Computer Graphics

Part I

entropia/CCC Karlsruhe

Two Parts

today:

- Color Perception
- Pixels
- Fractals
- Geometry and Material
- Rendering

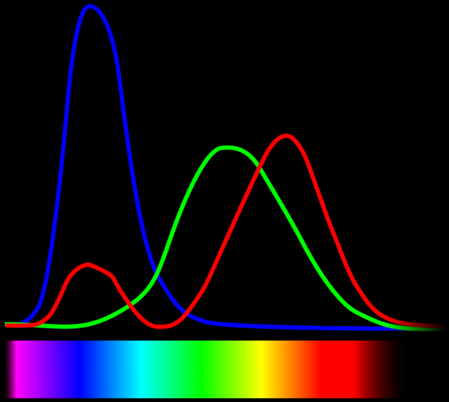
Two Parts

next time:

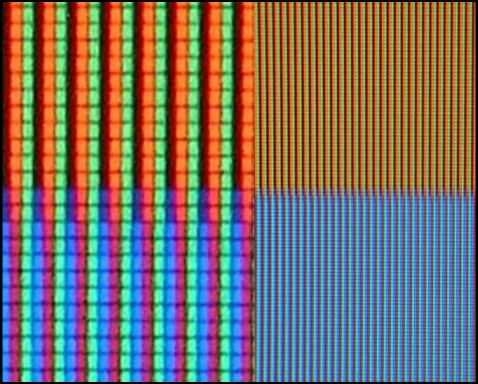
- Realtime Rendering
- hardware
- wicked shit
- demos

Spectrum

Color Perception



RGB



RGB

Traditional Color Formats

- 5+6+5 bits (RGB) = 16bit ("HighColor")
- 3*8 bits (RGB) = 24bit ("TrueColor")
- 4*8 bits (RGBA) = 32bit

Dynamic Range

defined as

- $r = \frac{max}{min}$

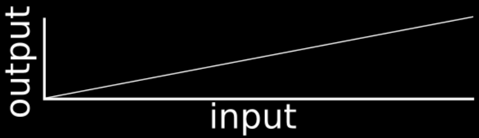
Dynamic Range

- human eye: 1,000,000
- computer displays: 1,000
- 24bit colors, per channel: 255

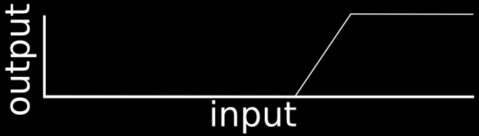
Tone Mapping

Reduce Dynamic Range while preserving good looks

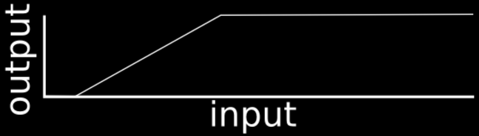
Tone Mapping



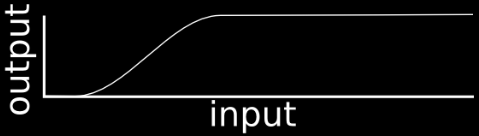
Tone Mapping



Tone Mapping



Tone Mapping



HDR Imaging

In classic Photography

- Tone Mapping by aperture/exposure
- no intelligent/custom Tone Mapping is possible

HDR Imaging

steps:

- acquire data from multiple LDR photographs
- generate a single HDR image
- Tone Map this image

HDR Imaging

HDR Imaging

HDR Rendering

- use HDR colors throughout the rendering process
- last step: Tone Map the rendered image

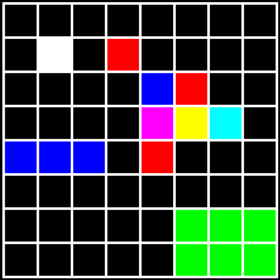
requires hardware support for 16bit floating-point textures

HDR Rendering

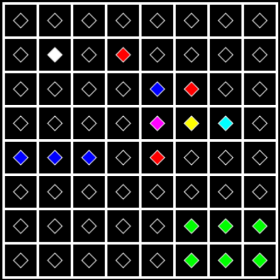
Pixels

pixel = "picture element"

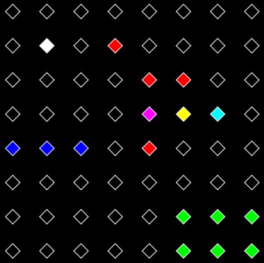
Pixels



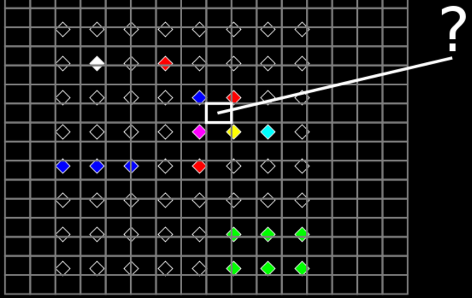
Pixels



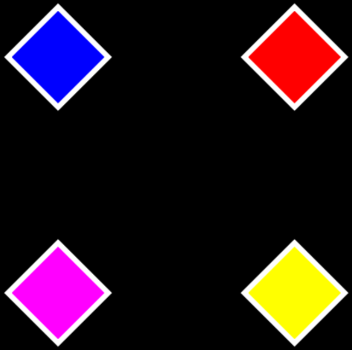
Pixels



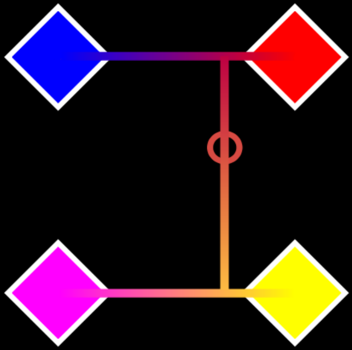
Resampling



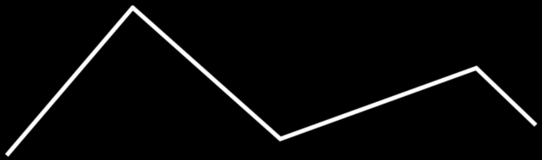
Sampling



Bilinear Filtering

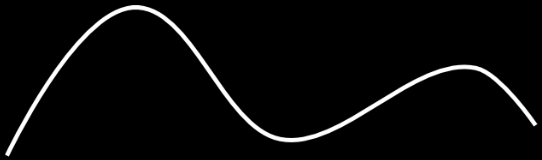


Linear Interpolation



Nearest Neighbour

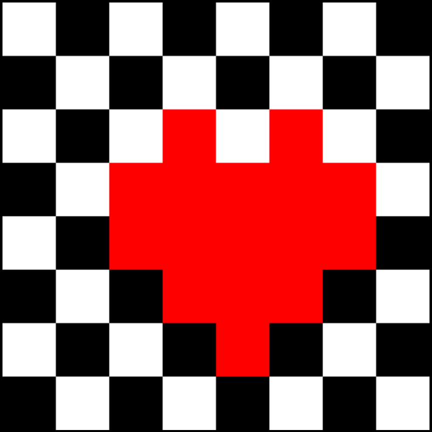
Smooth Interpolation



Various Techniques

- Nearest Neighbour
- Linear
- Cubic: stitch many $f(x) = ax^3 + bx^2 + cx + d$ together
- Optimal: *sinc* Filter

Nearest Neighbour



Linear Interpolation

Cubic Interpolation

Signal Processing

Treat color information as a signal:

- pixel distance = sampling rate
- use Highpass/Lowpass Filters to access frequencies

$\longrightarrow$ JPG etc

Scaling small amounts

Downsampling

- must use more than four pixels
- perform a Lowpass Filter
- often ignored

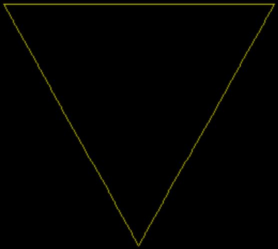
Nearest Neighbour

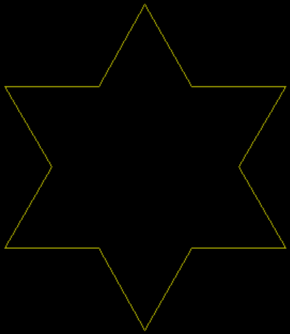
Linear filtering

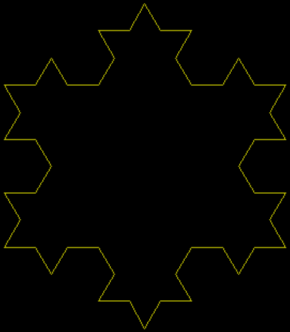
In Practice

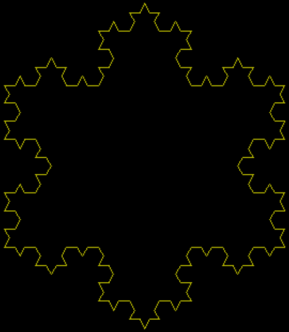
- Nearest Neighbour: crappy software (browsers, pdf viewers)
- Bilinear: Graphics Hardware
- Bicubic etc.: Image Processing

Fractals

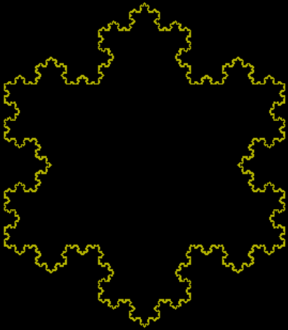




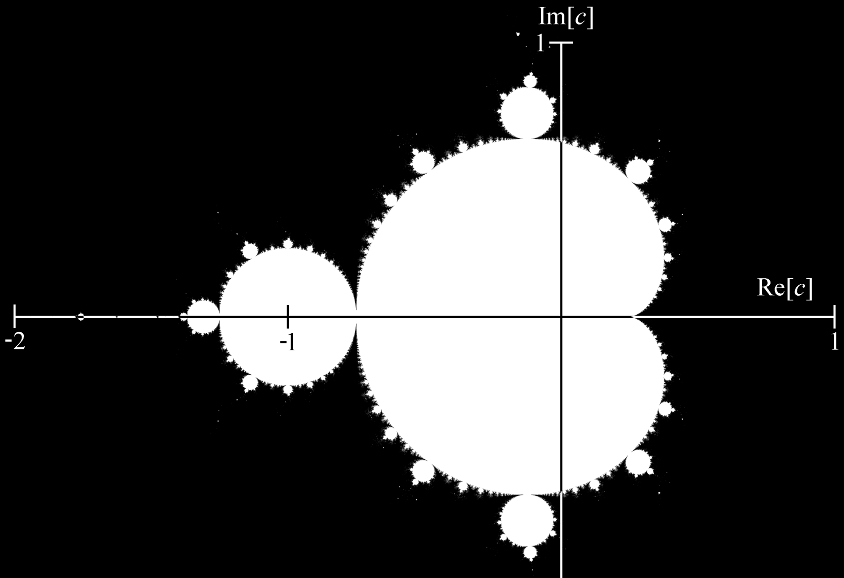




etc.



Mandelbrot Set



omg complex

- $0 + 2i$
- $1 + 0i$
- $2 + 7i$

$i^2 = -1$

$(3+2i)*(1+1i) = 3+5i+2i^2 = 1+5i$

The Formula

$$\boxed{z_{i+1} = z_i^2 + c} \tag{1}$$

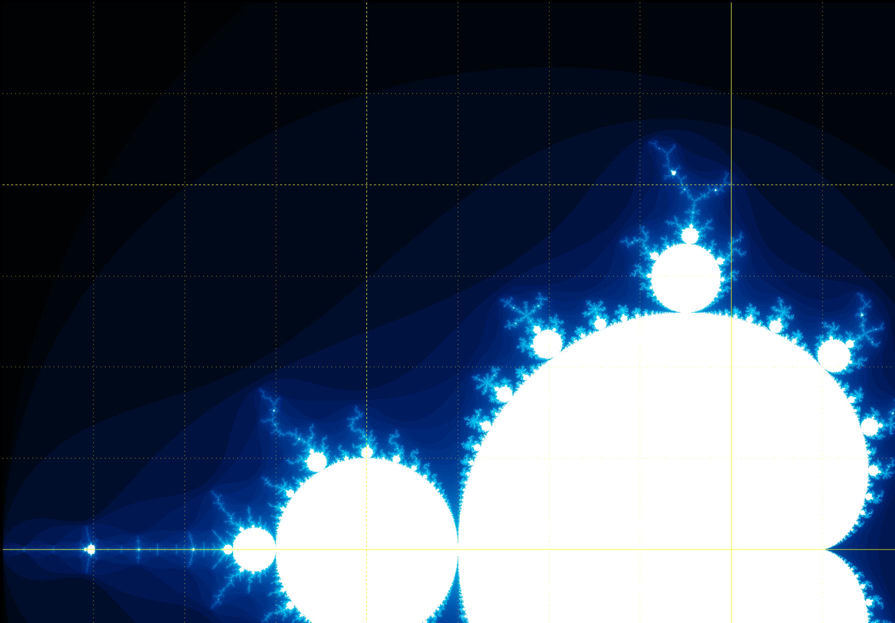
where

- $z_0 := 0 + 0i$
- $c := x + yi$

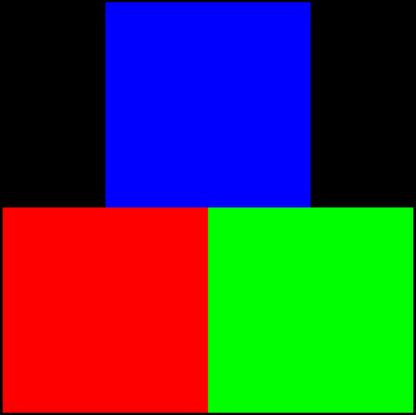
for every pixel:

- $z \longrightarrow 0 \Rightarrow z \in M$
- $z \longrightarrow \infty \Rightarrow z \notin M$

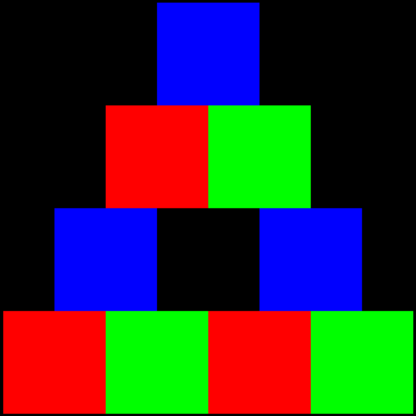
with color



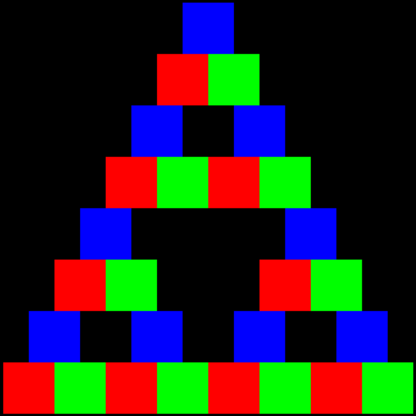
Iterated Function System



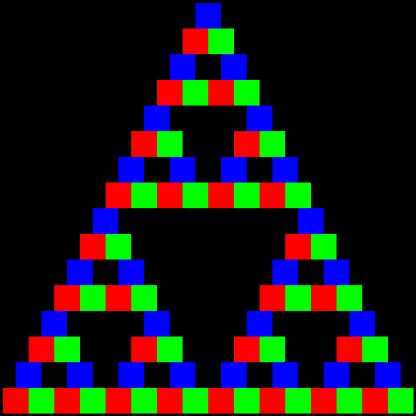
Iterated Function System



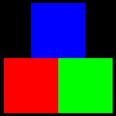
Iterated Function System

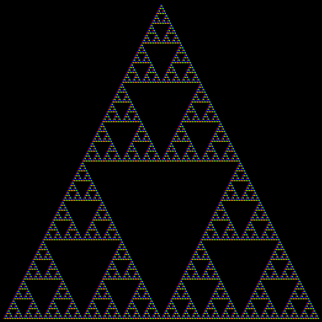


Iterated Function System

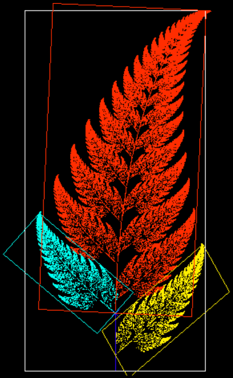


Iterated Function System

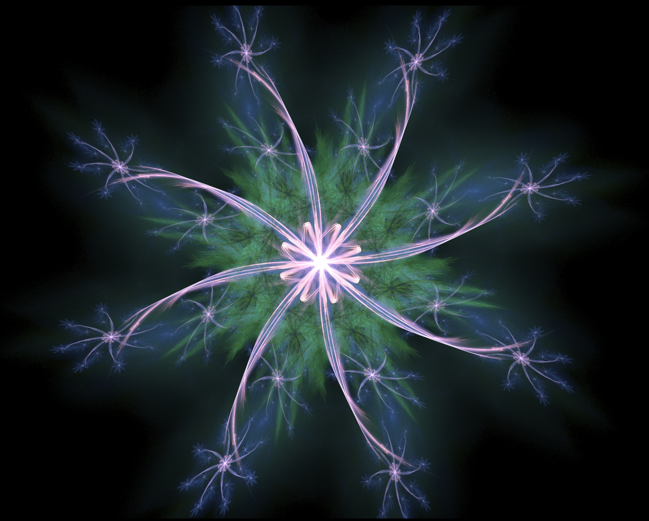




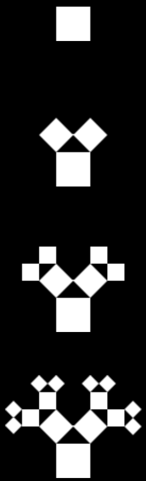
another IFS



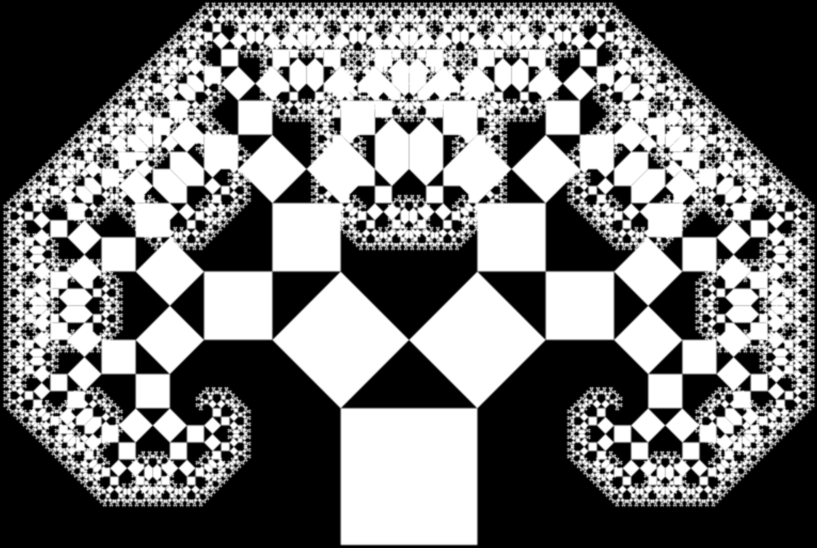
“Fractal Flames”



Nature



Nature



Nature

Nature

Pause?

Fractal Noise

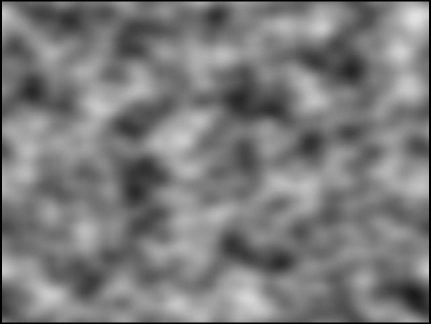
We want a pattern that

- is “random”
- contains structure of various sizes
- looks “natural”

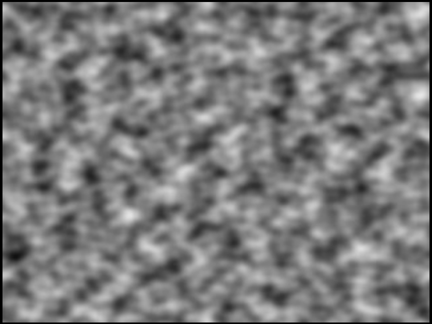
blurred Noise

Next octave

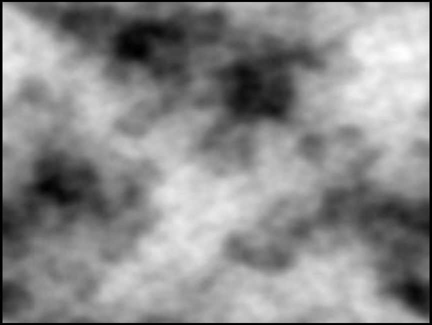
Next octave



Next octave



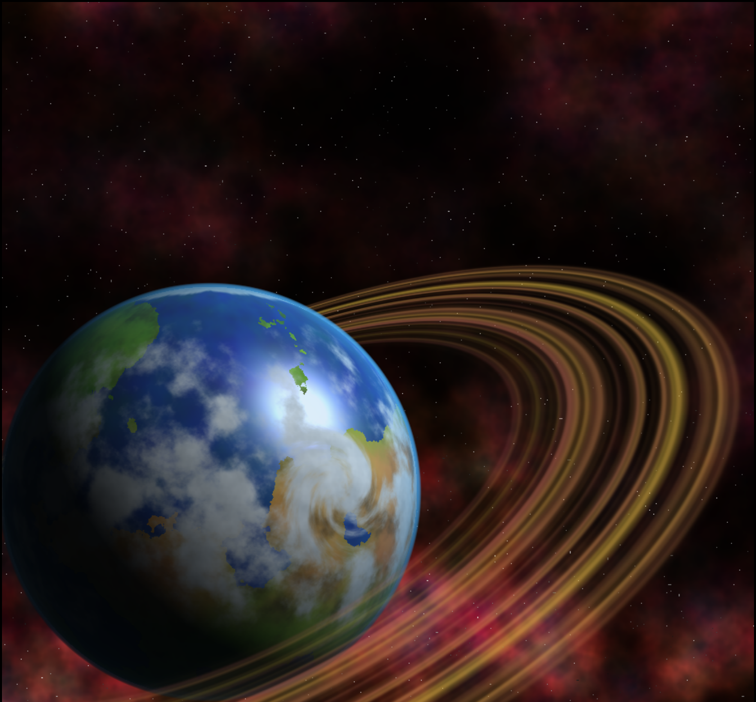
Summing four octaves



This is called “Perlin Noise” and everybody uses it.

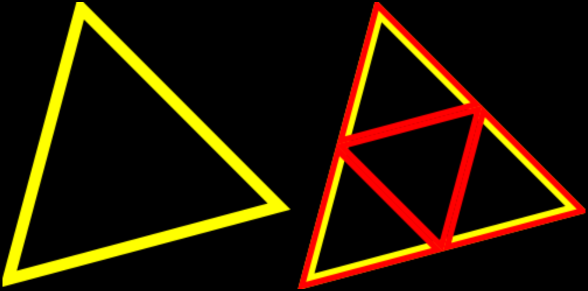
Usage

- clouds
- height fields (terrain)
- wood, marble
- can be animated by using more dimensions



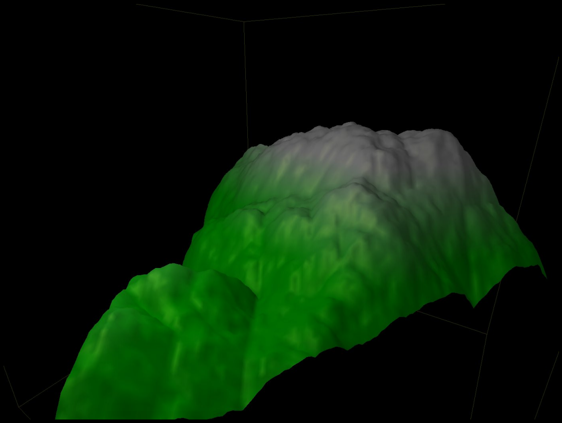
More Terrain

Take a mesh of triangles and repeat:

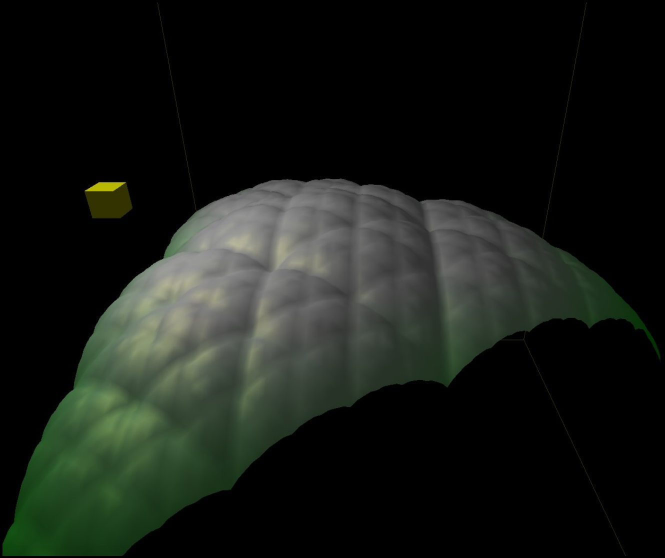


- subdivide by cutting the edges at their midpoints
- displace the midpoints

Midpoint Displacement



sucks, because



Rendering

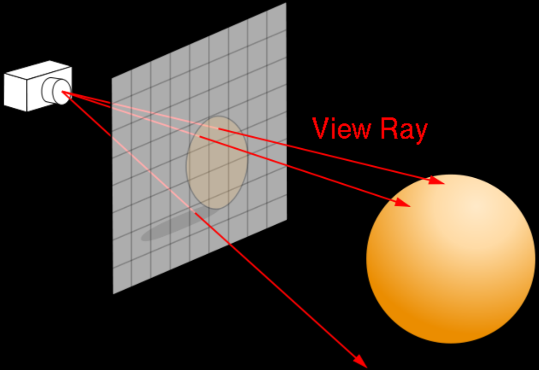
input:

- Geometry
- Material

output:

- pixels

Rendering



Geometry Representation

- Geometric Primitives
- Voxels, Point Clouds
- Polygons
- Isosurfaces

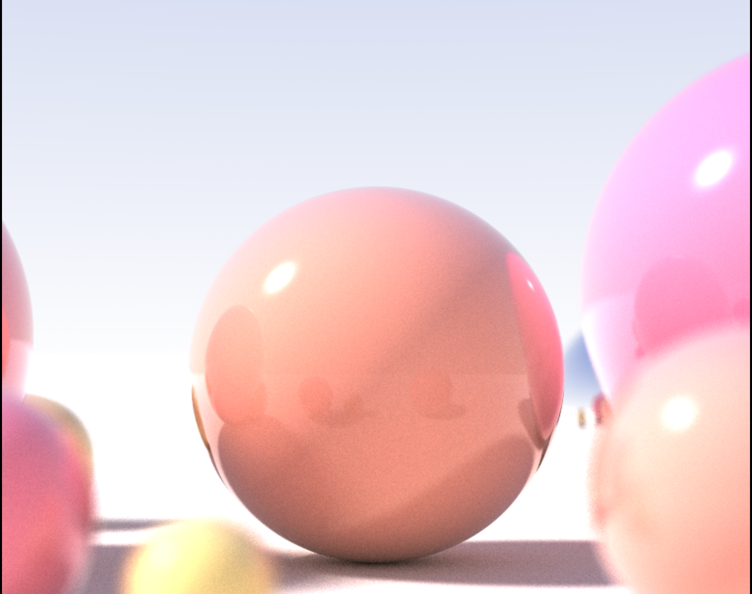
Geometric Primitives

Spheres, Cubes, Cylinders, ...

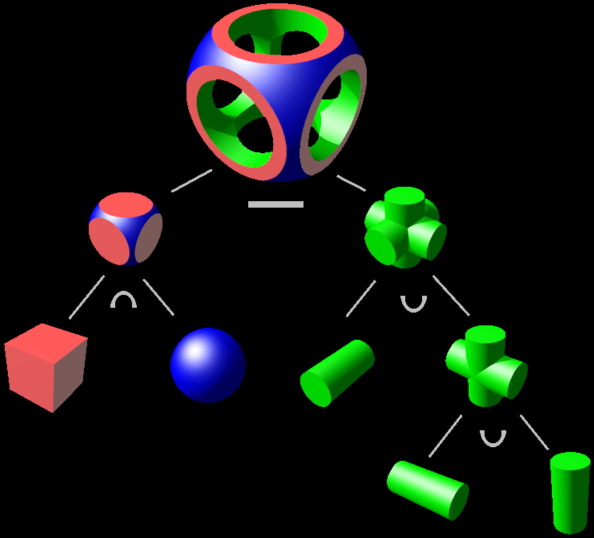
- Sphere stored as center, radius
- Store transformations, boolean operations

(Constructive Solid Geometry)

Geometric Primitives



CSG

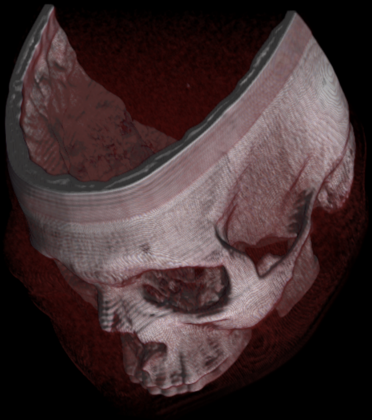


Voxels

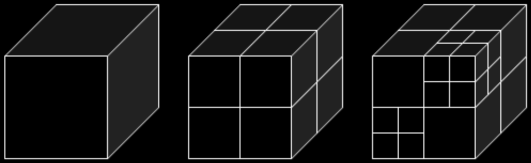
- Divide 3D space into equally sized voxels
- Store one bit per voxel

Used in medical imaging, old (and maybe future) games, movies

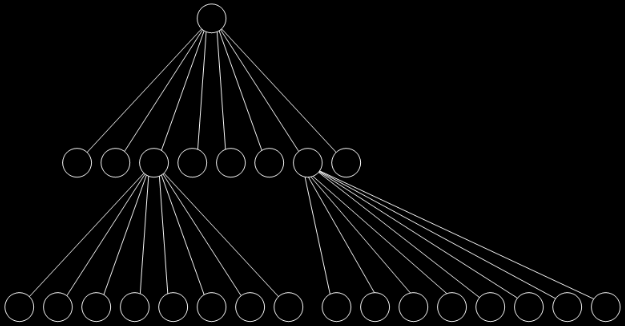
Voxels



Octree



Octree



Polygons

- Store points and connectivity
- Provides no concept of "inside" and "outside"

Used in games, movies

Polygons

Polygons

- generally unsorted ("polygon soup")
- need to be triangulated for rendering $\rightsquigarrow$ triangles.

Isosurfaces

- $f : \mathbb{R}^3 \longrightarrow \mathbb{R}$
- visualize the surface where that function yields some constant value c:
  $f(\vec{x}) = c$

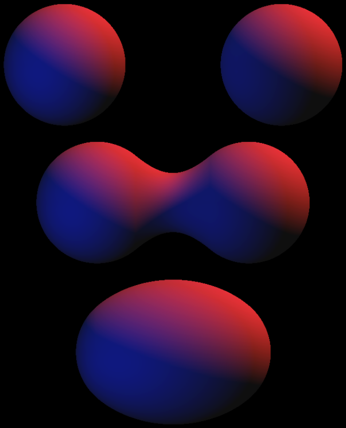
Example

f is the sum of the "distances" between $\vec{x}$ and two given points $\vec{a}$ and $\vec{b}$:

$$f(\vec{x}) = myDist(\vec{x}, \vec{a}) + myDist(\vec{x}, \vec{b})$$

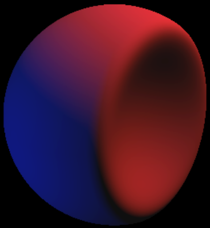
Example

$$f(\vec{x}) = myDist(\vec{x}, \vec{a}) + myDist(\vec{x}, \vec{b})$$



Example

$$f(\vec{x}) = myDist(\vec{x}, \vec{a}) - myDist(\vec{x}, \vec{b})$$

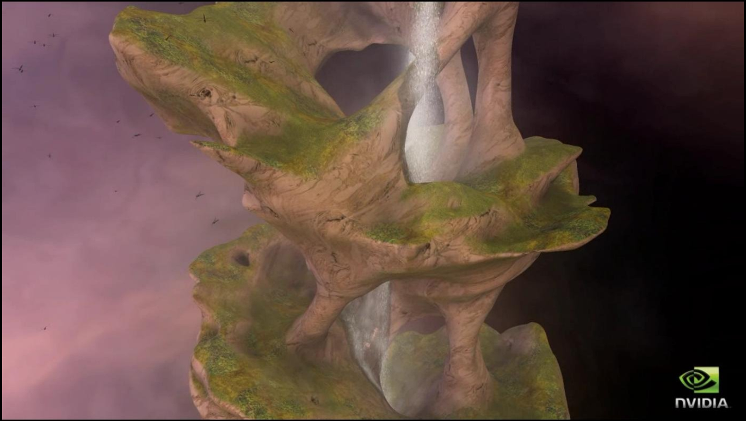


Rendering Isosurfaces

two possibilites:

- walk view rays while evaluating f, approximate intersection
- transform into a lots of polygons

Isosurface



Appearance

- lighting
- material

Illumination

- local
- global

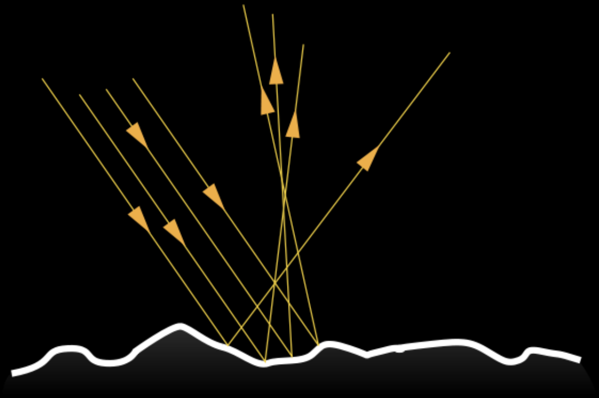
Material

"Material" models

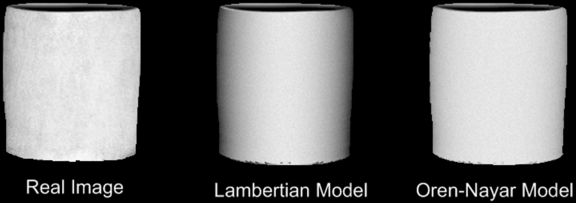
- micro-geometry
- color
- light transmission

Micro-Scale

micro-scale roughness: matte surface



Diffuse Reflection



Meso-Scale

meso-scale roughness: small visible bumps

Macro-Scale

macro-scale roughness: we have geometry for that

Color

usually stored in one or more textures

Appearance

can vary according to

- light color
- light angle, viewing angle (velvet)
- environment (mirror, glass)

$\rightsquigarrow$ there can be no comprehensive model

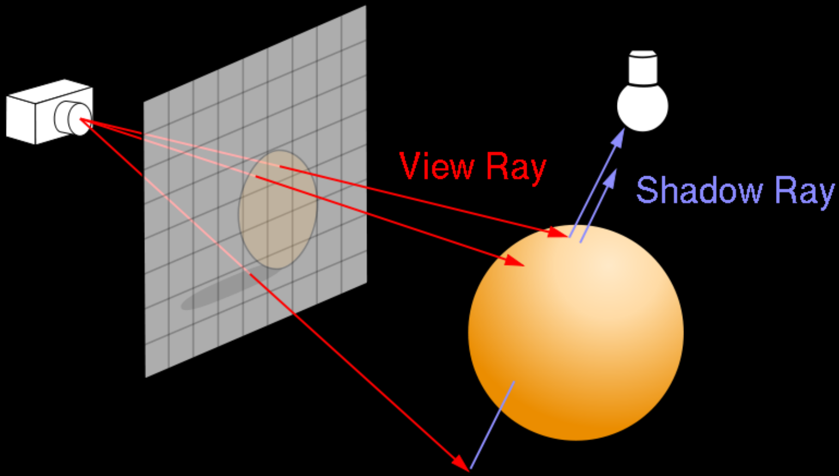
Offline Rendering

- static scene, no user interaction
- no hard time constraints

Realtime Rendering

- time constraint: 20msec
- a whole industry built around clever cheating

Ray Tracing



Ray Tracing

Tracing "photons" from the light source is also possible

Rasterization

for each object:

- project it onto the viewing plane
- paint all the pixels it covers

Painter's Algorithm

- sort objects
- draw from back to front

Sucks

- wasteful in scenes with high occlusion
- can't handle intersections
- sorting is in $O(n \log n)$

Z-Buffer Algorithm

- draw objects in any order
- for every pixel, store distance to camera in a buffer
- paint pixel only if distance to camera is lower

Next time

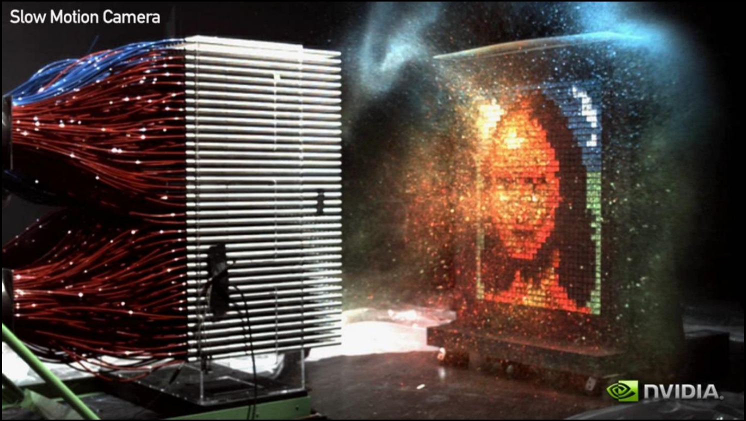
- graphics pipeline and hardware
- wicked techniques
- GPGPU, future technologies
- demos


CRAIG DONNER AND HENRIK WANN JENSEN - RENDERED USING DALI - 2005

NVIDIA


NVIDIA

500 GFLOPS vs. 10 GFLOPS


Slow Motion Camera
NVIDIA

Questions?