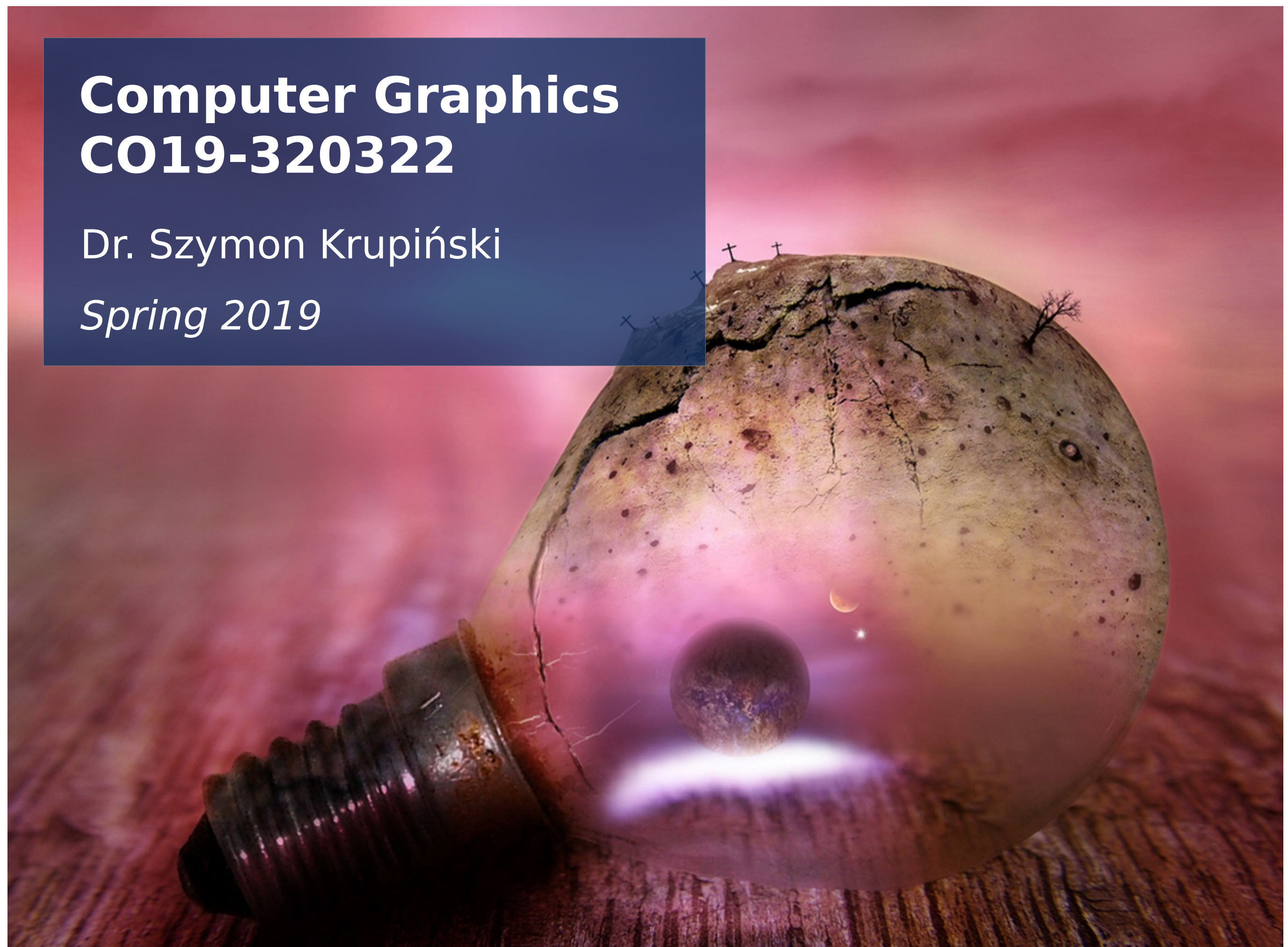
# Lecture 12: Textures and interpolation

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

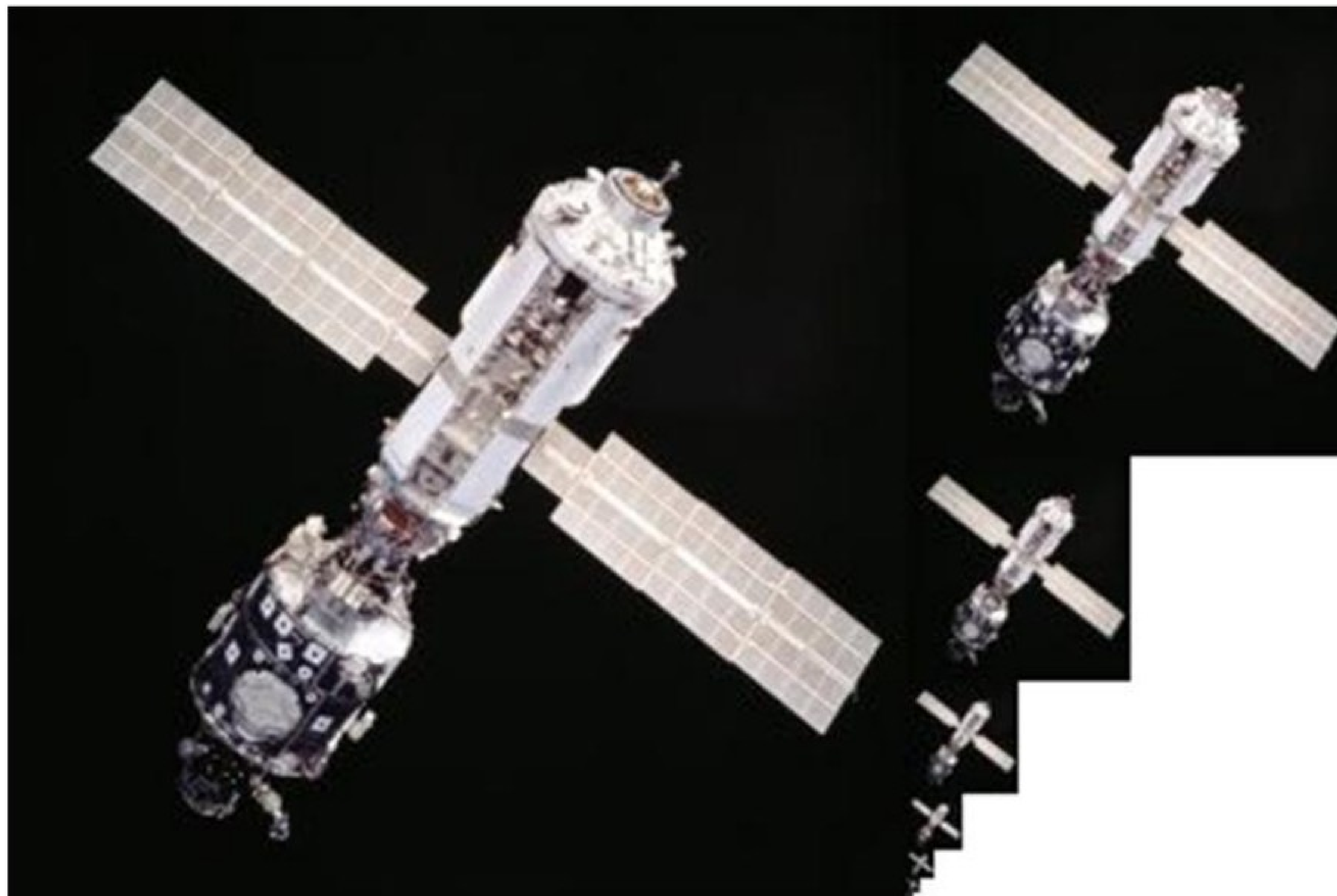
Dr. Szymon Krupiński

*Spring 2019*



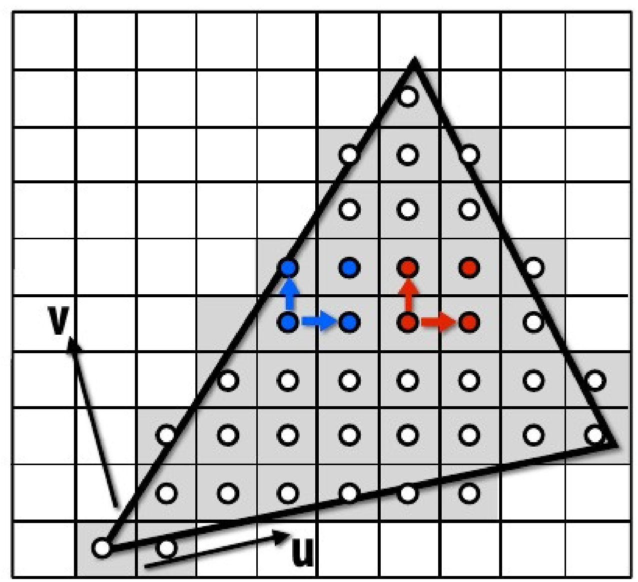
# Back to mipmaps

- Mipmap packing:

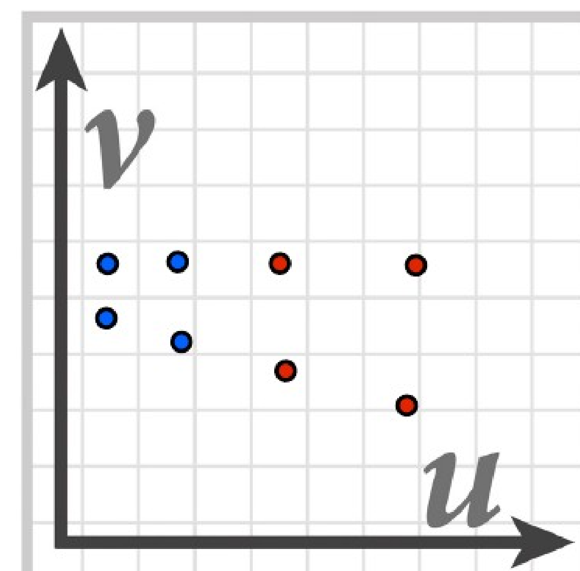


# How to compute d for Mipmaps?

- How to decide which level of resolution d to use for a mipmap?
  - let's look at how the mapping from screen (x,y) to texture (u,v) varies together using derivatives
  - Construct variable $d$ which will be used to select the mipmap level

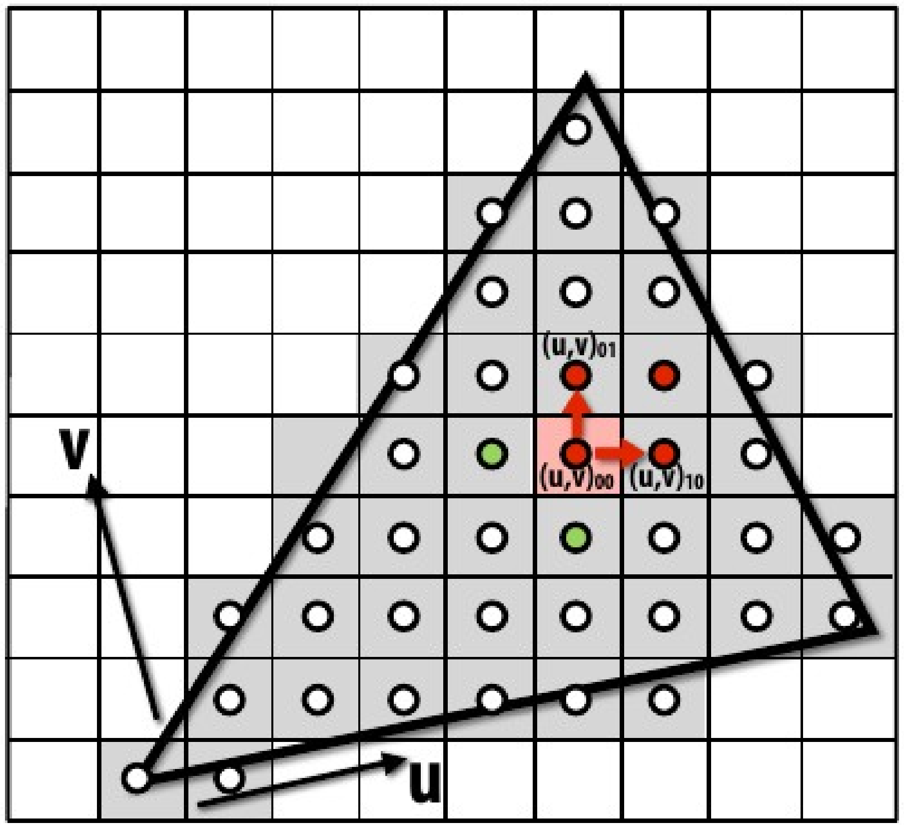


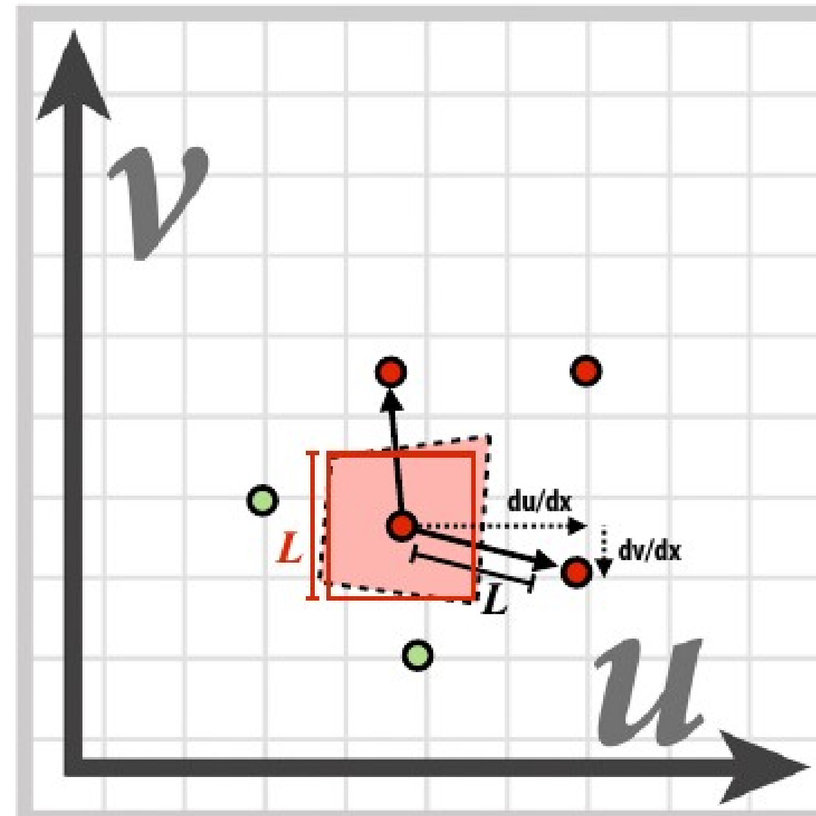
Screen space



Texture space

# How to compute d for Mipmaps?




$$\frac{du}{dx} = u_{10} - u_{00} \qquad \frac{dv}{dx} = v_{10} - v_{00}$$

$$\frac{du}{dy} = u_{01} - u_{00} \qquad \frac{dv}{dy} = v_{01} - v_{00}$$

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

$$d = log_2 L$$

# Mipmaps

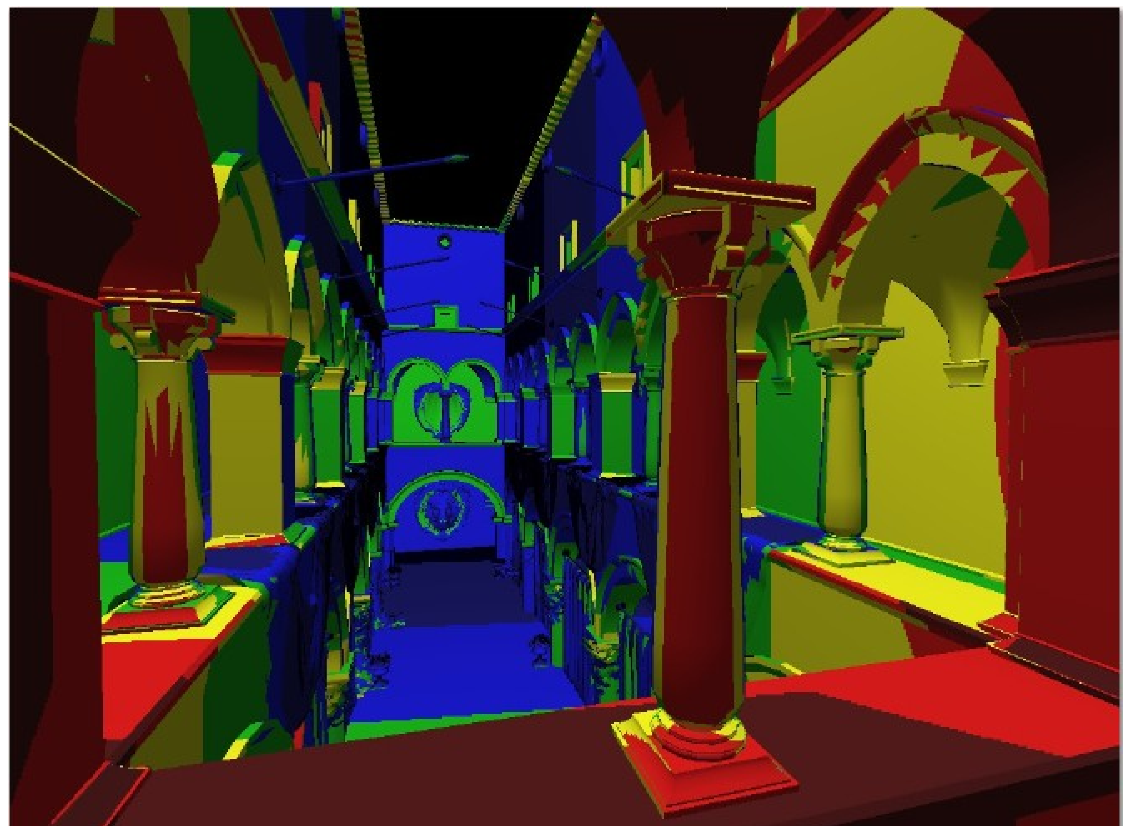
- Effect of (bilinear) sampling at different levels of $d$

d=2

d=4

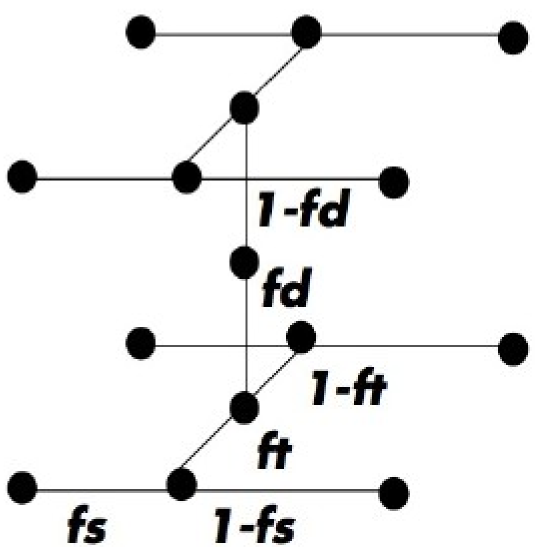
# Mipmaps

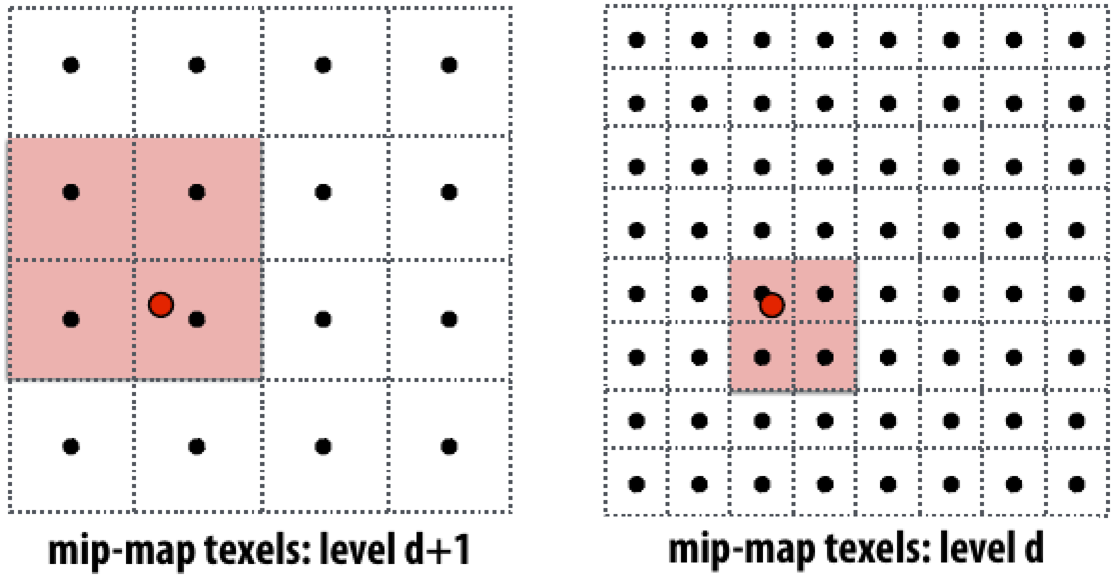
- Which level of $d$ do we need?



# Mipmaps

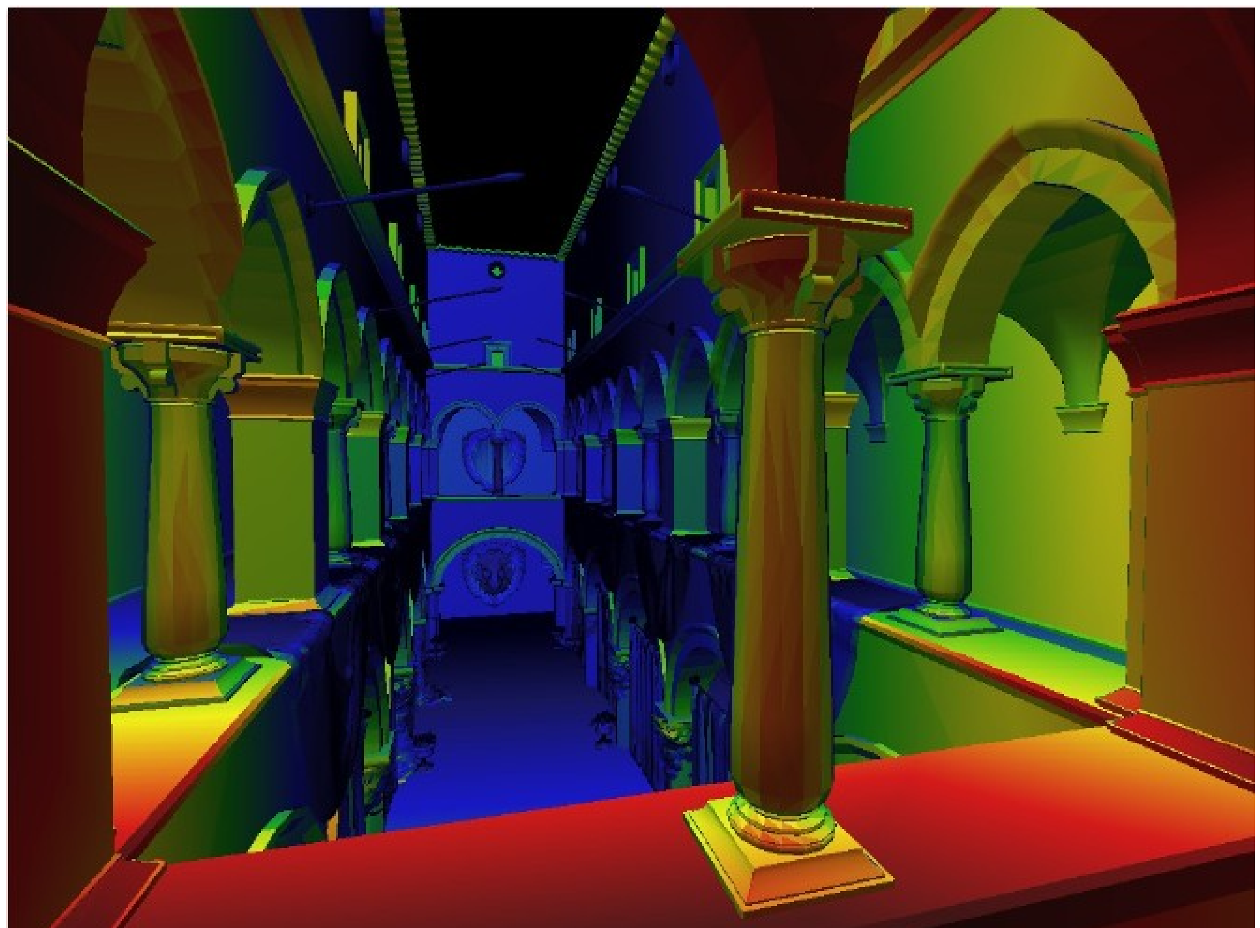
- What if we want to sample smoothly between different levels?
  - “trilinear” interpolation: bilinear interpolation between texel values, then interpolate between neighboring levels of mipmap



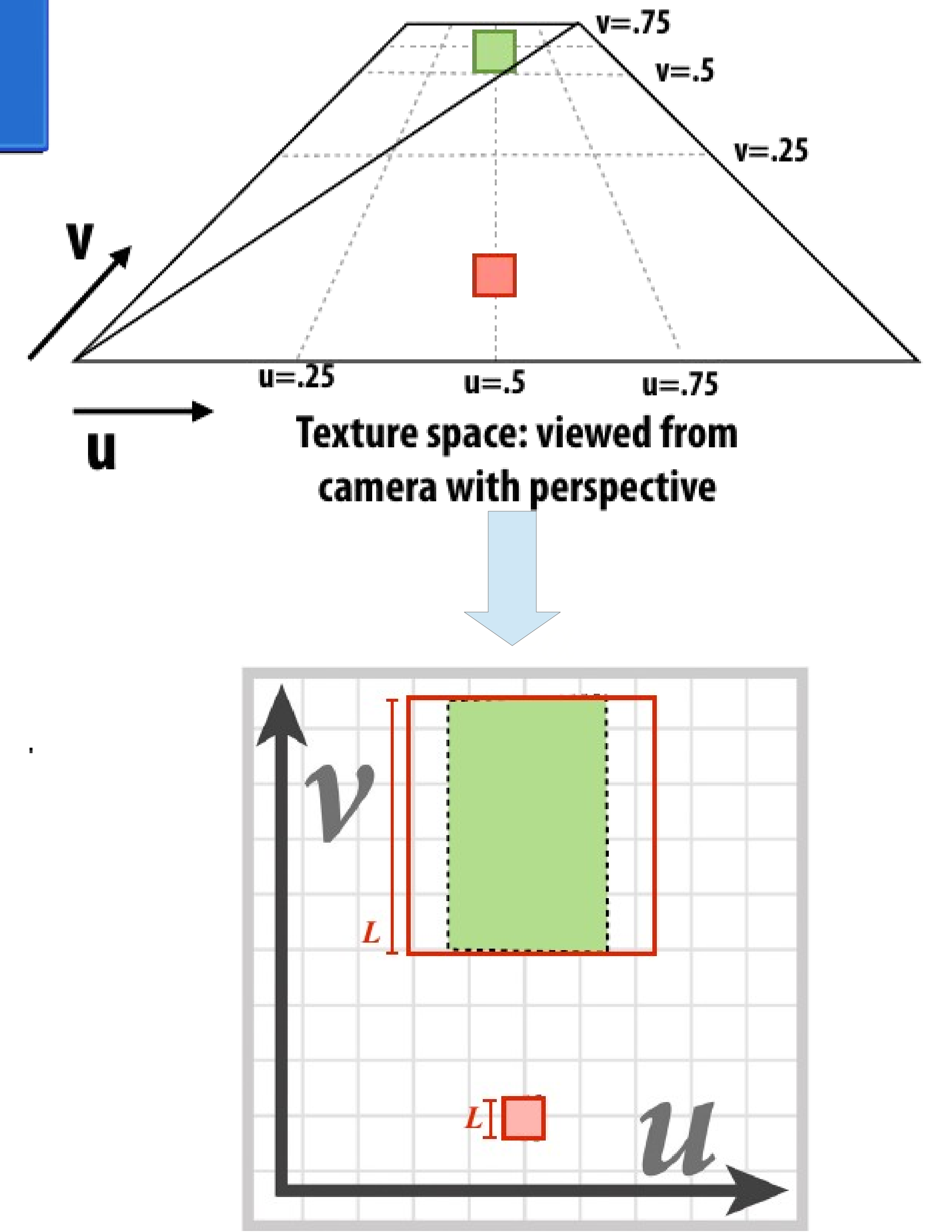
# Mipmaps

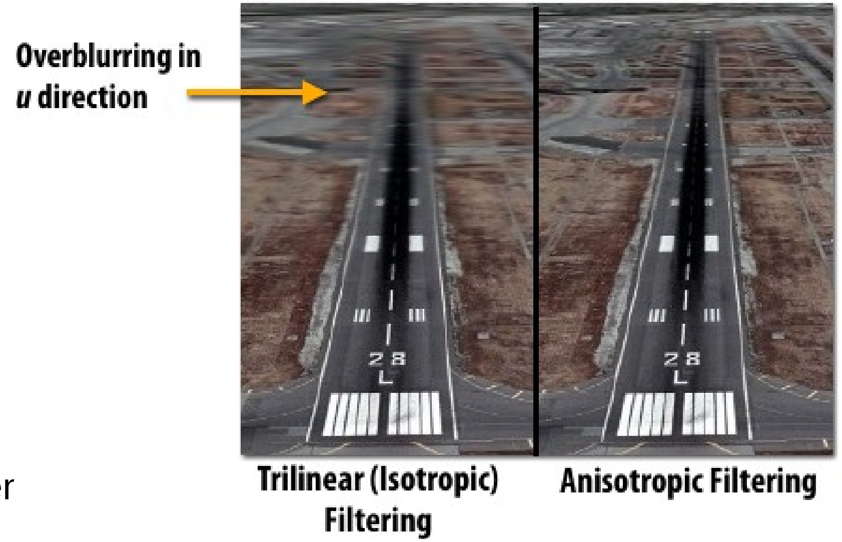
- Using "trilinear" filtering, we can express continuous $d$



# Filtering vs. isotropy

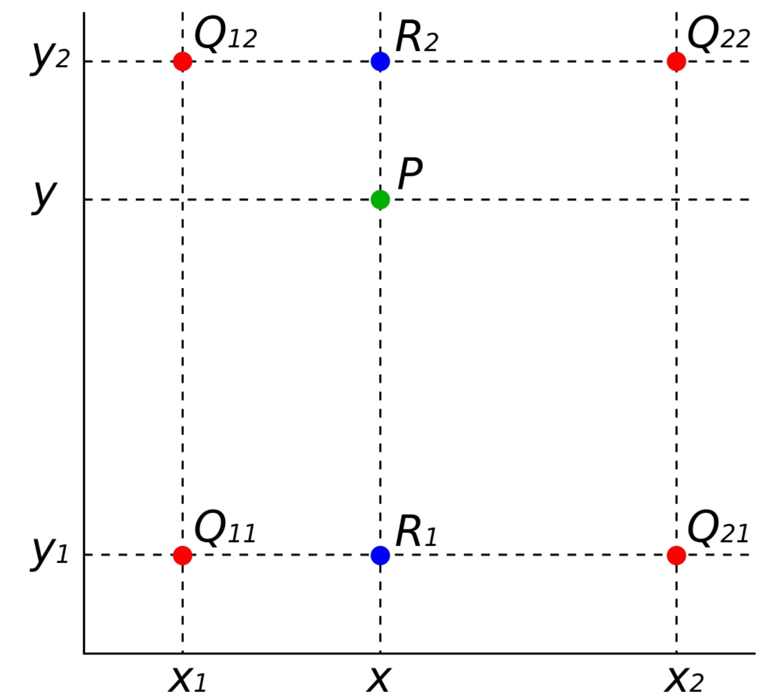
- Trilinear filtering is still based on the fixed grid of pixels
  - With the effect of perspective, our sampling frequency needs can be different in different direction
  - We need anisotropic filters...

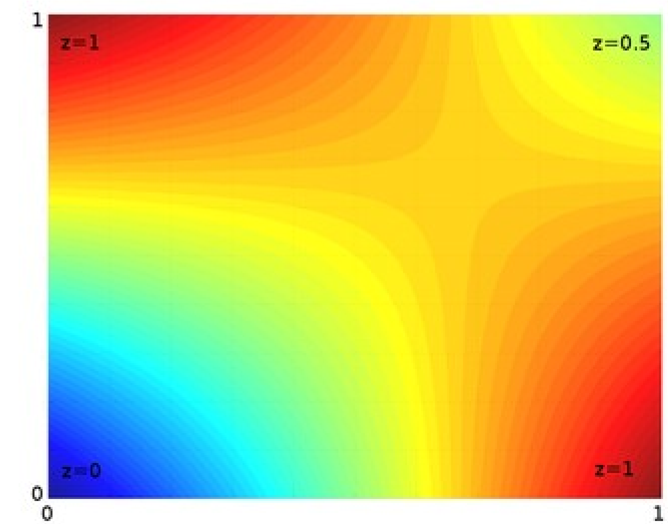


# Interpolation: back to triangles

- Bilinear interpolation seems to work well for rectangles
  - How can we make it work for triangles which are ubiquitous in computer graphics?
  - Recap: we need an interpolation method for
    - Color sampling given different vertex colors
    - (Up-)sampling texture from low resolution rasters



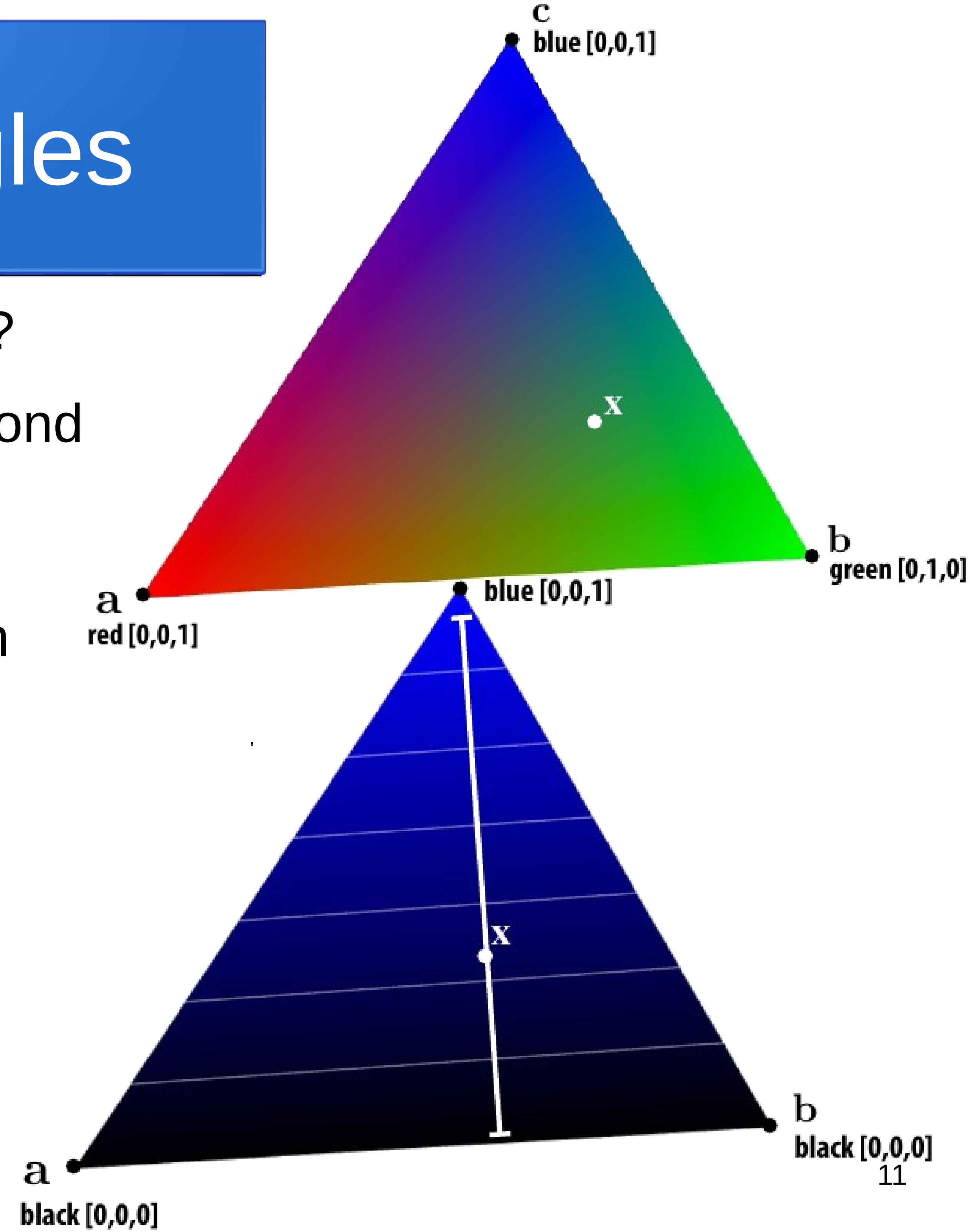
# Interpolation in triangles

- Interpolate at point x from 3 values?
- Let's simplify the situation for a second – what if the bottom vertices had identical values?
  - We could use linear interpolation between the blue vertex and the bottom edge (b-a) to obtain the interpolated value of x

$$\mathbf{x} = (1-t)\begin{bmatrix}0 & 0 & 1\end{bmatrix} + t\begin{bmatrix}0 & 0 & 0\end{bmatrix}$$

with

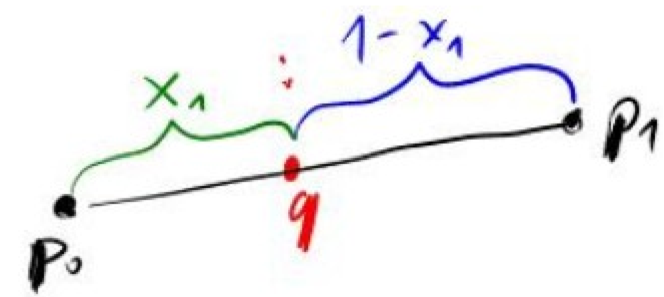
$$t = \frac{\text{dist. x to b-a}}{\text{dist. c to b-a}}$$



# Interpolation in triangles

$$\mathbf{x} = (1-t)\begin{bmatrix}0 & 0 & 1\end{bmatrix} + t\begin{bmatrix}0 & 0 & 0\end{bmatrix}$$

$$t = \frac{\text{dist. x to b-a}}{\text{dist. c to b-a}}$$




- In 1-D it quickly reduces to the well known formula!

$$\mathbf{q} = (1 - x_1)\mathbf{p_0} + x_1\mathbf{p_1}$$

# Barycentric coordinates

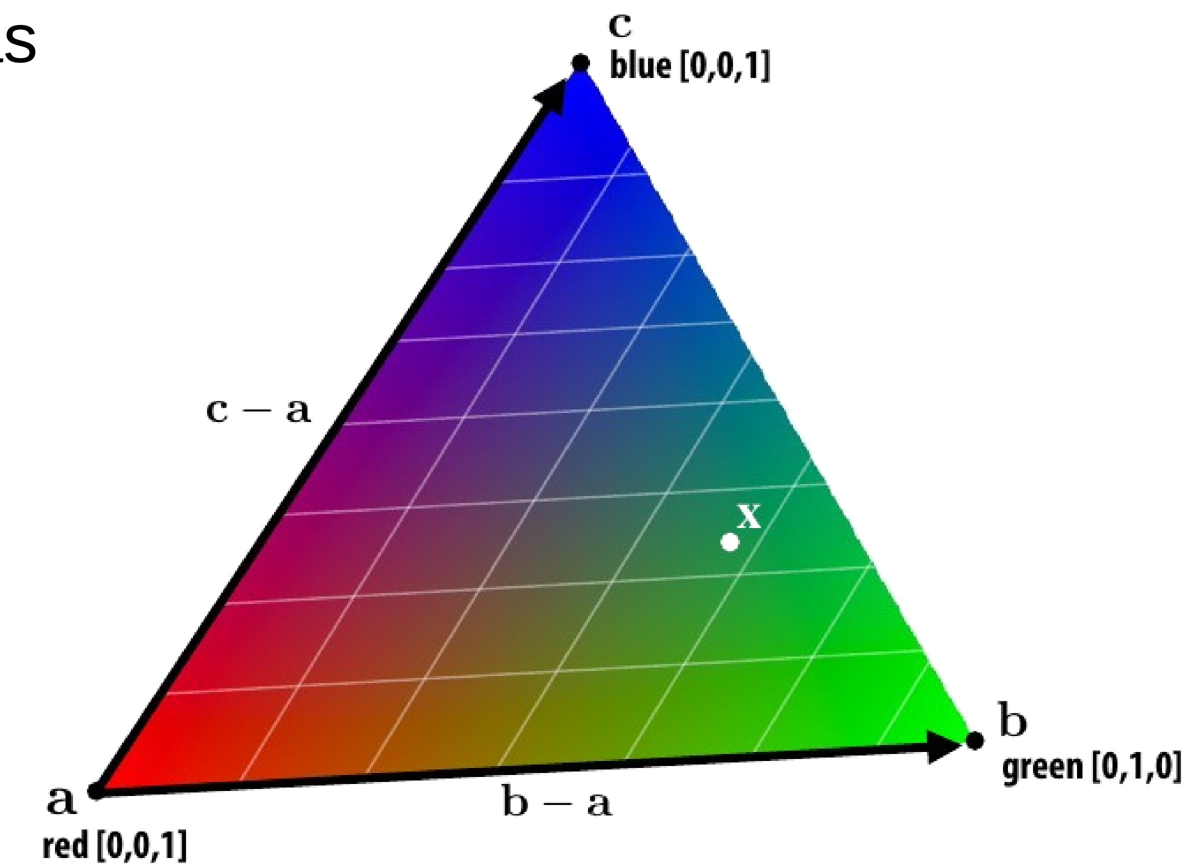
- Let us imagine sides (b-a) and (c-a) as non-orthogonal basis in which other points can be expressed, including all points in the triangle itself

$$\begin{aligned}\mathbf{x} &= \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}) \\ &= (1 - \beta - \gamma)\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c} \\ &= \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}\end{aligned}$$

$$\alpha + \beta + \gamma = 1$$

$$\mathbf{x}_{color} = \alpha\mathbf{a}_{color} + \beta\mathbf{b}_{color} + \gamma\mathbf{c}_{color}$$

- This looks surprisingly elegant!



# Barycentric coordinates

- The three scalar parameters can be said to be proportional to distances as we expressed it in the motivating example, e.g. β varies with the distance of x to edge (c-a)

$$kE_{\mathbf{ac}}(\mathbf{x}_x, \mathbf{x}_y) = \beta$$

$$kE_{\mathbf{ac}}(\mathbf{b}_x, \mathbf{b}_y) = 1$$

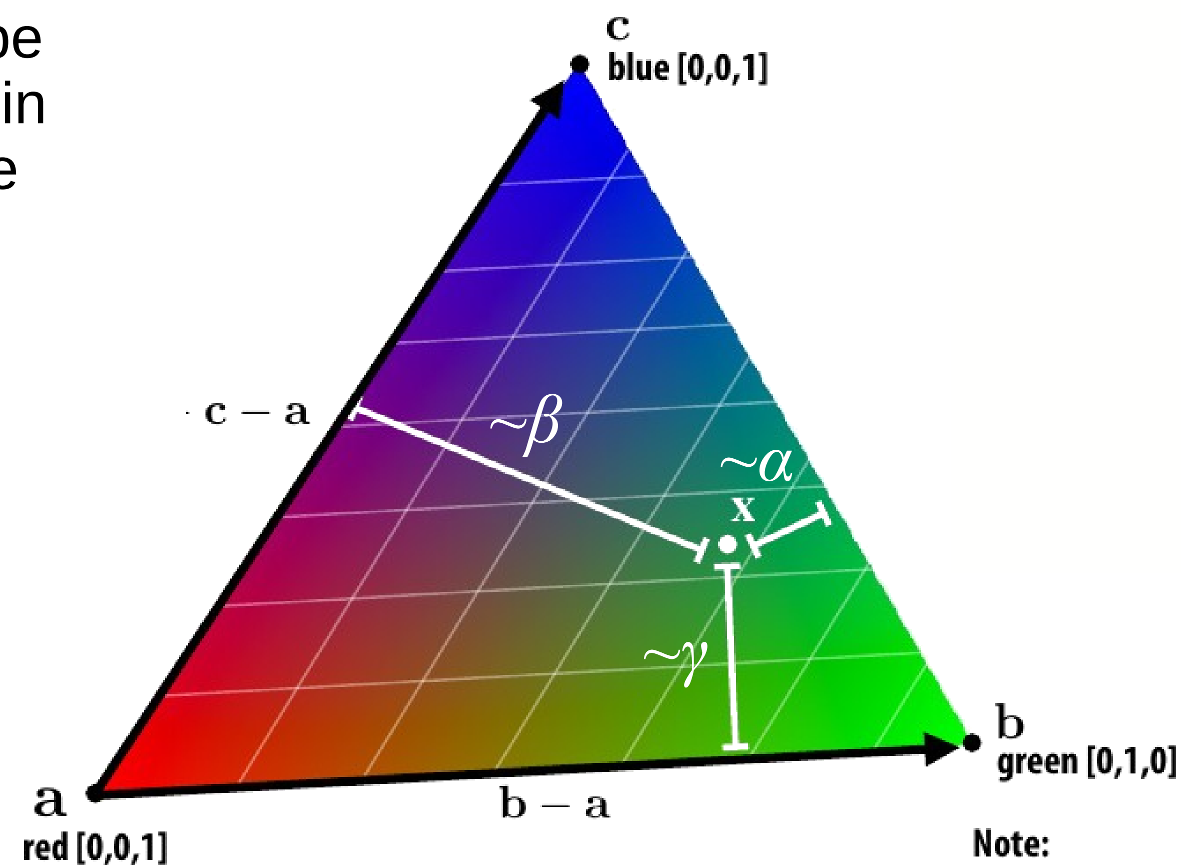
$$\beta = \frac{E_{\mathbf{ac}}(\mathbf{x}_x, \mathbf{x}_y)}{E_{\mathbf{ac}}(\mathbf{b}_x, \mathbf{b}_y)}$$

$$\beta = \frac{(\mathbf{a}_y - \mathbf{c}_y)\mathbf{x}_x + (\mathbf{c}_x - \mathbf{a}_x)\mathbf{x}_y + \mathbf{a}_x\mathbf{c}_y - \mathbf{c}_x\mathbf{a}_y}{(\mathbf{a}_y - \mathbf{c}_y)\mathbf{b}_x + (\mathbf{c}_x - \mathbf{a}_x)\mathbf{b}_y + \mathbf{a}_x\mathbf{c}_y - \mathbf{c}_x\mathbf{a}_y}$$

- Let's call *(α,β,γ)* **barycentric coordinates**
- They are an affine function of x

$$\mathbf{x}_{\text{color}} = \alpha\mathbf{a}_{\text{color}} + \beta\mathbf{b}_{\text{color}} + \gamma\,\mathbf{c}_{\text{color}}$$

$$\mathbf{x}_{\text{color}} = A\mathbf{x}_x + B\mathbf{x}_y + C$$



# Barycentric coordinates

- In fact, we can treat *(α,β,γ)* as a ratio of areas of the corresponding parts of the triangle "trisected" by lines from the vertices to x

$$\alpha = A_A/A$$

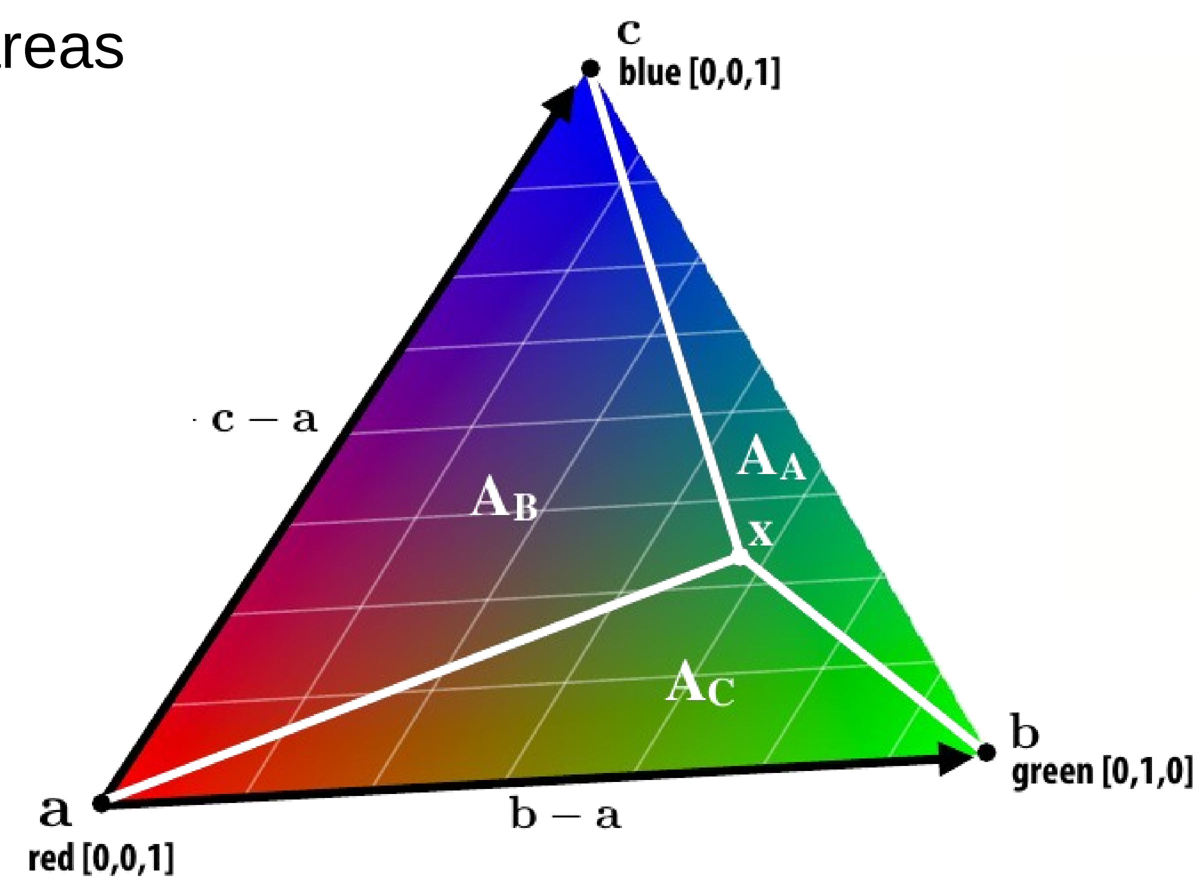
$$\beta = A_B/A$$

$$\gamma = A_C/A$$

- The sum of *(α,β,γ)* is always 1

$$\alpha + \beta + \gamma = 1$$

- It works even if we move to higher dimensions!
- … what if x is outside of triangle?



# Barycentric coordinates

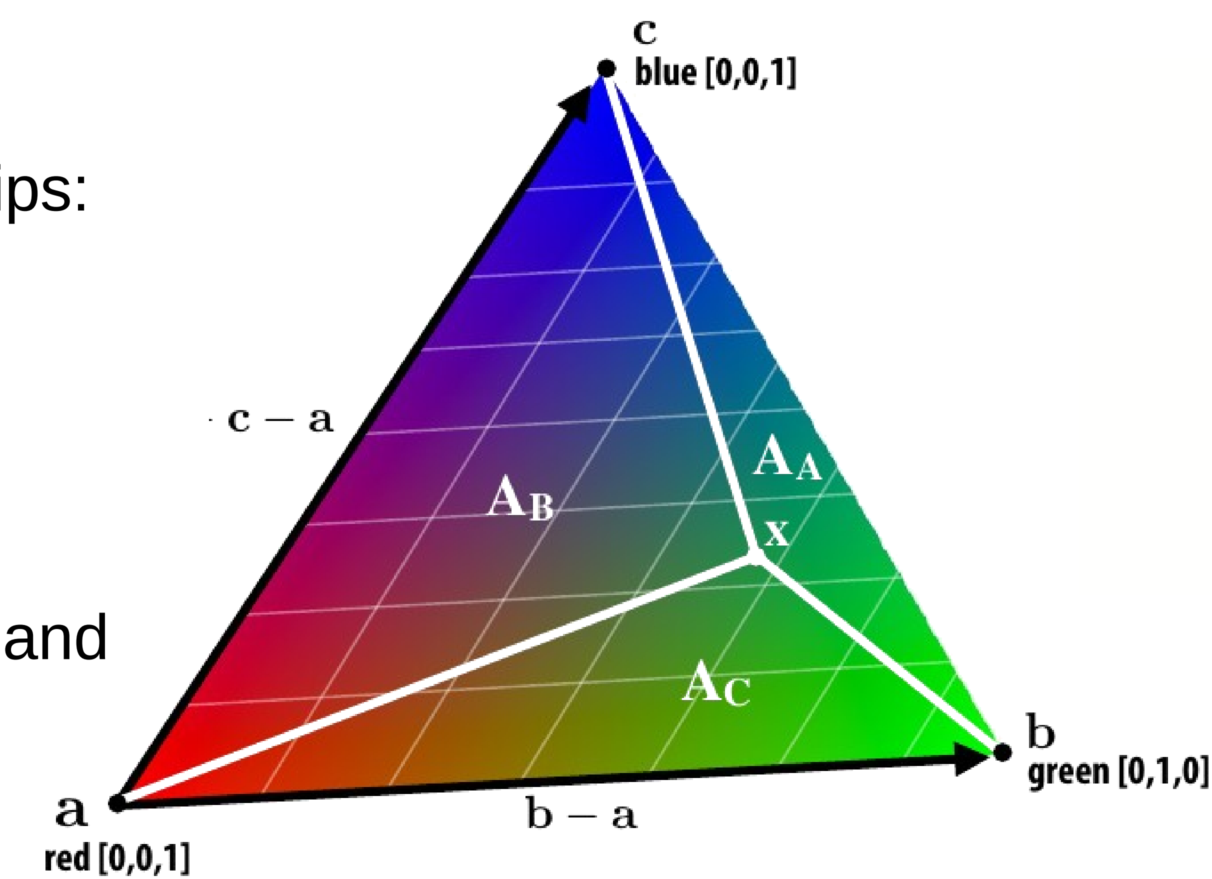
- Stated otherwise: any surface attribute f defined at any points a, b and c we can express as a system of affine relationships:

$$f_a = A\mathbf{a}_x + B\mathbf{a}_y + C$$

$$f_b = A\mathbf{b}_x + B\mathbf{b}_y + C$$

$$f_c = A\mathbf{c}_x + B\mathbf{c}_y + C$$

- $A$, $B$ and $C$ can be treated as unknowns and algebraically determined
- Then, we can express any $f_x$ using the same scheme



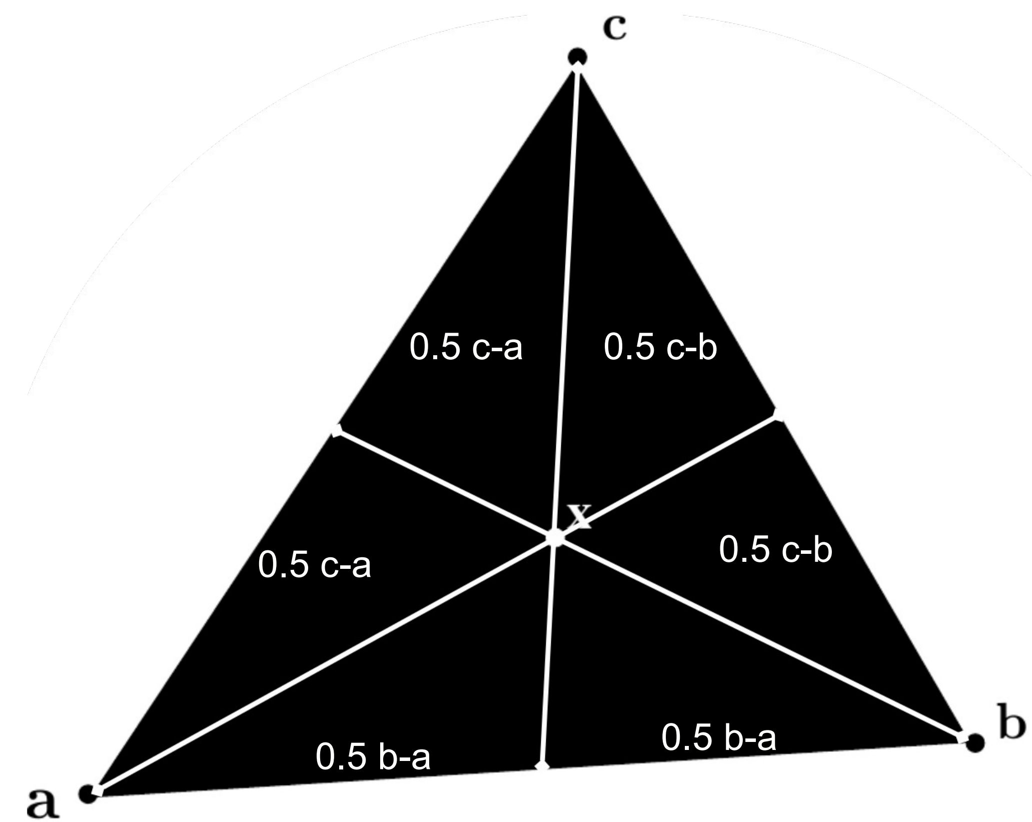
# Barycentric coordinates

- Some interesting barycentric points
  - ...the barycentre itself

    = the centre of mass of a triangle

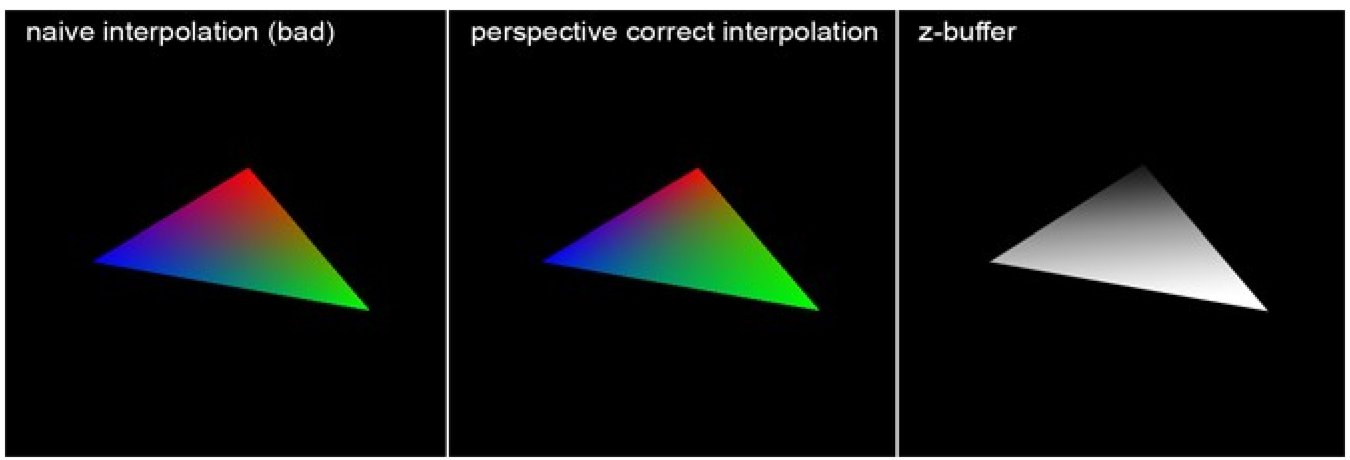
$$\mathrm{x} = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$$

  - It makes a lot of sense, given that the three subtriangles must “balance” each other out at that point

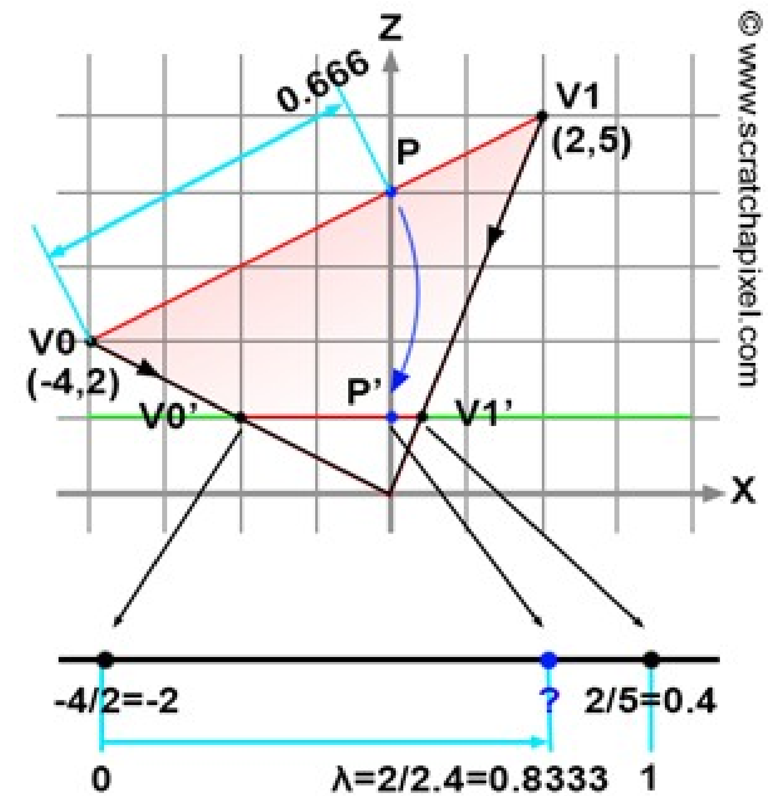


# Interpolation

- Barycentric interpolation works like linear interpolation
  - Perspective projection preserves lines, but does not preserve distances



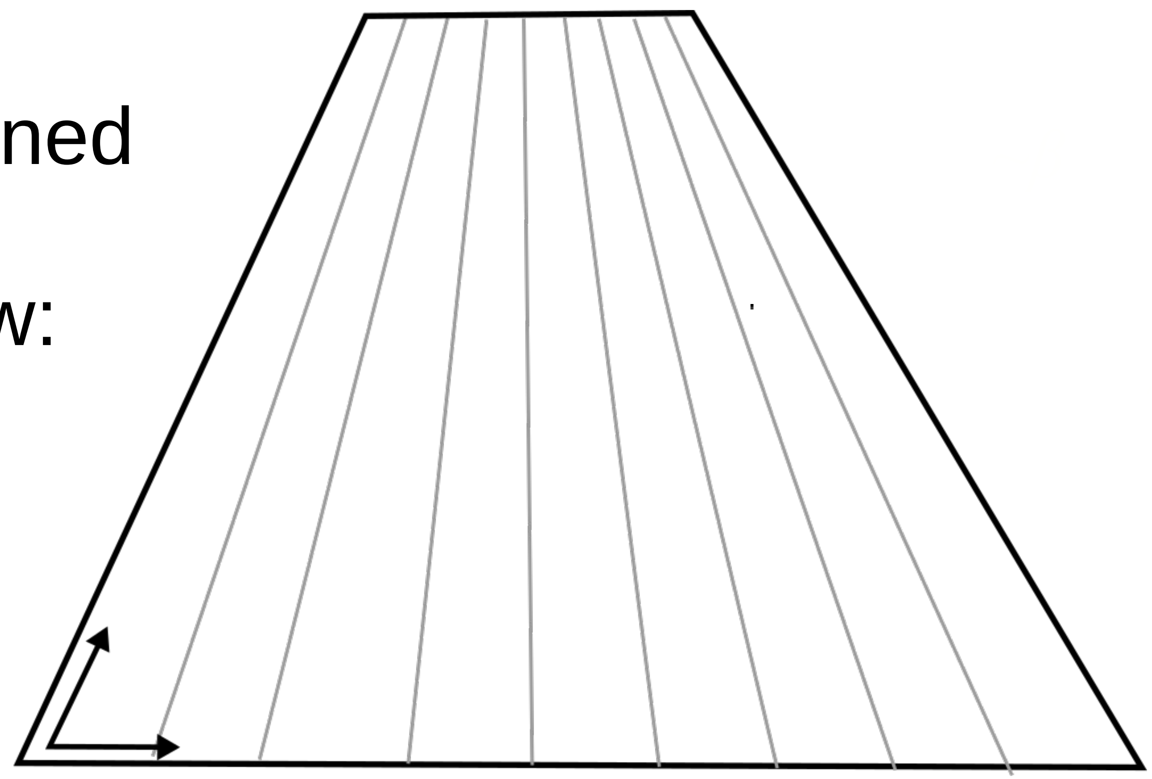

  - Correction – use z-buffer to renormalize the (u,v) coordinates!




A proof that an oblique edge projected to a horizontal segment does not conserve the propotions – here, a point P virtually moves in terms of its barycentric position

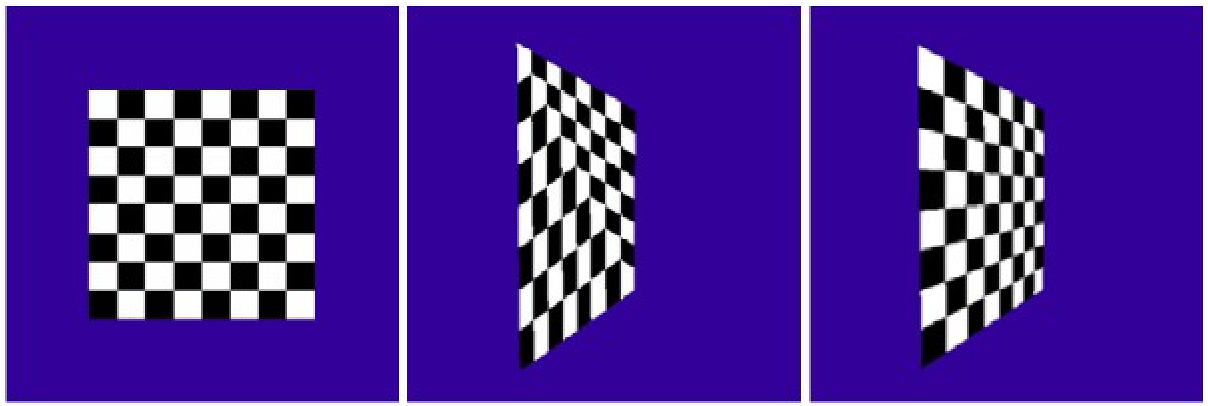
# Barycentric coordinates

- We want a result of texturing of a transformed geometry to look good
  - Imagine an inclined rectangle in a perspective view:



# Barycentric coordinates

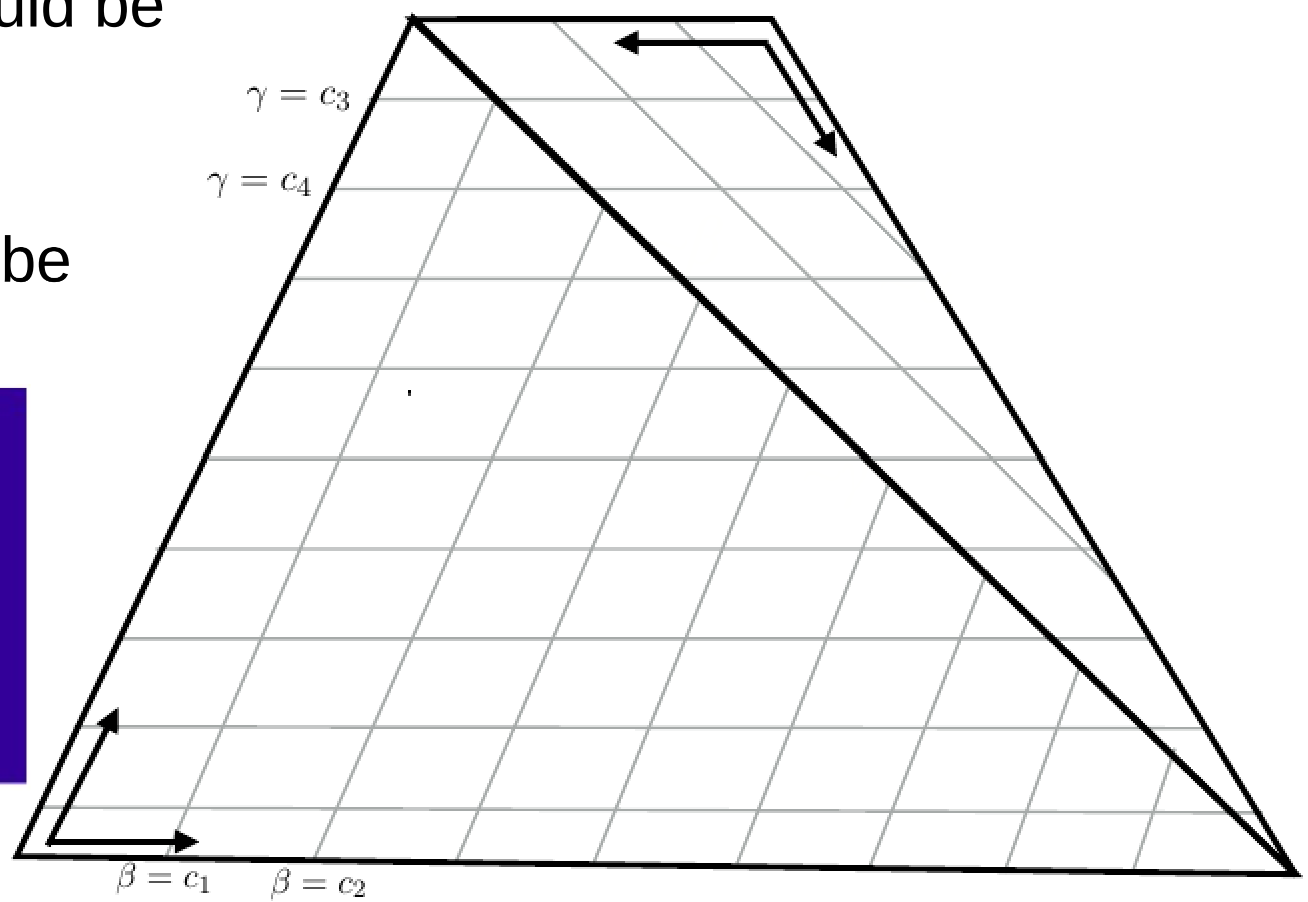
- Yet with all the geometries subdivided into triangles, when individual triangles are processed separately, the result could be far from ideal!
  - The impact of the perspective projection on interpolation must be corrected



Before transf.

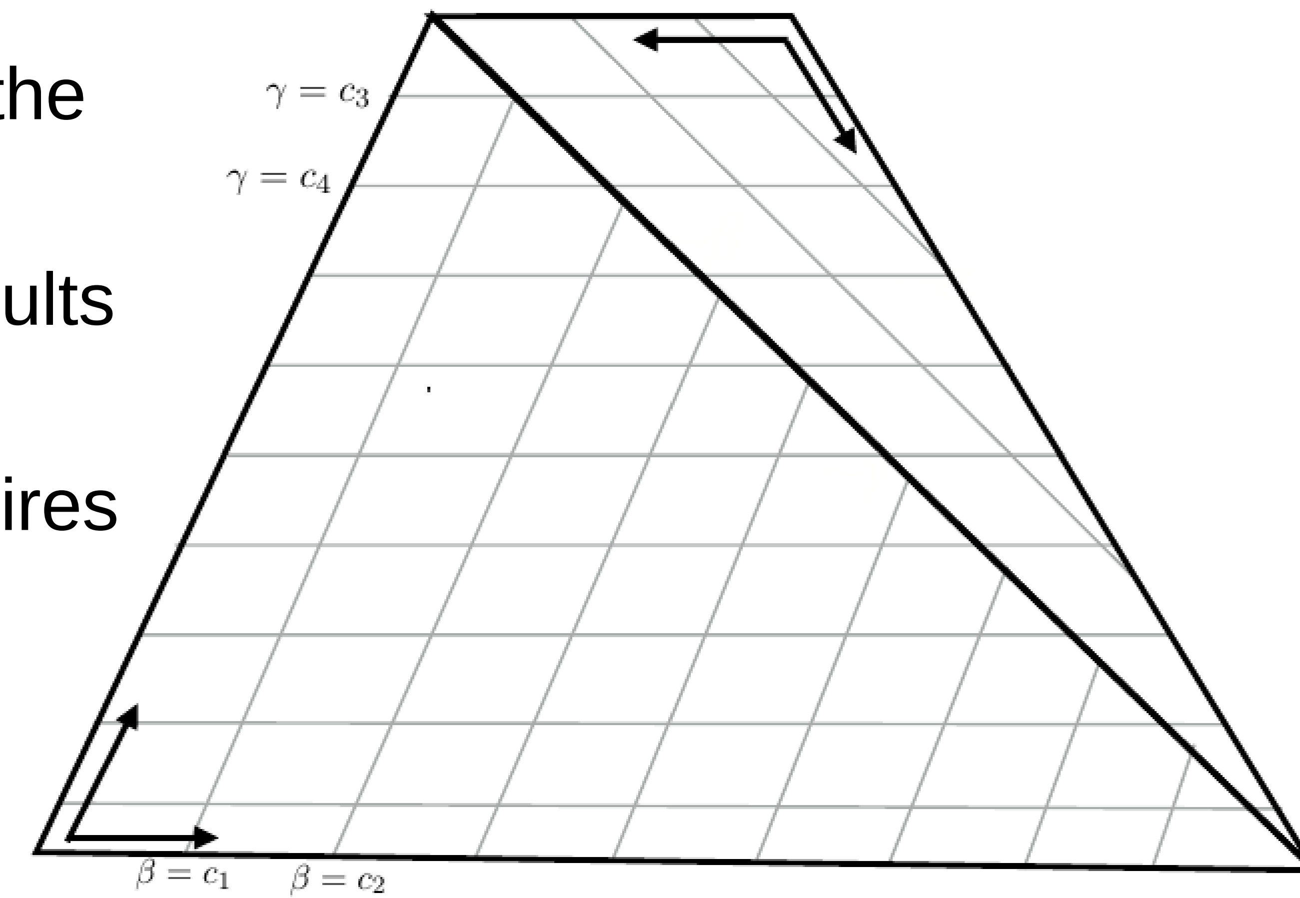
What we might get

What we want



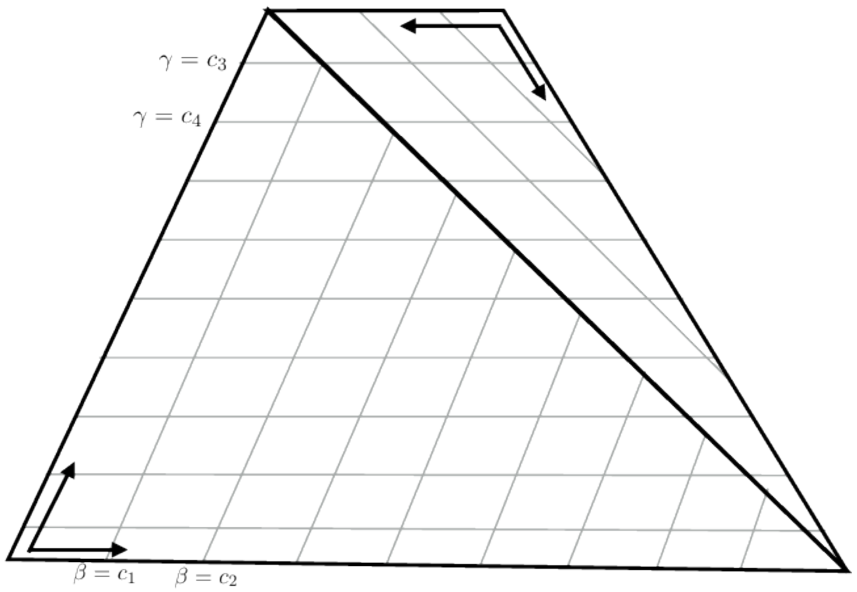
# Screen space barycentric coordinates correction

- Perspective transformation nonlinearly changes a triangles shape, leading to different barycentric weights before/after the transformation
- Interpolating in screen space results in texture distortion
- Interpolating in world space requires projecting all pixel locations backwards from screen space to world space, which is expensive



# Screen space barycentric coordinates correction

- Perspective transformation nonlinearly changes a triangles shape, leading to different barycentric weights before/after the transformation
- Interpolating in screen space results in texture distortion
- Interpolating in world space requires projecting all pixel locations backwards from screen space to world space, which is expensive



# Screen space barycentric coordinates correction

- As input we have a triangle projected into screen space with three vertices: $p_0, p_1$ and $p_2$, with depths $z_0, z_1$ and $z_2$ respectively

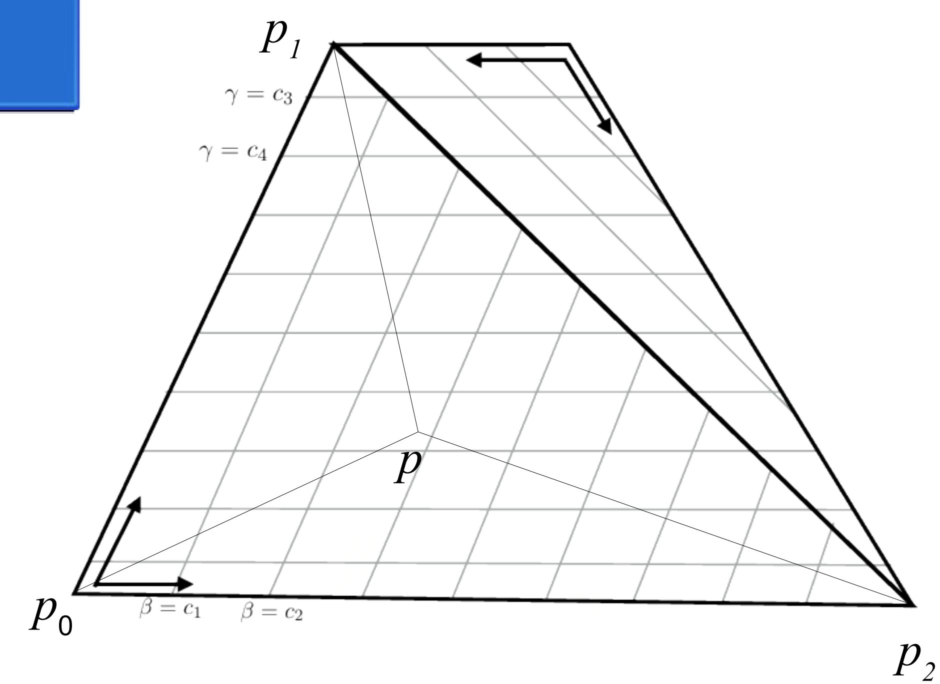
- The barycentric coordinates of any point $p$ in that triangle in screen space:

$$\boldsymbol{p}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \boldsymbol{\alpha p_0} + \boldsymbol{\beta p_1} + (\mathbf{1} - \boldsymbol{\alpha} - \boldsymbol{\beta})\boldsymbol{p_2}$$

- For texture/color/normal sampling, use corrected coordinated in world space $\boldsymbol{\alpha^w}, \boldsymbol{\beta^w}, (\mathbf{1} - \boldsymbol{\alpha^w} - \boldsymbol{\beta^w})$ with:

$$\boldsymbol{\alpha^w} = \frac{\boldsymbol{z_1 z_2 \alpha}}{\boldsymbol{z_0 z_1} + \boldsymbol{z_1 \alpha}(\boldsymbol{z_2} - \boldsymbol{z_0}) + \boldsymbol{z_0 \beta}(\boldsymbol{z_2} - \boldsymbol{z_1})}$$

$$\boldsymbol{\beta^w} = \frac{\boldsymbol{z_0 z_2 \beta}}{\boldsymbol{z_0 z_1} + \boldsymbol{z_1 \alpha}(\boldsymbol{z_2} - \boldsymbol{z_0}) + \boldsymbol{z_0 \beta}(\boldsymbol{z_2} - \boldsymbol{z_1})}$$



# Thank you!

- Questions?