TEXTURE MAPPING ≡

HOW TO ACHIEVE VISUAL COMPLEXITY
make things look interesting

① METHOD 1:
LOTS OF SMALL PIECES

MODEL THE COMPLEXITY (like real world)

Why not?
- Hard to model
- Hard to draw fast
- Sampling issues (lots of triangles / pixel)
- Hard to maintain the models
- Hard to store the models

An aside – what are the pieces?
Triangles
- single color
- color per vertex
  - GOURAUD SHADING

② METHOD 2 – fewer big pieces
complex colorings on each piece

paintings of flat objects (analogy)

Why painting rather than modeling?

- easier (use 20 tools)
- less to store
- less to model
- faster to draw
- can get sampling right, easily

(less to transform) — (we'll see)

Why not?

doesn't truly model non-flat

- parallax
- self shadowing
- illumination effects

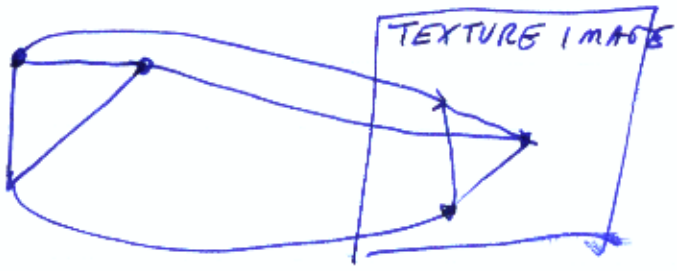
(these are advanced ways to fake this)

How do we do this?

TEXTURE MAPPING!

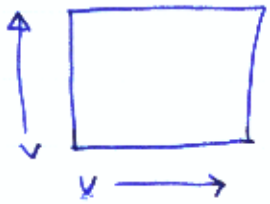
(and its variants)

map from surface to a piece of a picture (or volume)



How does this work

① Texture image
(sometimes called the texture map)



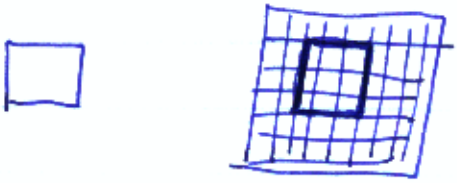
might be 3 or 1 (or 4) dimensional

② Texture Co-ordinates - for each vertex, Where in texture Image does it go

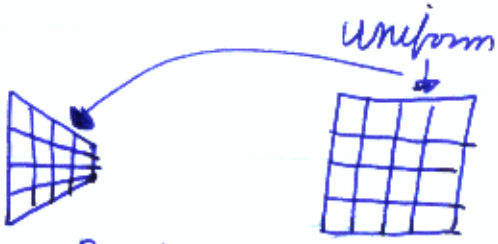
③ interpolation method - how to generate coords for points inside of triangle

as you scan triangle, each pixel generates texture coordinates

Easy - Linear interp



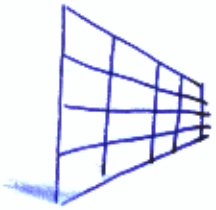
Problem: perspective fore-shortening



OLD VIDEO-GAMES

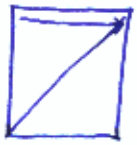
Bi-LINEAR

answer: perspective correct textures



HAVE TRIANGLE

u, v, x, y, z at each corner



as we scan

interpolate $u, v, \frac{x}{w}, \frac{y}{w}, \frac{z}{w}$

real texture co-ord is $\frac{a}{w}, \frac{b}{w}$

interpolate $a \in \frac{u_0}{w_0} \rightarrow \frac{u_1}{w_1}$

its really a little more complicated

Shirley covers this really well 9.4.1

divide per pixel - (perspetive correct)

used to be expensive

now all hardware does it

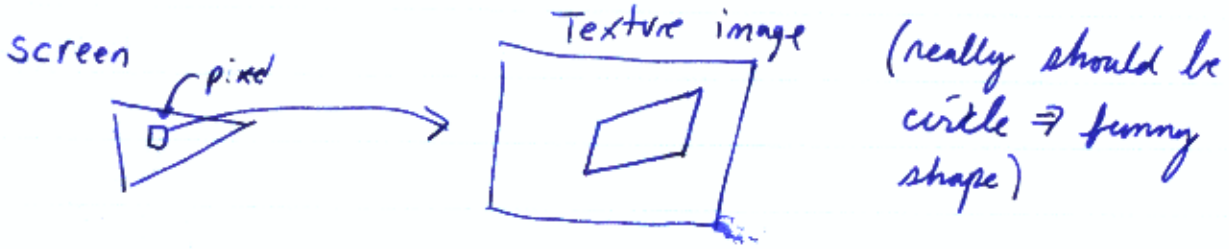
now we know the u,v for the pixel

what color is it?
look up in the image

This is the same as sampling any image

Two issues -
sampling off the grid (interpolation)
proper filtering

The problem



① dealing w/ floating point u,v (but a point sample)
Bilinear interpolate
- fine approximation if area is smaller than 1 pixel
- fast!

② filter requires averaging over area
generally need to simplify shape for speed

What simplifications on shape?

(1) rectangular regions
- summed area tables

sum over region in constant time w/ precomputed table - area above and to the left

$A = B - C - D + E$

↳ 4 lookups, but need table - overflow issues

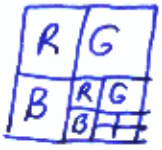
need to know rectangle

(2) square regions centered @ point

if square is small - easy
use interpolation and point sample

suppose square is not small -
decimate image until it is small

need images half size - precompute these -

Easy to store 

MIP-MAP

BUT HALF IMAGE GIVES POOR RESOLUTION -

Interpolate Levels!

TRI-LINEAR INTERPOLATION

↳ Almost all modern graphics hardware does