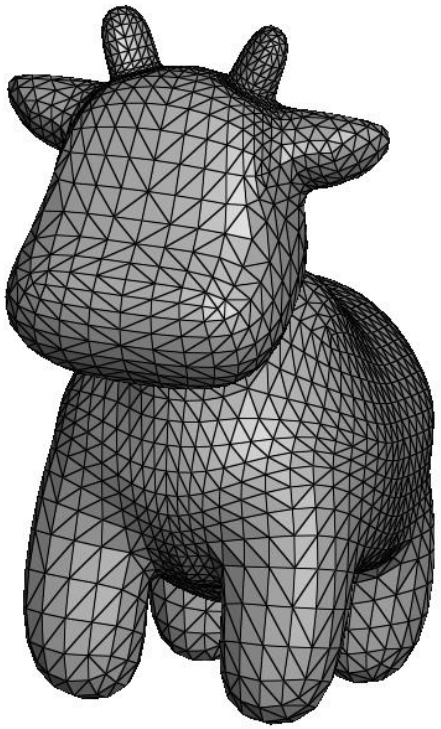
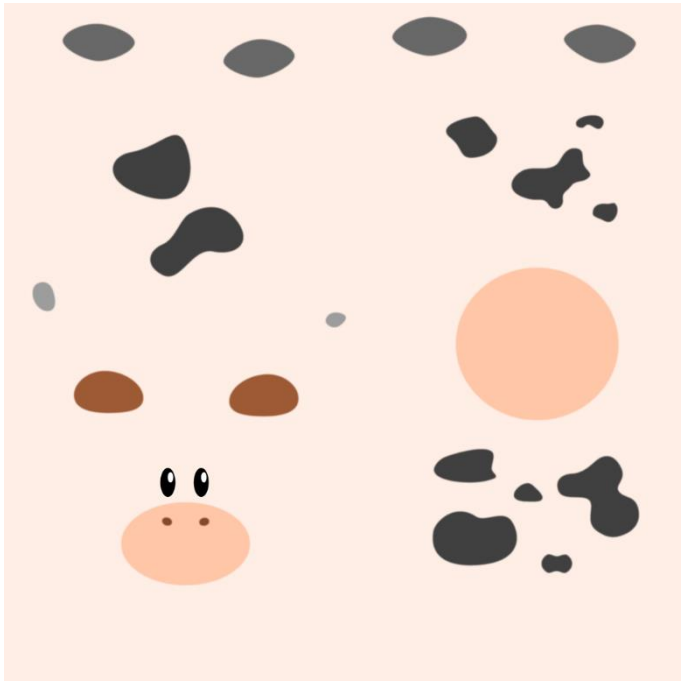
Texture mapping

Texture mapping

Map image information (e.g. color) to surface points (usually on a triangle mesh)



Triangle mesh



Texture image

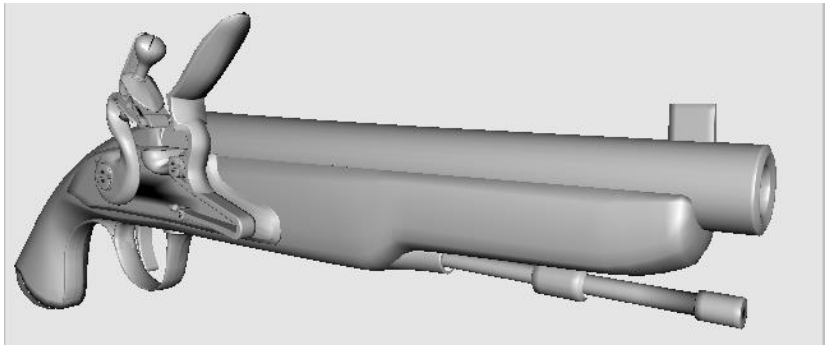
Textured mesh

Spot model by K. Crane

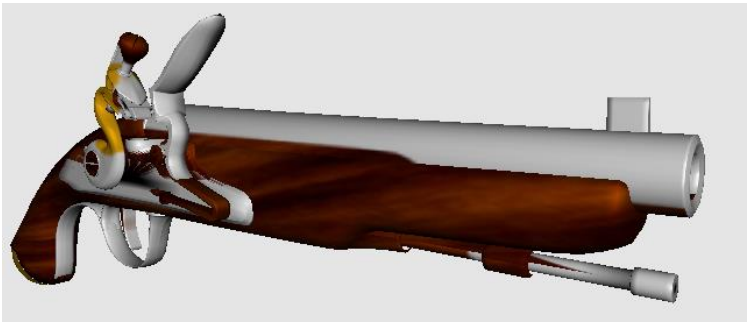
Texture mapping

Allow to achieve different visual effects

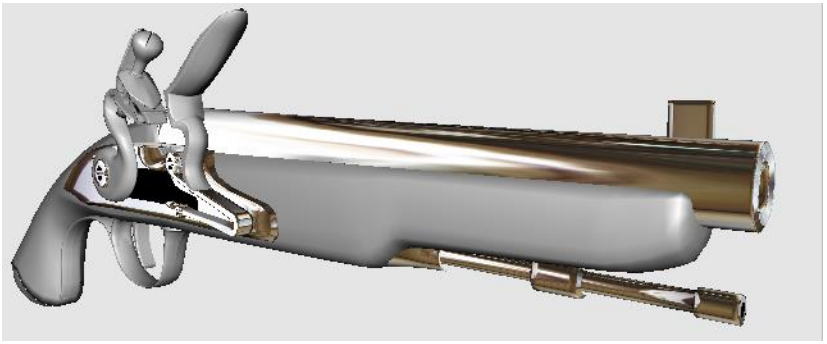
Original model



Textured object



Using environment mapping



Final model

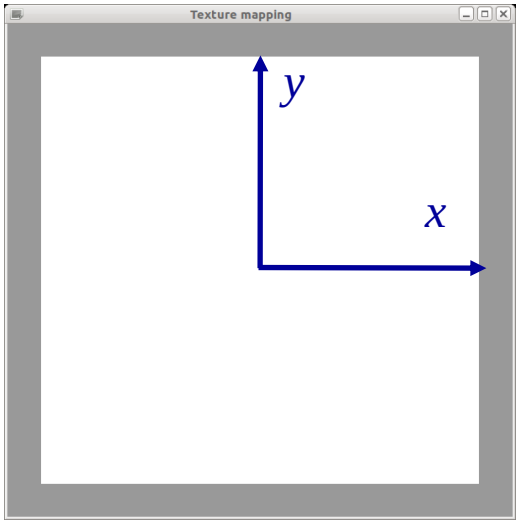
Image credit: Y. Nishidate

Texture mapping

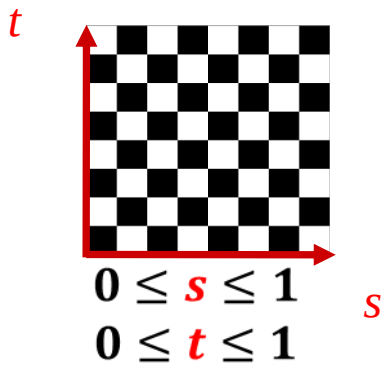
- Associate a collection of coordinates $(s_1, \ldots, s_n)$ to each mesh vertex, $n = 1, 2, 3, or\ 4$, $0 \leq s_i \leq 1$.
- Look up the color of the image at position $(s_1, \ldots, s_n)$ to define the color of a vertex.
- Implemented as part of the shading process (after rasterization).
- Most important case in computer graphics is n=2: two-dimensional textures (images) are mapped to surfaces.

Simple example of 2D texture mapping

Quadrilateral



Texture image
(n x m texels)

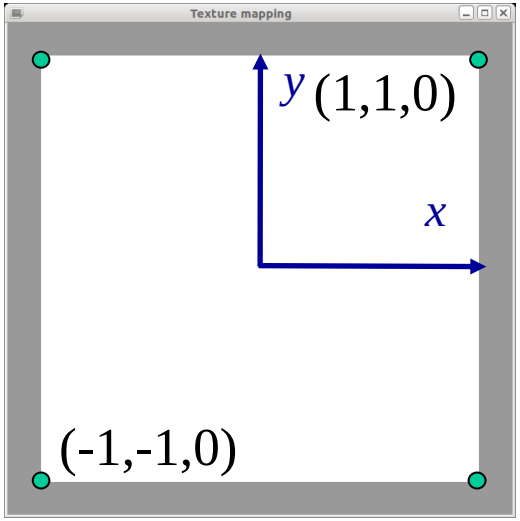


Coordinates in texture space coordinates described by variables s and t.
They range over interval [0,1].
Texture elements are called *texels*.

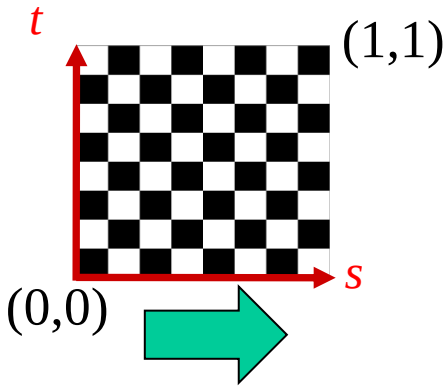
Coordinate (*s*,*t*) = (0, 0) corresponds to texel (0, 0) and (*s*,*t*) = (1, 1) corresponds to texel (n-1, m-1).

Simple example of 2D texture mapping

Quadrilateral

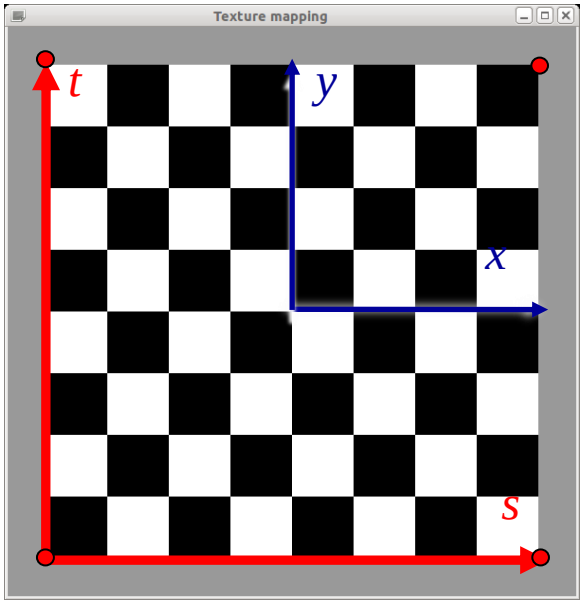


Texture image
(n x m texels)



Texel (n,m) corresponds to (s,t) = (1,1)

Textured quadrilateral




```
glBegin(GL_QUADS);
  glVertex3d(-1.0, -1.0, 0.0);
  glVertex3d( 1.0, -1.0, 0.0);
  glVertex3d( 1.0,  1.0, 0.0);
  glVertex3d(-1.0,  1.0, 0.0);
glEnd();
```

```
glBegin(GL_QUADS);
  glTexCoord2d(0.0, 0.0); glVertex3d(-1.0, -1.0, 0.0);
  glTexCoord2d(1.0, 0.0); glVertex3d( 1.0, -1.0, 0.0);
  glTexCoord2d(1.0, 1.0); glVertex3d( 1.0,  1.0, 0.0);
  glTexCoord2d(0.0, 1.0); glVertex3d(-1.0,  1.0, 0.0);
glEnd();
```

Parameterization

Parameterization: Finding a mapping from a 3D surface to 2D.

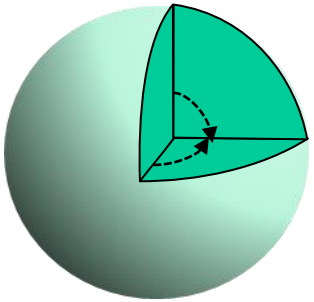
Difficult in general: e.g. non-flat surfaces, surfaces without boundary (sphere), surfaces with holes (torus).

Computer Graphics case: smooth surfaces often approximated by triangle meshes. Problem: assign texture coordinates to each mesh vertex.

Parameterization

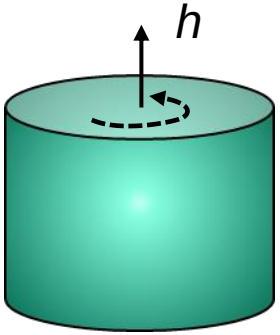
"Simpler" cases: objects with natural parameterizations.

- Sphere: spherical coordinates (θ, φ), $\theta \in [0, 2\pi), \varphi \in [0, \pi]$



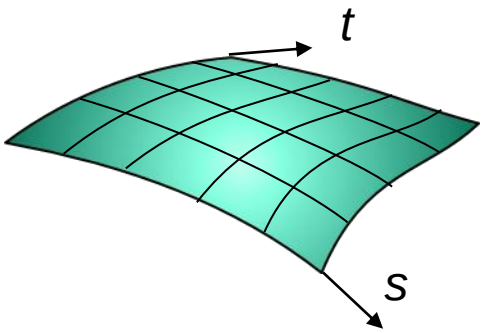
$$x = R\cos\theta\sin\varphi$$
$$y = R\sin\theta\sin\varphi$$
$$z = R\cos\varphi$$

- Cylinder: cylindrical coordinates $(\theta, h),\ \theta \in [0, 2\pi]$



$$x = R\cos\theta$$
$$y = R\sin\theta$$
$$z = h$$

- Parametric surfaces (e.g. Bezier or B-spline)

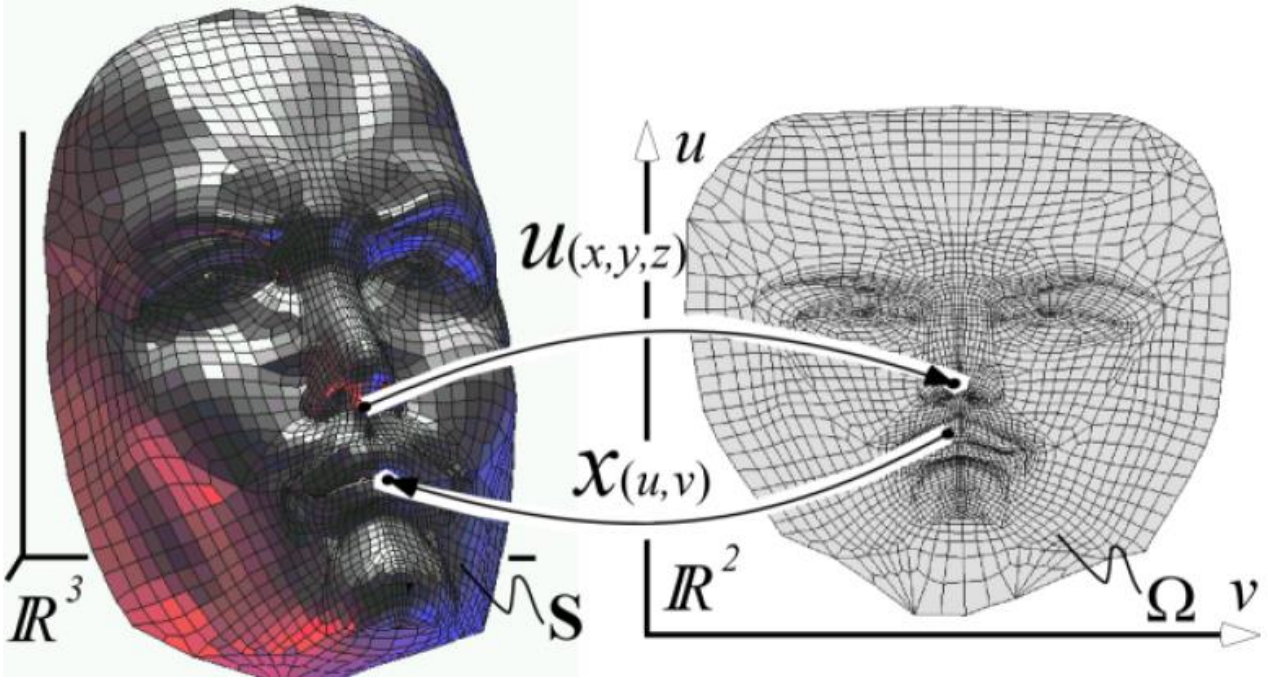


Mesh parameterization

Goal: Find a 2D parameterization of 3D surface.

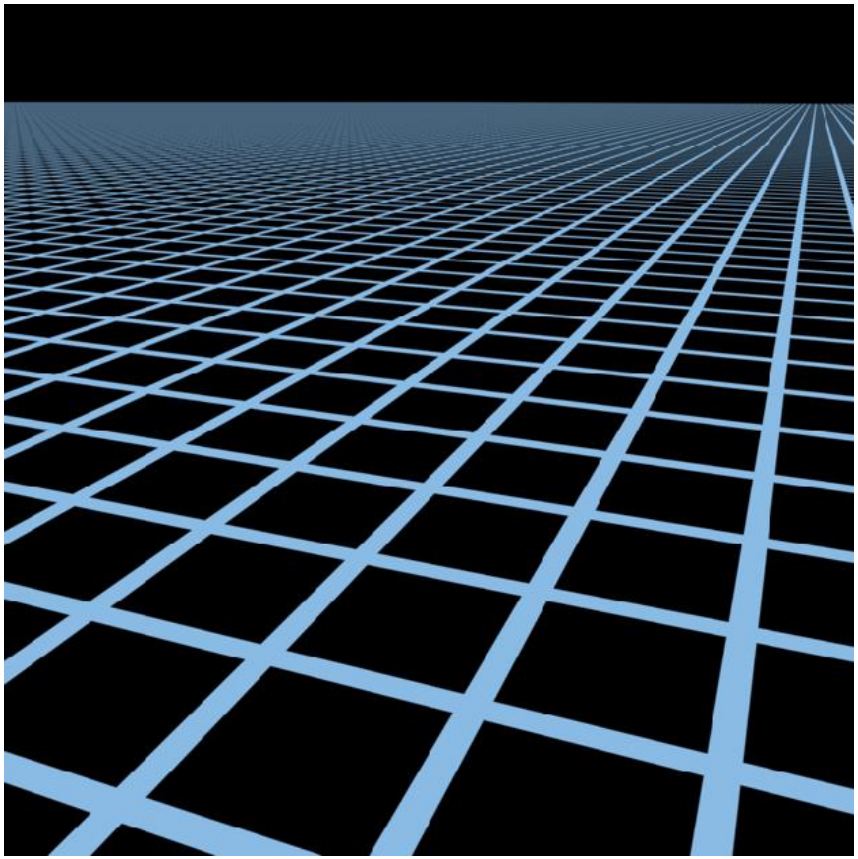
Common techniques include:

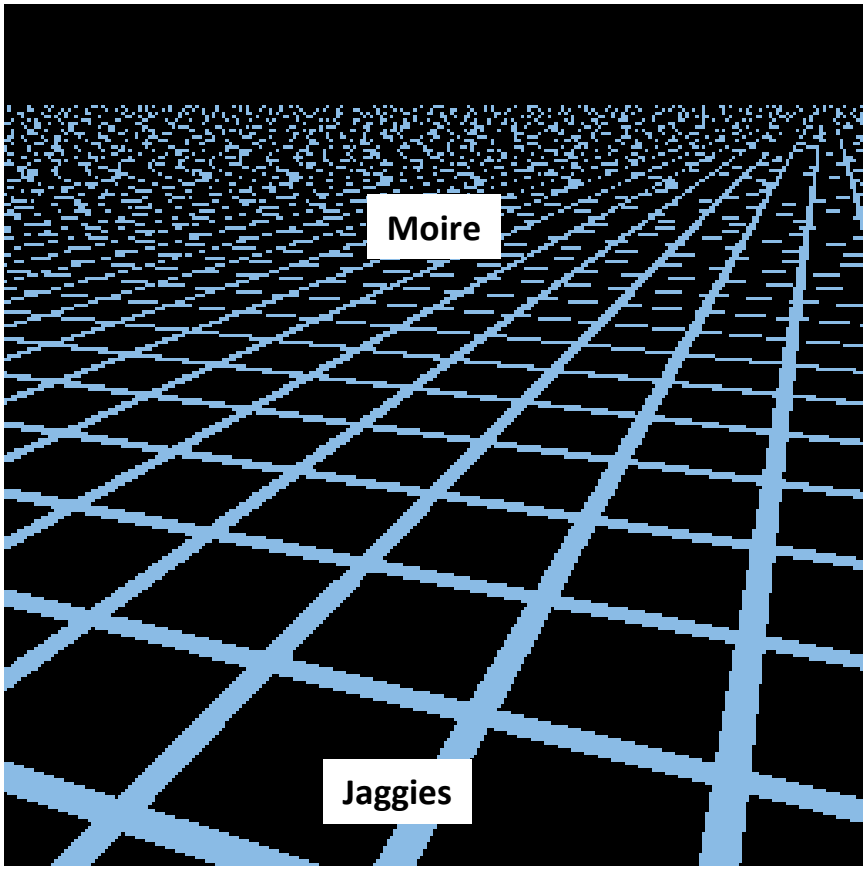
- Seam cutting,
- Planar embedding,
- Chart packing / atlas generation.



Texture sampling

- Aliasing of textures is a major problem.
- Texture coordinates rarely map exactly to the center of a texel.

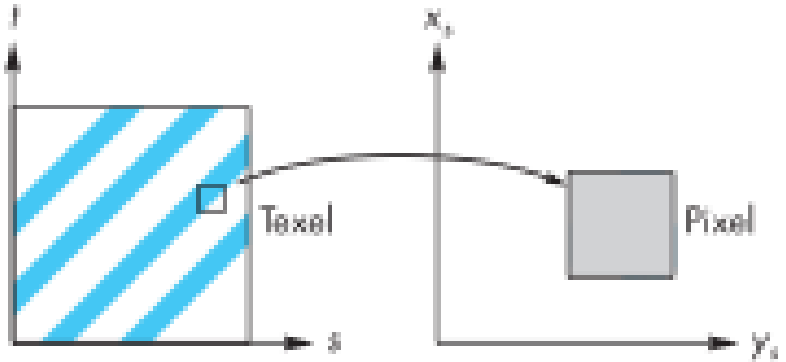




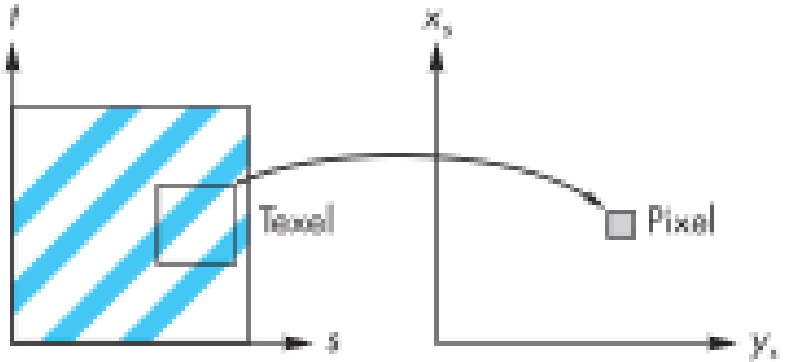
Real-time rendering, Haines et al.

Texture sampling

- Size of pixel on the screen may be smaller or larger than one texel.
- Smaller (Minification): One pixel sample per multiple texel samples.
- Larger (Magnification): Multiple pixel samples per texel sample.

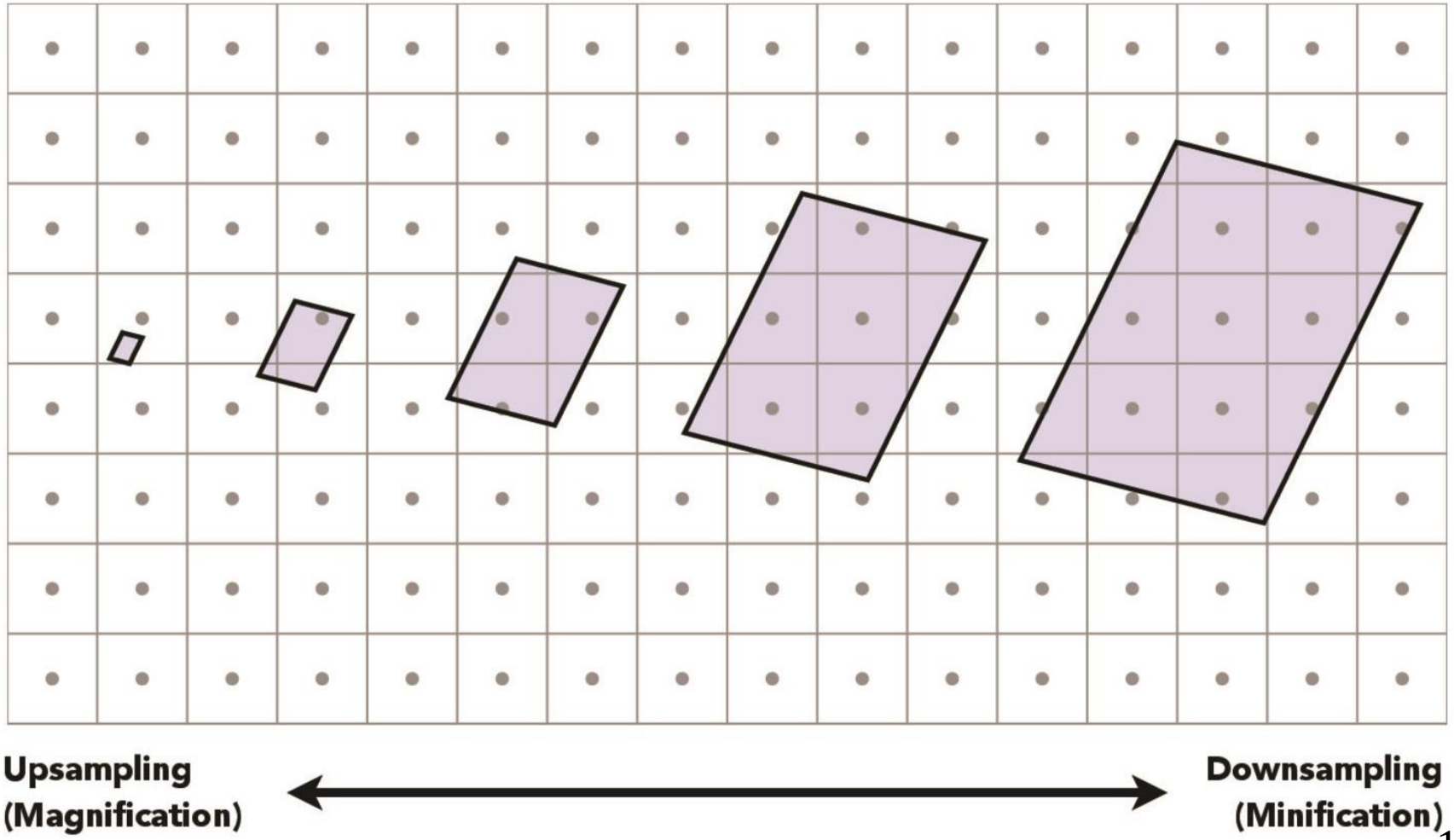


Minification



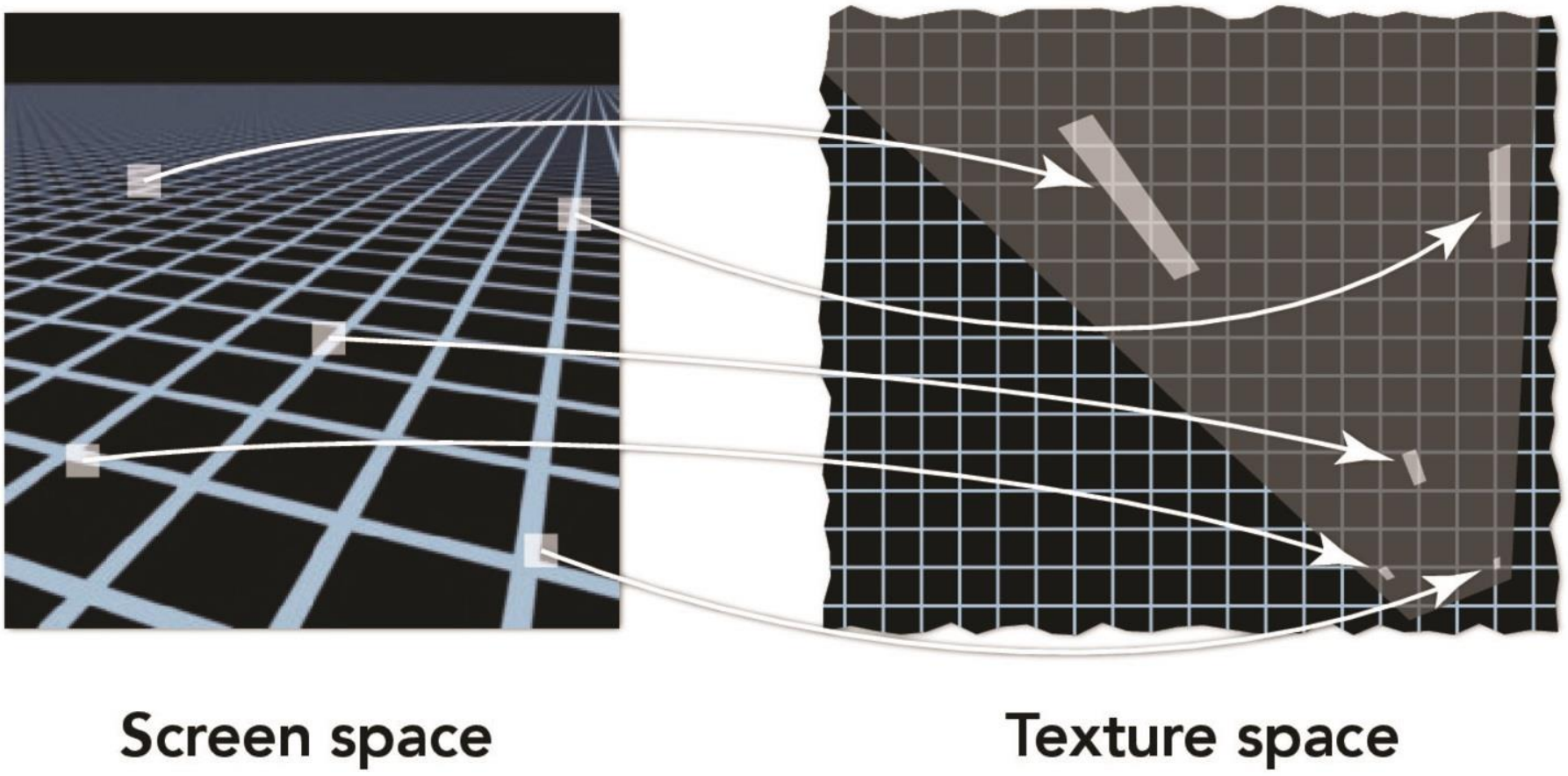
Magnification

Screen pixel area vs texel area



Real-time rendering, Haines et al.

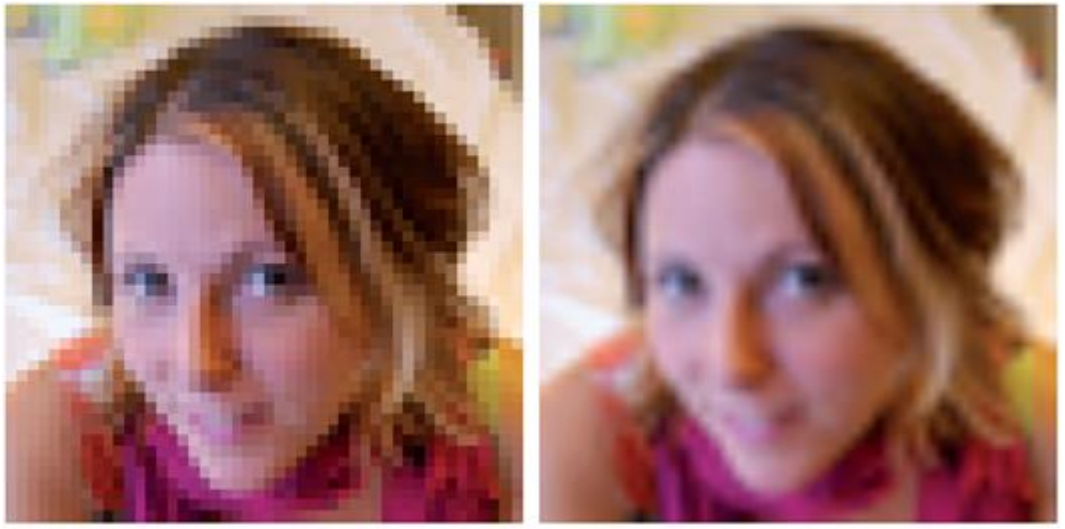
Screen pixel area vs texel area



Real-time rendering, Haines et al.

Texture magnification

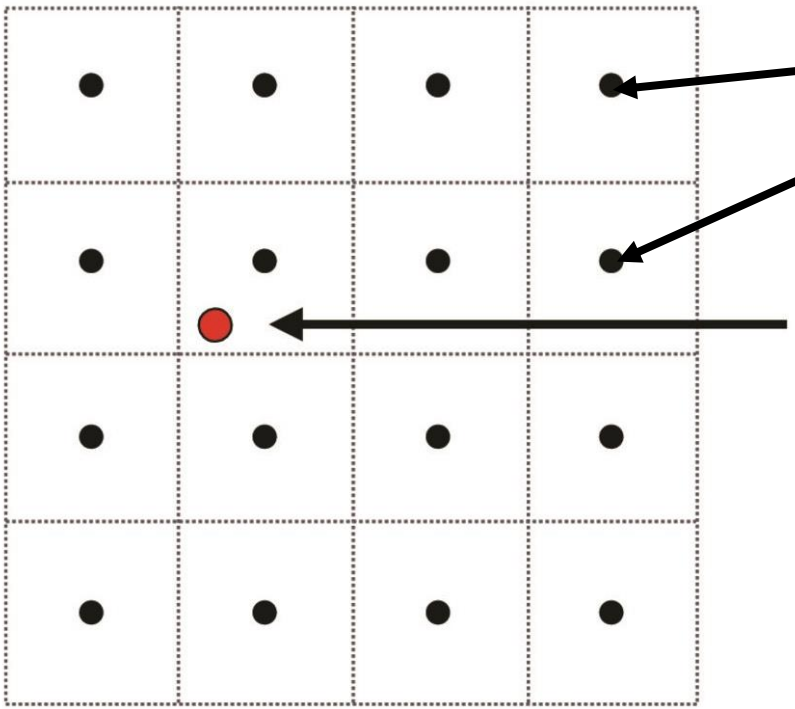
Several pixels correspond to a single texel, e.g. magnification of a 48x48 texture onto 320x320 pixels



Nearest Bilinear

Real-time rendering, Haines et al.

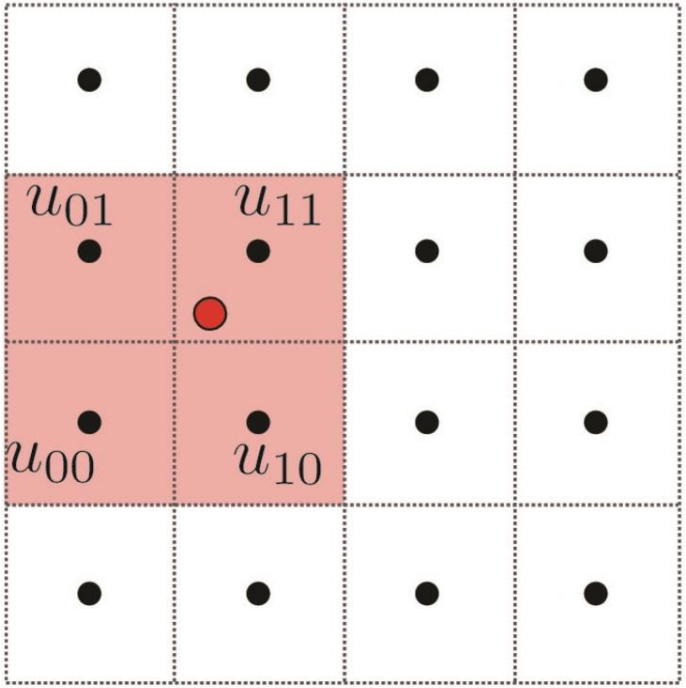
Bilinear filtering



Texture sample locations

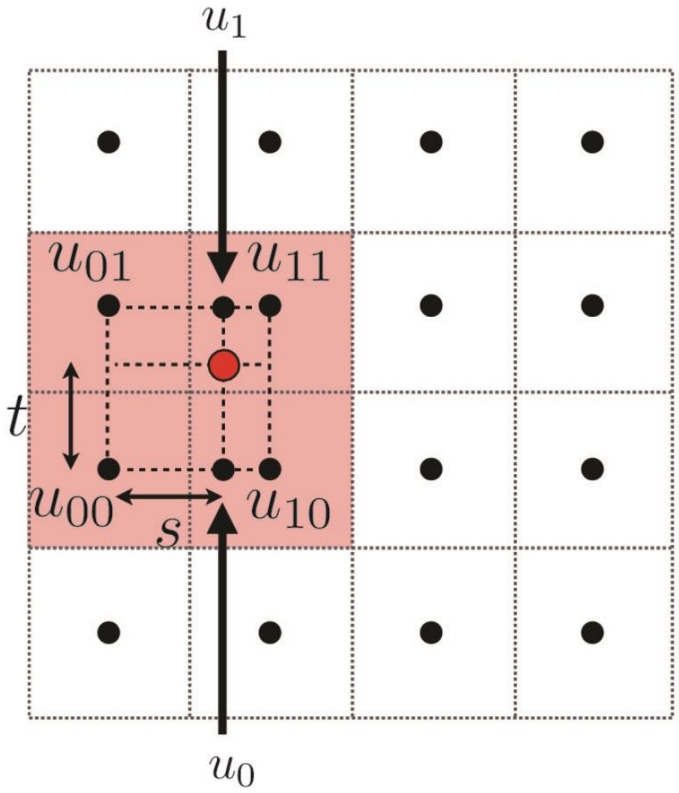
Point at which we want to sample the texture value

Bilinear filtering



Take the 4 nearest neighbors in texture space ($u_{00}, u_{10}, u_{11}, u_{01}$)

Bilinear filtering



The sample value is: $lerp(t, u_0, u_1)$, where:

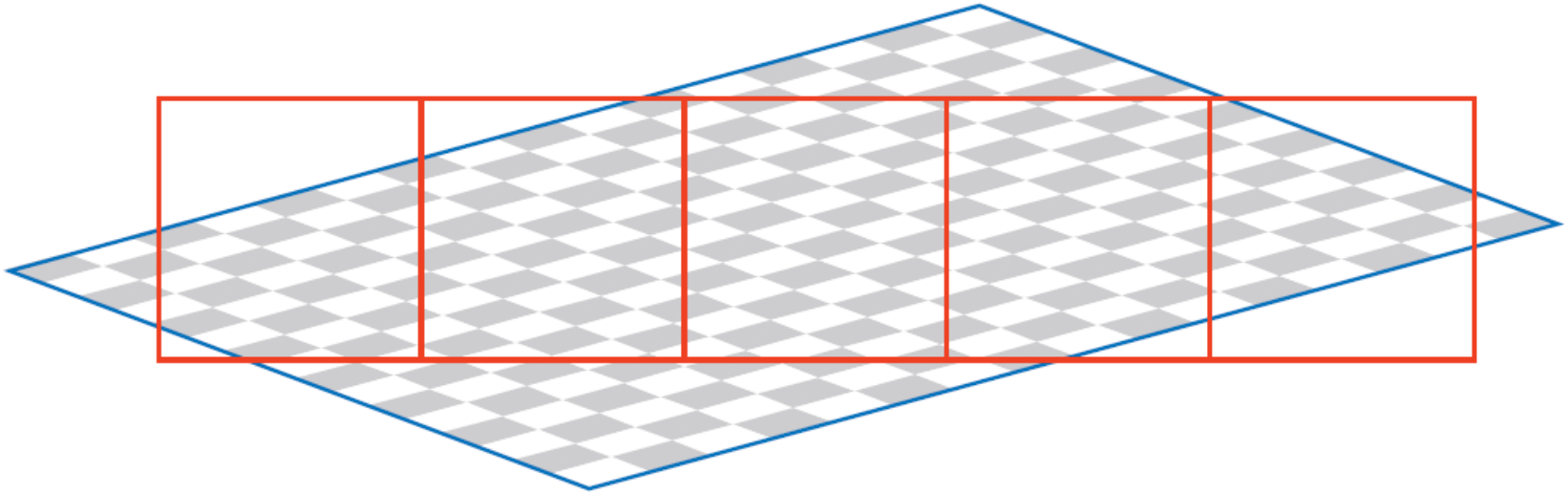
$u_0 = lerp(s, u_{00}, u_{10})$

$u_1 = lerp(s, u_{01}, u_{11})$

```
float
lerp(float x, float v0, float v1) {
 return v0 + x*(v1-v0);
}
```

Texture minification

When a textured shape is zoomed out, several texels correspond to a pixel. Which one to use?



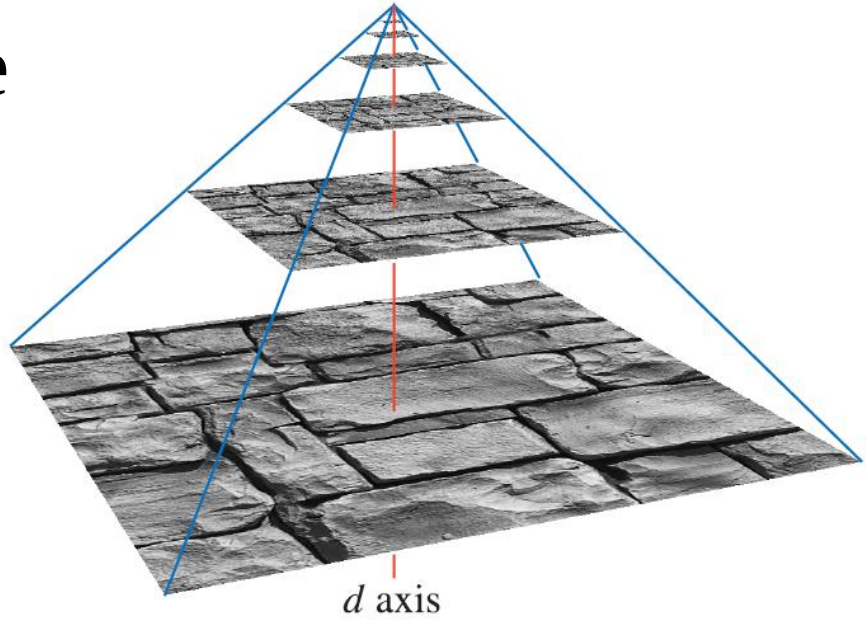
Texture minification

- Many texels can correspond to a one pixel area
- Shape of pixel footprint can be complex
- Solution:
  - Low-pass filter combined with down sampling the texture
  - Use a texture resolution matching screen sampling rate

Mipmapping

Instead of storing only one image, store multiple images at different resolutions.

Depending on the amount of minification, determine which image to fetch from.



Mipmapping

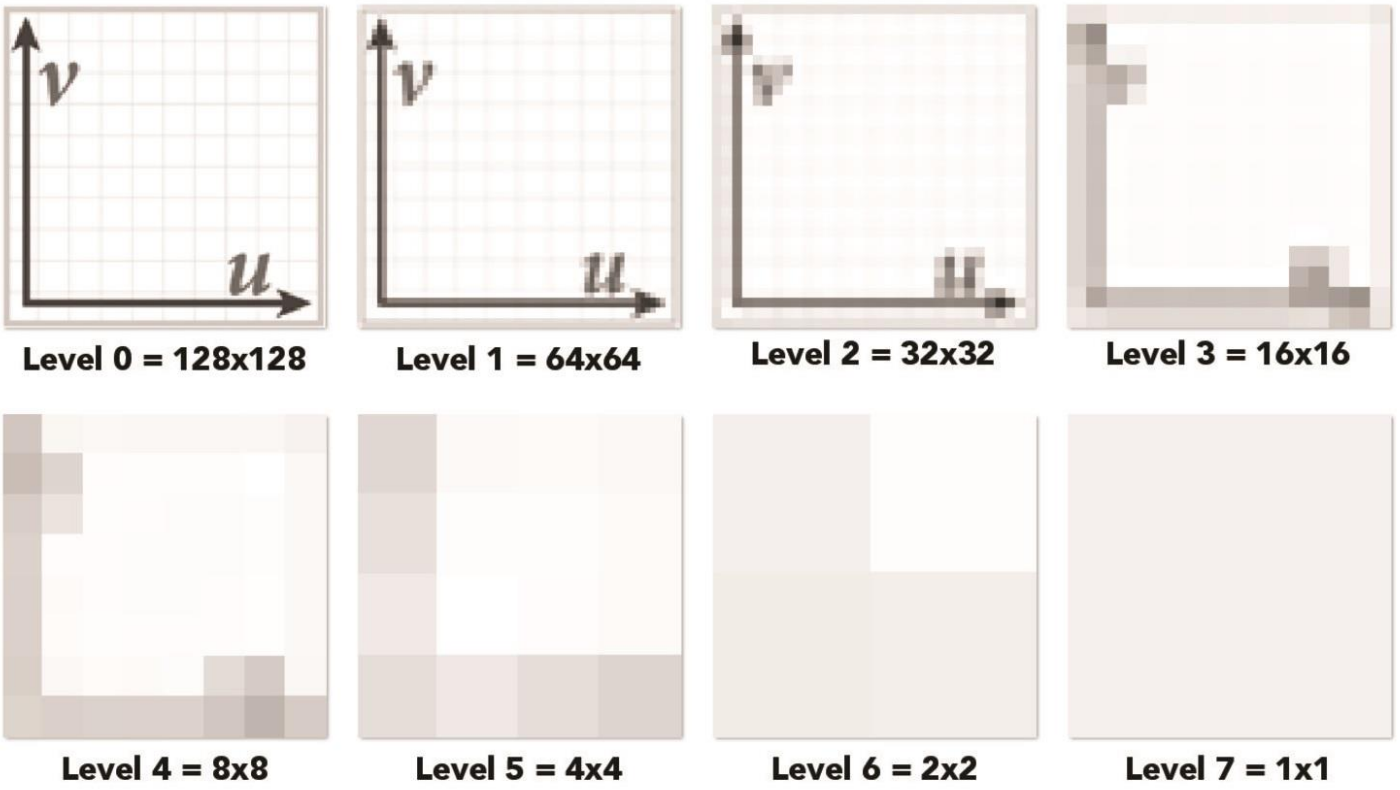
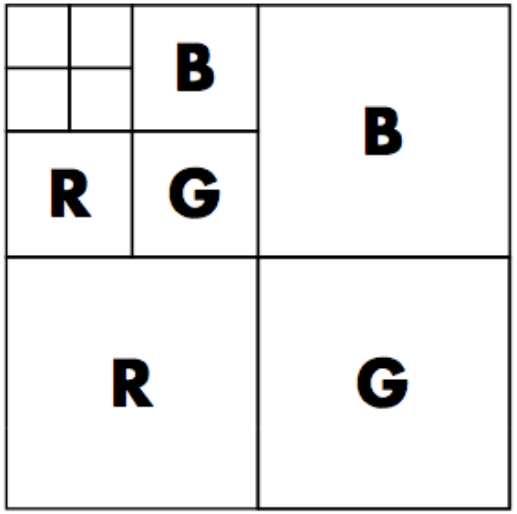
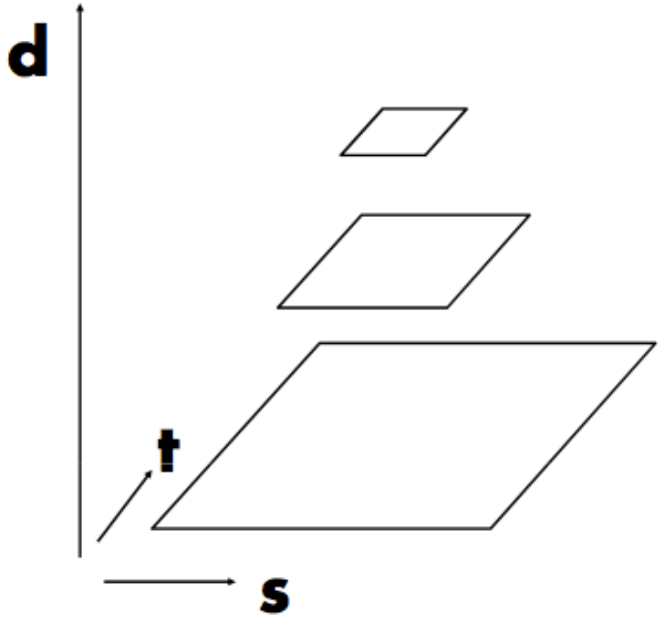


Image credit: learn.graphics

Mipmapping



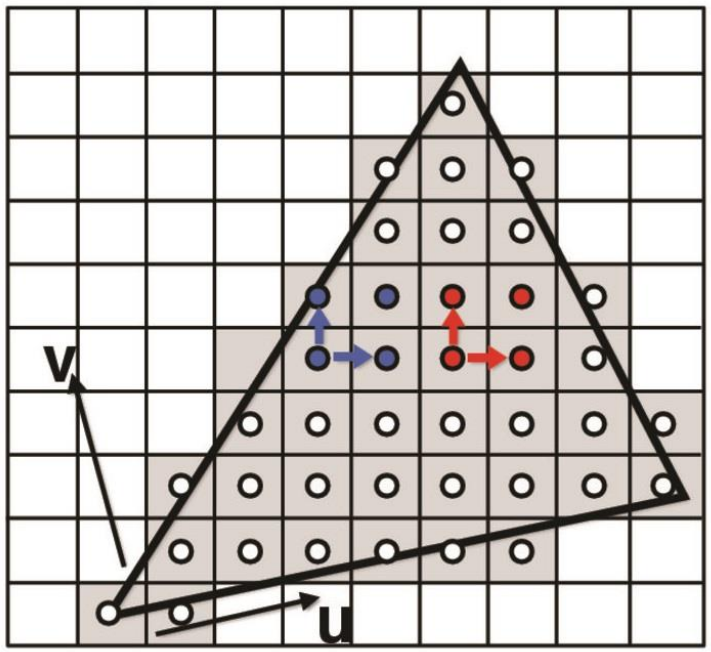
Efficient layout in memory

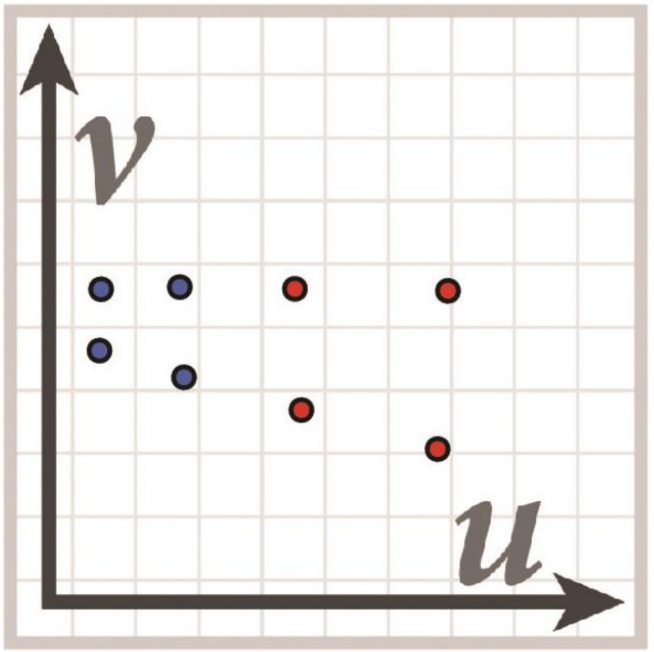


Hierarchy of texture at different resolution (d)

Image credit: learn.graphics

Computing the mipmap level



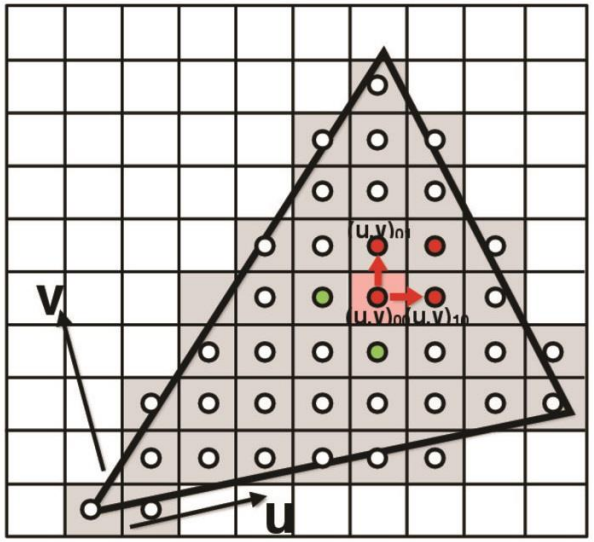


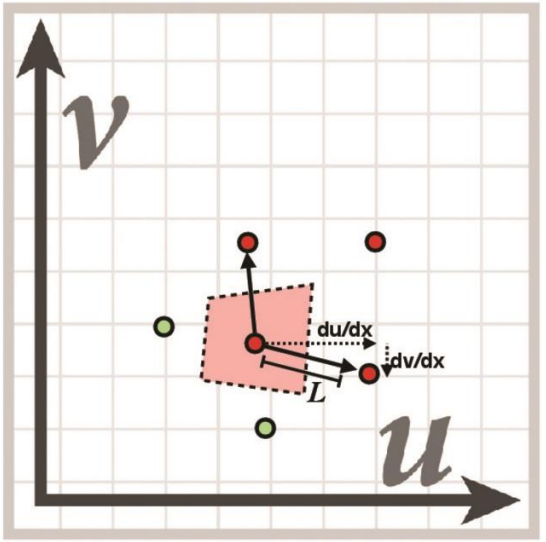
Screen space **Texture space**

Estimate the level by using texture coordinates of neighboring screen samples

Image credit: learn.graphics

Computing the mipmap level





$$L = max\left(\sqrt{\left(\frac{du}{dx}\right)^2 + \left(\frac{dv}{dx}\right)^2}, \sqrt{\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2}\right)$$

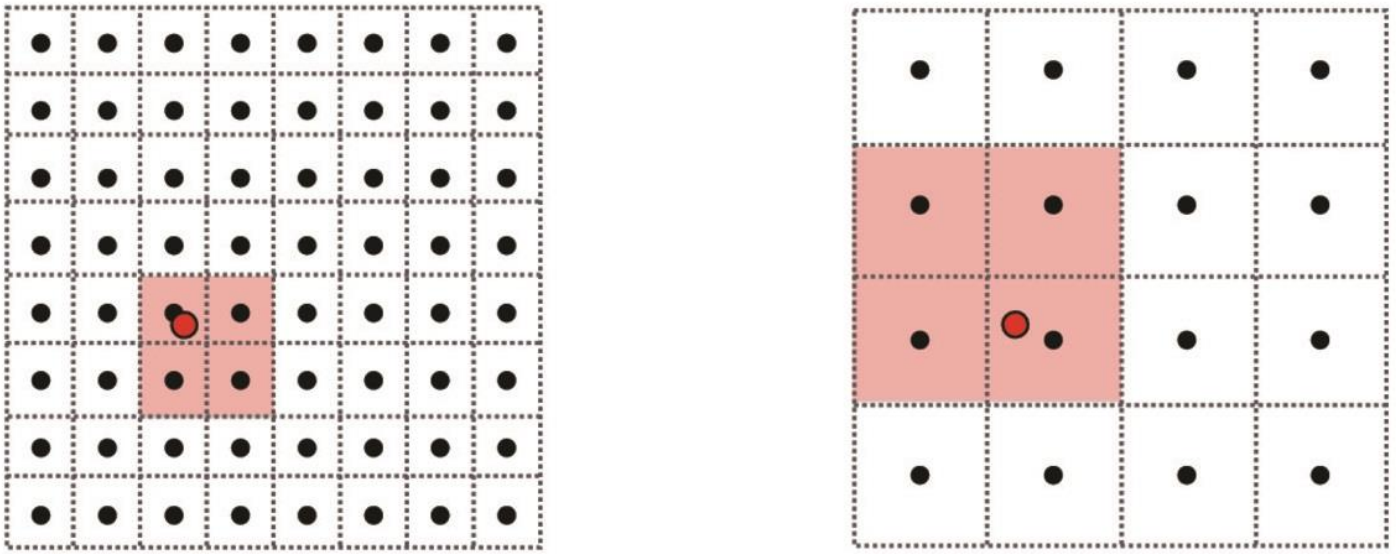
$$d = \log_2 L$$

Image credit: learn.graphics

How to use mipmapping

- Find the nearest level, and perform bilinear filtering in this level
- Find the nearest two levels (d and (d+1)), perform bilinear filtering in each, and then linearly interpolate between the two

Trilinear filtering

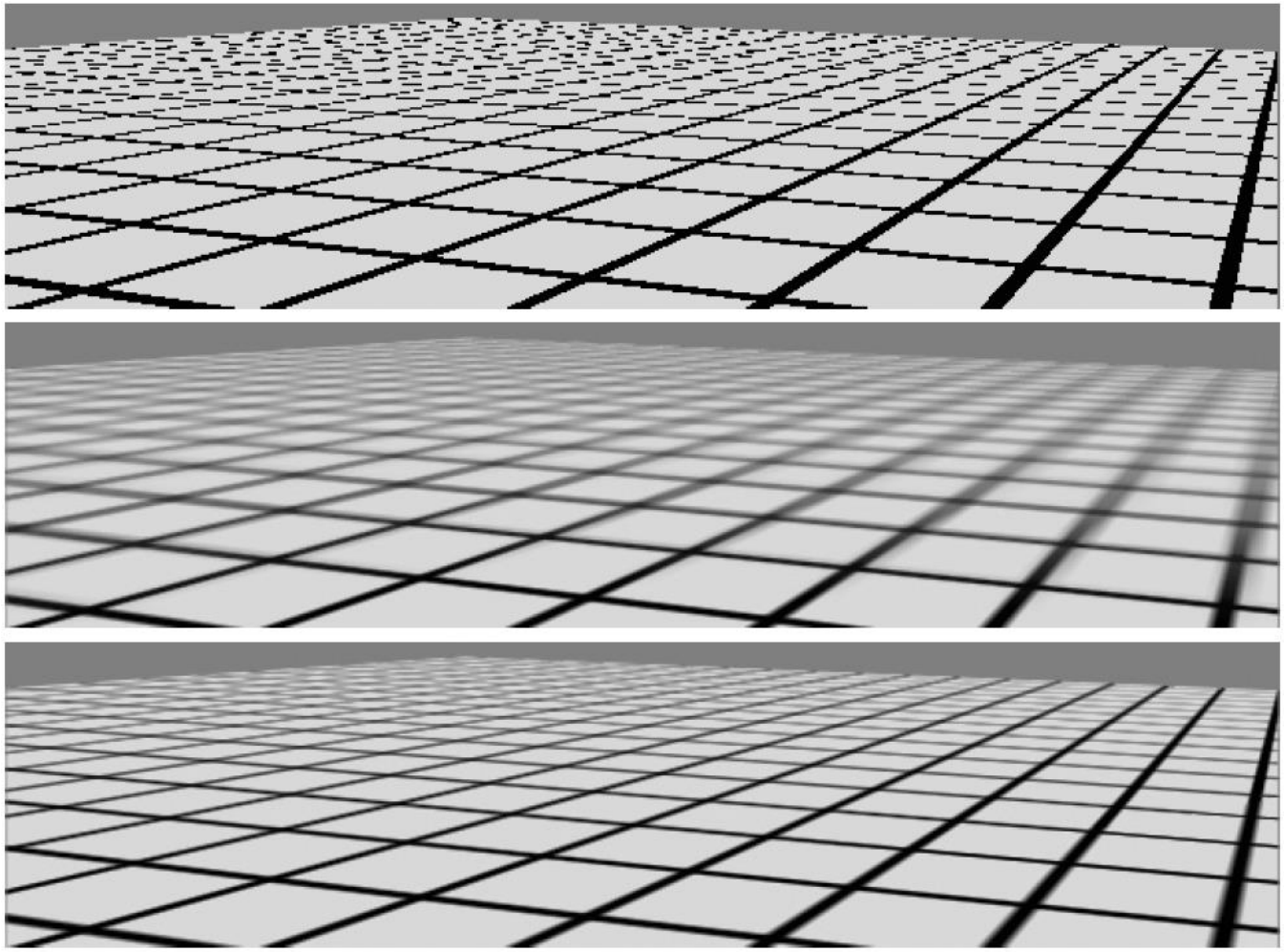


Level d
Bilinear filtering

Level d+1
Bilinear filtering

Linearly interpolate based on continuous values of d

Mipmapping



No filtering

Mip-mapping

Summed-area tables

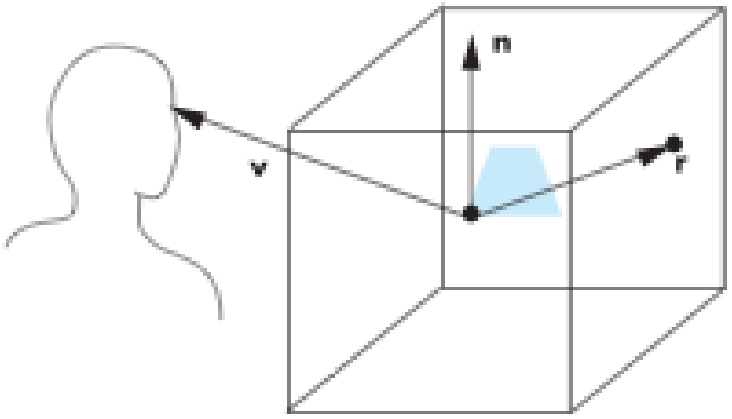
Real-time rendering, Haines et al.

Environment mapping

Variant of texture mapping for approximating specular reflections mirroring the environment.

Generate a spherical/cubic map of the environment of an object.

Texture values are dynamically reflected off the surface patch.



Spherical mapping

Let r be the reflected view vector (from origin to vertex) in eye coordinates, and n the normal at the vertex (also in eye coordinates)

$$\boldsymbol{r} = \boldsymbol{u} - 2\boldsymbol{n}(\boldsymbol{n}.\boldsymbol{u})$$

Let $m = 2\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}$

The texture coordinates are then given by

$$s = \frac{r_x}{m} + 0.5$$

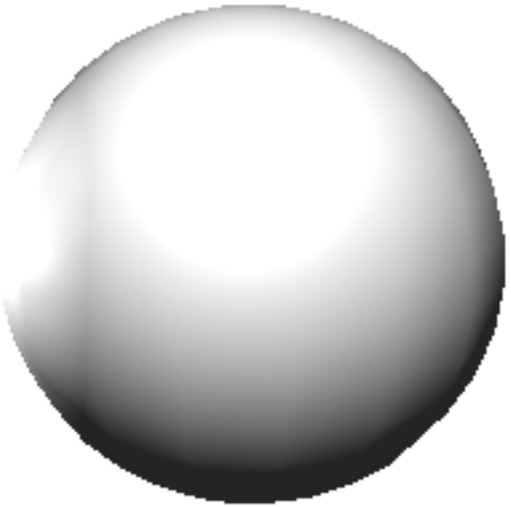
$$t = \frac{r_y}{m} + 0.5$$

Bump mapping

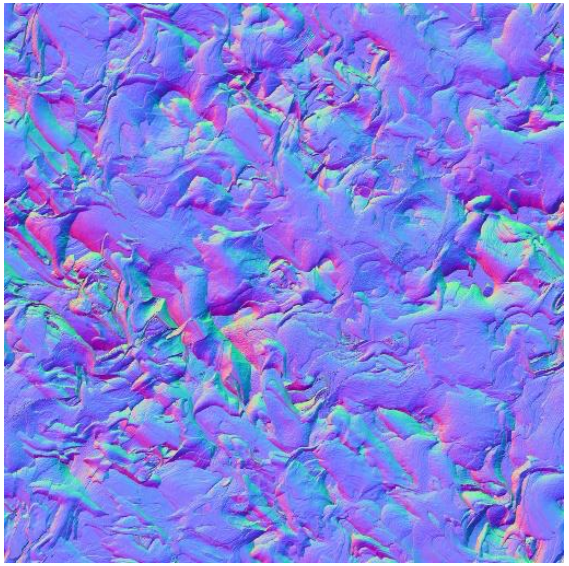
Many parts of the lighting calculations depend on surface normals.

We can store perturbations to normals in a texture map.

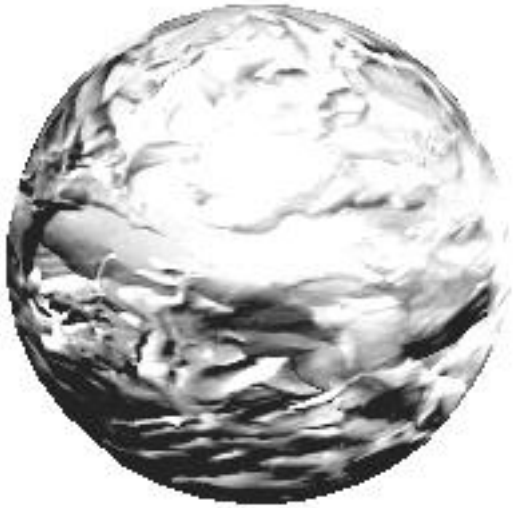
Shading gives then appearance of a bumpy surface.



Sphere under Phong illumination



Normal map



Sphere with normal map

Texture mapping in OpenGL

To use textures in OpenGL:

1. Load/Create number of images.
2. Generate texture objects. (glGenTextures)
3. Specify a texture image. (glTexImage2D)
4. Specify texture parameters (filtering, color blending).
   (glTexParameter{if}, glTexEnv{if})
5. Enable texture mapping. (glEnable)
6. Specify which texture to be used. (glBindTexture)
7. Specify texture coordinates corresponding to each vertex.
   (glTexCoord2{fd})

1. Load/Create images

Pixel colors of images are loaded/created and stored in arrays (usually array of GLubyte). Texture will be created by passing these arrays to OpenGL API functions.

```
#define IMAGE0_WIDTH  64  //number of pixels in width dir
#define IMAGE0_HEIGHT 64  //number of pixels in height dir
GLubyte aImage0[IMAGE0_WIDTH*IMAGE0_HEIGHT*4];//RGBA

#define IMAGE1_WIDTH  256
#define IMAGE1_HEIGHT 256
GLubyte aImage1[IMAGE1_WIDTH*IMAGE1_HEIGHT*3];//RGB

//create an image and store it to "aImage0"
//[user should define the function]
MakeImage();

//load an image and store it to "aImage1"
//[user should define the function]
LoadImage();
```

2. Generation of texture objects

Generate texture objects; *n* = number of textures,
textureIDs = pointer to the variable/array to store texture objects

```
glGenTextures(GLsizei n, GLuint *textureIDs)
```

Delete texture objects : clean up textures when they become unnecessary

```
glDeleteTextures(Glsizei n,
                 const GLuint *textureIDs)
```

If 2 images are created/loaded by the programmer,
the following code will create objects for the 2 textures

```
#define NUM_TEXTURE 2
GLuint naTexID[NUM_TEXTURE];

glGenTextures(NUM_TEXTURE, naTexID);
```

and the following code is used when to release the texture objects

```
glDeleteTextures(NUM_TEXTURE, naTexID);
```

3. Specify a texture image

Specify the array (image) to be used as a two-dimensional texture

```
glBindTexture(GLenum target, GLunit textureName)
```

```
glTexImage2D(GLenum target, GLint level,
   GLint iformat, GLsizei width, GLsizei height,
   GLint border, GLenum format, GLenum type,
   GLvoid *tarray);
```

```
target: GL_TEXTURE_2D
textureName: texture object to be assigned to image
level: Level of mipmapping
iformat: how the image is stored in texture memory
          (GL_RGB(24bit color) or GL_RGBA(32bit color))
width, height: size of image
border: No longer used and should be set to 0
format: specify how the pixels in the image in processor memory
        are stored(GL_RGB or GL_RGBA)
type: specify how how the pixels in the image in processor
      memory are stored (GL_UNSIGNED_BYTE, etc)
tarray: pointer to the image data
```

3. Specify a texture image

```
//assign naTexID[0] to the image "aImage0"
glBindTexture(GL_TEXTURE_2D, naTexID[0])
glTexImage2D(GL_TEXTURE2D, 0, GL_RGBA,
             IMAGE0_WIDTH, IMAGE0_HEIGHT, 0, GL_RGBA,
             GL_UNSIGNED_BYTE, aImage0);

//assign naTexID[1] to the image "aImage1"
glBindTexture(GL_TEXTURE_2D, naTexID[1])
glTexImage2D(GL_TEXTURE2D, 0, GL_RGB,
             IMAGE1_WIDTH, IMAGE1_HEIGHT, 0, GL_RGB,
             GL_UNSIGNED_BYTE, aImage1);
```

Note : Mipmap generation

```
int gluBuild2DMipmaps(GLenum target,
          GLint iformat, GLsizei width, GLsizei height,
          GLenum format, GLenum type,
          const GLvoid *tarray);
```

Constructs a series of mipmaps and calls `glTexImage2D` to load the images. Parameters are exactly the same as those for `glTexImage2D()`. A value of 0 is returned if all of the bitmaps are constructed successfully. Use this instead of `glTexImage2D` to use mipmapping.

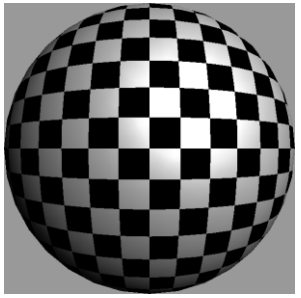
4. Texture and shading

Select how to mix texture with polygon color/shading

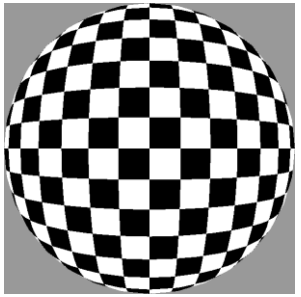
```
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
          GLenum mode);
```

mode can be: `GL_MODULATE`, `GL_DECAL`, `GL_REPLACE`, or `GL_BLEND`

`GL_MODULATE` : mix texture with lighting/shading color

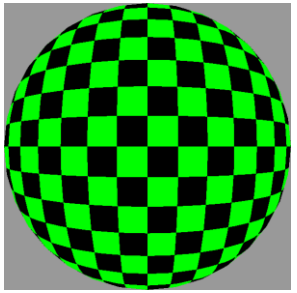


`GL_DECAL` : replace the object color by texture
transparency is taken into account



`GL_REPLACE` : replace the object color by texture
transparency is ignored

`GL_BLEND` : mix texture with the object color
(glColor3d(0, 1, 0) is specified in this example)



4. Texture filtering

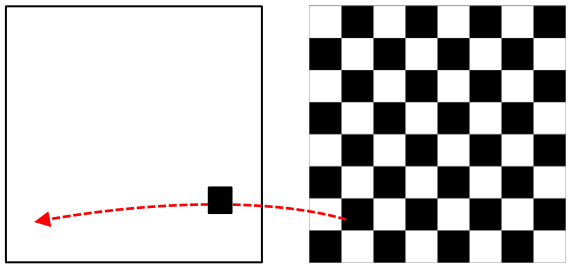
Specify filtering method to be used:

```
glTexParameteri(GL_TEXTURE_2D,GLenum pname,GLenum param);
```

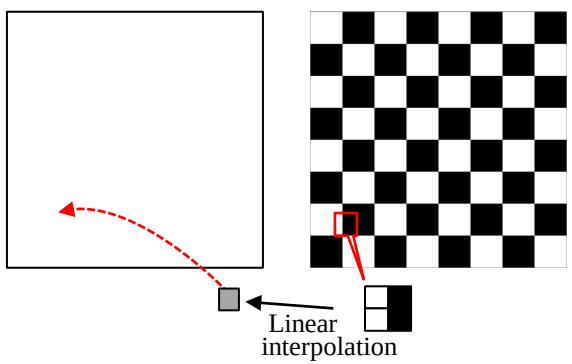
pname: `GL_TEXTURE_MIN_FILTER` or `GL_TEXTURE_MAG_FILTER`

param: `GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_LINEAR`

GL_NEAREST



GL_LINEAR



For minification:

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

For magnification:

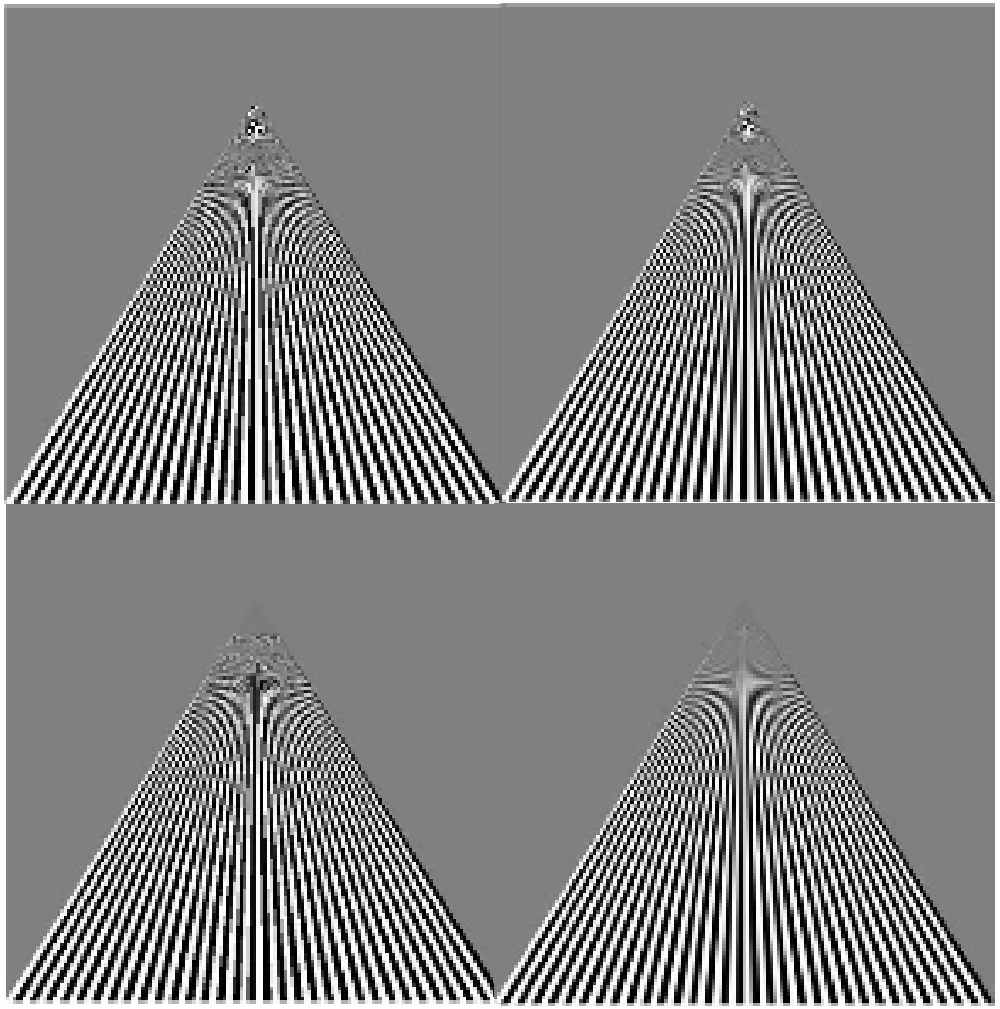
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                GL_LINEAR);
```

4. Specify texture parameters

Parameters (for minification only):

- GL_NEAREST_MIPMAP_NEAREST: use the best resolution and the closest point
- GL_NEAREST_MIPMAP_LINEAR: perform filtering (linear interpolation) within the best mipmap (resolution)
- GL_LINEAR_MIPMAP_NEAREST: point sampling using linear filtering between mipmap
- GL_LINEAR_MIPMAP_LINEAR: apply both filters

Mipmapping parameters



Top left: point sampling
Top right: linear filtering
Bottom left: mipmapping point sampling
Bottom right: mipmapping linear filtering

4. Wrapping

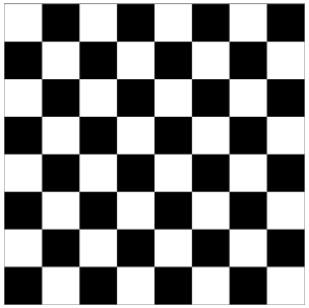
Specify how to extrapolate textures when texture coordinates "0.0<s,t" or "s,t>1.0"

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
                GLenum mode);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
                GLenum mode);
```

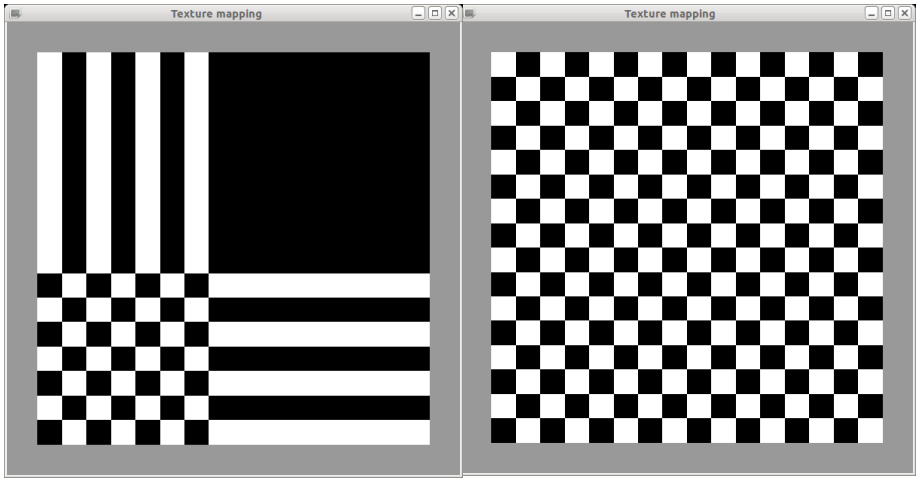
mode can be: `GL_CLAMP` (keep value of the edge) or `GL_REPEAT` (repeat the texture)

If we draw a rectangle using the following code and texture.

```
glBegin(GL_QUADS);
  glTexCoord2d(0.0, 0.0);
  glVertex3d(-1.0, -1.0, 0.0);
  glTexCoord2d(2.0,  0.0);
  glVertex3d( 1.0, -1.0, 0.0);
  glTexCoord2d(2.0,  2.0);
  glVertex3d( 1.0,  1.0, 0.0);
  glTexCoord2d(0.0,  2.0);
  glVertex3d(-1.0,  1.0, 0.0);
glEnd();
```



The result using GL_CLAMP and GL_REPEAT will be:



GL_CLAMP GL_REPEAT

5. Enable texture mapping and
6. Specify the texture object to be used

Enabling texture

```
glEnable(GL_TEXTURE_2D);
```

Disabling texture

```
glDisable(GL_TEXTURE_2D);
```

Use different textures for drawing objects

```
glEnable(GL_TEXTURE_2D);

glBindTexture(GL_TEXTURE_2D, naTexID[0]);
DrawObject1(); //Draw using texture associated with naTexID[0]

glBindTexture(GL_TEXTURE_2D, naTexID[1]);
DrawObject2(); //Draw using texture associated with naTexID[1]

glDisable(GL_TEXTURE_2D);
DrawObject3(); //Draw without texture
```

Example: Texture mapping

```
#include <GLUT/glut.h> // #include <GL/glut.h> on Solaris
#include <stdlib.h>
#include <stdio.h>
#define IMAGE_WIDTH  (64)
#define IMAGE_HEIGHT (64)

static GLubyte checkImage[IMAGE_HEIGHT*IMAGE_WIDTH*4];
static GLuint texName[1];

void makeCheckImage(void) {
   int i, j, c;
   for (i = 0; i < IMAGE_HEIGHT; i++) {
      for (j = 0; j < IMAGE_WIDTH; j++) {
         c = ((((i&0x8)==0)^((j&0x8))==0))*255;
         checkImage[i][j][0] = (GLubyte) c;
         checkImage[i][j][1] = (GLubyte) c;
         checkImage[i][j][2] = (GLubyte) c;
         checkImage[i][j][3] = (GLubyte) 255;
      }
   }
}
```

Init: create texture

```
void init(void) {
  glClearColor (0.5, 0.5, 0.5, 0.0);
  glShadeModel(GL_FLAT);
  glEnable(GL_DEPTH_TEST);
  glPixelStorei(GL_UNPACK_ALIGNMENT, 1); // row alignment of pixels

  glGenTextures(1, texName);
  makeCheckImage();
  glBindTexture(GL_TEXTURE_2D, texName[0]);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                  GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                  GL_NEAREST);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, IMAGE_WIDTH,
         IMAGE_HEIGHT, 0, GL_RGBA, GL_UNSIGNED_BYTE, checkImage);
}
```

Display: map texture (glTexCoord)

```
void display(void) {
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   glEnable(GL_TEXTURE_2D);
   glBindTexture(GL_TEXTURE_2D, texName[0]);

   glBegin(GL_QUADS);
   glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
   glTexCoord2f(0.0, 1.0); glVertex3f(-2.0, 1.0, 0.0);
   glTexCoord2f(1.0, 1.0); glVertex3f(0.0, 1.0, 0.0);
   glTexCoord2f(1.0, 0.0); glVertex3f(0.0, -1.0, 0.0);

   glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
   glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
   glTexCoord2f(1.0, 1.0); glVertex3f(2.41421,1.0,-1.41421);
   glTexCoord2f(1.0, 0.0); glVertex3f(2.41421,-1.0,-1.41421);
   glEnd();

   glFlush();
   glDisable(GL_TEXTURE_2D);
}
```