

TIEA311

Tietokonegrafiikan perusteet

kevät 2019

(“Principles of Computer Graphics” – Spring 2019)

Copyright and Fair Use Notice:

The lecture videos of this course are made available for registered students only. Please, do not redistribute them for other purposes. Use of auxiliary copyrighted material (academic papers, industrial standards, web pages, videos, and other materials) as a part of this lecture is intended to happen under academic “fair use” to illustrate key points of the subject matter. The lecturer may be contacted for take-down requests or other copyright concerns (email: paavo.j.nieminen@jyu.fi).

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/>.

License: Creative Commons BY-NC-SA

Original license terms apply. Re-arrangement and new content copyright 2017-2019 by *Paavo Nieminen* and *Jarno Kansanaho*

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/>

TIEA311 - Today in Jyväskylä

Last lecture plan:

- ▶ Shading, texture mapping: Cover the principles up to Phong model and texture coordinates.
- ▶ Cherry-pick title slides from advanced stuff that we mostly defer to the follow-up course (starts next week) and/or future self-study.
- ▶ When **learning**, don't think about deadlines. **Think about learning!**
- ▶ My deadlines are flexible, since this course is about graphics and programming and **not about meeting time-to-market**. Leave that to project courses and traineeship periods!

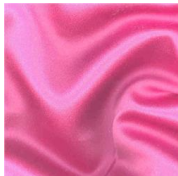
Shading & Material Appearance



© ACM. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

Lighting and Material Appearance

- Input for realistic rendering
 - Geometry, Lighting and Materials
- Material appearance
 - Intensity and shape of highlights
 - Glossiness
 - Color
 - Spatial variation, i.e., texture (next Tuesday)

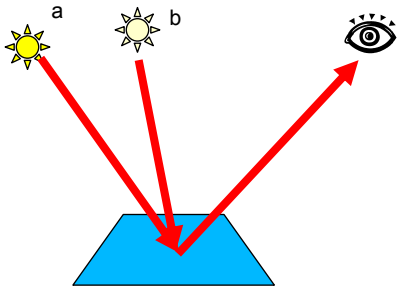


Unit Issues - Radiometry

- We will not be too formal in this class
- Issues we will not really care about
 - Directional quantities vs. integrated over all directions
 - Differential terms: per solid angle, per area
 - Power? Intensity? Flux?
- Color
 - All math here is for a single wavelength only; we will perform computations for R, G, B separately
 - Do not panic, that just means we will perform every operation three times, that is all

Light Sources

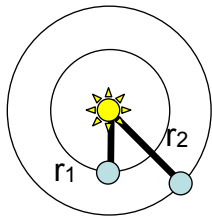
- Today, we only consider point light sources
 - Thus we do not need to care about solid angles
- For multiple light sources, use linearity
 - We can add the solutions for two light sources
 - $I(a+b) = I(a) + I(b)$
 - We simply multiply the solution when we scale the light intensity
 - $I(s a) = s I(a)$



Yet again, linearity
is our friend!

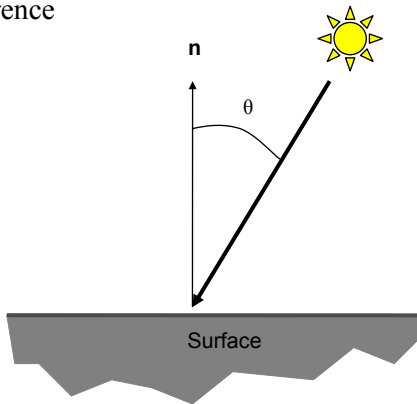
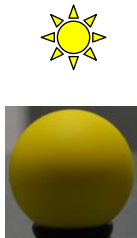
Intensity as Function of Distance

- $1/r^2$ fall-off for isotropic point lights
 - Why? An isotropic point light outputs constant power per solid angle (“into all directions”)
 - Must have same power in all concentric spheres
 - Sphere’s surface area grows with $r^2 \Rightarrow$ energy obeys $1/r^2$
- ... but in graphics we often cheat with or ignore this.
 - Why? Ideal point lights are kind of harsh
 - Intensity goes to infinity when you get close – not great!
 - In particular, $1/(ar^2+br+c)$ is popular



Incoming Irradiance

- The amount of light energy received by a surface depends on incoming angle
 - Bigger at normal incidence, even if distance is const.
 - Similar to winter/summer difference
- How exactly?
 - Cos θ law
 - Dot product with normal

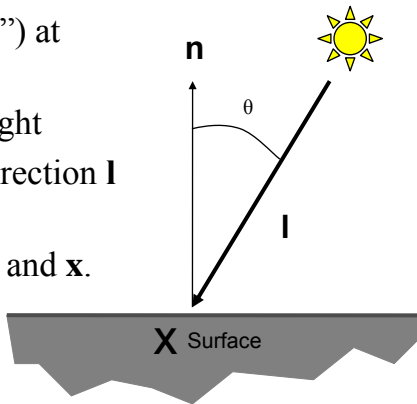


Incoming Irradiance for Pointlights

- Let's combine this with the $1/r^2$ fall-off:

$$I_{in} = I_{light} \cos \theta / r^2$$

- I_{in} is the irradiance (“intensity”) at surface point \mathbf{x}
- I_{light} is the “intensity” of the light
- θ is the angle between light direction \mathbf{l} and surface normal \mathbf{n}
- r is the distance between light and \mathbf{x} .

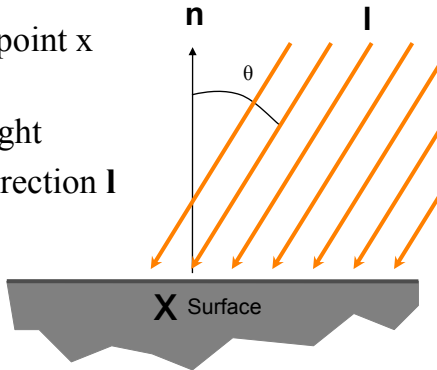


Directional Lights

- “Pointlights that are infinitely far”
 - No falloff, just one direction and one intensity

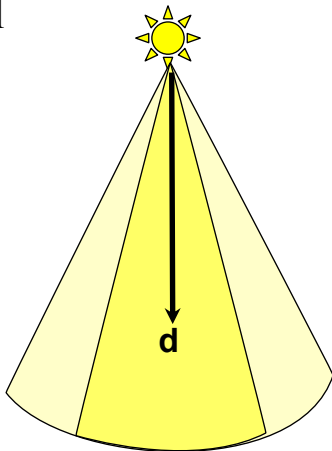
$$I_{in} = I_{light} \cos \theta$$

- I_{in} is the irradiance at surface point x from the directional light
- I_{light} is the “intensity” of the light
- θ is the angle between light direction \mathbf{l} and surface normal \mathbf{n}
 - Only depends on \mathbf{n} , not \mathbf{x} !

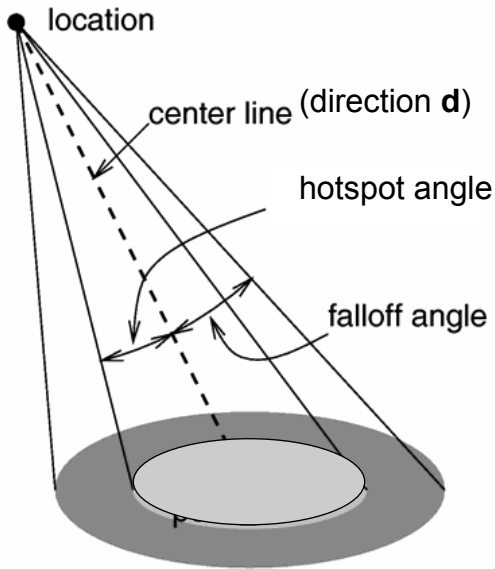


Spotlights

- Pointlights with non-uniform directional emission
- Usually symmetric about a central direction \mathbf{d} , with angular falloff
 - Often two angles
 - “Hotspot” angle:
No attenuation within the central cone
 - “Falloff” angle: Light attenuates from full intensity to zero intensity between the hotspot and falloff angles
- Plus your favorite distance falloff curve



Spotlight Geometry



Adapted from
POVRAY documentation

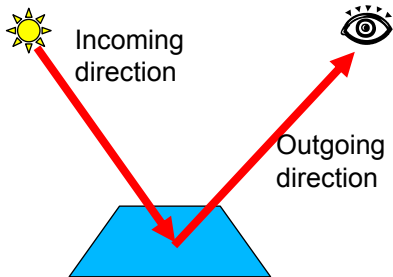
Questions?

Quantifying Reflection – BRDF

- Bidirectional Reflectance Distribution Function
- Ratio of light coming from one direction that gets reflected in another direction
 - Pure reflection, assumes no light scatters into the material
- Focuses on angular aspects, not spatial variation of the material
- **How many dimensions?**

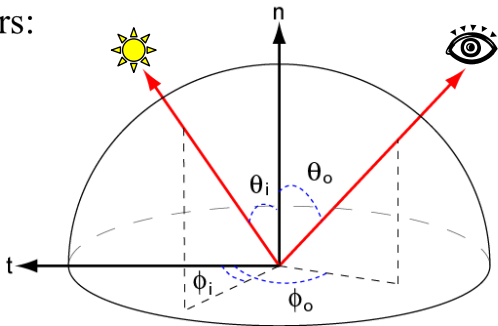


© ACM. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.



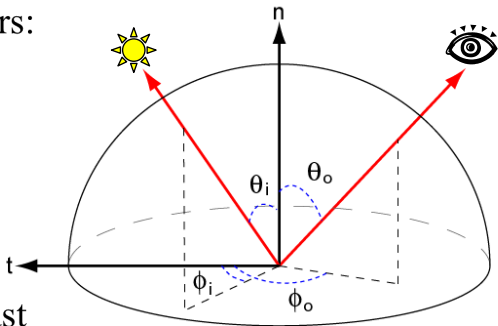
BRDF f_r

- Bidirectional Reflectance Distribution Function
 - 4D: 2 angles for each direction
 - $\text{BRDF} = f_r(\theta_i, \phi_i; \theta_o, \phi_o)$
 - Or just two unit vectors:
 $\text{BRDF} = f_r(\mathbf{l}, \mathbf{v})$
 - \mathbf{l} = light direction
 - \mathbf{v} = view direction



BRDF f_r

- Bidirectional Reflectance Distribution Function
 - 4D: 2 angles for each direction
 - $\text{BRDF} = f_r(\theta_i, \phi_i; \theta_o, \phi_o)$
 - Or just two unit vectors:
 $\text{BRDF} = f_r(\mathbf{l}, \mathbf{v})$
 - \mathbf{l} = light direction
 - \mathbf{v} = view direction
 - The BRDF is aligned with the surface; the vectors \mathbf{l} and \mathbf{v} must be in a local coordinate system

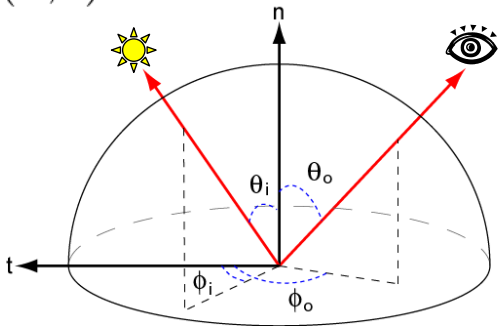


BRDF f_r

- Relates incident irradiance from every direction to outgoing light.
How?

$$I_{\text{out}}(\mathbf{v}) = I_{\text{in}}(\mathbf{l}) f_r(\mathbf{v}, \mathbf{l})$$

**\mathbf{l} = light direction
(incoming)**
 **\mathbf{v} = view direction
(outgoing)**



BRDF f_r

- Relates incident irradiance from every direction to outgoing light.
How?

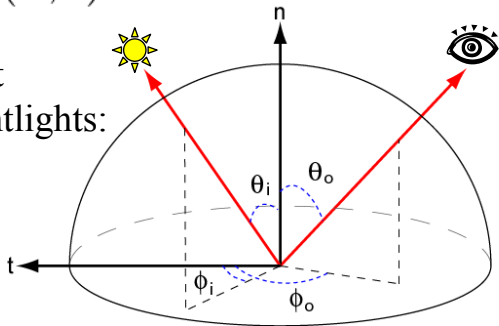
$$I_{\text{out}}(\mathbf{v}) = I_{\text{in}}(\mathbf{l}) f_r(\mathbf{v}, \mathbf{l})$$

- Let's combine with what we know already of pointlights:

$$I_{\text{out}}(\mathbf{v}) =$$

$$\frac{I_{\text{light}} \cos \theta_i}{r^2} f_r(\mathbf{v}, \mathbf{l})$$

\mathbf{l} = light direction (incoming)
 \mathbf{v} = view direction (outgoing)

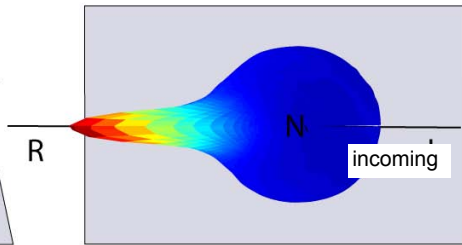
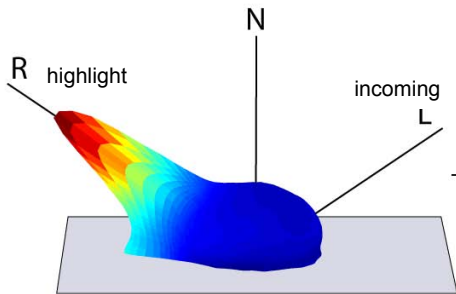


2D Slice at Constant Incidence

- For a fixed incoming direction, view dependence is a 2D spherical function
 - Here a moderate specular component



© ACM. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.



Courtesy of Mitsubishi Electric Research Laboratories, Inc. Used with permission.

Example: Plot of "PVC" BRDF at 55° incidence

Isotropic vs. Anisotropic

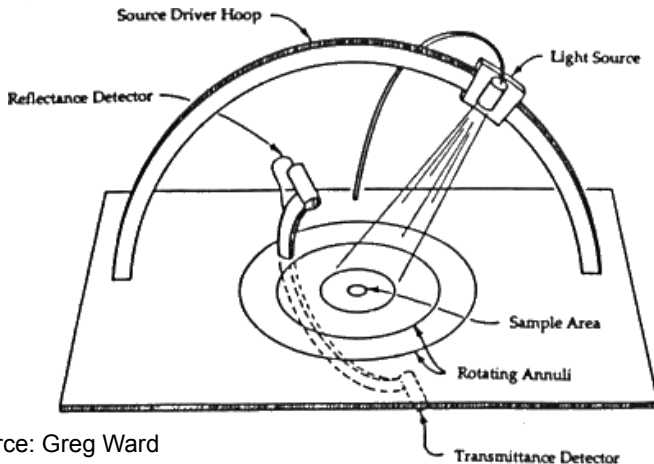
- When keeping \mathbf{l} and \mathbf{v} fixed, if rotation of surface around the normal does not change the reflection, the material is called isotropic
- Surfaces with strongly oriented microgeometry elements are anisotropic
- Examples:
 - brushed metals,
 - hair, fur, cloth, velvet



Westin et.al 92

How do we obtain BRDFs?

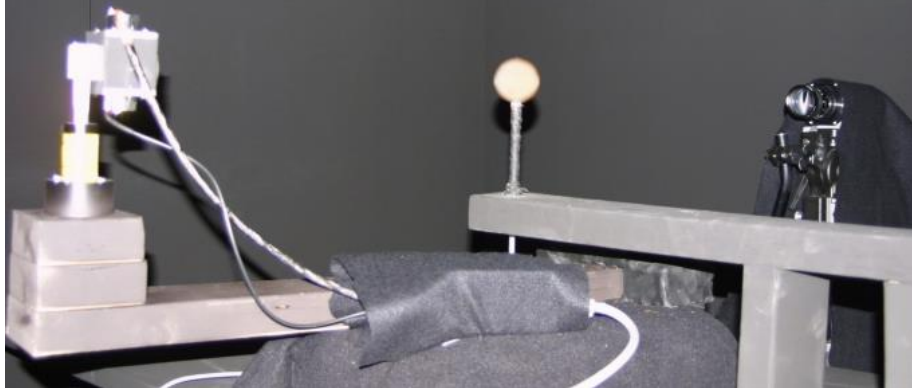
- One possibility: Gonioreflectometer
 - 4 degrees of freedom



Source: Greg Ward

How Do We Obtain BRDFs?

- Another possibility: Take pictures of spheres coated with material, rotate light around a 1D arc
 - This gives 3DOF => isotropic materials only



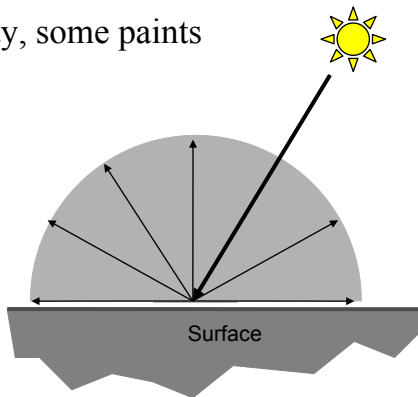
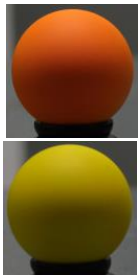
Parametric BRDFs

- BRDFs can be measured from real data
 - But tabulated 4D data is too cumbersome for most uses
- Therefore, parametric BRDF models represent the relationship between incident and outgoing light by some mathematical formula
 - The appearance can then be tuned by setting parameters
 - “Shininess”, “anisotropy”, etc.
 - Physically-based or Phenomenological
 - They can model with measured data (examples later)
- Popular models: Diffuse, Blinn-Phong, Cook-Torrance, Lafortune, Ward, Oren-Nayar, etc.

Questions?

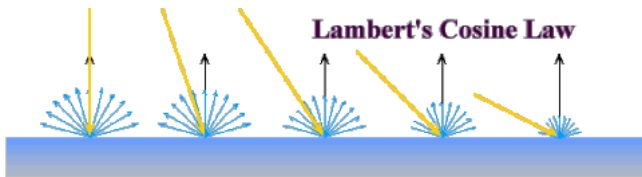
Ideal Diffuse Reflectance

- Assume surface reflects equally in all directions.
- An ideal diffuse surface is, at the microscopic level, a very rough surface.
 - Example: chalk, clay, some paints



Ideal Diffuse Reflectance

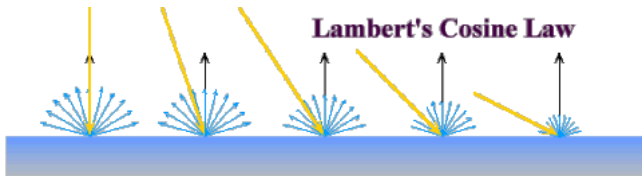
- Ideal diffuse reflectors reflect light according to Lambert's cosine law
 - The reflected light varies with cosine even if distance to light source is kept constant



Ideal Diffuse Reflectance

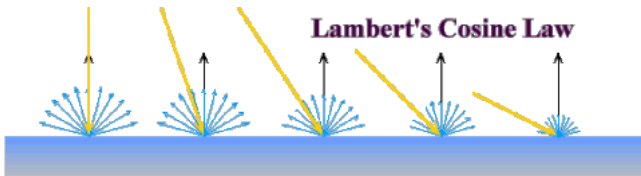
- Ideal diffuse reflectors reflect light according to Lambert's cosine law
 - The reflected light varies with cosine even if distance to light source is kept constant

Remembering that incident irradiance depends on cosine, what is the BRDF of an ideally diffuse surface?



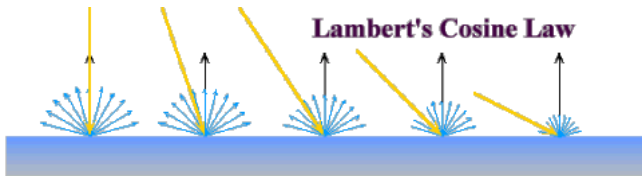
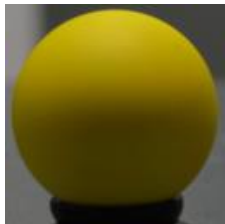
Ideal Diffuse Reflectance

- The ideal diffuse BRDF is a constant $f_r(\mathbf{l}, \mathbf{v}) = \text{const.}$
 - What constant ρ/π , where ρ is the *albedo*
 - Coefficient between 0 and 1 that says what fraction is reflected
 - Usually just called “diffuse color” k_d
 - You have already implemented this by taking dot products with the normal and multiplying by the “color”!



Ideal Diffuse Reflectance

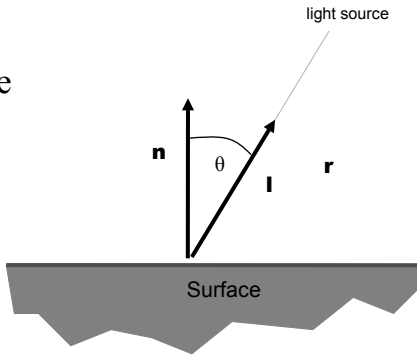
- This is the simplest possible parametric BRDF
 - One parameter: k_d
 - (One for each RGB channel)



Ideal Diffuse Reflectance Math

- Single Point Light Source
 - k_d : diffuse coefficient (color)
 - \mathbf{n} : Surface normal.
 - \mathbf{l} : Light direction.
 - L_i : Light intensity
 - r : Distance to source
 - L_o : Shaded color

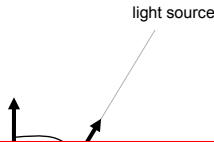
$$L_o = k_d \max(0, \mathbf{n} \cdot \mathbf{l}) \frac{L_i}{r^2}$$



Ideal Diffuse Reflectance Math

- Single Point Light Source
 - k_d : diffuse coefficient (color)
 - \mathbf{n} : Surface normal.
 - \mathbf{l} : Light direction.
 - L_i : Light intensity
 - r : Distance to source
 - L_o : Shaded color

$$L_o = k_d \max(0, \mathbf{n} \cdot \mathbf{l}) \frac{L_i}{r^2}$$

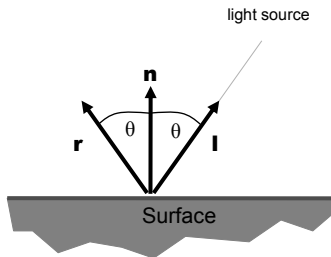


Do not forget
to normalize
your \mathbf{n} and \mathbf{l} !

We do not want light from below the surface! From now on we always assume (on this lecture) that **dot products are clamped to zero** and skip writing out the `max()`.

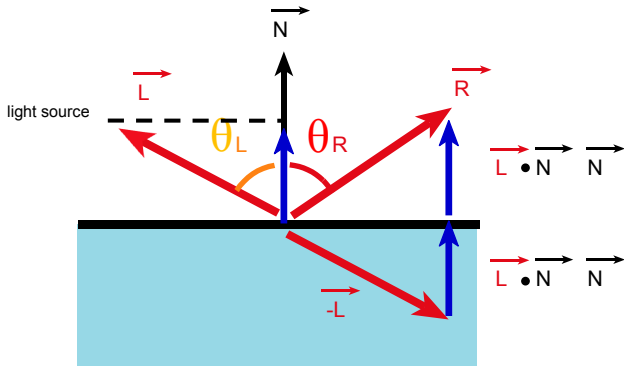
Ideal Specular Reflectance

- Reflection is only at mirror angle
- View dependent
 - Microscopic surface elements are usually oriented in the same direction as the surface itself.
 - Examples: mirrors, highly polished metals.



Recap: How to Get Mirror Direction

- Reflection angle = light angle
 - Both \mathbf{R} & \mathbf{L} have to lie on one plane
- $\mathbf{R} = -\mathbf{L} + 2(\mathbf{L} \cdot \mathbf{N})\mathbf{N}$



Ideal Specular BRDF

- Light **only** reflects to the mirror direction
- A Dirac delta multiplied by a specular coefficient k_s
- Not very useful for point lights, only for reflections of other surfaces
 - Why? You cannot really see a mirror reflection of an infinitely small light!

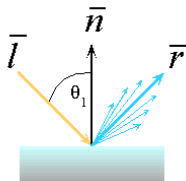
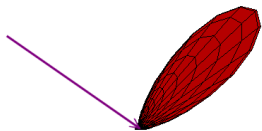
Non-ideal Reflectors

- Real glossy materials usually deviate significantly from ideal mirror reflectors
 - Highlight is blurry
- They are not ideal diffuse surfaces either ...



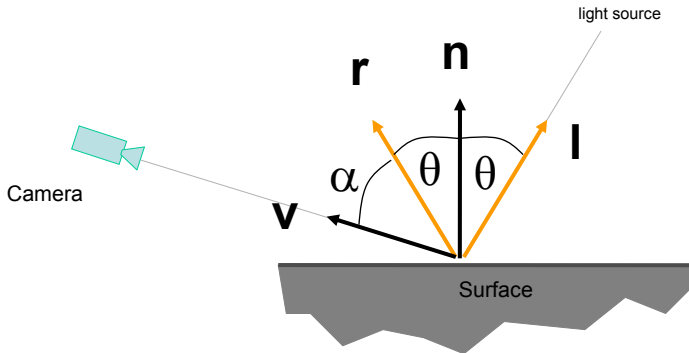
Non-ideal Reflectors

- Simple Empirical Reasoning for Glossy Materials
 - We expect most of the reflected light to travel in the direction of the ideal mirror ray.
 - However, because of microscopic surface variations we might expect some of the light to be reflected just slightly offset from the ideal reflected ray.
 - As we move farther and farther, in the angular sense, from the reflected ray, we expect to see less light reflected.



The Phong Specular Model

- How much light is reflected?
 - Depends on the angle α between the ideal reflection direction \mathbf{r} and the viewer direction \mathbf{v} .

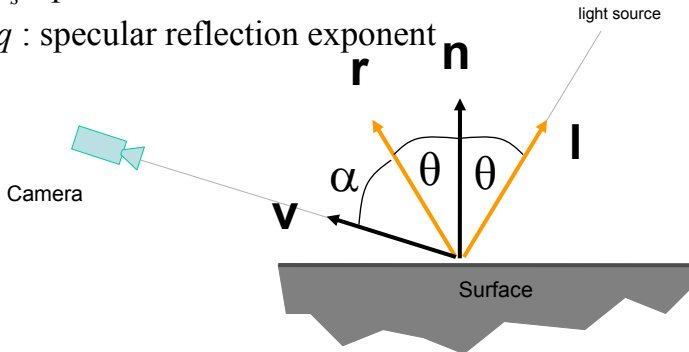


The Phong Specular Model

$$L_o = k_s (\cos \alpha)^q \frac{L_i}{r^2} = k_s (\mathbf{v} \cdot \mathbf{r})^q \frac{L_i}{r^2}$$

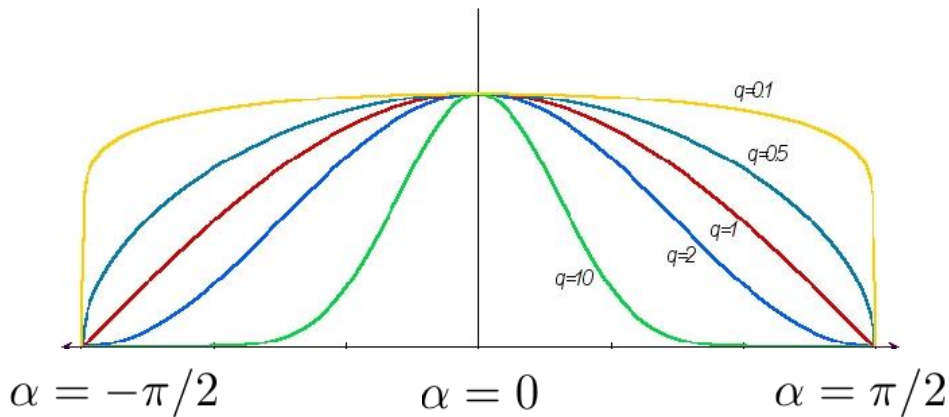
- Parameters

- k_s : specular reflection coefficient
- q : specular reflection exponent



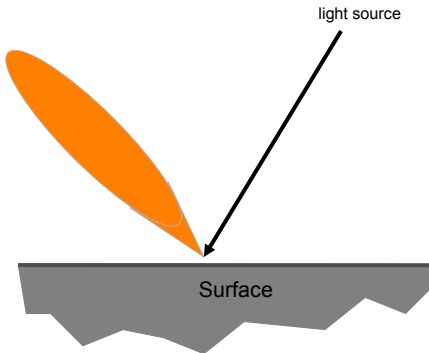
The Phong Model

- Effect of q – the specular reflection exponent



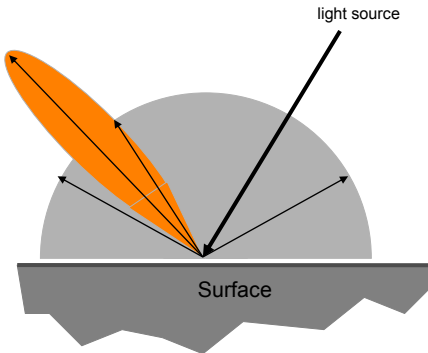
Terminology: Specular Lobe

- The specular reflection distribution is usually called a “lobe”
 - For Phong, its shape is $(\mathbf{r} \cdot \mathbf{v})^q$



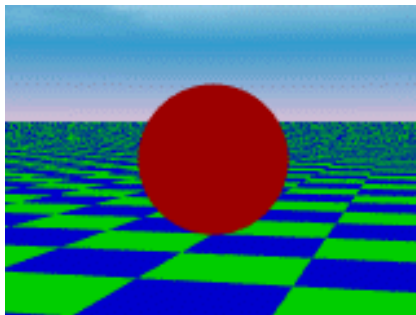
The Complete Phong Model

- Sum of three components:
ideal diffuse reflection +
specular reflection +
“ambient”.



Ambient Illumination













- Represents the reflection of all indirect illumination.
- This is a total hack!
- Avoids the complexity of indirect (“global”) illumination



Putting It All Together

- Phong Illumination Model

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

Phong	ρ_{ambient}	ρ_{diffuse}	ρ_{specular}	ρ_{total}
$\phi_i = 60^\circ$				
$\phi_i = 25^\circ$				
$\phi_i = 0^\circ$				

Putting It All Together

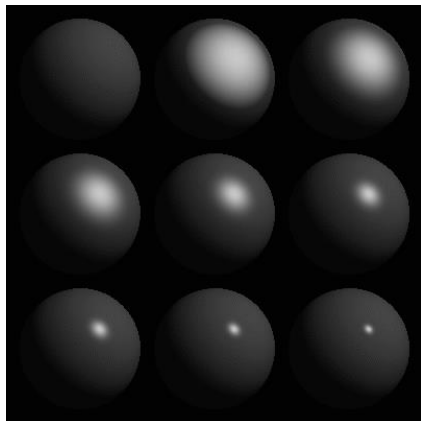
- Phong Illumination Model

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

- Is it physically based?
 - No, does not even conserve energy, may well reflect more energy than what goes in
 - Furthermore, it does not even conform to the BRDF model directly (we are taking the proper cosine for diffuse, but not for specular)
 - And ambient was a total hack

Phong Examples

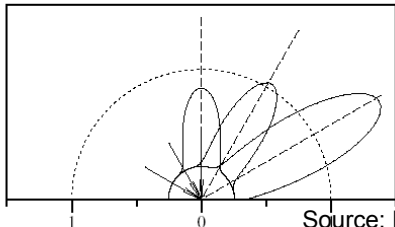
- The spheres illustrate specular reflections as the direction of the light source and the exponent q (amount of shininess) is varied.



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

Fresnel Reflection

- Increasing specularity near grazing angles.
 - Most BRDF models account for this.



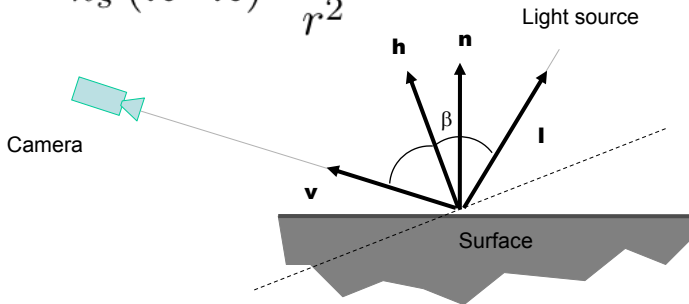
Source: Lafortune et al. 97

Questions?

Blinn-Torrance Variation of Phong

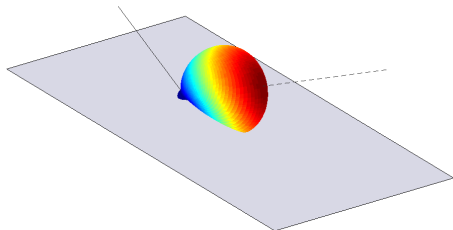
- Uses the “halfway vector” \mathbf{h} between \mathbf{l} and \mathbf{v} .

$$L_o = k_s \cos(\beta)^q \frac{L_i}{r^2}$$
$$= k_s (\mathbf{n} \cdot \mathbf{h})^q \frac{L_i}{r^2}$$
$$\mathbf{h} = \frac{\mathbf{l} + \mathbf{v}}{\|\mathbf{l} + \mathbf{v}\|}$$

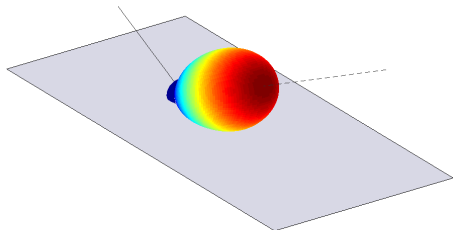


Lobe Comparison

- Half vector lobe
 - Gradually narrower when approaching grazing
- Mirror lobe
 - Always circular



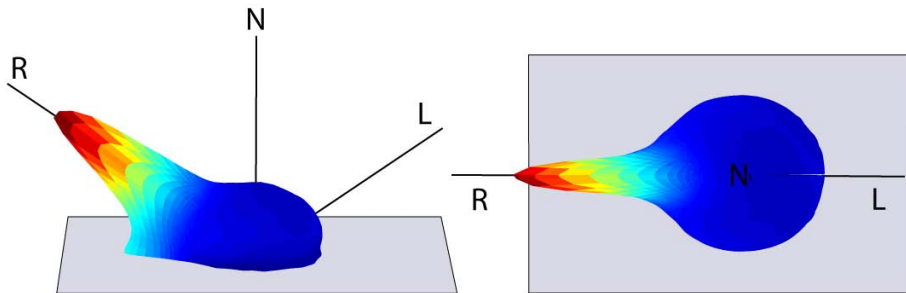
Half vector lobe



Mirror lobe

Half Vector Lobe is Better

- More consistent with what is observed in measurements ([Ngan, Matusik, Durand 2005](#))



Courtesy of Mitsubishi Electric Research Laboratories, Inc. Used with permission.

Example: Plot of "PVC" BRDF at 55° incidence

Questions?

Microfacet Theory

- Example
 - Think of water surface as lots of tiny mirrors (microfacets)
 - “Bright” pixels are
 - Microfacets aligned with the vector between sun and eye
 - But not the ones in shadow
 - And not the ones that are occluded

Image of sunset removed due to copyright restrictions.

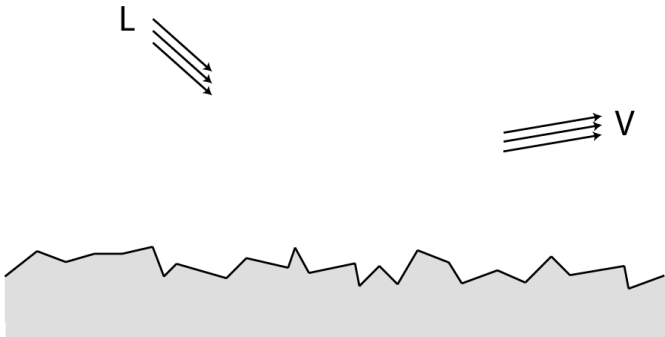
Microfacet Theory

- Model surface by tiny mirrors
[Torrance & Sparrow 1967]



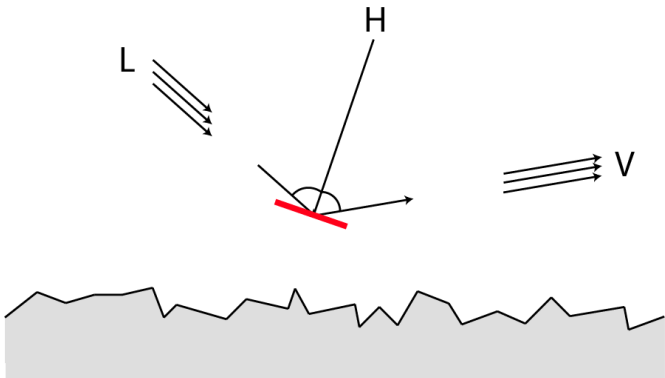
Microfacet Theory

- Value of BRDF at (L, V) is a product of
 - number of mirrors oriented halfway between L and V



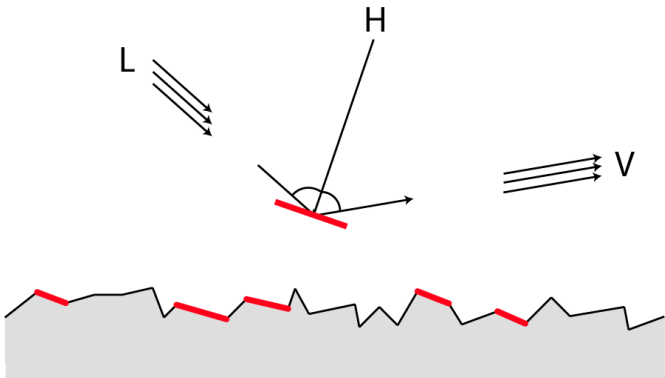
Microfacet Theory

- Value of BRDF at (L, V) is a product of
 - number of mirrors oriented halfway between L and V



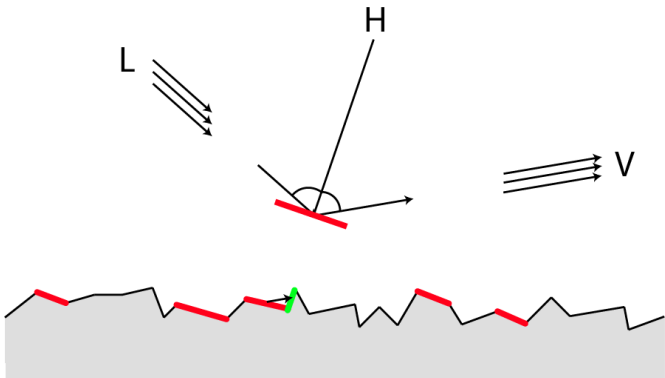
Microfacet Theory

- Value of BRDF at (L, V) is a product of
 - number of mirrors oriented halfway between L and V



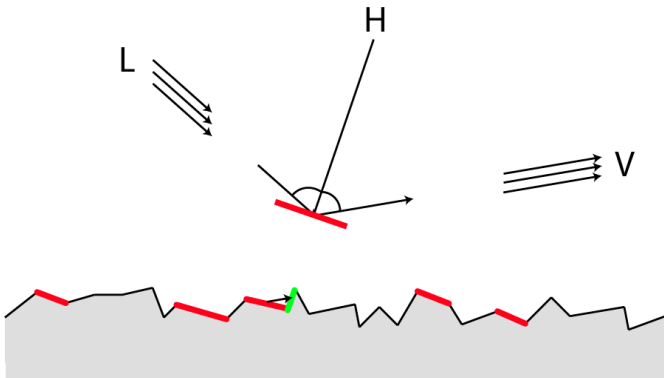
Microfacet Theory

- Value of BRDF at (L, V) is a product of
 - number of mirrors oriented halfway between L and V
 - ratio of the un(shadowed/masked) mirrors



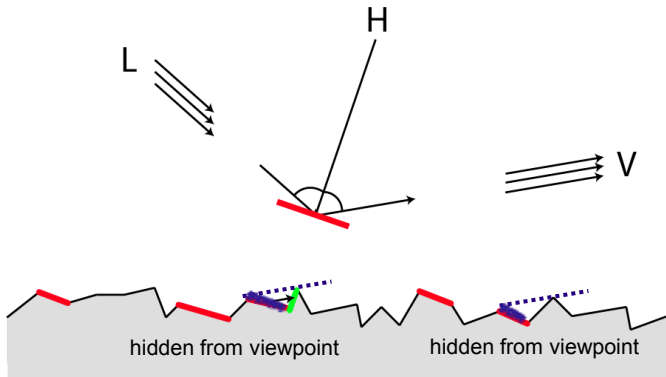
Microfacet Theory

- Value of BRDF at (L, V) is a product of
 - number of mirrors oriented halfway between L and V
 - ratio of the un(shadowed/masked) mirrors
 - Fresnel coefficient



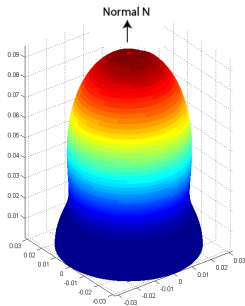
Shadowing and Masking

- Some facets are hidden from viewpoint
- Some are hidden from the light



Microfacet Theory-based Models

- Develop BRDF models by imposing simplifications [Torrance-Sparrow 67], [Blinn 77], [Cook-Torrance 81], [Ashikhmin et al. 2000]
- Model the distribution $p(H)$ of microfacet normals
 - Also, statistical models for shadows and masking



spherical plot of a Gaussian-like $p(H)$

Full Cook-Torrance Lobe

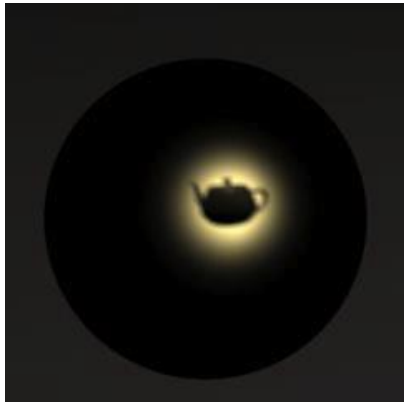
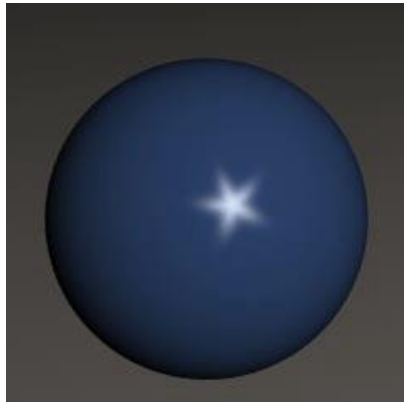
- ρ_s is the specular coefficient (3 numbers RGB)
- D is the microfacet distribution
 - δ is the angle between the half vector H and the normal N
 - m defines the roughness (width of lobe)
- G is the shadowing and masking term
- Need to add a diffuse term

$$K = \frac{\rho_s}{\pi} \frac{DG}{(N \cdot L)(N \cdot V)} \text{Fresnel}(F_0, V \cdot H)$$

where $G = \min\left\{1, \frac{2(N \cdot H)(N \cdot V)}{(V \cdot H)}, \frac{2(N \cdot H)(N \cdot L)}{(V \cdot H)}\right\}$ and $D = \frac{1}{m^2 \cos^4 \delta} e^{-[(\tan \delta)/m]^2}$

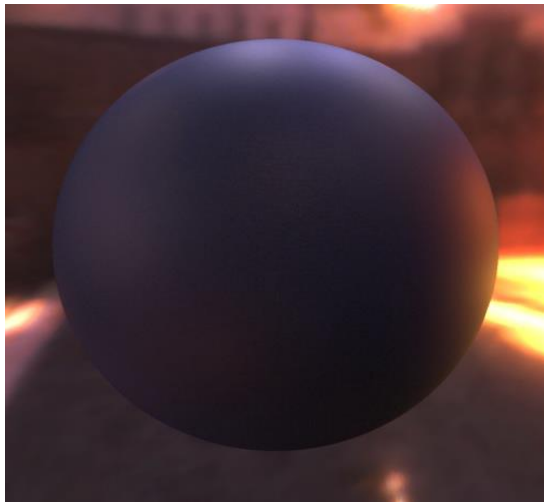
Questions?

- “Designer BRDFs” by [Ashikhmin et al.](#)



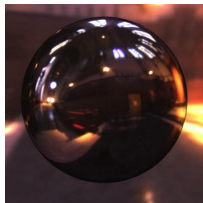
© ACM. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

BRDF Examples from Ngan et al.



Material – Dark blue paint

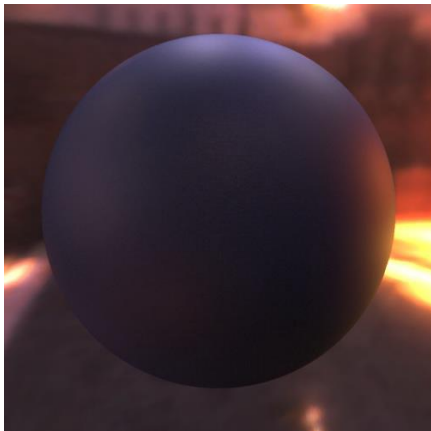
Lighting



Courtesy of Mitsubishi Electric Research Laboratories, Inc. Used with permission.

Dark Blue Paint

Acquired data



Blinn-Phong

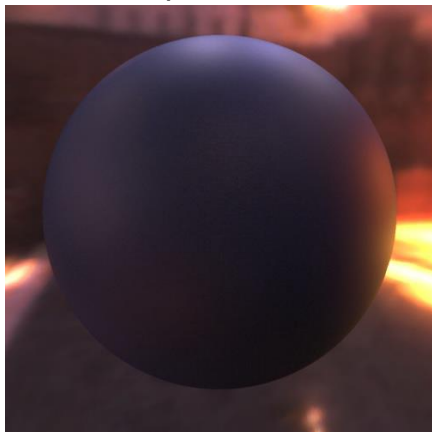


Courtesy of Mitsubishi Electric Research Laboratories, Inc. Used with permission.

Finding the BRDF model parameters that best reproduce the real material
Material – Dark blue paint

Dark Blue Paint

Acquired data



Cook-Torrance



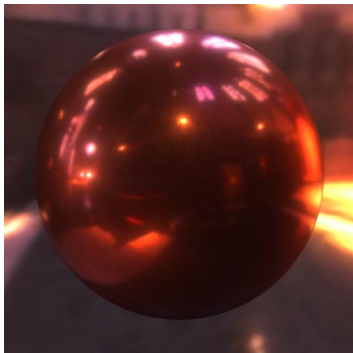
Courtesy of Mitsubishi Electric Research Laboratories, Inc. Used with permission.

Finding the BRDF model parameters that best reproduce the real material
Material – Dark blue paint

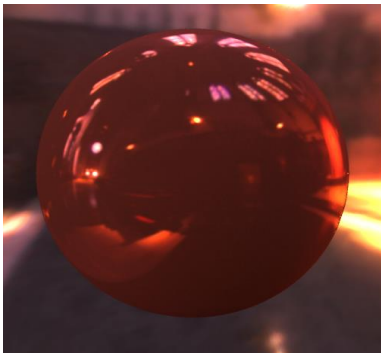
Observations

- Some materials impossible to represent with a single lobe

Acquired data



Cook-Torrance

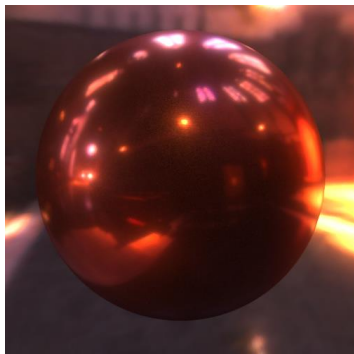


Material – Red Christmas Ball

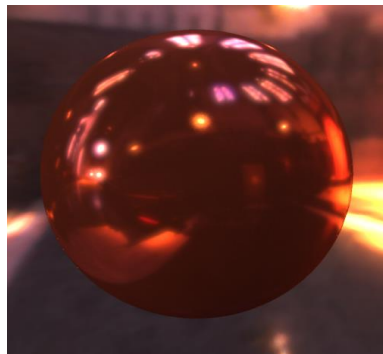
Adding a Second Lobe

- Some materials impossible to represent with a single lobe

Acquired data



Cook-Torrance 2 lobes



Material – Red Christmas Ball

Courtesy of Mitsubishi Electric Research Laboratories, Inc. Used with permission.

Image-Based Acquisition

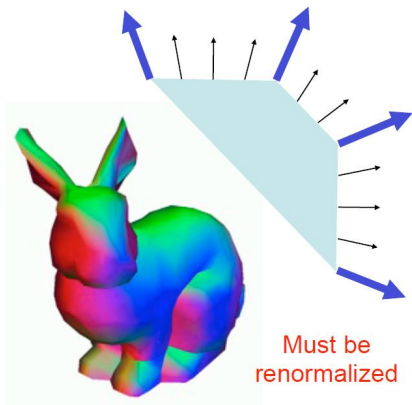
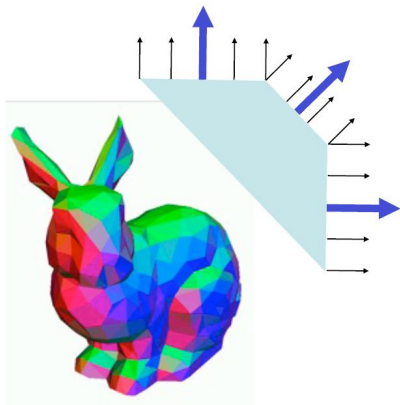
- A Data-Driven Reflectance Model, SIGGRAPH 2003
 - The data is available
<http://people.csail.mit.edu/wojciech/BRDFDatabase/>



Phong Normal Interpolation

(Not Phong
Shading)

- Interpolate the average vertex normals across the face and use this in shading computations
 - Again, use barycentric interpolation!



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. . .

Particle Systems and ODEs

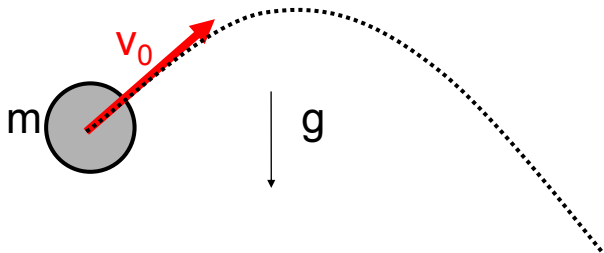
Image removed due to copyright restrictions.

Types of Animation

- Keyframing
- Procedural
- Physically-based
 - Particle Systems: **TODAY**
 - Smoke, water, fire, sparks, etc.
 - Usually heuristic as opposed to simulation, but not always
 - Mass-Spring Models (Cloth) **NEXT CLASS**
 - Continuum Mechanics (fluids, etc.), finite elements
 - Not in this class
 - Rigid body simulation
 - Not in this class

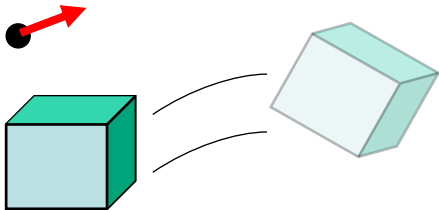
Types of Animation: Physically-Based

- 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



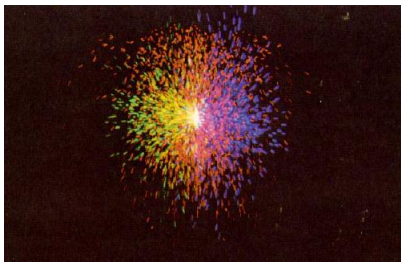
Types of Dynamics

- Point
- Rigid body
- Deformable body
(include clothes, fluids, smoke, etc.)

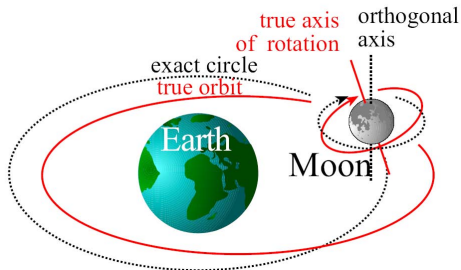


Today We Focus on Point Dynamics

- Lots of points!
- Particles systems
 - Borderline between procedural and physically-based



© ACM. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

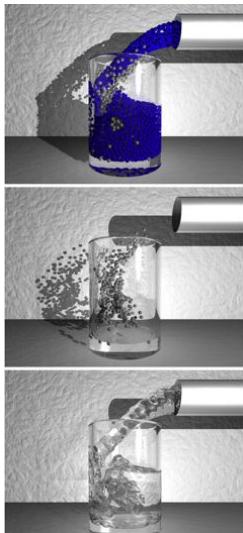


Particle Systems Overview

- **Emitters** generate tons of “particles”
- Describe the external **forces** with a force field
- **Integrate** the laws of mechanics (ODEs)
- In the simplest case, each particle is **independent**
- If there is enough **randomness** (in particular at the emitter) you get nice effects
 - sand, dust, smoke, sparks, flame, water, ...

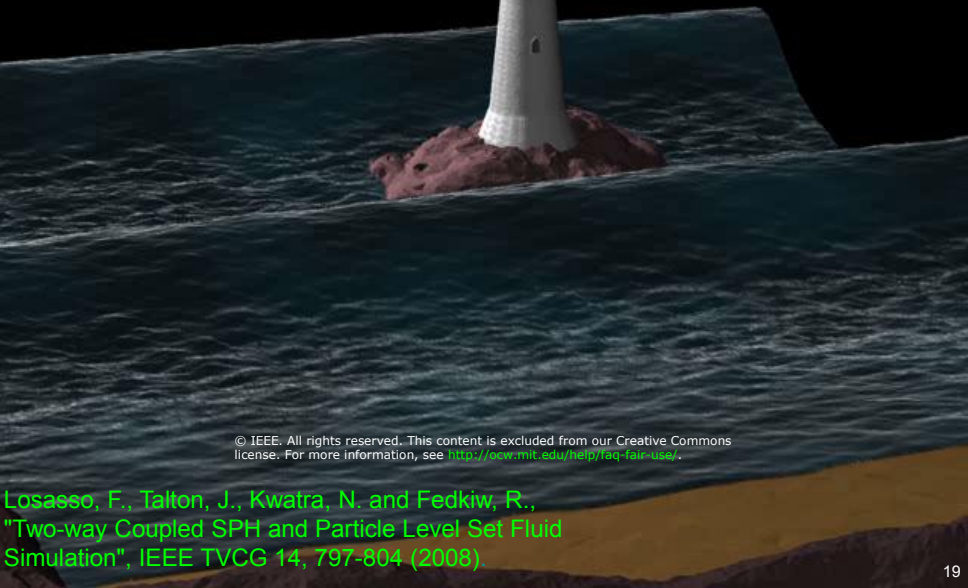
Images of particle systems removed due to copyright restrictions.

- It's not all hacks:
 - Smoothed Particle Hydrodynamics (SPH)
 - A family of “real” particle-based fluid simulation techniques.
 - Fluid flow is described by the **Navier-Stokes Equations**, a nonlinear partial differential equation (PDE)
 - SPH discretizes the fluid as small packets (particles!), and evaluates pressures and forces based on them.



© ACM. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

These Stanford folks use SPH for resolving the small-scale spray and mist that would otherwise be too much for the grid solver to handle.



© IEEE. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

Losasso, F., Talton, J., Kwatra, N. and Fedkiw, R.,
"Two-way Coupled SPH and Particle Level Set Fluid
Simulation", IEEE TVCG 14, 797-804 (2008).

Take-Home Message

- Particle-based methods can range from pure heuristics (hacks that happen to look good) to “real” simulation
- Basics are the same:
Things always boil down to integrating ODEs!
 - Also in the case of grids/computational meshes

Andrew Selle et al.



© ACM. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

What is a Particle System?

- Collection of many small simple pointlike things
 - Described by their current state: position, velocity, age, color, etc.
- Particle motion influenced by external force fields and internal forces between particles
- Particles created by **generators** or **emitters**
 - With some randomness
- Particles often have lifetimes
- Particles are often independent
- Treat as points for dynamics, but rendered as anything you want

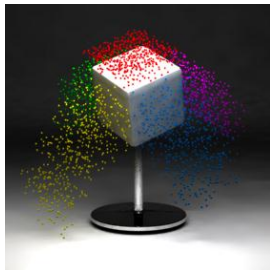


Image courtesy of [Halixi72](#) on Wikimedia Commons. License: CC-BY-SA. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

Simple Particle System: Sprinkler

PL: linked list of particle = empty;
spread=0.1; *//how random the initial velocity is*
colorSpread=0.1; *//how random the colors are*

For each time step

 Generate k particles

 p=new particle();
 p->position=(0,0,0);
 p->velocity=(0,0,1)+spread*(rnd(), rnd(), rnd());
 p.color=(0,0,1)+colorSpread*(rnd(), rnd(), rnd());
 PL->add(p);

Image by Jeff Lander removed due to copyright restrictions.

For each particle p in PL

 p->position+=p->velocity*dt; *//dt: time step*
 p->velocity-=g*dt; *//g: gravitation constant*
 glColor(p.color);
 glVertex(p.position);

Demo with Processing

- <http://processing.org/learning/topics/simpleparticlesystem.html>

Ordinary Differential Equations

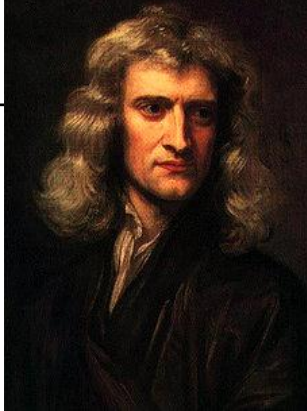
$$\frac{d \mathbf{X}(t)}{dt} = f(\mathbf{X}(t), t)$$

- Given a function $f(\mathbf{X}, t)$ compute $\mathbf{X}(t)$
- Typically, *initial value problems*:
 - Given values $\mathbf{X}(t_0) = \mathbf{X}_0$
 - Find values $\mathbf{X}(t)$ for $t > t_0$
- We can use lots of standard tools

Newtonian Mechanics

- Point mass: 2nd order ODE

$$\vec{F} = m\vec{a} \quad \text{or} \quad \vec{F} = m \frac{d^2 \vec{x}}{dt^2}$$




This image is in the public domain.
Source: [Wikimedia Commons](#).

- Position \mathbf{x} and force \mathbf{F} are vector quantities
 - We know \mathbf{F} and m , want to solve for \mathbf{x}
- You have all seen this a million times before

Now, Many Particles

- We have N point masses
 - Let's just stack all \mathbf{x} s and \mathbf{v} s in a big vector of length $6N$
 - \mathbf{F}^i denotes the force on particle i
 - When particles don't interact, \mathbf{F}^i only depends on \mathbf{x}_i and \mathbf{v}_i .

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{v}_1 \\ \vdots \\ \mathbf{x}_N \\ \mathbf{v}_N \end{pmatrix} \quad f(\mathbf{X}, t) = \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{F}^1(\mathbf{X}, t) \\ \vdots \\ \mathbf{v}_N \\ \mathbf{F}^N(\mathbf{X}, t) \end{pmatrix}$$


 f gives $d/dt \mathbf{X}$, remember!

Path through a Vector Field

- $\mathbf{X}(t)$: path in multidimensional phase space

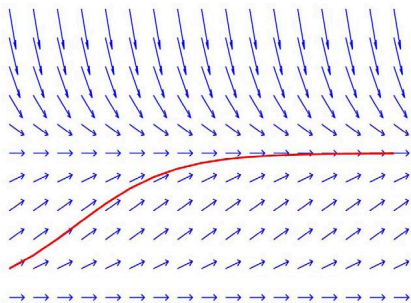


Image by MIT OpenCourseWare.

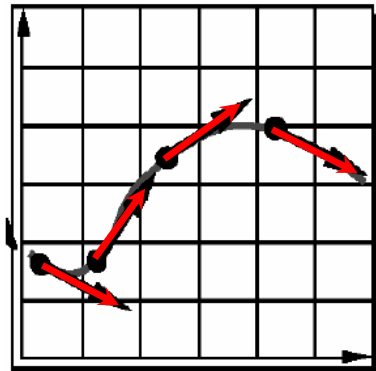
$$\frac{d}{dt}\mathbf{X} = f(\mathbf{X}, t)$$

“When we are at state \mathbf{X} at time t , where will \mathbf{X} be after an infinitely small time interval dt ?”

- $f = d/dt \mathbf{X}$ is a vector that sits at each point in phase space, pointing the direction.

Intuitive Solution: Take Steps

- Current state \mathbf{X}
- Examine $f(\mathbf{X}, t)$ at (or near) current state
- Take a step to new value of \mathbf{X}



$$\frac{d}{dt}\mathbf{X} = f(\mathbf{X}, t)$$

$$\Rightarrow \text{“}d\mathbf{X} = dt f(\mathbf{X}, t)\text{”}$$

$f = d/dt \mathbf{X}$ is a vector that sits at each point in phase space, pointing the direction.

Euler's Method

- Simplest and most intuitive
- Pick a **step size** h
- Given $\mathbf{X}_0 = \mathbf{X}(t_0)$, take step:

$$t_1 = t_0 + h$$

$$\mathbf{X}_1 = \mathbf{X}_0 + h f(\mathbf{X}_0, t_0)$$

- Piecewise-linear approximation to the path
- **Basically, just replace dt by a small but finite number**

Euler's method: Inaccurate

- Moves along tangent; can leave solution curve, e.g.:

$$f(\mathbf{X}, t) = \begin{pmatrix} -y \\ x \end{pmatrix}$$

- Exact solution is circle:

$$\mathbf{X}(t) = \begin{pmatrix} r \cos(t+k) \\ r \sin(t+k) \end{pmatrix}$$

- Euler spirals outward
no matter how small h is
 - will just diverge more slowly

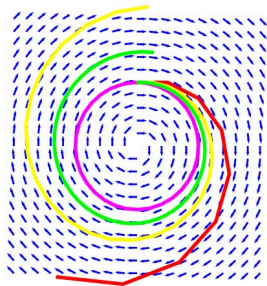


Image by MIT OpenCourseWare.

More Accurate Alternatives

- Midpoint, Trapezoid, Runge-Kutta
 - Also, “implicit methods” (next week)

More on this during next class

- Extremely valuable resource: [SIGGRAPH 2001 course notes on physically based modeling](#)

Processing demo

- <http://processing.org/learning/topics/smokeparticlesystem.html>

Massive software

- <http://www.massivesoftware.com/>
- Used for battle scenes in the Lord of The Rings

Processing demo

- <http://processing.org/learning/topics/flocking.html>

Particle Modeling [Reeves 1983]

- The grass is made of particles
 - The entire lifetime of the particle is drawn at once.
 - This can be done procedurally on the GPU these days!



Particle Systems and ODE Solvers II, Mass-Spring Modeling

With slides from Jaakko Lehtinen
and others

Image removed due to copyright restrictions.

Euler's Method: Inaccurate

- Moves along tangent; can leave solution curve, e.g.:

$$f(\mathbf{X}, t) = \begin{pmatrix} -y \\ x \end{pmatrix}$$

- Exact solution is circle:

$$\mathbf{X}(t) = \begin{pmatrix} r \cos(t+k) \\ r \sin(t+k) \end{pmatrix}$$

- Euler spirals outward
no matter how small h is
 - will just diverge more slowly

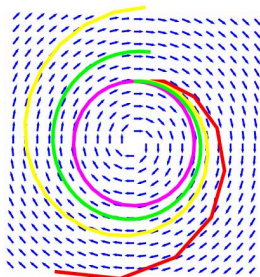
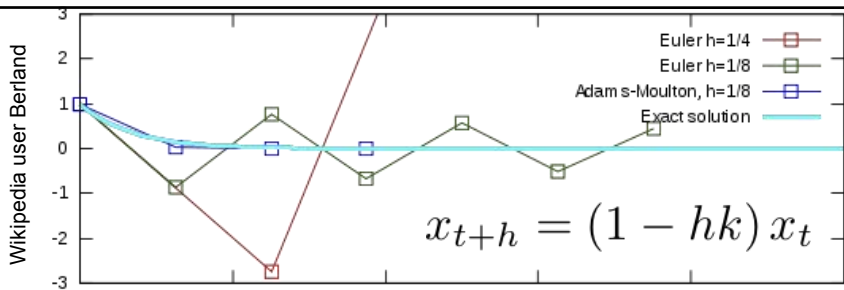


Image by MIT OpenCourseWare.

Euler's Method: Not Always Stable



This image is in the public domain. Source: [Wikimedia](#)

- Limited step size!
 - When $0 \leq (1 - hk) < 1 \Leftrightarrow h < 1/k$
things are fine, the solution decays
 - When $-1 \leq (1 - hk) \leq 0 \Leftrightarrow 1/k \leq h \leq 2/k$
we get oscillation
 - When $(1 - hk) < -1 \Leftrightarrow h > 2/k$ things explode

Analysis: Taylor Series

- Expand exact solution $\mathbf{X}(t)$

$$\mathbf{X}(t_0 + h) = \mathbf{X}(t_0) + h \left(\frac{d}{dt} \mathbf{X}(t) \right) \Big|_{t_0} + \frac{h^2}{2!} \left(\frac{d^2}{dt^2} \mathbf{X}(t) \right) \Big|_{t_0} + \frac{h^3}{3!} (\dots) + \dots$$

- Euler's method approximates:

$$\mathbf{X}(t_0 + h) = \mathbf{X}_0 + h f(\mathbf{X}_0, t_0) \quad \dots + O(h^2) \text{ error}$$

$$h \rightarrow h/2 \Rightarrow \text{error} \rightarrow \text{error}/4 \text{ per step} \times \text{twice as many steps} \\ \rightarrow \text{error}/2$$

- First-order method: Accuracy varies with h
- To get 100x better accuracy need 100x more steps

Can We Do Better?

- Problem: f varies along our Euler step
- Idea 1: look at f at the arrival of the step and compensate for variation

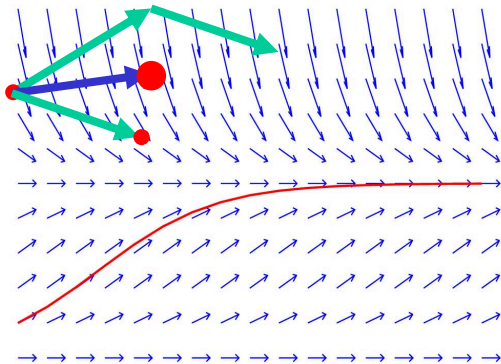


Image by MIT OpenCourseWare.

2nd Order Methods

- This translates to...

$$\begin{aligned} f_0 &= f(\mathbf{X}_0, t_0) \\ f_1 &= f(\mathbf{X}_0 + h f_0, t_0 + h) \end{aligned}$$

- and we get

$$\mathbf{X}(t_0 + h) = \mathbf{X}_0 + \frac{h}{2}(f_0 + f_1) + O(h^3)$$

- This is the *trapezoid method*
 - Analysis omitted (see 6.839)
- Note: What we mean by “2nd order” is that the error goes down with h^2 , not h – the equation is still 1st order!

Can We Do Better?

- Problem: f has varied along our Euler step
- Idea 2: look at f after a smaller step, use that value for a full step from initial position

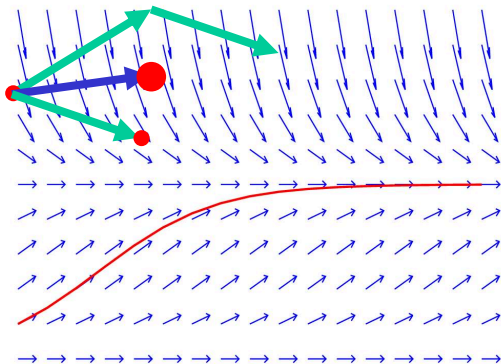


Image by MIT OpenCourseWare.

2nd Order Methods Cont'd

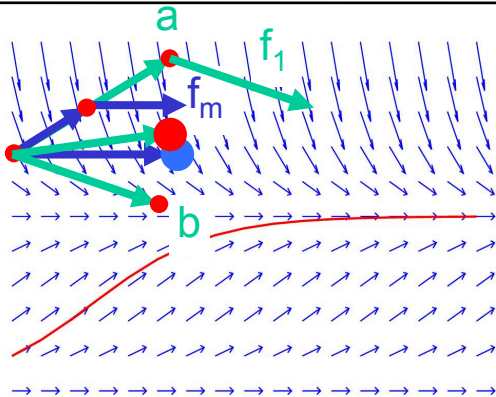
- This translates to...

$$\begin{aligned} f_0 &= f(\mathbf{X}_0, t_0) \\ f_m &= f\left(\mathbf{X}_0 + \frac{h}{2} f_0, t_0 + \frac{h}{2}\right) \end{aligned}$$

- and we get $\mathbf{X}(t_0 + h) = \mathbf{X}_0 + h f_m + O(h^3)$
- This is the *midpoint method*
 - Analysis omitted again,
but it's not very complicated, see [here](#).

Comparison

- **Midpoint:**
 - $\frac{1}{2}$ Euler step
 - evaluate f_m
 - full step using f_m
- **Trapezoid:**
 - Euler step (a)
 - evaluate f_l
 - full step using f_l (b)
 - average (a) and (b)
- Not exactly same result,
but same order of accuracy



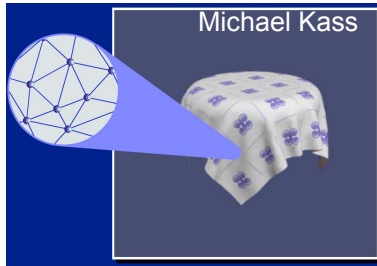
Can We Do Even Better?

- You bet!
- You will implement **Runge-Kutta** for assignment 3
- Again, see **Witkin, Baraff, Kass: Physically-based Modeling Course Notes, SIGGRAPH 2001**

- See eg
<http://www.youtube.com/watch?v=HbE3L5CIdQg>

Mass-Spring Modeling

- Beyond pointlike objects: strings, cloth, hair, etc.
- Interaction between particles
 - Create a network of spring forces that link pairs of particles



© ACM. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

- First, slightly hacky version of cloth simulation
- Then, some motivation/intuition for *implicit integration* (NEXT LECTURE)

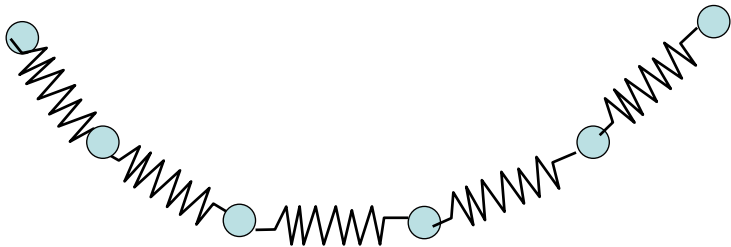
Springs



Image courtesy of [Jean-Jacques MILAN](#) on Wikimedia Commons. License: CC-BY-SA. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

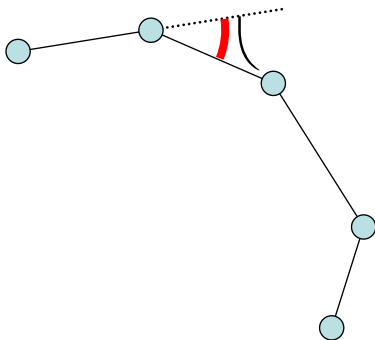
How Would You Simulate a String?

- Springs link the particles
- Springs try to keep their rest lengths and preserve the length of the string
- Not exactly preserved though, and we get oscillation
 - Rubber band approximation



Hair

- Linear set of particles
- Length-preserving **structural** springs like before
- **Deformation** forces proportional to the angle between segments
- **External** forces



Springs for Cloth

- Network of masses and springs
- **Structural** springs:
 - link (i, j) and $(i+1, j)$;
and (i, j) and $(i, j+1)$
- **Deformation:**
 - Shear springs
 - (i, j) and $(i+1, j+1)$
 - Flexion springs
 - (i, j) and $(i+2, j)$;
 (i, j) and $(i, j+2)$
- See [Provot's Graphics Interface '95 paper](#) for details

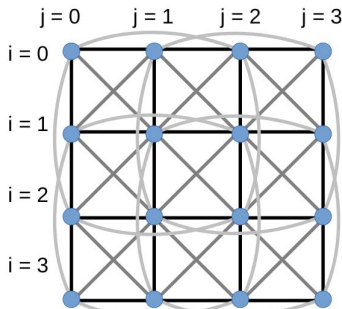


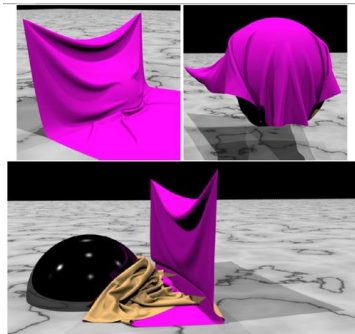
Image by MIT OpenCourseWare.

Provot 95

Collisions

Robert Bridson, Ronald Fedkiw & John Anderson
Robust Treatment of Collisions, Contact
and Friction for Cloth Animation
SIGGRAPH 2002

- Cloth has many points of contact
- Need efficient collision detection and stable treatment

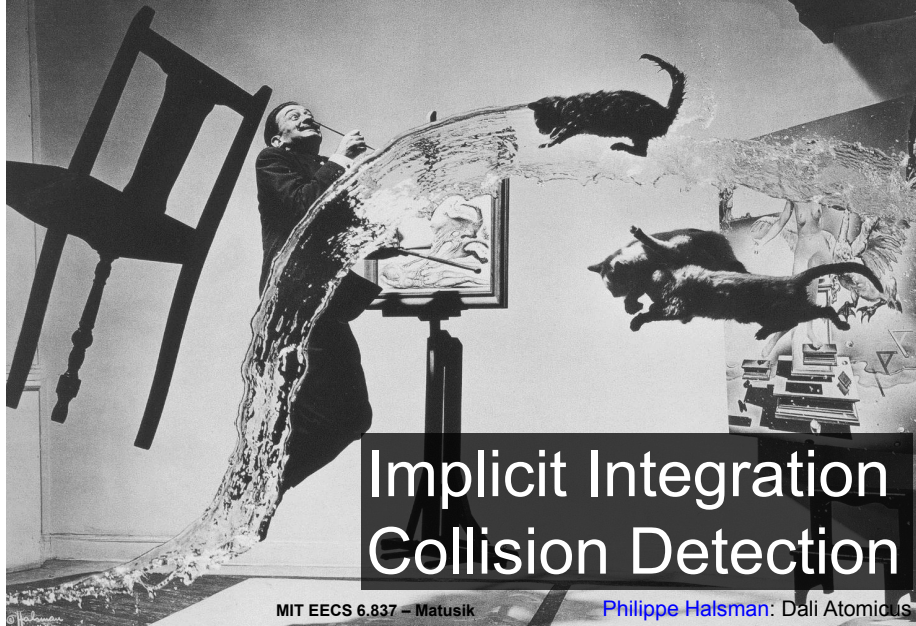


© ACM. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

© source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

Cool Cloth/Hair Demos

- Robert Bridson, Ronald Fedkiw & John Anderson: Robust Treatment of Collisions, Contact and Friction for Cloth Animation SIGGRAPH 2002
- Selle. A, Su, J., Irving, G. and Fedkiw, R., "Robust High-Resolution Cloth Using Parallelism, History-Based Collisions, and Accurate Friction," IEEE TVCG 15, 339-350 (2009).
- Selle, A., Lentine, M. and Fedkiw, R., "A Mass Spring Model for Hair Simulation", SIGGRAPH 2008, ACM TOG 27, 64.1-64.11 (2008).



Implicit Integration Collision Detection

MIT EECS 6.837 – Matusik

Philippe Halsman: Dali Atomicus

Plan

- Implementing Particle Systems
- Implicit Integration
- Collision detection and response
 - Point-object and object-object detection
 - Only point-object response

ODEs and Numerical Integration

$$\frac{d \mathbf{X}(t)}{dt} = f(\mathbf{X}(t), t)$$

- Given a function $f(\mathbf{X}, t)$ compute $\mathbf{X}(t)$
- Typically, *initial value problems*:
 - Given values $\mathbf{X}(t_0) = \mathbf{X}_0$
 - Find values $\mathbf{X}(t)$ for $t > t_0$
- We can use lots of standard tools

Integrator Comparison

- **Midpoint:**

- $\frac{1}{2}$ Euler step
- evaluate f_m
- full step using f_m

- **Trapezoid:**

- Euler step (a)
- evaluate f_1
- full step using f_1 (b)
- average (a) and (b)

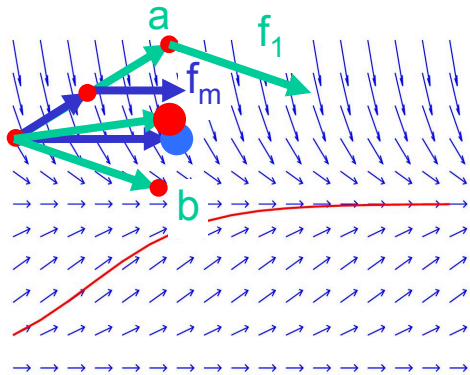


Image by MIT OpenCourseWare.

- **Better than Euler but still a speed limit**

Midpoint Speed Limit

- $x' = -kx$
- First half Euler step: $x_m = x - 0.5 h k x = x(1 - 0.5 h k)$
- Read derivative at x_m : $f_m = -k x_m = -k(1 - 0.5 h k)x$
- Apply derivative at origin:
 $x(t+h) = x + h f_m = x - h k (1 - 0.5 h k)x = x(1 - h k + 0.5 h^2 k^2)$
- Looks a lot like Taylor...
- We want $0 < x(t+h)/x(t) < 1$
 $-h k + 0.5 h^2 k^2 < 0$
 $h k (-1 + 0.5 h k) < 0$
For positive values of h & $k \Rightarrow h < 2/k$
- Twice the speed limit of Euler

Stiffness

- In more complex systems, step size is limited by the largest k .
 - One stiff spring can ruin things for everyone else!
- Systems that have some big k values are called *stiff systems*.
- In the general case, k values are eigenvalues of the local Jacobian!

From the siggraph PBM notes

Simple Closed Form Case

Implicit Euler is unconditionally stable!

- Explicit Euler: $x(t+h) = (1-hk) x(t)$
- Implicit Euler: $x(t+h) = x(t) + h x'(t+h)$
 $x(t+h) = x(t) - h k x(t+h)$
 $= x(t) / (1+hk)$
 - It is a hyperbola!
 $1/(1+hk) < 1,$
when $h, k > 0$

Newton's Method – N Dimensions

- Now locations \mathbf{X}_i , \mathbf{X}_{i+1} and F are N -D
- Newton solution of $F(\mathbf{X}_{i+1}) = 0$ is just like 1D:

$$J_F(\mathbf{X}_i)(\mathbf{X}_{i+1} - \mathbf{X}_i) = -F(\mathbf{X}_i)$$

$N \times N$ Jacobian matrix unknown N -D step from current to next guess

$$J_F(\mathbf{X}_i) = \left[\frac{\partial F}{\partial X} \right]_{\mathbf{X}_i}$$

- Must solve a linear system at each step of Newton iteration
 - Note that also Jacobian changes for each step

Detecting Collisions

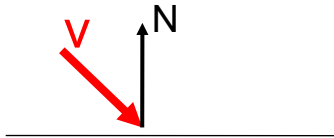
- Easy with implicit equations of surfaces:

$H(x,y,z) = 0$ on the surface

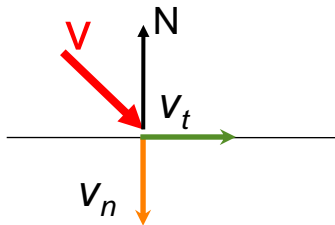
$H(x,y,z) < 0$ inside surface

- So just compute H and you know that you are inside if it is negative
- More complex with other surface definitions like meshes
 - A mesh is not necessarily even closed, what is inside?

Collision Response for Particles



Collision Response for Particles



$$V = V_n + V_t$$

normal component
tangential component

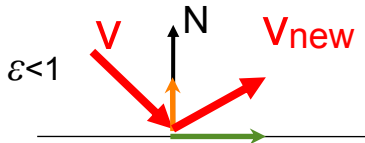
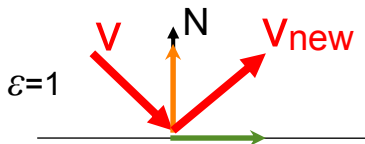
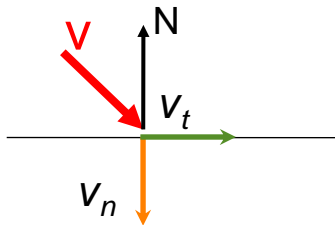
Collision Response for Particles

- Tangential velocity v_t
often unchanged
- Normal velocity v_n reflects:

$$v = v_t + v_n$$

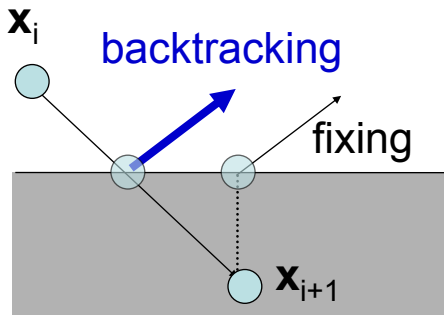
$$v \leftarrow v_t - \varepsilon v_n$$

- Coefficient of restitution ε
- When $\varepsilon = 1$, mirror reflection



Collisions – Overshooting

- Usually, we detect collision when it is too late: we are already inside
- Solution: Back up
 - Compute intersection point
 - Ray-object intersection!
 - Compute response there
 - Advance for remaining fractional time step
- Other solution: Quick and dirty hack
 - Just project back to object closest point



Collision Detection in Big Scenes

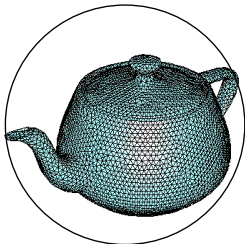
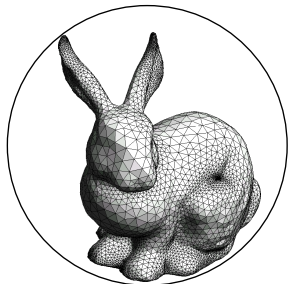
- Imagine we have n objects. Can we test all pairwise intersections?
 - Quadratic cost $O(n^2)$!
- Simple optimization: separate static objects
 - But still $O(\text{static} \times \text{dynamic} + \text{dynamic}^2)$

Hierarchical Collision Detection

- Use simpler conservative proxies (e.g. bounding spheres)
- Recursive (hierarchical) test
 - Spend time only for parts of the scene that are close
- Many different versions, we will cover only one

Bounding Spheres

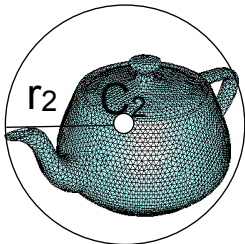
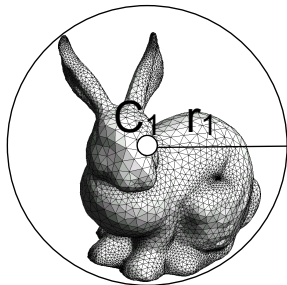
- Place spheres around objects
- If spheres do not intersect, neither do the objects!
- Sphere-sphere collision test is easy.



Courtesy of Patrick Laug. Used with permission.

Sphere-Sphere Collision Test

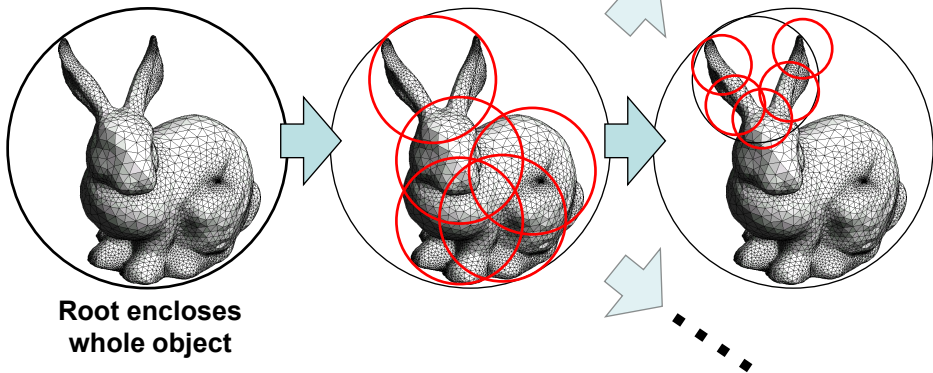
- Two spheres, centers C_1 and C_2 , radii r_1 and r_2
- Intersect only if $\|C_1C_2\| < r_1 + r_2$



Courtesy of Patrick Laug. Used with permission.

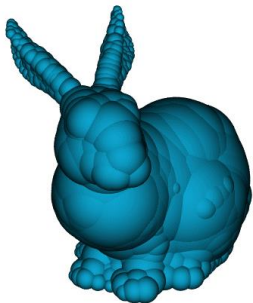
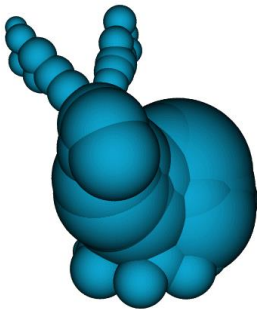
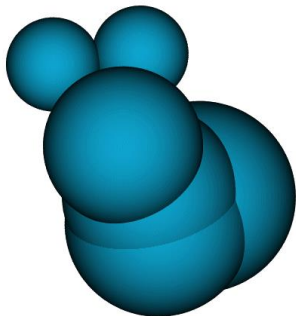
Hierarchical Collision Test

- Hierarchy of bounding spheres
 - Organized in a tree
- Recursive test with early pruning



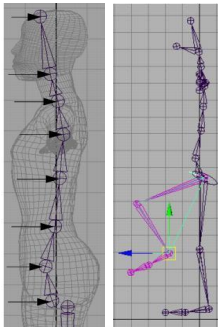
Examples of Hierarchy

- <http://isg.cs.tcd.ie/spheretree/>



How Do They Animate Movies?

- Keyframing mostly
- Articulated figures, inverse kinematics
- Skinning
 - Complex deformable skin, muscle, skin motion
- Hierarchical controls
 - Smile control, eye blinking, etc.
 - Keyframes for these higher-level controls
- A huge time is spent building the 3D models, its skeleton and its controls (rigging)
- Physical simulation for secondary motion
 - Hair, cloths, water
 - Particle systems for “fuzzy” objects



© Maya tutorial. All rights reserved. This content is excluded from our Creative Commons license. For more information, see <http://ocw.mit.edu/help/faq-fair-use/>.

TIEA311 - last slide

Thus, we scratched the surface of something massive.

Where, when, and how to continue:

- ▶ Either TIES471 **next week**
- ▶ Or TIES471 **next year** (hobby projects and more math in-between?)
- ▶ Theses, hobby projects
- ▶ The more math you can do, the better – to move from “copy-paster” to “developer” of algorithms!
- ▶ Remember that the **tax payers spend a lot of money** every day to pay for **your subscription** to research articles! **Make it count! Learn!** Just browse journals through the university IP, using VPN from home.

THANK YOU!