



Displayalgoritmer:
rastrering,
klippning, HSE,
LOD, ...
Grip, DGI, DOA
vt2009

Lars Kjeldahl, några bilder lånade
av Gustav Taxén

Implementation aspects for display of computer graphics

Steps in the rendering process:

- Modeling (definition of objects, e.g. a structure with polygons)
- Geometric processing (coordinate transformations, clipping, hidden surface elimination)
- Rasterization (transformation of object descriptions to pixels)
- Display (display of frame buffer on screen, includes e.g. aliasing)

We will focus on some common algorithms in this process

Objektrum och bildrum

Objektrum

Man arbetar den
matematiska
beskrivningsnoggrannheten

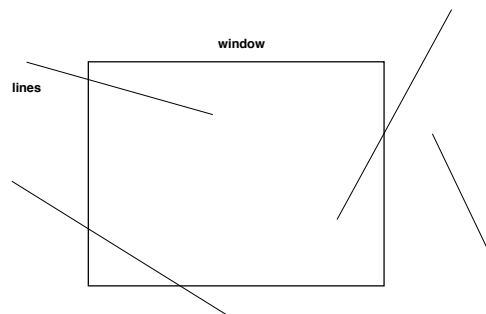
Det blir antalet objekt som
avgör komplexiteten

Bildrum

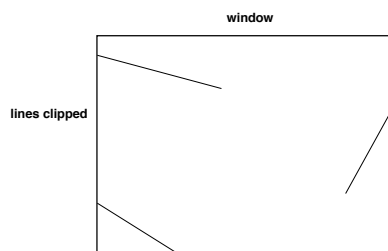
Man arbetar med den
noggrannhet som ges
bildpunkterna (pixlarna)

Det blir upplösningen
som avgör komplexiteten

The clipping problem



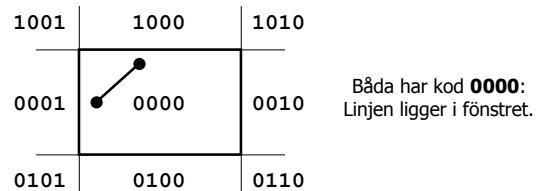
The clipping problem



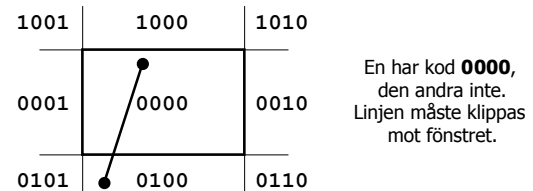
Klippning i 2D: Cohen-Sutherland

1001	1000	1010
0001	0000	0010
0101	0100	0110

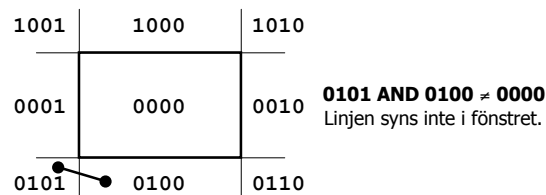
Klippning i 2D: Cohen-Sutherland



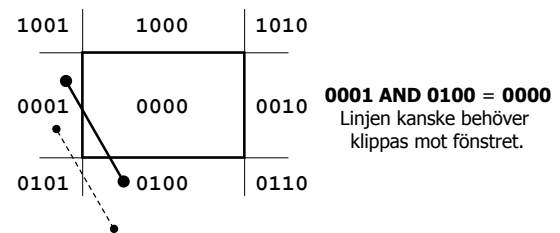
Klippning i 2D: Cohen-Sutherland



Klippning i 2D: Cohen-Sutherland



Klippning i 2D: Cohen-Sutherland



Parameter clipping

(Cyrus-Beck, Liang-Barsky)

Use $p(\alpha) = (1 - \alpha) * p1 + \alpha * p2$

Clipping is done by first calculating α for lines that you might clip (less work than to calculate both x and y)

To clip with the edge $x = x_{min}$ we get: $\alpha = (x_{min} - x1) / (x2 - x1)$

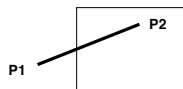
$\alpha1 = 0; \alpha2 = 1;$

for (all four edges of window) {

$\alpha = \text{calculate}(\text{edgenr});$

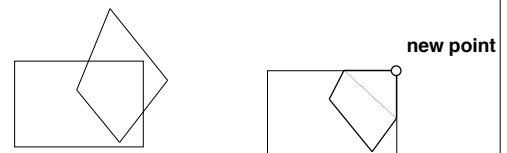
if entering(edgenr) $\alpha1 = \max(\alpha1, \alpha)$ else $\alpha2 = \min(\alpha2, \alpha);$

if $\alpha2 > \alpha1$ then drawline($\alpha1, \alpha2$);



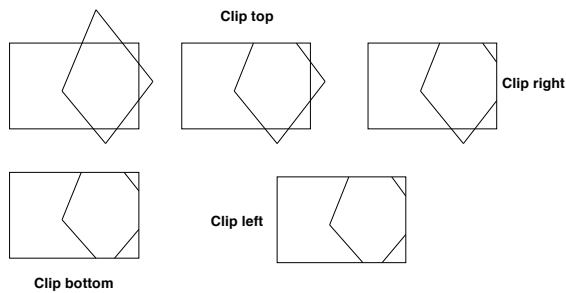
Polygon clipping

Can we use the line clipping algorithm to clip a polygon?



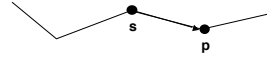
With line clipping the new point could not be included

Sutherland-Hodgeman's algorithm



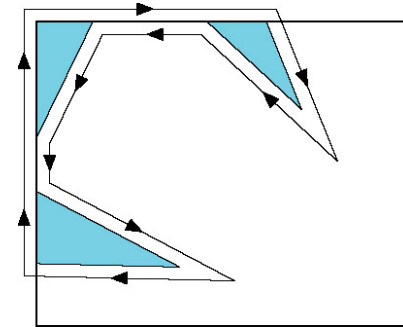
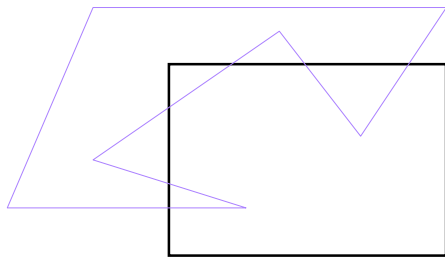
Note that in each clipping phase of the polygon you have to keep the order of the vertices

Sutherland-Hodgeman's algorithm

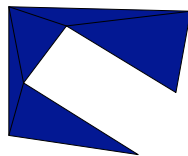


```
for all_window_edges {
  p=polypoint(1);
  for n=2 to n+1 {
    s=p; p=polypoint(n);
    if inside(s) and inside(p) then save(p);
    if inside(s) and outside(p) then save(intersect(s,p));
    if outside(s) and inside(p) then {save(intersect(s,p));
    save(p) }
  }
  update(polypoint);
}
```

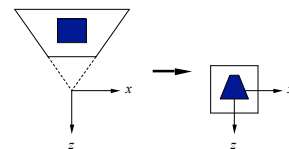
What happens if the clipping give more than one polygon?



Uppdelning i konvexa polygoner
(tessellation)



Klippning i 3D

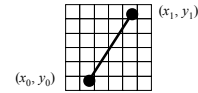


Rastrering



Diskretisering av en matematisk linje eller polygon.

Rastrering av linjer: Digital Differential Analyzer (DDA)

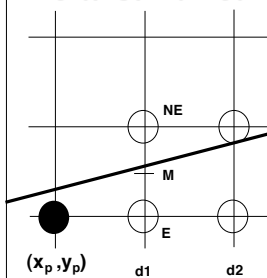


Låt linjens lutning vara $m = (y_1 - y_0) / (x_1 - x_0)$ och antag att $0 < m < 1$.

Sätt $x = x_0$ och $y = y_0$.
För varje $x = x + 1$, låt $y = y + m$.
Avrunda och rita ut pixel.

Fungerar för $m < 1$ Vad gör vi om $m > 1$?

Rastrering av linjer med heltalsaritmetik: Bresenham



Line equation can be written as $F(x,y)=dy*x-dx*y+B*dx=0$

The sign of $F(x_M, y_M)$ decides if we should choose E or NE

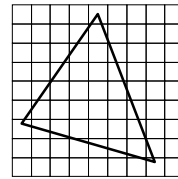
We denote $d1=F(x_M, y_M)=dy*(x_p+1)-dx*(y_p+0.5)+B*dx$

Suppose we get $d1 < 0$, choose E
 $d2=F(x_M+1, y_M)=dy*(x_p+2)-dx*(y_p+0.5)+B*dx=d1+dy$

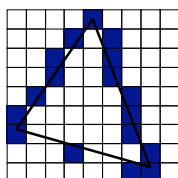
Similar result if $d1 \geq 0$

http://en.wikipedia.org/wiki/Bresenham's_line_algorithm

Rastrering av polygoner



Rastrering av polygoner

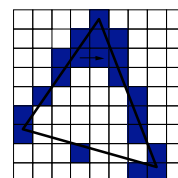


Bresenham
för att hitta x -värdena.

Lagra interpolerade
värden längs med kanterna.

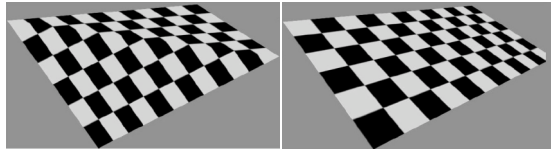
Sortera m.a.p. x -värden.

Rastrering av polygoner



Fyll och interpolera
värden från vänster till höger.

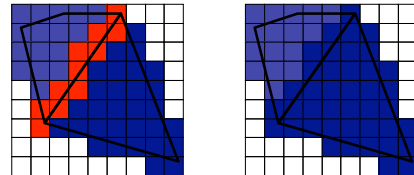
Rastrering av polygoner



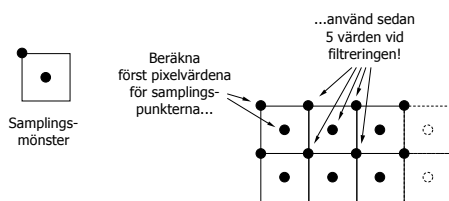
Linjär interpolation

Perspective-correct

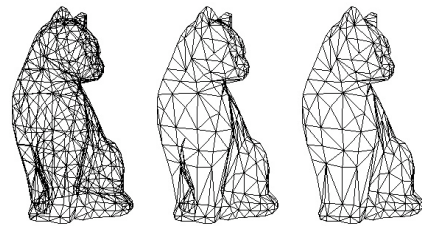
Rastrering av polygoner, särskild hantering av kanterna kan ibland behövas



Multi-Sample Anti-Aliasing (MSAA)



Borttagning av skymda ytor



Hidden surface removal

Backface culling

Hidden surface removal

<http://medialab.di.unipi.it/web/IUM/Waterloo/node70.html#SECTION00116000000000000000>

Hidden surface removal

Two main approaches:

- investigate visibility for every pixel, i.e. resolution dependent (image space)
- investigate visibility for every object, i.e. calculate intersections etc for all objects, lines etc with the precision of the computer (object space)

Main ideas

Try to use the properties the objects, such as:

- **convex objects?**
- **few objects, perhaps only one object?**
- **objects non intersecting?**
- **static view?**
- **advanced rendering wanted?**

Back-face removal

The back-facing polygons can be removed through a simple test
 $\text{abs}(\theta) < 90$ or $n \cdot v > 0$

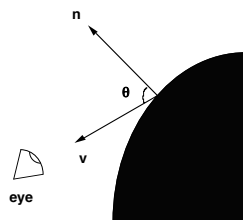
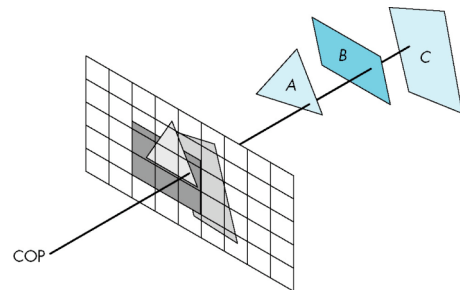


Image space hidden surface removal



Z-buffer algorithm



Z-buffer algorithm

Use an extra buffer with depth values for each pixel

```
for every polygon {
  for each pixel in polygon {
    if newpolygonddepth(x,y) < depthval_in_Zbuffer(x,y)
    then {depthval_in_Zbuffer(x,y) <-
      newpolygonddepth(x,y);
      framebuffervalue <- newpolygonvalue }
  }
}
```



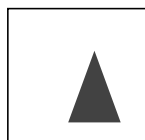
framebuffer



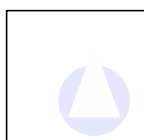
Z-buffer



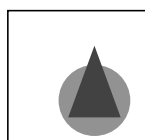
framebuffer



Z-buffer



framebuffer



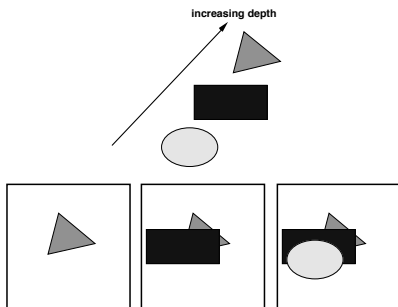
Z-buffer

Z-buffring

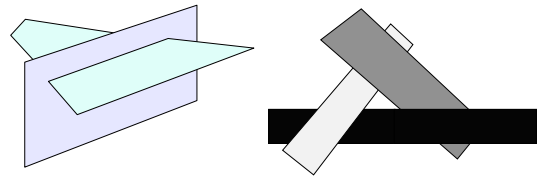
- Fördelar:
 - Effektiv.
 - Funkar oberoende av komplexiteten i det som ritas.
 - Enkel att implementera i hårdvara.
- Nackdelar:
 - Aliasing! 8 bitar $\Leftrightarrow$ 256 distinkta djupvärden.
 - Krävs extra minne på grafikortet.

Depth sort algorithm

1. Sort all object according to depth
2. Paint (rasterize) them into frame buffer in the sorted order (start with object far away)

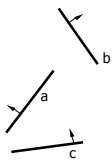


Problem med painter's algorithm



Vi måste dela åtminstone någon polygon

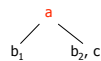
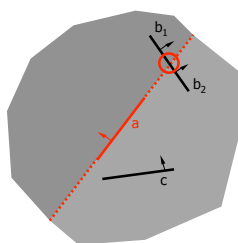
Binary Space Partitioning Trees (BSP-träd)



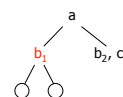
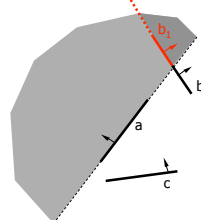
Löser sorteringsproblemet för statiska, plana, konvexa polygoner (som dock kan ha hur många hörn som helst).



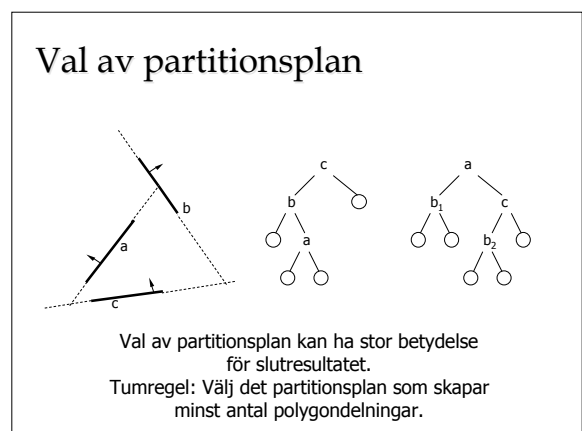
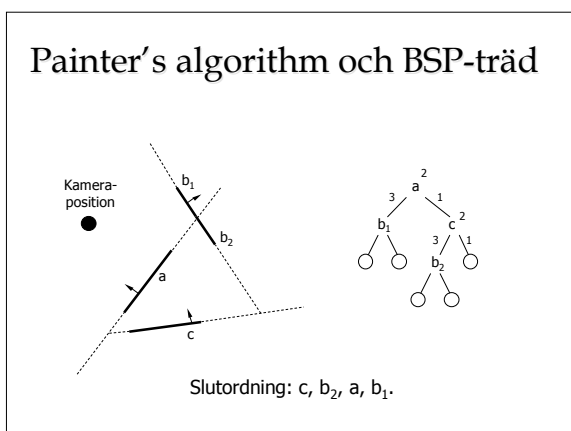
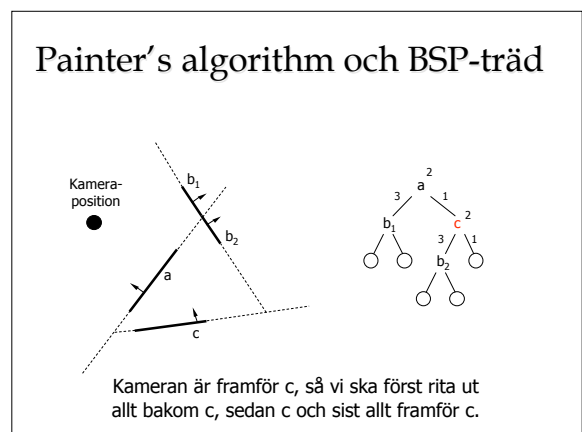
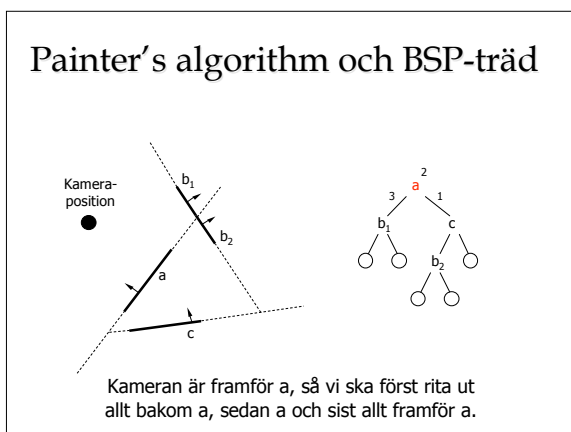
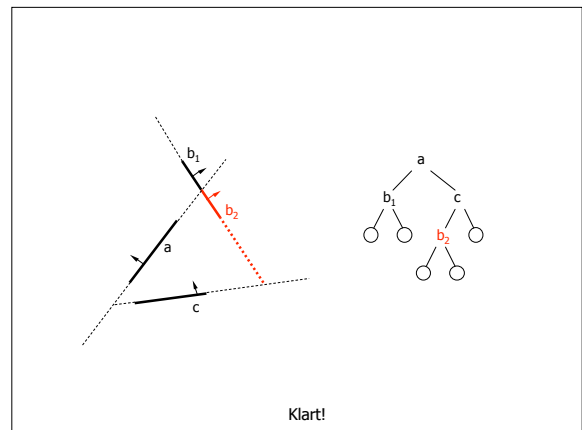
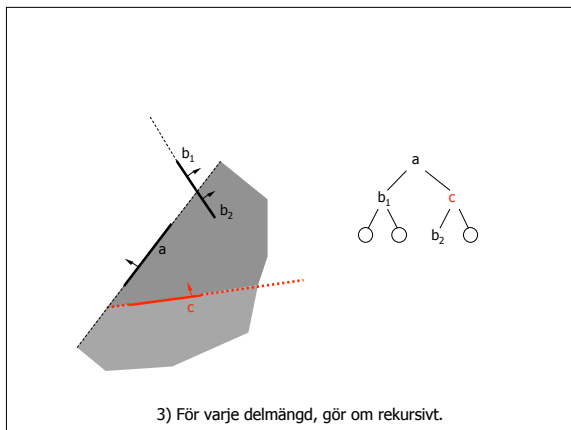
1) Välj en polygon.



2) Dela rummet i två delar och partitionera.



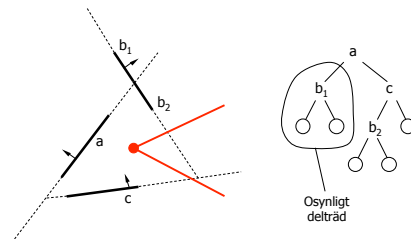
3) För varje delmängd, gör om rekursivt.



Overdraw

- Med painter's algorithm kommer polygoner långt bak att ritas över av polygoner längre fram.
- Betyder att ljussättning, texturering, m.m. appliceras i onödan för de flesta polygoner!
- Detta problem kallas för "overdraw".
- Så varför används BSP-träd om man lika gärna kunde använda z-buffring för att undvika overdraw?

Kombination av BSP-träd och Z-buffring



Eftersom kamerans frustum inte korsar a:s plan, kan vi strunta i allt som ligger framför a!
Använd sedan z-buffring för att undvika overdraw.

Detail culling, Level of Detail

Rita inte saker som är långt bort eller förenkla dem!
Kombineras ofta med någon form av dimma-effekt.



Ultima IX – Ascension, Origin Systems



Morrowind, Bethesda Softworks

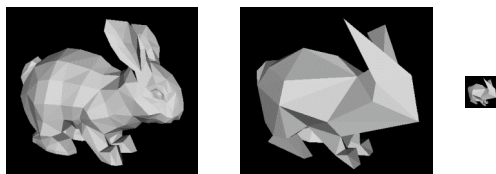
Level-of-detail



Just Cause – Avalanche Studios

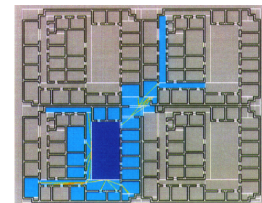
Även om man måste rita många objekt kan man ofta spara resurser genom att använda grövre detaljnivåer långt bort från kameran.

Level-of-detail



Potentially Visible Sets / Portaler

- Närliggande variant:
- Dela världen i celler (kanske m.h.a. BSP-träd).
- För varje cell C, ta reda på vilka andra celler som kan synas från C. Lagra i en lista.
- Vid körning, ta reda på vilken cell kameran är i. Rita den cellen. Rita alla celler i PVS:en.



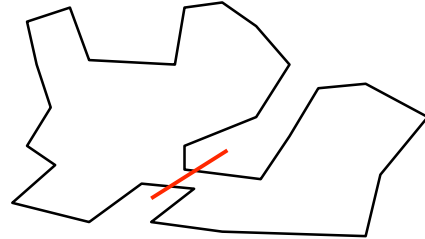
<http://graphics.lcs.mit.edu/~seth/pubs/pubs.html>

Exempel



<http://www.cs.virginia.edu/%7Eluebke/publications/portals.html>

Portaler



Portaler kan specificeras automatiskt med hjälp av algoritmer eller placeras ut för hand i en 3D-editor.

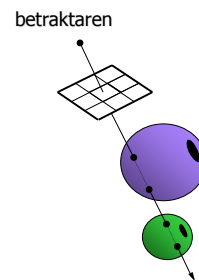
Portaler



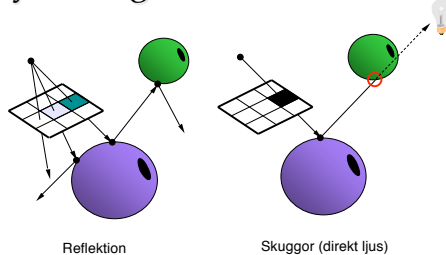
Quake II, Id Software

Ray casting

- Skicka en riktad linje (ray) från kameran (betraktaren) genom varje pixel och ta reda på om/var den korsar alla föremål.
- Välj den korsning som ligger närmast kameran.
- Sätt pixelfärgen till färgen på motsvarande föremål.

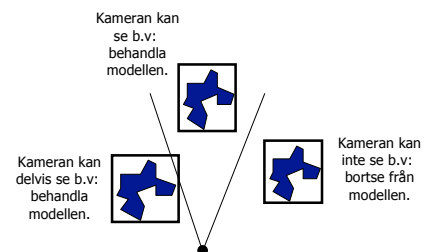


Ray tracing



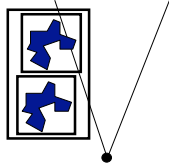
Ray tracing är en s.k. **global belysningsmodell** och kan ses som en slags utbyggd ray casting

Bounding volumes



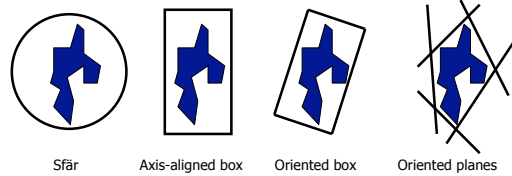
Användbart för modeller med begränsad utsträckning i rummet och för saker som flyttas omkring.

Hierarkiska bounding volumes



Hierarkier kan också skapas dynamiskt under körning för föremål som flyttas omkring.

Exempel på olika sorters bounding volumes



Sfär

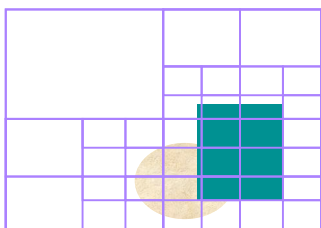
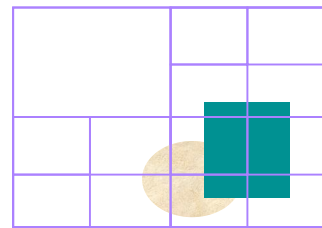
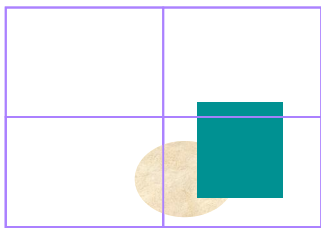
Axis-aligned box

Oriented box

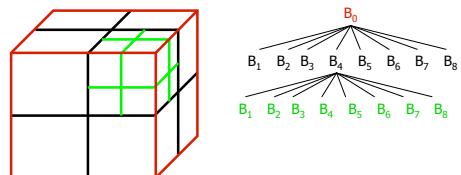
Oriented planes

Warnock algorithm

Divide the plane into four squares



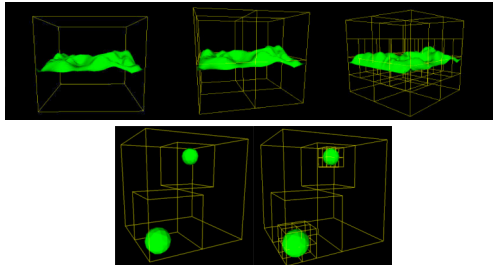
Quadtrees / Octrees



Variant av BSP-träd som kan vara enklare att hantera.
Smidiga för stora, statiska modeller med stor detaljrikedom.

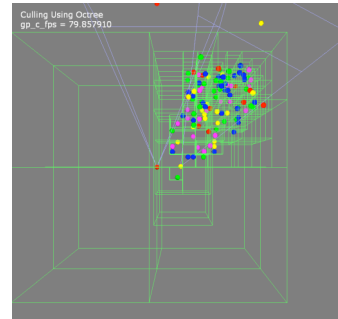
Kombineras med Z-buffring på ungefär samma sätt som i BSP-fallet.

Octrees

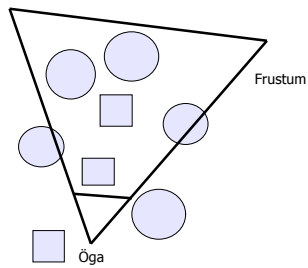


<http://www.gametutorials.com/Tutorials/OpenGL/Octree.htm>

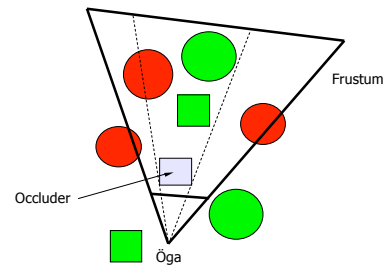
Octrees



Occlusion culling

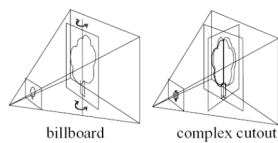


Occlusion culling

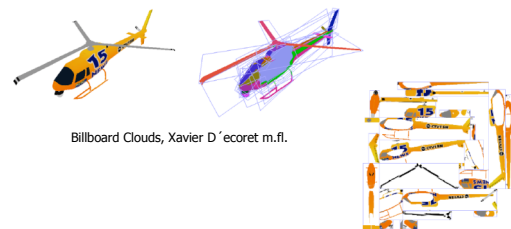


http://www.gamasutra.com/features/19991109/moller_haines_01.htm

Impostors



Impostors



Placerar man texturen smart kan man approximera föremål på ett sådant sätt att man kan rotera dem också!

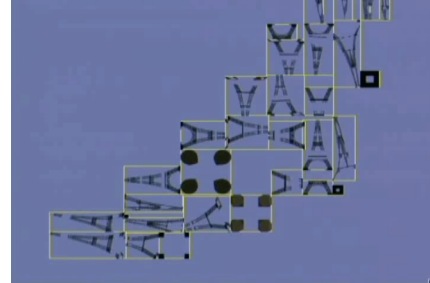
Impostors



Original

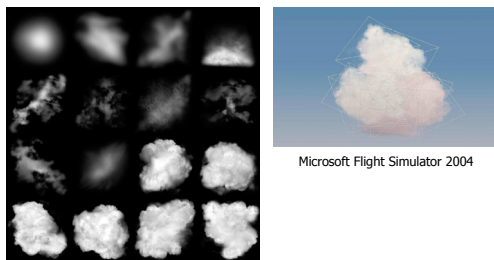
Billboard cloud

Impostors



Billboard Clouds, Xavier D'ecoret m.fl.

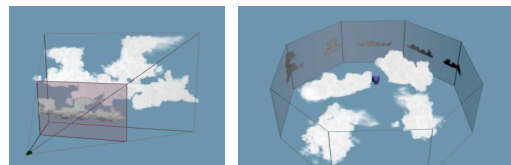
Impostors



Microsoft Flight Simulator 2004

Moln är ett exempel på objekt som ofta kan ersättas av impostors.

Impostors



Microsoft Flight Simulator 2004

I flygsimulatorfallet lönar det sig att arbeta med impostors i två steg: dels som ersättning för 3D-moln, dels för att approximera moln långt från kameran.