Chapter 2

A top-down approach -
How to make shaded images?
Graphics API vs. application API

• Graphics API
  – Support rendering pipeline for polygon stream
  – Pipeline process supported by graphics hardware
  – Examples
    • OpenGL, Direct 3D

• Application API
  – A software engine that calls graphics API for rendering a scene
    • Scene management, real-time rendering, animation modules,…
  – Home made programs vs. programming using application API
    • Either one will call graphics API for rendering polygons
  – Examples
    • OpenInventer, WTK, Unreal3 (game engine), Ogre, Utility,…
How to make shaded images?
Rendering Pipeline

- Per-Vertex Operations
- Primitive Assembly
- Clip Viewport Cull
- Fragment Processing
- Per Fragment Operations
- Frame Buffer Operations

(App. Memory)
(Pixels)

Pixel Groups
Vertices
Fragments
Textures
How to make shaded images?

Input

- **Set up scene, light, viewing data**
  - **Object geometry and material**
    - Geometry: Primitives, ex., polygon meshes
      - vertex attributes
        » Coordinates, normal, color, and texture coordinates
    - Texture maps for material and others
      - Color map, normal map, bump map, displacement map, light map, environment map,…
  - **Lighting information**
    - Light setting, light control parameters
  - **Viewing information and view volume**
    - Eye point, view direction/orientation, view window/volume
• **Per-vertex operations**

**View transformation/lighting stage**

- Apply viewing transformation to transform polygon vertices to view coordinate system and then do the projection
- Transform vertex normal
- Compute lighting (Phong model) at each vertex in view coordinate system
- Generate (if necessary) and transform texture coordinates
• **Primitive assembly**
  – **Input to the pipeline – vertices**
  – **Assemble vertices to primitives**
  • Polygon, triangle, polyline
Rendering Pipeline

- **Clipping**
  - Clip the triangle against the normalized view volume
  - Back-face culling
  - Perspective division
Note on Back-Face Culling

- Front-facing faces have normals facing forward the viewer.
- A polygon is front-facing when the angle between \( N \) and \(-V\), lies between \(-90^\circ\) and \(90^\circ\). i.e., a polygon is front-facing if

\[-V \cdot N = \lVert -V \rVert \cdot \lVert N \rVert \cos \theta > 0\]

where \( V \) is the vector from \( P \) to any point on the polygon.
Note on Back-Face Culling

Assuming perspective view projection has been performed (in normalized device coord.), or parallel projection is used

- Right-handed screen coord. system: viewing direction along negative \(Z_v\) axis.

\[ V = (0, 0, -1) \]
\[ N \cdot -V = C \]
if \(C < 0\), back-facing
if \(C > 0\), front-facing
Note on Back-Face Culling - 3

- Left handed screen coor. system:
  Viewing direction along the positive $Z_v$ direction

$V = (0, 0, 1)$
$-V \cdot N = -C < 0$

i.e. $C > 0$

$N = (A, B, C)$
If $C > 0 \Rightarrow$ Back face.
Rendering Pipeline

- Rasterization
  - Input are transformed vertices and associated colors and texture coordinates, depths
  - Scan convert polygons, and yield fragments consisting of depths, color, alpha value, and texture coordinates by linear interpolation, except for the texture coordinates, which are done in a perspectively correct way
Rendering Pipeline
Rendering Pipeline

- Fragment processing
  - Access texture maps via a lookup and does blending of pixel color and texture color
    - Also handle multi-texturing
  - Fog
    - Blend the fog color with the fragment color based on the depth value
Rendering Pipeline

- **Per-fragment operations**
  - **Alpha test** (compare with alpha, keep or drop it) and **Alpha blending**
  - **Stencil test** (mask the fragment depending on the content of the stencil buffer)
  - **Depth test** (**z buffer algorithm**)
  - **Dithering** (make the color look better for low res display mode)
  - **Fragments passing all tests are written to frame-buffer**
Rendering Pipeline

- Application Operations
- Geometry
- Primitive Assembly
- Viewport Cut
- Fragment Processing
- Texture Memory
- Frame Buffer Operations
- Frame Control

Pixel Groups
Vertices
Fragments
Textures

Comp. Graphics (U), Chap 2 Global View
CGGM Lab., CS Dept., NCTU  Jung Hong Chuang
Rendering Pipeline

- Frame-buffer operations
  - Operations performed on the whole frame buffer
  - Accumulation buffer operations for multipass rendering
OpenGL Viewing Pipeline

Viewing projection

- View transform
- View-Volume Clipping
- Window-viewport mapping
- Back-face culling
OpenGL Viewing Pipeline

Original vertex data → Modelview matrix → Transformed eye coordinates → Projection matrix → Clip coordinates → Perspective division → Normalized device coordinates

Viewport transformation → Window coordinates

Comp. Graphics (U), Chap 2 Global View
Viewing Pipeline -1

• View transformation
  – Transformation objects from world coordinate system to the view coordinate system
    • For fast perspective projection
    • Involves translation and rotation

• View projection
  – Transforms the view volume to a normalized view volume and does the orthogonal projection.
    • Easy for hardware 3-D clipping
    • Any projection becomes orthogonal projection
    • Clipping can be unified for perspective and parallel projection
    • Involves translation, shearing, and scaling
Viewing Pipeline - 2
View Transformation

[Diagram showing view transformation and frustum.]
Viewing Pipeline

View Projection

View-volume culling w.r.t. view volume in world/eye space
Converting view frustum to normalized view volume
Viewing Pipeline - 4

View Projection

• Converting view frustum to normalized view volume ensures
  – Any type of projection becomes orthographic projection
    • Distort the objects and do the orthogonal projection
    • Orthographic projection of the distorted objects
      = the projection of the original object
  – Preserves depth ordering
Viewing Pipeline - 5

View Projection

View-volume culling w.r.t. normalized view volume in screen space
Viewing Pipeline

Window-to-Viewport Mapping

- Window-to-viewport mapping
  - Map the window on the view plane to a viewport on the screen
  - Involves translation and scaling
3-D Geometric Transformations

• Building up a scene using 3-D transformation to instance objects
  – For a complex object or scene, each subpart may have their own local coordinate systems.
  – Instancing a subpart by transforming the local coordinate system by a series or translations, scalings, and rotations

• Projecting 3D objects onto a 2D view plane
  – Involves scaling, translation, rotation, and shearing
Scene/Object Instancing
Translation

- Translation
  - Displaces points by a fixed distance in a given direction, specified by a displacement vector.
  - A rigid-body transformation
  - The definition makes no reference to a frame.
  - The matrix has 3 degrees of freedom for specifying the displacement vector

Matrix form in 3D? No. It is not linear, but affine
Rotation

- **A rigid-body linear transformation**
- **Cases**
  - 2D rotation w.r.t the origin
  - 2D rotation w.r.t a fixed point
  - 3D rotation w.r.t. an axis
  - 3D rotation w.r.t. an arbitrary axis
Scaling

- **A non-rigid-body linear transformation**
- To specify a scaling, we need to specify a fixed point and a direction in which we wish to scale, and a scale factor $\alpha$
- **Negative value of $\alpha$ gives us reflection.**
Shearing

- A non-rigid-body **linear** transformation.
- Shear in one axis direction, others unchanged
- Example: shear in $x$-direction

\[
\begin{cases}
  x' = x + y \cot \theta \\
  y' = y \\
  z' = z
\end{cases}
\]
Affine transformation

- Linear transformation \( X' = LX \)
- Affine transformation \( X' = LX + b \)
  - Preserving affine combinations of points
    - Interior points of a line will be linear combination of the transformations of vertices
      - So what we must deal with are vertices
  - Preserve lines and parallelism of lines, but not lengths and angles. Examples:
    - Rotation, scaling, shearing, translation
    - Rotating a unit cube by a degree and scale in \( x \), not in \( y \)
Affine transformation \(^{-2}\)

Preserves affine combinations/lines

Two points: \(P_0, P_1\)

Affine transformation: \(A = Lx + b\)

Claim: \(A(\alpha P_0 + (1 - \alpha) P_1) = \alpha A(P_0) + (1 - \alpha) A(P_1)\)

\[
A(P(\alpha)) = L[\alpha P_0 + (1 - \alpha) P_1] + b \\
= \alpha L(P_0) + (1 - \alpha) L(P_1) + [\alpha + (1 - \alpha)] b \\
= \alpha [L(P_0) + b] + (1 - \alpha) [L(P_1) + b] \\
= \alpha A(P_0) + (1 - \alpha) A(P_1)
\]

Preserves relative ratios too!!
Affine transformation \( \mathbf{T} \)
Preserving affine combinations

Preserves relative ratios!!
Affine transformation

Preserving parallelism of lines

Lines: \( P(\alpha) = P_0 + \alpha v \parallel Q(\beta) = Q_0 + \beta v \)

Affine transformation: \( A = Lx + b \)

Claim: \( A(P(\alpha)) \parallel A(Q(\beta)) \)

\[
A(P(\alpha)) = A(P_0 + \alpha v) \\
= L(P_0 + \alpha v) + b \\
= [L(P_0) + b] + \alpha L(v) \\
= A(P_0) + \alpha L(v)
\]

\[
A(Q(\beta)) = A(Q_0) + \beta L(v)
\]

where \( L(v) \) is a direction vector, independent on \( P_0 \) or \( Q_0 \).
Homogeneous Coordinates

- We need a uniform matrix representation so that matrix concatenation can represent a series of affine transformations using a single matrix.

- Converting from Cartesian space to the homogeneous space

  Mapping \((x, y, z) \rightarrow (X, Y, Z, W)\) by
  \[x = X/W, \quad y = Y/W, \quad z = Z/W, \quad W \neq 0\]
Homogeneous Coordinates

Cartesian coordinate: \((x, y)\)

Homogeneous coordinate: \((X, Y, W)\)

where \(x = X / W\) and \(y = Y / W\)

Properties:

\(\oplus\) \((x, y)\) corresponds to a line in \((X, Y, W)\) space.

\(\oplus\) Dehomogenize \((X, Y, W)\) \(\Rightarrow\) \((X / W, Y / W, 1)\)

\(\oplus\) Dehomogenize all points \((X, Y, W)\) forms the plane defined by \(W = 1\) in homogeneous space.
Homogeneous Coordinates

- An affine transformation $A$ can be represented in homogeneous space as a $4\times4$ matrix

$$
\begin{bmatrix}
    x' \\
    y' \\
    z' \\
    1
\end{bmatrix}
= A(\overline{x}) = L\overline{x} + b =
\begin{bmatrix}
    \alpha_{11} & \alpha_{12} & \alpha_{13} & b_1 \\
    \alpha_{21} & \alpha_{22} & \alpha_{23} & b_2 \\
    \alpha_{31} & \alpha_{32} & \alpha_{33} & b_3 \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix}
$$

- This transformation has 12 degrees of freedom
Transformation in homogeneous coordinates -1

- Translation: \( p' = T \ p \)

\[
T(\alpha_x, \alpha_y, \alpha_z) = \begin{bmatrix}
1 & 0 & 0 & \alpha_x \\
0 & 1 & 0 & \alpha_y \\
0 & 0 & 1 & \alpha_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

- Scaling: \( p' = S \ p \)

\[
S(\beta_x, \beta_y, \beta_z) = \begin{bmatrix}
\beta_x & 0 & 0 & 0 \\
0 & \beta_y & 0 & 0 \\
0 & 0 & \beta_y & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Transformation in homogeneous coordinates

- **Rotation:** $p' = R_z p$
  
  \[
  R_z(\theta) = \begin{bmatrix}
  \cos \theta & -\sin \theta & 0 & 0 \\
  \sin \theta & \cos \theta & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]

- **Shearing:** $p' = H p$
  
  \[
  H_x(\theta) = \begin{bmatrix}
  1 & \cot \theta & 0 & 0 \\
  0 & 1 & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]
Concatenation of transformations

- $q = CBAp = C(B(Ap)) = Mp$, where $M = CBA$

**Examples**
- Rotation about a fixed point
- General rotation about the origin
- Rotation about an arbitrary axis
Concatenation of transformations

- Rotation about a fixed point
  - A cube with its center p and its sides aligned with the axes. Rotate it about z-axis, about its center.
  - $M = T(p)R_z(\theta)T(-p)$
Concatenation of transformations -3

- Rotation about the origin
  - A cube with its center \( p \) at the origin and its sides aligned with the axes.
  - \( R = R_x R_y R_z \)
Concatenation of transformations -4

- Rotation about an arbitrary point and line
  - Move the fixed point to origin
  - Rotate such that the vector is coincident to z-axis
  - Rotate about z-axis

\[ M = T(p_0)R_x(-\theta_x)R_y(-\theta_y)R_z(\theta_z)R_y(\theta_y)R_x(\theta_x)T(-p_0) \]
Projections

- Parallel projection

- Perspective projection
Projections
Parallel projection

Diagram of parallel projection with a three-dimensional object, a projector, and a projection plane.
Projections
Parallel projection
Projections
Orthogonal projection

Projectors are orthogonal to projection surface
Projections

Oblique projection

Arbitrary relationship between projectors and projection plane
Projection
Perspective Projections
Projections
Perspective projection
Projections
Perspective projection
A Practical Viewing System

- Users specify
  - Projection type
  - A camera position $C$ and a viewing direction vector $N$ (= view plane normal), and an upvector $V$
  - A view plane distance $d$, near clipping plane distance $n$, and a far clipping plane distance $f$.
  - A view window is specified but it is symmetrically disposed about the center of the view plane.
Viewing Parameters

World coordinate system

View coordinate system

View plane window

Camera position

Here without near and far clipping planes
Perspective view volume
Parallel view volume

![Parallel View Volume Diagram]

- Parallelepiped View Volume
- Front Plane
- Back Plane
- Parallel Projection
- Window

Far Near
View coordinate system

- View coordinate system \((u, v, n)\)
  - \(v = V\), \(n = -N\), \(u = v \times n\), where \(V\) is perpendicular to \(n\)
Viewing Pipeline

• **View transformation: $T_{\text{view}}$**
  - world coord. sys. $\rightarrow$ view coord. sys.
  - $T_{\text{view}} = B \cdot T$
  - For simpler and faster clipping and projection

• **View projection: $T_p$**
  - view coord. sys. $\rightarrow$ screen coord. sys.
  - Convert view volume to normalized view volume and then do orthogonal projection.
  - $T_p = T_{\text{pers2}} \cdot T_{\text{pers1}}$
View Transformation

• View transform matrix $T_{view}$

$$\begin{bmatrix} x_v, y_v, z_v, 1 \end{bmatrix} = T_{view} \begin{bmatrix} x_w, y_w, z_w, 1 \end{bmatrix}^T$$

• $T_{view}$ can be written as $T_{view} = B \cdot T$, where
  - $T$ translates the world coordinate origin to $C$.
  - $B$ rotates any vector expressed in the world coordinate system into the view coordinate system by mapping $(u, v, n)$ to $(e_1, e_2, e_3)$. 
View Projection

(left, top, −N)

perspective transformation

Comp. Graphics (U), Chap 2 Global View 60 CGGM Lab., CS Dept., NCTU Jung Hong Chuang
View Projection

\[ z = z_{\text{min}} \quad \rightarrow \quad x \quad \rightarrow \quad z = z_{\text{max}} \]

COP

\[ z = 0 \]

\[ z = 1 \]

\[ x = 1 \]
View Projection

- Mapping view volume to normalized view volume, then following by orthogonal projection
  - Distort the objects and then do the orthogonal projection

- Why?
  - Easy view-volume clipping and projection
  - Standard sized window
Viewing Pipeline Summary

View transform
V-V clipping in homogeneous space
Window-Viewport mapping
Lighting Effects

- Lighting effects involve
  - light propagation
  - Light-material interaction
    - The interaction of light with a surface, in terms of the surface properties and the nature of the incident light.

- Computing lighting effects in OpenGL fixed pipeline
  - Vertex-lighting + Polygonal shading

- Some remarks
  - Local illumination vs. global illumination.
  - Polygonal shading vs. Per-pixel lighting

For lighting effect/shading, see Chap 5 of Angel’s book (6th ed.)
Lighting Effects

• Local lighting
  – Consider lights that directly come from the light sources
    • The light-surface interaction depend on only the surface material properties, surface’s local geometry, and parameters of the light sources
  – Can be added to a fast pipeline graphics architecture

• Global lighting
  – Lighting is a recursive process and amounts to an integral equation (the rendering equation)
  – Approximations: ray tracing and radiosity
Light Sources

- **Light source**
  - An object that emits light only through internal sources

- **Illumination function**
  - Each point \((x,y,z)\) on the surface of the light source can emit light that is characterized by

\[
I(x, y, z, \theta, \phi, \lambda)
\]

- \((\theta, \phi)\) : the direction of emission
- \(\lambda\) : wavelength
Light Sources -2

- On a surface point that is illuminated
  - Total contribution of the source is obtained by integrating over the surface of the source.
  - The calculation is difficult for a distributed light source.
Light Sources

- Four types of light source
  - **Ambient lighting**
    - Provides a uniform light level in the scene.
    - Can be characterized by a constant intensity $I_a = [I_{ar}, I_{ag}, I_{ab}]$
    - Each surface can reflect this light differently.
  - **Point light sources**
  - **Spotlights**
  - **Distant light**
Light Sources

• **Point light sources**
  – An ideal point source emits light equally in all directions.
  – A point source at \( p_0 \) can be characterized by
    \[
    I(p_0) = \begin{bmatrix}
    I_r(p_0) \\
    I_g(p_0) \\
    I_b(p_0)
    \end{bmatrix}
    \]
  – At a surface point \( p \), the intensity of light received from the point source is
    \[
    I(p, p_0) = \frac{1}{|p - p_0|^2} I(p_0)
    \]
Light Sources

- **Spotlight**
  - A point source with limited angles at which light can be seen.
  - Represented as a cone
  - Distribution of the light within the cone
    - Uniform
    - Represented as a function of $\theta$, such as $\cos^e \theta$
**Light Sources**

- **Distant light sources**
  - The light source is far from the surface.
  - The source becomes point sources.
  - Illuminate objects with parallel rays of light.
  - In calculation, the location of the source is replaced by the direction of the parallel rays.
Light and matter

• Light and surface
  – We see the color of the light reflected from the surface toward our eyes
  – Human’s perception vs. computer image
Light and matter

- **Light-material interaction**
  - When light strikes a surface, some of it is absorbed, some of it is reflected, and some of it is transmitted through the material.
  - Specular surfaces, diffuse surfaces, translucent surfaces.
Light and matter

Light incidents at a point will be reflected, absorbed, scattered, and transmitted.
Phong Reflection Model

- A simple model that compromises between acceptable results and processing cost.
  - Computes light intensity at a surface point, assuming point light source.
  - Considers only local lighting reflected and regards global lighting as an ambient term.

- Other reflection models
  - Cook and Torrance model
  - Warn’s method for modeling illumination source
  - BRDF/BSSRDF model
Phong Reflection Model

-2

Approximates local lighting

Approximates global lighting

light reflected  light scattered  light absorbed  light transmitted

specular  diffuse  ambient term  Phong model
Phong Reflection Model

- An efficient computation of light-material interactions.
- Consists of three terms
  - Ambient term
  - Diffuse term
  - Specular term

\[
I = k_a L_a + \frac{1}{a + bd + cd^2} (k_d L(I \cdot n) + k_s L(r \cdot v)^\alpha)
\]
Diffuse Reflection

• A perfect diffuse scatters light equally in all directions, independent on the viewer's position
  – appears equally bright from all views

• Responsible for the color of the object
  – Example: A green object absorbs white light and reflects the green component of the light

• It is characterized by rough surfaces
  – Light rays with slightly different angles are reflected at markedly different angles.

Perfectly diffuse surfaces are so rough that there is no preferred angle of reflection.
Diffuse Term

- Intensity of the diffuse reflected light
  \[ I_d = K_d I_i \cos \theta \]
  
  - \( I_i \): the intensity of the light source
  - \( \theta \): angle between the surface normal \( n \) and the vector \( l \)
    from the surface point to the point light source
  - \( K_d \): wavelength-dependent diffuse reflectivity, \( 0 < K_d < 1 \)

- Note that
  \[ \cos \theta = l \cdot n \]
  where \( l \) is a unit vector pointing from the surface point to the light source
Diffuse Term $-2$
Specular Reflection

- For a perfect reflector, reflected light is seen only in the direction of \( R \)
  - In practice, specular reflection is not perfect and reflected light can be seen for viewing directions close to the direction of \( R \) (and produce highlight)
Specular Reflection

- A highlight is a surface area from which the viewer can see the specular reflection.
- Reflection range (or highlight area) depends on the surface roughness.
- The color of the specular reflection is the color of the light source.
  - For example:
    - A green surface illuminated by white light has color green from diffuse reflection and color white from specular reflection.
Specular Term

\[ I_s = I_i K_s \cos^{n_s} \phi = I_i K_s (R \cdot V)^{n_s} \]

- \( n_s \) simulates the surface roughness:
  - perfect specular \( H-A : 0 \), \( n_s : \infty \)
  - very glossy surface \( H-A : \) smaller, \( n_s : \) large
  - less glossy surface \( H-A : \) large, \( n_s : \) smaller

- \( K_s \) is the specular reflection coefficient which depends on surface property and \( \theta \).
  - Transparent material
    - \( K_s(\theta) \) is 1 when \( \theta = 90 \) and decrease when \( \theta \) decreases.
  - Opaque materials
    - \( K_s(\theta) \) is nearly constant for any \( \theta \)
Specular Term -2

larger $n$

smaller $n$
Ambient Term

• For surfaces that are visible from the view point but invisible from the point light source, should we render black?
  – No, made visible by ambient light since they are illuminated by other objects
  – Ambient term is the result of multiple reflection from walls and objects, and is incident on a surface from all directions.
Ambient Term

- It can be approximated by a constant

\[ I = I_a k_a \]

where \( I_a \) is a user-specified constant that approximates the global lighting effect and \( k_a \) is the ambient reflection coefficient
Phong Model

\[ l(r,g,b) = l_a K_a(r,g,b) + l_i \left[ k_d(r,g,b) [L \cdot N] + K_s \cos^{ns} \phi \right] \]

or

\[ l_r = l_a K_{ar} + l_i \left[ k_{dr} [L \cdot N] + K_s \cos^{ns} \phi \right] \]
\[ l_g = l_a K_{ag} + l_i \left[ k_{dg} [L \cdot N] + K_s \cos^{ns} \phi \right] \]
\[ l_b = l_a K_{ab} + l_i \left[ k_{db} [L \cdot N] + K_s \cos^{ns} \phi \right] \]

- **Control the color of the objects by appropriate setting of the diffuse reflection coefficients.**
- **Color of light sources are controlled by the specular reflection coefficients.**
Phong Model

The image shows a Phong model of a teapot and cups, with three different stages of shading:

1. Diffuse shading
2. Specular shading
3. Final shaded image

The equation symbolizes the combination of these stages to achieve the final shaded model.
Phong Model

- **Multiple light sources**

\[
I_r = I_a K_a + \sum I_i [K_d (L_n \cdot N)] + K_s \cos^{ns} \phi_n
\]

- **The intensity at P is a function of the viewing vector V due to specular reflection.**

i.e.

- Ambient term (constant)
- Diffuse term (independent on V)
- Specular term (dependent on V)
Note on Computing vectors

• Normal vectors
  – Triangle normal
  – Vertex normal
  – Plane
    • Plane equation
    • Three non-coplanar points
  – Implicit surfaces
    • $F(x, y, z)=0$
  – Parametric surfaces
    • $P(u, v)$

• Angle of reflection
Shading

• Apply a "point" reflection model over the entire surface of an object is impossible

• Polygon shading methods
  – **Constant shading**
  – **Gouraud shading**
    • Pixel’s intensity is obtained by bilinear interpolation of vertex’s intensities
    • Produces incorrect highlights on a polygon
  – **Phong shading**
    • Pixel’s normal is obtained by bilinear interpolation of vertex’s normals
    • Can give more accurate specular effects
Shading

- Flat shading
- Gouraud shading
- Phong shading

Normal interpolation
Gouraud Shading
Phong Shading
Gouraud Shading

- Compute (in world coord. system) the intensity at each vertex of the polygon by Phong model
  - $N$ is in general the surface normal at the vertex, can be approximated by the average of the normals of the adjacent polygons.
  - When polygons are clipped, the normals at clipped vertices are interpolated in world coord. or view coord.
Gouraud Shading

- Interpolate the intensities over the polygon

\[ \begin{align*}
    I_a &= \frac{1}{y_1 - y_2} \left[ I_1(y_s - y_2) + \left( I_2(y_1 - y_s) \right) \right] \\
    I_b &= \frac{1}{y_1 - y_4} \left[ I_1(y_s - y_4) + \left( I_4(y_1 - y_s) \right) \right] \\
    I_s &= \frac{1}{x_b - x_a} \left[ I_a(x_b - x_s) + \left( I_b(x_s - x_a) \right) \right]
\end{align*} \]

- Incremental calculation
  - \( \Delta x \): incremental change along a scan line
  - \( \Delta I_s \): change in intensity

\[ \begin{align*}
    \Delta I_s &= \frac{\Delta x}{x_b - x_a} ( I_b - I_a ) \\
    I_{s,n} &= I_{s,n-1} + I_s
\end{align*} \]
Gouraud Shading

- Gouraud shading cannot correctly generate highlights
Gouraud Shading

Smaller polygons for better highlights
Phong Shading

- Interpolate normals instead of intensities

  - Compute the vertex normals
  - Interpolate the normal for interior point (in screen coord. system)
  - Compute the intensity at interior point (in world coord. System, or view coord. system)

- Phong shading is computationally more expensive than Gouraud shading
Phong Shading - 2

\[ N_a = \frac{1}{y_1 - y_2} \left[ N_1(y_s - y_2) + (N_2(y_1 - y_s)) \right] \]

\[ N_b = \frac{1}{y_1 - y_4} \left[ N_1(y_s - y_4) + (N_4(y_1 - y_s)) \right] \]

\[ N_s = \frac{1}{x_b - x_a} \left[ N_a(x_b - x_s) + (N_b(x_s - x_a)) \right] \]

- **Incremental calculation**
  - \( \Delta x \): incremental change along a scan line
  - \( \Delta I_s \): change in intensity

\[
\begin{cases}
\Delta N_S = \Delta x \frac{N_b - N_a}{x_b - x_a} \\
N_{S,N} = N_{S,N-1} + \Delta N_S
\end{cases}
\]
Phong Shading

$K_s$: increases from left, $N_s$: increases from top; $K_a$ and $K_d$: constant
Rendering Pipeline

- Set up a polygonal mesh for the scene.
- Culling back-facing polygons.
  (in WC, or view coord, or screen coord.)
- Apply viewing transformation and projection.
- Clip polygons against view volume.
  (in 3D screen space or homogeneous space)
- Scan convert polygons.
- Shade pixels by incremental shading methods.
- Apply hidden surface removal: Z-buffer.
Z-Buffer HSR - 1

- A duplicate memory in the size of frame buffer holds the depths for each pixel in the frame buffer.
  - A current point \((i, j)\) holds the smallest z-value so far encountered. During the processing of a polygon, we either write the intensity of \((i, j)\) into the frame buffer or not, depending on whether the depth \(z < Z\)-Buffer\((i, j)\)
Z-Buffer HSR - 2
Z-Buffer HSR
Z-Buffer HSR - 4

Figure 3-20 A triangle pierces a square.

Comp. Graphics (U), Chap 2 Global View 106 CGGM Lab., CS Dept., NCTU Jung Hong Chuang
Z-Buffer HSR
Z-buffer HSR - 6

• Advantage of Z-buffer method
  – Easy to implement and usually hardware supported.
  – No upward limit on the scene complexity.

• Disadvantages
  – Space consuming is very high.
    • 23-32 bits per pixel is usually deemed sufficient.
  – The scene has to be scaled to this fixed range of z so accuracy within the scene is maximized.
/* polygon by polygon basis, with polygon in any convenient order */
for all \((x,y)\) do 
\[ \text{Z-buffer}(x,y) := \text{maximum-depth}; \]
for each polygon do
\[ \text{for } y = y_{\text{min}} \text{ to } y_{\text{max}} \text{ do} \]
/* scan convert one scan-line at a time */
begin
\[ \text{calculate } x, z, \text{ and } l \text{ for left/right end points:} \]
\[ X_{\text{left}}, X_{\text{right}}, Z_{\text{left}}, Z_{\text{right}}, l_{\text{left}}, l_{\text{right}} \]
\[ \text{for } x = x_{\text{left}} \text{ to } x_{\text{right}} \text{ do} \]
\[ \text{linearly interpolate } z \& \ l \]
\[ \text{if } z < \text{Z-buffer}(x,y) \text{ then} \]
\[ \text{Z-buffer}(x, y) := z; \]
\[ \text{Frame-buffer}(x, y) := l; \]
end
Rendering Pipeline
Detail Steps

1. Per-Vertex
   Operations
2. Primitive
   Assembly
3. Clip
   Viewport
   Cull
4. Fragment
   Processing
5. Per
   Fragment
   Operations

App. Memory

(Pixels)

(Geometry)

Pixel
Transfer

Pixel
Pack

Pixel
Unpack

Texture
Memory

Frame
Buffer

Read
Control

Vertices

Fragments

Textures

Pixel Groups
Global Illumination, texture Mapping, multi-pass rendering

Pipeline supported:
- Texture mapping techniques
  - Multi-texturing, bump map, displacement map, light map, normal map, environment map, shadow map...
- Multi-pass rendering
- Shadow Computing
  - Improve depth cues
  - View-independent
  - Hard shadow vs. soft shadow
Global Illumination, texture Mapping, multi-pass rendering

Not supported by fixed pipeline

- Global illumination
  - Ray tracing, Monte Carlo ray tracing, Photon maps
    - Ray tracing: Good for specular scenes
    - Monte Carlo ray tracing, Photon maps: good for many types of surface
    - View-dependent, naturally for dynamic scenes, can be GPU accelerated

- Radiosity
  - Good for diffuse scenes
  - View-independent, Practically useful for static navigation
Examples for Different Methods

- Gouraud shading
- Phong shading
- Texture mapping
- Environment mapping & shadowing
- Ray tracing
- Radiosity
Radiance Example
Gouraud Shading

[Image of a jet flying over a cloudy sky]
Phong Shading with Bump Map texture
Texture mapping
Multitexturing

Texture map

Light map
3D Texture Mapping

Surfaces modulated by three-dimensional texture fields. (left) Cube of cubes: the centre point of each cube is used to obtain a colour from the RGB cube. (below) Wood grain: harmonic functions (Fourier synthesis). (bottom left) Marble texture: equivalent to harmonic function plus noise. (bottom right) Using a three-dimensional noise function.
Bump Mapping
Bump Mapping
Bump mapping

Bump map + shadowing

md2shader demo!!
Displacement mapping
Normal map

Original

Normal maps

$M^5$

$M^5 + NM$
Multi-pass rendering

Figure 8: An example of a high quality image generated at 60 Hz using multipass rendering techniques
Multi-pass rendering
Gloss mapping

Gloss Map Example

Shiny puddles

Specular lighting contribution (per-vertex lighting) \times \text{(modulate)} \quad \text{Gloss map texture} \quad \text{Diffuse lighting contribution (per-vertex lighting)} \quad = \quad \text{Final combined result}

Multi-pass rendering
Light mapping

Without Light map

[From Channa’s article]
Multi-pass rendering
Light mapping

The white rhombus type object (on the right hand side) represents a point light source.

[From Channa’s article]
Multi-pass rendering
Light mapping

Dynamic light map [From 3D Games by Watt et al.]
Complex shadow animation: a door opening sequence
Multi-pass rendering
Light mapping

Dynamic light map
Complex shadow animation: a door opening sequence
Multi-pass rendering

Light mapping

Another example

Texture map without filtering
[From 3D Games, by Watt et al.]

Texture map with mipmapping
Multi-pass rendering
Light mapping

Light map without filtering

Light map with linear filtering
Multi-pass rendering
Light mapping

With mipmapped texture and filtered light map

Fog map with linear filtering

Comp. Graphics (U), Chap 2  Global View  134

CGGM Lab., CS Dept., NCTU  Jung Hong Chuang
Multi-pass rendering
Light mapping

With mipmapped texture and linear filtered light, and fog map.
Multi-pass rendering
Light mapping

Another example

Without Light map
Multi-pass rendering
Light mapping

With light map
Environment mapping

A room scene
Environment mapping
A Comparison to Ray Tracing
Multi-pass shadowing
Properties

Shadow provides information about the relative positions of objects
[From Real-Time Soft Shadow]

Comp. Graphics (U), Chap 2 Global View 140 CGGM Lab., CS Dept., NCTU Jung Hong Chuang
Multi-pass shadowing
Properties

Shadow provides information about the geometry of the occluder
[From Real-Time Soft Shadow]
Multi-pass shadowing Properties -4

Shadow provides information about the geometry of the receiver
[From Real-Time Soft Shadow]
Hard shadows

Three colored lights. Diffuse/specular bump mapped animated characters with shadows. 34 fps on GeForce4 Ti 4600; 80+ fps for one light.
Soft shadows

Cluster of 12 dim lights approximating an area light source. Generates a soft shadow effect; careful about banding. 8 fps on GeForce4 Ti 4600.

The cluster of point lights.