Ray Tracing

Kadi Bouatouch
IRISA
Email: kadi@irisa.fr

IRISA

Introduction to Computer Graphics

IRISA

Context

- Representation and visualization of things which do not exist
  - Create 3D virtual worlds
    - Choose the model of the objects we want to represent
  - Visualize them
    - Choose a representation model
  - Animate these worlds

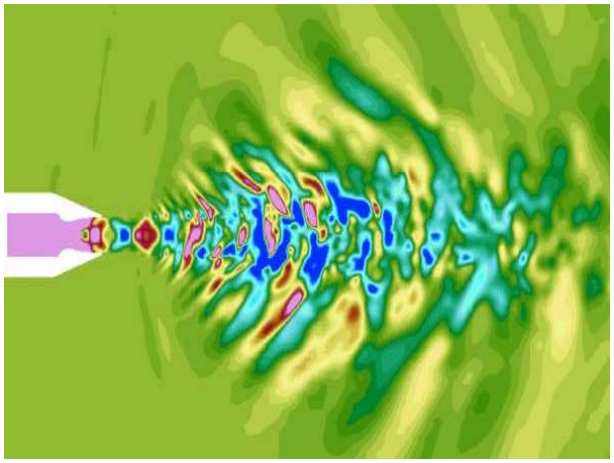
IRISA

Interest in 3D virtual worlds

- Computer aided design
- Manufacturing
- Movies
- Entertainement: video games
- Data Visualization
- Virtual Reality
- Working without risk
- Simulation

IRISA

Application

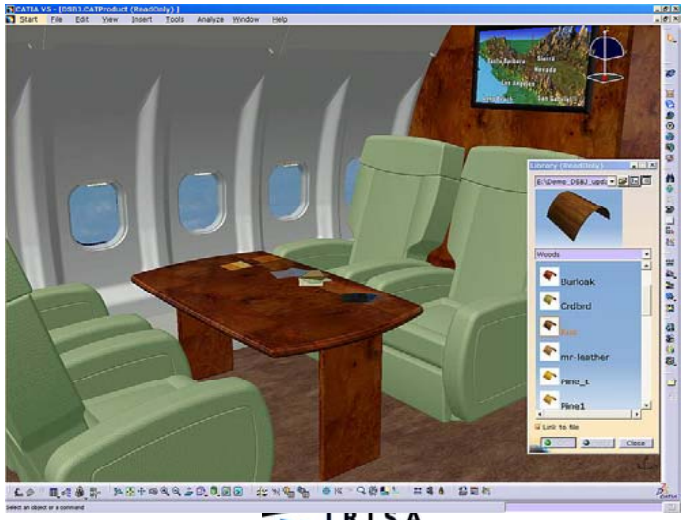


IRISA

Application

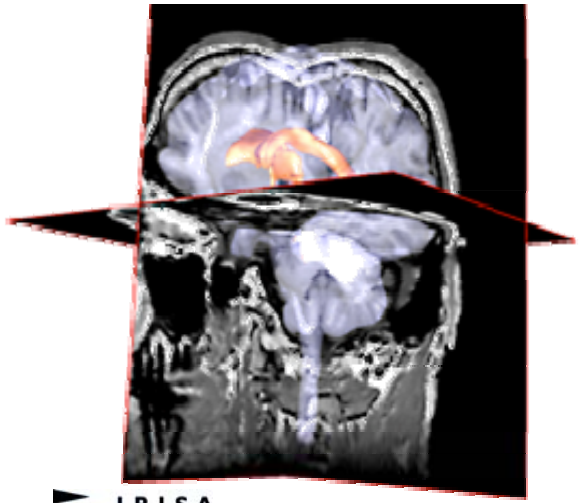
IRISA

Application



IRISA

Application



IRISA

Application

IRISA

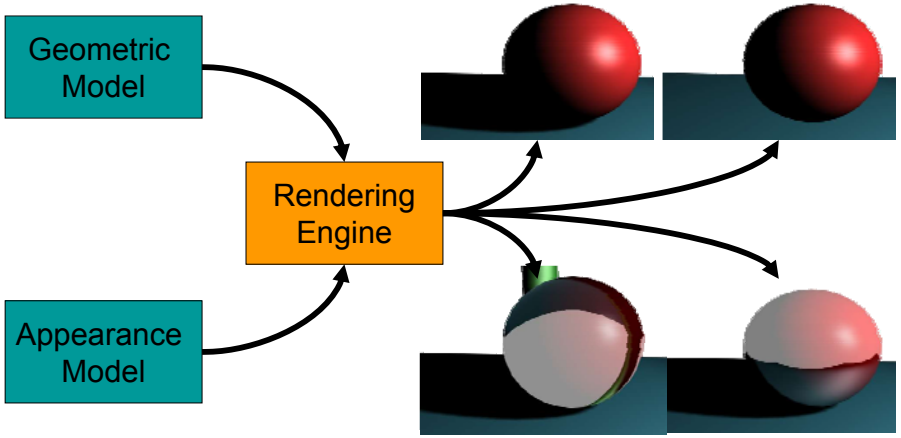
Application

IRISA

Application

IRISA

Rendering Engine



IRISA

Rendering Engine

- Real-time
  - Z-buffer
    - 10 images per second >> 0,1s
    - 25 images per second >> 0,04 s
    - 60 images per second >> 0,016 s
    - 120 images per second >> 0,008 s
- Non real-time
  - Ray Tracing, Path Tracing, Photon Mapping
  - Radiosity

IRISA

Illumination Model

- Expresses the light intensity at a point due to:
  - The light sources giving rise to :
    - diffuse and specular reflections
    - shadows
  - The reflectance and the transmittance of the object
- $I_r = I_d + I_s$
  - I_d : diffuse intensity
  - I_s : specular intensity

IRISA

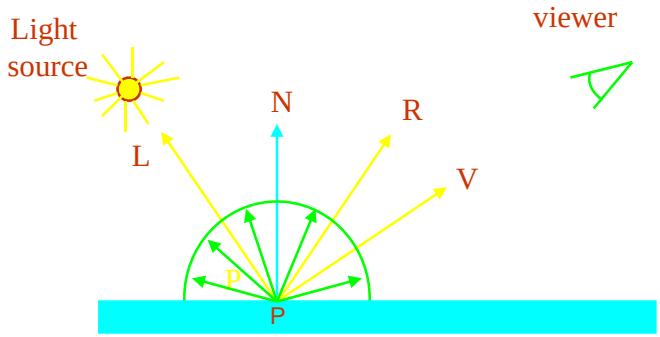
Diffuse Reflection

- Lambert Reflection (Diffuse):
  - Part of the incident light penetrates the object then arises from the object with the same intensity in all the directions
  - $I_d = K_d \cdot I_{source} \cdot \cos(N, L) / d^2$
    - K_d : diffuse color of the object
    - I_{source} : intensity of the light source
    - N : normale at point P
    - L : light direction
    - d : distance between the light source and point P
  - Related to the microscopic roughness of the object :
    - The more K_d is high, the more important is diffusion

IRISA

Diffuse Reflection

- $I_d = K_d \cdot I_{source} \cdot \cos(N, L) / d^2$
- d = distance of the light source to the lit point P



IRISA

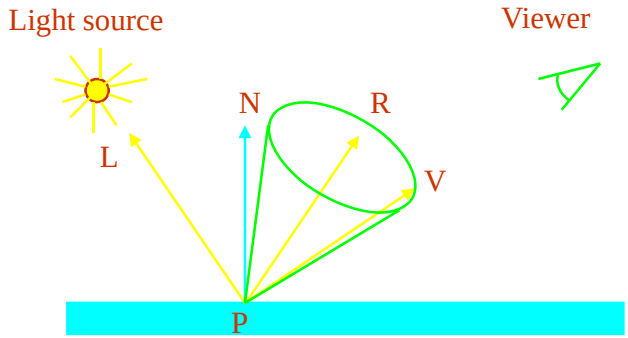
Specular Reflection

- Phong Model:
  - reflection by the object's surface of the part of the incident light that did not penetrates the object; it depends on the view direction V
  - $I_s = K_s \cdot I_{source} \cdot \cos^n(R, V) / d^2$
    - K_s : specular color of the object
    - I_{source} : intensity of the light source
    - R : direction of ideal specular reflection
    - V : view direction
    - n : shininess or roughness
      - n high: shiny, tight reflection cone
      - n small: mat object, wide reflection cone
      - For glass, n = 200
      - Gives the size of highlight

IRISA

Specular Reflection

- $I_s = K_s \cdot I_{source} \cdot \cos^n(R, V) / d^2$



IRISA

The Phong Illumination Model

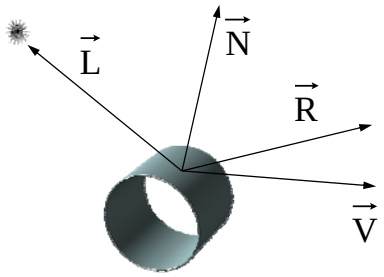
$$I_{local} = \sum_{i=0}^{nbLum} I_i \times \frac{vis(i)}{d_i^{\,2}} \times \left(k_d \left(\vec{N} \cdot \vec{L_i} \right) + k_s \left(\vec{R_i} \cdot \vec{V} \right)^n \right)$$

K_s : specular color (R,G,B)

K_d : diffuse color (R,G,B)

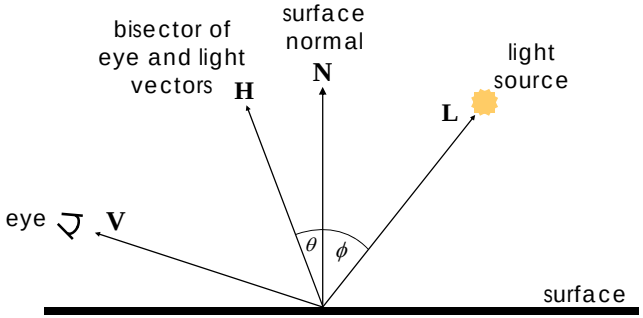
L_i : lighting direction of source i

I_i : intensity of source i (R,G,B)



IRISA

The Phong Reflection Model



$$R = I_l \left(k_d \cos(\phi) + k_s \cos^n(\theta) \right) + k_d Ia$$

IRISA

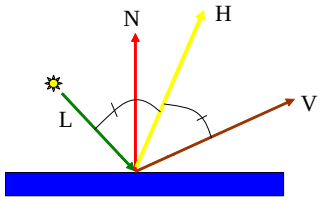
Some Remarks …

- If multiple sources, sum their contributions
- Several directions have to be known
  - The normal to the objects, the light source directions and the view direction:
    - the directions L, N et R are coplanar
    - The angle (L, N) is equal to (N, R)
- K_s and K_d are 3-component vectors:
  - Red, Green, Blue
- $I_d + I_s = (k_d \,.\, \cos(N, L) + k_s \,.\, \cos^n(R, V)) \,.\, I_{source} / d^2$
  - The term between () = BRDF

IRISA

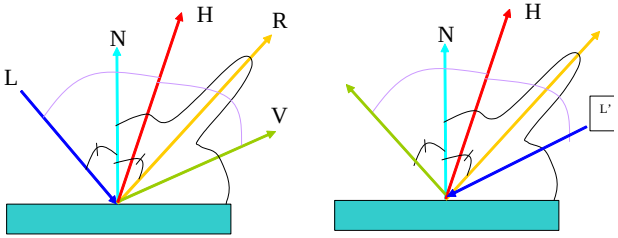
Some Remarks …

- $I_s = ks \,.\, \langle N,H \rangle^n \,.\, I_{source}$
- If the surface is perfectly specular, n is very large
- $\langle N,H \rangle^n$ is not negligible only for (N,H) = 0
- Thus Ir = ks . I_{source}
- (N,H) = 0 means that the incident and reflection angles are equal
- Only 1 reflected ray: because we assume the surface perfectly specular



IRISA

Some Remarks …

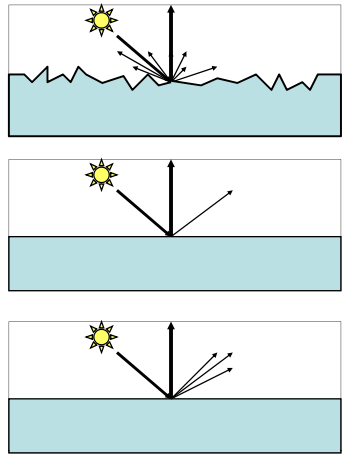


- Suppose (L',N) = (V,N) and (V',N) = (L,N)
- Then : (N,H) = (N,H')
- Ir = ks . $\langle N,H \rangle^n$. Is
- Ir' = ks . $\langle N',H' \rangle^n$. Is'
- Thus : ks . $\langle N,H \rangle^n$ = ks . $\langle N,H' \rangle^n$
- This is the reciprocity of the reflection model

IRISA

Reflections

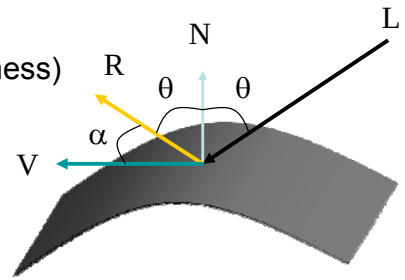
- We normally deal with a perfectly diffuse surface.
- With ray-tracing, we can easily handle perfect reflections.
- Phong allows glossy reflections of the light source.



IRISA

To sum up

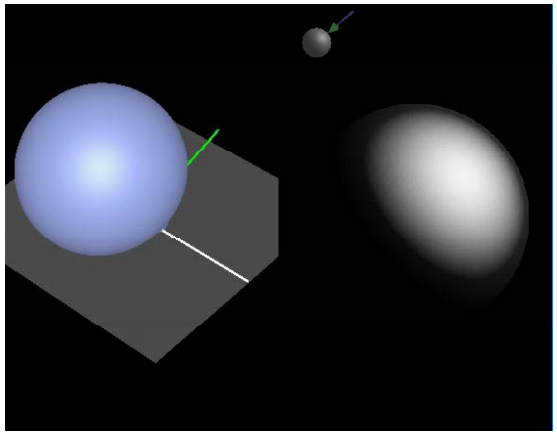
- Division of the reflectance into 2 components
  - Diffuse
    - one 3D vector k_d
  - Specular
    - one 3D vector k_s
    - one coefficient n (shininess)



IRISA

Example

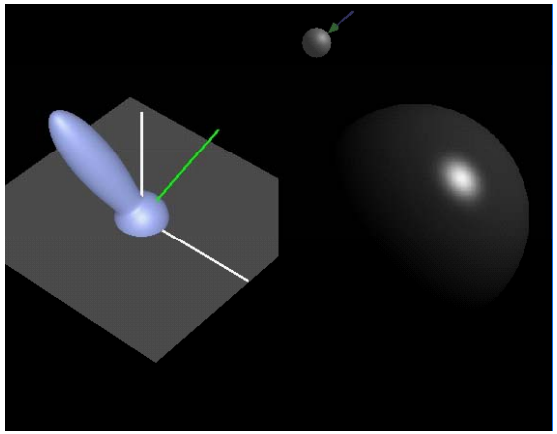
•Parameters :
- K_d =0.25
- K_s =0.75
- n=1.0



IRISA

Approximation de la réflectance

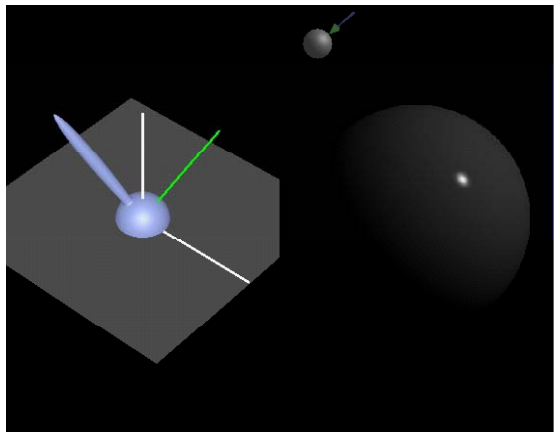
•Paramètres :
- K_d =0.25
- K_s =0.75
- n=50.0



IRISA

Approximation de la réflectance

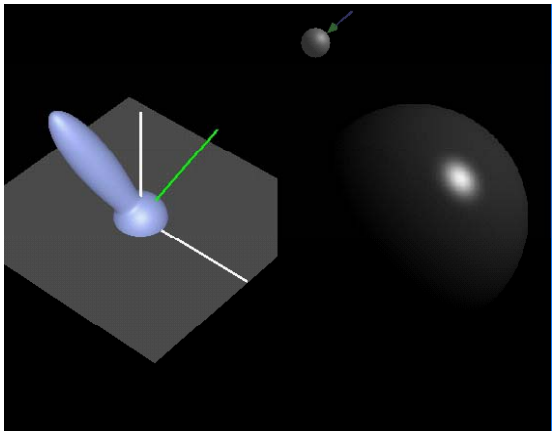
•Paramètres :
- K_d =0.25
- K_s =0.75
- n=200.0



IRISA

Approximation de la réflectance

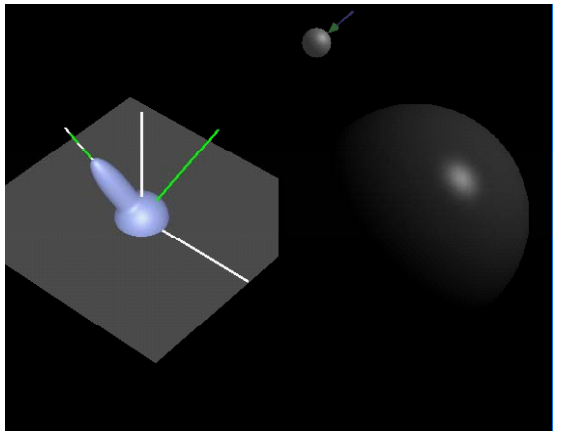
- Paramètres :
  - K_d =0.25
  - K_s =0.75
  - n=50.0



IRISA

Approximation de la réflectance

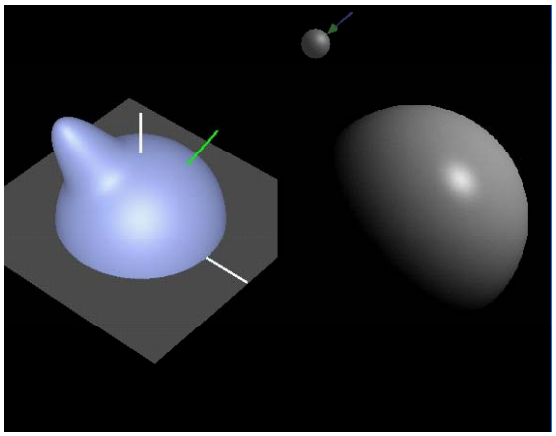
- Paramètres :
  - K_d =0.25
  - K_s =0.25
  - n=50.0



IRISA

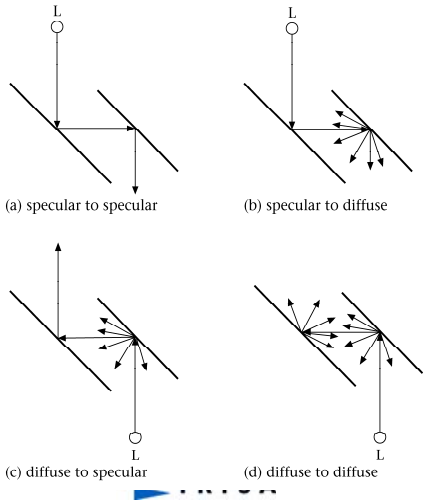
Approximation de la réflectance

- Paramètres :
  - K_d =0.75
  - K_s =0.25
  - n=50.0



IRISA

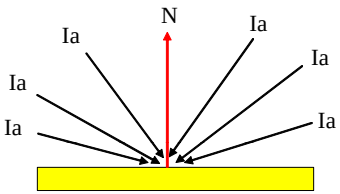
Recap: Different Light Transports



IRISA

Ambient Term

-The indirect diffuse component due to multiple reflections is supposed to be the result of the diffuse reflection of an ambient term Ia

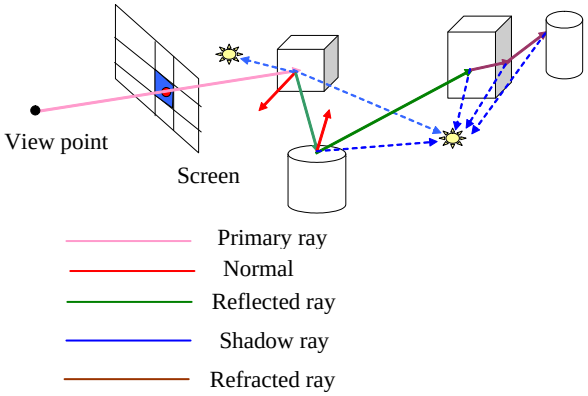


- $I_{id} = k_d \cdot I_a$

- Ia is the same for all the surfaces

IRISA

Principle



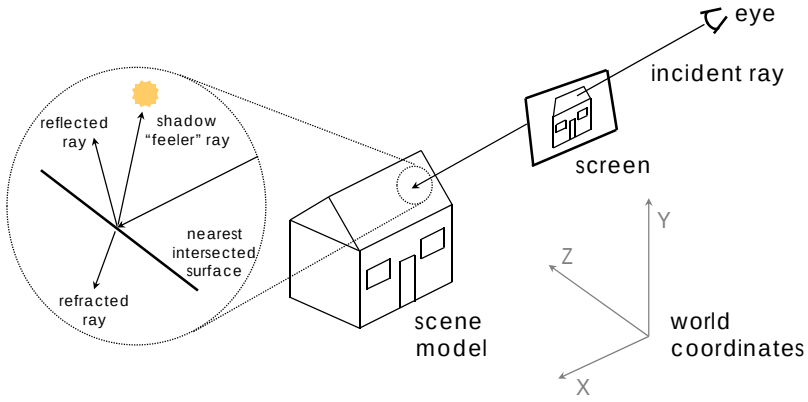
IRISA

Principle

- Trace a primary ray passing through a pixel
- P : intersection point
- Compute the contribution of the sources to P by tracing shadow rays toward the light sources.
- If a shadow ray intersects an opaque object between P and the light source then P is shadowed
- Compute the contribution to P of other points within the scene by tracing secondary rays: reflected and refracted
- A reflected ray is traced only if the material is specular
- A refracted ray is traced only if the material is transparent
- A secondary ray intersects the scene at a point P'
- Again compute the contribution of the sources to P' by tracing shadow rays toward the light sources.
- Repeat the process
- Each ray brings its contribution to the luminance of a point

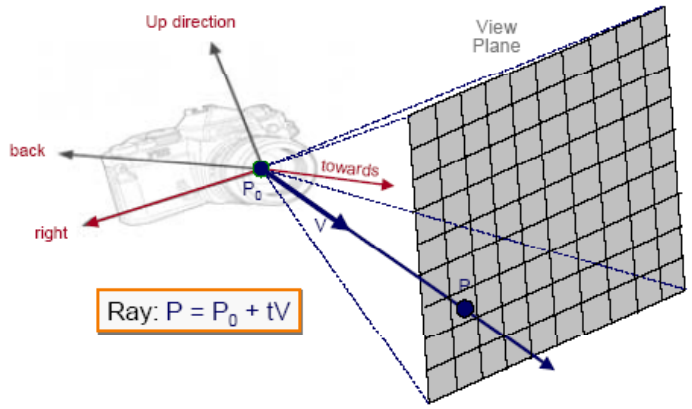
IRISA

Principle



IRISA

Principle



IRISA

Principle

2D Example

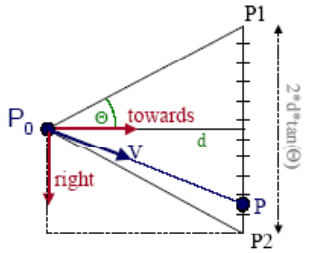
right = towards x up

Θ = frustum half-angle
d = distance to view plane

P1 = P0 + d*towards – d*tan(Θ)*right
P2 = P0 + d*towards + d*tan(Θ)*right

P = P1 + (i/width + 0.5) * (P2 - P1)
  = P1 + (i/width + 0.5) * 2*d*tan (Θ)*right

V = (P - P0) / ||P - P0 ||



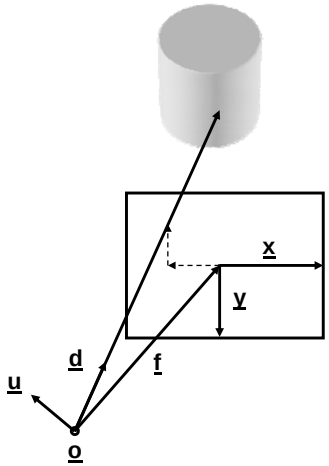
Ray: P = P0 + tV

IRISA

Ray Generation

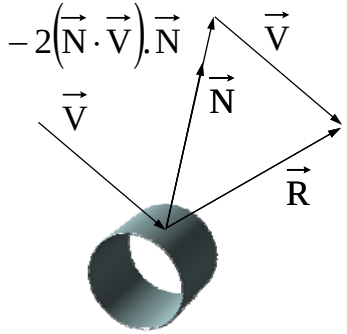
- Pinhole camera

```
for (x= 0; x < xres; x++)
  for (y= 0; y < yres; y++)
  {
    d= f + 2(x/xres - 0.5)·x
         + 2(y/yres - 0.5)·y;
    d= d/|d|; // Normalize
    r.d = d; r.o = o ;
    color= ray_cast(r,scene,depth);
    write_pixel(x,y,color);
  }
```



IRISA

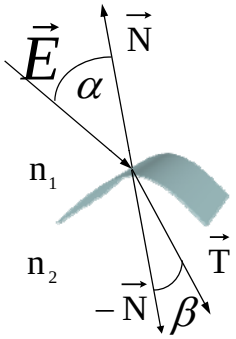
REFLECTION



$$\vec{R} = -2(\vec{N}\cdot\vec{V}).\vec{N} + \vec{V}$$

IRISA

REFRACTION



$$n_1 \sin(\alpha) = n_2 \sin(\beta)$$

IRISA

Refraction: Using Snell's Law

$$\frac{\sin\alpha}{\sin\beta} = \frac{\eta_2}{\eta_1} = \eta_{21}$$

- Using this law it is possible to show that:

$$T = -\eta_{12}E + N\left(\eta_{12}\cdot\cos\alpha - \sqrt{1+\eta_{12}^2\cdot\left(\cos^2\alpha - 1\right)}\right)$$

- Note that if the root is negative then total internal reflection has occurred and you just reflect the vector as normal

IRISA

Ray-Tracing: Pseudocode

- For each ray **r** from eye to pixel, color the pixel with the value returned by ray_cast(**r** , scene,depth):

```
ray_cast(r, scene,depth)
{
   If(depth >Max_Depth) {color ← black}
   else {
      If (intersection(r,scene)) {
      p ← point_of_intersection(r, scene);
      u← reflect(r, p);
      v← refract(r, p);
      color ← phong_direct(p, r) +
           k_s × ray_cast(u, scene, depth+1) +
           kt× ray_cast(v, scene , depth+1);
       } else color ← background_color ;
    }
return(color);
}
```

IRISA

Pseudocode Explained

- **p** ← point_of_intersection(**r**, scene);
  - Compute **p**, the point of intersection of ray **r** with the scene
- **u**← reflect(**r**, **p**); **v**← refract(**r**, **p**);
  - Compute the reflected ray **u** and the refracted ray **v** using Snell's Laws

IRISA

Pseudocode Explained

- phong(**p**, **r**)
  - Evaluate the Phong reflection model for the ray **r** at point **p** on surface s, taking shadowing into account
- $k_s \times$ ray_cast(**u,scene,depth**)
  - Multiply the contribution from the reflected ray **u** by the specular color k_s for surface s containing **p.** **Only (specular-to-specular)* light transport is handled. Ideal specular (mirror) reflection**
- $k_t \times$ ray_cast(**v,scene,depth**)
  - Multiply the contribution from the refracted ray **v** by the specular-refraction coefficient k_t for surface s. **Only (specular-refraction)* light transport is handled**

IRISA

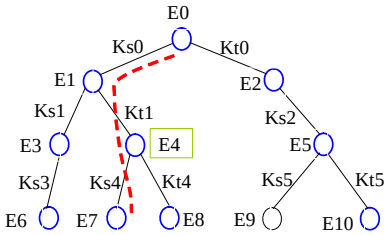
About Those Calls to ray_cast()...

- The function ray_cast() calls itself *recursively*
- There is a potential for infinite recursion
  - Consider a "hall of mirrors"
- Solution: limit the depth of recursion
  - A typical limit is five calls deep
  - Note that the deeper the recursion, the less the ray's contribution to the image, so limiting the depth of recursion does not affect the final image much

IRISA

About Those Calls to ray_cast()...

- Another solution
  - Ei: direct lighting at point Pi
  - Ks: vector (R,G,B)
  - Kt: scalar ranging between 0 and 1
  - Contribution of the red path



- I : Intensity due to this ray path :

I = Ks0 . (Kt1 (Ks4 . E7 + E4) + E1)
= Ks0 . Kt1 . Ks4 . E7 + Ks0 . .Kt1 . E4 + Ks0 . E1

- Stop tracing rays when the cumulative product Ks.Kt… is below a certain threshold

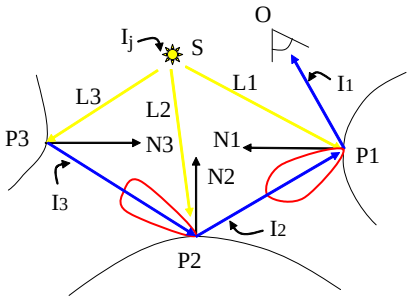
IRISA

Example

H1 : bisecting line of angle S P3 P2
H2 : bisecting line of angle S P2 P1
H1 : bisecting line of angle S P1 O
Idai : intensity due to direct lighting and the ambient term for point Pi

Idai = kdi . Ia
+ kdi . Is . cos(Li,Ni)
+ ksi . Is . cos(Ni,Hi)^n

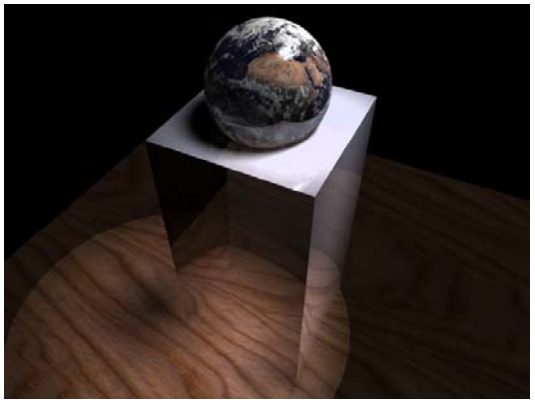
I3 = Ida3
I2 = Ida2 + ks2 . I3
I1 = Ida1 + ks1 . I2



IRISA

Reflections

- If only one reflected ray is considered, then ray-tracing will only handle perfect mirrors.



IRISA

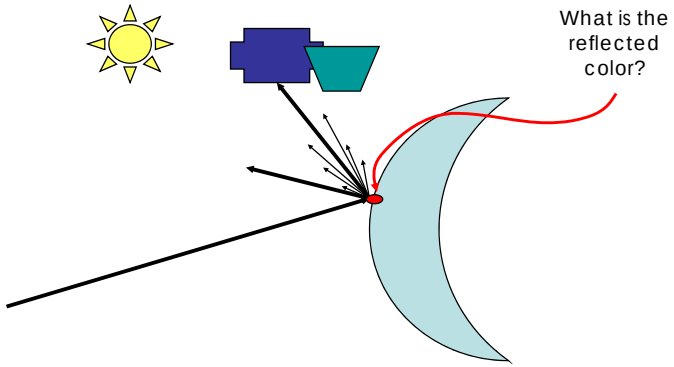
Reflections

- Glossy reflections (multiple reflected rays) blur the reflection.

IRISA

Reflections

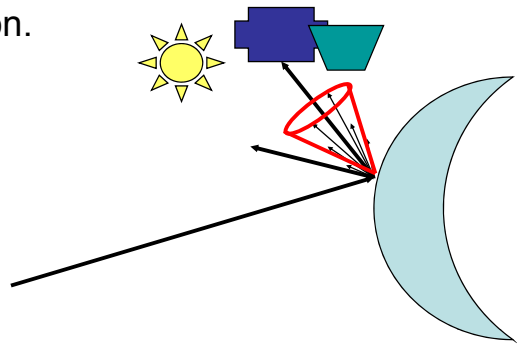
- Mathematically, what does this mean?



IRISA

Glossy Reflections

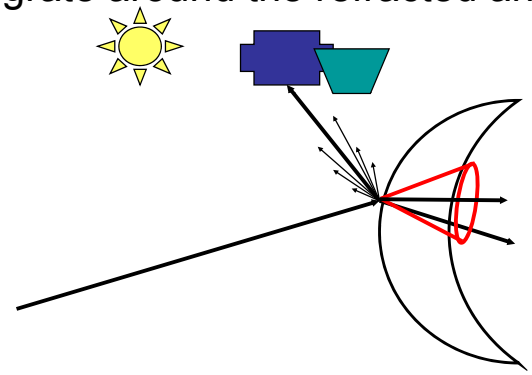
- We need to integrate the color over the reflected cone.
- Weighted by the reflection coefficient in that direction.



IRISA

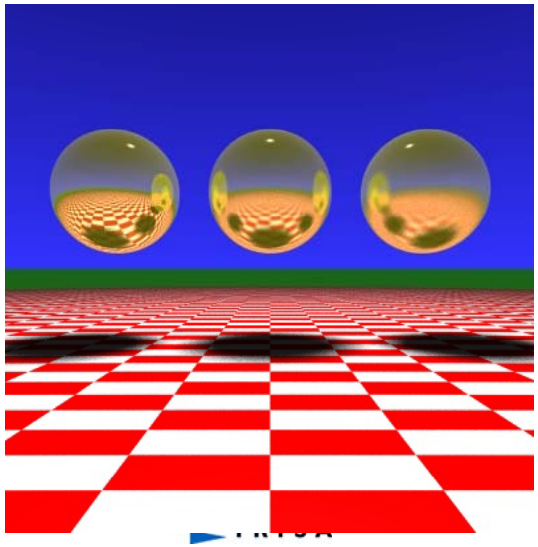
Translucency

- Likewise, for blurred refractions, we need to integrate around the refracted angle.

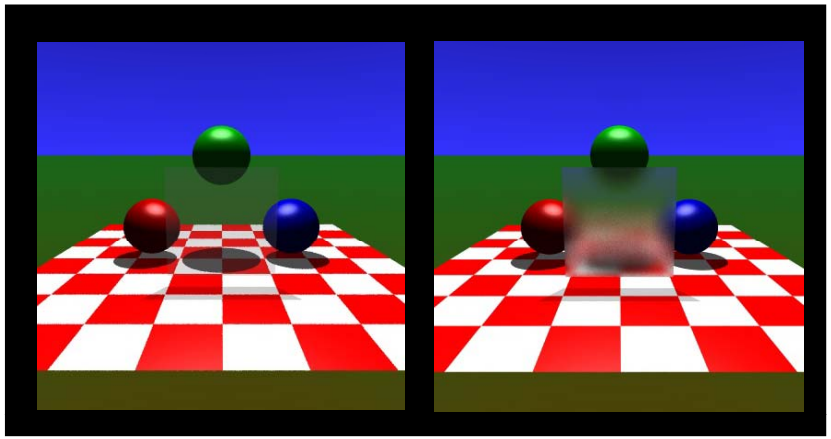


IRISA

Translucency



Translucency



IRISA

Calculating the integrals

- How do we calculate these integrals?
  - Two-dimensional of the angles and ray-depth of the cone.
  - Unknown function -> the rendered scene.
- Use Monte-Carlo integration

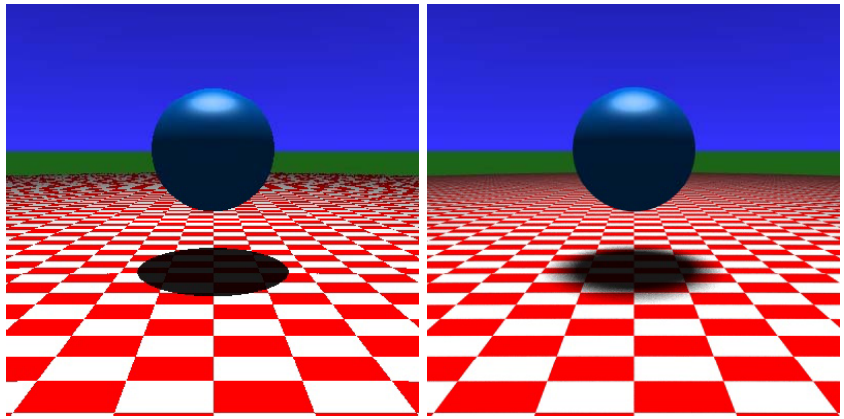
IRISA

Shadows

- Ray tracing casts shadow from a point light source.
- Many light sources are illuminated over a finite area.
- The shadows between these are substantially different.
- Area light sources cast soft shadows
  - Penumbra
  - Umbra

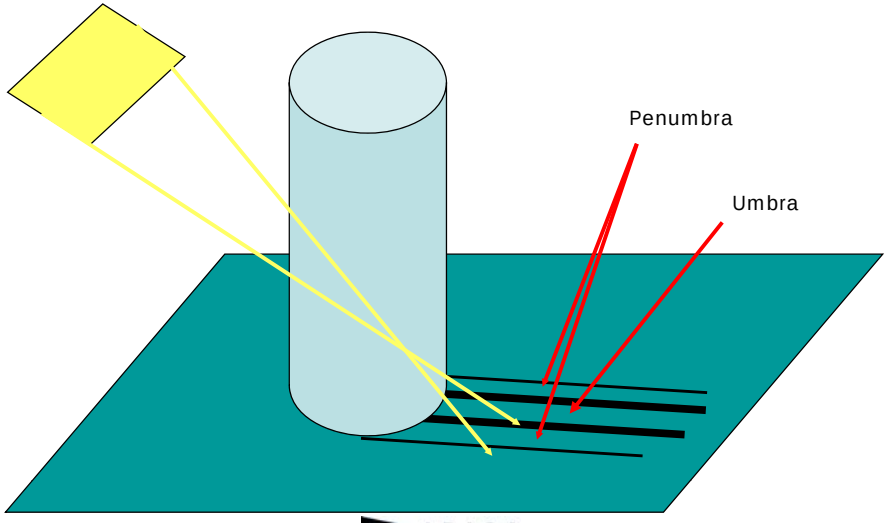
IRISA

Soft Shadows



IRISA

Soft Shadows



IRISA

Soft Shadows

- Umbra – No part of the light source is visible.
- Penumbra – Part of the light source is occluded and part is visible (to a varying degree).
- Which part? How much? What is the Light Intensity reaching the surface?

IRISA

Pros and Cons of Ray Tracing

- Advantages of ray tracing
  - All the advantages of the Phong model
  - Also handles shadows, reflection, and refraction
- Disadvantages of ray tracing
  - Computational expense
  - No diffuse inter-reflection between surfaces
  - Not physically accurate
- Other techniques exist to handle these shortcomings, at even greater expense!

IRISA

An Aside on Antialiasing

- Our simple ray tracer produces images with noticeable “jaggies”
- Jaggies and other unwanted artifacts can be eliminated by antialiasing:
  - Cast multiple rays through each image pixel
  - Color the pixel with the average ray contribution
  - An easy solution, but it increases the number of rays, and hence computation time, by an order of magnitude or more

IRISA

Intersection

Principle

- The scene is supposed to be expressed in the world coordinate system (WCS).
- It may be: A set of independent objects
- Purpose: intersect a scene with a ray whose equation is given by :
- $P = P_0 + t \, . \, D$
- where :

  P_0 is the ray origin;

  D = (dx, dy, dz) is the direction vector of the ray ;

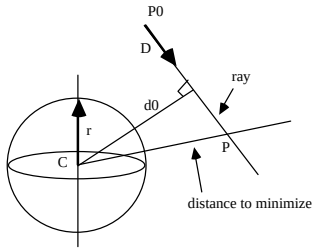
  $t > 0$
- Intersection result = { ti / ti is a value of t corresponding to an intersection point }.
- Only the closest point to the ray origin is used to compute shading and secondary shot rays.

IRISA

Intersection

Sphere

- d_0: Orthogonal distance between the ray and the center of the sphere of radius r and center C
- $P = P_0 + t \, . \, D$: the ray equation
- $P_0 = (X_0, Y_0, Z_0)$ D=(dx,dy,dz)
- If $d_0^2 <= r^2$, then the ray intersects the sphere
- Intersection points = solutions of $|| P_0 - C ||^2 + 2t \, . \, (P_0 - C) \bullet D + t^2 \, . \, || D ||^2 = r^2$
- d_0 is evaluated by minimizing the distance d between C and a point P on the ray.
- This gives: $d^2 = || P_0 + t \, . \, D - C ||^2 = || P_0 - C ||^2 + 2t \, . \, (P_0 - C) \bullet D + t^2 \, . \, || D ||^2$
- By setting to 0 the derivative of d^2 , we obtain : $t = ((P_0 - C) \bullet D \; / || D ||^2) = - (P_0 - C) \bullet D$
- After substitution : $d_0^2 = || P_0 - C ||^2 – ((P - C) \bullet D)^2$



IRISA

Intersection

Axis-aligned Parallelepiped

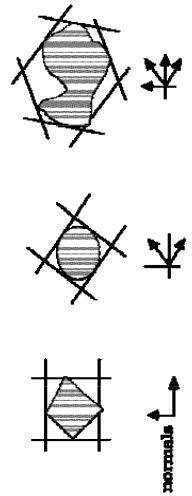
- Faces: perpendicular to the axes of the world coordinate system.
- First, the intersections between the ray and the faces x = x1 and x = x2 are computed.
- Two values of t are then obtained
- t1 = (x1 - x0) / dx and t2 = (x2 - x0) / dx.
- Interval: [Ix, Mx] = [min(t1, t2), max(t1, t2)]
- Same processing applied to the faces perpendicular to the y and z axes. Two other intervals: [Iy, My] and [Iz, Mz]
- The result is then an intersection interval given by :
  [I, M] = [max(Ix, Iy, Iz), min(Mx, My, Mz)]
- If I <= M then the ray intersects the parallelepiped bounding volume, otherwise it does not intersect it
- Closest intersection point: t=I

IRISA

Intersection

Polyhedron

- Polyhedron = set of pairs of parallel faces
- Ni: normal to a pair of faces
- A pair of parallel faces is called slab



IRISA

Intersection

Polyhedron

- The intersection test is similar to that of a parallelepiped, except that the faces are not perpendicular to the axes of the coordinate system
- For each pair i, compute interval $[I_i, M_i]$
- Let N be the normal to a face
- N ● P + d = 0 the equation of the plane containing the face.

IRISA

Intersection

Polyhedron

- The value of t corresponding to the intersection between the ray and this face is computed by substituting the ray equation into that of the plane :
  t = - (d + N ● P0) / N ● D
- For each slab i , N=Ni and ===→

$$t = \alpha_i * d + \beta i$$

$$\alpha i = \frac{-1}{Ni \bullet D}$$

$$\beta i = \frac{-Ni \bullet P0}{Ni \bullet D}$$

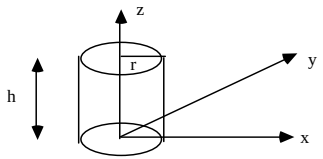
- Given a slab i, these values are the same for all the polyhedra used as object bounding volumes

IRISA

Intersection

Cylinder

- The cylinder : intersection between an infinite height cylinder and the subspace delimited by two planes which equations are
  z = 0 and z= h
- The intersection between the ray an the infinite height cylinder is first performed. This yields a first interval [t1,t2]
- The intersection with the two planes gives a second interval [t3, t4].
- The final intersection interval [I, M] results from the combination of these two intervals (as for the parallelepiped).



IRISA

Intersection

Cylinder: continued

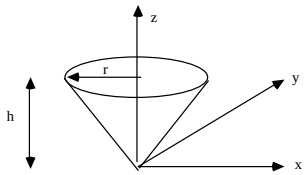
- obtaining [t1, t2]
  - The equation of the infinite height cylinder :
  - $x^2 + y^2 = r^2$
  - Substituting the ray equation in this equation we obtain:
    $t^2 . (dx^2 + dy^2) + 2t . (x_0 . dx + y_0 . dy) + (x_0^2 + y_0^2 - r2) = 0$
  - Solving this equation gives the interval [t1, t2].
- *obtaining [t3, t4]*
  - Let A and B the two values of t resulting from the intersection with the two planes :
    $A = - z_0 / dz$ and $B = (h - z_0) / dz$
  - We get :
    t3 = min(A,B) and t4 = max(A, B)

IRISA

Intersection

Cone

- Intersection: performed in the LCS of the cone
- Cone: intersection between an infinite height cone and the subspace delimited by two planes, the equations of which are z = 0 and z = h.
- Intersection between the ray and the infinite height cone is first performed.
- The equation of this cone is given by :
  h2 . (x2 + y2) - r2 . z2 = 0.



IRISA

Intersection

Cone

- Substituting the ray equation in this equation yields an interval [t1, t2].
- Then the planes are in their turn intersected to give a second interval [t3, t4] such that :
  t3 = min(A, B) and t4 = max(A, B)
- where A = - z0 / dz and B = (h - z0) / dz.
- The final interval is the combination of these two intervals (as for the cylinder)

IRISA

Intersection

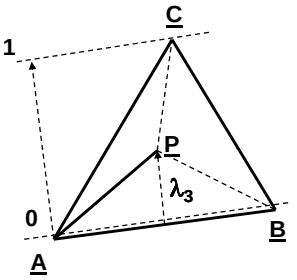
Polygon

- Several ray-polygon intersection methods have been proposed in the literature.
- Only two of them are presented .
- For all these methods, the intersection process consists of two steps :
  - *First step: Ray-Plane intersection test*
    - the goal of the first step is to perform the intersection between the ray and the plane containing the polygon
  - *Second step: Inside - Outside test*
    - the second step tests if the resulting point is inside or outside the polygon.

IRISA

Intersection - Triangle

- Barycentric coordinates
  - Non-degenerate triangle ABC
    $\underline{P} = \lambda_1\underline{A} + \lambda_2\underline{B} + \lambda_3\underline{C}$
  - $\lambda_1 + \lambda_2 + \lambda_3 = 1$
  - $\lambda_i >= 0$
  - λ_3 = area(APB) / area(ACB),
  - λ_2 = area(APC) / area(ACB),
  - λ_1 = area(CPB) / area(ACB),
  - Area(APB)= $\frac{1}{2}\det(P\vec{A}, P\vec{B}) = \frac{1}{2}\left|P\vec{A}\times P\vec{B}\right| = \frac{1}{2}\left|P\vec{A}\right|\left|P\vec{B}\right|\sin(\hat{P})$



IRISA

Intersection

- **Polygon: Snyder's method**

- Ray-triangle intersection: extension to a polygon.
- Let Pi be the vertices of a triangle and Ni the associated normals which are used for normal interpolation across the triangle.
- Normal to the triangle: N = (P1 - P0) x (P2 - P0)
- A point P lying on the triangle plane satisfies :
  P ● N + d = 0 where d = - P0 ● N.
- To intersect a ray P = O + t . D with a triangle, first compute the t parameter of the intersection between the ray and the triangle plane
  t = (d - N ● O) / N ● D.

IRISA

Intersection

Polygon: Snyder's method

- Projecting the triangle into any other plane, except one that is orthogonal to the triangles plane will not change the barycentric coordinates of the triangle.
- This allows to simplify computations, since we can choose any of the coordinate system's three axis-aligned planes to project our triangle, thus throwing away one of the three coordinates and reducing the barycentric equations to R^2.
- For reasons of numerical stability we want to choose the dominant axis of the triangles normal for the projection.
- An index i0 is computed: equal either to 0 if | Nx | is maximum (i.e. the x axis is dominant) or to 1 if | Ny | is maximum or to 2 if | Nz | is maximum.

IRISA

Intersection

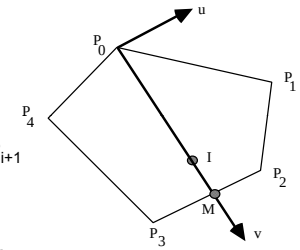
Polygon: Snyder's method

- Let i1 and i2 (i1, i2 ∈{0, 1, 2}) be two unequal indices different from i0. Compute the i1 and i2 components of the intersection point I:
  $Ii1 = Oi1 + t \cdot Di1$ and $Ii2 = Oi2 + t \cdot Di2$
- The inside-outside test can be performed by computing scalars ß0, ß1 and ß2 according to :
  $ßi = [(Pi+2 - Pi+1) \times (I - Pi+1)]i0 \ / \ [N]i0$
- The ßi are the barycentric coordinates of the point where the ray intersects the triangle plane.
- I is inside the triangle if and only if $0 \le ß \le 1$ for i {0, 1, 2}.
- The interpolated normal at point I is given by :
  $N' = ß0 \cdot N0 + ß1 \cdot N1 + ß2 \cdot N2.$
- Snyder's method can be easily extended up to polygons.
- The main idea is to consider a polygon as a union of triangles.

Intersection

Marchal's method

- I is the ray-plane intersection point.
- The Pi are transformed to the two dimensional coordinates system (u, v) whose origin is vertex P_0.
- The plane of this coordinates system is the polygon plane.
- The inside-outside test determines if an edge P_iP_{i+1} intersects the v axis at a point M (this may occur when the u components of P_i and P_{i+1} have different signs).
- If so, and if $P_0I < P_0M$ then I is inside the polygon, else it is outside.
- On the other hand, if none of the edges intersect the v axis, then I lies outside the polygon.



Intersection

Marchal's method

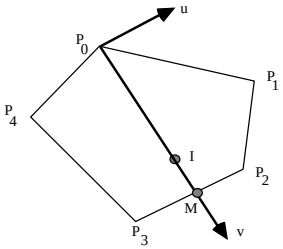
- The interpolated normal at point I is given by :

  $N_I = (P_0I / P_0M) \bullet N_M + (1 - P_0I / P_0M) \bullet N_0$

- where the normal N_M at point M is given by :

  $N_M = (P_iM / P_iP_{i+1}) \bullet N_{i+1} + (1 – P_iM/ P_iP_{i+1}) \cdot N_i$

and N_i, N_{i+1} are the normals at point P_i and P_{i+1}. P_iP_{i+1} is the intersected edge.

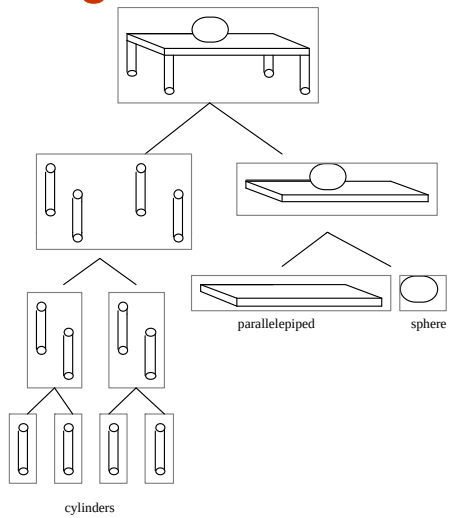


Bounding box

- To reduce the amount of ray-object intersections, its is absolutely necessary to use a hierarchical data structure .
- This data structure is a tree of bounding volumes.
- Bounding volumes are simple geometric objects which fit around the objects.
- They are chosen to be simple to intersect with a ray, such as spheres or parallelepipeds that have faces perpendicular to the axes.

Bounding box

- Example of a hierarchy of bounding boxes : binary tree.



IRISA

Bounding Volume

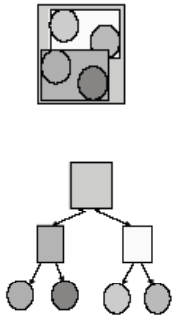
Different kinds of bounding Volume

- **Parallelepiped**
  - For the sake of speed up, the faces of this bounding volume are perpendicular to the axes of the World Coordinates System.
  - Its perspective projection onto the screen plane is often used to filter the primary rays (rays starting at the eye location).
- **Sphere and Ellipsoid**
  - They may be used to filter the reflected and refracted rays and those directed to the light sources.
- **Polyhedron**
  - Intersection of slabs: a slab is a pair of parallel faces

IRISA

Bounding Volume Hierarchy

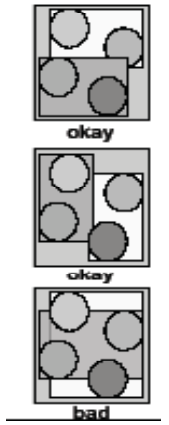
- Organize objects into a tree
- Group objects in the tree
  - based on spatial relationships
- Each node in the tree contains a bounding box of all the objects below it



IRISA

Bounding Volume Hierarchy (BVH)

- Determining optimal BVH structure is NP-hard problem
- Heuristic approaches:
  - Cost models (minimize volume or surface area)
  - Spatial models
- Categories of approaches:
  - Top down
  - Bottom up

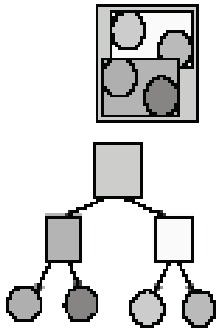


IRISA

Median Cut BVH Construction

Top down approach:

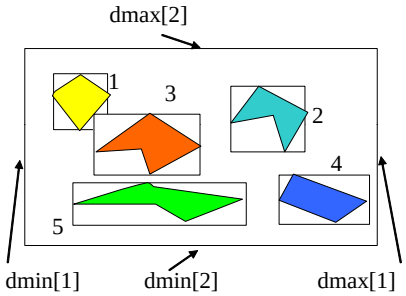
- Sort objects by position on axis
  - cycle through x,y,z
  - use center of bounding box
- Insert tree node with half of objects on left and half on right



IRISA

Median Cut BVH Construction

1. L= {list of bounding volume numbers}
2. Choose widest slab:
   dmax[2] – dmin[2] or dmax[1] – dmin[1]
   (In this example :
   max width = dmax[1] – dmin[1]
3. Then choose slab of max width
4. Sort the bounding volumes wrp to increasing dmin[number_of_widest_slab]
5. We get a sorted list L = {1,5,3,2,4}
6. Split L into two sub-lists L1 and L2
7. We get : L1 = {1,5,3} L2 = {2,4}
8. Go to 1 with L = L1 then L = L2



Leaf = one or more objects

IRISA

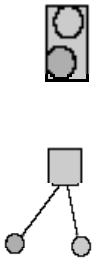
Bottom up BVH Construction

- Add objects one at a time to tree
- Insert to subtree that would cause smallest increase to area

IRISA

Bottom up BVH Construction

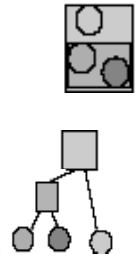
- Add objects one at a time to tree
- Insert to subtree that would cause smallest increase to area



IRISA

Bottom up BVH Construction

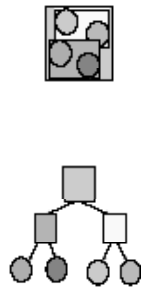
- Add objects one at a time to tree
- Insert to subtree that would cause smallest increase to area



IRISA

Bottom up BVH Construction

- Add objects one at a time to tree
- Insert to subtree that would cause smallest increase to area



IRISA

Intersection Test Using the BVH

- Once the hierarchy of bounding volumes has been built, the ray-scene intersection test is performed as follows.
  - The hierarchy is searched from the root to the leaves.
  - During this search, at a node N, the associated bounding volume is checked for an intersection with the current ray.
  - If the bounding volume of N is intersected, those of its children node are in their turn checked for an intersection.
- This process is repeated recursively and ends up at the leaf nodes.
- Else, if the bounding volume of N is not intersected by the ray, the associated subtree is left out, that is, it is not searched, which saves time.

IRISA

Spatial Subdivision

- The rectangular bounding volume of the scene is subdivided into 3D cells
- Each cell contains a few objects of the scene
- When a ray enters a cell, we check the objects within this cell for an intersection with the ray
- If the intersection process ends up with success then no need to check the rest of the objects
- If the ray fails to hit any object in the cell then it moves to the next 3D cell
- Repeat the process until intersection is found

IRISA

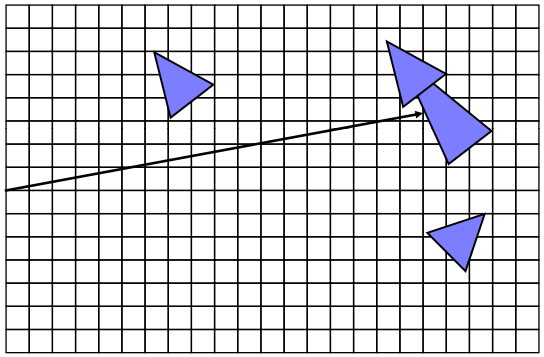
Spatial Subdivision

- **Two procedures**
  - A procedure which performs a spatial subdivision of the scene into 3D cells, each of them containing a small portion of the database
  - A second procedure which determines the next cell along a ray

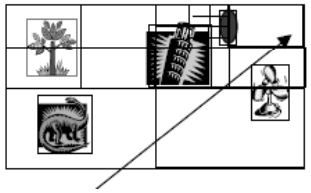
IRISA

Spatial Subdivision

- **Two procedures**



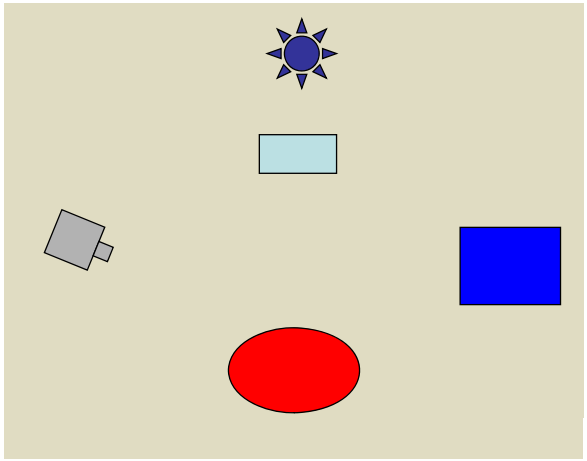
Uniform Grid

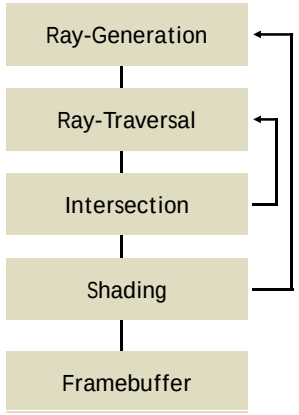


Non uniform Grid

IRISA

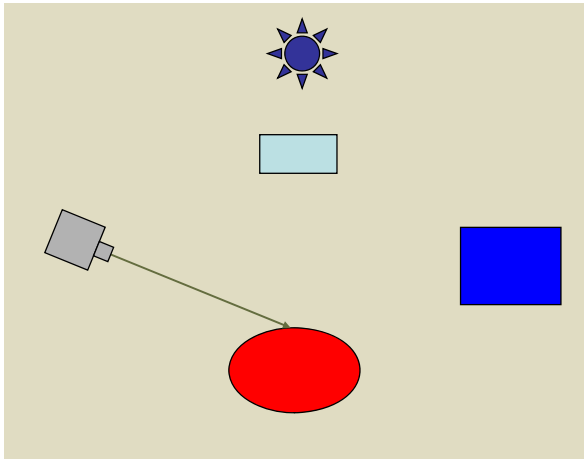
What is Ray Tracing in a subdivided space?

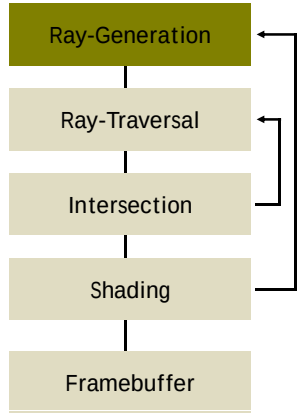




IRISA

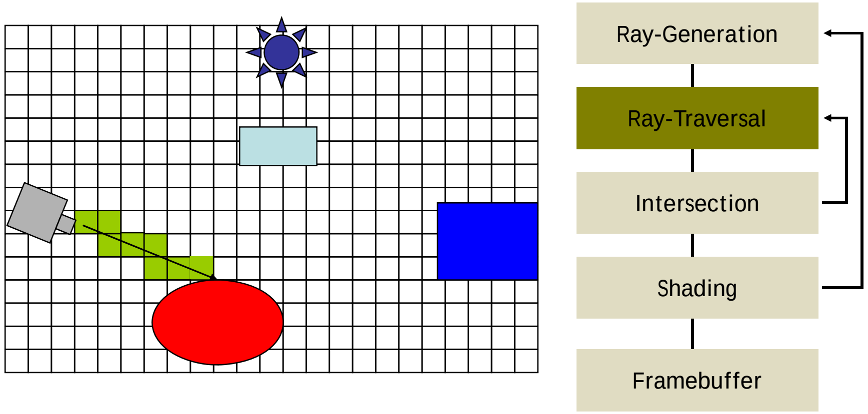
What is Ray Tracing in a subdivided space?



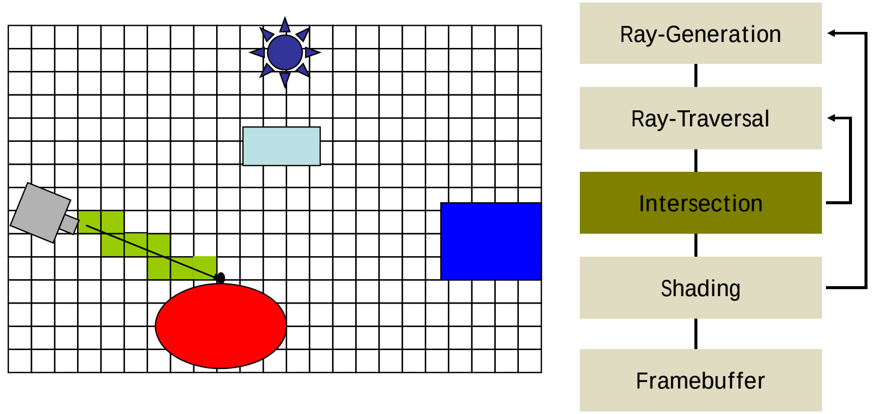


IRISA

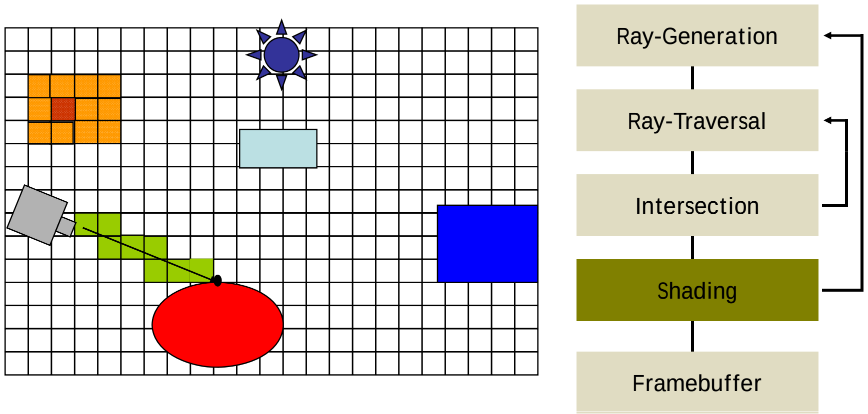
What is Ray Tracing in a subdivided space?



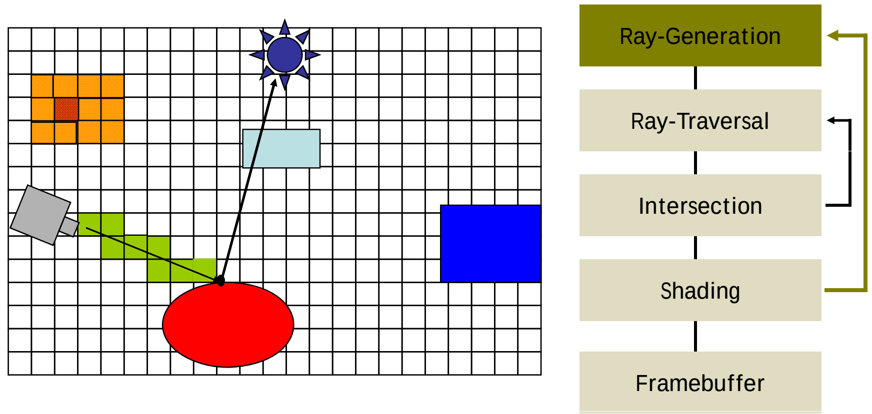
What is Ray Tracing in a subdivided space?



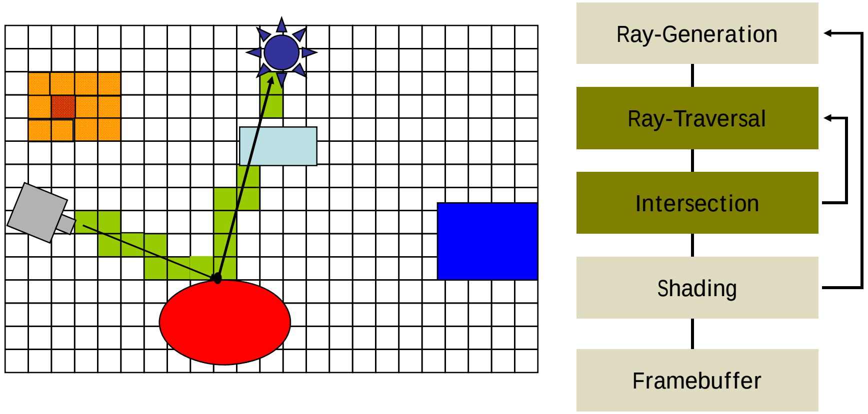
What is Ray Tracing in a subdivided space?



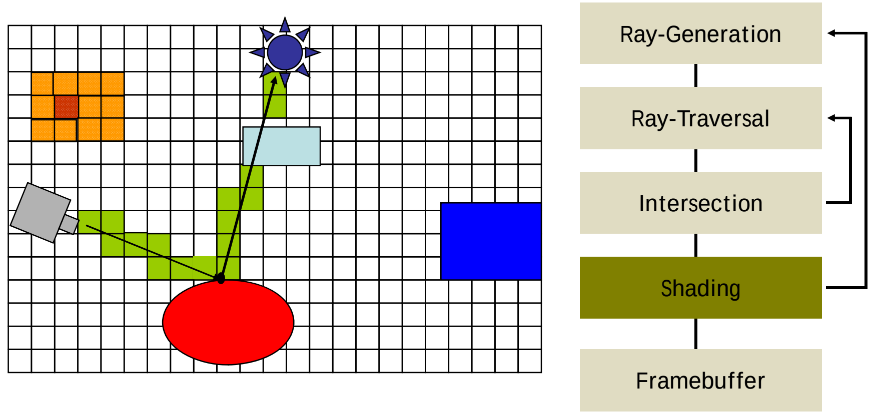
What is Ray Tracing in a subdivided space?



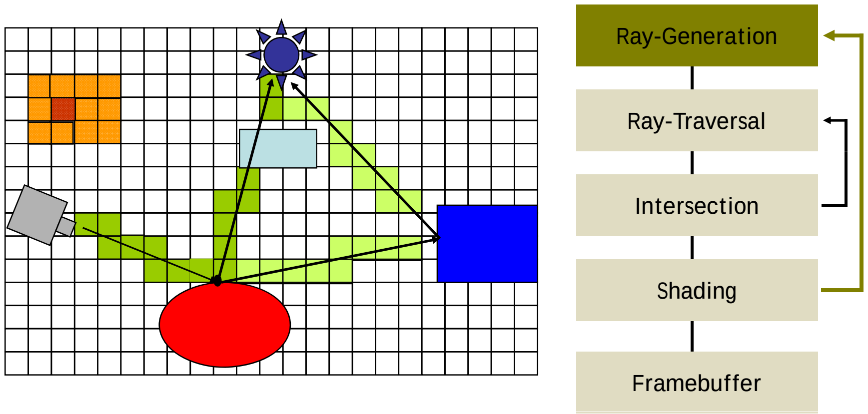
What is Ray Tracing in a subdivided space?



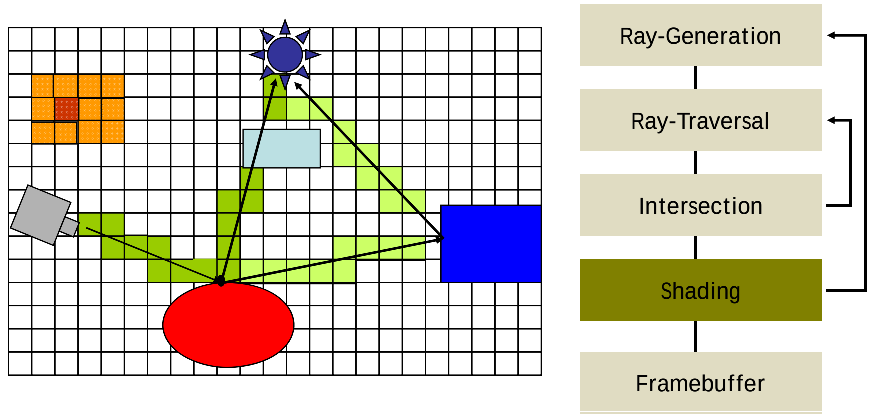
What is Ray Tracing in a subdivided space?



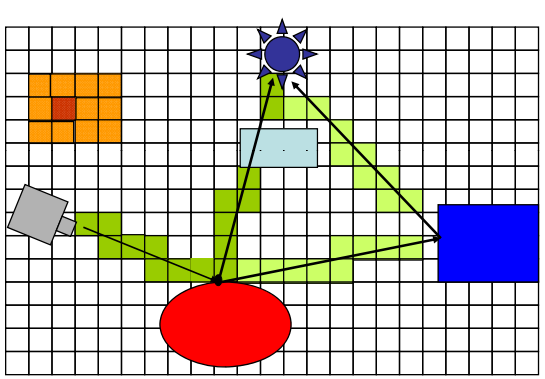
What is Ray Tracing in a subdivided space?

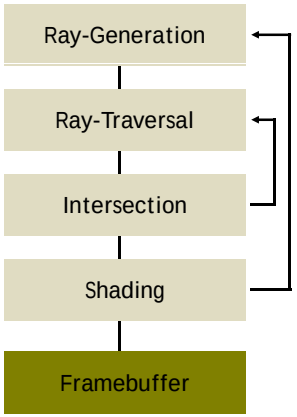


What is Ray Tracing in a subdivided space?



What is Ray Tracing in a subdivided space?

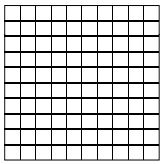


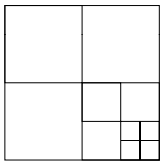


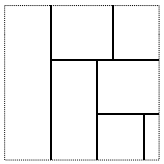
IRISA

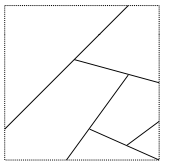
Spatial Subdivision

- Different kinds of subdivision


Uniform Spatial Sub


Quadtree/Octree

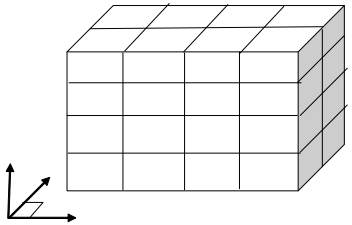

kd-tree


BSP-tree

IRISA

Uniform Grid

- The rectangular bounding volume of the scene is subdivided into a uniform 3D grid of rectangular cells
- The grid is represented by a 3D array, the indices of which are i, j and k corresponding to the x, y and z axes respectively
- Each cell is represented by a data structure containing a pointer to the objects partially or totally within the cell



IRISA

Uniform Grid

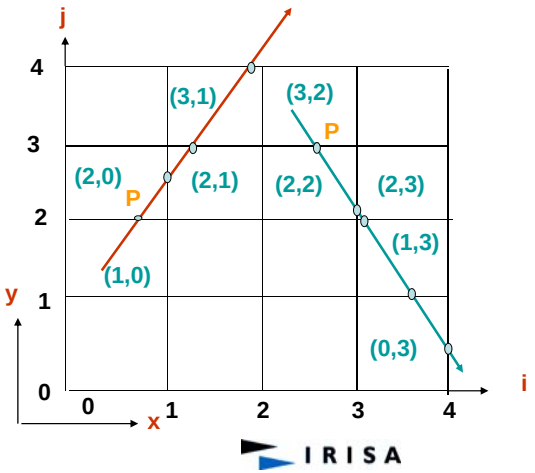
Ray Traversal Algorithm: Classical Method

- Let G[i][j][k] be the 3D array representing the 3D grid
- Let P the point where the ray leaves the current cell and D the ray direction
- P is the outgoing point
- Let w be the axis perpendicular to the face which contains P
- Let u (x, y or z) be the index (i, j or k) of the current cell corresponding to w
- If Dw > 0 then the index u of the next cell is u = u + 1, the other indices are unchanged
- Else it is : u = u – 1
- Example :
- If w = z then u = k
- If Dz > 0 then the index of the next cell along the ray is k = k + 1, while the other indices do not change
- If the current cell is G[i][j][k] then the next cell along the ray is G[i][j][k + 1] if Dz > 0, or G[i][j][k - 1] if Dz < 0

IRISA

Uniform Grid

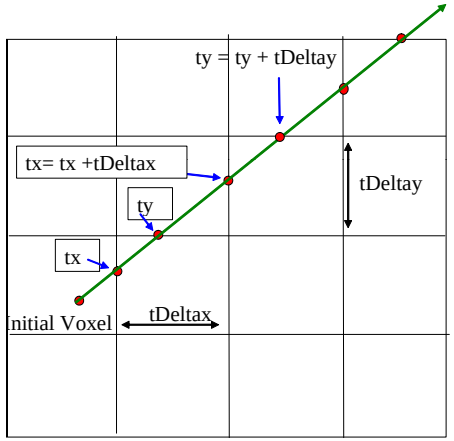
Ray Traversal Algorithm: Classical Method



IRISA

Uniform Grid

- **Ray Traversal Algorithm: Amanatide's Algorithm**



IRISA

Uniform Grid

Ray Traversal Algorithm: Amanatide's Algorithm

Initialization

- Ray equation : P = P0 + t . D
- Identify the voxel containing the ray origin O
- If O is outside the grid, find the point through which the ray enters the grid and determine the adjacent voxel
- X, Y and Z : voxel indices
- StepX, stepY and stepZ : initialized to 1, incremented or decremented as the ray crosses the voxel boundaries
- tx, ty and tz : values of t corresponding to the points resulting from the intersection between the ray and 3 faces of the initial voxel
- tDeltaX, tDeltaY and tDeltaZ : distance travelled by the ray between two successive faces perpendicular to the x, y and z faces respectively

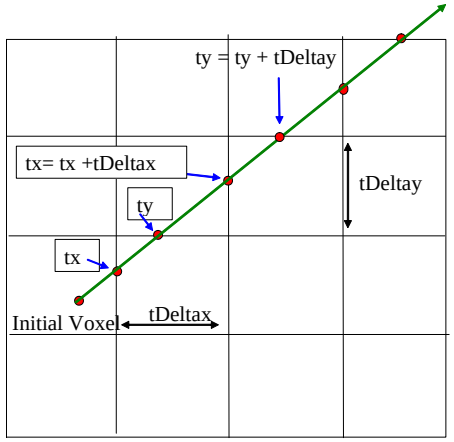
IRISA

Uniform Grid

Ray Traversal Algorithm: Amanatide's Algorithm

Algorithm

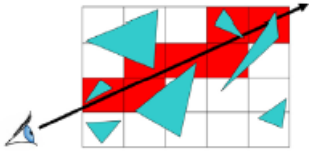
```
Min = min(tx,ty,tz) ;
switch(Min)
{
    case tx :
            X += stepX ;
            tx += tDeltax ;
            break ;
    case ty
            Y += stepY ;
            ty += tDeltay ;
            break ;
    case ty
            Z += stepZ ;
            tz += tDeltaz ;
            break ;
}
```



IRISA

Uniform Grid

- Advantages?
  - easy to construct
  - easy to traverse
- Disadvantages?
  - may be only sparsely filled
  - geometry may still be clumped (say, densely grouped)



IRISA

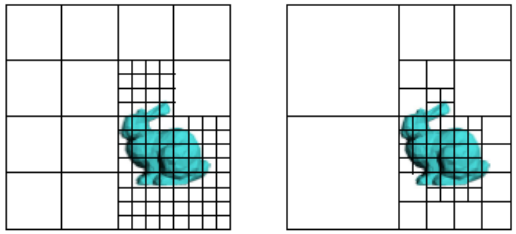
Non Uniform Grid

- The rectangular bounding volume of the scene is recursively sliced :
  - either simultaneously by 3 planes perpendicular to the x, y and z axes: Octree
  - or by one plane at a time perpendicular to an axis: Kd-tree, Bsp tree
  - or by one plane at a time non necessary perpendicular to an axis: Bsp tree
- Each slicing plane divides a space (a 3D cell) into two subspaces (3D cells)
- The subdivision process stops either when a cell contains partially or totally a minimum number of objects, or the maximum subdivision level is reached for each axis
- The result is a linear array of rectangular cells or a binary tree or an octree
- Each cell is represented by a data structure containing a pointer to the objects partially or totally within it

IRISA

Non Uniform Grid

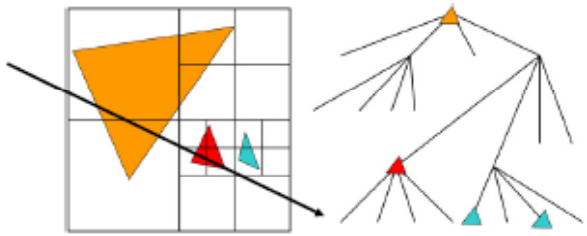
- Subdivide until each cell contains no more than *n* elements, or maximum depth *d* is reached



IRISA

Non Uniform Grid

- Advantages?
  - grid complexity matches geometric density
- Disadvantages?
  - more expensive to traverse (especially octree)



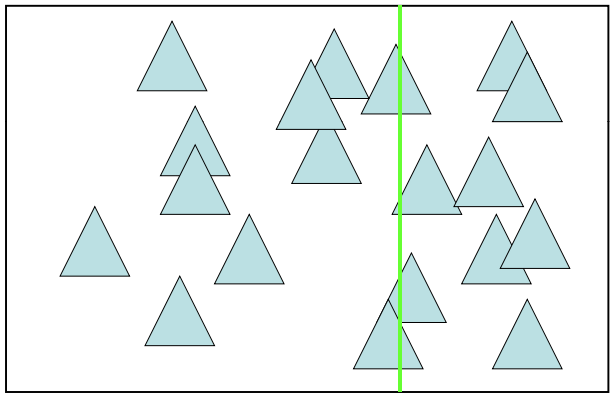
IRISA

Non Uniform Grid: Kd-Tree

- Subdivide only 1 dimension
- Do not subdivide at the center
- Which axis to pick?
- What point on the axis to pick?
- One heuristic:
  - Sort objects on each axis
  - Pick point corresponding to “middle” object
  - Pick axis that has “best” distribution of objects
  - L = n/2, R = n/2 (ideal), where L→Left and R→Right
  - Realistically,
    - minimize (L-R) and
    - L approx. n/2, R approx. n/2

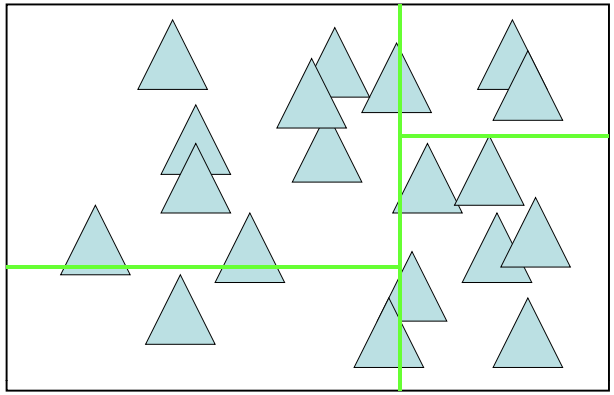
IRISA

kD-Trees



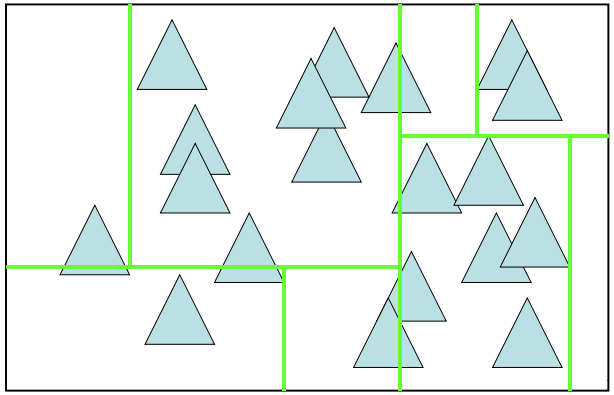
IRISA

kD-Trees



IRISA

kD-Trees



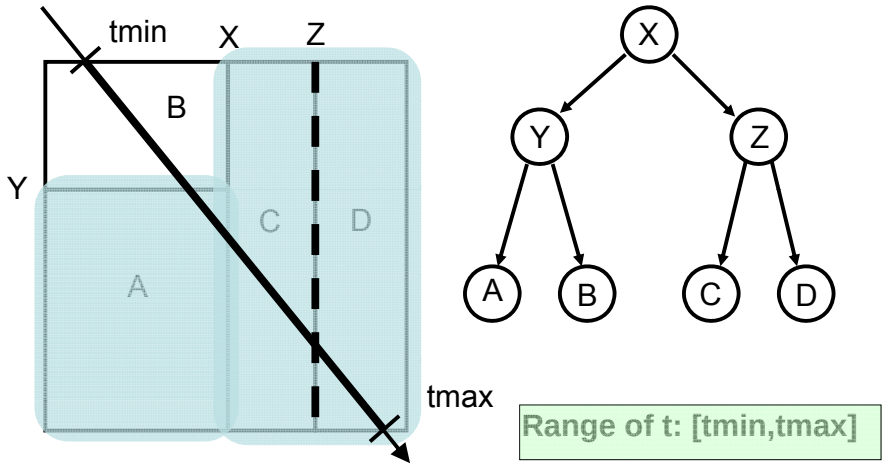
IRISA

kD-Trees: Data Structure

```
Struct KdTreeNode {
  int   axis;    // Both, x or y or z split plane (0,1,2), 3 for leaf
  float   value;      // Interior, split position x, y or z
  int  nPrims;  // Leaf
  Bounding_Box  bounds;
 KdTreeNode   *LeftChild;          // interior
 KdTreeNode   *RightChild;        // interior
}
```

IRISA

KD-Tree: Traversal



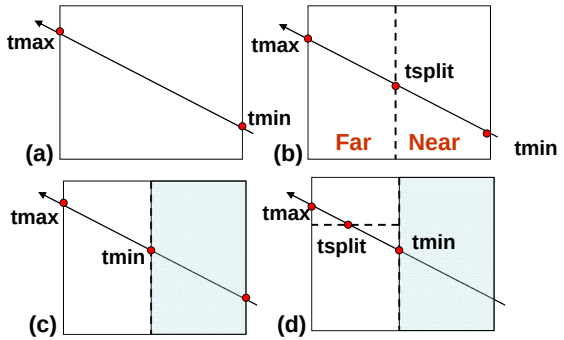
IRISA

KD-Tree: Traversal

- Input: a tree and a ray
- Search for the first intersected primitive in the tree
- Traversal: start from the root
- Use of a stack
- First range of t, [tmin,tmax]: associated with the scene bounding box
- Internal node encountered: ray is classified wrp to the splitting plane
  - If range lies entirely in one side of the plane, traversal moves to the appropriate child
  - If the range straddles the plane, traversal will continue to the first child hit by the ray while the second child is pushed onto the stack along with its range [tmin,tmax]
- Traversal proceeds down the tree, occasionally pushing items onto the stack, until a leaf node is reached.

IRISA

KD-Tree: Traversal



- (d) Sequence continues, processing tree nodes in depth first, front-to-back traversal, until closest intersection is found or the ray exists the tree

- (a) Initial parametric range [tmin, tmax] : intersection ray-bounding box
- (b) The ray first enters the child “near” which has [tmin,tsplit] as parametric range. If leaf then intersection, otherwise child nodes are processed
- (c) If no hit or a hit beyond [tmin,tsplit] then “far node” is processed

IRISA

KD-Tree: Traversal

```
kd-search( tree, ray )
        (global-tmin, global-tmax) = intersect( tree.bounds, ray )
{
search-node( tree.root, ray, global-tmin, global-tmax )
}

search-node( node, ray, tmin, tmax )
{
      if( node.is-leaf )
          search-leaf( node, ray, tmin, tmax )
      else
          search-split( node, ray, tmin, tmax )
}
```

IRISA

KD-Tree: Traversal

```
search-split( split, ray, tmin, tmax ) {
  a = split.axis
  thit = ( split.value - ray.origin[a] ) / ray.direction[a]
  (first, second) = order( ray.direction[a], split.left,split.right )
  if( thit >= tmax or thit < 0 )
    search-node( first, ray, tmin, tmax )
  else if( thit <= tmin )
          search-node( second, ray, tmin, tmax )
        else {
          stack.push( second, thit, tmax )
          search-node( first, ray, tmin, thit )
        }
}
```

IRISA

KD-Tree: Traversal

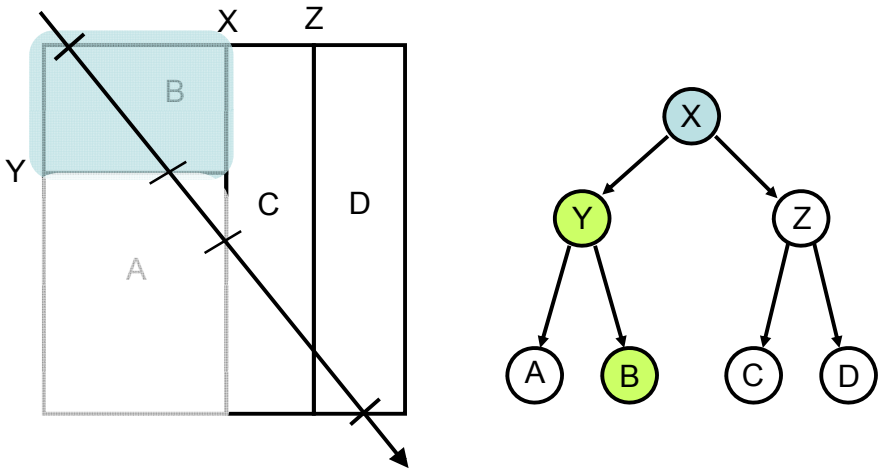
```
search-leaf( leaf, ray, tmin, tmax ) {
  // search for a hit in this leaf
  if( found-hit and hit.t < tmax )
      succeed( hit )
  else
      continue-search( leaf, ray, tmin, tmax )
}

continue-search( leaf, ray, tmin, tmax ){
  if( stack.is-empty )
      fail()
  else {
     (n, tmin, tmax) = stack.pop()
     search-node( n, ray, tmin, tmax )
  }
}
```

- **Remark**

If stack empty, then no intersection along the ray and the search terminates

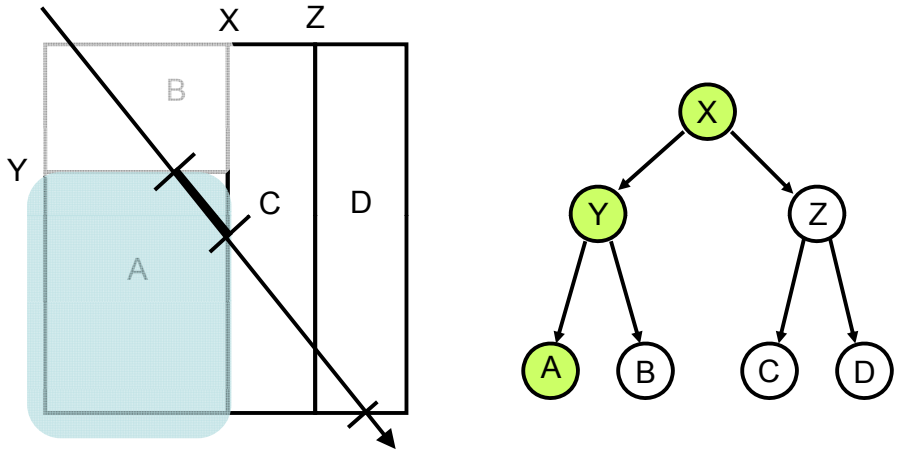
IRISA

KD-Tree Traversal



IRISA

Kd-tree traversal: Observation



Current leaf's tmax = Next leaf's min

IRISA

Kd-tree traversal: Observation

- Eliminate stack operations
- How?
  - If the traversal reaches a leaf and fails to find a hit:
    - Restart the search at the root
    - With tmin advanced to the end of the leaf
    - The first leaf intersected by the modified range is the next leaf that needs to be traversed

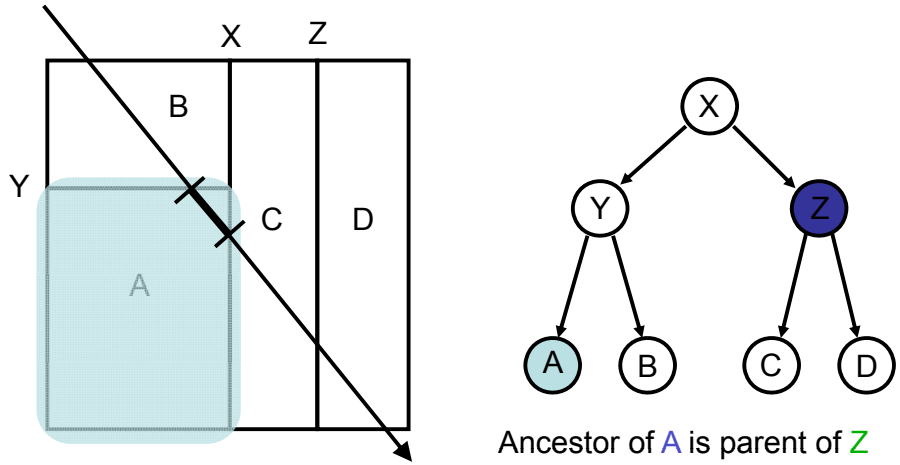
IRISA

Kd-tree traversal: Restart

```
continue-search( leaf, ray, tmin, tmax )
{
  if( tmax == global-tmax )
     fail()
  else {
     tmin = tmax
     tmax = global-tmax
     search-node( tree.root, ray, tmin, tmax )
   }
}
```

IRISA

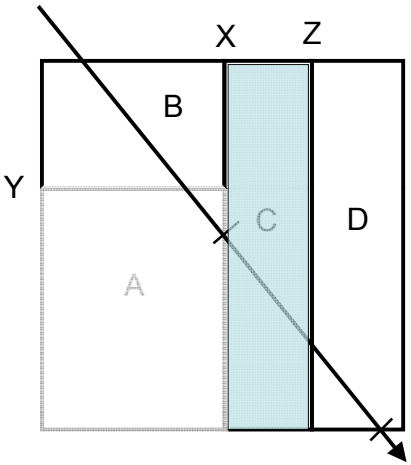
Observation

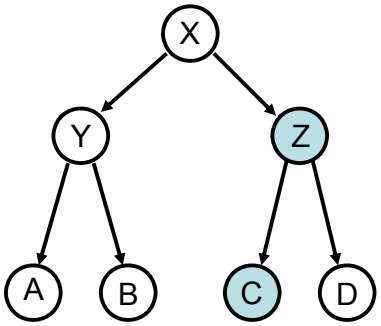


Ancestor of A is parent of Z

IRISA

Kd-tree: Observation





Ancestor of A is parent of Z

IRISA

Kd-tree: Backtrack

- In the traditional, a node pushed onto the stack is always the other (second) child of one of the current node's ancestors
- Thus, possible to reach the parent of the node atop the stack by following a chain of parent links (which we can store in the nodes of the tree) from the current node.
- If we again employ the tactic of advancing *tmin* to the end of the last leaf visited, then we will be able to recognize the appropriate parent as the closest ancestor that has a nonempty intersection with the remaining (*tmin*; *tmax*) range.
- Bounding boxes are stored with internal nodes
- Parents links are stored in all nodes
- Increase per-node storage

IRISA

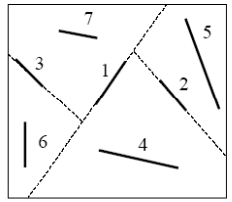
KD-Backtrack

```
continue-search( leaf, ray, tmin, tmax ) {
   if( tmax == global-tmax )
      fail()
   else  {
      tmin = tmax
      tmax = global-tmax
      backtrack( leaf.parent, ray, tmin, tmax )
    {
}

backtrack( split, ray, tmin, tmax ) {
   (t0,t1) = intersect( split.bounds, ray, tmin, tmax )
   if( no-intersection )
      backtrack( split.parent, ray, tmin, tmax )
   else
      search( split, ray, t0, t1 )
}
```

IRISA

BSP Tree

- Generalization of kd-trees
- Splitting plane is not axis aligned
- Used in games: DOOM



IRISA

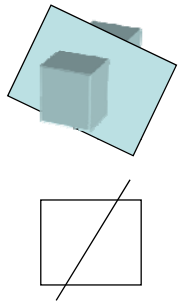
BSP tree

- A Binary Space Partitioning (BSP) tree data structure
  - Recursive, Hierarchical subdivision of *n-dimensional* space into convex subspaces.
- BSP tree construction
  - Partition a subspace by a hyper-plane that touches the edge of the subspace.
  - The result is two new subspaces that can be further partitioned by recursive application of the method.
- A "hyperplane" in an n-dimensional space is an n-1 dimensional object which can be used to divide the space into two half-spaces.

IRISA

BSP tree

- example:
  - In three dimensional space, the "hyperplane" is a plane.
  - In two dimensional space, it is a line.
- BSP trees are extremely versatile, because they are powerful sorting and classification structures.
  - Hidden surface removal
  - Ray tracing hierarchies
  - Solid modeling
  - Robot motion planning.
- Intensive time and space preprocessing vs. linear display algorithm.



IRISA

Building a BSP tree

- Given a set of polygons in three dimensional space, we would like to build a BSP tree which contains all of the polygons.
- The algorithm to build a BSP tree:
  - Select a partition plane.
  - Partition the set of polygons with the plane.
  - Recurse with each of the two new sets.
- The choice of partition plane depends on how the tree will be used, and what sort of efficiency criteria you have for the construction.

IRISA

Building a BSP tree

- For some purposes, it is appropriate to choose the partition plane from the input set of polygons
  - Scan-conversion
- Other applications may benefit more from axis aligned orthogonal partitions
  - Ray tracing
  - Space subdivision.
- It is desirable to have a balanced tree, where each leaf contains roughly the same number of polygons.
- It is desirable to minimize polygon splitting.
  - Finding the optimal split is hard, we use a heuristic
    - Testing the plane against a small random number of (5-6) polygons for split.

IRISA

BSP tree: Partitioning

- Classify each member of the set with respect to the plane.
- If a polygon lies entirely on one side of the hyper-plane
  - It is added to the partition set for the proper side.
- If a polygon spans the plane – keep in the node
- If the polygon intersect the hyper-plane
  - Split it as needed and add the parts the proper sets.

IRISA

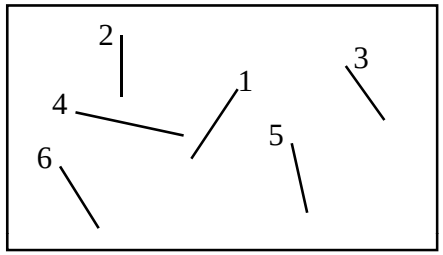
BSP tree: **When to stop?**

- The decision to terminate the tree construction is a matter of the specific application.
  - Some applications will benefit from termination when the number of polygons in a leaf node is below a maximum value.
  - Other methods continue until every polygon is placed in an internal node.
- Another criteria that can be used is the maximum tree depth.

IRISA

BSP tree: example

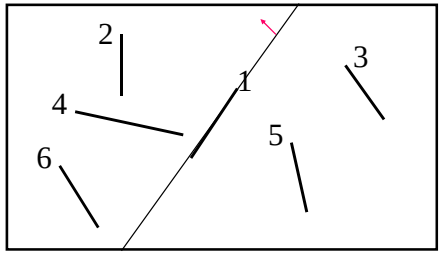
- One of the most important properties of BSP trees is that it is view independent.
- For example, consider the following case:



IRISA

BSP tree: example

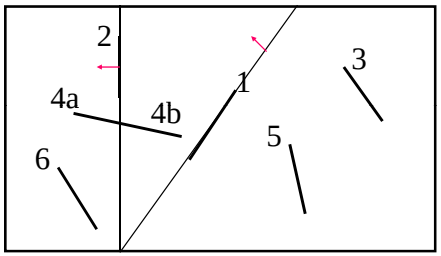
- Splitting the plane using the ordered lines from the input, we get the following:

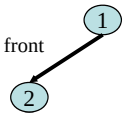


1

IRISA

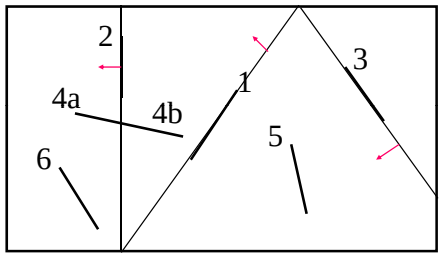
BSP tree: example

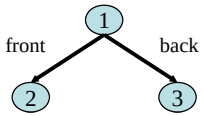




IRISA

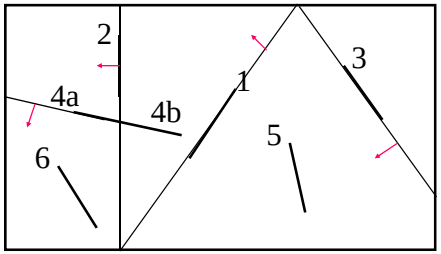
BSP tree: example

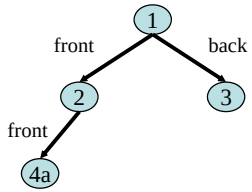




IRISA

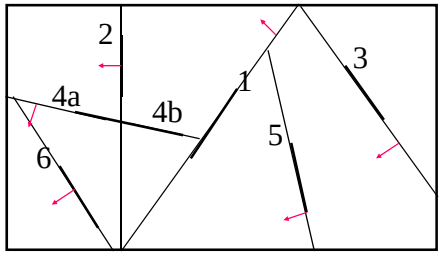
BSP tree: example

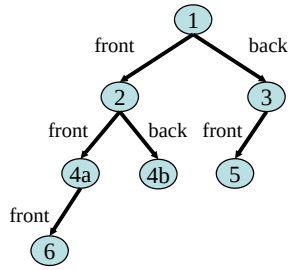




IRISA

BSP tree: example

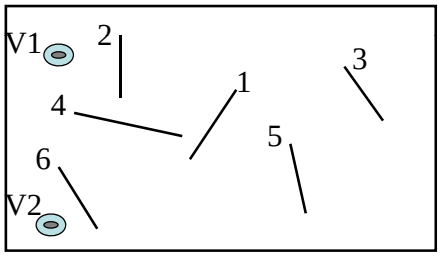


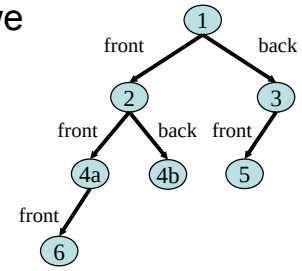


IRISA

BSP tree: example

- Now, we can choose several view points, and choose the painting order according to the tree we have created.





IRISA

BSP tree: Hidden Surface Removal

- Probably the most common application of BSP trees is hidden surface removal in three dimension.
- BSP trees provide an elegant, efficient method for sorting polygons via a depth first tree walk. This fact can be exploited in a back to front "*painter's algorithm*".
- The idea behind the *painter's algorithm* is to draw polygons far away from the eye first, followed by drawing those that are close to the eye.
- Hidden surfaces will be written over in the image as the surfaces that obscure them are drawn.
- Can assist in 3D clipping.
- Can support Back Face Culling.

IRISA

BSP tree: Painting

- One reason that BSP trees are so elegant for the painter's algorithm is that the splitting of complex polygons is an automatic part of tree construction.
- When building a BSP tree specifically for hidden surface removal, the partition planes are usually chosen from the input polygon set.
- However, any arbitrary plane can be used if there are no intersecting or concave polygons.

IRISA

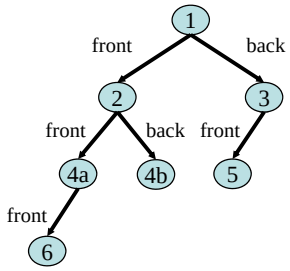
BSP tree: Drawing the scene

- To draw the contents of the tree:
  - Perform a back to front tree traversal.
  - Begin at the root node and classify the eye position with respect to the partition plane.
    - Draw the subtree at the far child from the eye
    - Draw the polygons in this node
    - Draw the near subtree.
  - Repeat this procedure recursively for each subtree.
- Front to back rendering is also possible.

IRISA

BSP tree: Hidden Surface Removal

- The painting order from V1:
  - 3, 5, 1, 4b, 2, 6, 4a
- The painting order from V2:
  - 3, 5, 1, 4b, 2, 4a, 6



IRISA

BSP tree: Ray Tracing

- Accelerating Ray Tracing
- Rectangular bounding volume of the scene: recursively subdivided
- Subdivision: Splitting planes are axis aligned
- Each splitting plane splits a cell into two equally sized sub-cells
- Choose x, y and z axis one at a time (alternate)

IRISA

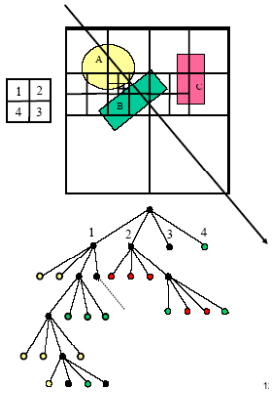
Octree

- Useful for reducing the number of ray-object intersections.
- The bounded 3D world to be ray traced is subdivided into cells of varying size. Each cell contains a list of objects (of approximately the same length) which intersect it.
- Given a ray to be traced, a list of cells intersected by the ray is determined. Intersection calculations are performed only with these objects.
- Furthermore, if the cells may be accessed in the order of advance of the ray, the procedure may terminate once the first intersection is discovered.

IRISA

Octree

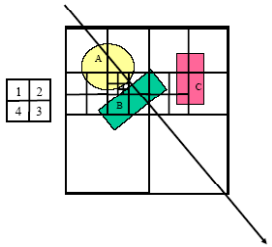
- Each node of the tree has eight children, corresponding to halving the space along all of the three axes.
- A node is a leaf if the subspace it represents intersects at most a given number of objects.
- The two basic operations needed for ray tracing octrees are:
  - Locating the leaf cell containing a given 3D point (point location).
  - Locating the next cell intersecting a given ray.
- The first is a standard octree traversal. The second is accomplished by repeating the first with a point along the ray just outside the current cell.



IRISA

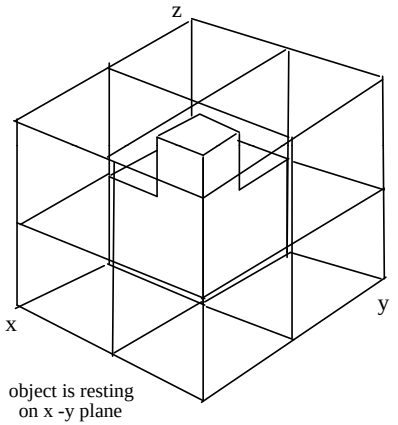
Octree

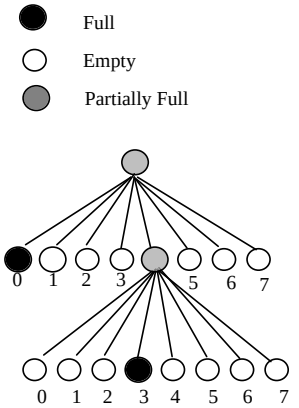
- Octrees ignore the *directionality* of objects.
- Subdivision is always in predefined directions and places.
- **Advantage:** Simple construction. Point location is easy.
- **Disadvantage:** Non-optimal subdivision (large trees).



IRISA

Octree: example





IRISA

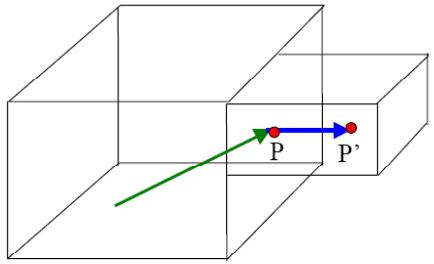
Octree: traversal

1. Determine the first intersection point **F** between the ray and the scene's axis aligned bounding box (SAABB)
2. Push **F** along the normal to the face containing it,
3. Pushing consists in adding to the P's coordinates a value deltax (resp. deltay, deltaz) which is equal to half the length of the x side (resp. y, z) of the smallest cell.
4. Search for the cell (containing **F**) in the tree
5. If no intersection in the cell, compute outgoing point **P**
6. Push **P** along the normal to the cell's face containing it
7. The results is another point **P'**
8. Search for the cell (containing **P'**) in the tree
9. Go to 1 until intersection

Remark: If **P** is on an edge or a vertex of a cell, push it simultaneously in the directions of the normals to the faces sharing it

IRISA

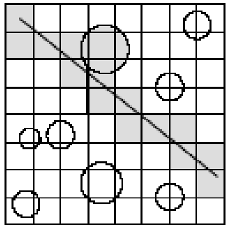
Octree: traversal

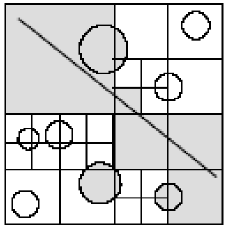


IRISA

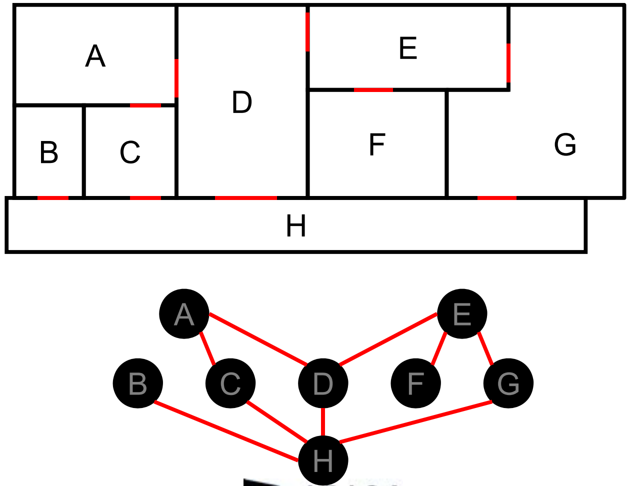
Uniform vs. Adaptive Subdivision

- Uniform: too much traversed empty cells
- Adaptive: less empty cells

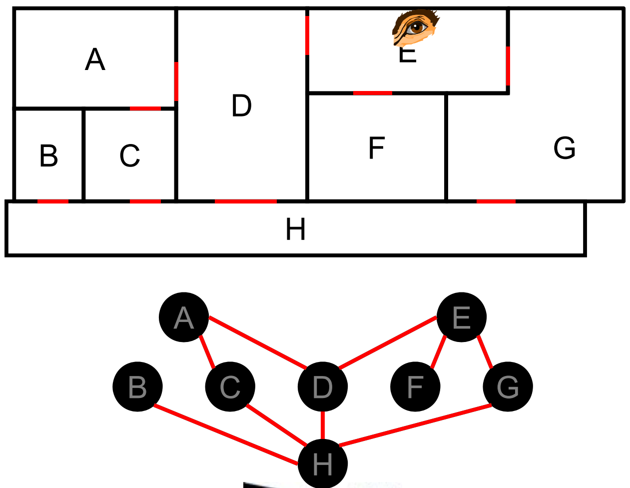




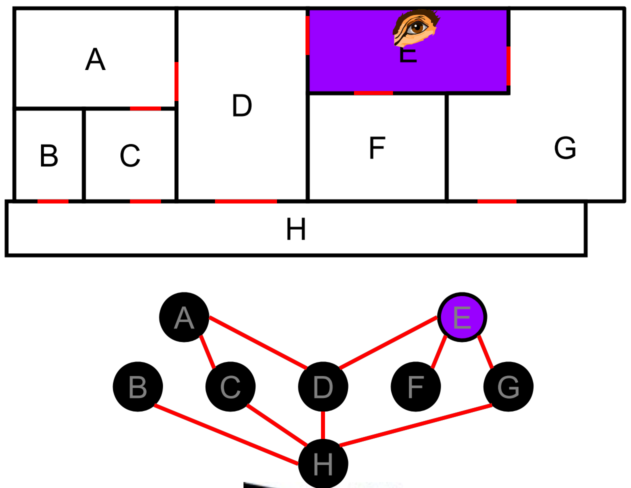
Cells & Portals



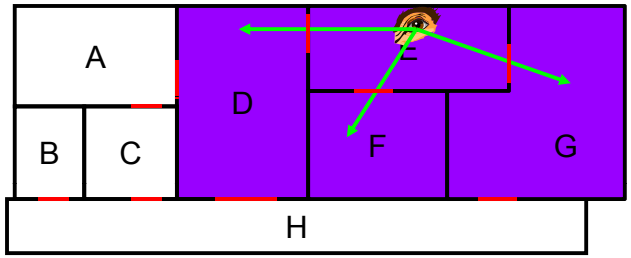
Cells & Portals

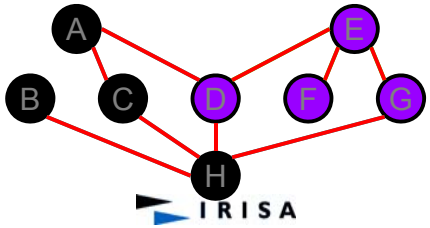


Cells & Portals



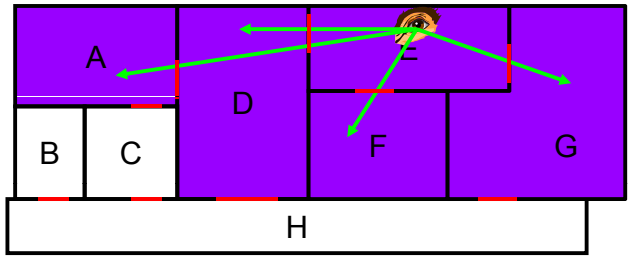
Cells & Portals

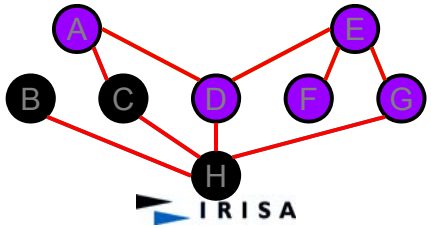




IRISA

Cells & Portals





IRISA

Cells & Portals

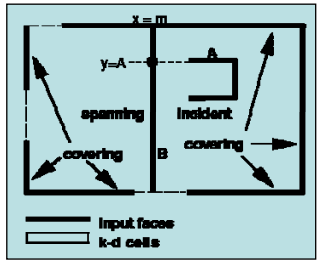
Teller and Sequin's Approach

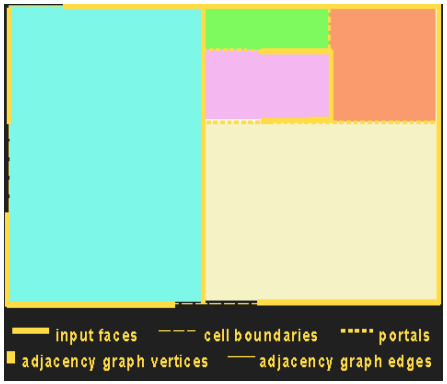
- Decompose space into convex cells: use walls as splitting polygons
- For each cell, identify its boundary edges into two sets: opaque or portal
- Precompute visibility among cells
- During viewing (eg, walkthrough phase), use the precomputed Potentially Visible polygon Set (PVS) of each cell to speed-up rendering

IRISA

Cells & Portals: Space Subdivision

Input Scene:

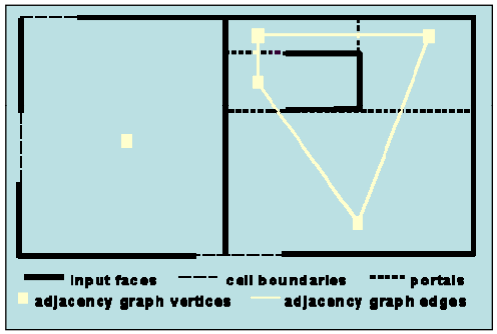




Spatial subdivision: Generated by computing a k-d tree of the input faces

IRISA

Determining Adjacent Information



IRISA

Finding Sightlines

Problem:

Given set of portals forming path from one portal to another, how do we determine if there is a sightline that passes through all those portals?

IRISA

Finding Sightlines

- **Key insight: orient portals**

- **All portal left end points must be on positive side of the line**

- **All portal right end points must be on negative side of the line**

IRISA

Finding Sightlines

- **Unknown line: S**
- **Constraints:**
  - **S . L >= 0 for all portal left points**
  - **S . R <= 0 for all portal right points**
- **This is a linear programming problem.**

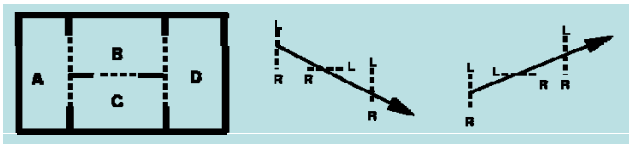
IRISA

Cell to Cell Visibility

- **Given a cell and the sightline algorithm, we need to find, for each cell, all cells visible from it**
- **Use adjacency lists to traverse graph depth First**
- **At each cell, recurse only if sightline test is positive**

IRISA

Computing the PVS of a cell



Linear programming problem:

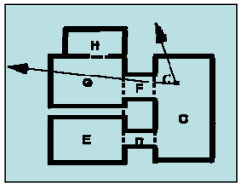
$$S \cdot R \geq 0, \quad \forall L \in L$$
$$S \cdot R \leq 0, \quad \forall R \in R$$

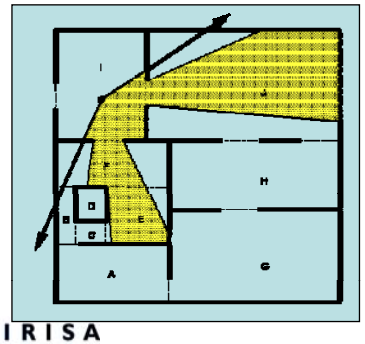
```
Find_Visible_Cells(cell C, portal sequence P, visible cell set V)
  V=V ∪ C
  for each neighbor N of C
    for each portal p connecting C and N
      orient p from C to N
      P' = P concatenate p
      if Stabbing_Line(P') exists then
        Find_Visible_Cells (N, P', V)
```

IRISA

Eye-to-Cell Visibility

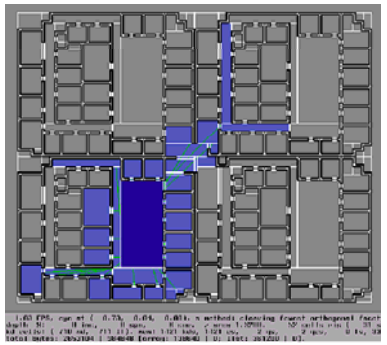
The eye-to-cell visibility of any observer is a subset of the cell-to-cell visibility for the cell containing the observer



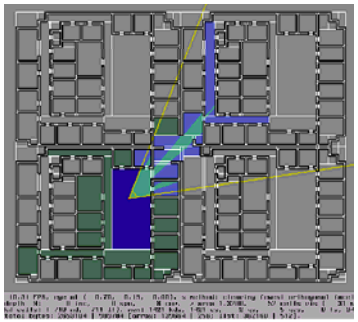


IRISA

Results



(a) A source cell (dark blue), its cell-to-cell visibility (light blue), and stabbing lines (green).



(d) The same observer, with a 60° view cone. The eye-to-cell visibility is shown in blue; the exact visible area is shown in blue-green. The green cells have been dynamically culled.

IRISA