

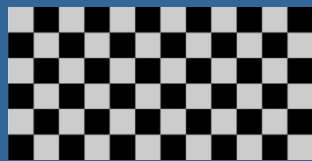
Texturing

Slides done by Tomas Akenine-Möller
and Ulf Assarsson

Department of Computer Engineering
Chalmers University of Technology

Texturing: Glue n-dimensional images onto geometrical objects

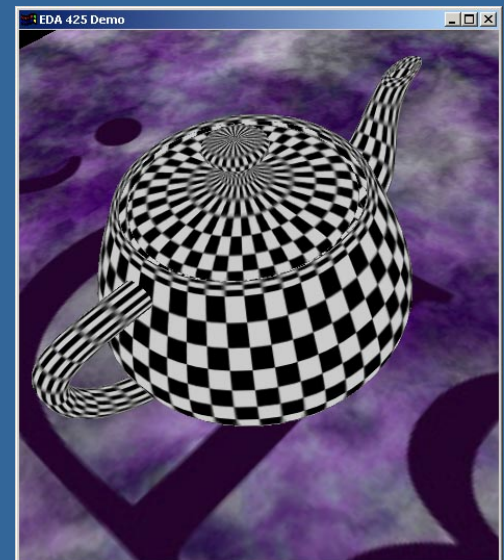
- Purpose: more realism, and this is a cheap way to do it
 - Bump mapping
 - Plus, we can do environment mapping
 - And other things



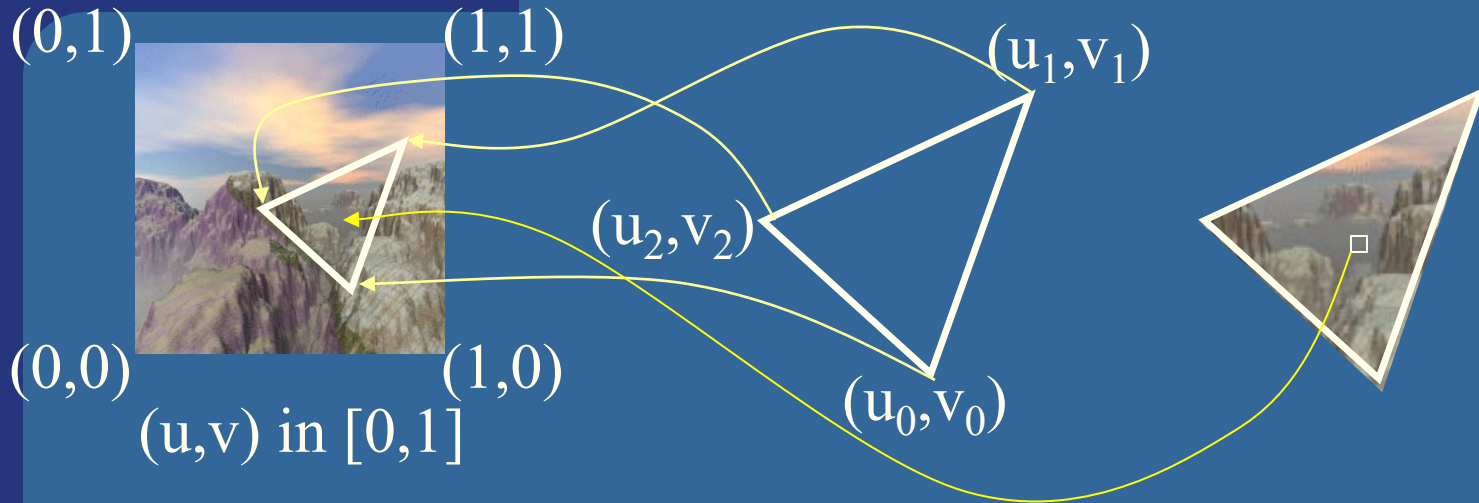
+



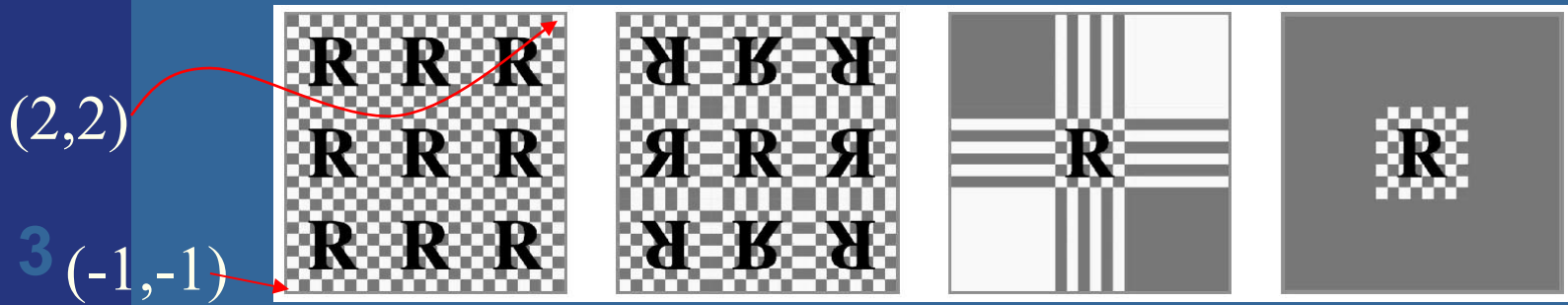
=



Texture coordinates

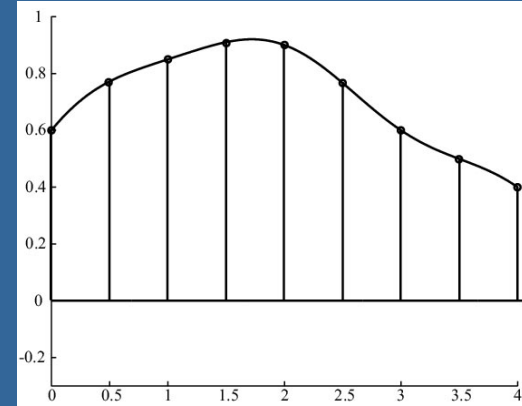
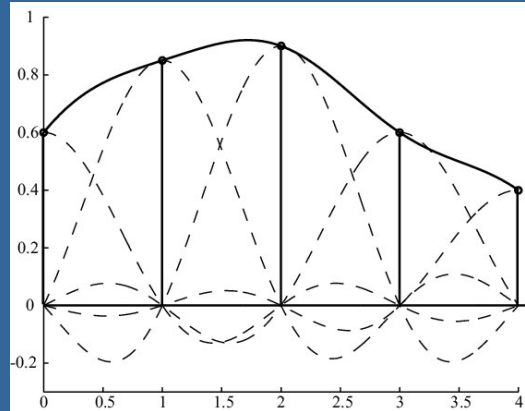


- What if $(u,v) > 1.0$ or < 0.0 ?
- To repeat textures, use just the fractional part
 - Example: $5.3 \rightarrow 0.3$
- Repeat, mirror, clamp, border:

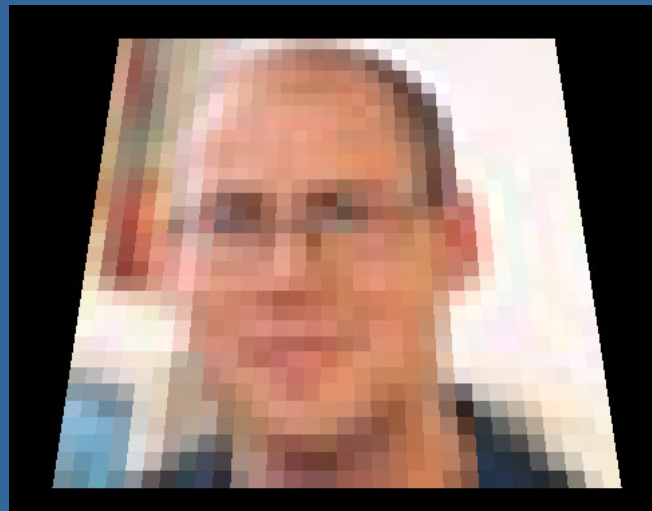


Texture magnification

- What does the theory say...

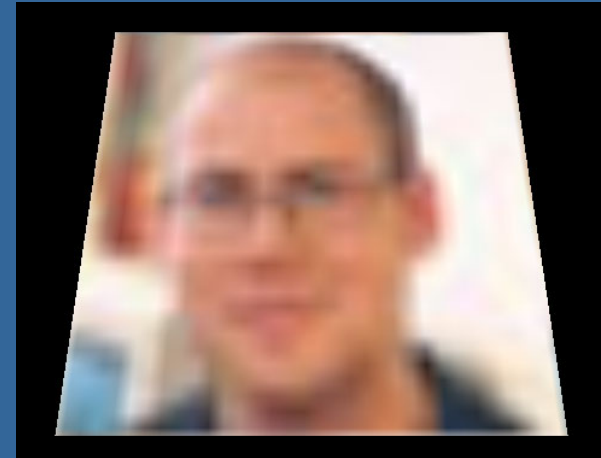
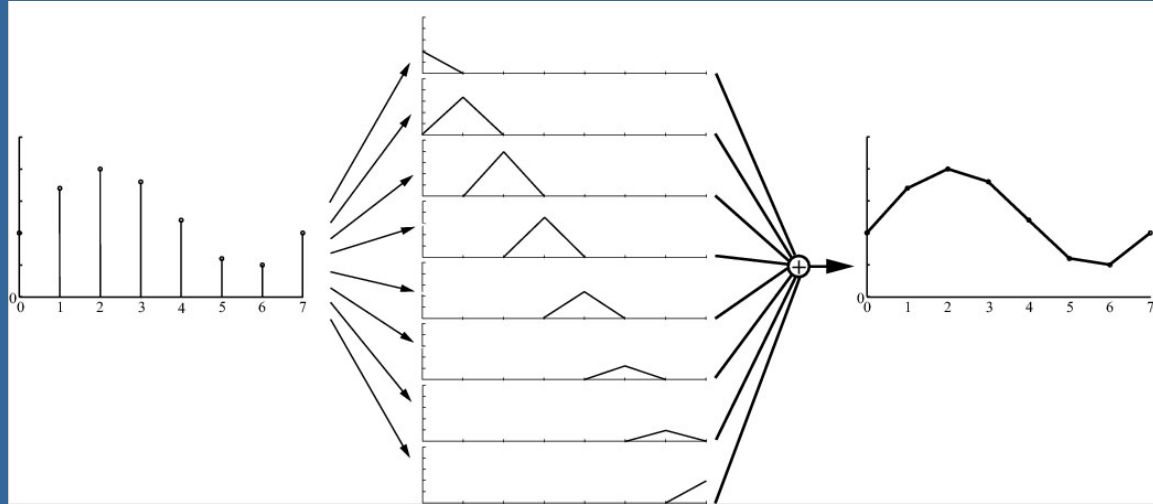


- $\text{sinc}(x)$ is not feasible in real time
- Box filter (nearest-neighbor) is
- Poor quality



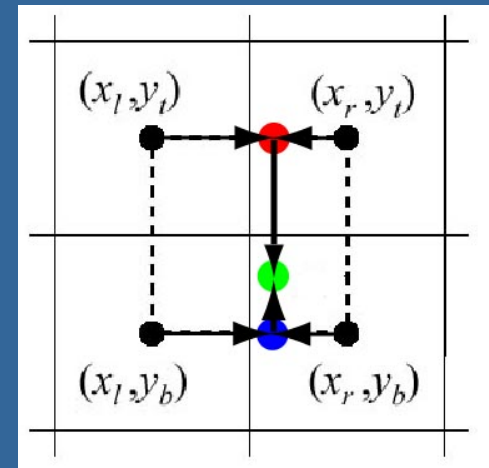
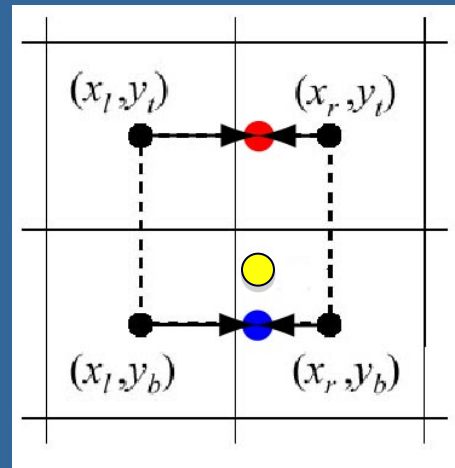
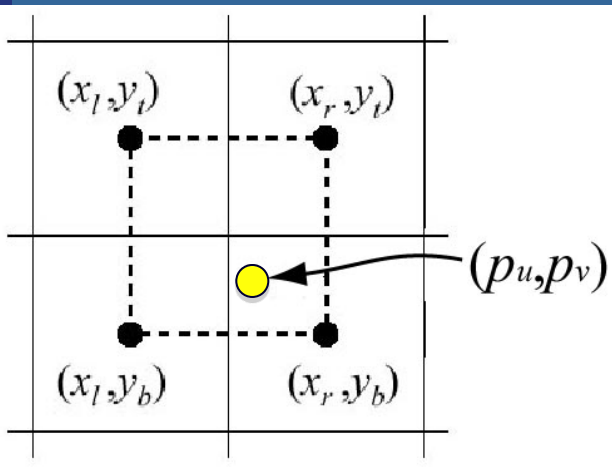
Texture magnification

- Tent filter is feasible!
- Linear interpolation
- Looks better
- Simple in 1D:
 - $(1-t) \cdot \text{color}_0 + t \cdot \text{color}_1$
 - How about 2D?



Bilinear interpolation

- Texture coordinates (p_u, p_v) in $[0, 1]$
- Texture images size: $n * m$ texels
- Nearest neighbor would access: $(\text{floor}(n * u + 0.5), \text{floor}(m * v + 0.5))$
- Interpolate 1D in x & y respectively



Bilinear interpolation

- Check out this formula at home
- $\mathbf{t}(u,v)$ accesses the texture map
- $\mathbf{b}(u,v)$ filtered texel
- (u',v') = fractional part of texel coordinate

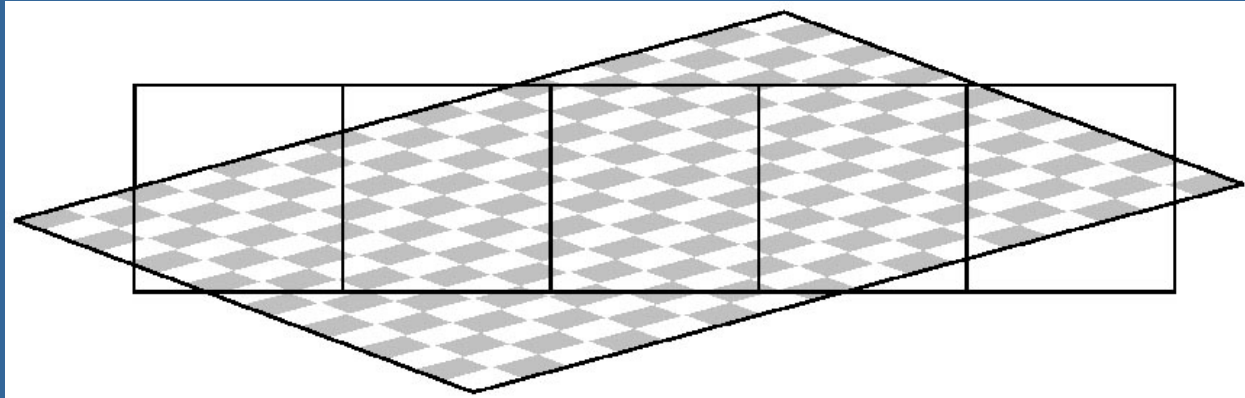
weights

$$(u', v') = (p_u - \lfloor p_u \rfloor, p_v - \lfloor p_v \rfloor).$$

$$\mathbf{b}(p_u, p_v) = (1 - u')(1 - v')\mathbf{t}(x_l, y_b) + u'(1 - v')\mathbf{t}(x_r, y_b) \\ + (1 - u')v'\mathbf{t}(x_l, y_t) + u'v'\mathbf{t}(x_r, y_t).$$

Texture minification

What does a pixel "see"?

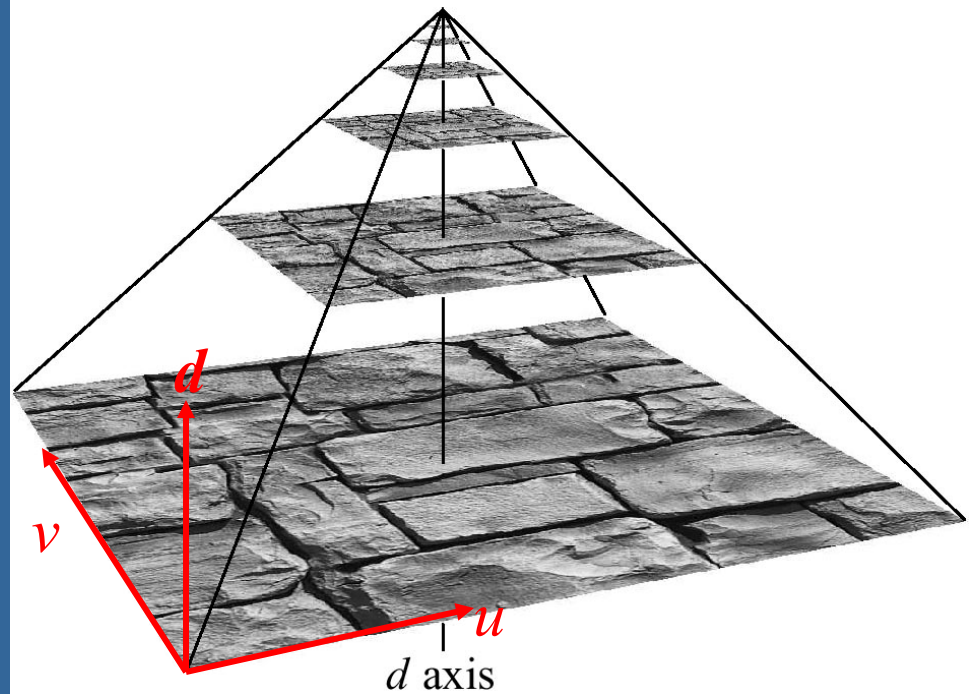


- Theory (sinc) is too expensive
- Cheaper: average of texel inside a pixel
- Still too expensive, actually

- Mipmaps – another level of approximation
 - Prefilter texture maps as shown on next slide

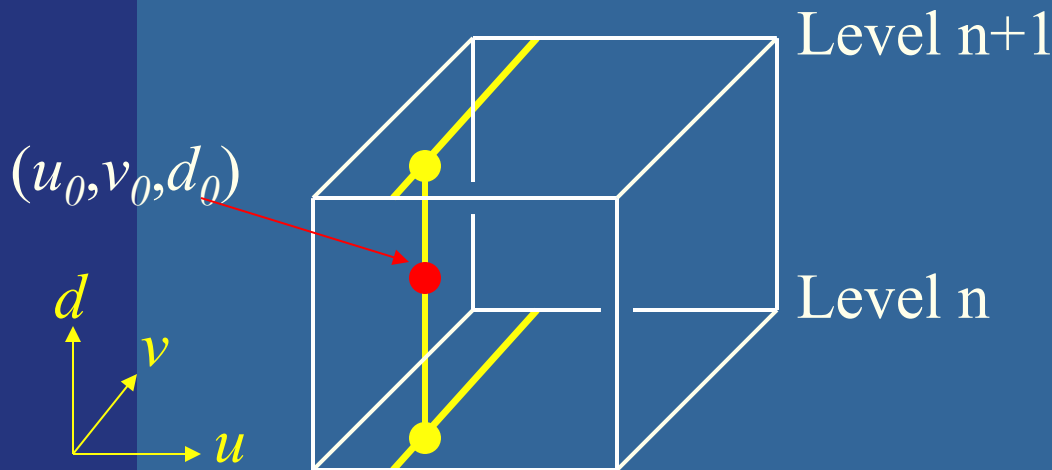
Mipmapping

- Image pyramid
- Half width and height when going upwards
- Average over 4 "child texels" to form "parent texel"
- Depending on amount of minification, determine which image to fetch from
- Compute d first, gives two images
 - Bilinear interpolation in each



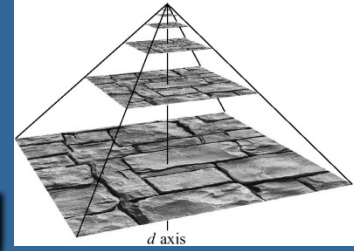
Mipmapping

- Interpolate between those bilinear values
 - Gives trilinear interpolation

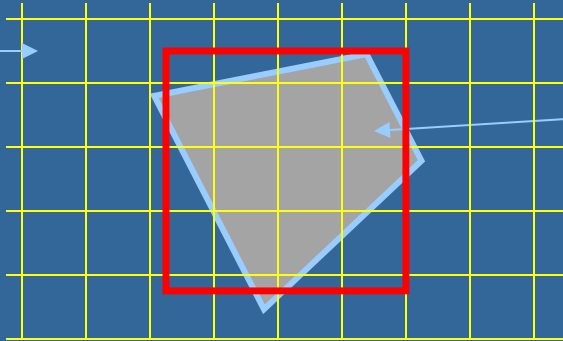


- Constant time filtering: 8 texel accesses
- How to compute d ?

Computing d for mipmapping



texel



pixel projected
to texture space

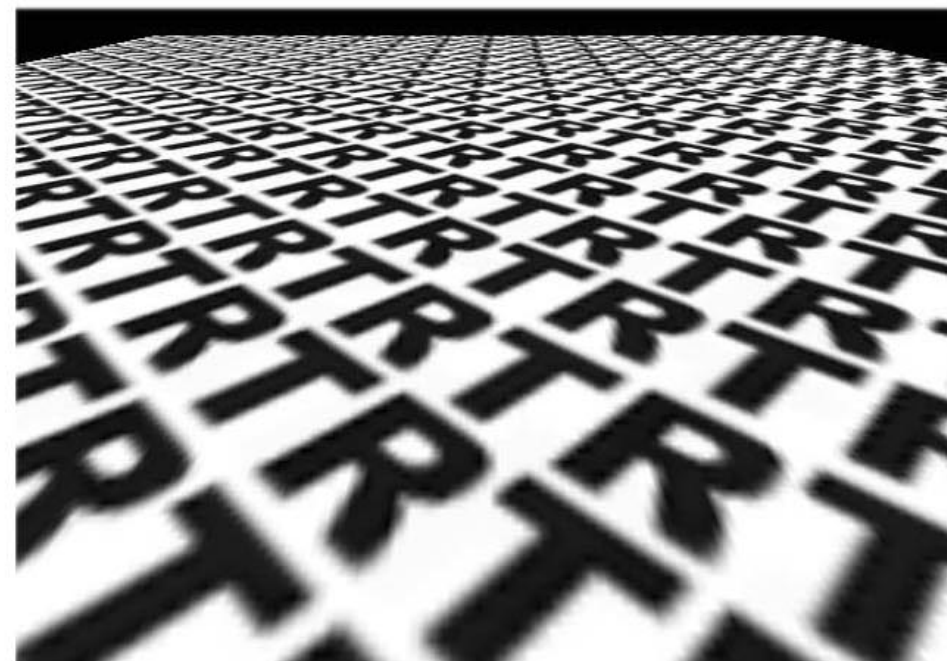
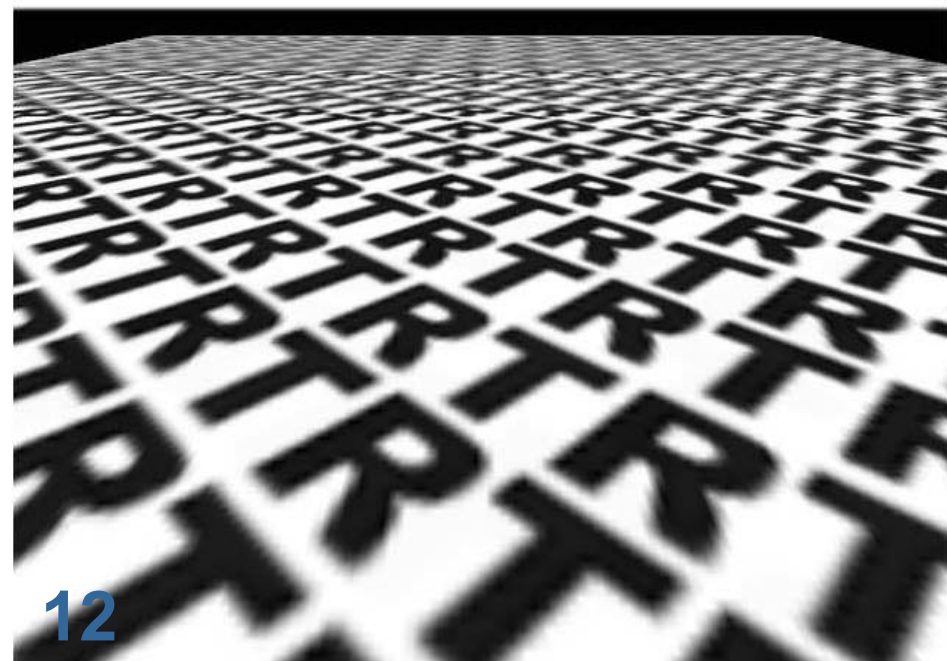
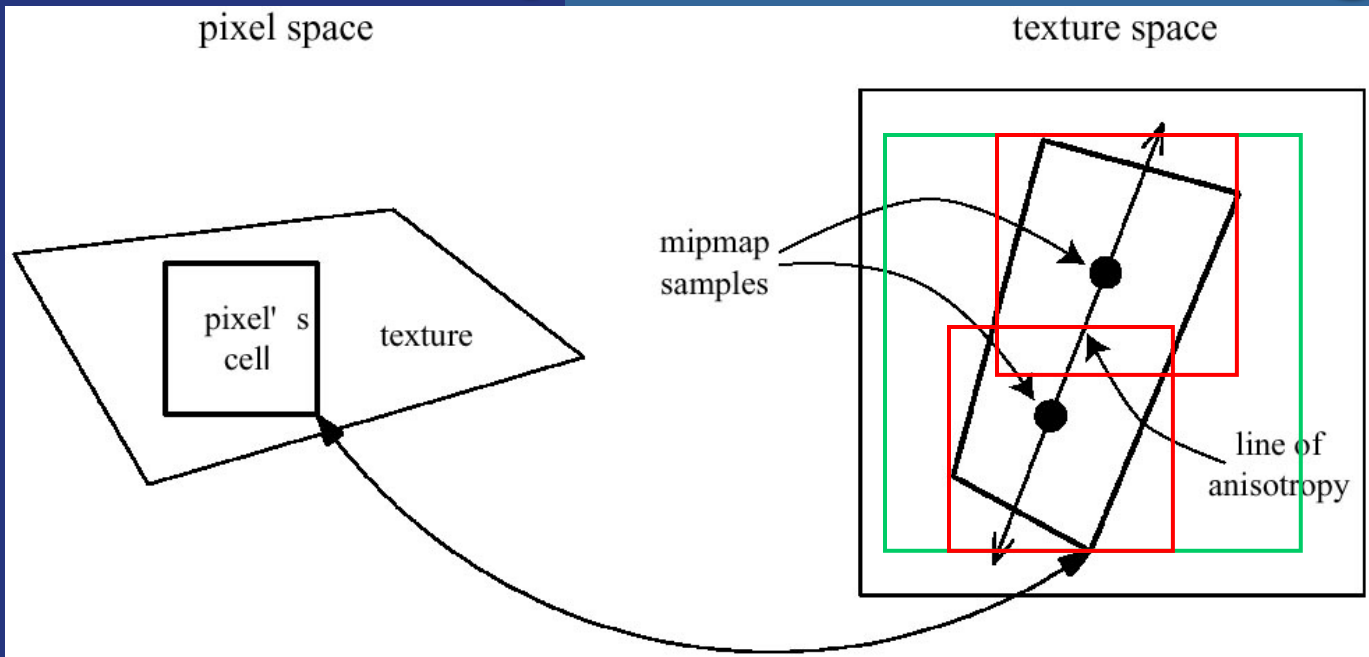
A = approximative area of quadrilateral

$$b = \sqrt{A}$$

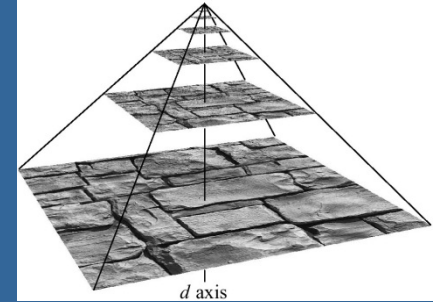
$$d = \log_2 b$$

- Approximate quad with square
- Gives overblur!
- Even better: anisotropic texture filtering
 - Approximate quad with several smaller mipmap samples

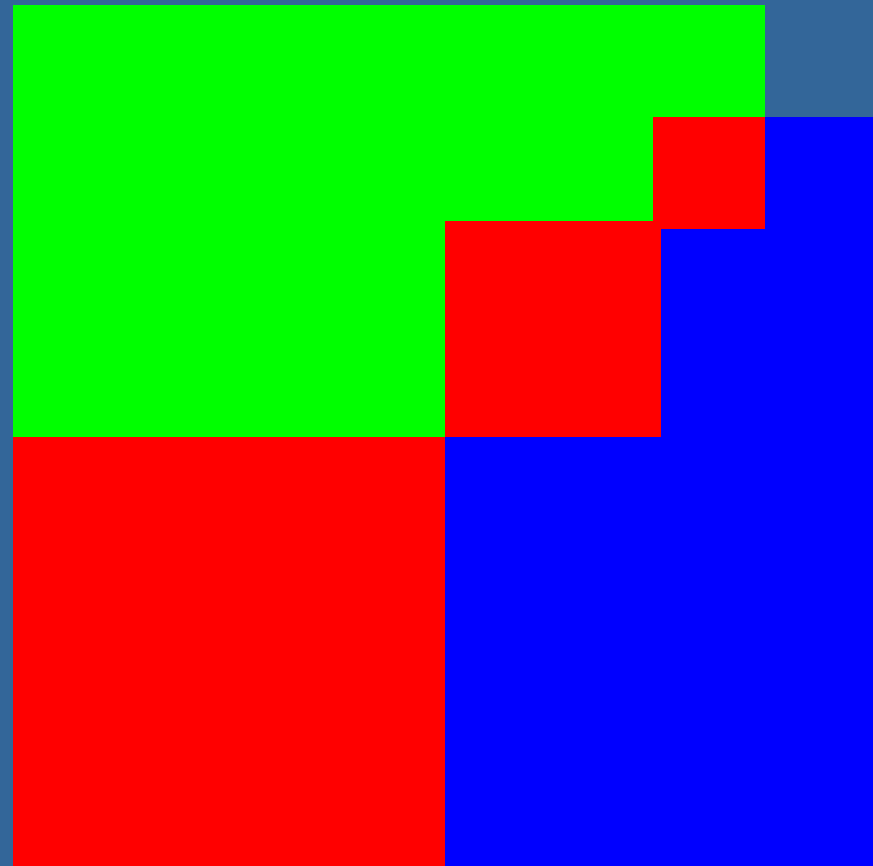
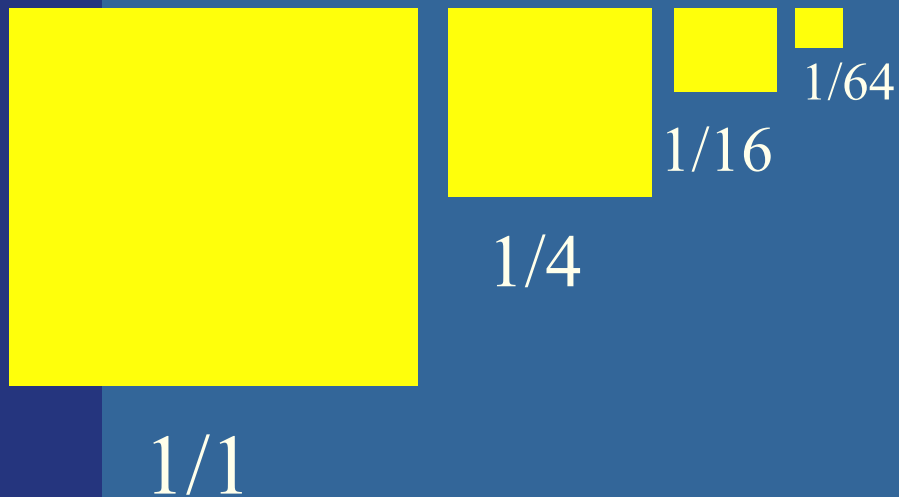
Anisotropic texture filtering



Mipmapping: Memory requirements



- Not twice the number of bytes...!



- Rather 33% more – not that much

Miscellaneous

- How to apply texturing:
 - Add, sub, etc as you like, using fragment shaders.

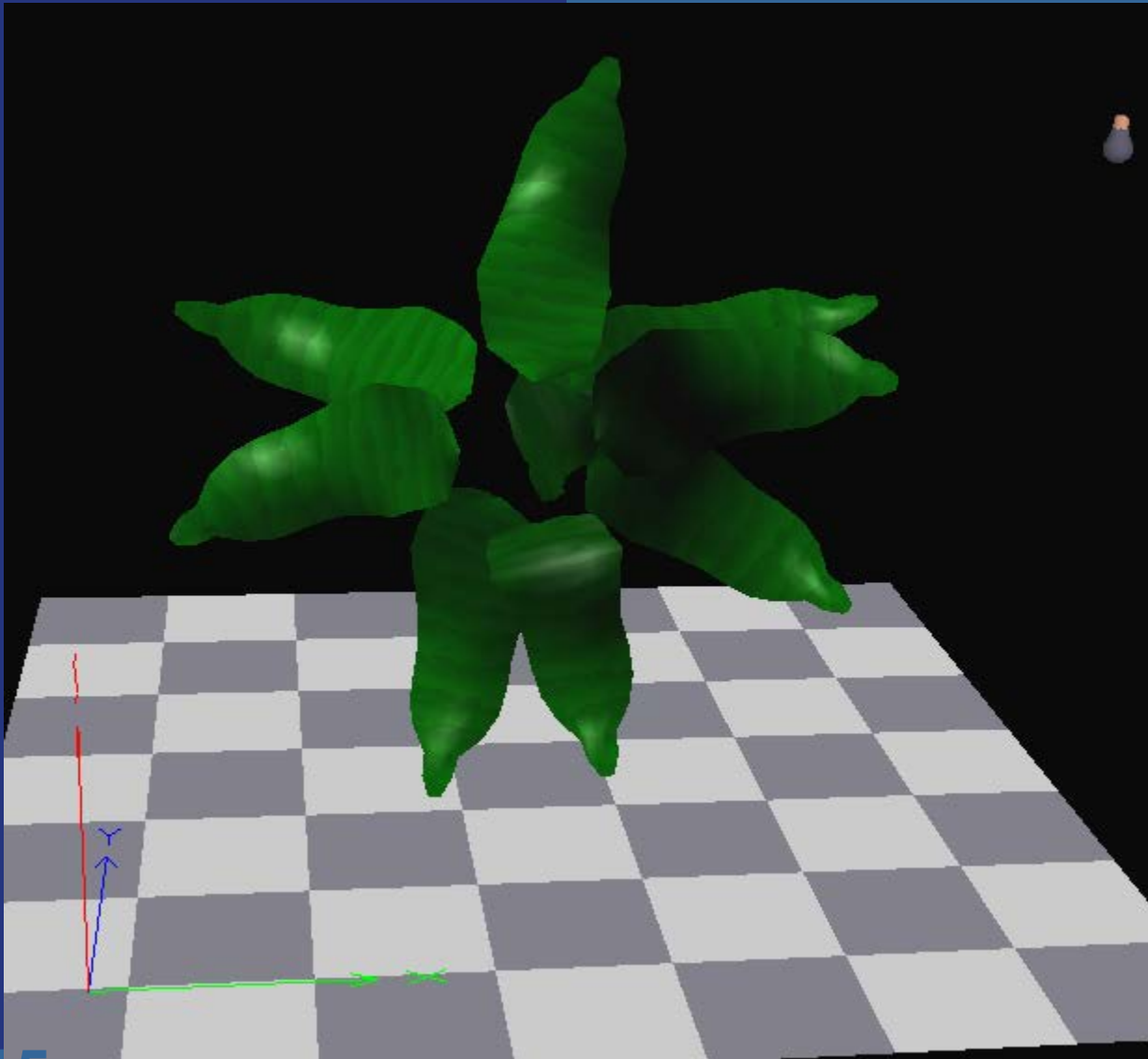
Common alternatives:

- Modulate (multiply texture with lighting)
- "Replace" (just use texture color)

Often:

- diffuseTexture, (specularTexture, ambientTexture)
- Instead of ambMtrl, diffMtrl, specMtrl

Modulate



Texture multiplied with result from lighting (amb, diff, spec)

Using textures in OpenGL

Do once when loading texture:

```
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);
int w, h, comp; // width, height, #components (rgb=3, rgba=4), #comp
unsigned char* image = stbi_load("floor.jpg", &w, &h, &comp, STBI_rgb_alpha);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);
free(image);
glGenerateMipmap(GL_TEXTURE_2D);

//Indicates that the active texture should be repeated over the surface
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
// Sets the type of mipmap interpolation to be used on magnifying and minifying the texture. These are the
// nicest available options.
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, 16);
```

Do every time you want to use this texture when drawing:

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture);
// Now, draw your triangles with texture coordinates specified
```

```
FRAGMENT SHADER

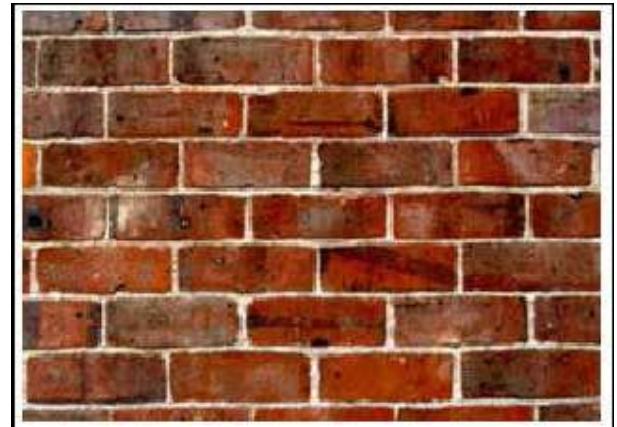
in vec2 texCoord;
void main()
{
    gl_FragColor = texture2D(0,
                            texCoord.xy);
}
```


Light Maps

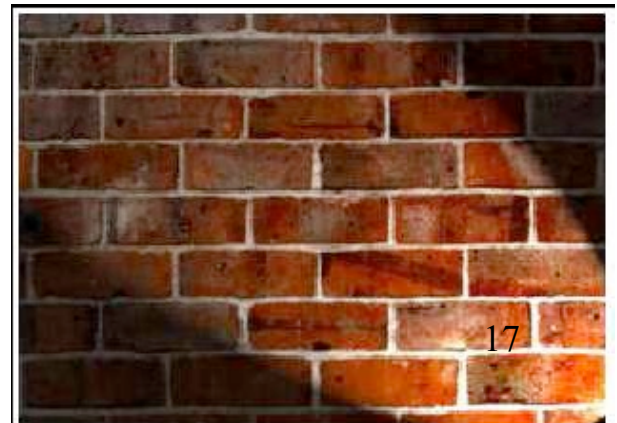
- Often used in games
- Mutliply both textures with each other in the fragment shader, or (old way):
 - render wall using brick texture
 - render wall using light texture and blending to the frame buffer



+



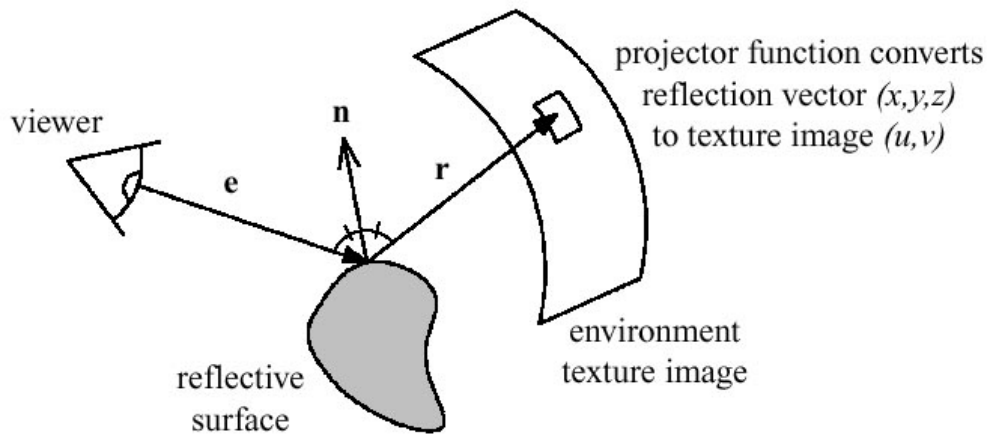
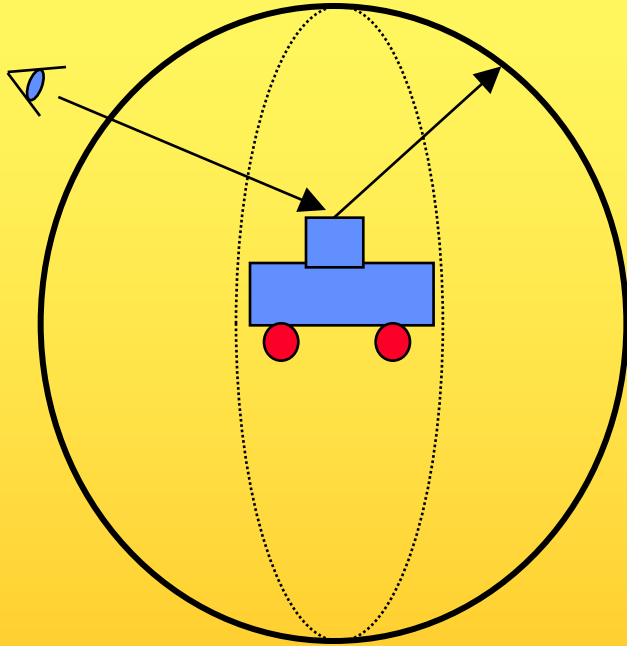
=



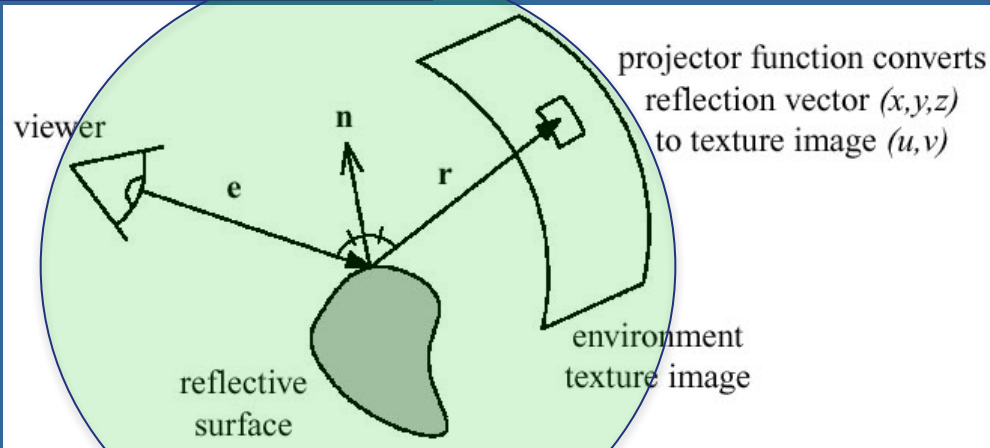


Light-mapped

Environment mapping



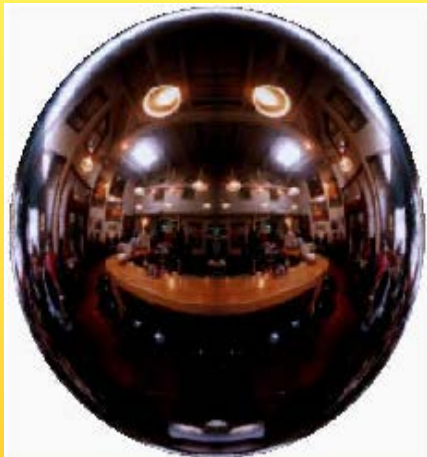
Environment mapping



- Assumes the environment is infinitely far away
- Sphere mapping
- Cube mapping is the norm nowadays
 - Advantages: no singularities as in sphere map
 - Much less distortion
 - Gives better result
 - Not dependent on a view position

Sphere map

- example



Sphere map
(texture)



Sphere map
applied on torus

Sphere Map

- Assume surface normals are available
- Then OpenGL can compute reflection vector at each pixel
- The texture coordinates s, t are given by:
 - (see OH 169 for details)

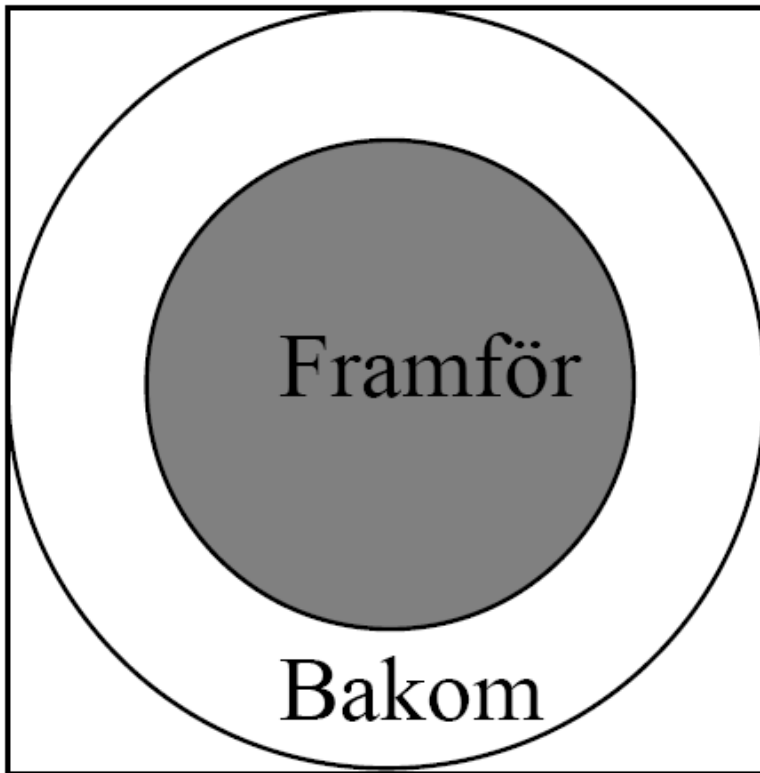
$$L = \sqrt{R_x^2 + R_y^2 + (R_z + 1)^2}$$

$$s = 0.5 \left(\frac{R_x}{L} + 1 \right)$$

$$t = 0.5 \left(\frac{R_y}{L} + 1 \right)$$



Sphere Map

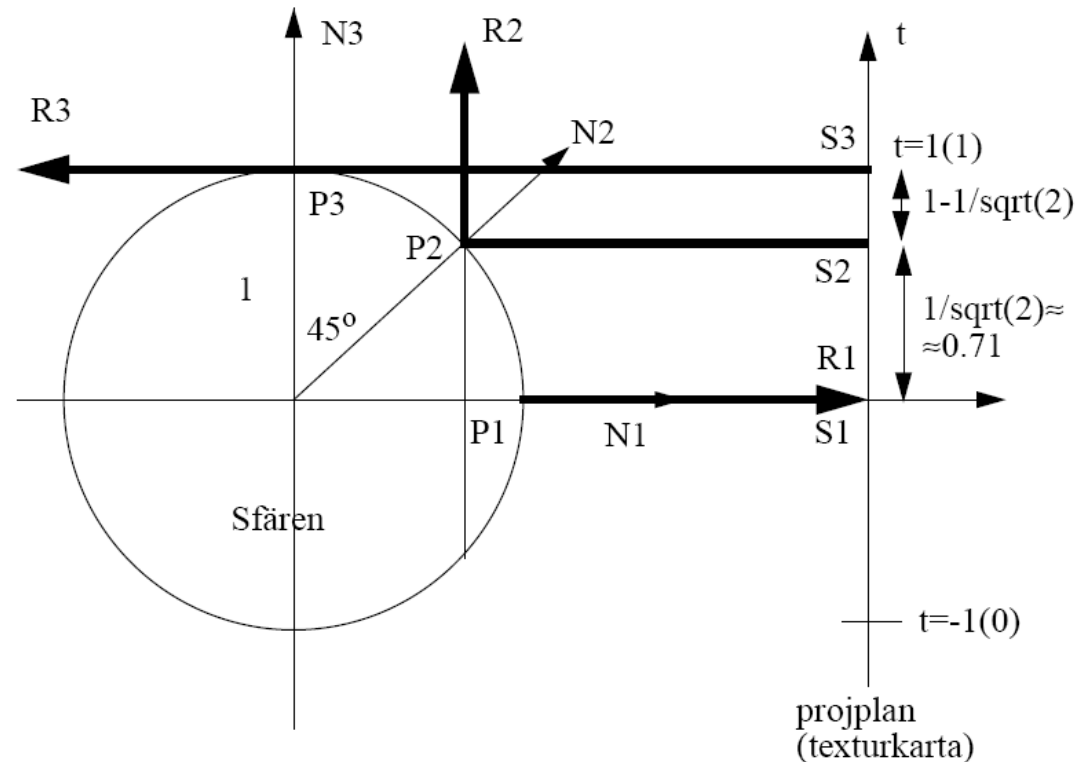


In front of the sphere.
Behind the sphere.

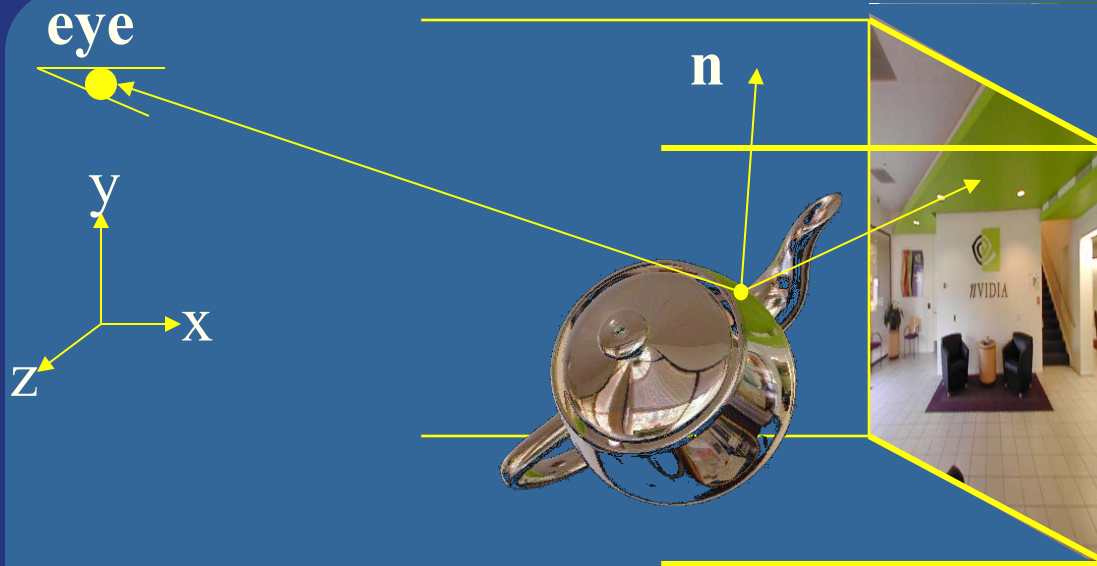
Sphere Map



- Infinitesimally small reflective sphere (infinitely far away)
 - i.e., orthographic view of a reflective unit sphere
- Create by:
 - Photographing metal sphere
 - Or,
 - Ray tracing
 - Transforming cube map to sphere map

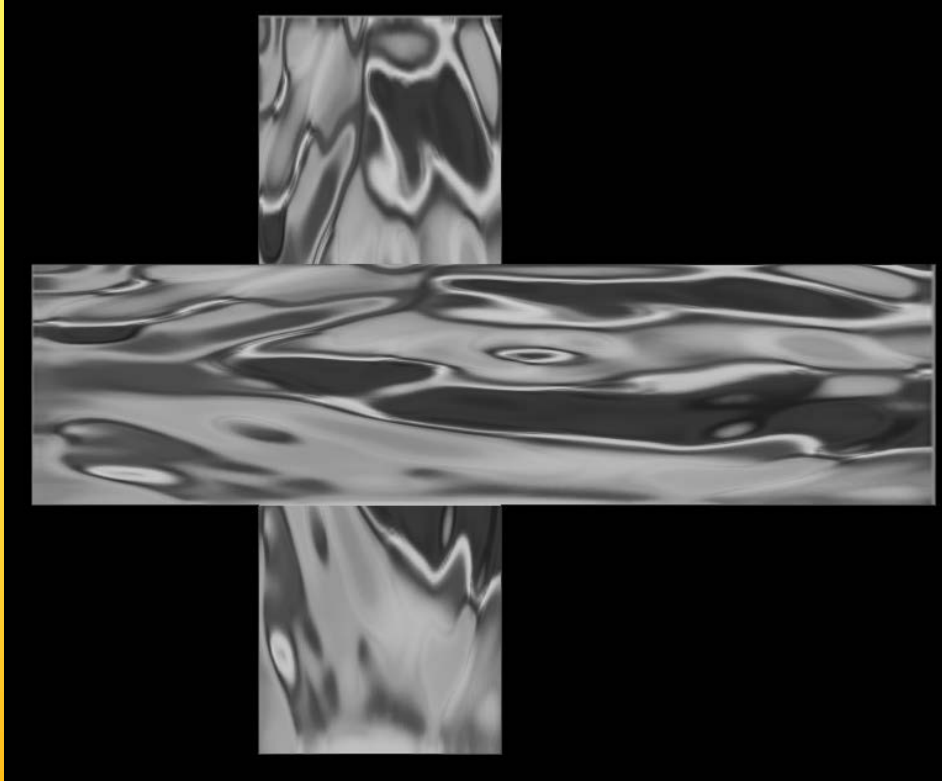


Cube mapping



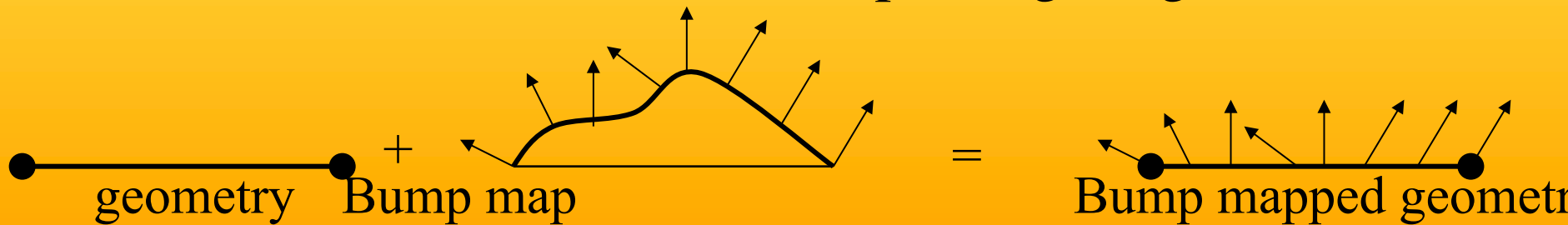
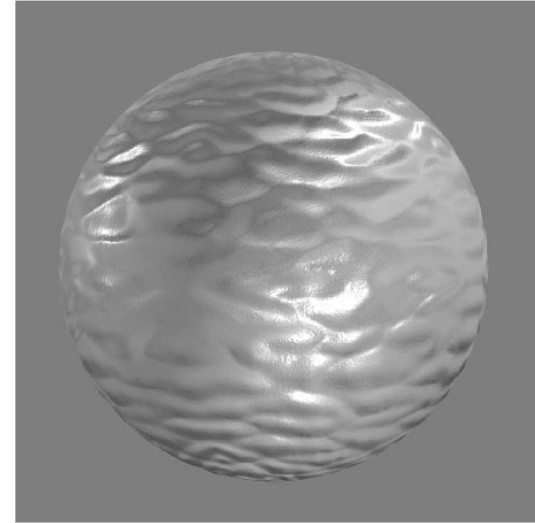
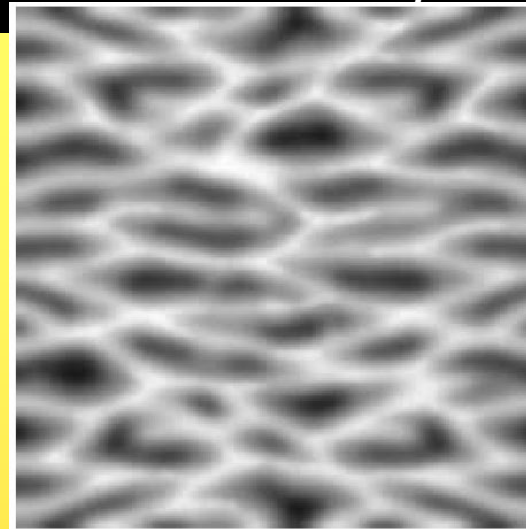
- Simple math: compute reflection vector, \mathbf{r}
- Largest abs-value of component, determines which cube face.
 - Example: $\mathbf{r}=(5,-1,2)$ gives POS_X face
- Divide \mathbf{r} by $\text{abs}(5)$ gives $(u,v)=(-1/5,2/5)$
- Remap from $[-1,1]$ to $[0,1]$, i.e., $((u,v)+(1,1))/2$
- Your hardware does all the work. You just have to compute the reflection vector. (See lab 4)

Example



Bump mapping

- by Blinn in 1978
- Inexpensive way of simulating wrinkles and bumps on geometry
 - Too expensive to model these geometrically
- Instead let a texture modify the normal at each pixel, and then use this normal to compute lighting

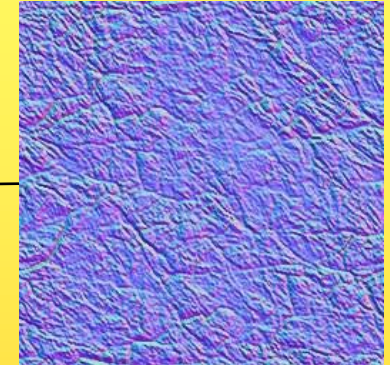
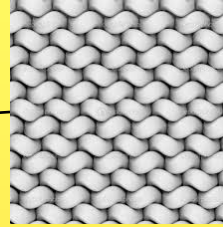


Stores heights: can derive normals

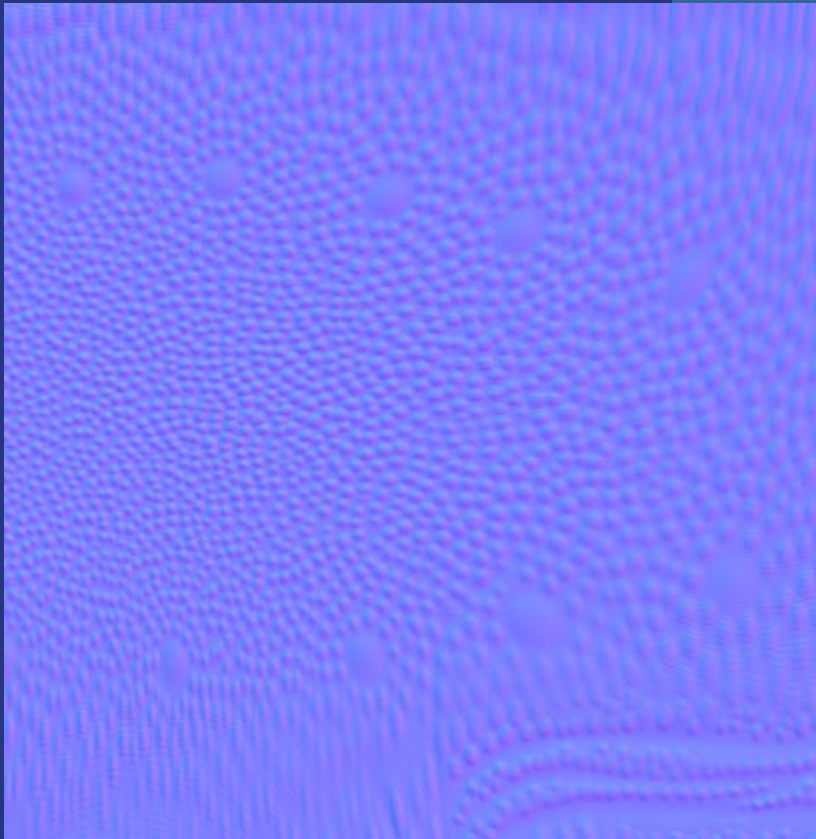
Bump mapping

Storing bump maps:

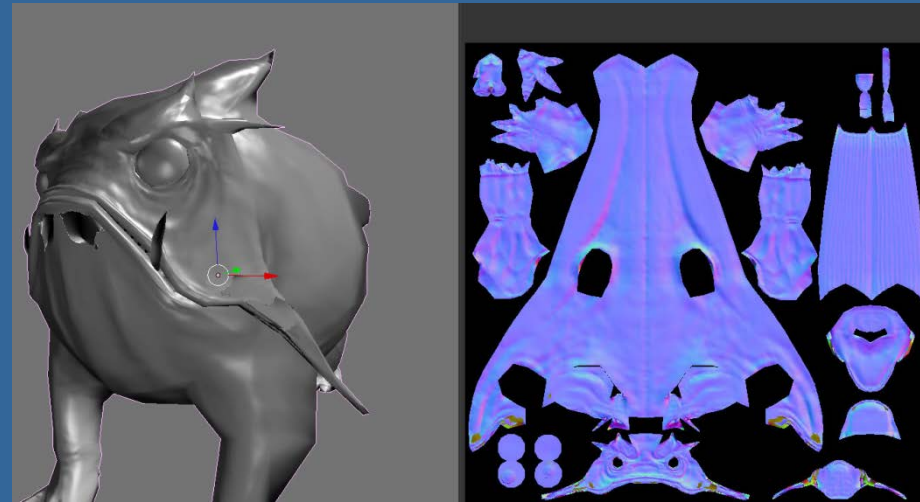
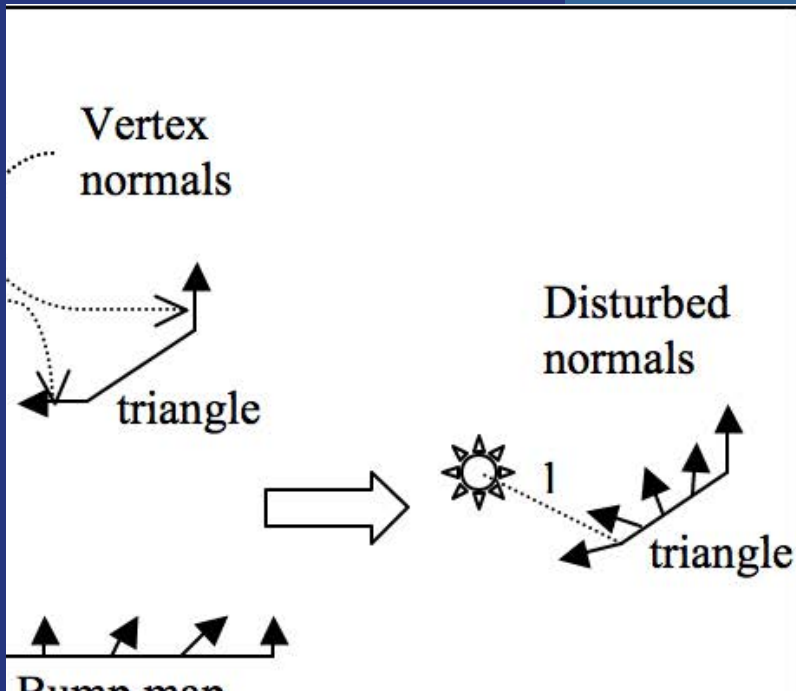
1. as a gray scale image
 2. As Δx , Δy distortions
 3. As normals (n_x, n_y, n_z)
- How store normals in texture (bump map):
 - $\mathbf{n}=(n_x, n_y, n_z)$ are in $[-1,1]$
 - Add 1, mult 0.5: in $[0,1]$
 - Mult by 255 (8 bit per color component)
 - Values can now be stored in 8-bit rgb texture



Bump mapping: example

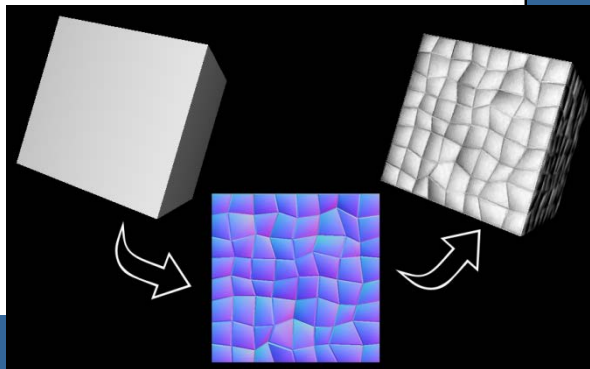


Bump mapping vs Normal mapping



Normal mapping – model space:

- Normals are stored directly in model space. I.e., as including both face orientation plus distortion.



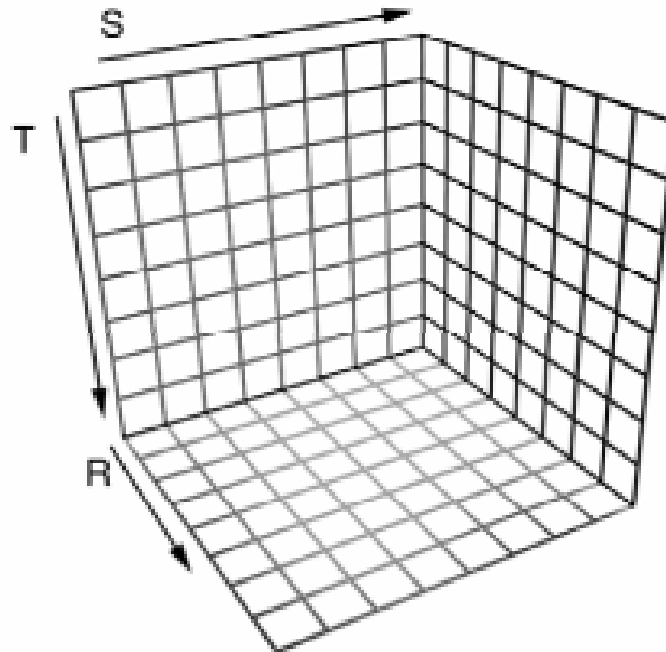
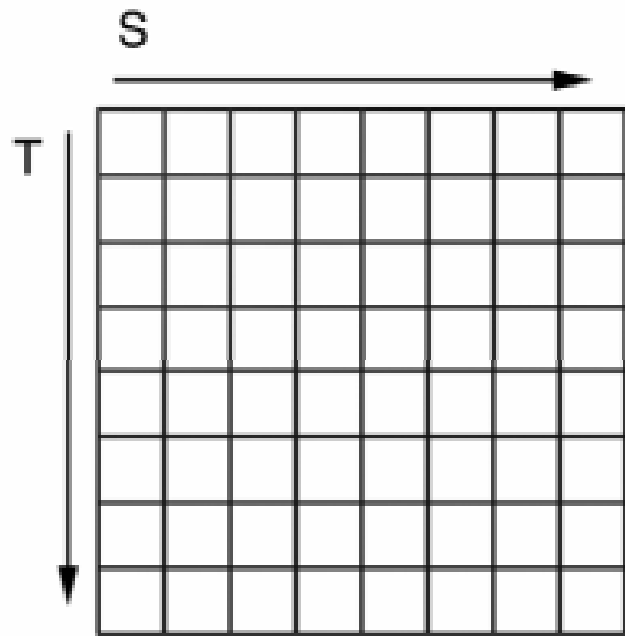
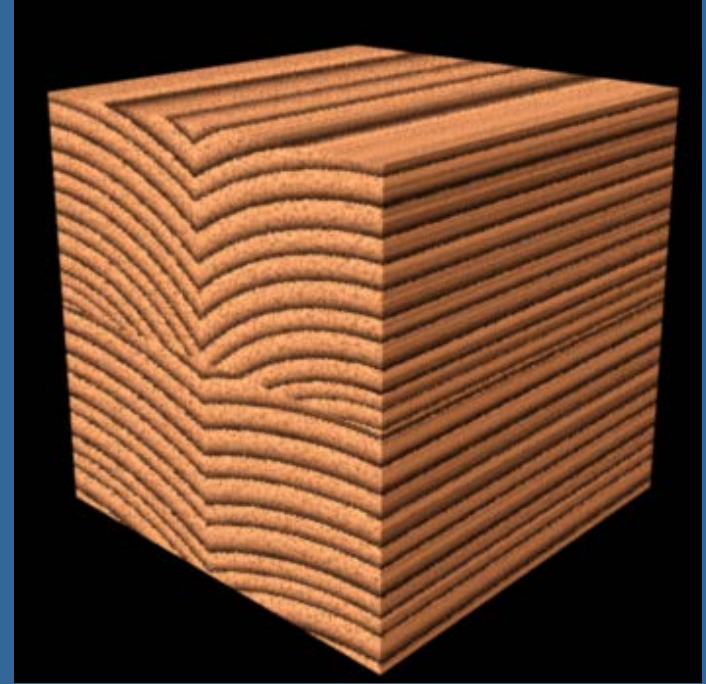
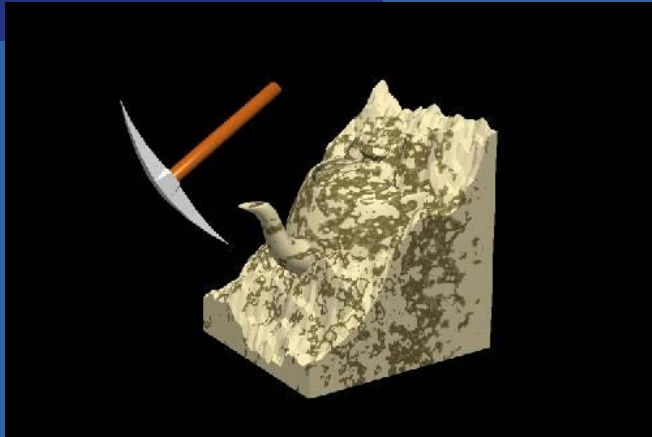
Bump mapping – tangent space:

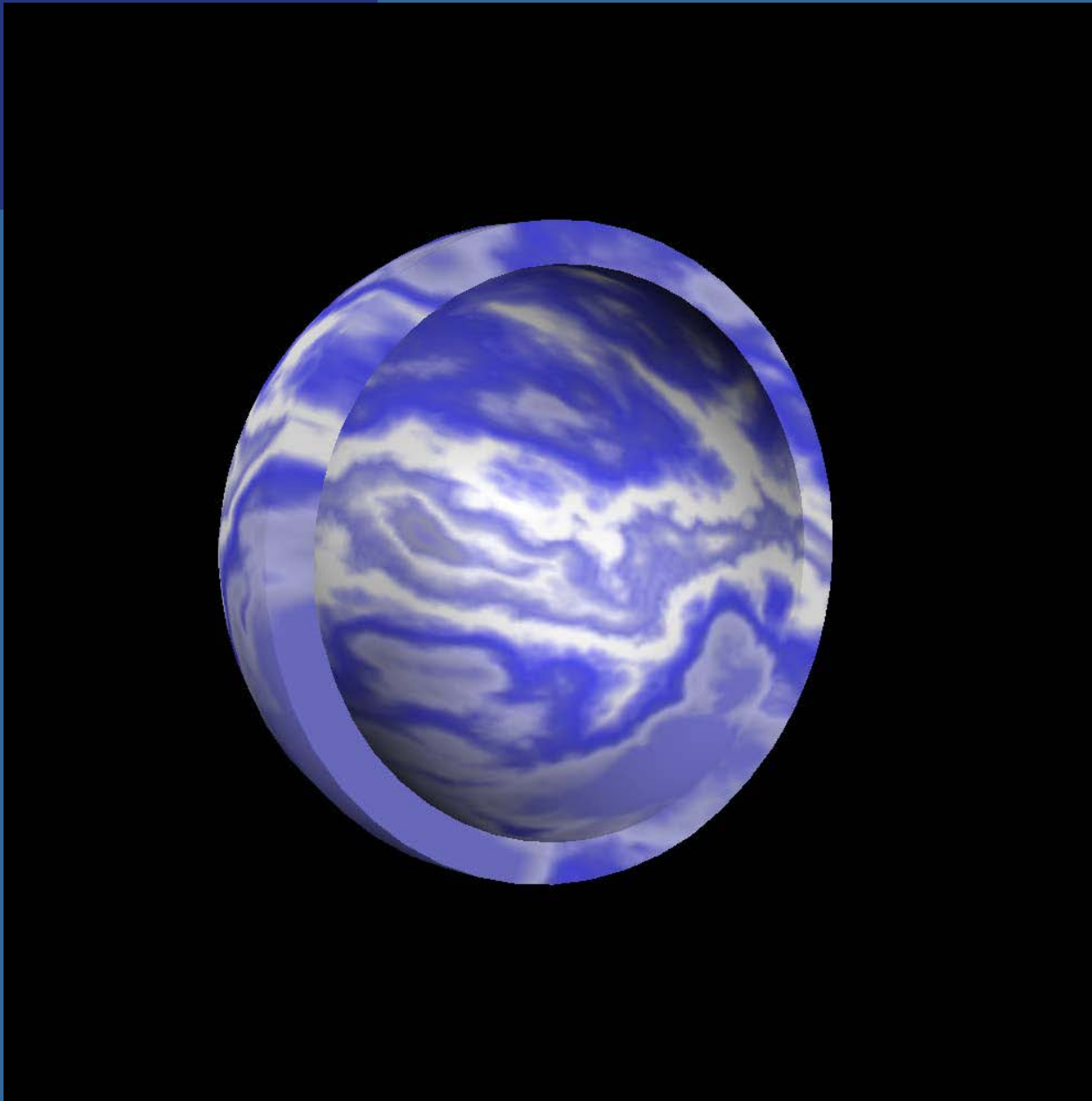
- Normals are stored as distortion of face orientation. The same bump map can be tiled/repeated and reused for many faces with different orientation

More...

- 3D textures:
 - Texture filtering is no longer trilinear
 - Rather quadlinear (linear interpolation 4 times)
 - Enables new possibilities
 - Can store light in a room, for example
- Displacement Mapping
 - Offsets the position per pixel or per vertex
 - Offsetting per vertex is easy in vertex shader
 - Offsetting per pixel is architecturally hard
 - Cannot be done in fragment shader
 - Can be done using Geometry Shader (e.g. Direct3D 10) by ray casting in the displacement map

2D texture vs 3D texture





From http://www.ati.com/developer/shaderx/ShaderX_3DTextures.pdf

Precomputed Light fields

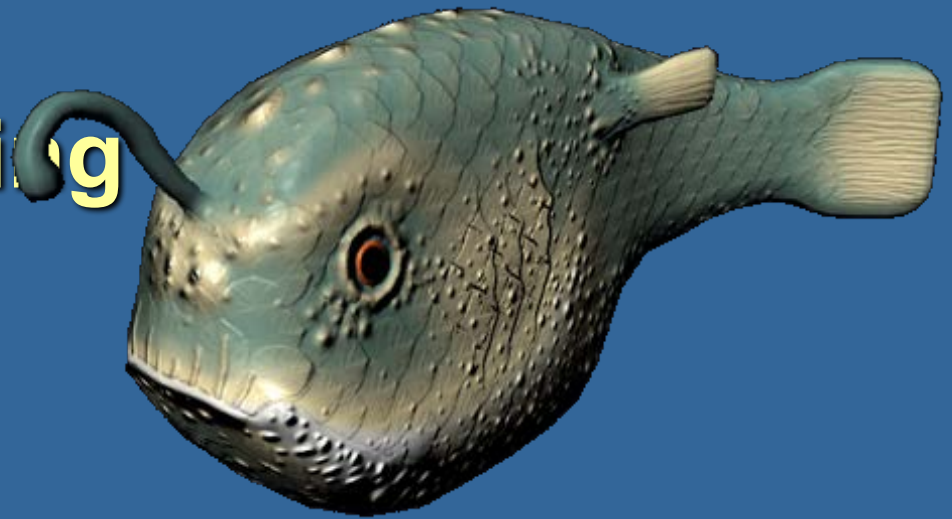


Max Payne 2 by Remedy Entertainment

Samuli Laine and Janne Kontkanen

Displacement Mapping

- Uses a map to displace the surface at each position
- Can be done with a Geometry Shader



Vertex Shader

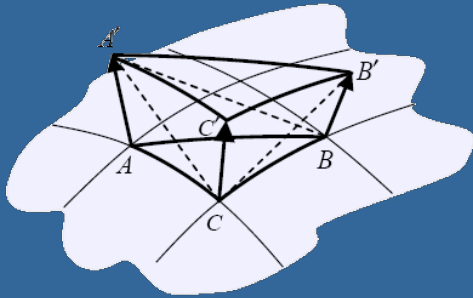
Geometry Shader

Pixel Shader

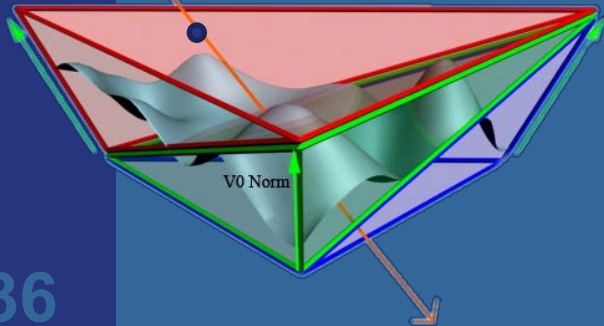
Geometry Shader Example

Generalized Displacement Maps

- Step 0: Process Vertices (VS)
- Step 1: Extrude Prisms (GS)



 Step 2: Raytrace! (PS)



Rendering to Texture

(See also
Lab 5)

```
/**
// Create a Frame Buffer Object (FBO) that we first render to and then use as a texture
**/
glGenFramebuffers(1, &frameBuffer); // generate framebuffer id
glBindFramebuffer(GL_FRAMEBUFFER, frameBuffer); // following commands will affect "frameBuffer"

// Create a texture for the frame buffer, with specified filtering, rgba-format and size
glGenTextures( 1, &texFrameBuffer );
glBindTexture( GL_TEXTURE_2D, texFrameBuffer ); // following commands will affect "texFrameBuffer"
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
glTexImage2D( GL_TEXTURE_2D, 0, 4, 512, 512, 0, GL_RGBA, GL_UNSIGNED_BYTE, NULL );

// Create a depth buffer for our FBO
glGenRenderbuffers(1, &depthBuffer); // get the ID to a new Renderbuffer
glBindRenderbuffer(GL_RENDERBUFFER, depthBuffer);
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT, 512, 512);

// Set rendering of the default color0-buffer to go into the texture
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D,
texFrameBuffer, 0);
glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER,
depthBuffer); // Associate our created depth buffer with the FBO
```

**Or simply render to back-buffer and copy into texture
using command: `glCopyTexSubImage ()`. But is slower.**³⁷

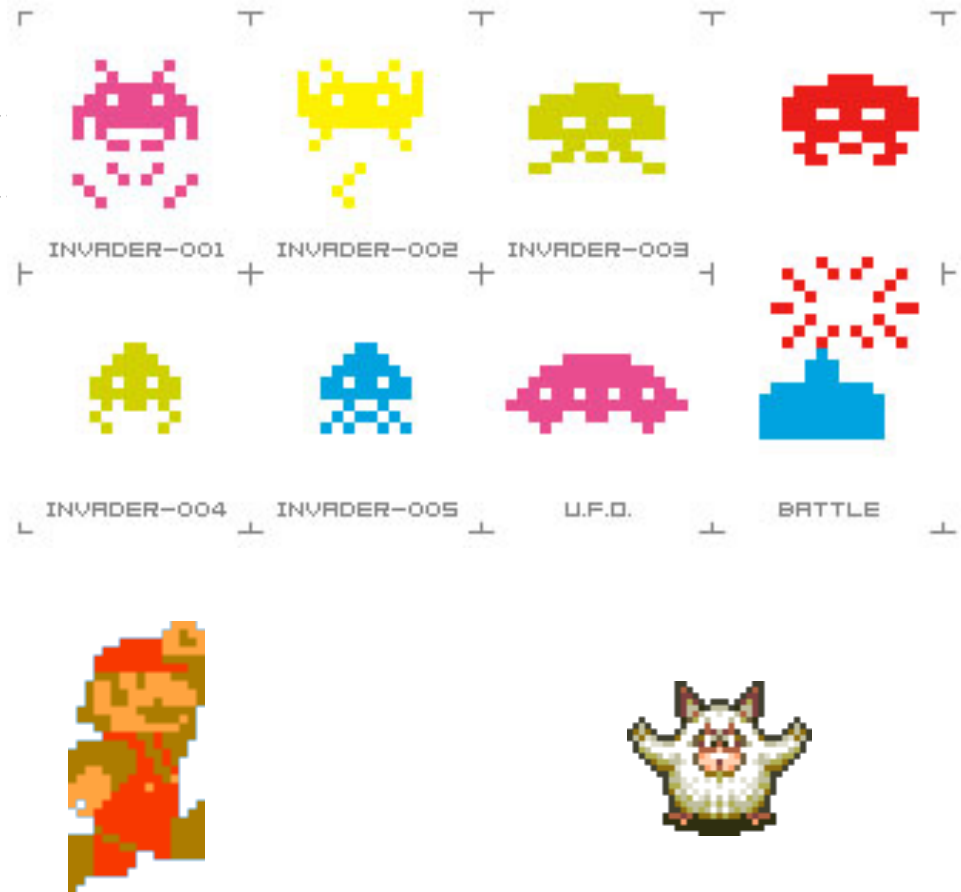
Sprites

Sprites (=älvor) was a technique on older home computers, e.g. VIC64. As opposed to billboards sprites does not use the frame buffer. They are rasterized directly to the screen using a special chip. (A special bit-register also marked colliding sprites.)

```
GLbyte M[64]=
{ 127,0,0,127, 127,0,0,127,
  127,0,0,127, 127,0,0,127,
  0,127,0,0, 0,127,0,127, 0,127,0,1
  0,127,0,0,
  0,0,127,0, 0,0,127,127, 0,0,127,1
  0,0,127,0,
  127,127,0,0, 127,127,0,127,
  127,127,0,127, 127,127,0,0};
```

```
void display(void) {
  glClearColor(0.0,1.0,1.0,1.0);
  glClear(GL_COLOR_BUFFER_BIT);
  glEnable (GL_BLEND);
  glBlendFunc (GL_SRC_ALPHA,
               GL_ONE_MINUS_SRC_ALPHA);
  glRasterPos2d(xpos1,ypos1);
  glPixelZoom(8.0,8.0);
  glDrawPixels(width,height,
              GL_RGBA, GL_BYTE, M);

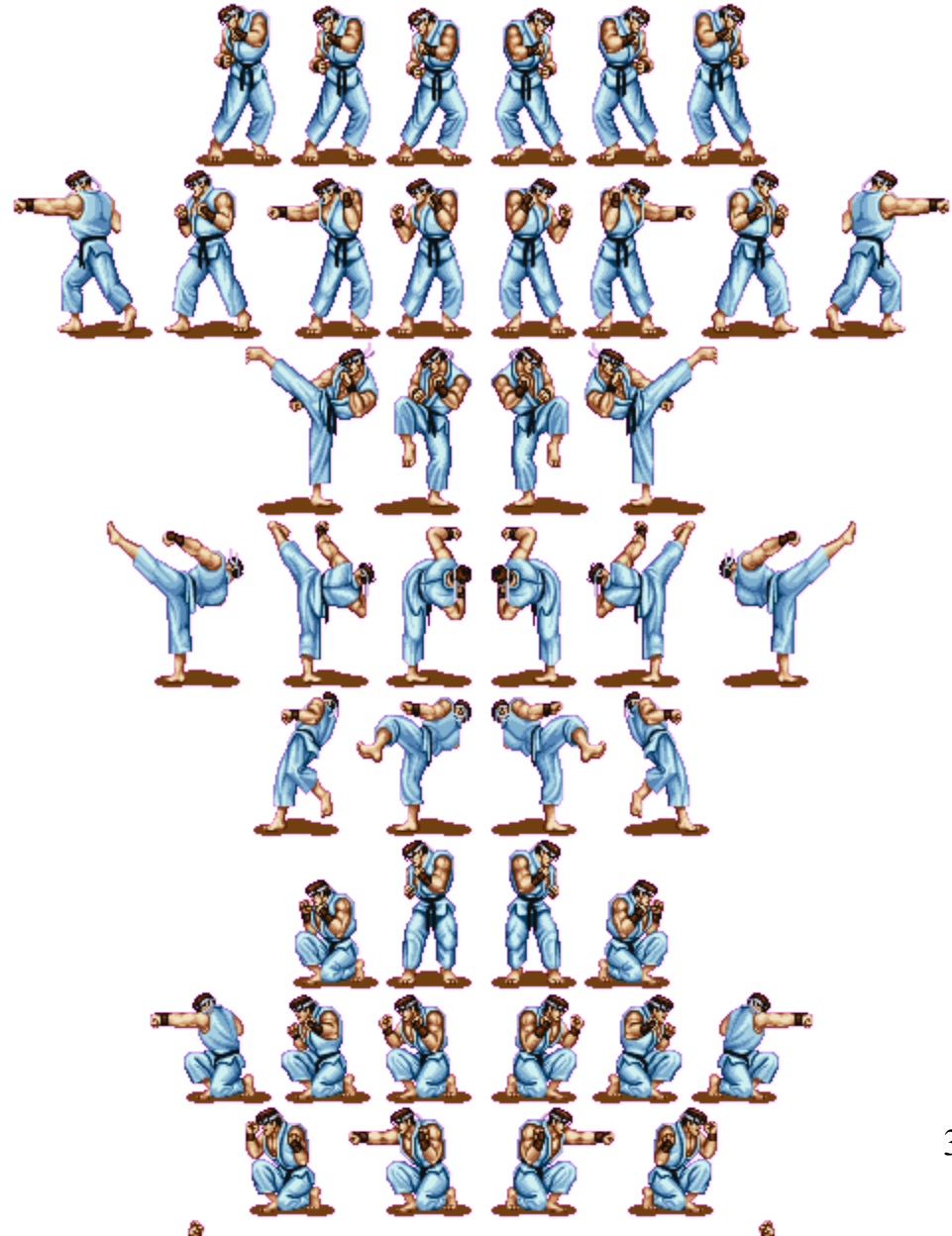
  glPixelZoom(1.0,1.0);
  SDL_GL_SwapWindow // "Swap buffers"
}
```



Sprites

Animation Maps

The sprites for Ryu in Street Fighter:

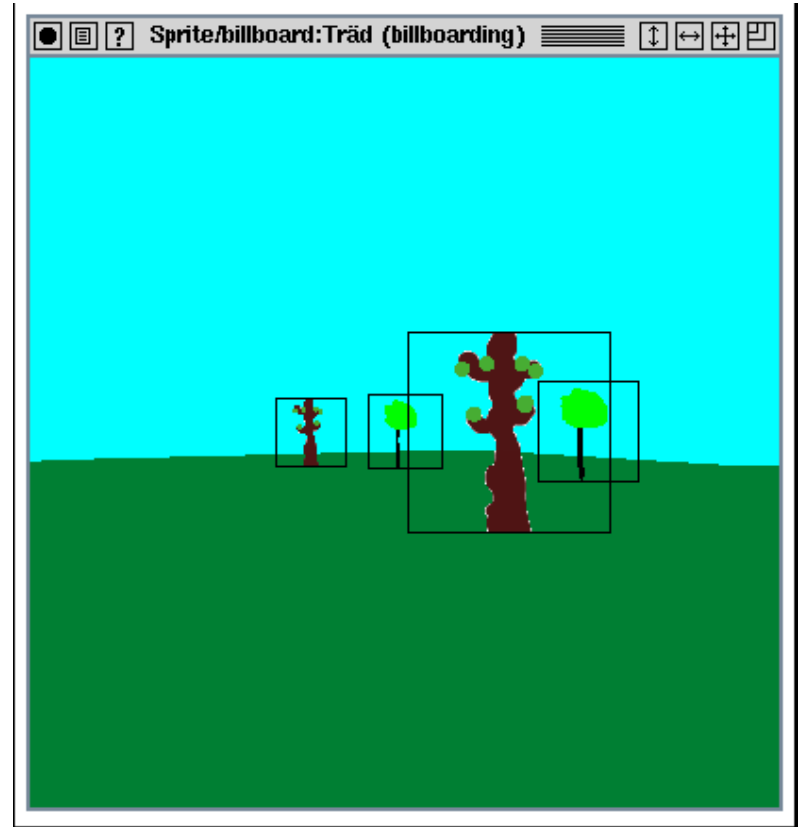
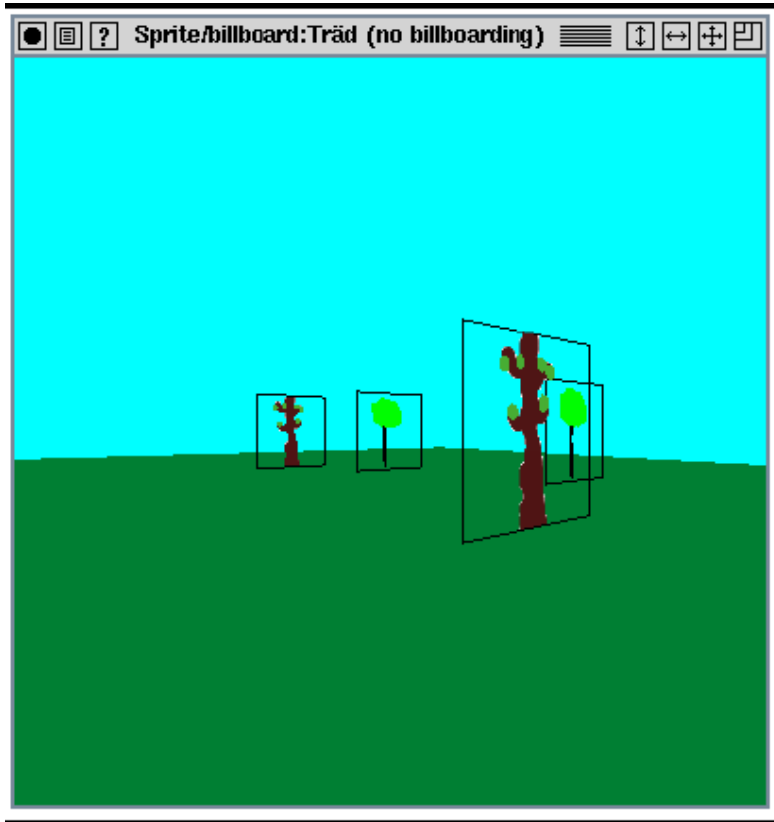


Billboards

- 2D images used in 3D environments
 - Common for trees, explosions, clouds, lens-flares



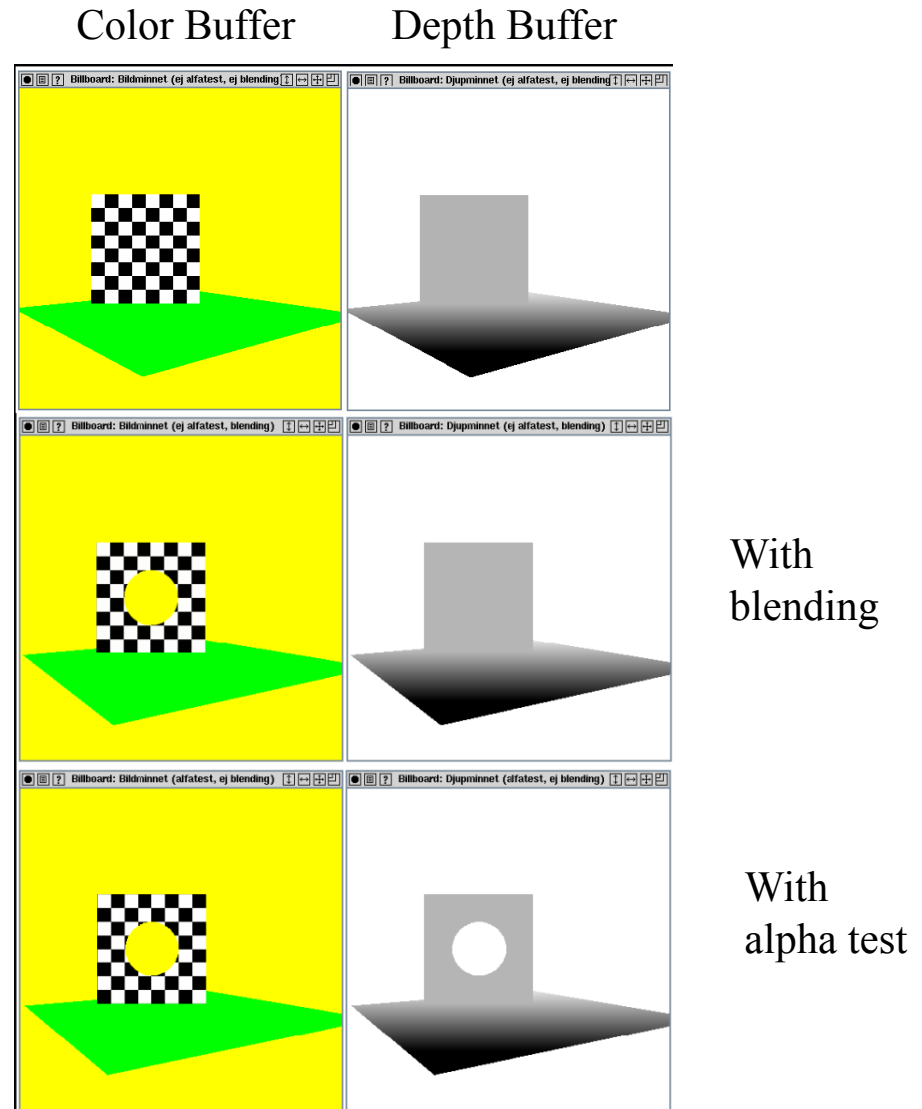
Billboards



- Rotate them towards viewer
 - Either by rotation matrix (see OH 288), or
 - by orthographic projection

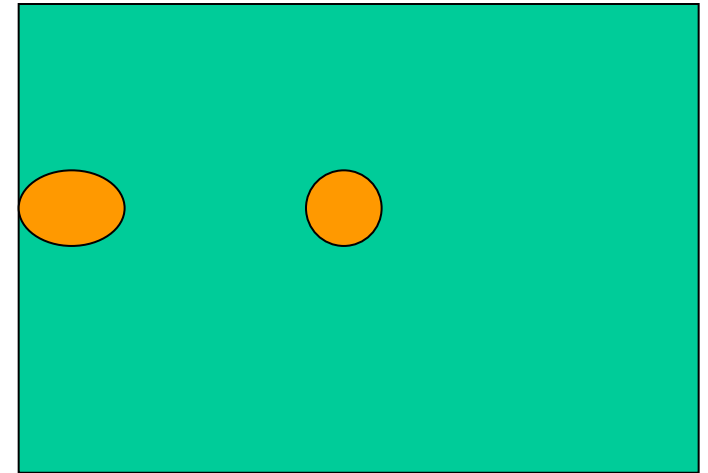
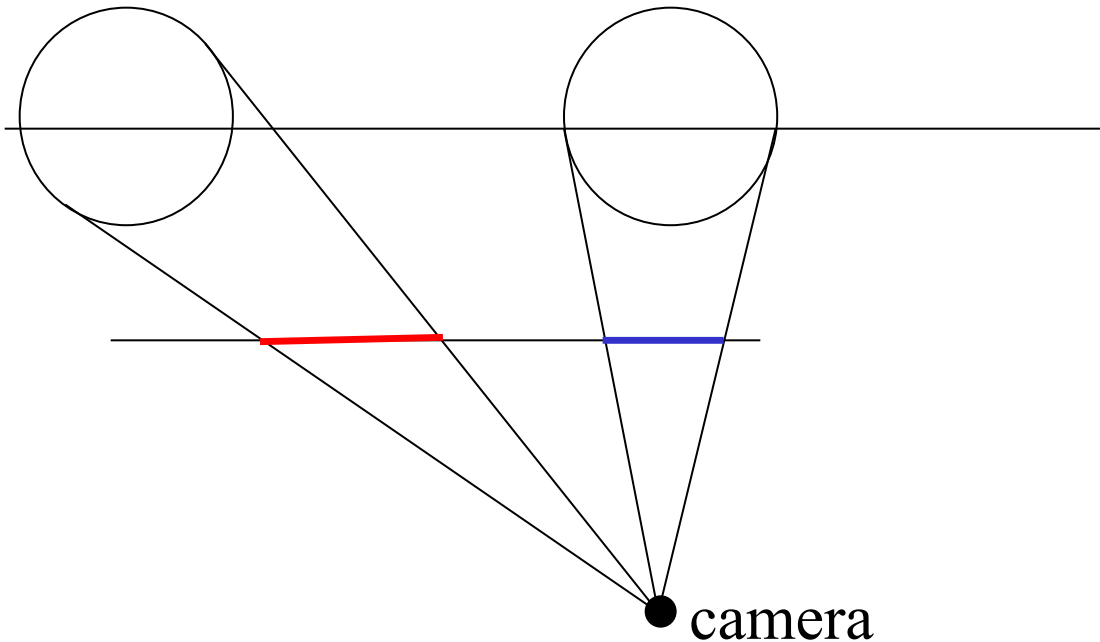
Billboards

- Fix correct transparency by blending AND using alpha-test
 - In fragment shader:
`if (color.a < 0.1) discard;`
- Or: sort back-to-front and blend
 - (Depth writing could then be disabled to gain speed)
 - `glDepthMask(0);`



Perspective distortion

- Spheres often appear as ellipsoids when located in the periphery. Why?



Exaggerated example

If our eye was placed at the camera position, we would not see the distortion. We are often positioned way behind the camera₄₃

Which is preferred?

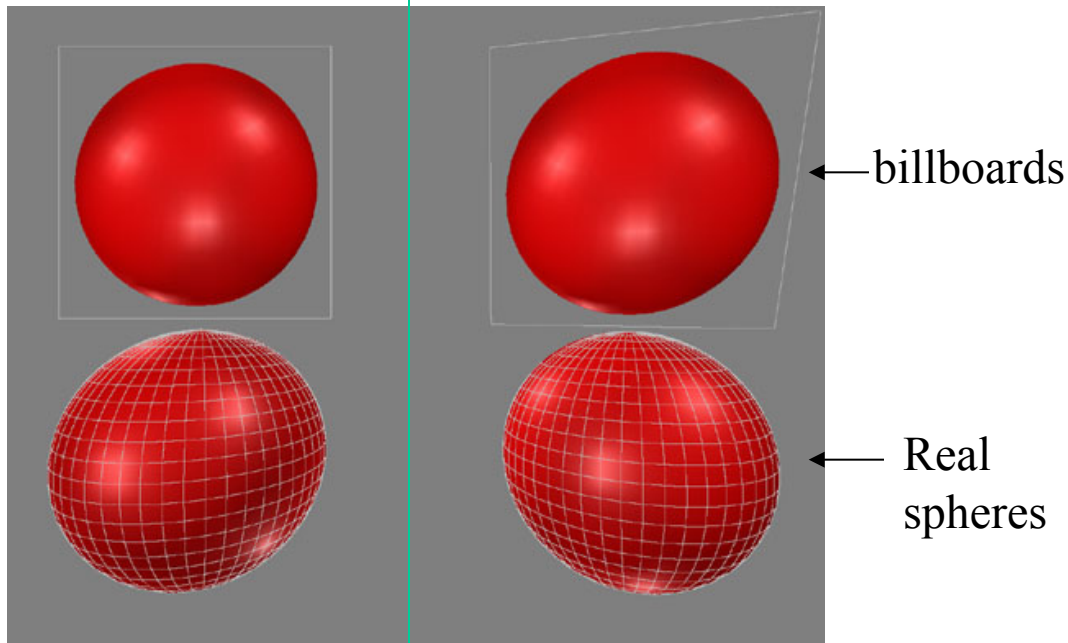
view plane aligned

viewpoint oriented

view plane

viewpoint

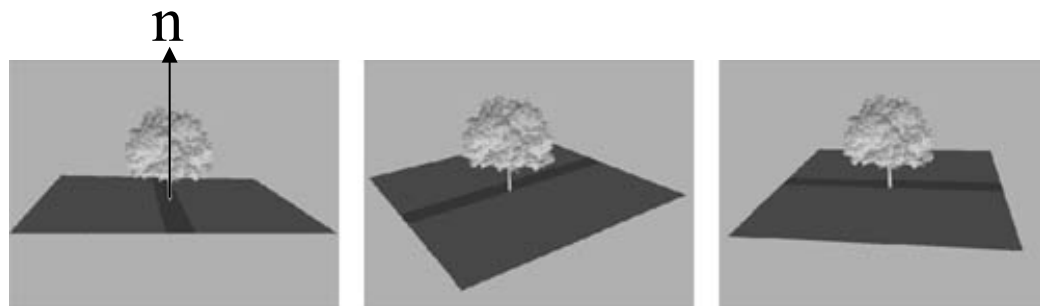
This is the result



Actually, viewpoint oriented is often preferred since it most closely resembles the result using standard 3D geometry



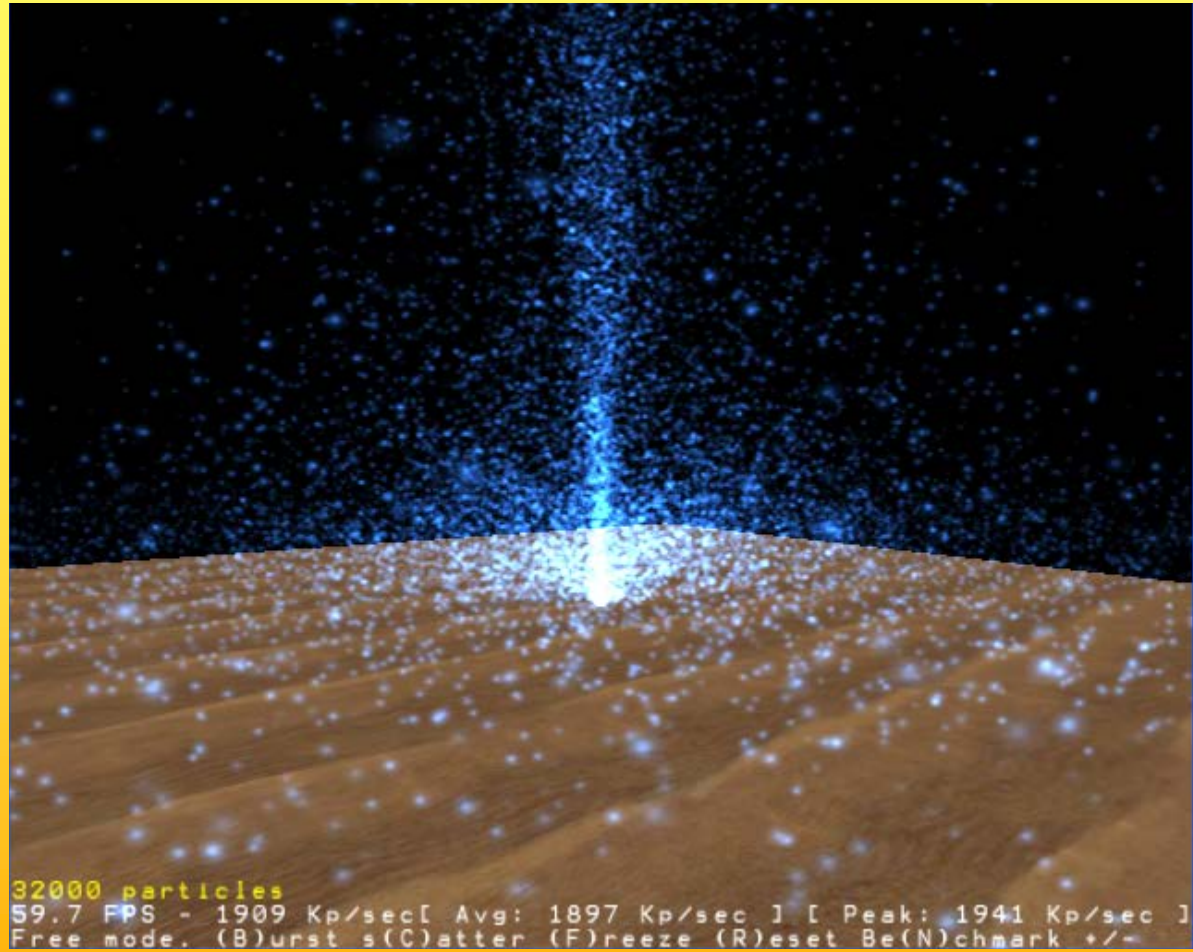
Also called *Impostors*



axial billboarding

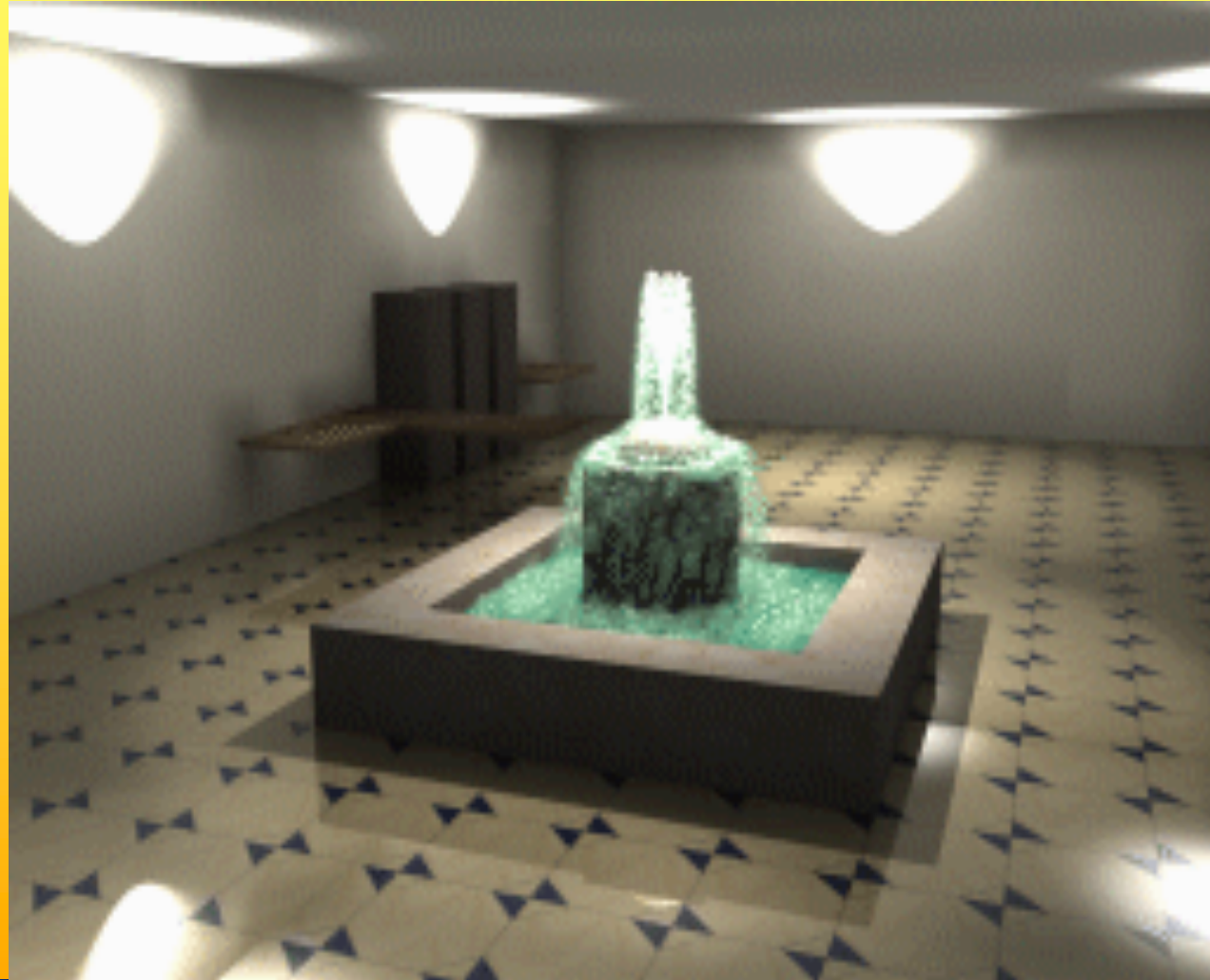
The rotation axis is fixed and disregarding the view position

Particle system



Particles

Partikelsystem

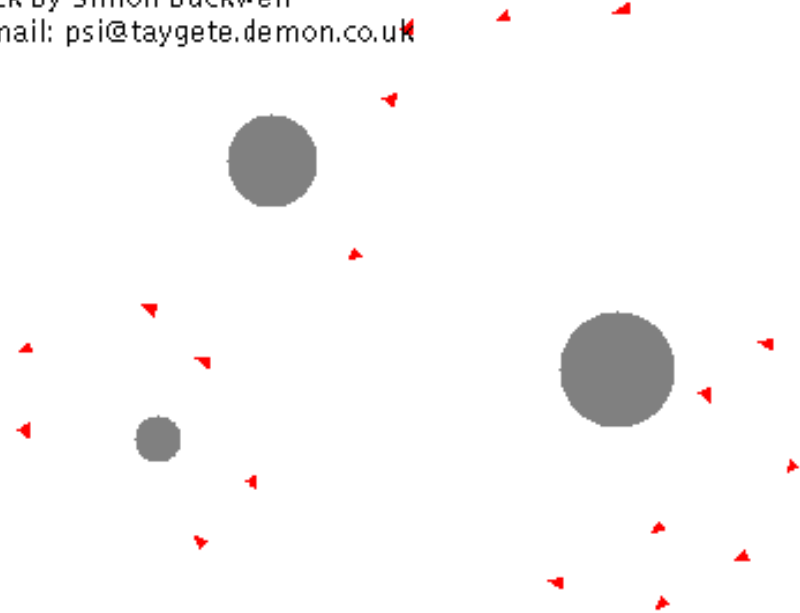




Particle Systems

- Boids (flock of birds), see OH 230
 - 3 rules:
 1. Separation: Avoid obstacles and getting too close to each other
 2. Alignment (strive for same speed and direction as nearby boids)
 3. Cohesion: steer towards center of mass of nearby boids

Flock By Simon Buckwell
e-mail: psi@taygete.demon.co.uk



What's most important?

Texturing:

- Filtering: magnification, minification
 - Mipmaps + their memory cost
 - How compute bilinear/trilinear filtering
 - #texel accesses
 - Anisotropic filtering
- Environment mapping – cube maps. How compute lookup.
- Bump mapping
- 3D-textures – what is it?
- Sprites
- Billboards/Impostors, viewplane vs viewpoint oriented, axial billboards, how to handle depth buffer for fully transparent texels.
- Particle systems