Non-Potorealistic Rendering

CS7GV3 – Real-time Rendering

Non-Photorealistic Rendering (NPR)

- Techniques for rendering that don’t strive for realism, but:
  - Style
  - Expressiveness
  - Abstraction
- Related terms:
  - Stylized rendering
  - Artistic rendering
  - Abstract rendering
  - Illustrative rendering (a subset)
  - Visualisation (sometimes)
  - Interpretive rendering

Applications: Stylization

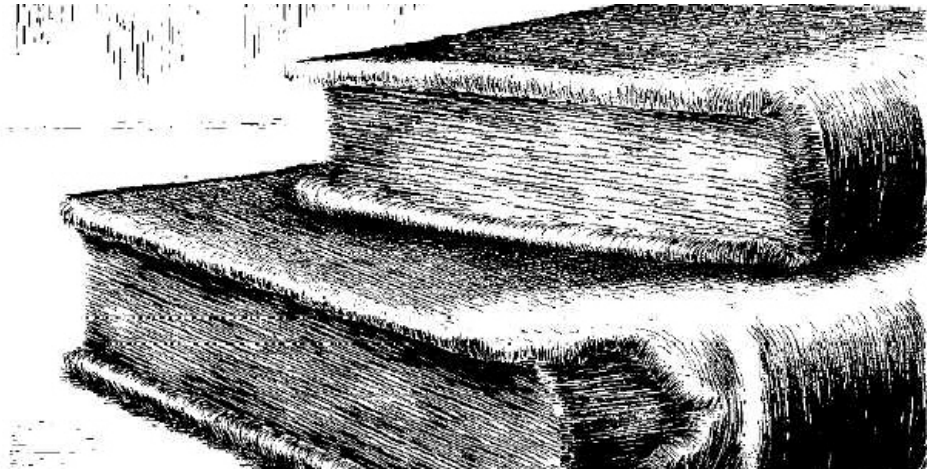
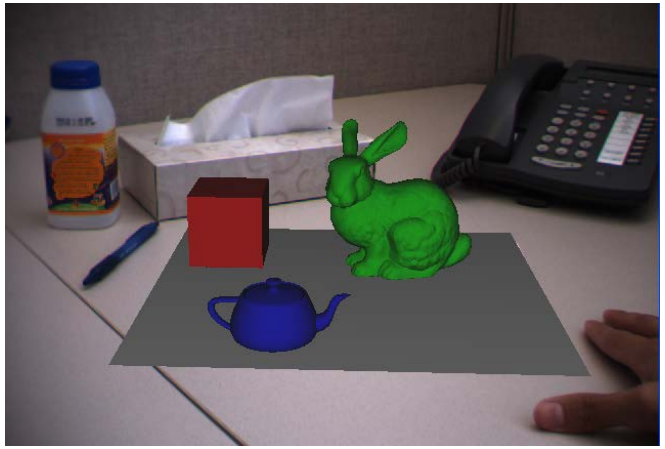
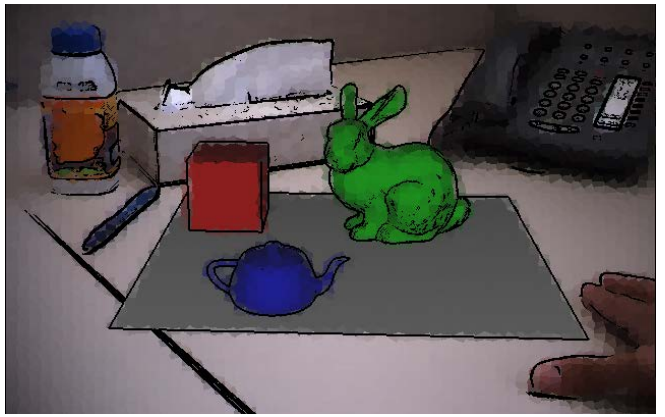


Image: [Salisbury et al 1997]

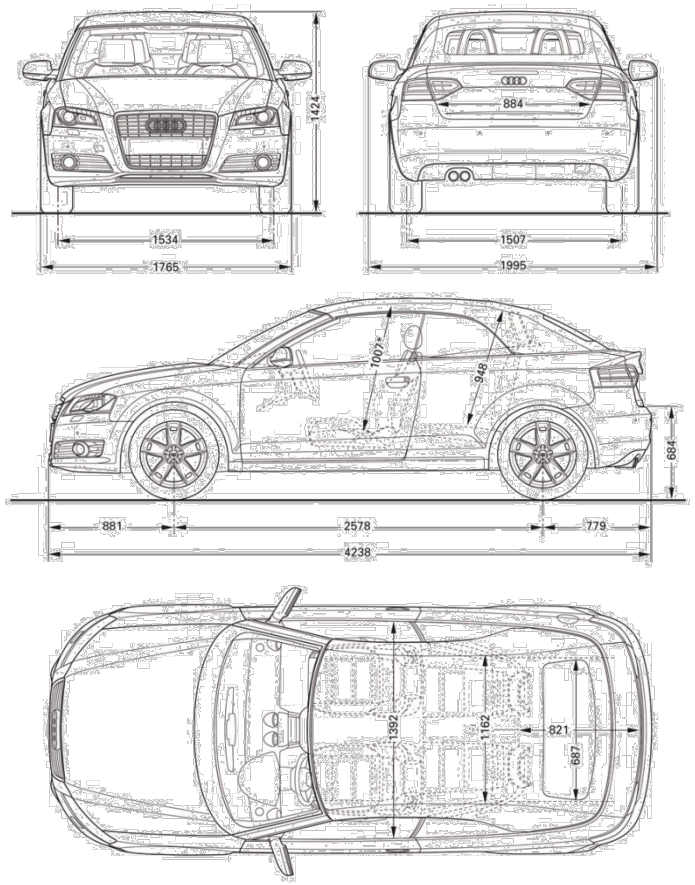
A Scanner Darkly © Warner Independent Pictures, 2006





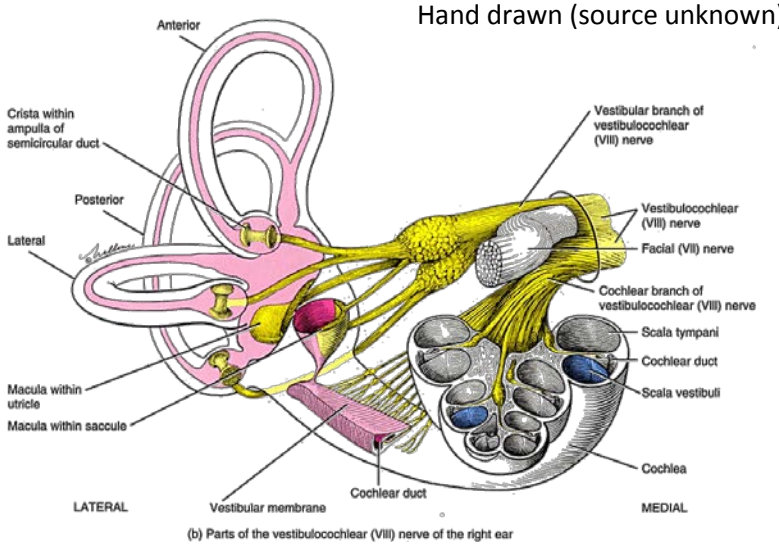
Merging of AR and VR [Chen et al 2008]

Applications: Function

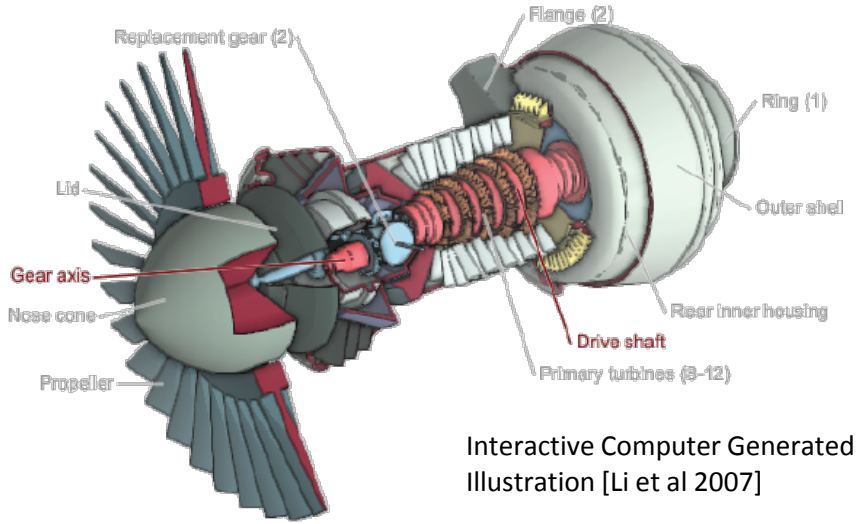


Artist Rendered (source unknown)

Hand drawn (source unknown)

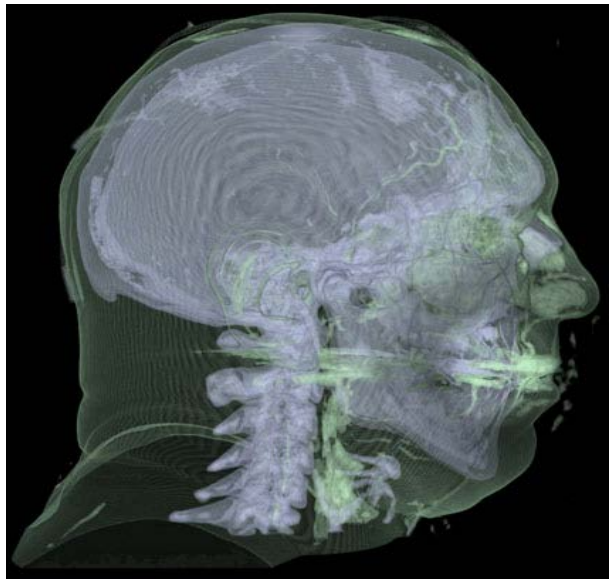


(b) Parts of the vestibulocochlear (VIII) nerve of the right ear



Interactive Computer Generated Illustration [Li et al 2007]

Applications: Analysis



Visualisation Image: [Kim 2006]

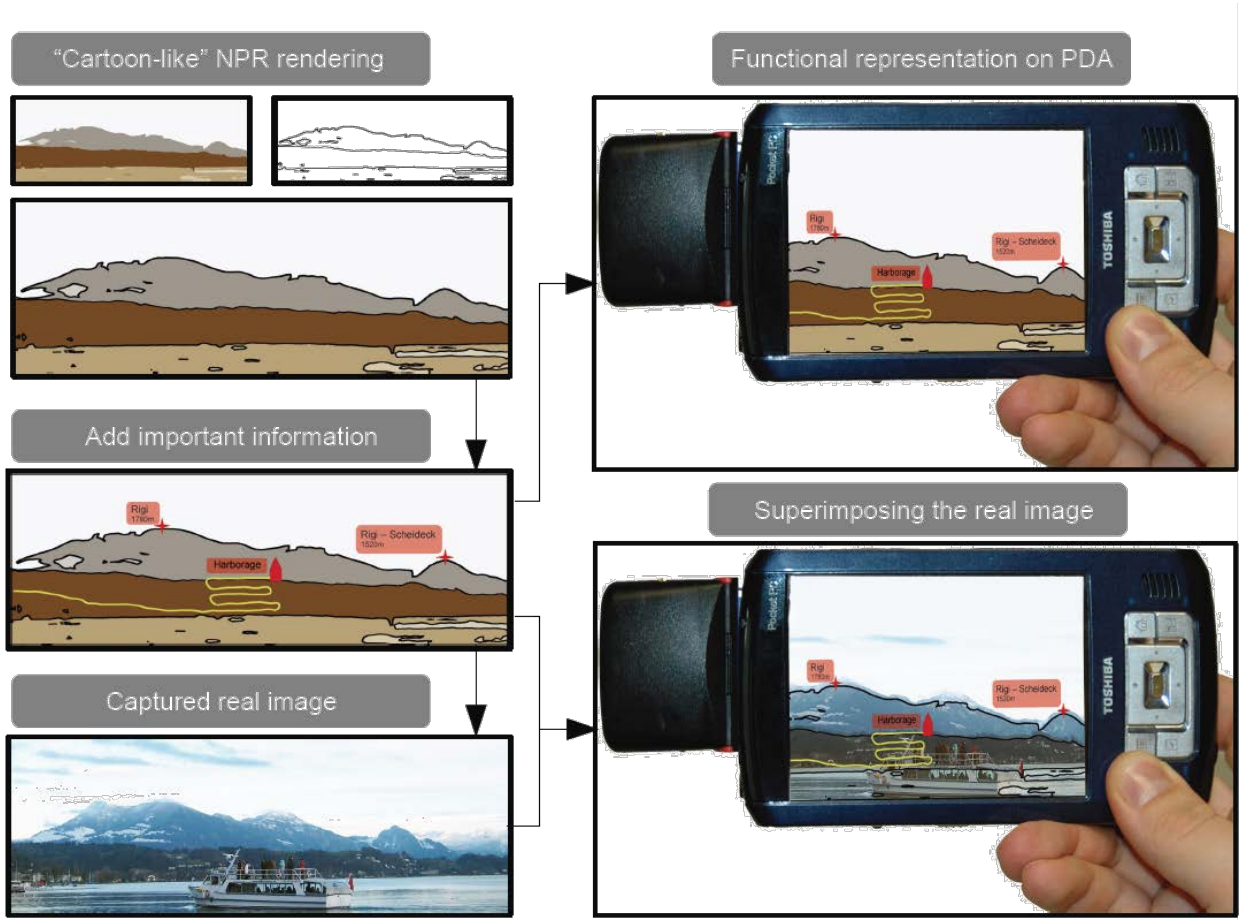
Direct Render

Stylised

Image: [Ebert 2000]

Head Image from Kim and Varshney “Saliency-guided Enhancement for Volume Visualization ” IEEE Visualization Conference 2006.
Body Images from Ebert and Rheingans “Volume Illustration: Non-Photorealistic Rendering of Volume Models”, IEEE Visualization Conference 2000.

Applications: Efficiency



NPR-Quake

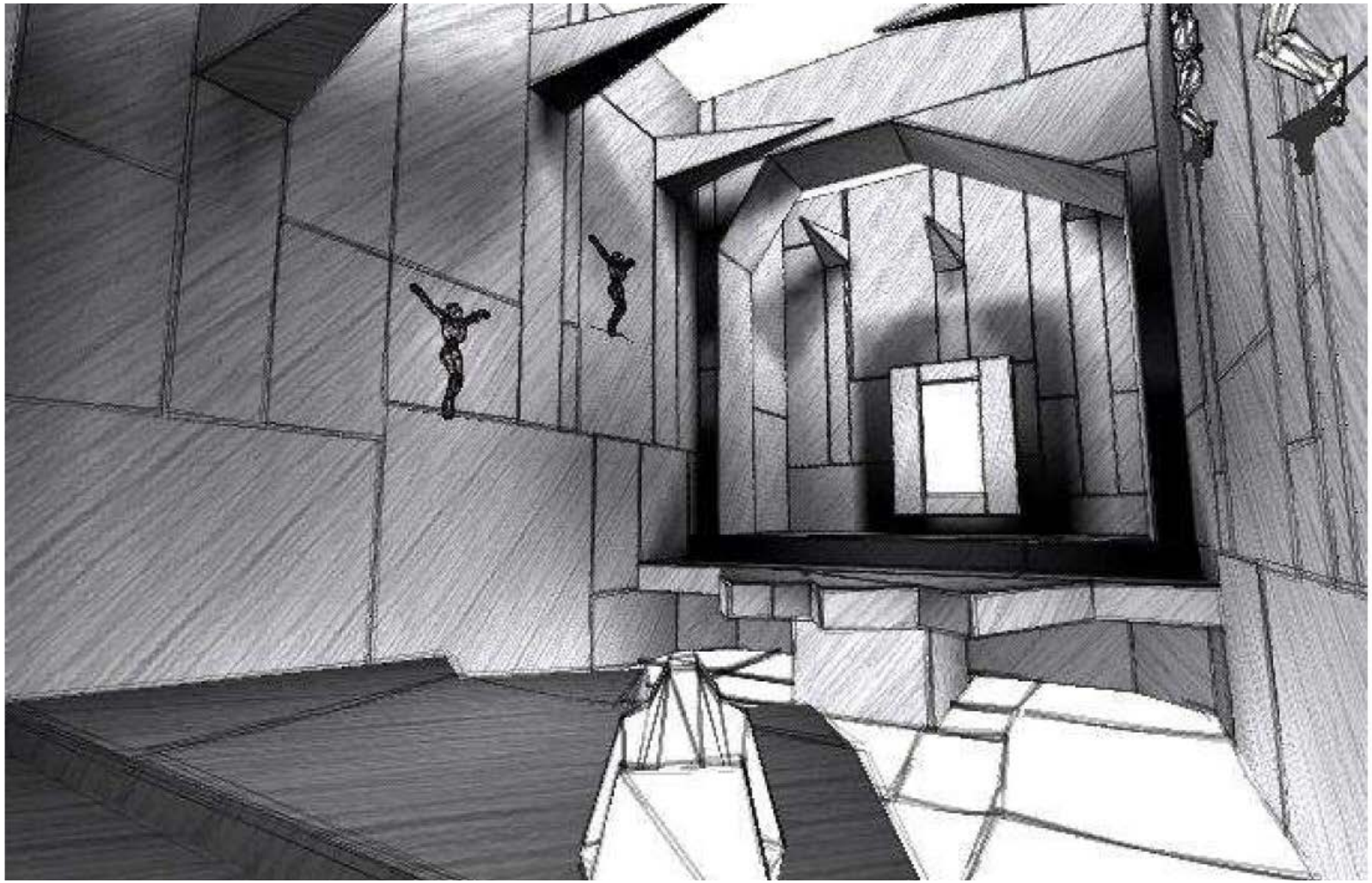


Image of a NPR quake level from Mohr et al 2002 “HijackGL--Non-Invasive Extensibility for Graphics Applications”
Code available at: http://www.cs.wisc.edu/graphics/Gallery/HijackGL/

Legend of Zelda: Windwalker (2002)

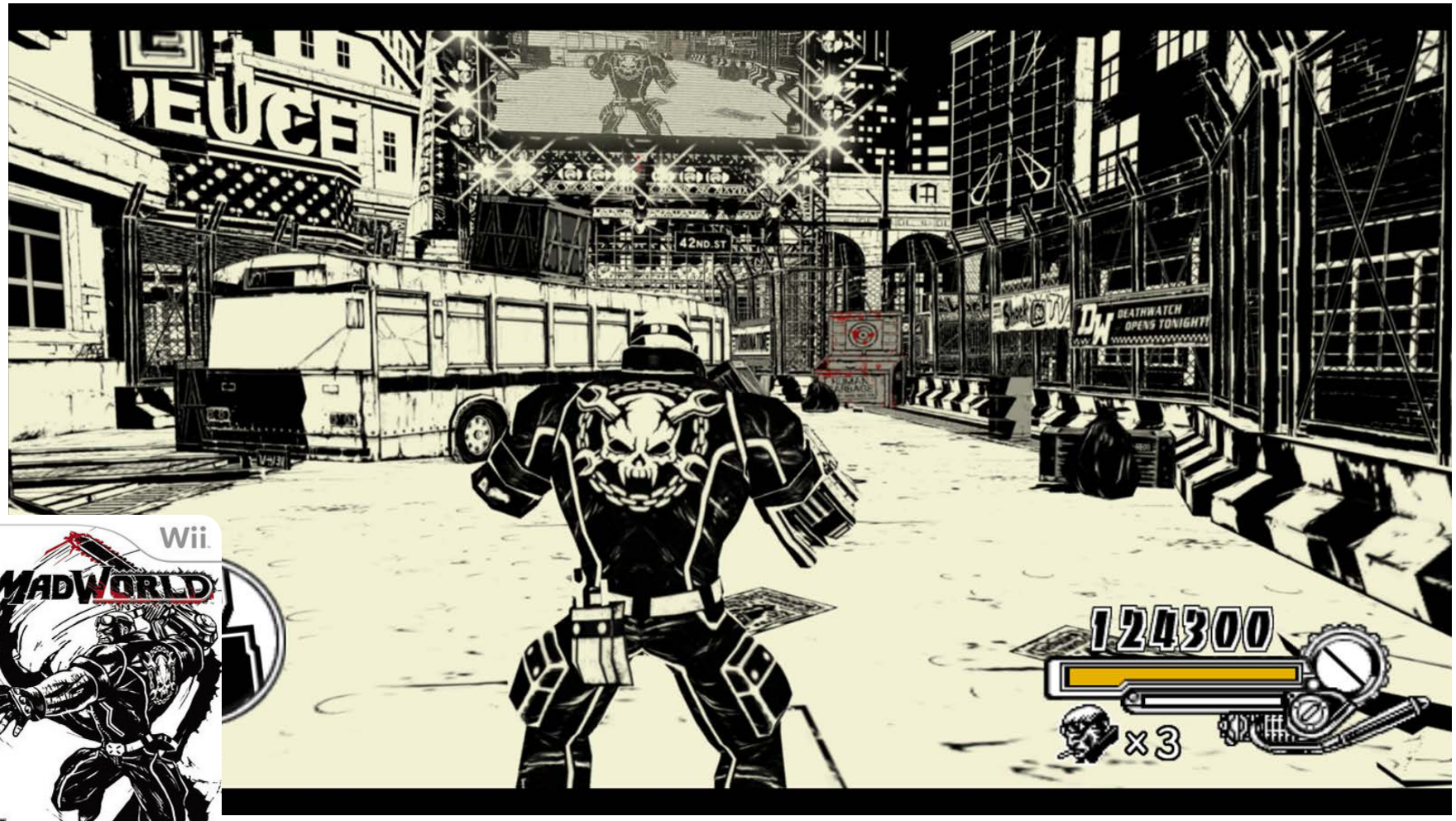
XIII (2003)

Okami (2006)

Eternal Sonata (2007)

Prince of Persia (2008)

MadWorld (2009)

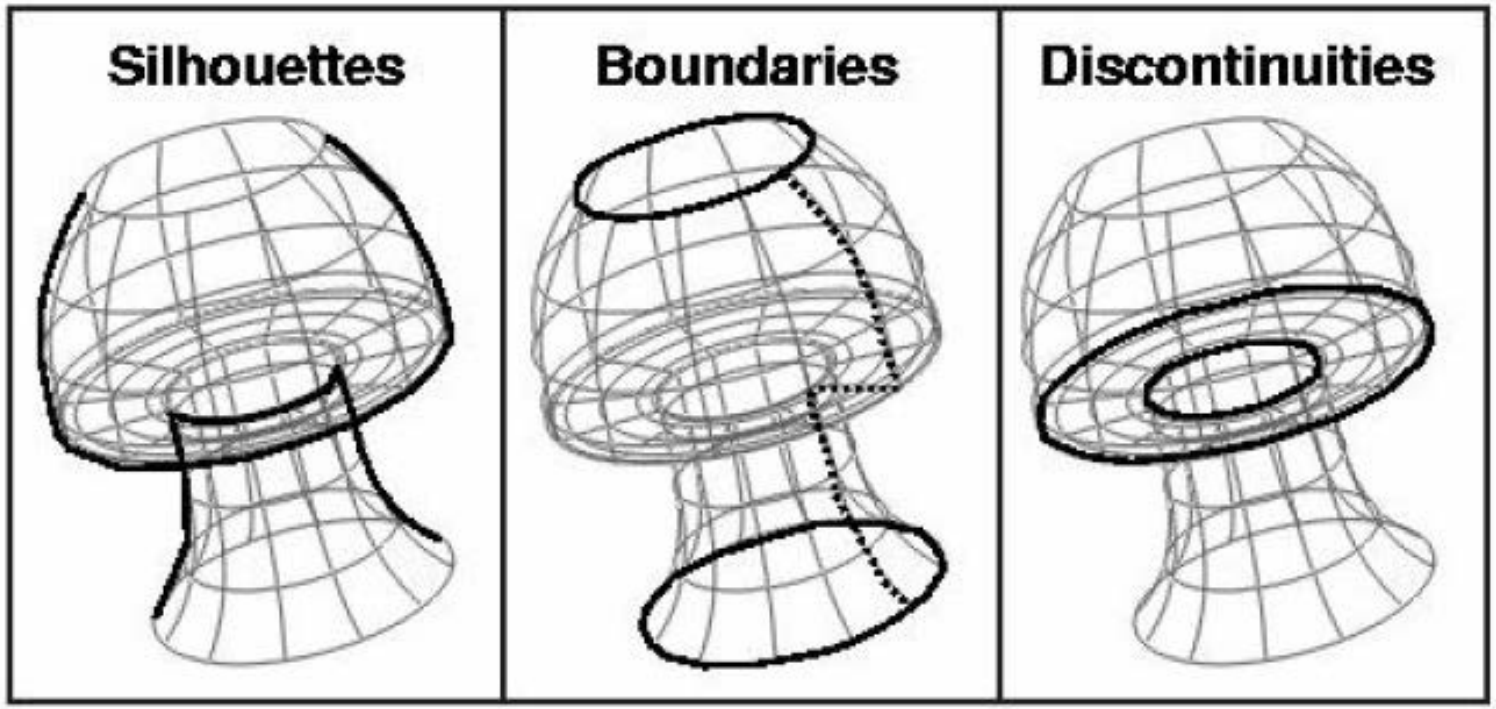


Mirror’s Edge (2010)

Borderlands (2010)

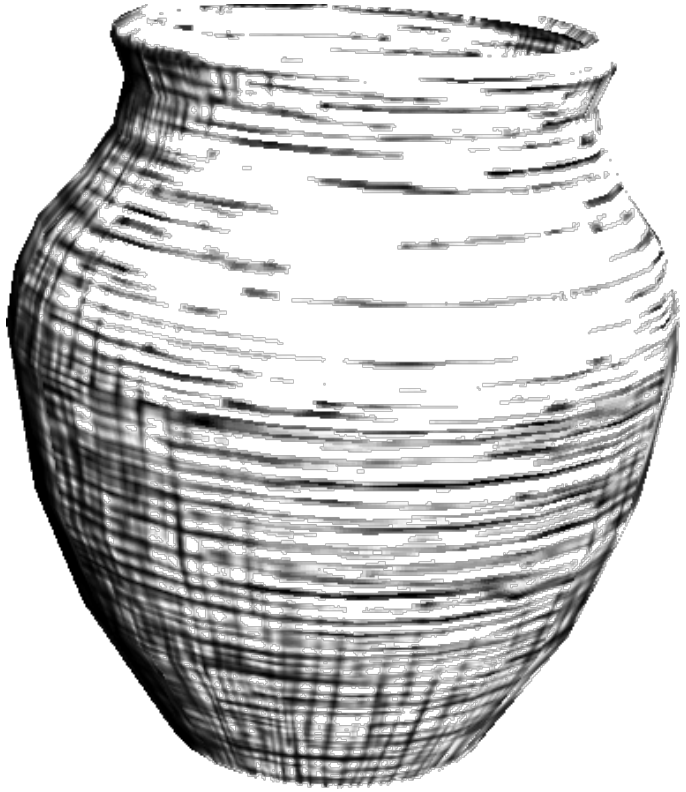
Classes of Abstraction

Lines



Texture

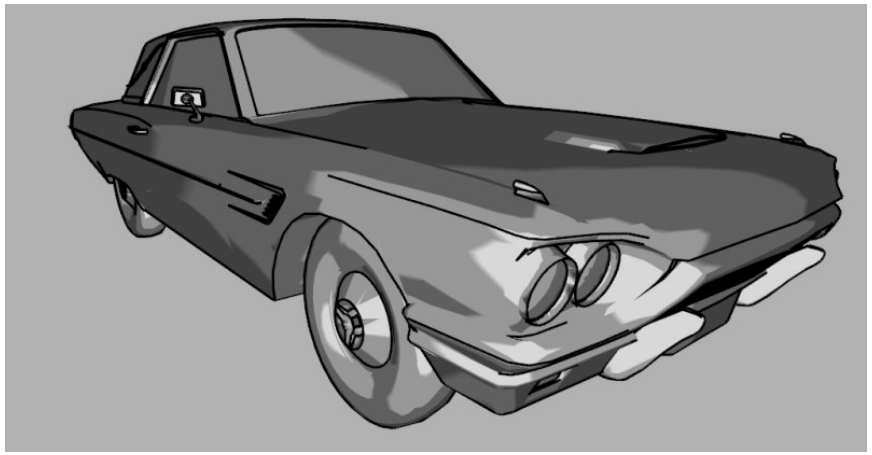
Textural Abstraction: Remove extraneous detail



Texture can also be used to convey information e.g. Shape & curvature using stroke textures.

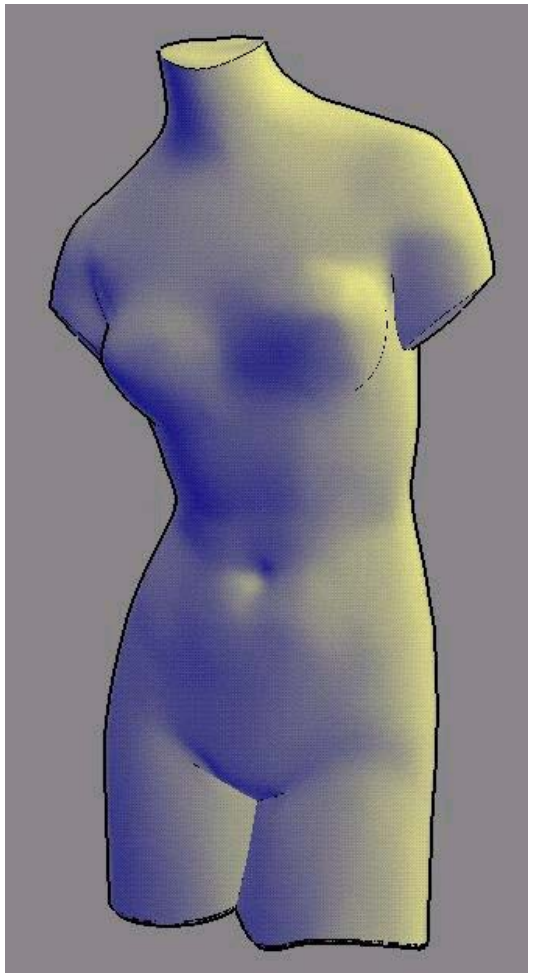
Left image from DeCarlo and Santella “Stylization and Abstraction of Photographs”, SIGGRAPH 2002
Right Image from Praun et al “Real-time Hatching”, Siggraph 2002

Colour and Shading



Discretize colours or levels of illumination:
simulates reduced colour palette used by artists.

Colour Abstraction: quantize colors to remove
unnecessary complexity

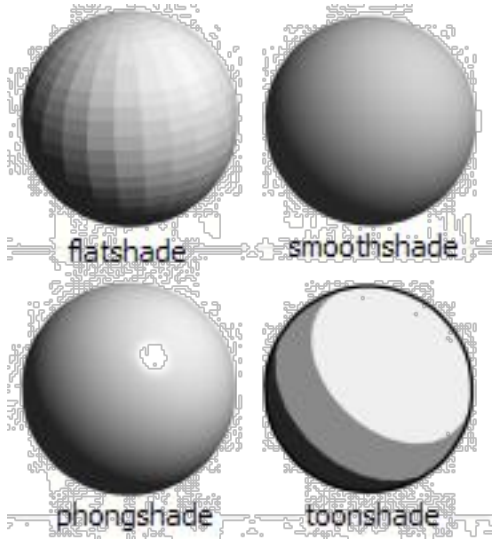


Shading can also be used to convey stronger info.
E.g. Gooch shading for better shape perception.

Simple Examples

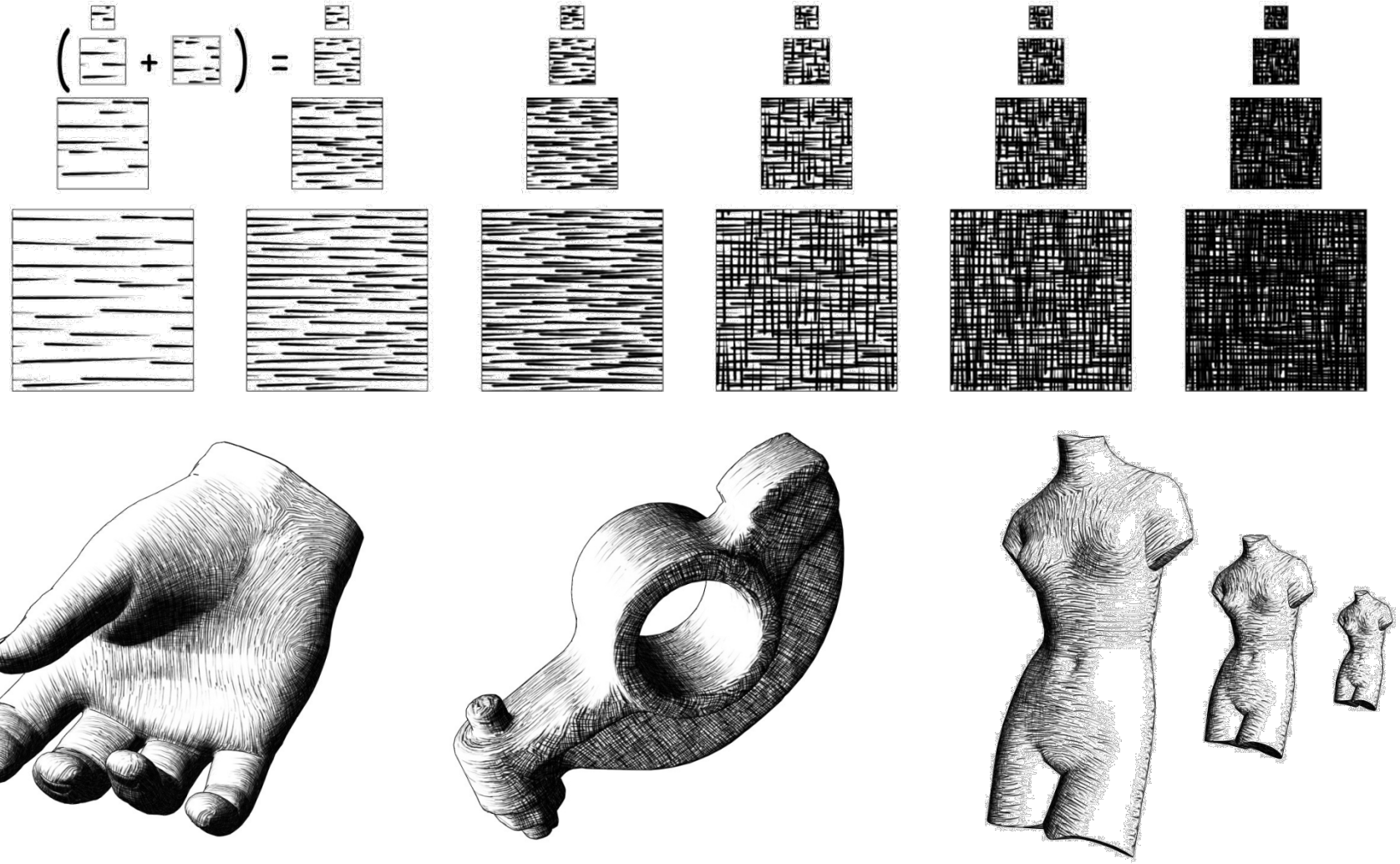
Toon Shading

Quantized ***N.L*** illumination to discrete number of bins.



Many people think this is the extend of NPR ... It isn't!

Tonal Art Maps

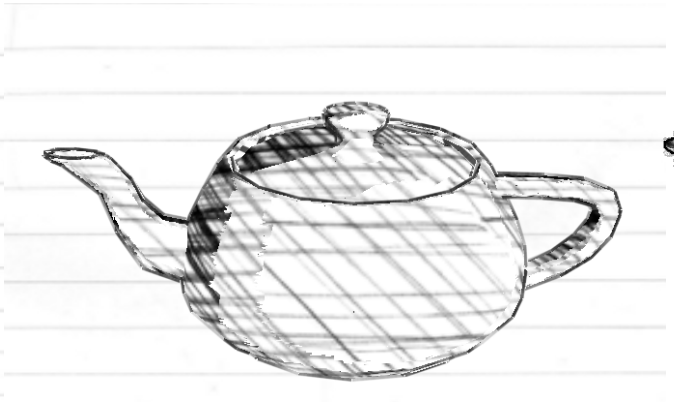


From Praun et al “Real-time Hatching”, Siggraph 2002

Tonal Art Maps

- Use textures that are hand drawn
- Calculate lighting first,
- Threshold illumination and apply a texture with relevant “darkness”

Textures oriented with polygons (arbitrary)



Textures oriented with screen

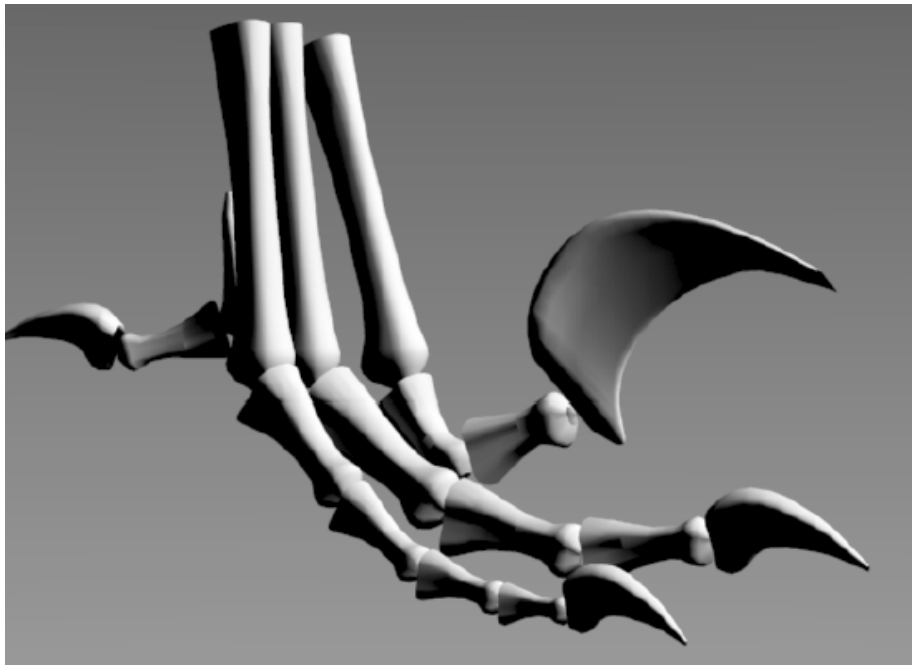
Textures oriented with parameterization

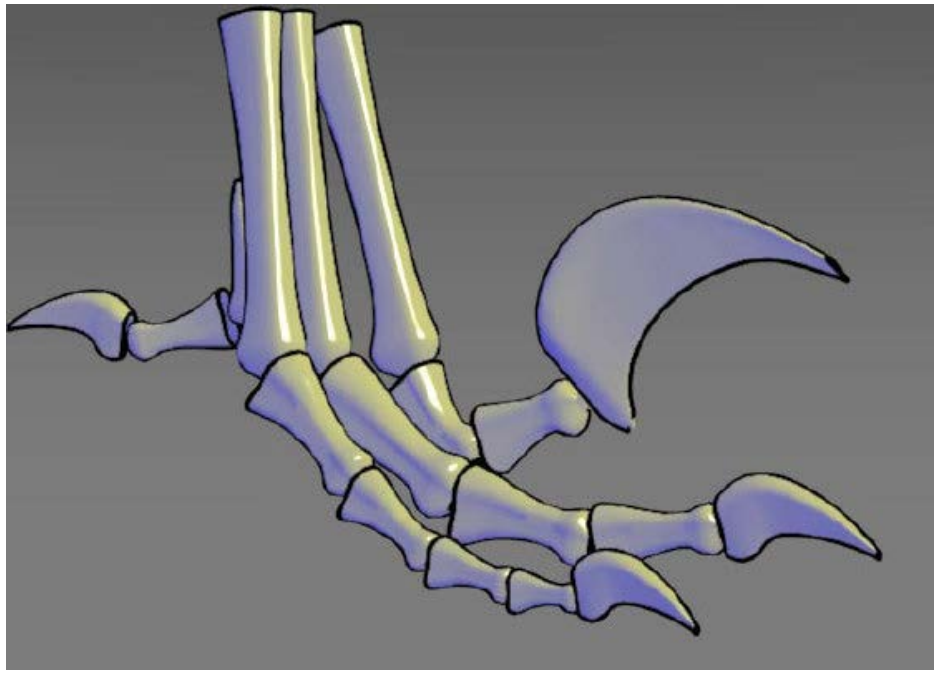
Colour Quantization

Colour/intensity quantization is a common Image Processing operation e.g. In Image Mosaics or Halftoning

Left Image from “Video Mosaics” http://www.cs.princeton.edu/gfx/pubs/Klein_2002_VM/index.php

Tone Shading (Non-Photorealistic Lighting Model)





Images from:
Gooch, Gooch, Shirley, Cohen, "A Non-Photorealistic Lighting Model for Automatic Technical Illustration," SIGGRAPH 98
http://doi.acm.org/10.1145/280814.280950

Tone Shading

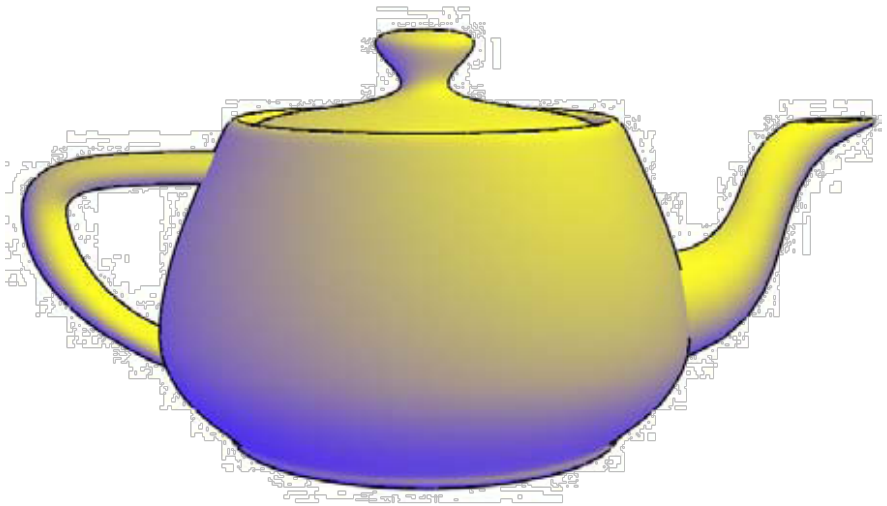
$$\mathbf{I} = \left(\frac{(1+\mathbf{N}\bullet\mathbf{L})}{2}\right)k_{warm} + \left(1-\frac{(1+\mathbf{N}\bullet\mathbf{L})}{2}\right)k_{cool}$$

$$k_{cool} = k_{blue} + \alpha\, k_d$$

$$k_{warm} = k_{yellow} + \beta\, k_d$$

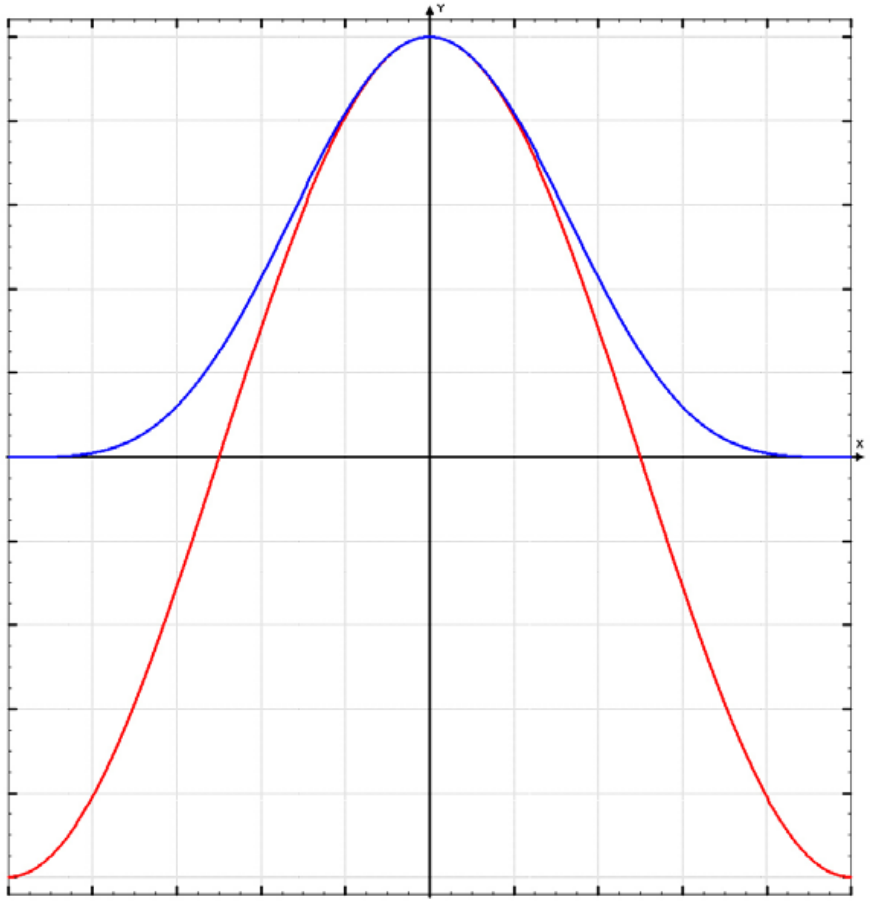
$$k_{blue} = (0,0,b) \qquad b \in [0,1]$$

$$k_{yellow} = (y,y,0) \qquad y \in [0,1]$$



α an β are user/artist specified variables that control the strength of the temperature shift

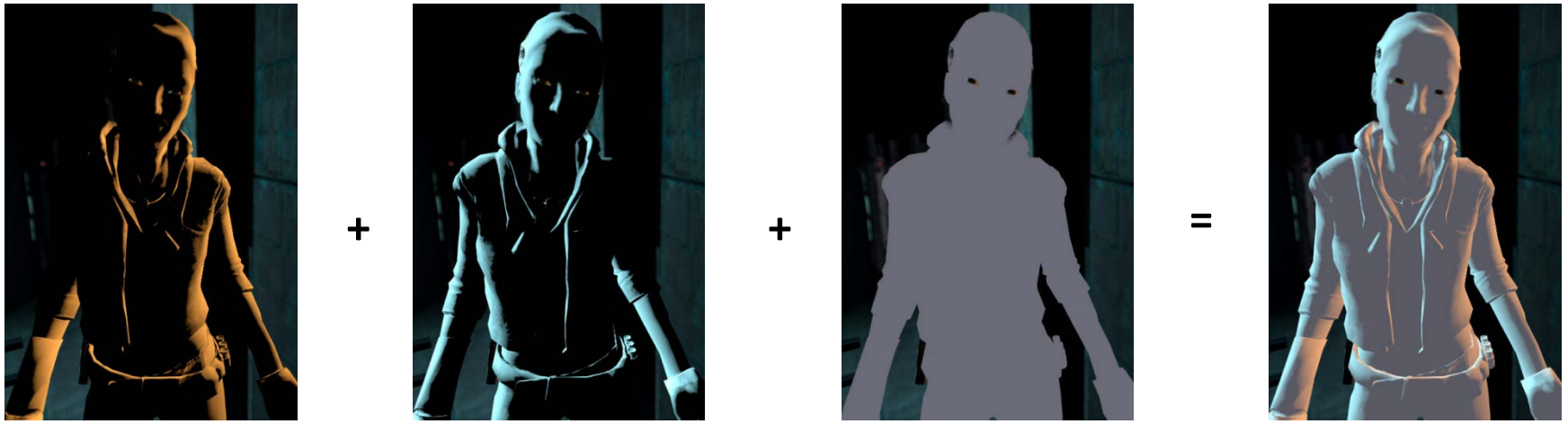
Half-Lambert



- Tone shading effectively results in compression of the *dynamic range*
- Similar technique is used in valve's half-life engine
- Instead of clamping the -1 to 1 range of N.L, scale and bias to 0 to 1 range
- Eliminates the flat look

Images from: http://www.valvesoftware.com/publications/2006/SIGGRAPH06_Course_ShadingInValvesSourceEngine.pdf

Half-Lambert



Lambertian 1 | Lambertian 2 | Ambient | Final Image

½ Lambertian 1 | ½ Lambertian 2 | Ambient Cube | Final Image

Images from: http://www.valvesoftware.com/publications/2006/SIGGRAPH06_Course_ShadingInValvesSourceEngine.pdf

Outlines

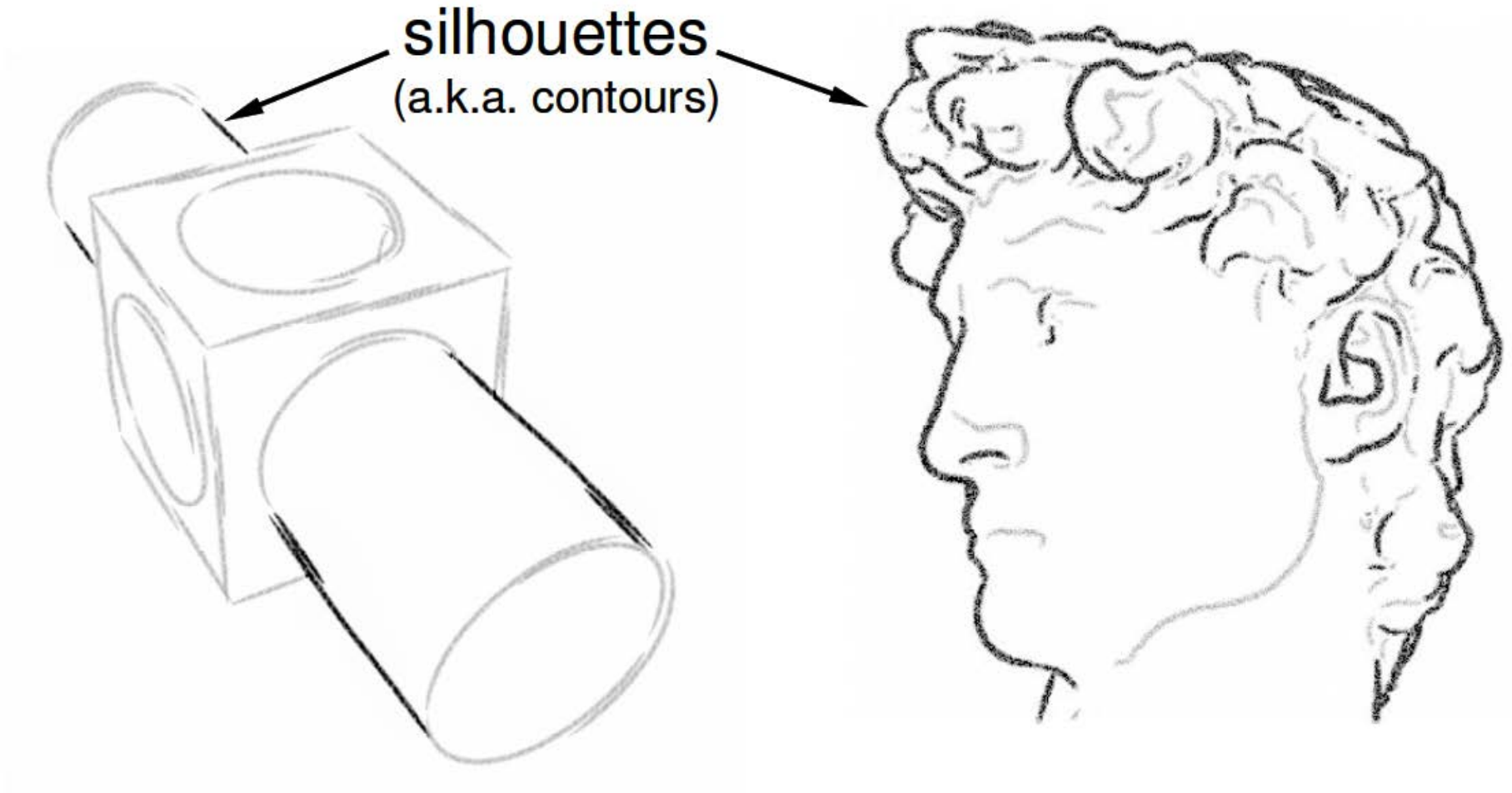


Image from Rusinkiewicz “Line Drawings from 3D Models, Part III Mathematical Description of Lines”, SIGGRAPH 2008 Course Notes.

Outlines

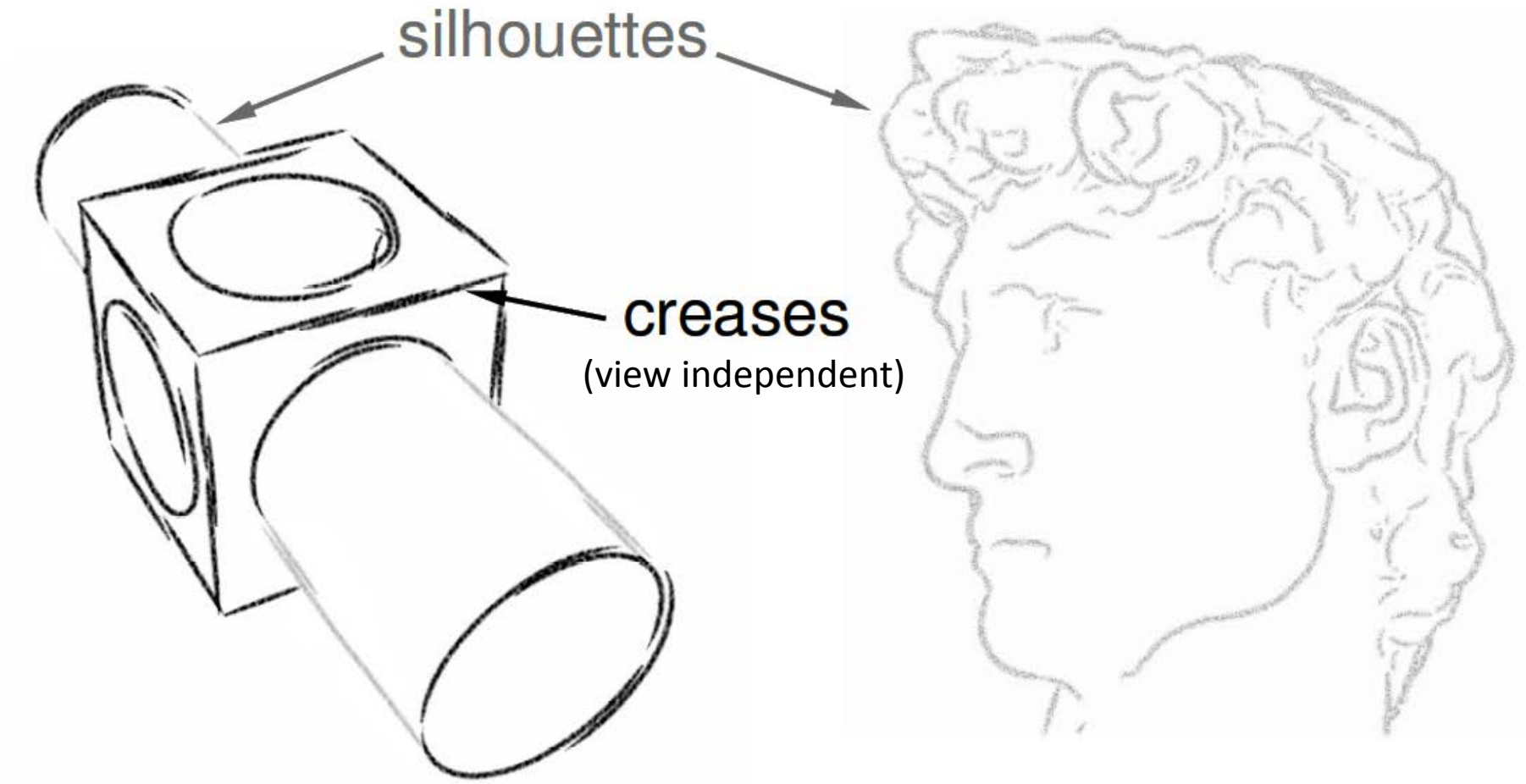


Image from Rusinkiewicz “Line Drawings from 3D Models, Part III Mathematical Description of Lines”, SIGGRAPH 2008 Course Notes.

Outlines

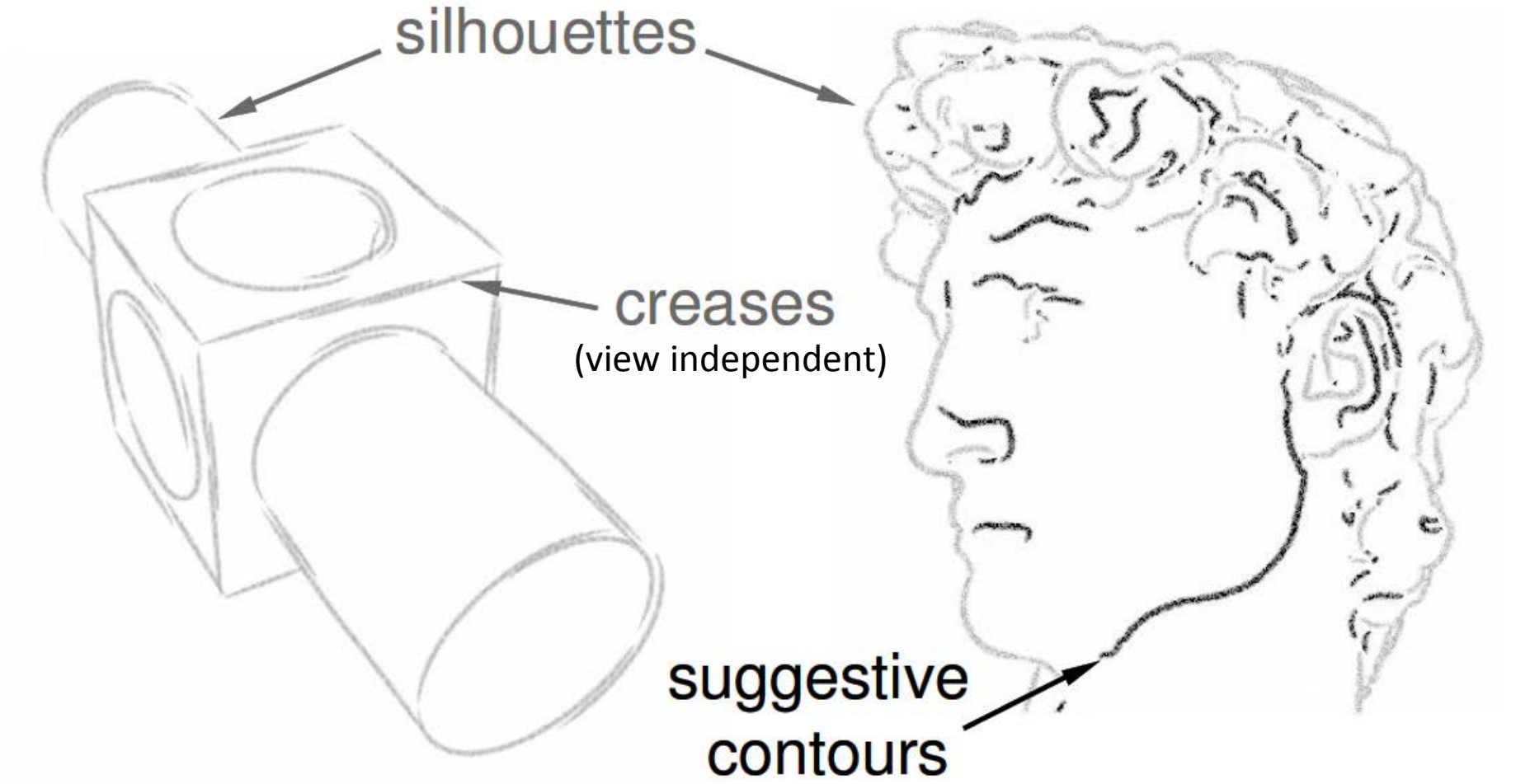


Image from Rusinkiewicz “Line Drawings from 3D Models, Part III Mathematical Description of Lines”, SIGGRAPH 2008 Course Notes.

Line Drawings from 3D Data

Many More classes if lines (some too expensive for real-time apps)

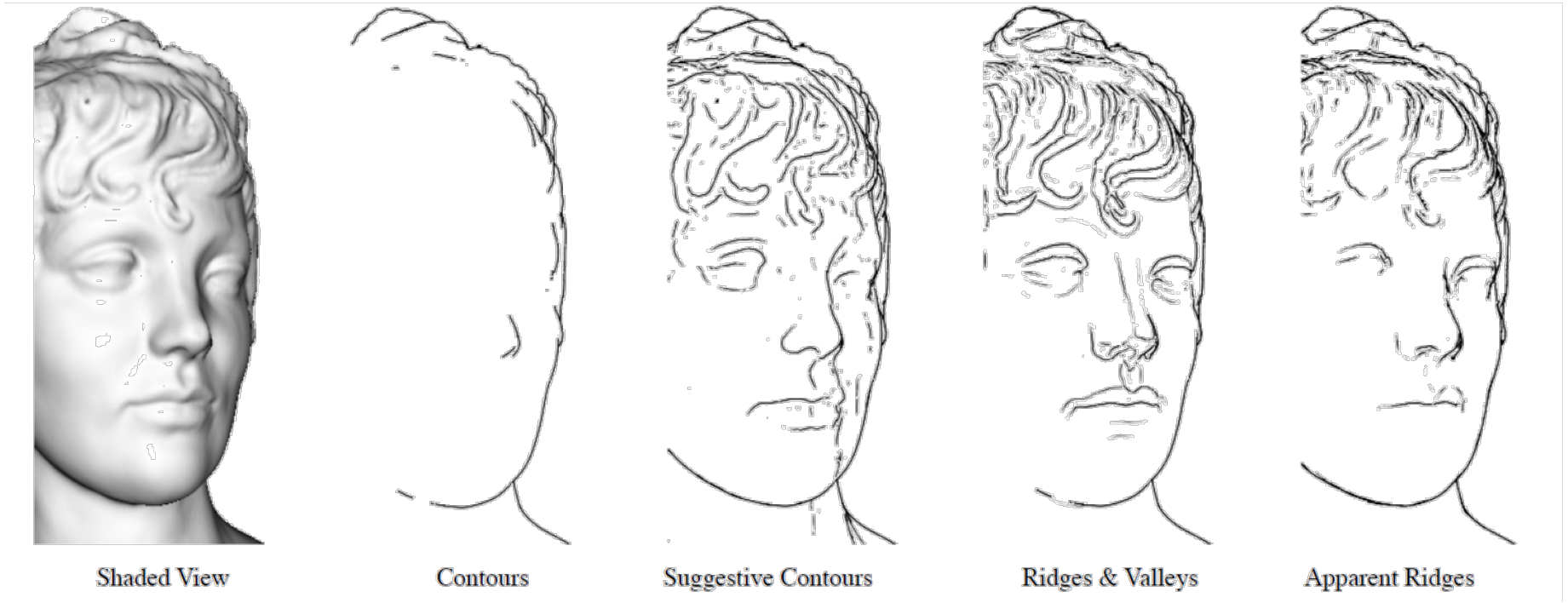
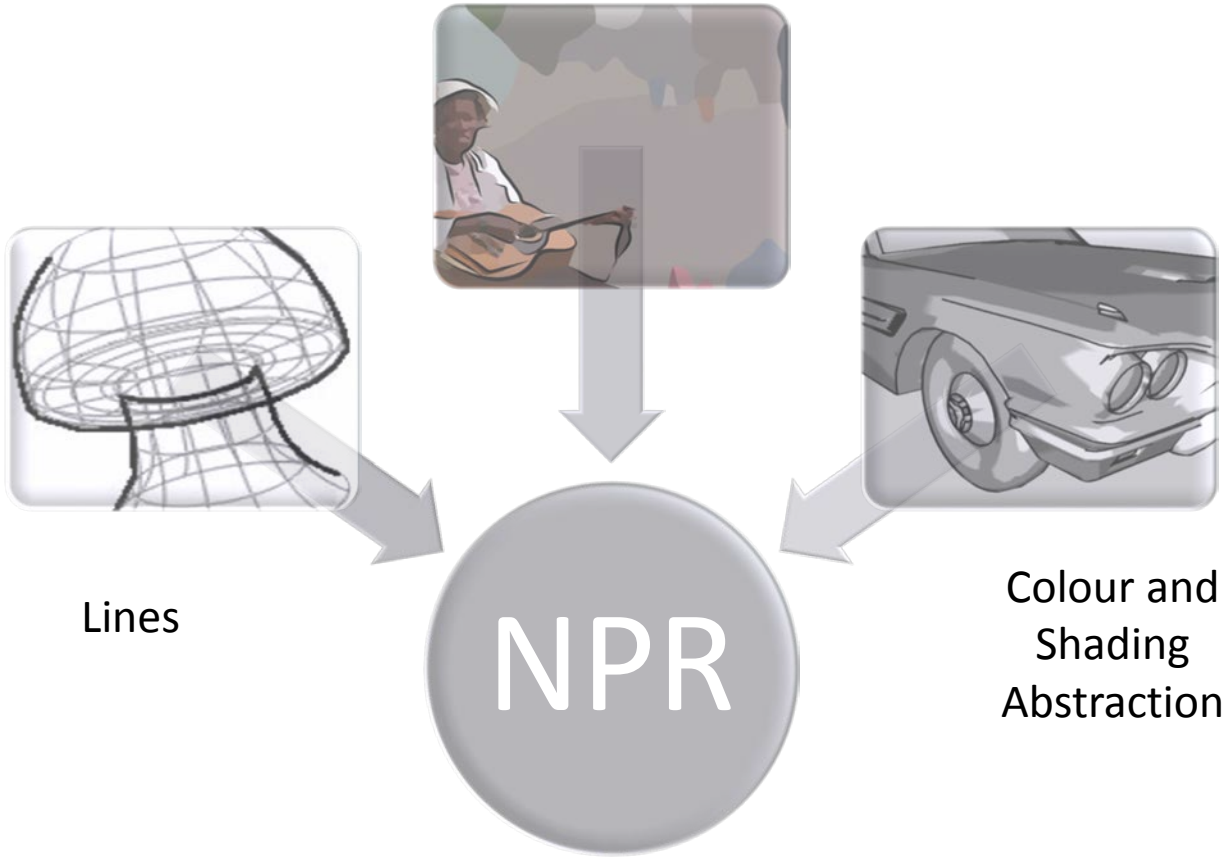


Image from: Tilke Judd, Fredo Durand, and Edward Adelson. “Apparent ridges for line drawing”. In *SIGGRAPH 2007.*

Real-time NPR Techniques

Abstraction and Enhancement



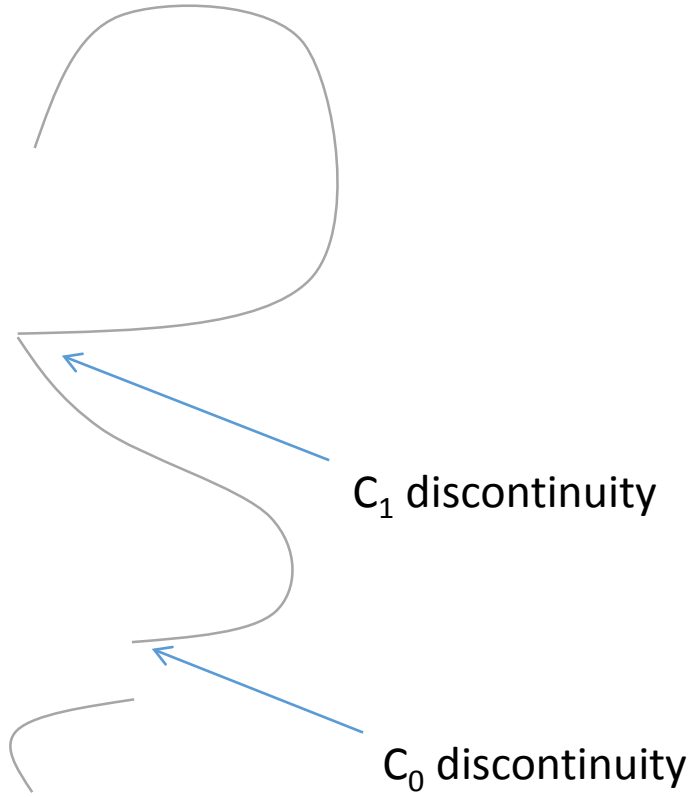
All NPR techniques basically work by removing detail in some places and enhancing others.
NPR is also highly amenable to multi-pass rendering techniques: each pass has some relevance to style.

Edges

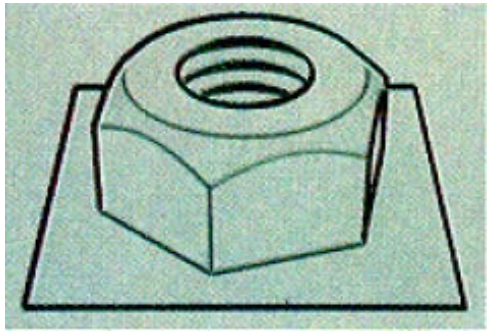
- Edges are an important cue in image perception
  - Shape perception
  - Boundary perception

- Most obvious edges are Silhouettes
  - But there are a number of other places where edges are useful

Discontinuities

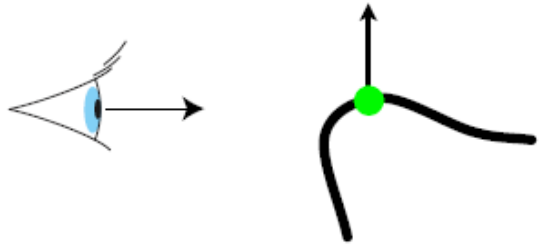


Creases



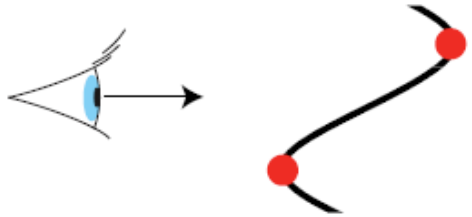
- Angle between two adjacent triangles is smaller than a certain threshold.
  - Simply inspect all normals
  - Can be pre-calculated
- Significant in “mechanical” models, not so much in “organic” models.

Contours



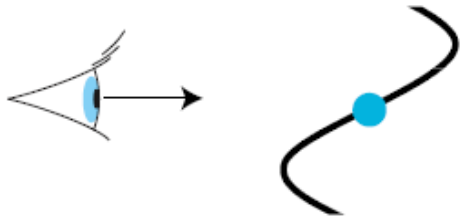
- Where object normal is perpendicular to view direction

Ridges and Valleys



- Maxima & minima of the curvature.

Suggestive Contours



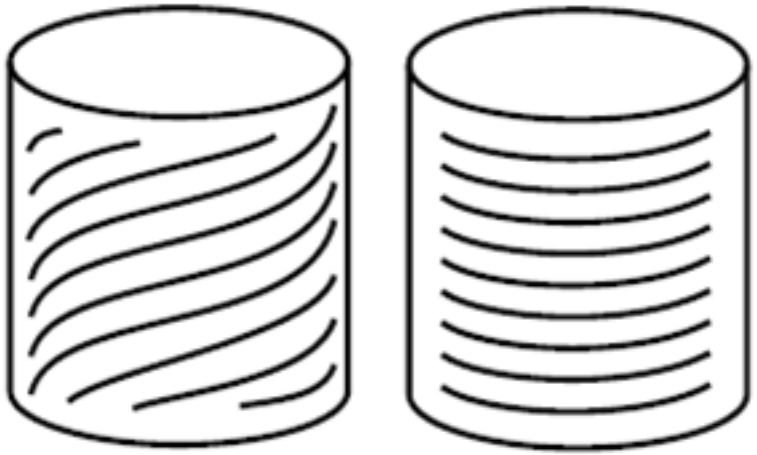
- Inflection points (were curve starts to go from decreasing to increasing curvature and vice versa)

Apparent Ridges

- Essentially a View Dependent form of Ridges and Valleys
- Extrema of max view-dependent curvature

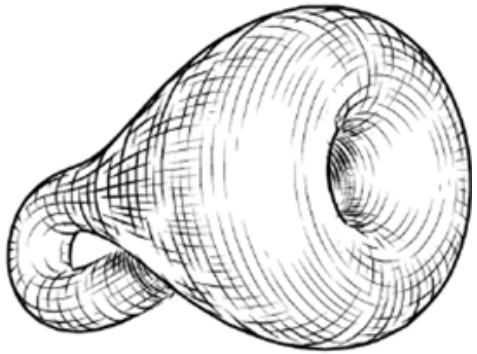
Curvature Lines

- Additionally various classes of contour lines on the surface of an object can communicate shape and curvature
- However choosing correct contour lines can make the shape even more apparent. Lines should align along the direction where curvature changes most.



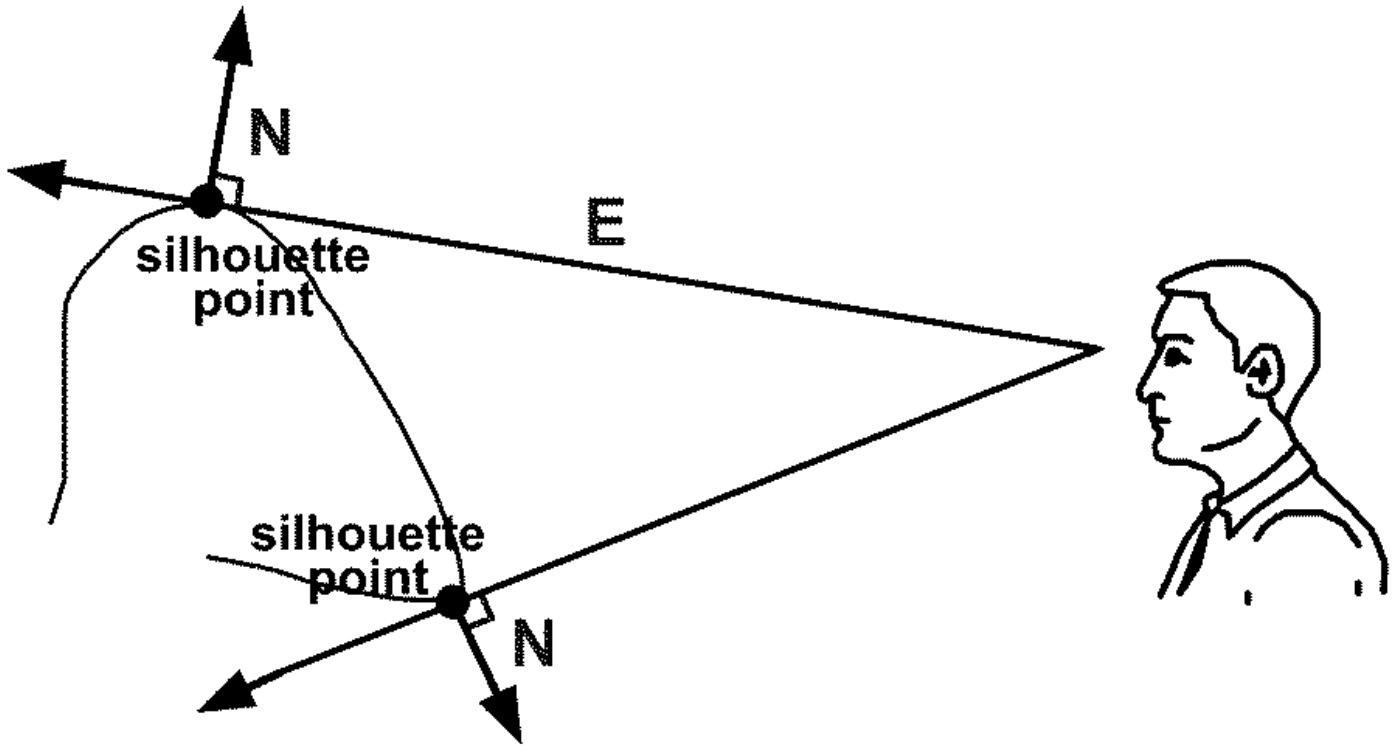
Geodesic Lines

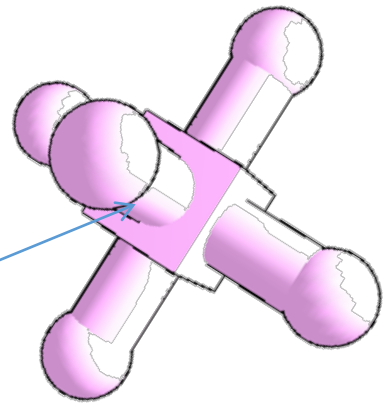
Lines along curvature



- Slightly difficult (and expensive) to calculate these on the fly
- But could be determined at the modelling stage – or using objects parameterization

Silhouettes

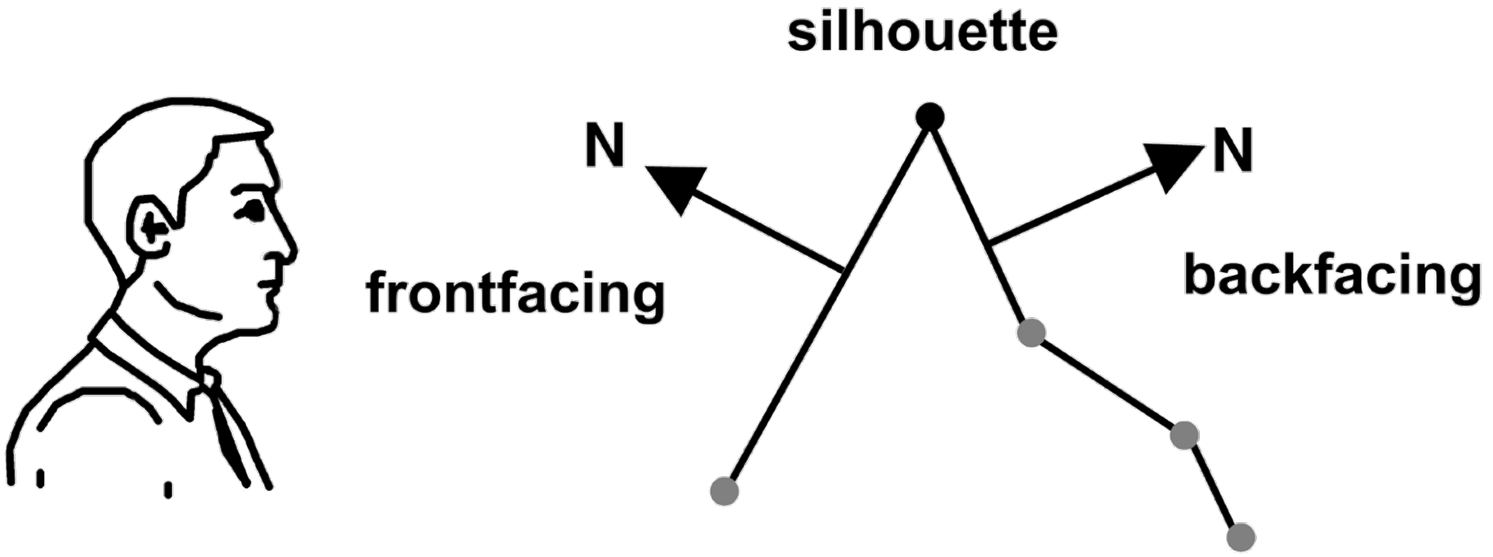




Important lines in 3D are where eye and normal are perpendicular.

This also applies to “interior silhouettes”

Silhouettes



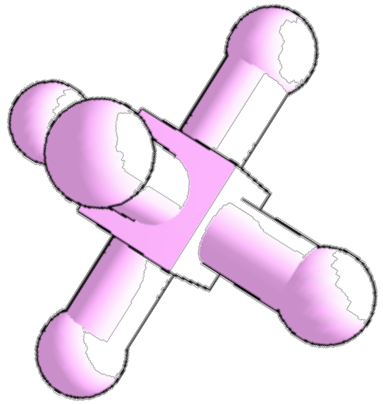
For Polyhedral meshes this can be simplified considerably:
Simply draw edges where a back-facing and front-facing polygon meet.

Explicitly calculate this on the fly. Basic idea: store a mesh data structure that specifically keeps track of neighbouring edges.
Many CPU techniques do this ... a little bit messy on GPU.

Silhouette Hack

Essentially: draw object in black slighlty scaled up and then draw object over this shaded. OpenGL example (Fixed Function):

```
void drawSilhouette( void (*drawObject) (void), int line_width = 2, bool p_smooth = true )
{
        static const double depth_cutoff = 0.001;
        glPushAttrib( GL_ENABLE_BIT );
        glDepthRange(depth_cutoff, 1);
        glEnable(GL_CULL_FACE);
        glCullFace (GL_FRONT); //assumes correct winding
        glPushAttrib( GL_POLYGON_BIT | GL_CURRENT_BIT );
            glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
            glLineWidth(line_width);
            glColor3f(0, 0, 0); //assume all silhouettes black
            drawObject();
        glPopAttrib();
        glDisable(GL_CULL_FACE);
        glColor3f(1, .71, 1);
        drawObject();
        glPopAttrib();
}
```



For a better alternative in OpenGL see:
http://www.opengl.org/resources/code/samples/sig99/advanced99/notes/node108.html

Image Lines

Standard edge detectors from image processing can be applied to an intermediate rendering of the 3D scene – accelerated in GPU

Difference of Gaussian (DoG)

Quick 2-pass method:
- apply Gaussian Blur
- subtract blur image from original to get edges

Sobel

Quick technique:
Apply kernels as follows:

Horizontal

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

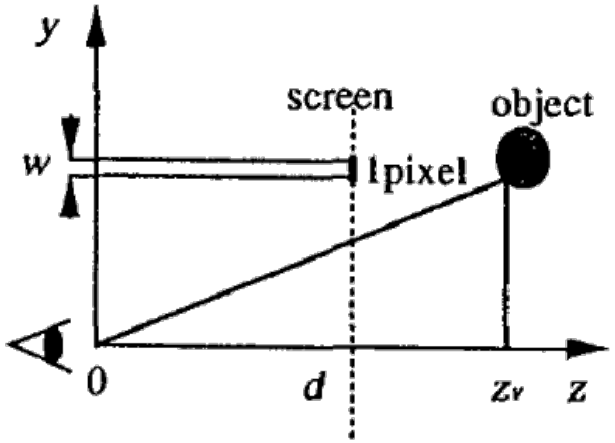
Vertical

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

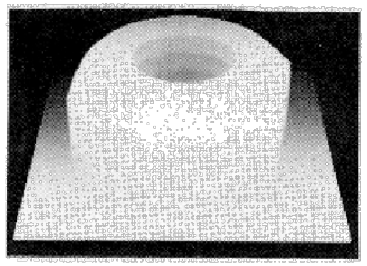
Canny

Very accurate but expensive to run full Canny in real-time

Image Space 3D Edges



$$z_s = \frac{d^2}{w z_v}$$



depth image

For a pixel X and its neighbours (A-H)

$$\begin{matrix} A & B & C \\ D & X & E \\ F & G & H \end{matrix}$$

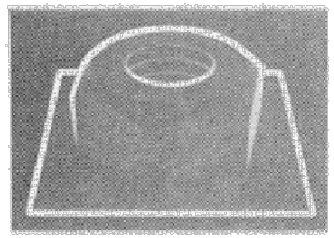
C_0 edges given by

$$C_0(X) = (|A + 2B + C - F - 2G - H| + |C + 2E + H - A - 2D - F|)/8$$

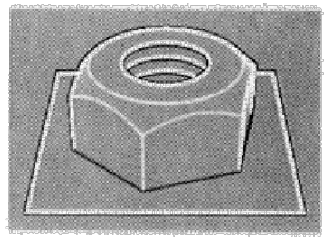
C_1 edges given by

$$C_1(X) = (8X - A - B - C - D - E - F - G - H)/3$$

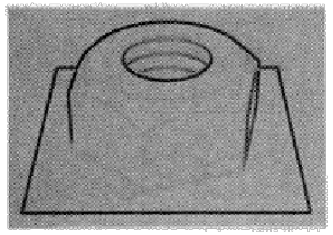
Above values need to be thresholded to only capture strong edges



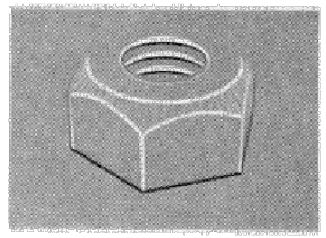
1st order differential



2st order differential



profile image



internal edge image

For some additional details: “Comprehensible Rendering of 3-D Shapes” – Saito Takahashi, SIGGRAPH 1990

Normal Map Edges

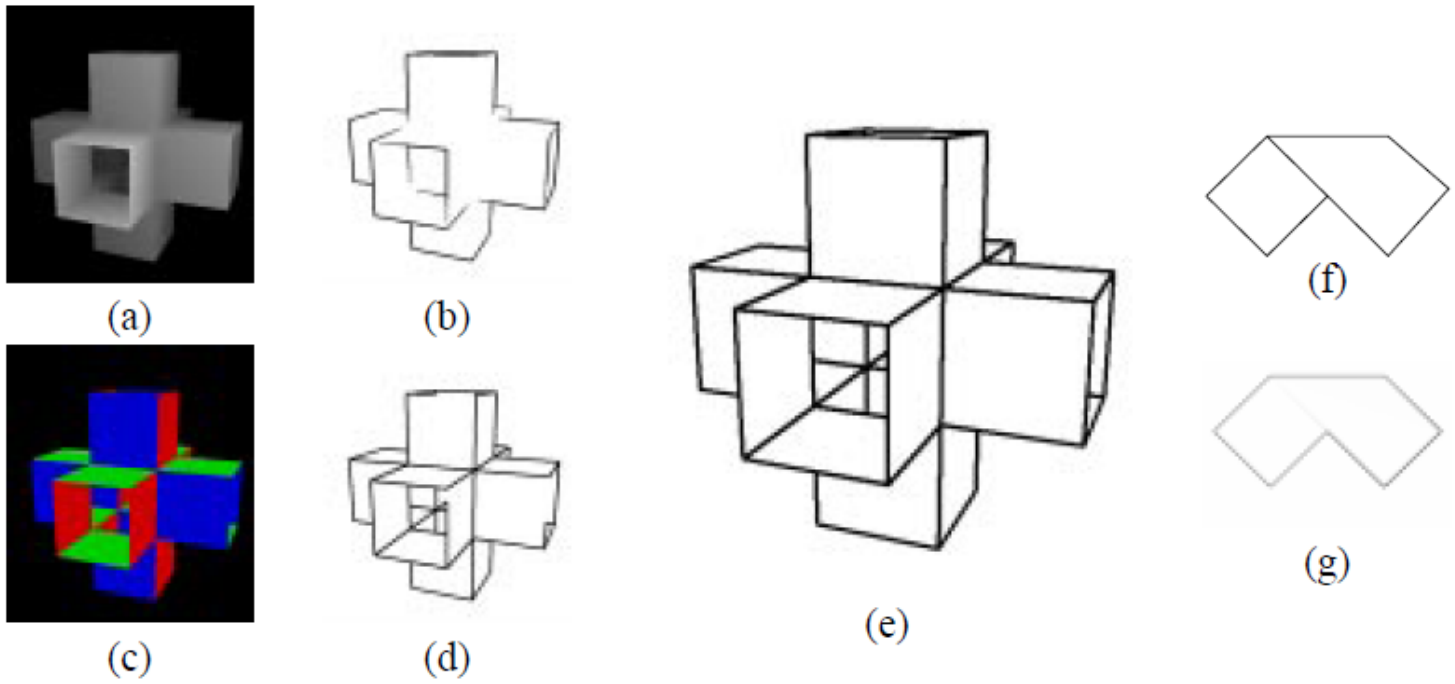


Figure 1: Outline drawing with image processing. (a) Depth map. (b) Edges of the depth map. (c) Normal map. (d) Edges of the normal map. (e) The combined edge images. (f) A difficult case folded piece of paper (g) Depth edges. (See also the Color Plates section of the course notes.)

Normal Map Edges

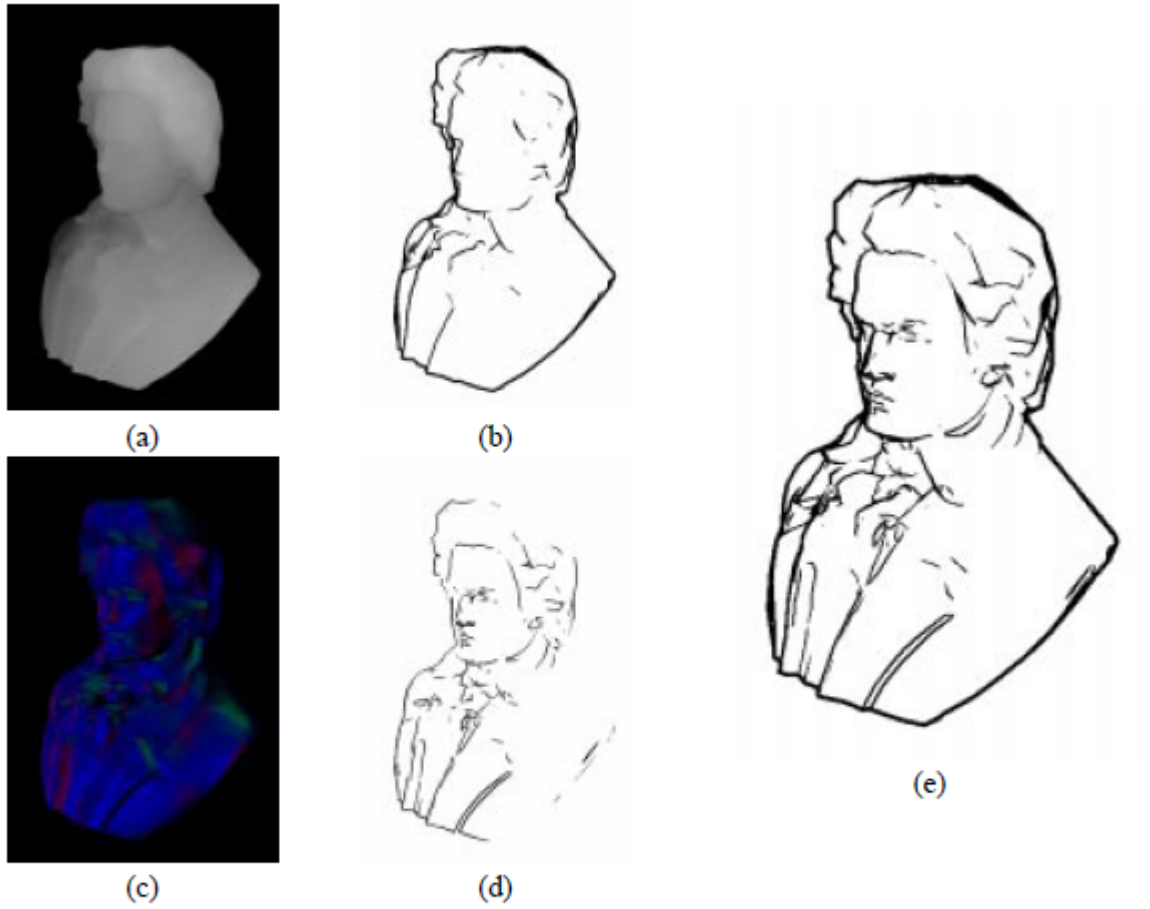


Figure 3: Outline detection of a more complex model. (a) Depth map. (b) Depth map edges. (c) Normal map. (d) Normal map edges. (e) Combined depth and normal map edges. (See also the Color Plates section of the course notes.)

Real-time Video Abstraction

Winemoller et al published a real-time technique accelerated on the GPU for image stylisation which achieves some really nice effects using image-based NPR

Similar techniques could be applied also in image space to the rendering of a 3D scene.

[Winnemöller et al] : Real-Time Video Abstraction: http://videoabstraction.net/

Texture Abstraction

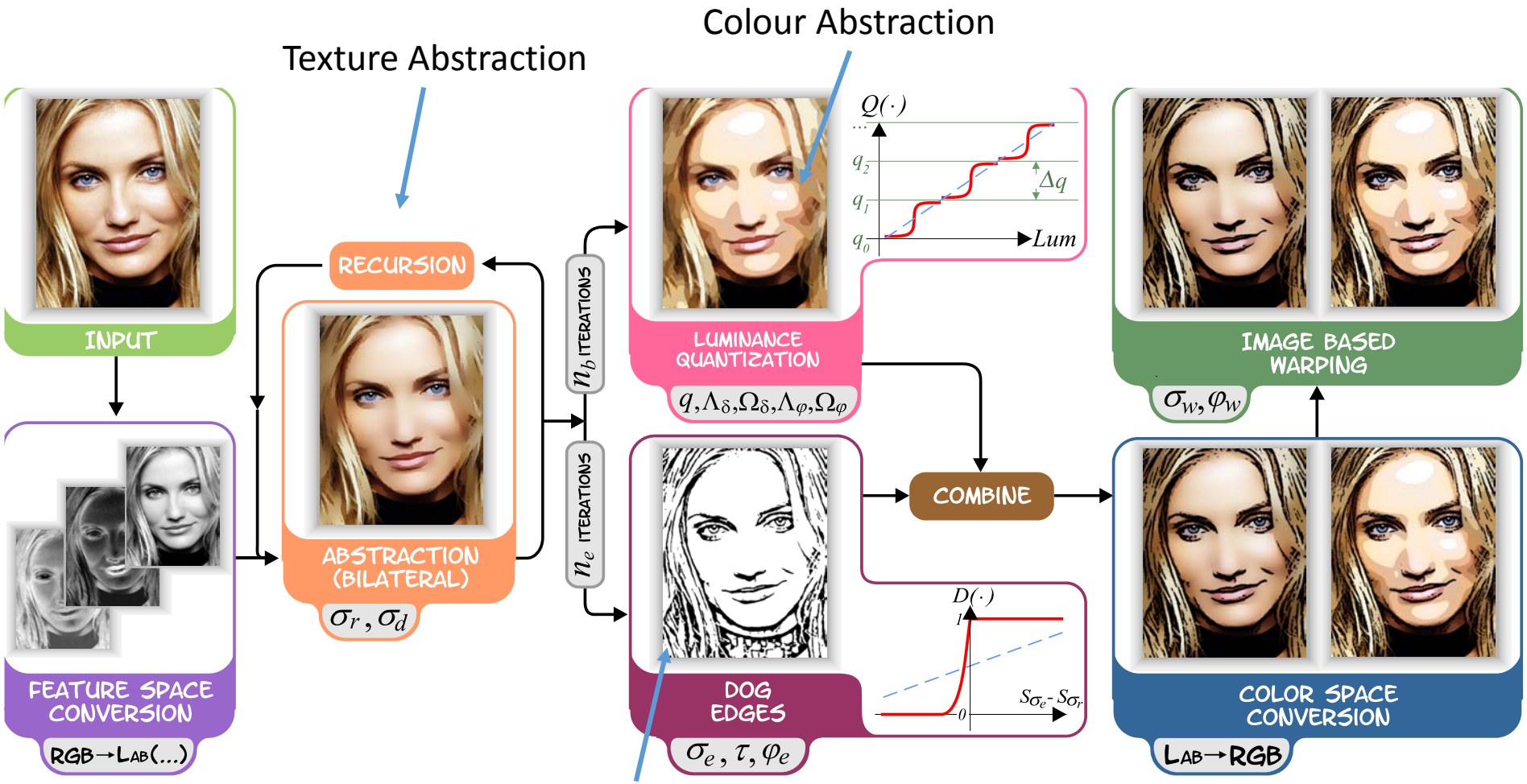
Textural detail is removed using a bilateral filter* (a.k.a. Anisotropic filter).

- Can be applied iteratively to achieve the required degree of abstraction.
- Example of an Edge Preserving Smooth which can be achieved in other ways.

* For details see: Tomasi & Manduci “Bilateral Filtering for Gray and Color Images” proc ICCV 1998

[Winnemöller et al] : Real-Time Video Abstraction: http://videoabstraction.net/

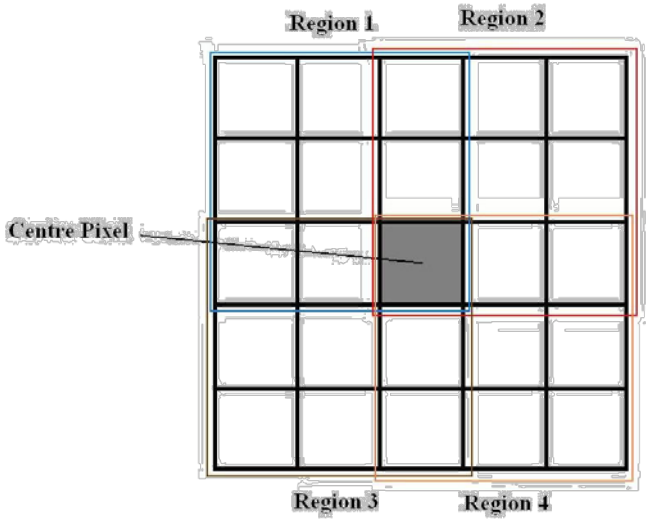
Real-time Video Abstraction



[Winnemöller et al] : Real-Time Video Abstraction: http://videoabstraction.net/

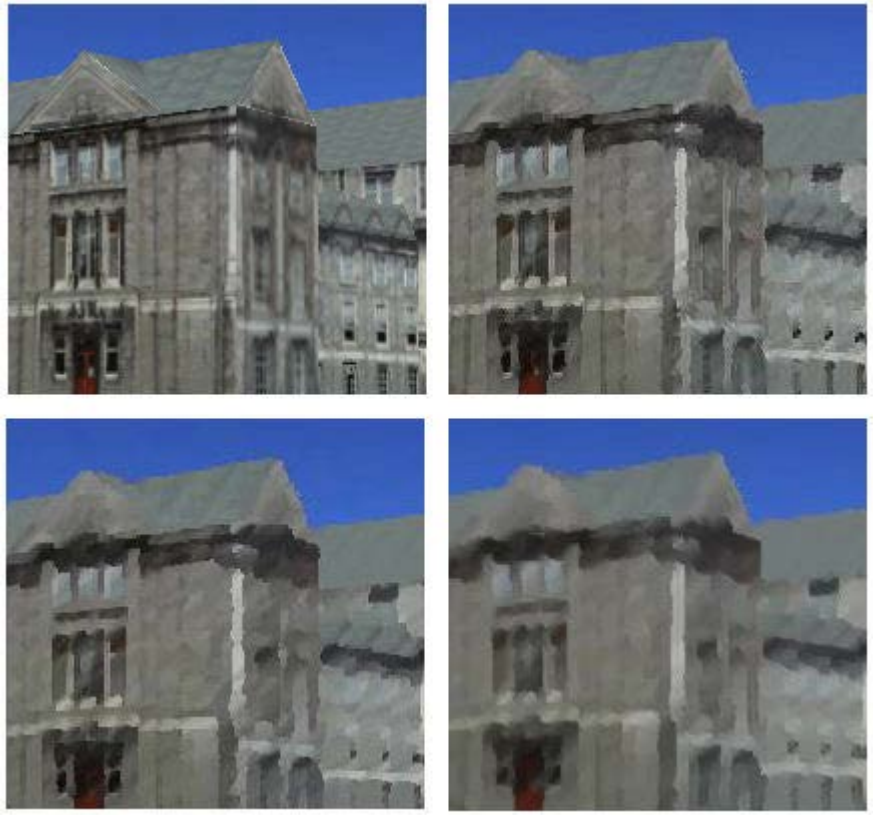
Texture Abstraction: Kuwuhara Filter

Goal: blur out high frequency details but preserve edges



Algorithm (Kuwuhara Filter)

- For a region of pixels e.g. 5x5 centred on the current pixel X.
- Take 4 sub-regions in 4 diagonal directions.
- For each sub-region calculate the max **variance**: $V = I_{max} - I_{min}$
- Take mean of region with max V and use this as colour of X



Increasing levels of abstraction can be generated by applying this filter repeatedly or using larger region.

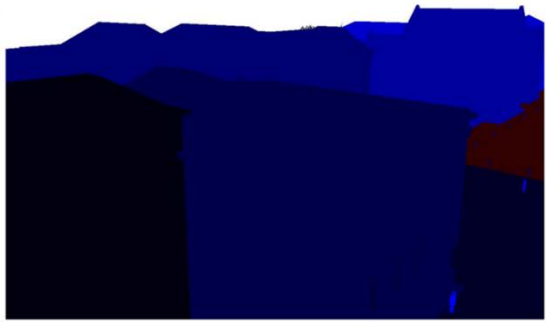
Image from redmond et al "A Hybrid Technique for Creating Meaningful Abstractions of Dynamic Scenes in Real-time" 2007

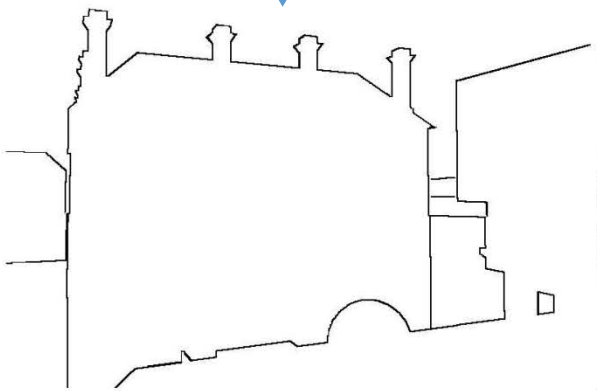
For a more detailed GPU Kuwuhara implementation see [Kypriandis 2009] http://www.kyprianidis.com/gpupro.html

Hybrid Edges

"3d" edges based on edge detection on depth/normal or id-buffer (each object rendered in a separate flat colour)

Id-buffer





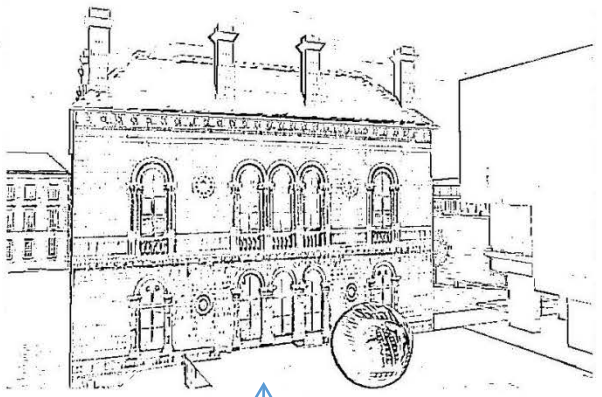
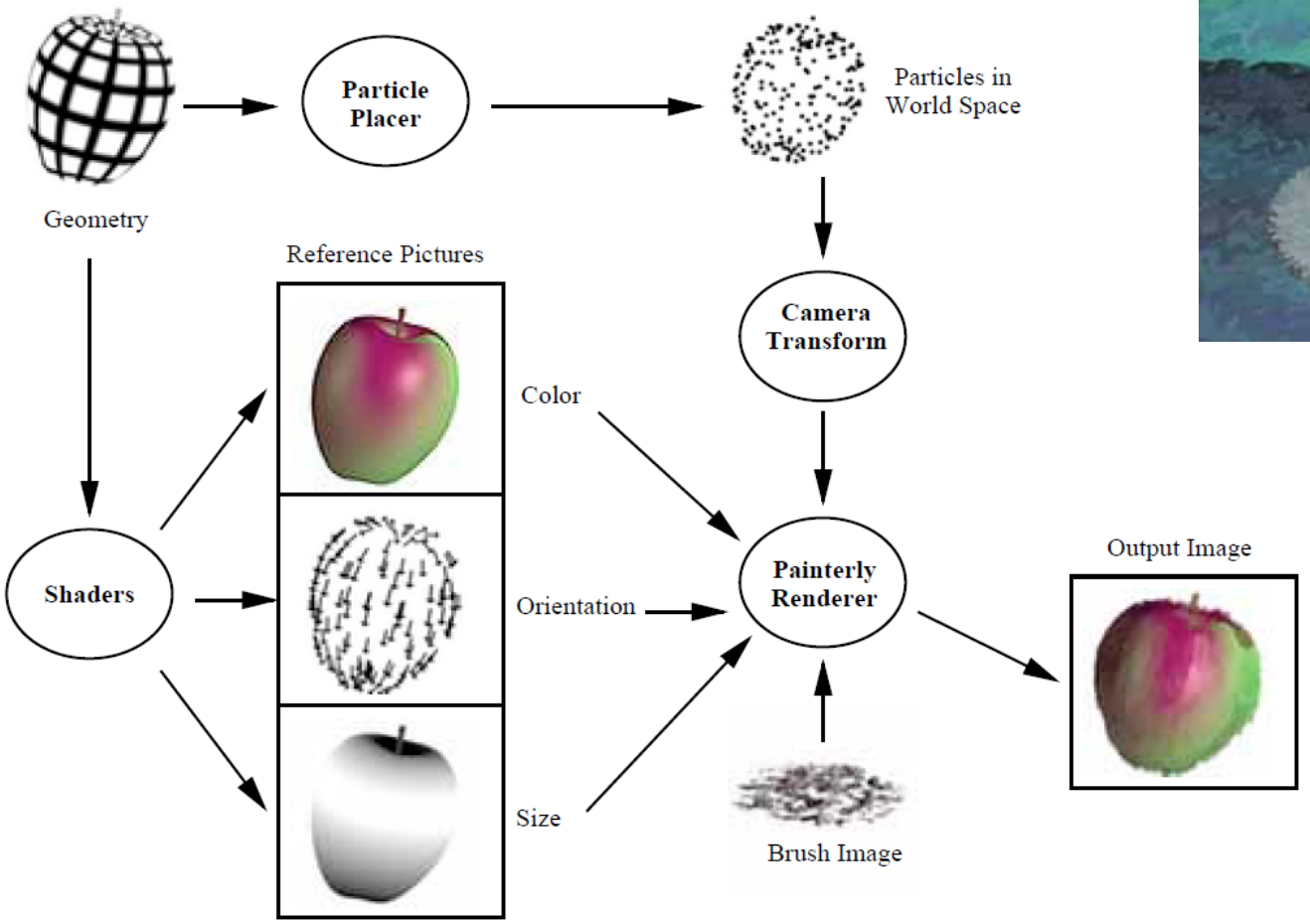


Image edges based on edge detection on the normally rendered textured scene

Combined edges and rendered scene

Images from Redmond et al "A Hybrid Technique for Creating Meaningful Abstractions of Dynamic Scenes in Real-time" 2007

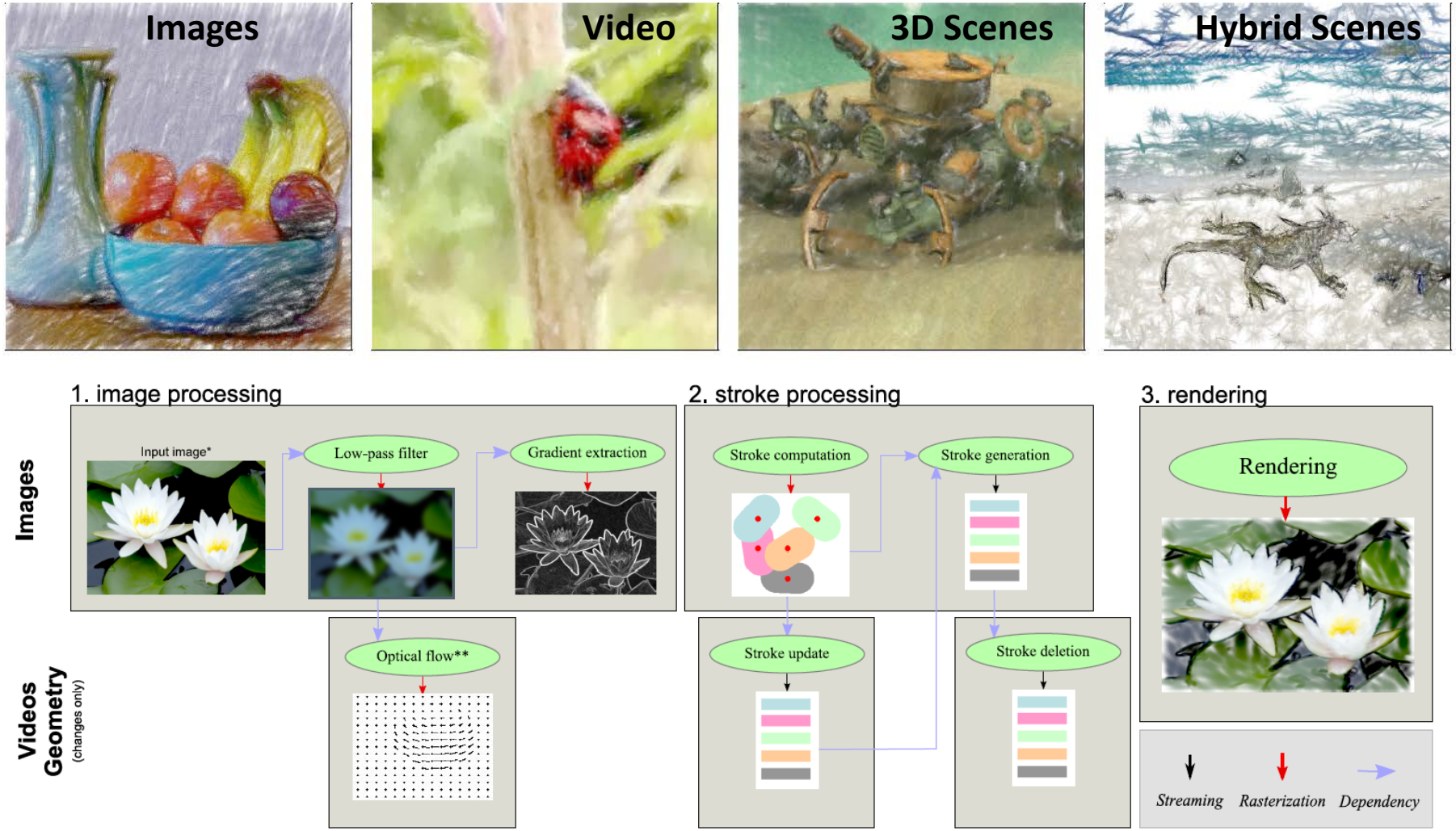
NPR Rendering with Particles



Pseudo-randomly place particles in the scene. Solve lighting at each particle and based on this render as a stroke primitive.

B. Meier “Painterly Rendering for Animation” 1996

Interactive Painterly Stylization



Lu et al. Interactive painterly stylization of images, videos and 3D animations. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games* (i3D) 2010.

Interactive Painterly Stylization

Coarse

Medium

Fine

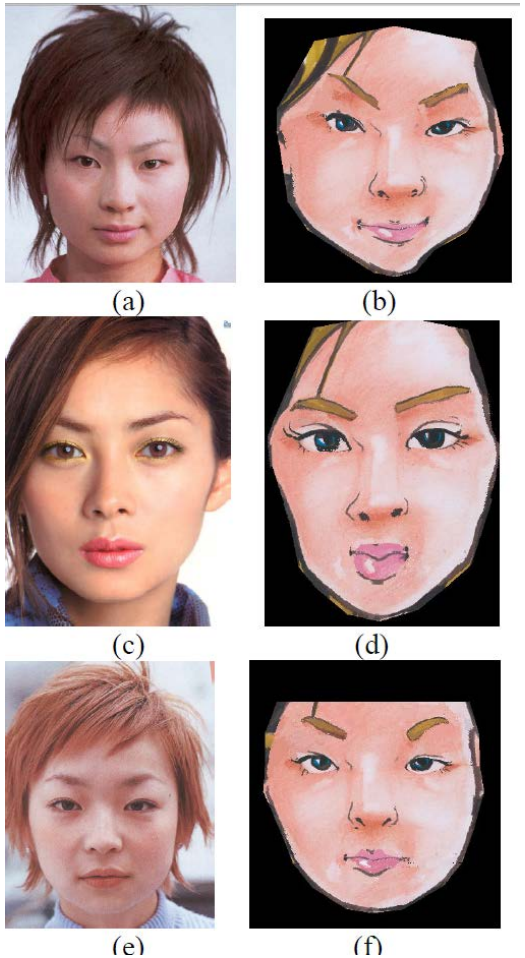
Final composite

Lu et al. Interactive painterly stylization of images, videos and 3D animations. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games* (i3D) 2010.

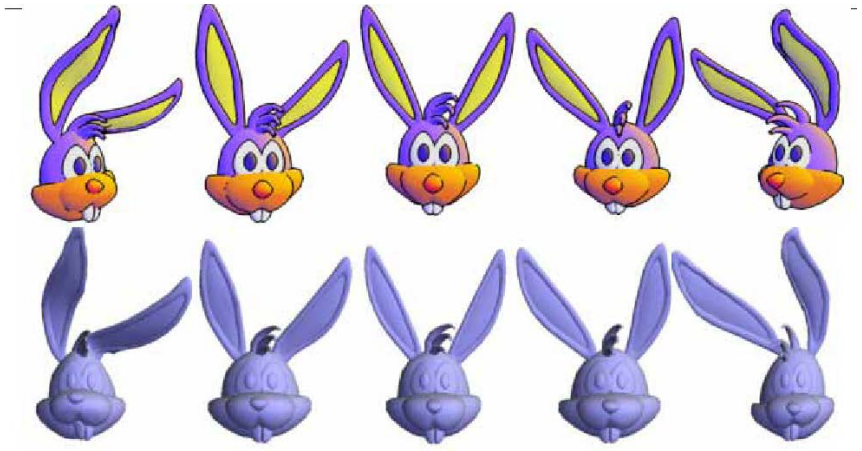
Other Related Stuff

Not Currently Real-time

Geometry and Animation Stylisation



Automatic Caricatures [Liao]



View-dependent Geometry [Rademacher]

Video stylization [Colomosse]

Cartoon Animation Filter [Wang]

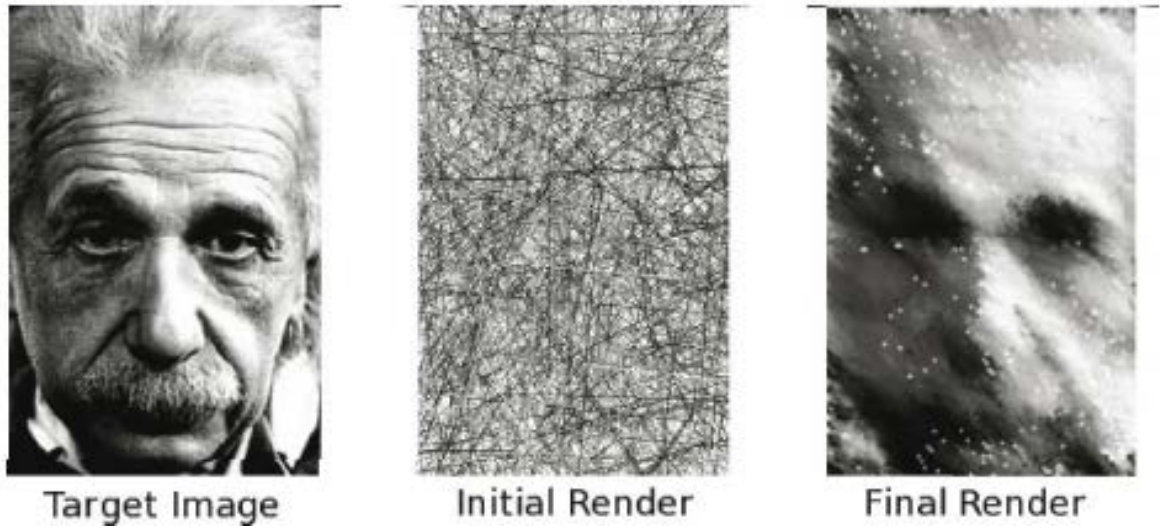
Other Stuff: NPR using AI

Artificial Ants attracted to edges navigate an image and create NPR.

From Semet et al “An Interactive Artificial Ant Approach to Non-Photorealistic Rendering” 2004.

Genetic Programming

Barile et al “Non-photorealistic Rendering Using Genetic Programming” 2008



Pigment Simulation

Real

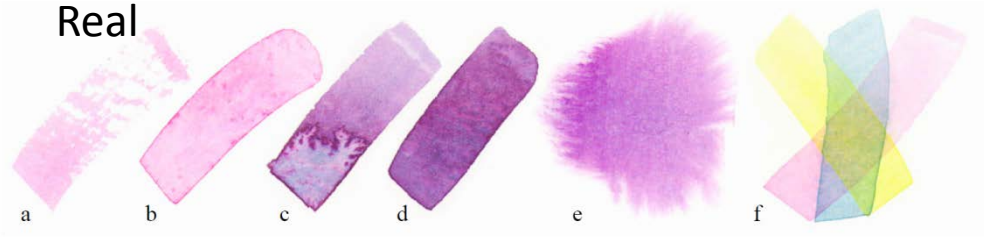
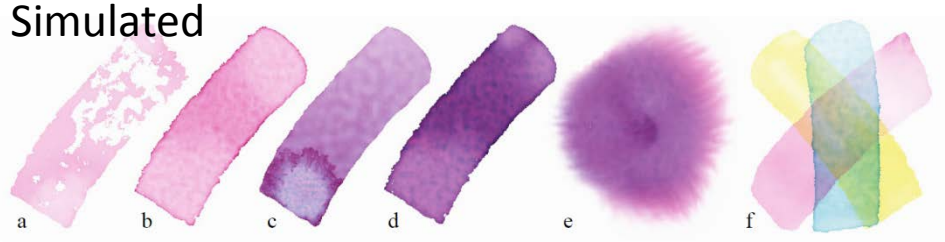


Figure 1 Real watercolor effects: drybrush (a), edge darkening (b), backruns (c), granulation (d), flow effects (e), and glazing (f).

Simulated



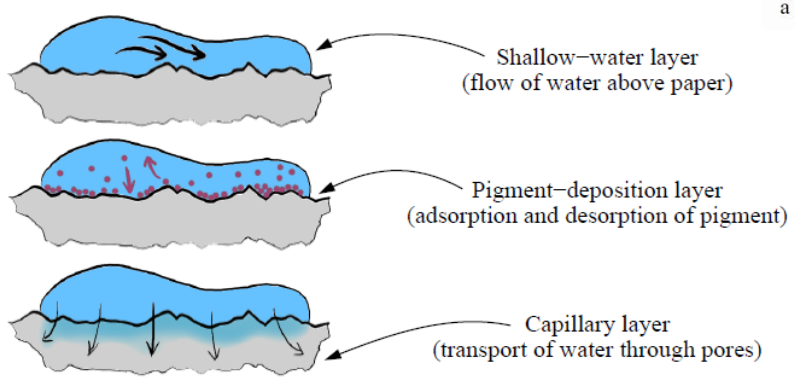


Figure 3 The three-layer fluid model for a watercolor wash.

Interaction of pigment with medium actually simulated physically using shallow water equations. Computationally expensive but quite “real”

Images from Curtis et al. “Computer generated Watercolor”. Siggraph 1997
Also see the following paper by Bousseau et al for a real-time approximation
http://artis.imag.fr/Publications/2006/BKTS06/watercolor.pdf

Issues

- Lots of different types of styles – not all work together. Dependent on:
  - What is the input data
    - 2D: Image, Textures
    - 3D: Model: mesh, parametric, points, voxels?
    - 4D: Video, Simulation/Animation
  - What type of stylisation is required
    - Edges – how detailed, classes of lines
    - Colour / Texture
    - Lighting
  - Frame to Frame Coherency?
  - How much user input vs. Automation?