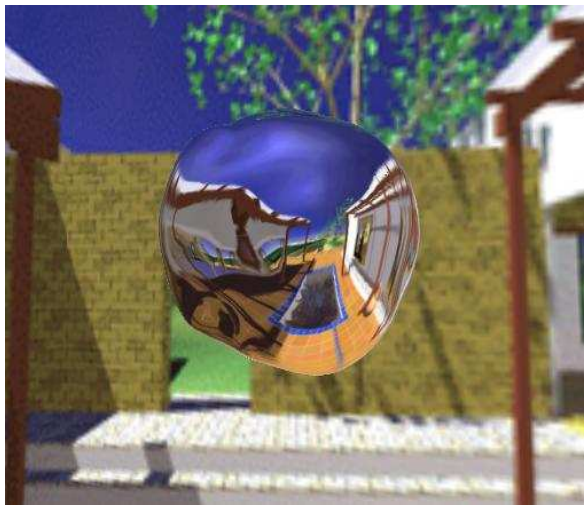
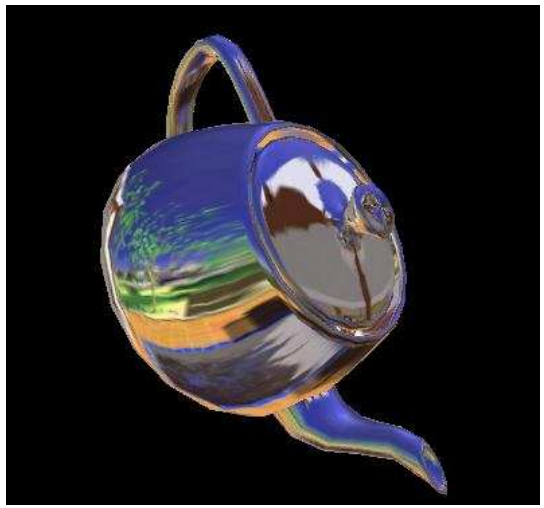
Textures

Andrzej Szymczak

October 21, 2013

Texture mapping

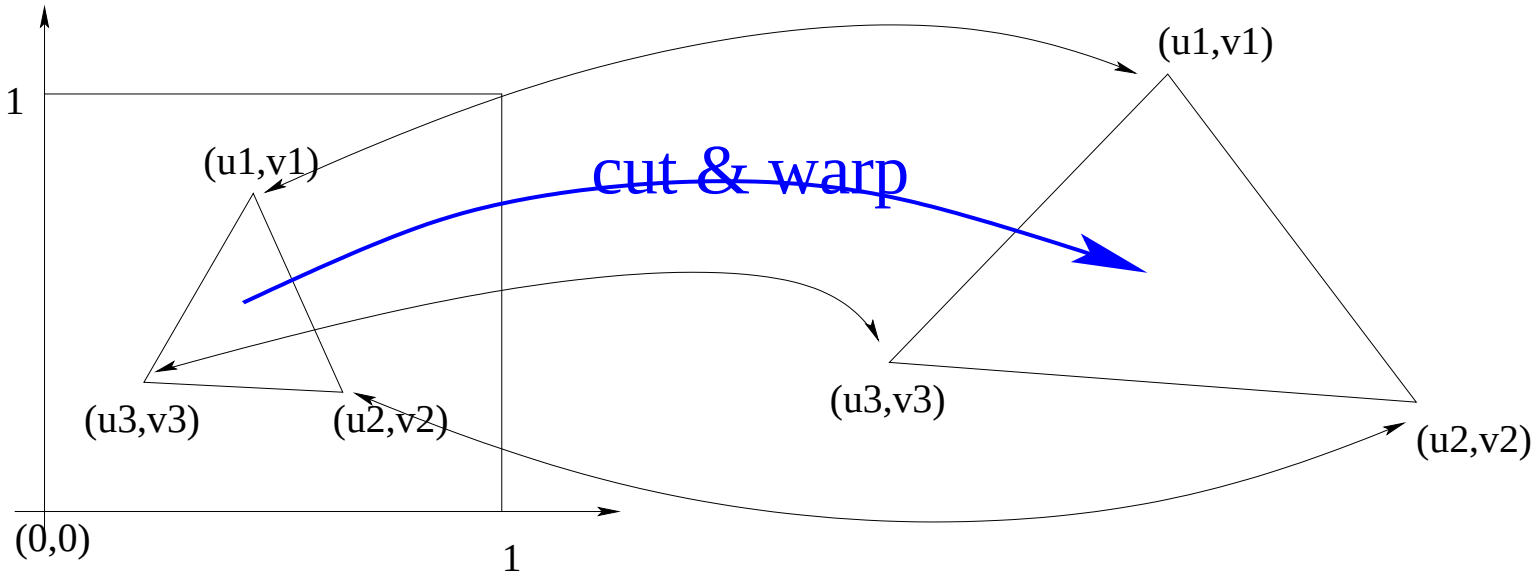
- Texture: assigns a subset of material properties (e.g. color, transparency, parameters describing light scattering...) to a point of the 1-, 2- or 3- dimensional space
- Textures are applied to the primitives
- Applied means that the data looked up from texture is used for color computation

Texture mapping in graphics pipeline

- Specify *texture coordinates* for every vertex
  - Can be a separate vertex attribute
  - ... or can be derived from other attributes
- Programmer has full control over how texture coordinates are used
  - They can be transformed during any programmable stage
    - The transformation is sometimes represented as *texture matrix* (in modern OpenGL, a uniform variable)
  - Texture lookup is allowed at each pipeline stage
    - Typically, during fragment processing

Texture mapping: typical scenario

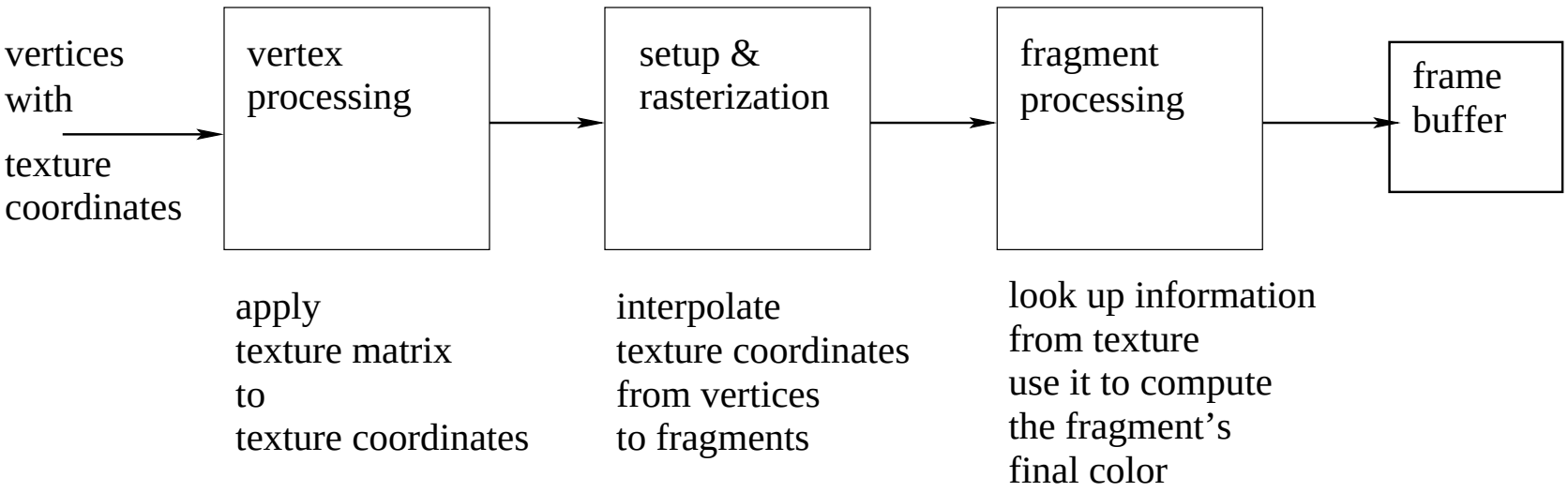


texture space

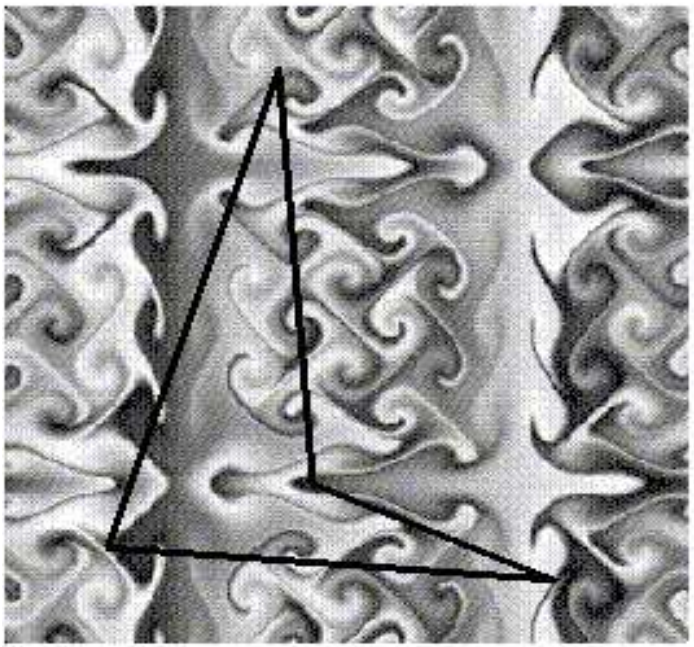
3D triangle with texture coordinates at the vertices

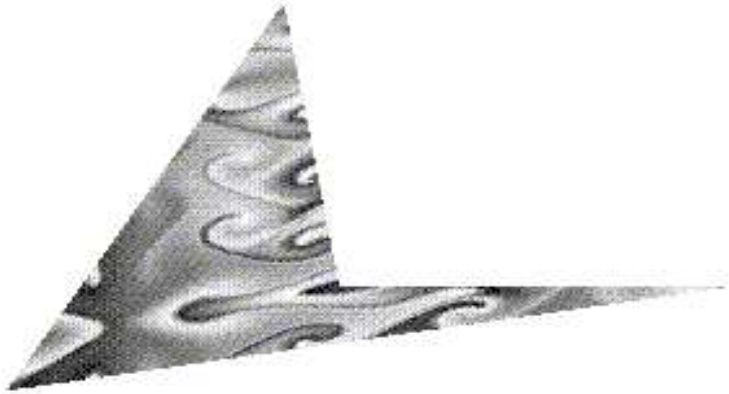
Texture mapping in graphics pipeline

- Typical use of texture in modern OpenGL
- The texture matrix is optional, but often convenient

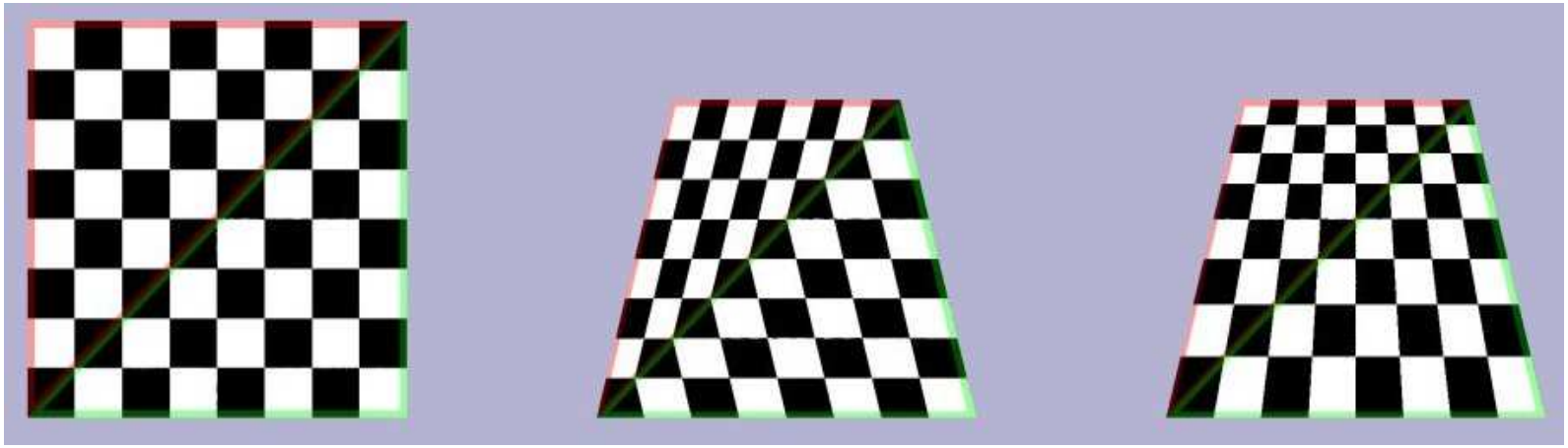


Texture mapping in graphics pipeline





Texture mapping in graphics pipeline: interpolation



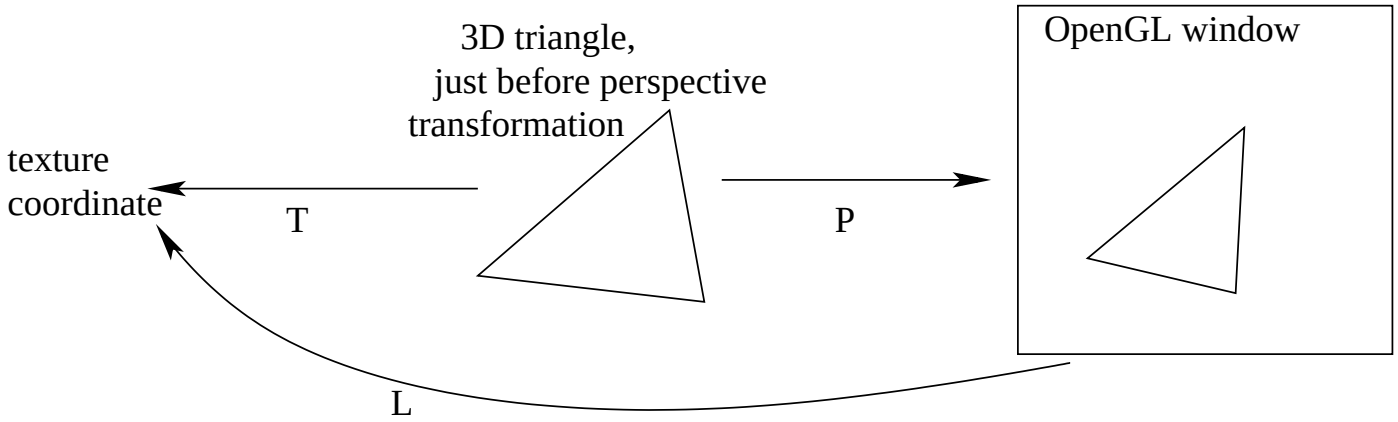
Parallel view

Linear interpolation of texture coordinates in screen space

Perspectively correct interpolation (linearly maps texture into the 3D triangles)

- Hardware supports perspectvely correct interpolation for texture coordinates
- Equivalent to performing two linear interpolations and dividing the results

Perspectively correct interpolation: why ratio of two linear functions?



- T: a mapping that assigns a texture coordinate to a point in the triangle; linear
- P: perspective transformation, $P(x, y, z) = (x/z, y/z, 1 - 1/z)$
- $L = T \circ P^{-1}$: function assigning perspectively correct interpolated texture coordinate to a point in the projection of the triangle
  - L is used to generate texture coordinates for fragments!

Why ratio?

- $L = T \circ P^{-1}$
- $P(x, y, z) = (x/z, y/z, 1 - 1/z)$
- Not hard to see that $P^{-1}(x, y, d) = (x/(1-d), y/(1-d), 1/(1-d))$
- T is linear, $T(x, y, z) = Ax + By + Cz + D$
- So: $L(x, y, d) = T \circ P^{-1}(x, y, d) = \frac{Ax+By+C+D(1-d)}{1-d}$.
- Notice that when L is evaluated for a fragment, d is the depth of the fragment; we know that it is a linear function of x and y, say $d = ax + by + c$
- ... putting all of it together: for a fragment at (x, y) with depth d, the interpolated texture value is $L(x, y, ax + by + c)$
- By the above formula for L, this is a ratio of two linear expressions in x and y
- Of course, it takes more careful matrix algebra to derive the actual formula for their coefficients; can be done though

Types of textures

- What does ‘lookup’ mean?
  - Procedural textures [evaluation of a formula]
  - Array textures [interpolation from samples]

Procedural textures

- We can specify a texture data using a formula, for example assigning a color to texture coordinates
- This is how *procedural textures* work
- In the graphics pipeline, one can implement procedural textures in the fragment shader or use the formula to generate an array based texture (discussed soon) on the CPU

Example

Formula for RGB values if texture coordinates are (x, y, z):

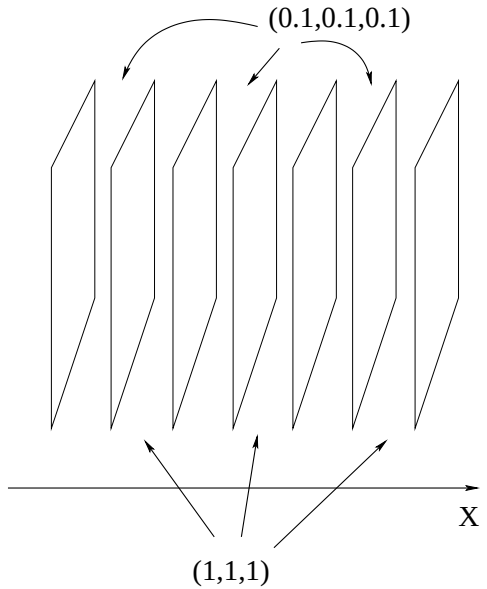
$$\begin{cases} (1,1,1) & \text{if } \sin(\omega x) > 0 \\ (0.1,0.1,0.1) & \text{otherwise} \end{cases}$$

What if this texture is used with texture coordinates equal to model coordinates for each vertex?...

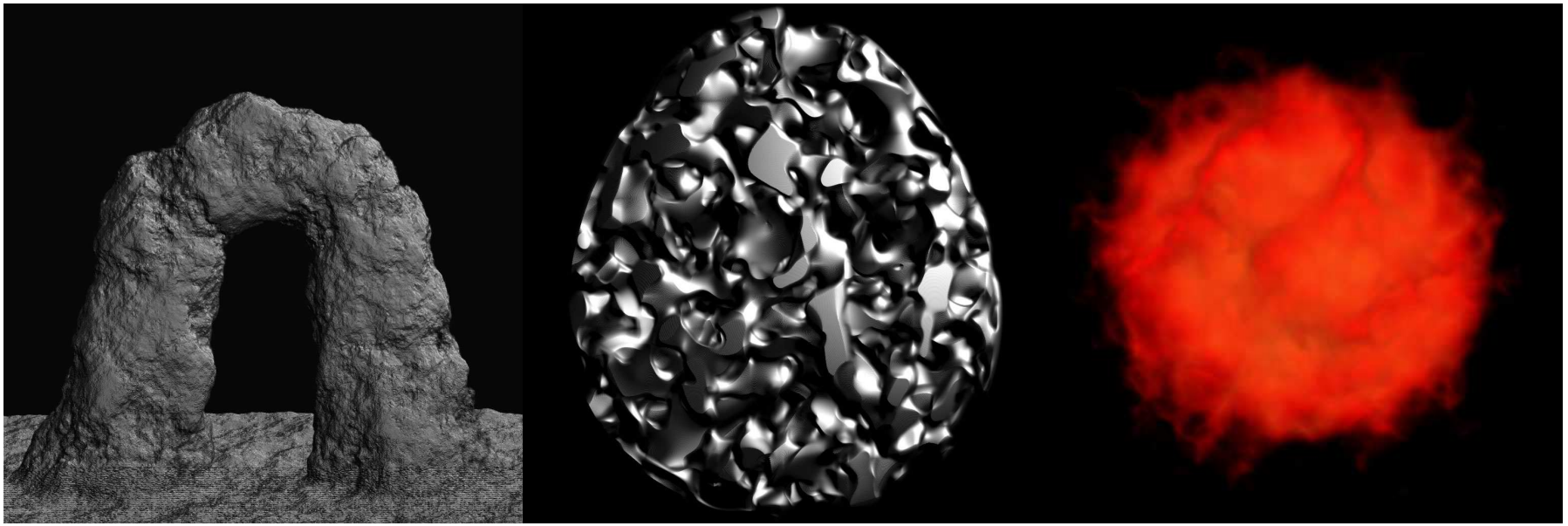
Procedural textures

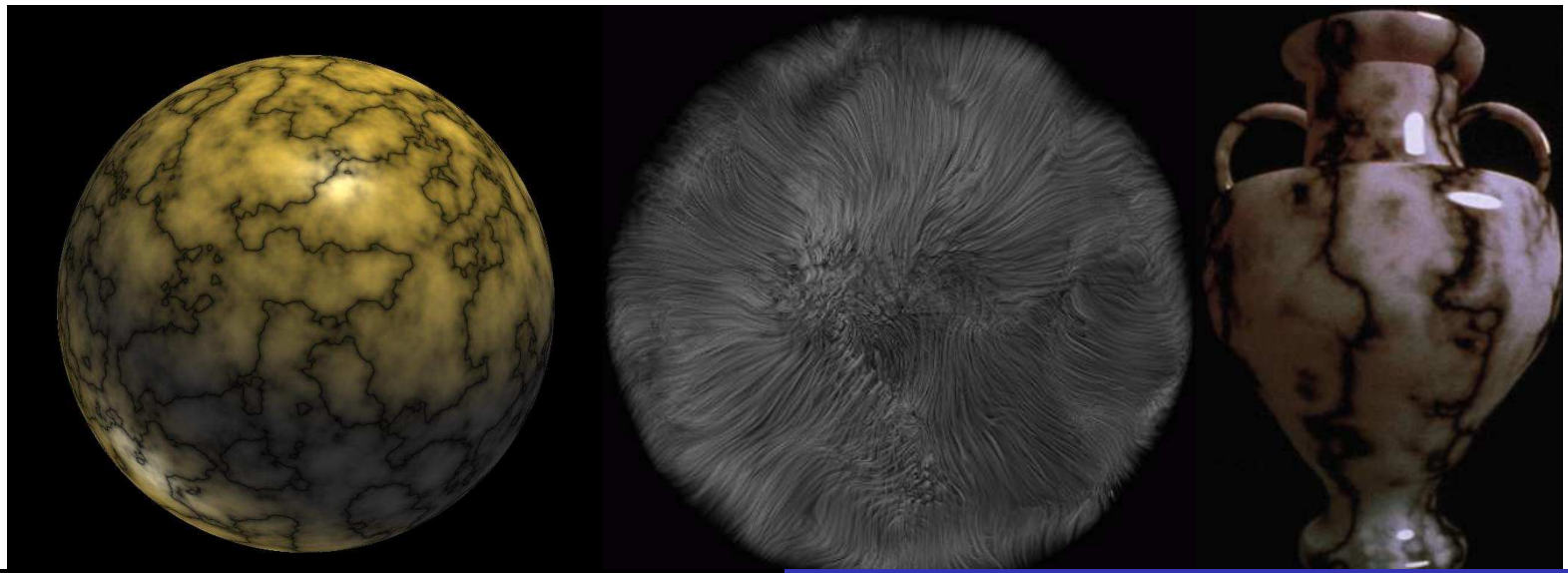
... We get stripes on the 3D model.

- Model coordinate space is divided into slabs, fragments corresponding to points in every second slab are bright and fragments from the other slabs are dark
- Linear interpolation of the texture coordinates yields the coordinates of the point on the object's surface that corresponds to the fragment



Procedural textures: not just a toy!



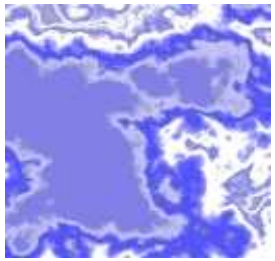


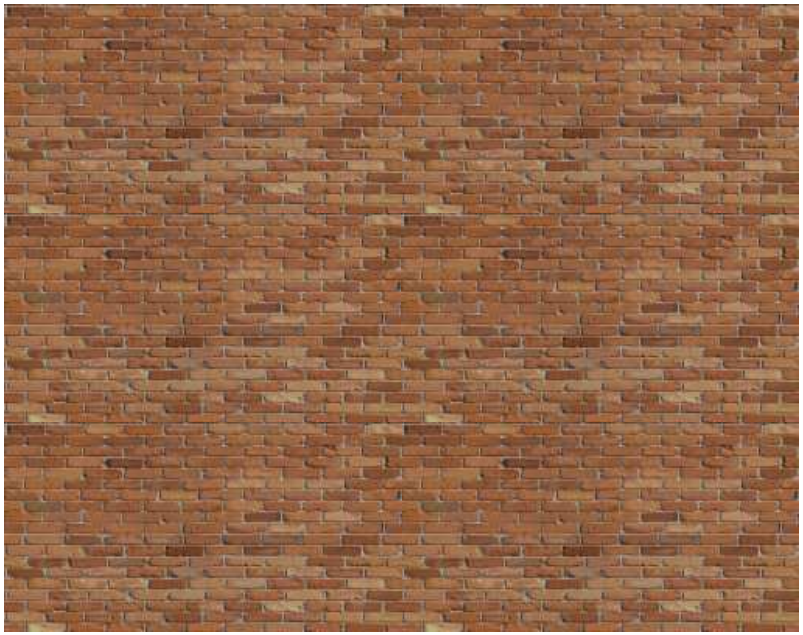
Array based textures

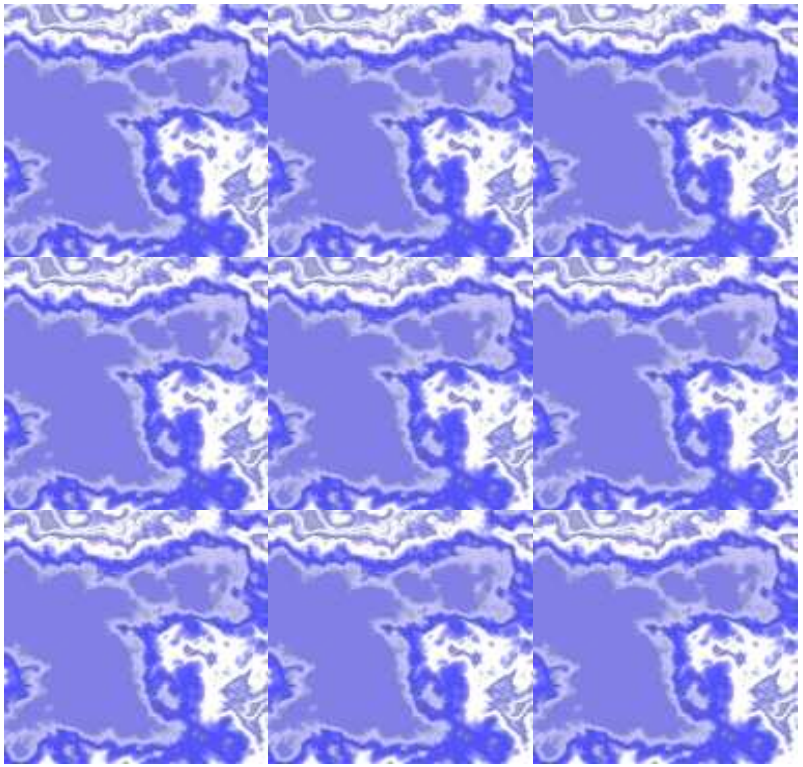
- Store the texture data in an array (1-, 2- or 3- dimensional)
- 2D array textures are very easy to get; for example, just take a photo of a piece of wood and then map it onto your 3D models to make them appear as if made of wood
- When looking up color from texture in GLSL, it is assumed that the image spans texture coordinate values between 0 and 1 (there are exceptions, e.g. texture rectangles)
- There are several ways of dealing with texture coordinates outside the range, for example:
  - Clamp: if a texture coordinate is greater than 1, replace it with 1; if less than 0, replace it with 0
  - Repeat ('tile'): replace a texture coordinate not in $[0, 1]$ with its fractional part

Array based textures: repeat

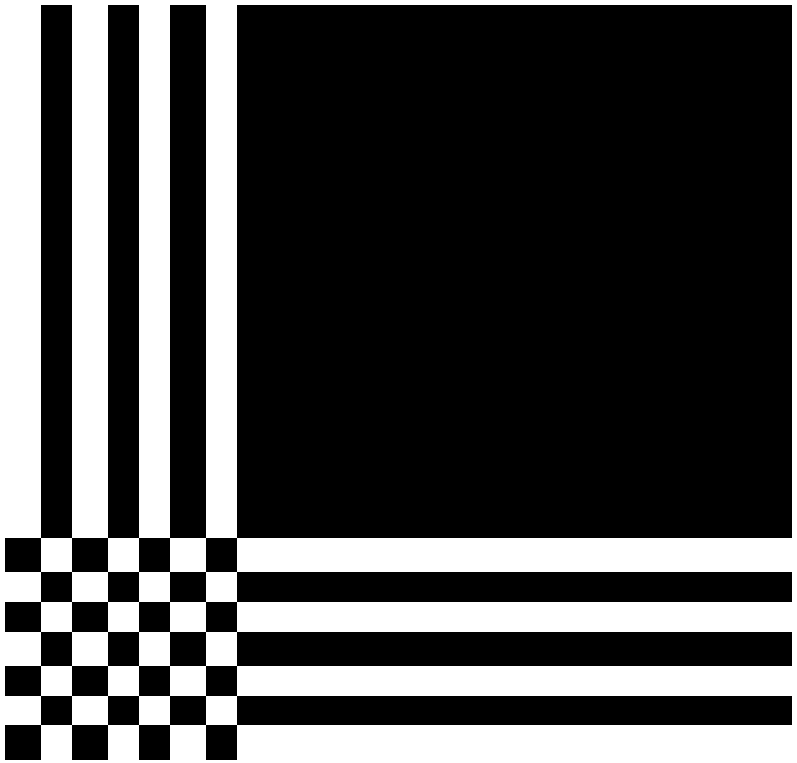
tilable texture vs non-tilable texture

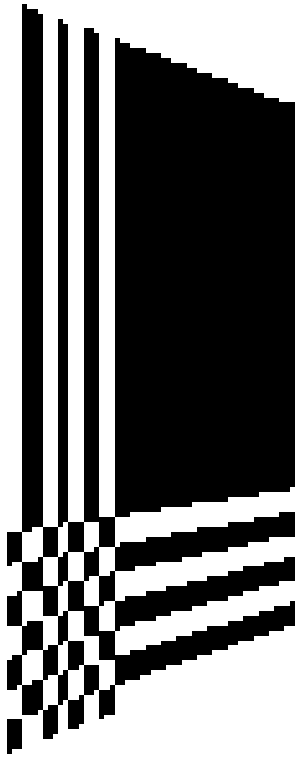






Array based textures: clamp





Array based textures: lookup; 2D case

- Assume the image resolution is $n_x \times n_y$ and the interpolated texture coordinates are u,v
- We'll assume $0 \leq u, v < 1$ (if not, clamp or repeat)
- There are two popular interpolation methods
  - Nearest pixel: look up data from pixel (i,j) where:

$$i := \lfloor u n_x \rfloor$$
$$j := \lfloor v n_y \rfloor$$

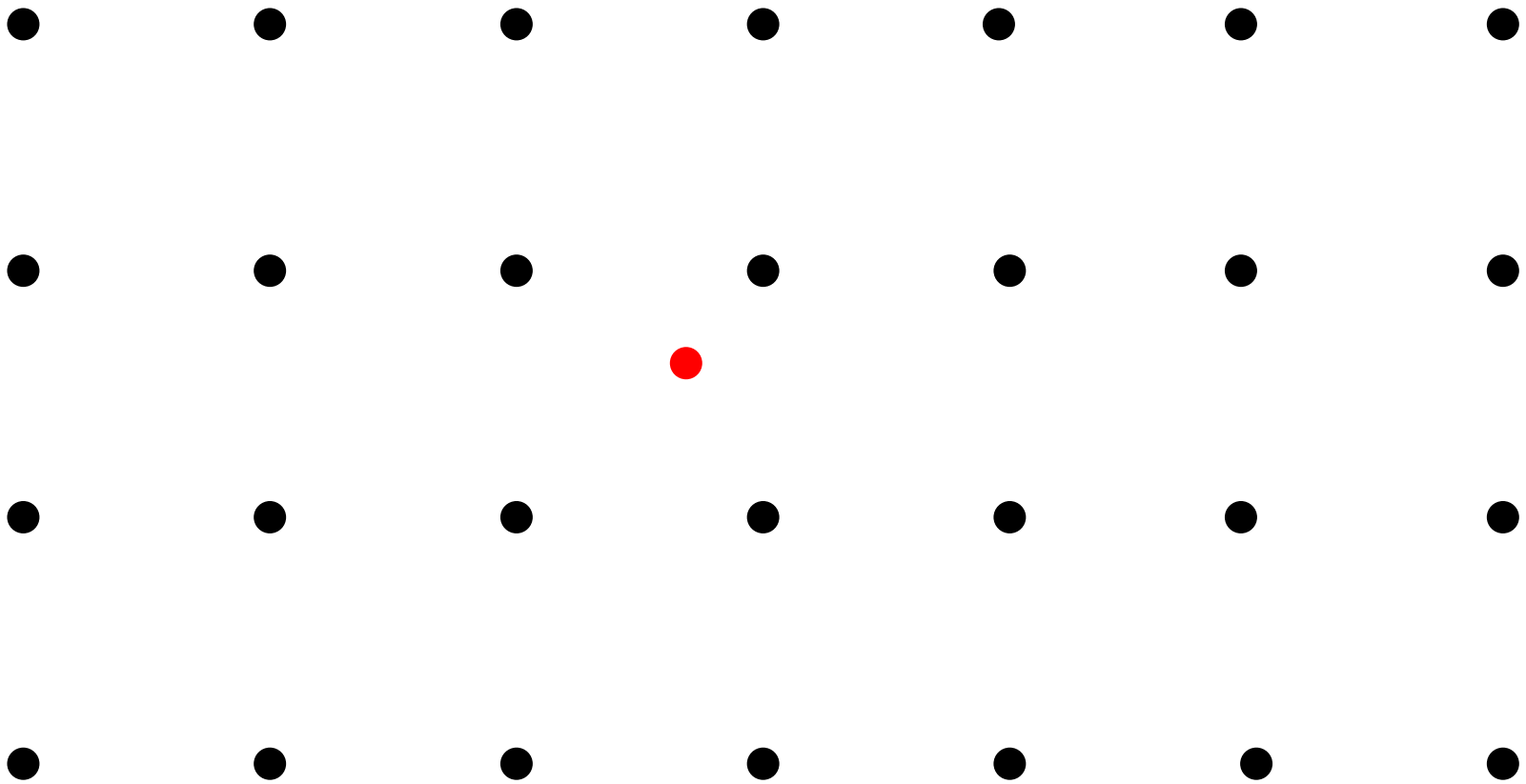
  - Bilinear interpolation: average the values of 4 pixels (i,j), $(i+1,j)$, $(i,j+1)$ and $(i+1,j+1)$

$$u' := n_x u - \lfloor n_x u \rfloor$$
$$v' := n_y v - \lfloor n_y v \rfloor$$

The interpolated value is:

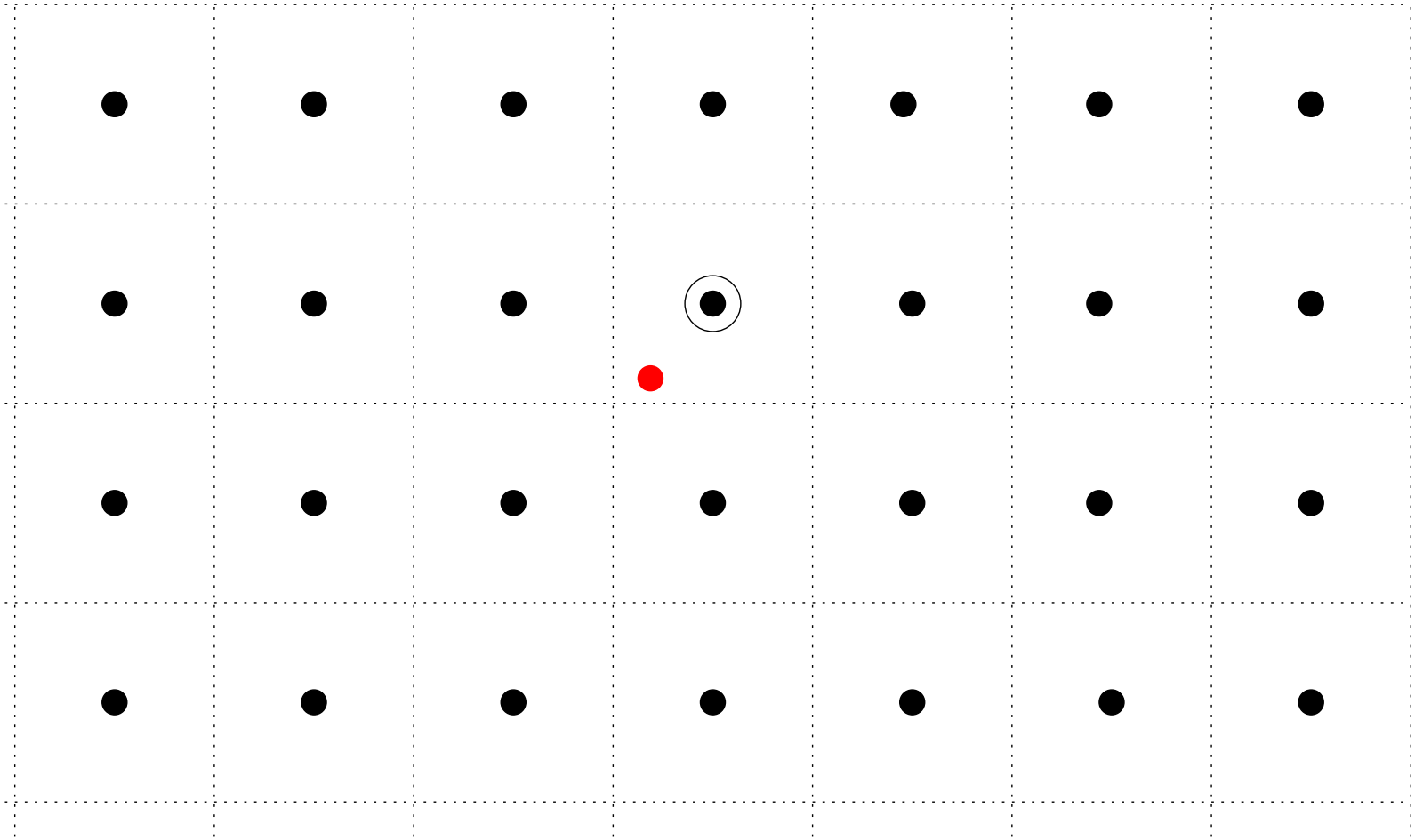
$$(1-u')(1-v')c_{ij}+(1-u')v'c_{i(j+1)}+u'(1-v')c_{(i+1)j}+u'v'c_{(i+1)(j+1)}$$

Array based textures: lookup

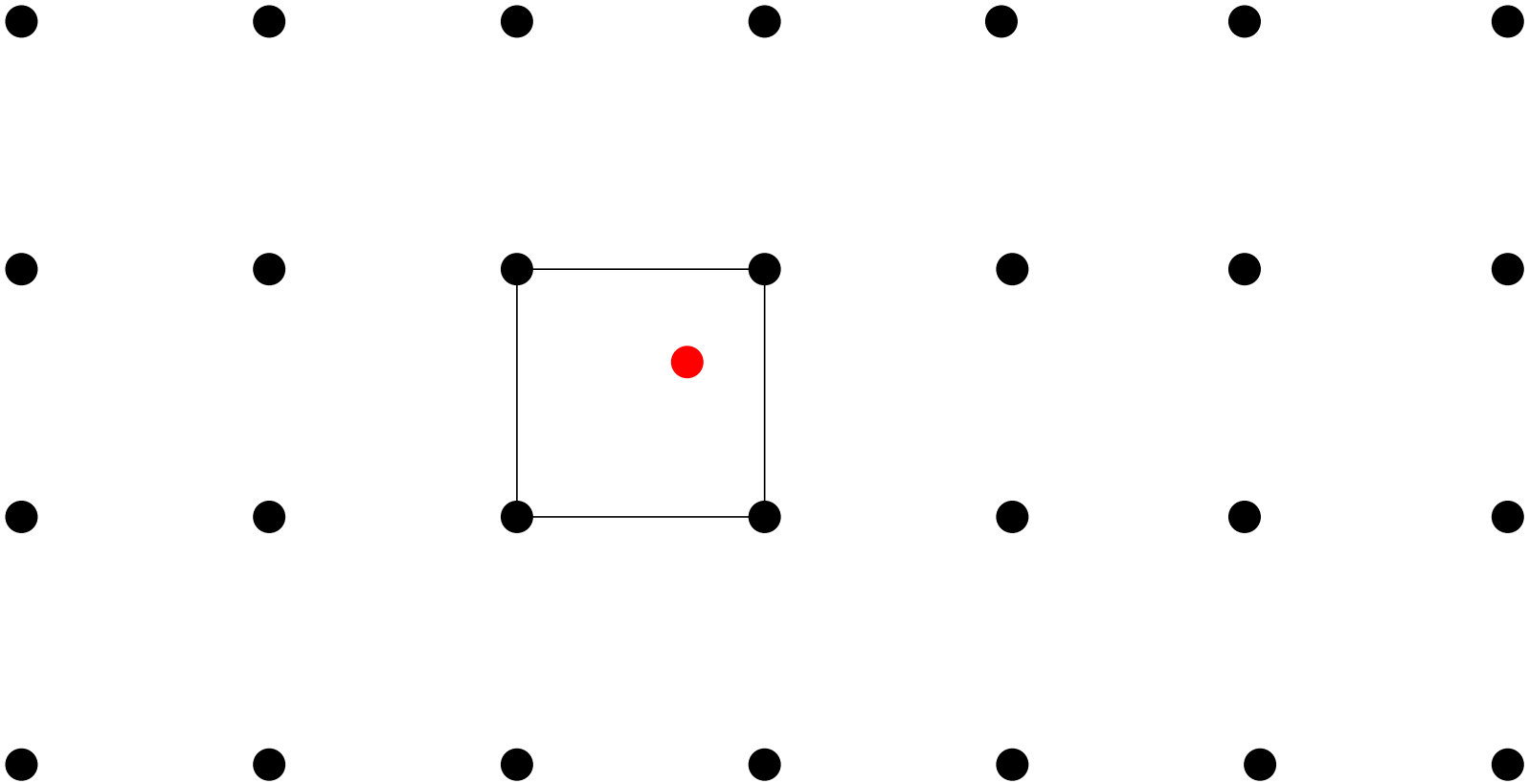


Array based textures: lookup; nearest pixel

'cells' of the same color: discontinuities along the cell boundaries

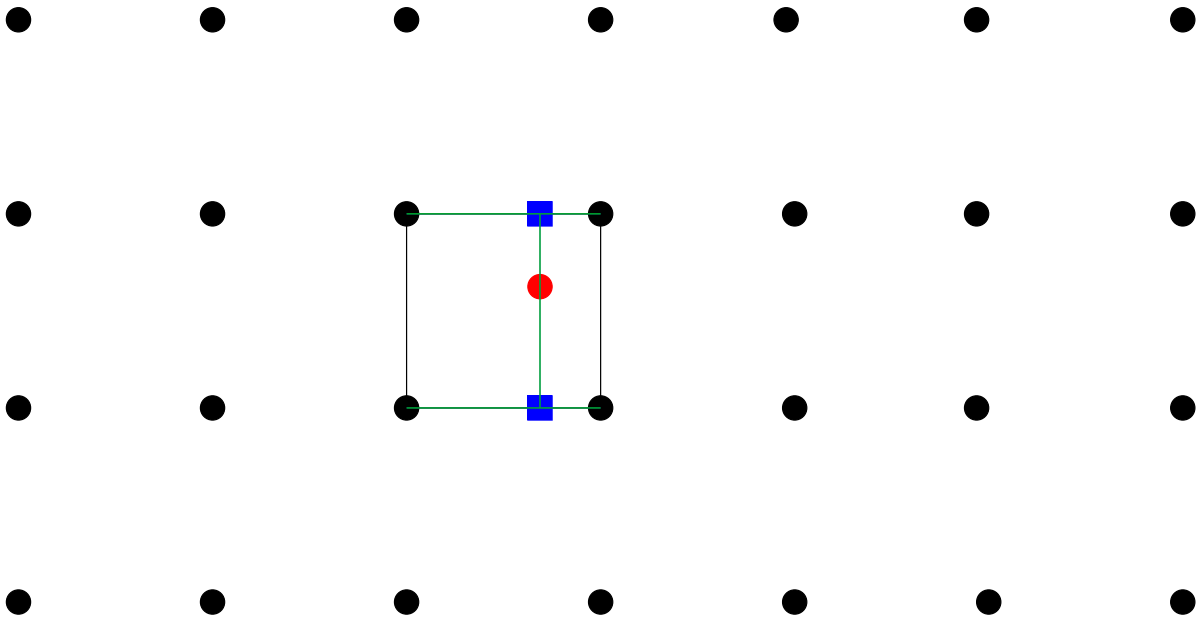


Array based textures: lookup; bilinear

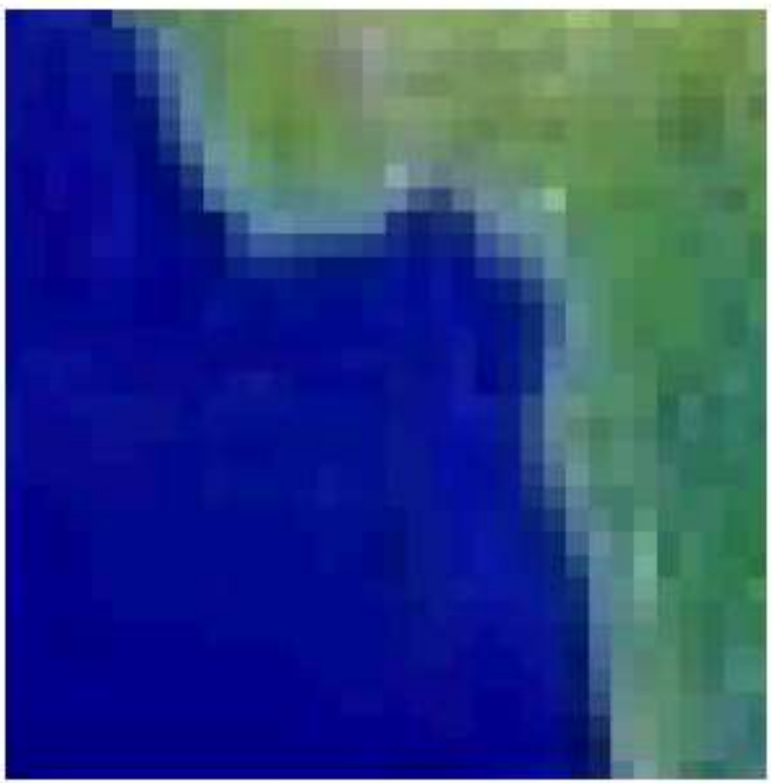


Array based textures: lookup; bilinear

- Linearly interpolate from pixels to blue squares along horizontal edges
- Then, linearly interpolate from blue squares to red circle
- Color varies continuously (better quality)
- Slower than nearest pixel lookup
- Bilinear vs Nearest lookup is a bit like Gouraud vs Flat shading in terms of both speed and quality



Nearest neighbor vs bilinear



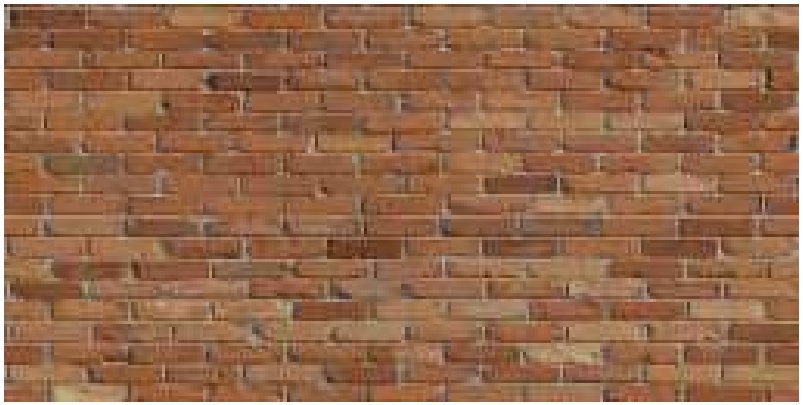
Nearest

Bilinear

Examples

Example: tilable texture on a torus

- Glue horizontal edges to get a brick cylinder
- Glue the ends of the cylinder to get brick torus



Example: volumetric textures

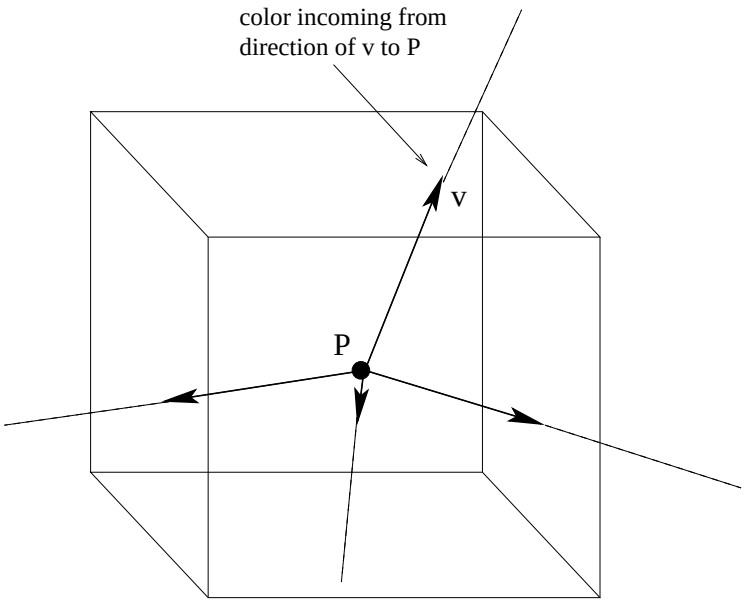
- A popular way of applying 3D textures: just use model coordinates scaled to 0...1 as texture coordinates
- Equivalent to carving the model from a block of material with color distribution provided by texture

Environment mapping

- Can be used to render mirror 3D objects (possibly, of complex geometry)
- Small region of interest R
- Assumption: Amount/color of light arriving at any point of R from a fixed direction $\vec{d}$ is the same, i.e. depends only on $\vec{d}$
- Never satisfied? Never mind!
- Incoming light: a colorful sphere/spherical image
- Capturing incoming light for all directions is not hard
- Spherical and cube (or cubical) environment maps

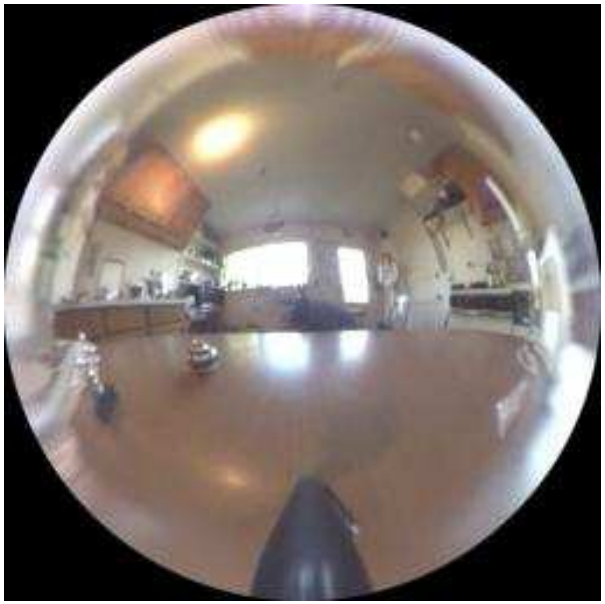
Cube environment map

- Can be built from six wide angle (90 degree field of view) square photographs
- If you don't have a wide angle lens, stitch more photographs together (omnidirectional panorama – similar to the panoramic images)
- For synthetic environments, really easy to make (just render 6 times)



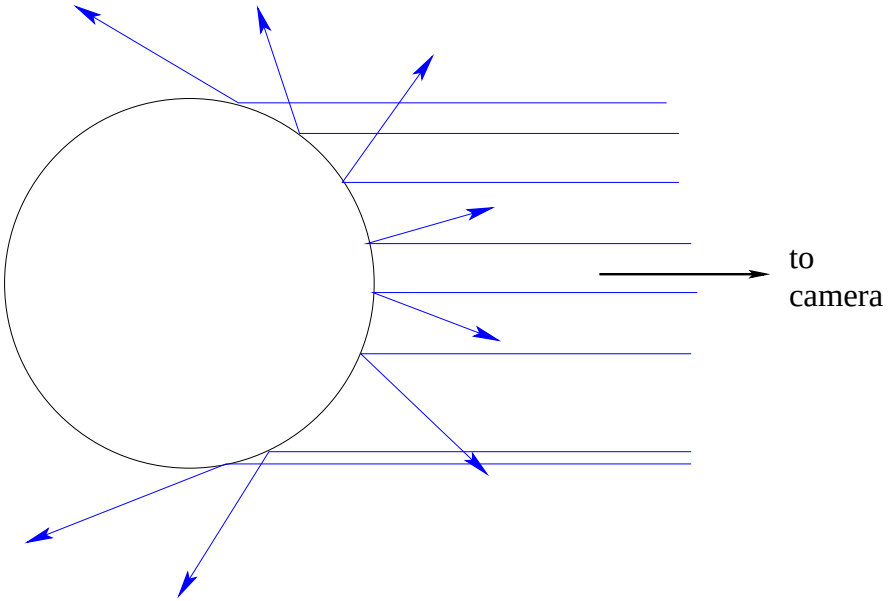
Spherical environment map

- A single photograph of a mirror sphere placed in the region of interest
- No need to stitch images together (one image is enough)
- Ideally, use a long [zoom] lens – you want to get as close to a parallel projection as you can



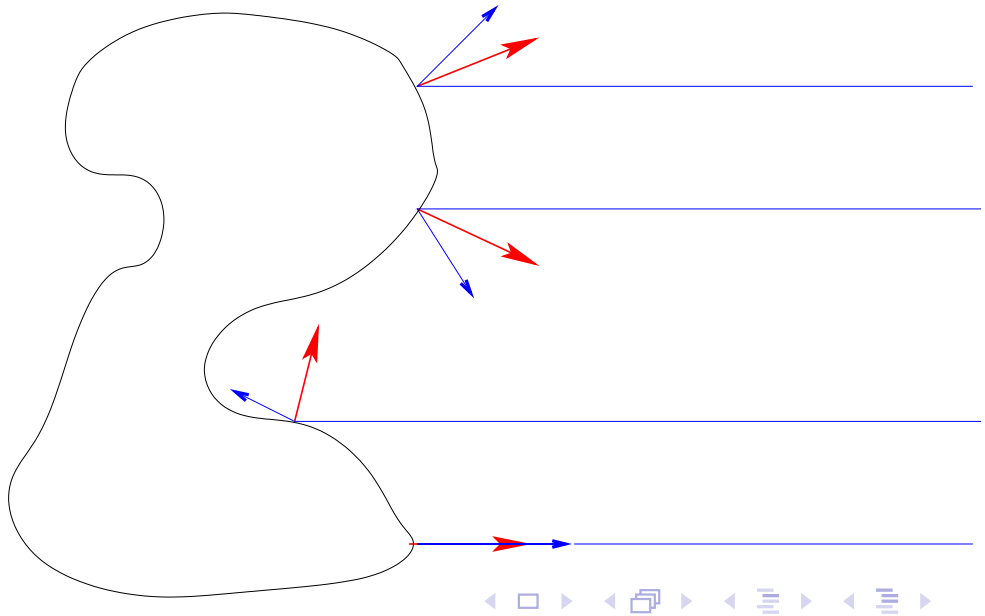
Spherical environment map

- A single photograph of a mirror sphere placed in the region of interest
- Incoming color is captured for every incoming direction (if the projection is parallel)
- Can be surprising at first
- Note the 'singularity' at silhouette points; this causes problems when using the same spherical map for different views



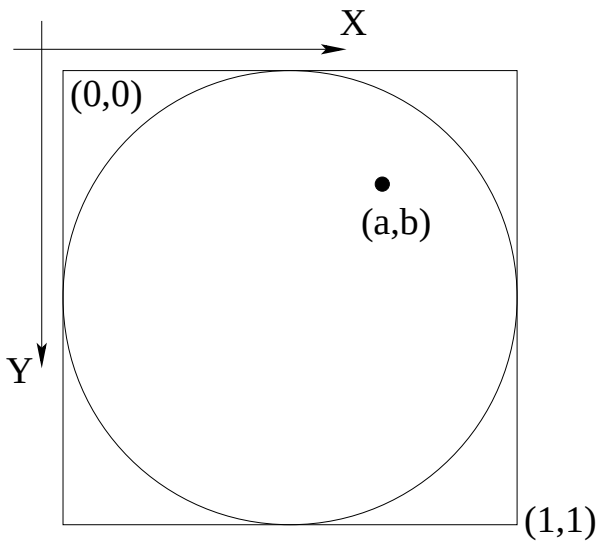
Mirror objects using the spherical environment map

- Spherical environment maps can be used to draw images of mirror objects in the region of interest, using the view identical to that of the camera used to take the picture of the mirror sphere
- Really simple for parallel view
- Use a simple approximation that ignores multiple reflections
- Key observation: reflected ray direction determined by the normal

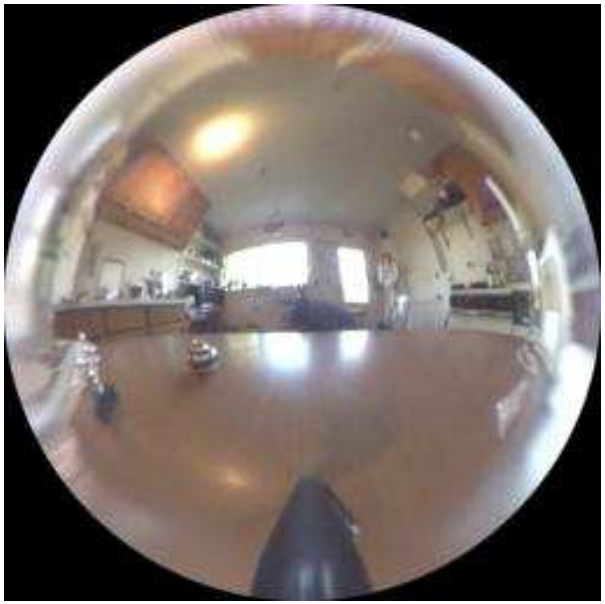


Mirror objects: math

- Assume the projection is in the direction of z-axis
- Assume mirror sphere centered at the origin and of unit radius
- Normal to the mirror sphere under the point (a, b) (in texture space): $(2a-1, 2b-1, \sqrt{1-(2a-1)^2-(2b-1)^2})$
- ... therefore, point on the sphere with unit normal (n_x, n_y, n_z) is under the point $\left[\frac{1+n_x}{2}, \frac{1+n_y}{2}\right]$ in the texture space
- Texture coordinate for a vertex with unit normal (n_x, n_y, n_z): $\left[\frac{1+n_x}{2}, \frac{1+n_y}{2}\right]$; point on sphere under that point and the vertex have the same normals



Mirror objects: Feline in the kitchen



Materials and/or complex illumination

- The sphere used to capture the environment does not necessarily be mirror