Detection of visible surfaces (revisited)

Backface culling
Painter's algorithm
Z-buffer
Ray-casting
Portals

A-buffer
Scan-line method
BSP trees
Area subdivision
Octrees

More than just getting the right pixel in the right place:

Maximize performance by

- **discarding polygons that will not be visible due to:**
  1) Clipping to viewing frustum
  2) Back-face culling
  3) Other possibilities?

- **drawing pixels only once (or as few times as possible)**

- **don’t make many-to-many checks between polygons!**

Low-level VSD

Works on pixel level or polygon level.

Always process all polygons in the scene.

Can never be sufficient for very large scenes!

covered so far:
Low-level VSD

(visible surface detection)

Backface culling
Painter's algorithm
Z-buffer

Not covered yet:
Scan-line method
BSP trees

All polygons are treated individually

Good for small scenes
(small total number of polygons).

Clipping

Part of the OpenGL pipeline

Hardware supported

Clips away any part of a polygon that is outside the viewing frustum

One side at a time

Still individual polygons!

High-level VSD

Large scenes, large or very large polygon count.

Only a small part of the scene is visible at a given time!

Process polygons in groups, with some kind of spatial information! Remove many polygons with each decision.

BSP trees (revisited)
Octrees
Domain-specific culling
Portals
PVS

Types of “visibility processing” algorithms:

- Exact algorithms
- Approximate algorithms
- Conservative algorithms

Exact:

Finds all visible or partially visible polygons

Drawback: Usually too computationally expensive

Approximative:

Finds most visible polygons, excludes most invisible ones

Fast. Some artifacts.

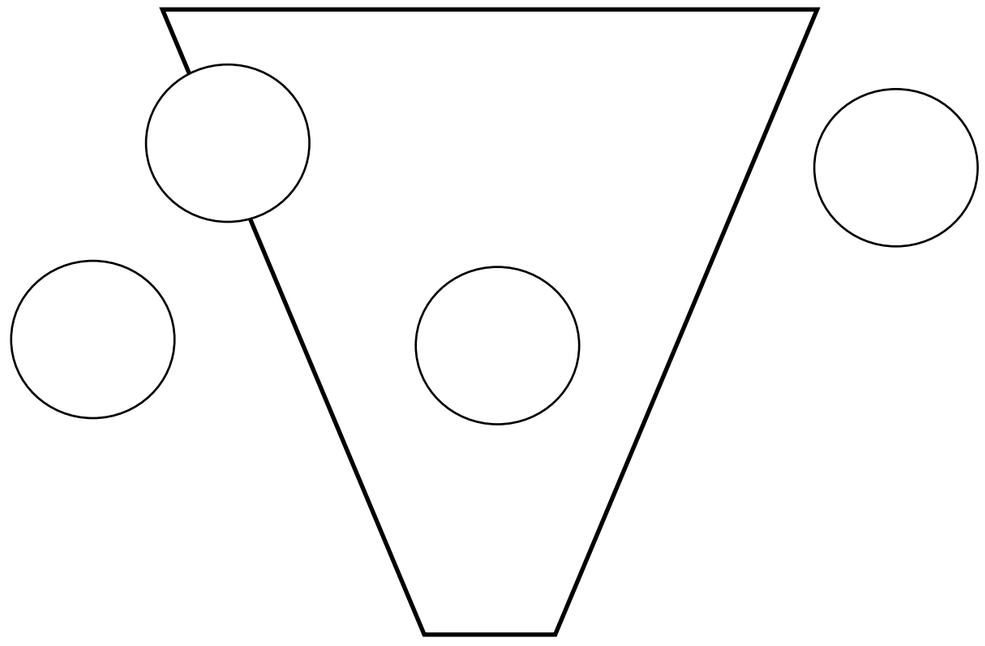
Conservative:

Finds all visible polygons but includes some invisible ones.

No artifacts. Potentially lower performance than “approximative”

Step 1. Frustum culling (View volume culling)

What polygons are inside the frustum?



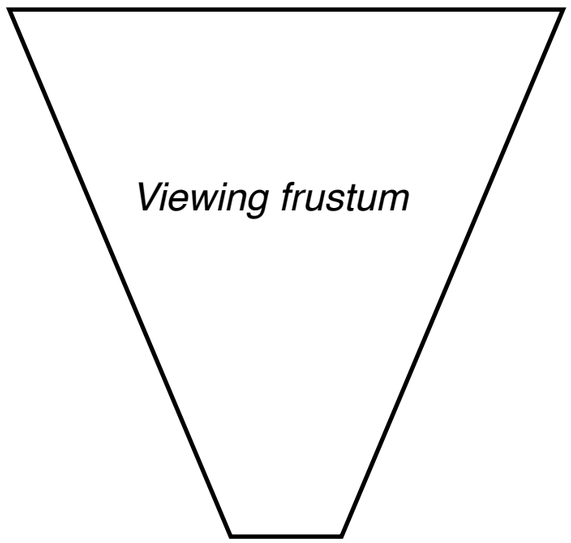
Principle: Make a subdivision of the scene, so tests can be done on groups, e.g. separate objects or limited parts of the scene.

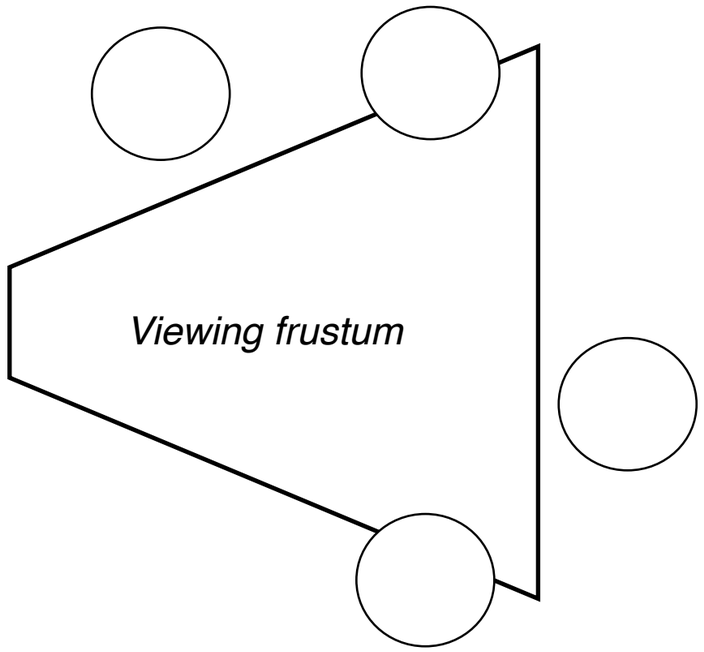
Frustum culling

Create plane equations for each frustum side

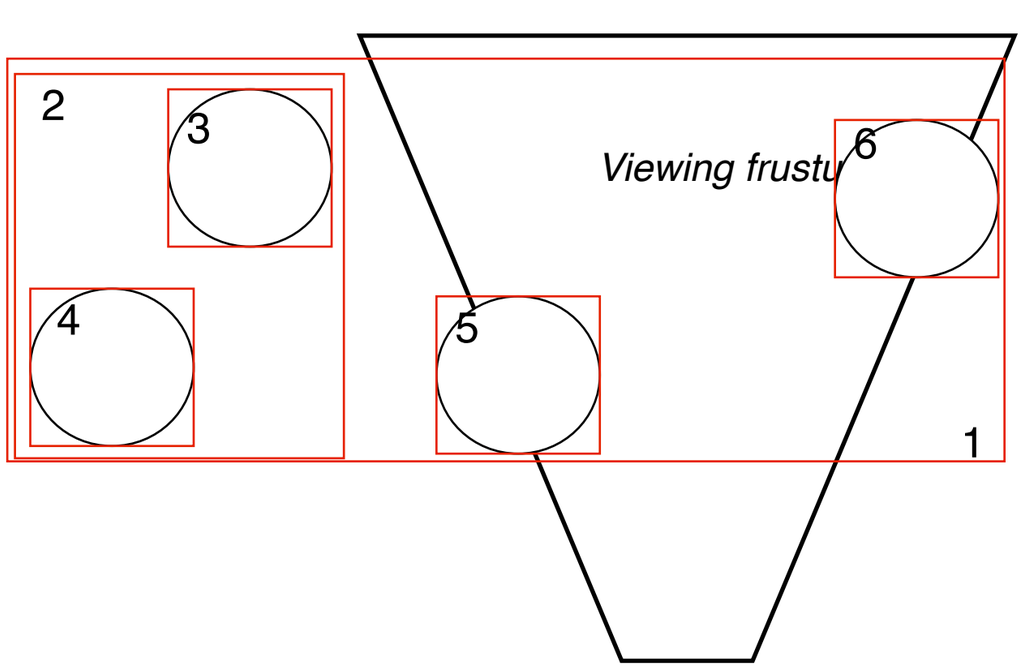
Transform to world coordinates

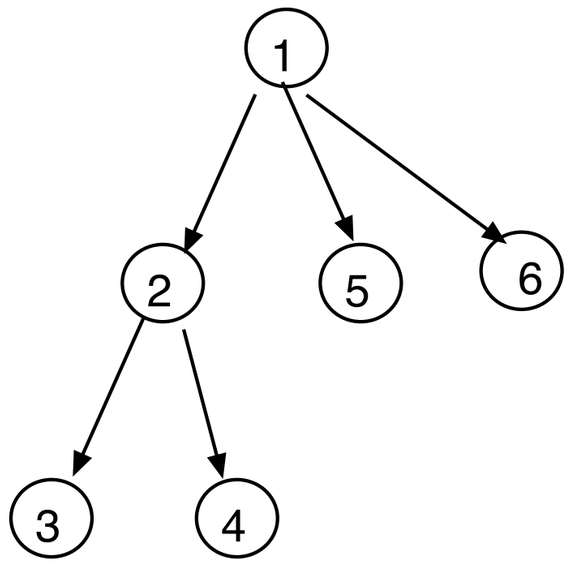
Test against bounding spheres of objects



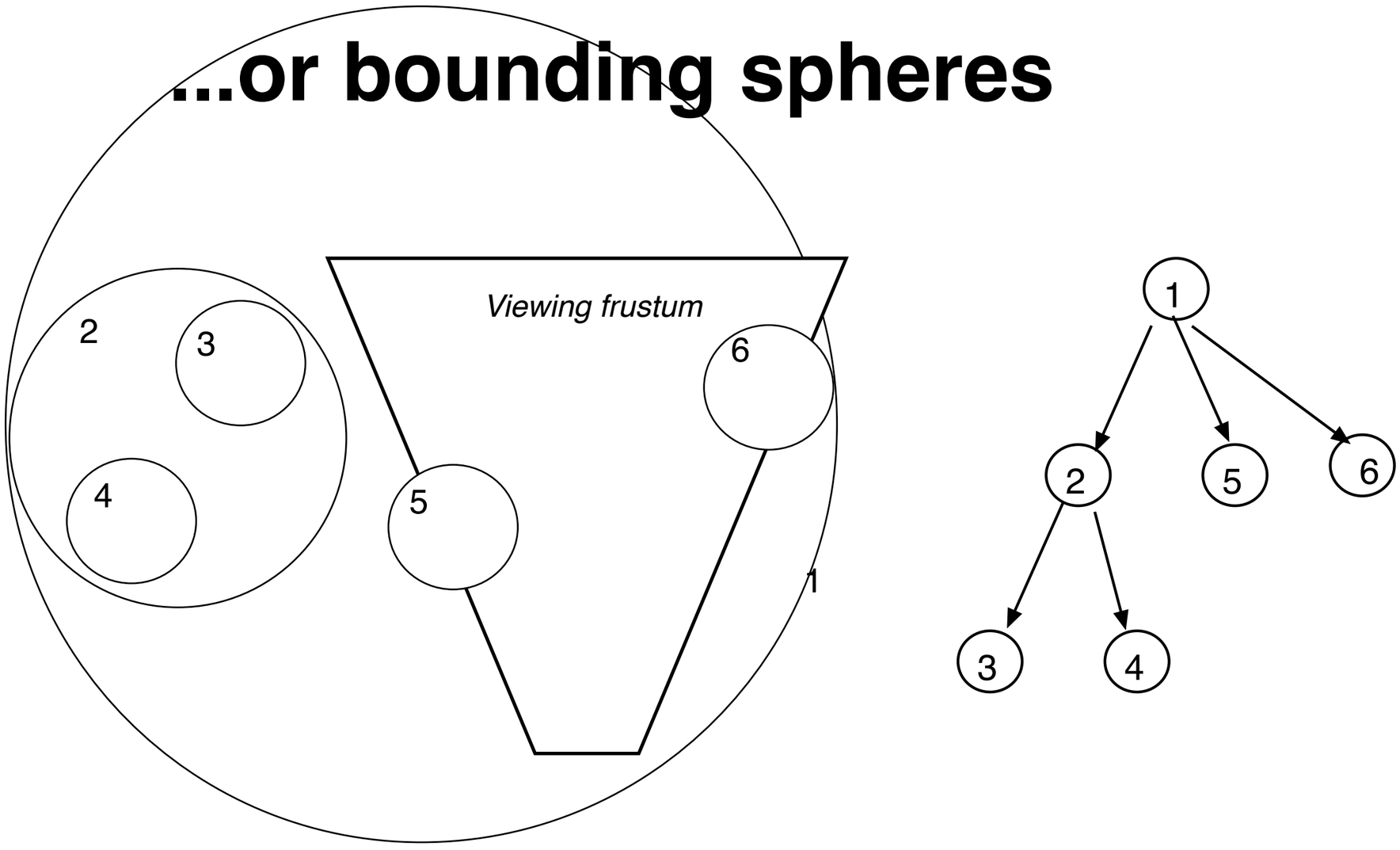


View volume culling using a group hierarchy with bounding boxes





...or bounding spheres



BSP trees for high-level VSD

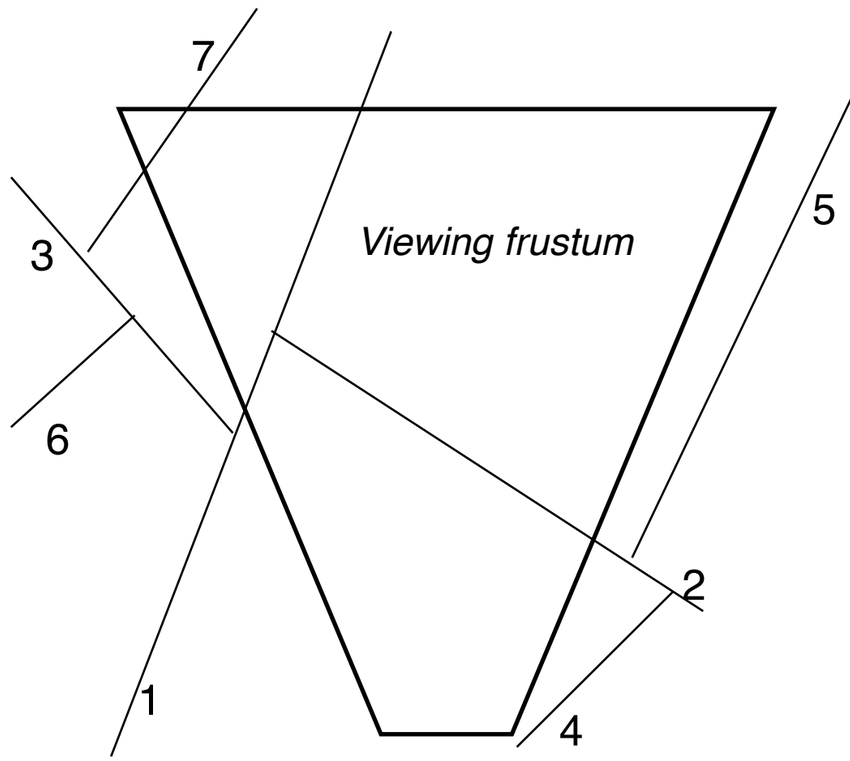
BSP trees simpify frustum culling!

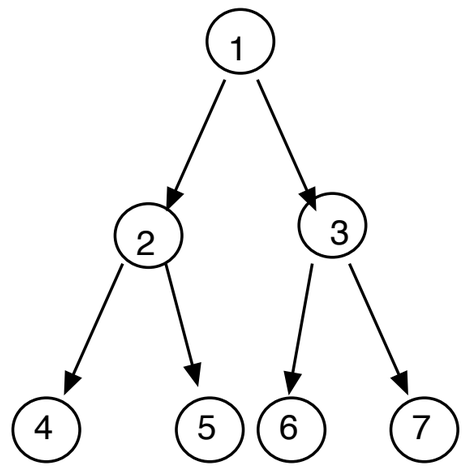
Any node in a BSP tree is a convex volume!

Whenever a volume falls outside the clipping frustum, ALL polygons below that node are removed!

BSP = Binary Space Partitioning

Frustum culling using a BSP tree

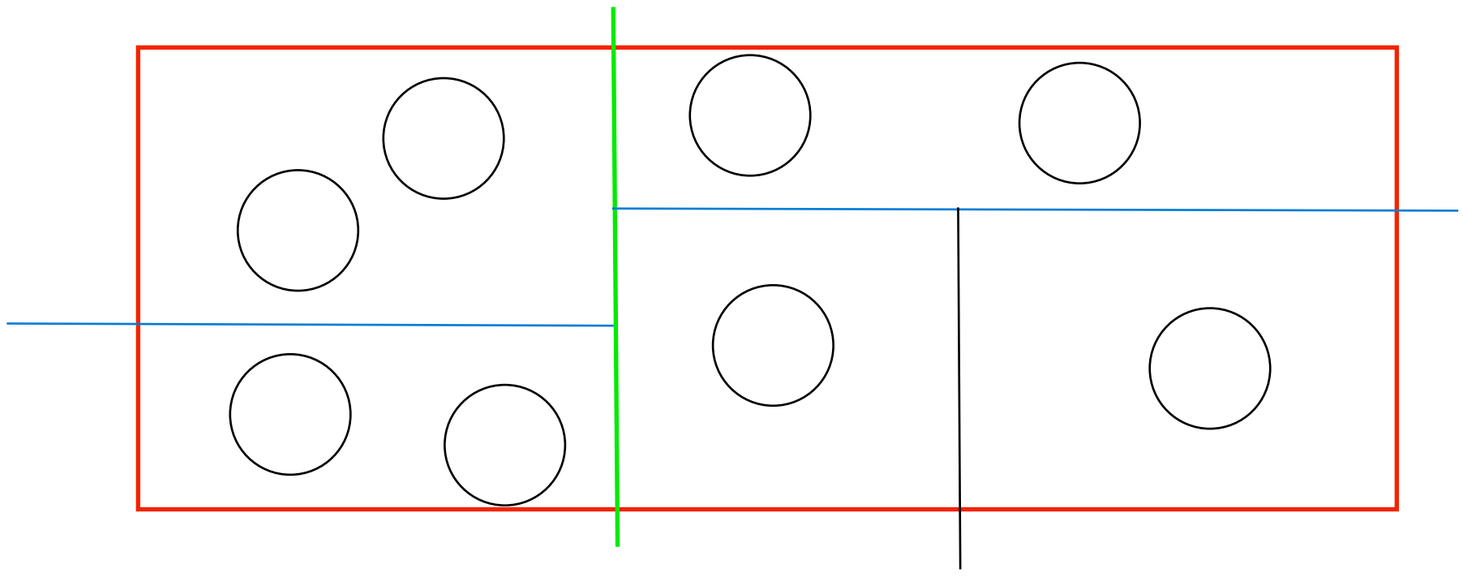




Frustum culling using a BSP tree:

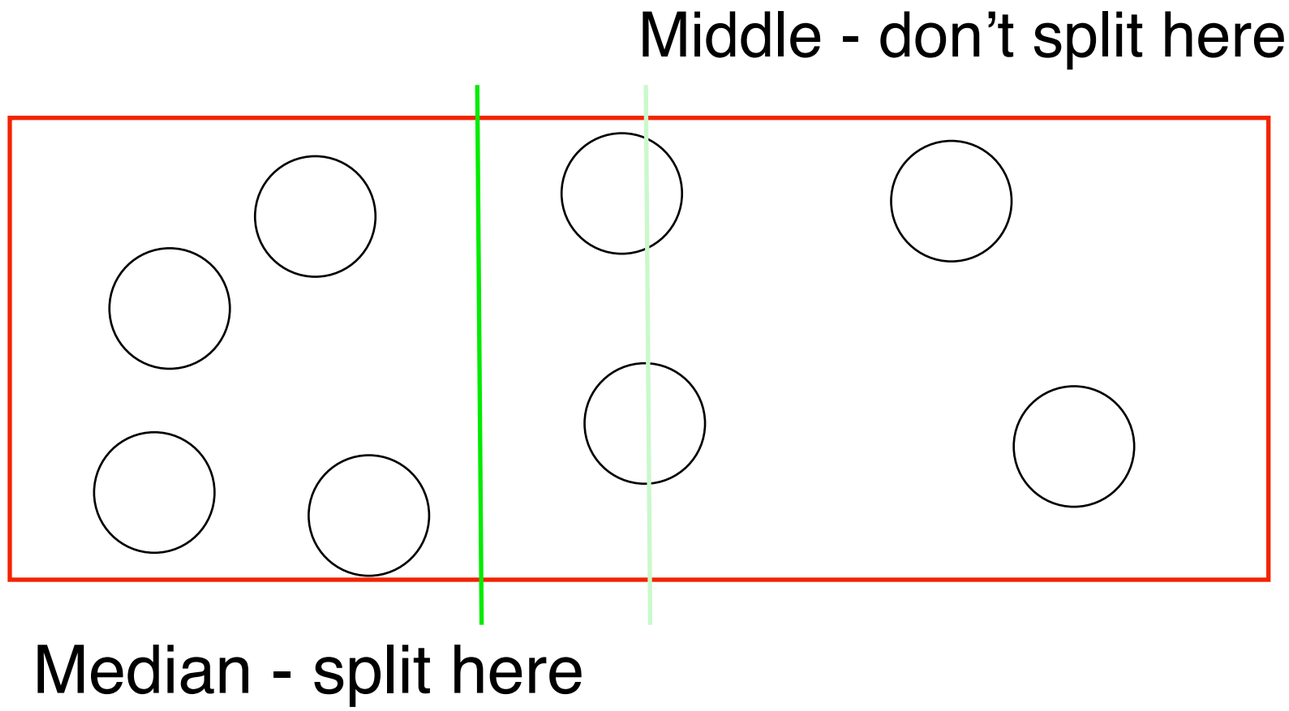
Usually axis aligned - ”kd-tree”

Very simple tests



Building a kd-tree

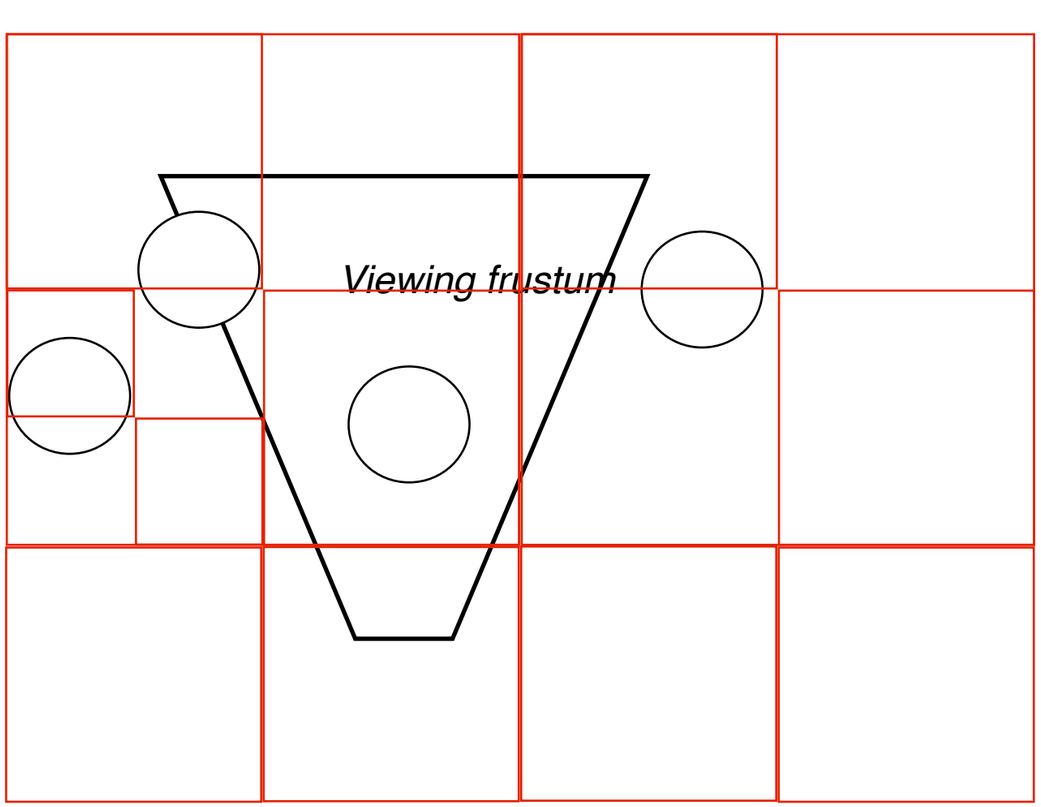
Split at median: Half of the geometry in each side of the splitting plane. Balanced kd-tree.



Octrees:

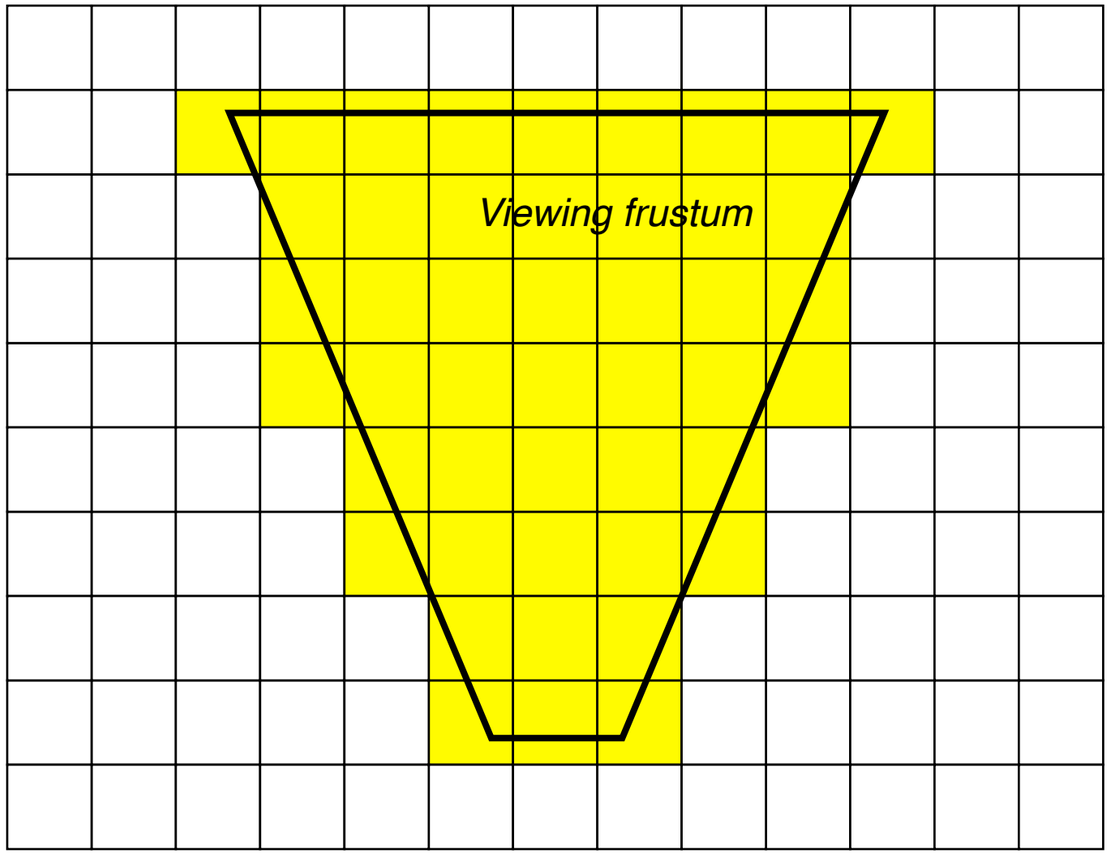
Non-uniform hierarcical space subdivision

Split cells in 8 until sufficient simplicity is achieved.



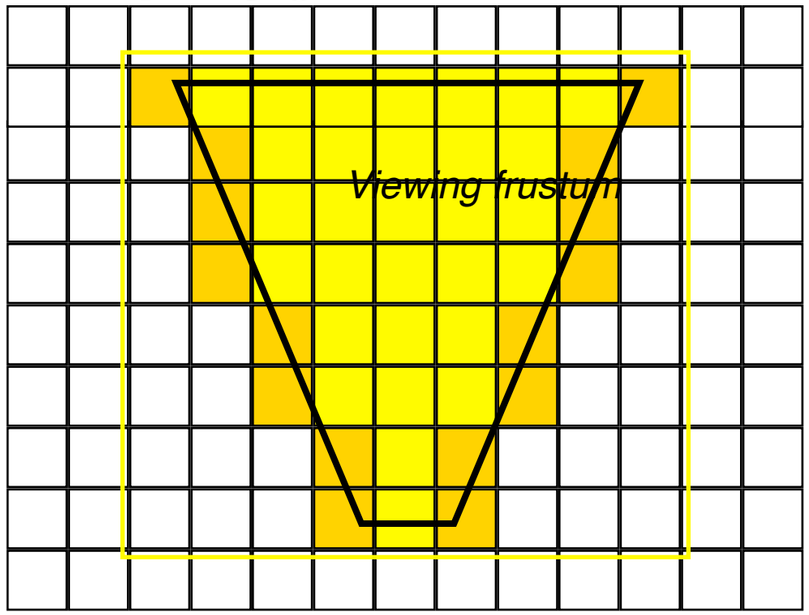
Uniform space subdivision

Simple common case: Terrain defined by a regular grid



Map the frustum edges to the grid coordinates

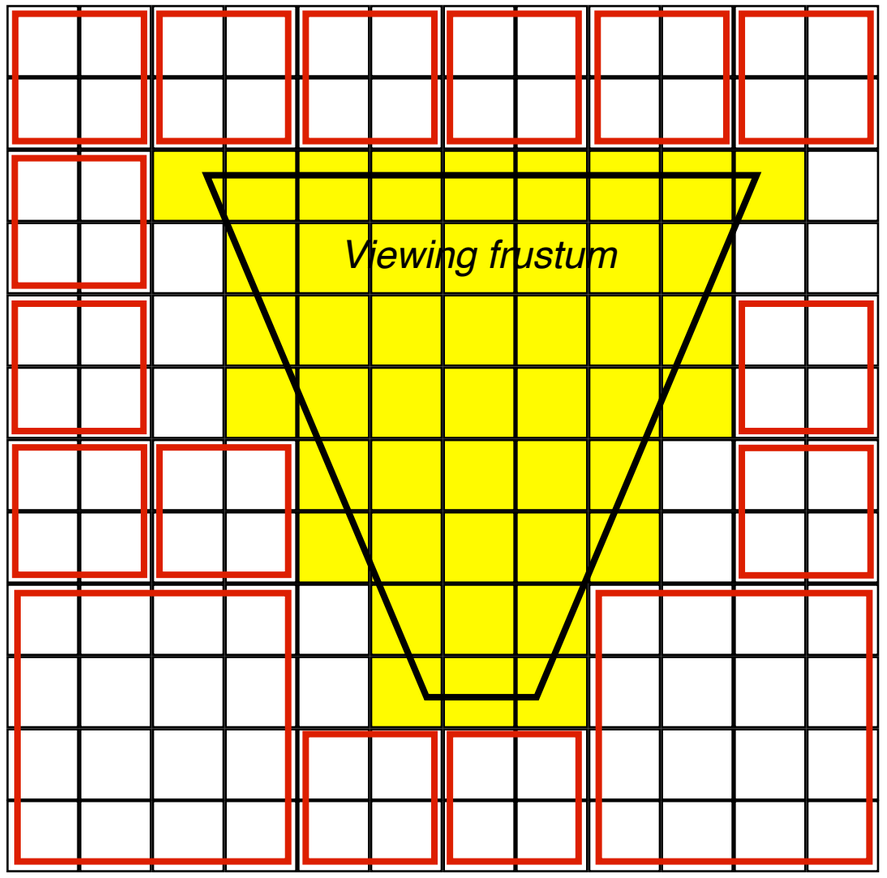
Draw all polygons between edges



Cheap quick hack version:

Find the bounding box of the frustum. Gives a simple 2D rectangle with grid spaces to draw. Up to 50% unnecessary polygons.

Grid, alternative approach: quadtree



Real-world example: Bugdom series

Fairly sparse environment, frustum culling is sufficient.