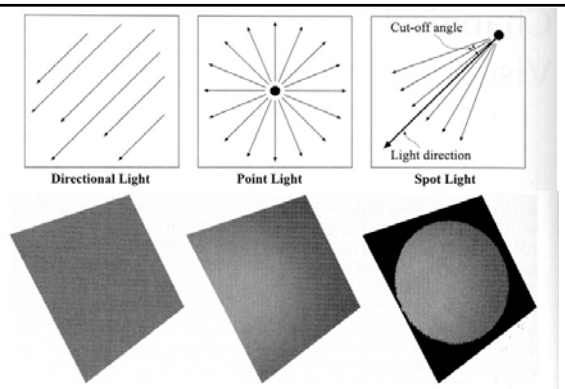


Figure 4.2. Here a square, subdivided into $100 \times 100 \times 2$ triangles, is lit by a directional light source (left), a point light source (middle), and a spotlight (right).

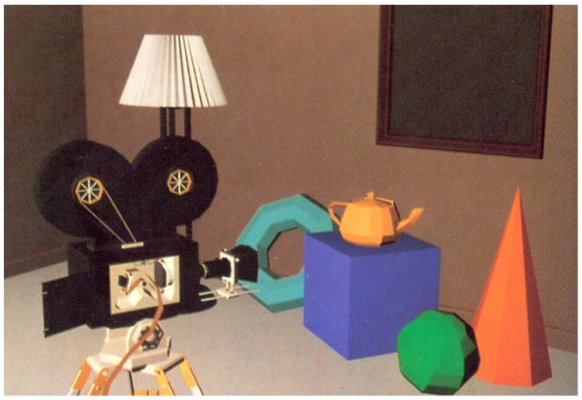
[RTR]

Shading models for polygons : Flat

Where does the lighting equation get applied?

Flat, constant, per-polygon, or faceted shading:

- Evaluate equation at one point on the polygon and apply the result to the entire polygon
- But:
  - $N \cdot L$ and $R \cdot V$ are not really constant across the surface, and polygon may represent curved surface



Flat shading

Another option: Gouraud shading

Gouraud shading, Intensity interpolation, color interpolation, per-vertex lighting

- Evaluate equation at polygon vertices and interpolate resulting color across polygon
- Even though a polygon is flat, when it represents a curved surface, it is given different normals at different vertices

Surface normals

- May be assigned to vertices based on an analytical model of curved surface
  (give an example)
- If unknown, can be estimated by averaging normals from surrounding polygons in a mesh

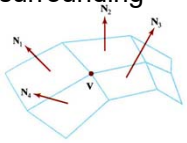


FIGURE 10–47 The normal vector at vertex V is calculated as the average of the surface normals for each polygon sharing that vertex.

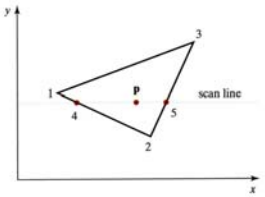
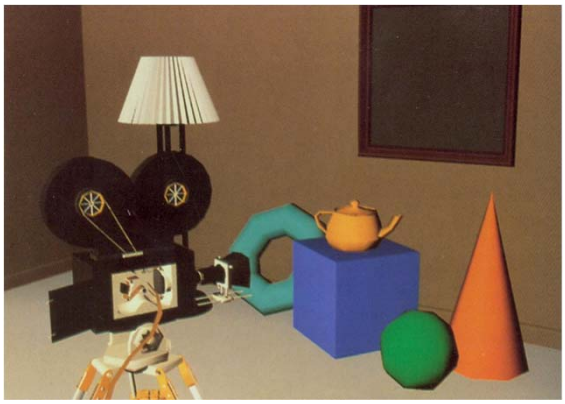


FIGURE 10–48 For Gouraud surface rendering, the intensity at point 4 is linearly interpolated from the intensities at vertices 1 and 2. The intensity at point 5 is linearly interpolated from intensities at vertices 2 and 3. An interior point p is then assigned an intensity value that is linearly interpolated from intensities at positions 4 and 5.

Interpolation note [aside]:

Interpolation of properties across triangle surface is not directly linear in screen space: perspective must be considered (equations in book are not sufficient, but do match some older implementations). Reciprocal of depth does vary linearly with screen coords and can be used as basis for efficient perspectively-correct interpolation (details not discussed).

[*HB*]



Gouraud shading

A third option: Phong shading

Phong shading, normal-vector interpolation, per-pixel lighting

- Interpolate surface normal from vertices and then apply lighting equation per pixel

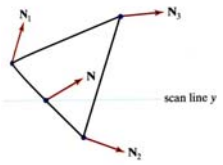


FIGURE 10–51 Interpolation of surface normals along a polygon edge.

[*HB*]

Phong shading (cont'd)

- Renormalize interpolated vectors unless it becomes too costly or vertex normals are very close
- Computationally more expensive than Gouraud shading
- Looks better, especially for specular reflection (why?)

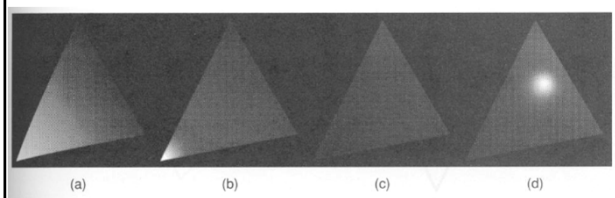
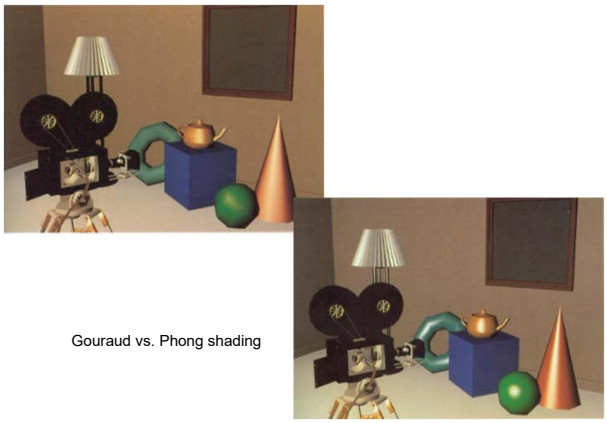
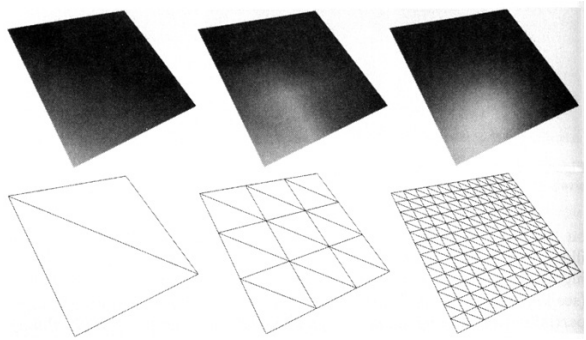


Fig. 16.21 A specular-reflection illumination model used with Gouraud shading and Phong shading. Highlight falls at left vertex: (a) Gouraud shading, (b) Phong shading. Highlight falls in polygon interior: (c) Gouraud shading, (d) Phong shading. (By David Kurlander, Columbia University.)

[*FVD*]



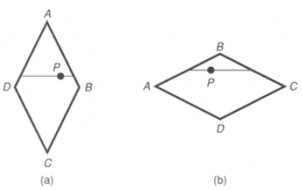
Gouraud vs. Phong shading



Improve Gouraud shading results using subdivision (or use a Phong shader instead) [*RTR*]

Notes about interpolated shading

- Object silhouettes remain polygonal
- Perspective distortion (solvable)
- Orientation dependence (also solvable)



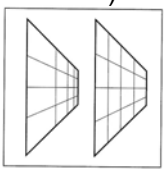
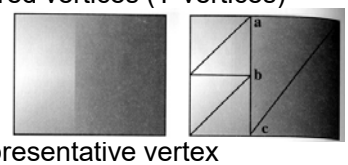


Figure 10.12. Left: correct perspective. Right: interpolation in screen space.

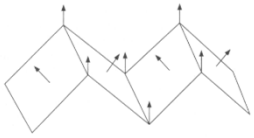
[*Above*: Orientation dependence.
Right: Perspective distortion, from *Fundamentals of Computer Graphics* by Peter Shirley, 2002]

notes (cont'd)

- Problems at shared vertices (T-vertices)



- Chance of unrepresentative vertex normals



[*FVD*]

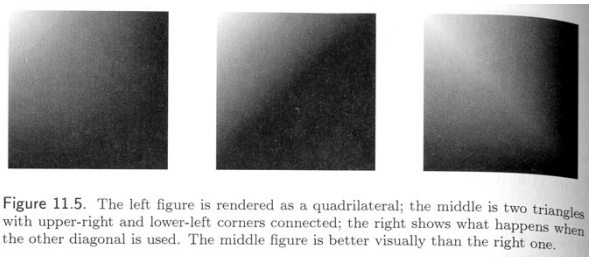
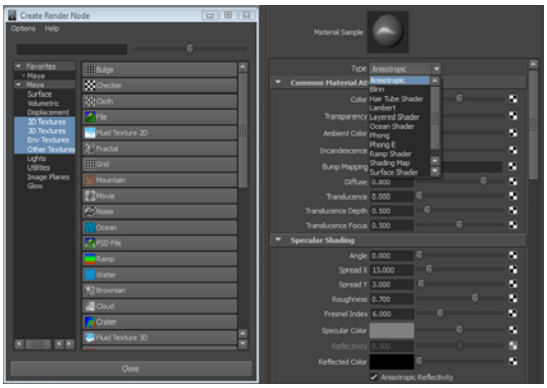


Figure 11.5. The left figure is rendered as a quadrilateral; the middle is two triangles with upper-right and lower-left corners connected; the right shows what happens when the other diagonal is used. The middle figure is better visually than the right one.

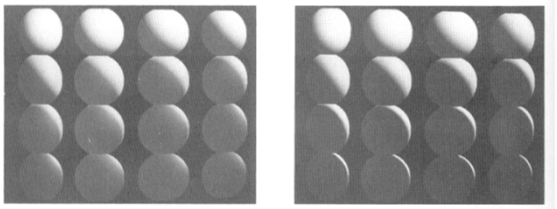
[*RTR*]



Adjusting a material in Maya. At right: drop-down selection of lighting equation and sliders for parameters. Parameters can also vary across surface according to textures (listed at left).

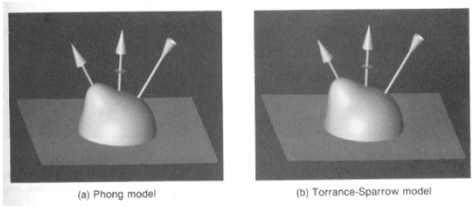
Other Reflection Models

- Copper, lunar surface, brushed metal, etc. need other models, e.g., Torrance-Sparrow

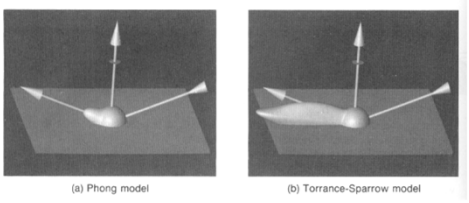


(a) Phong model (b) Torrance-Sparrow model

[*FVD*]



(a) Phong model (b) Torrance-Sparrow model



(a) Phong model (b) Torrance-Sparrow model

[*FVD*]

Other reflection models cont'd

- Anisotropic – not symmetrical about normal

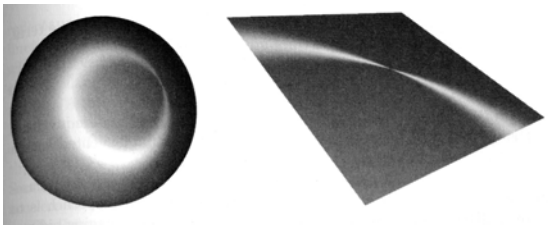
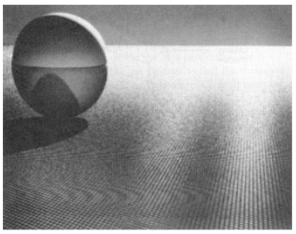


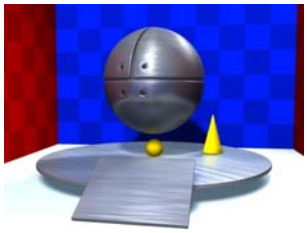
Figure 6.12. Anisotropic surfaces, rendered using general cosine lobe basis functions by Lafortune et al. [451].

[*RTR*]

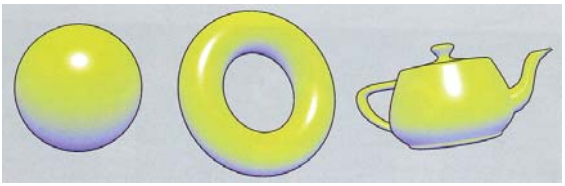
Other reflection models cont'd

- Anisotropic – not symmetrical about normal





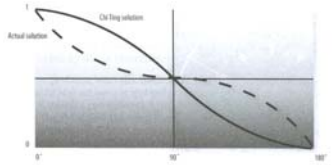
Non-photorealistic rendering (e.g., for technical illustration or cel shading)



Example: Diffuse-like shading blends between warm & cool colors, specular term creates white highlight, and second rendering pass adds outline by rendering only back-facing polygon edges as thick black lines.

[*3Dlabs, Inc.*]

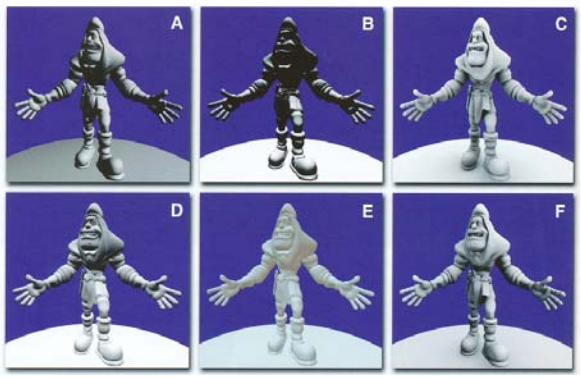
Hemisphere lighting



Shade based on angle between surface normal and "up" direction.

Instead of point light, it's more like a cloudy sky + ground : Maximum illumination for upward-pointing surface, minimum for downward-pointing.

[*OpenGL Shading Language, 2nd Edition*, R. Rost, 2006]



A & B : Standard direct lighting equation, C: Ambient occlusion,
D: Hemisphere lighting, F: Ambient occlusion combined with diffuse lighting,
E: Spherical harmonic lighting (not discussed in class) [*3Dlabs, Inc.*]

Shade trees and shader language (1984)

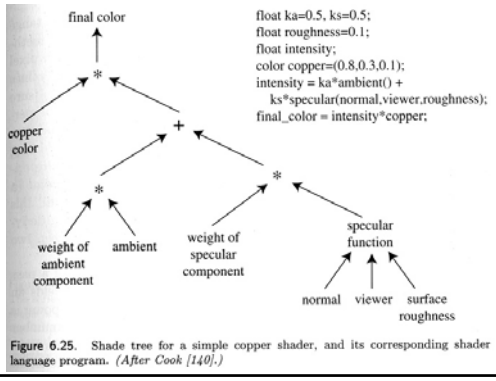


Figure 6.25. Shade tree for a simple copper shader, and its corresponding shader language program. *(After Cook [140].)*

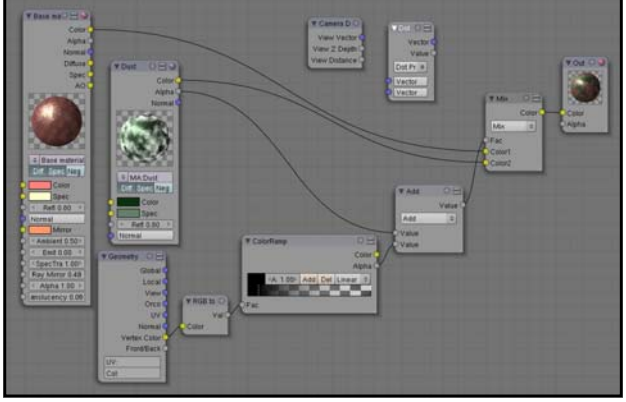
Renderman Shading Language example

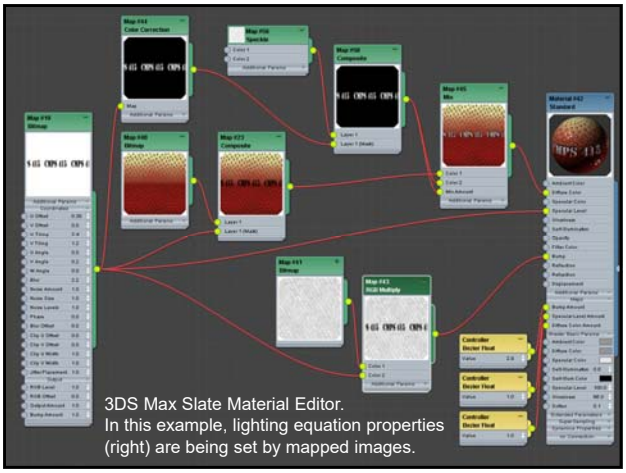
```
surface SimpleDiffuse()
{
     normal Nf = faceforward(normalize(N), I);
     Ci = Cs * diffuse(Nf);
     Oi = 1;
}
```

Sets output surface color (Ci) to input surface color (Cs) multiplied by diffuse color (intensities) of incident light considering forward-facing normal. Sets Opacity (Oi) to 1.

Except Nf, variables are Renderman's standard input / output parameters for the shader.

Blender's node editor for materials. Input and output channels of nodes are connected to combine shaders or object/scene variables to define material appearance.





3DS Max Slate Material Editor.
In this example, lighting equation properties (right) are being set by mapped images.

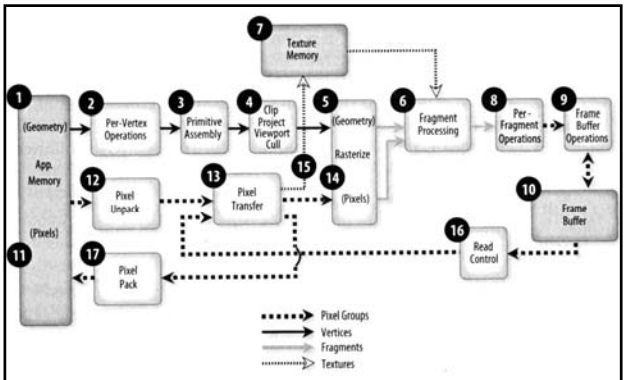


Figure 1.1 Overview of OpenGL operation

[*OpenGL Shading Language, 2nd Edition*, R. Rost, 2006]

1) Application-controlled memory (could use graphics card memory with buffer objects)
2) Per-vertex ops. For example:
   - Apply modelview and projection to vertices
   - Transform normals and texture coordinates
   - Per-vertex lighting for Gouraud shading
3) Primitive assembly: Collect vertices for complete primitives
4) Primitive processing: Clipping, perspective division, viewport transform, culling of back (front) faces if enabled

5) Rasterization: generates fragments (pixels with window coords, depth, color, texture coordinate, etc., interpolated)
6) Fragment processing: For example:
   - Apply texture based on coord and settings
   - Per-pixel lighting for Phong shading
   - Apply fog equation
7) Texture memory
8) Per-fragment ops such as depth test, stencil test, blending
9) Ops affecting frame buffer writes

Notes:

The rest deals mostly with pixels: reading (16), writing and organizing into memory (17), color scaling and biasing (13), etc.

The diagram is conceptual: there are various implementations / optimizations

It is somewhat dated and evolving: e.g., addition of geometry shader before (4)

Lighting Review

- Illumination equation: what do the different terms represent; what is their visual effect
- Basic shading methods (flat, gouraud, phong): compare them

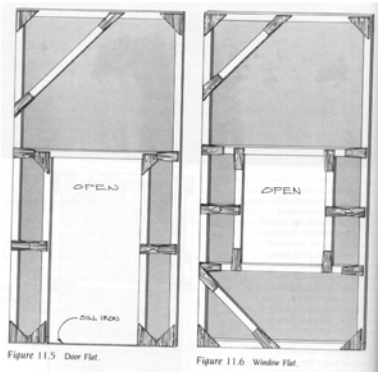


Figure 11.5 Door Flat. Figure 11.6 Window Flat.

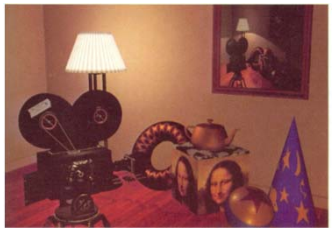
Door and window flats from *The Art of Theater* (D. J. Sporre)

Cardboard set by Cortec

Halloween crypt by "DeadTed" (halloweenforum.com)

Texture mapping (pattern/color mapping)

- Most basic map: surface color set according to an image (stored or function-generated)



Texture mapping (cont'd)

- Adds fine surface detail without requiring additional geometric primitives
- Surface color is determined from texture map elements ("texels")
- Texture coords (usually specified at polygon vertices) are used to map between texture space and object surface

Mapping of pixel area to texture area

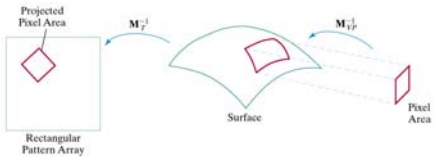


Figure 10-105
Texture mapping by projecting pixel areas to texture space.

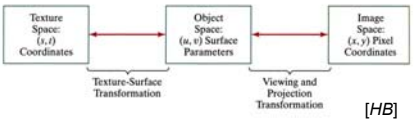


FIGURE 10-104
Coordinate reference systems for two-dimensional texture space, object space, and image space.

[*HB*]

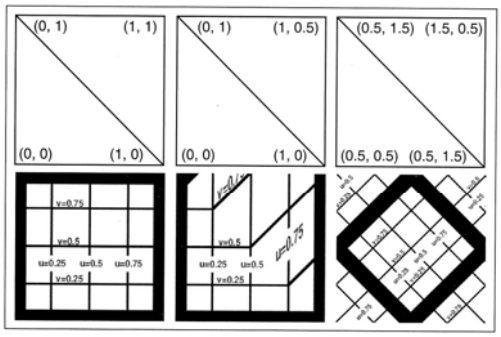
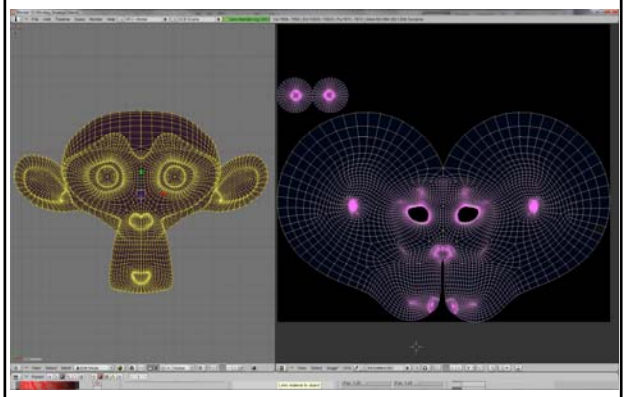
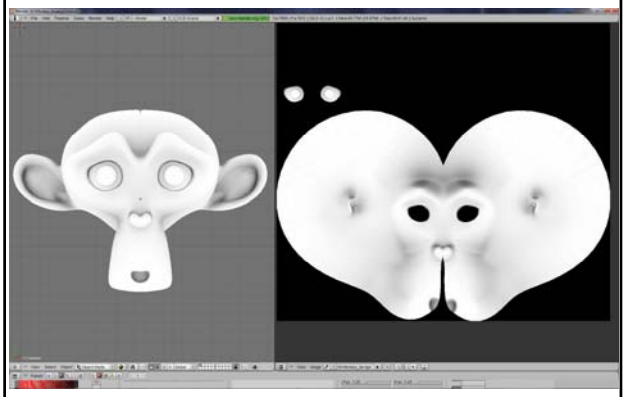


Figure 10.10. Various mesh textures obtained by changing (u,v) coordinates stored at vertices.

WARNING: Right image has errors; corrected in lecture.

[*Fundamentals of Computer Graphics*, Peter Shirley, 2002]



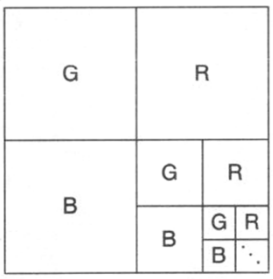


A basic mapping process for 2D textures

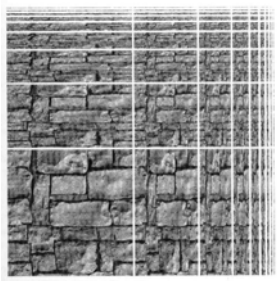
- At each vertex of a 3D object, a 2D texture coordinate is specified
- When a polygon is rendered, texture coordinates are interpolated to produce a texture coordinate for each rendered pixel
- This interpolated texture coordinate specifies a point on the texture map

Common approaches to find pixel color

- Fast method: select texel nearest to the interpolated texture coord (GL_NEAREST)
- For better quality: weighted average based on a group of texels, so magnified/minified maps look smoother
- Mipmaps or ripmaps - precompute multiple resolutions of the texture map, and pick from the most suitable resolution



MIPmap structure



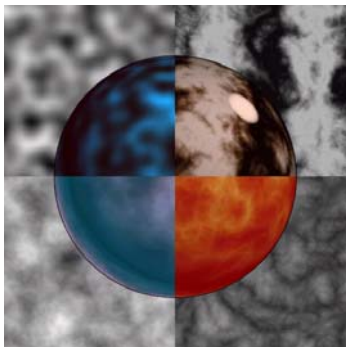
ripmap structure

[*RTR*]

Procedural texture maps

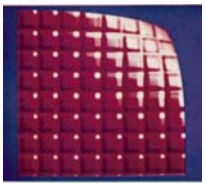
- Texture can come from a mathematical function rather than a stored image

Procedural noise

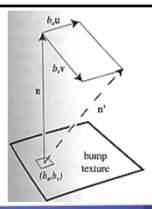


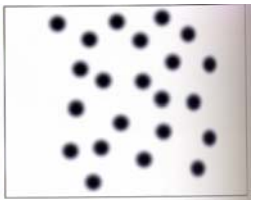
Bump mapping

- Bump mapping perturbs the surface normal to create the illusion of bumps. Grayscale map represents these bumps (level represents bump height or depth)



[Images from the 1970s, by Jim Blinn]



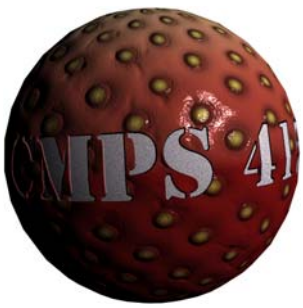


[Top left: perturbations (in two tangent directions) applied to normal.
Top right: bumpy surface side view (bump mapping simulates, but on flat surface).
From *Real-Time Rendering* (top) and Watt's *3D Computer Graphics* (bottom)]

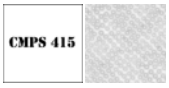
Bump mapping

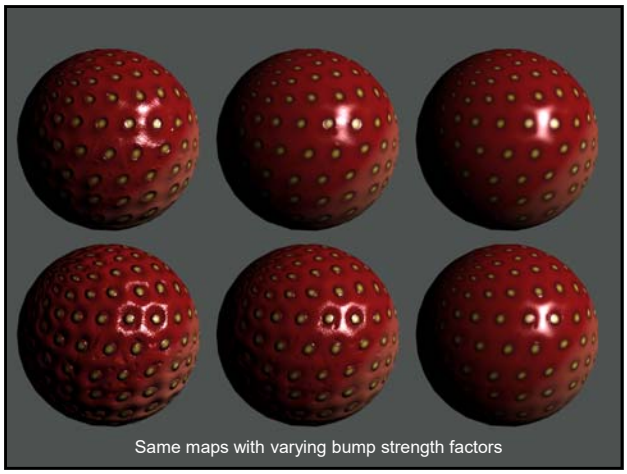
- It does *not* change surface geometry, although it appears to add bumps.
- Eqn. 10-116 approximates an adjusted normal for any point in the bump map
  - Main idea: rates of change in bump map value w.r.t. bump map axes give perturbation amounts in two directions tangent to surface. Add these to normal and renormalize.
- Related techniques: normal maps, parallax mapping

Color maps only
(diffuse color)



Added bump maps
[strawberry maps are from Filter Forge]

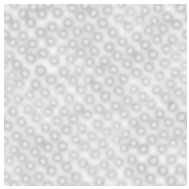




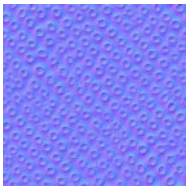
Same maps with varying bump strength factors

Bump maps only (no color map) with changing light position. Perturbing the normal allows light direction to be taken into account in the bump illusion (unlike for a simple color map). At sphere boundaries and edges of the "CMPS 415" text, we see there are no actual bumps in geometry.

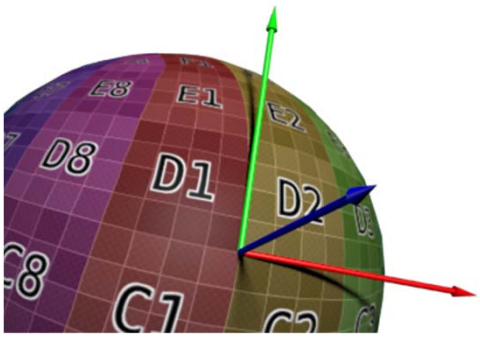
- Normal map: same effect as bump map
  - Normal stored directly in 3 color channels, rather than computed using bump value. Can improve efficiency and accuracy.
  - UV-relative normal; straight-outward normal is (0,0,1) so normal maps tend to look blue

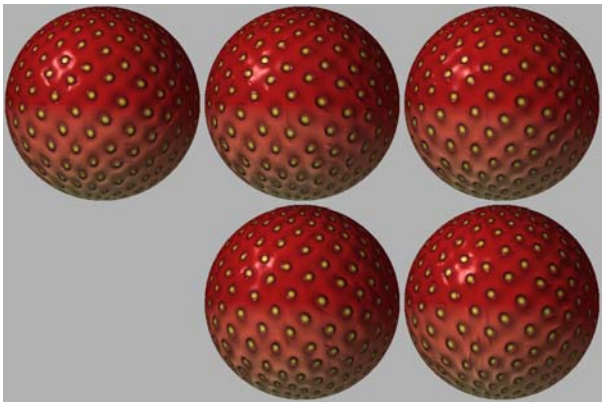


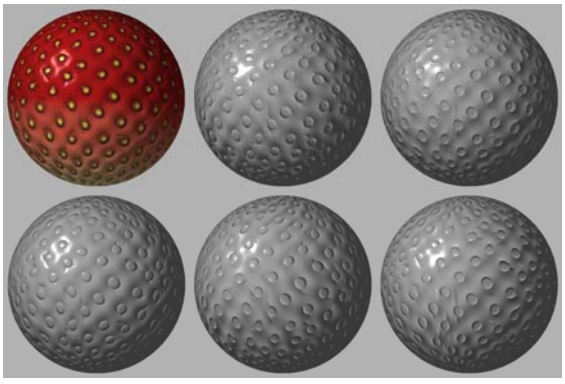
Bump map version



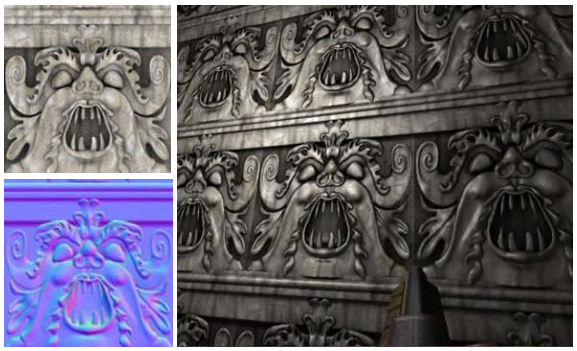
Normal map version



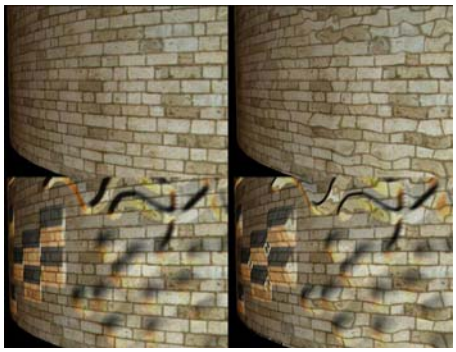




Color Map + Normal Map



Parallax mapping: an extension that distorts the pixel texture coordinates to better simulate the effects that would occur if a bump were really there. Still no actual geometry change.

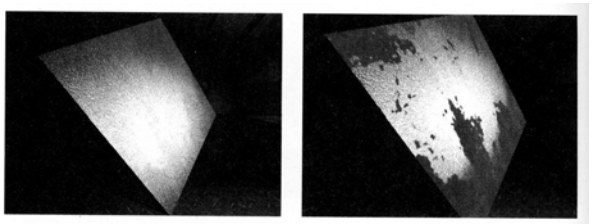


Texture + Bump Parallax Mapping [Terry Welsh]

Some related techniques and terms

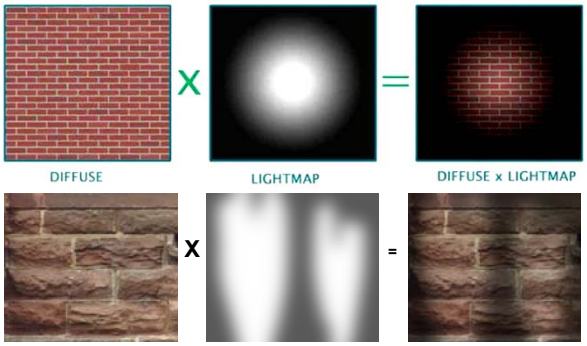
- Displacement maps: related to bump maps, but actual geometry is perturbed
- Environment maps (reflection maps): image of surroundings mapped to surface according to reflected view vector
- Light maps: precomputed lighting info
- Others: to map almost any parameter of lighting calculation

Gloss Mapping - scales specular component

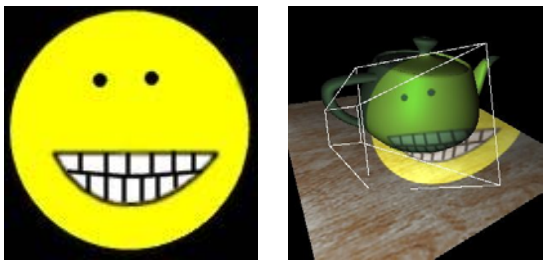


[Rust example with 0 specular level at rusted areas, *RTR*]

Light maps (dark maps)

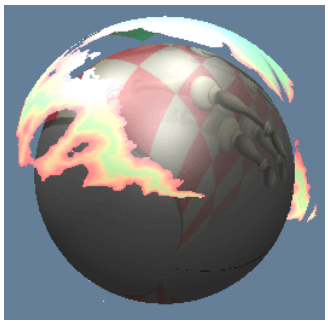


Projecting a texture



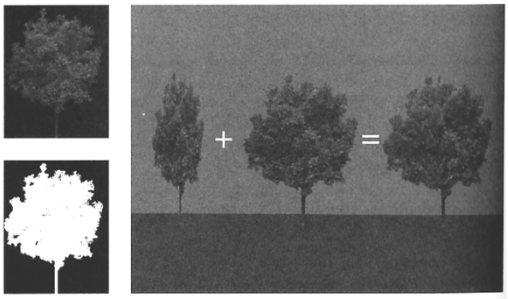
[Texture indexed by light direction relative light frame; Cass Everitt *www.r3.nu*]

Transparency, using alpha (A of RGBA)



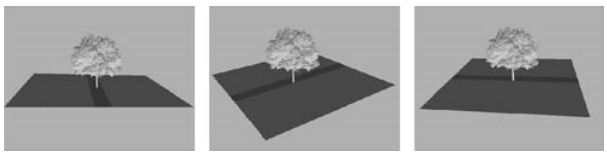
Two concentric spheres with alpha channel controlling transparency of outer sphere

Billboarding



Map with color (top left) and alpha (bottom left) components, painted onto quadrilaterals

Axial Billboard



Tree billboard rotating about a vertical axis so it always faces the viewer

[*RTR*]

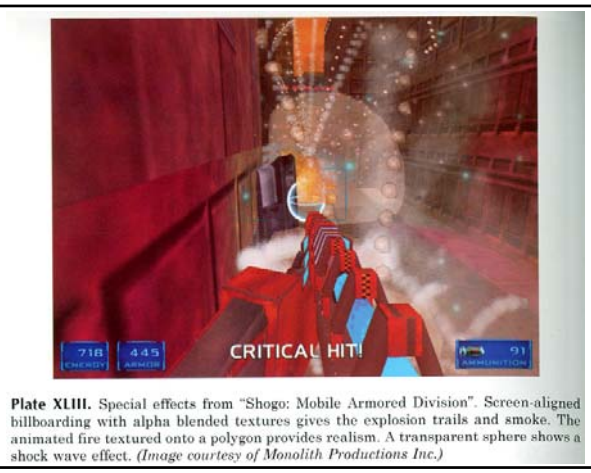
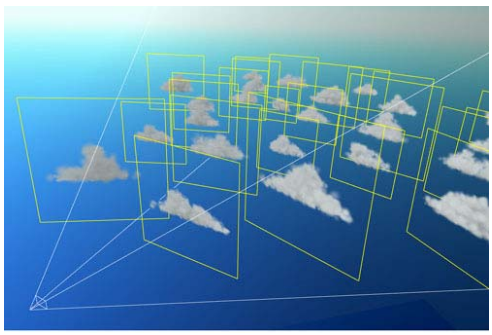


Plate XLIII. Special effects from "Shogo: Mobile Armored Division". Screen-aligned billboarding with alpha blended textures gives the explosion trails and smoke. The animated fire textured onto a polygon provides realism. A transparent sphere shows a shock wave effect. *(Image courtesy of Monolith Productions Inc.)*

World-oriented imposters



[Mark Harris *www.markmark.net*]

Nearby clouds are imposters, distant clouds are a "skybox"

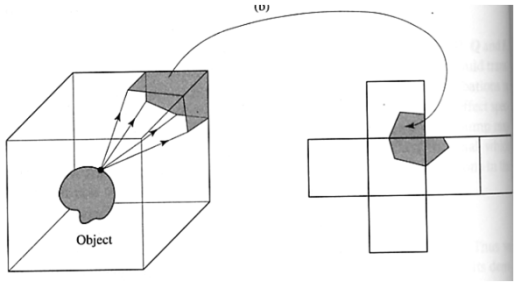
Avatar Imposters for real-time crowd simulation [Tecchia et al., 2002]

Environment map example

Cubic reflection map: a technique for cheap, approximate, mirror-like reflections. The map is precomputed by considering a cube surrounding a reflective object and projecting the scene onto the 6 sides.

[from *3D Computer Graphics*]

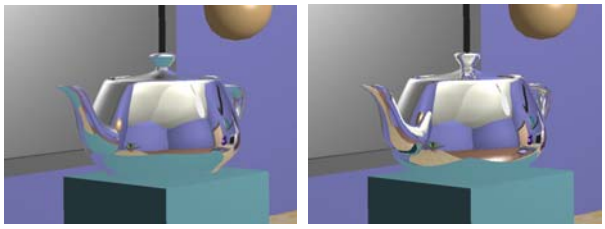
Then, lighting determines surface color by reflecting the view vector about the surface normal and mapping the result to a location on the cube map.



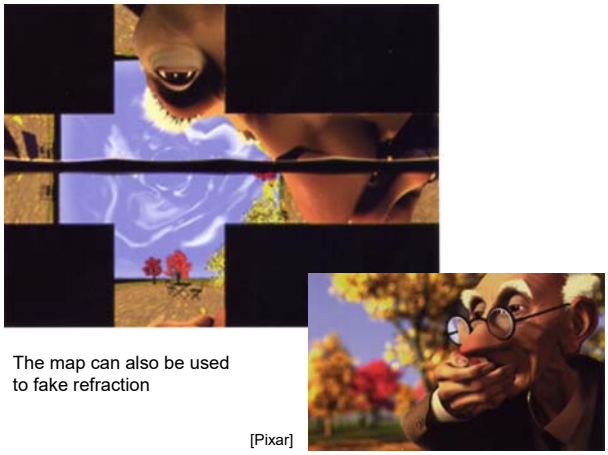
[*The Computer Image*, Watt and Policarpo, 1998]

Environment map vs. ray tracing

This only produces accurate reflection for a surface at the box's center, since that is the projection point used when rendering the map
What differences do you see when comparing to a more accurate technique?



[*3D Computer Graphics*]



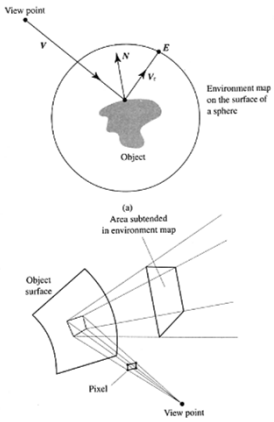
The map can also be used to fake refraction

[Pixar]

Use of a cubic environment map for reflection, refraction, and as a "skybox" background [NVIDIA]

Spherical environment map

A 360-degree fisheye view created
From a a 180-degree fisheye photo



[diagram from *The Computer Image*]

Some more examples

Environment mapping by Jim Blinn in the 1970s

Use of a photographed light probe by Gene Miller in 1982

[Terminator II robot by ILM]