			Readings	Homework
Tue	3	Materials	Chap 14	
Wed	4	Lighting setup exercise		HW #3
Thur	5	<Children's Day>		Due (5/6)
	10	Shaders (Review+)	Chap 1~14	HW #4
	11	Open Lab		
	12	Color / Shading	Chap 19/Ext	
	17	Raytracing	Chap 20	
	18	Open Lab		
	19	Light	Chap 21	Due: May 24 11:59PM
	24	Texture Mapping 1	Chap 15	
< E11: #307 >	25	Texture mapping exercise		HW #5
	26	**Quiz / HW#5 / Q&A** < N1: #112 >		
	31	Texture Mapping 2	Chap 15	
7-10PM	1	CUDA Special Lab (by NVIDIA)		
< N1: #102 >	2	Sampling	Chap 16	
	7	Samplling/Reconstruction	Chap 16/17	
	8	Open Lab		
	9	Geometirc modeling	Chap 22	
	14	Animation	Chap 23	
	21	Final Exam		

Announcement: GPU-based Accelerated Computing and Deep Learning

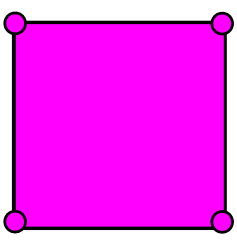
- **NVIDIA Special Lab Tomorrow (6/1) 7~10 PM**
  - Not mandatory, but attendance will be checked!
  - (Bonus attendance point will be granted)
  - **N1 : Room 102**
  - Instructor: **Hyun-Gon Yu (Solution Architect) NVIDIA Korea**
  - Contents
    - CUDA and OpenACC programming
    - Deep learning modeling using DIGITS (GPU-based Deep Learning tool)
  - Must bring your own notebook computer (with its power cord!)
- Please follow the instruction on preparation file!

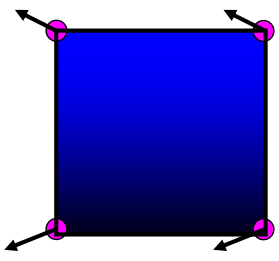
Texture Mapping

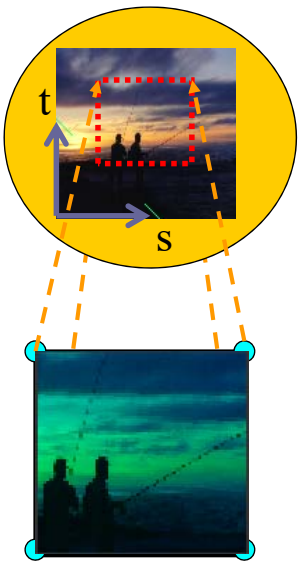
Chapter 15

‘coloring’

- ‘shading’
- ‘texture mapping’
-







(LAB)
Process of Texture Mapping in OpenGL

1] Specify textures in texture objects

2] Set texture filter

3] Set texture function

4] Set texture wrap mode

5] Bind texture object

6] Enable texturing

7] Supply texture coordinates for vertex

Appendix A.4 Adding a Texture

- To set up the texture

Set active hardware texture unit

Generate a name texture and bind it as the “current 2D texture”

We’ll look at these in the next chapter

Read image

Load that data into the texture

```
...
glActiveTexture(GL_TEXTURE0);
glGenTextures(1, &h_texture);
glBindTexture(GL_TEXTURE_2D, h_texture);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINE
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINE
int twidth, theight;
packed_pixel_t *pixdata = ppmread("reachup.ppm",
    &twidth, &theight);
assert(pixdata);
glTexImage2D(GL_TEXTURE_2D, 0, GL_SRGB,
    twidth, theight, 0,
    GL_RGB, GL_UNSIGNED_BYTE, pixdata);
free(pixdata);
...
```

In the Lab

```
//TODO: Initialize first texture
texture[0] = loadBMP_custom("Judy.bmp");
for (int i = 0; i < 3; i++) textureID[i][0] =
glGetUniformLocation(addPrograms[i], "myTextureSampler");
…
//TODO: pass the first texture value to shader
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture[0]);
glUniform1i(textureID[program_cnt][0], 0);


….
```

Texture mapping

Vertex Shader

```
#version 330

uniform  float uVertexScale;

in vec2 aVertex;
in vec2 aTexCoord;        ← Vertex attribute
in vec3 aColor;

out vec3 vColor;
out vec2 vTexCoord;       ← Varying variable

void main()
{
  gl_Position = vec4(aVertex.x*uVertexScale,aVertex.y, 0,1);
  vColor=aColor;
  vTexCoord=aTexCoord;
}
```

Texture mapping

- Fragment Shader

```
uniform float uVertexScale;
uniform sampler2D texUnit0;
```

Special data type refers to an OpenGL texture unit

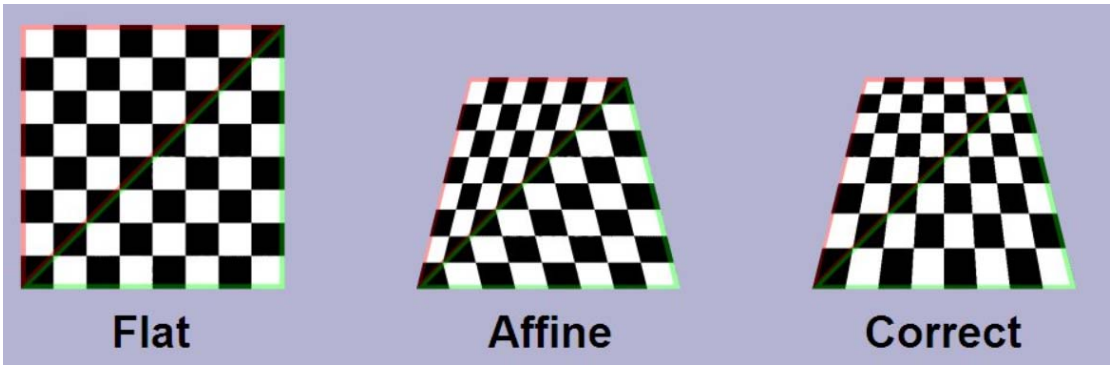
```
in vec2 vTexCoord;
in vec3 vColor;

out vec4 fragColor;

void main(void)
{
  vec4 color0 = vec4(vColor.x, vColor.y, vColor.z, 1);
  vec4 color1 = texture2D(texUnit0, vTexCoord);

  float lerper = clamp(.3 *uVertexScale,0.,1.);
  fragColor = (lerper)*color1 + (1.-lerper)*color0;
}
```

color1 is fetched using the special GLSL function texture2D, from the texture unit pointed to by texUnit0, using the texture coord paased in as vTexCoord

Mixing 2 colors

<Lecture 13> Varying Variable

- Perspective correctness



https://en.wikipedia.org/wiki/Texture_mapping

<Lecture 13> Varying Variable

- **Perspective correct** texturing accounts for the vertices' positions in 3D space, rather than simply interpolating a 2D triangle.
- This achieves the correct visual effect, but it is slower to calculate.
- Instead of interpolating the texture coordinates directly, the coordinates are divided by their depth (relative to the viewer), and the reciprocal of the depth value is also interpolated and used to recover the perspective-correct coordinate.
- This correction makes it so that in parts of the polygon that are closer to the viewer the difference from pixel to pixel between texture coordinates is smaller (stretching the texture wider), and in parts that are farther away this difference is larger (compressing the texture).

<Lecture 13> Varying Variable

- Affine texture mapping directly interpolates a texture coordinate u_a between two endpoints u_0 and u_1

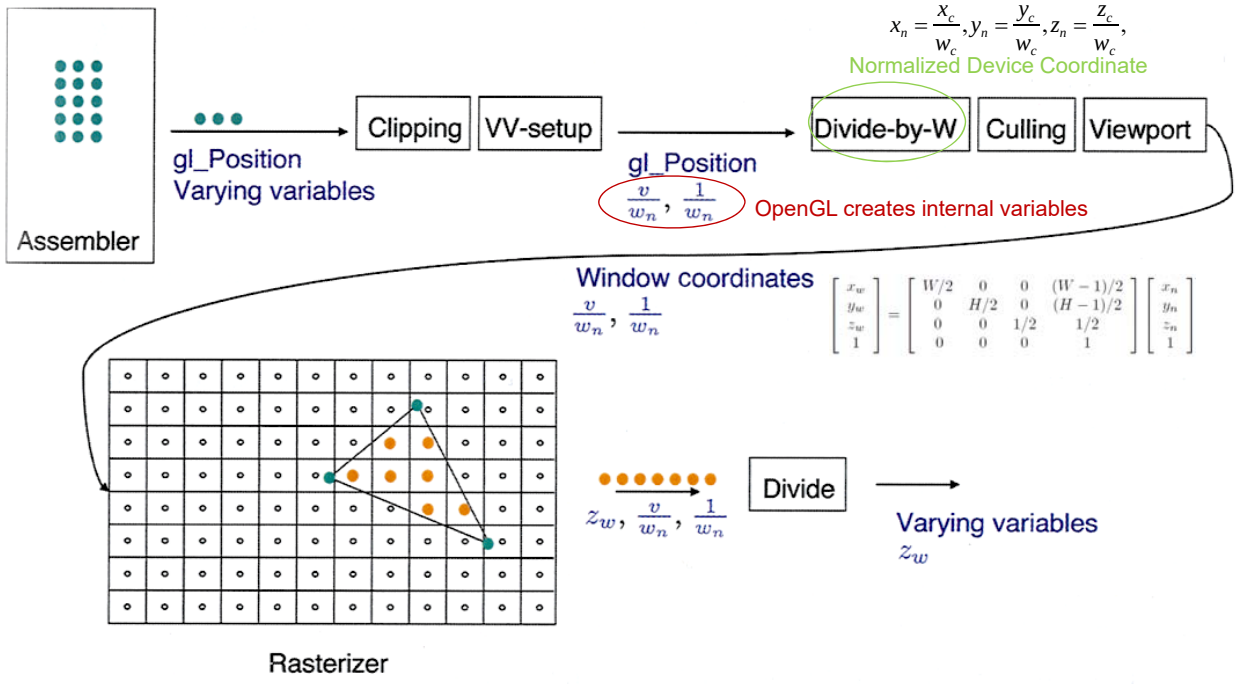
$$u_\alpha = (1-\alpha)u_0 + \alpha u_1 \text{ where } 0 \le \alpha \le 1$$

- Perspective correct mapping interpolates after dividing by depth z, then uses its interpolated reciprocal to recover the correct coordinate

$$u_\alpha = \frac{(1-\alpha)\frac{u_0}{z_0} + \alpha\frac{u_1}{z_1}}{(1-\alpha)\frac{1}{z_0} + \alpha\frac{1}{z_1}}$$

- All modern 3D graphics hardware implements perspective correct texturing

Interpolating varying variables



https://www.opengl.org/wiki/Type_Qualifier_(GLSL

Type Qualifier

- **A type qualifier** is used in the OpenGL Shading Language (GLSL) to modify the storage or behavior of global and locally defined variables.
- These qualifiers change particular aspects of the variable, such as where they get their data from and so forth.
  - Storage qualifiers
    - Constant
    - Shader stage inputs and outputs
    - Uniforms
    - Buffer
    - …
  - Layout
  - …

- Vertex shader
- Tessellation shader
- Geometry shader
- Fragment shader
- Interpolation qualifiers

https://www.opengl.org/wiki/Type_Qualifier_(GLSL

Shader stage inputs and outputs

- Vertex shader inputs
  - User-defined inputs for vertex shaders are called vertex attributes. They are passed via vertex arrays to the vertex shader (usally from data stored in Buffered Objects)
- Interpolation qualifiers
  - Certain inputs and outputs can use interpolation qualifiers. These are for any values which could be interpolated as a result of rasterization.
    - Vertex shader outputs
    - Fragment shader inputs
  - Interpolation qualifiers control how interpolation of values happens across a triangle or other primitive.

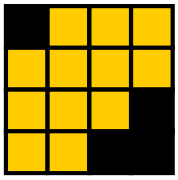
Interpolation qualifiers

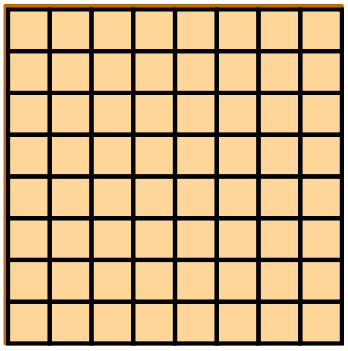
- flat
  - The value will not be interpolated.
  - The value given to the fragment shader is the value from the Proviking Vertex for that primitive
- nopersective
  - The value will be linearly interpolated in window-space.
  - This is usually not what you want, but it can have its uses.
- smooth
  - The value will be interpolated in a perspective-correct fashion. This is the default if no qualifier is present.

Exercise

My polygon 1 (2x2)

My texture image (4x4)



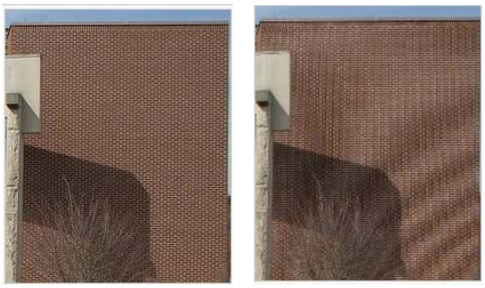


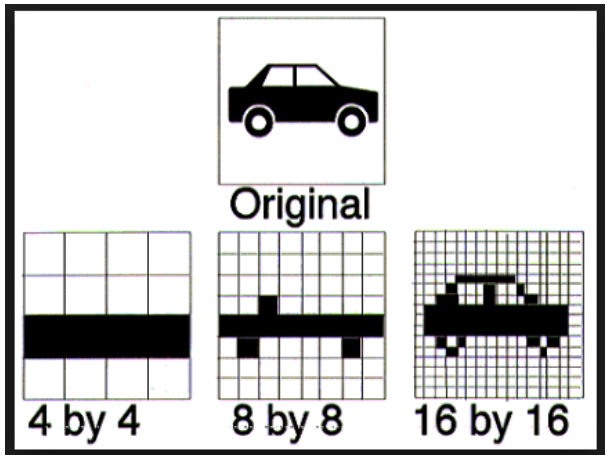
My polygon 2 (8x8)

Texture Sampling

- Aliasing of textures is a major problem.

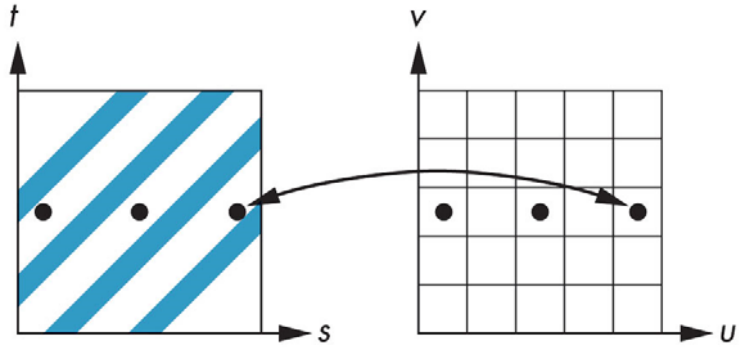
  <Wikipedia> In signal processing and related disciplines, **aliasing** refers to an effect that causes different signals to become indistinguishable when sampled. It also refers to the distortion or artifact that results when the signal reconstructed from samples is different from the original continuous signal.





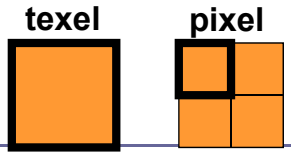
Texture Sampling

- Aliasing of textures is a major problem.



- Magnification
  - Texel is larger than one pixel
- Minification
  - Texel is smaller than one pixel

magification



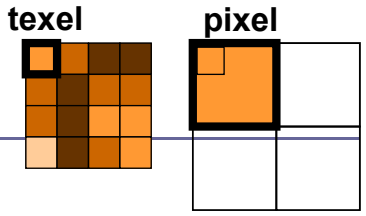
- Nearest neighbor
- Bilinear interpolation

A texel covers several pixels

Pixelation – blocky appearance

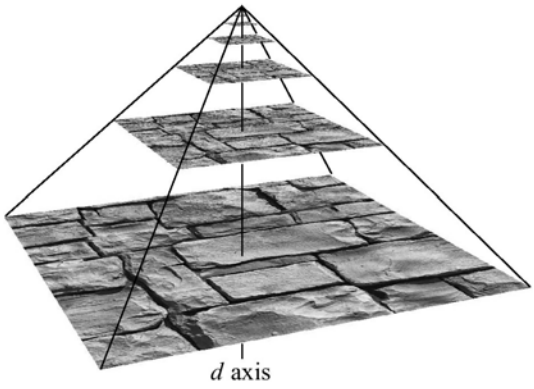
Linear interpolation

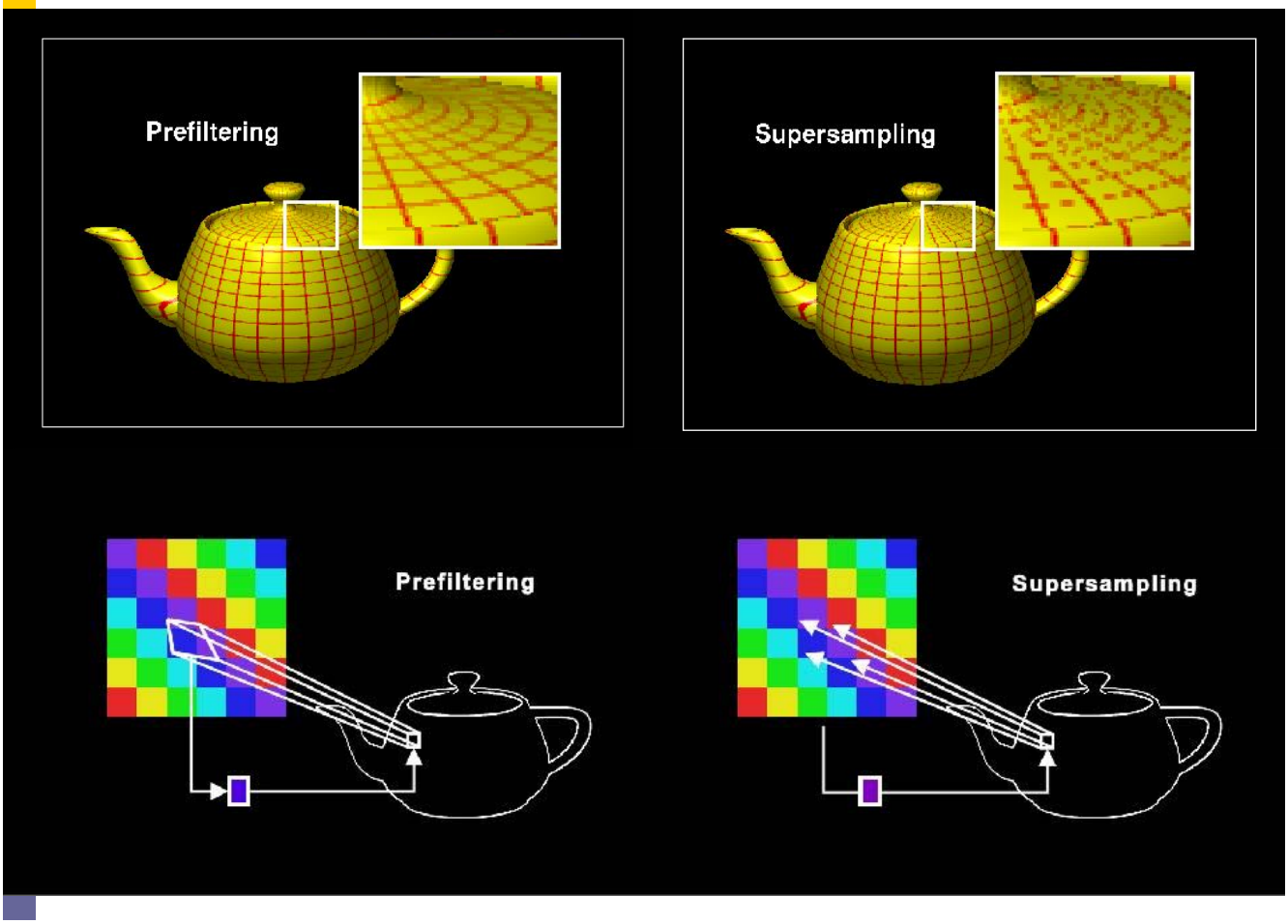
minification

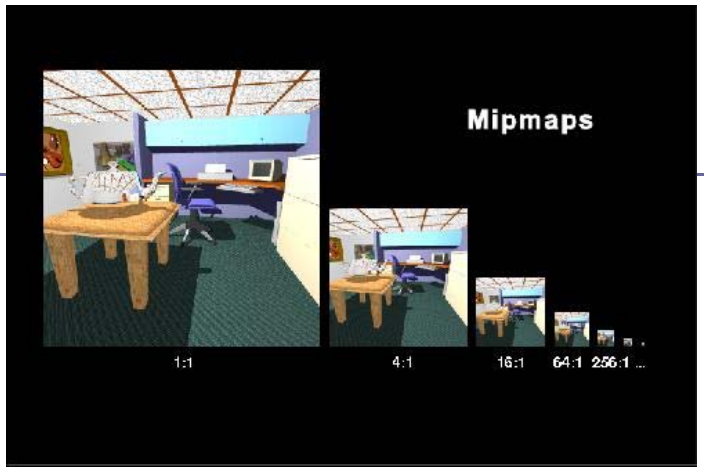


- Point sampling (nearest neighbor)
  - Severe aliasing
- Supersampling
- Mipmapping
  - Mip (multum in pravo, Latin)
    {many things in a small place}

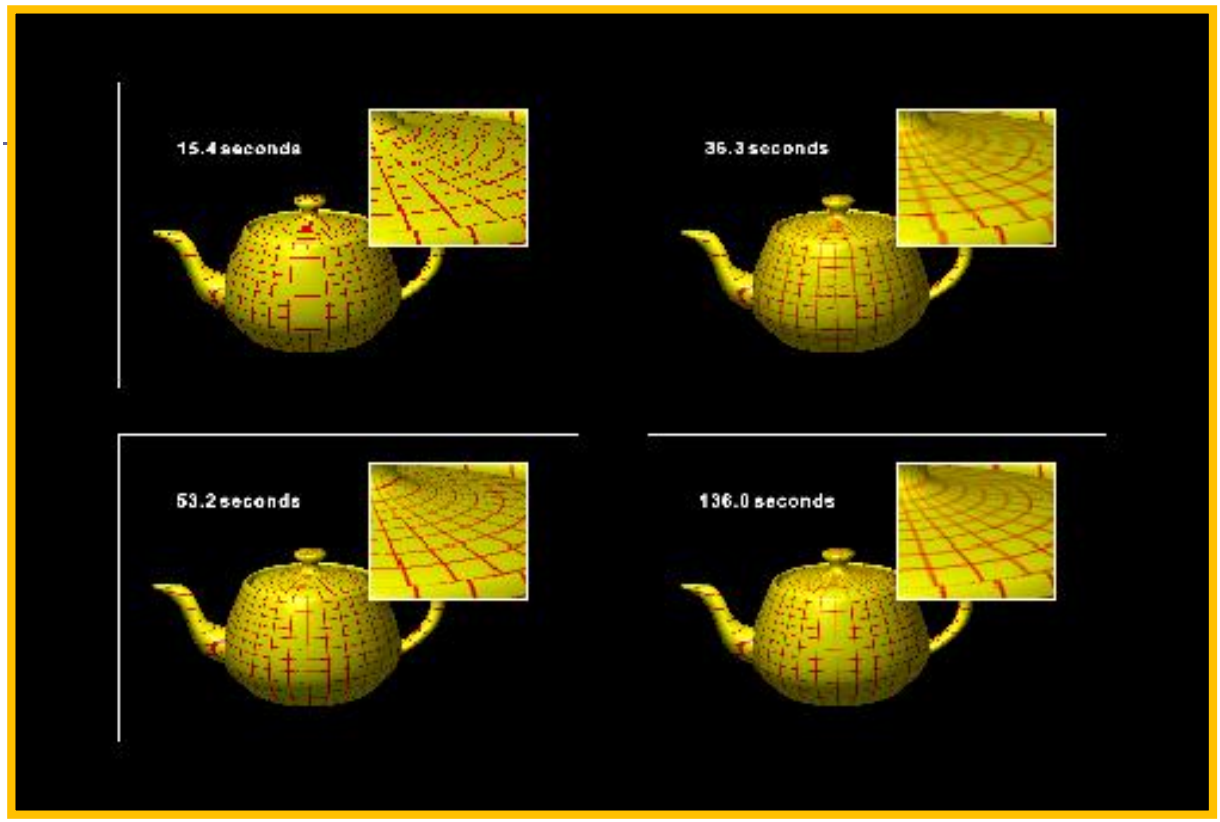
A texel covers only a fraction of the pixels





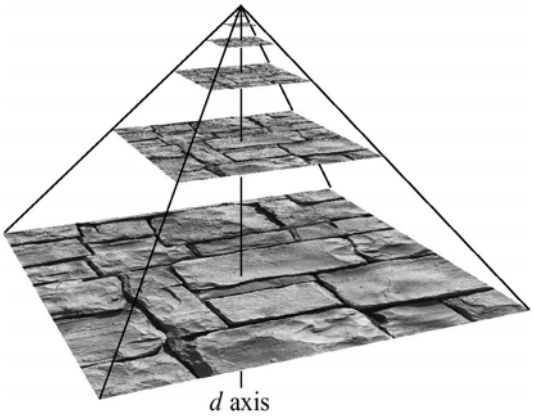

Mipmaps
1:1
4:1
16:1
64:1
256:1 ...


9:1
9:1
1:1
4:1
16:1
64:1
256:1 ...


15.4 seconds
36.3 seconds
53.2 seconds
136.0 seconds

Sampling problems

- Minification
  - Point sampling
    - Severe aliasing
  - Super sampling
  - Mipmapping



- Magnification
  - Nearest neighbor
    - Blocky looking
  - Bilinear interpolation
    - Blurred image

Siggraph 2012 ***Réflexion***. Best in Show Award

https://youtu.be/o5qEV6l1bN4