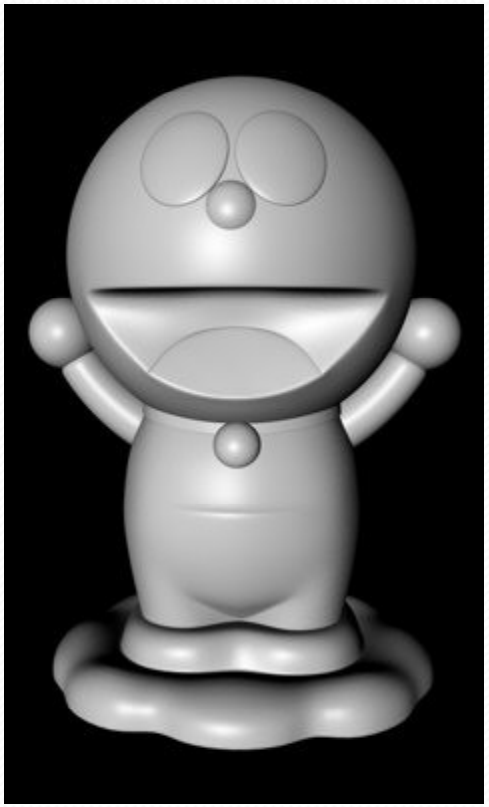
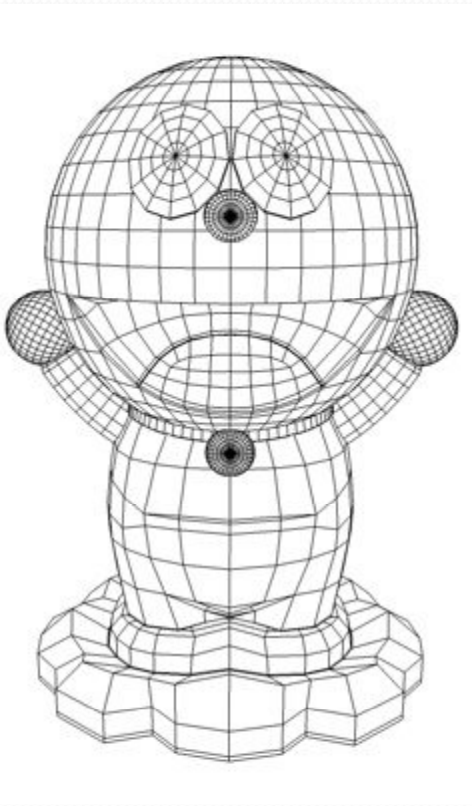
Mapping Techniques

CS3241 Computer Graphics

A ray traced image






389

Mapping Techniques

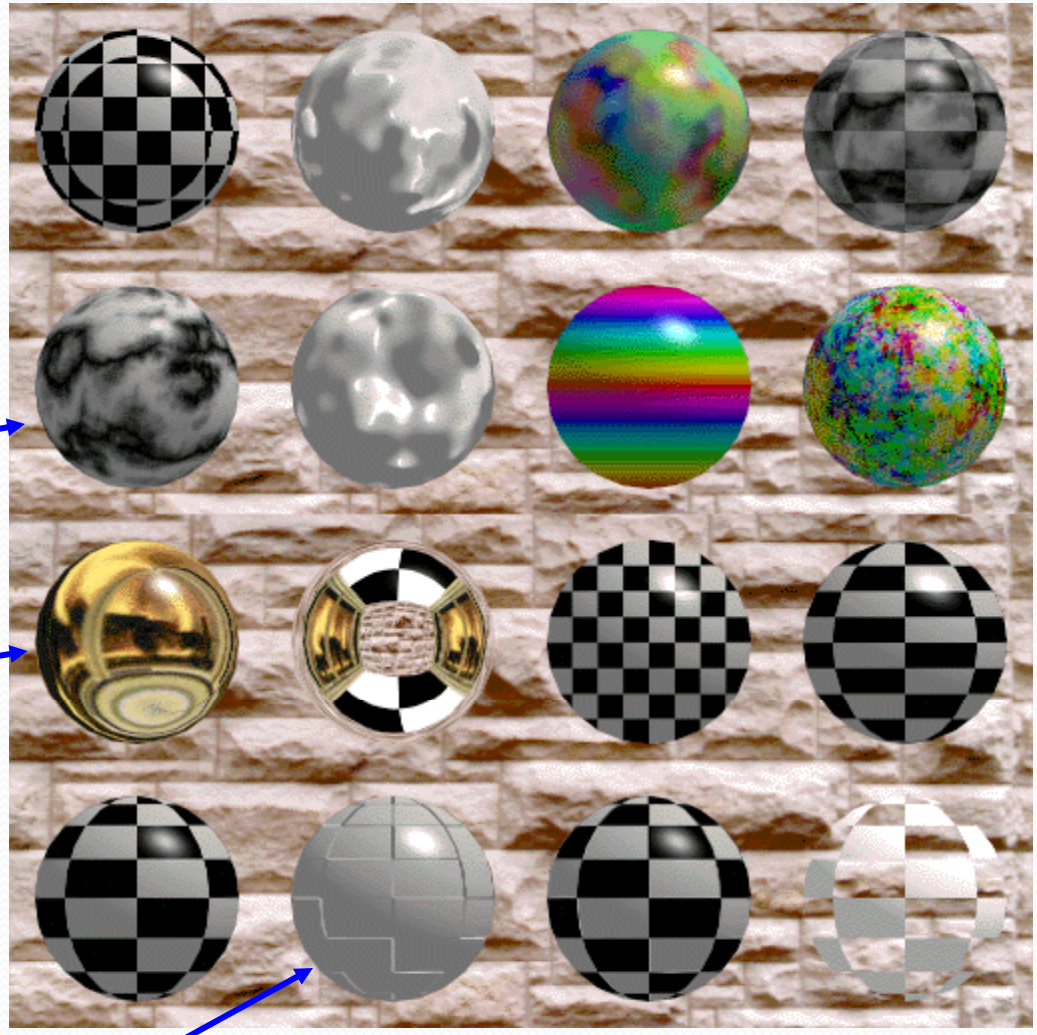
Mapping Techniques

- Instead of modeling as a geometric structure
  - "Stick" a picture onto the surface instead
  - Illusion of a surface of great complexity
    - although the surface might just be a single polygon

- Topics:
  I. Texture mapping
  II. Environmental mapping
  III. Bumping

Examples of Surface Mapping

- All are spheres like Lab 3



Simple texture mapping

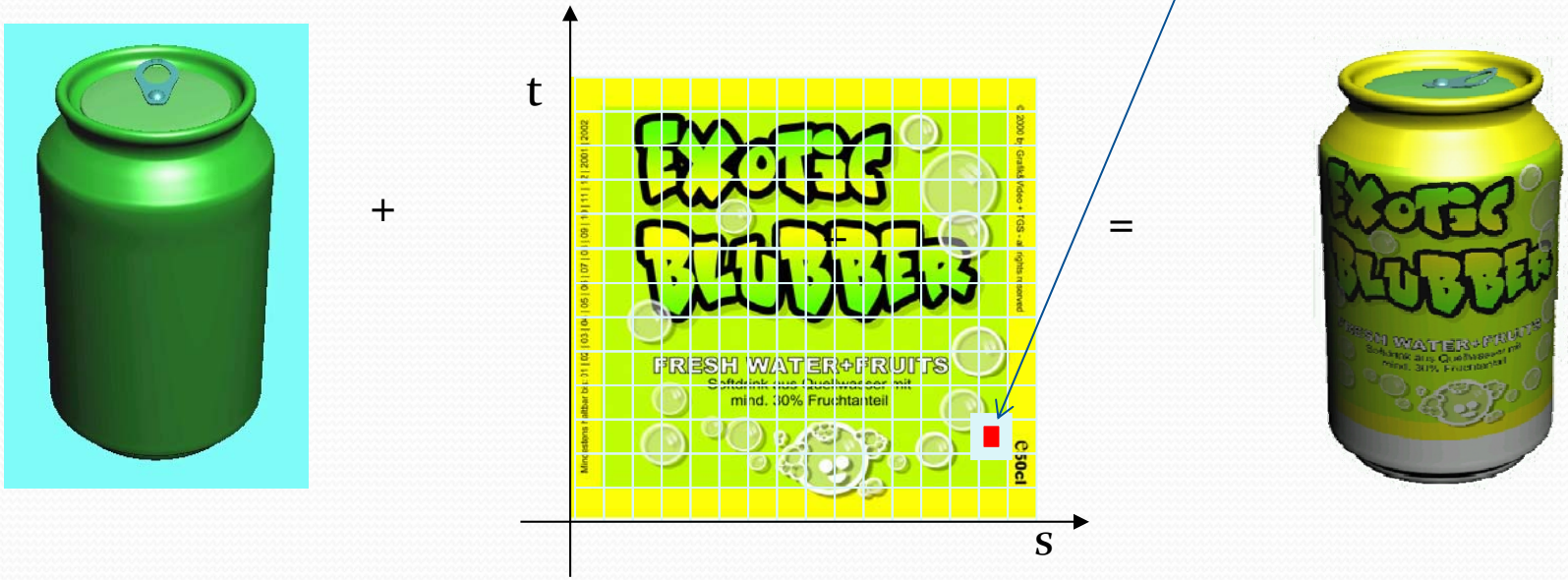
Environment mapping

Bump mapping

Texture Mapping

Textures

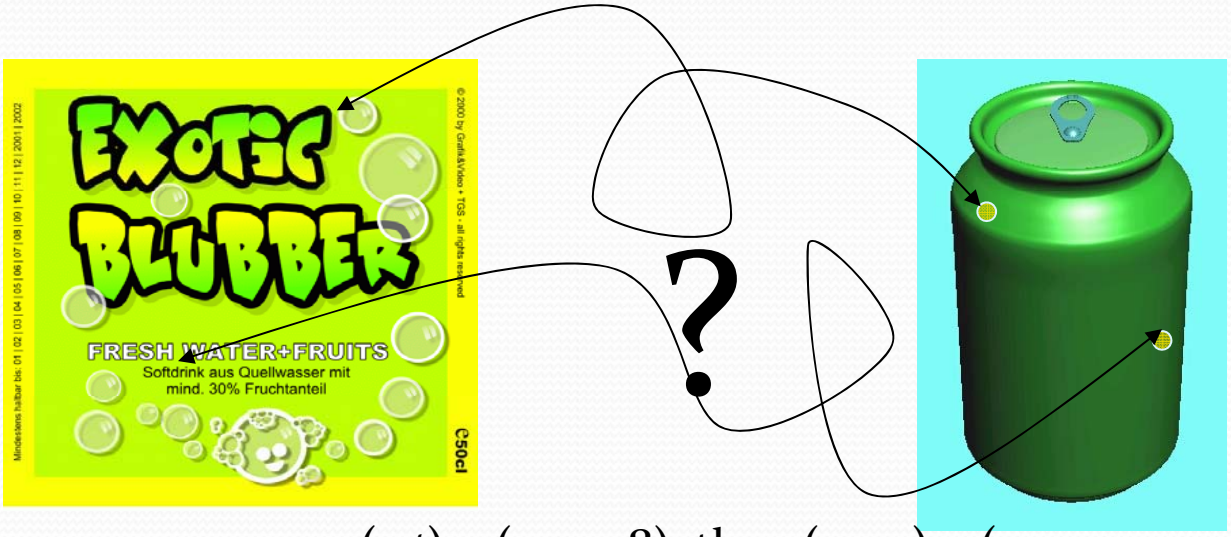
- Texture patterns are applied onto a surface
- Textures are normal images stored in a 2D array
  - Scanned images or regular patterns generated by a program.
  - Each pixel in a texture data are called a texel. And it's color value is denoted as T(s,t)



Texture Mapping

- Texture mapping involves the linking of T(s, t) to a geometric surface definition that is itself mapped to screen coordinates for display.
- First, we define a relationship between a point on a surface/object with a 2D coordinate (s,t). We called this the **surface parameterization**.

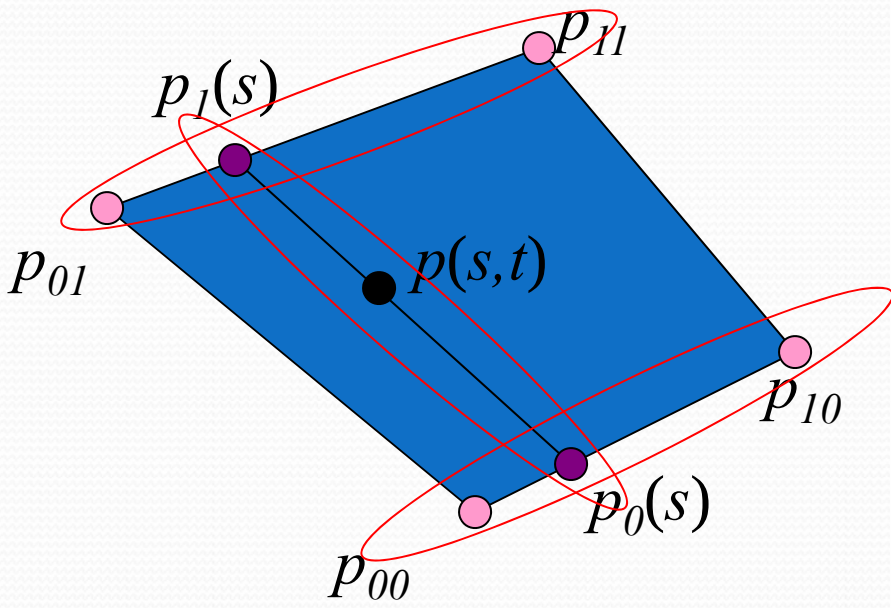
e.g. s = 0.2, t = 0.15, then x=100,y=200,z=100



(s,t) = (0.9,0.8), then (x,y,z) = (200,340,-900)

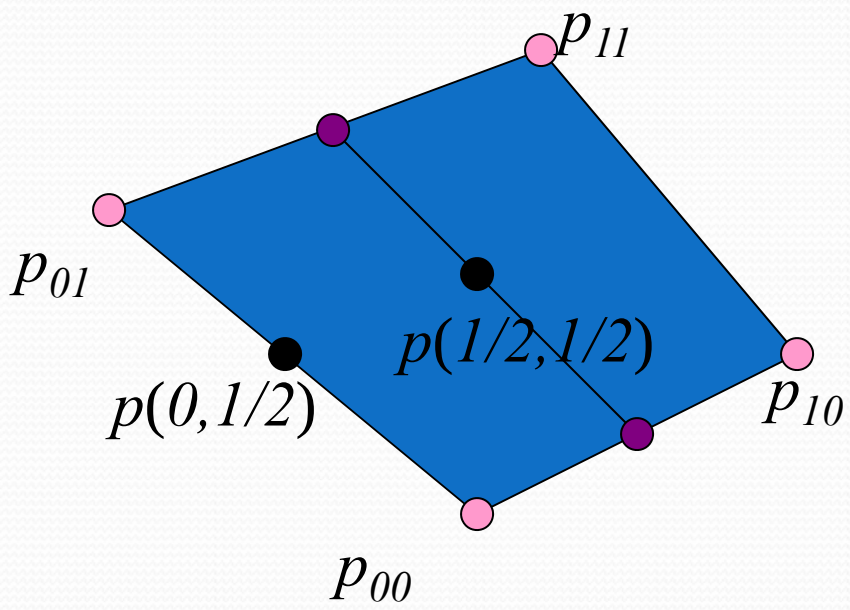
Recap: Parametric Surfaces

- For example, a parametric quadrilateral
- Given four vertices p_{00}, p_{01}, p_{10}, and p_{11}
- Assuming they are coplanar
- $p_0(s) = (1\text{-}s)\, p_{00} + s\, p_{10}$
- $p_1(s) = (1\text{-}s)\, p_{01} + s\, p_{11}$
- $p(s,t) = (1\text{-}t)\, p_0(s) + t\, p_1(s)$



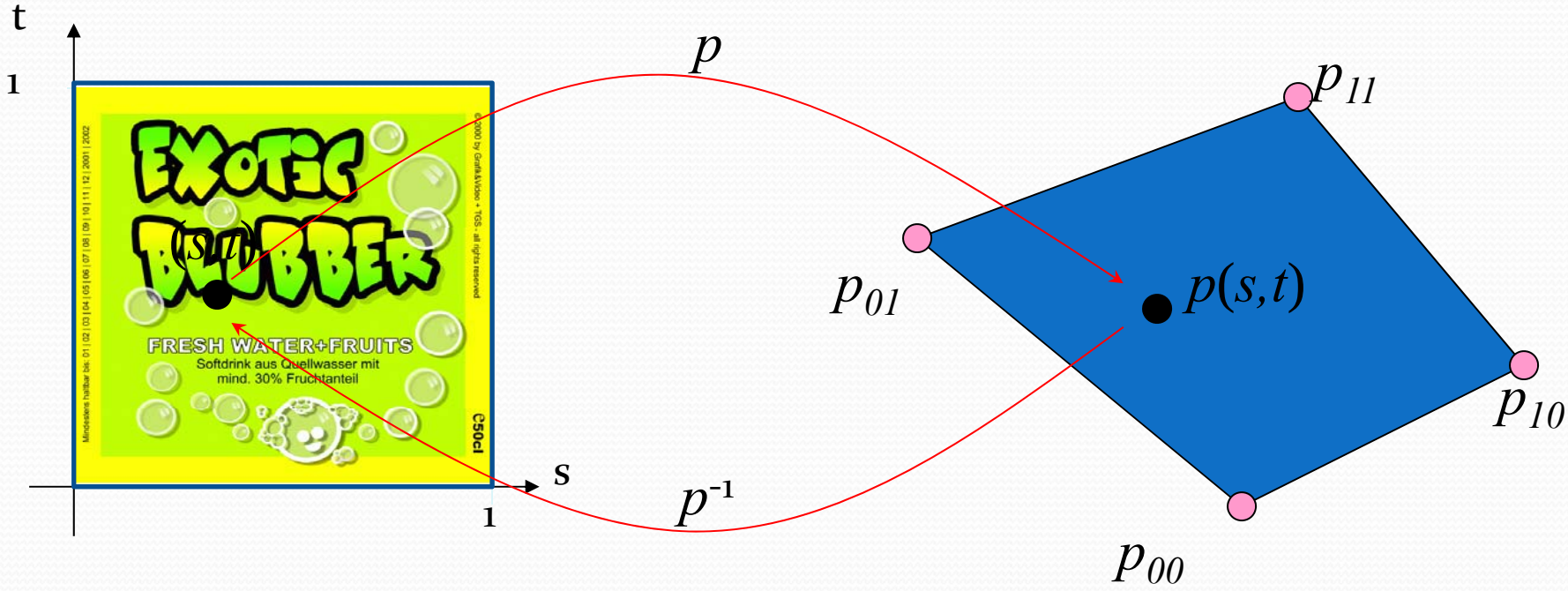
Parametric Surfaces

- For example
  - $p(0,0) = p_{00}$
  - $p(0,1/2) = (p_{00} + p_{01})/2$
  - $p(1/2,1/2)$: mid point of every corner



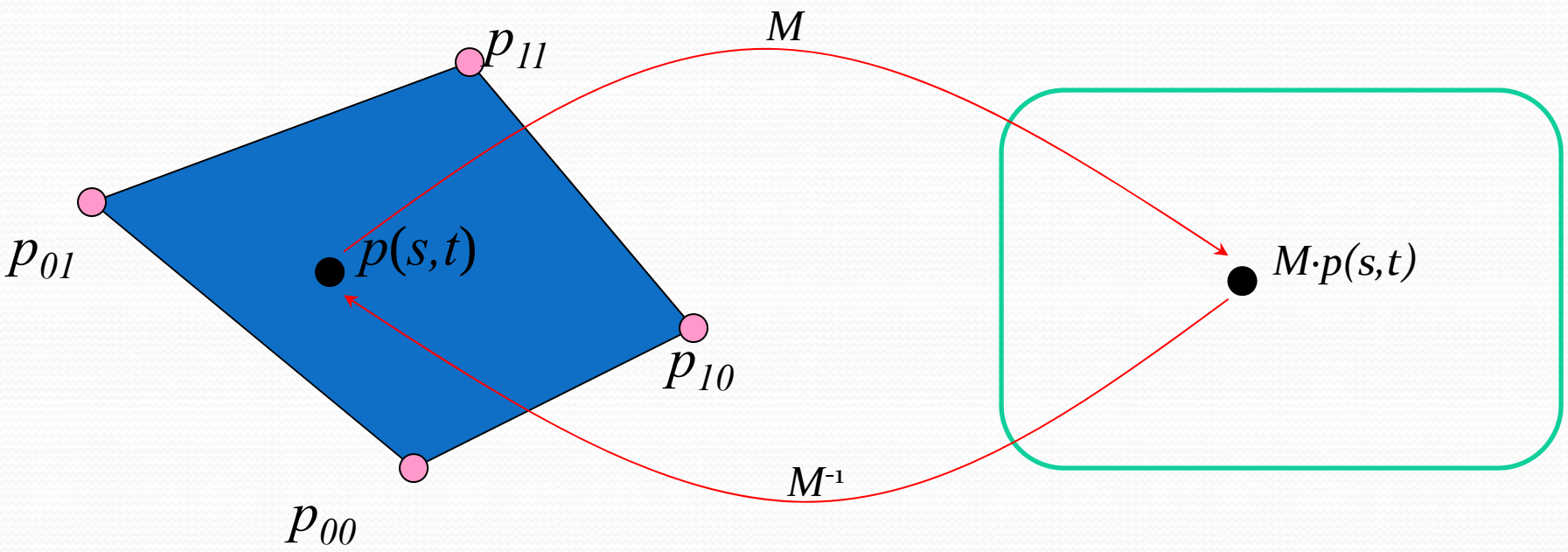
Surface Parameterization

- The function p is a function from (s,t) to the world coordinate (x,y,z)
- The function p^{-1} is a function from the world coordinate (x,y,z) to back to (s,t)



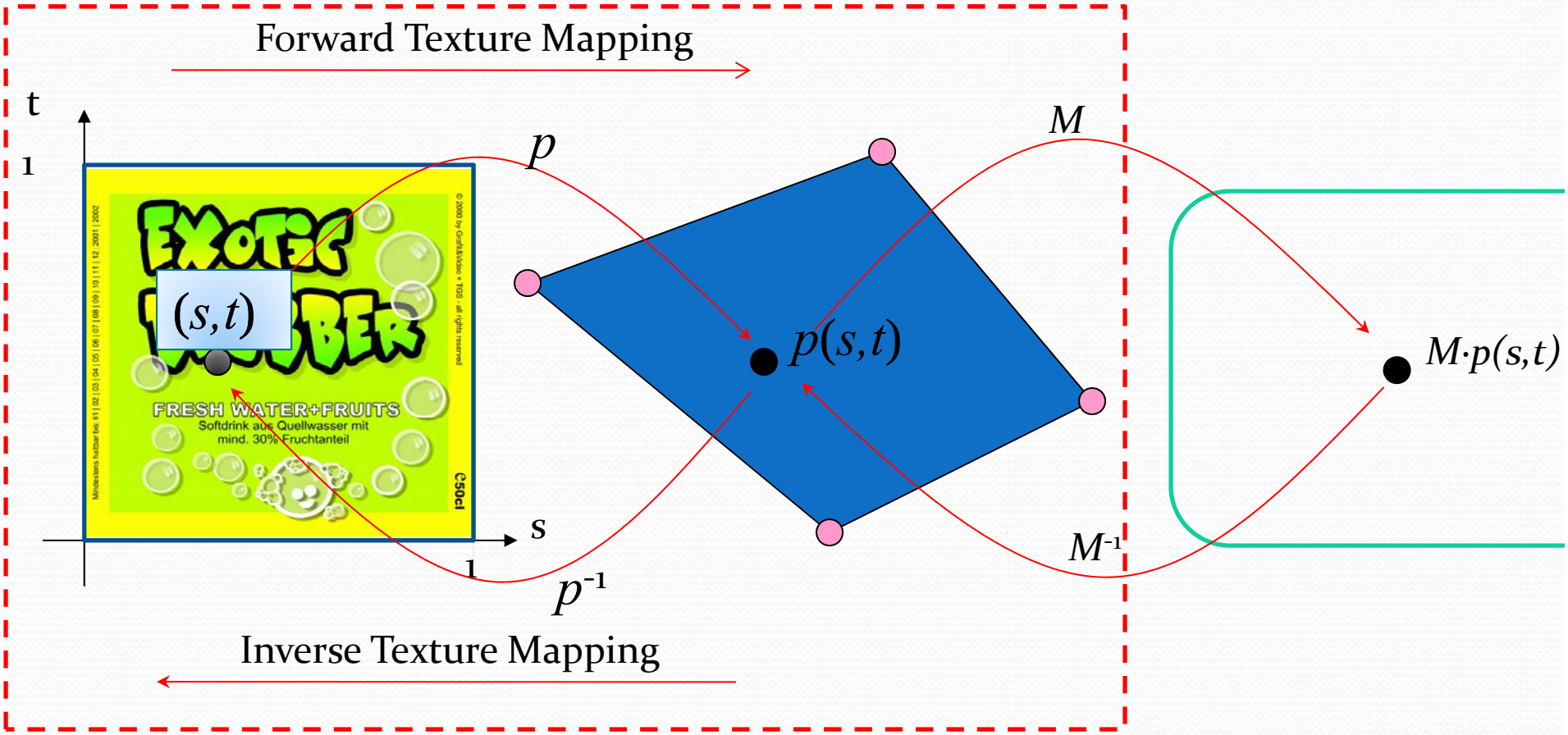
Recap: Perspective Projection

- And the perspective matrix bring the point in the world coordinates to the screen coordinates by multiplying a matrix M
- The inverse function is M^{-1}



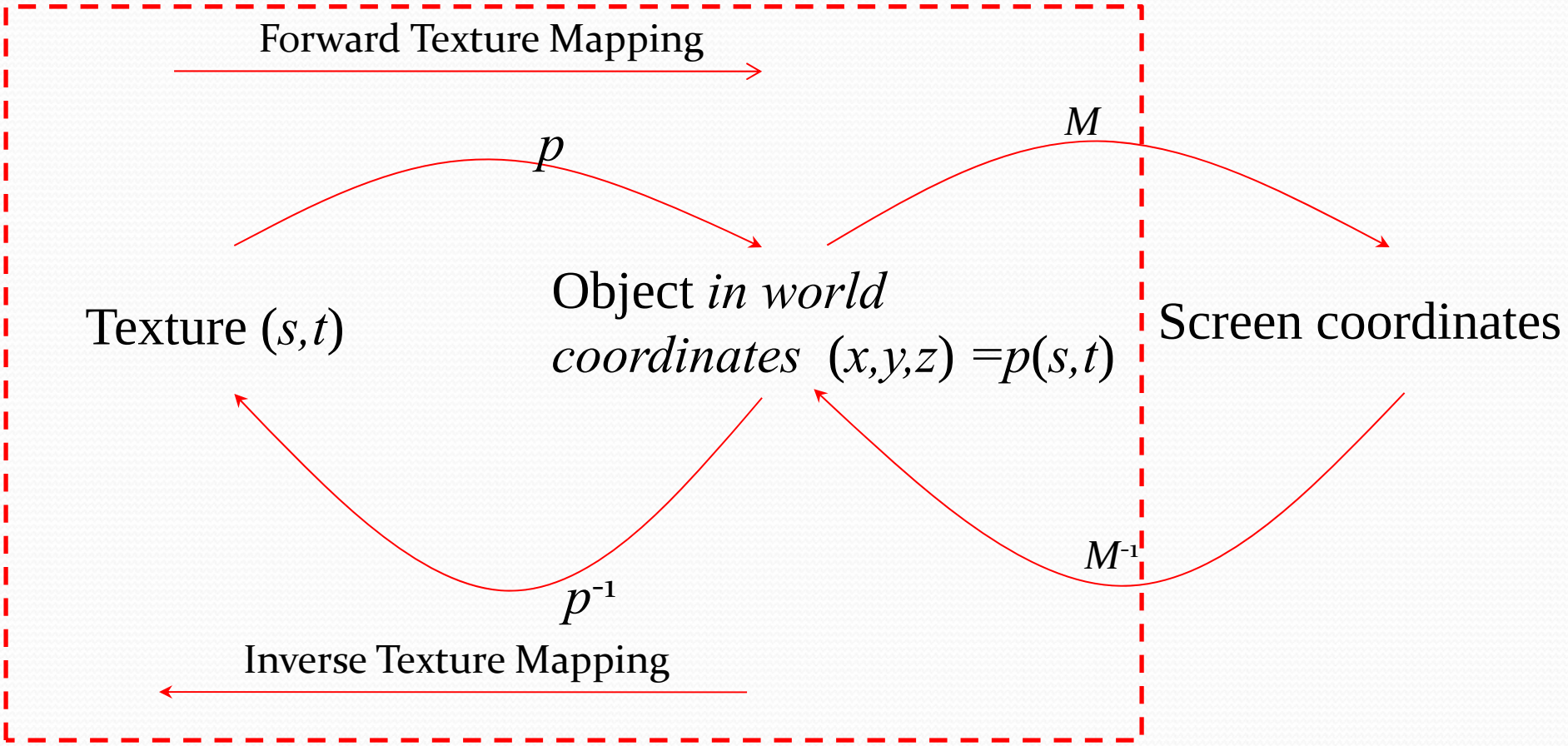
Surface Parameterization

- So there is a parameterization from the (s,t) domain to the screen coordinates



Surface Parameterization

- So there is a parameterization from the (s,t) domain to the screen coordinates



Texture Mapping in OpenGL

Texture Mapping in OpenGL

- Steps in using simple 2D texture mapping in OpenGL :
  1. Specify texture
  2. Draw the scene, supplying both texture and geometric coordinates

Step 1. Specify Texture

- Copy the **texture-image data** from the texture pattern **T**(s, t),
  - e.g. from a .ppm file, to a 512 x 512 array `my_texels`.

```
Glubyte  my_texels [ImageWidth][ImageHeight][3];
```

- Specify that this array is to be used as a 2D texture (usually as part of program initialization by

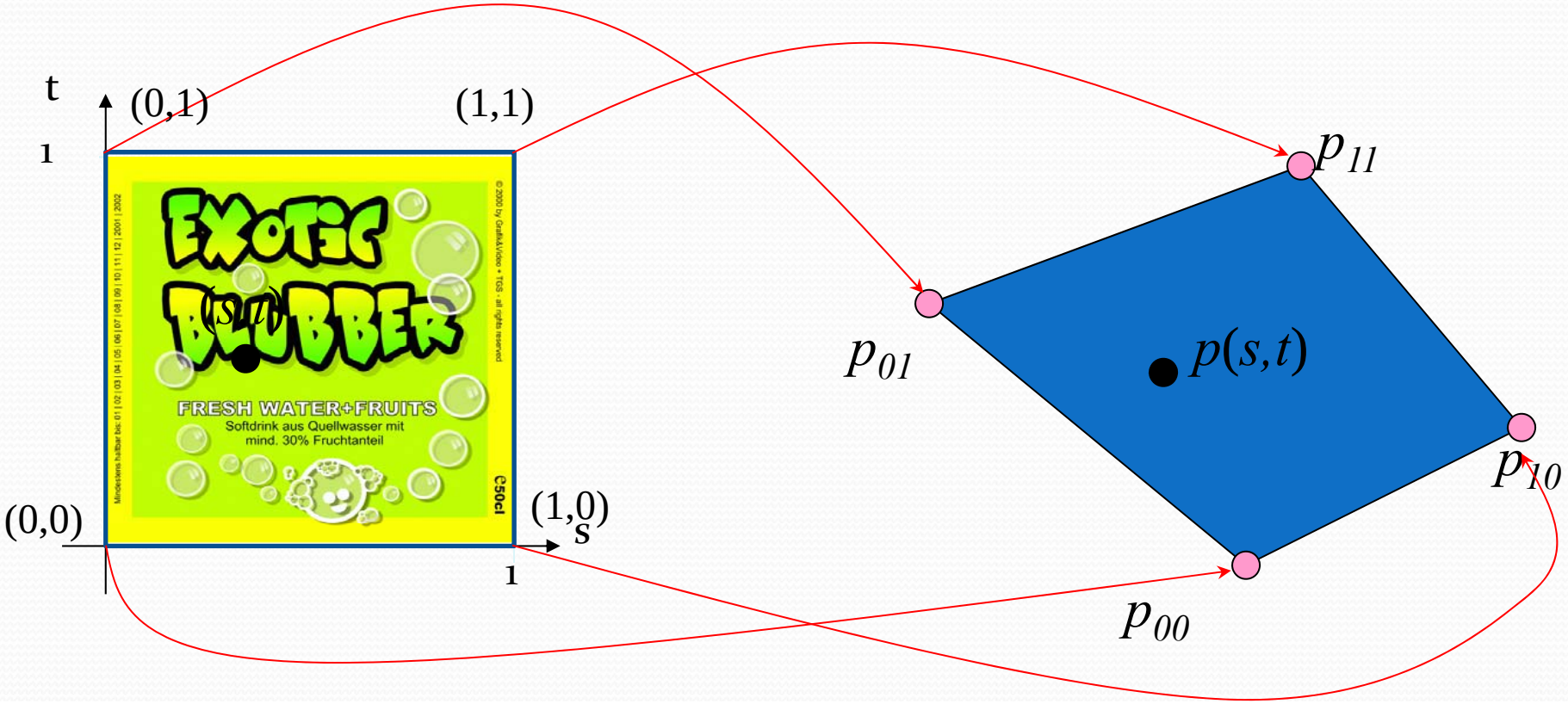
```
glTexImage2D (GL_TEXTURE_2D, level, components,
              width, height, border,
              format, type, tarray);
```

```
e.g.   glTexImage2D ( GL_TEXTURE_2D, 0, 3,
                      512, 512, 0,
                      GL_RGB, GL_UNSIGNED_BYTE, my_texels );
```

Width/height must be 2^n+2x(border)

Mapping the Vertices

- The Texture Coordinates that maps to the vertices
  - Associate (0,0) to p_{00} , (1,0) to p_{10} , etc....



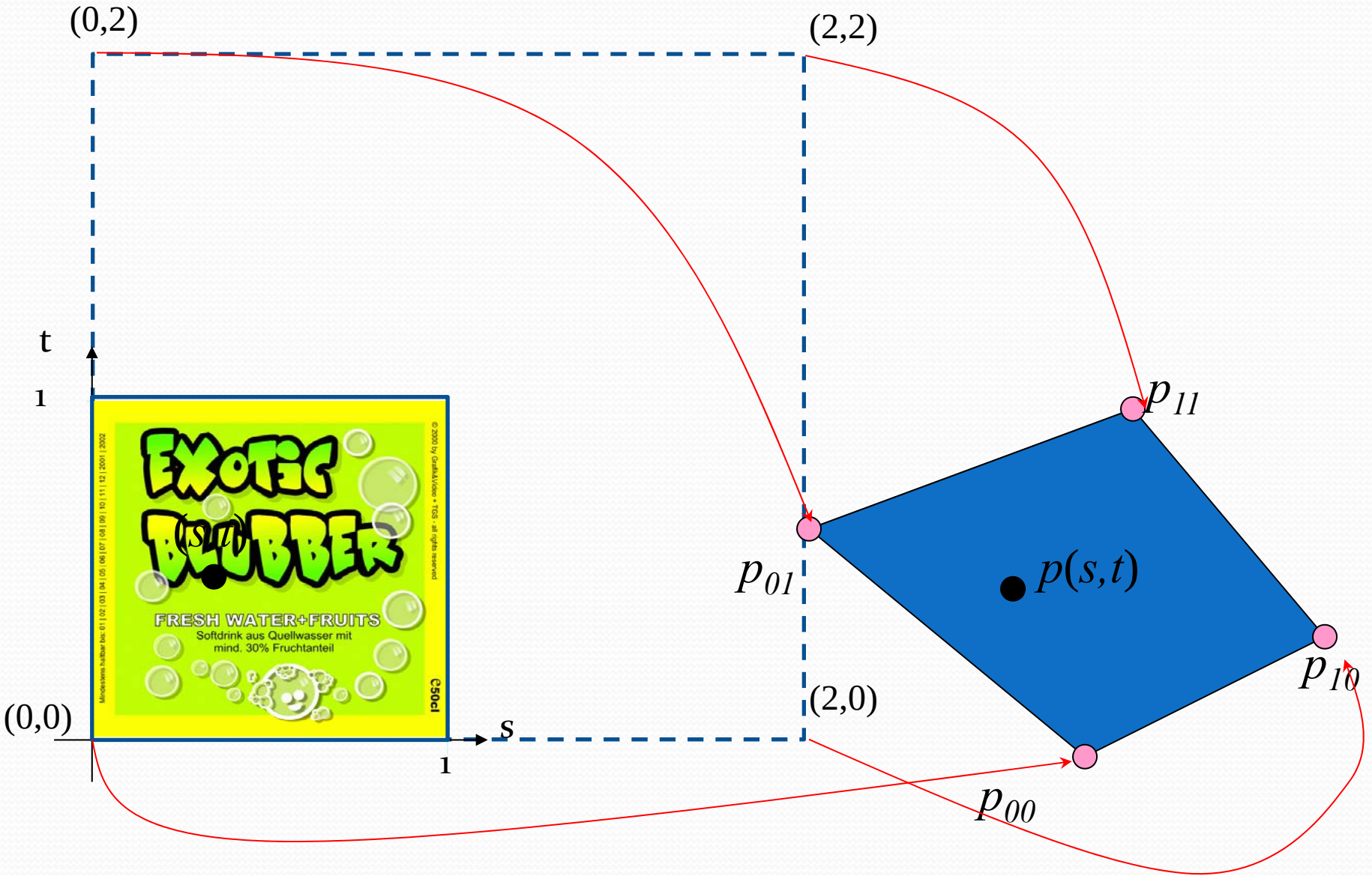
Step 2. Draw the scene

- To draw a texture mapped quadrilateral, both the object coordinates and the texture coordinates must be provided side-by-side using code such as:

```
glEnable(GL_TEXTURE_2D )
glBegin ( GL_POLYGON );
  glTexCoord2f( 0.0, 0.0 );
  glVertex2f( x1, y1, z1 );
  glTexCoord2f( 1.0, 0.0 );
  glVertex2f( x2, y2, z2 );
  glTexCoord2f( 1.0, 1.0 );
  glVertex2f( x3, y3, z3 );
  glTexCoord2f( 0.0, 1.0 );
  glVertex2f( x4, y4, z4 );
glEnd ();
```

What if?

```
glBegin ( GL_POLYGON );
  glTexCoord2f( 0.0, 0.0 );
  glVertex2f( x1, y1, z1 );
  glTexCoord2f( 2.0, 0.0 );
  glVertex2f( x2, y2, z2 );
  glTexCoord2f( 2.0, 2.0 );
  glVertex2f( x3, y3, z3 );
  glTexCoord2f( 0.0, 2.0 );
  glVertex2f( x4, y4, z4 );
glEnd ();
```


(0,2)
(2,2)
t
1
(s,t)
EXOTIC
BUBBER
FRESH WATER+FRUITS
Softdrink aus Quellwasser mit
mind. 30% Fruchtanteil
50cl
(0,0)
s
1
(2,0)
p_{11}
p_{01}
$p(s,t)$
p_{10}
p_{00}

Between Step 1 and 2

- Use `glTexParameter*( )` to specify how the texture is to be wrapped how the colors of pixels are to be filtered.
- Texture pattern defined in `glTexImage2D` normally covers a unit square in (*s* ,*t*) space. With larger range, e.g. [0,3] in both *s*, *t* direction, a number of OpenGL parameters can be used to control the mapping in more detail.
  - Repeat vs clamping

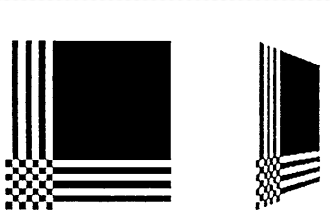


Figure 9-7 Clamping a Texture

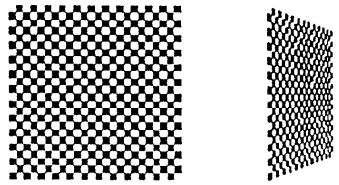
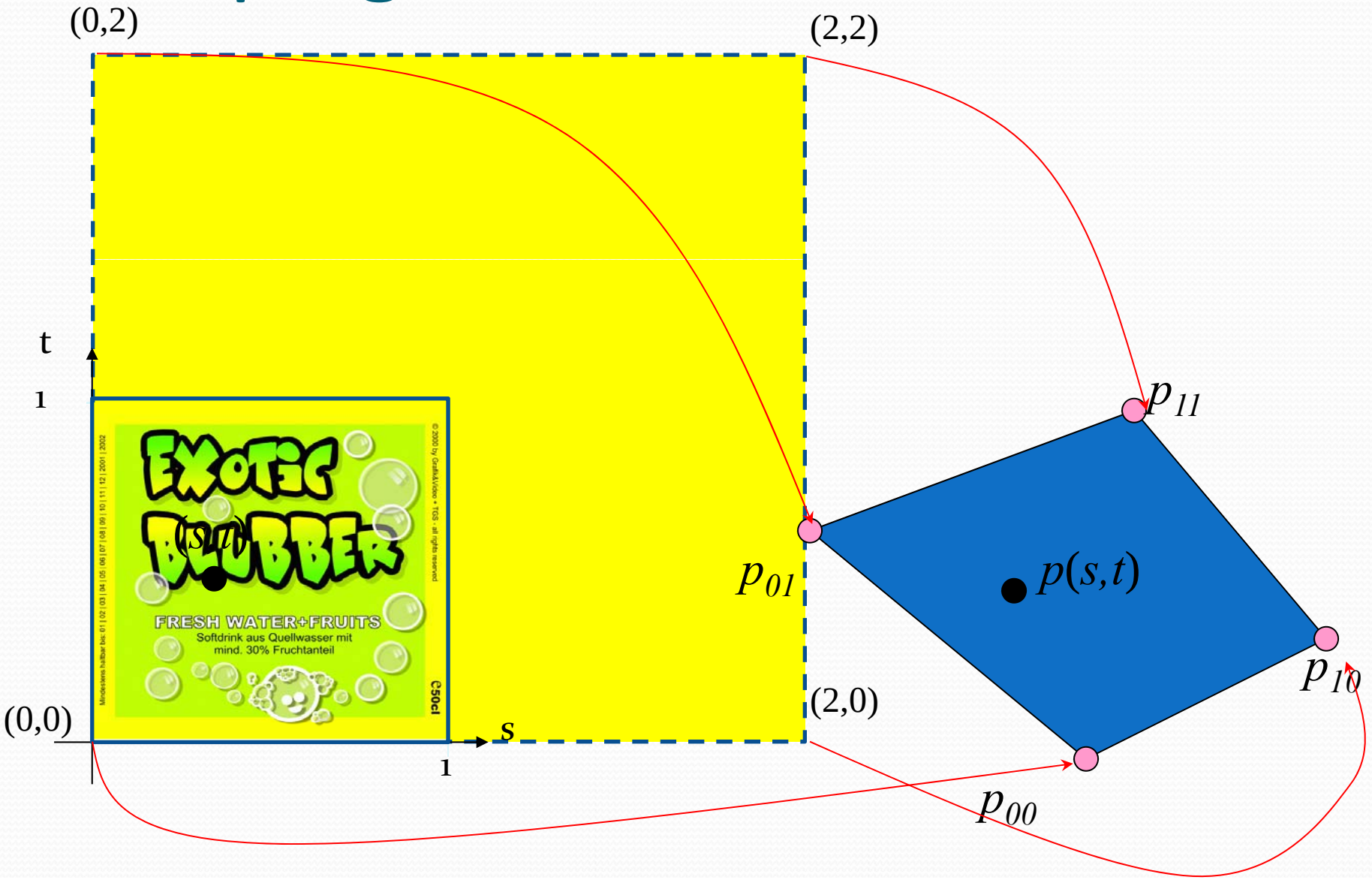


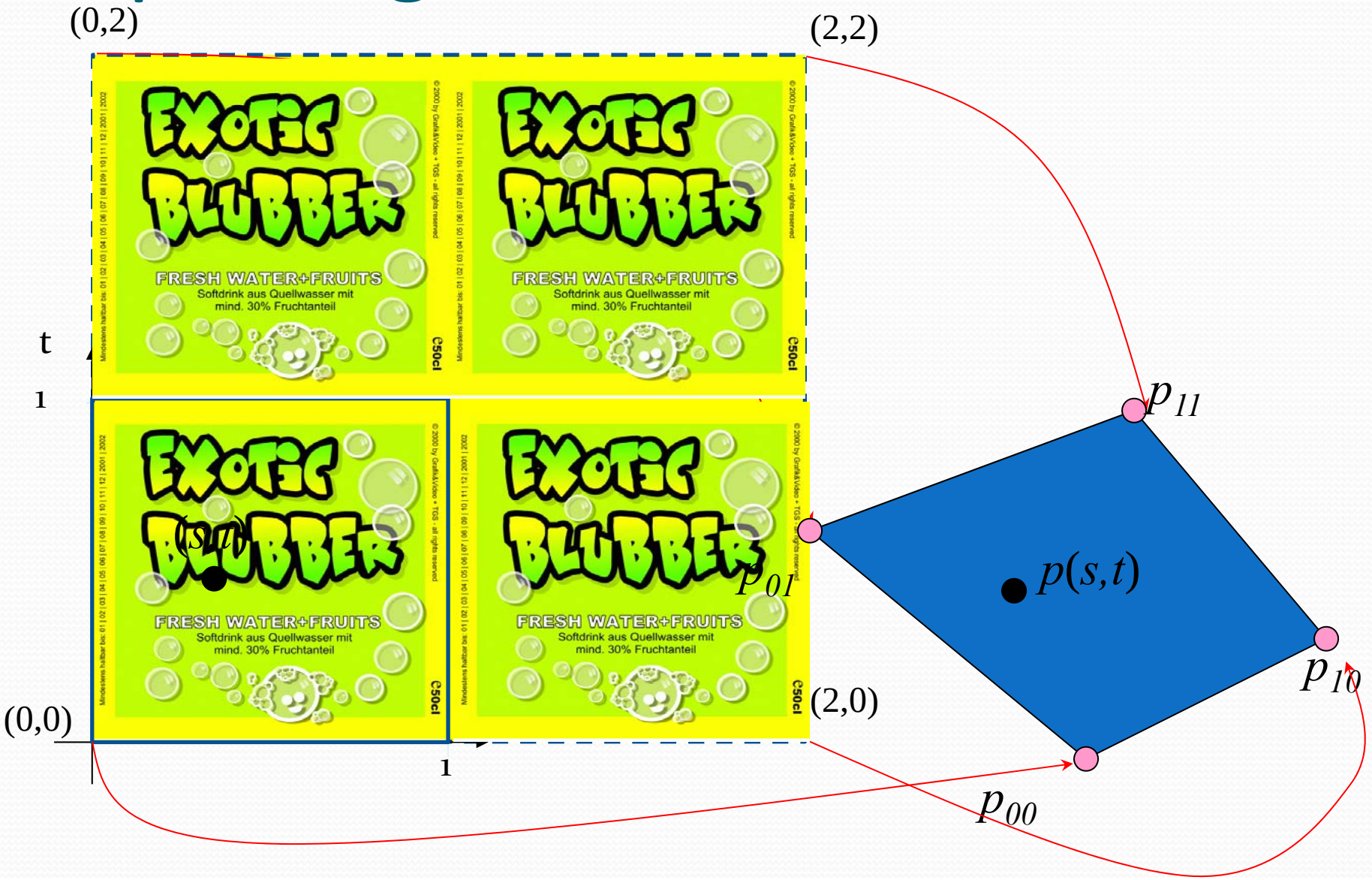
Figure 9-6 Repeating a Texture

Figure 9-8 Repeating and Clamping a Texture

Clamping


(0,2)
(2,2)
t
1
(s,t)
EXOTIC
BUBBER
FRESH WATER+FRUITS
Softdrink aus Quellwasser mit
mind. 30% Fruchtanteil
50cl
(0,0)
s
1
(2,0)
p_{01}
p_{11}
$p(s,t)$
p_{10}
p_{00}

Repeating


(0,2)
(2,2)
t
1
(s,t)
p_{01}
p_{11}
$p(s,t)$
p_{10}
(0,0)
(2,0)
1
p_{00}
EXOTIC BLUBBER
FRESH WATER+FRUITS
Softdrink aus Quellwasser mit
mind. 30% Fruchtanteil
0,50cl

Between Step 1 and 2

- Recall: OpenGL maintains 3 matrix stacks:
  - ModelView, Projection and Texture.
- Normally the texture matrix stack is not modified. In case there is a need to change the size or orientation of the texture pattern, use, e.g.

```
glMatrixMode (GL_TEXTURE);  // enter texture matrix
    mode
glRotated ( …. );
// other matrix manipulations  …..
glMatrixMode (GL_MODELVIEW);  // remember to set it
    back     // to default matrix mode
```

More Complicated Surface

- e.g. a cylindrical surface with radius r, height h

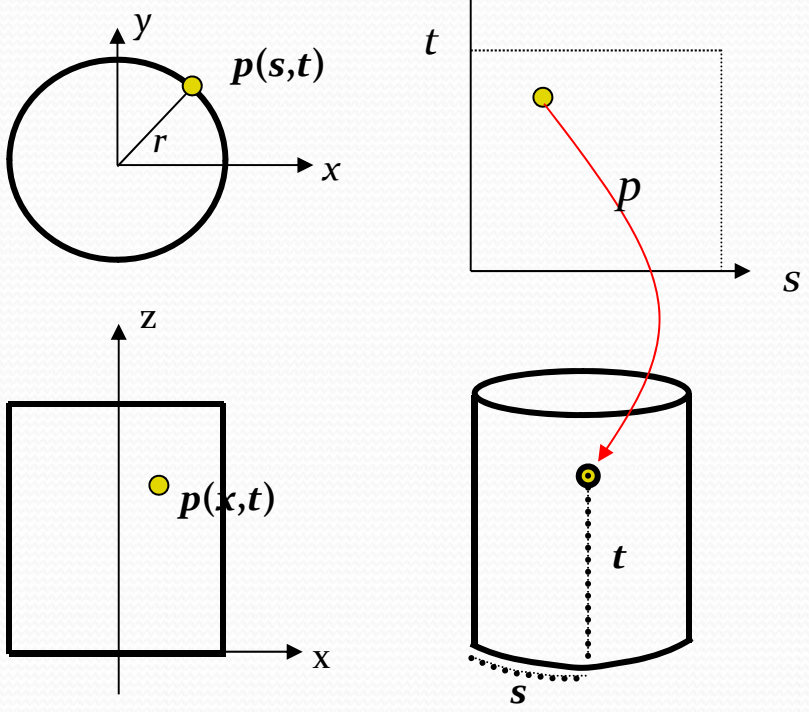
$$x(s,t) = r \cos 2\pi s ,$$

$$y(s,t) = r \sin 2\pi s ,$$

$$z(s,t) = t\, h .$$

- For $0 < s , t < 1$
  - $p(s,t) = (x(s,t), y(s,t), z(s,t))$
- So, a point (x,y,z) on the surface is a function of (s,t), namely:

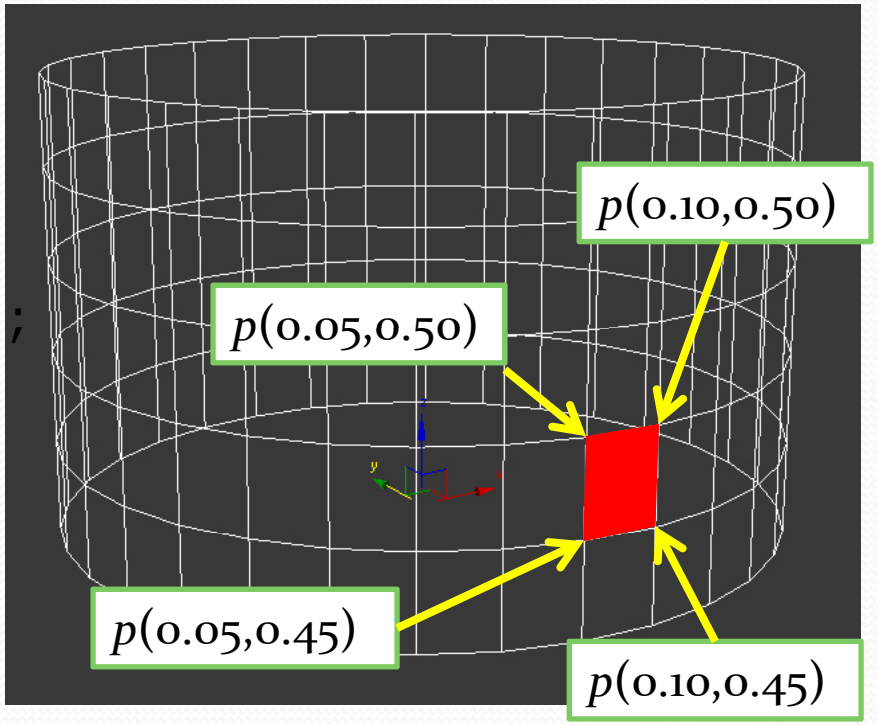
$$(x,y,z) = p(s,t)$$



Drawing the Cylinder

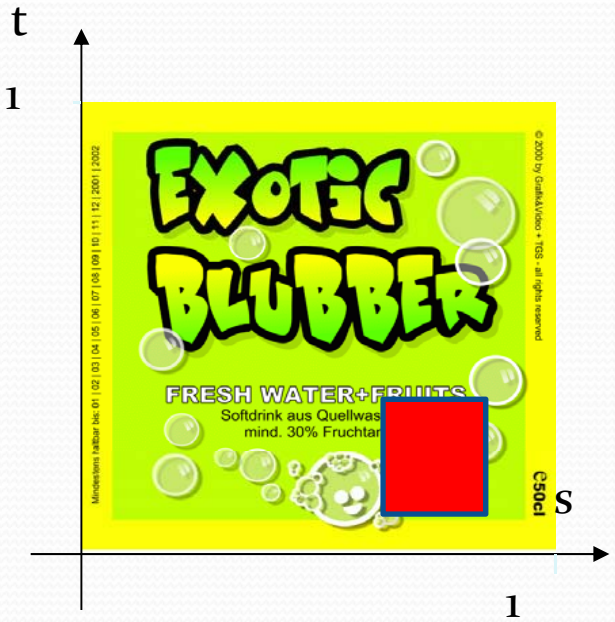
- Decomposing the cylinder into small rectangles (or polygons)

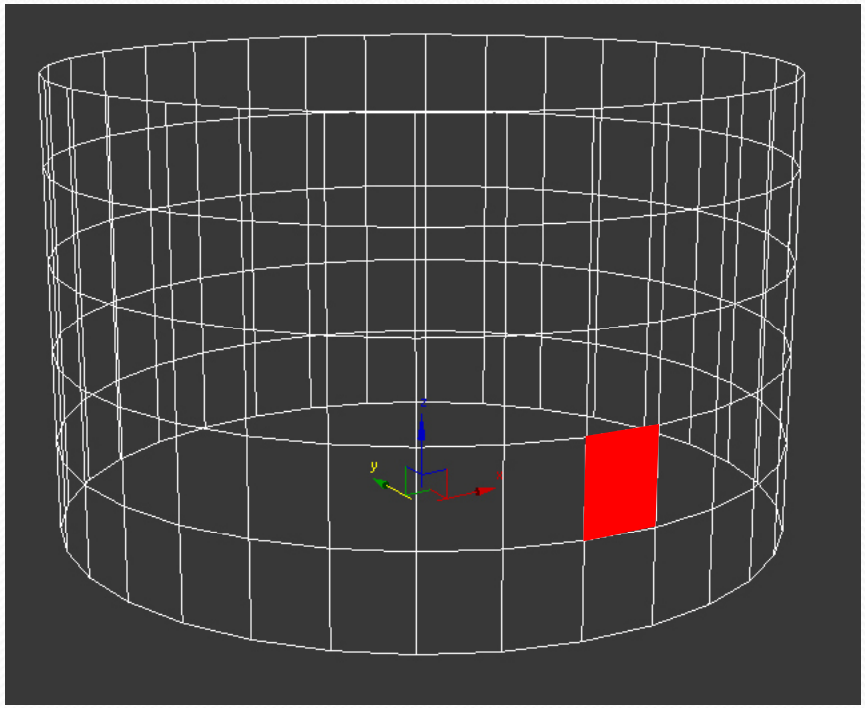
```
for (t=0.0;t<1.0;t+=0.05)
  for(s=0.0;s<1.0;s+=0.05)
  {
    glBegin(GL_POLYGON);
      glVertex3fv(p(s,t));
      glVertex3fv(p(s+0.05,t));
      glVertex3fv(p(s+0.05,t+0.05));
      glVertex3fv(p(s,t+0.05));
    glEnd();
  }
```



Drawing the Cylinder

- Decomposing the cylinder into small rectangles (or polygons)
- For each rectangle, map the four corners to some texture coordinates

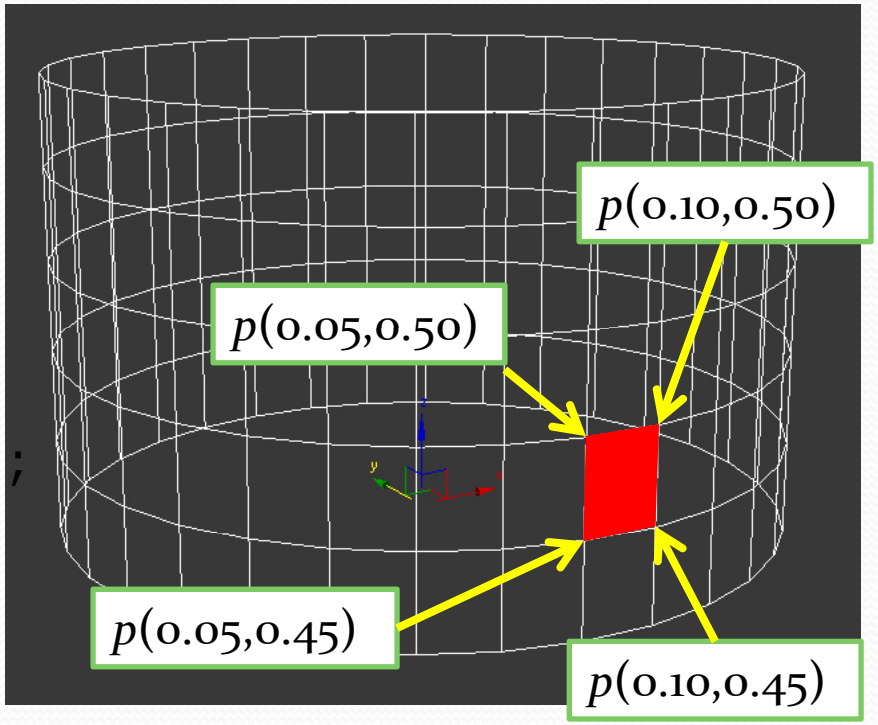




Drawing the Cylinder

- For each rectangle, map the four corners to some texture coordinates

```
for (t=0.0;t<1.0;t+=0.05)
  for(s=0.0;s<1.0;s+=0.05)
  {
    glBegin(GL_POLYGON);
      glTexCoord2f(s,t);
      glVertex3fv(p(s,t));
      glTexCoord2f(s+0.05,t);
      glVertex3fv(p(s+0.05,t));
      glTexCoord2f(s+0.05,t+0.05);
      glVertex3fv(p(s+0.05,t+0.05));
      glTexCoord2f(s,t+0.05);
      glVertex3fv(p(s,t+0.05));
    glEnd();
  }
```

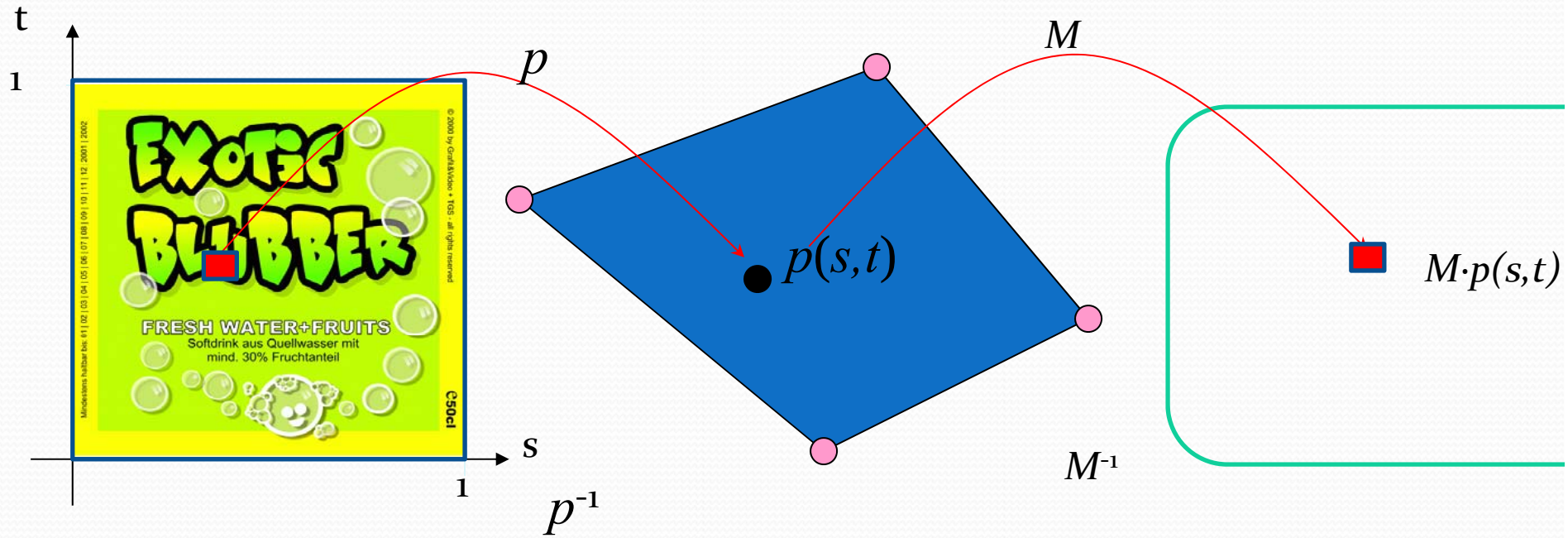


How does it work?

Inverse Texture Mapping

Surface Parameterization

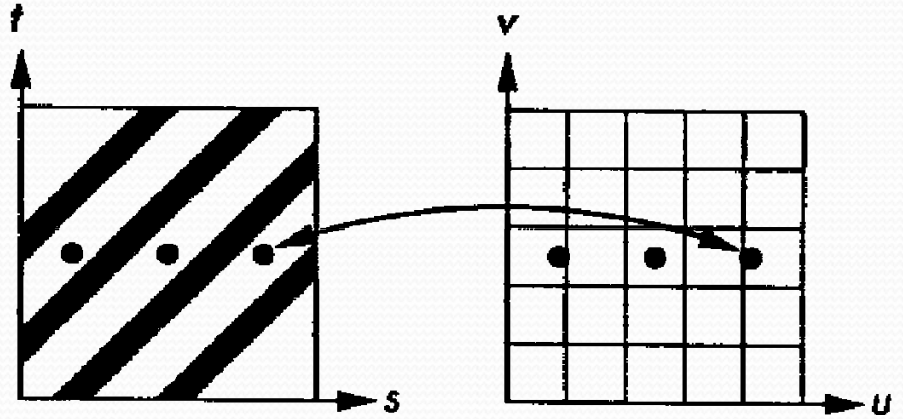
- Does it mean that, for every texel (a small area ΔA)in the texture map, we draw a pixel on the screen at world coordinates $M(p(s,t))$ with the texel color?

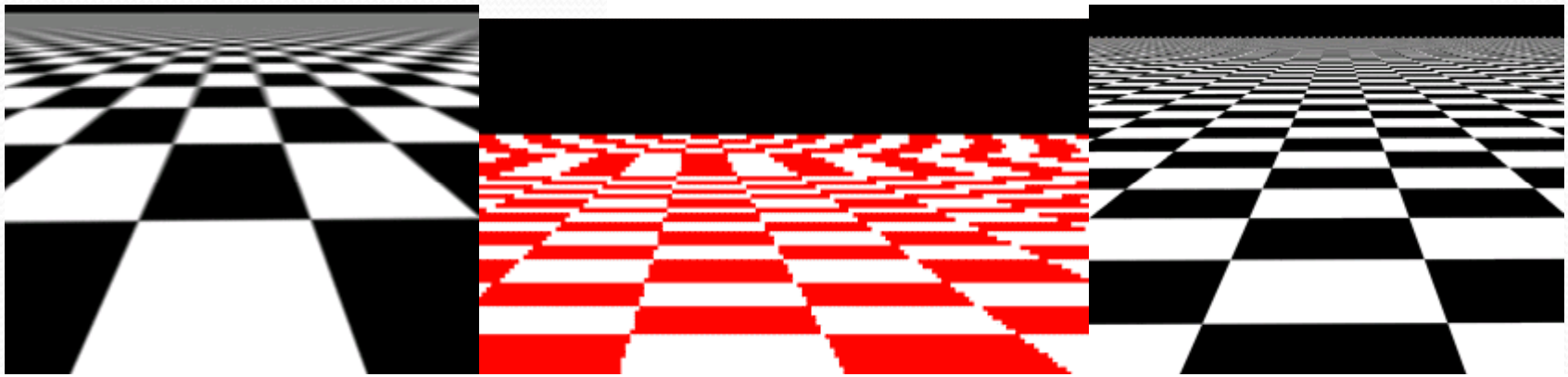


Aliasing

- Aliasing in texture mapping is due to the limited resolution of both the texture map and the frame buffer. This problem is most visible when the texture is periodic.

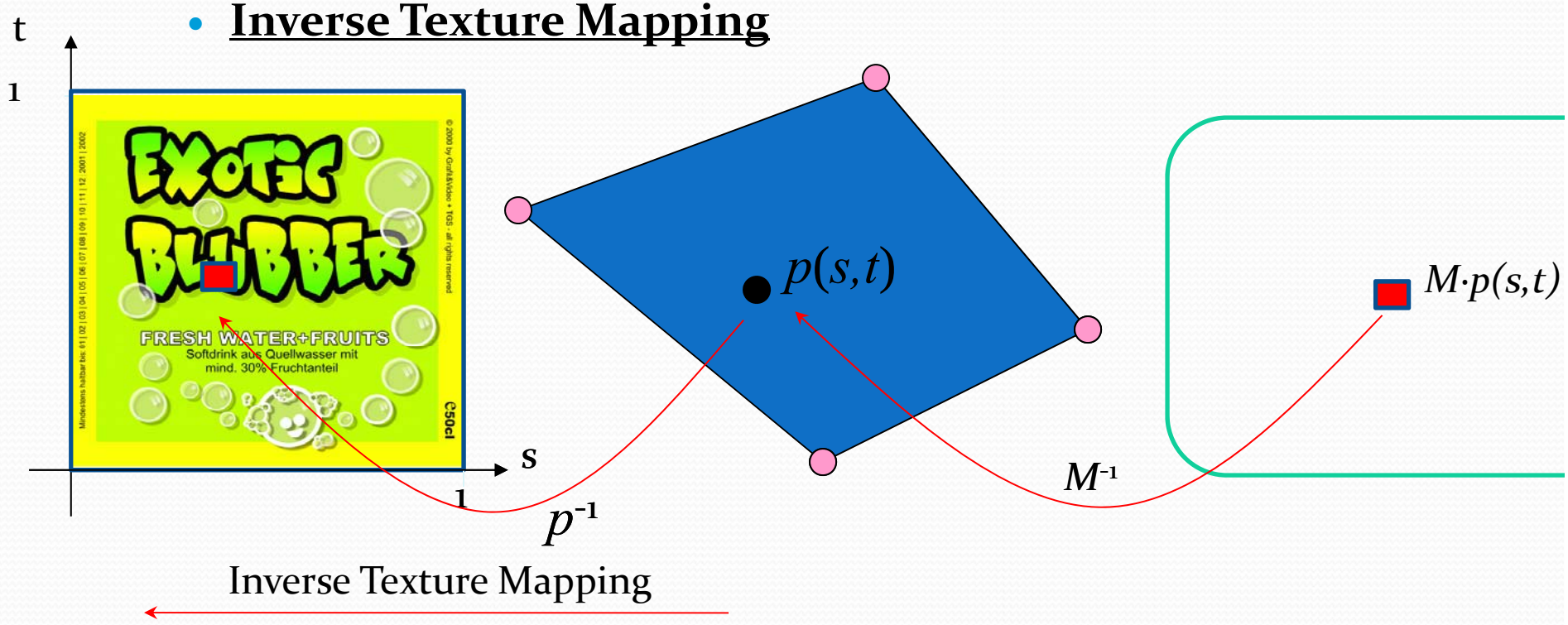
Aliasing in texture generation





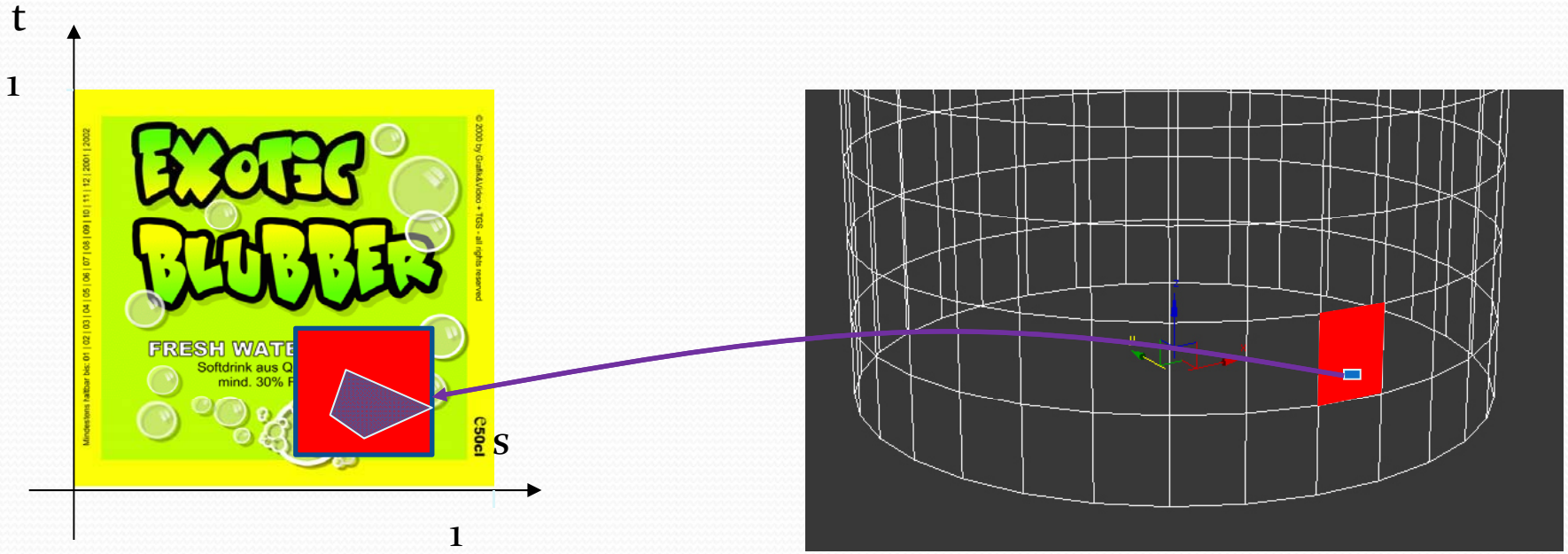
Mapping "Direction"

- Instead of mapping from the texture to the object, then to the screen (which causes aliasing)
  - We do the backward direction, namely
    - **Inverse Texture Mapping**



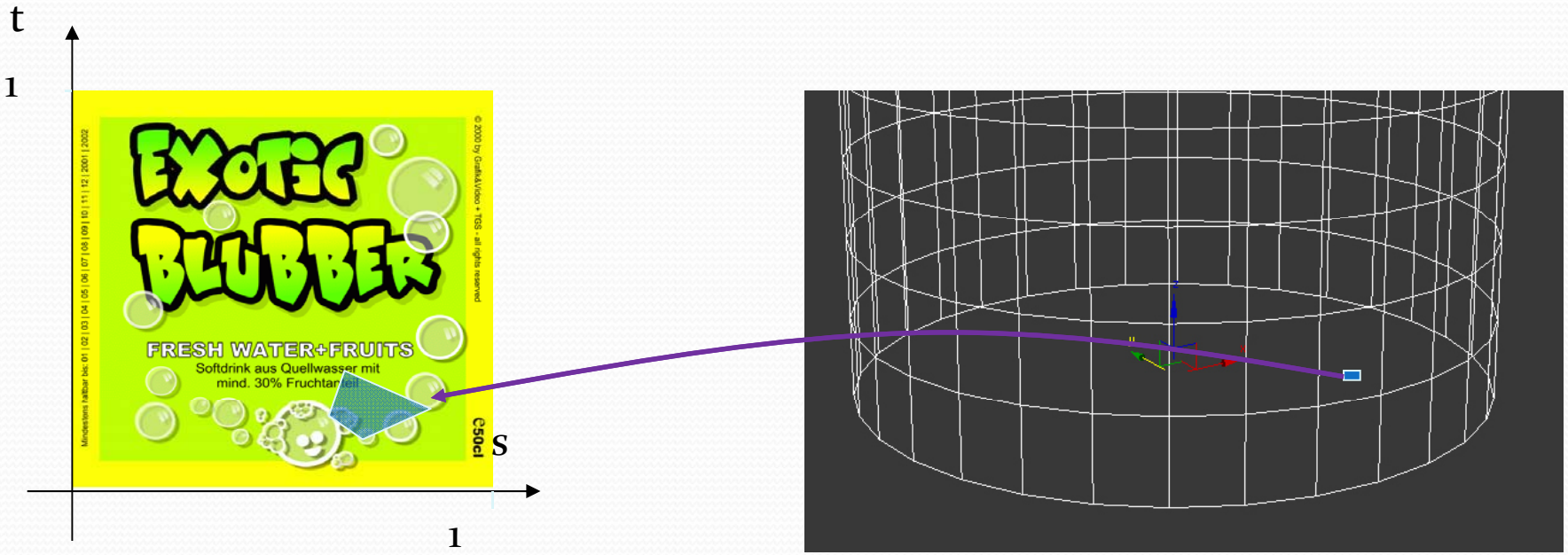
Inverse Texture Mapping

- Since an image is created pixel-by-pixel in the screen space, inverse texture mapping (mapping from screen space to texture space) is most commonly used.
- The function can be derived from the forward mapping function. The problem is reduced to determining a textured color for each pixel.



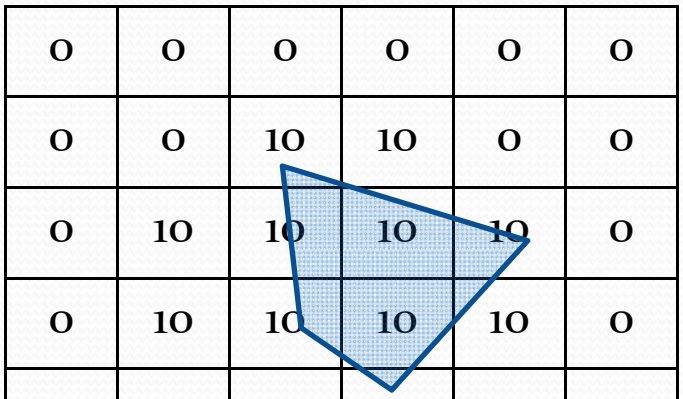
Color of a Pixel

- The color of the pixel during the SCA is
  - The average of the area it covers when it was inversely mapped back to the texture
- Problem: Rendering one pixel needs time to sum up all the texels in the area
  - If the area on the texture is larger, the computing time is longer

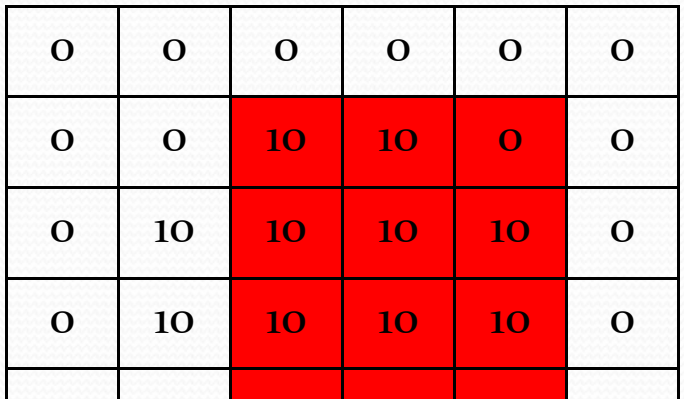


Color of a Pixel

- Compute average of a certain area in the texture covered by that pixel
- Instead, we approximate it by the sum of its bounding box
  - (Sum / number of texels)



Texture



Texture

How to Sum up in O(1) Time?

- Instead of storing the texture image, we store the "Summed area table" as following
  - Each entry of the table is the sum of all pixels that are below and left of it for all RGB values in the texture

0	0	0	0	0	0
0	0	10	10	0	0
0	10	10	10	10	0
0	10	10	10	10	0
0	0	10	10	0	0
0	0	0	0	0	0

Texture

0	20	60	100	120	120
0	20	60	100	120	120
0	20	50	80	100	100
0	10	30	50	60	60
0	0	10	20	20	20
0	0	0	0	0	0

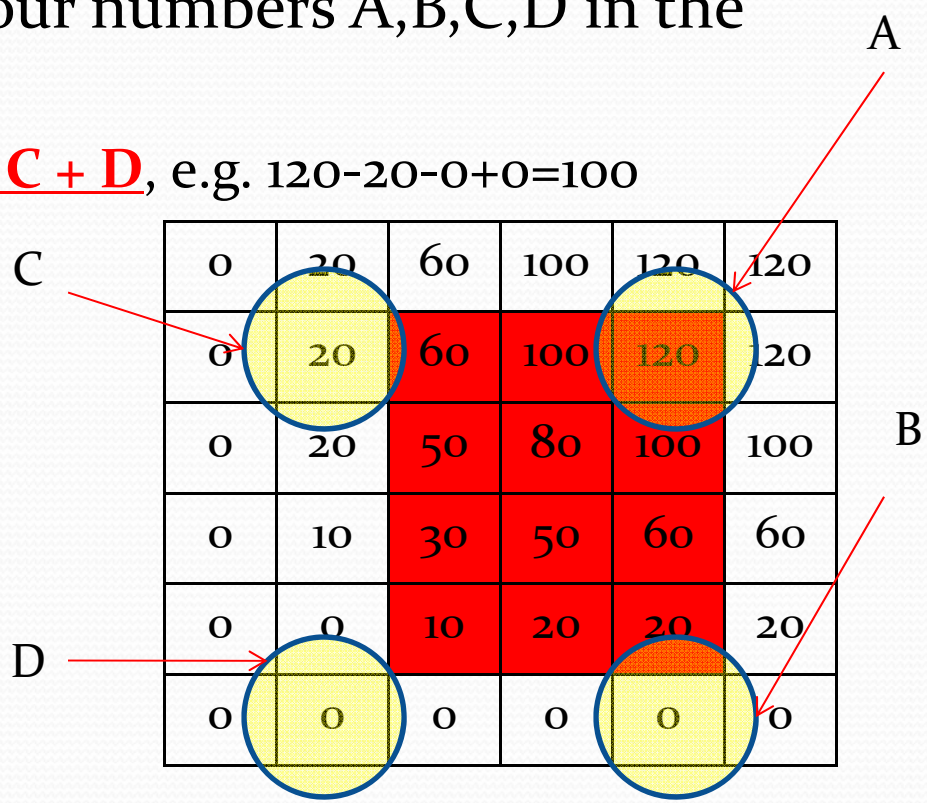
Summed area table

How to Sum up in O(1) Time?

- In order to compute the red area
  - We just need to find the four numbers A,B,C,D in the summed up table
    - The sum is equal to **A – B – C + D**, e.g. 120-20-0+0=100

0	0	0	0	0	0
0	0	10	10	0	0
0	10	10	10	10	0
0	10	10	10	10	0
0	0	10	10	0	0
0	0	0	0	0	0

Texture



Summed area table

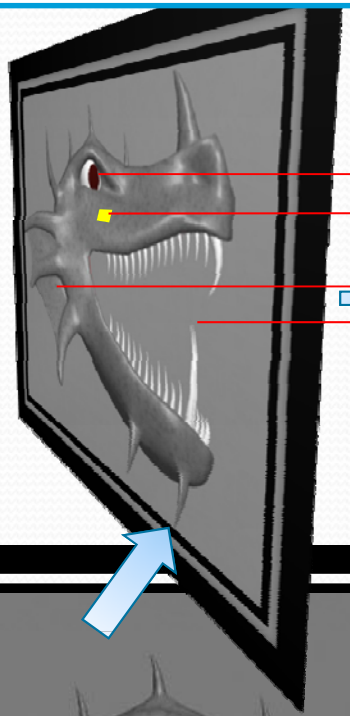
Two Steps Mapping

And Environmental, Reflection mappings

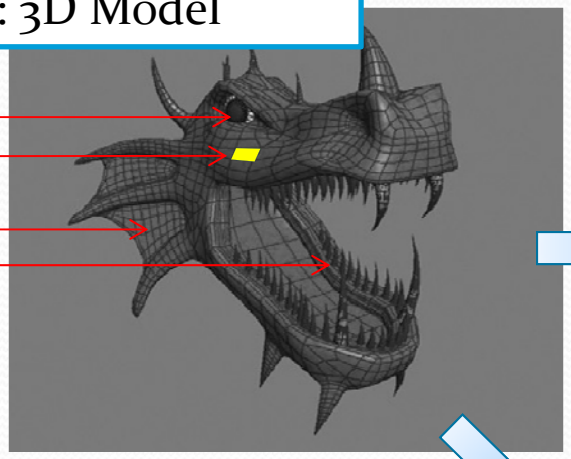
Problem

- However, it is troublesome to map EACH vertex of an object to a texture

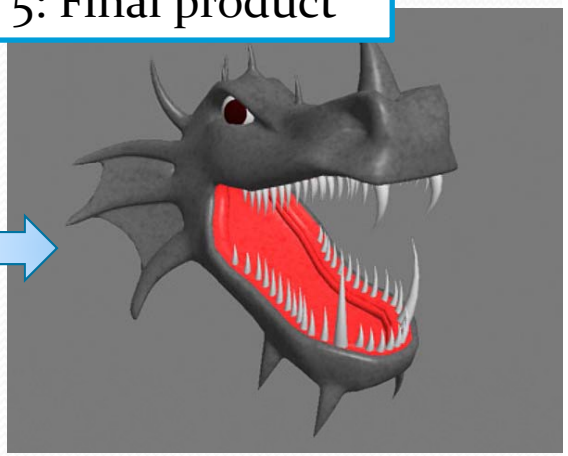
4: Map to the 3D model
(By parallel projection

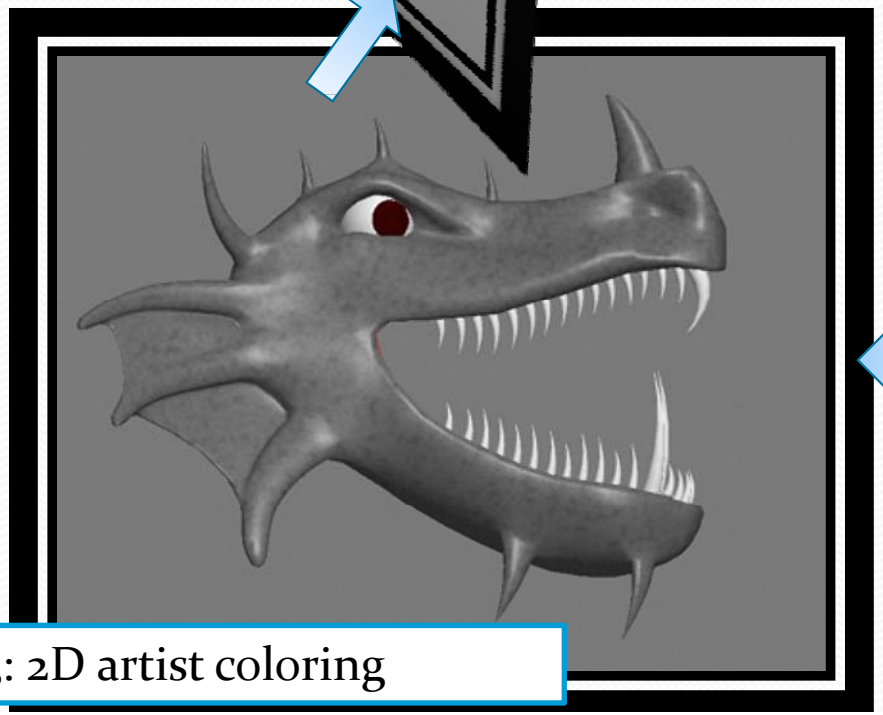


1: 3D Model

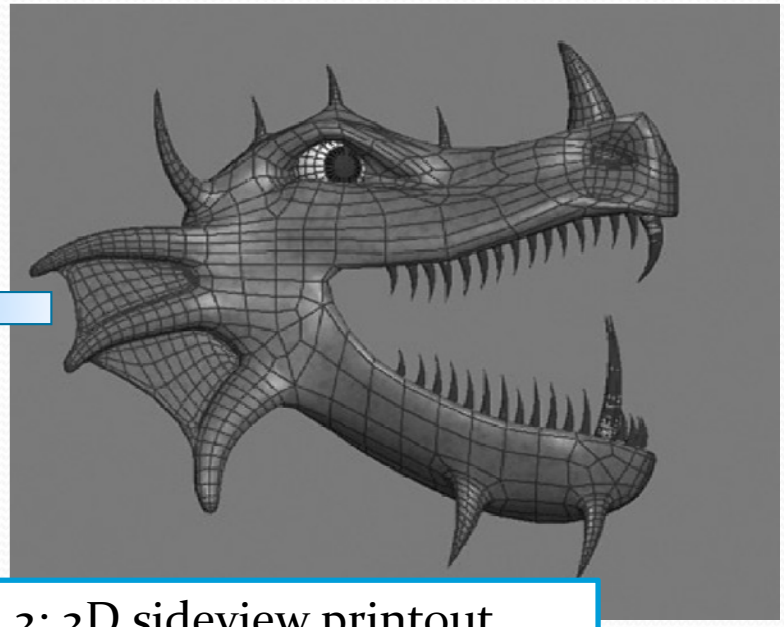


5: Final product

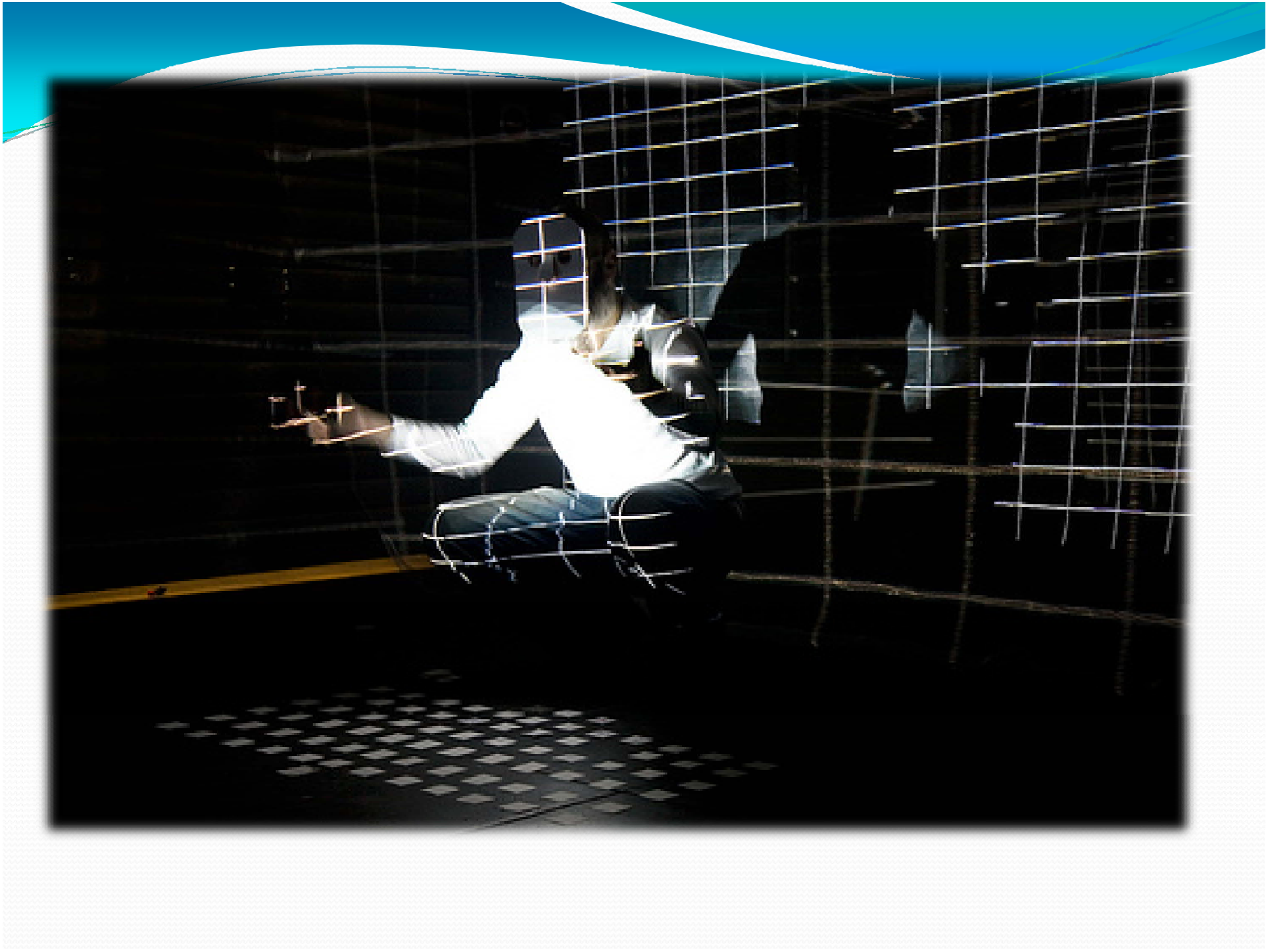




3: 2D artist coloring

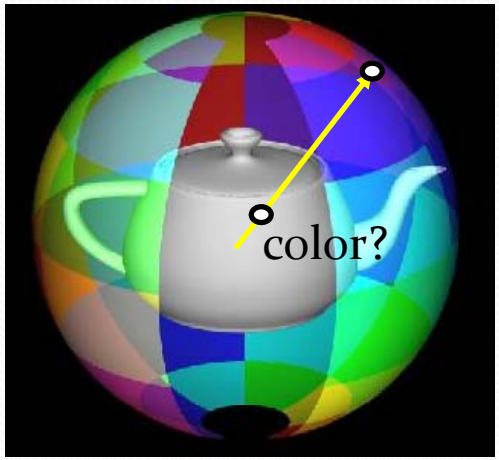


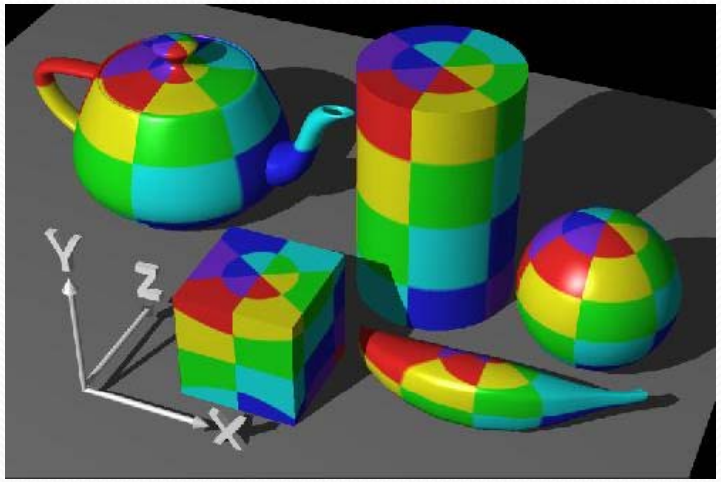
2: 2D sideview printout



Two-step Mapping Method

- 1st step: maps texture to a simple3D **intermediate** surface, such as a plane, a cylinder, sphere or cube.
- 2nd step: the intermediate surface containing the mapped texture is mapped to the surface being rendered.

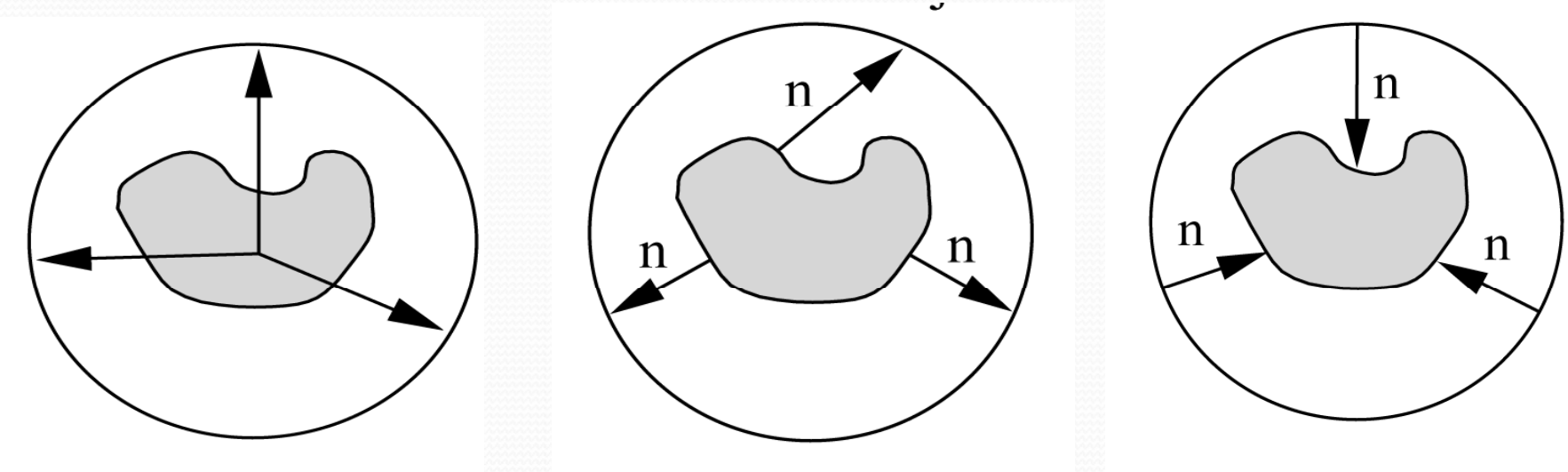




- Many different ways of doing it

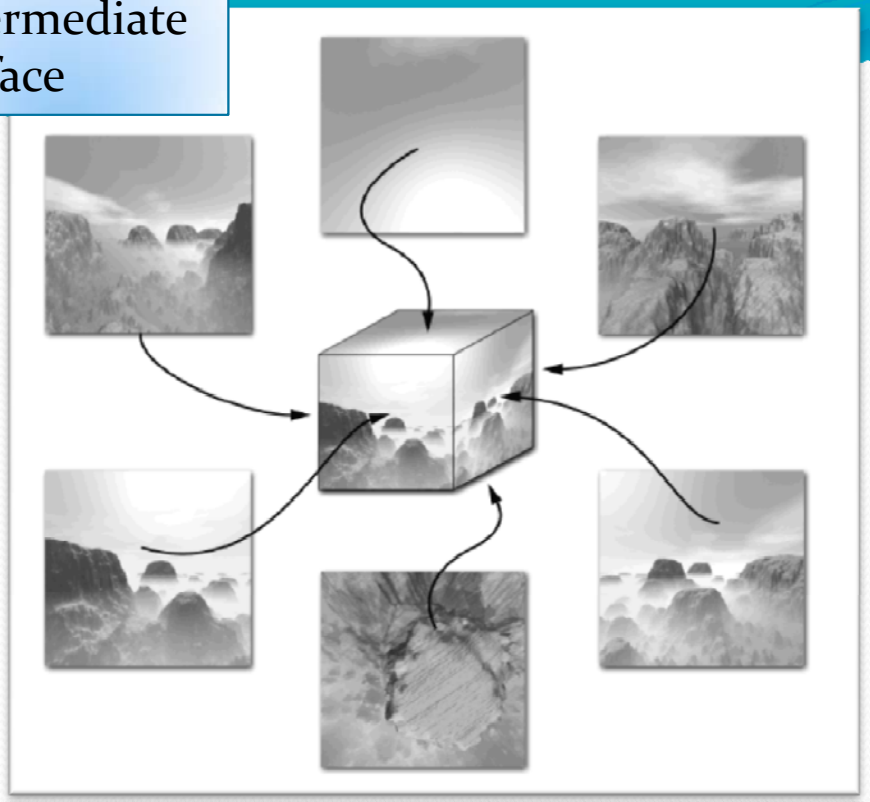
Ways for Second Steps

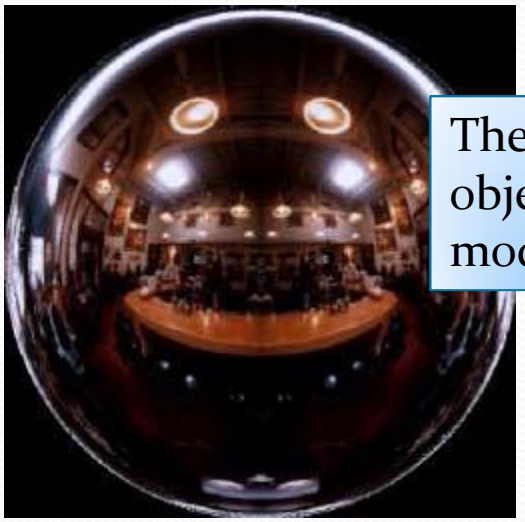
1. From the centroid of the object, draw a line going through the object to the intermediate surface
   - Whatever the color is on the line, assign that color to the object surface
2. Similar, but follow the surface normal of the object
3. Follow the surface normal of the intermediate surface



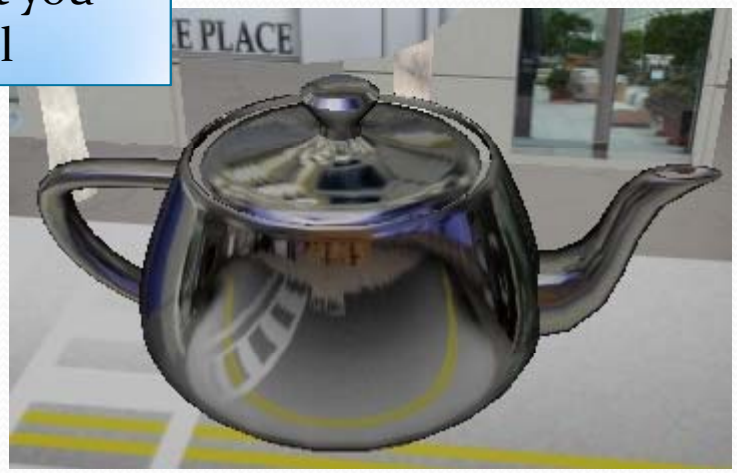
Environmental Mapping

- If the picture on the intermediate object is your environment



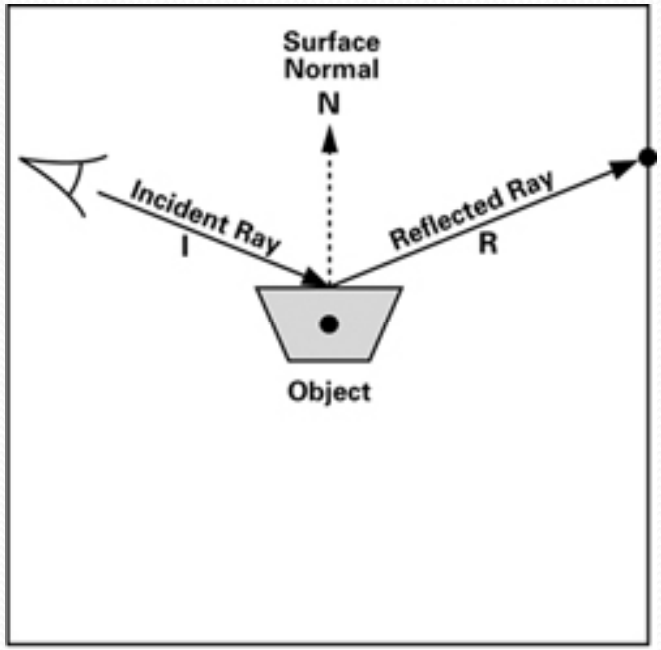


The real 3D object you model



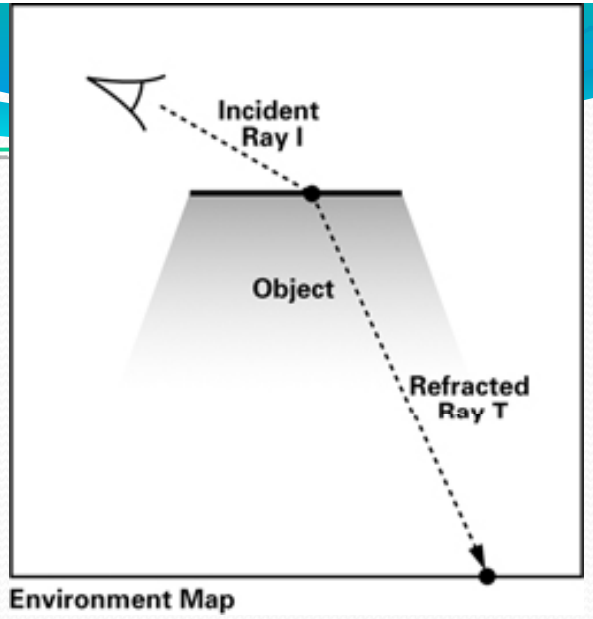
Reflection Map

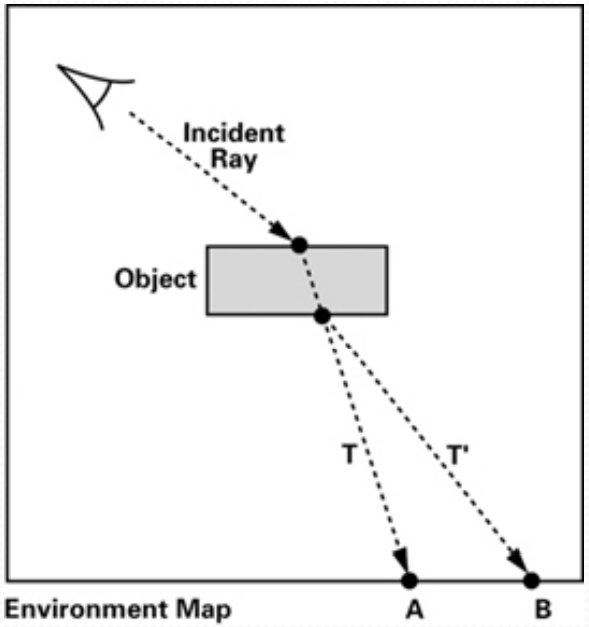
- If the mapping from the intermediate object to your model depends on the viewpoint
  - Environmental texture on the object
    - Moves if the view point moves
    - Moves if the object moves also

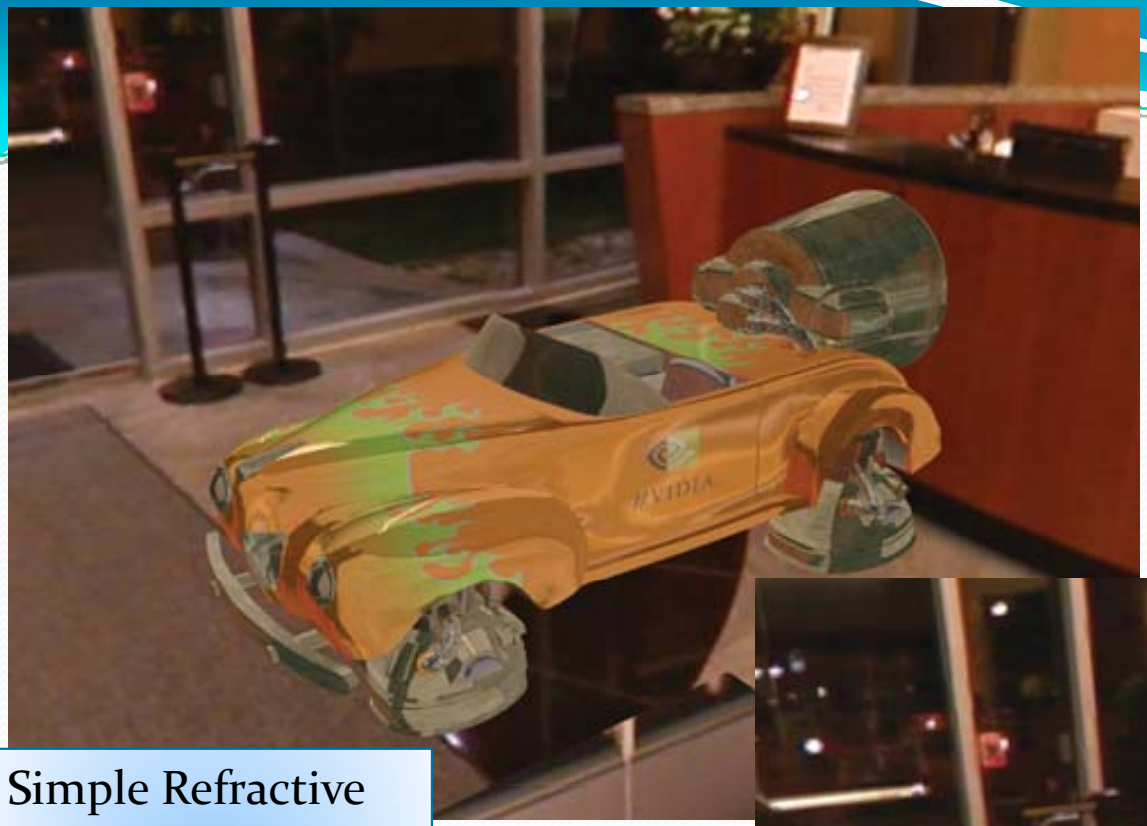


Refractive Map

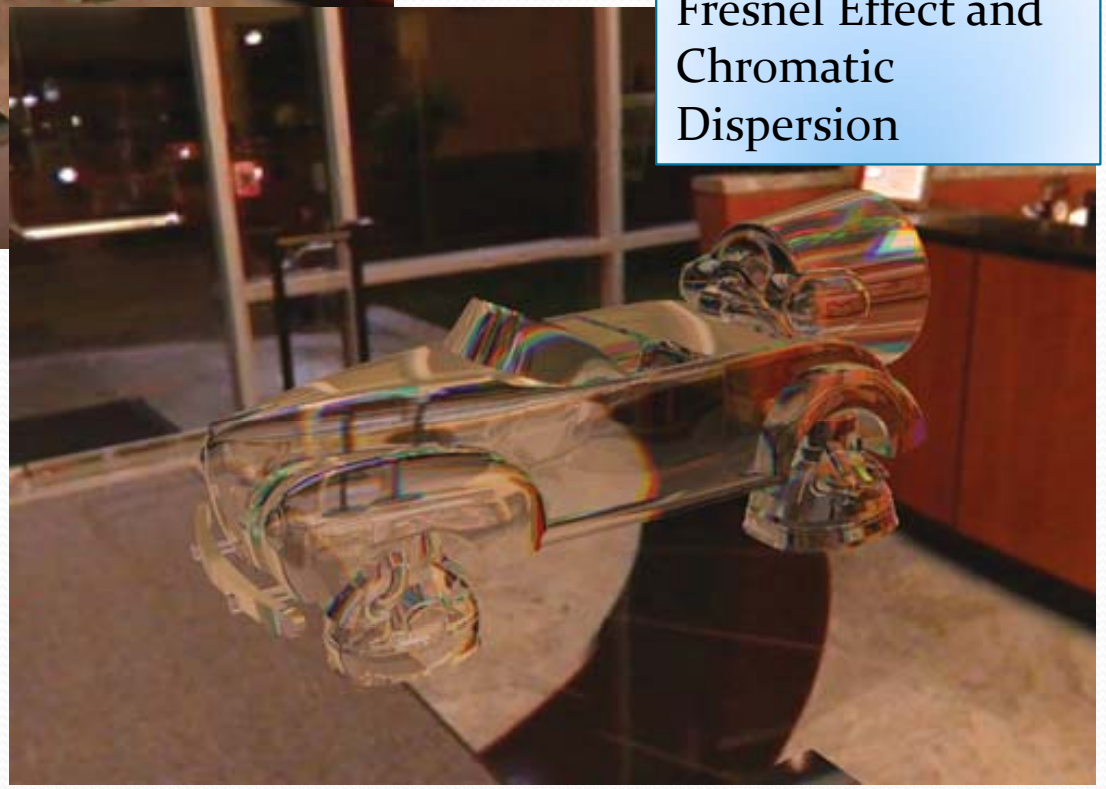
- Same as reflection map, just following the refraction instead of the reflection
  - However, usually use one refraction on the surface of the object
  - Using two refractions is better, but the computation will be increased substantially







Simple Refractive Map



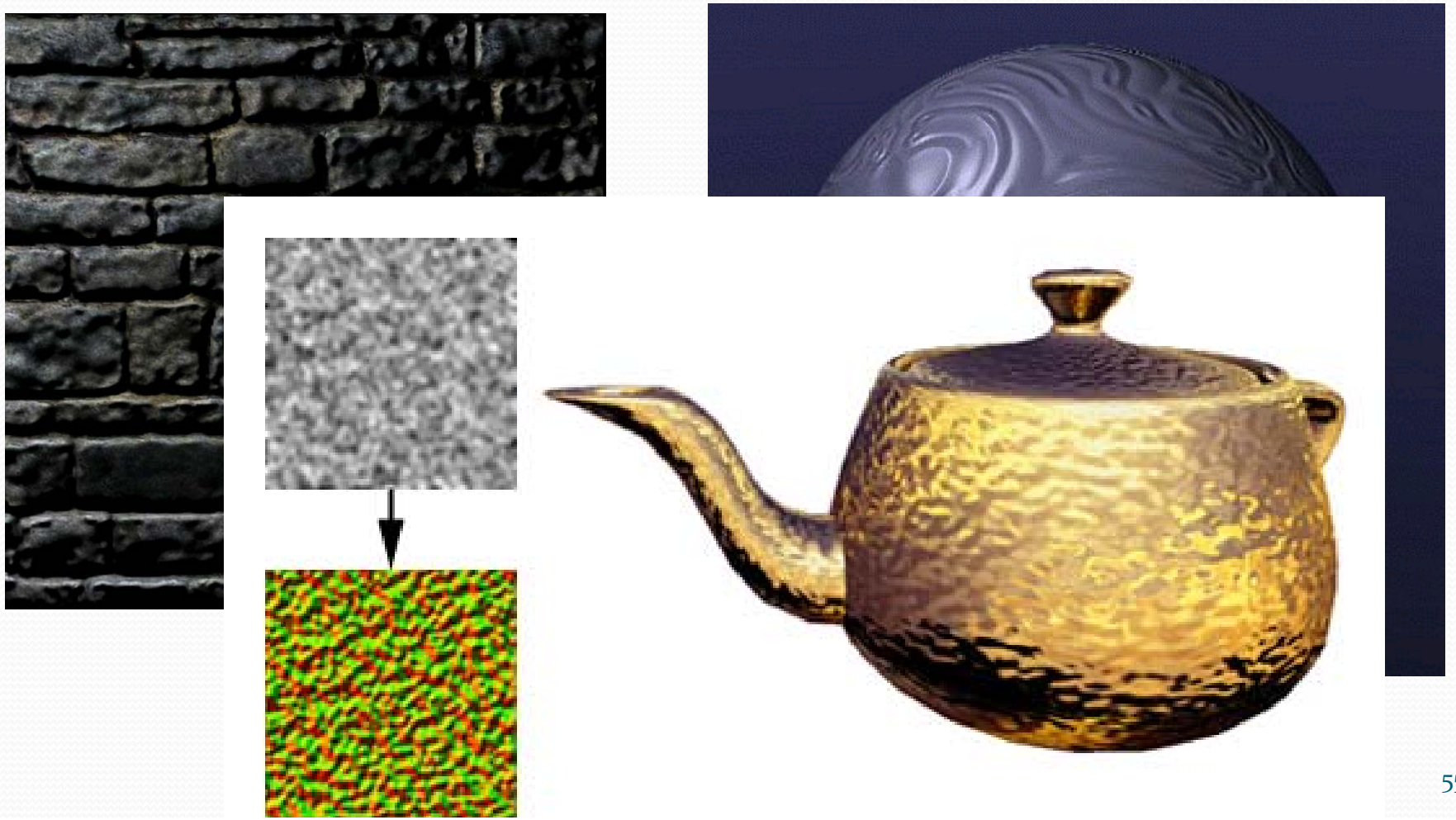
With additional Fresnel Effect and Chromatic Dispersion

Two Step Mappings

- Fast enough to create reflection/refraction on real time application
- However, the environmental texture is “pre-computed”
  - Therefore, it is **NOT** accurate
  - However, if it’s in a fast pace application like games, it’s hard to detect the “flaws”

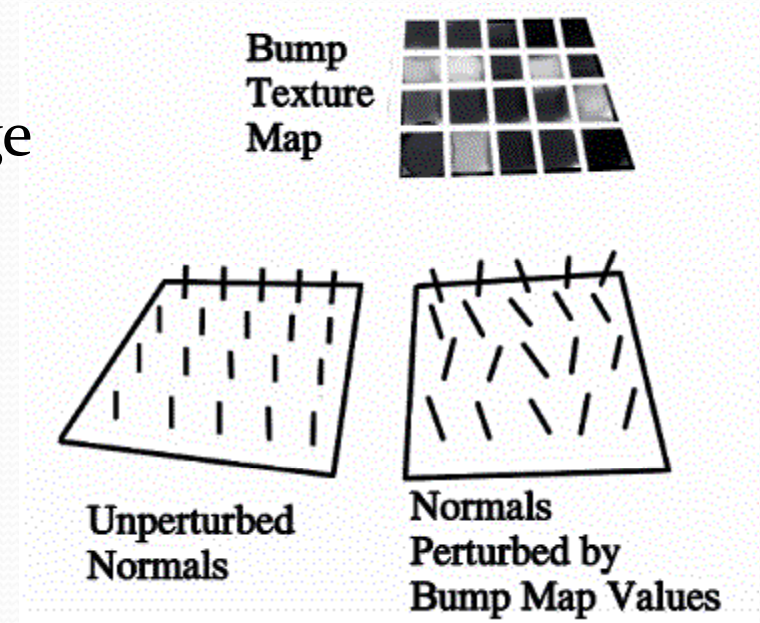
Bump Mapping

Bump Mapping



Texture Mapping vs Bump Mapping

- Texture Mapping
  - Based on texture data, you change the color of each pixel on a polygon
- Bump Mapping
  - Based of a function, you change the normal direction of each pixel on a polygon



OpenGL

- A reference to OpenGL Bump Mapping
  - http://www.paulsprojects.net/tutorials/simplebump/simplebump.html