Shading & Texturing
Overview

- **Scan conversion**
  - Figure out which pixels to draw/fill

- **Shading**
  - Determine a color for each filled pixel

- **Texture Mapping**
  - Describe shading variation within polygon interiors
Shading

- How do we choose a color for each filled pixel?
  - Each illumination calculation for a ray from the eyepoint through the view plane provides a radiance sample
    - How do we choose where to place samples?
    - How do we filter samples to reconstruct image?

Emphasis on methods that can be implemented in hardware
Ray Casting

- Simplest shading approach is to perform independent lighting calculation for every pixel
- When is this unnecessary?

\[ I = I_E + K_A I_{AL} + \sum_i (K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i) \]
Polygon Shading

- Can take advantage of spatial coherence
  - Illumination calculations for pixels covered by same primitive are related to each other

- Algorithms:
  - Flat Shading
  - Gouraud Shading
  - Phong Shading
Flat Shading

- What if a faceted object is illuminated only by directional light sources and is either diffuse or viewed from infinitely far away?
Flat Shading

- One illumination calculation per polygon

- Assign all pixels inside each polygon the same color
Flat Shading

- Objects look like they are composed of polygons
  - OK for polyhedral objects
  - Not so good for ones with smooth surfaces
Gouraud Shading

- What if smooth surface is represented by polygonal mesh with a normal at each vertex?

\[
I = I_E + K_A I_{AL} + \sum_{i} (K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i)
\]
Gouraud Shading

- Step 1: One lighting calculation per vertex
  - Assign pixels inside polygon by interpolating colors computed at vertices
Gouraud Shading

- Step 2: Bilinearly interpolate colors at vertices down and across scan lines

\[
A = \alpha l_1 + (1-\alpha) l_3
\]

\[
B = \beta l_2 + (1-\beta) l_3
\]

\[
I = \varphi A + (1-\varphi) B
\]
Gouraud Shading

- Smooth shading over adjacent polygons
- Curved surfaces
- Illumination highlights
- Soft shadows

Mesh with shared normals at vertices
Gouraud Shading

- Produces smoothly shaded polygonal mesh
  - Piecewise linear approximation
  - Need fine mesh to capture subtle lighting effects

Flat Shading

Gouraud Shading
Gouraud Shading

- Produces smoothly shaded polygonal mesh
  - Piecewise linear approximation
  - Need fine mesh to capture subtle lighting effects

Poor behavior of specular light

Same sphere – high polygon count
Phong Shading

- What if polygonal mesh is too coarse to capture illumination effects in polygon interiors?

\[
I = I_E + K_A I_{AL} + \sum_i (K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i)
\]
Phong Shading

- One lighting calculation per pixel
- Approximate surface normals for points inside polygons by bilinear interpolation of normals from vertices
Phong Shading

- Bilinearly interpolate surface normals at vertices down and across scan lines.

\[ A = \alpha N_1 + (1-\alpha)N_3 \]
\[ B = \beta N_2 + (1-\beta)N_3 \]
\[ I = \phi A + (1-\phi)B \]
Guarard Vs. Phong Shading

- Guarard: calculate color in vertices and interpolate pixel colors
- Phong: interpolate normals and calculate colors at each pixel
Polygon Shading Algorithms

Wireframe

Flat

Gouraud

Phong
Shading Issues

- Problems with interpolated shading:
  - Polygonal silhouettes
  - Perspective distortion
  - Orientation dependence (due to bilinear interpolation)
  - Problems at T-vertices
  - Problems computing shared vertex normals
Overview

- **Scan conversion**
  - Figure out which pixels to draw/fill

- **Shading**
  - Determine a color for each filled pixel

- **Texture Mapping**
  - Describe shading variation within polygon interiors
3D Rendering Pipeline

3D Geometric Primitives

- Modeling Transformation
  - Transform into 3D world coordinate system

- Lighting
  - Illuminate according to lighting and reflectance

- Viewing Transformation
  - Transform into 3D camera coordinate system

- Projection Transformation
  - Transform into 2D camera coordinate system

- Clipping
  - Clip primitives outside camera’s view

- Scan Conversion
  - Draw pixels (includes texturing, hidden surface, etc.)

Image
Surface Textures

Polygonal model + surface texture
Surface Textures

- Add visual detail to surfaces of 3D objects

Example: Google Earth
Textures

- Describe color variation in interior of 3D polygon
  - When scan converting a polygon, vary pixel colors according to values fetched from a texture, (not the vertexes color)
Overview

- Texture mapping methods
  - Mapping
  - Filtering
  - Parameterization

- Texture mapping applications
  - Modulation textures
  - Illumination mapping
  - Bump mapping
  - Environment mapping
  - Image-based rendering
Texture Mapping

- **Steps:**
  - Define texture
  - Specify mapping from texture to surface
  - Lookup texture values during scan conversion

![Diagram showing texture mapping between the texture coordinate system and the modeling coordinate system.](image-url)
Texture Mapping

- When scan convert, map from ...
  - screen coordinate system \((s,t)\) to
  - modeling coordinate system \((x,y)\) to
  - texture coordinates \((u,v)\)

<table>
<thead>
<tr>
<th>Texture Coordinate System</th>
<th>Modeling Coordinate System</th>
<th>Screen Coordinate System</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s)</td>
<td>(t)</td>
<td>(x)</td>
</tr>
<tr>
<td>(u)</td>
<td>(v)</td>
<td>(y)</td>
</tr>
</tbody>
</table>

\((0,0), (1,0), (1,1), (0,1)\)
Texture Mapping

- Texture mapping is in fact a 2D projective transformation
  - texture coordinate system: (u,v) to
  - screen coordinate system (t,s)
image coordinates
$u$ and $v$ range from 0 to 1

$P_{ij} = \left( r \cos \frac{2\pi i}{10}, h \frac{j}{5}, r \sin \frac{2\pi i}{10} \right)$

$u$-coordinate of vertex $P_{ij}$: $i/10$

$v$-coordinate of vertex $P_{ij}$: $j/5$
Texture Mapping

- During scan conversion we interpolate texture coordinates down/across scan lines.

- **Affine texture mapping.** Noticeable discontinuity between adjacent triangles when these triangles are at an angle to the plane of the screen.
**Perspective correct:** account for vertices’ positions in 3D space, rather than simply interpolating a 2D triangle

Linear interpolation of texture coordinates

\[ u_\alpha = (1 - \alpha)u_0 + \alpha u_1 \]

Correct interpolation with perspective divide

\[ u_\alpha = \frac{(1 - \alpha)\frac{u_0}{z_0} + \alpha\frac{u_1}{z_1}}{(1 - \alpha)\frac{1}{z_0} + \alpha\frac{1}{z_1}} \]

Dividing each texture coordinate is like projecting it
• closer to viewer, difference from pixel to pixel between texture coordinates is smaller (stretching the texture wider)
• parts farther away, difference is larger (compressing the texture)
Aliasing?

- Must sample texture to determine color at each pixel in image
Texture Aliasing

- Image mapped onto polygon
- Occur when screen resolution differs from texture resolution
- Magnification aliasing
  - Screen resolution finer than texture resolution
  - Multiple pixels per texel
- Minification aliasing
  - Screen resolution coarser than texture resolution
  - Multiple texels per pixel
Minification & Magnification

Minification:
More than one texel per pixel

Magnification:
Less than one texel per pixel
Aliasing Problem

Point sampling

After area filtering
Magnification Filtering

- Nearest neighbor

- Linear interpolation
Minification Filtering

- Multiple texels per pixel
- Potential for aliasing since texture signal bandwidth greater than framebuffer
- Box filtering requires averaging of texels
- Precomputation of pre-filtered images
  - MIP Mapping
  - Summed Area Tables
MIP Mapping (Lance Williams, 1983)

- “Multum In Parvo” (much in little)
- Create a resolution pyramid of textures
  - Repeatedly subsample texture at half resolution
  - Until single pixel
  - Need extra storage space
MIP Mapping

- Accessing
  - Use texture resolution closest to screen resolution
  - Or interpolate between two closest resolutions
  - Fast, easy for hardware
Summed-area Tables (Frank Crow, 1984)

- A 2D table the size of the texture.
- At entry \((i,j)\), store the sum of all texels in the rectangle defined by \((0,0)\) and \((i,j)\).
- Sum of all texels is easily obtained from:
  \[
  \text{area} = A - B - C + D
  \]
Summed-area Tables

- At each entry keep sum of all values from top left to bottom right

\[ T(i, j) = \sum_{i'<i, j'<j} I(i', j') \]

- Can be computed in one pass:

\[ T(i, j) = I(i, j) + T(i-1, j) + T(i, j-1) - T(i-1, j-1) \]
Breakdown

\[ T = T + T - T \]
Computing Any Rectangle Sum

- Sum of all values within any rectangle?
  \[
  \sum_{i_0 < i' \leq i_1, j_0 < j \leq j_1} I(i', j') = T(i_1, j_1) - T(i_0, j_1) - T(i_1, j_0) + T(i_0, j_0)
  \]

- Better ability to capture very oblique projections

- But, cannot store values in a single byte
Breakdown

\[ T_{i_0 j_0} T_{i_0 j_1} = T_{i_1 j_0} T_{i_1 j_1} - T_{i_0 j_0} T_{i_1 j_1} + T_{i_0 j_1} T_{i_1 j_0} \]
Summed Area Table (SAT)

- Determine the rectangle with the same aspect ratio as the bounding box and the same area as the pixel mapping.
Mip-maps

- Find level of the mip-map where the area of each mip-map pixel is closest to the area of the mapped pixel.
Elliptical Weighted Average (EWA) Filter

- Treat each pixel as circular, rather than square.
- Mapping of a circle is elliptical in texel space.

![Diagram of elliptical mapping](image)
Textures as Lookup Tables

- Key Idea: anything can be parametrized using “texture”
- Instead of using direct or “calculated” value, one uses the value as an index into a lookup table

\[ f(i, j) = \text{val}(i, j) \]

\[ f(i, j) = T(\text{val}(i, j)) \]
Modulation Textures

- Map texture values to scale factor

\[ I = T(s, t)(I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) S_L I_L + K_T I_T + K_S I_S) \]
modeling specular reflections

- Replace model of light to one based on a texture lookup:
- Treat eye vector as a point of the unit sphere, and use it to index into a texture-mapped sphere to look up light
- Write eye vector in polar coordinates, $(\theta, \varphi)$: $u = \theta/2\pi$, $v = \varphi/\pi + 1/2$
- Environment map: A fisheye-lens photograph taken from the center of a scene
Environment Mapping

- Map texture values to any surface material parameter
  - $K_A$
  - $K_D$
  - $K_S$
  - $K_T$
  - Coefficient $n$

$$K_T = T(s,t)$$

$$I = I_E + K_A I_A + \sum_L (K_D (N \cdot L) + K_S (V \cdot R)^n) S_L I_L + K_T I_T + K_S I_S$$
Environment Mapping

- An image-based lighting technique for approximating the appearance of a reflective surface by means of a precomputed texture image. The texture is used to store the image of the distant environment surrounding the rendered object.
Environment Stored Textures
Bump Mapping

- Map texture values to perturbations of surface normals
Bump Mapping

- assume that at each surface point $P$ we have a pair of unit vectors $t1$ and $t2$
- Map $R,G$ from $[-128,127]$ to $[-1,1]$
- Adjust normal from bump map at $P$: $\hat{n} = S(n + rt1 + gt2)$. 
Bump Mapping

Note the silhouette is smooth ➔ no real geometry!
Z-Buffer Algorithm

- Any pixel rendered holds a color and a distance from the viewer (z).
- Pixels are accumulated in a buffer using the z-buffer algorithm:
  
  Clear Z buffer (set z to infinity and store bg-color)
  For each new pixel P(i,j) rendered
  
  If the distance \( z_P \) of P is smaller than \( Z(i,j) \)
  
  store the \( z_P \) & color of P in \( Z(i,j) \)
  
  Send the buffer to display
Z-Buffer

A simple three dimensional scene

Z-buffer representation
Interactive Shadows

- A surface area that is not "visible" by the light source but is visible by the viewer appears as a shadowed area.
Shadow Maps: Z-Buffer Shadow Generation

- Uses multi-pass rendering
  - Render scene from the light source "view-point" and save the Z depth instead of the color. This is called the shadow map and indicates the closest pixel to the light
  - Render the scene as usual from the observer and save Z depth values (z-buffer).
  - Transform every pixel \((x,y,z)\) in the z-buffer to the light source coordinate space \((x',y',z')\)
  - Project \((x',y',z')\) onto the shadow map (the “light” z-buffer) to get \((x_s,y_s)\) and compare the z value at that pixel to \(z'\)
  - If they are identical, that pixel is illuminated by the light source, otherwise (\(z'\) must be larger) the original \((x,y,z)\) pixel is shadowed!
Comparing the “Z” Values

Camera Viewpoint

Light “Viewpoint”
Light's Z-buffer

Observer's Z-buffer

Observer's Image

With shadows.
3D Rendering Pipeline Done!

3D Geometric Primitives

- **Modeling Transformation**: Transform into 3D world coordinate system
- **Lighting**: Illuminate according to lighting and reflectance
- **Viewing Transformation**: Transform into 3D camera coordinate system
- **Projection Transformation**: Transform into 2D camera coordinate system
- **Clipping**: Clip primitives outside camera’s view
- **Scan Conversion**: Draw pixels (includes texturing, hidden surface, etc.)
- **Image**