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

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

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

\[
I = I_E + K_A I_{AL} + \sum_i \left( K_D (\mathbf{N} \cdot \mathbf{L}_i) I_i + K_S (\mathbf{V} \cdot \mathbf{R}_i)^n I_i \right)
\]
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

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

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

\[
A = \alpha I_1 + (1-\alpha)I_3
\]

\[
B = \beta I_2 + (1-\beta)I_3
\]

\[
I = \varphi A + (1-\varphi)B
\]
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 reflectance model)
Phong Shading

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

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

\[ A = \alpha N_1 + (1-\alpha)N_3 \]

\[ B = \beta N_2 + (1-\beta)N_3 \]

\[ I = \phi A + (1-\phi)B \]
Shading Issues

• Problems with interpolated shading:
  ○ Polygonal silhouettes
  ○ Perspective distortion (due to screen-space interpolation)
  ○ Problems computing shared vertex normals
  ○ Problems at T-junctions
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

Angel Figure 9.3
Surface Textures

- Add visual detail to surfaces of 3D objects
Surface Textures

- Add visual detail to surfaces of 3D objects

[Daren Horley]
Texture Mapping Overview

• Texture mapping stages
  ◦ Parameterization
  ◦ Mapping
  ◦ Filtering

• 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
  - Look up texture values during scan conversion
Texture Mapping

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

- Texture mapping is a 2D projective transformation
  - texture coordinate system: \((s,t)\) 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 Filtering

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

- Aliasing is a problem

Point sampling

Area filtering
Texture Filtering

- Ideally, use elliptically shaped convolution filters

In practice, use rectangles or squares
Texture Filtering

- Size of filter depends on projective warp
  - Compute prefiltered images to avoid run-time cost
    - Mipmaps
    - Summed area tables

Angel Figure 9.14
Mipmaps

- Keep textures prefiltred at multiple resolutions
  - Usually powers of 2
  - For each pixel, linearly interpolate between two closest levels (i.e., 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 combine four entries: $S_1 - S_2 - S_3 + S_4$
  - Better ability to capture oblique projections, but still not perfect
Texture Mapping Overview

• Texture mapping stages
  ○ Parameterization
  ○ Mapping
  ○ Filtering

• 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: function gives projection
Option: unfold the surface
Option: make an atlas

charts  atlas  surface

[Sander2001]
Texture Mapping Overview

• Texture mapping stages
  ◦ Parameterization
  ◦ Mapping
  ◦ Filtering

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

Texture values scale result of lighting calculation

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

Map texture values to surface material parameter

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

$$I = I_E + K_A I_A + \sum_L \left( K_D(s,t)(N \cdot L) + K_S(V \cdot R)^n \right) S_L I_L + K_T I_T + K_S I_S$$
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 \(\rightarrow\)
Texture Mapping Summary

• Texture mapping stages
  - 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 surface 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

Modeling Transformation

Lighting

Viewing Transformation

Projection Transformation

Clipping

Viewport Transformation

Scan Conversion

3D Modeling Coordinates

3D World Coordinates

3D World Coordinates

3D Camera Coordinates

2D Screen Coordinates

2D Screen Coordinates

2D Image Coordinates

2D Image Coordinates

Depth sort

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

Maintain color & depth of closest object per pixel

- Framebuffer now RGBAz – initialize z to far plane
- Update only pixels with depth closer than in z-buffer
- Depths are interpolated from vertices, just like colors
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

3D Primitives

Modeling Transformation

Lighting

Viewing Transformation

Projection Transformation

Clipping

Viewport Transformation

Scan Conversion

Image

3D Modeling Coordinates

3D World Coordinates

3D Camera Coordinates

2D Screen Coordinates

2D Image Coordinates

2D Screen Coordinates

2D Image Coordinates

3D World Coordinates

Z-Buffer
Hidden Surface Removal Algorithms

Figure 29. Characterization of ten opaque-object algorithms: A Comparison of the algorithms.

[Sutherland '74]
Rasterization Summary

• Scan conversion
  ◦ Sweep-line algorithm

• Shading algorithms
  ◦ Flat, Gouraud

• Texture mapping
  ◦ Mipmaps

• Visibility determination
  ◦ Z-buffer

This is all in hardware
GPU Architecture

Vertex Processing

Texture and Fragment Processing

Z-Compare and Blend

GeForce 6 Series Architecture
Actually …

- Graphics hardware is programmable

Device-level APIs

Applications
Using DirectX
- HLSL
  Compute Shaders

Applications
Using OpenCL
- OpenCL C
  Compute Kernels

Applications
Using the
CUDA Driver API
- C for CUDA
  Compute Kernels

Applications
Using C, C++, Fortran,
Java, Python, ...
- C for CUDA
  Compute Functions

Language Integration

DirectX
Compute

OpenCL
Driver

CUDA Driver

CUDA Support in OS Kernel

CUDA Parallel Compute Engines
inside NVIDIA GPUs

PTX (ISA)

www.nvidia.com/cuda
Trend ...

- GPU is general-purpose parallel computer

www.nvidia.com/cuda