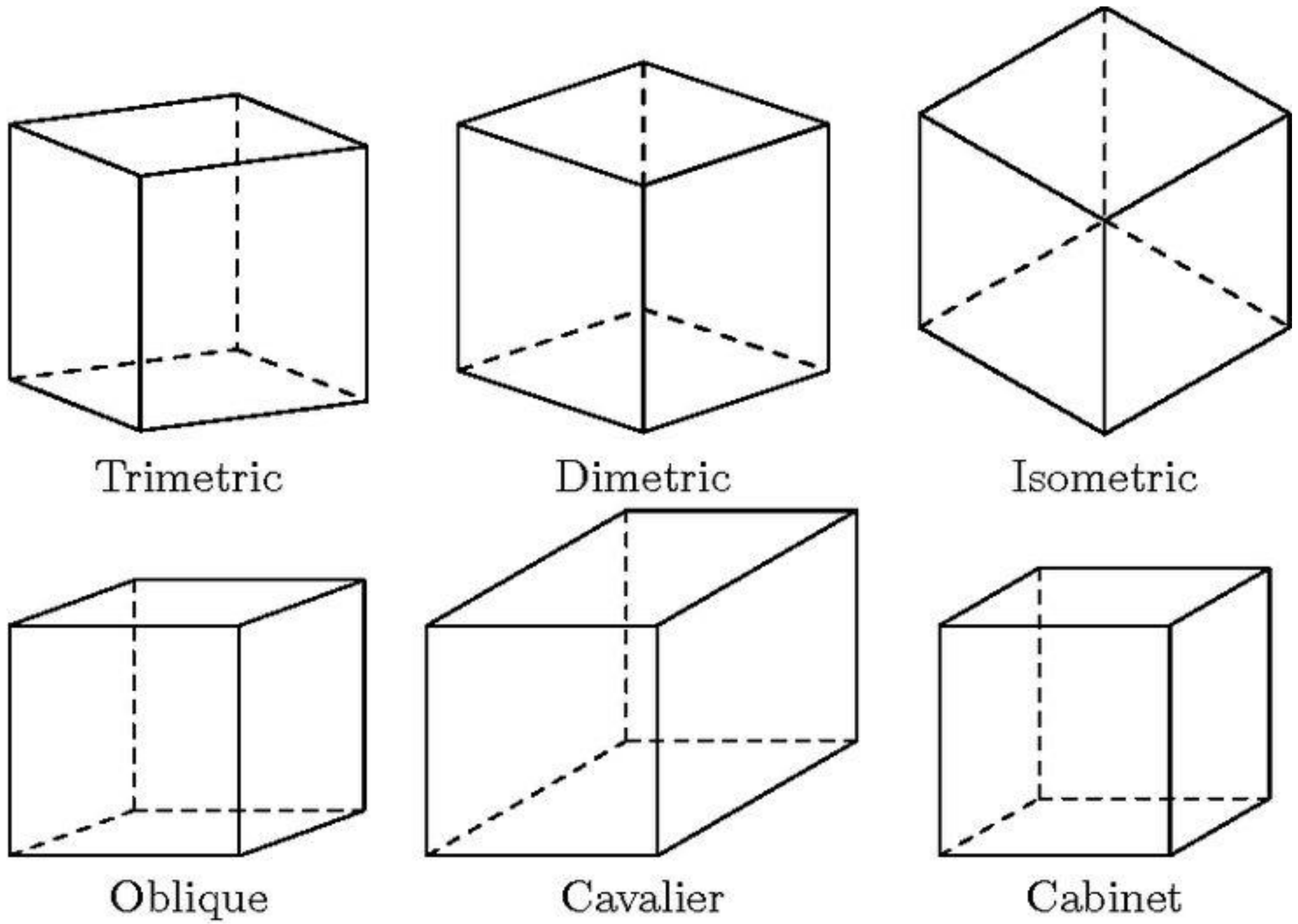
Computer Graphic

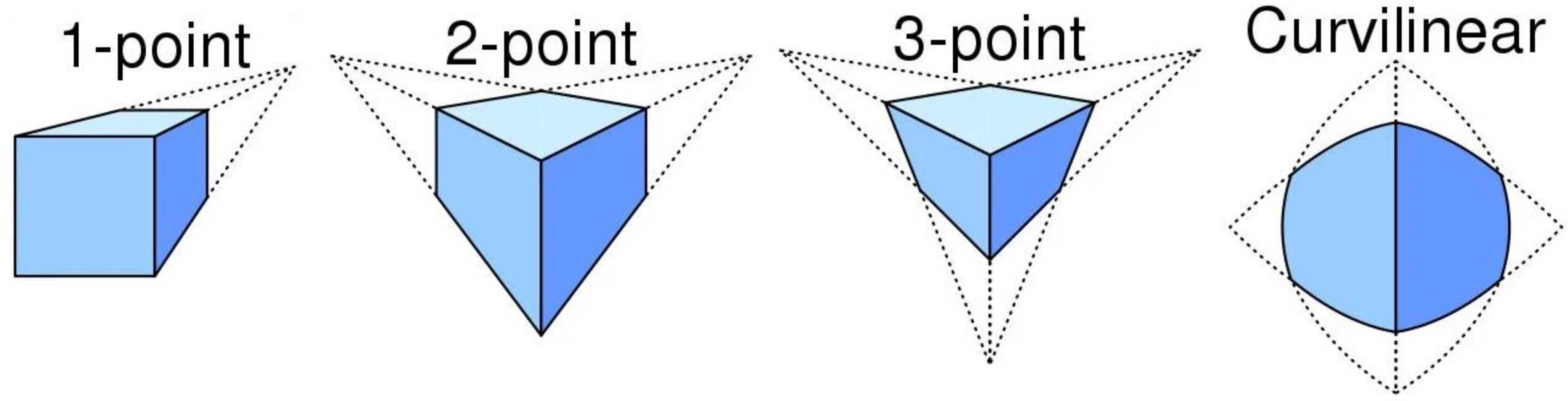
Review

YUDI WIDHIYASANA

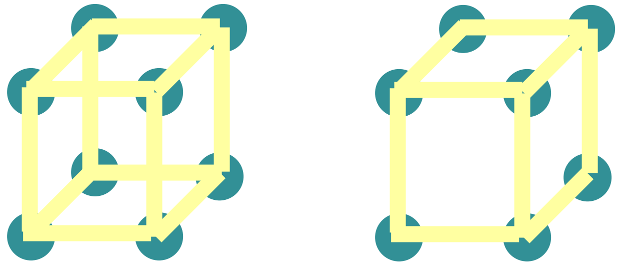
Parallel Projection



Perspective Projection



3D View



Hidden Surface Removal

Visibility

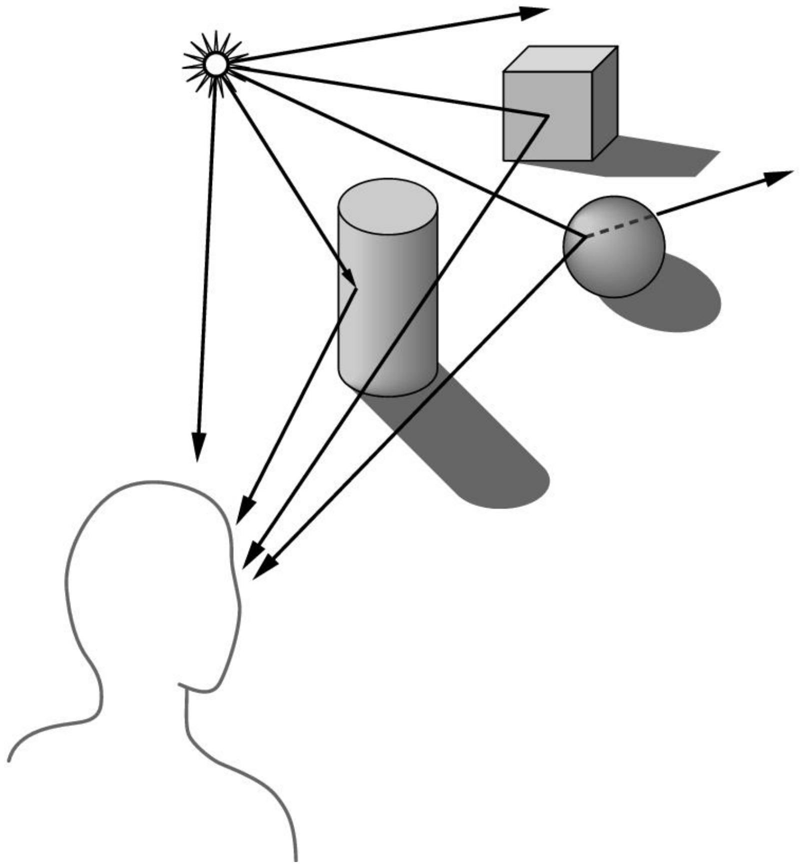
Depth Cueing

Need for shading

How do you make something look 3D?

Shading that is appropriate for the lighting is the primary cue to 3D appearance

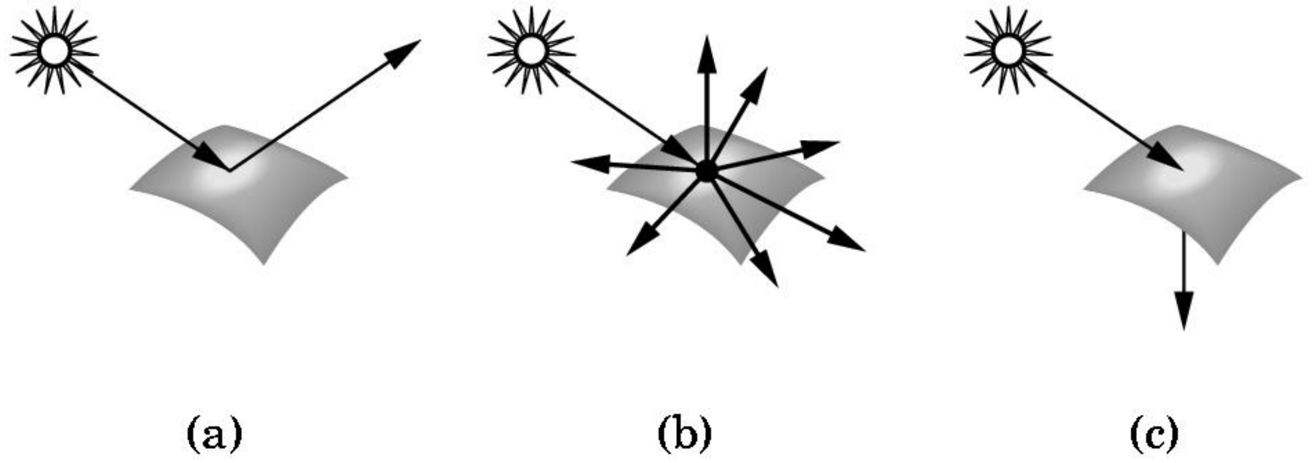
[What are some other cues?]



Local illumination

To make lighting fast, we will initially restrict our attention to:

- Light source, one surface, and viewer (ignore inter-object reflections, shadows)
- Ambient, diffuse, and specular reflection (ignore transparency, refraction, reflection, …)



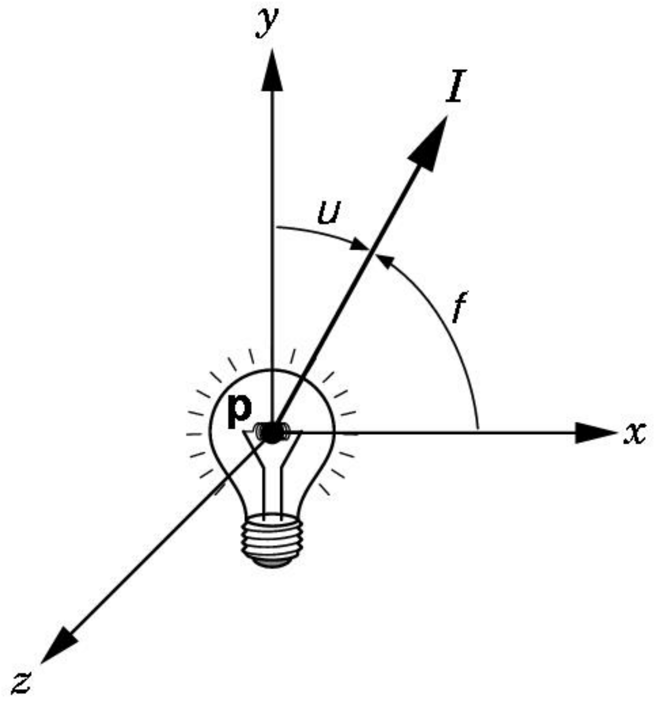
Light sources

In general, a light source is a rather complicated thing. It can emit different amounts of light for each

- Location (x, y, z)
- Direction (θ, ϕ)
- Wavelength (λ)

Illumination function:
I(x, y, z, θ, ϕ, λ)

Examples: ambient, point, area, spot, distant, …

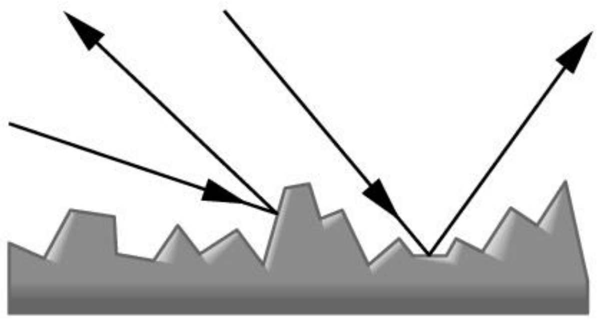


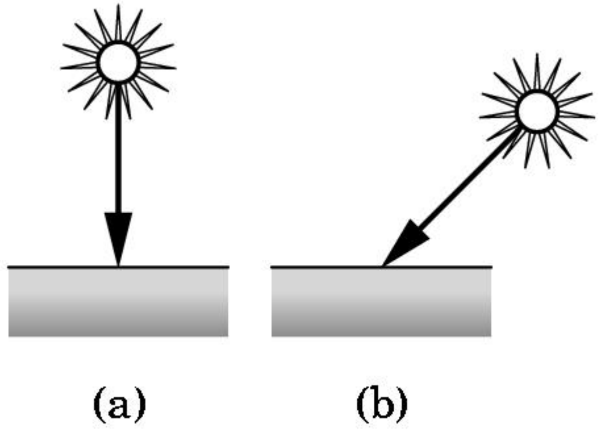
Diffuse term

A perfectly diffuse reflector is so rough that it scatters light equally in all directions

But note that when the light comes in at an angle, the same energy is spread out over larger area

Such surfaces are called *Lambertian surfaces* (obeying Lambert's Law)





Diffuse shading

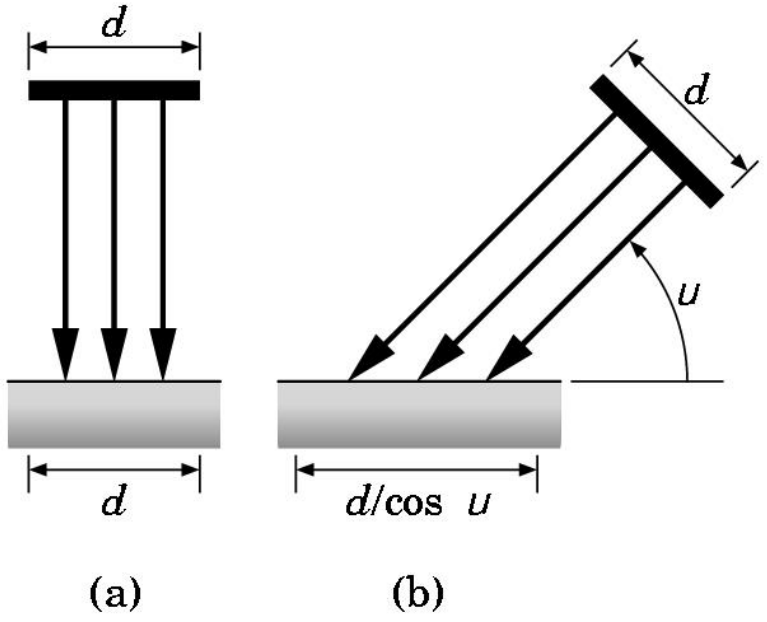
At noon, illum. is 1

As the angle θ (u in figure) decreases, illumination goes to zero

Illumination is proportional to $\cos(\theta)$
(Lambert's law)

$\cos(\theta) = \mathbf{l} \cdot \mathbf{n}$

$I_d = k_d\, \mathbf{l} \cdot \mathbf{n}\, L_d$



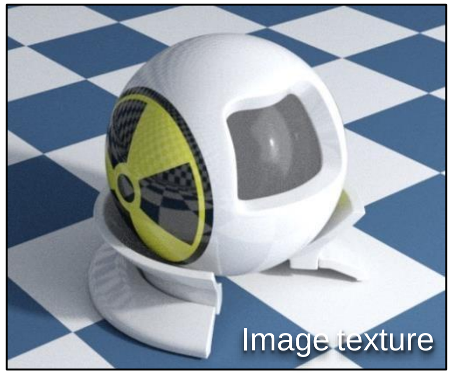
Texturing

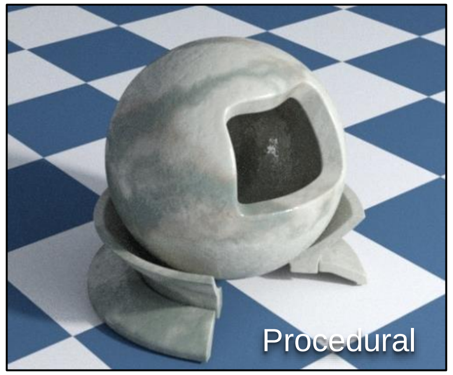
Image texturing:

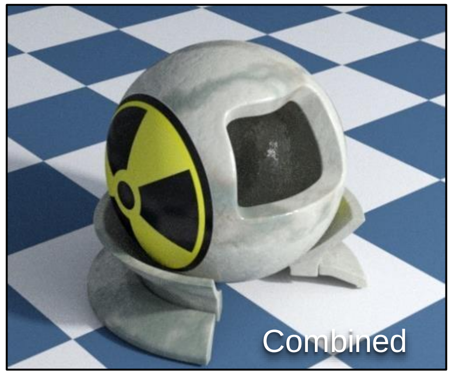
- The spatial/temporal patterns are expressed in the form of a digitized bitmap
- A bitmap texture can be an array of values of 1/2/3 + time dimensions
- Textures are stored in GPU/CPU memory and sampled during rendering

Procedural texturing:

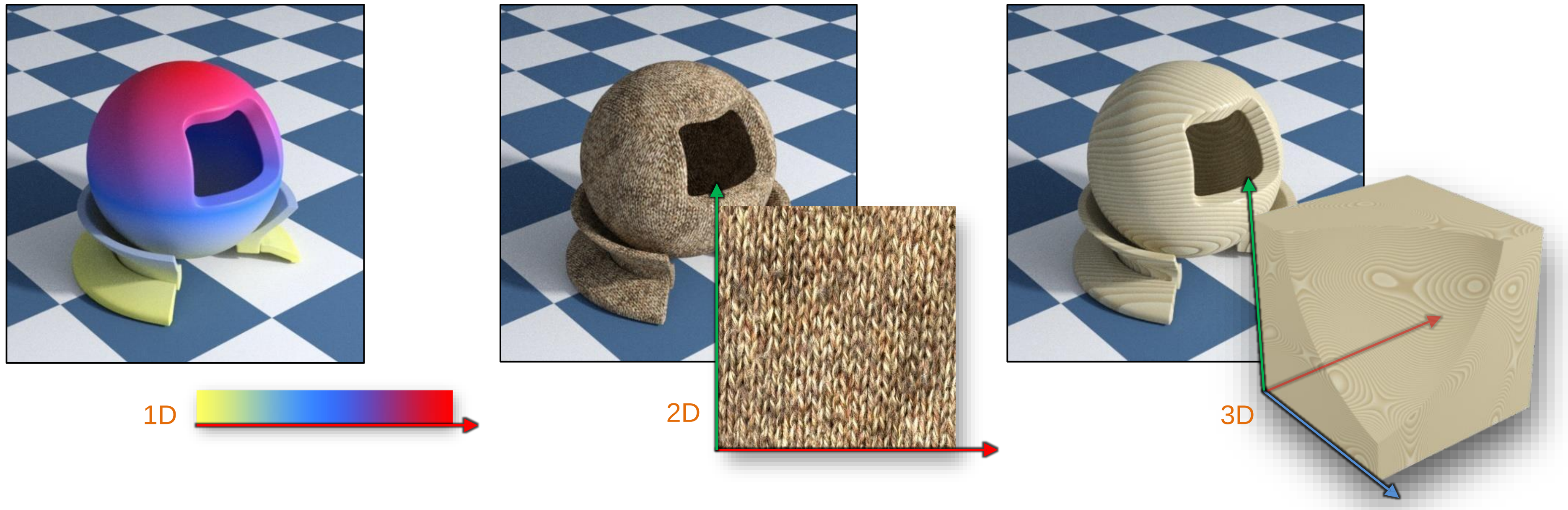
•The spatial/temporal patterns are generated using a function or algorithm







Texturing

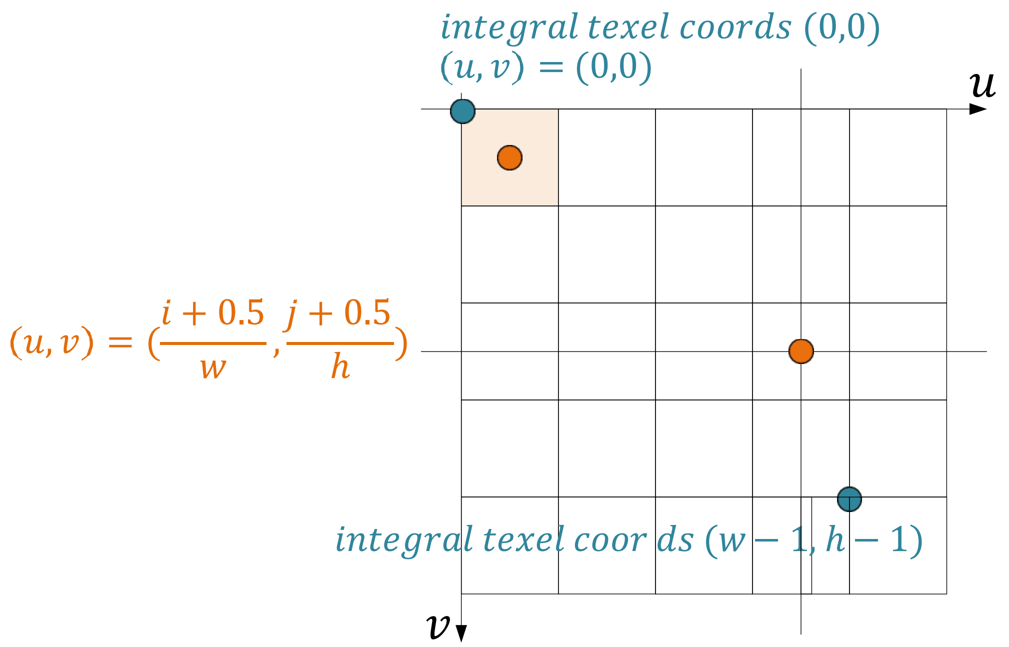


- A 1D-3D image texture is defined in a texture parametric space
- The parametric space is usually considered normalized w.r.t the dimensions of the raster

– For example, a 2D raster is defined on a plane with two parameters (e.g. u,v)

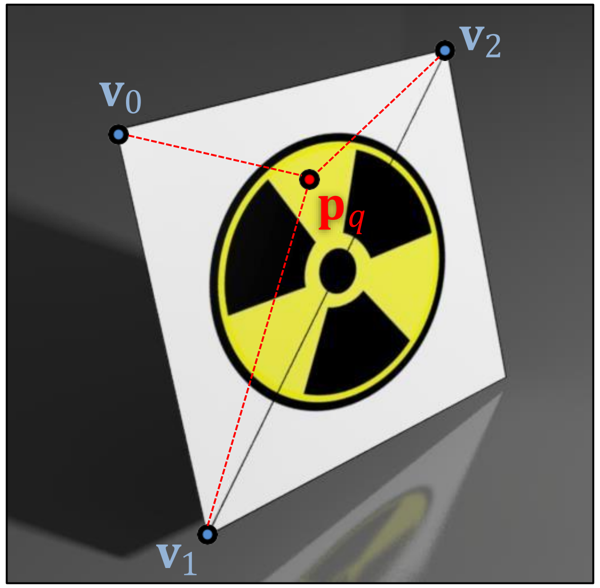
Texturing

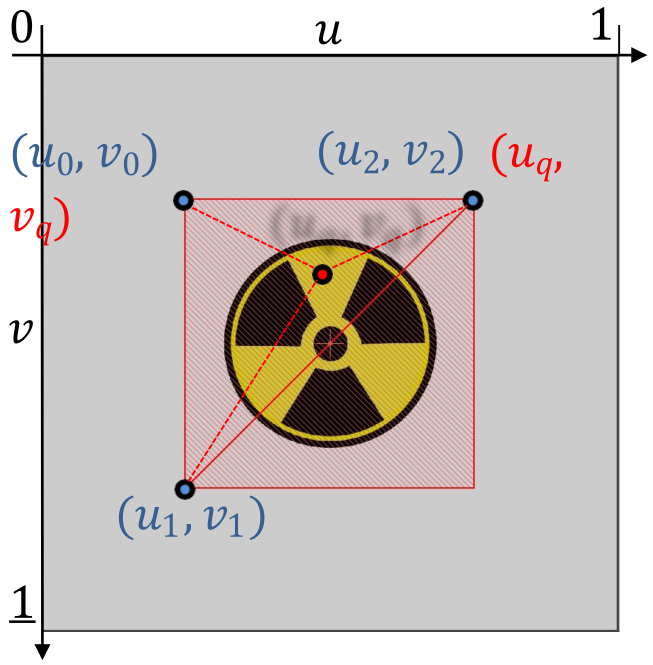
- The smallest accessible element in a 1D/2D/3D raster texture is the texel (texture element)
- Texels are considered discrete samples on the raster and their integral coordinates correspond to corners of the raster elements


$integral\ texel\ coords\ (0,0)$
$(u,v) = (0,0)$
u
$(u,v) = (\frac{i+0.5}{w}, \frac{j+0.5}{h})$
$integral\ texel\ coords\ (w-1, h-1)$
v

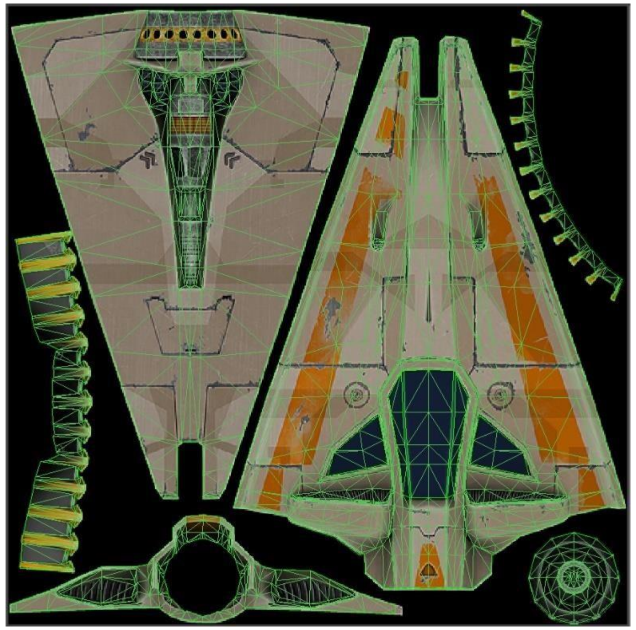
Texturing

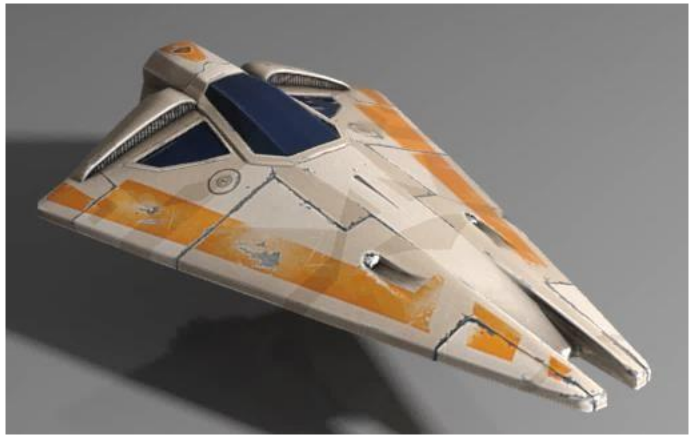
- Texture coordinates on arbitrary locations on the triangles are interpolated from the tex. coordinates of the triangle vertices, using the same barycentric coordinates used for other attributes





Texturing



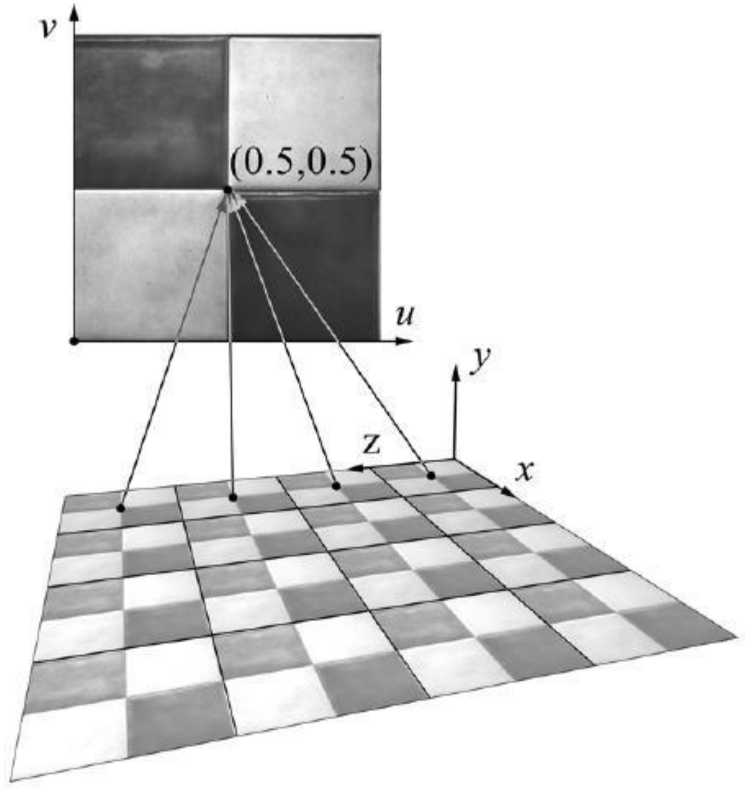


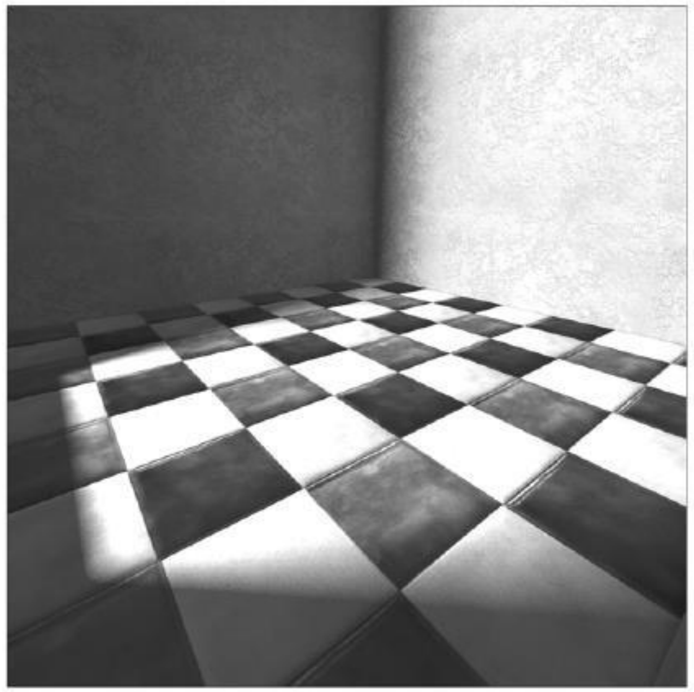
- Fine adjustments of the triangle texture coordinates can be done directly on the parametric space

  –Vertex texture coordinates can be manipulated using a 2D editor, the UV Editor

  –Texture coordinates with vertex connectivity can be rendered with orthographic projection as (u,v,0) points

Texturing

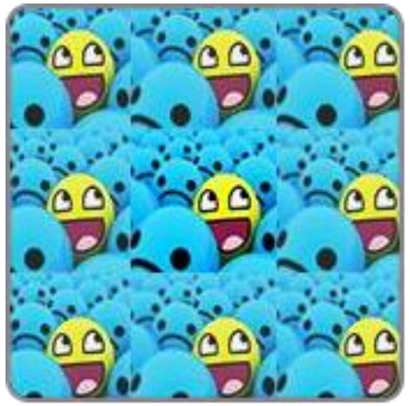


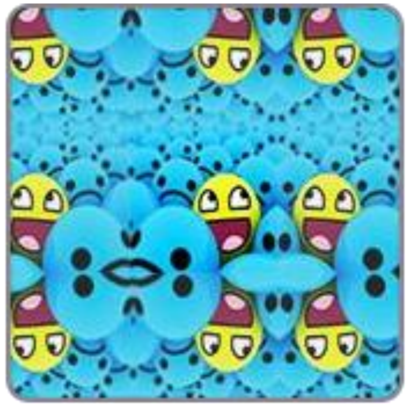


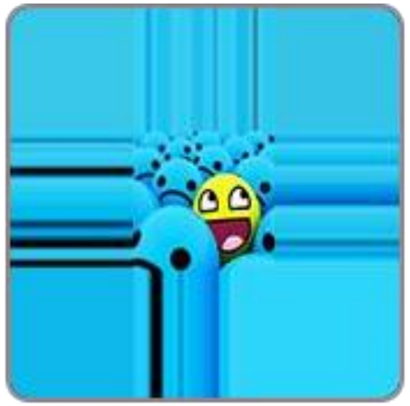
- In general, multiple points on the geometry may index the same texture coordinates → The mapping is not necessarily bijective

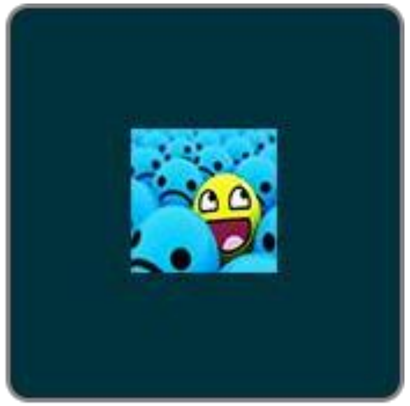
Texturing

- The parametric space coordinates lie in the range [0,1]
- Therefore, all texture coordinates are conformed to this range using a texture wrapping function. Typical examples:


GL_REPEAT


GL_MIRRORED_REPEAT


GL_CLAMP_TO_EDGE


GL_CLAMP_TO_BORDER

$$s = s - \lfloor s \rfloor$$

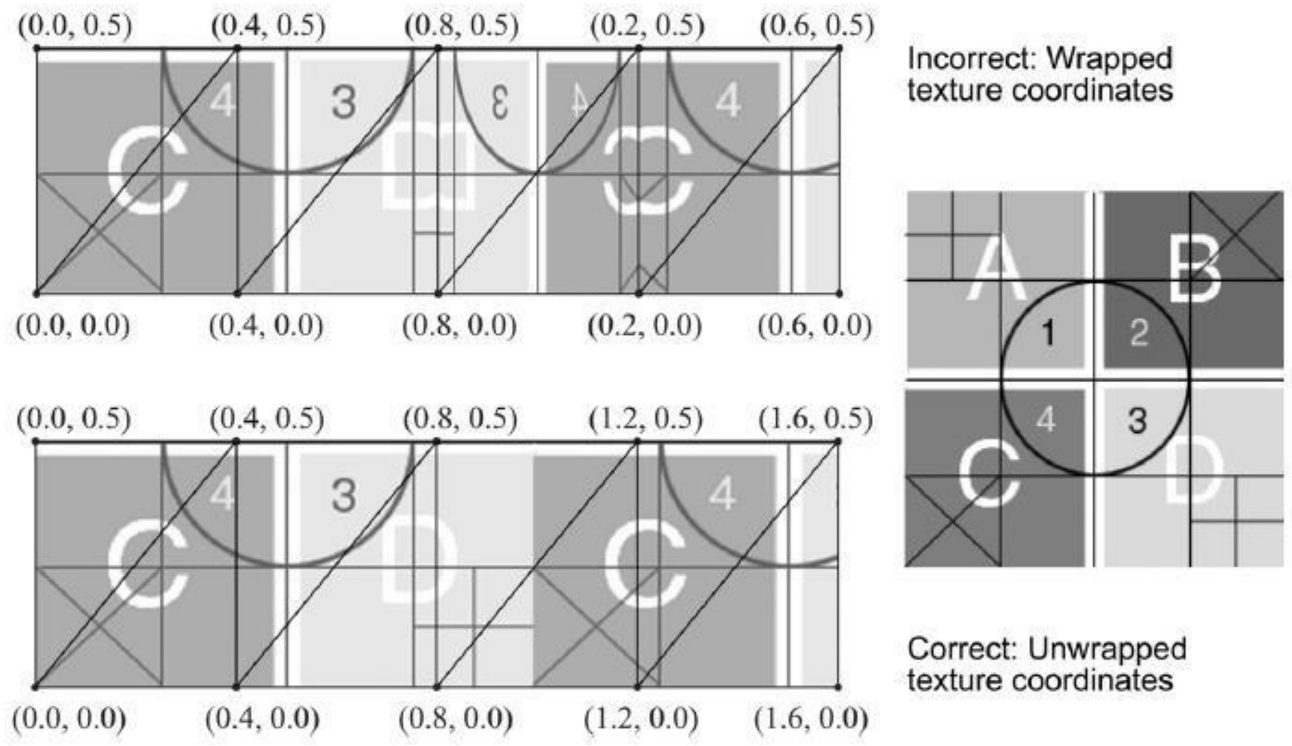
$$s = \begin{cases} s - \lfloor s \rfloor, & \lfloor s \rfloor \ even \\ 1 - s + \lfloor s \rfloor, & \lfloor s \rfloor \ odd \end{cases}$$

$$s = \max(\min(1, s - \lfloor s \rfloor), 0)$$

$$color = \begin{cases} Sample(s), & 0 \le s \le 1 \\ border, & otherwise \end{cases}$$

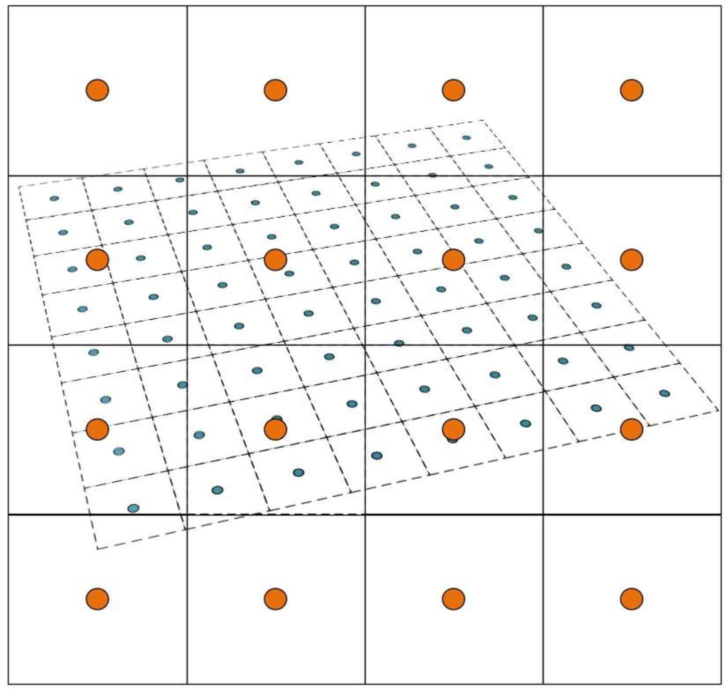
Texturing

- However, wrapping is not performed during assignment to vertices
- Interpolation may fold them back, producing erroneous results



Texturing

- When many texels correspond to a single pixel sample (area), then we have texture minification
- The texture is insufficiently sampled, resulting in distortion and noise



● Image samples ● Texel centers

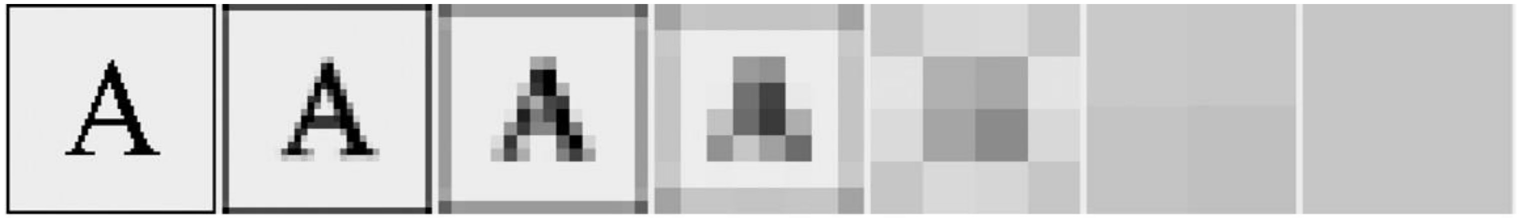
Texturing

- It is impractical to determine the pre-image texels and filter them at run-time

– The filter may just as well cover up to the entire image!

- We pre-filter the texture data using square filters of increasing size and store them
- This process is called MIP-Mapping: “Multum In Parvo” (many

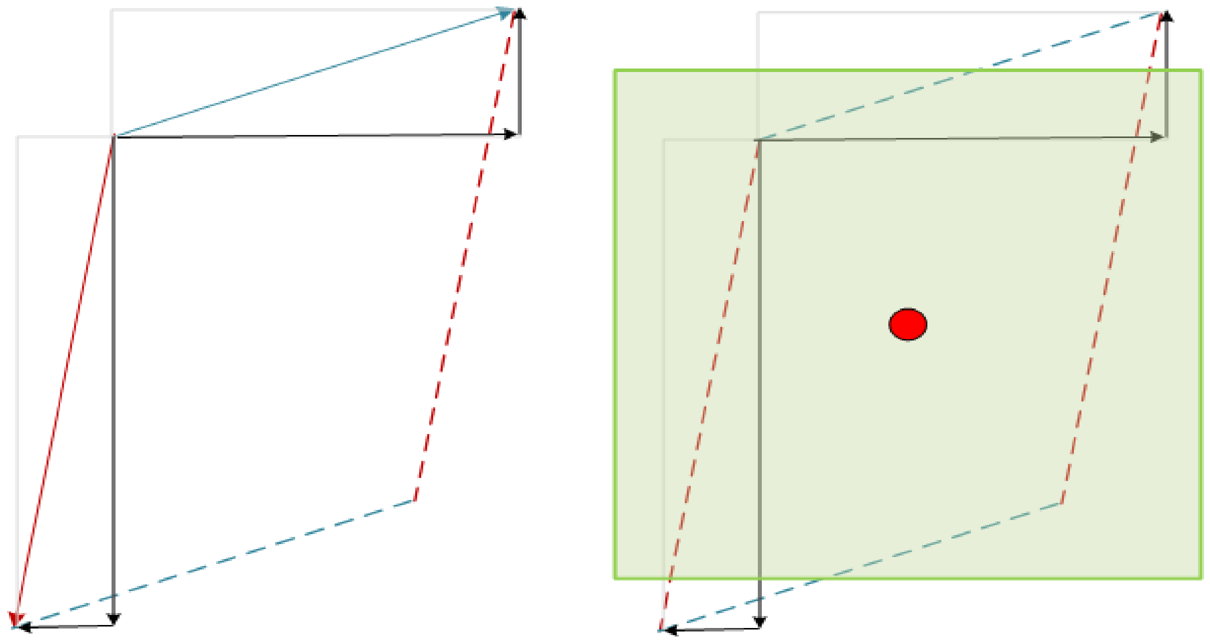
things in a small place)



Mip-map size	64X64	32X32	16X16	8X8	4X4	2X2	1X1
Filter size	1X1	2X2	4X4	8X8	16X16	32X32	64X64

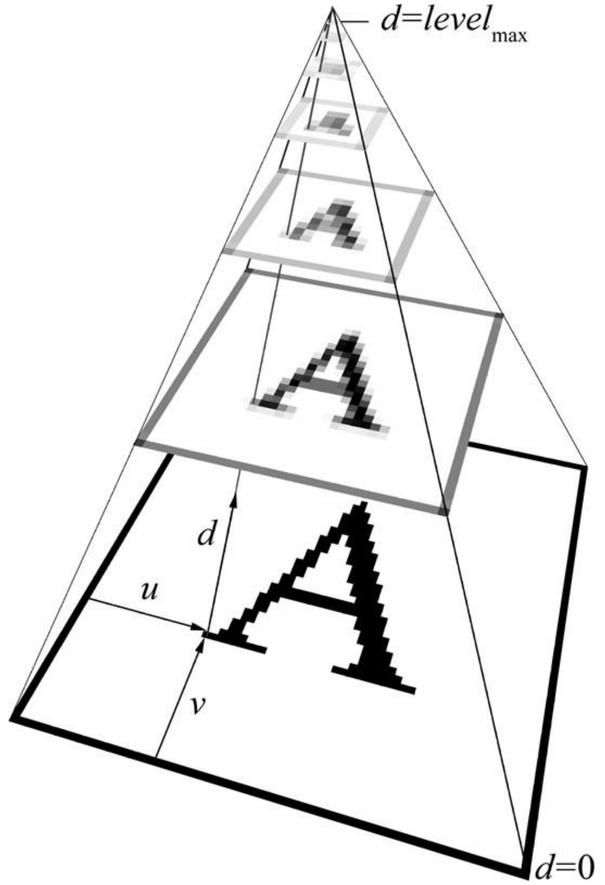
Texturing

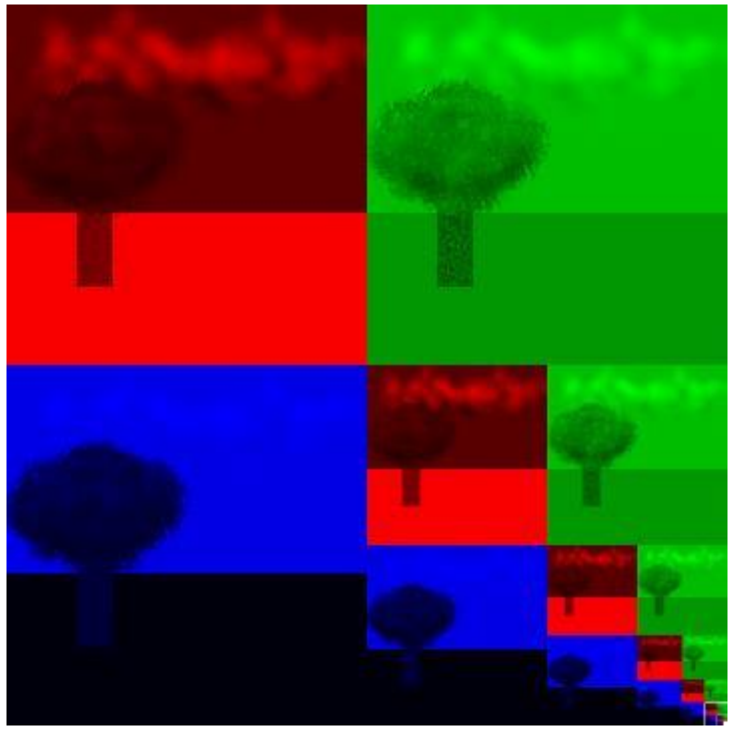
- At run time, we determine the most compatible filtered "version" of the texture and use the corresponding pre- filtered data at (u,v)
- The pre-image is approximated by a square region centered at (u,v)



Texturing

MIP Map hierarchy. We select which pre-filtered version of the image to use, according to the pixel derivatives of the uv coordinates





MIP storage. The total storage area is increased by 33%

Texturing

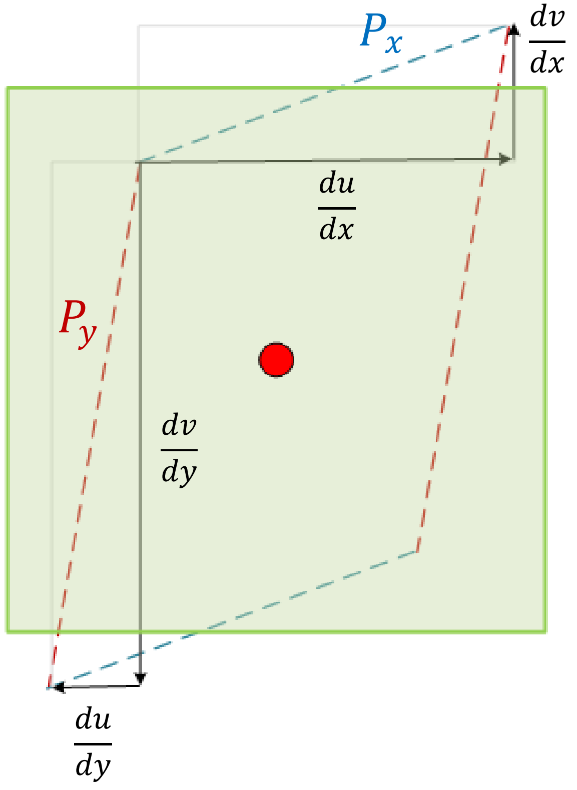
$$P_x = \sqrt{\left(\frac{du}{dx}\right)^2 + \left(\frac{dv}{dx}\right)^2}$$

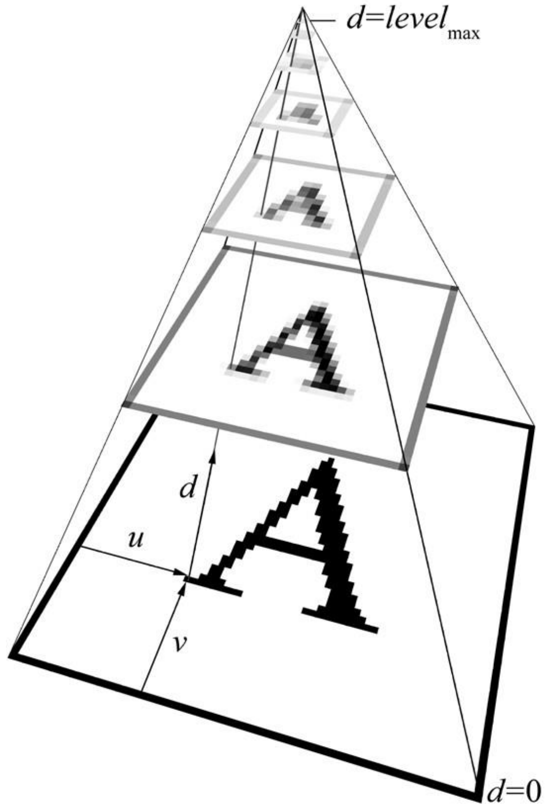
$$P_y = \sqrt{\left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dy}\right)^2}$$

$$P = \max\{P_x, P_y\}$$

$$\lambda = \log_2(P)$$

$$d = \begin{cases} level_{max} & \lambda > level_{max} \\ 0 & otherwise \\ \lambda & \lambda < 0 \end{cases}$$


P_x
$\frac{dv}{dx}$
$\frac{du}{dx}$
P_y
$\frac{dv}{dy}$
$\frac{du}{dy}$


$d=level_{max}$
d
u
v
$d=0$