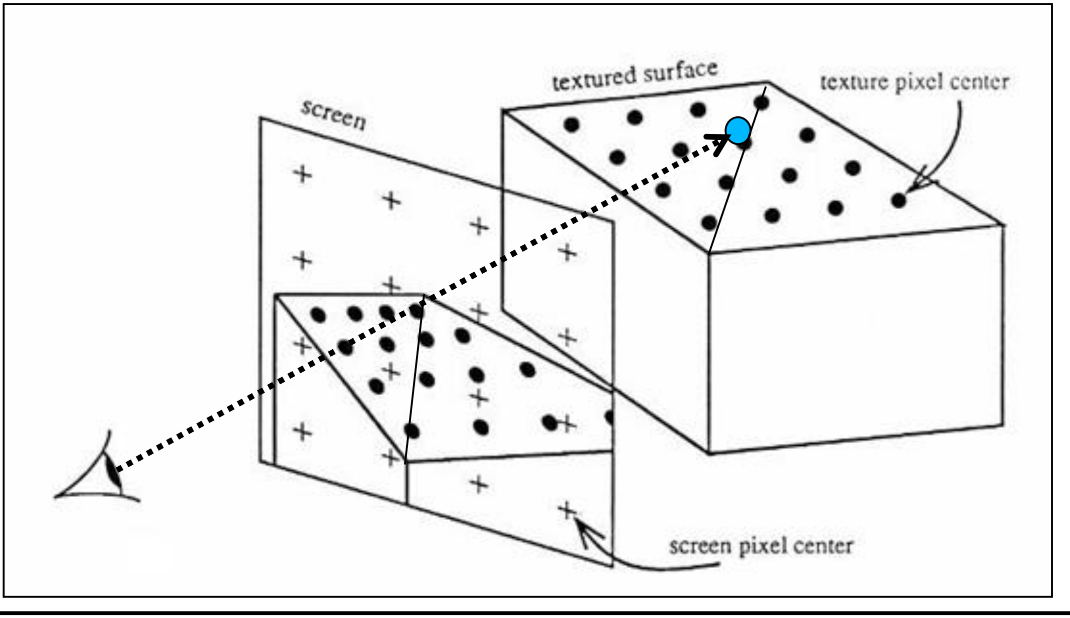
Computer Graphics

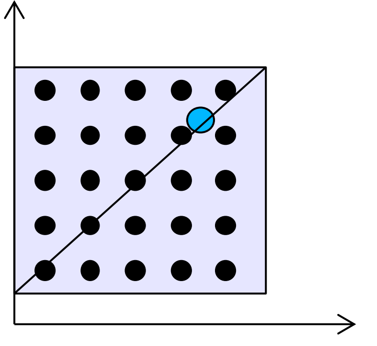
Texture Filtering

Philipp Slusallek

Reconstruction Filter

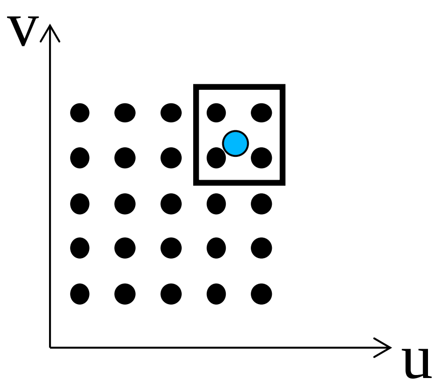
- **Simple texture mapping in a ray-tracer**
  - Ray hits surface, e.g. a triangle
  - Each triangle vertex also has an arbitrary texture coordinate
    - Map this vertex into 2D texture space (aka. texture parameterization)
  - Use barycentric coordinates to map hit point into texture space
    - Hit point generally does not exactly hit a texture sample
    - Use reconstruction filter to find color for hit point



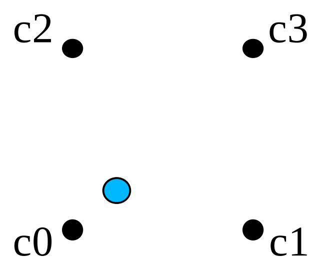


Texture Space

Nearest Neighbor "Interpolation"

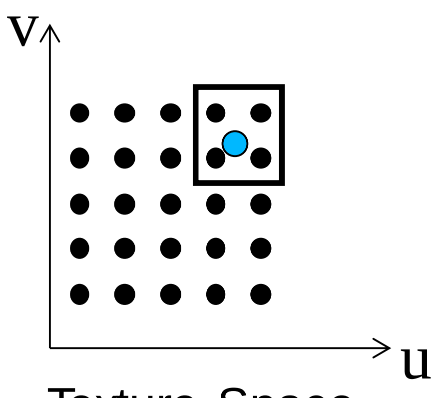


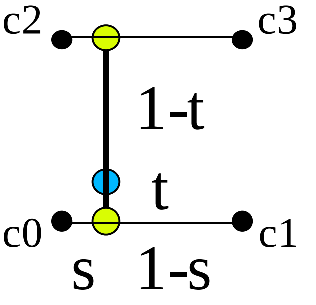
Texture Space



- **How to compute the color of the pixel?**
  - Choose the closest texture sample
    - Rounding of the texture coordinate in texture space
    - c = tex[min($\lfloor$ u * resU $\rfloor$, resU – 1) ,
      min($\lfloor$ v * resV $\rfloor$, resV – 1)];

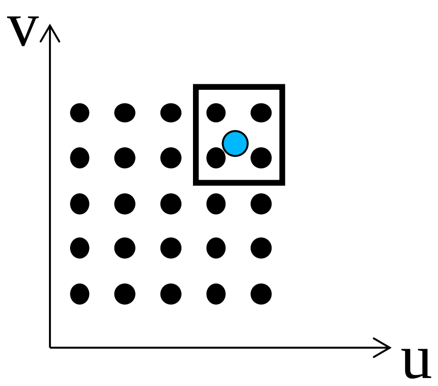
Bilinear Interpolation



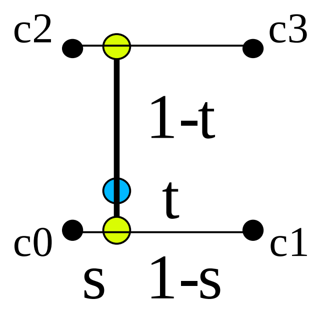


- **How to compute the color of the pixel?**
  - Interpolate between surrounding four pixels
  - $c = (1-t)\,(1-s)\,c0 + (1-t)\,s\,c1 + t\,(1-s)\,c2 + t\,s\,c3$

Bilinear Interpolation



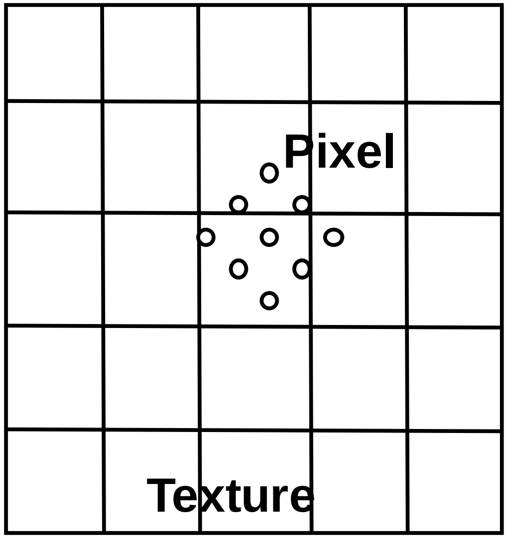
Texture Space

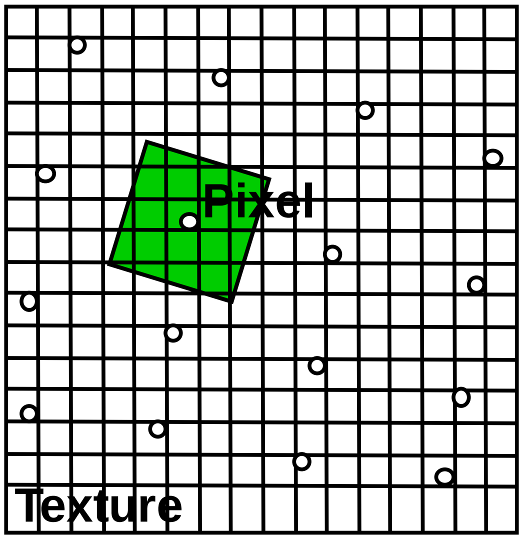


- **Can be done in two steps:**
  - $c = (1-t)\ (\ (1-s)\ c0 + s\ c1\ ) + t\ (\ (1-s)\ c2 + s\ c3\ )$
  - Horizontally: twice between left and right samples using fractional part of the texture coordinate (*1-s*, *s*):
    - $i0 = (1-s)\ c0 + s\ c1$
    - $i1 = (1-s)\ c2 + s\ c3$
  - Vertically: between two intermediate results (*1-t*, *t*):
    - $c = (1-t)\ i0 + t\ i1$

Filtering

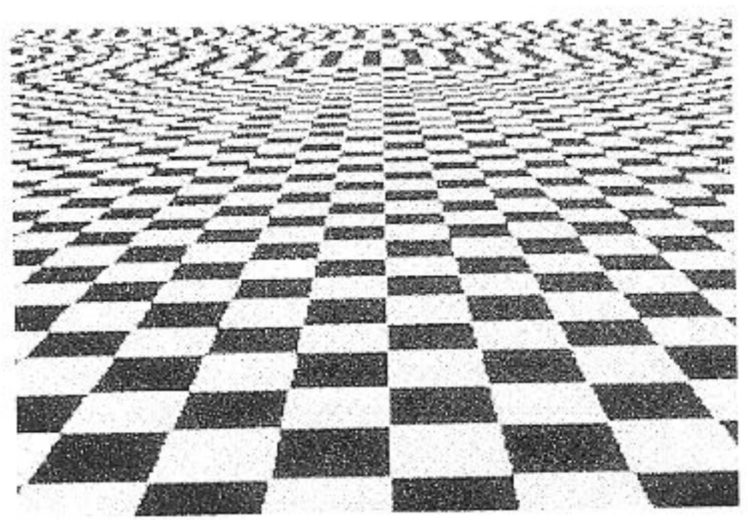
- **Magnification (Zoom-in)**
  - Map few texels onto many pixels
  - Reconstruction filter:
    - Nearest neighbor interpolation:
      - Take the nearest texel
    - Bilinear interpolation:
      - Interpolation between 4 nearest texels
      - Need fractional accuracy of coordinates
    - Higher order interpolation
- **Minification (Zoom-out)**
  - Map many texels to one pixel
    - Aliasing: Reconstructing high-frequency signals with low-frequency sampling
  - Antialising (low-pass filtering)
    - Averaging over (many) texels associated with the given pixel
    - Computationally expensive

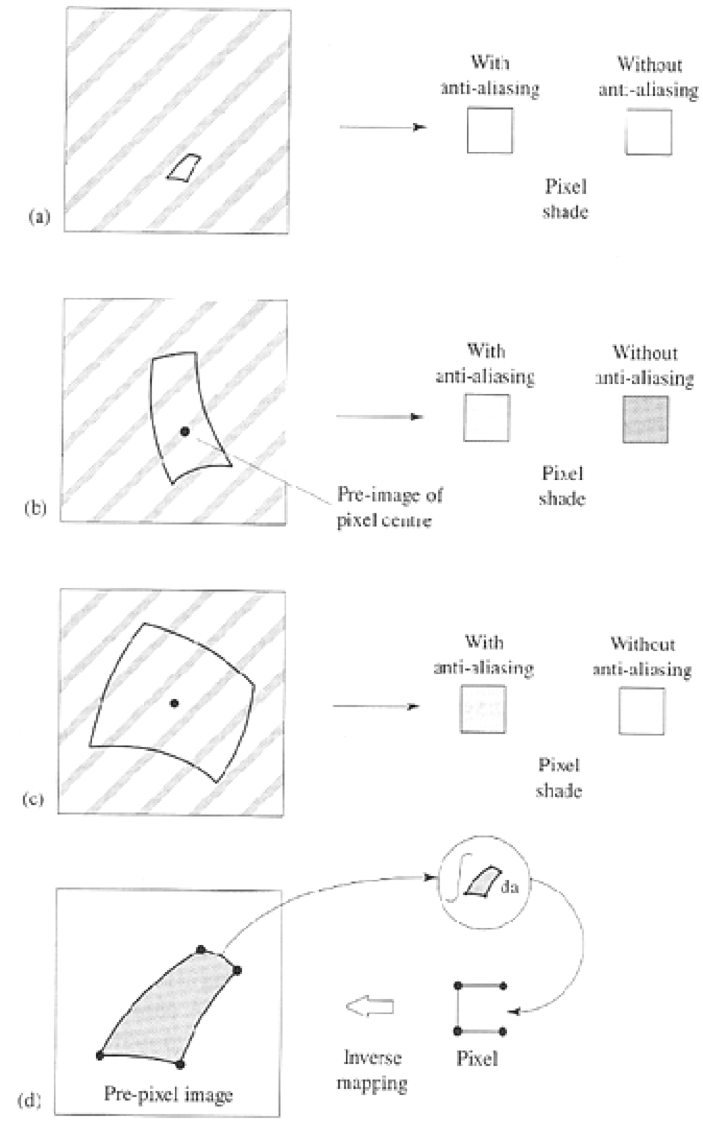




Aliasing Artifacts

- **Aliasing**
  - Texture insufficiently sampled
  - Incorrect pixel values
  - “Randomly” changing pixels when moving
- **Integration of Pre-Image**
  - Integration over pixel footprint in texture space





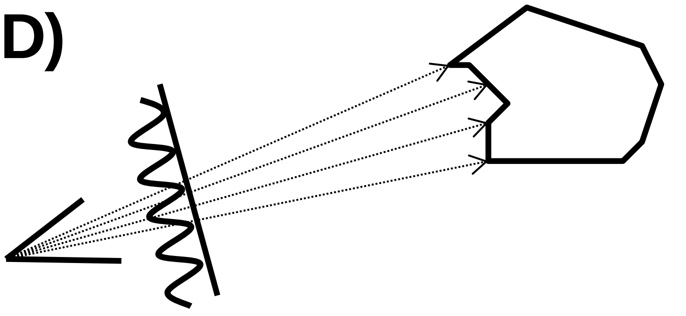
Sensors

- **Measurement of signal**
  - Conversion of a continuous signal to discrete samples by integrating over the sensor field
    - Weighted with some sensor sensitivity function *P*

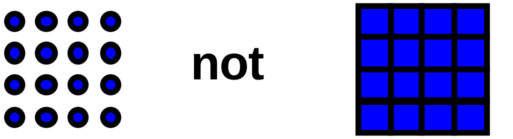
$$R(i,j) = \int_{A_{ij}} E(x,y)\, P_{ij}(x,y)\, dx dy$$

  - Similar to physical processes
    - Different sensitivity of sensor to photons
- **Examples**
  - Photo receptors in the retina
  - CCD or CMOS cells in a digital camera
- **Virtual cameras in computer graphics**
  - Analytic integration is expensive or even impossible
    - Needs to sample and integrate numerically
  - Ray tracing: mathematically ideal point samples
    - Origin of aliasing artifacts !

The Digital Dilemma



- **Nature: continuous signal (2D/3D/4D)**
  - Defined at every point
- **Acquisition: sampling**
  - Rays, pixels/texels, spectral values, frames, ... (aliasing !)
- **Representation: discrete data**
  - Discrete points, discretized values

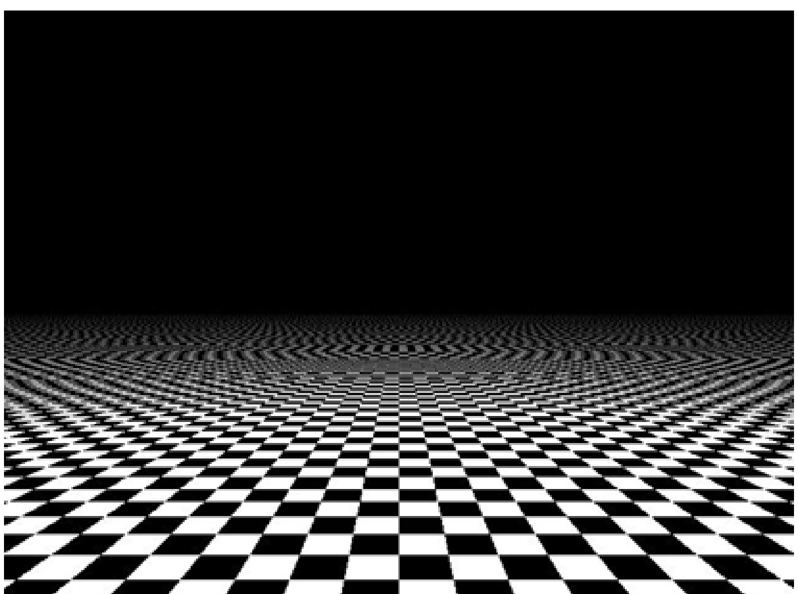


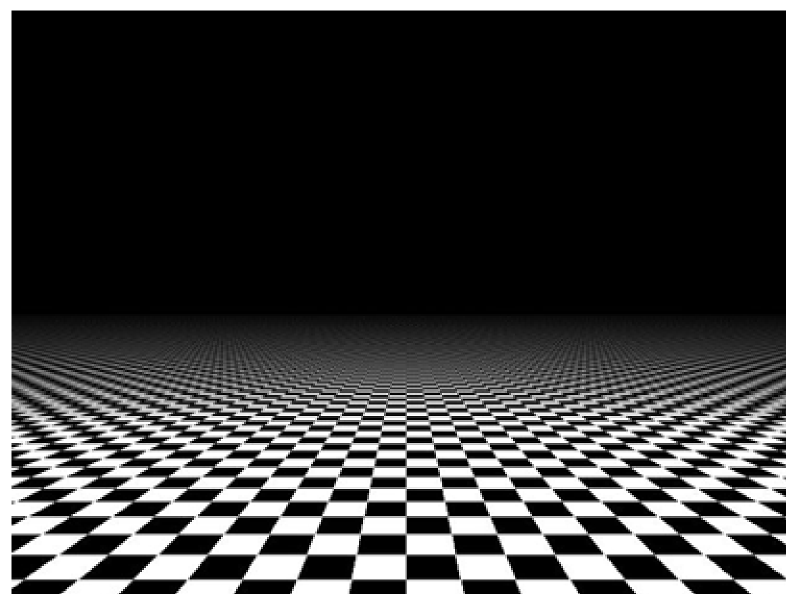
Pixels are usually point sampled

- **Reconstruction: filtering**
  - Recreate continuous signal
- **Display and perception (!)**
  - Hopefully similar to the original signal, no artifacts

Aliasing Example

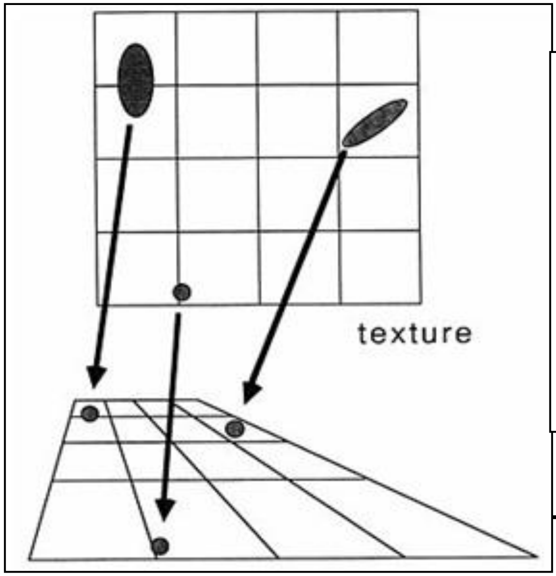
- **Ray tracing**
  - Textured plane with one ray for each pixel (say, at pixel center)
    - No texture filtering: equivalent to modeling with b/w tiles
  - Checkerboard period becomes smaller than two pixels
    - At the Nyquist sampling limit
  - Hits textured plane at only one point per pixel
    - Can be either black or white – essentially by “chance”
    - Can have correlations at certain locations

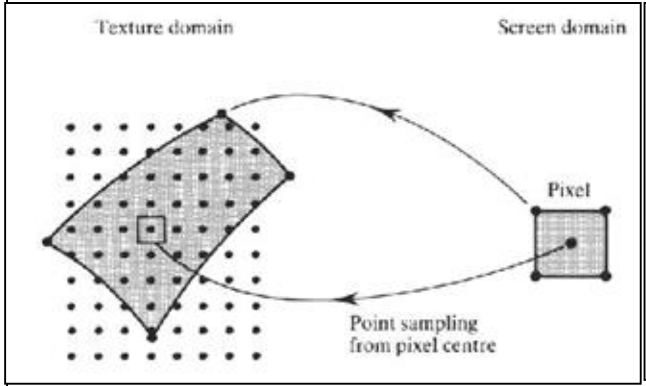


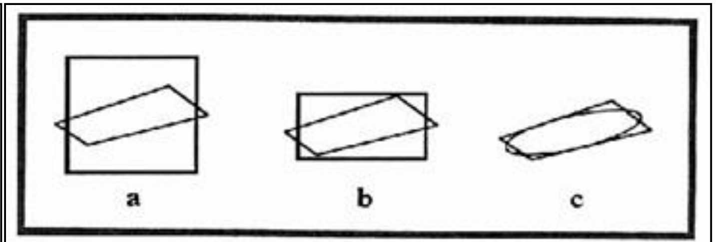


Pixel Pre-Image in Texture Space

- **Circular pixel footprints have elliptic pre-images on planar surfaces**
- **Square screen pixels form quadrilaterals**
  - On curved surface shape can be arbitrary (non-connected, etc…)
- **Possible approximation by quadrilateral or parallelogram**
  - Or taking multiple samples within a pixel







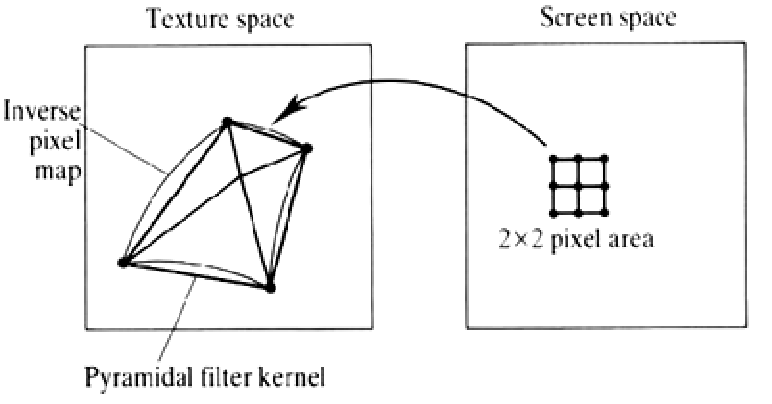
Approximating a quadrilateral texture area with (a) a square, (b) a rectangle, and (c) an ellipse. Too small an area causes aliasing; too large an area causes blurring.

Space-Variant Filtering

- **Space-variant filtering**
  - Mapping from texture space *(u,v)* to screen space *(x,y)* not affine
  - Filtering changes with position
- **Space-variant filtering methods**
  - Direct convolution
    - Numerically compute the integral
  - Pre-filtering
    - Precompute the integral for certain regions $\Rightarrow$ more efficient
    - Approximate actual footprint with precomputed regions

Direct Convolution

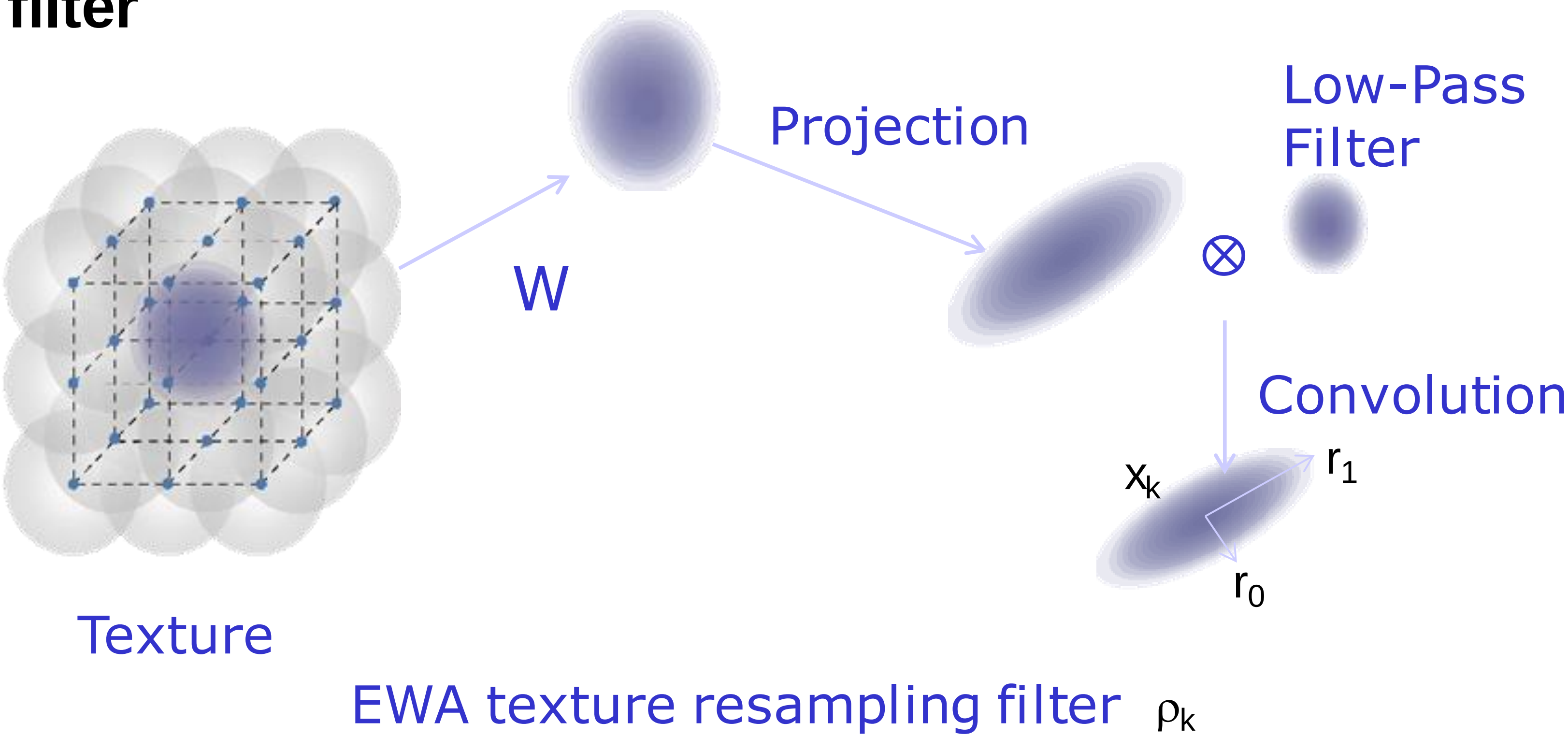
- **Convolution in texture space**
  - Texels weighted according to distance from pixel center (e.g. pyramidal filter kernel)
  - Essentially a low-pass filter



- **Convolution in image space**
  - Center the filter function on the pixel (in image space) and find its bounding rectangle.
  - Transform the rectangle to the texture space, where it is a quadrilateral whose sides are assumed to be straight.
  - Find a bounding rectangle for this quadrilateral.
  - Map all pixels inside the texture space rectangle to screen space.
  - Form a weighted average of the mapped texels (e.g. using a two-dimensional lookup table indexed by each sample's location within the pixel).

EWA Filtering

- **EWA: Elliptical Weighted Average**
- **Compensate aliasing artifacts caused by perspective projection**
- **EWA Filter = low-pass filter ⊗ warped reconstruction filter**

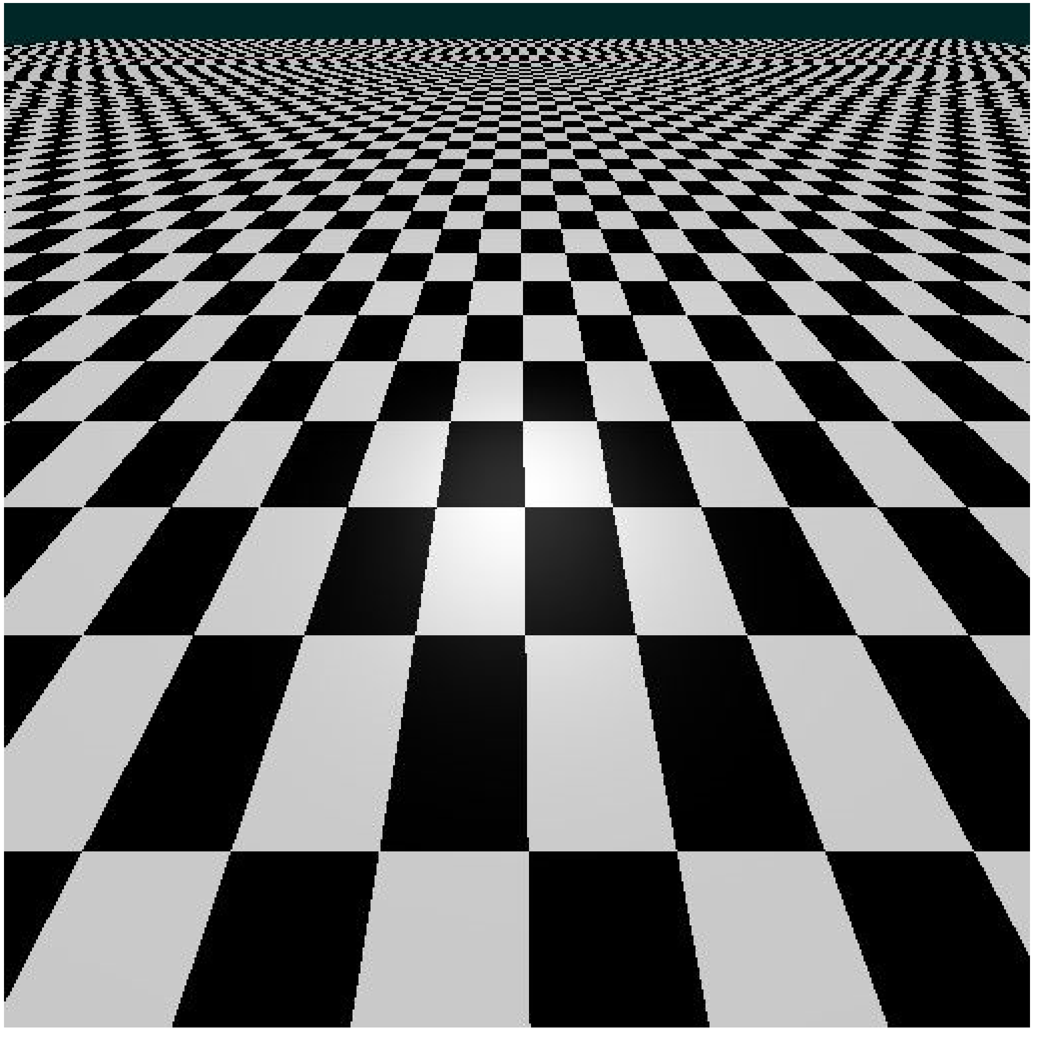


EWA Filtering

- **Four step algorithm:**
  1. Calculate the ellipse
  2. Choose low-pass filter
  3. Scan conversion in the ellipse
  4. Determine the color of the pixel

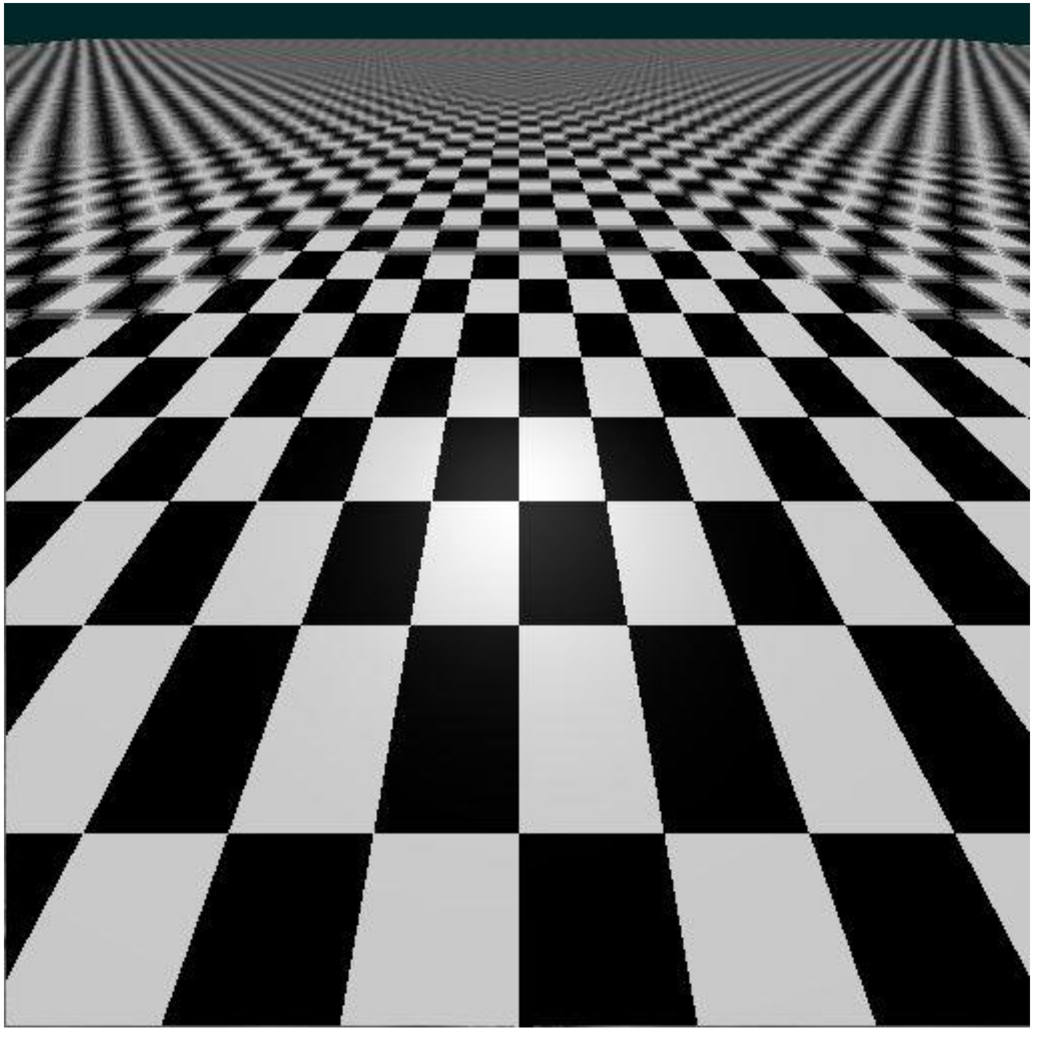
Without Anti-Aliasing

- **Checker board gets distorted**



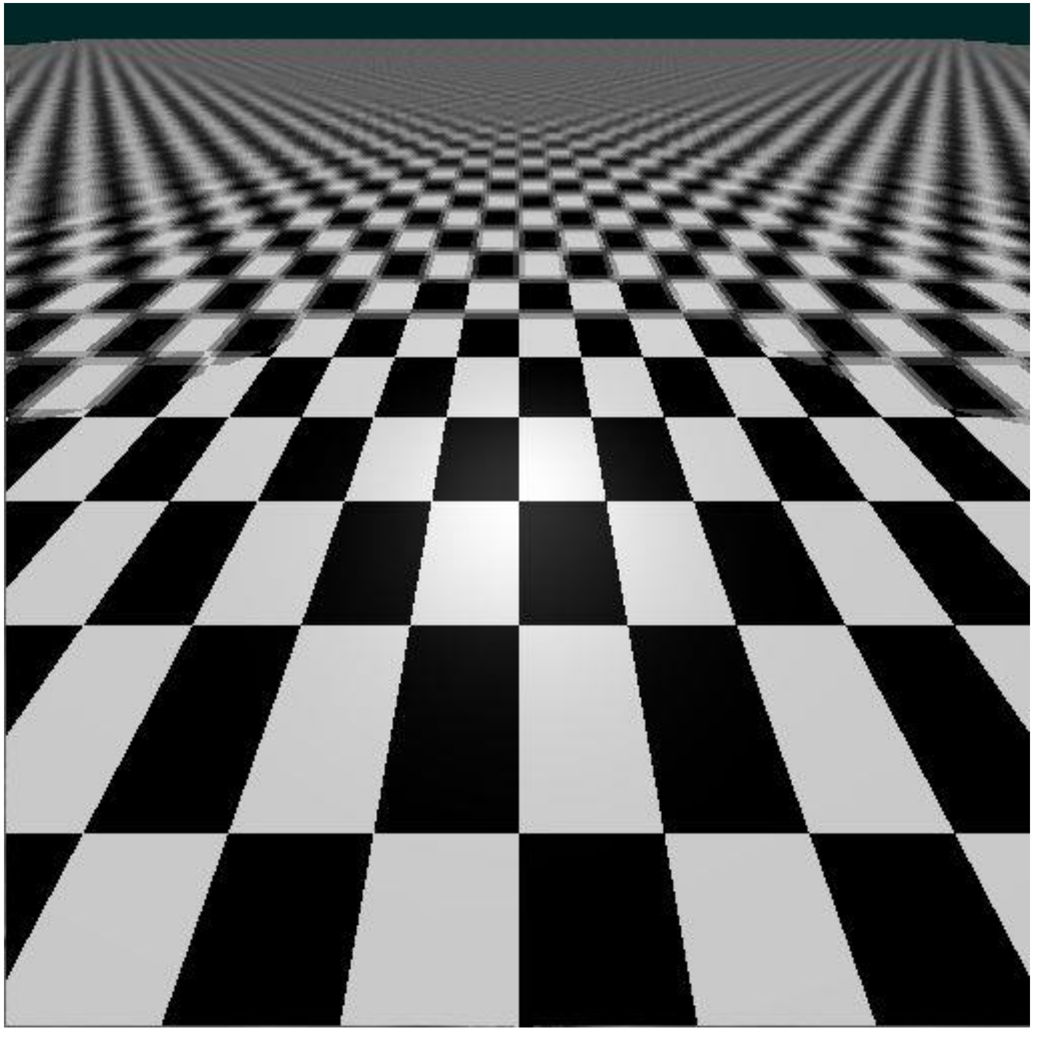
EWA Filtering

- **Elliptical filtering plus Gaussian**



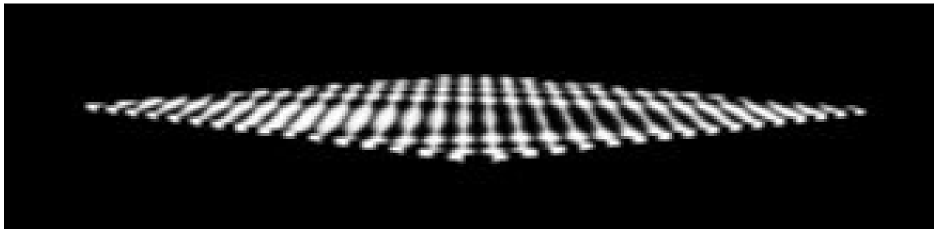
EWA Filtering

- **Gaussian blur selected too large ⇒ blurry image**

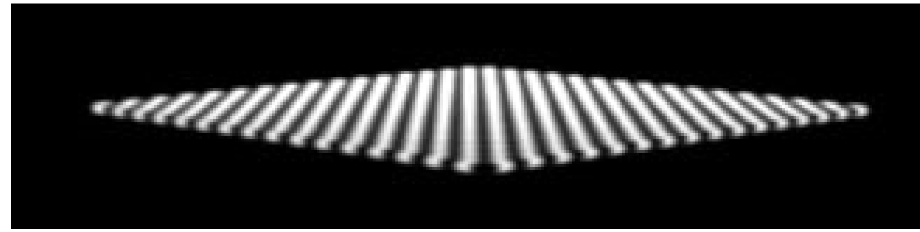


EWA Splatting

Zoom-out

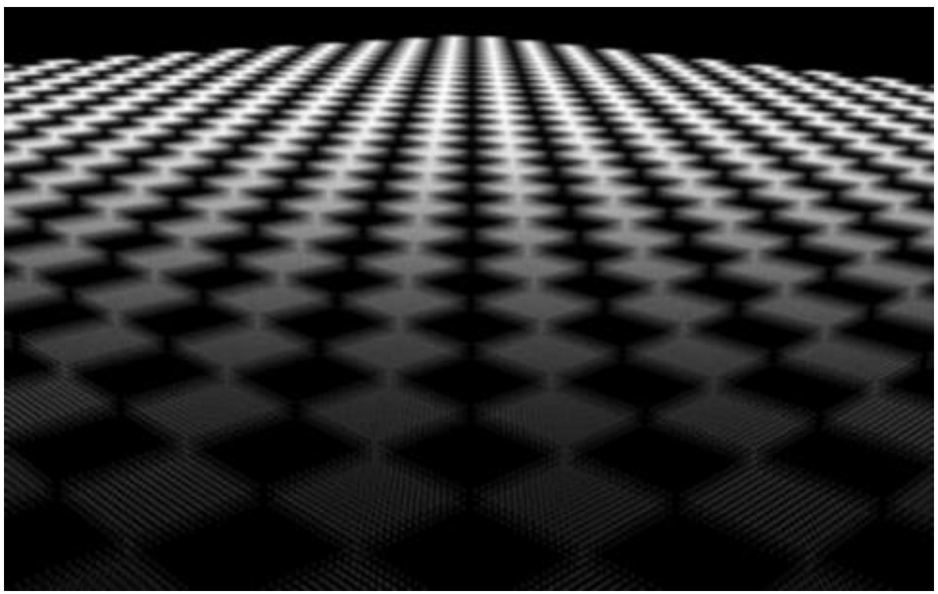


Reconstruction filter only

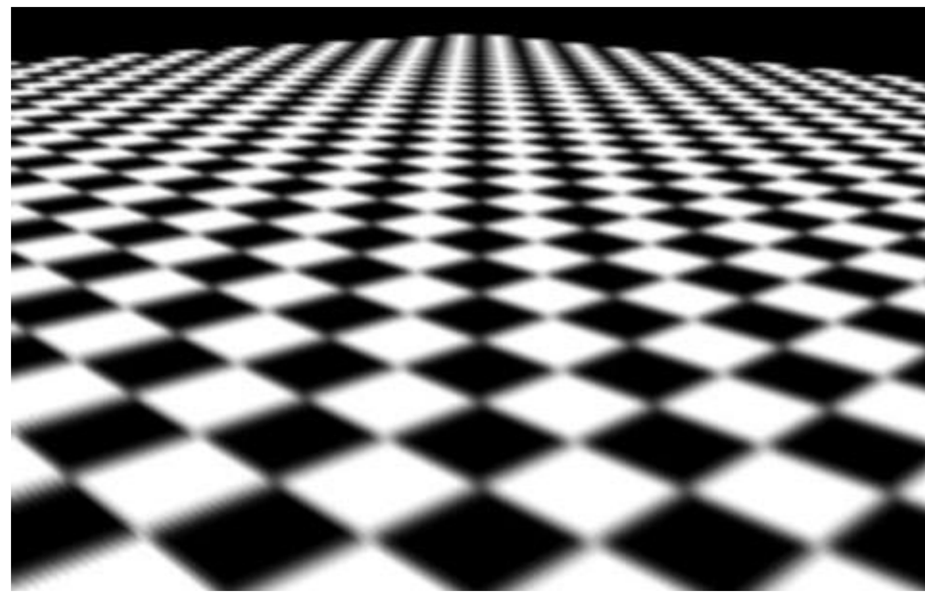


EWA filter

Zoom-in



Low-pass filter only



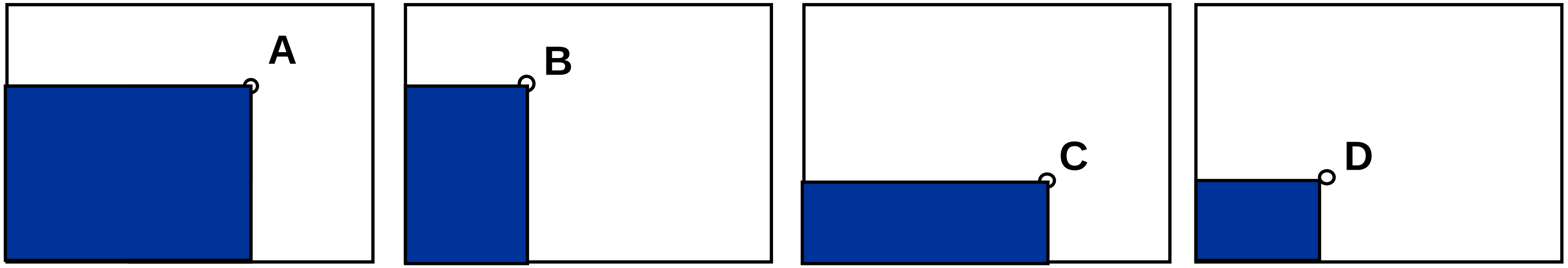
EWA filter

Pre-Filtering

- **Direct convolution methods are slow**
  - A pixel pre-image can be arbitrarily large
    - Along silhouettes
    - At the horizon of a textured plane
  - Can require averaging over thousands of texels
  - Texture filtering cost grows in proportion to projected texture area
- **Speed-up**
  - The texture can be prefiltered before rendering
    - Only a few samples are accessed for each screen sample
  - Two data structures are commonly used for prefiltering:
    - Integrated arrays (summed area tables - SAT)
    - Image pyramids (MIP-maps)
  - Space-variant filtering

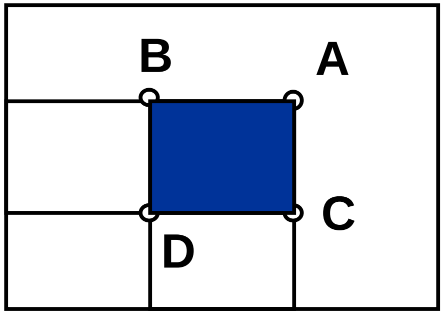
Summed Area Tables (SAT)

- **Per texel, store sum from *(0, 0)* to *(u, v)***



- **Evaluation of 2D integrals in constant time!**

$$\int_{Bx}^{Ax}\int_{Cy}^{Ay} \mathrm{I(x,y)}\,dx\,dy = \mathrm{A - B - C + D}$$

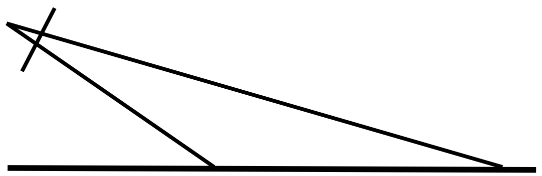


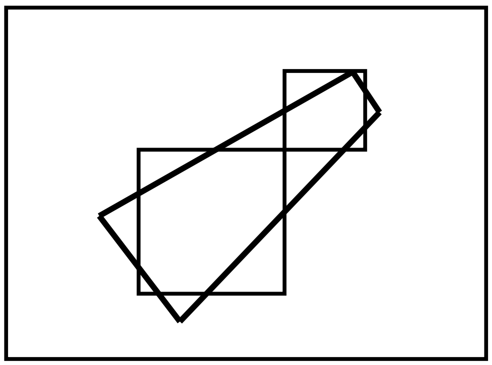
- **Many bits per texel (sum over million of pixels!)**

Integrated Arrays

- **Footprint assembly**
  - Good for space variant filtering
    - E.g. inclined view of terrain
  - Approximation of the pixel area by rectangular texel-regions
  - The more footprints the better accuracy
- **In practice**
  - Often fixed number of area samples
  - Done by sampling multiple locations within a pixel (e.g. 2x2), each with smaller footprint

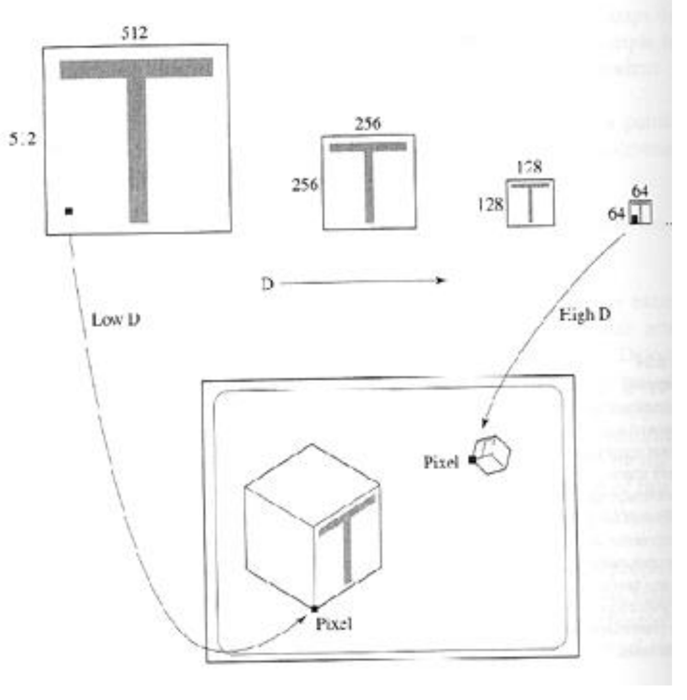
  ➔ Anisotropic (Texture) Filtering (AF)
    - GPUs allow selection of #samples (e.g. 4x, 8x, etc.)





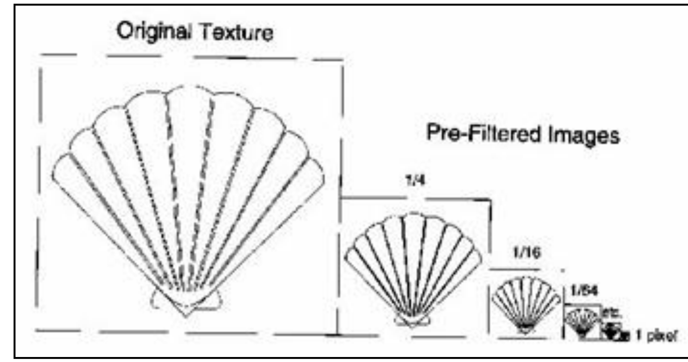
MIP-Mapping

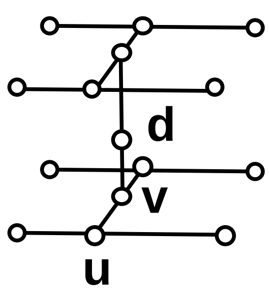
- **Texture available in multiple resolutions**
  - Pre-processing step averaging surrounding texels
  - Discrete number of filter sizes (powers of 2)
- **Rendering**
  - Select appropriate texture resolution level *n* (per pixel !!!)
  - Texel size(*n*) < extent of pixel footprint < texel size(*n+1*)

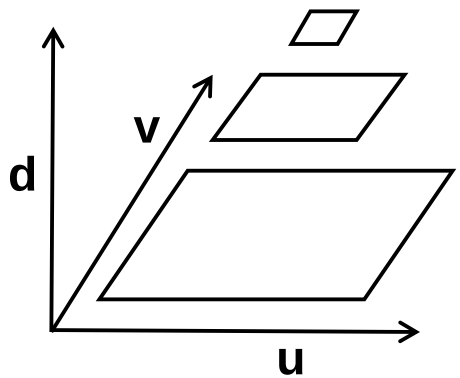


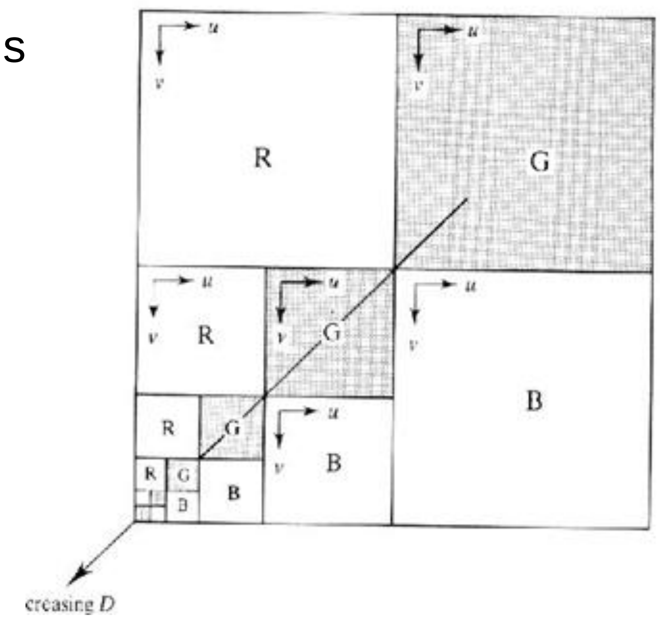
MIP-Mapping (2)

- **Multum In Parvo (MIP): much in little**
- **Hierarchical resolution pyramid**
  - Repeated averaging over 2x2 texels
- **Rectangular arrangement (RGB)**
- **Reconstruction**
  - Tri-linear interpolation of 8 nearest texels
    - Bilinear interpolation in levels *n* and *n+1*
    - Linear interpolation between the two levels



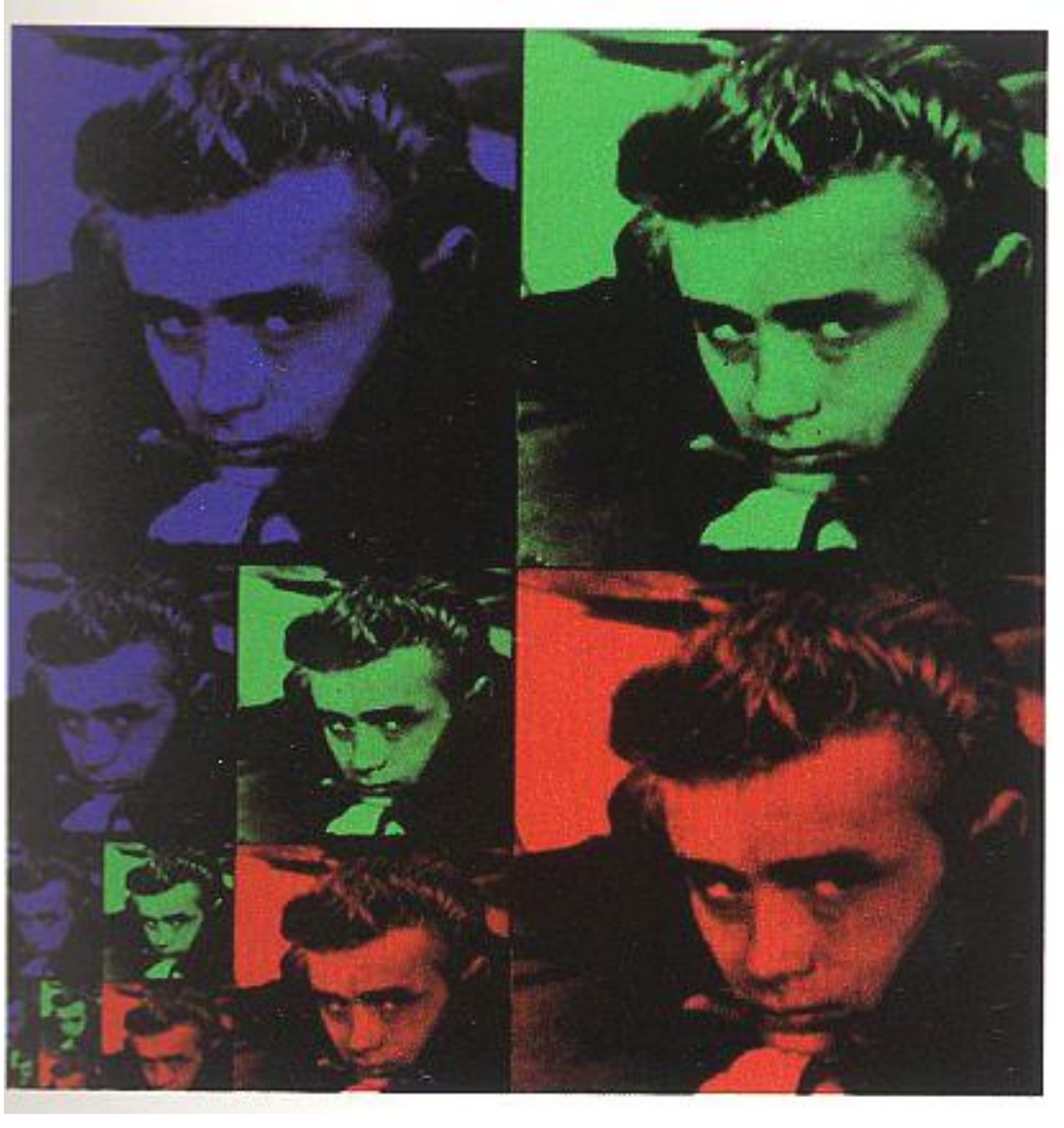






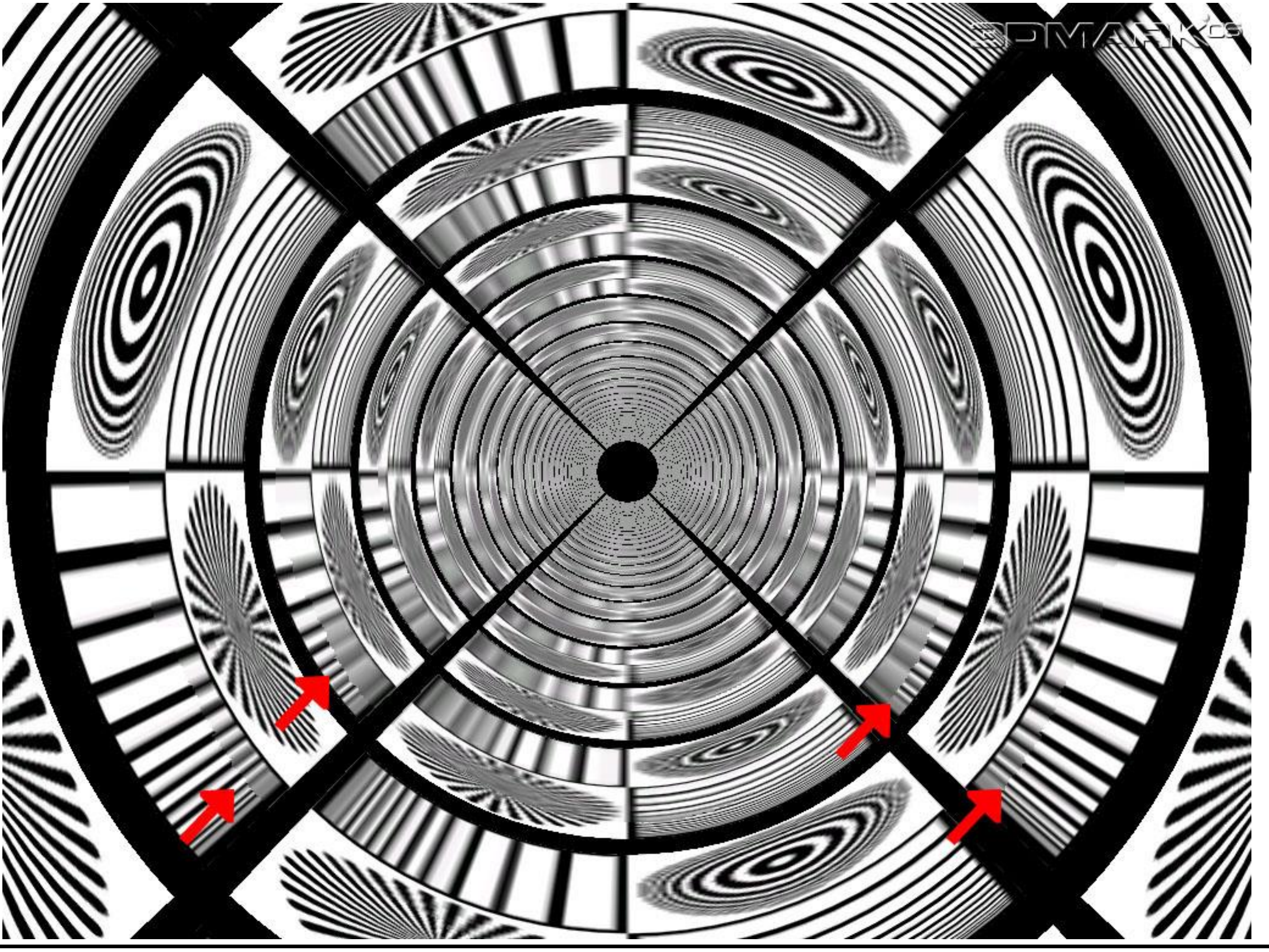
  - “Brilinear”: Trilinear only near transitions
    - Avoid reading 8 texels, most of the time

MIP-Map Example



Hardware Texture Filtering

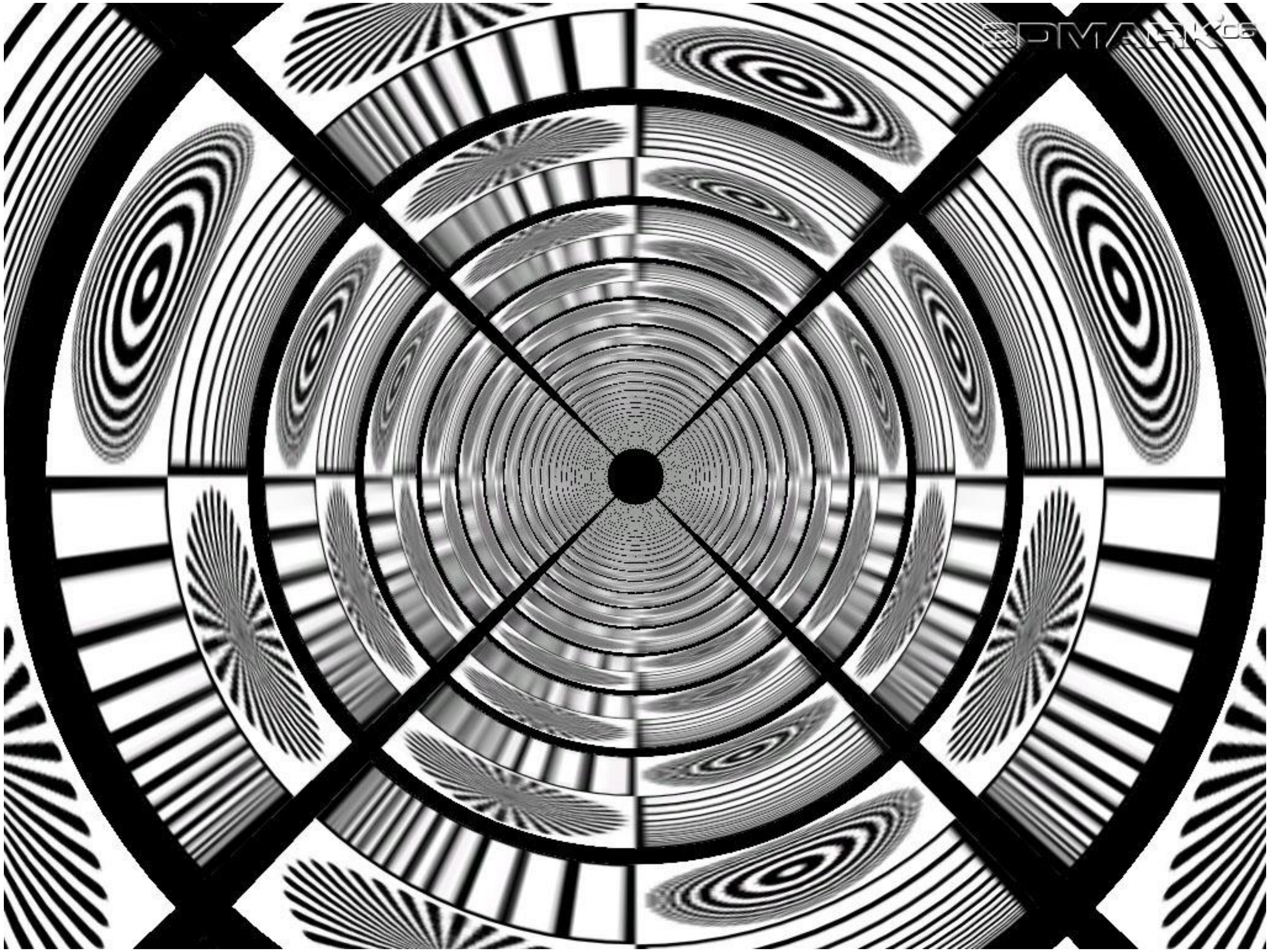
- **Bilinear filtering (in std. textured tunnel benchmark)**
  - Clearly visible transition between MIP-map levels



www.extremetech.com

Hardware Texture Filtering

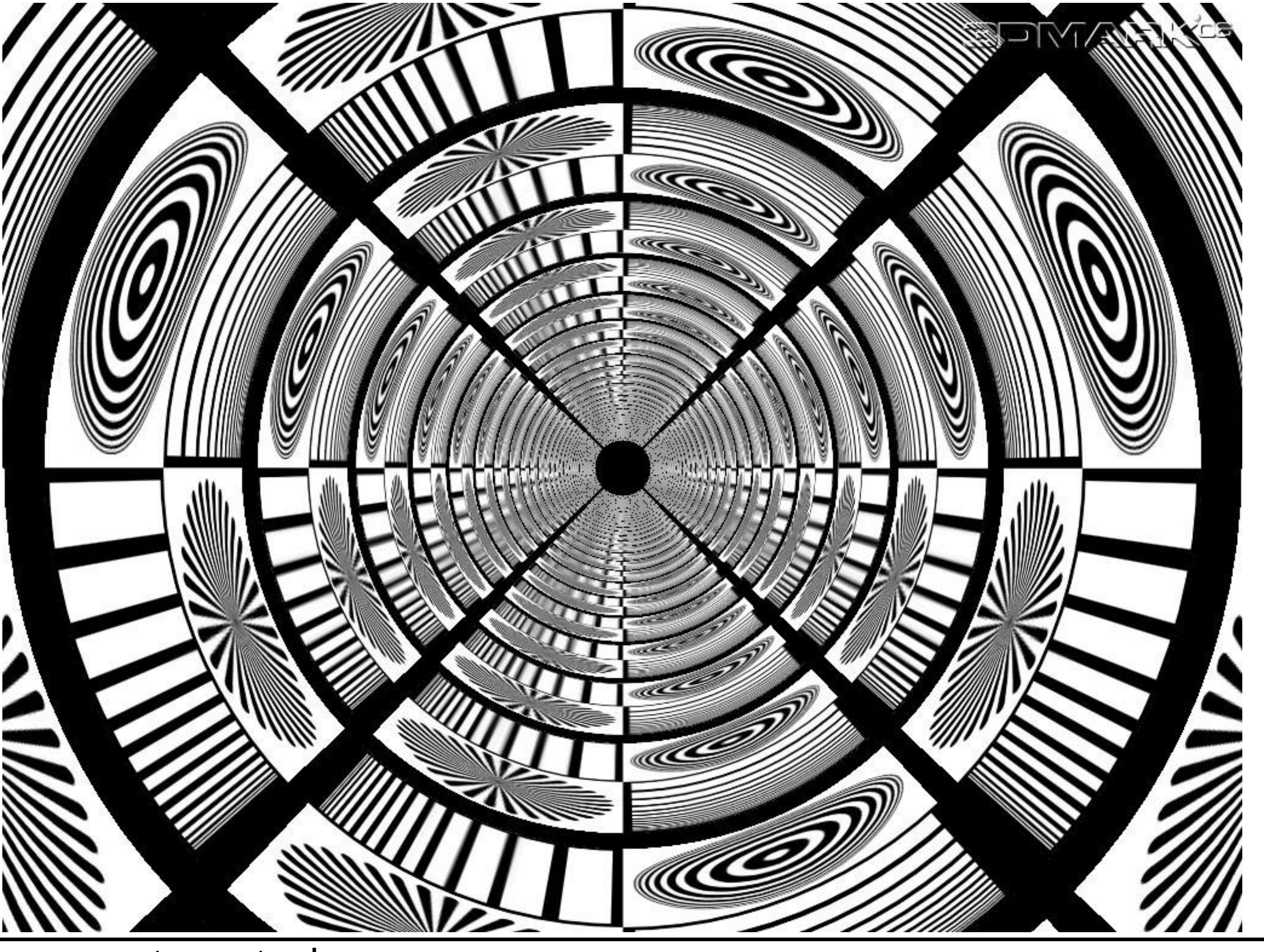
- **Trilinear filtering**
  - Hides the transitions between MIP-map levels



www.extremetech.com

Hardware Texture Filtering

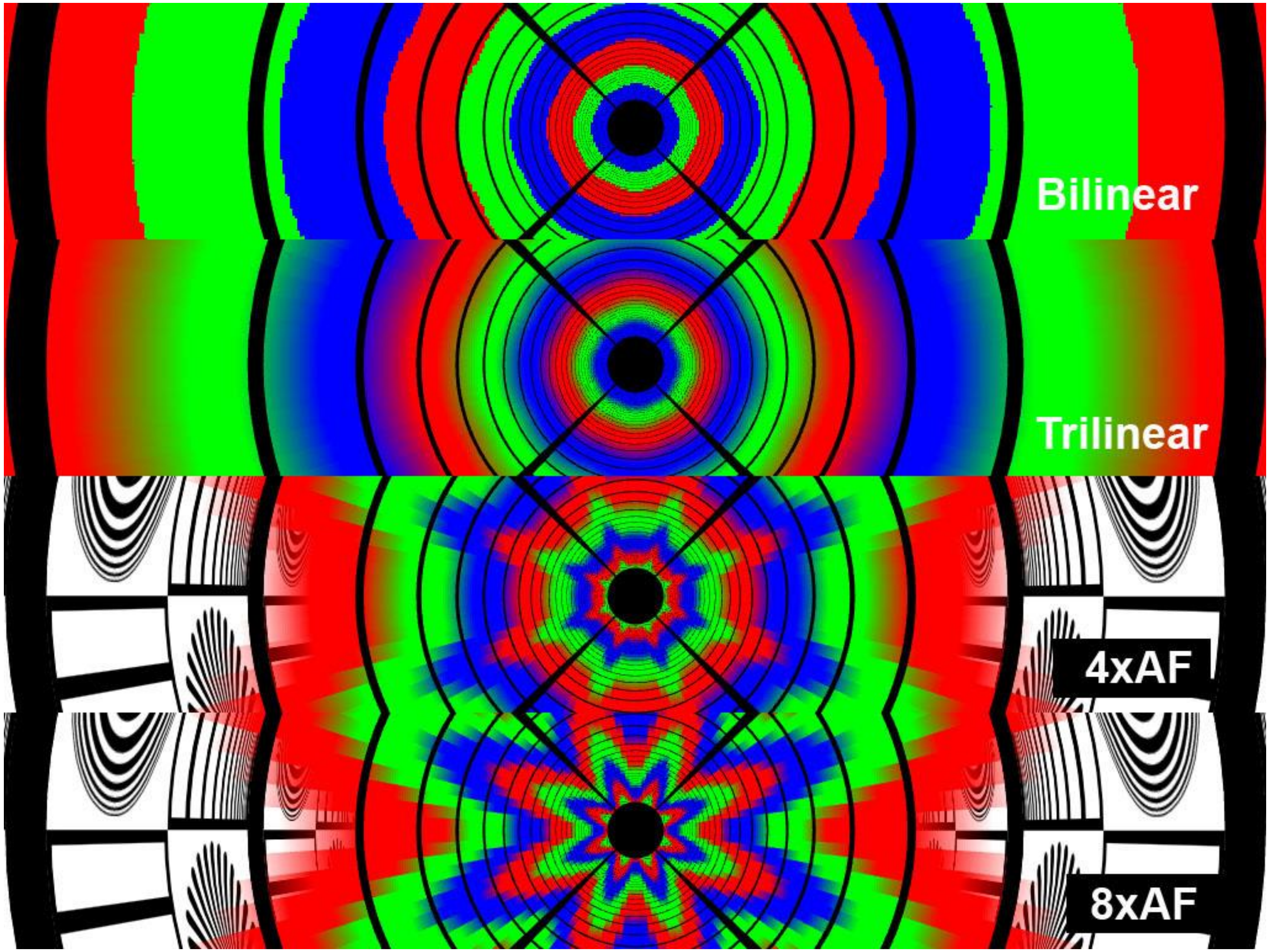
- **Anisotropic filtering (8x)**
  - Makes the textures much sharper along azimuthal coordinate



www.extremetech.com

Hardware Texture Filtering

- **Bilinear vs. trilinear vs. anisotropic filtering**
  - Using colored MIP-map levels



www.extremetech.com

Texture Caching in Hardware

- **All GPUs have small texture caches**
  - Designed for local effects (streaming cache)
    - No effects between frames, or so!
- **Mipmapping ensures ~1:1 ratio**
  - From pixel to texels
  - Both horizontally & vertically
- **Pixels rendered in small 2D groups**
  - Basic block is 2x2 „quad“
    - Used to compute „derivatives“
    - Using divided differences (left/right, up/down)
  - Lots of local coherence
- **Bi-/tri-linear filtering needs adjacent texels (up to 8 for trilinear)**
  - Most often just 1-2 new texel per pixel not in (local) cache