Scan Conversion

Adam Finkelstein & Tim Weyrich
Princeton University
COS 426, Spring 2008

Overview

- Scan conversion
  - Figure out which pixels to fill
- Shading
  - Determine a color for every filled pixel
- Texture mapping
  - Describe shading variation within polygon interiors
- Visible surface determination
  - Figure out which surface is front-most at every pixel

Scan Conversion

- Render an image of a geometric primitive by setting pixel colors

  ```c
  void SetPixel(int x, int y, Color rgba)
  ```

- Example: Filling the inside of a triangle

Triangle Scan Conversion

- Properties of a good algorithm
  - Symmetric
  - Straight edges
  - Antialiased edges
  - No cracks between adjacent primitives
  - MUST BE FAST!
Triangle Scan Conversion

- Properties of a good algorithm
  - Symmetric
  - Straight edges
  - Antialiased edges
  - No cracks between adjacent primitives
  - MUST BE FAST!

Simple Algorithm

- Color all pixels inside triangle

```c
void ScanTriangle(Triangle T, Color rgba){
    for each pixel P at (x,y){
        if (Inside(T, P))
            SetPixel(x, y, rgba);
    }
}
```

Inside Triangle Test

- A point is inside a triangle if it is in the positive halfspace of all three boundary lines
  - Triangle vertices are ordered counter-clockwise
  - Point must be on the left side of every boundary line

```c
Boolean Inside(Triangle T, Point P) {
    for each boundary line L of T {
        Scalar d = L.a*P.x + L.b*P.y + L.c;
        if (d < 0.0) return FALSE;
    }
    return TRUE;
}
```

Simple Algorithm

- What is bad about this algorithm?

```c
void ScanTriangle(Triangle T, Color rgba){
    for each pixel P at (x,y){
        if (Inside(T, P))
            SetPixel(x, y, rgba);
    }
}
```

Triangle Sweep-Line Algorithm

- Take advantage of spatial coherence
  - Compute which pixels are inside using horizontal spans
  - Process horizontal spans in scan-line order
- Take advantage of edge linearity
  - Use edge slopes to update coordinates incrementally
Triangle Sweep-Line Algorithm

```c
void ScanTriangle(Triangle T, Color rgba){
    for each edge pair {
        initialize x_L, x_R;
        compute dx_L/dy_L and dx_R/dy_R;
        for each scanline at y
            for (int x = x_L; x <= x_R; x++)
                SetPixel(x, y, rgba);
        x_L += dx_L/dy_L;
        x_R += dx_R/dy_R;
    }
}
```

Historical note:
Bresenham’s Algorithm
integer-only version
of line calculation
(good for hardware)

Polygon Scan Conversion

- Fill pixels inside a polygon
  - Triangle
  - Quadrilateral
  - Convex
  - Star-shaped
  - Concave
  - Self-intersecting
  - Holes

What problems do we encounter with arbitrary polygons?

![Convex Polygon](image1.png)
![Concave Polygon](image2.png)

Inside Polygon Rule

- What is a good rule for which pixels are inside?
  - Concave
  - Self-Intersecting
  - With Holes

Inside Polygon Rule

- Odd-parity rule
  - Any ray from P to infinity crosses odd number of edges
    - Concave
    - Self-Intersecting
    - With Holes
  - What if you hit a vertex?

Polygon Sweep-Line Algorithm

- Incremental algorithm to find spans, and determine insideness with odd parity rule
  - Takes advantage of scanline coherence

![Triangle](image3.png)
![Polygon](image4.png)
Polygon Sweep-Line Algorithm

```c
void ScanPolygon(Triangle T, Color rgba){
    sort edges by maxy
    make empty "active edge list"
    for each scanline (top-to-bottom) {
        insert/remove edges from "active edge list"
        update x coordinate of every active edge
        sort active edges by x coordinate
        for each pair of active edges (left-to-right)
            SetPixels(x_i, x_{i+1}, y, rgba);
    }
}
```

Hardware Scan Conversion

- Convert everything into triangles
  - Scan convert the triangles

Overview

- Scan conversion
  - Figure out 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

\[ I = I_E + K_A I_{Alph} + \sum_i (K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^s I_i) \]

Polygon Shading

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

\[ I = I_E + K_A I_{Alph} + \sum_i (K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^s I_i) \]
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

\[ I = I_d + K_d I_d + \sum_i (K_d (N \cdot L_i) I_i + K_i (V \cdot R_i) I_i) \]

- 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?

\[ I = I_D + K_1 I_L + \sum \{K_D (N \cdot L_i) I_i + K_A (V \cdot R_i) I_i\} \]

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

\[ A = \alpha I_1 + (1-\alpha) I_2 \]
\[ B = \beta I_2 + (1-\beta) I_3 \]
\[ I = \phi A + (1-\phi) B \]

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?

\[ I = I_0 + K_d I_d + \sum (K_D (N \cdot L) I_t + K_s (V \cdot R)^* I_d) \]

- Approximate surface normals for points inside polygons by bilinear interpolation of normals from vertices

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

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

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

Overview

- Scan conversion
  - Figure out 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

![Texture Diagram](image)

*Angel Figure 9.3*

### Surface Textures

- Add visual detail to surfaces of 3D objects

![Surface Textures Image](image)

[Daren Horley]

### 3D Rendering Pipeline (for direct illumination)

3D Rendering Pipeline:
- 3D Primitives
- Modeling Transformation
- 3D Modeling Coordinates
- 3D World Coordinates
- Lighting
- 3D World Coordinates
- Viewing Transformation
- 3D Camera Coordinates
- Projection Transformation
- 3D Screen Coordinates
- Clipping
- 3D Image Coordinates
- Scan Conversion
- 2D Image Coordinates
- 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 Diagram](image)
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 is a 2D projective transformation
  - texture coordinate system: (t,s) to
  - image coordinate system (x,y)

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

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

Angel Figure 9.5

Texture Filtering

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

Magnification  Minification

Angel Figure 9.14

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

Angel Figure 9.14

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

Angel Figure 9.14

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

geometry + image = texture map

• 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

[Piponi2000]

Option: make an atlas

charts atlas surface

[Sander2001]

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

$\mathcal{I} = T(s,t)\mathcal{I}_t + K_d\mathcal{I}_d + \sum_{i=1}^{n} (K_s(N \cdot L) + K_a(V \cdot R))s_i I_i + K_r I_i + K_i I_i$
Illumination Mapping

Map texture values to surface material parameter

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

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

$$I = I_e + K_a I_s + \sum_i (K_d (N \cdot L) + K_s (F \cdot R)^s) S_i I_s + K_r I_r + K_g I_g$$

Bump Mapping

Texture values perturb surface normals

H&B Figure 14.100

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

H&B Figure 14.93
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

Overview

- Scan conversion
  - Figure out 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

3D Rendering Pipeline

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

Depth sort

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

Scan-Line Algorithm

- For each scan line, construct and sort spans

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

Example span

- Example scan line
  - \(z=1.8\)
  - \(z=3.8\)
Scan-Line Algorithm

- For each scan line, construct and sort spans
  - Sort by depths within each scan line

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

Hidden Surface Removal Algorithms

Summary

- Scan conversion
  - Sweep-line algorithm
- Shading algorithms
  - Flat, Gouraud
- Texture mapping
  - Mipmaps
- Visibility determination
  - Z-buffer

This is all in hardware