Non-photorealistic Rendering

Art Rendering

From: ATI Radeon® 9700 Real-Time Demos
A Brief History of (Western) Painting

- Prehistoric
- Egyptian
- Medieval
- Renaissance
  - A peak in realism
- Impressionism
- Modernism
  - And so on…
NPR – What is it all about?

- Creating Images and scenes which depict models and view according to style, not adhering to photorealism
  - Painterly rendering
  - Cartoons
  - Sketching and line art
  - Technical illustrations
  - Etc.

From: www.archicad.ca/?cat=10

Painterly Rendering

- Simulating different styles of painting
  - Brush strokes
  - Diffusion of ink
  - Color mixing and palette
  - Different styles

From: “Abstraction and Rendering of Sparsely Scanned Outdoor Environments” by Xu et al.
Example 1: Curved Brush Strokes

- “Painterly Rendering with Curved Brush Strokes of Multiple Sizes” by Aaron Hertzmann
- Using different brush sizes to capture details
- Observation: Artists start with large brushes and move on to smaller ones to add detail
The Algorithm

- **Input:** A source image and a set of brush sizes, expressed by radii: $R_1, R_2 \ldots R_n$, sorted by descending order.
  - Drawing coarse strokes layers before finer ones.

- **Algorithm:**

```plaintext
function Paint(sourceImage, R_1, ..., R_n)
    Canvas := A constant color image
    for each R
        referenceImage = GaussianBlur(sourceImage, R)
        PaintLayer(canvas, referenceImage, radius)
    Return canvas
```
The Algorithm

- Reference image – blurring the source image with a Gaussian.
  - Features smaller than current brush size disappear.

- PaintLayer(canvas, referenceImage, Radius)
  - Create a jittered grid on the image, which is radius dependant
    - Distance between pixels is roughly the brush’s size.
    - Why jittered?
  - For each pixel on the grid we compute the total error of this area
  - If larger than a tolerance, find point of maximum error, and create a stroke of color starting at that point
    - The different types of strokes depict different styles.
    - Simplest is circular stroke.
procedure paintLayer(canvas, referenceImage, R) 
{
    S := a new set of strokes, initially empty

    // create a pointwise difference image
    D := difference(canvas, referenceImage)

    grid := f_g R

    for x=0 to imageWidth stepsize grid do
        for y=0 to imageHeight stepsize grid do
            
                // sum the error near (x,y)
                M := the region (x-grid/2..x+grid/2,
                                y-grid/2..y+grid/2)

                areaError := Σ_{i,j∈M} D_{i,j} / grid^2

                if (areaError > T) then
                    
                        // find the largest error point
                        (x_1, y_1) := arg max_{i,j∈M} D_{i,j}
                        s := makeStroke(R, x_1, y_1, referenceImage)
                        add s to S
                    
                
    paint all strokes in S on the canvas,
        in random order
}
Different Strokes

Source

Impressionist

Expressionist

Colorist Wash

From: http://www.postulate.org
Example 2: Cel-shading

- A style which mimics cartoon rendering
- Properties:
  - Silhouette outlines
  - Limited number of colors in shading
    - Making the object look flat and hand-drawn

From:

From:
http://www.maxon.net/pages/products/modules/sketchandtoon/fill_e.html

© Michael Welter
Example 2: Cel-shading

オラはここで育った孫悟空だ！！ とっとと帰れ！！！
Cel-shading – Algorithm

- Render the silhouette:
  - Move each vertex slightly in the normal direction
  - Render back faces in solid color
  - Using inverted back-face culling.

- Render the front faces
  - Compute shading for each fragment
  - Quantize result
More Examples
Example 3: Line Art and Sketching

Adaptive extraction of isoparametric curves

Importance-Based rendering
Based on “A Few Good Lines” by Sousa et al.

Humans interpret line drawings remarkably well

Used by scientists and artists to represent 3D object

*Pure* line drawings consists of lines only, no tones

Expressive, simple and fun
A few good lines

- Input: Polygonal mesh
- Main algorithm:
  - Extract feature edges and classify them
  - Construct graphs based on the edges and create chains
  - Extrude the chains in 3D to create ribbons
  - Fit splines to the ribbons edges
  - Render the ribbons in black and the model in white
Feature edge classification

- **Outline edges**
  - **Silhouette** edges shared by front- and back-facing faces
  - **Boundary** edges incident to only one face

- **Interior edges**
  - **Creases** edges between faces of dihedral angle between $(\min, \max)$
  - **Cap** and **Pit** edges situated in a convex and concave regions respectively
Cap and Pit

- Maximum curvature approximation
  - Find the minimum radius of curvature
  - \( r = \tan(\theta) \frac{|PA|}{2} \)
  - For 3D replace PC by an approximation to the normal N
  - Consider all PA\(_i\)'s
  - Estimate \( \theta_i = \arccos(N \cdot PA_i) \)
  - Take the minimal \( r_i = \tan(\theta_i) |PA_i| \)
Chaining and Creating Ribbons

- Construct 5 graphs for each type of line
  - Graph may be directed or undirected
  - Direction chosen randomly
  - Directed graph will result in more and shorter chains
- Create chains on the edges using a greedy method
- Extrude chain vertices in the normal direction
  - Amount is proportional to curvature
- Fit splines to Ribbons
Rendering process

- Chaining
- Extruding
- Rendering
- Fitting

Diagram: Geometric representation with points and lines indicating the rendering process.
Results
Pencil Rendering

From: Real-Time Pencil Rendering by Lee et al. (NPAR 2006)
Technical Illustration

- Typical characteristics:
  - Edges and silhouette lines in black
  - Matte objects in colors far from B&W, warmth of color indicates surface normal, and a single light usually produces highlights
  - No shadows
  - Metal objects seem anisotropic (not uniformly shining in every direction)
    - Usually, lines are streaked in the direction of minimum curvature.

From: A Non-Photorealistic Lighting Model For Automatic Technical Illustration by Gooch et al.
Last but not least: ASCII ART!

From: www.pingmag.jp/2006/02/09/ascii-art/

From: http://www.instructables.com/id/ASCII-Art/