Advanced Ray Tracing

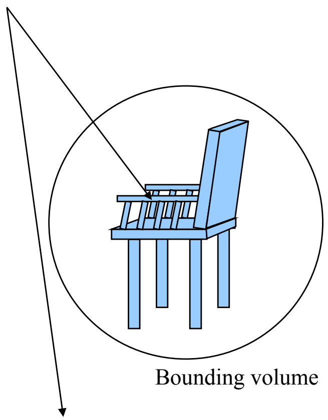
- Acceleration Methods
- Distributed Ray Tracing
- Advanced Illumination

Acceleration Methods

- Suppose there are 100,000 objects in the scene (moderate complexity by polygon-rendering standards), and the image is 1000 x 1000 pixels.
- In brute force ray tracer, each primary ray does 100,000 ray-object intersection tests.
- 10^6 primary rays => 10^{11} ray-object intersection tests
- If each test takes 50 - 500 floating point operations at average 2 nSecs per flop (1GHz machine), that's 10^4 – 10^5 secs, i.e. 2.8 – 28 hours rendering time.
- Plus cost of shadow test rays and reflections etc!
- Need to reduce per-ray intersection tests
- Methods:
  - Bounding volumes
  - Vista and light buffers
  - Space subdivision
  - Ray coherence

Bounding Volumes

- Idea:
  - find a volume (sphere or box) that encloses a complex object
  - Test ray against bounding volume
  - If it misses, don't test ray against object
- Particularly efficient with hierarchical scenes – whole sub-trees get skipped
  - e.g. if a chair is made up of arms, back, seat, legs, but ray misses bounding sphere, big win!



ray misses bounding volume – do one intersection test instead of 96!

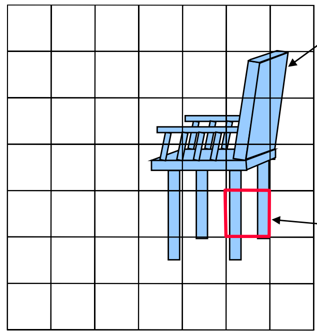
Bounding Volumes (cont'd)

- Automatic placement of bounding volumes is problematic
- Most scene-description languages for RT let you specify bounding volumes

Vista Buffer

- Do an initial projection of scene onto viewplane
  - Use usual polygon-rendering methods
- Make a list of which objects cover (partially or completely) each square "pixel" region.
  - Using a pixel *region* rather than a *point* allows us to do supersampling (or whatever) for antialiasing.
- Do primary ray intersection tests only with the objects that intersect each pixel region.

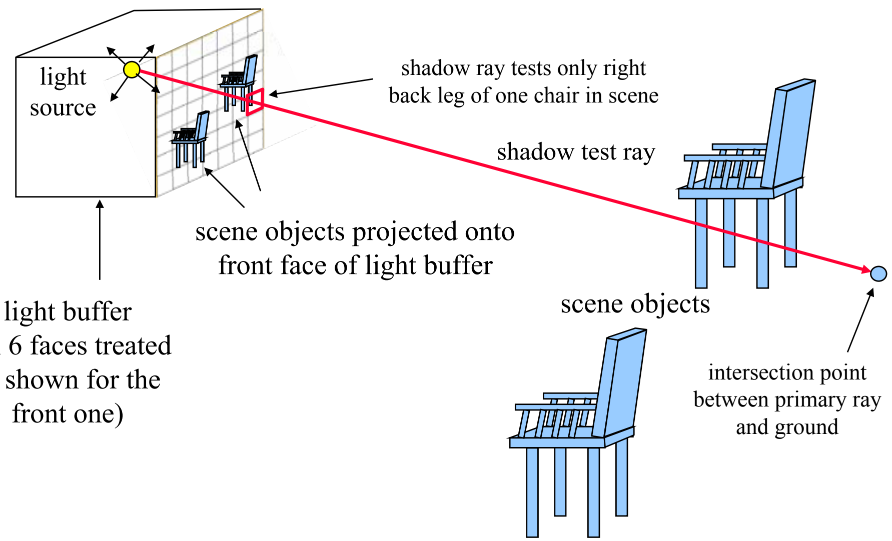
Vista Buffer



Light Buffer

- Shadow rays a major cost factor (90%??)
  - Usually have lots of lights
- Build a box around each point light source. Pixelate each face
- Project scene onto each face, making a list of all objects covering each pixel region
- For each shadow test ray
  - Determine which pixel region of which face of light buffer it passes through
  - Do intersection tests only with objects that project to that pixel

Light Buffer

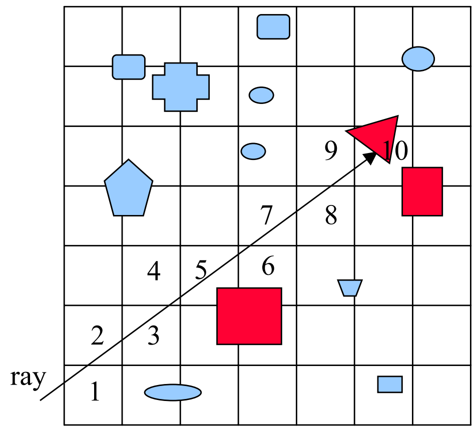


Space Subdivision

- Subdivide scene space in some way
- Determine what objects intersect each region of the subdivided space
- Trace ray through succession of sub-regions
  - Test only against objects within each subregion
  - Terminate if get a hit
- Subdivision schemes:
  - Regular grid ("Enumerated space")
  - Octree
  - BSP-tree

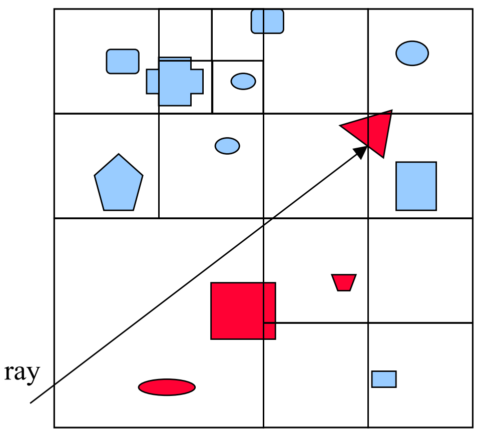
"Enumerated space"

- Subdivide scene space into a regular cell grid
- Pre-process scene
  - for each cell, make a list of relevant objects
- Trace each ray through the cell grid
  - Determine sequence of cells traversed
  - Intersect ray only with "relevant objects" of each cell [shown in red]
- Only a tiny percentage of objects get tested.



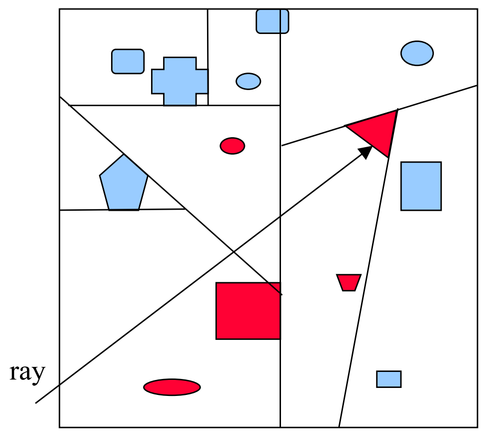
Octree

- As for enumerated space, but recursively subdivide scene space cube into 8 sub-cubes
  - Continue until few enough objects in cell (2 in example shown) or maximum subdivision level reached
- Advantage: step quickly over empty space
- Disadvantage: traversal algorithm much harder



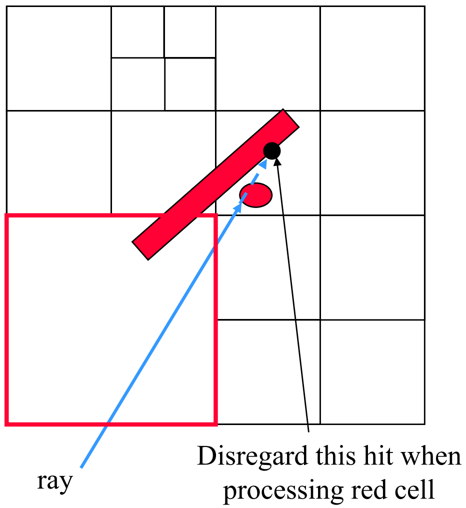
BSP-tree

- Use Binary Space Partitioning tree
  - As in visibility notes but subdivide only until "sufficiently few" objects in each leaf
- Intersection of ray with scene is a simple recursive descent
  - Much easier than other 2 methods
  - UDOO: why?
- But dealing with the fact that object boundaries lie on clipping planes is tricky.



With all spatial subdivision schemes:

- Must be careful to traverse cells in right order
- Must check ALL objects intersecting a cell and take nearest hit
- If ray hits an object, hit must be within the cell to be counted [see Fig.]



Ray Coherence

- Rather than subdividing space, subdivide space of all rays
  - Ray space is five dimensional: 3D starting point, 2D direction
- Idea
  - Consider the *beam* of all rays that start within a given cube of scene space, and head in a certain direction (with a certain tolerance).
  - Find all scene objects intersected by that beam
  - This set is the *candidate set* of scene objects for all rays in the beam
    - Small fraction of total object set
    - Only build candidate set once
    - Recursively refine beam size as required
  - See: Arvo & Kirk *Fast Ray Tracing by Ray Classification*, Proc. of SIGGRAPH '87, p55-64, 1987. Also, Halstead MSc thesis (AU).
- But …
  - Hard to implement and gain over space subdivision is arguable

Distributed Ray Tracing

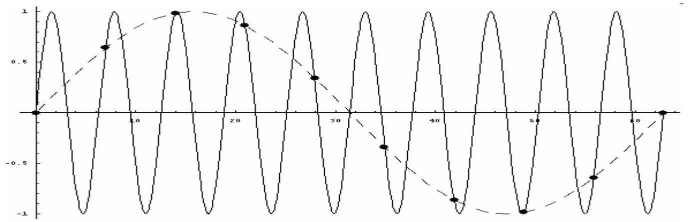
Main reference: *Stochastic Sampling in Computer Graphics”*. Rob Cook. ACM Transactions on Graphics 5 1 Jan 1986, pp 51-72.

Point Sampling

- Ray tracing is a *POINT SAMPLING* process
  - A pixel colour is a sample along a single ray
  - Shadow test is for a point light source
  - Reflected/refracted ray is a sample of the incoming light in a single direction
- All of these are WRONG!
  - A pixel colour should be an average colour for the region around the pixel
  - Real light sources have area – they aren’t points
  - Real surfaces aren’t perfect mirrors – there is some scattering involved.

Point Sampling and Aliasing

- Consider point-sampling both a high-frequency and a low-frequency signal



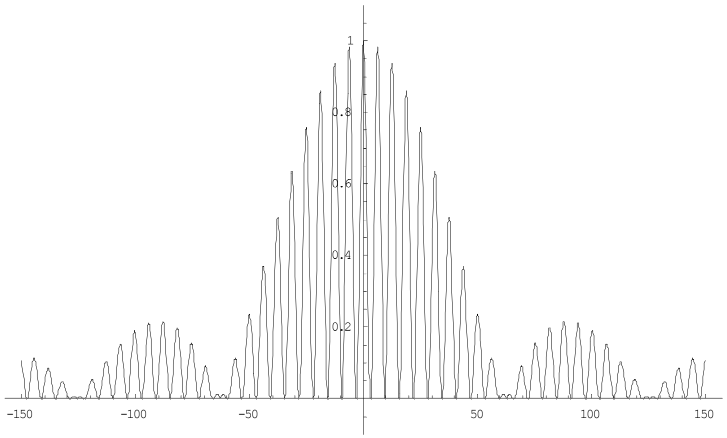
- Both signals give the same sample sets!
  - They are said to be "aliases" of each other
- The impression we get of a low frequency signal from a set of samples of a high-frequency signal is called an *aliasing artifact*
- *jaggies* on edges and *Moiré patterns* when sampling repetitive signals (e.g. texture) are examples of aliasing artifacts.

Principles of antialiasing

- Aliasing artifacts are a result of ignoring the *Sampling Theorem*
  - *If you want to be able to unambiguously reconstruct a signal from its samples, the sample frequency must be at least twice the highest frequency present in the signal*
- Graphics signals (i.e. images) are fundamentally discontinuous, i.e. have infinite frequencies present
- So solution is to "filter out" the high frequencies before sampling
- But – only need filtered value at sample points
- So effectively what we need is some sort of weighted average of the image in the neighbourhood of the sample point.

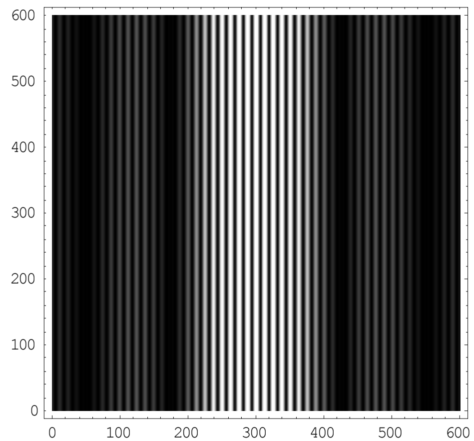
A Test Function

f(x,y)=(1+sin(x))abs(sinc(x/20))



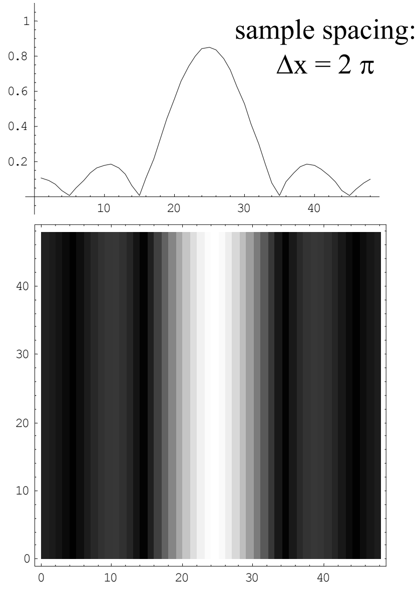
The test function as an image

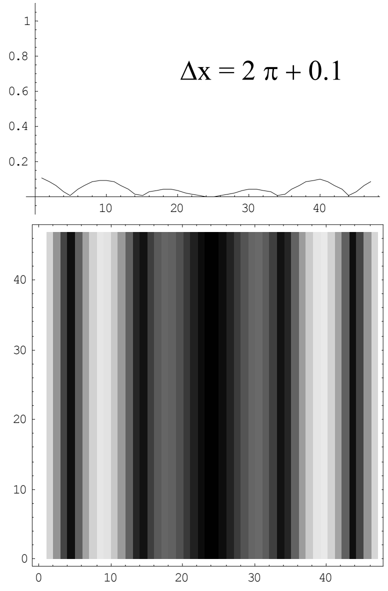
f(x,y)=(1+sin(x))abs(sinc(x/20))



Finely sampled over same x range, i.e. [-150,150]. Axis labels are just sample number aka pixel number.

The point-sampled test function



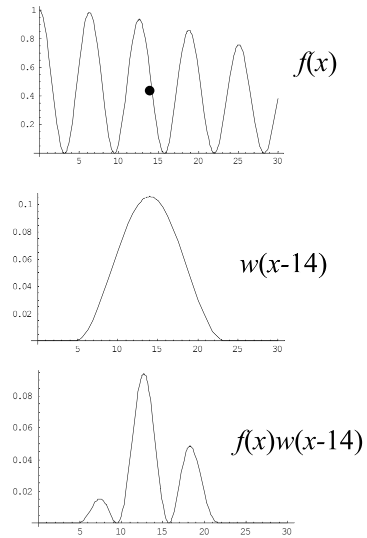


Filtering

- Rather than point sampling we should compute an average around the point.

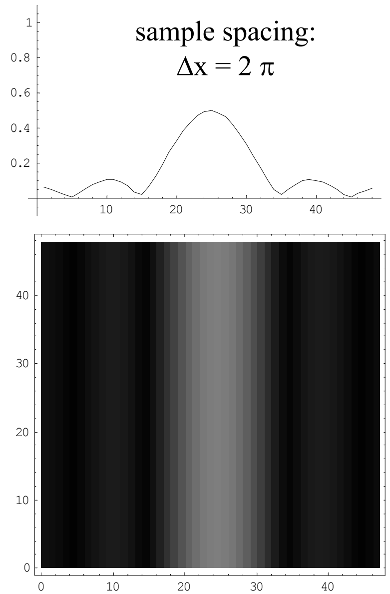
$$\overline{f}(x_0) = \int f(x)w(x - x_0)dx$$

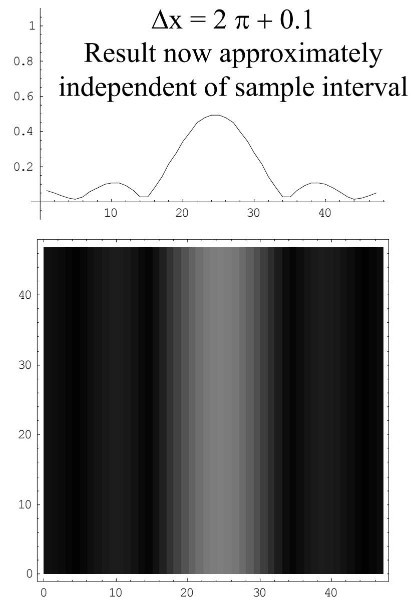
- The weighting function w is called a *filter*.
  - Must be normalised so its integral is 1



Integral is 0.45. Plotted as dot above.

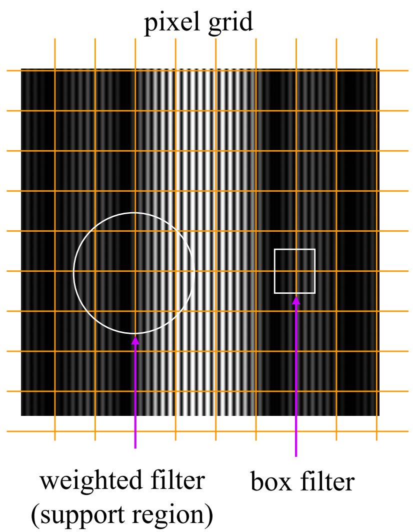
Filtered test function





Relevance to ray tracing

- If scene has fine structure, need to compute an average colour around pixel centre.
- *Box filter* is often used
  - $w(x,y) = 1$ within the square, 0 elsewhere
  - *Easy but bad*
- Weighted filters much better
- Radius typically 1.5-1.7 pixel intervals.

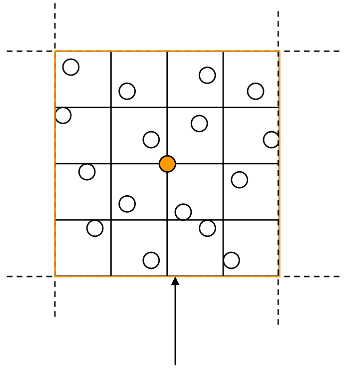


How to compute the average?

- How can we compute an average colour when all we can do is get point samples (one per ray?)
- Answer: use *Monte Carlo integration* aka *Stochastic Sampling*
- Distribute rays "randomly" over the filter region
- But to avoid clumping, subdivide filter area, take one randomly positioned sample from each subregion.
  - Statisticians calls this *stratified sampling*

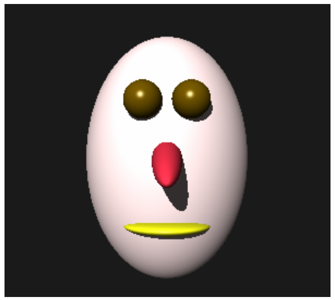
Stochastic sampling with box filter

- aka "jittered grid" sampling
- Subdivide square region around pixel into an n x n subgrid
- Take one sample from each subregion.
  - Typically just uniformly distributed
  - But can use Gaussian distribution
- Box filter is poor at removing Moiré patterns but reasonably good for jaggies.

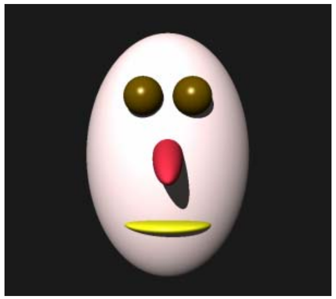


one pixel region

Example

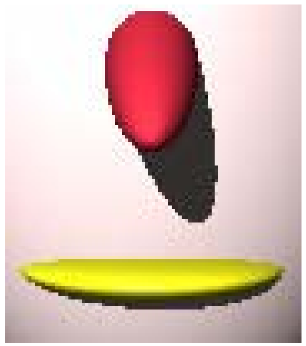


No antialiasing

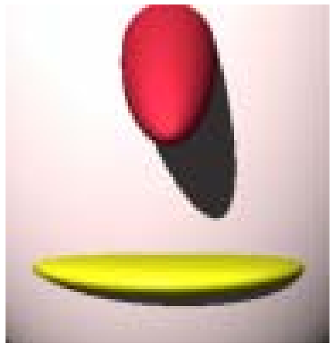


3 x 3 supersampling

Example close up



No antialiasing



3 x 3 supersampling

Stochastic sampling with a weighted filter

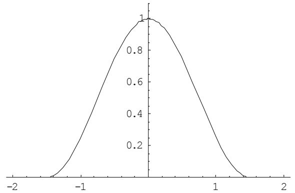
- Could subdivide filter into equal-area region and weight samples but that's wasteful.
  - Samples near the boundary get very little weighting
- Instead use *importance sampling*
- Break filter area into regions with equal *integral* of the weight function.
- Take one sample from each region
- Just average the samples (no weighting needed)

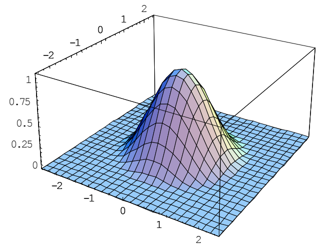
Stochastic sampling with a weighted filter (cont'd)

- Consider cylindrically symmetric *Hamming* filter

$$w(r) = k\left(1 + \cos\left(\frac{\pi r}{r_{max}}\right)\right) \quad \text{where } r = \sqrt{x^2 + y^2}$$

  - k is a normalization constant

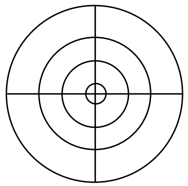


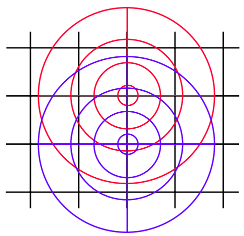


- For importance sampling, need to carve this into equal *volume* portions.

Stochastic sampling with a weighted filter (cont'd)

- e.g. 4 quadrants, four annuli => 16 regions
- Integral of filter = 1/16 for each region
  - So total integral = 1
    - Required for normalization
  - UDOO: compute the radii of the annuli and the normalization constant k
    - *They're NOT equally spaced*
- Take one random sample from within each region
  - Random azimuth angle, but statistical distribution in r is a bit tricky
    - Why?
- Produces excellent filtering BUT because filters overlap, sampling is very wasteful.

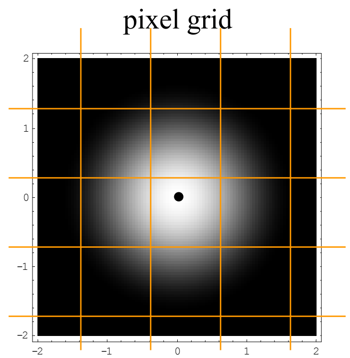




A better way of using weighted filters for antialiasing in ray tracing

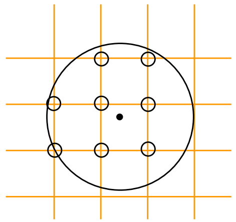
[Unpublished method due to Brian Smits.]

- Take uniformly distributed samples as for box filtering
- Composite (i.e. add) each sample into the image using the weighted filter as a *footprint* function to weight the sample
  - A technique related to *splatting* in volume visualization
  - Splat is centred on the sample point



The *footprint* of the Hamming filter ($r = 1.7$)

Compositing the footprint



- Footprint covers several pixels
  - Remember: r_{max} typically 1.5 – 1.7
  - Need to compute weights for each covered pixel
- Want Sum[weights]=1/NumSamplesPerPixel
- "Obvious" method:

$$weight_{pixel} = \frac{w(\|pixelCentre - sampleCentre\|)}{NumSamplesPerPixel \int w\left(\sqrt{x^2+y^2}\right)dxdy}$$

- Introduces some noise
  - But probably not noticeable?
  - Variation of sum of weights with sample position for a Hamming filter is given in the next slide

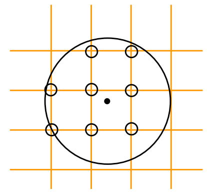
Compositing the footprint (cont'd)

Table: variation of sample sum with sample centre

Filter	r_{max}	Integral	MinSumSamples	MaxSumSamples
Hamming	1.50	2.10	0.97	1.04
Hamming	1.70	2.70	0.97	1.01
Hamming	2.00	2.74	0.99	1.01
Cone	1.50	2.36	0.90	1.09
Cone	1.70	3.03	0.95	1.10
Cone	2.00	4.19	0.98	1.02

"Cone" filter is:

$$r = \begin{cases} 0 & r \geq r_{\max} \\ 1 - \dfrac{r}{r_{\max}} & otherwise \end{cases}$$



Distributed Ray Tracing

- Stochastic sampling is a way of computing integrals
  - For antialiasing, the integral is the weighted image colour
- There are other integrals involved in ray tracing:
  - Temporal antialiasing
  - The *Rendering Equation*
  - Depth of field

Temporal Antialiasing

- If scene contains moving objects and we sample at regular fixed time intervals (every frame in an animation) we can get *temporal aliasing:*
  - Motion appears jerky
    - = "jaggies" in the time dimension
  - Can get stroboscopic effects
    - In movies, wagon wheels go backwards
    - Moving fans and machinery can appear stationary when viewed with pulsing light source
    - = "Moiré patterns" in time
- Solution is to distribute samples in time as well as space
  - Image is then averaged over the frame time
  - Corresponds to "exposure time" in movie camera
  - Gives "motion blur"

Temporal Antialiasing (cont'd)

- Simplistic way: choose random time in range [t – T/2, t + T/2] for each supersample ray, where T = interFrameTime.
  - Set positions/orientations of all moving scene objects (and maybe the camera) to correspond to that time
- Much better to use stratified sampling
  - i.e., supersampling in time, with jitter
  - Avoids "clumping" of samples in time
  - If have *n* spatial samples/pixel, subdivide the frame time into *n* "subframes" too
  - Randomly map the *n* subframes onto the *n* spatial supersamples
    - Jitter each temporal supersample
  - Cook suggests pre-computing the map [see Fig. 8 in handout].
- Note that this is "box filtering" in time
  - For better results could use weighted filter in time, too.

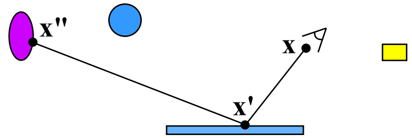
The Rendering Equation

F&vD, section 16.11

- Defines the colour I of a surface at some wavelength

$$I(\mathbf{x},\mathbf{x}') = g(\mathbf{x},\mathbf{x}')\left[\varepsilon(\mathbf{x},\mathbf{x}') + \int_S \rho(\mathbf{x},\mathbf{x}',\mathbf{x}'')I(\mathbf{x}',\mathbf{x}'')d\mathbf{x}''\right]$$

- In words: the intensity I of some surface point **x'** when seen from some viewpoint **x** is the product of the geometric visibility g [0 or $1/r^2$] of **x '** from **x** times the sum of the light ε emitted from **x'** towards **x** and the total light reflected towards **x** from **x '** from all points in the environment.
- The latter term is obtained by integrating all the light (photons) that comes from a point **x''**, hits point **x** and reflects towards **x** over all points **x''.**
- The reflection function ρ is the "Bi-directional Reflectance Distribution Function", or BRDF.



Global Illumination

- The rendering equation is recursive
  - e.g. light illuminates the floor, floor illuminates the ceiling, ceiling illuminates the floor
- Illumination calculation methods that attempt to solve the interreflections (usually with major restrictions) are called *global illumination algorithms*.
  - Most common class is *radiosity algorithms* which typically solve the light levels assuming most surfaces are diffuse reflectors.
- Ray tracing is sometimes called a global illumination method
  - but to earn the name it should do something a bit better than casting just simple shadow and reflection rays in my opinion!

Relevance to Ray Tracing

- Colour of point hit by primary ray should really by obtained by integrating over all incoming directions rather than just the directions to the point light sources and a mirror reflection direction.
- Impractical in general
  - But see the *Radiance* ray tracer by Greg Ward [SIGGRAPH Proc. 88]
    - Does a radiosity solution and gets actual physical illumination levels
    - Used in architecture
- More easily, though:
  - can get soft shadows (umbrae + penumbrae) by stochastically sampling over the area of non-point light sources (e.g. fluorescent tubes)
  - can get blurry reflections by integrating over a *range* of directions around the mirror reflection
  - can get better specular highlights of lights (including area light sources)
    - Can use much better reflection models/BRDFs than Phong
    - See F&vD

Sampling the extra dimensions

- Often don’t need to cast any new rays!
  - Assuming antialiasing is already being used
- Just make sure that the multiple rays cast for each pixel are also distributed over the other dimensions
  - e.g. if doing *n* by *n* supersampling, might subdivide area light source into *n* by *n* subareas, randomly map supersamples to these subareas when doing shadow testing.
- But may need more rays if sampling introduces too much noise, e.g. a very large light source and/or multiple partial occluders.

Depth of Field

- Camera photos have a focal plane over which scene objects are in focus.
- Objects are successively defocussed away from that plane.
- Easily simulated with stochastic sampling
  - for each pixel
    - Determine focal point of pixel in lens (stage 1 physics lens formula)
    - Distribute supersampling rays over the lens area
    - Cast rays from jittered points on lens into the scene *through the in-focus point*