

Imaging pipeline and post-processing

Display and images

# Digital imagery

- Digital imagery appears in all forms of media. Most of these are:
  - Digital photos or other types of 2D pictures scanned or loaded in computer
  - **Generated in 3D using modeling and rendering software**
  - **<image: variety of digital images: real, rendered, video, photo>**

# Display device

- Images are shown on a **display device**
  - e.g., monitor, television screen, etc.
  - <images: variety of display devices>
- Display devices are adapted for showing **raster images → raster display devices**
- Raster display device is made of **arrays of pixels → discrete representation**
  - For example, each pixel of image created with digital camera (and thus rendering) stores red, green and blue value used to drive the red, green and blue values of screen pixel.
  - <image: raster device and raster image>

- Number of rendered images for display per second is called frames per second (FPS):
  - Real-time graphics  $> 60$  fps
  - Interactive graphics  $1 < x < 20$  fps. TODO
  - Offline graphics  $< 1$  fps

# Raster image graphics

- Raster images are **rectangular array of values** called **pixels**.
- All values in such have same type, examples:
  - Floating point numbers representing levels of **gray**
  - Floating point triplets representing mixture of red, green and blue (**RBG**)
  - Floating point numbers representing **depth** from camera
- Such rectangular array of numbers can be interpreted in many ways
  - This is powerful since we can store/encode any information in such array
  - Numbers in array do not have particular significance until stored in certain **standardized file format**

# Naive image file format: PPM

- Portable Pixel Map (PPM) text-based\* version file format is useful for understanding how basics of image organization works
- Header:
  - Width (w) and height (h) of image is specified
  - Maximum color value
- Pixel values:
  - $3 * w * h$  color values representing red, green and blue components of pixel
  - Pixel values ordering: left-to-right, top-to-bottom
  - Separated by white-space
- Notes:
  - Easy to write and analyze
  - Inefficient and highly redundant (compression is not present)
  - Little information about image

```
P3
# feep.ppm
4 4
15
0 0 0 0 0 0 0 0 0 15 0 15
0 0 0 0 15 7 0 0 0 0 0 0
0 0 0 0 0 0 0 15 7 0 0 0
15 0 15 0 0 0 0 0 0 0 0 0
```

\* Text-based version carries code P3, binary version carries code P5. More information: <https://netpbm.sourceforge.net/doc/ppm.html>

# Raster image file formats

- Images are stored in many formats; typically the storage format closely related to the **display format**
- File formats use different compression strategies:
  - **Lossless** compression is one in which data occupies less space but from which original data can be reconstructed.
  - **Lossy** compression is resulting in even less occupied space, but original data can not be restored. Nevertheless, existing data is enough for intended use
- In some cases, content is described in terms of **channels**
  - e.g., red values for all pixels represents red **color channel**
  - e.g., depth values form depth channel
  - e.g., transparency values from alpha channel
- Some formats store **metadata**
  - e.g., bit depth of color channel (e.g., 8 bit)
  - e.g. information on when and with which program the image was produced



# Tip: file formats in production

- Blender:

[https://docs.blender.org/manual/en/latest/files/media/image\\_formats.html](https://docs.blender.org/manual/en/latest/files/media/image_formats.html)

- Houdini:

[https://www.sidefx.com/docs/houdini/io/formats/image\\_formats.html](https://www.sidefx.com/docs/houdini/io/formats/image_formats.html)

# Raster image file formats

- Examples of often used raster image file formats are:
  - Tag Image File Format - JPEG
  - Portable Network Graphics - PNG
  - Tag Image File Format - TIFF/TIF
- Examples of advanced image file formats:
  - OpenEXR (<https://github.com/AcademySoftwareFoundation/openexr>)
  - High Dynamic Range Image - HDRI
- Library for handling different file formats for computer graphics applications:  
<https://github.com/OpenImageIO/oiio>

# JPEG

- Most digital cameras produce JPEG images
- Thus they are de-facto standard for storing gray or RGB values
- This format is lossy
  - Problematic for comparing the image due to differences in compression algorithm

# PNG

- PNG file format is more compact than PPM and equally easy to use.
- Lossless data compression
- Developed as an improved replacement for Graphics Interchange Format (GIF)
- Supports:
  - RGB images with and without alpha channel
  - Grayscale images with and without alpha channel

# TIFF

- TIFF images store multiple channels
  - Each channel store description of contents
- TIFF is ideal for representing intermediate or final results
  - In image editing and compositing tools, multiple images are often blended or laid atop one another
- Compression
  - uncompressed
  - Compressed with lossless scheme
  - Compressed with lossy scheme

# Note:vector graphics images

- Alternative to raster graphics image is **vector graphics image**
  - Images are created directly from geometric shapes defined on 2D Cartesian coordinate system.
  - Information in such image is stored as points, lines, curves and polygons
- Today, raster-based monitors and printers are typically used
  - Nevertheless, vector data and software is used in applications where geometric precision is required: engineering, architecture, typography
  - To display an image, rendering is performed to evaluate analytically defines shapes on raster display

# Sampling and reconstruction

Clash between continuous and discrete representations

# Continuous and discrete

- Due to raster display devices, rendered images must be discrete array of pixels
- 3D scene color values, on the other hand, are continuously changing
- Rendering process, in order to create a discrete image of 3D scene, samples the pixels on film plane and for each sample calculates color
- Quality of final discrete 2D image is highly determined by:
  - How is the sampling of pixel performed
  - Methods which combine pixel samples in pixel color

<image: 3D scene which is cont and image which is discrete>



- Image: show how discretization process can cause aliasing for motivation, aliasing will be then explained

# Scene sampling and Image buffer

- Process of rendering images is inherently a sampling task
- Generation of image is the process of sampling a 3D scene in order to obtain colors for each pixel in the image
  - In rendering process, camera description is used to generate samples on film plane for which color is calculated
  - Color of each pixel for each sample is stored in **image buffer** – intermediate representation of image used by renderer
  - Final image is reconstructed from pixel samples
- 3D scene is continuous description which must be discretized for display
  - Doing so, the amount information is reduced → problems!

# Aliasing

Sampling process might result in **aliasing** – an unwanted artifact in spatial or temporal domain (spatial vs temporal sampling)

- Aliasing happens if signal is sampled at too low frequency than the original and thus reconstruction can not be done correctly
- Sampling frequency must be twice the maximum frequency of signal which is sampled – **sampling theorem** – **Nyquist rate/limit**
- As maximum frequency must be present, signal must be band-limited
- **3D scenes are normally never band-limited** when rendering with point samples
  - Edges of triangles or shadow boundaries produce signal that changes discontinuously and produces frequencies which are infinite
  - Any rapid change of color can cause aliasing problems: specular highlights
  - Also, pixel footprint in the scene may contain lots of objects which are hard to sample even with dense number of samples per pixel
- Point sampling is almost always used and entirely avoiding aliasing is not possible
- At times, it is possible to know when the signal is band limited and then anti-aliasing is performed
  - Example is texture usage: frequency of texture sampling can be compared to sampling rate of pixel and if texture sampling is higher then algorithms for band limiting the texture are used.
- Sampled signal must be reconstructed to recover original signal → **filtering** - final image is reconstructed from pixel samples

- Examples of aliasing:
  - Jagged edges of line or triangle edge, flickering highlights (firefys)
  - TODO: images

# Antialiasing\*

- Different methods:
  - Tradeoff between: quality, ability to capture sharp details or other phenomena, appearance, memory, speed and computing power
- Screen-based antialiasing methods:
  - Operate only on the output samples of the pipeline
- 
- RTR 5.4.
- [https://www.pbr-book.org/3ed-2018/Sampling\\_and\\_Reconstruction/Image\\_Reconstruction](https://www.pbr-book.org/3ed-2018/Sampling_and_Reconstruction/Image_Reconstruction)

\* In terms of signal processing this can be seen as sampling, filtering and reconstruction.

# Antialiasing: sampling rate

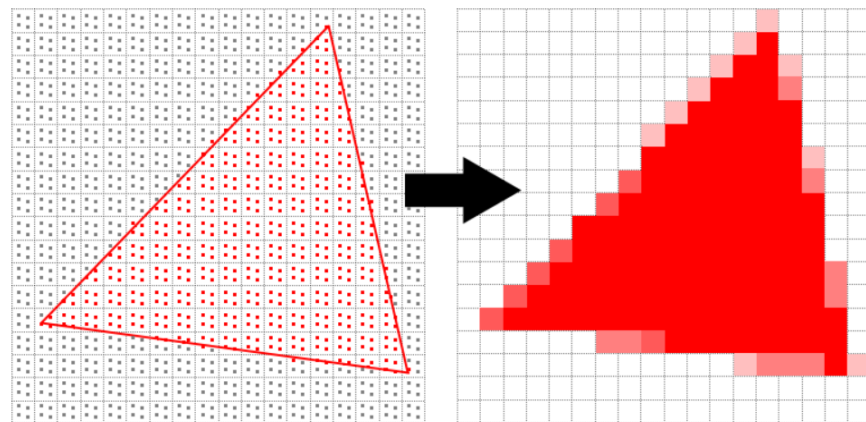
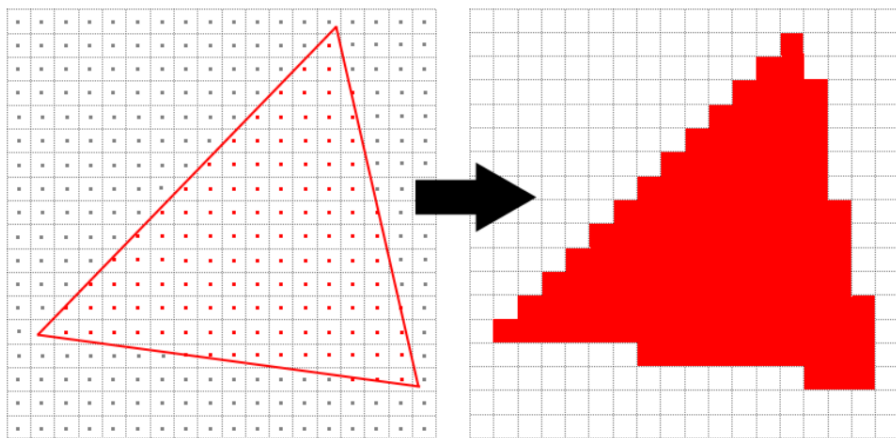
- Often problem causing aliasing is low sampling rate
- Let's see what happens with one sample per pixel:
  - TODO RTR fig 5.24
- Let's see what happens with four samples per pixel:
  - TODO RTR fig 5.24

# Antialiasing: sampling rate

- By using more samples per pixel and blending those in some fashion a better pixel color can be computed
- Screen-based antialiasing schemes:
  - Sampling pattern is used for the screen
  - Samples are points samples
  - Pixel color is sum of weighted pixel samples

# MSAA

- Aliasing appears always when there are a lot of details under a pixel footprint in the scene
- To solve this, multiple sample per pixels can be used to “catch” high frequency details.
  - This method is called Multisample anti-aliasing (MSAA)



MSAAx4



# Supersampling




- Supersampling or oversampling Antialiasing algorithms take more than one sample per pixel
  - Full-screen anti-aliasing (FSAA) or super sampling antialiasing (SSAA)

Limitations of display device

# Limitation of display device

- **Display devices are limited** in:
  - Resolution (number of pixels)
  - Brightness (intensity)
  - Contrast
  - Color (gamut)
- Rendered images may contain values which can not be directly shown on display devices.

# Display encoding

- All input values (e.g., texture image) for rendering and all values during rendering must be in **linear** colorspace.
  - Linear values are needed for correct addition and multiplication operations
- On the other hand, display devices have **nonlinear** (power law) relationship between input voltage and display radiance
  - Early display devices that were based on cathode-ray tube (CRT). As energy level applied to pixel is increased, the radiance emitted doesn't grow linearly but according to power law. For example, pixel set to 50% will emit  $0.5^2$  amount of light.
  - LCDs and other display devices mimic the CRT response, although have different intrinsic tone response curves.
  - Relationship between digital values in image buffer and radiance levels emitted from the display is described with electrical-optical transfer function (EOTF)\*
- Problem 1: image which is rendered is in linear colorspace and will not display correctly if shown directly
  - Linear values will appear to dim on the screen 
- Problem 2: when working in modeling tools where we pick texture images or colors on screen, those colors are encoded for display device so we can see them properly
  - They can not be used in rendering computation directly because they are in non-linear space: shading computations are correct for linear values 
- When encoding linear color values for display, the goal is to cancel out the effect of display transfer function: inverse of display transfer function is applied to color values – **gamma correction**
  - Standard transfer function for personal computer display is called **sRGB**
  - : texture → decoding → shading → encoding → display (decoding)>

\* also optical-electrical transfer function describe the other end of the process for image and video capture devices

# Gamma correction

- Gamma correction can be seen as last step in rendering and post-processing – when everything is computed and image is ready for display.
  - To encode rendered image which is in linear colorspace (x) for sRGB display (y) the following formula is applied to all color values:
    - $y = x^{1/\gamma}$ , where  $\gamma = 2.2$
- Everything we see on the screen (e.g., image textures or color-pickers in modeling tools) is display-encoded data and those values must be decoded to linear values for rendering.
  - To decode sRGB display encoded values (y) to linear colorspace (x) the following formula is applied to all color values:
    - $x = y^{\gamma}$ , where  $\gamma = 2.2$

# High dynamic range display encoding

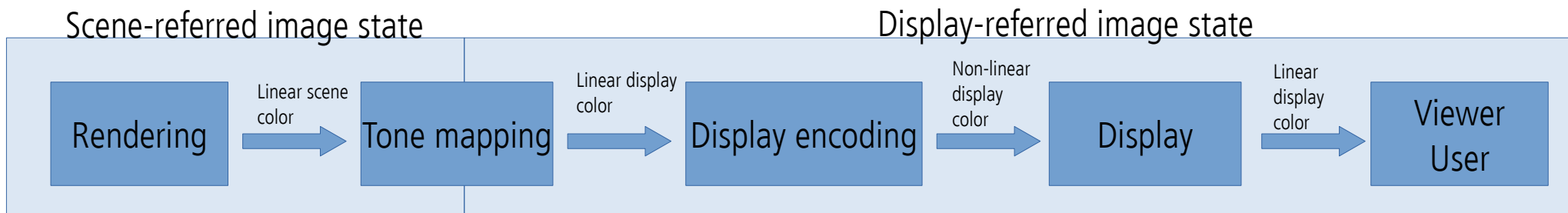
- By now we have discussed standard dynamic range (SDR):
  - SDR Monitors use sRGB display standard
  - SDR Televisions use Rec. 709 and Rec. 1886 display standards
- Both SDR monitors and televisions have same RGB **gamut** and **white point** (D65), also similar non-linear encoding curves and luminance levels
- HDR displays use Rec. 2020 and Rec. 2100 standards
  - Rec. 2020 has significantly wider color gamut and same white point (D65) as Rec. 709 and sRGB color spaces
  - Rec. 2100 defines two non-linear display encodings: perceptual quantizer (PQ) and hybrid-log gamma (HLG)
- <image comparing gamuts>
- Therefore, different methods must be used for transferring rendered images in linear colorspace to HDR display:
  - HDR10
  - scRGB
  - Dolby Vision

# Tone mapping

- Reminder: **Rendering** is process of calculating image color based on 3D scene description.
  - The final results is – pixel values in display buffer – still needs to be determined.
- Reminder: **Display encoding** is process in which linear color (radiance) values are converted to nonlinear values for display hardware.
  - Inverse of display EOTF is applied so that linear values in image buffer match the linear radiance emitted by the display
- Between rendering and display encoding there is **tone mapping step**.
- But let's see the big picture of **imaging pipeline** first.

# Imaging pipeline

- Tone mapping is step between **image states**\*:
  - **Scene-referred**
    - Defined in reference to scene color (radiance) values.
    - Physically-based rendering computations are correct for this state
  - **Display-referred**
    - Defined in reference to display color (radiance) values.



\* Display limitations require non-linear transform between two states



# Tone mapping

- Tone mapping (end-to-end or scene-to-screen transfer) is about converting scene color values to display radiance values
  - Goal 1: **image reproduction** - create display-referred image that reproduces, as closely as possible given display and viewing properties, perceptual impression that viewer would have if they were observing original scene.
  - Goal 2: **preferred image reproduction** – create display-referred image that looks better (to some criteria) than original scene.

# Tone mapping: image reproduction

- Reproduce a similar perceptual impression as the original scene, problems:
  - Original scene **luminance** exceeds display capabilities
  - **Saturation** (purity) of original scene also exceeds display capabilities
- **Exposure** is another term critical to image reproduction
  - Exposure in photography refers to controlling the amount of light falling on film/sensor.
  - In rendering, exposure is a linear scaling operation performed on scene-referred image before tone mapping is applied.

# Tone mapping: image reproduction

- <image: image with and without image reproduction tone mapping>

# Tone mapping: image reproduction

- **Global tone mapping:**
  - Scaling by exposure
  - Then applying tone reproduction
- Local tone mapping:
  - Different mapping pixel-to-pixel based on surrounding pixels and other factors

# Global tone mapping

- Tone reproduction
  - TODO - RTR 8.2
- Exposure
  - TODO - RTR 8.2

# Tone mapping: preferred image reproduction

- Creative manipulation of image colors to obtain image desired artistic “look” - is called **color grading**.
- **Grading look up tables** (LUT) contain desired color transformations which are applied on image
  - e.g., baked color curves
- Color grading is possible on:
  - Scene-referred images: produce hi-fi results
  - Display-referred images: easier to set-up

# Color grading

- <example of color graded images>

# More into topic

- <https://developer.nvidia.com/sites/default/files/akamai/gameworks/hdr/UHDCOLORforGames.pdf>
- Color grading:  
<http://filmicworlds.com/blog/minimal-color-grading-tools/>
- <https://cinematiccolor.org/>
-