Computer Graphics (CS 543)
Lecture 11b: Tone Mapping, Noise & Procedural Textures

Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

Tone Mapping

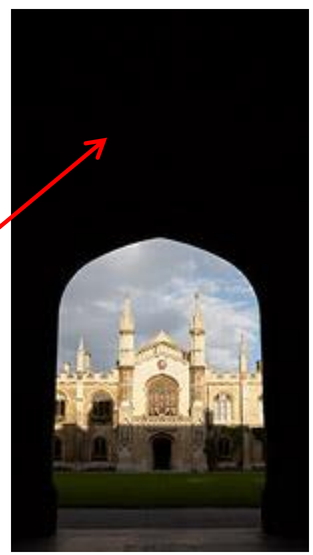
High Dynamic Range

- Sun’s brightness is about 60,000 lumens
- Dark areas of earth has brightness of 0 lumens
- Basically, world around us has range of 0 – 60,000 lumens **(High Dynamic Range)**
- However, monitor has ranges of colors between 0 – 255 **(Low Dynamic Range)**
- New file formats have been created for HDR images (wider ranges). (E.g. OpenEXR file format)

High Dynamic Range

- Some scenes contain **very bright** + **very dark** areas
- Using uniform scaling factor to map actual intensity to displayed pixel intensity means:
  - Either some areas are unexposed, or
  - Some areas of picture are overexposed

Under exposure



Over exposure

Tone Mapping

- Technique for scaling intensities in real world images (e.g HDR images) to fit in displayable range
- Try to capture feeling of real scene: **non-trivial**
- **Example:** If coming out of dark tunnel, lights should seem bright
- **General idea:** apply different scaling factors to diffferent parts of the image

Tone Mapping

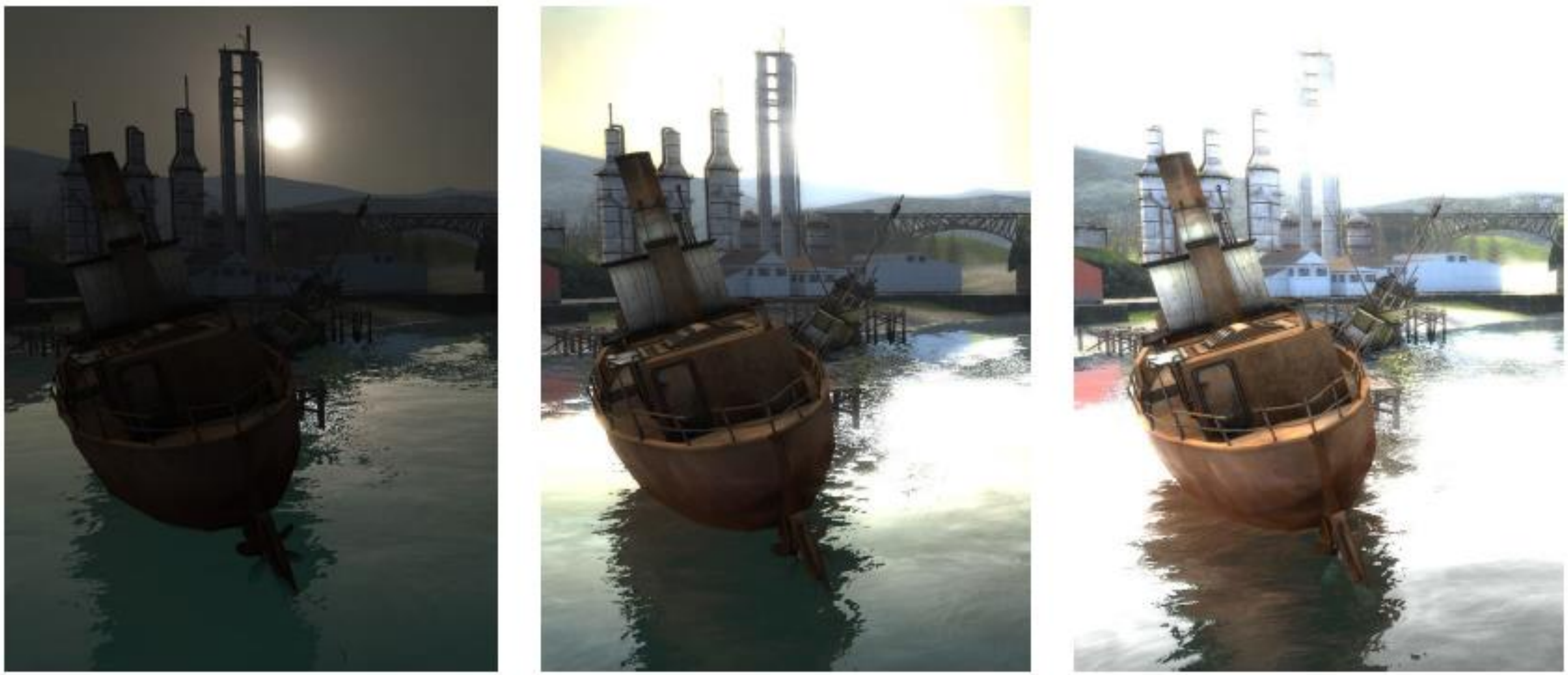


***Figure 10**. Scene from Lost Coast at Varying Exposure Levels*

Types of Tone Mapping Operators

- **Global:** Use same scaling factor for all pixels
- **Local:** Use different scaling factor for different parts of image
- **Time-dependent:** Scaling factor changes over time
- **Time independent:** Scaling factor does NOT change over time
- Real-time rendering usually does **NOT** implement local operators due to their complexity

Simple (Global) Tone Mapping Methods

Maping to mean value

Division by maximum

Clipping on value 1

Interval mapping (interactive calibration)

Exponential mapping

Motion Blur

- Motion blur caused by exposing film to moving objects
- Motion blur: Blurring of samples taken over time (temporal)
- Makes fast moving scenes appear less jerky
- 30 fps + motion blur better than 60 fps + no motion blur

Motion Blur

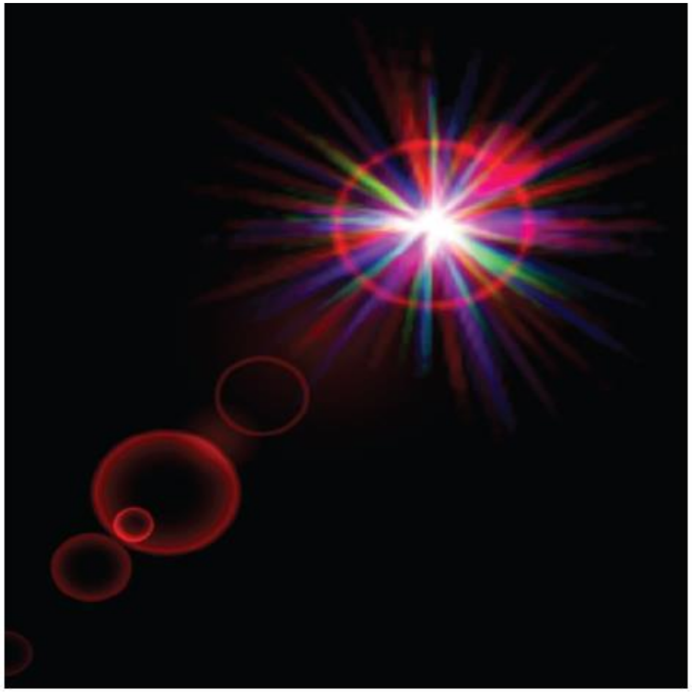
- Basic idea is to average series of images over time
- Move object to set of positions occupied in a frame, blend resulting images together
- Can blur moving average of frames. E.g blur 8 images
- **Velocity buffer:** blur in screen space using velocity of objects

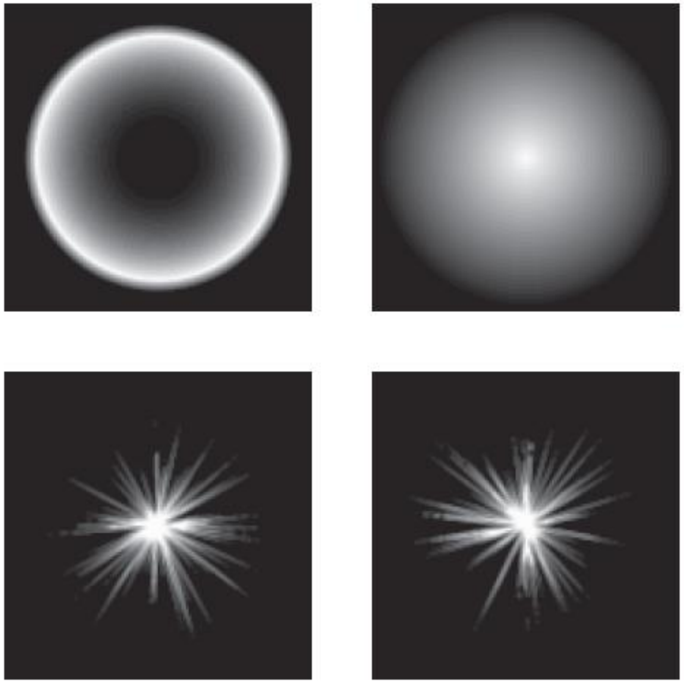
Depth of Field

- We can simulate a real camera
- In photographs, a range of pixels in focus
- Pixels outside this range are out of focus
- This effect is known as **Depth of field**

Lens Flare and Bloom

- Caused by lens of eye/camera when directed at light
- Halo – refraction of light by lens
- Ciliary Corona – Density fluctuations of lens
- Bloom – Scattering in lens, glow around light





Halo, Bloom, Ciliary Corona – top to bottom

3D and Noise Textures

Solid 3D Texture

Ref: Computer Graphics using OpenGL (Third edition) by Hill and Kelley, pg 648-656

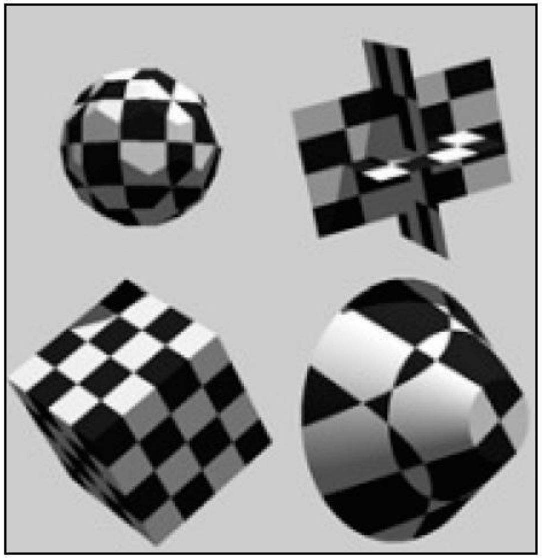
- Sometimes called 3D texture
- As if object is carved out of textured material. E.g. Wood, marble
- Texture: Each (x,y,z) point maps to (r,g,b) color
  - *f(x,y,z) -> (r,g,b)*

Checkerboard Texture

- Imagine cubes of alternating color, each of dimension (S.x, S.y, S.z) placed next to each other
- A 3D texture for a checkerboard pattern can be written as:

  $jump(x, y, z) = [(int)(x/S.x) + (int)(y/S.y) + (int)(z/S.z)] \% 2$

- 3D texture lookup returns color 1 if *jump = 0* and color 2 if *jump = 1*



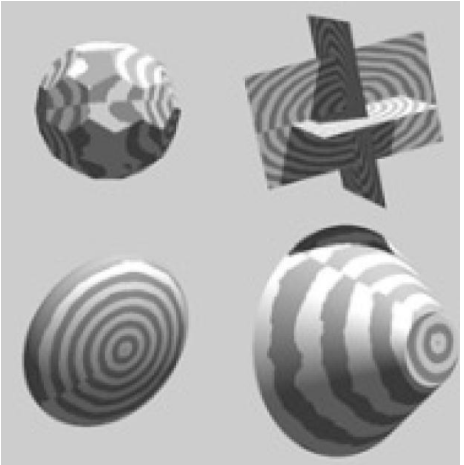
Wood Texture

- Grain in log of wood due to concentric rings varying color
- As distance from some axis increases, functions jumps back and forth between 2 values
- This effect can be simulated with the modulo function

rings(r) = ((*int*) *r*) % 2

where

$$r = \sqrt{x^2 + y^2}$$



- Rings jumps between 0 and 1 as *r* increases from 0.
- The following texture jumps between *D* and *D + A*

simple_wood(x, y, z) = D + A * rings(r/M));

- Produces rings of thickness M that are concentric about z axis

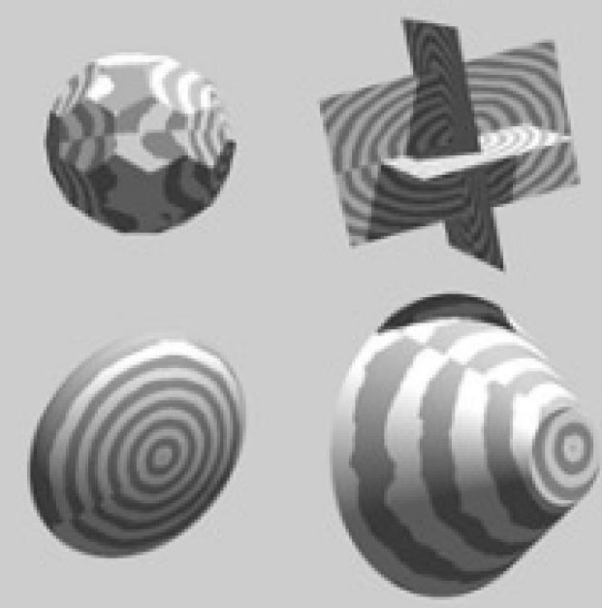
Wood Texture (Contd)

- Can wobble rings by adding component that varies azimuth θ about the z axis

rings(r/M + Ksin(θ/N))

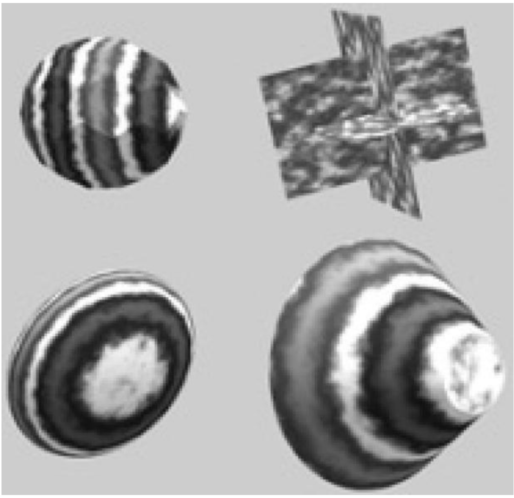
- To add a twist to the wobbling grain:

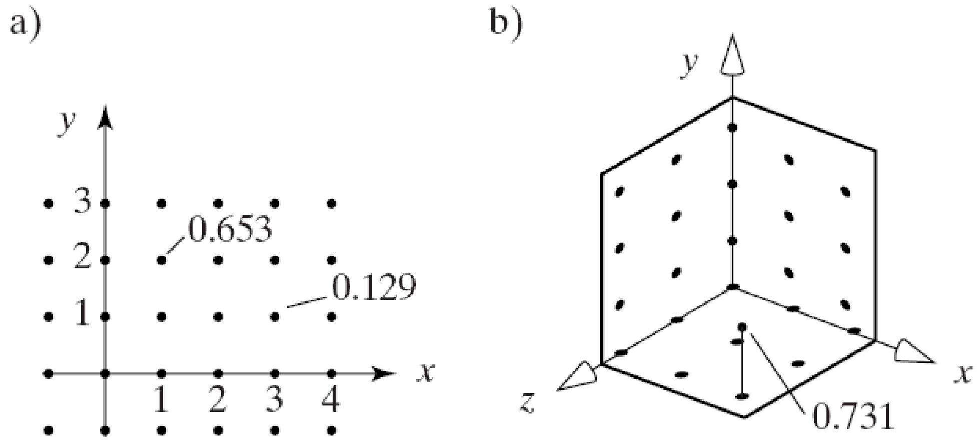
rings(r/M + Ksin(θ/N + Bz))



Marble

- Grain of marble is quite chaotic
- Marble can be simulated by function that produces a “random value” at each (x,y,z) point in space
- Imagine each (x,y,z) point assigned with a random value. E.g. (2,2,1) = 0.7341
- Random values could be stored in massive lookup table. Typically generated on the fly





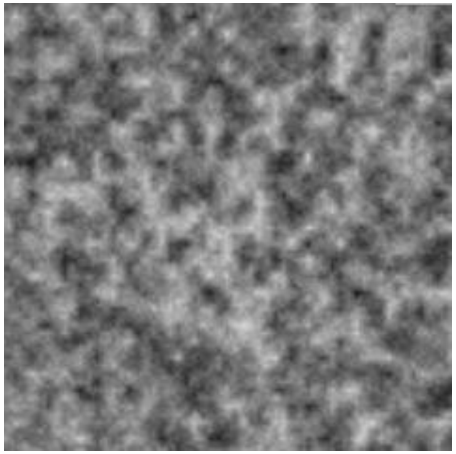
Turbulence

- More interesting noise created by mixing noise functions fluctuating at different rates (1x, 2x, 4x, etc)
- More rapidly fluctuating components given progressively smaller strengths

$$turb(s,x,y,z) = \frac{1}{2}noise(s,x,y,z) + \frac{1}{4}noise(2s,x,y,z) + \frac{1}{8}noise(4s,x,y,z)$$

- Can add more terms, summing *k* terms. Example below for M=3

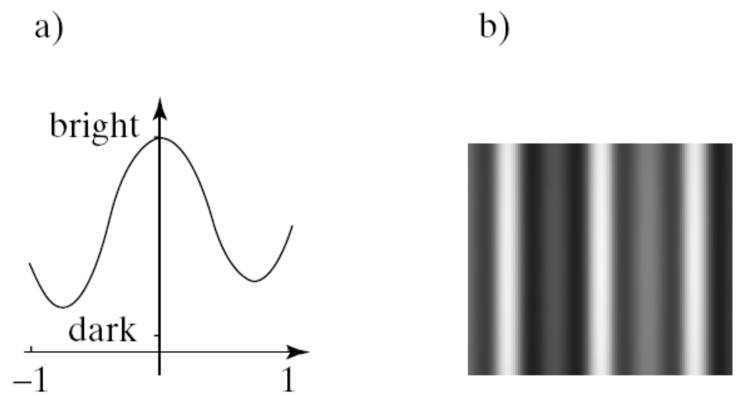
$$turb(s,x,y,z) = \frac{1}{2}\sum_{k=0}^{M}\frac{1}{2^k}noise(2^k,s,x,y,z)$$



Marble Texture

- General idea:
  - give the marble's veins smoothly fluctuating behavior (e.g. in z direction)
  - Perturb the veins using turb( ) function
- For instance, start with texture that is constant in *x* and *y,* smoothly varying in *z*

marble(x, y, z) = undulate(sin(z));



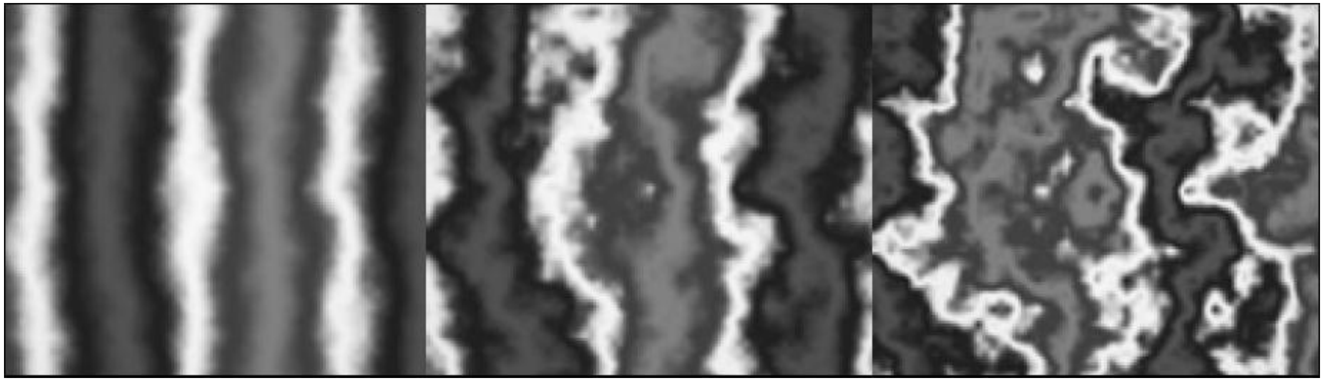
- Above function is too regular
- Modulate sin( ) argument with some turbulence

marble(x, y, z) = undulate(sin(z + A *turb(*s, *x, y, z)));*

Marble Texture (Contd)

marble(*x*, *y*, *z*) = *undulate*(*sin*(*z* + A *turb*(s, *x*, *y*, *z*)));

- Parameter *s* makes turbulence vary more or less rapidly at different points
- Parameter *A* changes amount of perturbation
- Example: $g = spline(\sin(2\pi z + A \times turb(5, x, y, z)))$



A = 1 **A = 3** **A = 6**

References

- Interactive Computer Graphics (6th edition), Angel and Shreiner
- Computer Graphics using OpenGL (3rd edition), Hill and Kelley
- Real Time Rendering by Akenine-Moller, Haines and Hoffman