Computer Graphics

Texture Filtering

Philipp Slusallek
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 pixels 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

• **Reconstruction: filtering**
  – Recreate continuous signal (ideally also taking next step in account)

• **Display and perception (on some mostly unknown device!)**
  – 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 eventually becomes smaller than two pixels
    • At the Nyquist sampling limit
  – Rays sample textured plane at only one point per pixel
    • Can be either black or white – essentially “by chance”
    • Can have correlations at certain locations (low vs. high frequencies)
Filtering

- **Magnification (Zoom-in, texel > pixel)**
  - Maps 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
    - Possibly also higher order interpolation

- **Minification (Zoom-out, pixel > texel)**
  - Maps many texels to one pixel
  - Aliasing: Reconstructing high-frequency signals with low-frequency sampling
  - Anti-aliasing (low-pass filtering)
    - Averaging over (many) texels associated with the given pixel
    - Can be computationally expensive!
Aliasing Artifacts

- **Aliasing**
  - When texture insufficiently sampled
  - Incorrect pixel values
  - “Randomly” changing pixels when moving

- **Integration of “Pre-Image”**
  - Integration over pixel footprint in texture space
Pixel Pre-Image in Texture Space

- Circular pixel footprints have elliptic pre-images on planar surfaces due to projection.
- Square screen pixels form quadrilaterals
  - On planar surfaces
  - On curved surfaces, shape can be arbitrary (non-connected, etc...)
- Possible approximation by rectangle or quadrilateral
  - Or taking multiple samples within a pixel (see later)
Space-Variant Filtering

- **Space-variant filtering**
  - Mapping from texture space \((u,v)\) to screen space \((x,y)\) not affine
    - E.g., due to projection (see later, in context of rasterization)
    - Filtering changes with position

- **Space-variant filtering methods**
  - Direct convolution
    - Numerically compute the integral, e.g., with many samples
    - Potentially really costly
  - Pre-filtering
    - Pre-compute the integral for predefined regions of the texture
      - Lookup of integral much more efficiently at runtime
    - Must approximate actual pixel footprint with pre-computed regions
Direct Convolution

- **Convolution in texture space**
  - Texels weighted according to distance from pixel center
    - E.g. pyramidal filter kernel, truncated sinc, etc.
    - 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 space, where it is a quadrilateral, whose sides are assumed to be straight
    - More efficient: Find a suitable axis-aligned bounding box/rectangle
  - Map all texels inside this texture region to screen space
  - Form a weighted average of the mapped texels
    - E.g. using a two-dimensional lookup table indexed by each texel’s location within the pixel
**EWA Filtering**

- **EWA: Elliptical Weighted Average**
  - Compensate aliasing artifacts caused by perspective projection
  - EWA Filter = low-pass filter $\otimes$ warped reconstruction filter
  - Gaussian filtered with Gaussian is still a Gaussian

- **Can use rasterization HW for fast rendering**
  - Draw rectangle with suitable texture coord. that projects to pixel

---

**Diagram:**

- Rasterization of distorted rectangle projecting to circle on pixel with supersampling
- Projection
- Convolution
- Low-Pass Filter
- Texture Space

---

**Text:**

EWA texture resampling filter
EWA Filtering

Without EWA filtering

With EWA filtering
**Footprint Assembly**

- **Footprint assembly: Approximation of pixel integral**
  - Good for space variant filtering
    - E.g. inclined view of terrain
  - Approximation of the pixel area by rectangular texel-regions
  - More footprints → 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 max #samples (e.g., 4x, 8x, etc.)
    - Selected depending on amount of anisotropy
  - Each sample has its own footprint area/extent
  - Each gets independently projected and filtered
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 space sample
  – Two data structures are commonly used for prefiltering:
    • Integrated arrays (summed area tables - SAT)
    • Image pyramids (MIP-maps)
Summed Area Tables (SAT)

- Per texel, store sum over area from \((0, 0)\) to \((u, v)\)

- Evaluation of 2D integrals over AA-boxes in constant time!

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

- Needs many bits per texel (sum over million of pixels!)
MIP-Mapping

- **Texture available in multiple resolutions**
  - Pre-processing step that filters textures in each step
  - Discrete number of texture sizes (powers of 2)

- **Rendering**
  - Select appropriate texture resolution level $n$ (per sample !!!)
    - s.t.: texel size($n$) < extent of sample footprint < texel size($n+1$)
  - Needs derivative of texture coordinates
  - Can be computed from differences between pixels (divided differences)
    - → Rendering of Quads (2x2 pixels)
MIP-Mapping (2)

- Multum In Parvo (MIP): “much in little
- Hierarchical resolution pyramid
  - Repeated filtering over texture by 2x
- 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

Reducing the domain for linear interpolation improves performance
MIP-Map Example
Hardware Texture Filtering

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

- Trilinear filtering
  - Hides the transitions between MIP-map levels
Hardware Texture Filtering

- Anisotropic filtering (8x)
  - Makes the textures much sharper along azimuthal coordinate
Hardware Texture Filtering

- Bilinear vs. Brilinear vs. anisotropic filtering
  - Using colored MIP-map levels
Texture Caching in Hardware

- **All GPUs have small texture caches**
  - Designed for local effects (streaming cache)
    - No effects between meshes, frames, or such!
- **Mipmapping ensures ~1:1 ratio**
  - Between pixel and 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 texels per pixel that are not in (local) cache