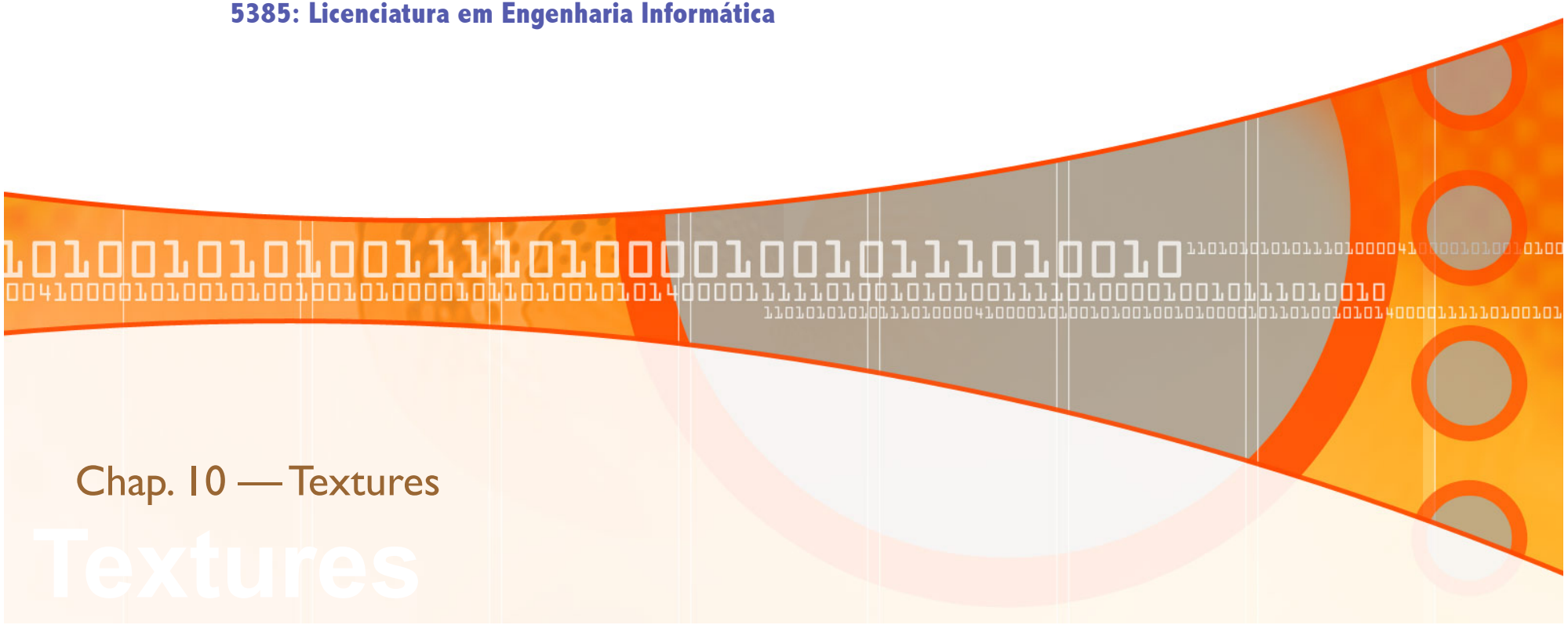
Computação Gráfica

5385: Licenciatura em Engenharia Informática



Chap. 10 — Textures

Textures

Outline

...:

- Objectives and motivation
- Notion of texture, texture mapping, texture patterns and texels
- Texture mapping on polygons, texture interpolation
- Wrapping modes
- Filtering modes
- Blending modes
- Mapping textures on geometric objects
  - planar mapping
  - cylindrical mapping
  - spherical mapping
  - box mapping

Objectivos

To introduce mapping methods:

- Texture mapping
- Environmental mapping
- Bump mapping
- Light mapping

They will not be taught!

We consider two basic strategies:

- Manual specification of coordinates
- Two-stage automated mapping

It will not be taught!

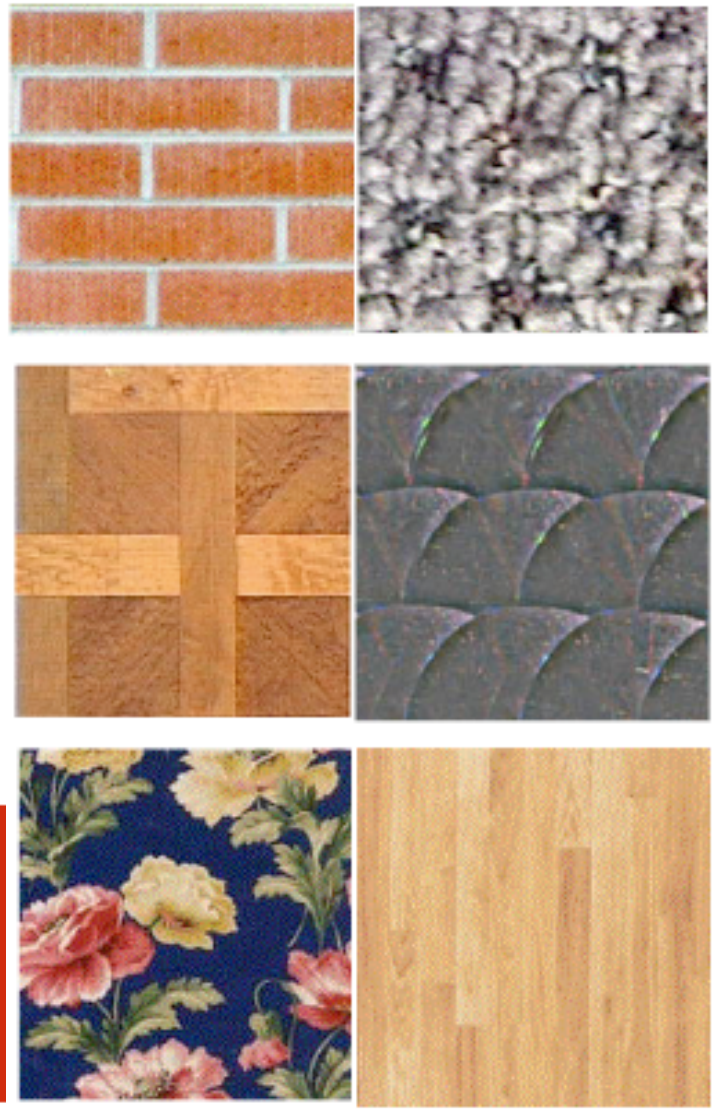
Notion of texture

Definition:

- It is an image with RGBA components.

Texture mapping:

- *A method to create complexity in an image without the overhead of building large geometric models.*

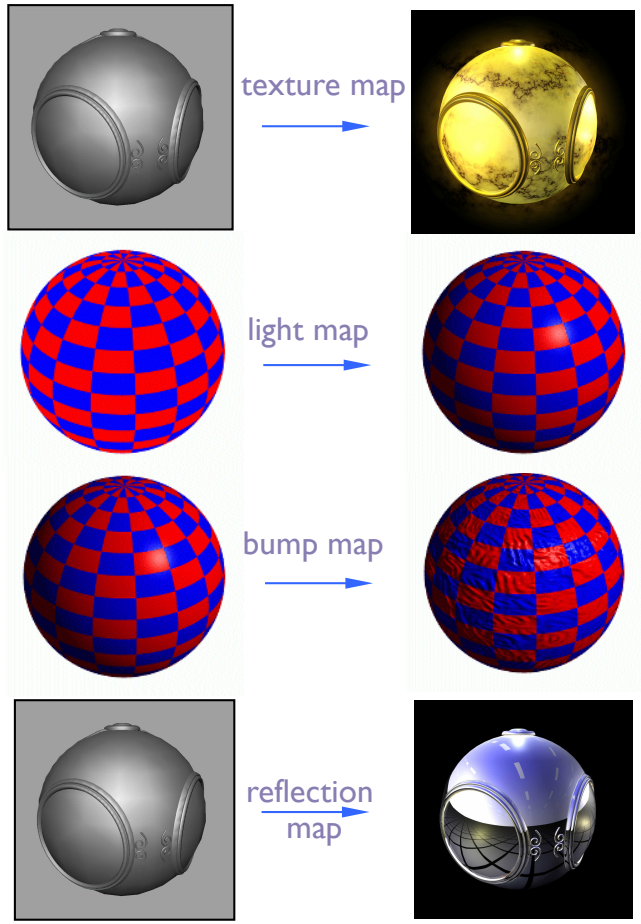


Motivation: adding realism

- *Objects rendered using Phong reflection model and Gouraud or Phong interpolated shading often appear rather 'plastic' and 'floating in air'*
- *Texture effects can be added to give more realistic looking surface appearance.*

Texturing techniques:

- ***Texture mapping***
  - Texture mapping uses pattern to be put on a surface of an object.
- ***Light maps***
  - Light maps combine texture and lighting through a modulation process
- ***Bump mapping***
  - Smooth surface is distorted to get variation of the surface
- ***Environmental mapping***
  - Environmental mapping (reflection maps) – enables ray-tracing like output

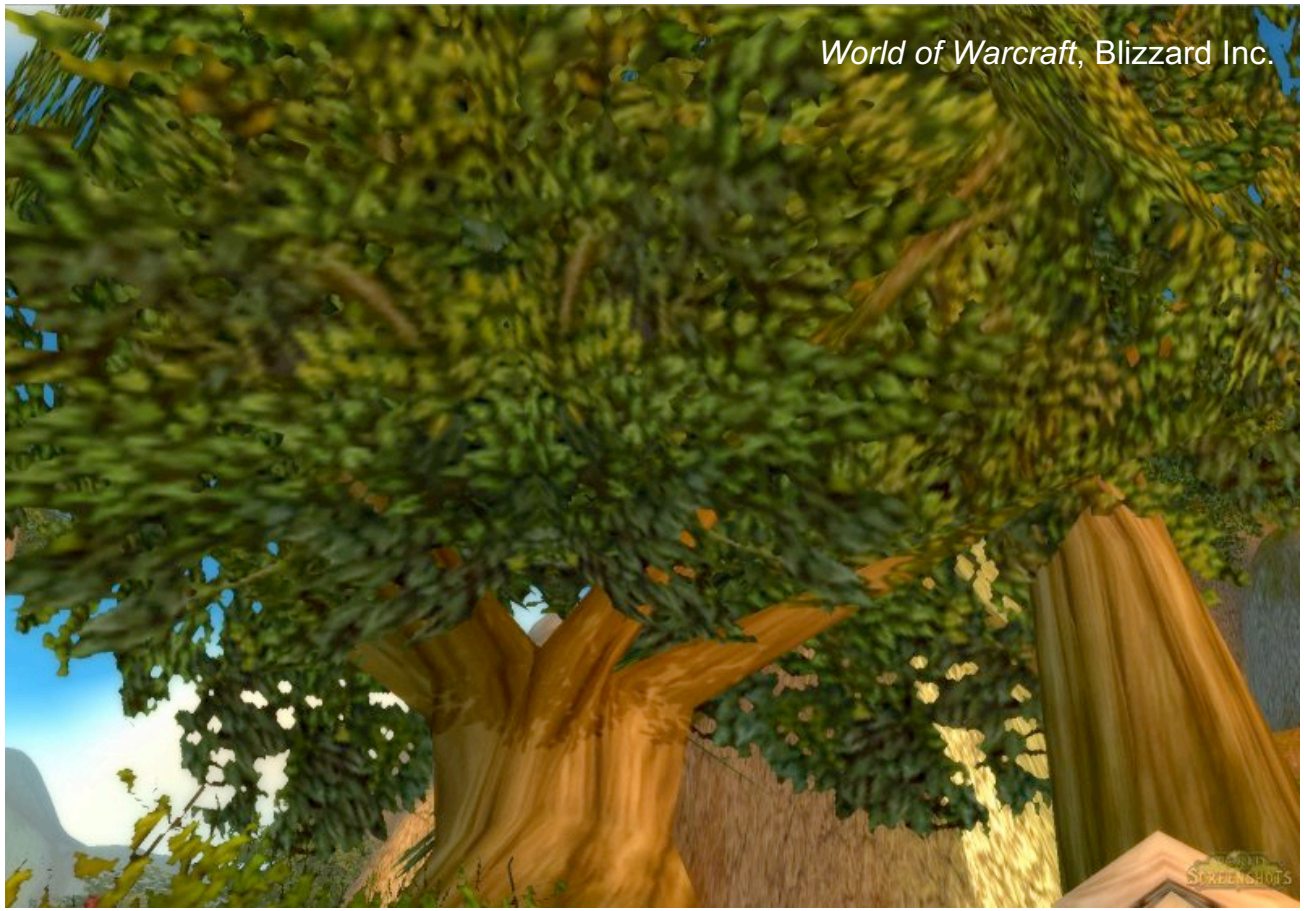


Motivation: add surface details...

- Texture mapping doesn't affect geometry processing, such as transformation, clipping, projection, …
- It does affect rasterization, which is highly accelerated by hardware.
- Textures can be easily replaced for more details: texture mod in games

Motivation: advantages

- More polygons (slow and hard to handle small details)
- Less polygons but with textures (much faster)



Motivation: adding surface detail

*The **obvious solution** is not the best:*

- breaking the scene into smaller and smaller polygons increases the detail.
- But it is very hard to model and very time-consuming to render.

*The **preferred solution** is texture mapping:*

- typically a 2D image 'painted' onto objects

Exemplos:

① – Model t-shirt with logo

- no need to model the letters and engine with triangles
- use large base polygon
- color it with the photograph

② – Subtle wall lighting

- No need to compute it at every frame
- No need to model it with a lot of constant color triangle
- Past photograph on large polygon

③ – Non-planar surfaces also work

- subdivide surface into planar patches
- assign photograph sub-regions to each individual patch

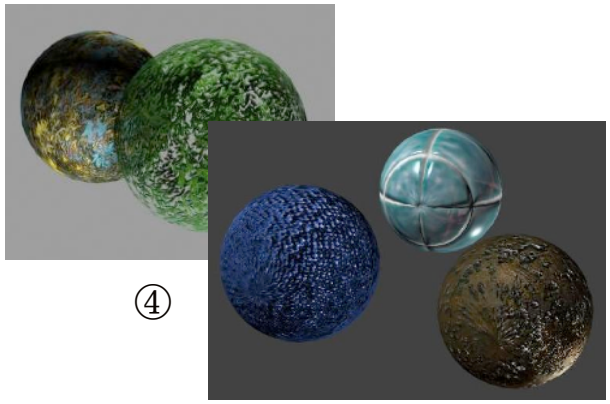
④ – Examples of modulating color, bumpiness, shininess, transparency with identical sphere geometry

①

②

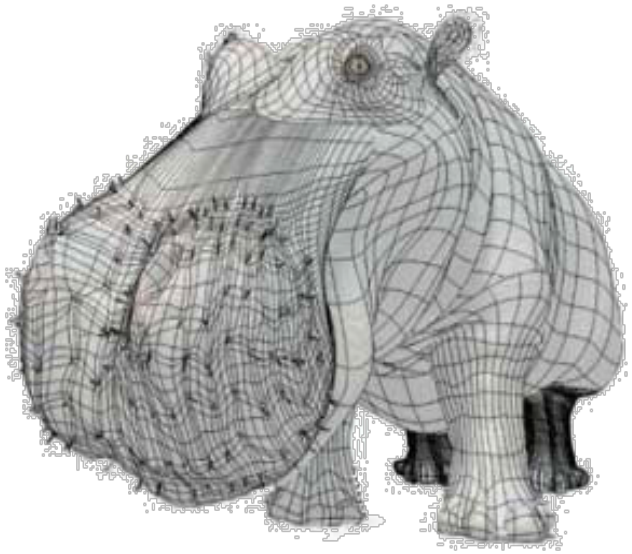
③

④



Textures:
at what point do things start to looking real?

- Surfaces "in the wild" are very complex
- Cannot model all the fine variations
- We need to find ways to add **surface detail**. How?



geometric model

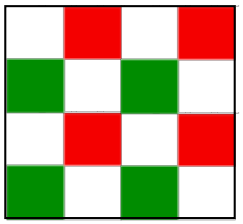
geometric model
+
shading

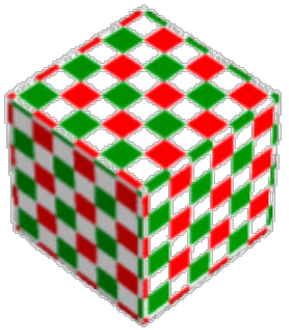
geometric model
+
shading
+
textures

Texture mapping, texture pattern, and texels

History:

- Developed by Catmull (1974), Blinn and Newell (1976), and others.





Texture mapping:

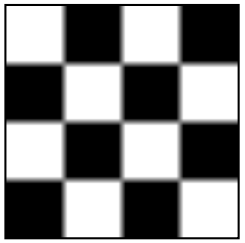
- adds surface detail by mapping texture patterns onto the surface.

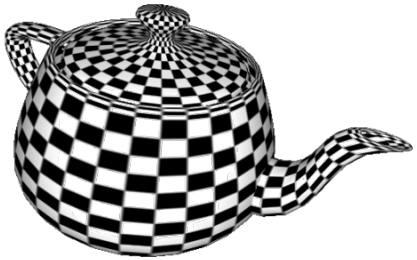
Texture pattern:

- Pattern is repeated. For example, the **texture pattern** for the cube aside is the following:

Texel: (short for "texture element")

- A **texel** is a pixel on a texture. For example, an 128x128 texture has 128x128 texels. On screen this may result in more or fewer pixels depending on how far away the object is on which the texture is used and also on how the texture is scaled on the object surface.





MAPPING TECHNIQUES

- Texture mapping
- *Environment mapping*
- *Bump mapping*
- *Light mapping*

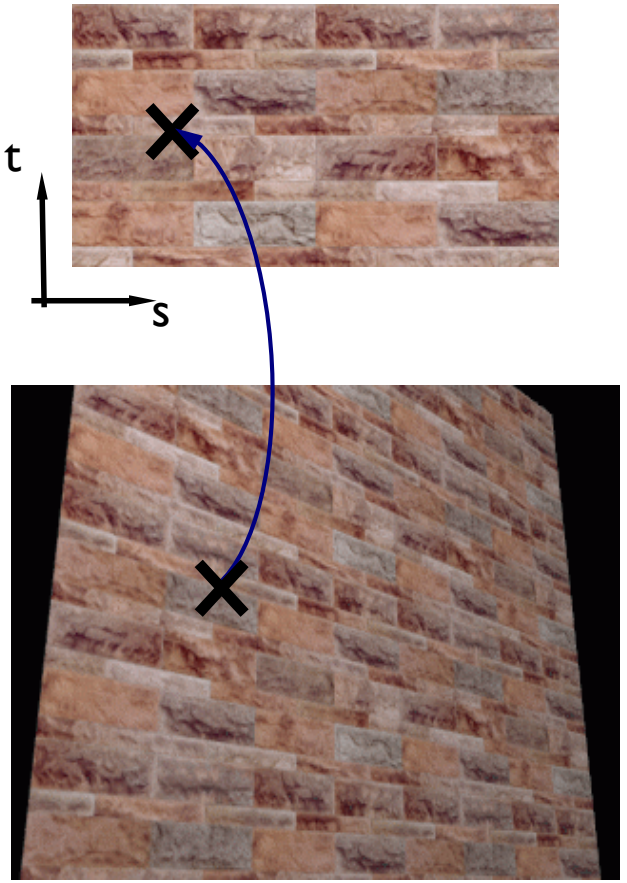
Texture Mapping

Question I: Which point of the texture do we use for a given point on the surface?

Question II: *And in the case of mapping a* ***texture*** *onto a polygon?*

Answer I:

- The texture is simply an image, with a 2D coordinate system (s,t).
  - Parameterize points in the texture with 2 coordinates: (s,t)
- Define the mapping from (x,y,z) in world space to (s,t) in texture space.
  - To find the color in the texture, take an (x,y,z) point on the surface, map it into texture space, and use it to look up the color of the texture.



Answer II:

- Specify (s,t) coordinates at vertices,
- Interpolate (s,t) for other points based on given vertices.

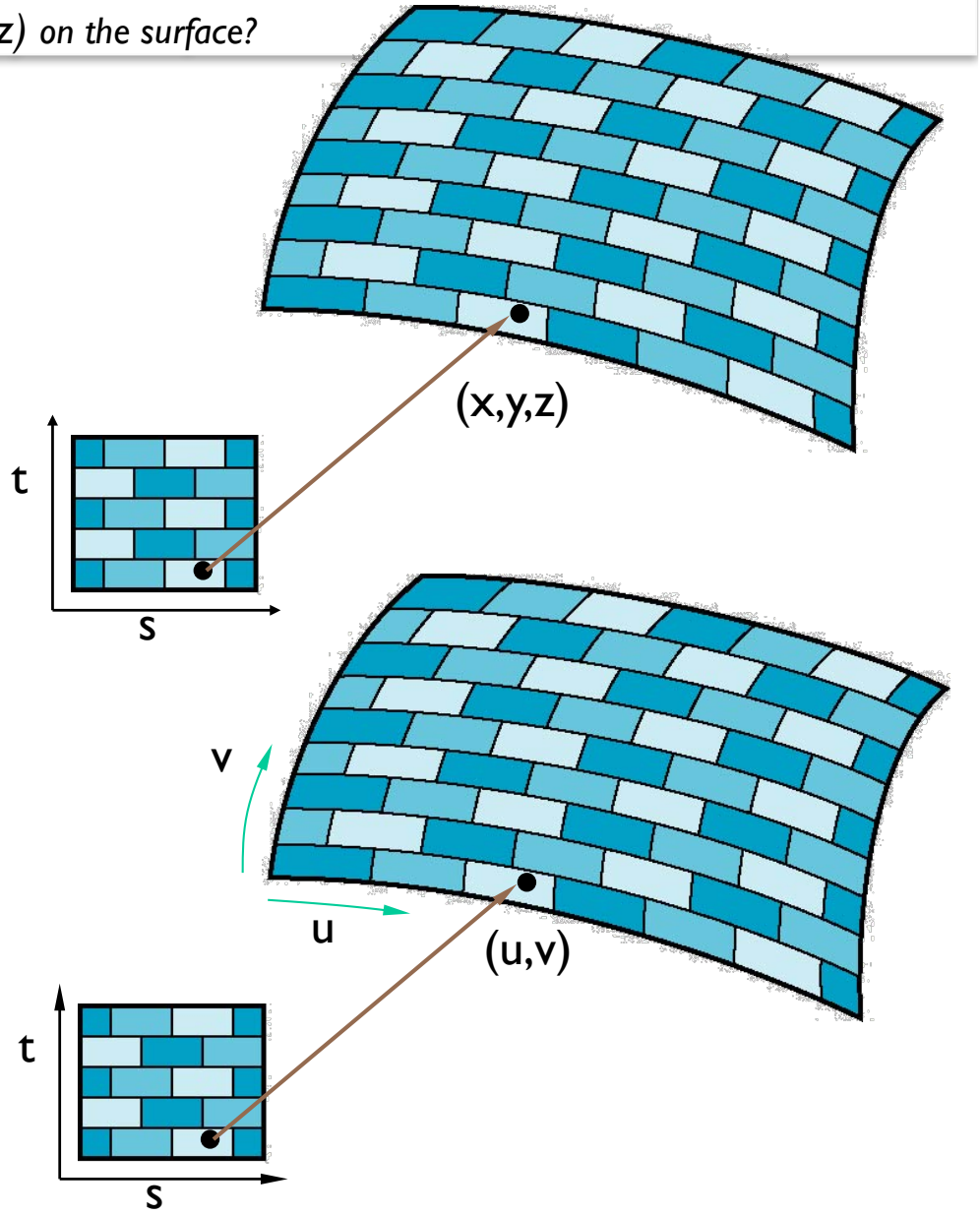
Texture to surface coordinate mapping

Question 1: The basic problem here is how to find the texture to surface mapping?

That is: Given a texture position (s,t), what is the position (x,y,z) on the surface?

Problem formulation:

- This problem requires 3 parametric functions to transforma a texel *(s,t)* into a Cartesian point *(x,y,z)* on the surface::
  - $x = \mathbf{X}(s, t)$
  - $y = \mathbf{Y}(s, t)$
  - $z = \mathbf{Z}(s, t)$
- Alternatively, we can use 2 parametric coordinate systems, the 2D image coordinates (s,t) and the 2D parameterization coordinates (u,v) that we assign to the 3D object



How to set (u,v) parametric coordinates?

- ***Manually***:

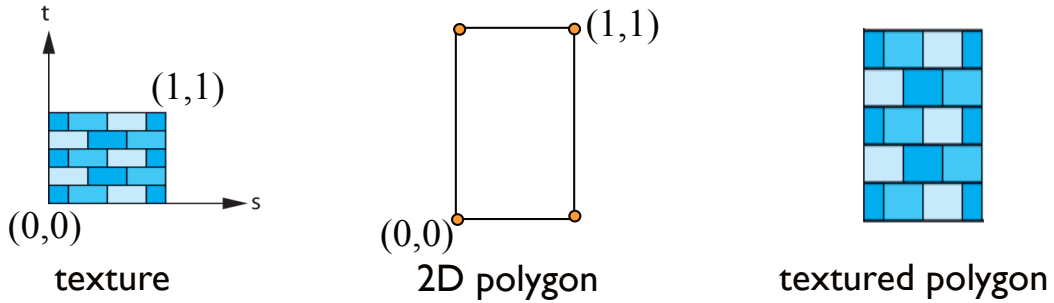
  Set the texture coordinates for each vertex ourselves

- ***Automatically***:

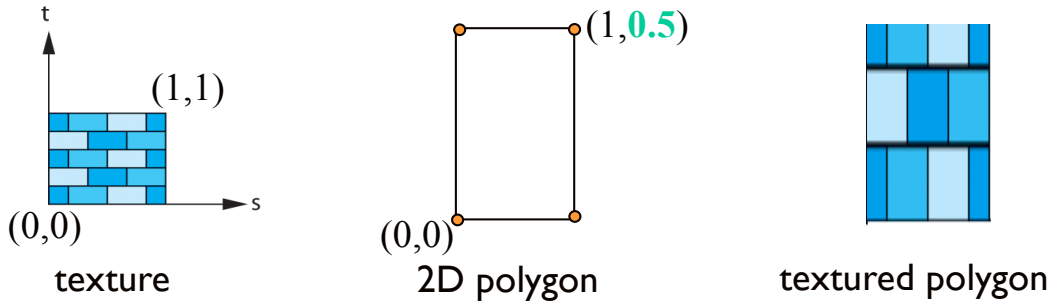
  Use an algorithm that sets the texture coordinates for us

Use an algorithm that sets the texture coordinates for us

- We can manually specify the texture coordinates at each vertex



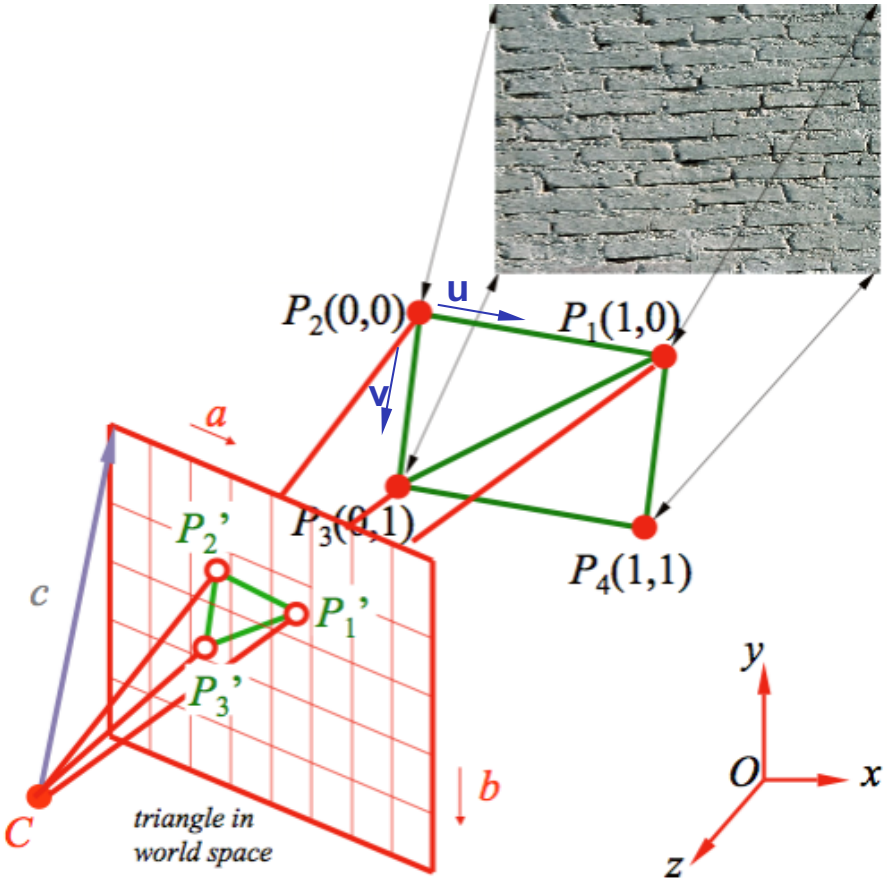
- We can chose alternate texture coordinates



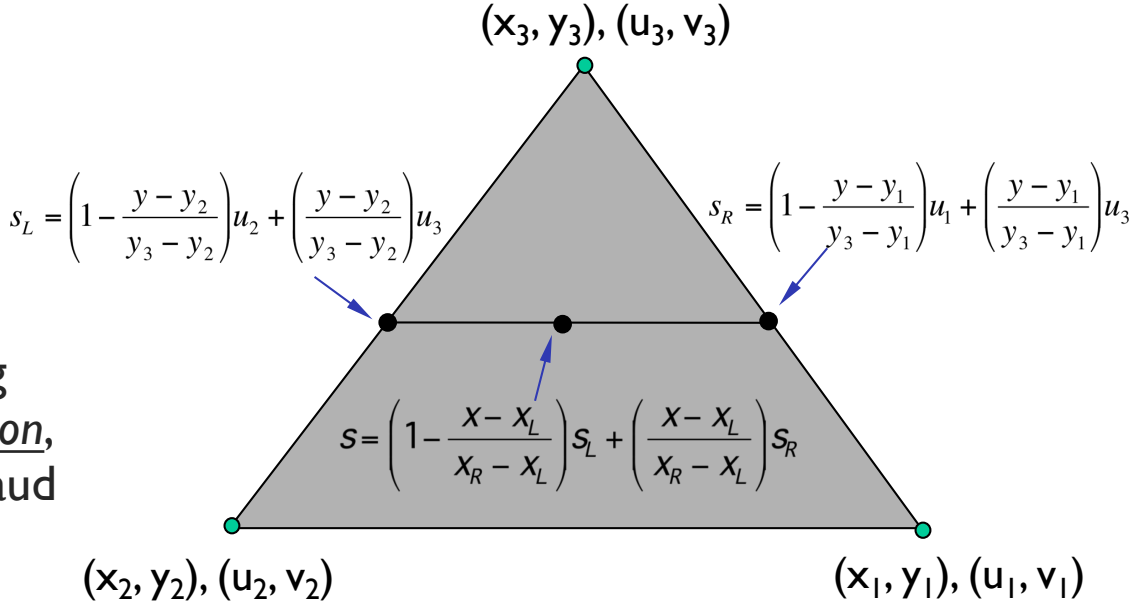
Mapping texture to polygons

Procedure:

- For polygon texture mapping, we explicitly define the (*u*,*v*) coordinates of the polygon vertices
- That is, we pin the texture at the vertices
- We interpolate within the triangle at the time of scan converting into screen space

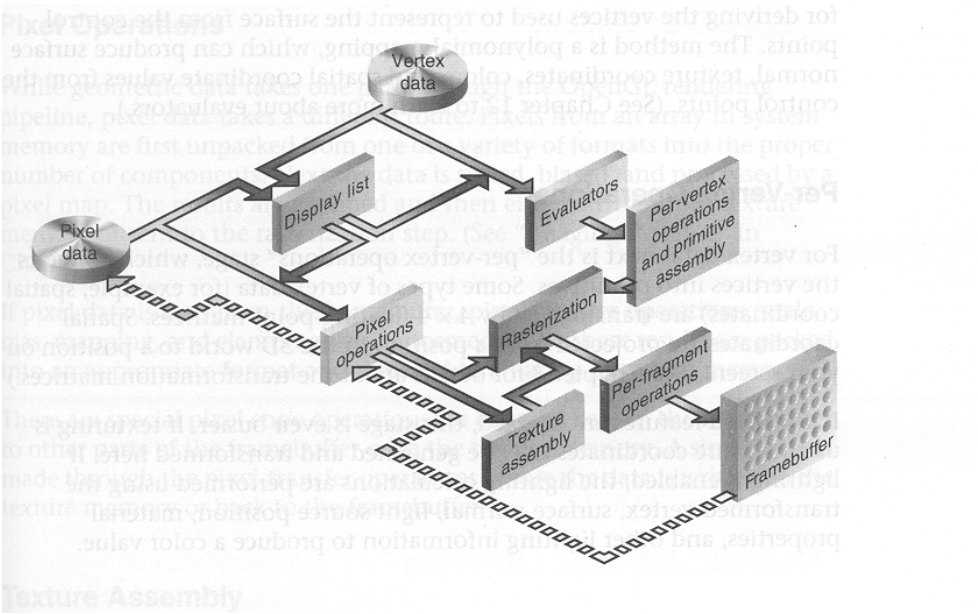


Texture interpolation



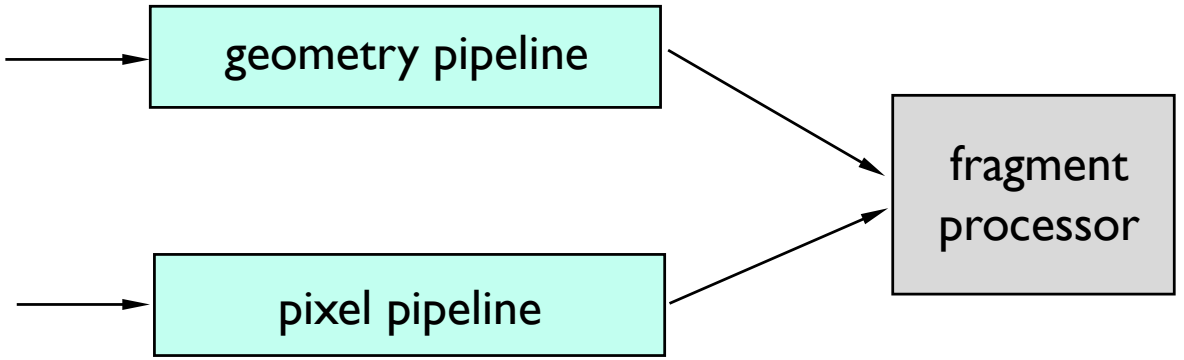
How is it done?:

- Interpolation is done during *rasterization* or *scan conversion*, similar as is done for Gouraud interpolated shading
- But rather than interpolate to get RGB values, we get (*u*,*v*) values which point to elements of texture map.
- Thus, texture mapping is done in canonical screen space as the polygon is rasterized
- When describing a scene, you assume that texture interpolation will be done in world space

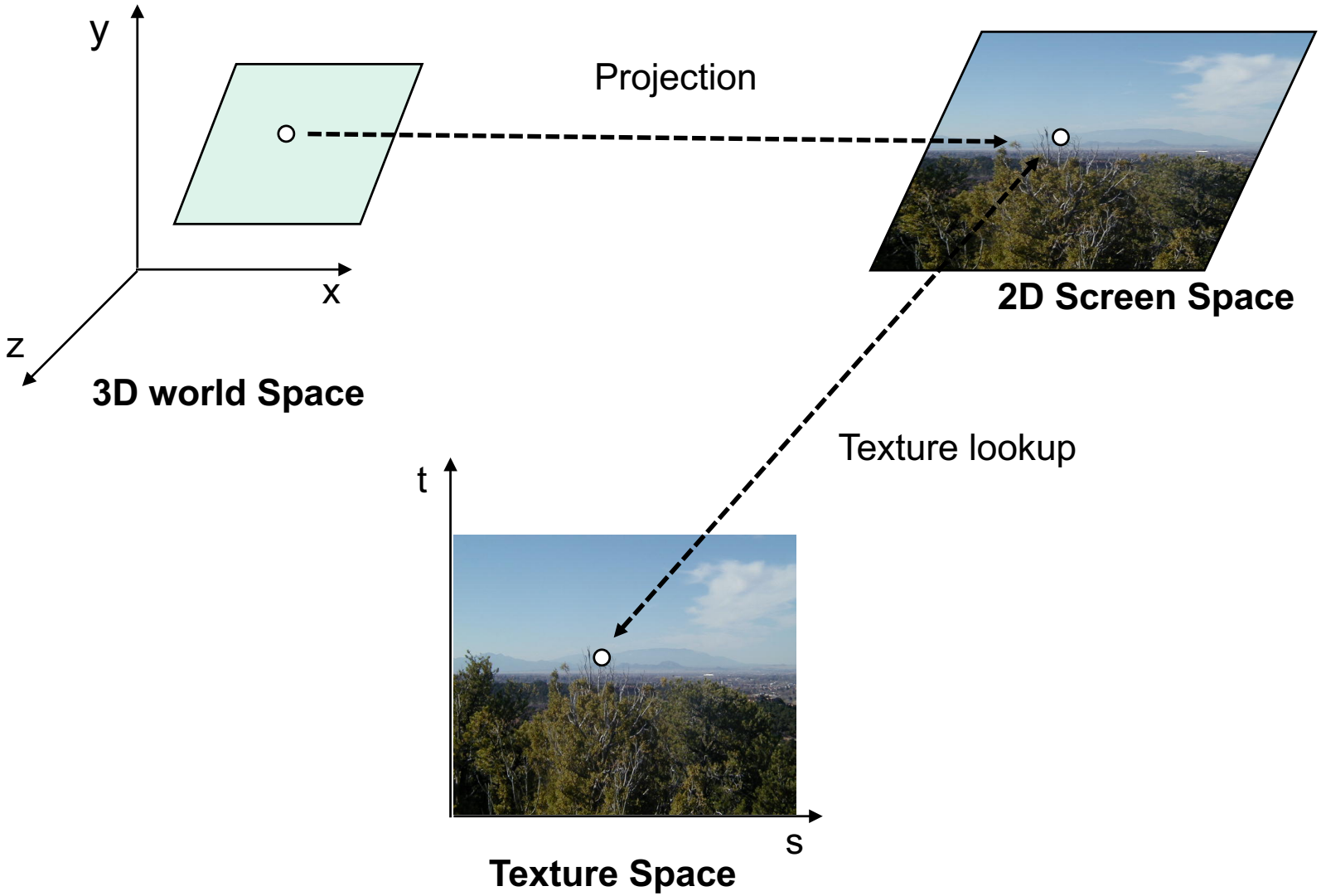


Texture mapping and the OpenGL pipeline

- Images and geometry flow through separate pipelines that join during fragment processing
- "Complex" textures do not affect geometric complexity



Texture mapping



Texturing in OpenGL

Texture Mapping

- uploading of the texture to the video memory
- set up texture parameters
- enable texturing
- the application of the texture onto geometry

Texturing in OpenGL: *4 main steps*

1. **Create/load texture with**
   - `glTexImage*D()`
   - Three methods:
     - read in an image in a jpg, bmp, ...
     - generate the texture yourself within application
     - copy image from color buffer
2. **Define texture parameters as to how texture is applied**
   - `glTexParameter*()`
   - wrapping, filtering, etc
3. **Enable texturing**
   - `glActiveTexture(GL_TEXTURE_*D)`
4. **Assign texture coordinates to vertices**
   - The mapping function is left up to you

Step 1:
Generation / transfer of the texture into graphics memory

Generation:

- Define a texture image as an array of *texels* (texture elements) in CPU memory :

  Glubyte myTexture[width][height][3];

- Each RGB value is specified to be an unsigned byte, between 0 and 255. For example, a blue color would be (0, 0, 255).

Transfer / uppoad:

- We use:

  void **glTexImage2D** (target,level,components,w,h,border,format,type,texture);

Parameters:

- `target` : type of texture, e.g. `GL_TEXTURE_2D`
- `level` : used for mipmapping = 0 (discussed later)
- `components` : elements per texel (for RGB)
- `w, h` : width and height of texture in pixels
- `border` : used for smoothing = 0 (don't worry about this)
- `format` : `texel` format e.g. `GL_RGB`
- `type` : rgb component format e.g. `GL_UNSIGNED_BYTE`
- `texture` : pointer to the texture array

Example:

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, myTexture);
```

The texture resolution must be power of 2.

Step2:
Specifying texture parameters

- OpenGL has a variety of parameters that determine how textures are applied:
  - Wrapping parameters determine what happens if *s* and *t* are outside the (0,1) range
  - Filter modes allow us to use area averaging instead of point samples
  - Mipmapping allows us to use textures at multiple resolutions
- The `glTexParameter()` function is a crucial part of OpenGL texture mapping, this function determines the behavior and appearance of textures when they are rendered.
- Take note that each texture uploaded can have its own separate properties, texture properties are not global.

Step2:
Specifying texture parameters

`glTexParameter()`

Target	Specifies the target texture
GL_TEXTURE_1D	One dimensional texturing.
GL_TEXTURE_2D	Two dimensional texturing.

Texture Parameter	Accepted values	Description
GL_TEXTURE_MIN_FILTER	GL_NEAREST , GL_LINEAR, GL_NEAREST _MIPMAP_NEAREST, GL_LINEAR_MIPMAP_NEAREST, GL_NEAREST _MIPMAP_LINEAR and GL_LINEAR_MIPMAP_LINEAR	The texture minification function is used when a single screen pixel maps to more than one texel, this means the texture must be shrunk in size. Default setting is GL_NEAREST _MIPMAP_LINEAR.
GL_TEXTURE_MAG_FILTER	GL_NEAREST or GL_LINEAR	The texture magnification function is used when the pixel being textured maps to an area less than or equal to one texel, this means the texture must be magnified. Default setting is GL_LINEAR.
GL_TEXTURE_WRAP_S	GL_CLAMP or GL_REPEAT	Sets the wrap parameter for the s texture coordinate Can be set to either GL_CLAMP or GL_REPEAT. Default setting is GL_REPEAT.
GL_TEXTURE_WRAP_T	GL_CLAMP or GL_REPEAT	Sets the wrap parameter for the t texture coordinate Can be set to either GL_CLAMP or GL_REPEAT. Default setting is GL_REPEAT.
GL_TEXTURE_BORDER_COLOR	Any four values in the [0, 1] range	Sets the border color for the texture, if border is present. Default setting is (0, 0, 0, 0).
GL_TEXTURE_PRIORITY	[0,1]	Specifies the residence priority of the texture, use to prevent OpenGL from swapping textures out of video memory. Can be set to values in the [0, 1] range. See *glPrioritizeTextures()* for more information or this article on Gamasutra.

Step2:
Specifying texture parameters

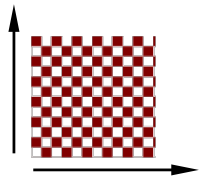
`glTexParameter()`

Target	Specifies the target texture
GL_TEXTURE_1D	One dimensional texturing.
GL_TEXTURE_2D	Two dimensional texturing.

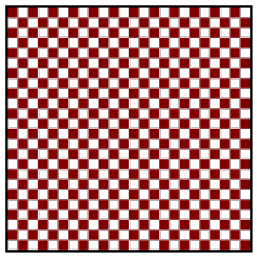
Parameter Value	Description
GL_CLAMP	Clamps the texture coordinate in the [0,1] range.
GL_REPEAT	Ignores the integer portion of the texture coordinate, only the fractional part is used, which creates a repeating pattern. A texture coordinate of 3.0 would cause the texture to tile 3 times when rendered.
GL_NEAREST	Returns the value of the texture element that is nearest (in Manhattan distance) to the center of the pixel being textured. Use this parameter if you would like your texture to appear sharp when rendered.
GL_LINEAR	Returns the weighted average of the four texture elements that are closest to the center of the pixel being textured. These can include border texture elements, depending on the values of GL_TEXTURE_ WRAP_S and GL_TEXTURE_ WRAP_T, and on the exact mapping. Use this parameter if you would like your texture to appear blurred when rendered.
GL_NEAREST_M IPMAP_NEAREST	Chooses the mipmap that most closely matches the size of the pixel being textured and uses the GL_NEAREST criterion (the texture element nearest to the center of the pixel) to produce a texture value.
GL_LINEAR_M IPMAP_NEAREST	Chooses the mipmap that most closely matches the size of the pixel being textured and uses the GL_LINEAR criterion (a weighted average of the four texture elements that are closest to the center of the pixel) to produce a texture value.
GL_NEAREST_M IPMAP_LINEAR	Chooses the two mipmaps that most closely match the size of the pixel being textured and uses the GL_NEARES T criterion (the texture element nearest to the center of the pixel) to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.
GL_LINEAR_M IPMAP_LINEAR	Chooses the two mipmaps that most closely match the size of the pixel being textured and uses the GL_LINEAR criterion (a weighted average of the four texture elements that are closest to the center of the pixel) to produce a texture value from each mipmap. The final texture value is a weighted average of those two values.

Step2 (contd.): *wrapping modes*

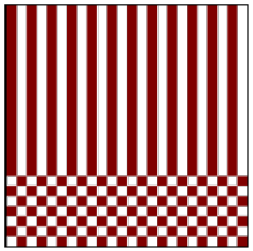
- Clamping : if $s,t > 1$ use color at 1, if $s,t < 0$ use color at 0
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);`
- Repeating : use s,t modulo 1
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);`



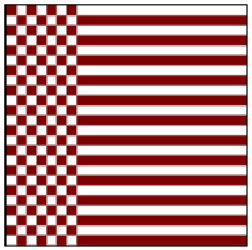
texture



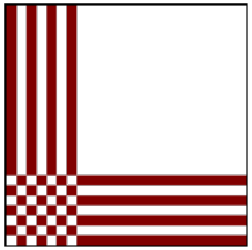
Wrap S : GL_REPEAT
Wrap T : GL_REPEAT



Wrap S : GL_REPEAT
Wrap T : GL_CLAMP



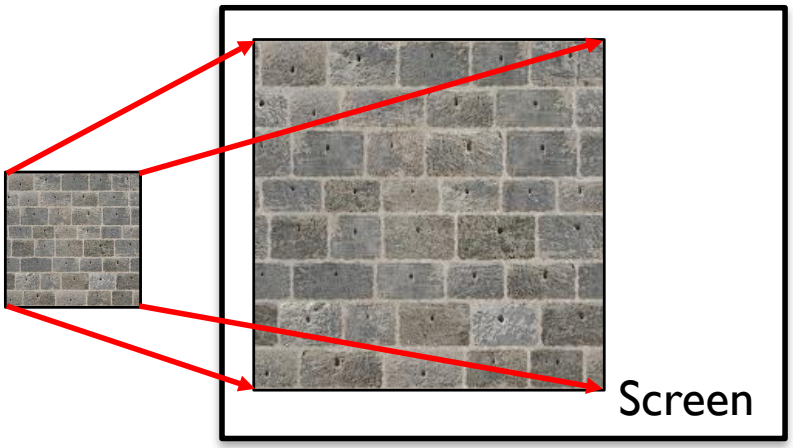
Wrap S : GL_CLAMP
Wrap T : GL_REPEAT



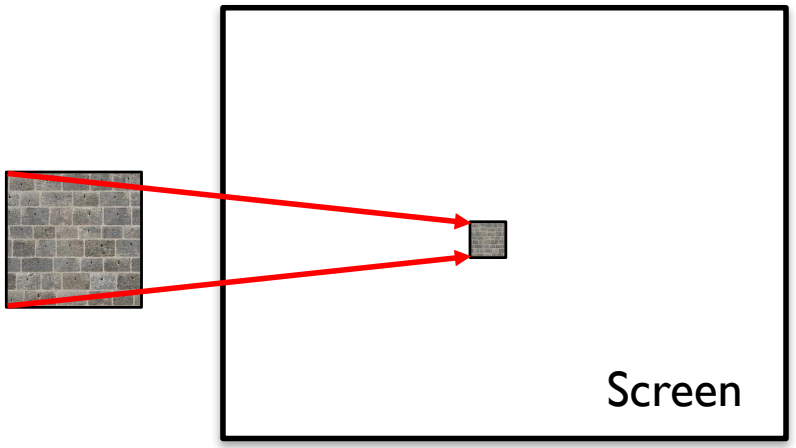
Wrap S : GL_CLAMP
Wrap T : GL_CLAMP

Step2: *filtering modes*

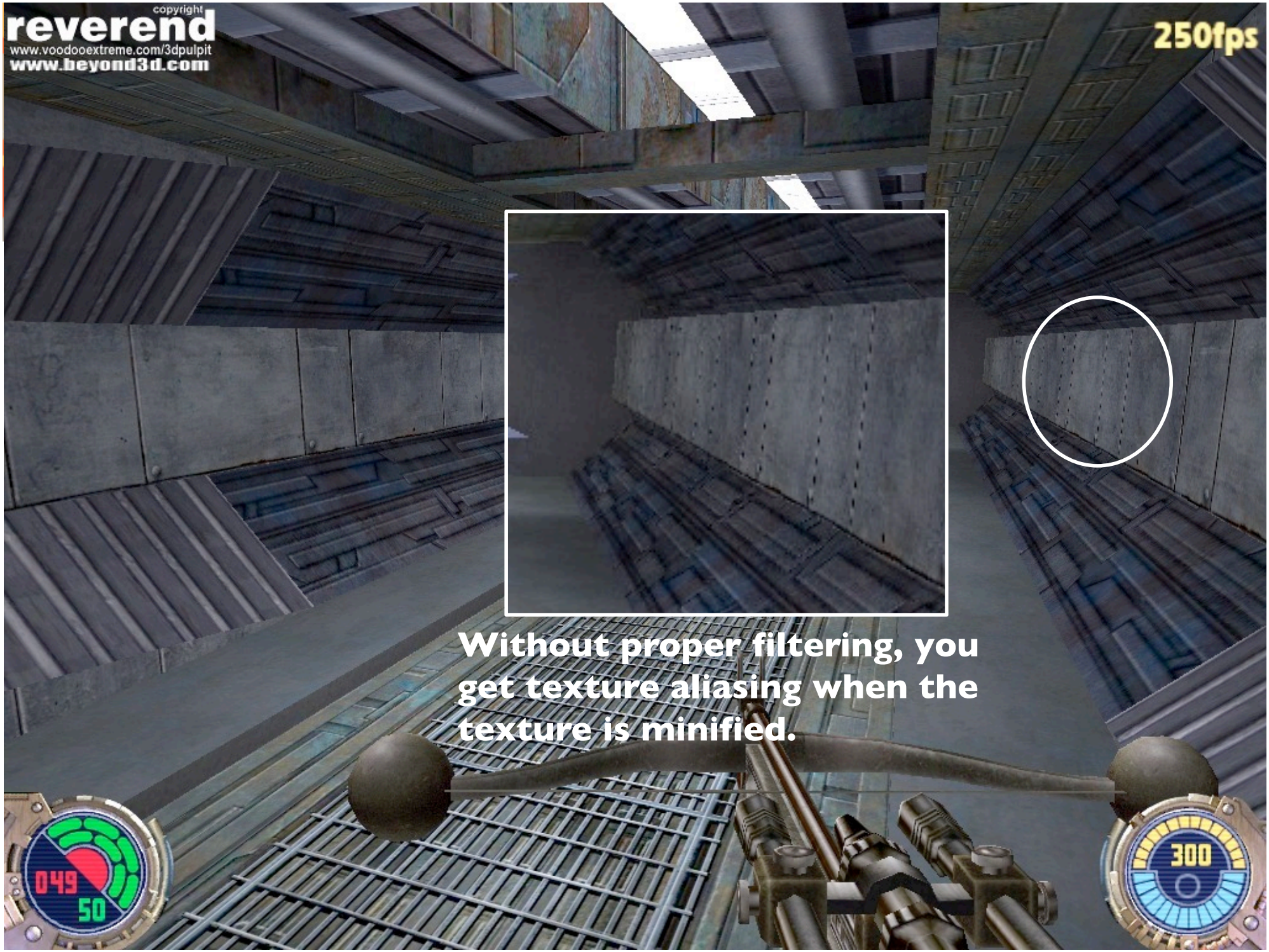
- Since a texture can be mapped arbitrarily to an image region, it can either be magnified or minified.
- Mag filter: To interpolate a value from neighboring texels
- Min filter: Combine multiple texels into a single value



Magnification



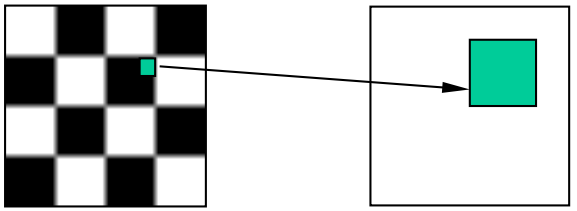
Minification


copyright
reverend
www.voodooextreme.com/3dpulpit
www.beyond3d.com
250fps
Without proper filtering, you get texture aliasing when the texture is minified.
049
50
300

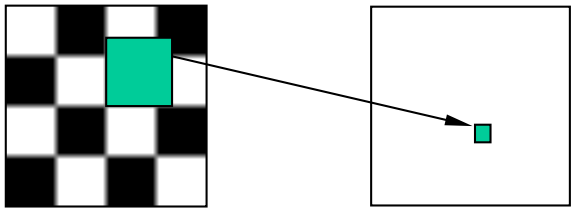
Step2 (contd.): *filtering modes*

Recall that:

- Texture map to surface takes place during rendering, much like in Gouraud shading:
  - Triangle rasterized
  - Each pixel mapped back to the texture
  - Use known values at vertices to interpolation over the texture
- Each pixel is associated with small region of surface *and* to a small area of texture.
- There are 3 possibilities for association :
  1. one texel to one pixel (rare)
  2. Magnification: one texel to many pixels
  3. Minification: many texels to one pixel



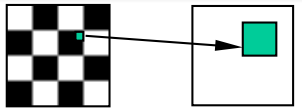
Magnification



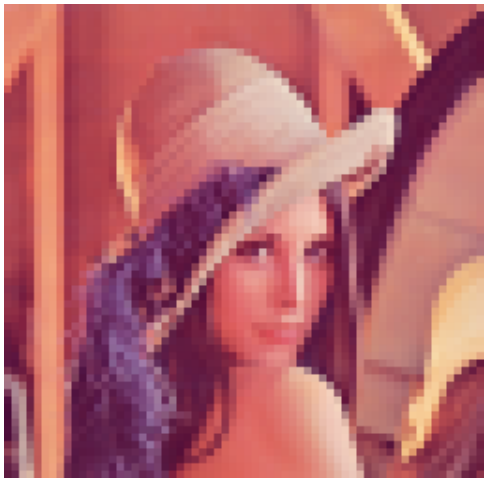
Minification

Step 2 (contd.): *Zoom in with magnification filter*

- *Pixel maps to a small portion of one texel*
- *Results in many pixels mapping to same texel*
- *Without a filtering method, aliasing is common*
- *Magnification filter: smooths transition between pixels*



Magnificação

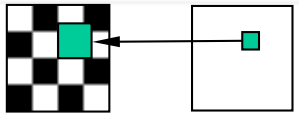


many pixels correspond to one texel
→ "blockiness" / jaggies / aliasing

solution: apply averaging
(magnification filter)

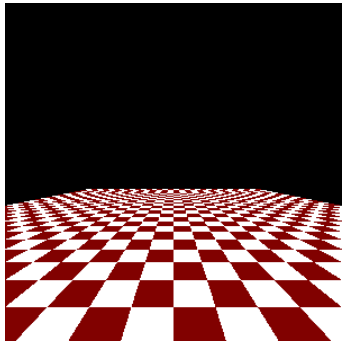
Step 2 (contd.):
Zoom out with minification filter

- *One pixel is mapped to many texels*
- *It is commonly found in perspective projection (foreshortening)*

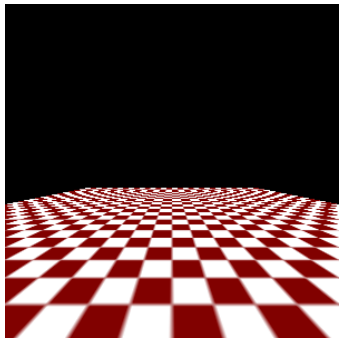


Minification

Foreshortening is a technique used in perspective to create the illusion of an object receding strongly into the distance or background. The illusion is created by the object appearing shorter than it is in reality, making it seem compressed. It is an excellent way to maximize the depth and dimension of paintings and drawings.
(taken from https://www.liveabout.com/definition-of-foreshortening-2577559)



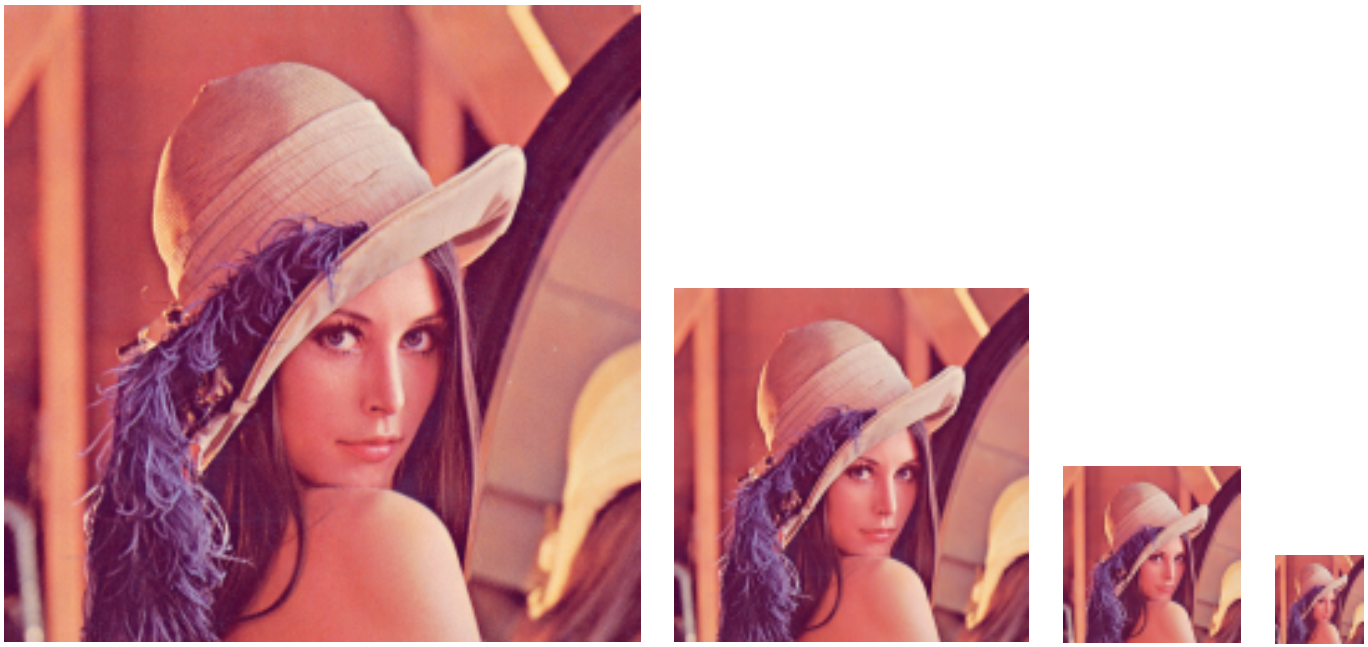
Perspective (foreshortening) and poor texture mapping causes visual deformation of the floor



Mipmaps improve the texture mapping

Step 2 (contd.):
Mipmaps are the best minification filter

- *"mip" means multum in parvo, ou "many things in a small place "*
- *Leading idea: create many textures of decreasing size, using one of them when adequate*
- *Pre-filtered textures = mipmaps*

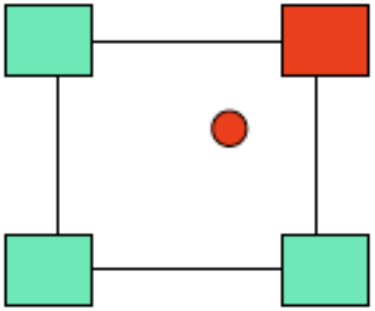


Step 2 (contd.):
Mipmaps to optimize storage

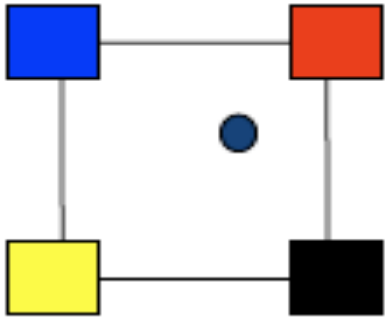
– *It is mandatory to provide texture sizes in power of 2 in relation to the original in 1x1*

Step 2 (contd.): *filtering modes*

OpenGL texture filtering



Nearest Neighbor
(fast, but with aliasing)



Bi-linear Interpolation
(slow, but less aliasing)

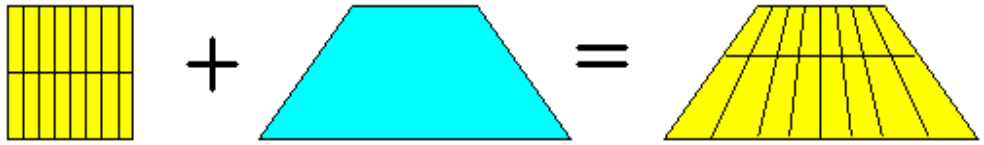
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST)

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
```

Step 2 (contd.): *texture color blending modes*

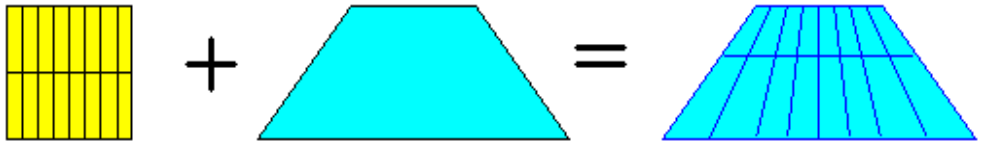
After a texture value is retrieved (may be further transformed), the resulting values are used to modify one or more polygon/surface attributes by means of the ***blending functions****:*

Blending functions:

- ***replace*:** replace surface color with texture color
- ***decal***: replace surface color with texture color, blend the color with underlying color with an alpha texture value, but the alpha component in the framebuffer is not modified
- ***modulate***: multiply the surface color by the texture color (shaded + textured surface). Need this for multitexturing (i.e., lightmaps).
- ***blend***: similar to modulation but add alpha-blending



REPLACE operation



MODULATE operation

Step 2 (contd.): *texture color blending modes*

- **Determine how to combine the texture color with the object color**
  - For example, GL_MODULATE: multiply texture with object color
  - GL_BLEND: linear combination of texture and object color
  - GL_REPLACE: use texture color to replace object color

- **For example:**

```
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE)
```

- **Remember to use GL_MODULATE (default) if you want to have the light effect.**

Step 3: Enable Texturing

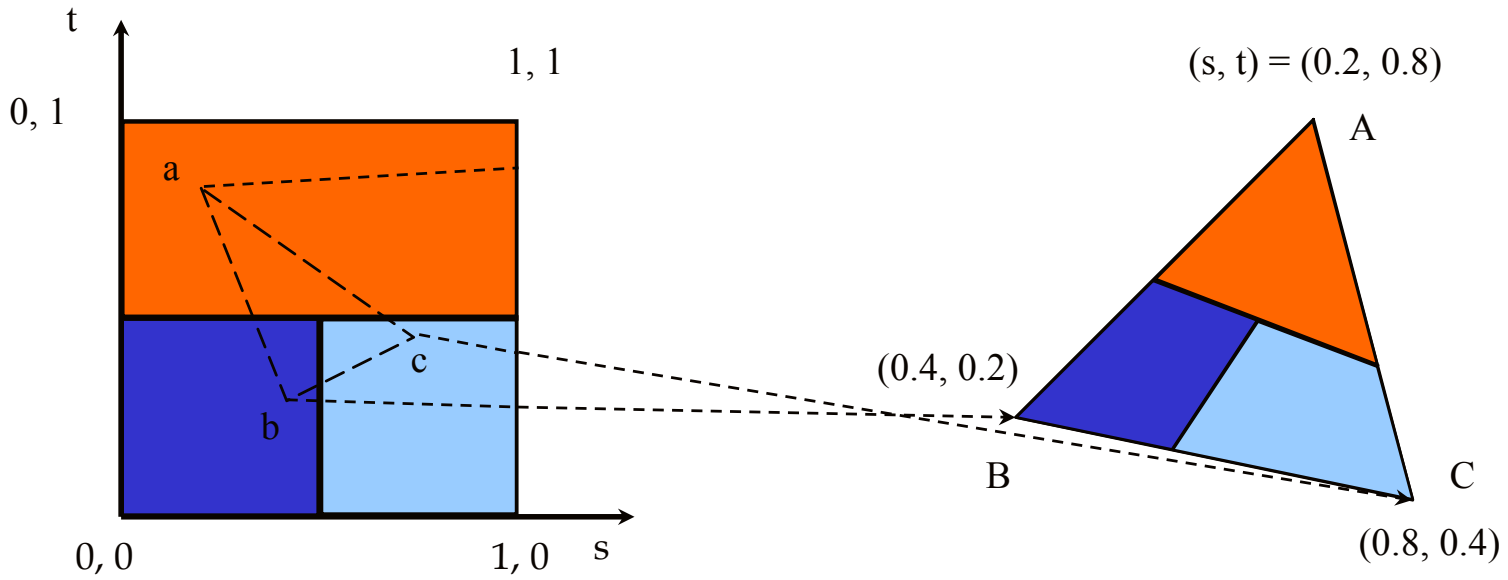
- To enable just call:
  - `glActiveTexture(GL_TEXTURE_2D)`

- What does texture mapping affect?
  - the current shading color of a pixel (after lighting) is multiplied by the corresponding texture color

- So, if the object is a near white color (0.8, 0.8, 0.8) at some point and the current texture color at that point is red (1, 0, 0), then when multiplied, it produces (0.8,0.0,0.0)

Step 4:
Mapping a texture to a triangle

- Assign the texture coordinates to triangle vertices
- The color of each triangle pixel is obtained by interpolation

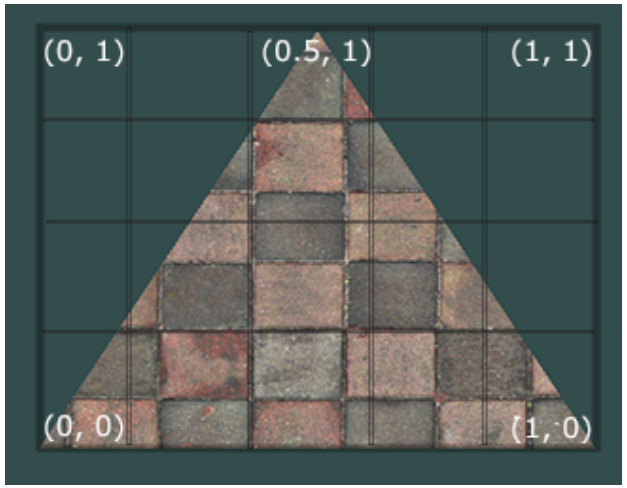


Texture Space **Object Space**

Step 4 (contd.): Mapping a texture: *example*

– How to texture a triangle? The following code assumes that texturing has been enabled and that there has been a texture uploaded with the id of 13.

```
float texCoords[] = {
   0.0f, 0.0f, // lower-left corner
   1.0f, 0.0f, // lower-right corner
   0.5f, 1.0f  // top-center corner
};
(…)
glBindTexture (GL_TEXTURE_2D, 13);
```



Step 4 (contd.): the whole process

"stb_image.h is a very popular single header image loading library by Sean Barrett that is able to load most popular file formats and is easy to integrate in your project(s). stb_image.h can be downloaded from here. Simply download the single header file, add it to your project as stb_image.h"
https://learnopengl.com/Getting-started/Textures

container.jpg


```
unsigned int texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);

// set the texture wrapping/filtering options (on the currently bound texture object)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

// load and generate the texture
int width, height, nrChannels;
unsigned char *data = stbi_load("container.jpg", &width, &height, &nrChannels, 0);
if (data) {
     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
     glGenerateMipmap(GL_TEXTURE_2D);
}
else
     std::cout << "Failed to load texture" << std::endl;

stbi_image_free(data);
```

Step 4 (contd.): the whole process including shaders

https://learnopengl.com/Getting-started/Textures

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;
layout (location = 2) in vec2 aTexCoord;

out vec3 ourColor;
out vec2 TexCoord;

void main() {
     gl_Position = vec4(aPos, 1.0);
     ourColor = aColor;
     TexCoord = vec2(aTexCoord.x, aTexCoord.y);
}
```

```
#version 330 core
out vec4 FragColor;
in vec3 ourColor;
in vec2 TexCoord;
// texture samplers
uniform sampler2D texture1;
uniform sampler2D texture2;
void main() {
     // linearly interpolate between both textures (80% container, 20% awesomeface)
     FragColor = mix(texture(texture1, TexCoord), texture(texture2, TexCoord), 0.2);
}
```

Further reading

https://learnopengl.com/Getting-started/Textures

https://open.gl/textures

http://www.opengl-tutorial.org/beginners-tutorials/tutorial-5-a-textured-cube/

Summary:

...:

- Objectives and motivation
- Notion of texture, texture mapping, texture patterns and texels
- Texture mapping on polygons, texture interpolation
- Wrapping modes
- Filtering modes
- Blending modes
- Mapping textures on geometric objects
  - planar mapping
  - cylindrical mapping
  - spherical mapping
  - box mapping