Monsters II:
An Alife demonstrator

Monsters II : An Alife demonstrator

- Jinigrid
- Monsters II
- Monsters I
- Rendering the world
- The virtual world
- Programming problems
- Questions

Jinigrid

- Aim was to provide a grid based in Java using Jini to co-ordinate services and clients.
- To make Jinigrid distributable.
- Problems
  - Makefiles
  - Jar libraries
  - classpaths vs.. codebases
- Solution?

Monsters II

- Herbivore World…
- A virtual world populated by autonomous creatures.
  - Creatures interact with each other and their environment.
  - They breed, and pass on genetic material which codes for behavior and appearance.
- Monsters I …

Creating Monsters II

- Create a rendering engine.
  - Must be flexible enough to accommodate different styles of worlds.
- Design a world biology from the bottom up.
  - Agree on basic nutritional elements synthesized and provided by the plants.
- Design a virtual machine to inhabit the world.
  - Programmable state machine.
  - Sufficiently complex to be interesting and unpredictable

The World

- The world is based upon a grid of squares.
  - Grid must be Z x (*n* x Z) for ease of fractal generation
- Each square has certain attributes:
  - Height(actually has x4 heights and x4 normals)
  - Temperature at the moment
  - Humidity at the moment
  - A flag which contains data about what is located on it.
  - The meaning of two other data items depends on what this first flag contains.

Rendering the world

- Rendering engine uses DirectX 7.0
- First we need to find out what is the minimum data we need to draw. This is dependant on the 'view frustrum' …
- To find the minimum area project the view frustrum down onto the x-z plane. …
- Sort an array of indexes to the renderable data based upon the squares textures. This prevents texture 'thrashing'.
- Render front-to-back when possible(Z-buffer).
- Render translucent materials last.

Depth perception

- Human eye only uses its ability to change focus for depth perception of nearby objects.
- Far objects use visual cues:
  - Light absorbtion
    - Depth fogging (also hides ‘popping’).
  - Parallax
    - Doom ‘bobbing’.
  - Known heights

The virtual world: the terrain

- The terrain is textured with either a base ‘grass’ texture, or a ‘subwater’ texture. It may be overlaid with a secondary texture if it has a ‘plant’ on it.
- Its textures are modulated by a base colour …
- Its base colour is dependant on its temperature,
- which is a function of its distance from the equator,
- its height,
- and any effects due to its plant occupant.
- This should create arctic and desert regions in the world, to which the inhabitants (plant and creature) can adapt.

The virtual world : the plants

- A plant occupies one whole square. It has a species and an age, but is otherwise not individual. There is no instances of 'plants'.
- Some plants are renderable as 3D objects, others merely provide a texture to use on that square.
- Plants do not die naturally, only by exposure or being eaten.
- Plants created offline using meta language. ...
- Plants reproduce by seeding nearby squares depending on environmental factors:
  - Proximity to fellow species (Conway)
  - Ambient temperature and humidity
  - Proximity to a resource (water/rock)

The virtual world: the Monsters

- This is the hard part…
- …and is still very vague.

The virtual world: the Monsters

- Each Monster is an individual instance.
- A Monster has DNA which codes for every part of its behaviour and appearance.

- Appearance:
  - Its appearance is based upon its genetic mutation of a basic Monster.
  - Each has a small individual texture, each distinct but a variation on its species.
  - Its appearance is also matched in its parameters, so large Monsters will weigh more and use more energy in movement.

The virtual world: the Monsters

- Reproduction
  - Monsters cannot reproduce cross species.
  - Monsters reproduce by fertilisation of ‘eggs’.
  - Monsters have genders.
- Senses
  - They can see, hear and smell their surroundings.
- Pyramid of needs
  - Food, water, sleep, reproduction, recreation, discovery.
- Adaptive behaviour (learning).
  - This requires them to have ‘memory’.

The virtual world: the Monsters

- Have the concept of a general set of abilities such as 'interact with'. These can be applied to anything in the world. It returns some value or changes a state. The Monster must be able to judge whether it prefers that state, and ranks that action as better or worse then it ranked it before.
- This is a nice idea but may take many generations before a single working Monster evolves.
- The environment must be diverse enough to allow 'niche' specialisation. Complexity is the key to non determinism and variation.

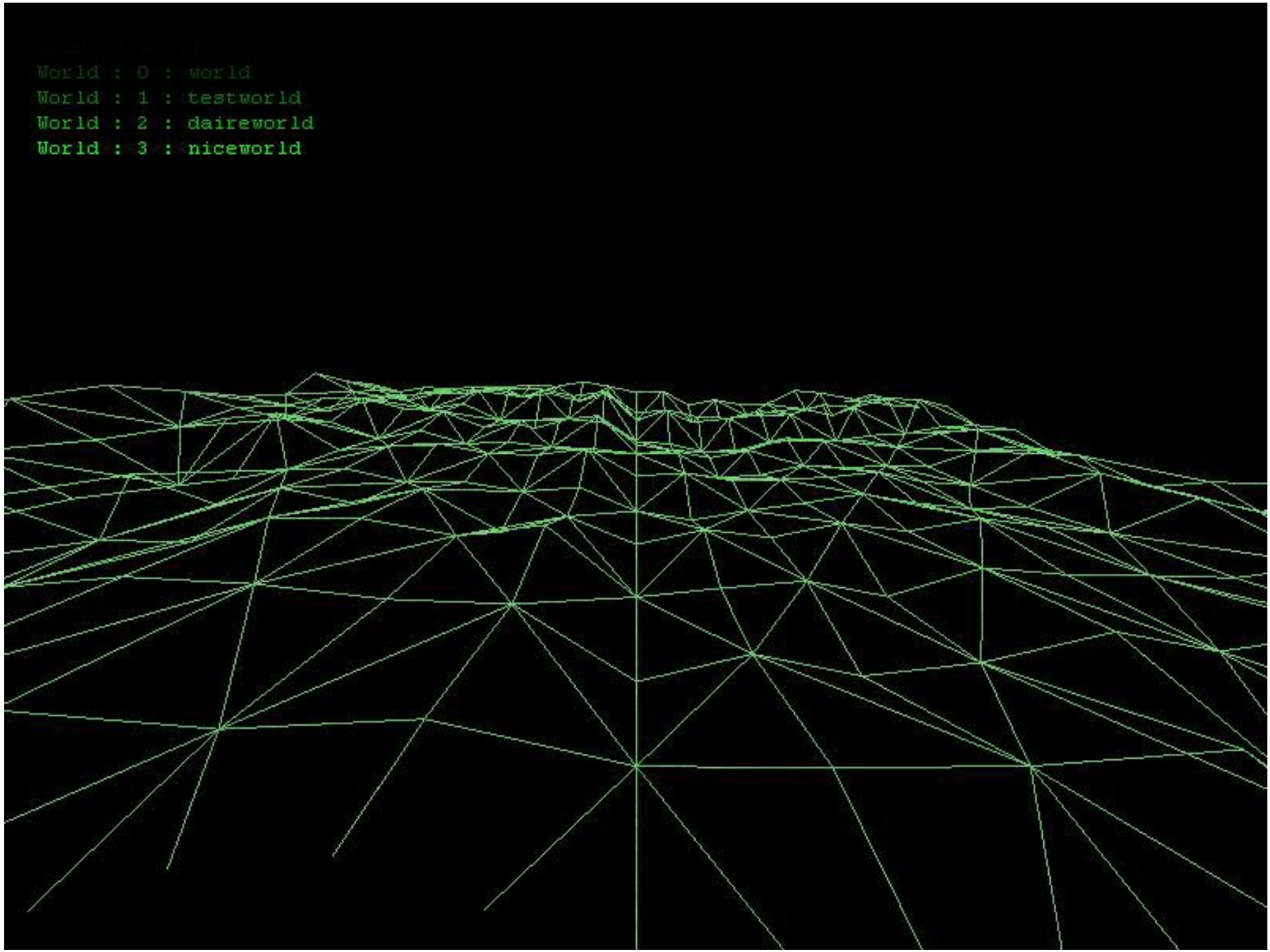
The world so far

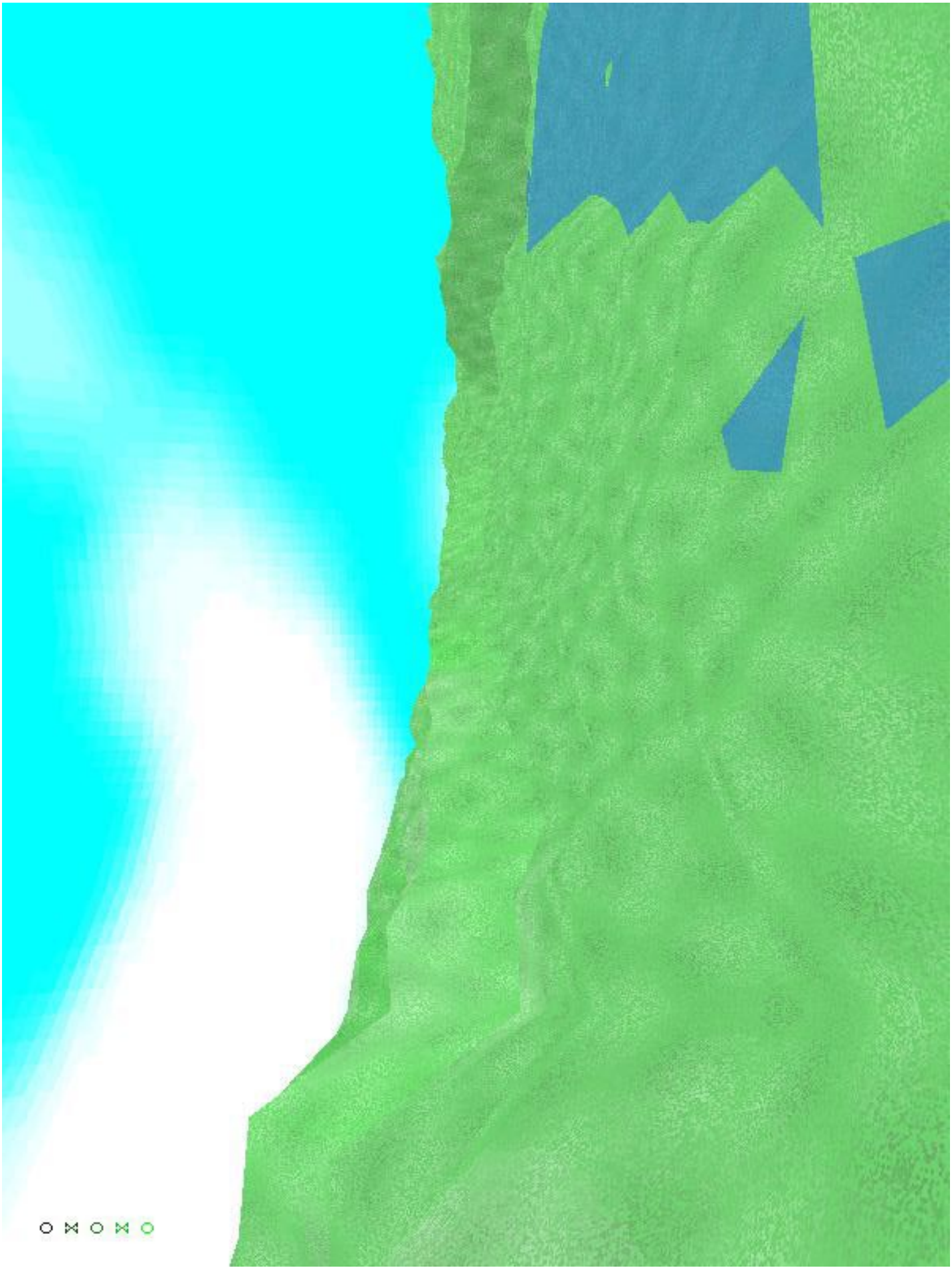
- Early world …
  - World 1 …
  - World 2 …
- Bug corrected world …

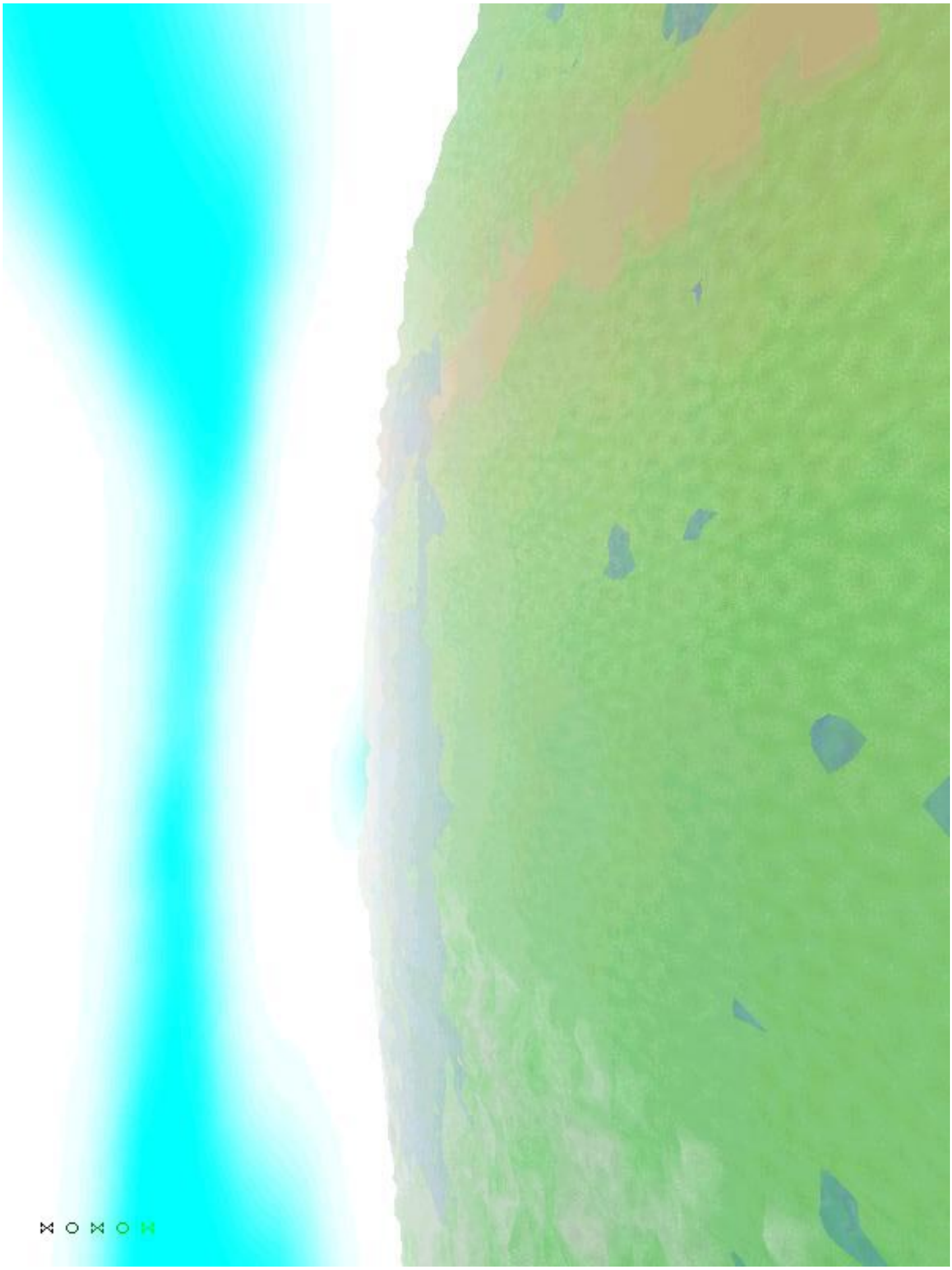
Problems...

- Fullscreen debugging.
- Time parameterisation.
- Aesthetical programming.

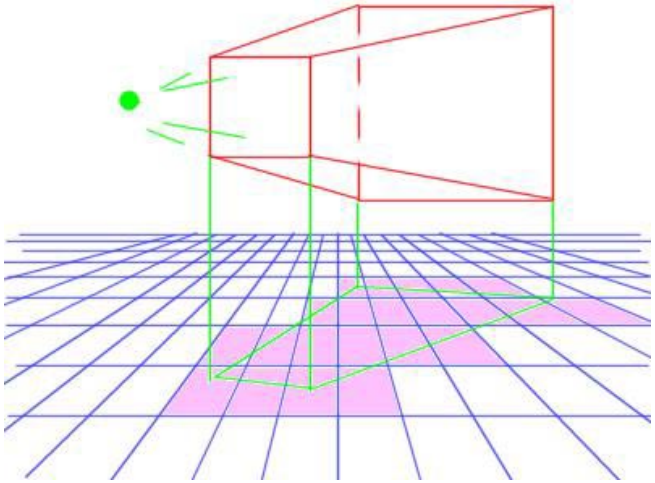
Any questions?


World : 0 : world
World : 1 : testworld
World : 2 : daireworld
World : 3 : niceworld

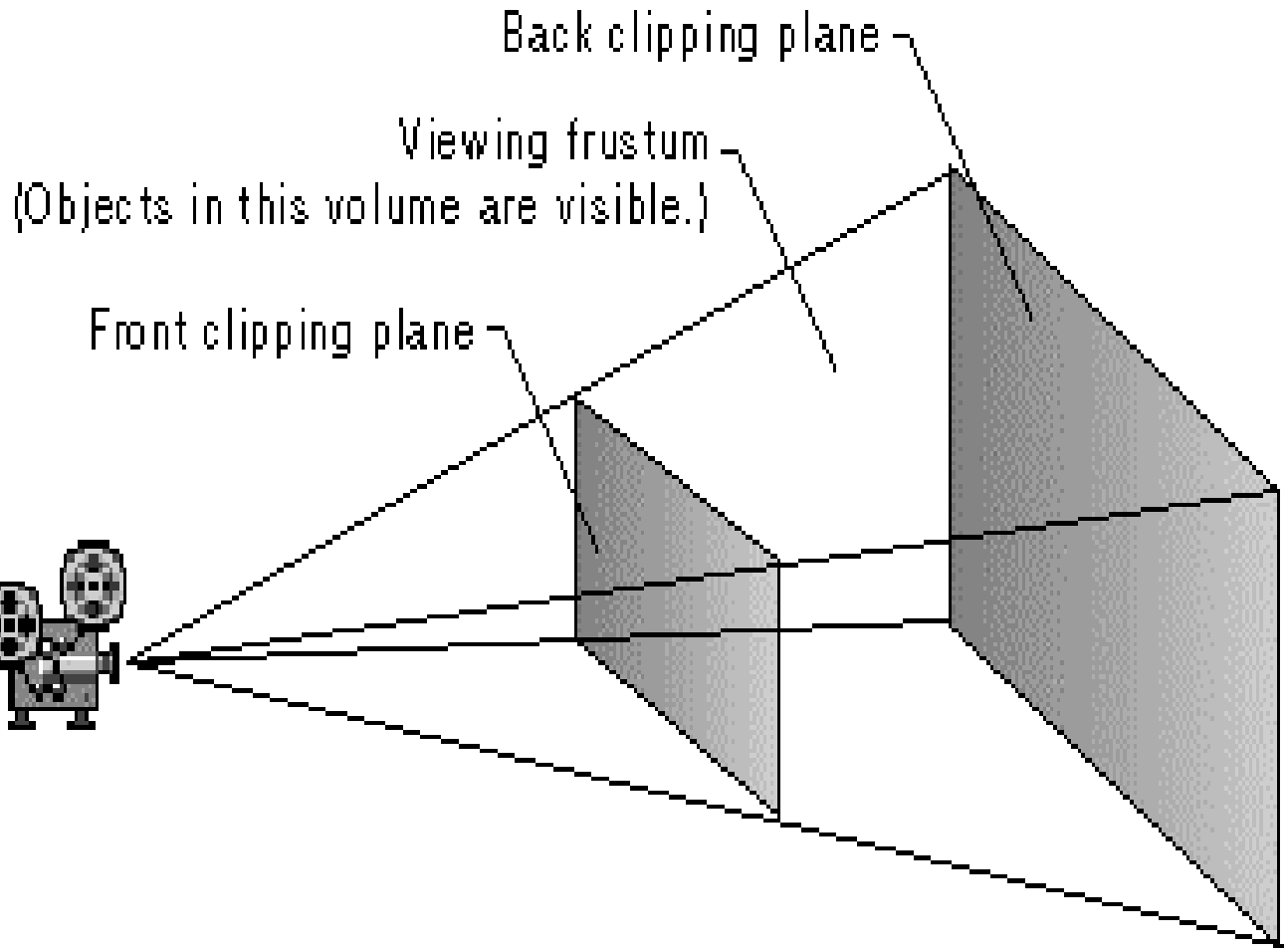




Visible area



View Frustrum



DirectX ‘modulate’

- Call
  - `IDirect3DDevice->(0, D3DTSSCOLOROP, D3DTOP_MODULATE);`
- Tells DirectX to blend texture pixels with underlying material pixels using the formula
  - Final RGBA = Texture RGBA X Material RGBA
- Also has
  - MODULATE2X, MODULATE4X, ADD, ADDSIGNED, ADDSIGNED2X, SUBTRACT, ADDSMOOTH, D3DTOP_BLENDDIFFUSEALPHA, D3DTOP_BLENDTEXTUREALPHA, D3DTOP_BLENDFACTORALPHA, D3DTOP_BLENDCURRENTALPHA, BLENDTEXTUREALPHAPM , MODULATECOLOR_ADDALPHA, MODULATEINVALPHA_ADDCOLOR, MODULATEINVCOLOR_ADDALPHA, BUMPENVMAP, BUMPENVMAPLUMINANCE and DOTPRODUCT3.

Programming plants

- The plants are created beforehand using a plant generator application. It allows the user to set the plants parameters and reactions to a variety of factors.
- At run time each plant species (identified by a unique ID) is passed, via the plant manager singleton, its squares parameters. The plant species returns a texture ID based on its programmed 'state' during the render call.
- During the Update call it takes some action (dies, grows, reproduces) based on its codes and the terrain parameters.