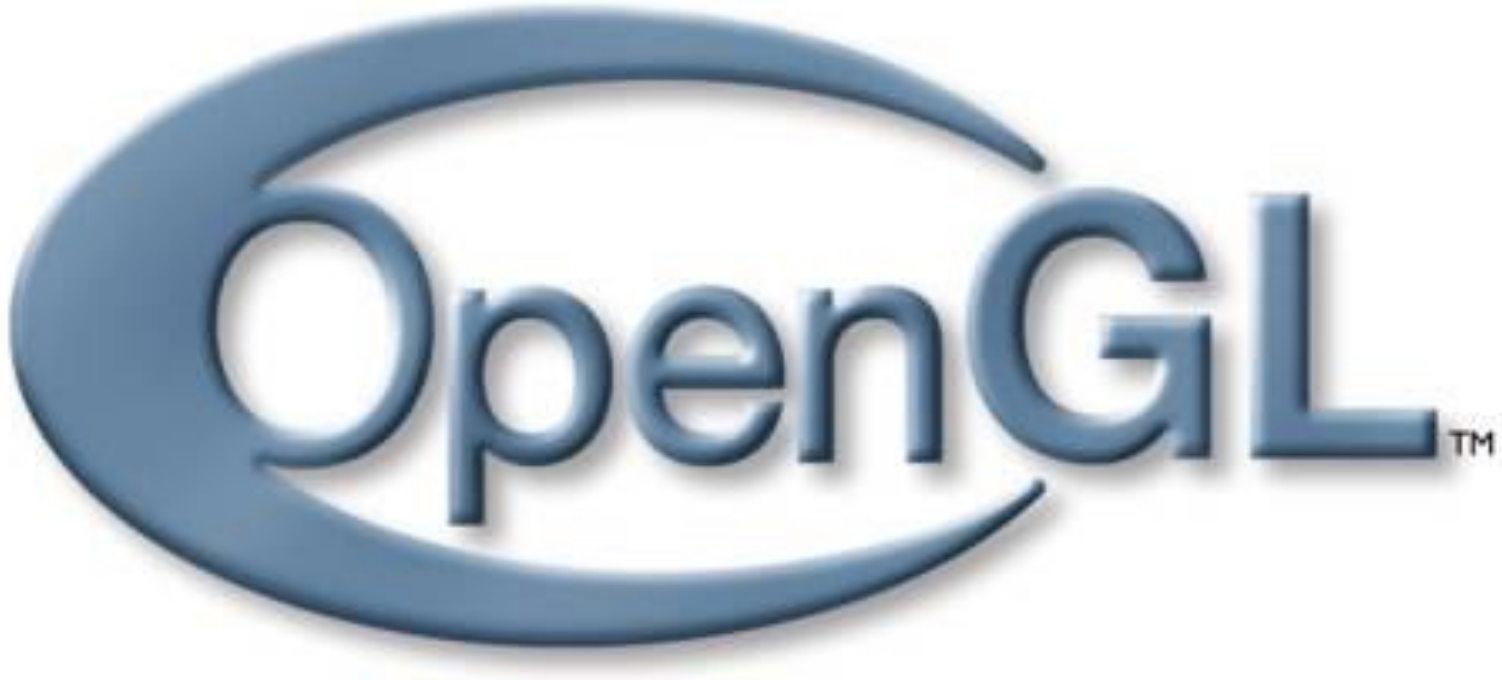


part 5

Martin Samuelčík
http://www.sccg.sk/~samuelcik
Room I4

Textures

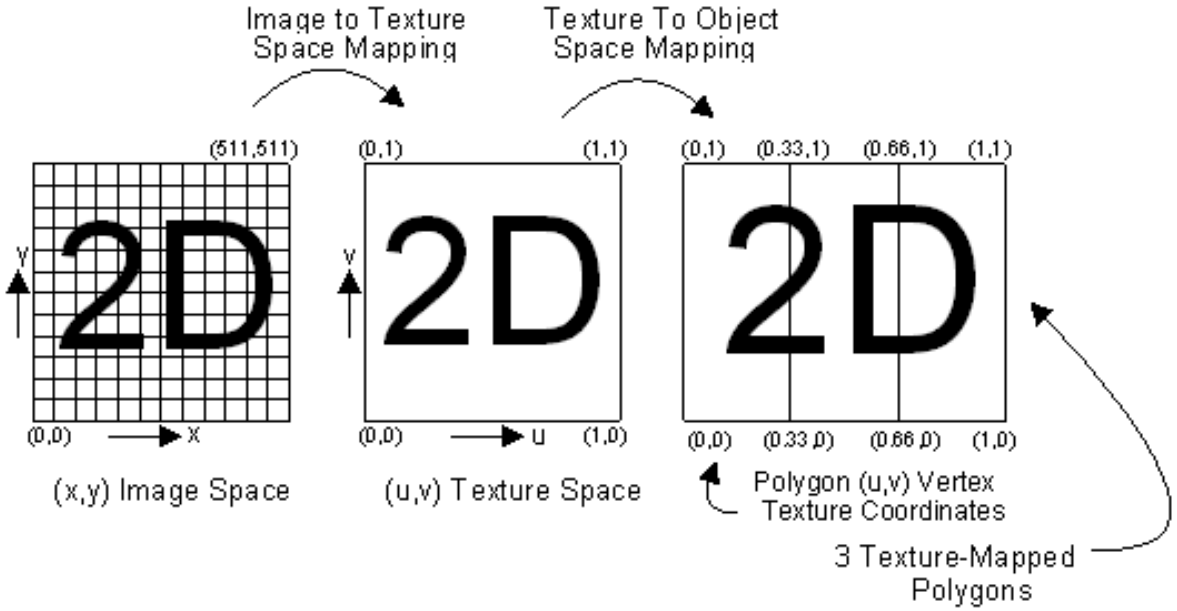
- Using pictures in virtual scene
- Mapping picture on objects
- Source, data
  - Procedural
  - Rendered
  - Photographs
  - From artists
  - …

Texture mapping

- We need to describe how image (texture) is mapped onto object
- Mapping on each polygon separately
- Defining new vertex attribute for each vertex – texture coordinates
- Texture coordinates – (s,q,r,t) in normalized texture coordinates
- Defining which pixel of texture is mapped onto given vertex

Texture coordinates

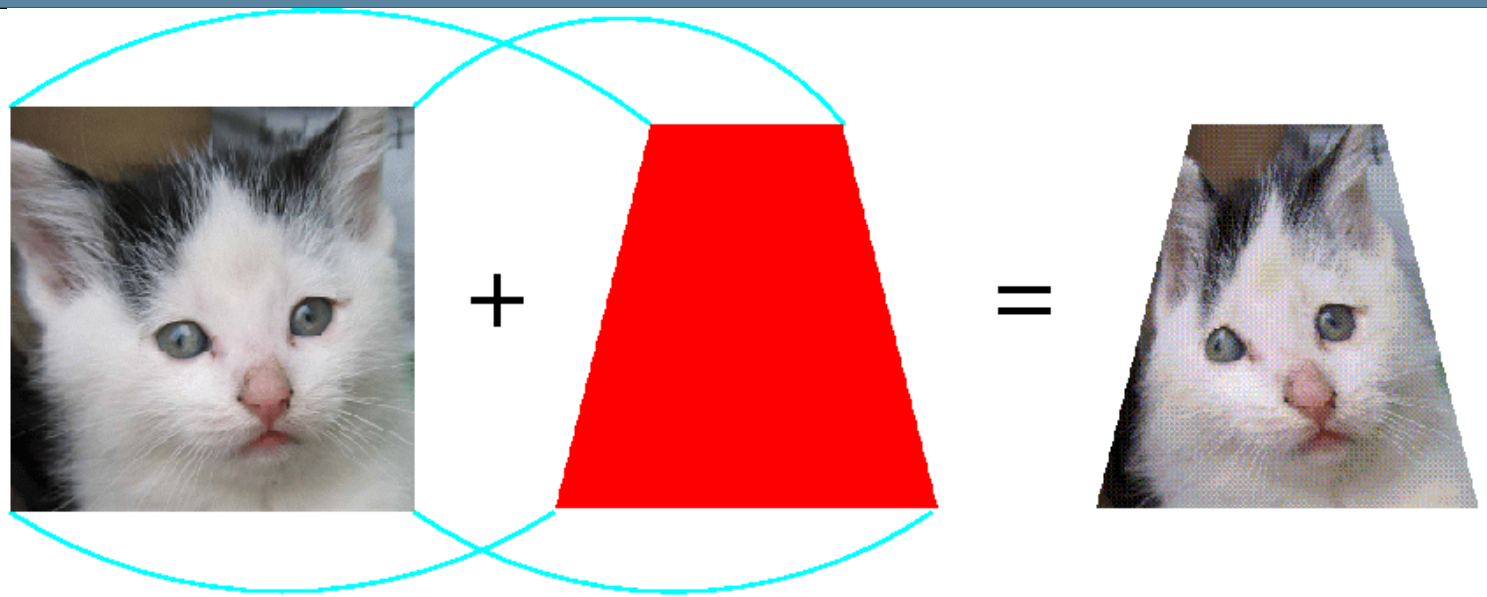
- 1D, 2D, 3D
- Defined in normalized space, where whole image is in region <0,1>x<0,1>x<0,1>
- Mapping of image with dimensions (width, height, depth) to this unit square



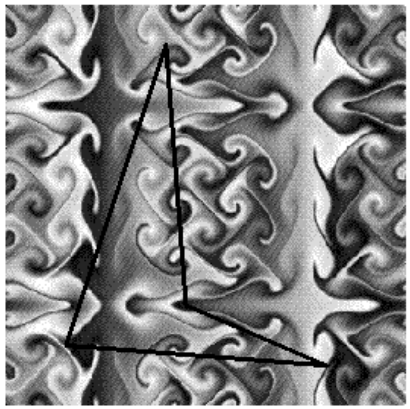
Texture mapping

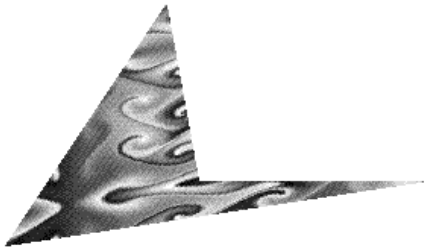
- Texture coordinates are given for each vertex as vertex attributes
- Rasterizer computes texture coordinates for each fragment using linear interpolation
- For each fragment, its texture coordinates are recomputed to be in interval <0,1>, then multiplied by texture dimensions and color is then fetched from texel using computed image coordinates

Texture mapping



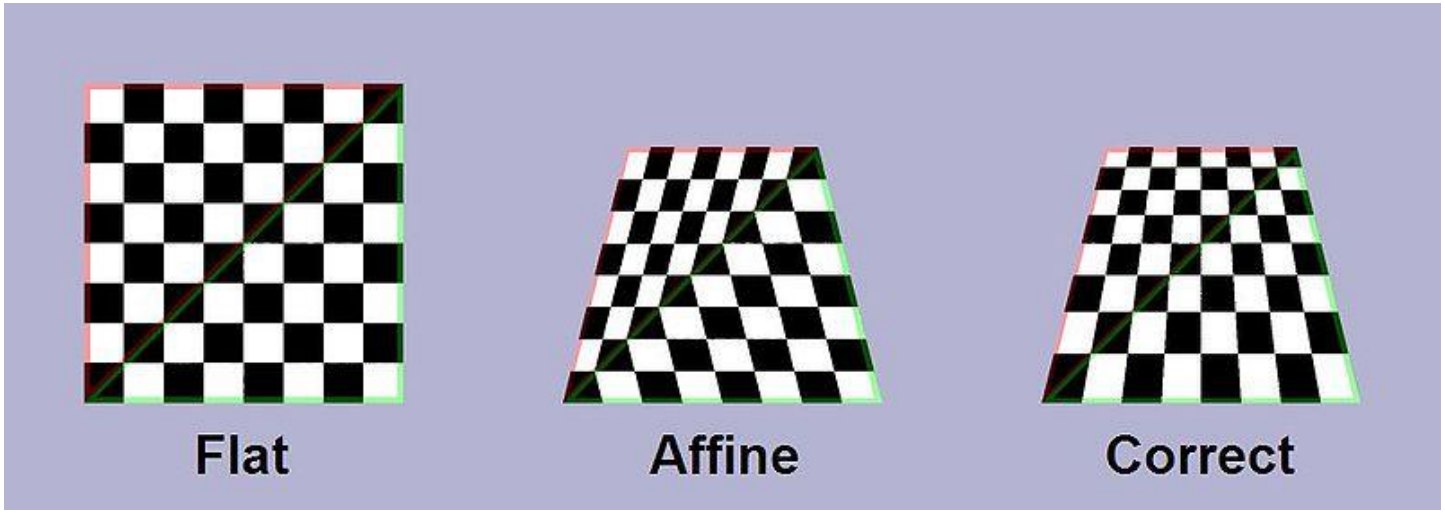
Texture Object Mapped Texture





Perspective correctness

- Bilinear interpolation of texture coordinates causes artifacts
- Adding perspective projection to interpolation of texture coordinates
- **glHint(GL_PERSPECTIVE_CORRECTION_HINT,GL_NICEST)**



OpenGL texture coordinates

- Immediate mode – state variable
  - Each attribute given separately
  - **void glTexCoord{1234}{sifd}(TYPE coords)**
  - **void glTexCoord{1234}{sifd}v(TYPE *coords)**
- Vertex arrays or VBO
  - All attributes in one array
  - **void glTexCoordPointer(GLint size, GLenum type, GLsizei stride, const GLvoid * pointer);**

Examples

```
// using vertex arrays
GLfloat vertices[] = {-4.0f,-4.0f,0.0f,  4.0f,-4.0f,0.0f,  4.0f,4.0f,0.0f,  -4.0f,4.0f,0.0f};
GLfloat texcoords[] = {0.0f,0.0f,  1.0f,0.0f,  1.0f,1.0f,  0.0f,1.0f};
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, vertices);
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
glTexCoordPointer(2, GL_FLOAT, 0, texcoords);
glDrawArrays(GL_QUADS, 24, 0, 4);
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_TEXTURE_COORD_ARRAY);

// using imeddiate mode
glBegin(GL_QUADS);
glTexCoord2f(0.0f, 0,0f);
glVertex3f(-4.0f, -4.0f, 0.0f);
glTexCoord2f(1.0f, 0,0f);
glVertex3f(4.0f, -4.0f, 0.0f);
glTexCoord2f(1.0f, 1,0f);
glVertex3f(4.0f, 4.0f, 0.0f);
glTexCoord2f(0.0f, 1,0f);
glVertex3f(-4.0f, 4.0f, 0.0f);
glEnd():
```

Texture matrix stack

- Texture coordinates are multiplied by a 4 × 4 texture matrix during vertex transformation before rasterization
- Texture matrix – state variable
- **glMatrixMode(GL_TEXTURE)**
- After transformation, 3 coordinates are divided by fourth, homogenous coordinate
- For light maps, shadow maps, volumetric textures, …

Enabling mapping

- Do not forget this, it is disabled by default
- **glEnable(GL_TEXTURE_2D)**
- **glDisable(GL_TEXTURE_2D)**

Texture objects

- Each texture with its properties stored in graphics memory as texture object
- Identified using integer number (name)
- Creating list of unused names
  - **void glGenTextures(GLsizei n, GLuint *textureNames)**
- Checking bound texture
  - **GLboolean glIsTexture(GLuint textureName)**

Using texture objects

- For each dimension of texture – current texture as state variable
- Setting texture as actual
- Actual texture is applied on object when texturing is enabled
- **void glBindTexture(GLenum target, GLuint textureName)**
  - target=GL_TEXTURE_1D, GL_TEXTURE_2D
- All operations with current texture are treated as operations with bound texture

Cleaning up

- We need to clear unnecessary texture resources in memory
- **void glDeleteTextures(GLsizei n, const GLuint *textureNames)**
- Deletes n texture objects given in array
- Also bounded texture can be deleted
- Cleaned automatically also at the end of application

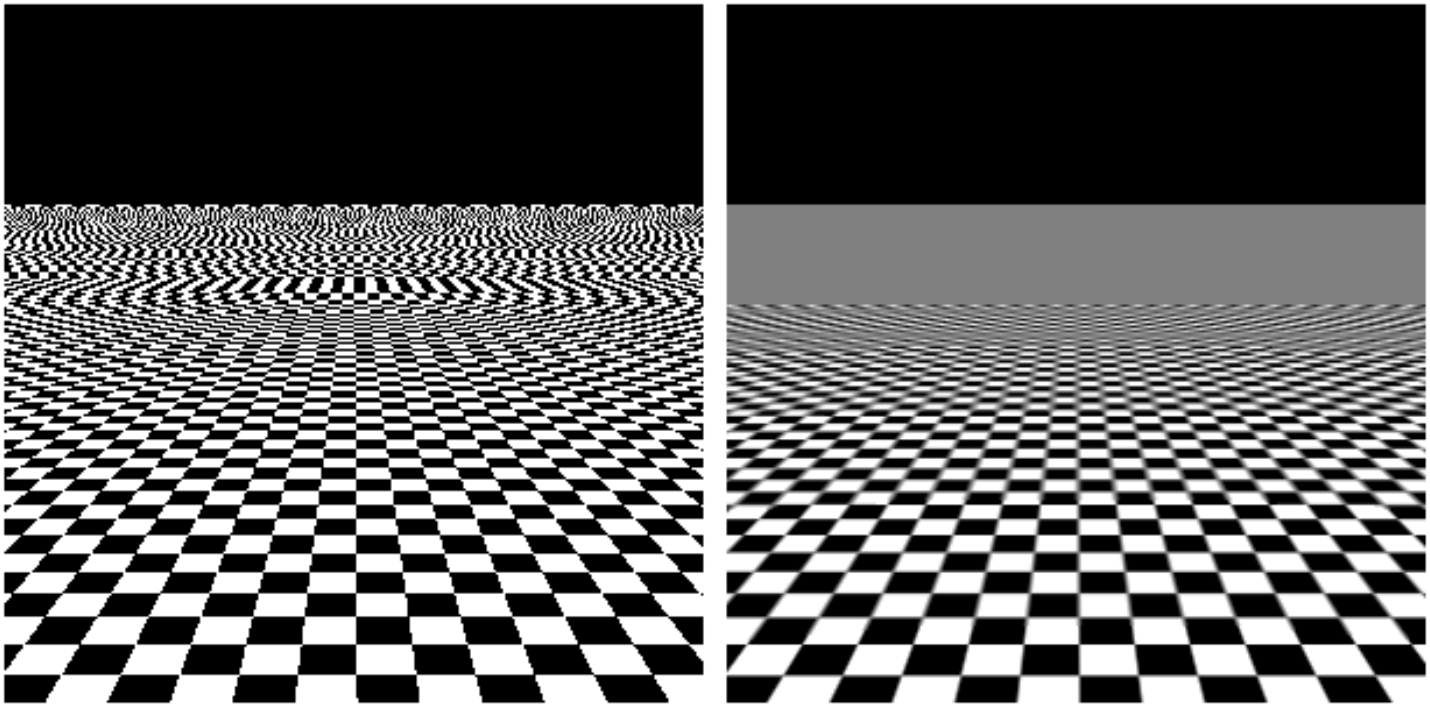
Texture data

- Specifying data for current texture
- **void glTexImage2D(GLenum target, GLint level, GLint internalFormat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *pixels)**
- target – GL_TEXTURE_2D (copying data), GL_PROXY_TEXTURE_2D (testing specification of texture, no pixels needed)

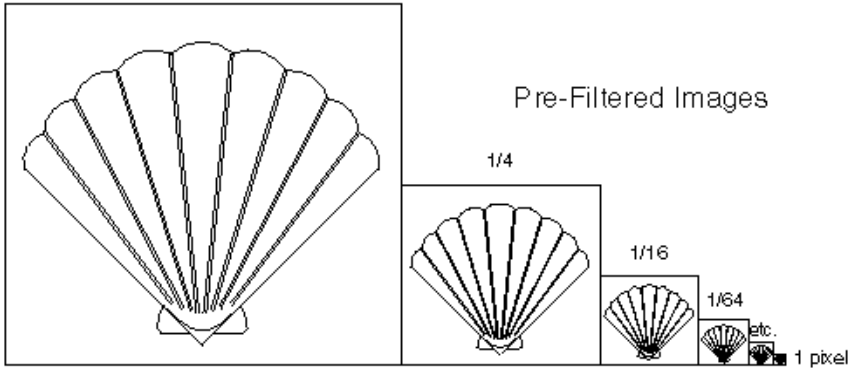
Level

- Mip-mapping – supplying multiple resolutions of texture map
- Removing under sampling artifacts
- OGL automatically determines usage of mipmapping
- Best texture is declared as first
- Generation of all levels with upload
  int gluBuild2DMipmaps(GLenum target, GLint components, GLint width, GLint height, GLenum format, GLenum type, const void *data);

Level



Original Texture



Internal format

- Storage data format in graphics memory

GL_ALPHA, GL_ALPHA4, GL_ALPHA8, GL_ALPHA12,
GL_ALPHA16, GL_LUMINANCE, GL_LUMINANCE4,
GL_LUMINANCE8, GL_LUMINANCE12,
GL_LUMINANCE16, GL_LUMINANCE_ALPHA,
GL_LUMINANCE4_ALPHA4, GL_LUMINANCE6_ALPHA2,
GL_LUMINANCE8_ALPHA8, GL_LUMINANCE12_ALPHA4,
GL_LUMINANCE12_ALPHA12,
GL_LUMINANCE16_ALPHA16, GL_INTENSITY,
GL_INTENSITY4, GL_INTENSITY8, GL_INTENSITY12,
GL_INTENSITY16, GL_RGB, GL_R3_G3_B2, GL_RGB4,
GL_RGB5, GL_RGB8, GL_RGB10, GL_RGB12, GL_RGB16,
GL_RGBA, GL_RGBA2, GL_RGBA4, GL_RGB5_A1,
GL_RGBA8, GL_RGB10_A2, GL_RGBA12, GL_RGBA16

Dimensions

- border – width of texture border
- width, height – recommended 2^m x 2^n
- Some older systems support only power of 2 dimensions
- Maximum size depends on implementation
  - **GLint texSize;**
  - **glGetIntegerv(GL_MAX_TEXTURE_SIZE, &texSize);**
- Border is used for repeating, blending …

Data

- Format and type of raw array of pixels
- For 2D texture array contains consecutive rows of texture from bottom to top
- format = GL_COLOR_INDEX, GL_RGB, GL_RGBA, GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA, GL_LUMINANCE, GL_LUMINANCE_ALPHA
- type = GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT, GL_INT, GL_UNSIGNED_INT, GL_FLOAT, GL_BITMAP
- pixels = array of texels

Data loading

- Source of data – usually external files
- Image formats – lots of compression and quantization algorithms
- OpenGL – raw uncompressed data
- Library for loading and editing images
- DevIL - openil.sourceforge.net
- ImageMagick - www.imagemagick.org
- FreeImage - freeimage.sourceforge.net
- SOIL - www.lonesock.net/soil.html
- UOGLSDK - glsdk.sourceforge.net

Updating texture

- Replacing part of texture with new data
- **void glTexSubImage2D(GLenum target, GLint level, GLint xoffset, GLint yoffset, GLsizei width, GLsizei height, GLenum format, GLenum type, const GLvoid *pixels)**
  - target – GL_TEXTURE_2D
  - level, format, type – similar to glTexImage2D
  - width, height – dimensions of the subregion, can be any nonnegative number
  - xoffset, yoffset – offset of starting texel
  - pixels – data of inserted image

Data from framebuffer

- **void void glCopyTexImage2D(GLenum target, GLint level, GLint internalFormat, GLint x, GLint y, GLsizei width, GLsizei height, GLint border)**
  - Copies rectangle region (x, y, width, height) from framebuffer to current 2D texture
  - Parameters are same as for glTexImage2D
- **void glCopyTexSubImage2D(GLenum target, GLint level, GLint xoffset, GLint yoffset, GLint x, GLint y, GLsizei width, GLsizei height)**
  - Using part of the screen as subtexture
  - Combination of previous functions

Texture parameters

- Part of texture objects
- Specifies how texture is treated and mapped
- Setting for current texture
- **void glTexParameter{if}(GLenum target, GLenum pname, TYPE param);**
- **void glTexParameter{if}v(GLenum target, GLenum pname, TYPE *param)**
  - target – which current texture is affected, GL_TEXTURE_1D, GL_TEXTURE_2D, …

Texture parameters

Parameter (pname)	Values (param)
GL_TEXTURE_WRAP_S	GL_CLAMP, GL_REPEAT
GL_TEXTURE_WRAP_T	GL_CLAMP, GL_REPEAT
GL_TEXTURE_MAG_FILTER	GL_NEAREST, GL_LINEAR
GL_TEXTURE_MIN_FILTER	GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_LINEAR
GL_TEXTURE_BORDER_COLOR	any four values in [0.0, 1.0]
GL_TEXTURE_PRIORITY	[0.0, 1.0] for the current texture object

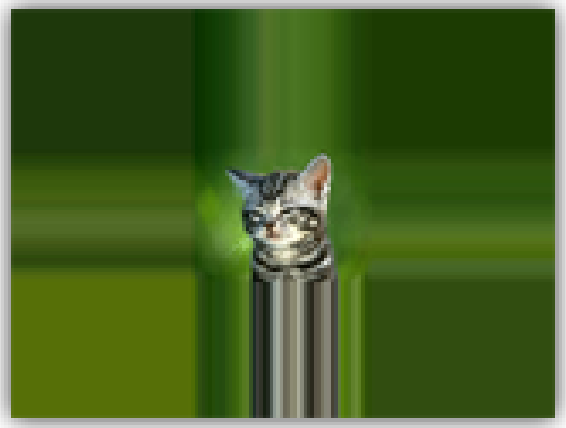
Wrapping

- How to process fragment with texture coordinates out of range <0,1>
- Specified for each coordinate separately
- GL_CLAMP – coordinate greater then 1 is clamped to 1, if less then 0 clamped to 0
- GL_REPEAT – the integer part of coordinate is ignored
- GL_CLAMP_TO_EDGE – coordinate is clamped to range <1/(2N), 1-1/(2N)>, N is size of texture in that direction…
- GL_MIRRORED_REPEAT – if integer part of coordinate is even, result is its fractional part, otherwise result is 1-(fractional part of coordinate)

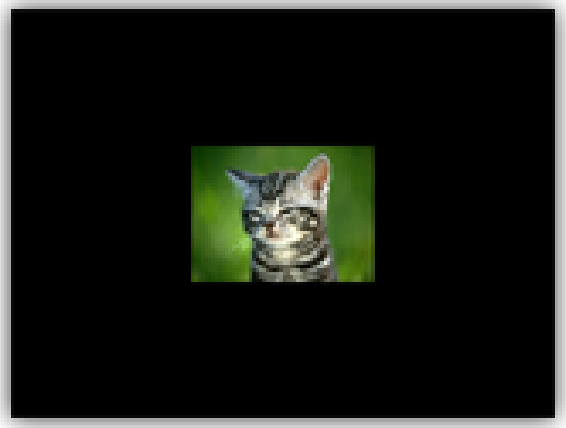
Wrapping

GL_REPEAT

GL_MIRRORED_REPEAT



GL_CLAMP_TO_EDGE



GL_CLAMP_TO_BORDER

Filtering

- Texels have exact textures coordinates
  - [(1+2i)/(2w), (1+2j)/(2h)]
  - i=0,…,w-1 ; j = 0,…,h-1
- Setting which texel colors are used when fragment texture coordinates are not exactly texel coordinates
- GL_TEXTURE_MIN_FILTER – used when texel projected on screen is minified
- GL_TEXTURE_MAG_FILTER – used when texel projected on screen is magnified

Filtering

- GL_NEAREST – based on texture coordinates, get color from nearest texel based on Manhattan metric
- GL_LINEAR – colors are fetched from 4 nearest texels and then combined using bilinear interpolation
- GL_*_MIPMAP_* - filtering between two best selected mipmap levels and inside each of these levels, possible trilinear interpolation, without this setting, mipmapping does not work

Filtering

GL_NEAREST

GL_LINEAR

Texture blending

- At the fragment processing level
- Result from texture mapping – (C_t, A_t)
- Original fragment color – (C_f, A_f)
- Mixing these two colors together – (C, A)
- **void glTexEnv{if}(GLenum target, GLenum pname, TYPE param)**
- **void glTexEnv{if}v(GLenum target, GLenum pname, TYPE *param)**
  - target - GL_TEXTURE_ENV
  - If pname is GL_TEXTURE_ENV_MODE, param can be GL_DECAL, GL_REPLACE, GL_MODULATE, GL_BLEND
  - If pname is GL_TEXTURE_ENV_COLOR, param is an array of R, G, B, and A, used only when GL_BLEND is enabled

Texture blending

Base Texture Internal Format	Replace Texture Function	Modulate Texture Function	Decal Texture Function	Blend Texture Function
GL_ALPHA	$C = C_f$, $A = A_t$	$C = C_f$, $A = A_f A_t$	undefined	$C = C_f$, $A = A_f A_t$
GL_LUMINANCE	$C = L_t$, $A = A_f$	$C = C_f L_t$, $A = A_f$	undefined	$C = C_f(1-L_t) + C_c L_t$, $A = A_f$
GL_LUMINANCE_ALPHA	$C = L_t$, $A = A_t$	$C = C_f L_t$, $A = A_f A_t$	undefined	$C = C_f(1-L_t) + C_c L_t$, $A = A_f A_t$
GL_INTENSITY	$C = I_t$, $A = I_t$	$C = C_f I_t$, $A = A_f I_t$	undefined	$C = C_f(1-I_t) + C_c I_t$, $A = A_f(1-I_t) + A_c I_t$
GL_RGB	$C = C_t$, $A = A_f$	$C = C_f C_t$, $A = A_f$	$C = C_t$, $A = A_f$	$C = C_f(1-C_t) + C_c C_t$, $A = A_f$
GL_RGBA	$C = C_t$, $A = A_t$	$C = C_f C_t$, $A = A_f A_t$	$C = C_f(1-A_t) + C_t A_t$, $A = A_f$	$C = C_f(1-C_t) + C_c C_t$, $A = A_f A_t$

Example

```
// initialization part, creating and setting texture object
GLuint texid;
glGenTextures(1, &texid);
glBindTexture(GL_TEXTURE_2D, texid);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, 64, 64, 0, GL_RGBA, GL_UNSIGNED_BYTE, data);
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
glHint(GL_PERSPECTIVE_CORRECTION_HINT,GL_NICEST);

// rendering part
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texid);
glColor4f(1.0f, 0.0f, 0.0f, 1.0f);
glBegin(GL_QUADS);
glTexCoord2f(0.0f, 0,0f);
glVertex3f(-4.0f, -4.0f, 0.0f);
glTexCoord2f(5.0f, 0,0f);
glVertex3f(4.0f, -4.0f, 0.0f);
glTexCoord2f(5.0f, 1,0f);
glVertex3f(4.0f, 4.0f, 0.0f);
glTexCoord2f(0.0f, 1,0f);
glVertex3f(-4.0f, 4.0f, 0.0f);
glEnd():
```

Advanced textures

- 1D, 3D textures
- Shadow textures
- Multitexturing
- Cube textures
- More render to texture options, FBO
- Blending modes
- Floating point textures
- Pixel and texture buffers

The End!

Questions?