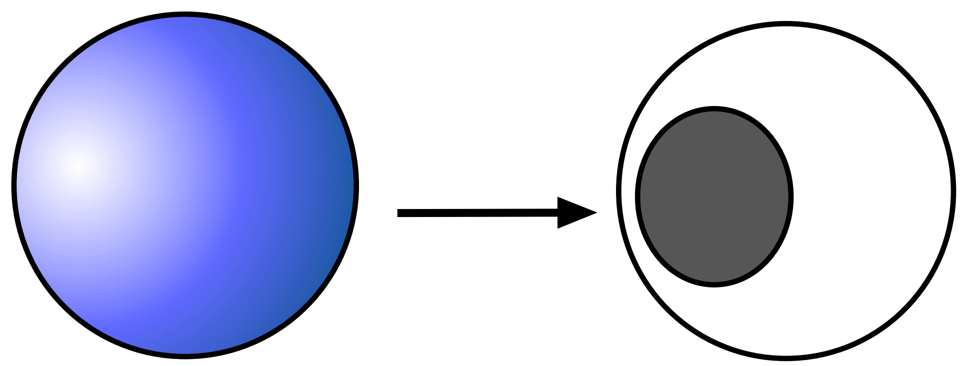
Shading and shadows

Shading will give you light variations due to shape, and the back side of objects will be shadowed.

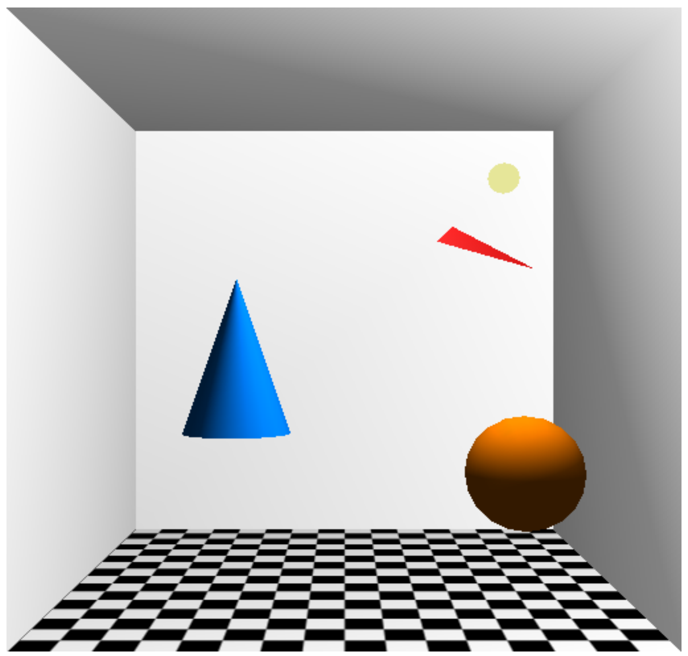
BUT, this will not produce shadows on one object cast by another object!

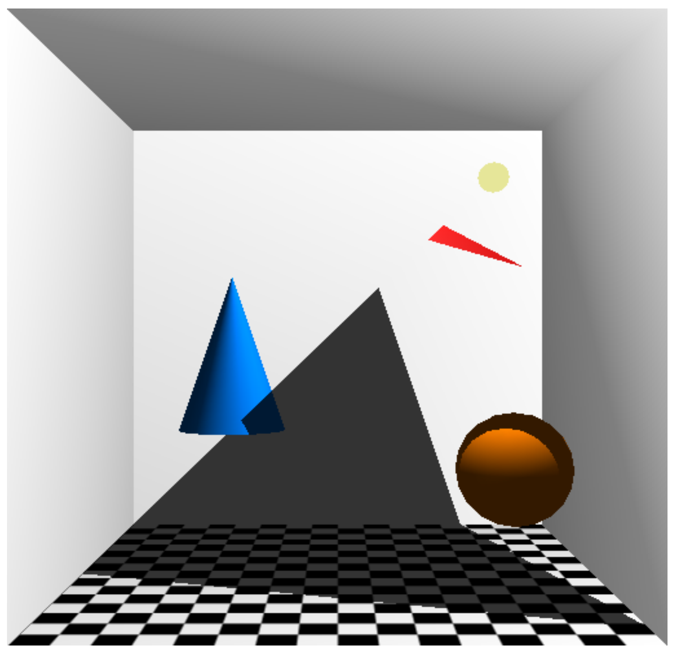


Shading and shadows

Local shading is easy (with simple light models)

Shadows are hard

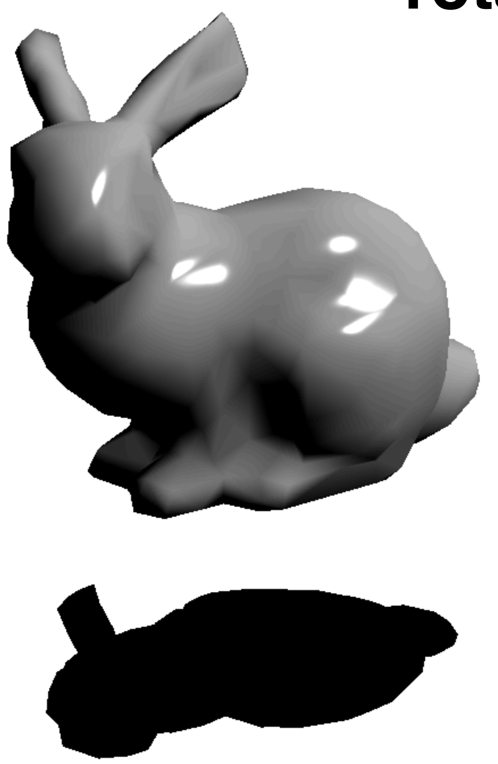




Simple shadows

Easiest: Linear planar shadow

Flatten object, paint black (optionally transparently), rotate and translate to appropriate position



Example:

```
Translate to surface
Scale by (1, 0, 1)
Rotate as desired
Draw model in black
```

Position after flattening

Flatten

No light or texture

Advanced shadows

Planar projective shadows

Shadow volumes

Shadow mapping

Soft shadows

-> Advanced course (TSBK03)

Also: Natural part of ray-tracing and radiosity

Surface detail

Shading:
takes away the surface detail of the polygons

Texture mapping and other mappings:
add the surface detail that we really want

Surface mapping techniques

Texture mapping
Billboards
Bump mapping
Light mapping
Environment mapping

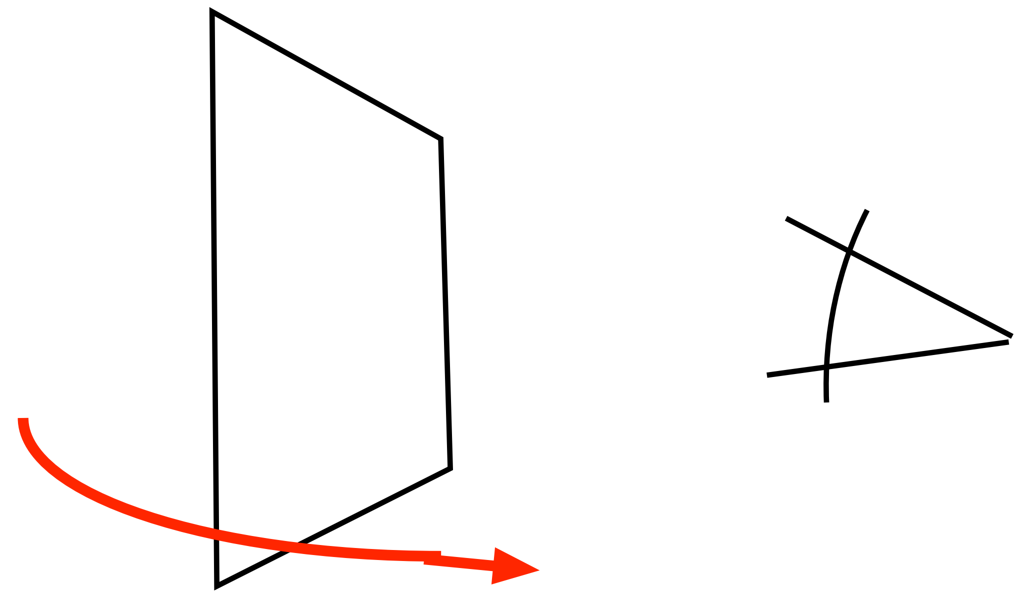
Texture mapping

In common use

Special support by the GPU hardware - not just a memory access

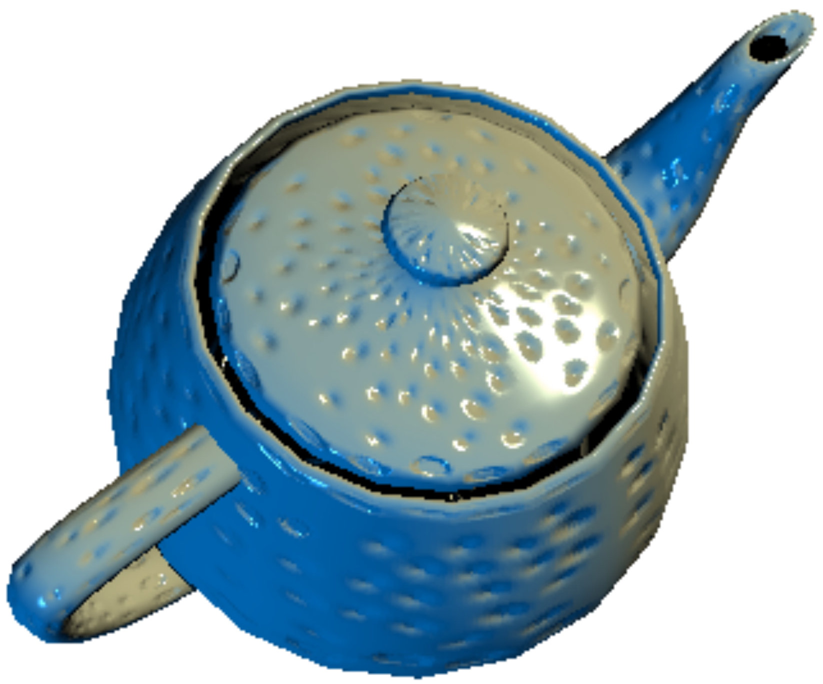
Billboards

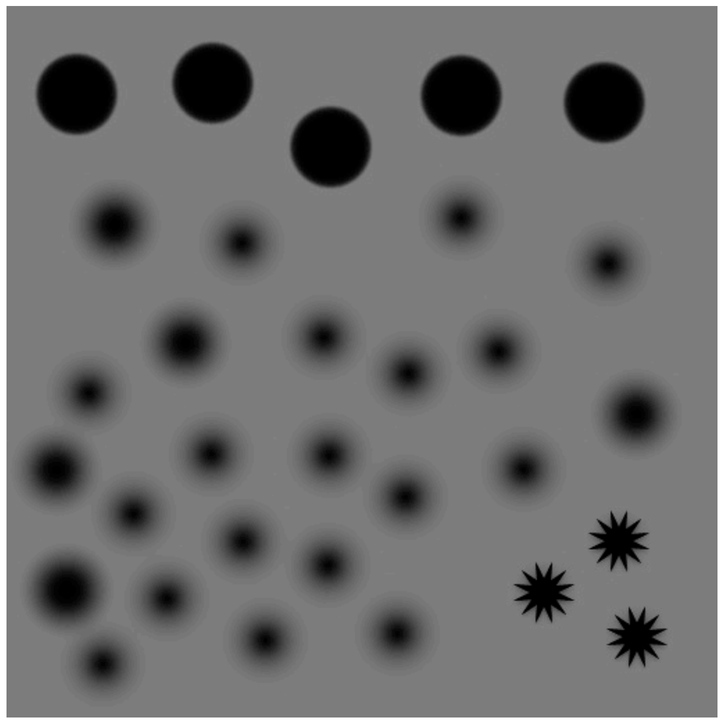
A billboard is a texture mapped polygon, which always faces the viewer



Bump mapping

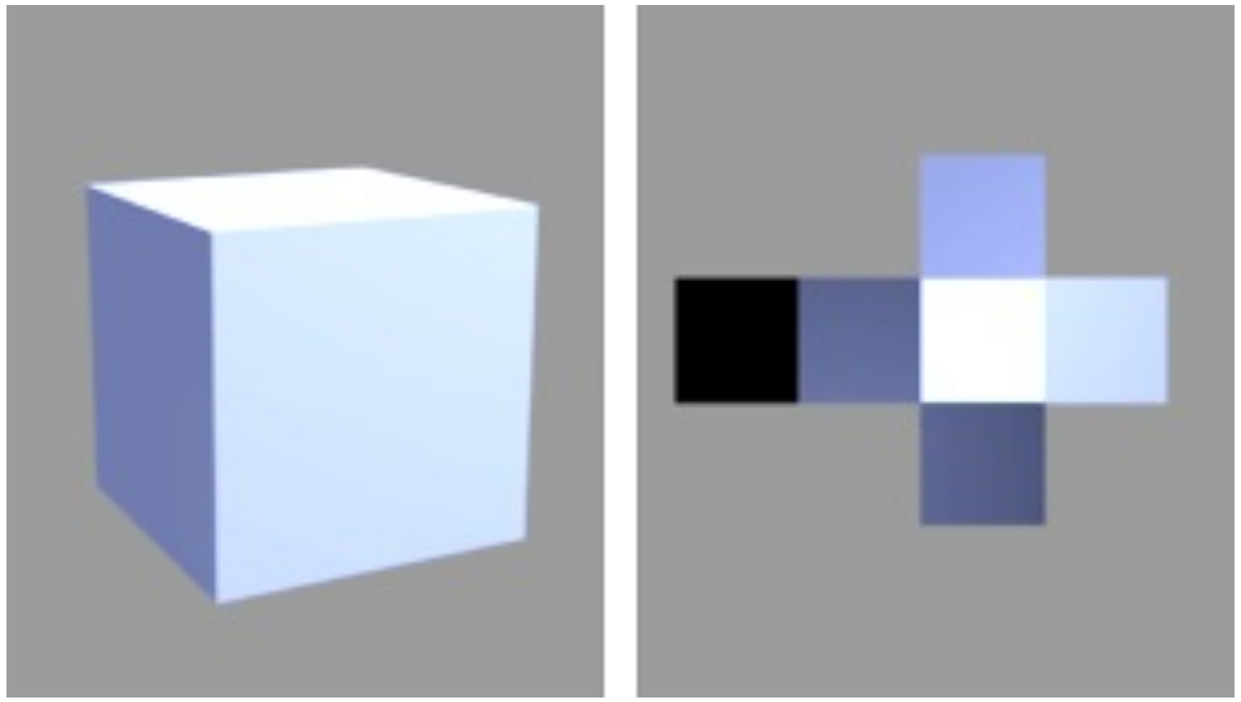
Simulates surface structure by manipulating the normal vector





Light mapping

Applies pre-calculated light to surfaces



(Image from Wikipedia)

Environment mapping

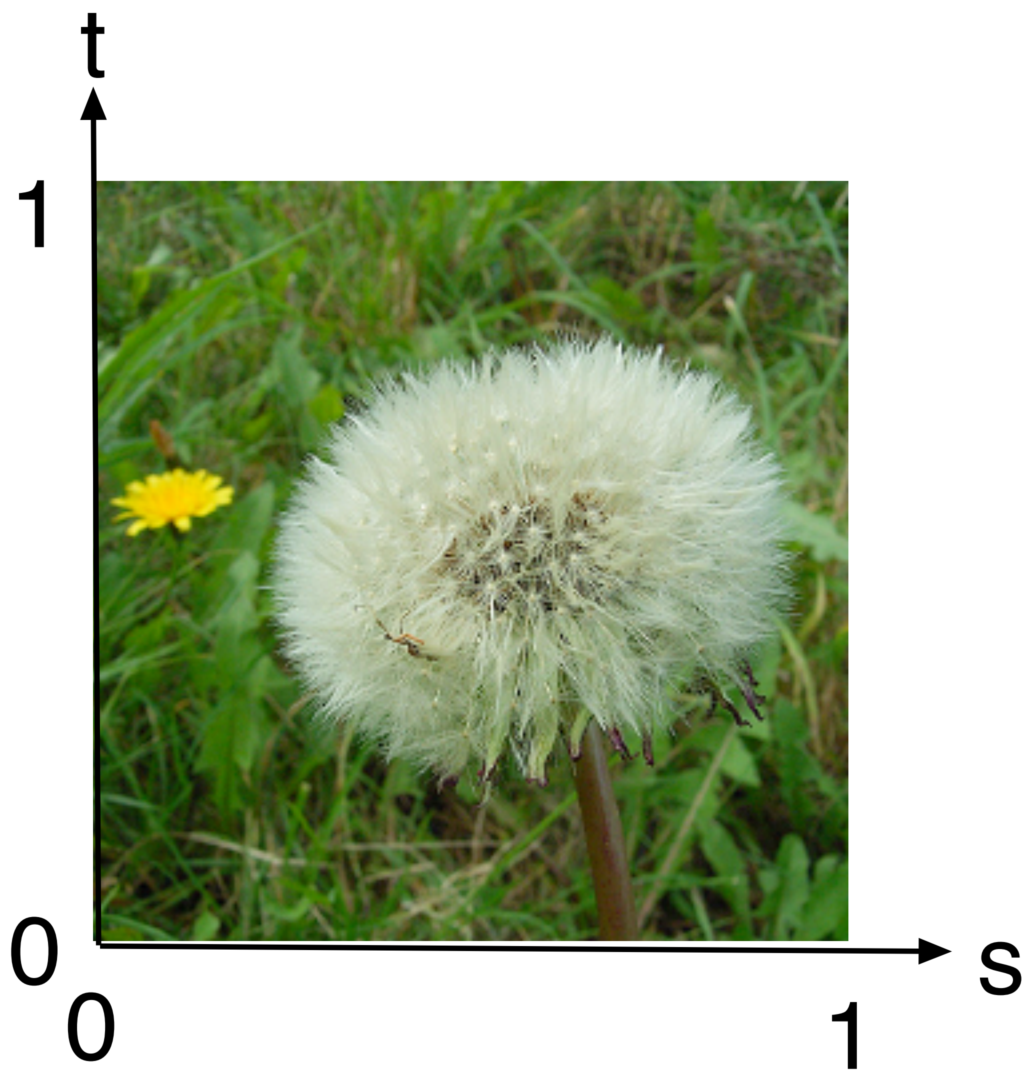
Maps an pre-rendered image as a reflection in the object

Texture mapping

“Wrap” a specified part of “texture space” onto an object

Consider the texture to be an elastic wrapping

Texture space



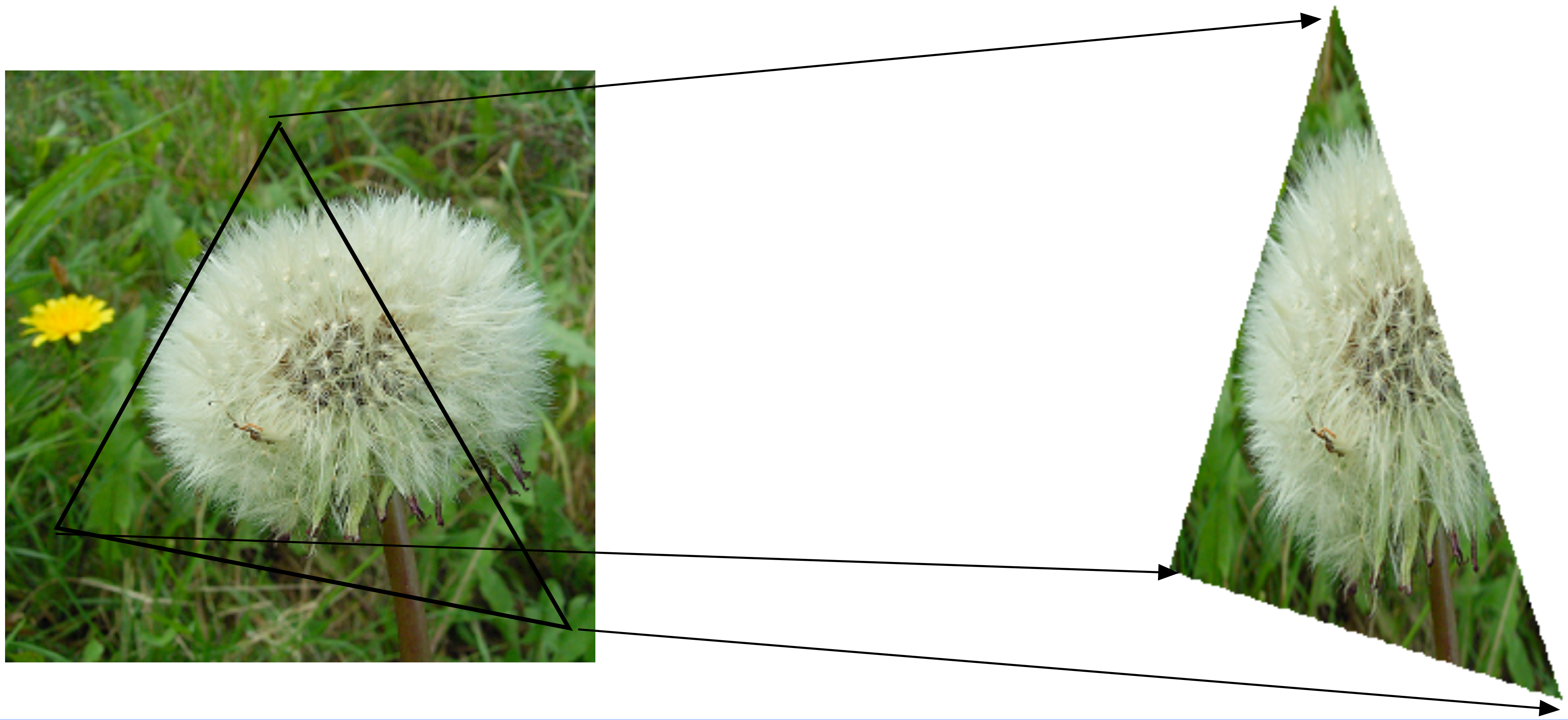
Texture = image used for texture mapping

Built from ”texels”

Texture space is usually 2-dimensional, (s, t), with textures defined in [0, 1]

Mapping from texture to surface

Each vertex has a texture coordinate, interpolate between, look up texture with interpolated coordinates.



Texture coordinates

Texture coordinates often included in models.

loadobj.c supports texture coordinates.

Pass as attribute array to vertex shader.

Interpolate from vertex to fragment shader.

Example:
Procedural texture

Texture generated by fragment shader!

- **Vertex shader passes on texture coordinates**
- **Texture coordinates are used in a texture generating function in the fragment shader**

Simpler than you might think!

Procedural texture, Vertex shader

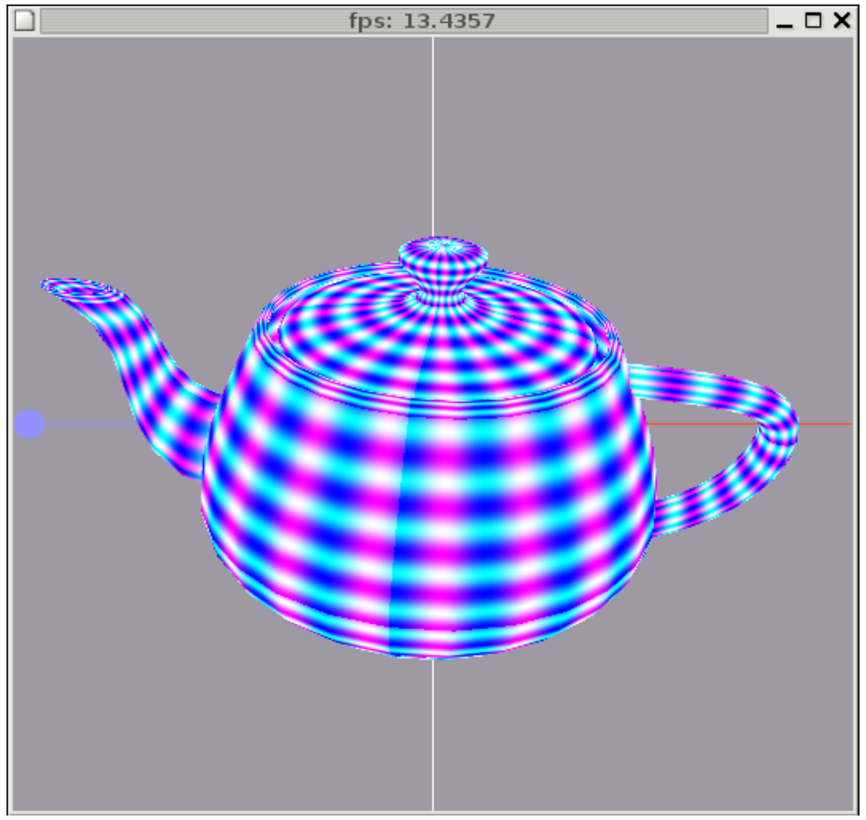
```
uniform mat4 proj;
uniform mat4 view;
out vec2 texCoord;
in vec2 inTexCoord;

void main()
{
gl_Position = proj * view * gl_Vertex;
texCoord = inTexCoord;
}
```

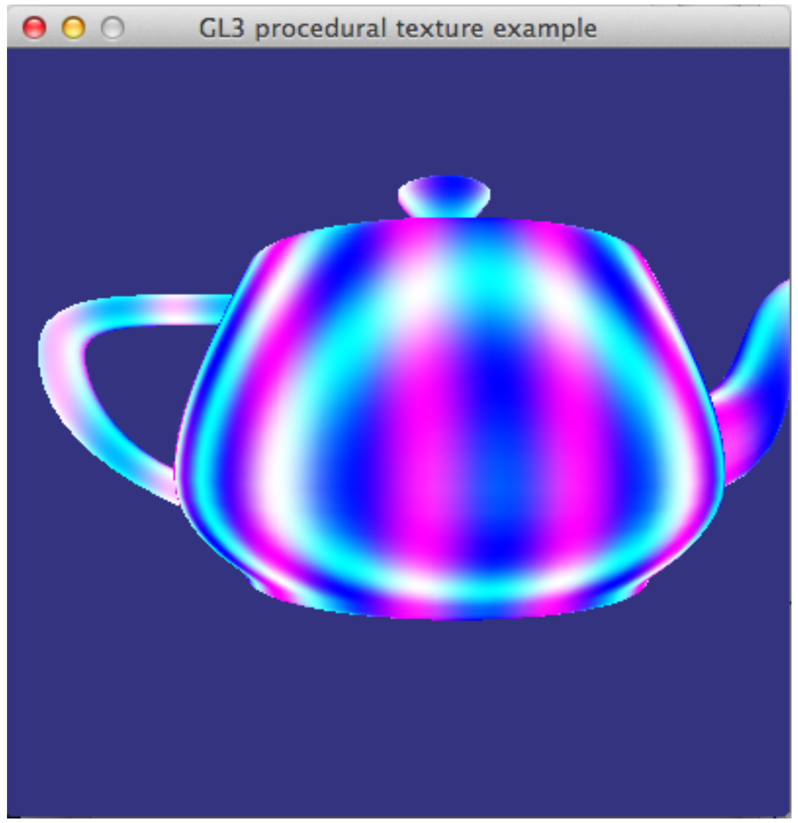
Procedural texture, Fragment shader

```
in vec2 texCoord;
out outColor;

void main()
{
float a = sin(texCoord.s*30)/2+0.5;
float b = sin(texCoord.t*30)/2+0.5;
outColor = vec4(a, b, 1.0, 0.0);
}
```

Procedural texture

Result





Texture objects

Referring to already loaded textures

glGenTextures(...);
reserves texture numbers, making them available to use

glBindTexture(...);
makes a texture the current one

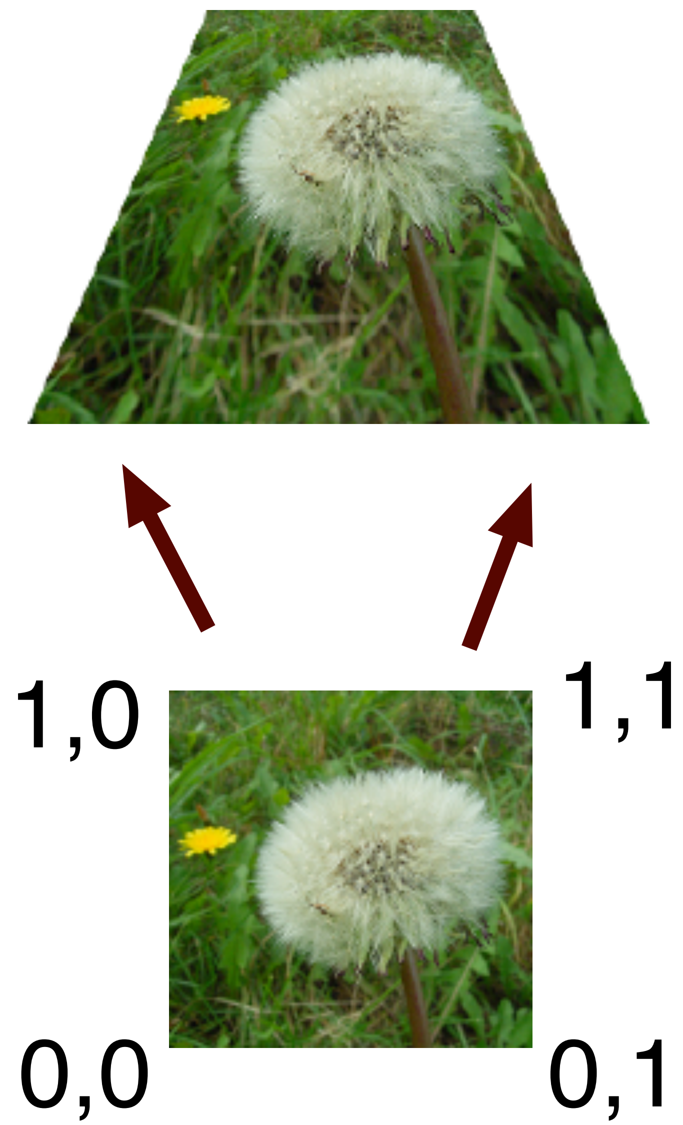
glTexImage2D(...);
loads a texture for the current texture number

A textured polygon

vertex list (x, y, z)

texture coordinate list (s, t)

index list common for both



Texture data

In order to use predefined texture data, they should be communicated from OpenGL!

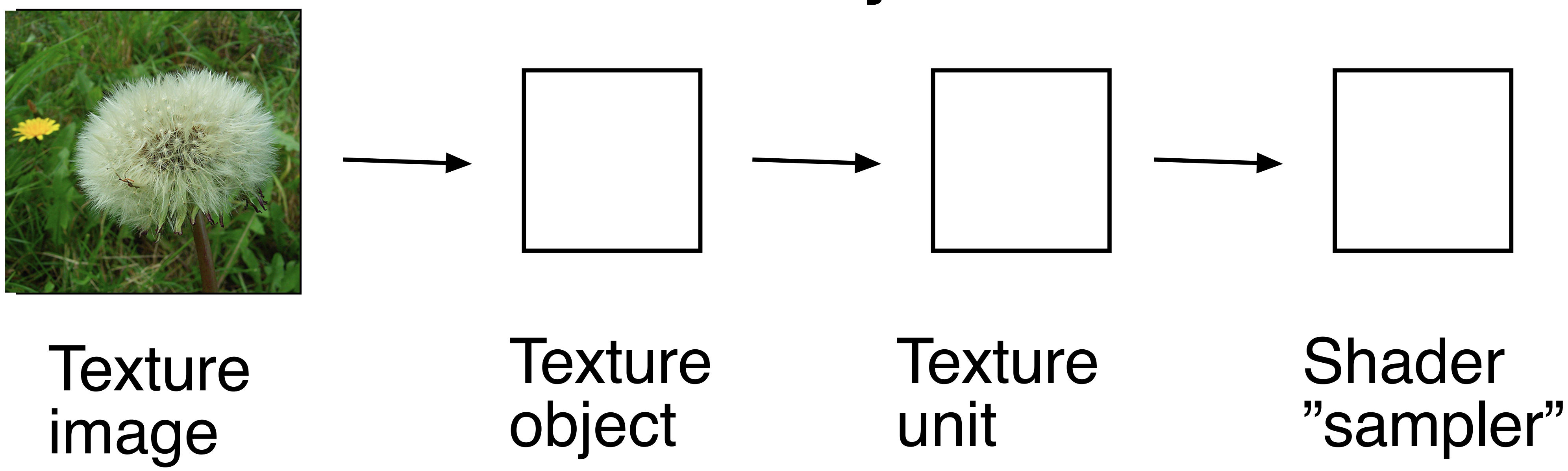
This is done by a “uniform”, a variable that can not be changed within a primitive.

“samplers”: pre-defined type for referencing texture data

Texture units

Textures are bound to ”texture units”, hardware resources for looking up textures

The shader uses the texture unit ID, not the texture object!



Texture access

Example:

```
uniform sampler2D tex;
out vec4 outColor;
in vec2 texCoord;

void main()
{
outColor = texture(tex, texCoord);
}
```

texture() performs texture access

Example: texture, uniform sampler:

```
GLuint tex;

glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, tex);
loc = glGetUniformLocation(PROG, "tex");
glUniform1i(loc, 0);
```

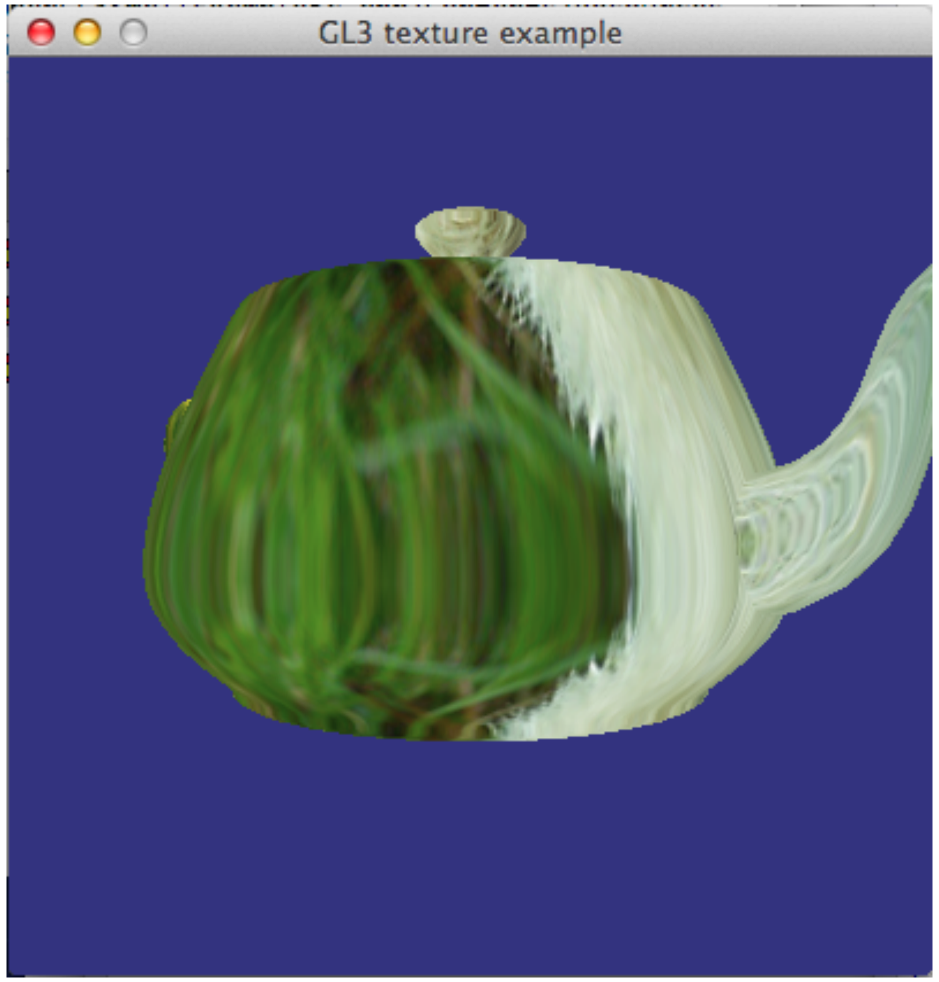
zero to glUniform1i = texture unit number!

Use in shader:

```
uniform sampler2D tex;

vec3 texval = vec3(texture(tex, gl_TexCoord[0].st));
```

Texture loaded from image

Result

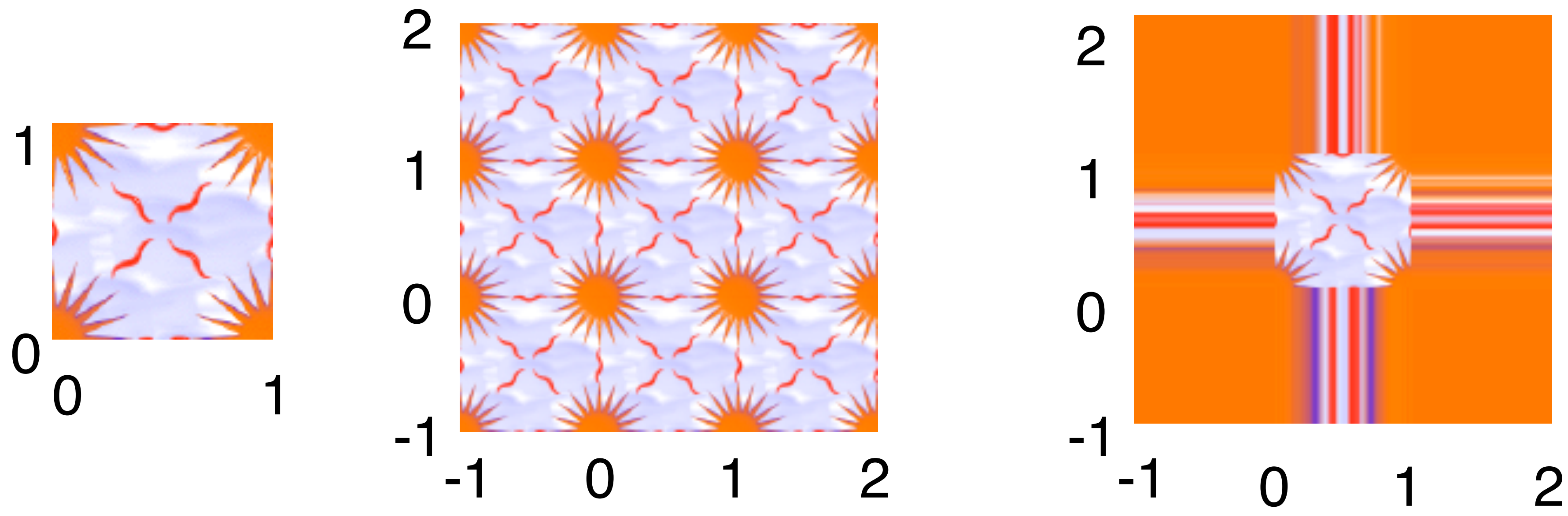


Texture parameters

glTexParameter(...);

GL_TEXTURE_WRAP_S
GL_TEXTURE_WRAP_T

GL_REPEAT
GL_CLAMP_TO_EDGE



Magnification and minification parameters:

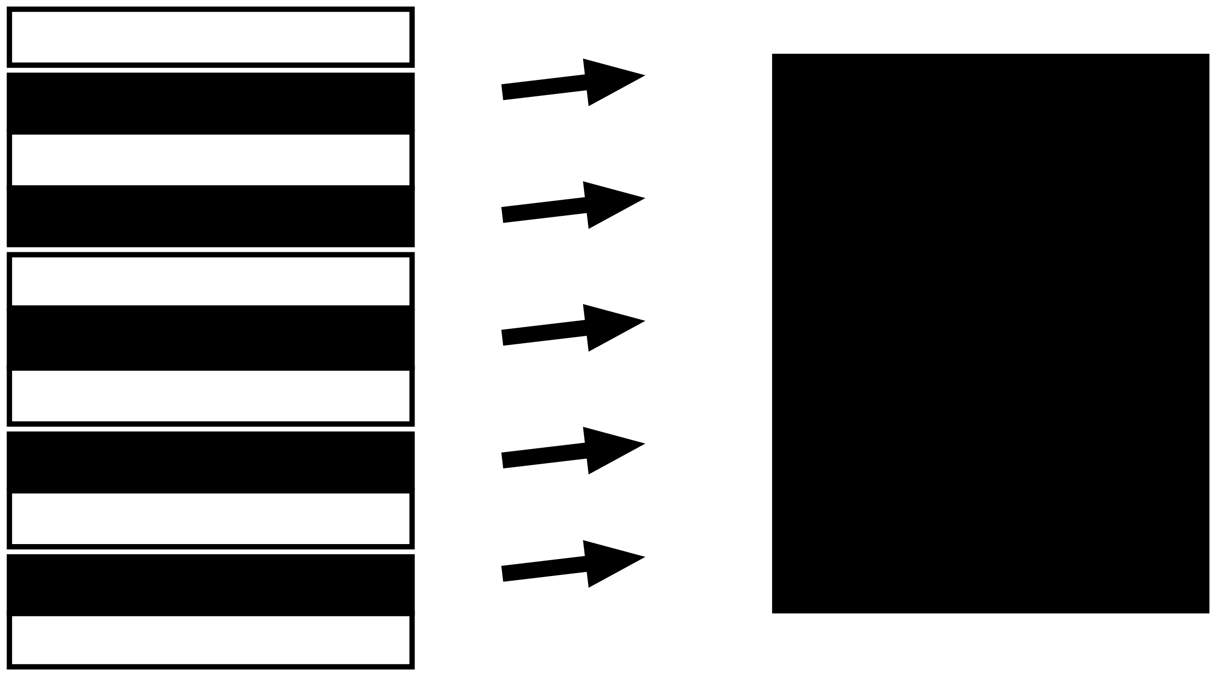
glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_MAG_FILTER, GL_NEAREST);

glTexParameteri(GL_TEXTURE_2D,
GL_TEXTURE_MIN_FILTER, GL_NEAREST);

Specifies what should happen when the texture doesn't match the pixel grid

Aliasing

A digital image is a sampled signal
If the signal is not band limited, aliasing will occur



Aliasing in texture mapping

At large distance, textures get smaller

=>

higher spatial frequencies on the screen

=>

increasing risk for aliasing!

Aliasing can be reduced by two methods:

Filtering

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
```

Mip-mapping

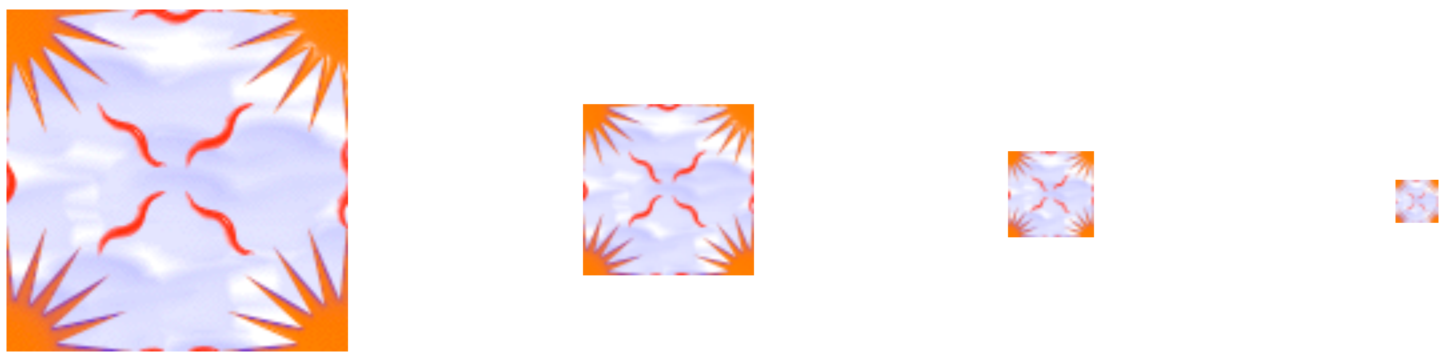
```
glGenerateMipmap();

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_NEAREST);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
```

MIP mapping

Texture mapping with anti-aliasing.

A resolution pyramid is built from every texture.

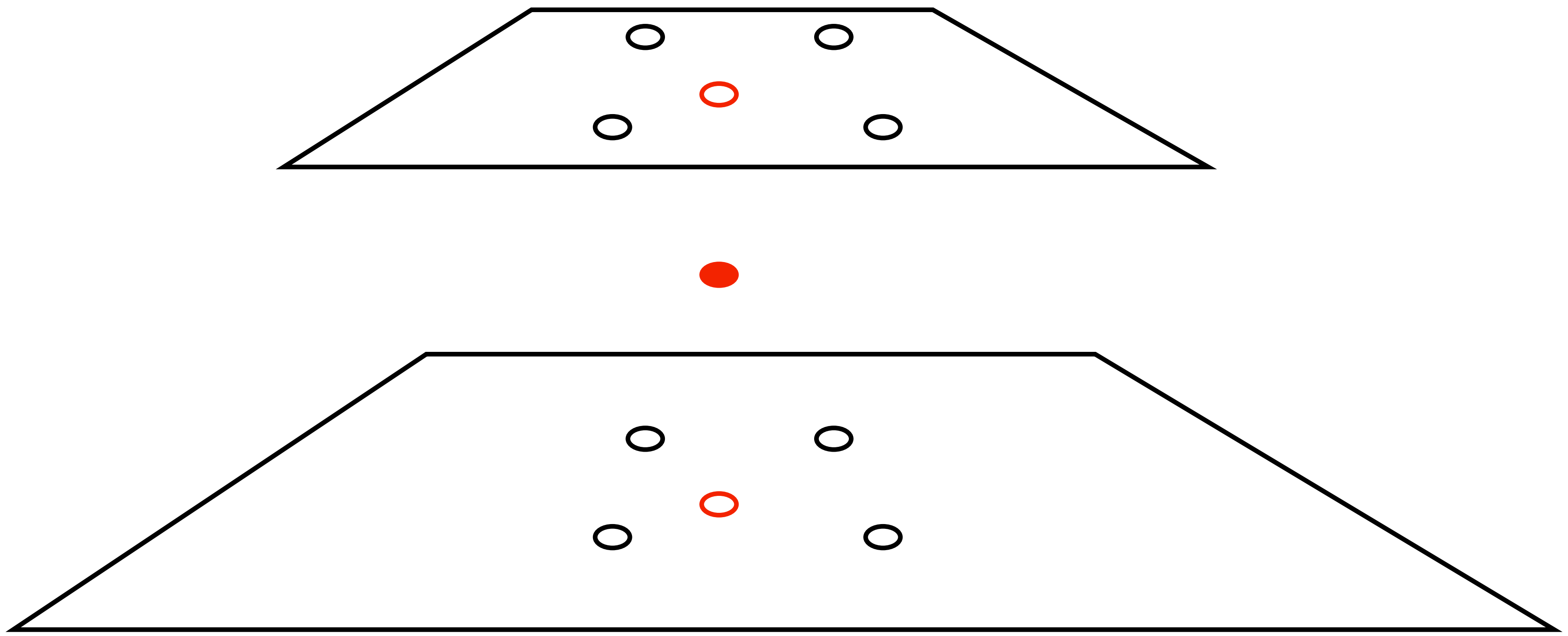
Memory cost: 33% more. Cheap!



128x128 64x64 32x32 16x16

MIP mapping filtering

Both within a level and between!



MIP mapping filtering

GL_NEAREST
GL_LINEAR
GL_NEAREST_MIPMAP_NEAREST
GL_LINEAR_MIPMAP_NEAREST
GL_NEAREST_MIPMAP_LINEAR
GL_LINEAR_MIPMAP_LINEAR

Preferred:
GL_LINEAR for magnification
GL_LINEAR_MIPMAP_LINEAR for minification

MIP mapping

Gives anti-aliasing at a very low cost.

Good results in most situations.

Aliasing problems remain at steep angles.