Real Numbers, Real Images

Greg Ward
Anyhere Software

Introduction

This tutorial develops the thesis that the real world is best represented by real numbers, which are approximated by floating point values in the computer. As the floating point unit (FPU) continues to accelerate, outpacing the arithmetic logic unit (ALU), it makes more and more sense to do our graphics calculations entirely with real numbers. We even see the mainstream changing in this direction with the introduction of floating point frame buffers in a new generation of graphics cards, such as NVidia's GeForce FX.

We will cover several diverse topics in this tutorial, though all are relevant to the thesis of realism through real numbers. The first topic is measurement – how to obtain reasonable input values for computer graphics renderings. The second topic is lighting simulation – local and global illumination approximations and how to get a perceptually accurate result, most of the time. The third topic is image representation – how can we store our results without compromising them or taking up our entire disk drive? The fourth topic is image display – how can we take a high dynamic-range image and display it on a standard CRT – and are there better display technologies around the corner? Fifth and finally, we discuss high dynamic-range photography – how we can short-cut parts of our simulation through the use of image-based rendering and "image-based lighting".

The first part of these notes includes the text of the course slides, suitable for printing. The second part consists of the slides themselves, two per page. The remainder is a set of reprinted articles, which are arranged in seven appendices. These are all indexed and accessible via PDF bookmarks, or using the outline on the following pages.

Course Outline

I. Introduction
- A. Overview of computer graphics rendering – where it's been, and where it seems to be going.
- B. Why "real" numbers are better for rendering and imaging.

II. Measurement
- A. How do we obtain surface reflectances?
- B. How do we obtain surface textures (and milli geometry)?
- C. How do we obtain light source distributions?
- D. What is the best color space to work in?

III. Lighting Simulation
- A. Approximating local illumination
- B. Approximating global illumination
- C. Dealing with motion
- D. Exploiting human perception to accelerate rendering

IV. Image Representation
- A. Traditional graphics image formats
- B. High dynamic-range image formats
- C. What's the difference?

V. Image Display
- A. Tone-mapping (overview)
- B. High dynamic-range tone-mapping
- C. High dynamic-range display

VI. Image-based Techniques
- A. High dynamic-range photography
- B. Image-based lighting
- C. Image-based rendering

VII. Conclusions

Included Reprints

Appendix A: Tools for Lighting Design and Analysis
Reprinted from SIGGRAPH 1996 Course Notes

Appendix B: The Materials and Geometry Format
Reprinted from SIGGRAPH 1996 Course Note

Appendix C: Picture Perfect RGB Rendering Using Spectral Prefiltering and Sharp Color Primaries
Reprinted from 2002 Eurographics Workshop on Renderin

Appendix D: Detail to Attention: Exploiting Visual Tasks for Selective Rendering
Reprinted from 2003 Eurographics Symposium on Rendering

Appendix E: Overcoming Gamut and Dynamic Range Limitations in Digital Images
Reprinted from1998 Color Imaging Conference

Appendix F: A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes
Reprinted from 1997 LBL Technical Report

Appendix G: A High Dynamic Range Display Using Low and High Resolution Modulators
Reprinted from 2003 Society for Information Display Symposium

Author's Biography

Greg Ward (a.k.a.Greg Ward Larson) graduated in Physics from UC Berkeley in 1983 and earned a Masters in Computer Science from SF State University in 1985. Since 1985, he has been working in the field of light measurement, simulation, and rendering variously at the Lawrence Berkeley National Lab, EPFL Switzerland, SGI, Shutterfly, and Exponent. He is the author of the widely used *Radiance* package for lighting simulation and rendering, and is currently doing independent consulting in the areas of 2-D and 3-D graphics software. See http://www.anyhere.com/gward/ for more information.

Real Numbers, Real Images

Greg Ward
Anyhere Software

Course Outline

I. Introduction
II. Measurement
III. Lighting Simulation
IV. Image Representation
V. Image Display
VI. Image-based Techniques
VII. Conclusions

I. Introduction

- Graphics rendering software & hardware
  - Past
  - Present
  - Future
- Will graphics hardware take over?
- Why "real" numbers are better for rendering and imaging

Rendering Software Past

- Hidden-surface removal in a polygonal environment
  - Optional textures, bump maps, env. maps
- Local illumination
  - Gouraud and Phong shading
  - Shadow maps – some of them analytical!
- Ray-tracing for global illumination
  - Quadric surfaces and specular reflections

Graphics Hardware Past

- Fixed, 8-bit range for lights & materials
- Integer color operations
  - Phong and Gouraud shading hardware
  - Sometimes linear, sometimes pre-gamma
- Limited texture & fragment operations
- Output is 24-bit RGB sent to DAC (digital to analog converter) for analog display

Graphics Hardware Present

- Floating-point (FP) sources and materials
- Mix of integer and FP operations
  - Operations in linear or near-linear color space
- Extensive use of textures and MIP-maps
  - Programmable pixel shaders w/ some FP
- Output converted to 24-bit sRGB
  - Blending usually done in integer space
  - Display via digital video interface (DVI)

Rendering Software Present

- Global illumination (GI) in complex scenes
  - Environments with > 105 primitives common
  - Programmable shaders are the norm
- Micropolygon architectures prevalent
- Radiosity sometimes used for GI
- Ray-tracing (RT) used more and more

Rendering Software Future

- Hyper-complex environments (> 107 primitives)
  - Procedural scene descriptions
  - "Localized" version of global illumination
- Micropolygon architectures hang on
- Radiosity as we know it disappears
- Ray-tracing and Monte Carlo take over
  - Graceful handling of large data sets
  - Ordered rendering improves memory access

Graphics Hardware Future

- Floating-point operations throughout
  - All operations in linear color space
- High-level GPU programming standard
  - Compilers for multipass rendering
- Output converted to 64-bit RGBA
  - Cards output "layers" rather than images
  - Post-card blending on a novel display bus
  - New, high dynamic-range display devices

Will Hardware Take Over?

- No, rendering software will always exist
  - Needed for testing new ideas
  - Ultimately more flexible and controllable
  - Hardware does not address specialty markets
- But, graphics hardware will dominate
  - Programmable GPUs add great flexibility
  - Speed will always be critical to graphics
  - Read-back performance must be improved!

Why Real Numbers Are Better for Rendering & Imaging

- The natural range of light is huge ~1012
  - Humans adjust comfortably over 8 orders
  - Humans see simultaneously over 4 orders
- Color operations, including blending, must reproduce 10000:1 contrasts with final accuracy of 1% or better to fool us
- Human color sensitivity covers about twice the area of an sRGB display gamut

Dynamic Range
CCIR-709 (sRGB) Color Space
HDR Imaging Approach

- Render/Capture floating-point color space
- Store entire perceivable gamut (at least)
- Post-process in extended color space
- Apply tone-mapping for specific display
- HDR used extensively at ILM, Digital Domain, ESC, Rhythm & Hues

HDR Imaging Is Not New

- B&W negative film holds at least 4 orders of magnitude
- Much of the talent of photographers like Ansel Adams was darkroom technique
- "Dodge" and "burn" used to bring out the dynamic range of the scene on paper
- The digital darkroom provides new challenges and opportunities

HDR Tone-mapping
Post-production Possibilities

II. Measurement

- How do we obtain surface reflectances?
- How do we obtain surface textures (and milli geometry)?
- How do we obtain light source distributions?
- What is the best color space to work in?

Macbeth ColorChecker™ Chart

- Digital photo with ColorChecker™ under uniform illumination
- Compare points on image and interpolate
- Best to work with HDR image
- Accurate to ~ 10 ΔE

Radiance macbethcal Program

- Computes grayscale function and 3x3 color transform
- Maintain the same measurement conditions
- Calibrated pattern or uniform color capture
- Accurate to ~ 6 ΔE

Spectrophotometer

- Commercial spectrophotometers run about $5K US
- Measure reflectance spectrum for simulation under any light source
- Accurate to ~ 2 ΔE

BRDF Capture 1

BRDF Capture 2

Combined Capture Method 1

- Pietà Project
  www.research.ibm.com/pieta
  [Rushmeier et al. EGWR '98]
- Multi-baseline stereo camera with 5 lights
- Captured geometry and reflectance
- Sub-millimeter accuracy

Combined Capture Method 2

- CURET database
  - www1.cs.columbia.edu/CAVE/curet/
  - [Dana et al. TOG '99]
- Capture BTF (bidirectional texture function)
- Interpolate BTF during rendering

Combined Capture Method 3

- Lumitexel capture
  - [Lensch et al. EGWR '01]
- Capture 3-D position + normal + color as function of source position
- Fit data locally to BRDF model
- Render from BRDF

Light Source Distributions

- Often ignored, light source distributions are the first order of lighting simulation
- Data is comparatively easy to obtain
  - Luminaire manufacturers provide data files
    - See www.ledalite.com/resources/software
  - American and European standard file formats
  - Hardcopy photometric reports also available

Luminaire Data

- Photometric reports contain candela information per output direction
- All photometric measurements assume a far–field condition
- Interpolate directions and assume uniform over area

Candela Conversion

- A candela equals one lumen/steradian
- A lumen is approximately equal to 0.0056 watts of equal-energy white light
- To render in radiance units of watts/sr-m2
  - Multiply candelas by 0.0056/d**A** where d**A** is projected area in each output direction in m2

What Color Space to Use?

1) How Does *RGB* Rendering Work and When Does It Not?
2) Can *RGB* Accuracy Be Improved?
3) Useful Observations
4) Spectral Prefiltering
5) The von Kries White Point Transform
6) Experimental comparison of 3 spaces

A Brief Comparison of Color Rendering Techniques

- Spectral Rendering
  - ✓ N spectrally pure samples
- Component Rendering
  - ✓ M vector basis functions
- *RGB* (Tristimulus) Rendering
  - ✓ Tristimulus value calculations

Spectral Rendering

1. Divide visible spectrum into N wavelength samples
2. Process spectral samples separately throughout rendering calculation
3. Compute final display color using CIE color matching functions and standard transformations

Component Rendering

[Peercy, Siggraph '93]

1. Divide visible spectrum into M vector bases using component analysis
2. Process colors using MxM matrix multiplication at each interaction
3. Compute final display color with 3xM matrix transform

RGB (Tristimulus) Rendering

1. Precompute tristimulus values
2. Process 3 samples separately throughout rendering calculation
3. Compute final display color with 3x3 matrix transform (if necessary)

Rendering Cost Comparison

Strengths and Weaknesses

Spectral Aliasing

The Data Mixing Problem

- Typical situation:
  - Illuminants known to 5 nm resolution
  - Some reflectances known to 10 nm
  - Other reflectances given as tristimulus
- Two alternatives:
  A. Reduce all spectra to lowest resolution
  B. Interpolate/synthesize spectra [Smits '99]

Status Quo Rendering

- White Light Sources
  - E.g., (R,G,B)=(1,1,1)
- *RGB* material colors obtained by dubious means
  - E.g., "That looks pretty good."
    - ✓ This actually works for fictional scenes!
- Color correction with ICC profile if at all

When Does RGB Rendering Normally Fail?

- When you start with measured colors
- When you want to simulate color appearance under another illuminant
- When your illuminant and surface spectra have sharp peaks and valleys

Can *RGB* Accuracy Be Improved?

- Identify and minimize sources of error
  - Source-surface interactions
  - Choice of rendering primaries
- Overcome ignorance and inertia
  - Many people render in *RGB* without really understanding what it means
  - White-balance problem scares casual users away from colored illuminants

A Few Useful Observations

a) Direct illumination is the first order in any rendering calculation
b) Most scenes contain a single, dominant illuminant spectrum
c) Scenes with mixed illuminants will have a color cast regardless

Picture Perfect *RGB* Rendering

1. Identify dominant illuminant spectrum
   a) Prefilter material spectra to obtain tristimulus colors for rendering
   b) Adjust source colors appropriately
2. Perform tristimulus (*RGB*) rendering
3. Apply white balance transform and convert pixels to display color space

Spectral Prefiltering
Prefiltering vs. Full Spectral Rendering

+ Prefiltering performed once per material vs. every rendering interaction
+ Spectral aliasing and data mixing problems disappear with prefiltering
- However, mixed illuminants and interreflections not computed exactly

Quick Comparison
The von Kries Transform for Chromatic Adaptation

Chromatic Adaptation Matrix

- The matrix **M***C* transforms *XYZ* into an "adaptation color space"
- Finding the optimal CAM is an under-constrained problem -- many candidates have been suggested
- "Sharper" color spaces tend to perform better for white balance transforms
  - See [Finlayson & Susstrunk, CIC '00]

Three Tristimulus Spaces for Color Rendering

- CIE *XYZ*
  - Covers visible gamut with positive values
  - Well-tested standard for color-matching
- *sRGB*
  - Common standard for image encoding
  - Matches typical CRT display primaries
- *Sharp RGB*
  - Developed for chromatic adaptation

XYZ Rendering Process

1. Apply prefiltering equation to get absolute *XYZ* colors for each material
   a) Divide materials by illuminant:

   b) Use absolute *XYZ* colors for sources
2. Render using tristimulus method
3. Finish w/ CAM and display conversion

sRGB Rendering Process

1. Perform prefiltering and von Kries transform on material colors
   a) Model dominant light sources as neutral
   b) For spectrally distinct light sources use:

2. Render using tristimulus method
3. Resultant image is *sRGB*

Sharp RGB Rendering Process

1. Prefilter material colors and apply von Kries transform to *Sharp RGB* space:

2. Render using tristimulus method
3. Finish up CAM and convert to display

Our Experimental Test Scene

Experimental Results

- Three lighting conditions
  - Single 2856°K tungsten light source
  - Single cool white fluorescent light source
  - Both light sources (tungsten & fluorescent)
- Three rendering methods
  - Naïve *RGB* (assumes equal-energy white)
  - Picture Perfect *RGB*
  - Full spectral rendering (380 to 720 nm / 69 samp.)
- Three color spaces (*XYZ*, *sRGB*, *Sharp RGB*)

Example Comparison (*sRGB*)

ΔE* Error Percentiles for All Experiments

Results Summary

- Prefiltering has ~1/6 the error of naïve rendering for single dominant illuminant
- Prefiltering errors similar to naïve in scenes with strongly mixed illuminants
- CIE *XYZ* color space has 3 times the rendering errors of *sRGB* on average
- *Sharp RGB* rendering space reduces errors to 1/3 that of *sRGB* on average

III. Lighting Simulation

- Approximating local illumination
- Approximating global illumination
- Dealing with motion
- Exploiting human perception to accelerate rendering

Local Illumination

- Local illumination is the most important part of rendering, and *everyone* gets it wrong (including me)
- Real light-surface interactions are incredibly complex, and humans have evolved to perceive many subtleties
- The better your local illumination models, the more realistic your renderings

LI Advice: Use Physical Range

- Non-metallic surfaces rarely have specular reflectances greater than 7%
  - Determined by the index of refraction, $n < 1.7$
- Physically plausible BRDF models obey energy conservation and reciprocity
  - Phong model often reflects > 100% of incident
- *RGB* reflectances may be slightly out of [0,1] range for highly saturated colors

LI Advice: Add Fresnel Factor

- Specular reflectance goes up near grazing for all polished materials – here is a good approximation for Fresnel reflection:

- Simpler & faster than standard formula
- Improves accuracy and appearance at silhouettes

Fresnel Approximation
LI Advice: Texture Carefully

- Pay attention to exactly how your image textures affect your average and peak reflectances
  - Are they still in a physically valid range?
- Use bump maps sparingly
  - Odd artifacts arise when geometry and surface normals disagree strongly
  - Displacement maps are better

LI Advice: Use BTF Model

- Use CURET data to model view-dependent appearance under different lighting using *TensorTexture* technique
  - See "TensorTextures", M. Alex O. Vasilescu and D. Terzopoulos, Sketch and Applications SIGGRAPH 2003 San Diego, CA, July, 2003.
    www.cs.toronto.edu/~maov/tensortextures/tensortextures_sigg03.pdf

Global Illumination

- Global illumination will not fix problems caused by poor local illumination, but…
  - GI adds another dimension to realism, and
  - GI gets you absolute answers for lighting
- Radiosity methods compute form factors
  - Says nothing about global illumination
- Ray-tracing methods intersect rays
  - Again, this is not a useful distinction

GI Algorithm Characteristics

- Traces rays
- Subdivides surfaces into quadrilaterals
- Employs form factor matrix
- Deposits information on surfaces
  - Using grid
  - Using auxiliary data structure (e.g., octree)
- Requires multiple passes

GI Example 1: Hemicube Radiosity [Cohen et al. '86]

- × Traces rays
- ✓ Subdivides surfaces into quadrilaterals
- ✓ Employs form factor matrix
- ✓ Deposits information on surfaces
  - ✓ Using grid
  - × Using auxiliary data structure (e.g., octree)
- ✓ Requires multiple passes

GI Example 2: Particle Tracing [Shirley et al. '95]

- ✓ Traces rays
- × Subdivides surfaces into quadrilaterals
  - ✓ But triangles, yes
- × Employs form factor matrix
- ✓ Deposits information on surfaces
  - × Using grid
  - ✓ Using auxiliary data structure (T-mesh)
- ✓ Requires multiple passes

GI Example 3: Monte Carlo Path Tracing [Kajiya '86]

- ✓ Traces rays
- × Subdivides surfaces into quadrilaterals
- × Employs form factor matrix
- × Deposits information on surfaces
- × Requires multiple passes

GI Example 4: *Radiance*

- ✓ Traces rays
- × Subdivides surfaces into quadrilaterals
- × Employs form factor matrix
- ✓ Deposits information on surfaces
  - × Using grid
  - ✓ Using auxiliary data structure (octree)
- × Requires multiple passes

The Rendering Equation
Radiance Calculation Methods

- Direct calculation removes large incident
- Indirect calculation handles most of the rest
- Secondary light sources for problem areas
- Participating media (adjunct to equation)

Radiance Direct Calculation

- Selective Shadow Testing
  - Only test significant sources
- Adaptive Source Subdivision
  - Subdivide large or long sources
- Virtual Light Source Calculation
  - Create virtual sources for beam redirection

Selective Shadow Testing

- Sort potential direct contributions
  - Depends on sources and material
- Test shadows from most to least significant
  - Stop when remainder is below error tolerance
- Add in untested remainder
  - Use statistics to estimate visibility

Selective Shadow Testing (2)

Adaptive Source Subdivision

Virtual Light Source Calculation

Indirect Calculation

- Specular Sampling
  - sample rays over scattering distribution
- Indirect Irradiance Caching
  - sample rays over hemisphere
  - cache irradiance values over geometry
  - reuse for other views and runs

Indirect Calculation (2)

Specular Sampling

Energy-preserving Non-linear Filters

Indirect Irradiance Caching

Indirect Irradiance Gradients

- From hemisphere sampling, we can also compute change w.r.t. position and direction
- Effectively introduces higher-order interpolation method, i.e., cubic vs. linear
- See [Ward & Heckbert, EGWR '92] for details

Irradiance Gradients (2)

Secondary Light Sources

- Impostor surfaces around sources
  - decorative luminaires
  - clear windows
  - complex fenestration
- Computing secondary distributions
  - the **mkillum** program

Impostor Source Geometry

- Simplified geometry for shadow testing and illumination computation
  - fits snugly around real geometry, which is left for rendering direct views

Computing Secondary Distributions

- Start with straight scene description
- Use **mkillum** to compute secondary sources
- Result is a more efficient calculation

Using Pure Monte Carlo

Using Secondary Sources

Participating Media

- Single-scatter approximation
- The mist material type
  - light beams
  - constant density regions
- Rendering method

Single-scatter Approximation

- Computes light scattered into path directly from specified light sources
- Includes absorption and ambient scattering

The *Mist* Material Type

- Demark volumes for light beams
- Can change medium density or scattering properties within a volume

Rendering Method

- After standard ray value is computed:
  - compute ambient in-scattering, out-scattering and absorption along ray path
  - compute in-scattering from any sources identified by *mist* volumes ray passes through
    - this step accounts for anisotropic scattering as well

What About Animation?

- Easy: render frames independently
  - What about motion blur?
  - Also, is this the most efficient approach?
- Better: Image-based frame interpolation
  - **Pinterp** program
    - First released in May 1990 (*Radiance* 1.2)
    - Combines pixels with depth for in-between frames
    - Motion-blur capability
    - Moving objects still a problem

Exploit Human Perception

- Video compression community has studied what motions people notice
- In cases where there is an associated task, we can also exploit *inattentional blindness*
- Image-based motion blur can be extended to objects with a little additional work

Perceptual Rendering Framework

- "Just in time" animation system
- Exploits inattentional blindness and IBR
- Generalizes to other rendering techniques
  - Demonstration system uses *Radiance* ray-tracer
  - Potential for real-time applications
- Error visibility tied to attention and motion

Rendering Framework

Example Frame w/ Task Objects

Error Map Estimation

- Stochastic errors may be estimated from neighborhood samples
- Systematic error bounds may be estimated from knowledge of algorithm behavior
- Estimate accuracy is not critical for good performance

Initial Error Estimate
Image-based Refinement Pass

- Since we know exact motion, IBR works very well in this framework
- Select image values from previous frame
  - Criteria include coherence, accuracy, agreement
- Replace current sample and degrade error
  - Error degradation results in sample retirement

Contrast Sensitivity Model

Error Conspicuity Model

Error Conspicuity Map

Final Sample Density

Implementation Example

- Compared to a standard rendering that finished in the same time, our framework produced better quality on task objects
- Rendering the same high quality over the entire frame would take about 7 times longer using the standard method

Example Animation

- The following animation was rendered at two minutes per frame on a 2000 model G3 laptop computer (Apple PowerBook)
- Many artifacts are intentionally visible, but less so if you are performing the task

Algorithm Visualization

IV. Image Representation

- Traditional graphics image formats
  - Associated problems
- High dynamic-range (HDR) formats
  - Standardization efforts

Traditional Graphics Images

- Usually 8-bit integer range per primary

- *sRGB* color space matches CRT monitors, not human vision

Extended Graphics Formats

- 12 or even 16 bits/primary in TIFF

- Photo editors (i.e., Photoshop™) do not respect this range, treating 65535 as white
- Camera raw formats are an archiving disaster, and should be avoided
- RGB still constrains color gamut

The 24-bit Red Green Blues

- Although 24-bit *sRGB* is reasonably matched to CRT displays, it is a poor match to human vision
  - People can see twice as many colors
  - People can see twice the log range

Q: Why did they base a standard on existing display technology?

A: Because signal processing *used* to be expensive…

High Dynamic Range Images

- High Dynamic Range Images have a wider gamut and contrast than 24-bit RGB
  - Preferably, the gamut and dynamic range covered exceed those of human vision

Advantage 1: an image standard based on human vision won't need frequent updates

Advantage 2: floating point pixels open up a vast new world of image processing

Some HDRI Formats

- *Pixar* 33-bit log-encoded TIFF
- *Radiance* 32-bit RGBE and XYZE
- IEEE 96-bit TIFF & Portable FloatMap
- LogLuv TIFF (24-bit and 32-bit)
- *ILM* 48-bit OpenEXR format

Pixar Log TIFF Codec

Purpose: To store film recorder input

- Implemented in Sam Leffler's TIFF library
- 11 bits each of log red, green, and blue
- 3.8 orders of magnitude in 0.4% steps
- ZIP lossless entropy compression
- Does not cover visible gamut
- Dynamic range marginal for image processing

Radiance RGBE & XYZE

Purpose: To store GI renderings

- Simple format with free source code
- 8 bits each for 3 mantissas + 1 exponent
- 76 orders of magnitude in 1% steps
- Run-length encoding (20% avg. compr.)
- RGBE format does not cover visible gamut
- Color quantization not perceptually uniform
- Dynamic range at expense of accuracy

Radiance Format (.pic, .hdr)
IEEE 96-bit TIFF

Purpose: To minimize translation errors

- Most accurate representation
- Files are enormous
  - 32-bit IEEE floats do not compress well

24-bit LogLuv TIFF Codec

Purpose: To match human vision in 24 bits

- Implemented in Leffler's TIFF library
- 10-bit LogL + 14-bit CIE (u',v') lookup
- 4.8 orders of magnitude in 1.1% steps
- Just covers visible gamut and range
- No compression

24 -bit LogLuv Pixel
32-bit LogLuv TIFF Codec

Purpose: To surpass human vision

- Implemented in Leffler's TIFF library
- 16-bit LogL + 8 bits each for CIE (u',v')
- 38 orders of magnitude in 0.3% steps
- Run-length encoding (30% avg. compr.)
- Allows negative luminance values

32-bit LogLuv Pixel

ILM OpenEXR Format

Purpose: HDR lighting and compositing

- 16-bit/primary floating point (sign-e5-m10)
- 9.6 orders of magnitude in 0.1% steps
- Wavelet compression of about 40%
- Negative colors and full gamut RGB
- Open Source I/O library released Fall 2002

ILM's OpenEXR (.exr)

HDRI Post-production

Example HDR Post-processing

Image Representation Future

- JPEG and other 24-bit formats here to stay
- Lossless HDRI formats for high-end
- Compressed HDRI formats are desirable for digital camera applications
  - JPEG 2000 seems like a possible option
    - Adobe doesn't like its proprietary inception
  - Others pushing for a "standard raw sensor" format, but I doubt it would work

V. Image Display

- How do we display an HDR image?
- There are really just two options:
  1. Tone-map HDRI to fit in displayable range
  2. View on a high dynamic-range display
- Many tone-mapping algorithms have been proposed for dynamic-range compression
- But, there are no HDR displays! (Or are there?)

HDRI Tone-mapping

- Tone-mapping (a.k.a. tone-reproduction) is a well-studied topic in photography
  - Traditional film curves are carefully designed
- Computer imaging offers many new opportunities for dynamic TRC creation
- Additionally, tone reproduction curves may be manipulated locally over an image

Tone-mapping to LDR Display

- A renderer is like an "ideal" camera
- TM is medium-specific and goal-specific
- Need to consider:
  - Display gamut, dynamic range, and surround
  - What do we wish to simulate?
    - Cinematic camera and film?
    - Human visual abilities and disabilities?

TM Goal: Colorimetric

TM Goal: Match Visibility

TM Goal: Optimize Contrast

One Tone-mapping Approach

- Generate histogram of log luminance
- Redistribute luminance to fit output range
- Optionally simulate human visibility
  - match contrast sensitivity
  - scotopic and mesopic color sensitivity
  - disability (veiling) glare
  - loss of visual acuity in dim environments

Histogram Adjustment

Contrast & Color Sensitivity

Veiling Glare Simulation

Other Tone Mapping Methods

- Retinex-based [Jobson et al. IEEE TIP July '97]
- Psychophysical [Pattanaik et al. Siggraph '98]
- Local Contrast [Ashikhmin, EGWR '02]
- Photographic [Reinhard et al. Siggraph '02]
- Bilateral Filtering [Durand & Dorsey, Siggraph '02]
- Gradient Domain [Fattal et al. Siggraph '02]

High Dynamic-range Display

- Early HDR display technology
  - Industrial high luminance displays (e.g., for air traffic control towers) not really HDR
  - Static stereo viewer for evaluating TMO's
- Emerging HDR display devices
  - Collaborative work at the University of British Columbia in Vancouver, Canada

Static HDR Viewer

HDR Viewer Schematic

Viewer Image Preparation

- Two transparency layers yield 1:104 range
  - B&W "scaling" layer
  - Color "detail" layer
- Resolution difference avoids registration (alignment) problems
- 120º hemispherical fisheye perspective
- Correction for chromatic aberration

Example Image Layers

UBC Structured Surface Physics Lab HDR Display

- First generation DLP/LCD prototype
  - 1024x768 resolution
  - 10,000:1 dynamic range
  - 7,000 cd/m2 maximum luminance
- Next generation device w/ LED backlight
  - Flat-panel design presented at SID
  - 10,000:1 DR and 10,000 max. luminance

UBC HDR Display Prototype

VI. Image-based Techniques

- High dynamic-range photography
  - Using *Photosphere*
- Image-based lighting
- Image-based rendering

HDR Photograhy

- Standard digital cameras capture about 2 orders of magnitude in sRGB color space
- Using multiple exposures, we can build up high dynamic range image of static scene
- In the future, manufacturers may build HDR imaging into camera hardware

Hand-held HDR Photography

- Use “auto-bracketing” exposure feature
- Align exposures horizontally and vertically
- Deduce camera response function using [Mitsunaga & Nayar ‘99] polynomial fit
- Recombine images into HDR image
- Optionally remove lens flare

Auto-bracket Exposures

LDR Exposure Alignment

Estimated Camera Response

Combined HDR Image

Tone-mapped Display

Best Single Exposure

Lens Flare Removal

Photosphere HDRI Browser

- Browses High Dynamic Range Images
  - *Radiance* RGBE format
  - TIFF LogLuv and floating point formats
  - OpenEXR short float format
- Makes HDR images from bracketed exposures
- Maintains Catalog Information
  - Subjects, keywords, albums, comments, etc.
- Tracks Image Files
  - Leaves file management & modification to user

Realized Features

- Fast, interactive response
- Thumbnails accessible when images are not
- Interprets Exif header information
- Builds photo albums & web pages
- Displays & edits image information
- Provides drag & drop functionality
- User-defined database fields

Unrealized Features

- Accurate color reproduction on all devices
- Plug-in interface for photo printing services
- Linux and Windows versions
- More supported image formats
  - Currently JPEG, TIFF, *Radiance*, OpenEXR

Browser Layout

Viewer Layout

Info Window Layout

Browser Files

Browser Architecture

Photosphere Demo

Image-based Lighting

- Photograph silver sphere using HDR method
- Place as environment map in scene to render
- Sample map to obtain background values

Image-based Rendering

- Mixed reality is the future for graphics
- High dynamic-range imaging is the key
- Accuracy in rendering is also critical for seamless integration
- A lot of work has been done in the areas of image-based lighting and rendering, but we've only scratched the surface
  - Films like *The Matrix* rely heavily on IBL/IBR

IBR/IBL Example

VII. Conclusions

- Two paths to realism:
  1. Work like nuts until it "looks OK," or
  2. Apply psychophysics of light and vision
- As authors of rendering software, we can save users a lot of (1) with a little of (2)
- Real numbers are needed for physical simulation, as values are unbounded
- The eye and brain are analog devices

Further Reference

- www.anyhere.com/gward
  - publication list with online links
  - LogLuv TIFF pages and images
- www.debevec.org
  - publication list with online links
  - *Radiance* RGBE images and light probes
  - *HDRshop* and related tools
- www.idruna.com
  - *Photogenics* HDR image editor
- radsite.lbl.gov/radiance
  - *Radiance* rendering software and links

Real Numbers, Real Images

Greg Ward
Anyhere Software

Radiance image courtesy Veronica Sundstedt & Patrick Ledda, Bristol University

Real Numbers, Real Images

Course Outline

I. Introduction
II. Measurement
III. Lighting Simulation
IV. Image Representation
V. Image Display
VI. Image-based Techniques
VII. Conclusions

I. Introduction

- Graphics rendering software & hardware
  - Past
  - Present
  - Future
- Will graphics hardware take over?
- Why "real" numbers are better for rendering and imaging

Rendering Software Past

- Hidden-surface removal in a polygonal environment
  - Optional textures, bump maps, env. maps
- Local illumination
  - Gouraud and Phong shading
  - Shadow maps – some of them analytical!
- Ray-tracing for global illumination
  - Quadric surfaces and specular reflections

Real Numbers, Real Images

Graphics Hardware Past

- Fixed, 8-bit range for lights & materials
- Integer color operations
  - Phong and Gouraud shading hardware
  - Sometimes linear, sometimes pre-gamma
- Limited texture & fragment operations
- Output is 24-bit RGB sent to DAC (digital to analog converter) for analog display

Real Numbers, Real Images

Graphics Hardware Present

- Floating-point (FP) sources and materials
- Mix of integer and FP operations
  - Operations in linear or near-linear color space
- Extensive use of textures and MIP-maps
  - Programmable pixel shaders w/ some FP
- Output converted to 24-bit sRGB
  - Blending usually done in integer space
  - Display via digital video interface (DVI)

Real Numbers, Real Images

Rendering Software Present

- Global illumination (GI) in complex scenes
  - Environments with > 10^5 primitives common
  - Programmable shaders are the norm
- Micropolygon architectures prevalent
- Radiosity sometimes used for GI
- Ray-tracing (RT) used more and more

Real Numbers, Real Images

Rendering Software Future

- Hyper-complex environments (> 10^7 primitives)
  - Procedural scene descriptions
  - "Localized" version of global illumination
- Micropolygon architectures hang on
- Radiosity as we know it disappears
- Ray-tracing and Monte Carlo take over
  - Graceful handling of large data sets
  - Ordered rendering improves memory access

Real Numbers, Real Images

Graphics Hardware Future

- Floating-point operations throughout
  - All operations in linear color space
- High-level GPU programming standard
  - Compilers for multipass rendering
- Output converted to 64-bit RGBA
  - Cards output "layers" rather than images
  - Post-card blending on a novel display bus
  - New, high dynamic-range display devices

Real Numbers, Real Images

Will Hardware Take Over?

- No, rendering software will always exist
  - Needed for testing new ideas
  - Ultimately more flexible and controllable
  - Hardware does not address specialty markets
- But, graphics hardware will dominate
  - Programmable GPUs add great flexibility
  - Speed will always be critical to graphics
  - Read-back performance must be improved!

Real Numbers, Real Images

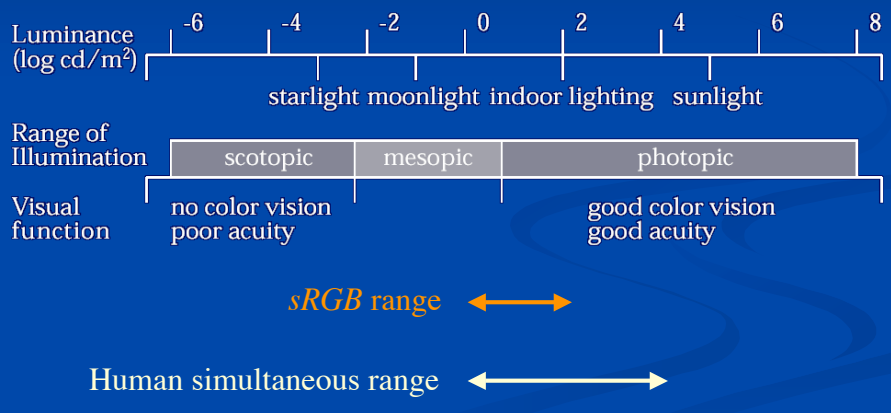
Why Real Numbers Are Better for Rendering & Imaging

- The natural range of light is huge ~10^{12}
  - Humans adjust comfortably over 8 orders
  - Humans see simultaneously over 4 orders
- Color operations, including blending, must reproduce 10000:1 contrasts with final accuracy of 1% or better to fool us
- Human color sensitivity covers about twice the area of an sRGB display gamut

Real Numbers, Real Images

Dynamic Range

From [Ferwerda et al, Siggraph '96]

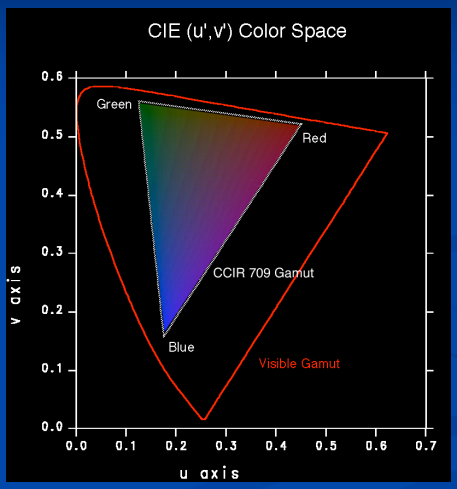


sRGB range ⟷

Human simultaneous range ⟷

Real Numbers, Real Images

CCIR-709 (sRGB) Color Space



Real Numbers, Real Images

HDR Imaging Approach

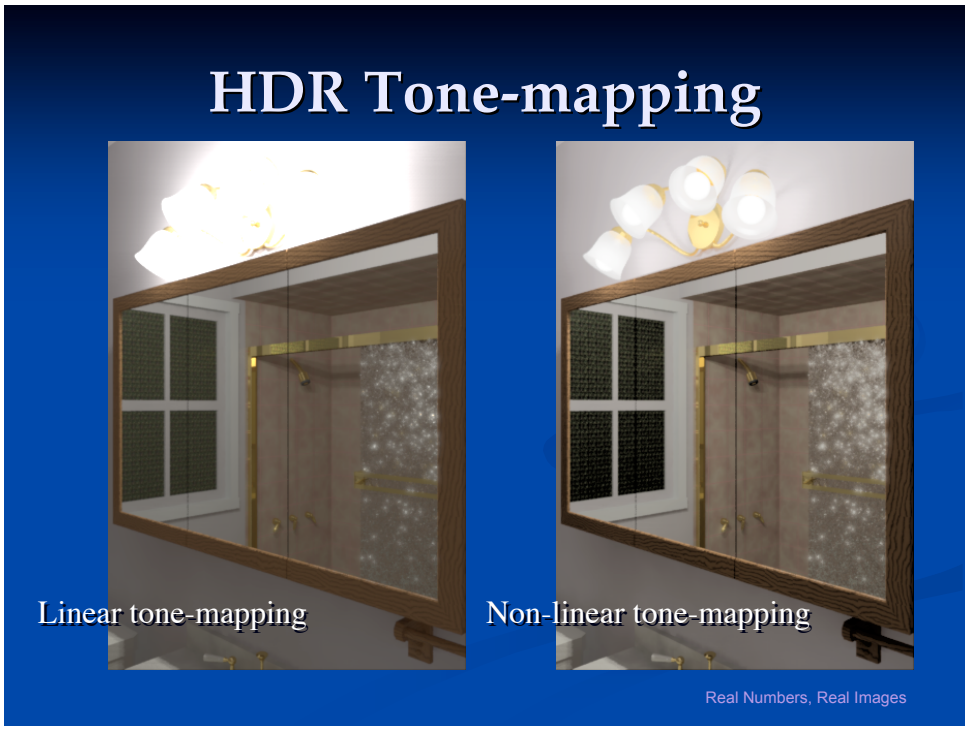
- Render/Capture floating-point color space
- Store entire perceivable gamut (at least)
- Post-process in extended color space
- Apply tone-mapping for specific display
- HDR used extensively at ILM, Digital Domain, ESC, Rhythm & Hues

Real Numbers, Real Images

HDR Imaging Is Not New

- B&W negative film holds at least 4 orders of magnitude
- Much of the talent of photographers like Ansel Adams was darkroom technique
- "Dodge" and "burn" used to bring out the dynamic range of the scene on paper
- The digital darkroom provides new challenges and opportunities

Real Numbers, Real Images

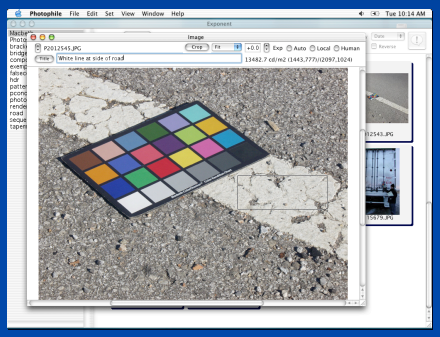

HDR Tone-mapping
Linear tone-mapping
Non-linear tone-mapping
Real Numbers, Real Images


Post-production Possibilities
Simulated glare
Low vision
Real Numbers, Real Images

II. Measurement

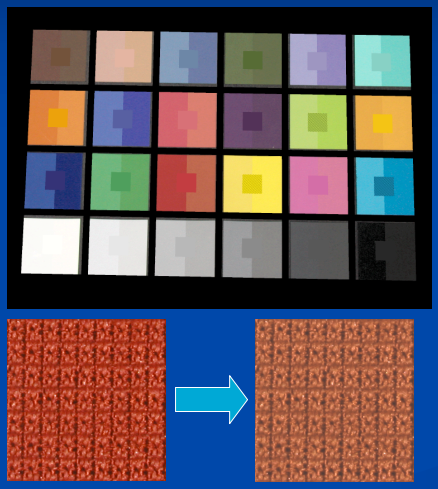
- How do we obtain surface reflectances?
- How do we obtain surface textures (and milli geometry)?
- How do we obtain light source distributions?
- What is the best color space to work in?

Macbeth ColorChecker™ Chart



- Digital photo with ColorChecker™ under uniform illumination
- Compare points on image and interpolate
- Best to work with HDR image
- Accurate to ~ 10 ΔE

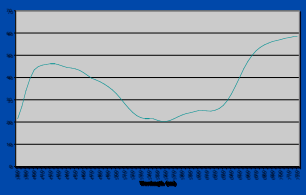
Radiance macbethcal Program



- Computes grayscale function and 3x3 color transform
- Maintain the same measurement conditions
- Calibrated pattern or uniform color capture
- Accurate to ~ 6 ΔE

Real Numbers, Real Images

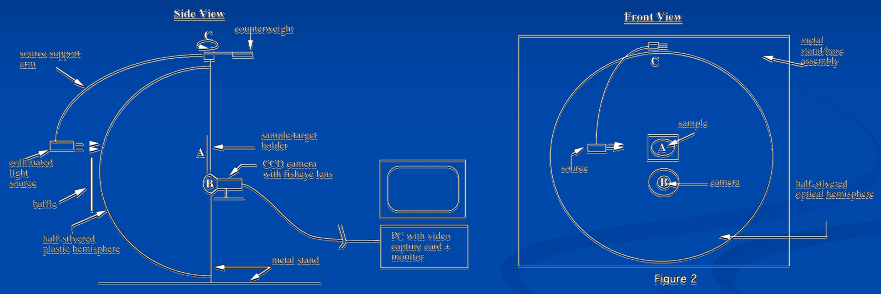
Spectrophotometer



- Commercial spectrophotometers run about $5K US
- Measure reflectance spectrum for simulation under any light source
- Accurate to ~ 2 ΔE

Real Numbers, Real Images

BRDF Capture 1



The LBL imaging gonioreflectometer [Siggraph '92] captures reflected directions at each incident direction using CCD camera

Real Numbers, Real Images

BRDF Capture 2

BRDF capture on round surfaces [Marschner et al. EGWR '99]

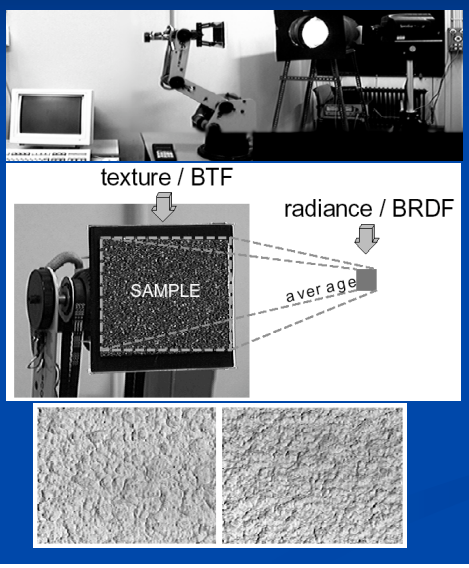
Real Numbers, Real Images

Combined Capture Method 1

- Pietà Project
  www.research.ibm.com/pieta
  [Rushmeier et al. EGWR '98]
- Multi-baseline stereo camera with 5 lights
- Captured geometry and reflectance
- Sub-millimeter accuracy

Real Numbers, Real Images

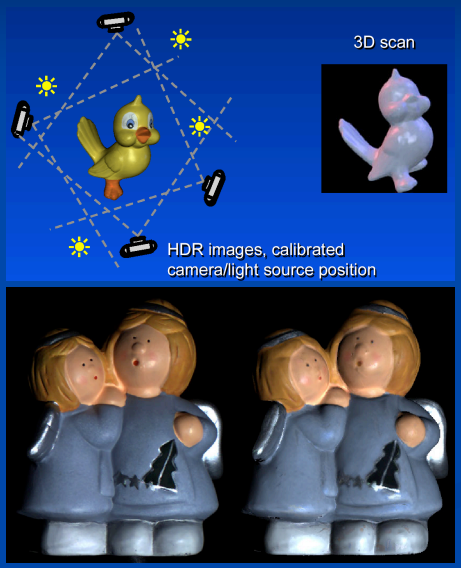
Combined Capture Method 2



- CURET database
  - www1.cs.columbia.edu/CAVE/curet/
  - [Dana et al. TOG '99]
- Capture BTF (bidirectional texture function)
- Interpolate BTF during rendering

Real Numbers, Real Images

Combined Capture Method 3



- Lumitexel capture
  - [Lensch et al. EGWR '01]
- Capture 3-D position + normal + color as function of source position
- Fit data locally to BRDF model
- Render from BRDF

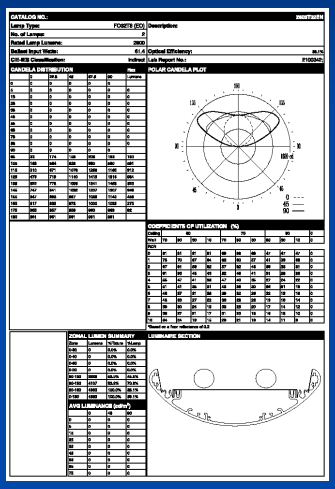
Real Numbers, Real Images

Light Source Distributions

- Often ignored, light source distributions are the first order of lighting simulation
- Data is comparatively easy to obtain
  - Luminaire manufacturers provide data files
    - See www.ledalite.com/resources/software
  - American and European standard file formats
  - Hardcopy photometric reports also available

Real Numbers, Real Images

Luminaire Data



- Photometric reports contain candela information per output direction
- All photometric measurements assume a far–field condition
- Interpolate directions and assume uniform over area

Real Numbers, Real Images

Candela Conversion

- A candela equals one lumen/steradian
- A lumen is approximately equal to 0.0056 watts of equal-energy white light
- To render in radiance units of watts/sr-m^2
  - Multiply candelas by 0.0056/d**A** where d**A** is projected area in each output direction in m^2

Real Numbers, Real Images

What Color Space to Use?

1) How Does *RGB* Rendering Work and When Does It Not?
2) Can *RGB* Accuracy Be Improved?
3) Useful Observations
4) Spectral Prefiltering
5) The von Kries White Point Transform
6) Experimental comparison of 3 spaces

Real Numbers, Real Images

A Brief Comparison of Color Rendering Techniques

- Spectral Rendering
  - ✓ N spectrally pure samples
- Component Rendering
  - ✓ M vector basis functions
- *RGB* (Tristimulus) Rendering
  - ✓ Tristimulus value calculations

Real Numbers, Real Images

Spectral Rendering

1. Divide visible spectrum into N wavelength samples
2. Process spectral samples separately throughout rendering calculation
3. Compute final display color using CIE color matching functions and standard transformations

Real Numbers, Real Images

Component Rendering

[Peercy, Siggraph '93]

1. Divide visible spectrum into M vector bases using component analysis
2. Process colors using MxM matrix multiplication at each interaction
3. Compute final display color with 3xM matrix transform

Real Numbers, Real Images

RGB (Tristimulus) Rendering

1. Precompute tristimulus values
2. Process 3 samples separately throughout rendering calculation
3. Compute final display color with 3x3 matrix transform (if necessary)

Real Numbers, Real Images

Rendering Cost Comparison

	Pre-processing	Multiplies / Interaction	Post-processing
Spectral	None	N (N ≥ 9)	N multiplies per pixel
Component	Vector analysis	MxM (M ≥ 3)	3×M per pixel
RGB	Little or none	3	0 to 9 per pixel

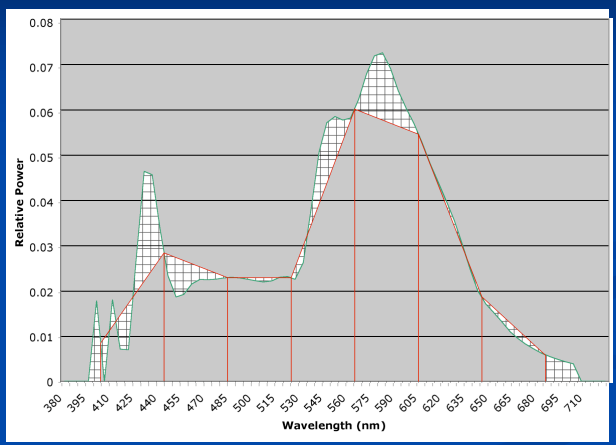
Real Numbers, Real Images

Strengths and Weaknesses

	Strengths	Weaknesses
Spectral	Potential accuracy	Cost, aliasing, data mixing
Component	Optimizes cost/benefit	Preprocessing requirements
RGB	Fast, widely supported	Limited accuracy

Real Numbers, Real Images

Spectral Aliasing



[Meyer88] suffers worse with only 4 samples

Real Numbers, Real Images

The Data Mixing Problem

- Typical situation:
  - Illuminants known to 5 nm resolution
  - Some reflectances known to 10 nm
  - Other reflectances given as tristimulus
- Two alternatives:
  A. Reduce all spectra to lowest resolution
  B. Interpolate/synthesize spectra [Smits '99]

Real Numbers, Real Images

Status Quo Rendering

- White Light Sources
  - E.g., (R,G,B)=(1,1,1)
- *RGB* material colors obtained by dubious means
  - E.g., "That looks pretty good."
    - ✓ This actually works for fictional scenes!
- Color correction with ICC profile if at all

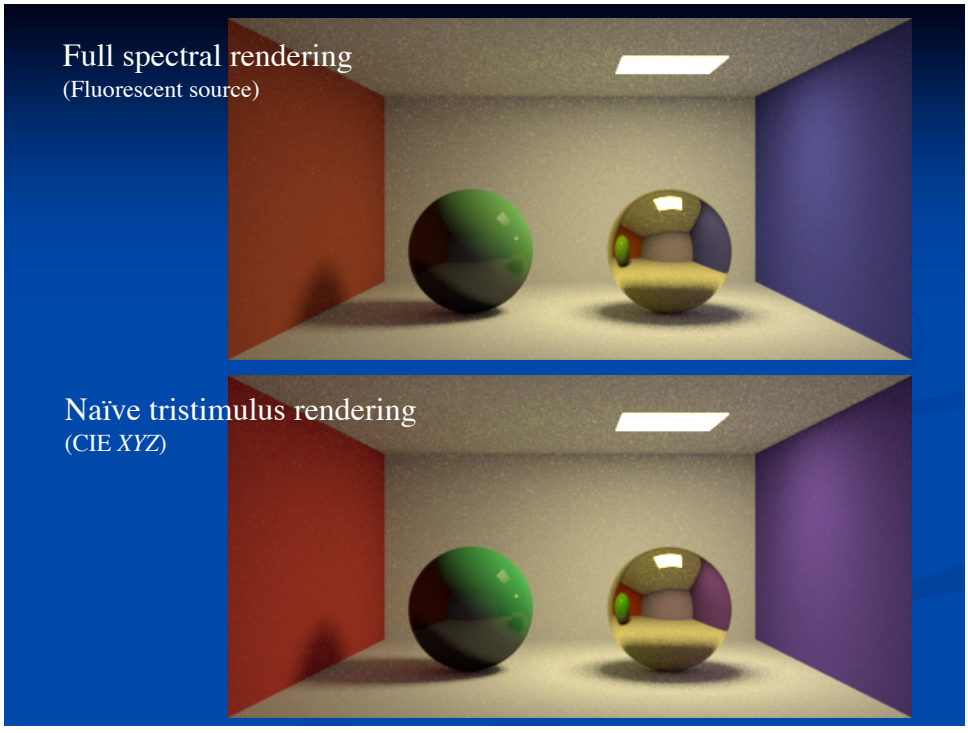
Real Numbers, Real Images

When Does RGB Rendering Normally Fail?

- When you start with measured colors
- When you want to simulate color appearance under another illuminant
- When your illuminant and surface spectra have sharp peaks and valleys

The Result: Wrong COLORS!

Real Numbers, Real Images



Can *RGB* Accuracy Be Improved?

- Identify and minimize sources of error
  - Source-surface interactions
  - Choice of rendering primaries
- Overcome ignorance and inertia
  - Many people render in *RGB* without really understanding what it means
  - White-balance problem scares casual users away from colored illuminants

Real Numbers, Real Images

A Few Useful Observations

a) Direct illumination is the first order in any rendering calculation

b) Most scenes contain a single, dominant illuminant spectrum

c) Scenes with mixed illuminants will have a color cast regardless

Conclusion: Optimize for the Direct Diffuse Case

Real Numbers, Real Images

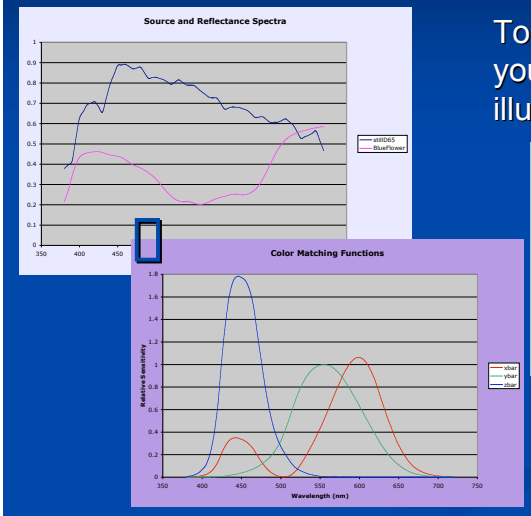
Picture Perfect *RGB* Rendering

1. Identify dominant illuminant spectrum
   a) Prefilter material spectra to obtain tristimulus colors for rendering
   b) Adjust source colors appropriately
2. Perform tristimulus (*RGB*) rendering
3. Apply white balance transform and convert pixels to display color space

From [Ward & Eydelberg-Vileshin EGWR '02]

Real Numbers, Real Images

Spectral Prefiltering



To obtain a tristimulus color, you *must* know the illuminant spectrum

$$X = \int I(\lambda)\rho(\lambda)\bar{x}(\lambda)\,d\lambda$$
$$Y = \int I(\lambda)\rho(\lambda)\bar{y}(\lambda)\,d\lambda$$
$$Z = \int I(\lambda)\rho(\lambda)\bar{z}(\lambda)\,d\lambda$$

XYZ may then be transformed by 3×3 matrix to any linear tristimulus space (e.g., *sRGB*)

Real Numbers, Real Images

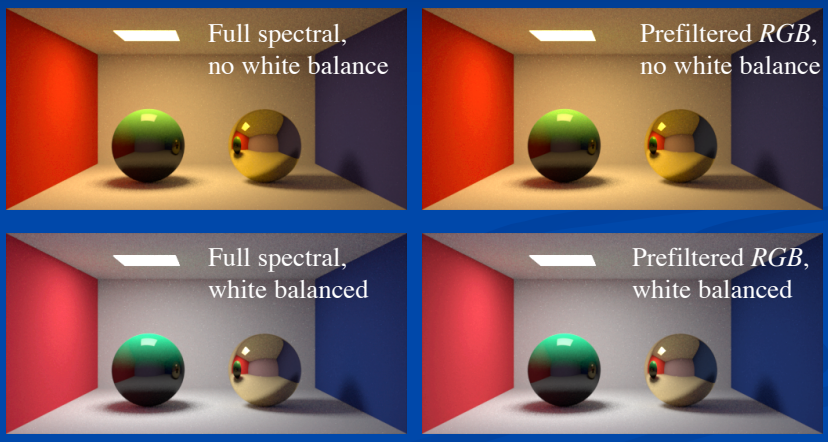
Prefiltering vs. Full Spectral Rendering

+ Prefiltering performed once per material vs. every rendering interaction
+ Spectral aliasing and data mixing problems disappear with prefiltering
- However, mixed illuminants and interreflections not computed exactly

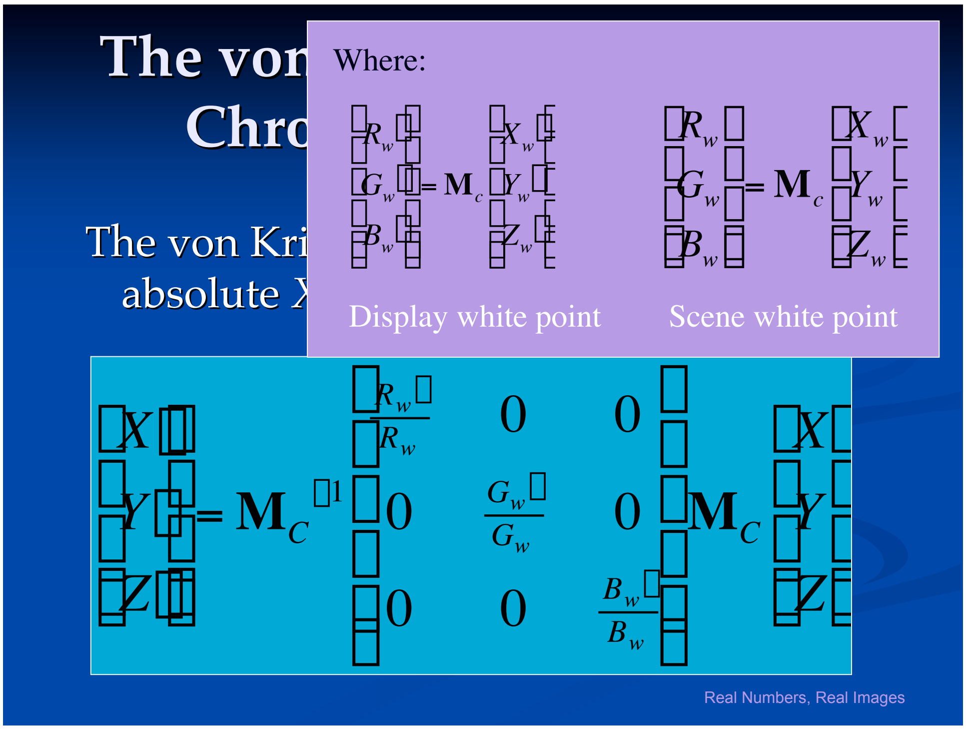
Regardless which technique you use, remember to apply white balance to result!

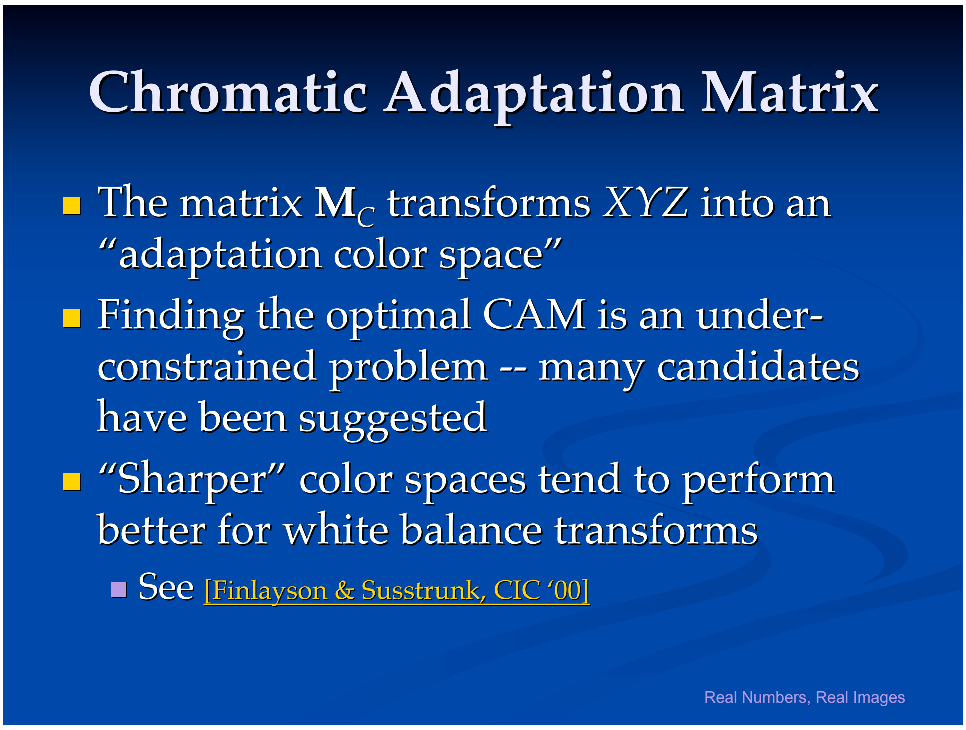
Real Numbers, Real Images

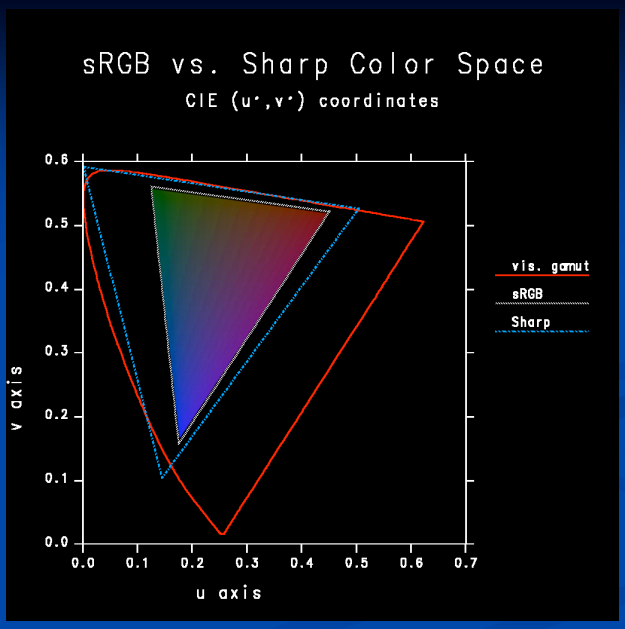
Quick Comparison



Real Numbers, Real Images


The von
Chro
The von Kri
absolute X
Where:
$\begin{bmatrix} R'_w \\ G'_w \\ B'_w \end{bmatrix} = \mathbf{M}_c \begin{bmatrix} X'_w \\ Y'_w \\ Z'_w \end{bmatrix}$
$\begin{bmatrix} R_w \\ G_w \\ B_w \end{bmatrix} = \mathbf{M}_c \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}$
Display white point
Scene white point
$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \mathbf{M}_C^{-1} \begin{bmatrix} \frac{R'_w}{R_w} & 0 & 0 \\ 0 & \frac{G'_w}{G_w} & 0 \\ 0 & 0 & \frac{B'_w}{B_w} \end{bmatrix} \mathbf{M}_C \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$
Real Numbers, Real Images


Chromatic Adaptation Matrix
The matrix $\mathbf{M}_C$ transforms *XYZ* into an "adaptation color space"
Finding the optimal CAM is an under-constrained problem -- many candidates have been suggested
"Sharper" color spaces tend to perform better for white balance transforms
See [Finlayson & Susstrunk, CIC '00]
Real Numbers, Real Images



Real Numbers, Real Images

Three Tristimulus Spaces for Color Rendering

- CIE *XYZ*
  - Covers visible gamut with positive values
  - Well-tested standard for color-matching
- *sRGB*
  - Common standard for image encoding
  - Matches typical CRT display primaries
- *Sharp RGB*
  - Developed for chromatic adaptation

Real Numbers, Real Images

XYZ Rendering Process

1. Apply prefiltering equation to get absolute *XYZ* colors for each material
   a) Divide materials by illuminant:

$$X_m^* = \frac{X_m}{X_w}, \quad Y_m^* = \frac{Y_m}{Y_w}, \quad Z_m^* = \frac{Z_m}{Z_w}$$

   b) Use absolute *XYZ* colors for sources
2. Render using tristimulus method
3. Finish w/ CAM and display conversion

Real Numbers, Real Images

sRGB Rendering Process

1. Perform prefiltering and von Kries transform on material colors
   a) Model dominant light sources as neutral
   b) For spectrally distinct light sources use:

$$R_s^* = \frac{R_s}{R_w}, \quad G_s^* = \frac{G_s}{G_w}, \quad B_s^* = \frac{B_s}{B_w}$$

2. Render using tristimulus method
3. Resultant image is *sRGB*

Real Numbers, Real Images

Sharp RGB Rendering Process

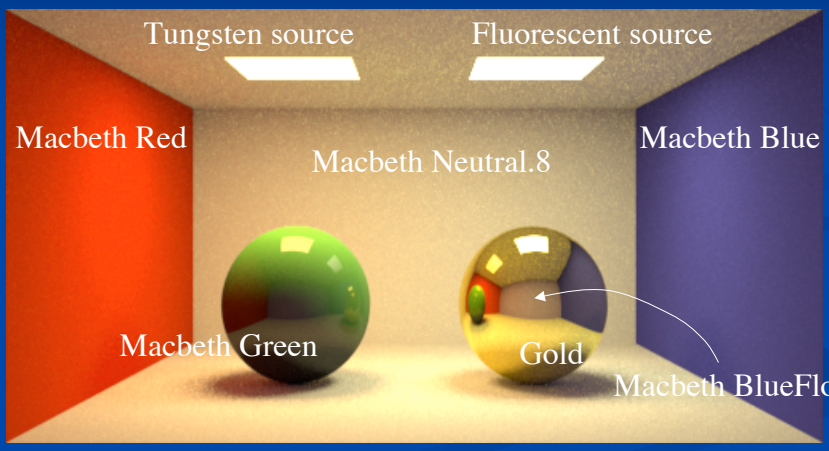
1. Prefilter material colors and apply von Kries transform to *Sharp RGB* space:

$$\begin{bmatrix} R_m^* \\ G_m^* \\ B_m^* \end{bmatrix} = \begin{bmatrix} \frac{1}{R_w} & 0 & 0 \\ 0 & \frac{1}{G_w} & 0 \\ 0 & 0 & \frac{1}{B_w} \end{bmatrix} \mathbf{M}_{Sharp} \begin{bmatrix} X_m \\ Y_m \\ Z_m \end{bmatrix}$$

2. Render using tristimulus method
3. Finish up CAM and convert to display

Real Numbers, Real Images

Our Experimental Test Scene



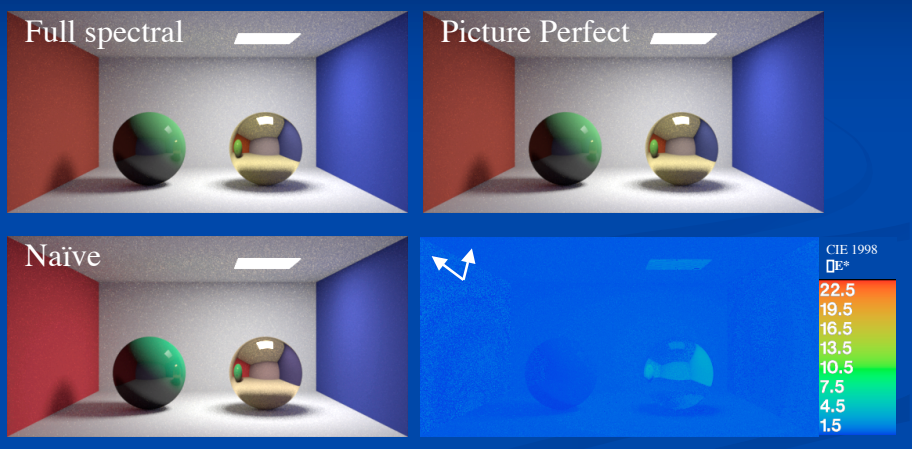
Real Numbers, Real Images

Experimental Results

- Three lighting conditions
  - Single 2856°K tungsten light source
  - Single cool white fluorescent light source
  - Both light sources (tungsten & fluorescent)
- Three rendering methods
  - Naïve *RGB* (assumes equal-energy white)
  - Picture Perfect *RGB*
  - Full spectral rendering (380 to 720 nm / 69 samp.)
- Three color spaces (*XYZ, sRGB, Sharp RGB*)

Real Numbers, Real Images

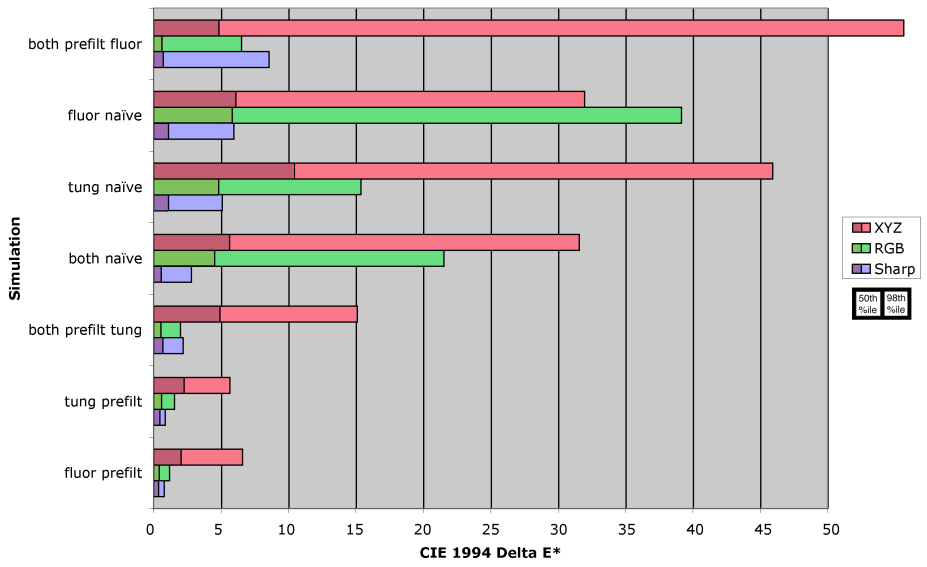
Example Comparison (*sRGB*)



CIE 1998 ΔE* of 5 or above is visible in side-by-side comparisons

Real Numbers, Real Images

ΔE* Error Percentiles for All Experiments



Results Summary

- Prefiltering has ~1/6 the error of naïve rendering for single dominant illuminant
- Prefiltering errors similar to naïve in scenes with strongly mixed illuminants
- CIE *XYZ* color space has 3 times the rendering errors of *sRGB* on average
- *Sharp RGB* rendering space reduces errors to 1/3 that of *sRGB* on average

Real Numbers, Real Images

III. Lighting Simulation

- Approximating local illumination
- Approximating global illumination
- Dealing with motion
- Exploiting human perception to accelerate rendering

Local Illumination

- Local illumination is the most important part of rendering, and *everyone* gets it wrong (including me)
- Real light-surface interactions are incredibly complex, and humans have evolved to perceive many subtleties
- The better your local illumination models, the more realistic your renderings

LI Advice: Use Physical Range

- Non-metallic surfaces rarely have specular reflectances greater than 7%
  - Determined by the index of refraction, $n < 1.7$
- Physically plausible BRDF models obey energy conservation and reciprocity
  - Phong model often reflects > 100% of incident
- *RGB* reflectances may be slightly out of [0,1] range for highly saturated colors

Real Numbers, Real Images

LI Advice: Add Fresnel Factor

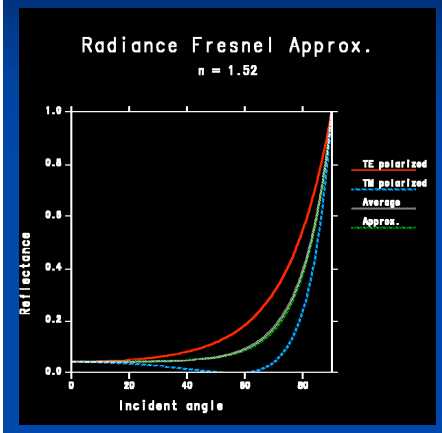
- Specular reflectance goes up near grazing for all polished materials – here is a good approximation for Fresnel reflection:

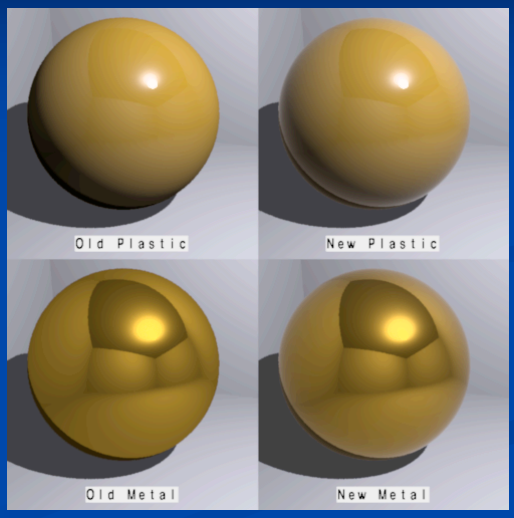
$$\rho_\theta \approx \rho_0 + (1-\rho_0)\left[\exp(-6\cos\theta) - \exp(-6)\right]$$

- Simpler & faster than standard formula
- Improves accuracy and appearance at silhouettes

Real Numbers, Real Images

Fresnel Approximation





LI Advice: Texture Carefully

- Pay attention to exactly how your image textures affect your average and peak reflectances
  - Are they still in a physically valid range?
- Use bump maps sparingly
  - Odd artifacts arise when geometry and surface normals disagree strongly
  - Displacement maps are better

LI Advice: Use BTF Model

- Use CURET data to model view-dependent appearance under different lighting using *TensorTexture* technique
  - See "TensorTextures", M. Alex O. Vasilescu and D. Terzopoulos, Sketch and Applications SIGGRAPH 2003 San Diego, CA, July, 2003.
    www.cs.toronto.edu/~maov/tensortextures/tensortextures_sigg03.pdf

Global Illumination

- Global illumination will not fix problems caused by poor local illumination, but…
  - GI adds another dimension to realism, and
  - GI gets you absolute answers for lighting
- Radiosity methods compute form factors
  - Says nothing about global illumination
- Ray-tracing methods intersect rays
  - Again, this is not a useful distinction

Real Numbers, Real Images

GI Algorithm Characteristics

- Traces rays
- Subdivides surfaces into quadrilaterals
- Employs form factor matrix
- Deposits information on surfaces
  - Using grid
  - Using auxiliary data structure (e.g., octree)
- Requires multiple passes

Real Numbers, Real Images

GI Example 1: Hemicube Radiosity [Cohen et al. '86]

- × Traces rays
- ✓ Subdivides surfaces into quadrilaterals
- ✓ Employs form factor matrix
- ✓ Deposits information on surfaces
  - ✓ Using grid
  - × Using auxiliary data structure (e.g., octree)
- ✓ Requires multiple passes

Real Numbers, Real Images

GI Example 2: Particle Tracing
[Shirley et al. '95]

- ✓ Traces rays
- × Subdivides surfaces into quadrilaterals
  - ✓ But triangles, yes
- × Employs form factor matrix
- ✓ Deposits information on surfaces
  - × Using grid
  - ✓ Using auxiliary data structure (T-mesh)
- ✓ Requires multiple passes

Real Numbers, Real Images

GI Example 3: Monte Carlo Path Tracing [Kajiya '86]

- ✓ Traces rays
- × Subdivides surfaces into quadrilaterals
- × Employs form factor matrix
- × Deposits information on surfaces
- × Requires multiple passes

Real Numbers, Real Images

GI Example 4: *Radiance*

- ✓ Traces rays
- × Subdivides surfaces into quadrilaterals
- × Employs form factor matrix
- ✓ Deposits information on surfaces
  - × Using grid
  - ✓ Using auxiliary data structure (octree)
- × Requires multiple passes

Real Numbers, Real Images

Scanned Photograph

Radiance Rendering

The Rendering Equation

Radiation Transport:

$$R_o(\theta_o,\phi) = \int\int f_r(\theta_o;\theta_i,\phi)\, R_i(\theta_i,\phi)\cos\theta_i\, d\theta_i \quad (1)$$

Participating Medium:

$$\frac{dR(s)}{ds} = \sigma_a R(s) \sigma_s R(s) + \frac{\sigma_s}{4\pi}\int R_i(\theta_i) P(\theta_i)\, d\theta \quad (2)$$

Real Numbers, Real Images

Radiance Calculation Methods

$$R_o(\theta_o,\phi) = \int\int f_r(\theta_o;\theta_i,\phi)\, R_i(\theta_i,\phi)\cos\theta_i\, d\theta_i \quad (1)$$

- Direct calculation removes large incident
- Indirect calculation handles most of the rest
- Secondary light sources for problem areas
- Participating media (adjunct to equation)

Real Numbers, Real Images

Radiance Direct Calculation

- Selective Shadow Testing
  - Only test significant sources
- Adaptive Source Subdivision
  - Subdivide large or long sources
- Virtual Light Source Calculation
  - Create virtual sources for beam redirection

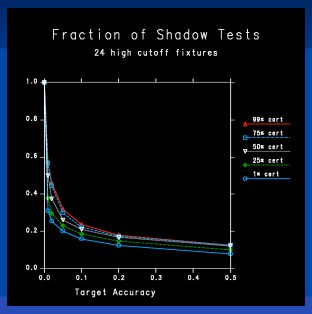
Real Numbers, Real Images

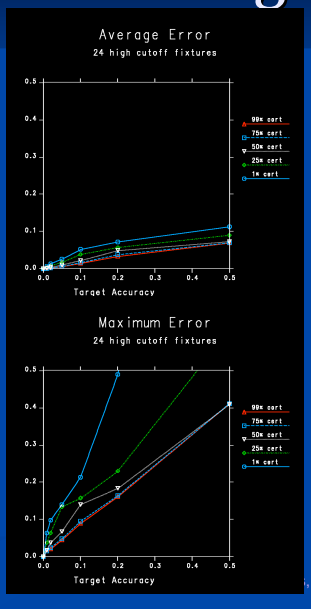
Selective Shadow Testing

- Sort potential direct contributions
  - Depends on sources and material
- Test shadows from most to least significant
  - Stop when remainder is below error tolerance
- Add in untested remainder
  - Use statistics to estimate visibility

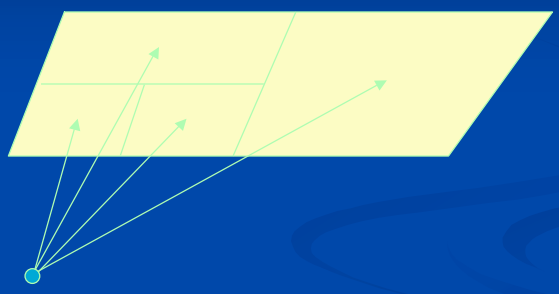
Real Numbers, Real Images

Selective Shadow Testing (2)



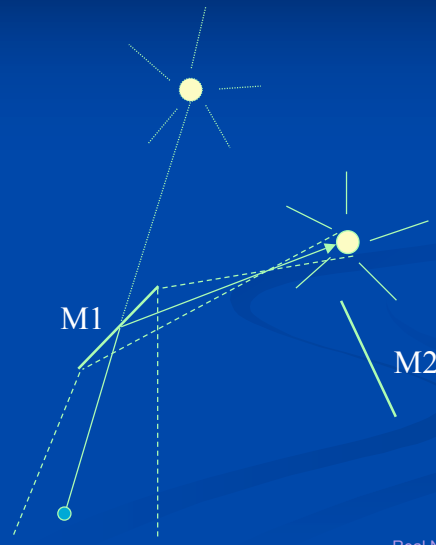


Adaptive Source Subdivision



Subdivide source until width/distance less than max. ratio

Virtual Light Source Calculation



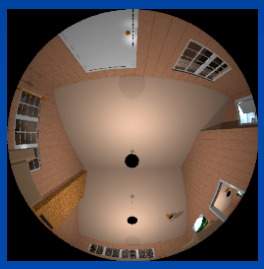
Real Numbers, Real Images

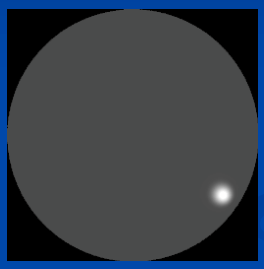
Indirect Calculation

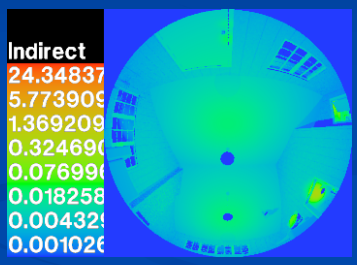
- Specular Sampling
  - sample rays over scattering distribution
- Indirect Irradiance Caching
  - sample rays over hemisphere
  - cache irradiance values over geometry
  - reuse for other views and runs

Real Numbers, Real Images

Indirect Calculation (2)





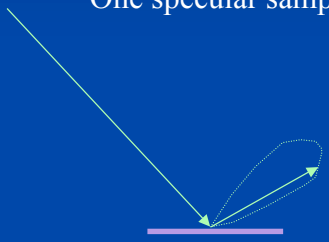


Indirect x BRDF =

Real Numbers, Real Images

Specular Sampling

One specular sample per pixel

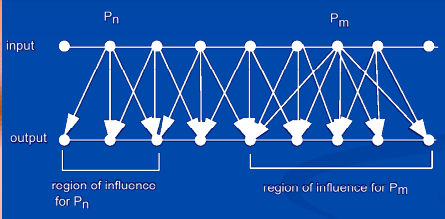


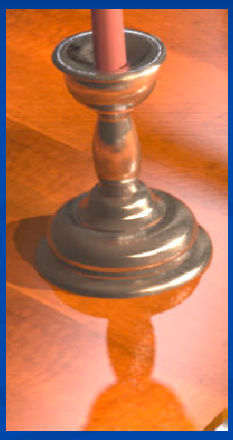
Filtering reduces artifacts

Real Numbers, Real Images

Energy-preserving Non-linear Filters

From [Rushmeier & Ward, Siggraph '94]

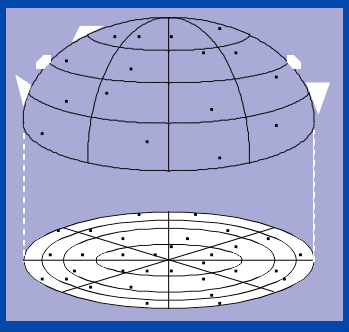


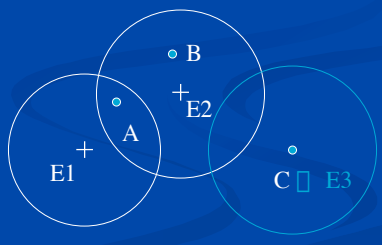


Real Numbers, Real Images

Indirect Irradiance Caching

Indirect irradiance is computed and interpolated using octree lookup scheme





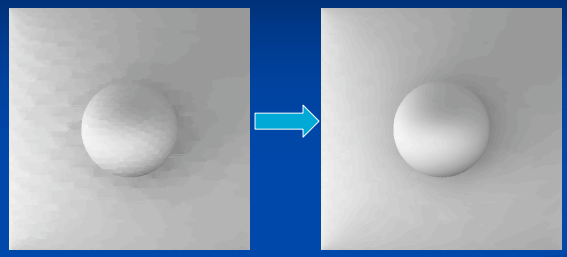
Real Numbers, Real Images

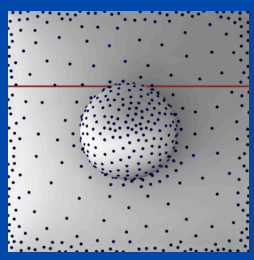
Indirect Irradiance Gradients

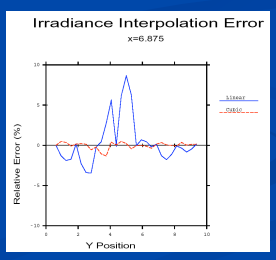
- From hemisphere sampling, we can also compute change w.r.t. position and direction
- Effectively introduces higher-order interpolation method, i.e., cubic vs. linear
- See [Ward & Heckbert, EGWR '92] for details

Real Numbers, Real Images

Irradiance Gradients (2)







Real Numbers, Real Images

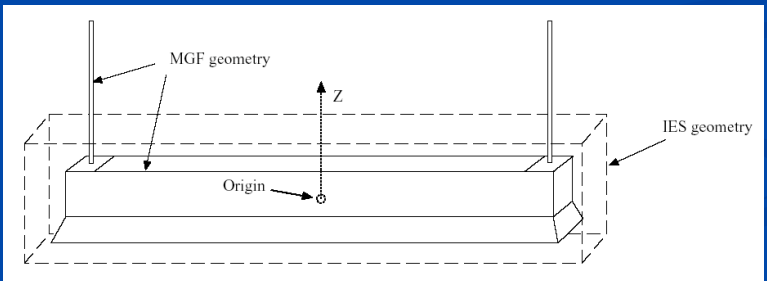
Secondary Light Sources

- Impostor surfaces around sources
  - decorative luminaires
  - clear windows
  - complex fenestration
- Computing secondary distributions
  - the **mkillum** program

Real Numbers, Real Images

Impostor Source Geometry

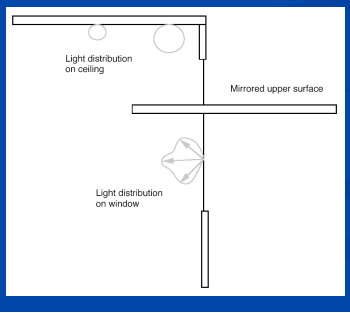
- Simplified geometry for shadow testing and illumination computation
  - fits snugly around real geometry, which is left for rendering direct views



Real Numbers, Real Images

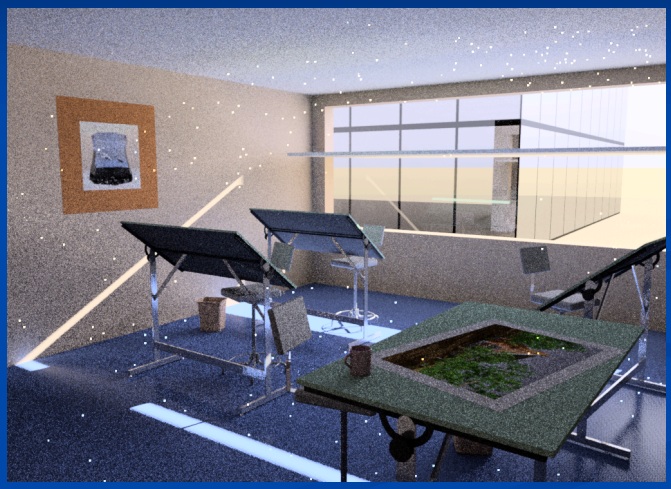
Computing Secondary Distributions

- Start with straight scene description
- Use **mkillum** to compute secondary sources
- Result is a more efficient calculation



Real Numbers, Real Images

Using Pure Monte Carlo



Real Numbers, Real Images

Using Secondary Sources

Real Numbers, Real Images

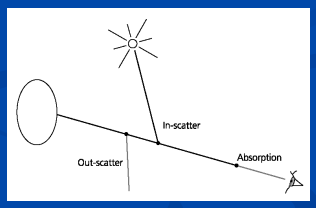
Participating Media

- Single-scatter approximation
- The mist material type
  - light beams
  - constant density regions
- Rendering method

Real Numbers, Real Images

Single-scatter Approximation

- Computes light scattered into path directly from specified light sources
- Includes absorption and ambient scattering

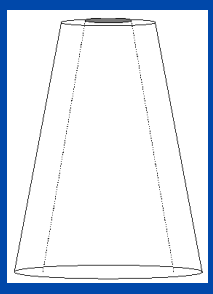


$$\frac{dR(s)}{ds} = \sigma_a R(s) \sigma_s R(s) + \frac{\sigma_s}{4\pi} \int R_i(\omega_i) P(\omega_i) d\omega \quad (2)$$

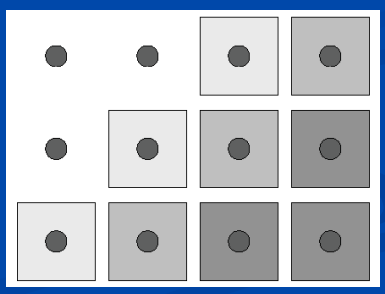
Real Numbers, Real Images

The *Mist* Material Type

- Demark volumes for light beams
- Can change medium density or scattering properties within a volume



Spotlight with enclosing *mist* volume

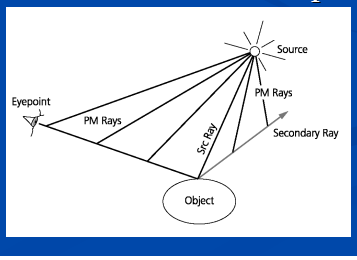


Mist volumes with different densities

Real Numbers, Real Images

Rendering Method

- After standard ray value is computed:
  - compute ambient in-scattering, out-scattering and absorption along ray path
  - compute in-scattering from any sources identified by *mist* volumes ray passes through
    - this step accounts for anisotropic scattering as well



Real Numbers, Real Images

What About Animation?

- Easy: render frames independently
  - What about motion blur?
  - Also, is this the most efficient approach?
- Better: Image-based frame interpolation
  - **Pinterp** program
    - First released in May 1990 (*Radiance* 1.2)
    - Combines pixels with depth for in-between frames
    - Motion-blur capability
    - Moving objects still a problem

Real Numbers, Real Images

Exploit Human Perception

- Video compression community has studied what motions people notice
- In cases where there is an associated task, we can also exploit *inattentional blindness*
- Image-based motion blur can be extended to objects with a little additional work

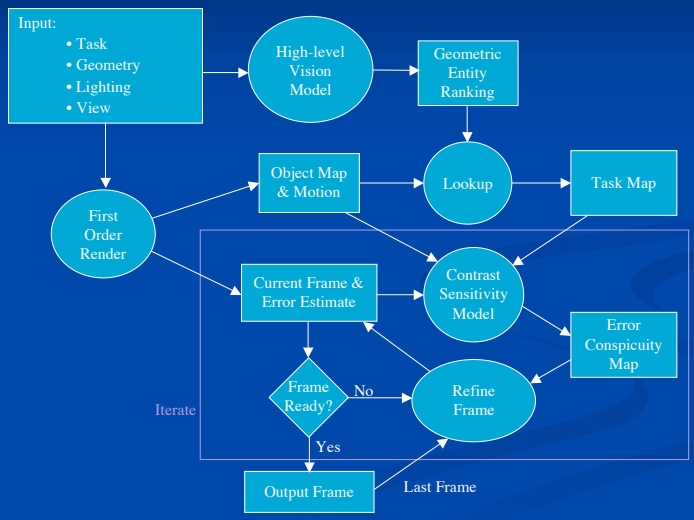
Real Numbers, Real Images

Perceptual Rendering Framework

- "Just in time" animation system
- Exploits inattentional blindness and IBR
- Generalizes to other rendering techniques
  - Demonstration system uses *Radiance* ray-tracer
  - Potential for real-time applications
- Error visibility tied to attention and motion

Real Numbers, Real Images

Rendering Framework



Real Numbers, Real Images

Example Frame w/ Task Objects

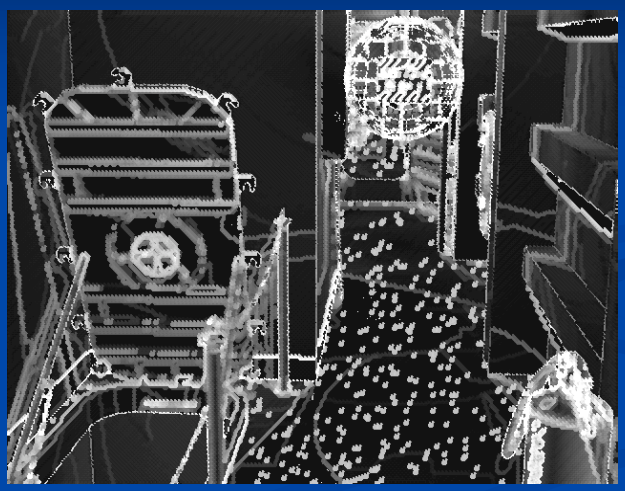
Real Numbers, Real Images

Error Map Estimation

- Stochastic errors may be estimated from neighborhood samples
- Systematic error bounds may be estimated from knowledge of algorithm behavior
- Estimate accuracy is not critical for good performance

Real Numbers, Real Images

Initial Error Estimate



Real Numbers, Real Images

Image-based Refinement Pass

- Since we know exact motion, IBR works very well in this framework
- Select image values from previous frame
  - Criteria include coherence, accuracy, agreement
- Replace current sample and degrade error
  - Error degradation results in sample retirement

Real Numbers, Real Images

Contrast Sensitivity Model

Additional samples are directed based on Daly's CSF model:

$$CSF(\rho, v_R) = k \cdot c_0 \cdot c_2 \cdot v_R \cdot (c_1 2\pi\rho)^2 \exp\left(\frac{c_1 4\pi\rho}{\rho_{\max}}\right)$$

where:

ρ is spatial frequency

v_R is retinal velocity

$k = 6.1 + 7.3\left|\log(c_2 v_R / 3)\right|^3$

$\rho_{\max} = 45.9/(c_2 v_R + 2)$

$c_0 = 1.14, \quad c_1 = 0.67, \quad c_2 = 1.7$ for CRT at 100 cd/m2

Real Numbers, Real Images

Error Conspicuity Model

Retinal velocity depends on task-level saliency:

$$v_R = \left| v_I \, \square \min(v_I \cdot S / S_{\max} + v_{\min},\ v_{\max}) \right|$$

where:

v_I = local pixel velocity (from motion map)
S = task-level saliency for this region
$S_{\max}$ = max. saliency in this frame, but not less than 1/0.82
$v_{\min}$ = 0.15°/sec (eye drift velocity)
$v_{\max}$ = 80°/sec (movement-tracking limit)

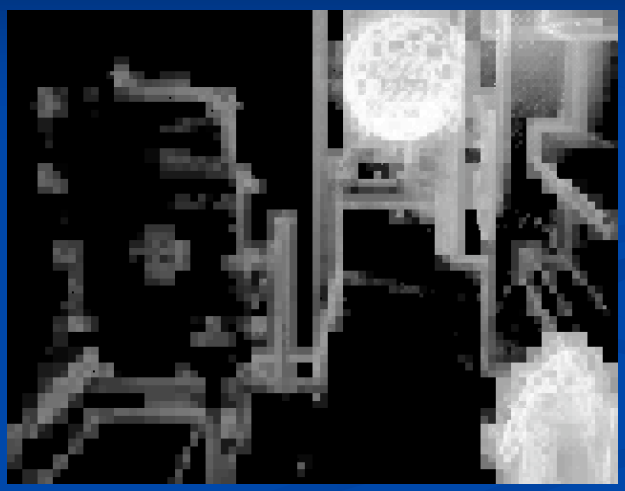
$$EC = S \cdot \max(E \cdot CSF / ND \square 1,\ 0)$$ *Error Conspicuity*

where:

E = relative error estimate for this pixel
ND = noticeable difference threshold

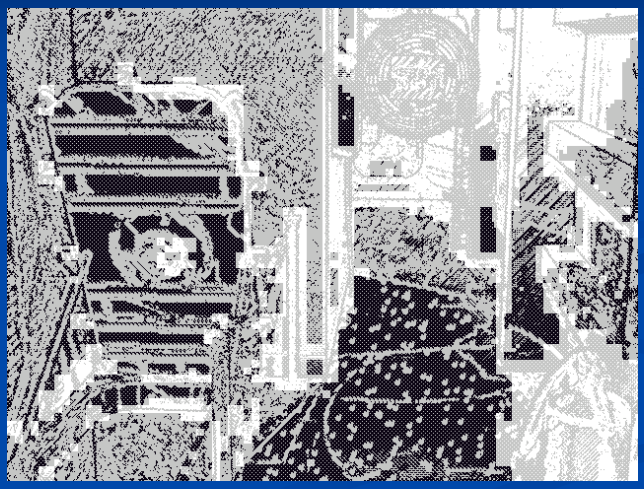
Real Numbers, Real Images

Error Conspicuity Map



Real Numbers, Real Images

Final Sample Density



Real Numbers, Real Images

Implementation Example

- Compared to a standard rendering that finished in the same time, our framework produced better quality on task objects
- Rendering the same high quality over the entire frame would take about 7 times longer using the standard method

Framework rendering

Standard rendering

Real Numbers, Real Images

Example Animation

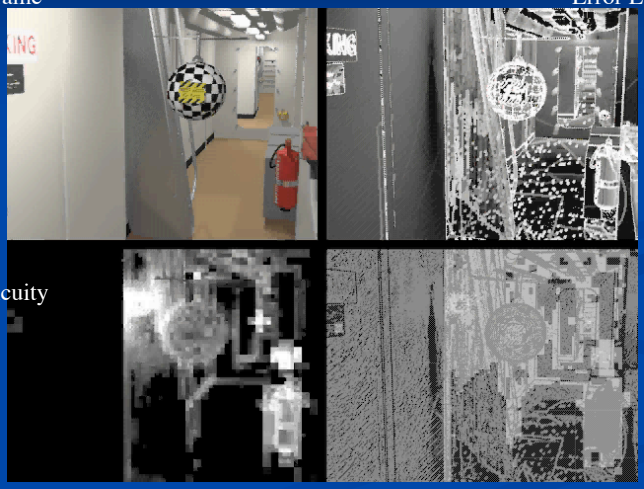
- The following animation was rendered at two minutes per frame on a 2000 model G3 laptop computer (Apple PowerBook)
- Many artifacts are intentionally visible, but less so if you are performing the task

Real Numbers, Real Images

Algorithm Visualization

Finished Frame

Error Estimate



Error Conspicuity

Final Samples

Click to animate

Real Numbers, Real Images

IV. Image Representation

- Traditional graphics image formats
  - Associated problems
- High dynamic-range (HDR) formats
  - Standardization efforts

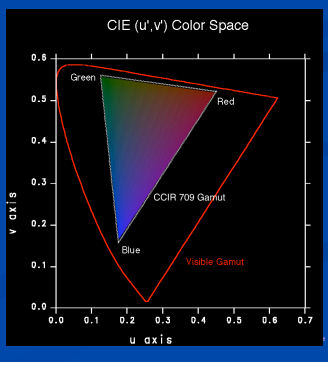
Traditional Graphics Images

- Usually 8-bit integer range per primary

$$L_{out} = L_{\max}\left(i/255\right)^{2.2}$$

Covers about 100:1 range

- *sRGB* color space matches CRT monitors, not human vision



Extended Graphics Formats

- 12 or even 16 bits/primary in TIFF

$$L_{out} = L_{\max}\left(i/65535\right)^{2.2}$$ > 500000:1 range

- Photo editors (i.e., Photoshop™) do not respect this range, treating 65535 as white
- Camera raw formats are an archiving disaster, and should be avoided
- RGB still constrains color gamut

Real Numbers, Real Images

The 24-bit Red Green Blues

- Although 24-bit *sRGB* is reasonably matched to CRT displays, it is a poor match to human vision
  - People can see twice as many colors
  - People can see twice the log range

Q: Why did they base a standard on existing display technology?

A: Because signal processing *used* to be expensive…

Real Numbers, Real Images

High Dynamic Range Images

- High Dynamic Range Images have a wider gamut and contrast than 24-bit RGB
  - Preferably, the gamut and dynamic range covered exceed those of human vision

Advantage 1: an image standard based on human vision won't need frequent updates

Advantage 2: floating point pixels open up a vast new world of image processing

Real Numbers, Real Images

Some HDRI Formats

- *Pixar* 33-bit log-encoded TIFF
- *Radiance* 32-bit RGBE and XYZE
- IEEE 96-bit TIFF & Portable FloatMap
- LogLuv TIFF (24-bit and 32-bit)
- *ILM* 48-bit OpenEXR format

Real Numbers, Real Images

Pixar Log TIFF Codec

Purpose: To store film recorder input

- Implemented in Sam Leffler's TIFF library
- 11 bits each of log red, green, and blue
- 3.8 orders of magnitude in 0.4% steps
- ZIP lossless entropy compression
- Does not cover visible gamut
- Dynamic range marginal for image processing

Real Numbers, Real Images

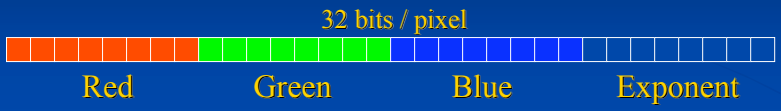
Radiance RGBE & XYZE

Purpose: To store GI renderings

- Simple format with free source code
- 8 bits each for 3 mantissas + 1 exponent
- 76 orders of magnitude in 1% steps
- Run-length encoding (20% avg. compr.)
- RGBE format does not cover visible gamut
- Color quantization not perceptually uniform
- Dynamic range at expense of accuracy

Real Numbers, Real Images

Radiance Format (.pic, .hdr)



(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

Real Numbers, Real Images

IEEE 96-bit TIFF

Purpose: To minimize translation errors

- Most accurate representation
- Files are enormous
  - 32-bit IEEE floats do not compress well

Real Numbers, Real Images

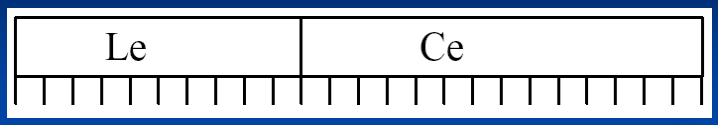
24-bit LogLuv TIFF Codec

Purpose: To match human vision in 24 bits

- Implemented in Leffler's TIFF library
- 10-bit LogL + 14-bit CIE (u',v') lookup
- 4.8 orders of magnitude in 1.1% steps
- Just covers visible gamut and range
- No compression

Real Numbers, Real Images

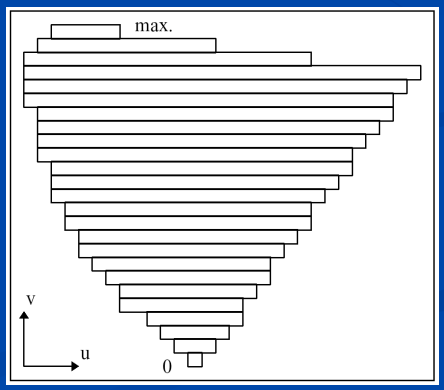
24 -bit LogLuv Pixel



$$L_e = \lfloor 64(\log_2 L + 12) \rfloor$$

$$u' = \frac{4x}{-2x + 12y + 3}$$

$$v' = \frac{9y}{-2x + 12y + 3}$$



Real Numbers, Real Images

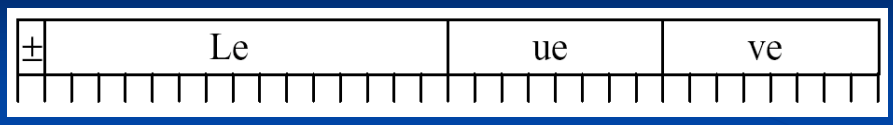
32-bit LogLuv TIFF Codec

Purpose: To surpass human vision

- Implemented in Leffler's TIFF library
- 16-bit LogL + 8 bits each for CIE (u',v')
- 38 orders of magnitude in 0.3% steps
- Run-length encoding (30% avg. compr.)
- Allows negative luminance values

Real Numbers, Real Images

32-bit LogLuv Pixel



$$L_e = \lfloor 256(\log_2 L + 64) \rfloor$$

$$u_e = \lfloor 410u' \rfloor$$

$$v_e = \lfloor 410v' \rfloor$$

$$u' = \frac{4x}{-2x+12y+3}$$

$$v' = \frac{9y}{-2x+12y+3}$$

Described along with 24-bit LogLuv in [Larson CIC '98]

Real Numbers, Real Images

ILM OpenEXR Format

Purpose: HDR lighting and compositing

- 16-bit/primary floating point (sign-e5-m10)
- 9.6 orders of magnitude in 0.1% steps
- Wavelet compression of about 40%
- Negative colors and full gamut RGB
- Open Source I/O library released Fall 2002

Real Numbers, Real Images

ILM's OpenEXR (.exr)

6 bytes per pixel, 2 for each channel, compressed

sign exponent mantissa

- 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 www.openexr.net

Real Numbers, Real Images

HDRI Post-production

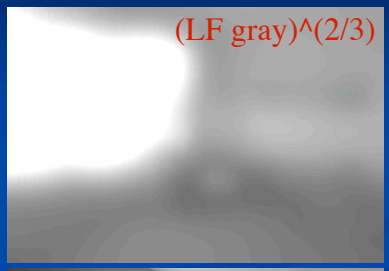
- Operators
  - Contrast & brightness
  - Color balance
  - Low vision
  - Glare
  - Motion blur
  - Lens flare
- Compositing
  - 16-bit log alpha
  - Post-prod. shading?

From [Debevec & Malik, Siggraph '97]

Real Numbers, Real Images

Example HDR Post-processing

÷



=

High dynamic-range + extended gamut = lots of cool tricks

Real Numbers, Real Images

Image Representation Future

- JPEG and other 24-bit formats here to stay
- Lossless HDRI formats for high-end
- Compressed HDRI formats are desirable for digital camera applications
  - JPEG 2000 seems like a possible option
    - Adobe doesn't like its proprietary inception
  - Others pushing for a "standard raw sensor" format, but I doubt it would work

Real Numbers, Real Images

V. Image Display

- How do we display an HDR image?
- There are really just two options:
  1. Tone-map HDRI to fit in displayable range
  2. View on a high dynamic-range display
- Many tone-mapping algorithms have been proposed for dynamic-range compression
- But, there are no HDR displays! (Or are there?)

Real Numbers, Real Images

HDRI Tone-mapping

- Tone-mapping (a.k.a. tone-reproduction) is a well-studied topic in photography
  - Traditional film curves are carefully designed
- Computer imaging offers many new opportunities for dynamic TRC creation
- Additionally, tone reproduction curves may be manipulated locally over an image

Real Numbers, Real Images

Tone-mapping to LDR Display

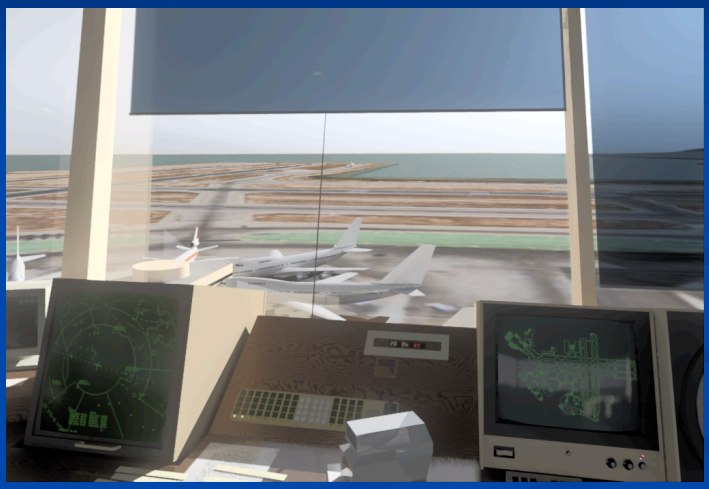
- A renderer is like an "ideal" camera
- TM is medium-specific and goal-specific
- Need to consider:
  - Display gamut, dynamic range, and surround
  - What do we wish to simulate?
    - Cinematic camera and film?
    - Human visual abilities and disabilities?

Real Numbers, Real Images

TM Goal: Colorimetric

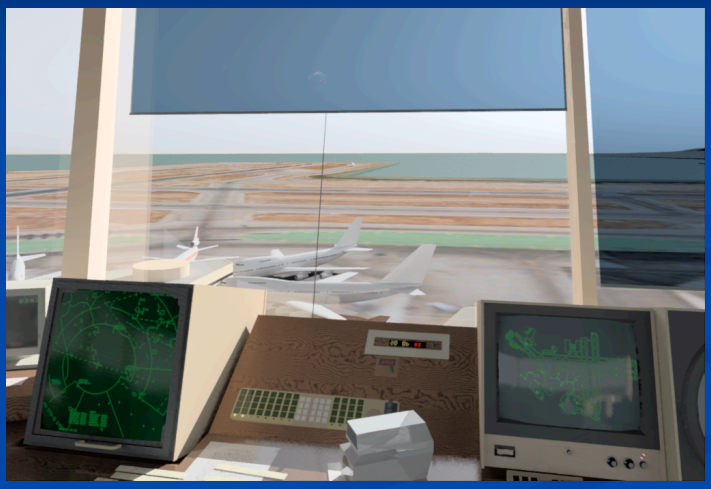
Real Numbers, Real Images

TM Goal: Match Visibility

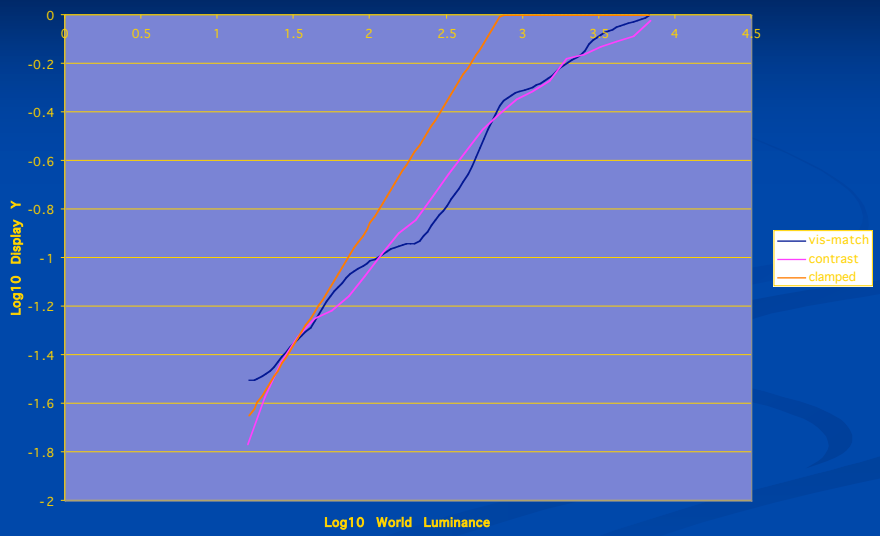


Real Numbers, Real Images

TM Goal: Optimize Contrast



Real Numbers, Real Images



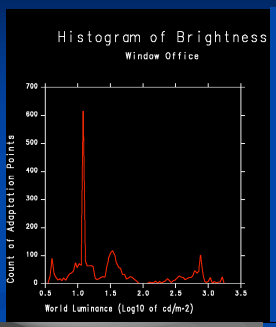
Real Numbers, Real Images

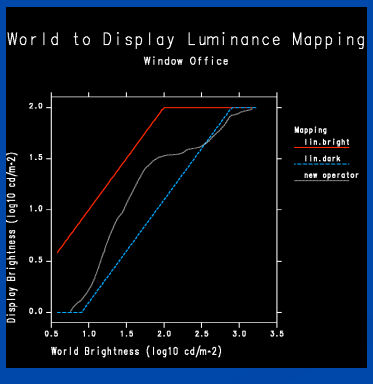
One Tone-mapping Approach

- Generate histogram of log luminance
- Redistribute luminance to fit output range
- Optionally simulate human visibility
  - match contrast sensitivity
  - scotopic and mesopic color sensitivity
  - disability (veiling) glare
  - loss of visual acuity in dim environments

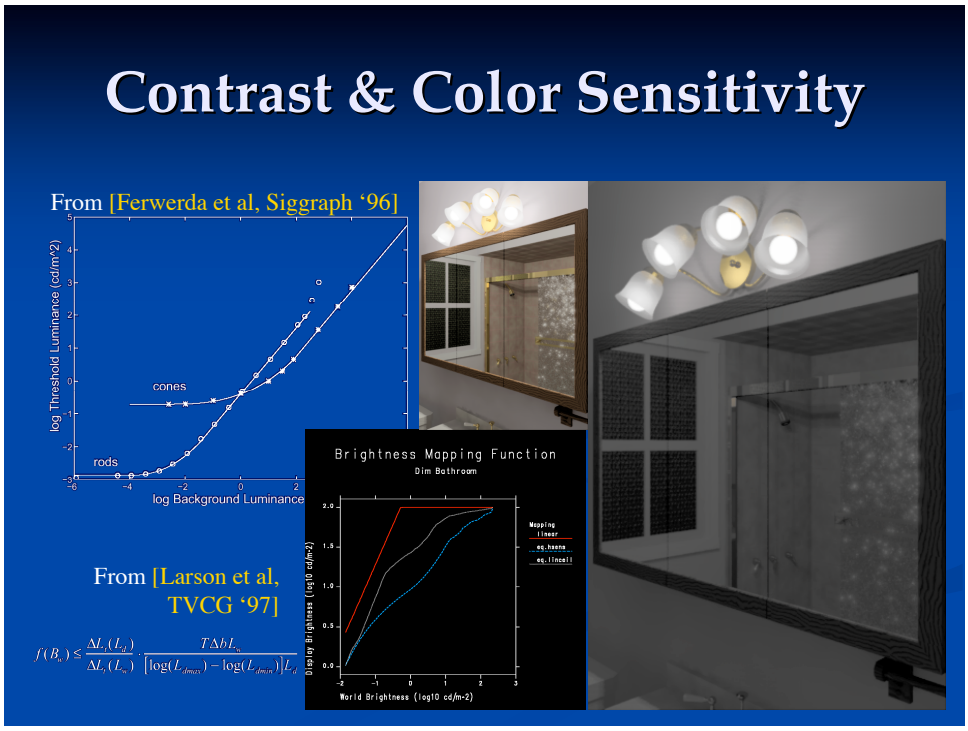
Real Numbers, Real Images

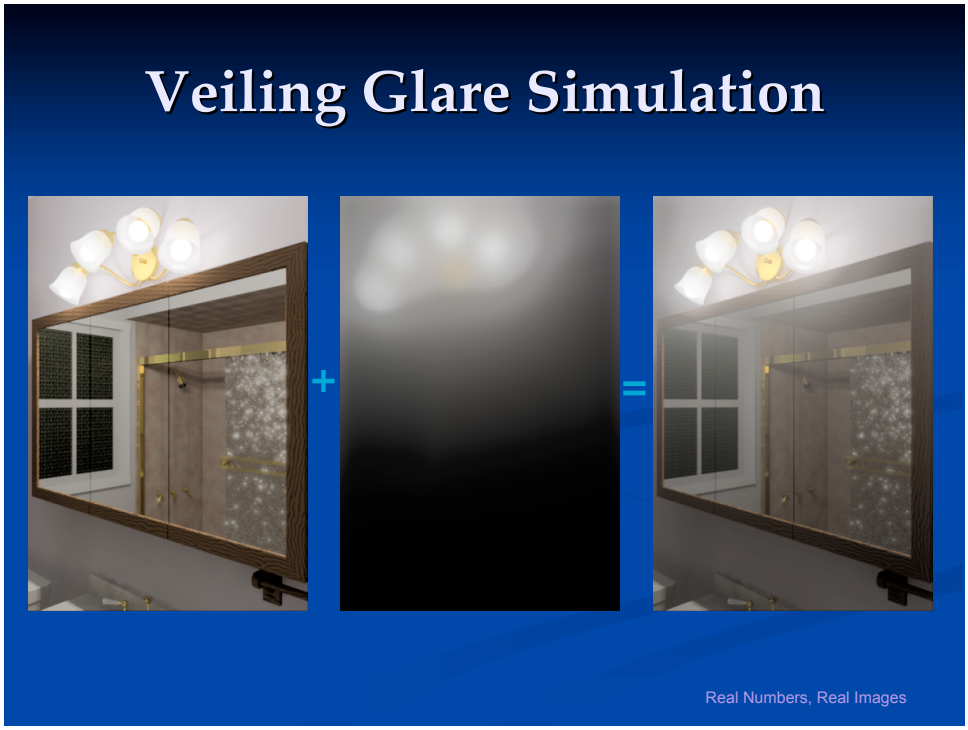
Histogram Adjustment





Result


Contrast & Color Sensitivity
From [Ferwerda et al, Siggraph '96]
log Threshold Luminance (cd/m^2)
log Background Luminance
cones
rods
From [Larson et al, TVCG '97]
Brightness Mapping Function
Dim Bathroom
Display Brightness (log10 cd/m-2)
World Brightness (log10 cd/m-2)
Mapping
linear
eq.heena
eq.lincel


Veiling Glare Simulation
+
=
Real Numbers, Real Images

Other Tone Mapping Methods

- Retinex-based [Jobson et al. IEEE TIP July '97]
- Psychophysical [Pattanaik et al. Siggraph '98]
- Local Contrast [Ashikhmin, EGWR '02]
- Photographic [Reinhard et al. Siggraph '02]
- Bilateral Filtering [Durand & Dorsey, Siggraph '02]
- Gradient Domain [Fattal et al. Siggraph '02]

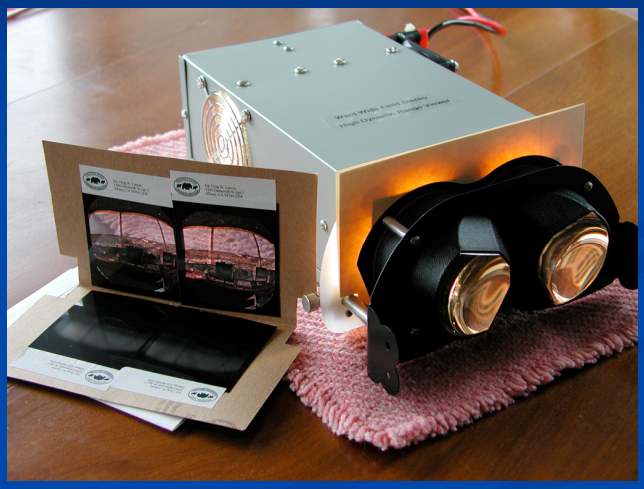
Real Numbers, Real Images

High Dynamic-range Display

- Early HDR display technology
  - Industrial high luminance displays (e.g., for air traffic control towers) not really HDR
  - Static stereo viewer for evaluating TMO's
- Emerging HDR display devices
  - Collaborative work at the University of British Columbia in Vancouver, Canada

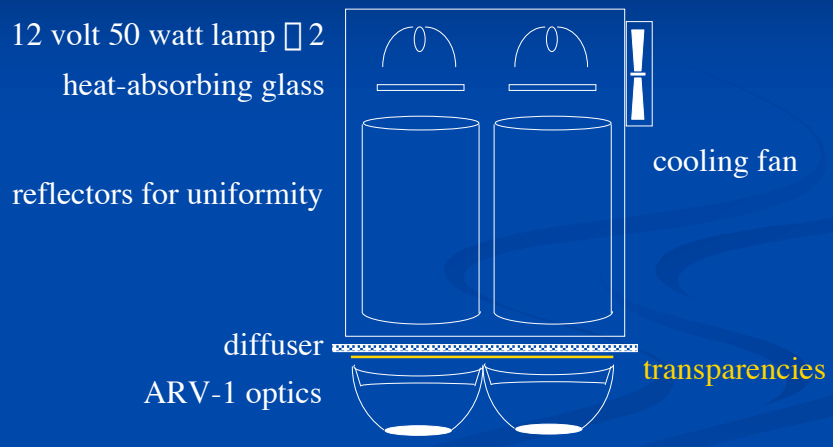
Real Numbers, Real Images

Static HDR Viewer



Real Numbers, Real Images

HDR Viewer Schematic



Real Numbers, Real Images

Viewer Image Preparation

- Two transparency layers yield 1:10^4 range
  - B&W "scaling" layer
  - Color "detail" layer
- Resolution difference avoids registration (alignment) problems
- 120° hemispherical fisheye perspective
- Correction for chromatic aberration

Real Numbers, Real Images

Example Image Layers

Scaling Layer

Detail Layer

UBC Structured Surface Physics Lab HDR Display

- First generation DLP/LCD prototype
  - 1024x768 resolution
  - 10,000:1 dynamic range
  - 7,000 cd/m^2 maximum luminance
- Next generation device w/ LED backlight
  - Flat-panel design presented at SID
  - 10,000:1 DR and 10,000 max. luminance

Real Numbers, Real Images

UBC HDR Display Prototype

Real Numbers, Real Images

VI. Image-based Techniques

- High dynamic-range photography
  - Using *Photosphere*
- Image-based lighting
- Image-based rendering

Real Numbers, Real Images

HDR Photograhy

- Standard digital cameras capture about 2 orders of magnitude in sRGB color space
- Using multiple exposures, we can build up high dynamic range image of static scene
- In the future, manufacturers may build HDR imaging into camera hardware

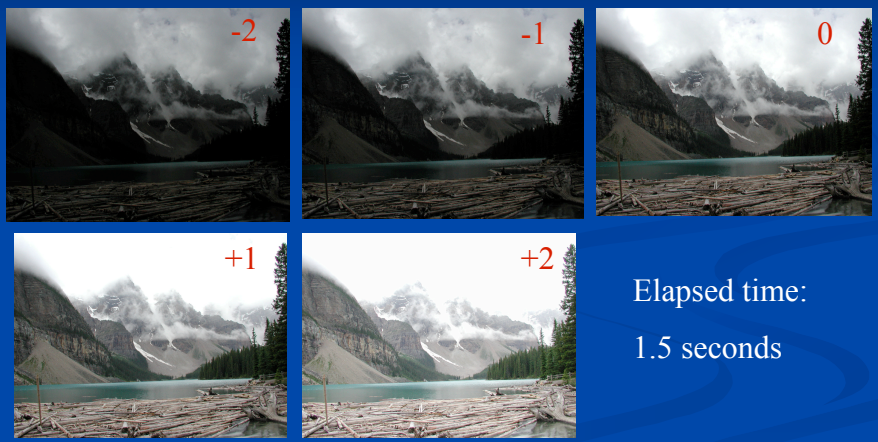
Real Numbers, Real Images

Hand-held HDR Photography

- Use "auto-bracketing" exposure feature
- Align exposures horizontally and vertically
- Deduce camera response function using [Mitsunaga & Nayar '99] polynomial fit
- Recombine images into HDR image
- Optionally remove lens flare

Real Numbers, Real Images

Auto-bracket Exposures



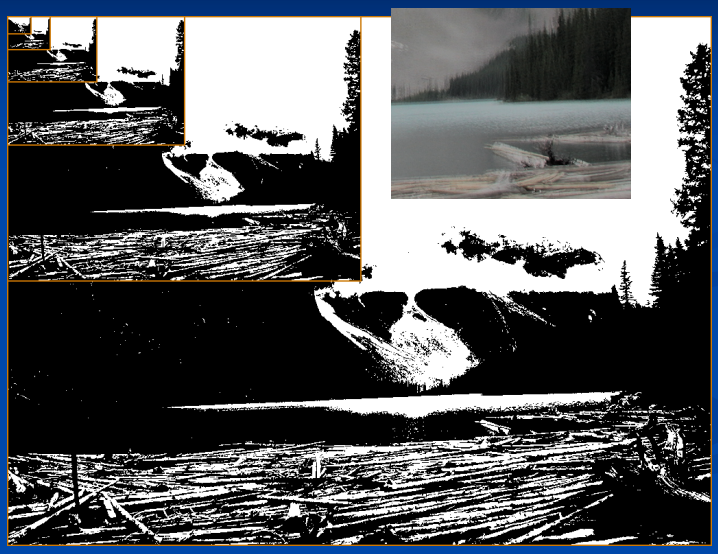
Elapsed time:

1.5 seconds

Real Numbers, Real Images

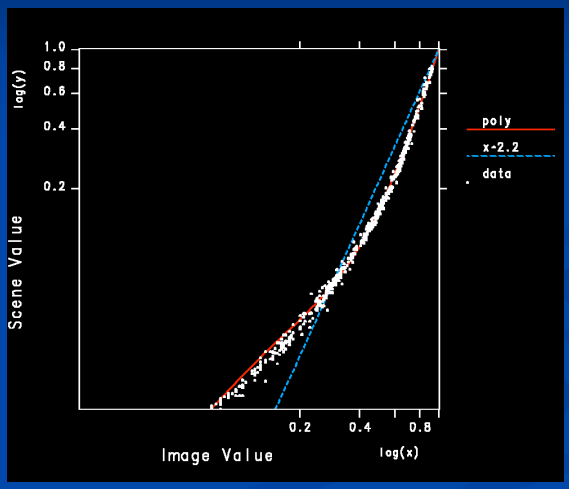
LDR Exposure Alignment

Align median threshold bitmaps at each level of pyramid



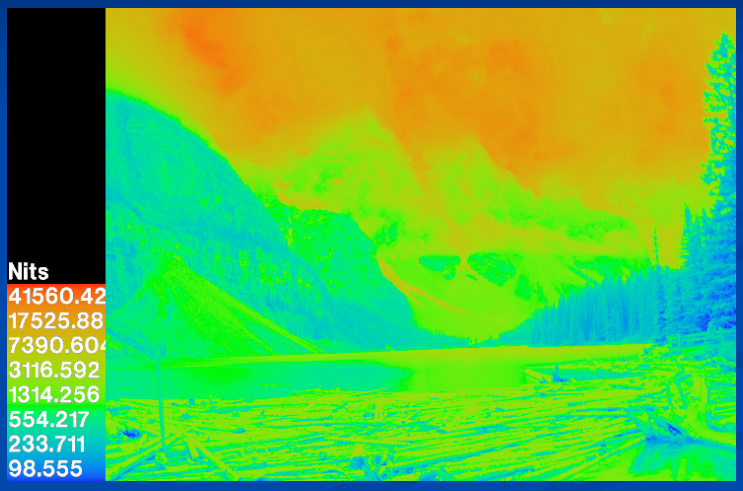
Real Numbers, Real Images

Estimated Camera Response



Real Numbers, Real Images

Combined HDR Image



Real Numbers, Real Images

Tone-mapped Display

Real Numbers, Real Images

Best Single Exposure

Real Numbers, Real Images

Lens Flare Removal

Before After

Real Numbers, Real Images

Photosphere HDRI Browser

- Browses High Dynamic Range Images
  - *Radiance* RGBE format
  - TIFF LogLuv and floating point formats
  - OpenEXR short float format
- Makes HDR images from bracketed exposures
- Maintains Catalog Information
  - Subjects, keywords, albums, comments, etc.
- Tracks Image Files
  - Leaves file management & modification to user

Real Numbers, Real Images

Realized Features

- Fast, interactive response
- Thumbnails accessible when images are not
- Interprets Exif header information
- Builds photo albums & web pages
- Displays & edits image information
- Provides drag & drop functionality
- User-defined database fields

Real Numbers, Real Images

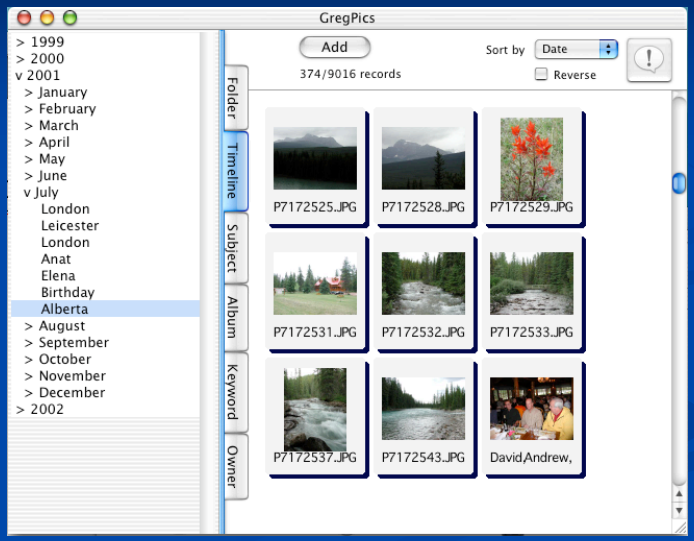
Unrealized Features

- Accurate color reproduction on all devices
- Plug-in interface for photo printing services
- Linux and Windows versions
- More supported image formats
  - Currently JPEG, TIFF, *Radiance*, OpenEXR

Real Numbers, Real Images

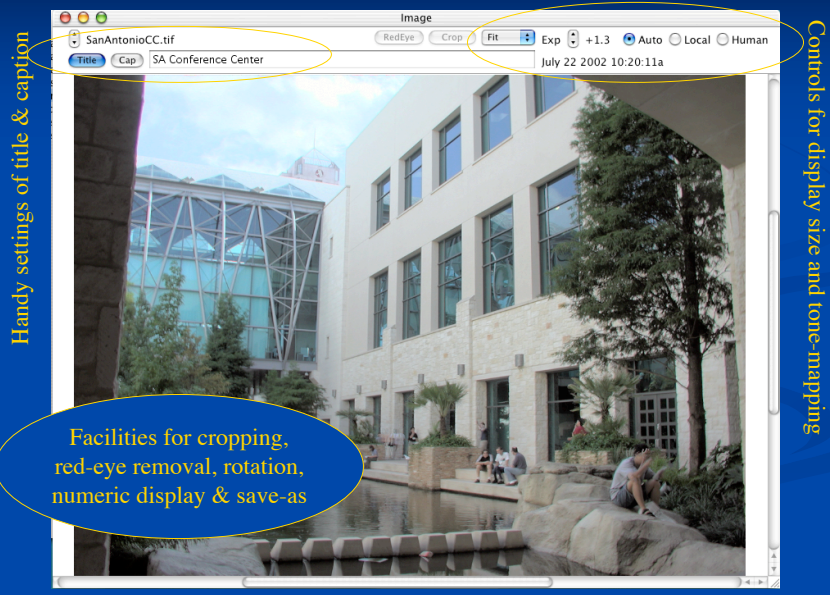
Browser Layout

Selector Tabs permit multiple image selection from file system or catalog DB



Thumbnail sizes up to 320-pixel resolution preview

Viewer Layout

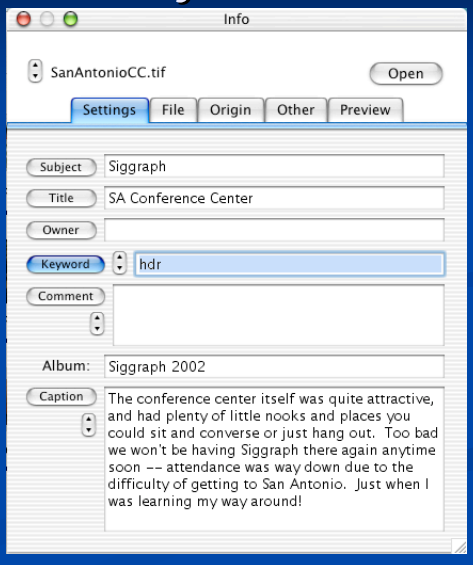


Info Window Layout

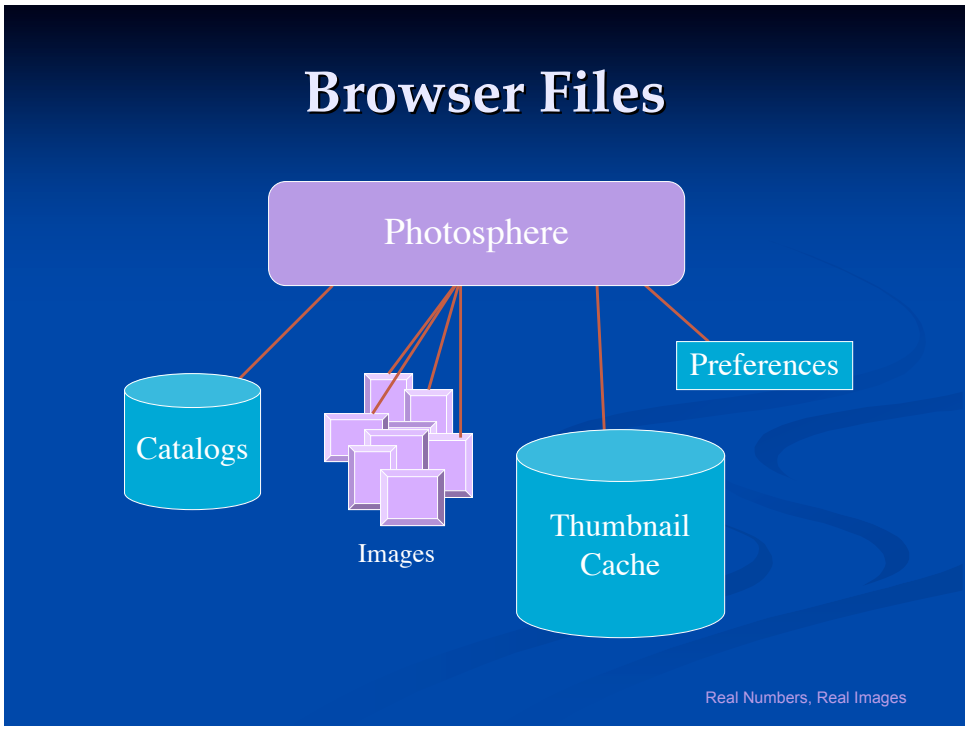
Provides convenient access to individual image settings and information

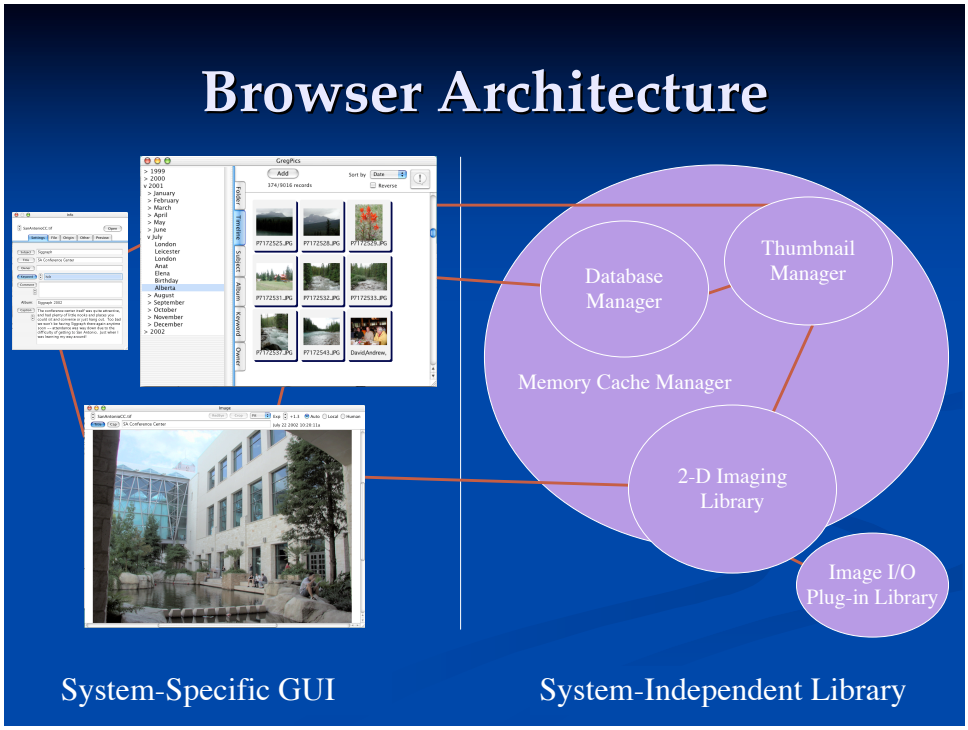
Most functionality is duplicated in application Set menu, which are more convenient for setting values on multiple images

A handy browser "pop-up" feature also provides a preview and detailed image information on any selected thumbnail, and info listing is offered as alternative to thumbnail display



Real Numbers, Real Images


Browser Files
Photosphere
Preferences
Catalogs
Images
Thumbnail Cache
Real Numbers, Real Images


Browser Architecture
Database Manager
Thumbnail Manager
Memory Cache Manager
2-D Imaging Library
Image I/O Plug-in Library
System-Specific GUI
System-Independent Library

Photosphere Demo

Photosphere

Available from www.anyhere.com

Real Numbers, Real Images

Image-based Lighting

- Photograph silver sphere using HDR method
- Place as environment map in scene to render
- Sample map to obtain background values

See www.debevec.org for more details and examples

Real Numbers, Real Images

Image-based Rendering

- Mixed reality is the future for graphics
- High dynamic-range imaging is the key
- Accuracy in rendering is also critical for seamless integration
- A lot of work has been done in the areas of image-based lighting and rendering, but we've only scratched the surface
  - Films like *The Matrix* rely heavily on IBL/IBR

Real Numbers, Real Images

IBR/IBL Example

Take a lousy model

Use captured images to fix it

Real Numbers, Real Images

VII. Conclusions

- Two paths to realism:
  1. Work like nuts until it "looks OK," or
  2. Apply psychophysics of light and vision
- As authors of rendering software, we can save users a lot of (1) with a little of (2)
- Real numbers are needed for physical simulation, as values are unbounded
- The eye and brain are analog devices

Real Numbers, Real Images

Further Reference

- www.anyhere.com/gward
  - publication list with online links
  - LogLuv TIFF pages and images
- www.debevec.org
  - publication list with online links
  - *Radiance* RGBE images and light probes
  - *HDRshop* and related tools
- www.idruna.com
  - *Photogenics* HDR image editor
- radsite.lbl.gov/radiance
  - *Radiance* rendering software and links

Real Numbers, Real Images

Appendix A:

Tools for Lighting Design and Analysis

Reprinted from SIGGRAPH 1996 Course Notes

Tools for Lighting Design and Analysis

Gregory J. Ward
Lawrence Berkeley National Laboratory
Berkeley, CA

ABSTRACT

Herein we describe some of the more useful tools and methods for applying computer technology to lighting design problems. We discuss input tools for 3D geometry, materials and photometry, simulation and rendering tools and methods, and output devices and formats. Numerous examples are given.

DISCLAIMER

A number of specific products and manufacturers will be mentioned in the text of these course notes. They are merely meant to serve as examples, and our references to them should not be construed as endorsements of their suitability or merit for any particular application. The only exceptions to this rule are products associated with the organizers of this course, which we will plug shamelessly.

Introduction

In this course, we have glimpsed some of the possibilities and applications of global illumination in architecture and entertainment. However, it is clear that there are still many details to be worked out. For instance, how do we generate the input necessary for a good global illumination calculation? What is the most appropriate calculation tool for our application? How can we best represent and present our results? Though these issues may seem peripheral, they will eventually determine the success or failure of our efforts.

In this section of the course, we will discuss some of the hardware and software tools available for generating 3D models with physical properties, performing appropriate lighting and rendering computations, and presenting the results. We will start with input tools and methods, then look at simulation and rendering, followed by output and communication. As part of the discussion of output, we will give some examples from lighting design work conducted at LBNL and elsewhere.

Input

There are three interrelated classes of data that are important to global illumination: geometry, materials and photometry. Most of us are familiar with geometry and its input, but few computer graphics practitioners have experience with physical materials, and even fewer understand photometry. This is because, even today, most rendering programs ignore physical lighting properties in favor of simplistic or artistic views of the world. Not so in the case of global illumination -- here we need to know exactly how light interacts with surfaces in order to predict their true appearance. The material models describe how light reflects off and transmits through surfaces, and the photometry describes how light emanates from light sources.

Tools for Geometry

Many of the tools and methods available for describing geometry will be familiar to those with experience in generating 3D computer graphics imagery. Nevertheless, we will go over some of the more useful tools here for the sake of completeness.

CAD Systems

Computer-Aided Design systems are the most common means of generating 3D input for global illumination. Sometimes, a CAD system is included in the rendering system itself, which ensures that the data is compatible and complete. However, writing a good geometric editor is very difficult, so the more common approach is to import 3D geometry in some standard format, such as IGES, DXF or VRML, and rely on a commercial CAD program to create the geometric description.

Among the commercial CAD programs, AutoDesk's *AutoCAD* is probably the most popular worldwide, because it has been around a long time and many companies have successfully marketed value-added software functionality and libraries for it. Unfortunately, because it was originally designed strictly as a 2D drafting tool, its 3D editing abilities are somewhat awkward and difficult to use. Many other CAD systems are available, some of them as shareware and freeware and some are much better than *AutoCAD*, but the choice of CAD system is usually made in advance by personal preferences or company requirements, so we will not discuss this further here.

The main issue when considering which CAD system to use it is whether or not it can produce a usable model for global illumination. Unfortunately, the only way to know for sure is to try it. In some cases, compatibility depends on *how* the system is used -- i.e., sticking to exportable primitives and labeling surfaces in such a way that the appropriate materials may later be linked to them. Other issues related to exportability include surface normal orientation, planarity, meshing and vertex accuracy.

3D Scanners and Digitizers

As fun as CAD programs are, we often wish for an easier method to input a model of an object we have on hand. After all, if the 3D geometry of a telephone or a stapler is right in front of us, why should we have to recreate it all over again in a CAD program?

Two types of devices can help us out, 3D scanners and 3D digitizers.

A 3D scanner is a device that automatically scans in an object to create a 3D model of that object. This is very difficult to do in general, because objects may have concavities and reflectance properties that cannot be captured or adversely affect the capturing process. There are only a few devices currently on the market for capturing full 3D geometry. The best known products come from Cyberware (http://cyberware.com), which use an off-axis laser scanning method. A London company manufacturing scanners is 3D Scanners Ltd. (http://www.3dscanners.com). In general, these devices are bulky and expensive.

3D digitizers are much cheaper and simpler in concept, but generating detailed models with them is time-consuming. They typically have an arm with a stylus attached, which is used to locate vertex points in 3-space. Some devices are wireless and cover a larger volume, but accuracy can be an issue. 3D digitizers are frequently advertised in graphics and CAD magazines.

Graph Paper

Sometimes the simplest solution is the best. A method that has worked for me for many years is one of projecting an object onto a piece of graph paper and tracing its outline. Coordinates can then be located and, provided the object has some symmetry, a nice 3D model can be constructed. An overhead projector may be used to get a nice silhouette, or if the object is reasonably flat, it may even be traced directly. The data points may then be entered into a CAD program or your favorite text editor.

FTP and Web Sites

Of all the methods for creating 3D models, the quickest and easiest will always be downloading them off the net. In some cases, models are offered free of charge by magnanimous individuals or organizations. Reputably the largest free repository was assembled by the Navy's Vislab at China Lake, and dubbed Avalon. This site also contains useful tips and information on converting models from various formats. Avalon is currently maintained by Viewpoint Datalabs, who has their own proprietary data models. Another point

of interest is the MGF web site, who also offers complete models with all the necessary materials and photometry for global illumination in a well documented format with an ANSI C parser. Here is an abbreviated list of 3D model sites (* means all data is free of charge):

http://www.viewpoint.com/avalon	Avalon, maintained by Viewpoint*
ftp://wuarchive.wustl.edu/graphics/graphics/mirrors/avalon	US mirror*
ftp://ftp.flashnet.it/avalon	Italian mirror*
ftp://sgi.felk.cvut.cz/pub/avalon	Czech mirror*
http://www.viewpoint.com	Viewpoint main site
http://radsite.lbl.gov/mgf	MGF web site*
http://www.acuris.com	Acuris web site
http://204.191.254.145	QuickDraw 3D Clipart

Tools for Materials

The materials in a scene or object description describe how light interacts with each surface. The important thing for global illumination is that the material models must be physically plausible, that is, they must adhere with some consistency to physical laws governing light transport. Unfortunately, the most commonly used material models in computer graphics have a very weak physical basis, and in most cases they do not yield plausible results if applied in their original form. (See [He91], [Ward92], [Schlick93] and [Lewis93] for good counterexamples.) Therefore, most materials that accompany computer graphics models must be reinterpreted in a physical context, or recreated from scratch.

In the case where the actual object or scene is available, one may wish to measure the materials directly. The methods and tools used for such measurement depend on the type of material being measured and the desired application and accuracy. Here are some examples.

Cameras and Scanners

When a material has a prominent texture or pattern and we want to capture this for a more realistic rendering, we can scan this pattern into our computer. If we have a sample of the material handy, we may scan it in directly by placing it on a flatbed scanner. This usually yields the highest quality results. If the object is in the field and unmovable, we might rather take a picture of it and scan in the picture using a 35mm scanner, or go directly to digital with a digital camera. Alternatively, we might capture frames from a video, though this tends to compromise resolution and quality quite a bit.

However we get the data to our computer, we will have difficulty getting accurate color and reflectance information unless we are very careful. The simplest method for maintaining accurate color and contrast in this kind of process is to transfer a reference chart in parallel, such as the Macbeth ColorChecker Chart[1]. When lighting is uniform and controlled, as in the case of a flatbed scanner, absolute color and reflectance accuracy can be quite good. Figure 1 shows the result of a calibration process using the Macbeth chart.

[1] Available in most photo supply stores, or contact Munsell Color, 405 Little Britain Rd., New Windsor, NY 12553, 1-800-MACBETH (USA) or 1-800-COLOUR (Canada).

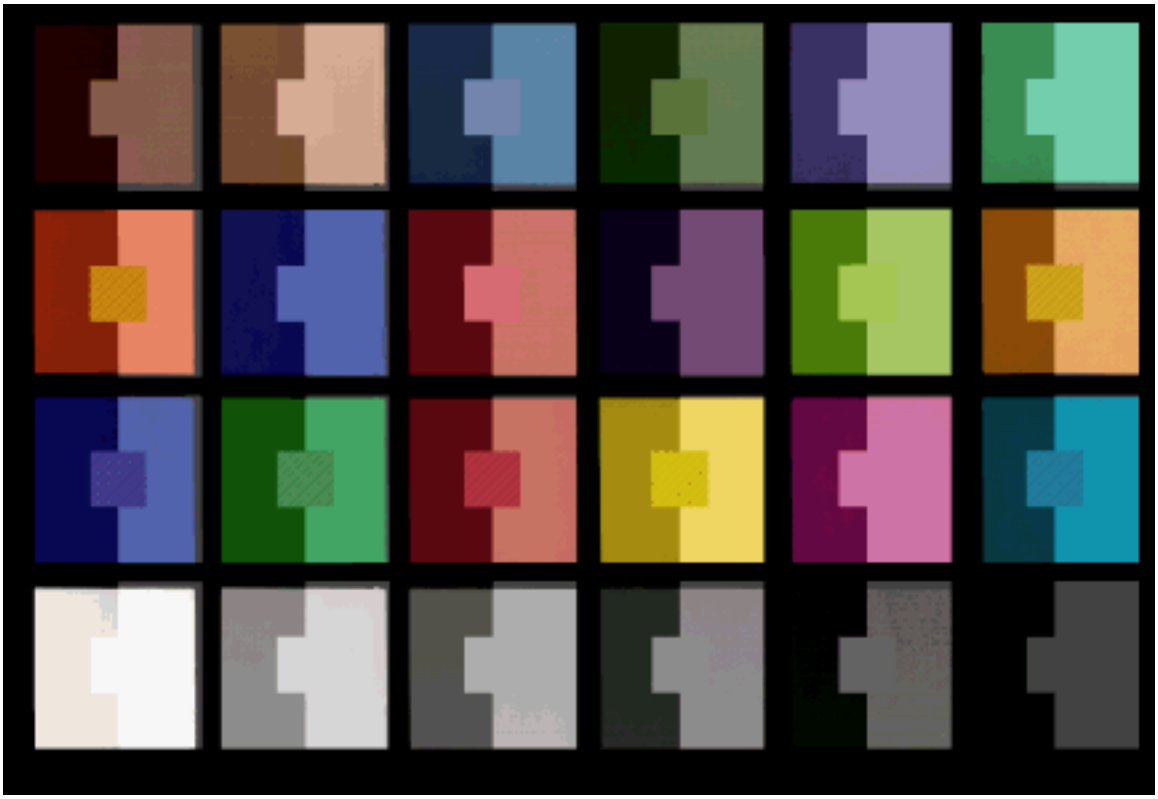


Figure 1. Automatic contrast and color calibration using a Macbeth ColorChecker. The left side of each square shows original scanner results. The midsection of each square shows target Macbeth color, according to published CIE chromaticity and reflectance values. The right side of each square shows results achieved through color/contrast correction step. Notice that grays are matched very accurately, as are most colors, but some highly saturated colors are off, which may indicate that these colors are outside of the device's gamut.

Spectrophotometers

If a surface has fairly uniform color, more accurate measurements of its reflectance are possible. A *spectrophotometer* (or *spectrometer* for short) is a device that measures the reflectance or transmittance (or emittance) of a surface at multiple wavelengths. Some devices, such as the Colortron by Lightsource (http://www.lightsource.com) use a diffraction grating to separate visible wavelengths. Figure 2 shows an example of a reddish surface measured in this way. Other devices use a set of highly selective color filters, such as those in the Minolta CM-2002. The CM-2002 has the added advantage of being able to measure reflectance with and without the specular (mirror) component. Figure 3 shows a diagram of the internal workings of this device.

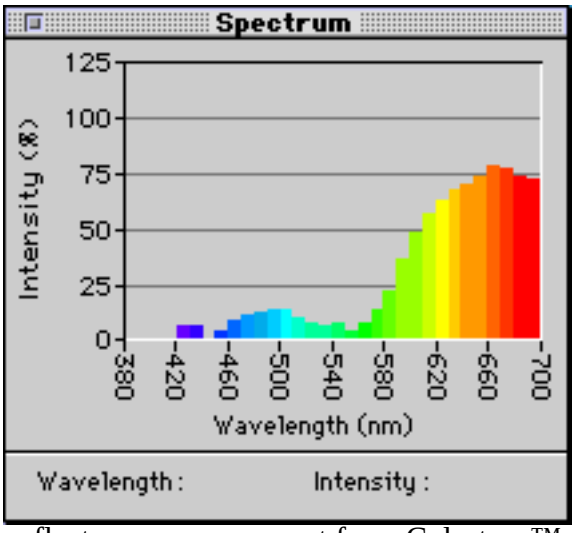


Figure 2. Spectral reflectance measurement from Colortron™ spectrometer. Each waveband is about 10 nm wide.

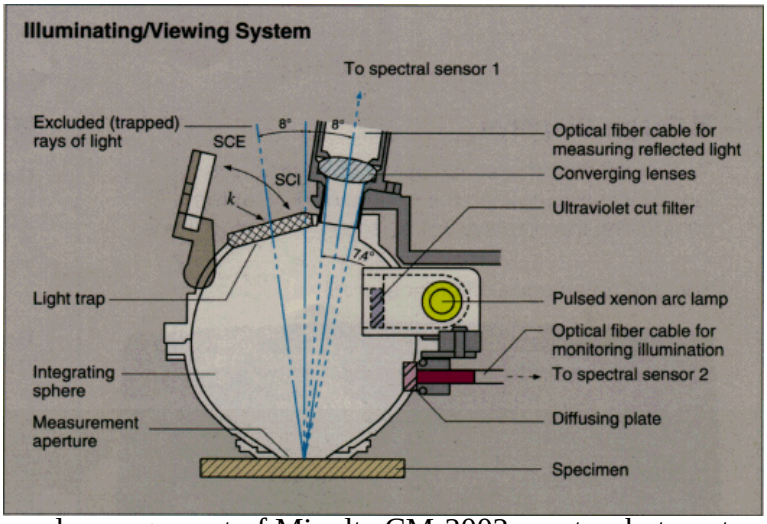


Figure 3. Internal arrangement of Minolta CM-2002 spectrophotometer. Note the trap door for excluding the specular component.

Gloss Meters

Cameras and spectrometers work fine for diffuse materials (or in the case of the Minolta CM-2002 described above, diffuse materials with some ideal specular component), but what about surfaces with some rough specular reflection? How do we measure and characterize material highlights? For many years, auto manufacturers and other industries concerned with the appearance of shiny paints have employed devices called *gloss meters* to do just this. A gloss meter takes a relative measurement of the intensity of a highlight at one or a few different angles. It is a purely relative measurement, and mixes highlight spread with intensity in a way that is difficult to translate to useful material properties. Therefore, we will not say anything more about these measurements here, as we do not recommend them for computer graphics or global illumination.

Gonioreflectometers

A *gonioreflectometer* is a device that measures reflection as a function of incident and reflected angle in order to arrive at the bidirectional reflectance (or transmittance) distribution function for a surface (BRDF or BTDF). Devices that measure this function as it depends on

wavelength also are called *goniospectroreflectometers*. (Sometimes people run out of breath and just call them all *goniometers* .) These devices tend to be large, expensive and non-portable. Figure 4 shows an imaging gonioreflectometer developed by the author, which is further described in [Ward92].

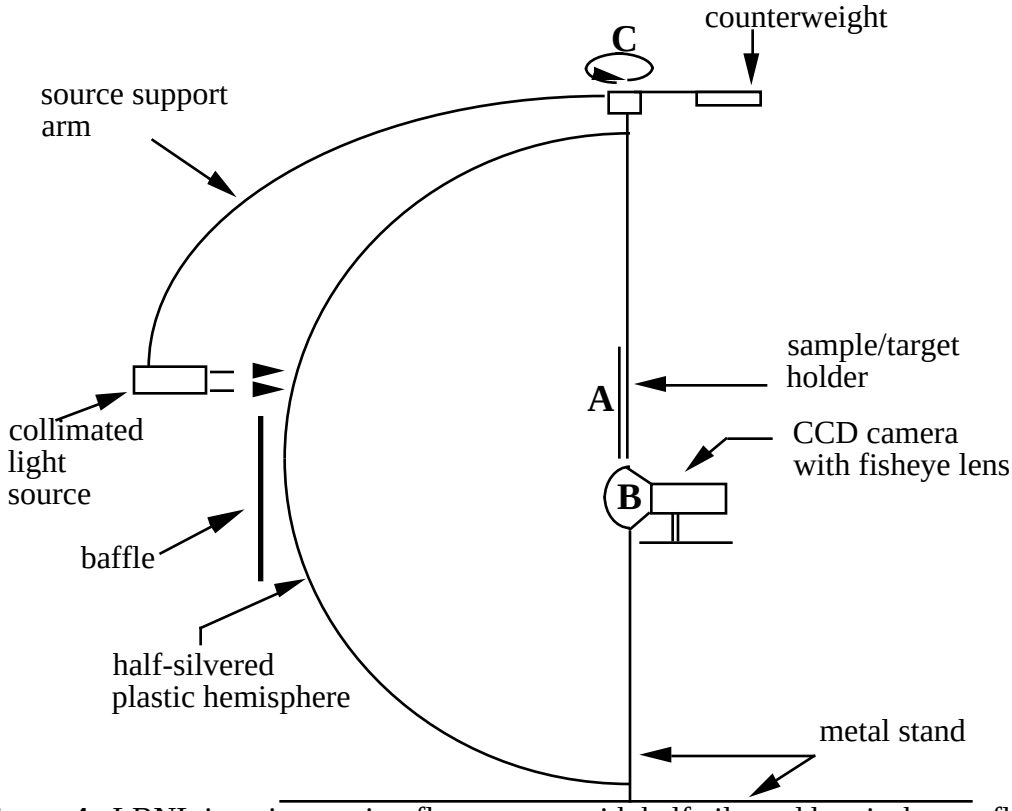


Figure 4. LBNL imaging gonioreflectometer, with half-silvered hemisphere reflecting light into fisheye lens of digital camera [Ward92].

We hope to see simpler and more practical methods developed in the near future. We especially need methods for characterizing BRDFs that are cheap and portable, like the method recently proposed by Karner et al, which uses a digital camera and a fixed source and reference material [Karner96]. Figure 5 shows this sample arrangement.

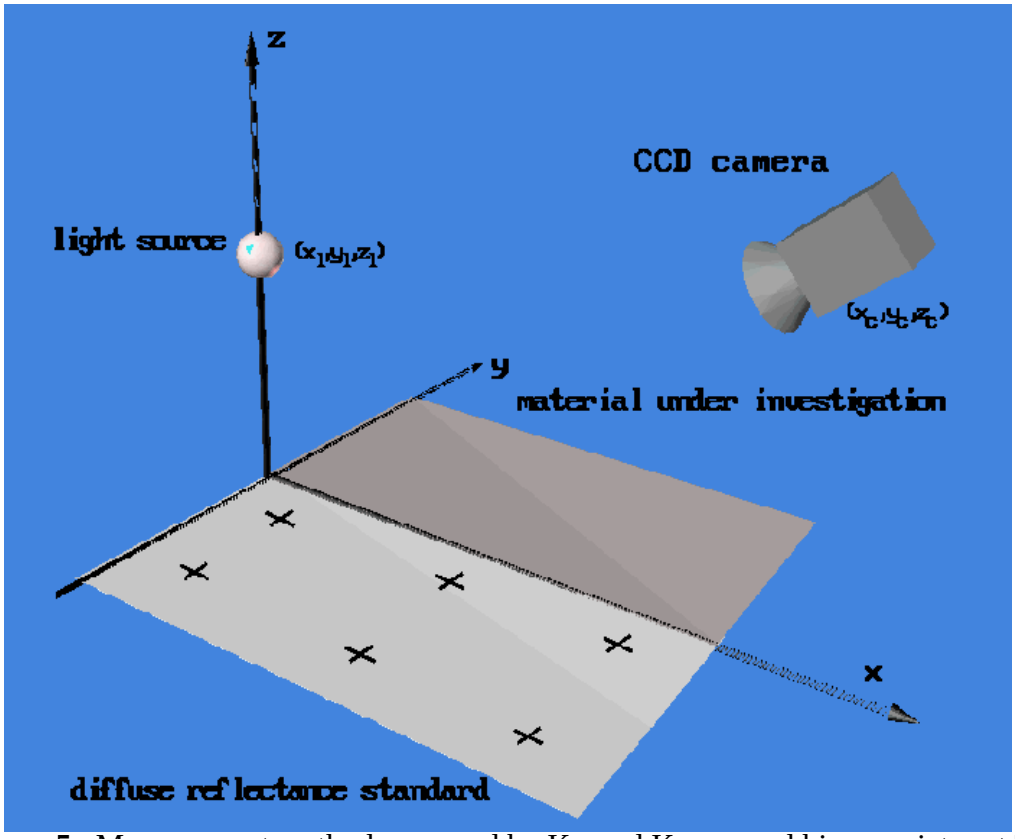


Figure 5. Measurement method proposed by Konrad Karner and his associates at TU Graz in Austria for obtaining BRDF using simple camera, source and reflectance standard arrangement [Karner96].

Tools for Photometry

Photometry is the measurement of light output from sources in an environment. In most scenes, this might include daylight or electric light. Daylight is generally described by the sun and a model sky distribution. The sun varies only in overall intensity, as determined by its altitude and the atmospheric conditions. Under cloud cover, the sun may not be visible at all. The sky model can be very simple (such as a uniform sky) or very sophisticated (such as a measured sky). Electric lighting usually means luminaires in the environment, each having a photometric output distribution. The photometry of a luminaire is generally available from its manufacturer, which relieves us of the nasty business of goniophotometric measurements.

Daylight Models

The International Commission on Illumination (CIE) has adopted three reference models for sky luminance distribution: the CIE overcast sky, the CIE clear sky and the CIE intermediate sky (http://chelsea.ios.com/~kdtjek/cieusa.html, pub #110). These models offer reasonable approximations to a completely overcast day, an ideal clear sky, and a sky that has some uniform cloud cover but still allows direct sun to seep through. Some global illumination programs may also provide a choice for a uniform sky distribution, which is merely a constant value over the entire hemisphere. This distribution is not very realistic, but may serve as a point of reference in some calculations. The CIE overcast sky model is most often used in *daylight factor* calculations (the ratio of interior to exterior horizontal illuminance), since the distribution does not vary with sun angle.

It is very important to understand the limitations of reference sky models. Since they serve only as ideals, they are not very good at predicting actual interior (or even exterior) illumination values. For this, we really need measured sky data. At the very least, we need

measurements of the global and diffuse sky components, which allow us to set approximately correct values for the sky and solar intensities. However, to get truly accurate results for a specific time of day and year, we need measurements of the sky distribution and solar component for that moment. Devices exist for taking such measurements, but like the gonioreflectometers discussed in the previous section, then tend to be expensive, bulky, and difficult to use.

Luminaire Photometry

If sky measurement is difficult, measuring the output of luminaires is even worse. Fortunately, most luminaire manufacturers do this for us, often sending the work out to independent laboratories to retain some semblance of objectivity. The data is then made available to us as photometric reports or standard luminaire data files. The latter is obviously preferable if our intent is to apply some calculation tool such as a global illumination or rendering program. All we need to check is that our program takes the provided format as input.

In the U.S., there is really only one generally agreed upon standard for luminaire data, put forth by the Illuminating Engineering Society of North America (IESNA or IES for short). It is available as IES publication LM-63-91, *Standard File Format for Electronic Transfer of Photometric Data*. (Replace the last two digits of the code with the year a new one becomes available.) Connect to the IESNA web site (http://www.iesna.org) for more information. A free parser for this format, written by Ian Ashdown, is available from the Ledalite website (http://www.ledalite.com).

In Europe, the CIE has put forth their own standard for luminaire data, available as pub #102. (See the CIE web site at http://chelsea.ios.com/~kdtjek/cieusa.html. Also, Andrew Glassner's book, *Principles of Digital Image Synthesis*, offers excellent summaries of this and the IESNA standard [Glassner95].) This is a relatively new standard, and there are still many other country-specific standards in use in Europe, such as CIBSE and LTLI. Hopefully, one day the world will agree on a single standard.

Combining Input

It would be nice if there were a common format for storing all the essential input information for global illumination. Currently, we must import everything into the simulation or rendering program we wish to use, with little hope of sharing this information between different calculations once it is combined.

Recently, several new formats have emerged for representing 3D geometry and materials, including VRML (a network extended version of SGI's Inventor format), QuickDraw 3D (from Apple) and 3D Studio. Unfortunately, none of these formats employ physically plausible material models, and none of them allow for light source photometry, so they do not really provide complete descriptions for global illumination.

There is currently one finished standard that makes a reasonable attempt to provide complete descriptions for global illumination in a neutral format, called the Materials and Geometry Format (MGF). (See http://radsite.lbl.gov/mgf.) MGF shall soon be integrated into the IESNA standard for luminaire data, described in the previous section. Two other efforts are underway to provide more ambitious and complete descriptions of objects for all types of simulation, the Industry Alliance for Interoperability (IAI) in the U.S. and STEP in Europe. These last two efforts will likely merge to one day provide a single, global standard for describing everything computers want to know about manufactured objects. (See http://elib.cme.nist.gov/pub/nipde regarding STEP or http://www.interoperability.com regarding IAI.)

Lighting Simulation and Rendering

The type of global illumination calculation we perform depends on the kind of input we have, the degree of accuracy we require, and the kind of output we want. Usually, it works best to think this through backwards, that is, decide *first* what kind of output we need for our application, then decide what simulation tool works best for this, then go about gathering the necessary input.

Lighting simulation and rendering can be thought of as a sort of continuum, starting with the most basic calculations and moving on towards the ultimate in predictive reality. Figure 6 shows this continuum with three example software packages. The horizontal axis shows the relative effort required on the input side, and the vertical axis shows the quality and detail produced on the output side.

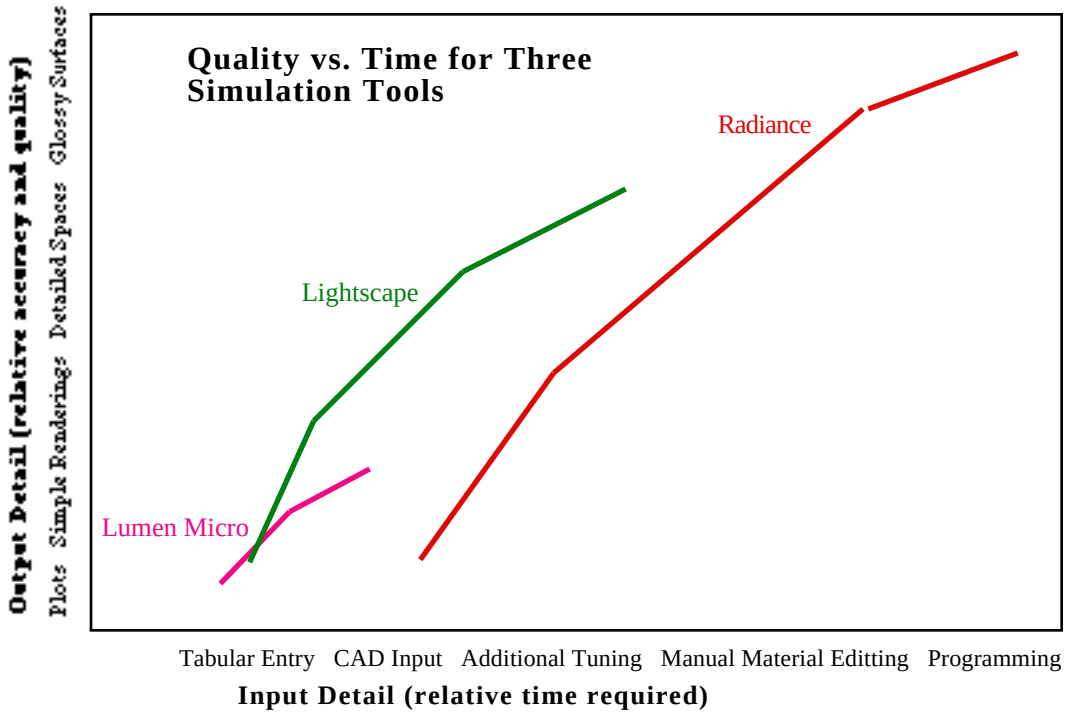


Figure 6. Three curves showing output quality versus input time for three lighting simulation tools.

Figure 6 shows that the output desired affects the level of effort required on the input side as well as the choice of simulation software. *Lumen Micro* is good for simple calculations and spaces, *Lightscape* works well for intermediate level of detail, and *Radiance* for high detail spaces [Ward94b]. *Lightscape* offers a nice user interface, which is why its curve falls slightly above the other two -- it takes less time to get the same quality of output. The fact that these are curves instead of straight lines reflects the law of diminishing returns as the user expends additional effort on the input side.

For more information on *Lumen Micro*, contact Lighting Technologies in Boulder, Colorado (303) 449-5791. *Lightscape* has a web page at http://www.lightscape.com. The official web site for *Radiance* is http://radsite.lbl.gov/radiance.

Output and Communication

From what we just noted in the previous section, we are treating last the topic that should be considered first, namely what sort of output is needed for a particular application. For some

applications, such as routine lighting design, simple line graphs or contour plots may be sufficient. In some cases, the designer or client may wish to see some crude renderings of the space without furniture or other details. More critical applications may require daylighting analysis or the inclusion of partitions and some furniture, though a diffuse approximation to interreflection may still be adequate. For critical visual applications such as interior design or control operations environments, full modeling of detailed furniture and reflection is often necessary.

In addition to considering the types of calculations required, it is important to think about what output media best convey the information to the intended audience. If the audience is an experienced lighting designer, very crude renderings or even point values may be sufficient. A typical design client (building owner or buyer) usually prefers color renderings. If the design is part of a competition or other demonstration, it may be worthwhile to produce an animation.

The time required to produce a particular result is generally a product two axes: input detail and output resolution. By "output resolution, " we mean the number of individual calculations needed. Point calculations tend to be sparse, so this is the cheapest output to produce. High resolution video is the most expensive output, since every pixel of every frame must be computed somehow. For simple environments, even high-resolution video can be produced in a reasonable time frame, but few people are interested in walk-throughs of empty spaces. Likewise, point calculations can be done fairly quickly even in a complicated environment, but this does not make good use of the detail in the model.

Let us summarize and demonstrate some of these output possibilities.

Numerical Lighting Values

Numerical lighting values are often used to decide if light levels are adequate in a space and to check that there are no problematic sources of visual discomfort or glare. Figure 7 shows a daylight factor contour plot generated from a small set of workplane illuminance values computed by the *ADELINE* software package (http://radsite.lbl.gov/adeline). Similar plots may be obtained for illuminance. Figure 8 shows a glare calculation, which locates extremely bright regions in the scene and determines visual discomfort probabilities for horizontal view directions. (Accurate glare calculations may require non-diffuse reflection models, and are among the more difficult numerical analyses used in lighting.)

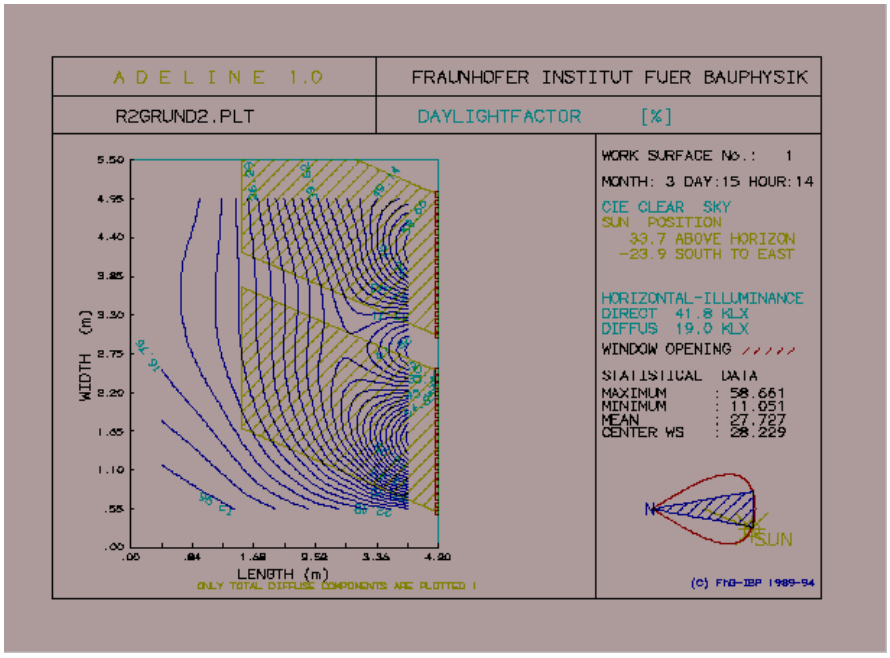


Figure 7. Daylight factor contours for workplane of two-windowed office space (empty), computed by *ADELINE*. Sun patches are shown as yellow, hatched regions.

Figure 8a. Visual comfort probability (VCP, glare) calculation for cubicle office space, showing position and intensity of glare sources, computed with *Radiance*.

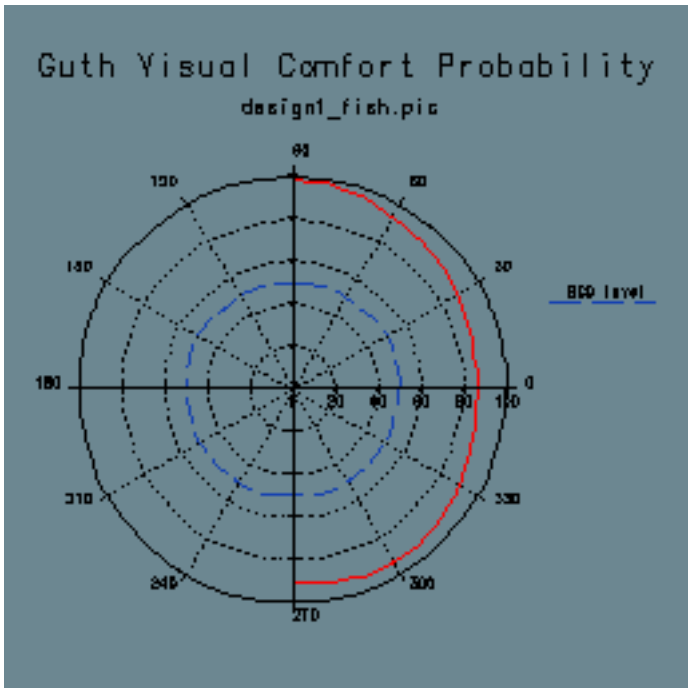


Figure 8b. Visual comfort probability (VCP, glare) calculation for cubicle office space, showing a numerical plot of VCP. The zero degree view is looking straight ahead in the image shown, and positive angles look to the right. A VCP of 75% means 75% of occupants are comfortable.

Renderings

The most natural medium for presenting lighting simulation results is of course the rendered image. Renderings are easy for anyone to interpret, as they attempt to show what a design space will look like. How well they achieve this aim depends on the modeled environment, the software, and the chosen display method.

The chief limitation of most display methods is dynamic range. A typical scene has luminance ratios on the order of 1000:1, with many daylight scenes exceeding 10000:1. Unfortunately, most display monitors have a maximum dynamic range of around 100:1, with film recorders doing slightly better and other hardcopy devices doing slightly worse. There simply are no generally available display methods that approach the dynamic range of a real scene. What this means is, what you see in an image is not always what you get in real life.

A few different researchers have addressed the "tone mapping" problem as it is called. The pioneering work in computer graphics was conducted by Tumblin and Rushmeier [Tumblin93], who developed an exponential function to produce roughly the same apparent brightness as real environments. This author took a slightly different approach [Ward94a], attempting to reproduce visible contrast differences with a linear operator. Ferwerda et al followed similar logic, and went on to include scotopic/photopic response (i.e., color sensitivity) and time adaptation [Ferwerda96].

Unfortunately, none of these approaches really solves the problem entirely, since the experience of a display with limited dynamic range will never duplicate the experience of a real environment. For example, an extremely bright source like an oncoming car on the highway will just appear as two white dots, rather than causing discomfort in the viewer [Spencer95]. We therefore need some other indication of luminance, such as the glare display shown in Figure 8 of the previous section, or a false color representation as shown in Figure 9.

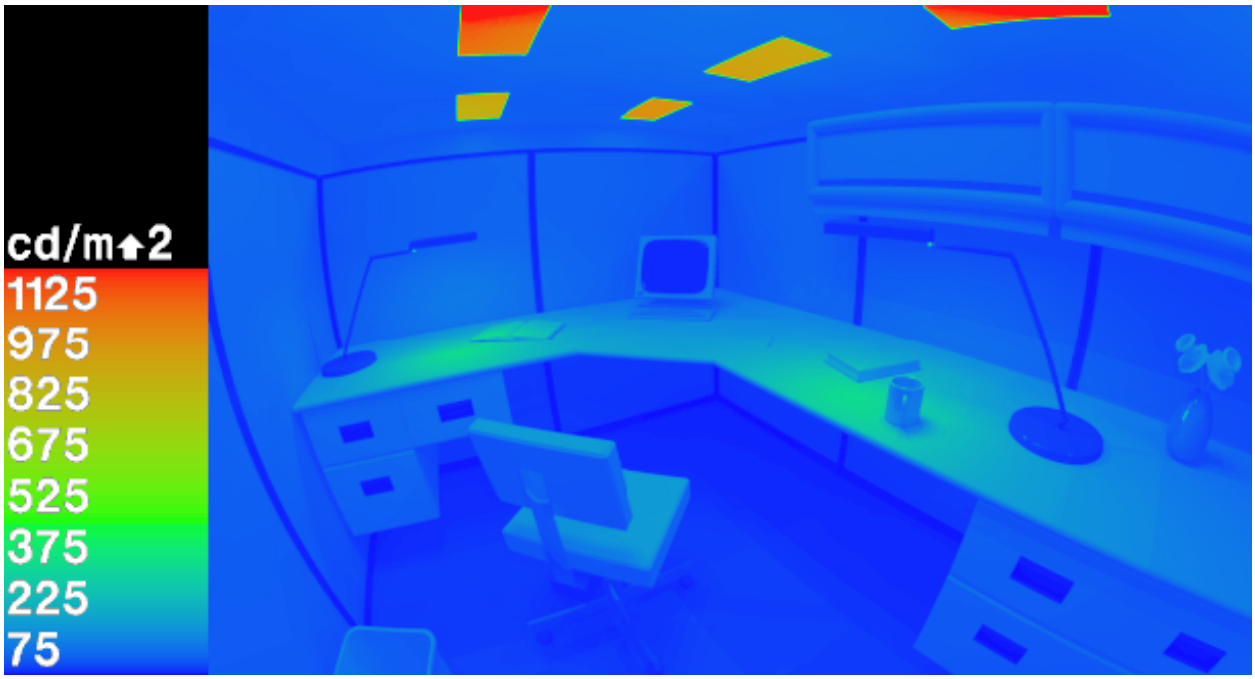


Figure 9. A "false color" image representation may be used to overcome limitations in the dynamic range of displays, replacing tone mapping with a numerical legend.

Another alternative on the horizon is high intensity VR display hardware, tied into the viewer's direction and adjusting the exposure dynamically and continuously. We will discuss this further in the final section on real-time interaction. Next, let's take a look at animation.

Animation

While plots and still images work well enough for most lighting design problems, there are occasions when the client wants to see more than can be shown in a few pictures. At its simplest, animation is merely stringing together many still images to get the illusion of motion. Sometimes, we may wish to move objects in the scene, though for lighting design, it is more likely we will want to simulate different lighting conditions (e.g., different times of day and year or different light settings). Let us discuss some of these possibilities in turn.

Camera Animation

The simplest kind of animation, and the cheapest to compute, is camera animation (also called "walk-through" animation). In this case, the scene and lighting remain static, while the viewpoint and direction move through space. In diffuse environments, such an animation can be computed in real-time for moderately complex spaces (10,000 polygons) on modern hardware. (We will touch on this again in the final section.) For very complex scenes and scenes with non-diffuse surfaces, however, other methods are necessary. Movie 1 shows a lower deck of a U.S. Navy cruiser, modeled with a few hundred thousand surfaces (which would expand to over a million polygons). A walk-through animation of this environment was computed using *Radiance* with a Z-buffer interpolation scheme similar to the one described by Chen and Williams [Chen93].

Movie 1. Walk-through animation of lower deck of U.S. Navy cruiser, computed with *Radiance* using Z-buffer frame interpolation.

A compromise approach that does not require computing multiple frames is Apple's QuickTime VR [Chen95], where a single panoramic view provides the user with a pivot point

in the environment. By computing panoramas at several positions in the design space, a certain degree of freedom is provided. We will mention this again in the final section.

Scene Animation

A favorite pastime in computer graphics, scene animation moves objects in the environment around. This is not used as much in lighting design and analysis, since it is expensive and not terribly informative.

Lighting Animation

Lighting animation shows what happens to a view as the lighting is changed over time. This is especially useful in the case of daylighting, which is highly variable throughout the day and year. Figure 10 shows four times of day for a small office with venetian blinds on the window, again computed with *Radiance*. In this study, different blind settings were examined as were algorithms for automatically controlling the blind angle throughout the day, and the results were produced on a QuickTime video. Some researchers have even optimized this type of solution using steerable filters [Nimeroff94].

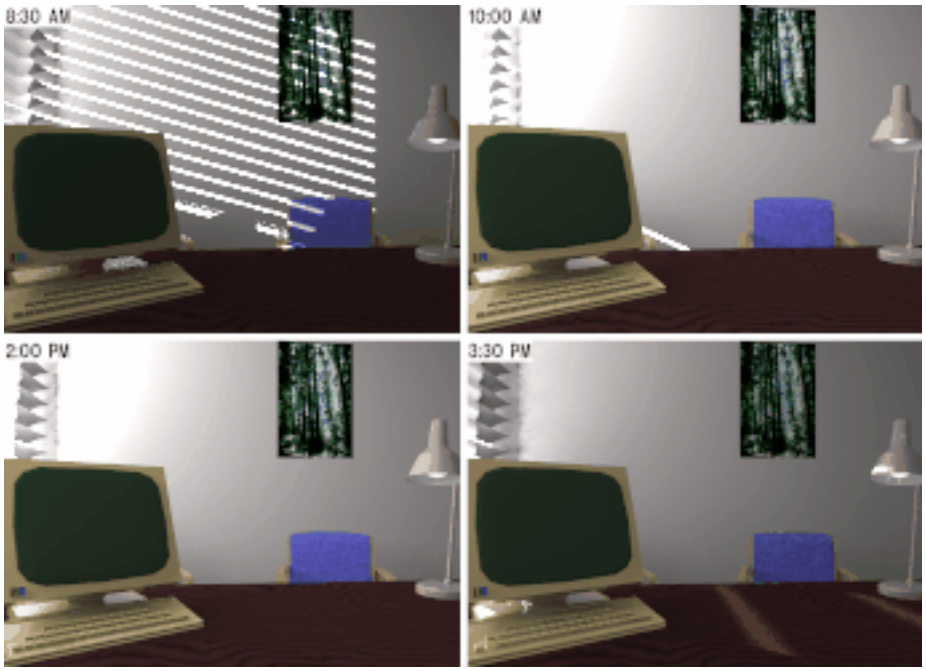


Figure 10. Four frames from a time-lapse animation of a small office with venetian blinds, computed by *Radiance*.

Real-time Interaction

Although animations are nice, they require a lot of preparation and production time, not to mention expensive video conversion equipment or, in the case of compressed video, compromises in quality. Modern Z-buffer graphics hardware is capable of displaying simple 3D spaces interactively in real-time, which is a much nicer way to view a scene. If our global illumination solution assumes diffuse surface reflection, we can use Gouraud shading to get a fairly accurate screen representation, provided the dynamic range is not too great. Figure 11 shows a radiosity rendering of an operating room, computed by *Lightscape*. The model contains a few thousand polygons, and a user may move around in this space freely on a modern graphics workstation. Displaying textures is more difficult, requiring a much more expensive class of machines for real-time interaction. Also, very complex geometry is unmanageable even for the most expensive graphics systems, leading a number of researchers to work on automatic model culling for real-time display [Funkhouser93] [Teller93].

Figure 11. Radiosity solution of operating room, computed by *Lightscape*. Real-time movement through this scene is possible on a moderately-priced graphics workstation, without textures.

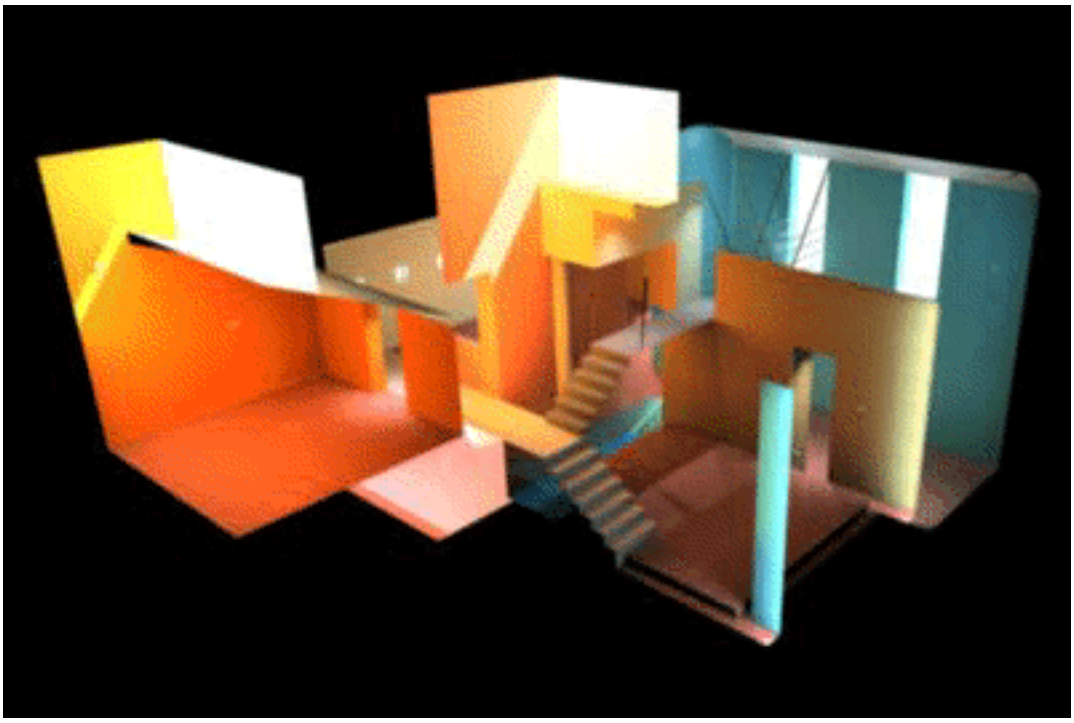


Figure 12. A cutaway view of a small, daylighted house, computed by *Lightscape*. The limited dynamic range of graphics hardware means that bright areas get washed out.

Figure 12 shows a cutaway view (back faces removed) of a small house, illuminated by daylight. This is a more interesting model to walk through, and is still simple enough to interact with in real-

time on a graphics workstation. However, the dynamic range of the space cannot be adequately represented, so some discretion must be applied in interpreting what one sees in this space. (This limitation is due primarily to the integer color computations of graphics hardware, not the radiosity solution itself, computed here by *Lightscape*.)

An even cheaper alternative to graphics hardware interaction, and one not limited to diffuse environments, is Apple's QuickTime VR [Chen95]. In this system, 360˚ cylindrical views (panoramas) are generated at specific view points throughout the environment. The user may move from one view point to the next, and at each point may look at the scene in any direction (except straight up or down). Because the view point does not move, specular interactions can be displayed correctly, and there is no limit placed on geometric complexity, though these two factors will of course influence the initial computation time. Also, the same limit in dynamic range applies to this system as well as the other hardware methods.

To bring global illumination into the world of truly visceral virtual reality, we need a wide-field, high-resolution, high dynamic range display system hooked to a real-time calculation of light transport in complex, textured, non-diffuse environments. As you might suspect, we are still some distance from attaining this goal. Even with massively parallel computer architectures, the time required to calculate specular and diffuse interactions in complex environments is too long for real-time update of high-resolution, wide-field displays. However, in the case of a head-mounted display, we know that the foveal region is the only thing that needs high resolution, and we can skimp on the rest. It may be possible, therefore, to optimize our calculation based on the viewer's gaze direction, and combine this with quick redisplay methods such as Z-buffer interpolation to reach real-time interactivity in complex environments.

This is the challenge that lies before us.

Acknowledgments

Saba Rofchaei produced most of the cruiser model shown in Movie 1 and the office model shown in Figure 11. Figure 12 was produced by Dewoolf Partnership, Architects of Rochester, NY. Figure 13 was produced by David Hileman of Toronto, Canada. The model in Figure 14 (color plate) was created by Charles Ehrlich for the Federal Aviation Administration.

References

[Chen93] Chen, Shenchang Eric, Lance Williams, "View Interpolation for Image Synthesis," *Computer Graphics Proceedings, Annual Conference Series*, 1993.

[Chen95] Chen, Shenchang Eric, "QuickTime VR -- An Image-Based Approach to Virtual Environment Navigation," *Computer Graphics Proceedings, Annual Conference Series*, 1995.

[Ferwerda96] Ferwerda, James A., Sumant Pattanaik, Peter Shirley, Donald P. Greenberg, "A Model of Visual Adaptation for Realistic Image Synthesis," *Computer Graphics Proceedings, Annual Conference Series*, 1996.

[Funkhouser93] Funkhouser, Thomas A., Carlo H. Sequin, "Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments," *Computer Graphics Proceedings, Annual Conference Series*, 1993.

[Glassner95] Glassner, Andrew S., *Principles of Digital Image Synthesis*, Appendix F., Morgan Kaufmann Publishing, 1995.

[He91] He, Xiao Dong, Kenneth E. Torrance, Francois X. Sillion, Donald P. Greenberg, "A Comprehensive Physical Model for Light Reflection," *Computer Graphics Proceedings*, Volume 25, Number 4, 1991.

[Karner96] Karner, K.F., H. Mayer, M. Gervautz, "An Image based Measurement System for Anisotropic Reflection," *EUROGRAPHICS Annual Conference Proceedings*, 1996.

[Lewis93] Lewis, Robert R., "Making Shaders More Physically Plausible," *Fourth EUROGRAPHICS Workshop on Rendering*, June 1993.

[Nimeroff94] Nimeroff, Jeffry S., Eero Simoncelli, Julie Dorsey, "Efficient Re-rendering of Naturally Illuminated Environments," *Fifth EUROGRAPHICS Workshop on Rendering*, June 1994.

[Schlick93] Schlick, Christophe, "A Customizable Reflectance Model for Everyday Rendering," *Fourth EUROGRAPHICS Workshop on Rendering*, June 1993.

[Spencer95] Spencer, Greg, Peter Shirley, Kurt Zimmerman, Donald P. Greenberg, "Physically-Based Glare Effects for Digital Images," *Computer Graphics Proceedings, Annual Conference Series*, 1995.

[Teller93] Teller, Seth, Pat Hanrahan, "Global Visibility Algorithms for Illumination Computations," *Computer Graphics Proceedings, Annual Conference Series*, 1993.

[Tumblin93] Tumblin, Jack, Holly E. Rushmeier, "Tone Reproduction for Realistic Images," *IEEE Computer Graphics and Applications*, November 1993.

[Ward92] Ward, Gregory J., "Measuring and Modeling Anisotropic Reflection," *Computer Graphics*, Volume 26, Number 2, July 1992.

[Ward94a] Ward, Greg, "A Contrast-Based Scalefactor for Luminance Display," *Graphics Gems IV*, editted by Paul Heckbert, Academic Press, 1994.

[Ward94b] Ward, Gregory J., "The RADIANCE Lighting Simulation and Rendering System," *Computer Graphics Proceedings, Annual Conference Series*, 1994.

Appendix B:

The Materials and Geometry Format

Reprinted from SIGGRAPH 1996 Course Notes

For models, materials, and further information, see website:

http://radsite.lbl.gov/mgf/

The Materials and Geometry Format

Greg Ward
Lawrence Berkeley Laboratory

1. Introduction

The Materials and Geometry Format (referred to henceforth as MGF) is a description language for 3-dimensional environments expressly suited to visible light simulation and rendering. The materials are physically-based and rely on standard and well-accepted definitions of color, reflectance and transmittance for good accuracy and reproducibility. The geometry is based on boundary representation using simple geometric primitives such as polygons, spheres and cones. The file format itself is terse but human-readable ASCII text.

1.1. What makes MGF special?

There are three principal reasons to use MGF as an input language for lighting simulation and physically-based rendering:

1. It's the only existing format that describes materials physically.
2. It is endorsed by the Illuminating Engineering Society of North America (IESNA) as part of their LM-63-1995 standard for luminaire data.
3. It's easy and fun to support since it comes with a standard parser and sample scenes and objects at the web site, "http://radsite.lbl.gov/mgf/HOME.html".

The standard parser provides both immediate and long-term benefits, since it presents a programming interface that is more stable even than the language itself. Unlike AutoCAD DXF and other de facto standards, a change to the language will not break existing programs. This is because the parser gives the calling software only those entities it can handle. If the translator understands only polygons, it will be given only polygons. If a new geometric primitive is included in a later version of the standard, the new parser that comes with it will still be able to express this entity as polygons. Thus, the urgency of modifying code to support a changing standard is removed, and long-term stability is assured.

This notion of *extensibility* is a cornerstone of the format, and it goes well beyond the extensibility of other languages because is guarantees that new versions of the standard will not break existing programs, and the new information will be used as much as possible. Other languages either require that all translators stay up to date with the latest standard, or allow forward compatibility by simply *ignoring* new entities. In MGF, if NURBS are added at some point and the translator or loader does not handle them directly, the new version of the parser will automatically convert them to smoothed polygons without changing a single line of the calling program. It is merely necessary to link to the new library, and all the new entities are supported†.

1.2. What does MGF look like?

MGF has a simple entity-per-line structure, with a similar appearance to Wavefront's .OBJ format. Each entity is specified by a short keyword, and arguments are separated by white space (tabs and/or spaces). A newline may be escaped with a backslash ('\'), in which case it counts as a space. Lines and continued lines may have up to 4096 characters, including newlines, tabs and

†If an old version of the parser encounters new entities it does not recognize, the default action is to ignore them, printing a warning message. This may be overridden to support custom entities, but such practice is discouraged because it weakens the standard.

spaces. A comment is an ignored entity whose keyword is the pound sign ('#').

Here is an MGF file that describes a simple two-drawer file cabinet:

```
# Conversion from inches to meters
xf -s .0254
# Surface material
m burgundy_formica =
        c
                cxy .362 .283
        rd .0402
        c
        rs .0284 .05
        sides 1
# Cabinet vertices
v fc.xy =
        p .05 0 0
v fc.xY =
        p .05 18 0
v fc.XY =
        p 35.95 18 0
v fc.Xy =
        p 35.95 0 0
# Cabinet
prism fc.xy fc.xY fc.XY fc.Xy 24
# Drawer vertices
v fcd.Xz =
        p 34 0 0
v fcd.XZ =
        p 34 0 10
v fcd.xZ =
        p 0 0 10
v fcd.xz =
        p 0 0 0
# Two drawers
o drawer
        xf -t 1 18.1 2 -a 2 -t 0 0 11
        prism fcd.xz fcd.Xz fcd.XZ fcd.xZ .9
        xf
o
# End of units conversion
xf
```

1.3. MGF's place in the world of standards

MGF was developed initially to support detailed geometric description of light fixtures for the IESNA luminaire data standard, publication LM-63†. Existing standards for geometric description were either too cumbersome (e.g. *Radiance)* or did not include physical materials (e.g. IGES). It was noted early on that a standard able to fully describe luminaires would necessarily be capable of describing other objects as well; indeed whole environments could be defined this way. Since the descriptions would be physical, they could serve as input to both lighting simulation and rendering software. A standard language for describing the appearance of physical objects has been lacking for some time, and current efforts in this direction (i.e. STEP) seem

†To obtain the latest version of this standard, write to: Illuminating Engineering Society of North America, 345 East 47th St., New York, NY 10017.

several years away from fruition. (There are other languages for describing realistic scenes that deserve mention here, such as VRML and the Manchester Scene Description Language, but none give specific attention to physical material properties and are thus unsuitable for lighting simulation.)

In short, we saw this as an opportunity to offer the lighting and rendering community a simple and easy-to-support standard for describing environments in a physically valid way. Our hope is that this will promote sharing color, material and object libraries as well as complete scene descriptions. Sharing libraries is of obvious benefit to users and software developers alike. Sharing scenes should also permit comparisons between rendering systems and intervalidation of lighting calculations. As anyone who works in this field knows, modeling is the most difficult step in creating any simulation or rendering, and there is no excuse for this data being held prisoner by a proprietary data format.

2. MGF Basics

The default coordinate system in MGF is right-handed with distances given in meters, though this can be effectively changed by specifying a global transformation. The transformation context is affected by the xf entity, and the whole of MGF can be understood in terms of entities and contexts.

2.1. Entities and Contexts

An *entity* in MGF is any non-blank line, which must be one of a finite set of command keywords followed by zero or more arguments. (As mentioned previously, an entity may continue over multiple lines by escaping the newline with a backslash.) Table 1 gives a list of entities and their expected arguments. Section 3 gives more detailed information on each entity.

A *context* describes the current state of the interpreter, and affects or is affected by certain entities as they are read in. MGF contexts can be divided into two types, *hierarchical contexts* and *named contexts*.

Hierarchical contexts are manipulated by a single entity and have an associated "stack" onto which new contexts are "pushed" using the entity. The last context may be "popped" by giving the entity again with no arguments. The two hierarchical contexts in MGF are the current transformation, manipulated with the xf entity, and the current object, manipulated with the o entity.

Context	Cntl. Entity	Default Value	Field Entities	Affects
Object	o	-	-	-
Transform	xf	-	-	f, fh, sph, cyl, cone, ring, torus, prism
Material	m	2-sided black	sides, rd, td, ed, rs, ts, ir	f, fh, sph, cyl, cone, ring, torus, prism
Color	c	neutral grey	cxy, cspec, cct, cmix	rd, td, ed, rs, ts
Vertex	v	(0,0,0), no normal	p, n	f, fh, sph, cyl, cone, ring, torus, prism

Table 2. MGF contexts and their related entities and default values.

Named contexts in contrast hold sets of values that are swapped in and out one at a time. There are three named contexts in MGF, the current material, the current color and the current vertex.

Keyword	Arguments	Interpretation
#	[anything ...]	a comment
o	[name]	begin/end object context
xf	[xform]	begin/end transformation context
i	pathname [xform]	include file (with transformation)
ies	pathname [-m f][xform]	include IES luminaire (with transformation)
c	[id [= [template]]]	get/set color context
cxy	x y	set CIE (x,y) chromaticity for current color
cspec	l_min l_max v1 v2 ...	set relative spectrum for current color
cct	temperature	set spectrum based on black body temperature
cmix	w1 c1 w2 c2 ...	mix named colors to make current color
m	[id [= [template]]]	get/set material context
sides	{1|2}	set number of sides for current material
rd	rho_d	set diffuse reflectance for current material
td	tau_d	set diffuse transmittance for current material
ed	epsilon_d	set diffuse emittance for current material
rs	rho_s alpha_r	set specular reflectance for current material
ts	tau_s alpha_t	set specular transmittance for current material
ir	n_real n_imag	set index of refraction for current material
v	[id [= [template]]]	get/set vertex context
p	x y z	set point position for current vertex
n	dx dy dz	set surface normal for current vertex
f	v1 v2 v3 ...	polygon using current material, spec. vertices
fh	v1 v2 v3 - ...	face with explicit holes
sph	vc radius	sphere
cyl	v1 radius v2	truncated right cylinder (open-ended)
cone	v1 rad1 v2 rad2	truncated right cone (open-ended)
prism	v1 v2 v3 ... length	truncated right prism (closed solid)
ring	vc rmin rmax	circular ring with inner and outer radii
torus	vc rmin rmax	circular torus with inner and outer radii

Table 1. MGF entities and their arguments. Arguments in brackets are optional. Arguments in curly braces mean one of the given choices must appear. Ellipsis (...) mean that any number of arguments may be given.

Each one may be associated with an identifier (any non-white sequence of printing ASCII characters beginning with a letter), and one of each is in effect at any given time. Initially, these contexts are unnamed, and invoking an unnamed context always returns to the original (default) values. (See Table 2 for a list of contexts, their related entities and defaults.)

It is easiest to think of a context as a "scratch space" where values are written by some entities and read by others. Naming a context allows us to reestablish the same scratch space later, usually for reference but it can be altered as well. Let us say we wanted to create a smooth blue plastic material with a diffuse reflectance of 20% and a specular reflectance of 4%:

```
# Establish a new material context called "blue_plastic"
m blue_plastic =
        # Reestablish a previous color context called "blue"
        c blue
        # Set the diffuse reflectance, which uses the above color
        rd .20
        # Get the unnamed color context (always starts out grey)
        c
        # Set the specular reflectance, which is uncolored
        rs .04 0
# We're done, the current material context is now "blue_plastic"
```

Note that the above assumes that we have previously defined a color context named "blue". If we forgot to do that, the above description would generate an "undefined" error. The color context affects the material context indirectly because it is read by the specular and diffuse reflectance entities, which are in turn written to the current material. It is not necessary to indent the entities that affect the material definition, but it improves readability. Note also that there is no explicit end to the material definition. As long as a context remains in effect, its contents may be altered by its field entities. This will not affect previous uses of the context, however. For example, a surface entity following the above definition will have the specified color and reflectance, and later changes to the material "blue_plastic" will have no effect on it.

Each of the three named contexts has an associated entity that controls it. The material context is controlled by the m entity, the color context is controlled by the c entity, and the vertex context is controlled by the v entity. There are exactly four forms for each entity. The first form is the keyword by itself, which establishes an unnamed context with predetermined default values. This is a useful way to set values without worrying about saving them for recall later. The second form is to give the keyword with a previously defined name. This reestablishes a prior context for reuse. The third form is to give the keyword with a name followed by an equals sign. (There must be a space between the name and the equals sign, since it is a separate argument.) This establishes a new context and assigns it the same default values as the unnamed context. The fourth and final form gives the keyword followed by a name then an equals then the name of a previous context definition. This establishes a new context for the first name, assigning the values from the second named context rather than the usual defaults. This is a convenient way create an alias or to modify a context under a new name (i.e. "save as").

2.2. Hierarchical Contexts and Transformations

As mentioned in the last subsection, there are two hierarchical contexts in MGF, the current object and the current transformation. We will start by discussing the current object, since it is the simpler of the two.

2.2.1. Objects

There is no particular need in lighting simulation or rendering to name objects, but it may help the user to know what object a particular surface is associated with. The o entity provides a convenient mechanism for associating names with surfaces. The basic use of this entity is as follows:

```
o object_name
        [object entities...]
        o subobject_name
                [subobject entities...]
        o
        [more object entities and subobjects...]
o
```

The o keyword by itself marks the end of an object context. Any number of hierarchical context

levels are supported, and there are no rules governing the choice of object names except that they begin with a letter and be made up of printing, non-white ASCII characters. Indentation is not necessary of course, but it does improve readability.

2.2.2. Transformations

MGF supports only rigid-body (i.e. non-distorting) transformations with uniform scaling. Unlike the other contexts, transformations have no associated name, only arguments. Thus, there is no way to reestablish a previous transformation other than to give the same arguments over again. Since the arguments are concise and self-explanatory, this was thought sufficient. The following transformation flags and parameters are defined:

-t dx dy dz	translate objects along the given vector
-rx degrees	rotate objects about the X-axis
-ry degrees	rotate objects about the Y-axis
-rz degrees	rotate objects about the Z-axis
-s scalefactor	scale objects by the given factor
-mx	mirror objects about the Y-Z plane
-my	mirror objects about the X-Z plane
-mz	mirror objects about the X-Y plane
-i N	repeat the following arguments N times
-a N	make an array of N geometric instances

Transform arguments have a cumulative effect. That is, a rotation about X of 20 degrees followed by a rotation about X of -50 degrees results in a total rotation of -30 degrees. However, if the two rotations are separated by some translation vector, the cumulative effect is quite different. It is best to think of each argument as acting on the included geometric objects, and each subsequent transformation argument affects the objects relative to their new position/orientation.

For example, rotating an object about its center is most easily done by translating the object back to the origin, applying the desired rotation, and translating it again back to its original position, like so:

```
# rotate an included object 20 degrees clockwise looking down
# an axis parallel to Y and passing through the point (15,0,-35)
xf -t -15 0 35 -ry -20 -t 15 0 -35
i object.mgf
xf
```

Note that the include entity, i, permits a transformation to be given with it, so the above could have been written more compactly as:

```
i object.mgf -t -15 0 35 -ry -20 -t 15 0 -35
```

Rotations are given in degrees counter-clockwise about a principal axis. That is, with the thumb of the right hand pointing in the direction of the axis, rotation follows the curl of the fingers.

The transform entity itself is cumulative, but in the reverse order to its arguments. That is, later transformations (i.e. enclosed transformations) are prepended to existing (i.e. enclosing) ones. A transform command with no arguments is used to return to the previous condition. It is necessary that transforms and their end statements ("xf" by itself) be balanced in a file, so that later or enclosing files are not affected.

Transformations apply only to geometric types, e.g. polygons, spheres, etc. Vertices and the components that go into geometry are not directly affected. This is to avoid confusion and the inadvertent multiple application of a given transformation. For example:

```
xf -t 5 0 0
v v1 =
        p 0 10 0
        n 0 0 1
xf -rx 180
# Transform now in effect is "-rx 180 -t 5 0 0"
ring v1 0 2
xf
xf
```

The final ring center is (5,-10,0) -- note that the vertex itself is not affected by the transformation, only the geometry that calls on it. The normal orientation is (0,0,-1) due to the rotation about X, which also reversed the sign of the central Y coordinate.

2.2.3. Arrays

The -a N transform specification causes the following transform arguments to be repeated along with the contents of the included objects N times. The first instance of the geometry will be in its initial location; the second instance will be repositioned according to the named transformation; the third instance will be repositioned by applying this transformation twice, and so on up to N-1 applications.

Multi-dimensional arrays may be specified with a single include entity by giving multiple array commands separated by their corresponding transforms. A final transformation may be given by preceding it with a -i 1 specification. In other words, the scope of an array command continues until the next -i or -a option.

The following MGF description places 60 spheres at a one unit spacing in a 3x4x5 array, then moves the whole thing to an origin of (15,30,45):

```
v v0 =
        p 0 0 0
xf -a 3 -t 1 0 0 -a 4 -t 0 1 0 -a 5 -t 0 0 1 -i 1 -t 15 30 45
sph v0 0.1
xf
```

Note the "-i 1" in the specification, which marks the end of the third array arguments before the final translation.

2.3. Detailed MGF Example

The following example of a simple room with a single door and six file cabinets shows MGF in action, with copious comments to help explain what’s going on.

```
# "ceiling_tile" is a diffuse white surface with 75% reflectance:
# Create new named material context and clear it
m ceiling_tile =
        # Specify one-sided material so we can see through from above
        sides 1
        # Set neutral color
        c
        # Set diffuse reflectance
        rd .75
# "stainless_steel" is a mostly specular surface with 70% reflectance:
m stainless_steel =
        sides 1
        c
        # Set specular reflectance to 50%, .08 roughness
        rs .5 .08
        # Other 20% reflectance is diffuse
        rd .2

# The following materials were measured with a spectrophotometer:
m beige_paint =
        sides 1
        # Set diffuse spectral reflectance
        c
                # Spectrum measured in 10 nm increments from 400 to 700 nm
                cspec 400 700 35.29 44.87 47.25 47.03 46.87 47.00 47.09 \\
                        47.15 46.80 46.17 46.26 48.74 51.08 51.31 51.10 \\
                        51.11 50.52 50.36 51.72 53.61 53.95 52.08 49.49 \\
                        48.30 48.75 49.99 51.35 52.75 54.44 56.34 58.00
        rd 0.5078
        # Neutral (grey) specular component
        c
        rs 0.0099 0.08000
m mottled_carpet =
        sides 1
        c
                cspec 400 700 11.23 11.28 11.39 11.49 11.61 11.73 11.88 \\
                        12.02 12.12 12.19 12.30 12.37 12.37 12.36 12.34 \\
                        12.28 12.22 12.29 12.45 12.59 12.70 12.77 12.82 \\
                        12.88 12.98 13.24 13.67 14.31 15.55 17.46 19.75
        rd 0.1245
m reddish_cloth =
        # 2-sided so we can observe it from behind
        sides 2
        c
                cspec 400 700 28.62 27.96 27.86 28.28 29.28 30.49 31.61 \\
                        32.27 32.26 31.83 31.13 30.07 29.14 29.03 29.69 \\
                        30.79 32.30 33.90 34.56 34.32 33.85 33.51 33.30 \\
                        33.43 34.06 35.26 37.04 39.41 42.55 46.46 51.00
        rd 0.3210
m burgundy_formica =
        sides 1
        c
                cspec 400 700  3.86  3.74  3.63  3.51  3.34  3.21  3.14 \\
                         3.09  3.08  3.14  3.13  2.91  2.72  2.74  2.72 \\
```

```
                2.60  2.68  3.40  4.76  6.05  6.65  6.75  6.68 \\
                6.63  6.56  6.51  6.46  6.41  6.36  6.34  6.34
        rd 0.0402
        c
        rs 0.0284 0.05000
m speckled_grey_formica =
        sides 1
        c
                cspec 400 700 30.95 44.77 51.15 52.60 53.00 53.37 53.68 \\
                        54.07 54.33 54.57 54.85 55.20 55.42 55.51 55.54 \\
                        55.46 55.33 55.30 55.52 55.81 55.91 55.92 56.00 \\
                        56.22 56.45 56.66 56.72 56.58 56.44 56.39 56.39
        rd 0.5550
        c
        rs 0.0149 0.15000

# 40' x 22' x 9' office space with no windows and one door

# All measurements are in inches, so enclose with a metric conversion:
xf -s .0254

# The room corner vertices:
v rc.xyz =
        p 0 0 0
v rc.Xyz =
        p 480 0 0
v rc.xYz =
        p 0 264 0
v rc.xyZ =
        p 0 0 108
v rc.XYz =
        p 480 264 0
v rc.xYZ =
        p 0 264 108
v rc.XyZ =
        p 480 0 108
v rc.XYZ =
        p 480 264 108

# The floor:
# Push object name
o floor
        # Get previously defined carpet material
        m mottled_carpet
        # Polygonal face using defined vertices
        f rc.xyz rc.Xyz rc.XYz rc.xYz
# Pop object name
o

# The ceiling:
o ceiling
        m ceiling_tile
        f rc.xyZ rc.xYZ rc.XYZ rc.XyZ
o
```

```
# The door outline vertices:
v do.xz =
        p 216 0 0
v do.Xz =
        p 264 0 0
v do.xZ =
        p 216 0 84
v do.XZ =
        p 264 0 84

# The walls:
o wall
        m beige_paint
        o x
                f rc.xyz rc.xYz rc.xYZ rc.xyZ
        o
        o X
                f rc.Xyz rc.XyZ rc.XYZ rc.XYz
        o
        o y
                f rc.xyz rc.xyZ rc.XyZ rc.Xyz do.Xz do.XZ do.xZ do.xz
        o
        o Y
                f rc.xYz rc.XYz rc.XYZ rc.xYZ
        o
o

# The door and jam vertices:
v djo.xz =
        p 216 .5 0
v djo.xZ =
        p 216 .5 84
v djo.XZ =
        p 264 .5 84
v djo.Xz =
        p 264 .5 0
v dji.Xz =
        p 262 .5 0
v dji.XZ =
        p 262 .5 82
v dji.xZ =
        p 218 .5 82
v dji.xz =
        p 218 .5 0
v door.xz =
        p 218 0 0
v door.xZ =
        p 218 0 82
v door.XZ =
        p 262 0 82
v door.Xz =
        p 262 0 0

# The door, jam and knob
```

```
o door
        m burgundy_formica
        f door.xz door.xZ door.XZ door.Xz
        o jam
                m beige_paint
                f djo.xz djo.xZ djo.XZ djo.Xz dji.Xz dji.XZ dji.xZ dji.xz
                f djo.xz do.xz do.xZ djo.xZ
                f djo.xZ do.xZ do.XZ djo.XZ
                f djo.Xz djo.XZ do.XZ do.Xz
                f dji.xz dji.xZ door.xZ door.xz
                f dji.xZ dji.XZ door.XZ door.xZ
                f dji.Xz door.Xz door.XZ dji.XZ
        o
        o knob
                m stainless_steel
                # Define vertices needed for curved geometry
                v kb1 =
                        p 257 0 36
                v kb2 =
                        p 257 .25 36
                        n 0 1 0
                v kb3 =
                        p 257 2 36
                # 1" diameter cylindrical base from kb1 to kb2
                cyl kb1 1 kb2
                # Ring at base of knob stem
                ring kb2 .4 1
                # Knob stem
                cyl kb2 .4 kb3
                # Spherical knob
                sph kb3 .85
        o
o

# Six file cabinets (36" wide each)
# ("filecab.inc" was given as an earlier example in Section 1.2)
o filecab.x
        # include a file as an array of three 36" apart
        i filecab.inc -t -36 0 0 -rz -90 -t 1 54 0 -a 3 -t 0 36 0
o
o filecab.X
        # the other three cabinets
        i filecab.inc -rz 90 -t 479 54 0 -a 3 -t 0 36 0
o

# End of transform from inches to meters:
xf

# The 10 recessed fluorescent ceiling fixtures
ies hlrs2gna.ies -t 1.2192 2.1336 2.74 -a 5 -t 2.4384 0 0 -a 2 -t 0 2.4384 0
```

3. MGF Entity Reference

There are currently 28 entities in the MGF specification. For ease of reference we have broken these into five categories:

1. General

#	[anything ...]	a comment
o	[name]	begin/end object context
xf	[xform]	begin/end transformation context
i	pathname [xform]	include file (with transformation)
ies	pathname [-m f][xform]	include IES luminaire (with transformation)

2. Color

c	[id [= [template]]]	get/set color context
cxy	x y	set CIE (x,y) chromaticity for current color
cspec	l_min l_max v1 v2 ...	set relative spectrum for current color
cct	temperature	set spectrum based on black body temperature
cmix	w1 c1 w2 c2 ...	mix named colors to make current color

3. Material

m	[id [= [template]]]	get/set material context
sides	{1|2}	set number of sides for current material
rd	rho_d	set diffuse reflectance for current material
td	tau_d	set diffuse transmittance for current material
ed	epsilon_d	set diffuse emittance for current material
rs	rho_s alpha_r	set specular reflectance for current material
ts	tau_s alpha_t	set specular transmittance for current material
ir	n_real n_imag	set index of refraction for current material

4. Vertex

v	[id [= [template]]]	get/set vertex context
p	x y z	set point position for current vertex
n	dx dy dz	set surface normal for current vertex

5. Geometry

f	v1 v2 v3 ...	polygon using current material, spec. vertices
fh	v1 v2 v3 - ...	face with explicit holes
sph	vc radius	sphere
cyl	v1 radius v2	truncated right cylinder (open-ended)
cone	v1 rad1 v2 rad2	truncated right cone (open-ended)
prism	v1 v2 v3 ... length	truncated right prism (closed solid)
ring	vc rmin rmax	circular ring with inner and outer radii
torus	vc rmin rmax	circular torus with inner and outer radii

NAME

- a comment

SYNOPSIS

[*anything*]

DESCRIPTION

A comment is a bit of text explanation. Since it is an entity like any other (except that it has no effect), there must be at least one space between the keyword (which is a pound sign) and the "arguments," and the end of line may be escaped as usual with the backslash character (’\’).

A comment may actually be used to hold auxiliary information such as view parameters, which may be interpreted by some destination program. Care should be taken under such circumstances that the user does not inadvertently mung or mimic this information in other comments, and it is therefore advisable to use an additional set of identifying characters to distinguish such data.

EXAMPLE

```
# The following include file is in inches, so convert to meters
i cubgeom.inc -s .0254
# Stuff we don’t want to see at the moment:
# i person.mgf -t 3 2 0
# ies hlrs3gna.ies -rz 90 -t 1.524 1.8288 2.74 \\
        -a 6 -t 1.8288 0 0 -a 2 -t 0 3.048 0
```

NAME

o - begin or end object context

SYNOPSIS

o [*name*]

DESCRIPTION

If *name* is given, we push a new object context onto the stack, which is to say that we begin a new subobject by this name†. If the o keyword is given by itself, then we pop the last object context off the stack, which means that we leave the current subobject.

All geometry between the start of an object context and its matching end statement is associated with the given name. This may be used in modeling software to help identify objects and subobjects, or it may be ignored altogether.

Object begin and end statements should be balanced in a file, and care should be taken not to overlap transform (xf) contexts with object contexts, especially when arrays are involved. This is because the standard parser will assign object contexts to instanced geometry, which can get confused with other object contexts if a clear enclosure is not maintained.

EXAMPLE

```
o body
        o torso
                i torso.mgf
        o
        o arm
                o left
                        i leftarm.mgf
                o
                o right
                        i leftarm.mgf -mx
                o
        o
o
```

SEE ALSO

xf

†A name is any sequence of printing, non-white ASCII characters beginning with a letter.

NAME

xf - begin or end transformation context

SYNOPSIS

xf [*transform*]

DESCRIPTION

If a set of *transform* arguments are given, we push a new transformation context onto the stack. If the xf keyword is given by itself, then we pop the last transformation context off the stack. The total transformation in effect at any given time is computed by prepending each set subcontext arguments onto those of its enclosing context. This and other details about transformation specifications are explained in some detail in section 2.2.2.

The following transformation flags and parameters are defined:

-t dx dy dz	translate objects along the given vector
-rx degrees	rotate objects about the X-axis
-ry degrees	rotate objects about the Y-axis
-rz degrees	rotate objects about the Z-axis
-s scalefactor	scale objects by the given factor
-mx	mirror objects about the Y-Z plane
-my	mirror objects about the X-Z plane
-mz	mirror objects about the X-Y plane
-i N	repeat the following arguments N times
-a N	make an array of N geometric instances

EXAMPLE

```
# Create 3x5 array of evenly-spaced spheres (grid size = 3)
v vc =
        p 0 0 0
xf -t 1 1 10 -a 3 -t 3 0 0 -a 5 -t 0 3 0
        sph vc .5
xf
```

SEE ALSO

i, ies, o

NAME

i - include MGF data file

SYNOPSIS

i *pathname* [*transform*]

DESCRIPTION

Include the information contained in the file *pathname.* If a *transform* specification is given, then it will be applied as though the include statement were enclosed by beginning and ending xf entities with this transformation.

The *pathname* will be interpreted relative to the enclosing MGF file. That is, if the file containing the include statement is in some parent or subdirectory, then the given pathname is appended to this directory. It is illegal to specify a *pathname* relative to the root directory, and the MGF standard requires that all filenames adhere to the ISO-9660 8.3 name format for maximum portability between systems. The directory separator is defined to be slash ('/'), and drive specifications (such as "c:") are not allowed. All pathnames should be given in lower case, and will be converted to upper case on systems that require it. (That way, there are no accidental name collisions.)

The suggested suffix for MGF-adherent files is ".mgf". Files that are not in metric units but are in MGF may be given any suffix, but we suggest using ".inc" as a convention.

EXAMPLE

```
# Define vertices for 62x30" partition
i pv62x30.inc
# Insert 2 62x30" partitions
o cpart1
        i partn.inc -t 75 130.5 0
o
o cpart3
        i partn.inc -t 186 130.5 0
o
# Define vertices for 62x36" partition
i pv62x36.inc
# Insert 62x36" partition
o cpart2
        i partn.inc -t 105 130.5 0
o
```

SEE ALSO

ies, o, xf

NAME

ies - include IESNA luminaire file

SYNOPSIS

ies *pathname* [**−m** *multiplier*] [*transform*]

DESCRIPTION

Load the IES standard luminaire information contained in the file *pathname.* If a *multiplier* is given, all candela values will be multiplied by this factor. (This option must appear first if present.) If a *transform* specification is given, then it will be applied as though the statement were enclosed by beginning and ending xf entities with this transformation.

The *pathname* will be interpreted relative to the enclosing MGF file, and all restrictions discussed under the i entity also apply to the IES file name. The suggested suffix is ".ies", but this has not been followed consistently by lighting manufacturers.

EXAMPLE

```
# Insert 10 2x4' fluorescent troffers in two groups
ies cf9pr240.ies -t 3.6576 2.1336 2.74 -a 3 -t 2.4384 0 0 -a 2 -t 0 2.4384 0
ies cf9pr240.ies -rz 90 -t 1.2192 1.8288 2.74 \\
        -a 2 -t 9.7536 0 0 -a 2 -t 0 3.048 0
```

SEE ALSO

i, o, xf

NAME

c - get or set the current color context

SYNOPSIS

c [*id* [= [*template*]]]

DESCRIPTION

If the c keyword is given by itself, then it establishes the unnamed color context, which is neutral (i.e. equal-energy) grey. This context may be modified, but the changes will not be saved.

If the keyword is followed by an identifier *id*, then it reestablishes a previous context. If the specified context was never defined, an error will result.

If the entity is given with an identifier followed by an equals sign ('='), then a new context is established, and cleared to the default neutral grey. (Note that the equals sign must be separated from other arguments by white space to be properly recognized.) If the equals sign is followed by a second identifier *template*, then this previously defined color will be used as a source of default values rather than grey. This is most useful for establishing a color alias.

EXAMPLE

```
# Define the color "red32"
c red32 =
        cxy .42 .15
# Make "cabinet_color" an alias for "red32"
c cabinet_color = red32

# Later in another part of the description...

# Get our cabinet color
c cabinet_color
# Get the geometry
i cabgeom.mgf
```

SEE ALSO

cct, cmix, cspec, cxy, m

NAME

cxy - set the CIE (x,y) chromaticity for the current color

SYNOPSIS

cxy *x y*

DESCRIPTION

This entity sets the current color using (x,y) chromaticity coordinates for the 1931 CIE standard 2 degree observer. Legal values for *x* and *y* are greater than zero and sum to less than one, and more specifically they must fit within the curve of the visible spectrum. The *x* coordinate roughly corresponds to the red part of the spectrum and the *y* coordinate corresponds to the green. The CIE z coordinate is implicit, since it is equal to (1-x-y).

All colors in MGF are absolute, thus colorimeter measurements should be conducted the same for surfaces as for light sources. Applying a standard illuminant calculation is redundant and introduces inaccuracies, and should therefore be avoided if possible.

Conversion between CIE colors and those more commonly used in computer graphics are described in the application notes section 6.1.1.

EXAMPLE

```
# Set unnamed color context
c
# Set CIE chromaticity to a bluish hue
cxy .15 .2
# Apply color to diffuse reflectance of 15%
rd .15
```

SEE ALSO

c, cct, cmix, cspec

NAME

cspec - set the relative spectrum for the current color

SYNOPSIS

cspec *l_min l_max o1 o2 ... oN*

DESCRIPTION

Assign a relative spectrum measured between *l_min* and *l_max* nanometers at evenly spaced intervals. The first value, *o1* corresponds to the measurement at *l_min*, and the last value, *oN* corresponds to the measurement at *l_max*. Values in between are separated by *(l_max-l_min)/(N-1)* nanometers. All values should be non-negative unless defining a component for complementary color mixing, and the spectrum outside of the specified range is assumed to be zero. (The visible range is 380 to 780 nm.) The actual units and scale of the measurements do not matter, since the total will be normalized according to whatever the color is modifying (e.g. photometric reflectance or emittance).

EXAMPLE

```
# Color measured at 10 nm increments from 400 to 700
m reddish_cloth =
        c
                cspec 400 700 28.62 27.96 27.86 28.28 29.28 30.49 31.61 \\
                        32.27 32.26 31.83 31.13 30.07 29.14 29.03 29.69 \\
                        30.79 32.30 33.90 34.56 34.32 33.85 33.51 33.30 \\
                        33.43 34.06 35.26 37.04 39.41 42.55 46.46 51.00
        rd 0.3210
```

SEE ALSO

c, cct, cmix, cxy

NAME

cct - set the current color to a black body spectrum

SYNOPSIS

cct *temperature*

DESCRIPTION

The cct entity sets the current color to the spectrum of an ideal black body radiating at *temperature* degrees Kelvin. This is often the most convenient way to set the color of an incandescent light source, but it is not recommended for fluorescent lamps or other materials that do not fit a black body spectrum.

EXAMPLE

```
# Define an incandescent source material at 3000 degrees K
m incand3000k =
        c
                cct 3000
        ed 1500
```

SEE ALSO

c, cmix, cspec, cxy

NAME

cmix - mix two or more named colors to make the current color

SYNOPSIS

cmix *w1 c1 w2 c2 ...*

DESCRIPTION

The cmix entity sums together two or more named colors using specified weighting coefficients, which correspond to the relative photometric brightness of each. As in all color specifications, the result is normalized so the absolute scale of the weights does not matter, only their relative values.

If any of the colors is a spectral quantity (i.e. from a cspec or cct entity), then all the colors are first converted to spectral quantities. This is done with an approximation for CIE (x,y) chromaticities, which may be problematic depending on their values. In general, it is safest to add together colors that are either all spectral quantities or all CIE quantities.

EXAMPLE

```
# Define RGB primaries for a standard color monitor
c R =
        cxy 0.640 0.330
c G =
        cxy 0.290 0.600
c B =
        cxy 0.150 0.060
# Mix them together in appropriate amounts for white
c white =
        cmix 0.265 R 0.670 G 0.065 B
```

SEE ALSO

c, cct, cspec, cxy

NAME

m - get or set the current material context

SYNOPSIS

m [*id* [= [*template*]]]

DESCRIPTION

If the m keyword is given by itself, then it establishes the unnamed material context, which is a perfect two-sided black absorber. This context may be modified, but the changes will not be saved.

If the keyword is followed by an identifier *id,* then it reestablishes a previous context. If the specified context was never defined, an error will result.

If the entity is given with an identifier followed by an equals sign ('='), then a new context is established, and cleared to the default material. (Note that the equals sign must be separated from other arguments by white space to be properly recognized.) If the equals sign is followed by a second identifier *template,* then this previously defined material will be used as a source of default values instead. This may be used to establish a material alias, or to modify an existing material and give it a new name.

The sum of the diffuse and specular reflectances and transmittances must not be greater than one (with no negative values, obviously). These values are assumed to be measured at normal incidence. If an index of refraction is given, this may modify the balance between diffuse and specular reflectance at other incident angles. If the material is one-sided (see sides entity), then it may be a dielectric interface. In this case, the specular transmittance given is that which would be measured at normal incidence for a pane of the material 5 mm thick. This is important for figuring the actual transmittance for non-planar geometries assuming a uniformly absorbing medium. (Diffuse transmittance will not be affected by thickness.) If the index of refraction has an imaginary part, then the surface is a metal and this implies other properties as well. The default index of refraction is that of a vacuum, i.e. (1,0).

EXAMPLE

```
# Define a blue enamel paint
m blue_enamel =
        c
                cxy 0.2771 0.2975
        rd 0.5011
        c
        rs 0.0100 0.0350
# Assign blue_enamel to be the color of the south wall
m swall_mat = blue_enamel
# ...
# South wall face
m swall_mat
f sv1 sv2 sv3 sv4
```

SEE ALSO

ed, ir, rd, rs, sides, td, ts

NAME

sides - set the number of sides for the current material

SYNOPSIS

sides { **1** | **2** }

DESCRIPTION

The sides entity is used to set the number of sides for the current material. If a surface is two-sided, then it will appear identical when viewed from either the front or the back. If a surface is one-sided, then it appears invisible when viewed from the back side. This means that a transmitting object will affect the light coming in through the front surface and ignore the characteristics of the back surface, unless the index of refraction is set. If the index of refraction is set, then the object will act as a solid piece of dielectric material. In either case, the transmission properties of the exiting surface should be the same as the incident surface for the model to be physically valid.

The default number of sides is two.

EXAMPLE

```
# Describe a blue crystal ball
m blue_crystal =
        ir 1.650000 0
          # Solid dielectrics must use one-sided materials
        sides 1
        c
        rs 0.0602 0
        c
              cxy 0.3127 0.2881
        ts 0.6425 0
v sc =
          p 10 15 1.5
sph sc .02
```

SEE ALSO

ed, ir, m, rd, rs, td, ts

NAME

rd - set the diffuse reflectance for the current material

SYNOPSIS

rd *rho_d*

DESCRIPTION

Set the diffuse reflectance for the current material to *rho_d* using the current color to determine the spectral characteristics. This is the fraction of visible light that is reflected from a surface equally in all directions according to Lambert's law, and is often called the "Lambertian component." Photometric reflectance is measured according to v(lambda) response function of the 1931 CIE standard 2 degree observer, and assumes an equal-energy white light source. The value must be between zero and one, and may be further restricted by the luminosity of the selected color. (I.e. it is impossible to have a violet material with a photometric reflectance close to one since the eye is less sensitive in this part of the spectrum.)

The default diffuse reflectance is zero.

EXAMPLE

```
# An off-white paint with 70% reflectance
m flat_white70 =
        c
                cxy .3632 .3420
        rd .70
```

SEE ALSO

c, ed, ir, m, rs, sides, td, ts

NAME

td - set the diffuse transmittance for the current material

SYNOPSIS

td *tau_d*

DESCRIPTION

Set the diffuse transmittance for the current material to *tau_d* using the current color to determine the spectral characteristics. This is the fraction of visible light that is transmitted through a surface equally in all (transmitted) directions. Like reflectance, transmittance is measured according to the standard v(lambda) curve, and assumes an equal-energy white light source. It is probably not possible to create a material with a diffuse transmittance above 50%, since well-diffused light will be reflected as well.

The default diffuse transmittance is zero.

EXAMPLE

```
# Model a perfect spherical diffuser, i.e. light hitting either side will be scattered equally in all directions
m wonderland_diffuser =
        c
        td .5
        rd .5
```

SEE ALSO

c, ed, ir, m, rd, rs, sides, ts

NAME

ed - set the diffuse emittance for the current material

SYNOPSIS

ed *epsilon_d*

DESCRIPTION

Set the diffuse emittance for the current material to *epsilon_d* lumens per square meter using the current color to determine the spectral characteristics. Note that this is emittance rather than exitance, and therefore does not include reflected or transmitted light, which is a function of the other material settings and the illuminated environment.

The total lumen output of a convex emitting object is the radiating area of that object multiplied by its emittance. Therefore, one can compute the appropriate *epsilon_d* value for an emitter by dividing the total lumen output by the radiating area (in square meters).

The default emittance is zero.

EXAMPLE

```
# A 100-watt incandescent bulb (1600 lumens) modeled as a sphere
m
        c
                cct 3000
        ed 87712
v cent =
        p 0 0 0
sph cent .0381
```

SEE ALSO

c, ir, m, rd, rs, sides, td, ts

NAME

rs - set the specular reflectance for the current material

SYNOPSIS

rs *rho_s alpha_r*

DESCRIPTION

Set the specular reflectance for the current material to *rho_s* using the current color to determine the spectral characteristics. The surface roughness parameter is set to *alpha_r,* which is the RMS height of surface variations over the autocorrelation distance (equivalent to RMS facet slope). A roughness value of zero means a perfectly smooth surface, and values greater than 0.2 are unusual. (See application notes section 6.1.2 for a comparison between the roughness parameter and Phong specular power.)

The default specular reflectance is zero.

EXAMPLE

```
# Define a slightly rough brass metallic surface
m rough_brass =
        c
                cxy .3820 .4035
        # 30% specular, 9% diffuse
        rs .30 .08
        rd .09
```

SEE ALSO

c, ed, ir, m, rd, sides, td, ts

NAME

ts - set the specular transmittance for the current material

SYNOPSIS

ts *tau_s alpha_t*

DESCRIPTION

Set the specular transmittance for the current material to *tau_s* using the current color to determine the spectral characteristics. The effective surface roughness is set to *alpha_t.* Rays will be transmitted with the same distribution as they would have been reflected with if this roughness value were given to the rs entity.

The default specular transmittance is zero.

EXAMPLE

```
# Define a green glass material (58% transmittance)
m glass =
        sides 2
      ir 1.52 0
      c
      rs 0.0725 0
        c
                cxy .23 .38
      ts 0.5815 0
# Define an uncolored translucent plastic (40% transmittance)
m translucent =
        sides 2
        ir 1.4 0
        c
        rs .045 0
        ts .40 .05
```

SEE ALSO

c, ed, ir, m, rd, rs, sides, td

NAME

ir - set the complex index of refraction for the current material

SYNOPSIS

ir *n_real n_imag*

DESCRIPTION

Set the index of refraction for the current material to *(n_real,n_imag).* If the material is a dielectric (as opposed to metallic), then *n_imag* should be zero. For solid dielectric objects, the material should be made one-sided. If it is being used for thin objects, then a two-sided material is appropriate. (See the sides entity.)

The default index of refraction is that of a vacuum, (1,0).

EXAMPLE

```
# Define polished aluminum material
m polished_aluminum =
        # Complex index of refraction (from physics table)
        ir .770058 6.08351
        c
        rs .75 0
```

SEE ALSO

c, ed, m, rd, rs, sides, td, ts

NAME

v - get or set the current vertex context

SYNOPSIS

v [*id* [= [*template*]]]

DESCRIPTION

If the v keyword is given by itself, then it establishes the unnamed vertex context, which is the origin with no normal. This context may be modified, but the changes will not be saved. (The unnamed vertex is never used except as a source of default values since all geometric entities call their vertices by name.)

If the keyword is followed by an identifier *id,* then it reestablishes a previous context. If the specified context was never defined, an error will result.

If the entity is given with an identifier followed by an equals sign ('='), then a new context is established, and cleared to the default vertex (the origin). (Note that the equals sign must be separated from other arguments by white space to be properly recognized.) If the equals sign is followed by a second identifier *template,* then this previously defined vertex will be used as a source of default values instead. This may be used to establish a vertex alias, or to modify an existing vertex and give it a new name.

A non-zero vertex normal must be given for certain entities, specifically ring and torus require a normal direction. An f or fh entity will interpolate vertex normals if given, and use the polygon plane normal otherwise. See the prism entry for an explanation of how it interprets and uses vertex normals. The other entities ignore vertex normals if present.

The actual position and normal direction for a vertex is determined at the time of use by a geometric entity. Specifically, the transformation in effect at the time the vertex is defined is irrelevant. The only transformation that matters is the one that is applied to the geometry itself. This prevents double-transformation of vertices and allows one set of vertices to be used for multiple purposes, e.g. the front and back sides of a drawer.

EXAMPLE

```
# Make a capped cylinder
v end1 =
        p 0 0 0
        n 0 0 -1
v end2 =
        p 0 0 1
cyl end1 1.2 end2
# Forgot normal for end2
v end2
        n 0 0 1
ring end1 0 1.2
ring end2 0 1.2
```

SEE ALSO

cone, cyl, f, fh, n, p, prism, ring, sph, torus

NAME

p - set the point location for the current vertex

SYNOPSIS

p *px py pz*

DESCRIPTION

Set the 3-dimensional position for the current vertex to *(px,py,pz)*. The actual position of the vertex will be determined by the transformation in effect at the time the vertex is applied to a geometric surface entity. The transform current when the position is set is irrelevant.

The default vertex position is the origin, (0,0,0).

EXAMPLE

```
# Make a small circle of 6 spheres
v scent =
        p 1 0 0
xf -a 6 -rz 60
        sph scent .05
xf
```

SEE ALSO

cone, cyl, f, fh, n, prism, ring, sph, torus, v

NAME

n - set the surface normal direction for the current vertex

SYNOPSIS

n *dx dy dz*

DESCRIPTION

Set the 3-dimensional surface normal for the current vertex to the normalized vector along *(dx,dy,dz)*. If this vector is zero, then the surface normal is effectively unset. The actual surface normal orientation of the vertex will be determined by the transformation in effect at the time the vertex is applied to a geometric surface entity. The current transform when the normal is set is irrelevant.

The default vertex normal is the zero vector (i.e. no normal).

EXAMPLE

```
# Make a chain of 10 interlocking doughnuts
v tcent =
        p 0 0 0
        n 0 1 0
xf -a 10 -rx 90 -t .2 0 0
        torus tcent .1 .2
xf
```

SEE ALSO

f, fh, p, prism, ring, torus, v

NAME

f - create an N-sided polygonal face

SYNOPSIS

f *v1 v2 ... vN*

DESCRIPTION

Create a polygonal face made of the current material by connecting the named vertices in order, and connecting the last vertex to the first. There must be at least three vertices, and if any vertex is undefined, an error will result.

The surface orientation is determined by the right-hand rule; when the curl of the fingers follows the given order of the vertices, the surface normal points in the thumb direction. Face vertices should be coplanar, though this is difficult to guarantee in a 3-dimensional specification.

If any vertices have associated surface normals, they will be used instead of the average plane normal, though it is safest to specify either all normals or no normals, and to stick with triangles when normals are used. Also, specified normals should point in the general direction of the surface for best results.

There is no explicit representation of holes in this entity, but see the fh entity for an alternative specification.

A hole may be represented implicitly in a face entity by connecting vertices to form "seams." For example, a wall with a window in it might look as shown in Figure 1. In many systems, the wall itself would be represented with the first list of vertices, (v1,v2,v3,v4) and the hole associated with that wall as a second set of vertices (v5,v6,v7,v8). Using the face entity, we must give the whole thing as a single polygon, connecting the vertices so as to create a "seam," as shown in Figure 2. This could be written as "f v1 v2 v3 v4 v5 v6 v7 v8 v5 v4".

It is very important that the order of the hole be opposite to the order of the outer perimeter, otherwise the polygon will be "twisted" on top of itself. Note also that the seam was traversed in both directions, once going from v4 to v5, and again returning from v5 to v4. This is a necessary condition for a proper seam.

The choice of vertices to make into a seam is somewhat arbitrary, but some rendering systems may not give sane results if you cross over a hole with part of your seam. If we had chosen to create the seam between v2 and v5 in the above example instead of v4 and v5, the seam would cross our hole and may not render correctly†.

†For systems that are sensitive to this, it is probably safest for their MGF loader/translator to re-expresses seams in terms of holes again, which can be done easily so long as vertices are shared in the fashion shown.

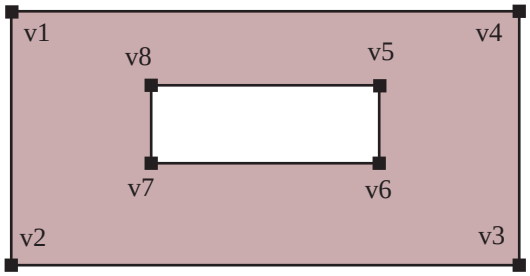


Figure 1. A wall face with a hole for a window.

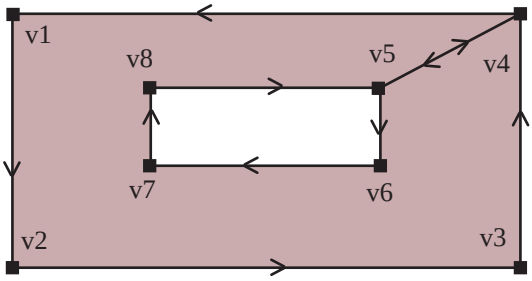


Figure 2. Connections between vertices. Note that edges coincide along "seam" between *v4* and *v5*.

EXAMPLE

```
# Make a pyramid
v apex =
        p 1 1 1
v base0 =
        p 0 0 0
v base1 =
        p 0 2 0
v base2 =
        p 2 2 0
v base3 =
        p 2 0 0
# Bottom
f base0 base1 base2 base3
# Sides
f base0 apex base1
f base1 apex base2
f base2 apex base3
f base3 apex base0
```

SEE ALSO

cone, cyl, fh, m, prism, ring, sph, torus, v

NAME

fh - create a polygonal face with explicit holes

SYNOPSIS

fh *p1 p2 ... - h1.1 h1.2 ... - h2.1 h2.2 ...*

DESCRIPTION

Create a polygonal face with optional holes made of the current material. The first contour is the outer perimeter, with vertices given in counter-clockwise order as seen from the front side (the same as the f entity). A hole is indicated by a hyphen ('-') followed by the hole's vertices, given in clockwise order as seen from the front side. Multiple hole contours are separated by additional hyphens. There must be at least three vertices for each contour, and the last vertex is implicitly connected to the first. If any vertex is undefined, an error will result.

If any vertices have associated surface normals, they will be used instead of the average plane normal, though it is safest to specify either all normals or no normals, and to stick with triangles when normals are used. Also, specified normals should point in the general direction of the surface for best results.

Vertices should not be shared between any two contours. I.e., a hole should not share a vertex or edge with the perimeter or another hole, or incorrect rendering may result.

EXAMPLE

```
# Make a wall with a window using an explicit hole.
# (See Figures 1 and 2.)
fh v1 v2 v3 v4 - v5 v6 v7 v8
```

SEE ALSO

cone, cyl, f, m, prism, ring, sph, torus, v

NAME

sph - create a sphere

SYNOPSIS

sph *vc rad*

DESCRIPTION

Create a sphere made of the current material with its center at the named vertex *vc* and a radius of *rad*. If the vertex is undefined an error will result.

The surface normal is usually directed outward, but will be directed inward if the given radius is negative. (This typically matters only for one-sided materials.) A zero radius is illegal.

EXAMPLE

```
# Create a thick glass sphere with a hollow inside
m glass =
        sides 1
        ir 1.52 0
        c
        rs .06 0
        ts .88 0
v cent =
        p 0 0 1.1
# The outer shell
sph cent .1
# The inner bubble
sph cent -.08
```

SEE ALSO

cone, cyl, f, fh, m, prism, ring, torus, v

NAME

cyl - create an open-ended, truncated right cylinder

SYNOPSIS

cyl *v1 rad v2*

DESCRIPTION

Create a truncated right cylinder of radius *rad* using the current material, starting at the named vertex *v1* and continuing to *v2*. The ends will be open, but may be capped using the ring entity if desired.

The surface normal will usually be directed outward, but may be directed inward by giving a negative value for *rad*. A zero radius is illegal, and *v1* cannot equal *v2*.

EXAMPLE

```
# A stylus with one rounded and one pointed end
o stylus
        v vtip0 =
                p 0 0 0
        v vtip1 =
                p 0 0 .005
        v vend =
                p 0 0 .05
        cyl vtip1 .0015 vend
        sph vend .0015
        cone vtip0 0 vtip1 .0015
o
```

SEE ALSO

cone, f, fh, m, prism, ring, sph, torus, v

NAME

cone - create an open-ended, truncated right cone

SYNOPSIS

cone *v1 rad1 v2 rad2*

DESCRIPTION

Create a truncated right cone using the current material. The starting radius is *rad1* at *v1* and the ending radius is is *rad2* at *v2*. The ends will be open, but may be capped using the ring entity if desired.

The surface normal will usually be directed outward, but may be directed inward by giving negative values for both radii. (It is illegal for the signs of the two radii to disagree.) One but not both radii may be zero, indicating that the cone comes to a point.

Although it is not strictly forbidden to have equal cone radii, the cyl entity should be used in such cases. Likewise, the ring entity must be used if *v1* and *v2* are equal.

EXAMPLE

```
# A parasol
o parasol
        v v1 =
                p 0 0 0
        v v2 =
                p 0 0 .75
        v v3 =
                p 0 0 .7
        m handle_mat
        cyl v1 .002 v2
        m parasol_paper
        cyl v2 0 v3 .33
o
```

SEE ALSO

cyl, f, fh, m, prism, ring, sph, torus, v

NAME

prism - create a closed right prism

SYNOPSIS

prism *v1 v2 ... vN length*

DESCRIPTION

Create a closed right prism using the current material. One end face will be enclosed by the named vertices, and the opposite end face will be a mirror image at a distance *length* from the original. The edges will be extruded into N quadrilaterals connecting the two end faces.

The order of vertices determines the original face orientation according to the right-hand rule as explained for the f entity. Normally, the prism is extruded in the direction opposite to the original surface normal, resulting in faces that all point outward. If the specified *length* is negative, the prism will be extruded above the original face and all surface normals will point inward.

If the vertices have associated normals, they are applied to the side faces only, and should generally point in the appropriate direction (i.e. in or out depending on whether *length* is negative or positive).

EXAMPLE

```
# Make a unit cube starting at the origin and \\
        extending to the positive octant
v cv0 =
        p 0 0 0
v cv1 =
        p 0 1 0
v cv2 =
        p 1 1 0
v cv3 =
        p 1 0 0
# Right hand rule has original face looking in -Z direction
prism cv0 cv1 cv2 cv3 1
```

SEE ALSO

cyl, cone, f, fh, m, ring, sph, torus, v

NAME

ring - create a circular ring with inner and outer radii

SYNOPSIS

ring *vc rmin rmax*

DESCRIPTION

Create a circular face of the current material centered on the named vertex *vc* with an inner radius of *rmin* and an outer radius of *rmax*. The surface orientation is determined by the normal vector associated with *vc*. If this vertex is undefined or has no normal, an error will result. The minimum radius may be equal to but not less than zero, and the maximum radius must be strictly greater than the minimum.

EXAMPLE

```
# The proverbial brass ring
o brass_ring
        m brass
        v end1 =
                p 0 -.005 0
                n 0 -1 0
        v end2 =
                p 0 .005 0
                n 0 1 0
        ring end1 .02 .03
        cyl end1 .03 end2
        ring end2 .02 .03
        cyl end2 -.02 end1
o
```

SEE ALSO

cyl, cone, f, fh, m, prism, sph, torus, v

NAME

torus - create a regular torus

SYNOPSIS

torus *vc rmin rmax*

DESCRIPTION

Create a torus of the current material centered on the named vertex *vc* with an inner radius of *rmin* and an outer radius of *rmax*. The plane of the torus will be perpendicular to the normal vector associated with *vc*. If this vertex is undefined or has no normal, an error will result.

If a torus with an inward facing surface normal is desired, *rmin* and *rmax* may be negative. The minimum radius may be zero, but may not be negative when *rmax* is positive or vice versa. The magnitude or *rmax* must always be strictly greater than that of *rmin.*

EXAMPLE

```
# The proverbial brass ring -- easy grip version
o brass_ring
        m brass
        v center =
                p 0 0 0
                n 0 1 0
        torus center .02 .03
o
```

SEE ALSO

cyl, cone, f, fh, m, prism, ring, sph, v

4. MGF Translators

Initially, there are six translators for MGF data, and three of these are distributed with the MGF parser itself, *mgfilt*, *mgf2inv* and *3ds2mgf*. Two of the other translators, *mgf2rad* and *rad2mgf* convert between MGF and the Radiance scene description language, and are distributed for free with the rest of the Radiance package†. The sixth translator, *mgf2meta*, converts to a 2-dimensional line plot, and is also distributed with Radiance.

Mgfilt is a simple but useful utility that takes MGF on its input and produces MGF on its output. It uses the parser to convert entities that are not wanted or understood, and produces only the requested ones. This is useful for seeing what exactly a program must understand when it supports a given set of entities, and may serve as a substitute for linking to the parser library for programmers who wish to interpret the ASCII input directly but without all the unwanted entities. In future releases of MGF, this utility will also be handy for taking new entities and producing older versions of MGF for translators that have not yet been updated properly.

Mgf2inv converts from MGF to Inventor or VRML format. Some information is lost, because these formats do not support physical light sources or materials.

3ds2mgf converts from 3D Studio binary format to MGF. Care must be taken to correct for errors in the material descriptions, since 3D Studio is completely non-physical.

---

†Radiance is available by anonymous ftp from hobbes.lbl.gov and nestor.epfl.ch, or by WWW from "http://radsite.lbl.gov/radiance/HOME.html"

NAME

mgfilt - get usable MGF entities from input

SYNOPSIS

mgfilt version [**input**..]
or
mgfilt e1,e2,.. [**input**..]

DESCRIPTION

Mgfilt takes one or more MGF input files and converts all the entities to the types listed. In the first form, a single integer is given for the *version* of MGF that is to be produced. Since MGF is in its first major release, this is not yet a useful form, but it will be when the second major release comes out. This has the necessary side-effect of expanding all included files. (See the i entity.)

In the second form, *mgfilt* produces only the entities listed in the first argument, which must be comma-separated. The listed entity order is not important, but all entities given must be defined in the current version of MGF. Unknown entities will be summarily discarded on the input, and a warning message will be printed to the standard error.

EXAMPLES

To take an MGF version 3 file and send it to a version 2 translator:

```
mgfilt 2 input.mgf | mgf2rad > input.rad
```

To take an MGF file and produce only flat polygonal faces with no materials:

```
mgfilt f,v,p,xf input.mgf > flatpoly.mgf
```

SEE ALSO

i, mgf2inv, mgf2rad, rad2mgf

NAME

mgf2inv - convert from MGF to Inventor or VRML format

SYNOPSIS

mgf2inv [**-1**|**-2**|**-vrml**] [**input**..]

DESCRIPTION

Mgf2inv takes one or more MGF input files and converts it to Inventor or VRML format. If the *−1* option is used, then Inventor 1.0 ASCII output is produced. If the *−2* option is used, then Inventor 2.0 ASCII output is produced. (This is the default.) If the *−vrml* option is used, then VRML 1.0 ASCII output is produced.

This converter does not work properly for light sources, since the output formats do not support IES-type luminaires with recorded distributions. Also, some material information may be lost because Inventor lacks a physically valid reflectance model.

EXAMPLES

To take an MGF file and convert it to VRML format:

```
mgf2inv -vrml myscene.mgf > myscene.iv
```

SEE ALSO

mgf2rad(1), mgfilt(1), 3ds2mgf(1), rad2mgf(1)

NAME

3ds2mgf - convert 3D Studio binary file to Materials and Geometry Format

SYNOPSIS

3ds2mgf input [**output**] [**-lMatlib**][**-xObjname**][**-sAngle**][**-aAnimfile**][**-fN**]

DESCRIPTION

3ds2mgf converts a 3D Studio binary scene description to the Materials and Geometry Format (MGF). If no output file name is given, the input root name will be taken as the output root, and an "mgf" extension will be added. This file will contain any light sources and materials, and an include statement for a similarly named file ending in "inc", which will contain the MGF geometry of all the translated 3DS meshes.

The MGF material names and properties for the surfaces will be those assigned in 3D Studio, unless they are named in one or more MGF material libraries given in a *-l* option.

The *-x* option may be used to exclude a named object from the output.

The *-s* option may be used to adjust automatic mesh smoothing such that adjacent triangle faces with less than the given angle between them (in degrees) will be smoothed. A value of zero turns smoothing off. The default value is 60 degrees.

The *-a* option may be used to specify a 3D Studio animation file, and together with the *-f* option, *3ds2mgf* will generate a scene description for the specified frame.

Note that there are no spaces between the options and their arguments.

LIMITATIONS

Obviously, since 3D Studio has no notion of physical materials, the translation to MGF material descriptions is very ad hoc, and it will usually be necessary to edit the materials and light sources in the output file or replace materials with proper entries from a material library using the *-l* option.

With smoothing turned on (i.e., a non-zero value for the *-s* option), vertices in the MGF output will not be linked in a proper mesh for each object. This is due to the way the automatic smoothing code was originally written, and is too difficult to repair. If a good mesh is needed, then smoothing must be turned off.

EXAMPLES

To convert a 3D Studio robot model to MGF without smoothing. (Output will be put into "robot.mgf" and "robot.inc".)

```
3ds2mgf robot.3ds -s0
```

To convert a DC10 jet model to MGF using a hand-created material library:

```
3ds2mgf dc10.3ds -ldc10mat.mgf
```

AUTHORS

Steve Anger, Jeff Bowermaster and Greg Ward
Extended from 3ds2pov 1.8.

SEE ALSO

mgf2inv(1), mgf2meta(1), mgf2rad(1)

NAME

mgf2rad - convert Materials and Geometry Format file to RADIANCE description

SYNOPSIS

mgf2rad [**−m matfile**][**−e mult**][**−g dist**] [**input**..]

DESCRIPTION

Mgf2rad converts one or more Materials and Geometry Format (MGF) files to a RADIANCE scene description. By definition, all output dimensions are in meters. The material names and properties for the surfaces will be those assigned in MGF. Any materials not defined in MGF will result in an error during translation. Light sources are described inline as IES luminaire files, and *mgf2rad* calls the program *ies2rad(1)* to translate these files. If an IES file in turn contains an MGF description of the local fixture geometry, this may result in a recursive call to *mgf2rad*, which is normal and should be transparent. The only side-effect of this additional translation is the appearance of other RADIANCE scene and data files produced automatically by *ies2rad.*

The *−m* option may be used to put all the translated materials into a separate RADIANCE file. This is not always advisable, as any given material name may be reused at different points in the MGF description, and writing them to a separate file loses the contextual association between materials and surfaces. As long as unique material names are used throughout the MGF description and material properties are not redefined, there will be no problem. Note that this is the only way to get all the translated materials into a single file, since no output is produced for unreferenced materials; i.e. translating just the MGF materials does not work.

The *−e* option may be used to multiply all the emission values by the given *mult* factor. The *−g* option may be used to establish a glow distance (in meters) for all emitting surfaces. These two options are employed principally by *ies2rad*, and are not generally useful to most users.

EXAMPLES

To translate two MGF files into one RADIANCE materials file and one geometry file:

```
mgf2rad -m materials.rad building1.mgf building2.mgf > building1+2.rad
```

To create an octree directly from two MGF files and one RADIANCE file:

```
oconv '\!mgf2rad materials.mgf scene.mgf' source.rad > scene.oct
```

FILES

tmesh.cal	Used to smooth polygonal geometry
*.rad	RADIANCE source descriptions created by ies2rad
*.dat	RADIANCE source data created by ies2rad
source.cal	Used for IES source coordinates

AUTHOR

Greg Ward

SEE ALSO

ies2rad(1), mgf2meta(1), obj2rad(1), oconv(1), rad2mgf(1), xform(1)

NAME

rad2mgf - convert RADIANCE scene description to Materials and Geometry Format

SYNOPSIS

rad2mgf [**−dU**] [**input**..]

DESCRIPTION

Rad2mgf converts one or more RADIANCE scene files to the Materials and Geometry Format (MGF). Input units are specified with the *−mU* option, where *U* is one of 'm' (meters), 'c' (centimeters), 'f' (feet) or 'i' (inches). The assumed unit is meters, which is the required output unit for MGF (thus the need to know). If the input dimensions are in none of these units, then the user should apply *xform(1)* with the *−s* option to bring the units into line prior to translation.

The MGF material names and properties for the surfaces will be those assigned in RADIANCE. If a referenced material has not been defined, then its name will be invoked in the MGF output without definition, and the description will be incomplete.

LIMITATIONS

Although MGF supports all of the geometric types and the most common material types used in RADIANCE, there is currently no support for advanced BRDF materials, patterns, textures or mixtures. Also, the special types "source" and "antimatter" are not supported, and all light source materials are converted to simple diffuse emitters (except "illum" materials, which are converted to their alternates). These primitives are reproduced as comments in the output and must be replaced manually if necessary.

The RADIANCE "instance" type is treated specially. *Rad2mgf* converts each instance to an MGF include statement, using the corresponding transformation and a file name derived from the octree name. (The original octree suffix is replaced by ".mgf".) For this to work, the user must separately create the referenced MGF files from the original RADIANCE descriptions. The description file names can usually be determined using the *getinfo(1)* command run on the octrees in question.

EXAMPLES

To convert three RADIANCE files (in feet) to one MGF file:

```
mgf2rad -df file1.rad file2.rad file3.rad > scene.mgf
```

To translate a RADIANCE materials file to MGF:

```
mgf2rad materials.rad > materials.mgf
```

AUTHOR

Greg Ward

SEE ALSO

getinfo(1), ies2rad(1), mgf2meta(1), mgf2rad(1), obj2rad(1), oconv(1), xform(1)

NAME

mgf2meta - convert Materials and Geometry Format file to Metafile graphics

SYNOPSIS

mgf2meta [**-t threshold**] **{x|y|z} xmin xmax ymin ymax zmin zmax** [**input**..]

DESCRIPTION

Mgf2meta converts one or more Materials and Geometry Format (MGF) files to a 2-D orthographic projection along the selected axis in the *metafile(1)* graphics format. All geometry is clipped to the specified bounding box, and the resulting orientation is as follows:

Projection	Orientation
x	Y-axis right, Z-axis up
y	Z-axis right, X-axis up
z	X-axis right, Z-axis up

If multiple input files are given, the first file prints in black, the second prints in red, the third in green and the fourth in blue. If more than four input files are given, they cycle through the colors again in three other line types: dashed, dotted and dot-dashed.

The *−t* option may be used to randomly throw out line segments that are shorter than the given *threshold* (given as a fraction of the plot width). Segments are included with a probability equal to the square of the line length over the square of the threshold. This can greatly reduce the number of lines in the drawing (and therefore improve the drawing speed) with only a modest loss in quality. A typical value for this parameter is 0.005.

All MGF material information is ignored on the input.

EXAMPLES

To project two MGF files along the Z-axis and display them under X11:

```
mgf2meta z 0 10 0 15 0 9 building1.mgf building2.mgf | x11meta
```

To convert a RADIANCE scene to a line drawing in RADIANCE picture format:

```
rad2mgf scene.rad | mgf2meta x ‘getbbox -h scene.rad‘ | meta2tga | ra_t8 -r > scene.pic
```

AUTHOR

Greg Ward

SEE ALSO

getbbox(1), meta2tga(1), metafile(5), mgf2rad(1), pflip(1), protate(1), psmeta(1), ra_t8(1), rad2mgf(1), t4014(1), x11meta(1)

5. MGF Parser Library

The principal motivation for creating a standard parser library for MGF is to make it easy for software developers to offer some base level of compliance. The key to making MGF easy to support in fact is the parser, which has the ability to express higher order entities in terms of lower order ones. For example, tori are part of the MGF specification, but if a given program or translator does not support them, the parser will convert them to cones. If cones are not supported either, it will convert them further into smoothed polygons. If smoothing (vertex normal information) is not supported, it will be ignored and the program will just get flat polygons. This is done in such a way that future versions of the standard may include new entities that old software does not even have to know about, and they will be converted appropriately. Forward compatibility is thus built right into the parser loading mechanism itself -- the programmer simply links to the new code and the new standard is supported without any further changes.

Language

The provided MGF parser is written in ANSI-C. This language was chosen for reasons of portability and efficiency. Almost all systems support some form of ANSI-compatible C, and many languages can cross-link to C libraries without modification. Backward compatibility to Kernighan and Ritchie C is achieved by compiling with the -DNOPROTO flag.

All of the data structures and prototypes needed for the library are in the header file "parser.h". This file is the best resource for the parser and is updated with each MGF release.

Mechanism

The parser works by a simple callback mechanism to routines that actually interpret the individual entities. Some of these routines will belong to the calling program, and some will be entity support routines included in the library itself. There is a global array of function pointers, called *mg_ehand.* It is defined thus:

```
extern int    (*mg_ehand[MG_NENTITIES])(int argc, char **argv);
```

Before parsing begins, this dispatch table is initialized to point to the routines that will handle each supported entity. Every entity handler has the same basic prototype, which is the same as the *main* function, i.e:

```
extern int    handler(int argc, char **argv);
```

The first argument is the number of words in the MGF entity (counting the entity itself) and the second argument is an array of nul-terminated strings with the entity and its arguments. The function should return zero or one of the error codes defined in "parser.h". A non-zero return value causes the parser to abort, returning the error up through its call stack to the entry function, usually *mg_load.*

A special function pointer for undefined entities is defined as follows:

```
extern int    (*mg_uhand)(int argc, char **argv);
```

By default, this points to the library function *mg_defuhand,* which prints an error message on the first unknown entity and keeps a count from then on, which is stored in the global unsigned integer *mg_nunknown.* If the *mg_uhand* pointer is assigned a value of NULL instead, parsing will abort at the first unrecognized entity. The reason this is not the default action is that ignoring unknown entities offers a certain base level of forward compatibility. Ignoring things one does not understand is not the best approach, but it is usually better than quitting with an error message if the input is in fact valid, but is a later version of the standard. The real solution is to update the interpreter by linking to a new version of the parser, or use a new version of the *mgfilt* command to convert the new MGF input to an older standard.

The *mg_uhand* pointer may also be used to customize the language for a particular application by adding entities, though this is discouraged because it tends to weaken the standard.

The skeletal framework for an MGF loader or translator is to assign function pointers to the *mg_ehand* array, call the parser initialization function *mg_init*, then call the file loader function *mg_load* once for each input file. This will in turn make calls back to the functions assigned to *mg_ehand*. To give a simple example, let us look at a translator that understands only flat polygonal faces, putting out vertex locations immediately after each "face" keyword:

```
#include <stdio.h>
#include "parser.h"

int
myfaceh(ac, av)                 /* face handling routine */
int     ac;
char    **av;
{
        C_VERTEX        *vp;    /* vertex structure pointer */
        FVECT       vert;       /* vertex point location */
        int     i;

        if (ac < 4)             /* check # arguments */
                return(MG_EARGC);
        printf("face\\n");      /* begin face output */
        for (i = 1; i < ac; i++) {
                if ((vp = c_getvert(av[i])) == NULL) /* vertex from name */
                        return(MG_EUNDEF);
                xf_xfmpoint(vert, vp->p);               /* apply transform */
                printf("%15.9f %15.9f %15.9f\\n",
                        vert[0], vert[1], vert[2]);     /* output vertex */
        }
        printf(";\\n");                 /* end of face output */
        return(MG_OK);                  /* normal exit */
}

main(argc, argv)            /* translate MGF file(s) */
int     argc;
char    **argv;
{
        int     i;
                                /* initialize dispatch table */
        mg_ehand[MG_E_FACE] = myfaceh;          /* ours */
        mg_ehand[MG_E_VERTEX] = c_hvertex;      /* parser lib */
        mg_ehand[MG_E_POINT] = c_hvertex;       /* parser lib */
        mg_ehand[MG_E_XF] = xf_handler;         /* parser lib */
        mg_init();                      /* initialize parser */
        for (i = 1; i < argc; i++)      /* load each file argument */
                if (mg_load(argv[i]) != MG_OK)      /* and check for error */
                        exit(1);
        exit(0);                        /* all done! */
}
```

Hopefully, this example demonstrates just how easy it is to write an MGF translator. Of course, translators get more complicated the more entity types they support, but the point is that one does not *have* to support every entity -- the parser handles what the translator does not. Also, the library includes many general entity handlers, further reducing the burden on the programmer.

This same principle means that it is not necessary to modify an existing program to accommodate a new version of MGF -- one need only link to the new parser library to comply with the new standard.

Division of Labor

As seen in the previous example, there are two parser routines that are normally called directly in an MGF translator or loader program. The first is *mg_init,* which takes no arguments but relies on the program having initialized those parts of the global *mg_ehand* array it cares about. The second routine is *mg_load,* which is called once on each input file. (A third routine, *mg_clear,* may be called to free the parser data structures after each file or after all files, if the program plans to continue rather than exit.)

The rest of the routines in a translator or loader program are called indirectly through the *mg_ehand* dispatch table, and they are the ones that do the real work of supporting the MGF entities. In addition to converting or discarding entities that the calling program does not know or care about, the parser library includes a set of context handlers that greatly simplify the translation process. There are three handlers for each of the three named contexts and their constituents, and two handlers for the two hierarchical context entities. To use these handlers, one simply sets the appropriate positions in the *mg_ehand* dispatch table to point to these functions. Additional functions and global data structures provide convenient access to the relevant contexts, and all of these are detailed in the following manual pages.

NAME

mg_init, mg_ehand, mg_uhand - initialize MGF entity handlers

SYNOPSIS

#include "parser.h"

void mg_init(**void**)

int mg_defuhand(**int** argc, **char** **argv)

extern int (*mg_ehand[MG_NENTITIES])(**int** argc, **char** **argv)

extern int (*mg_uhand)(**int** argc, **char** **argv)

extern unsigned mg_nunknown

DESCRIPTION

The parser dispatch table, *mg_ehand* is initially set to all NULL pointers, and it is the duty of the calling program to assign entity handler functions to each of the supported entity positions in the array. The entities are given in the include file "parser.h" as the following:

```
#define MG_E_COMMENT    0       /* #       */
#define MG_E_COLOR      1       /* c       */
#define MG_E_CCT        2       /* cct     */
#define MG_E_CONE       3       /* cone    */
#define MG_E_CMIX       4       /* cmix    */
#define MG_E_CSPEC      5       /* cspec   */
#define MG_E_CXY        6       /* cxy     */
#define MG_E_CYL        7       /* cyl     */
#define MG_E_ED         8       /* ed      */
#define MG_E_FACE       9       /* f       */
#define MG_E_INCLUDE    10      /* i       */
#define MG_E_IES        11      /* ies     */
#define MG_E_IR         12      /* ir      */
#define MG_E_MATERIAL   13      /* m       */
#define MG_E_NORMAL     14      /* n       */
#define MG_E_OBJECT     15      /* o       */
#define MG_E_POINT      16      /* p       */
#define MG_E_PRISM      17      /* prism   */
#define MG_E_RD         18      /* rd      */
#define MG_E_RING       19      /* ring    */
#define MG_E_RS         20      /* rs      */
#define MG_E_SIDES      21      /* sides   */
#define MG_E_SPH        22      /* sph*/
#define MG_E_TD         23      /* td      */
#define MG_E_TORUS      24      /* torus   */
#define MG_E_TS         25      /* ts      */
#define MG_E_VERTEX     26      /* v       */
#define MG_E_XF         27      /* xf      */

#define MG_NENTITIES    28      /* total # entities */
```

Once the *mg_ehand* array has been set by the program, the *mg_init* routine must be called to complete the initialization process. This should be done once and only once per invocation, before any other parser routines are called.

The *mg_uhand* variable points to the current handler for unknown entities encountered on the input. Its default value points to the *mg_defuhand* function, which simply increments the global variable *mg_nunknown*, printing a warning message on the standard error on the first offense. (This message may be avoided by incrementing *mg_nunknown* before processing begins.) If *mg_uhand* is assigned a value of NULL, then an unknown entity will return an *MG_EUNK* error, which will cause the parser to abort. (See the *mg_load* page for a list of errors.) If the *mg_uhand* pointer is assigned to another function, that function will receive any unknown entities and their arguments, and the parsing will abort if the new function returns a non-zero error value. This offers a convenient way to customize the language by adding non-standard entities.

DIAGNOSTICS

If an inconsistent set of entities has been set for support, the *mg_init* routine will print an informative message to standard error and abort the calling program with a call to *exit.* This is normally unacceptable behavior for a library routine, but since such an error indicates a fault with the calling program itself, recovery is impossible.

SEE ALSO

mg_load, mg_handle

NAME

mg_load, mg_clear, mg_file, mg_err - load MGF file, clear data structures

SYNOPSIS

#include "parser.h"

int mg_load(**char** *filename)

void mg_clear(**void**)

extern MG_FCTXT *mg_file

extern char *mg_err[MG_NERRS]

DESCRIPTION

The *mg_load* function loads the named file, or standard input if *filename* is the NULL pointer. Calls back to the appropriate MGF handler routines are made through the *mg_ehand* dispatch table.

The global *mg_file* variable points to the current file context structure, which may be useful for the interpretation of certain entities, such as ies, which must know the directory path of the enclosing file. This structure is of the defined type *MG_FCTXT*, given in "parser.h" as:

```
typedef struct mg_fctxt {
        char   fname[96];               /* file name */
        FILE  *fp;                      /* stream pointer */
        int    fid;                     /* unique file context id */
        char   inpline[4096];           /* input line */
        int    lineno;                  /* line number */
        struct mg_fctxt    *prev;       /* previous context */
} MG_FCTXT;
```

DIAGNOSTICS

If an error is encountered during parsing, *mg_load* will print an appropriate error message to the standard error stream and return one of the non-zero values from "parser.h" listed below:

```
#define MG_OK           0       /* normal return value */
#define MG_EUNK         1       /* unknown entity */
#define MG_EARGC        2       /* wrong number of arguments */
#define MG_ETYPE        3       /* argument type error */
#define MG_EILL         4       /* illegal argument value */
#define MG_EUNDEF       5       /* undefined reference */
#define MG_ENOFILE      6       /* cannot open input file */
#define MG_EINCL        7       /* error in included file */
#define MG_EMEM         8       /* out of memory */
#define MG_ESEEK        9       /* file seek error */
#define MG_EBADMAT      10      /* bad material specification */
#define MG_ELINE        11      /* input line too long */
#define MG_ECNTXT       12      /* unmatched context close */

#define MG_NERRS        13
```

If it is inappropriate to send output to standard error, the calling program should use the routines listed under *mg_open* for better control over the parsing process.

The *mg_err* array contains error messages corresponding to each of the values listed above in the native country's language.

SEE ALSO

mg_fgetpos, mg_handle, mg_init, mg_open

NAME

mg_open, mg_read, mg_parse, mg_close - MGF file loading subroutines

SYNOPSIS

#include "parser.h"

int mg_open(MG_FCTXT *fcp, **char** *filename)

int mg_read(**void**)

int mg_parse(**void**)

void mg_close(**void**)

DESCRIPTION

Most loaders and translators will call the *mg_load* routine to handle the above operations, but some programs or entity handlers require tighter control over the loading process.

The *mg_open* routine takes an uninitialized *MG_FCTXT* structure and a file name as its arguments. If *filename* is the NULL pointer, the standard input is "opened." The *fcp* structure will be set by *mg_open* prior to its return, and the global *mg_file* pointer will be assigned to point to it. This variable must not be destroyed until after the file is closed with a call to *mg_close.* (See the *mg_load* page for a definition of *mg_file* and the *MG_FCTXT* type.)

The *mg_read* function reads the next input line from the current file, returning the number of characters in the line, or zero if the end of file is reached or there is a file error. If the value returned equals MG_MAXLINE-1, then the input line was too long, and you should return an MG_ELINE error. The function keeps track of the line number in the current file context *mg_file,* which also contains the line that was read.

The *mg_parse* function breaks the current line in the *mg_file* structure into words and calls the appropriate handler routine, if any. Blank lines and unsupported entities cause a quick return.

The *mg_close* routine closes the current input file (unless it is the standard input) and returns to the previous file context (if any).

DIAGNOSTICS

The *mg_open* function returns *MG_OK* (0) normally, or *MG_ENOFILE* if the open fails for some reason.

The *mg_parse* function returns *MG_OK* if the current line was successfully interpreted, or one of the defined error values if there is a problem. (See the *mg_load* page for the defined error values.)

SEE ALSO

mg_fgetpos, mg_handle, mg_init, mg_load

NAME

mg_fgetpos, mg_fgoto - get current file position and seek to pointer

SYNOPSIS

#include "parser.h"

void mg_fgetpos(MG_FPOS *pos)

int mg_fgoto(MG_FPOS *pos)

DESCRIPTION

The *mg_fgetpos* gets the current MGF file position and loads it into the passed *MG_FPOS* structure, *pos*.

The *mg_fgoto* function seeks to the position *pos*, taken from a previous call to *mg_fgetpos*.

DIAGNOSTICS

If *mg_fgoto* is passed an illegal pointer or one that does not correspond to the current *mg_file* context, it will return the *MG_ESEEK* error value. Normally, it returns *MG_OK* (0).

SEE ALSO

mg_load, mg_open

NAME

mg_handle, mg_entity, mg_ename, mg_nqcdivs - entity assistance and control

SYNOPSIS

int mg_handle(**int** en, **int** ac, **char** *av)

int mg_entity(**char** *name)

extern char mg_ename[MG_NENTITIES][MG_MAXELEN]

extern int mg_nqcdivs

DESCRIPTION

The *mg_handle* routine may be used to pass entities back to the parser to be redirected through the *mg_ehand* dispatch table. This method is recommended rather than calling through *mg_ehand* directly, since the parser sometimes has its own support routines that it needs to call for specific entities. The first argument, *en*, is the corresponding entity number, or -1 if *mg_handle* should figure it out from the first *av* argument.

The *mg_entity* function gets an entity number from its name, using a hash table on the *mg_ename* list.

The *mg_ename* table contains the string names corresponding to each MGF entity in the designated order. (See the *mg_init* page for the list of MGF entities.)

The global integer variable *mg_nqcdivs* tells the parser how many subdivisions to use per quarter circle (90 degrees) when tesselating curved geometry. The default value is 5, and it may be reset at any time by the calling program.

DIAGNOSTICS

The *mg_handle* function returns *MG_OK* if the entity is handled correctly, or one of the predefined error values if there is a problem. (See the *mg_load* page for a list of error values.)

The *mg_entity* function returns -1 if the passed name does not appear in the *mg_ename* list.

SEE ALSO

mg_init, mg_load, mg_open

NAME

isint, isflt, isname - determine if string fits integer or real format, or is legal identifier

SYNOPSIS

int isint(**char** *str)

int isflt(**char** *str)

int isname(**char** *str)

DESCRIPTION

The *isint* function checks to see if the passed string *str* matches a decimal integer format (positive or negative), and returns 1 or 0 based on whether it does or does not.

The *isflt* function checks to see if the passed string *str* matches a floating point format (positive or negative with optional exponent), and returns 1 or 0 based on whether it does or does not.

The *isname* function checks to see if the passed string *str* is a legal identifier name. In MGF, a legal identifier must begin with a letter and contain only visible ASCII characters (those between decimal 33 and 127 inclusive). The one caveat to this is that names may begin with one or more underscores ('_'), but this is a trick employed by the parser to maintain a separate name space from the user, and is not legal usage otherwise.

Note that a string that matches an integer format is also a valid floating point value. Conversely, a string that is not a floating point number cannot be a valid integer.

These routines are useful for checking arguments passed to entity handlers that certain types in certain positions. If an invalid argument is passed, the handler should return an *MG_ETYPE* error.

SEE ALSO

mg_init, mg_load

NAME

c_hvertex, c_getvert, c_cvname, c_cvertex - vertex entity support

SYNOPSIS

#include "parser.h"

int c_hvertex(**int** argc, **char** **argv)

C_VERTEX *c_getvert(**char** *name)

extern char *c_vname

extern C_VERTEX *c_cvertex

DESCRIPTION

The *c_hvertex* function handles the MGF vertex entities, v, p and n. If either p or n is supported, then v must be also. The assignments are normally made to the *mg_ehand* array prior to parser initialization, like so:

```
mg_ehand[MG_E_VERTEX] = c_hvertex;      /* support "v" entity */
mg_ehand[MG_E_POINT] = c_hvertex;       /* support "p" entity */
mg_ehand[MG_E_NORMAL] = c_hvertex;      /* support "n" entity */
/* other entity handler assignments... */
mg_init();              /* initialize parser */
```

If vertex normals are not understood by any of the program-supported entities, then the *MG_E_NORMAL* entry may be left with its original NULL assignment.

The *c_getvert* call takes the name of a defined vertex and returns a pointer to its *C_VERTEX* structure, defined in "parser.h" as:

```
typedef FLOAT  FVECT[3];    /* a 3-d real vector */

typedef struct {
        int     clock;              /* incremented each change -- resettable */
        char   *client_data;        /* pointer to private client data */
        FVECT       p, n;           /* point and normal */
} C_VERTEX;             /* vertex context */
```

The *clock* member will be incremented each time the value gets changed by a p or n entity, and may be reset by the controlling program if desired. This is a convenient way to keep track of whether or not a vertex has changed since its last use. To link identical vertices, one must also check that the current transform has not changed, which is uniquely identified by the global *xf_context->xid* variable, but only if one is using the parser library's transform handler. (See the *xf_handler* page.) The *client_data* pointer may be used to index private application data for vertex linking, etc. This pointer is initialized to NULL when the context is created, and otherwise ignored by the parser library.

It is possible but not recommended to alter the shared contents of the vertex structure returned by *c_getvert.* Normally it is read during the interpretation of entities using named vertices.

The name of the current vertex is given by the global *c_cvname* variable, which is set to NULL if the unnamed vertex is current. The current vertex value is pointed to by the global variable *c_cvertex*, which should never be NULL.

DIAGNOSTICS

The *c_hvertex* function returns *MG_OK* (0) if the vertex is handled correctly, or one of the predefined error values if there is a problem. (See the *mg_load* page for a list of errors.)

The *c_getvert* function returns NULL if the specified vertex name is undefined, at which point the calling function should return an *MG_EUNDEF* error.

SEE ALSO

c_hcolor, c_hmaterial, mg_init, mg_load, xf_handler

NAME

c_hcolor, c_getcolor, c_ccname, c_ccolor, c_ccvt, c_isgrey - color entity support

SYNOPSIS

#include "parser.h"

int c_hcolor(**int** argc, **char** **argv)

C_COLOR *c_getcolor(**char** *name)

extern char *c_ccname

extern C_COLOR *c_ccolor

void c_ccvt(C_COLOR *cvp, **int** cflags)

int c_isgrey(C_COLOR *cvp)

DESCRIPTION

The *c_hcolor* function supports the MGF entities, c, cxy, cspec, cct and cmix. It is an error to support any of the color field entities without supporting the c entity itself. The assignments are normally made to the *mg_ehand* array prior to parser initialization, like so:

```
mg_ehand[MG_E_COLOR] = c_hcolor;    /* support "c" entity */
mg_ehand[MG_E_CXY] = c_hcolor;      /* support "cxy" entity */
mg_ehand[MG_E_CSPEC] = c_hcolor;    /* support "cspec" entity */
mg_ehand[MG_E_CCT] = c_hcolor;      /* support "cct" entity */
mg_ehand[MG_E_CMIX] = c_hcolor;     /* support "cmix" entity */
/* other entity handler assignments... */
mg_init();              /* initialize parser */
```

If the loader/translator has no use for spectral data, the entries for cspec and cct may be left with their original NULL assignments and these entities will be re-expressed appropriately as tristimulus values.

The *c_getcolor* function takes the name of a defined color and returns a pointer to its *C_COLOR* structure, defined in "parser.h" as:

```
#define C_CMINWL    380                         /* minimum wavelength */
#define C_CMAXWL    780                         /* maximum wavelength */
#define C_CNSS      41                          /* number of spectral samples */
#define C_CWLI      ((C_CMAXWL-C_CMINWL)/(C_CNSS-1))
#define C_CMAXV         10000                   /* nominal maximum sample value */
#define C_CLPWM         (683./C_CMAXV)          /* peak lumens/watt multiplier */

typedef struct {
        int     clock;          /* incremented each change */
        char    *client_data;   /* pointer to private client data */
        short   flags;          /* what's been set */
        short   ssamp[C_CNSS];  /* spectral samples, min wl to max */
        long    ssum;           /* straight sum of spectral values */
        float   cx, cy;         /* xy chromaticity value */
        float   eff;            /* efficacy (lumens/watt) */
} C_COLOR;              /* color context */
```

The *clock* member will be incremented each time the value gets changed by a color field entity, and may be reset by the calling program if desired. This is a convenient way to keep track of whether or not a color has changed since its last use. The *client_data* pointer may be used to

index private application data. This pointer is initialized to NULL when the context is created, and otherwise ignored by the parser library. The *flags* member indicates which color representations have been assigned, and is an inclusive OR of one or more of the following:

```
#define C_CSSPEC    01      /* flag if spectrum is set */
#define C_CDSPEC    02      /* flag if defined w/ spectrum */
#define C_CSXY      04      /* flag if xy is set */
#define C_CDXY      010     /* flag if defined w/ xy */
#define C_CSEFF     020     /* flag if efficacy set */
```

It is possible but not recommended to alter the contents of the color structure returned by *c_getcolor.* Normally, this routine is never called directly, since there are no entities that access colors by name other than c.

The global variable *c_ccname* points to the name of the current color, or NULL if it is unnamed. The variable *c_ccolor* points to the current color value, which should never be NULL.

The *c_ccvt* routine takes a *C_COLOR* structure and a set of desired flag settings and computes the missing color representation(s).

The *c_isgrey* function returns 1 if the passed color is very close to neutral grey, or 0 otherwise.

DIAGNOSTICS

The *c_hcolor* function returns *MG_OK* (0) if the color is handled correctly, or one of the predefined error values if there is a problem. (See the *mg_load* page for a list of errors.)

The *c_getcolor* function returns NULL if the specified color name is undefined, at which point the calling function should return an *MG_EUNDEF* error.

SEE ALSO

c_hmaterial, c_hvertex, mg_init, mg_load

NAME

c_hmaterial, c_getmaterial, c_cmname, c_cmaterial - material entity support

SYNOPSIS

#include "parser.h"

int c_hmaterial(**int** argc, **char** **argv)

C_MATERIAL *c_getmaterial(**char** *name)

extern char *c_cmname

extern C_MATERIAL *c_cmaterial

DESCRIPTION

The *c_hmaterial* function supports the MGF entities, m, ed, ir, rd, rs, sides, td, and ts. It is an error to support any of the material field entities without supporting the m entity itself. The assignments are normally made to the *mg_ehand* array prior to parser initialization, like so:

```
mg_ehand[MG_E_MATERIAL] = c_hmaterial;   /* support "m" entity */
mg_ehand[MG_E_ED] = c_hmaterial;         /* support "ed" entity */
mg_ehand[MG_E_IR] = c_hmaterial;         /* support "ir" entity */
mg_ehand[MG_E_RD] = c_hmaterial;         /* support "rd" entity */
mg_ehand[MG_E_RS] = c_hmaterial;         /* support "rs" entity */
mg_ehand[MG_E_SIDES] = c_hmaterial;      /* support "sides" entity */
mg_ehand[MG_E_TD] = c_hmaterial;         /* support "td" entity */
mg_ehand[MG_E_TS] = c_hmaterial;         /* support "ts" entity */
/* other entity handler assignments... */
mg_init();                  /* initialize parser */
```

Any of the above entities besides m may be unsupported, but the parser will not attempt to include their effect into other members, e.g. an unsupported rs component will not be added back into the rd member. It is therefore safer to support all of the relevant material entities and make final approximations from the complete *C_MATERIAL* structure.

The *c_getmaterial* function takes the name of a defined material and returns a pointer to its *C_MATERIAL* structure, defined in "parser.h" as:

```
#define C_1SIDEDTHICK     0.005         /* assumed thickness of 1-sided mat. */

typedef struct {
        int     clock;          /* incremented each change -- resettable */
        char    *client_data;           /* pointer to private client data */
        int     sided;          /* 1 if surface is 1-sided, 0 for 2-sided */
        float   nr, ni;         /* index of refraction, real and imaginary */
        float   rd;             /* diffuse reflectance */
        C_COLOR  rd_c;  /* diffuse reflectance color */
        float   td;             /* diffuse transmittance */
        C_COLOR  td_c;  /* diffuse transmittance color */
        float   ed;             /* diffuse emittance */
        C_COLOR  ed_c; /* diffuse emittance color */
        float   rs;             /* specular reflectance */
        C_COLOR  rs_c;  /* specular reflectance color */
        float   rs_a;           /* specular reflectance roughness */
        float   ts;             /* specular transmittance */
        C_COLOR  ts_c;  /* specular transmittance color */
        float   ts_a;           /* specular transmittance roughness */
} C_MATERIAL; /* material context */
```

The *clock* member will be incremented each time the value gets changed by a material field entity, and may be reset by the calling program if desired. This is a convenient way to keep track of whether or not a material has changed since its last use. The *client_data* pointer may be used to index private application data. This pointer is initialized to NULL when the context is created, and otherwise ignored by the parser library.

All reflectance and transmittance values correspond to normal incidence, and may vary as a function of angle depending on the index of refraction. A solid object is normally represented with a one-sided material. A two-sided material is most appropriate for thin surfaces, though it may be used also when the surface normal orientations in a model are unreliable.

If a transparent or translucent surface is one-sided, then the absorption will change as a function of distance through the material, and a single value for diffuse or specular transmittance is ambiguous. We therefore define a standard thickness, *C_1SIDEDTHICK*, which is the object thickness to which the given values correspond, so that one may compute the isotropic absorptance of the material.

It is possible but not recommended to alter the contents of the material structure returned by *c_getmaterial*. Normally, this routine is never called directly, since there are no entities that access materials by name other than m.

The global variable *c_cmname* points to the name of the current material, or NULL if it is unnamed. The variable *c_cmaterial* points to the current material value, which should never be NULL.

DIAGNOSTICS

The *c_hmaterial* function returns *MG_OK* (0) if the color is handled correctly, or one of the predefined error values if there is a problem. (See the *mg_load* page for a list of errors.)

The *c_getmaterial* function returns NULL if the specified material name is undefined, at which point the calling function should return an *MG_EUNDEF* error.

SEE ALSO

c_hcolor, c_hvertex, mg_init, mg_load

NAME

obj_handler, obj_clear, obj_nnames, obj_name - object name support

SYNOPSIS

int obj_handler(**int** argc, **char** **argv)

void obj_clear(**void**)

extern int obj_nnames

extern char **obj_name

DESCRIPTION

The *obj_handler* routine should be assigned to the *MG_E_OBJECT* entry of the parser's *mg_ehand* array prior to calling *mg_load* if the loader/translator wishes to support hierarchical object names.

The *obj_clear* function may be used to clear the object name stack and free any associated memory, but this is usually not necessary since o begin and end entities are normally balanced in the input.

The global *obj_nnames* variable indicates the number of names currently in the object stack, and the *obj_name* list contains the name strings in the same order as they were encountered on the input. (I.e. the most recently pushed name is last.)

DIAGNOSTICS

The *obj_handler* function returns *MG_OK* (0) if the color is handled correctly, or one of the predefined error values if there is a problem. (See the *mg_load* page for a list of errors.)

SEE ALSO

mg_init, mg_load, xf_handler

NAME

xf_handler, xf_clear, xf_context, xf_argend - transformation support

SYNOPSIS

int xf_handler(**int** argc, **char** **argv)

void xf_clear(**void**)

extern XF_SPEC *xf_context

extern char **xf_argend

DESCRIPTION

The *xf_handler* routine should be assigned to the *MG_E_XF* entry of the parser's *mg_ehand* array prior to calling *mg_load* if the loader/translator wishes to support hierarchical transformations. (Note that all MGF geometric entities require this support.)

The *xf_clear* function may be used to clear the transform stack and free any associated memory, but this is usually not necessary since xf begin and end entities are normally balanced in the input.

The global *xf_context* variable points to the current transformation context, which is of the type *XF_SPEC*, described in "parser.h":

```
typedef struct xf_spec {
        long   xid;                     /* unique transform id */
        short  xac;                     /* context argument count */
        short  rev;                     /* boolean true if vertices reversed */
        XF     xf;                      /* cumulative transformation */
        struct xf_array     *xarr;      /* transformation array pointer */
        struct xf_spec      *prev;      /* previous transformation context */
} XF_SPEC;              /* followed by argument buffer */
```

The *xid* member is a identifier associated with this transformation, which should be the same for identical transformations, as an aid to vertex sharing. (See also the *c_hvertex* page.) The *xac* member indicates the total number of transform arguments, and is used to indicate the position of the first argument relative to the last one pointed to by the global *xf_argend* variable.

The first transform argument starts at *xf_argv*, which is a macro defined in "parser.h" as:

```
#define xf_argv         (xf_argend - xf_context->xac)
```

Note that accessing this macro will result in a segmentation violation if the current context is NULL, so one should first test the second macro *xf_argc* against zero. This macro is defined as:

```
#define xf_argc         (xf_context==NULL ? 0 : xf_context->xac)
```

Normally, neither of these macros will be used, since there are routines for transforming points, vectors and scalars directly based on the current transformation context. (See the *xf_xfmpoint* page for details.)

The *rev* member of the *XF_SPEC* structure indicates whether or not this transform reverses the order of polygon vertices. This member will be 1 if the transformation mirrors about an odd number of coordinate axes, thus inverting faces. The usual thing to do in this circumstance is to interpret the vertex arguments in the reverse order, so as to bring the face back to its original orientation in the new position.

The *xf* member contains the transformation scalefactor (in xf.sca) and 4x4 homogeneous matrix (in xf.xfm), but these will usually not be accessed directly. Likewise, the *xarr* and *prev* members point to data that should not be needed by the calling program.

DIAGNOSTICS

The *xf_handler* function returns *MG_OK* (0) if the color is handled correctly, or one of the predefined error values if there is a problem. (See the *mg_load* page for a list of errors.)

SEE ALSO

mg_init, mg_load, obj_handler, xf_xfmpoint

NAME

xf_xfmpoint, xf_xfmvect, xf_rotvect, xf_scale - apply current transformation

SYNOPSIS

void xf_xfmpoint(FVECT pnew, FVECT pold)

void xf_xfmvect(FVECT vnew, FVECT vold)

void xf_rotvect(FVECT nnew, FVECT nold)

double xf_scale(**double** sold)

DESCRIPTION

The *xf_xfmpoint* routine applies the current transformation defined by *xf_context* to the point *pold*, scaling, rotating and moving it to its proper location, which is put in *pnew*. (As for *xf_xfmvect* and *xf_rotvect*, the two arguments may point to the same vector.)

The *xf_xfmvect* routine applies the current transformation to the vector *vold*, scaling and rotating it to its proper location, which is put in *vnew*. The only difference between *xf_xfmpoint* and *xf_xfmvect* is that in the latter, the final translation is not applied.

The *xf_rotvect* routine rotates the vector *nold* using the current transformation, and stores the result in *nnew*. No translation or scaling is applied, which is the appropriate action for surface normal vectors for example.

The *xf_scale* function takes a scalar argument *sold* and applies the current scale factor, returning the result.

SEE ALSO

xf_handler

6. Application Notes

6.1. Relation to Standard Practices in Computer Graphics

For those coming from a computer graphics background, some of the choices in the material model may seem strange or even capricious. Why not simply stick with RGB colors and a Phong specular component like everyone else? What is the point in choosing the number of sides to a material?

In the real world, a surface can have only one side, defining the interface between one volume and another. Many object-space rendering packages (e.g. z-buffer algorithms) take advantage of this fact by culling back-facing polygons and thus saving as much as 50% of the preprocessing time. However, many models rely on an approximation whereby a single surface is used to represent a very thin volume, such as a pane of glass, and this also can provide significant calculational savings in an image-space algorithm (such as ray-tracing). Also, many models are created in such a way that the front vs. back information is lost or confused, so that the back side of one or more surfaces may have to serve as the front side during rendering. (AutoCAD is one easily identified culprit in this department.) Since both types of surface models are useful and any rendering algorithm may ultimately be applied, MGF provides a way to specify sidedness rather than picking one interpretation or the other.

The problem with RGB is that there is no accepted standard, and even if we were to set one it would either be impossible to realize (i.e. impossible to create phosphors with the chosen colors) or it would have a gamut that excludes many saturated colors. The CIE color system was very carefully conceived and developed, and is the standard to which all photometric measurements adhere. It is therefore the logical choice in any standard format, though it has been too often ignored by the computer graphics community.

Regarding Phong shading, this was never a physical model and making it behave basic laws of reciprocity and energy balance is difficult. More to the point, specular power has almost nothing to do with surface microstructure, and is difficult to set properly even if every physical characteristic of a material has been carefully measured. This is the ultimate indictment of any physical model -- that it is incapable of reproducing any measurement whatsoever.

Admittedly, the compliment of diffuse and specular component plus surface roughness and index of refraction used in MGF is less than a perfect model, but it is serviceable for most materials and relatively simple to incorporate into a rendering algorithm. In the long term, MGF shall probably include full spectral scattering functions, though the sheer quantity of data involved makes this burdensome from both the measurement side and the simulation side.

6.1.1. Converting between Phong Specular Power and Gaussian Roughness

So-called specular reflection and transmission are modeled using a Gaussian distribution of surface facets. The roughness parameters to the rs and ts entities specify the root-mean-squared (RMS) surface facet slope, which varies from 0 for a perfectly smooth surface to around .2 for a fairly rough one. The effect this will have on the reflected component distribution is well-defined, but predicting the behavior of the transmitted component requires further assumptions. We assume that the surface scatters light passing through it just as much as it scatters reflected light. This assumption is approximately correct for a two-sided transparent material with an index of refraction of 1.5 (like glass) and both sides having the given RMS facet slope.

Oftentimes, one is translating from a Phong exponent on the cosine of the half-vector-to-normal angle to the more physical but less familiar Gaussian model of MGF. The hardest part is translating the specular power to a roughness value. For this, we recommend the following approximation:

```
roughness = sqrt(2/specular_power)
```

It is not a perfect correlation, but it is about as close as one can get.

6.1.2. Converting between RGB and CIE Colors

Unlike most graphics languages, MGF does not use an RGB color model, simply because there is no recognized definition for this model. It is based on computer monitor phosphors, which vary from one CRT to the next. (There is an RGB standard defined in the TV industry, but this has a rather poor correlation to most computer monitors and it is impossible to express many real-world colors within its limited gamut.)

MGF uses two alternative, well-defined standards, spectral power distributions and the 1931 CIE 2 degree standard observer. With the CIE standard, any viewable color may be exactly represented as an (x,y) chromaticity value. Unfortunately, the interaction between colors (i.e. colored light sources and interreflections) cannot be specified exactly with any finite coordinate set, including CIE chromaticities. So, MGF offers the ability to give reflectance, transmittance or emittance as a function of wavelength over the visible spectrum. This function is still discretized, but at a user-selectable resolution. Furthermore, spectral colors may be mixed, providing (nearly) arbitrary basis functions, which can produce more accurate results in some cases and are merely a convenience for translation in others.

Conversion back and forth between CIE chromaticity coordinates and spectral samples is provided within the MGF parser. Unfortunately, conversion to and from RGB values depends on a particular RGB definition, and as we have said, there is no recognized standard. We therefore recommend that you decide yourself what chromaticity values to use for each RGB primary, and adopt the following code to convert between CIE and RGB coordinates.

```
#ifdef  NTSC
#define  CIE_x_r          0.670        /* standard NTSC primaries */
#define  CIE_y_r          0.330
#define  CIE_x_g          0.210
#define  CIE_y_g          0.710
#define  CIE_x_b          0.140
#define  CIE_y_b          0.080
#define  CIE_x_w          0.3333             /* monitor white point */
#define  CIE_y_w          0.3333
#else
#define  CIE_x_r          0.640        /* nominal CRT primaries */
#define  CIE_y_r          0.330
#define  CIE_x_g          0.290
#define  CIE_y_g          0.600
#define  CIE_x_b          0.150
#define  CIE_y_b          0.060
#define  CIE_x_w          0.3333             /* monitor white point */
#define  CIE_y_w          0.3333
#endif

#define CIE_D             (       CIE_x_r*(CIE_y_g - CIE_y_b) + \
                          CIE_x_g*(CIE_y_b - CIE_y_r) + \
                          CIE_x_b*(CIE_y_r - CIE_y_g)        )
#define CIE_C_rD          ( (1./CIE_y_w) * \
                          ( CIE_x_w*(CIE_y_g - CIE_y_b) - \
                            CIE_y_w*(CIE_x_g - CIE_x_b) + \
                            CIE_x_g*CIE_y_b - CIE_x_b*CIE_y_g   ) )
#define CIE_C_gD          ( (1./CIE_y_w) * \
                          ( CIE_x_w*(CIE_y_b - CIE_y_r) - \
                            CIE_y_w*(CIE_x_b - CIE_x_r) - \
```

```
                        CIE_x_r*CIE_y_b + CIE_x_b*CIE_y_r   ) )
#define CIE_C_bD        ( (1./CIE_y_w) * \
                        ( CIE_x_w*(CIE_y_r - CIE_y_g) - \
                          CIE_y_w*(CIE_x_r - CIE_x_g) + \
                          CIE_x_r*CIE_y_g - CIE_x_g*CIE_y_r    ) )

#define CIE_rf          (CIE_y_r*CIE_C_rD/CIE_D)
#define CIE_gf          (CIE_y_g*CIE_C_gD/CIE_D)
#define CIE_bf          (CIE_y_b*CIE_C_bD/CIE_D)

float  xyz2rgbmat[3][3] = {     /* XYZ to RGB */
        {(CIE_y_g - CIE_y_b - CIE_x_b*CIE_y_g + CIE_y_b*CIE_x_g)/CIE_C_rD,
         (CIE_x_b - CIE_x_g - CIE_x_b*CIE_y_g + CIE_x_g*CIE_y_b)/CIE_C_rD,
         (CIE_x_g*CIE_y_b - CIE_x_b*CIE_y_g)/CIE_C_rD},
        {(CIE_y_b - CIE_y_r - CIE_y_b*CIE_x_r + CIE_y_r*CIE_x_b)/CIE_C_gD,
         (CIE_x_r - CIE_x_b - CIE_x_r*CIE_y_b + CIE_x_b*CIE_y_r)/CIE_C_gD,
         (CIE_x_b*CIE_y_r - CIE_x_r*CIE_y_b)/CIE_C_gD},
        {(CIE_y_r - CIE_y_g - CIE_y_r*CIE_x_g + CIE_y_g*CIE_x_r)/CIE_C_bD,
         (CIE_x_g - CIE_x_r - CIE_x_g*CIE_y_r + CIE_x_r*CIE_y_g)/CIE_C_bD,
         (CIE_x_r*CIE_y_g - CIE_x_g*CIE_y_r)/CIE_C_bD}
};

float  rgb2xyzmat[3][3] = {     /* RGB to XYZ */
        {CIE_x_r*CIE_C_rD/CIE_D,CIE_x_g*CIE_C_gD/CIE_D,CIE_x_b*CIE_C_bD/CIE_D},
        {CIE_y_r*CIE_C_rD/CIE_D,CIE_y_g*CIE_C_gD/CIE_D,CIE_y_b*CIE_C_bD/CIE_D},
        {(1.-CIE_x_r-CIE_y_r)*CIE_C_rD/CIE_D,
         (1.-CIE_x_g-CIE_y_g)*CIE_C_gD/CIE_D,
         (1.-CIE_x_b-CIE_y_b)*CIE_C_bD/CIE_D}
};


cie_rgb(rgbcolor, ciecolor)             /* convert CIE to RGB */
register float  *rgbcolor, *ciecolor;
{
        register int  i;

        for (i = 0; i < 3; i++) {
                rgbcolor[i] =       xyz2rgbmat[i][0]*ciecolor[0] +
                                    xyz2rgbmat[i][1]*ciecolor[1] +
                                    xyz2rgbmat[i][2]*ciecolor[2] ;
                if (rgbcolor[i] < 0.0)          /* watch for negative values */
                        rgbcolor[i] = 0.0;
        }
}


rgb_cie(ciecolor, rgbcolor)             /* convert RGB to CIE */
register float  *ciecolor, *rgbcolor;
{
        register int  i;

        for (i = 0; i < 3; i++)
                ciecolor[i] =rgb2xyzmat[i][0]*rgbcolor[0] +
                                    rgb2xyzmat[i][1]*rgbcolor[1] +
```

```
                rgb2xyzmat[i][2]*rgbcolor[2] ;
}
```

An alternative to adopting the above code is to use the MGF "cmix" entity to convert from RGB directly by naming the three primaries in terms of their chromaticities, e.g:

```
c R =
        cxy 0.640 0.330
c G =
        cxy 0.290 0.600
c B =
        cxy 0.150 0.060
```

Then, converting from RGB to MGF colors is as simple as multiplying each component by its relative luminance in a cmix statement, for instance:

```
c white =
        cmix 0.265 R 0.670 G 0.065 B
```

For the chosen RGB standard, the above specification would result a pure white. The reason the coefficients are not all 1 as you might expect is that cmix uses relative luminance as the standard for its weights. Since blue is less luminous for the same energy than red, which is in turn less luminous than green, the weights cannot be the same to achieve an even spectral balance. Unfortunately, computing these relative weights is not straightforward, though it is given in the above macros as CIE_rf, CIE_gf and CIE_bf. (The common factors in these macros may of course be removed since cmix weights are all relative.) Alternatively, one could measure the actual full scale luminance of the phosphors with a luminance probe and get the same relative values.

6.2. Relation to IESNA LM-63 and Luminaire Catalogs

Recently, the Illuminating Engineering Society of North America (IESNA) adopted MGF as the official standard for representing luminaire geometry and materials. The way this works in an IES luminaire data file is through the addition of a keyword called LUMINOUSGEOMETRY, which is given on a line in the header portion of a file (before the TILT specification) like so:

[LUMINOUSGEOMETRY] *mgf_file*

The given MGF file must exist relative to the directory containing the IES file (i.e. the same stipulations and restrictions on pathnames apply as for the MGF i entity). Furthermore, the position of the MGF geometry must be such that the gross geometric specification of emitting surfaces in the IES file completely blocks or encloses the luminous portions of the MGF description. Specifically, any ray traced towards the MGF geometry must strike the IES gross geometry before it strikes any luminous surface in the MGF description. This provides a convenient way of preventing overcounting in the illumination calculation, while still allowing for accurate fixture appearance.

To give two examples, let us consider first a recessed can, followed by a hanging direct/indirect fluorescent fixture.

The most appropriate IES geometric specification for the emitting area of a can light would be a circular disk. Since the IES gross geometry gives only the diameter of the disk, the actual 3-dimensional placement is implicitly defined as having a center at the origin, with the radiating disk facing in the negative Z direction (nadir, downwards). The MGF geometry would then be placed such that any luminous portion was above this disk, and no portion of it would obstruct the IES geometry. The most sensible position therefore has the IES disk flush with the MGF can opening, as shown in Figure 3.

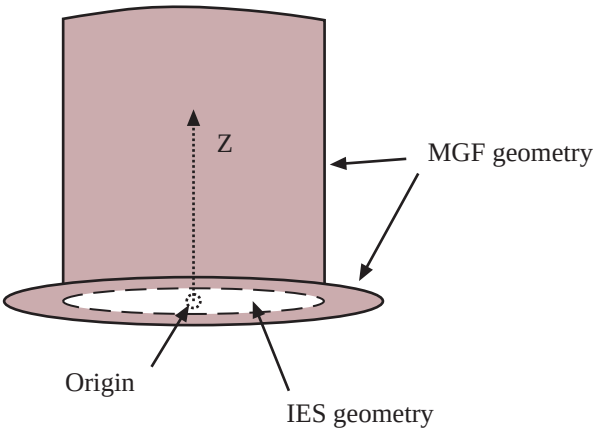


Figure 3. Geometric representation of can downlight fixture, and placement of IES simple geometry.

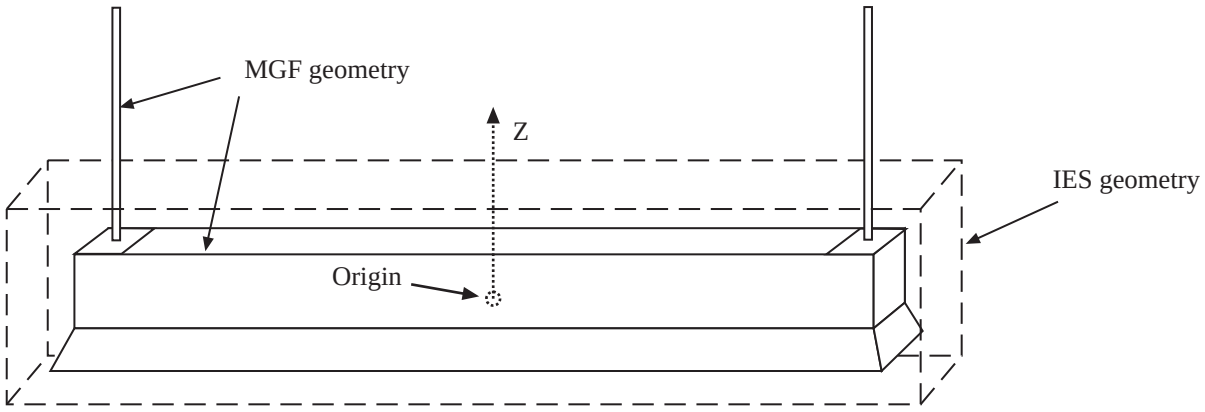


Figure 4. A hanging direct/indirect fixture and the surrounding IES simple geometry.

In the case of a direct/indirect fluorescent fixture, light will exit both the top and the bottom sides, and the IES geometry must enclose the radiating portion of the fixture entirely. It is acceptable to have additional MGF geometry above the fixture so long as it does not radiate, which is what we must do if we wish to include the support rods, as shown in Figure 4.

Note that the origin is always in the exact center of the IES geometry.

Not all fixtures will fit the simple IES geometry specification so nicely. For odd-shaped fixtures, it may be necessary to use an IES geometry that does not match the radiating area terribly well in order that it completely block or enclose the required MGF specification.

The unit of length in the MGF file is always meters, regardless of the units specified in the enclosing IES file. However, any and all multipliers applied to the candlepower data in the IES file will also be applied to the emittance of surfaces in the MGF specification, so that one MGF file may serve similar luminaires that differ in their total output.

7. Credits

The MGF language grew out of a joint investigation into physical representations for rendering undertaken by the author (Greg Ward of LBL) and Holly Rushmeier of the National Institute of Standards and Technology. After deciding that a complete and robust specification was an extreme challenge, we shelved the project for another time. A few months later, the author spoke with Ian Ashdown and Robert Shakespeare, who are both members of the IES Computing Committee, about the need for extending the existing data standard to include luminaire geometry and near-field photometry. We then moved forward as a team towards a somewhat less ambitious approach to physical materials and geometry that had the advantage of simplicity and the possibility of support with a standard parser library. The author went to work over the next two months on the detailed design of the language and an ANSI-C parser, with regular feedback from the other three team members. Several months and several versions later, we arrived at release 1.0, which is the occasion of this document's creation.

Funding for this work... would be nice.

Appendix C:

Picture Perfect *RGB* Rendering Using Spectral Prefiltering and Sharp Color Primaries

Reprinted from 2002 Eurographics Workshop on Rendering

Citation:

Ward, Greg, Elena Eydelberg-Vileshin, "Picture Perfect RGB Rendering Using Spectral Prefiltering and Sharp Color Primaries." *Thirteenth Eurographics Workshop on Rendering* (2002), P. Debevec and S. Gibson (Editors), June 2002.

Thirteenth Eurographics Workshop on Rendering (2002)
P. Debevec and S. Gibson (Editors)

Picture Perfect *RGB* Rendering Using Spectral Prefiltering and Sharp Color Primaries

Greg Ward[†]

Elena Eydelberg-Vileshin[‡]

Abstract

Accurate color rendering requires the consideration of many samples over the visible spectrum, and advanced rendering tools developed by the research community offer multispectral sampling towards this goal. However, for practical reasons including efficiency, white balance, and data demands, most commercial rendering packages still employ a naive RGB *model in their lighting calculations. This results in colors that are often qualitatively different from the correct ones. In this paper, we demonstrate two independent and complementary techniques for improving* RGB *rendering accuracy without impacting calculation time: spectral prefiltering and color space selection. Spectral prefiltering is an obvious but overlooked method of preparing input colors for a conventional* RGB *rendering calculation, which achieves exact results for the direct component, and very accurate results for the interreflected component when compared with full-spectral rendering. In an empirical error analysis of our method, we show how the choice of rendering color space also affects final image accuracy, independent of prefiltering. Specifically, we demonstrate the merits of a particular transform that has emerged from the color research community as the best performer in computing white point adaptation under changing illuminants: the Sharp* RGB *space.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture

1. Introduction

It is well-known that the human eye perceives color in a three-dimensional space, owing to the presence of three types of color receptors. Early psychophysical research demonstrated conclusively that three component values are sufficient to represent any perceived color, and these values may be quantified using the CIE *XYZ* tristimulus space[20]. However, because the spectrum of light is continuous, the interaction between illumination and materials cannot be accurately simulated with only three samples. In fact, no finite number of fixed spectral samples is guaranteed to be sufficient — one can easily find pathological cases, for example, a pure spectral source mixed with a narrow band absorber, that require either component analysis or a ludicrous number of fixed samples to resolve. If the rendered spectrum is inaccurate, reducing it to a tristimulus value will usually not hide the problem.

Besides the open question of how many spectral samples to use, there are other practical barriers to applying full spectral rendering in commercial software. First, there is the general dearth of spectral reflectance data on which to base a spectral simulation. This is consistent with the lack of any kind of reflectance data for rendering. We are grateful to the researchers who are hard at work making spectral data available[3, 19], but the ultimate solution may be to put the necessary measurement tools in the hands of people who care about accurate color rendering. Hand-held spectrophotometers exist and may be purchased for the cost of a good laser printer, but few people apply them in a rendering context, and to our knowledge, no commercial rendering application takes spectrophotometer data as input.

The second practical barrier to spectral rendering is white balance. This is actually a minor issue once you know how to address it, but the first time you render with the correct

[†] Exponent – Failure Analysis Associates, Menlo Park, California
[‡] Department of Computer Science, Stanford University, Palo Alto, California

source and reflectance spectra, you are likely to be disappointed by the strong color cast in your output. This is due to the change in illuminant from the simulated scene to the viewing condition, and there is a well-known method to correct for this, which we will cover in Section 2.

The third practical barrier to the widespread acceptance of spectral rendering is what we call the "data mixing problem." What if the user goes to the trouble of acquiring spectral reflectances for a set of surfaces, but they also want to include materials that are characterized in terms of *RGB* color, or light sources that are specified to a different spectral resolution? One may interpolate and extrapolate to some extent, but in the end, it may be necessary to either synthesize a spectrum from *RGB* triples a la Smits' method[14], or reduce all the spectral data to *RGB* values and fall back on three component rendering again.

The fourth practical barrier to full spectral rendering is cost. In many renderings, shading calculations dominate the computation, even in *RGB*. If all of these calculations must be carried out at the maximum spectral resolution of the input, the added cost may not be worth the added benefit.

Many researchers in computer graphics and color science have addressed the problem of efficient spectral sampling[8, 7]. Meyer suggested a point-sampling method based on Gaussian quadrature and a preferred color space, which requires only 4 spectral samples and is thus very efficient[11]. Like other point sampling techniques, however, Meyer's method is prone to problems when the source spectrum has significant spikes in it, as in the case of common fluorescent lighting. A more sophisticated approach employing orthonormal basis functions was presented by Peercy, who uses characteristic vector analysis on combinations of light source and reflectance spectra to find an optimal, orthonormal basis set[13]. Peercy's method has the advantage of handling spiked and smooth spectra with equal efficiency, and he demonstrated accurate results with as few as three orthonormal bases. The additional cost is comparable to spectral sampling, replacing N multiplies in an N-sample spectral model with $M \times M$ multiplies in an M-basis vector model. Examples in his paper showed the method significantly out-performing uniform spectral sampling for the same number of operations. The cost for a 3-basis simulation, the minimum for acceptable accuracy in Peercy's technique, is roughly three times that of a standard *RGB* shading calculation.

In this paper, we present a method that has the same overall accuracy as Peercy's technique, but without the computational overhead. In fact, no modification at all is required to a conventional *RGB* rendering engine, which multiplies and sums its three color components separately throughout the calculation. Our method is not subject to point sampling problems in spiked source or absorption spectra, and the use of an *RGB* rendering space all but eliminates the data mixing problem mentioned earlier. White adaptation is also accounted for by our technique, since we ask the user to identify a dominant source spectrum for their scene. This avoids the dreaded color cast in the final image.

We start with a few simple observations:

1. The direct lighting component is the first order in any rendering calculation, and its accuracy determines the accuracy of what follows.
2. Most scenes contain a single dominant illuminant; there may be many light sources, but they tend to all have the same spectral power distribution, and spectrally differentiated sources make a negligible contribution to illumination.
3. Exceptional scenes, where spectrally distinct sources make roughly equal contributions, cannot be "white balanced," and will look wrong no matter how accurately the colors are simulated. We can be satisfied if our color accuracy is no worse on average than standard methods in the mixed illuminant case.

The spectral prefiltering method we propose is quite simple. We apply a standard CIE formula to compute the reflected *XYZ* color of each surface under the dominant illuminant, then transform this to a white-balanced *RGB* color space for rendering and display. The dominant sources are then replaced by white sources of equal intensity, and other source colors are modified to account for this adaptation. By construction, the renderer gets the exact answer for the dominant direct component, and a reasonably close approximation for other sources and higher order components.

The accuracy of indirect contributions and spectrally distinct illumination will depend on the sources, materials, and geometry in the scene, as well as the color space chosen for rendering. We show by empirical example how a sharpened *RGB* color space seems to perform particularly well in simulation, and offer some speculation as to why this might be the case.

Section 2 details the equations and steps needed for spectral filtering and white point adjustment. Section 3 shows an example scene with three combinations of two spectrally distinct light sources, and we compare the color accuracy of naive *RGB* rendering to our prefiltering approach, each measured against a full spectral reference solution. We also look at three different color spaces for rendering: CIE *XYZ*, linear *sRGB*, and the Sharp *RGB* space. Finally, we conclude with a summary discussion and suggestions for future work.

2. Method

The spectral prefiltering method we propose is a straightforward transformation from measured source and reflectance spectra to three separate color channels for rendering. These input colors are then used in a conventional rendering process, followed by a final transformation into the display *RGB* space. Chromatic adaptation (i.e., white balancing) may take place either before or after rendering, as a matter of convenience and efficiency.

2.1. Color Transformation

Given a source $I(\lambda)$ and a material $\rho_m(\lambda)$ with arbitrary spectral distributions, the CIE describes a standard method for deriving a tristimulus value that quantifies the average person's color response. The *XYZ* tristimulus color space is computed from the CIE "standard observer" response functions, $\bar{x}$, $\bar{y}$, and $\bar{z}$, which are integrated with an arbitrary source illuminant spectrum and surface reflectance spectrum as shown in Eq. (1), below:

$$\begin{aligned} X_m &= \int I(\lambda)\,\rho_m(\lambda)\,\bar{x}(\lambda)\,d\lambda \\ Y_m &= \int I(\lambda)\,\rho_m(\lambda)\,\bar{y}(\lambda)\,d\lambda \\ Z_m &= \int I(\lambda)\,\rho_m(\lambda)\,\bar{z}(\lambda)\,d\lambda \end{aligned} \tag{1}$$

For most applications, the 1971 2° standard observer curves are used, and these may be found in Wyszecki and Stiles[20].

Eq. (1) is very useful for determining metameric color matches, but it does not give us an absolute scale for color appearance. For example, there is a strong tendency for viewers to discount the illuminant in their observations, and the color one sees depends strongly on the ambient lighting and the surround. For example, Eq. (1) might compute a yellow-orange color for a white patch under a tungsten illuminant, while a human observer would still call it "white" if they were in a room lit by the same tungsten source. In fact, a standard photograph of the patch would show its true yellow-orange color, and most novice photographers have the experience of being startled when the colors they get back from their indoor snapshots are not as they remembered them.

To provide for the viewer's chromatic adaptation and thus avoid a color cast in our image after all our hard work, we apply a von Kries style linear transform to our values prior to display[17]. This transform takes an *XYZ* material color computed under our scene illuminant, and shifts it to the equivalent, apparent color XYZ' under a different illuminant that corresponds to our display viewing condition. All we need are the *XYZ* colors for white under the two illuminants as computed by Eq. (1) with $\rho_m(\lambda) = 1$, and a 3×3 transformation matrix, M_C, that takes us from *XYZ* to an appropriate color space for chromatic adaptation. (We will discuss the choice of M_C shortly.) The combined adaptation and display transform is given in Eq. (2), below:

$$\begin{bmatrix} R'_m \\ G'_m \\ B'_m \end{bmatrix} = M_D\, M_C^{-1} \begin{bmatrix} \frac{R'_w}{R_w} & 0 & 0 \\ 0 & \frac{G'_w}{G_w} & 0 \\ 0 & 0 & \frac{B'_w}{B_w} \end{bmatrix} M_C \begin{bmatrix} X_m \\ Y_m \\ Z_m \end{bmatrix}, \tag{2}$$

where

$$\begin{bmatrix} R_w \\ G_w \\ B_w \end{bmatrix} = M_C \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix}$$

for the scene illuminant, and similarly for the display white point, (X'_w, Y'_w, Z'_w).

The display matrix, M_D, that we added to the standard von Kries transform, takes us from CIE *XYZ* coordinates to our display color space. For an *sRGB* image or monitor with D65 white point[15], one would use the following matrix, followed by a gamma correction of $1/2.2$:

$$M_{sRGB} = \begin{bmatrix} 3.2410 & -1.5374 & -0.4986 \\ -0.9692 & 1.8760 & 0.0416 \\ 0.0556 & -0.2040 & 1.0570 \end{bmatrix}$$

If we are rendering a high dynamic-range scene, we may need to apply a tone-mapping operator such as Larson et al[6] to compress our values into a displayable range. The tone operator of Pattanaik et al even incorporates a partial chromatic adaptation model[12].

The choice of which matrix to use for chromatic adaptation, M_C, is an interesting one. Much debate has gone on in the color science community over the past few years as to which space is most appropriate, and several contenders seem to perform equally well in side-by-side experiments[2]. However, it seems clear that *RGB* primary sets that are "sharper" (more saturated) tend to be more plausible than primaries that are inward of the spectral locus[4]. In this paper, we have selected the *Sharp* adaptation matrix for M_C, which was proposed based on spectral sharpening of color-matching data[17]:

$$M_{\text{Sharp}} = \begin{bmatrix} 1.2694 & -0.0988 & -0.1706 \\ -0.8364 & 1.8006 & 0.0357 \\ 0.0297 & -0.0315 & 1.0018 \end{bmatrix}$$

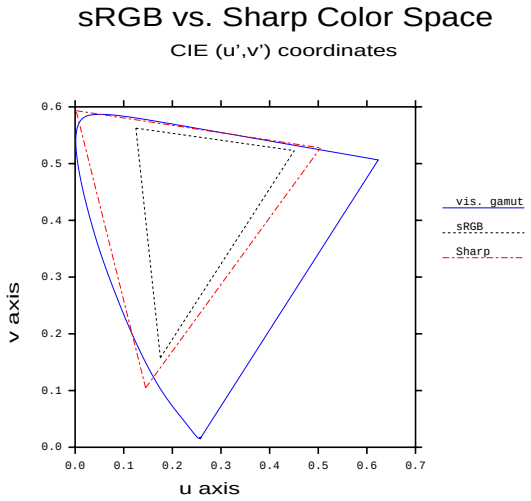


Figure 1: *A plot showing the relative gamuts of the* sRGB *and Sharp color spaces.*

Figure 1 shows a CIE (u',v') plot with the locations of the *sRGB* and Sharp color primaries relative to the visible gamut. Clearly, one could not manufacture a color monitor with Sharp primaries, as they lie just outside the spectral locus. However, this poses no problem for a color transform or a rendering calculation, since we can always transform back to a displayable color space.

In fact, the Sharp primaries may be preferred for rendering and *RGB* image representation simply because they include a larger gamut than the standard *sRGB* primaries. This is not an issue if one can represent color values less than zero and greater than one, but most image formats and some rendering frameworks do not permit this. As we will see in Section 3, the choice of color space plays a significant role in the final image accuracy, even when gamut is not an issue.

2.2. Application to Rendering

We begin with the assumption that the direct-diffuse component is most important to color and overall rendering accuracy. Inside the shader of a conventional *RGB* rendering system, the direct-diffuse component is computed by multiplying the light source color by the diffuse material color, where color multiplication happens separately for each of the three *RGB* values. If this calculation is accurate, it must give the same result one would get using Eq. (1) followed by conversion to the rendering color space. In general, this will not be the case, because the diffuse *RGB* for the surface will be based on some other illuminant whose spectrum does not match the one in the model.

For example, the CIE (x,y) chromaticities and Y-reflectances published on the back of the Macbeth ColorChecker chart[9] are measured under standard illuminant *C*, which is a simulated overcast sky. If a user wants to use the Macbeth color Purple in his *RGB* rendering of an interior space with an incandescent (tungsten) light source, he might convert the published (Y,x,y) reflectances directly to *RGB* values using the inverse of M_{sRGB} given earlier. Unfortunately, he makes at least three mistakes in doing so. First, he is forgetting to perform a white point transform, so there is a slight red shift as he converts from (Y,x,y) under the bluish illuminant *C* to the more neutral D65 white point of *sRGB*. Second, the tungsten source in his model has a slight orange hue he forgets to account for, and there should be a general darkening of the surface under this illuminant, which he fails to simulate. Finally, the weak output at the blue end of a tungsten spectrum makes purple very difficult to distinguish from blue, and he has failed to simulate this metameric effect in his rendering. In the end, the rendering shows something more like violet than the dark blue one would actually witness for this color in such a scene.

If the spectra of all the light sources are equivalent, we can precompute the correct result for the direct-diffuse component and replace the light sources with neutral (white) emitters, inserting our spectrally prefiltered *RGB* values as the diffuse reflectances in each material. We need not worry about how many spectral samples we can afford, since we only have to perform the calculation once for each material in a preprocess. If we intend to render in our display color space, we may even perform the white balance transform ahead of time, saving ourselves the final 3×3 matrix transform at each pixel.

In Section 3, we analyze the error associated with three different color spaces using our spectral prefiltering method, and compare it statistically to the error from naive rendering. The first color space we apply is CIE *XYZ* space, as recommended by Borges[1]. The second color space we use is linear *sRGB*, which has the CCIR-709 *RGB* color primaries that correspond to nominal CRT display phosphors[15]. The third color space is the same one we apply in our white point transformation, the Sharp *RGB* space. We look at cases of direct lighting under a single illuminant, where we expect our technique to perform well, and mixed illuminants with indirect diffuse and specular reflections, where we expect prefiltering to work less effectively.

When we render in CIE *XYZ* space, it makes the most sense to go directly from the prefiltered result of Eq. (1) to *XYZ* colors divided by white under the same illuminant:

$$X_m* = \frac{X_m}{X_w} \qquad Y_m* = \frac{Y_m}{Y_w} \qquad Z_m* = \frac{Z_m}{Z_w}$$

We may then render with light sources using their absolute *XYZ* emissions, and the resulting *XYZ* direct diffuse component will be correct in absolute terms, since they will be remultiplied by the source colors. The final white point adjustment may then be combined with the display color transform exactly as shown in Eq. (2).

When we render in *sRGB* space, it is more convenient to perform white balancing ahead of time, applying both Eq. (1) and Eq. (2) prior to rendering. All light sources that match the spectrum of the dominant illuminant will be modeled as neutral, and spectrally distinct light sources will be modeled as having their *sRGB* color divided by that of the dominant illuminant.

When we render in the Sharp *RGB* space, we can eliminate the transformation into another color space by applying just the right half of Eq. (2) to the surface colors calculated by Eq. (1):

$$\begin{bmatrix} R_m* \\ G_m* \\ B_m* \end{bmatrix} = \begin{bmatrix} \frac{1}{R_w} & 0 & 0 \\ 0 & \frac{1}{G_w} & 0 \\ 0 & 0 & \frac{1}{B_w} \end{bmatrix} M_{\text{Sharp}} \begin{bmatrix} X_m \\ Y_m \\ Z_m \end{bmatrix},$$

Dominant illuminants will again be modeled as neutral, and spectrally distinct illuminants will use:

$$R_s* = \frac{R_s}{R_w} \qquad G_s* = \frac{G_s}{G_w} \qquad B_s* = \frac{B_s}{B_w}$$

The final transformation to the display space will apply the

remaining part of Eq. (2):

$$\begin{bmatrix} R_d \\ G_d \\ B_d \end{bmatrix} = M_D M_{\text{Sharp}}^{-1} \begin{bmatrix} R'_w & 0 & 0 \\ 0 & G'_w & 0 \\ 0 & 0 & B'_w \end{bmatrix} \begin{bmatrix} R'_m \\ G'_m \\ B'_m \end{bmatrix}.$$

3. Results

Our test scene was constructed using published spectral data and simple geometry. It consists of a square room with two light sources and two spheres. One sphere is made of a smooth plastic with a 5% specular component, and the other sphere is made of pure, polished gold (24 carat). The diffuse color of the plastic ball is Macbeth Green[9]. The color of elemental gold is computed from its complex index of refraction as a function of wavelength. The ceiling, floor, and far wall are made of the Macbeth Neutral.8 material. The left wall is Macbeth Red, and the right wall is Macbeth Blue. The near wall, seen in the reflection of the spheres, is the Macbeth BlueFlower color. The left light source is a $2856^\circ K$ tungsten source (i.e., Standard Illuminant A). The right light source is a cool white fluorescent.

All spectral data for our scene were taken from the material tables in Appendix G of Glassner's *Principles of Digital Image Synthesis*[5], and these are also available in the Materials and Geometry Format (MGF)[18]. For convenience, the model used in this paper has been prepared as a set of MGF files and included with our image comparisons in the supplemental materials.

Figure 2 shows a Monte Carlo path tracing of this environment with fluorescent lighting using 69 evenly spaced spectral samples from 380 to 720 nm, which is the resolution of our input data. Using our spectral prefiltering method with the cool white illuminant, we recomputed the image using only three *sRGB* components, taking care to retrace exactly the same ray paths. The result shown in Figure 3 is nearly indistinguishable from the original, with the possible exception of the reflection of the blue wall in the gold sphere. This can be seen graphically in Figure 5, which plots the CIE 1994 Lab $\Delta E*$ color difference[10] in false color. A $\Delta E*$ value of one is just noticeable if the colors are adjacent, and we have found values above five or so to be visible in side-by-side image comparisons.

Using a naive assumption of an equal-energy illuminant, we recomputed the *sRGB* material colors from their reflectance spectra and rendered the scene again, arriving at Figure 4. The rendering took the same time to finish, about a third as long as the full-spectral rendering, and the results are quite different. Both the red wall and the green sphere have changed lightness and saturation from the reference image, the blue wall is reflected as purple in the gold sphere, and the $\Delta E*$ errors shown in Figure 6 are over 20 in large regions. Clearly, this level of accuracy is unacceptable for critical color evaluations, such as selecting a color to repaint the living room.

Illum	Method	*XYZ*		*sRGB*		Sharp	
		50%	98%	50%	98%	50%	98%
tung	naive	10.4	45.9	4.8	15.4	0.8	5.1
	prefilt	2.3	5.7	0.6	1.5	0.5	0.9
fluor	naive	6.1	32.0	5.8	39.2	1.1	6.0
	prefilt	2.0	6.6	0.4	1.2	0.4	0.8
both	naive	5.6	31.6	4.5	21.5	0.6	2.8
	prefilt tung	4.9	15.1	0.5	2.0	0.7	2.2
	prefilt fluor	4.8	55.1	0.6	6.5	0.7	8.6
	Average	5.7	27.4	2.8	12.5	0.7	3.8

Table 1: *CIE 1994 Lab $\Delta E*$ percentiles for our example scene.*

We repeated the same comparisons in CIE *XYZ* and Sharp *RGB* color spaces, then changed the lighting configuration and ran them again. Besides the fluorescent-only lighting condition, we looked at tungsten-only and both sources together. Since the lumen output of the two sources is equal, it was not clear which one to choose as the dominant illuminant, so we applied our prefiltering technique first to one source then to the other. Altogether, we compared 21 combinations of light sources, color spaces, and rendering methods to our multispectral reference solution. The false color images showing the $\Delta E*$ for each comparison are included in the supplemental materials, and we summarize the results statistically in Table 1 and Figure 7.

Table 1 gives the 50th percentile (median) and 98th percentile $\Delta E*$ statistics for each combination of method, lighting, and color space. These columns are averaged to show the relative performance of the three rendering color spaces at the bottom. Figure 7 plots the errors in Table 1 as a bar chart. The 50th percentile errors are coupled with the 98th percentile errors in each bar. In all but one simulation, the Sharp *RGB* color space keeps the median error below the detectable threshold, and the majority of the Sharp renderings have 98% of their pixels below a $\Delta E*$ of five relative to the reference solution, a level at which it is difficult to tell the images apart in side-by-side comparisons. The smallest errors are associated with the Sharp color space and spectral prefiltering with a single illuminant, where 98% of the pixels have errors below the detectable threshold. In the mixed illuminant condition, spectral prefiltering using tungsten as the dominant illuminant performs slightly better than a naive assumption, and prefiltering using cool white as the dominant illuminant performs slightly worse. The worst performance by far is seen when we use CIE *XYZ* as the rendering space, which produces noticeable differences above five for over 2% of the pixels in every simulation, and a *median* $\Delta E*$ over five in each naive simulation.

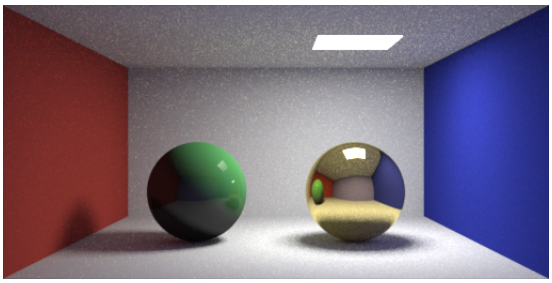


Figure 2: *Our reference multi-spectral solution for the fluorescent-only scene.*

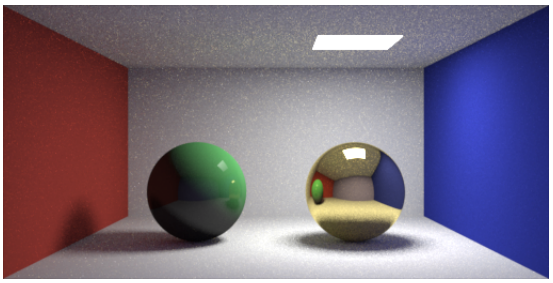


Figure 3: *Our prefiltered* sRGB *solution for the fluorescent-only scene.*

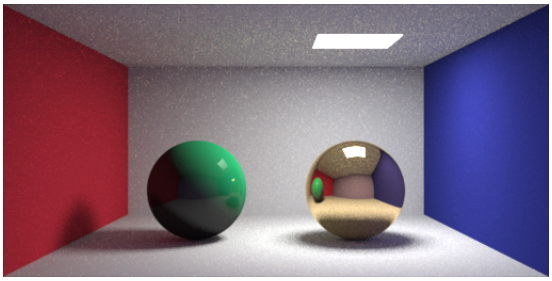


Figure 4: *Our naive* sRGB *solution for the fluorescent-only scene.*

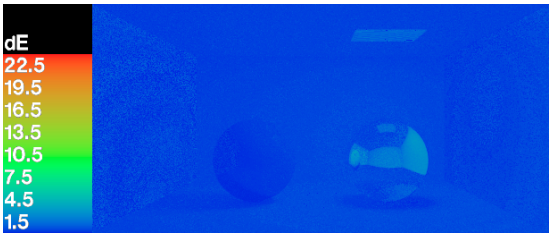


Figure 5: *The* $\Delta E*$ *error for the prefiltered* sRGB *solution.*

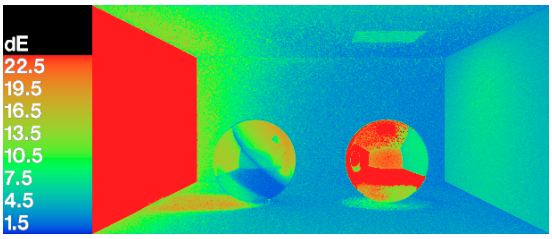


Figure 6: *The* $\Delta E*$ *error for the naive* sRGB *solution.*

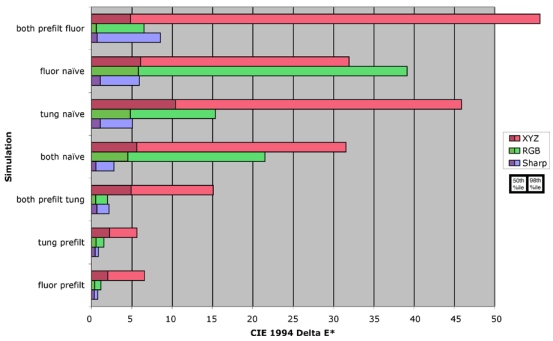


Figure 7: *Error statistics for all solutions and color spaces.*

4. Conclusions

In our experiments, we found spectral prefiltering to minimize color errors in scenes with a single dominant illuminant spectrum, regardless of the rendering color space. The median CIE 1994 Lab $\Delta E*$ values were reduced by a factor of six on average, to levels that were below the detectable threshold when using the *sRGB* and Sharp color spaces. Of the three color spaces we used for rendering, the CIE *XYZ* performed the worst, generating median errors that were above the detectable threshold even with prefiltering, and five times the threshold without prefiltering, meaning the difference was clearly visible over most of the image in side-by-side comparisons to the reference solution. In contrast, the Sharp *RGB* color space, favored by the color science community for chromatic adaptation transforms, performed exceptionally well in a rendering context, producing median error levels that were at or below the detectable threshold both with and without prefiltering.

We believe the Sharp *RGB* space works especially well for rendering by minimizing the representation error for tristimulus values with axes that are aligned along the densest regions of *XYZ* space, perceptually. This property is held in common with the AC_1C_2 color space recommended by Meyer for rendering for this reason[11]. In fact, the AC_1C_2 space has also been favored for chromatic adaptation, indicating the strong connection between rendering calculations and von Kries style transforms. This is evident in the diagonal matrix of Eq. (2), where white point primaries are mul-

tiplied in separate channels, analogous to the color calculations inside a three-component shader. Just as a white point shifts in a von Kries calculation, so do colors shift as they are reflected by a material.

The combination of spectral prefiltering and the Sharp *RGB* space is particularly effective. With prefiltering under a single illuminant, 98% of the pixels were below the detectable error threshold using the Sharp *RGB* space, and only a single highlight in the gold sphere was distinguishable in our side-by-side comparisons. We included a polished gold sphere because we knew its strong spectral selectivity and specularity violated one of our key assumptions, which is that the direct-diffuse component dominates the rendering. We saw in our results that the errors using prefiltering for the gold sphere are no worse than without, and it probably does not matter whether we apply our prefiltering method to specular colors or not, since specular materials tend to reflect other surfaces more than light sources in the final image, anyway. However, rendering in a sharpened *RGB* space always seems to help.

We also tested the performance of prefiltering when we violated our second assumption of a single, dominant illuminant spectrum. When both sources were present and equally bright, the median error was still below the visible threshold using prefiltering in either the *sRGB* or Sharp color space. Without prefiltering, the median jumped significantly for the *sRGB* space, but was still below threshold for Sharp *RGB* rendering. Thus, prefiltering performed no worse on average than the naive approach for mixed illuminants, which was our goal as stated in the introduction.

In conclusion, we have presented an approach to *RGB* rendering that works within any standard framework, adding virtually nothing to the computation time while reducing color difference errors to below the detectable threshold in typical environments. The spectral prefiltering technique accommodates sharp peaks and valleys in the source and reflectance spectra, and user-selection of a dominant illuminant avoids most white balance problems in the output. Rendering in a sharpened *RGB* space also greatly improves color accuracy, independent of prefiltering. Work still needs to be done in the areas of mixed illuminants and colored specular reflections, and we would like to test our method on a greater variety of example scenes.

Acknowledgments

The authors would like to thank Maryann Simmons for providing timely reviews of the paper in progress, and Albert Meltzer for critical editing and LaTeX formatting assistance. We also wish to thank the anonymous reviewers, who we hope will make themselves nonymous at the workshop so we may discuss their points in person, as there was not room to include such discussion in a short paper.

References

1. C. Borges. Trichromatic Approximation for Computer Graphics Illumination Models. *Proc. Siggraph '91*.
2. Anthony J. Calabria and Mark D. Fairchild. Herding CATs: A Comparison of Linear Chromatic-Adaptation Transforms for CIECAM97s. *Proc. 9th Color Imaging Conf.*, pp. 174–178, 2001.
3. Kristin J. Dana, Bram van Ginneken, Shree K. Nayar and Jan J. Koenderink. Reflectance and Texture of Real World Surfaces. *ACM TOG*, **15**(1):1–34, 1999.
4. G. D. Finlayson and P. Morovic. Is the Sharp Adaptation Transform more plausible than CMCCAT2000? *Proc. 9th Color Imaging Conf.*, pp. 310–315, 2001.
5. Andrew S. Glassner. Principles of Digital Image Synthesis. Morgan Kaufmann, 1995.
6. G. W. Larson, H. Rushmeier and C. Piatko. A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes. *IEEE Transactions on Visualization and Computer Graphics*, **3**(4) (December 1997).
7. Laurence T. Maloney. Evaluation of Linear Models of Surface Spectral Reflectance with Small Numbers of Parameters. *J. Optical Society of America A*, **3**(10):1673–1683 (October 1986).
8. David Marimont and Brian Wandell. Linear Models of Surface and Illuminant Spectra. *J. Optical Society of America A*, **9**(11):1905–1913 (November 1992).
9. C. S. McCamy, H. Marcus and J. G. Davidson. A color-rendition chart. *J. Applied Photographic Engineering*, **2**(3):95–99 (summer 1976).
10. R. McDonald and K. J. Smith. CIE94 - a new colur-difference formula. *Soc. Dyers Col.*, **111**:376-9, Dec 1995.
11. Gary Meyer. Wavelength Selection for Synthetic Image Generation. *Computer Vision, Graphics and Image Processing*, **41**:57–79, 1988.
12. Sumanta N. Pattanaik, James A. Ferwerda, Mark D. Fairchild and Donald P. Greenberg. A multiscale model of adaptation and spatial vision for realistic image display. *Proc. Siggraph '98*.
13. Mark S. Peercy. Linear color representations for full speed spectral rendering. *Proc. Siggraph '93*.
14. Brian Smits. An RGB to Spectrum Conversion for Reflectances. *J. Graphics Tools*, **4**(4):11–22, 1999.
15. Michael Stokes et al. A Standard Default Color Space for the Internet – sRGB. Ver. 1.10, November 1996. `http://www.w3.org/Graphics/Color/sRGB`.
16. S. Sueeprasan and R. Luo. Incomplete Chromatic Adaptation under Mixed Illuminations. *Proc. 9th Color Imaging Conf.*, pp. 316–320, 2001.

17. S. Süsstrunk, J. Holm and G. D. Finlayson. Chromatic Adaptation Performance of Different *RGB* Sensors. *IS&T/SPIE Electronic Imaging*, SPIE **4300**, Jan. 2001.

18. Greg Ward et al. Materials and Geometry Format. `http://radsite.lbl.gov/mgf`.

19. Harold B. Westlund and Gary W. Meyer. A BRDF Database Employing the Beard-Maxwell Reflection Model. *Graphics Interface 2002*.

20. Günter Wyszecki and W. S. Stiles. Color Science: Concepts and Methods, Quantitative Data and Formulae. John Wiley & Sons, New York, 2nd ed., 1982.

Appendix D:

Detail to Attention: Exploiting Visual Tasks for Selective Rendering

Reprinted from 2003 Eurographics Symposium on Rendering

Citation:

Cater, Kirsten, Alan Chalmers, Greg Ward, "Detail to Attention: Exploiting Visual Tasks for Selective Rendering," *Fourteenth Eurographics Symposium on Rendering* (2003), P. Christensen and D. Cohen-Or (Editors), June 2003.

Eurographics Symposium on Rendering 2003
Per Christensen and Daniel Cohen-Or (Editors)

Detail to Attention: Exploiting Visual Tasks for Selective Rendering

K. Cater,[1†] A. Chalmers[1] and G. Ward[2]

[1] Department of Computer Science, The University of Bristol, UK
[2] Anyhere Software, U.S.A

Abstract

*The perceived quality of computer graphics imagery depends on the accuracy of the rendered frames, as well as the capabilities of the human visual system. Fully detailed, high fidelity frames still take many minutes even hours to render on today's computers. The human eye is physically incapable of capturing a moving scene in full detail. We sense image detail only in a 2° foveal region, relying on rapid eye movements, or saccades, to jump between points of interest. Our brain then reassembles these glimpses into a coherent, but inevitably imperfect, visual percept of the environment. In the process, we literally lose sight of the unimportant details. In this paper, we demonstrate how properties of the human visual system, in particular **inattentional blindness**, can be exploited to accelerate the rendering of animated sequences by applying **a priori** knowledge of a viewer's task focus. We show in a controlled experimental setting how human subjects will consistently fail to notice degradations in the quality of image details unrelated to their assigned task, even when these details fall under the viewers' gaze. We then build on these observations to create a perceptual rendering framework that combines predetermined **task maps** with spatiotemporal contrast sensitivity to guide a progressive animation system which takes full advantage of image-based rendering techniques. We demonstrate this framework with a **Radiance** ray-tracing implementation that completes its work in a fraction of the normally required time, with few noticeable artifacts for viewers performing the task.*

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation - Viewing Algorithms I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - Animation I.4.8 [Image Processing and Computer Vision]: Scene Analysis -Time-varying imagery

1. Introduction

One of the central goals in computer graphics is to produce the best *perceived* image in the least amount of time. Advanced rendering techniques such as ray-tracing and global illumination improve image quality, but at a commensurate cost. In many cases, we end up spending significant effort improving details the viewer will never notice. If we can find a way to apply our effort selectively to the small number of regions a viewer attends in a given scene, we can improve the perceived quality without paying the full computational price.

Most computer graphics serve some specific visual task

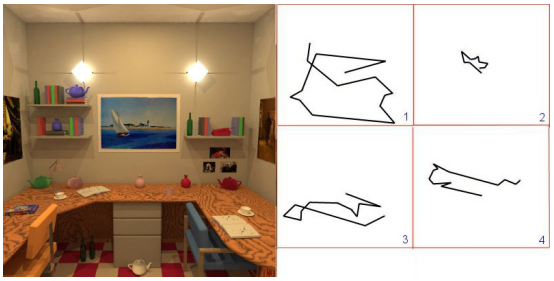


Figure 1: *Effects of a task on eye movements. Eye scans for observers examined with different task instructions; 1. Free viewing, 2. Remember the central painting, 3. Remember as many objects on the table as you can, 4. Count the number of books on the shelves.*

† cater@cs.bris.ac.uk

– telling a story, advertising a product, playing a game, or simulating an activity such as flying. In the majority of cases, objects relevant to the task can be identified in advance, and the human visual system focuses its attention on these objects at the expense of other details in the scene. Figure 1 shows a rendered image for which we instructed participants to perform a number of arbitrary tasks. The eye-tracking scans demonstrate that subjects focus on task-related objects and fail to attend other details in the scene. In this paper, we show experimentally that it is possible to render scene objects not related to the task at lower resolution without the viewer noticing any reduction in quality.

We take advantage of these findings in a computational framework that applies high-level task information to deduce error visibility in each frame of a progressively rendered animation. By this method, we are able to generate high quality animated sequences at constant frame rates in a fraction of the time normally required. A key advantage to this technique is that it only depends on the task, not on the viewer. Unlike the foveal detail rendering used in flight simulators, there is no need for eye-tracking or similar single-viewer hardware to enable this technology, since attentive viewers participating in the same task will employ similar visual processes.

We begin with a review of previous work in perceptually-based rendering, focusing on areas most closely related to our technique. We then present an experimental validation of selective rendering using a *task map* to control image detail. These results are followed by a description and demonstration of our perceptual rendering framework, which extends these ideas to incorporate a model of spatiotemporal contrast sensitivity, enabling us to predict local error visibility. In our implementation of this framework, we use ray-tracing and image-based rendering to compute an animated sequence in two minutes per frame.

2. Previous Work

Visual attention is a coordinated action involving conscious and unconscious processes in the brain, which allow us to find and focus on relevant information quickly and efficiently. If detailed information is needed from many different areas of the visual environment, the eye does not scan the scene in a raster-like fashion, but jumps so that the relevant objects fall sequentially on the fovea. These jumps are called *saccades* [32].

There are two general visual attention processes, labelled *bottom-up* and *top-down*, which determine where humans locate their visual attention [10]. The bottom-up process is purely stimulus driven, for example, a fire in the dark, a red apple in a green tree, or the lips and eyes of another person the most mobile and expressive elements of a face. In all these cases, the visual stimulus captures attention automatically without volitional control. This is evolutionary; the movement may be danger lurking behind a bush, or we may need to find ripe fruit for our meal. In contrast, the top-down process is under voluntary control, and focuses attention on one or more objects that are relevant to the observer's goal when studying a scene. Such goals might include looking for a lost child, searching for an exit, or counting the number of books on a shelf, as shown in Figure 1.

General knowledge of the human visual system has been used to improve the quality of the rendered image [6, 8, 17, 18, 23, 24]. Other research has investigated how complex model detail can be reduced without any reduction in the viewer's perception of the models [13, 21, 25, 30]. Along these lines, Maciel and Shirley's visual navigation system used texture mapped primitives to represent clusters of objects to maintain high and approximately constant frame rates [14]. The application of visual attention models in computer graphics has so far exploited only peripheral vision and the bottom-up visual attention process, as we discuss below.

2.1. Peripheral Vision

Due to the fact that the human eye only processes detailed information from a relatively small part of the visual field, it is possible to reduce detail in the periphery without upsetting visual processing. In numerous studies, Loschky and McConkie [12] used an eye-linked, multiple resolution display that produces high visual resolution only in the region to which the eyes are directed. They were able to show that photographic images filtered with a window radius of 4.1° produced results statistically indistinguishable from that of a full, high-resolution display. The display they propose does, however, encounter the problem of updating the multi-resolution image after an eye movement without disturbing the visual processing. Their work has shown that the image needs to be updated after an eye saccade within 5 milliseconds of a fixation, otherwise the observer will detect the change in resolution. These high update rates were only achievable using an extremely high temporal resolution eye tracker, and pre-storing all possible multi-resolution images that were to be used.

In another experiment, Watson et al. [29] evaluated the effectiveness of high detail insets in head-mounted displays. The high detail inset they used was rectangular and was always presented at the finest level of resolution. Three inset conditions were investigated: a large inset - half the complete display's height and width, a small inset size - 30 % of the complete display's height and width, and no inset at all. The level of peripheral resolution was varied at: fine resolution 320x240, medium resolution 192x144 and coarse resolution 64x48. Their results showed that although observers found their search targets faster and more accurately in a full high resolution environment, this condition was not significantly better than the high-resolution inset displays with either medium or low peripheral resolutions.

2.2. Saliency Models

Low-level saliency models determine what visual features will involuntarily attract our attention in a scene. Visual psychology researchers such as Yarbus [32], Itti and Koch [9] and Yantis [31] showed that the visual system is highly sensitive to features such as edges, abrupt changes in color, and sudden movements. This low-level visual processing has been exploited in computer graphics by Yee et al. [33] to accelerate animation renderings with global illumination, by applying a model of visual attention to identify conspicuous regions. Yee constructs a spatiotemporal error tolerance map, called the *Aleph map*, from spatiotemporal contrast sensitivity and a low-level *saliency map*, for each frame in an animation. The saliency map is obtained by combining the conspicuity maps of intensity, color, orientation and motion. The Aleph map is then used as a guide to indicate where more rendering effort should be spent in computing the lighting solution, significantly improving the computational efficiency during animation. Subsequent work by Marmitt and Duchowski [16] showed, however, that such bottom-up visual attention models do not always predict attention regions in a reliable manner.

Our rendering framework in Section 4 extends Yee's work, modeling task-level saliency rather than automatic visual processes, and deriving a map of *error conspicuity* in place of error tolerance. This permits us to finish a frame when errors have become invisible, or render the best possible frame in a fixed period of time optimizations Yee's method does not support.

2.3. Inattentional Blindness

In 1967, the Russian psychologist Yarbus recorded the fixations and saccades observers made while viewing natural objects and scenes. Observers were asked to answer a number of different questions concerning the depicted situation in Repin's picture "An Unexpected Visitor" [32]. This resulted in substantially different saccade patterns, each one being easily construable as a sampling of those picture objects that were most informative for the answering of the question, as shown in Figure 2.

Cater et al. [1] showed that conspicuous objects in a scene that would normally attract the viewer's attention are ignored if they are not relevant to the task at hand. In their experiments, viewers were presented with two animations. One was a full, high-quality rendering, while in the other, only the pixels in visual angle of the fovea (2°) centered around the location of a task within the environment were rendered at high quality. This high quality was blended to a much lower quality in the rest of the image. They showed that when observers were performing the task within an animation, their visual attention was fixed exclusively on the area of the task, and they consistently failed to notice the significant difference in rendering quality between the two animations.

We have extended the work in Cater et al. [1] to be able to distinguish between the effects of peripheral vision and *inattentional blindness*, which is the failure of an observer to see unattended items in a scene [15]. We present our results in the following section, where we substitute still images for the animation to ensure that the observed effect is not merely a result of resolution loss in the periphery, but a true exhibition of inattentional blindness.

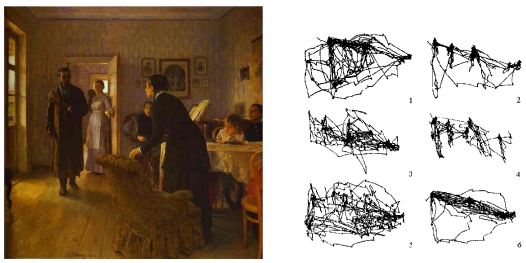


Figure 2: *Repin's picture was examined by subjects with different instructions; 1. Free viewing, 2. Judge their ages, 3. Guess what they had been doing before the unexpected visitor's arrival, 4. Remember the clothes worn by the people, 5. Remember the position of the people and objects in the room, 6. Estimate how long the visitor had been away* [32].

3. Task Maps: Experimental Validation

In this section, we demonstrate inattentional blindness experimentally in the presence of a high-level task focus. Our hypothesis was that viewers would not notice normally visible degradations in an image that did not affect the clarity of the objects we instructed them to seek. The experiments confirmed our hypothesis with a high level of certainty. An appropriate conjunctive search was selected as the task, with no pre-attentive cues, such as color, to differentiate the task objects from the other objects in the scene, this prevented any pop-out effects [26]. The task chosen for this experiment was to count the number of teapots in a computer generated scene. For ease of experimental setup a still image was used, however, previous work has proven that this method works just as well for animations [1].

A pre-study was run with 10 participants to find out how long subjects took to perform the task, this was found to be on average 2 seconds to count the five teapots in the image. A pilot study was then conducted to deduce the appropriate image resolution to use for the main experiment. 32 participants were shown 24 pairs of images at random, and asked if they could distinguish a change in resolution or quality between the two images. Each image was displayed for 2 seconds. One image was always the High Quality image rendered at a 3072x3072 sampling resolution, whilst the other image was one selected from images rendered at sampling resolutions of 256x256, 512x512, 768x768, 1024x1024, 1536x1536 and 2048x2048. In half of the pairs of images, there was no change in resolution; i.e., they saw two 3072x3072 resolution images. The results can be seen in Figure 3.

All the participants could easily detect a quality difference with the resolutions of 256x256 through to 1024x1024 in comparison to a resolution of 3072x3072. 72% still detected a quality difference between a resolution image of 1536x1536 and 3072x3072. However, it was decided that we would use a resolution of 1024x1024 in our main study as 100% of participants in the pilot study detected the difference.

The main study involved two models of an office scene, the only difference being the location of items in the scene, mainly teapots (Figure 4). Each scene was then rendered to three different levels of resolution quality, the entire scene at High Quality (HQ), a sampling resolution of 3072x3072 (Figure 4), the entire scene at Low Quality (LQ), a sampling resolution of 1024x1024 (Figure 6b), and Selective Quality (SQ). The Selective Quality image was created by selectively rendering the majority of the scene in low quality (1024x1024) apart from the visual angle of the fovea (2°) centered on each teapot, shown by the black circles in Figure 5, which were rendered at the higher rate corresponding to 3072x3072 sampling. The high quality images took 8.6 hours to render with full global illumination in *Radiance* [28] on a 1 GHz Pentium processor, whilst the images for the low quality were rendered in half this time, and the Selective Quality in 5.4 hours.

In the study, a total of 96 participants were considered. Each subject saw two images, each displayed for 2 seconds. Table 1 describes the conditions tested with 32 subjects for the HQ/HQ condition and 16 subjects for the other conditions. We know from the pilot study that all participants should be able to detect the rendering quality difference if given no task; i.e., they are simply looking at the images for 2 seconds. The task chosen to demonstrate the effect of inattentional blindness had the subjects counting teapots located all around the scene. There were 5 teapots in both images. By placing the teapots all over the scene, we were able to see whether or not having to scan the whole image, and thus fixate on low quality as well as high quality regions, would mean that the viewers would indeed be able to detect the rendering quality difference. To minimize experimental bias, the choice of which condition to run was randomized, and for each 8 were run in the morning and 8 in the afternoon. Subjects had a variety of experience with computer graphics, and all exhibited normal or corrected vision in testing.

Before beginning the experiment, the subjects read a sheet of instructions on the procedure of the particular task they were to perform. After each participant had read the instructions, they were asked to clarify that they understood the task. They then placed their head on a chin rest that was located 45cm away from a 17-inch monitor. The chin rest was located so that their eye level was approximately level with the centre of the screen. The participants' eyes were allowed to adjust to the ambient lighting conditions before the experiment was begun. The first image was displayed for 2 seconds, then the participant stated out loud how many teapots they saw. Following this, the second image was displayed for 2 seconds, during which the task was repeated.

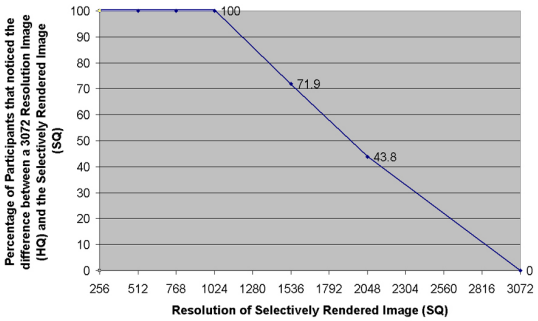


Figure 3: *Results from the pilot study: determining a consistently detectable rendering resolution difference.*

Figure 4: *High Quality (HQ) image rendered with a sampling resolution of 3072x3072.*

On completion of the experiment, each participant was asked to fill out a detailed questionnaire. This questionnaire asked for some personal details including age, sex, and level of computer graphics knowledge. The participants were then asked detailed questions about the quality of the two images they had seen. Finally, the subjects were shown a high quality and a low quality image side-by-side and asked which one they saw for the first and second displayed images. This was to confirm that participants had not simply failed to remember that they had noticed a quality difference, but actually could not distinguish the correct image when shown it from a choice of two.

3.1. Results

Figure 7 shows the overall results of the experiment. Obviously, the participants did not notice any difference in the rendering quality between the two HQ images (they were

the same). Of interest is the fact that, apart from two cases in the HQ/SQ conditions, the viewers performing the task consistently failed to notice any difference between the HQ rendered image and the SQ image. Surprisingly, nearly 20% of the viewers in the HQ/LQ condition were so engaged in the task that they failed to notice any difference between these very different quality image.

Figure 5: *Selective Quality (SQ) image showing the high quality rendered circles located over the teapots.*

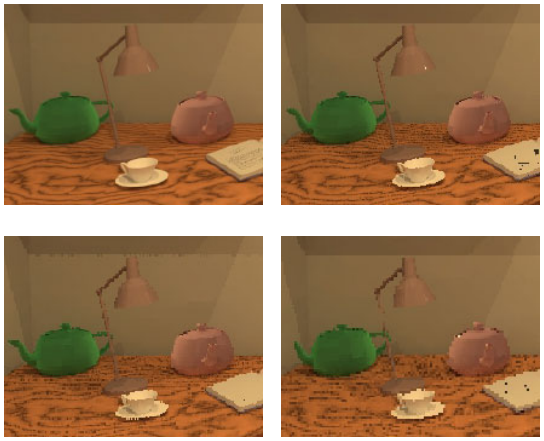


Figure 6: *Sampling resolutions: a(top left) 3072x3072 (HQ), b(top right) 1024x1024 (LQ), c(bottom left) 768x768 (LQ), d(bottom right) 512x512 (LQ)*

3.2. Statistical Analysis

Statistical analysis shows where our results are significant. The appropriate method of analysis is a "paired samples" *t-test for significance*, and since each subject had a different random selection of the images, an unrelated t-test was applied [2]. By performing comparisons of the other image pairings to the HQ/HQ data, we could determine whether the results were statistically significant.

When the observers were counting teapots, the difference between HQ/HQ and HQ/LQ counts were statistically very significant. For a two-tailed test with the $df = 62$ (df is related to the number of subjects), t must be greater than or equal to 2.0 for significance with $p < 0.05$ (less than 5% chance of random occurrence). The result for the pair-wise comparison of HQ/HQ and HQ/LQ was $t = 11.6$ with $p < 0.05$.

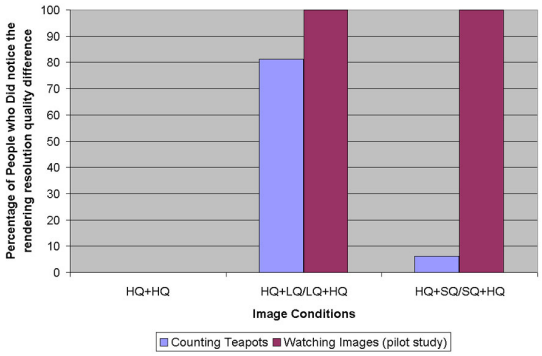


Figure 7: *Experimental results for the two tasks: counting the teapots vs. simply looking at the images.*

Acronym	Description
HQ	**High Quality:** Entire animation rendered at a sampling resolution of 3072x3072.
LQ	**Low Quality:** Entire animation rendered at a sampling resolution of 1024x1024.
SQ	**Selective Quality:** A sampling resolution of 1024x1024 all over the image apart from the visual angle of the fovea (2°) centered around each teapot, shown by the circles in Figure 5, which are rendered to a sampling resolution of 3072x2072.

Table 1: *The ordering image pairs shown in the experiment were: (1)HQ/HQ, (2)HQ/LQ, (3)LQ/HQ, (4)HQ/SQ, (5)SQ/HQ*

However, if we analyze statistics on the pair-wise comparison of HQ/HQ and HQ/SQ, the results are not statistically significant the *null hypothesis* is retained, as $t = 1.4$, $df = 62$, and $p > 0.1$. From this we can conclude that when observers were counting teapots, the HQ/HQ images and the HQ/SQ images produced the same result; i.e., the observers thought they were seeing the same pair twice, with no alteration in rendering quality. However, when the observers were simply looking at the images without searching for teapots in the pilot study, the result was significantly different; i.e., the observers could distinguish that they were shown two images rendered at different qualities.

An additional experiment was run to see at what value

the results became significantly different from the HQ resolution of 3072x3072. At a sampling resolution of 768x768 (Figure 6c) the results were only just significant, $t = 2.9$, $df = 62$, and $p < 0.05$. I.e., only 7 participants, out of the 32 people studied, noticed the difference between the high quality image and a selectively rendered image whilst performing the teapot counting task. This only increased to 8 people out of 32 when the sampling resolution was dropped again to 512x512 (Figure 6d)!

3.3. Verification with an Eye-tracker

To confirm that the attention of an observer was being fully captured by the task of counting teapots, the experiment was repeated using the Eyelink Eyetracking System developed by SR Research Ltd. and manufactured by SensoMotoric Instruments. Figure 8 shows an example of a scan path of an observer whilst performing the counting teapots task for 2 seconds. Whilst all the observers had slightly different scan paths across the images, they fixated both on the teapots and on other objects as well. The vases seemed to be the most commonly non-teapot object fixated upon, due to the fact they were the most similar looking item in the scene to a teapot. It could be deduced that the participants were making fixations on non-teapot objects in the image to make sure whether or not they were in fact a teapot, whatever the case these fixations were not enough for the observers to distinguish the different quality to which they were rendered.

Figure 9 shows the perceptual difference between the selective quality (SQ) and low quality (LQ) images computed using Daly's Visual Difference Predictor [3, 20]. The recorded eye-scan paths clearly cross, and indeed fixate, on areas of high perceptual difference. We can therefore conclude that the failure to distinguish the difference in rendering quality between the teapots, selectively rendered to high quality, and the other low quality objects, is *not* due purely to peripheral vision effects. The observers are fixating on low quality objects, but because they are not relevant to the given task of counting teapots, they fail to notice the reduction in rendering quality. This is inattentional blindness.

These results demonstrate that inattentional blindness, and not just peripheral vision, may be exploited to significantly reduce the rendered quality of a large portion of the scene without having any significant effect on the viewer's perception of the scene.

4. A Perceptual Rendering Framework

By our experiments, we know that selective rendering is cost effective for briefly viewed still images, and in fact task focus seems to override low-level visual attention when it comes to noticing artifacts. In the more general case of animated imagery, we can take even greater advantage of inattentional blindness, because we know the eye preferentially tracks salient objects at the expense of other details [1]. Using Daly's model of human contrast sensitivity for moving images [4, 5], and Yee's insight to substitute saliency for movement-tracking efficacy [33], we can apply our *a priori* knowledge of task-level saliency to optimize the animation process.

Figure 8: *An eye scan for an observer counting the teapots. The X's are fixation points and the lines are the saccades.*

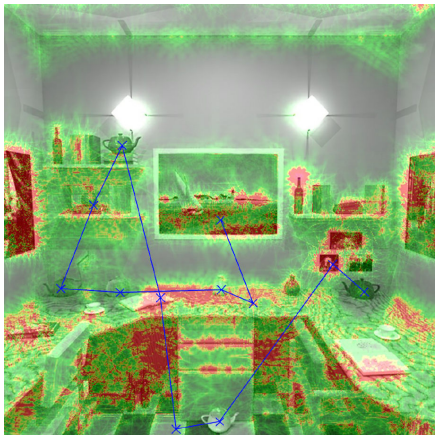


Figure 9: *Perceptual difference between SQ and LQ images using VDP [3]. Red denotes areas of high perceptual difference.*

The approach we describe has a number of key advantages over previous methods using low-level visual perception. First, task-level saliency is very quick to compute, as it is derived from a short list of important objects and their known whereabouts. Second, we have introduced a direct estimate of pixel error (or uncertainty), avoiding the need for expensive image comparisons and Gabor filters as required by other perceptually based methods [33, 18]. Third, we render animation frames progressively, enabling us to specify exactly how long we are willing to wait for each image, or stopping when the error has dropped below the visible threshold. Frames are still rendered in order, but the time spent refining the images is under our control. Our initial implementation of this framework is suitable for quick turnaround animations at about a minute per frame, but it is our eventual goal

to apply these methods to interactive and real-time rendering [22, 27].

We have designed a general framework for progressive rendering that permits iterative frame refinement until a target accuracy or time allotment has been reached. A frame may be refined by any desired means, including improvements to resolution, anti-aliasing, level of detail, global illumination, and so forth. In our demonstration system, we focus primarily on resolution refinement (i.e., samples/pixel), but greater gains are possible by manipulating other rendering variables as well.

4.1. Framework

The diagram shown in Figure 10 shows an overview of our system. The boxes represent data, and the ovals represent processes. The inputs to the system, shown in the upper left, are the viewer's known task, the scene geometry, lighting, and view, all of which are a function of time. The processes shown outside the "Iterate" box are carried out just once for each frame. The processes shown inside the box may be applied multiple times until the frame is considered "ready", by whatever criteria we set. In most cases, we call a frame ready when we have exhausted our time allocation, but we can also break from iteration when our *error conspicuity* (EC) drops below threshold over the entire image.

Our framework is designed to be general, and our implementation is just one realization. We start by explaining the basic methods that are applied once per frame, followed by the interactive methods for frame refinement. This overview pertains to any rendering algorithm one might use, from radiosity to ray-tracing to multi-pass hardware rendering. The Implementation section that follows details some of the specific techniques we used in our ray-tracing realization, and highlights our results.

Referring to Figure 10, our high-level vision model takes the task and geometry as input, and produces a table quantifying relative object importance for this frame. We call this the geometric entity ranking. Specifically, we derive a table of positive real numbers, where zero represents an object that will never be looked at, and 1 is the importance of scene objects unrelated to the task at hand. Normally, only task-relevant objects will be listed in this table, and their importance values will typically be between 1.5 and 3, where 3 is an object that must be followed very closely in order to complete the task.

For the first order rendering, we may use any method that is guaranteed to finish before our time is up. From this initial rendering, we will need an object map and depth value for each pixel. If subsampling is applied and some pixels are skipped, we must separately project scene objects onto a full resolution frame buffer to obtain this map. The pixel motion map, or image flow, is computed from the object map and our knowledge of object and camera movement relative to the previous frame. The object map is also logically combined with the geometric entity ranking to obtain the *task map*. This is usually accessed via a lookup into the ranking table, and does not require actual storage in a separate buffer.

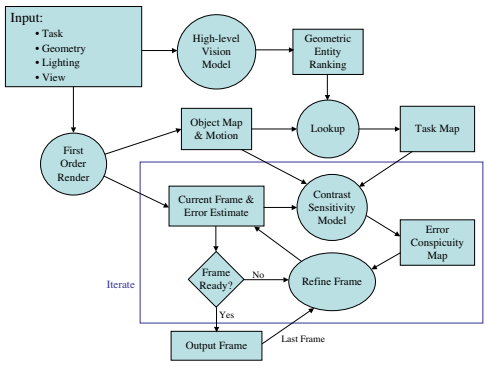


Figure 10: *A framework for progressive refinement of animation frames using task-level information.*

Once we have a first order rendering of our frame and maps with the object ID, depth, motion, and task-level saliency at each pixel, we can proceed with image refinement. First, we compute the relative uncertainty in each pixel estimate. This may be derived from our knowledge of the underlying rendering algorithm, or from statistical measures of variance in the case of stochastic methods. We thought at first that this might pose a serious challenge, but it turns out to be a modest requirement, for the following reason. Since there is no point in quantifying errors that we cannot correct for in subsequent passes, we only need to estimate the difference between what we have and what we might get after further refinement of a pixel. For such improvements, we can usually obtain a reasonable bound on the error. For example, going from a calculation with a constant ambient term to one with global illumination, the change is generally less than the ambient value used in the first pass, times the diffuse material color. Taking half this product is a good estimate of the change we might see in either direction by moving to a global illumination result. Where the rendering method is stochastic, we can collect neighbor samples to obtain a reasonable estimate of the variance in each pixel neighborhood and use this as our error estimate [11]. In either case, error estimation is inexpensive as it only requires local information, plus our knowledge of the scene and the rendering algorithm being applied.

With our current frame and error estimate in hand, we can make a decision whether to further refine this frame, or finish it and start the next one. This "frame ready" decision may be based as we said on time limits or on some overall test of frame quality. In most cases, we will make at least one refinement pass before we move on, applying *image-based rendering* (IBR) to gather useful samples from the previous frame and add them to this one.

In an IBR refinement pass, we use our object motion map to correlate pixels from the previous frame with pixels from this frame. This improves our ability to decide when and where IBR is likely to be beneficial. We base our selection of replacement pixels on the following heuristics:

1. The pixel pair in the two frames corresponds to the same point on the same object, and does not lie on an object boundary.
2. The error estimate for the previous frame's pixel must be less than the error estimate for the current frame's pixel by some set amount. (We use 15%.)
3. The previous frame's pixel must agree with surrounding pixels in the new frame within some tolerance. (We use a 32% relative difference.)

The first criterion prevents us from using pixels from the wrong object or the wrong part of the same object. We test for position correspondence by comparing the transformed depth values, and for object boundaries by looking at neighboring pixels above, below, right, and left in our object map. The second criterion prevents us from degrading our current frame estimate with unworthy prior pixels. The third criterion reduces pollution in shadows and highlights that have moved between frames, though it also limits the number of IBR pixels we take in highly textured regions. If a pixel from the previous frame passes these three tests, we overwrite the current pixel estimate with the previous one, and reset the error to the previous value degraded by the amount used for the second criterion. In this way, IBR pixels are automatically retired as we move from one frame to the next.

Let us assume there is time for further refinement. Once we have transferred what samples we can using IBR, we determine which pixels have noticeable, or conspicuous, errors so we may select these for improvement. Here we combine the spatiotemporal *contrast sensitivity function* (CSF) defined by Daly [4, 5] with our task-level saliency map. Daly's CSF model is a function of two variables, spatial frequency, ρ, and retinal velocity, v_R:

$$CSF(\rho, v_R) = k \cdot c_0 \cdot c_2 \cdot v_R \cdot (c_1 2\pi\rho)^2 exp(-\frac{c_1 4\pi\rho}{\rho_{max}}) \quad (1)$$

where:

$k = 6.1 + 7.3|\log(c_2 v_R/3)|^3$

$\rho_{max} = 45.9/(c_2 v_R + 2)$

$c_0 = 1.14, c_1 = 0.67, c_2 = 1.7$ for CRT at $100 cd/m^2$

Following Yee [33], we substitute saliency for movement-tracking efficacy, based on the assumption that the viewer pays proportionally more attention to task-relevant objects in their view. The equation for retinal image velocity (in$^\circ$/second) thus becomes:

$$v_R = |v_1 - \min(v_1 \cdot S/S_{max} + v_{min}, v_{max})| \quad (2)$$

where:

v_1 = local pixel velocity (from motion map)

S = task-level saliency for this region

S_{max} = max. saliency in this frame, but not less than 1/0.82

$v_{min} = 0.15^\circ$/sec (eye drift velocity)

$v_{max} = 80^\circ$/sec (movement-tracking limit)

The eye's movement tracking efficacy is computed as S/S_{max}, which assumes the viewer tracks the most salient object in view perfectly. Daly [5] recommends an overall value of 82% for the average efficacy when tracking all objects in a scene at once, so we do not allow S_{max} to drop below 1/0.82. This prevents us from predicting perfect tracking over the whole image when no task-related objects are in view.

Since peak contrast sensitivity shifts towards lower frequencies as retinal velocity increases, objects that the viewer is not tracking because they are not important will be visible at lower resolution than our task-relevant objects. However, if the entire image is still or moving at the same rate, the computed CSF will be unaffected by our task information. Because of this, we reintroduce our task map as an additional multiplier in the final error conspicuity map, which we define as:

$$EC = S \cdot \max(E \cdot CSF/ND - 1, 0) \quad (3)$$

where:

E =relative error estimate for this pixel

ND = noticeable difference threshold

Because the relative error multiplied by the CSF yields the normalized contrast, where 1.0 is just noticeable, we introduce a threshold difference value, ND, below which we deem errors to be insignificant. A value of 2 JNDs is the threshold where 94% of viewers are predicted to notice a difference, and this is the value commonly chosen for ND.

To compute the CSF, we also need an estimate of the peak stimulus spatial frequency, ρ. We obtain this by evaluating an image pyramid. Unlike previous applications of the CSF to rendering, we are not comparing two images, so we do not need to determine the relative spatial frequencies in a difference image. We only need to know the uncertainty in each frequency band to bound the visible difference between our current estimate and the correct image. This turns out to be a great time-saver, as it is the evaluation of Gabor filters that usually takes longest in other approaches. Because the CSF falls off rapidly below spatial frequencies corresponding to the foveal diameter of 2°, and statistical accuracy improves at lower frequencies as well, we need only compute our image pyramid up to a ρ of 0.5 cycles/degree.

Our procedure is as follows. We start by clearing our EC map, and subdividing our image into 2° square cells. Within each cell, we call a recursive function that descends a local image pyramid to the pixel level, computing EC values and summing them into our map on the return trip. At each pyramid level, the EC function is evaluated from the stimulus frequency (1/subcell radius in $^\circ$), the task-level saliency, the

combined error estimate, and the average motion for pixels within that subcell. The task-level saliency for a subcell is determined as the maximum of all saliency values within a 2° neighborhood. This may be computed very quickly using a 4-neighbor check at the pixel level, where each pixel finds the maximum saliency of itself and its neighbors 1° up, down, left, and right. The saliency maximum and statistical error sums are then passed back up the call tree for the return evaluation. The entire EC map computation, including a statistical estimation of relative error, takes less than a second for a 640x480 image on a 1 GHz Pentium processor.

5. Implementation

In our implementation of the above framework, we modified the *Radiance* lighting simulation and rendering engine [28] to perform progressive animation. Figure 11 shows a frame from a 4-minute long animation we computed at 640x480 resolution using this software. Figure 12a shows our estimate of relative error at each pixel in the first order rendering, and Figure 12b shows the corresponding error conspicuity map. The viewer was assigned the task of counting certain objects in the scene related to fire safety. There are two task objects visible in this image, the fire extinguisher and the narrator's copter (the checkered ball), so the regions around these objects show strongly in the conspicuity map. Figure 13a shows the final number of samples taken at each pixel in the refined frame, which took two minutes to compute on a single 400 MHz G3 processor. We found this time sufficient to render details on the task-related objects, but too short to render the entire frame accurately. We wanted there to be artifacts in order to demonstrate the effect of task focus on viewer perception. About 50% of the pixels received IBR samples from the previous frame, and 20% received one or more high quality refinement samples.

For comparison, Figure 13b shows the scene rendered as a still image in the same amount of time. Both images contain artifacts, but the animation frame contains fewer sampling errors on the task-related objects. In particular, the fire extinguisher in the corner, which is one of the search objects, has better anti-aliasing than the traditionally rendered image. This is at the expense of some detail on other parts of the scene, such as the hatch door. Since the view is moving down the corridor, all objects will be in motion, and we assume the viewer will be tracking the task-related objects more than the others. Rendering the entire frame to the same detail as the task objects in Figure 11 takes *7 times longer* than our optimized method. Although direct comparisons are difficult due to differences in the rendering aims, Yee et al. demonstrated a 4-10 times speedup in [33] and Myszkowski et al. showed a speedup of roughly 3.5 times in [19]. This shows that we are able to achieve similar speedups controlling only rendered sampling resolution. If we were to refine the global illumination calculation also, similar to Yee, we could achieve even greater gains.

There are only a few aspects of our framework that we must tailor to a ray-tracing approach. Initially, we compute a low quality, first order rendering from a quincunx sampling of the image plane, where one out of every 16 pixels is sampled. (This sampling pattern is visible in unrefined regions of Figure 13a.) To obtain the object and depth maps at unsampled locations, we cast rays to determine the first intersected object at these pixels. We then estimate our rendering error by finding the 5 nearest samples to each pixel position, and computing their standard deviation. This is a very crude approximation, but it suited our purposes well. In cases where the high-quality samples in the refinement pass have an interreflection calculation that the initial samples do not, we use the method described earlier for estimating the error due to a constant ambient term.

Following the IBR refinement described in the previous section, and provided we are not out of time, we then compute the error conspicuity map, sorting our pixels from most to least conspicuous. For pixels whose EC value are equal (usually 0), we order from highest to lowest error, then from fewest to most samples. Going down this list, we add one high-quality ray sample to each pixel, until we have sampled them all or run out of time. If we manage to get through the whole list, we recompute the error conspicuity map and re-sort. This time, we only add samples to the top 1/8th of our list before sorting again. We find we get smoother animations by sampling each pixel at least once before honing in on the regions we deem to be conspicuous. We could insist on sampling every pixel in our first order rendering, but this is sometimes impossible due to time constraints. Therefore, we incorporate it in our refinement phase, instead.

Figure 11: *A frame from our task-based animation.*

Prior to frame output, we perform a final filtering stage to interpolate unsampled pixels and add motion blur. Pixels that did not receive samples in the first order rendering or subsequent refinements must be given a value prior to output. We apply a Gaussian filter kernel whose support corresponds to our initial sample density to arrive at a weighted average of the 4 closest neighbors. Once we have a value at each pixel, we multiply the object motion map by a user-specified blur parameter, corresponding to the fraction of a frame time the virtual camera's shutter is open. The blur vector at each

pixel is then applied using an energy-preserving smear filter to arrive at the final output image. This technique is crude in the sense that it linearizes motion and does not discover obstructed geometry, but we have not found this to be objectionable in any of our tests. However, the lack of motion blur on shadows does show up as one of the few distracting artifacts in our implementation. This filtering operations take a small fraction of a CPU second per video resolution frame, and are inconsequential to the overall rendering time.

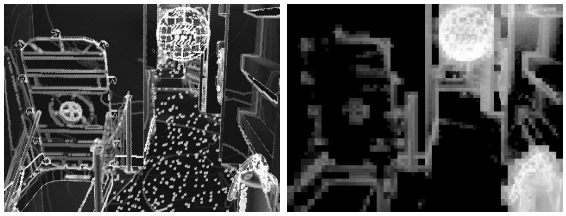


Figure 12: *a(left) Initial frame error, b(right) Initial error conspicuity.*

Figure 13: *a(left) Final frame samples, b(right) Standard rendering taking same time as Figure 11.*

Of our two minute rendering time for the frame shown in Figure 11, 1 second is spent updating the scene structures, 25 seconds is spent computing the 19,200 initial samples and the object map, 0.25 seconds is spent on IBR extrapolation, 0.9 seconds to compute the error map (times three evaluations), 1.25 seconds for the EC map, 0.4 seconds for filtering, and the remaining 90 seconds to compute about 110,000 high quality refinement samples. In this test, the Radiance rendering parameters were set so there was little computational difference between an initial sample and a high-quality refinement sample; we did not evaluate diffuse interreflections for either. Our method's combined overhead for a 640x480 frame is thus in the order of 14 seconds, 10 of which are spent computing the object map by ray casting. Intuitively and by our measurements, this overhead scales linearly with the number of pixels in a frame.

It is worth noting that IBR works particularly well in our progressive rendering framework, allowing us to achieve constant frame generation times over a wide range of motions. When motion is small, IBR extrapolation from the previous frame provides us with many low-error samples for our first refinement pass. When motion is great, and thus fewer extrapolated samples are available, the eye's inability to track objects and the associated blur means we do not need as many. This holds promise for realistic, real-time rendering using this approach with hardware support.

6. Conclusions and Future Work

As our experiments demonstrate, inattentional blindness may be exploited to accelerate rendering by reducing quality in regions that are unrelated to a given task. Extending this idea, we have designed a progressive animation framework that combines an indexed *task map* with a spatiotemporal *contrast sensitivity function* to determine which image areas need further refinement. Adding our knowledge of pixel uncertainty and movement between frames, we derive an *error conspicuity map*, which identifies noticeable artifacts in the presence of this task. We focus additional ray samples in these regions, and augment our results with IBR samples from the previous frame. We then apply the pixel movement map again to simulate motion blur in the final output.

Much work remains. Our current implementation performs poorly when subsequent refinement corrects for systematic errors in the initial estimate. This may result in noticeable discontinuities in the output, which makes it difficult to employ rendering methods that do not converge smoothly. Some intelligent blending or error dissipation is required if we wish to combine hardware rendering with ray-tracing, for example. At the level of the perceptual model, we would like to take advantage of masking effects to further reduce sampling in busy regions [7]. However, visual masking models have yet to be extended to the temporal domain, even though we know they are affected by movement. We would also like to find a sensible way to combine task-level information with low-level saliency. To apply them together, we need to know which visual processes dominate and under what conditions. Again, additional psychophysical research is required.

Human perception determines to a large extent what we do in computer graphics and indeed, why we do it. It seems fitting, therefore, that we should pay close attention to the attention graphics consumers pay to us. Exploiting task-level models of visual perception is one way to improve the viewing experience within a limited budget of time and resources.

Acknowledgements

This research is funded by the Engineering and Physical Sciences Research Council (Award No: 00301786). We are indebted to Tom Troscianko, Karol Myszkowski, Hector Yee and Scott Daly for all their help. We would also like thank everyone who attended the experiment; we would not have got any results if it were not for them.

References

1. K. Cater, A. Chalmers, and P. Ledda. Selective Quality Rendering by Exploiting Human Inattentional Blindness: Looking but not Seeing In Proceedings of *Symposium on Virtual Reality Software and Technology* 2002, ACM. pp. 17–24. 3, 6

2. H. Coolican. *Research Methods and Statistics in Psychology*. Hodder and Stoughton Educational, U.K., 1999. 5

3. S. Daly. The Visible Differences Predictor: an algorithm for the assessment of image fidelity. In A.B. Watson, editor, *Digital Image and Human Vision*, 1993, Cambridge, MA: MIT Press, pp. 179–206. 6

4. S. Daly. Engineering observations from spatiovelocity and spatiotemporal visual models. In IS and T/SPIE Conference on *Human Vision and Electronic Imaging III*, 1998, SPIE Proceedings Vol. 3299, pp. 180–193. 6, 8

5. S. Daly. Engineering observations from spatiovelocity and spatiotemporal visual models. Chapter 9 in *Vision Models and Applications to Image and Video Processing*, 2001, ed. C. J. van den Branden Lambrecht, Kluwer Academic Publishers. 6, 8

6. J.A. Ferwerda, S.N. Pattanaik, P.S. Shirley, and D.P. Greenberg. A Model of Visual Adaptation for Realistic Image Synthesis. In Proceedings of *SIGGRAPH 1996*, ACM Press / ACM SIGGRAPH, New York. H. Rushmeier, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, pp. 249–258. 2

7. J.A. Ferwerda, S.N. Pattanaik, P.S. Shirley, and D.P. Greenberg. A Model of Visual Masking for Computer Graphics. In Proceedings of *SIGGRAPH 1997*, ACM Press / ACM SIGGRAPH, New York. T. Whitted, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, pp. 143–152. 10

8. D.P. Greenberg, K.E. Torrance, P.S. Shirley, J. Arvo, J.A. Ferwerda, S.N. Pattanaik, A.E. Lafortune, B. Walter S-C. Foo and B. Trumbore. A Framework for Realistic Image Synthesis. In Proceedings of *SIGGRAPH 1997*,(special session), ACM Press / ACM SIGGRAPH, New York. T. Whitted, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, pp. 477–494. 2

9. L. Itti and C. Koch. A saliency-based search mechanism for overt and covert shifts of visual attention. In *Vision Research*, 2000, Vol. 40, No. 10-12, pp. 1489–1506. 3

10. W. James A saliency-based search mechanism for overt and covert shifts of visual attention. *Principles of Psychology*, New York: Holt. 2

11. M. Lee, R. Redner and S. Uselton. Statistically Optimized Sampling for Distributed Ray Tracing. In Proceedings of *SIGGRAPH 1985*, ACM Press / ACM SIGGRAPH, New York. Computer Graphics Proceedings, Annual Conference Series, ACM, Vol. 19, No. 3. 7

12. L.C. Loschky, G.W. McConkie, J. Yang and M.E. Miller. Perceptual Effects of a Gaze-Contingent Multi-Resolution Display Based on a Model of Visual Sensitivity. ARL Federated Laboratory Advanced Displays and Interactive Displays Consortium, *Advanced Displays and Interactive Displays Fifth Annual Symposium*, 2001, pp. 53–58. 2

13. D. Luebke and B. Hallen. Perceptually driven simplification for interactive rendering. In Proceedings of *12th Eurographics Workshop on Rendering*, 2001, pp. 221–223. 2

14. P.W.C Maciel and P. Shirley. Visual Navigation of Large Environments Using Textured Clusters. In Proceedings of *Symposium on Interactive 3D Graphics*, 1995, pp. 95–102. 2

15. A. Mack and I. Rock. Inattentional Blindness. In Proceedings of *Symposium on Interactive 3D Graphics*, Massachusetts Institute of Technology Press, 1998. 3

16. G. Marmitt and A.T. Duchowski Modeling Visual Attention in VR: Measuring the Accuracy of Predicted Scanpaths. *Eurographics 2002*, Short Presentations, pp. 217–226. 3

17. A. McNamara, A.G. Chalmers, T. Troscianko and I. Gilchrist. Comparing Real and Synthetic Scenes using Human Judgements of Lightness. In *12th Eurographics Workshop on Rendering 2000*, B Peroche and H Rushmeier (eds), pp. 207–219. 2

18. K. Myszkowski, T. Tawara, H. Akamine and H-P. Seidel. Perception-Guided Global Illumination Solution for Animation Rendering. In Proceedings of *SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, New York. E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, pp. 221–230. 2, 6

19. K. Myszkowski, R. Przemyslaw and T. Tawara. Perceptually-informed Accelerated Rendering of High Quality Walkthrough Sequences. In proceedings of the *Eurographics Workshop on Rendering 1999*, G.W. Larson and D. Lischinksi, Eds., pp. 13–26. 9

20. K. Myszkowski. The Visible Differences Predictor: Applications to global illumination problems. In proceedings of the *Eurographics Workshop on Rendering 1998*, G. Drettakis and N. Max, Eds., pp. 223–236. 6

21. C. O'Sullivan, J. Dingliana, G. Bradshaw, and A. McNamara. Eye-tracking for Interactive Computer Graphics. In proceedings of the *11th European Conference on Eye Movements (ECEM 11)*, 2001, Turku, Finland. 2

22. S. Parker, W. Martin, P-P. Sloan, P. Shirley, B. Smits, and C. Hansen. Interactive ray tracing. In Symposium on *Interactive 3D Graphics*, 1999, ACM, pp. 119–126. 7

23. S.N. Pattanaik, J.A. Ferwerda, M.D. and D.P. Greenberg. A Multiscale Model of Adaptation and Spatial Vision for Realistic Image Display. In Proceedings of *SIGGRAPH 1998*, ACM Press / ACM SIGGRAPH, New York. M. Cohen, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, pp. 287–298. 2

24. M. Ramasubramanian, S.N. Pattanaik, and D.P. Greenberg. A Perceptually Based Physical Error Metric for Realistic Image Synthesis. In Proceedings of *SIGGRAPH 1999*, ACM Press / ACM SIGGRAPH, New York. A. Rockwood, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, pp. 73–82. 2

25. M. Reddy. Perceptually Modulated Level of Detail for Virtual Environments. *Ph.D. Thesis (CST- 134-97)*, University of Edinburgh, 1997. 2

26. A. Treisman and J. Souther. Search asymmetry: A diagnostic for preattentive processing of separable features. In *Journal of Experimental Psychology: General*, 1985, 114 (3), pp. 285–310. 3

27. I. Wald, T. Kollig, C. Benthin, A. Keller and P. Slusallek. Interactive global illumination using fast ray tracing. In proceedings of the *13th Eurographics Workshop on Rendering 2002*, S.J. Gortler and K. Myszkowski, Eds., Springer-Verlag, pp. 9–19. 7

28. G. Ward. The RADIANCE Lighting Simulation and Rendering System. In Proceedings of *SIGGRAPH 1994*, ACM Press / ACM SIGGRAPH, New York, Computer Graphics Proceedings, Annual Conference Series, ACM, pp. 459–472. 4, 9

29. B. Watson, A. Friedman and A. McGaffey. An evaluation of Level of Detail Degradation in Head-Mounted Display Peripheries. In *Presence*, 6, 6, pp. 630–637. 2

30. B. Watson, A. Friedman and A. McGaffey. Measuring and Predicting Visual Fidelity. In Proceedings of *SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, New York, E. Fiume, Ed., Computer Graphics Proceedings, Annual Conference Series, ACM, pp. 213–220. 2

31. S. Yantis. Attentional capture in vision. In *Converging operations in the study of selective visual attention*,A. Kramer, M. Coles and G. Logan (eds), American Psychological Association, pp. 45–76. 3

32. A.L. Yarbus. Eye movements during perception of complex objects. In *Eye Movements and Vision*, L. A. Riggs, Ed., 1967, Plenum Press, New York, Chapter VII, pp. 171–196. 2, 3

33. H. Yee, S. Pattanaik and D.P. Greenberg. Spatiotemporal sensitivity and Visual Attention for efficient rendering of dynamic Environments. In *ACM Transactions on Computer Graphics*, 2001, Vol. 20, No. 1, pp. 39–65. 3, 6, 8, 9

Appendix E:

Overcoming Gamut and Dynamic Range Limitations in Digital Images

Reprinted from1998 Color Imaging Conference

Citation:

Larson, Gregory Ward, "Overcoming Gamut and Dynamic Range Limitations in Digital Images,", *Proceedings of the Sixth Color Imaging Conference*, November 1998.

Overcoming Gamut and Dynamic Range Limitations in Digital Images

Gregory Ward Larson
Silicon Graphics, Inc.
Mountain View, California

Abstract

The human eye can accommodate luminance in a single view over a range of about 10,000:1 and is capable of distinguishing about 10,000 colors at a given brightness. By comparison, typical CRT displays have a luminance range less than 100:1 and cover about half of the visible color gamut. Despite this difference, most digital image formats are geared to the capabilities of conventional displays, rather than the characteristics of human vision. In this paper, we propose two compact encodings suitable for the transfer, manipulation, and storage of full range color images. The first format is a replacement for conventional RGB images, and encodes color pixels as log luminance values and CIE (u',v') chromaticity coordinates. We have implemented and distributed this encoding as part of the standard TIFF I/O library on the net. The second format is proposed as an adjunct to conventional RGB data, and encodes out-of-gamut (and out-of-range) pixels in a supplemental image, suitable as a layer extension to the Flashpix standard. This data can then be recombined with the original RGB layer to obtain a high dynamic range image covering the full gamut of perceivable colors. Finally, we demonstrate the power and utility of full gamut imagery with example images and applications.

Introduction

What is the ultimate use of a digital image? How will it be presented? Will it be modified or adjusted? What kind of monitor will it be displayed on? What type of printer will it be sent to? How accurate do the colors need to be? More often than not, we don't know the answers to these questions a priori. More important, we don't know how these questions will be answered 10 or 100 years from now, when everything we know about digital imaging will have changed, but someone may still want to use our image. We should therefore endeavor to record image data that will be valuable under a broad range of foreseeable and postulated circumstances. Although this seems problematic, there is a simple solution. We may not be able to predict the technology, but we can predict that people will still be the primary consumers.

Most commonly used image standards based on current display technology, i.e., CRT monitors, rather than something less apt to change, i.e., human vision. All RGB standards are limited to a fraction of the visible gamut, since this gamut cannot be contained between any three *real* colors. Even Kodak's PhotoYCC encoding is ultimately geared for CRT display, and doesn't encompass the full gamut of colors or cover more than two orders of magnitude in brightness. The human eye is capable of perceiving at least four orders of magnitude in a daylit scene, and adapting more gradually over seven *additional* orders of magnitude, which means that most digital images encode only a small fraction of what a human observer can see.

In this sense, negative photography is superior to digital imaging in its ability to capture the dynamic range of a scene. A typical, consumer-grade color negative film has about 5-8 f-stops of *exposure latitude*, meaning that it can capture regions of a scene that are 2^5 to 2^8 times brighter than the camera's exposure setting (or dimmer if the image is overexposed), and still have enough range left over to reproduce each region*. Of course, most prints do not make use of the full range, unless a photographer picks up a wand or a cutout in the darkroom, but its presence permits re-exposure during the printing process to optimize the appearance of salient features, such as a person's face.

* To compute the latitude of a film or recording medium, take the log to the base 2 of the total usable dynamic range, from darkest unique value to brightest, and subtract 5 f-stops, which is the approximate range required for a usable image. There are about 3.3 f-stops per order of magnitude.

The question to ask is this: in 10 years or 100 years, what medium will be preferred for old photographs, a digital image, or a negative? Unless we change the way digital images are encoded, the answer in most cases will be a negative. Even considering aging and degradation (processes that can be partially compensated), a negative has both superior resolution and greater dynamic range than an RGB or YCC image. This needn't be the case.

In this paper, we present a compact pixel encoding using a log representation of luminance and a CIE (u',v') representation of color. We call this a *LogLuv* encoding. A log luminance representation means that at any exposure level, there will be equal brightness steps between values. This corresponds well with human visual response, whose contrast threshold is constant over a wide range of adaptation luminances (Weber's law). For color, the use of an approximately uniform perceptual space enables us to record the full gamut of visible colors using step sizes that are imperceptible to the eye. The combination of these two techniques permits us to make nearly optimal use of the bits available to record a given pixel, so that it may be reproduced over a broad range of viewing conditions. Also, since we are recording the full visible gamut and dynamic range, the output or display device can be *anything* and we won't be able to detect any errors or artifacts from our representation, simply because they will be reproduced below the visible threshold.

In this paper, we describe our LogLuv pixel encoding method, followed by a description of our extension to Sam Leffler's free TIFF library. We then put forth a proposal for extending the Flashpix format, and follow this with an example image to demonstrate the value of this encoding, ending with a brief conclusion.

Encoding Method

We have implemented two LogLuv pixel encodings, a 24-bit encoding and a 32-bit encoding. The 24-bit encoding breaks down into a 10-bit log luminance portion and a 14-bit, indexed uv coordinate mapping. Color indexing minimizes waste, allowing us to cover the irregular shape of the visible gamut in imperceptible steps. The 32-bit encoding uses 16 bits for luminance and 8 bits each for u' and v'. Compared to the 24-bit encoding, the 32-bit version provides greater dynamic range and precision at the cost of an extra byte per pixel. The exact interpretations of these two encodings are described below.

24-bit Encoding

In 24 bits, we can pack much more visible information than is commonly stored in three gamma-compressed 8-bit color primary values. By separating luminance and using a log encoding, we can use 10 bits to record nearly 5 orders of magnitude in 1.1% relative steps that will be imperceivable under most conditions. The remaining 14 bits will be used to store a color index corresponding to the smallest distinguishable patch size on a uv color chart. The bit allocation is shown graphically in Fig. 1.

Figure 1. 24-bit encoding. L_e is the encoded log luminance, and C_e is the encoded uv color index.

To compute the integer encoding L_e from real luminance, L, the we use the formula given in Eq. 1a. To compute real luminance from L_e, we use the inverse formula given in Eq. 1b.

$$L_e = \left\lfloor 64\left(\log_2 L + 12\right)\right\rfloor \qquad (1a)$$

$$L = \exp_2\left[\left(L_e + 0.5\right)/64 - 12\right] \qquad (1b)$$

In addition, an L_e value of 0 is taken to equal 0.0 exactly. An L_e value of 1 corresponds to a real luminance value of 0.000248 on an arbitrary scale, and the maximum L_e value of 1023 corresponds to a real value of 15.9 for a dynamic range of 65,000:1, or 4.8 orders of magnitude. It is difficult to compare this range to an 8-bit gamma-compressed encoding, because 1.1% accuracy is possible only near the very top of the 8-bit range. Allowing the luminance error to go as high as 5%, the dynamic range of an 8-bit encoding with a nominal gamma of 2.2 is 47:1, or 1.7 orders of magnitude. This leaves less than one f-stop of exposure latitude, compared to 11 f-stops for our 10-bit log encoding.

To capture full-gamut chrominance using only 14 bits, we cannot afford to waste codes on imaginary colors. We therefore divide our "perceptually uniform" (u',v') color space [8] into equal area regions using a scanline traversal over the visible gamut. This encoding concept is shown graphically in Fig. 2. The actual encoding has many more scanlines of course (163 to be exact), but the figure shows roughly how they are laid out. The minimum code value (0) is at the lower left, and codes are assigned left to right along each scanline until the maximum value (just less than 2^{14}) is assigned to the rightmost value on the top scanline.

$$u' = \frac{4x}{-2x+12y+3} \qquad (2a)$$

$$v' = \frac{9y}{-2x+12y+3} \qquad (2b)$$

To encode a given color, we start with the standard conversion from CIE (x,y) chromaticity to (u',v') shown in Eq. 2. We then look up the appropriate scanline for our *v'* value based on a uniform scanline height, and compute the position within the scanline using our uniform cell width. The index C_e is equal to the total of the scanlines below us plus the cells to the left in this

scanline. Cell width and height are both set to 0.0035 in our implementation, which corresponds to slightly less than the minimum perceptible step in this color space and uses up nearly all of the codes available in 14 bits.

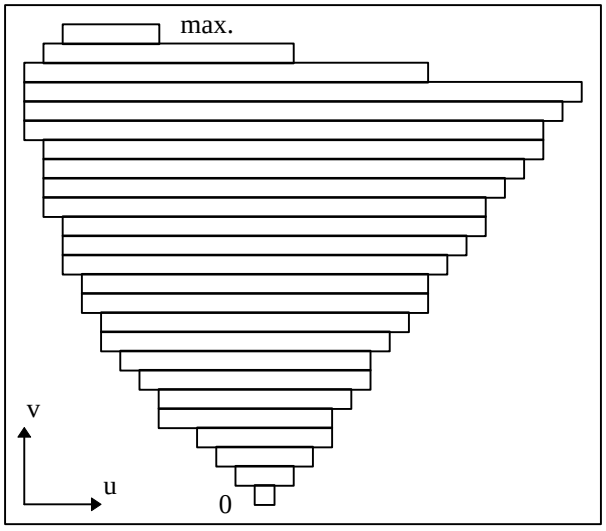


Figure 2. Scanline traversal of (u,v) coordinate space for 14-bit chromaticity encoding.

To get back the (x,y) chromaticity corresponding to a specific color index, we may either use a 16 Kentry look-up table, or apply a binary search to find the scanline containing corresponding to our C_e index. Once we have our original (u',v') coordinates back, we can apply the inverse conversion given in Eq. 3 to get the CIE chromaticity coordinates. (Note that this final computation may also be avoided using the same look-up table.)

$$x = \frac{9u'}{6u' - 16v' + 12} \qquad (3a)$$

$$y = \frac{4v'}{6u' - 16v' + 12} \qquad (3b)$$

32-bit Encoding

The 32-bit encoding is actually simpler, since we have 16 bits for (u',v'), which is more than enough that we can dispense with the complex color indexing scheme. The encoding of luminance is similar, with the addition of a sign bit so that negative luminances may also be encoded. In the remaining 15 bits, we can record over 38 orders of magnitude in 0.27% relative steps, covering the full range of perceivable world luminances in imperceptible steps. The bit breakdown is shown in Fig. 3.

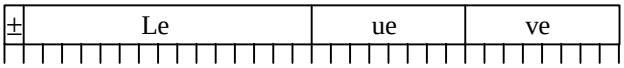


Figure 3. Bit allocation for 32-bit pixel encoding. MSB is a sign bit, and the next 15 bits are used for a log luminance encoding. The uv coordinates are separate 8-bit quantities.

The conversion to and from our log luminance encoding is given in Eq. 4. The maximum luminance using this encoding is 1.84×10^{19}, and the smallest magnitude is 5.44×10^{-20}. As in the 10-bit encoding, an L_e value of 0 is taken to be exactly 0.0. The sign bit is extracted before encoding and reapplied after the conversion back to real luminance.

$$L_e = \lfloor 256(\log_2 L + 64) \rfloor \qquad (4a)$$

$$L = \exp_2\left[(L_e + 0.5)/256 - 64\right] \qquad (4b)$$

As we mentioned, the encoding of chrominance is simplified because we have enough bits to record u_e and v_e separately. Since the gamut of u and v values is between 0 and 0.62, we chose a scale factor of 410 to go between our [0,255] integer range and real coordinates, as given in Eq. 5.

$$u_e = \lfloor 410u' \rfloor \qquad (5a)$$

$$v_e = \lfloor 410v' \rfloor \qquad (5b)$$

$$u' = (u_e + 0.5)/410 \qquad (5c)$$

$$v' = (v_e + 0.5)/410 \qquad (5d)$$

This encoding captures the full color gamut in 8 bits each for u_e and v_e. There will be some unused codes outside the visible gamut, but the tolerance this gives us of 0.0017 units in uv space is already well below the visible threshold. Conversions to and from CIE (x,y) chromaticities are the same as given earlier in Eqs. 2 and 3.

TIFF Input/Output Library

The LogLuv encodings described have been embedded as a new `SGILOG` compression type in Sam Leffler's popular TIFF I/O library. This library is freely distributed by anonymous ftp on ftp.sgi.com in the "/graphics/tiff/" directory.

When writing a high dynamic range (HDR) TIFF image, the LogLuv *codec* (compression/decompresson module) takes floating point CIE XYZ scanlines and writes out 24-bit or 32-bit compressed LogLuv-encoded values. When reading an HDR TIFF, the reverse conversion is performed to get back floating point XYZ values. (We also provide a simple conversion to 24-bit gamma-compressed RGB for the convenience of readers that do not know how to handle HDR pixels.)

An additional tag is provided for absolute luminance calibration, named `TIFFTAG_STONITS`. This is a single floating point value that may be used to convert Y values returned by the reader to absolute luminance in candelas per square meter. This tag may also be set by the application that writes out a HDR TIFF to permit calibrated scaling of values to a reasonable brightness range, where values of 1.0 will be displayed at the maximum output of the destination device. This scale factor may also necessary for calibration of the 24-bit format due to its more limited dynamic range.

Run-length Compression

Although at first it may appear that the 24-bit code is a more compact representation, the 32-bit encoding offers some advantages when it comes to applying nondestructive techniques to reduce storage requirements. By separating the bytes into four streams on each scanline, the 32-bit encoding can be efficiently compressed using an adaptive run-length encoding [3]. Since the top byte containing the sign bit and upper 7 log luminance bits changes very slowly, this byte-stream submits very well to run-length encoding. Likewise, the encoded u_e and v_e byte-streams compress well over areas of constant color. In contrast, the 24-bit encoding does not have a nice byte-stream breakup, so we do not attempt to run-length encode it, and the resulting files are quite often larger than the same data stored in the 32-bit format.

Grayscale Images

For maximum flexibility, a pure luminance mode is also provided by the codec, which stores and retrieves run-length encoded 16-bit log luminance values using the same scheme as applied in the 32-bit LogLuv encoding. There is no real space savings over a straight 32-bit encoding, since the u_e and v_e byte-streams compress to practically nothing for grayscale data, but this option provides an explicit way to specify floating point luminance images for TIFF readers that care.

Raw I/O

It is also possible to decode the raw 24-bit and 32-bit LogLuv data retrieved from an HDR TIFF directly, and this has some advantages for implementing fast tone mapping and display algorithms. In the case of the 24-bit format, one can simply multiply the output of a 1 Kentry L_e table and a 16 Kentry C_e table to get a tone-mapped and gamma-compressed RGB result. The 32-bit encoding requires a little more work, since its precomputed tables are 32 and 64 Kentries, but the same logic applies.

We have implemented this type of integer-math tone-mapping algorithm in an HDR image viewer, and it takes about a second to load and display a 512 by 512 picture on a 180 MHz processor.

Example TIFF Code and Images

Use of this encoding is demonstrated and sample images are provided on the following web site:

http://www.sgi.com/Technology/pixformat/

A converter has been written to and from the *Radiance* floating point picture format [6][7], and serves as an example of LogLuv codec usage. The web site itself also offers programming tips and example code segments.

Example TIFF images using the 32-bit LogLuv and 16-bit LogL encoding are provided on the web site. These images are either scanned from photographic negatives or rendered using *Radiance* and converted to the new TIFF format. Some images are rendered as 360° QuickTime VR panoramas suitable for experiments in HDR virtual reality.

Proposed Extension to Flashpix

The *Flashpix* format was originally developed by Kodak in collaboration with Hewlett-Packard, Live Picture and Microsoft. Its definition and maintenance has since been taken over by the Digital Imaging Group, a consortium of these and other companies. Flashpix is basically a multiresolution JPEG encoding, optimized for quick loading and editing at arbitrary pixel densities. It supports standard RGB as well as YCC color spaces with 8 bits/primary maximum resolution. For further information, see the DIG web site:

http://www.digitalimaging.org

Because Flashpix starts with 8-bit gamma-compressed color primaries, the dynamic range is limited to the same 1.7 orders of magnitude provided by other 24-bit RGB encodings. Furthermore, since JPEG encoding is applied, there will be additional losses and artifacts depending on the source image and the compression quality setting.

We cannot directly replace the JPEG-encoded Flashpix image with our own, alternate format, since this would violate standard compatibility as put forth by Kodak and enforced by the DIG. We must therefore provide any enhancement to the format as an optional extension, which results in a certain amount of redundancy in our case since the same pixels may be represented by two encodings. This is unavoidable.

For our extension, we need a second layer of "deeper" image data be provided for Flashpix users and applications that demand it. There are two ways we might go about this. The simplest method is to completely duplicate the source image in a 24 or 32-bit/pixel LogLuv encoding. On average, this will take roughly four to sixteen times as much space as the original JPEG encoding. A more sophisticated method is to replace only those pixels that are out of gamut or otherwise inadequate in the original encoding. We discuss this method below.

High Dynamic Range Extension Layer

Our proposed extension consists of a layer added to the standard Flashpix format. This layer contains two logical elements, a *presence map* of which pixels are included in the layer, and the list of corresponding 24-bit LogLuv pixels. The presence map may be represented by an entropy-encoded bitmap, which will typically take up 5% to 15% as much space as the JPEG layer. The extended pixels themselves will take between one half and four times as much space as the original JPEG layer, depending on the proportion of out-of-gamut pixels in the original image.

For an image that is entirely within gamut in the JPEG encoding, the presence map will compress to almost nothing, and there will be no LogLuv pixels, so the total overhead will be less than 1% of the original image. If the image is mostly out of the JPEG gamut, then the presence map might take half a bit per pixel, and the additional data will be the same size as a 24-bit RGB image. A typical high dynamic range image with 15% out-of-gamut pixels will take roughly the same space for the extension layer as the multiresolution JPEG layer, so the total image size will be about twice what it was originally. If the information is being accessed over the internet, the HDR layer may be loaded as an option, so it does not cost extra unless and until it is needed.

Example Results

Fig. 4a shows a scanned photograph as it might appear on a PhotoCD using a YCC encoding. Since YCC can capture up to "200% reflectance," we can apply a tone mapping operator to bring this extra dynamic range into our print, as shown in Fig. 5a. However, since many parts of the image were brighter than this 200% value, we still lose much of the sky and circumsolar region, and even the lighter asphalt in the foreground. In Fig. 4b, we see where 35% of the original pixels are outside the gamut of a YCC encoding.

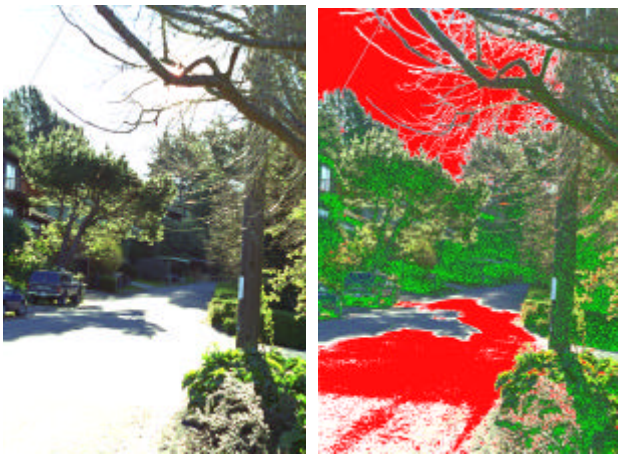


Figure 4. The left image (a) shows a PhotoYCC encoding of a color photograph tone-mapped with a linear operator. The right image (b) shows the out-of-gamut regions. Red areas are too bright or too dim, and green areas have inaccurate color.

Fig. 5b shows the same color negative scanned into our 32-bit/pixel high dynamic range TIFF format and tone mapped using a histogram compression technique [4]. Fig. 6c shows the same HDR TIFF remapped using the perceptual model of Pattanaik et al [5]. Figs. 6a and 6b show details of light and dark areas of the HDR image whose exposure has been adjusted to show the detail captured in the original negative. Without an HDR encoding, this information is either lost or unusable.

Figure 5. The left image(a) shows the YCC encoding after remapping with a high dynamic range tone operator [4]. Unfortunately, since YCC has so little dynamic range, most of the bright areas are lost. The right image (b) shows the same operator applied to a 32-bit HDR TIFF encoding, showing the full dynamic range of the negative.

Figure 6. The upper-left image (a) shows the circumsolar region reduced by 4 f-stops to show the image detail recorded on the negative. The lower-left image (b) shows house details boosted by 3 f-stops. The right image (c) shows our HDR TIFF mapped with the Pattanaik-Ferwerda tone operator [5].

Discussion

It is clear from looking at these images that current methods for tone-mapping HDR imagery, although better than a simple S-curve, are less than perfect. It would therefore be a mistake to store an image that has been irreversibly tone mapped in this fashion, as some scanner

software attempts to do. Storing an HDR image allows us to take full advantage of future improvements in tone mapping and display algorithms, at a nominal cost.

Besides professional photography, there are a number of application areas where HDR images are key. One is lighting simulation, where designers need to see an interior or exterior space as it would really appear, plus they need to evaluate things in terms absolute luminance and illuminance levels. Since an HDR image can store the real luminance in its full-gamut coverage, this information is readily accessible to the designer. Another application is image-based rendering, where a user is allowed to move about in a scene by warping captured or rendered images [1]. If these images have limited dynamic range, it is next to impossible to adapt the exposure based on the current view, and quality is compromised. Using HDR pixels, a natural view can be provided for any portion of the scene, no matter how bright or how dim. A fourth application area is digital archiving, where we are making a high-quality facsimile of a work of art for posterity. In this case, the pixels we record are precious, so we want to make sure they contain as much information as possible. At the same time, we have concerns about storage space and transmission costs, so keeping this data as compact as possible is important. Since our HDR format requires little more space than a standard 24-bit encoding to capture the full visible gamut, it is a clear winner for archiving applications.

Our essential argument is that we can make better use of the bits in each pixel by adopting a perceptual encoding of color and brightness. Although we don't know how a given image might be used or displayed in the future, we do know something about what a human can observe in a given scene. By faithfully recording this information, we ensure that our image will take full advantage of any future improvements in imaging technology, and our basic format will continue to find new uses.

Conclusion

We have presented a new method for encoding high dynamic range digital images using log luminance and uv chromaticity to capture the entire visible range of color and brightness. The proposed format requires little additional storage per pixel, while providing significant benefits to suppliers, caretakers and consumers of digital imagery.

Through the use of re-exposure and dynamic range compression, we have been able to show some of the benefits of HDR imagery. However, it is more difficult to illustrate the benefits of a larger color gamut without carefully comparing hard copy output of various multi-ink printers. Also, since we currently lack the ability to capture highly saturated scenes, our examples would have to be contrived from individual spectral measurements and hypothetical scenes. We therefore leave this as a future exercise.

Future work on the format itself should focus on the application of lossy compression methods (such as JPEG and fractal image encoding) for HDR images. Without such methods, the storage cost for a given resolution may hinder broad acceptance of this representation. Another extension we should look at is multispectral data, which is needed for remote imaging and some types of lighting simulation.

References

1. Paul Debevec, "Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-Based Graphics with Global Illumination and High Dynamic Range Photography," *Computer Graphics (Proceedings of ACM Siggraph 98)*.
2. Paul Debevec, Jitendra Malik, "Recovering High Dynamic Range Radiance Maps from Photographs," *Computer Graphics (Proceedings of ACM Siggraph 97)*.
3. Andrew Glassner, "Adaptive Run-Length Encoding," in *Graphics Gems II*, edited by James Arvo, Academic Press, (1991).
4. Greg Larson, Holly Rushmeier, Christine Piatko, "A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes," *IEEE Transactions on Visualization and Computer Graphics*, **3**, 4, (1997).
5. Sumant Pattanaik, James Ferwerda, Mark Fairchild, Don Greenberg, "A Multiscale Model of Adaptation and Spatial Vision for Realistic Image Display," *Computer Graphics (Proceedings of Siggraph 98)*.
6. Greg Ward, "The RADIANCE Lighting Simulation and Rendering System," *Computer Graphics (Proceedings of Siggraph 94)*.
7. Greg Ward, "Real Pixels," in *Graphics Gems II*, edited by James Arvo, Academic Press, (1991).
8. Gunter Wyszecki, W.S. Stiles, *Color Science: Concepts and Methods, Quantitative Data and Formulae*, Second Edition, Wiley, (1982).

Biography

Gregory Ward Larson is a member of the technical staff in the engineering division of SGI. He graduated with an AB in Physics in 1983 from the UC Berkeley, and earned his Master's in CS from San Francisco State in 1985. Greg has done work in physically-based rendering, surface reflectance measurements, and electronic data standards. He is the developer of the widely-used *Radiance* synthetic imaging system and the MGF exchange standard for scene data.

Greg may be reached by e-mail at *gregl@sgi.com*.

Appendix F:

A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes

Reprinted from 1997 LBL Technical Report

Citation:

Larson, Gregory Ward, Holly Rushmeier, Christine Piatko, "A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes," *IEEE Transactions on Visualization and Computer Graphics*, Vol. 3, No. 4, December 1997.

This paper appeared in IEEE Transactions on Visualization and
Computer Graphics, Vol. 3, No. 4, December 1997.

A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes

Gregory Ward Larson†
Building Technologies Program
Environmental Energy Technologies Division
Ernest Orlando Lawrence Berkeley National Laboratory
University of California
1 Cyclotron Road
Berkeley, California 94720

Holly Rushmeier
IBM T.J. Watson Research Center

Christine Piatko††
National Institute for Standards and Technology

January 15, 1997

This paper is available electronically at:
http://radsite.lbl.gov/radiance/papers

† Author's current address: Silicon Graphics, Inc., Mountain View, CA.
†† Author's current address: JHU/APL, Laurel, MD.

LBNL 39882
UC 400

A Visibility Matching Tone Reproduction Operator for High Dynamic Range Scenes

Gregory Ward Larson
Lawrence Berkeley National Laboratory

Holly Rushmeier
IBM T.J. Watson Research Center

Christine Piatko
National Institute for Standards and Technology

ABSTRACT

We present a tone reproduction operator that preserves visibility in high dynamic range scenes. Our method introduces a new histogram adjustment technique, based on the population of local adaptation luminances in a scene. To match subjective viewing experience, the method incorporates models for human contrast sensitivity, glare, spatial acuity and color sensitivity. We compare our results to previous work and present examples of our techniques applied to lighting simulation and electronic photography.

Keywords: Shading, Image Manipulation.

1 Introduction

The real world exhibits a wide range of luminance values. The human visual system is capable of perceiving scenes spanning 5 orders of magnitude, and adapting more gradually to over 9 orders of magnitude. Advanced techniques for producing synthetic images, such as radiosity and Monte Carlo ray tracing, compute the map of luminances that would reach an observer of a real scene. The media used to display these results -- either a video display or a print on paper -- cannot reproduce the computed luminances, or span more than a few orders of magnitude. However, the success of realistic image synthesis has shown that it is possible to produce images that convey the appearance of the simulated scene by mapping to a set of luminances that can be produced by the display medium. This is fundamentally possible because the human eye is sensitive to relative rather than absolute luminance values. However, a robust algorithm for converting real world luminances to display luminances has yet to be developed.

The conversion from real world to display luminances is known as *tone mapping*. Tone mapping ideas were originally developed for photography. In photography or video, chemistry or electronics, together with a human actively controlling the scene lighting and the camera, are used to map real world luminances into an acceptable image on a

display medium. In synthetic image generation, our goal is to avoid active control of lighting and camera settings. Furthermore, we hope to improve tone mapping techniques by having direct numerical control over display values, rather than depending on the physical limitations of chemistry or electronics.

Consider a typical scene that poses a problem for tone reproduction in both photography and computer graphics image synthesis systems. The scene is a room illuminated by a window that looks out on a sunlit landscape. A human observer inside the room can easily see individual objects in the room, as well as features in the outdoor landscape. This is because the eye adapts locally as we scan the different regions of the scene. If we attempt to photograph our view, the result is disappointing. Either the window is over-exposed and we can't see outside, or the interior of the room is under-exposed and looks black. Current computer graphics tone operators either produce the same disappointing result, or introduce artifacts that do not match our perception of the actual scene.

In this paper, we present a new tone reproduction operator that reliably maps real world luminances to display luminances, even in the problematic case just described. We consider the following two criteria most important for reliable tone mapping:

1. Visibility is reproduced. You can see an object in the real scene if and only if you can see it in the display. Objects are not obscured in under- or over-exposed regions, and features are not lost in the middle.
2. Viewing the image produces a subjective experience that corresponds with viewing the real scene. That is, the display should correlate well with memory of the actual scene. The overall impression of brightness, contrast, and color should be reproduced.

Previous tone mapping operators have generally met one of these criteria at the expense of the other. For example, some preserve the visibility of objects while changing the impression of contrast, while others preserve the overall impression of brightness at the expense of visibility.

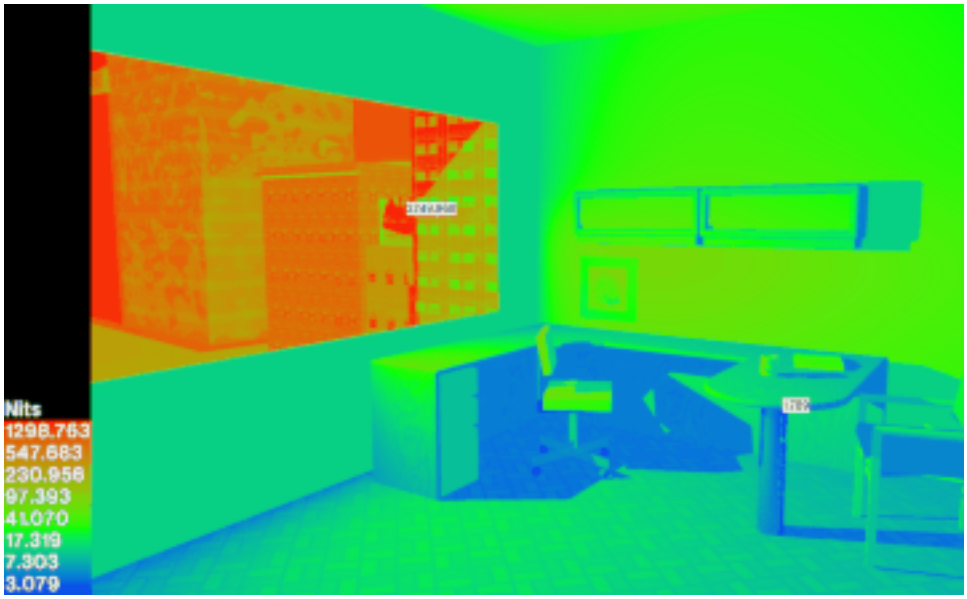


Figure 1. A false color image showing the world luminance values for a window office in candelas per meter squared (cd/m^2 or Nits).

The new tone mapping operator we present addresses our two criteria. We develop a method of modifying a luminance histogram, discovering clusters of adaptation levels and efficiently mapping them to display values to preserve local contrast visibility. We then use models for glare, color sensitivity and visual acuity to reproduce imperfections in human vision that further affect visibility and appearance.

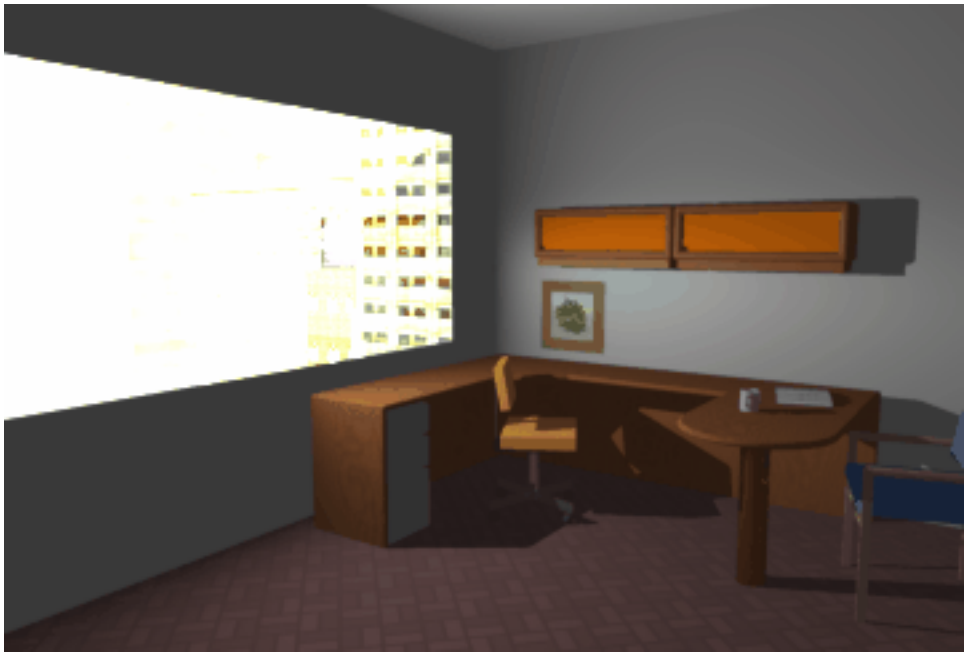


Figure 2. A linear mapping of the luminances in Figure 1 that over-exposes the view through the window.

Figure 3. A linear mapping of the luminances in Figure 1 that under-exposes the view of the interior.

Figure 4. The luminances in Figure 1 mapped to preserve the visibility of both indoor and outdoor features using the new tone mapping techniques described in this paper.

2 Previous Work

The high dynamic range problem was first encountered in computer graphics when physically accurate illumination methods were developed for image synthesis in the 1980's. (See Glassner [Glassner95] for a comprehensive review.) Previous methods for generating images were designed to automatically produce dimensionless values more or less evenly distributed in the range 0 to 1 or 0 to 255, which could be readily mapped to a display device. With the advent of radiosity and Monte Carlo path tracing techniques, we began to compute images in terms of real units with the real dynamic range of physical illumination. Figure 1 is a false color image showing the magnitude and distribution of luminance values in a typical indoor scene containing a window to a sunlit exterior. The goal of image synthesis is to produce results such as Figure 4, which match our impression of what such a scene looks like. Initially though, researchers found that a wide range of displayable images could be obtained from the same input luminances -- such as the unsatisfactory over- and under-exposed linear reproductions of the image in Figures 2 and 3.

Initial attempts to find a consistent mapping from computed to displayable luminances were ad hoc and developed for computational convenience. One approach is to use a function that collapses the high dynamic range of luminance into a small numerical range. By taking the cube root of luminance, for example, the range of values is reduced to something that is easily mapped to the display range. This approach generally preserves visibility of objects, our first criterion for a tone mapping operator. However, condensing the range of values in this way reduces fine detail visibility, and distorts impressions of brightness and contrast, so it does not fully match visibility or reproduce the subjective appearance required by our second criterion.

A more popular approach is to use an arbitrary linear scaling, either mapping the average of luminance in the real world to the average of the display, or the maximum non-light source luminance to the display maximum. For scenes with a dynamic range similar to the display device, this is successful. However, linear scaling methods do not maintain visibility in scenes with high dynamic range, since very bright and very dim values are clipped to fall within the display's limited dynamic range. Furthermore, scenes are mapped the same way regardless of the absolute values of luminance. A scene illuminated by a search light could be mapped to the same image as a scene illuminated by a flashlight, losing the overall impression of brightness and so losing the subjective correspondence between viewing the real and display-mapped scenes.

A tone mapping operator proposed by Tumblin and Rushmeier [Tumblin93] concentrated on the problem of preserving the viewer's overall impression of brightness. As the light level that the eye adapts to in a scene changes, the relationship between brightness (the subjective impression of the viewer) and luminance (the quantity of light in the visible range) also changes. Using a brightness function proposed by Stevens and Stevens [Stevens60], they developed an operator that would preserve the overall impression of brightness in the image, using one adaptation value for real scene, and another adaptation value for the displayed image. Because a single adaptation level is used for the scene, though, preservation of brightness in this case is at the expense of visibility. Areas that are very bright or dim are clipped, and objects in these areas are obscured.

Ward [Ward91] developed a simpler tone mapping method, designed to preserve feature visibility. In this method, a non-arbitrary linear scaling factor is found that preserves the impression of contrast (i.e., the visible changes in luminance) between the real and displayed image at a particular fixation point. While visibility is maintained at this adaptation point, the linear scaling factor still results in the clipping of very high and very low values, and correct visibility is not maintained throughout the image.

Chiu et al. [Chiu93] addressed this problem of global visibility loss by scaling luminance values based on a spatial average of luminances in pixel neighborhoods. Values in bright or dark areas would not be clipped, but scaled according to different values based on their spatial location. Since the human eye is less sensitive to variations at low spatial frequencies than high ones, a variable scaling that changes slowly relative to image features is not immediately visible. However, in a room with a bright source and dark corners, the method inevitably produces display luminance gradients that are the opposite of real world gradients. To make a dark region around a bright source, the transition from a dark area in the room to a bright area shows a decrease in brightness rather than an increase. This is illustrated in Figure 5 which shows a bright source with a dark halo around it. The dark halo that facilitates rendering the visibility of the bulb disrupts what should be a symmetric pattern of light cast by the bulb on the wall behind it. The reverse gradient fails to preserve the subjective correspondence between the real room and the displayed image.

Inspired by the work of Chiu et al., Schlick [Schlick95] developed an alternative method that could compute a spatially varying tone mapping. Schlick's work concentrated on improving computational efficiency and simplifying parameters, rather than improving the subjective correspondence of previous methods.

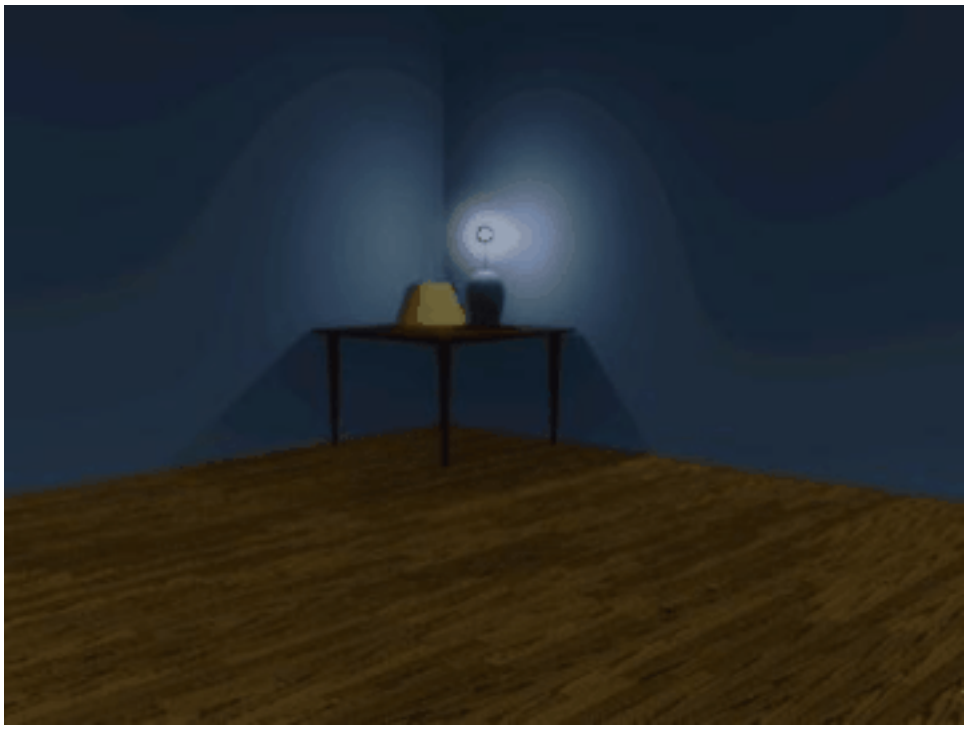


Figure 5. Dynamic range compression based on a spatially varying scale factor (from [Chiu93]).

Contrast, brightness and visibility are not the only perceptions that should be maintained by a tone mapping operator. Nakamae et al. [Nakamae90] and Spencer et al. [Spencer95] have proposed methods to simulate the effects of glare. These methods simulate the scattering in the eye by spreading the effects of a bright source in an image. Ferwerda et al. [Ferwerda96] proposed a method that accounts for changes in spatial acuity and color sensitivity as a function of light level. Our work is largely inspired by these papers, and we borrow heavily from Ferwerda et al. in particular. Besides maintaining visibility and the overall impression of brightness, the effects of glare, spatial acuity and color sensitivity must be included to fully meet our second criterion for producing a subjective correspondence between the viewer in the real scene and the viewer of the synthetic image.

A related set of methods for adjusting image contrast and visibility have been developed in the field of image processing for image enhancement (e.g., see Chapter 3 in [Green83]). Perhaps the best known image enhancement technique is histogram equalization. In histogram equalization, the grey levels in an image are redistributed more evenly to make better use of the range of the display device. Numerous improvements have been made to simple equalization by incorporating models of perception. Frei [Frei77] introduced histogram hyperbolization that attempts to redistribute perceived brightness, rather than screen grey levels. Frei approximated brightness using the logarithm of luminance. Subsequent researchers such as Mokrane [Mokrane92] have introduced methods that use more sophisticated models of perceived brightness and contrast.

The general idea of altering histogram distributions and using perceptual models to guide these alterations can be applied to tone mapping. However, there are two important

differences between techniques used in image enhancement and techniques for image synthesis and real-world tone mapping:

1. In image enhancement, the problem is to correct an image that has already been distorted by photography or video recording and collapsed into a limited dynamic range. In our problem, we begin with an undistorted array of real world luminances with a potentially high dynamic range.
2. In image enhancement, the goal is to take an imperfect image and *maximize* visibility or contrast. Maintaining subjective correspondence with the original view of the scene is irrelevant. In our problem, we want to maintain subjective correspondence. We want to *simulate* visibility and contrast, not maximize it. We want to produce visually accurate, not enhanced, images.

3 Overview of the New Method

In constructing a new method for tone mapping, we wish to keep the elements of previous methods that have been successful, and overcome the problems.

Consider again the room with a window looking out on a sunlit landscape. Like any high dynamic range scene, luminance levels occur in clusters, as shown in the histogram in Figure 6, rather than being uniformly distributed throughout the dynamic range. The failure of any method that uses a single adaptation level is that it maps a large range of sparsely populated real world luminance levels to a large range of display values. If the eye were sensitive to absolute values of luminance difference, this would be necessary. However, the eye is only sensitive to the fact that there are bright areas and dim areas. As long as the bright areas are displayed by higher luminances than the dim areas in the final image, the absolute value of the difference in luminance is not important. Exploiting this aspect of vision, we can close the gap between the display values for high and low luminance regions, and we have more display luminances to work with to render feature visibility.

Another failure of using a uniform adaptation level is that the eye rapidly adapts to the level of a relatively small angle in the visual field (i.e., about 1°) around the current fixation point [Moon&Spencer45]. When we look out the window, the eye adapts to the high exterior level, and when we look inside, it adapts to the low interior level. Chiu et al. [Chiu93] attempted to account for this using spatially varying scaling factors, but this method produces noticeable gradient reversals, as shown in Figure 5.

Rather than adjusting the adaptation level based on spatial location in the image, we will base our mapping on the population of the luminance adaptation levels in the image. To identify clusters of luminance levels and *initially* map them to display values, we will use the cumulative distribution of the luminance histogram. More specifically, we will start with a cumulative distribution based on a logarithmic approximation of brightness from luminance values.

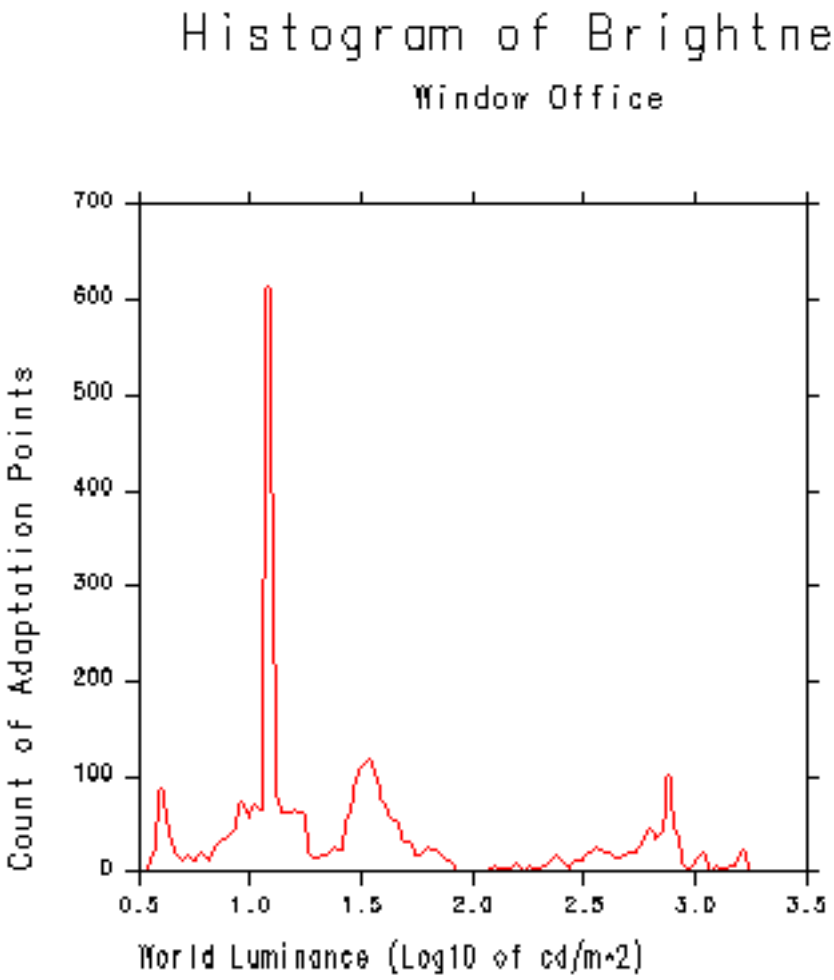


Figure 6. A histogram of adaptation values from Figure 1 (1° spot luminance averages).

First, we calculate the population of levels from a luminance image of the scene in which each pixel represents 1° in the visual field. We make a crude approximation of the brightness values (i.e., the subjective response) associated with these luminances by taking the logarithm of luminance. (Note that we will not display logarithmic values, we will merely use them to obtain a distribution.) We then build a histogram and cumulative distribution function from these values. Since the brightness values are integrated over a small solid angle, they are in some sense based on a spatial average, and the resulting mapping will be local to a particular adaptation level. Unlike Chiu's method however, the mapping for a particular luminance level will be consistent throughout the image, and will be order preserving. Specifically, an increase in real scene luminance level will always be represented by an increase in display luminance. The histogram and cumulative distribution function will allow us to close the gaps of sparsely populated luminance values and avoid the clipping problems of single adaptation level methods. By deriving a single, global tone mapping operator from locally averaged adaptation levels, we avoid the reverse gradient artifacts associated with a spatially varying multiplier.

We will use this histogram only as a starting point, and impose restrictions to preserve (rather than maximize) contrast based on models of human perception using our knowledge of the true luminance values in the scene. Simulations of glare and variations in spatial acuity and color sensitivity will be added into the model to maintain subjective correspondence and visibility. In the end, we obtain a mapping of real world to display luminance similar to the one shown in Figure 7.

For our target display, all mapped brightness values below 1 cd/m^2 (0 on the vertical axis) or above 100 (2 on the vertical axis) are lost because they are outside the displayable range. Here we see that the dynamic range between 1.75 and 2.5 has been compressed, yet we don't notice it in the displayed result (Figure 4). Compared to the two linear operators, our new tone mapping is the only one that can represent the entire scene without losing object or detail visibility.

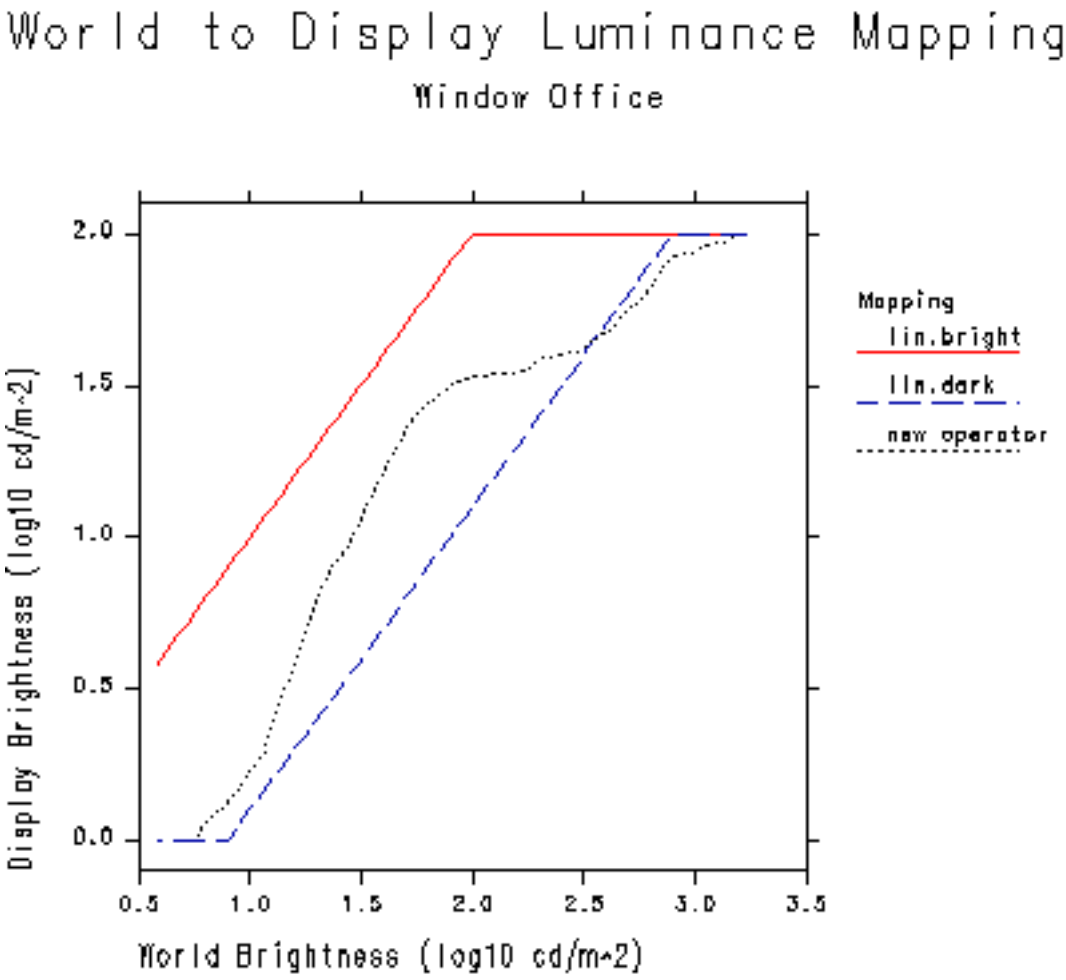


Figure 7. A plot comparing the global brightness mapping functions for Figures 1, 2, and 3, respectively.

In the following section, we illustrate this technique for histogram adjustment based on contrast sensitivity. After this, we describe models of glare, color sensitivity and visual

acuity that complete our simulation of the measurable and subjective responses of human vision. Finally, we complete the methods presentation with a summary describing how all the pieces fit together.

4 Histogram Adjustment

In this section, we present a detailed description of our basic tone mapping operator. We begin with the introduction of symbols and definitions, and a description of the histogram calculation. We then describe a naive equalization step that partially accomplishes our goals, but results in undesirable artifacts. This method is then refined with a linear contrast ceiling, which is further refined using human contrast sensitivity data.

4.1 Symbols and Definitions

L_w	= world luminance (in candelas/meter^2)
B_w	= world brightness, $\log(L_w)$
L_{wmin}	= minimum world luminance for scene
L_{wmax}	= maximum world luminance for scene
L_d	= display luminance (in candelas/meter^2)
L_{dmin}	= minimum display luminance (black level)
L_{dmax}	= maximum display luminance (white level)
B_{de}	= computed display brightness, $\log(L_d)$ [Equation (4)]
N	= the number of histogram bins
T	= the total number of adaptation samples
$f(b_i)$	= frequency count for the histogram bin at b_i
Δb	= the bin step size in $\log(cd/m^2)$
P(b)	= the cumulative distribution function [Equation (2)]
log(x)	= natural logarithm of x
log10(x)	= decimal logarithm of x

4.2 Histogram Calculation

Since we are interested in optimizing the mapping between world adaptation and display adaptation, we start with a histogram of world adaptation luminances. The eye adapts for the best view in the fovea, so we compute each luminance over a 1° diameter solid angle corresponding to a potential foveal fixation point in the scene. We use a logarithmic scale for the histogram to best capture luminance population and subjective response over a wide dynamic range. This requires setting a minimum value as well as a maximum, since the logarithm of zero is -∞. For the minimum value, we use either the minimum 1° spot average, or 10^{-4} cd/m^2 (the lower threshold of human vision), whichever is larger. The maximum value is just the maximum spot average.

We start by filtering our original floating-point image down to a resolution that roughly corresponds to 1° square pixels. If we are using a linear perspective projection, the pixels on the perimeter will have slightly smaller diameter than the center pixels, but they will still be within the correct range. The following formula yields the correct resolution for

1° diameter pixels near the center of a linear perspective image:

$$S = 2 \tan(\theta/2) / 0.01745 \quad (1)$$

where:

S = width or height in pixels

θ = horizontal or vertical full view angle

0.01745 = number of radians in 1°

For example, the view width and height for Figure 4 are 63° and 45° respectively, which yield a sample image resolution of 70 by 47 pixels. Near the center, the pixels will be 1° square exactly, but near the corners, they will be closer to 0.85° for this wide-angle view. The filter kernel used for averaging will have little influence on our result, so long as every pixel in the original image is weighted similarly. We employ a simple box filter.

From our reduced image, we compute the logarithms of the floating-point luminance values. Here, we assume there is some method for obtaining the absolute luminances at each spot sample. If the image is uncalibrated, then the corrections for human vision will not work, although the method may still be used to optimize the visible dynamic range. (We will return to this in the summary.)

The histogram is taken between the minimum and maximum values mentioned earlier in equal-sized bins on a log(luminance) scale. The algorithm is not sensitive to the number of bins, so long as there are enough to obtain adequate resolution. We use 100 bins in all of our examples. The resulting histogram for Figure 1 is shown in Figure 6.

4.2.1 Cumulative Distribution

The cumulative frequency distribution is defined as:

$$P(b) = \frac{\sum_{b_i < b} f(b_i)}{T} \quad (2)$$

where:

$T = \sum_{b_i} f(b_i)$ (i.e., the total number of samples)

Later on, we will also need the derivative of this function. Since the cumulative distribution is a numerical integration of the histogram, the derivative is simply the histogram with an appropriate normalization factor. In our method, we approximate a continuous distribution and derivative by interpolating adjacent values linearly. The derivative of our function is:

$$\frac{dP(b)}{db} = \frac{f(b)}{T \Delta b} \quad (3)$$

where:

$\Delta b = \frac{[\log(L_{wmax}) - \log(L_{wmin})]}{N}$ (i.e., the size of each bin)

Figure 8. Rendering of a bathroom model mapped with a linear operator.

4.3 Naive Histogram Equalization

If we wanted all the brightness values to have equal probability in our final displayed image, we could now perform a straightforward histogram equalization. Although this is not our goal, it is a good starting point for us. Based on the cumulative frequency distribution just described, the equalization formula can be stated in terms of brightness as follows:

$$B_{de} = \log(L_{dmin}) + \left[\log(L_{dmax}) - \log(L_{dmin})\right] \cdot P(B_w) \qquad (4)$$

The problem with naive histogram equalization is that it not only compresses dynamic range (contrast) in regions where there are few samples, it also *expands* contrast in highly populated regions of the histogram. The net effect is to exaggerate contrast in large areas of the displayed image. Take as an example the scene shown in Figure 8. Although we cannot see the region surrounding the lamps due to the clamped linear tone mapping operator, the image appears to us as more or less normal. Applying the naive histogram equalization, Figure 9 is produced. The tiles in the shower now have a mottled appearance. Because this region of world luminance values is so well represented, naive

histogram equalization spreads it out over a relatively larger portion of the display's dynamic range, generating superlinear contrast in this region.

Figure 9. Naive histogram equalization allows us to see the area around the light sources but contrast is exaggerated in other areas such as the shower tiles.

4.4 Histogram Adjustment with a Linear Ceiling

If the contrast being produced is too high, then what is an appropriate contrast for representing image features? The crude answer is that the contrast in any given region should not exceed that produced by a linear tone mapping operator, since linear operators produce satisfactory results for scenes with limited dynamic range. We will take this simple approach first, and later refine our answer based on human contrast sensitivity.

A linear ceiling on the contrast produced by our tone mapping operator can be written thus:

$$\frac{dL_d}{dL_w} \le \frac{L_d}{L_w} \qquad (5a)$$

That is, the derivative of the display luminance with respect to the world luminance must not exceed the display luminance divided by the world luminance. Since we have an expression for the display luminance as a function of world luminance for our naive histogram equalization, we can differentiate the exponentiation of Equation (4) using the chain rule and the derivative from Equation (3) to get the following inequality:

$$\exp(B_{de}) \cdot \frac{f(B_w)}{T\Delta b} \cdot \frac{\log(L_{dmax}) - \log(L_{dmin})}{L_w} \leq \frac{L_d}{L_w} \qquad (5b)$$

Since L_d is equal to $\exp(B_{de})$, this reduces to a constant ceiling on f(b):

$$f(b) \leq \frac{T\Delta b}{\log(L_{dmax}) - \log(L_{dmin})} \qquad (5c)$$

In other words, so long as we make sure no frequency count exceeds this ceiling, our resulting histogram will not exaggerate contrast. How can we create this modified histogram? We considered both truncating larger counts to this ceiling and redistributing counts that exceeded the ceiling to other histogram bins. After trying both methods, we found truncation to be the simplest and most reliable approach. The only complication introduced by this technique is that once frequency counts are truncated, T changes, which changes the ceiling. We therefore apply iteration until a tolerance criterion is met, which says that fewer than 2.5% of the original samples exceed the ceiling.[1] Our pseudocode for `histogram_ceiling` is given below:

```
boolean function histogram_ceiling()
tolerance := 2.5% of histogram total
repeat {
     trimmings := 0
     compute the new histogram total T
     if T < tolerance then
          return FALSE
     foreach histogram bin i do
          compute the ceiling
          if f(b_i) > ceiling then {
               trimmings += f(b_i) - ceiling
               f(b_i) := ceiling
          }
} until trimmings <= tolerance
return TRUE
```

This iteration will fail to converge (and the function will return FALSE) if and only if the dynamic range of the output device is already ample for representing the sample luminances in the original histogram. This is evident from Equation (5c), since Δb is the world brightness range over the number of bins:

$$f(b_i) \leq \frac{T}{N} \cdot \frac{\left[\log(L_{wmax}) - \log(L_{wmin})\right]}{\left[\log(L_{dmax}) - \log(L_{dmin})\right]} \qquad (5d)$$

[1] The tolerance of 2.5% was chosen as an arbitrary small value, and it seems to make little difference either to the convergence time or the results.

If the ratio of the world brightness range over the display brightness range is less than one (i.e., our world range fits in our display range), then our frequency ceiling is less than the total count over the number of bins. Such a condition will never be met, since a uniform distribution of samples would still be over the ceiling in every bin. It is easiest to detect this case at the outset by checking the respective brightness ranges, and applying a simple linear operator if compression is unnecessary.

We call this method *histogram adjustment* rather than histogram equalization because the final brightness distribution is *not equalized*. The net result is a mapping of the scene's high dynamic range to the display's smaller dynamic range that minimizes visible contrast distortions, by compressing under-represented regions without expanding over-represented ones.

Figure 10 shows the results of our histogram adjustment algorithm with a linear ceiling. The problems of exaggerated contrast are resolved, and we can still see the full range of brightness. A comparison of these tone mapping operators is shown in Figure 11. The naive operator is superlinear over a large range, seen as a very steep slope near world luminances around $10^{0.8}$.

Figure 10. Histogram adjustment with a linear ceiling on contrast preserves both lamp visibility and tile appearance.

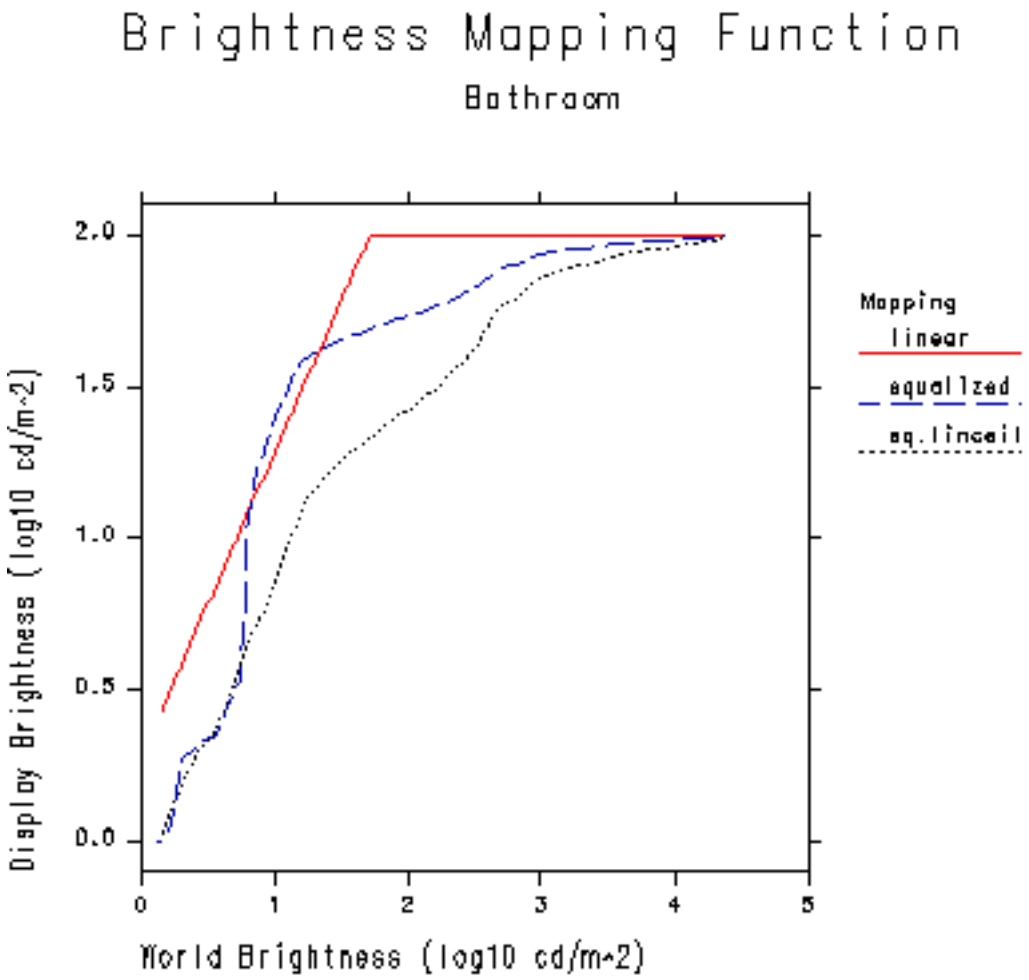


Figure 11. A comparison of naive histogram equalization with histogram adjustment using a linear contrast ceiling.

The method we have just presented is itself quite useful. We have managed to overcome limitations in the dynamic range of typical displays without introducing objectionable contrast compression artifacts in our image. In situations where we want to get a good, natural-looking image without regard to how well a human observer would be able to see in a real environment, this may be an optimal solution. However, if we are concerned with reproducing both visibility and subjective experience in our displayed image, then we must take it a step further and consider the *limitations* of human vision.

4.5 Histogram Adjustment Based on Human Contrast Sensitivity

Although the human eye is capable of adapting over a very wide dynamic range (on the order of 10^9), we do not see equally well at all light levels. As the light grows dim, we have more and more trouble detecting contrast. The relationship between adaptation luminance and the minimum detectable luminance change is well studied [CIE81]. For consistency with earlier work, we use the same detection threshold function used by Ferwerda et al. [Ferwerda96]. This function covers sensitivity from the lower limit of

human vision to daylight levels, and accounts for both rod and cone response functions. The piecewise fit is reprinted in Table 1.

log10 of just noticeable difference	applicable luminance range
-2.86	$\log10(L_a) < -3.94$
$(0.405 \log10(L_a) + 1.6)^{2.18} - 2.86$	$-3.94 \le \log10(L_a) < -1.44$
$\log10(L_a) - 0.395$	$-1.44 \le \log10(L_a) < -0.0184$
$(0.249 \log(L_a) + 0.65)^{2.7} - 0.72$	$-0.0184 \le \log10(L_a) < 1.9$
$\log10(L_a) - 1.255$	$\log10(L_a) \ge 1.9$

Table 1. Piecewise approximation for $\Delta L_t(L_a)$.

We name this combined sensitivity function:

$$\Delta L_t(L_a) = \text{"just noticeable difference" for adaptation level } L_a \qquad (6)$$

Ferwerda et al. did not combine the rod and cone sensitivity functions in this manner, since they used the two ranges for different tone mapping operators. Since we are using this function to control the maximum reproduced contrast, we combine them at their crossover point of $10^{-0.0184}$ cd/m^2.

To guarantee that our display representation does not exhibit contrast that is more noticeable than it would be in the actual scene, we constrain the slope of our operator to the ratio of the two adaptation thresholds for the display and world, respectively. This is the same technique introduced by Ward [Ward91] and used by Ferwerda et al. [Ferwerda96] to derive a global scale factor. In our case, however, the overall tone mapping operator will not be linear, since the constraint will be met at all potential adaptation levels, not just a single selected one. The new ceiling can be written as:

$$\frac{dL_d}{dL_w} \le \frac{\Delta L_t(L_d)}{\Delta L_t(L_w)} \qquad (7a)$$

As before, we compute the derivative of the histogram equalization function (Equation (4)) to get:

$$\exp(B_{de}) \cdot \frac{f(B_w)}{T\Delta b} \cdot \frac{\log(L_{dmax}) - \log(L_{dmin})}{L_w} \le \frac{\Delta L_t(L_d)}{\Delta L_t(L_w)} \qquad (7b)$$

However, this time the constraint does not reduce to a constant ceiling for f(b). We notice that since L_d equals $\exp(B_{de})$ and B_{de} is a function of L_w from Equation (4), our

ceiling is completely defined for a given P(b) and world luminance, L_w:

$$f(B_w) \le \frac{\Delta L_t(L_d)}{\Delta L_t(L_w)} \cdot \frac{T \Delta b L_w}{[\log(L_{dmax}) - \log(L_{dmin})] L_d} \qquad (7c)$$

where:

L_d = $\exp(B_{de})$, B_{de} given in Equation (4)

Once again, we must iterate to a solution, since truncating bin counts will affect T and P(b). We reuse the `histogram_ceiling` procedure given earlier, replacing the linear contrast ceiling computation with the above formula.

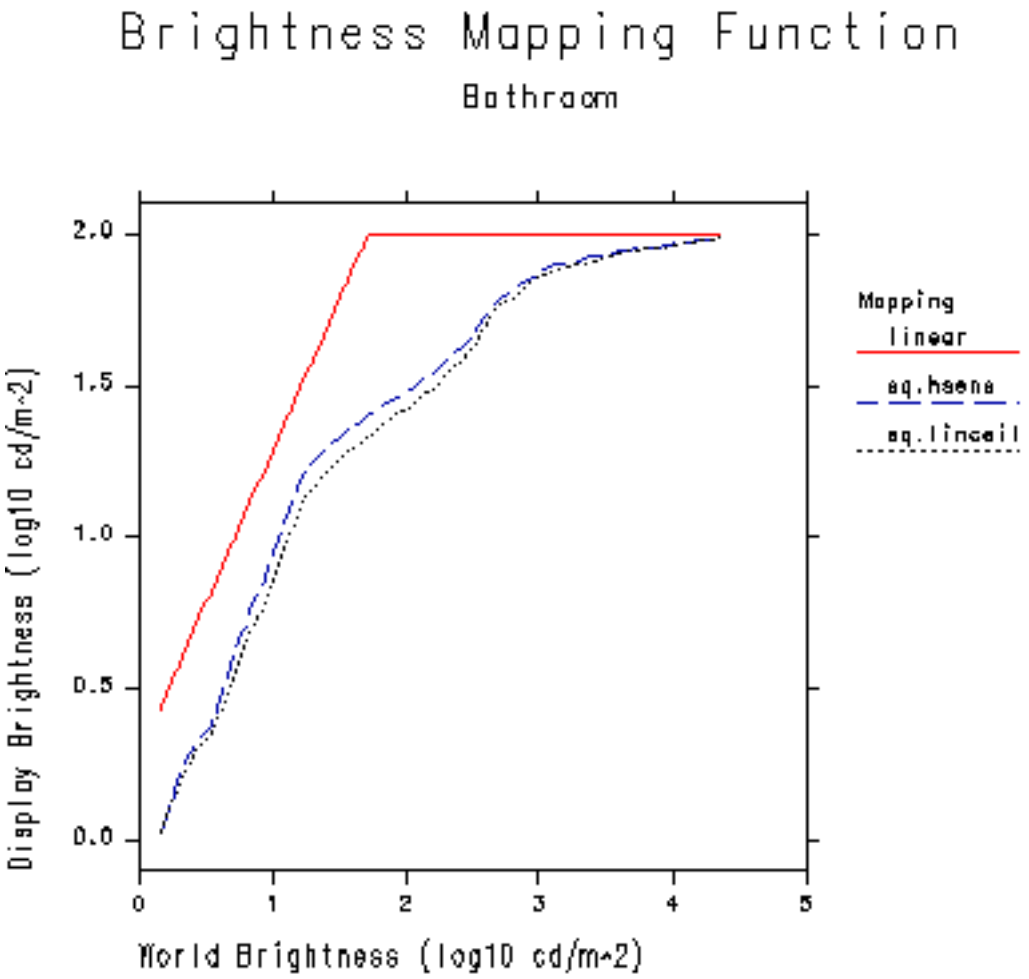


Figure 12. Our tone mapping operator based on human contrast sensitivity compared to the histogram adjustment with linear ceiling used in Figure 10. Human contrast sensitivity makes little difference at these light levels.

Figure 12 shows the same curves for the linear tone mapping and histogram adjustment with linear clamping shown before in Figure 11, but with the curve for naive histogram

equalization replaced by our human visibility matching algorithm. We see the two histogram adjustment curves are very close. In fact, we would have some difficulty differentiating images mapped with our latest method and histogram adjustment with a linear ceiling. This is because the scene we have chosen has most of its luminance levels in the same range as our display luminances. Therefore, the ratio between display and world luminance detection thresholds is close to the ratio of the display and world adaptation luminances. This is known as Weber's law [Riggs71], and it holds true over a wide range of luminances where the eye sees equally well This correspondence makes the right-hand side of Equations (5b) and (7b) equivalent, and so we should expect the same result as a linear ceiling.

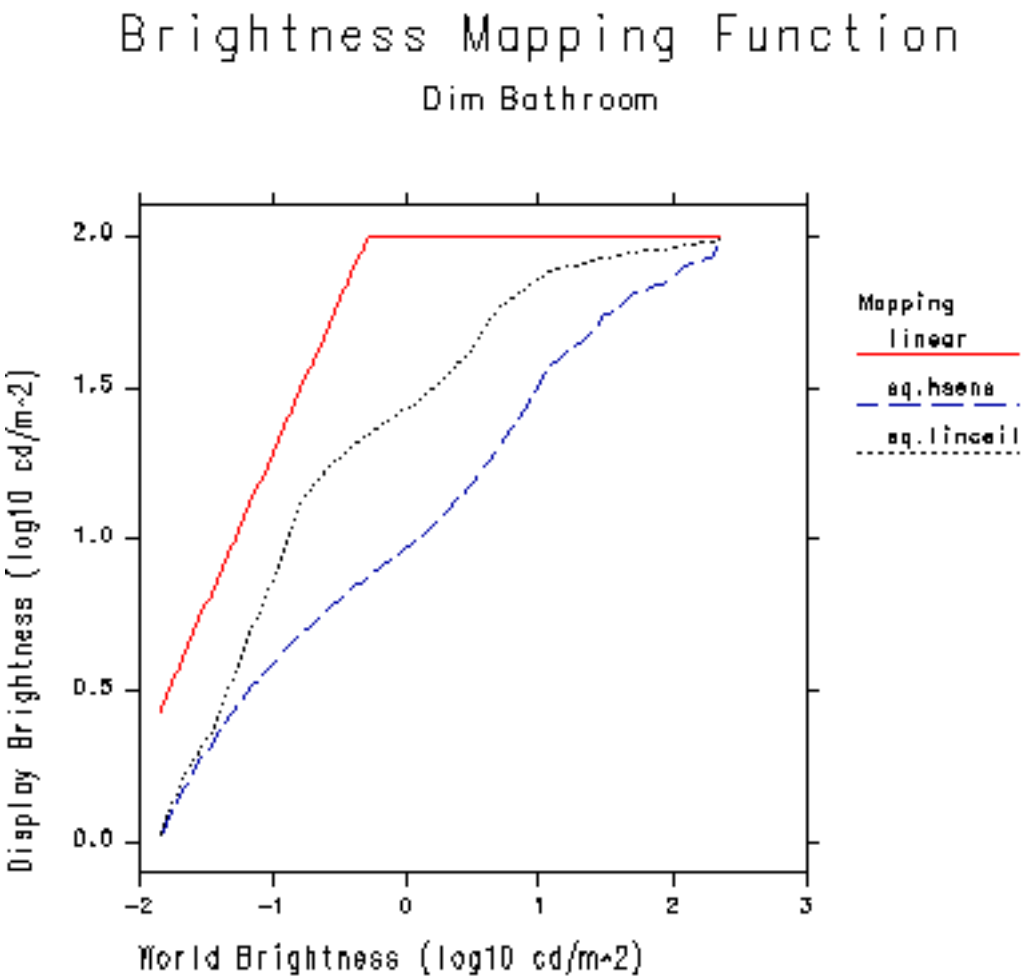


Figure 13. The brightness map for the bathroom scene with lights dimmed to 1/100th of their original intensity, where human contrast sensitivity makes a difference.

To see a contrast sensitivity effect, our world adaptation would have to be very different from our display adaptation. If we reduce the light level in the bathroom by a factor of 100, our ability to detect contrast is diminished. This shows up in a relatively larger detection threshold in the denominator of Equation (7c), which reduces the ceiling for the

frequency counts. The change in the tone mapping operator is plotted in Figure 13 and the resulting image is shown in Figure 14.

Figure 13 shows that the linear mapping is unaffected, since we just raise the scale factor to achieve an average exposure. Likewise, the histogram adjustment with a linear ceiling maps the image to the same display range, since its goal is to reproduce linear contrast. However, the ceiling based on human threshold visibility limits contrast over much of the scene, and the resulting image is darker and less visible everywhere except the top of the range, which is actually shown with higher contrast since we now have display range to spare.

Figure 14 is darker and the display contrast is reduced compared to Figure 10. Because the tone mapping is based on *local adaptation* rather than a single global or spot average, threshold visibility is reproduced *everywhere* in the image, not just around a certain set of values. This criterion is met within the limitations of the display's dynamic range.

Figure 14. The dimmed bathroom scene mapped with the function shown in Figure 13.

5 Human Visual Limitations

We have seen how histogram adjustment matches display contrast visibility to world visibility, but we have ignored three important limitations in human vision: glare, color sensitivity and visual acuity. Glare is caused by bright sources in the visual periphery, which scatter light in the lens of the eye, obscuring foveal vision. Color sensitivity is lost in dark environments, as the light-sensitive rods take over for the color-sensitive cone system. Visual acuity is also impaired in dark environments, due to the complete loss of cone response and the quantum nature of light sensation.

In our treatment, we will rely heavily on previous work performed by Moon and Spencer [Moon&Spencer45] and Ferwerda et al. [Ferwerda96], applying it in the context of a locally adapted visibility-matching model.

5.1 Veiling Luminance

Bright *glare sources* in the periphery reduce contrast visibility because light scattered in the lens and aqueous humor obscures the fovea; this effect is less noticeable when looking directly at a source, since the eye adapts to the high light level. The influence of glare sources on contrast sensitivity is well studied and documented. We apply the work of Holladay [Holladay26] and Moon and Spencer [Moon&Spencer45], which relates the effective adaptation luminance to the foveal average and glare source position and illuminance.

In our presentation, we will first compute a low resolution veil image from our foveal sample values. We will then interpolate this veil image to add glare effects to the original rendering. Finally, we will apply this veil as a correction to the adaptation luminances used for our contrast, color sensitivity and acuity models.

Moon and Spencer base their formula for adaptation luminance on the effect of individual glare sources measured by Holladay, which they converted to an integral over the entire visual periphery. The resulting glare formula gives the effective adaptation luminance at a particular fixation for an arbitrary visual field:

$$L_a = 0.913 L_f + \frac{K}{\pi} \iint_{\theta > \theta_f} \frac{L(\theta,\phi)}{\theta^2} \cos(\theta)\sin(\theta)\, d\theta\, d\phi \qquad (8)$$

where:

L_a	= corrected adaptation luminance (in cd/m^2)
L_f	= the average foveal luminance (in cd/m^2)
$L(\theta,\phi)$	= the luminance in the direction (θ,ϕ)
θ_f	= foveal half angle, approx. 0.00873 radians (0.5°)
K	= constant measured by Holladay, 0.0096

The constant 0.913 in this formula is the remainder from integrating the second part assuming one luminance everywhere. In other words, the periphery contributes less than 9% to the average adaptation luminance, due to the small value Holladay determined for K. If there are no bright sources, this influence can be safely neglected. However, bright sources will significantly affect the adaptation luminance, and should be considered in our model of contrast sensitivity.

To compute the veiling luminance corresponding to a given foveal sample (i.e., fixation point), we can convert the integral in Equation (8) to an average over peripheral sample values:

$$L_{vi} = 0.087 \cdot \frac{\sum_{j \neq i} \frac{L_j \cos(\theta_{i,j})}{\theta_{i,j}^2}}{\sum_{j \neq i} \frac{\cos(\theta_{i,j})}{\theta_{i,j}^2}} \qquad (9)$$

where:

L_{vi} = veiling luminance for fixation point i
L_j = foveal luminance for fixation point j
$\theta_{i,j}$ = angle between sample i and j (in radians)

Since we must compute this sum over all foveal samples j for each fixation point i, the calculation can be very time consuming. We therefore reduce our costs by approximating the weight expression as:

$$\frac{\cos\theta}{\theta^2} \approx \frac{\cos\theta}{2 - 2\cos\theta} \qquad (10)$$

Since the angles between our samples are most conveniently available as vector dot products, which is the cosine, the above weight computation is quite fast. However, for large images (in terms of angular size), the L_{vi} calculation is still the most computationally expensive step in our method due to the double iteration over i and j.

To simulate the effect of glare on visibility, we simply add the computed veil map to our original image. Just as it occurs in the eye, the veiling luminance will obscure the visible contrast on the display by adding to both the background and the foreground luminance.[2] This was the original suggestion made by Holladay, who noted that the effect glare has on luminance threshold visibility is equivalent to what one would get by adding the veiling luminance function to the original image [Holladay26]. This is quite straightforward once we have computed our foveal-sampled veiling image given in Equation (9). At each image pixel, we perform the following calculation:

$$L_{pvk} = 0.913 L_{pk} + L_v(k) \qquad (11)$$

where:

L_{pvk} = veiled pixel at image position k
L_{pk} = original pixel at image position k
$L_v(k)$ = interpolated veiling luminance at k

The $L_v(k)$ function is a simple bilinear interpolation on the four closest samples in our veil image computed in Equation (9). The final image will be lighter around glare sources, and just slightly darker *on* glare sources (since the veil is effectively being spread away from bright points). Although we have shown this as a luminance calculation, we retain color information so that our veil has the same color cast as the responsible glare source(s).

[2]The contrast is defined as the ratio of the foreground minus the background over the background, so adding luminance to both foreground and background reduces contrast.

Figure 15 shows our original, fully lit bathroom scene again, this time adding in the computed veiling luminance. Contrast visibility is reduced around the lamps, but the veil falls off rapidly (as $1/\theta^2$) over other parts of the image. If we were to measure the luminance detection threshold at any given image point, the result should correspond closely to the threshold we would measure at that point in the actual scene.

Figure 15. Our tone reproduction operator for the original bathroom scene with veiling luminance added.

Since glare sources scatter light onto the fovea, they also affect the local adaptation level, and we should consider this in the other parts of our calculation. We therefore apply the computed veiling luminances to our foveal samples as a correction *before* the histogram generation and adjustment described in Section 4. We deferred the introduction of this correction factor to simplify our presentation, since in most cases it only weakly affects the brightness mapping function.

The correction to local adaptation is the same as Equation (11), but without interpolation, since our veil samples correspond one-to-one:

$$L_{ai} = 0.913 L_i + L_{vi} \qquad (12)$$

where:

L_{ai} = adjusted adaptation luminance at fixation point i
L_i = foveal luminance for fixation point i

We will also employ these L_{ai} adaptation samples for the models of color sensitivity and visual acuity that follow.

5.2 Color Sensitivity

To simulate the loss of color vision in dark environments, we use the technique presented by Ferwerda et al. [Ferwerda96] and ramp between a scotopic (grey) response function and a photopic (color) response function as we move through the mesopic range. The lower limit of the mesopic range, where cones are just starting to get enough light, is approximately 0.0056 cd/m^2. Below this value, we use the straight scotopic luminance. The upper limit of the mesopic range, where rods are no longer contributing significantly to vision, is approximately 5.6 cd/m^2. Above this value, we use the straight photopic luminance plus color. In between these two world luminances (i.e., within the mesopic range), our adjusted pixel is a simple interpolation of the two computed output colors, using a linear ramp based on luminance.

Since we do not have a value available for the scotopic luminance at each pixel, we use the following approximation based on a least squares fit to the colors on the Macbeth ColorChecker Chart™:

$$Y_{scot} = Y \cdot \left[1.33 \cdot \left(1 + \frac{Y+Z}{X} \right) - 1.68 \right] \qquad (13)$$

where:

Y_{scot} = scotopic luminance
X,Y,Z = photopic color, CIE 2° observer (Y is luminance)

This is a very good approximation to scotopic luminance for most natural colors, and it avoids the need to render another channel. We also have an approximation based on RGB values, but since there is no accepted standard for RGB primaries in computer graphics, this is much less reliable.

Figure 16 shows our dimmed bathroom scene with the human color sensitivity function in place. Notice there is still some veiling, even with the lights reduced to 1/100th their normal level. This is because the relative luminances are still the same, and they scatter in the eye as before. The only difference here is that the eye cannot adapt as well when there is so little light, so everything appears dimmer, including the lamps. The colors are clearly visible near the light sources, but gradually less visible in the darker regions.

Figure 16. Our dimmed bathroom scene with tone mapping using human contrast sensitivity, veiling luminance and mesopic color response.

5.3 Visual Acuity

Besides losing the ability to see contrast and color, the human eye loses its ability to resolve fine detail in dark environments. The relationship between adaptation level and foveal acuity has been measured in subject studies reported by Shaler [Shaler37]. At daylight levels, human visual acuity is very high, about 50 cycles/degree. In the mesopic range, acuity falls off rapidly from 42 cycles/degree at the top down to 4 cycles/degree near the bottom. Near the limits of vision, the visual acuity is only about 2 cycles/degree. Shaler's original data is shown in Figure 17 along with the following functional fit:

$$R(L_a) \approx 17.25\arctan(1.4\log10(L_a) + 0.35) + 25.72 \qquad (15)$$

where:

$R(L_a)$ = visual acuity in cycles/degree

L_a = local adaptation luminance (in cd/m^2)

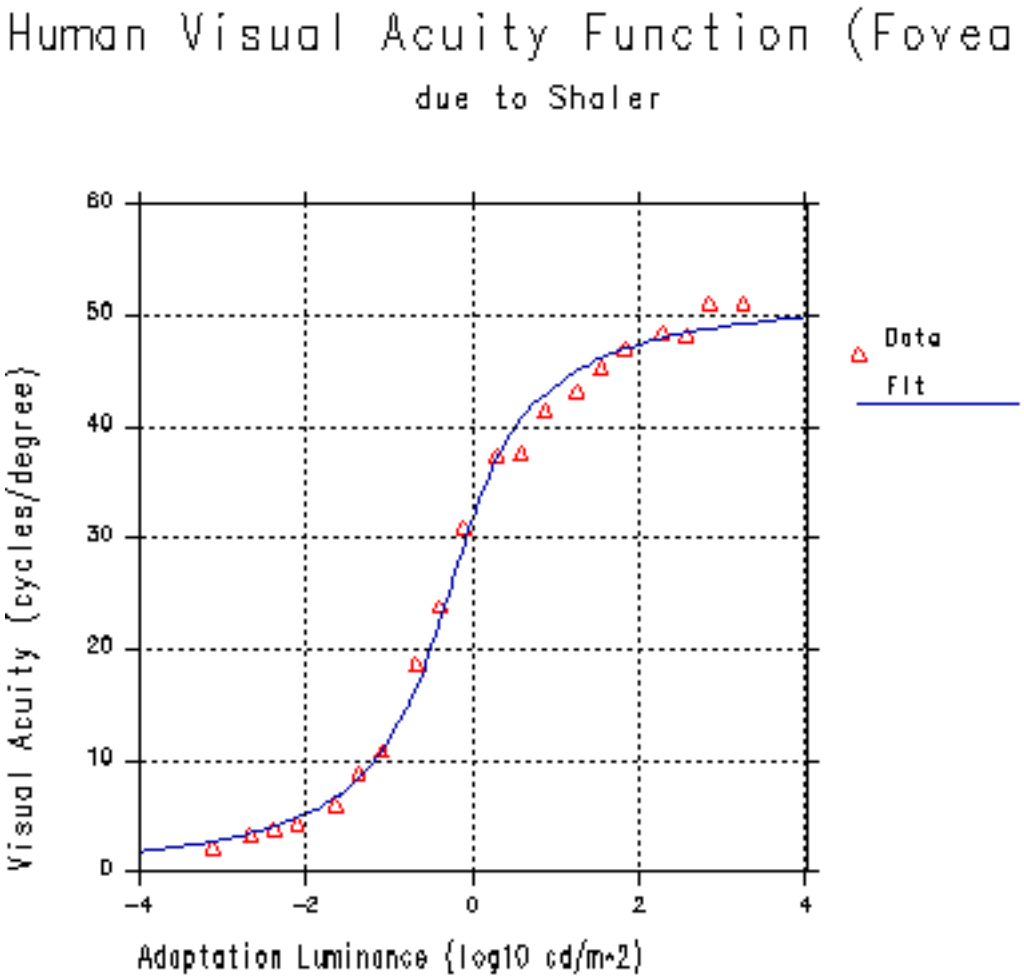


Figure 17. Shaler's visual acuity data and our functional fit to it.

In their tone mapping paper, Ferwerda et al. applied a global blurring function based on a single adaptation level [Ferwerda96]. Since we wish to adjust for acuity changes over a wide dynamic range, we must apply our blurring function locally according to the foveal adaptation computed in Equation (12). To do this, we implement a variable-resolution filter using an image pyramid and interpolation, which is the *mip map* introduced by Williams [Williams83] for texture mapping. The only difference here is that we are working with real values rather than integers pixels.

At each point in the image, we interpolate the local acuity based on the four closest (veiled) foveal samples and Shaler's data. It is very important to use the foveal data (L_{ai}) and not the original pixel value, since it is the fovea's adaptation that determines acuity. The resulting image will show higher resolution in brighter areas, and lower resolution in darker areas.

Figure 18 shows our dim bathroom scene again, this time applying the variable acuity operator applied together with all the rest. Since the resolution of the printed image is low, we enlarged two areas for a closer look. The bright area has an average level around

25 cd/m^2, corresponding to a visual acuity of about 45 cycles/degree. The dark area has an average level of around 0.05 cd/m^2, corresponding to a visual acuity of about 9 cycles/degree.

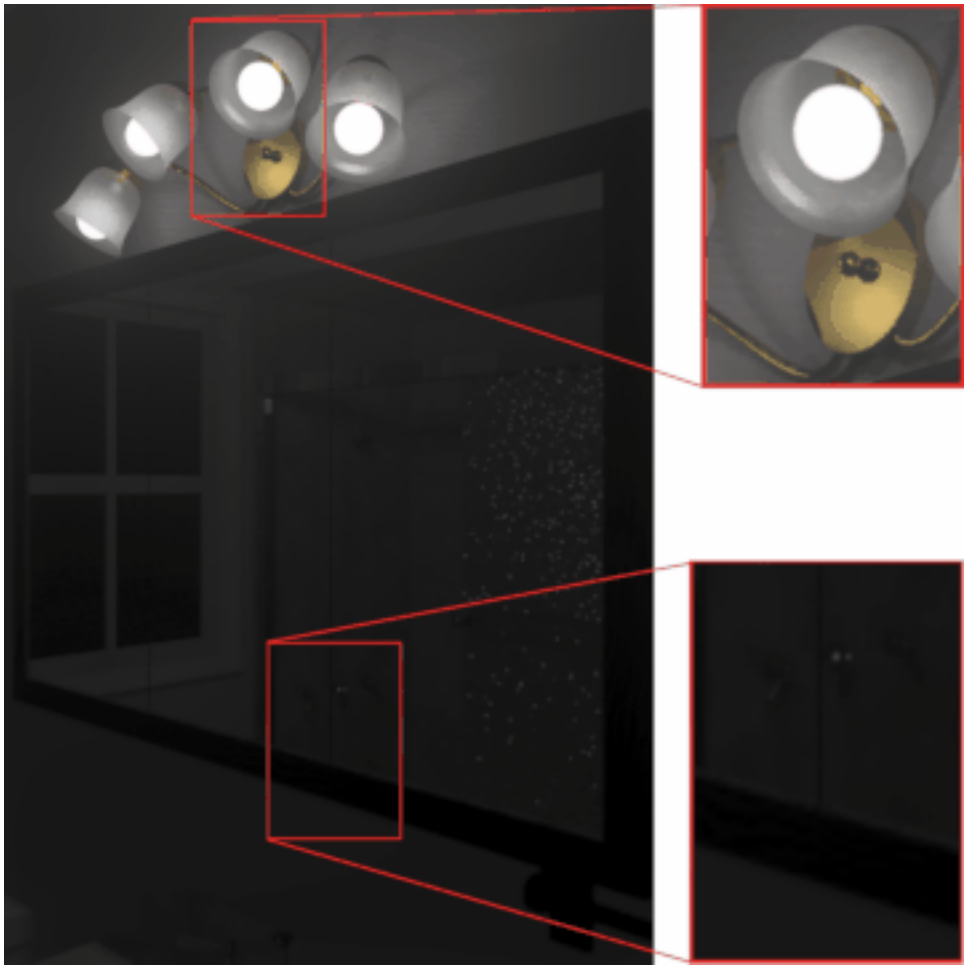


Figure 18. The dim bathroom scene with variable acuity adjustment. The insets show two areas, one light and one dark, and the relative blurring of the two.

6 Method Summary

We have presented a method for matching the visibility of high dynamic range scenes on conventional displays, accounting for human contrast sensitivity, veiling luminance, color sensitivity and visual acuity, all in the context of a local adaptation model. However, in presenting this method in parts, we have not given a clear idea of how the parts are integrated together into a working program.

The order in which the different processes are executed to produce the final image is of critical importance. These are the steps in the order they must be performed:

```
procedure match_visibility()
compute 1° foveal sample image
compute veil image
add veil to foveal adaptation image
add veil to image
blur image locally based on visual acuity function
apply color sensitivity function to image
generate histogram of effective adaptation image
adjust histogram to contrast sensitivity function
apply histogram adjustment to image
translate CIE results to display RGB values
end
```

We have not discussed the final step, mapping the computed display luminances and chrominances to appropriate values for the display device (e.g., monitor RGB settings). This is a well studied problem, and we refer the reader to the literature (e.g., [Hall89]) for details. Bear in mind that the mapped image accounts for the black level of the display, which must be subtracted out before applying the appropriate gamma and color corrections.

Although we state that the above steps must be carried out in this order, a few of the steps may be moved around, or removed entirely for a different effect. Specifically, it makes little difference whether the luminance veil is added before or after the blurring function, since the veil varies slowly over the image. Also, the color sensitivity function may be applied anywhere after the veil is added so long as it is before histogram adjustment.

If the goal is to optimize visibility and appearance without regard to the limitations of human vision, then all the steps between computing the foveal average and generating the histogram may be skipped, and a linear ceiling may be applied during histogram adjustment instead of the human contrast sensitivity function. The result will be an image with all parts visible on the display, regardless of the world luminance level or the presence of glare sources. This may be preferable when the only goal is to produce a nice-looking image, or when the absolute luminance levels in the original scene are unknown.

7 Results

In our dynamic range compression algorithm, we have exploited the fact that humans are insensitive to relative and absolute differences in luminance. For example, we can see that it is brighter outside than inside on a sunny day, but we cannot tell how much brighter (3 times or 100) or what the actual luminances are (10 cd/m^2 or 1000). With the additional display range made available by adjusting the histogram to close the gaps between luminance levels, visibility (i.e., contrast) within each level can be properly preserved. Furthermore, this is done in a way that is compatible with subjective aspects of vision.

In the development sections, two synthetic scenes have served as examples. In this section, we show results from two different application areas -- lighting simulation and electronic photography.

Figure 19. A simulation of a shipboard control panel under emergency lighting.

Figure 20. A simulation of an air traffic control console.

Figure 21. A Christmas tree with very small light sources.

7.1 Lighting Simulation

In lighting design, it is important to simulate what it is like to *be* in an environment, not what a photograph of the environment looks like. Figures 19 and 20 show examples of real lighting design applications.

In Figure 19, the emergency lighting of a control panel is shown. It is critical that the lighting provide adequate visibility of signage and levers. An image synthesis method that cannot predict human visibility is useless for making lighting or system design judgments.

Figure 20 shows a flight controller's console. Being able to switch back and forth between the console and the outdoor view is an essential part of the controller's job. Again, judgments on the design of the console cannot be made on the basis of ill-exposed or arbitrarily mapped images.

Figure 21 is not a real lighting application, but represents another type of interesting lighting. In this case, the high dynamic range is not represented by large areas of either high or low luminance. Very high, almost point, luminances are scattered in the scene. The new tone mapping works equally well on this type of lighting, preserving visibility

while keeping the impression of the brightness of the point sources. The color sensitivity and variable acuity mapping also correctly represent the sharp color view of areas surrounding the lights, and the greyed blurring of more dimly lit areas.

Figure 22. A scanned photograph of Memorial Church.

7.2 Electronic Photography

Finally, we present an example from electronic photography. In traditional photography, it is impossible to set the exposure so all areas of a scene are visible as they would be to a human observer. New techniques of digital compositing are now capable of creating images with much higher dynamic ranges. Our tone reproduction operator can be applied to appropriately map these images into the range of a display device.

Figure 22 shows the interior of a church, taken on print film by a 35mm SLR camera with a 15mm fisheye lens. The stained glass windows are not completely visible because the recording film has been saturated, even though the rafters on the right are too dark to see. Figure 23 shows our tone reproduction operator applied to a high dynamic range version of this image, called a *radiance map*. The radiance map was generated from 16 separate exposures, each separated by one stop. These images were scanned, registered, and the full dynamic range was recovered using an algorithm developed by Debevec and Malik

[Debevec97]. Our tone mapping operator makes it possible to retain the image features shown in Figure 23, whose world luminances span over 6 orders of magnitude.

The field of electronic photography is still in its infancy. Manufacturers are rapidly improving the dynamic range of sensors and other electronics that are available at a reasonable cost. Visibility preserving tone reproduction operators will be essential in accurately displaying the output of such sensors in print and on common video devices.

Figure 23. Histogram adjusted radiance map of Memorial Church.

8 Conclusions and Future Work

There are still several degrees of freedom possible in this tone mapping operator. For example, the method of computing the foveal samples corresponding to viewer fixation points could be altered. This would depend on factors such as whether an interactive system or a preplanned animation is being designed. Even in a still image, a theory of probable gaze could be applied to improve the initial adaptation histogram. Additional modifications could easily be made to the threshold sensitivity, veil and acuity models to simulate the effects of aging and certain types of visual impairment.

This method could also be extended to other application areas. The tone mapping could be incorporated into global illumination calculations to make them more efficient by

relating error to visibility. The mapping could also become part of a metric to compare images and validate simulations, since the results correspond roughly to human perception [Rushmeier95].

Some of the approximations in our operator merit further study, such as color sensitivity changes in the mesopic range. A simple choice was made to interpolate linearly between scotopic and photopic response functions, which follows Ferwerda et al. [Ferwerda96] but should be examined more closely. The effect of the luminous surround on adaptation should also be considered, especially for projection systems in darkened rooms. Finally, the current method pays little attention to absolute color perception, which is strongly affected by global adaptation and source color (i.e., white balance).

The examples and results we have shown match well with the subjective impression of viewing the actual environments being simulated or recorded. On this informal level, our tone mapping operator has been validated experimentally. To improve upon this, more rigorous validations are needed. While validations of image synthesis techniques have been performed before (e.g., Meyer et al. [Meyer86]), they have not dealt with the level of detail required for validating an accurate tone operator. Validation experiments will require building a stable, non-trivial, high dynamic range environment and introducing observers to the environment in a controlled way. Reliable, calibrated methods are needed to capture the actual radiances in the scene and reproduce them on a display following the tone mapping process. Finally, a series of unbiased questions must be formulated to evaluate the subjective correspondence between observation of the physical scene and observation of images of the scene in various media. While such experiments will be a significant undertaking, the level of sophistication in image synthesis and electronic photography requires such detailed validation work.

The dynamic range of an interactive display system is limited by the technology required to control continual, intense, focused energy over millisecond time frames, and by the uncontrollable elements in the ambient viewing environment. The technological, economic and practical barriers to display improvement are formidable. Meanwhile, luminance simulation and acquisition systems continue to improve, providing images with higher dynamic range and greater content, and we need to communicate this content on conventional displays and hard copy. This is what tone mapping is all about.

9 Acknowldgments

The authors wish to thank Robert Clear and Samuel Berman for their helpful discussions and comments. This work was supported by the Laboratory Directed Research and Development Funds of Lawrence Berkeley National Laboratory under the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

10 References

[Chiu93]

K. Chiu, M. Herf, P. Shirley, S. Swamy, C. Wang and K. Zimmerman "Spatially nonuniform scaling functions for high contrast images," *Proceedings of Graphics Interface '93,* Toronto, Canada, May 1993, pp. 245-253.

[CIE81]
CIE (1981) *An analytical model for describing the influence of lighting parameters upon visual performance*, vol 1. Technical foundations. CIE 19/2.1, Techical committee 3.1

[Debevec97]
Debevec, Paul and Jitendra Malik, "Recovering High Dynamic Range Radiance Maps from Photographs," *Proceedings of ACM SIGGRAPH '97*.

[Ferwerda96]
J. Ferwerda, S. Pattanaik, P. Shirley and D.P. Greenberg. "A Model of Visual Adaptation for Realistic Image Synthesis," *Proceedings of ACM SIGGRAPH '96*, p. 249-258.

[Frei77]
W. Frei, "Image Enhancement by Histogram Hyperbolization," *Computer Graphics and Image Processing*, Vol 6, 1977 286-294.

[Glassner95]
A. Glassner, *Principles of Digital Image Synthesis*, Morgan Kaufman, San Francisco, 1995.

[Green83]
W. Green *Digital Image Processing: A Systems Approach*, Van Nostrand Reinhold Company, NY, 1983.

[Hall89]
R. Hall, *Illumination and Color in Computer Generated Imagery*, Springer-Verlag, New York, 1989.

[Holladay26]
Holladay, L.L., *Journal of the Optical Society of America*, 12, 271 (1926).

[Meyer86]
G. Meyer, H. Rushmeier, M. Cohen, D. Greenberg and K. Torrance. "An Experimental Evaluation of Computer Graphics Imagery," *ACM Transactions on Graphics*, January 1986, Vol. 5, No. 1, pp. 30-50.

[Mokrane92]
A. Mokrane, "A New Image Contrast Enhancement Technique Based on a Contrast Discrimination Model," *CVGIP: Graphical Models and Image Processing*, 54(2) March 1992, pp. 171-180.

[Moon&Spencer45]
P. Moon and D. Spencer, "The Visual Effect of Non-Uniform Surrounds", *Journal of the Optical Society of America*, vol. 35, No. 3, pp. 233-248 (1945)

[Nakamae90]
E. Nakamae, K. Kaneda, T. Okamoto, and T. Nishita. "A lighting model aiming at drive simulators," *Proceedings of ACM SIGGRAPH 90*, 24(3):395-404, June, 1990.

[Rushmeier95]
H. Rushmeier, G. Ward, C. Piatko, P. Sanders, B. Rust, "Comparing Real and Synthetic Images: Some Ideas about Metrics," *Sixth Eurographics Workshop on Rendering*, proceedings published by Springer-Verlag. Dublin, Ireland, June 1995.

[Schlick95]
C. Schlick, "Quantization Techniques for Visualization of High Dynamic Range Pictures," *Photorealistic Rendering Techniques* (G. Sakas, P. Shirley and S. Mueller, Eds.), Springer, Berlin, 1995, pp.7-20.

[Spencer95]
G. Spencer, P. Shirley, K. Zimmerman, and D. Greenberg, "Physically-based glare effects for computer generated images," *Proceedings ACM SIGGRAPH '95*, pp. 325-334.

[Stevens60]
S. S. Stevens and J.C. Stevens, "Brightness Function: Parametric Effects of adaptation and contrast," *Journal of the Optical Society of America*, 53, 1139. 1960.

[Tumbline93]
J. Tumblin and H. Rushmeier. "Tone Reproduction for Realistic Images," *IEEE Computer Graphics and Applications*, November 1993, 13(6), 42-48.

[Ward91]
G. Ward, "A contrast-based scalefactor for luminance display," In P.S. Heckbert (Ed.) *Graphics Gems IV*, Boston, Academic Press Professional.

[Williams83]
L. Williams, "Pyramidal Parametrics," *Computer Graphics*, v.17,n.3, July 1983.

Appendix G:

A High Dynamic Range Display Using Low and High Resolution Modulators

Reprinted from 2003 Society for Information Display Symposium

Citation:

Seetzen, Helge, Lorne Whitehead, Greg Ward, "A High Dynamic Range Display Using Low and High Resolution Modulators," *The Society for Information Display International Symposium*, May 2003.

Eurographics 2003

P. 54.2: A High Dynamic Range Display Using Low and High Resolution Modulators

Helge Seetzen, Lorne A. Whitehead

Dept. of Physics and Astronomy, University of British Columbia, Vancouver, Canada

Greg Ward

Albany, CA, USA

Abstract

We have developed an emissive high dynamic range (HDR) display that is capable of displaying a luminance range of 10,000cd/m^2 to 0.1cd/m^2 while maintaining all features found in conventional LCD displays such as resolution, refresh rate and image quality. We achieve that dynamic range by combining two display systems – a high resolution transmissive LCD and a low resolution, monochrome display composed of high brightness light emitting diodes (LED). This paper provides a description of the technology as well as findings from a supporting psychological study that establishes that correction for the low resolution display through compensation in the high resolution display yields an image which does not differ perceptibly from that of a purely high resolution HDR display.

1. Introduction

The ultimate goal of digital display systems is to present images that are visually indistinguishable from the real setting they portray. Conventional display technology (LCD, CRT, plasma, etc) have achieved part of that goal by introducing both spatial resolution and refresh rates that are beyond the visual acuity of a human viewer. However, even the highest quality displays available today are incapable of showing the true luminance (brightness) range we perceive in real life. Every day we encounter light sources in our natural environment that are several orders of magnitude brighter than any conventional display. A typical fluorescent light fixture has a luminance of approximately 2,000cd/m^2 and on a sunny day objects illuminated by the sun can easily have luminance values up to 10,000cd/m^2. But current computer monitors can only display images within a luminance range of approximately 1cd/m^2 to 300cd/m^2, and as a result are unable to display luminance-realistic images.

2. High Dynamic Range Display

To overcome the dynamic range limitation of conventional displays, the HDR technology replaces the uniform backlight of an LCD by an active matrix array of ultra high brightness white LEDs. These current-controlled diodes are capable of emitting over 250,000cd/m^2 at maximum current and no emission in the off state with an effective 8-bit resolution between those states if driven by an 8-bit Digital to Analog Converter. The LED array then effectively constitutes a very low-resolution (5mm per LED), but very high brightness display. The low-resolution image of the LED array is then projected through a color LCD, which displays a similar, but high resolution, version of the image. This modification is described pictorially in Figure 1. The arrangement of the LED does not necessarily have to be as shown but can be hexagonal for closer packing or any other arrangement that is appropriate for the application.

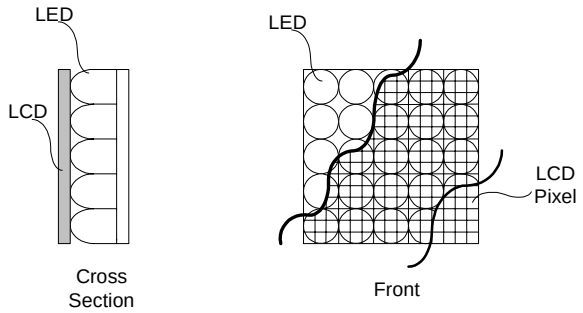


Figure 1. Layout of LED array behind color LCD

This double modulation then defines the boundary of the dynamic range of the HDR display. The darkest state is produced by a dark LED behind an LCD pixel set to black, the brightest state is produced by an LED driven at maximum current behind an LCD pixel set to maximum transmission. Such a 'multiplication' of two 8-bit display systems results in a 16-bit dynamic range with an adequate number of non-linearly distributed distinct luminance levels to create a smoothly addressable gradient per color.

3. Blur Correction Method

The optics of the HDR display have been designed such that each LED produces a smoothly varying illuminance pattern on the front display, with the luminance distribution of adjacent LCD overlapping to the extent required to yield fairly uniform luminance when all rear pixels are on.

As described, this leads to a perceivable blur of the final output image. This can be counteracted by appropriate corrections to the front image (i.e., the image on the LCD) since the nature of the blur is known. Based on a 16-bit input image, the display driver can establish the optimum setting for the LED array, which provides a known luminance distribution across the array. It is then possible to divide the 16-bit input image by the known LED array luminance distribution to get the image transmission values that need to be displayed on the LCD. Physically, these two images are multiplied as the light passes from each LED through a cluster of LCD pixels. This method will generally re-create the high-resolution image quality defined in the 16-bit input file. The exceptions are at very high contrast boundaries, where the dynamic range of the LCD is insufficient to make the appropriate image correction. The following example will illustrate the steps of this method.

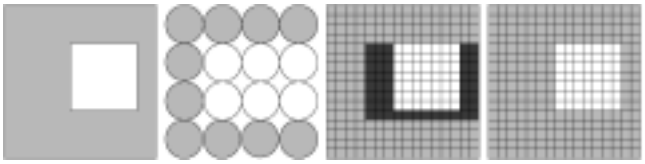


Figure 2. Composite Formation of HDR image. From left to right: Desired image, LED setting, LCD panel setting and final HDR image.

In Figure 2, we wish to display a picture of a bright square upon a dull gray background. On a regular monitor, the square would be a cluster of white pixels, and the gray box would be a cluster of gray pixels. When using the modified HDR technology, the boundary between the white and the gray falls on several 4x4 pixel clusters, each corresponding to a single LED. (For ease of visualization in this example the size ratio of LED to LCD pixel has been set to 1:4.) The LED behind the cluster has to be set to maximum brightness to make the white as bright as possible (assuming that the white square is as bright as the maximum output of the display). Conversely, the LEDs behind the gray border have to be set to a low output because the border isn't very bright. The small part of gray that happens to be in the 4x4 pixel group that also has the bright white light in it, and is thus backlit by the maximum brightness LED, has to be treated differently by the system. That gray region in that area will look a lot brighter than the gray in all the other 4x4 pixel groups. To counter this effect the system sets these apparently gray pixels to a significantly darker shade of gray in the image on the front LCD display, thereby reducing the final output to the same gray as the one in the neighboring 4x4 pixel groups. Basically, in the all gray 4x4 pixel groups we have a low-light LED modulated by a mediumly transmissive LCD pixel resulting in light gray while in the 4x4 pixel groups with some part of the white square in it we have a bright light LED modulated by a very weakly transmissive LCD pixel resulting once again in light gray.

Through this method it is possible to use a very low resolution backplane behind a conventional LCD display without losing resolution. This has several significant advantages. The lower resolution of the backplane drastically reduces the computational effort in computing and transmitting the image. The file size of HDR images thus does not have to increase at all (compared to conventional 8-bit image files) since the small amount of extra information for the backplane (less than 1% of the LCD resolution) can be stored in the available free space that is part of most conventional file formats (JPEG, TIF, etc). Furthermore, the extra information is small enough that no modification of the videostream from PC to display is necessary (i.e. there is no direct requirement for a 16-bit graphic card). Finally, such a design allows complete backward compatibility. A user with an HDR display and an HDR image file can enjoy the improved HDR image. A user with an HDR display but lacking the appropriate file can still easily view the full content of a conventional 8-bit image file (the front image component will be displayed and the HDR display, after not finding the small extra backplane 'tag' in the file, will set the LED array to some uniform luminance level to emulate a conventional 8-bit LCD display for the duration of that image). And a user with access to an HDR file but no HDR display can still view a very reasonably tone mapped representation of the image on the conventional display. (The display will simply ignore the extra HDR tag.)

4. Psychology Background

The human visual system has evolved to comfortably view a luminance range of 10,000cd/m^2 to 0.1cd/m^2. But even within the given range our brightness perception is not perfect. In particular, optical imperfections in our eyes limit our brightness perception, including scattering in the cornea, lens and retina, and diffraction in the coherent cell structures on the outer radial areas of the lens. These effects are responsible for the "bloom" and "flare lines" seen around bright objects. The diffraction effect causes a lenticular halo, which is ignored in this project as it does not significantly impact the perceived image.

Bloom [1] (often termed "disability glare" or "veiling luminance") is the result of light scattering in the ocular media contributed roughly equally from the cornea, crystalline lens and retina scattering. Figure 4 illustrates an example of bloom. Light from source A scatters inside the eye onto the same receptors as if coming from source B, thus adding an effective luminance L_e. Since light is added to both the dark and light parts of B, the effective contrast ratio L_2/L_1 is reduced. The magnitude of L_e depends on the angle of separation α and the luminance and solid angle of the source.

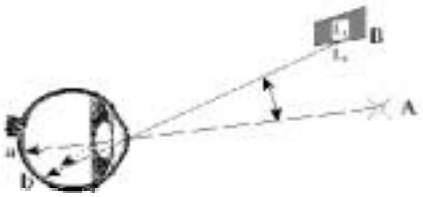


Figure 3: Bloom effect due to intraocular scattering.

Empirical psychophysics research led to a point spread function P(α) for the bloom effect [1].

$$P(\alpha) = \eta\delta(\alpha) + \frac{c}{f(\alpha)}$$

Equation 1: Bloom Point Spread Function.

The constant η represents the fraction of the light that is not scattered and δ(α) is the ideal point spread function. k is an empirically determined calibration constant. The function f(α) has been successfully modeled to very high precision with a first order term of $f(\alpha) = \alpha^2$.

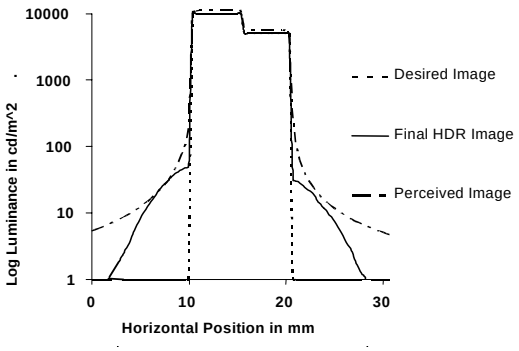


Figure 4. Comparison of display and eye introduced blur at a contrast boundaries of 0cd/m^2 to 10,000cd/m^2, 10,000cd/m^2 to 5,000cd/m^2 and 5,000cd/m^2 to 1cd/m^2.

Using this model and the average constants as outlined, it is possible to approximate the perceived luminance pattern corresponding to each image on the HDR display. In particular, the model provides a description of the perceived blur at each high contrast boundary. As long as this perceived blur is more significant than the image degradation of the HDR display due to lack of LCD dynamic range, then no degradation will be perceived. For the blur introduced by a 5mm LED at a viewing distance of 30cm or more, this will be the case.

5. Psychophysical Validation

In order to validate the predictions of the psychophysical model described above, we have carried out a test with 20 observers. The test included two stages of comparison of real and test images. The images used were a photograph of the Stanford Memorial Church (a 16-bit image) and a test image designed to show all possible boundaries between 16 luminance levels, each twice as high as the last. All images were shown as pairs on a 15" screen size at a viewing distance of 50cm.

The first stage was designed to validate the general claim that high dynamic range images appear more realistic and pleasant than low dynamic range images shown on a conventional display. For this comparison, the test image and the real scene were shown side by side in a random arrangement of low and high dynamic range settings. The low dynamic range image was presented on the high dynamic range display by setting the rear modulator (i.e. the low resolution image plane) to a uniform gray level of the same brightness as a conventional LCD backlight. This leaves only the dynamic range of the front image plane for modulation of the light and thus emulates the display capabilities of a conventional LCD, since the front image plane is simply such a conventional LCD.

The second stage was designed to provide empirical data for the degree of discomfort und unrealism, if any, associated with the blur introduced by the rear modulator. In order to vary the degree of blur in the rear modulator we replaced the matrix of LEDs with a monochrome digital mirror projector whose light passes through a Fresnel lens and the appropriate diffuser before reaching the LED. Optically this is the equivalent of the LED matrix but with the benefit of control over the resolution of the rear modulator. In this stage, the subjects were exposed to two adjacent high dynamic range images of the same scene (either the real scene or the test image). One of the two images was randomly chosen to be the reference image featuring a resolution match between the front and rear modulator (i.e. the same high resolution at both the LCD and the projection with pixel by pixel alignment of both images). The other side of the test image presented the same scene but with a varying degree of blur in the rear modulator. The blur was created by blurring the image data for the projector. The appropriate blur correction image was then displayed on the LCD. At a 5mm blur this is equivalent to the blur introduced by the LED backplane. We investigated 4 different sizes of the low-resolution 'pixel' (2.5mm, 5mm, 10mm, 15mm).

For each set of images, the participants were asked to provide ratings on 5 semantic differential bipolar adjective pairs (*bright - dim, interesting - monotonous, sharp-smooth, pleasant – unpleasant, realistic - unrealistic*). The last scale (*realistic-unrealistic*) was omitted for the test image. In general, we expected that high dynamic range image would be considered brighter, less uniform, more interesting and more realistic than corresponding low dynamic range images. In the comparison of blurred and reference high dynamic range images we expected that the blurred images would be perceived as progressively smoother and potentially less pleasant and less realistic with increasing blur. For the comparison of blurred and reference images there could be a small difference in the perception of brightness. The artefact halos introduced by the blurred image at high contrast boundaries could trigger a perception of brightness as our visual system considers halos of this kind to be indicators of bright areas.

6. Results – Display Performance

A fully functioning prototype of the HDR display was constructed using a 6" diagonal color LCD and a 16x12 LED array. The prototype uses a 2% transmissive LCD and is capable of showing a maximum brightness of over 3,000cd/m^2. We have tested the setup with commercially available 7% transmissive LCD (such as those found in most laptops) and measured a maximum brightness of over 10,000cd/m^2.

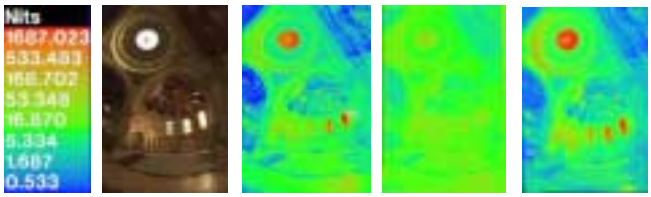


Figure 5. Luminance maps of the memorial church high dynamic range image. From left to right: Scale, image tone-mapped to 8-bit, false-color of original 16-bit image, conventional display (NEC MultiSync XE21), HDR display.

7. Results – Psychophysical Validation

The first stage of the HDR display quality test provided the anticipated results. Low dynamic range (i.e. 8-bit) images were perceived as significantly less bright, less interesting and somewhat less pleasant and less realistic. Perception of the sharpness of the image was unaffected by the reduction from 16-bit to 8-bit as one would expect given that both images where shown at the same spatial resolution.

The comparison of non-blurred and increasingly blurred high dynamic range images indicated that no degradation of the image was perceived even with significant blur of the rear display. In particular, we did not observe any decrease in the perception of sharpness of the image in the range of blur sizes used in the test (2.5mm to 15mm). Instead, the blurred images were consistently observed as sharper than the non-blurred high dynamic range image. We believe that this is the result of the blur compensation features found in the front display which might slightly overcompensate for the blur in the rear display. Such overcompensation could lead to very slight dark edges around bright areas and slightly lighter edges around dark areas. This effect is unnoticeable during close inspection of any particular area but might leads to a crisper overall appearance of the image.

All other scales (brightness, interest, pleasantness and realism) followed approximately equal trends and consequently all four scales will be treated as a general quality scale in the following. The blurred test pattern was perceived to be of equal quality as the non-blurred test pattern through the entire range of increasing blur from 2.5mm to 15mm. This result is consistent with the psychological model of intraocular scattering and the assumption that even very large blur size will not lead to perceptible

degradations even in fairly artificial scenes composed effectively entirely of sharp high contrast boundaries.

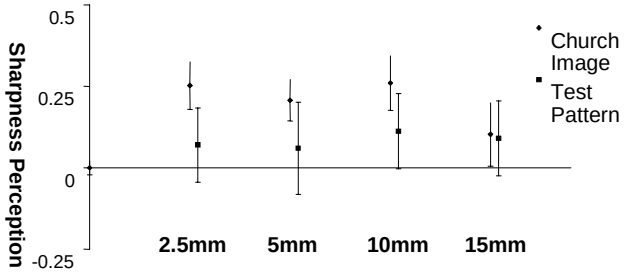


Figure 6. Sharpness ratings at increasing blur of the rear display. A rating below 0 (up to -0.5) indicates that the 8-bit image was perceived as less bright than the non-blurred 16-bit image and vice versa for ratings above 0 (up to 0.5).

At small blur sizes, the Memorial Church image was perceived to have higher general quality than the non-blurred version of the image. This higher quality perception diminished with increasing blur size and at 15mm both the blurred and non-blurred images were perceived to be of approximately equal quality. We believe that the higher quality perception at small blur sizes is the result of sub-pixel misalignment of the two display layers which would lead to a small loss of high spatial frequency contrast in the non-blurred image. This effect does not occur in any blurred image since the effective pixel size of the rear display is so much higher than the pixel size of the high resolution display that sub-pixel misalignment becomes insignificant.

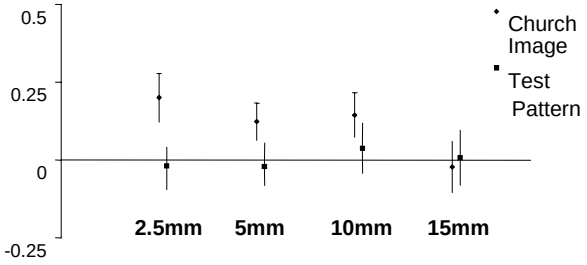


Figure 7. General quality ratings at increasing blur of the rear display.

The test results provide statistically significant support for the postulate that 2.5-15mm blur of the rear display does not degrade the perception of sharpness or general display quality. In addition, it is clear from the first stage of the test that 8-bit low dynamic range images are perceived as vastly inferior to realistic 16-bit high dynamic range images.

8. Future Work

The presence of an active backlight offers additional opportunities to enhance the performance of the display and overcome several challenges of conventional LCD displays. Two such challenges are motion blur and color gamut problems of the LCD technology. Due to the low refresh rate of LCDs it is often impossible to show moving objects without a motion trail. Several display manufacturers have proposed a solution to this problem [2]. By appropriately flashing the backlight in sync with the LCD, it is possible to significantly reduce motion blur. This operation is challenging if the backlight is a conventional fluorescent tube (or array thereof) but simple if the backlight already is an active matrix array of multiplexed LEDs.

Similarly, the use of LEDs in the backplane overcomes the critical limitations of the LCD color gamut [3]. It is possible to replace the white LEDs with combined red, green and blue LEDs which are driven as single elements. Such RGB LEDs provide a spectrum with three narrow peaks at red, green and blue, allowing for a significantly better color gamut than LCD or even CRT displays. Like the solution of the motion blur problem, this benefit comes at almost no additional cost and effort as a natural consequence of using the HDR technology.

9. Conclusion

The HDR technology yields a significantly enhanced representation of real scenes by portraying the entire visual range that is comfortably accessible to humans, with the added potential of an increased color gamut. It does so without any noticeable banding of luminance steps or any degradation of other characteristics of conventional LCDs (resolution, refresh rate, etc). In particular, despite the physical imperfection at high contrast boundaries due to the lower resolution of the LED array, there is no perceived blur at those boundaries necause such effects are masked by intraocular scattering. As a result, the HDR display is perceptually free of degradation across the entire 10,000cd/m^2 to 0.1cd/m^2 luminance range. It is clear that 16-bit images are significantly more desirable than 8-bit images and consequently any 16-bit display is desirable. But the HDR display with low and high resolution modulators not only offers 16-bit image quality but achieves this high dynamic range without the need for a 16-bit videostream and without the costly requirement for two high resolution display layers. These advantages come at no discernable image quality cost.

10. Acknowledgements

The authors thank Paul Debevec for providing the Stanford Memorial Church high dynamic range photograph used in the psychophysical test. Further thanks goes to Thomas Wan for assisting with the design and implementation of the viewing test.

11. References

[1] Vos, J., Disability glare- a state of the art report, CIE Journal 3, 2 (1984), 39-53

[2] Bruinink, J. et al, Improved Motion-Picture Quality of LCD TV, SID Symposium 2002

[3] Ohtsuk, H. et al, 18.1-inch XGA TFT-LCD with Wide Color Reproduction using High Power LED-Backlighting, SID Symposium 2002