Textures in real-time graphics

© 2004-2018 Josef Pelikán
CGG MFF UK Praha

pepca@cgg.mff.cuni.cz
http://cgg.mff.cuni.cz/~pepca/

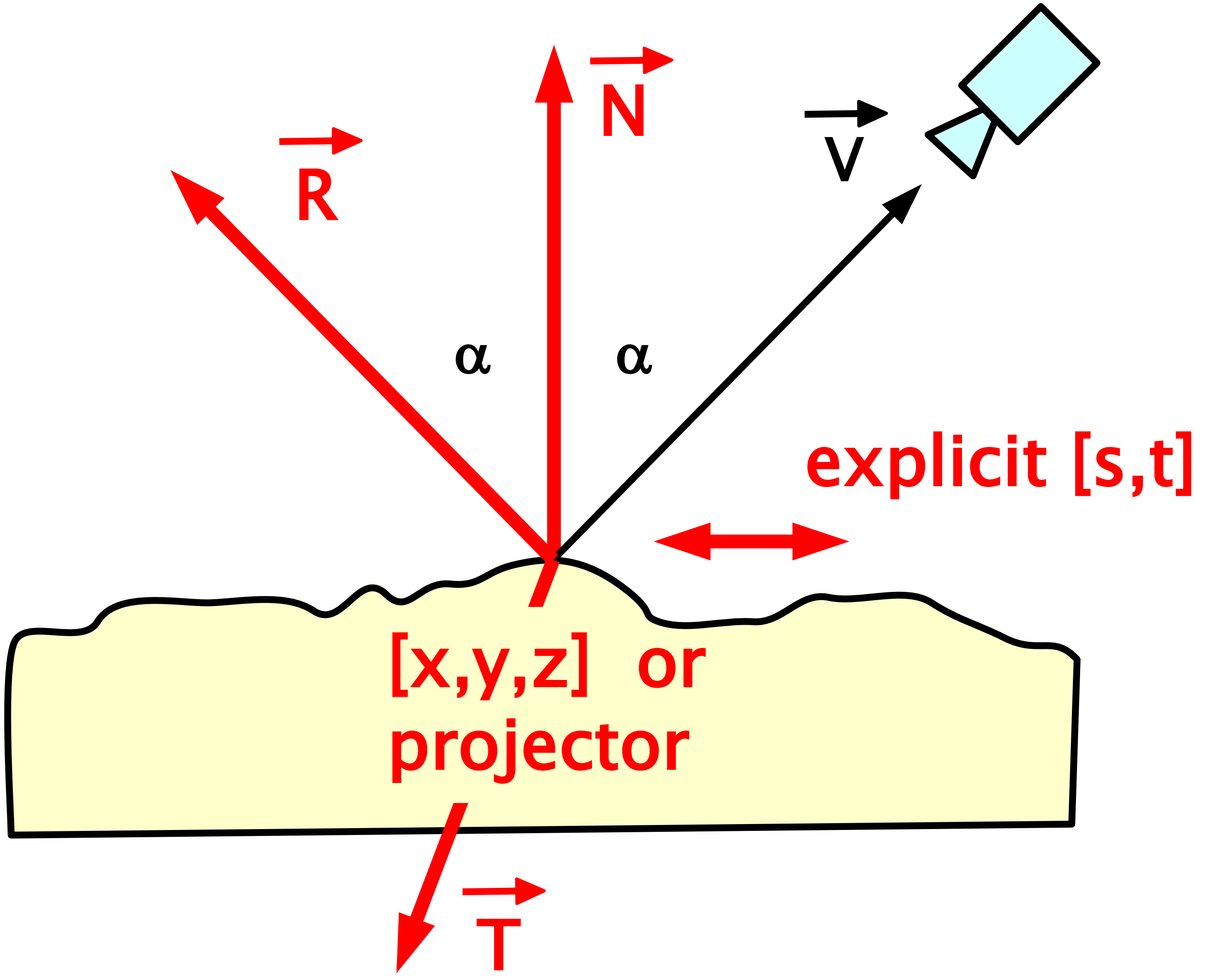
Textures

- **appearance enhancement**
  - **color** modulation (raster image = “bitmap”)
  - **“bump-texture”** (substitution for detailed geometry)
  - possible modulation of **more quantities**: transparency, reflectance, environment light

Texture definition:

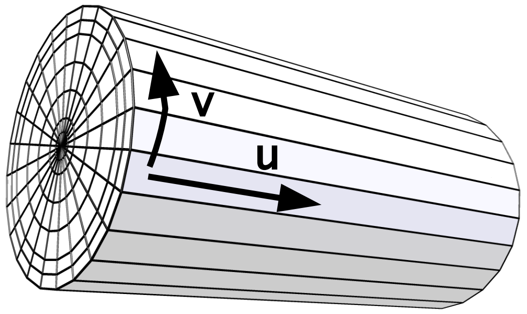
- **1D, 2D data array** (“bitmap texture”)
  - more common, HW capability
- **3D data array** (“volume texture”)
- **procedural** – callback algorithm in every fragment (programmable GPU)

Texture domain



Texture mapping

- **2D textures** have to be **mapped** to an object surface
  - **texture coordinates [u, v]** ([s, t] in OpenGL) defined in every vertex
  - GPU interpolates them correctly into individ. fragments
  - bitmap data need to be interpolated (among adjacent texture pixels = “**texels**”)



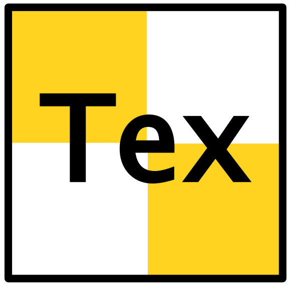
Automatic "projectors"

- old OpenGL was capable of some **3D → 2D projec-tions**
  - implicit (procedural) mapping
- **available projectors** (see glTexGen)
  - spherical (singularities at the poles)
  - cylindrical
  - linear projection (often/as well for 1D textures)
- configurable **texture-transform matrix**
- modern solution: **vertex shaders**

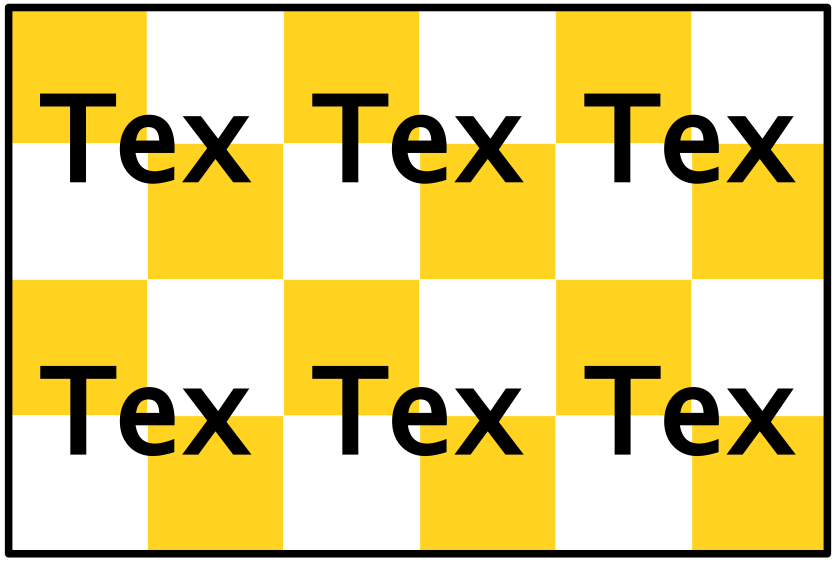
Texture repetition

- **standard** texture-coordinates domain: **[0,1]^D**
  - handling of out-of-range values?
- **cyclic repetition** (repeat, wrap, tile)
- **mirroring** (mirror, flip)
  - every other tile is flipped
  - better continuity
- **nearest texel** (clamp, clamp to edge)
  - robustness on the texture border
- **explicit border value** (border, clamp to border)
  - one value or borderline row/column of the bitmap

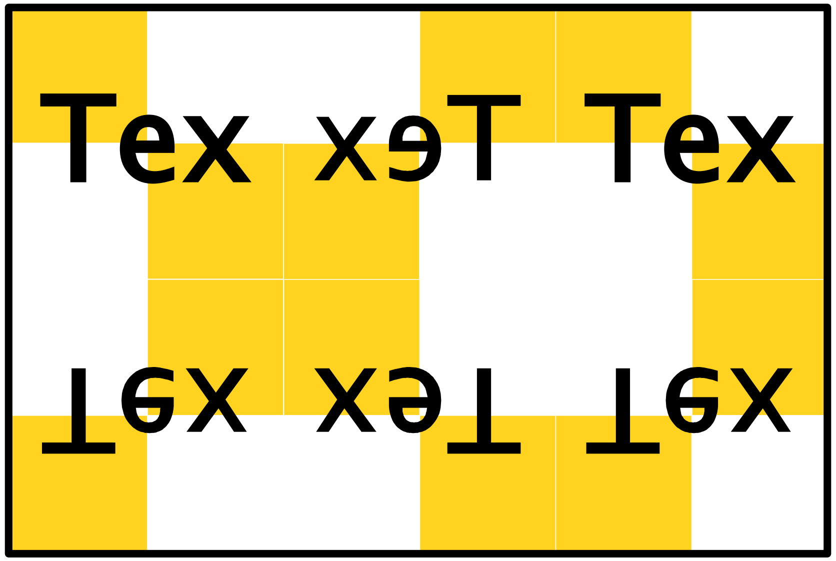
Repetition: repeat, mirror



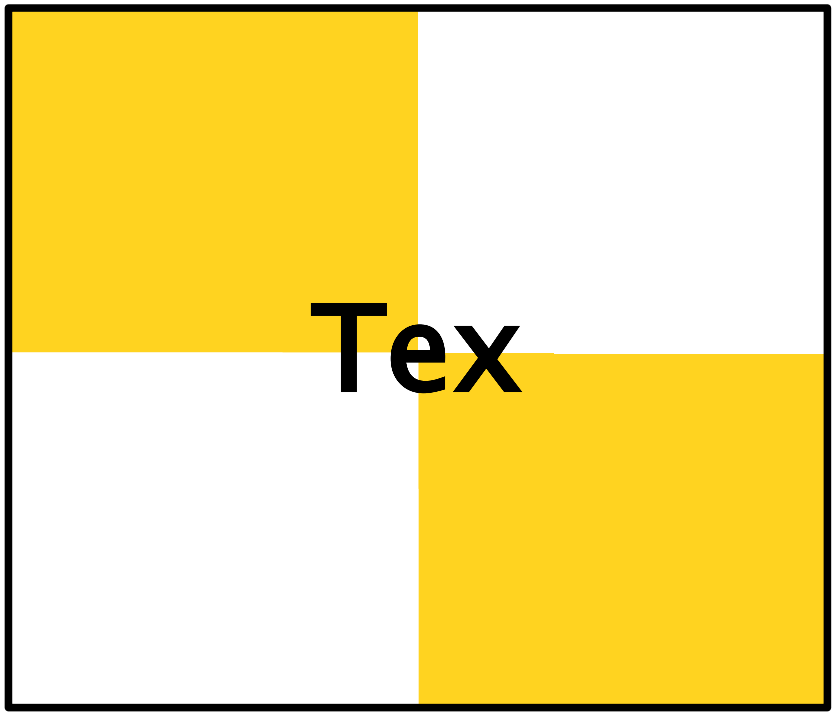
repeat



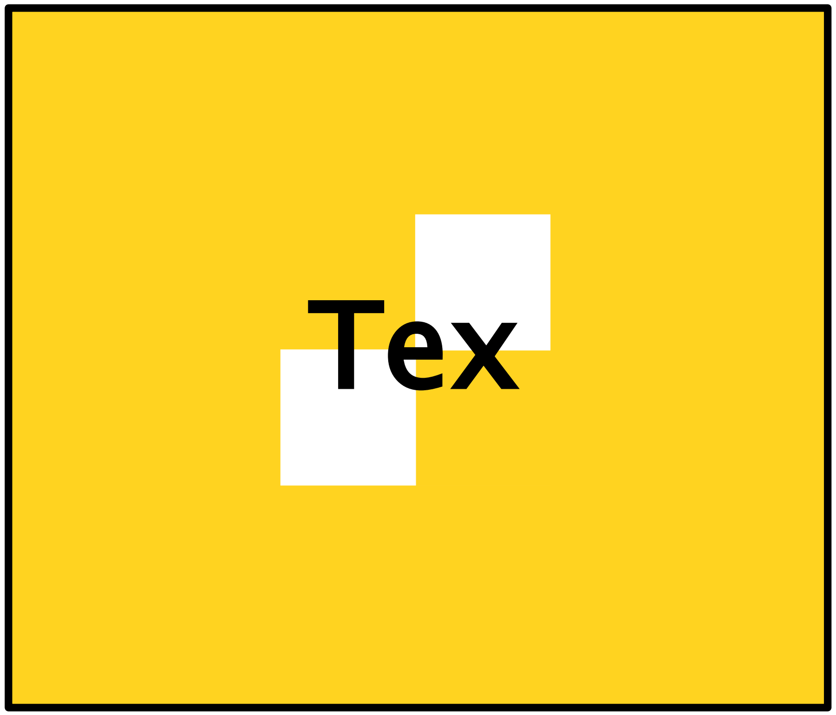
mirror



Repetition: clamp, border



clamp



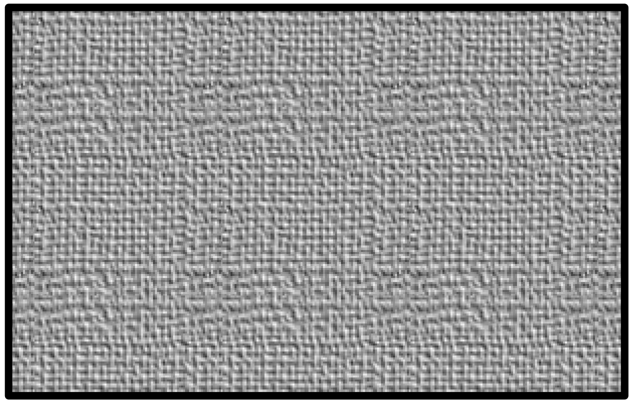
border

Texture–coordinate transform

- matrix transform performed by a GPU
  - general **homogeneous matrix** **4×4**
  - vector of texture coordinates: **[s, t, r, q]**
  - **q** plays the role of a homogeneous component here (… projective transformation)
- **OpenGL**: another <u>transform matrix stack</u>
  - GL_TEXTURE mode
  - specific settings for each texture unit
- modern solution: **vertex shaders**

Texture combination

$\otimes$



- modern GPUs (since TNT) can **combine more textures** in one fragment (“**multitexturing**”)
  - global (low-frequency) **basis** + **detail** texture
  - pre-computed **lighting** (“light-map”)
  - ***“environment maps”*** – reflection of a surround scene
  - …

Texture combination

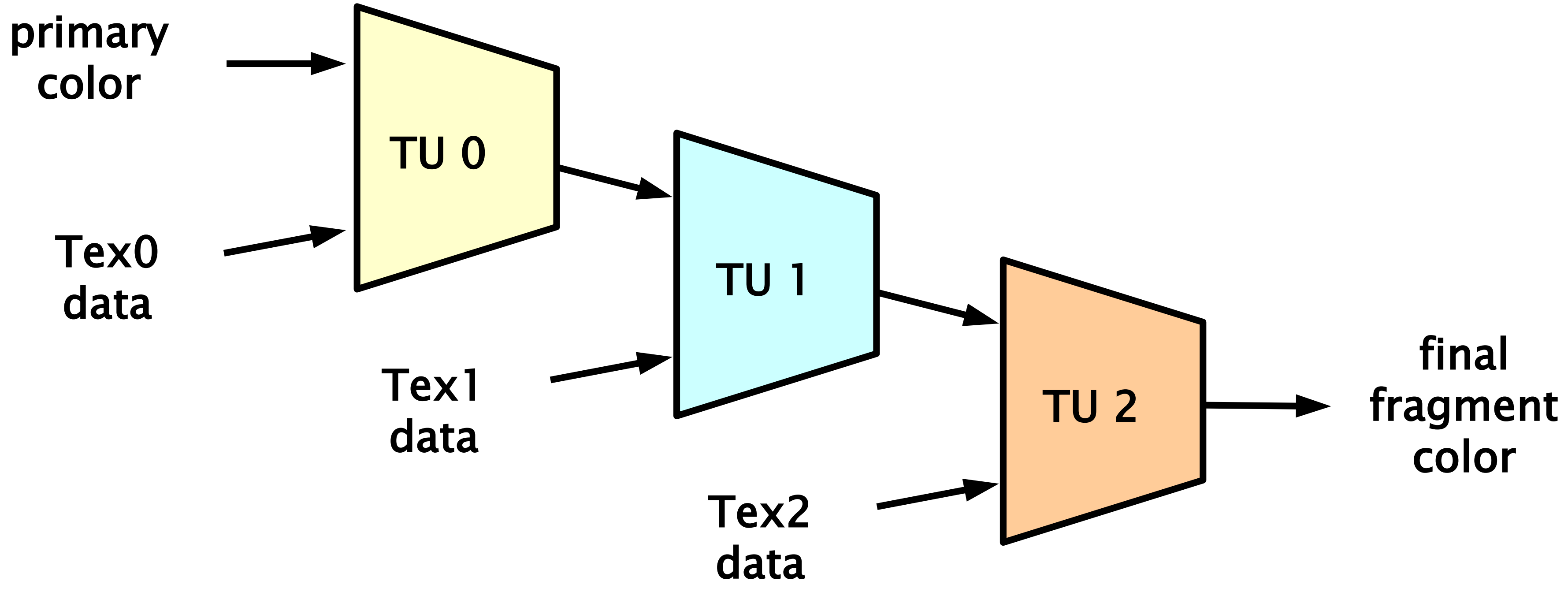
- common combination operators
  - REPLACE (source is ignored)
  - MODULATE (multiplication – values are abated)
  - DECAL (semi-transparent texture on an original surface)
  - INTERPOLATE (lerp, 2 sources)
  - DOT3_RGB[A] (inner product, for 3D)
  - ADD, ADD_SIGNED, SUBTRACT, ..
- **programmable GPU** (in “fragment shader”): arbitrary formula

Texture units (TU)

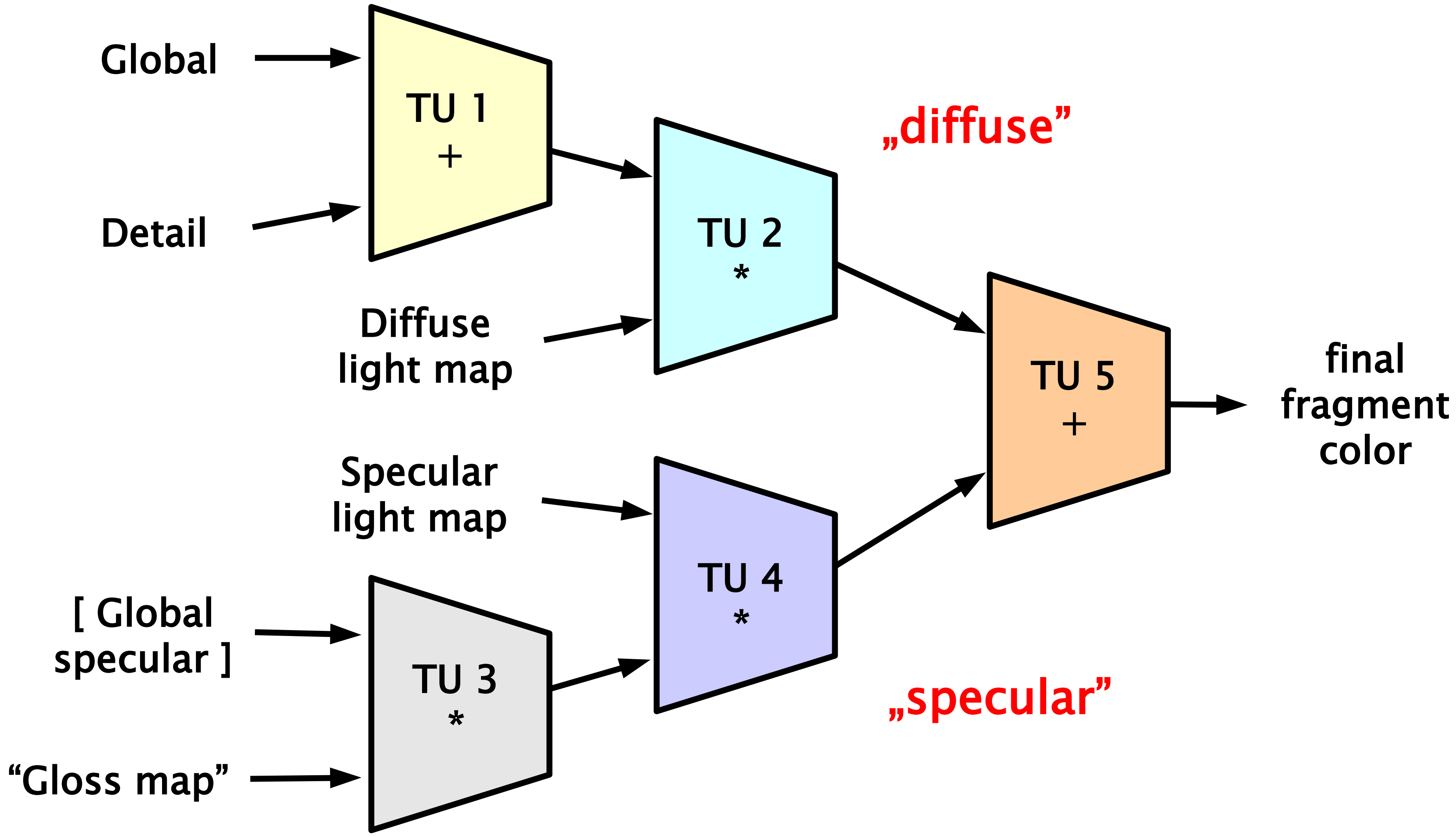
- one **texture unit** (**TU**) handles one bitmap source
  - **value fetch** (including *filtering*, see later)
  - compute required **combination operator**
  - **result** is stored in an output fragment
- **inputs** (arguments) of a combination operator
  - source color (output of a preceding TU / primary color)
  - texture data (read/filtered texel)
  - constant color value (see GL_TEXTURE_ENV_COLOR)
  - "primary color" = pre-texture color = interpolated value from a rasterizer

Texture cascade

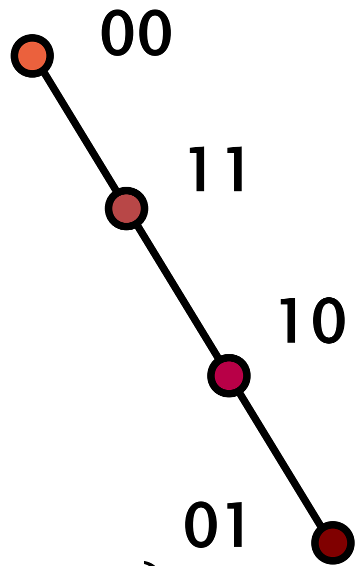
- linear TU chain:



More complex example



Texture compression – S3TC



- invention: S3 in DirectX 6 (1998)
  - DXTC, in OpenGL: S3TC, DXT1
- **fixed compression ratio**
  - necessary for memory management
  - 4:1 to 6:1 **lossy compression**
- decomposition into rectangle **"tiles"** (4×4 px)
  - each tile: **two 16-bit colors** and **sixteen 2-bit indices** (together – 4 bpp)
    - two extreme colors (R5G6B5), two more in-between colors (or one in-between and black)
    - each pixel is represented by a reference to one color

Texture compression, advanced

- NVIDIA **VTC** (Volume Texture Compression)
  - 3D variant of S3TC
  - data blocks 4×4×1, 4×4×2, 4×4×4

- 3DFX compression **FXT1** (1999)
  - **8×4** texels stored in **128 bits** (also 4 bpp)
  - **4 different block formats** (encoder decides)
    - CC_HI: 2× R5G5B5, 32× 3-bit code (5 betw., 1 transpar.)
    - CC_CHROMA: 4× R5G5B5, 32× 2-bit code (orig. colors)
    - CC_MIXED: 4× R5G5B5, 32× 2-bit code (2 for each 4×4 rectangle, complicated sub-formats)
    - CC_ALPHA: 3× R5G5B5A5, 32× 2-bit code (lerp/nlerp)

Advanced texturing techniques

Most important advanced approaches:

- ***"MIP-mapping"*** and **anisotropic filtering**
- ***"gloss mapping"*** (glossy reflection)
- ***"light mapping"*** (alt: "*dark mapping*") – lighting
- ***"shadow mapping"*** – pre-computed shadow
- ***"bump mapping"*** (normal-vector modulation)
- ***"environment mapping"*** (environment reflection)
- **multi-pass** processing, **"multi-texturing"**

- **GPU programming not needed** (in most cases)

Texture filtering

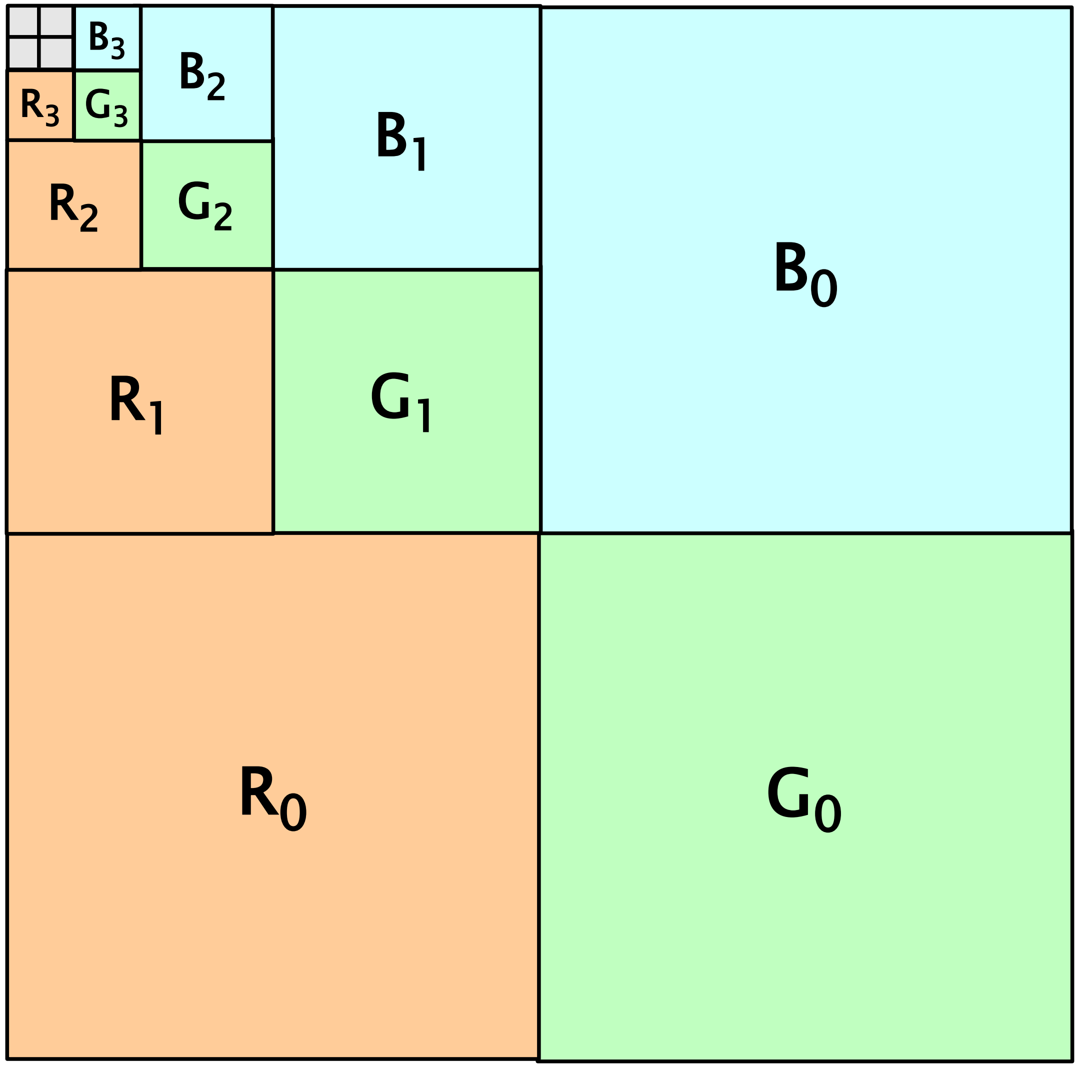
- texture "seen from a distance" should be **filtered** (~raster image sub-sampling)
  - otherwise "***alias***" will appear (especially disturbing in motion)

- **pre-processing techniques**
  - "**MIP-map**" ("multum in parvo"), most popular (HW)
  - "**ripmap**" (Hewlett-Packard), anisotropic miniatures
  - **anisotropic filtering** – dynamic method, usually MIP-map + number of linear samples
  - **summary tables** – pre-computed UL rectangle sums

MIP-mapping

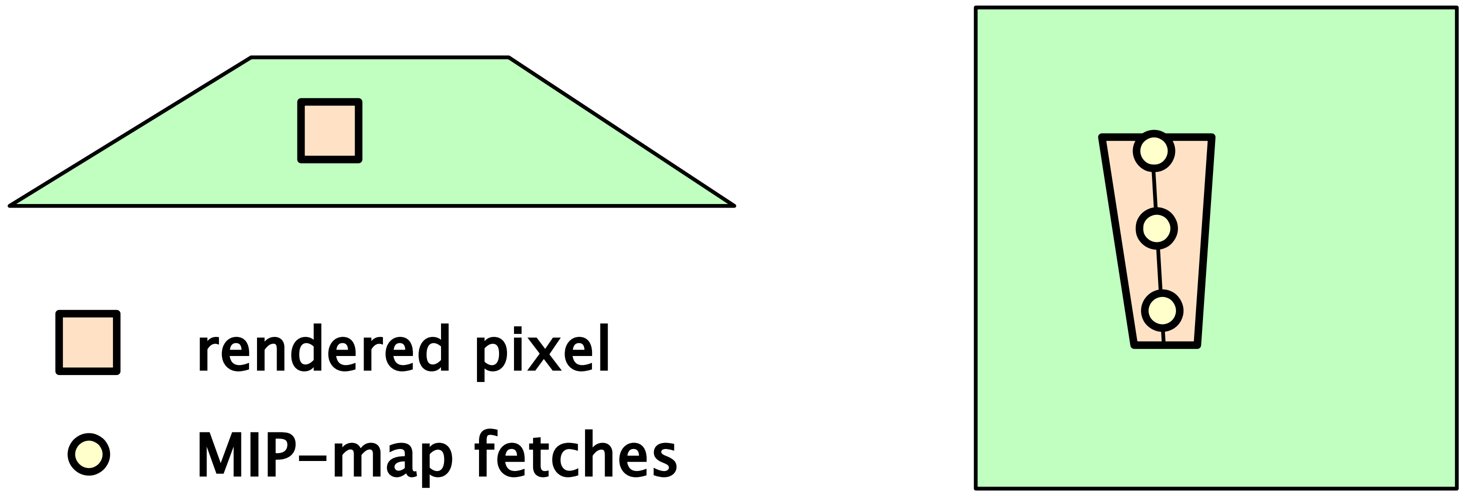
- texture subsampling in advance – **binary fractional resolutions** (1/4, 1/16, etc. – HW supported)
  - high quality sub-sampling with averaging
  - **3-component color** (RGB) – convenient arrangement in memory
- **MIP-map utilization**
  - compute **level** (according to required texture scaling)
  - either **single texel fetch** (speed)
  - or **interpolation** between two adjacent **MIP-map levels** or even bi-linear interpolation in the levels (at most 8 fetches = quality)

MIP-map



Anisotropic filtering

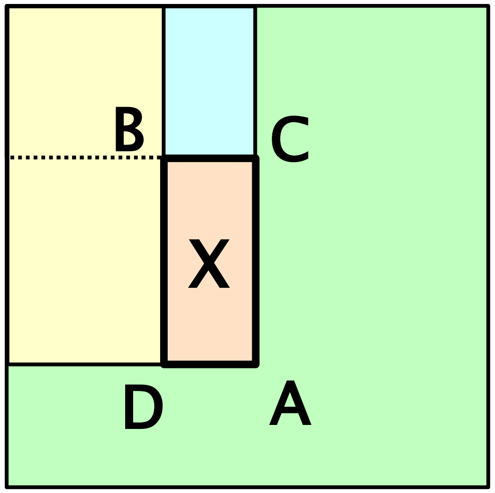
- back-projected **screen pixel** = deformed quadrangle
  - **MIP-map level** according the higher sub-sampling (shorter size)
  - **multi-sampling** (averaging) along the longer side



Summary tables

X = A + B – C – D

A, B, C, D ... upper left sums



(pre–computing uses dynamic programming)

- **arbitrary-rectangle sum** (average)
  - pre-computed summary table
  - higher precision is needed(minimal 24bpp instead of 8bpp)

Texture effects

- direct **color modification** (modulation)
  - composition (smooth/detail texture, additive noise, dirt)
  - see “Multi-texturing“, **RGB** and **α**
- **“decal”** effect
  - semi-transparent decal apllied on the object surface
  - “billboards“, “imposters” (see)
- **bumpy surface** (**“bump mapping”**)
  - normal-vector modulation
- **material-property** modification
  - reflectance, gloss, ..

Texture effects II

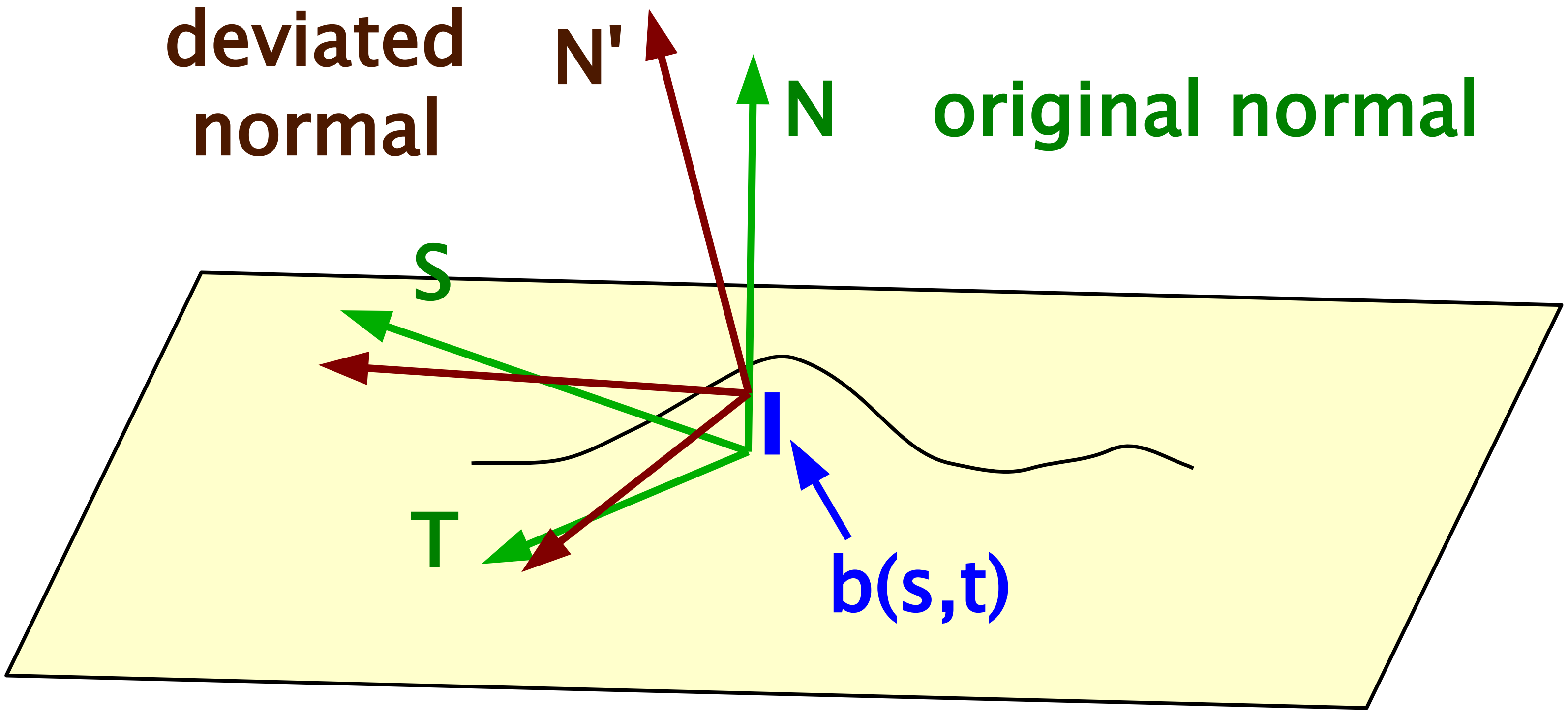
- **“light mapping“**, **“shadow mapping”**
  - pre-computed light or shadow
  - surface reflectance is modulated by the map
- **“environment mapping”** (EM)
  - simulation of **ideal mirror surface**, surrounding scene is reflected by it
  - GPU capable of reflected ray computation and apropriate texture addressing (e.g. “cube mapping”)
  - EM can be further used for:
    - specular reflection „S”
    - “bump-mapping“ + EM combination

Bump-mapping

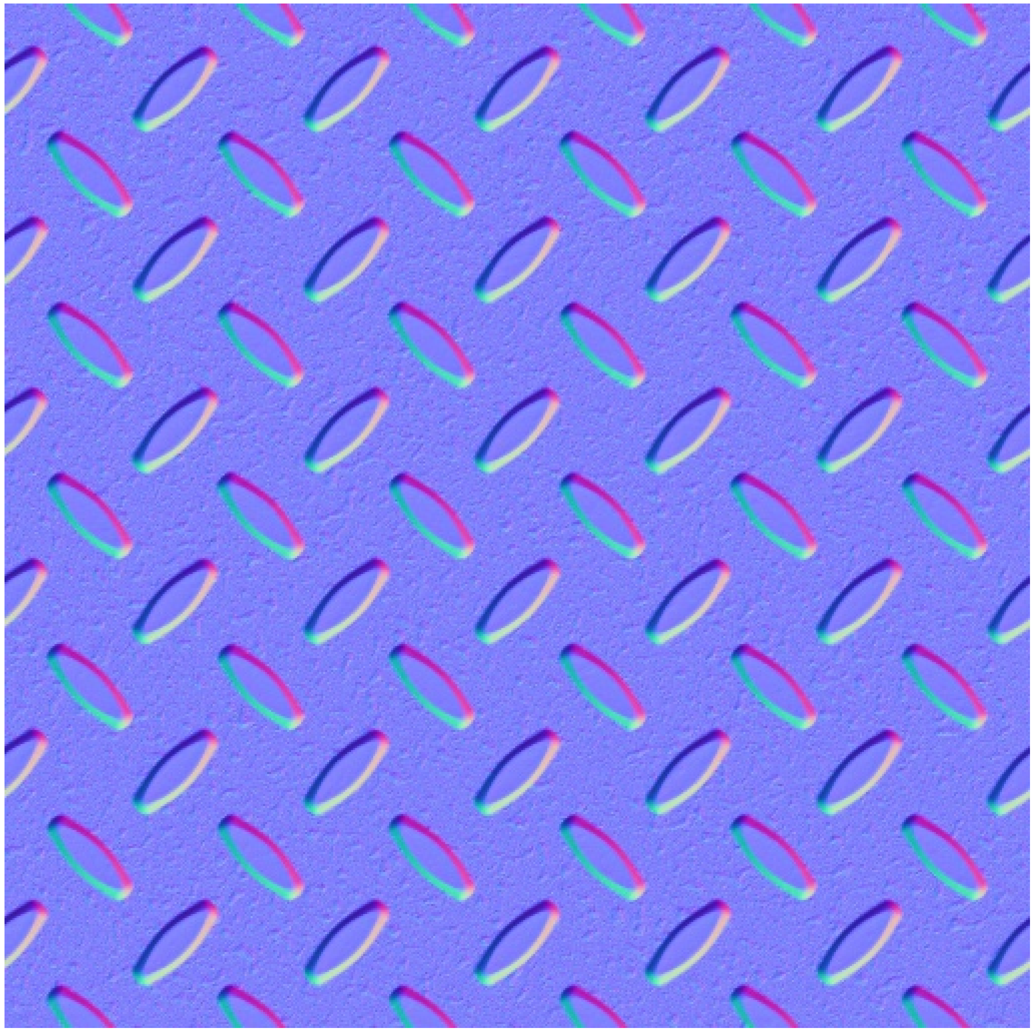
- special texturing technique – impression of a **bumpy surface**
  - replaces complicated macro-geometry
  - modifies (modulates) **normal vector** in every pixel
  - human observer thinks that a surface is actually bumpy (much of the impression is inferred from specular reflec-tions)
- HW implementation
  - "normal-mapping", multi-texturing
  - Phong shading (normal interpolation) is recommended

Scheme

- **input:** difference function **b(s,t)** defines **oriented distance** of the actual surface from the ideal (smooth) one



Normal map example

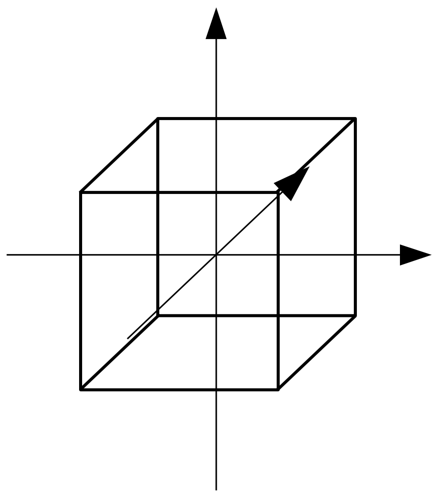


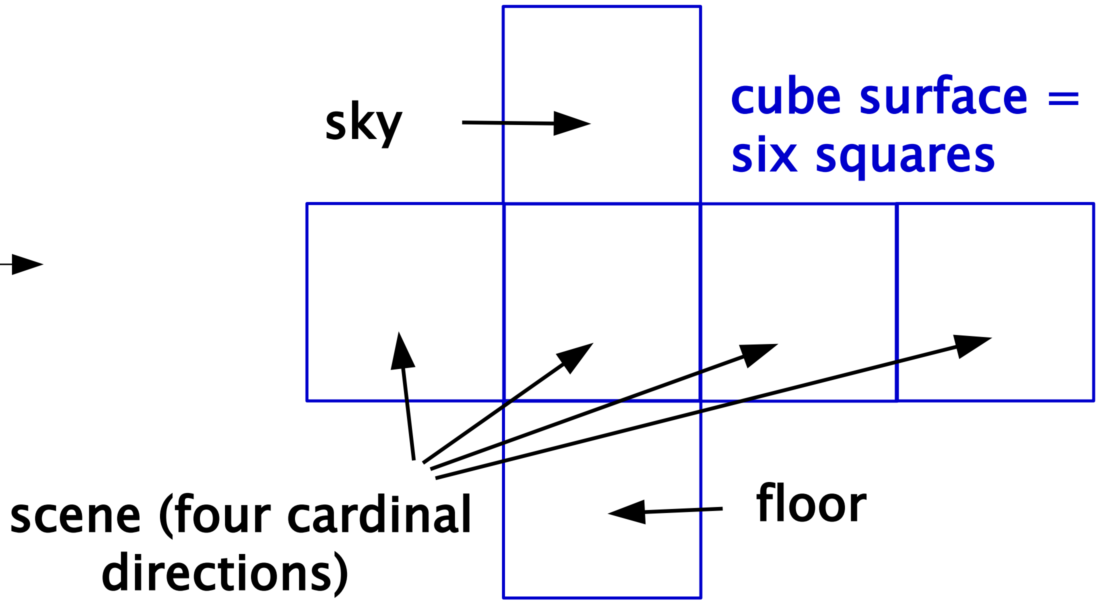
More texturing techniques

- **texture domain** (addressing)
  - **3D coordinate [x, y, z] – *"volume texture"***
    - inner object structure (wood, marble, ...)
  - **normal vector N**
    - normal-dependent quantity (e.g. static diffuse shading)
  - **reflection vector R – "*environment mapping*"**
    - pre-computed surround images (provided by CPU or GPU) stored in special texture type ("cube map")
    - **surround scene** is reflected on a shiny object surface
    - **light map** (pre-computed multi-source light, areal source lighting, advanced reflectance models, etc.)

Environment mapping (EM)

- reflection vector **R** converted to
  - **spherical coordinates** – more complicated
  - six cube faces – **“cube mapping”**



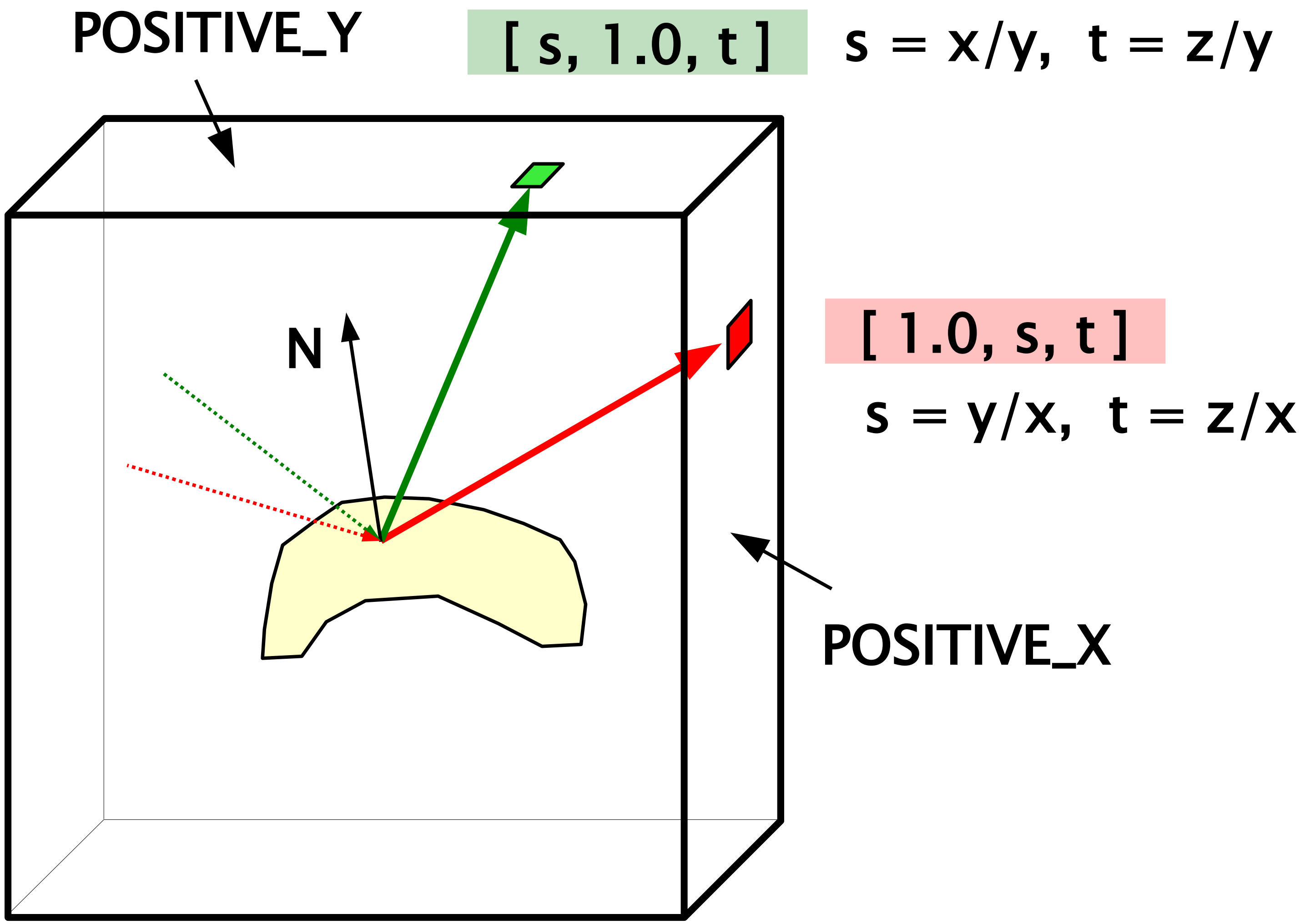


Cube mapping

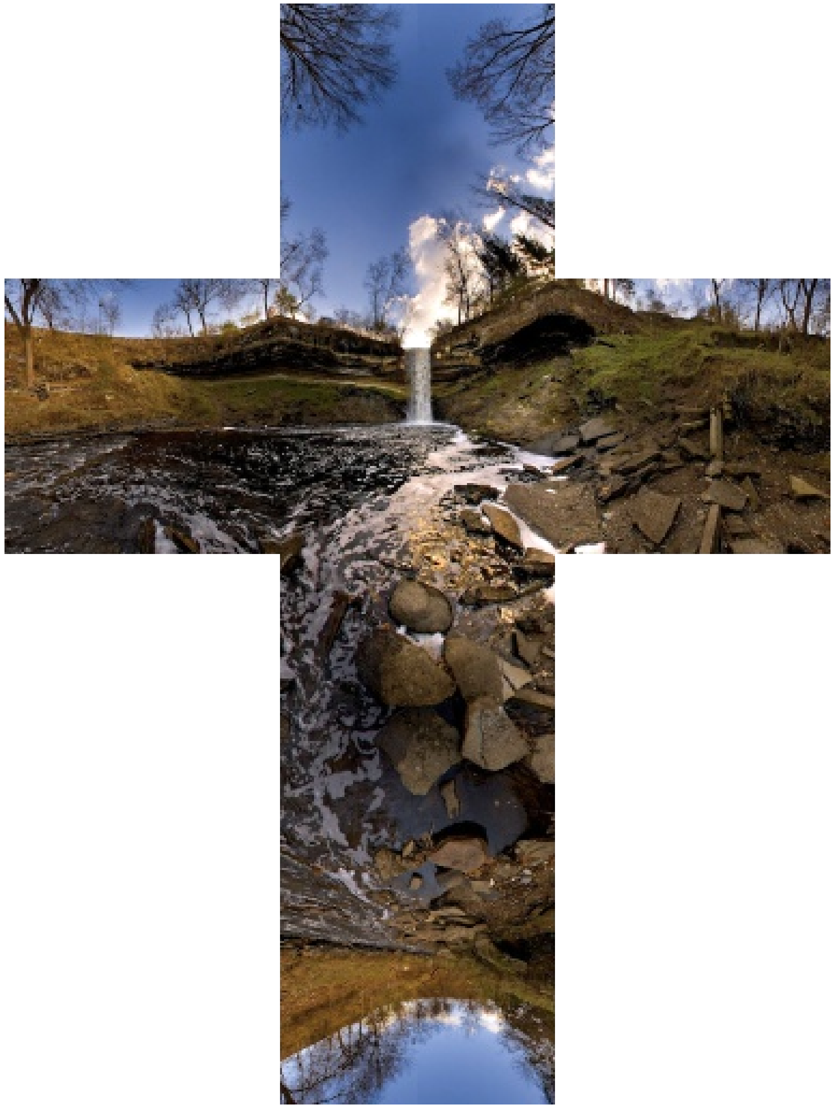
- **Greene**, 1986
  - simpler than previously used spherical mapping
- cube-map texture consists of **6 square bitmaps**
  - POSITIVE_X, NEGATIVE_X, POSITIVE_Y, …
  - easy data acquisition – e.g. GPU rendering in real-time
  - easy **bitmal adressing**, no vector normalization needed, only a division

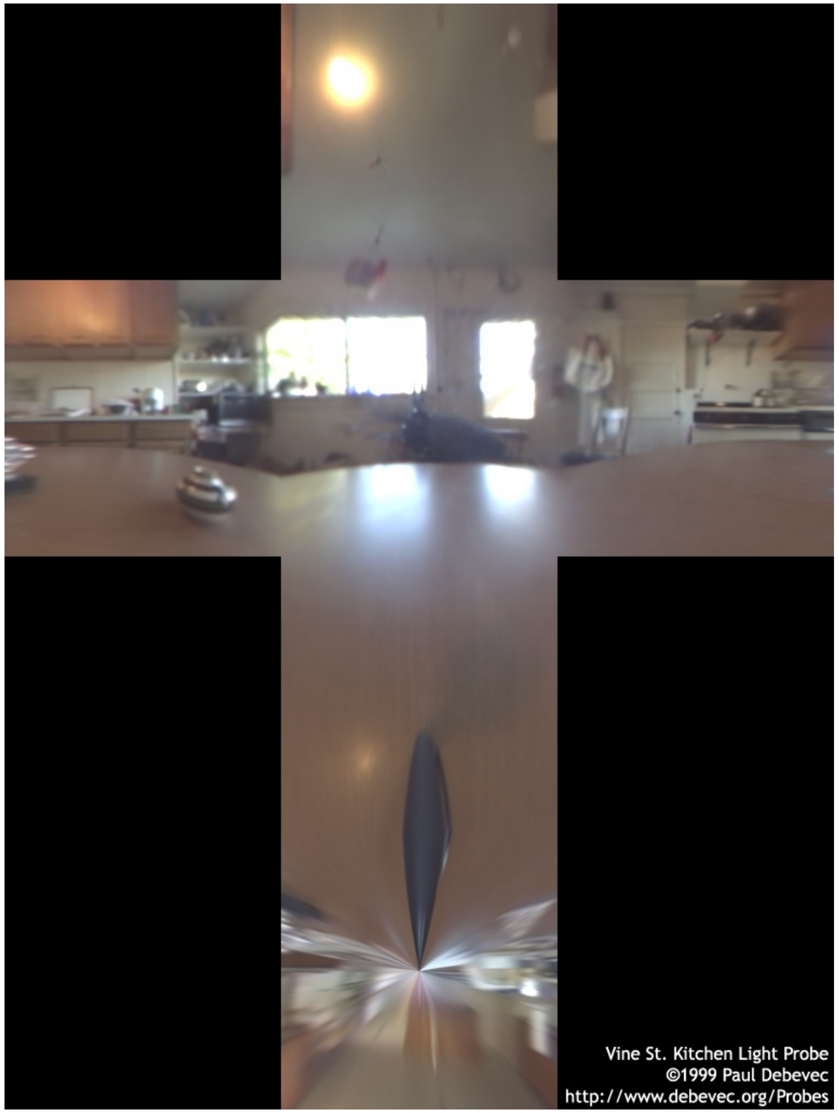
    1. select max-value component ⇒ face
    2. compute 2D coordinates (two divisions)

Cube mapping



Cube mapping examples

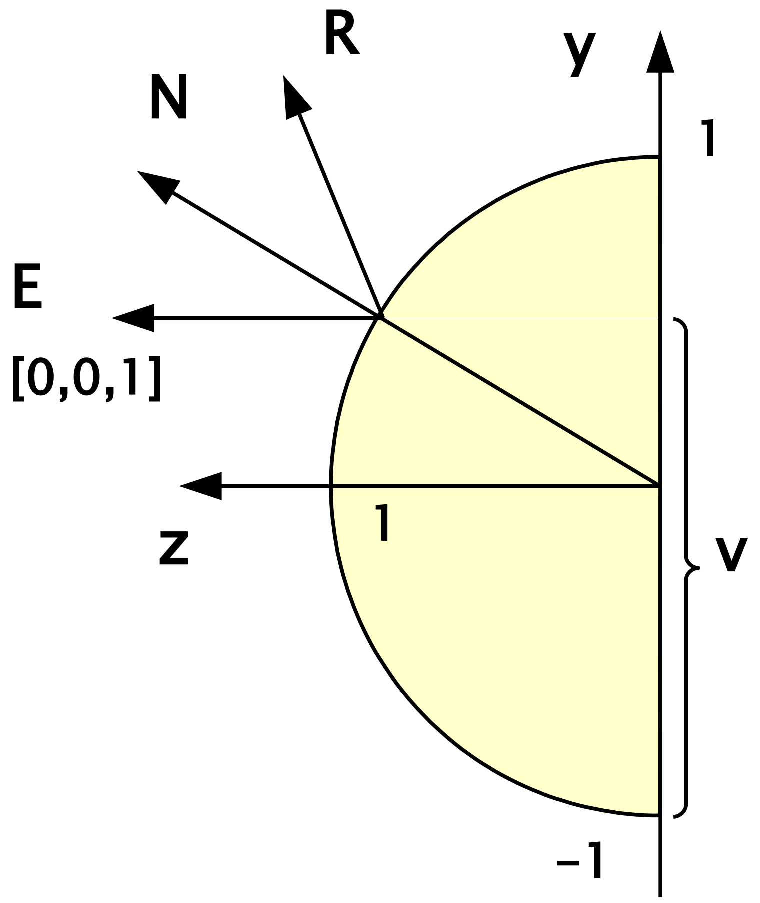




More directional mappings (advanced)

- **sphere mapping** (Miller, Hoffman: 1984)
  - direct implementation of reflection on **ideal sphere** ("light probe")
  - only <u>one square texture</u> needed, but distortions near sphere contour
- **paraboloid mapping** (Heidrich, Seidel: 1998)
  - <u>two photographs</u> from opposite directions
  - no borderline distortion, memory efficient
  - more uniform sampling of the direction space
  - implementation: <u>perspective projection</u> of texture coords.

Sphere mapping



$$R = E - 2\,(N \cdot E)\,N$$

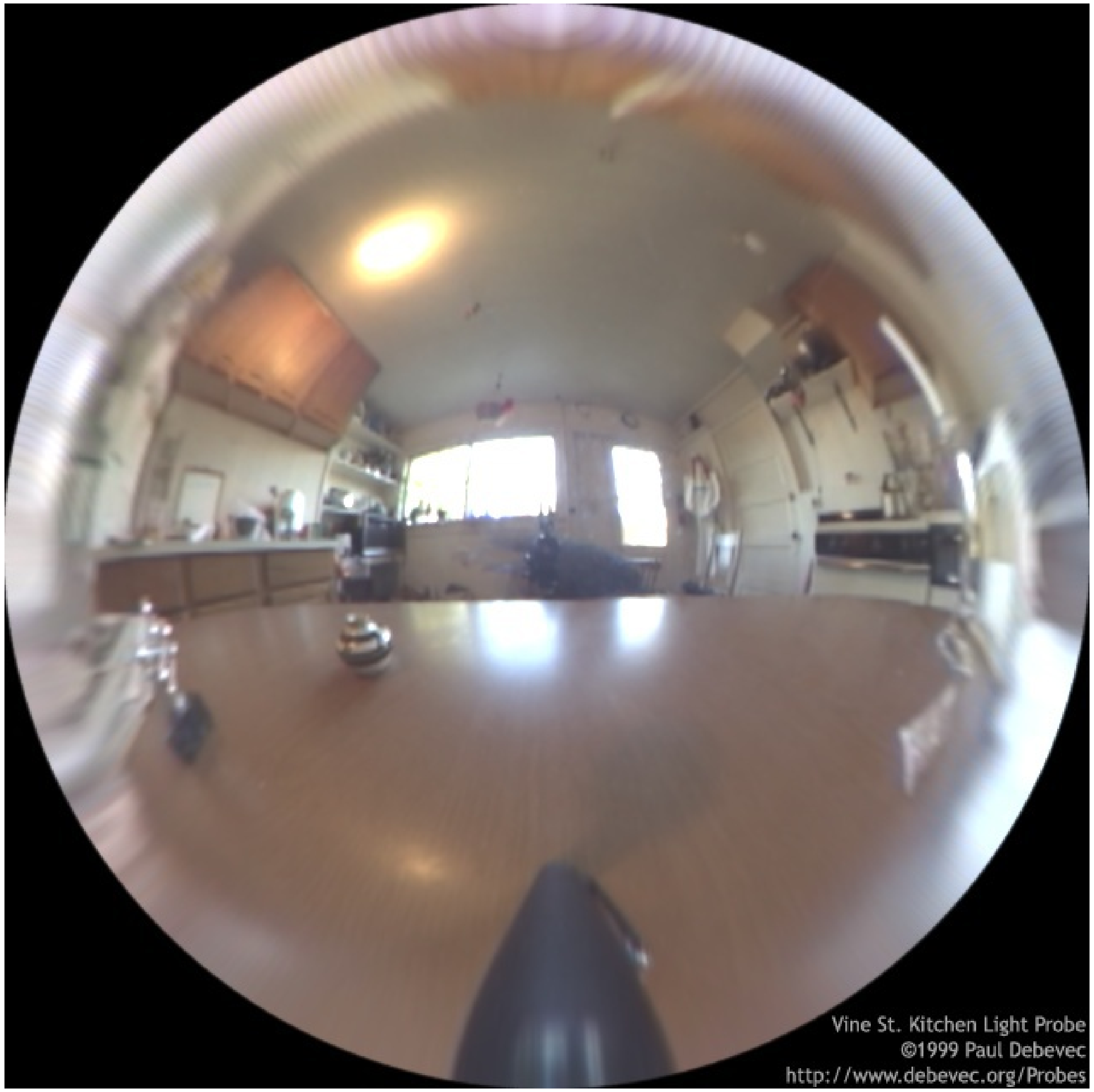
$$m = \sqrt{R_x^2 + R_y^2 + (R_z + 1)^2}$$

$$u = \frac{R_x}{2m} + 0.5$$

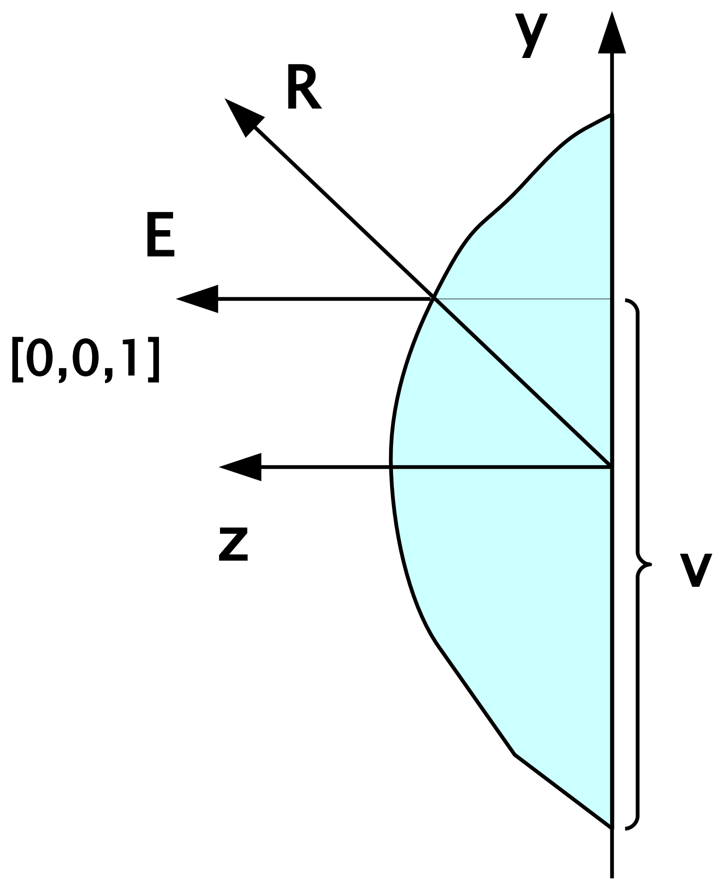
$$v = \frac{R_y}{2m} + 0.5$$

Projective transformation needed!

Light-probe example



Paraboloid mapping



$$R = E - 2\,(N \cdot E)\,N$$

$$u = \frac{R_x}{2\,(1 + R_z)} + 0.5$$

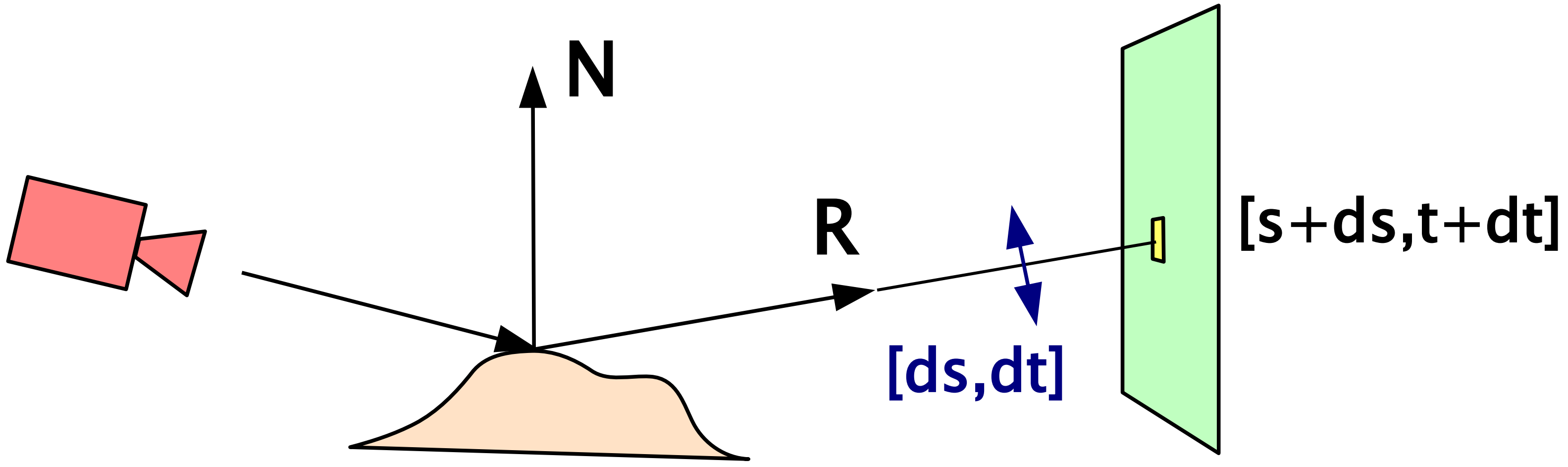
$$v = \frac{R_y}{2\,(1 + R_z)} + 0.5$$

Projective transformation needed!

Environment-map + bump-map

- GPUs are capable of spherical & paraboloid mapping

- “**environment-mapping**” can be effectively com-bined with „**bump-mapping**”
  - instead of normal modulation we modulate **[s, t]** for EM



Sources

- Tomas Akenine-Möller, Eric Haines: ***Real-time rendering, 2nd edition***, A K Peters, 2002, ISBN: 1568811829
- OpenGL ARB: ***OpenGL Programming Guide, 4th edition***, Addison Wesley, 2004, ISBN: 0321173481
- J. Žára, B. Beneš, J. Sochor, P. Felkel: ***Moderní počítačová grafika***, 2nd edition, Computer Press, 2005, ISBN: 8025104540