Interactive Computer Graphics

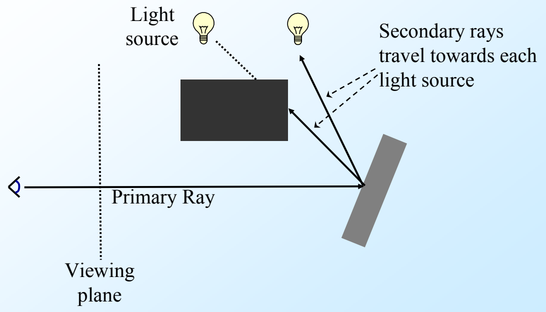
Lecture 10:

Ray Tracing and Constructive Solid Geometry

Ray tracing with secondary rays

- Ray tracing using just primary rays produces images similar to normal polygon rendering techniques
- Recursive ray tracing, with secondary rays produces more realistic images by adding
  - shadows
  - reflections
  - transparency

Ray tracing to find shadows



Ray tracing: Shadows

- Illumination model with shadows:

$$I = k_a I_a + \sum_i s_i I_i \left[k_d(\mathbf{n}\cdot\mathbf{l}_i) + k_s(\mathbf{v}\cdot\mathbf{l}_i')^t\right]$$

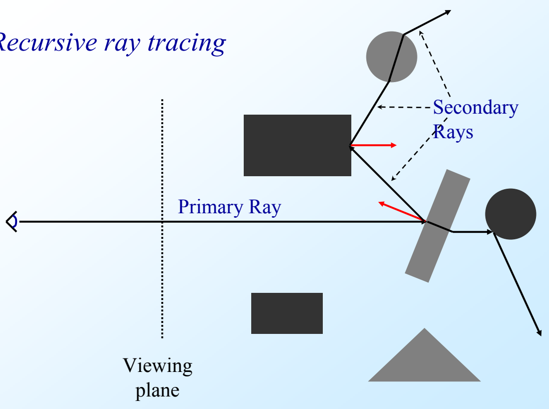
- The sum is taken over each light source
- s_i is a delta function:

$$s_i = \begin{cases} 0 & \text{if light source is obscured} \\ 1 & \text{if light source is not obscured} \end{cases}$$

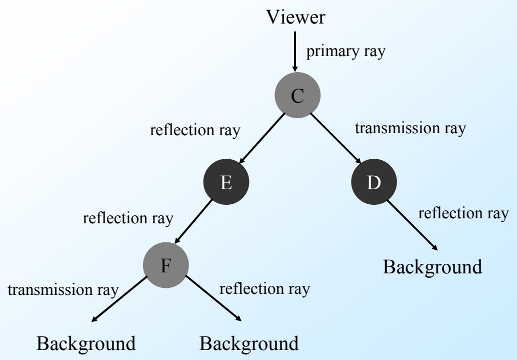
Ray tracing: Reflection and Refraction

1. Trace primary ray to determine nearest intersection
2. Cast new ray in the direction of reflection or refraction
3. Trace secondary rays like primary rays and repeat
4. Stop recursion if
   - ray hits light source
   - ray hits background
   - maximum recursion depth is reached

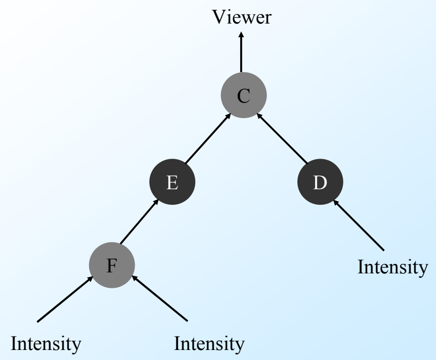
Recursive ray tracing



Recursive ray tracing tree



Collecting the illumination



Recursive Ray tracing

- If no object intersects the ray, the ray tracing tree will be empty and the pixel will be assigned the value of the background.
- Intensities are accumulated starting from leaf nodes upwards to the root node
- Intensity from each node in the tree is attenuated by the distance from the parent node and added to the intensity of the parent node

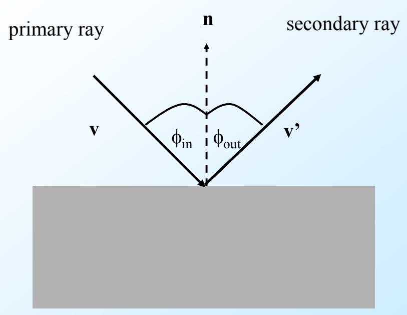
Ray tracing: Reflections and transparency

- Illumination model with shadows, reflections and transparency

$$I = k_a I_a + \sum_i s_i I_i \left[k_d (\mathbf{n} \cdot \mathbf{l}_i) + k_s (\mathbf{v} \cdot \mathbf{l}_i')^t \right] + k_r I_r + k_t I_t$$

- k_r is the reflection coefficient of the reflected ray
- I_r is the intensity of the reflected ray
- k_t is the transmission coefficient of the transmitted ray
- I_t is the intensity of the transmitted ray

Reflections



Reflections

- To calculate illumination as a result of reflections
  - calculate the direction of the secondary ray at the intersection of the primary ray with the object.
  - assume that
    - n is the surface normal
    - v is the direction of the primary ray
    - v' is the direction of the secondary ray as a result of reflections

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

Reflections

The **v**, **v’** and **n** are unit and coplanar so:

$$\mathbf{v}' = \alpha\,\mathbf{v} + \beta\,\mathbf{n}$$

Taking the dot product with **n** yields the eq.:

$$\mathbf{n}\cdot\mathbf{v}' = \alpha\,\mathbf{v}\cdot\mathbf{n} + \beta = \mathbf{v}\cdot\mathbf{n}$$

Requiring v’ to be a unit vector yields the second eq.:

$$1 = \mathbf{v}'\cdot\mathbf{v}' = \alpha^2 + 2\,\alpha\,\beta\,\mathbf{v}\cdot\mathbf{n} + \beta^2$$

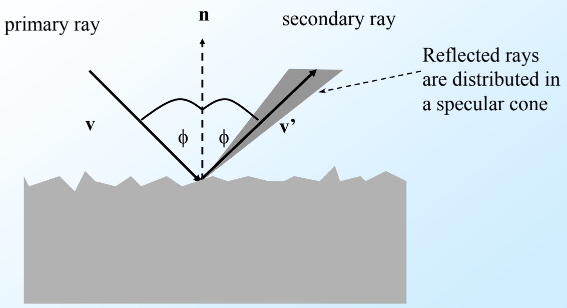
- Solving both equation yields:

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

Reflection

- Perfect reflection implies that the ray is only refracted in one direction.
- Perfect reflection is a good approximation for
  - mirror
  - smooth or polished surfaces
- but perfect reflection is a not good approximation for
  - rough surfaces
  - uneven surface
- Reflection light can be modeled as a large number of rays scattered around the principal direction of reflection
  - using a large number of rays is computationally impossible
  - using *n* rays randomly distributed allows the creation of realistic effects

Reflections

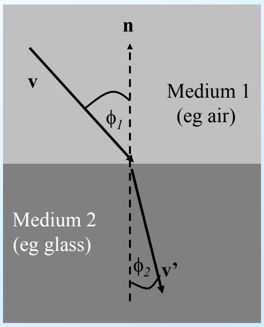


Translucent Objects

- The angle of the refracted ray can be determined by Snell’s law:

$$k_1 \sin(\phi_1) = k_2 \sin(\phi_2)$$

- k_1 is a constant for medium 1
- k_2 is a constant for medium 2
- ϕ_1 is the angle between the incident ray and the surface normal
- ϕ_2 is the angle between the refracted ray and the surface normal



Refraction

- In vector notation Snell’s law can be written:

$$k_1(\mathbf{v}\times\mathbf{n}) = k_2(\mathbf{v}'\times\mathbf{n})$$

- The direction of the refracted ray is

$$\mathbf{v}' = \frac{k_1}{k_2}\left(\left[\sqrt{(\mathbf{n}\cdot\mathbf{v})^2 + \left(\frac{k_2}{k_1}\right)^2 - 1} - \mathbf{n}\cdot\mathbf{v}\right]\cdot\mathbf{n} + \mathbf{v}\right)$$

Refraction

- This equation only has a solution if

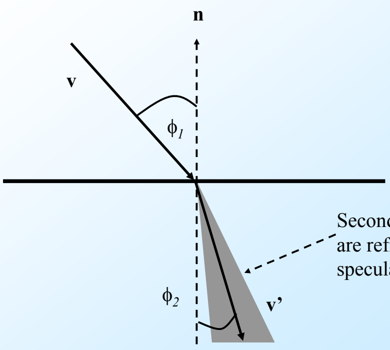
$$(\mathbf{n}\cdot\mathbf{v})^2 > 1 - \left(\frac{k_2}{k_1}\right)^2$$

- This illustrates the physical phenomenon of the limiting angle:
  - if light passes from one medium to another medium whose index of refraction is low, the angle of the refracted ray is greater than the angle of the incident ray
  - if the angle of the incident ray is large, the angle of the refracted ray is larger than 90°
  - ➨ the ray is reflected rather than refracted

Refraction

- Perfect refraction implies that the ray is only refracted in one direction.
- Perfect refraction is a good approximation for
  - clear glass
  - water
- but perfect refraction is a not good approximation for
  - frosted glass
  - dust
- Refraction light can be modeled as a large number of rays scattered around the principal direction of refraction
  - using a large number of rays is computationally impossible
  - using *n* rays randomly distributed allows the creation of realistic effects

Refraction

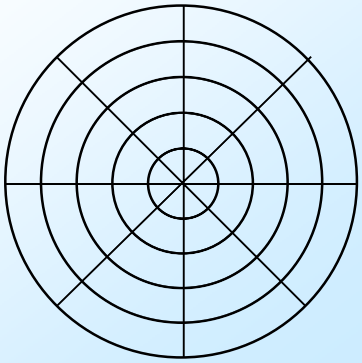


Secondary rays are refracted in a specular cone

Monte –Carlo Simulations

Monte Carlo simulations are used to make estimates of variables that cannot be computed in real time.

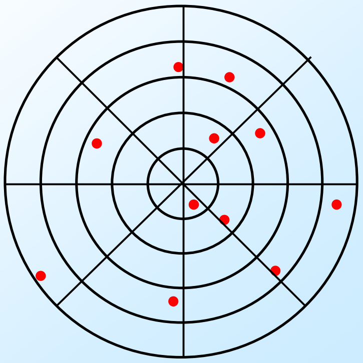
For example transmission of light through frosted glass or reflection from dull surfaces



Monte –Carlo Simulations

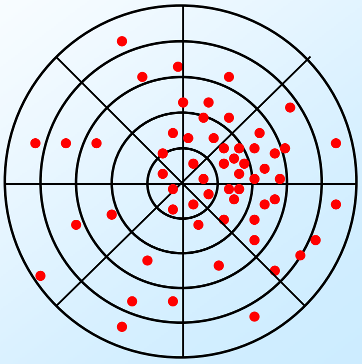
To determine the light arriving in a specular cone we make a random selection of rays in that cone and trace them recursively. The light intensity is set to the average of the secondary rays.

The rays would usually be normally distributed about the centre



Monte –Carlo Simulations

Clearly the more samples we take the better the simulation, but in a recursive ray tracing context computational demands will place a limit on this.



Constructive Solid Geometry (CSG)

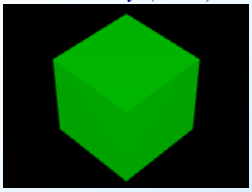
- Real and virtual objects can be represented by
  - solid models such as spheres, cylinders and cones
  - surface models such as triangles, quads and polygons
- Surface models can be rendered either by
  - object-order rendering (polygon rendering)
  - image-order rendering (ray tracing)
- Solid models can only be rendered by ray tracing
- Solid models are commonly used to describe man-made shapes
  - computer aided design
  - computer assisted manufacturing

Constructive Solid Geometry (CSG)

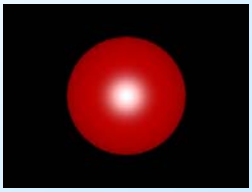
- CSG combines solid objects by using three (four) different boolean operations
  - intersection ($\cap$)
  - union (+)
  - minus (–)
  - (complement)
- In theory the minus operation can be replaced by a complement and intersection operation
- In practice the minus operation is often more intuitive as it corresponds to removing a solid volume

Constructive Solid Geometry (CSG)

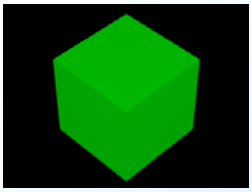
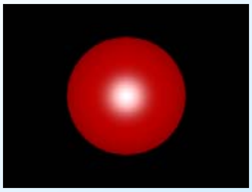
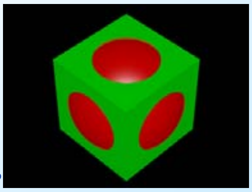
- Box



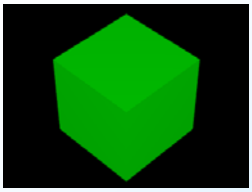
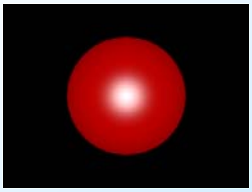
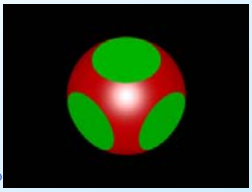
- Sphere



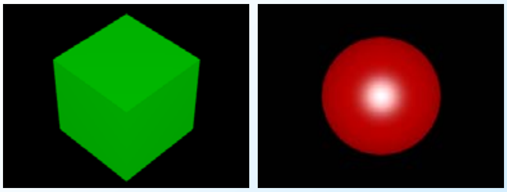
Constructive Solid Geometry (CSG): Union

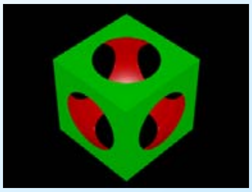




Constructive Solid Geometry (CSG): Intersection

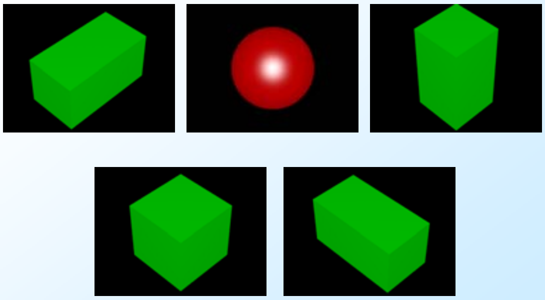




Constructive Solid Geometry (CSG): Minus

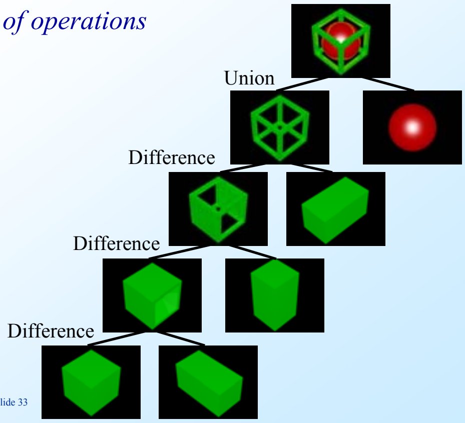




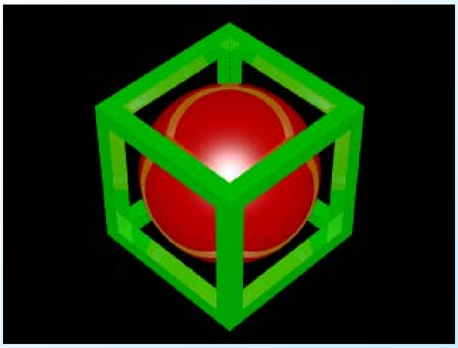
CSG trees: Primitive objects



CSG tree of operations

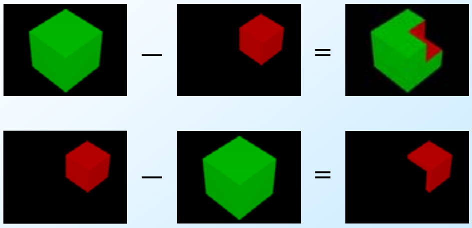


CSG Final Complex Object



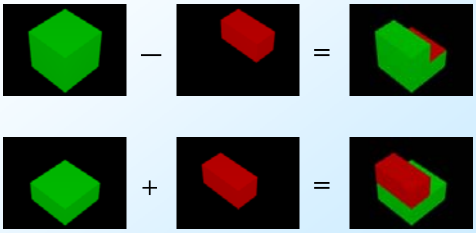
CSG trees

- CSG operations are not commutative:

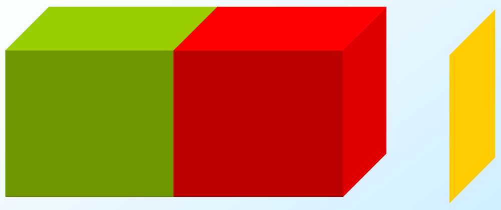


CSG trees

- CSG operations are not unique:



CSG trees: Degeneration Problems



Cube A ∩ Cube B = Plane

Ray tracing CSG trees

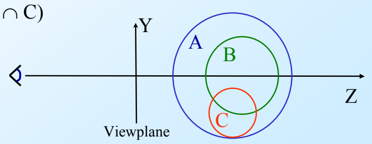
- CSG trees must be rendered by ray tracing
- CSG trees must be traversed in a depth first manner
- traversal starts at the leaf nodes
- traversal of each node yields a list of line segments of the ray that pass through the solid object
- the list of lines segments is passed to parent node and processed accordingly

Example: Ray tracing CSG trees

- Concrete example:
  - the viewpoint is at $\mathbf{p}_v = (0, 0, -10)$
  - the ray passes through viewing plane at $\mathbf{p}_i = (0, 0, 0)$.
- Spheres:
  - Sphere A with center $\mathbf{p}_s = (0, 0, 8)$ and radius r = 5
  - Sphere B with center $\mathbf{p}_s = (0, 0, 9)$ and radius r = 3
  - Sphere C with center $\mathbf{p}_s = (0, -3, 8)$ and radius r = 2

Ray tracing CSG trees

- Calculate the intersections of the ray for the following objects, assuming the spheres defined before.
  a. A + B + C
  b. A – B
  c. (B – A) ∩ C
  d. A + (B ∩ C)



Ray tracing CSG trees

- A + B + C: The ray enters the object at (0, 0, 3) and exits the object at (0, 0, 13)
- A – B: The ray enters the object at (0, 0, 3), leaves the object at (0, 0, 6), enters the object again at (0, 0, 12) and leaves the object at (0, 0, 13).
- (B – A) ∩ C: B – A produces an empty object list, the ray has therefore no intersections.
- A + (B ∩ C): The ray enters the object at (0, 0, 3) and exits the object at (0, 0, 13)

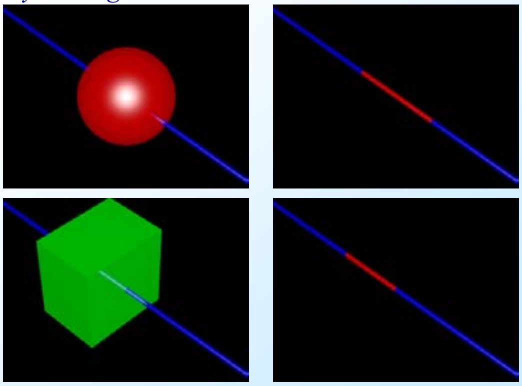
Ray tracing CSG trees

- The intersections of a ray and a CSG object tree may be characterized by a list of the μ values of the ray equation:

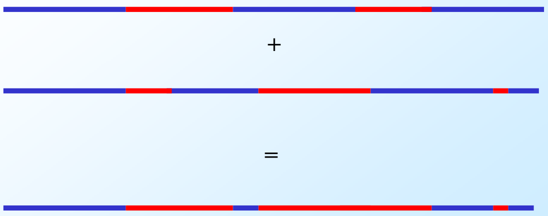
$$(\mu_1, \mu_2, \ldots, \mu_n)$$

- Each list of line segments will either contain
  - an odd number of intersection points (the viewpoint is inside the solid object)
  - an even number of intersection points (the viewpoint is outside the solid object)
  - an empty list of intersection points

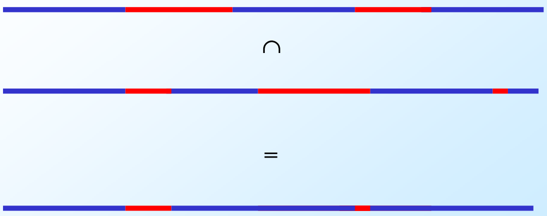
Ray tracing CSG trees



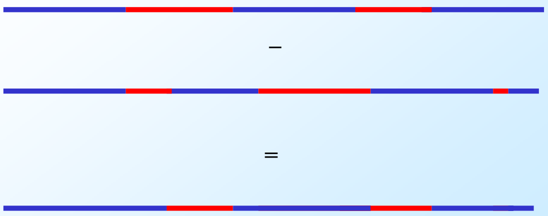
Ray tracing CSG trees: Union



Ray tracing CSG trees: Intersections



Ray tracing CSG trees: Minus

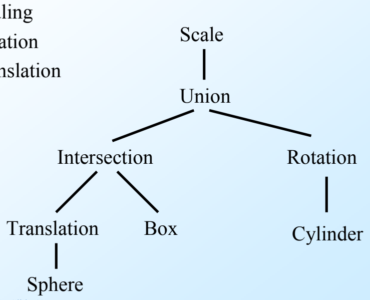


Ray tracing CSG trees

- CSG trees can be pruned during ray tracing:
  - if the left or right subtree of an intersection operation returns an empty list, then the other subtree need not be processed.
  - if the left subtree of a minus operation returns an empty list, then the right subtree need not be processed.
- CSG trees can use bounding boxes/spheres to speed up rendering:
  - each primitive that does not belong the currently processed bounding volume may be represented by an empty intersection list

Extending CSG trees

- Adding transformations as primitive operations:
  - scaling
  - rotation
  - translation


Scale
Union
Intersection
Rotation
Translation
Box
Cylinder
Sphere