Lecture 13: Imaging pipeline and post-processing

DHBW, Computer Graphics

Lovro Bosnar

29.3.2023.

Syllabus

- 3D scene
  - Object
  - Light
  - Camera
- Rendering
- Image and display

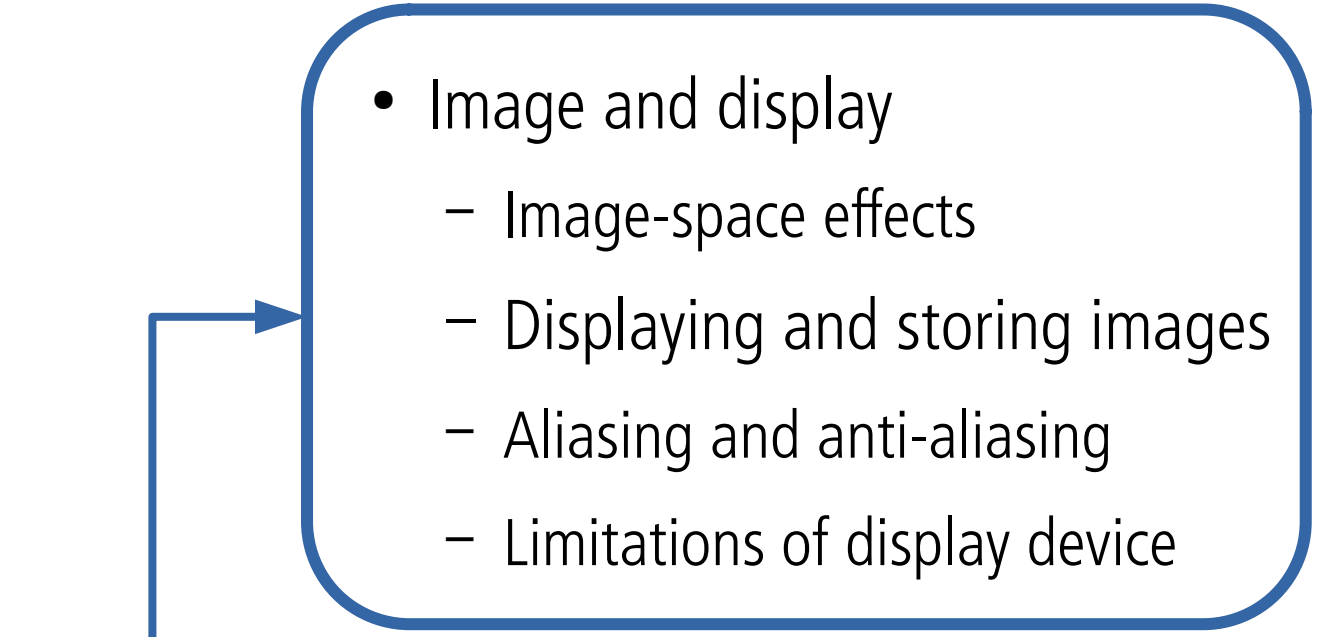
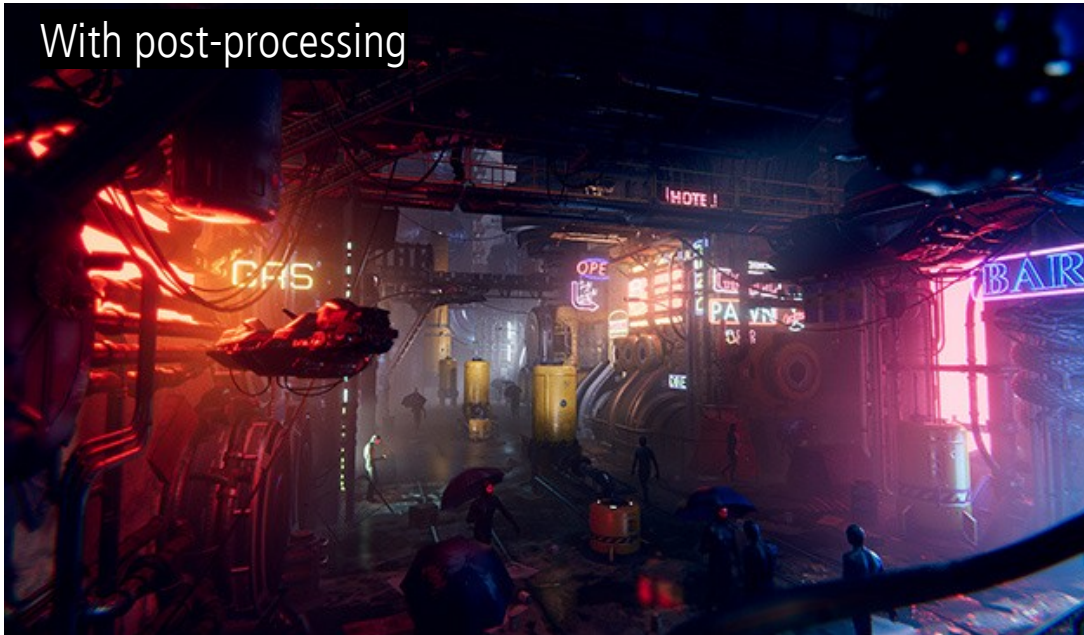


Image-space effects

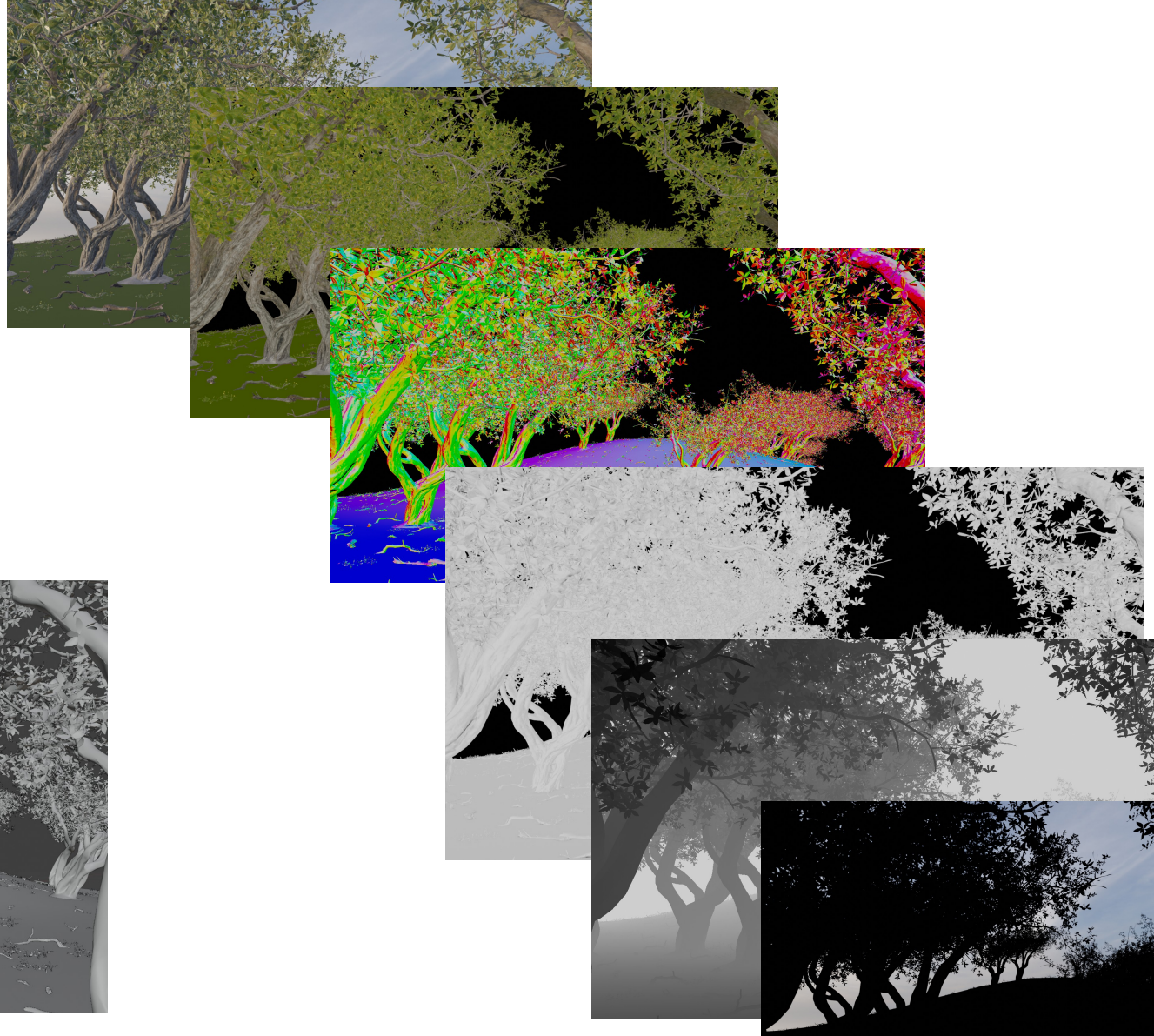
Image-space effects

- Rendered image can be directly sent to display or additionally modified before display → **post-processing**
- Post-processing is set of operations done on rendered image giving additional possibilities for realistic or artistic visuals → **image-space effects**



Post-processing data

- Post-processing of one frame can use multiple buffers (images) obtained during rendering:
  - Color, diffuse, normal, AO, depth, environment



Performing post-processing

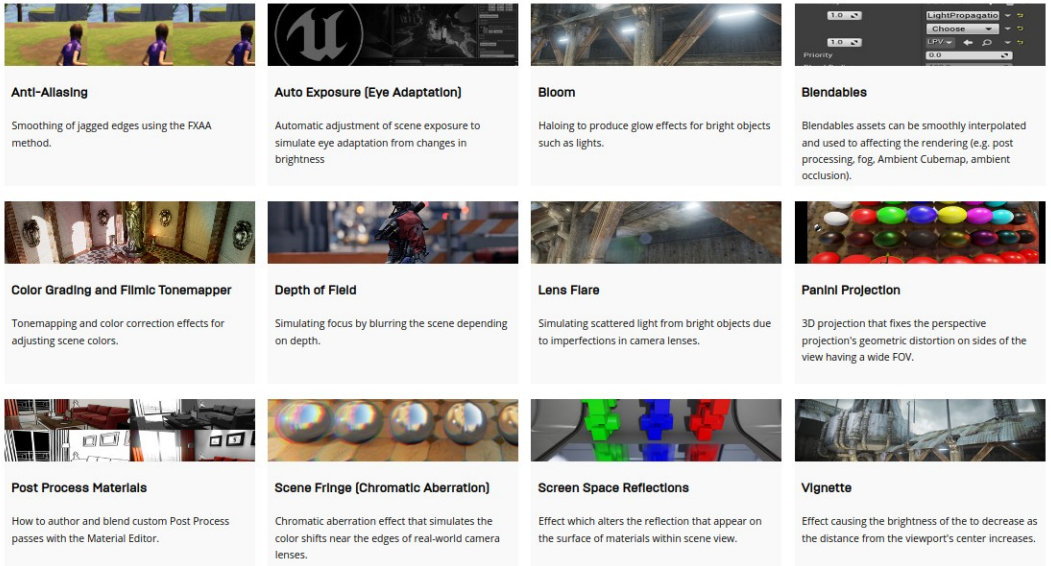
- Depending on rendering environment, post-processing of frame can be performed:
  - Real-time, multi-pass, render-time
    - **3D scene is rendered to an offscreen buffer**, such as color (image) buffer
    - Resulting image is treated as image texture which is **applied on screen filling quad**
    - **Post-processing** is done on this texture **using programmable GPU shaders** (e.g., fragment or compute shaders)
  - Offline, compositing software
    - 3D scene is rendered to image file
    - Post-processing is done on images which are stored as separate frames


Render-time, interactive post-processing

Offline post-processing

Post-processing in practice

- Real-time, multi-pass:
  - Unity: https://docs.unity3d.com/Manual/PostProcessingOverview.html
  - Godot: https://docs.godotengine.org/en/stable/tutorials/shaders/custom_postprocessing.html
  - Unreal: https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/PostProcessEffects/
  - OpenGL: https://learnopengl.com/In-Practice/2D-Game/Postprocessing
- Offline:
  - Blender: https://docs.blender.org/manual/en/latest/compositing/introduction.html
  - Nuke: https://www.foundry.com/products/nuke-family/nuke
  - After Effects: https://www.adobe.com/products/aftereffects.html
  - Discussion: https://www.youtube.com/watch?v=7g4xCV0iv4w&ab_channel=InspirationTuts

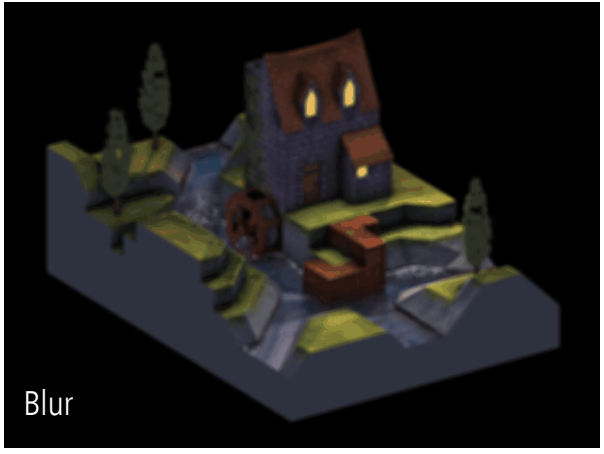


Unreal post-process

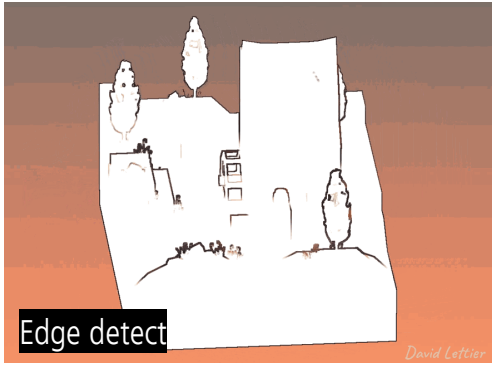
Nuke post-process

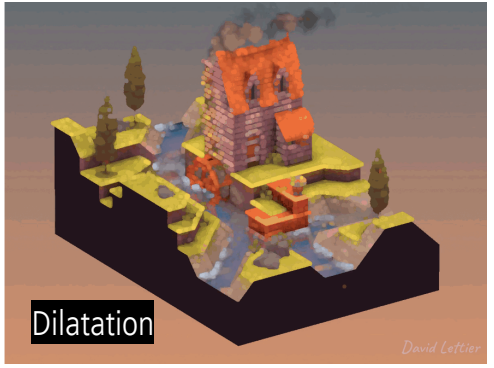
Post-processig via Image processing

- Rendered image can be post-processed as any other image using image processing techniques (filtering kernels) in shader or CPU.



https://lettier.github.io/3d-game-shaders-for-beginners/posterization.html





Post-processing and NPR

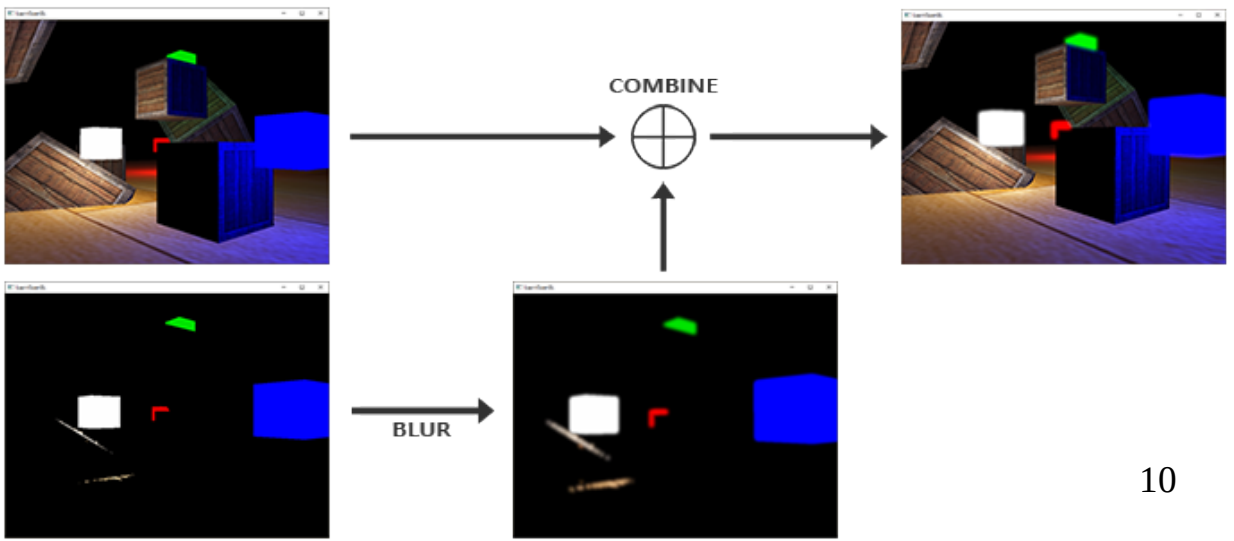
- Often image processing techniques are used to achieve non-photo realistic rendering effects

Edge detection (Borderlands)

Edge detection and quantization (Okami)

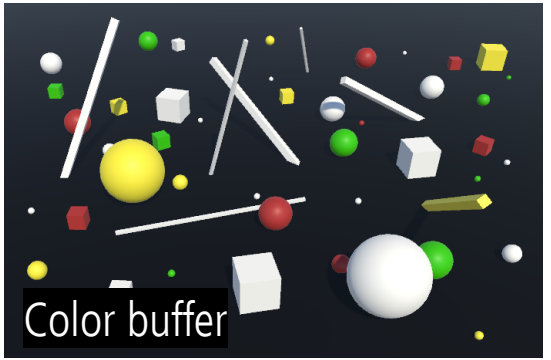
Bloom

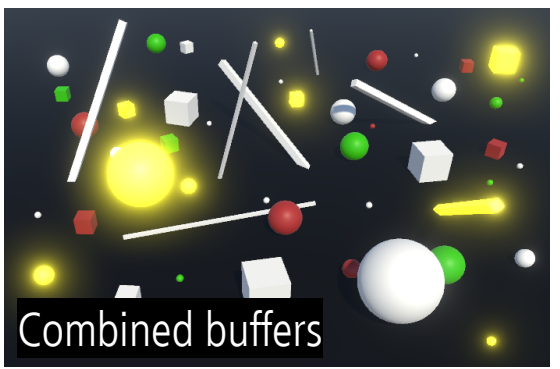
- Effect where extremely bright area spills over adjoining pixels → making image look overexposed where it is bright
- Concept:
  - Render whole scene in color buffer
  - Render only bright objects into separate buffer
  - Blur buffer with bright objects only (e.g., Gaussian Blur)
  - Combine buffers



https://learnopengl.com/Advanced-Lighting/Bloom

Bloom





https://catlikecoding.com/unity/tutorials/advanced-rendering/bloom/

Lens flare

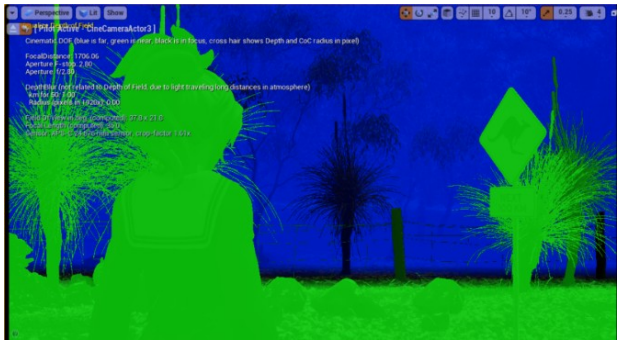
- Flare phenomena: halo and ciliary corona are caused by light traveling through lens system
- Dependent on light source position
- Various approaches, e.g., light streaks from bright object can be simulated using steerable filter

Lens flare (Battlefield)

Lens flare simulation https://resources.mpi-inf.mpg.de/lensflareRendering/pdf/flare.pdf

Depth of field

- Depth of field: range where objects are in focus. Outside objects are blurred
- Amount of blur is related to aperture size: smaller aperture size → increased depth of field
  - Pinhole cameras: infinite depth of field
- One solution: create separate image layers where near and far layers are blurred and combined with layer in focus using back to front compositing
- Various methods exist



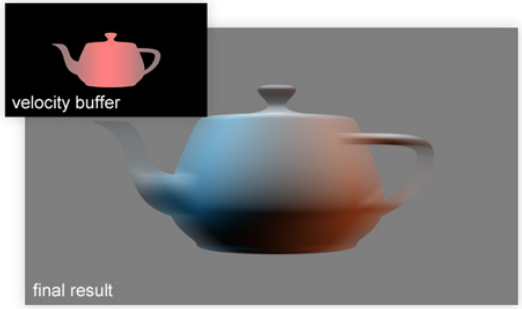
https://www.versluis.com/2022/01/ue4-dof/
https://catlikecoding.com/unity/tutorials/advanced-rendering/depth-of-field/

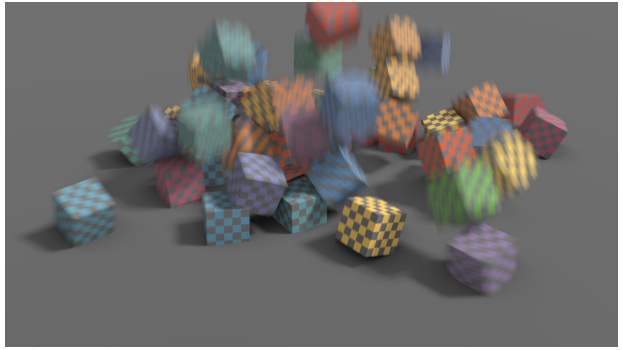
Motion blur

- Motion blur comes from movement of an object across the screen or from camera movement:
- Different sources of motion blur require different methods
  - **Camera orientation changes** → e.g., **circular blur** around view axis if camera is rotating around view axis or **directional blur** based on orientation direction
  - **Camera position changes** → radial blur
  - **Object position/orientation changes** → motion of each object is computed using **depth or velocity buffer buffer** for determining amount of blur

Directional blur due to camera orientation change (Tomb raide

Radial blur due to camera position change (Need for speed)





Object rotation and position change blur using velocity buffer
http://john-chapman-graphics.blogspot.com/2013/01/per-object-motion-blur.html

Displaying and storing images

Digital imagery

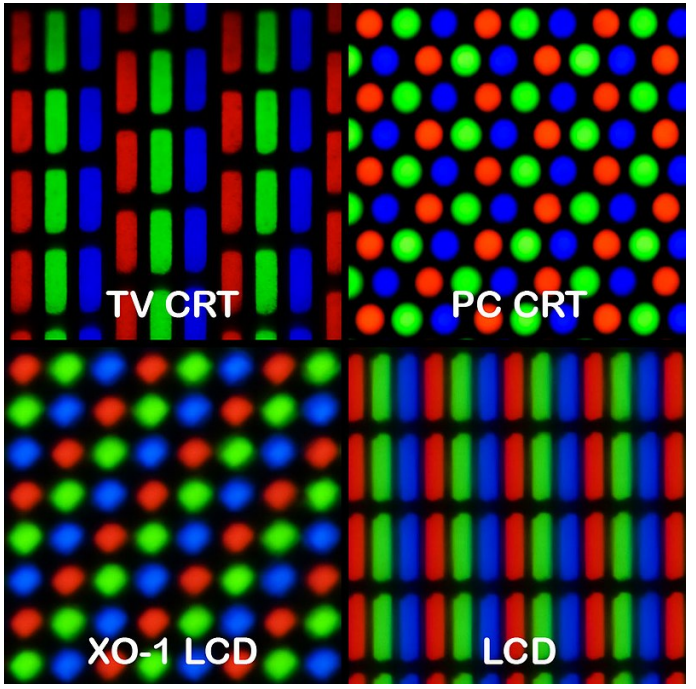
- Digital imagery appears in all forms of media. Most of these are:
  - Digital photos or other types of 2D pictures scanned or loaded in computer
  - **Generated in 3D using modeling and rendering software**

Digital photography https://www.warnerbros.com/movies/blade-runner-2049

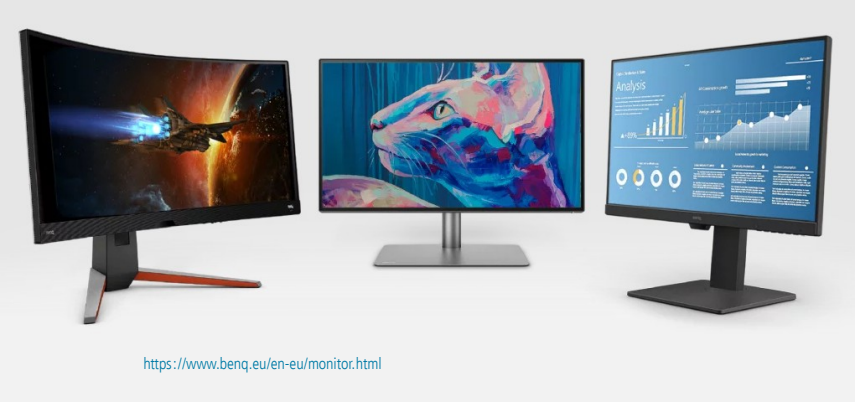
Rendered image: https://www.thewitcher.com/en/witcher3

Display device

- Digital images are shown on a **raster display device → raster images**
  - e.g., monitor, television screen, etc.
- Raster display device is made of **arrays of pixels → discrete representation**
  - For example, each pixel of image created with digital camera (or rendering) stores red, green and blue value used to drive the red, green and blue values of screen pixel.



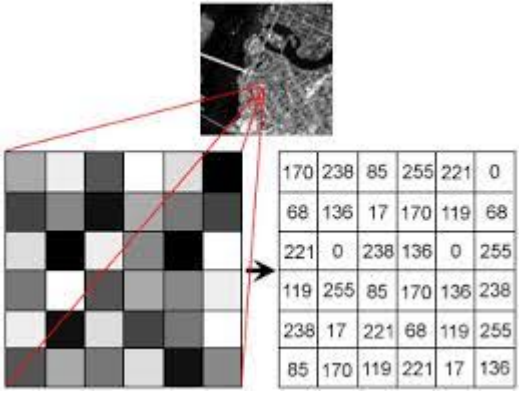
https://en.wikipedia.org/wiki/Pixel_geometry

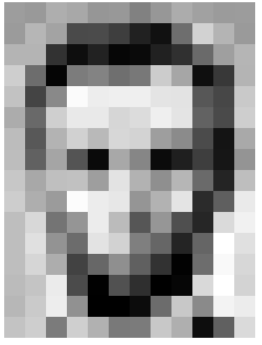


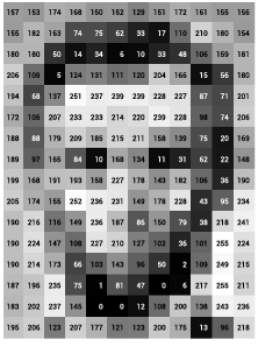
https://www.benq.eu/en-eu/monitor.html

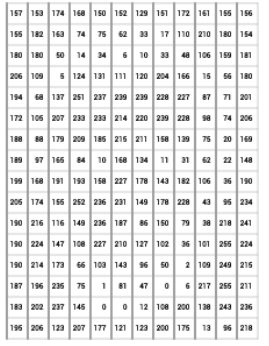
Raster image graphics

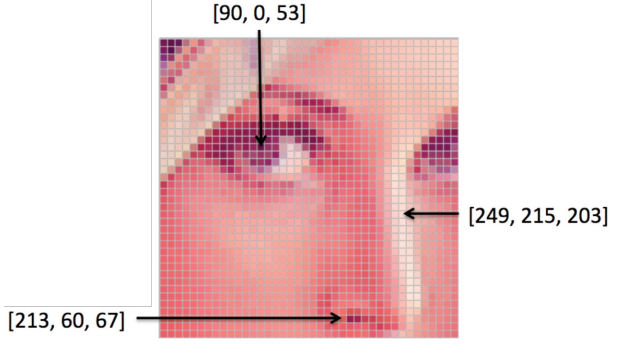
- Raster images are **rectangular array of values** called **pixels.**
- All values in such have same type, examples:
  - Floating point numbers representing levels of **gray**
  - Floating point triplets representing mixture of red, green and blue (**RBG**)
  - Floating point numbers representing **depth** 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**









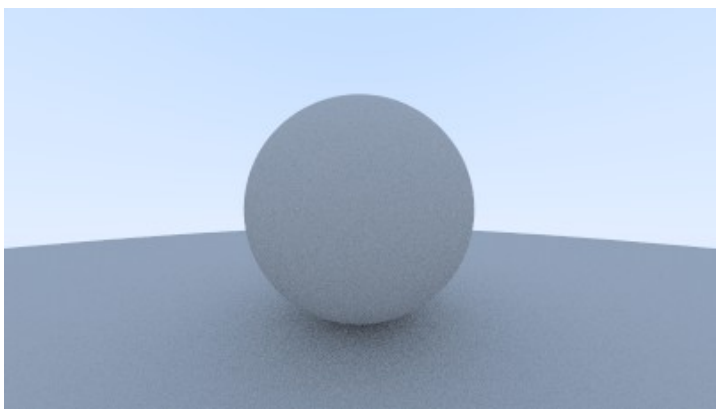


https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

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

```
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
```



https://raytracing.github.io/books/RayTracingInOneWeekend.html

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

Naive image file format: PPM

- Characteristics:
  - 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
```

Raster image file formats

Format	Channel Depth	Alpha	Meta data	DPI	Extensions
BMP	8bit	✗	✗	✓	`.bmp`
Iris	8, 16bit	✓	✗	✗	`.sgi` `.rgb` `.bw`
PNG	8, 16bit	✓	✓	✓	`.png`
JPEG	8bit	✗	✓	✓	`.jpg` `.jpeg`
JPEG 2000	8, 12, 16bit	✓	✗	✗	`.jp2` `.jp2` `.j2c`
Targa	8bit	✓	✗	✗	`.tga`
Cineon & DPX	8, 10, 12, 16bit	✓	✗	✗	`.cin` `.dpx`
OpenEXR	float 16, 32bit	✓	✓	✓	`.exr`
Radiance HDR	float	✓	✗	✗	`.hdr`
TIFF	8, 16bit	✓	✗	✓	`.tif` `.tiff`
WebP	8bit	✓	✓	✓	`.webp`

https://docs.blender.org/manual/en/latest/files/media/image_formats.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 reconstructred.
  - **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 channed (e.g., 8 bit)
  - e.g. information on when and with which program the image was produced

https://www.gimp.org/tutorials/ImageFormats/

File formats in production

- **Digital painting and image manipulation**
  - Krita: bmp, jp2, **jpeg**, ora, pdf, **png**, ppm, raw, **tiff**, xcf https://krita.org/en/item/krita-features/
  - Gimp: XCF, **JPG**, **PNG**, GIF, **TIFF**, Raw, etc. https://www.gimp.org/tutorials/ImageFormats/
- **3D modeling**
  - Blender: BMP, Iris, **PNG**, **JPEG**, Targa, OpenEXR , **TIFF**, HDR, etc. https://docs.blender.org/manual/en/latest/files/media/image_formats.html
  - Houdini: **png**, gif, **jpg**, **tiff**, etc. https://www.sidefx.com/docs/houdini/io/formats/image_formats.html
- **Game engines:**
  - Unity: BMP, **TIF**, TGA, **JPG**, **PNG**, PSD, etc. https://docs.unity3d.com/2019.2/Documentation/Manual/AssetTypes.html
  - Godot: BMP, OpenEXR, **JPEG**, **PNG**, SVG, etc. https://docs.godotengine.org/en/stable/tutorials/assets_pipeline/importing_images.html

Raster image file formats

- Often used raster image file formats are:
  - Tag Image File Format - **JPEG**
  - Portable Network Graphics - **PNG**
  - Tag Image File Format - **TIFF/TIF**
- 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

Cons:

- Lossy
  - Problematic for comparing the image due to differences in compression algorithm
  - Repeated editing degrades image quality
- Artifacts can be seen for computer generated graphics and text
- Doesn't support transparency
- Color channels are coded on 8 bits

Pros:

- Efficient file compression
- Universally supported for display
- Most digital cameras produce JPEG images

- Recommended for display and storage of photography

http://www.faqs.org/faqs/jpeg-faq/part1/

PNG

Pros:

- Lossless format
- Supports transparency
- Small file size for most computer graphics generated images
- Support by all browsers
- PNG file format is more compact than PPM and equally easy to use.
- Support for RGB(A) and grayscale images

Cons:

- Complex images are heavier
- Color channels coded in 8 bits

- Developed as an improved replacement for Graphics Interchange Format (GIF)
- Recommended for web page widgets, computer graphics and screenshots

TIFF

Pros:

- Color coding in 16 bits
- Supported by all image processing software
- TIFF images store multiple channels (images/layers)
  - Each channel store description of contents
- Various compression schemes
  - uncompressed
  - Compressed with lossless scheme
  - Compressed with lossy scheme

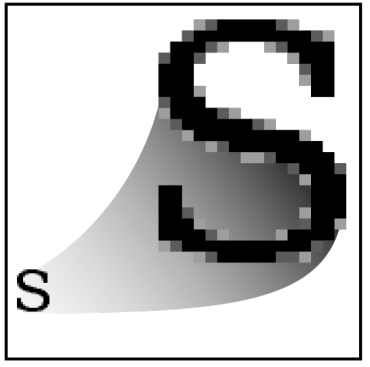
Cons:

- Heavier on complex images

- Storage and exchange of high quality images
- 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

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



Raster
GIF, JPEG, PNG

Vector
SVG

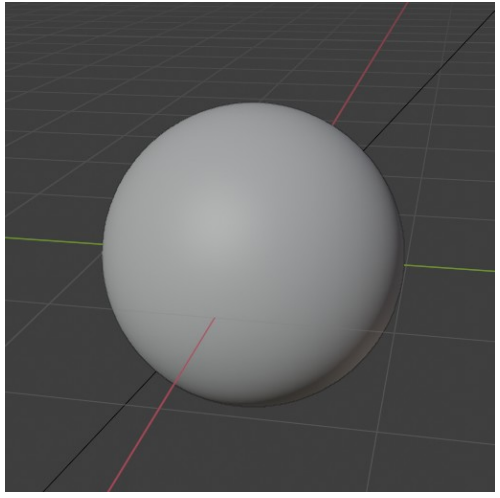
https://en.wikipedia.org/wiki/Vector_graphics

Aliasing and anti-aliasing

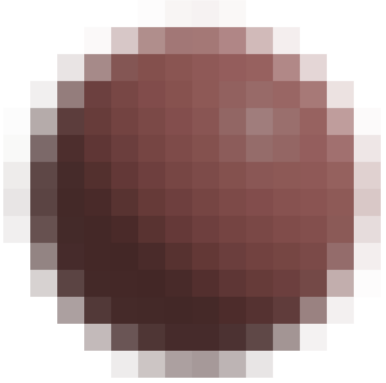
When continuous becomes discrete

Continuous and discrete

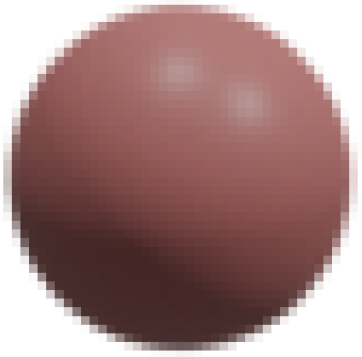
- Images are **discrete** array of pixels
- 3D scene objects, lights and thus color values are **continuously** changing
- Rendering process, in order to create a discrete image of 3D scene, **samples the pixels on image plane**, finds corresponding sample in 3D scene and calculates color of the sample.



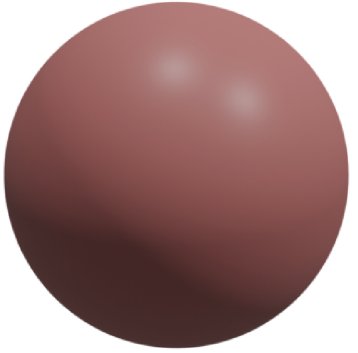
Continuous object in 3D scene



Render: 20x20px image size



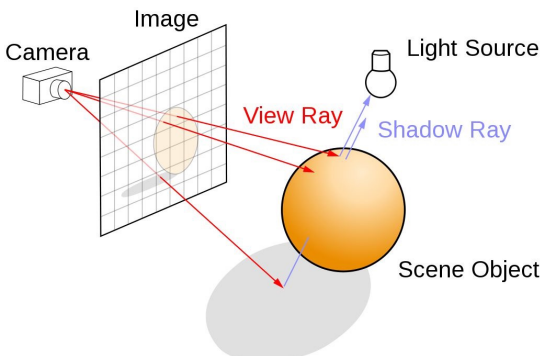
Render: 64x64px image size

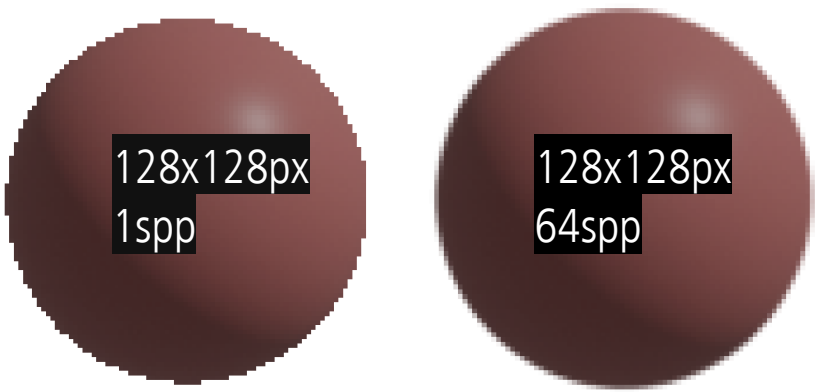


Render: 512x512px image size

Scene sampling and Image buffer

- Process of rendering images is inherently a sampling task
- Rendering is the process of creating sampling a 3D scene in order to obtain colors for each pixel in the image
  - Using camera information, samples per image pixel are generated
  - For each sample, corresponding 3D scene point is found and color is calculated
  - Color of each pixel for each sample is combined into **image buffer**

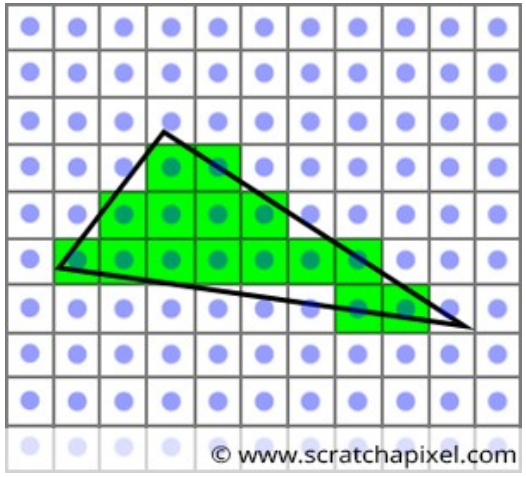




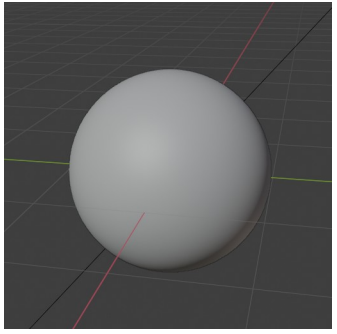
Note that pixel can have only one color!
Pixels overlapping multiple objects/textures in 3D scene must be sampled with multiple rays to obtain correct representation.

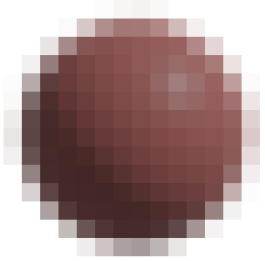
Aliasing

- Discretizing 3D scene (continuous information) may cause **Aliasing**
  - Jagged edges, flickering highlights (firelfys), Moiré pattern

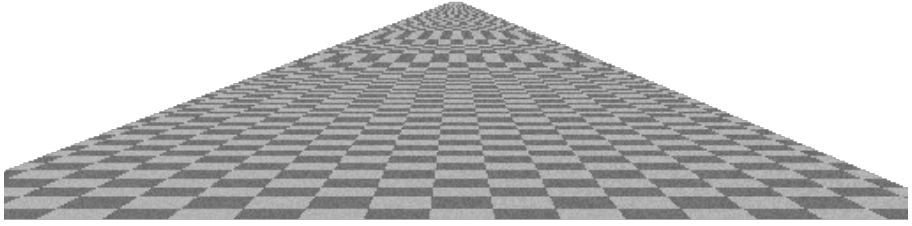


Discretization





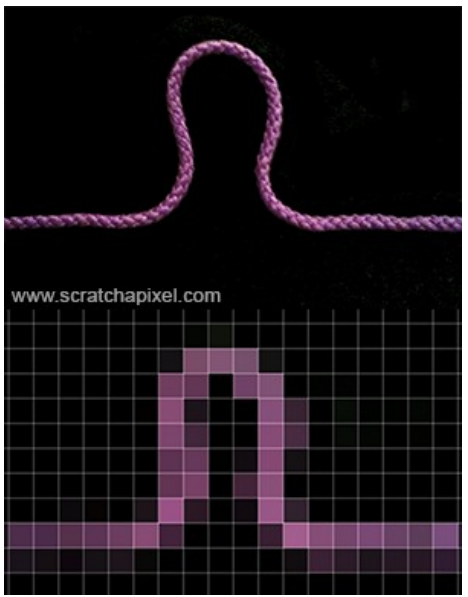
Problem: Jagged edges



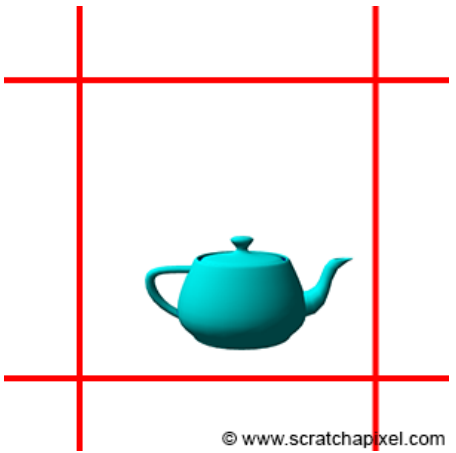
Problem: Moiré pattern

Aliasing: intuition

- Projecting pixel into 3D scene can cover larger portion of 3D scene which contain multiple objects and thus color variation.
- Any kind of rapid (high-frequency) change in geometry edges, textures, shadows, highlights can cause aliasing.



Thread contains large amount of details covered by only one pixel



Extreme example: one whole object in pixel footprint.

Changes in geometry, textures, highlights → sources of aliasing.

Aliasing

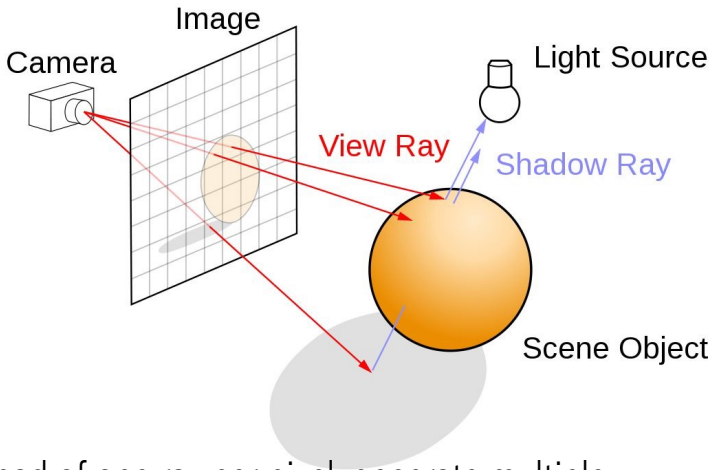
Sampling process might result in **aliasing** – an unwanted artifact in spatial or temporal domain (spatial vs temporal sampling)

- 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: frequrency 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

Anti-aliasing

- Different anti-aliasing methods:
  - Anti-aliasing can be applied on: geometry, textures, shadows, highlights, etc.
  - Trade-off between: quality, ability to capture sharp details or other phenomena, appearance, memory, speed and computing power
- Most common anti-aliasing method: **screen-based anti-aliasing**
  - **Multiple samples per pixel**

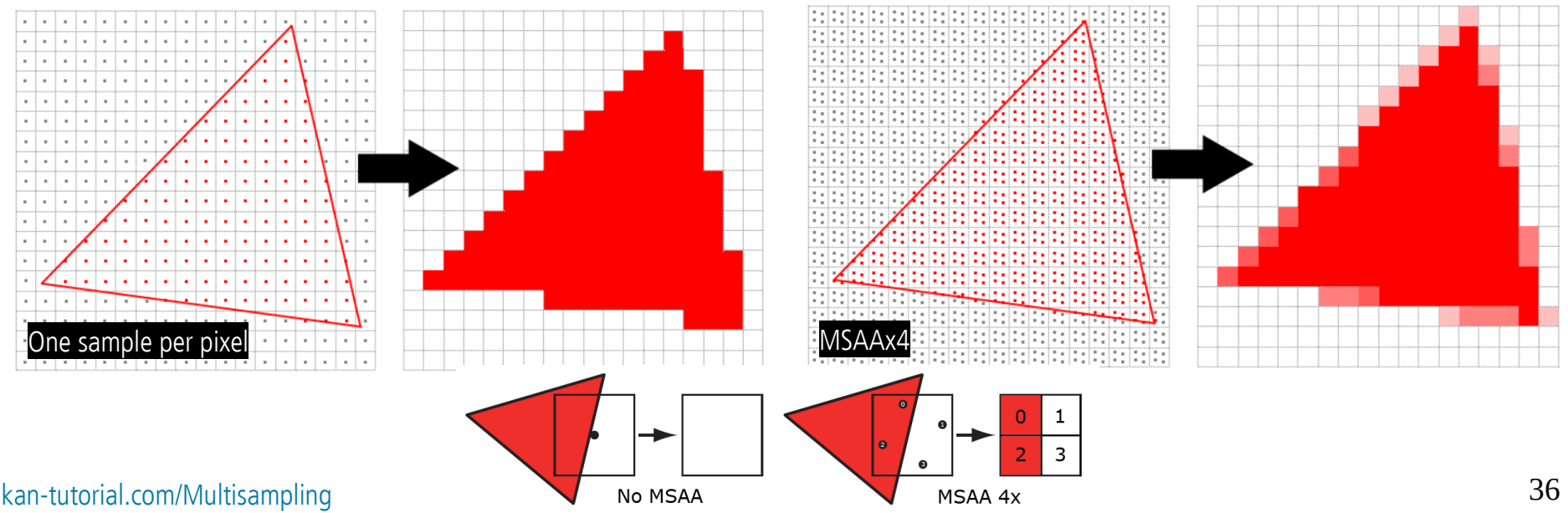


Instead of one ray per pixel, generate multiple rays - samples

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

Multisample anti-aliasing

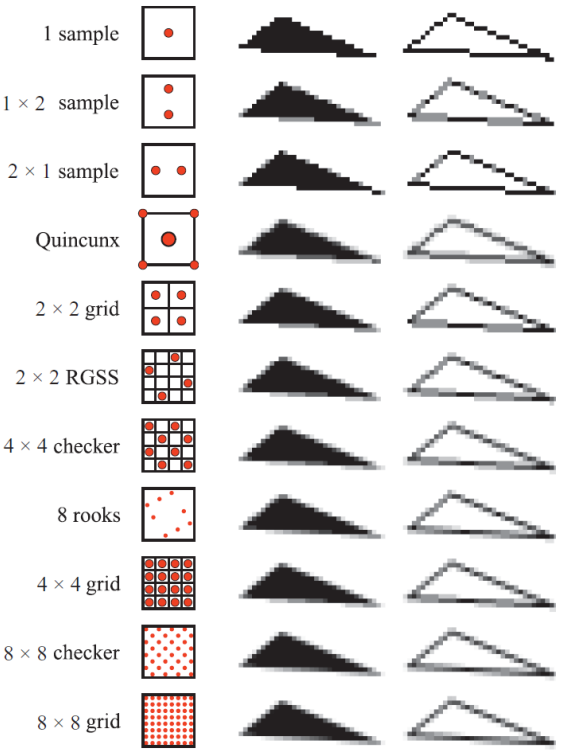
- Aliasing appears always when there are a lot of details under a pixel footprint in the scene
- To solve this, multiple sample per pixels can be used to "catch" high frequency details.
  - This method is called Multisample anti-aliasing (MSAA)



https://vulkan-tutorial.com/Multisampling
https://therealmjp.github.io/posts/msaa-overview/

Antialiasing: sampling schemes

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



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.

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

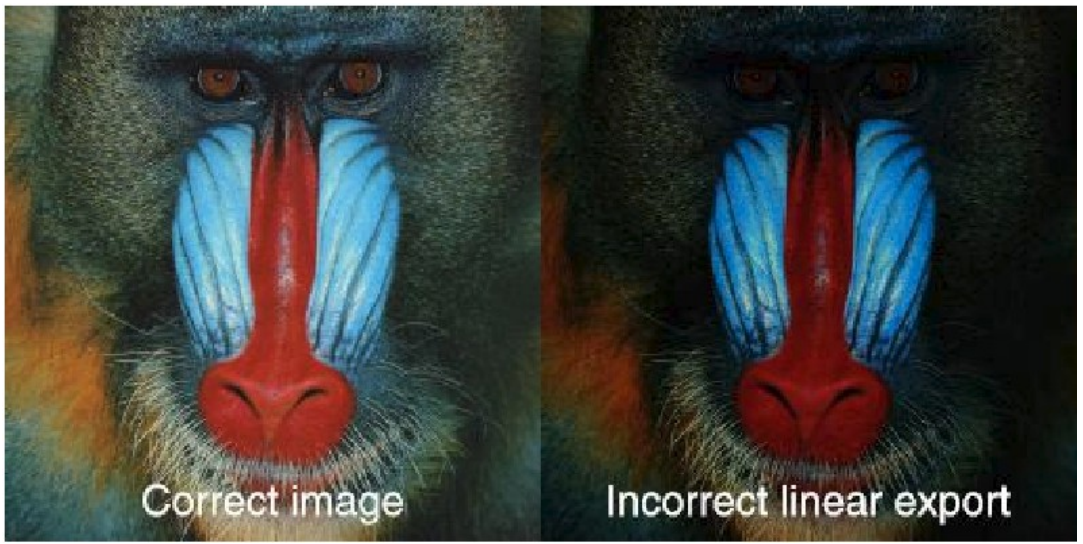
Display limitations

- Display devices have **nonlinear relationship between input voltage and display amount of light**
  - Early display devices: cathode-ray tube (CRT).
    - As energy level [illegible] linearly but according to power law.
    - E.g., Pixel set to [illegible]
  - LCDs and other d[illegible]ent intrinsic tone response curves.
  - Relationship betw[illegible]tted from the display is described with **electrical-op**[illegible]

Shorten this whole part

Problem

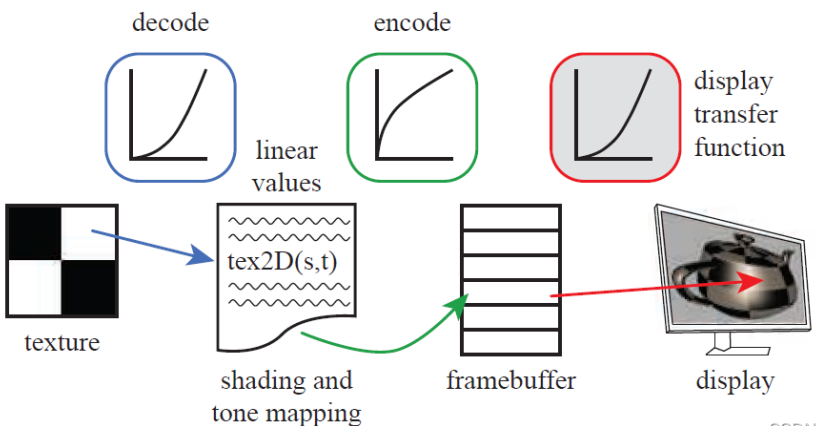
- Rendering calculations are performed using linear color space and thus color buffer will contain linear color space values
- Image containing **linear color space** will not display correctly if shown directly on a display device
  - Linear values will appear too dim on the screen



https://www.sidefx.com/docs/houdini/render/linear.html

Solution: display encoding

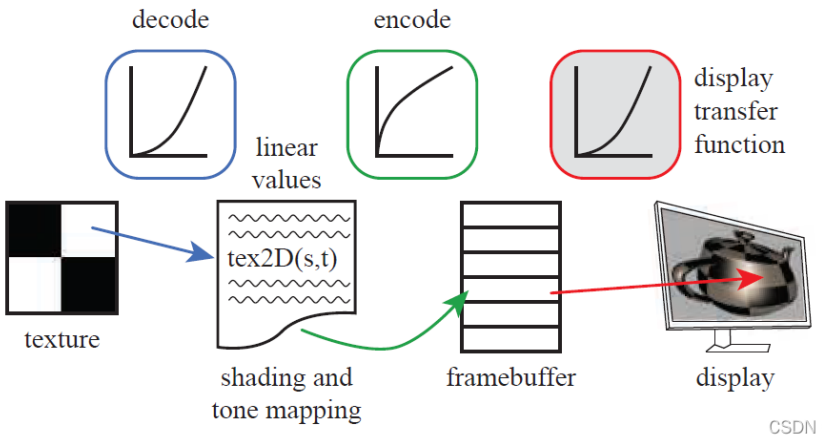
- When encoding linear color values for display, the goal is to cancel out the effect of display transfer function: inverse of display transfer function (EOTF) is applied to color values in image buffer – **gamma correction**
  - Standard transfer function for personal computer display is called **sRGB**



https://www.realtimerendering.com/

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`



Modeling and rendering requirements

- 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
  - Working with linear color gives more accurate arithmetical results
  - Makes color faster and easier for the computer to deal with.
  - Doubling the numeric values of a color in linear color space doubles the intensity and has no loss of precision

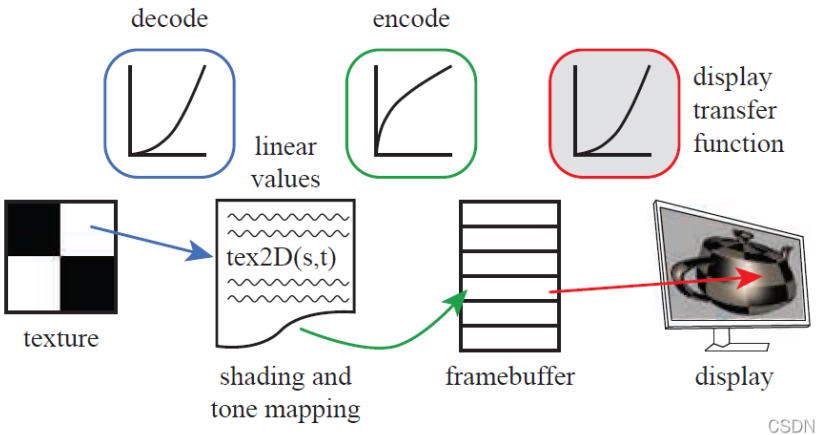
https://www.sidefx.com/docs/houdini/render/linear.html

Problem

- When working in modeling tools users pick texture images or colors on screen - those colors are encoded for display device so we can see them properly
  - Those colors can not be used in rendering computation directly because they are in non-linear space

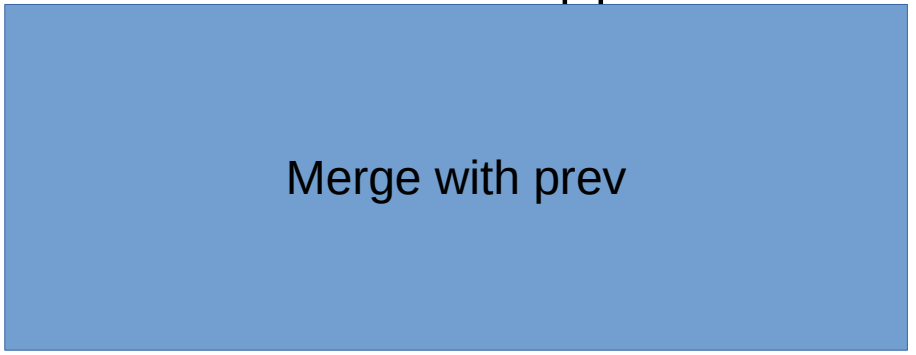
Solution: inverse gamma correction

- 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`



Advanced display encoding

- We discussed display encoding for standard dynamic range monitors (SDR) which use sRGB display standard
- High dynamic range (HDR) displays and different display devices used other display standards which must be applied accordingly.



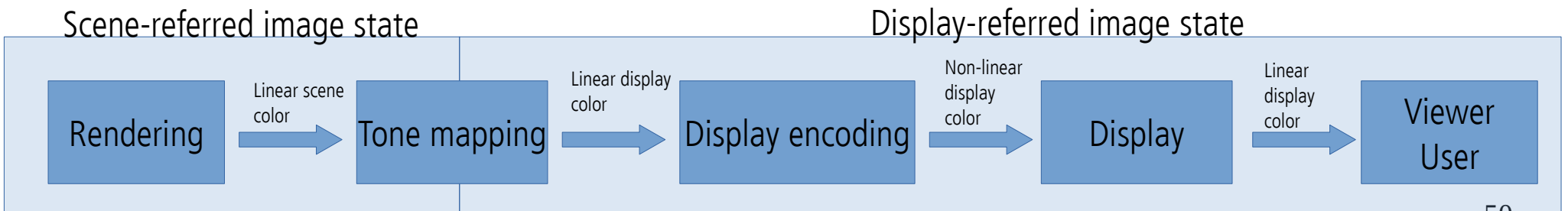
Tone mapping

- **Rendering:** calculating image color based on 3D scene description.
  - The results are pixel values in display buffer – display results still needs to be determined.
- **Display encoding** is process in which linear color (radiance) values are converted to nonlinear values for display hardware.
- Between rendering and display encodin
- **Imaging pipeline** describes color handlin play.

Merge with next

Imaging pipeline

- Tone mapping is step between following **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 color (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: (1) 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 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.

https://learnopengl.com/Advanced-Lighting/HDR

Image reproduction: exposure

Different exposures of the same render:

https://docs.blender.org/manual/en/latest/render/color_management.html

Image reproduction

- Process of transforming floating point color values to the expected [0,1] range for LDR without loosing too much detail
- **Global tone mapping:**
  - Scale by exposure
  - Applying tone reproduction transform
- Local tone mapping:
  - Different mapping pixel-to-pixel based on surrounding pixels and other factors

Global tone mapping: Exposure scaling

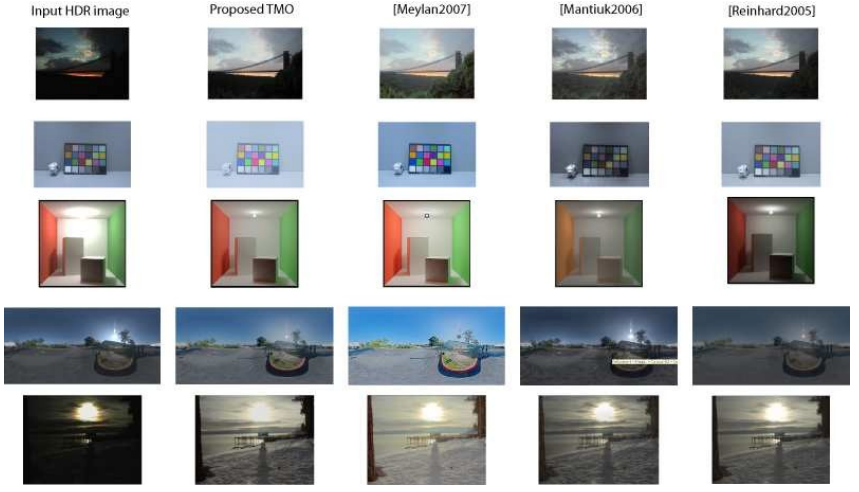
- Commonly used method from computing exposure is to compute average of pixel brightness of rendered image – **Reinhard method**

Merge with next

Global tone mapping: tone reproduction transform

- **Reinhard tone reproduction operator**
  - Early method for tone mapping
  - Leaves dark values unchanged and bright values asymptotically to white
- ACES tone mapping
  - Standard for tone managing for motion picture and television
  - Emerging in real-time applications: Unity and Unreal
- Other tone reproduction operators:
  - Drago, Duiker, Day, etc.

$$V_{\text{out}} = \frac{V_{\text{in}}}{V_{\text{in}} + 1}$$



https://www.researchgate.net/publication/221039985_Spatio-temporal_Tone_Mapping_Operator_Based_on_a_Retina_Model

Tone mapping: (2) preferred image reproduction

- Creative manipulation of image colors to obtain image desired artistic “look” - is called **color grading**.
- **Grading look up tables** (LUTs) 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

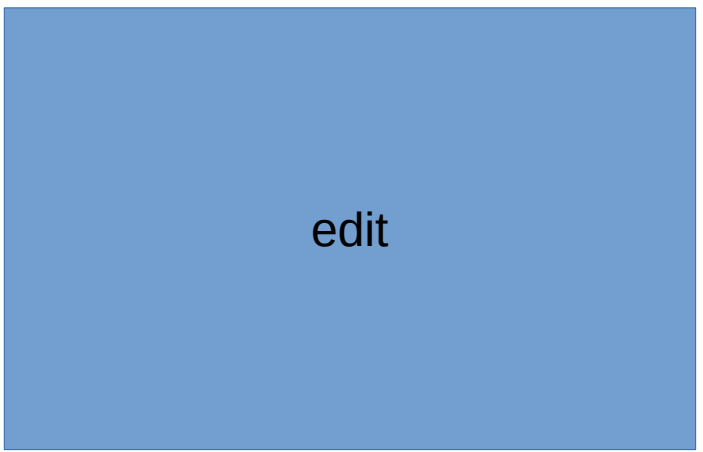
Color grading

https://advances.realtimerendering.com/other/2016/naughty_dog/NaughtyDog_TechArt_Final.pdf

More into topic

- Image space effects:
  - Motion blur: https://www.nvidia.com/docs/io/8230/gdc2003_openglshadertricks.pdf
  - Motion blur: https://developer.nvidia.com/gpugems/gpugems3/part-iv-image-effects/chapter-27-motion-blur-post-processing-effect
  - Motion blur: https://www.bryanmcphail.com/wp/?p=600
  - Motion blur: https://docs.blender.org/manual/en/latest/render/cycles/render_settings/motion_blur.html
  - NPR: https://en.wikipedia.org/wiki/Non-photorealistic_rendering
  - General post-processing: https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@7.1/manual/integration-with-post-processing.html

- https://developer.nvidia.com/sites/default/files/akamai/gameworks/hdr/UHDColorForGames.pdf
- Color grading:
  - http://filmicworlds.com/blog/minimal-color-grading-tools/
  - https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/PostProcessEffects/ColorGrading/
- https://cinematiccolor.org/



Repository

- https://github.com/lorentzo/IntroductionToComputerGraphics