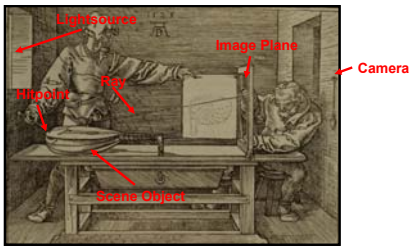
Ray Tracing - Part II

A structural change…

Elmar Eisemann
Delft University of Technology (TU Delft)

Basic principles of ray tracing

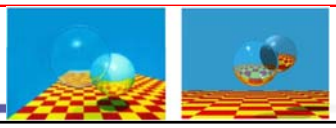


Send straight rays from the camera through the image plane and evaluate the lighting at the first hitpoint with the scene.

Recursive Ray Tracing

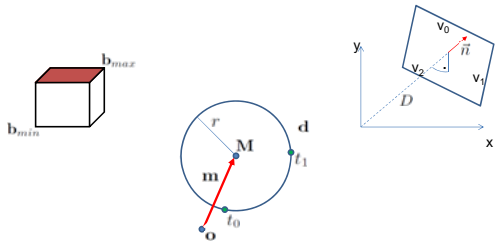
aka. Whitted-Style Ray Tracing [Whitted1980]

- For all pixels
  - Compute ray from camera through pixel
  - Compute first hitpoint
  - Compute direct lighting and evaluate shadows
  - Compute reflected and refracted ray
  - Recursively continue for both rays



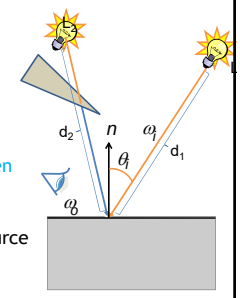
Ray / Object Intersection

- Intersections with different objects



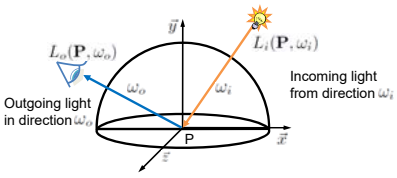
Direct Light and Shadows

- Do we need to shade the hitpoint
- Shadow test
  - Test if any object lies between hitpoint and light source
  - If in shadow skip further computation for this light source



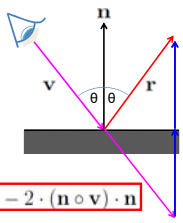
BRDF (Material)

- BRDF ratio of incoming light distributed into specific directions



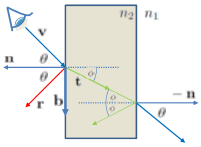
Reflection

- Reflection on mirror
- All vectors normalized



$$\mathbf{r} = \mathbf{v} - 2 \cdot (\mathbf{n} \circ \mathbf{v}) \cdot \mathbf{n}$$

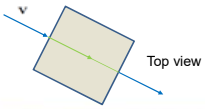
Refraction



- Snell's Law

$$n_1 \sin\theta = n_2 \sin\phi$$

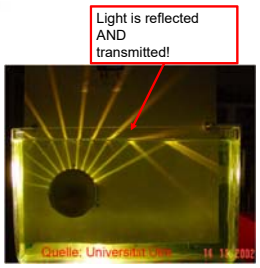
$$\mathbf{t} = \sin\phi\mathbf{b} - \cos\phi\mathbf{n} = \frac{n_1}{n_2}(\mathbf{v} - (\mathbf{v} \circ \mathbf{n}) \cdot \mathbf{n}) - \mathbf{n} \cdot \sqrt{1 - \frac{n_1^2(1 - (\mathbf{v} \circ \mathbf{n})^2)}{n_2^2}}$$



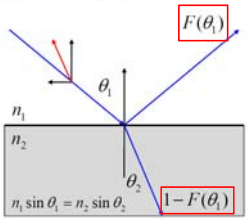
- The transmitted ray always lies in plane spanned by viewing vector and normal

Reflection and Refraction Tradeoff

- The amount of reflection and refraction varies with angle
- Result given by Fresnel equation
- Depends on material and angle



Reflection and Refraction Tradeoff



Schlick's Approximation:

$$F(\theta_1) \approx F_0 + (1 - F_0)(1 - \cos\theta_1)^5$$

F_0 is a constant - the reflection for upright light

Today

- How to make ray tracing fast?
  - Acceleration Data Structures
- How to make it beautiful?
  - Distribution Ray Tracing

Motivation

- Naïve Ray Tracing : Each ray vs each primitive
  - E.g. 1920x1080 pixels (Full HD)
  - Macbook Pro (Singlecore 3,306 GFlops)
  - Intersection test ~70 Flops, i.e. roughly 4 * 10^{10} intersection tests/s
  - 1 light, just 1 bounce reflection requires 4 rays

Stanford Bunny
69,451 triangles
576.05 * 10^9 tests
0.2 seconds

Happy Buddha
1,087,716 triangles
9.2 * 10^{12} tests
4 minutes

Lucy
28,055,742 triangles
232.8 * 10^{12} tests
97 minutes

St. Matthews
372 million triangles
3092 * 10^{12} tests
21 hours

Motivation for acceleration

Slower than MacBook Pro

13

Motivation for acceleration

- Ray Tracing is slow!
- Complexity O(nm)
  - n objects
  - m rays

14

Motivation for acceleration

- BUT the last lecture was useful:
- Even with acceleration at least 50% spent on intersection tests
- i.e. without ~99.99%
- There is potential to speed up your ray tracer by ~10,000% !

How to accelerate ray tracing?

- Effective algorithms for intersection tests and vector operations
  - effective formulae
  - effective algorithms
- Cache-Optimization
  - Avoid global memory access
  - Keep data in caches
  - Exploit Coherence
- Parallelisation
  - Millions of rays available, all can be traced in parallel
- BUT: all this are just linear speed-ups...

16

Acceleration

- Acceleration Data Structures
  - Reduce number of intersection tests by „clustering“ objects

Spheres	10	91	820	7381	66430
Brute-force	2.5	11.4	115.0	2677.0	24891.0
Bottom-up	2.1	3.0	4.5	6.5	10.2

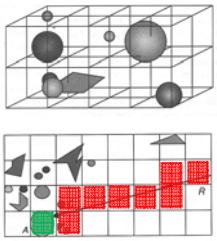
Acceleration Data Structures

- Reduce number of intersection tests
- Object Partitioning Schemes
  - Bounding Volume Hierarchy
- Space Partitioning Schemes
  - Uniform Grid
  - Octree
  - kd-Tree
- Directional Techniques
  - Ray Classification
- Implicit Acceleration Data Structures
  - Divide-and-Conquer
  - Geometry Presorting

18

Uniform Grid

- Grid
  - Partitioning into equal, fixed sized „voxels“
- Building a grid structure
  - Partition the bounding box
  - Resolution: often $\sqrt[3]{n}$
  - Inserting objects
    - Insert into all voxels overlapping objects bounding box
    - Easily optimized
- Traversal
  - Iterate through all voxels in order as pierced by the ray
  - Compute intersection with objects in each voxel
  - Stop if intersection found in current voxel



19

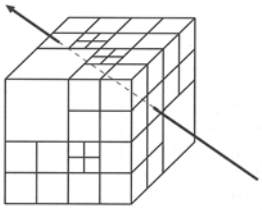
Uniform Grid: Issues

- Grid traversal
  - Requires enumeration of voxel along ray → 3D-DDA
  - Simple and hardware-friendly
- Grid resolution
  - Strongly scene-dependent
  - Cannot adapt to local density of objects
    - Problem: „Teapot in a stadium“
  - Possible solution: grids within grids – hierarchical grids
- Objects in multiple voxels
  - Store only references
  - Use mailboxing to avoid multiple intersection computations
    - Store (ray, object)-tuple in small cache (e.g. with hashing)
    - Do not intersect if found in cache
  - Original mailbox uses ray-id stored with each triangle
    - Simple, but likely to destroy CPU caches

20

Octree

- Hierarchical space partitioning
  - Start with bounding box of entire scene
  - Recursively subdivide voxels into 8 equal sub-voxels and assign objects to all sub-voxels they overlap
  - Stopping-Criteria: Too few triangles & maximum depth
- Problems
  - Pretty complex traversal algorithms
  - Slow to refine complex regions
- Probably the slowest variant for classic ray tracing But great for Sparse Voxel Octrees (Unreal 4-Engine)

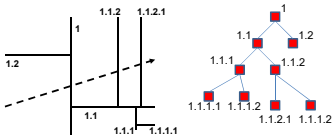


21

kD-Trees

– Recursive space partitioning with axis-aligned half-spaces

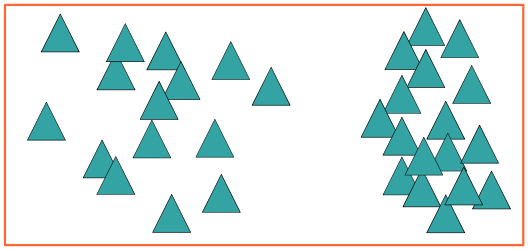
– “Best known method” - Stoll (SIGGRAPH '05)



...if you plan to ray trace only static scenes

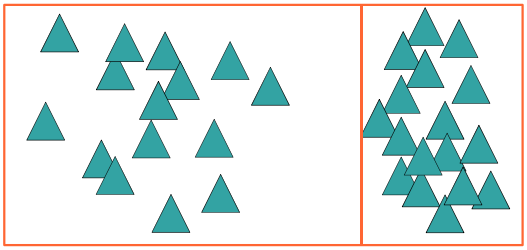
22

kD-Trees



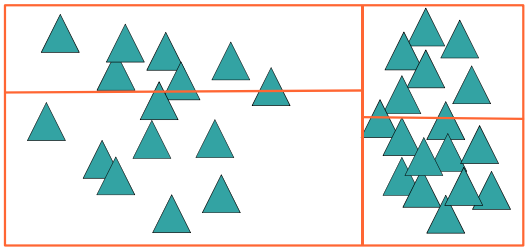
23

kD-Trees

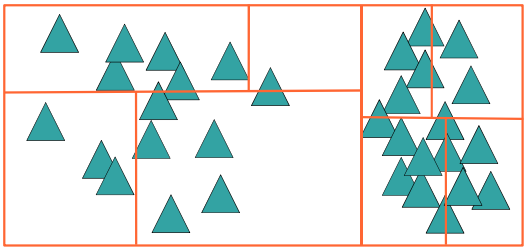


24

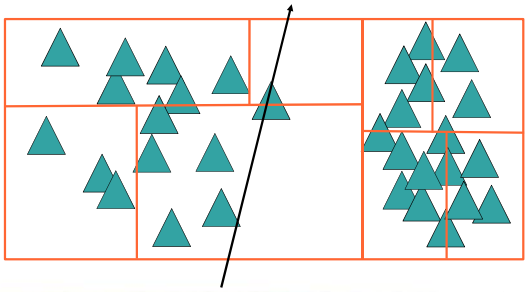
kD-Trees



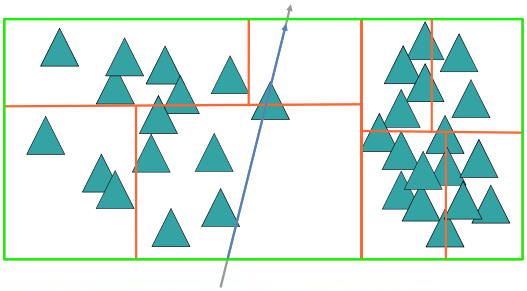
kD-Trees



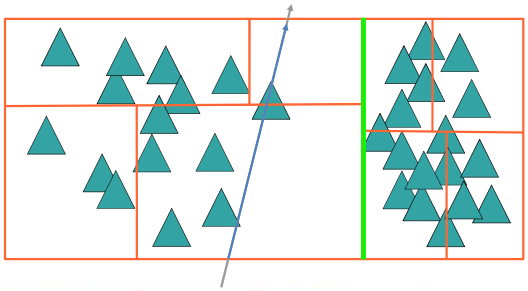
kD-Trees



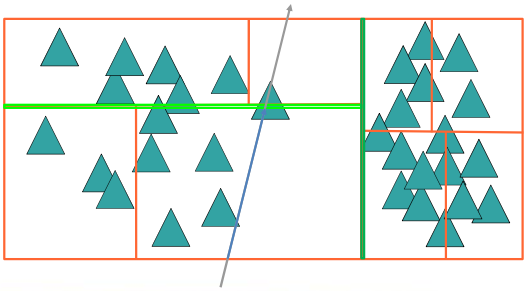
kD-Trees



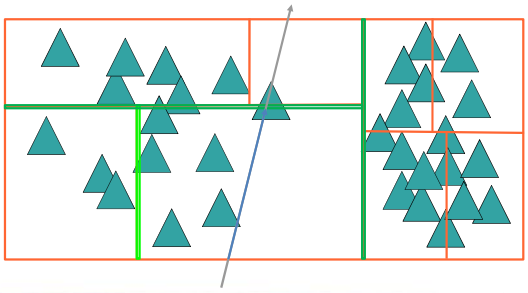
kD-Trees



kD-Trees

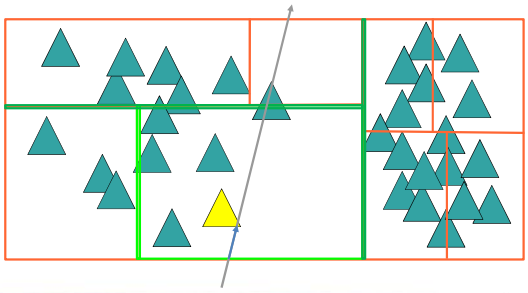


kD-Trees



31

kD-Trees



32

Introduction to kD-Trees

- Binary Tree:
  - Leaves: Contain scene objects or object list
  - Inner nodes: *Splitting plane* and *children pointer data*
- *Stopping criterion:*
  - *Maximum Depth, number of objects, cost function, …*
- *Advantages:*
  - *Adaptive (can handle the "Teapot in a stadium"-Problem)*
  - *Compact (only 8 bytes per node necessary)*
  - *Easy fast in-order traversal (first object hit is almost always the right one)*

33

Building kD-Trees

- *Given:*
  - *Axis-aligned bounding box of the scene ("cell")*
  - *List of geometry primitives*
- *Basic Algorithm:*
  1. *Pick axis-aligned plane to split cell into two*
  2. *Shift geometry into children, split up if needed*
  3. *Goto 1, until termination criterion is fulfilled*

34

Building kD-Trees

- *"Intuitive" kD-Tree building*
  - *Split Axis:*
    - *Round robin (x,y,z,…) OR Largest extend*
  - *Split Location:*
    - *Middle of extend OR Median of geometry*

These techniques stink!

35

Good way of building kD-Trees

- *Based on cost optimization*
  - *What is the cost of tracing a ray through a cell ?*

*Cost(cell) = C_trav + P(hit L) * Cost(L) + P(hit R) * Cost(R)*

C_trav: Costs for split plane intersection

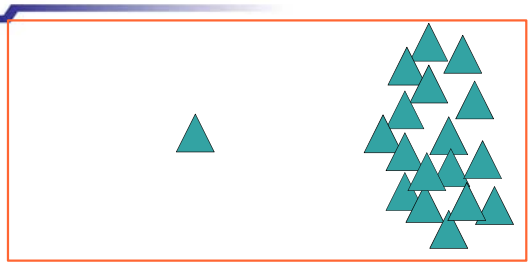
P(hit L): Probability that ray hits left child

P(hit R): Probability that ray hits right child

Cost(X): Cost for child X (~number of triangles)

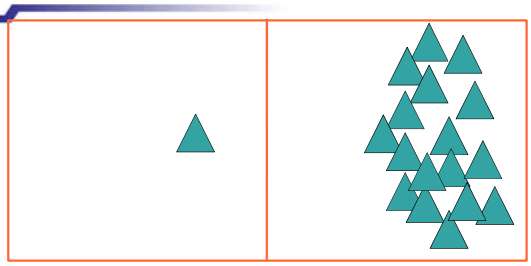
36

Building kD-Trees



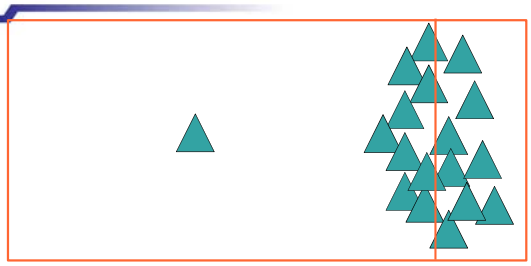
37

Split at the middle



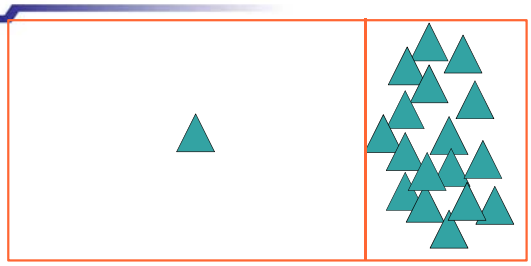
38

Split at the median



39

Cost optimized rule



40

Building kD-Trees

- *Good kD-Tree building*
  - *Based on cost optimization*

    *Cost(cell) = C_trav + P(hit L) * Cost(L) + P(hit R) * Cost(R)*
  - *P equals surface area of the cell*
  - *Cost of child cells equals roughly the triangle count*
  - *Can also be used as a stopping criterion*
- *Algorithm*
  1. *Choose a set of splitting plane candidates, e.g. use objects borders or a few bins (16 is often enough)*
  2. *Evaluate cost function*
  3. *Choose splitting plane with lowest cost*
  4. *Recurse*

41

Building kD-Trees

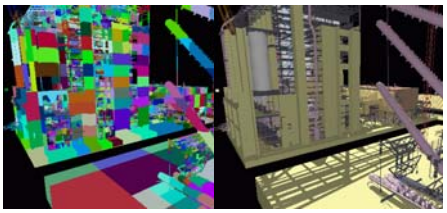
- *Good kD-Tree building*
  - *ONLY an approximation!*
  - *Real Cost:*

$$C(Tree) = \sum_{n \in nodes} \frac{SA(V_n)}{SA(V_S)} \kappa_T + \sum_{l \in leaves} \frac{SA(V_l)}{SA(V_S)} \kappa_I$$

Finding the optimal tree today is considered infeasible except for trivial scenes.

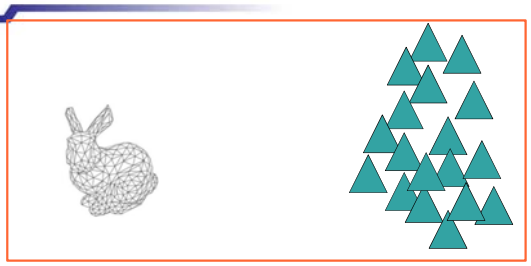
Building kD-Trees

- *Good kD-Tree building*
  - *Conclusion:*



43

Do we have to use isolated triangles?



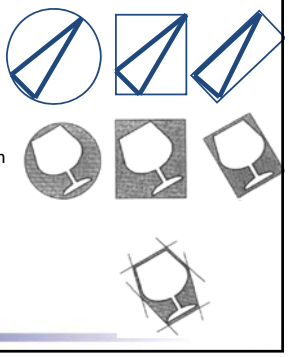
Triangles often form objects,

which are often moved coherently…

44

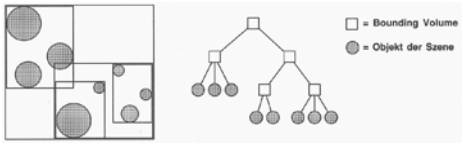
Bounding Volumes

- Observation
  - Bound geometry with BV
  - Only compute intersection if ray hits BV
- Sphere
  - Very fast intersection computation
  - Often inefficient too large
- Axis-aligned box
  - Simple computation (min-max)
  - Also often large, but…
  - Most common representation
- Non axis-aligned box
  - Aka. „oriented bounding box (OBB)“
  - Often better fit
  - Very complex computation
- Slabs
  - Pairs of half spaces
  - Complex test
  - Fixed number of orientations
  - Fairly fast computation



BVH

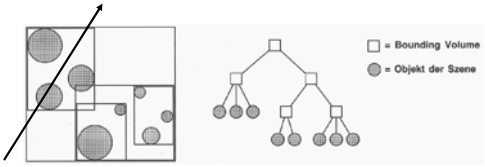
- Bounding Volume Hierarchy
  - Hierarchical organization of Bounding Volumes
  - In-order traversal difficult



46

BVH

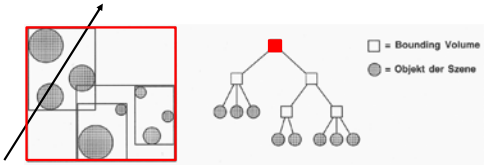
- Bounding Volume Hierarchy
  - Hierarchical organization of Bounding Volumes
  - In-order traversal difficult



47

BVH

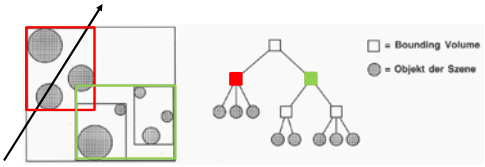
- Bounding Volume Hierarchy
  - Hierarchical organization of Bounding Volumes
  - In-order traversal not easily achieved



48

BVH

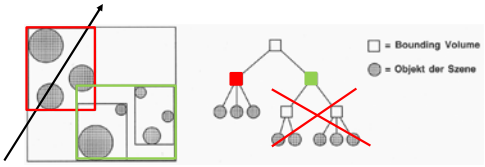
- Bounding Volume Hierarchy
  - Hierarchical organization of Bounding Volumes
  - In-order traversal not easily achieved



49

BVH

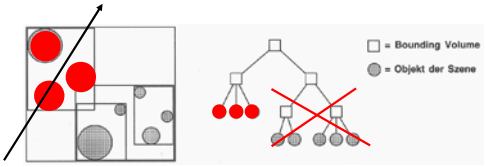
- Bounding Volume Hierarchy
  - Hierarchical organization of Bounding Volumes
  - In-order traversal not easily achieved



50

BVH

- Bounding Volume Hierarchy
  - Hierarchical organization of Bounding Volumes
  - In-order traversal not easily achieved

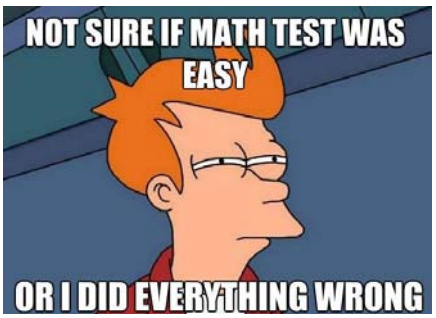


51

Pit Stop

- Kd trees very good for static scenes
- BVH mostly interesting for fast construction
  - Dynamic scenes?!
- Many different structures exist
- Construction process of the structure affects:
  - Efficiency
  - Memory footprint

Questions?



Distributed Ray Tracing

- Distributed Ray Tracing
  - Anti-Aliasing / Supersampling
  - Glossy Reflections
  - Translucency
  - Depth-of-Field
  - Motion Blur
  - Area Light Sources

Distributed Ray Tracing

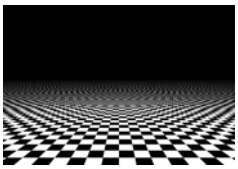
- aka.
  - Distribution Ray Tracing or
  - Stochastic Ray Tracing
- Rendering of „soft“ phenomena
- Idea:
  - Nature is not sharp
  - A single ray is not enough,
    e.g., perfect reflection
    is not realistic
    (brushed steel)

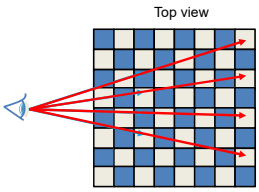
Distributed Ray Tracing

- You have seen a case, where too much information per pixel caused artifacts…

Example Aliasing / Undersampling

- Aliasing example
  - One ray for each pixel (say, at pixel center)
  - Checkerboard period becomes smaller pixels
    - black or white by “chance”



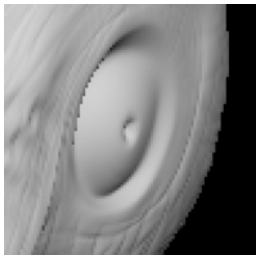


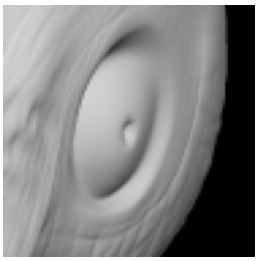
57

What is the difference?

58

What is the difference?

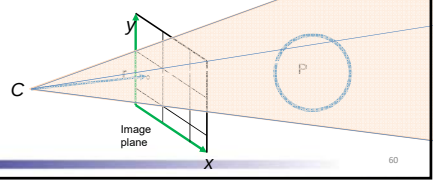




59

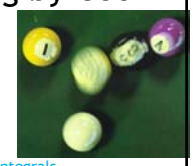
Supersampling

- Each pixel is a cone of light towards the eye
- A single ray is a point sample
- Hence, the mismatch and artifacts
- Need the mean radiance over the pixel‘s „surface“ ← Integral



60

Distributed Ray Tracing by Cook



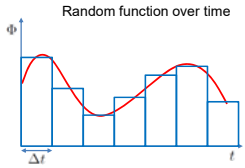
[Cook1984 – Distributed Ray Tracing] Lucasfilm

- Ray Tracing is a sampling problem.
  We have point samples (our rays)
  but need integrals (e.g., light at a pixel)
- Elegant framework for various visual effects
- Idea: Describe phenomena as multidimensional integrals
  and sample them together, not separate (Numerical solution)
  - Sampling the image plane reduces aliasing
  - Sampling reflected rays following a specular distribution function leads to gloss
  - Sampling the transmitted ray produces translucency (blurred transparency)
  - Sampling the solid angle of the light produces soft shadows
  - Sampling the camera lens produces depth of field
  - Sampling in time produces motion blur

61

Fun with Numeric Integration

- What is an integral?
- An integral is the area underneath a function
- But how do I compute the integral if I don't know the function?



Δt : Small t

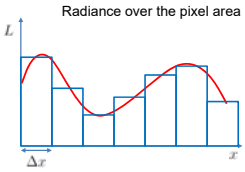
dt : Very, very, very small t

$$\sum_{i=1}^{N} \Phi_i \cdot \Delta t \xrightarrow{N \to \infty, \Delta t \to 0} \int_0^{t_{max}} \Phi(t) \cdot dt$$

62

Fun with Numeric Integration

- What is an integral?
- An integral is the area underneath a function
- But how do I compute the integral if I don't know the function?



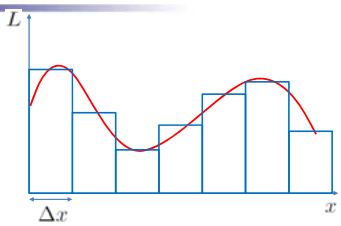
Δx : Small x

dx : Very, very, very small x

$$\sum_{i=1}^{N} L_i \cdot \Delta x \xrightarrow{N \to \infty, \Delta x \to 0} \int_{x_{min}}^{x_{max}} L(x) \mathrm{d}x$$

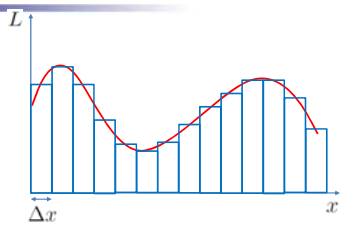
63

Fun with Numeric Integration



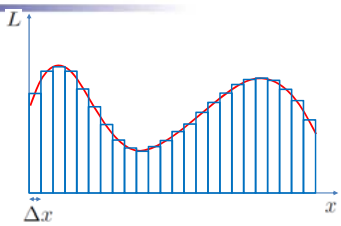
64

Fun with Numeric Integration



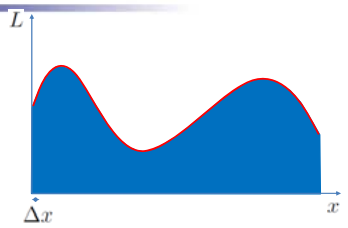
65

Fun with Numeric Integration

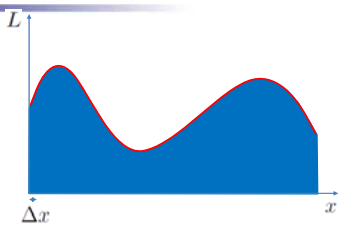


66

Fun with Numeric Integration



Fun with Numeric Integration



- Under the assumption that the function is constant for a small Δx the integral is just a sum of rectangular areas.
- The height of these rectangles is just the function value (radiance).

Supersampling of a single pixel

- Compute mean radiance over a single pixel
- Evaluate point samples at grid points
- Average

$$\frac{1}{(y_{max}-y_{min})\cdot(x_{max}-x_{min})}\int_{y_{min}}^{y_{max}}\int_{x_{min}}^{x_{max}} L(x,y)\mathrm{d}x\mathrm{d}y$$

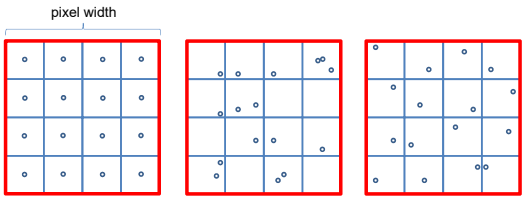
$$\approx \frac{1}{(N\cdot\boxed{\Delta y})\cdot(N\cdot\boxed{\Delta x})}\sum_{i=1}^{N}\sum_{j=1}^{N} L_{i,j}\boxed{\Delta x\Delta y}$$

$$= \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} L_{i,j}$$

(just the average of all samples taken)

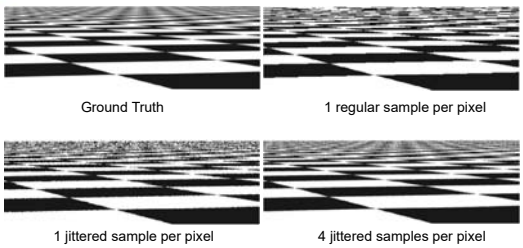
Sampling Pattern

- 16x Supersampling (16 primary rays per pixel)



Regular Sampling | Random Sampling | Stratified Sampling (Jittered Sampling)

Example

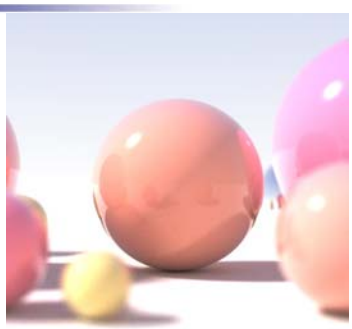


Ground Truth | 1 regular sample per pixel

1 jittered sample per pixel | 4 jittered samples per pixel

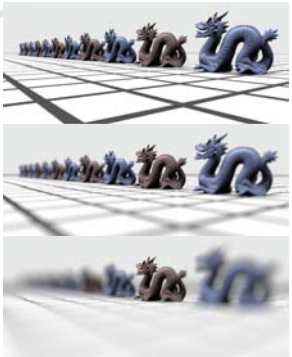
Is averaging non-regular samples mathematically correct?

Depth-of-Field

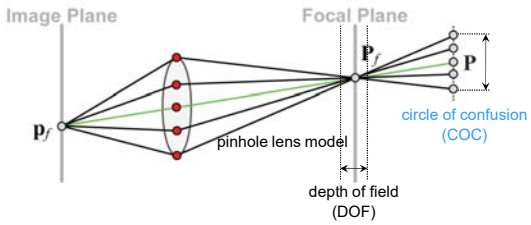


Depth-of-Field

- Objects are sharp only at a certain distance (focal distance)
- This range may vary depending on lens and aperture



Camera Model ?!

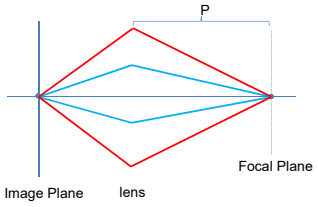


Depth of Field

[Lee, Eisemann, Seidel - SIGGRAPH'10]

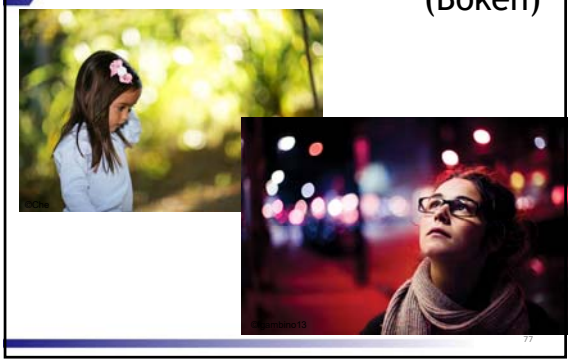
Aperture

- Opening to determine the ray bundle through the lens; tradeoff sharpness/brightness



76

Examples Circle of confusion (Bokeh)



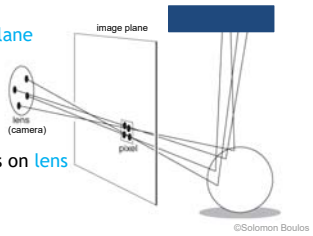
77

Aperture

78

Depth-of-Field Implementation

- Simple Implementation
- In CG
  image plane = focal plane
- Create „lens plane“ orthogonal to image plane/focal plane
- Sample random points on lens and / or pixel
- Trace rays

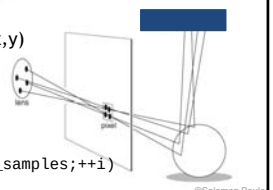


Depth-of-Field Implementation

- Pseudo-Code for the pixel (x,y)

```
// random sampling on
// both lens and pixel

result = (0,0,0);
for(int i=0; i<number_of_samples;++i)
{
  l = random position on lens;
  p = random position inside pixel (x,y);
  r = ray from l to p;
  result += trace(r);
}
result /= number_of_samples;
```

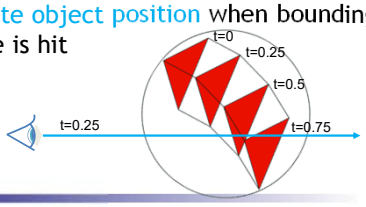


Motion Blur

- Motion Blur conveys a lot of information about the motion in an image
- Integral over time

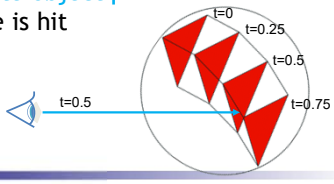
Motion Blur

- Trace rays at different times
  - Each ray has an associated time stamp
- Bounding Volumes around moving objects then build acceleration structure
- Compute object position when bounding volume is hit



Motion Blur

- Trace rays at different times
  - Each ray has an associated time stamp
- Bounding Volumes around moving objects then build acceleration structure
- Compute object position when bounding volume is hit



Motion Blur

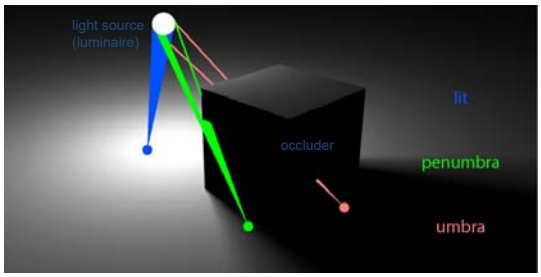
- Not only geometry moves
- Also light or camera

Soft Shadows

Hard Shadows **Soft Shadows**

85

Soft Shadows

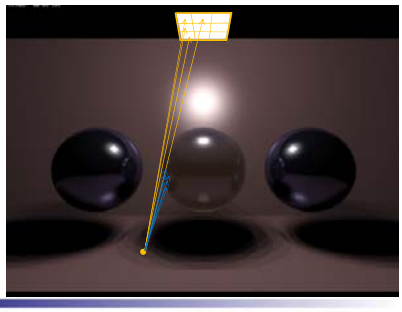


- Light sources have a non-infinite area

86

Distributed Implementation

- Replace light source by many emitters



87

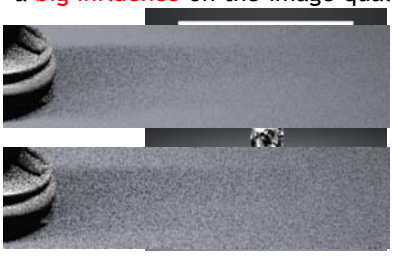
Distributed Implementation

- Average all samples
- Converges to correct result with increasing number of samples
  (if samples are uniformly distributed)

88

Combination of Effects

- Sampling of the different integrals has a big influence on the image quality



1 eye sample
16 shadow samples

16 eye samples
1 shadow sample

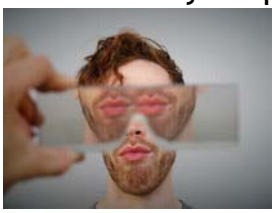
89

Summary

- Acceleration Data Structures
  - Uniform Grid
  - Octree
  - Kd-Tree
  - BVH
- Distribution Ray Tracing
  - Supersampling
  - Depth-of-Field
  - Motion Blur
- Global Illumination
  +Outlook

blog.imgtec.com

Thank you very much
for your attention
and have fun with your project!



3ders.org