Texturing

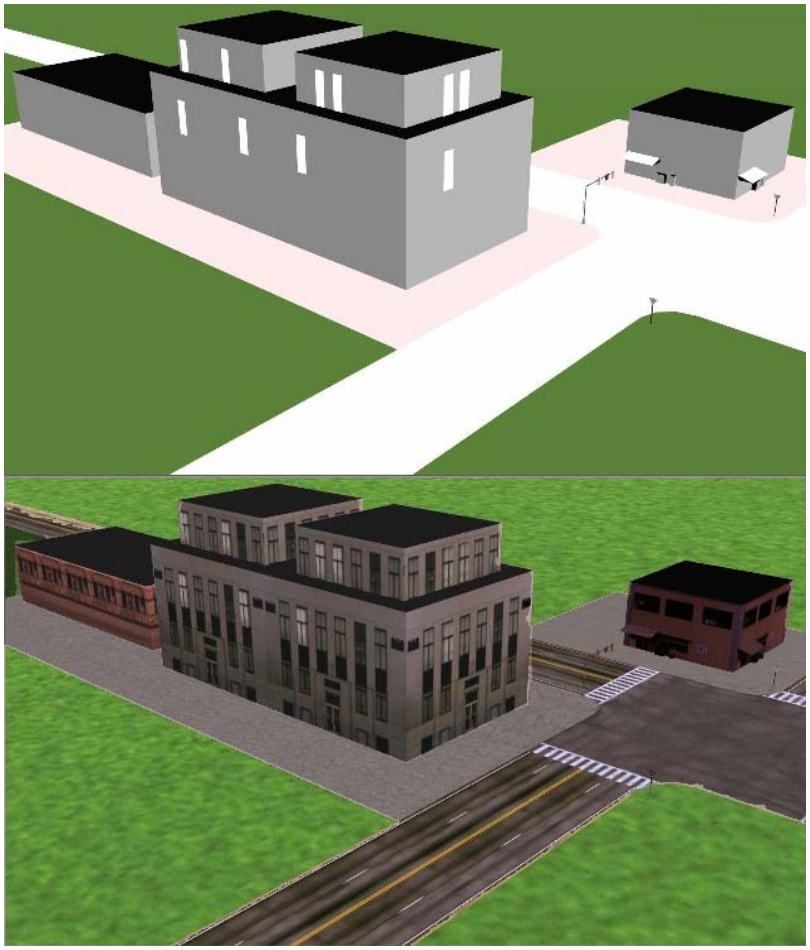
Peter Sikachev,
Stefan Jeschke,
Eduard Gröller

Institute of Computer Graphics and Algorithms

Vienna University of Technology

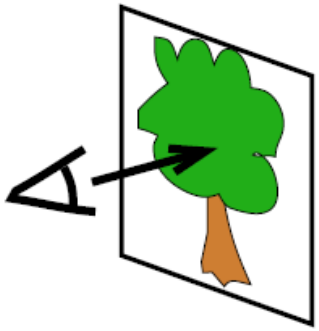
Why Texturing?

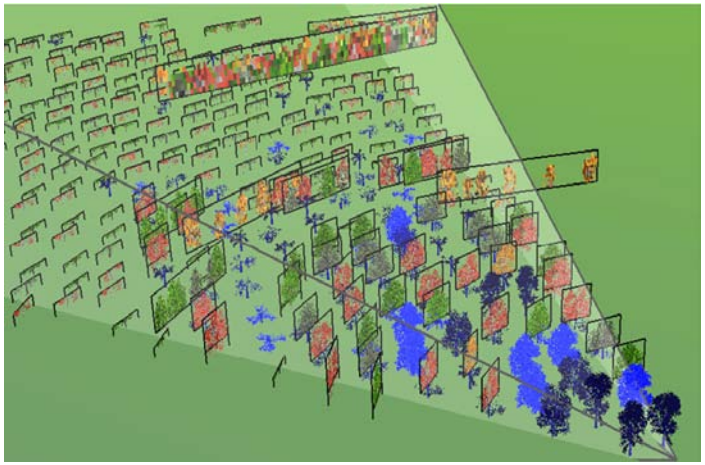
- Idea: enhance visual appearance of plain surfaces by applying fine structured details

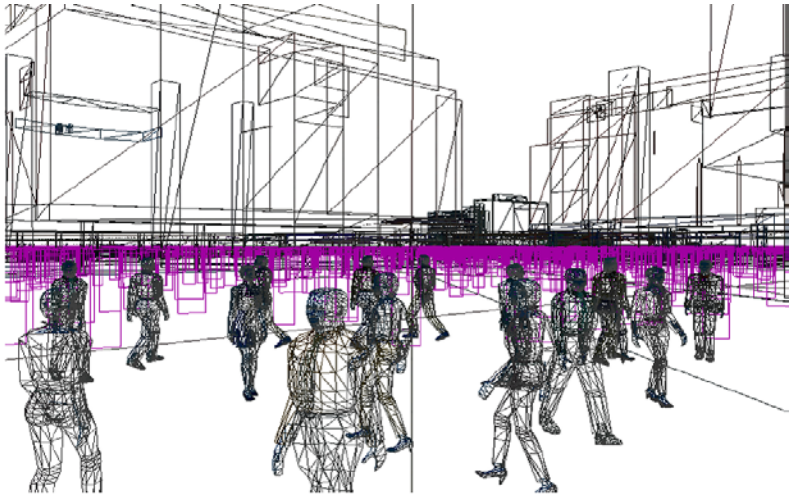


Why Texturing?

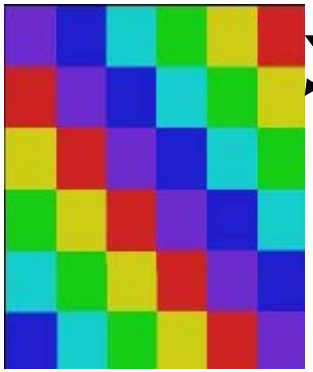
- Also possible: model very complex objects just by using simple textured geometry

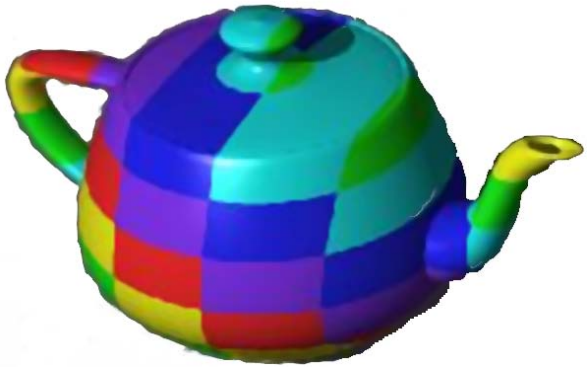






Texturing: General Approach





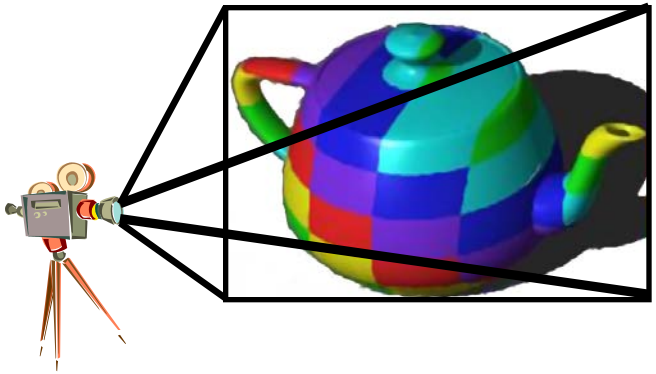
Texture space *(u,v)*

Object space $(\bar{x}, \bar{y}, \bar{z})$

Image Space (x_I, y_I)

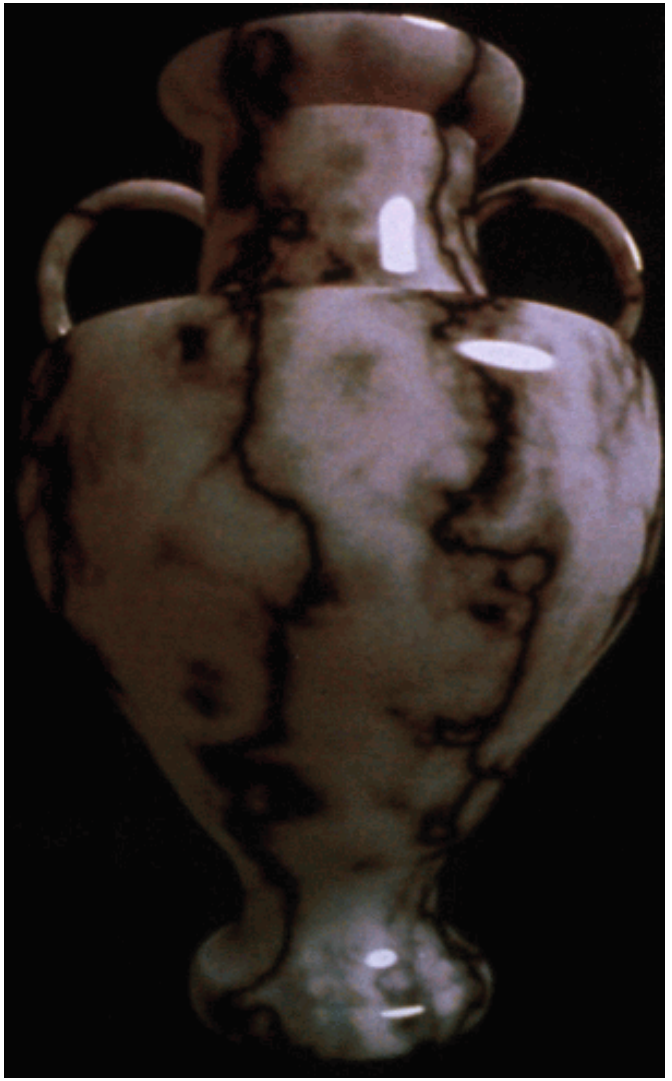
Parametrization

Rendering (Projection etc.)



Procedural Texturing

- Sampled textures drawbacks
  - Take a lot of memory
  - Texture fetches are slow
  - Parameterization is always needed
  - Aliasing
- Procedural textures
  - Texture is generated on-the-fly per fragment



Texture Definition

- A texture is a **function**:
  - *Procedural*: fast to evaluate, but limited variety
  - *Sampled*: most common method
    - Raster images that are taken with a digital camera, scanned or synthesized
- Textures can be defined:
  - In 3D object space: *“3D texturing“, “cube mapping”*
  - On the 2D object surface: *"texture mapping“*
  - As a 1D function: *“lookup table“*

Cube Mapping

- Encoding of environment is needed
- 3D texture is an overkill: why?

Cube Mapping

- Encoding of environment is needed
- 3D texture is an overkill:
  - Memory

Cube Mapping

- Encoding of environment is needed
- 3D texture is an overkill:
  - Memory
  - Takes time to generate

Cube Mapping

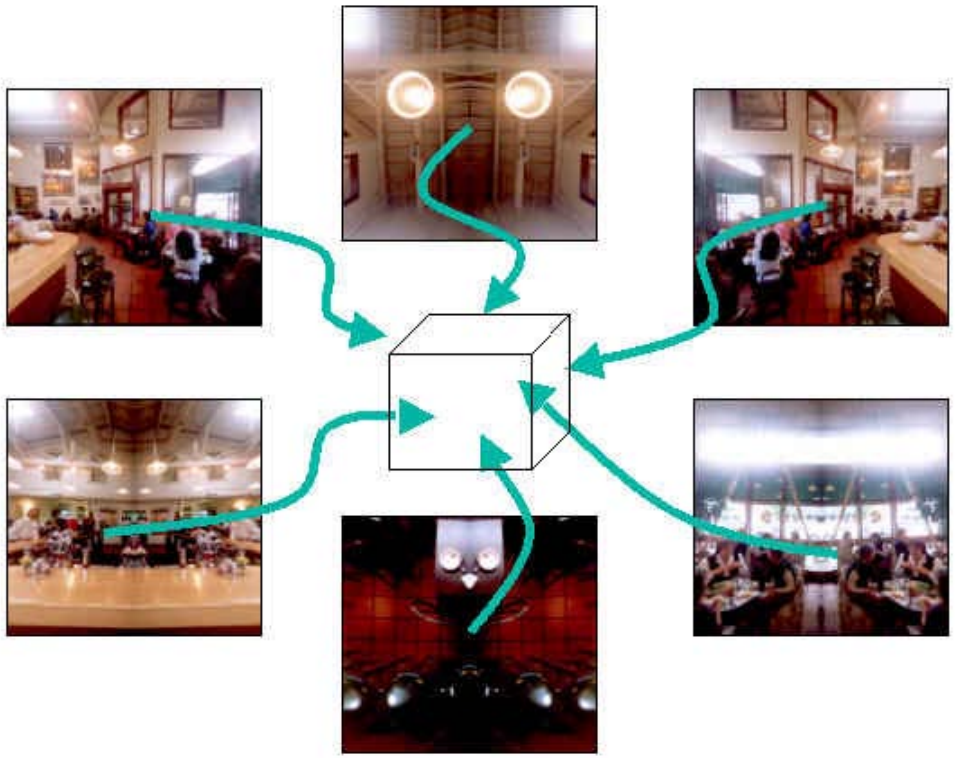
- Encoding of environment is needed
- 3D texture is an overkill:
  - Memory
  - Takes time to generate
  - Takes time to sample

Cube Mapping

- Encoding of environment is needed
- 3D texture is an overkill:
  - Memory
  - Takes time to generate
  - Takes time to sample
- Solutions?

Cube Mapping

- Encoding of environment is needed
- 3D texture is an overkill:
  - Memory
  - Takes time to generate
  - Takes time to sample
- Solution:
  - Take 6 photos from scene center
  - Texture is addressed via 3D vector, which has semantics of direction, not position

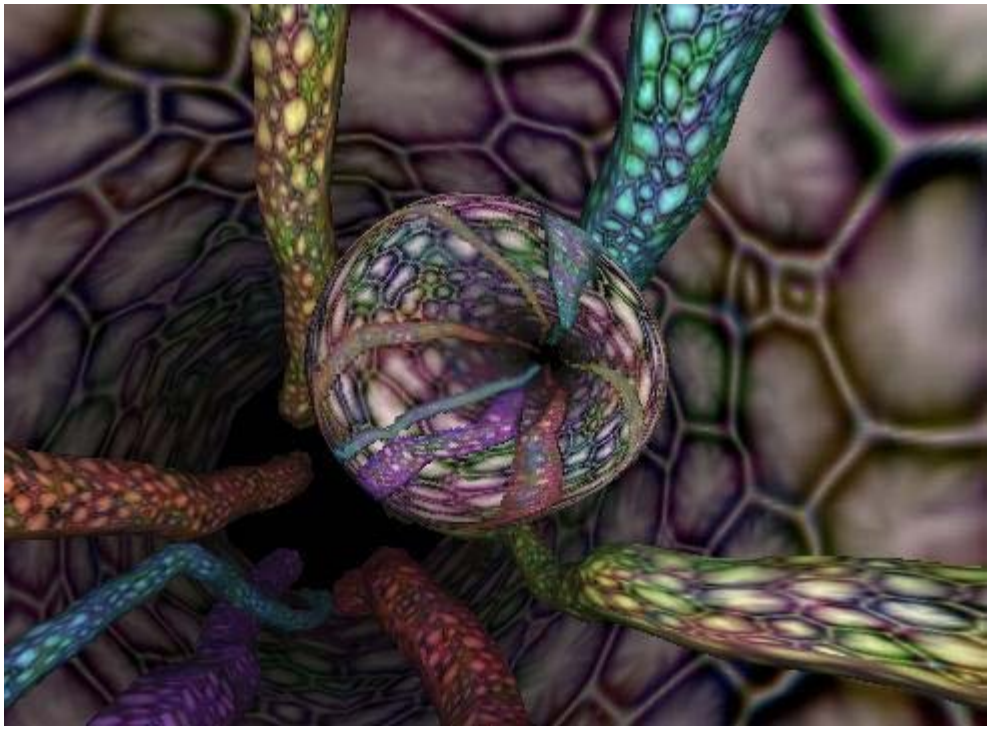


Environment Mapping

- Application: specular object in the environment

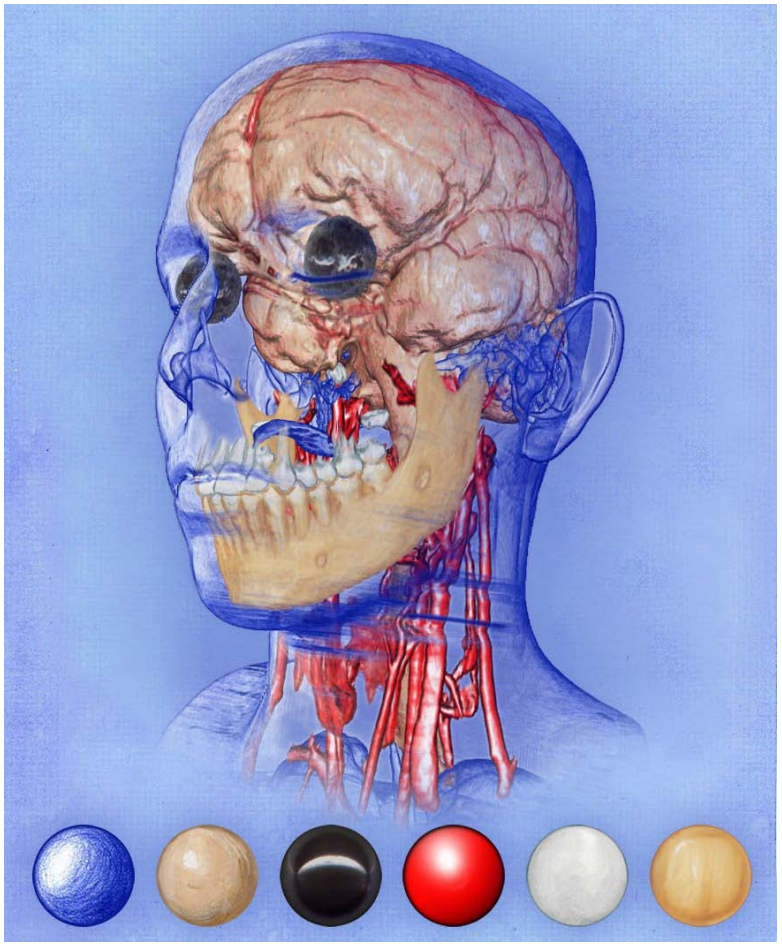
Cube map

Rendered scene



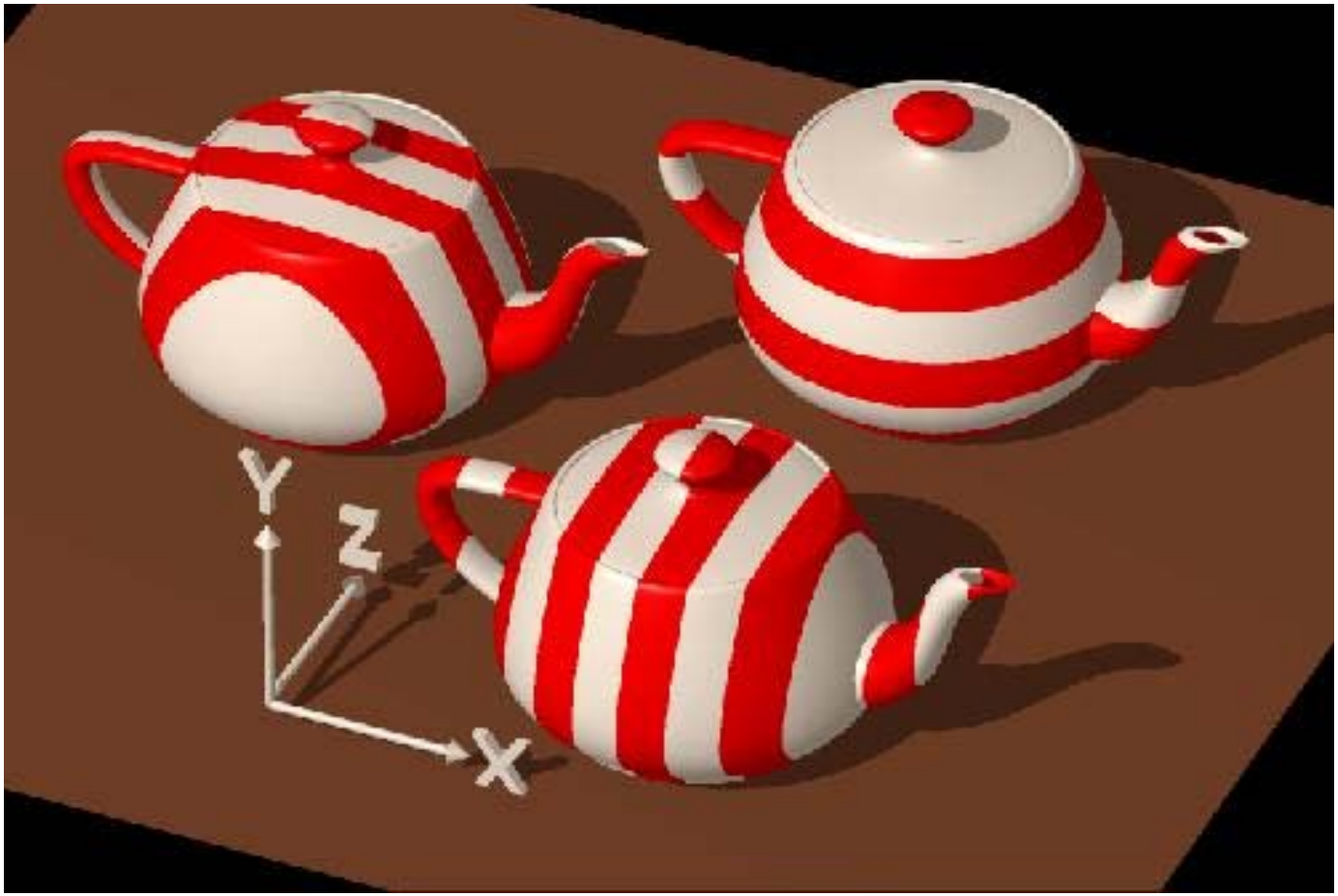
3D Texturing

- Not directly applied for polygonal models
- Representation of some 3D value field
- Applications:
  - Volume rendering
  - Media (gas/liquid) rendering
  - Hybrid rendering
  - etc



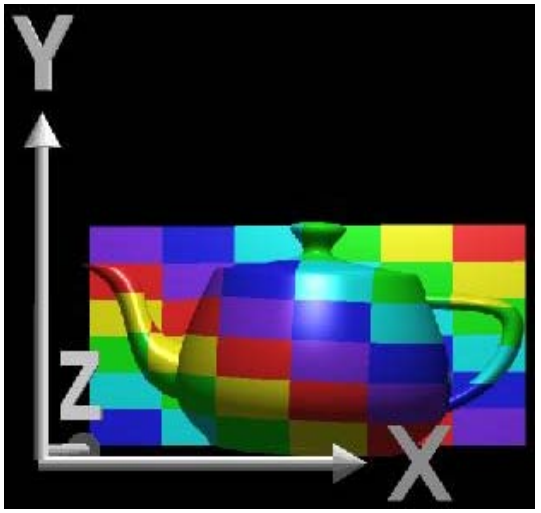
Texturing: Parametrization

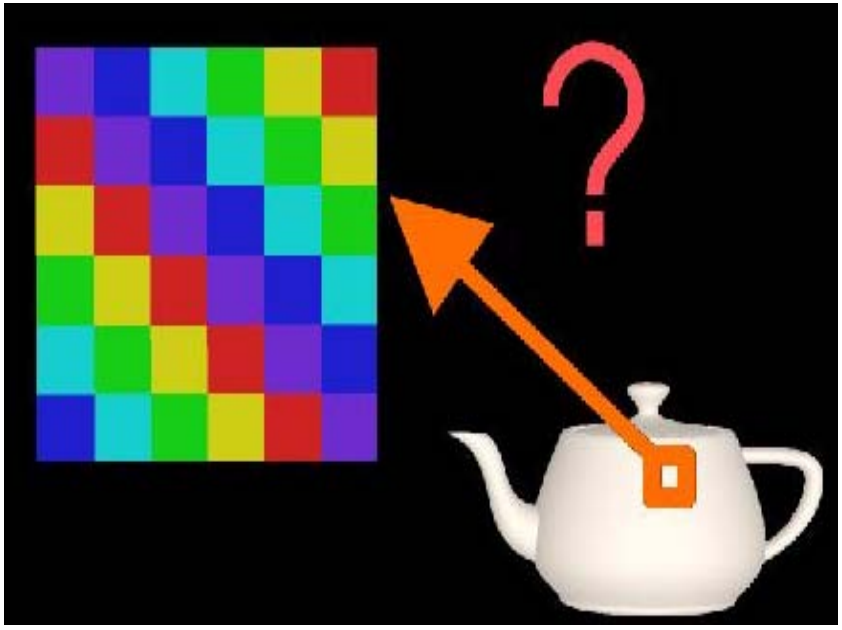
- ***1D*** texture: parameter can have arbitrary domain (along one axis, incident angle, etc.)

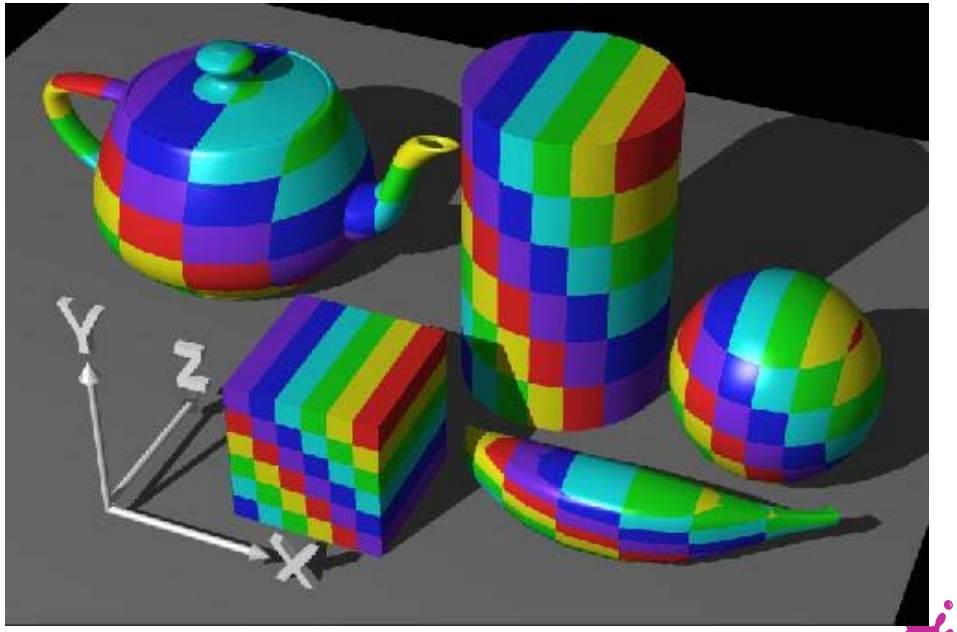


Texturing: Parametrization

- ***2D*** texture
  - Projection of 2D data
  - Many possibilities for definition

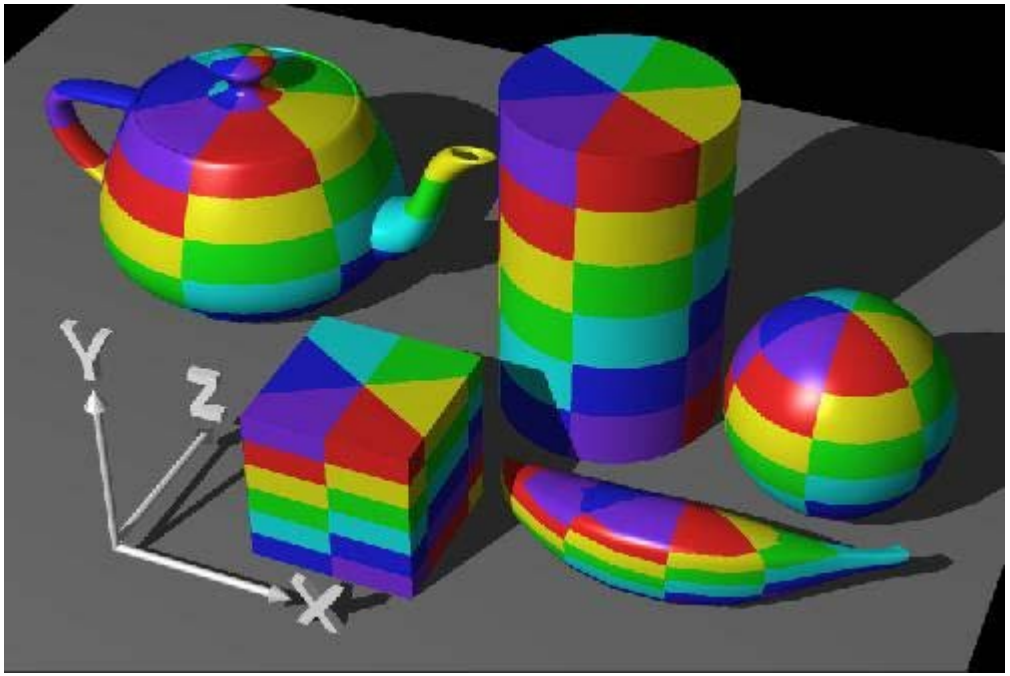






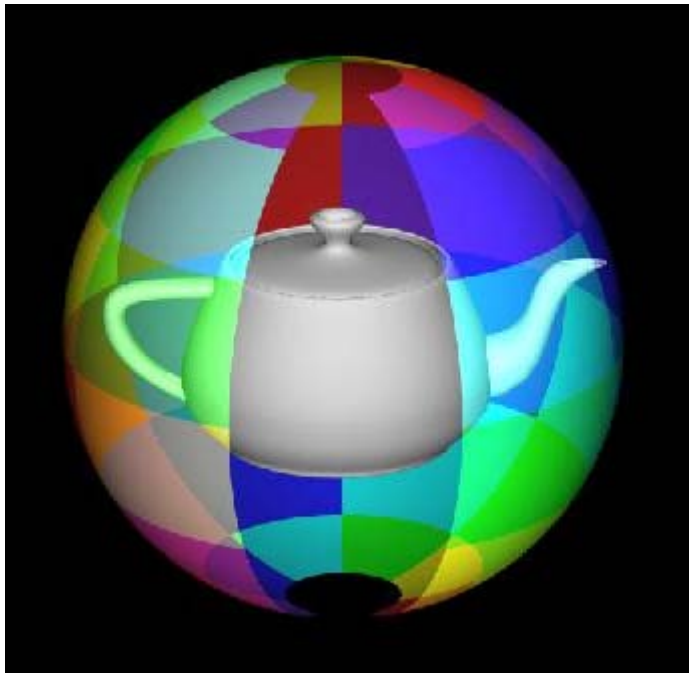
Texturing: Parametrization

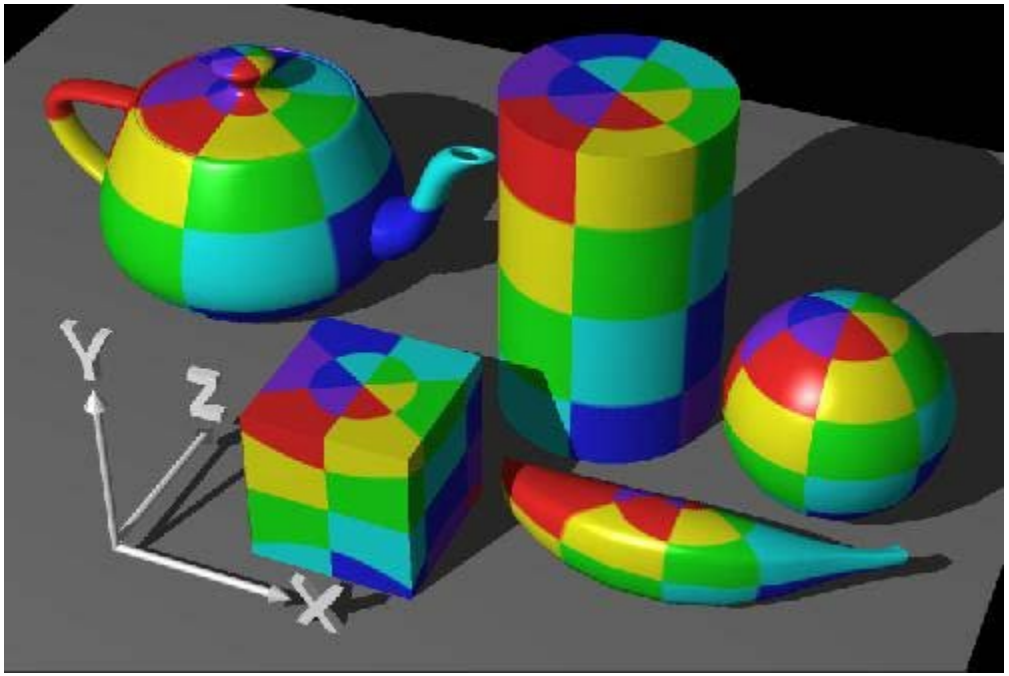
- *Other 2D texture: cylindrical parametrization*
  - Depending on cylindrical coordinates of each point



Texturing: Parametrization

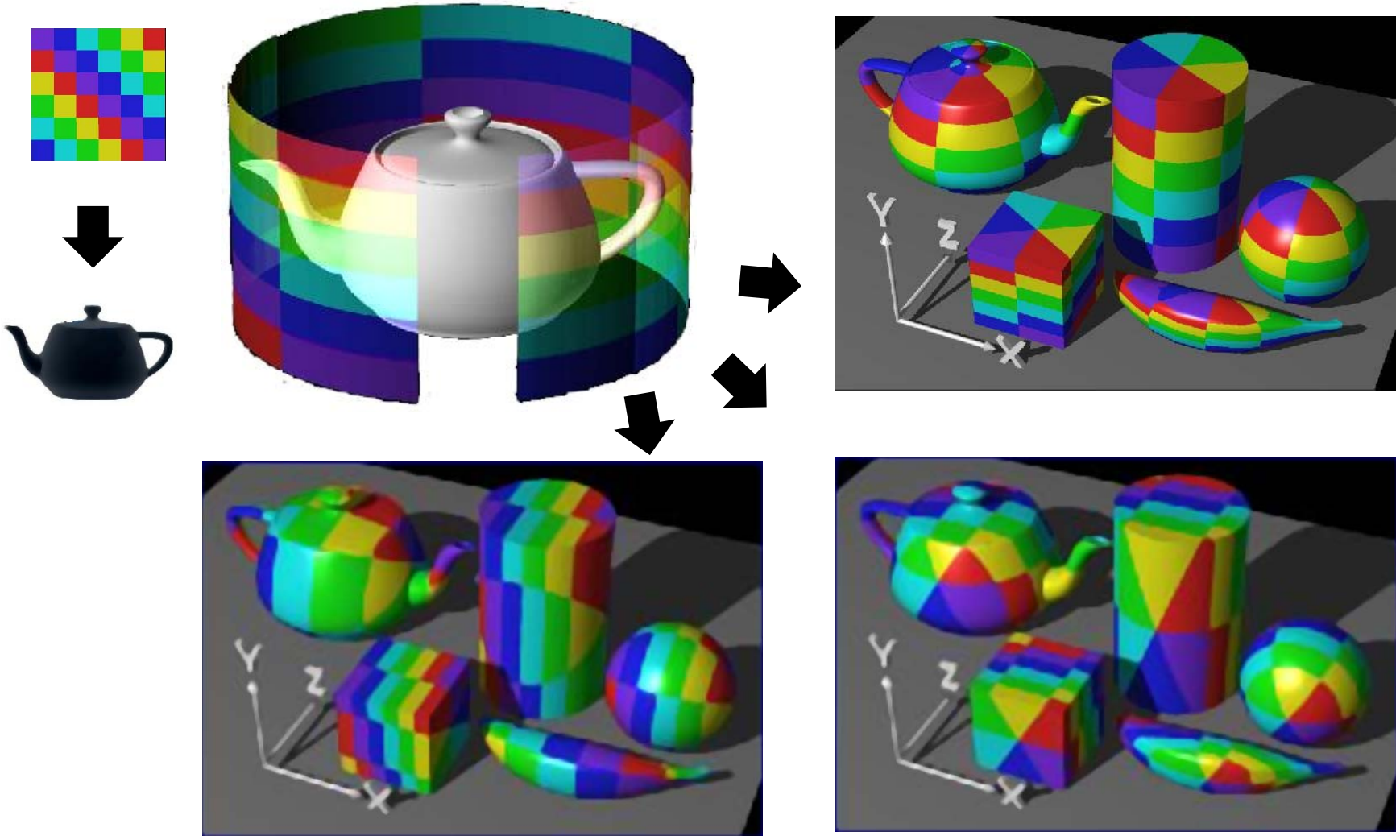
- *Other 2D texture: spherical parametrization*
  - Depending on spherical coordinates of each point





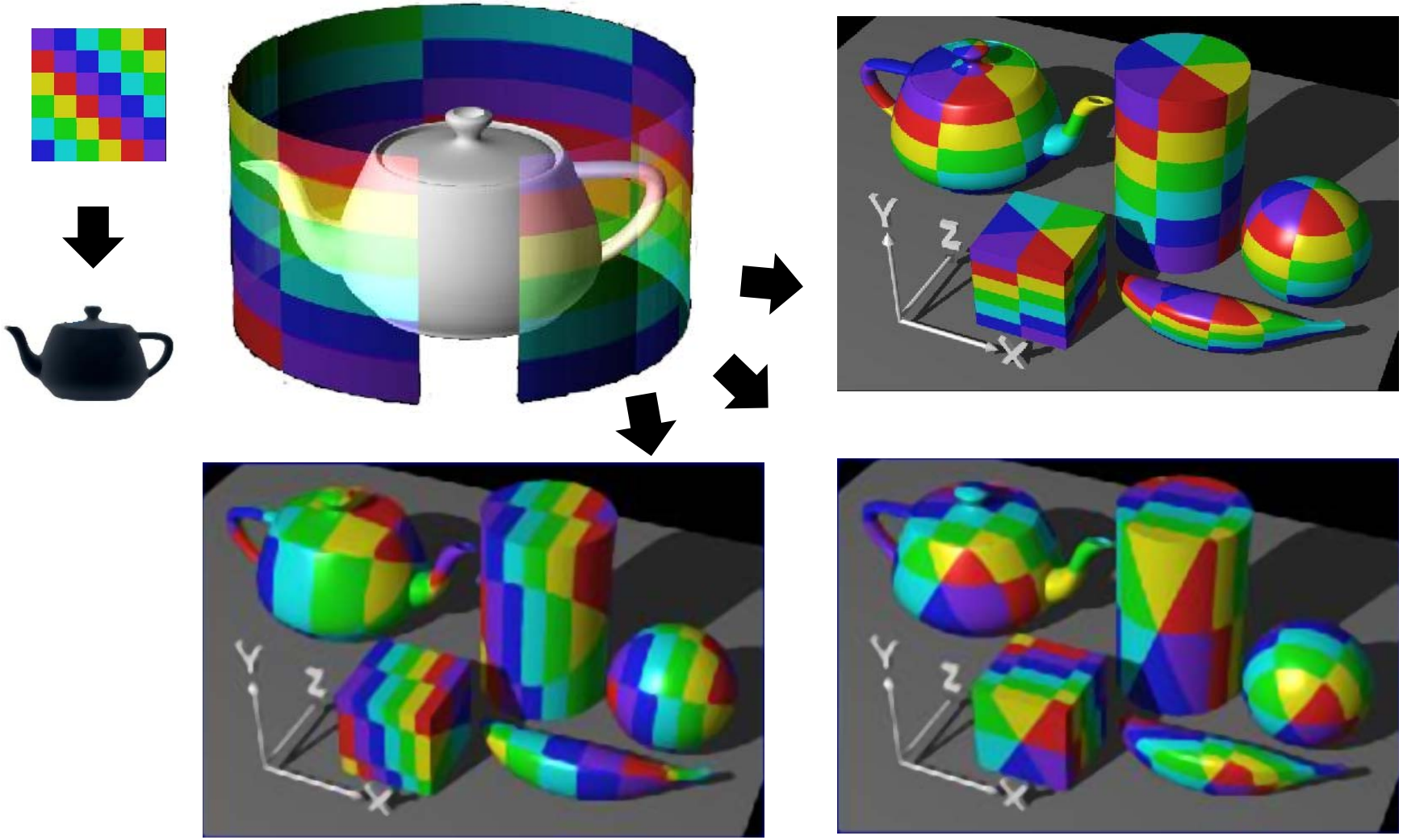
Texture Mapping: Parametrization

- Difficulty: how to minimize texture distortions?



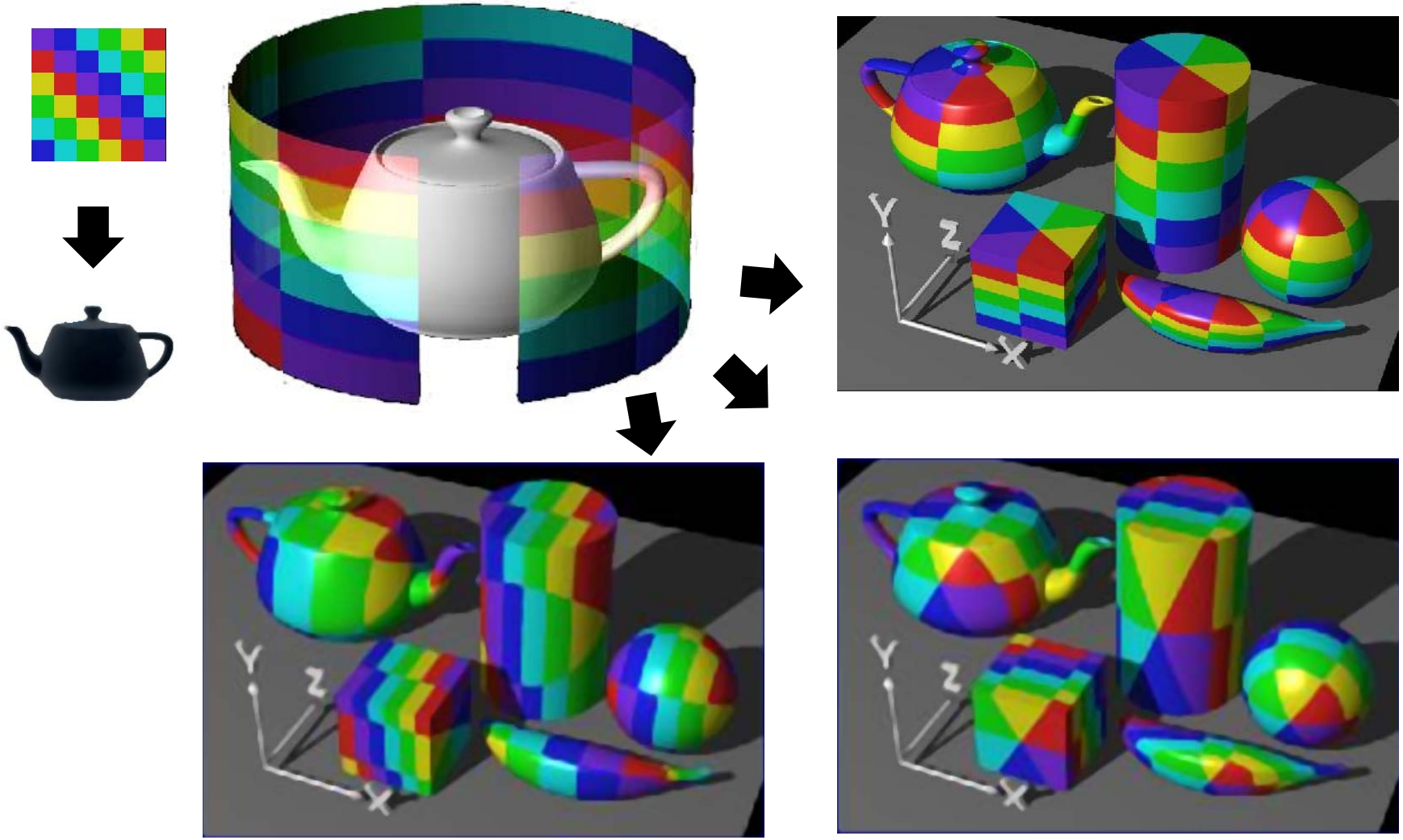
Texture Mapping: Parametrization

- Other problems?



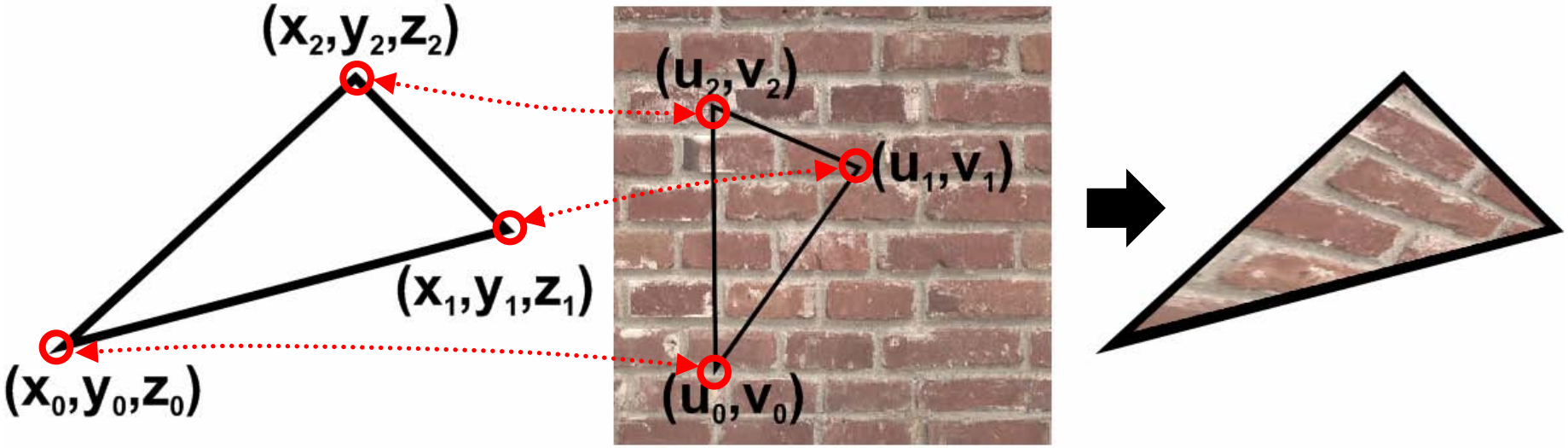
Texture Mapping: Parametrization

- Non-convex objects: mapping is not bijective



Texture Mapping

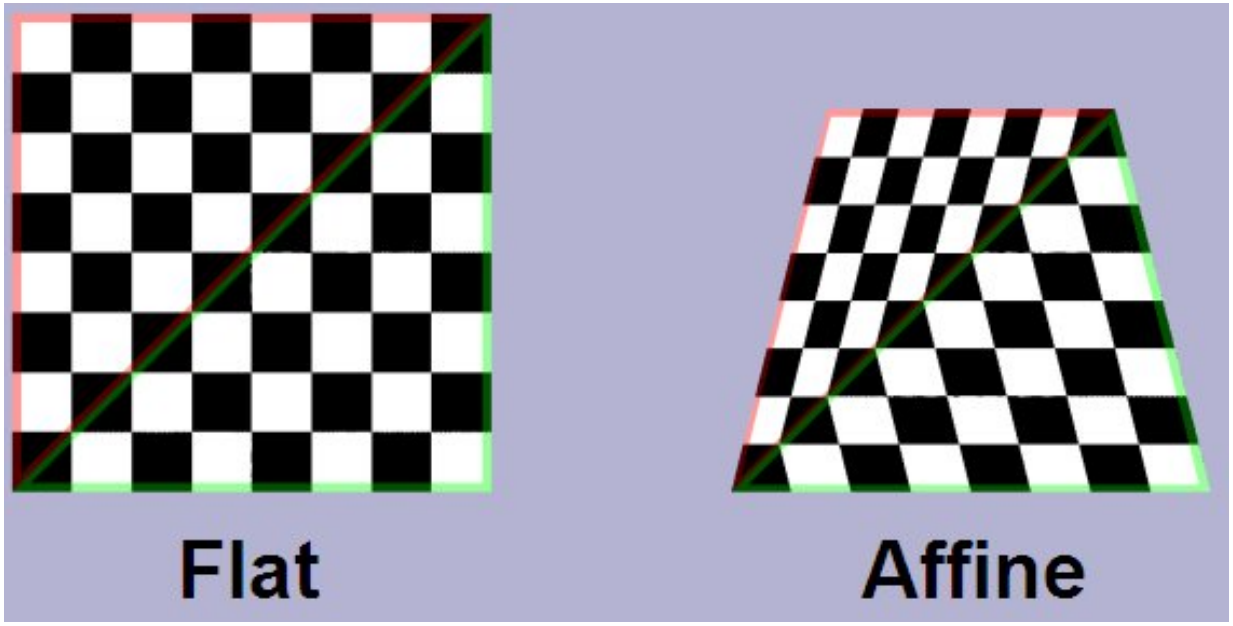
- Parametrization per vertex
  - Apply *(u,v)* texture coordinates to every vertex
- At runtime, ***(bi)linearly interpolate*** between these coordinates during rasterization



Perspective Correctness

- Problem: perspective correctness
  - Affine texture mapping does not take into account the depth information

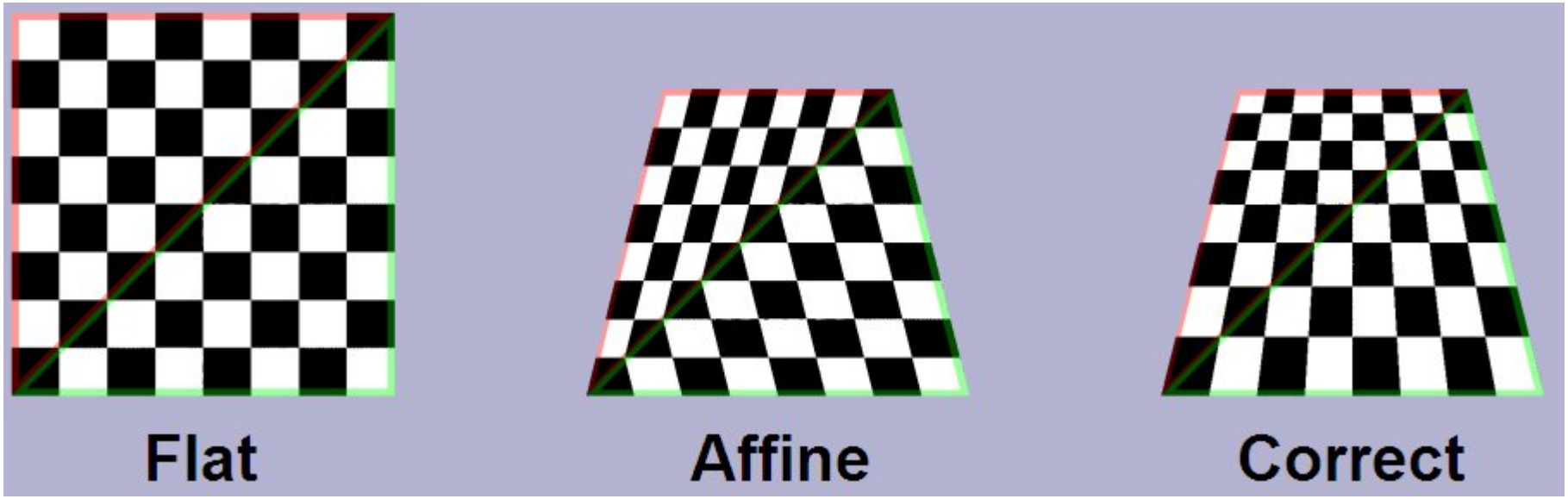
$$u_\alpha = (1-\alpha)u_0 + \alpha u_1; 0 \le \alpha \le 1$$



Perspective Correctness

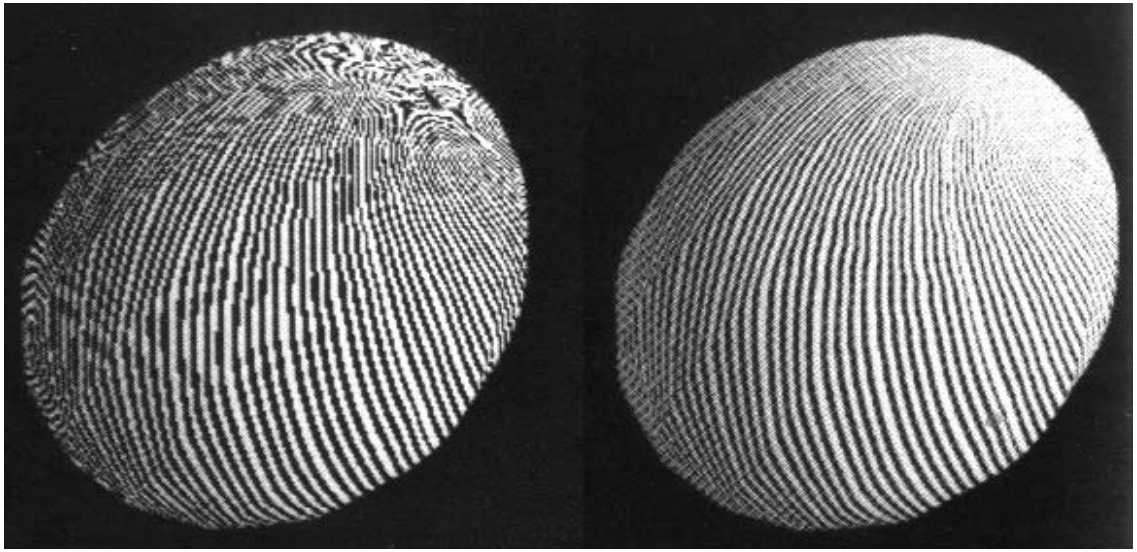
- Solution
  - Interpolate coordinates, divided by depth and depth reciprocals

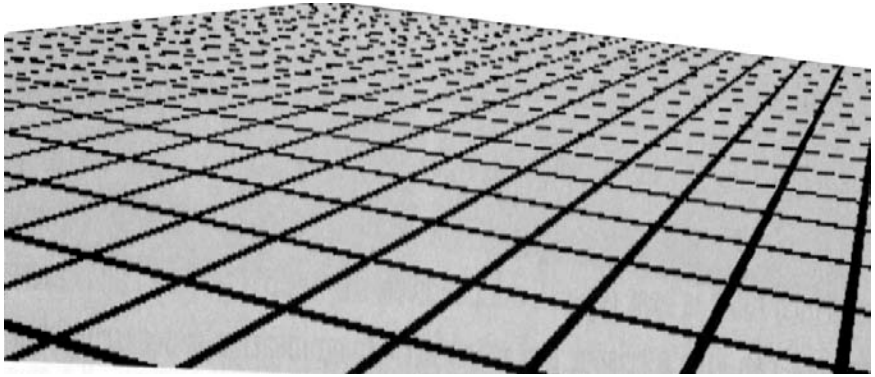
$$u_\alpha = \frac{(1-\alpha)\frac{u_0}{z_0} + \alpha\frac{u_1}{z_1}}{(1-\alpha)\frac{1}{z_0} + \alpha\frac{1}{z_1}}$$

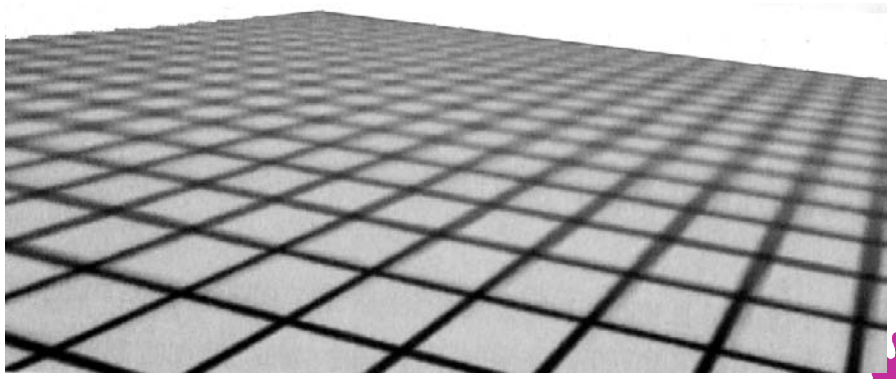


Aliasing

- Problem: aliasing
  - One pixel in image space covers many texels

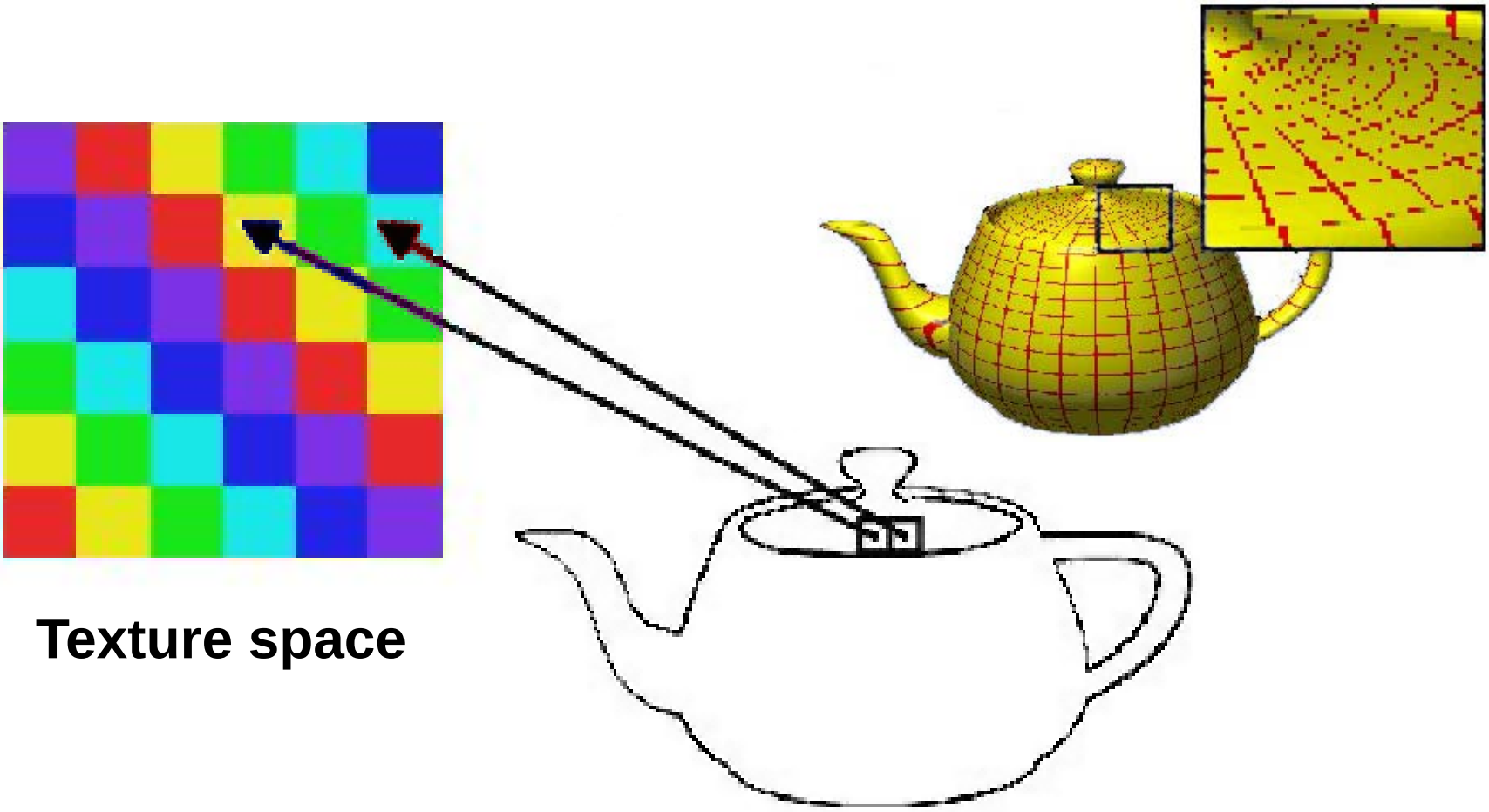






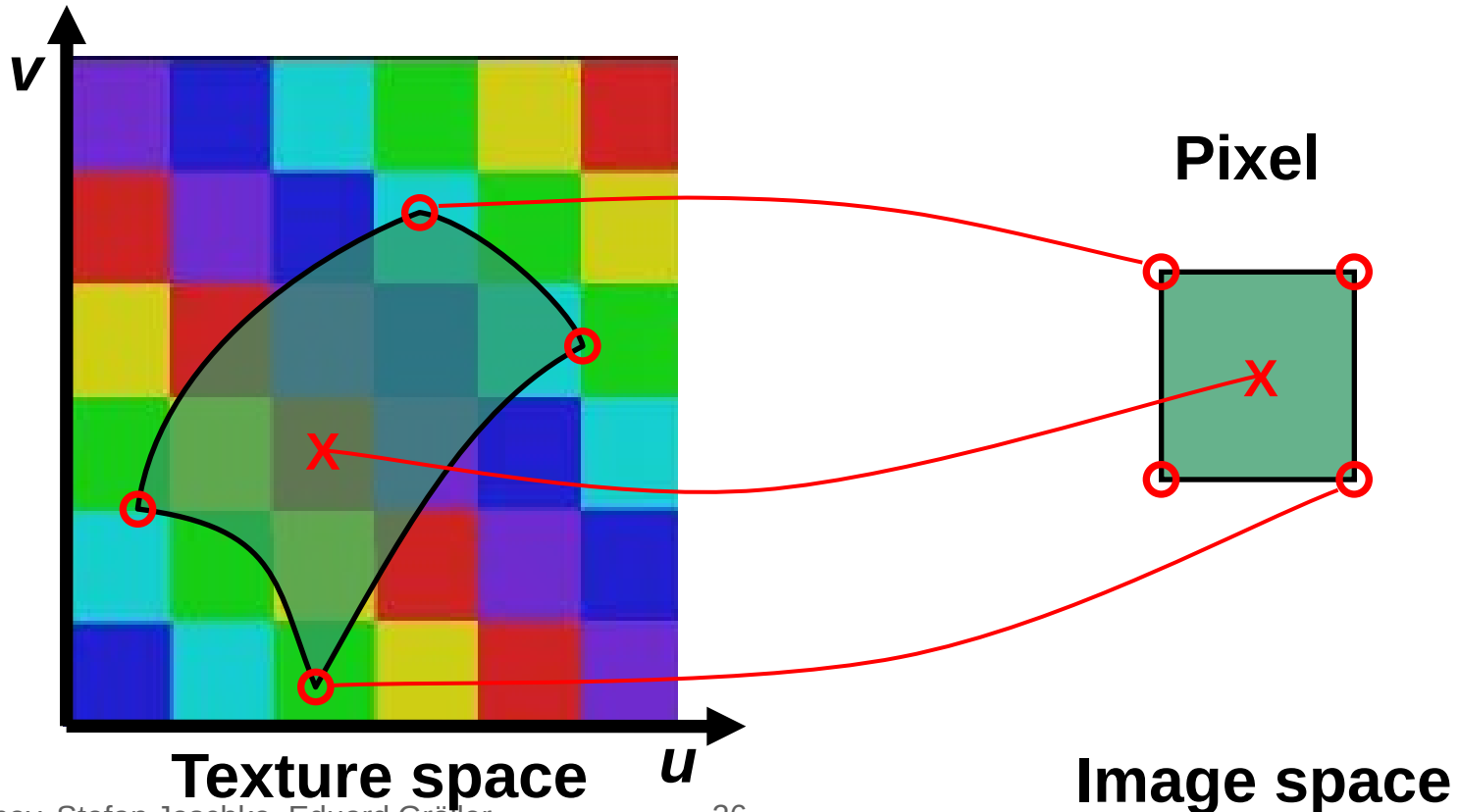
Aliasing

- Caused by *undersampling*: texture information is lost



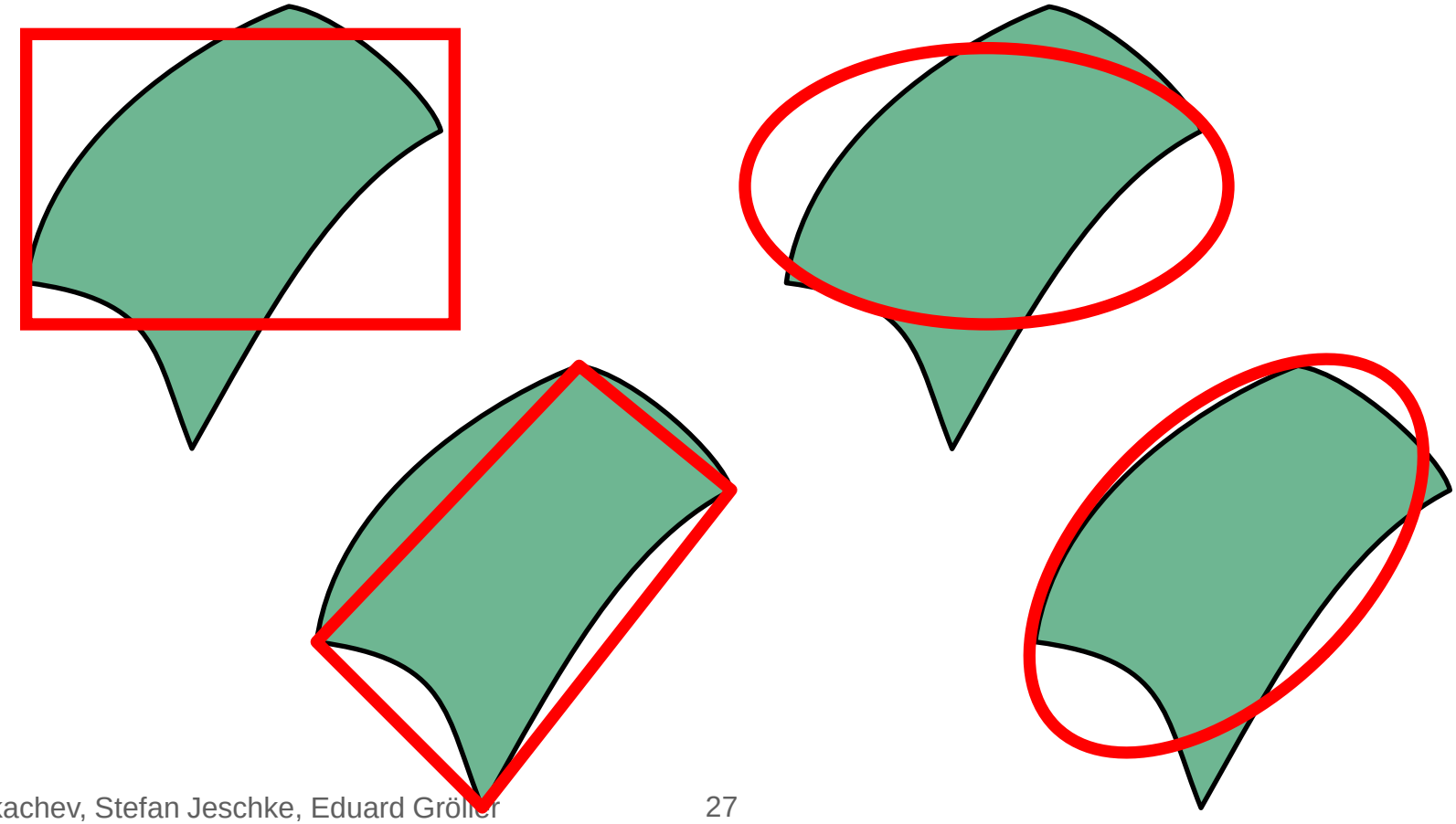
Anti-Aliasing

- A good pixel value is the weighted mean of the pixel area projected into texture space
  - Calculation at *runtime* or in a *preprocess*



Anti-Aliasing

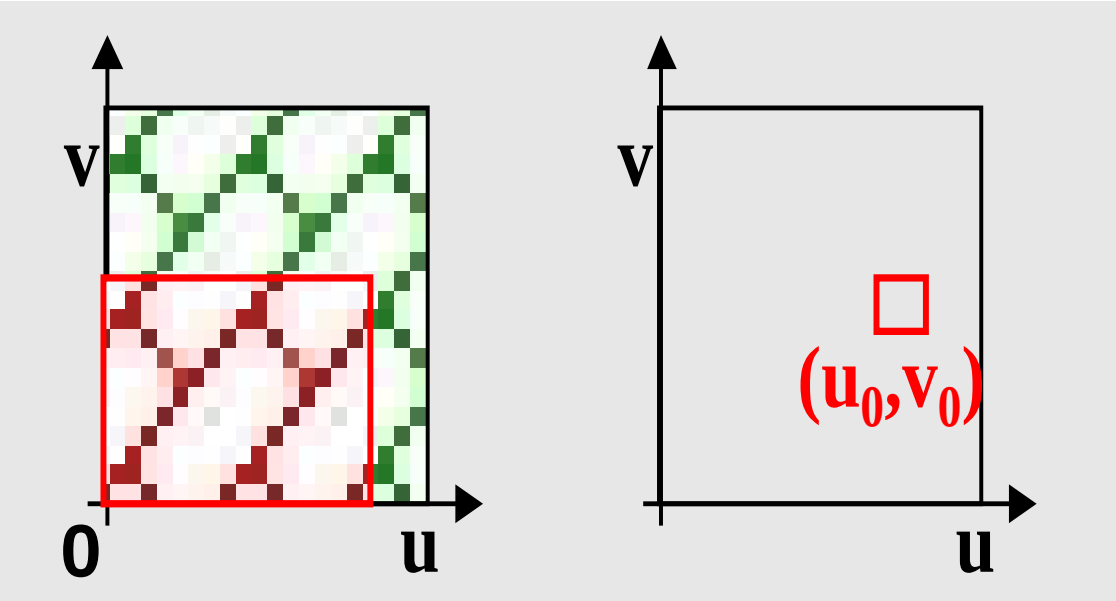
- Calculation of the weighted mean at runtime (called *"direct convolution"*)
  - Often used approximations



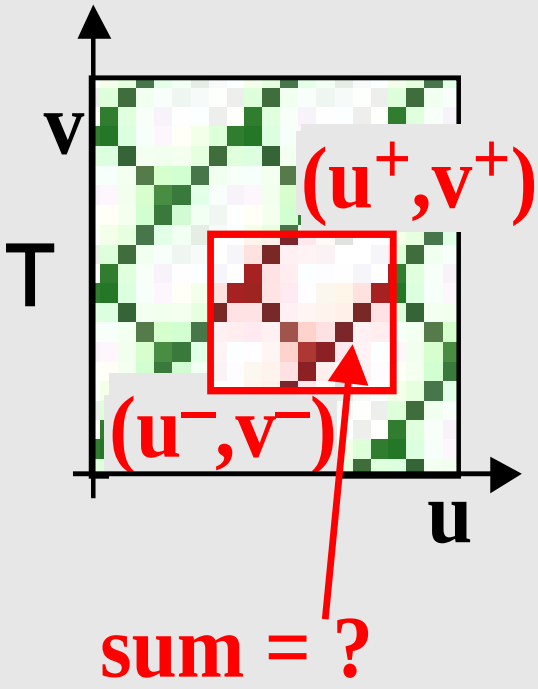
Anti-Aliasing

- Calculation of the weighted mean in a *preprocess* (called *"prefiltering"*)
  - *Mip-mapping* (+ optional *anisotropic filtering*)
- Summed area table S
  - Precalculation of rectangle sums for each pixel

$$S(u_0,v_0) = \sum_{\substack{u \le u_0 \\ v \le v_0}} T(u,v)$$

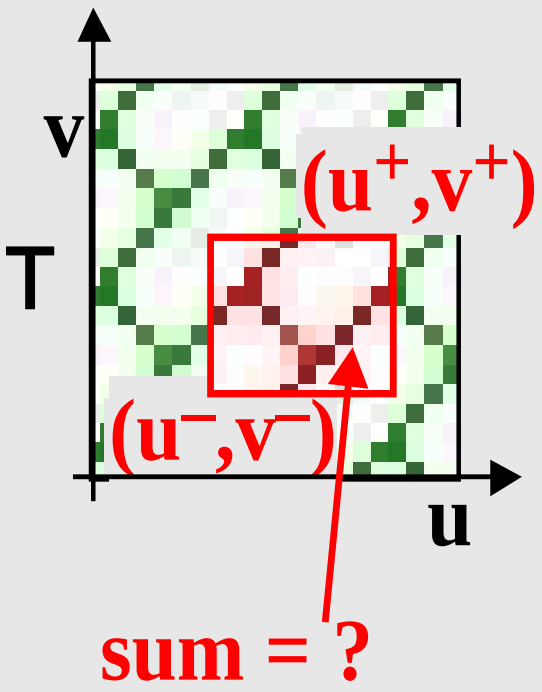


Anti-Aliasing: Summed Area Table

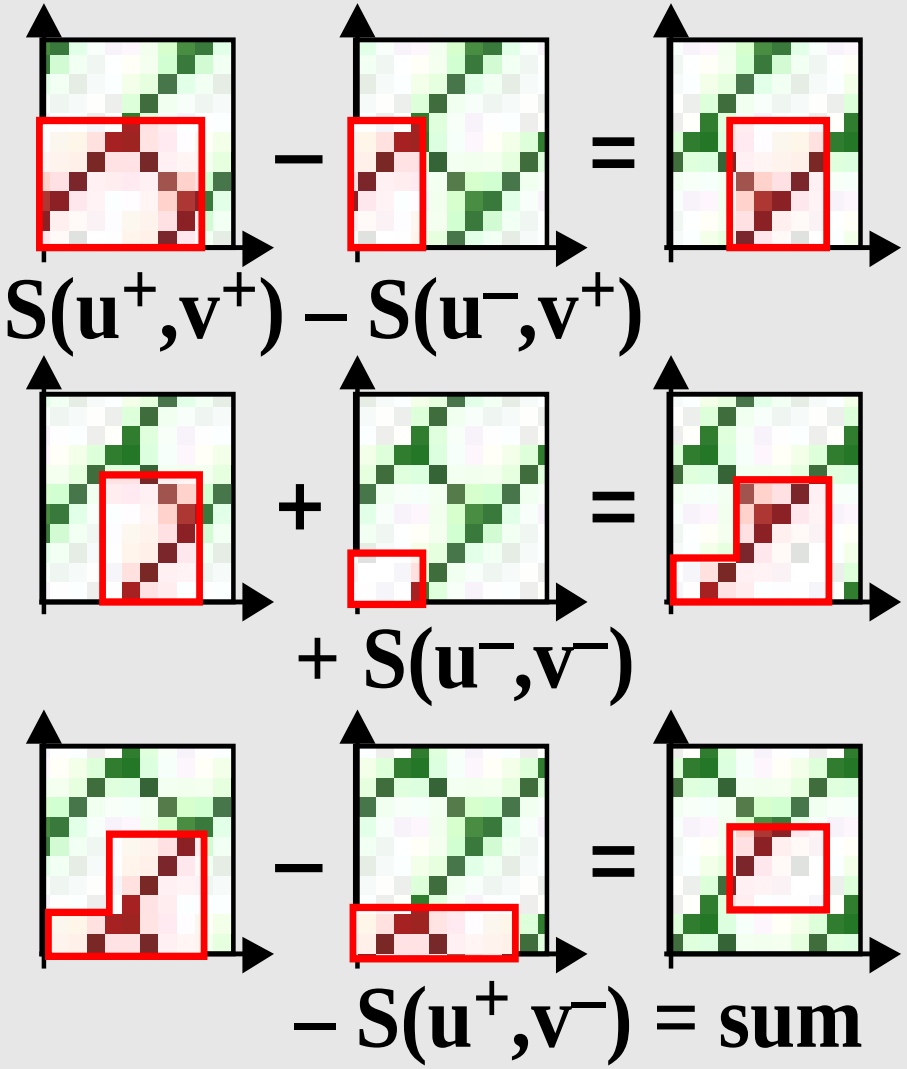


Effort/pixel is constant!

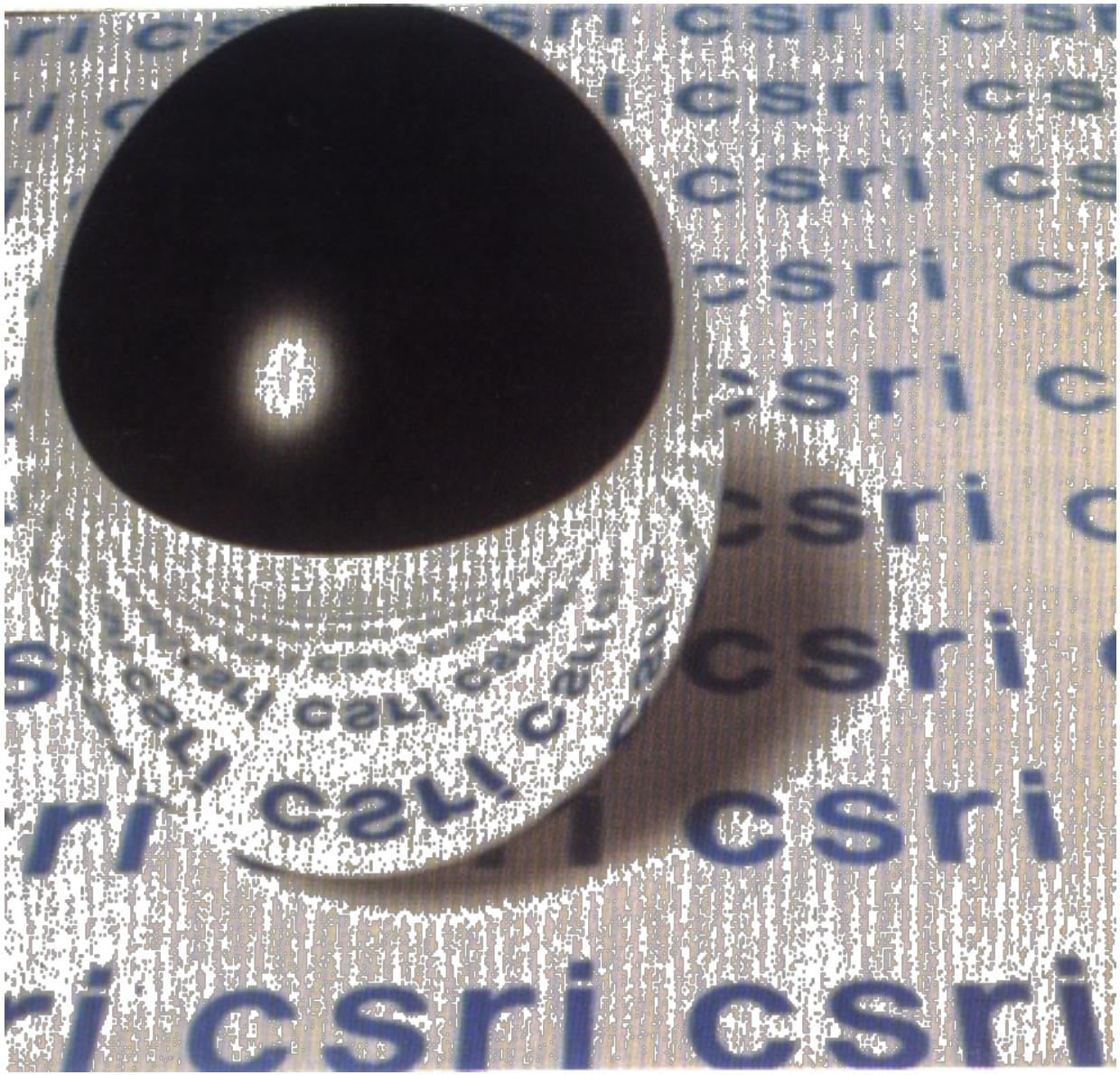
Anti-Aliasing: Summed Area Table



Effort/pixel is constant!

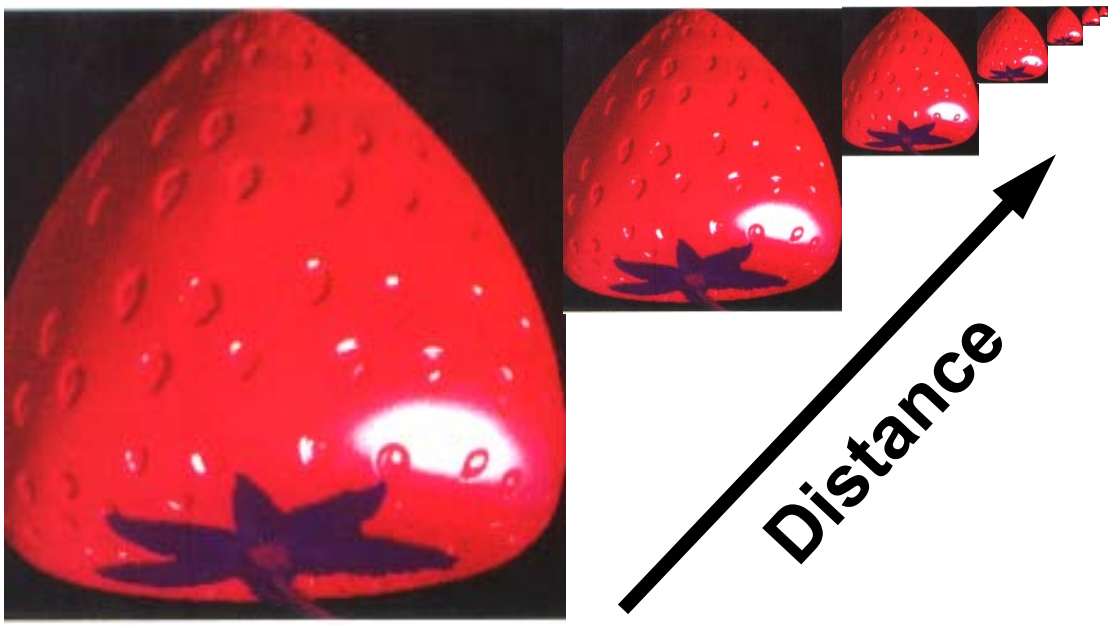


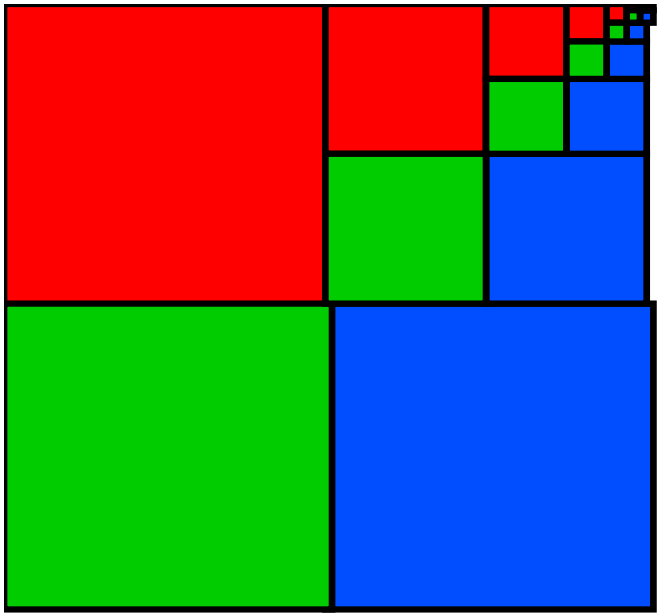
Anti-Aliasing: Example (Summed Area T.)



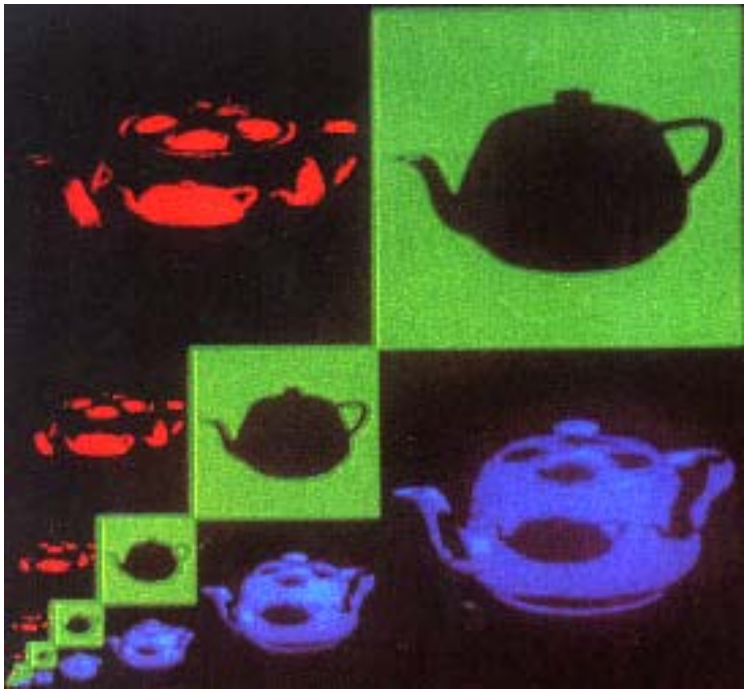
Anti-Aliasing: MIP Mapping

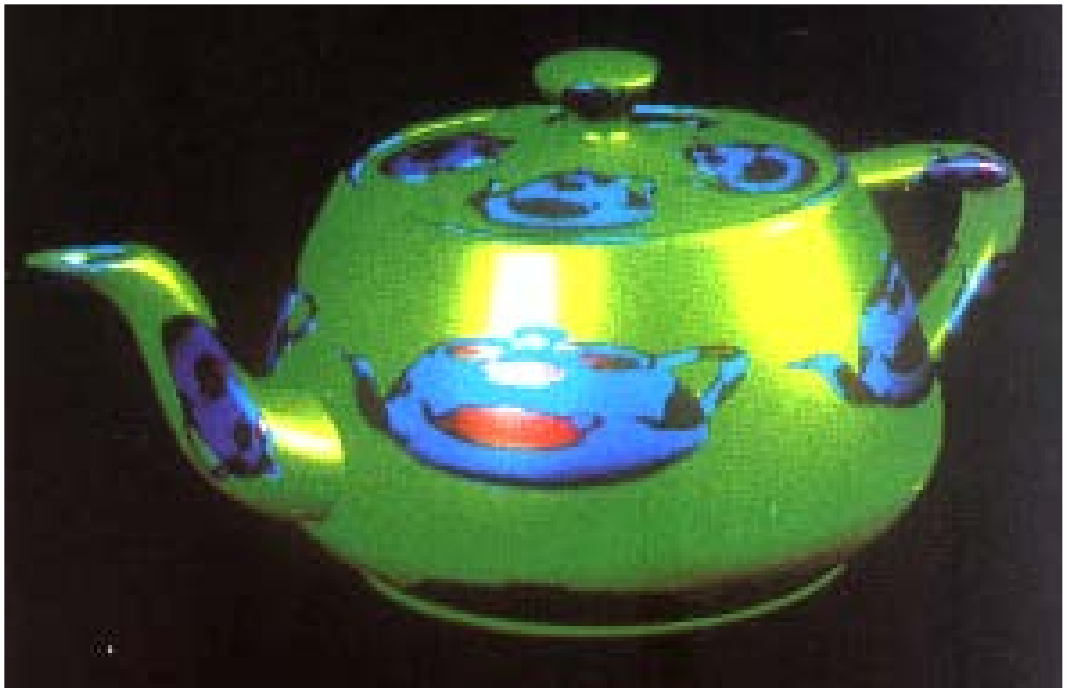
- MIP Mapping (“Multum In Parvo”)
  - Texture size is reduced by factors of 2 (*downsampling* = "much info on a small area")
  - Simple (4 pixel average) and memory efficient
  - Last image is only ONE texel





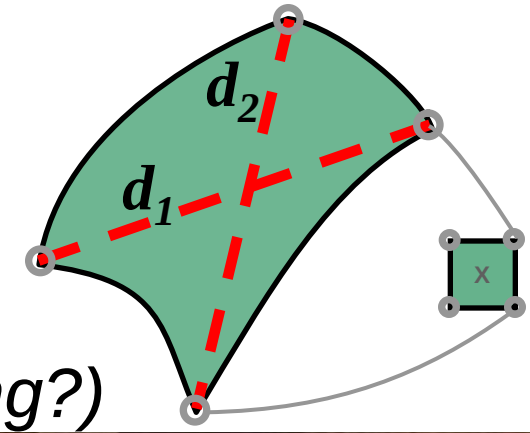
Anti-Aliasing: Example (Mip Mapping)

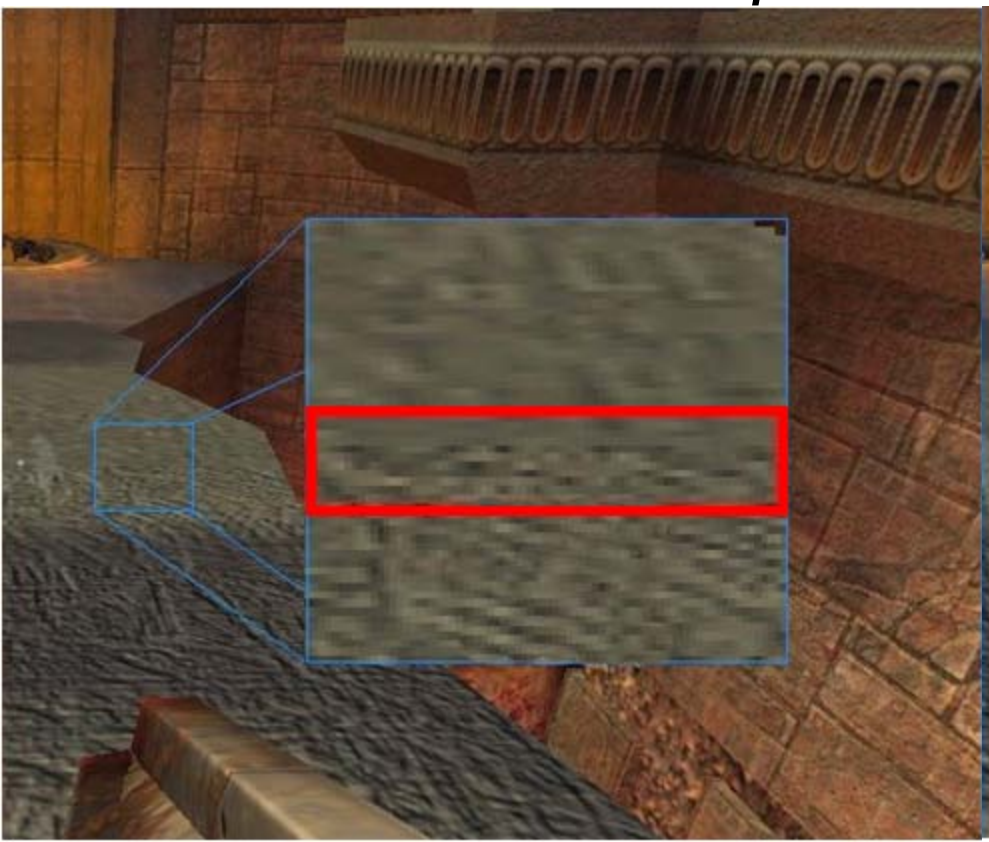


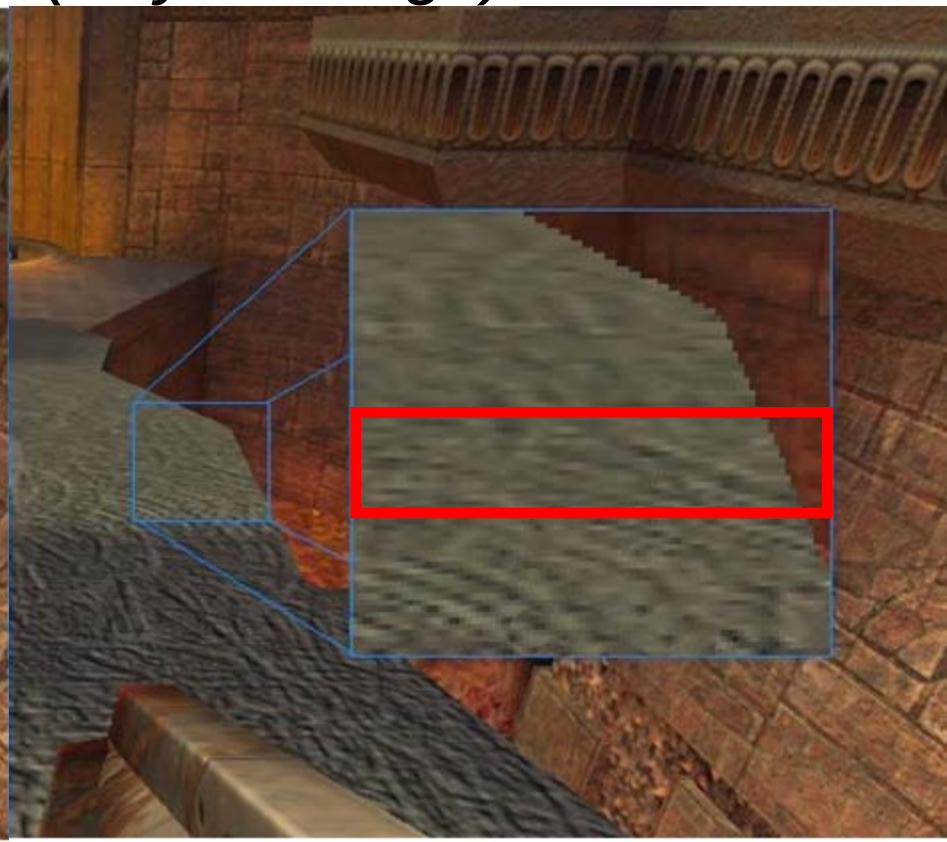


Anti-Aliasing: MIP Mapping

- 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 (why aliasing?)*

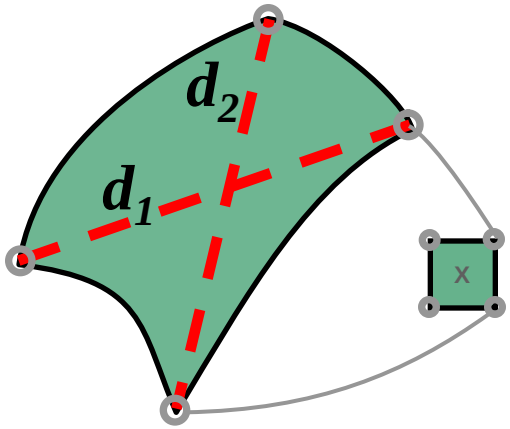


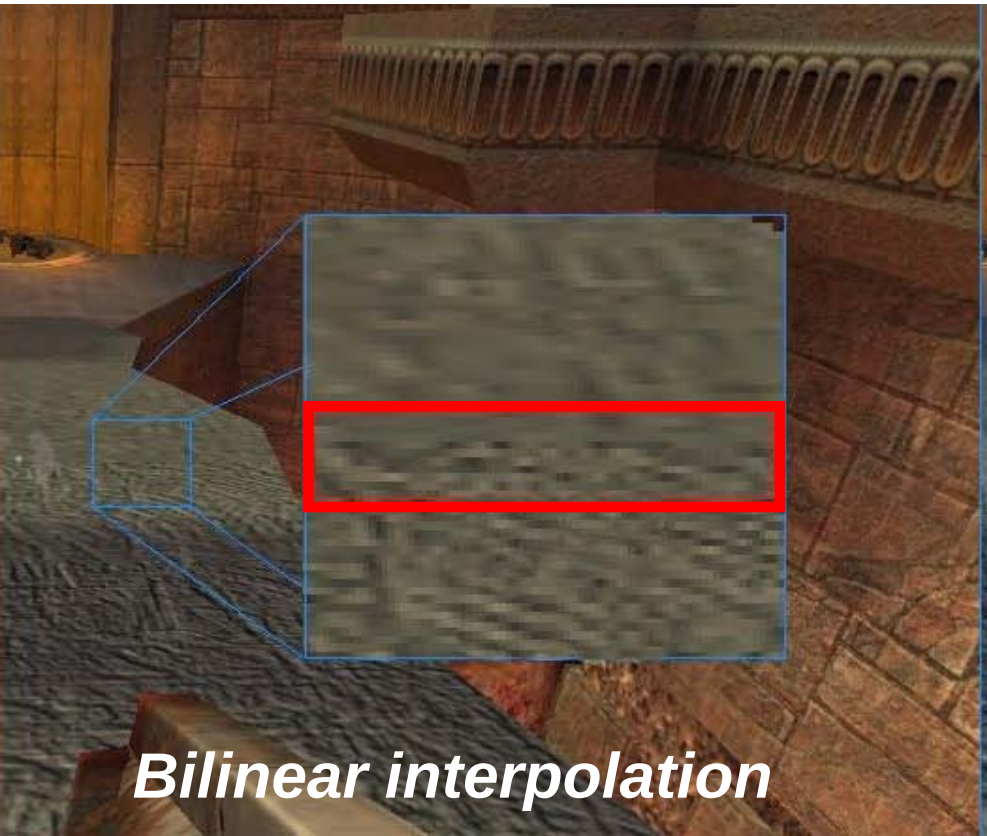


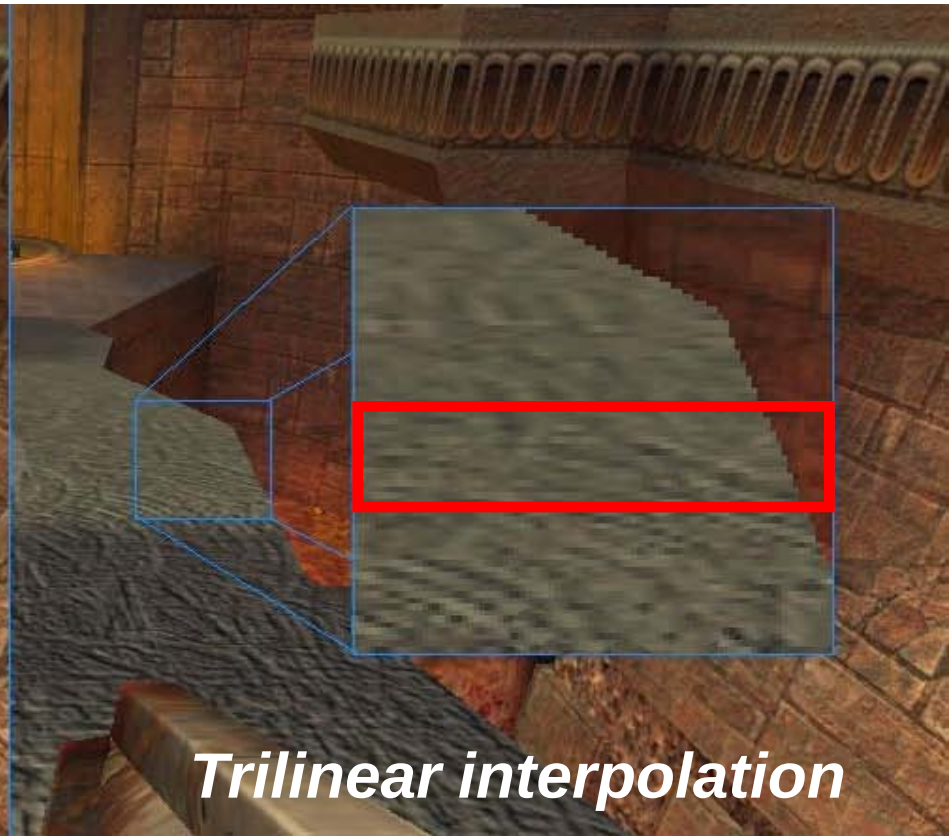


Anti-Aliasing: MIP Mapping

- 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*

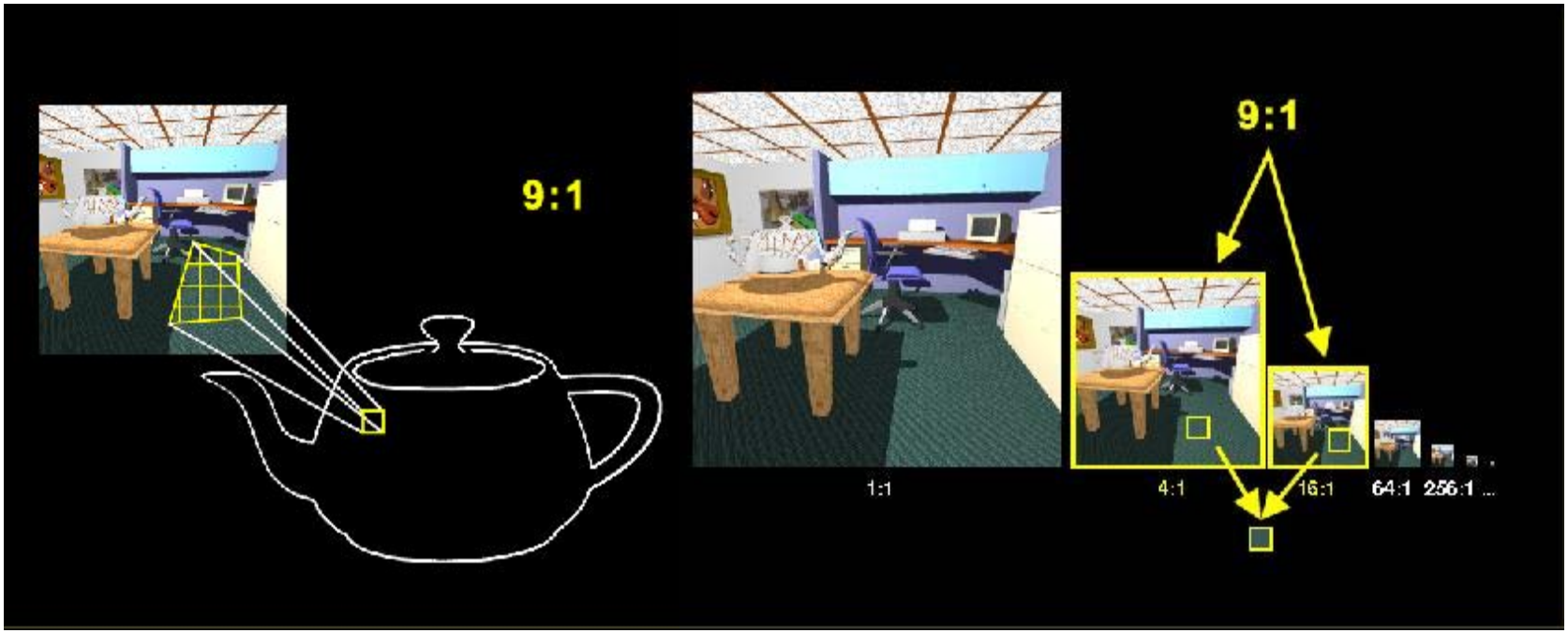






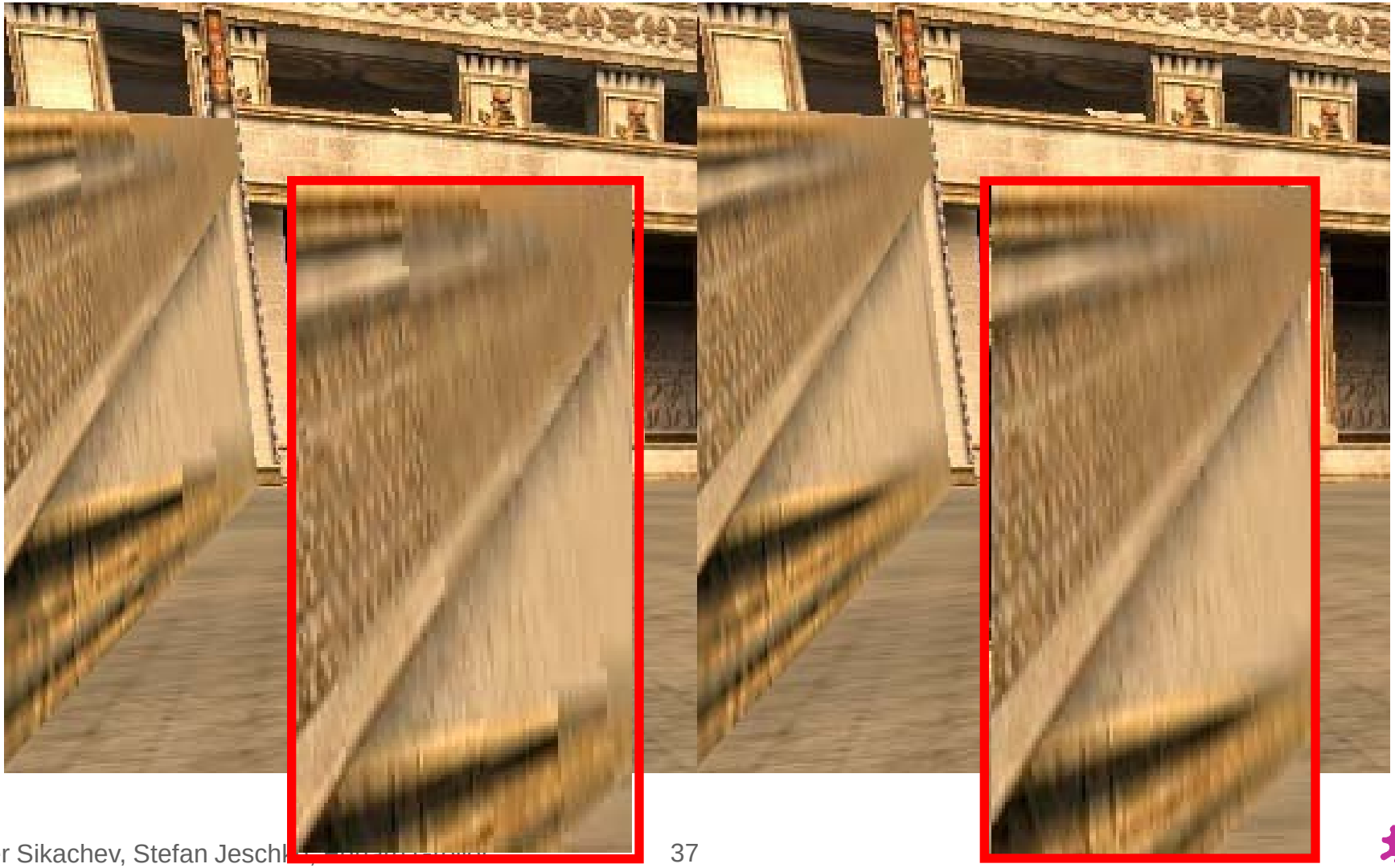
Anti-Aliasing: MIP Mapping

- 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)



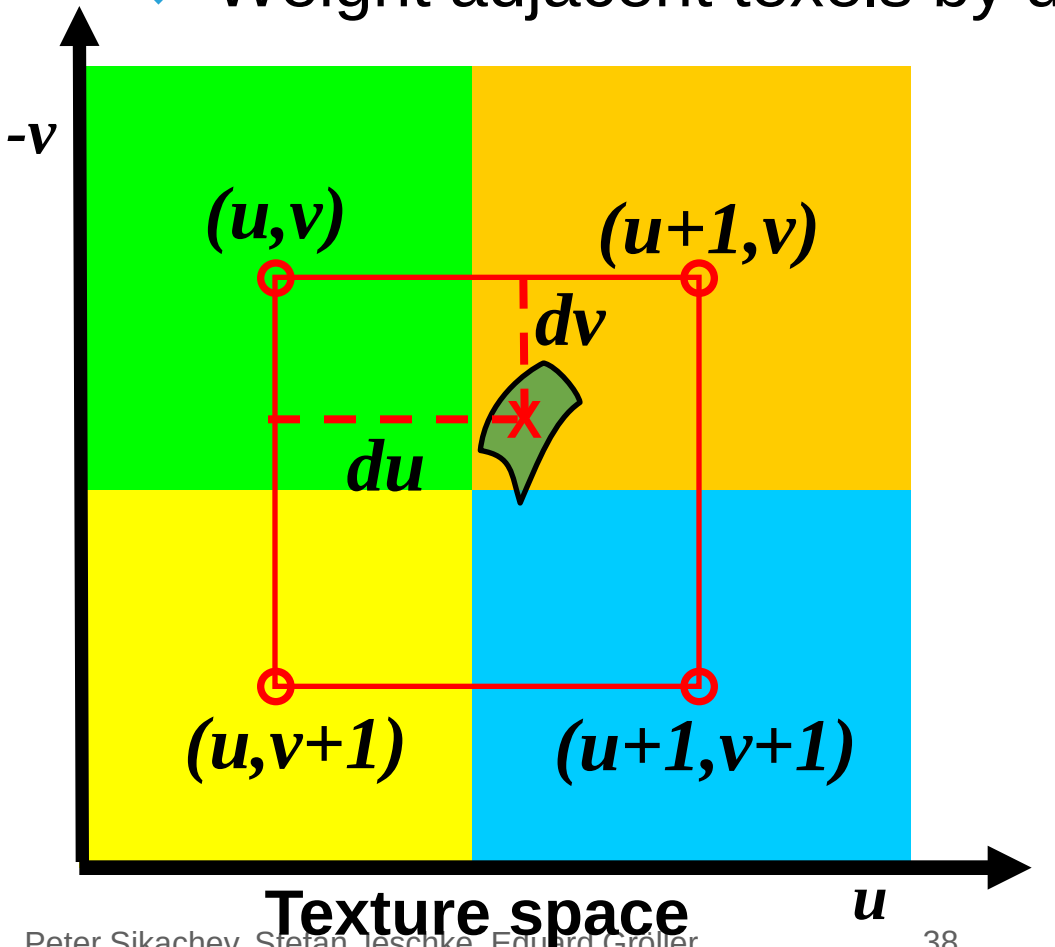
Anti-Aliasing: Mip Mapping (Example)

- Other example for bilinear vs. trilinear filtering



Anti-Aliasing

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



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

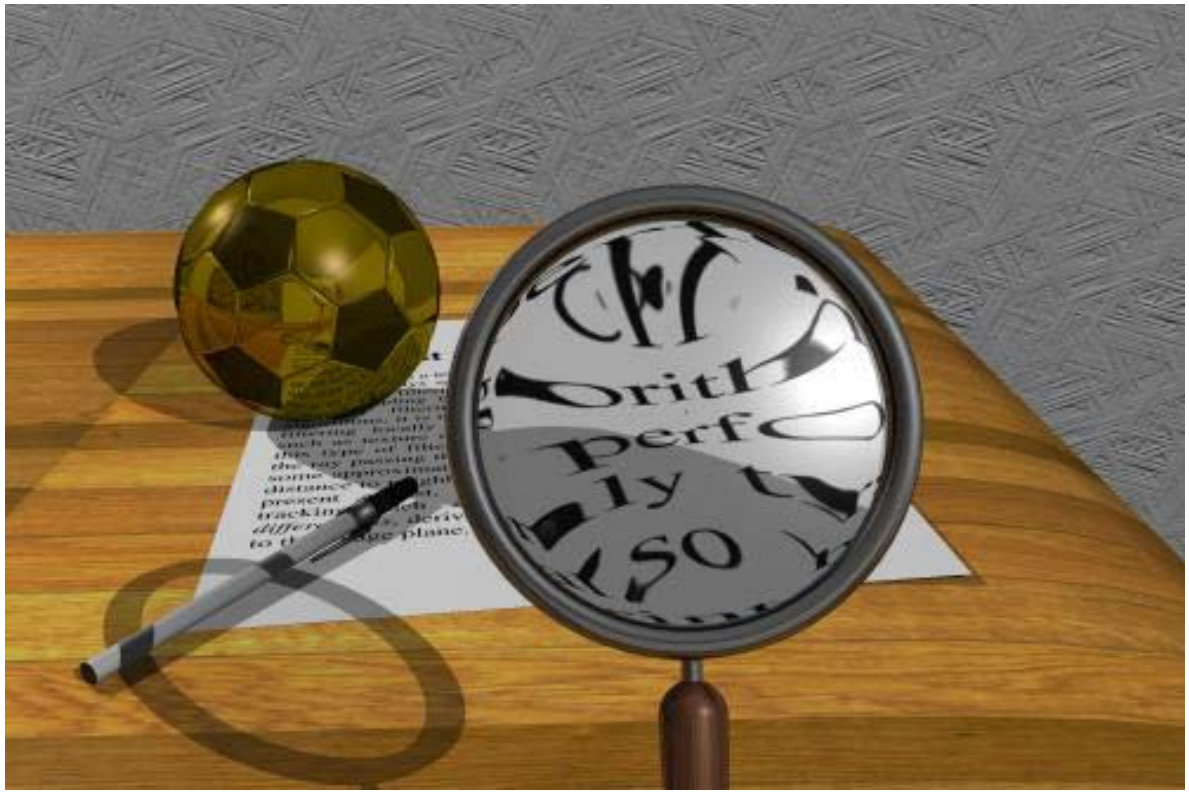
Anti-Aliasing (Bilinear Filtering Example)

Original image

Nearest neighbor

Bilinear filtering

Anti-Aliasing Examples (1/2)



No mip-mapping

Anti-Aliasing Examples (2/2)

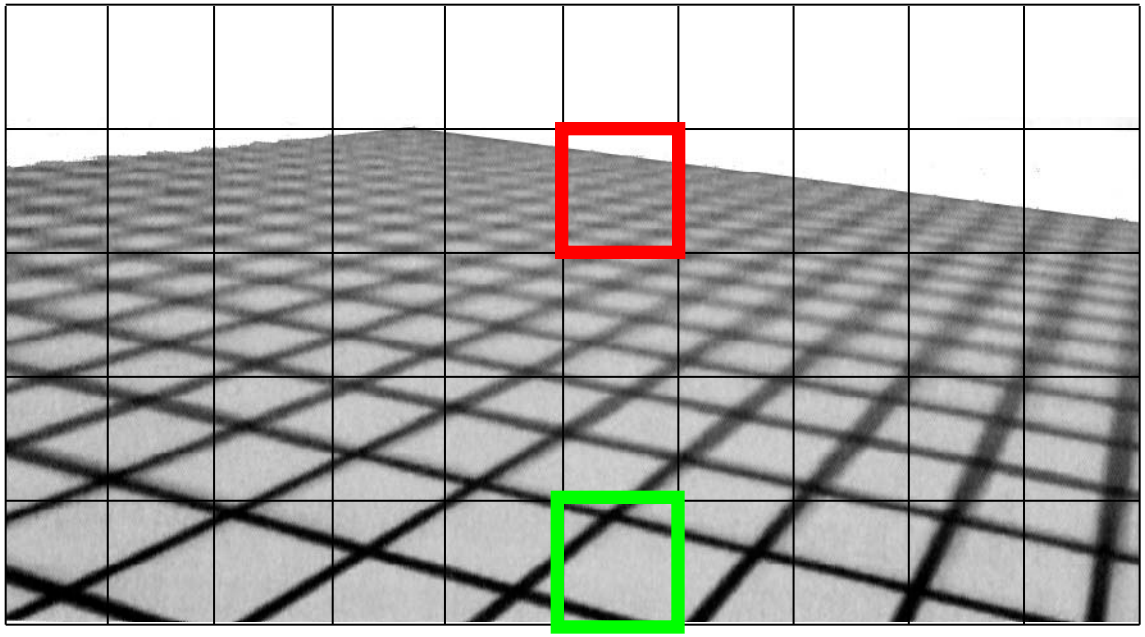
simple
mip-mapping

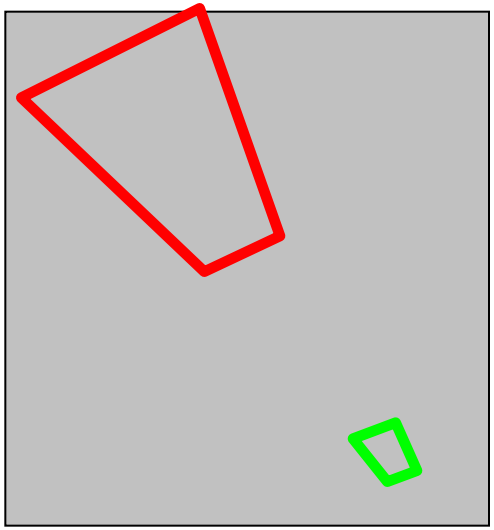
ray
differentials

+ anisotropic
filtering

Anti-Aliasing: Anisotropic Filtering

- Anisotropic Filtering
  - View dependent filter kernel
  - Implementation: *summed area table* (see above), "*RIP Mapping*", "*footprint assembly*"

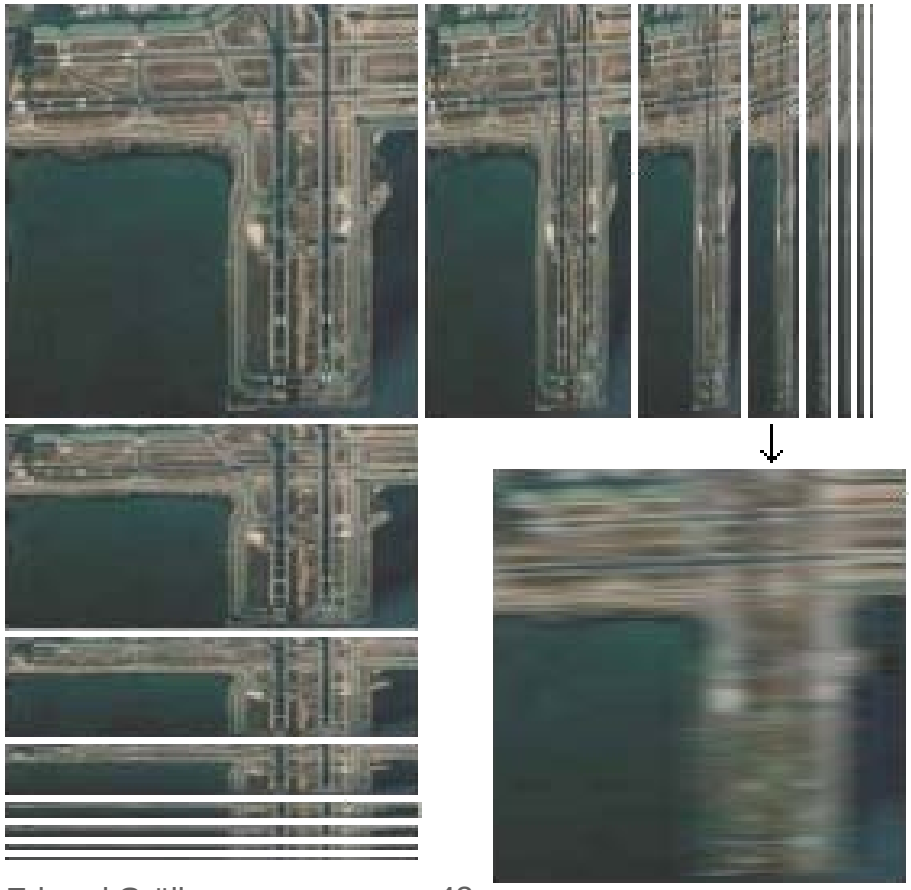




Texture space

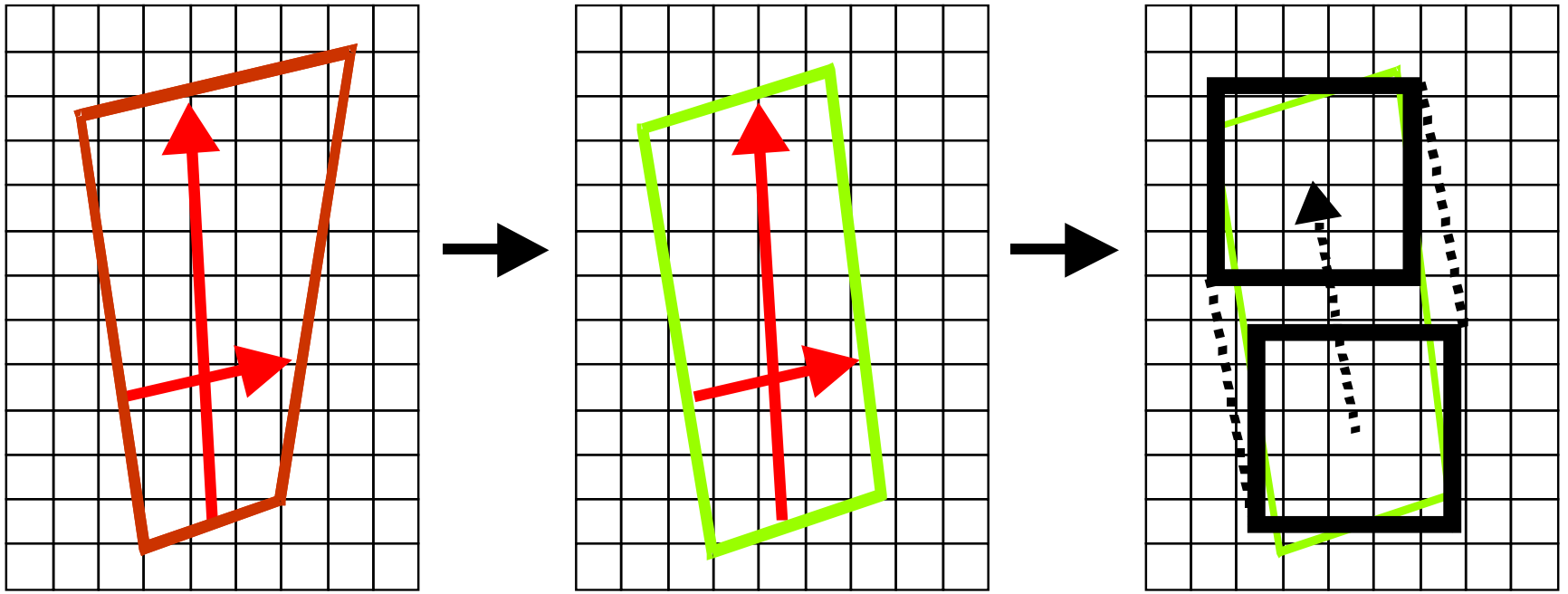
Anti-Aliasing: Anisotropic Filtering

- *"RIP Mapping"*: like MIP Maps but half resolution in x and y direction *separately*
  - Fast, but needs more memory than MIP Mapping



Anti-Aliasing: Anisotropic Filtering

- *"Footprint assembly"*
  - Approximate pixel footprint by parallelogram
  - Sample along the parallelogram using MIP mapping



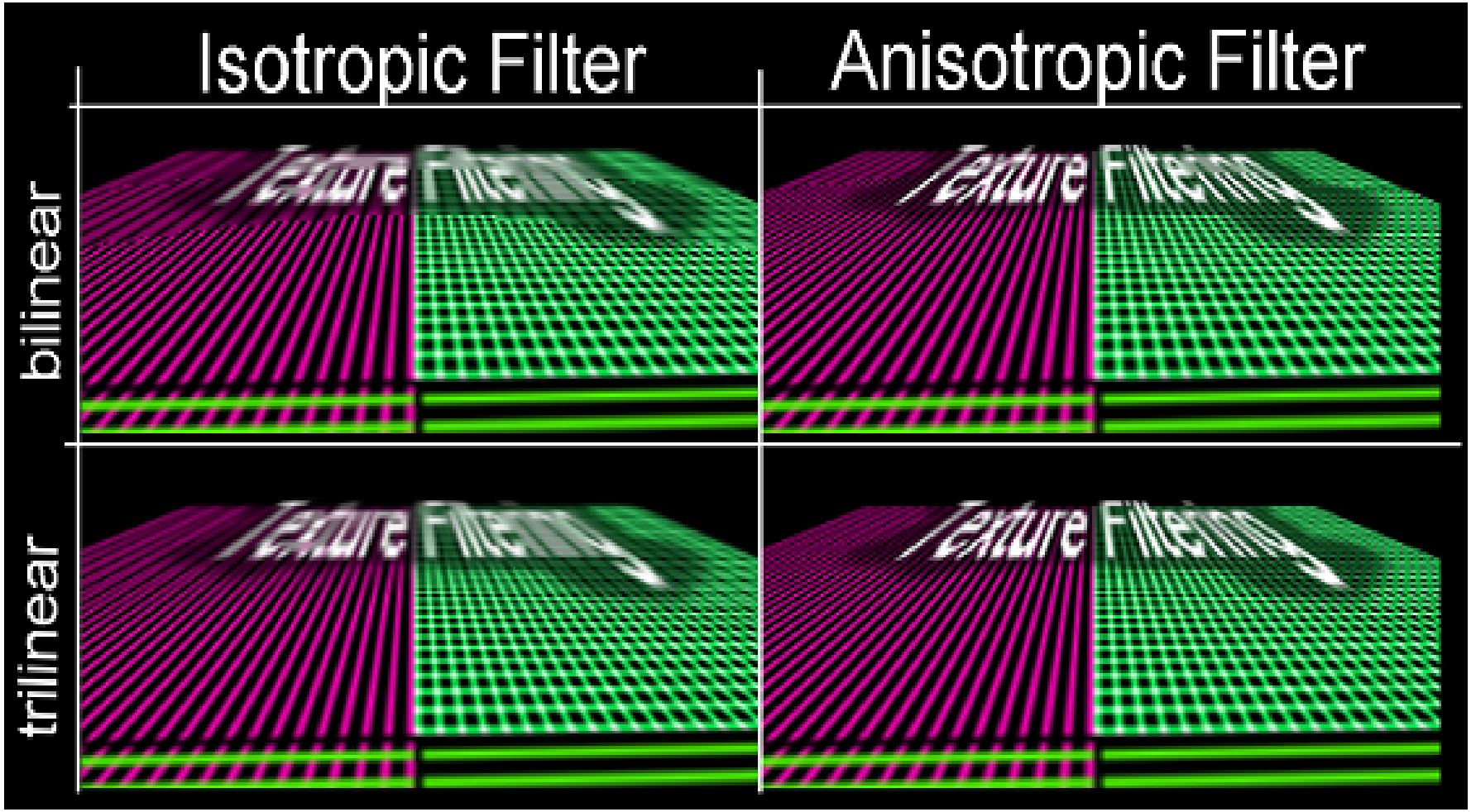
Footprint in texture space

Approximation by parallelogram

Sampling using Mip Mapping

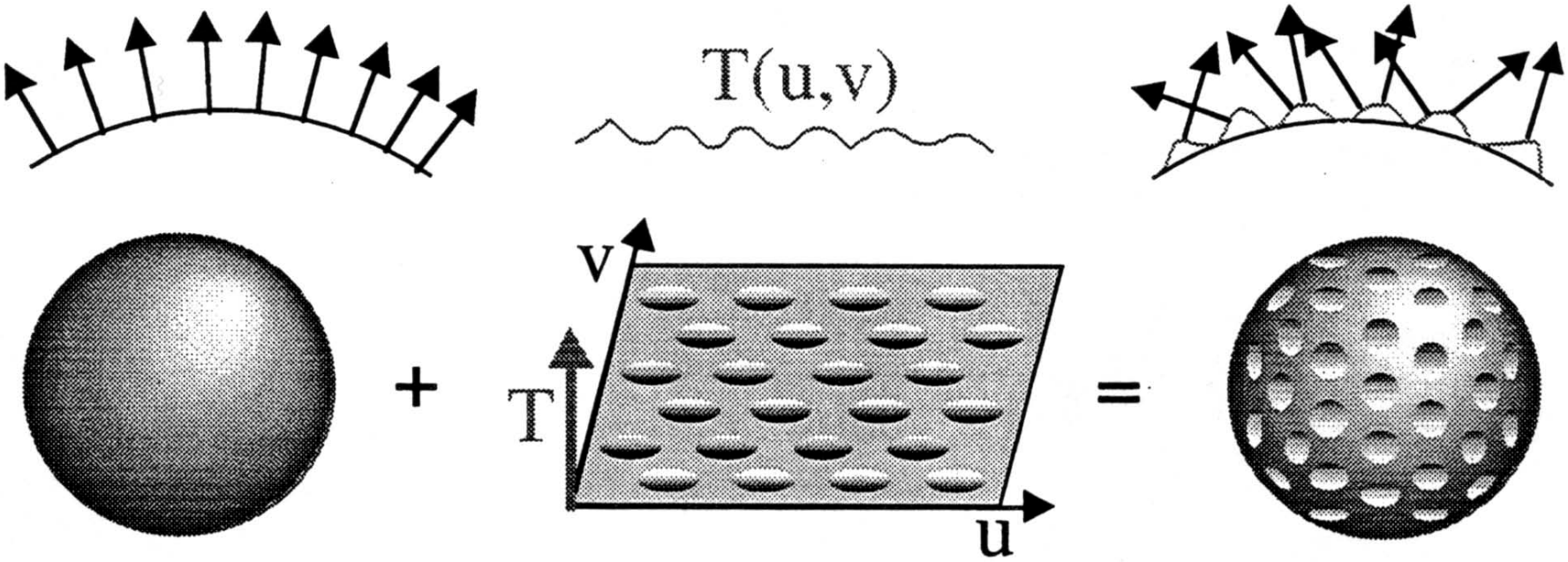
Anti-Aliasing: Anisotropic Filtering

- Example

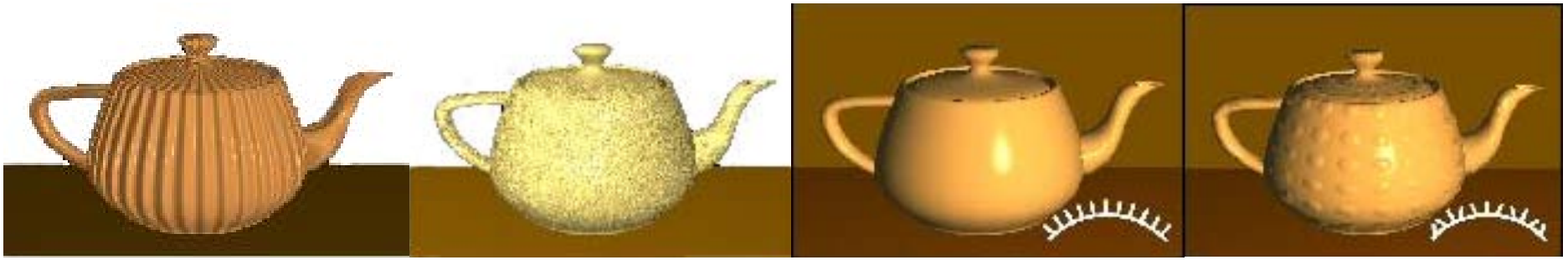


Bump Mapping

- Preprocess: compute normal vectors from height field
- Runtime: use computed normals for illumination

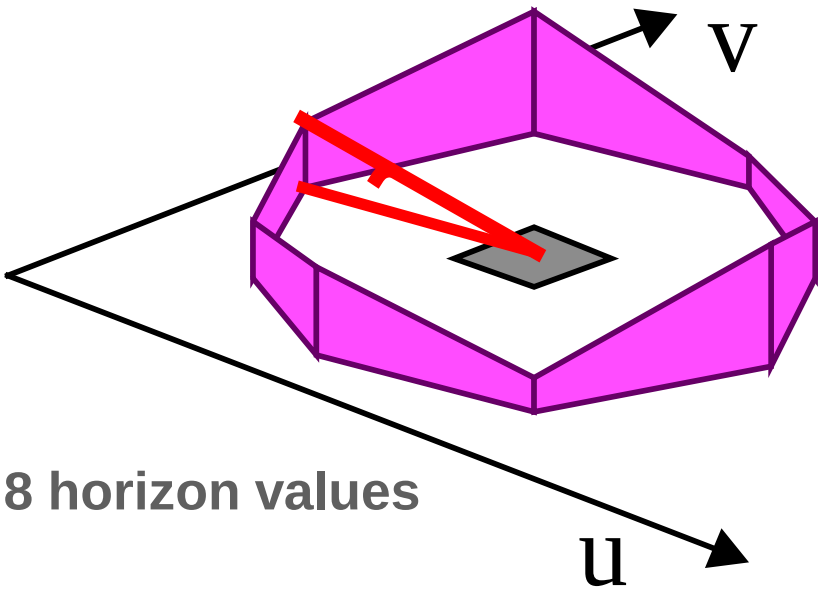


Bump Mapping Examples



Horizon Mapping

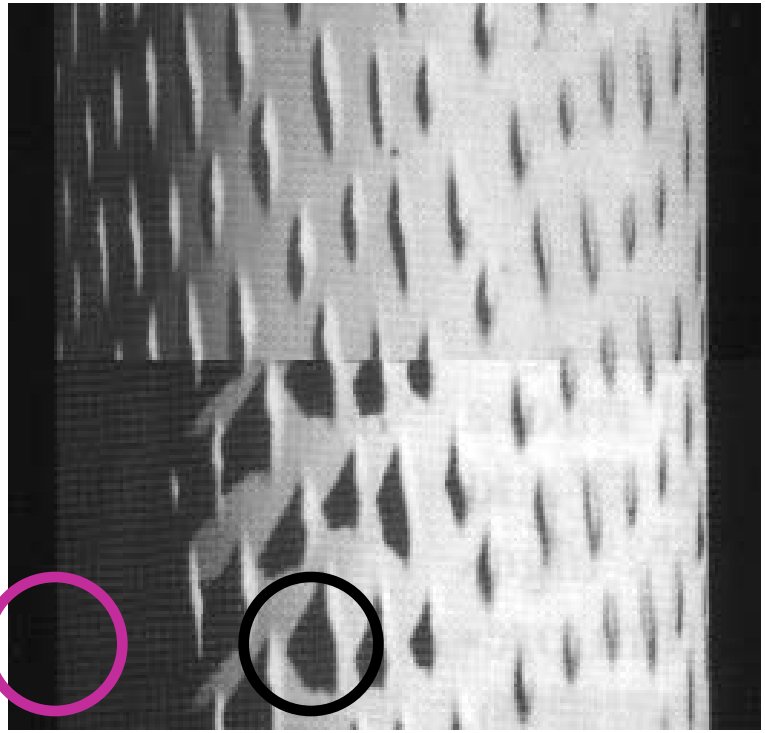
- Improve bump mapping with (local) shadows
- Preprocess: compute n horizon values per texel
- Runtime:
  - Interpolate horizon values
  - Shadow accordingly



Horizon Mapping Examples (1/2)

without

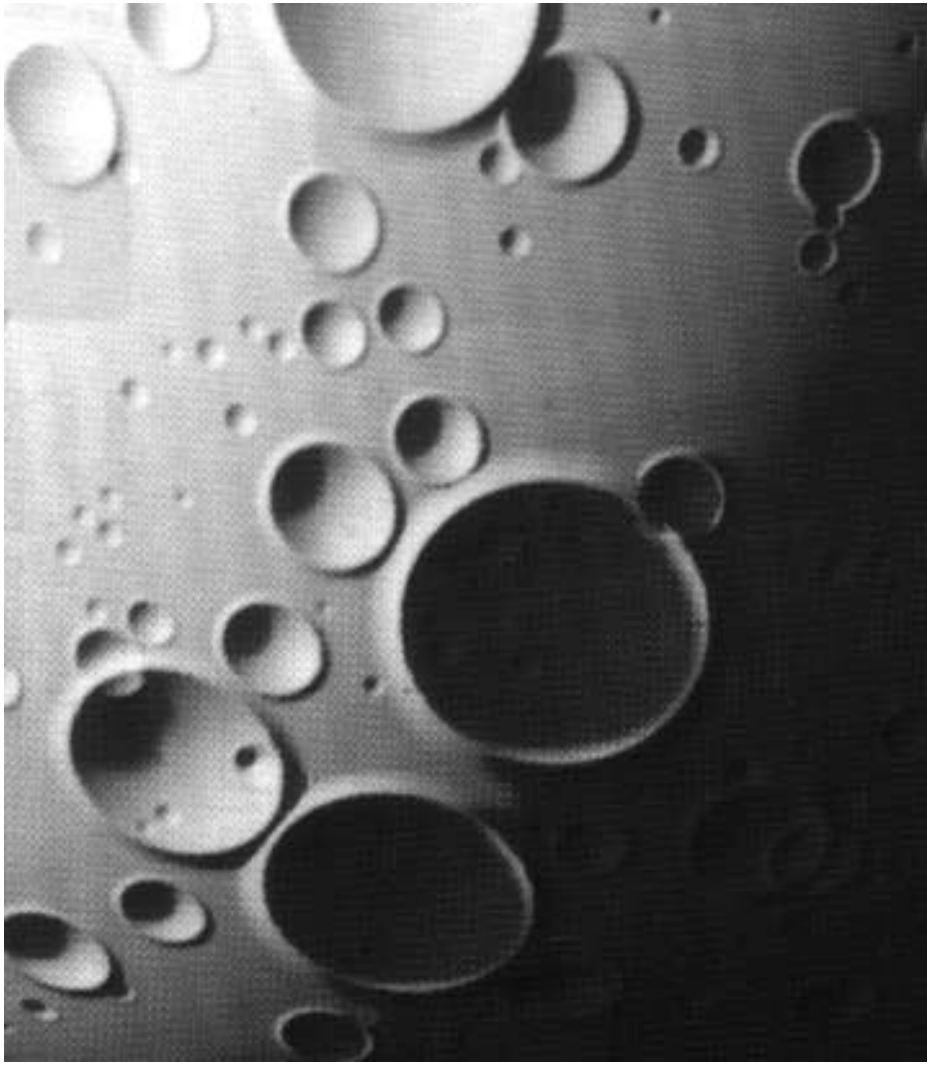
with



Shadows from bumps

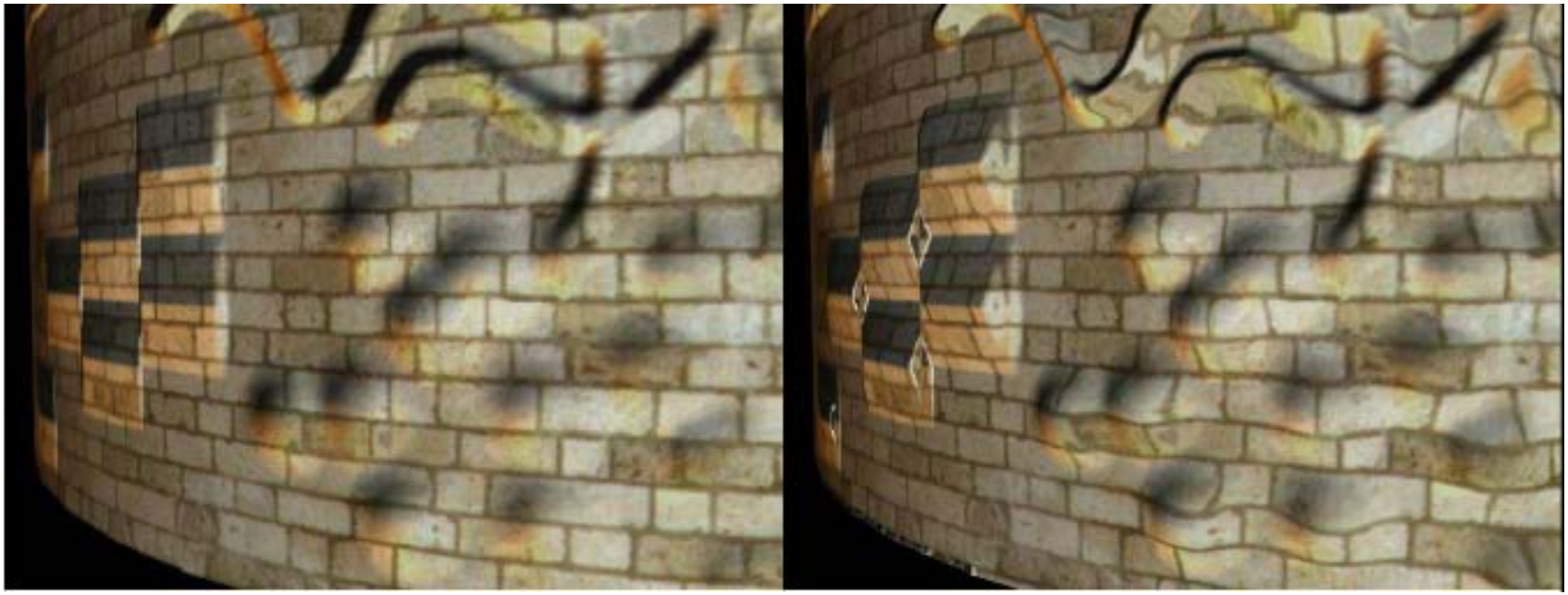
No light on rear side of the object

Horizon Mapping Examples (2/2)



Parallax Mapping

- Parallax: apparent movement of close objects in front of further objects
  - Preprocess: calculate *"texture coordinate shifts"*
  - Runtime: “crd. shifts” alter the texture lookup position
    - Problem: occlusions cannot be modelled



Bump mapping **Parallax + Bump mapping**

Parallax Mapping

- Trace the ray locally until it crosses the surface

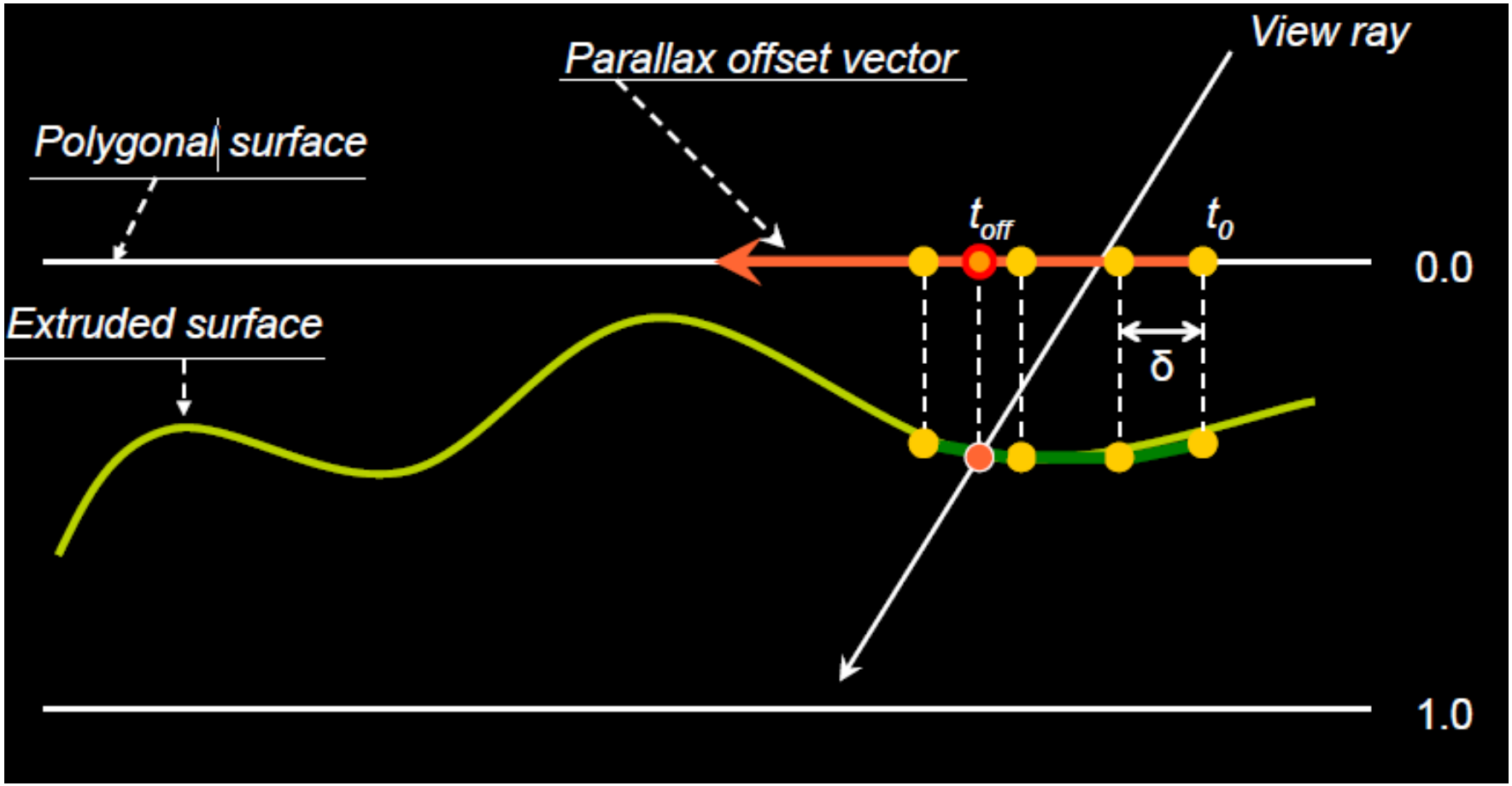


image courtesy of Natalya Tatarchuk

Parallax Mapping: Accuracy Issues

- Banding artifacts at acute angles
- Why do they occur?

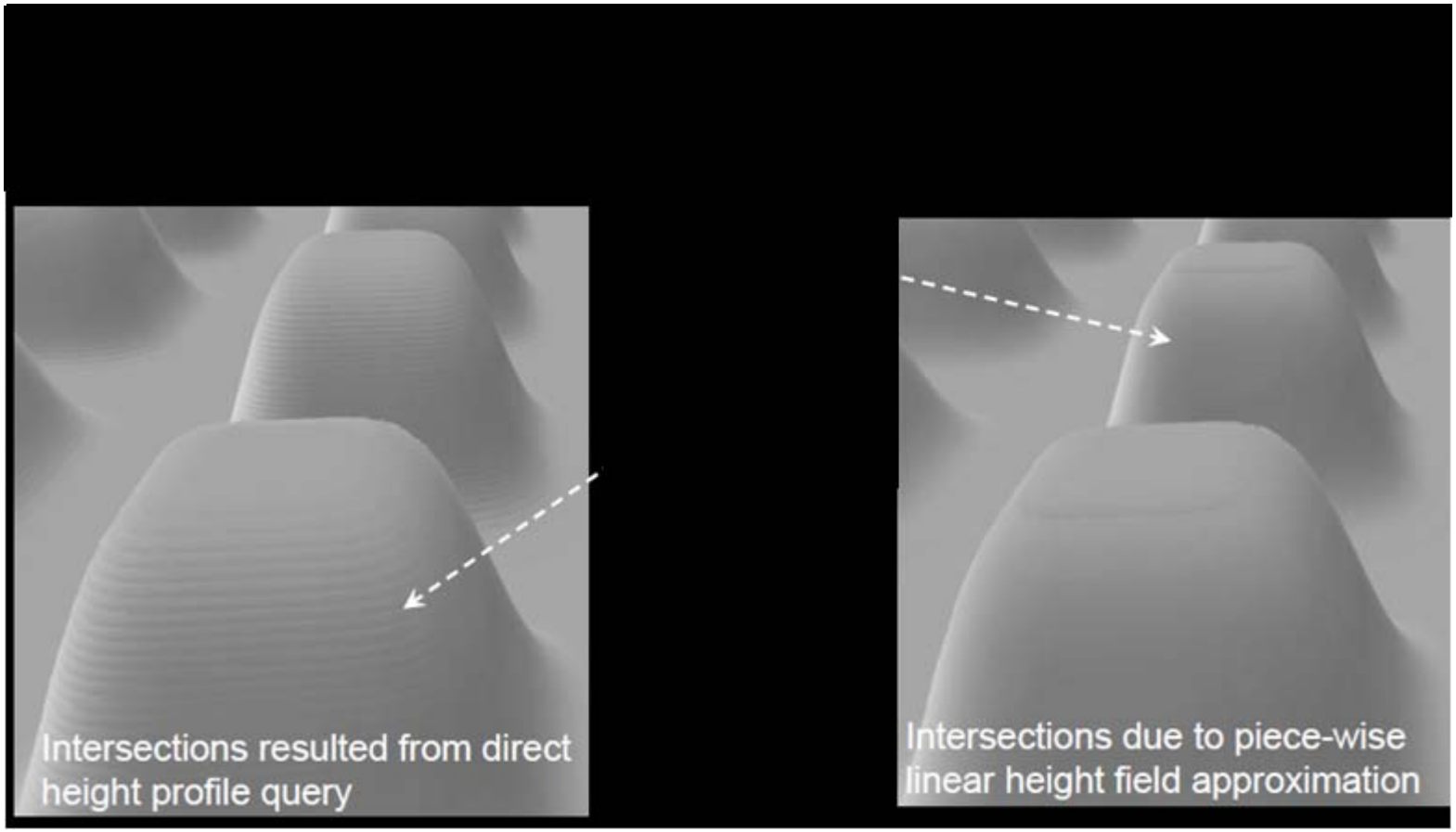


image courtesy of Natalya Tatarchuk

Parallax Mapping: Accuracy Issues

- Banding artifacts at acute angles
- Solution: approximate intersection with point B

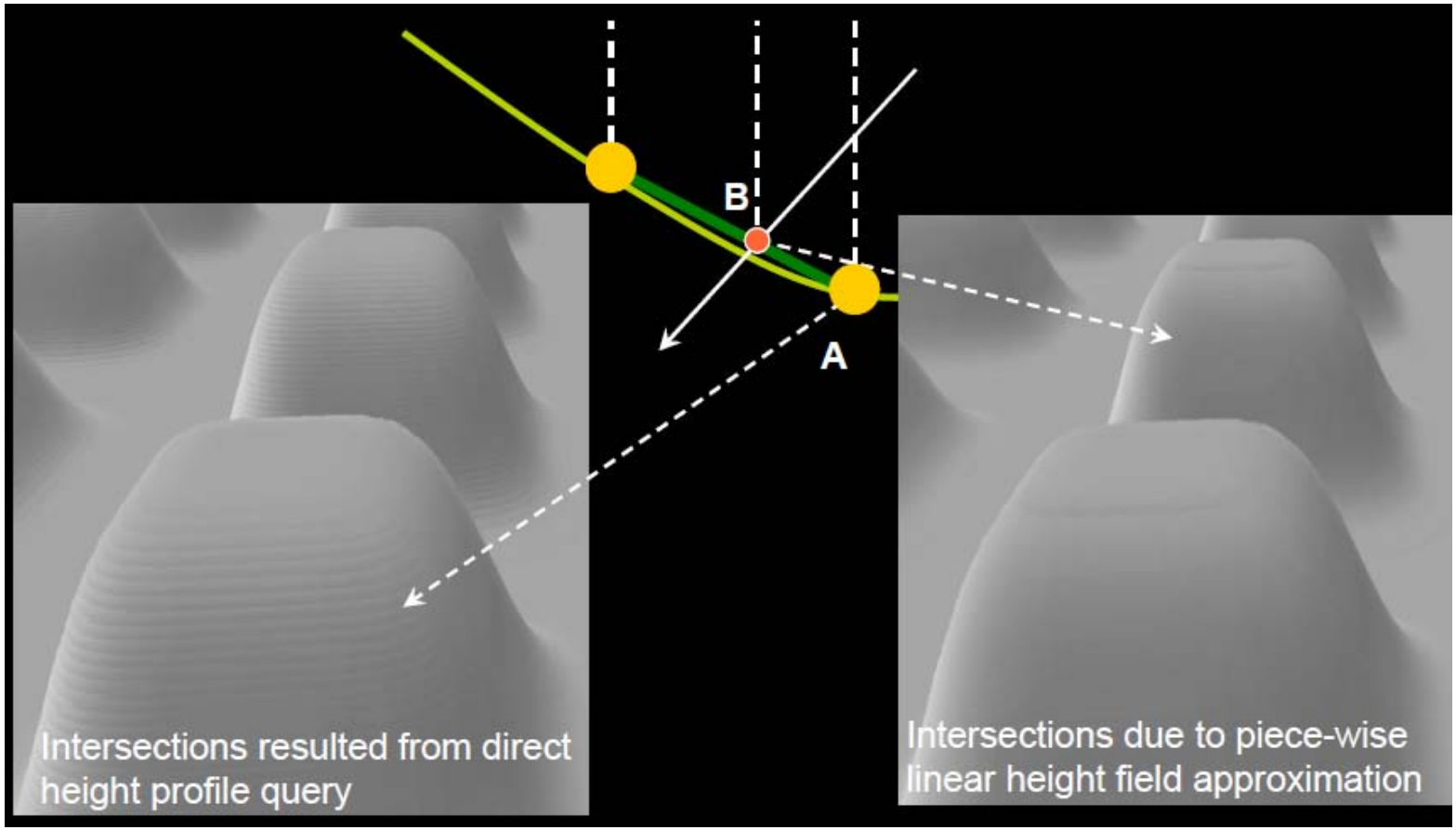
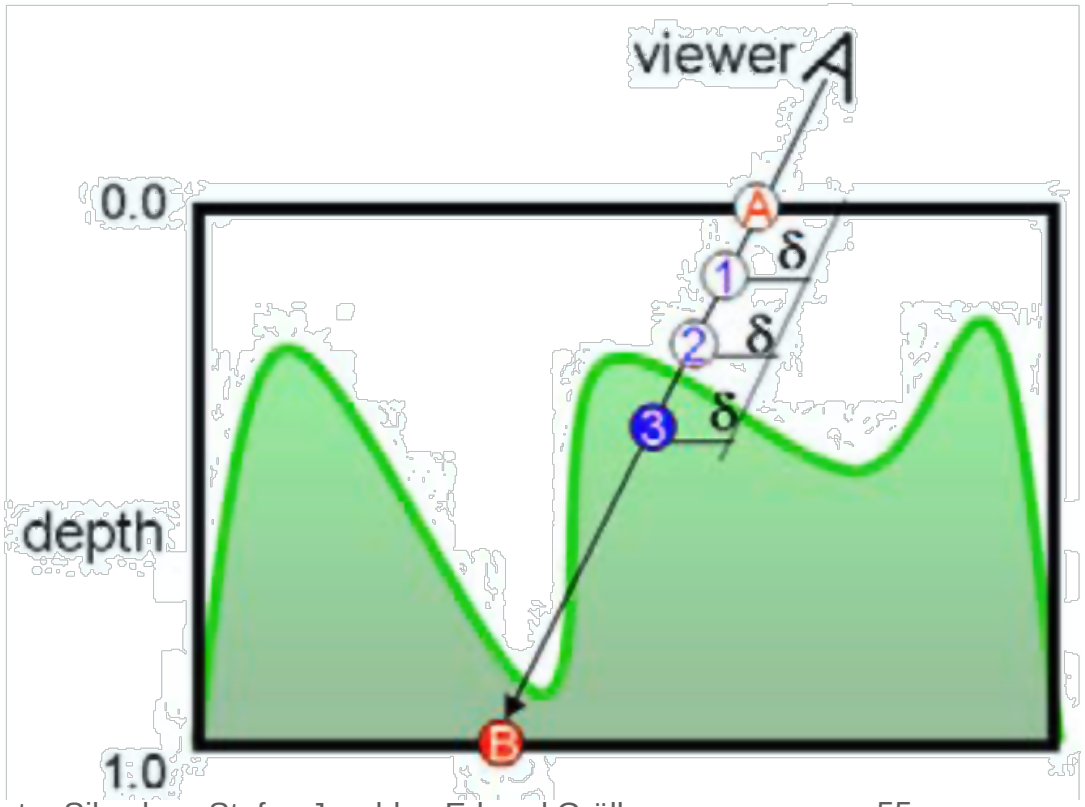


image courtesy of Natalya Tatarchuk

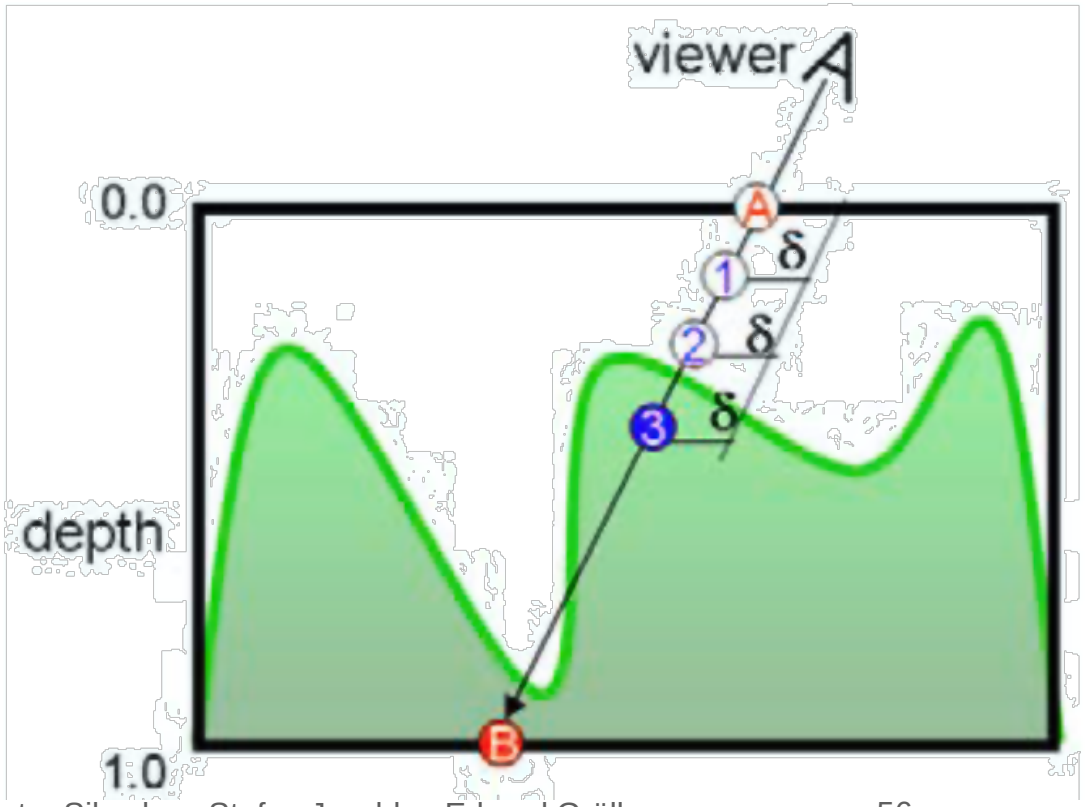
Relief Mapping

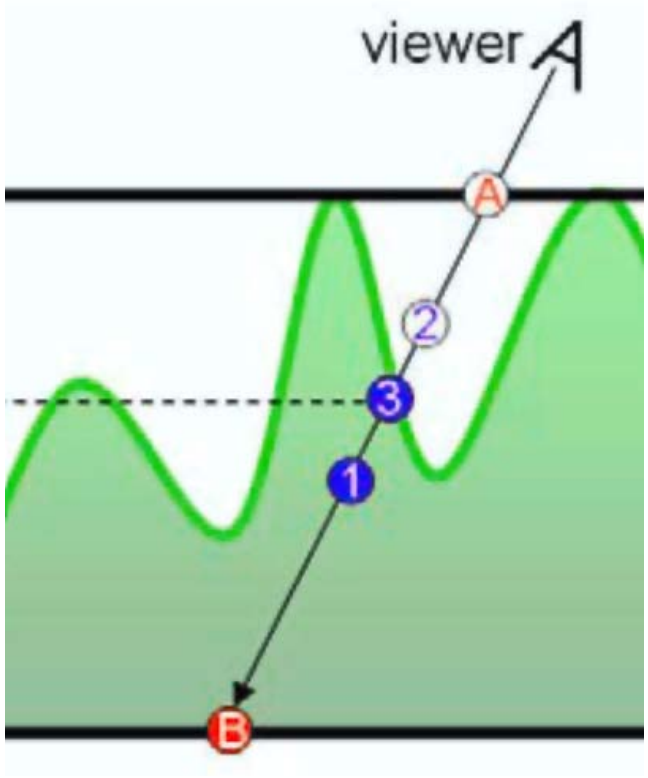
- At runtime: perform ray casting in the pixel shader
  - Calculate entry (A) and exit point (B)
  - March along ray until intersection with height field is found
  - How to accelerate?



Relief Mapping

- At runtime: perform ray casting in the pixel shader
  - Calculate entry (A) and exit point (B)
  - March along ray until intersection with height field is found
  - Binary search to refine the intersection position





Relief Mapping Examples (1)

Bump mapping

Parallax mapping

Relief mapping

Relief Mapping Examples (2)