Film

Digital Image Synthesis
Yung-Yu Chuang
11/5/2008

with slides by Pat Hanrahan and Matt Pharr

Film

- **`Film`** class simulates the sensing device in the simulated camera. It determines samples' contributions to the nearby pixels and writes the final floating-point image to a file on disk.
- Tone mapping operations can be used to display the floating-point image on a display.
- **`core/film.*`**

`Film`

```
class Film {
public:
  Film(int xres, int yres)
       : xResolution(xres), yResolution(yres) {}
  virtual ~Film() {}
  virtual void AddSample(Sample &sample, Ray &ray,
                         Spectrum &L, float alpha);
  virtual void WriteImage();
  virtual void GetSampleExtent(int *xstart,int
   *xend,
                               int *ystart, int
   *yend);
  // Film Public Data
  const int xResolution, yResolution;
};
```

`Camera` uses this to compute raster-to-camera transform

`ImageFilm`

- **`film/image.cpp`** implements the only film plug-in in pbrt. It simply filters samples and writes the resulting image.

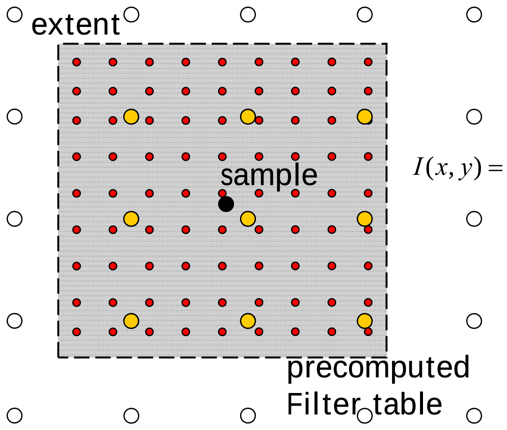
```
ImageFilm::ImageFilm(int xres, int yres,Filter *filt,
float crop[4],string &filename, bool premult, int wf)
{
  ...
  pixels = new BlockedArray<Pixel>(xPixelCount,
                                   yPixelCount);
  <precompute filter table>

}
```

useful for debugging, in NDC space

write frequency

AddSample



$$I(x,y)=\frac{\sum_i f(x-x_i,y-y_i)L(x_i,y_i)}{\sum_i f(x-x_i,y-y_i)}$$

grid of pixels

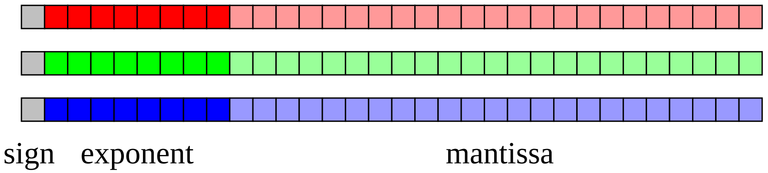
find the nearest neighbor
in the filter table

WriteImage

- Called to store the final image or partial images to disk
- The device-independent RGB is converted to the device-dependent RGB. First, convert to device-independent XYZ. Then, convert to device-dependent RGB according to your display. Here, pbrt uses the HDTV standard.
- Pbrt uses the EXR format to store image.

Portable floatMap (.pfm)

- 12 bytes per pixel, 4 for each channel

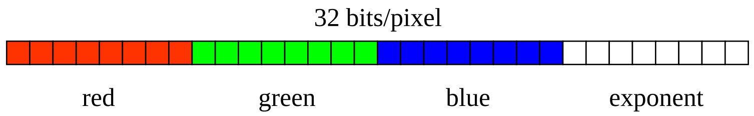


Text header similar to Jeff Poskanzer's .ppm image format:

```
PF
768 512
1
<binary image data>
```

Floating Point TIFF similar

Radiance format (.pic, .hdr, .rad)



(145, 215, 87, 149) =

(145, 215, 87) * 2^(149-128) =

1190000 1760000 713000

(145, 215, 87, 103) =

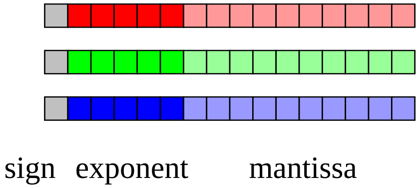
(145, 215, 87) * 2^(103-128) =

0.00000432 0.00000641 0.00000259

Ward, Greg. "Real Pixels," in Graphics Gems IV, edited by James Arvo, Academic Press, 1994

ILM’s OpenEXR (.exr)

- 6 bytes per pixel, 2 for each channel, compressed



- Several lossless compression options, 2:1 typical
- Compatible with the “half” datatype in NVidia's Cg
- Supported natively on GeForce FX and Quadro FX

- Available at http://www.openexr.net/

Tone mapping

- Converts HDR images to LDR image for display

```
void ApplyImagingPipeline(float *rgb,
       int xResolution, int yResolution,
       float *yWeight, weights to convert RGB to Y
       float bloomRadius, float bloomWeight,
       const char *toneMapName,
       const ParamSet *toneMapParams,
       float gamma, float dither,
       int maxDisplayValue)
```

- Not called in pbrt, but used by tools. It is possible to write a Film plugin to call tone mapping and store regular image.

Image pipeline

- Possibly apply bloom effect to image
- Apply tone reproduction to image
- Handle out-of-gamut RGB values
- Apply gamma correction to image
- Map image to display range
- Dither image

Bloom

without bloom

with bloom

blurred glow

a much brighter feel

Bloom

- Apply a very wide filter that falls off quickly to obtain a filtered image

$$f(x,y)=\left(1-\frac{\sqrt{x^2+y^2}}{d}\right)^4$$

- Blend the original image and the filtered image by a user-specified weight to obtain the final image

---

Tone mapping

- Two categories:
  - Spatially uniform (global): find a monotonic mapping to map pixel values to the display's dynamic range
  - Spatially varying (local): based on the fact that human eye is more sensitive to local contrast than overall luminance
- **`core/tonemap.h, tonemaps/*`**

```
class ToneMap {
public:
  // ToneMap Interface
  virtual ~ToneMap() { }
  virtual void Map(const float *y,int xRes,int yRes,
      float maxDisplayY, float *scale) const = 0;
};
```

input radiance array

display's limit

scale factor for each pixel

Maximum to white

```
class MaxWhiteOp : public ToneMap {
public:
   // MaxWhiteOp Public Methods
   void Map(const float *y, int xRes, int yRes,
            float maxDisplayY, float *scale) const {
        // Compute maximum luminance of all pixels
        float maxY = 0.;
        for (int i = 0; i < xRes * yRes; ++i)
              maxY = max(maxY, y[i]);
        float s = maxDisplayY / maxY;
        for (int i = 0; i < xRes * yRes; ++i)
              scale[i] = s;
   }
};
```

1. Does not consider HVS, two images different in scales will be rendered the same
2. A small number of bright pixels can cause the overall image too dark to see

---

Results

input

max-to-white

Contrast-based scale

- Developed by Ward (1994); compress the range but maintain the JND (just noticeable difference)

$$\Delta Y(Y^a) = 0.0594(1.219 + (Y^a)^{0.4})^{2.5}$$

- If the radiance is Y^a, the difference larger than ΔY is noticeable.

display radiance

real radiance

- Find s so that $\Delta Y(Y_d^a) = s\Delta Y(Y_w^a)$; it gives

$$s = \left(\frac{1.219 + (Y_d^a)^{0.4}}{1.219 + (Y_w^a)^{0.4}} \right)^{2.5}$$

- We calculate the log average radiance as ΔY_w^a

Results

input

contrast-based

Varying adaptation luminance

- It computes a local adaptation luminance that smoothly varies over the image. The local adaptation luminance is then used to compute a scale factor.
- How to compute a local adaptation luminance? Find most blurred value $Bs(x,y)$ so that the local contrast $lc(x,y)$ is smaller than a threshold.

$$lc(s,x,y) = \frac{B_s(x,y) - B_{2s}(x,y)}{B_s(x,y)}$$

$$Y^a(x,y) = B_s(x,y)$$

Varying adaptation luminance

- With the smooth local adaptation luminance image, the scale can be computed in a similar way to contrast-based method.

$$s(x,y) = \frac{T(Y^a(x,y))}{Y^a(x,y)}$$

target display luminance

$$T(Y) = Y_d^{\max} \frac{C(Y) - C(Y_{\min})}{C(Y_{\max}) - C(Y_{\min})} \qquad C(Y) = \int_0^Y \frac{dY'}{TVI(Y')}$$

capacity function (intensity levels in terms of JND)

$$C(Y) = \begin{cases} Y/0.0014 & Y < 0.0034 \\ 2.4483 + \log(Y/0.0034)/0.4027 & 0.0034 \le Y < 1 \\ 16.563 + (Y-1)/0.4027 & 1 \le Y \le 7.2444 \\ 32.0693 + \log(Y/7.2444)/0.0556 & \text{otherwise} \end{cases}$$

Results

with fixed radius

base on local contrast

Spatially varying nonlinear scale

- Empirical approach which works very well in practice. Similar to Reinhard 2002.

$$s(x,y)=\frac{\left(1+\frac{y(x,y)}{Y_{\max}^2}\right)}{1+y(x,y)}$$

not the luminance Y, but the y component in XYZ space

Results

input

nonlinear scale

Final stages

- Handle out-of-range gamut: scale by the maximum of three channels for each pixel (if the max > 1.0)
- Apply gamma correction: inverse gamma mapping for CRT's gamma mapping
- Map to display range: scaled by maxDisplayValue (usually 255)
- Dither image: add some noise in pixel values