CS 380 - GPU and GPGPU Programming
Lecture 10: GPU Texturing 1

Markus Hadwiger, KAUST

Reading Assignment #5 (until Mar. 21)

Read (required):

- GLSL book, chapter 6 (*Simple Shading Example*)
- GLSL book, chapter 8.1-8.3 (*Shader Development*)

Read (optional):

- GLSL book, chapter 7 (*OpenGL Shading Language API*)

Programming Assignment #2 (until Apr. 4)

Shading example

- Phong shading, procedural shading, sphere tessellation
- OpenGL/GLSL vertex, geometry, fragment shaders

Download framework:

`http://faculty.kaust.edu.sa/sites/markushadwiger/Documents/...`

`... /CS380_prog2_Windows.zip`	for Windows
`... /CS380_prog2_Linux_MacOS.tgz`	for Linux and MacOS X

Windows: solution/project file for Visual Studio 2008 (easy to do your own)

Linux/MacOS: Makefile. Look at it and edit as needed!

Programming Assignment #2 (until Apr. 4)

```
#version 120

uniform vec3 LightPosition;

const float SpecularContribution = 0.3;
const float DiffuseContribution  = 1.0 - SpecularContribution;

varying float lightIntensity;
varying vec3 texPos;

void main(void)
{
    // TODO 1:
    //
    // move lighting computations from here (i.e., vertex shader)
    // to fragment shader to do Phong shading (interpolation) with
    // the Phong lighting model instead of Gouraud shading with the
    // Phong lighting model. For more information see
    //
    // http://en.wikipedia.org/wiki/Gouraud_shading
    // http://en.wikipedia.org/wiki/Phong_shading

    vec3 ecPosition = vec3(gl_ModelViewMatrix * gl_Vertex);
    vec3 tnorm      = normalize(gl_NormalMatrix * gl_Normal);
    vec3 lightVec   = normalize(LightPosition - ecPosition);
    vec3 reflectVec = reflect(-lightVec, tnorm);
    vec3 viewVec    = normalize(-ecPosition);
    float diffuse   = max(dot(lightVec, tnorm), 0.0);
    float spec      = 0.0;
```

Programming Assignment #2 (until Apr. 4)

```
#version 120

varying vec3 texPosGS;
varying float lightIntensityGS;

void main(void)
{
    // TODO 2:
    //
    // Use texPosGS to implement the examples of chapters
    // 11.1, 11.2, and 11.3 in the "OpenGL Shading Language"
    // book. Provide key mappings in "CS380_prog2.c" to switch
    // between these examples.
    //
    // optional: implement (procedural) bump mapping (normal mapping)
    //           as in chapter 11.4

    float val = lightIntensityGS;
    gl_FragColor = vec4 (val, val, val, 1.0);
}
```

Programming Assignment #2 (until Apr. 4)

```
#version 120
#extension GL_EXT_geometry_shader4 : enable

varying in float lightIntensity[3];
varying in vec3 texPos[3];

varying out float lightIntensityGS;
varying out vec3 texPosGS;

void main(void)
{
   // TODO 3:
   //
   // Move the subdivision of the sphere in drawtri() in drawSphere.c
   // here to produce a finer tessellation (reuse the 'd' key, see
   // "CS380_prog2.c", to switch between the different resolutions).
   // This will only work for the sphere from "drawSphere.c", do not
   // care about the other models.

   for(int i=0; i< gl_VerticesIn; ++i){
        lightIntensityGS = lightIntensity[i];
        texPosGS = texPos[i];
        gl_Position = gl_PositionIn[i];
        EmitVertex();
   }

   EndPrimitive();
}
```

Semester Project (Proposal until Apr. 4!)

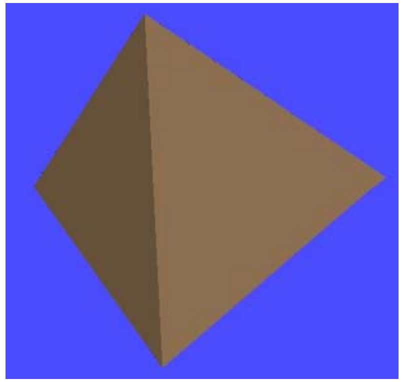
- Try to find your own topic of interest
  - Pick something that you think is really cool
  - Can be completely graphics or completely computation or both combined
  - Browse GPU Gems 2, GPU Gems 3 books (in library!) for ideas, browse SDK examples, look online
  - Can be built on NVIDIA OpenGL SDK or NVIDIA CUDA SDK
  - Amount of work ~all four programming assignments together
- Write project proposal
  - 1-2 pages (pdf), just overview of plan
  - Talk to us before you start writing! (before spring break!) (regarding content and complexity)
  - Hand in proposal after spring break (Apr. 4)
- Project presentations May 22-25, before that write report

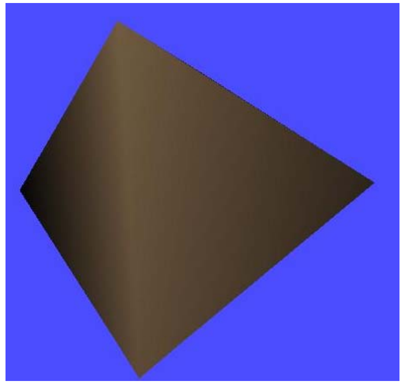
GPU Texturing

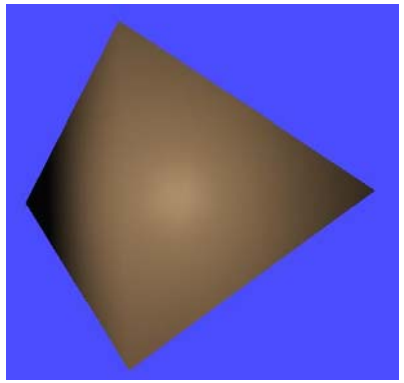
Rage / id Tech 5 (id Software)

Before We Start: Shading

- Flat shading
  - compute light interaction per polygon
  - the whole polygon has the same color
- Gouraud shading
  - compute light interaction per vertex
  - interpolate the colors
- Phong shading
  - interpolate normals per pixel
- Remember: difference between
  - Phong Lighting Model
  - Phong Shading

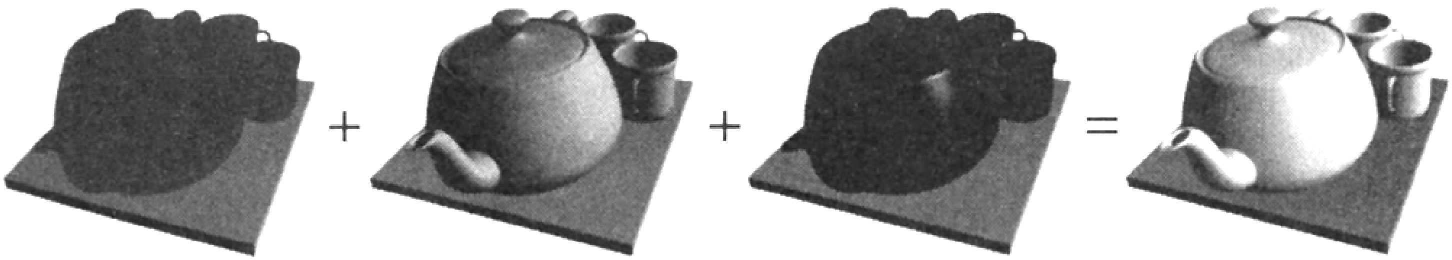






Traditional OpenGL Lighting

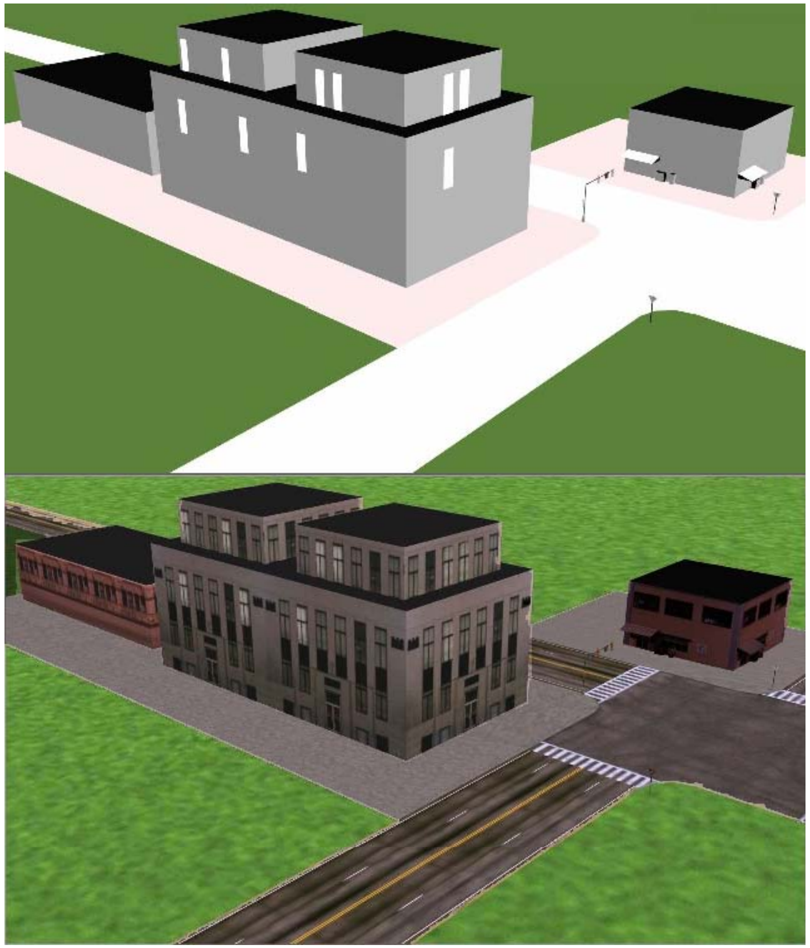
- Phong lighting model at each vertex (glLight, …)
- Local model only (no shadows, radiosity, …)
- ambient + diffuse + specular (glMaterial!)



- Fixed function: Gouraud shading
  - Note: need to interpolate specular separately!
- Phong shading: evaluate Phong lighting model in fragment shader (per-fragment evaluation!)

Why Texturing?

- Idea: enhance visual appearance of surfaces by applying fine / high-resolution details



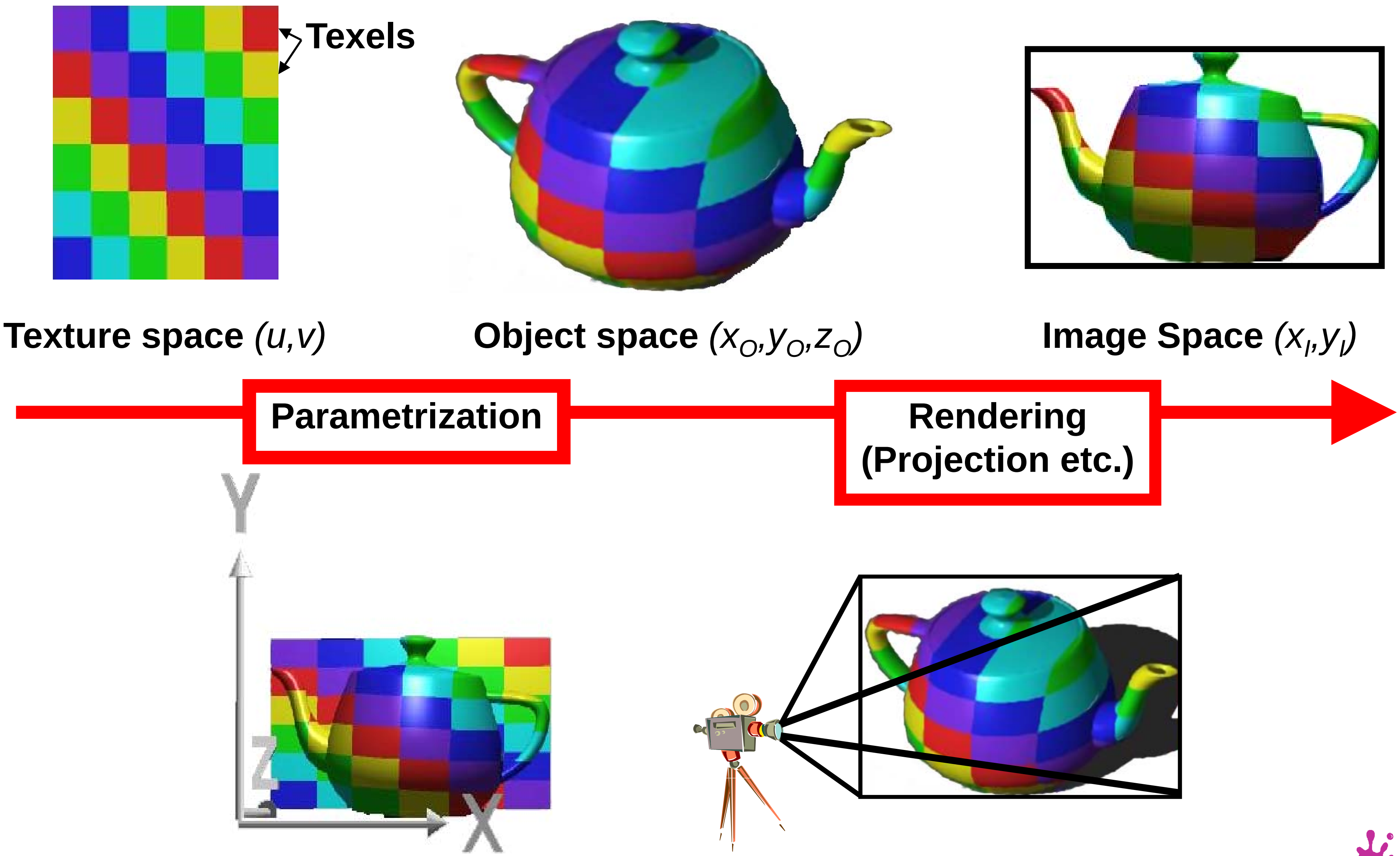
OpenGL Texture Mapping

- Basis for most real-time rendering effects
- Look and feel of a surface
- Definition:
  - A *regularly sampled function* that is mapped onto every *fragment* of a surface
  - Traditionally an image, but…
- Can hold arbitrary information
  - Textures become general data structures
  - Sampled and interpreted by fragment programs
  - Can render into textures → important!

Types of Textures

- Spatial layout
  - Cartesian grids: 1D, 2D, 3D, 2D_ARRAY, …
  - Cube maps, …
- Formats (too many), e.g. OpenGL
  - GL_LUMINANCE16_ALPHA16
  - GL_RGB8, GL_RGBA8, …: integer texture formats
  - GL_RGB16F, GL_RGBA32F, …: float texture formats
  - compressed formats, high dynamic range formats, …
- External format vs. internal (GPU) format
  - OpenGL driver converts from external to internal

Texturing: General Approach

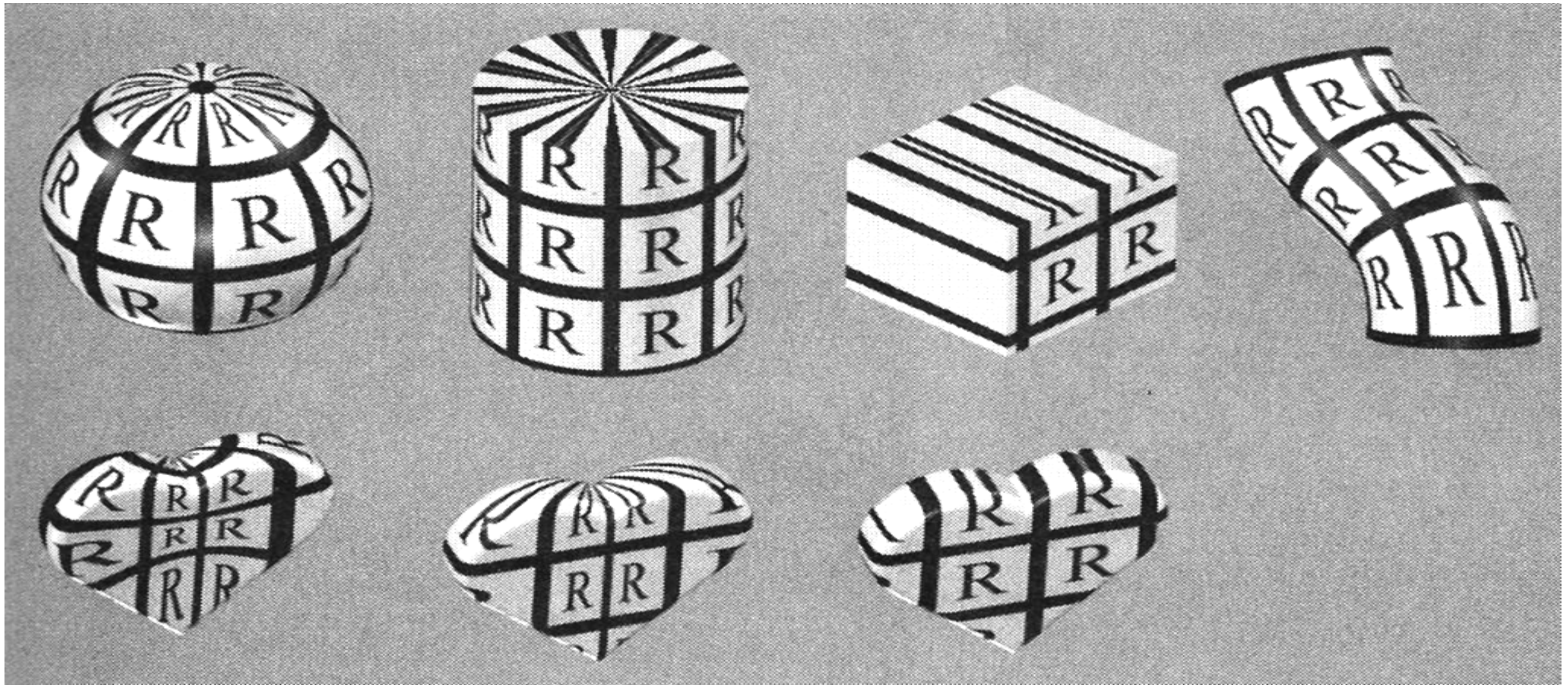


Texture Projectors

Where do texture coordinates come from?

- Online: texture matrix/texcoord generation
- Offline: manually (or by modeling program)

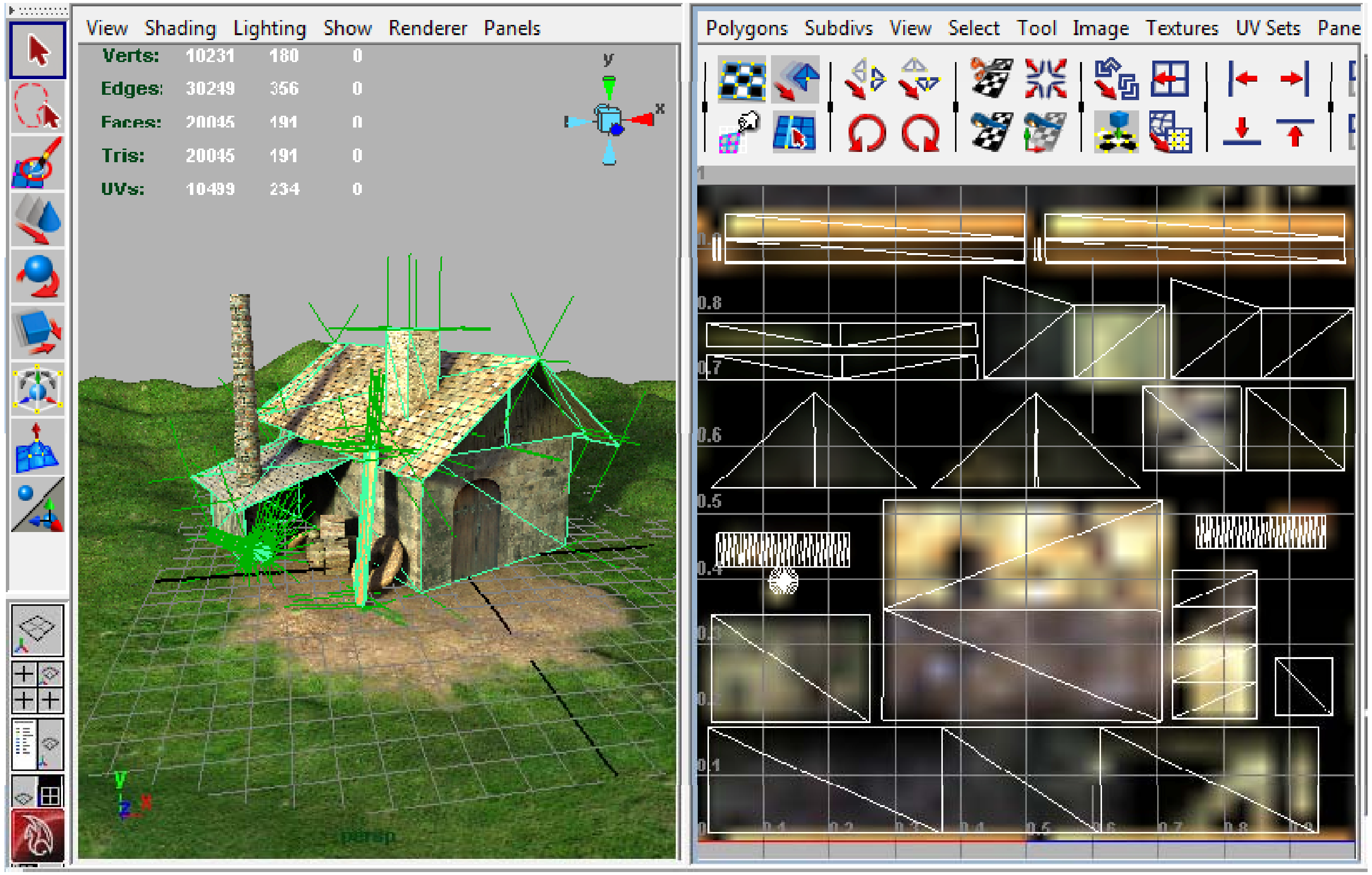
spherical *cylindrical* *planar* *natural*



Texture Projectors

Where do texture coordinates come from?

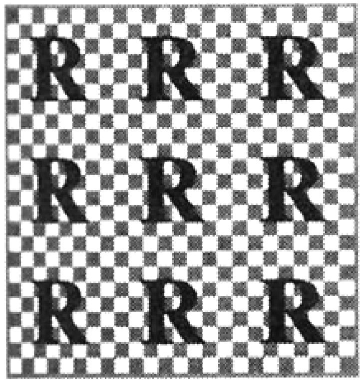
- Offline: manual UV coordinates by DCC program
- Note: a modeling problem!



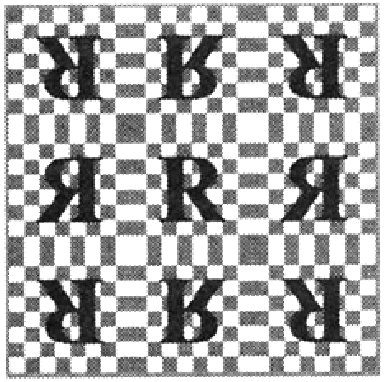
Texture Wrap Mode

- How to extend texture beyond the border?
- Border and repeat/clamp modes
- Arbitrary (s,t,…) → [0,1] x [0,1] → [0,255] x [0,255]

repeat

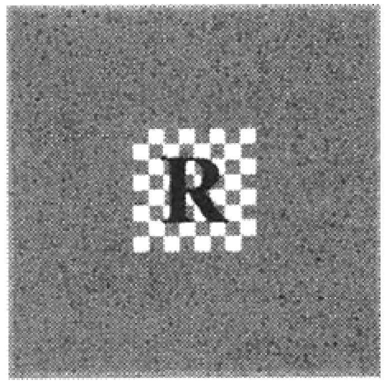


mirror/repeat



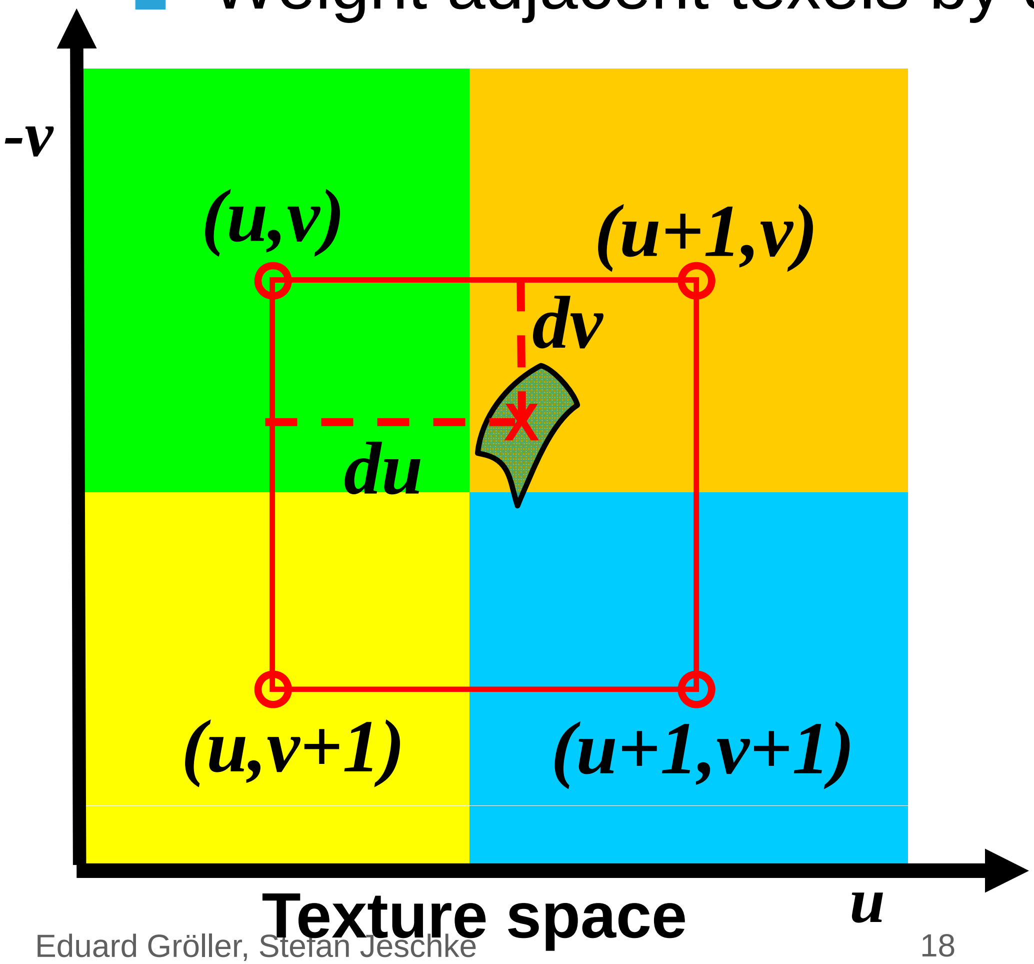
clamp

border



Texture Reconstruction: Magnification

- Bilinear reconstruction for texture magnification ($D<0$) (*"upsampling"*)
  - Weight adjacent texels by distance to pixel position



$$
\begin{aligned}
&T(u+du,v+dv) \\
&= du \cdot dv \cdot T(u+1,v+1) \\
&+ du \cdot (1-dv) \cdot T(u+1,v) \\
&+ (1-du) \cdot dv \cdot T(u,v+1) \\
&+ (1-du) \cdot (1-dv) \cdot T(u,v)
\end{aligned}
$$

Magnification (Bilinear Filtering Example)

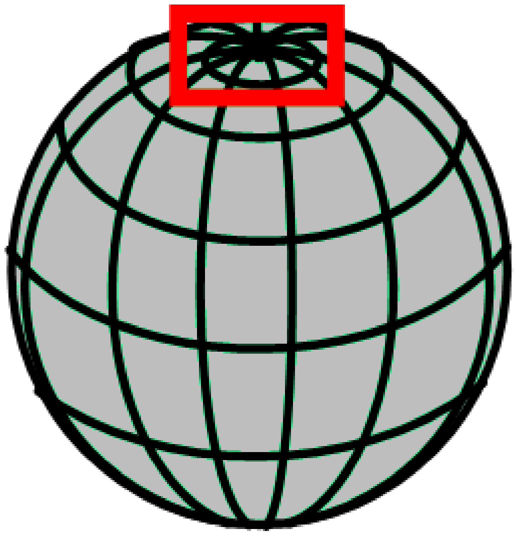
Original image

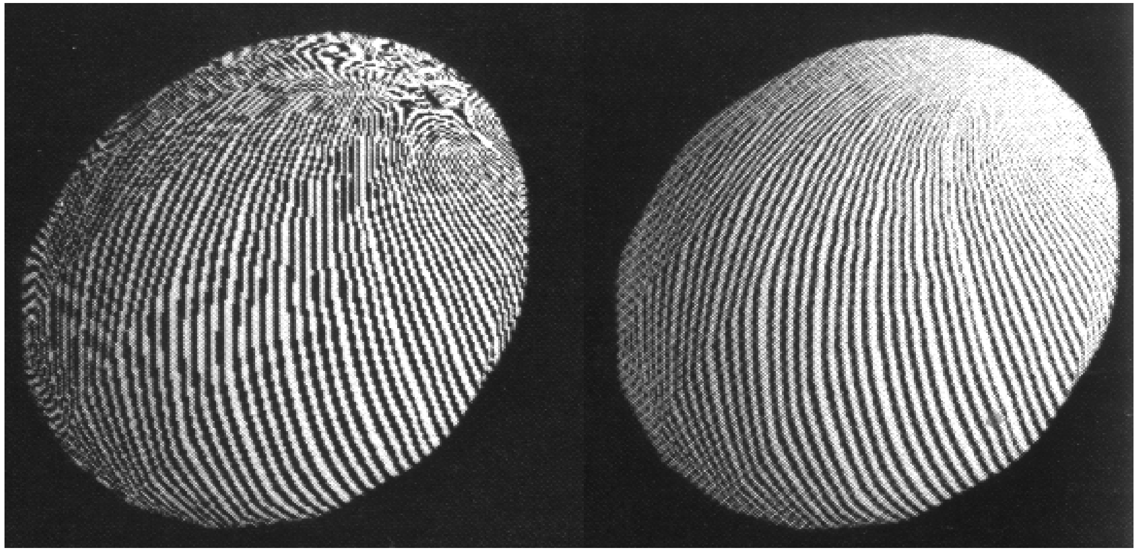
Nearest neighbor

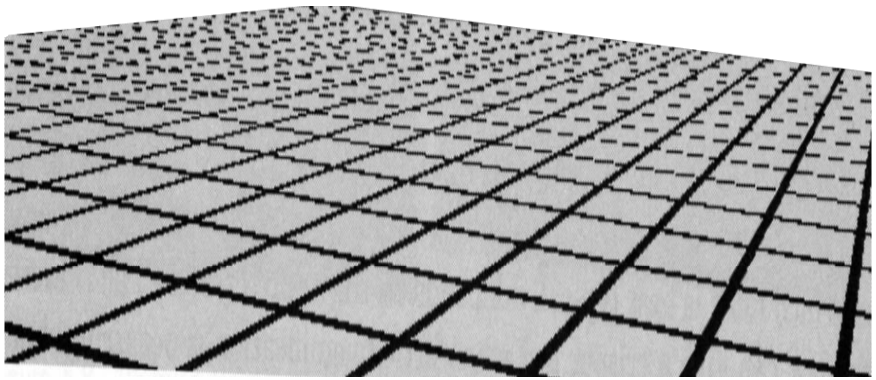
Bilinear filtering

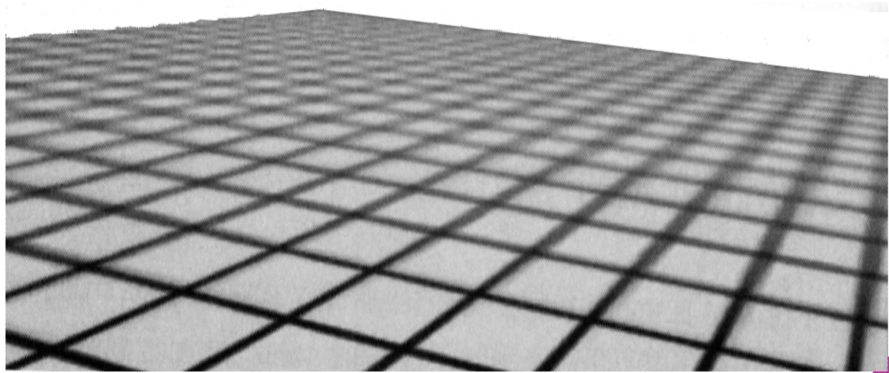
Texture Aliasing: Minification

- Problem: One pixel in image space covers many texels



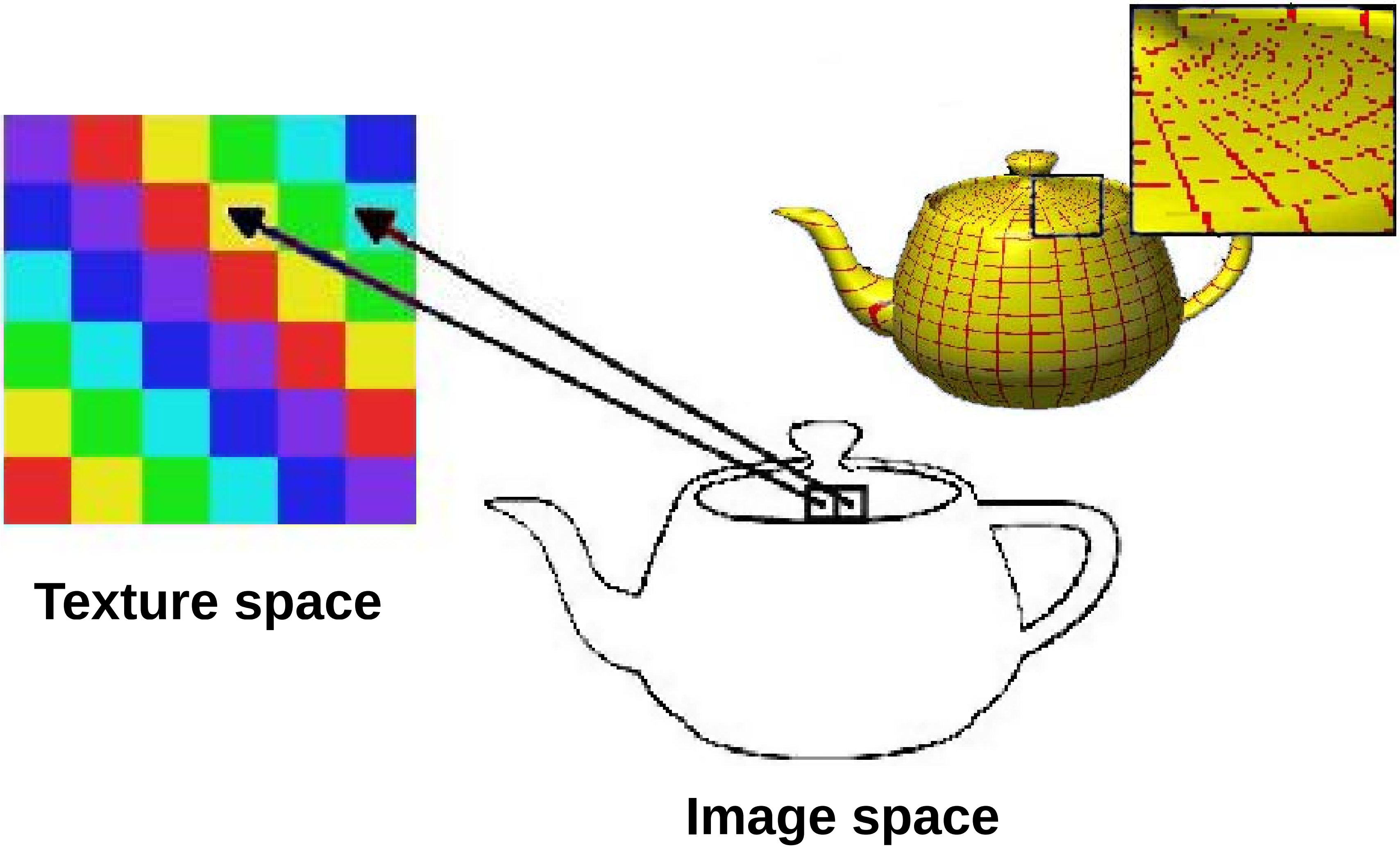






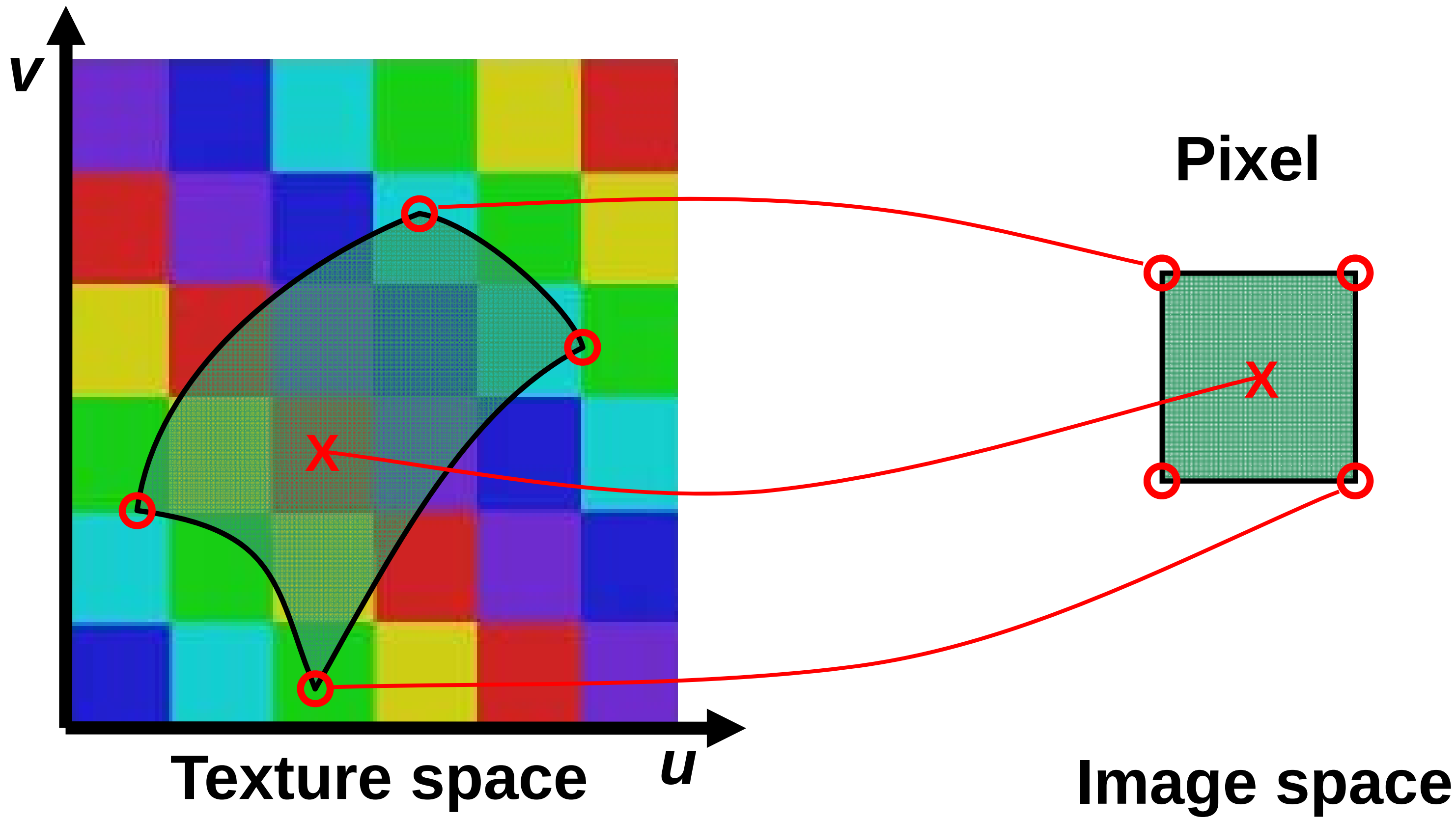
Texture Aliasing: Minification

- Caused by *undersampling*: texture information is lost



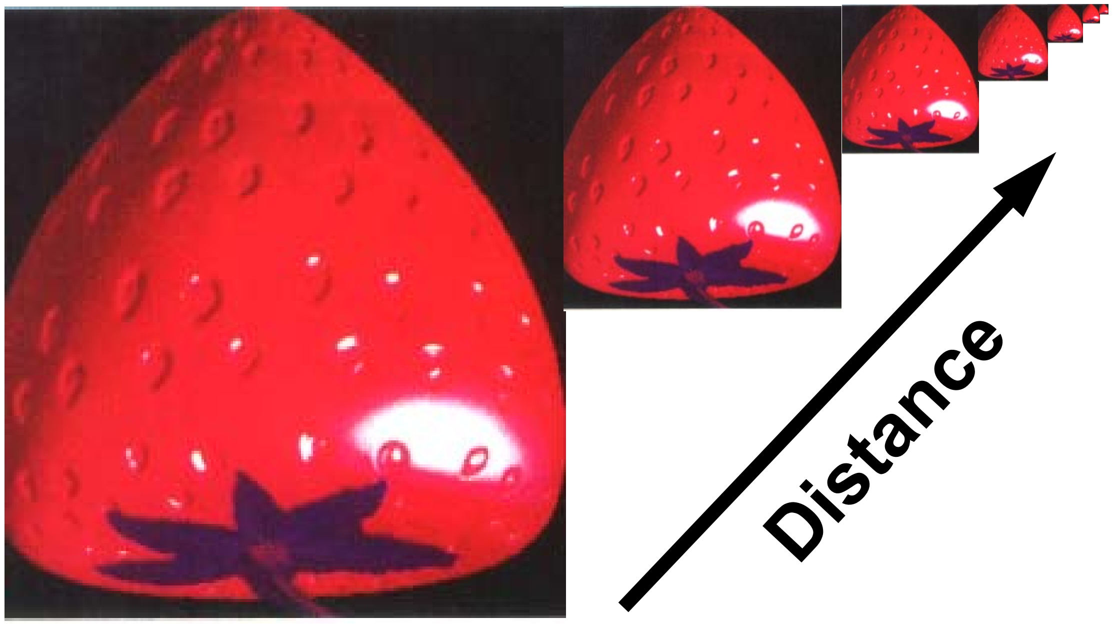
Texture Anti-Aliasing: Minification

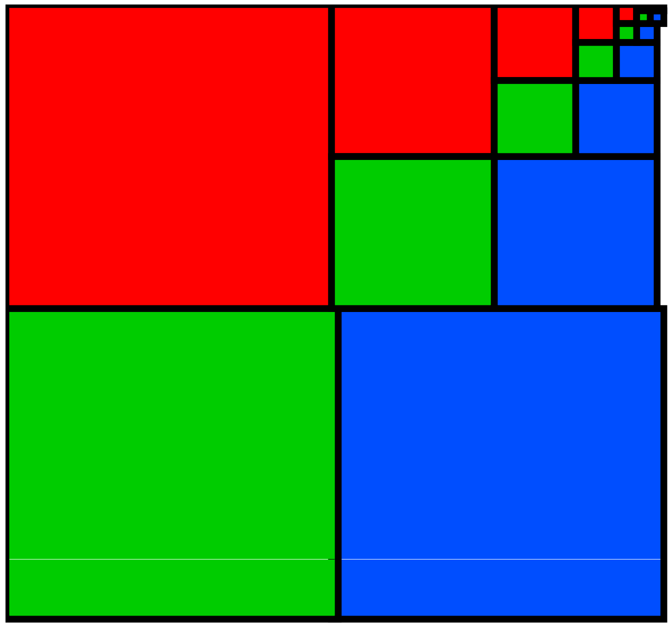
- A good pixel value is the weighted mean of the pixel area projected into texture space



Texture Anti-Aliasing: MIP Mapping

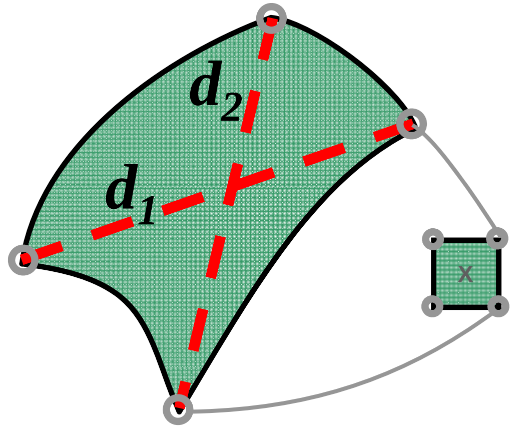
- MIP Mapping (“Multum In Parvo”)
  - Texture size is reduced by factors of 2 (*downsampling* = "many things in a small place")
  - Simple (4 pixel average) and memory efficient
  - Last image is only ONE texel

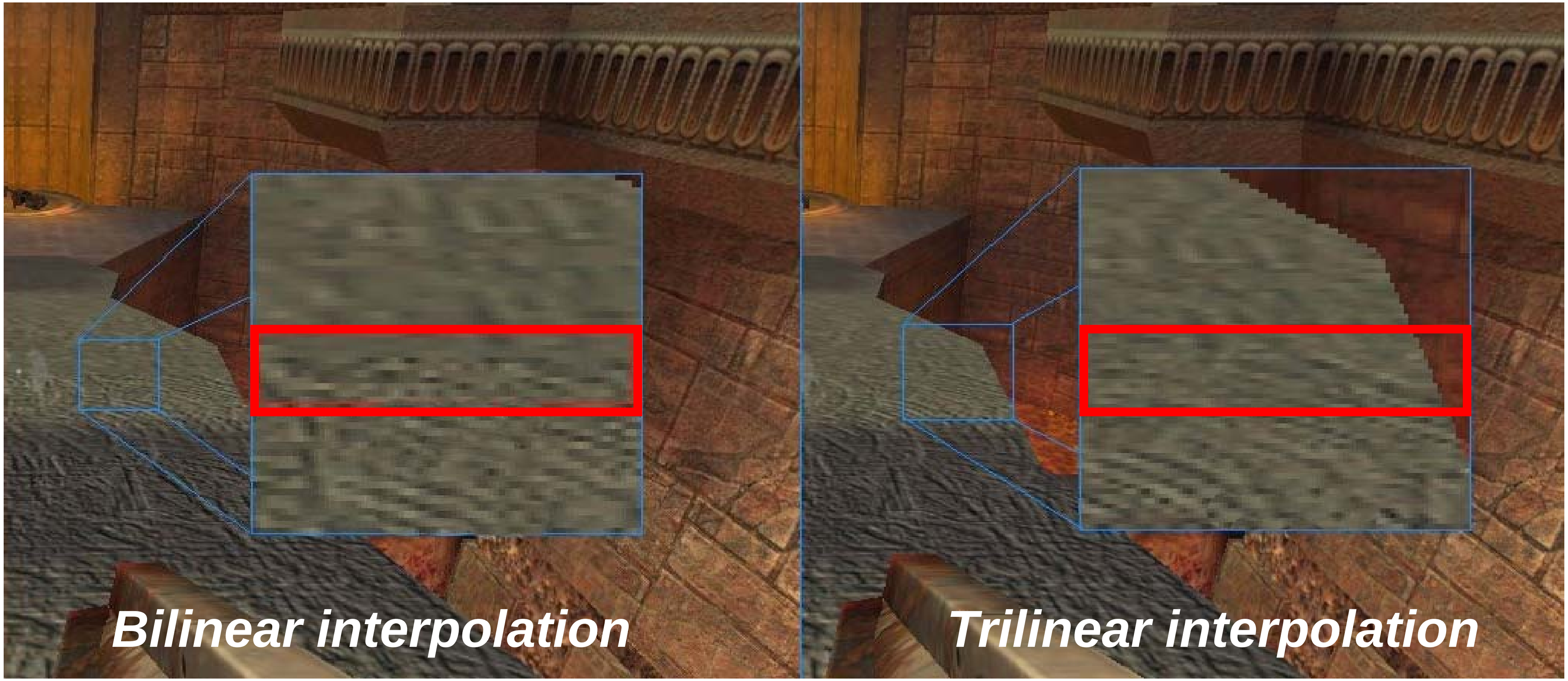




Texture Anti-Aliasing: MIP Mapping

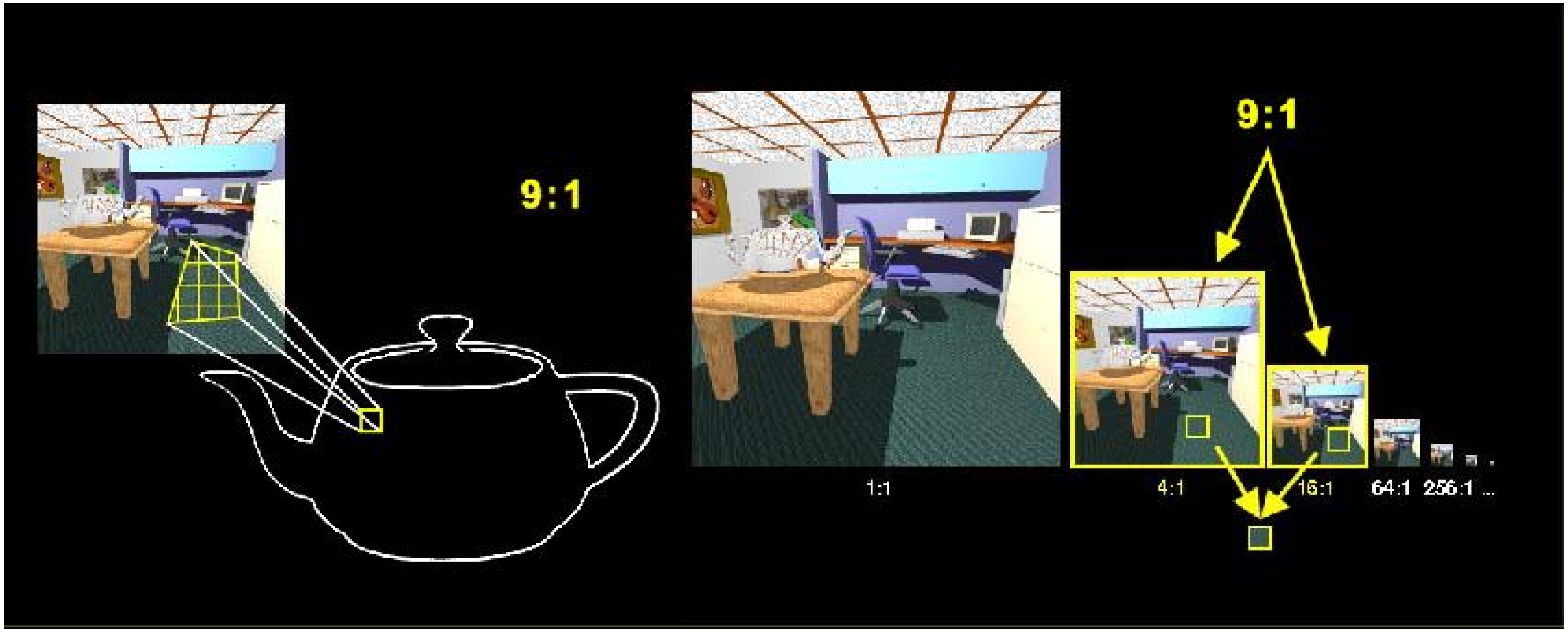
- MIP Mapping Algorithm
- $D := ld(max(d_1,d_2))$ *"Mip Map level"*
- T_0 := value from texture $D_0 = trunc\ (D)$
  - Use *bilinear interpolation*





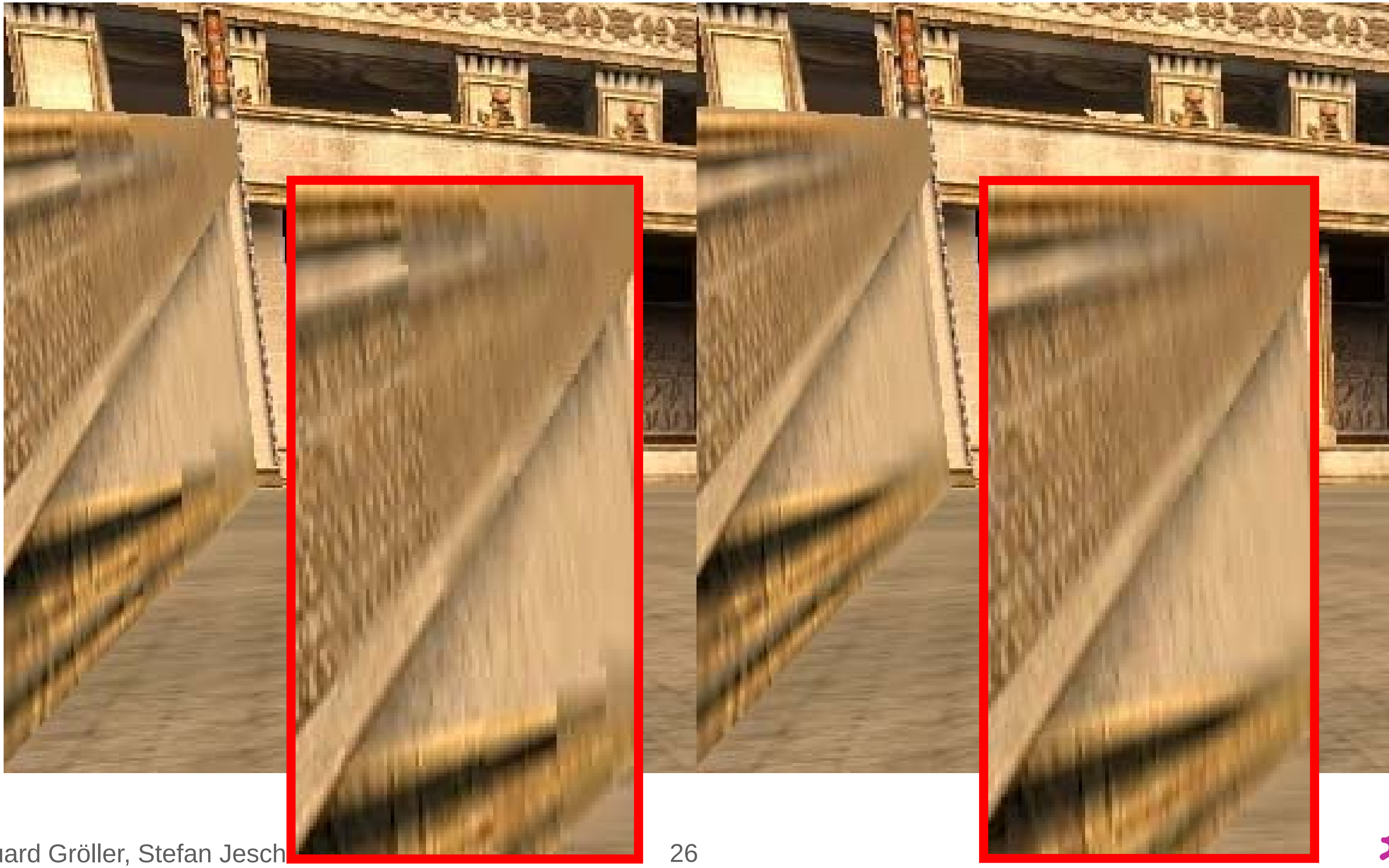
Texture Anti-Aliasing: MIP Mapping

- Trilinear interpolation:
  - T_1 := value from texture $D_1 = D_0 + 1$ (bilin.interpolation)
  - Pixel value $:= (D_1 - D) \cdot T_0 + (D - D_0) \cdot T_1$
    - Linear interpolation between successive MIP Maps
  - Avoids "Mip banding" (but doubles texture lookups)



Texture Anti-Aliasing: MIP Mapping

- Other example for bilinear vs. trilinear filtering



Thank you.

Thanks for slides and images

- Michael Wimmer, Stefan Jeschke, Meister Eduard Gröller, TU Vienna