LA METHODE DU LANCER

DE RAYON

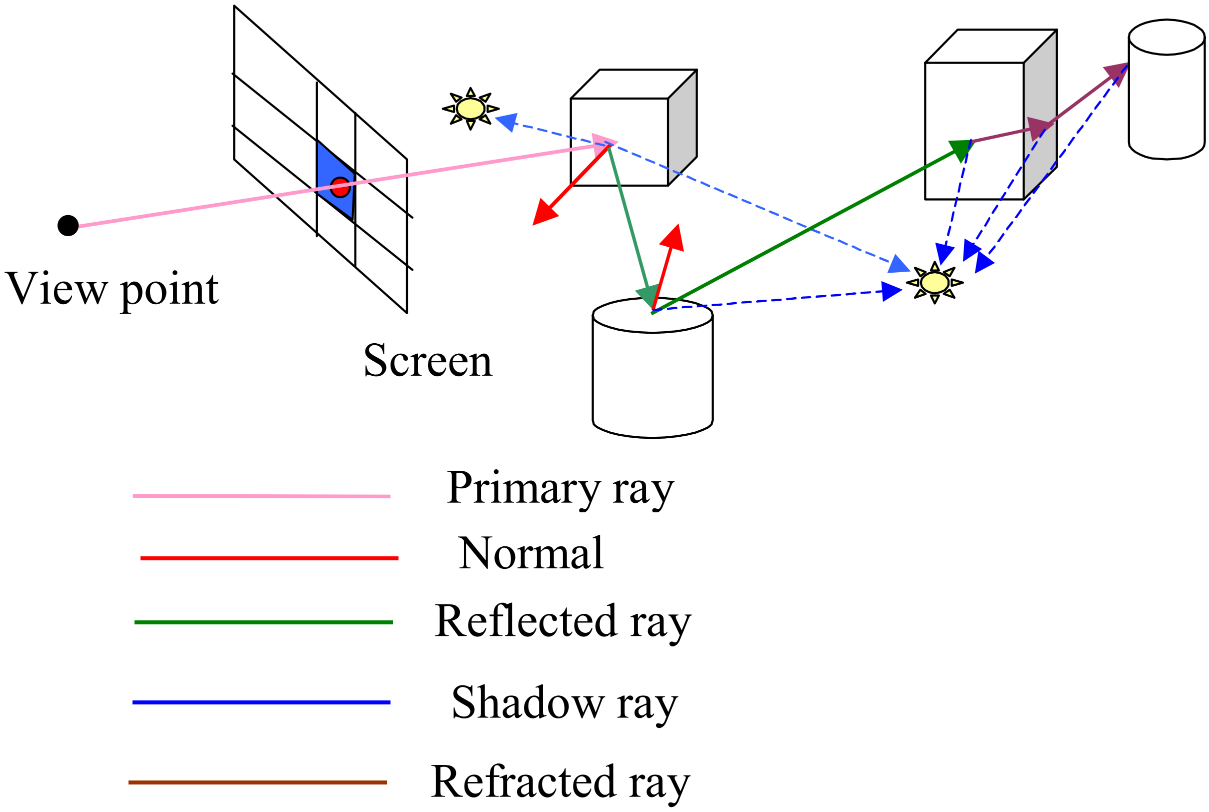
Kadi BOUATOUCH

En anglais :

RAY TRACING

RAY CASTING

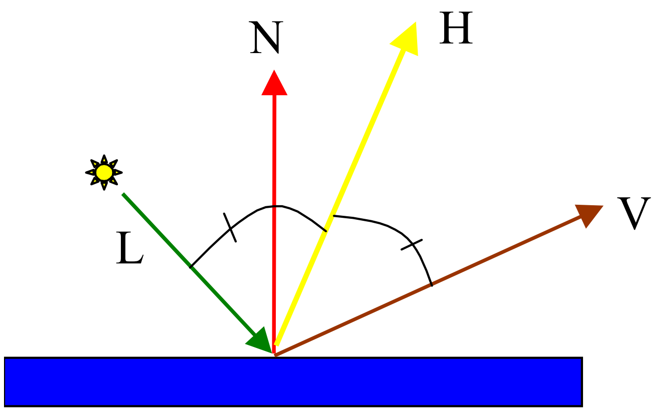
PRINCIPE



- 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

THE ILLUMINATION MODEL

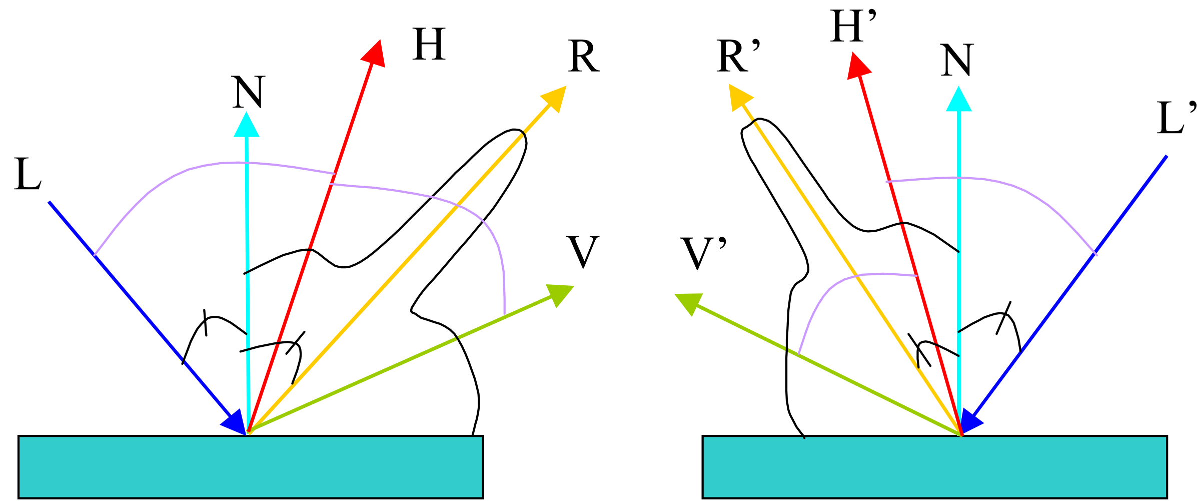
The specular reflection



- $Ir = ks . <N,H>^{n} . I_{source}$
- If the surface is perfectly specular n is very large
- $<N,H>^{n}$ 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

THE ILLUMINATION MODEL

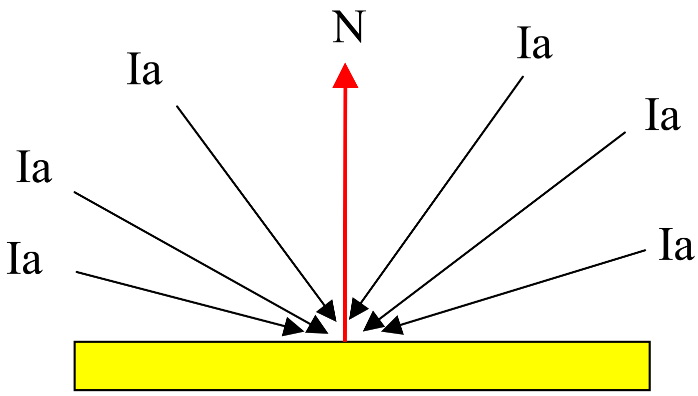
Reciprocity of the reflection model



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

THE ILLUMINATION MODEL

Ambient term



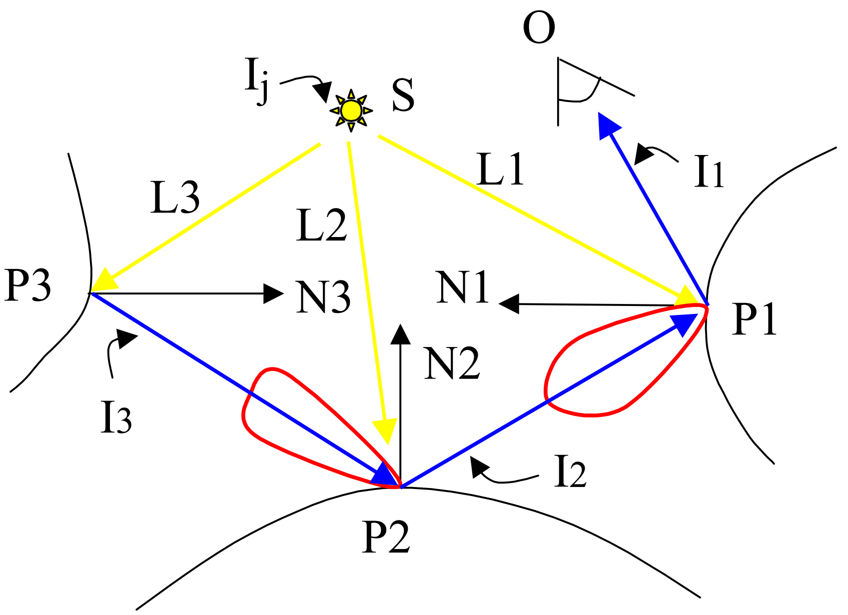
- The indirect diffuse component I_{id} due to multiple reflections is supposed to be the result of the diffuse reflection of an ambient term Ia

- Ia is the same for all the surfaces

- $I_{id} = kd. I_{obj} . Ia$

THE ILLUMINATION MODEL

The different components



- H1 : bisecting line of angle S P3 P2
- H2 : bisecting line of angle S P2 P1
- H1 : bisecting line of angle S P1 O
- Ida_i : intensity due to direct lighting and the ambient term for point Pi
- $Ida_i = kd_i \,.\, C_i \,.\, Ia$
  $+ kd_i \,.\, C_i \,.\, Is \,.\, \cos(Li,Ni)$
  $+ ks_i \,.\, Is \,.\, \cos(Ni,Hi)^n$

- $I_3 = Ida_3$

- $I_2 = Ida_2 + ks_2 \,.\, I_3$

- $I_1 = Ida_1 + ks_1 \,.\, I_2$

THE ILLUMINATION MODEL

The complete illumination model

- Legend :
    - dd : direct diffuse
    - ds : direct specular
    - is : indirect specular
    - t : transmitted or refracted
    - I_{ref} : intensity carried by the reflected ray
    - I_{tran} : intensity carried by the refracted ray
    - C_i : object's color
    - I_j : intensity of light source j
- $I = I_{amb} + I_{dd} + I_{ds} + I_{is} + I_t$
- $I_{amb} = kd. C_i . Ia$
- $I_{dd} = kd_i . C_i . \Sigma_j I_j . \cos(Li,Ni)$
- $I_{ds} = ks. \Sigma_j I_j. \cos(Ni,Hi)^n$
- $I_{is} = ks . I_{ref}$
- $I_t = kt . I_{tran}$
- For a material : kd + ks + kt = 1

THE ILLUMINATION MODEL

The illumination algorithm

- Ray r : equation $P = P_0 + t \, . \, D$

- $I(r) = I_{amb}$ (inter(r,Scene))

  $+ \ \Sigma_j \ I_{dd}$ (j, inter(r,Scene))

  $+ \ \Sigma_j \ I_{ds}$ (j, inter(r,Scene))

  + ks . I (reflected_ray)

  + kt . I (refracted_ray)

- I(r) : recursive function calculating the global intensity brought by a ray r

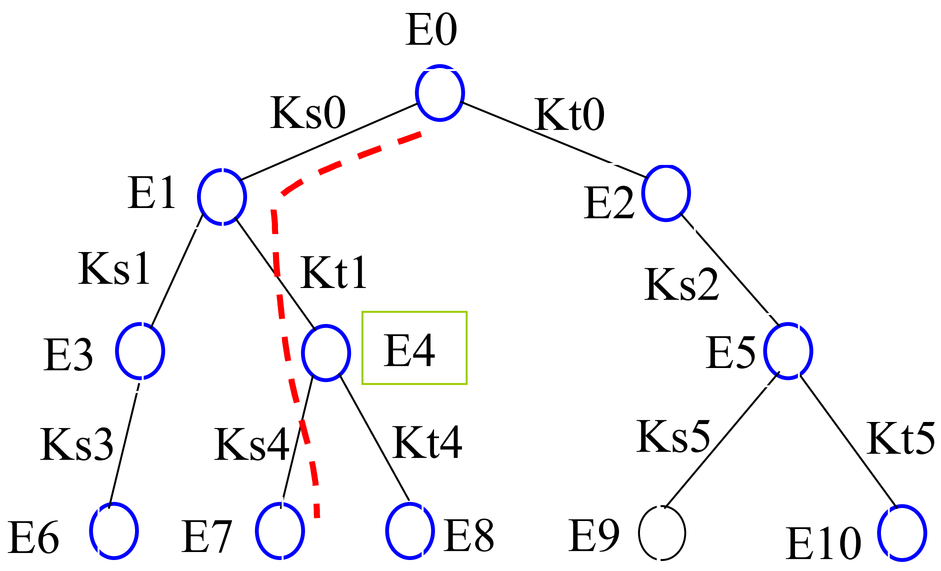
- I_{amb} , I_{dd} and I_{ds} are functions computing the ambient, direct-diffuse and direct-specular components respectively

- Scene : data structure representing the scene

- Each source is indexed by j

THE ILLUMINATION MODEL

How to stop tracing rays ?



- I : Intensity due to this ray path :

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

$$= Ks0 \,.\, Kt1 \,.\, Ks4 \,.\, E7 + Ks0 \,.\, .Kt1 \,.\, E4 + Ks0 \,.\, E1$$

- Ei : intensities due to the light sources ; direct lighting

- Stop tracing rays when the cumulative product is below a certain threshold

Intersection computation : principle

- The scene is supposed to be expressed in the world coordinate system (WCS).

- It may be: A set of independent objects

- An object may be a CSG tree (Constructive Solid Geometry) which is a binary tree whose leaves are primitive objects like sphere, cylinder, cone and whose nodes are boolean operators like union, intersection and difference.

- The purpose is to intersect a scene by a ray whose equation is given by :

$$P = P_0 + t \cdot D$$

- where :
  P_0 is the ray origin ;

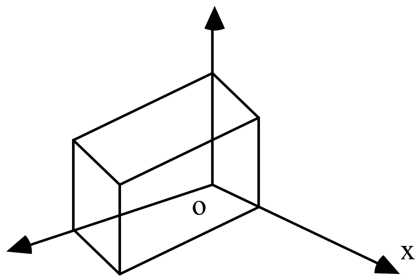
  $D = (dx, dy, dz)$ is the direction vector of the ray ;

- $t > 0$

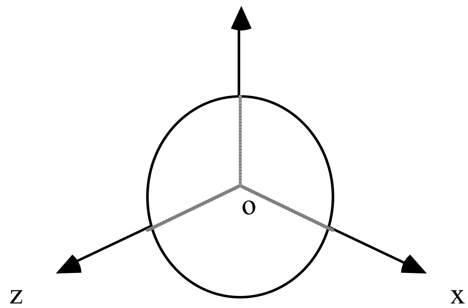
- Intersection result = $\{ t_i \,/\, t_i$ is a value of t corresponding to an intersection point $\}$.

- Only the closest point to the ray origin, is used to compute the lights contribution and to shoot secondary rays.

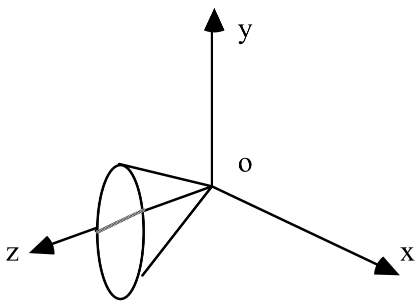
- To simplify the intersection computing, each object may described in a local coordinates system (LCS)



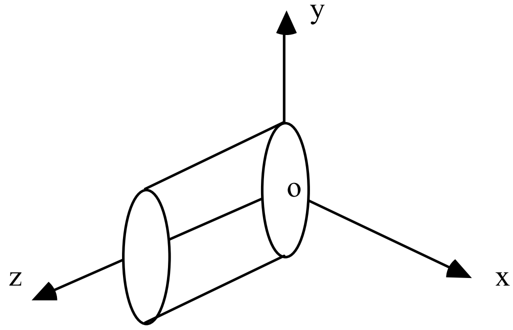
parallelepiped



sphere



cone



cylinder

- In this case two transformation matrices are then associated with each object :

- the first one allows the transformation of a point in the WCS to a point in the local coordinates system,

- the second one allows the inverse transformation.

- Ray-object intersection is performed in the LCS. With this aim in view, the ray is transformed into the LCS.

- This simplifies both the computations of the ray-object intersection and that of the normal.

- Since t is a scalar, its value is not affected by this transformation.

- To compute the closest intersection point, the smallest value of t is substituted in the ray equation expressed in the WCS. The transformation LCS-WCS is then not necessary.

- As for the normal calculation, it is performed on the LCS, then it is transformed onto the WCS.

Bounding volumes

- 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.

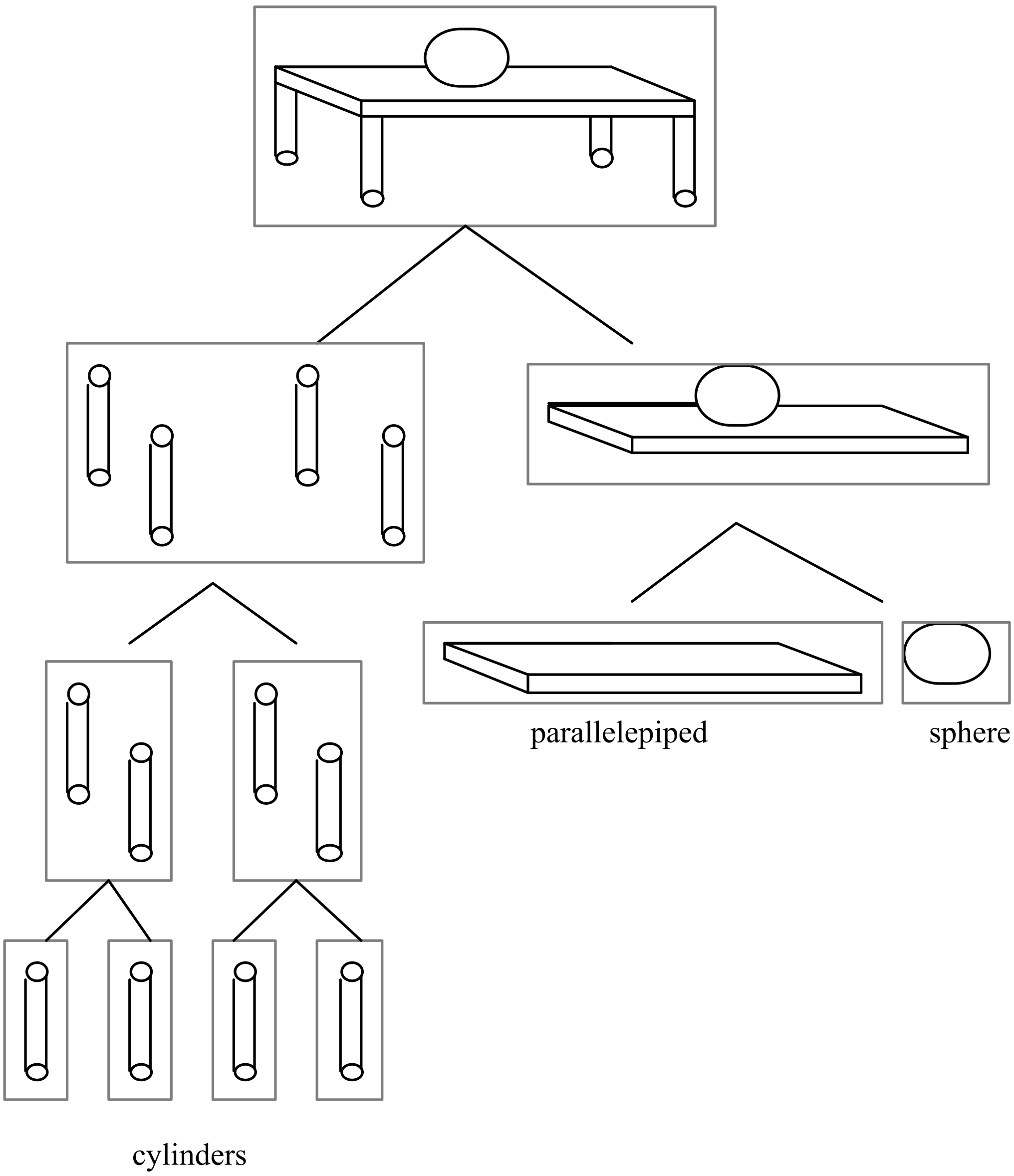
- Bounding volumes are simple geometric objects which fit around the objects.

- They are chosen to be simple to intersect with a ray, such as spheres or parallelepipeds that have faces perpendicular to the axes.

- The building of this hierarchy consists in picking some of these bounding volumes and surrounding them with another bounding volume. This process is repeated recursively until a bounding volume is generated that surrounds the whole scene.

Bounding volumes

Example of hierarchy of bounding volumes : binary tree.



Bounding volumes

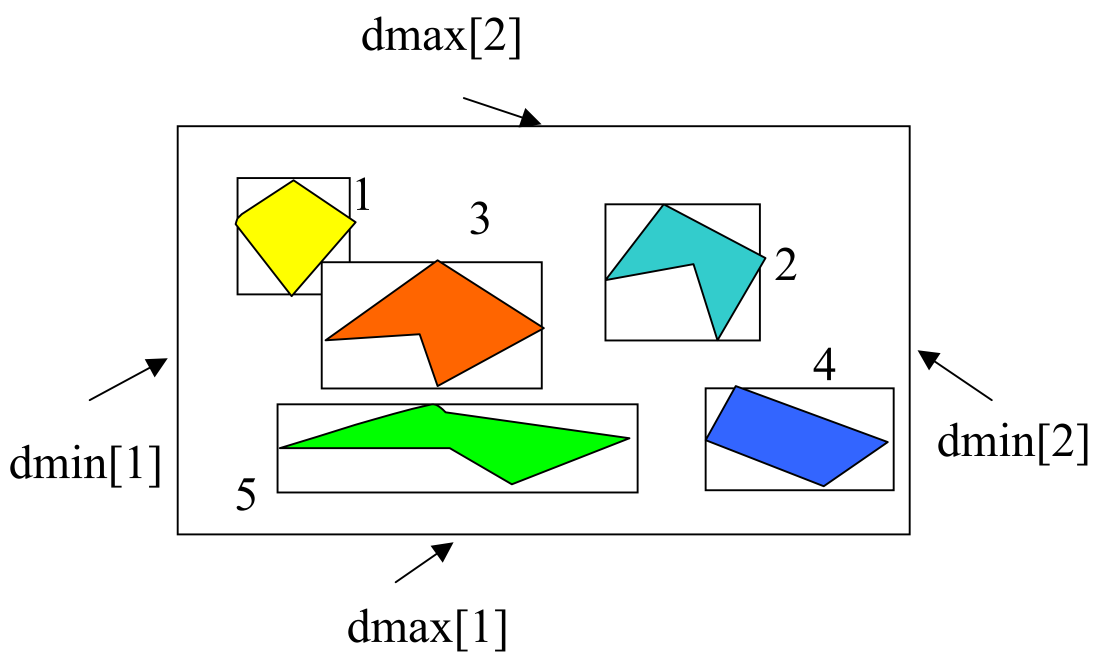
Hierarchy

- It is very important to find a way to choose a tree that reduces the rendering time.

- Trying to construct manually a tree is very tedious and not efficient.

- A better method consists in dividing the scene into halves along one axis and surround each half with a bounding volume. This process is applied recursively on each half.

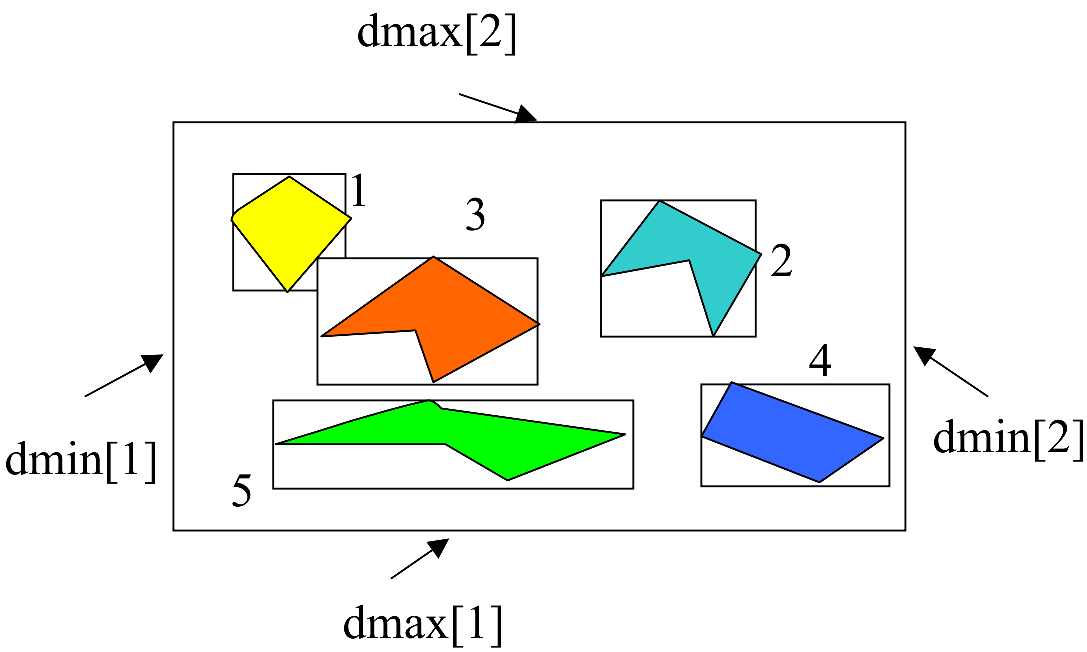
Median cut method



Bounding volumes

Hierarchy : Median cut method

1. Search for max slab
2. L= {liste of bounding volume numbers}
3. dmax[2] – dmin[2] or dmax[1] – dmin[1]
4. In this example : max = dmax[1] – dmin[1]
5. Then choose slab 1
6. Sort the bounding volumes with respect to increasing dmin[1]
7. We get a sorted list L = {1,5,3,2,4}
8. Spit L into two sub-lists L1 and L2
9. We get : L1 = {1,5,3}   L2 = {2,4}
10. Go to 1 with L = L1 then L = L2



Bounding volumes

Hierarchy: Median cut

Data structures

TYPE

```
-  ttab_ptr_obj = array|1..Nb_obj] of integer ;
-  tvol_engl = struct { /* bounding volume type  */
                    dmin : arrray[1..N_Slab] of real ;
                    dmax : array[1..N_Slab] of real ;
                  }
-  tengl_obj = struct {  /* hierarchy node */
                    tab : ttab_ptr_obj ;
                    eng : tvol_engl ;
                    number : integer;
                  }
-  obj = struct {
                    vol_eng : tvol_engl ;
                    par_geo : tparam_geom ;
                    par_photo : tparam_photo ;
                    }
-  ttab_obj = array[1..Nb_obj] of object ;
-  tHier = array[1..Nmax] of tengl_obj ;
```

VAR

```
-  tab_obj : ttab_obj ;
-  Hier : tHier ;
-  tabp : ttab_ptr_obj
```

Bounding volumes

Hierarchy Median cut : Algorithm

```
Procedure create_Hierarchy(tabp : ttab_ptr_obj ;
                    ind_beg, ind_end, depth : integer) ;
begin
/* Compute bounding volume for tabp */
/* Result : 2 arrays dmin and dmax */
bounding_vol(tabp, dmin, dmax, ind_beg,ind_end) ;
/* Hierarchy saved as a bin tree in an array Hier */
for i := ind_beg to ind_end {
  Hier[depth].tab[i-ind_deb+1] := tabp[i] ;
}
Hier[depth].eng.dmin := dmin ;
Hier[depth].eng.dmax := dmax ;
Hier[depth].number := ind_end – ind_beg + 1 ;
/* Stop splitting the list if the number of leaf' objects is
smaller than Max_obj */
if (ind_end – ind_beg – 1) <= Max_obj { return  };
index := 1 ;
d_partition := Huge_Negative_Number ;
for i :=1 to Nb_Slab {
      if (dmax[i] – dmin[i])  > d_partition
         { d_partition := dmax[i] – dmin[i] ; index := i }
}
/* List sorting with respect to increasing dmin[index] */
quick_sort_wrt_dmin(tabp, index) ;
m := ind_beg + n ;
create_Hierarchy(tabp, ind_beg, m, 2*depth) ;
create_Hierarchy(tabp, m + 1, ind_end, 2*depth + 1) ;
end
```

Bounding volumes

Hierarchy : Goldsmith’s et al.’s method

- Interesting method: proposed by Goldsmith and Salmon .

- The used strategy is a heuristic tree search

- Objects are added successively and the tree is searched to find a suitable insertion point for each new node.

- Since not all nodes of the tree can be considered as a point for insertion, the search must follow only few paths.

- The choice of sub-trees to search from a given node is determined by the smallest increase in surface area of the node's bounding volume that would occur if the new node was to be inserted as a child of it.

- During the search, at each level of the tree, the new node is considered as a prospective child of each node that will be searched.

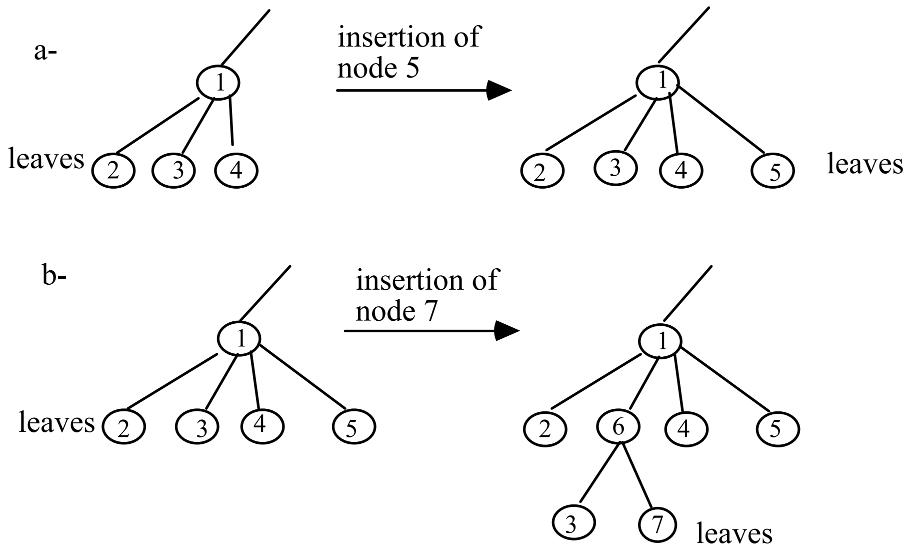
- The tree is evaluated with the proposed insertion and the location with the smallest increase in tree cost is saved.

- When the search reaches a leaf node, the new node and the leaf node are proposed as children of a new non leaf node.
- Bottom-up evaluation after each insertion of a new node

Bounding volumes

Hierarchy : Goldsmith's et al.'s method

Example of hierarchy



Bounding volumes

Hierarchy : CSG Model

Case of CSG tree modelled scenes

- Data structure of each leaf of the CSG tree is extended by adding to it the bounding volume of the leaf.

- Bottom-up search of the tree in order to compute the bounding volumes of the non leaf nodes.

- These bounding volumes are in their turn added to the data structure of the associated nodes.

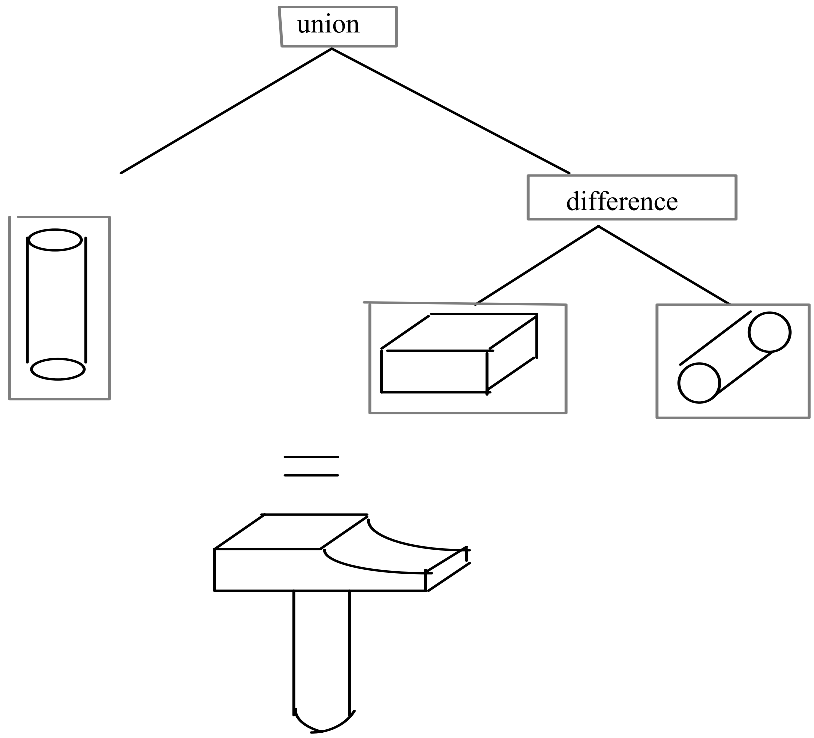
- Their evaluation depends on the boolean operator associated with the nodes.

- The bounding volume of the root bounds the whole scene.

Bounding volumes

Hierarchy : CSG Model

Case of CSG tree modelled scenes



Ray-scene intersection test

using the hierarchy

- 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.

Different kinds of bounding volumes

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.

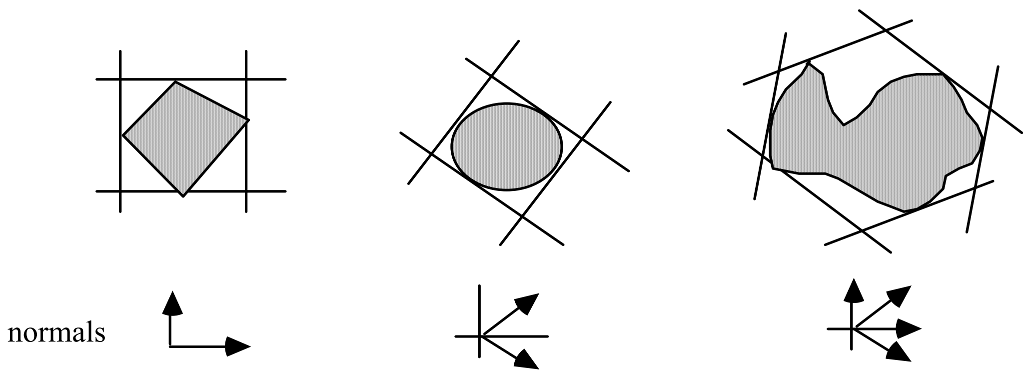
Different kinds of bounding volumes

Polyhedron : Intersection of Slabs

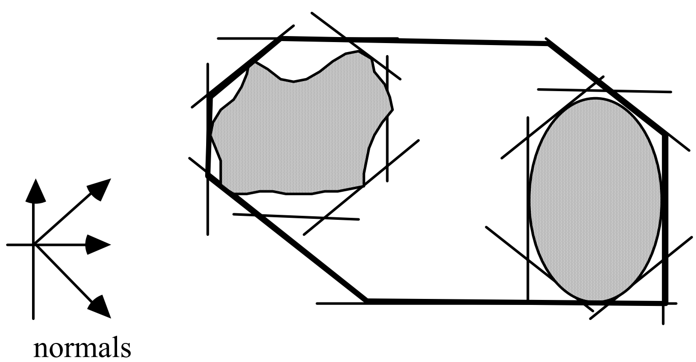
- The objects are bounded by polyhedra whose sizes may be different but whose faces' normals have constant direction vectors.

- These direction vectors as well as the number of faces are chosen by the user before the synthesis phase.

- Example of polyhedral bounding volumes.



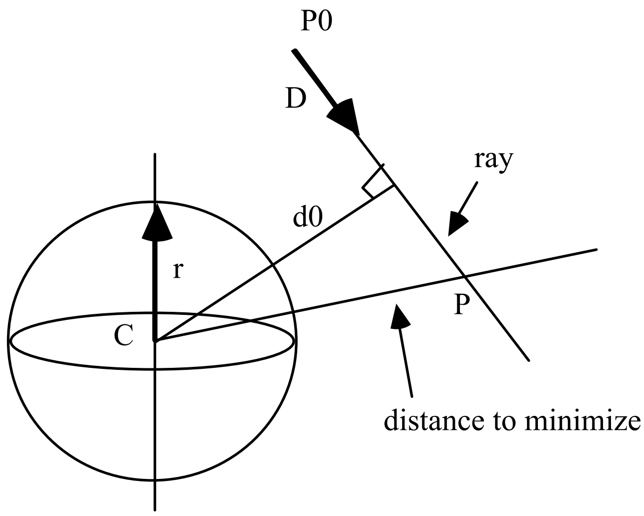
- It is easy to build a hierarchy with polyhedral bounding volumes.



Intersection Test

Sphere

- Orthogonal distance d_0^2 between the center of the sphere and the ray
- If d_0^2 is smaller than or equal to the square of the radius of the sphere, then the ray intersects the sphere, otherwise it does not intersect it



- Let C be the center of the sphere and let $P = P_0 + t \,.\, D$ be the ray equation. d_0 is evaluated by minimizing the distance between C and a point P on the ray.
- This gives

$$d^2 = \| P_0 + t \,.\, D - C \|^2 = \| P_0 - C \|^2 + 2t \,.\, (P_0 - C) \,.\, D + t^2 \,.\, \| D \|^2$$

- By setting to 0 the derivative of d^2, we obtain :

$$t = ((P_0 - C) \,.\, D \;/ \| D \|^2) = - (P_0 - C) \,.\, D$$

- After substitution : $d_0^2 = \| P_0 - C \|^2 - ((P - C) \,.\, D)^2$

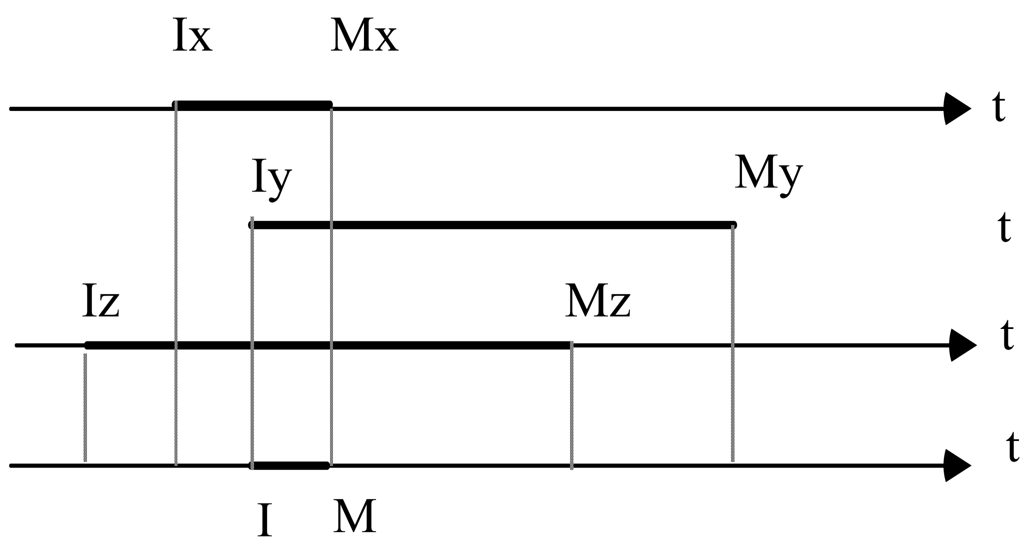
Intersection Test

Parallelepiped

- The faces of the parallelepiped are 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.
- 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 parallelepipedic bounding volume, otherwise it does not intersect it



Intersection Test

Polyhedron

- The intersection test is similar to the previous one, except that the faces are not perpendicular to the axes of the eye coordinates system

- Interval : [I, M]

- Let N be the normal of a face

- N . P + d = 0 the equation of the plane containing the face.

- The value of t corresponding to the intersection between the ray and this face is computed by substituting the ray equation in that of the plane : $t = -(d + N \cdot P_0) / N \cdot D$
- For a 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 object bounding volumes

Intersection with simple objects

Sphere

- Intersection points : solutions of the following equation

$$\| P_0 - C \|^2 + 2t \,.\, (P_0 - C) \,.\, D + t^2 \,.\, \| D \|^2 = r^2$$

- Intersection : performed in the local coordinates system of the sphere
- $\| P_0 \|^2 + 2t \,.\, P_0 \,.\, D + t^2 \,.\, \| D \|^2 = r^2$

Parallelepiped

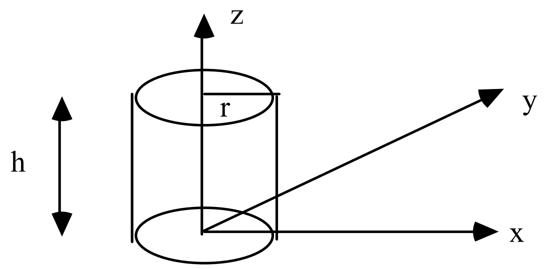
- The way to compute the ray-parallelepiped intersection has been shown previously.

- [I, M] : interval intersection.

- If I <= M then the intersection exists and in addition, I and M are the values of the parameter t corresponding to the intersection points. Otherwise it does not exist.

Intersection with simple objects

Cylinder



cylinder and its LCS.

- The cylinder is supposed to be the result of the 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).

Intersection with simple objects

Cylinder

- 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 :

$$t^2 . (dx^2 + dy^2) + 2t . (x0 . dx + y0 . dy) + (x0^2 + y0^2 - r^2) = 0$$

  - 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 :

$$A = - z0 / dz \quad \text{and} \quad B = (h - z0) / z0$$

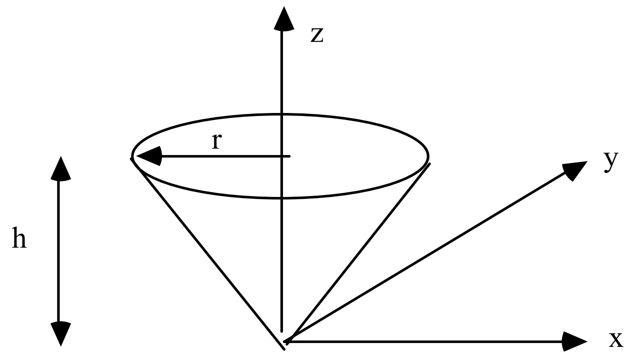
  - We get :

$$t3 = \min(A,B) \quad \text{and} \quad t4 = \max(A, B)$$

Intersection with simple objects

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 $z = h$.

- The intersection between the ray and the infinite height cone is first performed. The equation of this cone is given by :

$$h^2 . (x^2 + y^2) - r^2 . z^2 = 0.$$

- 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 :

$$t3 = \min(A, B) \quad \text{and} \quad t4 = \max(A, B)$$

- where $A = - z0 / dz$ and $B = (h - z0) / dz$.
- The final interval is the combination of these two intervals (as for the cylinder).

Intersection with simple objects

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

  - *Inside - Outside test*

    - the second step tests if the resulting point is inside or outside the polygon.

Intersection with simple objects

Polygon

Snyder's method

- Snyder's method method concerns the ray-triangle intersection. It will be extended to a polygon.

- Let P_i be the vertices of a triangle and let N_i the associated normals which are used for normal interpolation across the triangle.

- Normal to the triangle: $N = (P_1 - P_0) \times (P_2 - P_0)$

- A point P lying on the triangle plane satisfies :

  $$P . N + d = 0 \text{where} \quad d = - P_0 . N.$$

- An index i_0 is computed to be equal either to 0 if $| N_x |$ is maximum or to 1 if $| N_y |$ is maximum or to 2 if $| N_z |$ is maximum.

- 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 :

  $$t = (d - N . O) / N . D.$$

Intersection with simple objects

Polygon

Snyder's method

- Let i_1 and i_2 ($i_1, i_2 \in \{0, 1, 2\}$) be two unequal indices different from i_0. Compute the i_1 and i_2 components of the intersection point I, by :

$$I_{i1} = O_{i1} + t \, . \, D_{i1} \quad \text{and} \quad I_{i2} = O_{i2} + t \, . \, D_{i2}$$

- The inside-outside test can be performed by computing scalars β_0, β_1 and β_2 according to :

$$\beta_i = [(P_{i+2} - P_{i+1}) \times (I - P_{i+1})]_{i0} \;/\; [N]_{i0}$$

- The β_i are the barycentric coordinates of the point where the ray intersects the triangle plane.

- I is inside the triangle if and only if $0 \leq \beta \leq 1$ for $i \in \{0, 1, 2\}$.

- The interpolated normal at point I is given by :

$$N' = \beta_0 \, . \, N_0 + \beta_1 \, . \, N_1 + \beta_2 \, . \, N_2 .$$

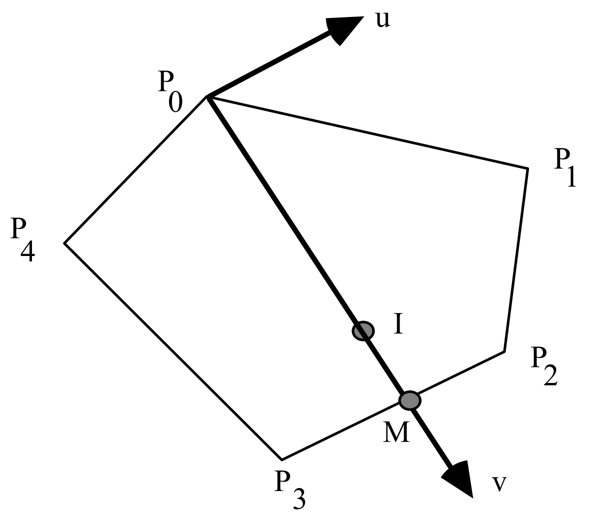
- Snyder's method can be easily extended up to polygons.

- The main idea is to consider a polygon as a union of triangles.

Intersection with simple objects

Polygon

Marchal's method



- I is the ray-plane intersection point.

- The P_i 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 with simple objects

Polygon

Marchal's method

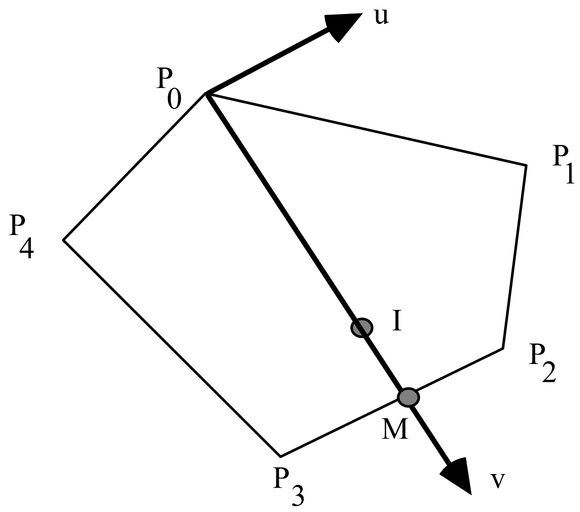
- The interpolated normal at point I is given by :

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

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

$$N_M = (P_iM / P_iP_{i+1}) . N_{i+1} + (1 - P_iM / P_iP_{i+1}) . N_i$$

- 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.



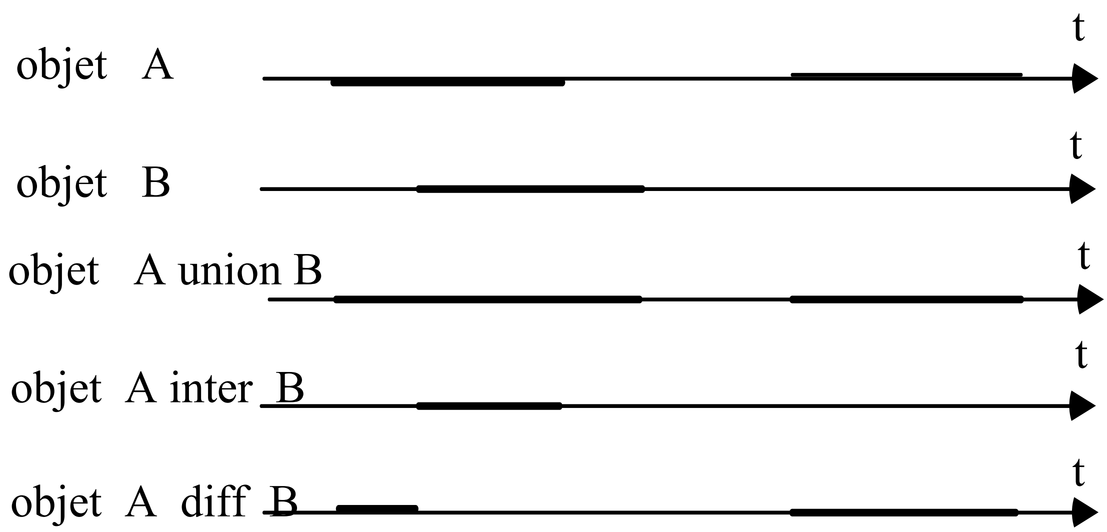
Composite objects

- A composite object may be created by performing set operations (union, difference, intersection) on simple or on other composite objects.

- A CSG tree is an example of composite object.

- The ray-object intersection results in a list of intervals as shown in the following figure .

- In this example, two objects are combined with each set operator. The intersection result is a list of two intervals, the length of which depends on the used set operation.



.

Intersection with algebraic surfaces

- An algebraic surface is defined by :

$$S(x, y, z) = \sum_{i=0}^{l} \sum_{j=0}^{m} \sum_{k=0}^{n} a_{ij} \, . \, x_i \, y_j \, z_k \qquad (6)$$

- The substitution in S(x, y, z) of the ray equation, gives a polynomial equation S*(t), the degree of which is
  d = l + m +n :

$$S^*(t) = \sum_{i=0}^{d} a_i \, . \, t^i \, .$$

- S*(t) may be solved with non linear programming techniques, such as the one of Laguerre, Newton or Bairstow.

- Thes techniques are iterative and converge only if they start from an initial value of t close to the exact root.

- To find a good initial value of t, one must isolate the roots by recursively subdividing the range of t into two equal sized subintervals, and by seeing if the resulting subintervals contain at least one root.

- This process terminates when the width of an interval is less than a given threshold.

Root isolation methods

Several root isolation methods are proposed in the literature. Only two of them are discussed :

- interval methods

- Collins's method

Interval method

- An interval is defined by an ordered pair of real numbers
  [a, b] with $a < b$.
- Interval method allows performing arithmetic operations on intervals using the operators +, -, * and / .
- Let op be an operator :

$$[a, b] \text{ op } [c, d] = \{ x \text{ op } y \text{ , such that } x \in [a, b] \text{ and } y \in [c, d] \}$$

- These operations can be performed algebraicly using the endpoints of the intervals, as shown in the following :

$$[a, b] + [c, d] = [a + c, b + d]$$

$$[a, b] - [c, d] = [a - c, b - d]$$

$$[a, b] * [c, d] = [\min(a*c, a*d, b*c, b*d), \max(a*c, a*d, b*c, b*d)]$$

$$[a, b] / [c, d] = [a, b] * [1/d, 1/c] \text{ provided that } 0 \notin [c, d]$$

- The division by an interval containing 0 may be defined as :

$$1 / [a, b] = [1/b, +\infty] \text{ if } a = 0,$$

$$= [-\infty, 1/a] \text{ if } b = 0,$$

$$= [-\infty, 1/a] \text{ union } [1/b, +\infty] \text{ if } a \leq 0 \leq b,$$

$$= [1/b, 1/a] \text{ if } a > 0 \text{ or if } b < 0 .$$

Interval method

- Let $f(x_1,\ldots, x_n)$ be a rational function, and let F be the corresponding interval rational function.
- If for each i, $1 \le i \le n$, x_i ranges over $[a_i, b_i]$ then

$$F([a_1, b_1],\ldots, [a_n, b_n]) \supseteq \{ f(x_1,\ldots, x_n) \text{ such that } x_i \in [a_i, b_i],\ 1 \le i \le n \} = \text{range of } f .$$

How the interval method can be used to solve a polynomial equation ?

- First, the range T of variable t, is determined by intersecting the ray with the bounding volume of the surface.

- After that, the method checks the possibility for the interval T (and its subintervals) to contain the value 0.

- This is done by interval evaluation of the polynomial equation .

- If this evaluation contains 0, then there is some chance for the polynomial to have real zeros.

- In this case, T is subdivided into two subintervals and the process is repeated for the subintervals in a recursion fashion.

- The recursion terminates when the width of the current subinterval is smaller than a threshold (in case of isolation) or when it can be treated as a single point which is a real root of the polynomial.

How the interval method can be used to solve a system of non linear equations ?

- The same technique can be used to isolate or to find the solutions of a system of non linear equations .

- For the sake of simplicity, consider a system of two polynomial equations where the two unknowns are u and v ranging respectively over U = [u1, u2] and V = [v1, v2] :

  $f(u,v) = 0$ with $(u,v) \in I$

  $g(u,v) = 0$ with $(u,v) \in I$.

- I = [u1, u2] x [v1, v2]

- The method checks the possibility for a solution to lie within the entire domain of the 2D interval I.

- This is done by interval evaluation of the functions f(u, v) and g(u, v).

- If both the evaluations contain 0, then there is some chance for the solution to exist.

- If so, I is subdivided into 2D subintervals and the process is repeated recursively as pointed out above.

Root isolation methods

Collins's method

- Let $P(x) = \sum_{i=0}^{n} a_i \cdot x^i$

- Descartes' rule states that the number of sign variations var(a_n, a_{n-1},…, a_0) exceeds the number of positive zeros, multiplicities counted, by an even non negative integer.

- Hence if var(P) is equal to 0, P has exactly no positive roots, and if var(P) is equal to 1, P has exactly one positive root.

- A surprising theorem which Uspensky attributes to Vincent in 1886 , shows that after a finite number of transformations

$P'(x) = P(x+1)$ and $P^*(x) = (x + 1)^n * P(1 / (x + 1))$

one arrives at polynomial having sign variation 1 or 0.

Root isolation methods

Collins's method : Algorithm

```
procedure real_root_isolation( P : polynomial ; var L :
                                     list_of_intervals) ;
var
    bound : real ;
    B : polynomial ;
    L' : list_of_intervals ;

begin

    { bound the positive roots of P by bound }

    bound := 2^k ;
    if k  >= 0 then B(x) := P(bound * x)
            else  B(x) := (1 / ( bound*2^n ) ) * P(bound * x) ;

    { call the isolation procedure which gives a list L' of
    isolation intervals of B }

    isolation_proc( B, 0, 1, 1, L' ) ;

    { call the procedure replace_L'_by_L to replace each interval
    [a_i, b_i ] by  [ bound * a_i, bound * b_i ] }

    replace_L'_by_L( L, L' ) ;

end;
```

```
procedure isolation_proc( B : polynomial ; min_int, max_int  :
                    real ; width : real ;var L : list_of_intervals ) ;

var
    L1, L2  :  list_of_intervals ;
    B*, B', B"  :  polynomial ;
    I   :  interval ;

 { min_int and max_int are respectively the smallest and largest
    endpoints of the current interval }
begin
    { transform the zeros of B in [0, 1] onto the zeros of B* in
     [0, ∞] }

    B*(x) := (x + 1)^n * B( 1 / (x + 1) ) ;
    { end of recursion }

    if var(B*) = 0 then begin
                         L := empty ;
                         return ;
                         end
                    else  if (var(B*) = 1 ) and (width <= threshold ) then
                         begin
                         I := [min_int, max_int] ;
                         insert_in_L( I ) ;
                         end ;
    { process the left-half subinterval by transforming the zeros of B in [0, 1/2]
      on the zeros of B in [0, 1] }

    B'(x) := 2^n * B(x / 2) ;
    isolation_proc( B', min_int, max_int - width / 2, width / 2, L1 ) ;

    { process the right-half subinterval by transforming the zeros of B in [1/2, 1]
      on the zeros of B  in [0, 1] }

    B"(x) := B'(x + 1) ;
    isolation_proc( B", min_int + width / 2, max_int, width / 2, L2 ) ;

    { put the two lists L1 and L2 in L }
    add_list ( L, L1 ) ;
    add_list ( L, L2 ) ;
end ;
```

Intersecting Bicubic surfaces

- $Q(u,v) = [\ x(u, v), y\ (u, v), z\ (u, v)\ ]$

$$= \sum_{i=0}^{3} \sum_{j=0}^{3} B_i(u) \,.\, B_j(v) \,.\, P_{ij} \qquad (7)$$

- where Pij are the control points of the surface, and $B_i(u)$, $B_j(v)$ the blending functions which determine the type of surface (B-spline, Bezier, Beta-spline…).

- These blending functions depend on the two parameters u and v which both range over [0, 1].

- A ray may be considered as intersection of two planes defined by :

$$[\ A1, B1, C1\ ] \,.\, [\ x, y, z\ ] = D1$$

$$[\ A2, B2, C2\ ] \,.\, [\ x, y, z\ ] = D2$$

- The ray equation iexpressed as :

$$[\ x, y, z\ ] = [\ x0, y0, z0\ ] + t \,.\, [\ dx, dy, dz\ ]$$

Intersecting Bicubic surfaces

- The two planes can be determined as follows :

$$[A1, B1, C1] = [x0, y0, z0] \times [dx, dy, dz]$$

$$[A2, B2, C2] = [A1, B1, C1] \times [dx, dy, dz]$$

$$D1 = [A1, B1, C1] \times [x0, y0, z0]$$

$$D2 = [A2, B2, C2] \times [x0, y0, z0]$$

- After substitution we obtain the following system :

$$\sum_{i=0}^{3} \sum_{j=0}^{3} ([A1, B1, C1] . P_{ij}) . B_i(u) . B_j(v) - D1 = 0$$

$$\sum_{i=0}^{3} \sum_{j=0}^{3} ([A2, B2, C2] . P_{ij}) . B_i(u) . B_j(v) - D2 = 0 \qquad (11)$$

Intersecting Bicubic surfaces

- Once these equations have been stated, ray-surface intersection may be performed by means of one of the existing methods. At least, three methods can be used :

- method which decomposes a patch into a set of planar polygons,

- method which subdivides recursively a patch into four patches. The recursion terminates when the bounding volume of a subpatch is intersected by the current ray and satisfies a size criterion.

- method which uses numerical techniques. The resultant method may be used (see Kajiya).

SPATIAL SUBDIVISION

Principle

- The rectangular bounding volume of the scene is subdivided into 3D cells

- Each cell contains a small portion 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

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

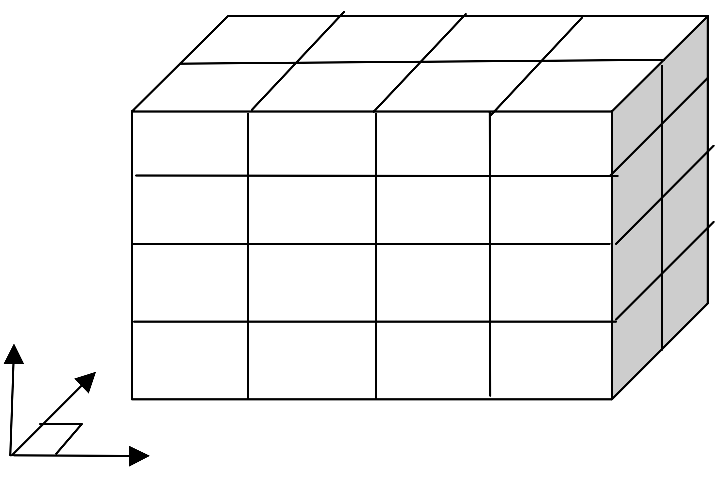
- A second procedure which determines the next cell along a ray

SPATIAL SUBDIVISION

Subdivision into a 3D uniform grid

Subdivision

- 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 objets partially or totally within the cell
- Example



SPATIAL SUBDIVISION

Subdivision into a 3D uniform grid

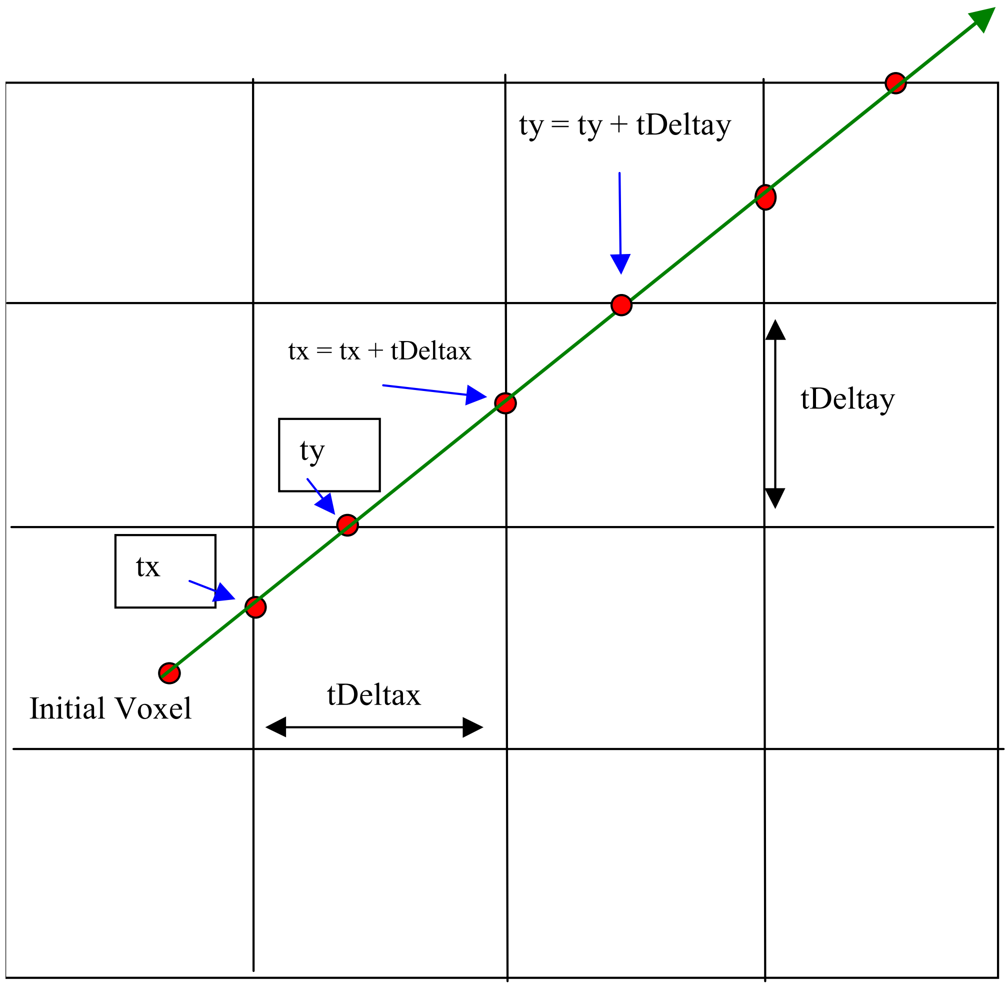
- Next cell along a ray : classical method

- 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 > 0then 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

SPATIAL SUBDIVISION

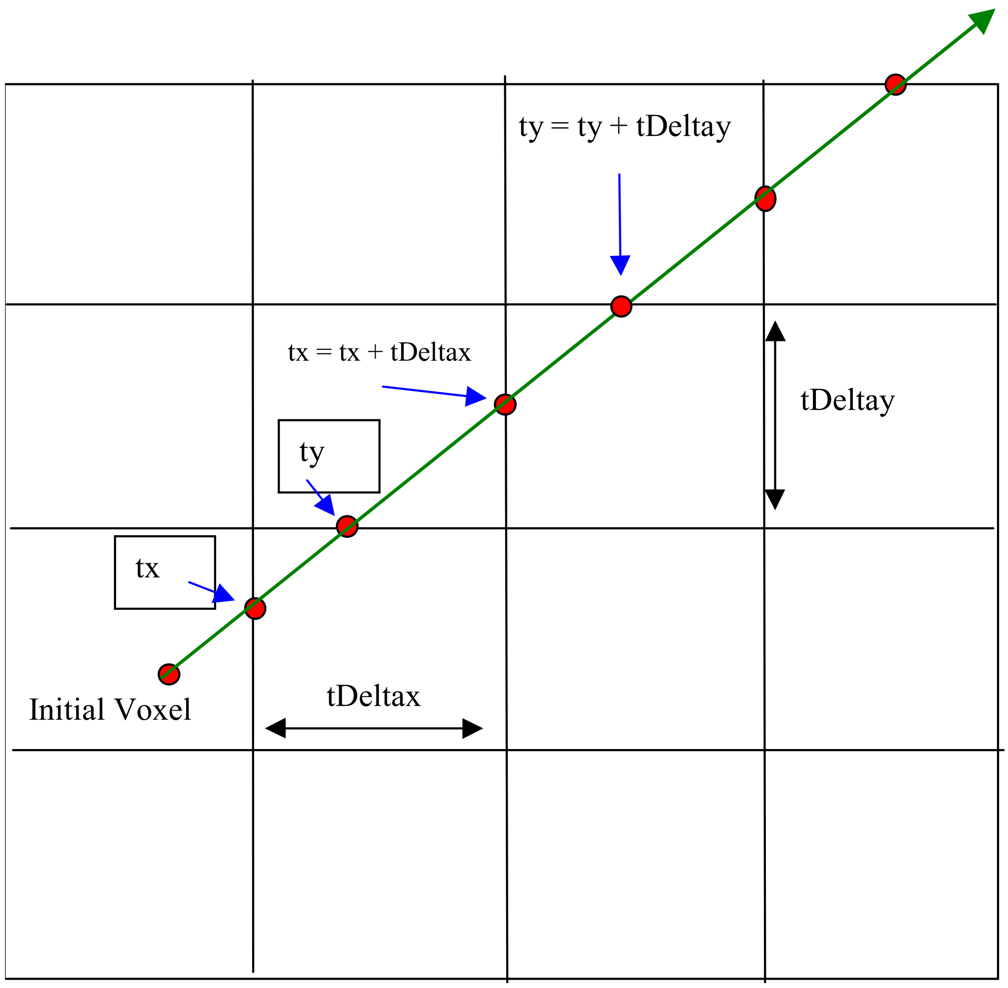
Subdivision into a 3D uniform grid

Next cell along a ray : Amanatides's method



- Initialization

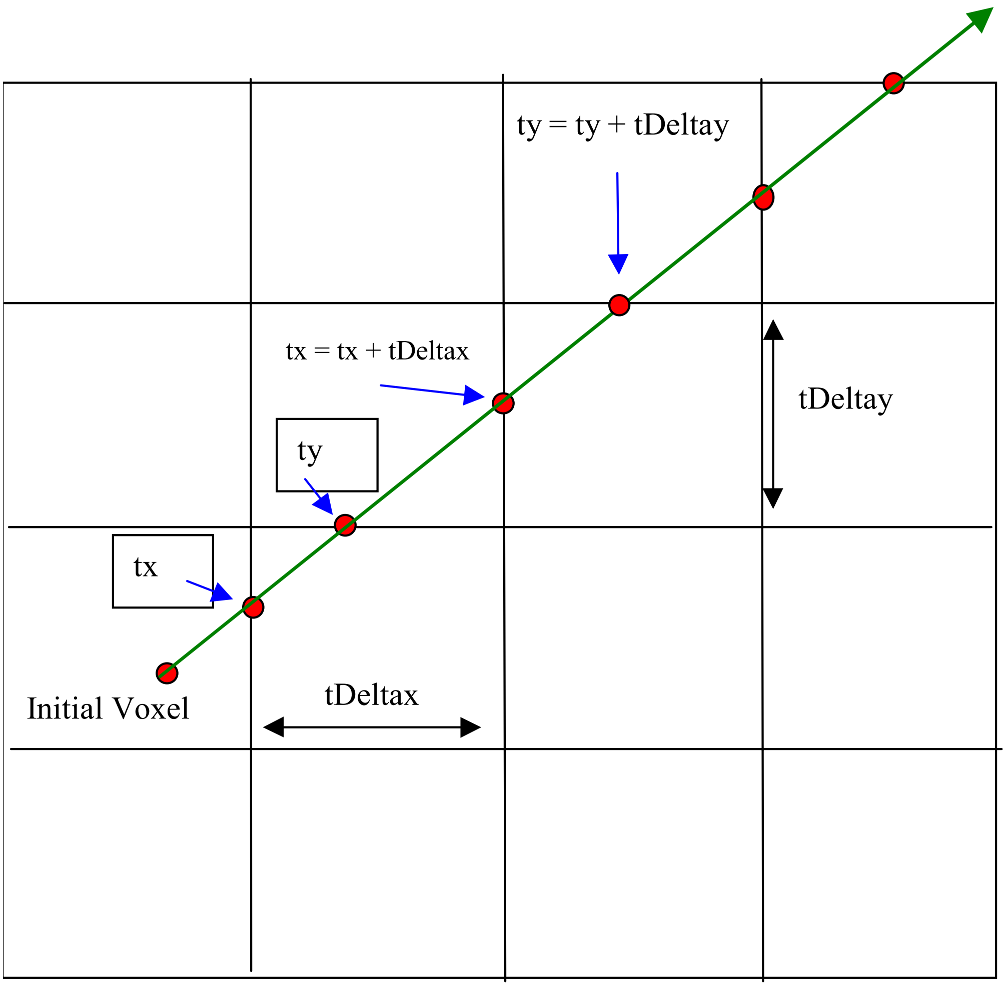
  - Ray equation : P = P0 + t . D
  - Identify the voxel containing the ray origin O
  - If O is outside the grid, find the point through which the ray enters the grid and determine the adjacent voxel
  - X, Y and Z : voxel indices
  - StepX, stepY and stepZ : initialized to 1, incremented or decremented 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



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 ;
}
```



SPATIAL SUBDIVISION

Subdivision into a non uniform grid

Subdivision

- The rectangular bounding volume of the scene is recursively sliced by 3 planes perpendicular to the x, y and z axes one after the other

- Each slicing plane divides a space (a 3D cell) into two subspaces (a3D cells) of equal dimensions

- 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

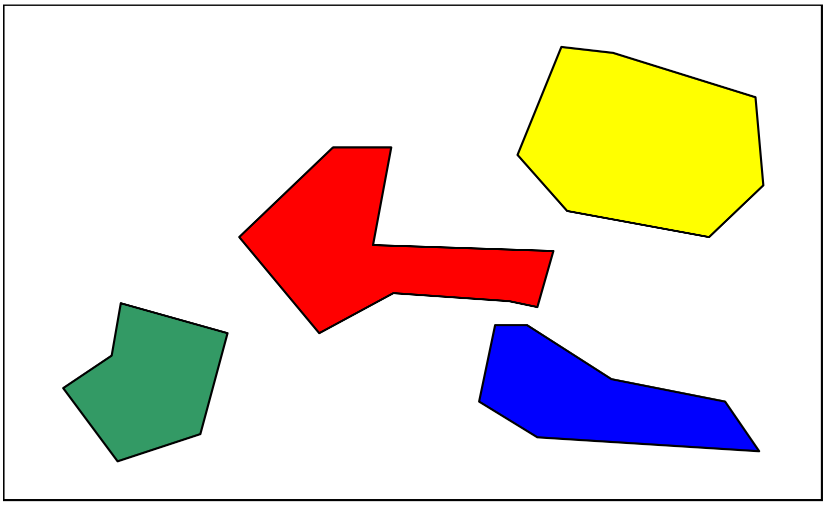
- Each cell is identified by a number

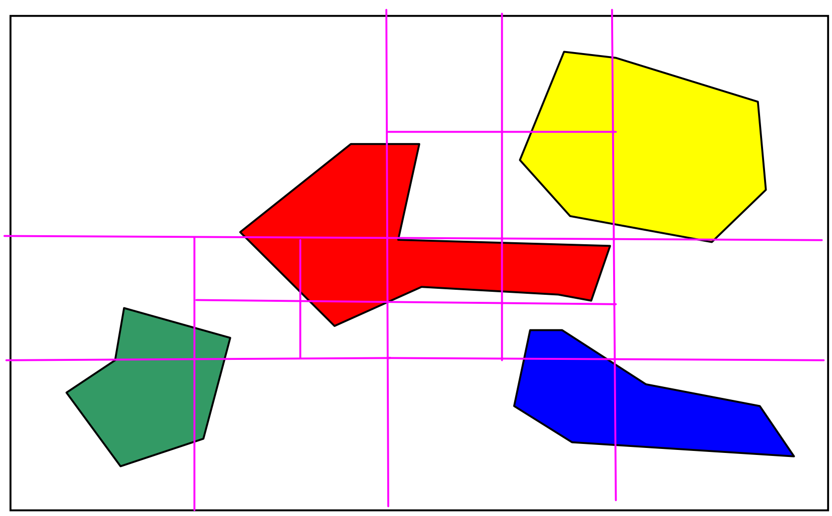
- Each cell is represented by a data structure containing a pointer to the objets partially or totally within it

SPATIAL SUBDIVISION

Subdivision into a non uniform grid

Subdivision





SPATIAL SUBDIVISION

Subdivision into a non uniform grid

Next cell along the ray

- P : out going point
- Push P along the normal to the outgoing face
- The results is another point P'
- 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.
- Determine the cell containing P'
- If P is on an edge or a vertex of a cell, push it simultaneously in the directions of the normals of the faces sharing it