# TIEA311 Tietokonegrafiikan perusteet
kevät 2019

("Principles of Computer Graphics" – Spring 2019)

# TIEA311 Tietokonegrafiikan perusteet – kevät 2019 (“Principles of Computer Graphics” – Spring 2019)

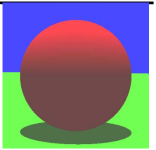
Adapted from: *Wojciech Matusik*, and *Frédo Durand*: 6.837 Computer Graphics. Fall 2012. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu/.



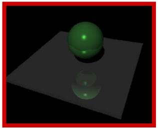
Frontpage of the local course version, held during Spring 2019 at the Faculty of Information technology, University of Jyväskylä:
`http://users.jyu.fi/~nieminen/tgp19/`

# Overview of Today

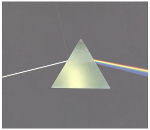
- Shadows



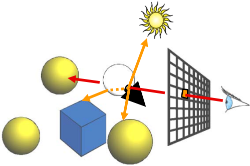
- Reflection



- Refraction

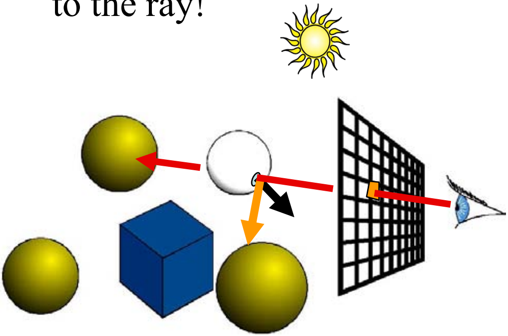


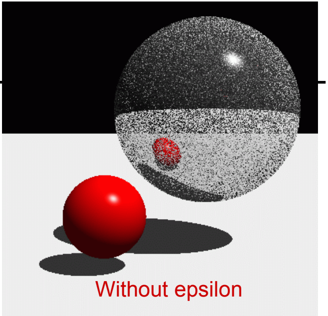
- Recursive Ray Tracing

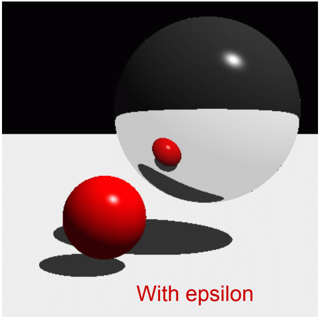


# Mirror Reflection

- Cast ray symmetric with respect to the normal
- Multiply by reflection coefficient $k_s$ (color)
- Don't forget to add epsilon to the ray!

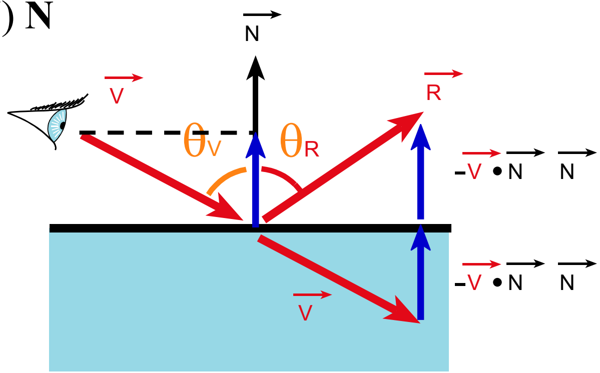




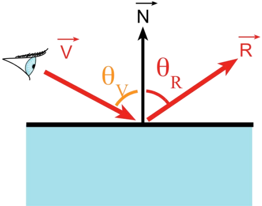
# Perfect Mirror Reflection

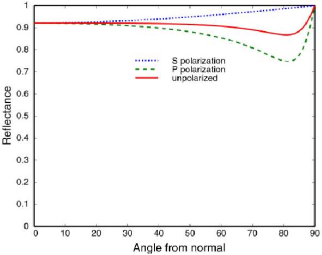
- Reflection angle = view angle
  - Normal component is negated
  - Remember particle collisions?
- $\mathbf{R} = \mathbf{V} - 2\,(\mathbf{V} \cdot \mathbf{N})\,\mathbf{N}$



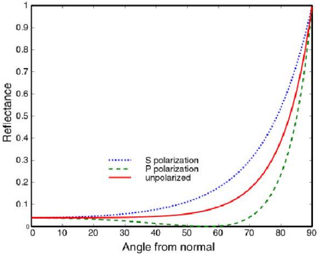
# Amount of Reflection


- Traditional ray tracing (hack)
  - Constant $k_s$
- More realistic (we'll do this later):
  - Fresnel reflection term (more reflection at grazing angle)
  - Schlick's approximation: $R(\theta)=R_0+(1-R_0)(1-\cos\theta)^5$
- Fresnel makes a big difference!




metal

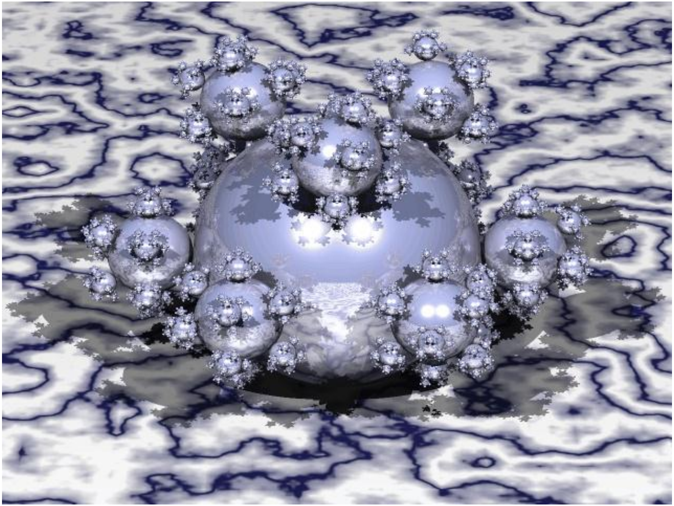



Dielectric (glass)

# Questions?

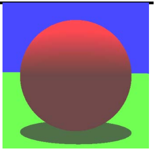
"Sphereflake" fractal

Henrik Wann Jensen

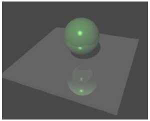


# Overview of Today

- Shadows

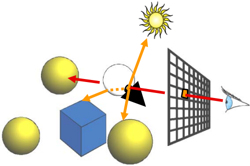


- Reflection



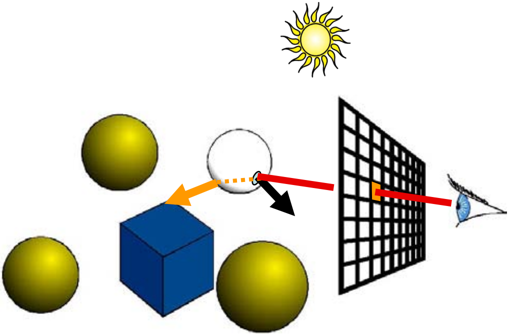
- Refraction

- Recursive Ray Tracing

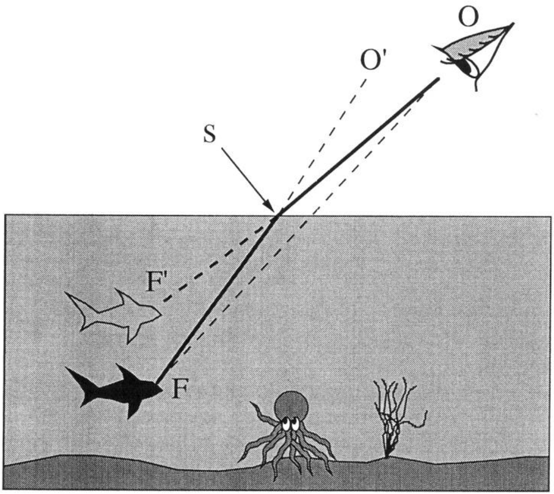


# Transparency (Refraction)

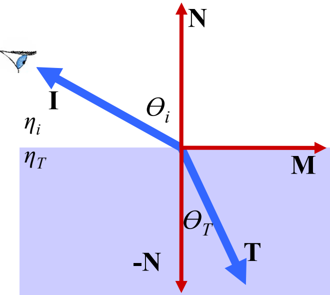
- Cast ray in refracted direction
- Multiply by transparency coefficient $k_t$ (color)



# Qualitative Refraction



# Refraction




Material 1, index of refraction $\eta_i$

Material 2, index of refraction $\eta_T$

**Snell-Descartes Law:**

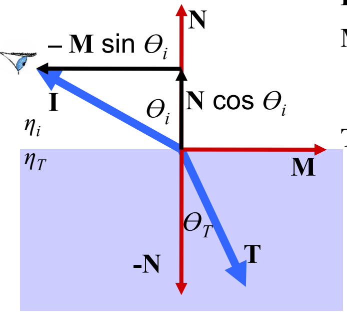
$$n_i \sin\theta_i = n_T \sin\theta_T$$

Refracted direction **T**?

$$\frac{\sin\theta_T}{\sin\theta_i} = \frac{n_i}{n_T} = n_r$$

**Relative index of refraction**

# Refraction




$$\mathbf{I} = \mathbf{N}\cos\Theta_i - \mathbf{M}\sin\Theta_i$$

$$\mathbf{M} = (\mathbf{N}\cos\Theta_i - \mathbf{I}) / \sin\Theta_i$$

$$\mathbf{T} = -\mathbf{N}\cos\Theta_T + \mathbf{M}\sin\Theta_T$$

$$= -\mathbf{N}\cos\Theta_T + (\mathbf{N}\cos\Theta_i - \mathbf{I})\sin\Theta_T / \sin\Theta_i$$ *Plug M*

$$= -\mathbf{N}\cos\Theta_T + (\mathbf{N}\cos\Theta_i - \mathbf{I})\,\eta_r$$

*let's get rid of the cos & sin*

$$= [\,\eta_r\cos\Theta_i - \cos\Theta_T\,]\,\mathbf{N} - \eta_r\mathbf{I}$$

$$= [\,\eta_r\cos\Theta_i - \sqrt{1-\sin^2\Theta_T}\,]\,\mathbf{N} - \eta_r\mathbf{I}$$

$$= [\,\eta_r\cos\Theta_i - \sqrt{1-\eta_r^2\sin^2\Theta_i}\,]\,\mathbf{N} - \eta_r\mathbf{I}$$

$$= [\,\eta_r\cos\Theta_i - \sqrt{1-\eta_r^2(1-\cos^2\Theta_i)}\,]\,\mathbf{N} - \eta_r\mathbf{I}$$

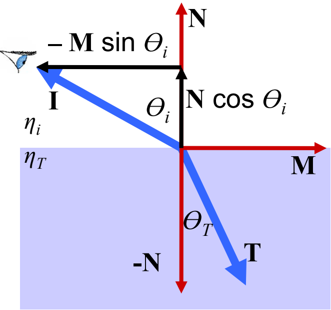
$$= [\,\eta_r(\mathbf{N}\cdot\mathbf{I}) - \sqrt{1-\eta_r^2(1-(\mathbf{N}\cdot\mathbf{I})^2)}\,]\,\mathbf{N} - \eta_r\mathbf{I}$$

**Snell-Descartes Law:**

$$n_i \sin\theta_i = n_T \sin\theta_T$$

$$\frac{\sin\theta_T}{\sin\theta_i} = \frac{n_i}{n_T} = n_r$$

# Refraction




$$\mathbf{I} = \mathbf{N}\cos\Theta_i - \mathbf{M}\sin\Theta_i$$

$$\mathbf{M} = (\mathbf{N}\cos\Theta_i - \mathbf{I}) / \sin\Theta_i$$

- **Total internal reflection** when the square root is imaginary (no refraction, just reflection)

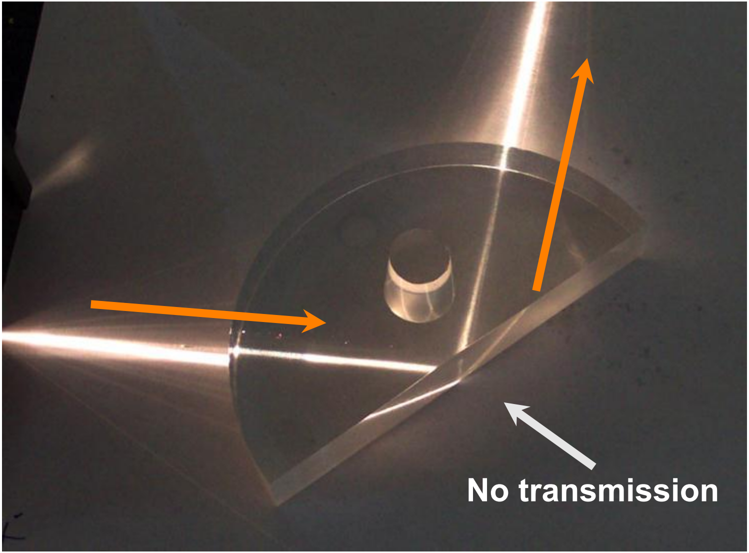
**Snell-Descartes Law:**

$$n_i \sin\theta_i = n_T \sin\theta_T$$

$$\frac{\sin\theta_T}{\sin\theta_i} = \frac{n_i}{n_T} = n_r$$

$$= [\,\eta_r(\mathbf{N}\cdot\mathbf{I}) - \sqrt{1-\eta_r^{\,2}(1-(\mathbf{N}\cdot\mathbf{I})^2)}\,]\,\mathbf{N} - \eta_r\mathbf{I}$$

# Total Internal Reflection



# Total Internal Reflection

Fig. 3.7A The optical manhole. From under water, the entire celestial hemisphere is compressed into a circle only 97.2° across. The dark boundary defining the edges of the manhole is not sharp due to surface waves. The rays are analogous to the crepuscular type seen in hazy air, Section 1.9. (Photo by D. Granger)

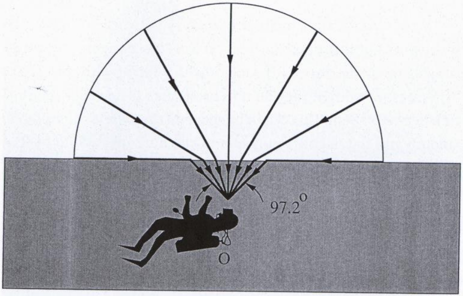
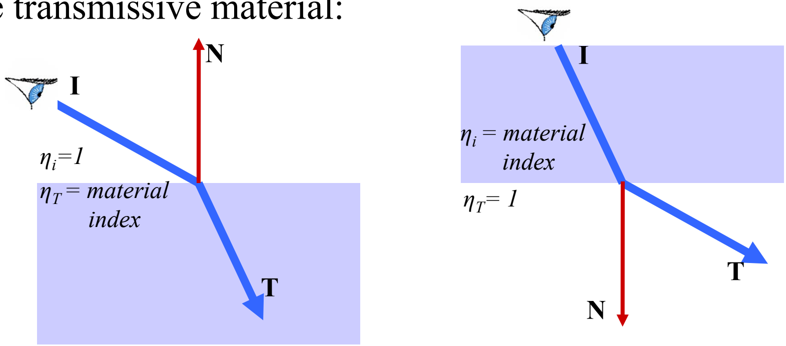



Fig. 3.7B The optical manhole. Light from the horizon (angle of incidence = 90°) is refracted downward at an angle of 48.6°. This compresses the sky into a circle with a diameter of 97.2° instead of its usual 180°.

# Refraction & Sidedness of Objects

- Make sure you know whether you're entering or leaving the transmissive material:



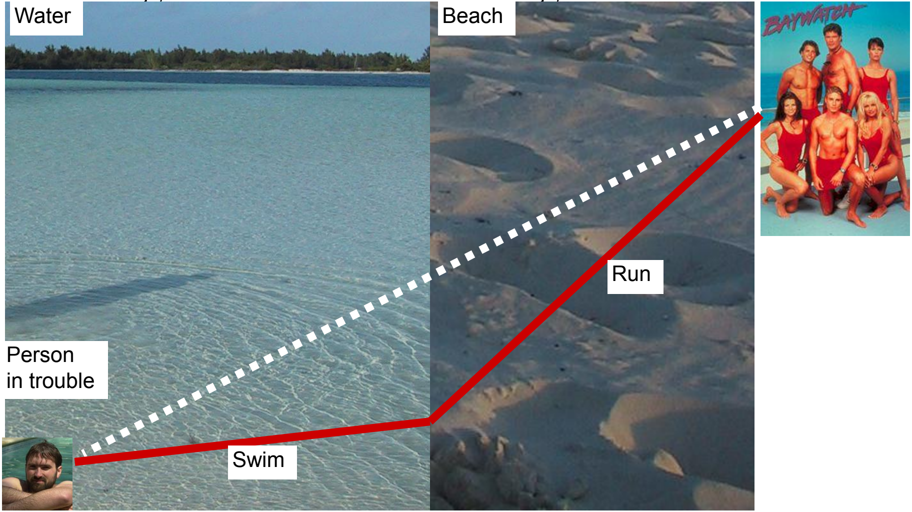

- Note: We won't ask you to trace rays through intersecting transparent objects **:-)**

# Cool Refraction Demo

- Enright, D., Marschner, S. and Fedkiw, R., SIGGRAPH 2002

# Refraction and the Lifeguard Problem

- Running is faster than swimming

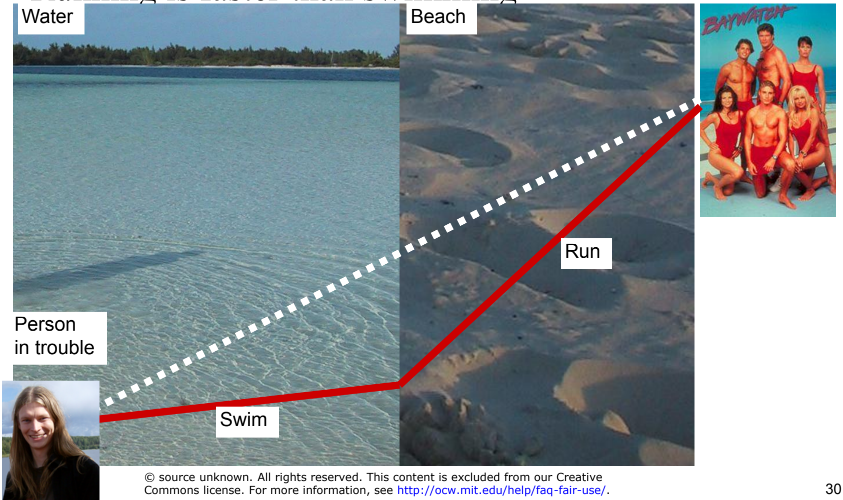


# TIEA311 - Today in Jyväskylä

Wait a moment. . . who is in trouble on this course?

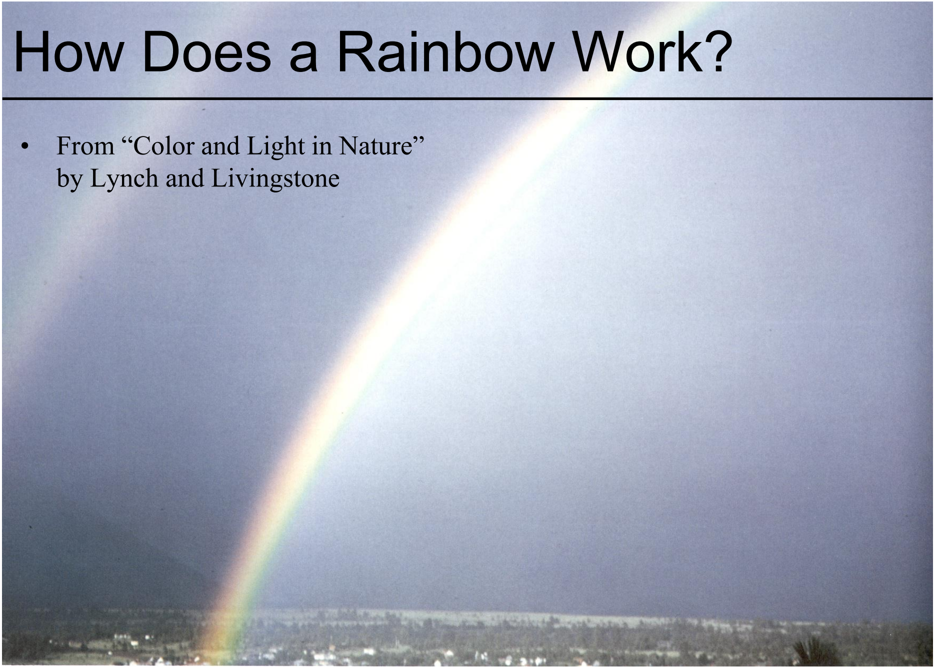
# Refraction and the Lifeguard Problem

- Running is faster than swimming



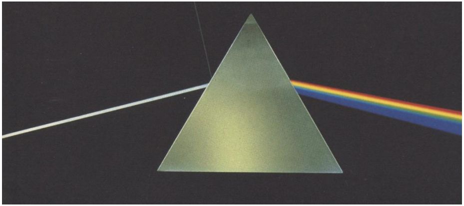
# How Does a Rainbow Work?

- From "Color and Light in Nature" by Lynch and Livingstone



# Wavelength

- Refraction is wavelength-dependent (dispersion)
  - Refraction increases as the wavelength of light decreases
  - violet and blue experience more bending than orange and red
- Newton's prism experiment
- **Usually ignored in graphics**



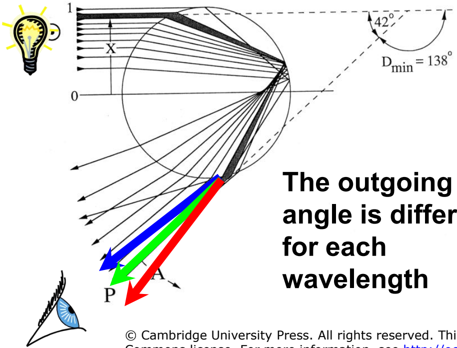
Pink Floyd, *The Dark Side of the Moon*



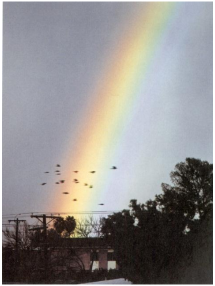
Pittoni, 1725, Allegory to Newton

# Rainbow

- Rainbow is caused by refraction + internal reflection + refraction
- Maximum for angle around 42 degrees
- Refraction depends on wavelength (dispersion)



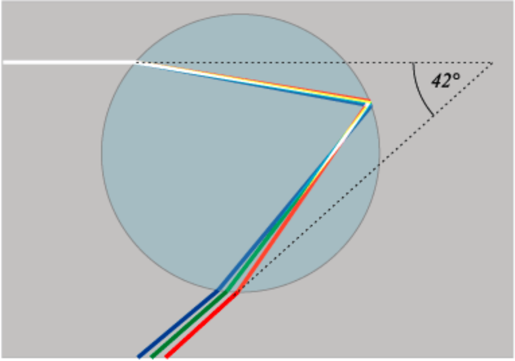

**The outgoing angle is different for each wavelength**



“Color and Light in Nature” by Lynch and Livingstone

# Rainbow

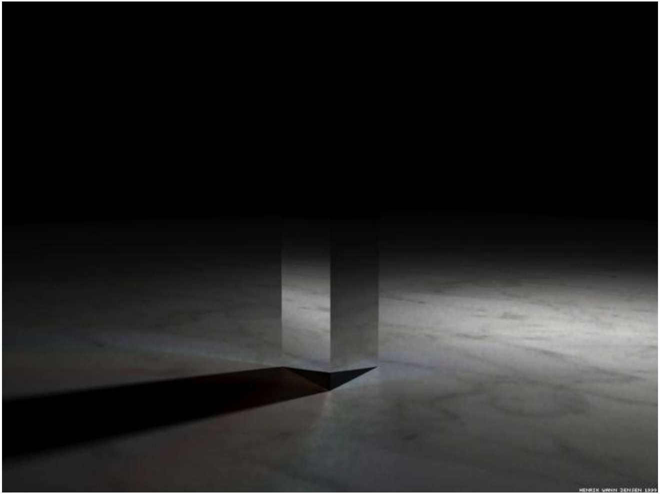
- Rainbow is caused by refraction + internal reflection + refraction
- Maximum for angle around 42 degrees
- Refraction depends on wavelength (dispersion)




This image is in the public domain. Source: Wikipedia.

# Dispersion

- Image by Henrik Wann Jensen using Photon Mapping



# Application: CAD for lenses

- Has revolutionized lens design
  - E.g. zoom lenses are good now

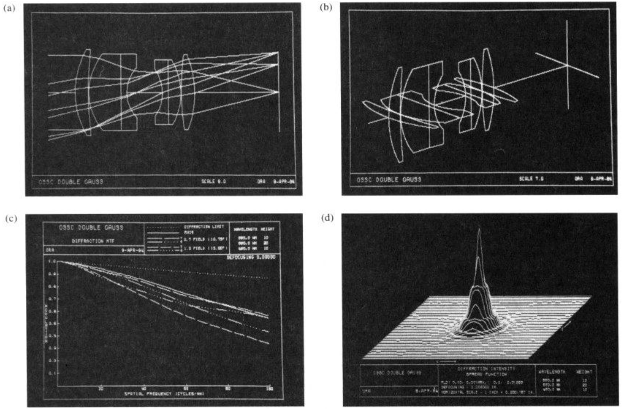
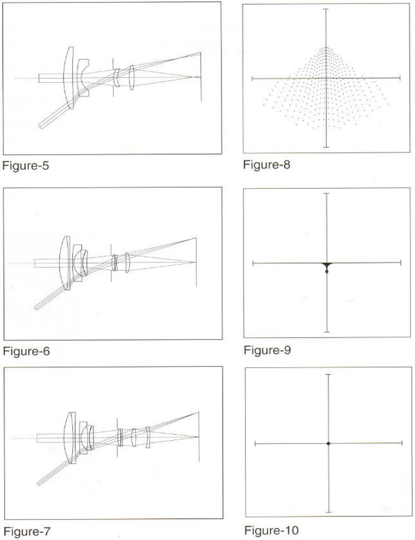



**Figure 11.50** An example of the kind of lens design information available via computer techniques. (Photos courtesy Optical Research Associates.)

From Hecht's Optics

# Lens design by Ray Tracing

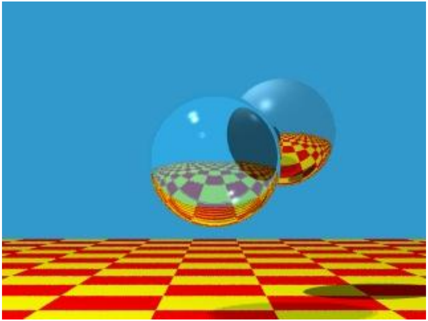
- Used to be done manually, by rooms full of engineers who would trace rays.
- Now software, e.g. Zemax
- More in 6.815/6.865 Computational Photography

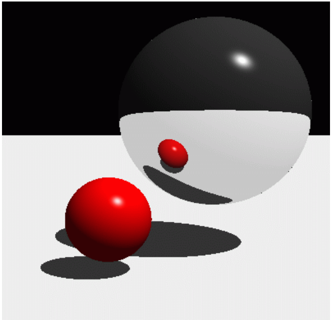



source: canon red book

# Let's Pause for a Moment...

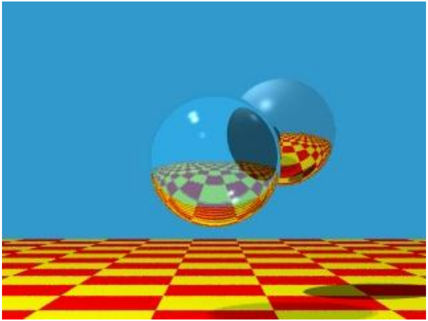
- Do these pictures look real?

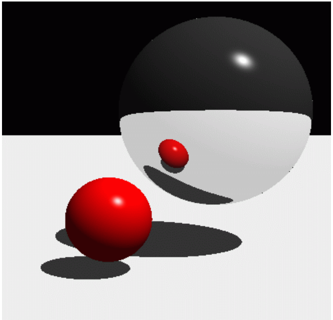




# What's Wrong then?

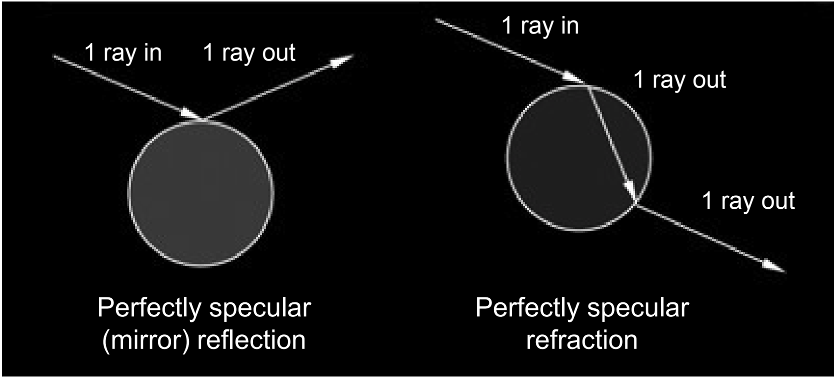
- No surface is a perfect mirror,
  no material interface is perfectly smooth





# What's Wrong then?

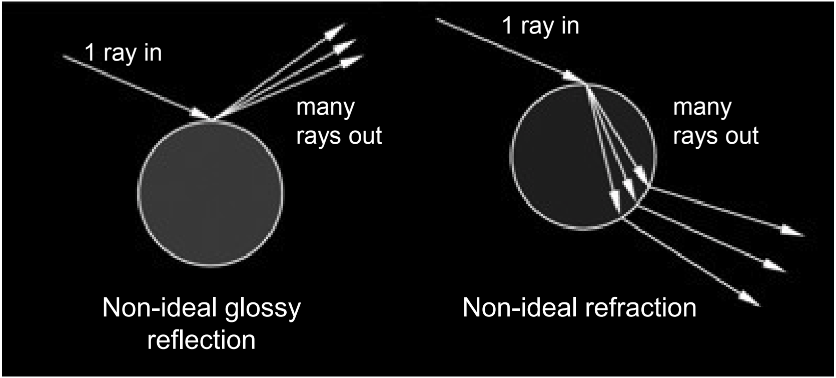
- No surface is a perfect mirror,
  no material interface is perfectly smooth




Adapted from blender.org

# Non-Ideal Reflection/Refraction

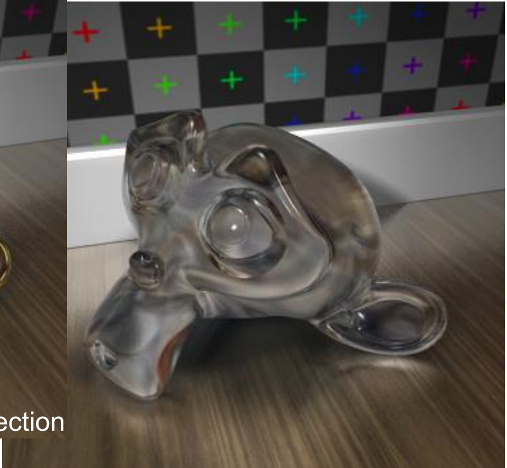
- No surface is a perfect mirror,
  no material interface is perfectly smooth




Adapted from blender.org

# Non-Ideal Reflection/Refraction

Glossy (as opposed to mirror) reflection



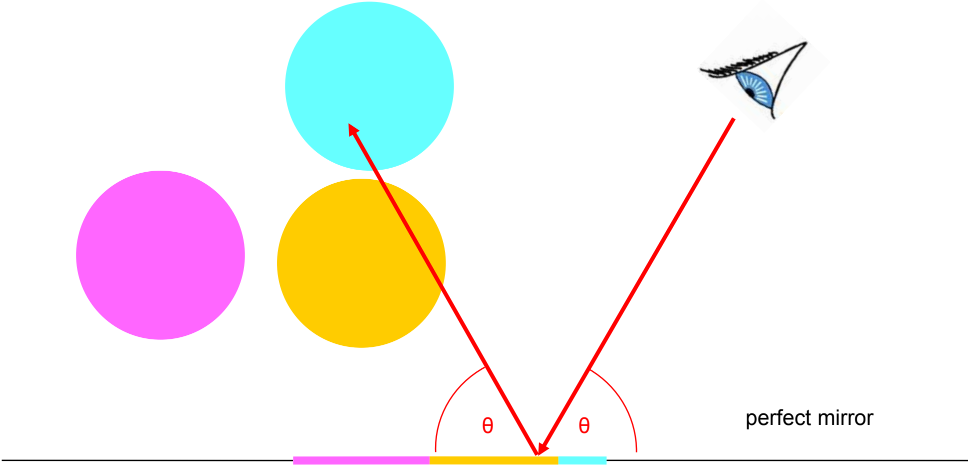
Glossy (as opposed to perfect) refraction



images from blender.org

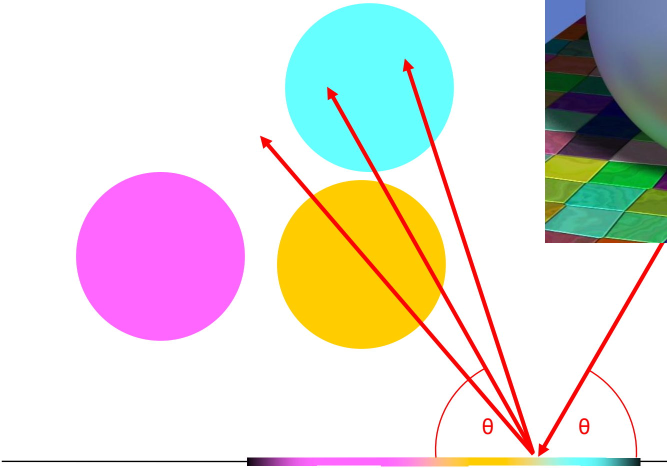
# Reflection

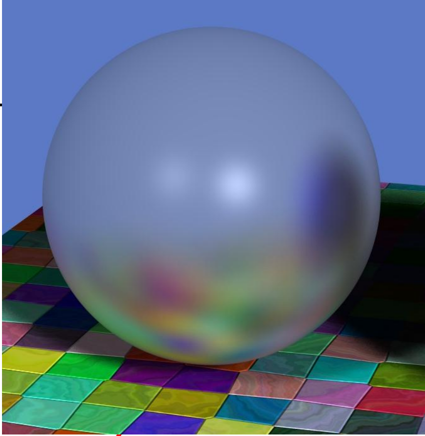
- One reflection ray per intersection



# Glossy Reflection

- Multiple reflection rays



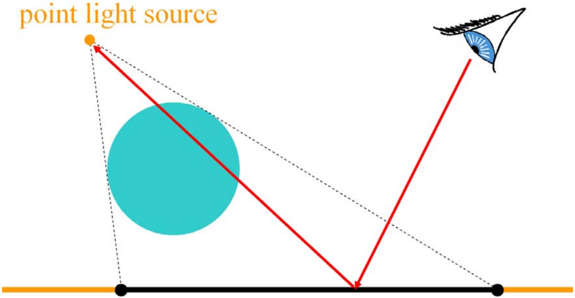



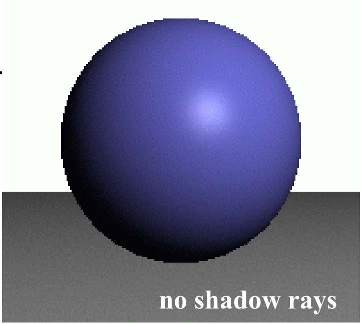
Courtesy of Justin Legakis.

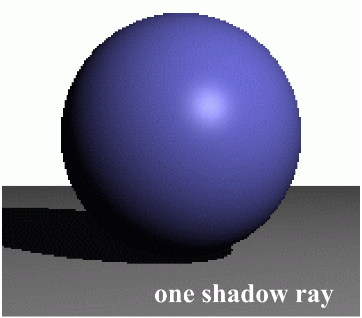
Justin Legakis

# Shadows

- One shadow ray per intersection per point light source



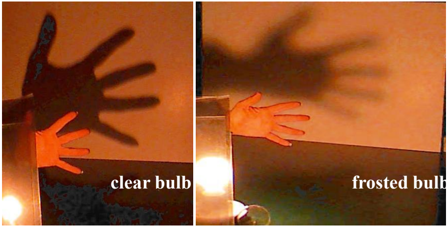



# Shadows & Light Sources

Image removed due to copyright restrictions.



**http://www.davidfay.com/index.php**



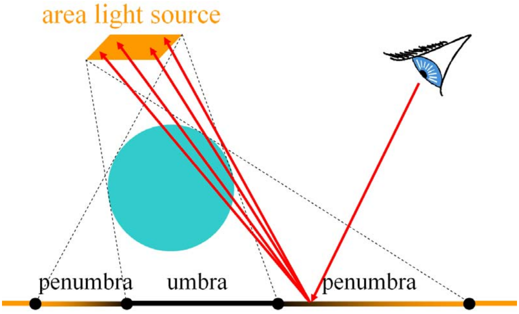

**http://3media.initialized.org/photos/2000-10-18/index_gall.htm**

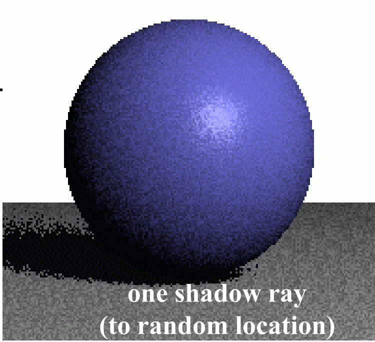


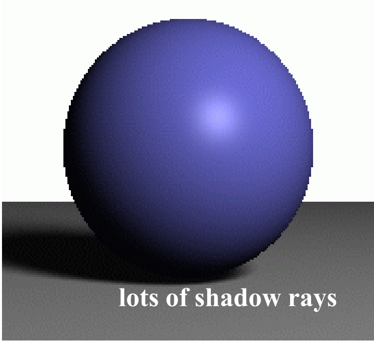
**http://www.pa.uky.edu/~sciworks/light/preview/bulb2.htm**

# Soft Shadows

- Multiple shadow rays to sample area light source

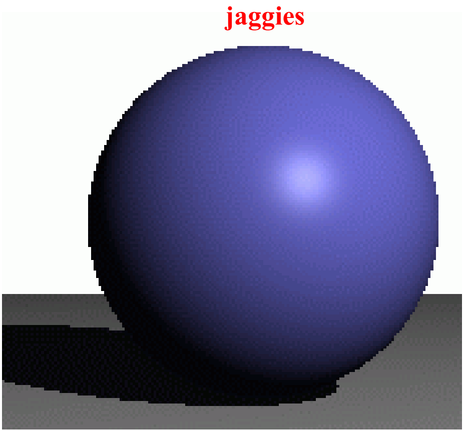


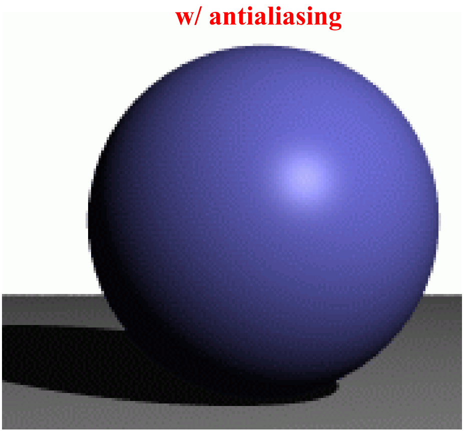



# Antialiasing – Supersampling

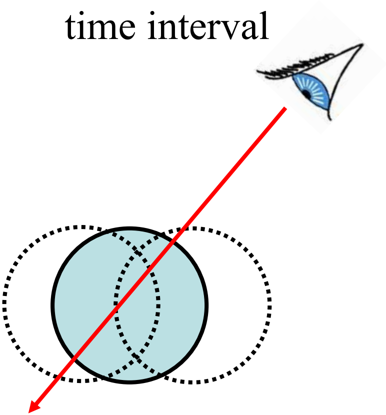
- Multiple rays per pixel



# Motion Blur

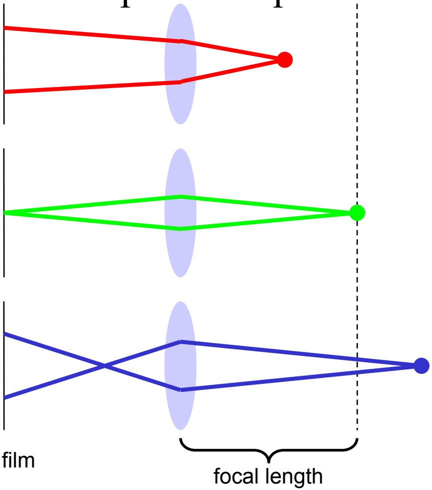
- Sample objects temporally over time interval

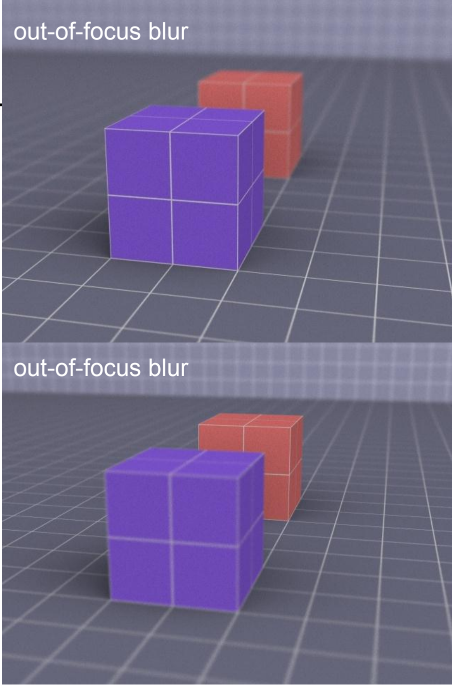


Rob Cook

# Depth of Field

- Multiple rays per pixel: sample lens aperture



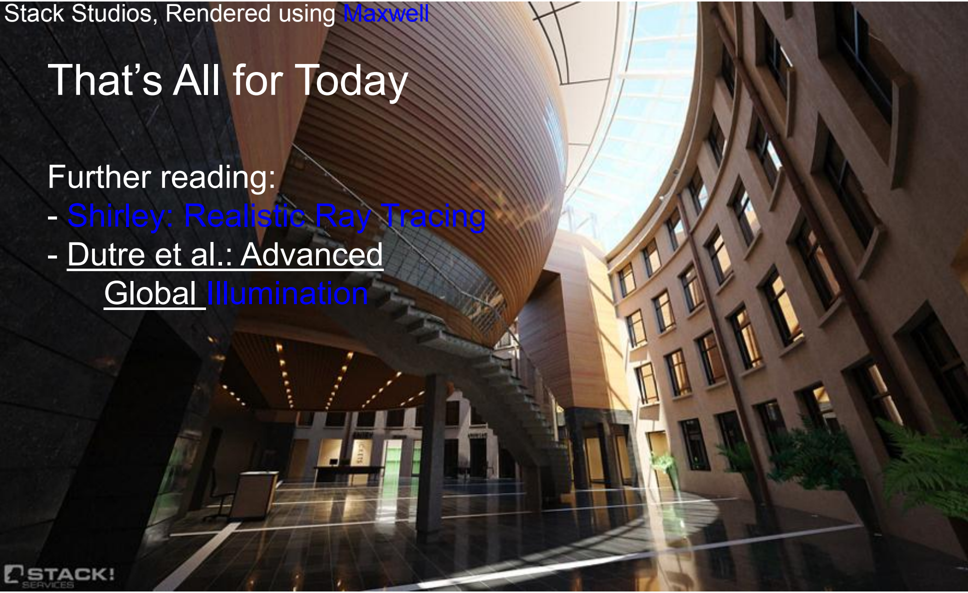




Justin Legakis

# Questions?

Henrik Wann Jensen


Courtesy of Henrik Wann Jensen. Used with permission.

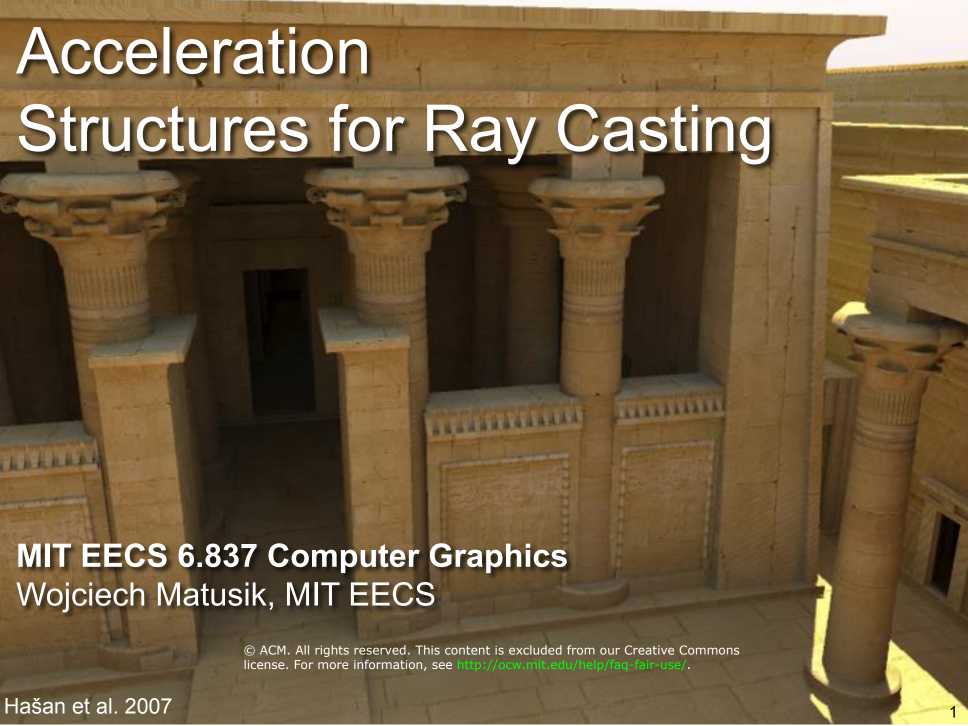



Stack Studios, Rendered using Maxwell

# That’s All for Today

Further reading:

- Shirley: Realistic Ray Tracing
- Dutre et al.: Advanced Global Illumination



# Acceleration Structures for Ray Casting

**MIT EECS 6.837 Computer Graphics**

Wojciech Matusik, MIT EECS

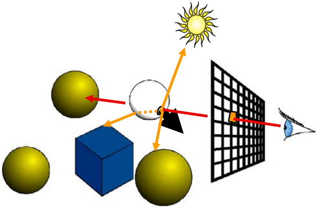


Hašan et al. 2007

# Recap: Ray Tracing

```
trace ray
    Intersect all objects
    color = ambient term
    For every light
        cast shadow ray
        color += local shading term
    If mirror
        color += color_refl *
                  trace reflected ray
    If transparent
        color += color_trans *
                 trace transmitted ray
```

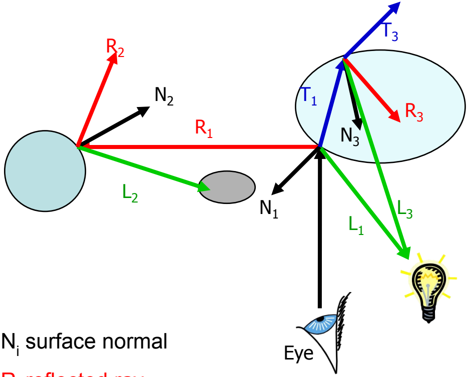
- ***Does it ever end?***



Stopping criteria:

- Recursion depth
  - Stop after a number of bounces
- Ray contribution
  - Stop if reflected / transmitted contribution becomes too small

# The Ray Tree

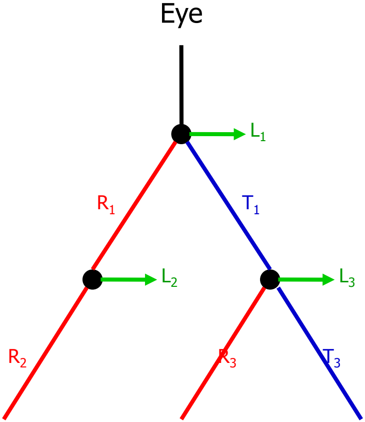



$N_i$ surface normal

$R_i$ reflected ray

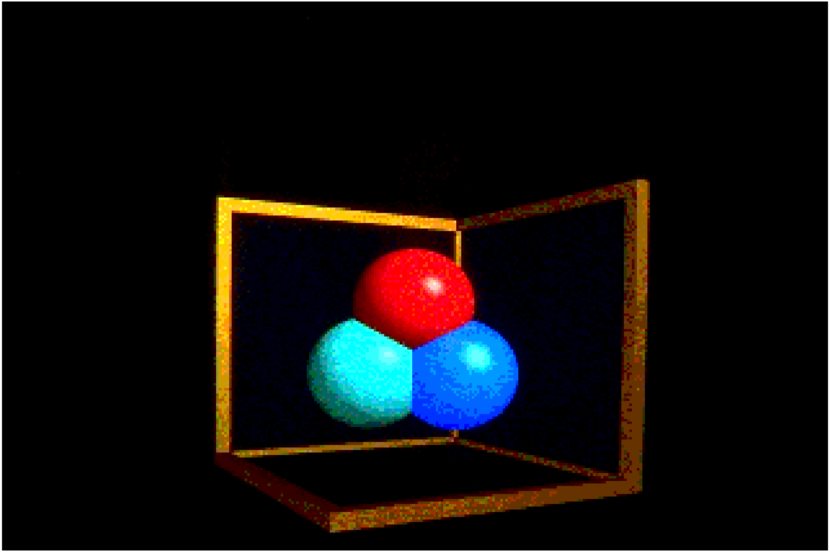
$L_i$ shadow ray

$T_i$ transmitted (refracted) ray

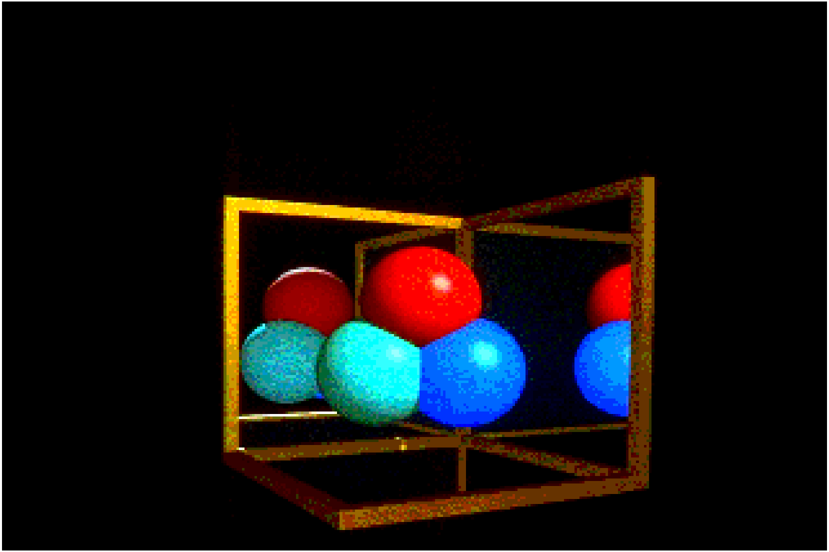



Complexity?

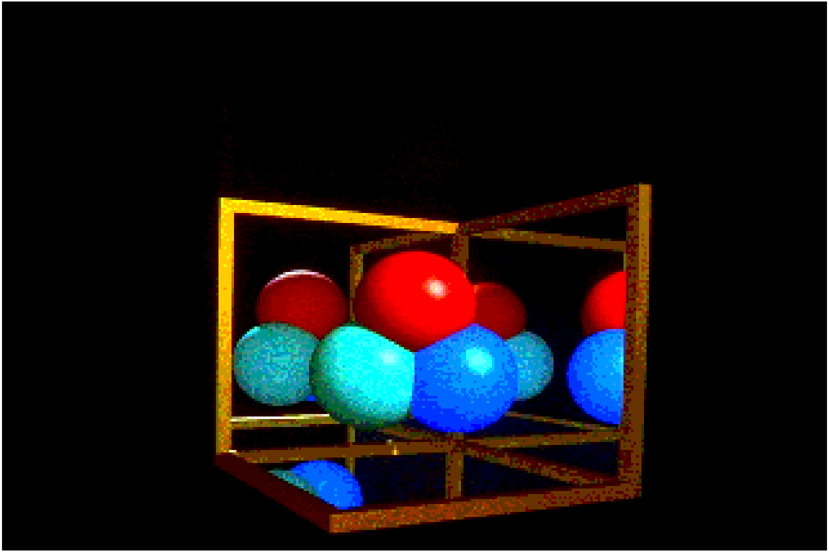
# Recursion For Reflection: None



# Recursion For Reflection: 1

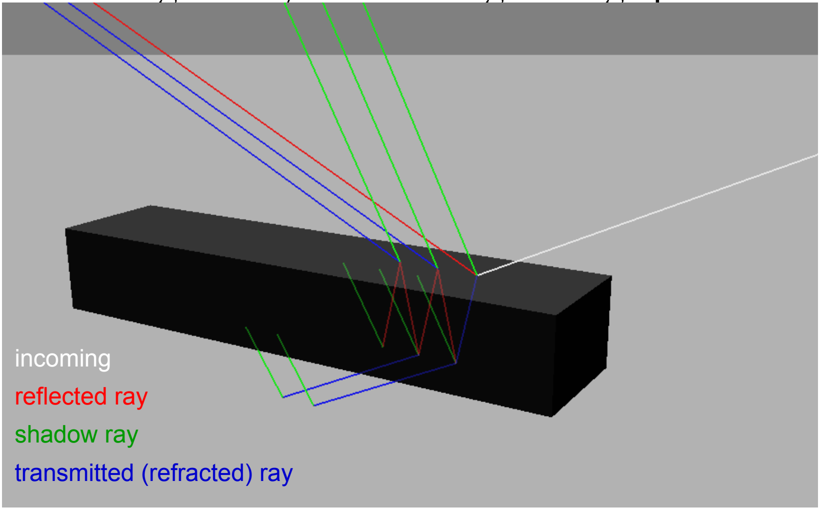


# Recursion For Reflection: 2



# Ray tree

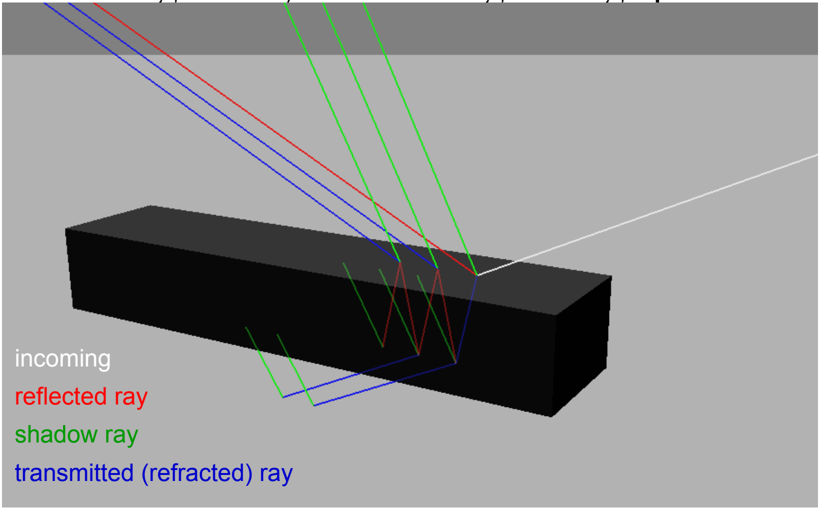
- Visualizing the ray tree for single image pixel



# Ray tree

This gets pretty complicated pretty fast!

- Visualizing the ray tree for single image pixel



# Ray Tracing Algorithm Analysis

- Lots of primitives
- Recursive
- Distributed Ray Tracing
  - Means using many rays for non-ideal/non-pointlike phenomena
    - Soft shadows
    - Anti-aliasing
    - Glossy reflection
    - Motion blur
    - Depth of field

cost ≈ height * width *
num primitives *
intersection cost *
size of recursive ray tree *
num shadow rays *
num supersamples *
num glossy rays *
num temporal samples *
num aperture samples *
. . .

**Can we reduce this?**

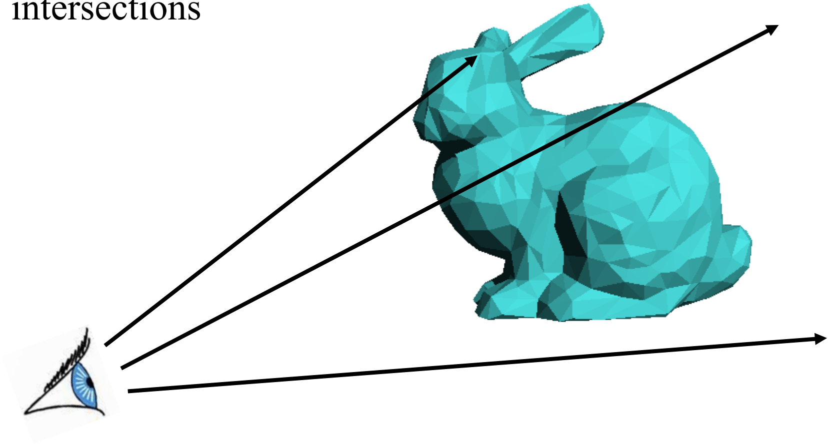
# Today

- Motivation
  - You need LOTS of rays to generate nice pictures
  - Intersecting every ray with every primitive becomes the bottleneck
- Bounding volumes
- Bounding Volume Hierarchies, Kd-trees

```
For every pixel
  Construct a ray from the eye
  For every object in the scene
     Find intersection with the ray
     Keep if closest
      Shade
```

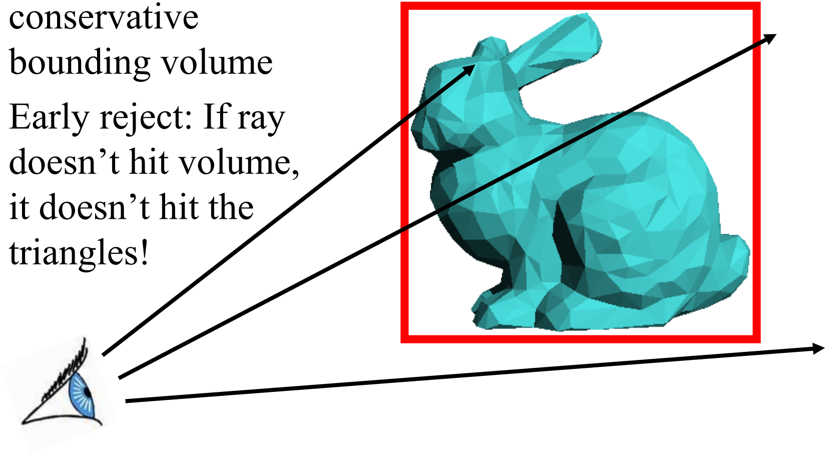
# Accelerating Ray Casting

- Goal: Reduce the number of ray/primitive intersections



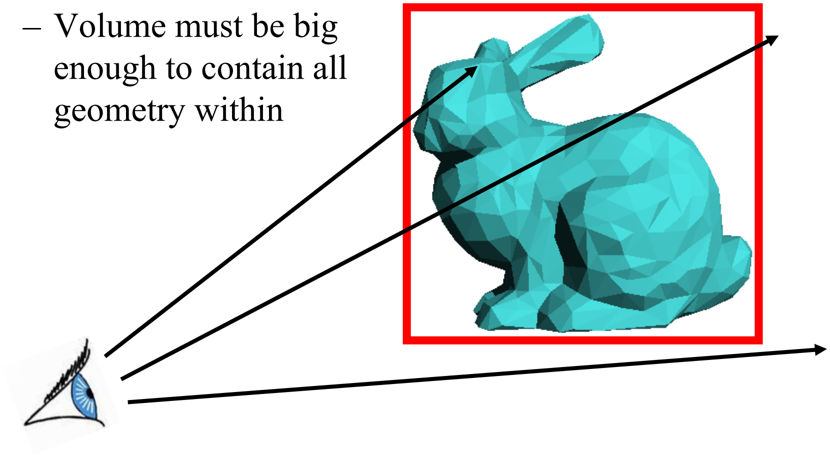
# Conservative Bounding Volume

- First check for an intersection with a conservative bounding volume
- Early reject: If ray doesn't hit volume, it doesn't hit the triangles!



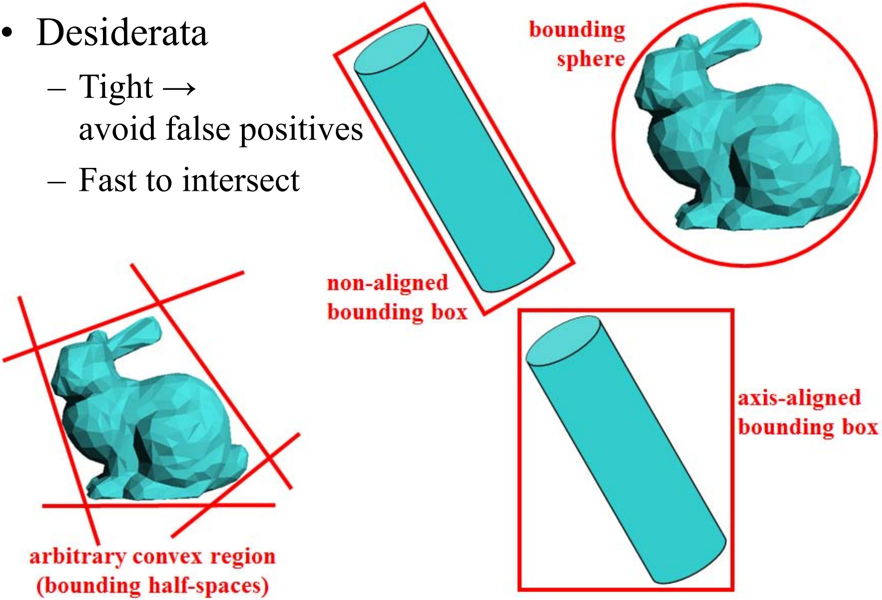
# Conservative Bounding Volume

- What does "conservative" mean?
  - Volume must be big enough to contain all geometry within



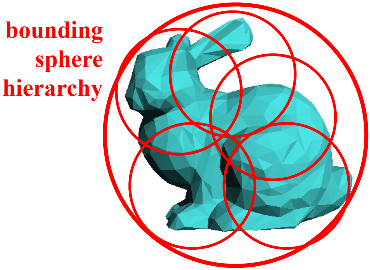
# Conservative Bounding Regions

- Desiderata
  - Tight →
    avoid false positives
  - Fast to intersect



# Bounding Volume Hierarchies

- If ray hits bounding volume, must we test all primitives inside it?
  - Lots of work, think of a 1M-triangle mesh
- You guessed it already, we'll split the primitives in groups and build recursive bounding volumes
  - Like collision detection, remember?



# TIEA311 - Today in Jyväskylä

Sorry, guys. . . If you want to learn more about this stuff, you need to do it all by yourselves. At this point, we skip a lot of material about optimizing algorithms (all sorts of similar algorithms, even if graphics is an example, btw.).

To learn more, you may want to check out (on your own time) “Lecture 14” of the original course, and some books and articles about ray tracing. Should you want to use this stuff in your own hobby projects, do ask about possibilities of getting credit points. State-of-the-art methods are good topics for Bachelor / Master thesis projects.

This course will now teleport over algorithmic optimizations. (This is, of course, called “course optimization”!) – we haven’t covered all the fundamentals yet, so let’s not get stuck with details. The next few slides from the MIT course give references for further study (not necessary for our course).

# Questions?

- Further reading on efficient Kd-tree construction
  - Hunt, Mark & Stoll, IRT 2006
  - Zhou et al., SIGGRAPH Asia 2008

Zhou et al.

# Optimizing Splitting Planes

- Most people use the Surface Area Heuristic (SAH)
  - MacDonald and Booth 1990, "Heuristic for ray tracing using space subdivision", Visual Computer
- Idea: simple probabilistic prediction of traversal cost based on split distance
- Then try different possible splits and keep the one with lowest cost
- Further reading on efficient Kd-tree construction
  - Hunt, Mark & Stoll, IRT 2006
  - Zhou et al., SIGGRAPH Asia 2008

# Hard-core efficiency considerations

- See e.g. Ingo Wald's PhD thesis
  - http://www.sci.utah.edu/~wald/PhD/
- Calculation
  - Optimized barycentric ray-triangle intersection
- Memory
  - Make kd-tree node as small as possible
    (dirty bit packing, make it 8 bytes)
- Parallelism
  - SIMD extensions, trace 4 rays at a time, mask results where they disagree

# Pros and Cons of Kd trees

- Pros
  - Simple code
  - Efficient traversal
  - Can conform to data

- Cons
  - costly construction, not great if you work with moving objects

# Questions?

- For extensions to moving scenes, see Real-Time KD-Tree Construction on Graphics Hardware, Zhou et al., SIGGRAPH 2008

# TIEA311 - Today in Jyväskylä

Sorry, guys. . . If you want to learn more about this stuff, you need to do it all by yourselves. At this point, we skip a lot of material about optimizing algorithms (all sorts of similar algorithms, even if graphics is an example, btw.).

To learn more, you may want to check out (on your own time) "Lecture 14" of the original course, and some books and articles about ray tracing. Should you want to use this stuff in your own hobby projects, do ask about possibilities of getting credit points. State-of-the-art methods are good topics for Bachelor / Master thesis projects.

This course will now teleport over algorithmic optimizations. (This is, of course, called "course optimization"!) – we haven't covered all the fundamentals yet, so let's not get stuck with details. The next few slides from the MIT course give references for further study (not necessary for our course).

# Texture Mapping & Shaders



**MIT EECS 6.837 Computer Graphics**

# Spatial Variation

- All materials seen so far are the same everywhere
  - In other words, we are assuming the BRDF is independent of the surface point **x**
  - No real reason to make that assumption





# Spatial Variation

- We will allow BRDF parameters to vary over space
  - This will give us much more complex surface appearance
  - e.g. diffuse color $k_d$ vary with x
  - Other parameters/info can vary too: $k_s$, exponent, normal

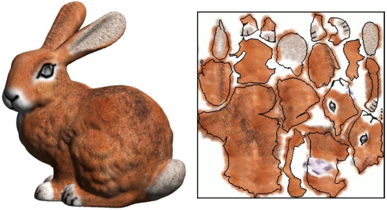




Courtesy of Fredo Durand. Used with permission.

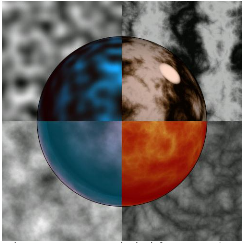
# Two Approaches

- From data : texture mapping
  - read color and other information from 2D images

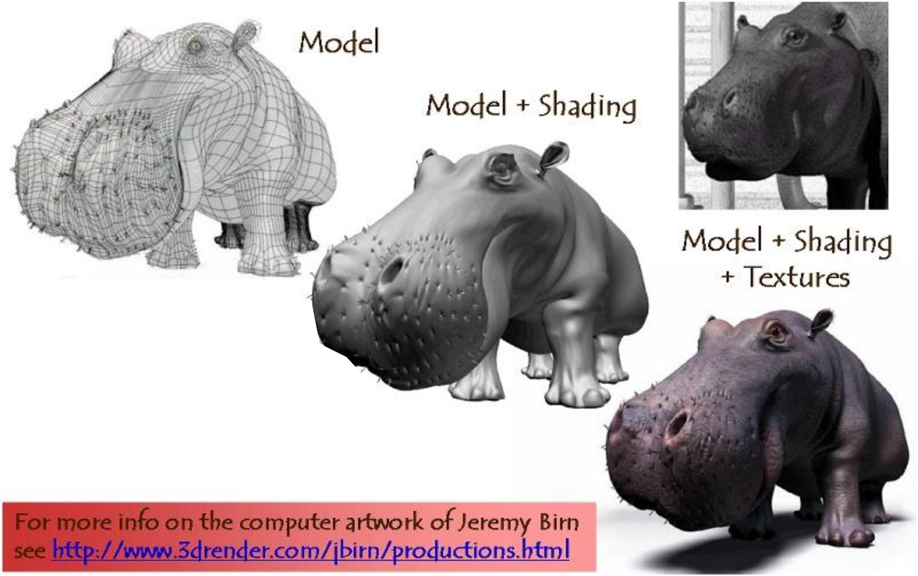




- Procedural : shader
  - write little programs that compute color/info as a function of location



# Effect of Textures




Courtesy of Jeremy Birn.

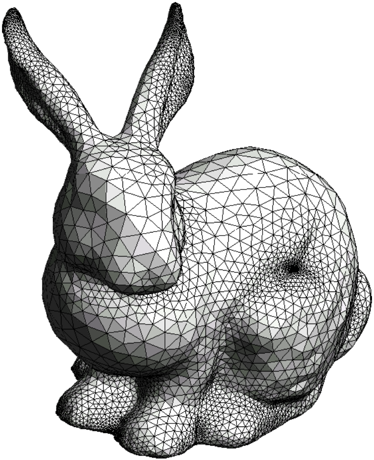
# Texture Mapping

Image of a cartoon of a man applying wall paper has been removed due to copyright restrictions.

# Texture Mapping

3D model

Texture mapped model



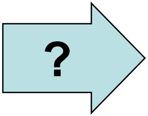







Image: Praun et al.

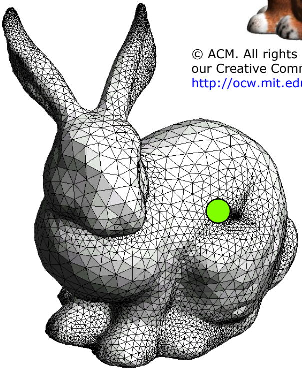
# Texture Mapping

Image: Praun et al.

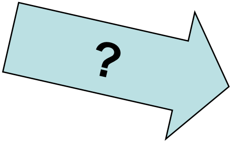
Texture mapped model



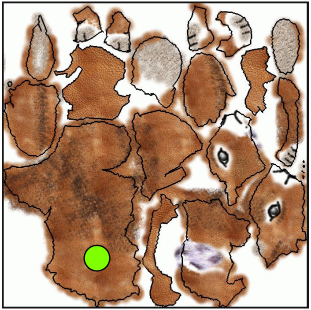
**We need a function that associates each surface point with a 2D coordinate in the texture map**








Texture map (2D image)



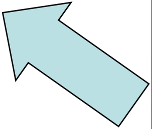
# Texture Mapping

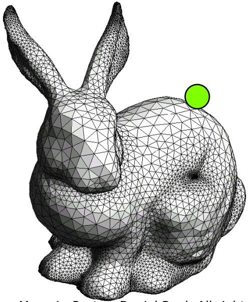
Image: Praun et al.

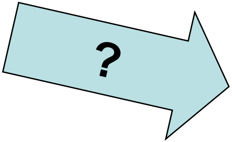
Texture mapped model



**For each point rendered, look up color in texture map**

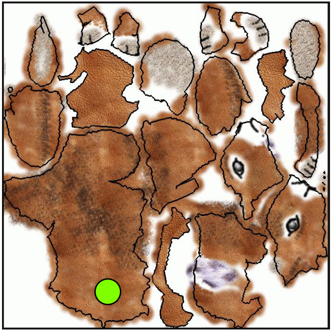






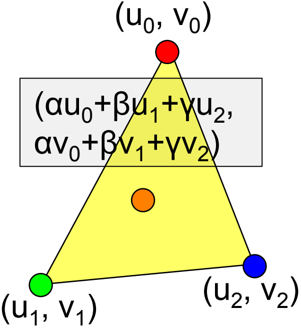


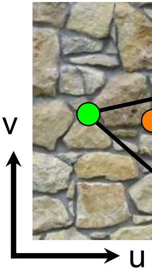

Texture map (2D image)



# UV Coordinates

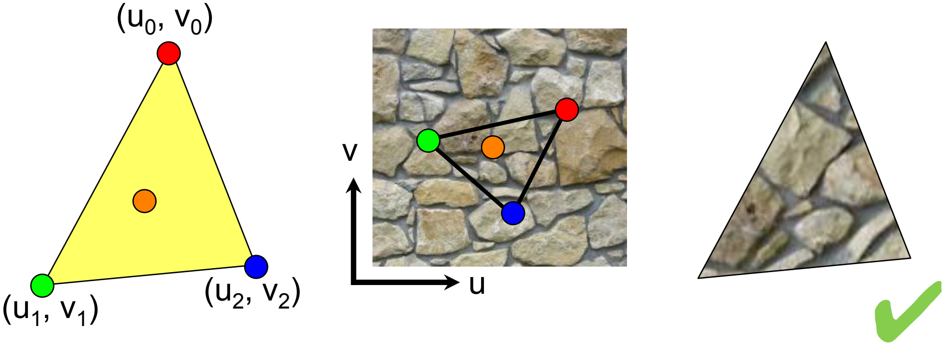
- Each vertex P stores 2D (u, v) “texture coordinates”
  - UVs determine the 2D location in the texture for the vertex
  - We will see how to specify them later
- Then we interpolate using barycentrics



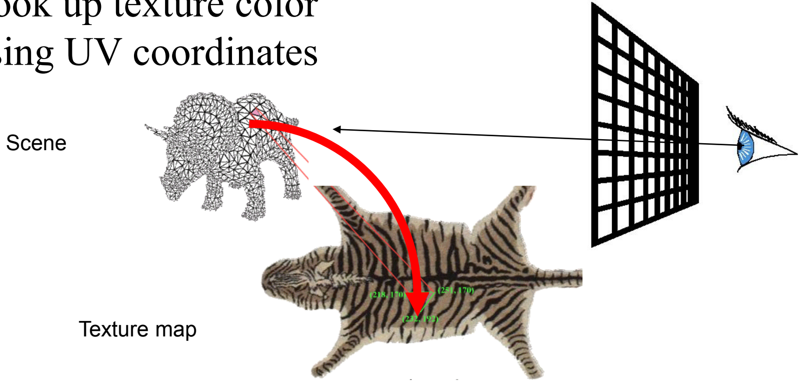
# UV Coordinates

- Each vertex P stores 2D (u, v) “texture coordinates”
  - UVs determine the 2D location in the texture for the vertex
  - We will see how to specify them later
- Then we interpolate using barycentrics



# Pseudocode – Ray Casting

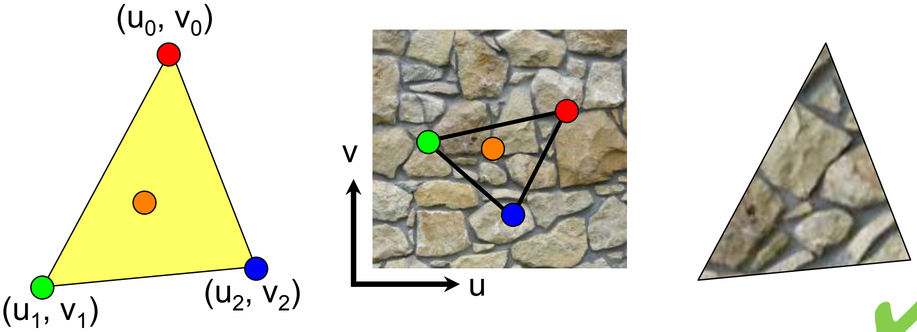
- Ray cast pixel (x, y), get visible point and α, β, γ
- Get texture coordinates (u, v) at that point
  - Interpolate from vertices using barycentrics
- Look up texture color using UV coordinates




Leonard McMillan, Computer Science at the University of North Carolina in Chapel Hill.

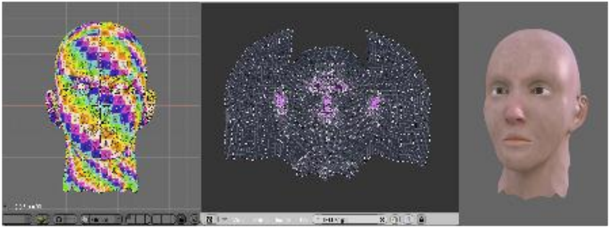
# UV Coordinates?

- Per-vertex (u, v) “texture coordinates” are specified:
  - Manually, provided by user (tedious!)
  - Automatically using parameterization optimization
  - Mathematical mapping (independent of vertices)



# Texture UV Optimization

- Goal : "flatten" 3D object onto 2D UV coordinates
- For each vertex, find coordinates U,V such that distortion is minimized
  - distances in UV correspond to distances on mesh
  - angle of 3D triangle same as angle of triangle in UV plane
- Cuts are usually required (discontinuities)

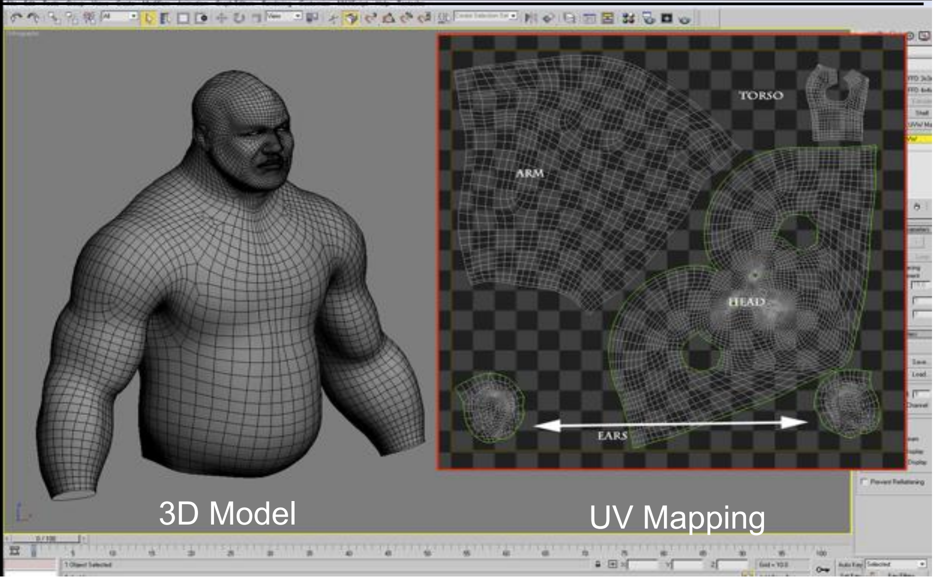


# To Learn More

- For this course, assume UV given per vertex
- Mesh Parameterization: Theory and Practice”
  - Kai Hormann, Bruno Lévy and Alla Sheffer *ACM SIGGRAPH Course Notes, 2007*
- http://alice.loria.fr/index.php/publications.html?redirect=0&Paper=SigCourseParam@2007&Author=Levy

Slide from Epic Games

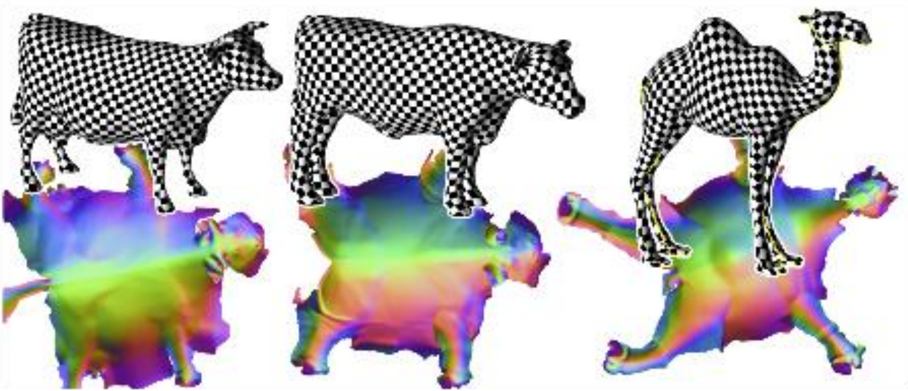
# Creating Torso Portion in Max



# 3D Model

- Information we need:
- Per vertex
  - 3D coordinates
  - Normal
  - 2D UV coordinates
- Other information
  - BRDF (often same for the whole object, but could vary)
  - 2D Image for the texture map

# Questions?



Some results computed by stretch $L_2$ minimization (parameterized models courtesy of Pedro Sander and Alla Sheffer).

# Mathematical Mapping

- What of non-triangular geometry?
  - Spheres, etc.

- No vertices, cannot specify UVs that way!

- Solution: Parametric Texturing
  - Deduce (u, v) from (x, y, z)
  - Various mappings are possible....

# Common Texture Coordinate Mappings

- Planar
  - Vertex UVs and linear interpolation is a special case!
- Cylindrical
- Spherical
- Perspective Projection

Images removed due to copyright restrictions.

# Projective Texture Example

- Modeling from photographs
- Using input photos as textures

Original photograph with marked edges | Recovered model | Model edges projected onto photograph | Synthetic rendering

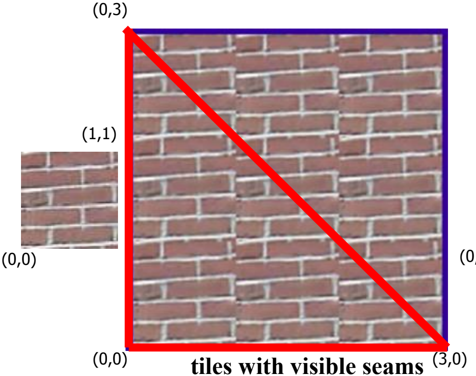


Figure from Debevec, Taylor & Malik
http://www.debevec.org/Research

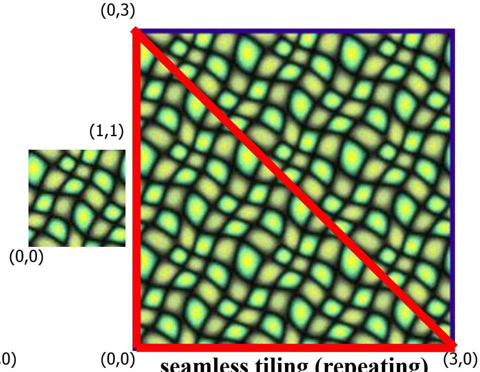
# Texture Tiling

**Note the range (0,1) unlike normalized screen coordinates!**

- Specify texture coordinates (u,v) at each vertex
- Canonical texture coordinates (0,0) → (1,1)
  - Wrap around when coordinates are outside (0, 1)




**tiles with visible seams**



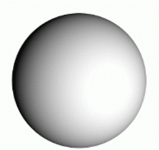

**seamless tiling (repeating)**

# Texture Mapping & Illumination

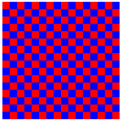
- Texture mapping can be used to alter some or all of the constants in the illumination equation
  - Diffuse color $k_d$, specular exponent $q$, specular color $k_s$...
  - Any parameter in any BRDF model!

$$L_o = \left[ k_a + k_d \left( \boldsymbol{n} \cdot \boldsymbol{l} \right) + k_s \left( \boldsymbol{v} \cdot \boldsymbol{r} \right)^q \right] \frac{L_i}{r^2}$$

  - $k_d$ in particular is often read from a texture map

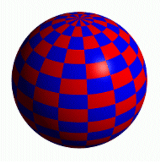


Constant Diffuse Color



Diffuse Texture Color

Texture used as Label



Texture used as Diffuse Color

# Gloss Mapping Example

Ron Frazier



Spatially varying $k_d$ and $k_s$

# Questions?

# We Can Go Even Further...

- The normal vector is really important in conveying the small-scale surface detail
  - Remember cosine dependence
  - The human eye is really good at picking up shape cues from lighting!

- We can exploit this and look up also the normal vector from a texture map
  - This is called “normal mapping” or “bump mapping”
  - A coarse mesh combined with detailed normal maps can convey the shape very well!

# Normal Mapping

- For each shaded point, normal is given by a 2D image normalMap that stores the 3D normal

For a visible point

interpolate UV using barycentric

// same as texture mapping

Normal = normalMap[U,V]

compute shading (BRDF) using this normal

$$L_o = \left[k_a + k_d\,(\boldsymbol{n} \cdot \boldsymbol{l}) + k_s\,(\boldsymbol{v} \cdot \boldsymbol{r})^q\right] \frac{L_i}{r^2}$$

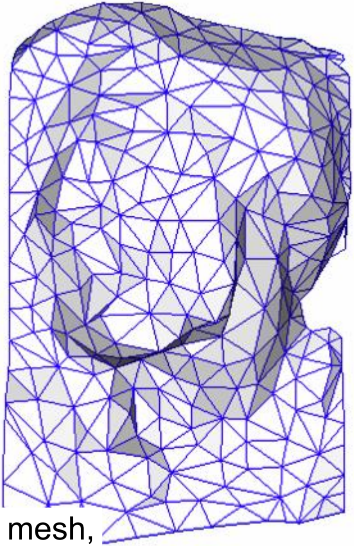
# Normal Map Example

Paolo Cignoni

Original Mesh
4M triangles

# Normal Map Example

Paolo Cignoni

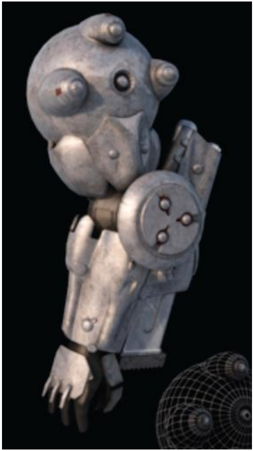


Simplified mesh,
500 triangles

Simplified mesh +
normal mapping

# Normal Map Example

Models and images: Trevor Taylor



Final render

Diffuse texture $k_d$

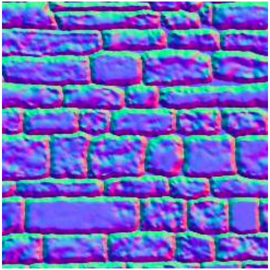
Normal Map

# Generating Normal Maps

- Model a detailed mesh
- Generate a UV parameterization for the mesh
  - A UV mapping such that each 3D point has **unique** image coordinates in the 2D texture map
  - This is a difficult problem, but tools are available
    - E.g., the DirectX SDK has functionality to do this
- Simplify the mesh (again, see DirectX SDK)
- Overlay simplified and original model
- For each point **P** on the simplified mesh, find closest point **P'** on original model (ray casting)
- Store the normal at **P'** in the normal map. **Done!**

# Normal Map Details

- You can store an object-space normal
  - Convenient if you have a unique parameterization
- ....but if you want to use a tiling normal map, this will not work
  - Must account for the curvature of the object!
  - Think of mapping this diffuse+normal map combination on a cylindrical tower
- Solution: Tangent space normal map
  - Encode a “difference” from the geometric normal in a local coord. system



# Questions?

Image from Epic Games has been removed due to copyright restrictions.

# Shaders (`Material` class)

- Functions executed when light interacts with a surface
- Constructor:
  - set shader parameters
- Inputs:
  - Incident radiance
  - Incident and reflected light directions
  - Surface tangent basis (anisotropic shaders only)
- Output:
  - Reflected radiance

# Shader

- Initially for production (slow) rendering
  - Renderman in particular
- Now used for real-time (Games)
  - Evaluated by graphics hardware
  - More later in the course
- Often makes heavy use of texture mapping

# Questions?

# Procedural Textures

- Alternative to texture mapping
- Little program that computes color as a function of *x,y,z*:

$f(x,y,z) \rightarrow color$

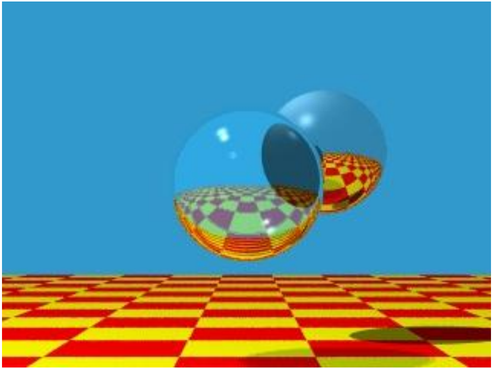
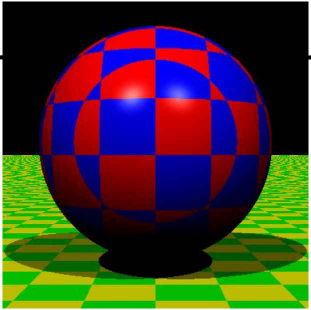


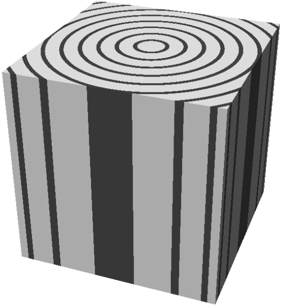


Image by Turner Whitted

# Procedural Textures



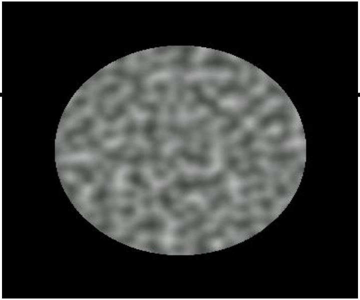
- Advantages:
  - easy to implement in ray tracer
  - more compact than texture maps (especially for solid textures)
  - infinite resolution



- Disadvantages
  - non-intuitive
  - difficult to match existing texture

# Questions?

# Perlin Noise





- Critical component of procedural textures
- Pseudo-random function
  - But continuous
  - band pass (single scale)
- Useful to add lots of visual detail

  http://www.noisemachine.com/talk1/index.html

  http://mrl.nyu.edu/~perlin/doc/oscar.html

  http://mrl.nyu.edu/~perlin/noise/

  http://en.wikipedia.org/wiki/Perlin_noise

  http://freespace.virgin.net/hugo.elias/models/m_perlin.htm

  (not really Perlin noise but very good)

  http://portal.acm.org/citation.cfm?id=325247

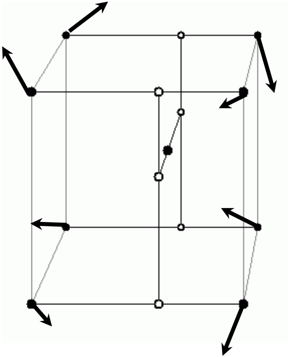
# Requirements

- Pseudo random
- For arbitrary dimension
  - 4D is common for animation
- Smooth
- Band pass (single scale)
- Little memory usage

- How would you do it?

# Perlin Noise

- Cubic lattice
- Zero at vertices
  - To avoid low frequencies
- Pseudo-random gradient at vertices
  - define local linear functions
- Splines to interpolate the values to arbitrary 3D points

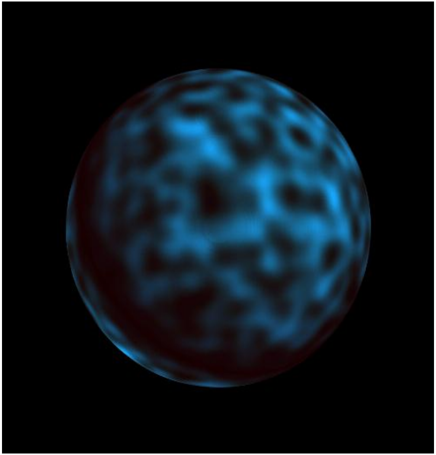


# TIEA311 - Today in Jyväskylä

- Basic idea of Perlin noise is nicely introduced on “Lecture 16” of the original course material.
- We skip it here. I hope the follow-up course starting next week has time for this, among many other wonderful things.
- Pseudo-random noise is very easy to incorporate in real-time graphics shaders. If you want, you can just “copy-paste” code that you trust (and that has a license that allows inclusion in your current work!)
- Next, we proceed directly to applications of Perlin noise.

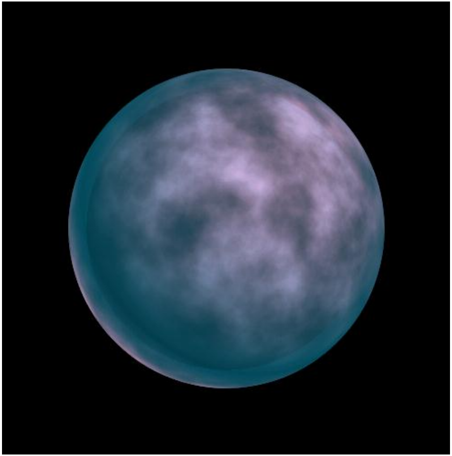
# Noise At One Scale

- A scale is also called an octave in noise parlance



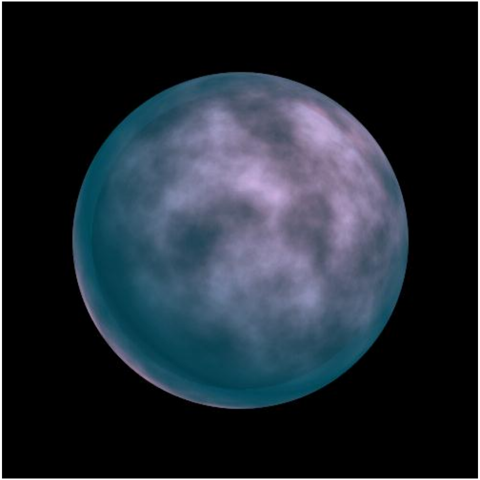
# Noise At Multiple Scales

- A scale is also called an octave in noise parlance
- But multiple octaves are usually used, where the scale between two octaves is multiplied by 2
  - hence the name octave



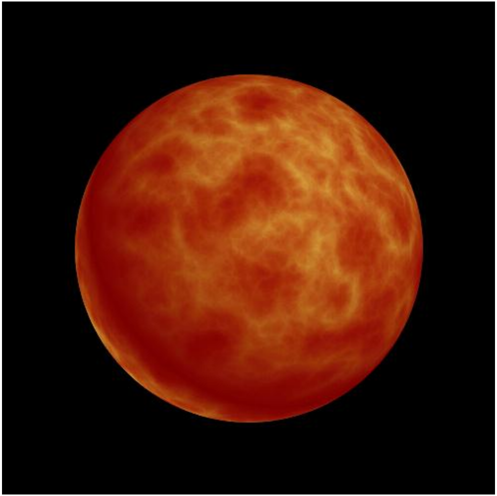
# Sum $1/f$ noise

- That is, each octave $f$ has weight $1/f$



# sum 1/*f* |*noise*|

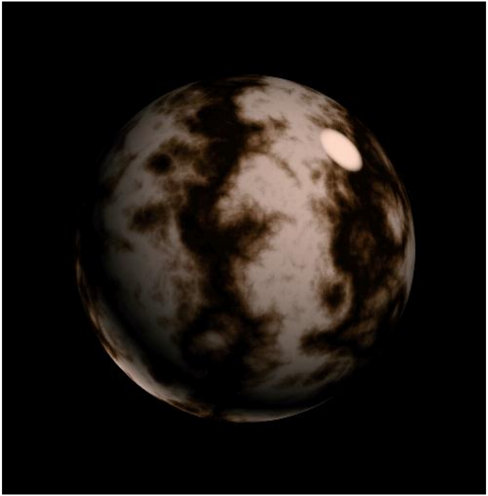
- Absolute value introduces *C1* discontinuities



- a.k.a. turbulence

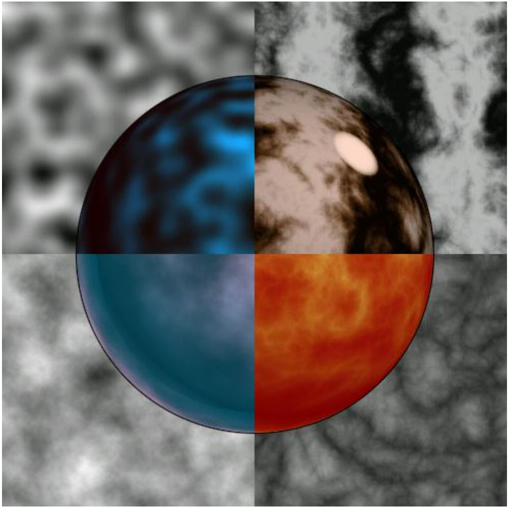
# *sin* (*x* + sum 1/*f* |*noise*|)

- Looks like marble!



# Comparison

•*noise* $\quad$ *sin*(*x* + sum 1/*f*( |*noise*| ))

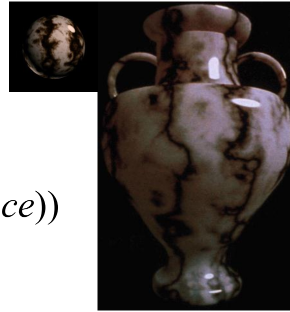


sum 1/*f*(*noise*) $\quad$ sum 1/*f*( |*noise*| )

# Questions?

# Noise For Solid Textures

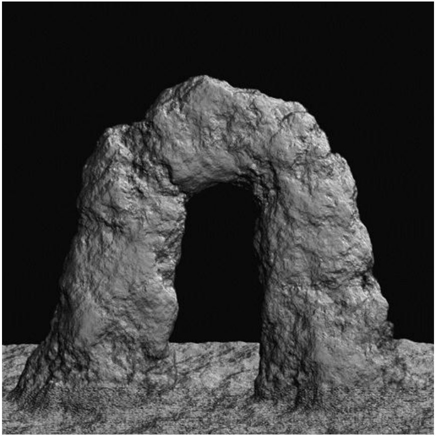
- Marble
  - recall *sin* (*x*[0] + sum 1/*f* |*noise*|)
  - *BoringMarble* = *colormap* (*sin*(*x*[0]))
  - *Marble* = *colormap* (*sin*(*x*[0]+*turbulence*))
  - http://legakis.net/justin/MarbleApplet/

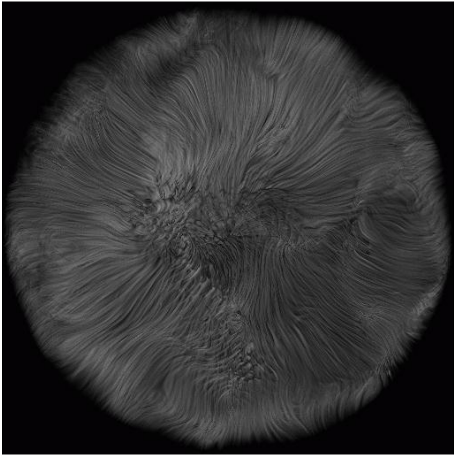




- Wood
  - replace *x* (or parallel plane) by radius
  - *Wood* = *colormap* (*sin*(*r*+*turbulence*))
  - http://www.connectedpixel.com/blog/texture/wood

# Other Cool Usage: Displacement, Fur





# Questions?

Image removed due to copyright restrictions. Please the image of “blueglass.gif” from
http://mrl.nyu.edu/~perlin/imgs/imgs.html.

# Shaders

- Noise: one ingredient of shaders
- Can also use textures
- Shaders control diffuse color, but also specular components, maybe even roughness (exponent), transparency, etc.
- Shaders can be layered (e.g. a layer of dust, peeling paint, mortar between bricks).
- Notion of shade tree
  - Pretty much algebraic tree
- Assignment 5:
  checkerboard shader based on two shaders

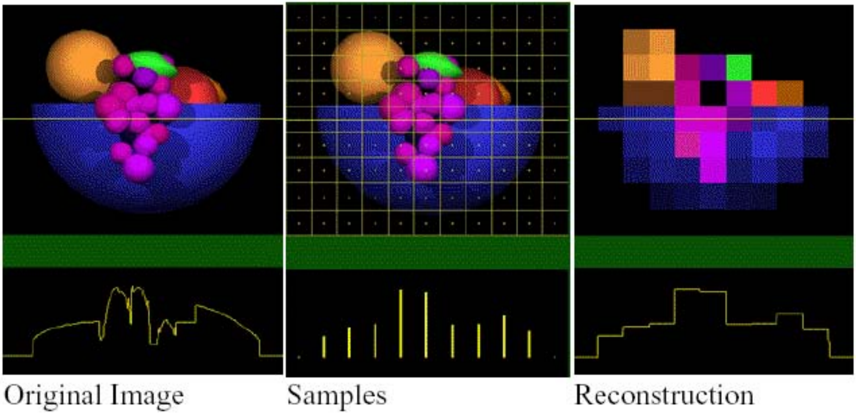
# Bottom Line

- Programmable shader provide great flexibility
- Shaders can be extremely complex
  - 10,000 lines of code!
- Writing shaders is a black art

# Sampling, Aliasing, & Mipmaps

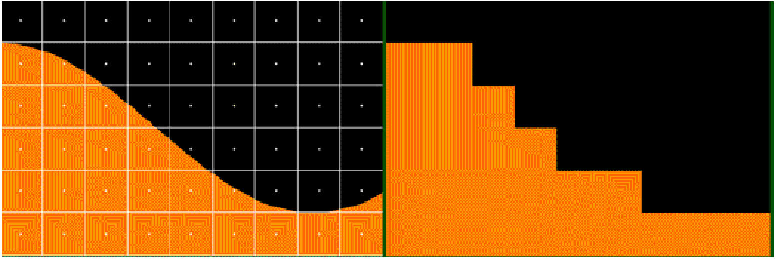
**MIT EECS 6.837 Computer Graphics**
Wojciech Matusik, MIT EECS

# Examples of Aliasing



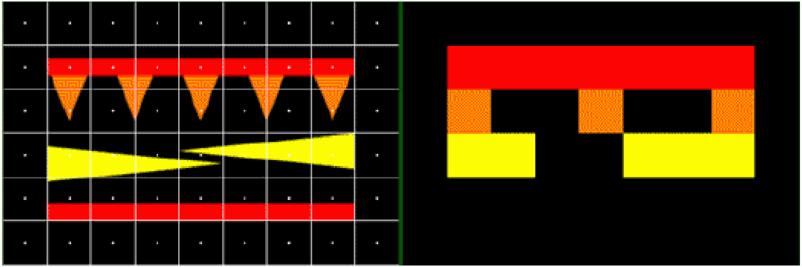
# Examples of Aliasing

## Jagged boundaries



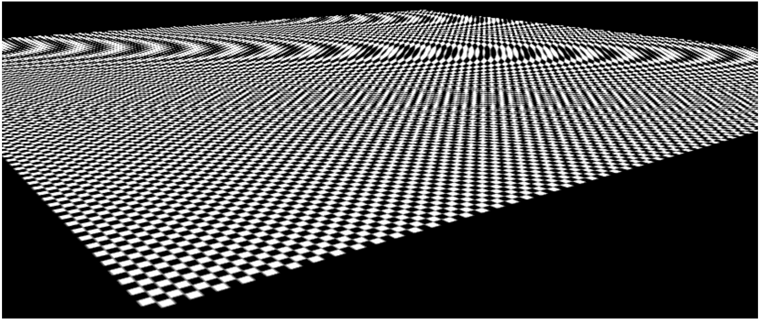
# Examples of Aliasing

Improperly rendered detail

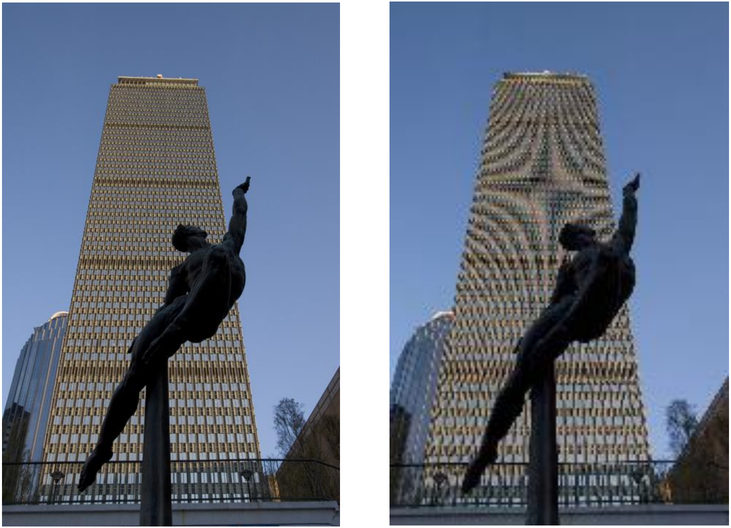


# Examples of Aliasing

Texture Errors



# In photos too





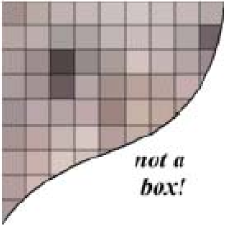
See also http://vimeo.com/26299355

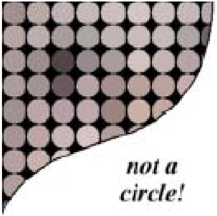
# Philosophical perspective

- The physical world is continuous, inside the computer things need to be discrete
- Lots of computer graphics is about translating continuous problems into discrete solutions
  - e.g. ODEs for physically-based animation, global illumination, meshes to represent smooth surfaces, rasterization, antialiasing
- Careful mathematical understanding helps do the right thing

# What is a Pixel?

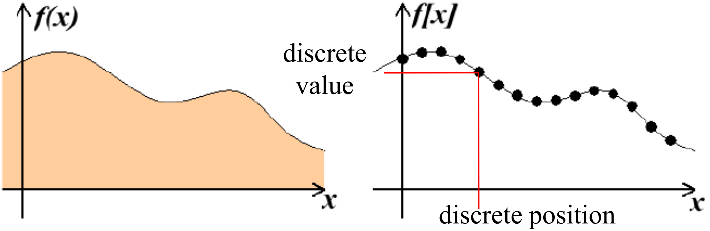
- A pixel is not:
  - a box
  - a disk
  - a teeny tiny little light
- A pixel "looks different" on different display devices
- A pixel is a sample
  - it has no dimension
  - it occupies no area
  - it cannot be seen
  - it has a coordinate
  - it has a value



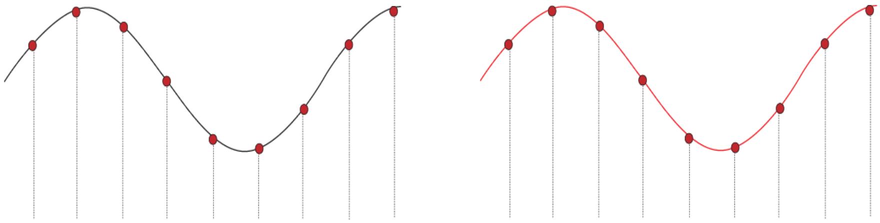
# More on Samples

- In signal processing, the process of mapping a continuous function to a discrete one is called *sampling*
- The process of mapping a continuous variable to a discrete one is called *quantization*
  - Gamma helps quantization
- To represent or render an image using a computer, we must both sample and quantize
  - Today we focus on the effects of sampling and how to fight them



# Sampling Density

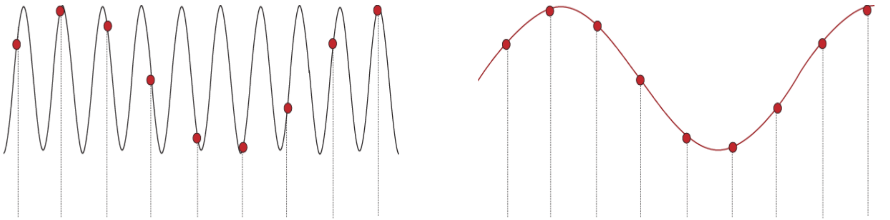
- If we're lucky, sampling density is enough



Input Reconstructed

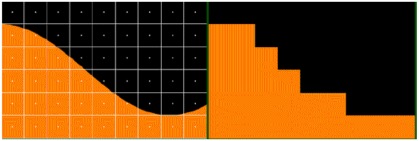
# Sampling Density

- If we insufficiently sample the signal, it may be mistaken for something simpler during reconstruction (that's aliasing!)
- This is why it’s called aliasing: the new low-frequency sine wave is an alias/ghost of the high-frequency one

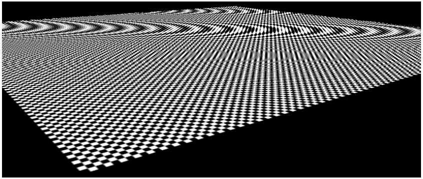


# Discussion

- Types of aliasing
  - Edges
    - mostly directional aliasing (vertical and horizontal edges rather than actual slope)
  - Repetitive textures
    - Paradigm of aliasing
    - Harder to solve right
    - Motivates fun mathematics







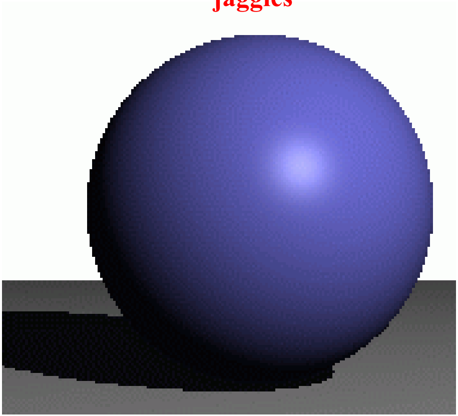
# Solution?

- How do we avoid that high-frequency patterns mess up our image?
- We blur!
  - In the case of audio, people first include an analog low-pass filter before sampling
  - For ray tracing/rasterization: compute at higher resolution, blur, resample at lower resolution
  - For textures, we can also blur the texture image before doing the lookup
- To understand what really happens, we need serious math

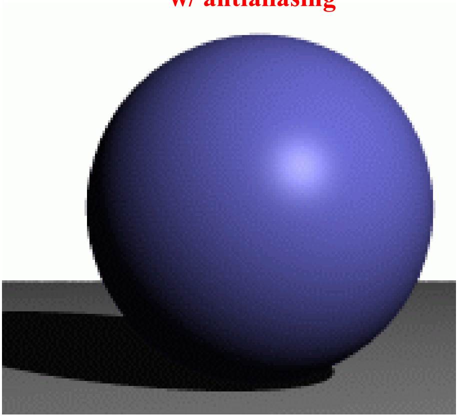
# In practice: Supersampling

- Your intuitive solution is to compute multiple color values per pixel and average them

**jaggies**

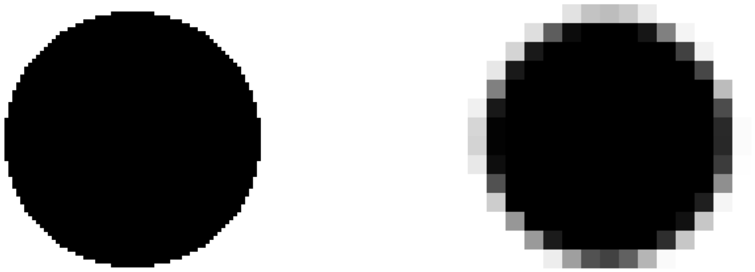


**w/ antialiasing**



# Uniform supersampling

- Compute image at resolution k*width, k*height
- Downsample using low-pass filter
  (e.g. Gaussian, sinc, bicubic)

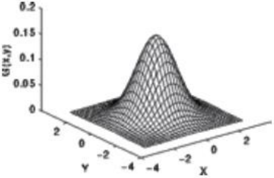


# Low pass / convolution

- Each output (low-res) pixel is a weighted average of input subsamples
- Weight depends on relative spatial position
- For example:
  - Gaussian as a function of distance
  - 1 inside a square, zero outside (box)

$\frac{1}{273}$

| | | | | |
|---|---|---|---|---|
| 1 | 4 | 7 | 4 | 1 |
| 4 | 16 | 26 | 16 | 4 |
| 7 | 26 | 41 | 26 | 7 |
| 4 | 16 | 26 | 16 | 4 |
| 1 | 4 | 7 | 4 | 1 |



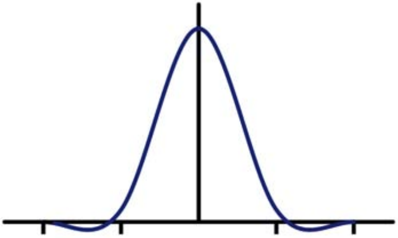

Gaussian



http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

# Recommended filter

- Bicubic
  - http://www.mentallandscape.com/Papers_siggraph88.pdf
- Good tradeoff between sharpness and aliasing



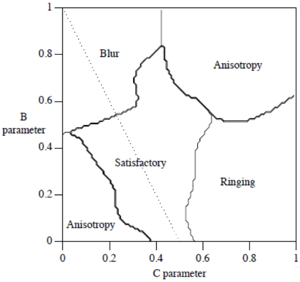
http://de.wikipedia.org/wiki/Datei:Mitchell_Filter.svg

# Choosing the parameters

- Empirical tests determined usable parameters
  - Mitchell, Don and Arun Netravali, "Reconstruction Filters in Computer Graphics", SIGGRAPH 88.

    http://www.mentallandscape.com/Papers_siggraph88.pdf

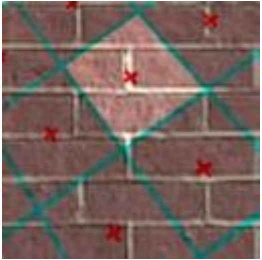
    http://dl.acm.org/citation.cfm?id=378514



# Spatial Filtering

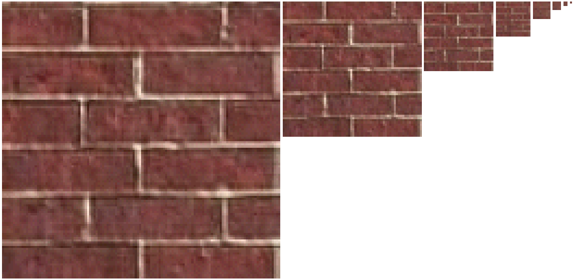
- Remove the high frequencies which cause artifacts in texture minification.
- Compute a spatial integration over the extent of the pixel
- This is equivalent to convolving the texture with a filter kernel centered at the sample (i.e., pixel center)!
- Expensive to do during rasterization, but an approximation it can be precomputed

projected texture in image plane



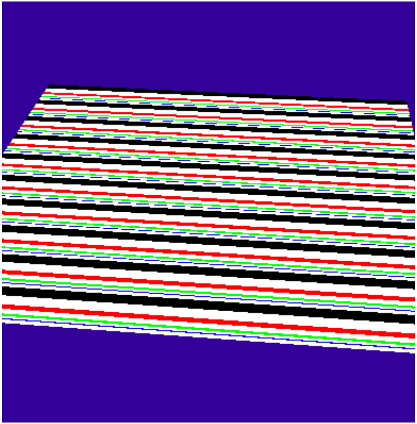
pixels projected in texture plane

# MIP Mapping

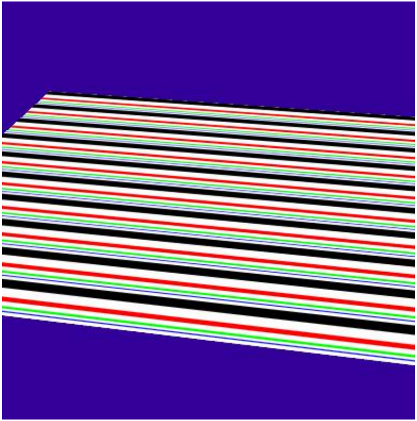


- Construct a pyramid of images that are pre-filtered and re-sampled at 1/2, 1/4, 1/8, etc., of the original image's sampling
- During rasterization we compute the index of the decimated image that is sampled at a rate closest to the density of our desired sampling rate
- MIP stands for *multum in parvo* which means *many in a small place*

# MIP Mapping Example



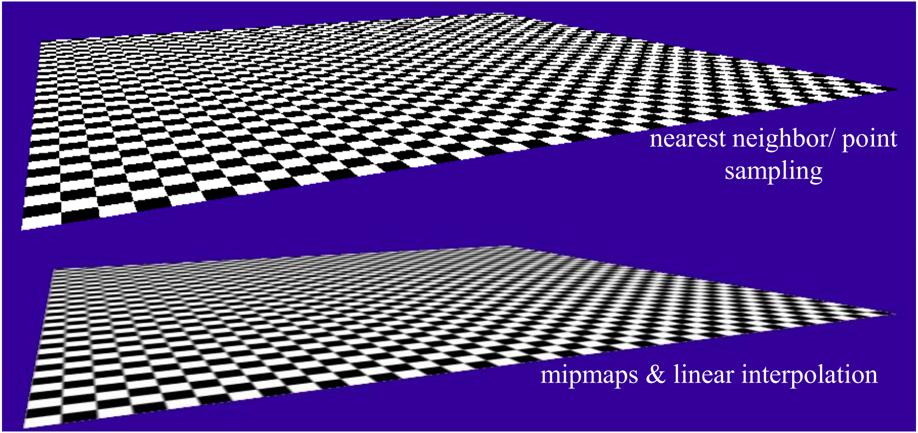
Nearest Neighbor



MIP Mapped (Bi-Linear)

# Examples of Aliasing

## Texture Errors



# TIEA311 - Today in Jyväskylä

- Much more about sampling issues and antialiasing on "Lecture 17" of the original course material.
- The previous few slides were just a low-resolution sample of the original slide set – (pun, intended, funny).
- As mentioned earlier, we gladly defer the theory to our local courses "TIES324 Signaalinkäsittely" and techniques to "TIES471 Reaaliaikainen renderöinti".

# TIEA311 - Today in Jyväskylä

Facing the fact that our original course material from MIT is a full-semester course whereas we only have one half, we need to cut stock a bit. On this lecture, we'll see "teasers" of what we skip, with ideas of where to fit similar material in our curriculum:

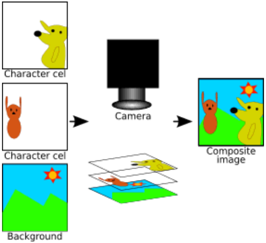
- ► While we cover animation from the original "Lecture 6", we skip **skinning**, and the skinning part of "Assignment 2".
- → This topic is covered in the follow-up course "Realtime Rendering" – skinning can be implemented in vertex shaders, which is also a topic of the follow-up course; benefits from quaternions, a piece of math suitable for the follow-up, too.
- ► We skip the original Lectures "7–9" about **physical models** and the practical "Assignment 3" that deals with those.
- → Maybe we could revive our own course about "physical models in computer animations" in the (near-ish?) future. . .

# Basics of Computer Animation
# Skinning/Enveloping

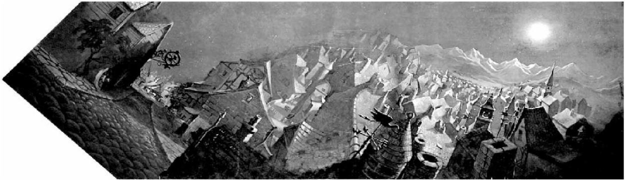
Many slides courtesy of Jovan Popovic, Ronen Barzel, and Jaakko Lehtinen

# Traditional Animation

- Draw each frame by hand
  - great control, but tedious
- Reduce burden with cel animation
  - Layer, keyframe, inbetween, …
  - Example: Cel panoramas (Disney's Pinocchio)







From ACM © 1997 "Multiperspective panoramas for cel animation."

# Traditional Animation Principles

- The in-betweening, was once a job for apprentice animators. Splines accomplish these tasks automatically. However, the animator still has to draw the keyframes. This is an art form and precisely why the experienced animators were spared the in-betweening work even before automatic techniques.
- The classical paper on animation by John Lasseter from Pixar surveys some the standard animation techniques:
- "*Principles of Traditional Animation Applied to 3D Computer Graphics,*" **SIGGRAPH'87**, pp. 35-44.
- See also The Illusion of Life: Disney Animation, by Frank Thomas and Ollie Johnston.

# Example: Squash and Stretch

- **Squash**: flatten an object or character by pressure or by its own power

- **Stretch**: used to increase the sense of speed and emphasize the squash by contrast

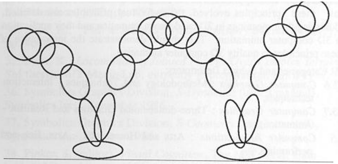


FIGURE 2. Squash & stretch in bouncing ball.

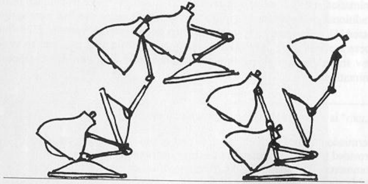


FIGURE 3. Squash & stretch in Luxo Jr.'s hop.



Image adapted from: Lasseter, John. "Principles of Traditional Animation applied to 3D Computer Animation." ACM SIGGRAPH Computer Graphics 21, no. 4 (July 1987): 35-44.

# Example: Timing

- Timing affects weight:
  - Light object move quickly
  - Heavier objects move slower

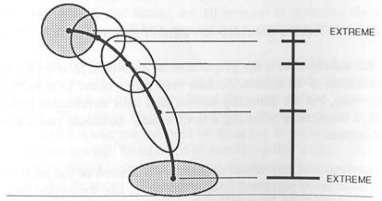


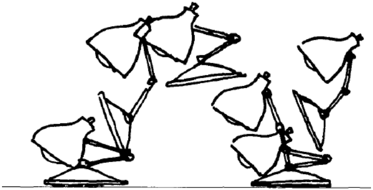

FIGURE 9. Timing chart for ball bounce.



- Timing completely changes the interpretation of the motion.

# Computer Animation

- How do we describe and generate motion of objects in the scene?

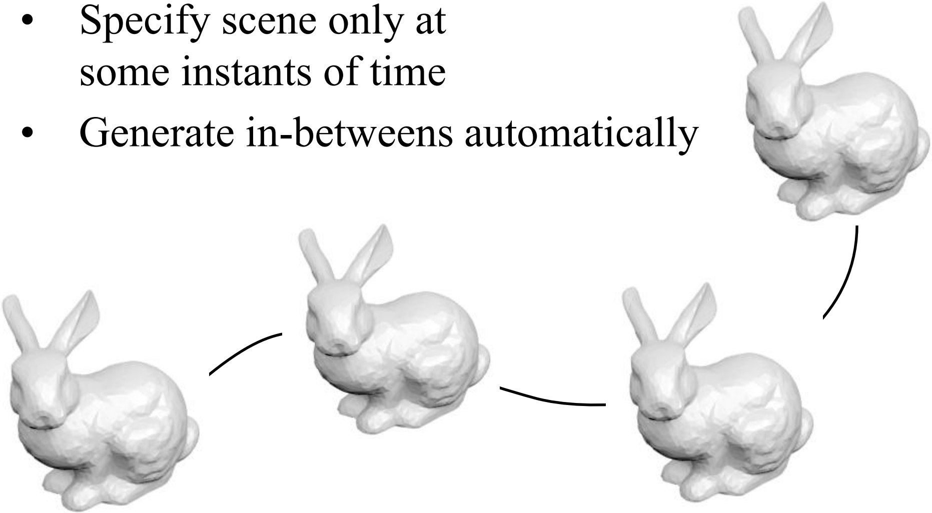




- Two very different contexts:
  - Production (offline)
    - Can be hardcoded, entire sequence know beforehand
  - Interactive (e.g. games, simulators)
    - Needs to react to user interaction, sequence not known

# Types of Animation: Keyframing

- Specify scene only at some instants of time
- Generate in-betweens automatically

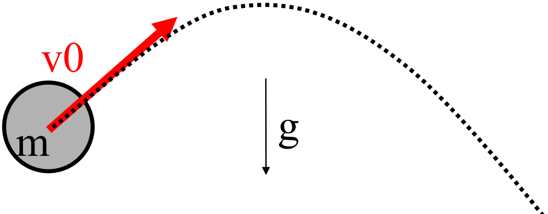


# Types of Animation: Procedural

- Describes the motion algorithmically
- Express animation as a function of small number of parameters
- Example
  - a clock/watch with second, minute and hour hands
  - express the clock motions in terms of a "seconds" variable
    - the clock is animated by changing this variable
- Another example: Grass in the wind, tree canopies, etc.

# Types of Animation: Physically-Based

- Assign physical properties to objects
  - Masses, forces, etc.
- Also procedural forces (like wind)
- Simulate physics by solving equations of motion
  - Rigid bodies, fluids, plastic deformation, etc.
- Realistic but difficult to control



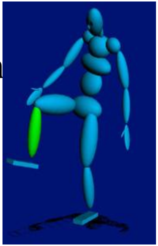
# Another Example

- Physically-Based Character Animation
  - Specify keyframes, solve for physically valid motion that interpolates them by “spacetime optimization”

- Anthony C. Fang and Nancy S. Pollard, 2003. Efficient Synthesis of Physically Valid Human Motion, ACM Transactions on Graphics 22(3) 417-426, Proc. SIGGRAPH 2003.http://graphics.cs.cmu.edu/nsp/projects/spacetime/spacetime.html

# Because we are Lazy...

- Animation is (usually) specified using some form of low-dimensional **controls** as opposed to remodeling the actual geometry for each frame.
  - Example: The joint angles (bone transformations) in a hierarchical character determine the pose
  - Example: A rigid motion is represented by changing the object-to-world transformation (rotation and translation).





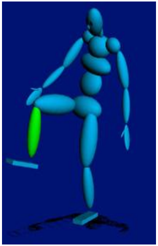
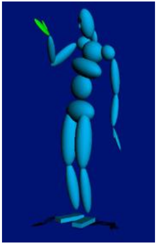
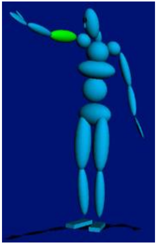
**"Blendshapes" are keyframes that are just snapshots of the entire geometry.**

Building 3D models and their animation controls is a major component of every animation pipeline.

Building the controls is called “rigging”.

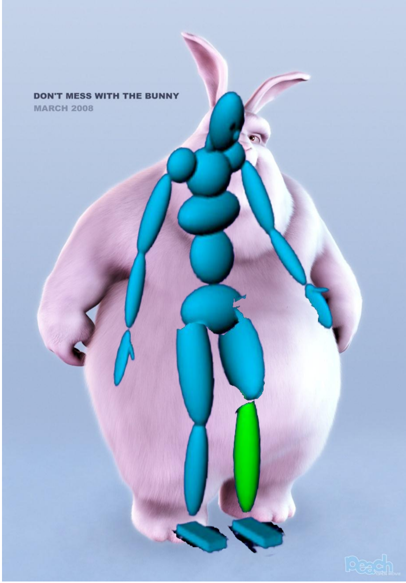
# Articulated Character Models

- Forward kinematics describes the positions of the body parts as a function of joint angles
  - Body parts are usually called "bones"
  - Angles are the low-dimensional control.
- Inverse kinematics specifies constraint locations for bones and solves for joint angles.

# Skinning Characters

- Embed a skeleton into a detailed character mesh
- Animate "bones"
  - Change the joint angles over time
  - Keyframing, procedural, etc.
- Bind skin vertices to bones
  - Animate skeleton, skin will move with it

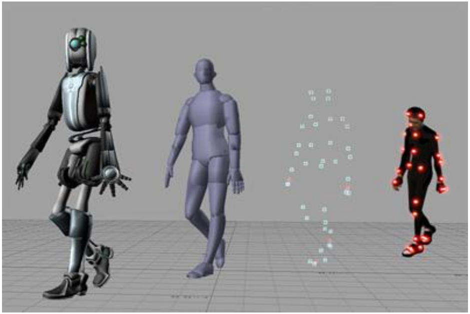


# Motion Capture

- Usually uses optical markers and multiple high-speed cameras
- Triangulate to get marker 3D position
  - (Again, structure from motion and projective geometry, i.e., homogeneous coordinates)
- Captures style, subtle nuances and realism
- But need ability to record someone

# Motion Capture

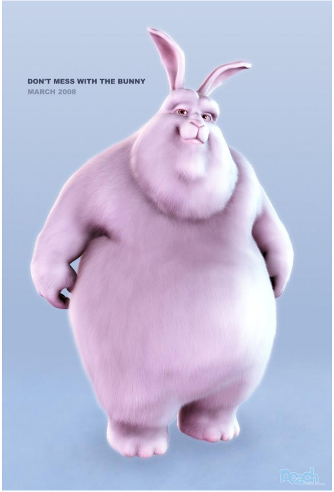
- Motion capture records 3D marker positions
  - But character is controlled using animation controls that affect bone transformations!
- Marker positions must be translated into character controls ("retargeting")



# Skinning/Enveloping

# Skinning

- We know how to animate a bone hierarchy
  - Change the joint angles, i.e., bone transformations, over time (keyframing)
- Embed a skeleton into a detailed character mesh
- Bind skin vertices to bones
  - Animate skeleton, skin will move with it
  - But how?



# Skinning/Enveloping

- Need to infer how skin deforms from bone transformations.
- Most popular technique: Skeletal Subspace Deformation (SSD), or simply Skinning
  - Other aliases
    - vertex blending
    - matrix palette skinning
    - linear blend skinning