LEI / LCC
DEPARTAMENTO DE INFORMÁTICA
UNIVERSIDADE DO MINHO

Real Time Visualization

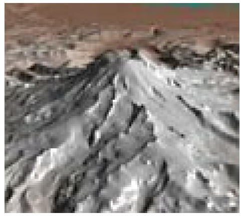
Back Face and View Frustum Culling

Spatial and Object Oriented Partitioning

Issue: Triangle count

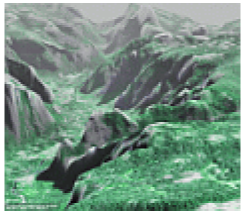
Buda: 1 million triangles

Power plant: 13 million triangles

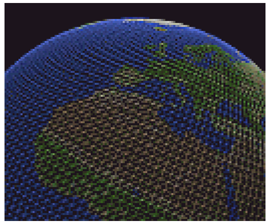


Terrain: 1.3 million triangles

Terrain: 512 million triangles



Terrain: 16 million triangles



Earth: 1 billion points

Issues

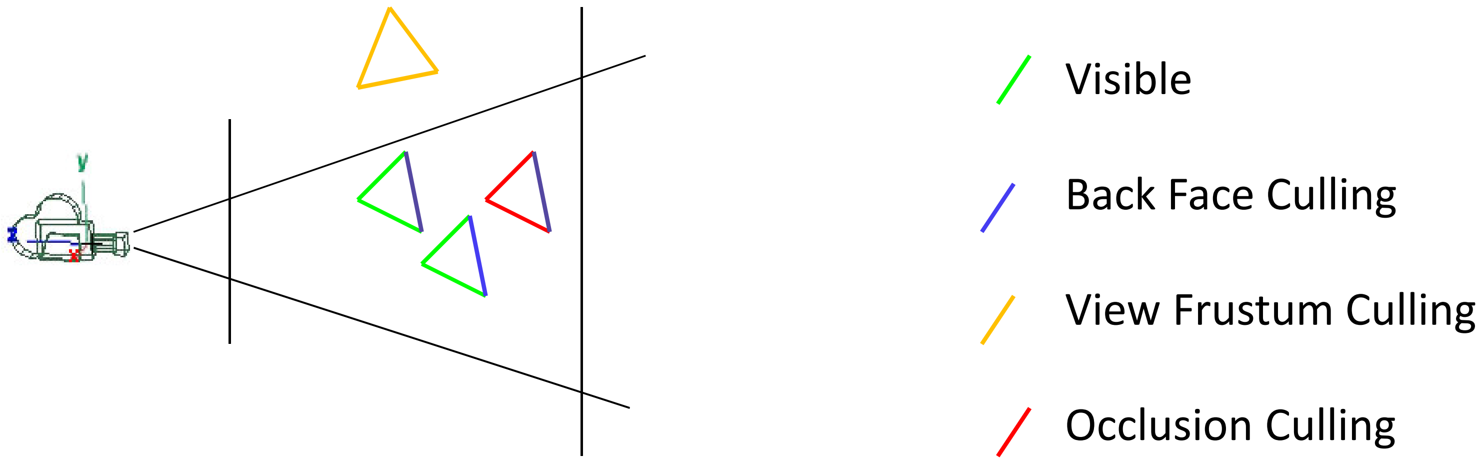
- For a given camera position, which triangles are relevant to create the rendered image?
- For distant models do we need a high level of detail?
- How to interact with models with such a high polygon count?

Culling

- Avoid (fully) processing every triangle/model
  - *Back Face Culling*
  - *View Frustum Culling*
    - Bounding Volumes
    - *Spatial Partitioning – BSP, K-d trees,* Quad and Octrees
  - *Occlusion Culling*

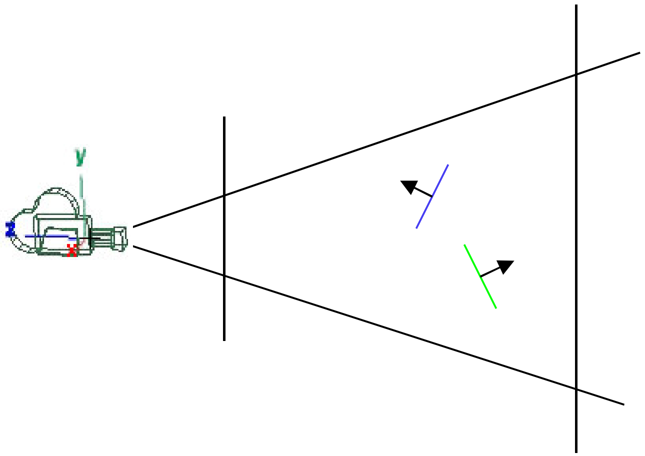
Culling

- Culling types:



Back Face Culling

- Do not process triangles facing away from the camera.



$$o = v \cdot n$$

```
if(o > 0)
        render
else
        cull
```

v – vector from triangle to camera
n – normal

Back Face Culling

- Using deprecated OpenGL
  - Enable/Disable

```
glEnable(GL_CULL_FACE);
glDisable(GL_CULL_FACE);
```

  - Define which face is visible

```
glCullFace(GL_BACK); // ou GL_FRONT
```

  - Define the default front orientation

```
glFrontFace(GL_CCW); // ou GL_CW
```

Back Face Culling

- Allows the elimination of large number of triangles

- Performed in hardware for every triangle

Back Face Culling

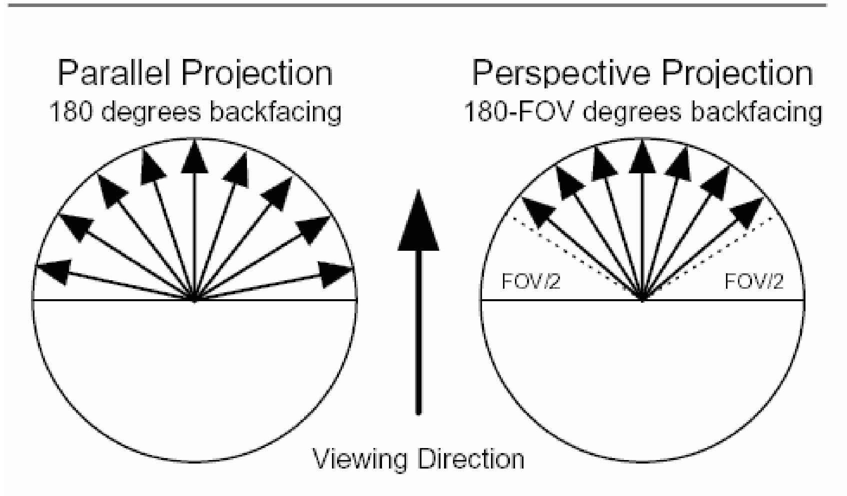
- The hardware based approach still requires the vertices to be sent to the graphics card;
- The elimination only occurs in the pipeline after the primitives are built;
- Ideally we could avoid the unnecessary communication ...
- ... and processing the vertices ...
- However, a CPU based solution for individual triangles would be too slow.

Back Face Culling

- Zhang and Hoff proposed:

  – Group triangles according to their normal

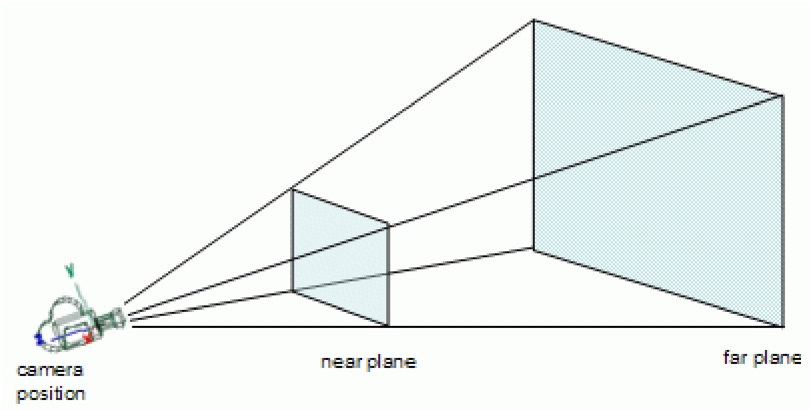
  – Work with groups instead of individual triangles



What are the issues with this approach?

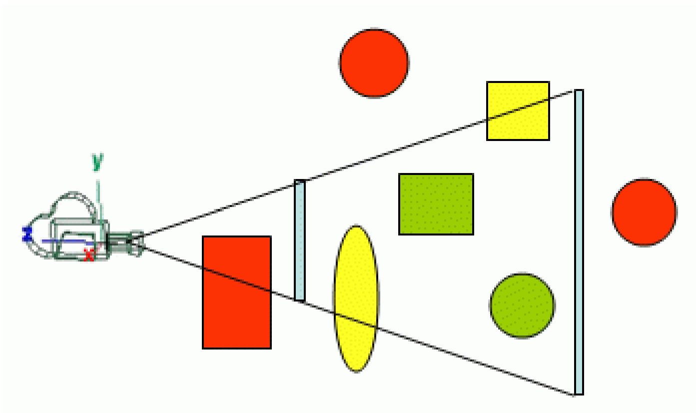
View Frustum Culling

- View Frustum Culling



View Frustum Culling

- Eliminate polygons outside the view frustum



Test the relative position of the triangle/object/volume to the frustum planes

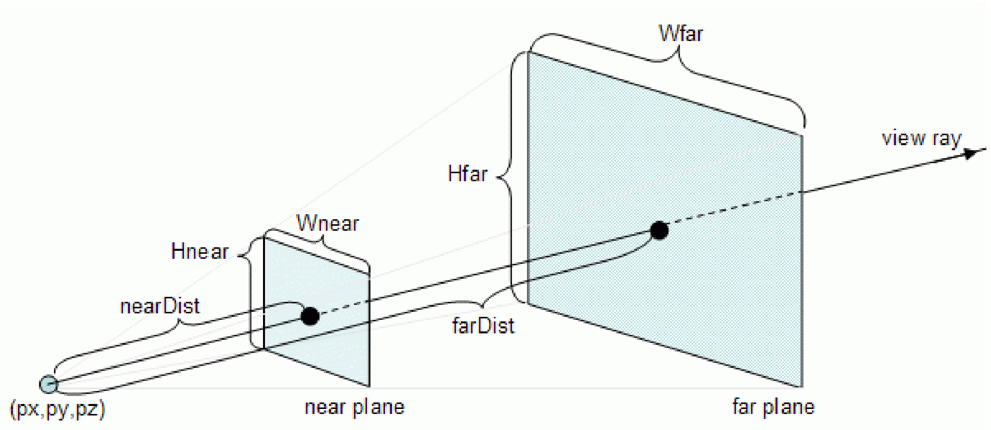
View Frustum Culling

- Steps:
  - Setup: Get the frustum plane equations (once per frame)
  - Test: For each vertex/triangle/object/volume test if it is inside/outside of the frustum

View Frustum Culling

- Describing the View Frustum

```
gluPerspective(fov, ratio, nearDist, farDist);
gluLookAt(px,py,pz, lx,ly,lz, ux,uy,uz)
```



$$Hnear = 2 \times \tan(\frac{fov}{2}) \times nearDist$$

$$Wnear = Hnear \times ratio$$

$$Hfar = 2 \times \tan(\frac{fov}{2}) \times farDist$$

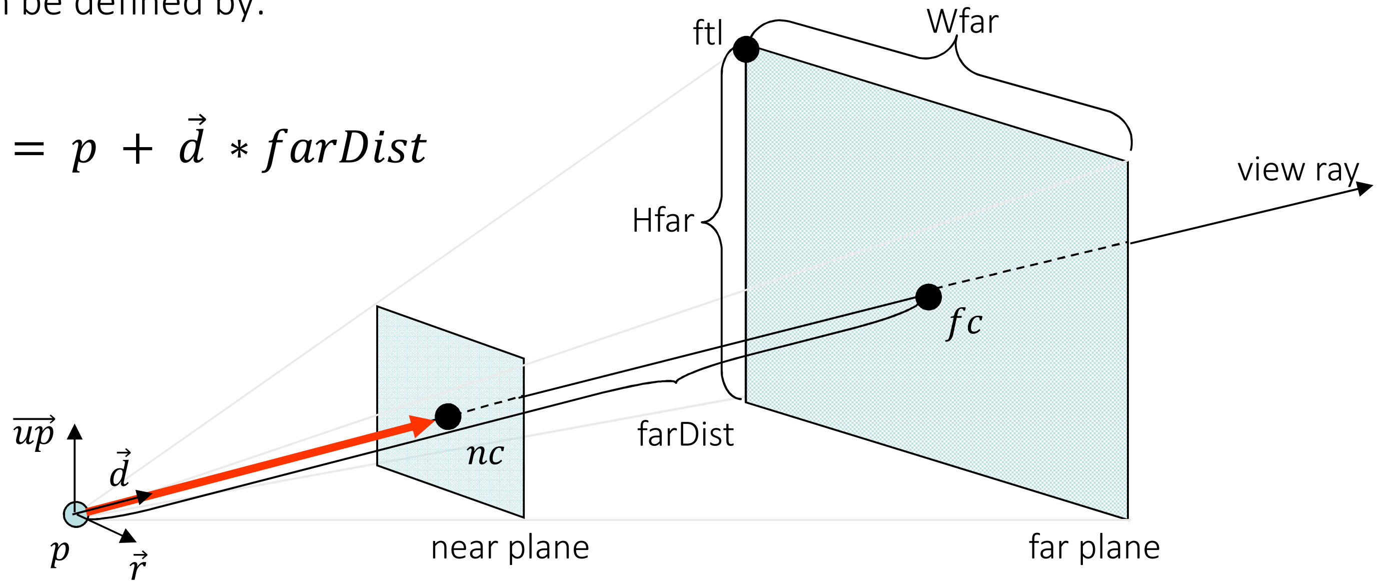
$$Wfar = Hfar \times ratio$$

View Frustum Culling

- Geometric Approach
  - A plane is defined by a normal and a point
  - *far plane* can be defined by:

    normal: $\vec{d}$

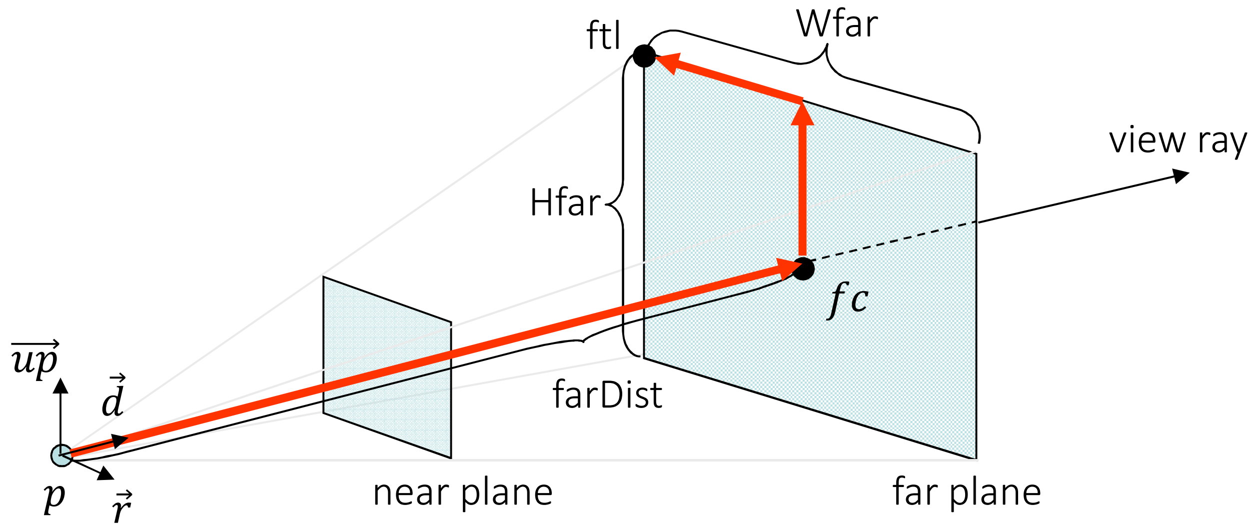
    point: $fc = p + \vec{d} * farDist$



View Frustum Culling

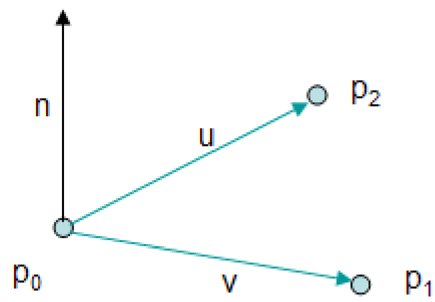
- Geometric Approach
  - To define the normal we require three points

$$fc = p + \vec{d} \times farDist$$
$$ftl = fc + \left(\overrightarrow{up} \times \frac{Hfar}{2}\right) - \left(\vec{r} \times \frac{Wfar}{2}\right)$$



View Frustum Culling

- Normalized plane equation $Ax + By + Cz + D = 0$
  - normal + point $\vec{n} = (n_x, n_y, n_z) = \vec{v} \times \vec{u}$
    $\vec{n} = \vec{n}/|\vec{n}|$



$$A = n_x \quad B = n_y \quad C = n_z$$

Point p_o is in the plane, hence

$$Ap_{0x} + Bp_{0y} + Cp_{0z} + D = 0 \Leftrightarrow$$
$$D = -Ap_{0x} - Bp_{0y} - Cp_{0z} = -\vec{n} \cdot p_0$$

View Frustum Culling

- Point – plane distance

  - Distance from point

    $$p = (p_x, p_y, p_z)$$

  - to plane

    $$Ax + By + Cz + D = 0$$

  - Is defined as

$$\boxed{dist(p) = Ap_x + Bp_y + Cp_z + D}$$

If dist(p) > 0 then p is on the side where the normal is pointing

View Frustum Culling

- Test
  - Point in frustum?
    - Assuming plane normals are pointing to the frustum's inside

```
int FrustumG::pointInFrustum(Vec3 &p) {

        int result = INSIDE;
        for(int i=0; i < 6; i++) {
                if (pl[i].distance(p) < 0)
                        return OUTSIDE;
        }
        return(result);
}
```

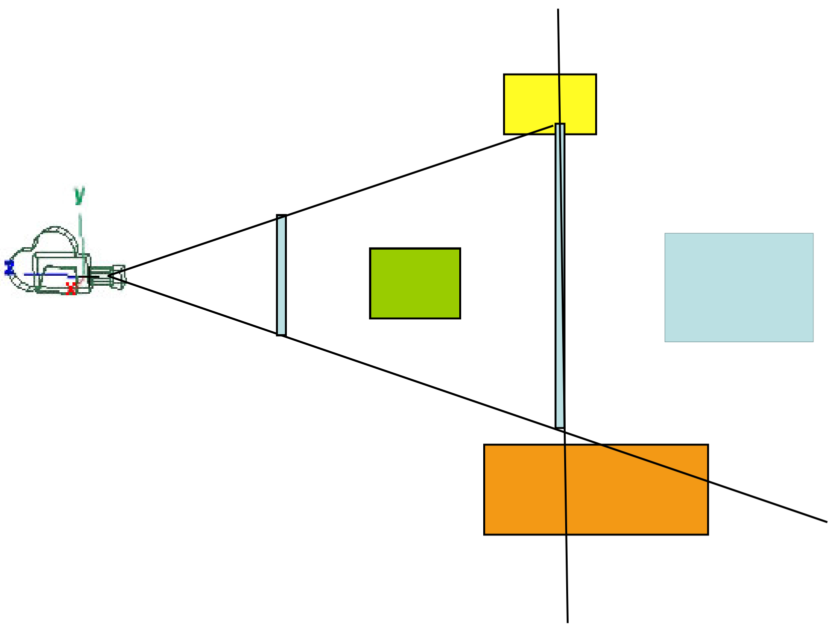
View Frustum Culling

- Test
  - Spheres

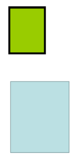
```
int FrustumG::sphereInFrustum(Vec3 &p, float radius) {
        float distance;
        int result = INSIDE;

        for(int i=0; i < 6; i++) {
                distance = pl[i].distance(p);
                if (distance < -radius)
                        return OUTSIDE;
                else if (distance < radius)
                        result = INTERSECT;
        }
        return(result);
}
```

View Frustum Culling

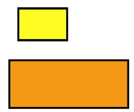
- Test
  - Boxes: Corner test



Simple cases:



Wht about these:

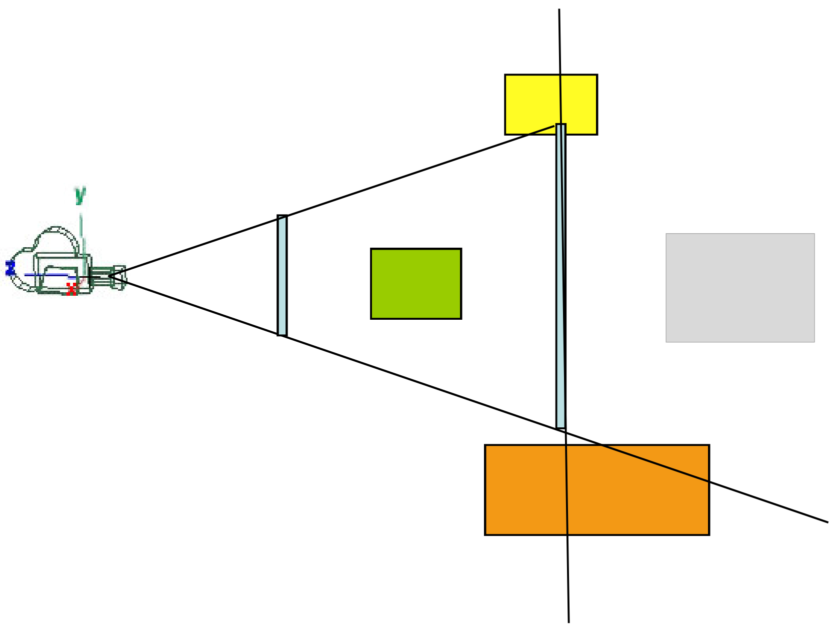


?

View Frustum Culling

- Test

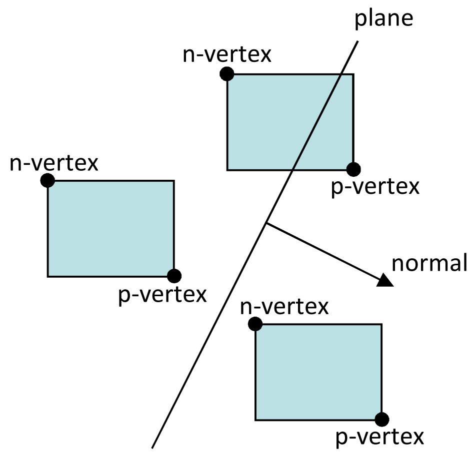
  - Boxes: Corner Test



Accept all boxes whose corners are not on the wrong side of a single plane

View Frustum Culling

- Test
  - Boxes




```
p = (xmin,ymin,zmin)
if (normal.x >= 0)
        p.x = xmax;
if (normal.y >=0))
        p.y = ymax;
if (normal.z >= 0)
        p.z = zmax;
```

```
n = (xmax,ymax,zmax)
if (normal.x >= 0)
        n.x = xmin;
if (normal.y >=0))
        n.y = ymin;
if (normal.z >= 0)
        n.z = zmin;
```

View Frustum Culling

- Can also be performed in:
  - Clip Space
  - Global Space (World Space)

View Frustum Culling

- Clip Space: Setup
  - Let *M* be the modelview matrix , *P* the projection matrix, and *p* a point in World Space

$$A = P \times M$$
$$p' = Ap$$

  - Then:
    - A converts points from World Space to Clip Space
    - *p'* is a point in Clip Space,

View Frustum Culling

- Clip Space - Setup
  - Getting the matrices with (deprecated) OpenGL:

```
float M[16],P[16];

glGetFloatv(GL_MODELVIEW_MATRIX,M);

glGetFloatv(GL_PROJECTION_MATRIX,P);
```

View Frustum Culling

- Multiplying matrices with OpenGL

- Code to compute A = P * M

```
glPushMatrix();

glLoadMatrixf(P);
glMultMatrixf(M);
float A[16];
glGetFloatv(GL_MODELVIEW_MATRIX, A);

glPopMatrix();
```

View Frustum Culling

- Clip Space: Test
  - Visible points are inside the cube, centered in the origin, with dimension = 2, i.e., it’s coordinates after the perspective divide are between -1 and 1 in all axis.
  - Let *p* be a point in World Space,
  - Then $p' = (x', y', z', w') = Ap$ is a point in Clip Space.
    - » *p′* is inside the view frustum if:

$$
\begin{aligned}
-w' &< x' < w' \\
-w' &< y' < w' \\
-w' &< z' < w'
\end{aligned}
$$

View Frustum Culling

- Clip Space: Test
  - Required operations:
    - 16 multiplications + 12 aditions to get the point in clip space
    - Up to 6 tests (<,>) to determine if the point is inside/outside.

View Frustum Culling

- World Space: Setup
  - Let $p=(x,y,z,w)$ e $p'=Ap=(x',y',z',w')$.
  - We know that
    - $-w' < x' < w'$

View Frustum Culling

- World Space: Setup

$$A = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \\ l_4 \end{bmatrix}$$

$$p' = A \times p = \begin{bmatrix} l_1 \times p \\ l_2 \times p \\ l_3 \times p \\ l_4 \times p \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

then (in Clip Space) if

$$-w' < x' < w'$$

We get (in World Space)

$$-p * l_4 < p * l_1 < p * l_4$$

View Frustum Culling

- World Space: Setup
    - $-p*l_4 < p*l_1 < p*l_4$
  - If *x* is on the right side of the left plane then:
    - $-p*l_4 < p*l_1$
    - $0 < p*l_1 + p*l_4$
    - $0 < p * (l_1 + l_4)$
    - $0 < x(a_{11}+a_{41}) + y(a_{12}+a_{42}) + z(a_{13}+a_{43}) + w(a_{14}+a_{44})$

View Frustum Culling

- World Space: Setup
  - The left plane is defined as

$$x(a_{11}+a_{41}) + y(a_{12}+a_{42}) + z(a_{13}+a_{43}) + w(a_{14}+a_{44}) = 0$$

  - Similar reasoning allows the extraction of the remaining planes
  - The planes can be computed directly from $A = MP$.

View Frustum Culling

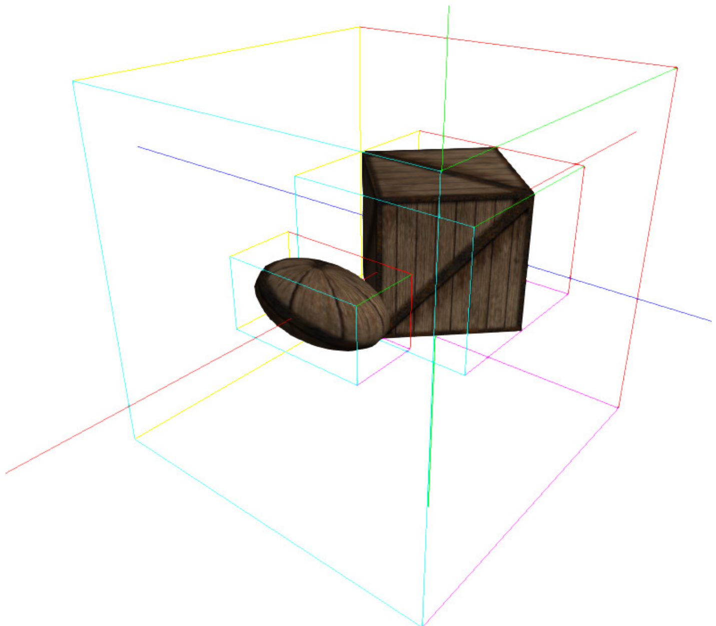
- Translation-Rotation Coherency (Assarsson and Möller)
  - ex: If an object is rejected by the left plane and the camera rotates to the right then the object will remain outside the view frustum.
    - What happens if we keep rotating?
  - ex: If an object is rejected by the near plane and the camera moves forward, then the object will still be outside the frustum.

View Frustum Culling

- Temporal Coherency (Assarsson and Möller)
  - Store for each object the plane that caused it to be rejected.
  - The stored plane should be the first to be tested.

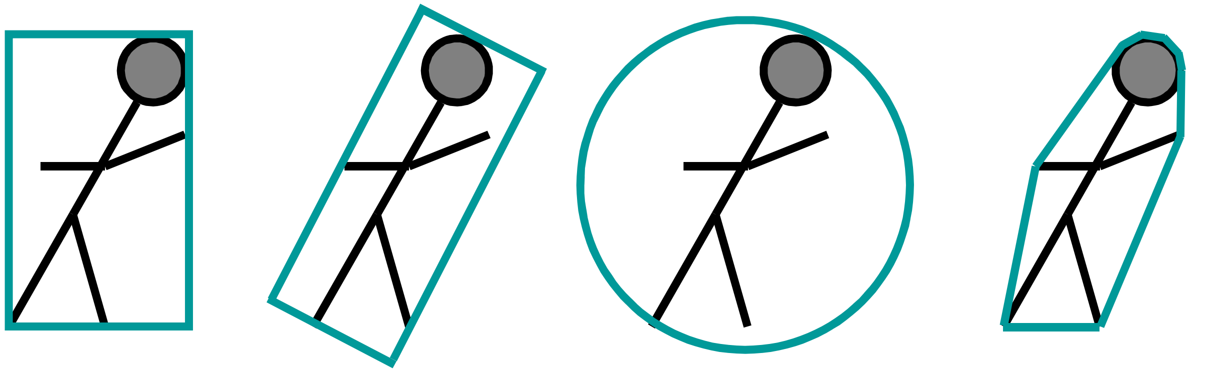
Bounding Volumes

- Bounding Volumes:
  - A closed volume that completely contains an object or objects.



Bounding Volumes

- Common bounding volume types:



AABB OBB Sphere Convex Hull

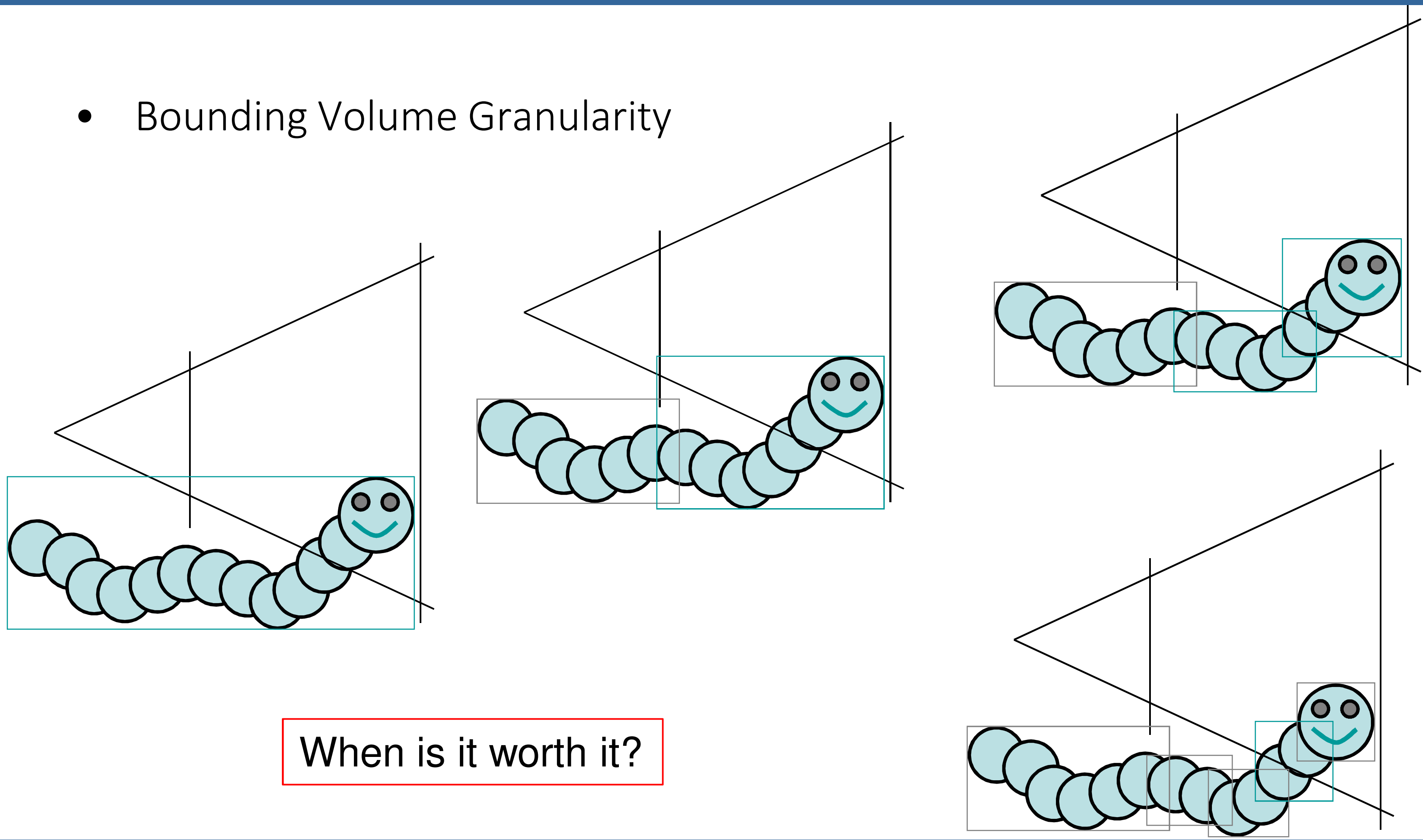
AABB = axis aligned bounding box, OBB = object aligned bounding box

Bounding Volumes

- Testing the BV allows the elimination of complex geometry with simple tests.
- What to do when the bounding volume is only partially inside the VF?

Bounding Volumes

- Bounding Volume Granularity



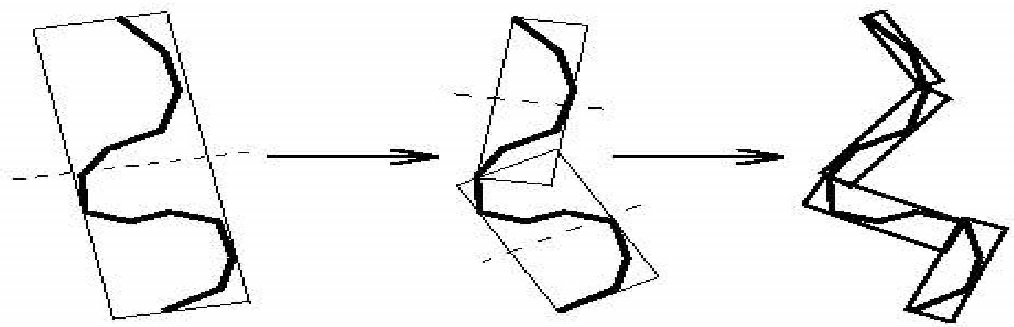
When is it worth it?

Bounding Volumes

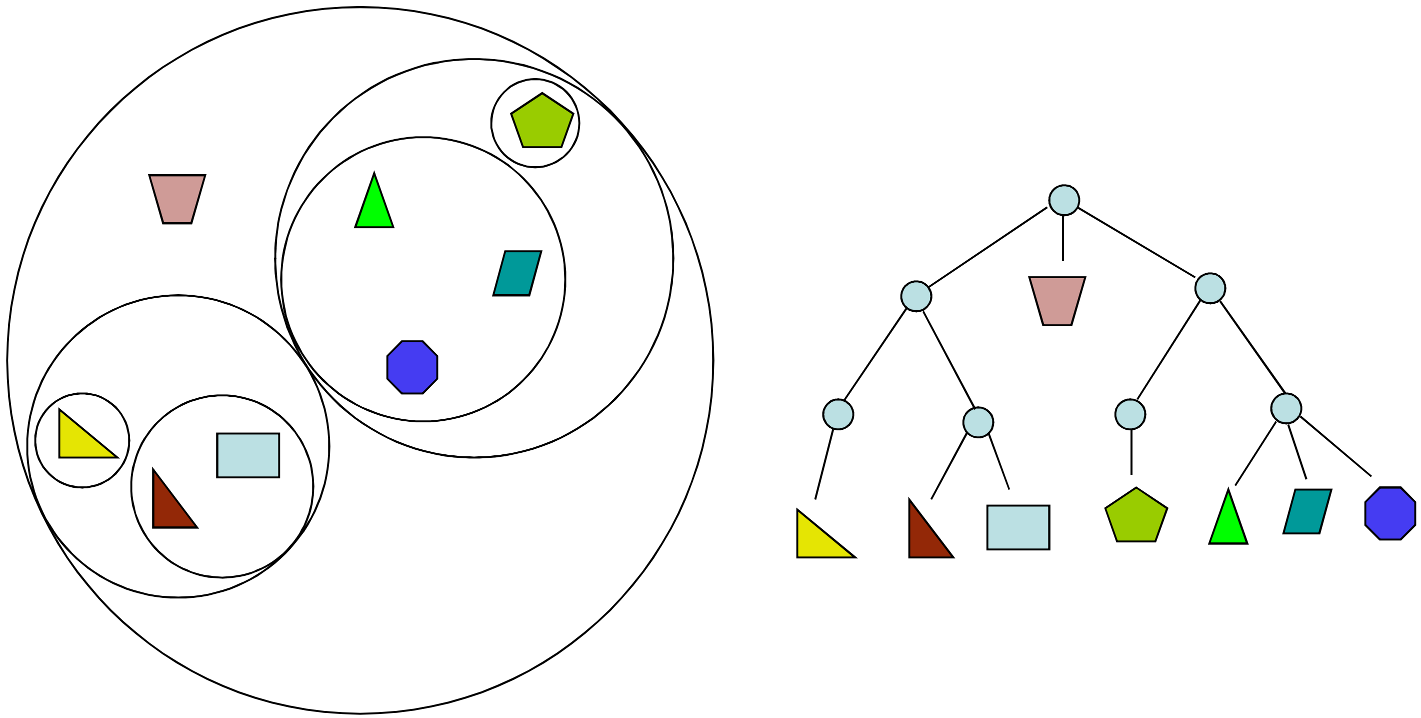
- Bounding Volume Granularity

=> Greater probability of rejection since we have less “empty space”

=> more tests are required, potentially less triangles are drawn

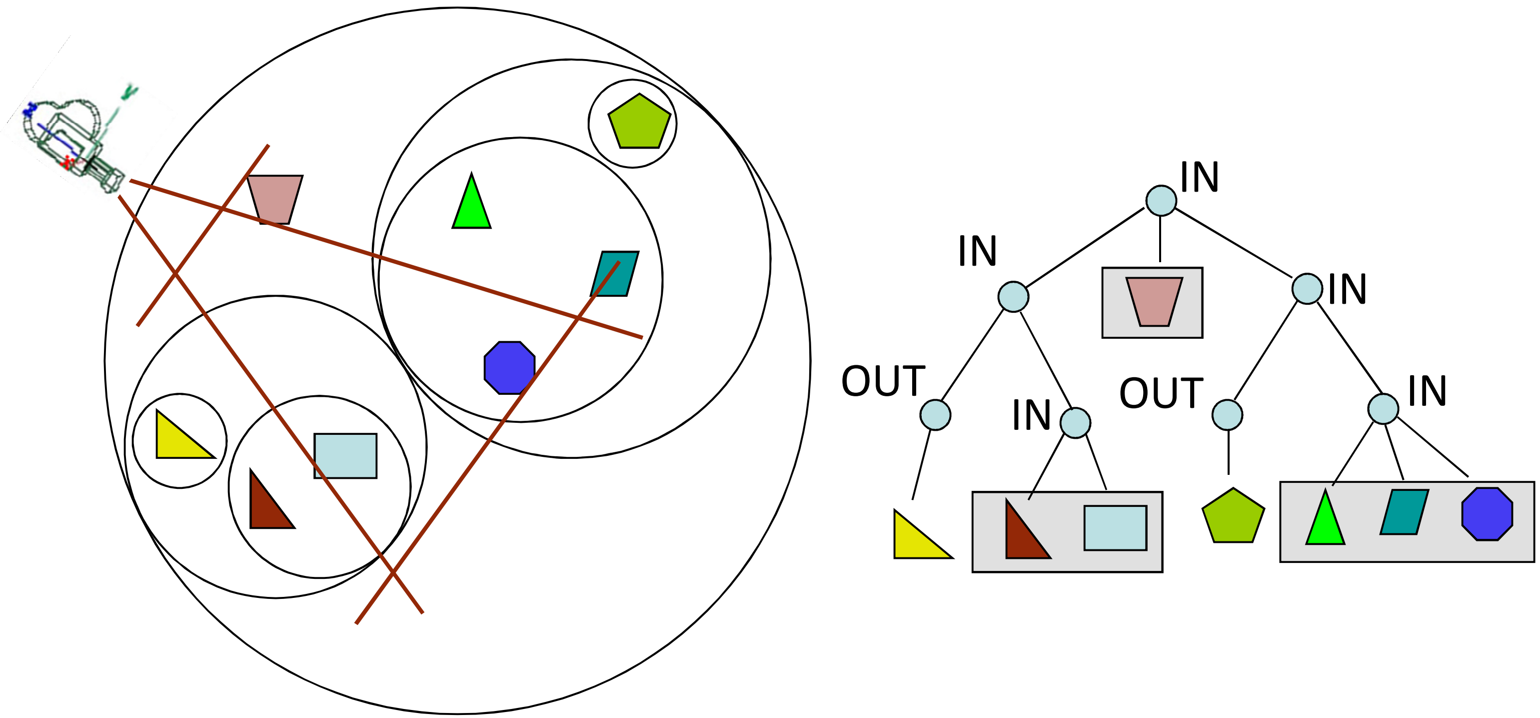


Hierarchical Bounding Volumes



Bounding Volumes Hierárquicos

- View Frustum Culling w/ BVH



Bounding Volumes

- A bounding volume based solution requires the explicit definition of objects:

object = { triangles}

- What if our scene is a “triangle soup”, without any semantics?

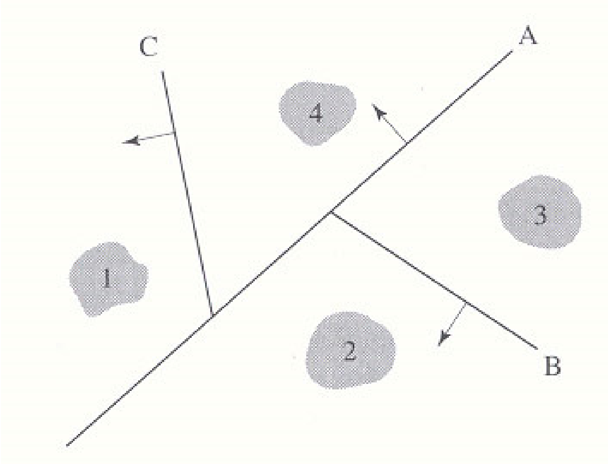
- A solution: Space Partitioning

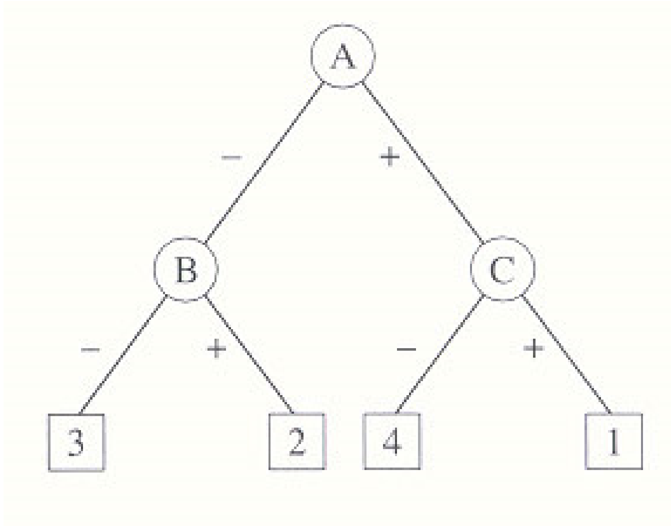
Space Partitioning - BSP

- BSP - Binary Space Partition
  - Using planes to recursively split the world in two
  - Results in a binary tree
  - The planes can be arbitrary
    - How to choose the planes?

Space Partitioning - BSP

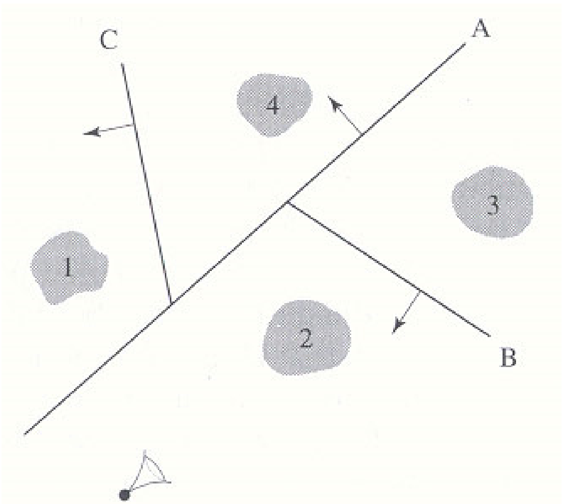
- Building a BSP

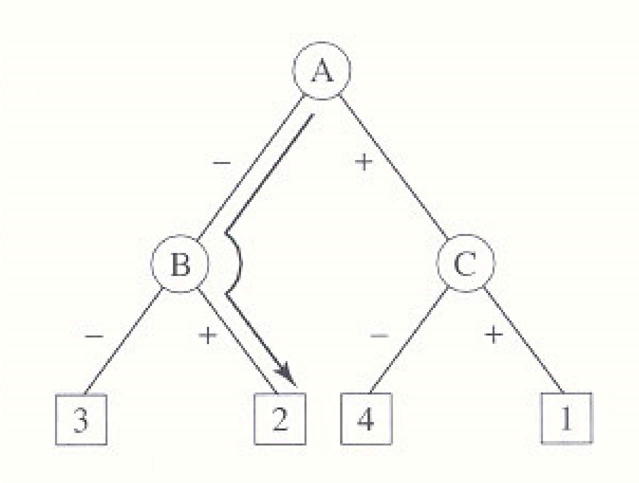




Space Partitioning - BSP

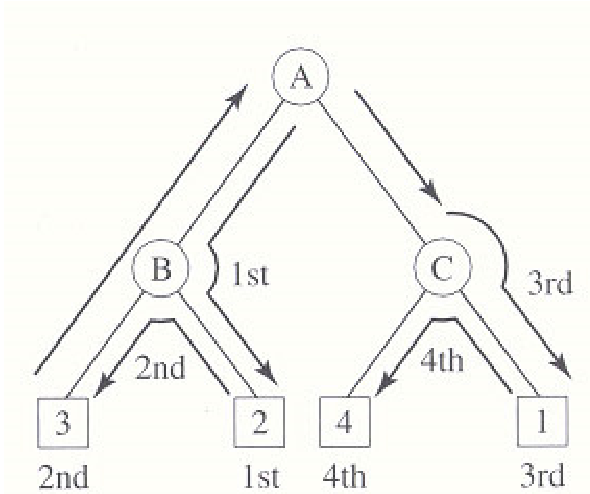
- Ordering triangles/objects





Object 2 is the “closest” to the camera

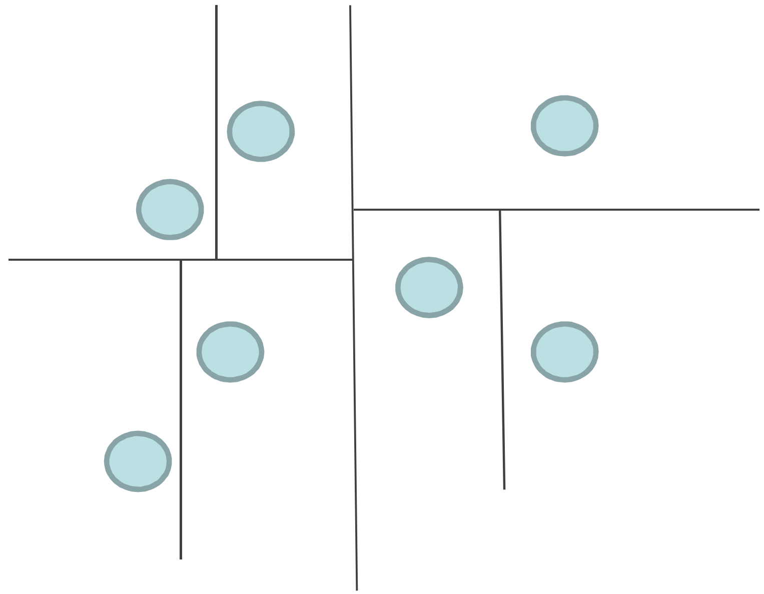
Ordered tree traversal



Space Partitioning– k-D trees

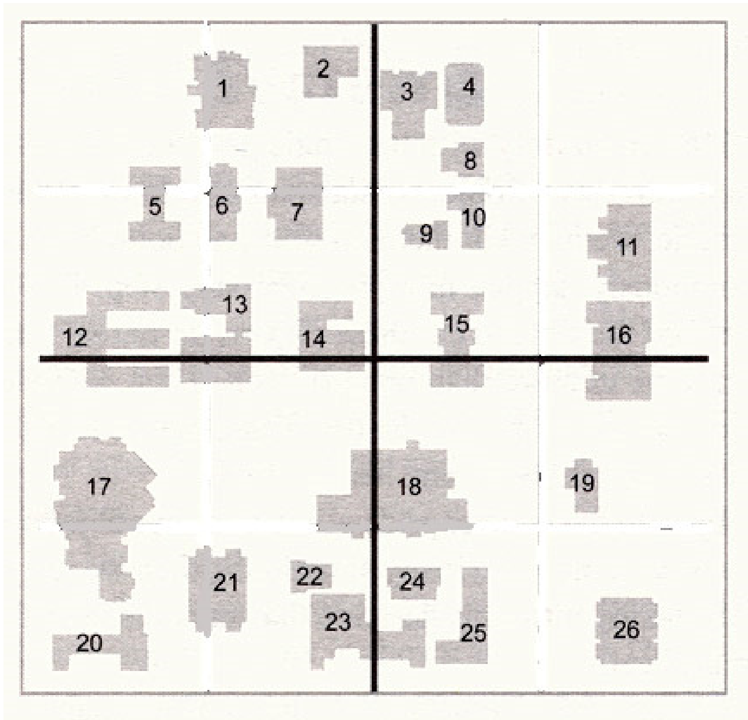
- Similar to BSPs but the planes are perpendicular to the axes.
- Building K-d tree:
  - Pick an axis, pick perpendicular plane and split the world in two regions.
  - Select a different axis. Select and a new perpendicular plane for each region (may have different planes for each region).
  - Iterate over all axis, and then restart the process.

Space Partitioning– k-D trees

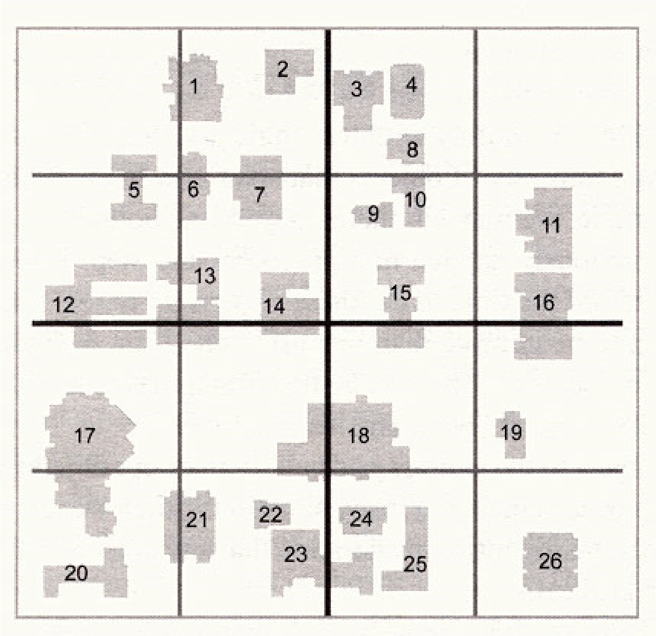


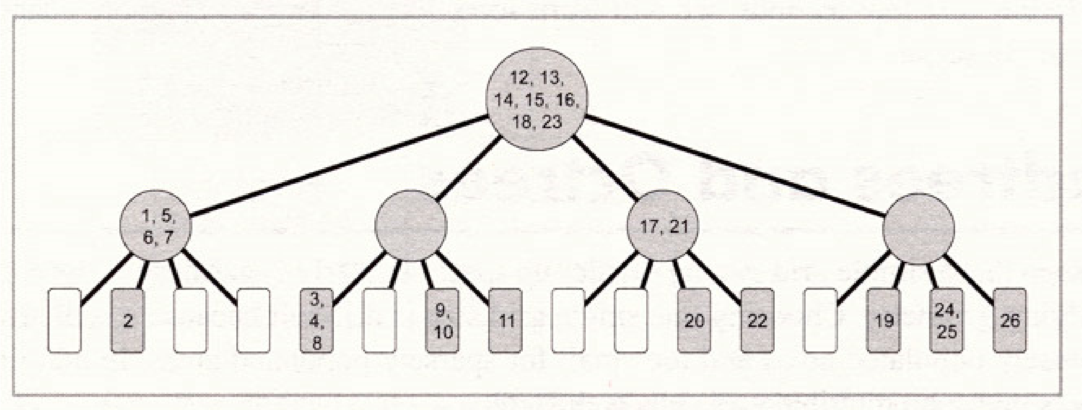
Space Partitioning - Quadtrees

- Divide the word recursively into quadrants.



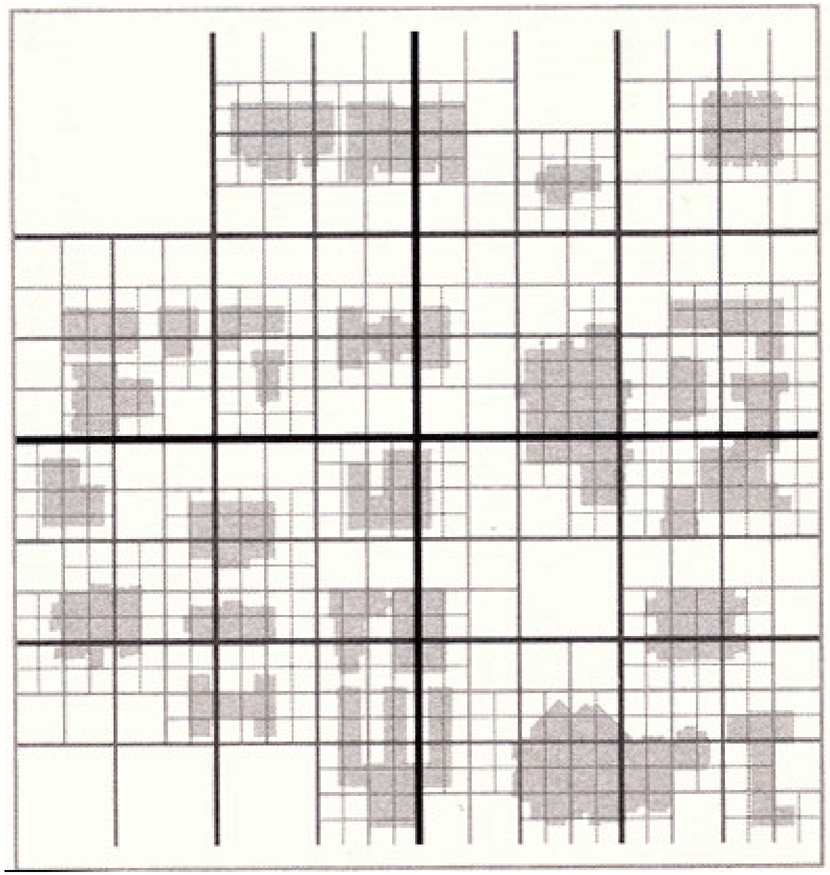
Space Partitioning - Quadtrees





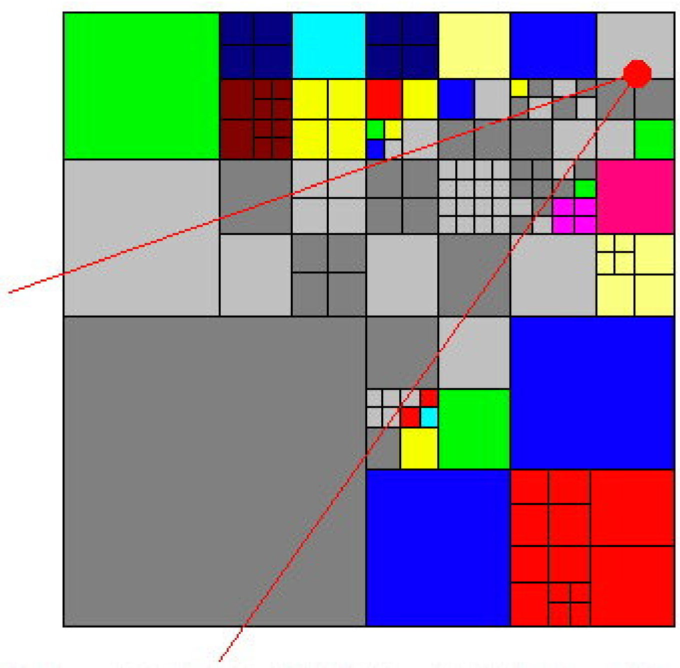
Space Partitioning - Quadtrees

- The recursion is not homogeneous



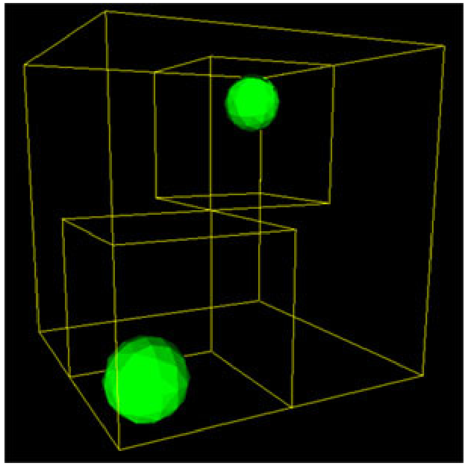
Space Partitioning - Quadtrees

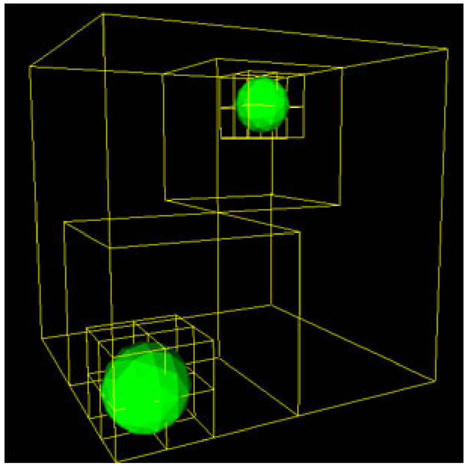
- View Frustum Culling with Quadtrees



Space Partitioning - Octrees

- Octrees:
  - Recursively divide the world into octants.





Spatial Partitioning

- Criteria to stop subdivision:
  - Cell polygon count has reached a threshold;
  - Tree's depth is getting too large;
  - Cell is too small.
- Why?

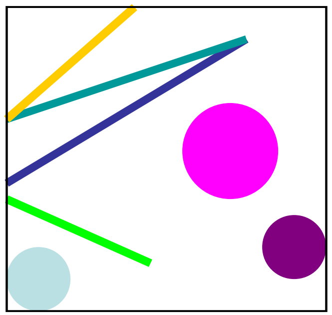
Space Partitioning

- What if an object/polygon occupies more than one cell?
- Possible solutions:
  - Include it in the parent cell;
  - Include it in both cells;
  - Split it such that each part fits in a single cell
- What are the merits and issues of each proposal?

BVH vs. Space Partitioning
(Kenneth E. Hoff III)

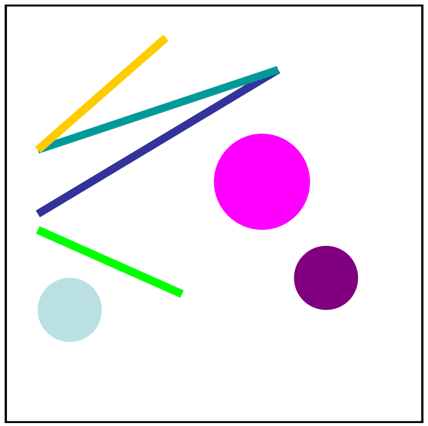
Bounding Volume Hierarchies

- Tightly fits objects
- Redundant spatial representation



Space Partitioning

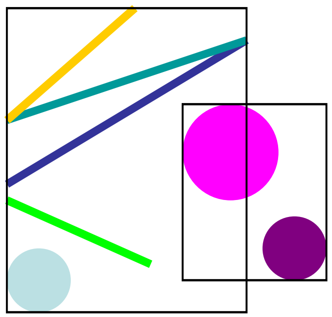
- Tightly fills space
- Redundant object representation



BVH vs. Space Partitioning (Kenneth E. Hoff III)

Bounding Volume Hierarchies

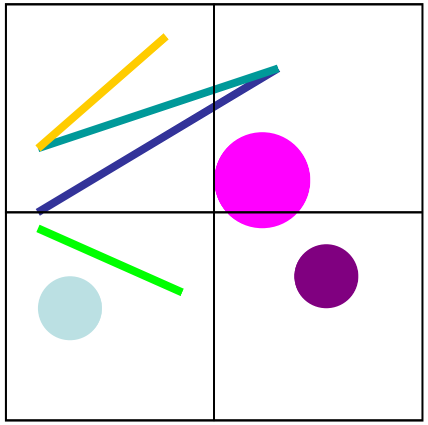
- Tightly fits objects
- Redundant spatial representation



Volumes overlap multiple objects

Space Partitioning

- Tightly fills space
- Redundant object representation

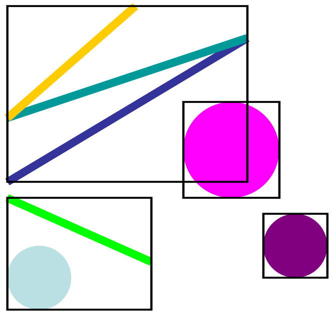


Objects overlap multiple volumes

BVH vs. Space Partitioning (Kenneth E. Hoff III)

Bounding Volume Hierarchies

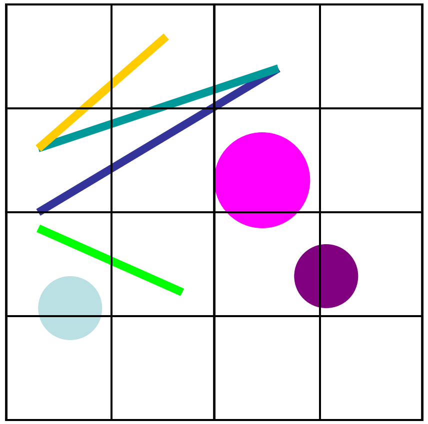
- Tightly fits objects
- Redundant spatial representation



Volumes overlap multiple objects

Space Partitioning

- Tightly fills space
- Redundant object representation

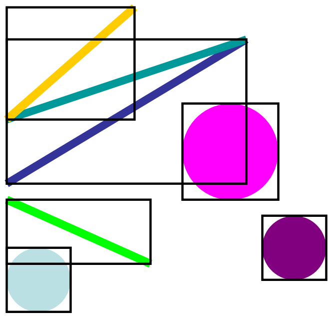


Objects overlap multiple volumes

BVH vs. Space Partitioning
(Kenneth E. Hoff III)

Bounding Volume Hierarchies

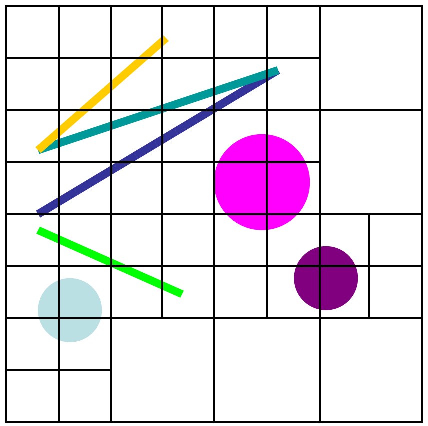
- Tightly fits objects
- Redundant spatial representation



Volumes overlap multiple objects

Space Partitioning

- Tightly fills space
- Redundant object representation



Objects overlap multiple volumes

Hierarchical Partition

- *Masking* (Assarsson and Möller)
  - Considering an object partially inside the VF, then the child nodes must be tested.
  - If the object is completely on the inside of a plane, then ...
  - => it's child nodes will also be on the inside of the same plane, i.e. the plane does not need to be tested.

References

- Fast Backface Culling using Normal Masks, Zhang and Hoff

- View Frustum Culling Tutorial,
  - http://www.lighthouse3d.com/tutorials/view-frustum-culling/

- Optimized View Frustum Culling Algorithms Ulf Assarsson and Tomas Akenine-Möller