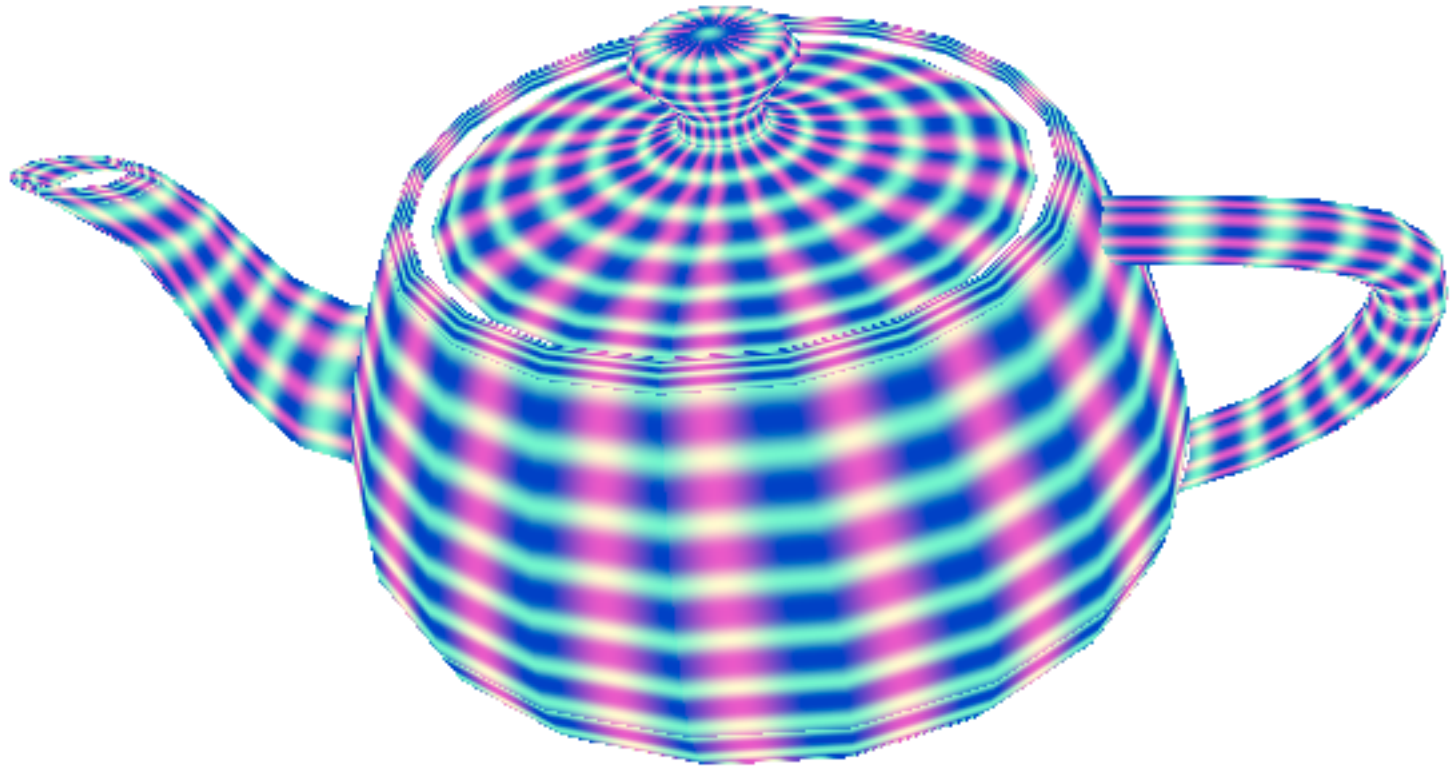
# TNM084
# Procedural images

Ingemar Ragnemalm, ISY



# Lecture 8

More on terrains

Geometry shaders

Tesselation shaders

Grass

## Lab 3

I think most are more or less done with the tree generation.

Using GLUGG for the tree generation seems to work just fine. A bug affecting Linux has been found in the terrain part and will be fixed ASAP.

You needed to figure out how to deal with recursion as well as using a matrix stack.

# Extensions on Lab 3

What can you do beyond the lab?

Extended terrains: Bigger terrains, more features for the terrains

Extended tree generation: Tropism, organic growth, more parameters, different tree types

## Projects

It is time to submit project proposals!

I hope I have given feedback on all proposals.

## Lecture questions

1. How many polygons do you test for frustum culling?

2. Do you do frustum culling in view or world coordinates?

3. How can you avoid aliasing in terrains?

4. When do you need adjacency in geometry shaders?

5. Why do we have different resulution for different edges with tesselation shaders?

## More on multi-patch terrains

Frustum culling

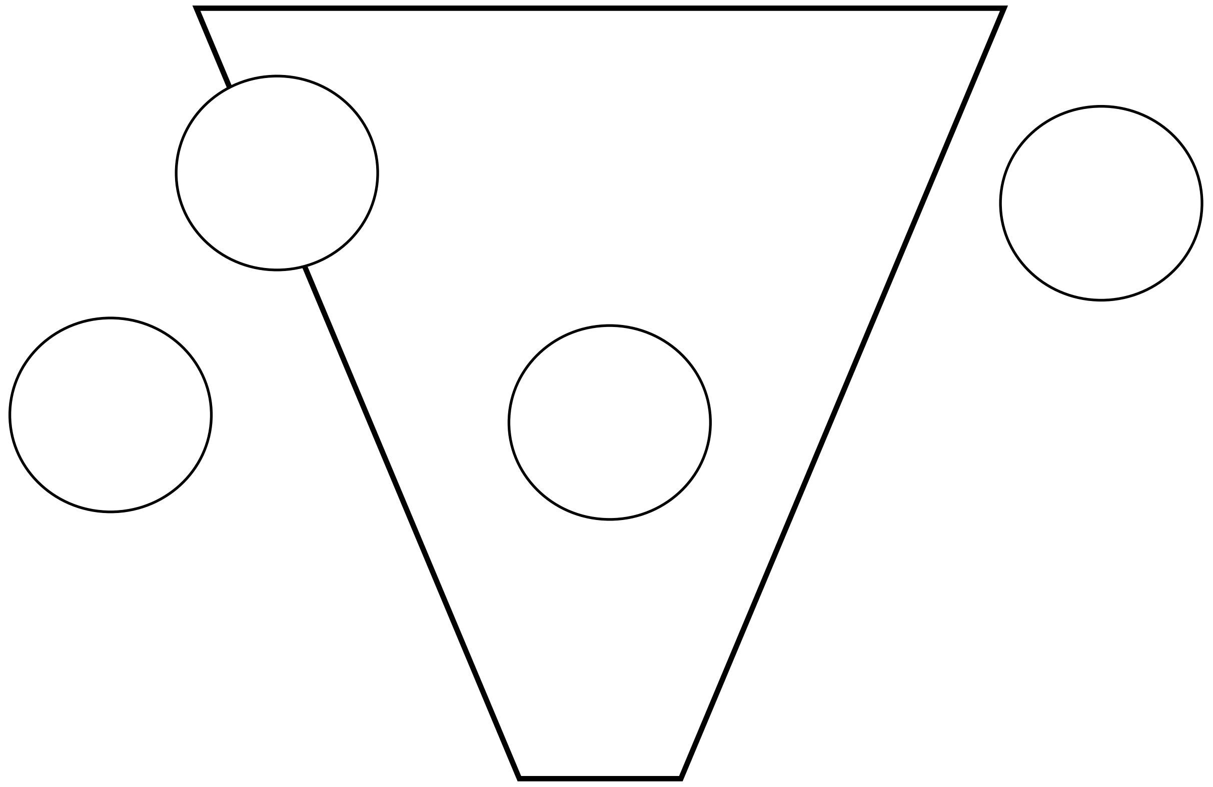
Level-of-detail

Using a repeating patch to fake it

Generating patches on the fly

## Step 1. Frustum culling (View volume culling)

### What polygons are inside the frustum?



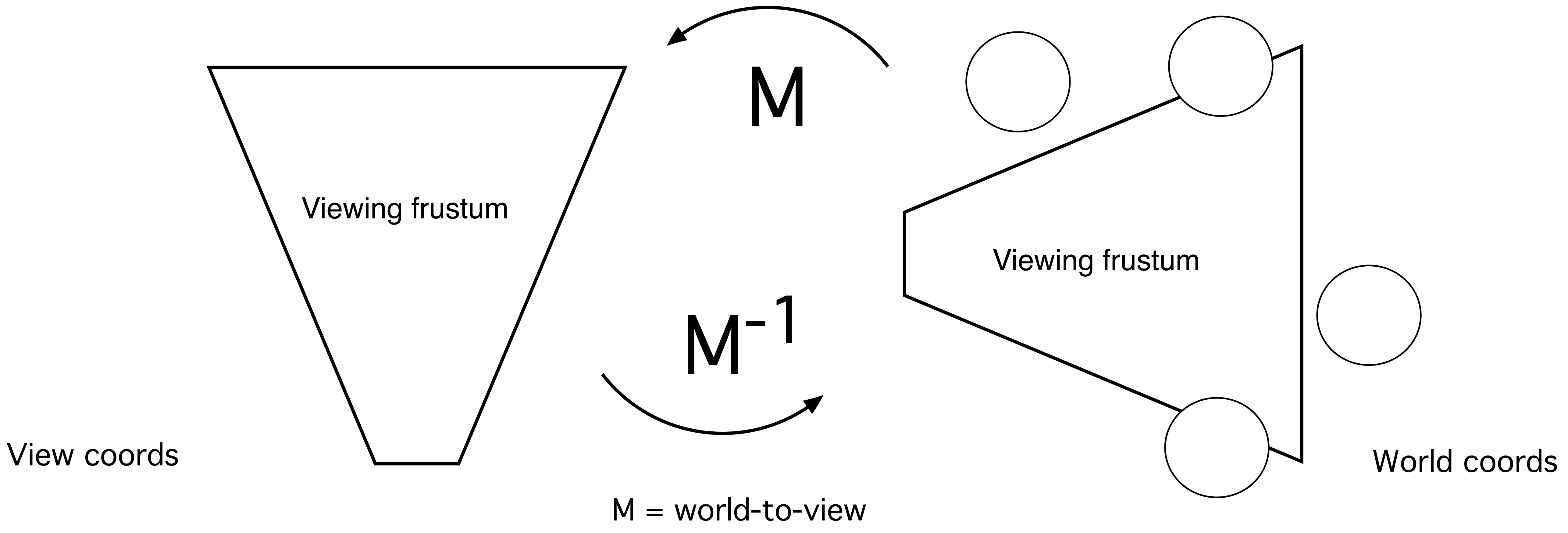
Principle: Make a subdivision of the scene, so tests can be done on groups, e.g. separate objects or limited parts of the scene.

## Frustum culling

Create plane equations for each frustum side

Transform from view to world coordinates

Test against bounding spheres of objects

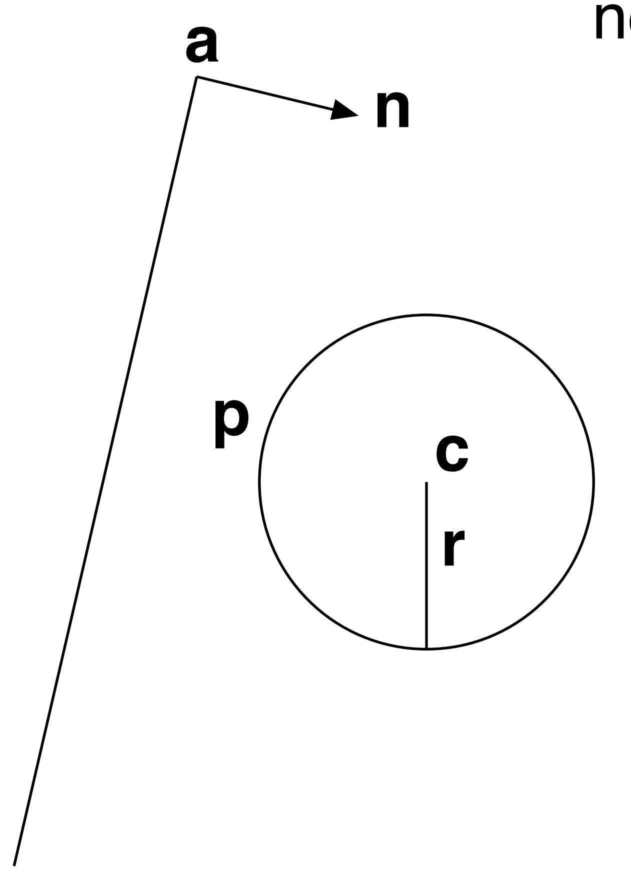


M = world-to-view

## The frustum culling test against a sphere

You need

center of sphere **c**
radius of sphere r
normalized normal of plane **n**
A point in the plane **a**



Point p nearest plane = **c** - **n**·r

Project **p** on n by **p•n**

Project **a** on **n** by **a•n**

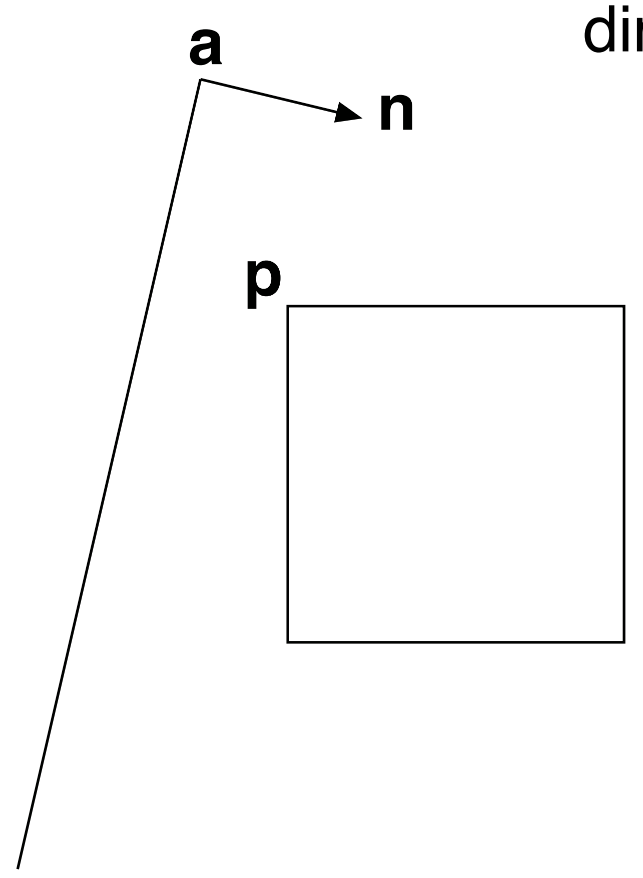
Check sign of **a•n** - **p•n**

## The frustum culling test against an AABB

Relevant for testing square patches!

You need

dimensions of cube, top, left, bottom, right, near, far
normalized normal of plane **n**
A point in the plane **a**



Testing all corners works, but:

Find relevant corner (farthest along -n) from signs of the normal!

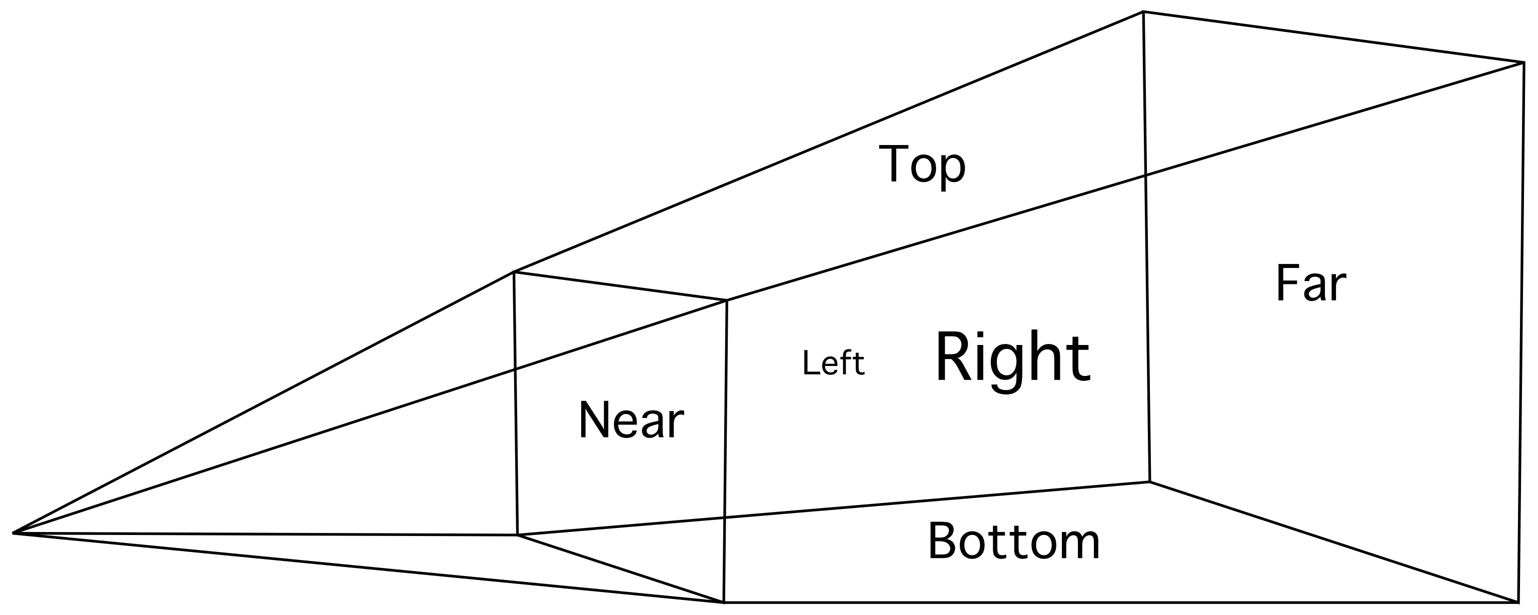
Test that point with dot product like before:
Check sign of **a•n** - **p•n**

## What planes to test?

You never need to test the near plane! (Why?)

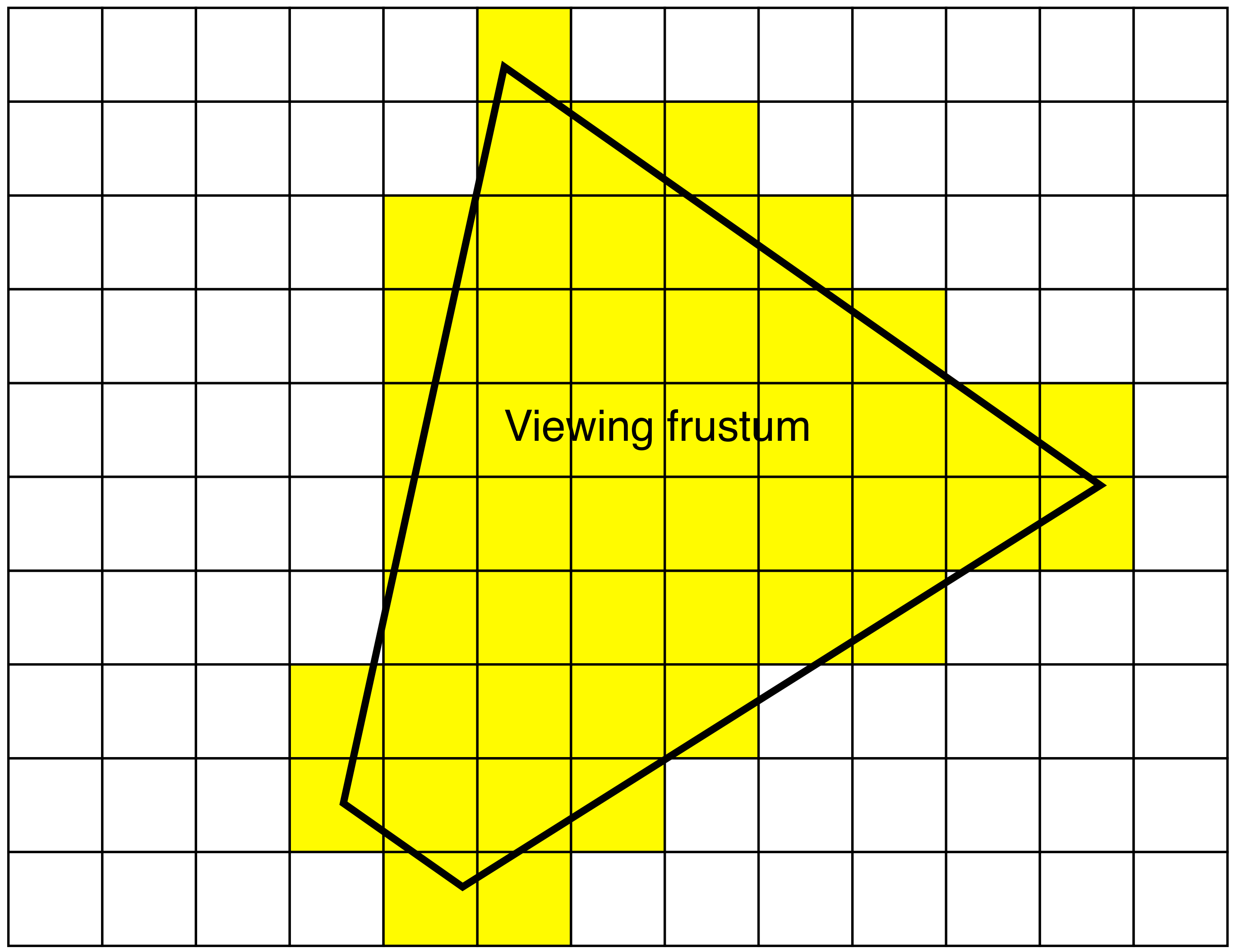
You usually need right, left and far.

Top and bottom are often unnecessary - especially when discussing terrains!



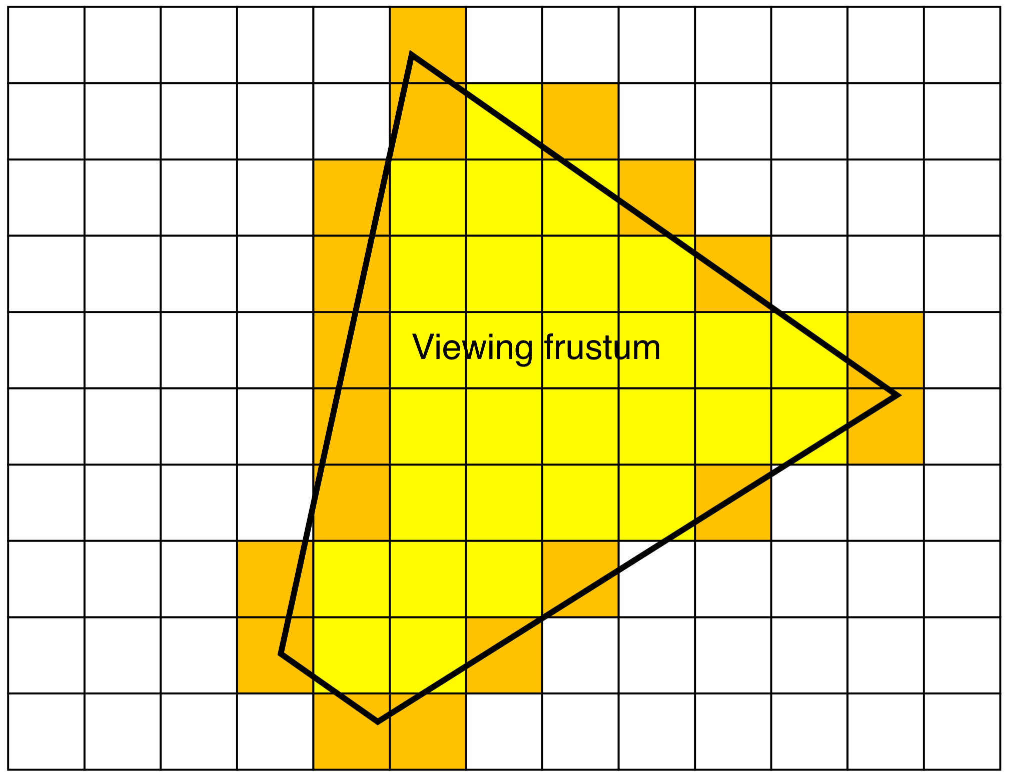
## Uniform space subdivision

Simple common case: Terrain defined as multiple patches of same size; forms a regular grid



## Map the frustum edges to the grid coordinates

Draw all patches 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.

## Level of detail

Fewer polygons

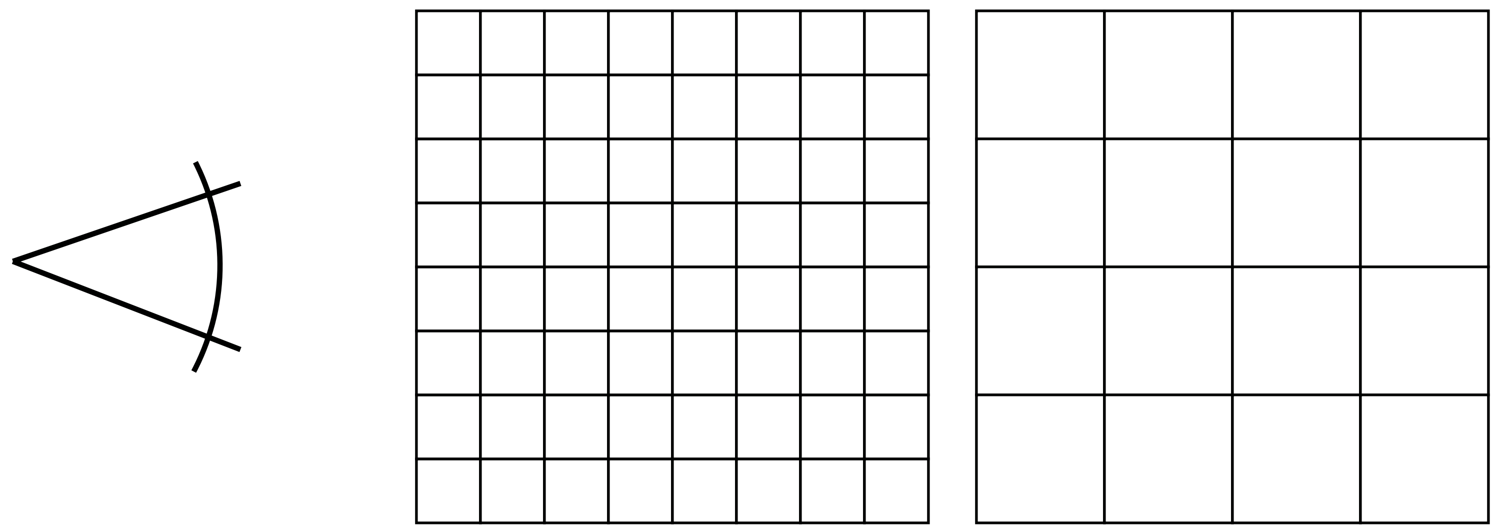
Terrains = Special case

Coarser patch on a distance

"Geometrical mip-mapping"

## Coarser detail on distance

Decide a level to keep constant resolution



Problem: Gaps in edges!

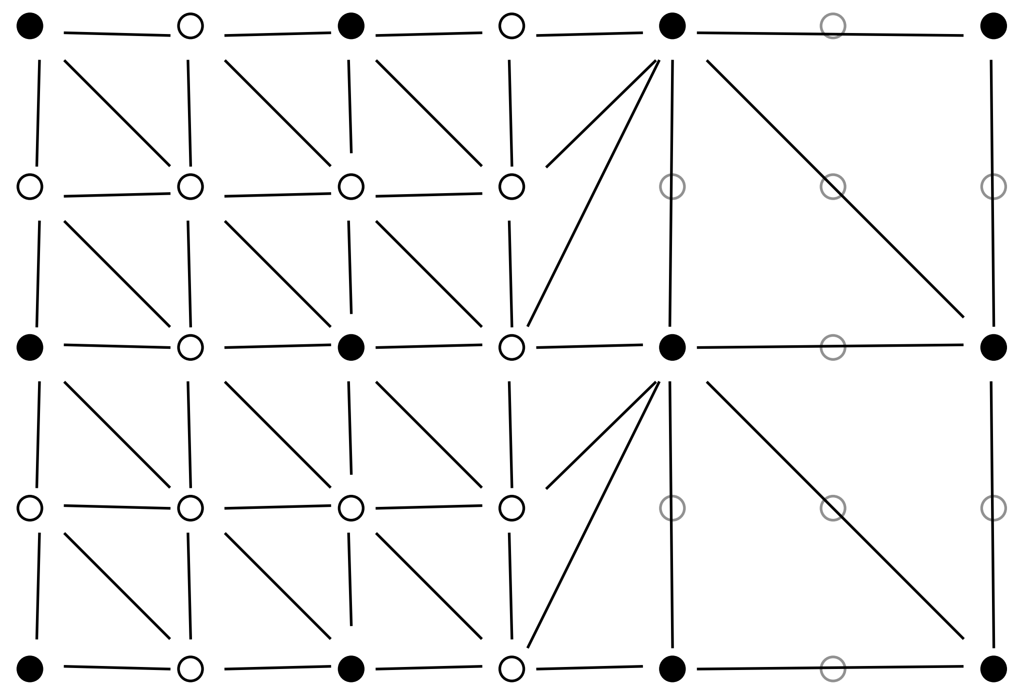
## Avoiding gaps

Two methods:

- Skirts
- Adjusting the terrain near the edge

## Patching edges between different levels

Modify the geometry at the edge to fit the neighbor

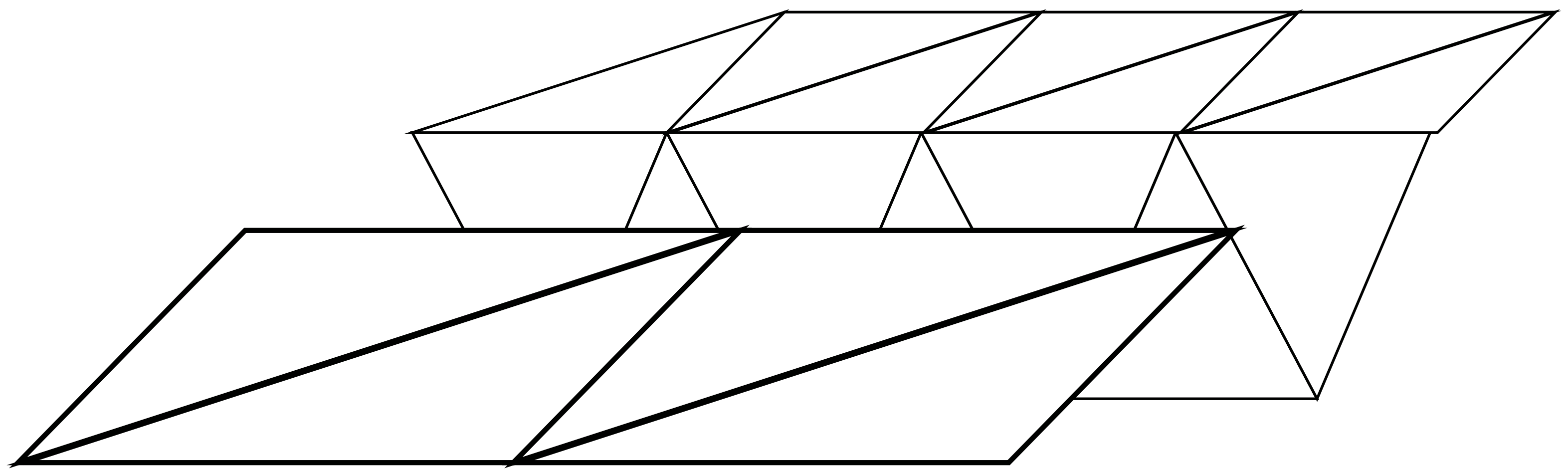


Decisions best taken at edges between patches!

## Patching edges between different levels

Second approach: "Skirts". Insert extra polygons at the edge in a way that will fill the gap.

More visible than adapting resolution, but lets you work with separate patches at different resolutions



## Limitless terrain

Open world-games need to expand arbitrarily far.

Two approaches:

- Generate new patches
- Repeat

## Generate new patches on the fly

Player moved out of the existing area

Generate new patches

Disposing left ones may be desirable for memory management

## Or use a large repeating terrain

Make the world so big that the player is unlikely to notice

or

make the world explicitly wraparound (sphere or torus)

You still want to split the world into patches!

## Level-of-detail and anti-aliasing

Level-of-detail and anti-aliasing are the same!

Limit the frequencies depending on distance!

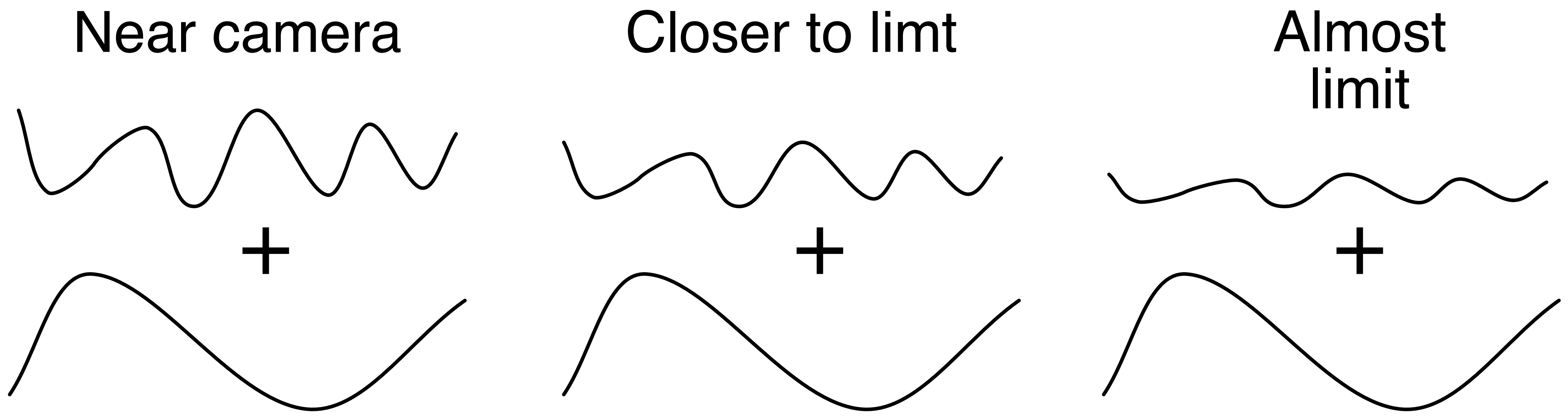
Less geometry = less rendering

No frequencies above the Nyquist frequency!

## Morph your FBM

When doing anti-aliasing on an FBM terrain, you may get the LOD and anti-aliasing by controlling the higher frequencies.

If generating FBM on the fly: Vary the amplitude (gain) of the highest frequency until it reaches zero at an appropriate distance.



## Morph for pre-generated terrain

For a pre-generated terrain, we need to save a resolution pyramid, as above, or save each FBM layer separately.

Question: What is the most work, generate on the fly or pre-generate and patch together? Can we generate on the fly in the long run?