Non-Photorealistic Rendering
Definitions

- The rendering of images and 3D models to look like they have been hand-drawn or painted using natural media
- More generally, the production of images which are not photo-realistic in appearance
- For artistic and/or utilitarian purposes
Non-Photorealistic Rendering

Two Main Areas

• We can achieve output which looks hand produced, most commonly by turning a digital image into an artistic version of it
  • As per Adobe Photoshop image filtering tools

• We can simulate the natural media themselves, and enable people and software to use tools like simulated paintbrushes, etc., for painting pictures. Need to achieve photorealism of paints etc.
  • As per Adobe Illustrator paint brush tools

Non-Photorealistic Rendering

Examples

By me (12 years ago) using Photoshop glowing edges, solar flare and plastic wrap image filters
Non-Photorealistic Rendering
Examples

Produced by hand by user sabena using the Art Rage software (www.artrage.com)

Non-Photorealistic Rendering
Examples

Produced by The Painting Fool (with help from CFDG)
A Painting Pipeline
Overview

Generative Process (e.g., CFDG) → Digital Image → Altered Image → Colour Regions → Painterly Rendering

- Filtering
- Segmentation
- Simulation

Photo Library

Digital Image

Painting Pipeline
Example

Final Painting (Pastels)
Lectures 7 and 8
Overview

• Image filtering
• Image segmentation
• Simulation of art materials and their usage
• Quick look at some other topics
  • Including ludic qualities of painting interfaces

Image Filtering
Overview

• Usually: performing some mathematical function on the bitmap information of a digital image to produce a new bitmap

• Sequences of individual *transformations* are performed on an image, and transformed images are layered over each other in different ways, producing *composites*

• We’ll look at some common transforms and composition methods, and at a tree structure for describing their sequential usage
Image Filtering

Examples

Image Filtering

General Routine for Transformations

- Images are represented as quadruples \{R,G,B,A\} of integer values where 
  \(R\)=red component, \(G\)=green, \(B\)=blue, and \(A\)=alpha (transparency) such that 
  \(0 \leq R, G, B \leq 255\). One quadruple per (x,y) co-ordinate in the image.

- The quadruple can be squeezed into the space of a 32-bit integer data type, using four lots of 8 bits, one for each of the R,G,B and A.

- Let \([P]_R\) be the red component of pixel P, likewise for \([P]_G\), \([P]_B\) and \([P]_A\).

- Alternatively, images are represented as quadruples of floats \{H,S,V,A\} where 
  \(H\)=hue, \(S\)=saturation, \(V\)=value and \(A\)=alpha such that \(0 \leq h < 360\) and 
  \(0 \leq s, v, a \leq 1\) (as per last lecture).

- Changes are more perceptually natural, but in practice, we usually have to do a conversion from RGB to HSV and back again.

- Basic routine: loop through each pixel (e.g., left to right, top to bottom) in the original image, and calculate a new RGB value based on the original RGB.
Image Filtering
Basic Transformations

- Define: safe(X) = min(max(X,0),255)

- Greyscale (desaturation):
  - R, G, B → \text{round(average}(R,G,B))
  - Alternatively: H → H, S → 0, V → V

- Inversion:
  - R → 255 - R, G → 255 - G, B → 255 - B
  - Same result: invert each of the first 24 bits in the 32-bit representation of the pixel

- Change brightness/saturation
  - Use HSV model: increase/decrease S or V
  - Alternatively for brightness: given a constant c, such that 0 ≤ c ≤ 255
    - R → safe(R + c), G → safe(G + c), B → safe(B + c)

- Add colour:
  - Given a fixed RGB value (r,g,b)
    - R → safe(R + r), G → safe(G + g), B → safe(B + b)
  - Alternatively (gives different results):
    - R → average(R, r), G → average(G, g), B → average(B, b)
    - Can also use weighted averages

- Add noise:
  - Given a random seed, and a range k
    - R → safe(R + r₁), G → safe(G + r₂), B → safe(B + r₃)
  - Where r₁, r₂ and r₃ are random integers between -k and k. Change for each pixel
  - Alternatively: randomly vary one of H, S or V
    - Gives different results

### Image Filtering
Basic Transformations - Examples

- Desaturation
- Inversion
- Brighter
Image Filtering
Basic Transformations - Examples

Adding Blue  Noise (k=100)  Noise (k=200)

Image Filtering
Median Transforms

- Define neighbourhoods:
  - Given a set of pairs of (x,y) movements: 
    \{((mx_1, my_1), (mx_2, my_2), ..., (mx_n, my_n))\}
  - For pixel (x,y), define N(x,y) to be the neighbourhood of pixels around (x,y) given by the set of pixels:
    - N(x,y) = \{(a,b) : a = x + mx_k, b = y + my_k, for some k such that 1 \leq k \leq n\}
  - Usually parameterise them in terms of a size \(\alpha\)
- Median:
  - Let Av(x,y) be the colour defined by the average RGB values over the neighbourhood of pixel (x,y), i.e., average(\{(a,b) : (a,b) \in N(x,y)\}), then:
    - R \rightarrow [Av(x,y)]_R
    - G \rightarrow [Av(x,y)]_G
    - B \rightarrow [Av(x,y)]_B

- Square neighbourhood (\(\alpha=1\))

- Square neighbourhood (\(\alpha=2\))

- Others: circles (\(\alpha=radius\)), plus (+) and cross (x)
Image Filtering
Median Transforms - Examples

\( \alpha = 10 \)

\( \alpha = 20 \)

Neighbourhood: Circle Square Cross

Image Filtering
Threshold Transforms

- Map ranges of pixels to a certain RGB value \((r,g,b)\)
- Range defined as a triple of pairs of RGB values: \((R_L, R_H), (G_L, G_H), (B_L, B_H)\)
  - \( R \rightarrow r \) if \((R_L \leq R \leq R_H)\), else \( R \rightarrow R \)
  - \( G \rightarrow g \) if \((G_L \leq G \leq G_H)\), else \( G \rightarrow G \)
  - \( B \rightarrow b \) if \((B_L \leq B \leq B_H)\), else \( B \rightarrow B \)
- For example, we may give a threshold of 10-20 for the red values, 30-40 for the green values, 50-60 for the blue values, and a mapping of 70, 80, 90 for the RGB components respectively. For every pixel, if one of its RGB component falls within the range for that particular colour component, it is replaced by the corresponding map. Thus a pixel with RGB value \((15,100,55)\), would be mapped to \((70,100,90)\).

\((R_L, r, R_H) = (109, 255, 154)\)
\((G_L, g, G_H) = (163, 255, 254)\)
\((B_L, b, B_H) = (11, 255, 187)\)

Q. Why is red still prominent?
Image Filtering

Lookup Transforms

- Simplified description here
- Homogenises pixels which are outside a brightness range, and stretches the brightness contrast of the remaining pixels
- Window is defined as a pair of brightness values \( (B_1, B_2) \)
- Level is defined as the multiplication factor for brightness
- Any pixel with brightness less than \( B_1 \) is mapped to black, any pixel with brightness greater than \( B_2 \) is mapped to white. Then, the brightness of the remaining pixels is adjusted so that pairs of pixels which used to differ in brightness by a unit of 1 now differ in brightness by Level
- For example, suppose \( \text{Window} = (0.5, 0.6) \) and \( \text{Level} = 5 \). Then all pixels with brightness less than 0.5 are mapped to black, all pixels with brightness bigger than 0.6 are mapped to white. Two pixels with brightness 0.55 and 0.56 would have their brightness changed to 0.53 and 0.58 to provide higher contrast

Image Filtering

Convolution Transforms

<table>
<thead>
<tr>
<th>w(-1,-1)</th>
<th>w(0,-1)</th>
<th>w(1,-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>w(-1,0)</td>
<td>w(0,0)</td>
<td>w(1,0)</td>
</tr>
<tr>
<td>w(-1,1)</td>
<td>w(0,1)</td>
<td>w(1,1)</td>
</tr>
</tbody>
</table>

- Again, we look at a simplified version
- Given a 3x3 matrix of weights as above, and letting \( P(a,b) \) be the pixel at co-ordinate \( (a,b) \)
- For each pixel at co-ordinate \( (x,y) \)
  - \( R \rightarrow \text{safe(round}(\Sigma_{ij\in\{-1,0,1\}} (w(i,j) \ast [P(x+i, y+j)]_\text{r}))) \)
  - Similarly for \( G \) and \( B \)
  - i.e., take a weighted average of the \( R, G, B \) values of the pixels around \( (x,y) \)
- Can have a matrix of any size, or, in general, a neighbourhood as before
Image Filtering
Convolution Transforms - Examples

<table>
<thead>
<tr>
<th>-1.0</th>
<th>-1.0</th>
<th>-1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.0</td>
<td>8.0</td>
<td>-1.0</td>
</tr>
<tr>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>-1.0</th>
<th>-2.0</th>
<th>0.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>2.0</td>
</tr>
<tr>
<td>0.0</td>
<td>2.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0.8</th>
<th>-1.3</th>
<th>0.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1.7</td>
<td>4.9</td>
<td>-2.1</td>
</tr>
<tr>
<td>0.98</td>
<td>-1.2</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Image Compositing
General Routine

- We have two input images that we want to produce a composite for
- Basic routine: loop through each pixel and for each co-ordinate (x,y), calculate the RGB value of the composite image at (x,y) based on the RGB values of the two input images at (x,y)
- Often use the binary encoding of the RGB values, so we can use binary operators
Let P1 be the pixel at (x,y) in the first input image and P2 be the pixel at the same co-ordinate in the second input image.

- **Add:** \[ R = \text{safe}(P1_R + P2_R), \quad G = \text{safe}(P1_G + P2_G), \quad B = \text{safe}(P1_B + P2_B) \]
- **Subtract:** \[ R = \text{safe}(P1_R - P2_R), \quad G = \text{safe}(P1_G - P2_G), \quad B = \text{safe}(P1_B - P2_B) \]
- **Multiply:** \[ R = \text{safe}(P1_R \times P2_R), \quad G = \text{safe}(P1_G \times P2_G), \quad B = \text{safe}(P1_B \times P2_B) \]
- **Divide:** \[ R = \text{safe}(P1_R / P2_R), \quad G = \text{safe}(P1_G / P2_G), \quad B = \text{safe}(P1_B / P2_B) \]
- **Maximum:** \[ R = \text{max}(P1_R, P2_R), \quad G = \text{max}(P1_G, P2_G), \quad B = \text{max}(P1_B, P2_B) \]
- **Minimum:** \[ R = \text{min}(P1_R, P2_R), \quad G = \text{min}(P1_G, P2_G), \quad B = \text{min}(P1_B, P2_B) \]
Image Compositing
Arithmetic and Extremal Compositors - Examples

Divide A by B

Subtract B from A

Divide B by A

Subtract A from B

Image Compositing
Binary Compositors

• Take the binary representation of the R, G and B values of each pixel in the two images

• Then take each pair of bits in the two bitstrings, and apply the logical operator, as per the truth table for that operator

• Examples given using black and white images, as this is more illustrative

• Black pixels: all the bits are zero

• White pixels: all the bits are one
Image Compositing
Binary Compositors

- **AND**: the only white pixels in the composite image are those where the pixels are both white in the two input images.
- **OR**: the only white pixels in the composite image are those where one of the pixels in either input images was white.
- **XOR**: the only white pixels in the composite image are those where the pixels in the two input images were different.

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**AND**

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**OR**

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**XOR**

Image Compositing
Binary Compositors - Examples

Inputs

**AND**

**OR**

**XOR**
Image Filter Trees

Overview

• When we use graphic design software for image filtering, we work with layers, which enable composition, and each layer will be a transformed image which has been through various transformations.

• This can be modelled as a tree structure, with unary nodes (transformations) and binary nodes (compositions).

• Tree nodes in our representation:

  Source (original) image  Transformation  Compositor

Example 1 - Embossed 10

Median
Circle neighbourhood
$\alpha = 9$

Convolution

\[
\begin{bmatrix}
0.87 & -1.30 & -0.12 \\
-0.52 & 4.25 & -0.47 \\
-0.29 & -1.62 & 0.55 \\
\end{bmatrix}
\]

Convolution

\[
\begin{bmatrix}
-1.57 & -1.21 & -0.16 \\
-1.09 & 0.06 & 1.44 \\
0.50 & 1.05 & 1.13 \\
\end{bmatrix}
\]
Image Filter Trees

Example 2 - Painterly 8

Add

Subtract

And

Median

Square neighbourhood

$\alpha = 13$

Filter Feast Software

Demo Video
Image Segmentation
Overview

• Aim is to turn an image into a set of colour regions

• Two stage process:
  • Find regions of pixels with similar colours
  • Find a suitable (smooth) boundary around each region

Image Segmentation
Example - 1000 regions
Image Segmentation

Finding Regions of Similar Colour

- Need to measure how far one pixel is from another
  - In terms of their RGB colours
  - Euclidean distance in RGB space of two pixels P1 and P2:
    \[
    \text{dist}(P1,P2) = \sqrt{([P1]_R-[P2]_R)^2 + ([P1]_G-[P2]_G)^2 + ([P1]_B-[P2]_B)^2}
    \]
  - There are similar distances involving the HSV measures

---

Image Segmentation

Finding Regions of Similar Colour

- Grow neighbourhoods of pixels which are within a threshold, $T$, of the neighbourhood's overall colour

1. Start at $S=(0,0)$ and make the RGB colour there, $C$, the colour for the whole neighbourhood. Make $\text{marker}=null$. Add the point $(0,0)$ to the neighbourhood

2. For each pixel $P$ at point $(x,y)$ in the neighbourhood that has not been expanded, check each of its neighbouring pixels for membership. That is, for each point $Q$ in this set: \(\{(x+i,y+j) : -1 \leq i,j \leq 1\}\) which is not already in the neighbourhood, check if $\text{dist}(S,Q) \leq T$. If so, add $Q$ to the neighbourhood, (unless it is already in there).

3. If you have looked at every member of the neighbourhood, and no new points have been added, then seal the neighbourhood as a region of colour. Then, look for the next pixel position which hasn’t been assigned to a neighbourhood yet (look left to right, top to bottom from $(0,0)$). If none can be found, then stop. Otherwise, make the new pixel the new $S$, and start from 1.
Finding Regions of Similar Colour

Example

1. \( N = \{(0,0)\} \), neighbourhood colour=blue,
\( \text{Neighbours}=\{n1=(0,1), n2=(1,0), n3=(1,1)\} \)
\( \text{dist}(n1,\text{blue})=\text{dist}(n2,\text{blue})=\text{dist}(n3,\text{blue})=0 \)

2. \( N = \{(0,0),(0,1),(1,0),(1,1)\} \),
\( \text{Neighbours}=\{n4=(0,2), n5=(1,2), n6=(2,0), n7=(2,1), n8=(2,2)\} \)
\( \text{dist}(n4,\text{blue})=\text{dist}(n6,\text{blue})=\text{dist}(n7,\text{blue})=0, \text{dist}(n5,\text{blue})=\text{dist}(n8,\text{blue}) > 0 \)

3. \( N = \{(0,0),(0,1),(1,0),(1,1),(0,2),(2,0),(2,1)\} \)

4. \( N = \{(0,0),(0,1),(1,0),(1,1),(0,2),(2,0),(2,1),(0,3)\} \)

5. \( N = \{(0,0),(0,1),(1,0),(1,1),(0,2),(2,0),(2,1),(0,3),(0,4)\} \)

6. \( N = \{(0,0),(0,1),(1,0),(1,1),(0,2),(2,0),(2,1),(0,3),(0,4)\} \)

7. Search and find \((1,2)\) . Set \( N = \{(1,2)\} \), neighbourhood colour=red

8. And so on...

Image Segmentation

Finding Regions of Similar Colour

- Possible pre-processing stage:
  - Start with a fixed set, \( S \), of colours from a palette
    - For instance art deco, art nouveau, urban chic, etc.
  - Map each pixel in the image to one colour from \( S \)
    - Such that it is closest to that colour than any other from \( S \)
    - Essentially another image filtering method
  - For each pixel \( P \) of colour \( X \), determine colour \( Y \in S \) such that
    \( \nexists Z \in S \text{ for which } \text{dist}(X, Z) < \text{dist}(X, Y) \)
  - Then transform your image: \( R \rightarrow [Y]_R, \ G \rightarrow [Y]_G, \ B \rightarrow [Y]_B \)
  - Make \( T = 0 \)
Image Segmentation

Finding Regions of Similar Colour

- Possible post-processing stage
  - Given a desired number of colour regions to achieve, N, merge together neighbouring regions until you have exactly N. Set T fairly low, so that you start with more regions than N
- Possible merging scheme:
  - Start with the smallest region (in terms of the number of pixels in the region), X, and merge this into the neighbouring region which is (a) the largest or (b) the closest in colour to X. Merge by adding together the pixels, and calculating a weighted sum for the colour of the combined region.
  - Alternatively, perform some boundary analysis, i.e., certain small regions might be highly contrasting to those around it, and hence should not be merged
  - Recalculate the region size and start again with the smallest region, or stop if you have N regions
- It is likely that you will want to sort the regions in descending size order, for rendering purposes (i.e., so that small regions are not covered up)

Finding and Smoothing Region Boundaries

- Define a boundary pixel in a colour region as having fewer than 8 neighbours which are also in the region
- In The Painting Fool:
  - Puts on a circle of radius 4 in place of each boundary pixel
  - Then reduces the region size by removing any boundary pixels. Does this 4 times
  - Then performs a backtracking search around the boundary pixels to identify a sequence of pixels which goes around the region
  - Does this for holes in the shape too
  - Then for each boundary sequence, it reduces the number of points defining the sequence, to have a minimum user-given distance between them
  - Then draws a smooth curve through all of the remaining boundary pixels (not using Bezier curves...)
Brief overview of Active Contours

- AKA ‘snakes’, used a lot in medical vision
- General idea: iteratively improve a path around the region until the path approximates the boundary of the region (but remains fairly smooth)
- Improvement is defined in terms of minimising an energy function which has two (possibly opposing) components
  - Firstly: internal energy is based on the distance between points in the path and the amount of curvature. So, lower energy paths have less curvature and less elasticity (the points are closer to each other)
  - Secondly: external energy is based on the amount of contrast in the image at the co-ordinate where the path point is
- The two energies give rise to a relatively smooth curve with relatively good fidelity to the region outline

Iteration 1 - start off with path points equally spread out and close to the region
Iteration 2 - move points closer into the region boundary to find places of low external energy
Iteration 3 - some points cannot go for lower external energy because it increases internal energy
Brief Overview of Bezier Curves

- Given a set of boundary points, Bezier curves can be used to fit a smooth curve through them. There are linear, quadratic, cubic, etc., and general curves.

- Cubic curves rely on two control points per pair of boundary points which guide how the curve goes between the two boundary points.

- The curve between two points is parameterised by $t \in [0, 1]$. When $t = 0$, the curve is at the first point, when $t = 1$, the curve is at the second point.

- Given boundary points $P_0$ and $P_3$, with control points $P_1$ and $P_2$, the points on the cubic curve between them can be calculated by:

  $$B(t) = (1 - t)^3 P_0 + 3(1 - t)^2 t P_1 + 3(1 - t)t^2 P_2 + t^3 P_3$$

- Bezier curves are useful for users to define curves, who tweak the control points, and they have advantages that they scale uniformly, etc. However, for our purposes, there are issues with choosing the control points (continuity is best when pairs of control points on either side of a boundary point are in a line).
Beziers Curves

Video

Suppose that points 0, 3, 6, 9 and 12 form the boundary of our colour region

See the usage of continuity hints

http://www.cs.princeton.edu/~min/cs426/jar/bezier.html (Applet + iPhone App!)

Quick Recap

• We now know how to filter an image to make it look different to start with, and how to find regions of similar colour and draw smooth boundaries around them

• Giving us a set of colour regions

• Which represents a given digital image

• Two remaining questions:

  • How do we simulate natural art materials for the strokes?
  • How do we break the paint regions into individual strokes?
Simulating Natural Media

• Need to simulate three things:
  • Backing materials: papers, canvases, etc.
  • Painting implements: brushes, palette knives, etc.
  • Pigment materials: pastels, paints, pencils, etc.
• We’ll concentrate on achieving the look of pencils, acrylic paints and watercolours

Possible Approach

• Using shape grammars as per the last lecture
Simulating Natural Media

Watercolour brush from Adobe Illustrator

- Simple method:
  - Scale, rotate and bend a fixed rendering process/template to fit the stroke path
  - This is effective, but there is too little variety in the strokes

Brush Paths

- Another simple method:
  - Given all the points on a stroke path, continuously sweep an image along the entire path of the stroke
  - For example, calligraphy strokes keep the pen nib at the same angle
  - Should soften the edges using transparency (alpha values)

Using seashore graphic design software
http://seashore.sourceforge.net/
Brush Paths

More Examples (from Adobe Photoshop)

- Enable ridging to simulate bristle paint load/height differences
- Enable starting and ending differences: brush head and/or transparency
- Rotate the brush head to match the curvature of the stroke
- Enable paints to mix as they are painted
- Simulate unmixed paints
- Simulate smudging and paint dispersion

Made with Art Rage: http://www.artrage.com/
Brush Paths
Adding Sophistication

- Increasing bristle colour variance
- Increasing intro/exit transparency
- Becoming increasingly grainy

Dispersion and bristle size

Made by The Painting Fool: www.thepaintingfool.com

Advanced Topics
in Non-photorealistic rendering

- Modelling watercolour ink flow
- Modelling graphite pencils
Simulating Watercolour Ink Flow

Simulating Graphite Pencils

FIGURE 4.6 Examples of computer-generated watercolor paintings: (a) was created using the CA approach, whereas (b) is the result from the fluid simulation technique. (See also color insert.)

FIGURE 4.13 Simulation results made by automatically sketching predefined images: 3B pencil (a), 6B pencil (b), and 6H pencil (c).

Some other examples of renditions of objects made with the simulation described in this section.
Painting Colour Regions

• We have ways to turn images into regions of colour and ways to simulate the look of art materials

• Each colour region can be painted in layers
  • Each layer could produce an outline of the region or fill the region in
  • In each case, we have to turn the region into a set of curves, which will be rendered as strokes

Painting Colour Regions

Outlining

• For each region, we know each point on the region boundary

• We can chop the boundary up into roughly equal sub-curves and paint each one as a stroke

• We can rationalise each stroke, for instance to be a straight line

• We can shrink the boundary and repeat the outlining a number of times
Painting Colour Regions
Outlining - Examples

Made by The Painting Fool: www.thepaintingfool.com

Painting Colour Regions
Filling

• We know the area that the region defines, so we can place strokes in the area to fill it in
• We can use parallel lines
  • To achieve cross hatching in two layers
  • Looks more natural to wobble the lines somewhat
• We can fill the region with ever decreasing circles
• We can randomly place lines inside, in different directions
• We can use a star fill approach
Painting Colour Regions
Filling - Examples

Star Fill
Example
Example Styles
Changing filters, segmentation styles, media, colour palettes

Made by The Painting Fool: www.thepaintingfool.com (see Amelie’s Progress)

Other Topics
Ludic Qualities of Painting Interfaces

• If the painting software is going to be interactive, we need to worry about how the user will enjoy using it

• Using interactive displays (tablets and tablet screens) is becoming commonplace. Thanks to the iPad, we will all interact with computers with our fingers directly onto a screen soon!

• Pens and tablets are touch sensitive, which gives another dimension: higher pressure means a thicker line (or more paint, etc)

• General area: haptic computing

http://gamma.cs.unc.edu/DAB/
Other Topics

Ludic Qualities of Painting Interfaces

Issues

• How do we approximate the lines that users have drawn? We can’t take their verbatim mouse points, because this would be too jagged. We need to experiment with smoothing algorithms.

• Should software draw the paint stroke as the user is painting it (like Art Rage), or draw a skeleton line, and then fill it in later (like The Painting Fool teaching interface). There are ludic advantages to the former (it feels better seeing the paint go on), but there are advantages to the latter - where the stroke started and stopped can be used to calculate the stroke; multiple strokes can be overlayed; strokes can be considered regions and filled in;

Other Topics

Printing

• Designing artwork for screen and for print are very different things, and it is especially difficult to get printed colours to match on-screen colours.

• Need more than 150 dots per inch (DPI) to fool the human eye (but 300 is better).

• So, for an A4 piece of paper (8.3in × 11.7in)
  • Your image needs to be at least 11.7 * 150 = 1755 pixels in its longest diameter, which is bigger than your laptop screen...
    • Hence your paint strokes have to look too big on screen.
  • For an A1 piece, you need around 10,000 pixels
  • Otherwise, people are going to see pixelation...
A Big, Big Image

- 12,000 x 80,000 = nearly a billion pixels
- Too large to fit into memory (tiling used). Took 24 processors 20 hours to produce
- Printed onto 317 photo-sized pieces of paper (600dpi) and stuck onto a 3m by 1m board

A Big, Big Image
A Big, Big Image
Other Topics

Impasto Painting
MEng. Project by Leon Dimitriou

Other Topics

Saliency Adaptive Painting and Cubism
Work by John Collomosse

http://info.ee.surrey.ac.uk/Personal/J.Collomosse/projects.php
Summary

- Non-photorealistic rendering is certainly possible, and software could possibly pass the trivial ‘Turing test’ of whether their art looks like a person physically produced it.

- We need to perform image filtering, image segmentation and simulation of natural art materials in order to achieve good effects.

- It’s a good idea to make the process as physically realistic as possible for interactive NPR.

More Links

- Interactive filtering tutorial (JAI):

- Advanced snakes model:
  - http://iacl.ece.jhu.edu/projects/gvf/

- Impasto painting software:
  - http://gamma.cs.unc.edu/IMPASTO/

- Non-realistic haptic feedback for virtual sculpture:

- NPR resources:
  - http://www.cs.utah.edu/npr/