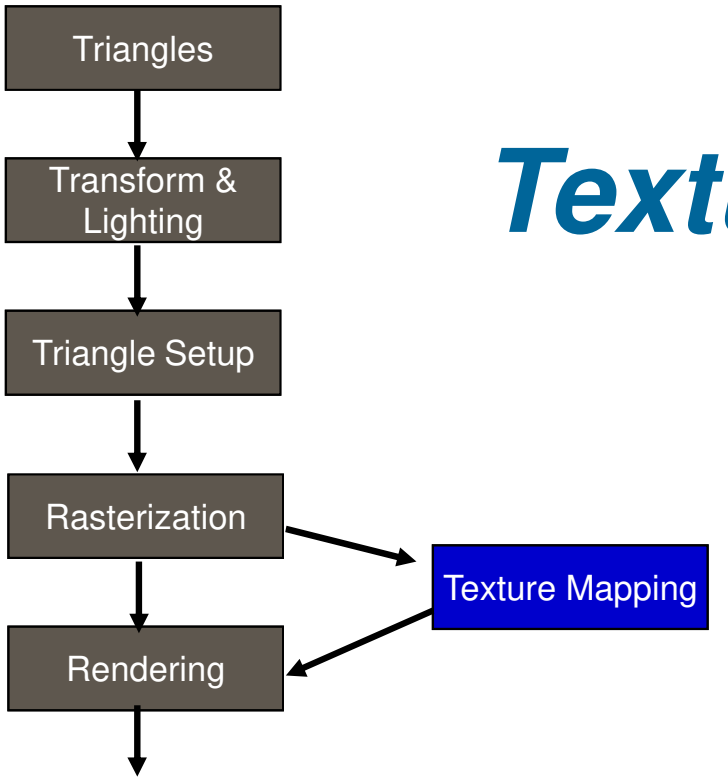
# Computer Graphics

*by Ruen-Rone Lee*
*ICL/ITRI*

# *Wrap up from last course*

- **Hidden surface removal**
  - **Backface culling**
  - **Z-buffer algorithm**
  - **BSP tree**
  - **Portal culling**

# Conventional 3D Graphics Pipeline




Conventional 3D Graphics Pipeline

## Texture Mapping (A)

***Concept of Texture Mapping***

***Texture Filtering***

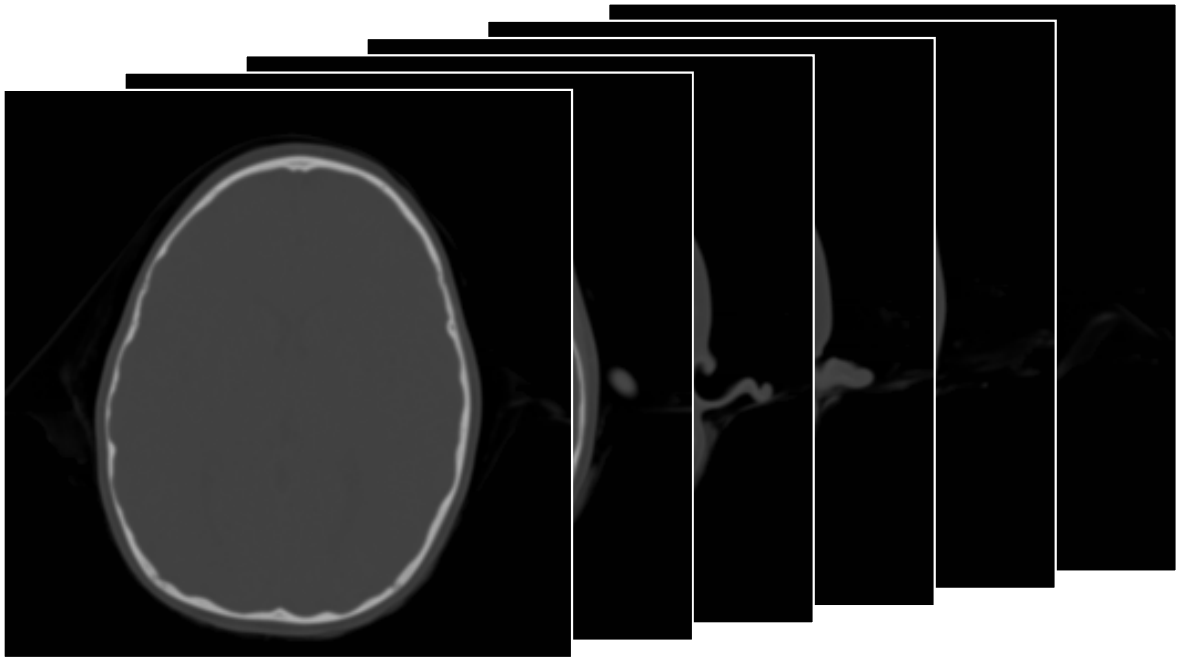
***Texture Mapping Applications***

# 2D Textures

- A 2D texture is a two-dimensional image
- Each texture element is called a texel

# 3D Textures

- A 3D texture is a set of 2D textures that constitute a three-dimensional volume texture
- Each texture element is called a voxel

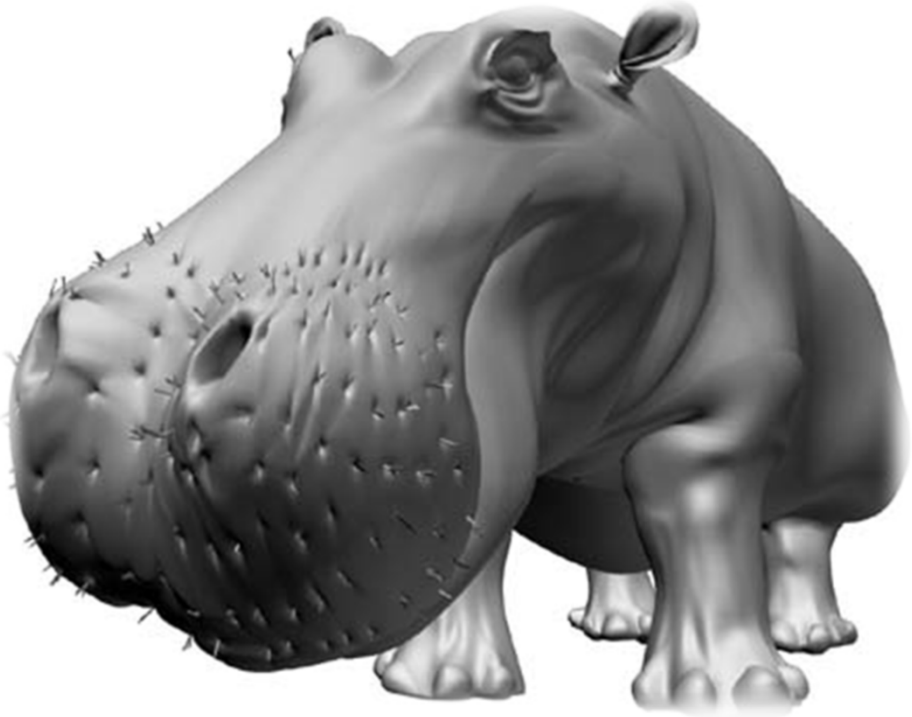


# Texture Mapping

Primitives + Texture = Result

# Example of Texture Mapping

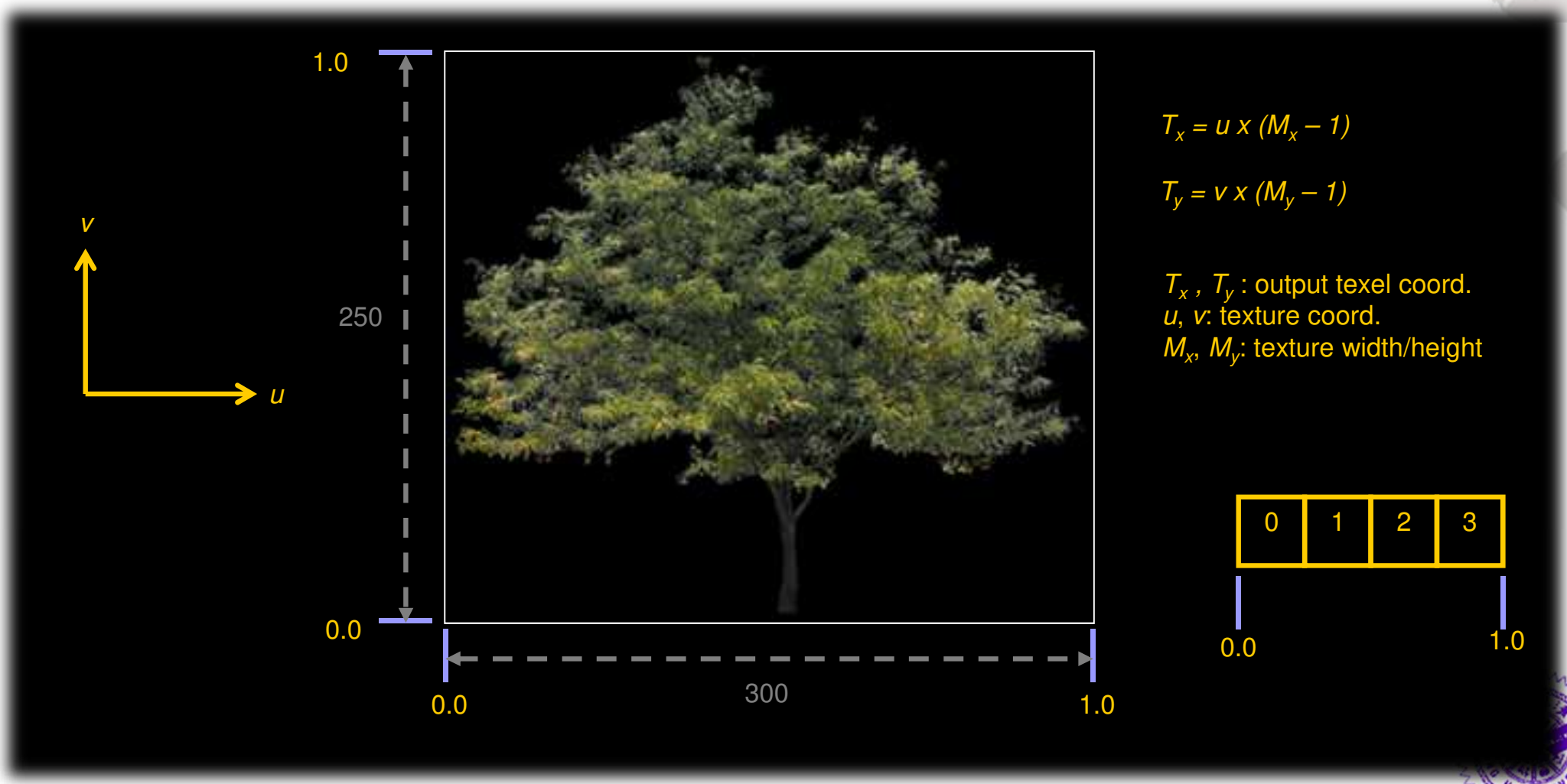
- **Improve realism: not just texture but also lighting**



Images from 3dRender.com

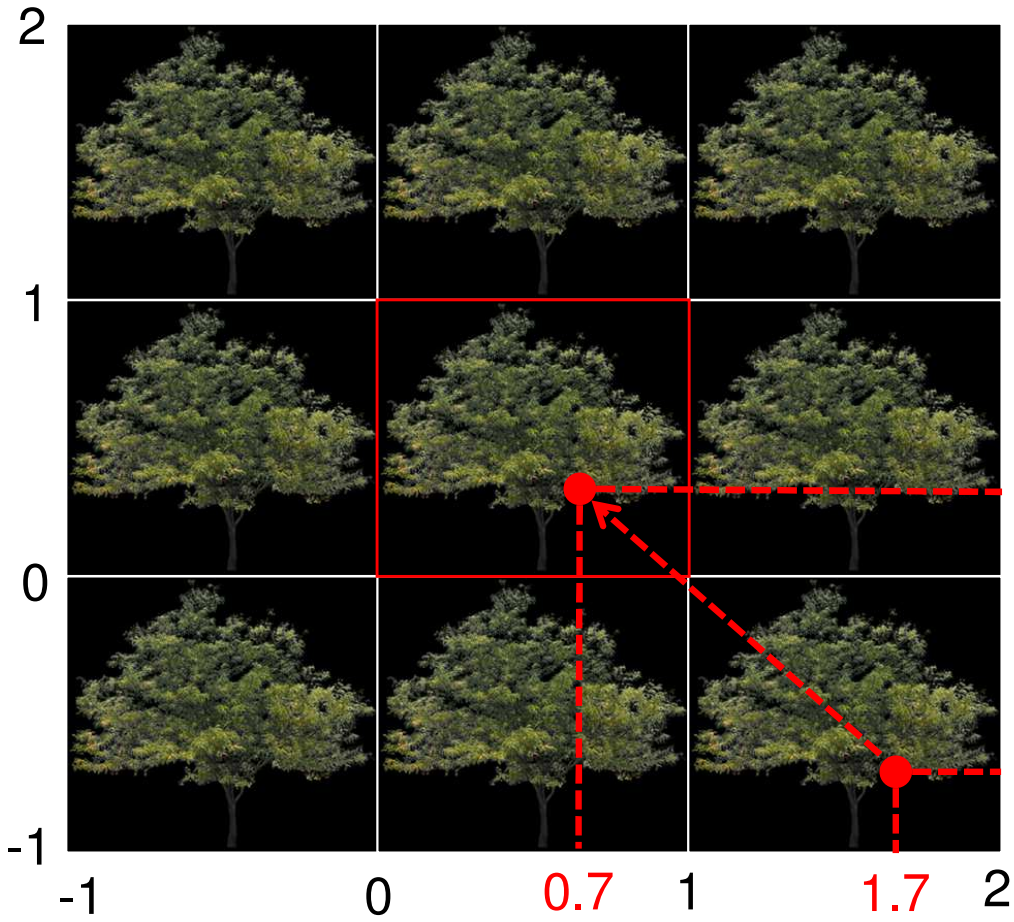
# Texture Coordinates

- **Texture Coordinates**



# Texture Coordinate Addressing Mode

- **Repeat or Wrap Mode**




*u' = frac(u)*

*v' = frac(v)*

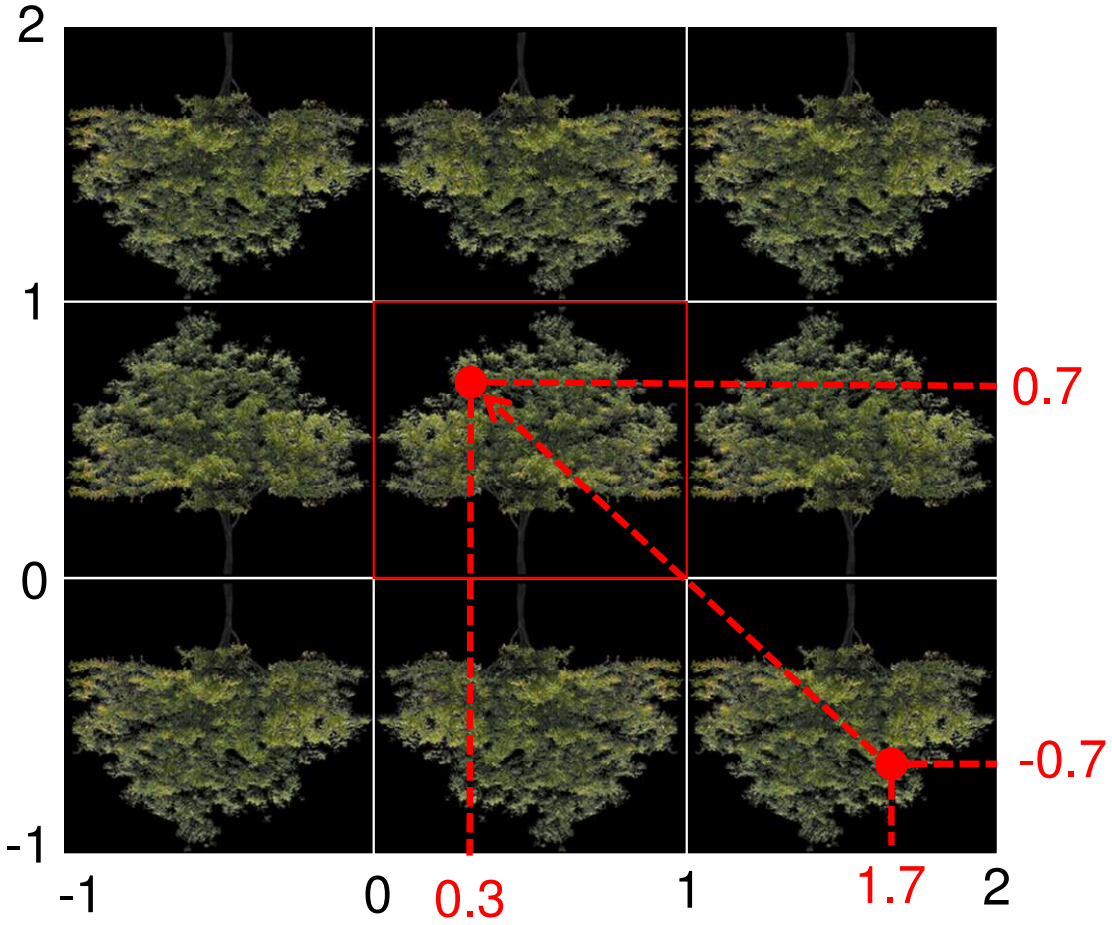
*frac(x) = x - ⌊x⌋*

*e. g., (u, v) = (1.7, -0.7)*

*(u', v') = (0.7, 0.3)*

# Texture Coordinate Addressing Mode

- **Mirror Mode**




$$u' = \begin{cases} frac(u) & if\ \lfloor u \rfloor\ is\ even \\ 1 - frac(u) & if\ \lfloor u \rfloor\ is\ odd \end{cases}$$

$$v' = \begin{cases} frac(v) & if\ \lfloor v \rfloor\ is\ even \\ 1 - frac(v) & if\ \lfloor v \rfloor\ is\ odd \end{cases}$$

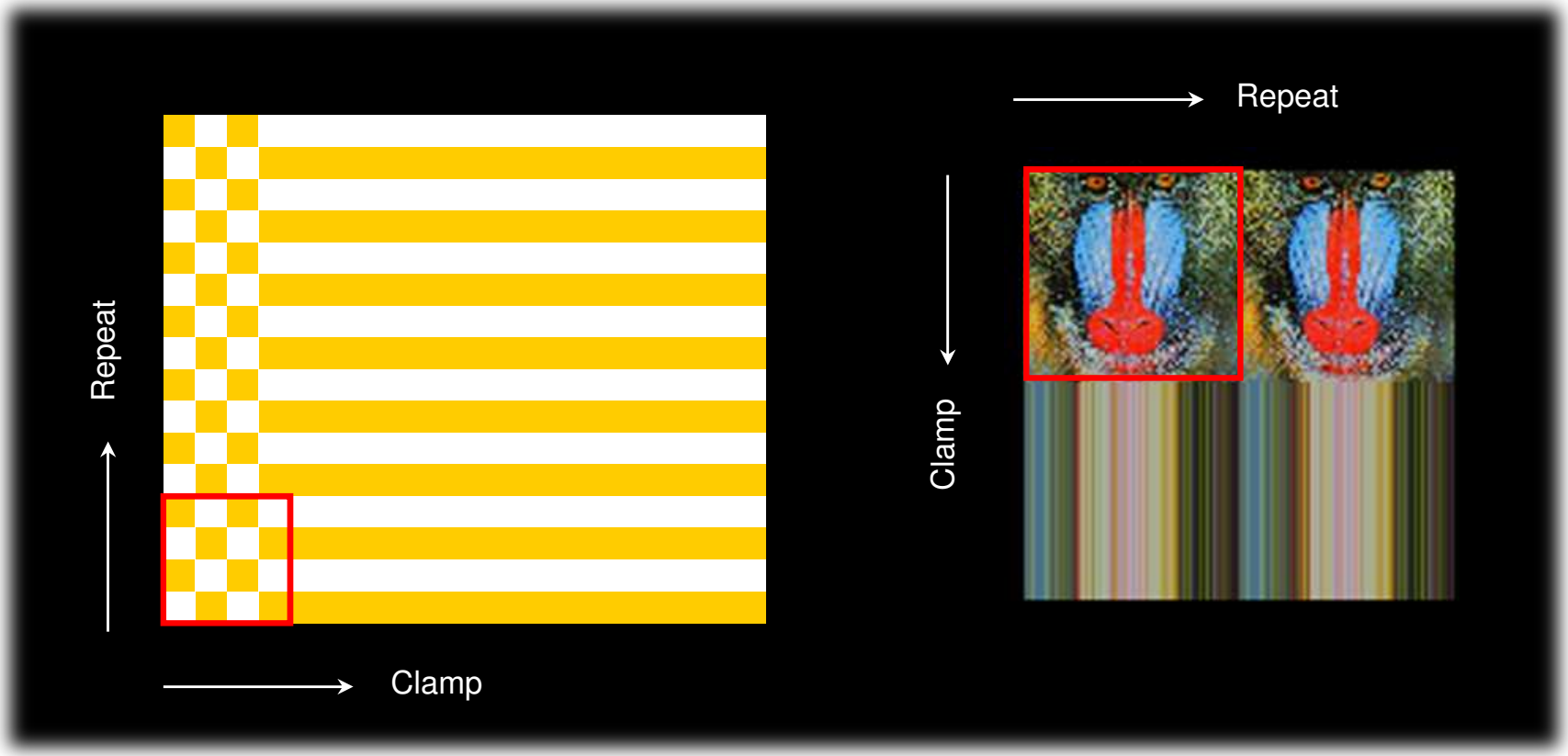
*frac(x) = x - ⌊x⌋*

*e. g., (u, v) = (1.7, -0.7)*

*(u', v') = (0.3, 0.7)*

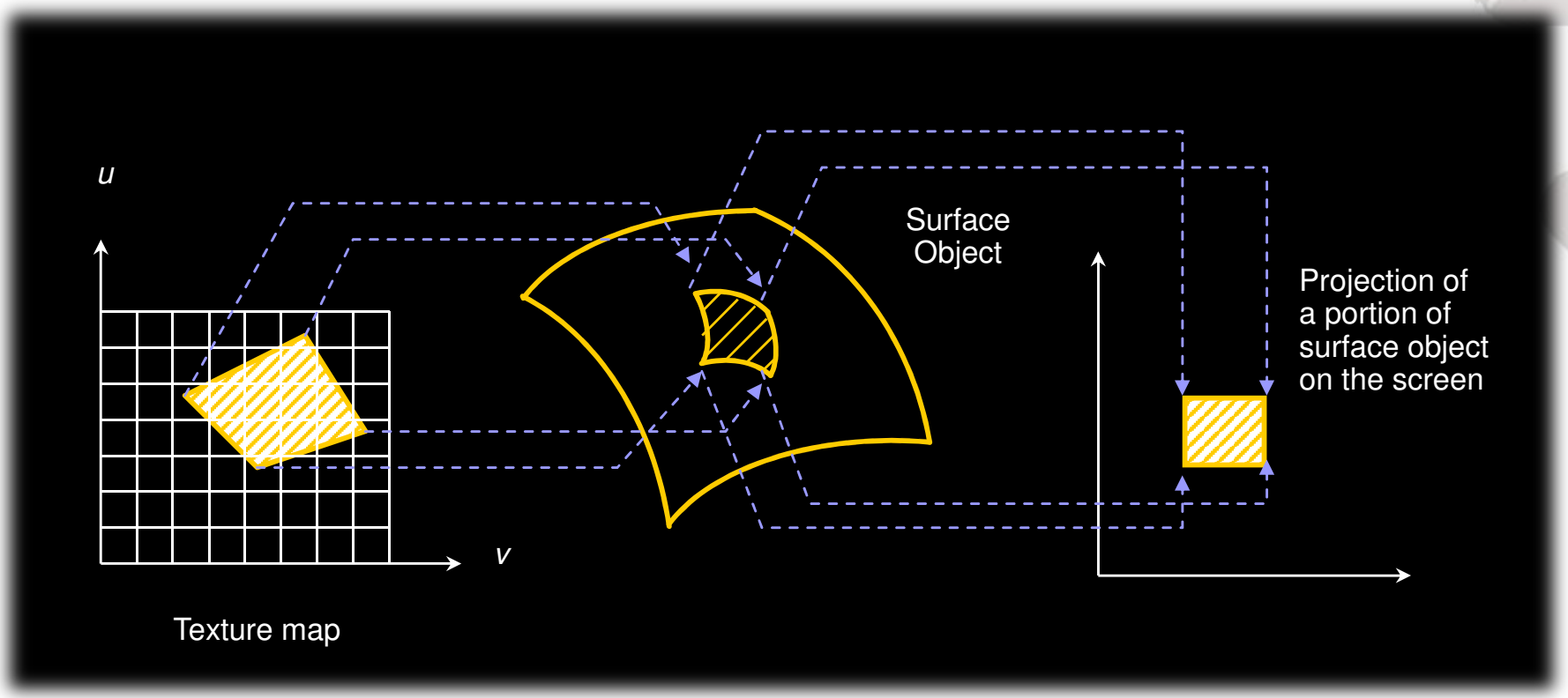
# Texture Coordinate Addressing Mode

- **Clamp Mode**



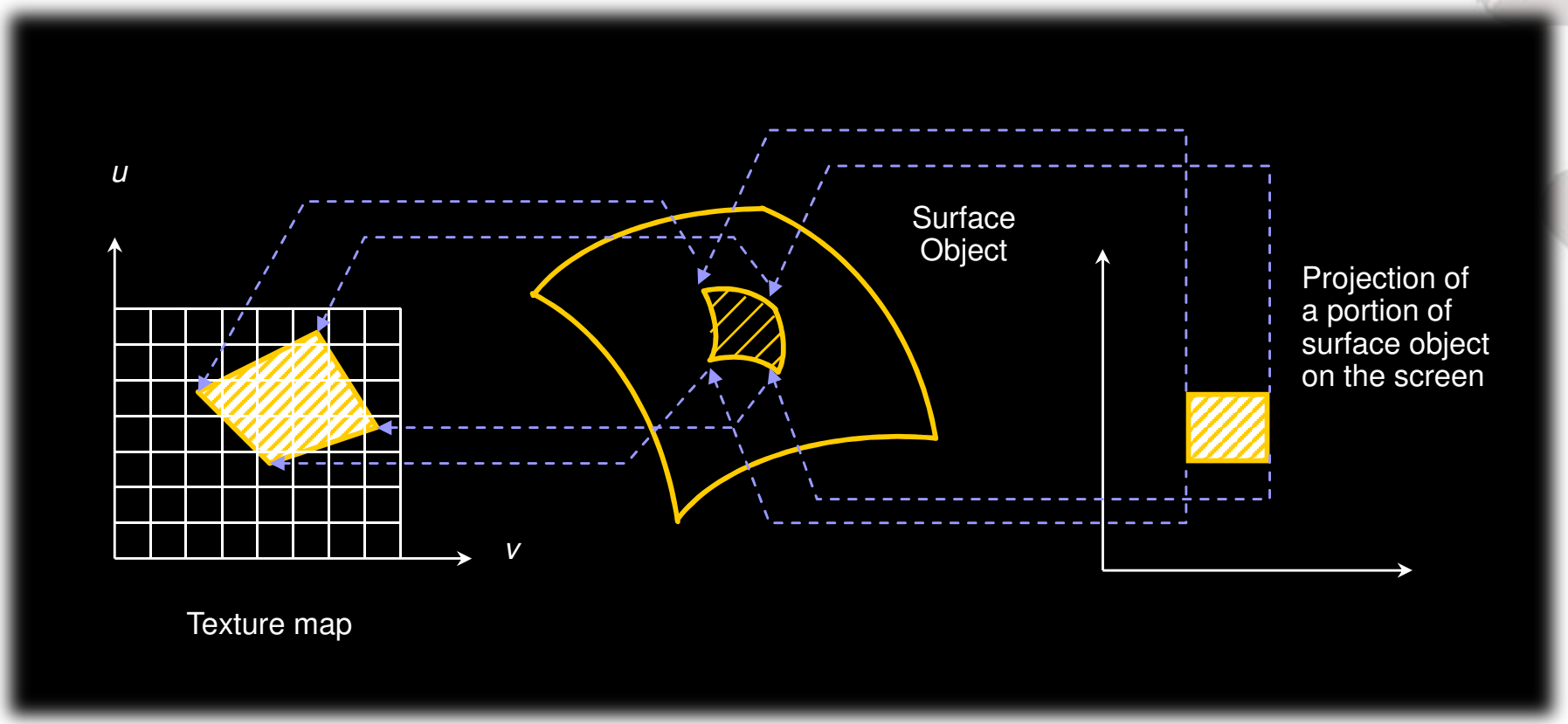
# Principle of Texture Mapping

- **Forward Mapping**



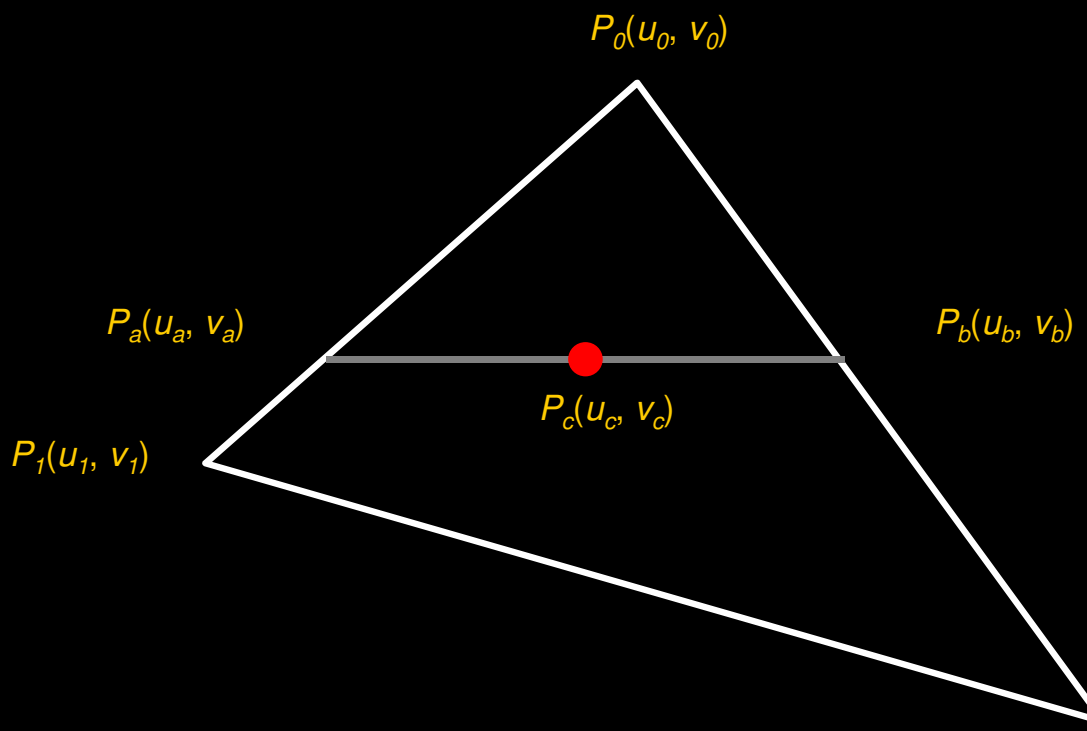
# Principle of Texture Mapping

- **Backward Mapping**



# Interpolating Texture Coordinates

- Similar to color or depth value interpolation



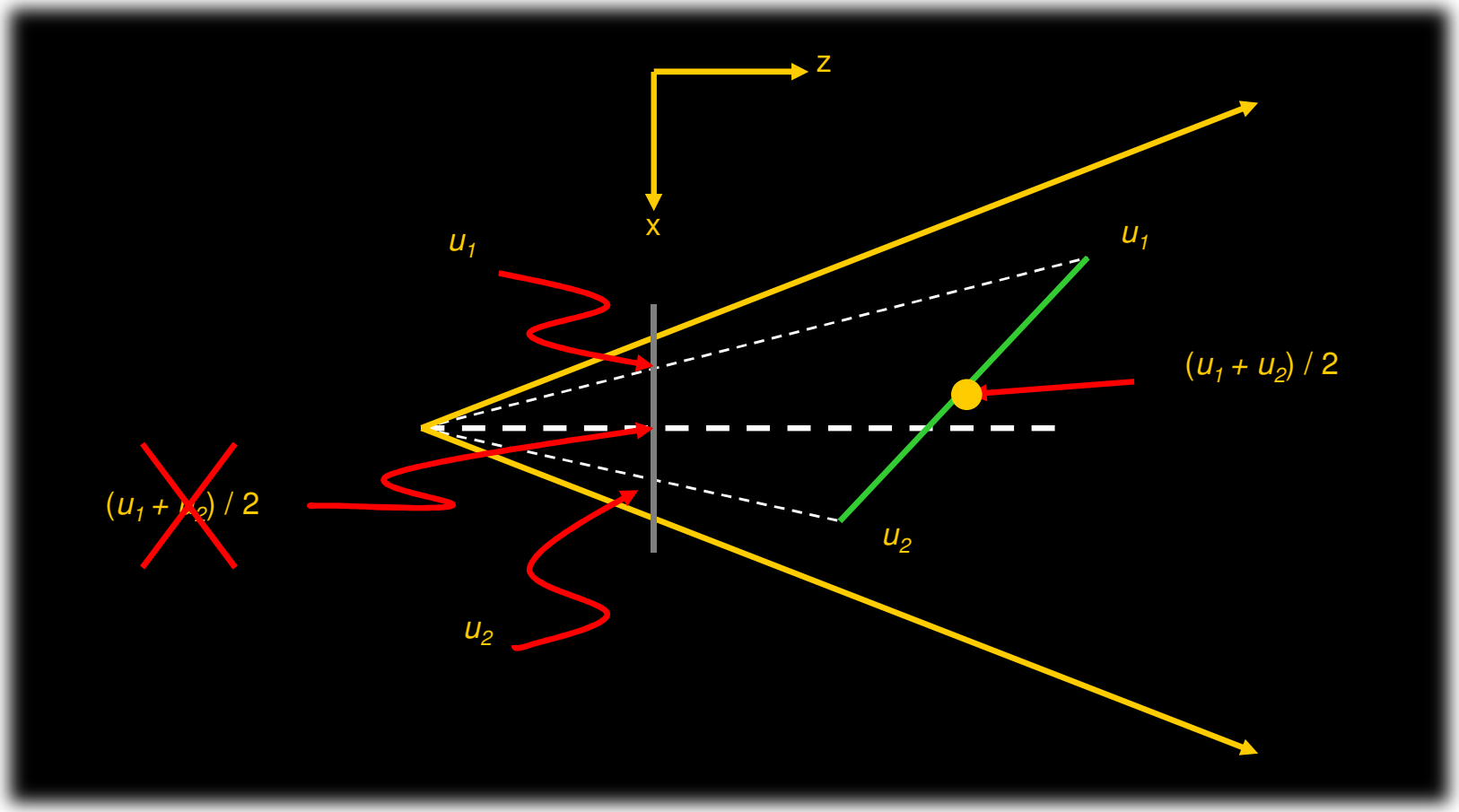

$$P_a = P_0(1 - \frac{y_a - y_0}{y_1 - y_0}) + P_1 \frac{y_a - y_0}{y_1 - y_0}$$

$$P_b = P_0(1 - \frac{y_b - y_0}{y_2 - y_0}) + P_2 \frac{y_b - y_0}{y_2 - y_0}$$

$$P_c = P_a(1 - \frac{x_c - x_a}{x_b - x_a}) + P_b \frac{x_c - x_a}{x_b - x_a}$$

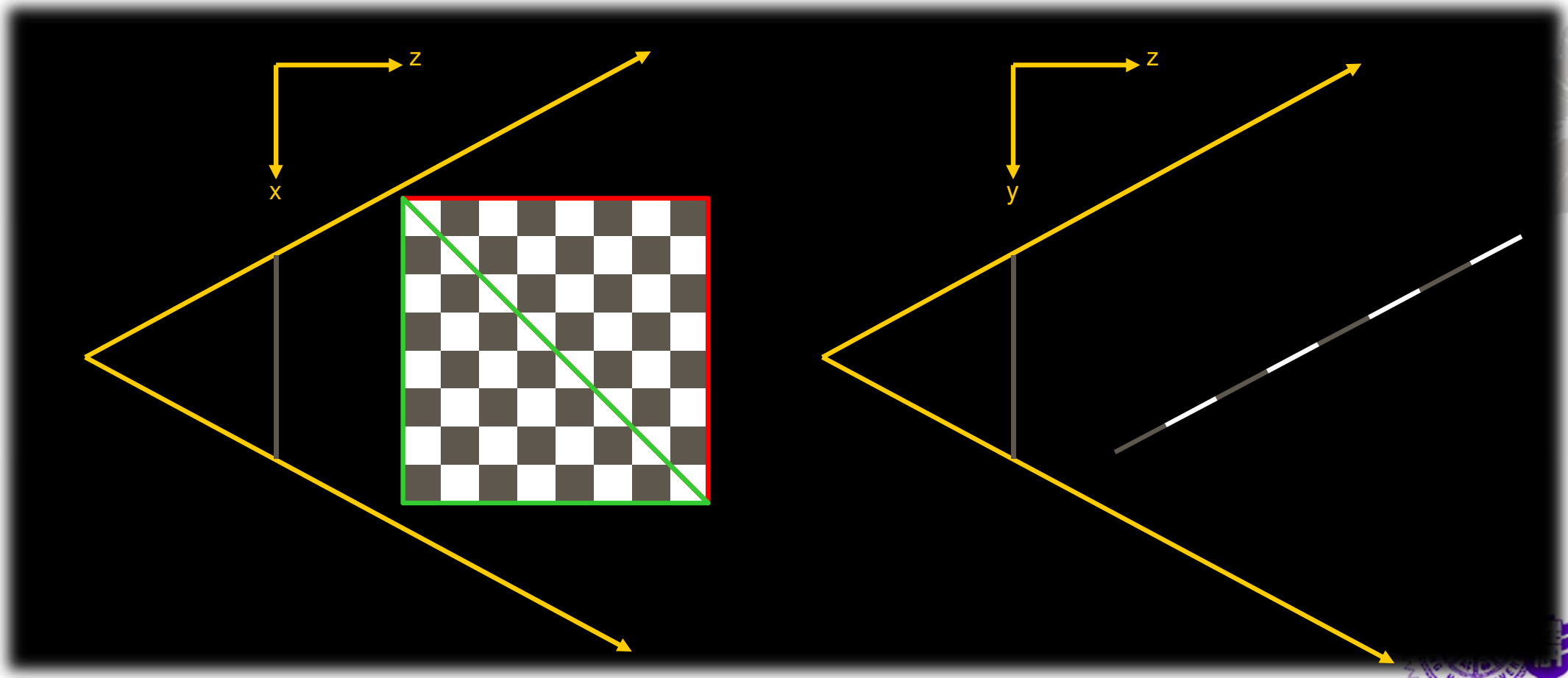
# Interpolating Texture Coordinates

- Linear interpolation in screen space does not imply linear interpolation in world space



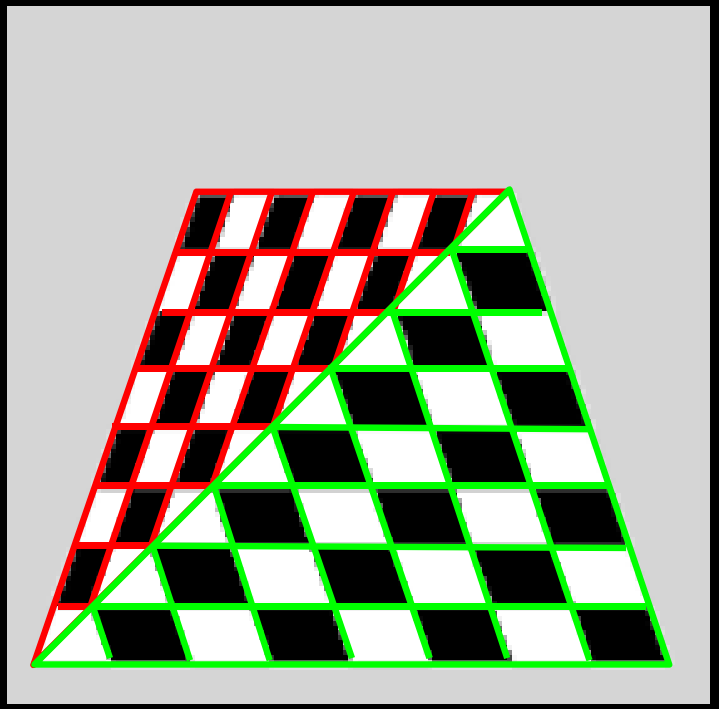
# Interpolating Texture Coordinates

- Example of using world space texture coordinates directly

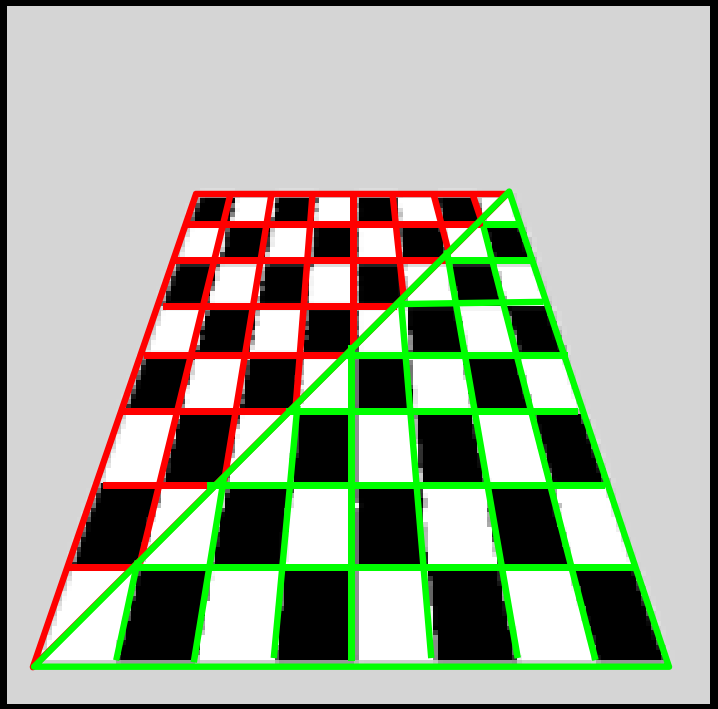


# Interpolating Texture Coordinates

- Result of using world space texture coordinates



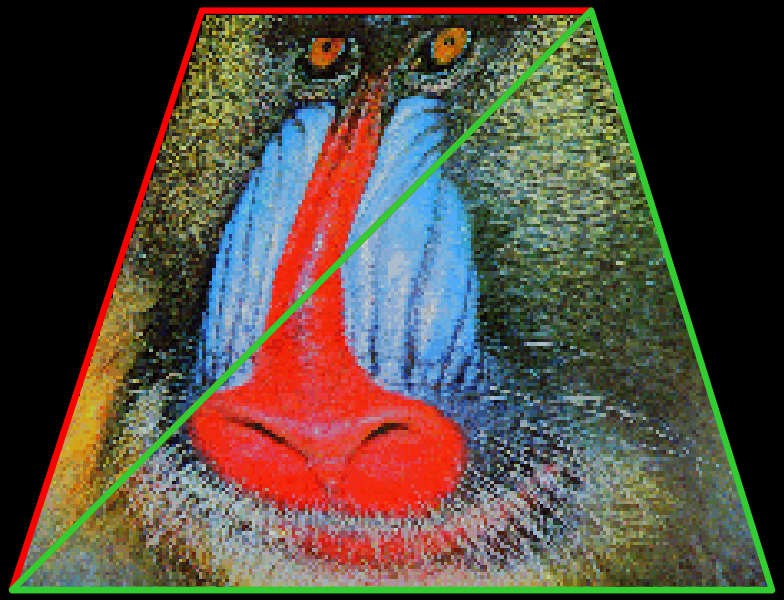
Actual result from the screen



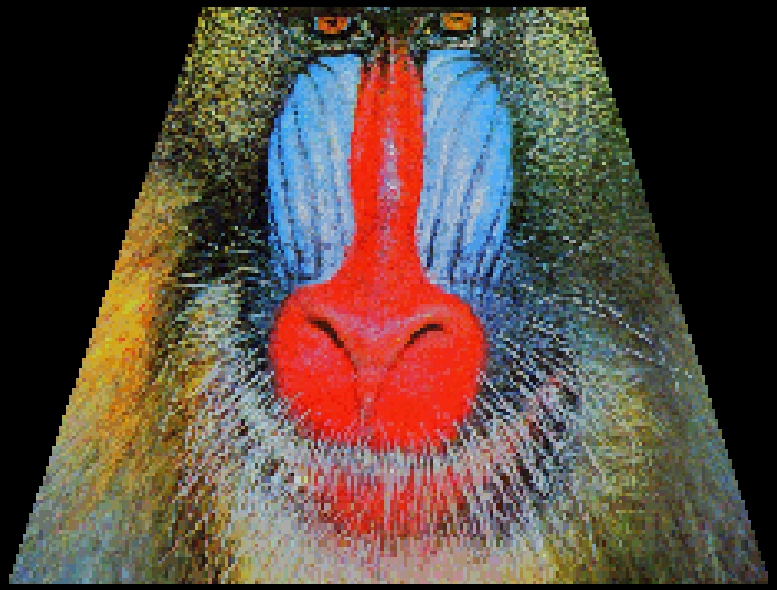
Expect to see on the screen

# Interpolating Texture Coordinates

- Another result of using world space texture coordinates



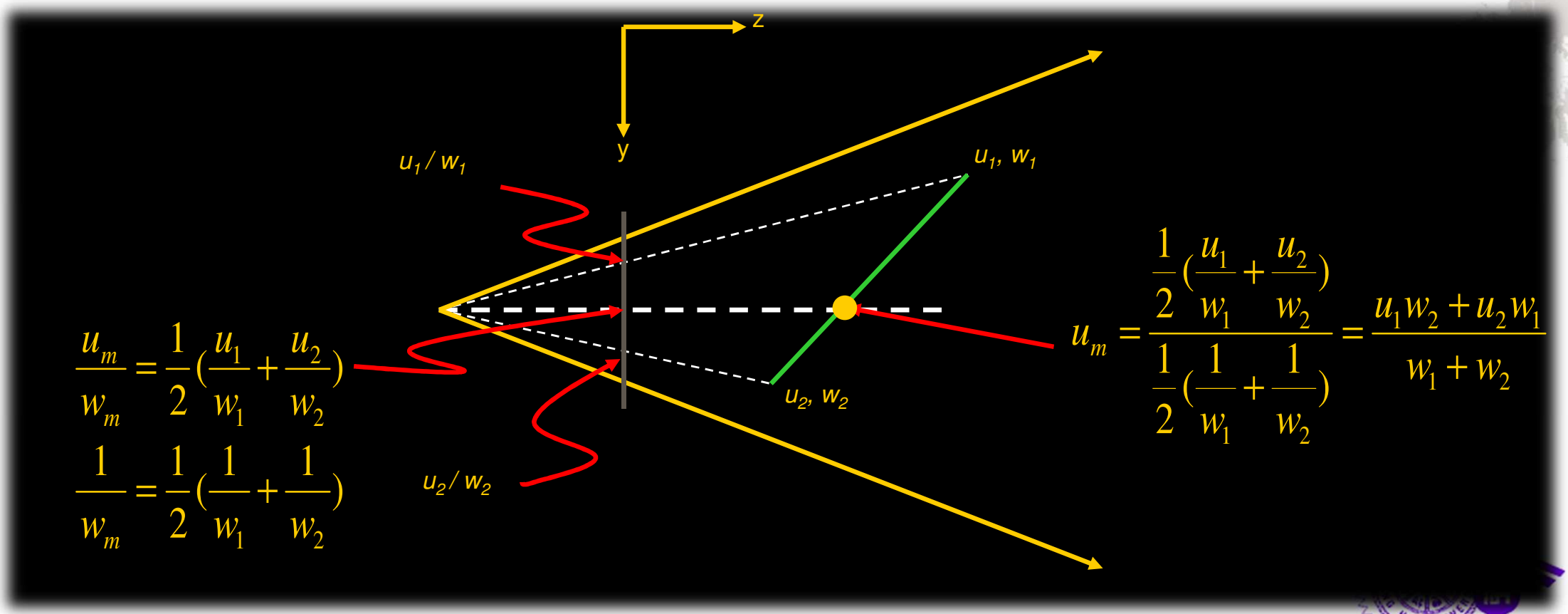
Actual result on the screen



Expect to see on the screen

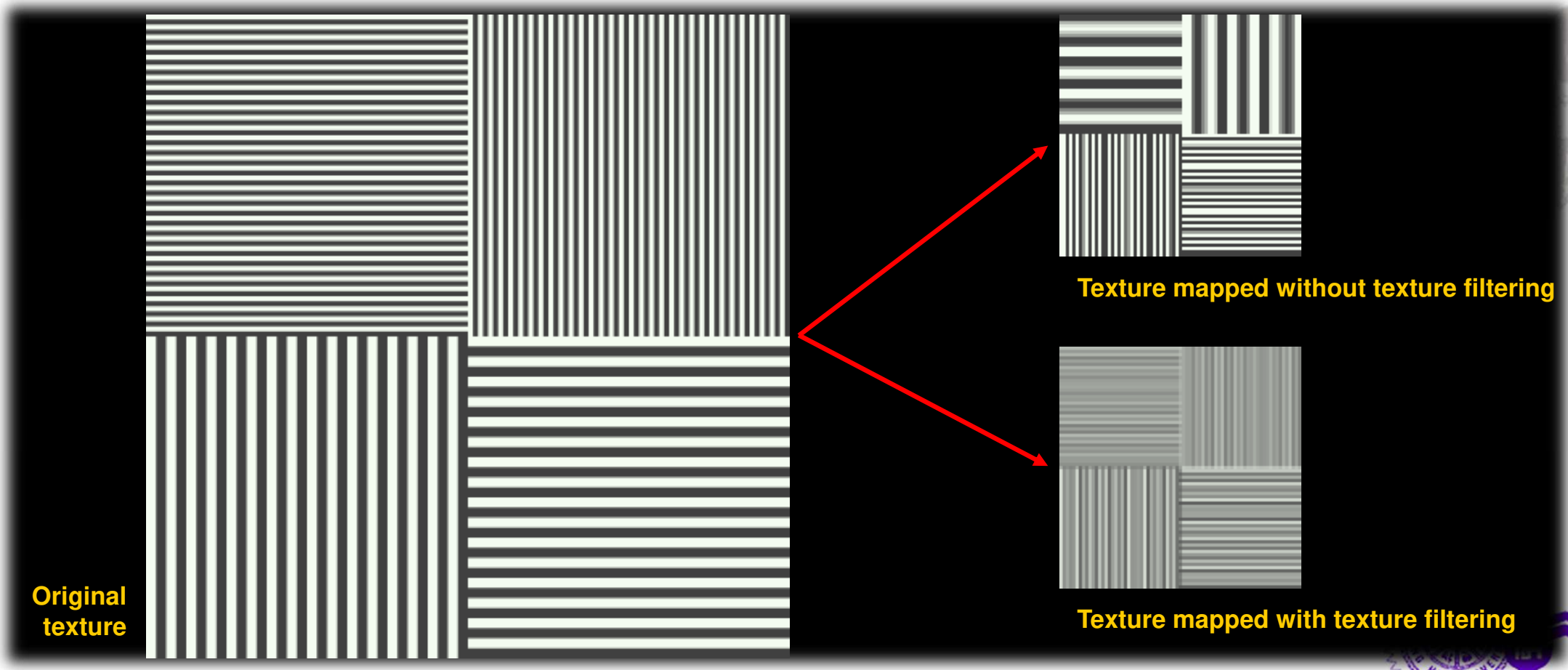
# Interpolating Texture Coordinates

- Using perspective correction to resolve the incorrect result of using world space texture coordinates



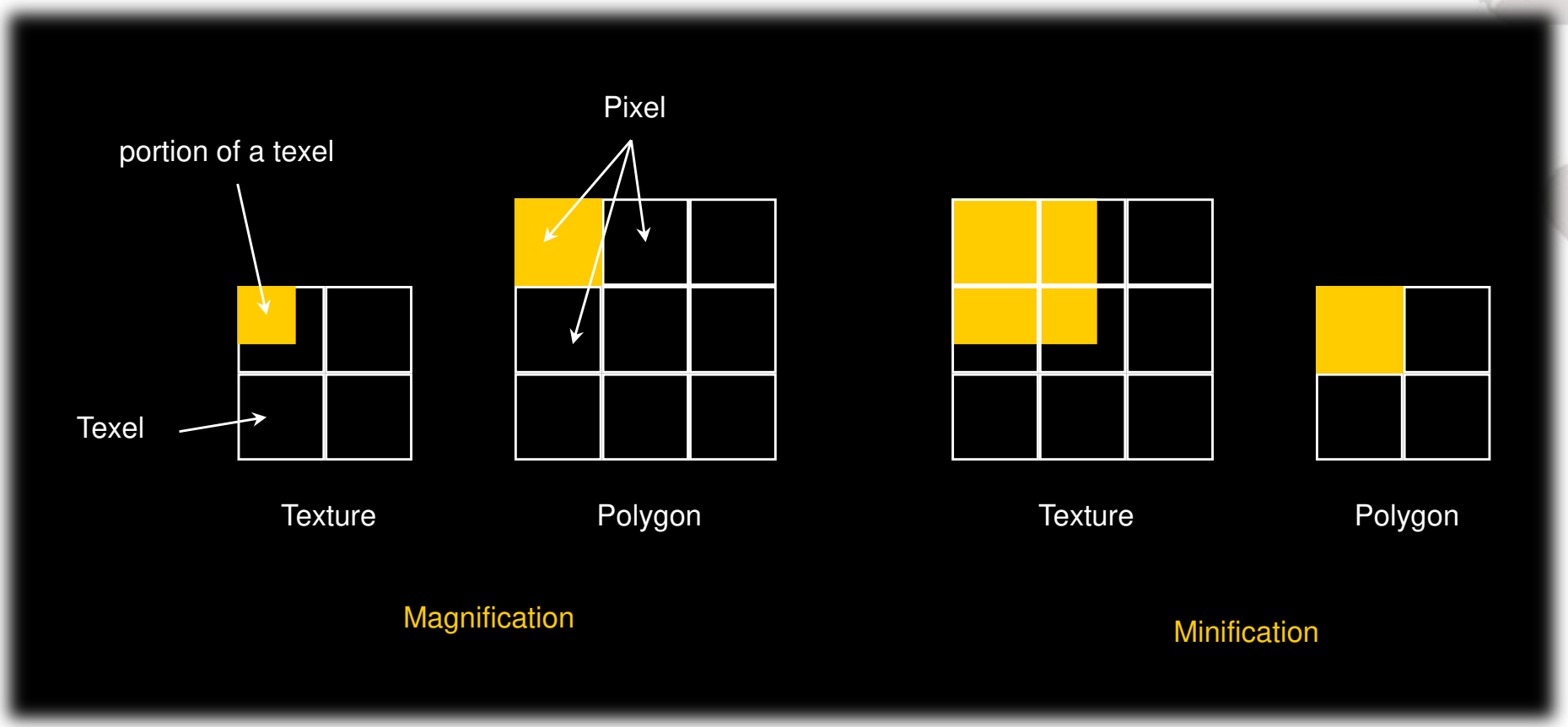
# Texture Filtering

- Why need texture filtering?
  - A method to resolve texture aliasing



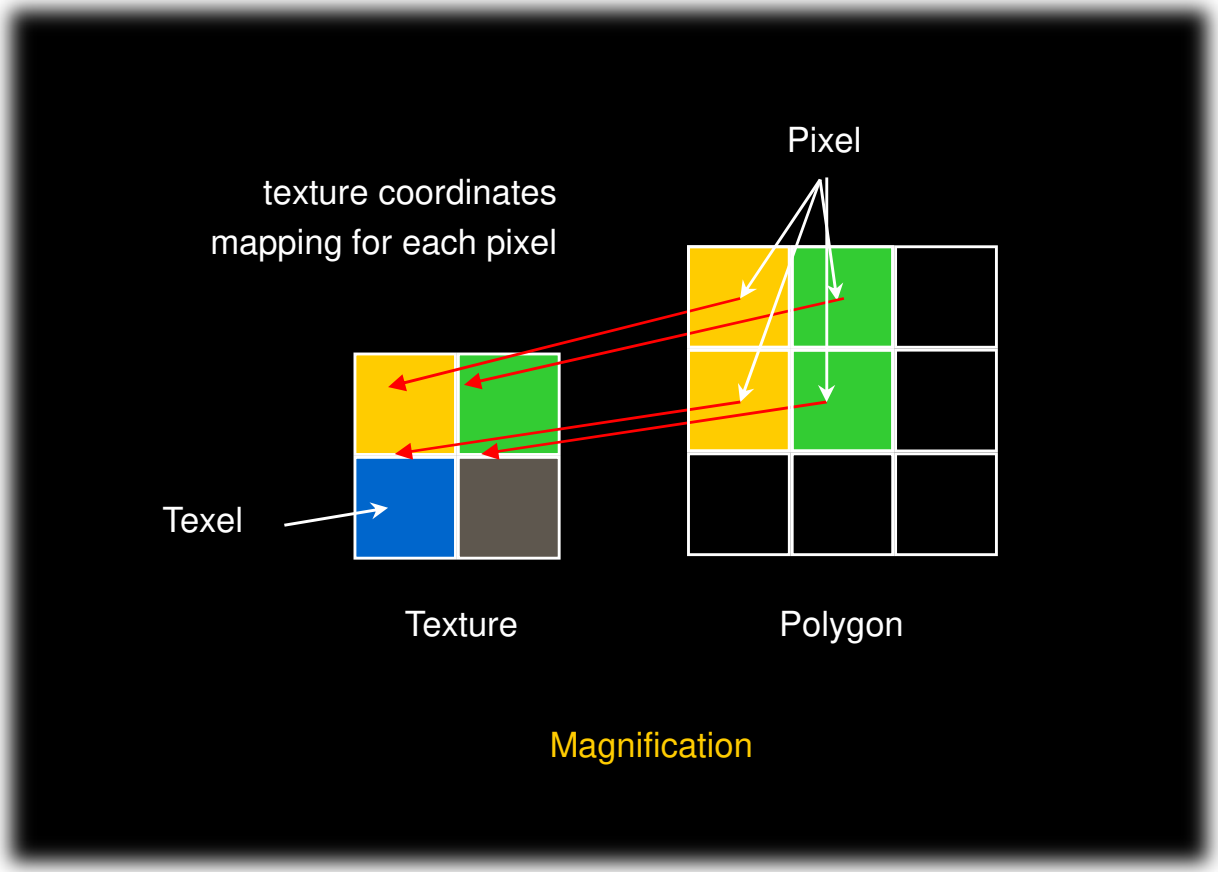
# Texture Filtering

- **Texture Filtering Modes**



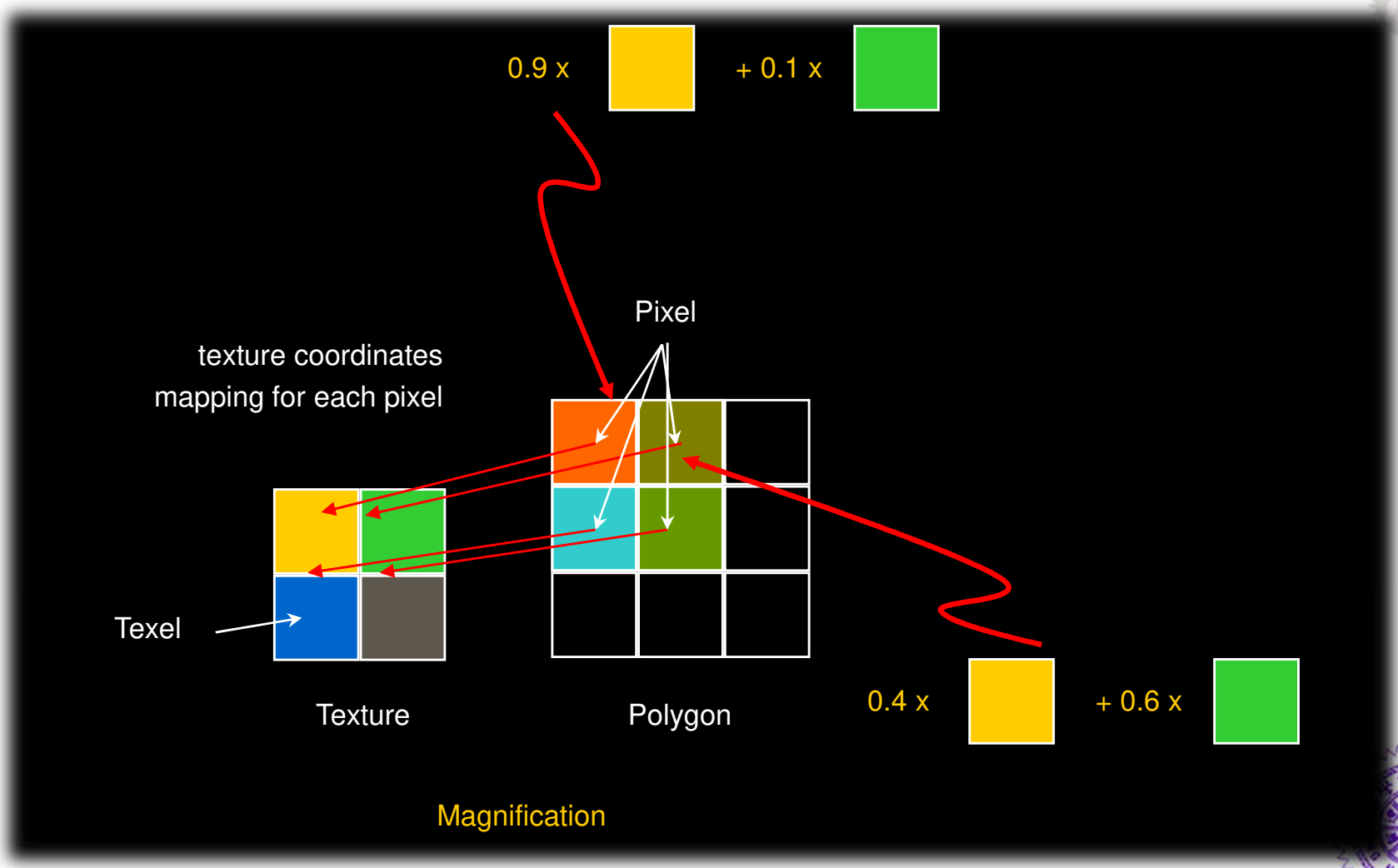
# Magnification

- Nearest Sampling



# Magnification

- Linear Sampling



# Magnification

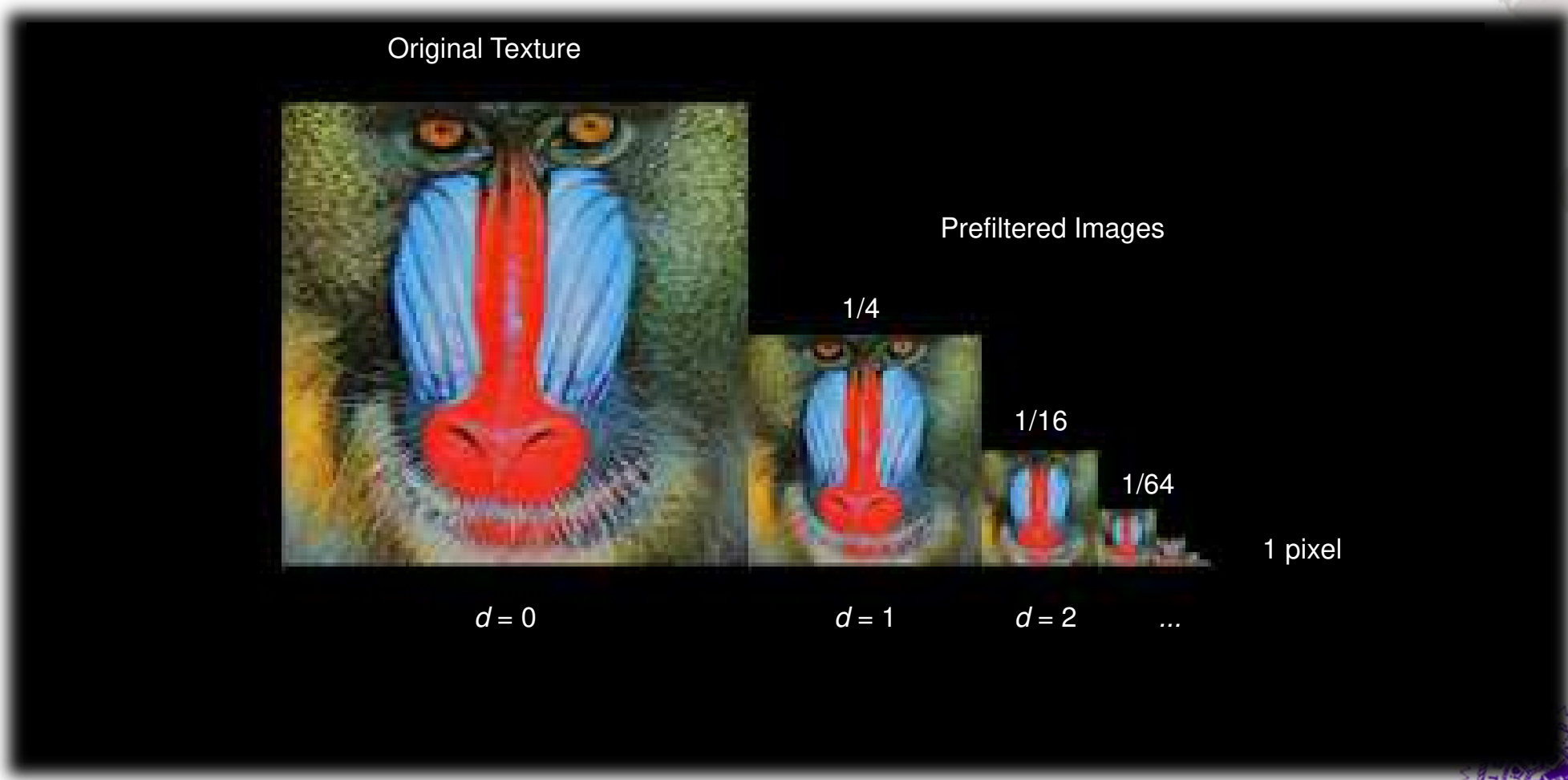
- Nearest Sampling vs. Linear Sampling

Nearest Sampling

Linear Sampling

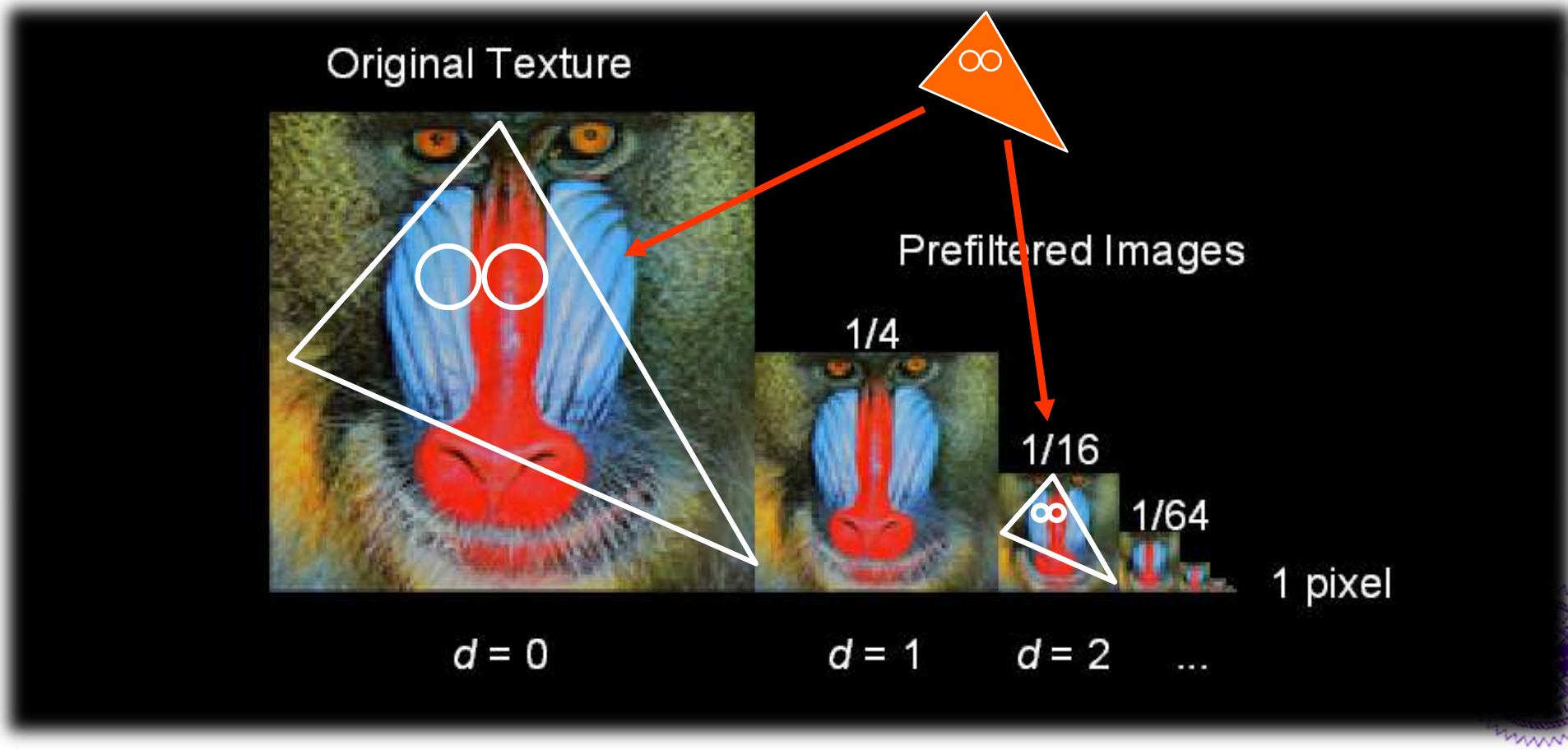
# Texture Mipmapping for Minification

- Texture Mipmap (Multum in Parvo)



# Level of Detail (LOD)

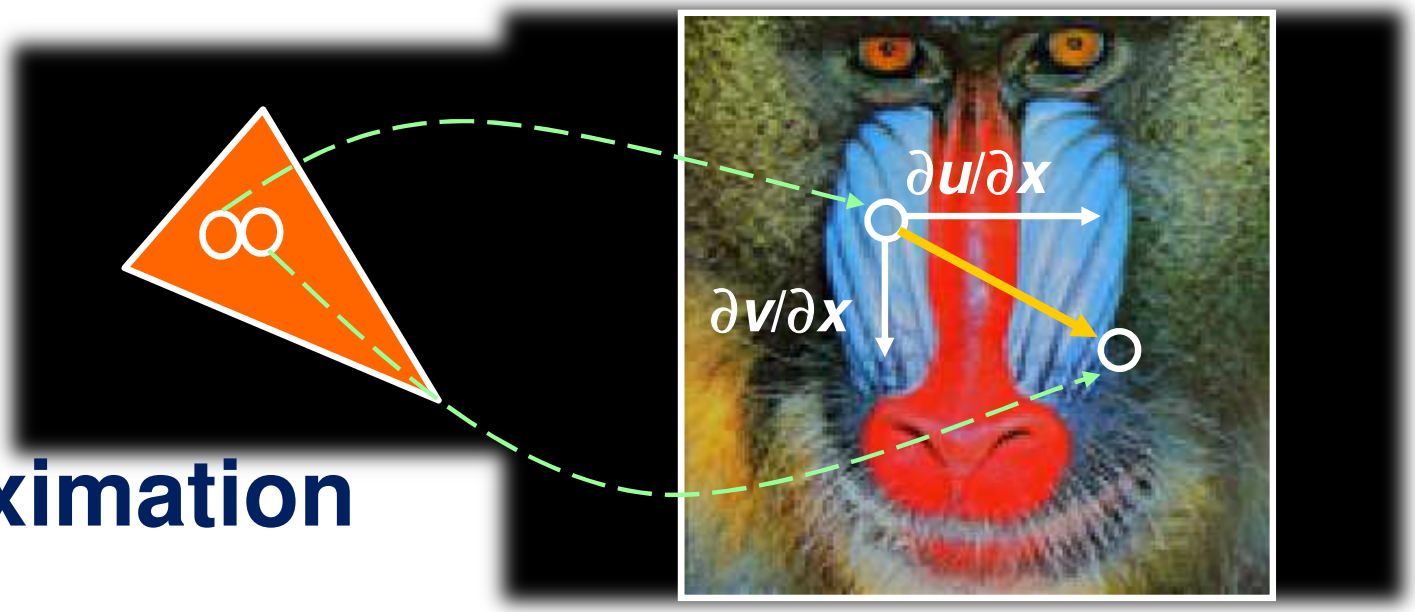
- Concept
  - Find a suitable pre-filtered image for later texture filtering



# LOD Calculation

- **Definition (OpenGL)**

$$LOD = \log_2\left(\max\left\{\sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2}, \sqrt{\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2}\right\}\right)$$



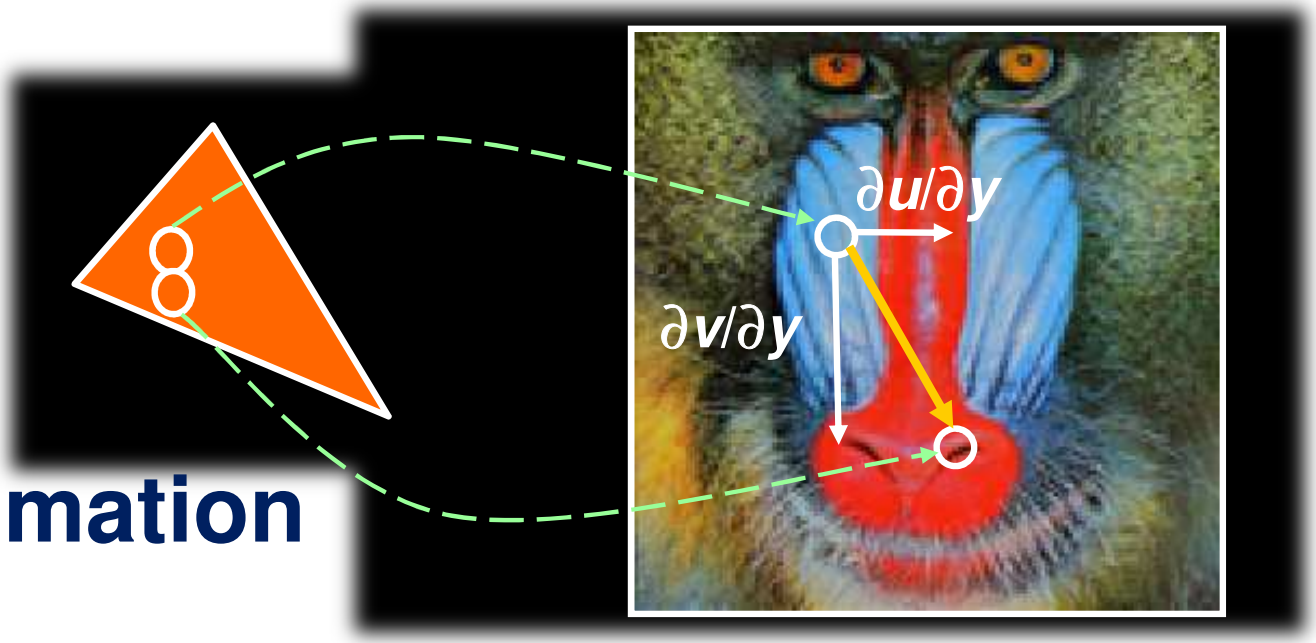

- **Approximation**

$$LOD = \log_2\left(\max\left\{\left|\frac{\partial u}{\partial x}\right|, \left|\frac{\partial v}{\partial x}\right|, \left|\frac{\partial u}{\partial y}\right|, \left|\frac{\partial v}{\partial y}\right|\right\}\right)$$

# LOD Calculation

- **Definition (OpenGL)**

$$LOD = \log_2\left(\max\left\{\sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2}, \sqrt{\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2}\right\}\right)$$

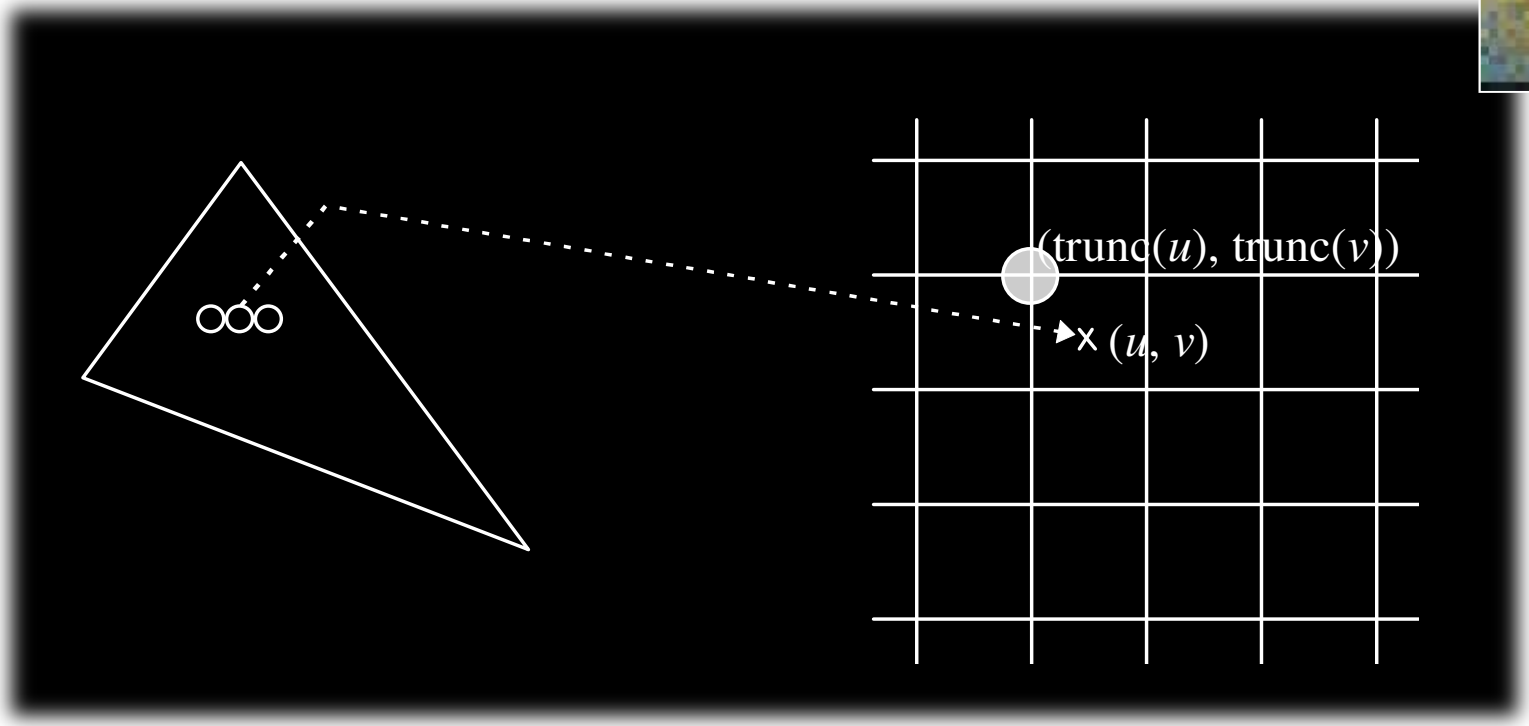



- **Approximation**

$$LOD = \log_2\left(\max\left\{\left|\frac{\partial u}{\partial x}\right|, \left|\frac{\partial v}{\partial x}\right|, \left|\frac{\partial u}{\partial y}\right|, \left|\frac{\partial v}{\partial y}\right|\right\}\right)$$

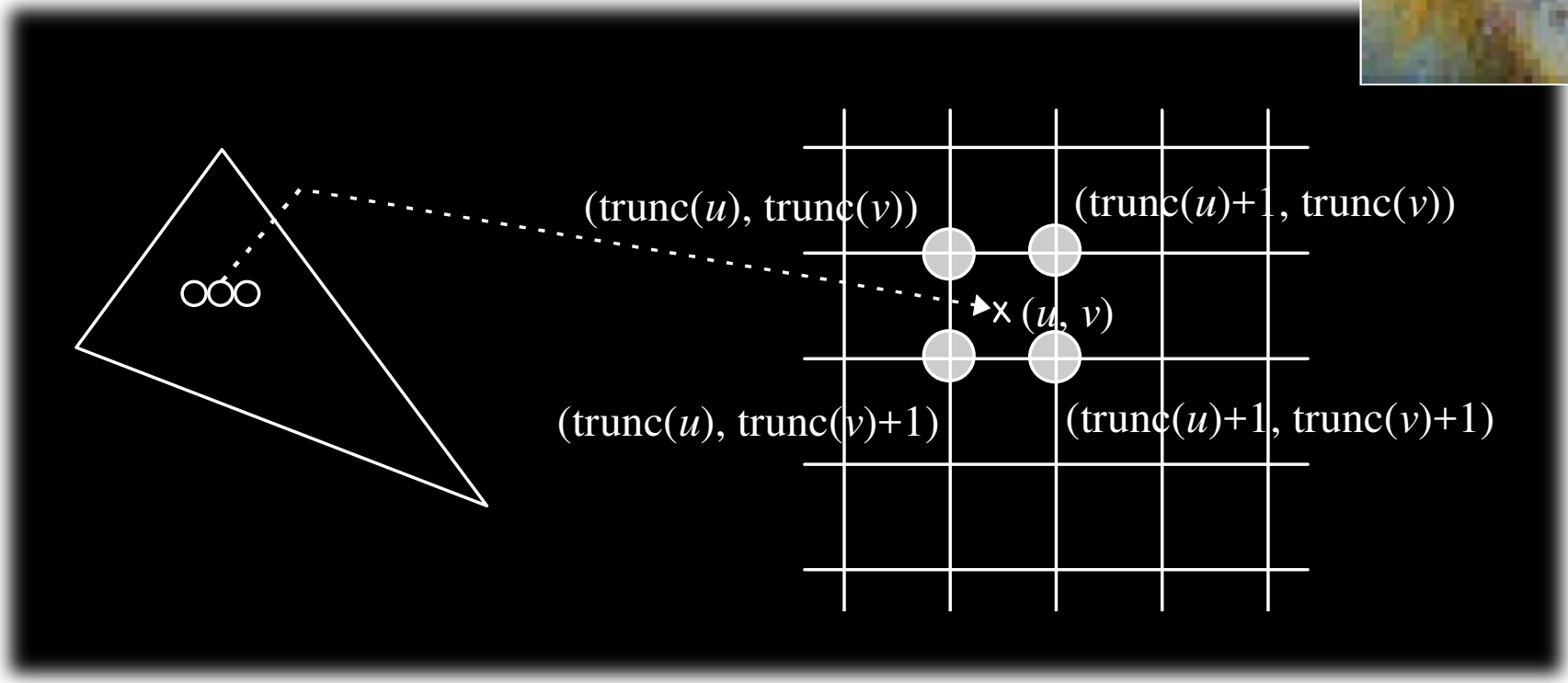
# Minification

- Nearest



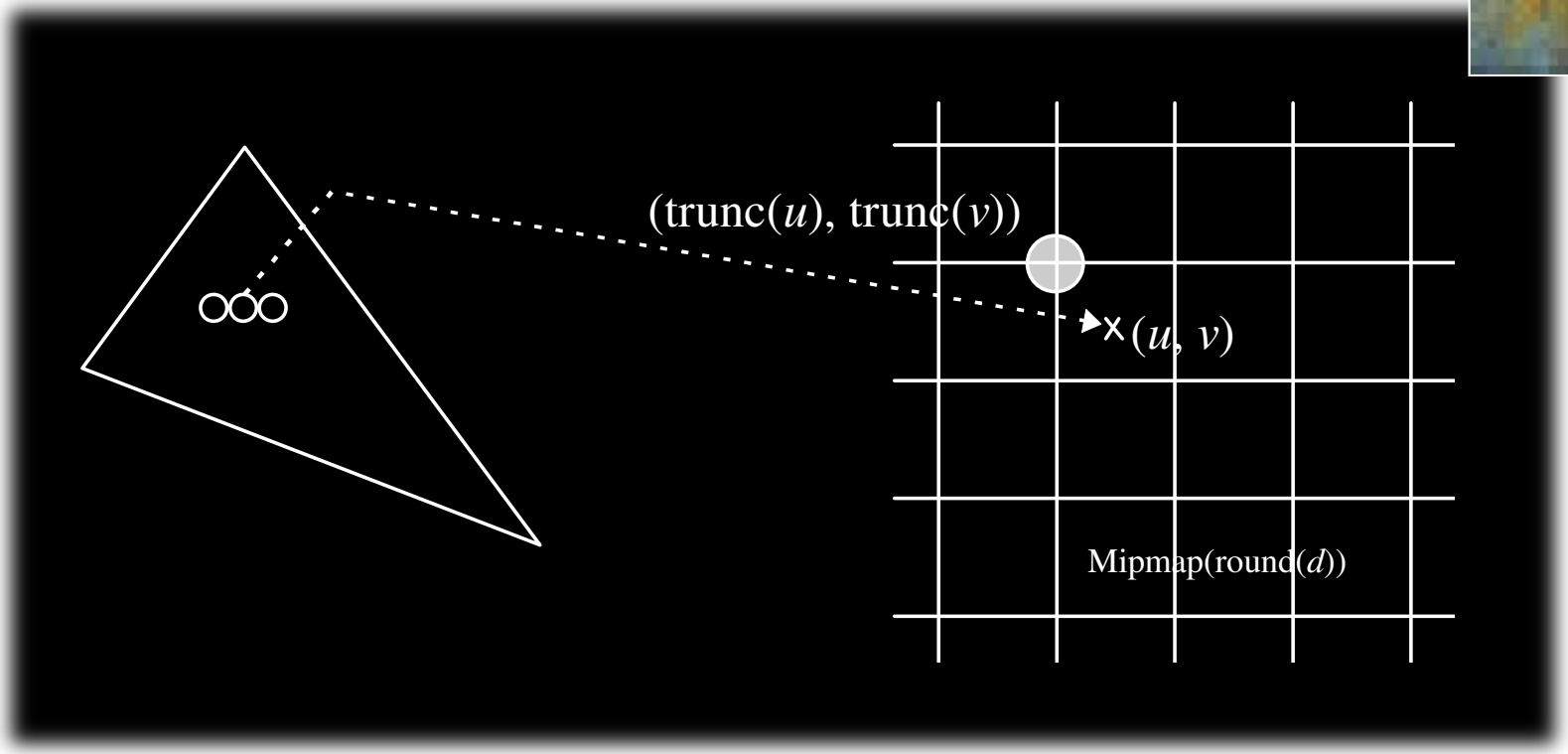
# Minification

- Linear



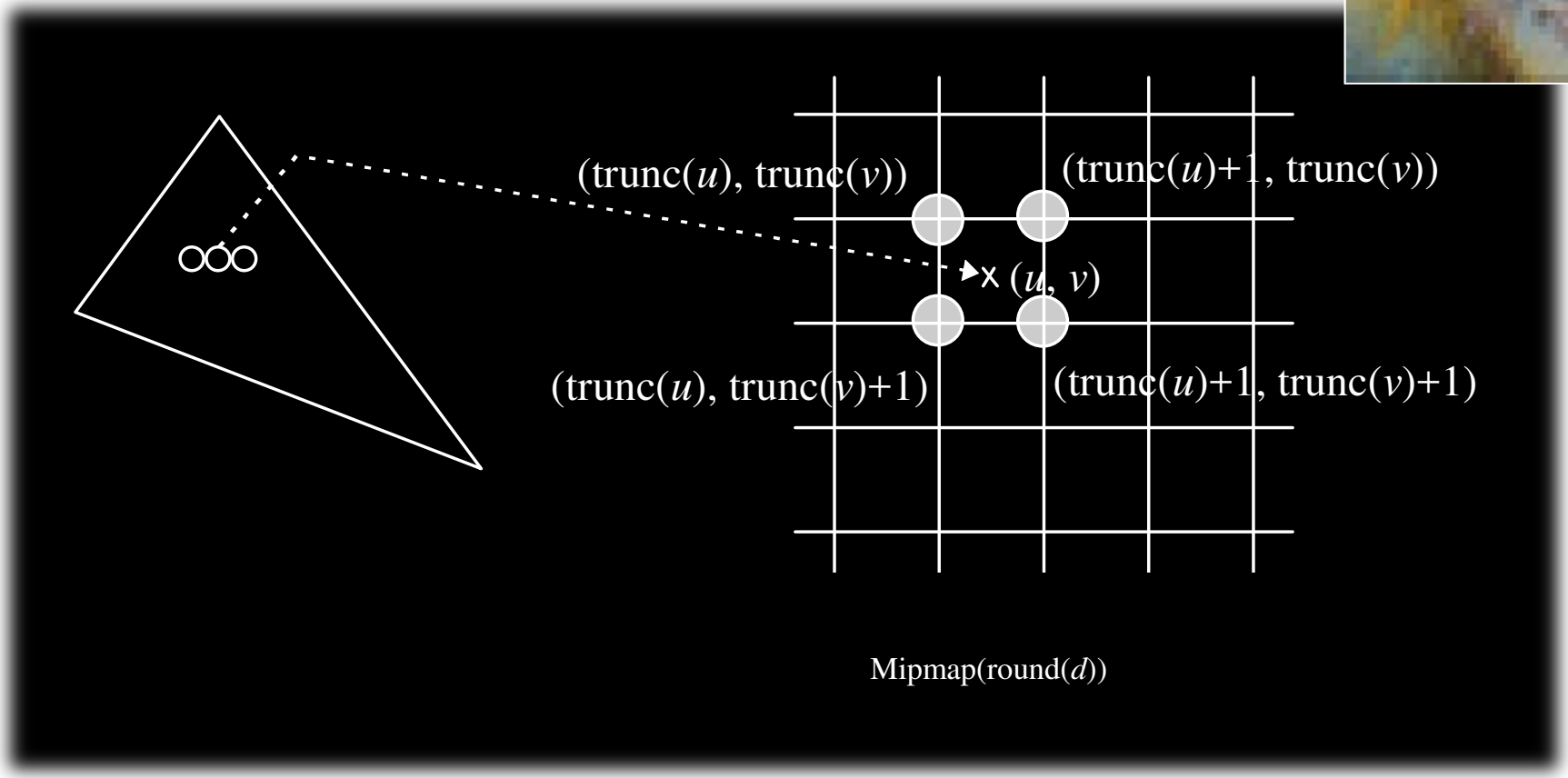
# Minification

- Nearest_Mipmap_Nearest



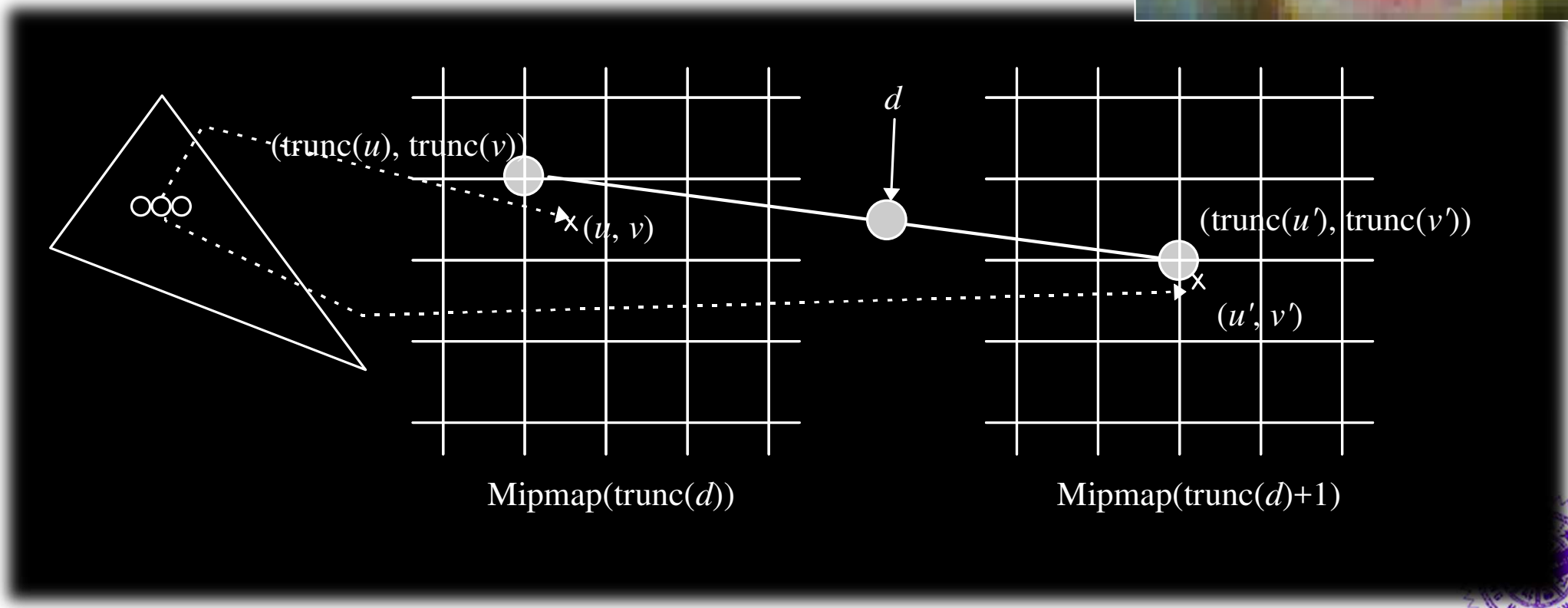
# Minification

- Linear_Mipmap_Nearest



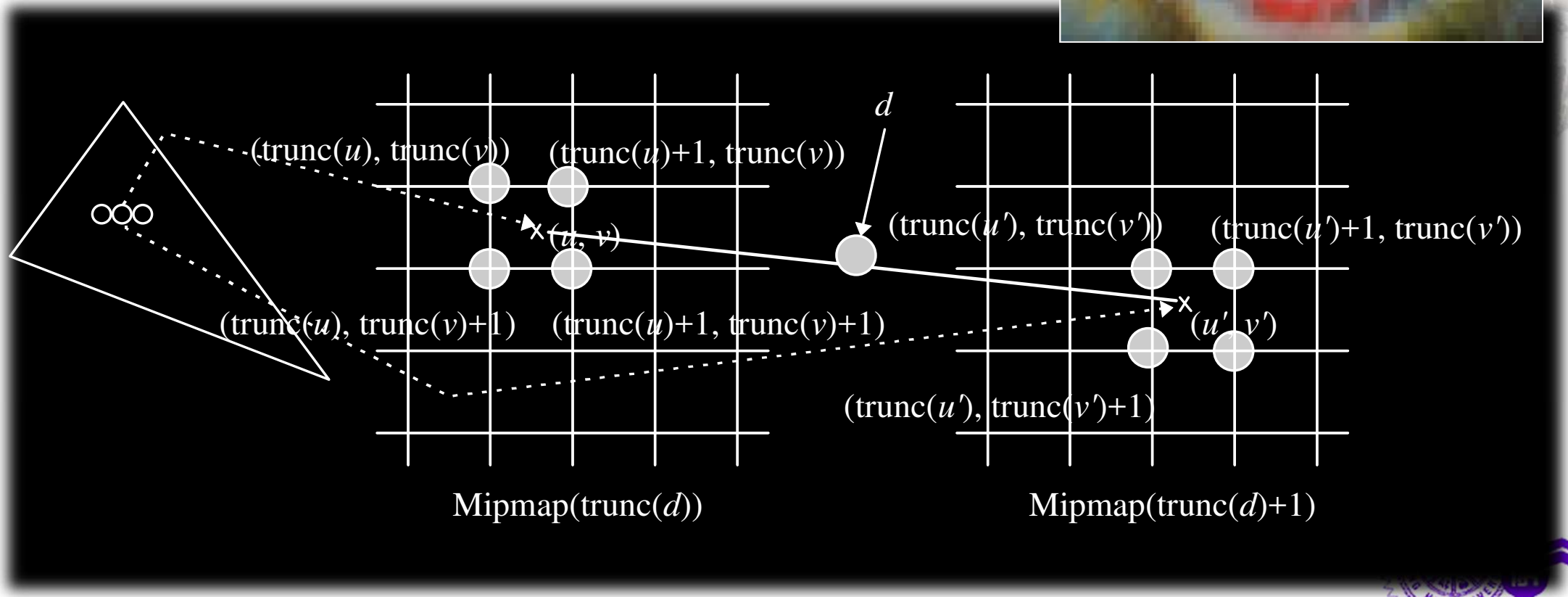
# Minification

- Nearest_Mipmap_Linear

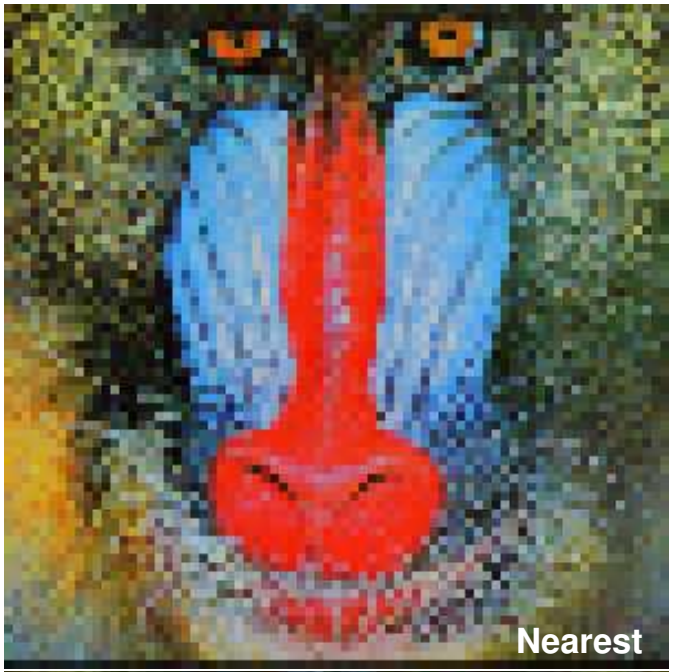


# Minification

- Linear_Mipmap_Linear



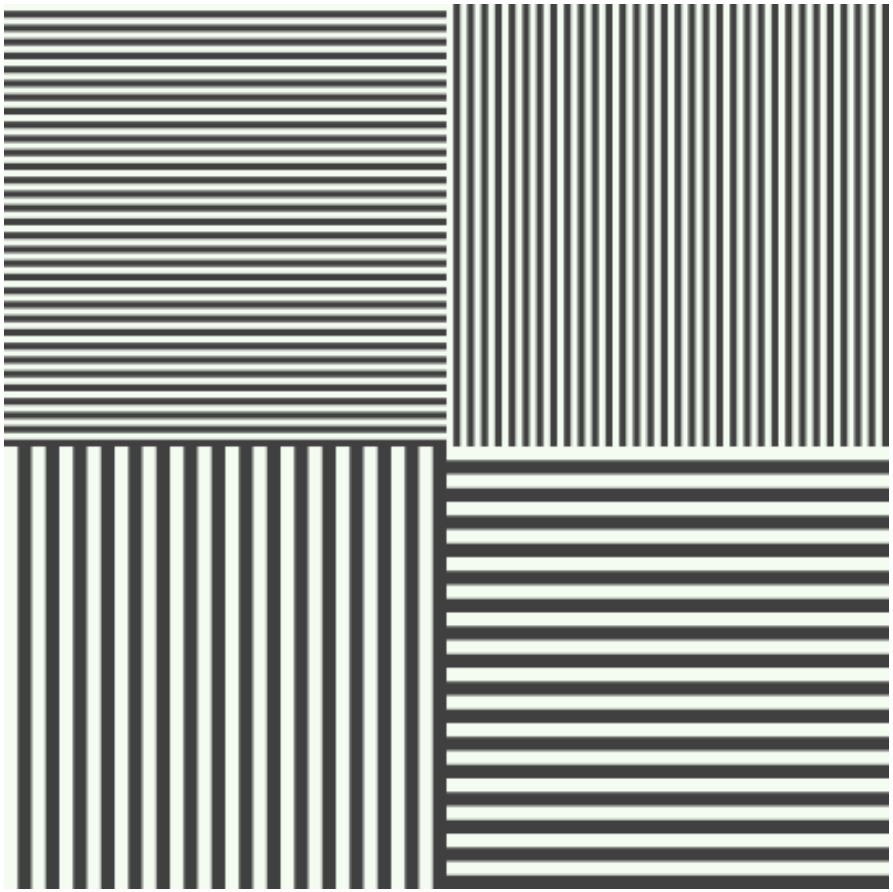
# Minification



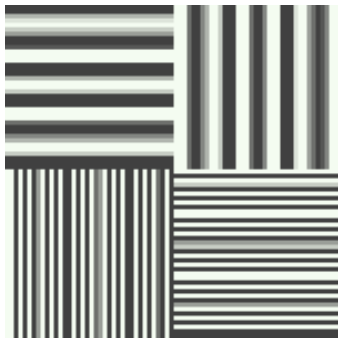




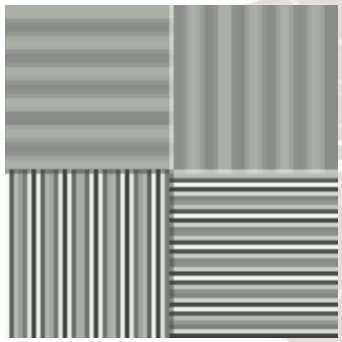

# Minification



Original texture



Nearest



Linear

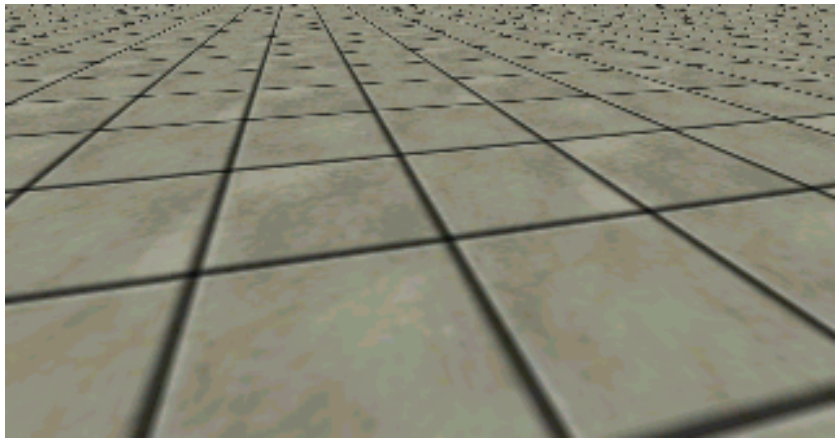
Nearest_Mipmap_Nearest

Linear_Mipmap_Nearest

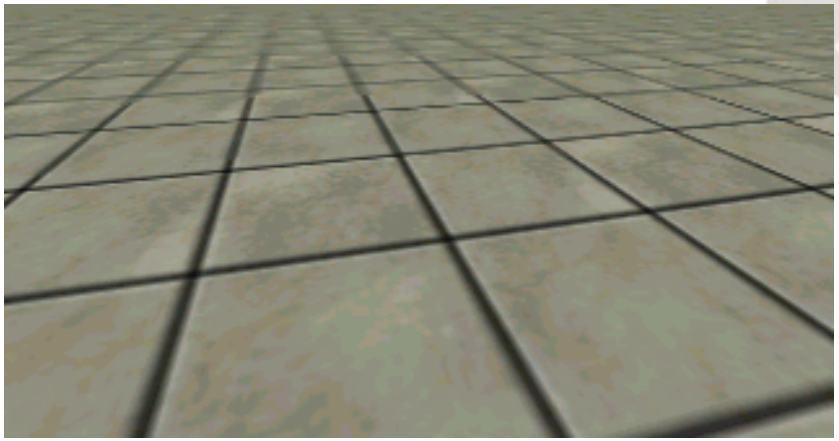
Nearest_Mipmap_Linear

Linear_Mipmap_Linear

# Application of Texture Mipmap



Without texture mipmaping



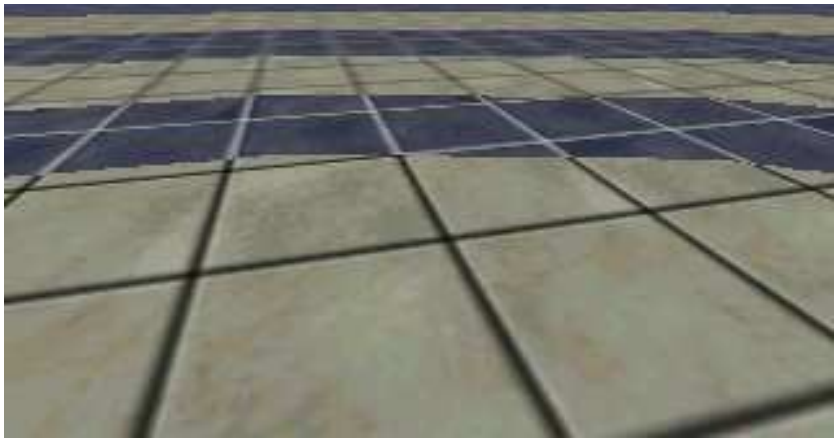
With texture mipmaping

$d = 3$

$d = 2$

$d = 1$

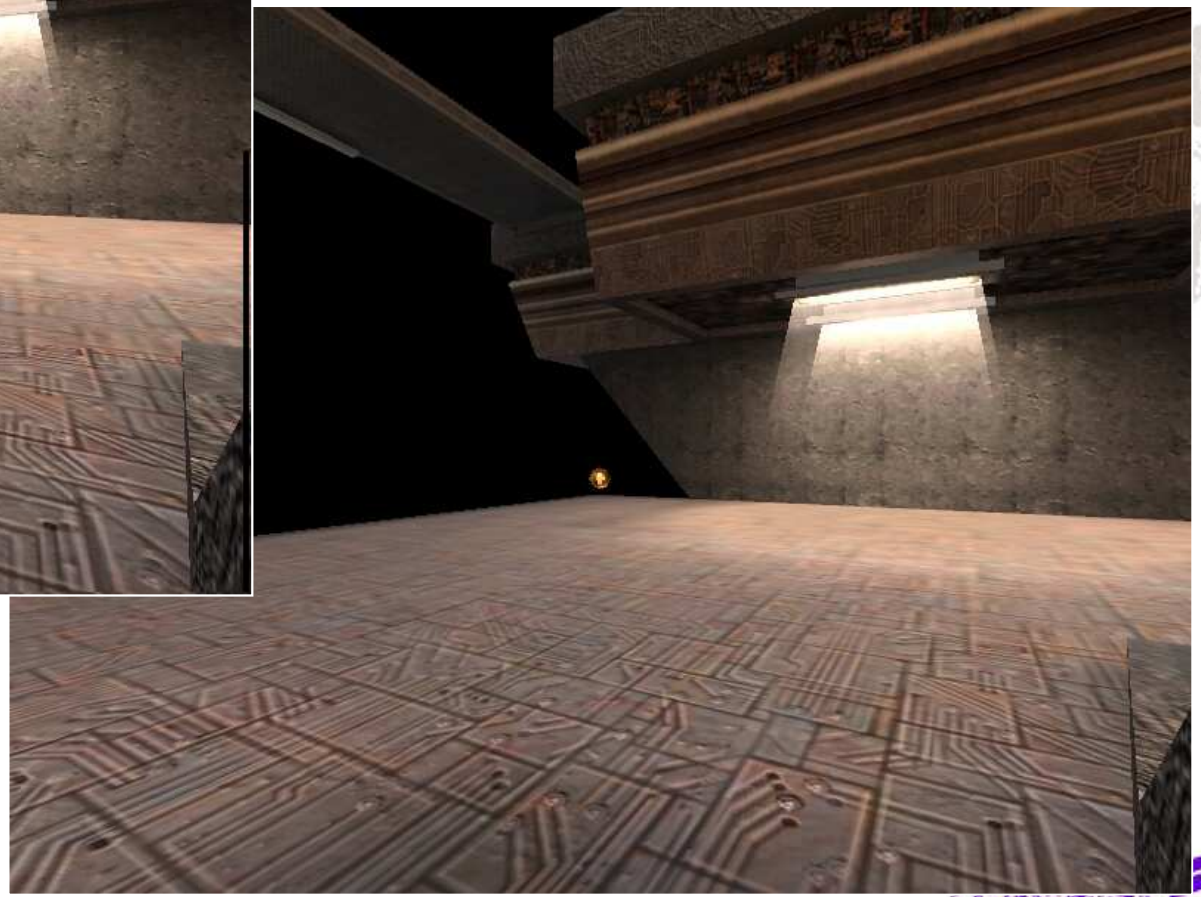
$d = 0$



Mipmap LODs

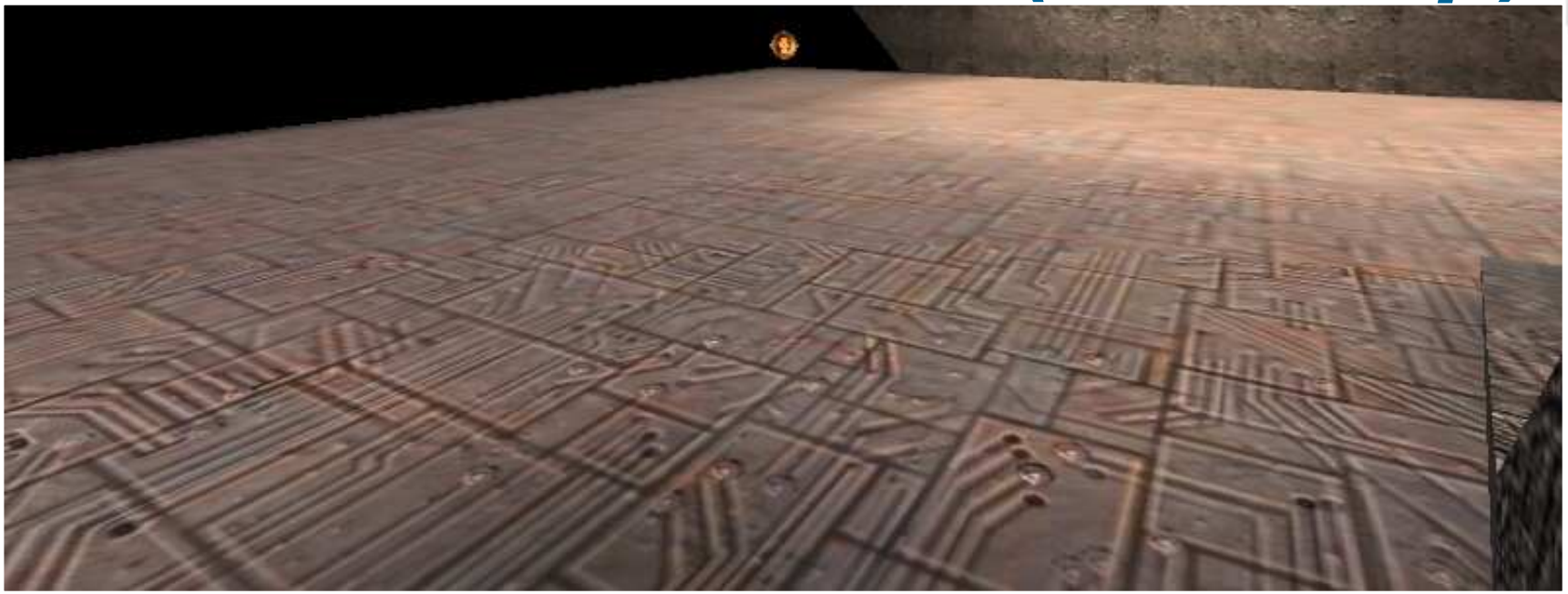
# Bi-linear vs. Tri-linear

Bi-linear filtering

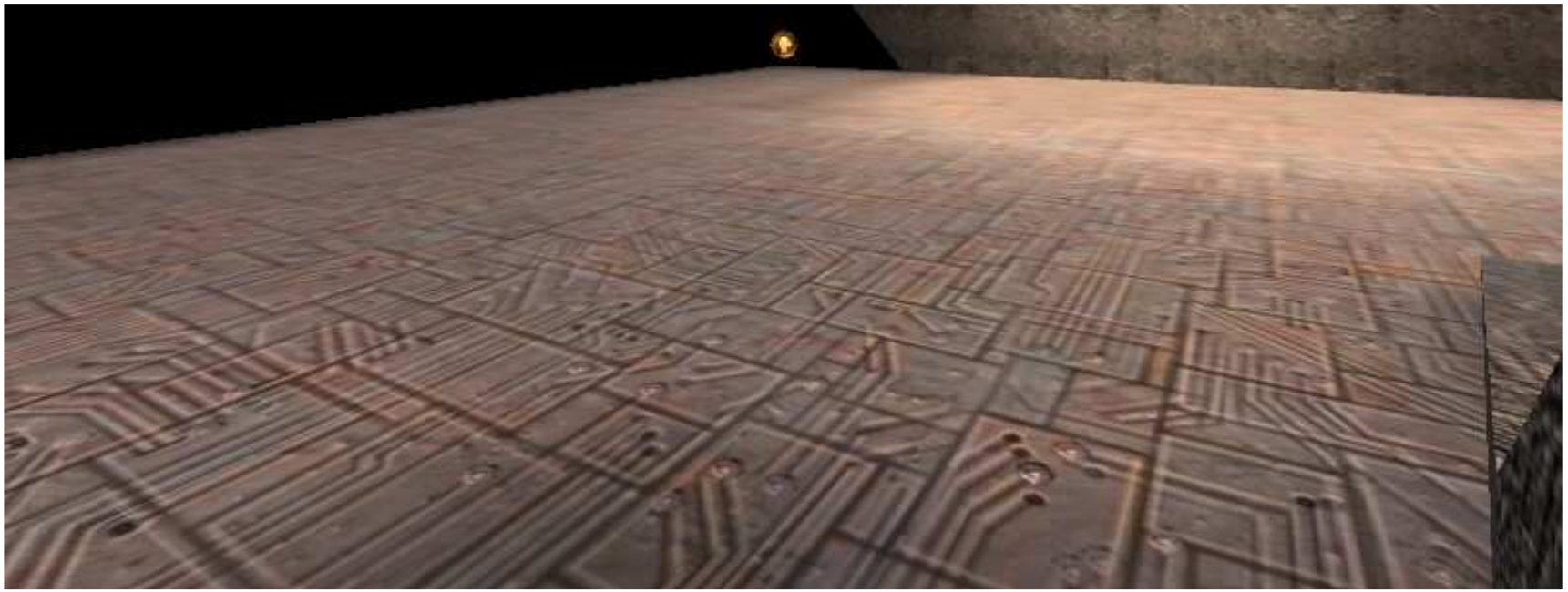


Tri-linear filtering

# Bi-linear vs. Tri-linear (Close-up)

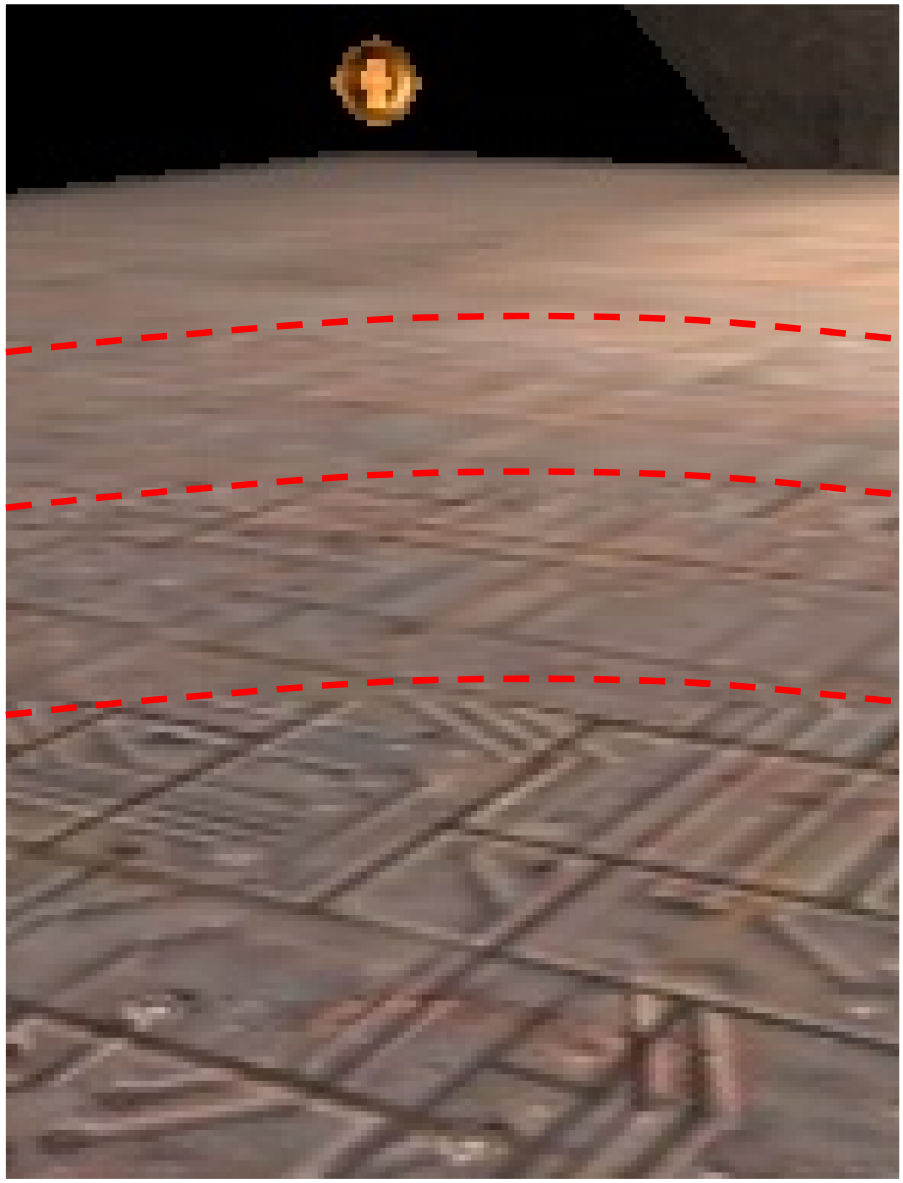


Bi-linear filtering

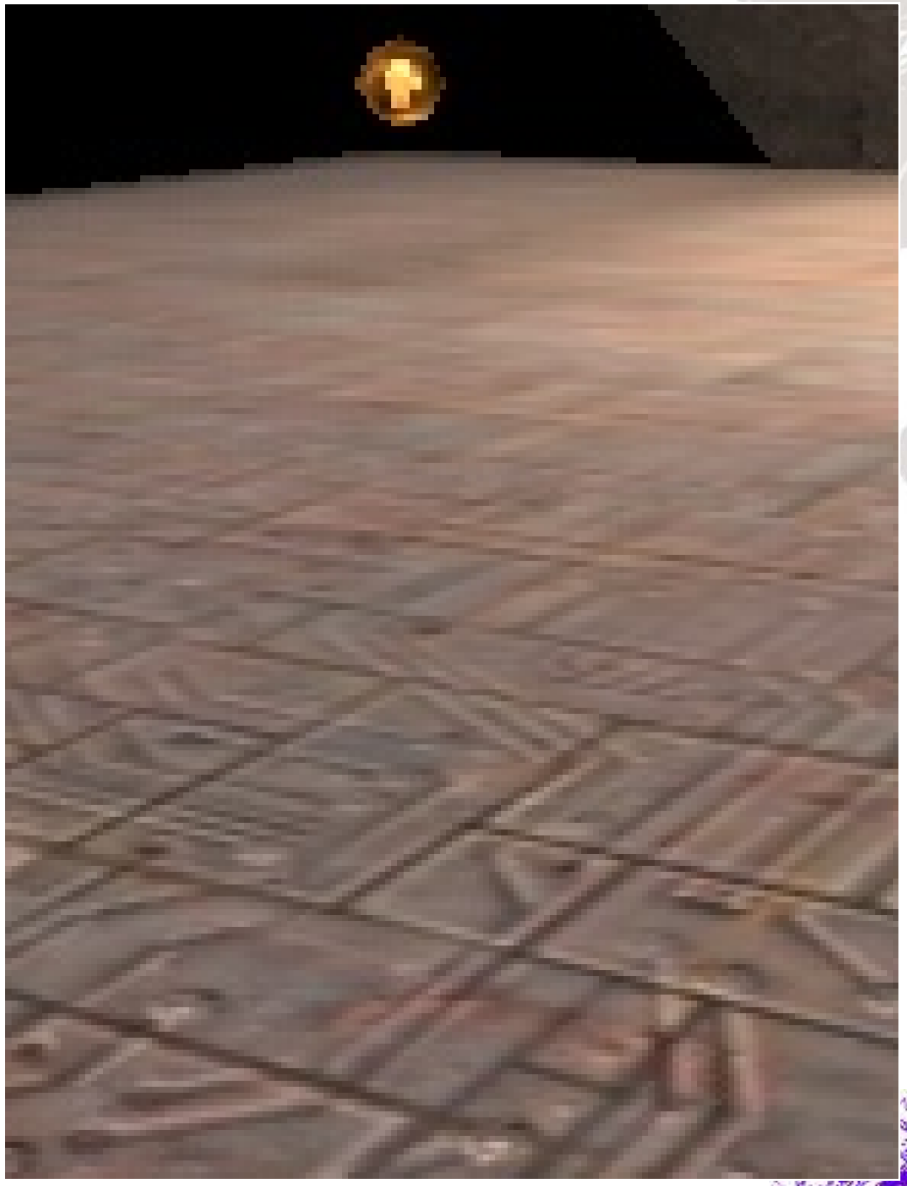


Tri-linear filtering

# Bi-linear vs. Tri-linear (Close-up more)



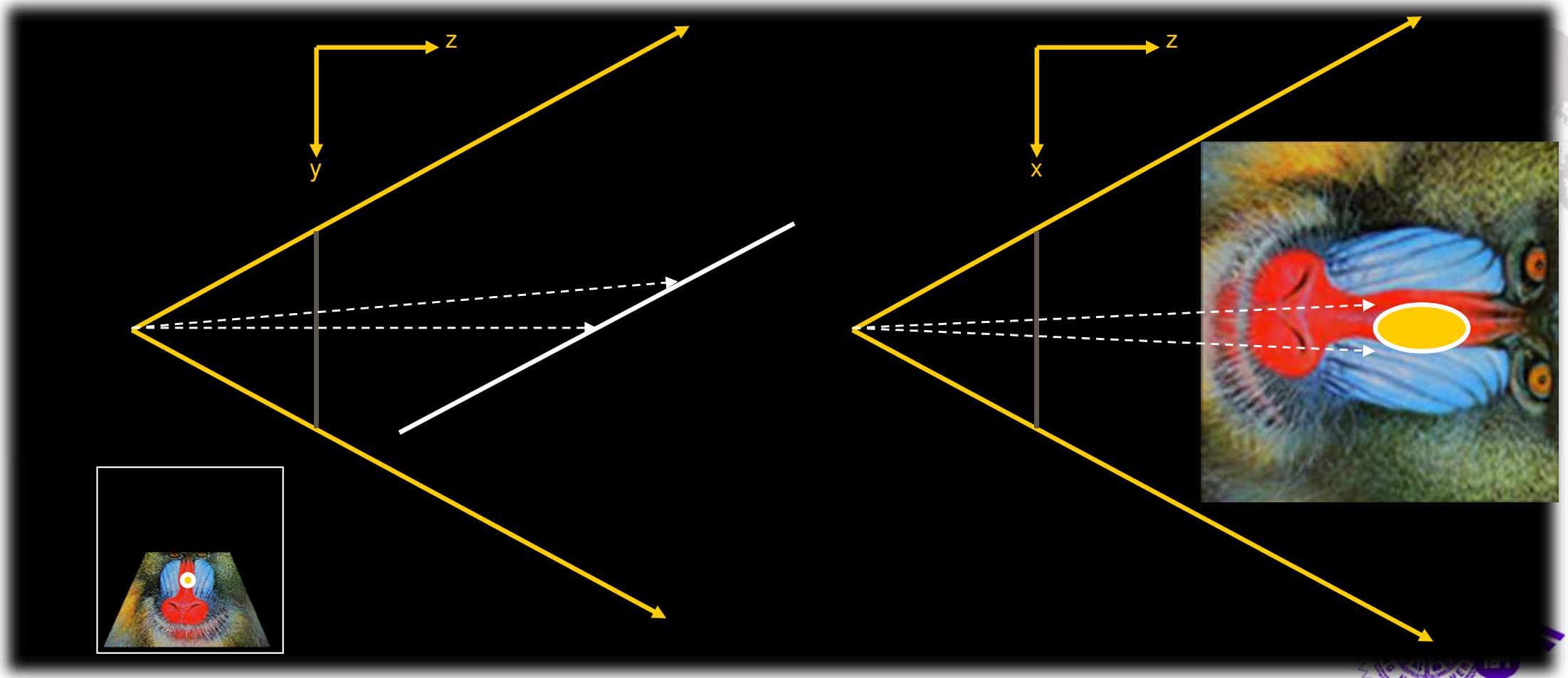
Bi-linear filtering



Tri-linear filtering

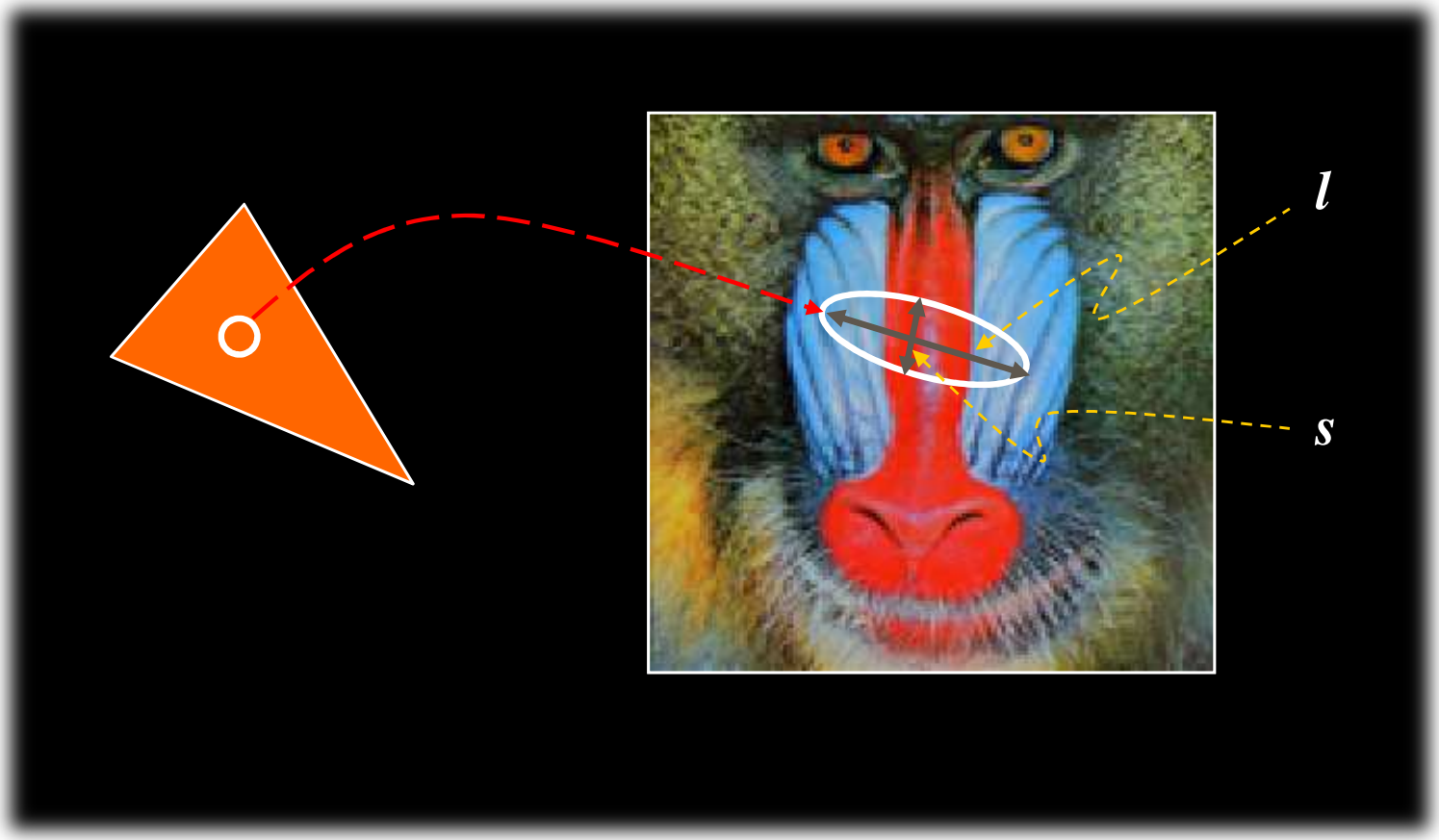
# Texture Mapped Region of a Pixel

- Texture mapped region of a projected pixel is not always a square



# Anisotropic Texture Filtering

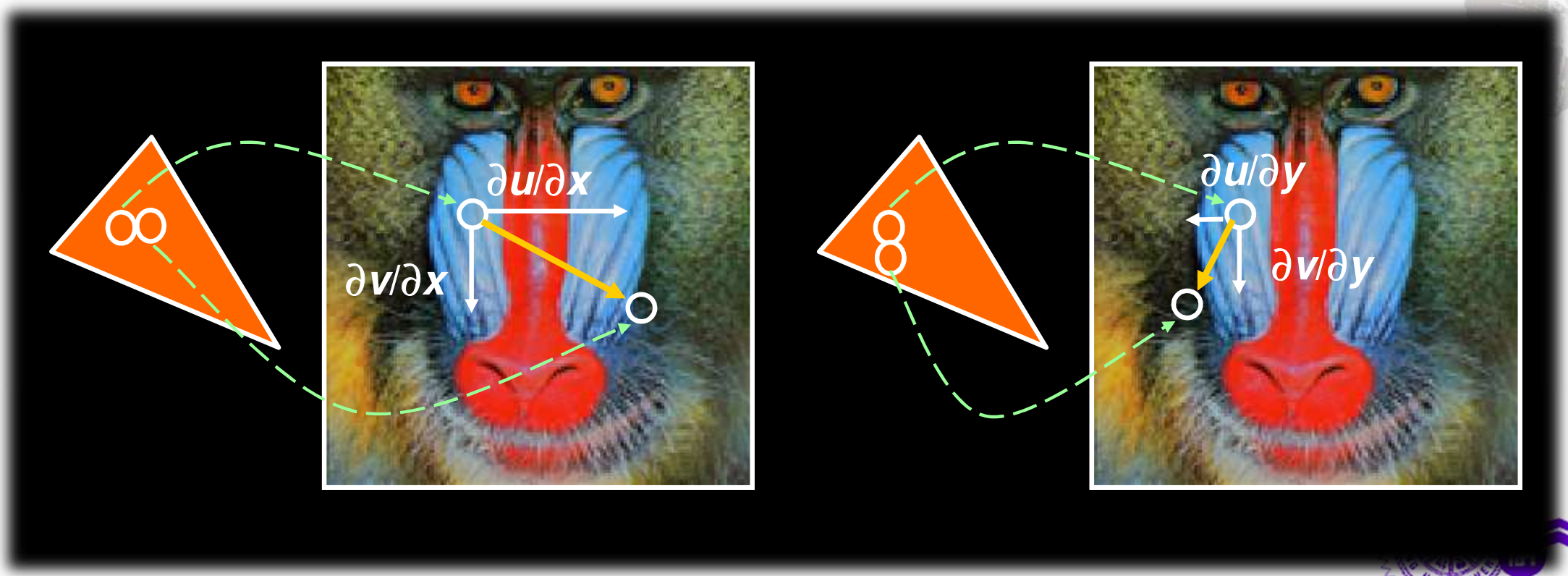
- Degree of Anisotropy
  - The ratio of $l/s$



# Anisotropic Texture Filtering

- **Approximation to the degree of anisotropy**

$$\text{Degree of anisotropy} = \sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2} \Big/ \sqrt{\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2}$$



# Anisotropic Texture Filtering

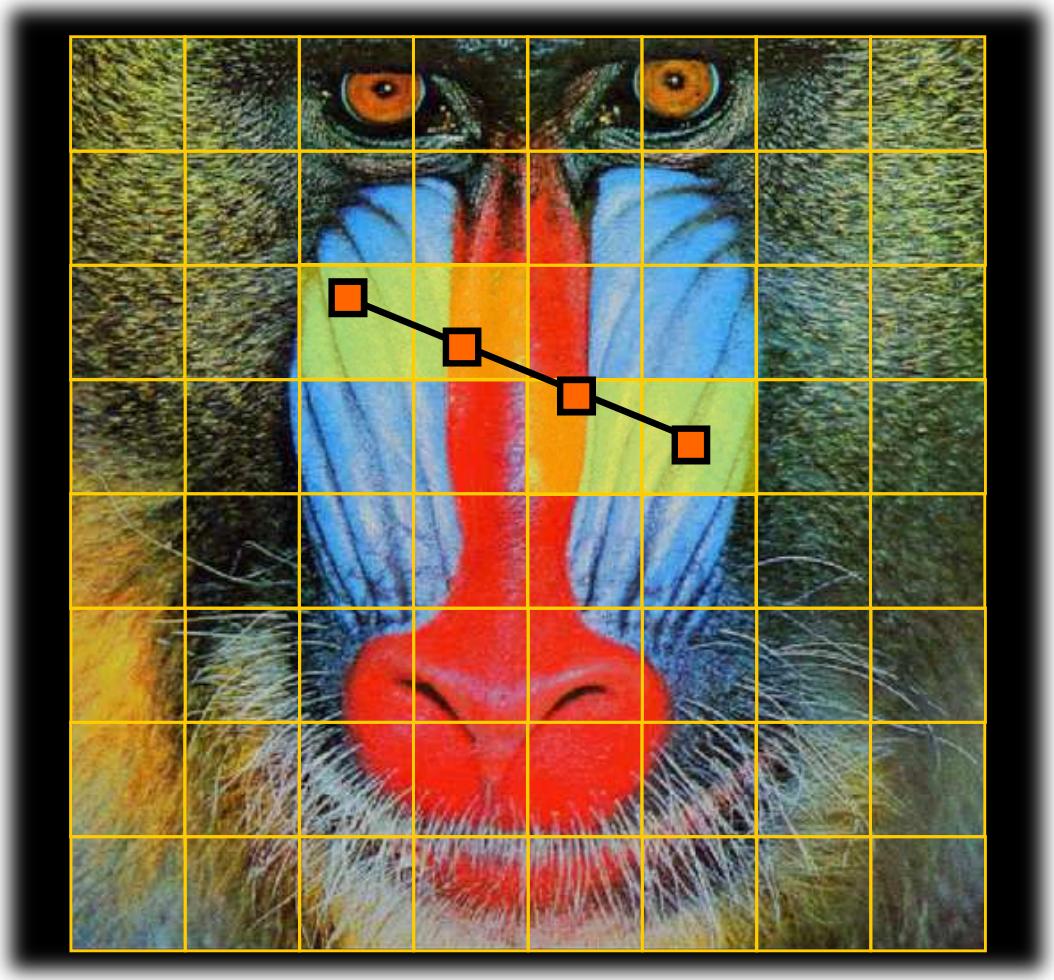
- **LOD calculation**

$$LOD = \log_2\left(\min\left\{\sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2}, \sqrt{\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2}\right\}\right)$$

- **Keep the details for smaller gradient**
- **Filter more along the larger gradient**

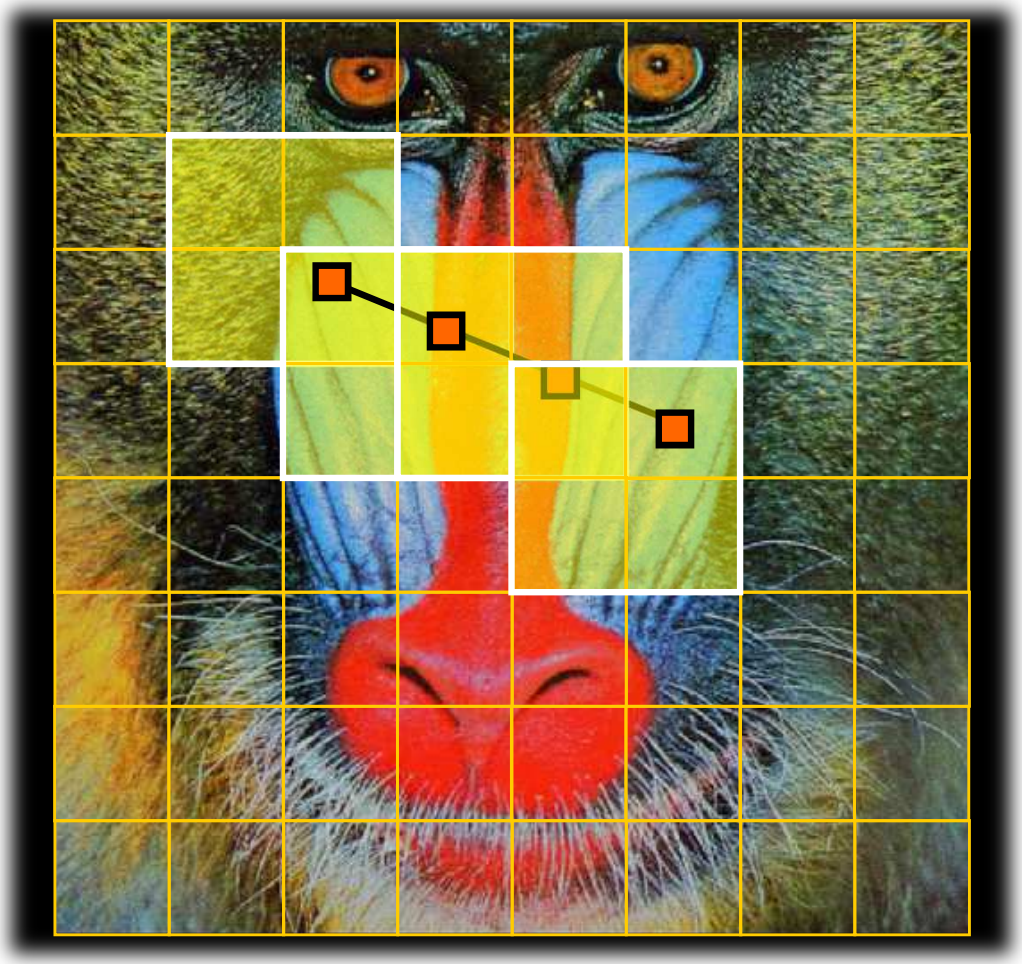
# Anisotropic Texture Filtering

- Nearest Filtering (Point Sampling)
  - Anisotropy = 4



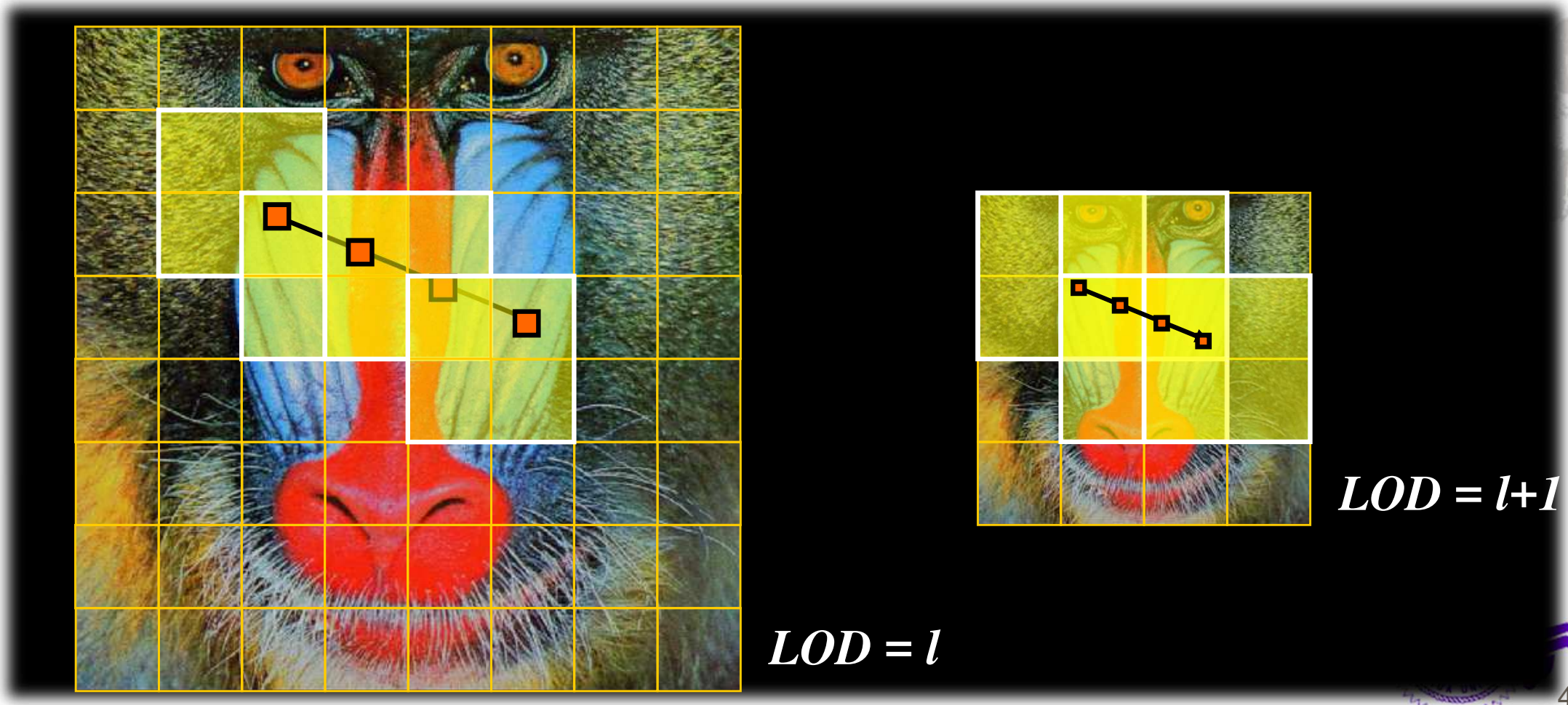
# Anisotropic Texture Filtering

- Linear Filtering (Bi-linear)
  - Anisotropy = 4

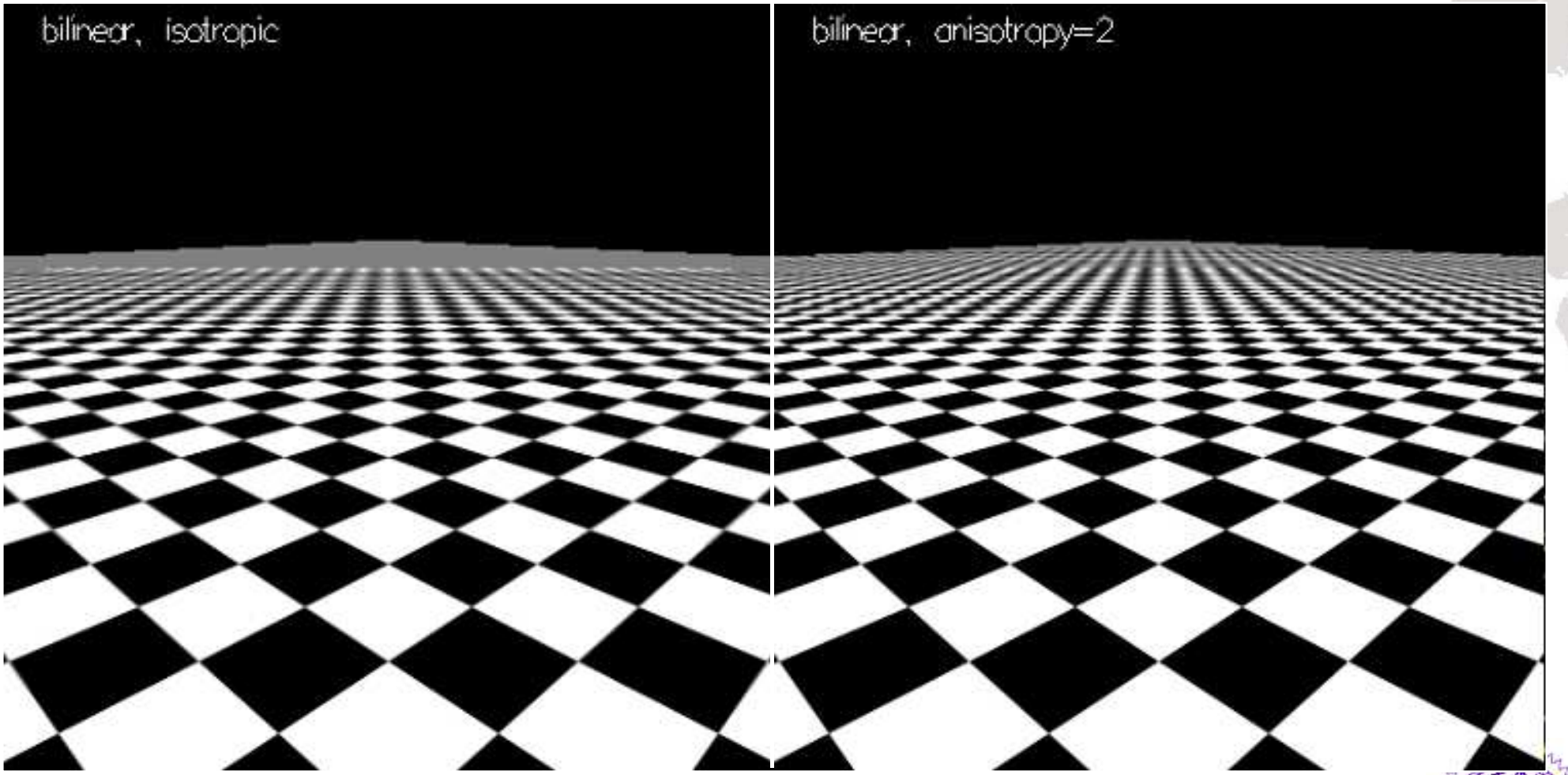


# Anisotropic Texture Filtering

- Tri-linear Filtering
  - Anisotropy = 4

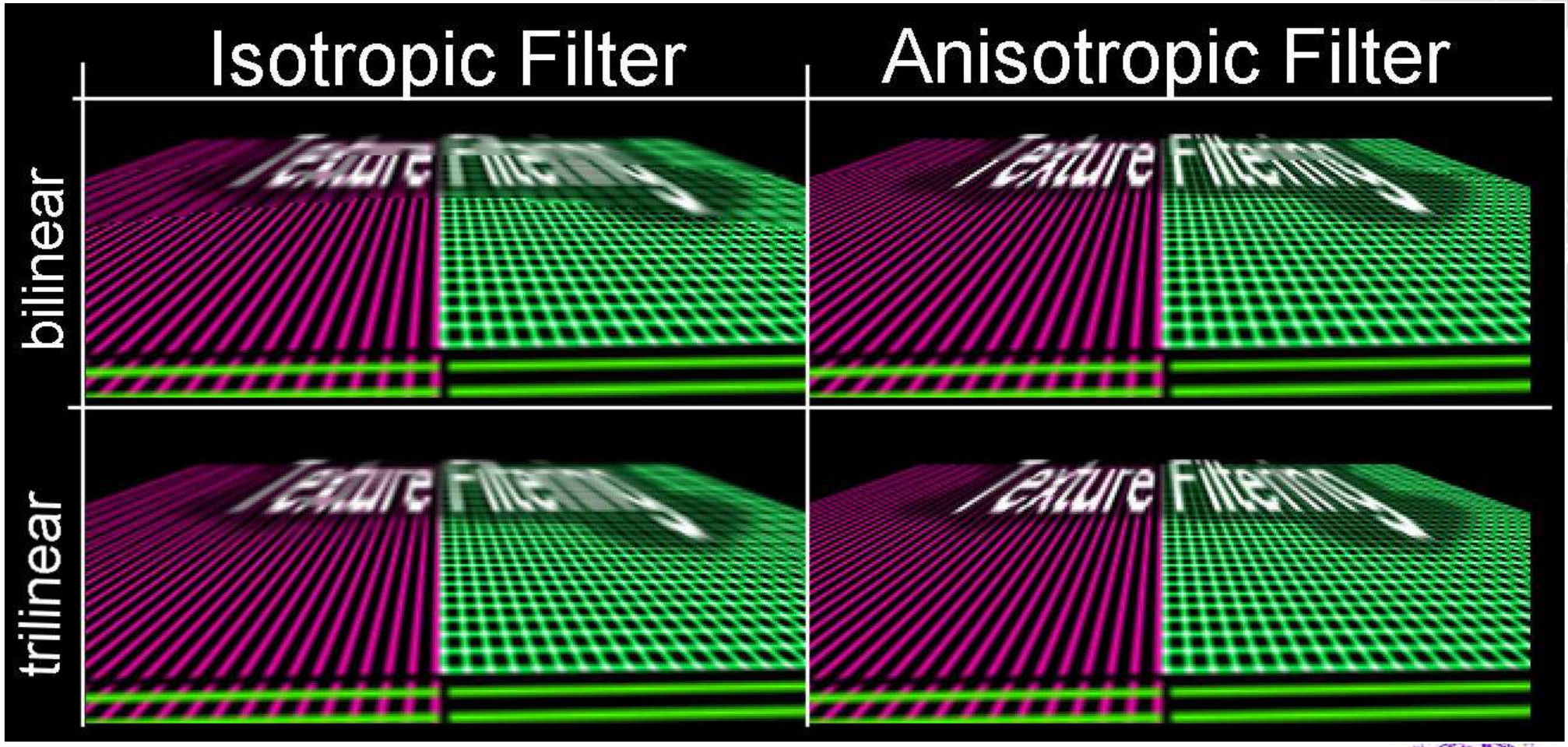


# Anisotropic Texture Filtering



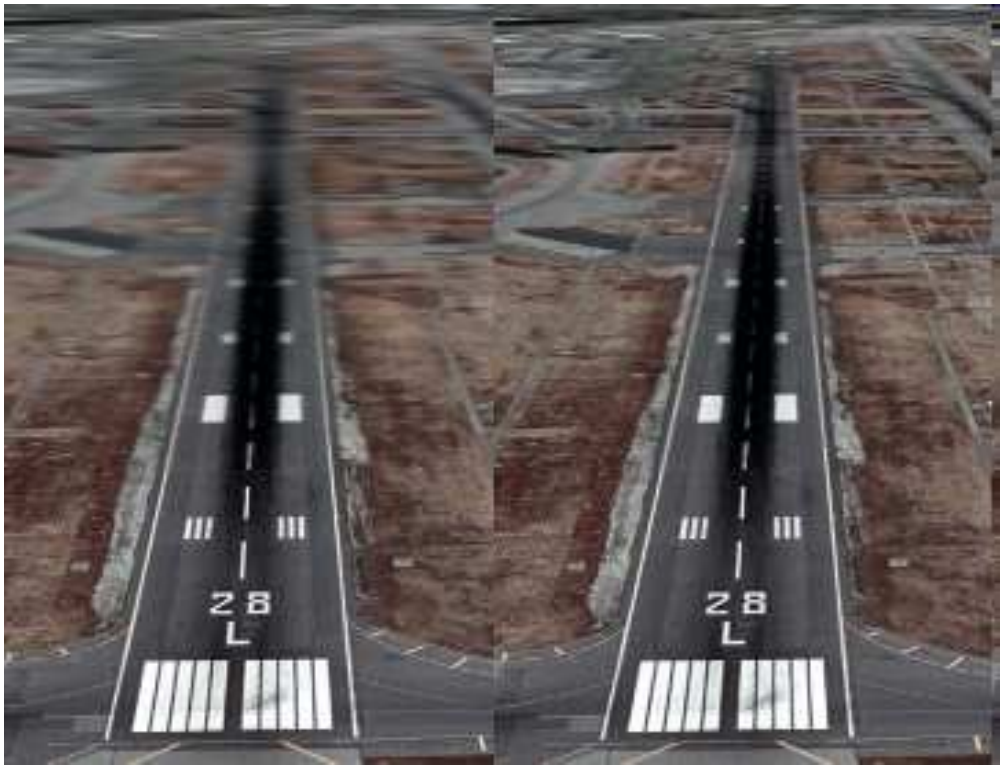
# Anisotropic Texture Filtering

# Anisotropic Texture Filtering



# Advantage of Anisotropic Filtering

- **Reducing blur and preserving surface detail at extreme viewing angles**

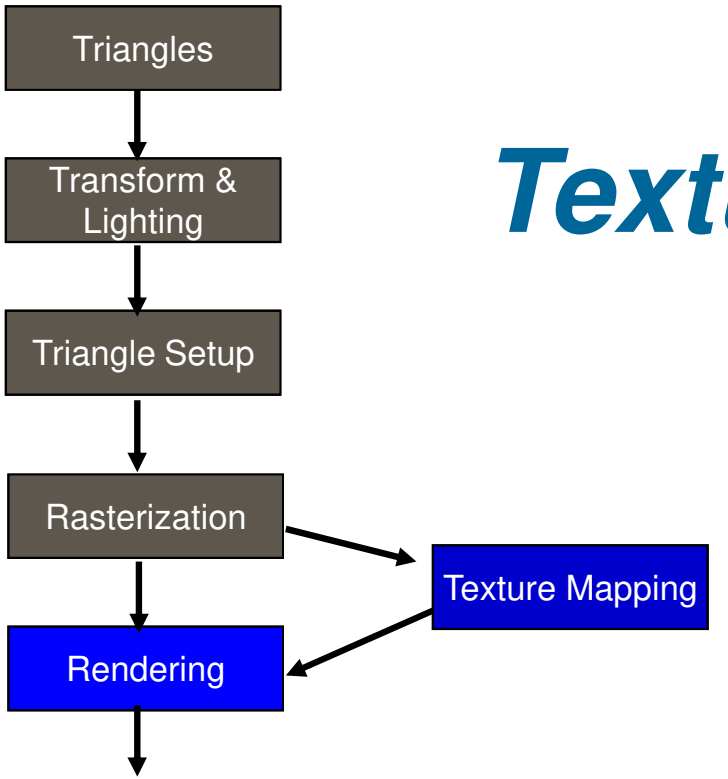



Mipmapping Anisotropic Filtering

# *Disadvantage of Anisotropic Filtering*

- **Require more texels to read for anisotropic filtering if the degree of anisotropy is high**
  - **Most hardware implementations provide the setting of maximum degree of anisotropy to limit the number of texels read so that the performance impact is managed**

# Conventional 3D Graphics Pipeline



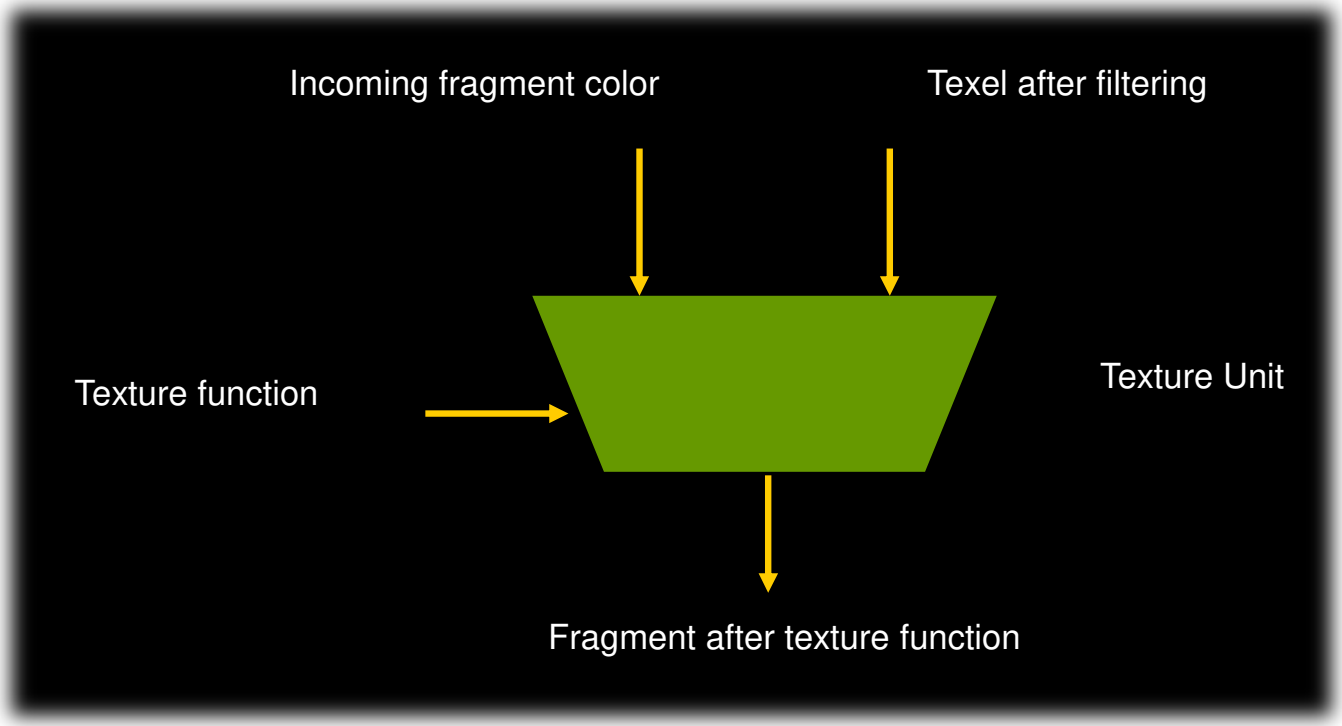

Conventional 3D Graphics Pipeline

## Texture Mapping (B)

***Advanced Texture Mapping***

# Texture Functions

- Texture functions are used to define how a texel is to combine with a fragment being processed



# Texture Functions

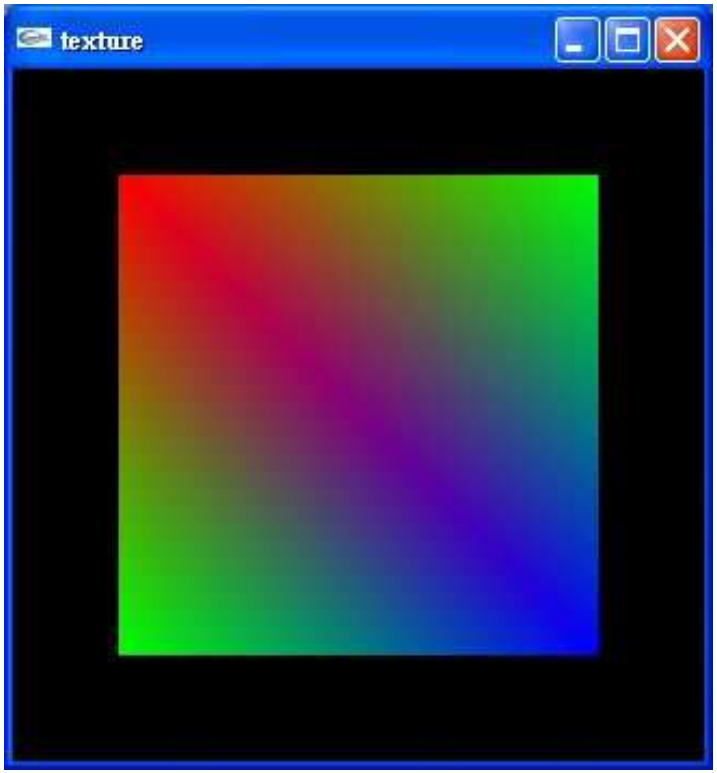
- **Eg. OpenGL texture functions**

| Base Internal Format | GL_REPLACE Function | GL_MODULATE Function | GL_DECAL Function |
|---|---|---|---|
| GL_ALPHA | $C = C_f$<br>$A = A_s$ | $C = C_f$<br>$A = A_f A_s$ | undefined |
| GL_LUMINANCE | $C = C_s$<br>$A = A_f$ | $C = C_f C_s$<br>$A = A_f$ | undefined |
| GL_LUMINANCE_ALPHA | $C = C_s$<br>$A = A_s$ | $C = C_f C_s$<br>$A = A_f A_s$ | undefined |
| GL_INTENSITY | $C = C_s$<br>$A = C_s$ | $C = C_f C_s$<br>$A = A_f C_s$ | undefined |
| GL_RGB | $C = C_s$<br>$A = A_f$ | $C = C_f C_s$<br>$A = A_f$ | $C = C_s$<br>$A = A_f$ |
| GL_RGBA | $C = C_s$<br>$A = A_s$ | $C = C_f C_s$<br>$A = A_f A_s$ | $C = C_f(1 - A_s) + C_s A_s$<br>$A = A_f$ |

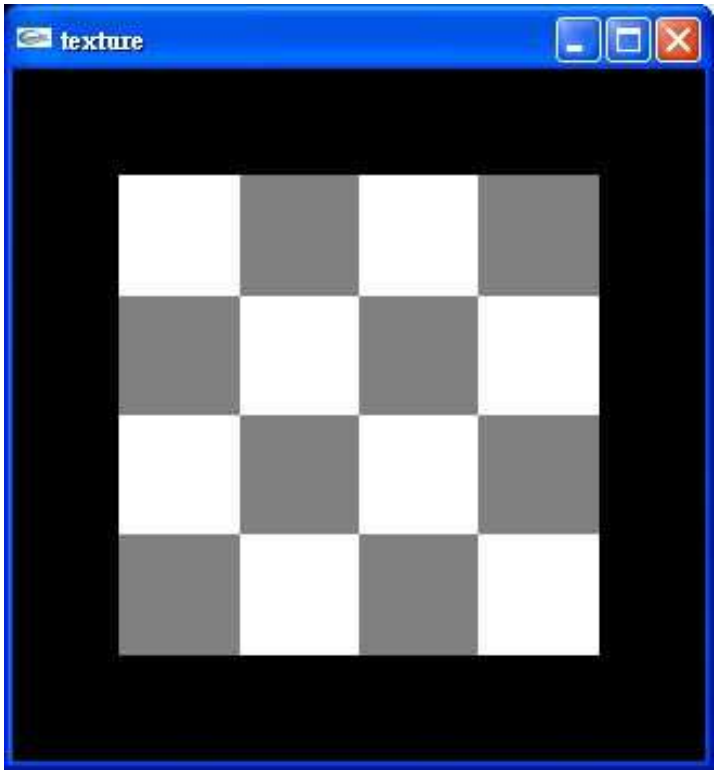
| Base Internal Format | GL_BLEND Function | GL_ADD Function |
|---|---|---|
| GL_ALPHA | $C = C_f$<br>$A = A_f A_s$ | $C = C_f$<br>$A = A_f A_s$ |
| GL_LUMINANCE | $C = C_f(1 - C_s) + C_c C_s$<br>$A = A_f$ | $C = C_f + C_s$<br>$A = A_f$ |
| GL_LUMINANCE_ALPHA | $C = C_f(1 - C_s) + C_c C_s$<br>$A = A_f A_s$ | $C = C_f + C_s$<br>$A = A_f A_s$ |
| GL_INTENSITY | $C = C_f(1 - C_s) + C_c C_s$<br>$A = A_f(1 - A_s) + A_c A_s$ | $C = C_f + C_s$<br>$A = A_f + A_s$ |
| GL_RGB | $C = C_f(1 - C_s) + C_c C_s$<br>$A = A_f$ | $C = C_f + C_s$<br>$A = A_f$ |
| GL_RGBA | $C = C_f(1 - C_s) + C_c C_s$<br>$A = A_f A_s$ | $C = C_f + C_s$<br>$A = A_f A_s$ |

# Texture Functions

- **Examples**




Fragment color



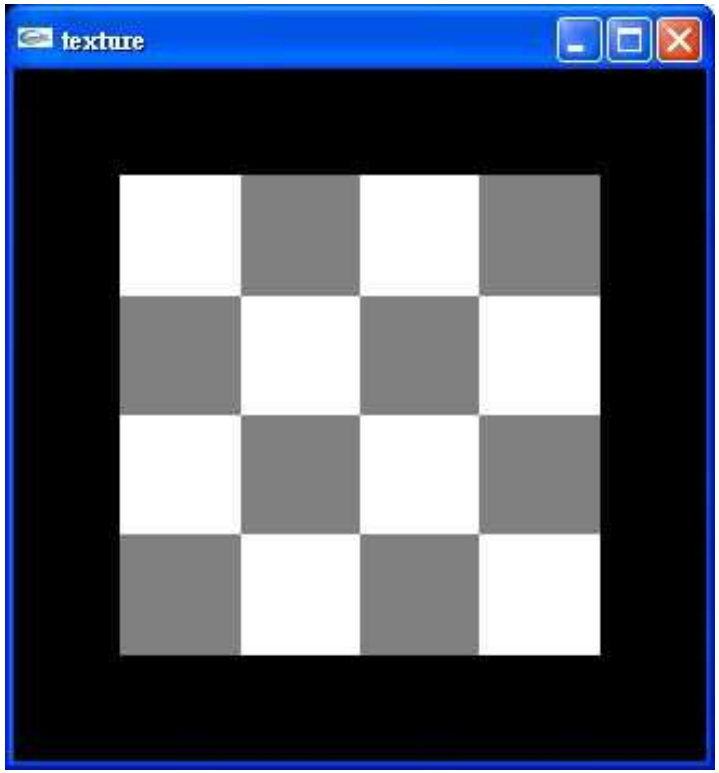

Texture
Alpha: 1.0 for white area
0.5 for gray area



Constant color

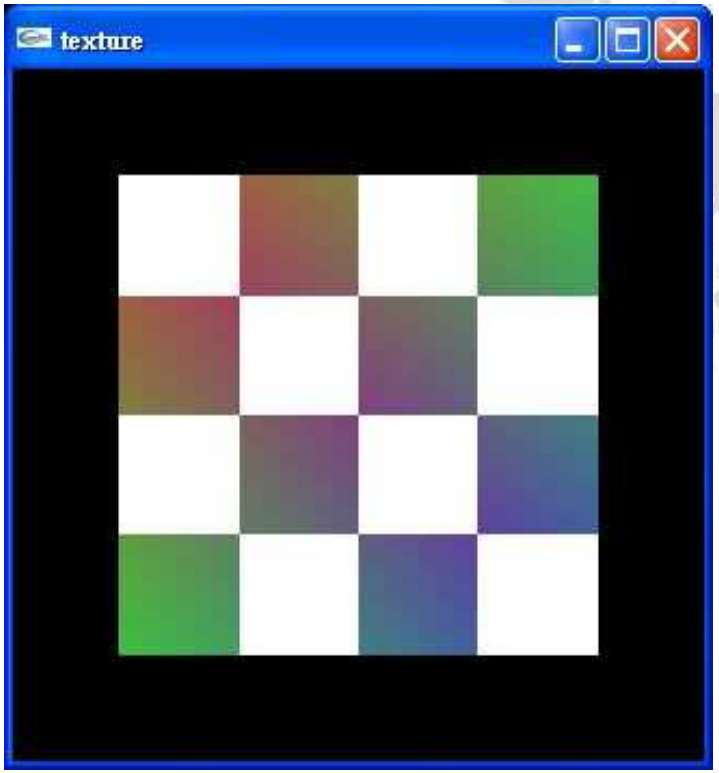
# Texture Functions

- Examples




Replace

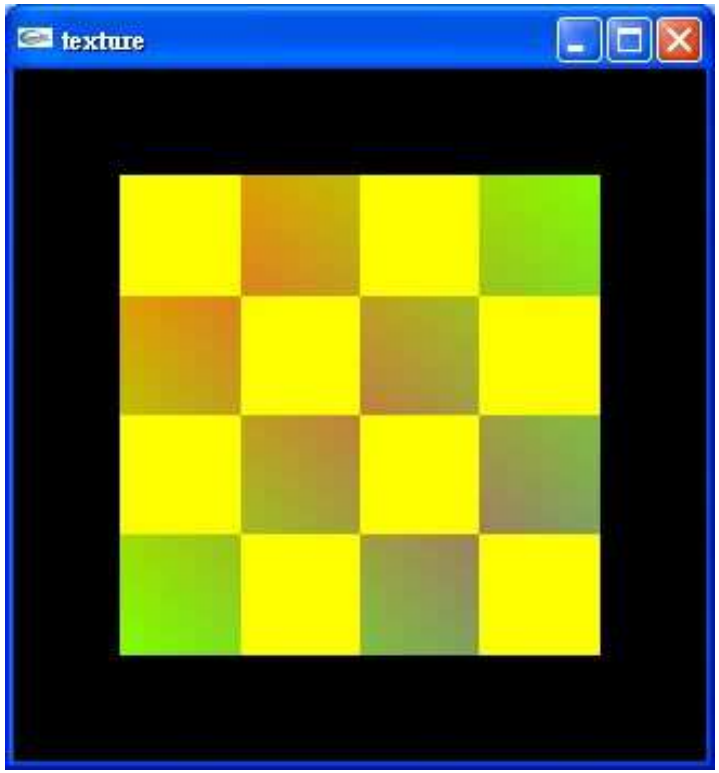

Modulate




Decal

# Texture Functions

- Examples

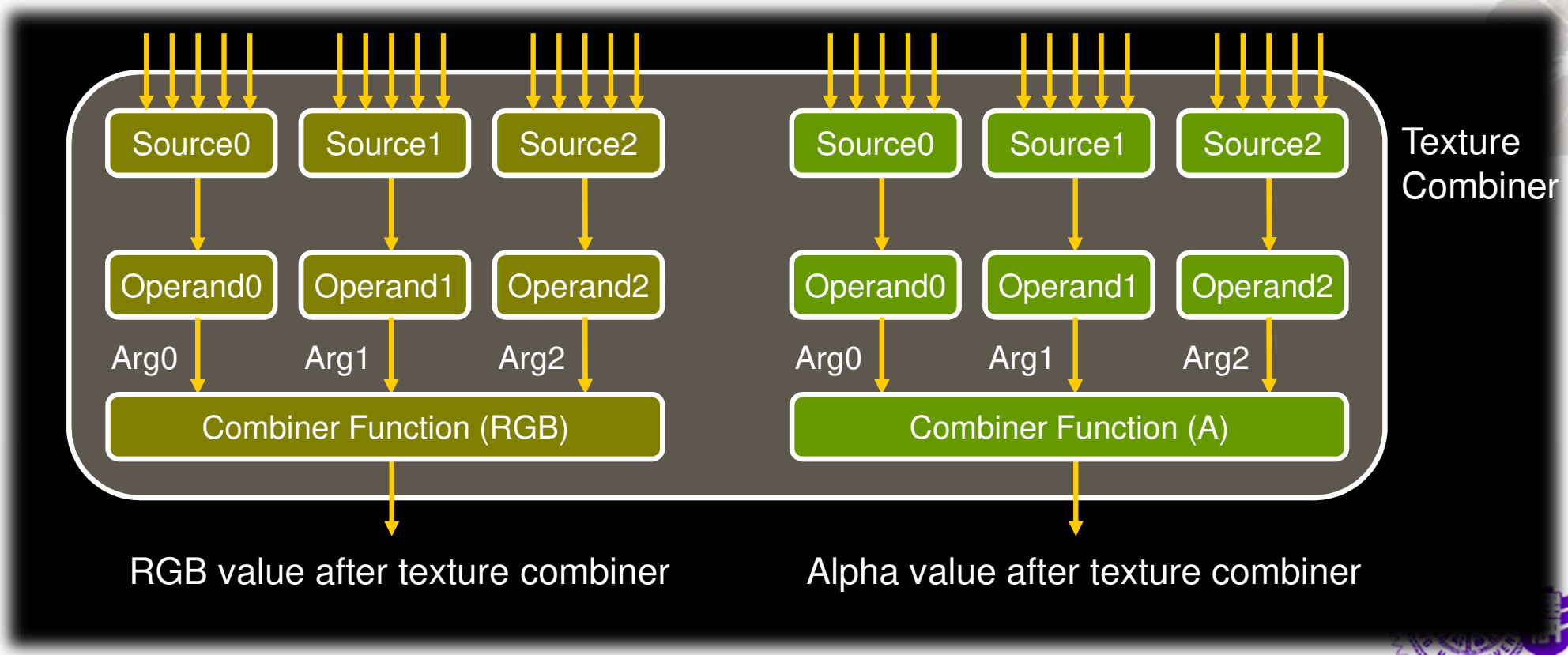



Blend


Add

# Texture Combiner Functions

- Texture combiner functions are more powerful texture functions with flexible combinations of arguments and operations



# Texture Combiner Functions

- **Eg. OpenGL texture combiner functions**

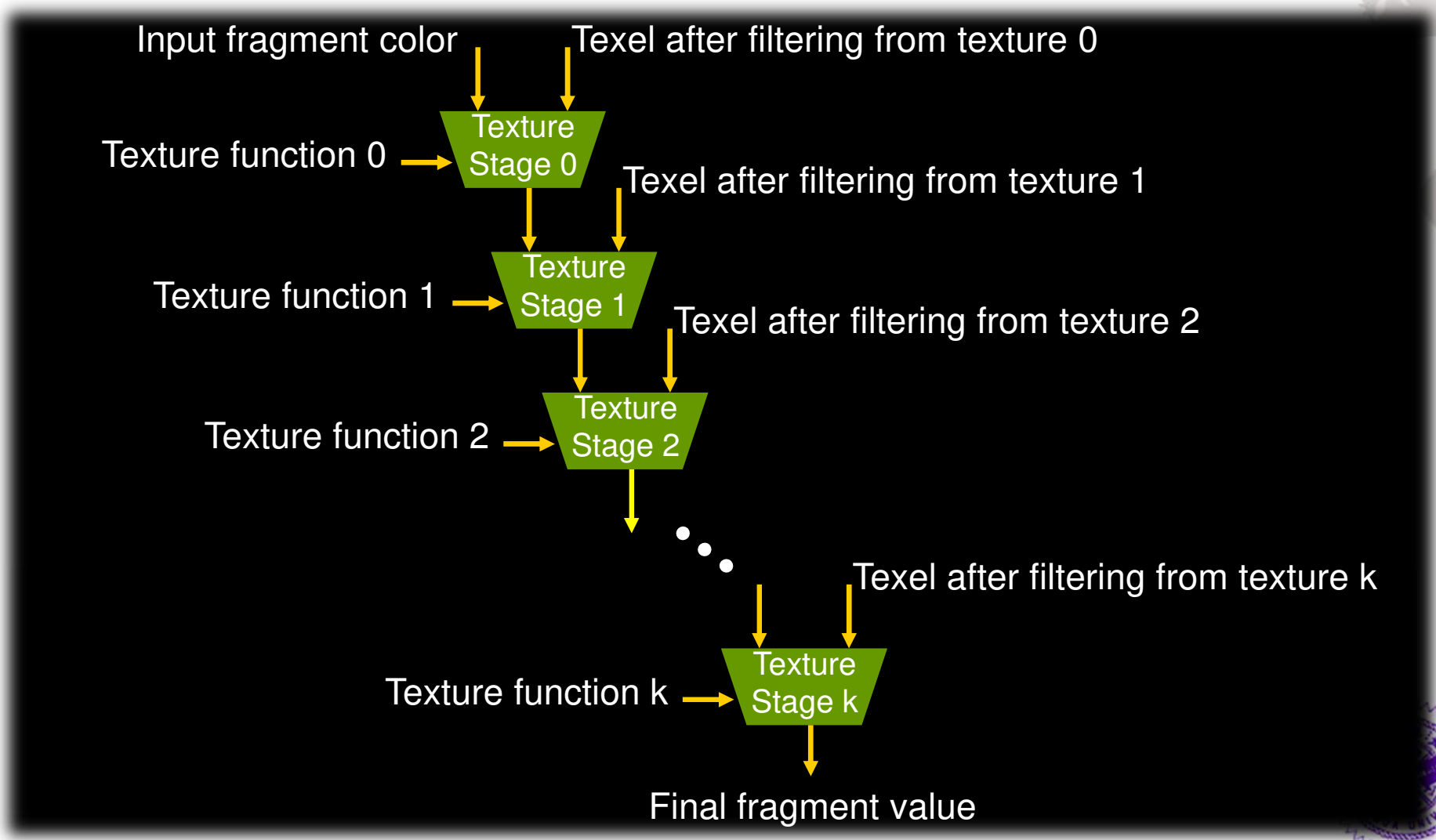
| glTexEnv param | Combiner Function |
|---|---|
| GL_REPLACE | $Arg0$ |
| GL_MODULATE (default) | $Arg0 * Arg1$ |
| GL_ADD | $Arg0 + Arg1$ |
| GL_ADD_SIGNED | $Arg0 + Arg1 - 0.5$ |
| GL_INTERPOLATE | $Arg0 * Arg2 + Arg1 * (1 - Arg2)$ |
| GL_SUBTRACT | $Arg0 - Arg1$ |
| GL_DOT3_RGB<br>GL_DOT3_RGBA | $4 * ((Arg0_r - 0.5) * (Arg1_r - 0.5) +$<br>$(Arg0_g - 0.5) * (Arg1_g - 0.5) +$<br>$(Arg0_b - 0.5) * (Arg1_b - 0.5))$ |

# Texture Combiner Functions

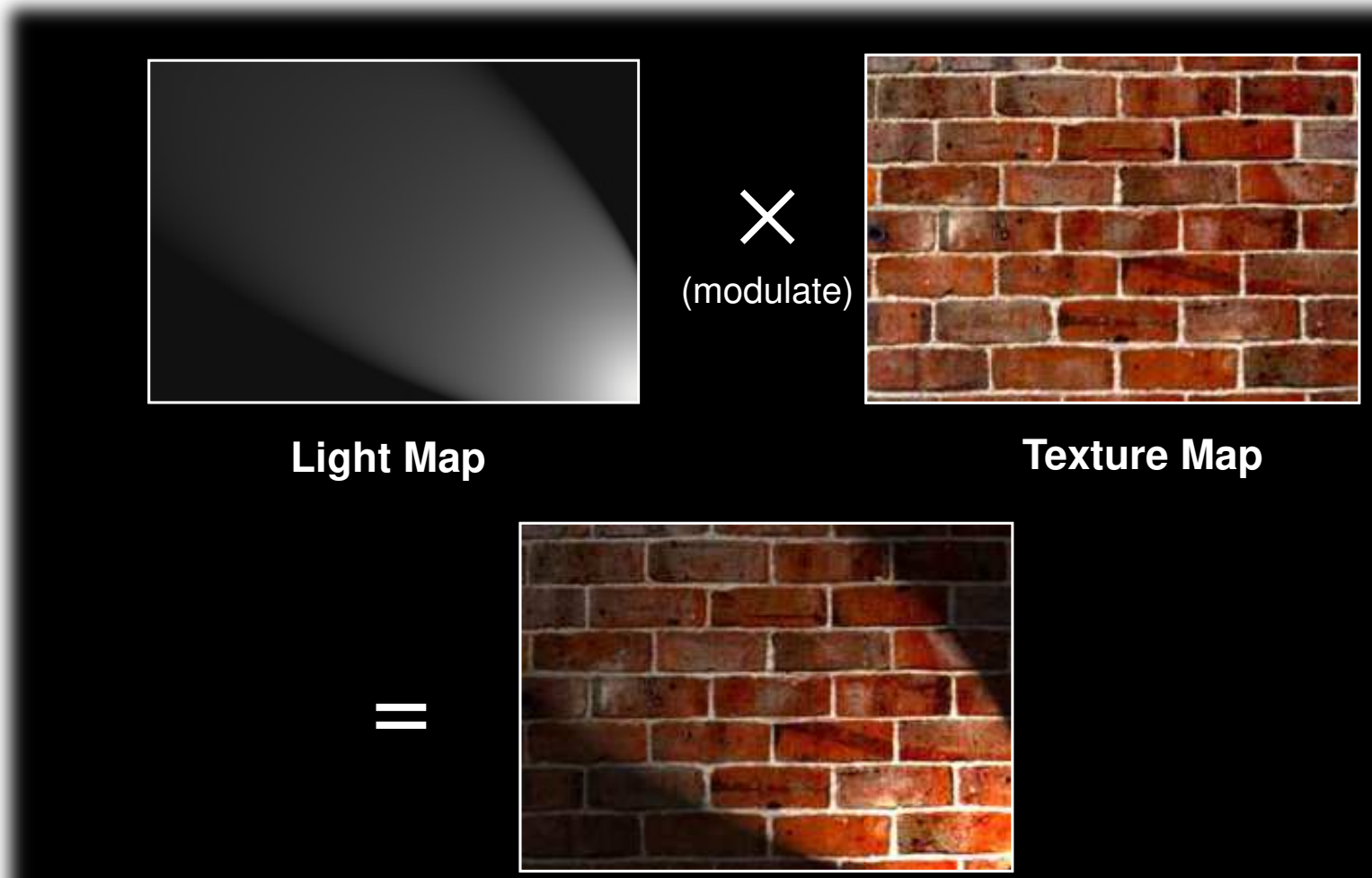
- Eg. OpenGL texture combiner functions
  - GL_SRC*i*_RGB or GL_SRC*i*_ALPHA could be GL_TEXTURE, GL_TEXTURE*n*, GL_CONSTANT, GL_PRIMARY_COLOR, or GL_PREVIOUS
  - GL_OPERAND*i*_RGB could be $SRC_{RGB}$, 1- $SRC_{RGB}$, $SRC_A$, or 1- $SRC_A$
  - GL_OPERAND*i*_ALPHA could be $SRC_A$, or 1- $SRC_A$

# Multi-Texturing

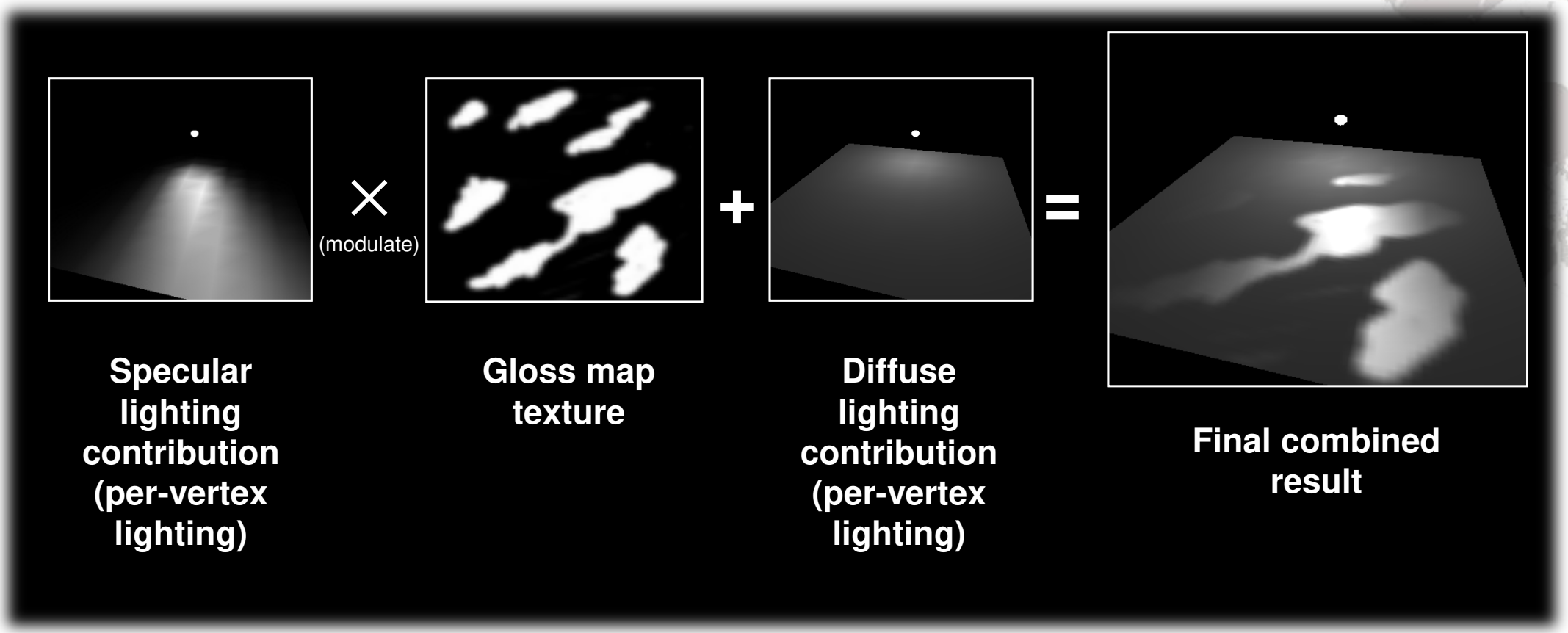
- Texture Stages



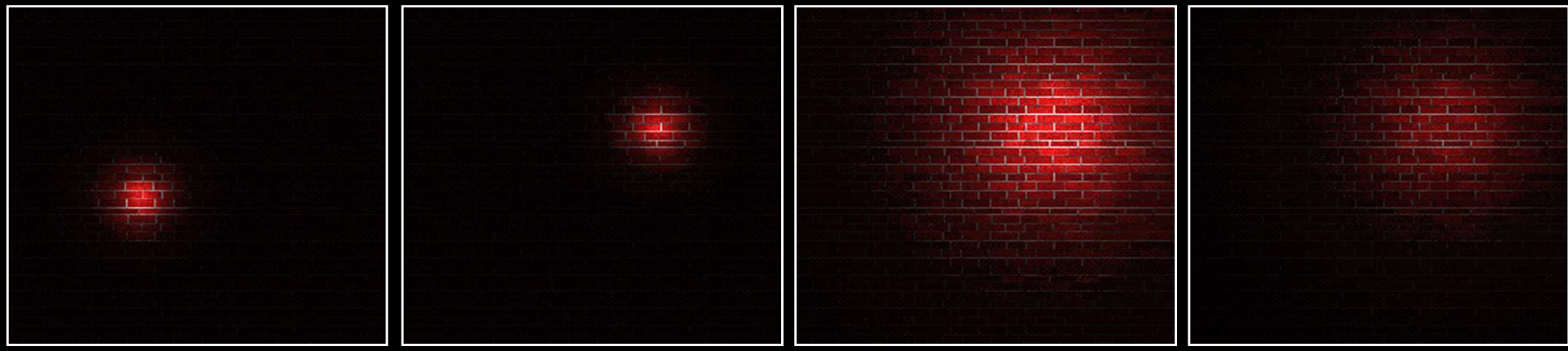
# Multi-Texturing



# Multi-texturing



# Texture Transform



Original

Translate Spotlight Texture Coordinates

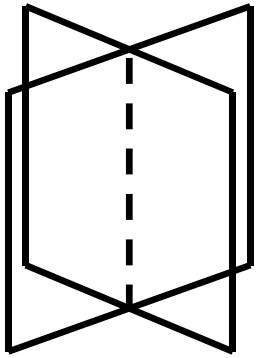
Scale Spotlight Texture Coordinates

Change Base Polygon Intensity

Use texture matrix to perform spotlight texture coordinates transformations.

# Billboard

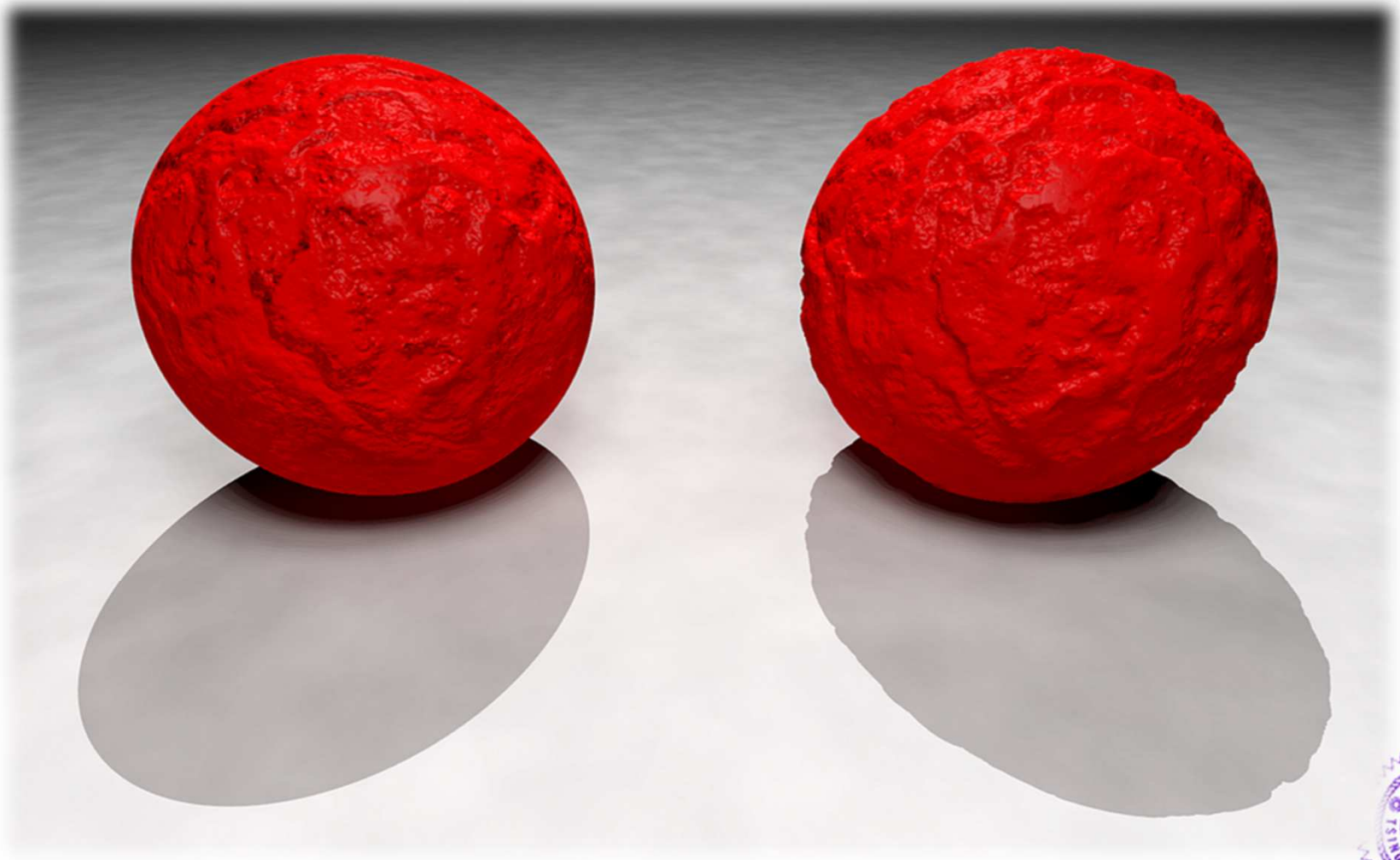
- Model details without increasing the complexity of geometric modeling



# Bump Mapping

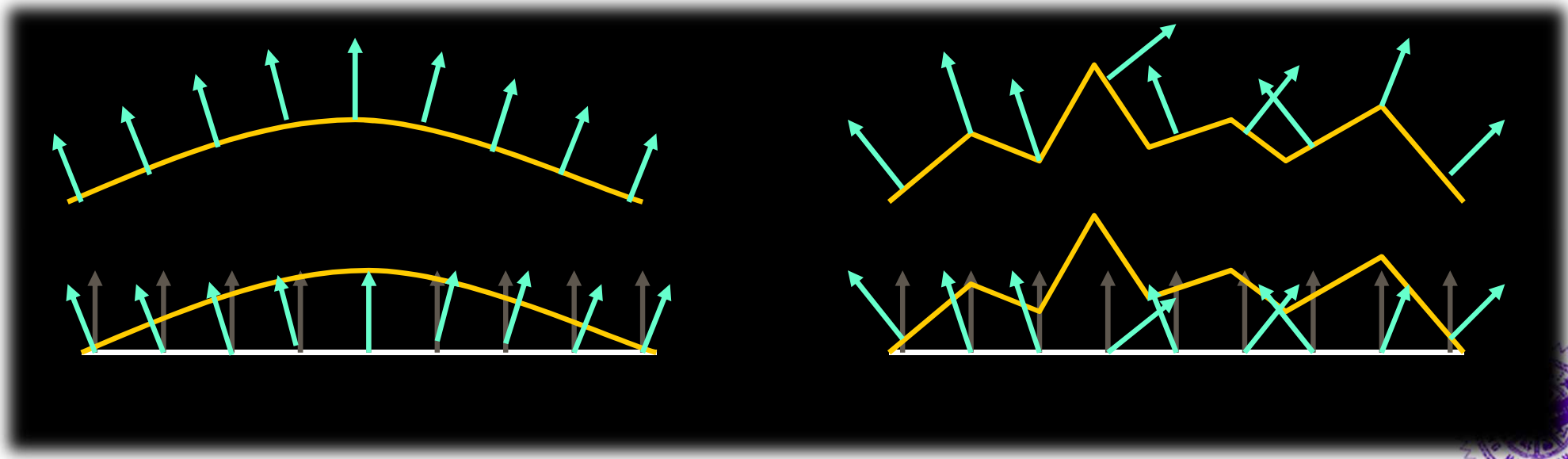
# Bump Mapping

- Bump mapping vs. displacement mapping



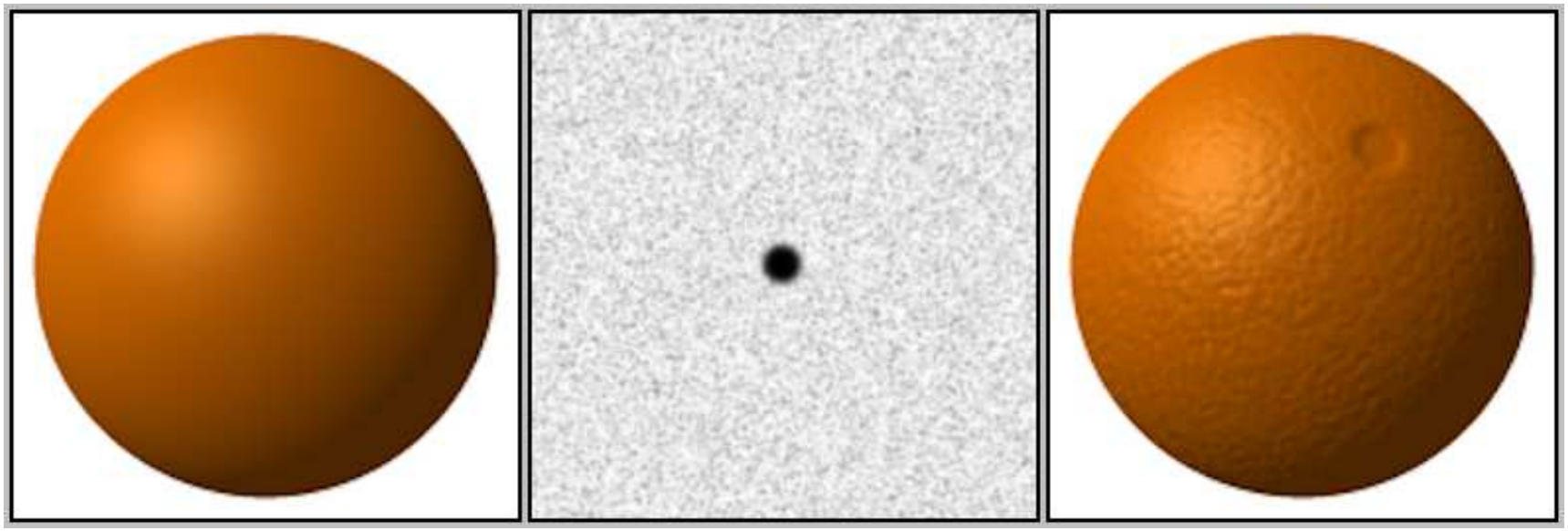
# Bump Mapping

- Apply texture map as the perturbation vectors to the pixel normals
- Apply lighting operation on the perturbed normals to create the illusion of bumpy surface



# Bump Mapping

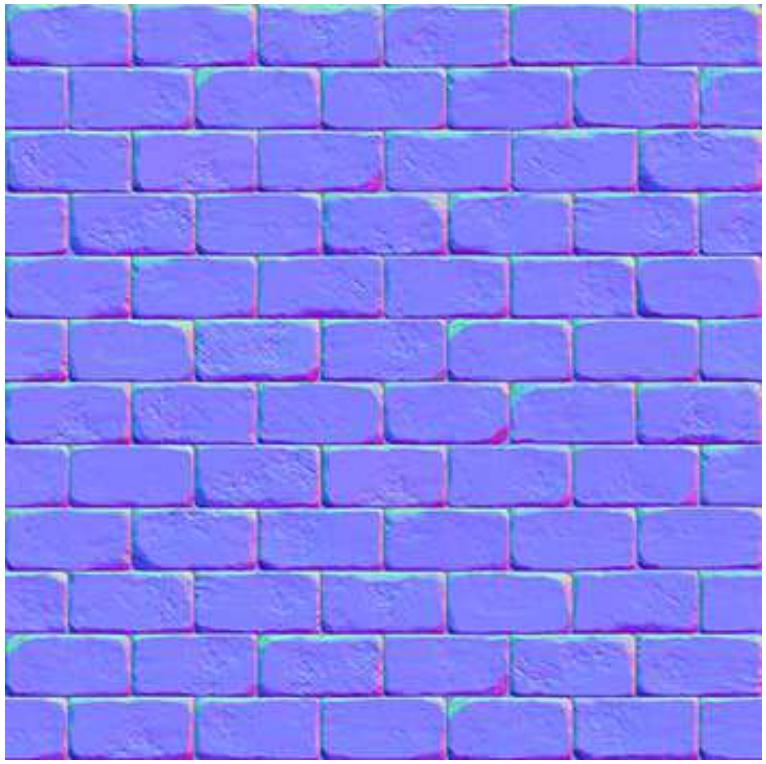
- **Using height map to alter the surface normal**



1. Look up the height in the heightmap that corresponds to the position on the surface.
2. Calculate the surface normal of the heightmap, typically using the finite difference method.
3. Combine the surface normal from step two with the true ("geometric") surface normal so that the combined normal points in a new direction.
4. Calculate the interaction of the new "bumpy" surface with lights in the scene using, for example, the Phong reflection model.

# Bump Mapping

- Using normal map to alter the surface normal



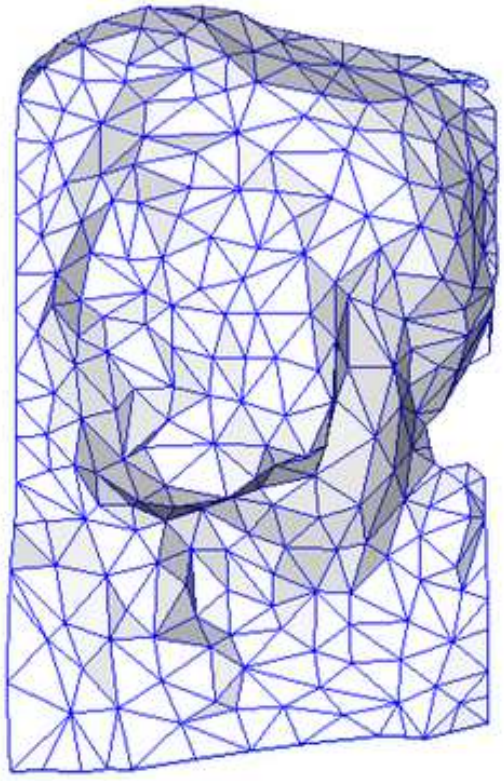
# Bump Mapping

- Using normal map to alter the surface normal

# Bump Mapping

- Using normal map to alter the surface normal

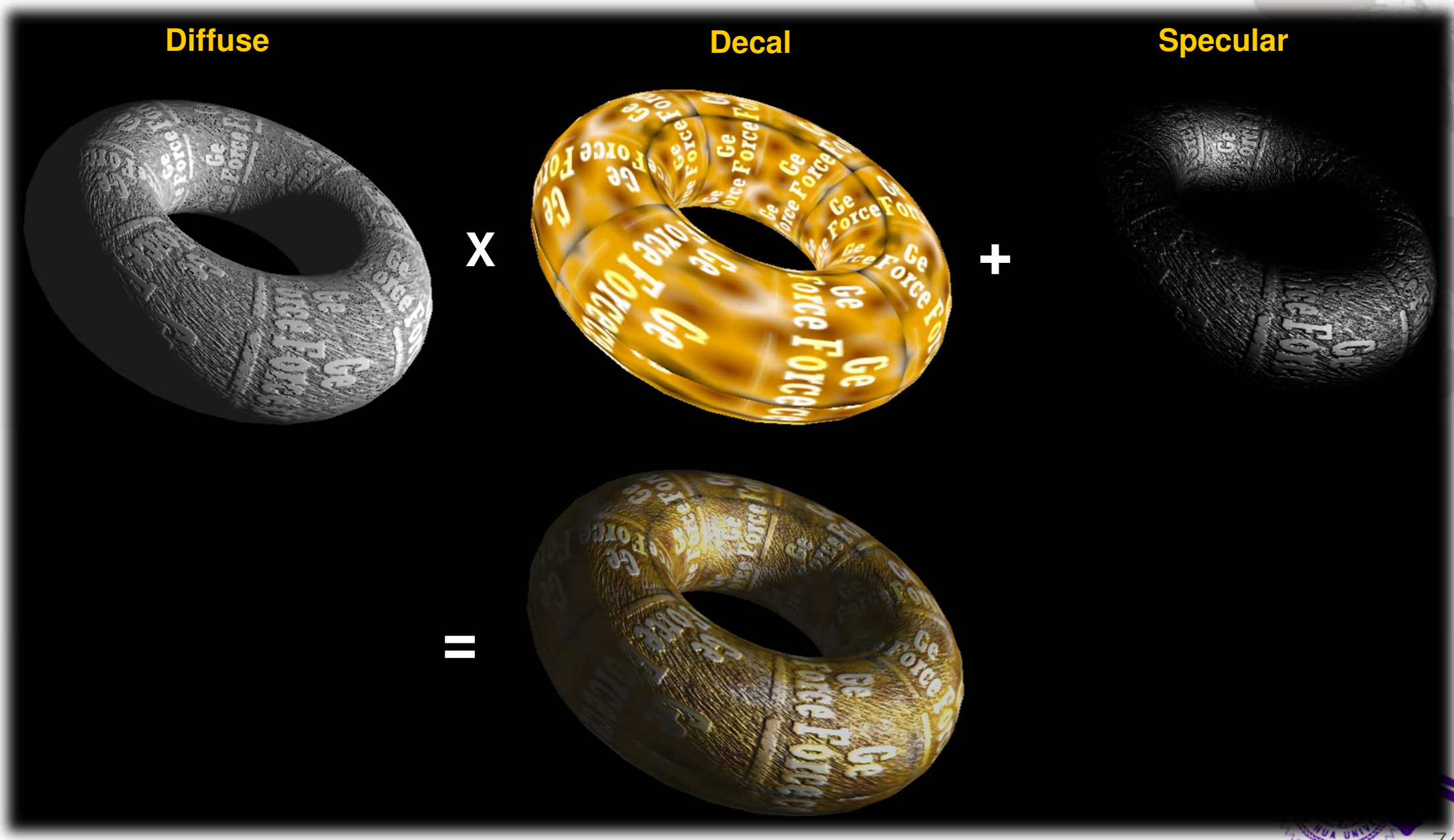
original mesh
4M triangles



simplified mesh
500 triangles

simplified mesh
and normal mapping
500 triangles

# Bump Mapping

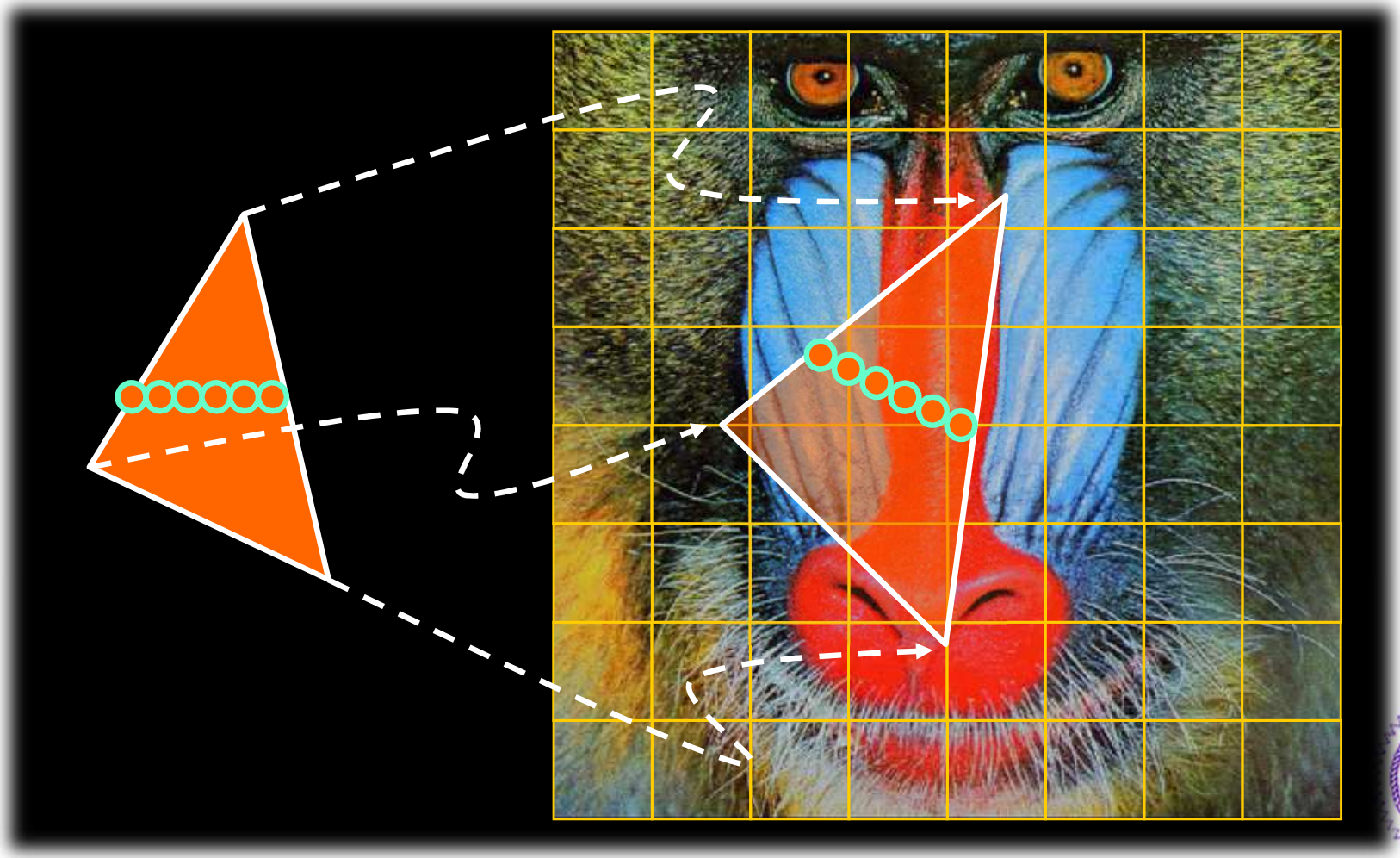


# Hardware Consideration

- Use texture cache to reduce the number of memory access
  - Access a group of texels instead of a single texel each time
  - Neighboring texels are likely to be accessed for the next fragment in texture mapping process
  - Texels are accessed and stored in the texture cache
  - Optimize the texel hit rate with suitable cache size and replacement rule

# Hardware Consideration

- Use tiled texture to utilize the locality of texture access

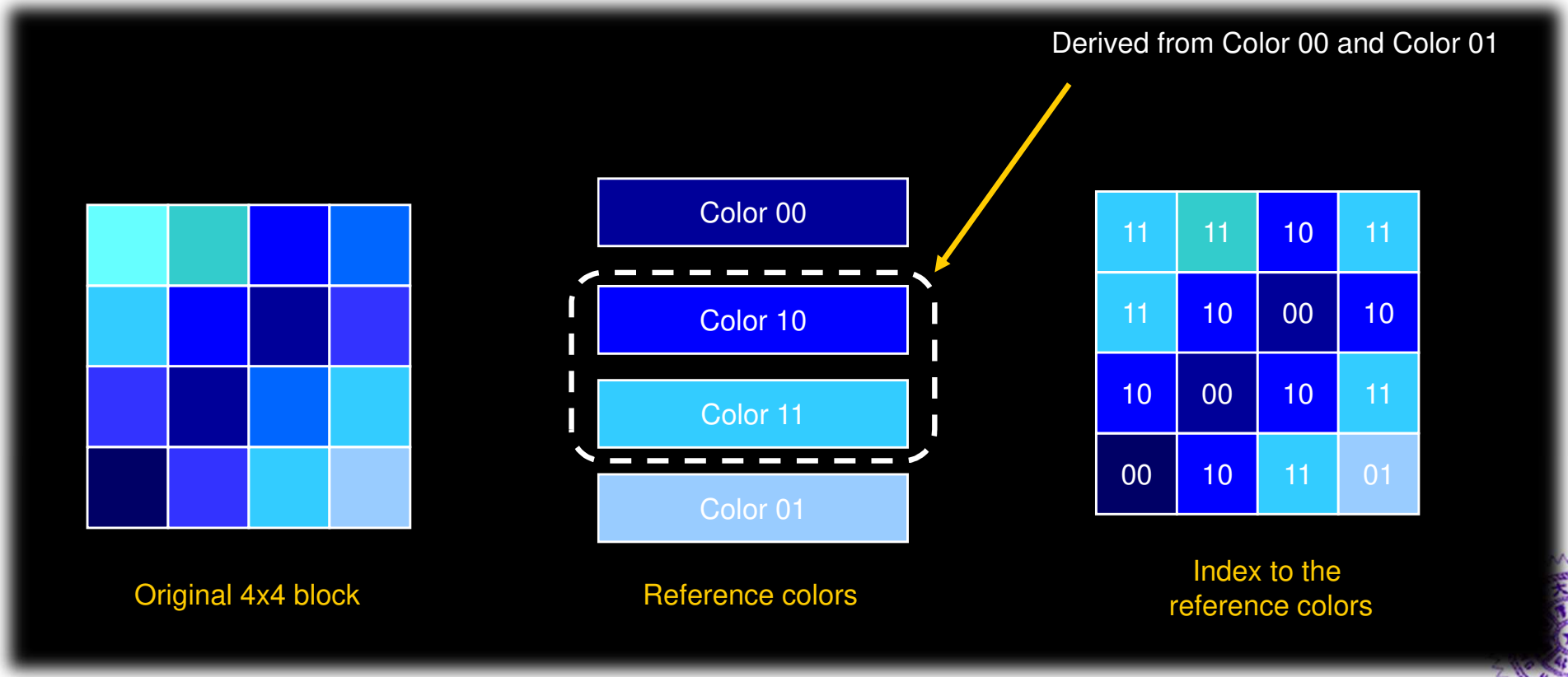


# Hardware Consideration

- Use compressed texture to reduce the memory bandwidth requirement
  - Decoding speed must fast
  - Good for random access
  - Lossy compression is allowed
  - Encoding speed is not as critical as other real time encoding requirement
  - Eg. S3TC, PVRTC, Ericsson TC

# Hardware Consideration

- S3TC Texture Compression
  - 4x4 block encoding
  - Fixed rate encode



# Compressed Textures

- Quality of compressed textures
- Eg. S3TC


Uncompressed Texture


S3TC compressed Texture

# Q&A