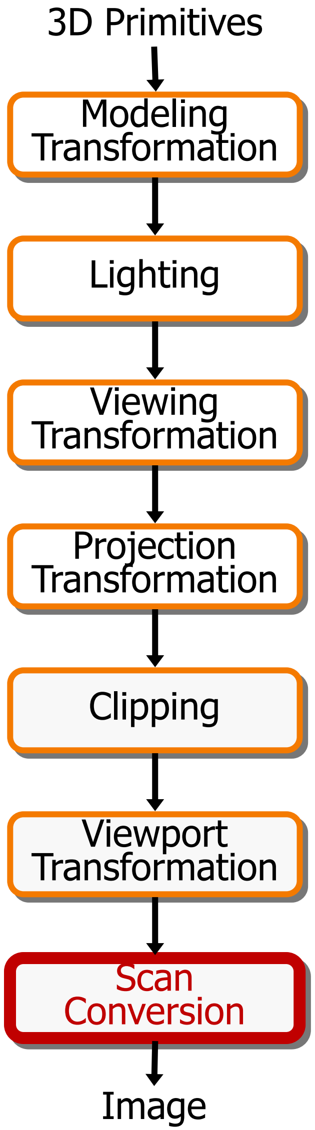
Scan Conversion

COS 426, Spring 2022

Felix Heide
Princeton University

3D Rendering Pipeline (for direct illumination)

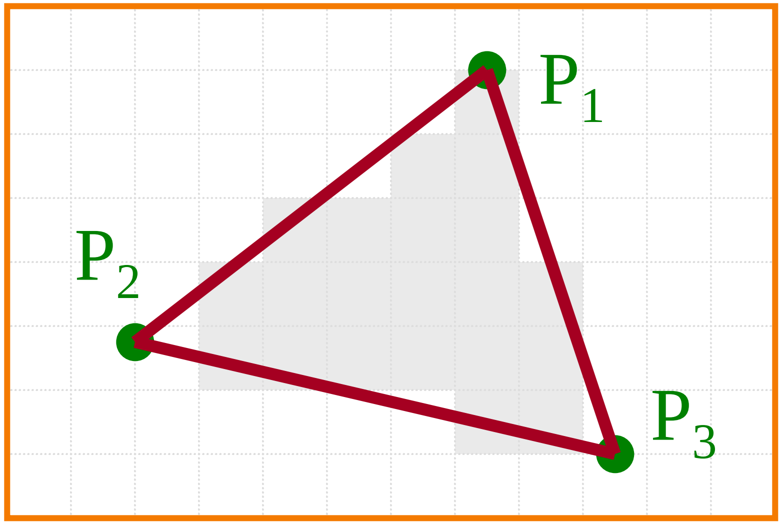


Rasterization

- 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

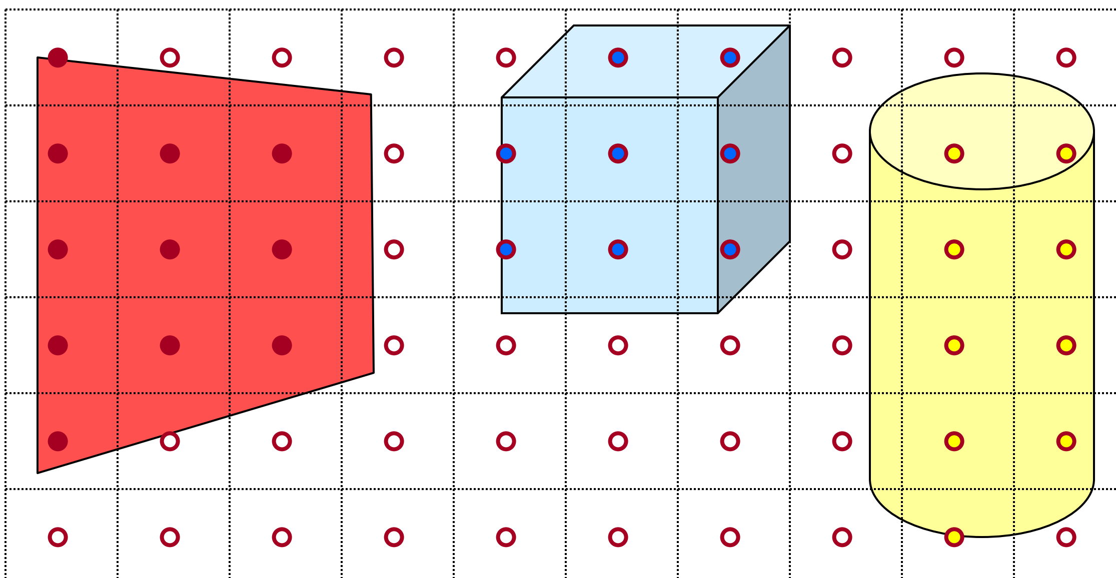
- How do we choose a color for each filled pixel?



Emphasis on methods that can be implemented in hardware

Ray Casting

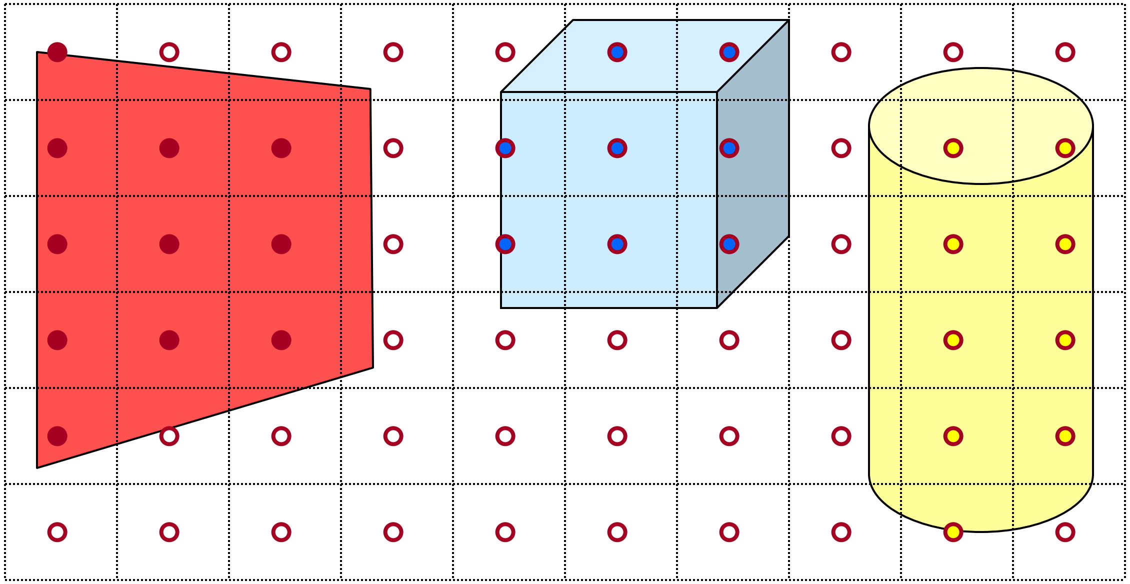
- Simplest shading approach is to perform *independent* lighting calculation for every pixel



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

Polygon Shading

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



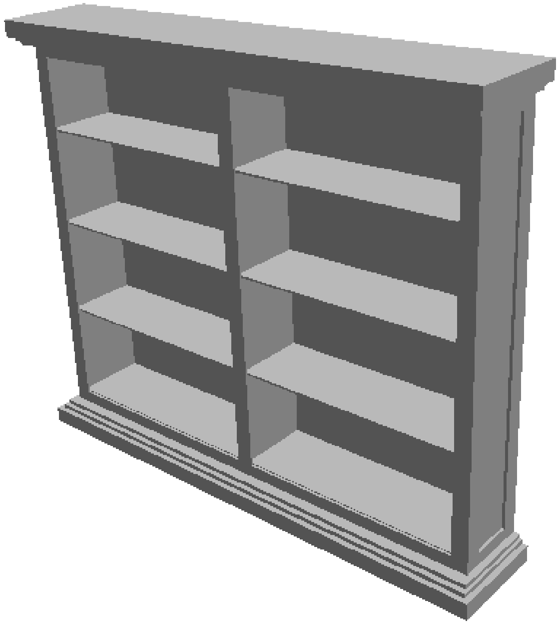
$$I = I_E + K_A I_{AL} + \sum_i \left(K_D (N \cdot L_i) I_i + K_S (V \cdot R_i)^n I_i \right)$$

Polygon Shading Algorithms

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

Flat Shading

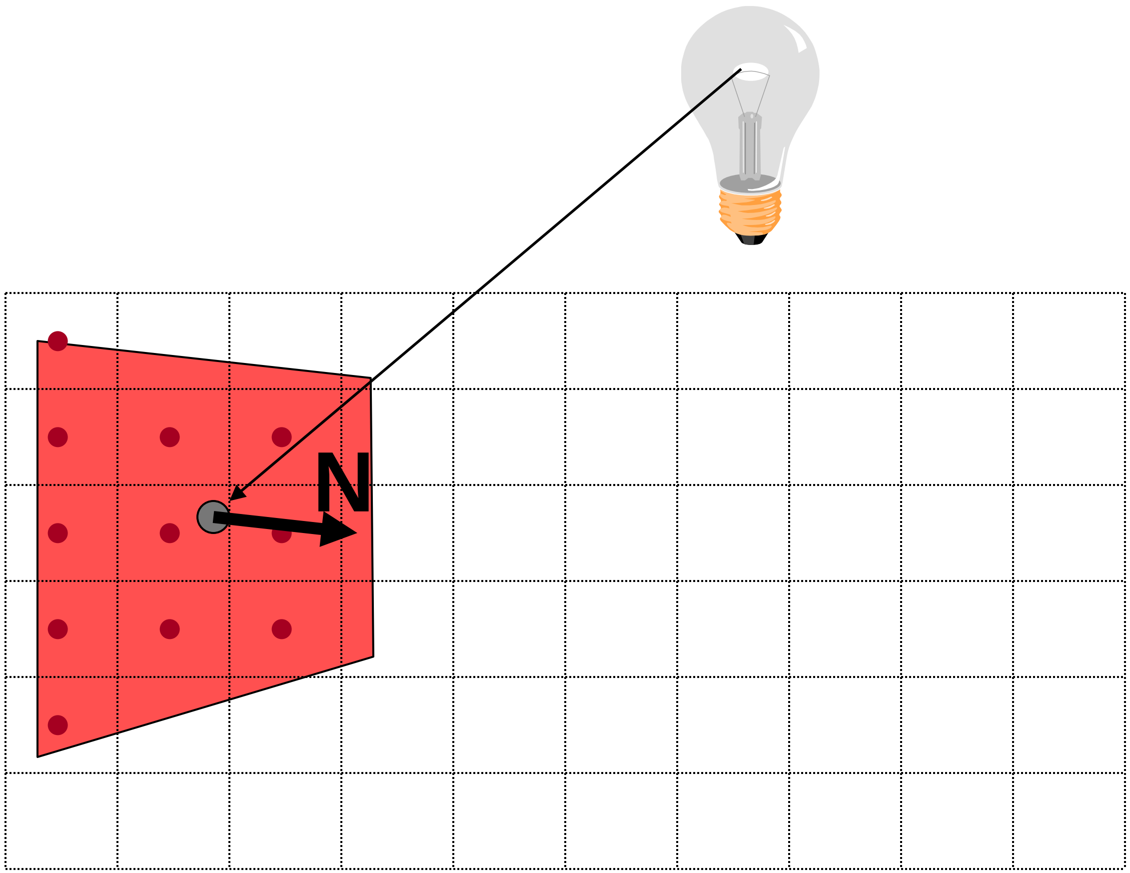
- What if a faceted object is illuminated only by directional light sources and is viewed from infinitely far away



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

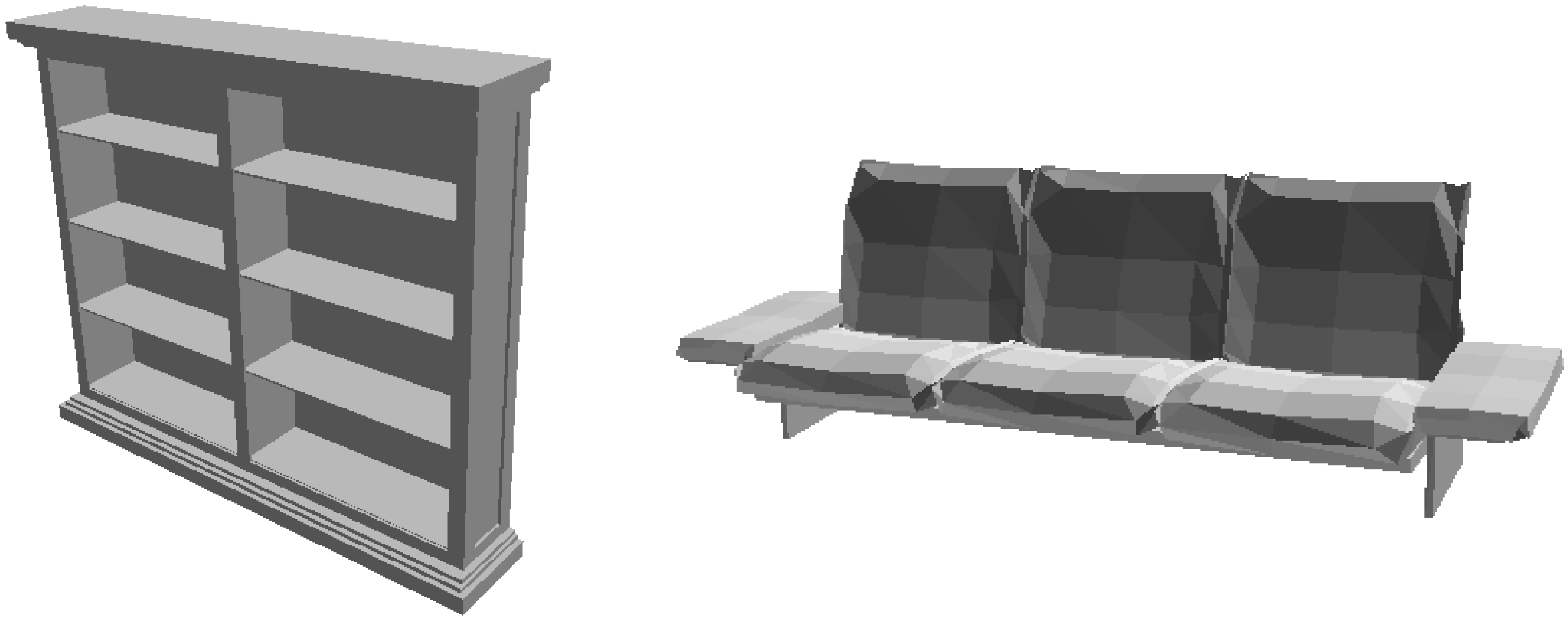
Flat Shading

- One illumination calculation per polygon
  - Assign all pixels inside each polygon the same color



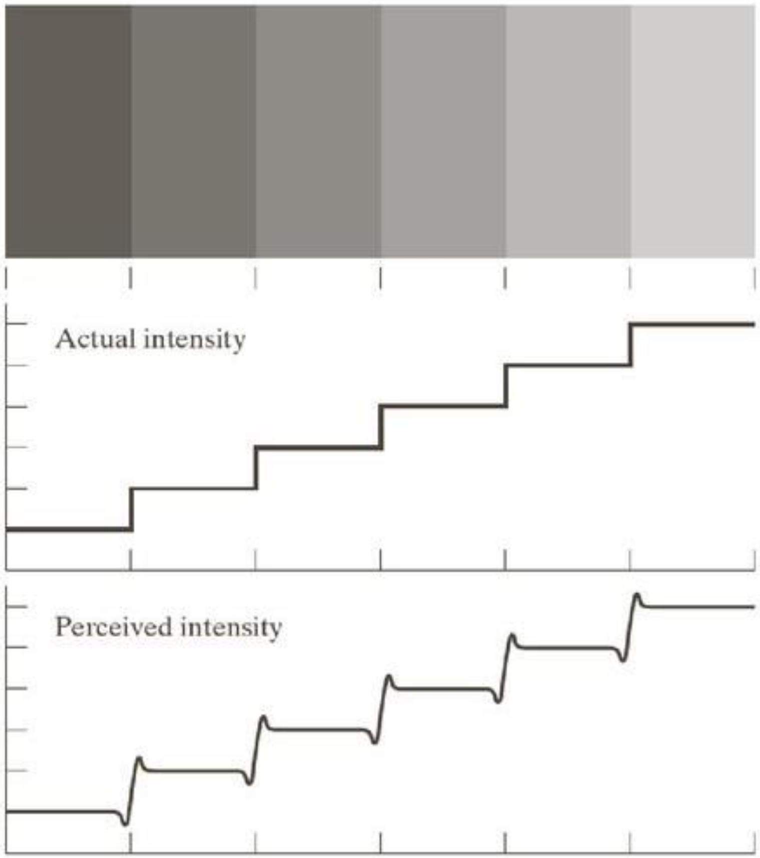
Flat Shading

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



Mach Band Effect

- Edges between adjacent shades of gray are perceived as exaggerated.

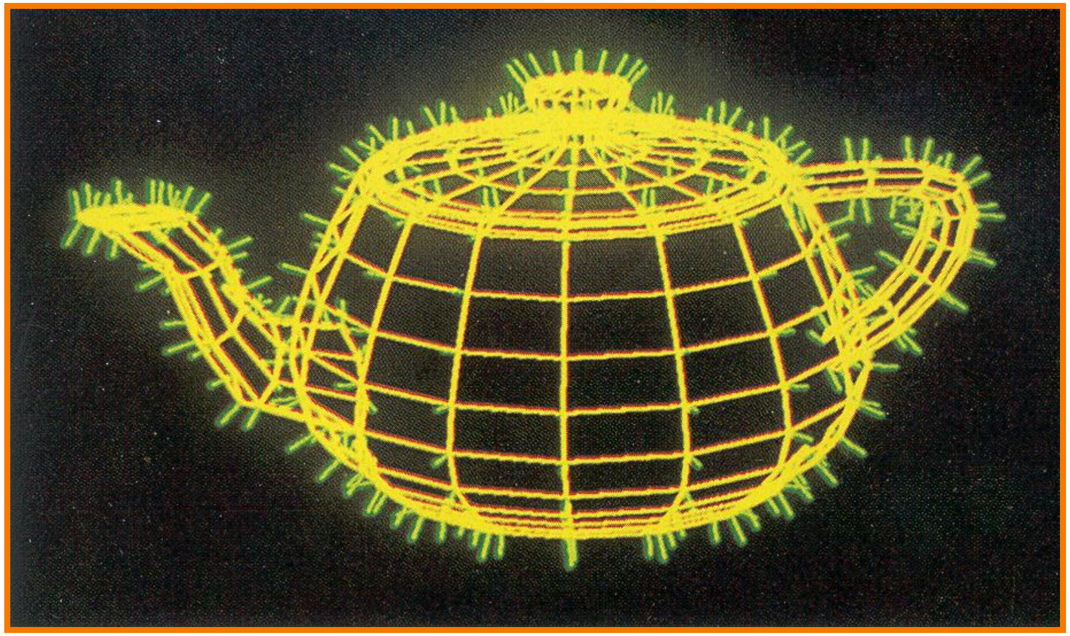


Polygon Shading Algorithms

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

Gouraud Shading

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

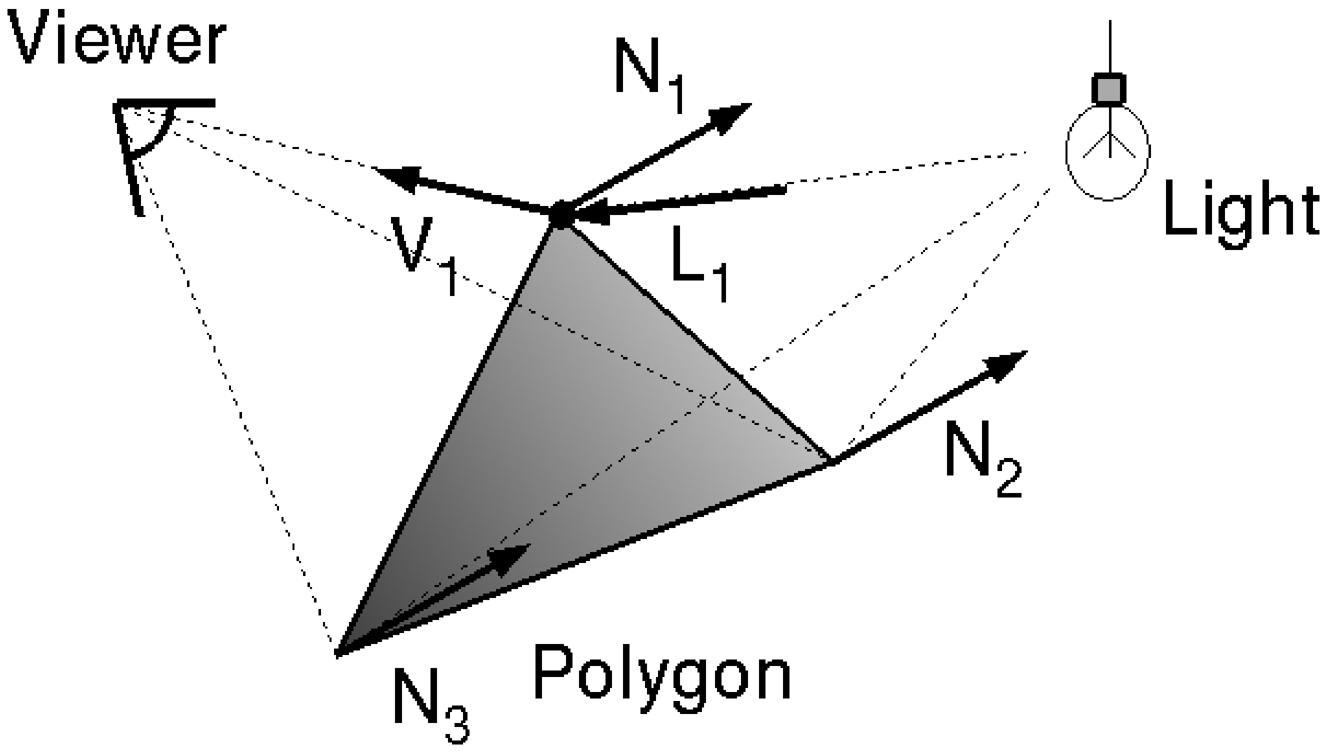


Watt Plate 7

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

Gouraud Shading

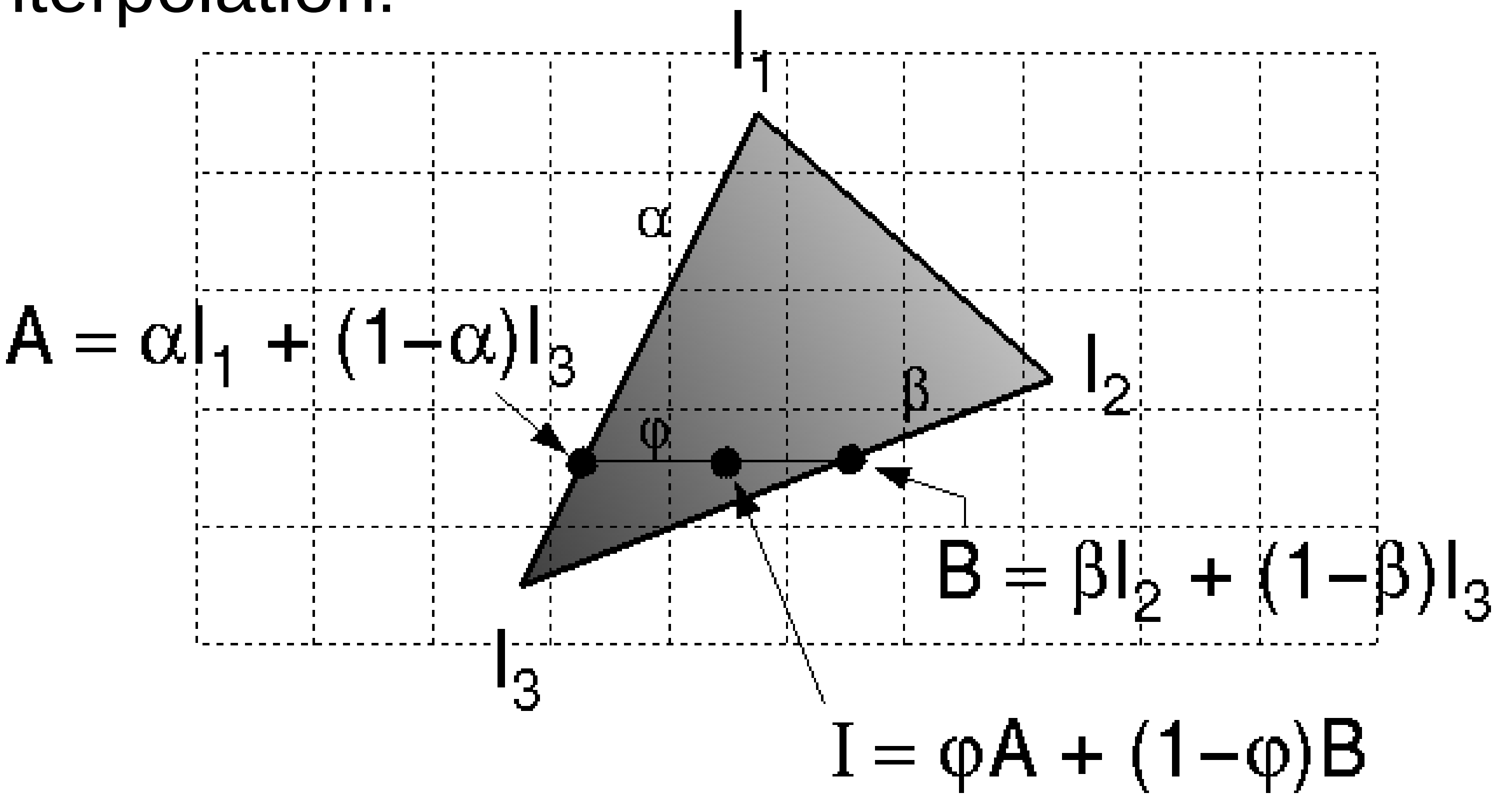
- One lighting calculation per vertex
  - Assign pixels inside polygon by interpolating colors computed at vertices



Gouraud Shading

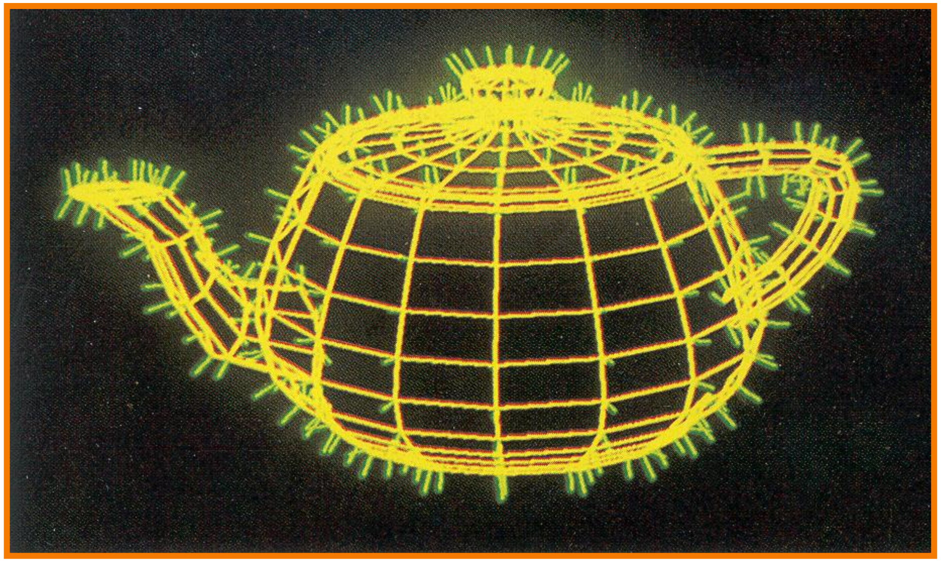
Bilinear interpolation of colors at vertices

- down and across scan lines = barycentric interpolation!



Gouraud Shading

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

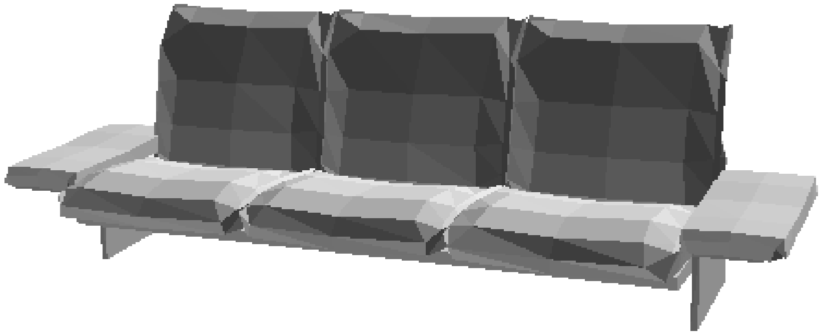


Mesh with shared normals at vertices

Watt Plate 7

Gouraud Shading

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

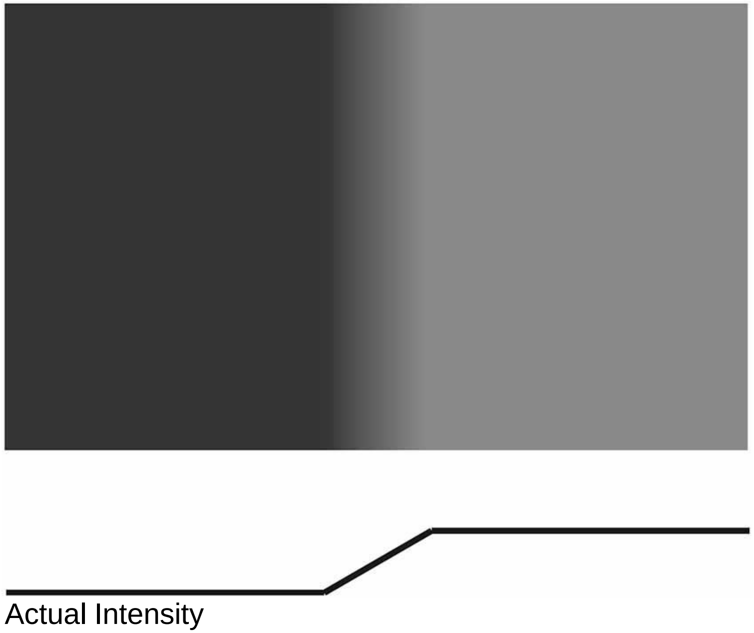


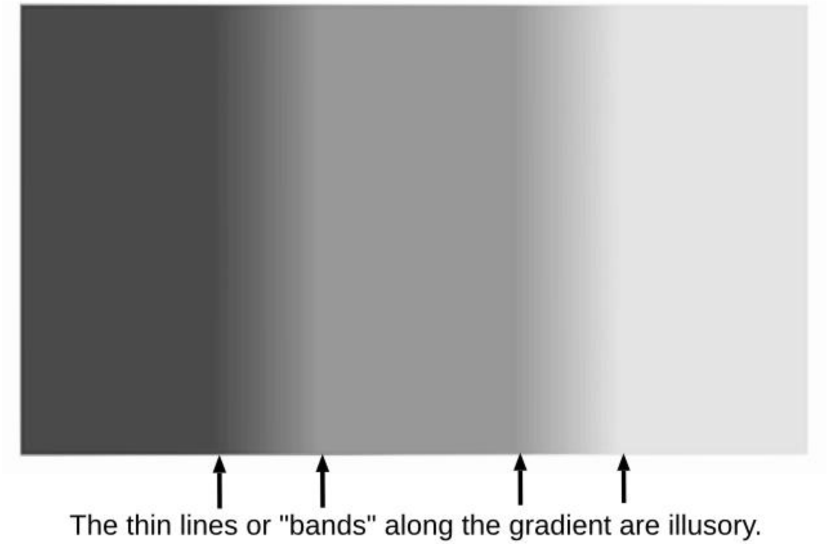
Flat Shading

Gouraud Shading

Mach Band Effect

- Mach Band Effect also affects Gouraud Shading for piecewise linear interpolation.



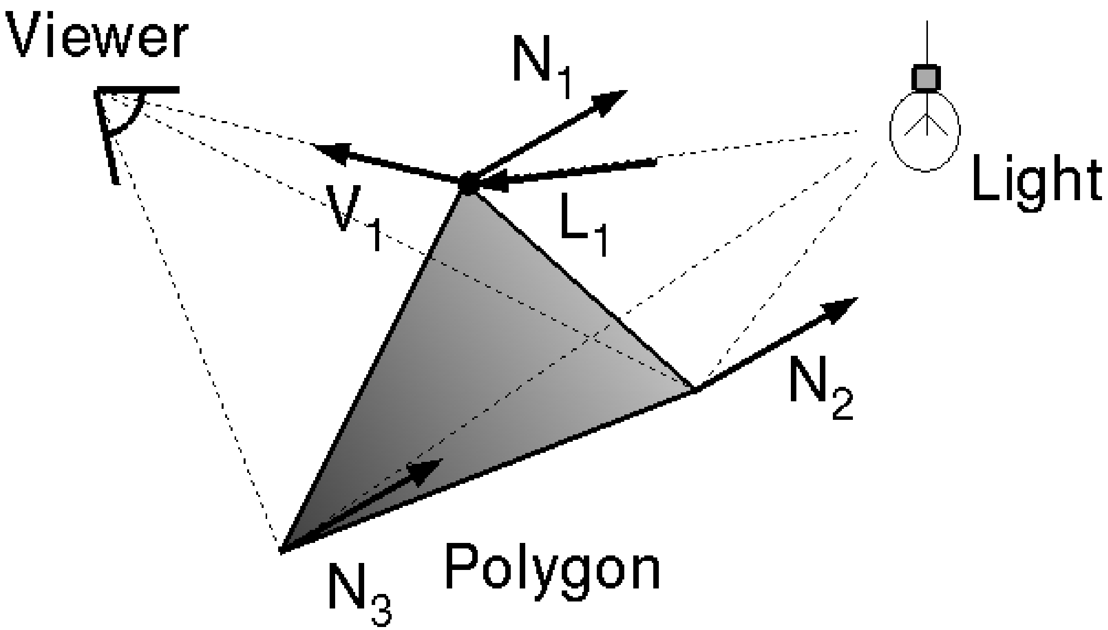


Polygon Shading Algorithms

- Flat Shading
- Gouraud Shading
- **Phong Shading** (≠ Phong reflectance model)

Phong Shading

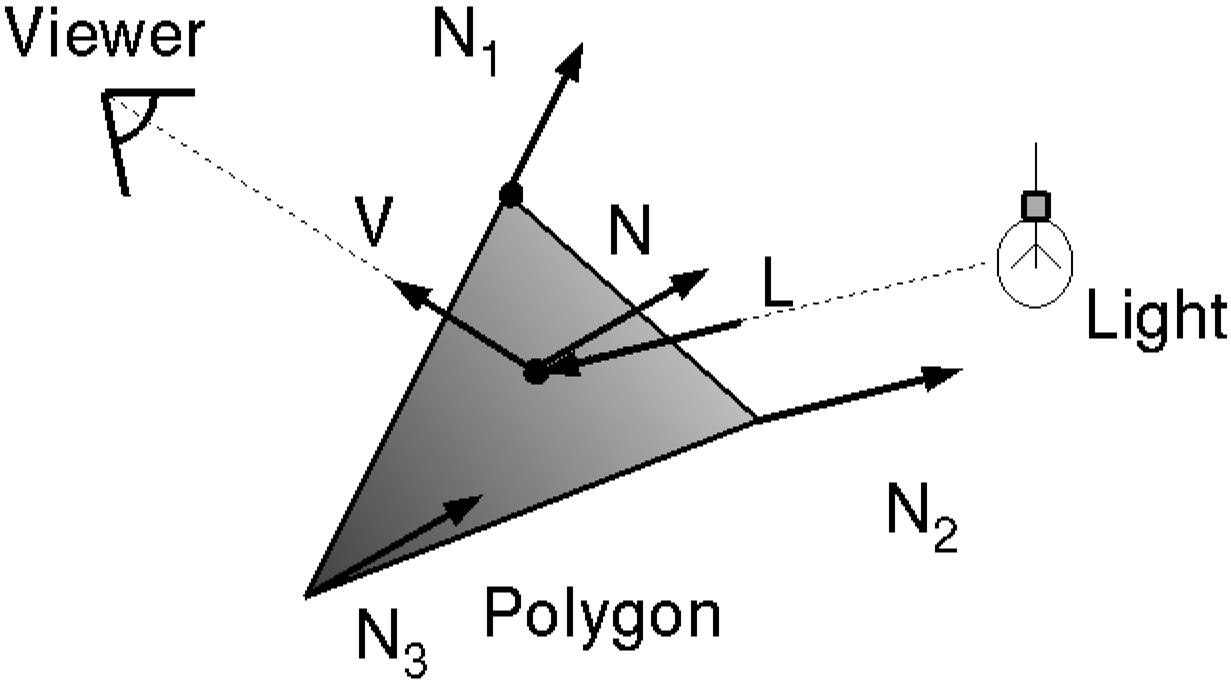
- What if polygonal mesh is too coarse to capture illumination effects in polygon interiors?



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

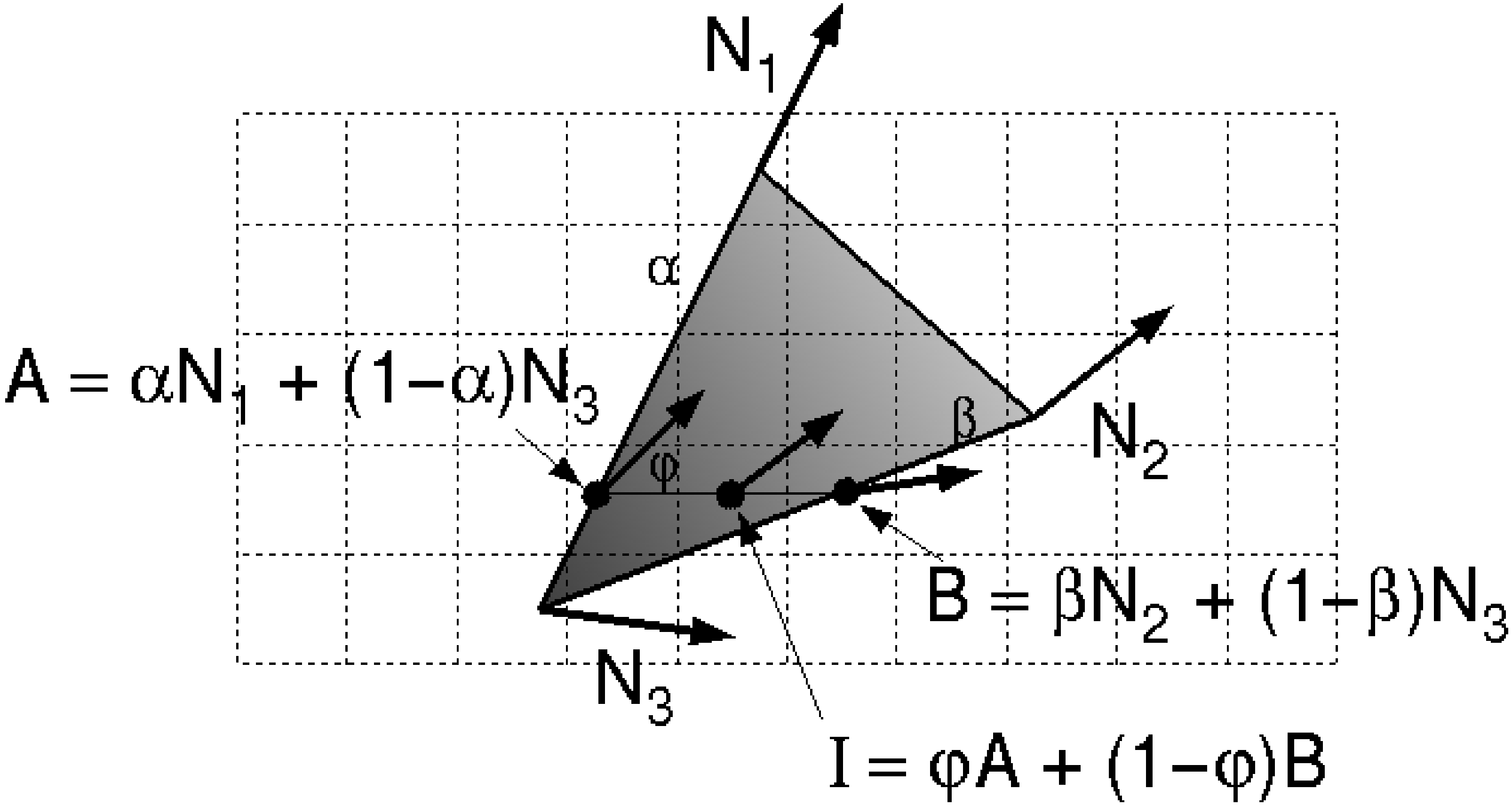
Phong Shading

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



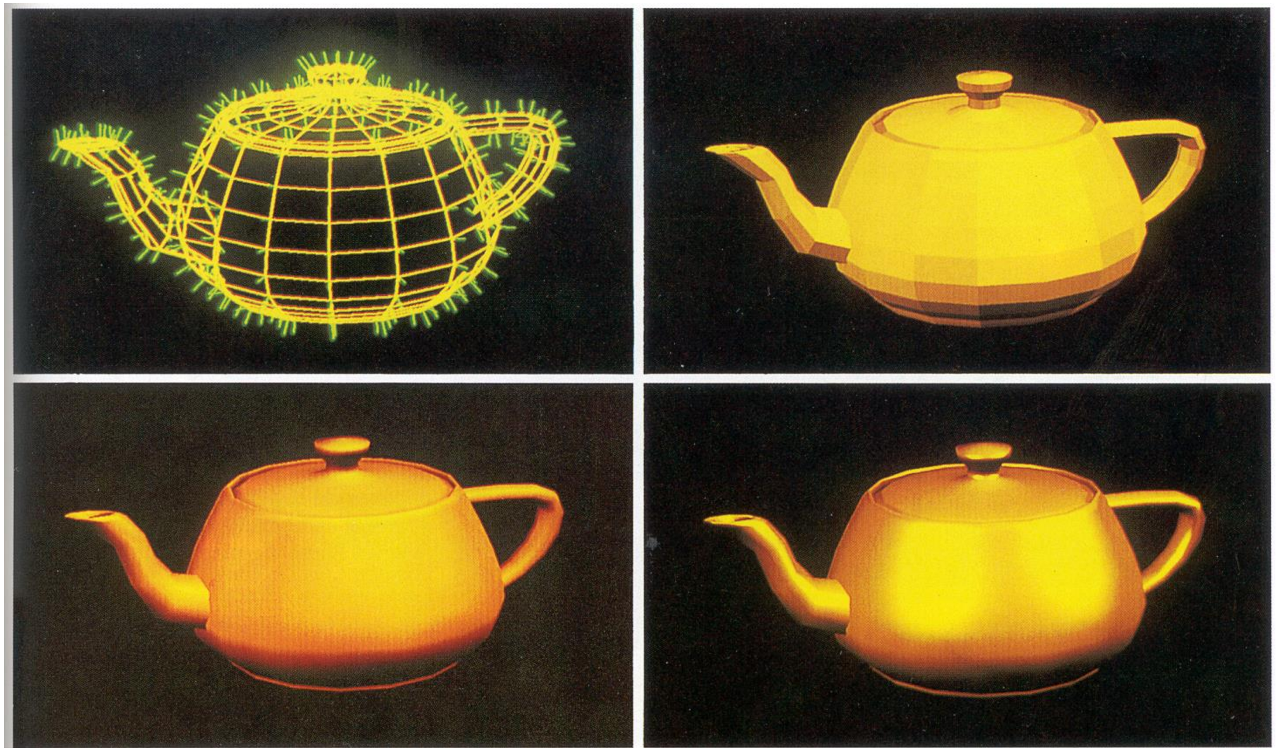
Phong Shading

Bilinear interpolation of surface normals at vertices



Polygon Shading Algorithms

Wireframe Flat



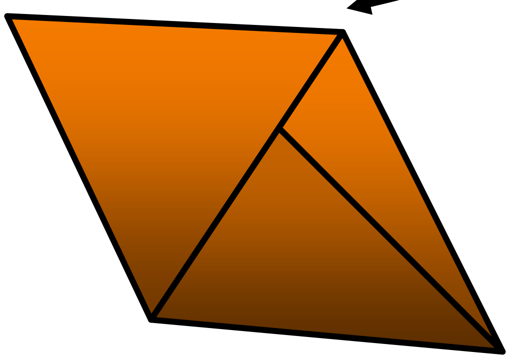
Gouraud Phong

Demo: https://threejs.org/docs/scenes/material-browser.html#MeshPhongMaterial

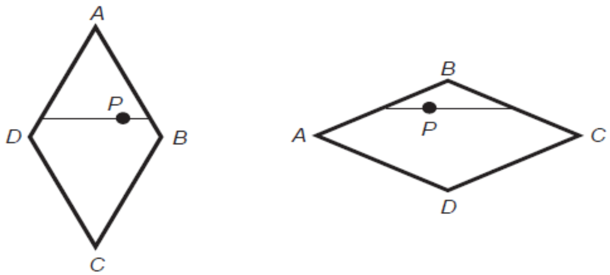
Watt Plate 7

Shading Issues

- Problems with interpolated shading:
  - Polygonal *silhouettes* still obvious
  - Perspective distortion (due to *screen-space interpolation*)
  - Problems at T-junctions



The results of interpolated-shading is not independent of the projected polygons position (Foley Figure 14.22).

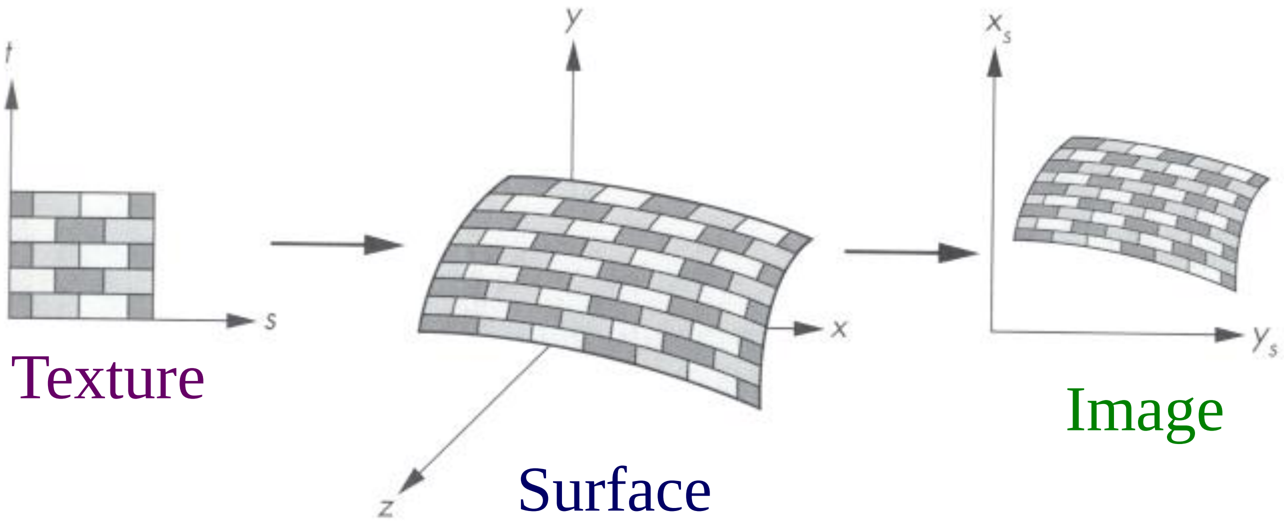


Rasterization

- 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

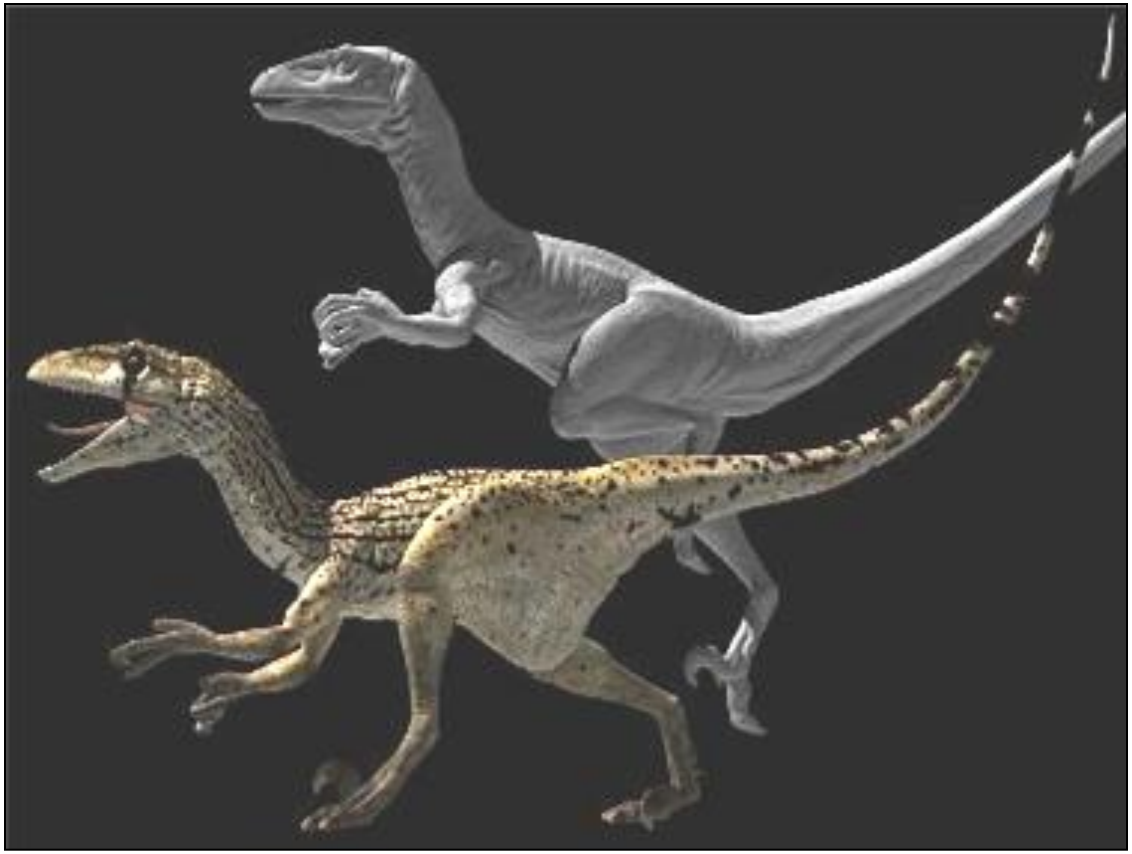
- Describe **color variation** in interior of 3D polygon
  - When scan converting a polygon, **vary pixel colors** according to values fetched from a texture image



Angel Figure 9.3

Textures

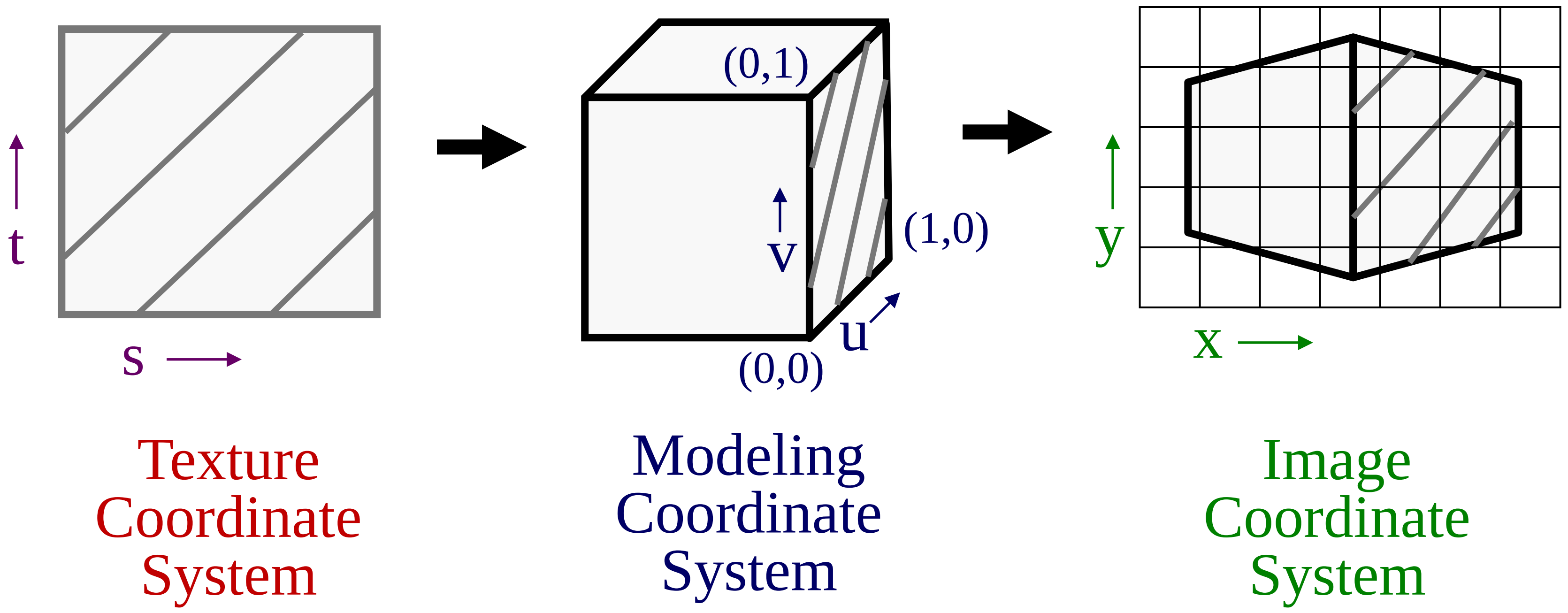
- Add visual detail to surfaces of 3D objects



[Daren Horley]

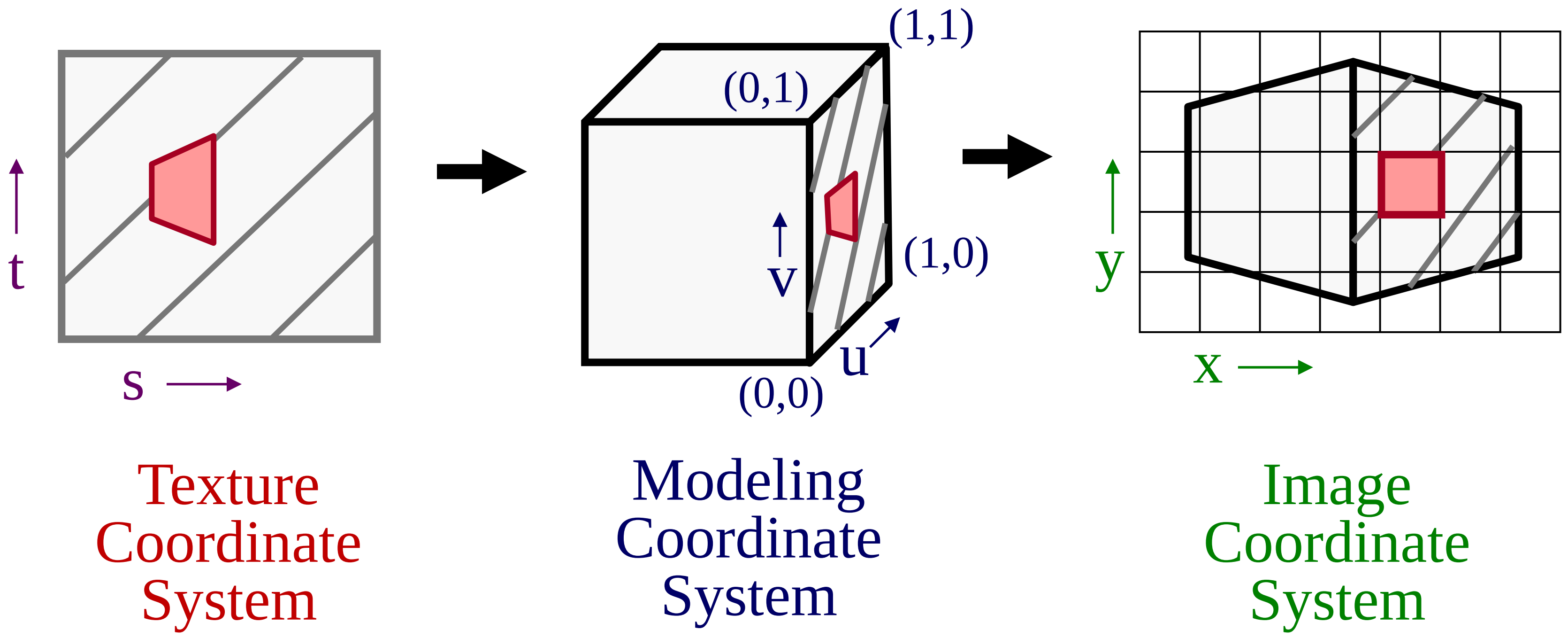
Texture Mapping

- Steps:
  1. Define texture
  2. Specify mapping from texture to surface
  3. 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)



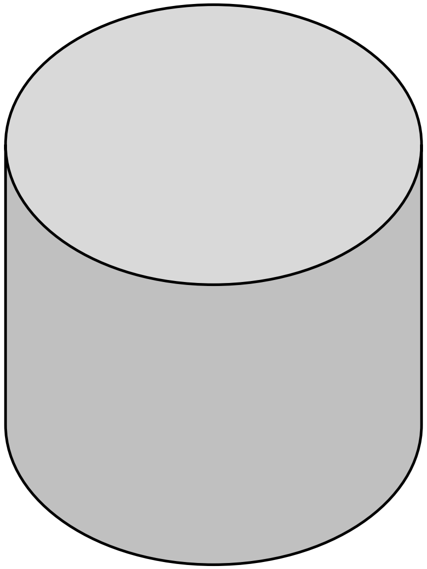
Texture Overview

- Texture mapping stages
  - Parameterization
  - Mapping
  - Filtering
- Texture mapping applications
  - Modulation textures
  - Illumination mapping
  - Bump mapping
  - Environment mapping
  - Image-based rendering
  - Non-photorealistic rendering

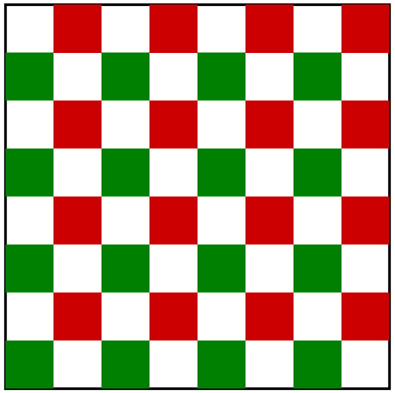
Texture Overview

- Texture mapping stages
  - Parameterization
  - Mapping
  - Filtering
- Texture mapping applications
  - Modulation textures
  - Illumination mapping
  - Bump mapping
  - Environment mapping
  - Image-based rendering
  - Non-photorealistic rendering

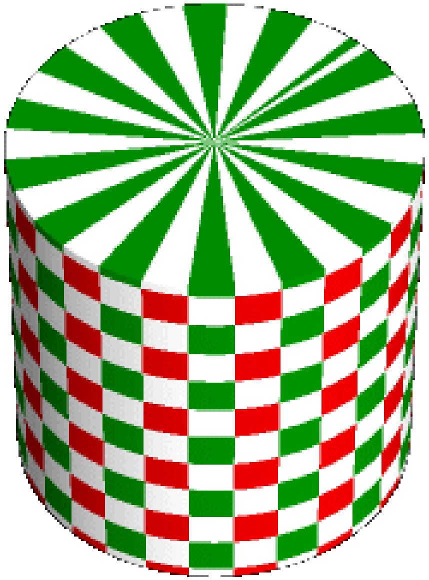
Texture Parameterization



geometry



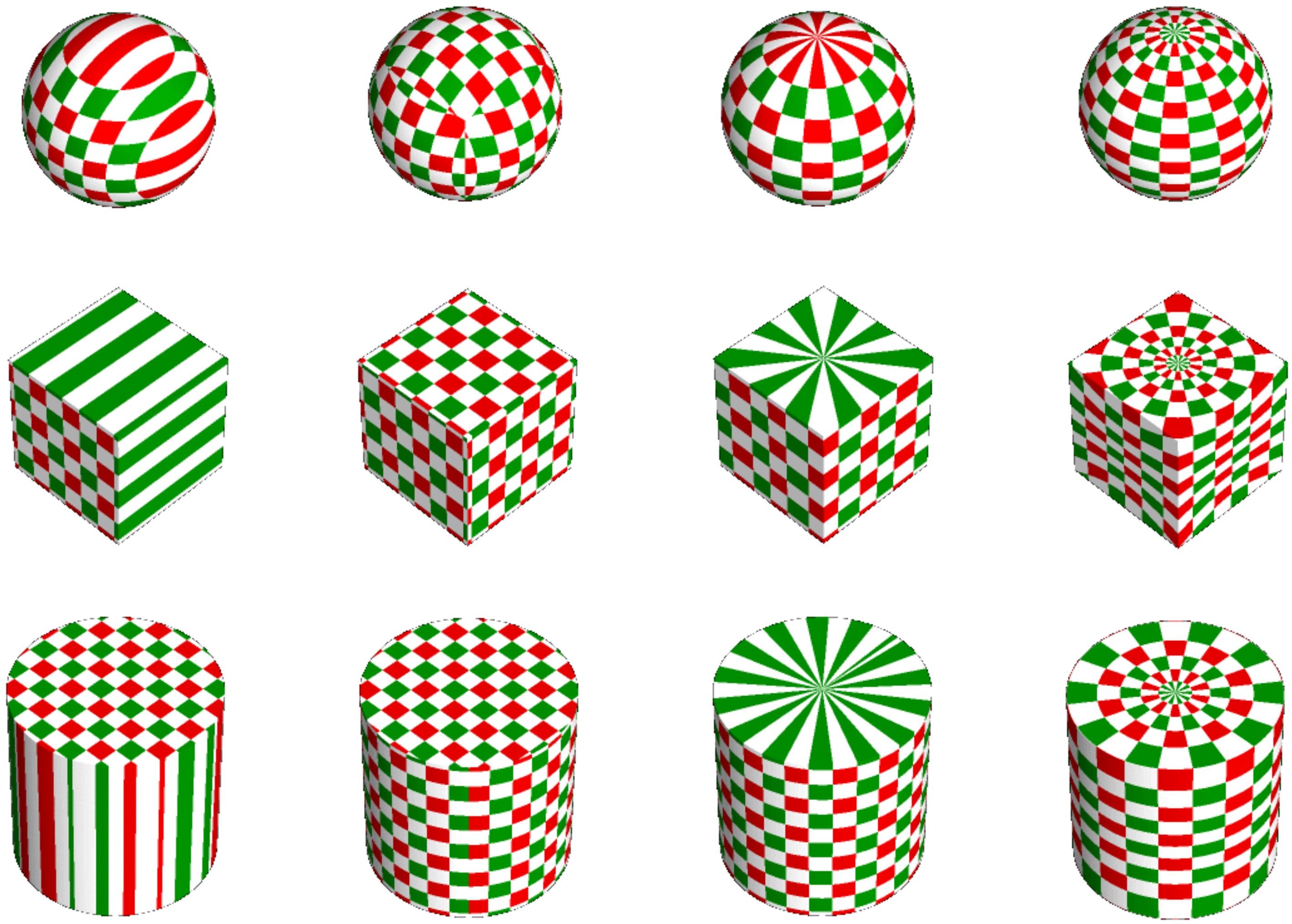
image



texture map

- Q: How do we decide ***where* on the geometry** each color from the image should go?

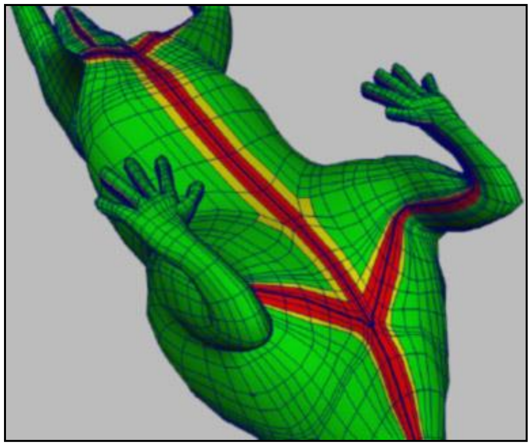
Texture Parameterization

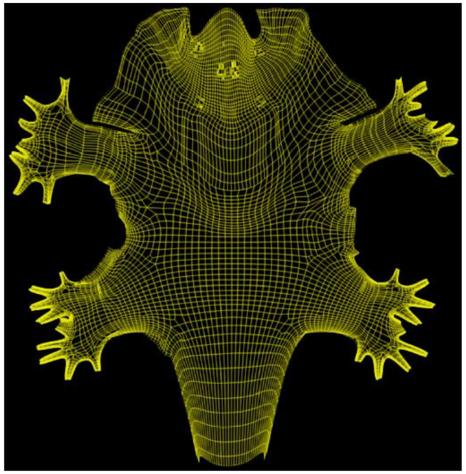


[Paul Bourke]

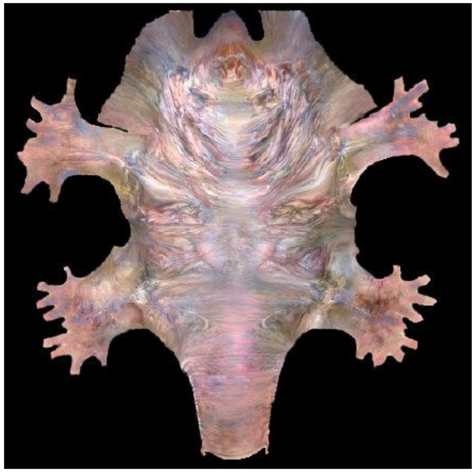
Texture Parameterization

Option1: unfold the surface





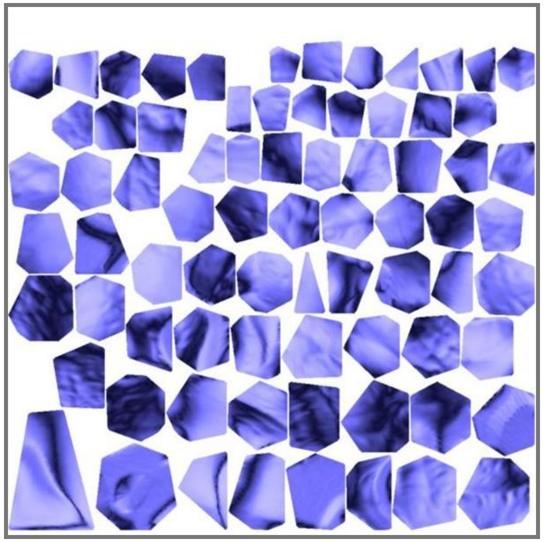
[Piponi2000]



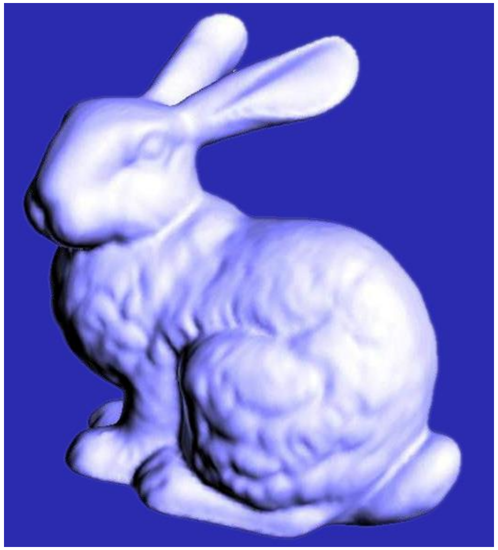
Texture Parameterization

Option2: make an atlas

charts



atlas



surface

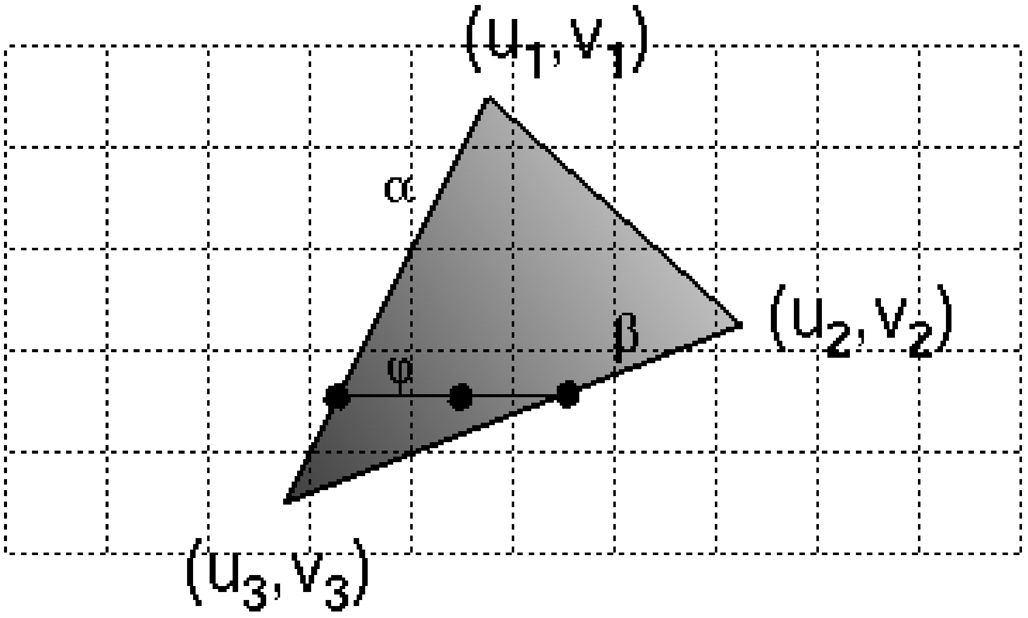
[Sander2001]

Texture Overview

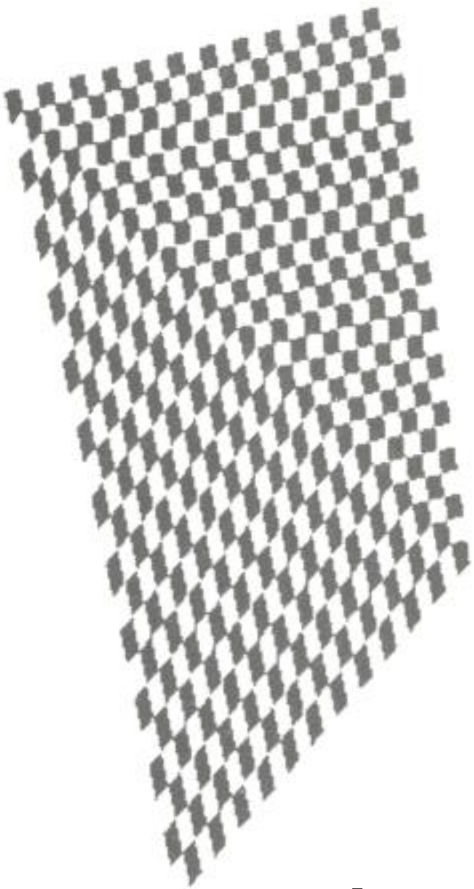
- Texture mapping stages
  - Parameterization
  - Mapping
  - Filtering
- Texture mapping applications
  - Modulation textures
  - Illumination mapping
  - Bump mapping
  - Environment mapping
  - Image-based rendering
  - Non-photorealistic rendering

Texture Mapping

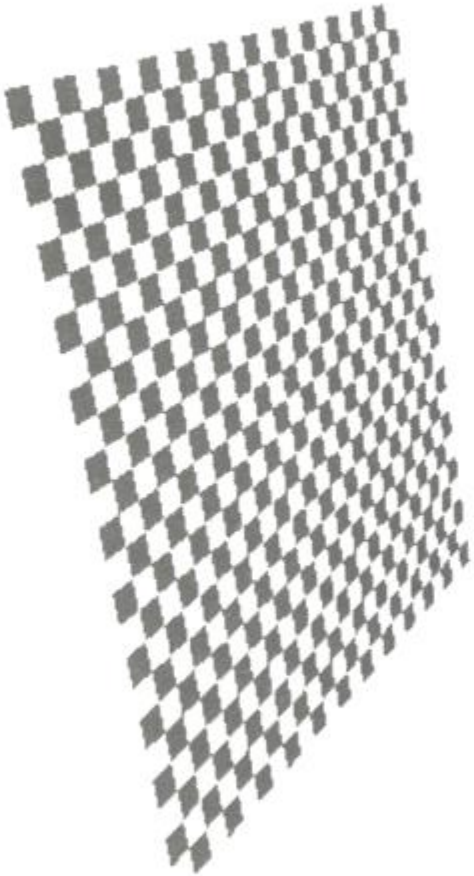
- Scan conversion
  - Interpolate texture coordinates down/across scan lines



Perspective Divide



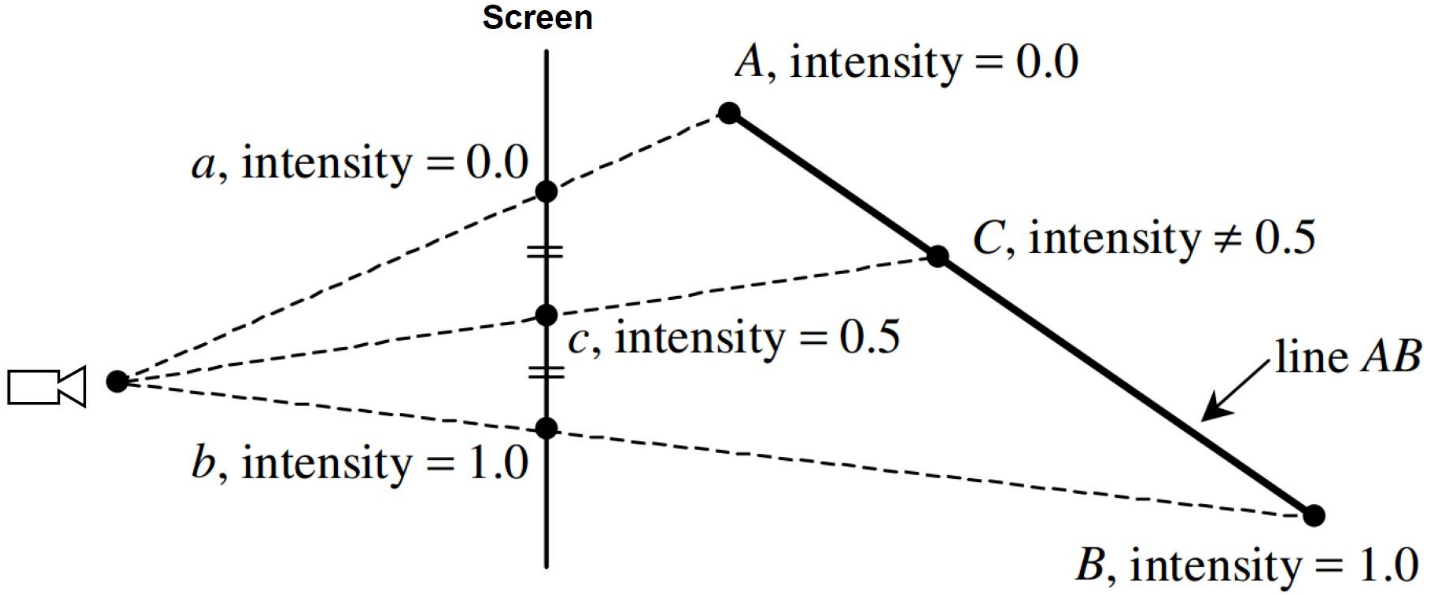
Linear interpolation
of texture coordinates



Correct interpolation

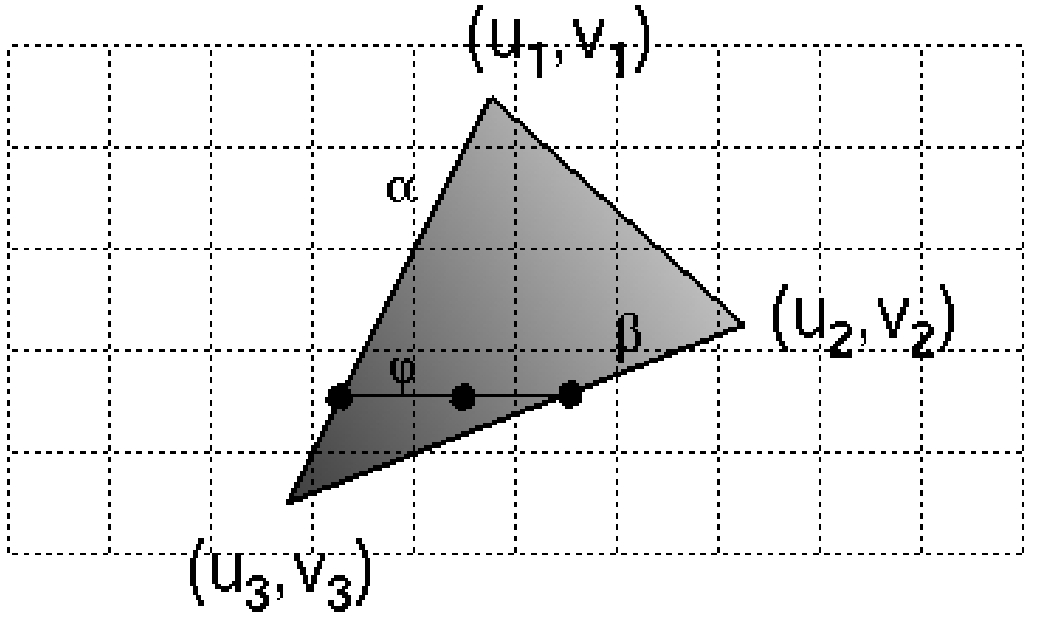
Hill Figure 8.42

Perspective Divide



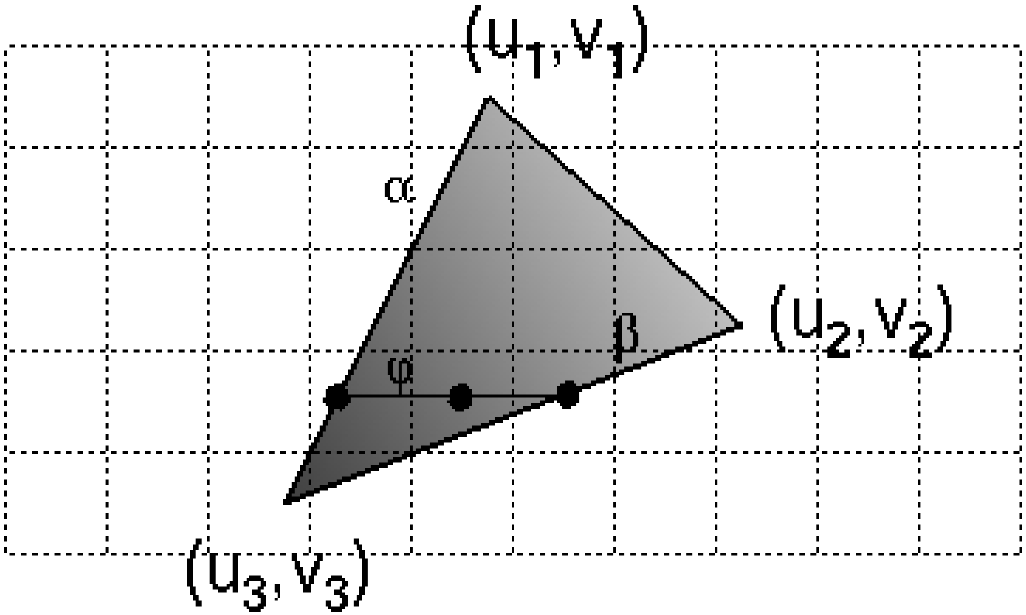
Texture Mapping

- Scan conversion
  - Interpolate texture coordinates down/across scan lines
  - Distortion due to bilinear interpolation approximation
    » Cut polygons into smaller ones, or



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



Perspective Divide

Assume triangle attribute varies linearly across the triangle

Attribute's value at 3D (non-homogeneous) point $P = \begin{bmatrix} x & y & z \end{bmatrix}^T$ **is then:**

$$f(x, y, z) = ax + by + cz$$

Get 2D homogeneous representation : $\begin{bmatrix} x_{\text{2D-H}} & y_{\text{2D-H}} & w \end{bmatrix}^T = \begin{bmatrix} x & y & z \end{bmatrix}^T$

Perspective Divide

Assume triangle attribute varies linearly across the triangle

Attribute's value at 3D (non-homogeneous) point $P = \begin{bmatrix} x & y & z \end{bmatrix}^T$ **is then:**

$$f(x, y, z) = ax + by + cz$$

Get 2D homogeneous representation : $\begin{bmatrix} x_{\text{2D-H}} & y_{\text{2D-H}} & w \end{bmatrix}^T = \begin{bmatrix} x & y & z \end{bmatrix}^T$

Rewrite attribute equation for f **in terms of 2D homogeneous coordinates:**

$$f = ax_{\text{2D-H}} + by_{\text{2D-H}} + cw$$

Perspective Divide

Assume triangle attribute varies linearly across the triangle

Attribute's value at 3D (non-homogeneous) point $P = \begin{bmatrix} x & y & z \end{bmatrix}^T$ **is then:**

$$f(x, y, z) = ax + by + cz$$

Get 2D homogeneous representation : $\begin{bmatrix} x_{\text{2D-H}} & y_{\text{2D-H}} & w \end{bmatrix}^T = \begin{bmatrix} x & y & z \end{bmatrix}^T$

Rewrite attribute equation for f **in terms of 2D homogeneous coordinates:**

$$f = ax_{\text{2D-H}} + by_{\text{2D-H}} + cw$$

$$\frac{f}{w} = a\frac{x_{\text{2D-H}}}{w} + b\frac{y_{\text{2D-H}}}{w} + c$$

Perspective Divide

Assume triangle attribute varies linearly across the triangle

Attribute's value at 3D (non-homogeneous) point $P = \begin{bmatrix} x & y & z \end{bmatrix}^T$ **is then:**

$$f(x, y, z) = ax + by + cz$$

Get 2D homogeneous representation : $\begin{bmatrix} x_{\text{2D-H}} & y_{\text{2D-H}} & w \end{bmatrix}^T = \begin{bmatrix} x & y & z \end{bmatrix}^T$

Rewrite attribute equation for f **in terms of 2D homogeneous coordinates:**

$$f = ax_{\text{2D-H}} + by_{\text{2D-H}} + cw$$

$$\frac{f}{w} = a\frac{x_{\text{2D-H}}}{w} + b\frac{y_{\text{2D-H}}}{w} + c$$

$$\frac{f}{w} = ax_{\text{2D}} + by_{\text{2D}} + c$$

Where $\begin{bmatrix} x_{\text{2D}} & y_{\text{2D}} \end{bmatrix}^T$ **are projected screen 2D coordinates (after homogeneous divide)**

Perspective Divide

Assume triangle attribute varies linearly across the triangle

Attribute's value at 3D (non-homogeneous) point $P = \begin{bmatrix} x & y & z \end{bmatrix}^T$ **is then:**

$$f(x, y, z) = ax + by + cz$$

Get 2D homogeneous representation : $\begin{bmatrix} x_{\text{2D-H}} & y_{\text{2D-H}} & w \end{bmatrix}^T = \begin{bmatrix} x & y & z \end{bmatrix}^T$

Rewrite attribute equation for f **in terms of 2D homogeneous coordinates:**

$$f = ax_{\text{2D-H}} + by_{\text{2D-H}} + cw$$

$$\frac{f}{w} = a\frac{x_{\text{2D-H}}}{w} + b\frac{y_{\text{2D-H}}}{w} + c$$

$$\frac{f}{w} = ax_{\text{2D}} + by_{\text{2D}} + c$$

Where $\begin{bmatrix} x_{\text{2D}} & y_{\text{2D}} \end{bmatrix}^T$ **are projected screen 2D coordinates (after homogeneous divide)**

So … $\frac{f}{w}$ **is affine function of 2D screen coordinates…**

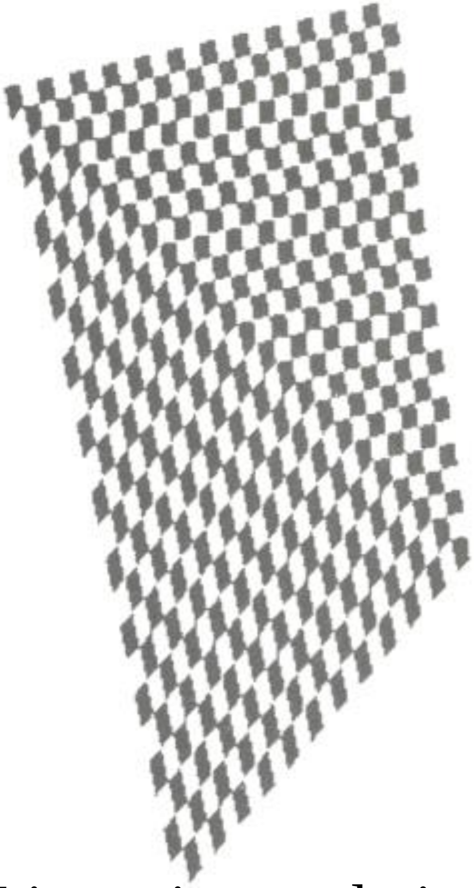
Perspective Divide

- Compute at each vertex *after perspective transformation*:
  - "Numerators" s/w, t/w
  - "Denominator" 1/w

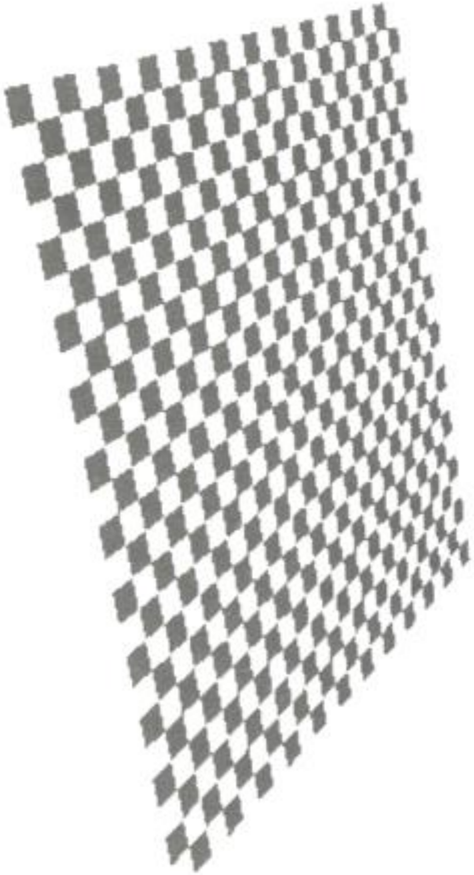
- Linearly interpolate s/w, and t/w and 1/w across the polygon

- At each pixel:
  - Perform perspective division of interpolated texture coordinates (s/w, t/w) by interpolated 1/w (i.e., numerator over denominator) to get (s, t)

Perspective Divide



Linear interpolation
of texture coordinates



Correct interpolation

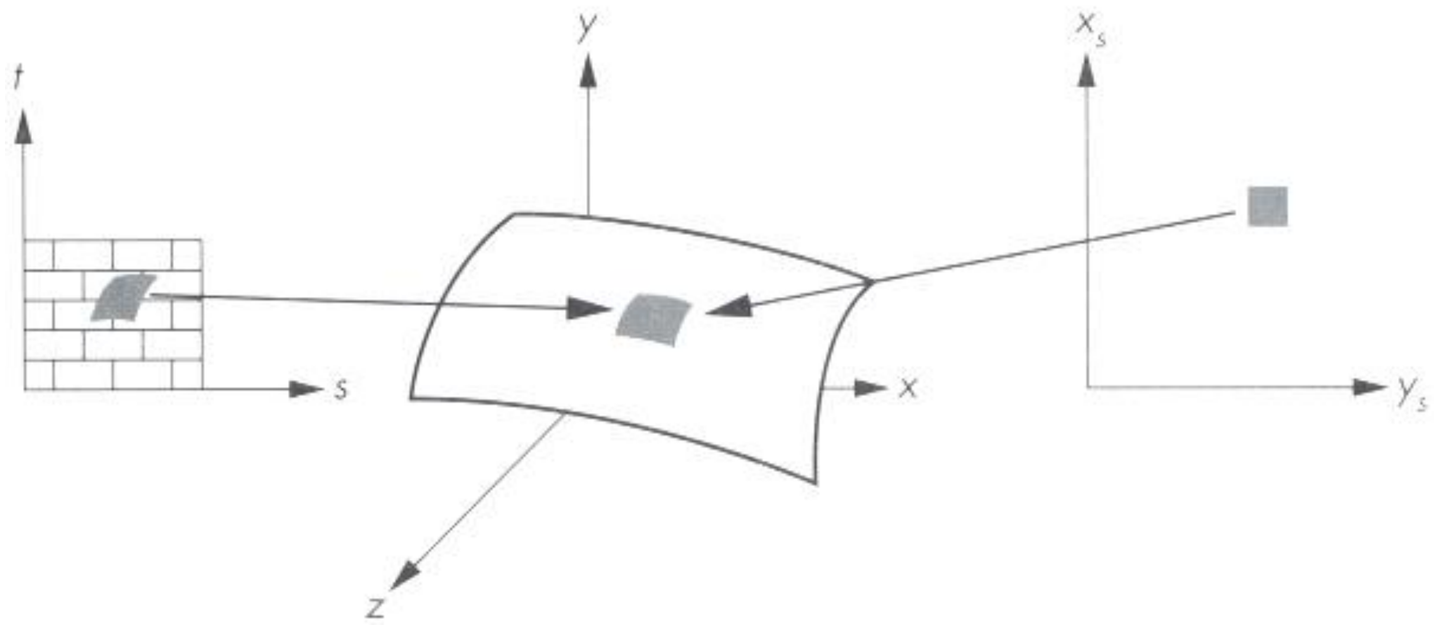
Hill Figure 8.42

Texture 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 Filtering

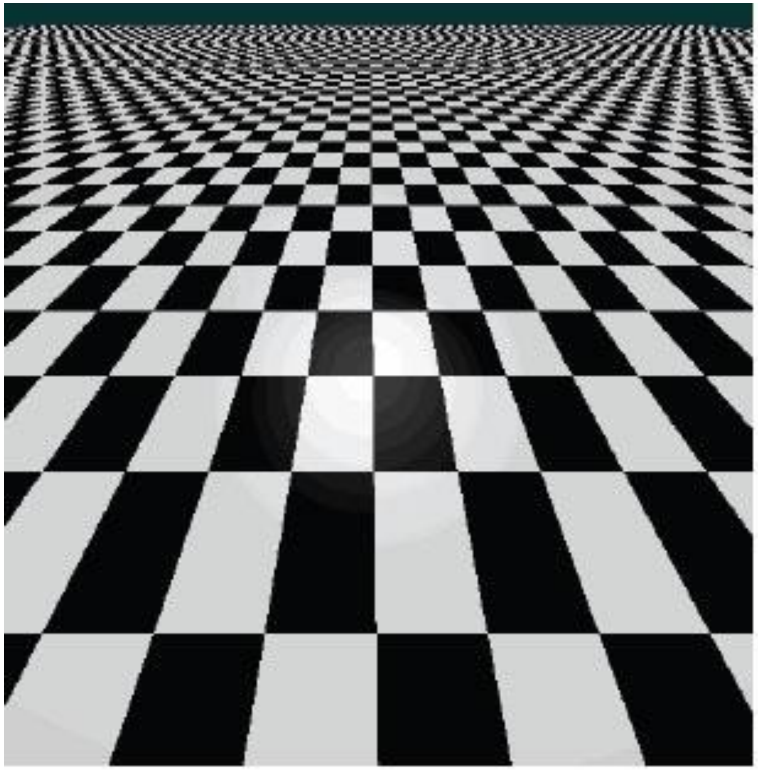
- Must sample texture to determine color at each pixel in image



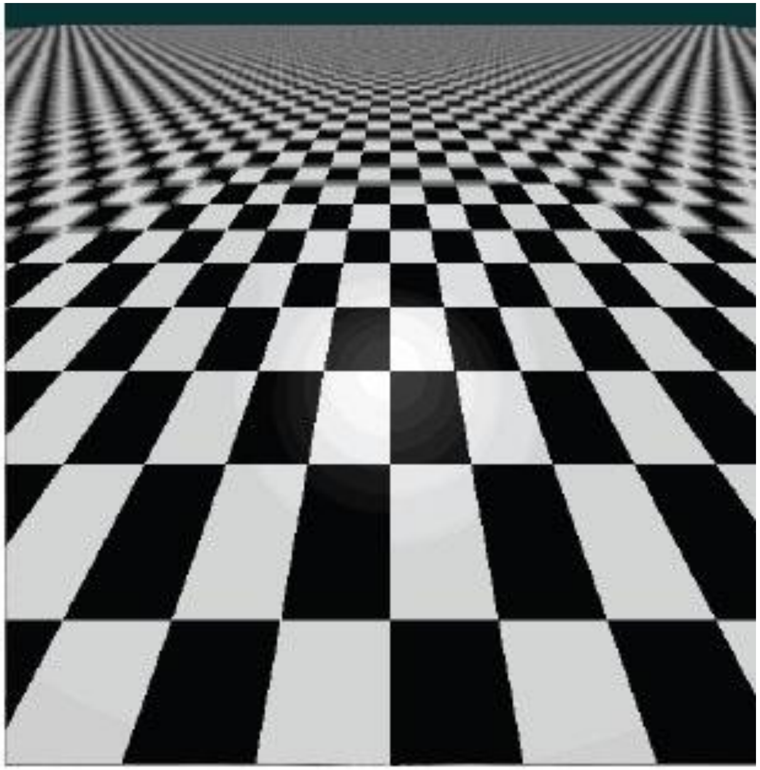
Angel Figure 9.4

Texture Filtering

- Aliasing is a problem



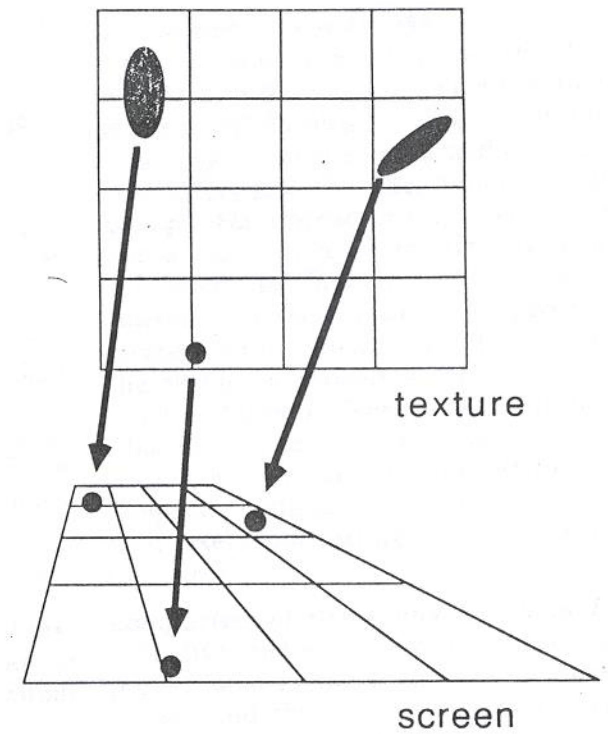
Point sampling



Area filtering

Texture Filtering

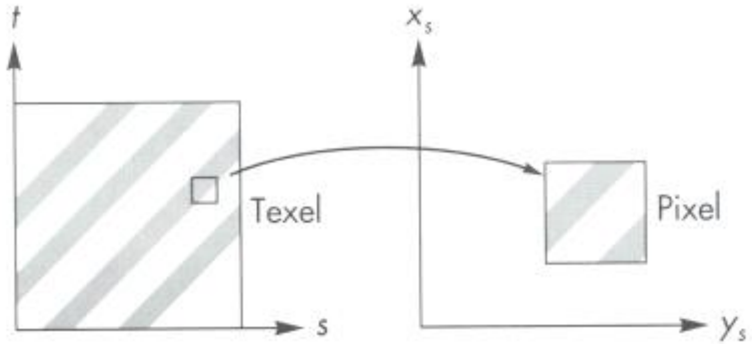
- Ideally, use elliptically shaped convolution filters



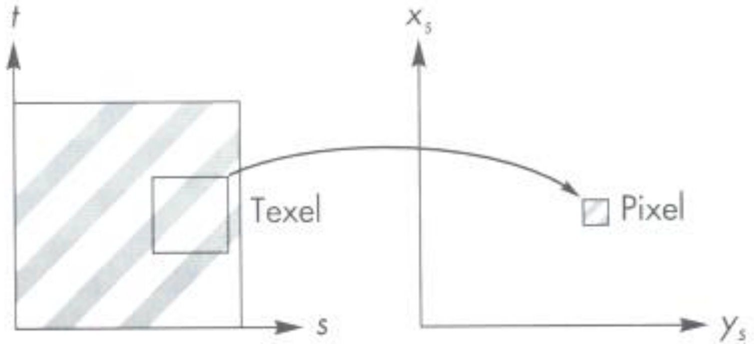
In practice, *use rectangles or squares*

Texture Filtering

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



Magnification

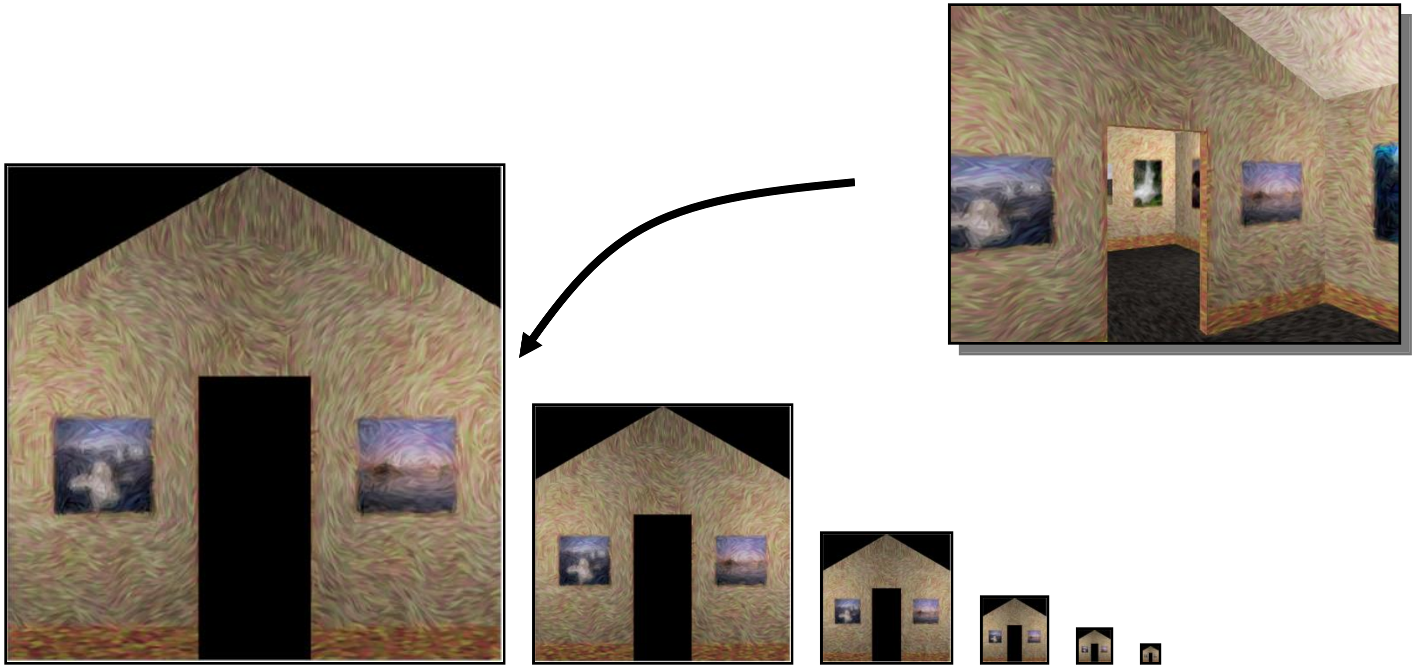


Minification

Angel Figure 9.14

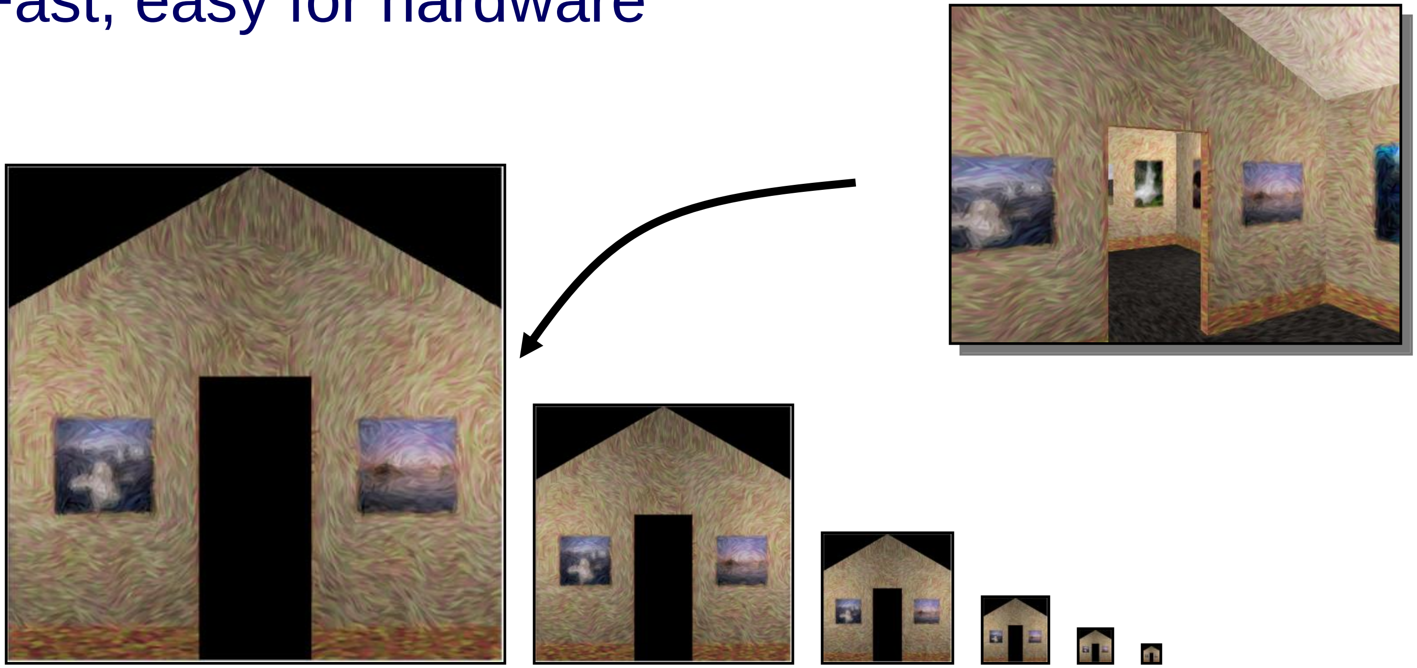
Mipmaps

- Keep textures prefiltered at multiple resolutions
  - Usually powers of 2



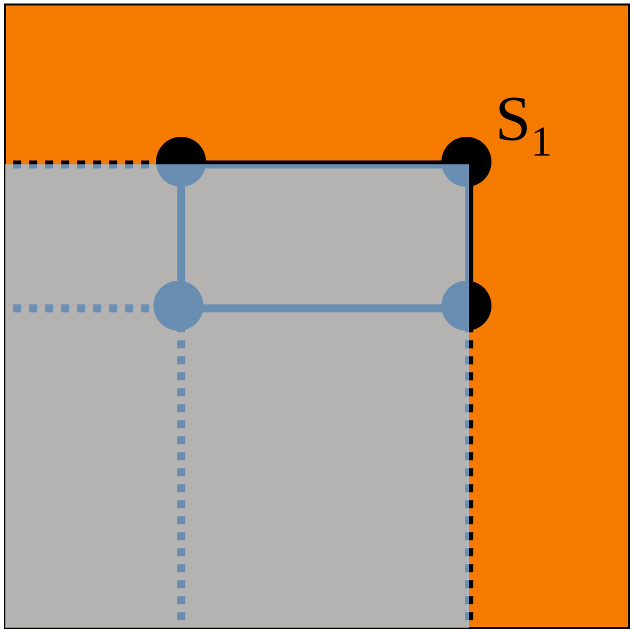
Mipmaps

- Keep textures prefiltered at multiple resolutions
  - Usually powers of 2
  - For each pixel, linearly interpolate between two closest levels (i.e., trilinear filtering)
  - Fast, easy for hardware



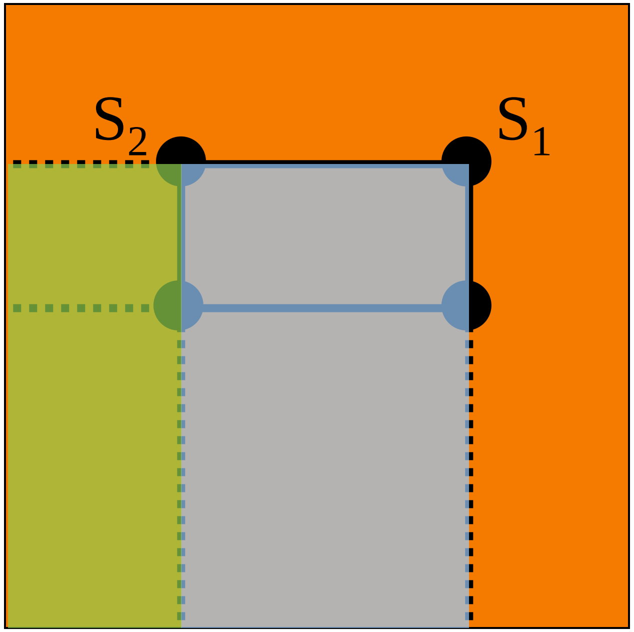
Summed-area tables

- At each texel keep sum of all values down & left
  - To compute sum of all values within a rectangle, simply combine four entries: S_1



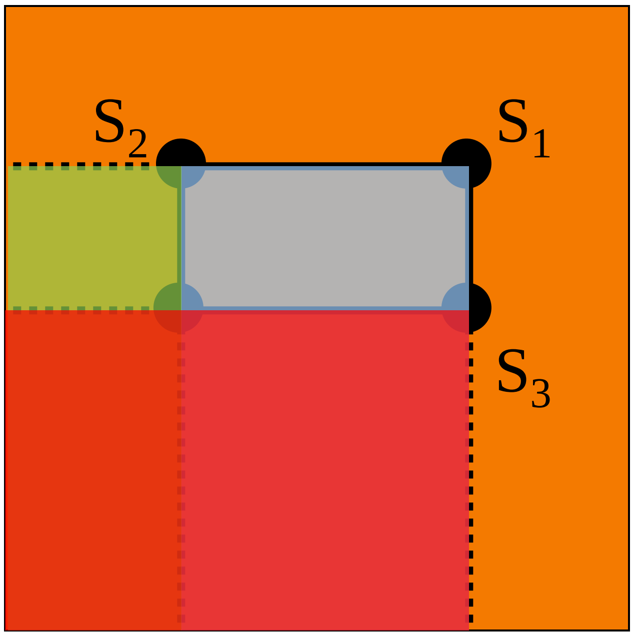
Summed-area tables

- At each texel keep sum of all values down & left
  - To compute sum of all values within a rectangle, simply combine four entries: $S_1 - S$



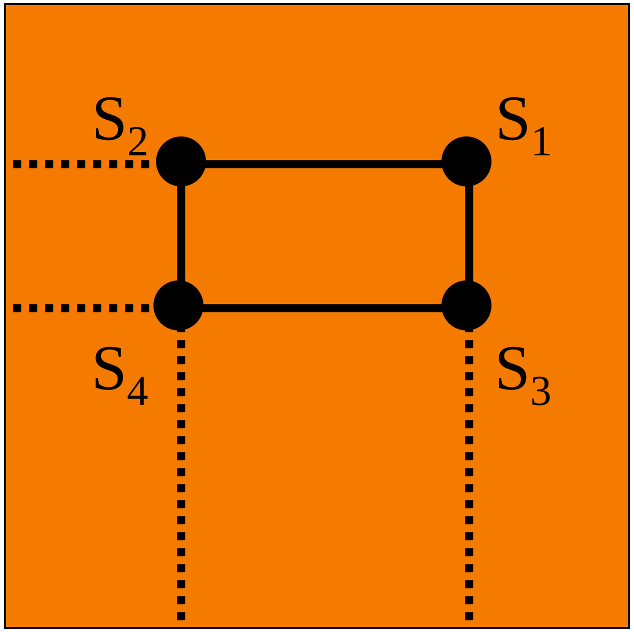
Summed-area tables

- At each texel keep sum of all values down & left
  - To compute sum of all values within a rectangle, simply combine four entries: $S_1 - S_2 - S_3$



Summed-area tables

- At each texel keep sum of all values down & left
  - 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



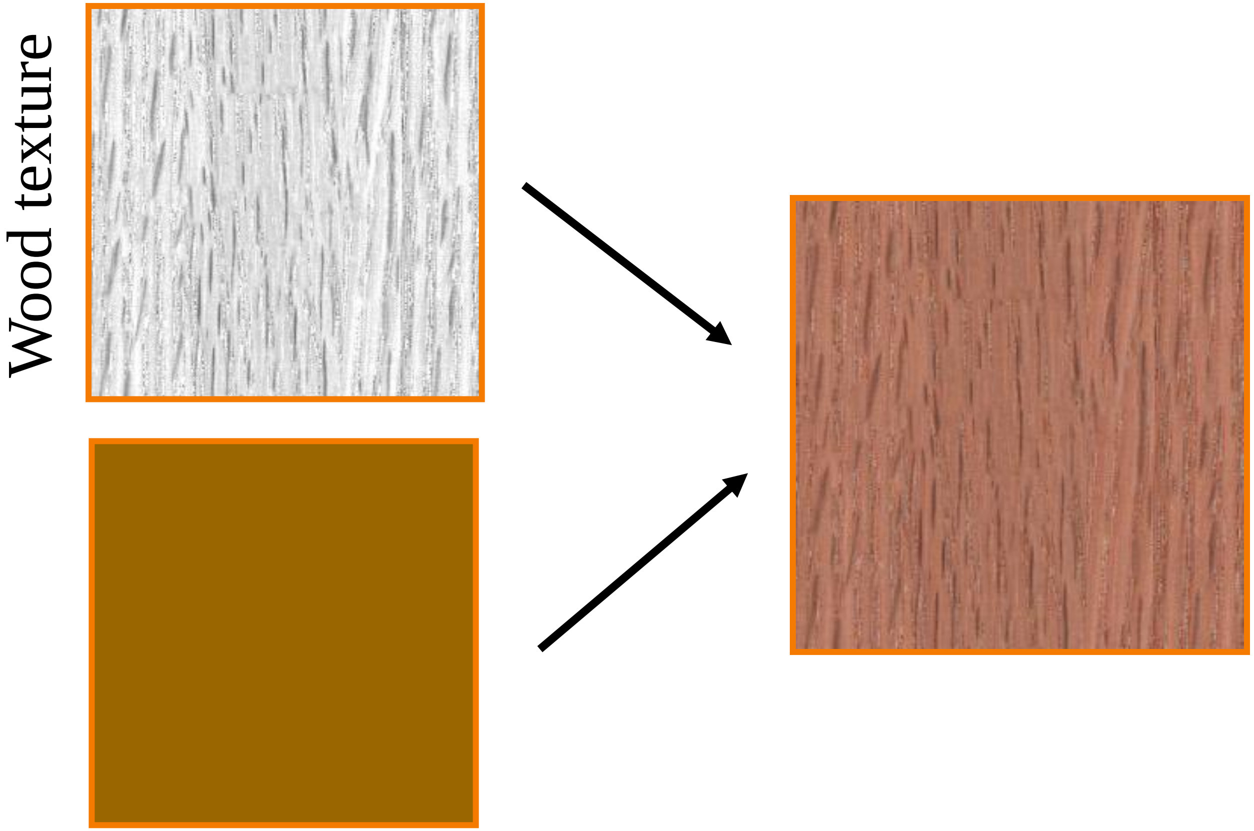
  - (Mipmaps are more common.)

Texture 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



Texture value

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

Illumination Mapping

Map texture values to surface material parameter

- K_A
- K_D
- K_S
- K_T
- n

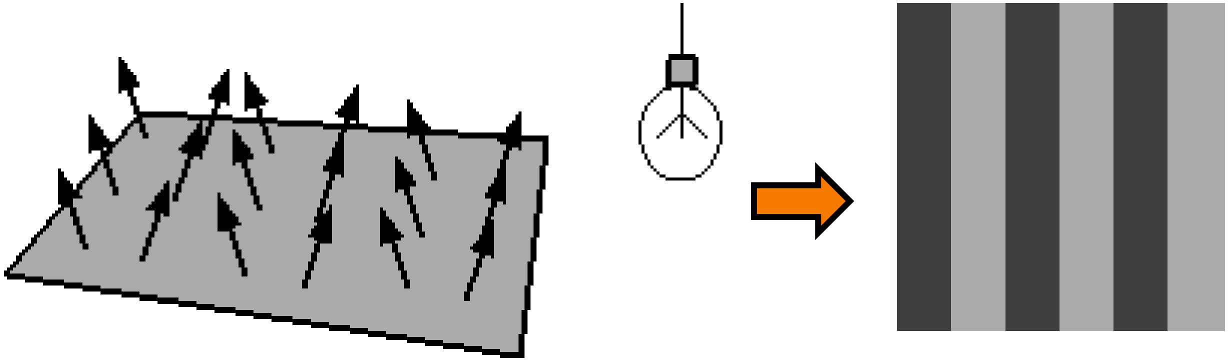
Texture value

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

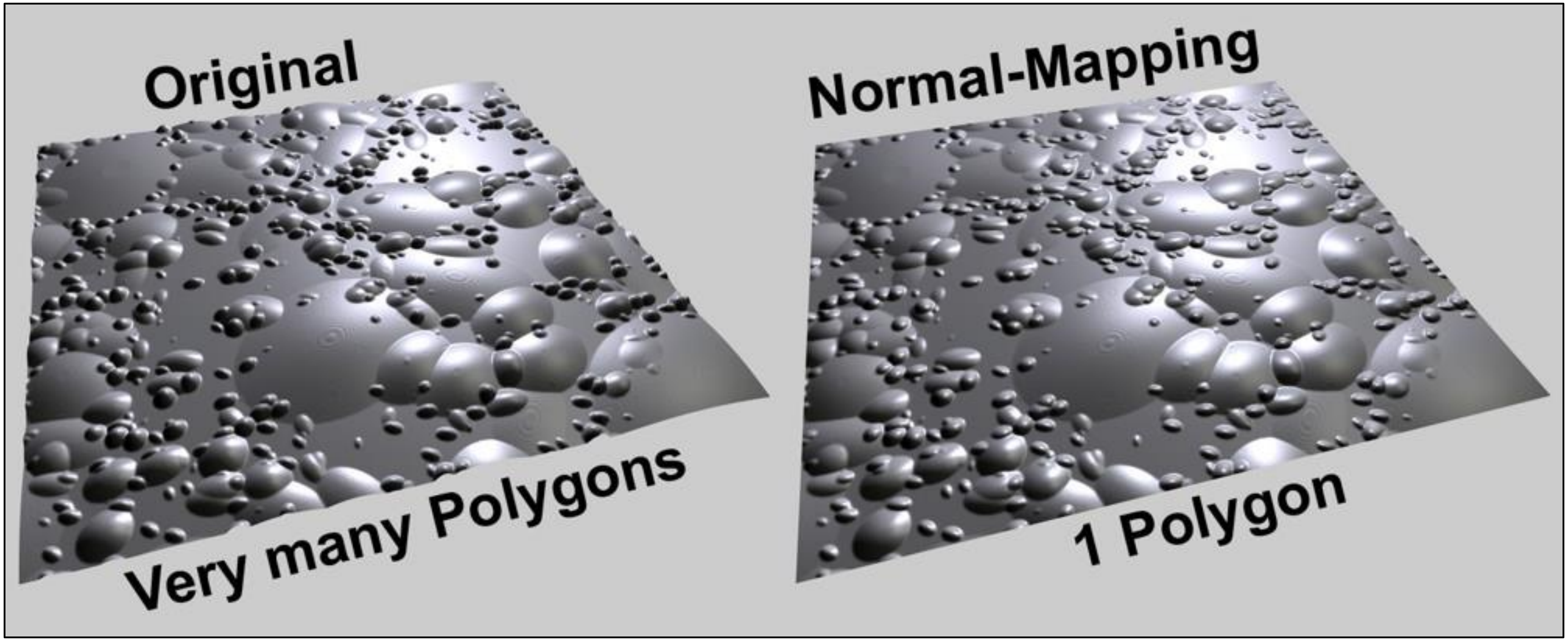
Bump/Normal Mapping

Texture values perturb surface normals:

- Encode normals (or offsets) in RGB
- Use gradient of grayscale image ("bump")

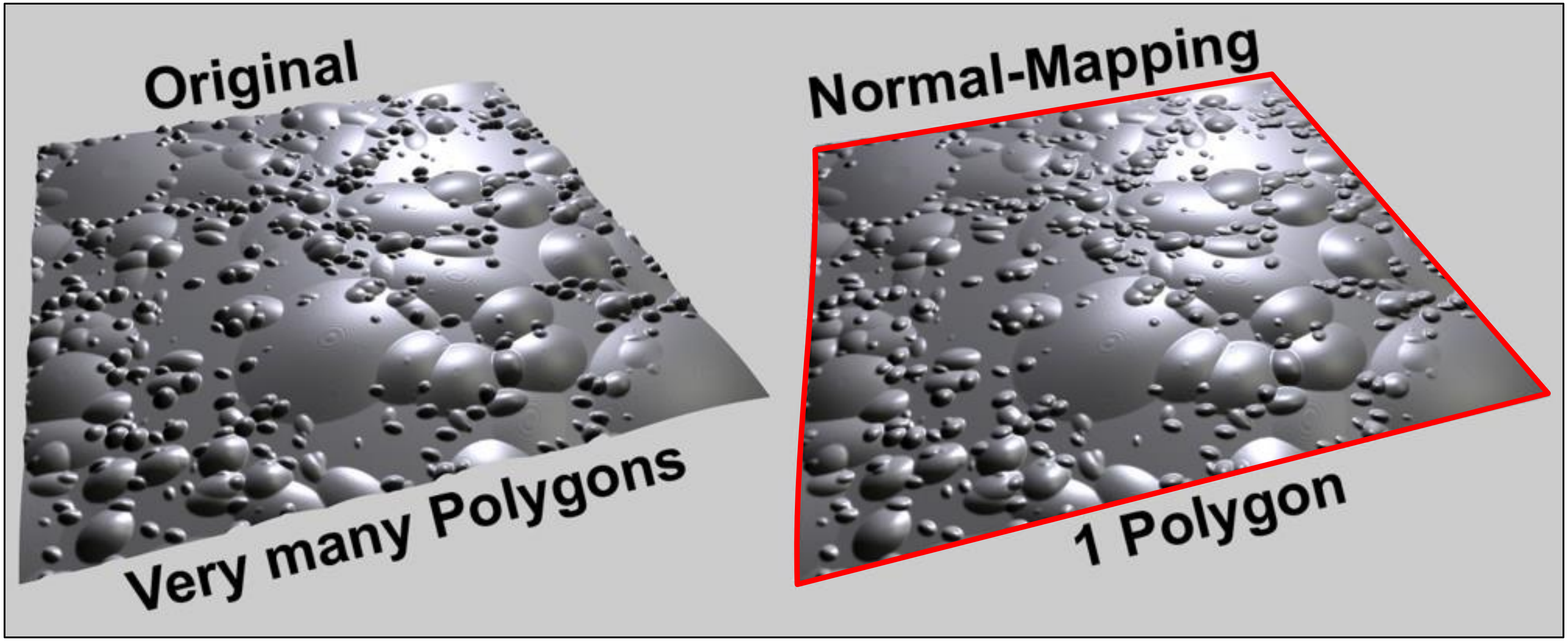


Normal Mapping



Graphisoft.com

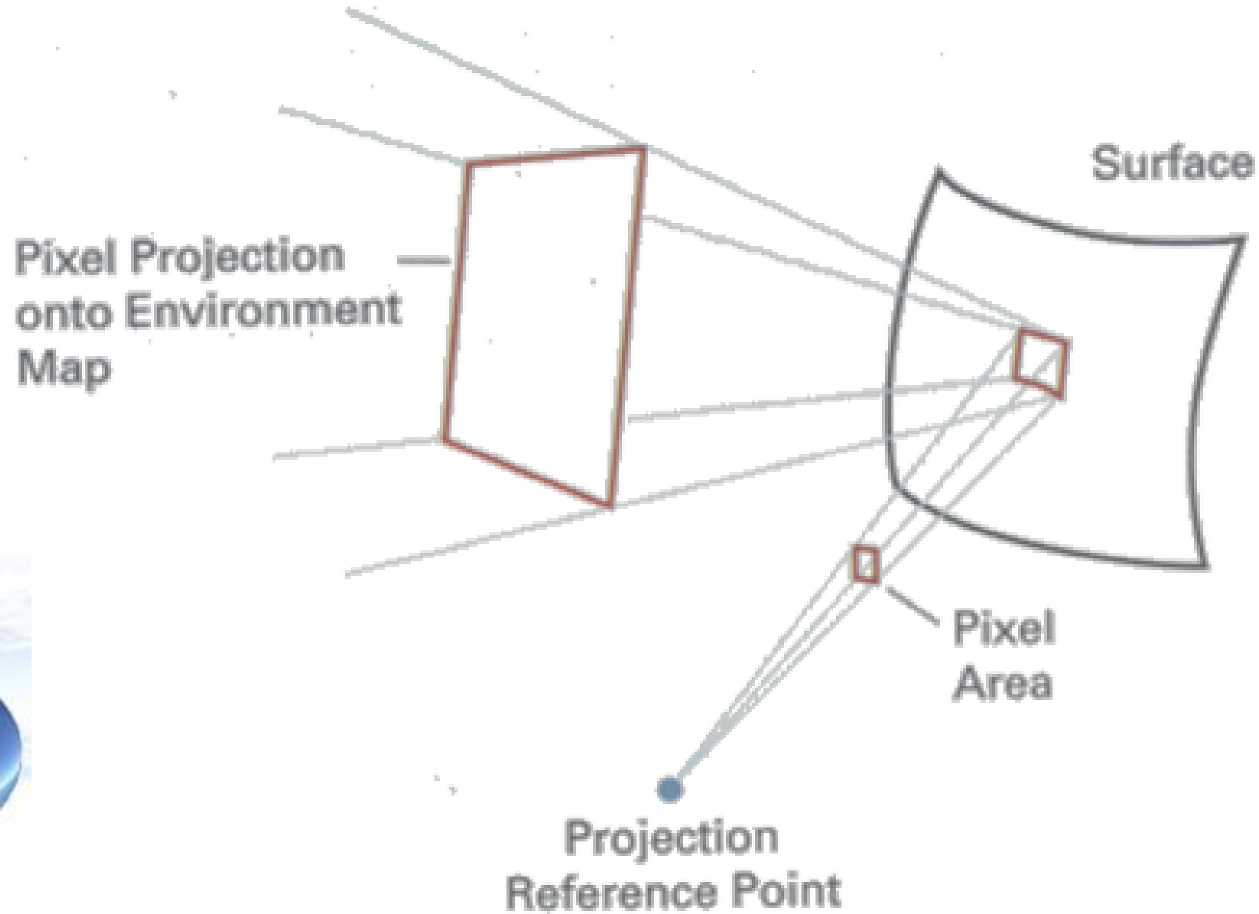
Normal Mapping



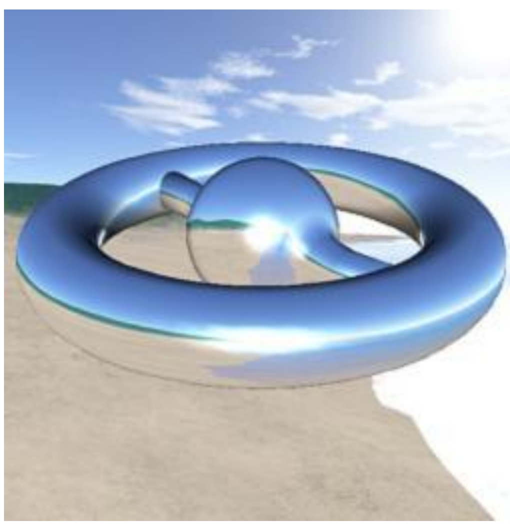
Graphisoft.com

Environment Mapping

Texture values are reflected off surface patch



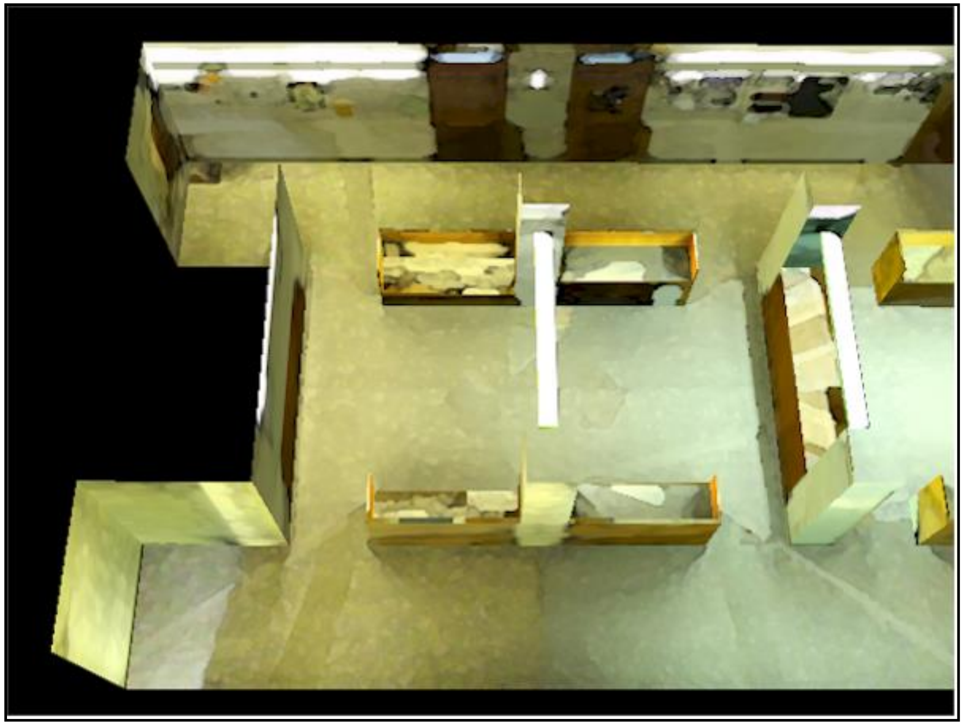
Gamer3D/Wikipedia



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

Solid textures

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

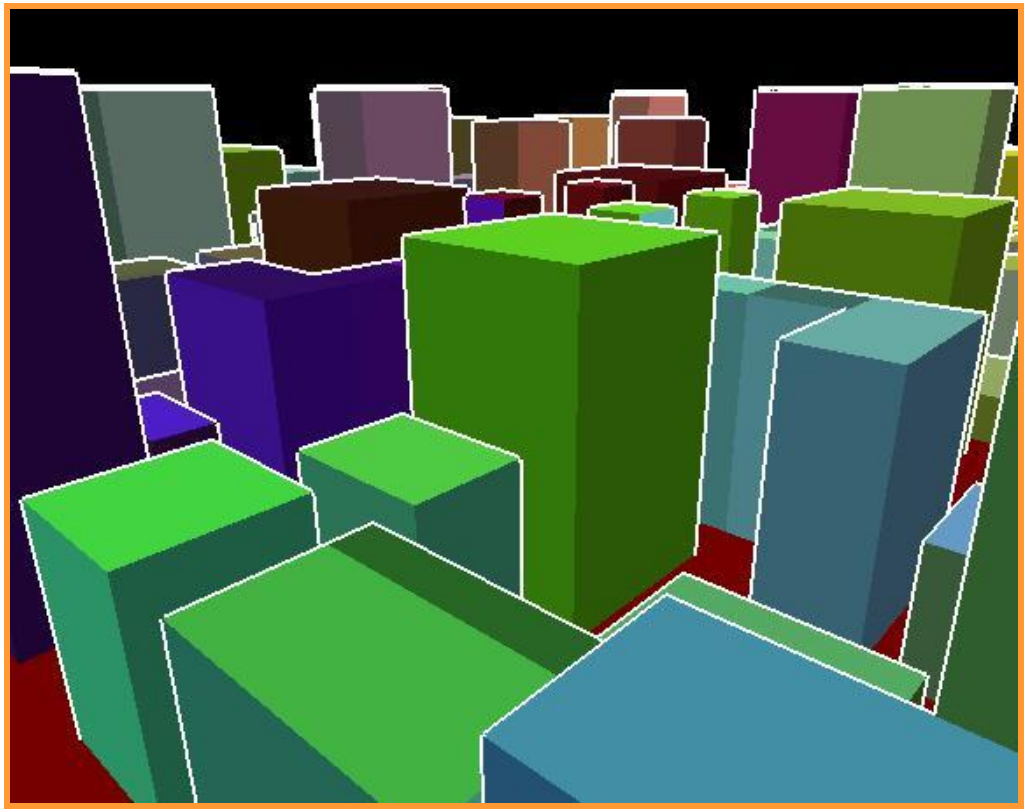
- Expensive storage, or
- Compute on the fly, e.g. Perlin noise →

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

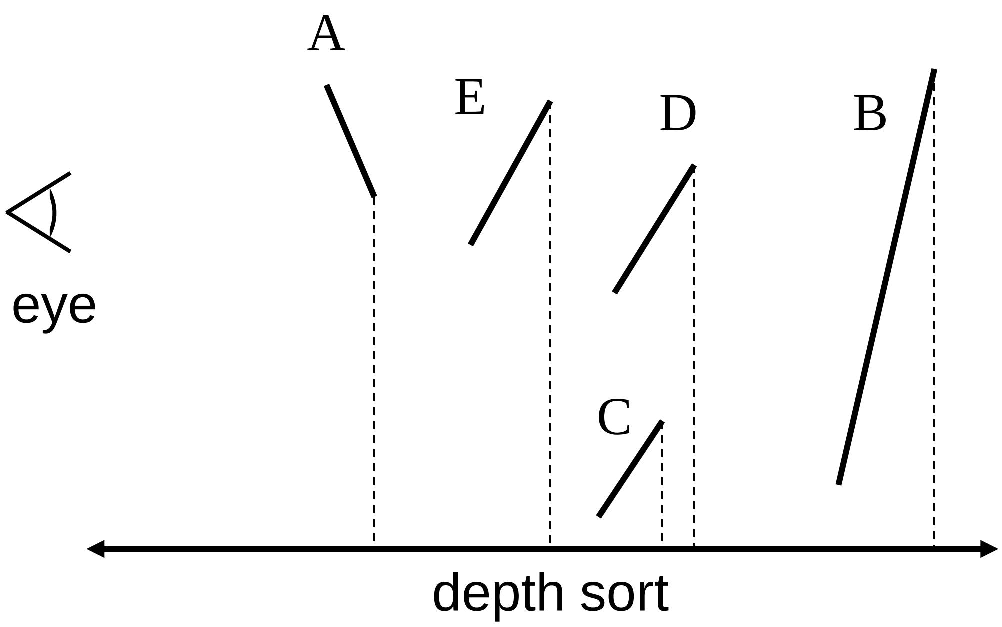
Make sure only front-most surface contributes to color at every pixel

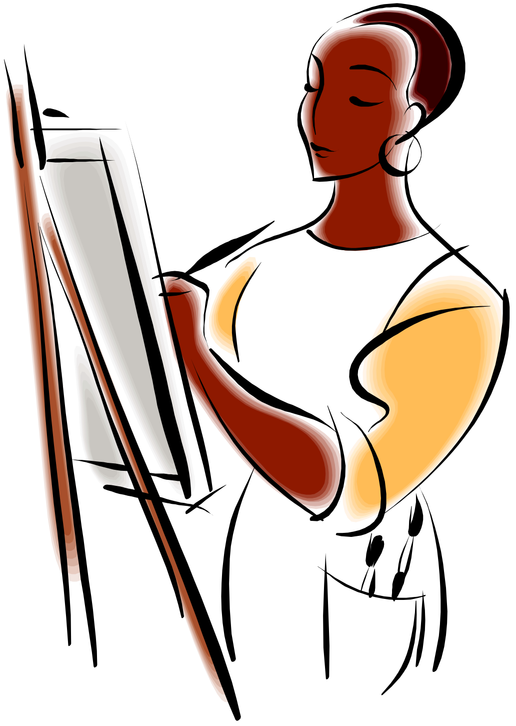


Depth sort

"Painter's algorithm"

1. First **sort surfaces** in order of decreasing *maximum* depth

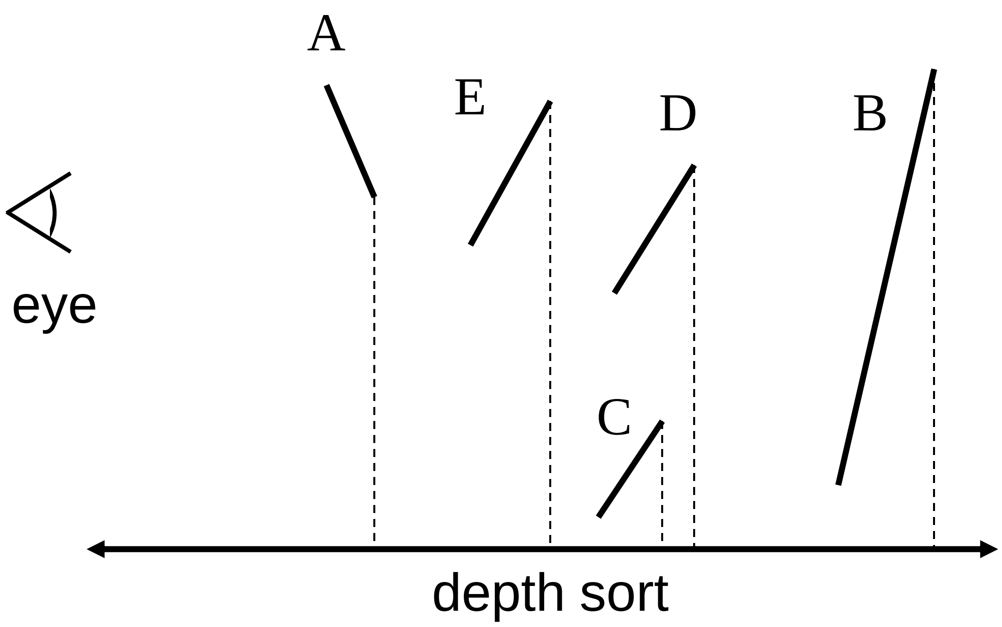


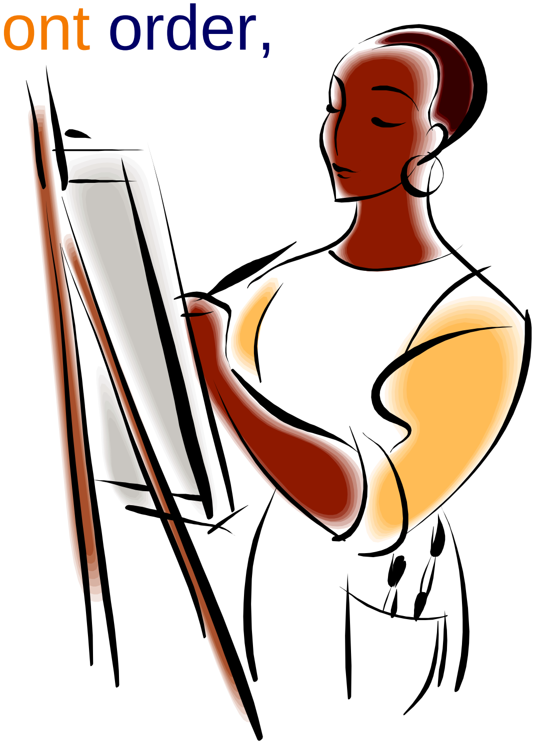


Depth sort

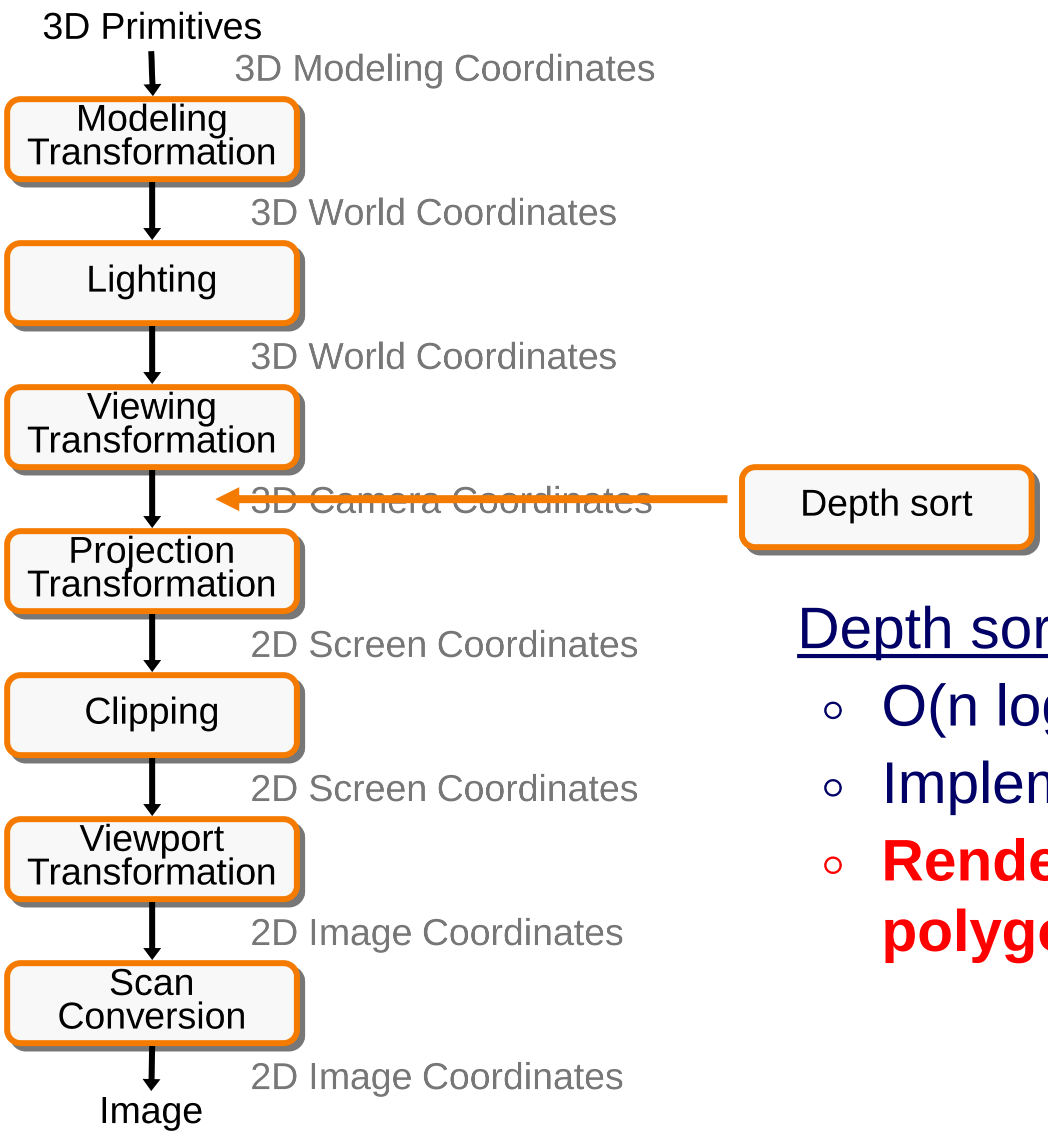
"Painter's algorithm"

1. First **sort surfaces** in order of decreasing *maximum* depth
2. Scan convert surfaces in back-to-front order, overwriting pixels





3D Rendering Pipeline



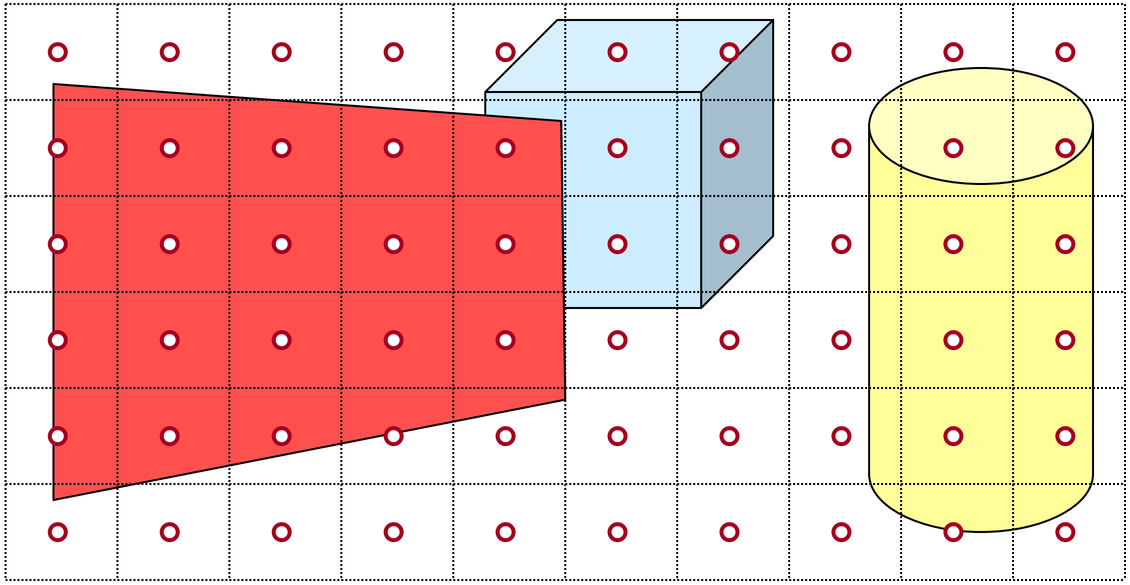
Depth sort comments

- O(n log n)
- Implemented in software
- **Render pixels of every polygon**

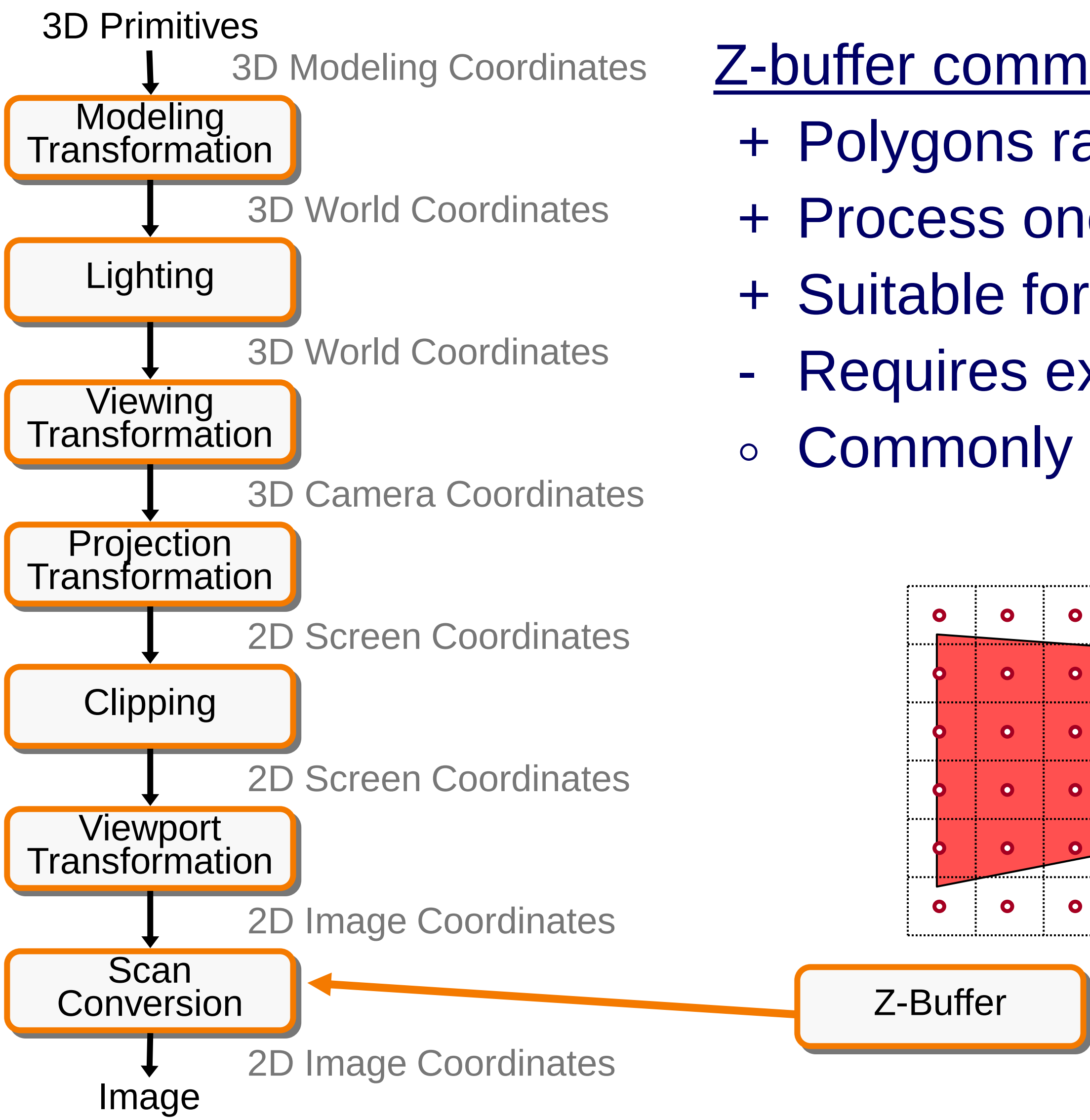
Z-Buffer

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 for in-primitive pixels from vertices, just like colors

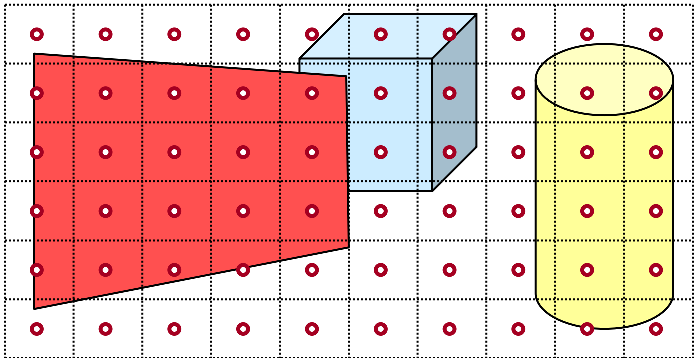


Z-Buffer



Z-buffer comments

- \+ Polygons rasterized in any order
- \+ Process one polygon at a time
- \+ Suitable for hardware pipeline
- \- Requires extra memory for z-buffer
- ○ Commonly in hardware



Hidden Surface Removal Algorithms

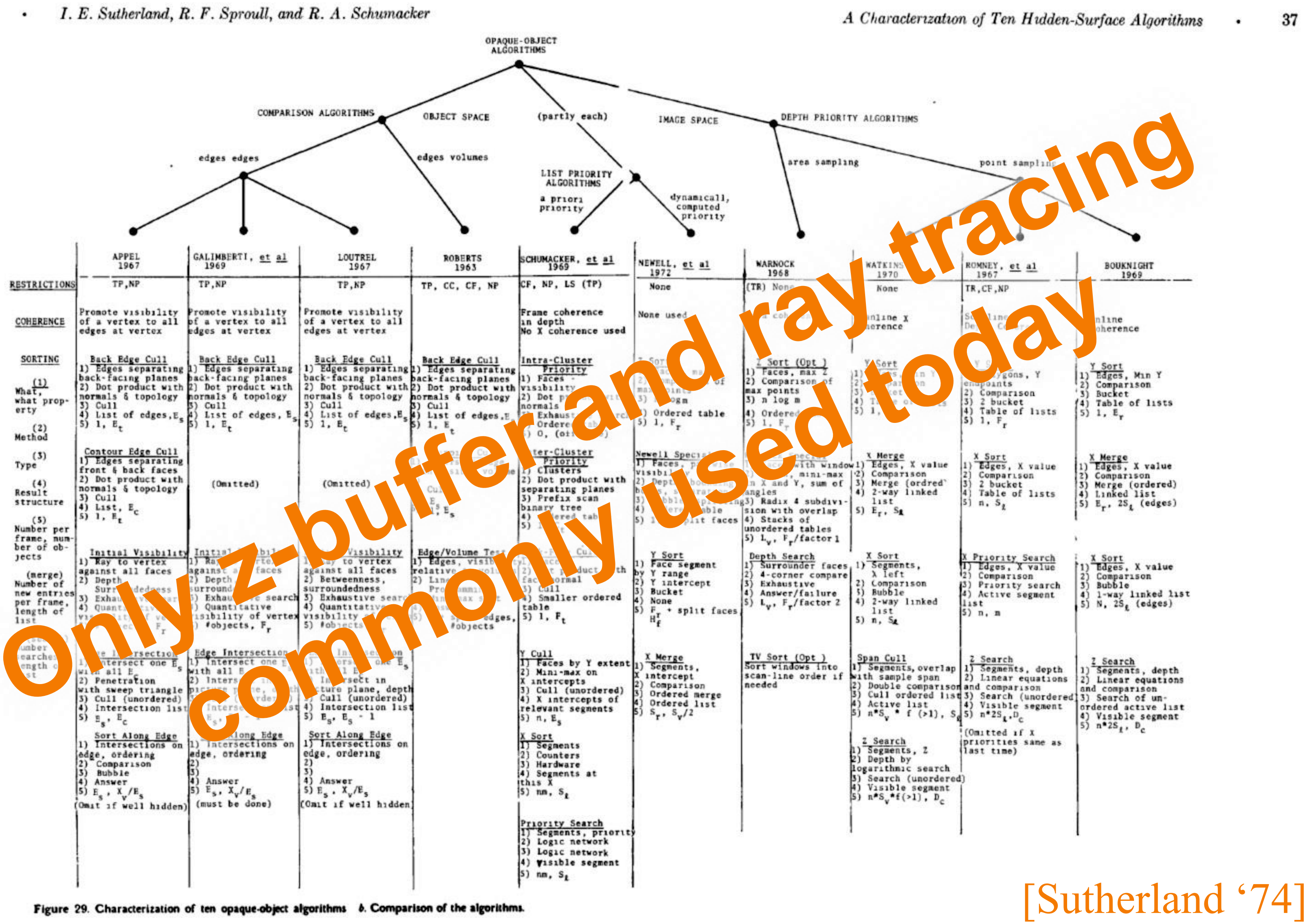


Figure 29. Characterization of ten opaque-object algorithms b. Comparison of the algorithms.

Only z-buffer and ray tracing commonly used today

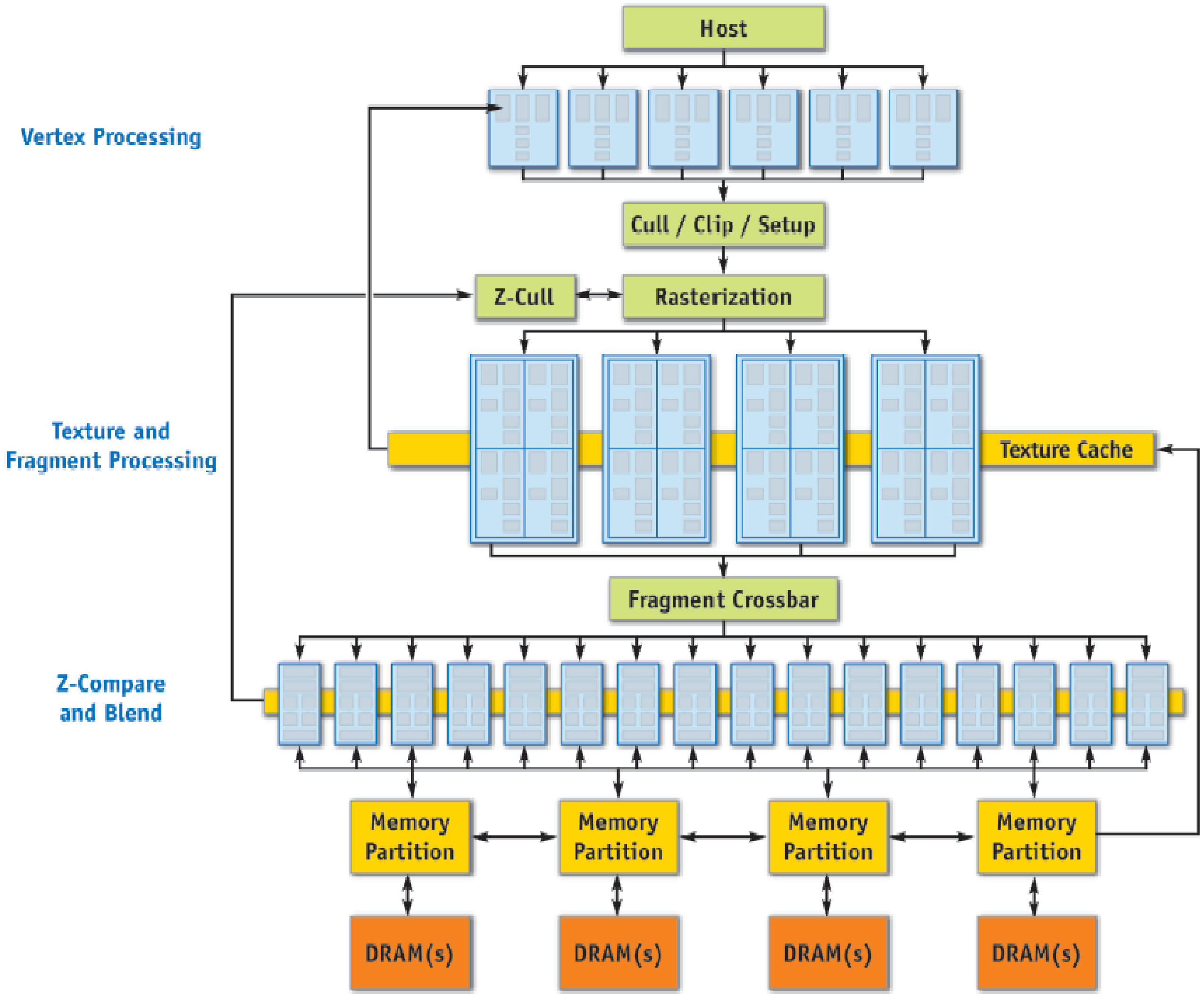
[Sutherland '74]

Rasterization Summary

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

This is all in hardware

GPU Architecture

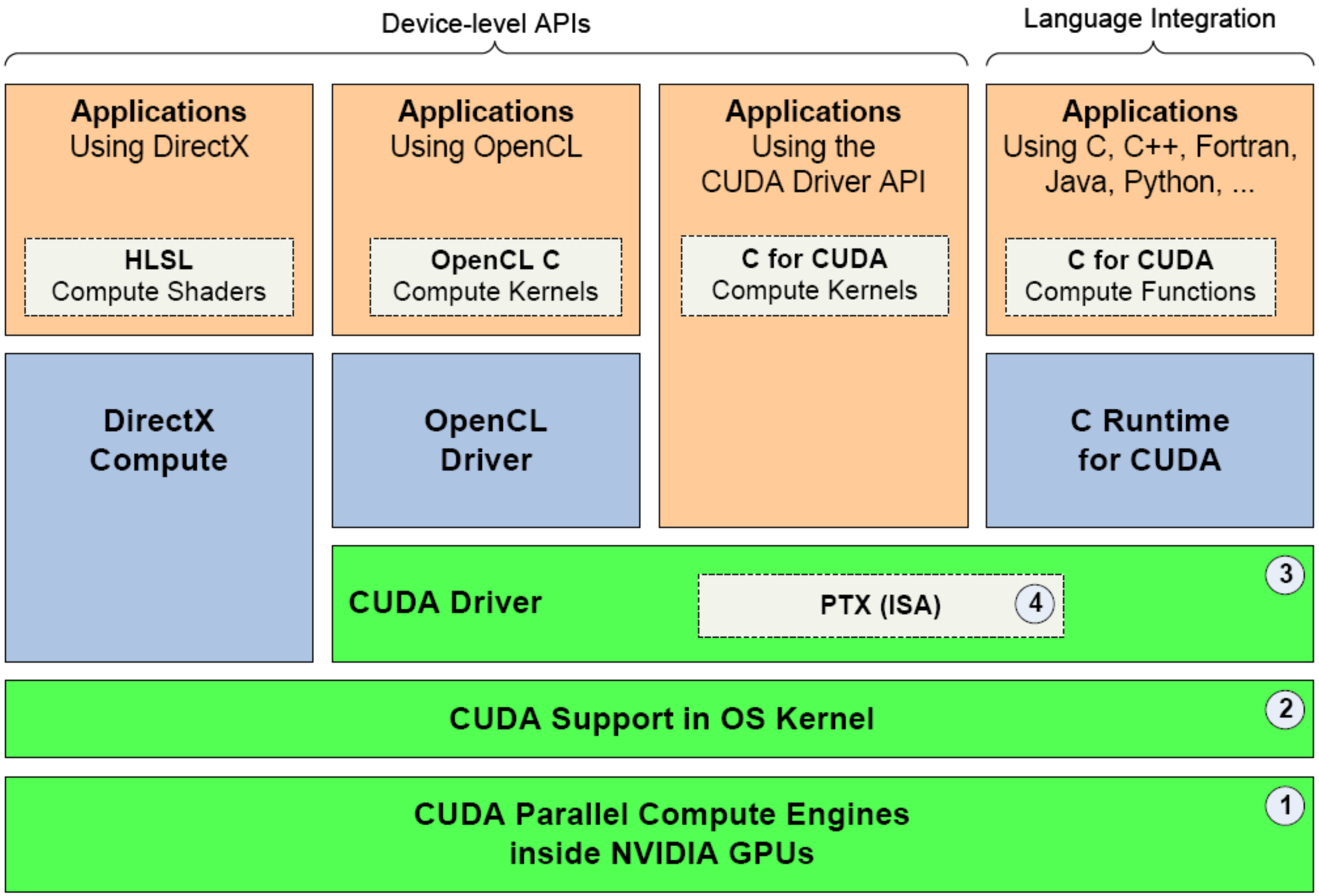


GeForce 6 Series Architecture

GPU Gems 2, NVIDIA

Actually …

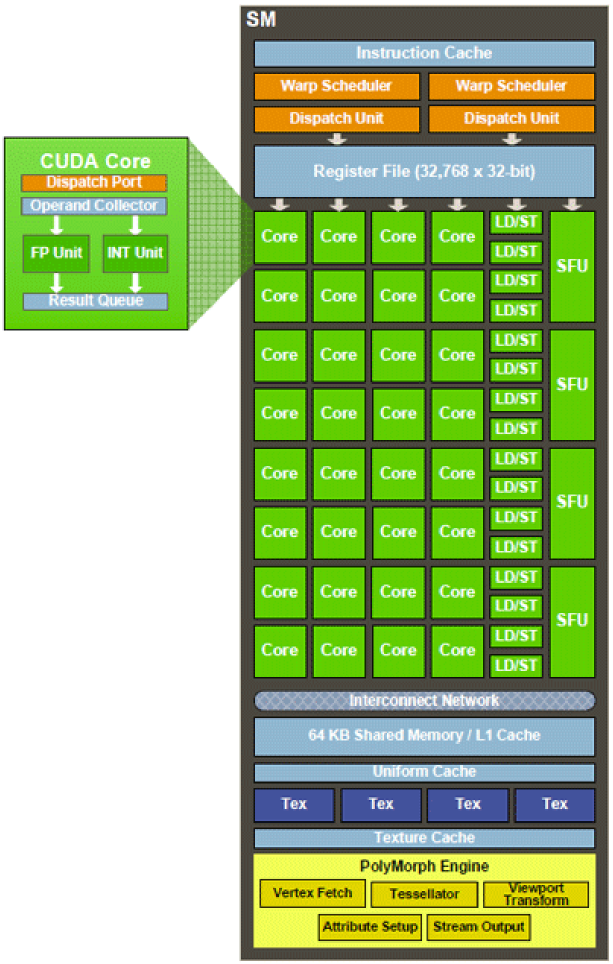
- Modern graphics hardware is programmable

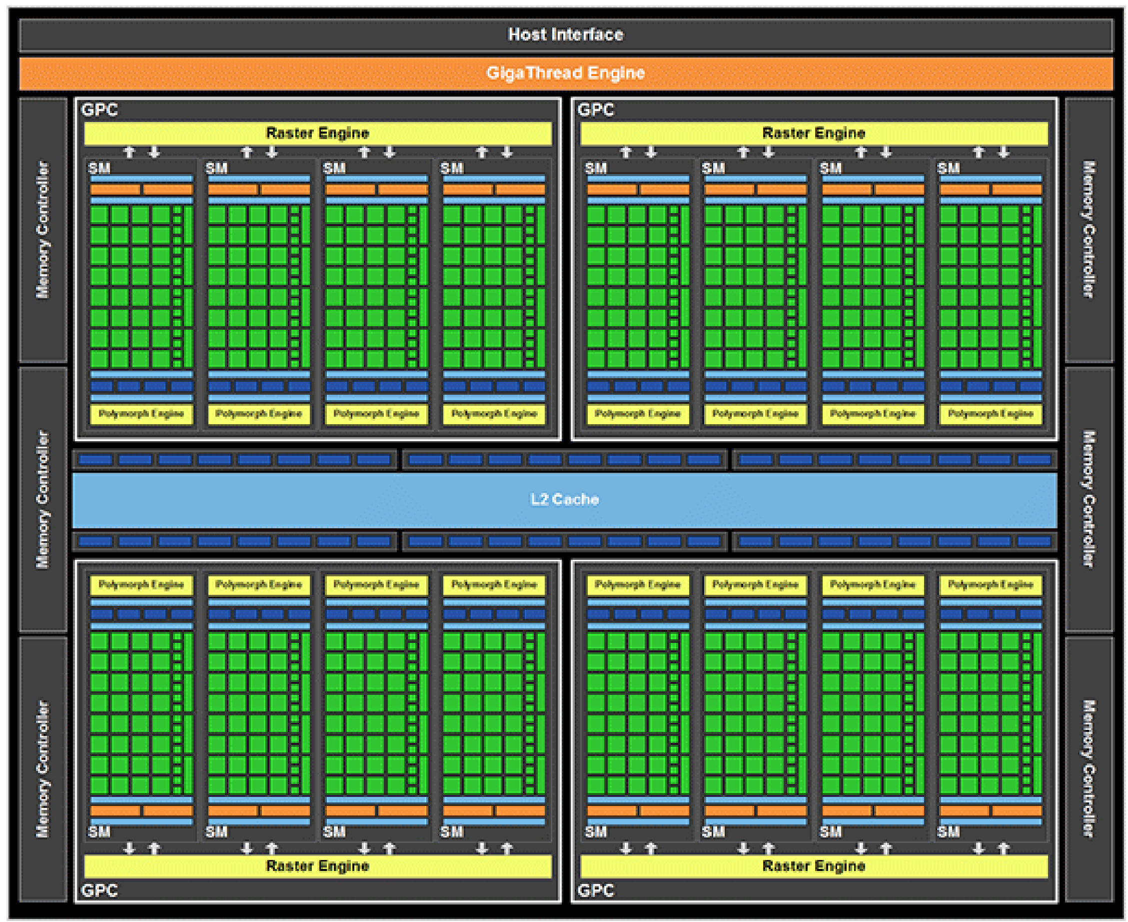


www.nvidia.com/cuda

Trend …

- GPU is general-purpose parallel computer





www.nvidia.com/cuda