3D Rasterization II

COS 426
3D Rendering Pipeline (for direct illumination)

3D Primitives

  Modeling Transformation

    Lighting

      Viewing Transformation

        Projection Transformation

          Clipping

            Viewport Transformation

              Scan Conversion

                Image
Rasterization

• Scan conversion
  ◦ Determine which pixels to fill

• Shading
  ◦ Determine a color for each filled pixel

• Texture mapping
  ◦ Describe shading variation within polygon interiors

• Visible surface determination
  ◦ Figure out which surface is front-most at every pixel
Rasterization

• Scan conversion (last time)
  ◦ Determine which pixels to fill

➢ Shading
  ◦ Determine a color for each filled pixel

• Texture mapping
  ◦ Describe shading variation within polygon interiors

• Visible surface determination
  ◦ Figure out which surface is front-most at every pixel
Shading

• How do we choose a color for each filled pixel?

Emphasis on methods that can be implemented in hardware
Ray Casting

- Simplest shading approach is to perform independent lighting calculation for every pixel.

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

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

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

- Flat Shading
- Gouraud Shading
- Phong Shading
Polygon Shading 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

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

- Flat Shading
- **Gouraud Shading**
- Phong Shading
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

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

- 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
Polygon Shading Algorithms

- Flat Shading
- Gouraud Shading
- Phong Shading
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

\[ A = \alpha N_1 + (1-\alpha)N_3 \]
\[ B = \beta N_2 + (1-\beta)N_3 \]
\[ I = \varphi A + (1-\varphi)B \]
Polygon Shading Algorithms

Wireframe

Flat

Gouraud

Phong
Shading Issues

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

• Scan conversion
  ○ Determine which pixels to fill

• Shading
  ○ Determine a color for each filled pixel

➤ Texture mapping
  ○ Describe shading variation within polygon interiors

• Visible surface determination
  ○ Figure out which surface is front-most at every pixel
Textures

- Describe color variation in interior of 3D polygon
  - When scan converting a polygon, vary pixel colors according to values fetched from a texture image
Surface Textures

- Add visual detail to surfaces of 3D objects

Polygonal model

With surface texture
Surface Textures

- Add visual detail to surfaces of 3D objects
3D Rendering Pipeline (for direct illumination)

3D Primitives

- Modeling Transformation
  - 3D Modeling Coordinates

- Lighting
  - 3D World Coordinates

- Viewing Transformation
  - 3D World Coordinates

- Projection Transformation
  - 3D Camera Coordinates

- Clipping
  - 2D Screen Coordinates

- Viewport Transformation
  - 2D Screen Coordinates

- Scan Conversion
  - 2D Image Coordinates

- Image

Texture mapping
Texture Mapping Overview

• Texture mapping methods
  ◦ Mapping
  ◦ Filtering
  ◦ Parameterization

• Texture mapping applications
  ◦ Modulation textures
  ◦ Illumination mapping
  ◦ Bump mapping
  ◦ Environment mapping
  ◦ Image-based rendering
  ◦ Non-photorealistic rendering
Texture Mapping

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

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

- Texture mapping is a 2D projective transformation
  - texture coordinate system: \((t,s)\) to
  - image coordinate system \((x,y)\)
Texture Mapping

• Scan conversion
  ◦ Interpolate texture coordinates down/across scan lines
  ◦ Distortion due to bilinear interpolation approximation
    » Cut polygons into smaller ones, or
    » Perspective divide at each pixel
Texture Mapping

Linear interpolation of texture coordinates

Correct interpolation with perspective divide

Hill Figure 8.42
Texture Mapping Overview

• Texture mapping methods
  ◦ Mapping
  ◦ Filtering
  ◦ Parameterization

• Texture mapping applications
  ◦ Modulation textures
  ◦ Illumination mapping
  ◦ Bump mapping
  ◦ Environment mapping
  ◦ Image-based rendering
  ◦ Non-photorealistic rendering
Texture Filtering

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

• Aliasing is a problem

Point sampling  Area filtering

Angel Figure 9.5
Texture Filtering

• Ideally, use elliptically shaped convolution filters

In practice, use rectangles
Texture Filtering

• Size of filter depends on projective warp
  ◦ Can prefiltering images
    » Mip maps
    » Summed area tables
Mip Maps

- Keep textures prefiltered at multiple resolutions
  - For each pixel, linearly interpolate between two closest levels (e.g., trilinear filtering)
  - Fast, easy for hardware
Summed-area tables

- At each texel keep sum of all values down & right
  - To compute sum of all values within a rectangle, simply subtract two entries
  - Better ability to capture very oblique projections
  - But, cannot store values in a single byte
Texture Mapping Overview

• Texture mapping methods
  ○ Mapping
  ○ Filtering
  ○ Parameterization

• Texture mapping applications
  ○ Modulation textures
  ○ Illumination mapping
  ○ Bump mapping
  ○ Environment mapping
  ○ Image-based rendering
  ○ Non-photorealistic rendering
Parameterization

• Q: How do we decide where on the geometry each color from the image should go?
Option: Varieties of projections

[Paul Bourke]
Option: unfold the surface
Option: make an atlas

charts  atlas  surface

[Sander2001]
Texture Mapping Overview

• Texture mapping methods
  ◦ Mapping
  ◦ Filtering
  ◦ Parameterization

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

Map texture values to scale factor

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

Map texture values to surface material parameter

- $K_A$
- $K_D$
- $K_S$
- $K_T$
- $n$

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

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

Texture values perturb surface normals
Bump Mapping
Environment Mapping

Texture values are reflected off surface patch

H&B Figure 14.93
Image-Based Rendering

Map photographic textures to provide details for coarsely detailed polygonal model
Solid textures

Texture values indexed by 3D location (x,y,z)

• Expensive storage, or
• Compute on the fly, e.g. Perlin noise
Texture Mapping Summary

• Texture mapping methods
  ◦ Parameterization
  ◦ Mapping
  ◦ Filtering

• Texture mapping applications
  ◦ Modulation textures
  ◦ Illumination mapping
  ◦ Bump mapping
  ◦ Environment mapping
  ◦ Image-based rendering
  ◦ Volume textures
Rasterization

• Scan conversion
  ◦ Determine which pixels to fill

• Shading
  ◦ Determine a color for each filled pixel

• Texture mapping
  ◦ Describe shading variation within polygon interiors

➢ Visible surface determination
  ◦ Figure out which surface is front-most at every pixel
Visible Surface Determination

- Make sure only front-most contributes to color at every pixel
Depth sort

- “Painter’s algorithm”
  - Sort surfaces in order of decreasing maximum depth
  - Scan convert surfaces in back-to-front order, overwriting pixels
3D Rendering Pipeline

- 3D Primitives
  - 3D Modeling Coordinates
  - Modeling Transformation
    - 3D World Coordinates
    - Lighting
      - 3D World Coordinates
      - Viewing Transformation
        - 3D Camera Coordinates
        - Projection Transformation
          - 2D Screen Coordinates
          - Clipping
            - 2D Screen Coordinates
            - Viewport Transformation
              - 2D Screen Coordinates
              - Scan Conversion
                - 2D Image Coordinates
                - Image

Depth sort comments:
- $O(n \log n)$
- Better with frame coherence?
- Implemented in software
- Render every polygon
- Often use BSP-tree or static list ordering
Z-Buffer

- Color & depth of closest object for every pixel
  - Update only pixels whose depth is closer than in buffer
  - Depths are interpolated from vertices, just like colors
Z-Buffer

3D Primitives ➔ 3D Modeling Coordinates

Modeling Transformation ➔ 3D World Coordinates

Lighting ➔ 3D World Coordinates

Viewing Transformation ➔ 3D Camera Coordinates

Projection Transformation ➔ 2D Screen Coordinates

Clipping ➔ 2D Screen Coordinates

Viewport Transformation ➔ 2D Image Coordinates

Scan Conversion ➔ 2D Image Coordinates

Image

Z-Buffer

Z-buffer comments

+ Polygons rasterized in any order
+ Process one polygon at a time
+ Suitable for hardware pipeline
- Requires extra memory for z-buffer
- Subject to aliasing (A-buffer)
  ○ Commonly in hardware
Hidden Surface Removal Algorithms

[Figure 29. Characterization of ten opaque-object algorithms & Comparison of the algorithms.]

[Sutherland '74]
Rasterization Summary

• Scan conversion
  ○ Sweep-line algorithm

• Shading algorithms
  ○ Flat, Gouraud

• Texture mapping
  ○ Mipmaps

• Visibiliity determination
  ○ Z-buffer

This is all in hardware
Actually ...

- Graphics hardware is programmable
Trend ...

- GPU is general-purpose parallel computer