

# Texture

# Texture Mapping

# Recall: Shading

Shading in Graphics: applying material to an object, with light response

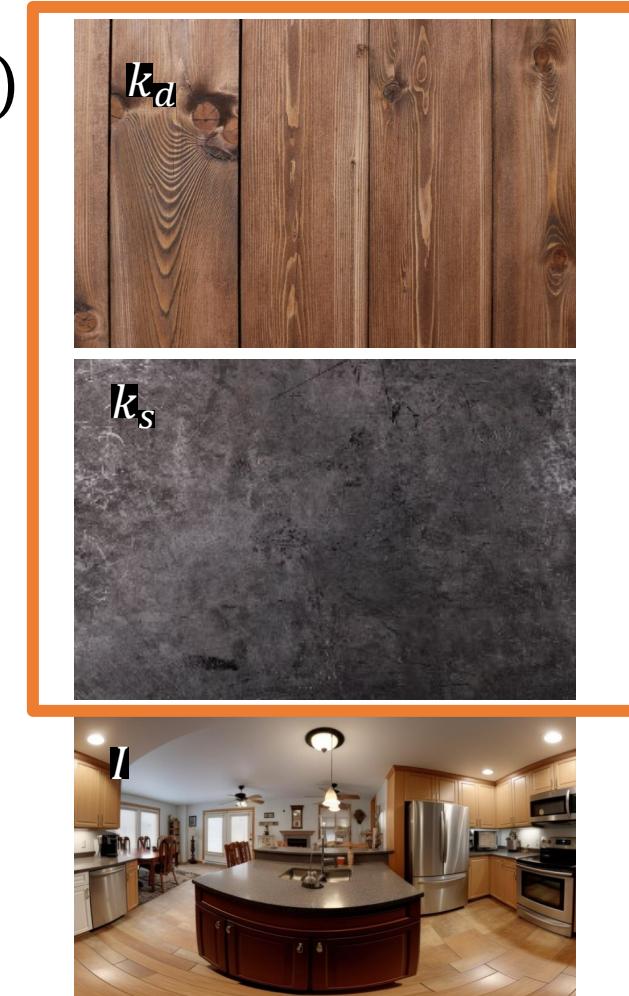
$$I = k_a I_a + f_{att} I_l (k_d \cos \theta + k_s (\cos \phi)^{n_s})$$



# Recall: Shading

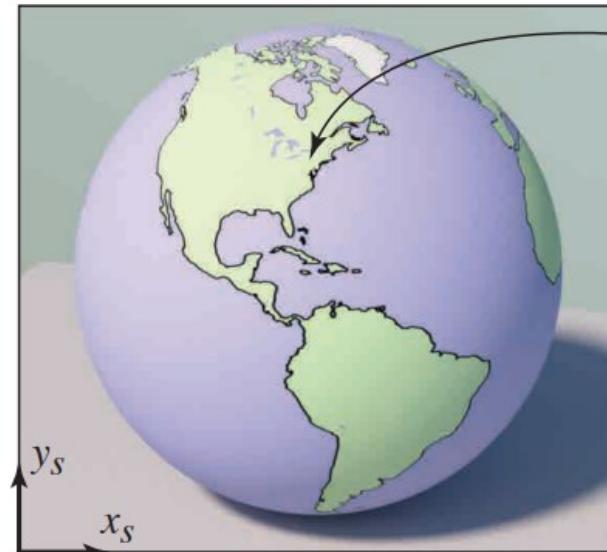
Shading in Graphics: applying material to an object, with light response

$$I = k_a I_a + f_{att} I_l (k_d \cos \theta + k_s (\cos \phi)^{n_s})$$

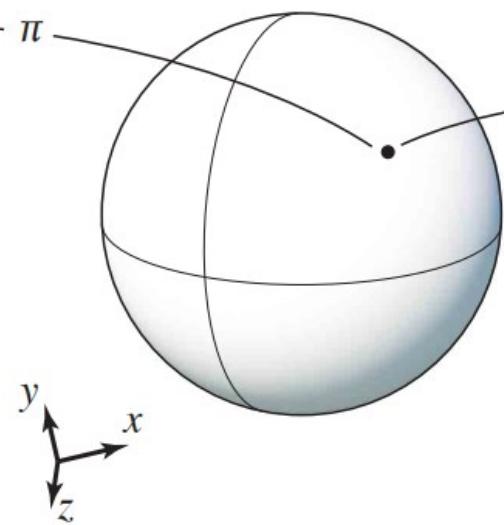


# Texture Mapping

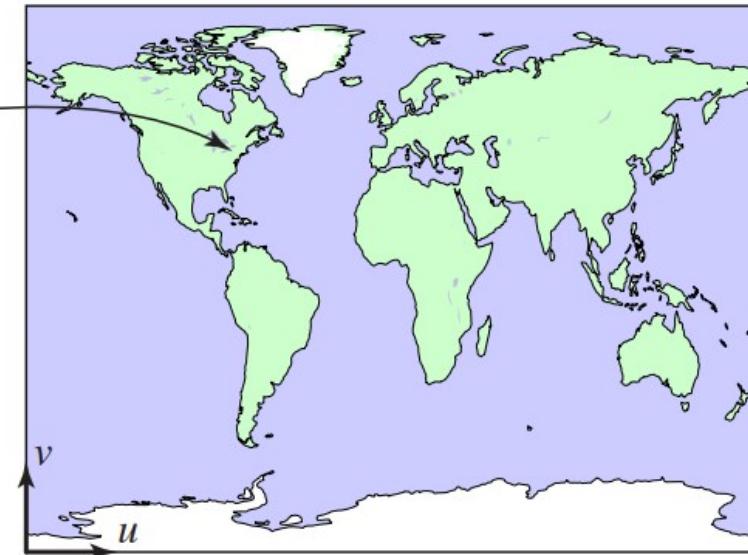
- Texture mapping: apply 2D texture onto 3D surface
- Through texture coordinate (uv coordinate) mapping



Screen space



World space



Texture space

# Texture Mapping

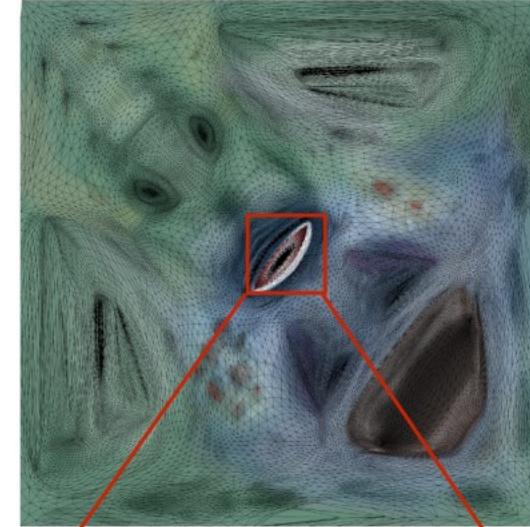
Rendering without texture



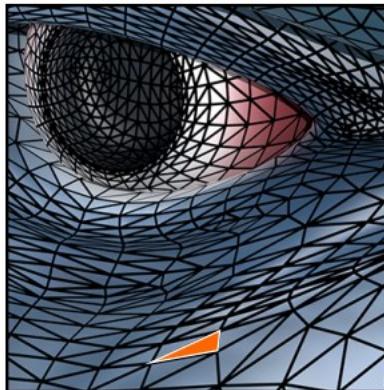
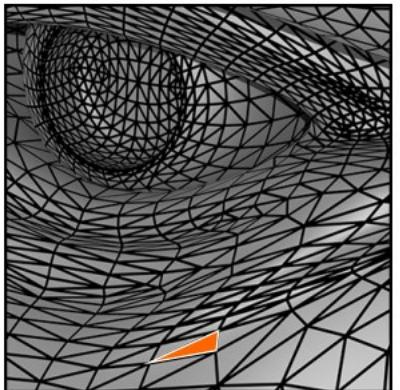
Rendering with texture



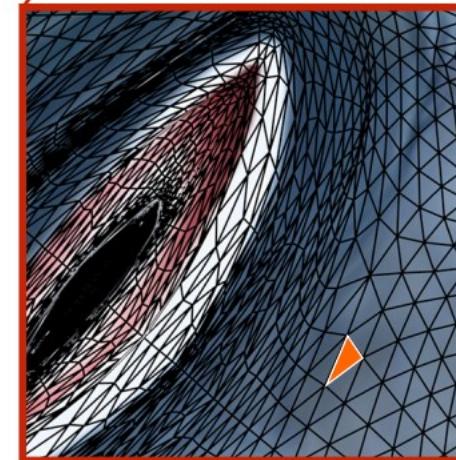
Texture



Zoom

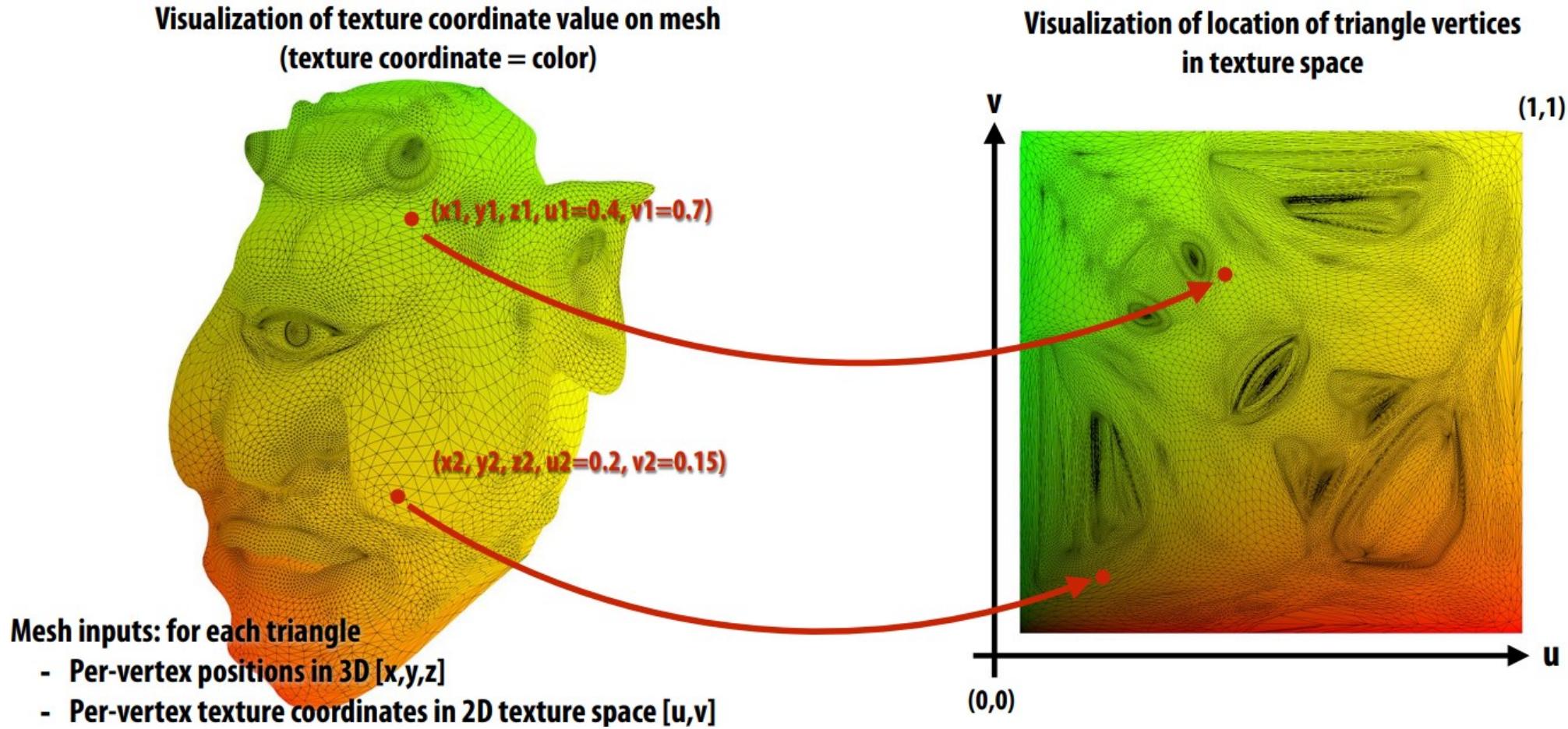


Each triangle “copies” a piece of  
the texture image to the surface.

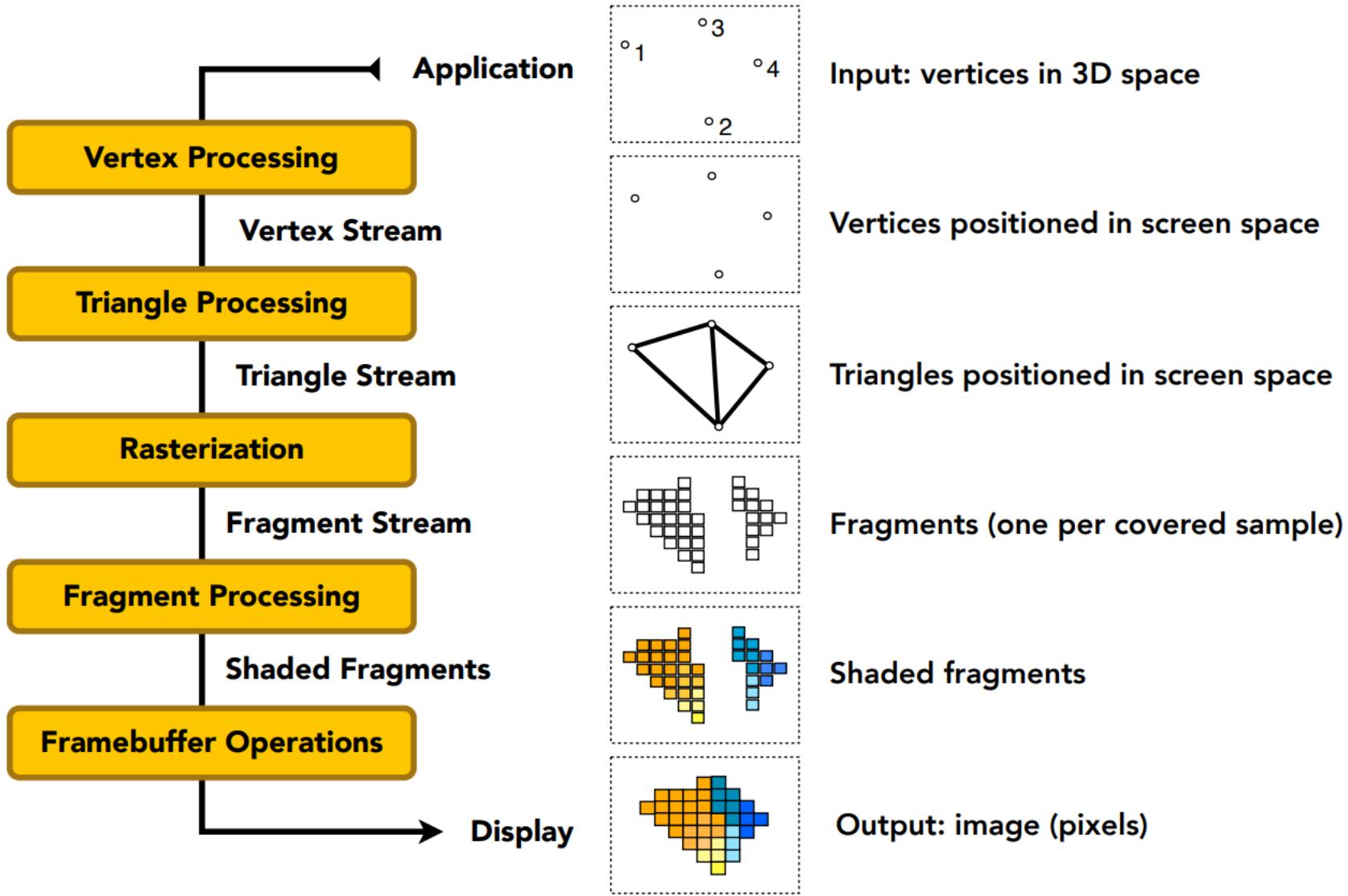


# Texture Coordinate

- How to compute texture coordinates is a big subject! Mesh parametrization!

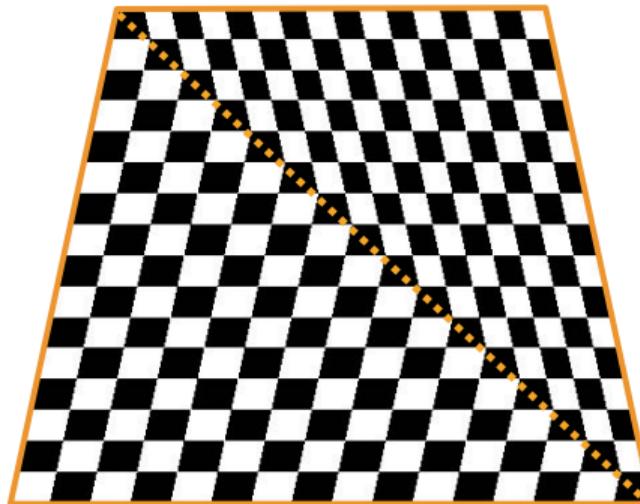


# Rendering Pipeline

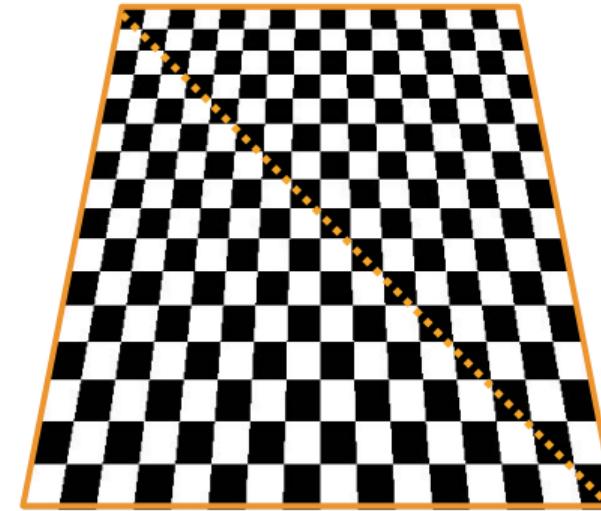


# Compared with 2D

- Texture Mapping  $\approx$  2D image warping
- Bilinear interpolation, MIPMAP ... is the same as 2D
- However directly warping image in 2D can lead to artifacts



Wrong mapping

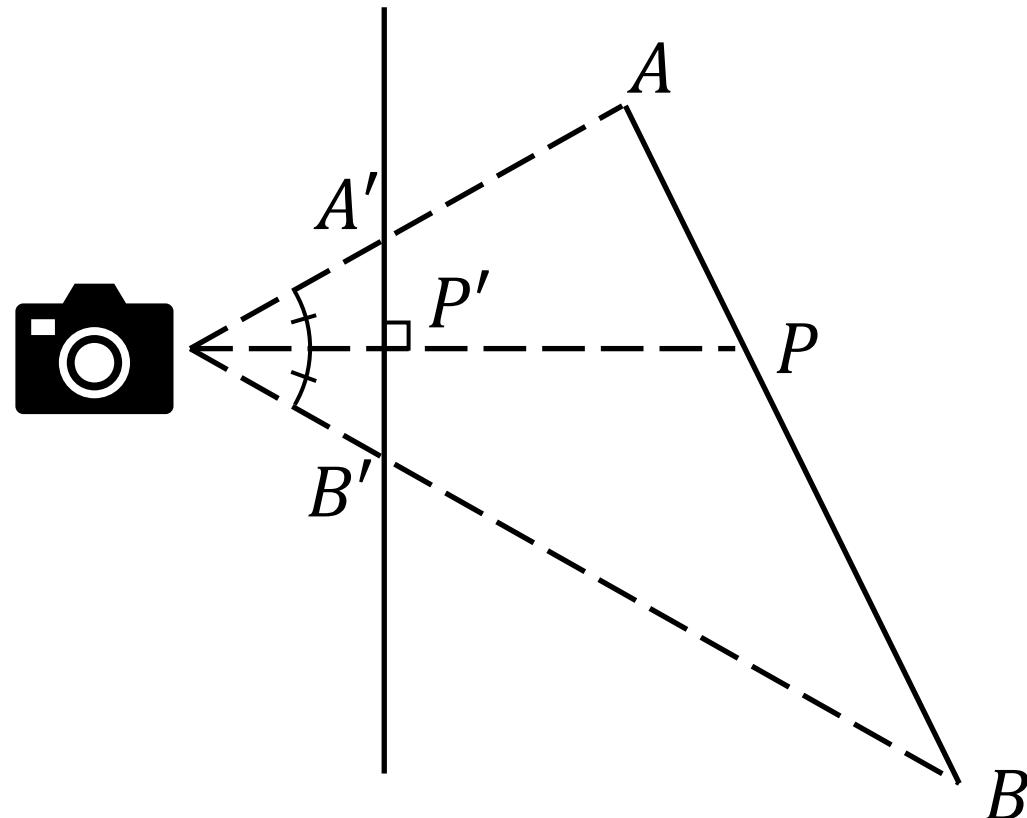


Perspectively correct

From Fatahalian et.al, CS248

# Perspective Transformation

- $\overline{A'P'} = \overline{B'P'}$ , but  $\overline{AP} \neq \overline{BP}$
- Screen space interpolation  $\neq$  world space interpolation

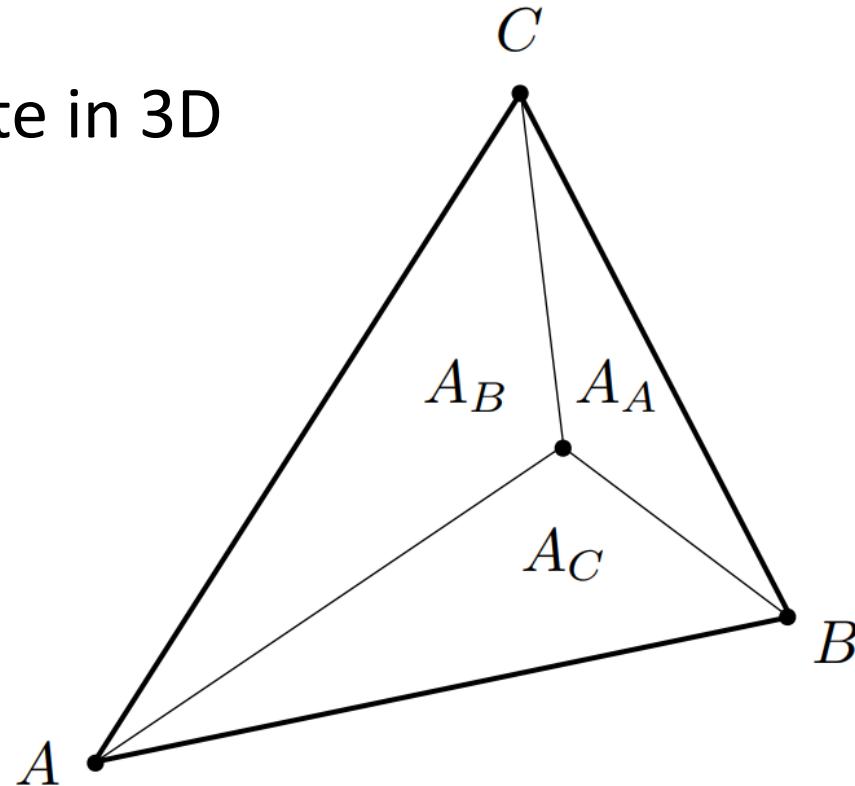


# World Space Interpolation

- Barycentric coordinates again!

$$\mathbf{x} = \alpha \mathbf{x}_A + \beta \mathbf{x}_B + \gamma \mathbf{x}_C, \alpha + \beta + \gamma = 1$$

- $\mathbf{x}$  is coordinate in 3D



$$\alpha = \frac{A_A}{A_A + A_B + A_C}$$

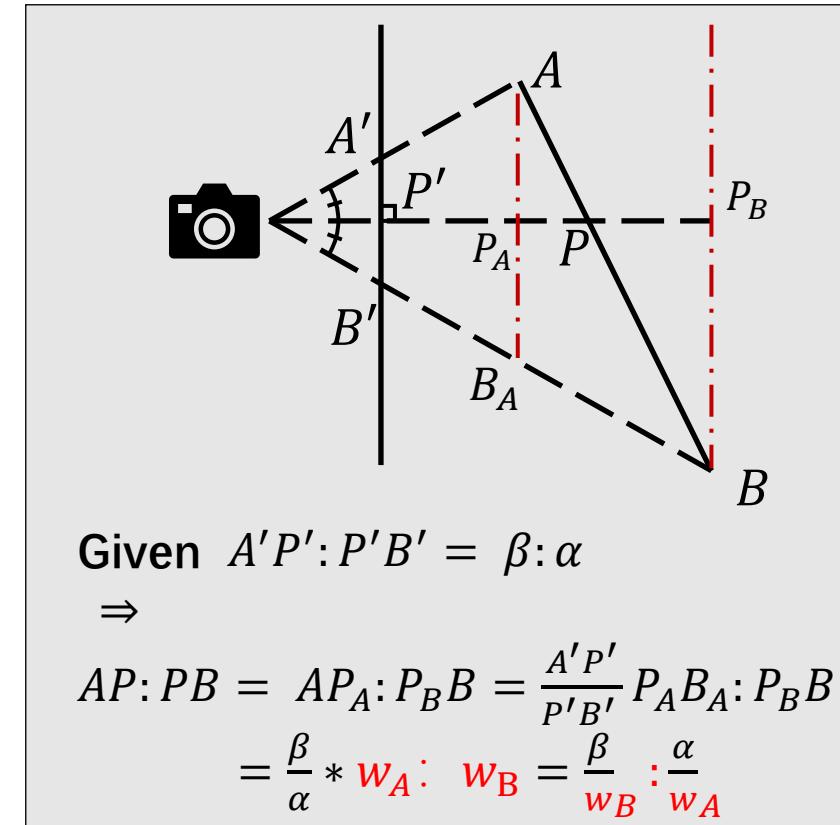
$$\beta = \frac{A_B}{A_A + A_B + A_C}$$

$$\gamma = \frac{A_C}{A_A + A_B + A_C}$$

# Perspective-correct Interpolation

$$f = \frac{\frac{\alpha}{w_A} f_A + \frac{\beta}{w_B} f_B + \frac{\gamma}{w_C} f_C}{\frac{\alpha}{w_A} + \frac{\beta}{w_B} + \frac{\gamma}{w_C}}$$

- $f_A, f_B, f_C$ : the value to be interpolated
- $\alpha, \beta, \gamma$ : barycentric coordinates **in screen coord.**
- $w_A, w_B, w_C$ : depth of each vertex **before** perspective transformation
- More details in: [Perspective-Correct Interpolation](#)



# Perspective-correct Interpolation

- Perspective-correct interpolation is used not only for texture coordinates, also for surface normals, depth values...
- Perspective-correct interpolation is default for modern graphics APIs, like OpenGL (implemented as fixed pipeline functions)
- Interestingly, if interpolating depth value in screen space, we get

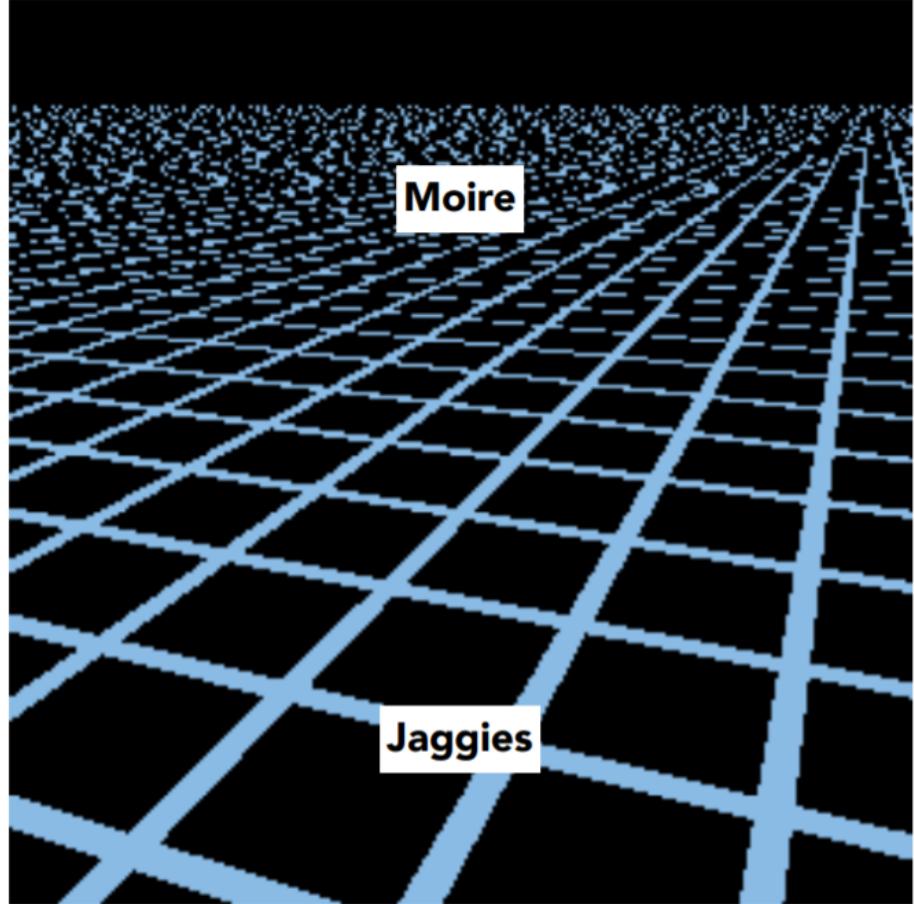
$$\frac{1}{w_p} = \frac{\alpha}{w_A} + \frac{\beta}{w_B} + \frac{\gamma}{w_C}$$

hint: when  $f=z$ , i.e, when  $f_A = w_A$ ,  
 $f_B = w_B$ ,  $f_C = w_C$ ,  $f_P = w_P$

$$f = \frac{\frac{\alpha}{w_A} f_A + \frac{\beta}{w_B} f_B + \frac{\gamma}{w_C} f_C}{\frac{\alpha}{w_A} + \frac{\beta}{w_B} + \frac{\gamma}{w_C}}$$

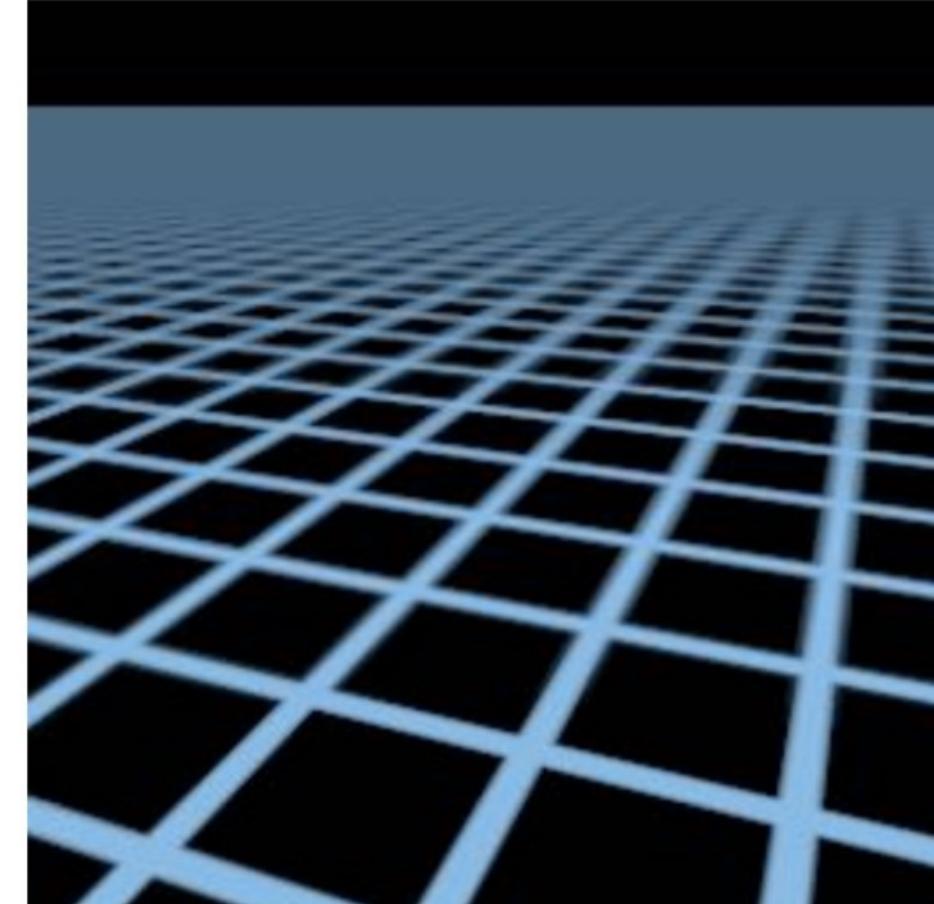
which is linearly interpolating the inverse of depth

# Anti-Aliasing



Aliasing

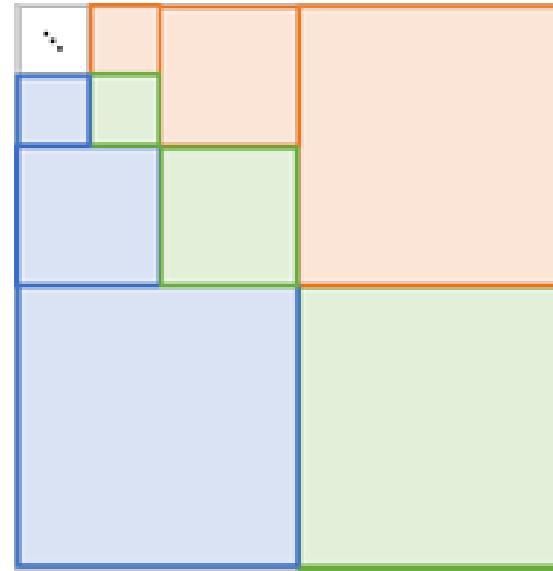
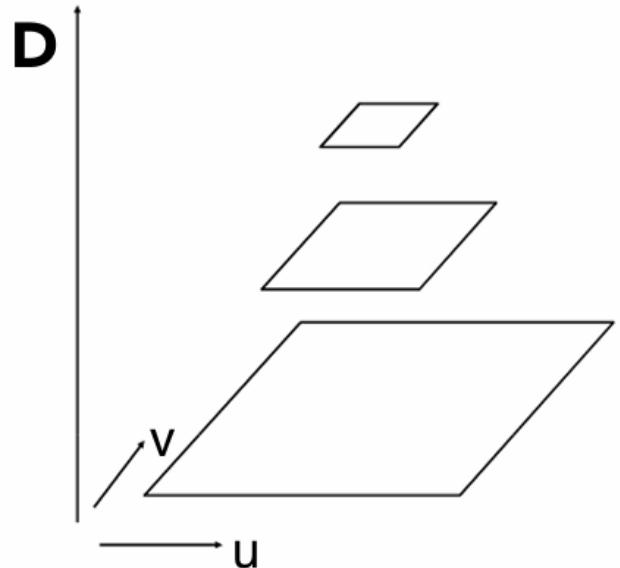
From Fatahalian et.al, CS248



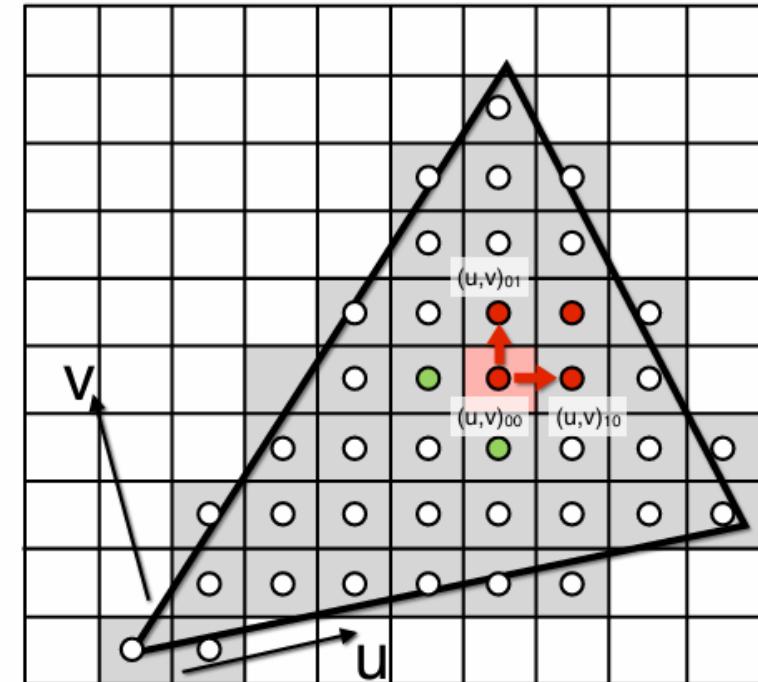
MIPMAP result

# Anti-Aliasing

## MIPMAP



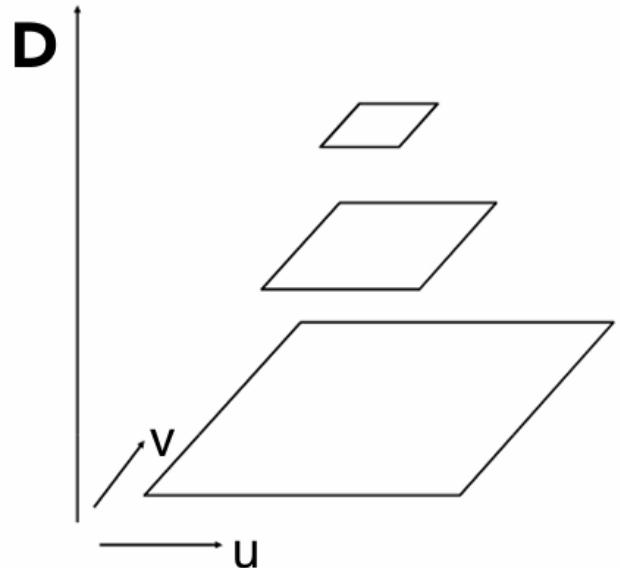
Which level ?



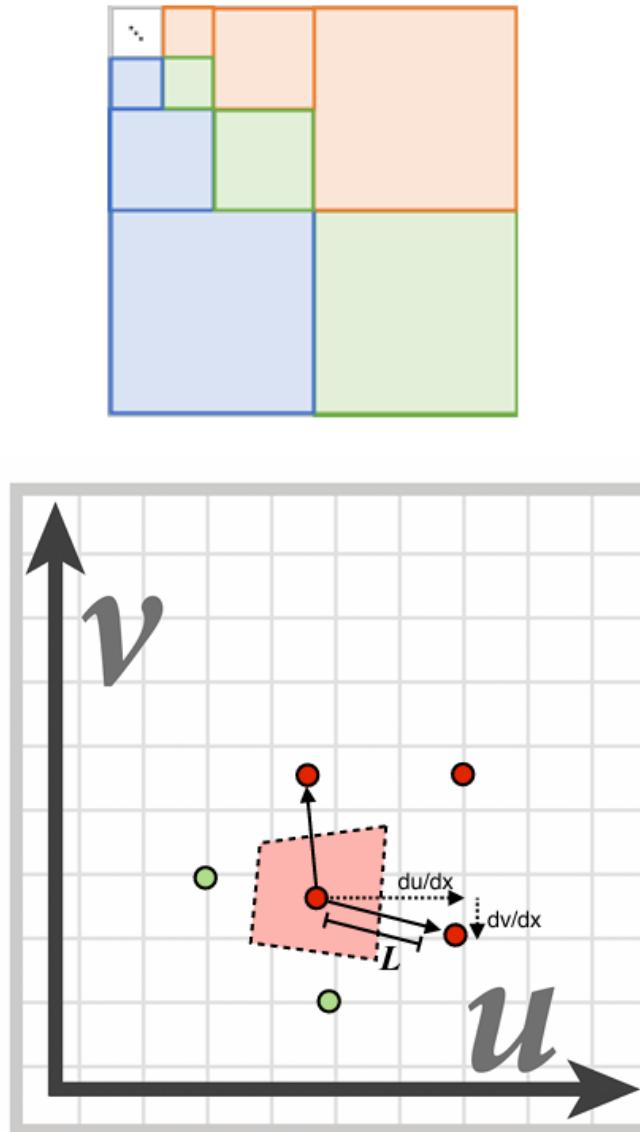
# Anti-Aliasing

## MIPMAP

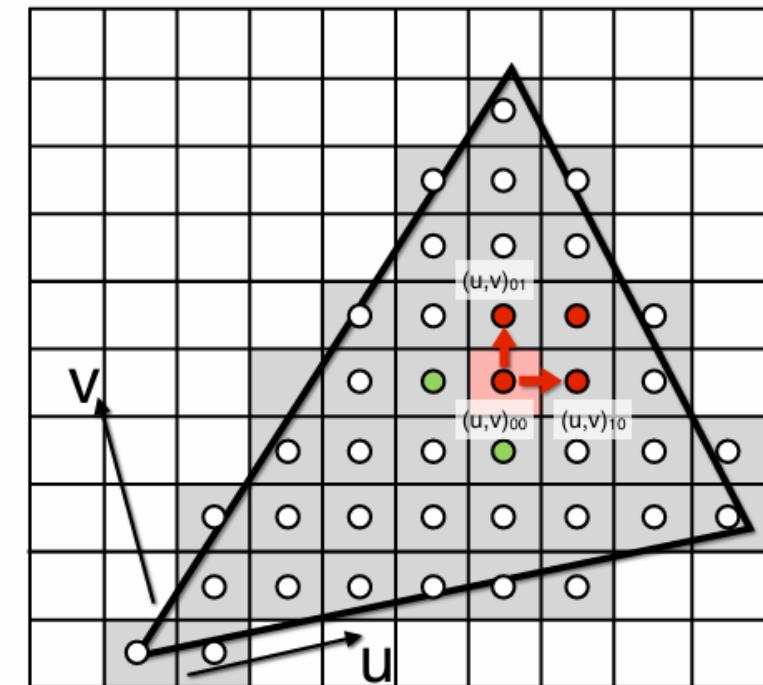
$$D = \log_2 L$$



$$L = \max \left( \sqrt{\left( \frac{du}{dx} \right)^2 + \left( \frac{dv}{dx} \right)^2}, \sqrt{\left( \frac{du}{dy} \right)^2 + \left( \frac{dv}{dy} \right)^2} \right)$$

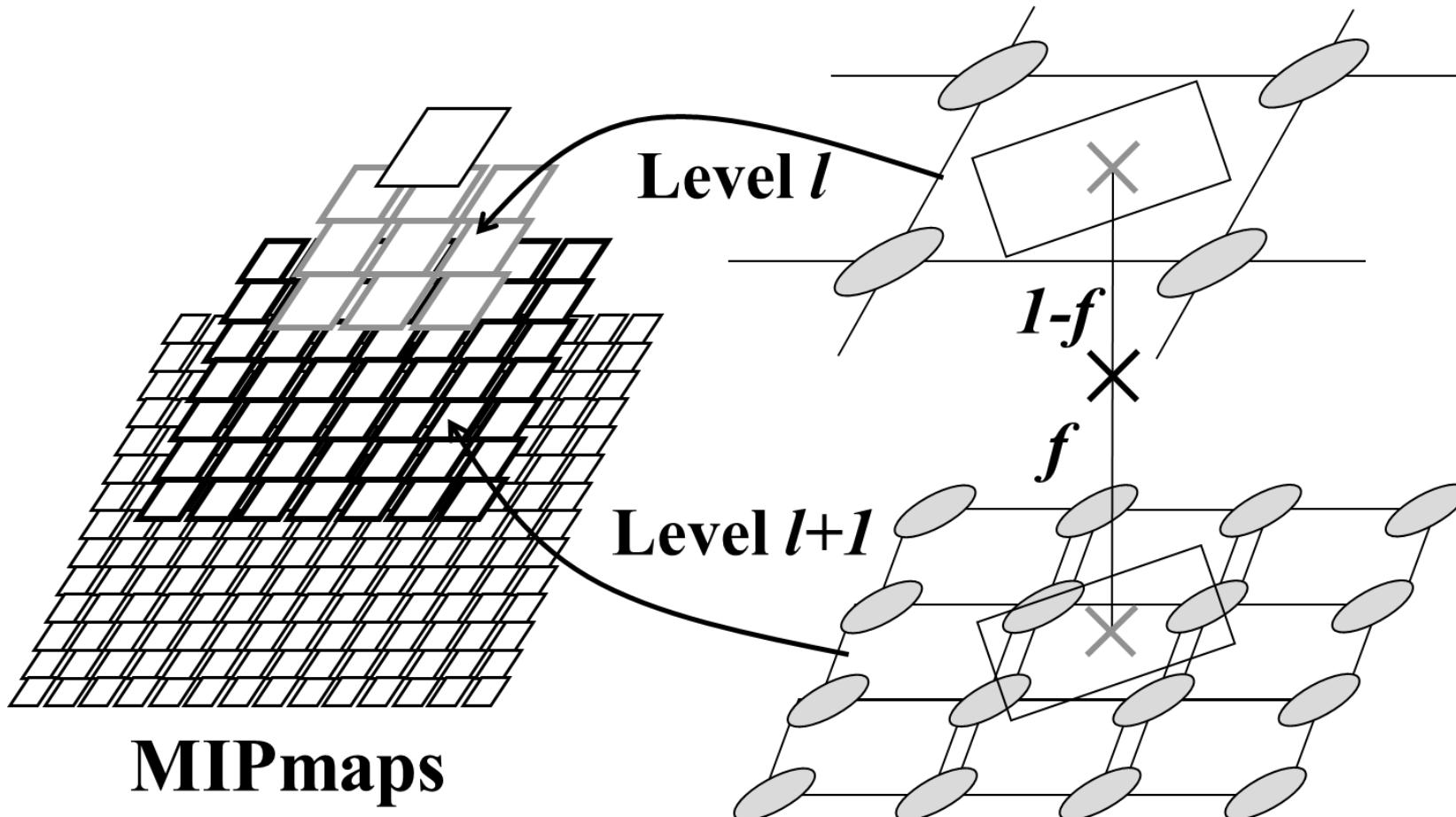


Which level ?



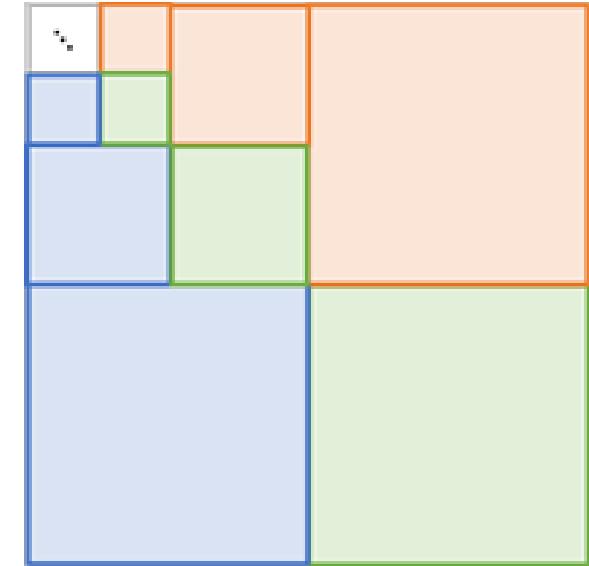
# Anti-Aliasing

MIPMAP

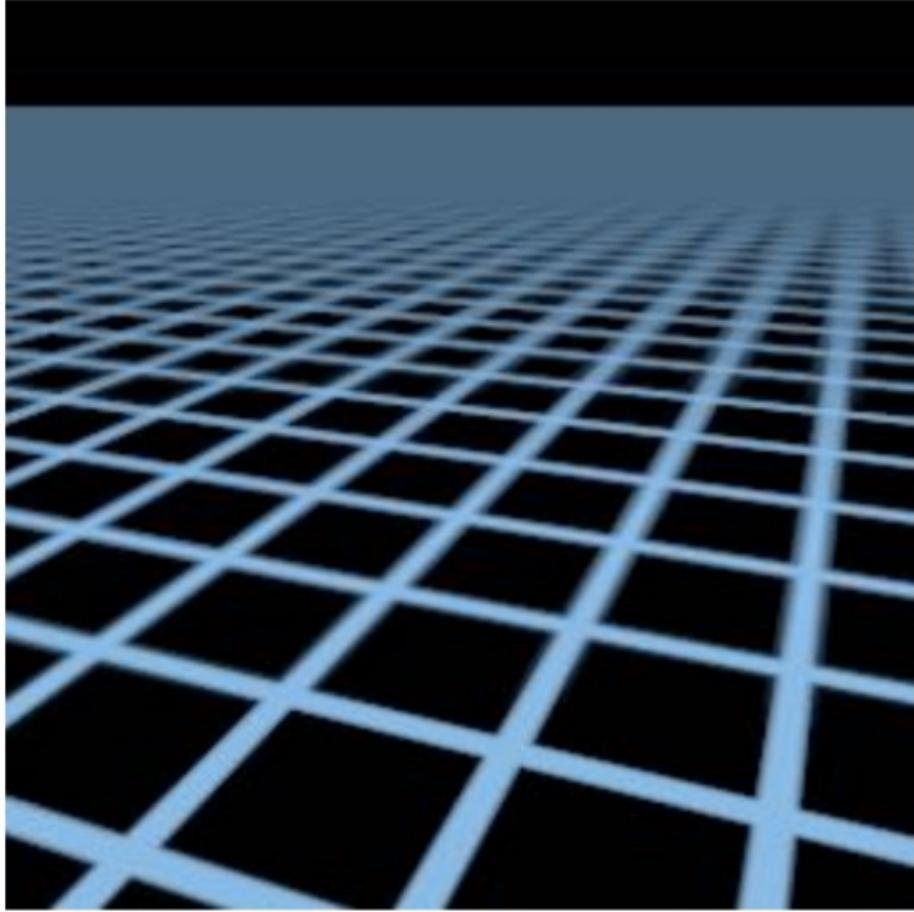


$$D = \log_2 L$$

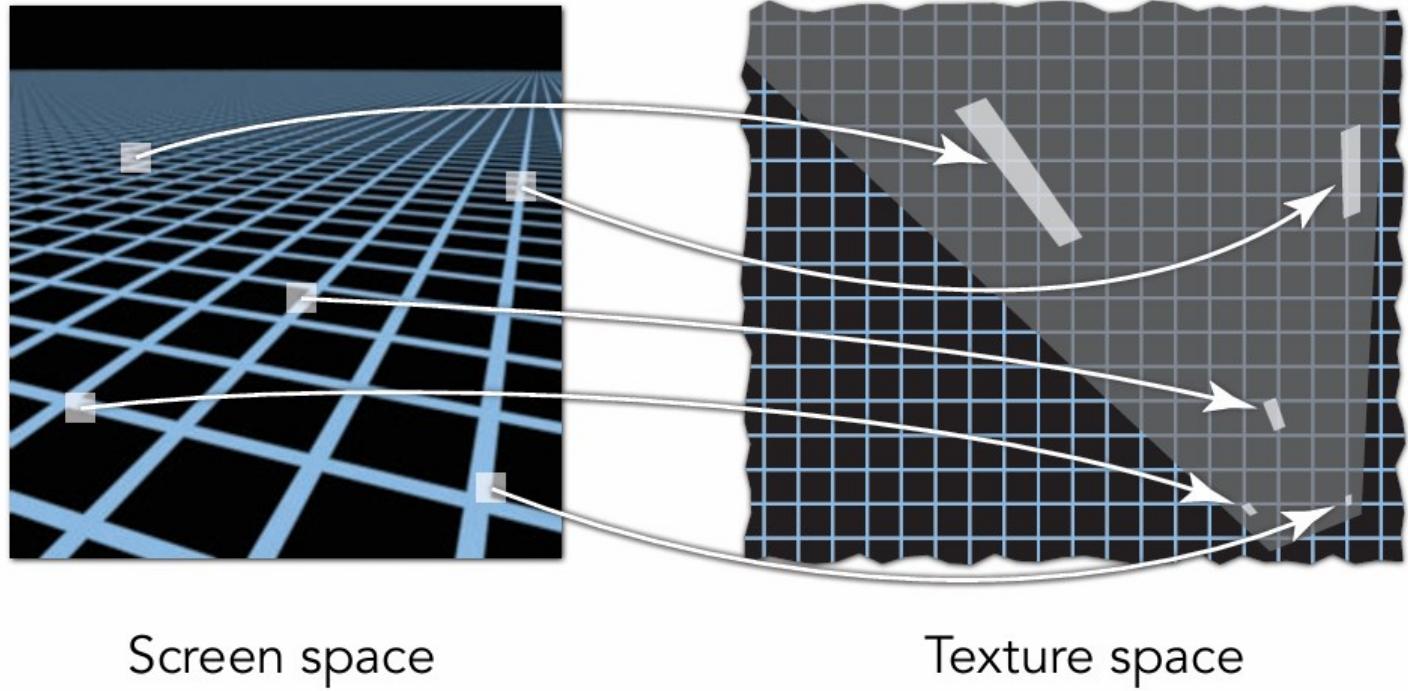
$$l = \text{int}(D)$$



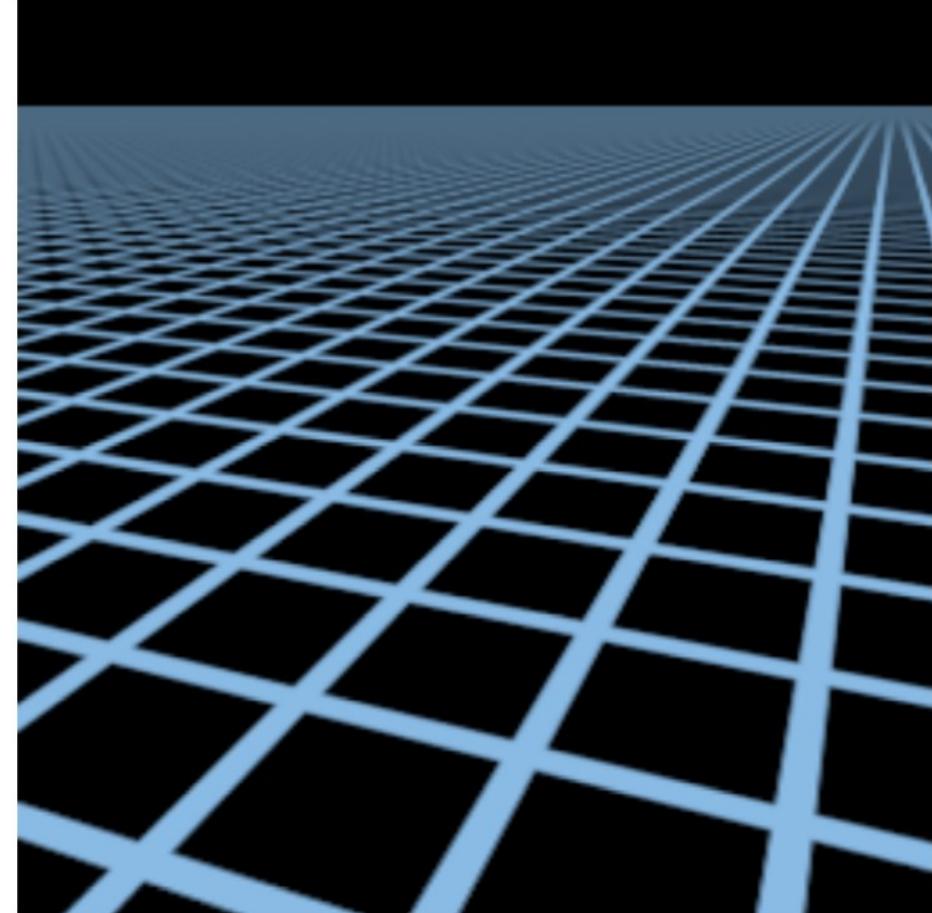
# Anti-Aliasing



MIPMAP result



# Anisotropic Anti-Aliasing

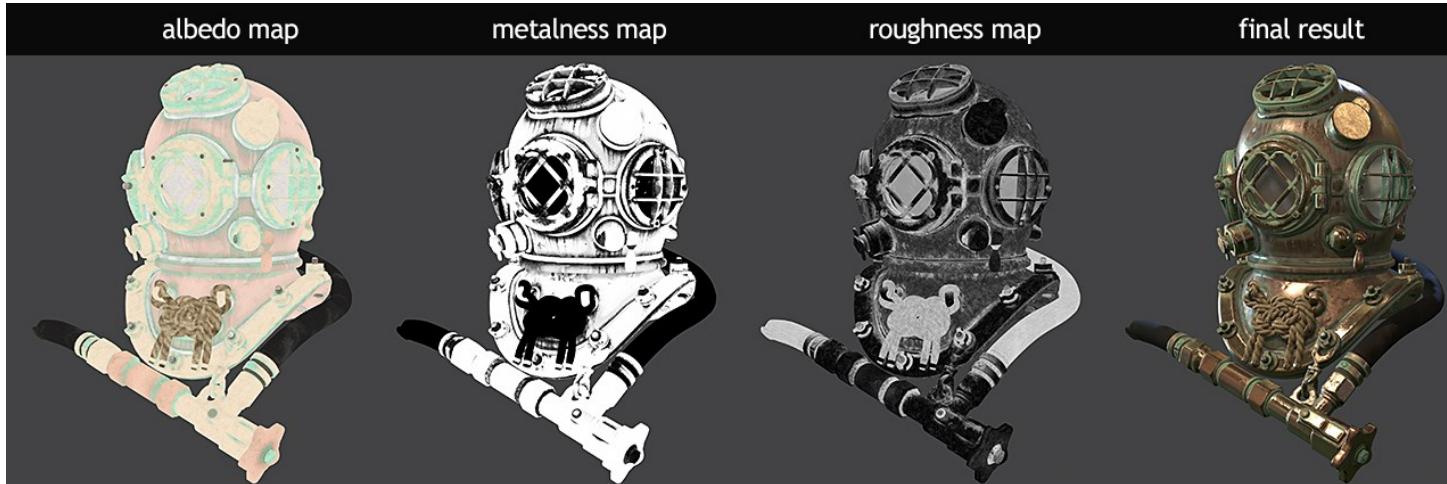


# Applications

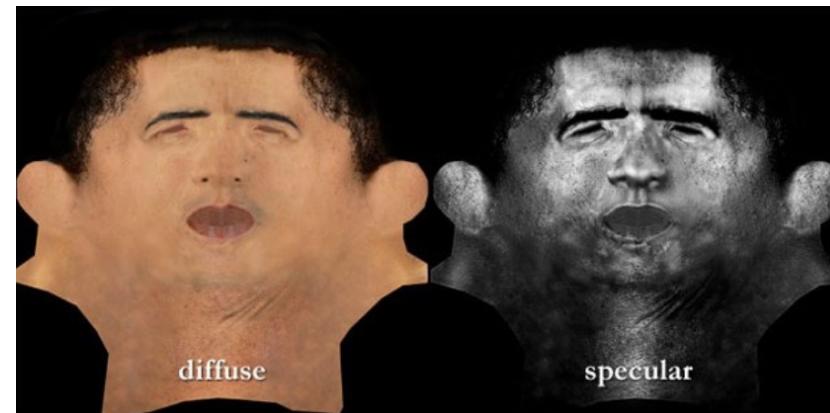
# For Appearance

- Describe colors, roughness, metalness...

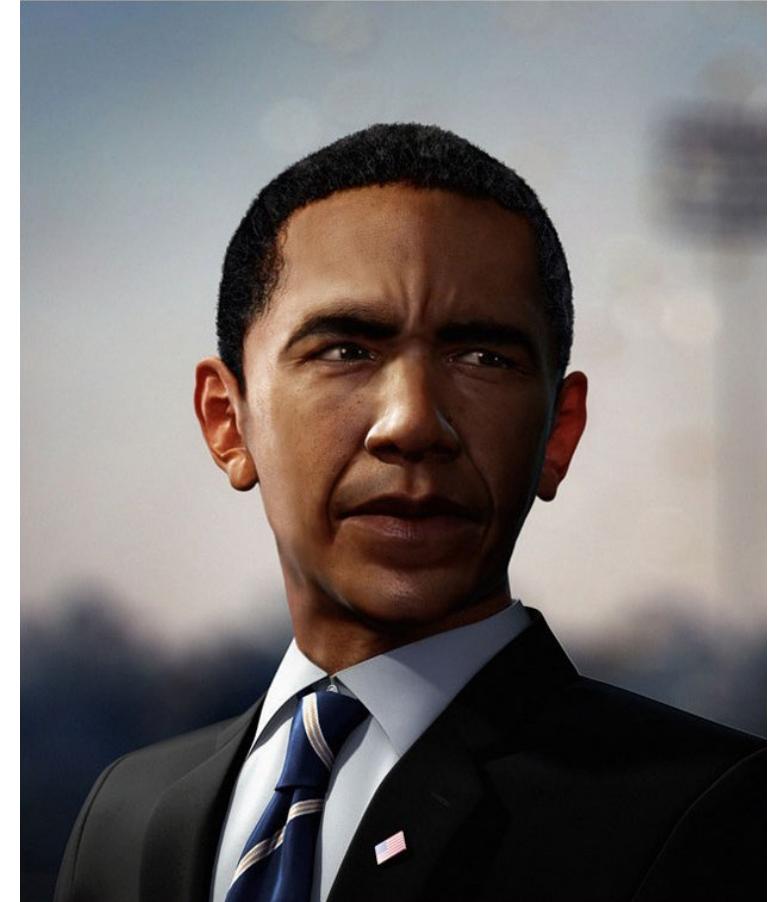
$$I = k_a I_a + f_{att} I_l (k_d \cos \theta + k_s (\cos \phi)^{n_s})$$



PBR Texture Conversion

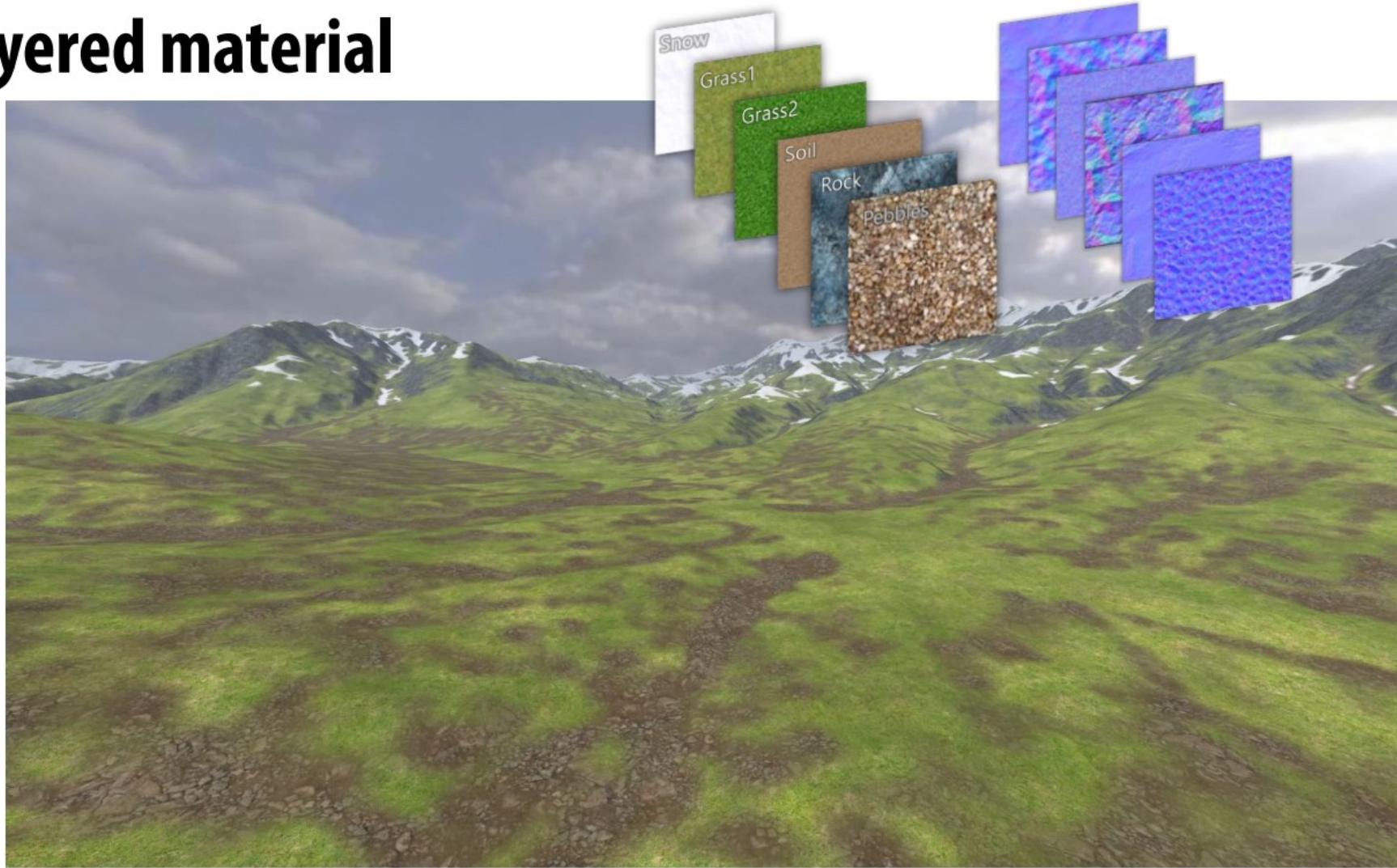


Making Of 'Barack'



# For Appearance

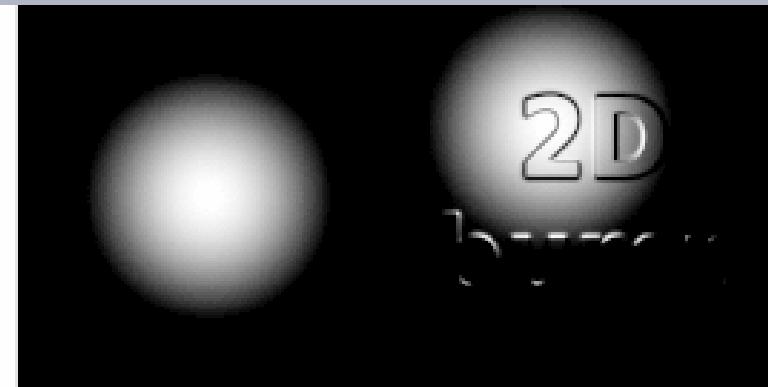
## Layered material



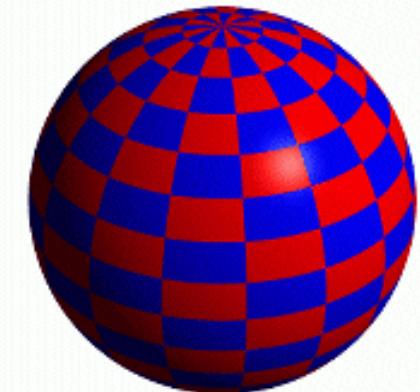
# For Geometry

- Bump map

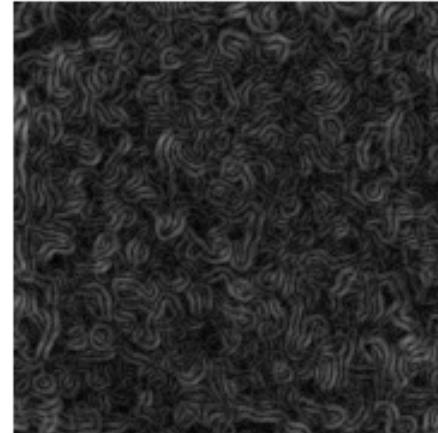
## 2D bump



The bump map is treated as a single-valued height function, whose **partial derivatives** tell how to **alter the true surface normal** at each point on the surface. Bump Mapping assumes that the Illumination model is applied at every pixel (as in Phong Shading or ray tracing).



Sphere w/Diffuse Texture



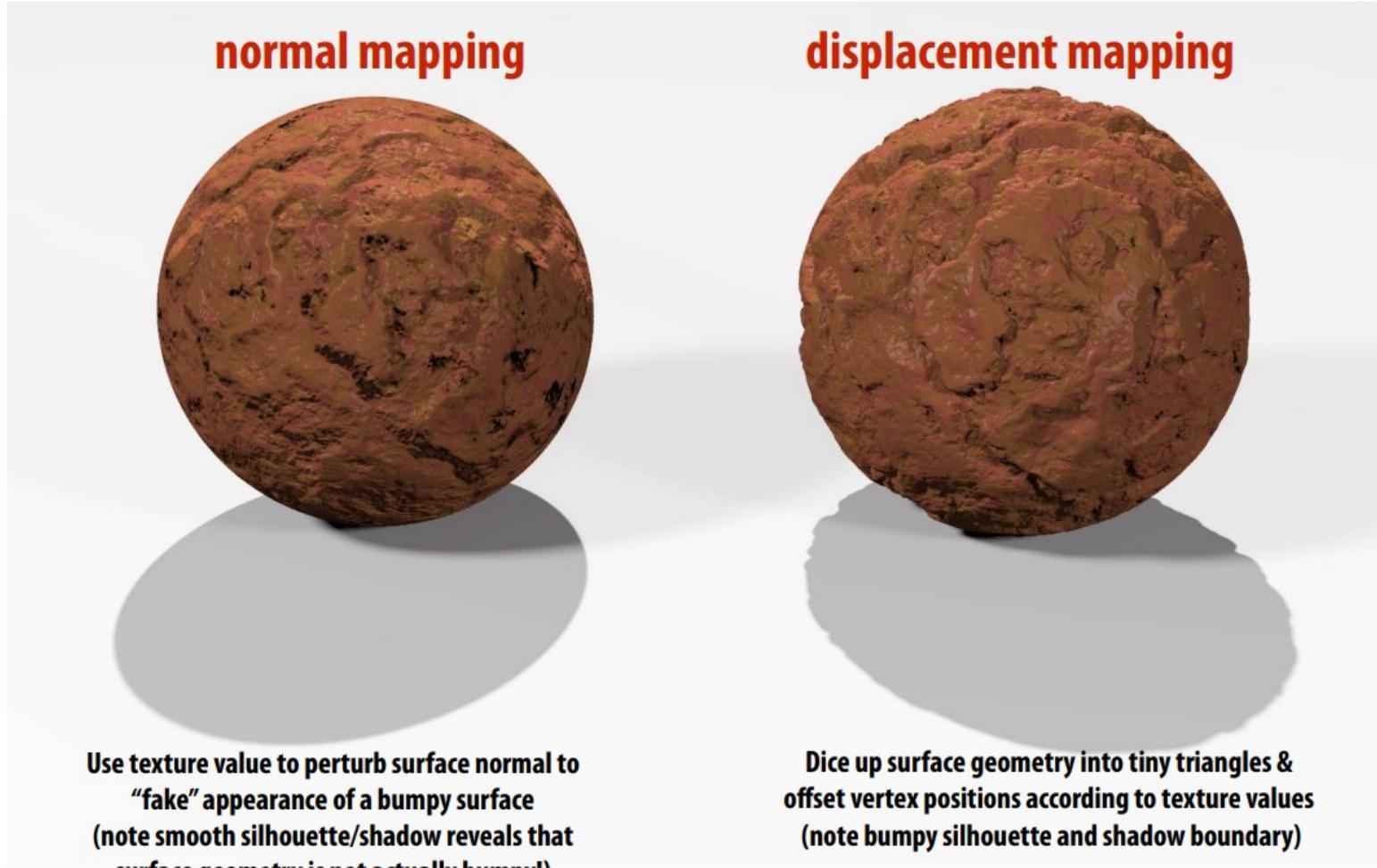
Swirly Bump Map



Sphere w/Diffuse Texture  
& Bump Map

# For Geometry

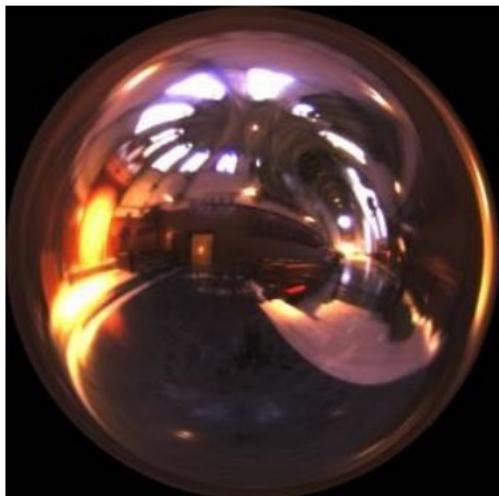
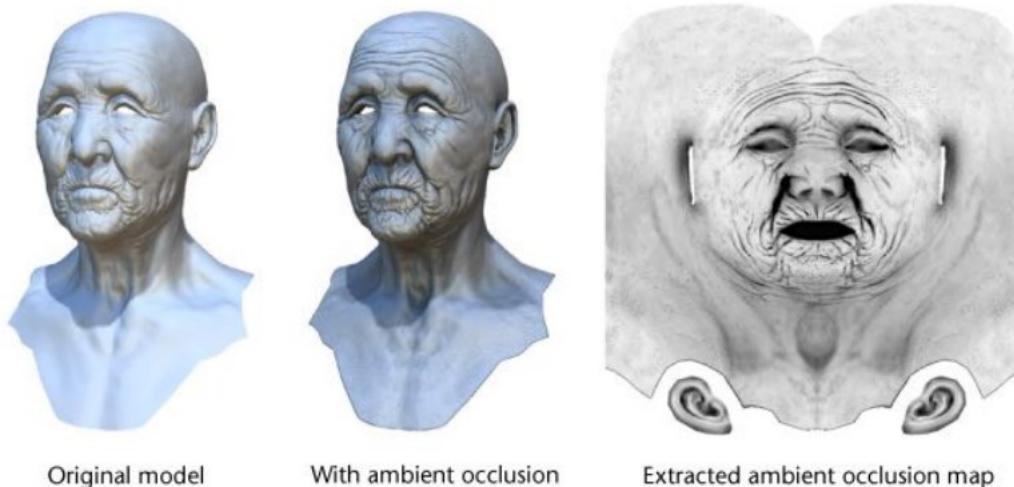
- Normal map, displacement map



From Fatahalian et.al, CS248

# For Lights and Shadows

$$I = k_a I_a + f_{att} I_l (k_d \cos \theta + k_s (\cos \phi)^{n_s})$$



Grace Cathedral environment map

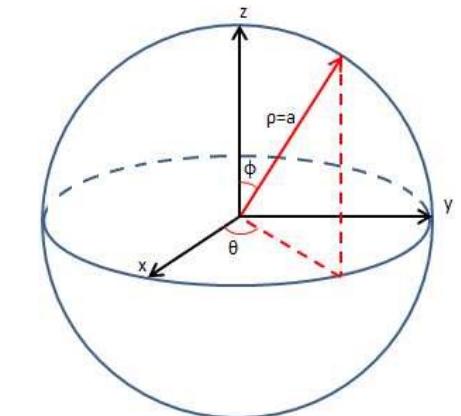


Environment map used in a rendering

$$I = k_a I_a + f_{att} I_l (k_d \cos \theta + k_s (\cos \phi)^{n_s})$$



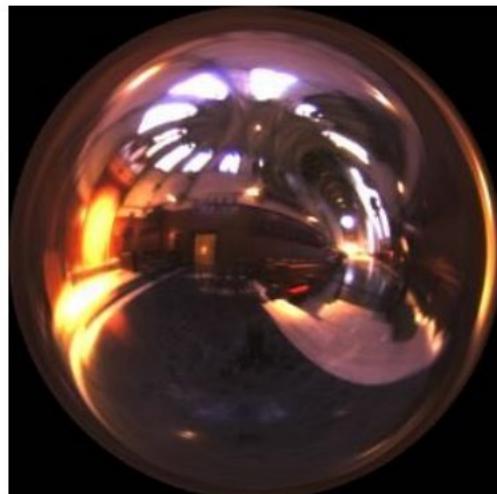
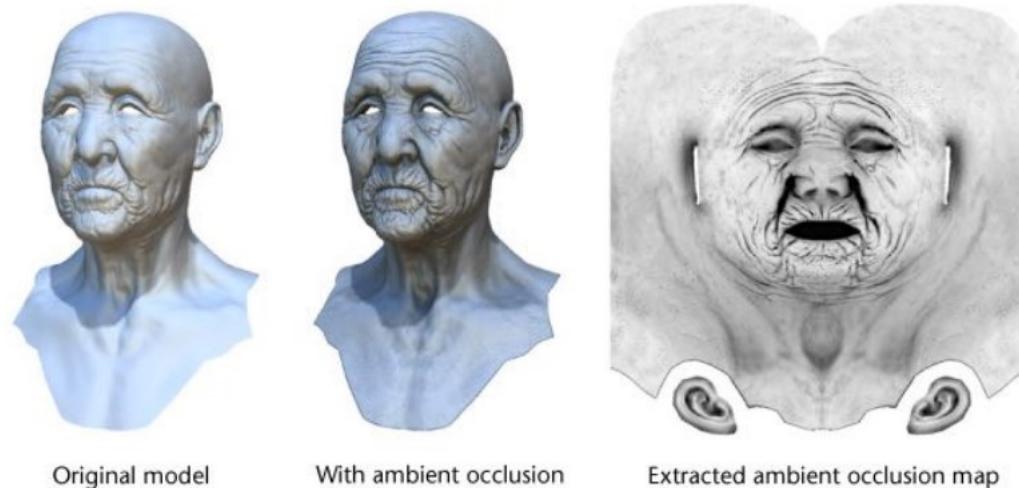
$\theta$



$\phi$

# For Lights and Shadows

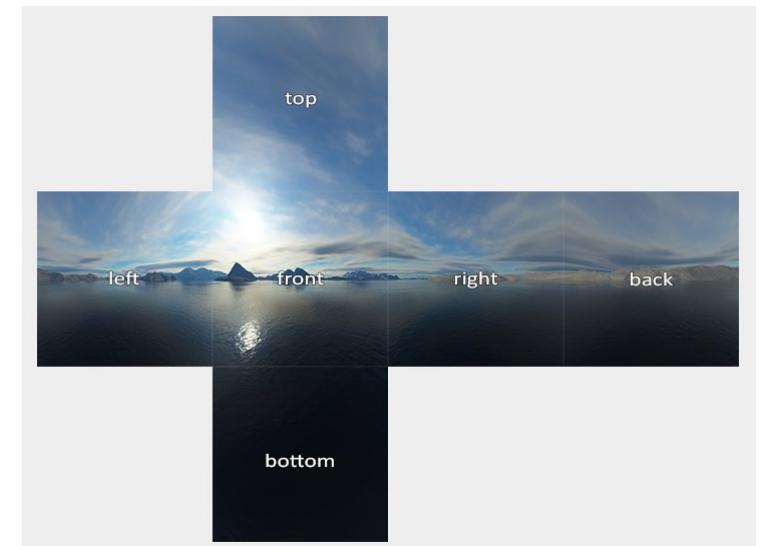
$$I = \mathbf{k}_a I_a + f_{att} I_l (k_d \cos \theta + k_s (\cos \phi)^{n_s})$$



Grace Cathedral environment map

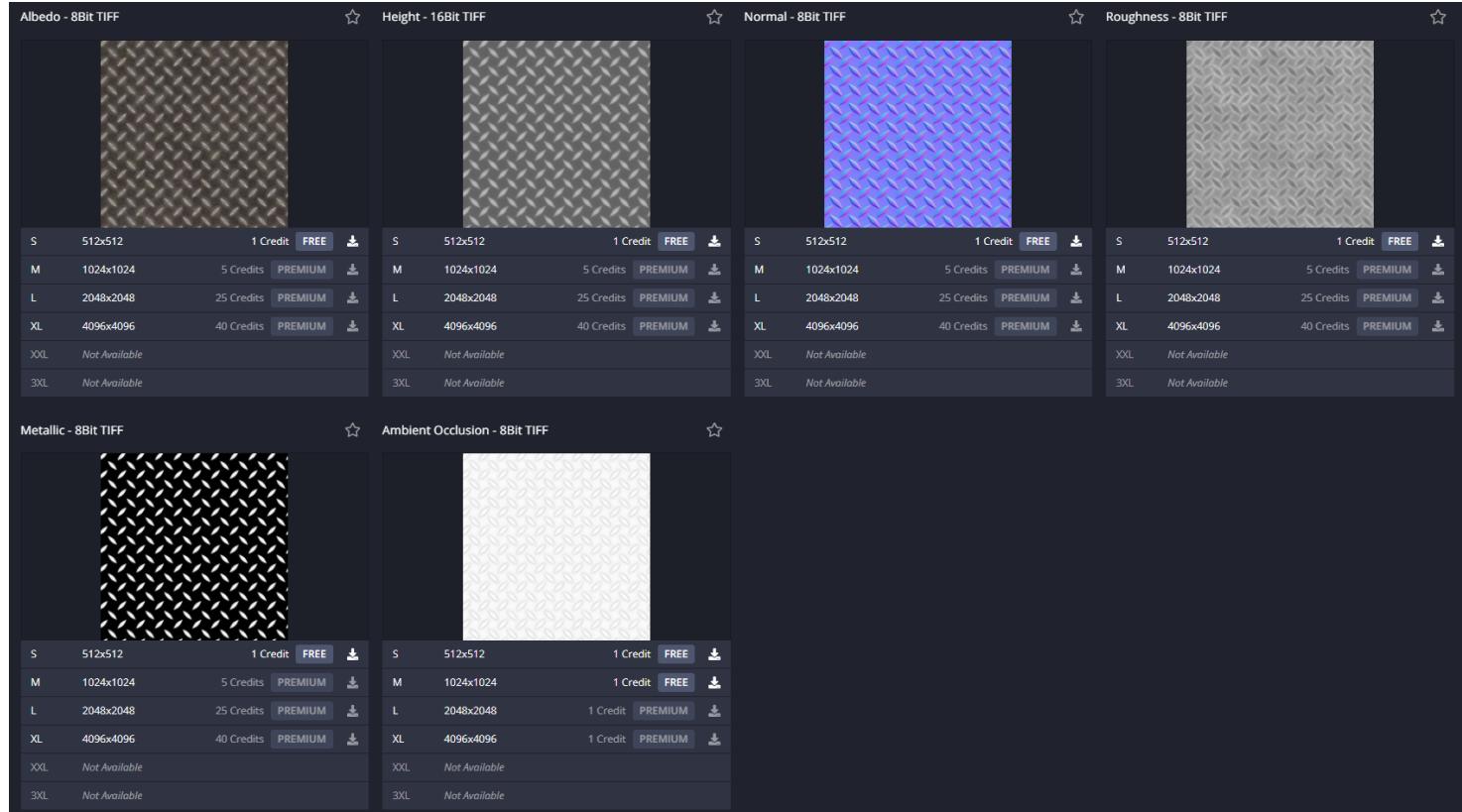


Environment map used in a rendering



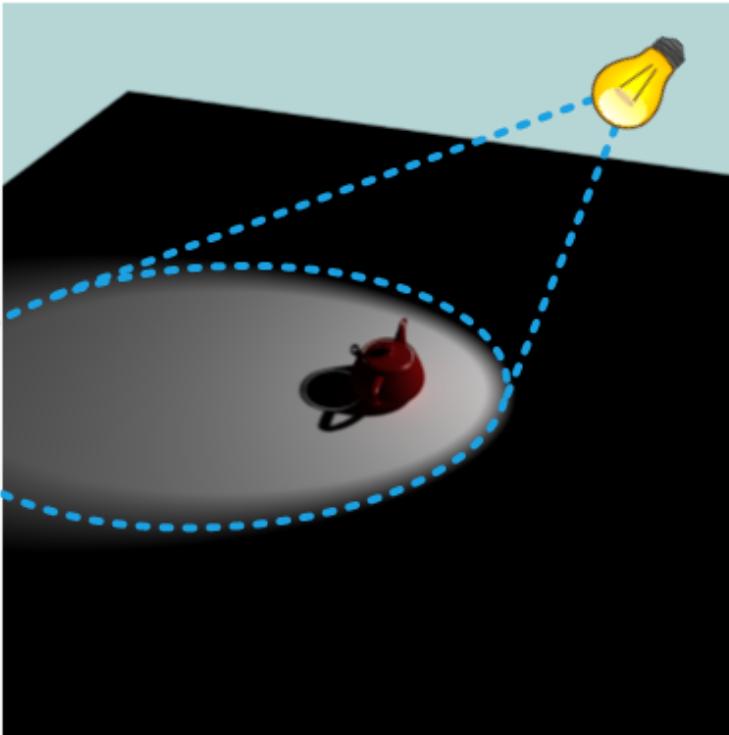
$$I = k_a I_a + \mathbf{f}_{att} I_l (k_d \cos \theta + k_s (\cos \phi)^{n_s})$$

# Color + Geometry + Ambient Occlusion

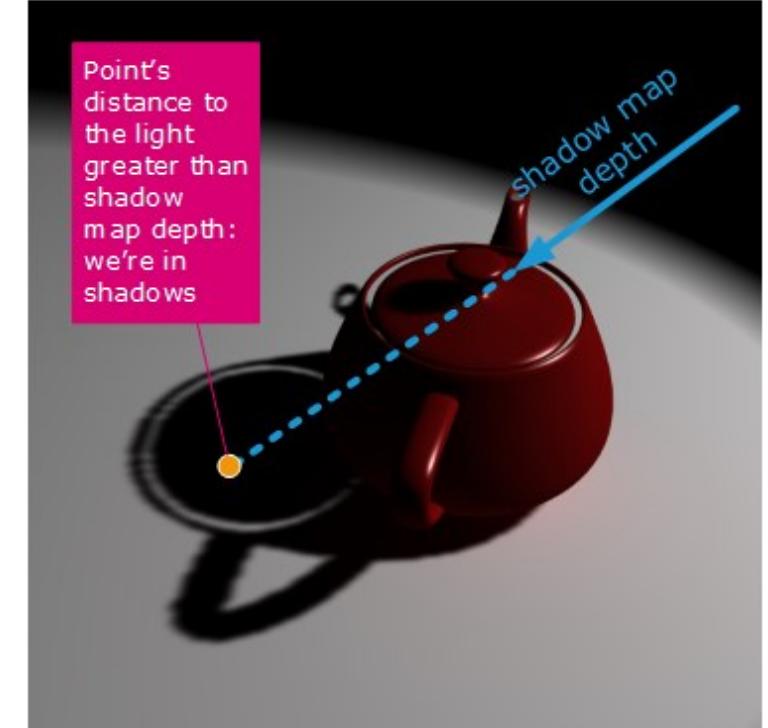
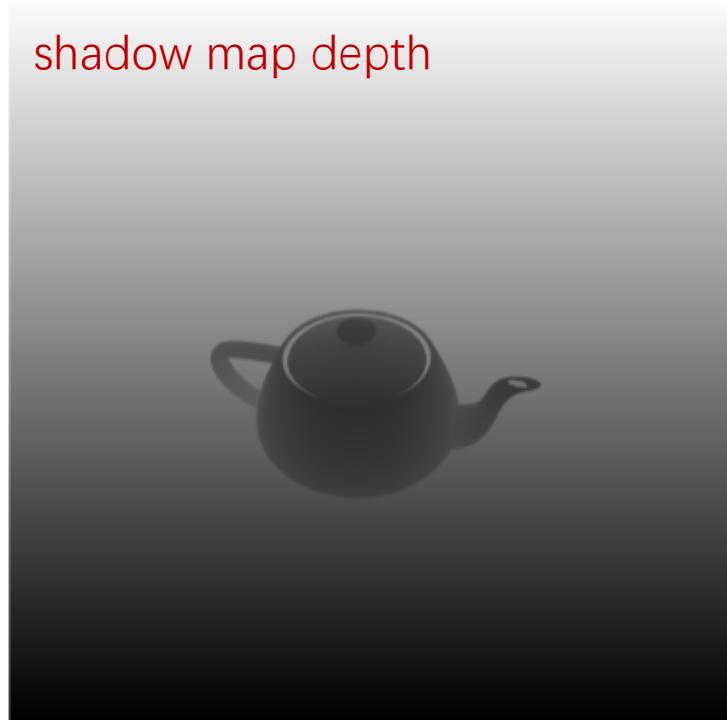


The material you can get online, from [textures.com](https://textures.com)

# Shadow Map



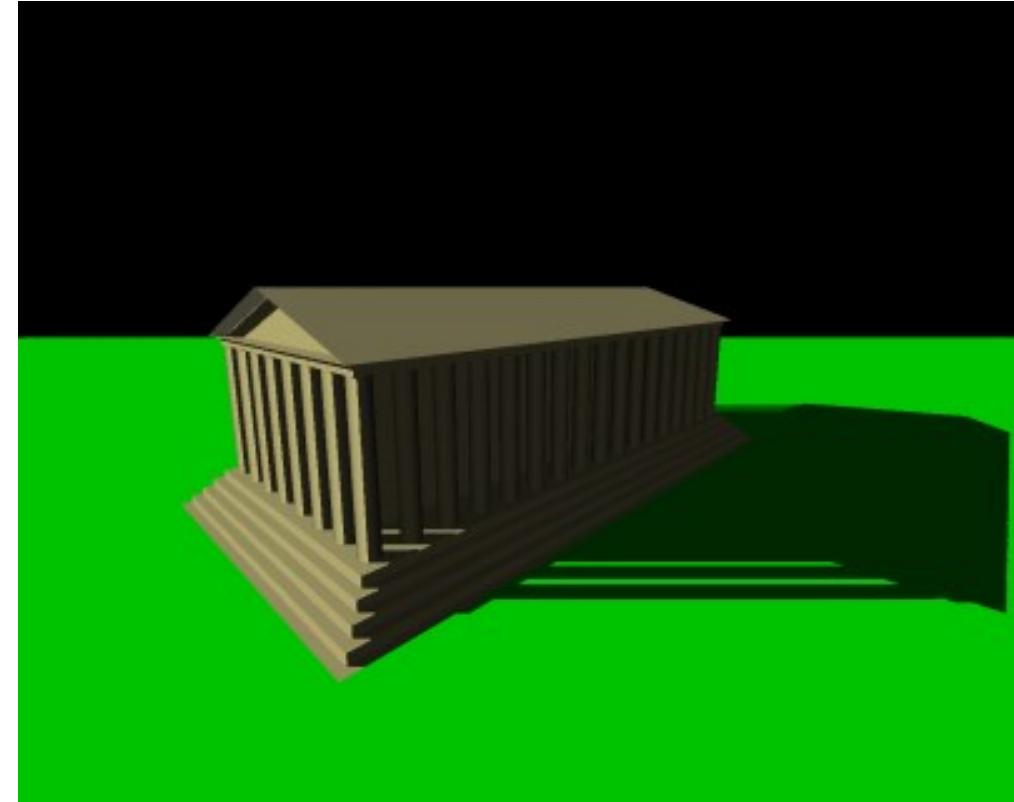
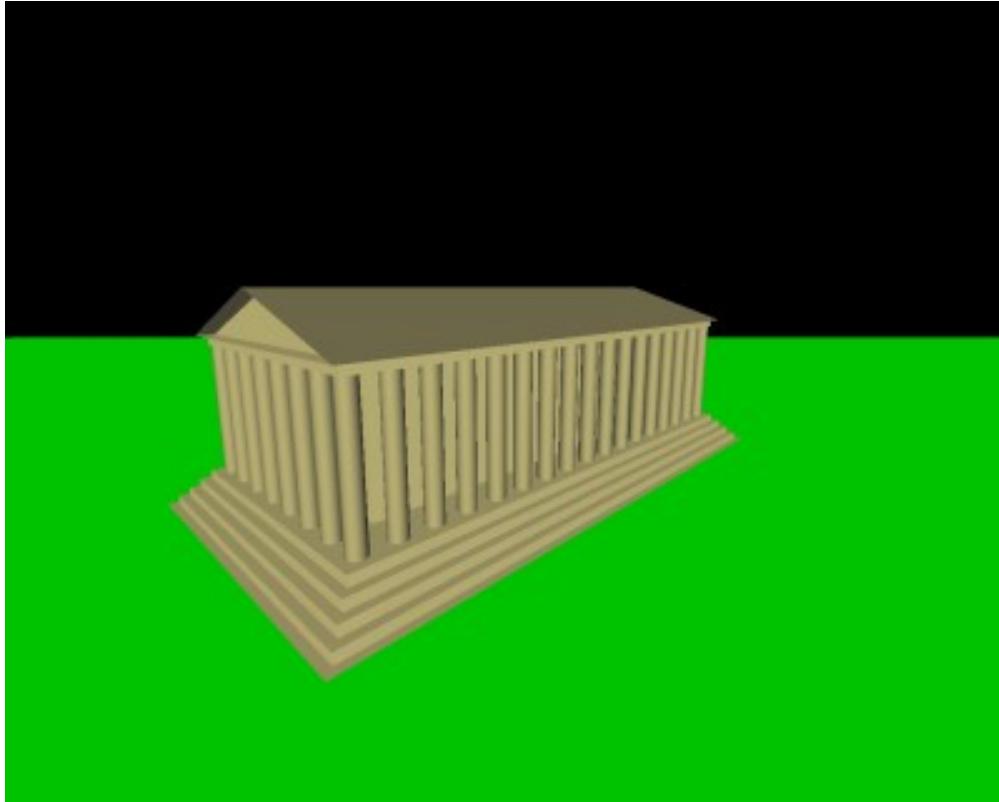
shadow map depth



Rendered from light source

If p's distance to the light is greater than the shadow map depth:  $I_p = k_a I_a$   
Else:  $I_p = k_a I_a + f_{att} I_l (k_d \cos \theta + k_s (\cos \phi)^{n_s})$

# Shadow Map



# Texture Generation

# From Capture



From [medaloot.com](http://medaloot.com)

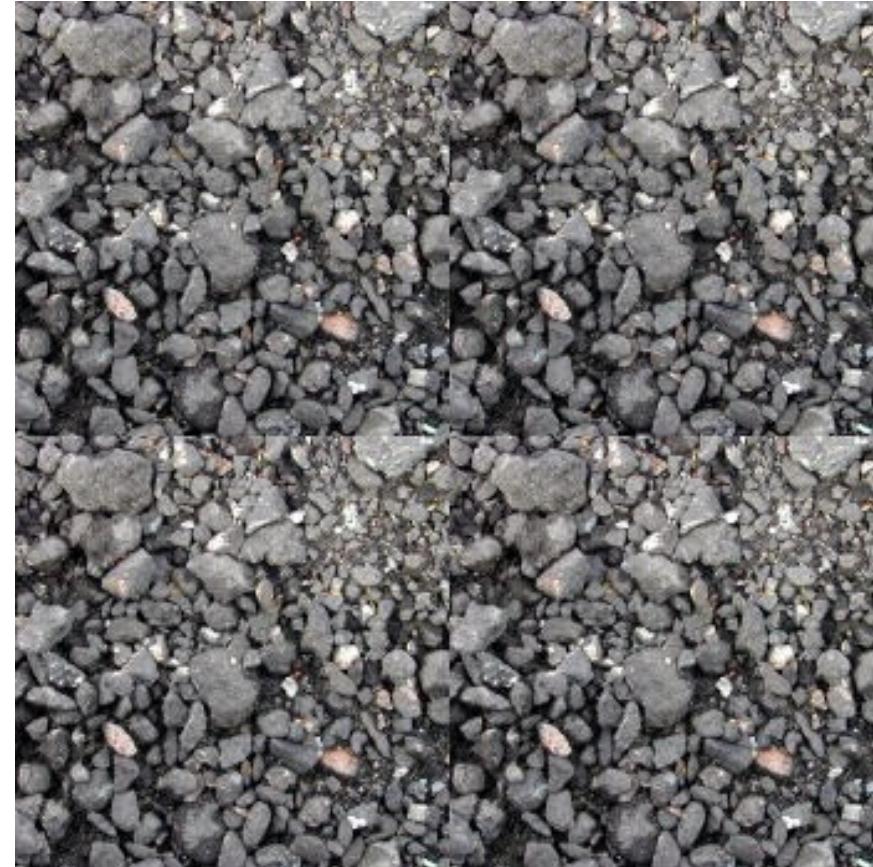


From [forums.sketchup.com](http://forums.sketchup.com)

# How to make it seamless?



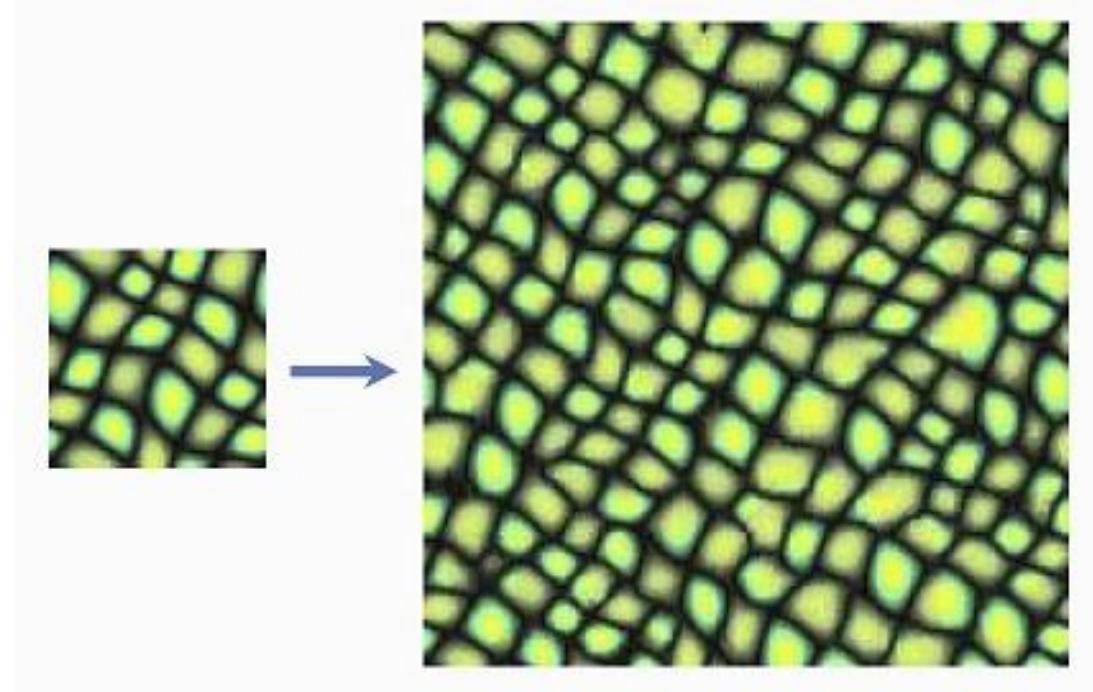
From [medaloot.com](http://medaloot.com)



From [imgonline.com](http://imgonline.com)

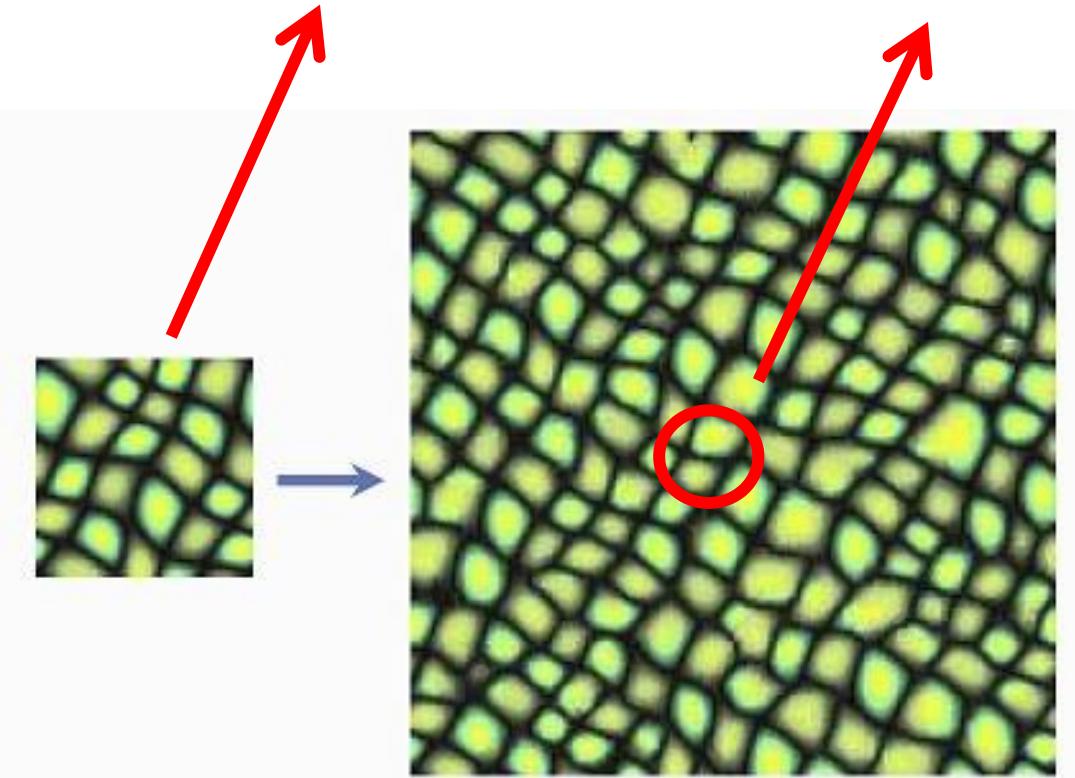
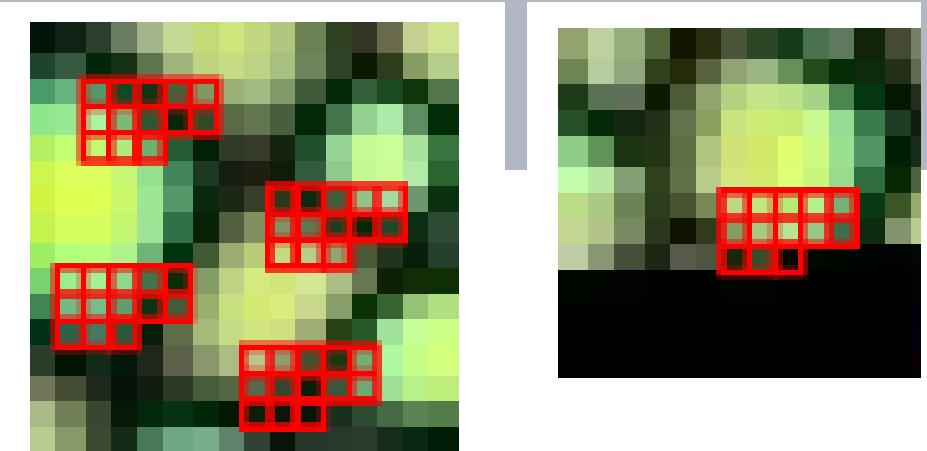
# How to make it seamless?

- Manually take care of the edges, or flip it!
- Texture synthesis (big research subject!)



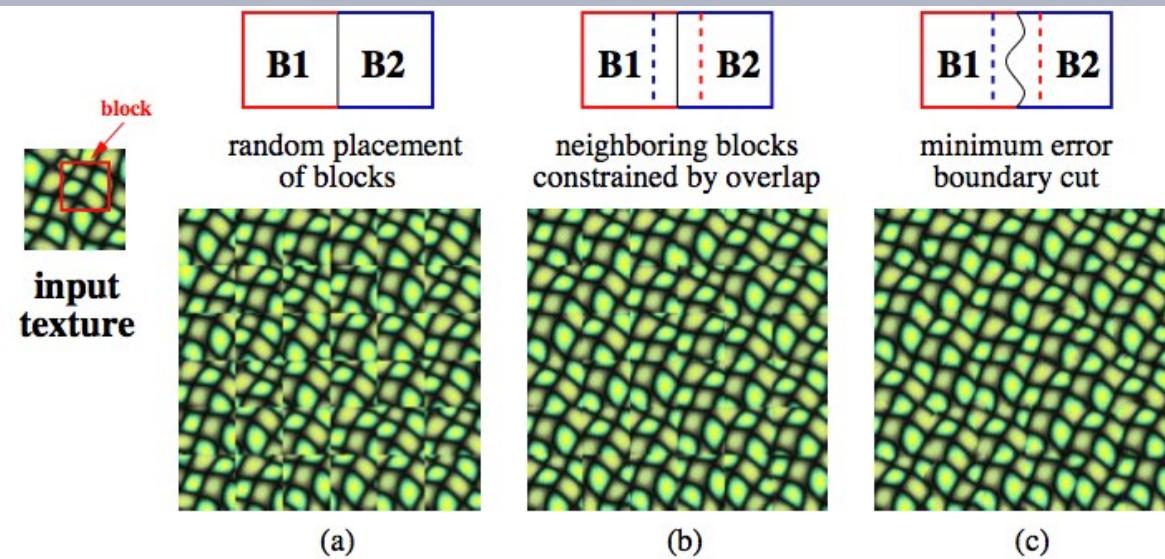
# How to make it seamless?

- Manually take care of the edges, or flip it!
- Texture synthesis (big research subject!)



# How to make it seamless?

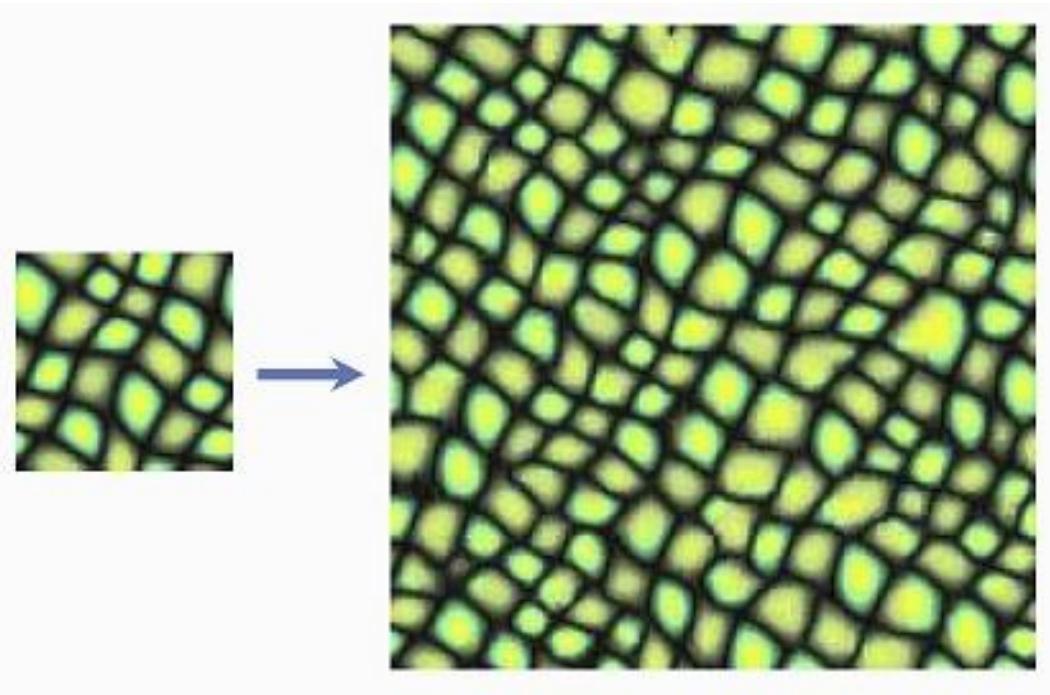
- Manually take care of the edges, or flip it!
- Texture synthesis (big research subject!)



(a)

(b)

(c)



# Procedural Texture

- Created using algorithms (from noise, or cellular automata, or ...)



Perlin noise (naturally for 3D!)



Andreas Bærentzen's self sculpture  
36

# AI Texture Generator

The screenshot shows the AI Texture Generator interface. On the left, there's a preview of a circular texture composed of various autumn leaves. Below the preview are four smaller circular thumbnails showing different textures. At the bottom left is a text input field with the word "leaves". To the right of the input field are two buttons: one with a green "G" icon and another with a blue "Generate" button. The main right side of the interface contains several sections: "Options" with a "Upload image" button, "image strength" slider set to 0.2, and a text input for "Type what you don't want to see in the texture (negative prompt)". Below this is a "Texture resolution" section with options: 512px (selected), 768px, 1024px, 1536px, and 2048px. There's also a "Number of images" section where the number "4" is selected from a row of buttons (2, 4, 6, 8). The bottom section is titled "Example prompts" with two visible entries: "Mossy Runic Bricks, stone, moss" and "Realistic. weathered stone wall".

<https://poly.cam/tools/ai-texture-generator>

# Thanks