Eye 300dpi at 30cm

Color Classification

Munsell: hue, value, chroma, with standard perceptual difference between colors
XYZ: three standard emission functions, defined by CIE: human visible values leads to a horseshoe in an x-y chromaticity diagram
Luv, Lab: perceptually uniform variants of XYZ
RGB: used in hardware, tiny triangle XYZ space
CMY: used in printers, invert RGB (absorption)
CMYK: add K because inks are not perfect absorbers, so replace achromatic portion of printed output with K
HSV, HLS: as Munsell, better for humans than using RGB

CRTs Electron gun on phosphor screen
Electromagnets to focus, deflect
Can add a shadow mask and a grid of differently colored phosphors for color display

LCDs Two layers of liquid crystal: turn off the twisting effect with a voltage to blank pixel out

Plasma Voltage across electrodes ionizes gas to give UV, excites phosphor

Printers Grayscale by halftoning (clumped dot) / dithering (randomized dot)
Color with multiple halftoned screens: angle to prevent Moire
More colors = larger gamut

Laser Charged drum selectively discharged by laser, coated with toner, pressed and cleaned

Inkjet Electrodes, bubbles, piezo surfaces or electrical fields pull or push ink onto a paper surface

Bresenham (integer end points, octant 1)

```
Dy = (y1 – y0); Dx = (x1 – x0);
y = x0; yf = 0; y = y0;
DRAW(x, y); while (x < x1) do {
  x++; yf += 2*Dy;
  if (yf > dx) { y++; yf -= 2*Dx }
  DRAW(x, y); }
```

Avoid floating point on yf by multiplying all operations involving it by 2*Dx. Can modify for FP operations by un-optimising and changing start point finding algorithm for floats

Difference Method (line, octant 1)

Observation: if $k = ax + by + c$ then k < 0 = above line, k > 0 = below line, k = 0 = on line
Given that a pixel is on the line the next pixel is either E or NE: make decision at (x+1, y+½)
If E then d' = d + a else d' = d + a + b

```
a = (y1 – y0); b = (x0 – x1);
c = y0*x1 - x0*y1; x = ROUND(x0);
y = ROUND(y0 – (x – x0)*(a/b));
d = a*(x + 1) + b*(y + ½) + c;
DRAW(x, y); while (x < (x1 - ½)) {
x++; if (d < 0) { d += a; } else
{ y++; d += a + b; }; DRAW(x, y);}
```

Difference Method (circle, octant 2)

$k = x^2 + y^2 - r^2$: k<0 = inside, k>0 = outside
Make decision at (x+1, y-½). Either E (d' = d + 2*x + 3) or SE (d' = d + 2x – 2y + 5)
Can extend to ovals, but use points of 45° slope, not octants and must be axis aligned.

Bezier Cubics

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

Continuity C_1: continuous in position and tangent vector
G_1: continuous in position, tangent vector in same direction
C_0: continuous in position only

Drawing Naïve method: use a fixed step size to draw some lines. But cannot fix step so all Beziers look good, and distance in real space not linearly related to distance in parameter space
Adaptive subdivision: keep dividing up the task of drawing until a straight line is "good enough" to approximate it. Test goodness by checking that P_1,P_2 are not more than d from the line between P_0 and P_3. Testing this distance done by finding s st. P(s) is closest to a fixed C: need $s = \frac{\overline{AB}.\overline{AC}}{|\overline{AB}|^2}$ (see p137)

Overhauser's Cubic

As Bezier, but don't have tangent vectors: instead, work one out from surrounding data points. Tangent at P_n is ½($P_{(n+1)}$ – $P_{(n-1)}$). Hence for points A, B, C, D have Bezier P_0 = B, P_3 = C, P_1 = B+(C-A)/6, P_2 = C-(D-B)/6

Douglas & Pucker

Simplify line chains: approximate chain as straight line, find C in chain at greatest distance from line, if this exceeds threshold approximate as 2 recursively simplified chains

Cohen-Sutherland

4 bit code for each segment of the plane divided by box lines: A=x<x_L, B=x>x_R, C=y<y_B, D=y>y_T, Q=ABCD. If Q_0=Q_1=0,

inside rectangle (accept), if $Q_1 \& Q_2 != 0$ both ends outside and in same half plane (reject), else intersect line with edge and start again (the 1 bits tell you which to clip against)

Scanline Filling

1. Take polygon edges and place in edge list sorted on lowest y value
2. Start with first scanline in polygon (lowest y): edges intersecting this move to the active edge list (AEL)
3. Repeat until AEL empty:
    a. For each edge in the AEL find the intersection point with the scanline, sort into ascending x
    b. Fill between pairs of intersection points
    c. Move to the next scanline, remove edges from AEL if endpoint < y, move edges to AEL if start point ≤ y

Efficiently calculate intersection points with incremental line drawing (store current x, dx, starting/ending y, do x+=dx on increment)
Be careful with endpoints exactly on scanlines!

Sutherland-Hodgman Polygon Clipping

Clip arbitrary polygon against convex polygon by iteratively clipping it by the edges of the convex one. Clip to a line by going around polygon edges keeping track of inside/outside and outputting appropriate points

Transforms

2D rotation: $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

3D rotation (about x-axis):

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

Homogenous Coordinates

$(x, y, w) = (\frac{x}{w}, \frac{y}{w})$

Allow translations: $\begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix}$

Concatenate by pre-multiply (non-commutative)

Projection

Parallel: $(x, y, z) \to (x, y)$

Perspective: $(x, y, z) \to (\frac{x}{z}, \frac{y}{z})$

Viewing Transform

For screen centre (0, 0, d) parallel to xy plane, z-axis into screen, y-axis up, x-axis to the right, eye at origin we have $(x', y') = (x\frac{d}{z}, y\frac{d}{z})$. Now need to transform world so these assumptions are met

For camera at (e_x, e_y, e_z), look point at (l_x, l_y, l_z), up along vector (u_x, u_y, u_z):

1. Translate eye point to origin
2. Scale so that eye point to look distance $|\overline{el}| = d$
3. Align $\overline{el}$ with z-axis by rotating about the y-axis into yz (angle $\theta = \cos^{-1}(\frac{l''_z}{\sqrt{l''^2_x + l''^2_z}})$) and then about the x-axis into z (angle $\theta = \cos^{-1}(\frac{l'''_z}{\sqrt{l'''^2_y + l'''^2_z}})$)
4. Ensure the up vector points along the positive y-axis by rotating around the z-axis (angle $\theta = \cos^{-1}(\frac{u''''_y}{\sqrt{u''''^2_x + u''''^2_y}})$)

Coordinates

Object □Modelling World □View. Viewing □Proj. Screen

3D Clipping

Front and back clipping planes clipped to on the viewing frustrum or 2D projection of it (must retain z during projection to use this)

Bezier Patches

$$P(s,t) = \sum_{i=0}^{3}\sum_{j=0}^{3} b_i(s) b_j(t) P_{i,j}$$

As Beziers w/ 16 control ps. Continuity similar.

Drawing

Simple method: use fixed increments to approximate the patch with polygons
Tolerance method: 3D extension of that for Beziers. Need to watch out for gaps in the resulting surface!

Depth Sort Rendering

Transform polygons into 2D retaining Z information and then do a ordering on z to get draw order: resolve ambiguities (overlapping) by splitting one polygon by the plane of another

Back Face

Remove those faces of a

Culling closed polygon that have normal vectors away from the viewpoint

BSP Tree

1. Select a polygon as the root
2. Divide remaining polygons into those in front of the selected polygon and those behind (those that are both are split into two)
3. Make two BSP trees, one from each subset: they are the front/back

Then when drawing, and viewpoint is in front of the root polygon: draw the back child tree, draw the root polygon, draw the front child tree

BSP can be reused between viewpoints (unlike sorting)

Z-Buffer As 2D scan conversion, but store written pixel z value and only overwrite the pixel if the incoming one is lower

Can interpolate z between points just as x is already

Anti-aliasing Alleviate effects of sampling (jaggies, lost polygons etc)

Area averaging: clip polygons to scanline, work out exact contribution!

Super-sampling: sample on a finer grid, take average

A-Buffer Sub pixel sampling only required in pixels partially covered by a polygon

Store list of masks per pixel in depth order showing how much is covered by a polygon

When drawing, iterate down the mask list finding out how many pixels are actually covered, do weighted average of mask colors for final color

Can discard masks behind a mask which is all 1s

To calculate mask calculate the mask for each edge bounded by the right hand side of the pixel (use lookup table) then XOR all masks

Diffuse Shading $I = I_l k_d (N.L)$ (L = normalized light source vector, N = surface normal, k_d = portion diffusely reflected, I_l = light source intensity)

Gourad Shading Calculate the diffuse illumination at each vertex rather than each polygon, interpolate it across polygon

Phong Shading $I = I_l k_s (R.V)^n$ (R = vector of perfect reflection, V = normalized viewer vector, k_s = portion specularly reflect, I_l = intensity, n = roughness coefficient)

For a polygon, interpolate the normal across the polygon to be able to calculate the reflection vector and do Phong shading at each point

Ambient Light A hack to simulate diffuse reflections: $I = I_a k_a$

Texturing Find texture space coordinate for object space coordinate: nearest neighbour, bilinear reconstruction, bicubic

If a pixel covers a large area of the texture must average texture across the area (down-sample): store multiple versions of the texture in MIP map to avoid doing this

Use texture to modify transparency, reflectiveness, surface normal (bumps)

Ray Tracing Shoot a ray from the eye through the centre of each observed pixel, take the colour of the closest object hit

Ray-Plane Intersection $P = O_b + sD$, $P.N + d = 0$, hence easy to find s

Ray-Polygon Intersection Intersect with plane of polygon then draw line from intersection to infinity and say an odd number of intersections with polygon edges means point is inside

Ray-Sphere Intersection $(P - C).(P - C) - r^2 = 0$, can find intersection by solving quadratic equation. No intersection: imaginary results

Special Effects Once you have the intersection point, normal can be found and hence shoot rays to lights to get diffuse/specular reflection with shadowing
Spawn new rays to determine mirrored color (beware cycle)
Allows for transparency and refraction by continuing ray

Sampling Single point, super sampling, adaptive super sampling
Grid, random, Poisson disc, jittered sampling methods

Distributed Ray Tracing Distribute multiple samples over some range
Anti-aliasing (distribute sampling rays over pixel area)
Soft shadows (distribute rays to area light source over some range of angles)
Depth of field (distribute camera position over a range)
Motion blur (over time)

Convolution Filtering
Blur: $\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$

Gaussian: $\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$

Edges: $\begin{bmatrix}1 & 1 & 1\\ 0 & 0 & 0\\ 1 & 1 & 1\end{bmatrix}$

Median Filtering Take median value of pixels in neighbourhood as new value (good for shot noise)

Point Processing Invert image, improve contrast, modify filter output, gamma correction ($p' = p^{1/\gamma}$)

Misc. Arithmetic (multiplication, subtraction), alpha blending

Halftoning Grow halftone dot from the centre, pixels must be connected (for printing)
Use simple matrix of numbers to store dot growth sequence

Ordered Dither Growth matrix as before, but pixels are evenly spread

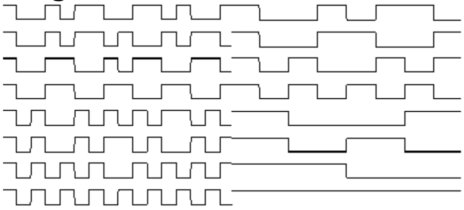
1-1 Pixel Mapping Turn a pixel on if its intensity is greater than or equal to the value of the corresponding cell in the tiled growth matrix

Error Diffusion Accumulated error in quantisation is pushed out to surrounding pixels and effects the direction in which they are rounded. Usually push dither down and right in ratio of 1:1

Encoding Variable length symbols
Difference mapping (pixels similar to those on each side)
Predictive mapping (use known values to guess next)
Run length encoding: simple or alternating regions of N different, M similar pixels

Transforms Transform N pixel values into coefficients on N basis functions, quantise these

Walsh-Hadamard H(u,v,x,y) is an array of weights. 1D version:

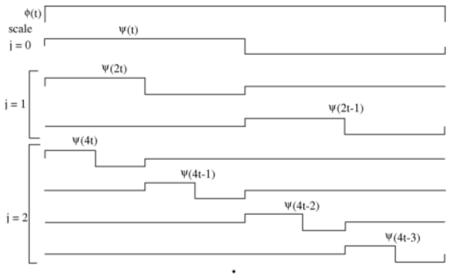


$h(x,y,u,v) = \frac{1}{N} H(u,v,x,y)$

Forward Transform $F(u,v) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)h(x,y,u,v)$

Backward Transform $f(x,y) = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} F(u,v)H(u,v,x,y)$

Wavelets Localised transformation function, scaled and shifted
Haar basis functions:



JPEG
1. Subtract 128 from each pixel value
2. Process each 8x8 image block in turn
3. Get the 2D DCT for each block
4. Quantise each coefficient by the values in the quantisation matrix
5. Linearize the quantised coefficients
6. Encode coefficients: DC coefficient coded relative to previous block, variable length code for non-zero AC coefficient + its preceding string of 0s

(i.e. anticipate many 0s in the output)