Computer Graphics

(Discrete Techniques, Texture etc.)

Thilo Kielmann
Fall 2003
Vrije Universiteit, Amsterdam
kielmann@cs.vu.nl

http://www.cs.vu.nl/~graphics/

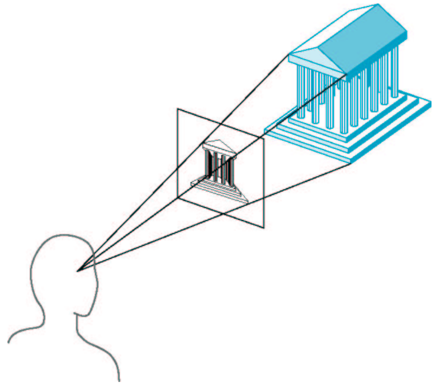
Written Exams

- the dates have been fixed now:
  - ⋆ 19 January 2004, 13:30-16:30, M129
  - ⋆ 18 June 2004, 13:30-16:30, S209 (second chance)
- registration via TISVU is mandatory! **(new)**
- rooms are subject to change, please check the boards of the "onderwijsbureau"

Outline for today

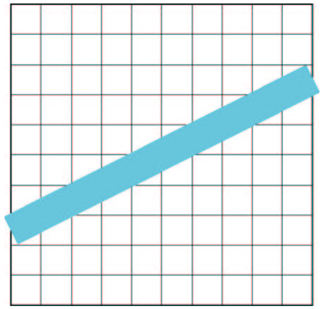
- From Geometric to Discrete Operations
- Buffer Writing
- Texture Mapping
- Environmental Maps
- Compositing (Blending)
- Special Effects (Accumulation Buffer)

Geometric Processing



Floating-point arithmetic has sufficient precision for "smooth" modeling.

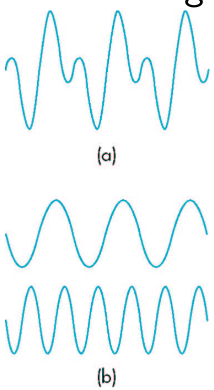
Discrete Operations



Transfer to integer-valued pixels introduces discretization problems. $\longrightarrow$ *aliasing* effects

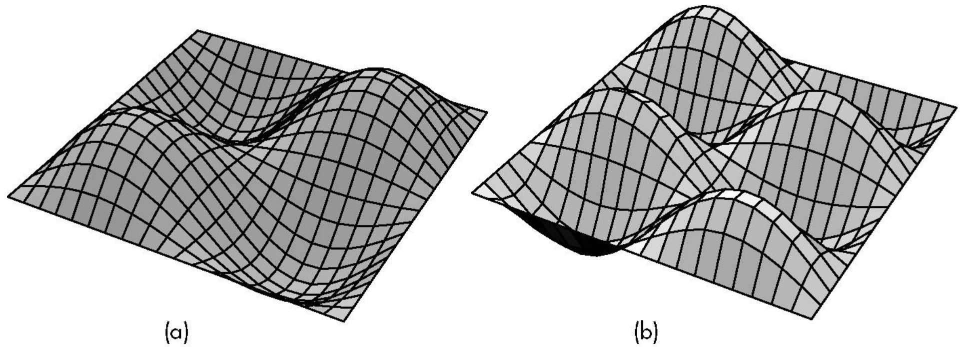
Sampling Theory

(where the name "aliasing" comes from)



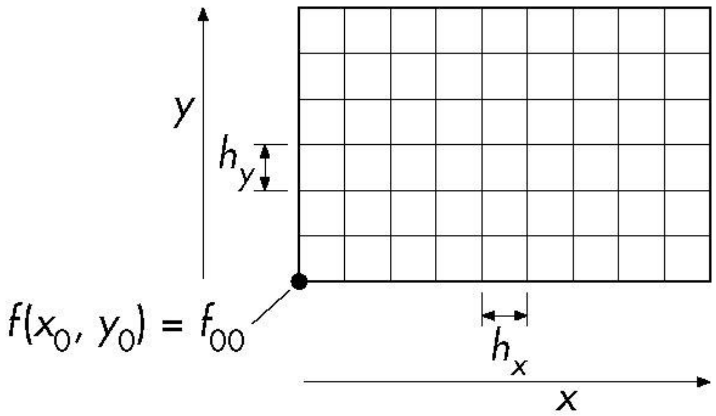
Decomposition of one-dimensional function (Fourier)

Two-dimensional Periodic Functions

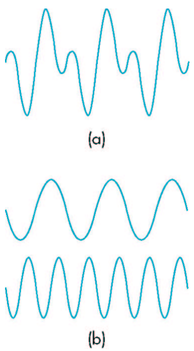


two-dimensional (color) functions

Rasterization: sampling a continuous function



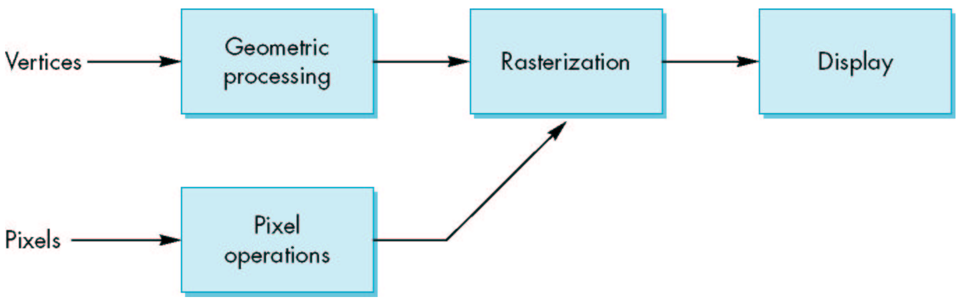
Sampling Theory (2)



Sampling theory: we need a sampling frequency at least twice as high as the highest frequency to be recorded correctly.
Higher frequencies (multiples) are **aliases** (get the same sampling data) of the frequencies thought to be sampled.

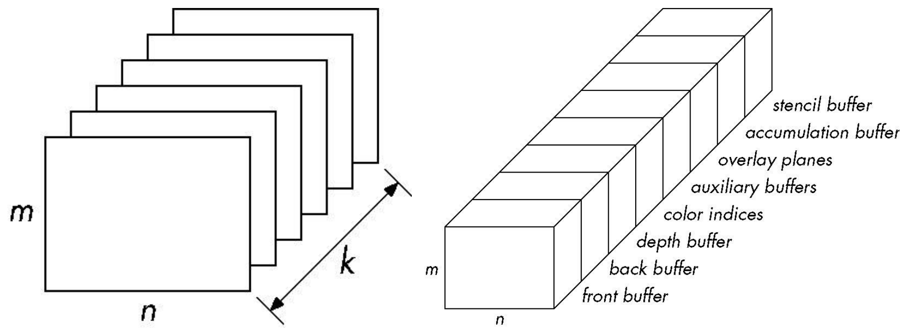
There is always *some* loss of quality in rasterized images.

Processing Pipelines in OpenGL



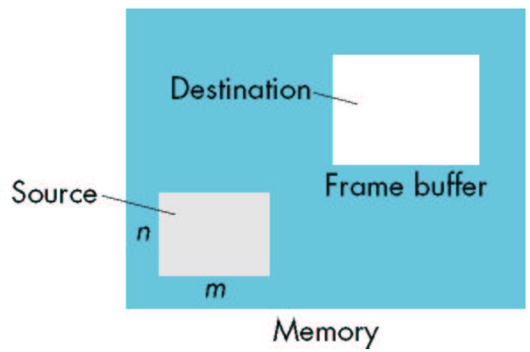
2 independent, parallel pipelines (until the very last step)

Buffers in OpenGL



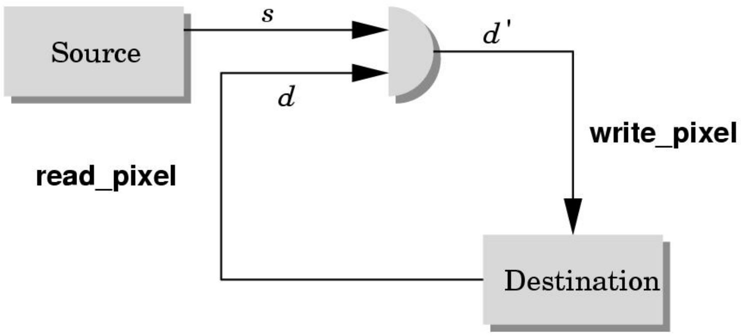
Finite size, depth k (bits per Pixel)
Examples: front buffer, back buffer, depth buffer

Writing into Buffers



block-wise operations, implementation in hardware
bitblt ("bit-block transfer")

Writing Modes



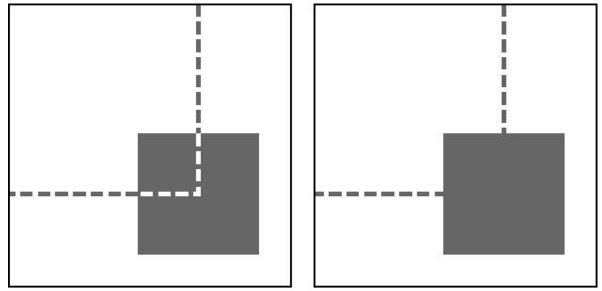
16 possible logical combinations
important: all 0, all 1, copy, OR, XOR

Writing Modes

		writing mode															
s	d	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Use glLogicOp (and glEnable before) to select an operation.

Normal writing modes



Mode 3 Mode 7

$d \leftarrow s$ $d \leftarrow s + d$

(writing the dashed line over the square)

Writing with XOR

$x = (x \oplus y) \oplus y$

exchange with backing store M e.g. containing a menu:
$S \leftarrow S \oplus M$
$M \leftarrow S \oplus M$
$S \leftarrow S \oplus M$
use for menus, cursers, rubberbanding

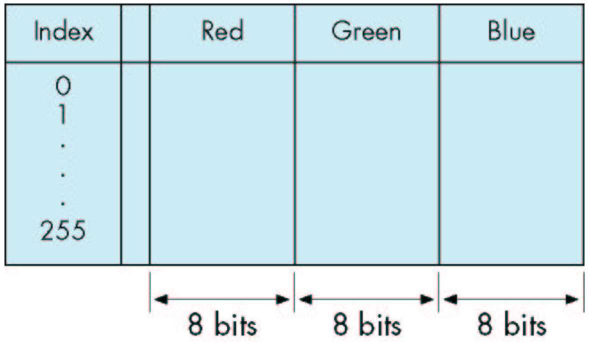
OpenGL Frame Buffers

- Color buffers (classical "frame buffers")
  front and back buffers
- Depth buffer
- Accumulation buffer
  accumulate image from multiple "overlayed" images
- Stencil Buffer
  special masking

Rendering Raster Images

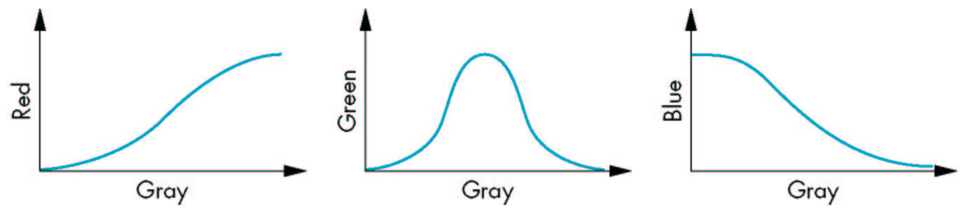
- `glRasterPos3f(x, y, z);`
  raster position, subject to model-view and projection matrices
- `glBitmap(width, height, xo, yo, xi, yi, bitmap);`
  puts bitmap to raster position
  (xo,yo) origin in bitmap, put at raster position
  (xi,yi) increment of raster postion, after rendering
- example: bitmap fonts

Color Lookup Tables



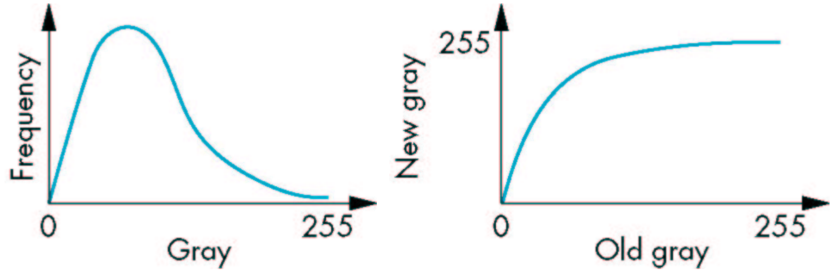
e.g. map grayshade to pseudocolors
used in data visualization, like Mandelbrot set
(set maps with `glPixelMap`)

Thermal Color Map

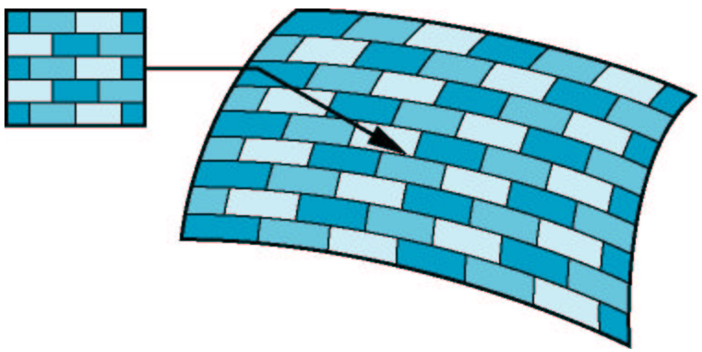


cold = blue, hot = red

Brightening Dark Images



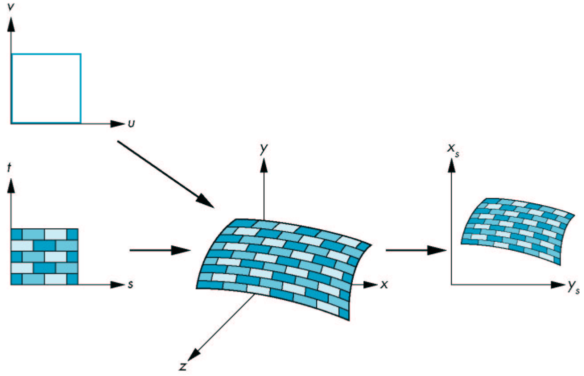
Texture Mapping



Pattern $\longrightarrow$ Surface

Adding lots of realism without complex vertex structures

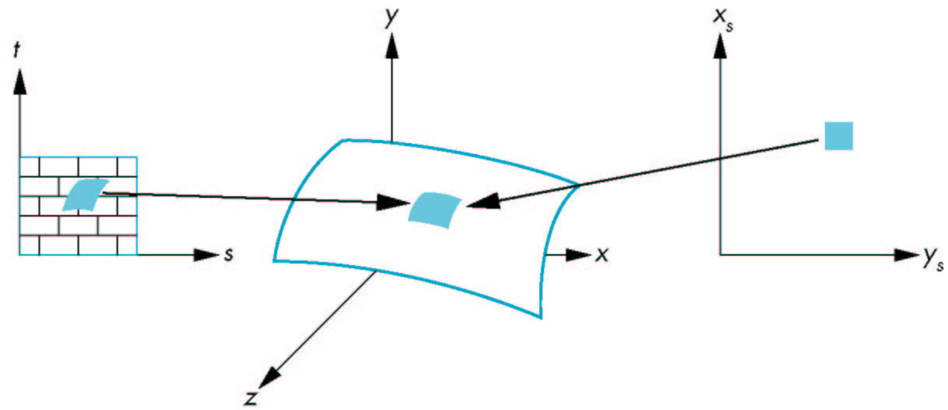
2D Texture Mapping (Parametric Surface)



$T(s,t) \rightarrow S(u,v) \qquad s,t \in [0..1]$

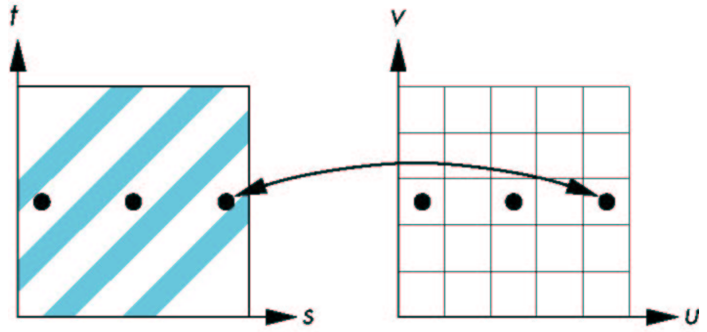
distortion may occur (e.g. rectangle to sphere)

Mapping back (Pixel to Texture Area)



texture area object area screen area

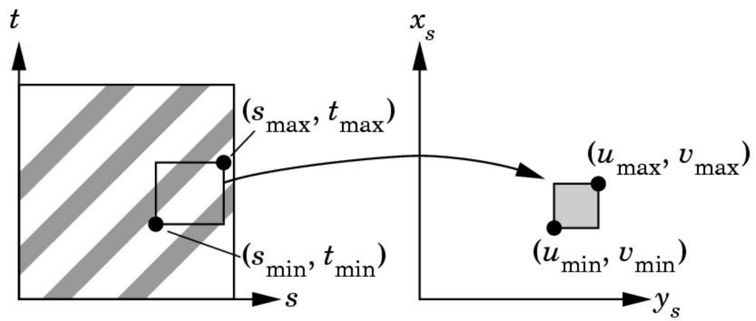
Aliasing in Texture Generation



may always catch white points
averaging doesn't really help always
(here, gives gray rather than stripes)

Linear Texture Mapping

Flat surfaces:

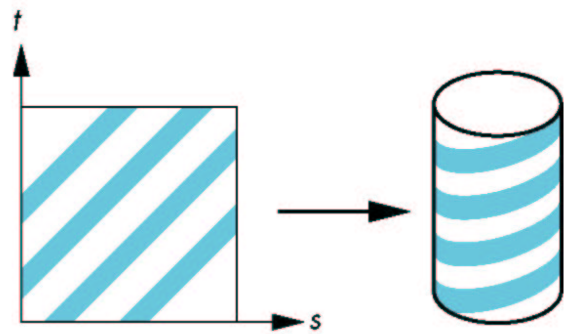


$u = as + bt + c \qquad v = ds + et + f$
(reversible as long as $ae \neq bd$)
Possibly stretches (distorts) texture to fit

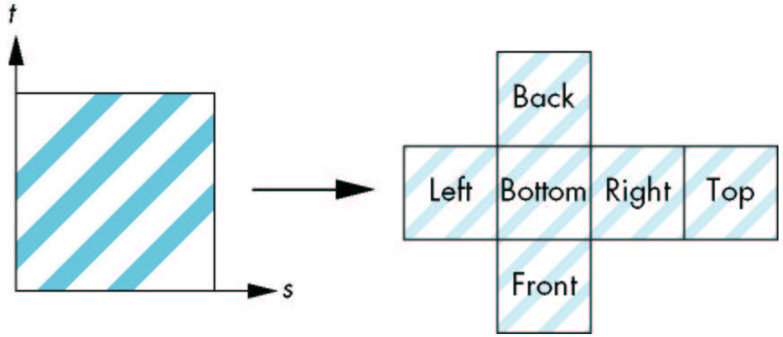
Problem: Mapping to non-flat surfaces

- This mapping can be fairly complex if not impossible
- Idea for solution:
  Map onto a simple 3D-surface and project from there to the object

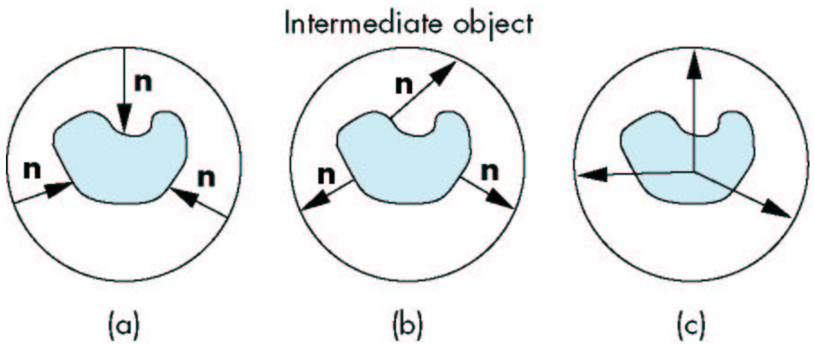
Mapping onto a Cylinder



Mapping onto a Box



Mapping onto the object using Normals



different ways of computing normals give different results

Texture in OpenGL

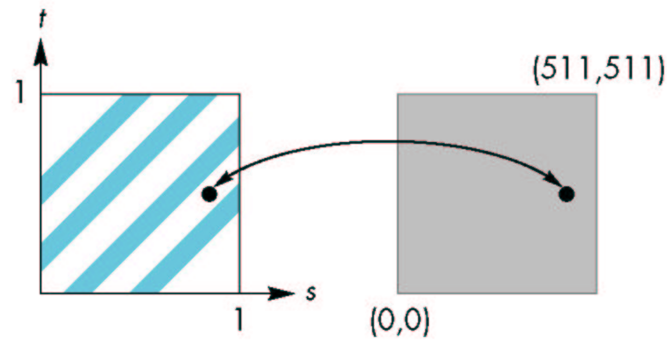
```
GLUbyte my_texels[512][512];    /* texel == texture pixel */

glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0,
             GL_RGB, GL_UNSIGNED_BYTE, my_texels);

glEnable(GL_TEXTURE_2D);

glTexImage2D(GL_TEXTURE_2D, level,  /* see manual */
             components,            /* which of RGBA is affected */
             width, height,
             border,                /* see manual */
             format,
             type, tarray);         /* matching the texture array */
```


Texture in OpenGL (3)

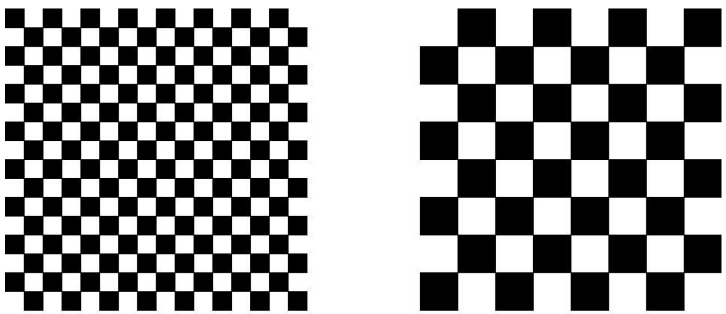
```
glBegin(GL_QUAD);
  glTexCoord2f(0.0,0.0);
  glVertex2f(x1,y1);
  glTexCoord2f(1.0,0.0);
  glVertex2f(x2,y2);
  glTexCoord2f(1.0,1.0);
  glVertex2f(x3,y3);
  glTexCoord2f(0.0,1.0);
  glVertex2f(x4,y4);
glEnd();
```

OpenGL interpolates texture coordinates between vertices.

Mapping to Texture Coordinates



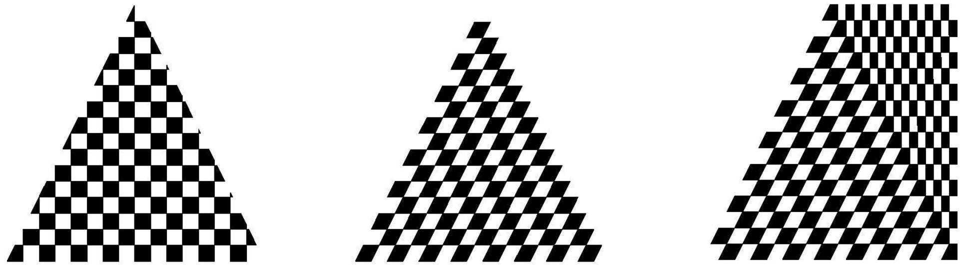
Mapping a Checkerboard Texture



(a) (b)

using whole texel array | using part of texel array

Mapping to Polygons



(a) (b) (c)

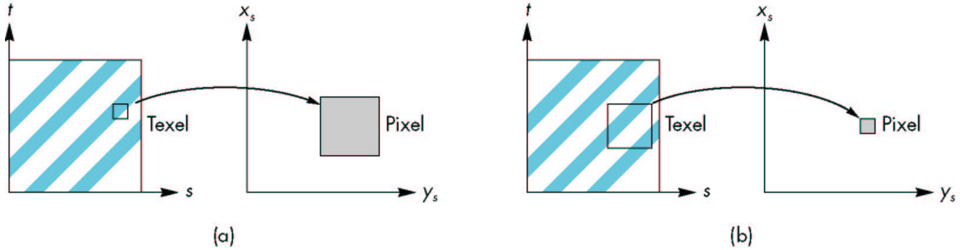
triangle | triangle | trapezoid

Texture Wraparound

What happens if texture coordinate (s or t) is not between $[0 \ldots 1]$:

```
/* whole texture repeated */
glTexParameter(GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameter(GL_TEXTURE_WRAP_T, GL_REPEAT);

/* border value taken */
glTexParameter(GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameter(GL_TEXTURE_WRAP_T, GL_CLAMP);
```

Magnification/Minification




```
/* take nearest texel */
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);

/* take average of closest 2 x 2 pixels */
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
```


Pre-built Low-resolution Texture

```
/* for a 64 x 64 array, this builds:
   32 x 32, 16 x 16, 8 x 8 , 4 x 4, 2 x 2, 1 x 1 */

/* create mipmaps: */
gluBuild2DMipmaps(GL_TEXTURE_2D,3,64,64,GL_RGB,
                  GL_UNSIGNED_BYTE,my_texels);

/* use mipmaps: */
glTexParameter(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
               GL_NEAREST_MIPMAP_NEAREST);
```


Interaction btw. Shading and Texture

- Texture determines object color:
  `GlTexEnv(GL_TEX_ENV,GL_TEX_ENV_MODE,` **GL_DECAL**`);`
- Texture modulates (changes) shading-color (default):
  `GlTexEnv(GL_TEX_ENV,GL_TEX_ENV_MODE,` **GL_MODULATE**`);`
- Perspective correction (if implemented):
  `glHint(GL_PERSPECTIVE_CORRECTION, GL_NICEST);`
  (corrects interpolation of texture coordinates)

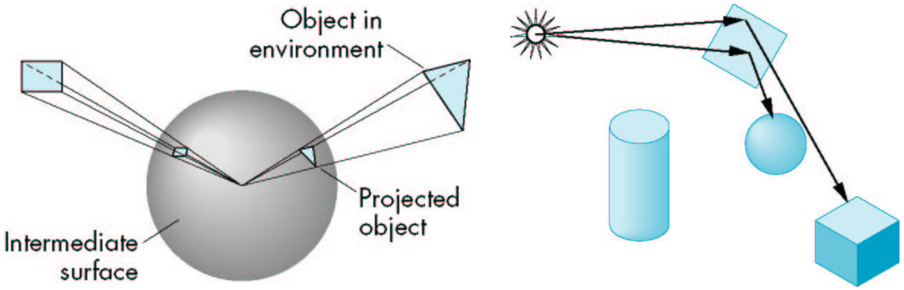
Texture Objects

- like display lists for texture images
- avoids reloading of textures as long as there is sufficient texture memory
- `void glGenTextures( GLsizei n, GLuint *textures )`
  generates texture ids (objects)

Using Texture Objects

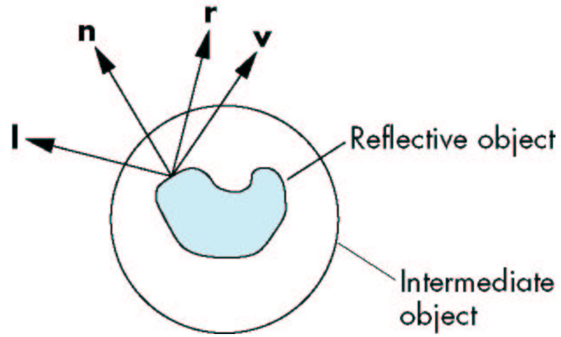
- Create texture object with data and state:
  `glBindTexture( target, id );`
  now do all the setup calls...
- Select texture before using:
  `glBindTexture( target, id );`
  now render vertices...
- target is GL_TEXTURE_2D (or GL_TEXTURE_1D)

Environmental Maps



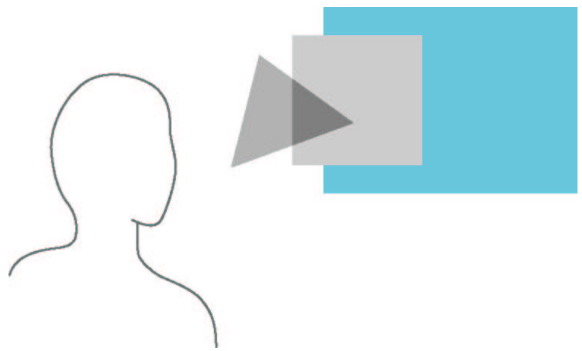
mapping reflections of other objects onto shiny surface
(much simpler than ray tracing)

Mapping from an intermediate object



treating objects like mirrors,
position of viewer gets important

Compositing Techniques



RGBA: A stands for α, the factor of transparency/opacity

$\alpha = 1$, object is opaque (no light gets through)
$\alpha = 0$, object is transparent

Blending

$s = [s_r \quad s_g \quad s_b \quad s_a]$
$d = [d_r \quad d_g \quad d_b \quad d_a]$
$d' = [b_r s_r + c_r d_r \quad b_g s_g + c_g d_g \quad b_b s_b + c_b d_b \quad b_a s_a + c_a d_a]$
b is source blending factor
c is destination blending factor

For blending n objects, either set each α to $1/n$
or use 1 for destination factor and α for source factor

```
glEnable(GL_BLEND);
glBlendFunc(source_factor, destination_factor);

/* GL_ONE, GL_ZERO, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA ... */
```

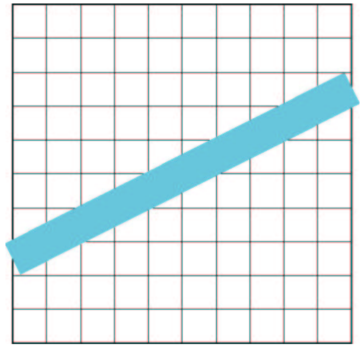

Blending and Rasterization Order

For using blending/compositing, the order in which polygons are rendered has to be carefully controlled.
"What is on top of which other polygon?"

Problem: hidden-surface removal
Polygons behind transparent polygons need to be rendered!

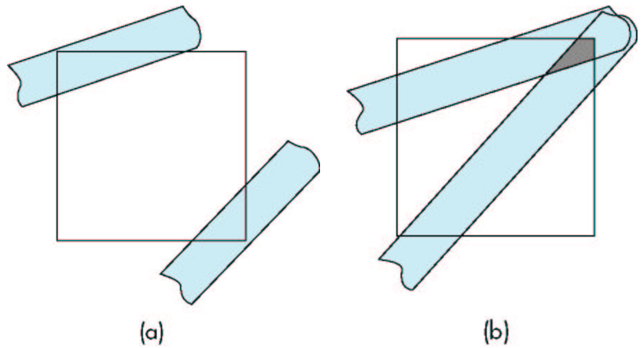
For each transparent polygon set `glDepthMask(GL_FALSE)`

Blending for Antialiasing



rastered lines do not perfectly match screen pixels

Blending Overlapping Polygons



shade pixels by blending the contributions of the individual polygons

Enabling Antialiasing

Enabling smoothing and blending:

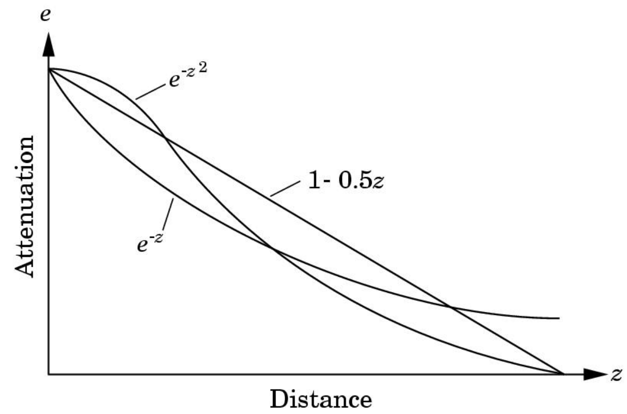
```
glEnable(GL_POINT_SMOOTH);
glEnable(GL_LINE_SMOOTH);
glEnable(GL_SMOOTHING);
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Caution: this can take quite some time!

Depth Cueing and Fog

- Idea (from pre-raster graphics age):
  dim objects that are further away to signal (cue) distance
- Fog idea:
  blend in a distance-dependent color for each object
  make the “air” only partially translucent:
  $C_{s'} = fC_s + (1-f)C_f$
  distance z, fog factor $f(z)$, scene color C_s, fog color C_f
- OpenGL: linear, exponential, Gaussian fog densities

Fog Densities



Enabling Fog

```
GLfloat fcolor[4] = {...};

glEnable(GL_FOG);
glFogf(GL_FOG_MODE, GL_EXP);
glFogf(GL_FOG_DENSITY, 0.5);
glFogfv(GL_FOG_COLOR, fcolor);
```

Use of the Accumulation Buffer

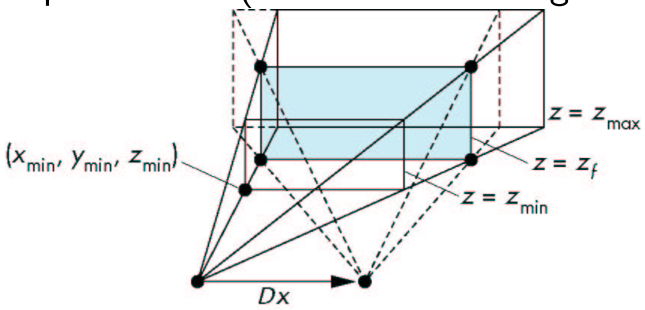
Combine (accumulate) multiple images into one:

```
glClear(GL_ACCUM_BUFFER_BIT);
for ( i=0; i < num_images; i++ ){
  glClear(GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT);
  display_image(i);
  glAccum(GL_ACCUM, 1.0 / (float) num_images);
}
glAccum(GL_RETURN, 1.0);
```


Applications of the Accumulation Buffer

- motion blur:
  let an object move and show its trajectory

- depth of field (simulate focal range of real camera)



Summary

- From Geometric to Discrete Operations
- Buffer Writing
- Texture Mapping
- Environmental Maps
- Compositing (Blending)
- Special Effects (Accumulation Buffer)
- Next week: Implementation of a Renderer