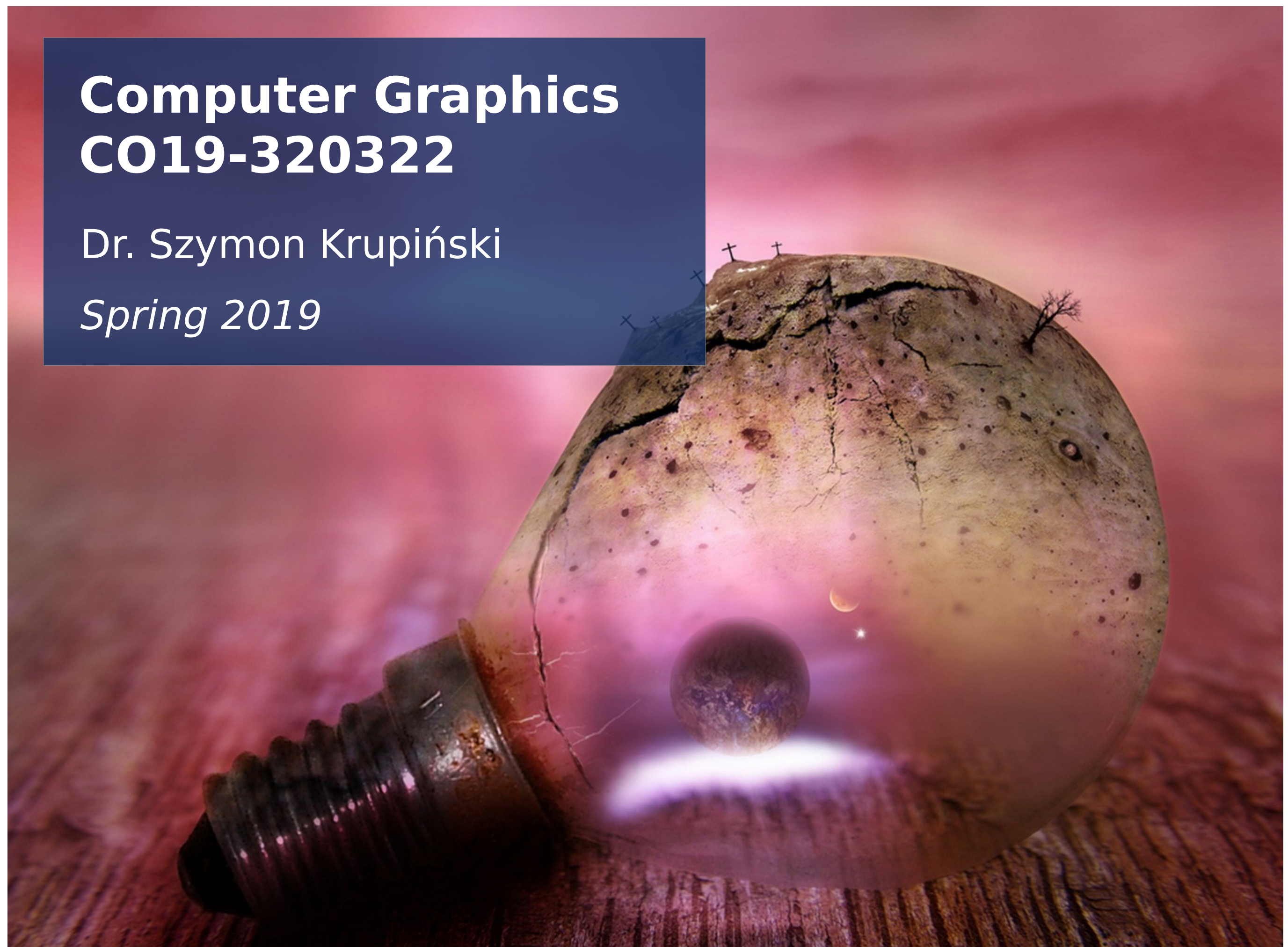
# Lecture 11: Textures and interpolation



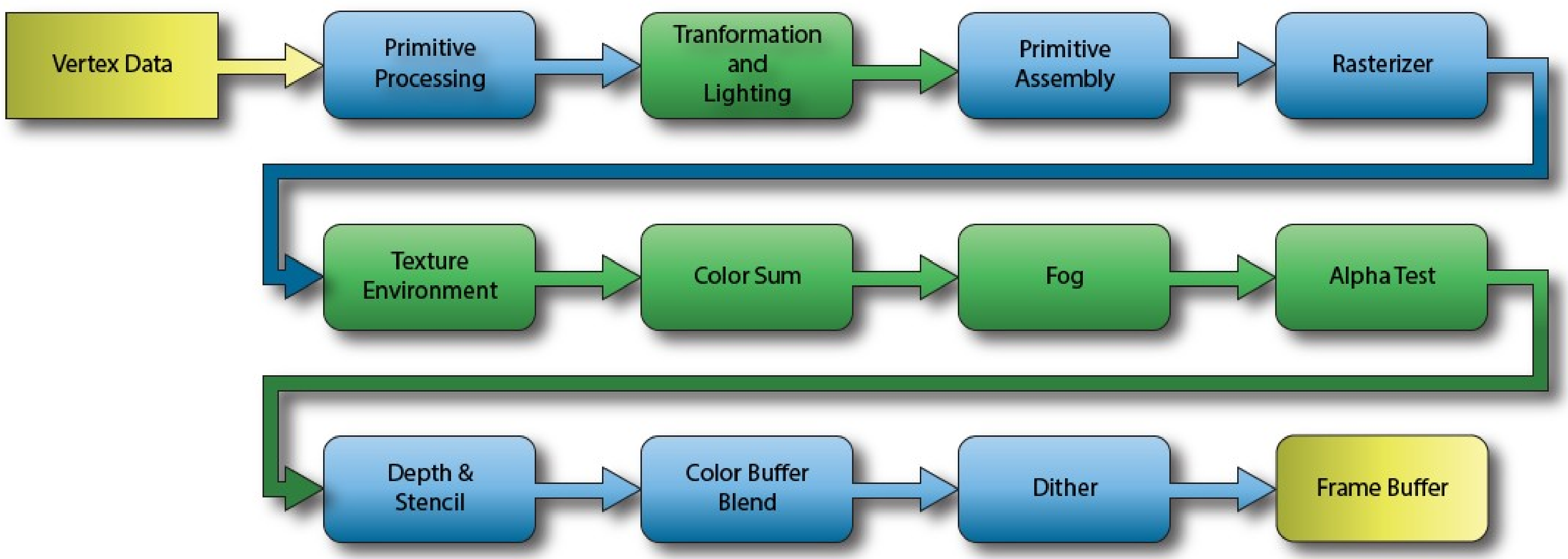
**Computer Graphics**
**CO19-320322**

Dr. Szymon Krupiński

*Spring 2019*

# Where are we?

- Understanding little steps of the rendering pipeline!

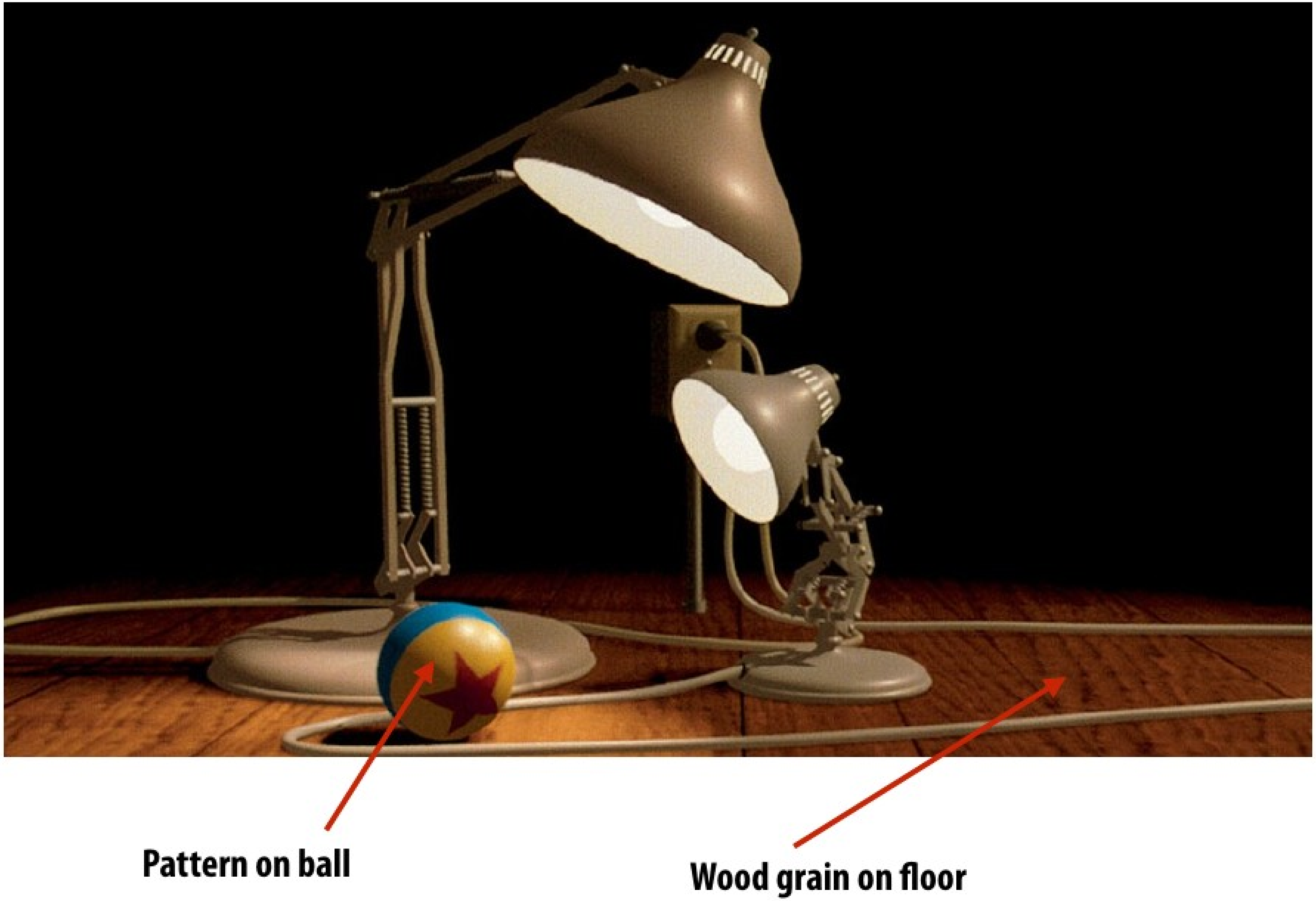



- Today: textures

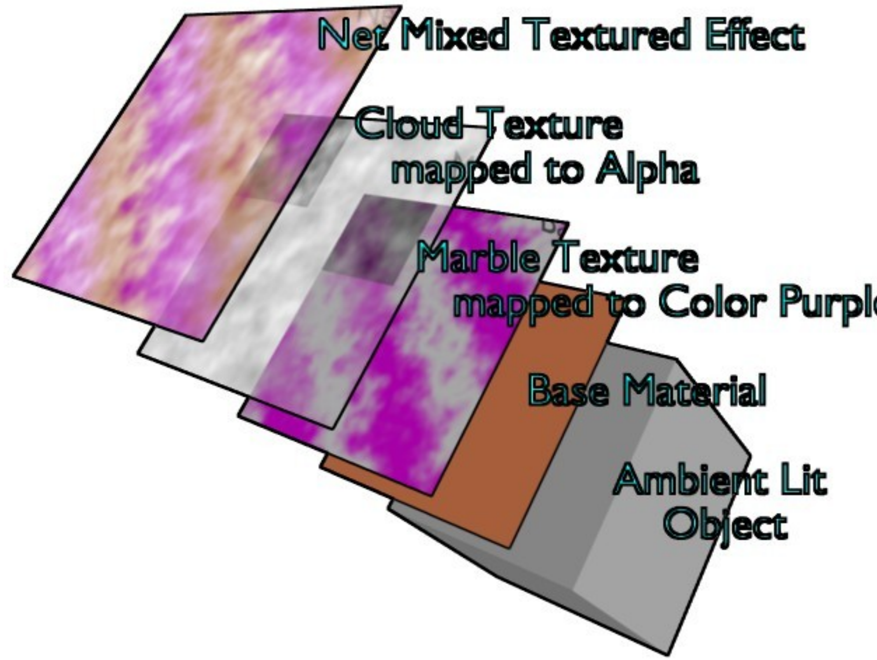
# Texture

- So far we have assumed that our geometry elements were just covered by pure colors
- They can have other attributes!
  - Materials (later) which determine how simulated light interferes with the object
  - Geometry modifications (later) which make their meshes and surfaces more interesting
  - ...and textures!

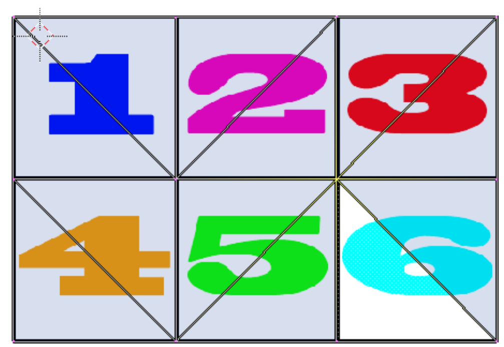
# Texture – use cases

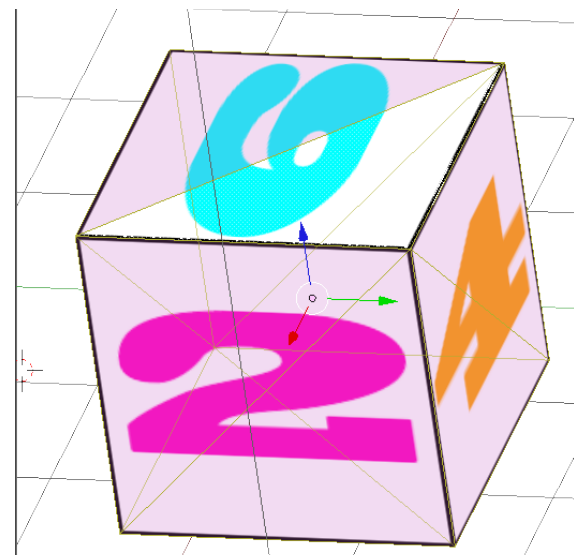


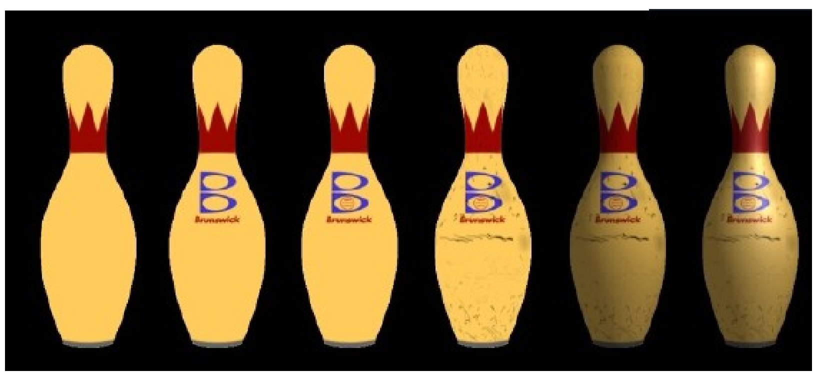
# Texture – use cases




- Can be raster images or procedural
  - From simple checker pattern or stripes to complex designs
  - Random effects: noise, rust, dirt…
- One texture can be designed to cover a complex geometry
- Can be used in a stack; each layer contributes some new details

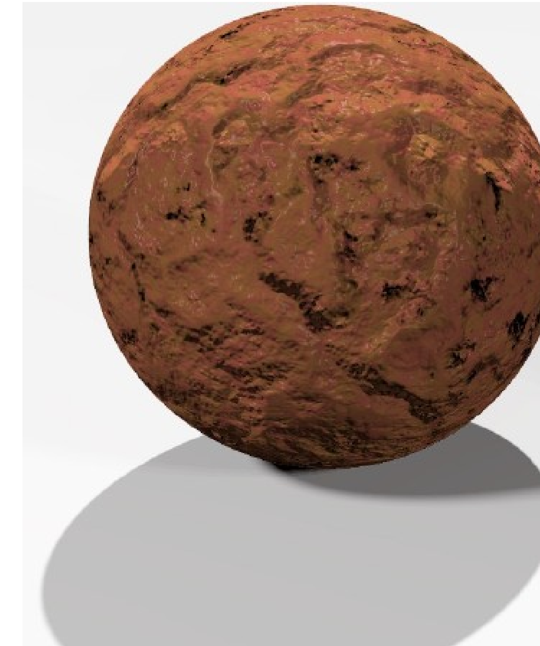






# Texture – use cases

- Textures can be also used to modify the way the geometry is displayed
  - Normal mapping – modifying the surface normals to introduce depth effects on the corresponding pixels
  - Or directly modifying the geometry
- “Baking” the render information into the texture to save calculations during real time display = render mapping

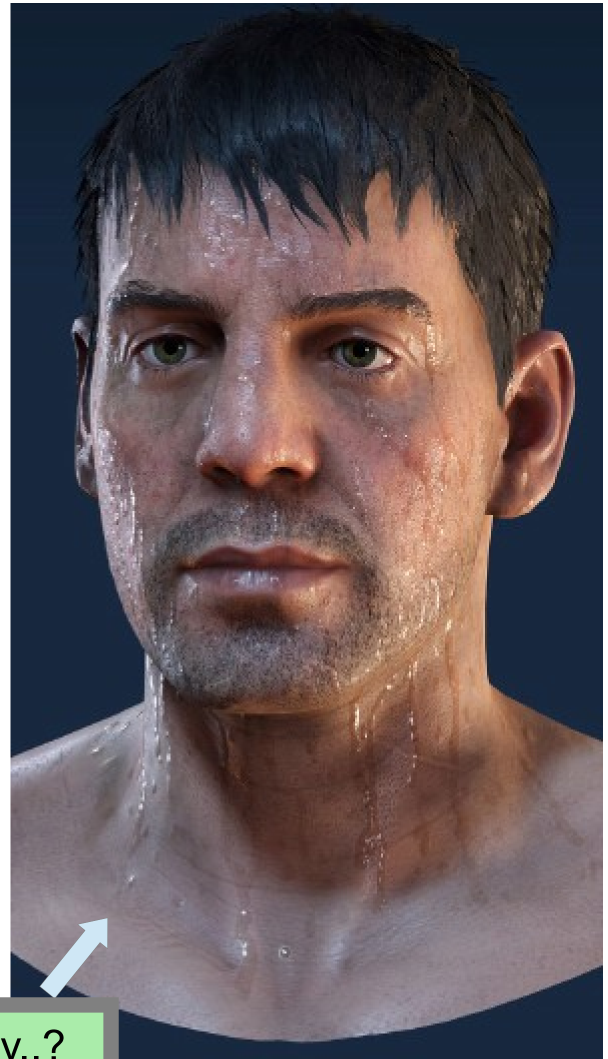


bumps simulated by texture
vs.
bumps in geometry

Original model With ambient occlusion Extracted ambient occlusion map

# Texture – caveats

- Textures can contribute greatly to the realistic looks of models or to making the scene more interesting
- But they are also a source of negative artifacts like aliasing
- They add complexity to the project

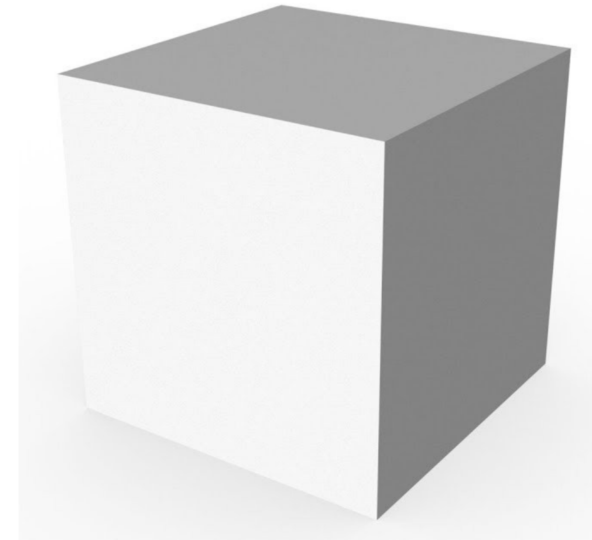



This level of fidelity..? Probably not real-time rendering.

# Texture – caveats

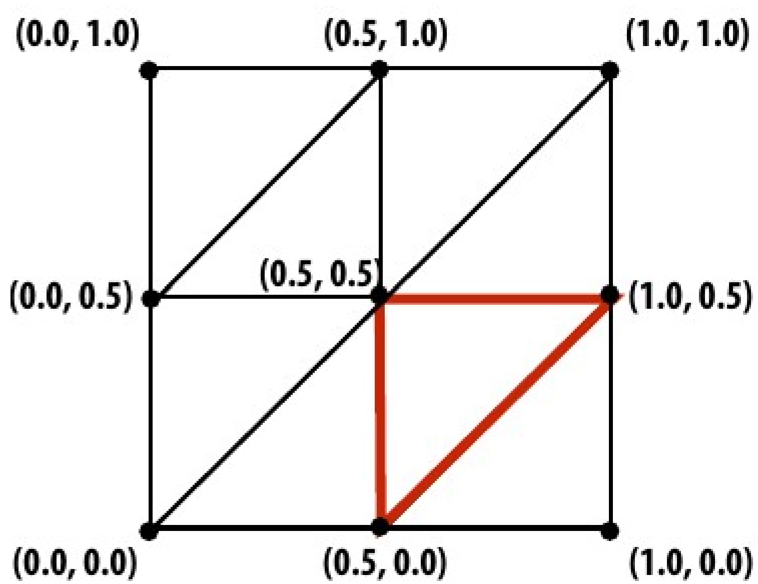
- Textures represent a significant amount of data
  - Colored cube on screen:
    8*(3 coordinates [2 bytes] + 3 color coordinates [1 byte]) = 72 bytes)
  - Textured cube on screen:
    (assuming 300 px side resolution)
    72 bytes + 6*300*300*(3 color coordinates [1 byte]) ≈ 1.5 Mb
  - This comparison is not fair (why?) but not far from reality either

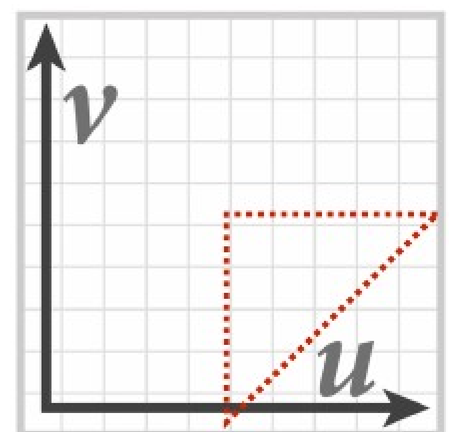
➔ Big load for the GPU and data bus

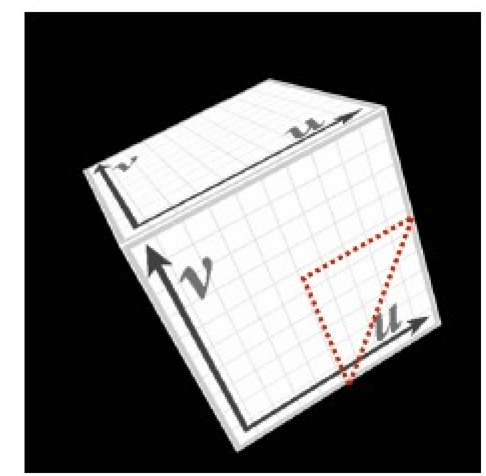


# Texture mapping

- “Texture mapping” defines a mapping from geometry surface coordinates (points on triangle) to points in texture domain
  - Texture coordinates denoted (u,v) are typically expressed in [0,1]



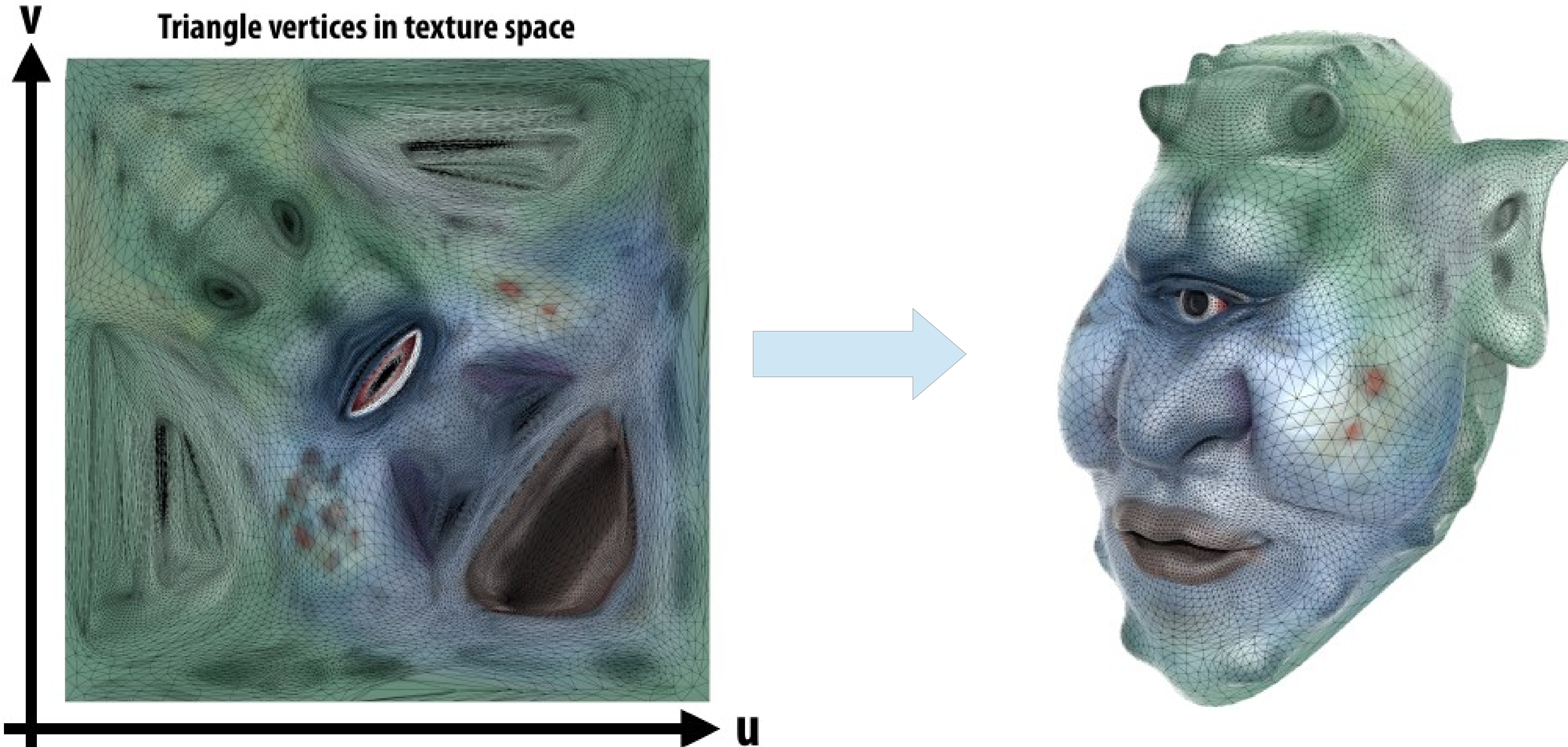




- We need a function translate vertex coordinates (x,y) to texture coordinates (u,v) and another function which will give us a pixel color based on (u,v)

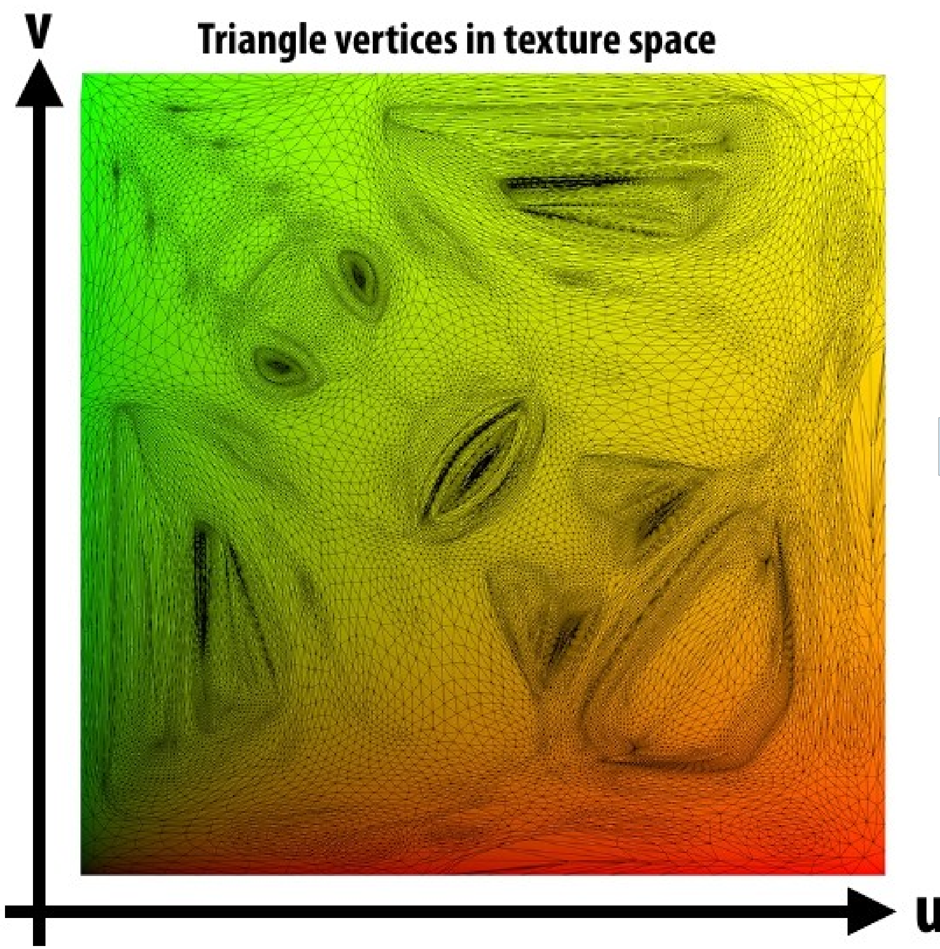
# Texture mapping

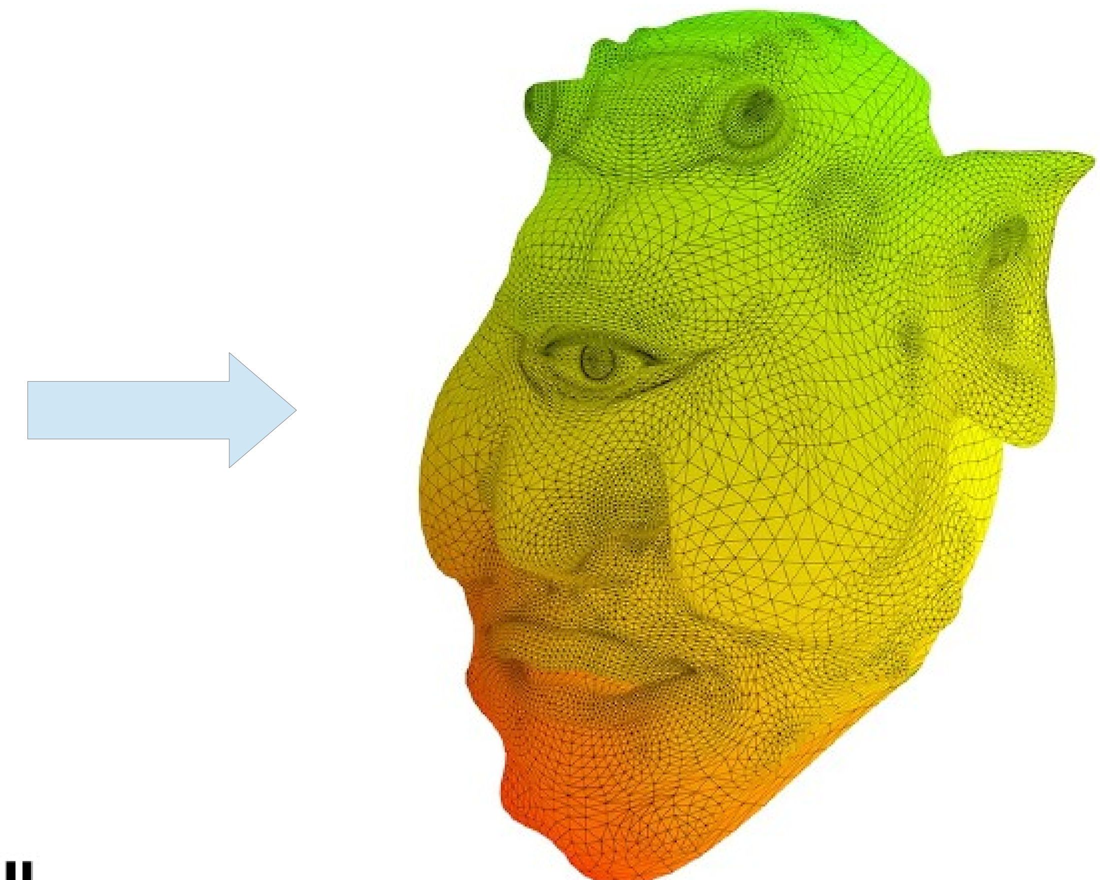
- The cube case was very simple!



# Texture mapping

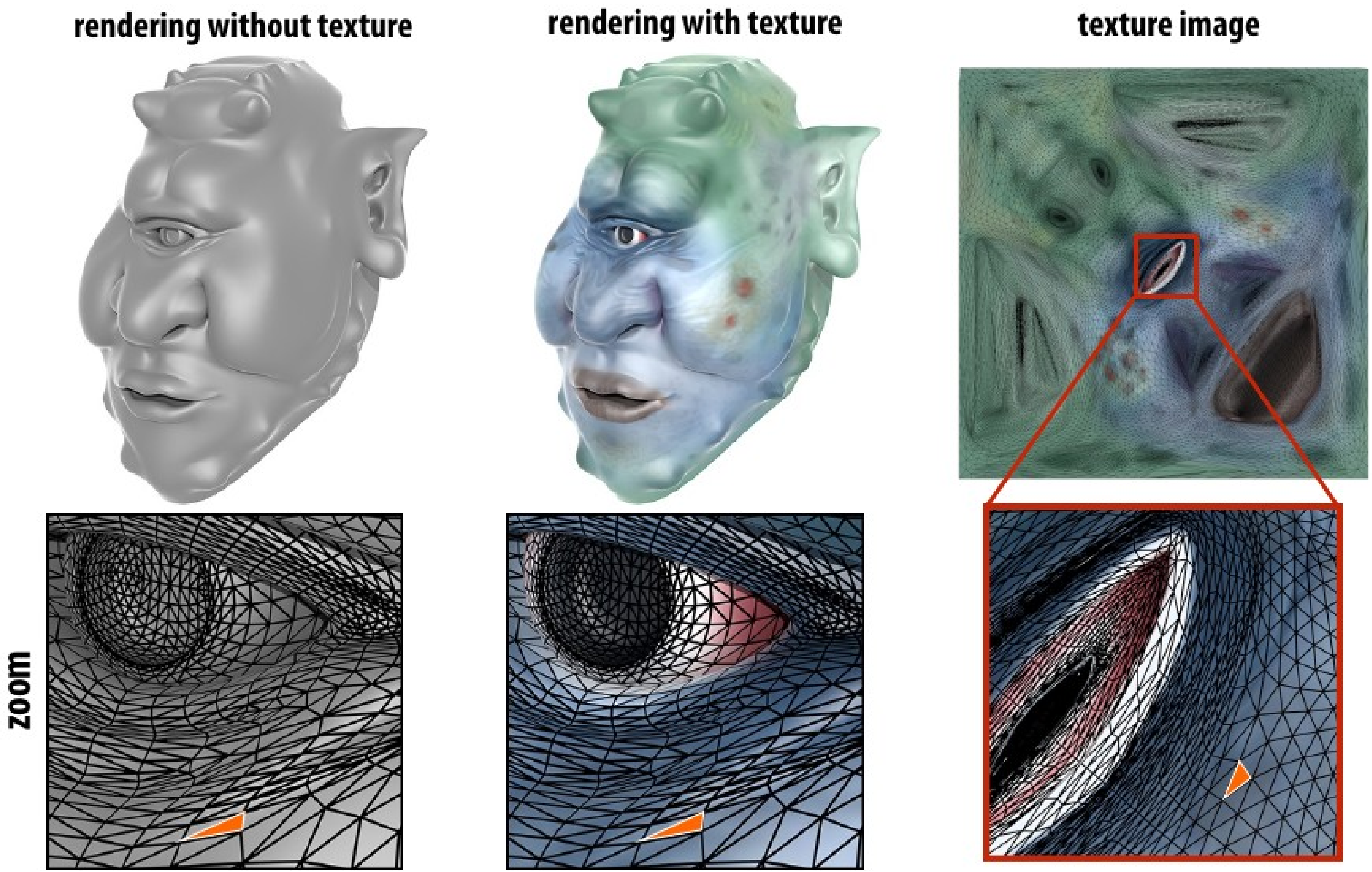
- How are the (u,v) coordinates expressed in this case?



# Texture mapping

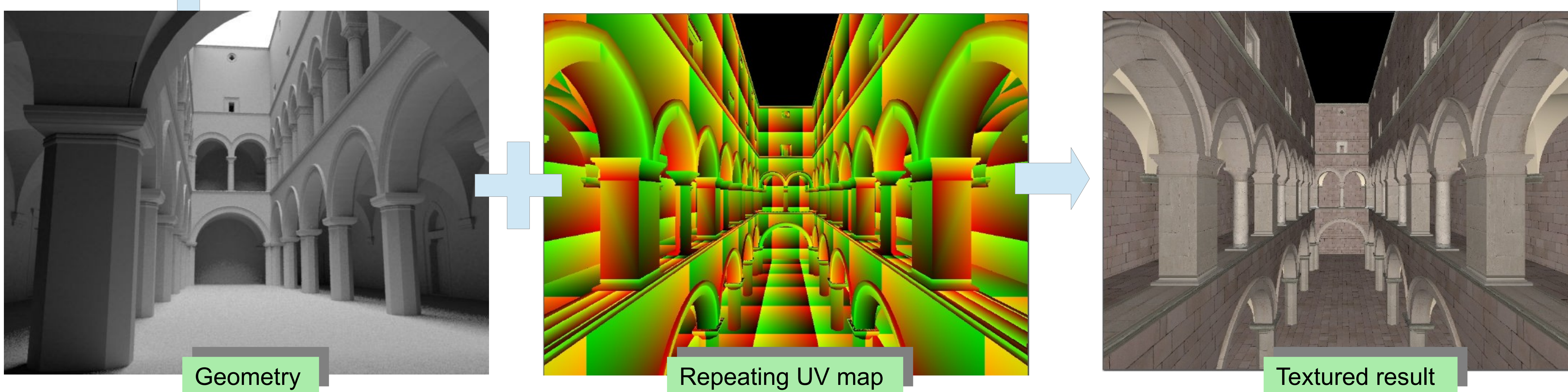
- The



# Texture mapping

- Exact mapping of the entire object can be avoided, e.g. by texture wrapping + repeat:

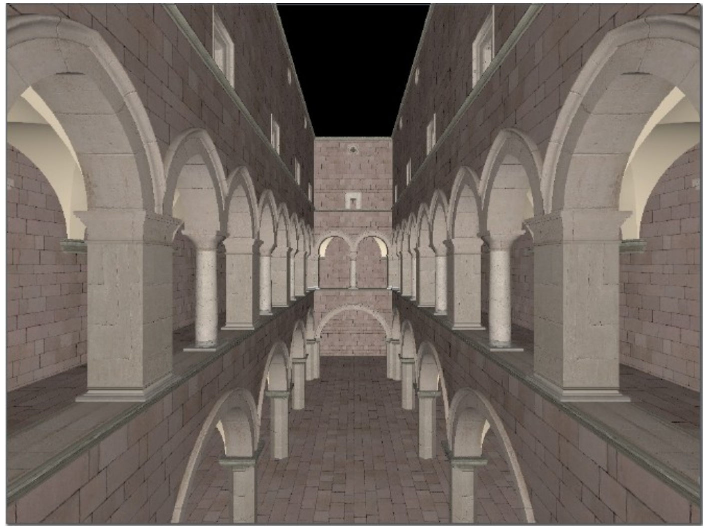

  - (u,v) coordinates are “wrapped” by mapping the [0,1] range repeatedly for values bigger than 1.0



# Texture mapping

- The rendering algorithm at this stage:

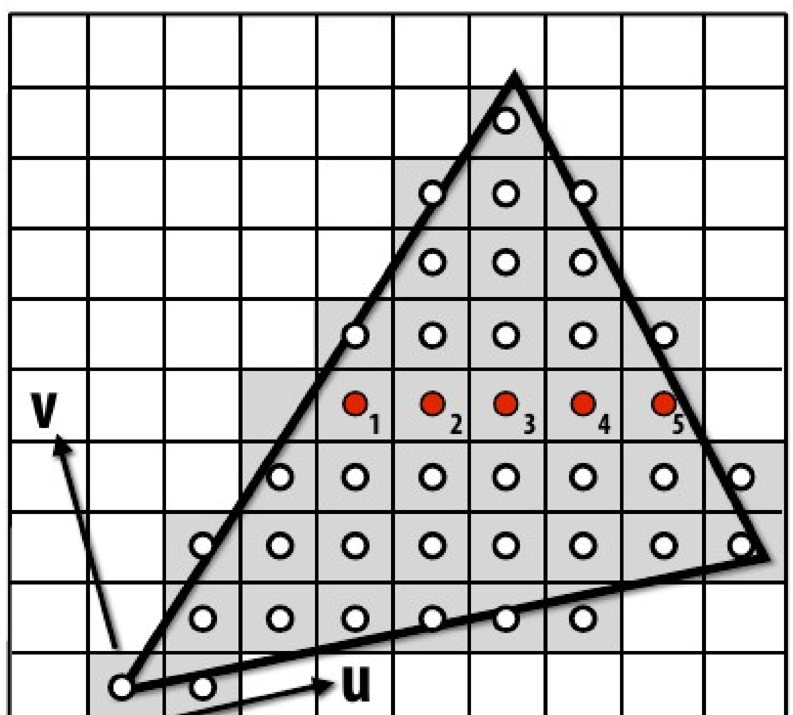
```
for each screen sample point(x,y) {
    // We know this part!
    check_coverage(x,y);
    check_occlusion(x,y);

    (u,v) = evaluate_texcoord_value_at(x,y);
    float texcolor = texture.sample(u,v);
    set_sample_color(texcolor);
}
```

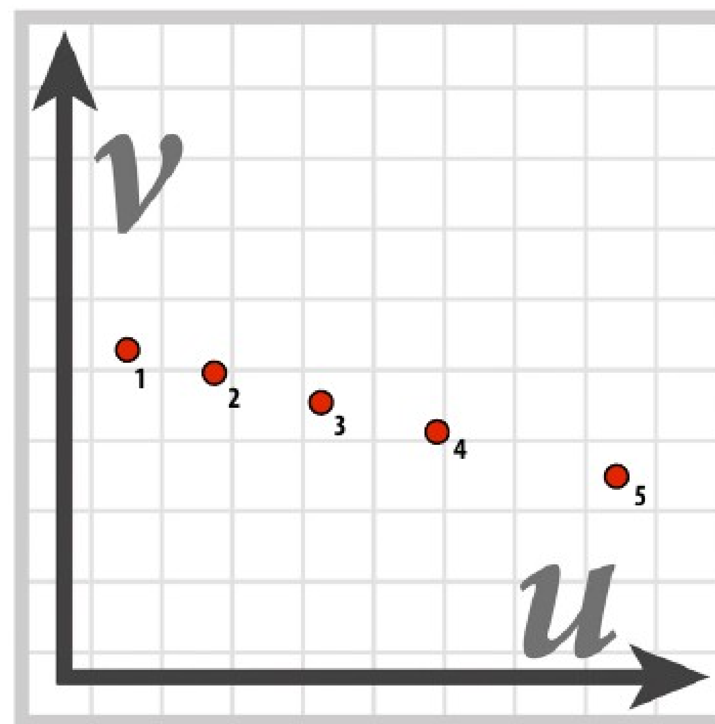


- The triangle which we are trying to render is probably transformed and projected. How does it affect the texture?

# Texture mapping

- The effects of projecting the geometry affects how the textures are sampled:

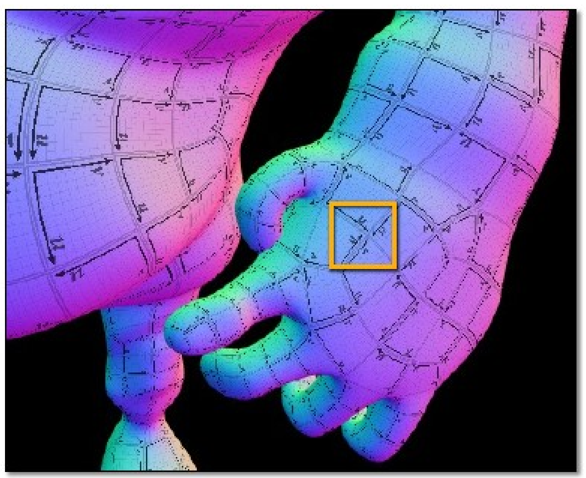


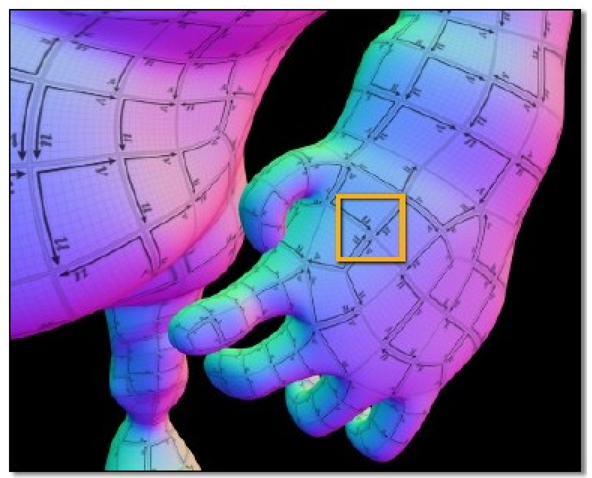

➔ Sampling theory comes back!

# Texture aliasing

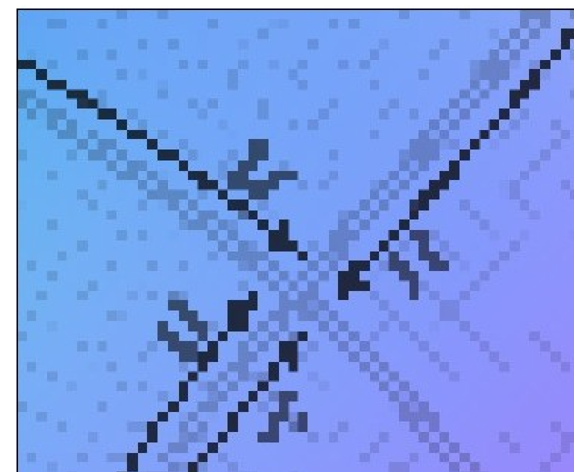
- Sampling can again bring aliasing
  - Solution: reduce the resolution (=cut down the high frequency!)



No pre-filtering of texture data (resulting image exhibits aliasing)



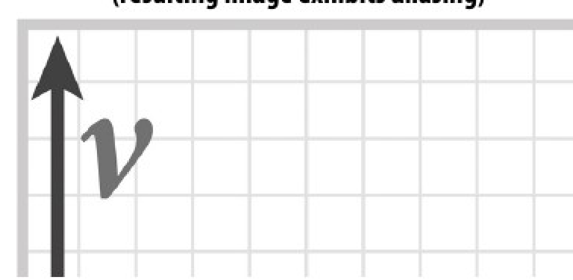
Rendering using pre-filtered texture data



No pre-filtering of texture data (resulting image exhibits aliasing)

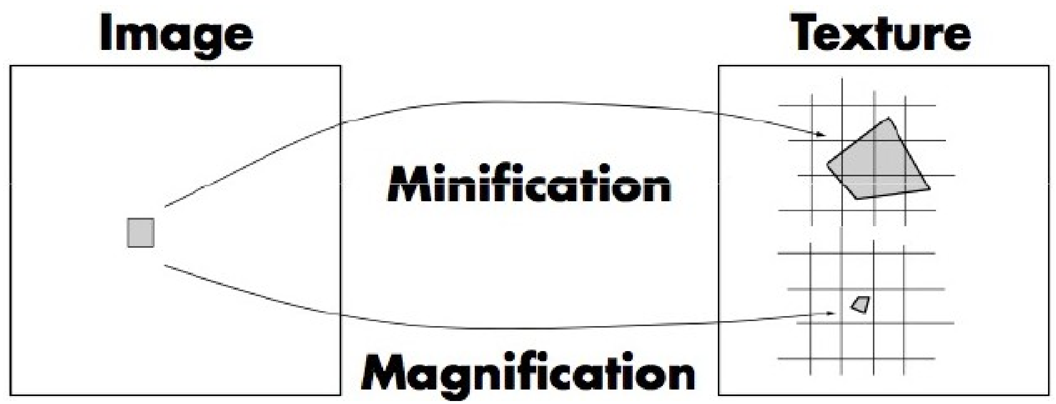
Rendering using pre-filtered texture data






# Texture filtering

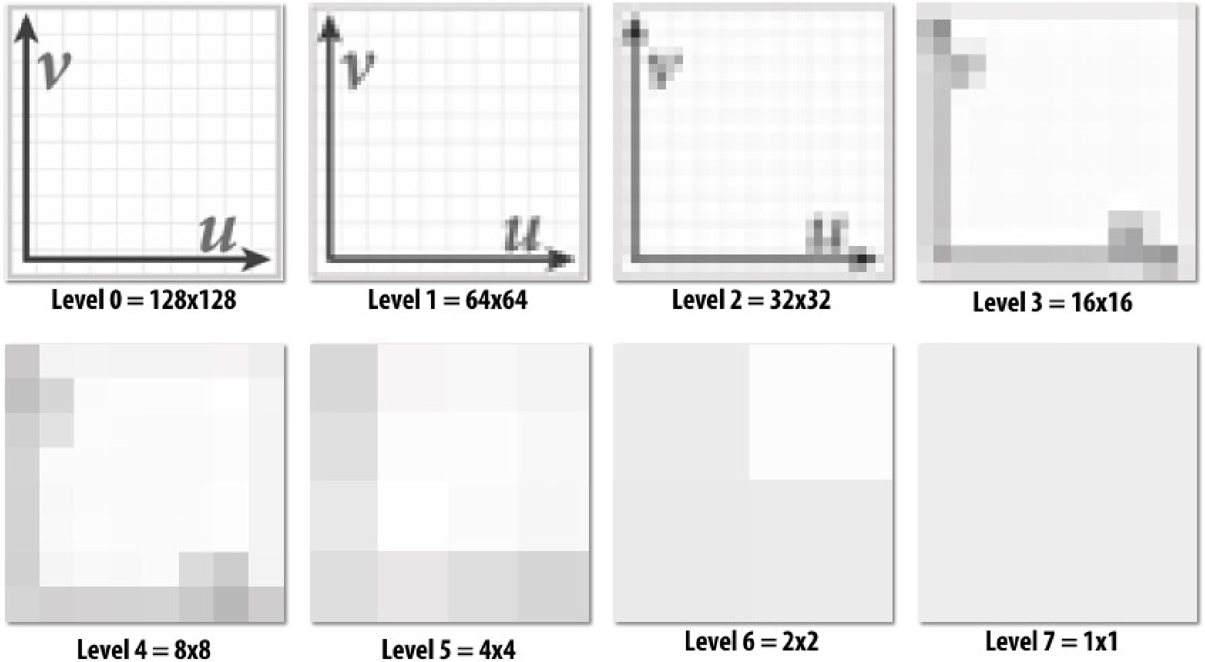
- We might need to juggle with the resolution of the texture




- Minification
  - Area of screen pixel maps to large region of texture (filtering required, e.g. averaging)
  - One **texel** corresponds to far less than a pixel on screen
  - Example: when scene object is very far away
- Magnification
  - Area of screen pixel maps to tiny region of texture (interpolation required)
  - One texel maps to many screen pixels
  - Example: when camera is very close to scene object (need higher resolution texture map)

# Mipmap

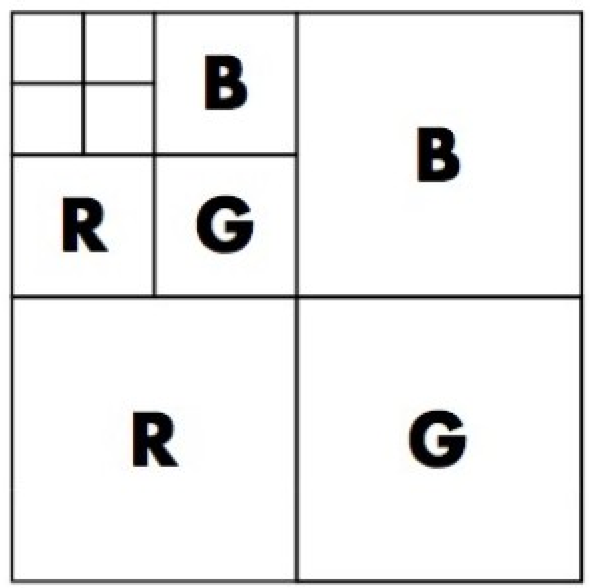
- Pre-calculate and store different resolutions levels of every texture (Lance Williams, 1983)



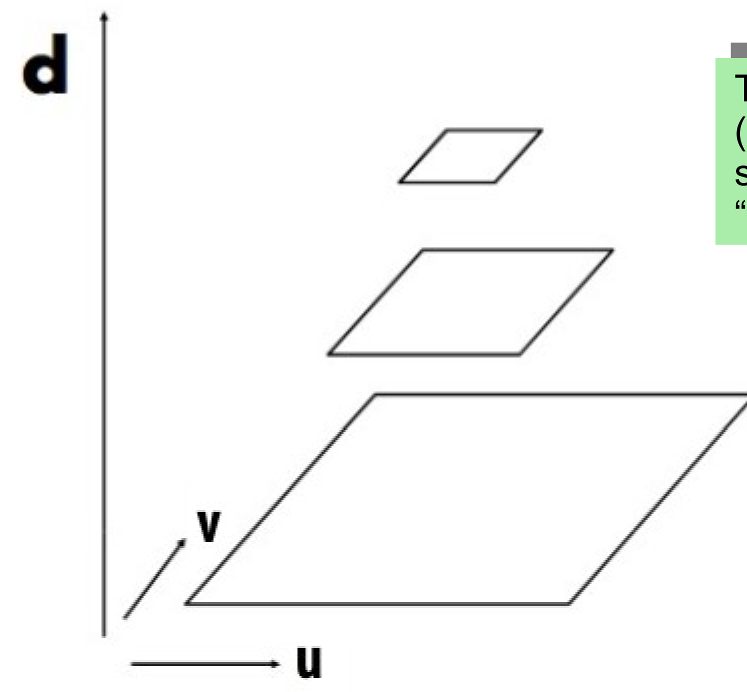

  - One of the reasons why until today we tend to store textures in chunks of size 2^n x 2^m pixels (not necessarily square)
  - Calculations like preparing mipmaps and operations related to UV were very deeply optimised in hardware
  - today, this restriction is often no longer in place but for the sake of compliance, the practice persists

# Mipmap

- The idea comes with an extreme level of optimisation!



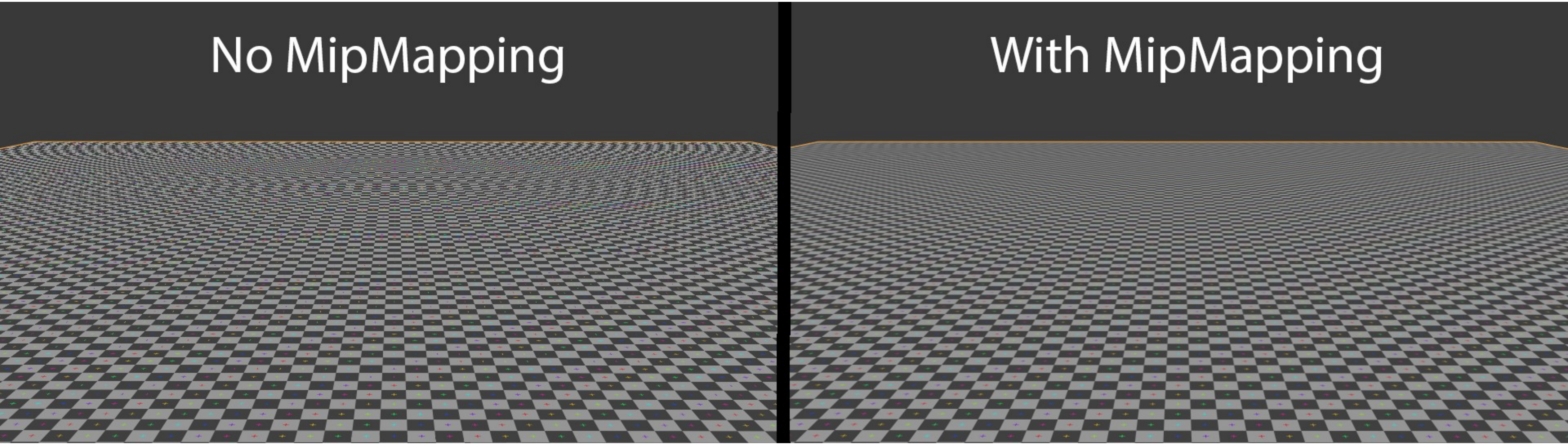

Williams' original proposed
mip-map layout




"Mip hierarchy"
level = *d*

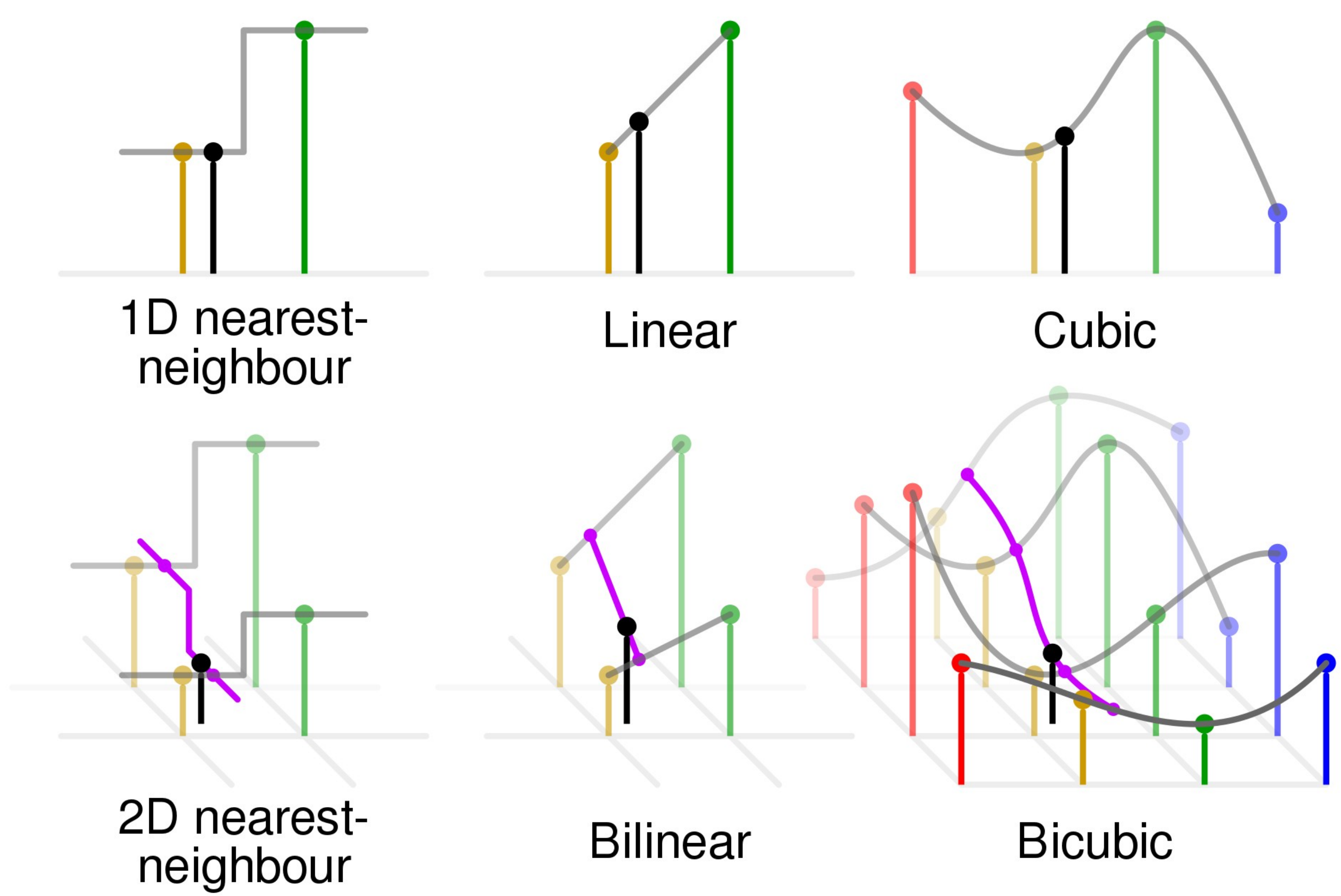
That's why mipmaps (or MIP maps) are sometimes called "pyramids"

# Mipmap - effect




- (will come handy in level of details (LOD) mapping...)

# Interpolation

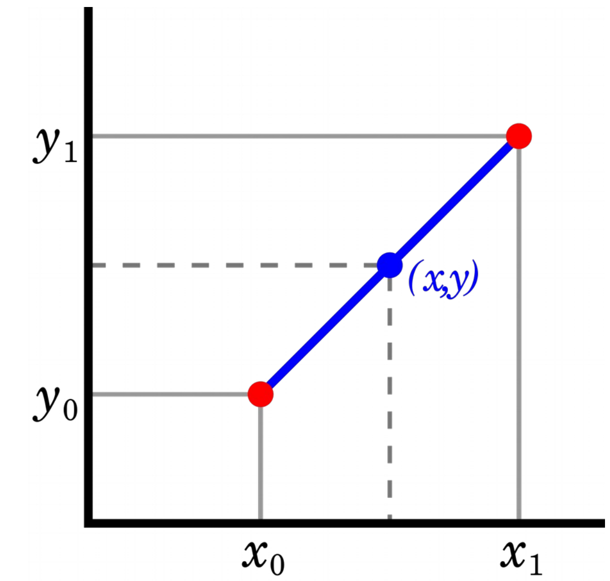


# Linear interpolation

- Easy calculation between two known values, given the distance on a straight line between them
- We have seen an example when discussing color and alpha management

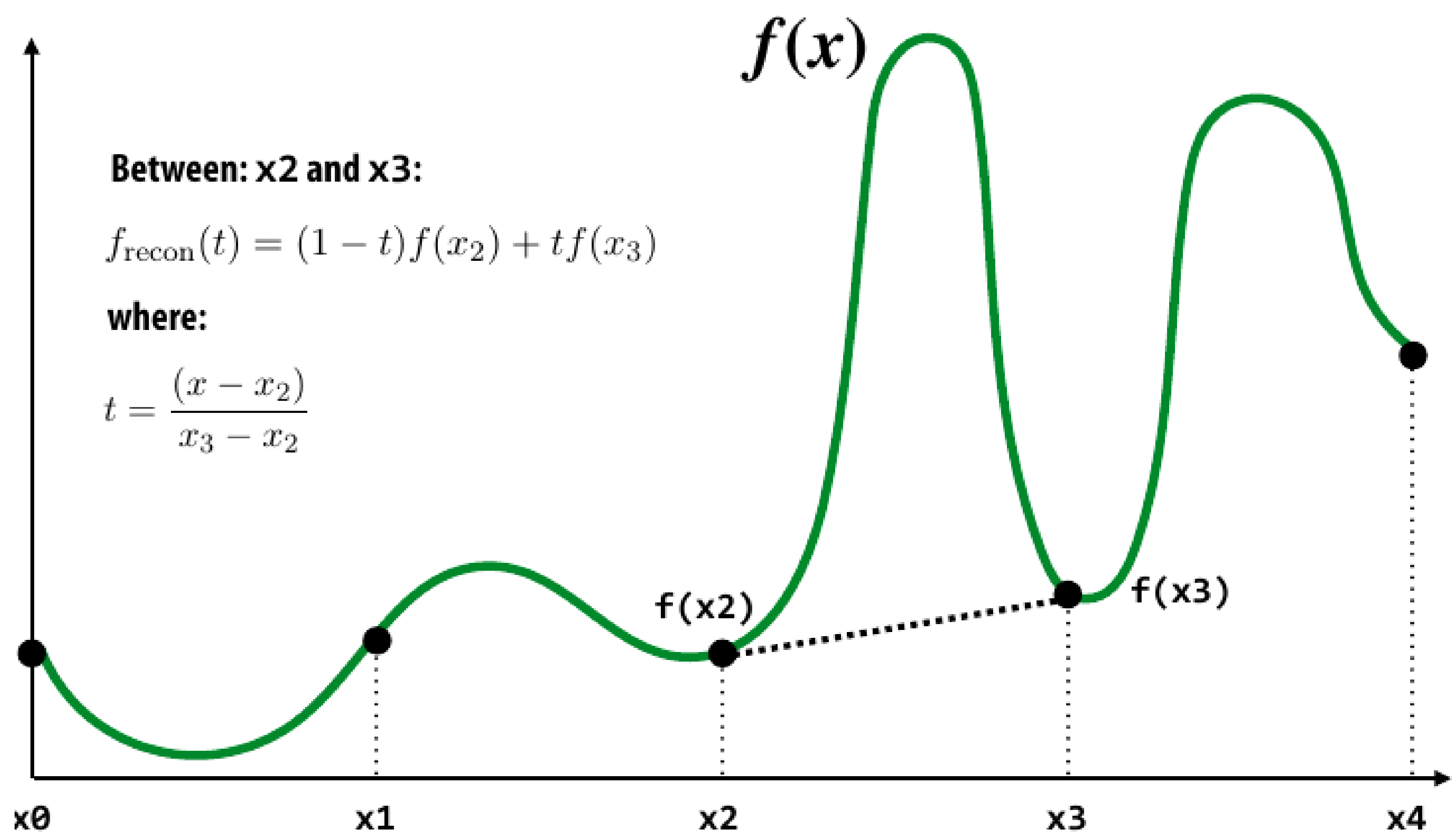
$$\alpha_C = \alpha_B + (1 - \alpha_B)\alpha_A$$

  - Attention: it was easier because we knew we were dealing with a value bounded in [0,1]



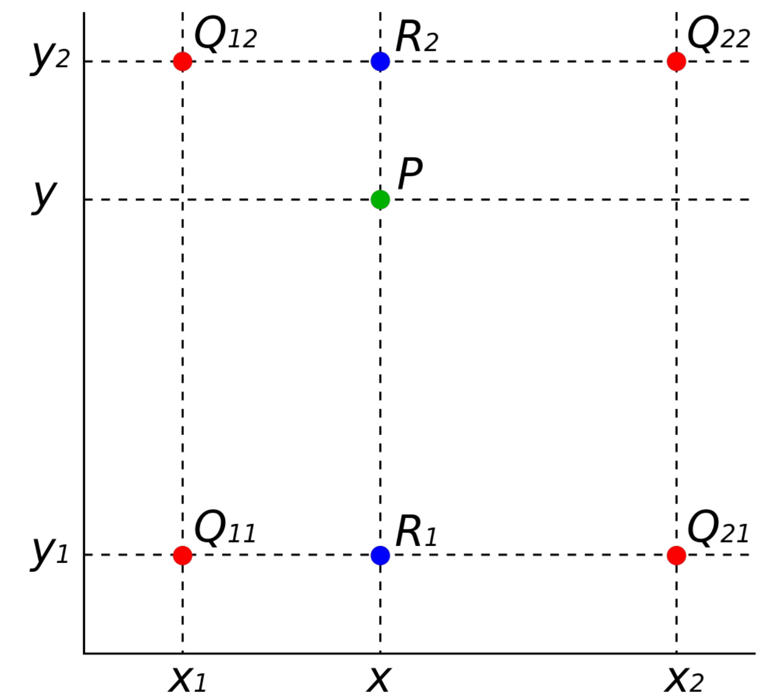

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0},$$

# Linear interpolation

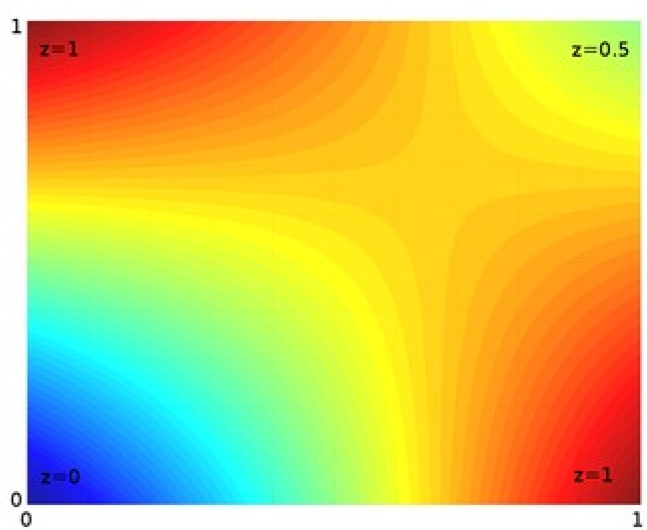


# Bilinear interpolation

- Linear interpolation is great for 1-D calculation. What about 2-D?
- We have reference values in four corners and want to evaluate a value at any point *(x,y)*
- Example: color interpolation!
(that's how it was probably done in the example code we saw so far)




$$
\begin{aligned}
f(x, y_1) &\approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}),\\
f(x, y_2) &\approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}).
\end{aligned}
$$




$$
\begin{aligned}
f(x, y) &\approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2)\\
&= \frac{y_2 - y}{y_2 - y_1} \left( \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \right) + \frac{y - y_1}{y_2 - y_1} \left( \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \right)\\
&= \frac{1}{(x_2 - x_1)(y_2 - y_1)} \big( f(Q_{11})(x_2 - x)(y_2 - y) + f(Q_{21})(x - x_1)(y_2 - y) + f(Q_{12})(x_2 - x)(y - y_1) + f(Q_{22})(x - x_1)(y - y_1) \big)\\
&= \frac{1}{(x_2 - x_1)(y_2 - y_1)} \begin{bmatrix} x_2 - x & x - x_1 \end{bmatrix} \begin{bmatrix} f(Q_{11}) & f(Q_{12}) \\ f(Q_{21}) & f(Q_{22}) \end{bmatrix} \begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix}.
\end{aligned}
$$

# Thank you!

- Questions?