Textures

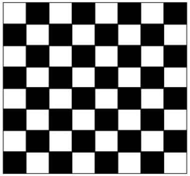
Textures

Texture = 1/2/3D data table.

Textures

Texture = 1/2/3D data table.

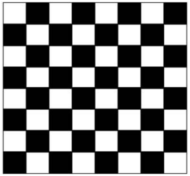
Often: (color values of) 2D picture.



Textures

Texture = 1/2/3D data table.

Often: (color values of) 2D picture.

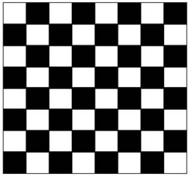


External file, generated online inside program (animated textures), or generated (offline or online) via rendering in OpenGL.

Textures

Texture = 1/2/3D data table.

Often: (color values of) 2D picture.



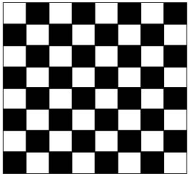
External file, generated online inside program (animated textures), or generated (offline or online) via rendering in OpenGL.

But texture data can be interpreted as anything, e.g. normal vectors, light maps/shadow maps, heightfields,. . .

Textures

Texture = 1/2/3D data table.

Often: (color values of) 2D picture.



External file, generated online inside program (animated textures), or generated (offline or online) via rendering in OpenGL.

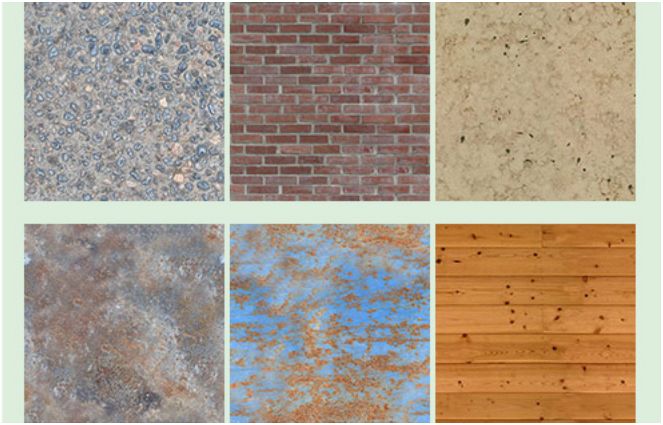
But texture data can be interpreted as anything, e.g. normal vectors, light maps/shadow maps, heightfields,...

A data entry in a texture table is called a texel.

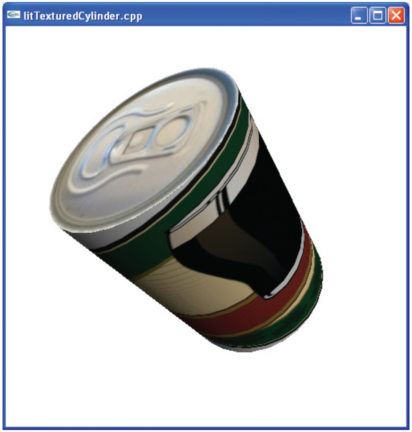
Use of Textures

- Generate detailed graphical content hardly possible with triangles (such clouds, skyboxes, plain pictures (posters, decals) on surfaces).
- Create illusion of structure, saving lots of triangles. Can be (low level) part of a level-of-detail scheme.
- Most of a game's graphical expression is via artwork using textures.
- Hold special-purpose data for use in rendering process.

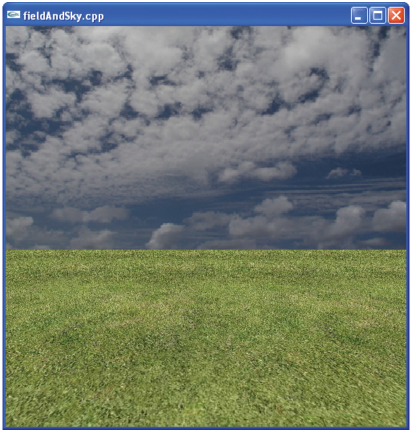
Examples


(From All Things Designed)

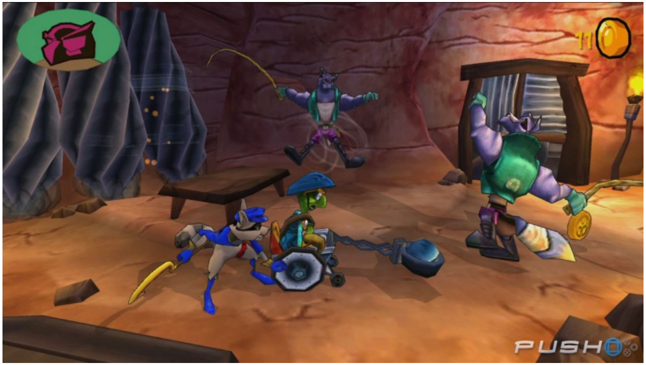
Examples



Examples

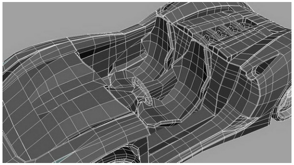


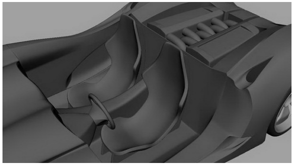
Examples

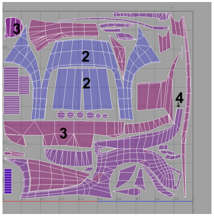


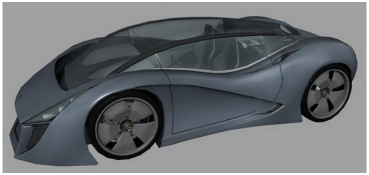
(From Sly Cooper)

Examples



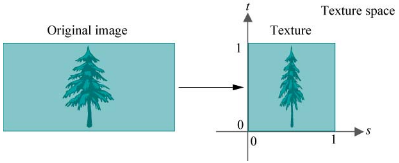






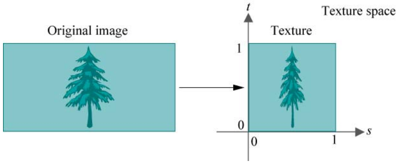
(By Valentin Nadolu)

Texture Coordinates



Texture data get mapped to $[0;1]^{1,2,3}$ in texture space.

Texture Coordinates

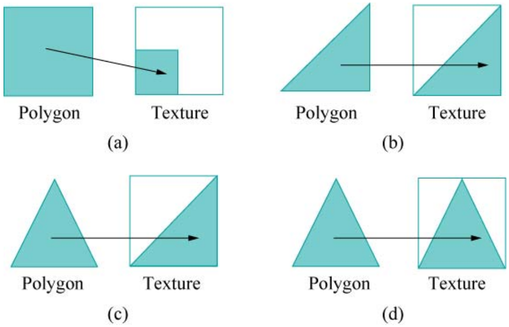


Texture data get mapped to $[0;1]^{1,2,3}$ in texture space.

When using textures, vertices are associated with texture coordinates.

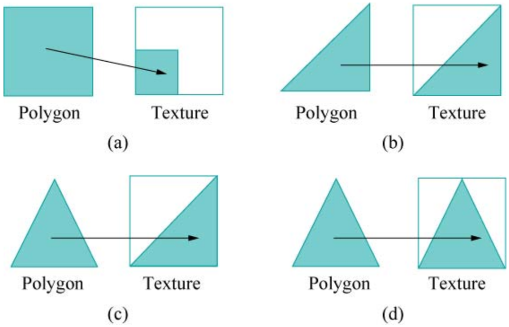
Texture Coordinates

Texture space points can be arbitrary:



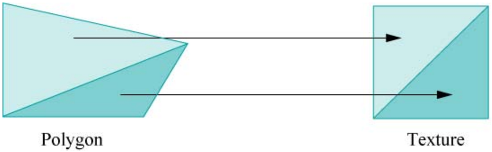
Texture Coordinates

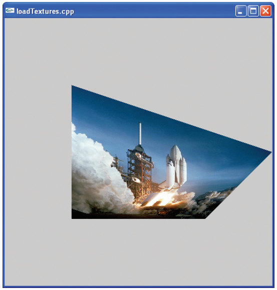
Texture space points can be arbitrary:

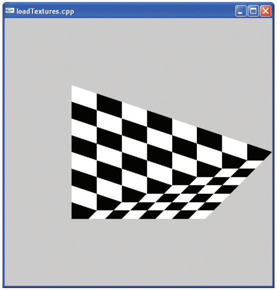


Points internally in triangle are associated with points in texture space using interpolation (via barycentric coordinates).

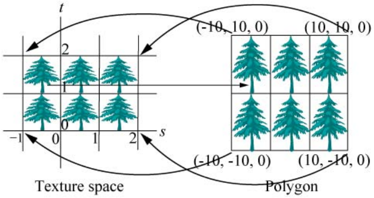
Interpolation Example

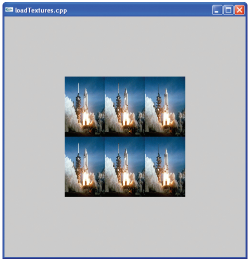






Coordinates Outside the Texture





Above is repetition

Below is clamping (on *s*-axis in texture space).

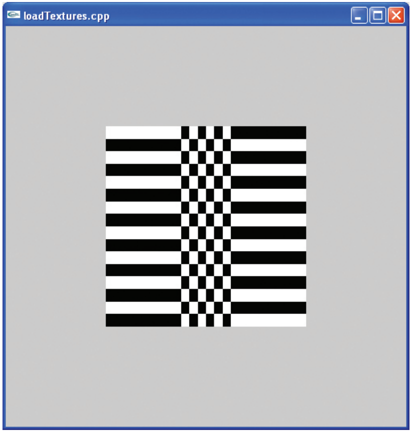
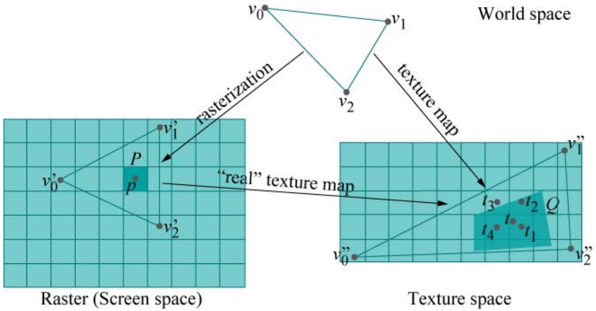


Table Lookup in Texture

Points on triangle do not correspond exactly to (middle of) texture cells.



- Nearest neighbor
- Linear interpolation (bilinear: linear interpolation on both *s*- and *t*-axes).

Minification/Magnification

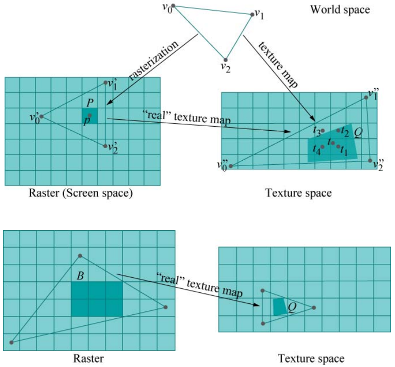
Associated pixels and texels may not be similar in size.

- Minification: one pixel corresponds to many texels.
- Magnification: one pixel corresponds to a fraction of a texel.

Minification/Magnification

Associated pixels and texels may not be similar in size.

- Minification: one pixel corresponds to many texels.
- Magnification: one pixel corresponds to a fraction of a texel.

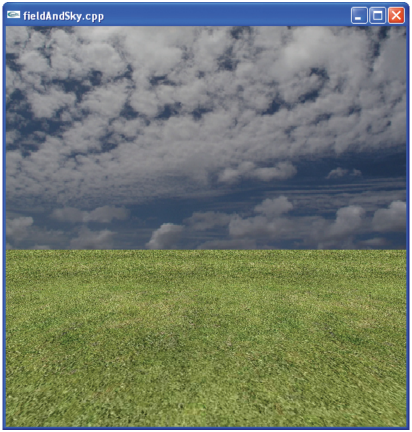


Problems:

- Magnification: Nearest neighbor lookup gives block-like result.
- Minification: Results of lookup for neighboring pixels are unrelated, and appear random. Even worse if viewing angle changes (flicker from the changing random lookups).

Problems:

- Magnification: Nearest neighbor lookup gives block-like result.
- Minification: Results of lookup for neighboring pixels are unrelated, and appear random. Even worse if viewing angle changes (flicker from the changing random lookups).



Remedies::

- Magnification: Use interpolation.
- Minification: use mipmaps.

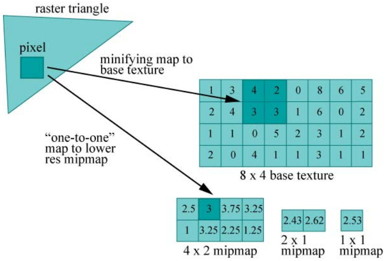
Remedies::

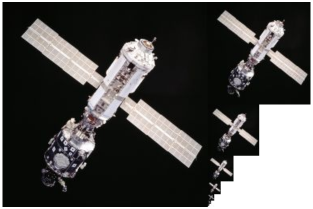
- Magnification: Use interpolation.
- Minification: use mipmaps.

Mipmap: Pre-made series of resolutions of a texture.

OpenGL has method for finding (during lookup) which one is closest to matching pixels and texels in size, and makes lookup in that. Or can use linear interpolation on lookups from two neighboring candidate resolutions. (Trilinear interpolation: also do bilinear interpolation in lookups).

Start with high-resolution texture. Create lower-resolution versions by averaging over sets of neighboring texels. (See GLU library.)

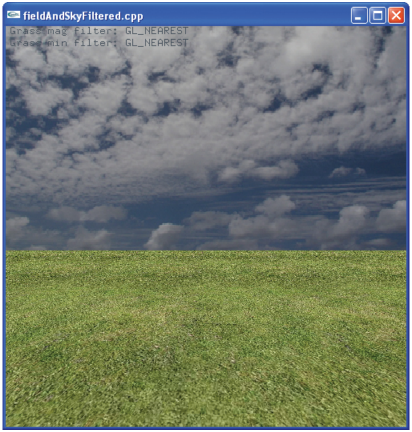




(By Mike Hick)

Example

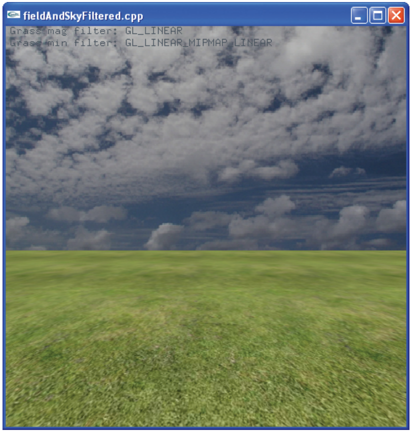
No mipmap, nearest neighbor:



```
Grass mag filter: GL_NEAREST
Grass min filter: GL_NEAREST
```

Example

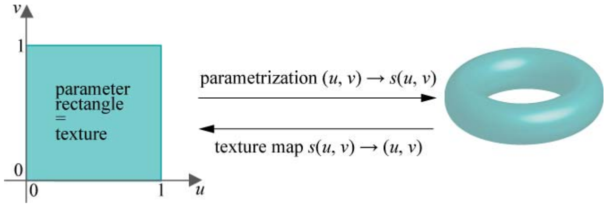
Using mipmap:



```
Grass mag filter: GL_LINEAR
Grass min filter: GL_LINEAR_MIPMAP_LINEAR
```

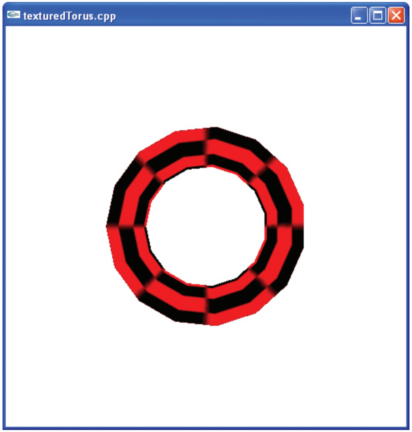
Parametrised Surfaces and Textures

Usually parameters can map simply to texture space


v
1
parameter rectangle = texture
0
0
1
u
parametrization $(u, v) \to s(u, v)$
texture map $s(u, v) \to (u, v)$

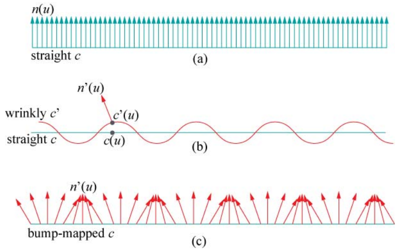
Example

Texture: red checkerboard.



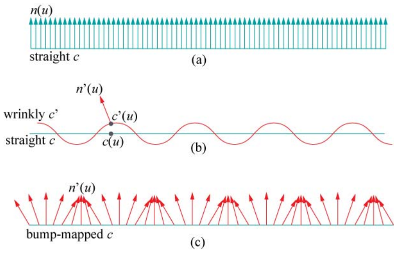
Texture Use: Bump mapping

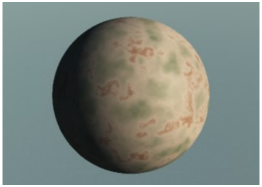
Store surface normals (or perturbation of normals) in texture.

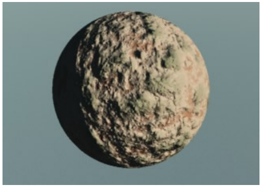


Texture Use: Bump mapping

Store surface normals (or perturbation of normals) in texture.



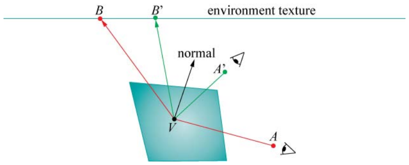


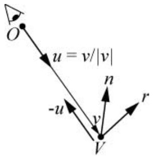


(Figure by `www.chromesphere.com`)

Texture Use: Environment Mapping

Relections can see environment. Make part of shading calculation.

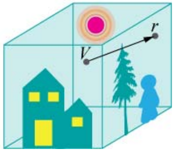




$$r = u - 2(n \cdot u)n$$

Environment Mapping

Easiest with Cube mapping:



Six single textures. Can each be generated online by rendering from center (even if moving) and saving framebuffer as texture.

Note: environment mapping assumes distance to environment large compared to extent of model (or else, correct lookup would not depend on only vector r).