Real-time NPR Techniques

cs7055: Real-time Rendering
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.
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 v.s. Automation?
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

- $C_1$ discontinuity
- $C_0$ discontinuity
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.
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.

- Slightly difficult (and expensive) to calculate these on the fly
- But could be determined at the modelling stage – or using objects parameterization
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 slightly scaled up and then draw object over this shaded. OpenGL example (Fixed Function):

```c
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:
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 realtime

Quick 2-pass method:
- apply Gaussian Blur
- subtract blur image from original to get edges
Image Space 3D Edges

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

\[
\begin{align*}
A & \quad B & \quad C \\
D & \quad X & \quad E \\
F & \quad G & \quad H
\end{align*}
\]

\( C_0 \) edges given by

\( C_1 \) edges given by

Above values need to be thresholded to only capture strong edges

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

// Look up four values from the normal/depth texture, offset along the // four diagonals from the pixel we are currently shading.
float2 edgeOffset = EdgeWidth / ScreenResolution;
float4 n1 = tex2D(NormalDepthSampler, texCoord + float2(-1, -1) * edgeOffset);
float4 n2 = tex2D(NormalDepthSampler, texCoord + float2(1, 1) * edgeOffset);
float4 n3 = tex2D(NormalDepthSampler, texCoord + float2(-1, 1) * edgeOffset);
float4 n4 = tex2D(NormalDepthSampler, texCoord + float2(1, -1) * edgeOffset);

// Work out how much the normal and depth values are changing.
float4 diagonalDelta = abs(n1 - n2) + abs(n3 - n4);
float normalDelta = dot(diagonalDelta.xyz, 1);
float depthDelta = diagonalDelta.w;

// Filter out very small changes, in order to produce nice clean results.
normalDelta = saturate((normalDelta - NormalThreshold) * NormalSensitivity);
deepthDelta = saturate((depthDelta - DepthThreshold) * DepthSensitivity);

// Does this pixel lie on an edge?
float edgeAmount = saturate(normalDelta + depthDelta) * EdgeIntensity;

// Apply the edge detection result to the main scene color.
scene *= (1 - edgeAmount);
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.
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
Real-time Video Abstraction

Texture Abstraction

Colour Abstraction

Setup

[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 [Kyprianidis 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)

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

Combined edges and rendered scene

NPR Rendering with Particles

Geometry → Particle Placer → Particles in World Space → Camera Transform → Painterly Renderer → Output Image

Reference Pictures: Color, Orientation, Size → Brush Image → Output Image

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

Interactive Painterly Stylization

Other Related Stuff

Not Currently Real-time
Geometry and Animation Stylisation

View-dependent Geometry [Rademacher]

Other Stuff: NPR using AI

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

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

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

Also see the following paper by Bousseau et al for a real-time approximation
http://artis.imag.fr/Publications/2006/BKTS06/watercolor.pdf