Non-photorealism

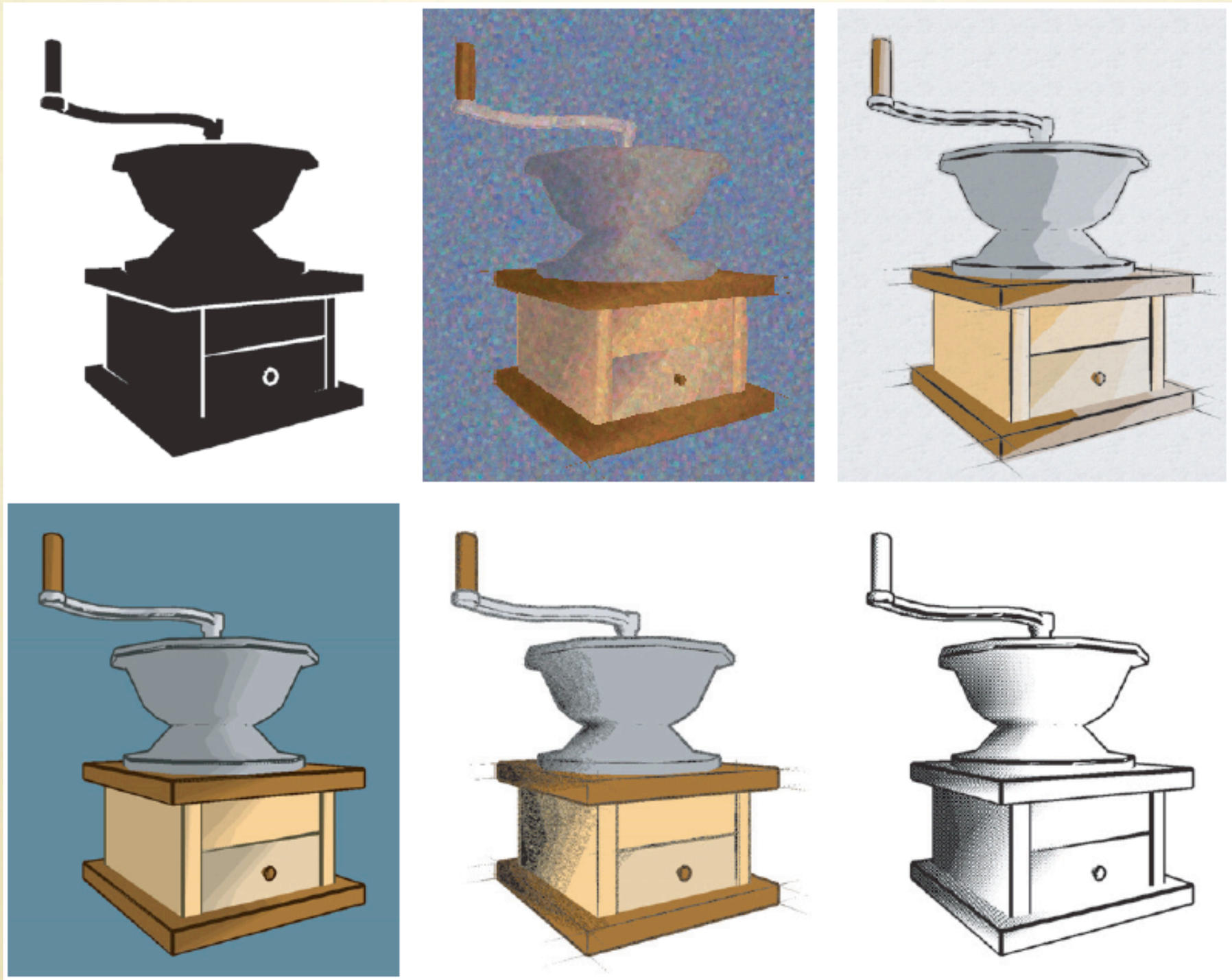
Christian Miller

CS 354 - Fall 2011

Different goals

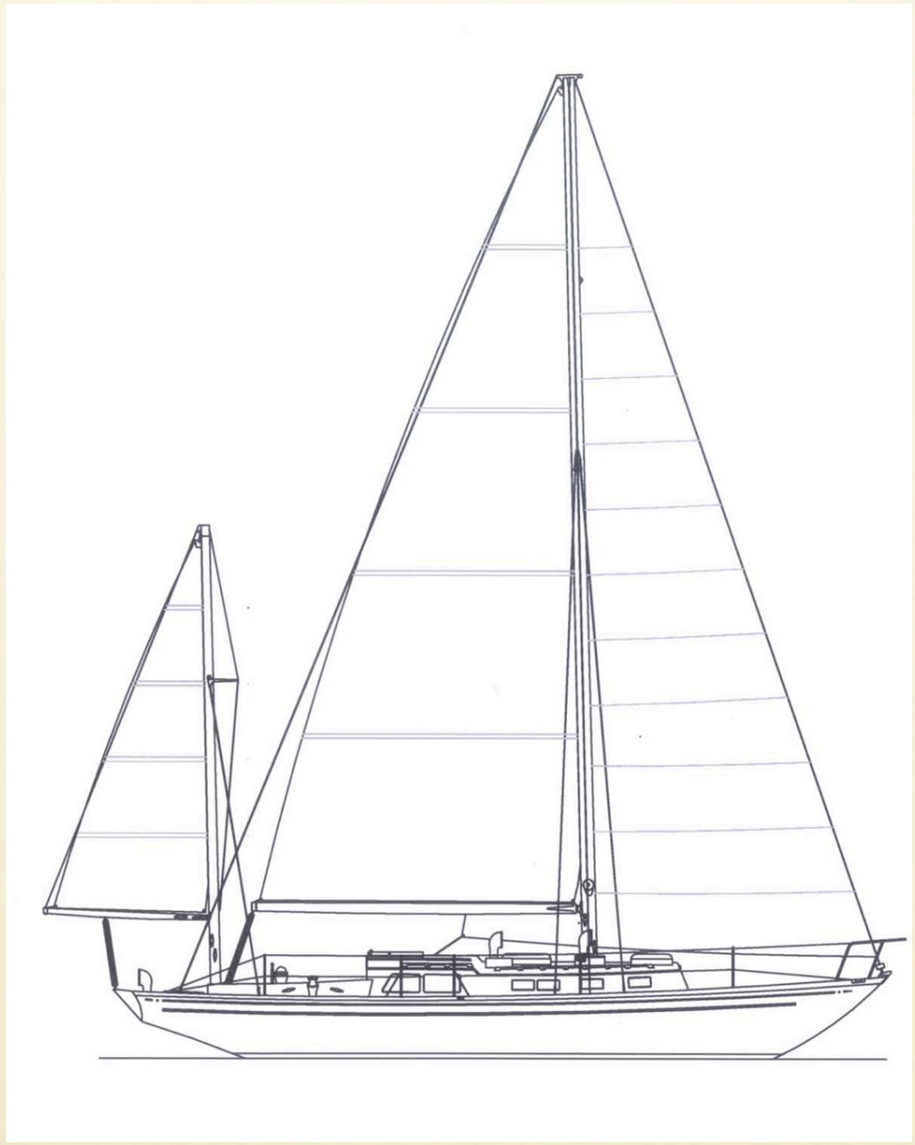
- Everything we've done so far has been working (more or less) towards photorealism
- But, you might not want realism as a stylistic choice
  - In fact, you usually don't want it...
- This is where non-photorealistic rendering (NPR) comes in

Like this



[RTR]

Line drawing



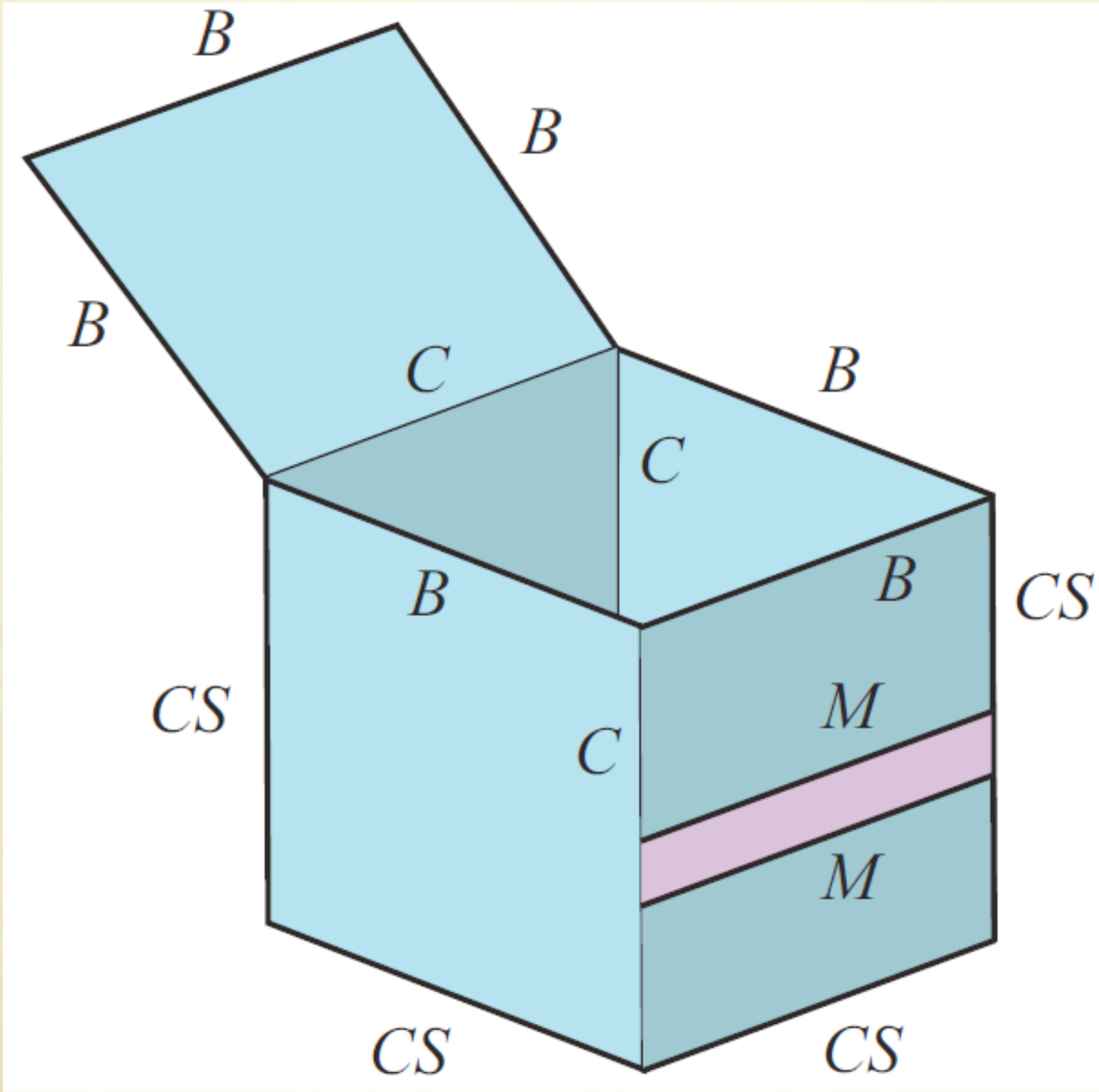
- Draw an object just using lines, no surfaces
- Often used in blueprints

[WP]

Edges to enhance

- Every object has a ton of edges to it
- Which edges you want to draw lines for depends on the type of edge it is:
  - Silhouette edges: the visible outline of an object
  - Crease edges: sharp angles within the outline
  - Boundary edges: hard edges of objects
  - Material edges: places where the color changes

Types of edges

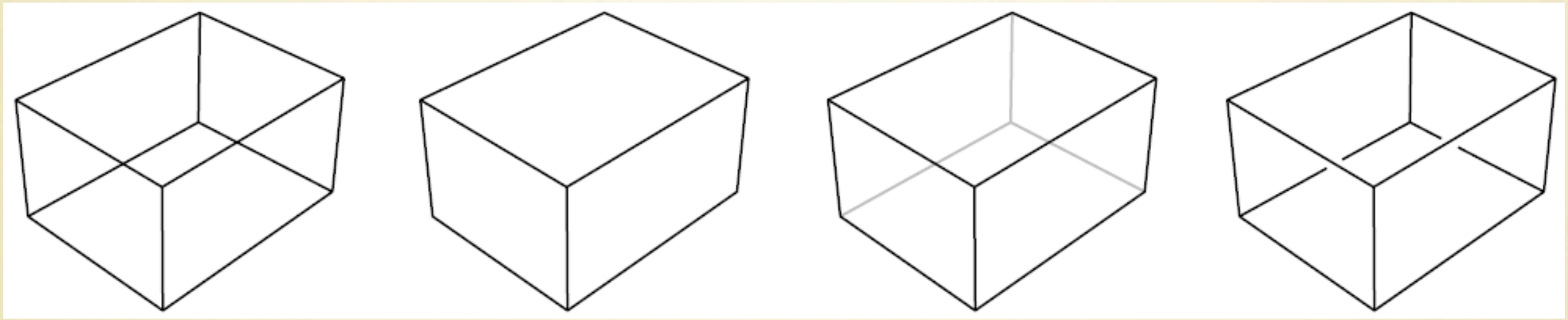


[RTR]

Finding mesh edges

- Material: picked a-priori during authoring
- Boundary: edges with only one attached triangle
- Crease: edges where the angle between the adjacent triangles is under some threshold
- Silhouette: edges where one triangle is front-facing and the other is back-facing (found at runtime)

Rendering modes

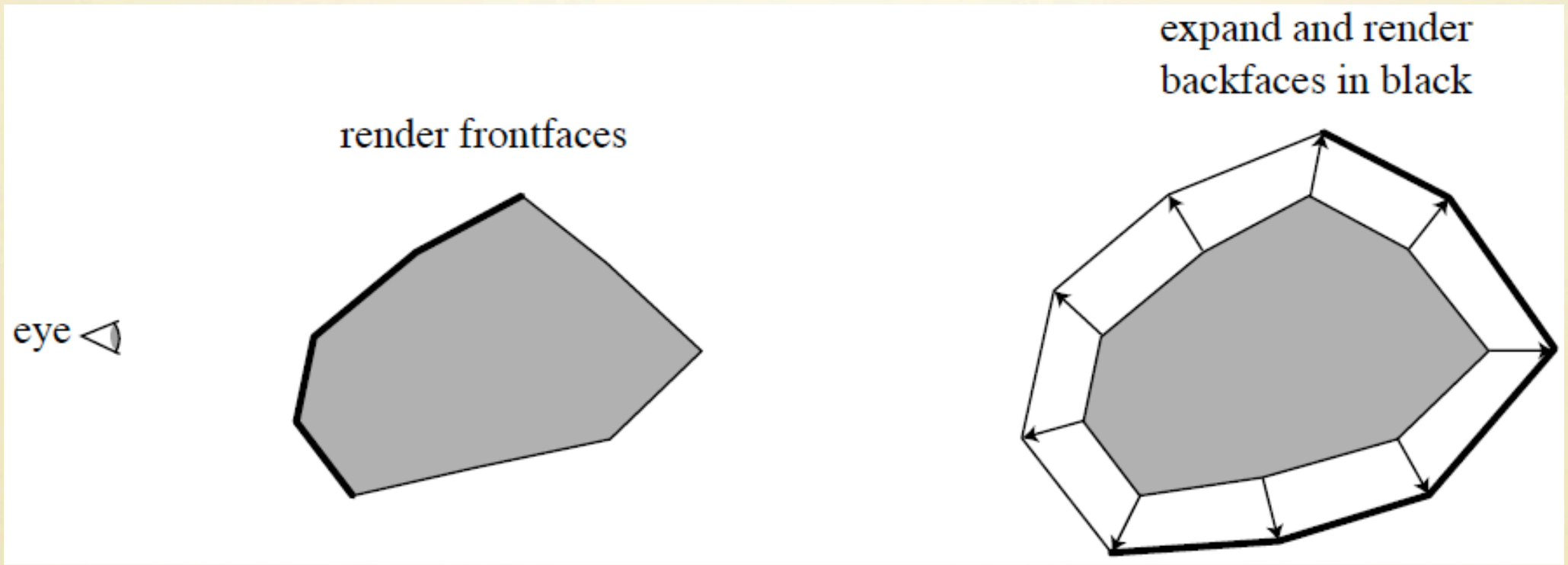


- Hidden line rendering becomes more difficult, since there are no faces to objects
- Usually draw all face geometry into just depth buffer, then draw biased lines with depth test enabled

Silhouette edges

- Silhouettes are expensive, since you need to test every edge on every frame
- Would be nice if there was some other way to render the edges
- Depth buffer / depth test can help us out

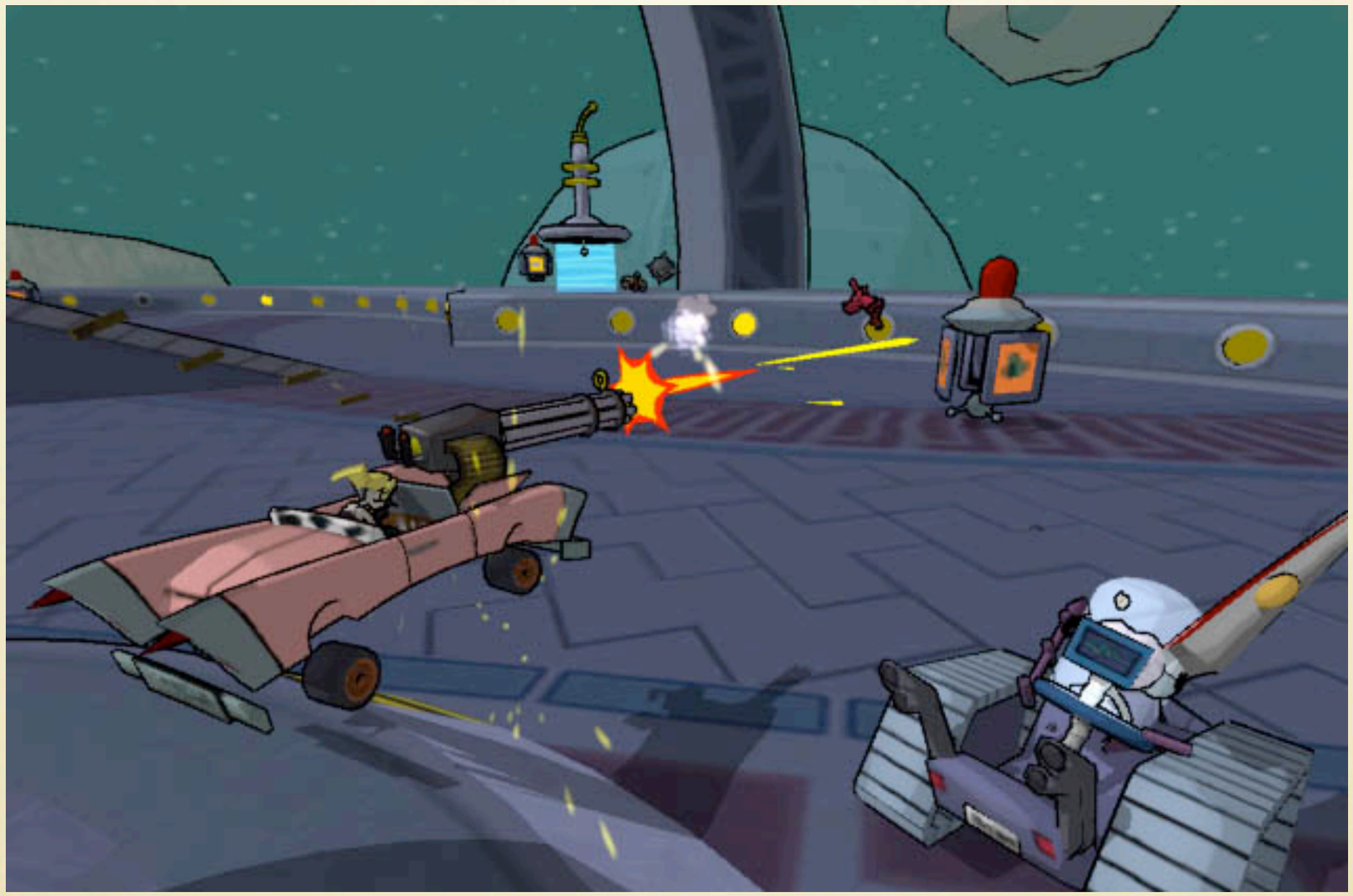
Cheap outline hack



- If your object is closed, simply expand the backfaces
- Doesn't guarantee consistent line width, only works on silhouettes

[RTR]

Example



[Cel Damage]

Filter-based methods

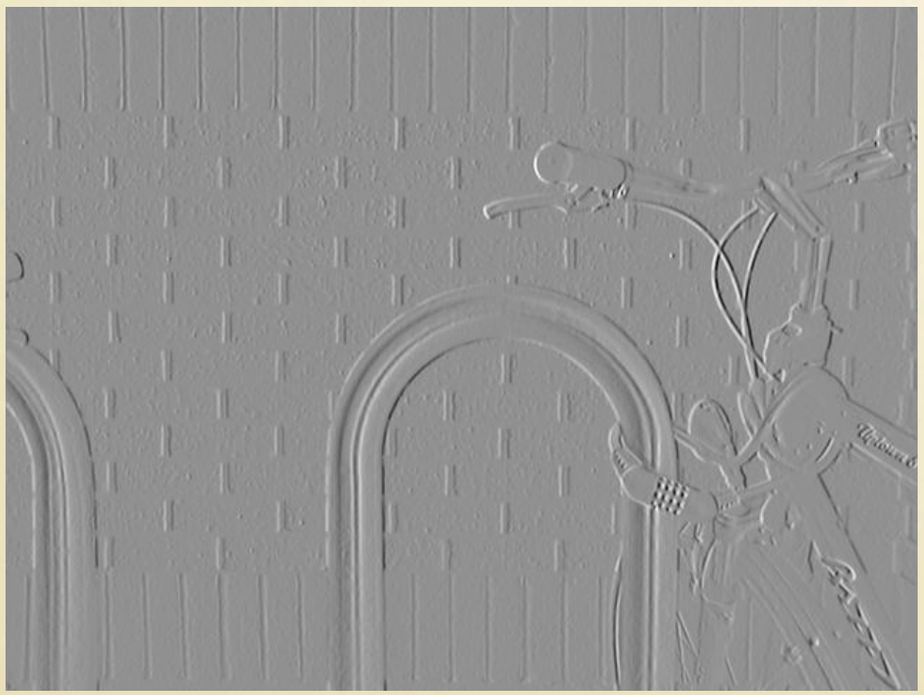
- Another approach: run a bunch of image filters
- Render image to depth buffer and normal buffer
- Draw a quad over the screen, have pixel shader sample all the buffers and render lines as appropriate

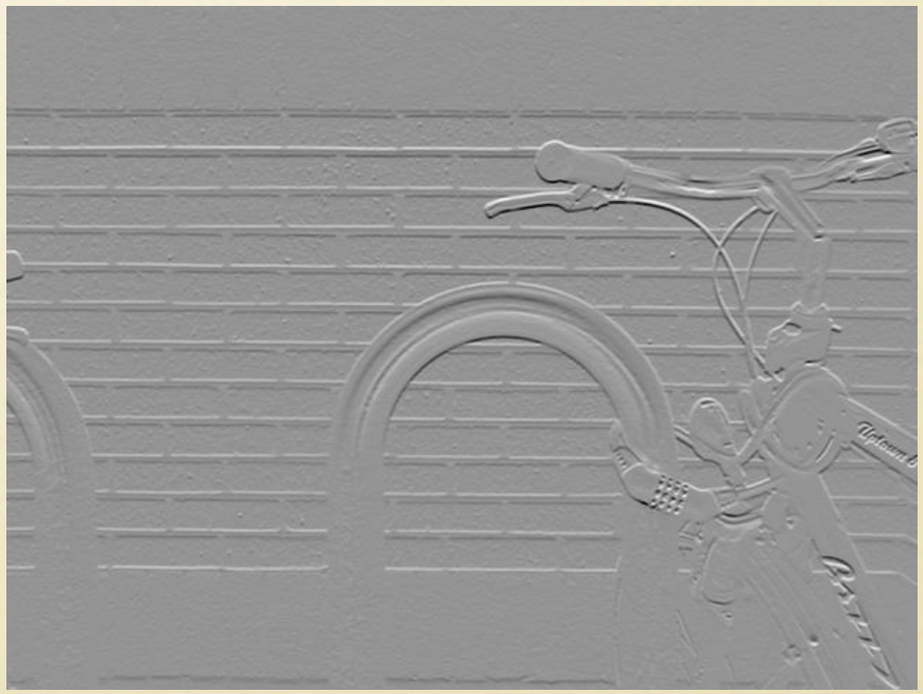
Detecting edges

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}$$

- Image filter: sample nearby pixels, then do some math on them to get a result
    - Technically, this is a 2D convolution operation
- Sobel filter: a common direction-dependent edge detection filter

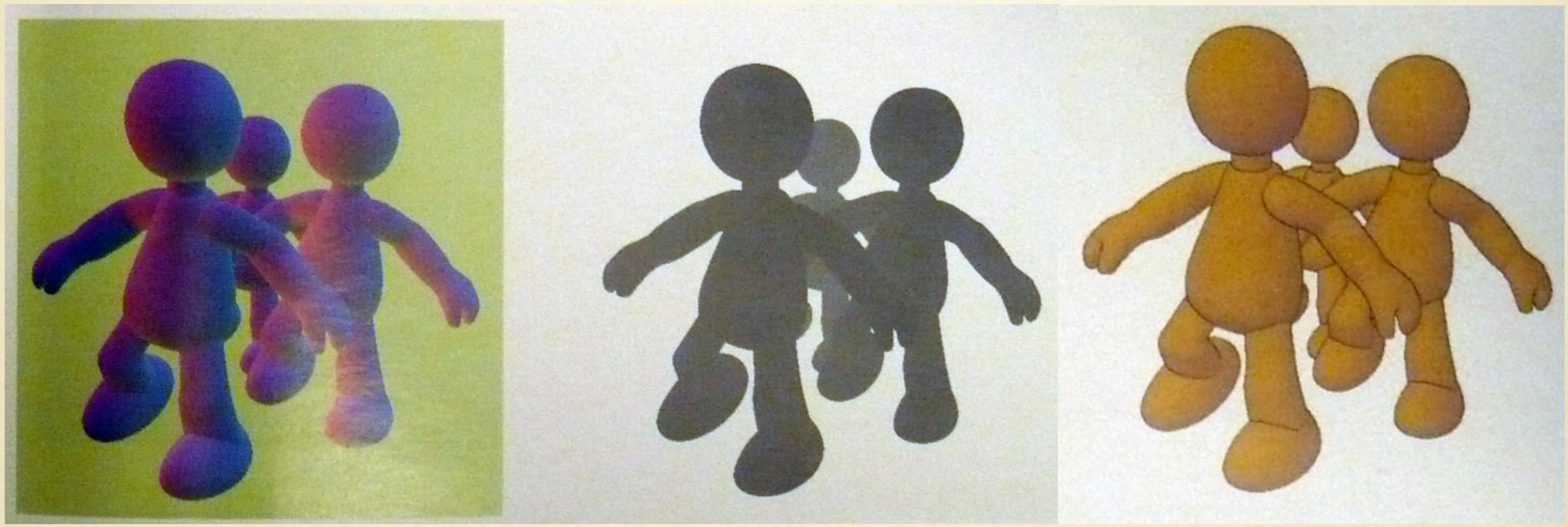
Sobel results





[WP]

Image-based edges



- Normal, depth, extracted edges from both

[SX]

Image-based edges

[Borderlands]

Brush strokes

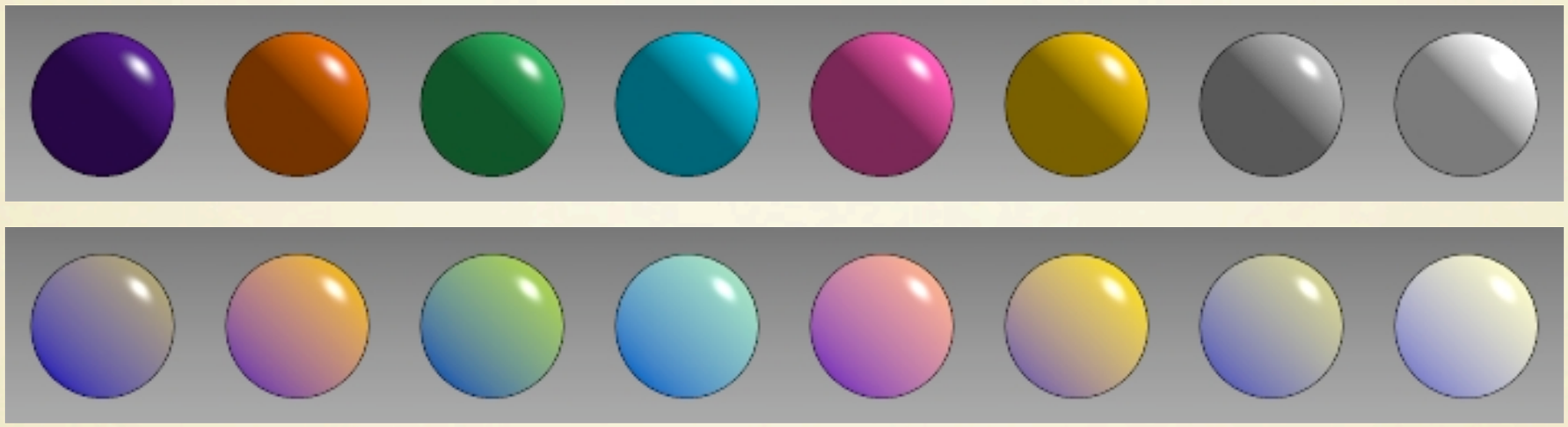
- If you string together extracted edges, you can render them as brush strokes
- Doesn't look natural in motion, but interesting nonetheless

[NM]

Shading

- Outlines are only part of the image, what about shading?
- We generally try to imitate other artistic styles, piggybacking on top of machinery we already have for graphics

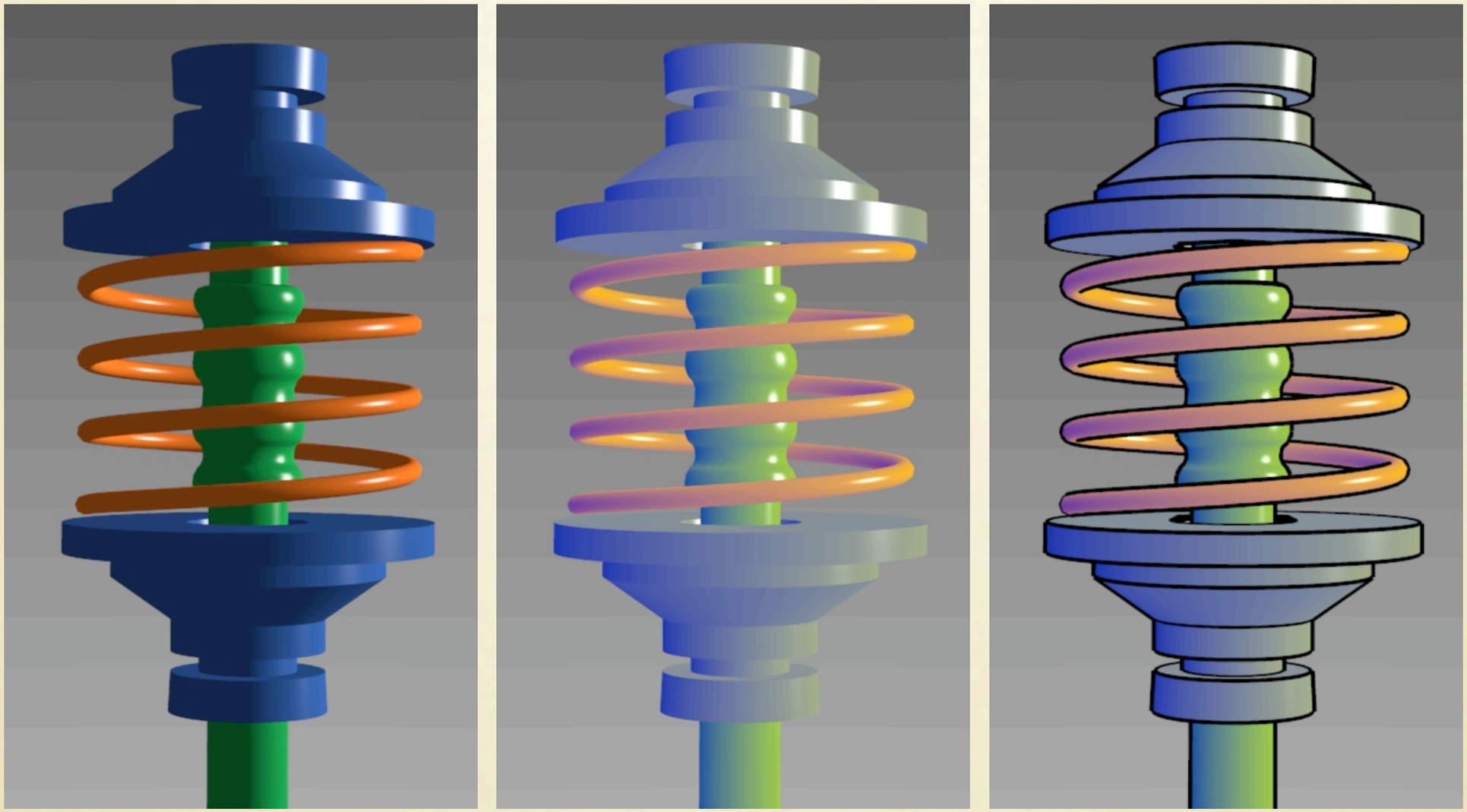
Gooch shading



- Start with diffuse shading, use result to look up a color in a gradient between a warm and cool color
- Fade continues into shadow side (negative N • L)
- Add Phong highlight if desired

[BG]

Gooch shading



- Great for technical illustration, not as harsh looking as regular shading

[BG]

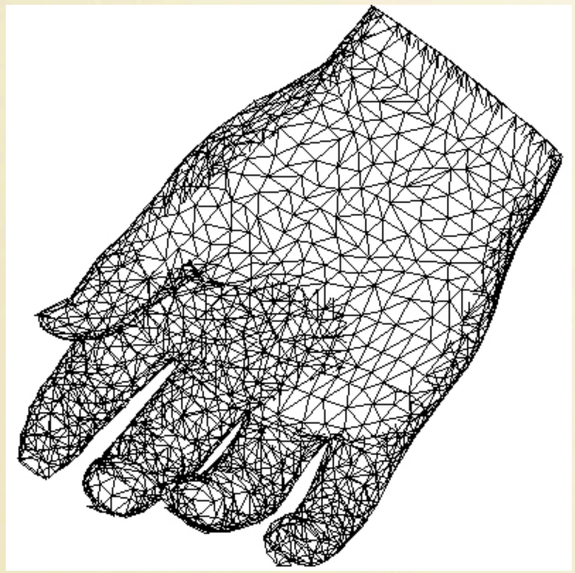
Cel shading

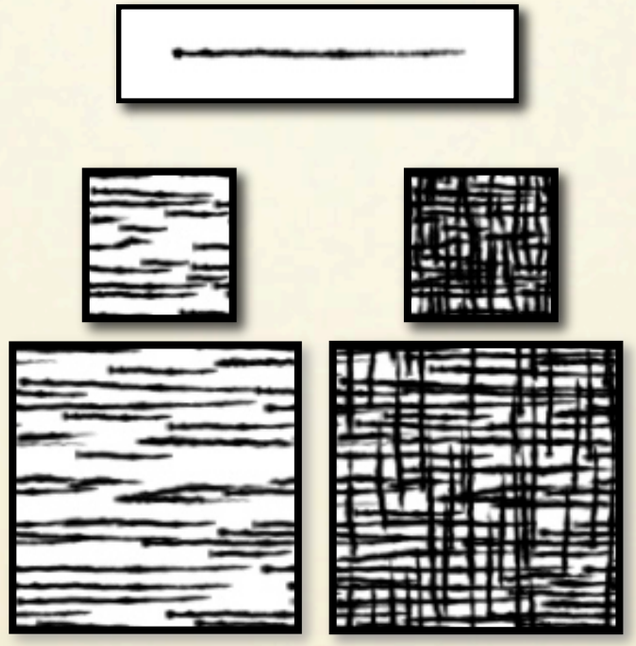
- Instead of using (N • L) to multiply diffuse color directly, assign constant brightness values to several ranges
- At runtime, use (N • L) to look up brightness to use, then continue with lighting equation as normal
- Gives a hand-drawn look, which is how it gets its name

Cel shading

[Jet Grind Radio,
Zelda: Wind Waker]

Textured shading

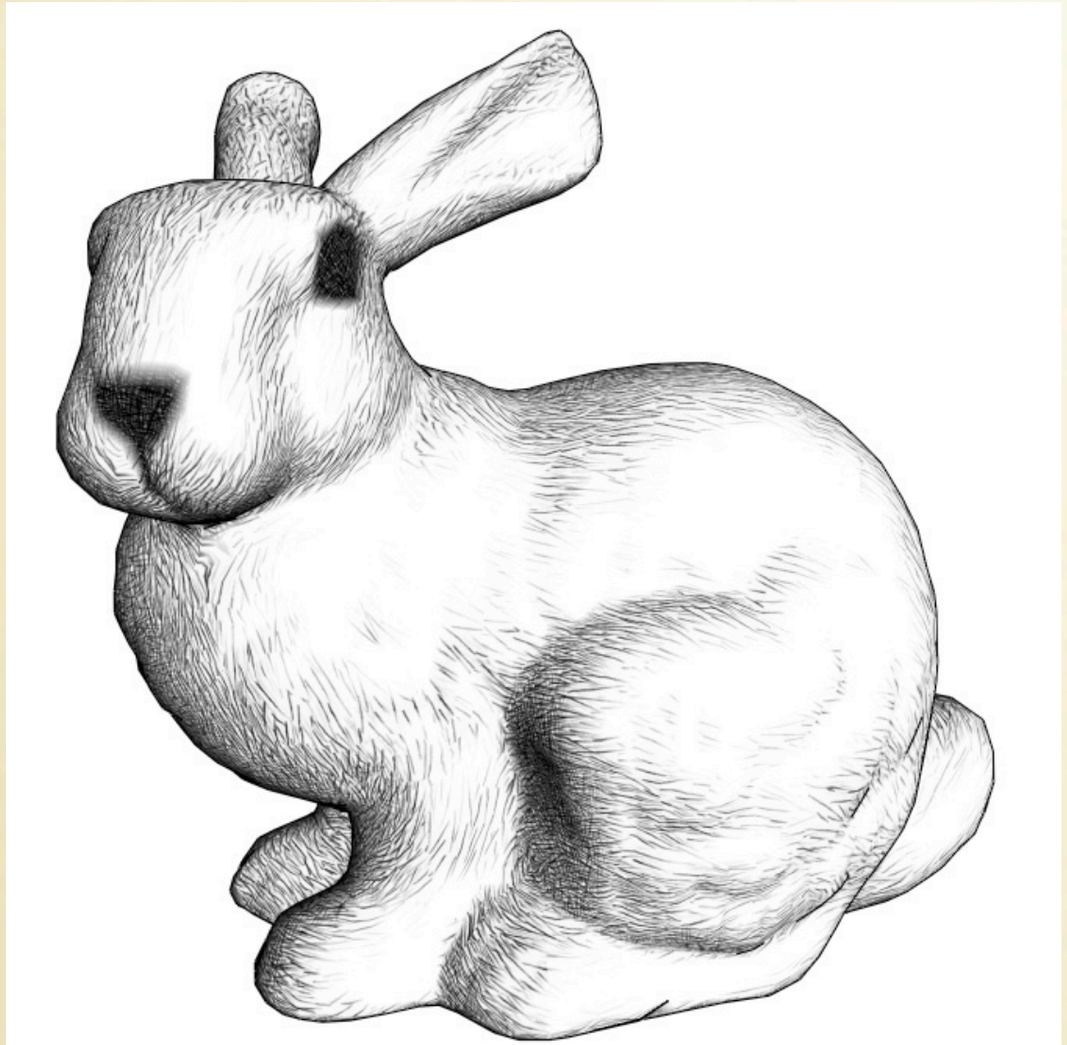


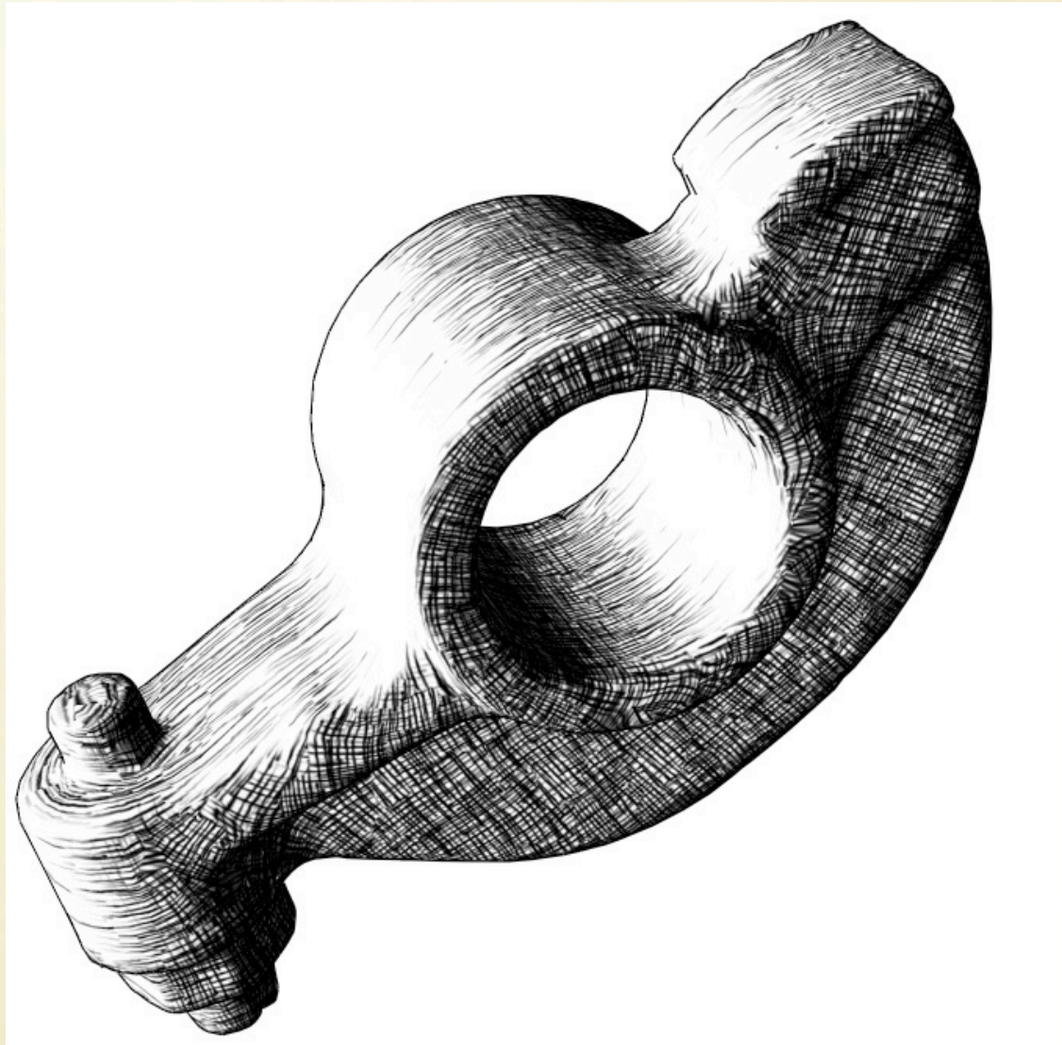


- Another idea: use (N • L) to look up which texture to use during shading
- Can have several textures with different "darkness" that are blended together to get the final result

[RH]

Textured shading

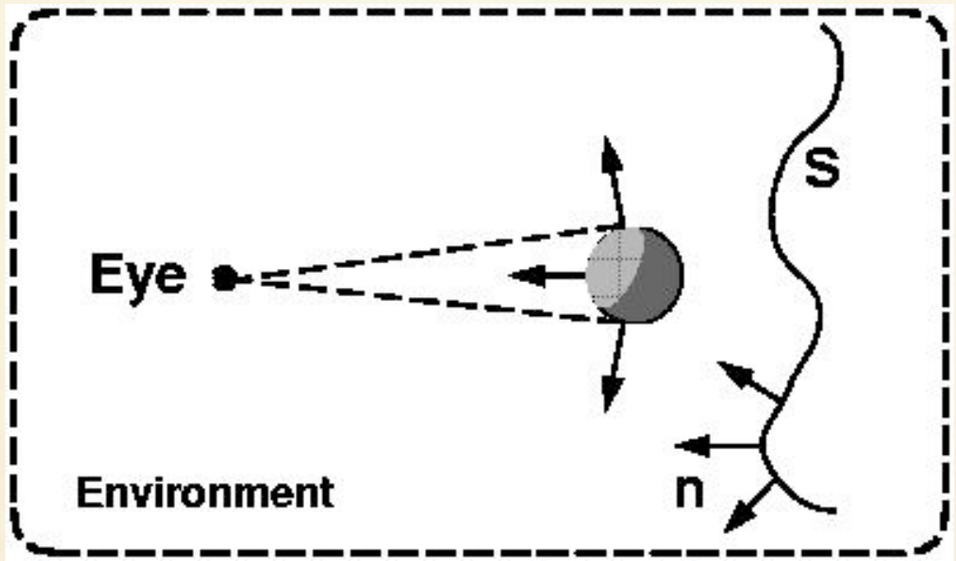




- Lots of details to get this to work right...

[RH]

Lit sphere



- Take a picture of a sphere lit the way you want it
  - Or it could be hand drawn
- Render a different mesh, and do paint-by-normal: if a position has a certain normal, find out where that normal would be on the sphere and take the corresponding color

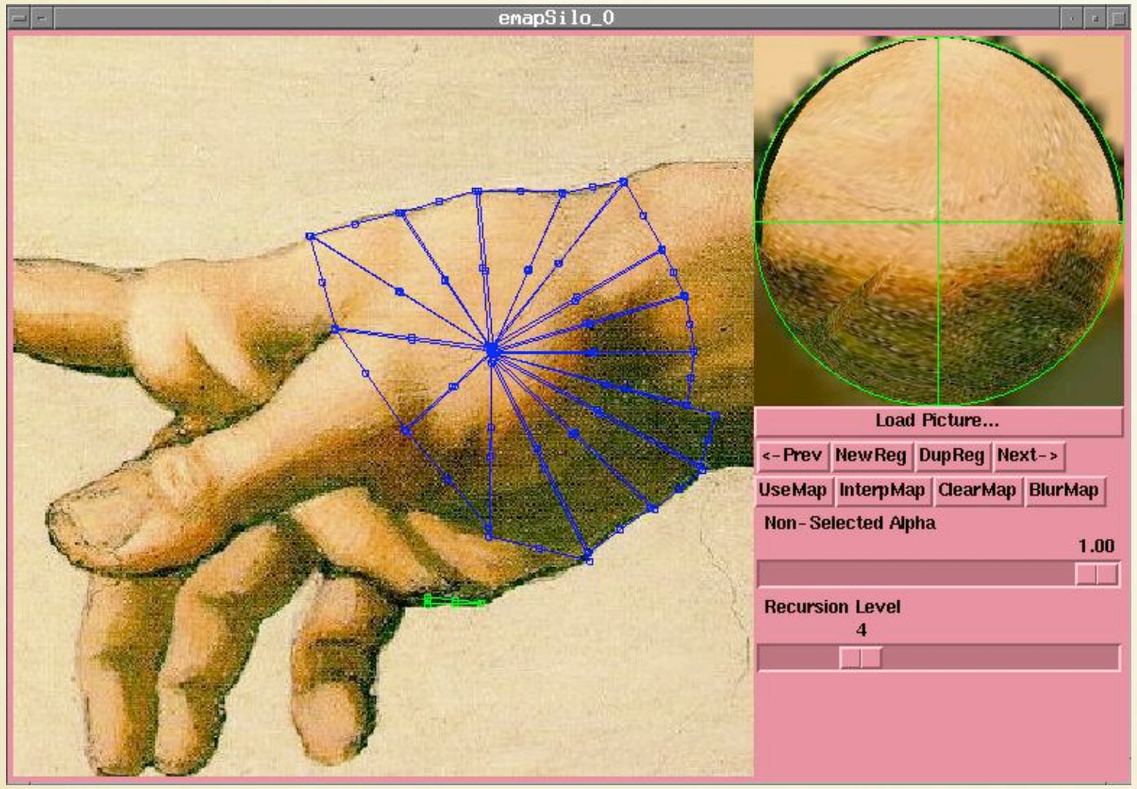
[LS]

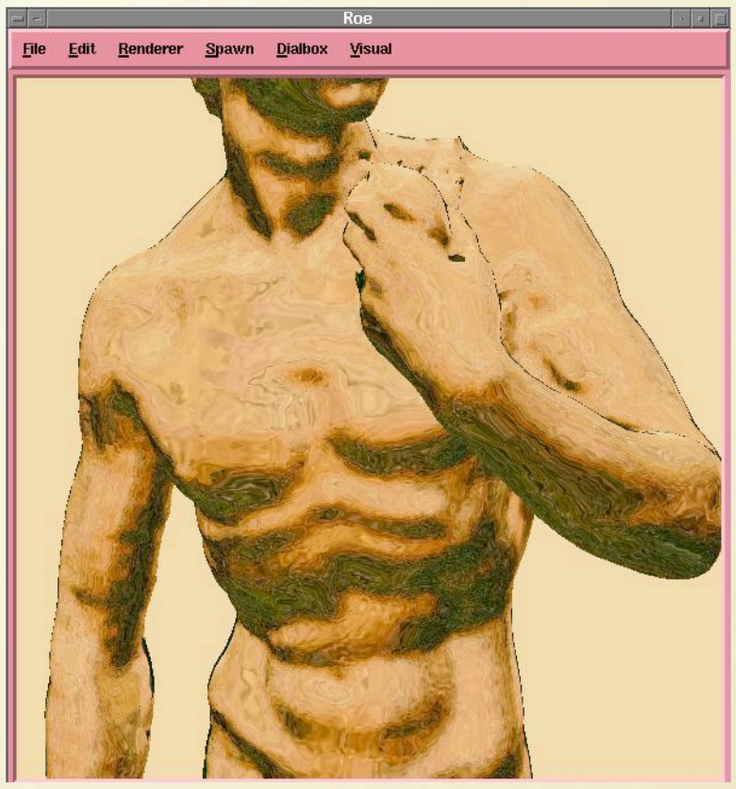
Lit sphere

- These spheres were hand-drawn and scanned, and the David model is rendered with paint-by-normal

[LS]

Lit sphere





- Like environment mapping without reflection
- Can only handle one view, light is fixed with viewer
- Can piece together a lit sphere from an image to "extract" the lighting from a painting [LS]

Painterly rendering

- Really has nothing to do with rendering...
- Idea: take a regular paint program, and make it automatically set brush color by sampling an image
- User "paints in" the image, using existing colors but new brushes
- Super simple to implement, and generates interesting results

Painterly rendering

FIGURES COURTESY...

- Real-Time Rendering, 3rd ed. [RTR]
    - Tomas Akenine-Moller, Eric Haines, Naty Hoffman
- Interactive Computer Graphics: A Top-Down Approach Featuring Shader-Based OpenGL, 6th ed. [ICG]
    - Edward Angel, Dave Shreiner
- Wikipedia [WP]
- Nothrup & Markosian, “Artistic Silhouettes: A Hybrid Approach”, NPAR 2000 [NM]
- Engel, Wolfgang (ed.), ShaderX [SX]
- Gooch et al., “A Non-Photorealistic Lighting Model for Automatic Technical Illustration.”, SIGGRAPH ’98 [BG]
- Sloan et al., “The Lit Sphere: A Model for Capturing NPR Shading from Art”, Graphics Interface 2001 [LS]
- Praun et al., “Real-Time Hatching”, SIGGRAPH 2001 [RH]