ITTI

Computer Graphics
and Visualisation

Lighting and Shading

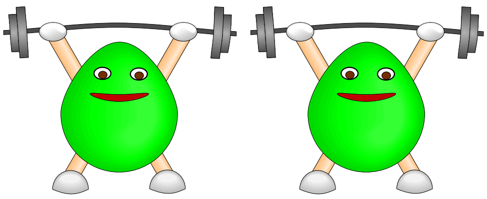
Overhead Projection (OHP) Overviews

Developed by

J Irwin
W T Hewitt
T L J Howard

Computer Graphics Unit
Manchester Computing
University of Manchester

Department of Computer Science
University of Manchester



THE UNIVERSITY
of MANCHESTER

Produced by the ITTI Gravigs Project
Computer Graphics Unit
Manchester Computing
MANCHESTER M13 9PL

First published by UCoSDA in October 1995

ISBN 1 85889 064 0

For further information on this and other modules please contact:
The ITTI Gravigs Project, Computer Graphics Unit, Manchester Computing.
Tel: 0161 275 6095 Email: *gravigs@mcc.ac.uk*

To order further copies of this or other modules please contact:
Mrs Jean Burgan, UCoSDA.
Tel: 0114 272 5248 Email: *j.burgan@sheffield.ac.uk*

These materials have been produced as part of the Information Technology Training Initiative, funded by the Information Systems Committee of the Higher Education Funding Councils.

The authors would like to thank Neil Gatenby for his assistance in providing the source of some material used in this document.

Printed by the Reprographics Department at Manchester Computing from PostScript source supplied by the authors.

Lighting and Shading

Introduction

- Generation of **realistic images** given description of 3D scene...
  - object shape
  - object location and orientation
  - surface properties
  - light sources
- ...and viewing information
  - find out which points are visible
  - find out how these points are lit
- Here, we are only concerned with the second of these problems

Introduction

Having chosen a view of our scene, need to establish how the visible points are illuminated

- Each point has two sources of illumination
  - **direct illumination**
    - light which arrives straight from the light sources
  - **indirect illumination**
    - light which arrives after interacting with the rest of the scene
- Can group algorithms according to how they handle these two components
  - **global illumination** algorithms
    - care is taken to evaluate both
  - **local illumination** algorithms
    - only direct light is accounted for

Introduction

Typically, when generating realistic images, one of two approaches is adopted

- **Empirically-based rendering**
  - image is found using enough fudge-factors to ensure an accurate image
- **Physically-based rendering**
  - image is found by modelling the processes which determine how light is transported around a real scene
- Physically-based algorithms are more computationally expensive, but give better realism

Introduction

We can further classify rendering algorithms

- **Image-space algorithms**
  - first find out visible points and then how these points are illuminated
  - e.g. ray-tracing
- **Object-space algorithms**
  - first find out how the whole scene is illumination and then which points are visible
  - e.g. radiosity

Introduction

This course will cover the following topics

- Local illumination modelling
  - points light sources
  - ambient lighting
  - diffuse and specular reflection
- Shading
  - flat, Gouraud and Phong
- Texture mapping and transparency
  - pattern mapping
  - bump mapping
  - environment mapping

Introduction

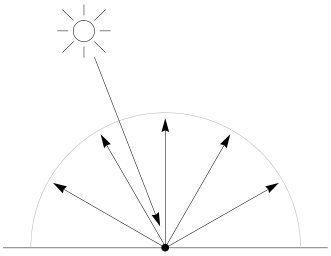
- Ray-tracing
  - image-space algorithm
  - sharp shadows
  - object intersections
  - acceleration techniques
  - image aliasing
- Radiosity
  - object-space algorithm
  - form-factors
  - hemi-cube
  - progressive refinement

Local Illumination Models

Types of Reflection

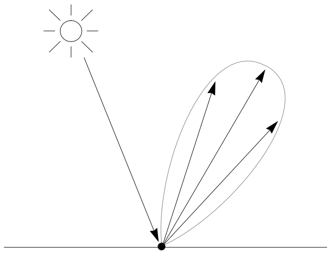
There are two ways in which reflection can occur

- **Diffuse reflection**
  - light is scattered uniformly, making the surface look *matte*



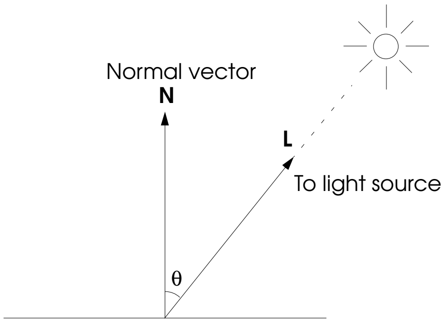
Types of Reflection

- **Specular reflection**
  - a large proportion of the incident light is reflected over a narrow range of angles, making the surface look *shiny*



- **Combined reflection**
  - in practice, many materials exhibit both of these effects to different degrees

Surface Orientation



- **N** is **surface normal** and
- **L** is direction *to* light source
- Vectors **N** and **L** are *unit vectors*
- θ is **angle of incidence**

Illumination Model 1

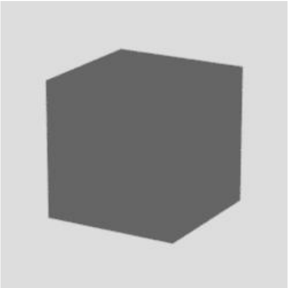
We only consider illumination from ambient lighting

- **Ambient lighting**
  - uniform illumination from all directions
  - arises from multiple reflections
  - I_a = intensity of ambient light
  - $I = k_d I_a$
- **Diffuse reflection coefficient**, k_d
  - measures reflectivity of suface for diffuse light
  - values in the range 0 - 1

Illumination Model 1

There is a problem...

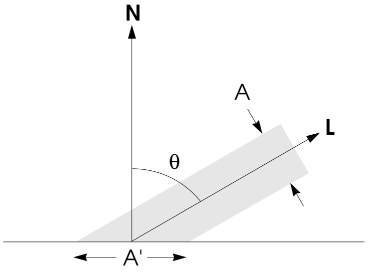
- Lose 3D information because an object is illuminated uniformly
- An example



- Therefore, need to consider reflection of light from a *localised* source

Lambert's Cosine Law

- Amount of light received from light source depends on orientation of surface



- ❑ light intensity I_p of area A
- ❑ light spread over area A'
- ❑ A' = A/cosθ and A = A' cosθ, so...
- ❑ effective intensity I_e at surface is

$$I_e = I_p \cos\theta$$

Diffuse Reflection

We can calculate intensity of diffusely reflected light as

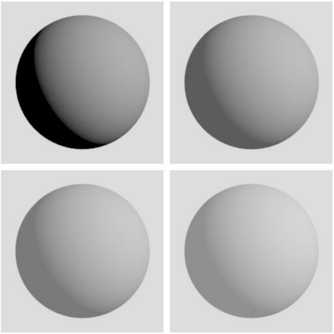
- $I = k_d I_p \cos\theta$
- But $\cos\theta = \mathbf{N} \bullet \mathbf{L}$ since **N** and **L** are unit vectors
- Amount of light diffusely reflected is

$$I = k_d I_p (\mathbf{N} \bullet \mathbf{L})$$

Illumination Model 2

We consider illumination from ambient and a point light source

- I = *ambient* + *diffuse*
- $I = k_d I_a + k_d I_p (\mathbf{N} \bullet \mathbf{L})$

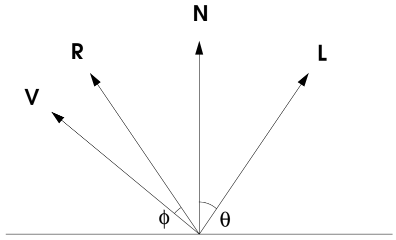


Specular Reflection

Not all surfaces exhibit diffuse reflection

- Surfaces that only show diffuse reflection are dull and **matte**
- In reality, many surfaces are *shiny*
  - ❏ at certain viewing angles, shiny surfaces produce *specular highlights*
  - ❏ highlights occur over a narrow range of angles
  - ❏ colour of highlight usually same as the illuminating light
- Mirrors are examples of ideal specular reflection
  - ❏ angle of incidence equals angle of reflection

More Notation!

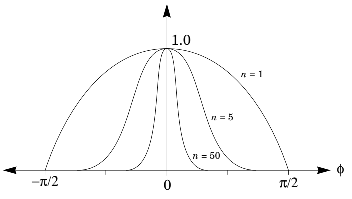


- **N** is surface normal
- **L** is direction *to* light source
- **V** is direction *towards* view point
- **R** is direction of ideal specular reflection
- The intensity of specular reflection depends on the angle ϕ such that $I_s \propto f(\phi)$

Phong Model

Phong's empirical model provides a simple way of controlling the size of the specular highlight

- $f(\phi) = \cos^n\phi$
- *n* depends on surface properties
  - ❏ for perfect reflector $n = \infty$
  - ❏ for very poor reflector $n = 1$
  - ❏ in practice use $1 \le n \le 200$



Phong Model

We can evaluate the cosine term solely in terms of vectors

- With **R** and **L** normalised

$$\cos\phi = \mathbf{R} \bullet \mathbf{V}$$

- Hence

$$I_s \propto (\mathbf{R} \bullet \mathbf{V})^n$$

- Specular reflection also depends on θ, so that

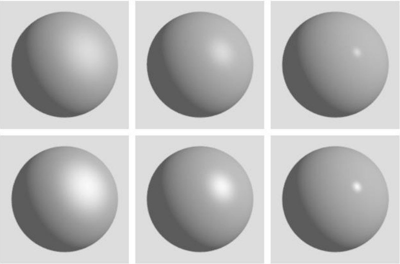
$$I_s \propto W(\theta)\ (\mathbf{R} \bullet \mathbf{V})^n$$

  - ❏ in practice, we set $W(\theta) = k_s$
  - ❏ k_s is the **coefficient of specular reflection**
  - ❏ k_s has values in the range 0 - 1

Illumination Model 3

Reflection from a surface with diffuse and specular properties

- *I = ambient + diffuse + specular*
- $I = k_d I_a + I_p\left[k_d(\mathbf{N} \bullet \mathbf{L}) + k_s(\mathbf{R} \bullet \mathbf{V})^n\right]$
- Examples



Light Source Distance

The intensity of light from a point light source depends on how far away it is

- Physically, intensity falls off as the **square** of the distance
  - the effective intensity of a light source I_p after travelling a distance d is

$$I_e = \frac{I_p}{4\pi d^2}$$

  - in practice, this does not works very well
- Instead, experiment shows that a better formula is

$$I_e = \frac{I_p}{d + d_0}$$

Illumination Model 4

Reflection from a surface when the attenuation of light is included

- *I = ambient + dist-factor(diffuse + specular)*

$$I = k_d I_a + \frac{I_p}{d + d_0}\left[k_d(\mathbf{N} \bullet \mathbf{L}) + k_s(\mathbf{R} \bullet \mathbf{V})^n\right]$$

  - this represents a linear fall-off of intensity
  - d_0 is a constant, used to prevent infinite intensity when $d = 0$
- Multiple light sources
  - use linear superposition

$$I = ambient + \sum_{i=1}^{n} diffuse_i + \sum_{i=1}^{n} specular_i$$

Colour

So far our models make no mention of colour, only light intensities

- Choose a **colour model** and apply the illumination model to each colour component
  - simple colour model is monitor RGB
    - surface defined by k_{dR}, k_{dG} and k_{dB}
    - similarly for the light source
  - an example for the Red component

$$I_R = k_{dR} I_{aR} + \frac{I_{pR}}{d + d_0}\left[k_{dR}(\mathbf{N} \bullet \mathbf{L}) + k_s(\mathbf{R} \bullet \mathbf{V})^n\right]$$

  - assumes specular highlight is the same colour as the light source
- More sophisticated, spectrally-based colour models are available

Applying the Illumination Model

Polygon Shading

Typically, objects are represented by meshes of *polygons*

■ Our illumination model computes the intensity at a *single point* on a surface

■ How can we compute the intensity across the polygon?

- ❏ compute the shade at the centre and use this to represent the whole polygon
  - **flat shading**
- ❏ compute the shade at all points
  - unnecessary and impractical
- ❏ compute shade at key points and interpolate for the rest
  - **Gouraud** and **Phong shading**

Flat Shading

This shading method is the simplest and most computationally efficient

■ Not realistic

- ❏ polygon structure is still evident
  - *faceted* appearance
- ❏ but we can refine the structue to reduce the visual effect

■ Problem with **Mach banding**

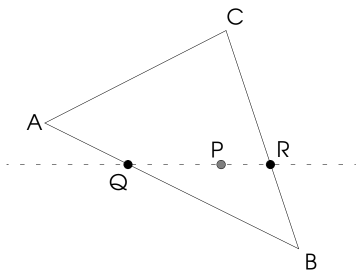
- ❏ the human visual system is extremely good at detecting edges - even when they are not there!
- ❏ abrupt changes in the shading of two adjacent polygons are perceived to be even greater

■ We need to smooth out these changes

Intensity Interpolation

This smooth shading method is also known as *Gouraud shading*

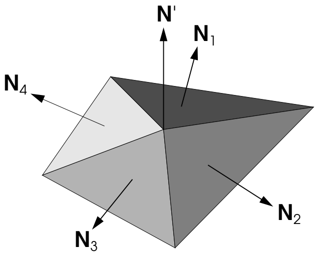
■ Given a polygon and a scan-line, the problem is to determine the intensity at an interior point, such as *P*



- ❏ for this we need the intensity values at the vertices *A*, *B* and *C*

Intensity Interpolation

- First compute the intensity values at each polygon vertex
  - ❏ need vertex normals
  - ❏ compute vertex normal approximately by averaging the surface normals of surrounding polygons



Intensity Interpolation

- Next compute the intensities at points *Q* and *R* where the scan-line and polygon intersect
  - ❏ use linear interpolation

$$I_Q = uI_B + (1-u)\,I_A \quad \text{and} \quad u = \frac{AQ}{AB}$$

$$I_R = wI_B + (1-w)\,I_C \quad \text{and} \quad w = \frac{CR}{CB}$$

- Finally, linearly interpolate between I_Q and I_R to get intensity at point *P*

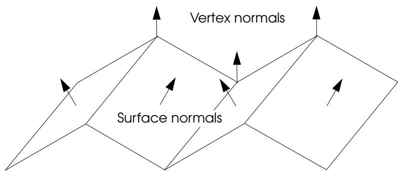
$$I_P = vI_R + (1-v)\,I_Q \quad \text{and} \quad v = \frac{QP}{QR}$$

- This method avoids shading discontinuity for adjacent polygons

Intensity Interpolation

Problems with Gouraud shading

- Smooths out real edges
  - ❏ compute two vertex normals, one for each side of the boundary
- May lose specular reflection if the highlight lies inside a single polygon
- Regular corrugated surfaces will appear to be shaded uniformly



Normal Vector Interpolation

This smooth shading technique is also known as *Phong shading*

- Similar to intensity interpolation, except we linearly interpolate the **surface normal vector** across the polygon
  - ❏ for the example situation described before we have

$$\mathbf{N}_Q = u\mathbf{N}_B + (1-u)\,\mathbf{N}_A$$
$$\mathbf{N}_R = w\mathbf{N}_B + (1-w)\,\mathbf{N}_C$$
$$\mathbf{N}_P = v\mathbf{N}_R + (1-v)\,\mathbf{N}_Q$$

- Now apply illumination model with interpolated normal to find intensity at required point

Normal Vector Interpolation

Pros and cons

- More realistic shading than intensity interpolation
  - specular highlights are preserved
  - Mach banding greatly reduced
- More computationally expensive
  - each interpolation requires complete shading calculation

Texture and Transparency

Texture

We now wish to model surfaces which are *not smooth* or *simply coloured.* There are two kinds of surface texture

- **Patterns** or **colour detail**
  - we superimpose a pattern on a smooth surface but the surface remains smooth
  - also known as *texture mapping*
- **Roughness**
  - we alter the uniformity of the surface using a *perturbation function* that effectively changes the geometry of the surface
  - for example, *bump mapping*
  - but also, *microfacet modelling*

Texture Mapping

Object space mapping

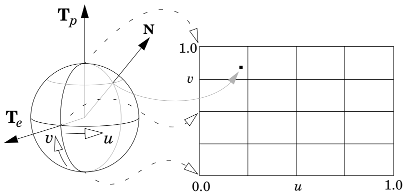
- We map an image onto the surface of an object
  - pattern space specified using (u, v)
  - object space by (θ, ϕ)
  - need to determine the mapping function defined by

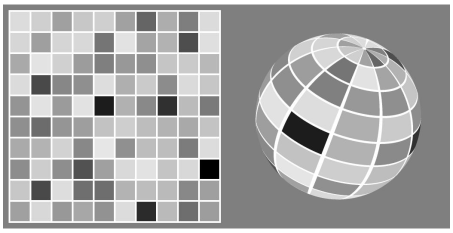
$$u = f(\theta, \phi) \qquad v = g(\theta, \phi)$$

- Hence, given a particular (θ, ϕ)
  - compute the corresponding (u, v)
  - shade the surface with the colour pointed to in the image by (u, v)

Texture Mapping

Mapping function for a sphere





Texture Mapping

There are other ways of mapping patterns

- **Parametric space mapping**
  - ❏ used when surface is defined *parametrically* (*s*, *t*)
  - ❏ similar to object space mapping
  - ❏ map (θ, ϕ) to (*s*, *t*)
- **Environment mapping**
  - ❏ used to simulate mirror-like reflections of the scene surrounding object

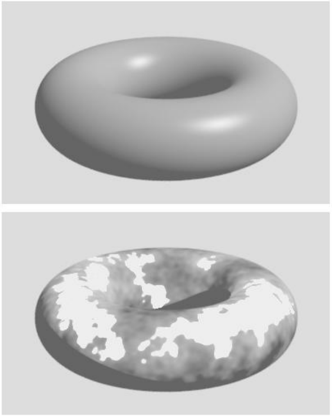
Bump Mapping

This method models features due to large-scale surface roughness

- Alters the uniformity of a surface using a *perturbation function*
  - ❏ for example, use a function which perturbs the normal vector
  - ❏ use new normal in illumination model
- Perturbation function can be defined
  - ❏ analytically
  - ❏ as a lookup table
- Different effects can be achieved
  - ❏ random function gives rough surface
  - ❏ a smoother function gives more regular feature

Bump Mapping

- An example of bump mapping



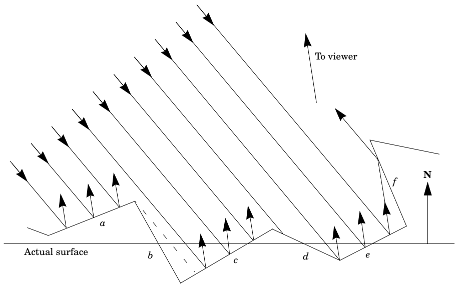
Bump Mapping

- This technique avoids explicitly modelling the geometry of the new surface
  - the *silhouette* is still smooth
  - roughening only becomes apparent when the shading model is applied
- Examples of use
  - texture of an orange
  - tread of tyre
  - etc

Cook-Torrance Model

This method models features due to small-scale roughness

- Surface modelled as collection of randomly oriented microscopic **facets**



Cook-Torrance Model

- Model accounts for three situations
  - facets which reflect light directly towards the viewer
  - facets which are in *shadow* of other facets
  - facets which reflect light which itself has been reflected from other facets - these multiple reflections contribute to the *diffuse reflection* from the surface
- Specular reflection coefficient of the overall surface given by

$$k_s = \frac{DGF}{\pi(\mathbf{N} \bullet \mathbf{V})(\mathbf{N} \bullet \mathbf{L})}$$

  - *D* is *distribution function* (Gaussian)
  - *G* is factor accounting for shadowing
  - *F* is the *Fresnel factor*

Cook-Torrance Model

- The Fresnel factor gives the fraction of light incident on a facet which is reflected rather than absorbed.
  - defined by

$$F = \frac{1}{2}\left[\frac{\sin^2(\phi - \theta)}{\sin^2(\phi + \theta)} + \frac{\tan^2(\phi - \theta)}{\tan^2(\phi + \theta)}\right]$$

  - θ and ϕ are the angles of incidence and reflection measured from the *facet normal* not the overall surface normal **N**
- Cook-Torrance model gives results similar to Phong´s except
  - for grazing angles of reflection
  - the highlight colour is not the same as light source
  - Phong is computationally less expensive

Transparency

Not all materials are opaque. Some objects allow light to be transmitted or refracted

- **Diffuse refraction**
  - transmitted light is scattered by internal and surface irregularities
  - surface appears *translucent* (frosted glass)
  - objects view through a diffuse refractor appear blurred
- **Specular refraction**
  - occurs in truly transparent materials
  - the direction of light rays are *bent* (lens)
  - objects viewed through a specular refractor are clearly seen

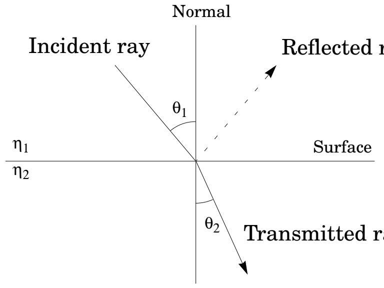
Transparency

Snell's Law

- Describes precisely how light behaves when moving from one medium to another

$$\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$$

  - η is the **refractive index** of the material
  - θ is angle of ray measured from normal



Transparency

Refractive index

- The refractive index is in general wavelength dependent
  - different colours will be bent by different amounts
  - this is called **dispersion**
  - we will ignore this effect since it is difficult to model and use an average value across the visible spectrum
- Snell's law is *significant*
  - example, light passing from air into heavy glass ($\eta = 1.5$) at $\theta = 30°$, will be bent by 11°°

Modelling Transparency

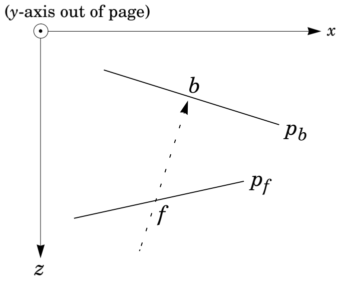
Non-refractive transparency

- Light paths are not bent
  - avoids computational overhead with trigonometric functions in Snell's law
- Then transparent objects will appear invisible!
- Introduce a **transmission coefficient** *t*, to measure transparency
  - opaque object has $t = 1$
  - perfectly transparent object has $t = 0$
  - *t* could be colour dependent (coloured glass, for example)

Modelling Transparency

Non-refractive transparency

- Example use



- The observed intensity is given by

$$I = tI_f + (1 - t)\, I_b$$

Global Illumination Models

Ray Tracing

This method simulates the global illumination distributed by specular light

- Uses the laws of **geometric optics**
  - ❑ follows light energy along *rays*
  - ❑ is a *recursive* procedure
  - ❑ is *view dependent*
- Includes an element to simulate **diffuse effects**
  - ❑ diffuse reflection from local illumination
  - ❑ otherwise surface will be black 'n' shiny
- Ray-tracing can be used on different kinds of surfaces, not just polygons
  - ❑ for example, spheres and cones

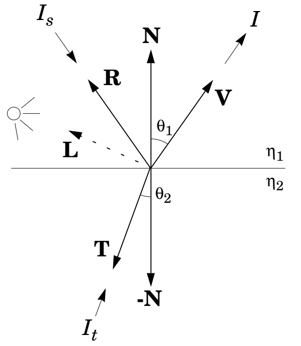
Ray Tracing

Overview of the procedure

- Rays are traced **from the view-point**
  - ❑ need to set-up *viewing system*
  - ❑ view-plane is discretized and mapped to image pixels
  - ❑ one *primary-ray* is traced through each pixel out into the scene
- Find *intersection* of primary-ray with object which is *nearest* view-point
  - ❑ object thus found represents surface visible through pixel
- At this point we have effectively performed a *hidden-surface removal* operation

Ray Tracing

Now determine the intensity of light leaving the surface intersection point



- This intensity will be due to a *local component* plus components due to *globally reflected* and *transmitted* light

Ray Tracing

- For a single point light source

$$I = k_d I_a + I_p \left[k_d (\mathbf{N} \bullet \mathbf{L}) + k_s (\mathbf{R} \bullet \mathbf{V})^n \right] + k_r I_r + k_t I_t$$

  - ❏ k_r is the **global specular reflection coefficient** (usually equal to k_s)
  - ❏ k_t is the **global specular transmission coefficient**
  - ❏ I_r *and* I_t are the intensities coming from directions **R** and **T**

- The global terms are calculated by spawning *secondary-rays* from intersection point in directions **R** and **T**
  - ❏ find intersection of secondary-rays with objects in the scene
  - ❏ apply intensity calculations again at new intersections

Ray Tracing

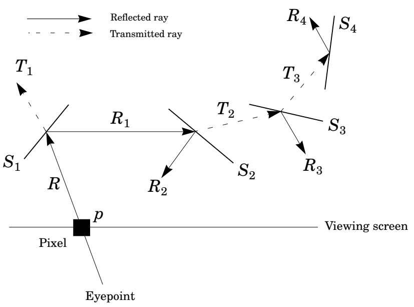
Recursive nature of ray-tracing

- We can repeat this process again and again
  - ❏ create new secondary-rays at each intersection point
  - ❏ use new rays to estimate I_r and I_t for each previous trace
- If k_r is zero, no global reflection occurs, surface is diffuse, so no need to spawn new specular rays
- Similarly for k_t, opaque surfaces have zero transmittance

Ray Tracing

Ray-tracing a particular pixel

- An schematic example



  - ❏ all surfaces are transparent except S_4
  - ❏ rays T_1, R_2, R_3, R_4 do not contribute to intensity at pixel

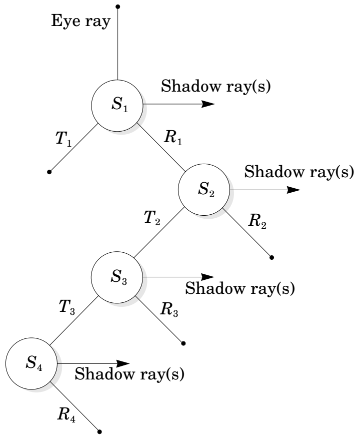
Ray Tracing

Intensity calculations at each point of intersection must take into account *shadows*

- First construct the **shadow ray**
  - origin at intersection point and direction towards the light source
- Test shadow ray against objects in scene
  - if hit is found *and* intersection is *nearer* than the light source, then point is in shadow
  - shadowed point does not contribute local illumination, except ambient
- Multiple light sources
  - shadow ray for each source

Ray Tracing

We can visualise the recursive process used in ray-tracing with a *ray-tree*



Ray Tracing

The ray-tree

- In practice, need to set-up maximum depth to which rays are traced
  - otherwise spend too much time on rays which contribute little to image
  - but if we have insufficient depth, will cause artifacts
- Three ways to control ray-tree depth
  - rays may leave scene
- return 'background' intensity
  - set absolute ray-tree depth
  - use *adaptive* depth control
- set threshold intensity returned by secondary-rays

Object Intersections

A major part of the ray-tracing algorithm is concerned with finding intersections with objects in the scene

- Each newly created ray must be tested against every object surface
  - if intersections are found, which one is the nearest?
- Need efficient intersection algorithms for all types of object
  - sphere, polygon, cone, box, cylinder, torus, etc
  - will illustrate how to calculate intersections with a sphere

Sphere Intersection

Vector equation of a ray

- Arbitrary ray defined *parametrically* as

$$\mathbf{r} = \mathbf{O} + \mathbf{D}t$$

  - **r** is position vector of point with parameter t
  - **O** is position vector of ray origin
  - **D** is *unit vector* in ray direction
- Parameter t is real-valued
  - represents a 'distance' along ray
  - for all primary rays the origin lies at the view point
- We require values of t at intersection points which are *positive*

Sphere Intersection

- Vector equation of a sphere

$$(\mathbf{r} - \mathbf{C}) \bullet (\mathbf{r} - \mathbf{C}) = R^2$$

  - **C** is position vector of sphere centre
  - R is the sphere's radius
- Substitute ray equation and solve for the parameter t, giving

$$t^2 - 2t\mu + \left[\lambda - R^2\right] = 0$$

$$\mu = \mathbf{D} \bullet \mathbf{T}$$

$$\lambda = \mathbf{T} \bullet \mathbf{T}$$

$$\mathbf{T} = \mathbf{C} - \mathbf{O}$$

- This is a quadratic in t, so

$$t = \mu \pm \sqrt{\gamma}$$

$$\gamma = \mu^2 - \lambda + R^2$$

Sphere Intersection

- Depending on the value of γ, we have one of three situations
  - $\gamma < 0$ ray misses sphere
  - $\gamma = 0$ one intersection which grazes sphere
  - $\gamma > 0$ two distinct intersections

$$t_1 = \mu + \sqrt{\gamma}$$

$$t_2 = \mu - \sqrt{\gamma}$$

- To find intersection points
  - substitute the t values back into the ray-equation

Optimisations

Naive ray-tracing can be very time consuming

- Typically 90% of computation time spent on object intersections
  - gets worse the greater number of objects in the scene
  - also get worse if we want higher resolution image (more primary-rays)
  - this slows the ray tracing method considerably
- Schemes are available to help which
  - limit the number of intersection tests
  - delay a costly test until a cheaper test has confirmed its necessity

Hierarchical Bounding Volumes

What is a bounding volume?

- Is a simple primitive which has *smallest* volume enclosing object
  - ❏ usually spheres and axis-aligned boxes
- During ray intersection test
  - ❏ first test the bounding volume
    - usually easier and faster
  - ❏ if bounding volume intersected then test the actual object
- Spheres are very popular
  - ❏ very efficient since we only need to know if intersection takes place, NOT where the intersection points are

Hierarchical Bounding Volumes

Hierarchical scheme?

- Clusters of bounding volumes within larger bounding volume
  - ❏ intersect ray with outer volume
  - ❏ then with inner volumes if necessary
- Can have any number of levels of bounding volumes
- Hierarchical scheme efficient for scenes with non-uniform distribution of objects
  - ❏ ray doesn't do much work in areas which it is not 'looking' at
- Very good speed-up in rendering time

3D Spatial Subdivision

More powerful partitioning schemes use the idea of *voxels*

- **Voxels** are axis-aligned rectangular prisms which are like bounding volumes except...
  - ❏ fill all of space occupied by scene
  - ❏ are non-overlapping
  - ❏ do not necessarily completely enclose any particular object
- Two varieties of spatial subdivision
  - ❏ **uniform**
    - all voxels are the same size
    - stacked together
  - ❏ **non-uniform**
    - octrees
    - voxel hierarchy

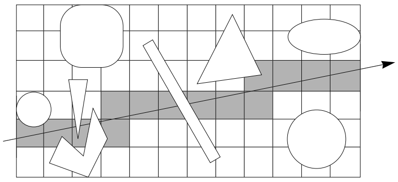
3D Spatial Subdivision

Uniform subdivision - *Voxel Grids*

- Constructing the voxel grid
  - ❏ surround the scene with a bounding cuboid
  - ❏ split cuboid into L x M x N smaller cuboids - these are the voxels
  - ❏ for each voxel keep a list of objects which encroach into its space
- Tracing a ray through the voxel grid
  - ❏ determine which voxels the ray passes through
  - ❏ only perform intersection tests on those objects which are in the voxel list
  - ❏ next, an illustration in 2D

3D Spatial Subdivision

Tracing a ray through the voxel grid



- We wish to find the first intersection
  - follow the ray voxel-by-voxel
  - if object is part of two adjacent voxels, keep results of intersection test
  - if object is hit, but intersection point is *outside* current voxel, then continue
  - only if intersection point occurs *within* the current voxel can we stop

3D Spatial Subdivision

What have we gained?

- Ray intersects objects further *down-stream* but didn't need to be tested
  - this would have been the case otherwise
  - only three voxels are considered before a valid intersection is found
  - 5 out of 8 possible intersection tests are avoided
- In practice, 3D scenes could contain 1000's of objects
  - gains are much more substantial
  - with spatial subdivision, we restrict intersection testing to objects in the *neighbourhood* of the ray's trajectory

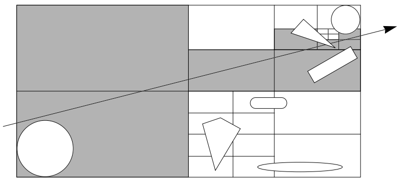
3D Spatial Subdivision

Non-uniform subdivision - *Octrees*

- Hierarchical tree of non-overlapping voxels of various sizes
  - emphasises the spatial distribution of objects in a scene
- Constructing the octree
  - surround scene with bounding cuboid
  - divide into *eight* equal-sized sub-volumes or voxels
  - keep list of objects associated with each sub-volume
  - if maximum number of objects/voxel is above threshold, then subdivide again
- Subdivision only occurs where there are lots of objects

3D Spatial Subdivision

Tracing a ray through an octree



- Procedure similar to uniform case
  - *six* voxels are considered before valid intersection is found
  - only *three* distinct intersection tests are actually made
  - ray passes through large, empty regions of the scene very quickly
  - useful work done only in high density regions

3D Spatial Subdivision

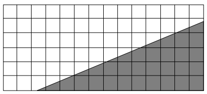
Controlling subdivision

- Trade off between voxel resolution and time spent traversing voxels
  - ❏ voxel resolution controlled by maximum number of objects/voxel
- These can be adjusted for optimum efficiency
- An example of what can be achieved
  - ❏ a uniform subdivision of a scene containing over 10,000 objects took 15 minutes to render
  - ❏ the same scene without voxelisation took 40 *days* to render!

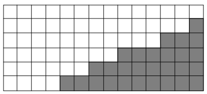
Image Aliasing

What is aliasing?

- Eye-rays passing through image plane **samples** the light distribution
  - ❏ discrete representation is only approximate
  - ❏ if we try to *reconstruct* the light distribution, distortions occur
  - ❏ this is **aliasing**



reality

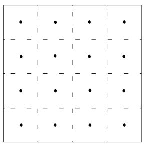


image

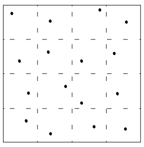
Image Aliasing

Anti-aliasing techniques

- Anti-aliasing attempts to reduce the effects of aliasing
  - ❏ usual way is to sample each pixel more than once - **supersampling**
  - ❏ two ways to supersample



Uniform



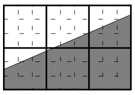
Jittered

  - ❏ **uniform** sampling passes eye-rays through set of *sub-pixels*
  - ❏ **jittered** sampling randomly displaces the uniform sampling array

Image Aliasing

Reconstruction

- We must combine the intensity information gathered by the supersampled rays
  - ❏ this is called **filtering**
- Many ways of filtering, for example
  - ❏ **box** - average the intensities
  - ❏ **gaussian** - average weighted by distance from centre of pixel



Pixel values

0	1/9	5/9
7/9	1	1

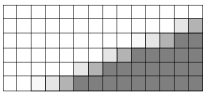


Image Aliasing

Adaptive supersampling

- Supersampling every pixel is generally not necessary
  - ❏ aliasing only noticeable along edges or other *high contrast* boundaries
  - ❏ adaptive schemes attempt to locate pixels which need special attention
- This is one way of doing this
  - ❏ first shoot rays through corners of pixel

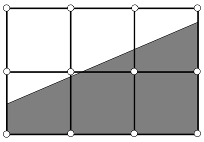
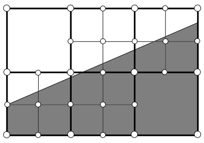


Image Aliasing

- ❏ for each pixel examine variation of these 4 samples
- ❏ if contrast too large (threshold user defined) split into 4 again



- ❏ repeat as required

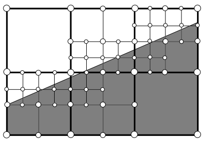


Image Aliasing

Adaptive supersampling

- **Contrast parameter**

$$C = \frac{I_{max} - I_{min}}{I_{max} + I_{min}}$$

  - ❏ I_{max} and I_{min} largest and smallest sample intensity
  - ❏ required for each colour band (R, G. B)
  - ❏ supersampling performed if any value exceeds pre-set values
  - ❏ for example, 0.25, 0.20, 0.40
- Adaptive supersampling is *very* efficient
  - ❏ many samples can be reused
  - ❏ in example, only 62 out of a possible total of 117 eye-rays are traced

Radiosity

This method simulates the global illumination distributed by diffuse light

- Attempts to accurately model the effects of ambient light
  - ❏ before we just had $I_a k_d$ with I_a constant
- Uses the laws of *energy conservation*
  - ❏ scenes are usually *closed* environments
- We assume all surfaces are purely diffuse reflectors
- Radiosity is a *view-independent* technique
  - ❏ we calculate the diffuse light leaving all surfaces
  - ❏ then render from any particular view point

Radiosity

The Radiosity Equation

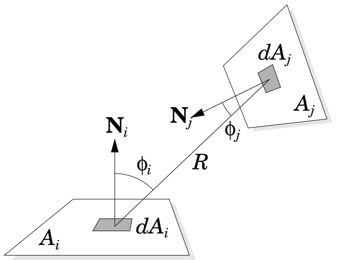
- Assume scene is comprised of discrete patches
- Energy equilibrium gives for each patch

$$A_iB_i = A_iE_i + \rho_i \sum_{j=1}^{n} B_jF_{ji}A_j$$

  - ❑ B_i is the **radiosity** of the patch
    - energy per unit time per unit area
  - ❑ E_i is the energy emitted (same units as B)
  - ❑ F_{ji} is the **form-factor**
    - energy leaving patch j which directly reaches patch i
  - ❑ ρ_i is the **diffuse reflectivity**(like k_d)
  - ❑ A_i is the **area** of the patch

Radiosity

Form-factors



- Assuming radiosity is constant across patch

$$F_{ji} = \frac{1}{A_j}\int_{A_j}\int_{A_i}\frac{\cos\phi_i\cos\phi_j}{\pi R^2}H_{ji}dA_idA_j$$

  - ❑ H_{ji} is a visibility factor equal to 1 if dA_j can see dA_i, otherwise 0

Radiosity

Form-factors

- **Reciprocity relation** for form-factors

$$A_iF_{ij} = A_jF_{ji}$$

  - ❑ applies to enclosed environment
  - ❑ useful for calculating form-factors
    - we can get F_{ij} easily once F_{ji} is known
- Also, for an enclosed environment

$$\sum_{j=1}^{n} F_{ij} = 1$$

  - ❑ follows from energy conservation
  - ❑ useful as a test of form-factor accuracy

Radiosity

Solving the equation

- With the reciprocity relation we can write

$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_jF_{ij}$$

- This is a set of simultaneous equations in the unknown quantities B_i
  - ❑ for example, a 3-patch scene

$$B_1 = E_1 + \rho_1 (F_{11}B_1 + F_{12}B_2 + F_{13}B_3)$$
$$B_2 = E_2 + \rho_2 (F_{21}B_1 + F_{22}B_2 + F_{23}B_3)$$
$$B_3 = E_3 + \rho_3 (F_{31}B_1 + F_{32}B_2 + F_{33}B_3)$$

  - ❑ we calculate the F_{ij} and we know the E_i hence we can determine the radiosities of each patch

Computing Form-Factors

There are no known analytical solutions to the form-factor integral equation

$$F_{ji} = \frac{1}{A_j}\int_{A_j}\int_{A_i}\frac{\cos\phi_i\cos\phi_j}{\pi R^2}H_{ji}dA_i dA_j$$

- We need a numerical technique
  - ❏ but double integrals are still tough
- Assume the distance between patches is large compared to their size
  - ❏ inner integral is approximately constant

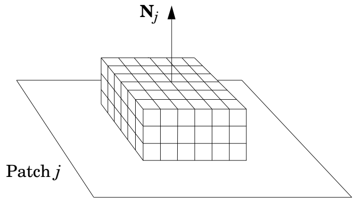
$$F_{ji} = \int_{A_i}\frac{\cos\phi_i\cos\phi_j}{\pi R^2}H_{ji}dA_i$$

  - ❏ form-factor from dA_j to A_i

Computing Form-Factors

We evaluate the simplified form-factor integral using a *projection method*

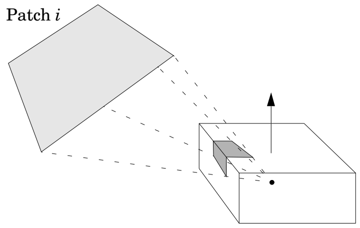
- The **hemi-cube algorithm** is one such method
  - ❏ half a cube of side 2 is centred about a patch *j*
  - ❏ each face is discretised uniformly into a number of pixels (user controllable)
  - ❏ commonly 50x50 or 100x100



Computing Form-Factors

The hemi-cube algorithm

- Next step is to project every other patch onto the surface of the hemi-cube...



- ... and determine which pixels are covered
  - ❏ if two patches project to the same pixel, the nearest one is stored
  - ❏ this accounts for the H_{ji} term

Computing Form-Factors

The hemi-cube algorithm

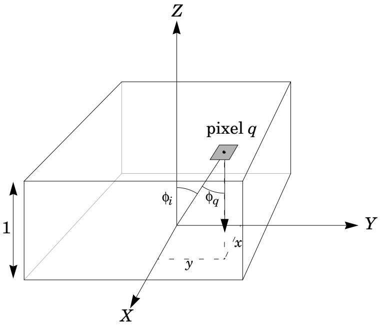
- Finally, we determine the form-factors by summing the **delta-form-factors** of the pixels which each patch projects to

$$F_{ji} = \sum_q \Delta F_q$$

  - ❏ delta-factors can be pre-calculated
- This gives us *n* form-factors relative to one patch
  - ❏ repeat operation with another hemi-cube centred about another patch
  - ❏ do this for all patches in scene

Computing Form-Factors

Delta-form-factors

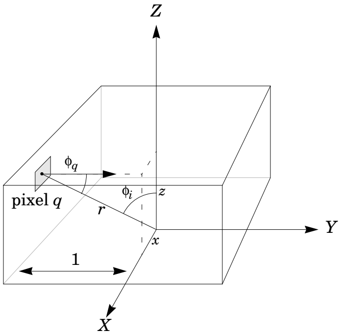


- For a pixel on the top-face

$$\Delta F_q = \frac{1}{\pi\left[x^2+y^2+1\right]^2}\Delta A$$

Computing Form-Factors

Delta-form-factors



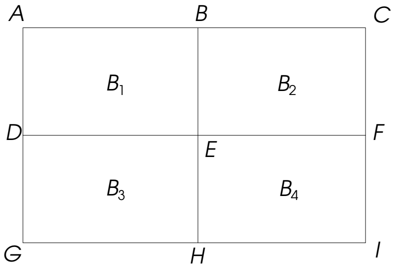
- For a pixel on a side-face

$$\Delta F_q = \frac{z}{\pi\left[x^2+z^2+1\right]^2}\Delta A$$

Rendering

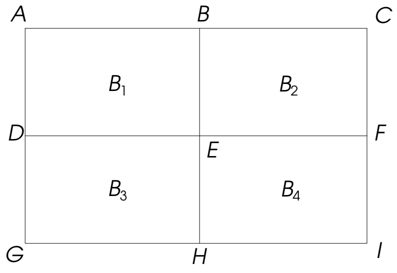
We now have *n* radiosity values, so how do we render a particular view?

- We want to smooth shade the surface
  - ❏ use Gouraud method
  - ❏ need radiosities at each patch vertex and then interpolate across the patch
  - ❏ For example



Extrapolation

- We **extrapolate** the radiosity values at the centre of the patches to the patch vertices
- There are 3 distinct cases:
  - ❏ internal vertices (e.g. vertex *E*)
  - ❏ edge vertices (e.g. vertex *B*)
  - ❏ corner vertices (e.g. vertex *A*)

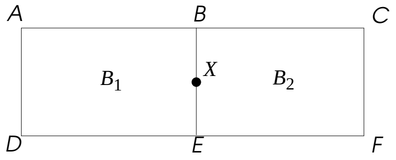


Extrapolation

- **Internal vertices**: average the radiosities of all the patches containing the vertex

$$B_E = (B_1 + B_2 + B_3 + B_4)/4$$

- **Edge vertices**: find nearest *internal* vertex and note that B_X can be expressed in two ways



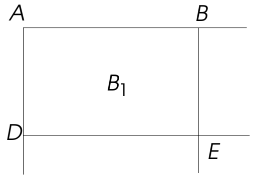
$$\frac{B_B + B_E}{2} = \frac{B_1 + B_2}{2}$$

  - hence

$$B_B = B_1 + B_2 - B_E$$

Extrapolation

- **Corner vertices**: find nearest *internal* vertex and note average of corner vertex and its internal vertex is the patch radiosity



  - for example

$$B_1 = (B_A + B_E)/2$$

  - hence

$$B_A = 2B_1 - B_E$$

Progressive Refinement Radiosity

So far we have covered the full-matrix radiosity method

- This requires
  - $O(n^2)$ storage for the form-factors
  - $O(n^2)$ time to solve radiosity equation
- **Progressive refinement** attempts to overcome these problems
  - gives $O(n)$ storage costs
  - gives $O(n)$ computation time costs *against some initial image*
- Progressive refinement combines image realism with *user interactivity*

Progressive Refinement

Formulation

- Recall the energy equilibrium equation...

$$A_iB_i = A_iE_i + \rho_i \sum_{j=1}^{n} B_jF_{ji}A_j$$

- ...and consider the interaction between two patches *i* and *j*

$$B_i(\text{due to } B_j) = \rho_i B_j F_{ji}\frac{A_j}{A_i}$$

- So applying a single hemi-cube at patch *j* we can find the contribution of this patch to the rest of the scene
  - but to be used in some *iterative* scheme, it's better to consider *changes* in radiosity

Progressive Refinement

Formulation

- Write this as

$$B_i(\text{due to } \Delta B_j) = \rho_i \Delta B_j F_{ji} \frac{A_j}{A_i}$$

  - ❏ ΔB_j is the *unshot* radiosity of patch j
    - due to its emission of light
    - or light received from other patches
- We use the progressive refinement method in the following way
- For each patch keep track of two radiosity values
  - ❏ current radiosity estimate, B_j
  - ❏ unshot radiosity, ΔB_j
  - ❏ initially both these are equal to E_j

Progressive Refinement

Formulation

- Choose patch with *largest* unshot energy
  - ❏ unshot energy is $\Delta B_j A_j$
- *Shoot* this energy to all other patches as described before
  - ❏ each patch will received a certain amount of energy which is added to both B_j and ΔB_j
- Set the unshot radiosity of the shooting patch to *zero*
- With new estimates of B_j, render an image
- Repeat this cycle again and again, rendering an image after each step

Progressive Refinement

Formulation

- After each step, all the ΔB_j will be *underestimates*
  - ❏ but each step reduces the relative size of the ΔB_j
  - ❏ hence the radiosity estimates, B_j, slowly *converge* to their full-matrix values
- The cycle is repeated until the *total unshot energy* in the whole scene falls below some predefined value

Progressive Refinement

Ambient contribution

- With progressive refinement, first few images will generally be *dark*
  - ❏ not all the energy has been distributed
  - ❏ only surfaces in the direct line of sight of the light source(s) are illuminated
- An **ambient radiosity term**, B_{amb}, is introduced *for display purposes only*
  - ❏ based upon how much unshot energy remains and how this could be distributed
  - ❏ when rendering, use $\rho_j B_{amb} + B_j$, instead of B_j
  - ❏ this value plays no part in subsequent refinements of B_j

Progressive Refinement

Calculating the ambient term

- First estimate the form-factors

$$F_{ji}^{est} = \frac{A_i}{\sum_{k=1}^{n} A_k}$$

- Next, determine average reflectance of the scene

$$\rho_{av} = \frac{\sum_{j=1}^{n} \rho_j A_j}{\sum_{j=1}^{n} A_j}$$

  - ❏ from this we can write the overall inter-reflection coefficient as

$$1 + \rho_{av} + \rho_{av}^2 + \rho_{av}^3 + \ldots = \frac{1}{1 - \rho_{av}}$$

Progressive Refinement

Calculating the ambient term

- Then ambient radiosity term is given by

$$B_{amb} = \frac{1}{1 - \rho_{av}} \sum_{i=1}^{n} \Delta B_i F_{ji}^{est}$$

- Note how the ambient term gracefully diminishes as the refinement progresses
  - ❏ $B_{amb} \rightarrow 0$ as $\Delta B_i \rightarrow 0$

These materials have been produced as part of the Information Technology Training Initiative, funded by the Information Systems Committee of the Higher Education Funding Councils.

For further information on this and other modules please contact:

The ITTI Gravigs Project, Computer Graphics Unit, Manchester Computing.
Tel: 0161 275 6095 Email: *gravigs@mcc.ac.uk*

To order additional copies of this or other modules please contact:

Mrs Jean Burgan, UCoSDA.
Tel:0114 272 5248 Email: *j.burgan@sheffield.ac.uk*

ISBN 1 85889 064 0